Statistisk sentralbyrå
Statistics Norway

MOSART 7.5
Teknisk dokumentasjon

Dennis Fredriksen og Pål Knudsen

TALL SOM FORTELLER
NOTATER / DOCUMENTS
2025/4

I serien Notater publiseres dokumentasjon, metodebeskrivelser, modellbeskrivelser og standarder.

Publisert: 22. januar 2025

ISBN 978-82-587-1086-5 (elektronisk)
ISSN 2535-7271 (elektronisk)

Standardtegn i tabeller	Symbol
Ikke mulig å oppgi tall Tall finnes ikke på dette tidspunktet fordi kategorien ikke var i bruk da tallene ble samlet inn.	.
Tallgrunnlag mangler Tall er ikke kommet inn i våre databaser eller er for usikre til å publiseres.	..
Vises ikke av konfidensialitetshensyn Tall publiseres ikke for å unngå å identifisere personer eller virksomheter.	:
Desimaltegn	,

Forord

Dette notatet dokumenterer IT-løsningene i versjon 7.5 av modellen MOSART. Forrige tekniske dokumentasjonen av modellen ble utgitt i 2015 og gjaldt versjon 5.7.

Denne versjonen inneholder en rekke oppdateringer og utvidelser, blant annet flere nye vedlegg som beskriver inkludering av nye personkjennetegn og opprettelse av nye pensjonsregler.

Statistisk sentralbyrå, 16. januar 2025

Linda Nøstbakken

Sammendrag

Dette notatet belyser MOSART i form av forutsetninger og implementering. MOSART er en dynamisk mikrosimuleringsmodell for framskriving og analyse av framtidig arbeidstilbud, utdanningsnivå og trygd.

Notatet vektlegger både operering og utvidelse av modellen.

Innhold

1. Innledning

Dette notatet dokumenterer IT-løsningene i modellen med tanke på anvendelse, oppdatering og utvikling av modellen. Forrige tekniske dokumentasjon av MOSART finnes i Fredriksen og Knudsen (2015), mens den nyeste generelle dokumentasjonen av modellen finnes i Andreassen med flere (2020). MOSART er en modell som starter med befolkningen i et basisår og simulerer det videre livsløpet til hvert enkelt individ. Basert på historiske og simulerte data kan modellen beregne blant annet pensjonsytelser med utgangspunkt i de faktiske pensjonsreglene. Regneteknisk gir dette en omfattende og komplisert kildekode, kombinert med store datamengder. Modellen kjøres derfor på en kraftig Linux-server og er implementert i C#.

1.1. Modellen MOSART

MOSART er en demografisk basert mikrosimuleringsmodell for skolegang, arbeidstilbud og trygd. Idéen bak modellen er å trekke et utvalg av befolkningen for deretter å simulere det videre livsløpet for hvert enkelt individ i dette utvalget.[1] Mikrosimulering er en formålstjenlig metode når det er mer hensiktsmessig å modellere sammenhengene på individnivå, for så å aggregere opp til totaltall, snarere enn å modellere sammenhengen mellom aggregerte størrelser. Spesielt gjelder dette når man beskriver atferd på mikronivå ved hjelp av ikke-lineære relasjoner og populasjonen er heterogen, slik at sammenhengen mellom mikrogrunnlaget og de aggregerte relasjoner blir komplisert. Analyser av regelverkene i skattesystemer og pensjonssystemer vil ofte falle inn i dette mønsteret.

Utvalget modellen starter med kalles **utgangspopulasjonen**, og inkluderer viktig informasjon om forhistorien til disse personene. Blant annet vil allerede opptjente pensjonsrettigheter i folketrygden påvirke pensjonsutbetalinger i mange tiår framover. Simuleringen skjer ved å trekke om bestemte begivenheter inntreffer for det enkelte individ i løpet av et år. Sannsynlighetene for at begivenhetene skal inntreffe avhenger av kjennetegn ved det enkelte individ selv. Disse sannsynlighetene/-begivenhetene er ofte knyttet til overganger, og kalles av den grunn **overgangssannsynligheter**. I denne versjonen av modellen simuleres begivenheter knyttet til inn- og utvandring, død, fødsler, husholdning/ekteskap, bosted, utdanning, trygd, yrkesdeltaking og inntekt/formue. På grunnlag av de simulerte livsløpene beregnes pensjonsytelser fra folketrygden, avtalefestet pensjon og tjenestepensjon. For hvert nytt år som går legges nye innvandrere og nye årskull til det utvalget modellen startet med. Resultatet av simuleringen blir en **modellpopulasjon** med livshistorien for hvert enkelt individ i utvalget. Da utvalget enten omfatter hele befolkningen eller er tilfeldig trukket (representativt), vil modellpopulasjonen kunne si noe om befolkningen i Norge i tiårene framover, gitt de forutsetningene som er gjort omkring utviklingen i overgangssannsynlighetene.

MOSART er basert på bestemte simuleringsteknikker som er viktige for å forstå oppbyggingen av modellen. Modellen er basert på **diskret tid** med kalenderåret som tidsenhet, og da typisk med status ved utgangen av året. Modellen simulerer et **tverrsnitt** av hele befolkningen, og hele befolkningen simuleres for ett år før modellen går videre til neste år. Tverrsnittsimulering gjør det mulig å ha interaksjoner mellom individer og legge restriksjoner på antall begivenheter av ulike typer i en eller flere perioder. Simuleringen er **rekursiv** ved at vi simulerer én og én begivenhet om gangen i en på forhånd fastlagt rekkefølge. Spesielt vil begivenheter som simuleres senere samme år kunne avhenge av begivenheter som er simulert tidligere samme år, ved at vi bruker såkalte betingede sannsynligheter. Hvert enkelt kjennetegn simuleres for hele befolkningen for et år, før modellen går videre til neste kjennetegn i samme år.

[1] Siden 2010 har de fleste vesentlige simuleringene omfattet hele befolkningen, og utvalg blir språklig kanskje litt misvisende. Samtidig vil en enkelt simulering kun omfatte et mulig utfall av hvert enkelt livsløp, og i den forstand er fortsatt modellen et utvalg. Det er også slik at vi bruker utvalg i utviklingen av MOSART (for å ta ned kjøretiden) og i enkelte tunge simuleringer.

Dokumentasjon av MOSART finnes i Andreassen med flere (1993), Fredriksen og Spurkland (1993), Fredriksen (1998), Gjefsen (2013) og Andreassen med flere (2020). Den modellversjonen som omtales i dette notatet har versjonsnummer MOSART 7.5, ofte forkortet mo75. Se også Vedlegg B for en oversikt over sammenhengen mellom de ulike modellversjonene.

1.2. Sammendrag

Simuleringsmodellen MOSART er skrevet i C# og kan eksekveres på de fleste allmenne operativ-systemer og typer datamaskiner (bare de har nok internminne). For tiden driftes modellen på Linux, knyttet til serveren 'sl-mosart-01' og disken '/mosart'. Alt arbeid på modellen forutsetter at man er pålogget denne serveren og jobber på data som ligger på denne disken. Notatet forutsetter at man har elementær kunnskap om Linux. Vedlegg A gir en oversikt over arkivsystemet i MOSART. Modellen kan med moderat arbeidsinnsats gjøres eksekverbar for andre operativsystemer, herunder Windows og macOS. Spesifikt har vi i en periode utviklet (og eksekvert) modellen på Windows XP.

Drift av modellen

MOSART simuleres ved å opprette en katalog under '/mosart/res', hvor navnet på katalogen også blir arbeidsnavnet på simuleringen/kjøringen/jobben. En ny katalog/simulering opprettes mest hensiktsmessig ved å eksekvere shellscriptet 'newjob', og man får da samtidig kopiert over alle filer man trenger for å eksekvere modellen.[2] Katalogen vil inneholde såkalte styrefiler (*.con), som er tekstfiler som angir verdier på parametere som styrer eller på annen måte inngår i simuleringen, eller som angir referanser til andre filer som inneholder simuleringsmodellen, overgangs-sannsynlighetene eller utgangspopulasjonen. I tillegg inneholder katalogen en kjørefil (job.exe), og ved å stå i katalogen og skrive 'job.exe' vil simuleringen bli eksekvert på riktig måte. Dette vil omfatte valg av modellvariant og av arbeidsområde, samt at alle resultatfiler fra simuleringen blir dirigert til denne samme katalogen, hvor resultatfilene kan leses i ettertid. Styrefilene og kjørefilen omtales som **brukergrensesnittet** i modellen, og selv om brukergrensesnittet i MOSART er enkelt, er det for vårt formål effektivt for å kunne eksekvere modellen med ulike spesifikasjoner.

Etter at simuleringen er fullført, vil katalogen inneholde brukergrensesnittet og samtlige resultatfiler. Sistnevnte vil omfatte egendokumentasjon, tabeller med aggregerte tall etter ønske og eventuelt en utskrift av modellpopulasjonen med utvalgte kjennetegn for hvert enkelt år. Det er lagt opp til at modellen skal være enkel å drifte, og at man kan eksekvere modellen uten å ha inngående kjennskap til hverken MOSART eller programmeringsspråkene som er benyttet. Kapittel 2 gir en oversikt over programmeringsspråk, arkivsystemer og hvordan man utfører ulike typer simuleringer. Vedlegg C (driftsstøtte) og vedleggene E-H (brukergrensesnittet) inneholder mer detaljert informasjon som støtter opp om bruk av modellen.

Simuleringsmodellen

MOSART som simuleringsmodell er skrevet i C#, og består av en hovedfil ('model.cs') som organiserer simuleringen, og en rekke underfiler med de ulike modulene i modellen. Simuleringen starter med egendokumentasjon og innlesing av overgangssannsynligheter og andre simulerings-parametere. Deretter følger innlesingen av utgangspopulasjonen, som er en mer tidkrevende arbeidsoppgave. Så kommer selve simuleringen, organisert som en løkke over de årene som inngår i simuleringen/datagrunnlaget.[3] Innenfor hvert år kalles de ulike modulene opp etter tur, slik at de

[2] Eksekvere vil her si at man skriver 'newjob' med tilhørende parametervalg på kommandolinja i Linux etterfulgt av et 'enter'.
[3] Denne løkka starter i 1967 som er det første året vi har data for. Modellen går gjennom de historiske årene på samme måte som den senere faktiske simuleringen, med den forskjell at det er historiske data som settes inn i hver modul hvert år. Dette forenkler pensjonsberegningene, og gjør samtidig at vi kan ta ut tabeller sømløst fra 1967 og framover, samt utføre historiske simuleringer i ulike varianter.

ulike kjennetegnene kan bli simulert i en fast rekkefølge. På slutten av hvert år produseres tabeller med aggregerte størrelser og en fil med én linje for hver person med utvalgte variabler. Når alle årene er simulert blir simuleringen avsluttet ved at alle resultatfiler blir lukket, modellpopulasjonen blir sortert, og med avsluttende egendokumentasjon.

MOSART er som nevnt skrevet i C#, et objektorientert programmeringsspråk. Kapittel 3 gir en kort innføring i C# sammen med noen praktiske råd for hvordan arbeidet med utviklingen av nye modellvarianter bør legges opp. Kapitlet gir også en innføring i hovedstrukturen i simuleringsmodellen, men det er en forutsetning at man samtidig leser selve kildekoden for å få noe mer enn en overfladisk forståelse. Vedleggene G-K omtaler enkelte av modulene i simuleringen i noe større detalj. Spesielt er det vedlagt en oppdatert beskrivelse av tabelluttaket i Vedlegg G og av utgangspopulasjonen i Vedlegg L og M. Se Vedlegg D for en oversikt over de eksterne filene som inngår i en simulering.

Trekkemetoder

MOSART er basert på stokastisk simulering ved at modellen trekker om bestemte begivenheter inntreffer for hvert enkelt individ hvert enkelt år. Disse trekningene tilfører modellen en egen usikkerhet tilsvarende den man får ved trekninger av utvalg, ofte omtalt som simuleringsstøy. Kapittel 4 omtaler noen metoder vi benytter for å redusere denne ekstra og for oss uønskede usikkerheten, med vekt på hvordan disse variansreduserende trekkemetodene bør anvendes i modellen. Kapitlet omtaler også metoder for å finne en tilfeldig person med gitte kjennetegn og metoder for justering av overgangssannsynligheter slik at modellen treffer gitte eksogene skranker. Spesielt vil kombinasjonen av variansreduserende trekkemetoder og justerte overgangssannsynligheter gjøre at modellen treffer disse skrankene tilnærmet eksakt selv med små utvalg.

Evaluering av IT-løsninger

Kapittel 5 vurderer enkelte sider av arbeidet med IT-løsninger for MOSART, spesielt kvalitetssikringen av modellen, og nedenfor oppsummeres noen hovedpunkter.

Kildekoden som utgjør simuleringsmodellen MOSART består alene av 139 000 linjer, og det er svært sannsynlig at en så omfattende kildekode inneholder skjulte feil i den forstand at simuleringen på noen punkter gjør noe annet enn tiltenkt, men på en slik måte at feilene ikke oppdages. Slike skjulte feil kan gjøre stor skade, spesielt når vi klarer å forklare eller fortolke et merkverdig resultat som egentlig var en feil i kildekoden.[4] Et nødvendig kriterium for god modellutvikling består i å redusere risikoen for slike skjulte feil.

Kapittel 5 omtaler også noen av de feilene i kildekoden som har krevd lang tids bruk for å bli oppdaget, og hvordan utviklingen av nye modellvarianter kan legges opp for å unngå gjentagelser av disse feilene. Imidlertid mangler vi fortsatt et opplegg for systematisk gjennomgang av resultatene for å avdekke feil ved nye modellvarianter og nye simuleringer. Avsnitt 5.3 presenterer noen elementer i et slikt opplegg. Tilsvarende har vi ikke noe systematisk opplegg for å kontrollere at simuleringsmodellen leser inn utgangspopulasjonen og overgangssannsynlighetene riktig, men disse modulene er nå konstruert slik at de vanligste innlesingsfeilene unngås. Videre savnes en systematisk arkivering og bearbeiding av skjulte feil som oppdages, med tanke på å utvikle arbeidsmetoder som gjør det vanskelig å begå slike feil. C# inneholder også noen merkverdigheter som gjør det lett å gjøre feil, som til en viss grad kunne vært unngått med et spesialdesignet kontrollprogram.

[4] Jamfør bekreftelsefelle/*confirmation bias*.

2. Drift av modellen

Kapittel 2 går gjennom den praktiske delen av driften av modellen, med en oversikt over programmeringsspråk, arkivsystem og eksekvering av modellen. Notatet gir ingen omfattende innføring i programmeringsspråkene eller applikasjonene som sådan. En måte å lære disse systemene på er å lese dette notatet, samtidig som man jobber med/leser kildekoden og bruker manualene til programmeringsspråkene. Helst bør man ha visse elementære kunnskaper i Linux hvis man skal ha utbytte av dette notatet for å jobbe med modellen.[5]

2.1. Programmeringsspråk i MOSART

MOSART eksekveres for tiden på operativsystemet Linux. Enkelte Linux-relaterte elementer er i teksten skrevet i fnutter (' '), dette gjelder servernavn, kommandoer til Linux, filnavn, deler av filnavn og kildekode.

Skal man eksekvere modellen eller på annen måte jobbe med modellen må brukeridentifikasjonen tilhøre MOSART-gruppen 'l_mosart' og man må ha en personlig katalog i '/mosart/wapi'. Skal man eksekvere modellen med et stort utvalg bør også enkelte parametere for mono og garbage collector være satt til gunstigere verdier enn standardverdiene[6]. Videre bør katalogen '/mosart/bin' være inkludert i ens egen path, slik at man får tilgang på felles driftsstøtte (se Vedlegg C for detaljer). Detaljene i dette avsnittet får man tilrettelagt av den i MOSART-gruppen som til enhver tid kan mest om Linux og C#.[7]

For tiden er MOSART knyttet opp mot serveren 'sl-mosart-01' og disken '/mosart'. Disken '/mosart' er i begrenset grad tilgjengelig fra de andre Linux-serverne, og de andre diskene er i begrenset grad tilgjengelig fra serveren 'sl-mosart-01. Og med unntak for C#, har ikke 'sl-mosart-01' tilgang til annet enn helt standard programvare. Det kan derfor være nødvendig å flytte filer mellom disker i Linux-systemet for å få koblet data og programvare, og flyttingen bør gå via området '/ssb/stamme04/mostry'. Store filer kan til tider være vanskelig å få flyttet, men komprimering av filene med **gzip** (dekomprimering med **gunzip**) kan gjøre det lettere.

Skal man utføre mer avanserte simuleringer, som krever endringer i kildekoden, er det en forutsetning at man kan programmeringsspråket C#. Avsnitt 3.1-3.3 tar opp en del grunnleggende egenskaper i C#, ellers er det også mye dokumentasjon på nettet. En god innføring, med lenker til utdypende informasjon, er tilgjengelig på https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp . Videre er det også en forutsetning at man behersker et egnet (sett av) redigeringsverktøy. Det mest effektive er **monodevelop**, se Vedlegg N. Alternativt kan man bruke standard hjelpeprogrammer i Linux, en av teksteditorene **nedit** eller **emacs**, samt verktøy som finner fram i filer som **find**, **grep** og **diff**. Dokumentasjon av programmene fås ved å skrive 'man *navn-på-program*' i Linux. Det er også mulig å bruke programvare under Windows eller macOS til redigering av filer under Linux ved å flytte filer fram og tilbake mellom Windows/macOS og Linux.

Noen ganger vil en arbeidsoppgave bestå av en serie kommandoer i Linux som man ønsker å eksekvere suksessivt, for eksempel finne en gruppe av filer ('find ... '), som man deretter ønsker dokumentasjon av ('ls ... '). Dette kan praktisk løses ved å bruke et **shellscript** som er en tekstfil som

[5] Kunnskaper som er forutsatt kjent er pålogging til Linux, manøvrering i katalogsystemet på Linux og kjennskap til en del elementære begreper. Noen begreper i Linux er likevel forklart i notatet.
[6] Shellskriptet 'job.exe', beskrevet senere, har disse verdiene innstilt.
[7] Plasseringen og syntaksen på disse egenskapene endrer seg over tid. Det mest hensiktsmessige er å plassere disse i påloggingsskriptet til Linux, for tiden '.bashrc' som ligger på ens egen hjemmekatalog. Se også omtale av garbage collector under avsnitt 3.4.

er eksekverbar[8] og som innledes med linja '#!/bin/sh'. De etterfølgende linjene vil inneholde kommandoer til Linux som blir eksekvert etter tur. Ved å skrive 'man sh' i Linux vil man få en manual for shellscript.

Awk er et programmeringsspråk som er hendig når man skal utføre et begrenset antall manipuleringer av filer og tekststrenger. Ved å skrive 'man awk' i Linux vil man få opp en manual for awk.

Det kan av og til være formålstjenlig å sortere modellpopulasjonen på person og år, og til dette bruker vi programmet **sort**, skriv 'man sort' i Linux for en dokumentasjon. Store filer krever at man angir et egnet arbeidsområde med nok lagringsplass[9].

Filer kan overføres mellom Linux og filsluse ved hjelp av **sftp** (secure file transfer program), noe dokumentasjon fås ved å skrive 'man sftp' i Linux, ellers finnes det også dokumentasjon på Byrånettet. Fra filslusen kan man flytte filer videre til Windows eller macOS, både i administrativ og produktiv sone. Det er kun filer uten personkjennetegn som er lov å flytte ut av Linux.

2.2. Arkivsystem

En vesentlig side ved driften og vedlikeholdet av en modell er hvordan filer med kildekode, parametere og andre data er lagret. Avsnitt 2.2 redegjør for arkivsystemet i MOSART og i Vedlegg A finnes en oversikt over de katalogene som er i bruk. Alle filer ligger på disken '/mosart', tilgjengelig fra serveren 'sl-mosart-01'.

Et trekk ved arkivsystemet er at vi kun bruker navn på kataloger for å skille mellom ulike modellvarianter og simuleringer. Har man for eksempel to simuleringer med katalognavn '*kat1*' og '*kat2*', vil filene som dokumenterer simuleringene hete henholdsvis '*kat1*/sim.information' og '*kat2*/sim.information'. Dette forenkler navnestrukturen, og det ville ellers blitt tilnærmet umulig å holde orden på filer, samt klare å finne navn som både tilkjennegir type innhold i filen og hvilken simulering den er produsert av.

På katalogen '/mosart/bin' finnes det en rekke shellscript som skal gjøre det lettere å drifte modellen, og vi kommer tilbake til noen av disse senere i notatet. Vedlegg C inneholder en oversikt over de shellscriptene som er i bruk, videre vil eksekvering av 'mosdok' også gi en kort oversikt.

Katalogen '/mosart/prog' inneholder en underkatalog for hver modellvariant (utgave av MOSART), og hver av disse underkatalogene vil inneholde sin variant av samtlige filer som inngår i simuleringsmodellen (enten disse er forskjellig fra modervarianten eller ikke). Vi bruker ellers ordene modellvariant og modellversjon litt om hverandre: Modellversjon vil normalt henvise til en variant som brukes som et referansepunkt og har egen dokumentasjon. Se Vedlegg B for en omtale av de ulike versjonene av MOSART. Modellvarianter vil være avledninger av en bestemt modellversjon hvor det kun er gjort (mindre) endringer for å få til simuleringer med (litt) andre forutsetninger. Dette notatet dokumenterer hva vi i filnavn med videre refererer til som MOSART 7.5, som ligger på katalogen '/mosart/prog/mo75_1b'. Videre varianter avledet av denne modellversjonen skal ha 'mo75' som forstavelse til sitt eget navn.

Kataloger med navn av typen '/mosart/input/v*ii*' vil inneholde utgangspopulasjonen og alle overgangssannsynligheter til en versjon av modellen med generasjonsnummer '*ii*' (i denne versjonen vil det si '/mosart/input/v75'). Overgangssannsynligheter for et tema vil være samlet på

[8] Eksekverbar betyr at Linux leser filen som om den inneholder kommandoer til Linux. En fil kan gjøres eksekverbar ved å utføre kommandoen 'chmod u+x *fil-navn*'.
[9] For eksempel kan man bruke 'sort -T /MOSART/tmp *navn-på-fil*', hvor parameteren '-T' angir arbeidsområdet for 'sort'.

egne kataloger, for eksempel vil alle overganger for inn- og utvandring være samlet på '/mosart/input/v75/migration'. Generelt skal første ledd i filnavnet tilkjennegi innholdet i filen, mens andre ledd skal angi versjonsnummer. For eksempel vil filen '/mosart/input/v75/migration/time_series_migration.v2023' inneholde en tidsserie (*time_series_*), for faktisk inn- og utvandring (*_migration_*) til og med 2023 (.v*2023*). Se Vedlegg A for katalogoversikt og Vedlegg D for filoversikt.

Merk at en katalog med overgangssannsynligheter kan inneholde ulike varianter av overgangs-sannsynlighetene, men da skal det kun være parameterverdiene som er forskjellige. I den grad man lager nye modellvarianter som krever et annet format på tabellene med overgangssannsynligheter, bør man enten tildele filen nytt navn (1.ledd), eller opprette en ny katalog hvis endringene er omfattende.

Simuleringene ligger i katalogen '/mosart/res', og hver simulering vil ha sin egen underkatalog inneholdende egendokumentasjon og resultatfiler. Merk at siste ledd i navnet på underkatalogen også er navnet på simuleringen. Det vil over tid bli liggende mange underkataloger på '/mosart/res', og det er derfor viktig med systematikk i navnene. Det er et viktig poeng at man *beholder* siste ledd i katalognavnet uendret av hensyn til muligheten for å finne igjen simuleringen med enkle metoder. Ved å samle grupper av simuleringer i egne kataloger kan man begrense antall kataloger som ligger direkte på '/mosart/res', gjøre det lettere å finne fram mellom ulike versjoner og åpne for resirkulering av navn ved oppdatering av modellen (for eksempel på referansebanen). Simuleringer med den versjonen som dokumenteres i dette notatet kan med hell få forstavelsen 'mo75_', eventuelt samles på underkataloger som starter på et slikt navn.

Hver natt tas det automatisk sikkerhetskopi (backup) av dataene som er lagret på '/mosart', og resultatet lagres i lengre tid. Imidlertid kan en hel dags arbeid gå tapt i de tilfeller systemet bryter sammen rett før backup. Går det noe tid før man oppdager at en fil er slettet eller ødelagt, skal man likevel kunne få hentet fram eldre sikkerhetskopier. Det forutsetter at man klarer å identifisere datoen for ødeleggelsen, og at det ikke har gått for lang tid. Av den grunn bør man ta egne kopier av særskilte viktige filer. Videre kan det være hensiktsmessig å ta en kopi rett før og rett etter store endringer. Førstnevnte gir en muligheten til å gå tilbake og starte på nytt, sistnevnte sikrer en mot tap av en dags arbeid.

En type feil det er vanskelig å beskytte seg mot er at filer blir endret uten at man er klar over det. Dette vil være en alvorlig feil fordi mye av dokumentasjonen av hver enkelt simulering består av referanser til de filene som er benyttet. Ved redigering bør man derfor unngå å endre filer som er benyttet i andre simuleringer, men snarere lage en kopi med et nytt navn. Filer som inngår i publiserte versjoner, bør generelt være skrivebeskyttet. Om filer er endret, kan undersøkes ved å se på datoopplysningene for en fil. Videre vil filene 'model.documentation' og 'sim.documentation' vise hvor filene til henholdsvis denne modellvarianten eller denne simuleringen opprinnelig er hentet fra. Filen 'sim.input' vil angi datoopplysninger for de filene som er benyttet på det tidspunktet simuleringen ble startet[10].

2.3. Hvordan eksekvere modellen

En simulering produseres i MOSART ved å opprette en ny katalog under '/mosart/res', og dernest kopiere over det såkalte brukergrensesnittet fra en modellvariant eller en tidligere simulering. Disse arbeidsoppgavene gjøres lettest ved å bruke shellscriptet 'newjob' på følgende måte (forutsatt at katalogen '/mosart/bin' inngår i ens egen path, se Vedlegg C):

[10] I teorien kan filene bli endret fra datoangivelsene blir rapportert til 'sim.input', og til simuleringen begynner å lese filene.

> newjob *navn-på-eksisterende-simulering navn-på-ny-simulering* **[↵]**

Det vil si at 'newjob' oppretter en katalog med navn '/mosart/res/*navn-på-ny-simulering*' og kopierer brukergrensesnittet fra en eksisterende simulering eller modellvariant med navnet '*navn-på-eksisterende-simulering*'. Uten videre spesifisering legges den nye simuleringen til samme underkatalog som den eksisterende simuleringen. Skriver man kun siste del av navnet på eksisterende simulering, leter 'newjob' etter denne jobben etter bestemte regler, og eventuell tvetydighet om navnet på eksisterende simulering rapporteres. Ytterligere parametere kan spesifiseres. Se Vedlegg C for detaljer.

Dernest må man forflytte seg til den nye katalogen. Man er da klar til å redigere den nye simuleringen. Er noen av opsjonene spesifisert feil, for eksempel at '*navn-på-eksisterende-simulering*' ikke finnes eller at '*navn-på-ny-simulering*' allerede finnes, skjer det ingenting og man får logget en feilmelding til skjerm.

Brukergrensesnittet[11]

I denne versjonen av MOSART består brukergrensesnittet av en kjørefil og en rekke styrefiler[12]. **Kjørefilen** er et shellscript som eksekverer simuleringsmodellen og andre nødvendige kommandoer til Linux. Kjørefilen har navnet 'job.exe', vil være generell, og skal ikke endres av bruker, se flere detaljer nedenfor. **Styrefiler** har navnestruktur '∗.con' og inneholder informasjon om hvilke filer som skal benyttes og verdier på simuleringsparametere. Styrefilene er rene tekstfiler som kan redigeres av en hvilken som helst editor. Ved siden av kommentarer vil styrefilene normalt bestå av to-tre kolonner med formatert informasjon. Første kolonne vil inneholde **styrevariabelen**, en tekststreng som identifiserer innholdet av parameteren som følger. Styrevariabelen brukes blant annet av simuleringsmodellen og av kjørefilen, for å identifisere innholdet på linjen. Styrevariabelen og navn på tilhørende variabel i kildekoden bør så langt det er mulig være identisk. I andre kolonne kommer parameterverdien, og dette skal normalt være et tall eller en tekst med navnet på en fil eller en katalog. I noen av styrefilene kan det enten være kommentarer eller ytterligere parametere etter andre kolonne. Merk at kolonnene er separert med en eller flere blanke og/eller tabulatortegn. Styrefilen 'output.con' har et noe annet format, da den både er modulisert (output.∗.con) og de fleste parameterlinjer i de underliggende filene inneholder mer enn en parameterverdi.

Styrefilen 'input.con' inneholder referanser til filer som skal benyttes, inkludert modellvariant. Styrefilen 'parameters.con' er hovedfilen for tallbaserte parametere. Det er for mange parametere til at alt kan samles i en fil, og store temaer er skilt ut som egne *.con-filer, hvor de fleste navnene er selvforklarende. Parametere som kan variere over pensjonssystemer, har styrefiler av typen 'navn.con' og '*navn_i*.con', hvor 'i' refererer til pensjonssystem for i>1. Alternativt, og anbefalt, kan parametere i ulike pensjonssystemer lettest håndteres av spesialfilen 'change_parameters.con'.

Gjennom 'output.con' og 'output.∗.con' kan man velge hvilken egendokumentasjon og hvilke tabeller som skal produseres, se Vedlegg F og G. Gjennom 'model_population_file.con' styrer man produksjonen av modellpopulasjonen, se Vedlegg H. Avsnitt 2.4 og Vedlegg E omtaler i mer detalj innholdet i styrefilene.

[11] Et grensesnitt er overgangen eller koblingen mellom to systemer. Brukergrensesnitt er de mulighetene den vanlige brukeren har for å eksekvere modellen med andre parametervalg. I MOSART vil dette bestå i å endre parameterverdier i tekstfiler.

[12] Styrefilen 'output.con' er nå modulisert på grunn av størrelsen og kompleksiteten. Undermodulene har navn 'output.∗.con', og simuleringsmodellen slår disse sammen til en fil, se Vedlegg G for detaljer. Styrefilen 'model_population_file.con' kan også ha flere forekomster, en for hver modellpopulasjon som skal skrives ut.

Eksekvering av simuleringsmodellen

Avsnitt 2.5 tar opp andre måter å eksekvere modellen, blant annet hvordan man skal eksekvere flere simuleringer i serie. Etter at redigeringen av styrefilene er ferdig, kan simuleringen utføres ved å skrive navnet på kjørefilen:

```
> /mosart/res/navn-på-ny-simulering/job.exe & [↵]
```

Gitt at man fremdeles står i samme katalog, kan man nøye seg med å skrive:

```
> job.exe & [↵]
```

Kjørefilen 'job.exe' finner først sin egen katalog, og sjekker om det pågår en simulering der allerede eller om innholdet i katalogen er skrivebeskyttet, og stopper eventuelt seg selv der og da (se omkjøring under Avsnitt 2.5 hvis det oppstår problemer med å eksekvere 'job.exe'). Deretter finner 'job.exe' modellvarianten slik det er angitt i 'input.con', og eksekverer denne varianten med kobling mot katalogen for 'job.exe'. Videre vil 'job.exe' styre alle feilmeldinger til filer hvor meldingene kan gjenfinnes i ettertid ('execution.log', 'sim.errors').

Tegnet '&' sikrer at simuleringen går i bakgrunnen (batch), slik at simuleringen fortsetter selv om man logger seg ut av Linux, og slik at tastaturet frigjøres og man kan jobbe videre med andre ting i Linux.

Simuleringen krever en del regnekraft, og i Avsnitt 2.6 kommer vi tilbake til hvor lang tid simuleringen vil ta og andre forhold som bør undersøkes før simuleringen startes.

Resultatfiler

Man kan følge simuleringen ved å eksekvere 'top' eller 'ps -alf', man får da informasjon om hva serveren jobber med. Forøvrig skrives all egendokumentasjon og alle tabeller ut fortløpende, og spesielt vil 'sim.control' vise hvor langt simuleringen har kommet. Alle meldinger som skrives til skjerm av simuleringen, fanges opp av filen 'sim.errors'. Dette omfatter også advarsler om irregulariteter i simuleringen (linjer som starter med 'WARNING: '). Blir simuleringen avbrutt av en feil, vil feilmeldingene også legges ut på filen 'sim.errors'. Jobber med et større omfang som kan bli eksekvert, er følgende:

1. Simuleringsmodellen (model.exe)
2. Sortering av modellpopulasjonen (sort ...)

Merk at simuleringen først termineres etter at en eventuell sortering er fullført, og i denne perioden vil simuleringsmodellen (model.exe) gå i dvale. At sortering pågår, vil framgå av siste linje i 'sim.control' og ved å følge pågående jobber med 'top' eller 'ps -alf'.

Resultatet fra simuleringen vil avhengig av valg i 'output.con', 'output.*.con' og 'model_population_file.con' bestå av brukergrensesnittet (*.con), egendokumentasjon ('sim.* ', 'execution.log'), tabeller med aggregerte tall ('r*_*.prn', 'sim.* ') og modellpopulasjon ('population*', 'sim.population. *'). Vedlegg F inneholder en oversikt over de filene som blir produsert. Alle resultatfilene havner på den katalogen som er benyttet (/mosart/res/*navn-på-ny-simulering*).

Tabeller og egendokumentasjon som produseres ('r*.prn', 'sim.* '), kan relativt enkelt overføres til Windows og bearbeides videre i andre systemer (Excel, Word). Merk at mange filer har like navn, det

er bare katalogen de ligger på som skiller dem fra hverandre (samt filens innhold, inkludert overskriften). Ved overføring til Windows eller macOS må man selv passe på å holde orden på dette.

2.4. Endring av parametere

Nye simuleringer med et annet innhold kan ofte skapes ved å endre parameterverdier i styrefilene ('*.con'), og nedenfor går vi gjennom noen av disse parameterne som endres mye og/eller har et ikke-intuitivt innhold. Parameternavn (i fnutter) er skrevet på samme måte som styrevariabelen i styrefilen. Se ellers Vedlegg E til H for mer detaljer.

Kontroll av brukergrensesnittet og simuleringen
Ønsker man å sammenlikne innholdet i brukergrensesnittet med en annen simulering kan man skrive kommandoen:

/mosart/res/*navn-på-denne-simulering*> diffcon *navn-på-den-andre-simuleringen* **[↵]**

Man vil da få skrevet til skjerm *alle* forskjeller mellom brukergrensesnittet ('job.exe', '*.con') i den katalogen man står og den katalogen man har spesifisert. Shellskriptet 'diffcon2' avgrenser sammenligningen til reelle forskjeller i parameterverdier i simuleringen (kommentarer utelates, men også forskjeller i tabelloppsettet og utskriften av modellpopulasjonen). Ønsker man å sammenlikne to filer i to simuleringer kan man skrive:

/mosart/res/*navn-på-denne-simulering*> diff -b *navn-på-fil navn-på-den-andre-simuleringen* **[↵]**

Man får da skrevet til skjerm alle forskjeller i filen '*navn-på-fil*' mellom den simuleringen man står i og den man har angitt. Dette kan være nyttig for å finne forskjeller i simuleringsmodellen og i tabeller hvor man har valgt å eksekvere tilsynelatende samme modell med samme random-seed. Se Vedlegg C for mer detaljer omkring disse shellscriptene ('diff' er et standard hjelpeprogram i Linux).

Simuleringsmodellen
Modellvarianten velges ved parameteren 'simulation_model' i styrefilen 'input.con'. Normalt er det unødvendig å endre denne, da hver modellvariant har sitt brukergrensesnitt hvor denne parameteren er satt riktig. Er man uheldig kan endringer av verdien for parameteren 'simulation_model' medføre at den valgte modellvarianten og brukergrensesnittet som er kopiert inn blir inkompatibelt.

Simuleringshorisont
Hvor langt fram i tid simuleringen skal gå bestemmes av parameteren 'sim_end' i styrefilen 'parameters.con'. Det er ingen øvre grense for simuleringshorisonten (så fremt man har nok internminne til rådighet og tid til å vente), gitt at alle filene med åpne tidsserier har blitt avsluttet med linjen 'REPEAT' (tidsserier er filer med navnestruktur 'time_series_*.*', se Vedlegg D). Er ikke dette tilfelle, vil maksimal simuleringshorisont være bestemt av den korteste av tidsseriene uten 'REPEAT'. Mange av tidsseriene går fram til år 2200, men det er bare unntaksvis at det er noen reell variasjon etter basisåret for forutsetningene.

Random-seed
Startverdien for rekken med tilfeldige tall i simuleringen er viktig. Standard er at denne settes til samme verdi i den nye simuleringen som i simuleringen man har kopiert brukergrensesnittet fra. Dette sikrer at den nye simuleringen følger den opprinnelige simuleringen eksakt fram til det

punktet det oppstår forskjeller i forutsetningene for simuleringene. I den grad endringene kun går på omfanget av resultatfiler, så skal alle sammenfallende resultatfiler bli identiske.

Ved å sette parameteren 'random_seed' i styrefilen 'parameters.con' til verdien 0 eller lavere, vil simuleringen i dette tilfellet bruke tidsangivelsen for fastsettelsen av random-seed, omregnet til et heltall mellom 0 og 2147483647, noe som kan tolkes som en tilfeldig startverdi. Var parameteren 'random_seed' satt til 0, vil den nye startverdien for random-seed bli skrevet inn i 'parameters.con' første gang simuleringen blir eksekvert. Var 'random_seed' lavere enn 0, så skrives ikke den nye startverdien inn i 'parameters.con', men fortsetter å stå på samme verdi (negativt tall).

Ved å sette 'random_seed' til et tall større enn null, men mindre enn 2147483647, vil dette tallet bli brukt som startverdi for random-seed.

Filen 'sim.information' vil på linjen for 'Initial value, ran.gen.' opplyse om hvilken verdi random-seed hadde for denne aktuelle simuleringen.

Tekniske parametere

En rekke parametere i 'parameters.con' er der for å optimalisere tekniske sider ved simuleringen, med vekt på litt ulike aspekter. Ved å sette parameterne 'no_of_threads' til (for eksempel) 40, 'maxdegreeofparallelism' til -1 og 'recycle_objects' til 1 (=på), vil kjøretidene reduseres betraktelig. Simuleringen legger da samtidig beslag på en større andel av serverens ressurser, og dette kan gå utover andre brukere. Se avsnitt 3.4 for detaljer.

Utgangspopulasjon

Utgangspopulasjonen velges ved parameteren 'population' i styrefilen 'input.con', og man spesifiserer kun katalognavnet for utvalgene slik de inngår i katalogen '/mosart/input/v75/population'. Det er tre typer utgangspopulasjoner som er tilgjengelige, og dette valget gjøres i 'parameters.con' ved parameteren 'initial_population'. Den ordinære utgangs-populasjonen er den som omtales nedenfor og som er basert på registerstatistikk, for tiden til og med året 2022. I tillegg finnes det en mulighet for å produsere en syntetisk utgangspopulasjon, for tiden med utgangspunkt i aggregert statistikk for året 1960. Utover dette vil den ordinære og simulerte utgangspopulasjonen gjennomføre simuleringen på samme måte. I tillegg finnes det en mulighet for å simulere et nærmere angitt utvalg av typehushold, men da bare med hensyn til skatt og overføringer for et nærmere angitt år. Simulering av typehushold har ikke vært brukt på flere år, og er trolig ikke operativ.

Under en modellversjon vil det ligge flere utgangspopulasjoner, og disse skilles på tre aspekter ved navnet, for eksempel '/mosart/input/v75/population/c_2022_s01'. Her vil 'c' referere til versjon,[13] det vil si kjøringen som produserte populasjonen, og alle med dette navnet vil ha samme type innhold. MOSART er i stadig utvikling, og når vi legger til nye variabler i utgangspopulasjonen må vi lage en ny versjon. Videre vil '2022' referere til basisåret for utgangspopulasjonen og '_s01' vil referere til utvalgsstørrelsen. '01' vil være en utvalgsandel på 0,01 eller 1 prosent. Normale simuleringer vil være basert på hele populasjonen, og da er hele endelsen '_s*ii*' utelatt. For testformål er det likevel svært nyttig å ha mindre utvalg, og normalt foreligger utvalgsstørrelsene som 100[14], 10, 1, 0,1 og

[13] Det er to versjoner her, 'c' som er foretrukket og har beregnet premier for privat tjenestepensjon som konstant over årene 2018, 2020 og 2021, og 'b' som beregner disse premiene separat for disse årene. Den første versjonen gir mer robuste estimater for små og mellomstore virksomheter, men mister en eventuell utvikling over tid.

[14] Vi omtaler også 100 prosent som et utvalg, selv om det faller utenom vanlig språkbruk, litt som at økonomer gjerne kan snakke om en gruppe eller en familie på én person. I tillegg er det noen dypere grunner til å holde på 100 prosent som et utvalg. En grunn er at simuleringsresultatet fortsatt er et utvalg, i og med at hver person kun er representert med ett av alle mulige framtidige livsløp. Av den grunn kunne også utgangspopulasjonen vært større enn den faktiske populasjonen, ved at hver person var representert flere ganger. MOSART simulerer heller ikke faktiske personer, modellen tar utgangspunkt i et utvalg av faktiske kjennetegn, og simulerer hva vi tror er sannsynlig gitt disse observerte kjennetegnene.

0,01 prosent. Normalt vil vi bruke det siste året vi har all statistikk tilgjengelig for som basisår, for tiden 2022. I andre sammenhenger kan aktualitet veie tyngre, og da foreligger det populasjoner med nyere basisår, men da med manglende statistikk på noen temaer. I atter andre sammenhenger kan muligheten for å simulere en noe lengre periode hvor vi har faktiske tall å sammenlikne med være formålet. Dette gjelder både for kalibrering av modellen og for uttesting/validering av modellen. 1998 er et mye brukt basisår i så måte, da dette er første år med ny utdanningsstatistikk.

Se ellers Vedlegg L og M for nærmere omtale av utgangspopulasjonen.

Tabelluttak
En simulering produserer egendokumentasjon og tabeller med aggregerte resultater, og produksjonen av disse styres gjennom 'output.con' og 'output.*.con'. Merk at et omfattende sett av tabeller vil øke kjøretiden betraktelig. Tabelluttaket er lagt opp slik at med noe kompetanse på bruk av modellen, skal det være enkelt å ta ut nye tabeller uten å måtte endre kildekoden i simuleringsmodellen. Vedlegg G gir en kort innføring i uttak av tabeller.

Modellpopulasjon
Utskrivingen av modellpopulasjonen vil forlenge kjøretiden og fylle opp disken '/mosart' med store filer. Det er derfor mulig å styre produksjonen av modellpopulasjonen i detalj, først gjennom parameteren 'no_of_model_population_files' på 'parameters.con', dernest styres innholdet gjennom styrefilene 'model_population_file.con', eventuelt 'model_population_file_*i*.con', med hensyn til hvilke personer, år og kjennetegn som skal med. Se Vedlegg H for detaljer.

Beskyttelse av resultatfiler
Parameteren 'protect_output' i styrefilen 'output.con' avgjør om resultatfilene skal gis skrivebeskyttelse, noe som forhindrer at filene kan bli endret eller ødelagt ved vanlig bruk. Merk at både katalogen og filene gis skrivebeskyttelse, da dette er nødvendig for å sikre en reell beskyttelse.

NB! Skrivebeskyttelse trer bare i kraft i den grad simuleringen fullføres på normal måte og er basert på et utvalg på minst én prosent av befolkningen (det vil si mer enn rene testutvalg).

Ved en eventuell omkjøring vil foreliggende resultatfiler normalt kopieres til en underkatalog med navn 'backup/backup_*dato-angivelse-for-forrige-simulering*'.

2.5. Andre måter å eksekvere modellen

Avsnitt 2.3 gjennomgikk en standard enkeltstående eksekvering av MOSART, mens vi her angir noen andre praktiske måter å eksekvere en eller flere simuleringer.

Flere simuleringer
Ofte vil man ønske å gjennomføre en gruppe simuleringer, og dette bør skje ved at man eksekverer simuleringene i serie (en om gangen) framfor parallelt (alle på en gang). Grunnen til dette er at en serie-kjøring vil kreve vesentlig mindre internminne og vil belaste serveren mindre med administrasjon av mange jobber. Den første simuleringen vil også bli ferdig tidligere slik at man kan ha en viss kontroll på at de andre simuleringene vil gi rimelige resultater. Serveren 'sl-mosart-01' håndterer likevel relativt greit å eksekvere parallelt tre ordinære simuleringer med 100 prosent av befolkningen. Ved fire simuleringer, eller ved mange parallelle pensjonssystemer, begynner serveren å slite.

En serie-kjøring er enklest å gjennomføre ved å sette opp katalogene med de ulike simuleringene og redigere styrefilene slik man ønsker dem. Dernest eksekveres et shellscript hvor hver linje i

shellscriptet inneholder det fullstendige navnet på kjørefilen i hvert sitt av disse oppsettene (det vil si '/mosart/res/*navn-på-simulering-i*/job.exe' på hver linje *uten* tegnet '&' etter seg). Ved eksekvering av shellscriptet vil simuleringene bli eksekvert en om gangen, og hvis en simulering går ut med feil, vil allikevel de neste bli eksekvert.

NB! Legger man inn tegnet '&' på slutten av hver linje i shellscriptet som starter serien av jobber, vil alle jobbene starte tilnærmet samtidig, som gir et effektivitetstap, men verre, det *kan* også føre til uforståelige feilmeldinger og sammenblanding av pid-numre.

Eksekvere modellen på ny

Det er fullt mulig og av og til hensiktsmessig å eksekvere om igjen en simulering som allerede har blitt eksekvert. Dette gjelder spesielt i en utviklingsfase av en modell/simulering hvor man leter etter feil og når man skal gjenopprette en modellpopulasjon. En allerede eksekvert simulering har en viss beskyttelse mot å bli eksekvert på nytt ved et uhell, for eksempel på grunn av en skrivefeil i et shellscript som setter i gang en serie jobber, eller fordi man står i en annen katalog enn den man tror. Beskyttelsen av simuleringen er knyttet til mulig skrivebeskyttelse av katalogen, filen 'stopfile' og eksekveringstillatelsen for 'job.exe'. Nedenfor omtales de håndgrepene som må til for å oppheve disse beskyttelsene.

Er katalogen for simuleringen skrivebeskyttet, bør man sjekke at det faktisk er denne simuleringen man skal eksekvere på nytt, før man gir seg selv skrivetillatelse på katalogen for simuleringen. Skrivebeskyttelse av de enkelte resultatfilene oppheves av simuleringsmodellen.

Ligger filen '**stopfile**' fremdeles i katalogen betyr det at simuleringen enten fortsatt går eller har blitt avbrutt av systemet eller deg selv. Er simuleringen avbrutt, vil 'job.exe'/ simuleringsmodellen normalt finne ut av dette selv og slette 'stopfile' før den går videre med simuleringen. Får man allikevel ikke startet simuleringen, kan det skyldes manglende opplysninger i 'stopfile'. Man bør da sjekke om simuleringen faktisk går på grunnlag av tilgjengelige opplysninger i 'stopfile', 'sim.information', 'sim.control', og 'sim.errors' (starttidspunkt, eier, server, kommando, pid-nummer, feilmeldinger), samt informasjon fra 'ps -alf' eller 'top'. Har simuleringen stanset, kan man slette 'stopfile' manuelt før man setter i gang simuleringen på nytt. Starter man simuleringen på nytt etter å ha slettet stopfile, uten at den forrige simuleringen faktisk hadde stoppet, vil den forrige simuleringen avbryte seg selv når den finner en ny 'stopfile', eventuelt gjenopprette sin egen 'stopfile' hvis den mangler fullstendig og deretter fortsette videre. Merk at i sjeldne tilfeller kan 'pid-nummeret' i 'stopfile' være forvekslet med en annen jobb.

I noen tilfeller kan man ønske å stanse en simulering som går, enten fordi man har satt den i gang med feil parametere eller fordi simuleringen har hengt seg opp og ikke kommer videre (krever for mye internminne, evige løkker, multiplikasjon med store tall). Opplysningene fra 'stopfile' er også grunnlaget for å stoppe simuleringen utenfra i slike tilfeller ('kill' *pid-nummer)*, og man kan da normalt starte simuleringen på nytt uten flere håndgrep.

I noen sjeldne tilfeller kan 'job.exe' ha mistet eksekveringstillatelsen, og da må dette tilbakestilles ('chmod u+x job.exe').

Filen 'execution.log' inneholder en oversikt over alle gangene en simulering er startet og eventuelt avsluttet (avbrytes simuleringen utenfra, for eksempel ved at serveren går ned, så logges ikke avslutningene).

Når simuleringen eksekveres på nytt skal alt av egendokumentasjon, tabeller, modellpopulasjon og sikkerhetskopi av brukergrensesnittet bli *flyttet* til en underkatalog med navn 'backup/backup_*startidspunkt-for-forrige-simulering*' eller bli slettet der resultater ikke foreligger.

Dette gjøres for å unngå mulig sammenblanding av nye og tidligere resultater på en og samme katalog, samt ha muligheten for å hente fram tidligere resultater der man av vanvare har startet en simulering på nytt til tross for 'stopfile' og andre typer beskyttelse. Siste versjon av brukergrensesnittet (*.con) ligger direkte på 'backup'.

NB! Eksekverer man en simulering om igjen mange ganger vil det samle seg opp mange sikkerhetskopier med et meget stort og plasskrevende antall filer. Ved å eksekvere shellscriptet 'delete_backup' fra resultatkatalogen, så vil alle sikkerhetskopier bli fjernet, se Vedlegg C for detaljer.

Endring av navn
Har man endret navn på simuleringen etter at man har brukt 'newjob', men før man eksekverer simuleringen, vil 'job.exe' sørge for at det *nye* navnet blir benyttet som arbeidsområde. Imidlertid vil ikke 'job.exe' endre katalognavnet i den dokumentasjonen som opprettes før simuleringen startes, og dette omfatter blant annet filen 'sim.documentation' i resultatkatalogen. Endrer man navn etter at simuleringen er eksekvert endres ikke noe av dokumentasjonen. Navneendringer bør ut fra disse grunner bare bestå i at man kun endrer plasseringen av katalogen, det vil si at man beholder (minimum) siste ledd i det opprinnelige katalognavnet.

Gjennomsnittet av flere simuleringer
Med muligheten til å simulere hele befolkningen har behovet for å ta gjennomsnittet av flere simuleringer mer eller mindre falt bort. Trekkemetodene har ikke vært gjennom noen videreutvikling, så det har heller ikke vært behov for å beregne standardavvik på repeterte simuleringer med ulik random-seed. Det forelå tidligere en applikasjon som beregnet gjennomsnitt og standardavvik ('/mosart/support/compute_average.cs'), men denne er for tiden ikke operativ.

Massivt repeterte simuleringer
Andre formål kan på nytt øke behovet for repeterte simuleringer, og to typer anvendelser er på gang. Den første gjelder simuleringer hvor man ønsker å finne forventningsverdier over livsløpet for hvert individ i utgangspopulasjon. Her har vi gjennomført en slik analyse (Brinch med flere, 2017), men også levert beregninger til pensjonsformueprosjektet (Fredriksen og Halvorsen, 2019). Den andre anvendelsen gjelder simuleringer med et stokastisk forløp i forutsetningene. Et eksempel kan være at pensjonssystemet får andre egenskaper ved en variabel vekst i levealder.

Begge anvendelsene krever et høyt antall simuleringer for å gi meningsfulle svar, i størrelsesorden hundre og opp mot noen tusen repetisjoner. Til dette kreves et opplegg som automatisk produserer nye simuleringer basert på en modervariant. Dette gjøres med kjørefilen 'jobr.exe', som lager en løkke som setter i gang nye kjøringer etter tur, og legger disse til underkatalogen 'repeat' med navn 'repeat_*i*'. Antall repeterte simuleringer bestemmes av 'no_of_repeated_simulations' i 'parameters.con'. Det er også mulig å eksekvere simuleringene parallelt gjennom parameteren 'no_of_parallel_repeated_simulations'. Krever hver enkelt simulering lite internminne (lite utvalg, simulering avgrenset til utvalgte årskull), lønner det seg trolig å bruke et høyt antall parallelle simuleringer, hvor hver enkelt simulering eksekveres uten parallellitet. Standard simuleringer bør eksekveres etter tur, men hvor hver enkelt simulering eksekveres med full parallellitet.

Bruddpunktet i simuleringene kan styres av 'change_random_seed_year' i 'parameters.con', er denne satt til 0 endres random-seed det året simuleringen starter. Før bruddpunktet er simuleringene identiske, etter bruddpunktet er de ulike forårsaket av simuleringsstøy.

Ønsker man å finne forventningsverdier på individnivå bør man velge et parameteroppsett som trekker dødelighet på en annen måte, det vil si med 'draw_cumulative_mortality' i 'parameters.con'

satt til 1. Se for øvrig '/mosart/res/mo74_1' for den nyeste anvendelsen for pensjonsformuestatistikken.

På grunn av et stort antall simuleringer bør/må tabelluttaket begrenses til et minimum. Det foreligger to separate shellscript som kan rydde opp i disse simuleringene ('delete_repeated_simulations', 'delete_failed_repeated_simulations', se Vedlegg C). Ofte mislykkes *enkelte* av simuleringene av maskintekniske årsaker, gjerne uten noen feilmeldinger i 'sim.errors'. Disse simuleringene må da slettes. Eksekverer man deretter 'jobr.exe' på nytt, så er det bare de simuleringene som mangler som blir eksekvert. Normalt går da simuleringen gjennom uten problemer, med mindre det faktisk var en feil i simuleringsmodellen som var årsaken.

Eksekvering i monodevelop
Det er også mulig å eksekvere modellen i monodevelop. Dette er i første rekke aktuelt ved redigering av modellen. Da kan det være formålstjenlig å etterprøve nylig innlagte endringer uten å gå omveien om kompilering og påfølgende eksekvering. Dette gjelder spesielt ved feilsøking, som er betydelig enklere i et miljø som monodevelop, da dette tilbyr mye nyttig funksjonalitet som med fordel kan benyttes.

Merk at eksekvering i monodevelop legger på en del overhead som medfører at kjøretiden øker, slik at etter at man er sikker på at endringene i modellen er rimelige bør benytte den ordinære framgangsmåten med oppstart ved 'job.exe'. En kort oversikt over redigering og eksekvering av modellen i monodevelop er gitt i Vedlegg N.

2.6. Forbruk av regnekraft

Forbruket av regnekraft i MOSART er såpass stort at man bør være oppmerksom på kjøretid, internminne og diskplass ved drift av modellen. Med et utvalg på 100 prosent av befolkningen, standard forutsetninger, to pensjonssystemer og lang simuleringshorisont kreves det anslagsvis 120 GB internminne. Skal man sikre effektiv drift, blant annet av hensyn til opprydning i internminnet (garbage collector), bør det dobbelte være til rådighet. Serveren 'sl-mosart-01' har 1 TB internminne tilgjengelig. Simuleringer med mange parallelle pensjonssystemer øker internminnebehovet bortimot proporsjonalt med antall pensjonssystemer. Beregninger av nåverdien av folketrygden øker både internminnebehovet og kjøretiden betydelig.

Vesentlig behov for diskplass oppstår først når man skal produsere modellpopulasjonen, og plassbehovet kan enkelt beregnes ved å multiplisere forventet antall personer × år i modellpopulasjonen med antall felter hver observasjon tar (antall felter framgår av feltbeskrivelsen i 'population.record', og denne kan produseres ved å eksekvere simuleringen med 'read_only_parameters' på først, se Vedlegg E). Tilgjengelig plass på disken finnes ved å skrive 'df -k /mosart'. Det er viktig å fjerne modellpopulasjoner når man er ferdig med å bruke de, modellpopulasjonene kan relativt enkelt gjenopprettes ved å eksekvere simuleringen på nytt med samme forutsetninger. Det finnes noen shellscript under '/mosart/bin' som gjør det lettere å rydde opp i simuleringer med videre, se 'delete_*' i Vedlegg C.

Anslaget på kjøretid avhenger av mange forhold, tallene under er rapportert med full parallellitet i simuleringen. Innlesing av overgangssannsynligheter og øvrig oppsett av modellen tar anslagsvis 12 sekunder. Innlesing av utgangspopulasjonen med hele befolkningen tar anslagsvis 4-6 minutter. Hvert enkelt år uten justering mot historiske tall, ekstra kontroll av husholdskjennetegn, produksjon av modellpopulasjon og tabeller tar anslagsvis 40-50 sekunder de første årene. Høy befolkningsvekst øker kjøretiden over tid. En standardsimulering, inkludert middelalternativet i befolkningsframskrivingene og et omfattende sett av tabeller, tar anslagsvis 1,5 timer fram til år 2100. Tilbakegående tabeller øker tiden det tar å lese inn utgangspopulasjonen og fullføre tabelluttaket i

startåret. Mange parallelle simuleringer vil også øke kjøretiden betraktelig, spesielt hvis man nærmer seg eller går utover faktisk internminne på serveren.

2.7. Flere pensjonssystemer

I MOSART er det mulig å beregne ytelsene fra flere (plurale) pensjonssystemer samtidig. Stokastikken holdes da konstant, slik at begivenhetene som inntreffer er de samme for hvert pensjonssystem. Dermed er det kun pensjonsutbetalingene som er forskjellige i en simulering. Dette forenkler sammenlikningen mellom egenskapene til to eller flere pensjonssystemer, i enkelte tilfeller kan det *radikalt* redusere den effekten trekningene har på anslaget på *differansen* mellom to slike pensjonssystemer.

Merk at pensjonssystemer her også omfatter beregninger av skatt, inntekt og sparing.

I kildekoden har indeksvariabelen for pensjonssystemer navnet 'ps', med følgende hjelpevariabler (blant annet); 'g.no_of_pension_systems', 'g.pension_system_one' (se under), 'g.no_pension_system' (i hovedsak i trekkemetodene) og 'g.pension_systems' (settet av alle pensjonssystemer i simuleringen). Merk at pensjonssystem 1 også har tallverdien 1 (i motsetning til C# som typisk har 0 som første element i en vektor).

I den grad pensjonssystemet har tilbakevirkning på begivenhetene som simuleres, så er det pensjonssystem 1 som får denne rollen. Det ligger også noen begrensninger på hva det faktisk er meningsfylt å regne på, i og med at livsløpene skal være like i de ulike pensjonssystemene. Legg spesielt merke til to parametere som styrer forbindelser mellom pensjonssystemer. 'actuarial_parameters_substitute_pension_system' i 'pension_parameters_*i*.con' bestemmer om pensjoneringsmønsteret tvunget skal følge et annet pensjonssystem, eller om det er reglene i det gjeldende pensjonssystemet som skal brukes (gjelder for eksempel krav til opptjening ved tidligpensjonering). Den andre er 'savings_ratio' i 'tax_parameters.v*iiii*.con' som bestemmer hvor mye av differansen i disponibel inntekt (forårsaket av endringer i skattesystemet) som skal spares.

Antall pensjonssystemer som skal omfattes av simuleringen spesifiseres i filen 'parameters.con', ved parameteren 'no_of_pension_systems'. Egenskapene for hvert av pensjonssystemene beskrives i hver sin utgave av styrefilen 'pension_parameters_*i*.con'. Disse må navngis i henhold til det pensjonssystemet de beskriver etter strukturen 'pension_parameters_*i*.con', der *i* angir hvilket pensjonssystem spesifikasjonen gjelder for. Dersom man eksempelvis ønsker å få beregnet ytelser etter 3 forskjellige pensjonssystemer, er det nødvendig med 3 utgaver av styrefilen, som må hete 'pension_parameters.con', 'pension_parameters_2.con' og 'pension_parameters_3.con'. Legg spesielt merke til at pensjonssystem 1 spesifiseres i 'pension_parameters.con'. Opprettes ikke 'pension_parameters_*i*.con', benytter simuleringen seg av tilsvarende parametere fra pensjonssystem 1.

I tillegg kan støtteparametere velges etter samme lest, da ved 'tax_system_*i*' i 'parameters.con', og grunnbeløp og særtillegg ved 'time_series_basic_amount_*i*', 'time_series_basic_amount_index_*i*' og 'time_series_special_supplement_*i*' i 'input.con'.

Alternativt (og anbefalt) kan man droppe bruk av parameterfilene med indeksering for pensjonssystem ('_*i*'), og heller legge de faktiske forskjellene inn i 'change_parameters.con'. Det blir da vesentlig enklere å holde oversikten og kontrollen på hva som ligger i hvert pensjonssystem.

Bruker man flere pensjonssystemer skal alle forskjeller i parameterverdier logges til fila 'sim.report.multiple_parameters'.

Resultatene skrives ut til separate filer, ofte navngitt ved 'r*_ps*i*.prn'.

Dersom det er angitt at modellpopulasjonen skal skrives ut (se Vedlegg H), vil de kjennetegn som omfattes av pensjonssystemer skrives ut flere ganger, dersom disse er valgt i 'model_population_file.con'.

MOSART kan ha problemer med å eksekvere *mange* pensjonssystemer på en gang når man samtidig simulerer hele populasjonen. En mulighet er å dele arbeidsoppgaven på flere simuleringer, men hvor pensjonssystem 1 er felles i alle simuleringene og random-seed også holdes uendret.

3. Simuleringsmodellen

Siktemålet med kapittel 3 er å lette arbeidet med å komme i gang med å sette seg inn i og videreutvikle applikasjonen som utgjør simuleringsmodellen MOSART. Omtalen holdes på et relativt overfladisk og generelt nivå, spesielt fordi modellen endrer seg stadig og vi ikke oppdaterer denne dokumentasjon ved enhver endring av modellen. Det er derfor en forutsetning at man samtidig har tilgang til de filene som utgjør og brukes av simuleringsmodellen (se Vedlegg D), samt dokumentasjon av modellen og en manual for C#. Det er et krav at kildekoden i seg selv skal være lesbar og utgjøre en vesentlig del av dokumentasjonen. Kapitlet gir en kort innføring i C# etterfulgt av noen praktiske råd for hvordan nye modellvarianter bør utvikles. Avsnitt 3.5 gir en viss oversikt over arkitekturen i simuleringsmodellen MOSART. Avsnitt 3.6 går inn på hvordan man kan få utnyttet regnekraften best mulig, da dette er en forutsetning for å kunne simulere hele befolkningen. Vedlegg G-K omtaler noen av modulene i litt mer detalj.

Det finnes mye informasjon om C# tilgjengelig på internett. Nettsiden https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ kan brukes som manual og oppslagsverk til C#, siden har en intern og velfungerende søkemotor og oppslagene er korte og godt forklart. Wikipedia (på engelsk) har en lengre side om «C Sharp (programming language)» som gir en introduksjon til C# og lenker videre til annen informasjon. Wikipedia på norsk har bare en stubb om C#. Ellers vil kompilatoren og simuleringene noen ganger skape feilmeldinger som er helt uforståelige. Ved å ta et nettsøk på noen av de (lengre og) idiosynkratiske frasene som inngår i feilmeldingen, kan man ofte få opp mye relevant informasjon (eller i det minste få del i andres frustrasjon over problemet).

3.1. Kildekode, kompilator og eksekvering av programmer i C#

Kildekoden til en applikasjon eller et program i C# ligger på en tekstfil med navnestruktur '*navn-på-program*.cs'. For å kunne eksekvere dette programmet må man kompilere det først, det vil si oversette teksten i kildekoden til instruksjoner som prosessoren forstår (maskinspråk). Kompileringen skjer ved følgende kommando (i prinsippet):

```
> mcs navn-på-program.cs [↵]
```

Kompilatoren 'mcs' kontrollerer blant annet at kildekoden er fri for syntaksfeil, det vil si tekst hvor kompilatoren ikke forstår hva applikasjonen vil. Kompilering av en applikasjon med syntaksfeil vil normalt gi rimelig forståelige feilmeldinger for en øvet utvikler. Når programmet er fritt for syntaksfeil får man ingen beskjed, og kompilatoren har laget en fil med navn '*navn-på-program*.exe' (noen ganger vil kompilatoren varsle om irregulariteter, men likevel til slutt melde at kompileringen var vellykket og legge ut en fil). Kompilerer man en ny versjon av et program (samme navn) blir den forrige versjonen av '*navn-på-program*.exe' overskrevet. Merk at når en slik rekompilering mislykkes vil den forrige versjonen av '*navn-på-program*.exe' bli liggende. Programmet kan nå eksekveres ved å skrive kommandoen (i prinsippet):

```
> mono navn-på-program.exe & [↵]
```

Ved å skrive 'man mcs' og 'man mono' i Linux vil man få en nærmere dokumentasjon av kompileringen og valg av opsjoner. I praksis må det følge med mange opsjoner med kompileringen, og vi har derfor flere shellscript som utfører oppgaven. Enkle testprogrammer hvor kildekoden er samlet på en fil kan kompileres ved:

```
> tkomp navn-på-program [↵]
```

Shellscriptet 'tkomp' følger med hver modellvariant. Programmet kan eksekveres ved:

```
> t.exe navn-på-program & [↵]
```

Shellscriptet 't.exe' følger også med hver modellvariant. Feilmeldinger og annen utskrift til skjerm fra simuleringen legges ut på en fil med navn '*navn-på-program*.errors'. Ved kompilering av programmer som er delt over flere filer må '*navn-på-program*.cs' erstattes med en liste med alle filene. Filene kan komme i en vilkårlig rekkefølge. Kompileringen av simuleringsmodellen gjøres med et eget shellscript, 'komp', se Avsnitt 3.4.

3.2. Noen grunnleggende begreper i C#

I dette avsnittet går vi kort gjennom noen grunnleggende begreper i C#. Noen av disse begrepene er vanskelige å forklare uten at man kan de andre (nye) begrepene, for eksempel klasse og objekt. Avsnittet bør av den grunn leses (minst) to ganger. Begrepene er forsøkt konkretisert med eksempler fra MOSART og mer enkle typeeksempler. I de enkle eksemplene kan C# virke overdrevet omstendelig, men i simuleringsmodellen som håndterer 5-10 millioner komplekse livsløp, er objektorienteringen nødvendig, med hensyn til både regnekraft og robusthet (unngå skjulte feil).

Mal for et enkelt program i C#

Kildekoden i et enkelt program i C# består av vanlig tekst og har en struktur som angitt i boks 3.2.2. Begrepene i boksen forklares i det følgende i omtrent den rekkefølgen de kommer i boksen. Noen steder angir vi i parentes det engelske navnet for de begrepene vi bruker på norsk. Boks 3.2.1 har et eksempel på et enkelt program som mer konkret angir hva de ulike elementene står for.

Boks 3.2.1. Et enkelt program

```
// Program som beregner areal av en sirkel
using System;

class finn_areal_av_sirkel
{
        static void Main()
        {
                // Referansen fas gir tilgang til metoder i programmet
                finn_areal_av_sirkel fas = new finn_areal_av_sirkel();

                // Opprett et objekt av typen sirkel
                Sirkel sirkel = new Sirkel();
                sirkel.radius = 5.67;
                sirkel.areal = fas.Beregn_areal_av_sirkel(sirkel.radius);

                // Skriv til skjerm
                Console.WriteLine("Radius : " + sirkel.radius);
                Console.WriteLine("Areal : " + sirkel.areal);
        }

        // Funksjon som en metode
        public double Beregn_areal_av_sirkel(double radius)
        {
                return 3.14 * radius * radius;
        }

        // Klasse for sirkel;
        class Sirkel
        {
                public double radius;
                public double areal;
        }
}
```

Boks 3.2.2. Mal for et enkelt program i C#

```
// Dokumentasjon

using System;
using System.navn-på-bibliotek;          // Referanse til eksterne bibliotek

class navn-på-program
{
        static void Main()               // Startpunktet for simuleringen
        {
                Instruksjoner;
        }

        Deklarasjon av metoder

        Deklarasjon av klasser
}
```

Skrifttyper

Boks 3.2.1 og andre eksempler på kildekode i denne dokumentasjonen følger et mønster når det gjelder skrifttyper. Uthevet skrift er navn/nøkkelord (keyword) som tilhører C# eller operativ-systemet, typiske eksempler kan være utsagn som 'if' og 'while'. Vanlig skrift er brukerdefinerte variabelnavn. Skrift i kursiv er enten helt vilkårlige forslag til variabelnavn og/eller deler av applikasjonen som det er valgfritt om inngår.

Merk at i kildekoden i C# må man skille mellom små og store bokstaver.

Kommentarer

Tekst som er kapslet inn av '//* ... *//' og all tekst på en linje som kommer etter '//', er kommentarer som C# utelater fra kompileringen. Teknisk sett er kommentarene unødvendige og kan komme hvor som helst i kildekoden. Noen ganger kan kommentarer likevel gjøre programmet adskillig mer lesbart ved at kommentarene skaper overskrifter i kildekoden og/eller på et mer overordnet plan forklarer hva kildekoden gjør. En omfattende og komplisert kildekode bør alltid innledes av noen kommentarlinjer som forklarer innholdet. I MOSART har vi valgt å bruke '//** ... **' for innramming av kommentarer.

Parenteser og tekstkvalifisatorer

I C# er *tekst* (string) og *bokstaver* (char) egne variabeltyper. Når man skal tilordne slike variabler verdier i teksten, må de kapsles inn av " (tekst) eller ' (bokstaver). Alle parentestyper ((, {, []) tilhører C#, og det som er mellom to tilhørende parenteser har kun gyldighet innenfor sine to parenteser. Spesielt vil klammeparenteser ({ ... }) kapsle inn sekvenser av kildekode som hører sammen, også kalt en **blokk**. Klammeparenteser brukes blant annet der man skal avgrense en metode eller klasse, eller når man skal utføre flere instruksjoner samlet etter en test (if) eller i en løkke (for, foreach, while). I en del andre programmeringsspråk brukes tilsvarende 'do-end' eller 'begin-end' for å kapsle inn sekvenser av instruksjoner som hører sammen.

Alle parenteser og tekstkvalifisatorer må være **balanserte**, det vil si at hver gang man har en innledende parentes, må det også være en tilhørende avsluttende parentes, og hver gang man

begynner på en tekststreng med en ", må man også avslutte tekststrengen med en ". Ubalanserte parenteser og tekstkvalifisatorer kan gi særdeles kompliserte feilmeldinger, spesielt i omfattende og kompliserte programmer, og det vil ofte være uklart hvor i kildekoden problemet ligger. Ved å kompilere underveis, i den grad det er mulig, kan man lettere lokalisere slik feil. Ved å dele programmet på flere filer kan også det begrense antallet linjer man må lete gjennom.

Eksterne bibliotek

Tilknyttet C# finnes en rekke eksterne bibliotek som inneholder mer avanserte metoder og klasser enn det som inngår i standardversjonen av C#. Disse bibliotekene inkluderes i programmet ved referansene i starten av filen, 'System' er nesten obligatorisk å bruke, den neste linja angir at man også inkluderer et (hypotetisk) bibliotek med navn '*navn-på-bibliotek*'.

Programmet

Utsagnet 'class *navn-på-program* { ... }' kapsler inn programmet, og det er en fordel om navnet på filen med kildekoden har samme navn som klassen (class) som definerer programmet. Disse to navnene behøver ikke være sammenfallende, og noen ganger oppstår det navnekollisjoner mellom filnavn, klasser og referanser til objekter av en klasse. I MOSART vil klassenavn ofte ha en forstavelse av typen 't_*navn-på-klasse*'.

Utsagnet 'static void Main () { ... }' er en metode (se nedenfor) som er startpunktet for applikasjonen. Det vil si at Main eksekveres først, og at alt blir nøstet opp derfra. Det betyr også at applikasjonen må ha en og bare en forekomst av Main.

Instruksjoner

En instruksjon (statement) i C# er en bit av kildekoden som forteller hva programmet skal gjøre og som avsluttes med et semikolon (;). Instruksjoner blir alltid eksekvert innenfor en metode, enten dette er Main eller en egendefinert metode. Instruksjonen kan gjerne gå over flere linjer. Et eksempel på en enkel instruksjon er:

```
double pi = 3.14;
```

Her deklareres en variabel av typen flyttall (double) med navn 'pi' og blir samtidig initialisert med en tallverdi på 3,14. Alle variabler (og referanser) må deklareres *før* de kan brukes, men de kan deklareres samtidig som de tilordnes en verdi (initialisering).

Metode

En metode i C# er en sekvens instruksjoner som kan kalles samlet.

Boks 3.2.3. Metode

```
// Dokumentasjon
public void metode-navn(parametere)
{
        instruksjoner;
}
```

En metode deklareres alltid innenfor en klasse, enten dette er programmet selv eller en egendefinert klasse. En metode skal alltid tilordnes et navn (*metode-navn*), gjerne intuitivt, språklig korrekt, systematisk og uten forkortelser. Man kan angi tilgjengeligheten for metoden, public gir maksimal tilgjengelighet. En metode skal alltid ha en type returverdi (return value), void betyr at metoden ikke returnerer noen resultater. Se boks 3.2.1 for en metode som har en returverdi (Beregn_areal_av_sirkel). En metode kan ha parametere hvor man overfører informasjon til metoden. I den grad metoden bearbeider disse parameterne, blir endringene i parameterne normalt ikke tilbakeført til den delen av programmet som kalte på metoden. Metoden kan kalles (eksekveres) av et objekt som er av den klassen som deklarerer metoden, se boks 3.2.1 for et eksempel.

Bruk av egendefinerte metoder er viktige av minst to grunner, metoder kan gi en oppdeling av programmet og metoder kan brukes til å standardisere og sentralisere arbeidsoppgaver.

En applikasjon i C# kan fort bli meget omfattende, og for eksempel vil kildekoden i simuleringsmodellen MOSART omfatte 139 000 linjer. Skal denne applikasjonen være lesbar må settet av instruksjoner deles opp i sekvenser som utfører ulike deler av simuleringen. Filen 'model.cs' i MOSART vil kun bestå av metoden Main og er på 822 linjer, selve simuleringen er på om lag 446 linjer. Dette er lesbart for en som kjenner MOSART og kan litt om C#. Hver linje vil her typisk kalle på en ekstern metode som utfører en del av simuleringen, for eksempel beregning av skatt for hele populasjonen for inneværende år. Denne moduliseringen må normalt gå i flere trinn før man har brutt ned programmet i deler som er så små og oversiktlige at de blir lesbare. Moduliseringen gir programmet struktur/arkitektur, men det er likevel helt avgjørende at man gir disse metodene gode navn, navn som gjør at man umiddelbart gjenkjenner innholdet.

Den andre viktige bruken er når man har en arbeidsoppgave med et visst omfang som gjentas på nesten samme måte flere steder i programmet. Hvis man klarer å generalisere denne arbeidsoppgaven kan man samle settet av instruksjoner i en metode. Kildekoden som kaller på metoden blir kortere og mer lettlest, utviklingsjobben forenkles også. En sentralisert metode forenkler også vedlikeholdet av modellen i de tilfeller man ønsker og endre eller utvikle en slik metode. Dette kommer spesielt til anvendelse i implementeringen av pensjonsregler.

Klasser, objekter, referanser

En klasse i C# er en mal for håndtering av informasjon som hører sammen:

Boks 3.2.4. Klasse

```
// Dokumentasjon
public class klasse-navn
{
        public klasse-navn(parametere)          // Konstruktør
        {
                instruksjoner;
        }

        Deklarasjon av variabler og referanser  // Informasjonsinnhold

        Deklarasjon av metoder                  // Håndtering av informasjonen

        Deklarasjon av klasser
}
```

I MOSART vil for eksempel klassen 'person' definere hvilke kjennetegn en person kan (skal) ha og noen metoder for å håndtere disse kjennetegnene. Et objekt vil være en realisasjon (en bestemt person), mens referanser vil gi tilgang til denne informasjonen (og metodene):

Boks 3.2.5. Objekter og referanser

```
klasse-navn navn1-på-referanse;         // Referanse deklareres
klasse-navn navn2-på-referanse;         // Referanse deklareres

// Tilordning av referanse og oppretting av objekt
navn1-på-referanse = new klasse-navn(parametere);

navn2-på-referanse = navn1-på-referanse;    // Tilordning av referanse

navn1-på-referanse = null;                  // Sletting av referanse
```

Tilsvarende som for variabler må også referanser deklareres før eller samtidig med at de tas i bruk. Et objekt opprettes ved instruksjonen 'new', og da settes det av en del av internminnet som lagrer dette objektet. Samtidig blir konstruktøren (constructor) eksekvert (se boks 3.2.4). Konstruktøren er en metode som deklareres i klassen og har samme navn som klassen. Konstruktøren skal ikke og kan ikke ha noen returverdi. En klasse kan ha flere konstruktører, bare de har forskjellige sett av parametere. Instruksjonen 'new' må eksekveres med de samme typene parametere som den konstruktøren man ønsker å bruke. En klasse må ikke nødvendigvis ha en konstruktør, da brukes 'new' uten parametere ('new *klasse-navn*()').

Når man oppretter et objekt er det vanlig at man samtidig tilordner en referanse til dette objektet, som i boks 3.2.5. I den neste linja tilordnes en annen referanse det samme objektet (en peker til den samme adressen). I den siste linja slettes innholdet i den første referansen. Merk at dette i seg selv ikke har konsekvenser for objektet. I det øyeblikket programmet ikke har noen referanser til

objektet, så er objektet tapt for programmet, og på et senere tidspunkt vil internminnet bli frigjort av garbage collector.

Metafor: Arkitekttegningen er klassen, huset er objektet, adresseopplysningene i folkeregisteret og telefonkatalogen er referanser.

3.3. Mer om C#

Avsnitt 3.3 går gjennom noen mer avanserte sider ved C# med spesiell relevans for MOSART.

Matriser og vektorer (array) [15]

I C# kan man jobbe på n-dimensjonale matriser (array), og dette er ofte en forutsetning for å kunne behandle informasjon på en effektiv og generell måte. Eksempelet under viser en todimensjonal matrise med tekst (string), hvor man setter opp en tabell med fornavn og etternavn og til slutt skriver ut alle personene med en enkel løkke (**for**). Merk at matrisen (*tabell*) er en referanse, hvor hvert element i dette tilfellet er en ordinær variabel (string). Innholdet i hver celle kunne også vært en referanse, for eksempel en ny underliggende matrise. Merk også at matriser i C# er nullbaserte, det vil si at første element i en dimensjon har indeks 0.

```
// Opprett to-dimensjonal tabell med tilhørende størrelse i hver dimensjon
string [,] tabell = new string [3, 2];

// Fyll tabellen med innhold
tabell[0, 0] = "Ole";
tabell[0, 1] = "Olsen";
tabell[1, 0] = "Hans";
tabell[1, 1] = "Hansen";
tabell[2, 0] = "Lars";
tabell[2, 1] = "Larsen";

// Skriv ut tabellen
for (int i = 0; i < 3; i++)
        Console.WriteLine(tabell[i, 0] + " " + tabell[i, 1];
```

List

C# har en type lister kalt 'List', som i mange sammenhenger kan behandle variable mengder informasjon på en veldig effektiv og fleksibel måte. List vil ha mye til felles med (1-dimensjonale) matriser, men kan endre størrelse underveis og har i tillegg en rekke metoder som gir lettere tilgang til informasjonen, samt sortering. Vi viser til kildekoden i MOSART og til Internett for mere informasjon. Merk at bruk av List på store datamengder, eller når de opprettes i et stort antall, kan ha betydelig negativ innvirkning på kjøretider og stabilitet, se avsnitt 3.6.

[15] Vi bruker de norske ordene vektor og matrise om hverandre i notatet, men begge deler refererer til begrepet 'array' i C#, med vektor som en 1-dimensjonal-matrise.

```
// Opprett liste
List <int> liste = new List <int> ();

// Fyll liste med innhold
liste.Add(314);
liste.Add(271);
liste.Add(1024);

// Skriv ut liste
foreach (int tall in liste)
        Console.WriteLine(" " + tall);
```

Klasse

Bruken av objekter gjør at all informasjon om for eksempel en person kan samles på ett og bare ett sted. Klasser gjør at innholdet i objektet kan gis en meget fleksibel utforming. Referanser gjør at mange kan få tilgang til informasjonen, eller mer presist, til enhver tid ha tilgang til informasjonen uten å måtte lete opp det aktuelle objektet.

Innkapslingen i et objekt gjør at man skjermer informasjonen mot irregulær bruk. Ved å deklarere en variabel som 'private' (i motsetning til 'public'), vil variabelen kun være tilgjengelig for metoder som er deklarert i klassen. På den måten kan man kontrollere hvordan en variabel kan bli endret. Slik **lokal programmering** vil redusere faren for skjulte feil gjennom utilsiktede endringer andre steder i applikasjonen. Tilsvarende i metoder bør man ikke deklarere en variabel før man trenger den, i det minste ikke på et høyere nivå enn der den brukes. Angir man ikke tilgjengelighet, bruker C# 'private' som standard.

Bruken av referanser gjør det vesentlig enklere å håndtere relasjoner mellom objekter, for eksempel foreldre-barn, ektepar og husholdninger. Man kan raskt og relativt kostnadsfritt hente ut informasjon om de personene man er relatert til, eventuelt bringe videre slik informasjon. I MOSART hvor vi simulerer familierelasjoner for hele populasjonen er dette nesten en nødvendig egenskap ved programmeringsspråket. Det er også *mulig* å håndtere hele populasjonen som en egendefinert liste, ved at hver person har en referanse til neste (og helst forrige) person i lista. Det er da mulig og relativt lite ressurskrevende å legge til eller trekke ut en person av lista, ved å koble om referansene til forrige-neste. Vi bruker denne typer lister lite i MOSART, fordi List ut fra en totalvurdering er mye mer effektiv.

Klasser, arv og polymorfisme

I C# kan en klasse være **avledet** (derived class) fra en **grunnklasse** (base class), og vil **arve** alle egenskapene i grunnklassen (inheritance). Fordelen er at man kan utvikle en felles infrastruktur og (hjelpe)metoder i grunnklassen, som kan komme til nytte i ulike anvendelser. Viktige anvendelser av såkalt arv i MOSART er innlesing av filer ('read_base.cs', 'class t_par'), tabelluttak ('table_base.cs', 'class t_table_base') og trekkemetoder ('draw_base.cs', 'class t_draw_base'). Ved arv kan man deklarere malen til en metode i grunnklassen ved å legge til nøkkelordet **virtual** i deklarasjonen, se boks 3.3.1. I en avledet klasse kan man gi metoden nytt innhold ved å deklarere metoden på nytt, nå med nøkkelordet **override**. Den eneste restriksjonen er at den nye metoden må ha samme ytre ramme eller signatur som før (tilgjengelighet, returverdi, type parametere).

Et objekt som er opprettet fra en avledet klasse, kan refereres til med en hvilken som helst av de klassene objektet tilhører (**polymorphism**), se igjen boks 3.3.1. Det er dermed mulig å bygge opp en liste av objekter med forskjellig innhold, avhengig av hva som skjer i simuleringen, men samtidig

kunne håndtere disse objektene på samme måte med hensyn til de grunnleggende egenskapene. Man kan for eksempel gjennom parameterfilene få laget lister av kjennetegn som skal skrives ut med et minimum av kildekode. Det framgår kanskje ikke av eksempelet i boks 3.3.1, men når det er hundrevis av variabler, og man skal få med både variabelverdier, formater og dokumentasjon, så blir det nesten avgjørende. Polymorfisme brukes mye i utskriften av modellpopulasjonen og i tabelluttaket.

Boks 3.3.1. Klasser og arv

```
// Skriv ut id-numre og alder
using System;
using System.Linq;

class program
{
        static void Main()
        {
                // Opprett liste med kjennetegn som skal skrives ut
                List <grunnklasse> liste = new List <grunnklasse> ();
                liste.Add(new t_id_number());
                liste.Add(new t_age());

                // Anta at populasjonsliste finnes
                // Anta 'class person' har kjennetegnene id_number og age

                // Skriv til skjerm
                foreach (person p in populasjonsliste)
                {
                        foreach (grunnklasse v in liste)
                                Console.Write(" " + v.x(p));
                        Console.WriteLine();
                }
        }

        class grunnklasse
        {
                public virtual double x(person p) {return 0;}
        }

        class t_id_number : base grunnklasse
        {
                public override double x(person p) {return person.id_number;}
        }

        class t_age : base grunnklasse
        {
                public override double x(person p) {return person.age;}
        }
}
```

Metoder som variabler (delegate)

Eksemplet med virtual-override viste hvordan man kunne endre innholdet i en metode. Det finnes en mer direkte måte å gjøre dette på gjennom bruk av delegate, se eksemplet under. Poenget er at '*navn*' er en metode som kan brukes i simuleringen, men denne kan gis ulikt innhold *underveis* i simuleringen, bare metodene den tilordnes har samme type og parametere.

```
// Del av kildekode
{
        navn = new t_navn(variant1);            // Metoden navn tilordnes variant1
}

public t_navn navn;
public delegate void t_navn(parametere);

public void variant1(parametere)
{
        instruksjoner;
}
```

Variabeltyper (value types)

I C# vil variabler være av en bestemt type, være tilordnet en verdi, og verdien vil kun være tilgjengelig gjennom variabelen. Det er spesielt numerisk informasjon som har mange typer, tilpasset om det er heltall eller flyttall, og tilhørende krav til variasjonsområde eller presisjon. **Flyttall** er metoden for å representere reelle tall i beregningene, ved at tallet er på eksponentiell form med et endelig antall desimaler.

Jo større presisjon eller variasjonsområde, jo mere plass må settes av i internminnet. Standardtypene er double og int. Variabeltypen **double** er flyttall, har et presisjonsnivå 15-16 siffer, og beslaglegger 8 byte. Variabeltypen **int** er heltall, har variasjonsområde (-2.147.483.648, 2.147.483.647), og beslaglegger 4 byte. Vår erfaring er at C# behandler variabeltypene double og int klart mest effektivt i regneoperasjoner, og at de bør foretrekkes med mindre det er *nødvendig* med større presisjon (decimal), større variasjonsområde (long) eller gir en *vesentlig* reduksjon i bruk av internminne (spesielt Array og List med byte, short).

Felles for heltallsvariablene er at de har øvre og nedre grenser for hvor store verdier de kan anta, avhengig av hvor mye plass de tillates å beslaglegge i internminnet. Noen ganger når simuleringen går utover de tillatte verdiene vil simuleringen stoppe (som er bra), noe ganger vil den tilordne verdier ved en modulo-operasjon (som kan gi ekstremt subtile feil).

```
byte x = 250;       // byte har tillatte verdier [0,255]
x += 10;            // x har nå verdien -4, gitt ved restverdien av 260 delt på 256

short z = 32000;    // short har tillatte verdier [-32768,32767]
z += 1000;          // z har nå verdien -32536, gitt ved restverdien av: 33000-nedre grense
                    // delt på 65536, pluss nedre grense
```

Spesielt ved bruk små heltallstyper (byte) så kan dette gi skjulte feil som er vanskelig å oppdage. Ved konvertering av variabler er det også mer robust å bruke metodene 'Convert.*' framfor 'cast', jamfør eksempelet:

```
int x = 300;
byte z = (byte) x;                    // z får verdien 44, etter samme regel som over;
byte w = Convert.ToByte(x);           // Simuleringen stanses med en feilmelding (overflow);
```

Metoder og overføring av parametere med 'ref'

En metode kan kalles med parametere. Normalt kan metoden gjøre hva den vil med disse parameterne, og dersom metoden endrer verdien av en overført parameter vil denne likevel forbli uendret når kontrollen returnerer til der metoden ble kalt fra. Dersom det er ønskelig å arbeide videre med den oppdaterte verdien kan parameteren overføres ved hjelp av nøkkelordet '**ref**', se boks 3.3.2.

Dette gjelder ordinære variabler. Når parameteren er en referanse, så er det hva referansen peker til som ikke blir oppdatert, men innholdet i objektet som referansen peker til blir endret, også utenfor metoden. Elementer som List og matriser er referanser, og innholdet i tabellene er objekter, og det er viktig å være klar over dette. Eksemplet i boks 3.3.2 belyser problemet.

Bokse 3.3.2. Overføring av parametere

```
// Del av kildekode
{
        int [] tabell = new int [2];
        tabell[1] = 22;

        int variabel1 = 33;
        int variabel2 = 44;

        Compute(tabell, variabel1, ref variabel2);
        Console.WriteLine(" " + tabell[1] + " " + variabel1 + " " + variabel2);
}

public void Compute(int [] tabell, int variabel1, ref int variabel2)
{
        tabell[1] = 55;   // Overføres
        tabell = null;            // Overføres ikke
        variabel1 = 66;  // Overføres ikke
        variabel2 = 77;  // Overføres
}
```

Her vil verdien av '*variabel1*', når simuleringen kommer til utskriftssetningen (Console.WriteLine(...)), fortsatt være '33' mens verdien av '*variabel2*' er endret til '77'. Siden '*variabel2*' er overført med 'ref' vil den oppdateringen som er gjort i metoden '*Compute*' gjelde videre i simuleringen. '*Variabel1*' er ikke overført med 'ref' slik at oppdateringen forsvinner når simuleringen går ut av metoden igjen.

Selv om '*tabell*' ikke er overført med 'ref' vil det likevel være dens oppdaterte verdier som beholdes i objektet/tabellen etter kallet på '*Compute*'. Dersom dette ikke er ønskelig må dens verdier tas vare på i hovedprogrammet, slik at '*tabell*' kan tilbakestilles med sine opprinnelige verdier etter kallet på '*Compute*'. Ønsker man å endre referansen '*tabell*', må man overføre også den med nøkkelordet 'ref'.

Om koding av konstanter

Tall som inngår i kildekoden bør i nesten alle tilfeller kodes med variabelnavn, av minst tre grunner. Det øker lesbarheten, det reduserer faren for feil, og det gjør det mulig å endre denne tallverdien fra et punkt i kildekoden. Slike konstanter kan gjerne deklareres som 'readonly', da det synliggjør at det er en konstant, og forhindrer at den endrer verdi andre steder i kildekoden. Hadde man i stedet kodet 'male' og 'female' som henholdsvis '1' og '2' ville det vært vanskelig å se hva det er man har puttet inn som argument i et metodekall eller en matrise. Og det ville vært et komplett mareritt hvis man skulle endre disse tallverdiene, for eksempel la de bytte plass. Man måtte da ha oppsøkt alle 1- og 2-tall i kildekoden, vurdert om det var kjønn, og så endret verdien.[16] Dette er en mer hypotetisk endring (kanskje), men for andre variabler som er i utvikling er det langt mer påkrevet. Det gjelder både sett av statusverdier (som her), men også for parametere i pensjonssystemet man antar er så faste (eller marginale) at man ikke har lagt det inn i 'pension_parameters.con', for eksempel 'actual_pension_reform_year'.

```
public readonly int male = 1;

public readonly int female = 2;
```

Eksterne moduler

Deklarasjonen av klasser kan legges ut på egne filer. På denne måten kan omfanget av hovedprogrammet reduseres betydelig, og bidra til økt lesbarhet. Et slikt modulisert hovedprogram må kompileres på en litt annen måte (i prinsippet):

```
> mcs fil-1.cs ... fil-n.cs [↵]
```

De ulike modulene kan komme i en vilkårlig rekkefølge. Kompileringen av MOSART gjøres lettest med shellscriptet 'komp' som følger med hver enkelt modellvariant. Feil som oppstår på grunn av feil bruk av parenteser og tekstkvalifikatorer er enklere å finne ut av når man har mange mindre filer, framfor en stor.

Ved siden av å redusere omfanget av filen med hovedprogrammet, vil bruk av eksterne moduler sikre at all kontakt mellom en metode (eller klasse) og omverdenen faktisk går gjennom metode- og klasseparameterne. Dermed blir risikoen for *utilsiktet* å endre en global variabel betydelig mindre (lokal programmering). Videre kan det bli enklere å slå sammen to modellvarianter som er utviklet fra en felles grunnversjon, spesielt hvis endringene berører ulike filer i de ulike variantene.

Opprydning av internminne (garbage collector)

En applikasjon (i C#) lagrer informasjonen den bruker i internminnet i serveren. Når informasjonen ikke skal brukes lenger, er det viktig at plassen i internminnet blir frigjort for andre formål. Applikasjonen vet når bruken av en ordinær variabel opphører, det skjer når det forlater den blokken hvor variabelen ble deklarert. Med objekter er det mer komplisert, blant annet fordi objektet selv ikke har noen kontroll på om andre deler av programmet har referanser til objektet. Frigjøring av internminne skjer når forbruket av internminne når et bestemt omfang, da stoppes eksekveringen av hovedprogrammet. Deretter blir en egen arbeidsoperasjon/eget program eksekvert (garbage collector). Ett prinsipp består i å merke eller flytte all informasjon som kan nås fra hovedprogrammet, de delene av internminnet som forblir umerket/uflyttet blir deretter frigjort

[16] Det vil fortsatt være krevende å gjøre denne endringen, fordi veldig mange av inputfilene i MOSART har kjønn som forklaringsvariabel/dimensjon, men det er det mulig å rette opp.

(slettet). Det er egne metoder for å unngå at det frigjorte internminnet ikke skal bli fragmentert. Når opprydningen er ferdig, fortsetter hovedprogrammet.

I MOSART vil opprydning av internminne stå for en vesentlig del av kjøretiden, hovedprogrammet settes tross alt på vent. Dette gjelder også etter at vi har optimalisert parameterne som styrer opprydningen og har modifisert kildekoden i retning av å belaste internminnet mindre. Opprydning av internminnet er også årsak til at noen simuleringer bryter sammen, eller at det oppstår andre maskintekniske feil. Det er derfor viktig å videreføre arbeidet med å belaste internminnet minst mulig, se avsnitt 3.6.

Informasjon om valg av metode for opprydning av internminne (Boehm, sgen) og valg av parameterverdier finnes ved å skrive 'man mono' og 'man sgen' i Linux. Noen av disse parameterne kan settes ved pålogging til Linux (filen '.cshrc' eller '.bashrc' på hjemmekatalogen). Sgen jobber noe saktere i starten av simuleringen hvor mye internminne skal allokeres for første gang, men tar igjen dette senere når jobben består i å rydde, spesielt ved store utvalg. Parameteren nursery-size bør settes til maksimalt nivå, for øyeblikket 4Gb (gigabyte).

Merk at det er umulig å slette et objekt i C#, og dette er en grunnleggende egenskap ved C#, da det gir mer hardfør kildekode. Sletter man et objekt man vet (tror) ikke lenger skal brukes, og antakelsen var feil, så vil det gi opphav til alvorlige feil det er vanskelig å finne ut av. Det er mulig å nullstille et objekt ved å endre verdien for en og en variabel i objektet, men det bare garbage collector som får slette objektet. Skal internminne brukt på et objekt frigjøres er det derfor avgjørende at alle referanser til objektet blir slettet.

3.4. Praktiske råd

Avsnitt 3.4 går gjennom en del praktiske råd for utvikling av nye varianter av simuleringsmodellen MOSART. Se også kapittel 2 for en gjennomgang av hvordan modellen eksekveres.

Ny modellvariant

Oppsettet til en ny modellvariant opprettes mest hensiktsmessig ved å bruke shellscriptet 'newmodel', se også Vedlegg C:

```
> /mosart/bin/newmodel navn-på-eksisterende-modellvariant navn-på-ny-modellvariant [↵]
```

'newmodel' oppretter en ny katalog med navnet '/mosart/prog/*navn-på-ny-modell-variant*', kopierer over alle nødvendige filer fra katalogen '/mosart/prog/*navn-på-eksisterende-modell-variant*', samt endrer referanser til modellnavn i 'input.con' og 'model_name.cs'. Brukergrensesnittet ('*.con', 'job.exe') behøver ikke endres med mindre man innfører nye parametere eller nye typer filer. Kildekoden ligger på filer med navnestruktur '*.cs', og disse redigeres ved en vanlig editor, for eksempel nedit eller emacs, eller med monodevelop.

Kompilering

Det inngår mye informasjon i kompileringen av simuleringsmodellen, men jobben kan gjøres med shellscriptet 'komp' som følger med hver modellvariant. 'komp' inneholder en referanse til en liste med alle filene/modulene som inngår ('komp.modules'), og kompilerer og lenker sammen disse filene. Det eneste vedlikeholdet som er nødvendig for at 'komp' skal fungere, er at referansene til eksterne moduler er altomfattende. Kompileringen utføres slik:

```
/mosart/prog/navn-på-ny-modellvariant/>komp [↵]
```

Siden simuleringsmodellen er fordelt på flere filer, er det nødvendig å kunne noen hjelpeprogrammer i Linux som håndterer filer raskt. Her er 'grep', 'find' og 'diff' hensiktsmessige til å henholdsvis finne strenger i filer, finne filer og sammenlikne filer. Monodevelop (eller et tilsvarende moderne redigeringsverktøy) er langt mer effektiv til dette. Se også avsnitt 2.1.

Normalt vil feilmeldingene fra kompileringen være overkommelige å forstå, samt ha referanse til hvor i kildekoden feilen ligger. Imidlertid kan feilmeldingene bli uforståelige hvis feilen ligger i feilaktig bruk av parenteser eller tekstkvalifikatorer, spesielt klammeparenteser ({}). En manglende klammeparentes gjør at resten av kildekoden kan bli helt uten mening for kompilatoren, og det kan være vanskelig å finne hvor feilen da ligger.

Feilsøking 1

Etter at simuleringsmodellen er kompilert uten å ha frembrakt feilmeldinger kan simuleringen startes ved å eksekvere kjørefilen 'job.exe' som følger med hver modellvariant (og hver simulering). Det er mulig å bruke modellvariantens egen katalog som arbeidsområde, eventuelt kan man opprette en egen (test)simulering for dette formålet. Normalt vil en ny modellvariant inneholde feil som kompilatoren ikke finner, men som stopper simuleringen fordi instruksjonen er umulig å gjennomføre, for eksempel åpne en fil som ikke eksisterer. Disse logiske feilene bør lukes ut ved å eksekvere simuleringen på små utvalg. Når modellvarianten virker på små utvalg, kan man gå videre til større utvalg.

I den grad simuleringen stopper vil alle feilmeldinger havne på filen 'sim.errors', meldingen vil ofte referere til en metode og en modul hvor feilen oppstod og hvor man kan starte letingen. I tillegg vil filer av typen 'sim.*' og 'r*_*.prn' være av stor hjelp, all den stund de skrives ut fortløpende. Spesielt vil 'sim.control' vise i hvilken modul simuleringen stoppet (den første modulen etter den siste som er skrevet ut). Arbeidet med å finne logiske feil kan være en tidkrevende oppgave. Ofte må man lage ad hoc utskrifter av variabler fra simuleringen til for eksempel 'sim.errors' for å finne feilen. Kommandoene 'try' og 'catch' kan forenkle dette arbeidet. Har man en sekvens kildekode hvor det oppstår en feil, kan man kapsle den inn med et 'try { ... }', etterfulgt av en kopi av den samme sekvensen nå kapslet inn med et 'catch { ... }'. Simuleringen vil alltid prøve den første versjonen ('try { ... }'), og bare i det tilfellet at det oppstår en feil vil den gjøre det som er angitt av 'catch { ... }'. Her kan man repetere kildekoden, og samtidig legge inn utskrift av verdier på variabler man tror kan avdekke årsaken til feilen:

```
try
{
        Sekvens kildekode med mulig feil
}
catch
{
        Utskrift av variabelverdier som kan belyse feilen
        Kopi av samme sekvens kildekode med mulig feil
}
```

I Monodevelop kan dette gjøres på en enklere måte. Er feilen sjelden, det vil si at man må simulere hele befolkningen for at den skal dukke opp, kan det være mer effektivt å bruke ad hoc skrevet kildekode med try-catch.

Feilsøking 2

Neste fase vil være at modellvarianten fungerer i den forstand at simuleringen ikke stopper opp, men det kan fortsatt være slik at modellen gjør noe annet enn det man hadde tiltenkt, og disse feilene kan være vanskelige å avdekke. I den grad avvikene kan passere som plausible resultater omtaler vi dette som såkalte **skjulte feil**. I denne fasen må resultatene sammenliknes med tidligere simuleringer for å kontrollere at endringene i det minste er rimelige. En god støtte vil være utskrift av livshistorier. Se ellers avsnitt 5.2 og 5.3 for forslag til kontrollrutine og metoder for å unngå skjulte feil.

Noen ganger kan hva som ser ut som én feil, ha sin årsak i flere feil. Når disse underliggende feilene noen steder nøytraliserer hverandre kan det være vanskelig å resonnere seg fram til årsaken til feilen man leter etter.

Maskintekniske feil

Noen ganger bryter simuleringen sammen på variable steder til tross for at man kjører med samme random-seed og forutsetninger, ofte med komplett uforståelige feilmeldinger. Vi refererer til dette som maskintekniske feil, feil som skyldes tilfeldigheter i hvordan serveren og operativsystemet ikke klarer å håndtere simuleringen. En årsak er at MOSART håndterer store mengder data, og det kan oppstå feil i allokeringen av og i opprydningen i internminnet (garbage collector). Det kan hjelpe å eksekvere modellen uten parallellitet og gjenbruk av objekter ('no_of_threads', 'maxdegreeofparallelism' og 'recycle_objects' i 'parameters.con').

Disse feilene er ugreie å finne ut av, og vi må trolig til en viss grad godta at de er der. Noen ganger skyldes de likevel feil og/eller svakheter i kildekoden. Når to tråder (se avsnitt 3.6) konkurrerer om den samme informasjonen, spesielt når de *skriver til* samme variabel, har simuleringen en høyere risiko for å bryte sammen. Dette er i tilfelle en feil i kildekoden, og kan sjekkes ut ved å koble ut parallellitet. Videre er vår erfaring at det påfører opprydningen av internminnet mye stress når data flyttes rundt, se avsnitt 3.6.

Redigering

Skal kildekoden bli lesbar bør en del generelle råd på programmeringsstil følges opp. Den kanskje viktigste faktoren er navnsettingen på variabler, referanser, metoder og klasser. Disse navnene bør være intuitive, språklig korrekte, systematiske, uten forkortelser og likevel om mulig korte. Intuitive variabelnavn gjør kildekoden lesbar og utgjør en helt fundamental del av dokumentasjonen. Systematiske variabelnavn gjør det overkommelig å huske hvordan variabelnavn skal skrives uten å gå tilbake til stedet de først ble referert. I MOSART har vi valgt å skrive variablene på engelsk og i størst mulig grad fullt ut. Sistnevnte øker lesbarheten og forenkler arbeidet med å gjøre navnene systematiske og enkle å huske (presist ortografisk). Det er også en fordel om navnene er grammatisk korrekte versus hva de beskriver, for eksempel en bevisst bruk av verb/substantiv, entall/flertall og fortid/nåtid. Samtidig bør variabelnavn være korte fordi det muliggjør kombinasjoner av navn/typer uten at de blir for lange. Enkelte mye brukte variabel- og typenavn er derfor likevel forkortet, og da helst til 1-3 bokstaver, se tabell 3.1 for en oversikt over noen av de mest vanlige.

Råd: Bruk nok tid på å finne gode variabelnavn før variabelen innarbeides i kildekoden. Dårlige variabelnavn kan fort innarbeide seg på flere hundre steder i en modellvariant på noen uker, og medføre betydelig arbeid med endringen når man tar hensyn til risikoen for feil i prosessen.

Tabell 3.1. Mye brukte forkortelser i simuleringsmodellen

Forkortelse	Navn/forklaring
aepe	*Already earned pension entitlements*
cpp	*Computed pension point*, beregnet uførepoeng
draw_	Trekkemetode/trekkeobjekt
fpp	*Final pension point*, sluttpoengtall
hfu	*Høyeste fullførte utdanning*
id	*Identification*
igu	*Igangværende utdanning*
lfp	*Labour force participation*
lme	*Labour market earnings*
loc_*	*Local*, hvis en klassevariabel leses utenfra ved hjelp av en metode, brukes forstavelsen 'loc_' på den interne representasjonen av variabelen
no_of_*	*Number of*, i betydningen antall
nos_*	*Number of states*, antall tilstander for en statusvariabel
p, q, ...	Referanse til personer, 'p' er normalen, øvrige bokstaver brukes når flere personer skal håndteres sammen
par_*	*Parameters*, objekter med overgangssannsynligheter har 'par_' som forstavelse, slik at objektet som har parameterne og objektet som simulerer et tema kan få ellers like navn
prob	*Probability*
ps	*Pension system*, iterasjonsvariabel brukt når man itererer over pensjonssystemer
reg	*Registration status*
sim	*Simulation*
t_*	*Type*, normalt vil klassenavn ha 't_' som forstavelse, slik at klasse og referanse til objekt kan få ellers like navn
wd	*Work distributor*
wl	*Work list*

Navn på filer, klasser og referanser bør i størst mulig grad være sammenfallende. Da vil ulike deler av programmet konkurrere om de samme navnene, slik at eksakt samme navn ikke er mulig. Systematisk navnelikhet kan likevel oppnås gjennom systematisk bruk av små/store bokstaver, forstavelser og/eller endelser. Spesielt vil klassenavn i MOSART gjennomgående ha forstavelsen 't_'.

I tillegg til intuitive navn på variabler, referanser, metoder og klasser, bør kildekoden suppleres med noe tekstlig dokumentasjon. Lengre kommentarer bør samles forut for en metode eller en klasse, da omfattende kommentarer gjør det vanskelig å få kildekoden inn på en skjermside. Kommentarer

inne i metoden bør enten være overskrifter[17] eller gå på forhold som er vanskelige og/eller kontraintuitive. Kommentarene bør heller ikke være for lange og/eller detaljerte, da kommentarene også skal oppdateres når kildekoden endres og dette kun avhenger av den enkelte utviklers selvdisiplin.

C# ignorerer all hvit støy, det vil si (blant annet) blanke tegn, tabulatorer og linjeskift som kommer ***mellom*** variabelnavn. Det gir stor fleksibilitet på layout, som er en annen viktig faktor for å gjøre kildekoden mest mulig lesbar. Dette går blant annet på innrykk og linjedeling. Under følger en beskrivelse av layouten i MOSART, og det er en stor fordel om denne også gjennomføres konsekvent, som for navnsetting, da et konsekvent layout er enklere å forholde seg til.

Innholdet i en underliggende blokk, det vil si kildekode mellom to klammeparenteser ({ ... }), bør ha et regelmessig innrykk for hvert nytt blokknivå for å gjøre kildekoden mer lettlest. Dette innrykket bør være fast, og stort nok til at det er synlig, men samtidig minst mulig for å unngå at moduler med mange innrykk havner helt på høyre side av skjermen. I MOSART bruker vi fire og bare fire blanke felter, da dette i dag er standard. Editorene mange bruker på Linux ('nedit', 'emacs'), har fast bredde på alle bokstaver og symboler, slik at fire blanke gir et fast visuelt inntrykk. I Word (med *vanlige* fonter) vil blanke få en variabel bredde, slik at vi i eksemplene her har vi brukt tabulatorer. Vi har som hovedregel valgt å plassere klammeparantesene ('{','}'), på egne linjer, spesielt der linjedeling er påkrevet.

I noen tilfeller kan det bli for mange innrykk og hele kildekoden havner som en smal tekst på høyre side av skjermen. Dette kan (og i mange tilfeller bør) motvirkes av å dele opp innholdet i flere lag av metodekall (deklarasjonen av hvert nivå av metoder kan jo starte (nesten) helt til venstre). Det er også noen andre triks som er tilgjengelige. Standard protokoll er at hver gang man skriver en løkke ('for', 'foreach') blir kildekoden rykket inn. I de tilfellene *en gruppe* av slike løkker er fullstendig sammenkoblet[18], det vil si bare har en felles blokk til slutt, kan man velge å ha bare ett innrykk. Noen metoder og løkker innledes med å avvise irrelevante personer, og standard metode er å kjøre en logisk test etterfulgt av en blokk som dekker hele metoden (if (*relevant*) {.....}). Alternativ kan man ***avbryte*** metoden eller løkka for de som er irrelevante for arbeidsoppgaven, og skrive resten av kildekoden uten det ekstra innrykket. Med metoder bruker man da 'return', med løkker bruker man 'continue' hvis man skal fortsette til neste person.

I C# kan hele kildekoden om man vil samles på en linje, som vil gjøre alle editorer for menneskelig bruk helt verdiløse. Vi har valgt er å ha en maksimal linjelengde på 115 tegn, som gjør at hele linja uten problemer kommer opp på skjerm i et format som er lesbart med normalt syn, for de fleste editorene (nedit, emacs 9pkt og monodevelop). Det er en fordel å gjøre dette helt systematisk, da man slipper å lure på om det står noe utenfor skjermbildet som man ikke ser. Maksimal linjelengde er noe som må vedlikeholdes av brukere, og det kan være en irritasjon fordi det å brekke linjer på en god måte kan være litt arbeidskrevende. Ved suboptimal linjedeling, for eksempel at man brekker linje på siste variabel-skille før 115 tegn, kan det som skjer i kildekoden både bli vanskelig å lese, og man kan bli forledet til å lese at det skjer noe annet enn det som faktisk står der (linjedelingen påvirker hvordan man leser kildekoden). Det vi har siktet mot i MOSART er å dele linjer ved hovedinndelingene i det logiske innholdet, for eksempel:

- I en metodedeklarasjon/anvendelse bør man først dele mellom metodetypen/navnet og metodeparameterne, ***før*** man eventuelt begynner å spre metodeparameterne utover flere linjer.

[17] Velger man gode metodenavn, vil overskriften ofte være det samme som navnet på metoden som følger. Imidlertid vil overskrifter med noen ekstra kommentarlinjer gjøre det lettere å se hvor metoden starter og slutter.
[18] For eksempel tre løkker over reg.status, kjønn og alder.

- I logiske tester bør man først dele mellom de ulike elementene i den logiske testen (det som er skilt av '&&' og '||'), deretter det som eventuelt står på hver sin side av den logiske operatoren ('==', '>', ...), og til slutt hver enkelt variabel. Presise innrykk er avgjørende for å gjøre kildekoden lesbar, spesielt at nivåer i logikken som hører sammen også får samme innrykk.

Ved å legge til en blank på slutten av hver linje, det gjelder på kildekoden, men også inputfilene, og noen ganger mellom variabelnavn (hvor det er frivillig), vil et enkelt søkeverktøy som 'grep' bli mer effektiv. Blant annet kan man søke opp et variabelnavn ved 'grep " *søkestreng* " *filnavn*', og samtidig slippe å få med alle andre variabelnavn som har '*søkestreng*' som en del av navnet sitt (og det er mange sammensatte variabelnavn i MOSART).

Ved overgang mellom metoder, klasser og andre relaterte instruksjoner, bør man legge inne ekstra linjeskift (luft), og eventuelt en ekstra overskrift i form av tre kommentarlinjer:

```
//!***************;
//!* Overskrift. *;
//!***************;
```

Regnehastighet og stabilitet
Generelt bør man i utviklingen av nye modellvarianter legge størst vekt på lav risiko for skjulte feil, samt at kildekoden skal være lettlest og lett å videreutvikle. Imidlertid er det også ønskelig å ta andre hensyn, blant annet til kjøretider og forbruk av regnekraft. Noen av disse erfaringene er utdypet i avsnitt 3.6, og er avgjørende for at MOSART kan simulere hele befolkningen på en gang.

3.5. Hovedstruktur i simuleringsmodellen

Simuleringsmodellen leses lettest ved å starte med hovedprogrammet ('model.cs') og kompilatoren ('komp', 'komp.modules'). Sistnevnte gir en oversikt over alle filene som inngår i simuleringsmodellen. Avsnittet om simuleringsmodellen i Vedlegg D gir også en oversikt over filene som inngår i simuleringsmodellen, med noe forklaring til hva de gjør. Vedlegg G-K omtaler noen av modulene i noe mer detalj. Filnavn og navn på klasser er eventuelt angitt i parentes i overskriftene.

Hovedprogram ('model.cs', 'class model')
Hovedprogrammet ligger på filen 'model.cs', omfatter kun metoden 'Main' (se avsnitt 3.2), og er delt opp i fire deler. Den første delen initialiserer parametere, de tre siste delene er samlet under overskriften 'Execute simulation'.

Første del ('Initialize parameters') deklarerer alle referanser og variabler som inngår i metoden 'Main' og initialiserer alle parametere som inngår i simuleringen. Dette inkluderer infrastruktur, innlesing av overgangssannsynligheter og andre parametere, oppsett av objekter som håndterer simuleringen og objekter som håndterer dokumentasjonen av simuleringen.

Andre del ('Initial population') leser inn utgangspopulasjonen og tilrettelegger manglende kjennetegn og dokumenterer utgangspopulasjonen i startåret for simuleringen.

Tredje del ('Simulation loop') er den egentlige simuleringen, og er i hovedsak en løkke ('while (...) { ... }') som går over hvert av de årene som skal simuleres, og som for hvert år kaller de ulike modulene som simulerer kjennetegnene. Simulering her vil også inkludere oppdatering av kjennetegn, tilordning og justering av sannsynligheter, utskrift av modellpopulasjonen og tabelluttak.

Fjerde og siste del ('Close simulation') er svært kort og avslutter simuleringen.

Felles konstanter og hjelpemetoder ('global.cs', 'class t_global', 'g')

Felles variabler og mindre hjelpemetoder er samlet i en egen klasse kalt det **globale objektet**. Dette omfatter relasjoner til operativsystemet, egendokumentasjon, metoder for innlesing av parametere fra brukergrensesnittet, utvalgsparametere, tidsangivelser, parametere for trekkemetoder, egendefinerte funksjoner og konstanter for personkjennetegn. Mange av disse egenskapene brukes av flere andre moduler, og samlingen i 'global.cs' gjør at vedlikeholdet kan skje på ett sted og gjøres tilgjengelig med en referanse ('t_global g'). Variable variabler er forsøkt holdt utenfor det globale objektet, dog med unntak for tidsangivelse og random-seed. Det globale objektet er noe av det første som opprettes i simuleringen, og klassen inneholder derfor også metoder for å sjekke om simuleringen kan/bør eksekveres, relasjoner til operativsystemet og for dokumentasjon av simuleringen. Vedlegg K gir en oversikt og omtale av variablene og hjelpemetodene i det globale objektet.

En del av innholdet i 'global.cs' burde vært flyttet ut i de modulene som eier kjennetegnet, det ville gjort 'global.cs' litt mer oversiktlig, men også forenklet vedlikeholdet av innholdet når det skjer lokalt.

Felles klasser

Vi har tilrettelagt en del felles (hjelpe)klasser som skal bidra til å forenkle kildekoden og utviklingen av MOSART, se Vedlegg J for en nærmere omtale av noen av disse. Dette omfatter trekkemetodene ('draw_*.cs'), som også er omtalt i kapittel 4. Klassen for personer omtales nedenfor. Organiseringen av populasjonen i lister skjer gjennom en overbygning ('population_list.cs', 'class t_population') og en klasse som håndterer hver kohort ('cohort_list.cs', 'class t_cohort_list'). Videre er det noen hjelpeklasser som understøtter spesielle sider ved simuleringen ('work_distributor.cs', 'binary_tree.cs', 'enumerators.cs' og 'iteration_parameters.cs').

Med unntak for klassen for personer vil innholdet i disse felles klassene normalt være likt fra modellvariant til modellvariant.

Klasse for personer ('person.cs', 'class person')

Det er en egen klasse som definerer hvilke kjennetegn en person skal ha. Dette er en klasse som ofte endrer seg ved utvikling av nye modellvarianter, og det er spesielt to hensyn å ta her.

For det første vil klassen for personer være den avgjørende faktoren for hvor mye internminne simuleringen trenger, og til en viss grad hvor stor belastning simuleringen legger på opprydningen av internminnet (garbage collector). Det er derfor viktig å avgrense modellen til de kjennetegnene man trenger, og eventuelt også avgrense variabeltypen til riktig omfang (byte, short, int, double). Det kan redusere belastningen på internminnet hvis man begrenser *antall* matriser (en vektor over antall pensjonssystemer med et objekt med alle pensjonsvariabler er langt å foretrekke framfor en vektor for hver pensjonsvariabel).

For det andre bør man være systematisk med å trekke nye personkjennetegn gjennom hele simuleringsmodellen, ikke bare i 'person.cs' og den aktuelle modulen som simulerer det nye kjennetegnet. Tre viktige stasjoner må nevnes:

- Personobjektene resirkuleres, og **alle kjennetegn må nullstilles når personen opphører**. Se for eksempel metoden 'clear()' i 'class person'.
- Kjennetegnet må legges til i modellpopulasjonen, dette er viktig for blant annet enklere feilsøk, se Vedlegg H.

- Kjennetegnet må legges inn tabelluttaket, se Vedlegg G.

Overgangssannsynligheter og andre parametere
Håndteringen av overgangssannsynlighetene, andre parametere og regelverk for pensjon/skatt er i størst mulig grad forsøkt lagt til egne klasser som leser inn og håndterer denne typen informasjon. Disse filene har navnestruktur 'read_*.cs', normalt med en navnebror som simulerer tilsvarende kjennetegn (sim_*.cs). Normalt vil innlesing av filer, tabelloppslag og beregning av overgangssannsynligheter kreve mye kildekode, men samtidig være relativt trivielle metoder. Ved å skille ut håndteringen av overgangssannsynligheter forkortes/forenkles modulene som simulerer kjennetegnene vesentlig, moduler som i utgangspunktet er kompliserte. I tillegg blir overgangssannsynlighetene og regelverket tilgjengelige for alle modulene som simulerer kjennetegn uten å skape for mange kryssavhengigheter. I en del tilfeller har vi også latt oppdateringen av kjennetegnene være i objektet som håndterer overgangssannsynlighetene. Dette bør gå klart fram av spesifikasjonen av metodene (kjennetegn som endres bør inngå som parametere i metodene). Derimot er anvendelsen av trekkemetodene ('t_draw_*') systematisk holdt ute av innlesingsmodulene for overgangssannsynligheter.

Objektene som håndterer overgangssannsynligheter er avledet av en grunnklasse for innlesing av filer ('read_base.cs'), se Vedlegg I for detaljer om grunnklassen og hvordan innlesingen og oppslagene i tabellene bør utføres. Spesielt vil klassen ha metoder som henter inn filnavn fra styrefilen 'input.con' og metoder som håndterer disse filene. Det er lagt vekt på at disse metodene skal stoppe simuleringen hvis det er irregulariteter i filene.

Simulering av kjennetegn
Simuleringen av hvert tema er organisert i egne filer/klasser med navnestruktur 'sim_*tema*.cs' og 'class t_*tema*'. Ulike tema er i denne versjonen migrasjon, dødelighet, fruktbarhet, hushold, region/flytting, utdanning, pensjon, pensjonering/trygd, arbeidstilbud, arbeidsinntekter og inntekt/skatt. Vi går ikke gjennom de ulike temaene her.

Generelt vil disse modulene inneholde trekkeobjekter, enkelte parametere som styrer simuleringen og en eller flere metoder som går gjennom hele befolkningen og simulerer et kjennetegn. Simulering vil her også omfatte tilordning og justering av overgangssannsynligheter, samt beregning av pensjon og skatt.

Oppdatering ('update.cs', 'class t_update')
Ved starten av hvert år som skal simuleres må en rekke hjelpevariabler (indikatorer) på personnivå nullstilles og avdøde personer uten relasjoner til personer som er i live fjernes fra populasjonslistene. Dette gjøres i en liten hjelpeklasse med navn 'class t_update'.

Innlesing av utgangspopulasjonen
Det foreligger tre typer utgangspopulasjoner, en med faktiske opplysninger fra et nyere startår, en simulert med 1960 som startår og en med typehusholdninger. Disse har navnestruktur 'read_*navn-på-populasjon*.cs', se Vedlegg L.

Modellpopulasjon
Simuleringen kan på slutten av hvert år skrive ut modellpopulasjonen til en tekstfil med en linje per person og år, for utvalgte personer, år og kjennetegn. Filen har navn 'model_population_file.cs', se Vedlegg H.

Tabelluttak

Simuleringen kan på slutten av hvert år skrive ut tabeller med aggregerte tall. Disse filene har navnestruktur 'table*.cs', se Vedlegg G. Fra 'komp' kalles også shellscriptet 'generate_output_variables', som oppdaterer filen 'output.variables' som angir hvilke variabler som inngår i tabelluttaket.

Modellpopulasjonen og tabelluttaket har det til felles at de henter ut opplysninger av personobjektet og tilrettelegger tallverdiene for videre behandling på et relativt generelt nivå. Det er en mulighet for å redusere kildekoden og antallet punkter man må gjennom for å legge inn nye variabler, hvis man finner en effektiv måte å håndtere dette på under en felles klasse.

Estimering av parametere

Mange av overgangssannsynlighetene (og andre parametere) i MOSART er estimert, beregnet eller hentet ut som tabeller fra MOSART selv. Dette har tidligere enten vært utført i separate modellversjoner eller inne i simuleringsrutinene. Det første har ofte gjort det vanskelig å reprodusere resultatene på nye versjoner, det siste har komplisert kildekoden for simuleringen til de grader at det har lammet videreutviklingen. I tillegg har det vært mange manuelle og kompliserte steg i etterkant, ofte i Excel.

For å effektivisere dette arbeidet har vi begynt å legge dette systematisk inn i egne moduler som ligger på utsiden av modellen, med navn som 'estimate_*tema*.cs'. Målet er at disse skal skrive ut brukerklare inputfiler, med egendokumentasjon og parametere. I første omgang er dette avgrenset til overgangssannsynligheter som er beregnet ved de observerte sannsynlighetene direkte (tabelluttak), som fortsatt er en god del av modellen. Målet bør være å utvide dette med mer avanserte metoder, for eksempel logistisk regresjon.

NB! Husk da å koble ut estimeringsrutinene etter brukt ('include_estimation' i 'estimate.con').

Befolkningsframskrivinger

MOSART simuleres slik at de demografiske forutsetningene skal være mest mulig like med de offisielle befolkningsframskrivingene fra Statistisk sentralbyrå. Dette sikres ved to moduler ('befinn.cs' og 'befreg.cs'), som er en kopi av aggregeringsrollen i befolkningsframskrivingene, inngår i MOSART. Disse to modulene beregner alle befolkningstall og bevegelser i folkemengden basert på forutsetningene omkring dødelighet, migrasjon og fruktbarhet. Fordelen er at vi fritt kan kombinere alternativer av forutsetninger, samt modifisere de demografiske forutsetningene. Det er også mulig å simulere befolkningsutviklingen med andre forutsetninger, blant ved leveranser til EU.

3.6. Effektiv bruk av regnekraft

For å kunne simulere MOSART med hele befolkningen er det absolutt nødvendig å bruke teknikker som utnytter regnekraften på riktig måte. Nedenfor følger noen av de erfaringene vi har gjort. Merk at noen av disse teknikkene kan øke kompleksiteten i kildekoden og øke faren for skjulte feil, begge deler er i seg selv uønsket.

Analyseverktøy

Ved utprøving av nye modellvarianter er det avgjørende å kunne teste ut den faktiske virkningen av hva man gjør, her kommer 'sim.control', 'sim.garbage_collector' og 'sim.memory' inn som nødvendige analyseverktøy. Disse filene rapporterer kjøretider, forbruk av internminne og opprydning av minnet. Ved å eksekvere modellen før og etter modifikasjonene (gjerne med parameterstyring, slik at man kan skifte mellom å kjøre med og uten endringene), og med ellers like forutsetninger, kan man få målt hva som skjer og få ledetråder til hva som øker kjøretidene.

C# har mange fikse metoder som kan forenkle kildekoden, og man bør bruke disse når de også er regneteknisk effektive. Enkelte av disse metodene er utviklet for andre rammebetingelser (små mengder med data), og kan fullstendig lamme en modell av typen MOSART. Før man tar i bruk slik metoder bør man sjekke ut effekten på kjøretider og stabilitet.

Parallellitet

Utviklingen i regnehastigheten i datamaskiner stod i lenge stille, men for likevel å forbedre den opplevde regnehastigheten for tunge arbeidsoppgaver regner moderne datamaskiner parallelt på de oppgavene som skal utføres. Serveren har flere regneenheter som kan jobbe uavhengig.

En enkelt applikasjon kan også utføre sine arbeidsoppgaver parallelt. Normalt eksekveres applikasjonen med en **hovedtråd** (main thread) hvor alt utføres i en fast rekkefølge, og det er bare en ting som skjer om gangen. Applikasjonen kan også eksekvere nye jobber, og hver slik ny jobb kalles en **tråd** (thread). Disse trådene vil ha tilgang til de samme data som hovedtråden, og alle trådene jobber samtidig (parallelt) på disse dataene. Vi har kommet et stykke i å implementere disse metodene i MOSART, gjennom **multithreading** og **Parallel**. Generelt reduserer dette kjøretidene med en faktor på 15[19] for simuleringen som helhet, og er dermed avgjørende for at modellen skal være praktisk operativ med hele befolkningen som modellpopulasjon.

I bruken av parallellitet i MOSART stiller vi som krav at metoden i seg selv ikke skal påvirke simuleringsresultatet. Noen ganger kan parallellitet påvirke simuleringen uten at dette er en egentlig feil, for eksempel gjennom å påvirke i hvilken rekkefølge personene blir simulert (se nedenfor). Vi har likevel behov for å kunne ***reprodusere*** alle resultatene våre, og dermed må vi også luke ut feil av denne typen. Parallellitet gjør at rekkefølgen av de ulike arbeidsoppgavene kan bli endret (genuin stokastikk i serveren gjør at en jobb kan bli ferdig tidligere i en simulering enn i en annen). Det igjen betyr at enhver rekkefølge av arbeidsoppgavene som eksekveres parallelt må være uten betydning for resultatet.

Første trinn i parallellitet er at man faktisk klarer å stykke opp simuleringen i biter som kan håndteres uavhengig. C# i seg selv gir ingen advarsler om ulike tråder jobber på samme data, annet enn at det øker risikoen for at simuleringen bryter sammen med referanse til maskintekniske feil. MOSART ligger vel til rette for en segmentering av simuleringen, fordi modellpopulasjonen er organisert i lister etter kjønn, ettårig alder og reg.status. Hver tråd kan da fortløpende tildeles en tråd for å simulere hva som skjer med denne kohorten. Det er likevel overraskende hvilke koblinger som finnes mellom ulike deler av simuleringen, og det har vært og er en stor jobb å holde orden på dette. Enkelte deler av simuleringen lar seg multithreade enkelt og med stor effekt, for eksempel beregning av skatter. Andre deler av simuleringen har vært nærmest umulig å få til korrekt multithreading og med minimal effekt på kjøretidene, husholdningsrelasjoner er et godt eksempel på det siste.

En metode for å sjekke at trådene ikke jobber på samme data er å eksekvere ellers identiske simuleringer med og uten parallellitet. Resultatene skal da bli identiske uavhengig av antall tråder som er benyttet.

En viktig kilde til overlapp er at trådene utilsiktet jobber på samme data, felles data her kan typisk være tellevariabler og trekkemetoder. Det er viktig at felles data får en segmentering som fanger opp alle mulige fordelinger og rekkefølger av fordeling av arbeidsoppgaver på trådene. Har man

[19] Anslått fra to enkle kjøringer med no_of_threads=40 og =1. Dette anslaget kan bli påvirket av innholdet i simuleringen og annen trafikk på serveren.

brukt kjønn og ettårig alder som grunnlag for segmenteringen av arbeidsoppgaver må tellevariabler og trekkemetoder ha minst samme inndeling.

En annen viktig kilde til overlapp er relasjoner mellom personer, og spesielt endringer i disse relasjonene, for eksempel pardannelse. Der man skal regne på en gruppe av personer uten å endre relasjonene, for eksempel ved skattlegging, så anbefales det at gruppen (husholdet) er utgangspunkt for beregningen. Rent praktisk går man gjennom alle personer, og når man finner personen som representerer husholdet ('household_head', med litt modifikasjoner), så går man gjennom alle personene i husholdet og beregner skatten.

*Der man skal **endre** relasjoner mellom personer anbefales det at man ikke bruker parallellitet.* Det er veldig vanskelig å gjennomskue alle koblinger mellom de personene som inngår i endringen, og det kan ofte gi feil som både er ekstremt subtile og krever at man kjører hele befolkningen for at skal dukke opp. Derimot er det mulig å multithreade forberedelser til slike endringer, for eksempel beregne/tilordne sannsynligheter, og i noen tilfeller også simulere kjennetegnet, men ta endringene i relasjonene i en egen modul i etterkant uten parallellitet.

En subtil feilkilde er at når man summerer desimaltall i en datamaskin, så er ikke rekkefølgen av summeringene likegyldige, fordi datamaskinen må avrunde fortløpende. Hvis parallellitet kan påvirke rekkefølgen av summeringene kan dette gi ørsmå utslag i summeringene, men når dette til slutt påvirker én stokastisk trekning bryter likheten sammen i resten av simuleringen. Det er ingen eksepsjonelle hendelser i MOSART, så dette blir til simuleringsstøy rundt den samme forventede banen. Det er likevel såpass problematisk at man bør unngå dette. *Skal man summere opp tellevariabler etter parallellitet må dette gjøres til slutt i en deterministisk rekkefølge.*

En annen (subtil) feilkilde er at maskingenerert simuleringsstøy (generert av rekkefølgen på trådene) kan påvirke sorteringer. I de tilfeller to objekter vurderes som like med hensyn til sorteringsvariablene vil rekkefølgen de har i den opprinnelige listen kunne påvirke hvor de havner i den sorterte listen (FIFO). Dette kan motvirkes av at man unngår at maskinstøy får påvirke rekkefølgen i den opprinnelige lista, og ved å forsøke å gjøre alle sorteringer uttømmende. Er det personobjekter man sorterer, kan man for eksempel til slutt, ved ellers likhet, sortere på ID-nummer (som er unikt).

Skal man legge inn parallellitet i nye deler av simuleringen anbefales det å følge en lignende modul, og søkeord vil være *Thread*, *thread* og *Parallel*.

For å teste ut parallelliteten er det avgjørende at man kan eksekvere simuleringen uten parallellitet og med samme random-seed. Resultatene skal da bli identiske, og det er nødvendig å gjøre slike tester fra tid til annen. Styrefilen 'parameters.con' har parametere for å styre dette.

Allokering av internminne

Etter hvert som simuleringen skrider fram, bruker den internminne (RAM) til å lagre informasjonen. Noe av denne informasjonen er bare relevant for et begrenset tidsrom, og for at simuleringen ikke skal bruke opp hele internminnet er det viktig at internminne som ikke lenger brukes blir frigjort. Opprydningen av internminnet (garbage collector) utgjør en vesentlig del av kjøretiden og er samtidig en årsak til at simuleringen bryter sammen av maskintekniske årsaker. Virkingen på kjøretiden kommer ved at simuleringen stanser opp når internminnet skal ryddes, ustabiliteten kommer trolig gjennom at det kan bli enorme mengder data som skal flyttes innenfor et begrenset område.

Data med kort levetid (variabler innad i en blokk) og data med lang levetid (hele simuleringen) er ikke et problem. Det er data som eksisterer lenge som hovedproblemet, og noen av disse kildene omtales under.

To vanskelige datatyper er *List* og *matriser* (array*)*. Disse elementene er helt nødvendige for å sikre en effektiv utvikling av kildekoden, men bør brukes med varsomhet. Hver gang man bruker en List eller matrise allokeres minne som utgjør en utfordring for opprydning av internminnet. Kan man resirkulere disse elementene vil det bli allokert langt mindre internminne, og dette vil også være formålstjenlig selv når disse elementene må nullstilles. Det vil også være klart å foretrekke å bruke én vektor av objekter, med mange variabler per objekt, framfor mange vektorer, gitt ved én vektor for hver variabel. I personkjennetegnet er det for eksempel en vektor over pensjonssystem, med et objekt som inneholder alle variablene som inngår i skatteberegningen.

Resirkulering av objekter er en risikabel metode - det vil for eksempel si at når en person går ut av simuleringen blir personobjektet nullstilt og stilt til rådighet når man senere skal opprette en ny person. Faren ligger i at når man legger til nye personkjennetegn kan man glemme å vedlikeholde metodene som nullstiller objektet. Virkningen på kjøretider og stabilitet tilsier likevel at resirkulering bør benyttes. Tilsvarende som for parallellitet er det avgjørende at man tester virkningen av resirkuleringen, og dette styres også gjennom parameteren 'recycle_objects' i 'parameters.con'. Ellers identiske simuleringer, men med og uten resirkuleringer, skal gi identiske resultater. Skal man legge inn resirkulering av en ny type objekter er det enklest å følge gangen i det som er gjort, og et søkeord er da *freelist*.

Innlesing og utskrift

Noe C# bruker mye tid og til dels internminne på er innlesing av store mengder data fra eksterne filer, og tilsvarende ved utskrift av store mengder data til eksterne filer. Det er spesielt overgangene tekst-til-tall og tall-til-tekst som ser ut til å skape problemer. Når man oppretter en ny tekst, eller legger en bit tekst til en annen tekst, utløses allokering av minne. Innebygde metoder i C# ser ut til å være lite effektive på dette punktet.

Innlesingen av parametere for overgangssannsynligheter går relativt uproblematisk med hensyn til kjøretider. Det er likevel etablert en egen klasse for denne innlesingen ('t_par'), men det er for å sikre både en enkel og framfor alt robust håndtering av innholdet i filene.

I innlesingen av utgangspopulasjonen og parametere fra befolkningsframskrivingene har vi lagt inn egne metoder som håndterer innlesingen på en effektiv måte. I utskriften av resultater fra befolkningsframskrivingene benytter vi også egne metoder for utskrift.

Vedlikehold av lister

I C# vil List gi en effektiv og fleksibel håndtering av variable mengder data. I MOSART blir likevel disse listene så store og/eller mange at en viss varsomhet er nødvendig. Dette gjelder spesielt håndteringen av modellpopulasjonen. Det som er spesielt krevende er å legge til eller fjerne en person på en plass tidlig i lista, slike operasjoner utløser en omplassering i internminnet av referansene til alle de etterfølgende personene.

Når man skal flytte personer inn i eller ut av en kohortliste kan denne operasjonen noen ganger settes på vent til begivenheten er simulert for alle personene i kohorten (inn- og utvandring, død, fødsler). Man kan da i etterkant sortere kohortlista slik at alle personer som skal flyttes kommer sist i lista, og fjerne de som skal flyttes enten som gruppe eller suksessivt til enhver tid siste person i lista. Samtidig kan da gjerne de øvrige personene sorteres i en tilfeldig rekkefølge, da det lett oppstår avhengigheter mellom personer som ligger tett på hverandre i lista. En slik sortering er også

krevende, men da blir det én slik operasjon for hele gruppen av personer som skal flyttes framfor én omfattende operasjon for hver person som skal flyttes.

Mellomlagring av resultater

I en del tilfeller skal resultatet av en beregning brukes flere ganger, og det er da en avveining om man skal bruke internminne til å lagre resultatet eller om man skal regne ut resultatet på nytt hver gang man skal bruke det. I tillegg til virkningen på internminnet og kjøretiden kan en slik mellomlagring utløse feil fordi man ikke har oppdatert variabelen korrekt.

Alder kan tjene som et eksempel. Hver person kan få beregnet sin alder som differansen til gjeldende år og fødselsåret til det årskullet man tilhører. Alternativt kan hver person få tildelt et kjennetegn for alder og så må simuleringen oppdatere denne variabelen ved starten av hvert år for samtlige personer. Med en høy pris på internminne er det nytt oppslag hver gang som er beste metode. Slik regneressursene er nå har vi gått langt i retning av å mellomlagre alle beregninger som kan tenkes å brukes flere ganger.

Slike mellomlagringer er ikke bare relevante på personnivå. Et annet eksempel er når en sannsynlighet er en funksjon av alder (annengradspolynom for eksempel), da kan det være bedre å opprette en vektor med det ferdige resultatet for hver alder.

I beregning av sannsynligheter gjør man ofte oppslag i matriser. Er det gjennomgående forklaringsvariabler i disse matrisene, for eksempel kjønn, kan det være formålstjenlig å opprette en overbygning med ett objekt for hver krysskombinasjon av de gjennomgående variablene, hvor hvert objekt har reduserte matriser uten de gjennomgående forklaringsvariablene. Hvert objekt vil ha en kopi av beregningsrutinen, og kan slå opp i matrisene uten å forholde seg til de gjennomgående variablene. Selv om det bare er kjønn som er gjennomgående kan dette gi klare utslag på kjøretidene.

4. Trekkemetoder

Simuleringen i MOSART bygger på en forutsetning om at hver person har gitte (overgangs-) sannsynligheter for å oppleve bestemte begivenheter hvert år. Modellen vil deretter trekke om hver enkelt person opplever disse begivenhetene, slik at sannsynligheten for at begivenheten inntreffer er lik overgangssannsynligheten gitt at man velger en vilkårlig startverdi for rekken av såkalte tilfeldige tall. Med tilfeldige trekninger for hver enkelt person vil det oppstå en variasjon i antall begivenheter som blir realisert i hver enkelt simulering, denne ekstra usikkerheten blir ofte omtalt som simuleringsstøy. Denne simuleringsstøyen er for nesten alle formål med MOSART uønsket. Det finnes mange metoder for å redusere den relative betydningen av simuleringsstøyen, den enkleste er å øke størrelsen på det utvalget som simuleres. Men selv når hele befolkningen simuleres kan denne simuleringsstøyen være til sjenanse, spesielt for anslag på år-til-år endringer og på forskjeller mellom to simuleringer med små utslag i de underliggende forutsetningene.

Det finnes mange trekkemetoder som ivaretar at hver enkelt person skal ha en gitt personlig sannsynlighet for å oppleve en begivenhet (gitt en tilfeldig startverdi for random-generatoren), samtidig som variansen på totalt antall begivenheter blir eliminert eller sterkt redusert. Hovedpoenget er å introdusere negative korrelasjoner mellom trekningene betinget av de realiserte hendelsene, men på en slik måte at de individuelle a priori sannsynlighetene ikke blir endret. Vi går gjennom noen ulike måter vi kan utføre slike trekninger i MOSART i dette kapitlet, med vekt på binomiske begivenheter.

Videre vil noen trekninger bestå i å finne en tilfeldig person med gitte kjennetegn, og avsnitt 4.8 drøfter noen metoder for å gjøre dette slik at personer i samme liste fortsatt kommer i en tilfeldig rekkefølge. Til slutt presenteres noen metoder for å justere overgangssannsynlighetene slik at vi treffer eksogene skranker for totalt antall begivenheter. Spesielt vil en kombinasjon av justerte sannsynligheter og en egnet variansreduserende metode gjøre at modellen treffer sine eksogene skranker tilnærmet presist selv med små utvalg.

4.1. Forventet antall begivenheter

Hver enkelt person *i* har et kjennetegn *x* som har denne strukturen:

(4.1.1) $$x_i = \begin{cases} 1 \; med \; sannsynlighet \; p_i \\ 0 \; ellers \end{cases}$$

Totalt antall begivenheter *X* i populasjonen er gitt ved:

(4.1.2) $$X = \sum_i X_i$$

Hvis trekkemetoden er korrekt, det vil si at hver person har sannsynligheten p_i for at begivenheten inntreffer, er forventet antall begivenheter gitt ved:

(4.1.3) $$E(X) = E(\sum_i x_i) = \sum_i E(x_i) = \sum_i p_i$$

Variansen på totalt antall begivenheter er gitt ved:

(4.1.4) $$Var(X) = Var(\sum_i x_i) = \sum_i Var(x_i) + \sum_i \sum_i Cov(x_i, x_i) \\ = \sum_i p_i\,(1 - p_i) + \sum_i \sum_i Cov(x_i, x_i)$$

Det er variansen i (4.1.4) vi kaller simuleringsstøy, og som vi ønsker å redusere mest mulig. Hvis vi tar sannsynlighetene p_i for gitt er det kovariansene vi kan gjøre noe med. En forutsetning for at en trekkemetode skal være variansreduserende blir da at trekkemetoden i seg selv må gi negativ kovarians mellom begivenheter for ulike personer. Det er da viktig at trekkemetoden ikke skaper negativ kovarians mellom to mulige begivenheter man ønsker å se i sammenheng. Dette kan spesifiseres som:

(4.1.5) $$Cov(x_i, x_i | i \text{ og } j \text{ er relaterte}) = 0$$

Relaterte begivenheter hvor vi ønsker denne typen uavhengighet er trekninger av en type begivenheter for samme person, for eksempel når denne begivenheten kan gjenta seg neste år. Hvis trekkemetoden i seg selv introduserer avhengigheter mellom *for eksempel* arbeidstilbudet fra en person ved ulike aldre, endres fordelingen over mulige livsløp. Tilsvarende må arbeidstilbudet fra ulike personer i samme hushold trekkes uavhengig for at det samlede arbeidstilbudet i husholdningene skal få riktig fordeling.

4.2. Ordinær trekkemetode for binomiske sannsynligheter

Den vanlige måten å utføre trekninger på er å la simuleringen skape tilfeldige tall med uniform fordeling i intervallet (0,1), for deretter å sammenlikne det tilfeldige tallet med overgangs-sannsynligheten. Er det tilfeldige tallet mindre enn sannsynligheten inntreffer begivenheten:

(4.2.1) $$x_i = \begin{cases} 1\ hvis\ z_{i<}\ p_i, \text{ og } z \text{ er en stokastisk variabel } z \sim U(0,1) \\ 0\ ellers \end{cases}$$

eller skrevet i C# i simuleringsmodellen[20]:

```
if (g.Uniform(... , 0, 1) < p)
        "handling hvis begivenheten inntreffer";
```

Hvor *u* er random-seed og *p* er sannsynligheten. Random-seed 'u' må tilordnes én startverdi før man tar i bruk 'u', og velger man denne startverdien vilkårlig vil trekkemetoden tilfredsstille kravene i avsnitt 4.1. Se avsnitt 2.4 og 4.5 for detaljer om random-seed. Trekkemetoden vil imidlertid tilføre variasjon i realisert antall begivenheter *X* gitt ved (gitt at p_i-ene er eksogene og trekningene uavhengige av hverandre):

(4.2.2) $$Var(X | \text{uavhengige trekninger}) = \sum_i p_i\,(1 - p_i)$$

[20] På grunn av mulige avrundingsfeil bør dette skrives som g.Uniform(... , 0, 0.9999999), da dette sikrer at begivenheten inntreffer hvis p=1, og utelukkes hvis p=0, samtidig som den er tilstrekkelig forventningsrett.

Nivåtallene fra simuleringen er i beskjeden grad påvirket av simuleringsstøyen gitt ved (4.2.2), med mindre man bruker utvalg på 1 prosent eller mindre. Derimot er det andre tall som er mer følsomme, herunder anslag på år-til-år endringer, langsiktige framskrivinger, forskjeller mellom virkningsberegninger og kalibrering av modellen. Her kan selv simulering av hele befolkningen ikke være nok for å redusere simuleringsstøyen til et akseptabelt nivå.

4.3. Variansreduserende trekkemetode for binomiske sannsynligheter

Det kan brukes ulike variansreduserende trekkemetoder som er slik at hver person fortsatt har en sannsynlighet p_i for å oppleve begivenheten, men samtidig slik at antall begivenheter som blir realisert beskrankes til forventningsverdien. Dette er realiserbart for binomiske sannsynligheter og kan enklest gjøres ved å summere sannsynlighetene for en type begivenheter fortløpende. Hver gang summen passerer et heltall lar man én begivenhet inntreffe (kildekoden i C#/MOSART omtales i avsnitt 4.5):

(4.3.1)

$$P_0 = z, z \text{ er en stokastisk variabel } z \sim U(0,1)$$

$$P_i = \begin{cases} P_{i-1} + p_i \text{ hvis } P_{i-1} < 1. \\ P_{i-1} - 1 + p_i \text{ hvis } P_{i-1} \geq 1 \end{cases}$$

$$x_i = \begin{cases} 1 \text{ hvis } P_i \geq 1 \\ 0 \text{ ellers} \end{cases}$$

I simuleringsmodellen implementeres dette ved at trekkemetoden er en klasse, eller inngår i en klasse, og at hver type begivenhet får et objekt av denne klassen med sin egen summevariabel *P*. Hva som er en type begivenhet kan gis litt forskjellig innhold. At for eksempel fødsler og pensjonering er to forskjellige typer begivenheter er ganske opplagt. Men man kan også si at fødsler for førstegangsfødende kvinner på 23 år er en type begivenhet som skal ha sitt eget trekkeobjekt og sin egen summevariabel. Hvordan man avgrenser hva som er en type begivenhet (antall trekkeobjekter), og rekkefølgen personene trekkes i, får mye å si for hvordan trekkemetoden vil fungere.

Trekkemetoden over tilfredsstiller kravene i avsnitt 4.1[21], men det oppstår en sterk negativ korrelasjon mellom personer som kommer tilstrekkelig tett etter hverandre i simuleringen. Det er derfor viktig at trekninger som skal være stokastisk uavhengige ikke kan komme (nær) etter hverandre i samme trekkeobjekt. Spesifikt betyr det at hvert trekkeobjekt (type begivenhet) må ha nok begivenheter og ikke-begivenheter per år, minst én av hver type hvis en person skal kunne benytte samme trekkeobjekt i to påfølgende år. Videre må medlemmer av samme hushold benytte ulike trekkeobjekter for at for eksempel det samlede arbeidstilbudet i husholdet skal få riktig fordeling.[22] Typisk vil de fleste trekkeobjektene være en matrise over kjønn og ettårig alder av hensyn til parallell prosessering, og det gjør at samme person aldri vil «møte seg selv» i trekkeobjektet (man blir ett år eldre for hvert år), og heller ikke ektefellen (motsatt kjønn).

[21] Med eksogene sannsynligheter p_i er dette relativt enkelt å vise når man forutsetter at startverdien *z* har en uniform fordeling (0,1). Nå vil sannsynlighetene være påvirket av forløpet i simuleringen, og da er dette vanskeligere å vise analytisk. Vi har imidlertid testet trekkemetodene, og brukt riktig er eventuelle skjevheter gitt ved (4.3.1) så små at de vanskelig lar seg påvise selv ved millioner av repeterte trekninger. For de multinomiske trekningene stiller det seg annerledes.

[22] Et veldig enkelt eksempel kan illustrere dette. La to ektefeller ha 50 prosent sannsynlighet for å jobbe, og slik at sannsynligheten for å jobbe er uavhengig av hva ektefellen gjør. Hvis ektefellene kommer etter hverandre i et trekkeobjekt med trekkemetoden i (4.3.1), så vil en og bare en av ektefellene jobbe. Den korrekte fordelingen er at i halvparten av husholdene så vil én være i jobb, i de resterende vil enten begge jobbe eller ingen jobbe. Er man opptatt av fordelingen av arbeid (og inntekt) over ektepar blir dette feil.

I den grad overgangssannsynlighetene p_i er upåvirket av simuleringen, vil variansen på totalt antall begivenheter av en type begrenses til den avrundingsfeilen som oppstår fordi sannsynlighetene ikke summerer seg nøyaktig opp til et heltall. Nå vil sannsynlighetene p_i i senere perioder være påvirket av hvilke personer som får begivenhetene i inneværende periode. Det er derfor et vesentlig poeng at trekkemetodene ikke bare reduserer variansen på antall begivenheter i inneværende periode, men også bidrar til å redusere variansen på de framtidige sannsynligheter. Det er også et poeng å redusere variansen på antall begivenheter av en type for ulike grupper av populasjonen, da vi i mange sammenhenger ser på utviklingen for mindre grupper.

Anvendelse av trekkemetoder
Gruppevis reduksjon i simuleringsstøyen kan oppnås på to måter. I den grad personene er sortert før de går inn i simuleringen vil metoden sikre at antall begivenheter i hver gruppe etter sorteringsvariablene blir begrenset til forventningsverdien. Alternativt kan man sikre dette ved å ha separate trekkeobjekter (summevariabler) for ulike grupper av populasjonen for hver enkelt begivenhet. I simuleringsmodellen er det ressurskrevende å få til en sortering utover det at simuleringen skjer automatisk sortert på kjønn og alder ut fra hvordan personene er organisert i populasjonslister. Ulemper med å bruke mange trekkeobjekter (summevariabler) er at det krever mer arbeid med kildekoden og i noen tilfeller også mer internminne. Videre kan man få problemer med at det blir få begivenheter per trekkeobjekt og at avrundingsfeilene vil bli mer vesentlige. Samtidig kan bruk av kjønn og alder som inndeling sikre at relaterte personer aldri kommer innom samme trekkeobjekt.

Ved parallellitet er det avgjørende at to tråder ikke jobber samtidig på samme trekkeobjekt (summevariabel), og at tilordningen av arbeidsoppgaver til trådene ikke har noen *mulighet* til å påvirke i hvilken rekkefølge populasjonen kommer inn i trekningene i et gitt trekkeobjekt. I praksis betyr det at trekkeobjektene (summevariablene) må deles opp etter de samme kjennetegnene som ligger til grunn for parallelliteten, som oftest kjønn og ettårig alder.

Videre finnes det en del metoder for å eliminere skjevheter som kan oppstå, se avsnitt 4.5.

I denne versjonen av MOSART er kjønn og ettårig alder brukt som variabler for inndeling av de aller fleste trekkeobjektene. Utover det er det få andre variabler som inngår, i første rekke variabler som har stor betydning for simuleringen av kjennetegnet (for eksempel om man er i jobb eller ikke, for det videre arbeidstilbudet).

4.4. Forrige variansreduserende trekkemetode

I forrige versjon av modellen benyttet vi en lignende variansreduserende metode som i forrige avsnitt, se Fredriksen (1998) for detaljer:

(4.4.1)

$Z_0 = 0$
$P_0 = 0$
z_i er en stokastisk variabel med uniform fordeling (0,1)

Hvis $P_{i-1} + p_i < 1$
Så:
    $P_i = P_{i-1} + p_i$
    Hvis $Z_i = 0$ og $p_i/(1-P_{i-1}) > z_i$
    Så:
        $x_i = 1, Z_i = 1$
    Ellers
        $x_i = 0, Z_i = 0$
Ellers:
    $P_i = P_{i-1} - 1 + p_i$
    Hvis $Z_{i-1} = 0$
    Så:
        $x_i = 1, Z_i = 0$
    Ellers:
        Hvis $P_i/(1-P_{i-1}) > z_i$
        Så:
            $x_i = 1, Z_i = 1$
        Ellers:
            $x_i = 0, Z_i = 0$
Slutt

Fordelene og problemene med denne trekkemetoden er helt analoge med trekkemetoden i (4.3.1). De eneste forskjellene som oppstår er at trekkemetoden (4.4.1) er litt mer komplisert å forklare, krever litt lenger kjøretid og øker variansen marginalt, spesielt på år-til-år endringer.

4.5. Implementering av trekkemetoder

I MOSART er trekkemetodene i avsnitt 4.2, 4.3 og 4.4 organisert i en egen klasse for binomiske trekninger, og hver anvendelse av trekkemetodene opprettes som et eget objekt. Klassen heter 't_draw_binomial' og ligger i filen 'draw_binomial.cs', se ellers vedlegg J for filreferanser. En trekkemetode kan opprettes ved følgende instruksjon:

```
t_draw_binomial draw_navn = new t_draw_binomial (g, identitet, pensjonssytem);
```

eller for eksempel

```
draw_navn = new t_draw_binomial [maksimum_indeks];
for (int i = 0; i < maksimum_indeks; i++)
    draw_navn[i] = new t_draw_binomial
        (g, identitet_ + g.left_int(i), ps);
```

'Identitet' er en tekststreng som bør (må) være unik for hvert trekkeobjekt og som brukes til å identifisere objektet når simuleringen eventuelt skriver ut feilmeldinger. 'Pensjonssystem' refererer til hvilket pensjonssystem (se avsnitt 2.7) trekkemetoden brukes i, typisk 'g.no_pension_system', da det bare unntaksvis er slik at noe simuleres uavhengig for ulike pensjonssystemer. Trekkeobjektene kan indekseres av to grunner. Den viktigste grunnen er der simuleringen jobber parallelt (samtidig) på flere personer, det er da nødvendig at hver mulig tråd får sitt unike trekkeobjekt som vil si at trekkeobjektet indekseres over for eksempel reg.status, kjønn, (ettårig) alder og pensjonssystem. I tillegg kan man indeksere over andre kjennetegn der man ønsker å oppnå sterk variansreduksjon på gruppenivå.

Trekkemetoden kan nå benyttes ved å skrive:

if (draw_*navn*.an_event**(***overgangssannsynlighet***))** ... **;**

eller

if (draw_*navn*(*indeks-verdi*).an_event(*overgangssannsynlighet***))** ... **;**

Hvilken trekkemetode som benyttes styres gjennom bestemte parametere i styrefilen 'parameters.con', se også vedlegg E. Parameteren 'stratified' bestemmer valg av trekkemetode; settes den til 1 brukes metoden i avsnitt 4.3.1, settes den til 2 brukes metoden i avsnitt 4.4.1, settes den til 0 blir den ordinære trekkemetoden 4.2.1 benyttet.

Parameteren 'stratified_restore' gjør at alle tellevariabler i de to variansreduserende trekkemetodene blir nullstilt på begynnelsen av hvert enkelt simuleringsår. De følgende trekkemetodene i avsnitt 4.6 og 4.7 har tilsvarende parametere, for øvrig er de tilknyttet den felles parameteren 'stratified_restore'.

Trekkemetodene kontrollerer for irregulariteter i sannsynlighetene og logger disse til filen 'sim.errors.draw', men uten å stoppe simuleringen. Irregulariteter som kontrolleres er sannsynligheter som er større enn én eller mindre enn null, samt multinomiske fordelinger som ikke summerer seg til tilnærmet én. Ved slutten av hvert år sjekkes det også for opphopning av informasjon i trekkemetodene for multinomiske begivenheter (se avsnitt 4.6). Ved å sette parameteren 'draw_report' i styrefilen 'output.documentation_and_special_tables.con' til 1, skrives det ut mer informasjon om alle trekkemetodene til filen 'sim.errors.draw' ved slutten av hvert år.

Det er mulig å overstyre valget av trekkemetode for utvalgte trekkemetoder ved å bruke navnet på metoden (det vil si tekstvariabelen '*identitet*' som skal følge hver trekkemetode) som styrevariabel i 'parameters.con', og deretter angi for denne metoden hvilken metode som skal benyttes. Slik overstyring blir logget til 'sim.errors.draw', hvis ikke er navnet angitt feil i 'parameters.con'.

4.6. Multinomiske begivenheter

For multinomiske begivenheter blir trekkemetodene mer kompliserte, og for denne typen begivenheter finnes det ingen tilsvarende enkel, forventningsrett og effektiv metode som i avsnitt 4.3[23]. Forsøker man å summere sannsynlighetene fortløpende for hvert enkelt utfall for å la en

[23] Med enkel mener vi at metoden kan håndtere ett individ om gangen og gjøre seg ferdig med denne personen uten å vite noe om de personene som kommer senere i simuleringen, samt at hele forhistorien kan oppsummeres til noen enkle målvariabler, for eksempel differansen mellom antall begivenheter og summen av sannsynlighetene. Med effektiv mener vi at usikkerheten begrenses til avrundingsfeilen som oppstår fordi summen av sannsynlighetene ikke summerer seg til heltall.

begivenhet inntreffe hver gang en av summene passerer et heltall vil man stadig få at noen personer enten får ingen utfall eller mer enn ett utfall, der de skal ha ett og bare ett utfall (for eksempel valg av fagfelt). Det betyr at man må velge mellom å gjøre kildekoden mer komplisert, bruke mer regnekraft, akseptere (små) forventningsskjevheter på individnivå eller akseptere at usikkerheten blir større enn den hadde behøvd å være. Vi skisserer nedenfor de metodene som er i bruk MOSART.

En ordinær trekkemetode kan organiseres på flere måter. I MOSART trekker modellen ett tilfeldig tall med uniform fordeling (0, 1) for hver mulig hendelse, og lar det skje en hendelse hvis sannsynligheten for hendelsen er større enn det tilfeldige tallet. Tilfeldig vil her si at det finnes en tallserie i C# som har denne egenskapen, og trekker vil si at man starter på et vilkårlig sted i denne tallserien, og hele tiden henter ut neste tall.

Enkle variansreduserende metoder

Det er mulig å etablere enkle variansreduserende trekkemetoder også for multinomiske begivenheter, i den betydning at man ikke trenger å vite noe om de som kommer etter den personen man simulerer og ved at all informasjonsbehandlingen skjer i standard metoder. Disse metodene er samlet i én klasse sammen med den ordinære trekkemetoden over. Klassen heter 'draw_multinomial', og etableres og brukes tilsvarende som i avsnitt 4.5:

```
t_draw_multinomial draw_navn new t_draw_multinomial
        (g, identitet, pensjonssystem, antall-utfall);
```

Trekkemetoden benytter et utfallsrom som er kompakt, et heltall og hvor første utfall har tallverdien 0. Dette forenkler implementasjonen og reduserer kjøretiden, og er uansett det formatet de fleste sannsynlighetene kommer på. Det fordrer en viss tilbakeregning til faktisk utfallsrom, men dette håndteres enklere utenfor trekkemetoden.

Trekkemetoden kan nå benyttes ved å skrive:

```
draw_navn.state(sannsynlighetsvektor);
```

Hvor 'sannsynlighetsvektor' er en vektor med sannsynligheter med samme egenskaper som angitt over og hvor hvert utfall har en fast plass med nummerering fra 0 til (*antall-utfall* – 1), 'state' returner da nummeret på utfallet som ble realisert. Krav og behov for sortering/ikke-sortering er de samme som for trekkemetoden i avsnitt 4.3. Parameteren 'stratified_multi' i styrefilen 'parameters.con' avgjør hvilken variansreduserende trekkemetode for multinomiske begivenheter som blir benyttet, settes denne lik 0 brukes den ordinære metoden med tilfeldig trekning. Metode 1 er forventningsrett og trolig i nærheten av å være så effektiv som en forventningsrett og enkel metode kan være, men krever normalt litt lenger kjøretid og noe mer internminne. Metode 2 krever lite regneressurser og gir bare uvesentlige forventningsskjevheter, men er på langt nær så variansreduserende som den første metoden.

OBS: Metode 1 kan under spesielle forhold kreve mye regnekraft, og da både i form av kjøretid og bruk av internminne. Slike irregulariteter logges til filen 'sim.errors.draw'. Et problem kan oppstå der sannsynligheten for ett bestemt av utfallene hele eller nesten hele tiden er stor, det vil si over 0,95-0,99. Da vil informasjon om de foregående trekningene hope seg opp, og simuleringen vil gå ut på grunn av både lange kjøretider og stort forbruk av internminne. Med så stor vekt på ett utfall kan det være like hensiktsmessig å bruke en hierarkisk trekning (det typiske utfallet først, deretter multinomisk på resten), eller hvis sannsynlighetene er homogene bruke en av tilnærmingene for

homogene sannsynligheter (se senere i dette avsnittet). Begivenheter med et stort antall utfall har også en tendens til disse egenskapene, og første håndgrep er da å begrense antall trekkegrupper for begivenheten, det vil si å bruke en felles for alle gruppene, og heller legge vekt på sorteringen. Videre bør man vurdere å bruke en av de andre trekkemetodene.

Hierarkisk trekning

Det er mulig å omgå problemet med multinomiske sannsynligheter ved å trekke hierarkisk, det vil si at man deler utfallene i to grupper, for deretter å trekke hvilken gruppe man skal fortsette med. Denne metoden gjentar man for den gruppen man valgte ut, og slik fortsetter man inntil man sitter igjen med en gruppe med ett og bare ett utfall. Et eksempel kan belyse det: I stedet for å velge ett av 22 fagfelt ved valg av utdanning, kan en person først velge om han skal inn i videregående utdanning eller i høyere utdanning. Velger han høyere utdanning kan han først velge om han skal inn på en høyskoleutdanning eller en universitetsutdanning. Velger han en høyskoleutdanning, kan han så velge mellom «harde» eller «myke» fag, og velger han «harde» fag står valget til slutt mellom ingeniørfag og økonomi. På hvert av disse trinnene kan man bruke en variansreduserende trekkemetode for binomiske sannsynligheter som i avsnitt 4.3. Dette tilsvarer på mange måter det vi gjør ved å trekke én og én begivenhet om gangen ellers i modellen. Ulempen med metoden er at den fort gjør kildekoden uoversiktlig hvis antallet utfall for én begivenhet er stort, slik som ved valg av fagfelt for videre utdanning (jamfør over). Videre medfører metoden tap av informasjon ved at sannsynlighetsfordelingen innad i de gruppene man velger bort ikke får være med å påvirke den videre simuleringen. Dette tilsvarer den egenskapen at selv om trekkemetoden eliminerer usikkerheten i inneværende periode, så oppstår det allikevel en usikkerhet i neste periode ved at det er tilfeldig hvilke sannsynligheter som overlever til neste periode.

Hierarkisk trekning bør derfor bare benyttes når antallet utfall for en begivenhet er lavt, hvor utfallene har en klar hierarkisk struktur, hvor undergruppene blir homogene og sannsynlighetsfordelingene innad i undergruppene er rimelig homogene. Et slikt eksempel kan være overgangen fra frisk til attføring eller uførhet, hvor man først kan trekke frisk eller attføring/uførhet, og deretter mellom attføring og uførhet hvis personen slutter å være frisk. Her er alle kravene over innfridd, spesielt vil sannsynlighetene for attføring og uførhet variere lite mellom ulike personer gitt at man skal inn på en av delene, samtidig som at man kontrollerer for kjønn og alder som skjer ved at vi trekker personene i den rekkefølgen.

Homogene sannsynligheter: enkel trekkemetode

I det tilfellet at sannsynlighetene er homogene kan det etableres en enklere trekkemetode som er enkel å bruke, krever lite regneressurser og som har en sterkt variansreduserende effekt. Klassen heter 't_draw_homogeneous', og benyttes ved:

```
t_draw_homogeneous draw_navn new t_draw_homogeneous
        (g, identitet, pensjonssystem, sannsynlighetsvektor);
```

Sannsynlighetene angis i det man oppretter trekkemetoden, blant annet for å binde trekkemetoden til å bruke samme sannsynligheter hele tiden. Det betyr også at sannsynlighetene for denne trekkemetoden ikke kan endres underveis i simuleringen med mindre man sletter og gjenoppretter trekkemetoden hver gang sannsynlighetene er justert. Trekkemetoden kan nå benyttes ved å skrive:

```
draw_navn.state();
```

Krav og behov for sortering/ikke-sortering er de samme som for trekkemetoden i avsnitt 4.3. Parameteren 'stratified_homogeneous' i styrefilen 'parameters.con' avgjør hvilken

variansreduserende trekkemetode som blir benyttet, og settes denne forskjellig fra 0 brukes den variansreduserende metoden ellers brukes den ordinære metoden med tilfeldig trekning.

Gruppevis homogene sannsynligheter
I det tilfellet at sannsynlighetene er gruppevis homogene og man har oversikt over den populasjonen man skal simulere, er det mulig å etablere en metode som utnytter informasjonen noe bedre enn den enkle trekkemetoden over. Metoden består i at man beregner hvor mange begivenheter man skal ha av hver type innen hver gruppe ved å ta heltallsverdien av produktet mellom sannsynligheten og antallet personer. Man kan så trekke tilfeldig hvilke personer som skal få hvilke begivenheter innen hver gruppe, gjerne slik at de spres jevnt utover gruppen hvis denne er sortert på andre kjennetegn enn de som påvirker sannsynlighetene. Restverdiene innen hver gruppe, det vil si det som er igjen etter at man har trukket fra antallet begivenheter fra summen av sannsynlighetene, anvendes som sannsynligheter på de personene som ikke ble trukket ut i første runde. Her kan man med hell benytte en av de variansreduserende trekkemetodene som er skissert over.

En klar ulempe med metoden er at den krever mer omfattende endringer i kildekoden. Videre krever metoden oversikt over hele populasjonen man skal simulere, også de som kommer etter den personen man til enhver tid simulerer. Videre taper metoden seg fort hvis gruppene med homogene sannsynligheter blir små, og det skal ikke mange forklaringsvariabler til før dette skjer. Metoden anbefales derfor bare når man har klart homogene sannsynligheter for store grupper, og to eksempler på dette i MOSART er valg av fagfelt og av klassetrinn i videre utdanning. Den virkelig store gruppen her er overgangen ut av grunnskolen for 16-åringer, men metoden har også betydning for mindre grupper, spesielt når utvalgsprosenten blir stor. Parameteren 'draw_homogeneous_groups' i styrefilen 'education_parameters.con' avgjør om denne tilnærmingen skal benyttes for valg av fagfelt og klassetrinn.

4.7. Uniforme fordelinger

I simuleringen av arbeidsinntekt, sparing og boligetterspørsel inngår det kontinuerlige restledd som til dels kan ha stor betydning for inntektsnivået for den enkelte. Effekten disse restleddene har på inntektsnivået har en viss homogenitet, og det kan derfor være et poeng å forsøke å trekke disse restleddene slik at de er spredd jevnt over mulighetsområdet. Mer presist vil restleddene være en transformasjon av en stokastisk variabel som er uniformt fordelt (0, 1), og poenget er få en observasjon i hver n-te fraktil når man til enhver tid betrakter de n siste observasjonene. For å gjøre metoden mer praktisk anvendbar trekker vi restleddene slik at de ti (første) påfølgende restledd er hentet tilfeldig innenfor hver sitt desil, og slik at rekkefølgen av desiler er tilfeldig. Hver ny gruppe av ti restledd trekkes på samme måte, men slik at man ikke trekker innenfor de prosentilene i hvert desil som allerede er brukt. Når hundre restledd er trukket ut, er alle prosentilene brukt opp, og metoden nullstilles. Metoden implementeres ved følgende to linjer:

```
t_draw_uniform draw_navn;
draw_navn = new t_draw_uniform(g, identitet, pensjonssystem);
```

Trekkemetoden kan benyttes ved å skrive:

```
draw_navn.z();
```

Hvor 'z' er tall med uniform fordeling (0, 1), enten tilfeldig eller med den angitte variansreduserende metoden over, avhengig av parameteren 'stratified_continuous' i styrefilen 'parameters.con'. De

samme reglene med hensyn til lokal avhengighet og sortering gjelder for denne metoden som for trekkemetoden i avsnitt 4.3. I simulering av personlige restledd for simulering av arbeidsinntekt tilordnes to restledd, ett for systematisk avvik i inntektsnivå og ett for årlige variasjoner i inntekt. Det er et poeng at parene av disse to restleddene er jevnest mulig fordelt utover en bi-uniform fordeling (0,1:0,1), og dette gjøres med en annen metode som er veldig analog i virkning, men noe komplisert å forklare (vi viser til kildekoden). Metoden implementeres ved følgende to linjer:

```
t_draw_bi_uniform draw_navn;
draw_navn = new t_draw_bi_uniform(g, identitet, pensjonssystem);
```

Trekkemetoden benyttes ved å skrive:

```
draw_navn.rv(z1, z2);
```

Hvor variablene 'z1' og 'z2' tilordnes hver sitt tall med uniform fordeling (0,1), enten tilfeldig eller med en variansreduserende metode, avhengig av parameteren 'stratified_continuous' i styrefilen 'parameters.con'. De samme reglene med hensyn til lokal avhengighet og sortering gjelder for denne metoden som for trekkemetoden i avsnitt 4.3. OBS: Metoden krever mye internminne hvis den får gå over for mange trekninger og den nullstilles derfor etter 4096 repetisjoner.

4.8. Finne tilfeldig person

En del av simuleringen består i å finne en tilfeldig person (med bestemte kjennetegn), for eksempel ved dannelse av samboerpar eller når et bestemt antall personer i et årskull skal utvandre ett år. Hvis man fjerner (utvandrer) første person som tilfredsstiller kravene vil personlistene bli skjeve, det vil si at personer med dette kjennetegnet vil bli underrepresentert i begynnelsen av listen. Spesielt hvis flere kjennetegn trekkes på denne måten vil man få koblinger mellom kjennetegn som er utilsiktet. Problemet kan løses på tre måter, enten ved å sortere hele listen på nytt[24], ved å gå gjennom personene på en tilfeldig måte eller ved å identifisere alle kvalifiserte personer og trekke en tilfeldig av disse.

Bruk av sorterte lister forutsetter at den gruppen man søker i har felles fødselsår og kjønn og at dette er forskjellig fra den personen man for øyeblikket simulerer (ved søk etter partner til et heterofilt forhold vil dette alltid være innfridd). Simulerer man hele befolkningen blir dette en særdeles krevende arbeidsoperasjon, både i form av kjøretider og ved at mye minne blir allokert i sorteringen.

Tilsvarende problemer oppstår om man identifiserer alle personer og legger disse til en egen liste. Når man simulerer hele befolkningen vil man få store lister som både øker kjøretiden vesentlig og krever omfattende allokering av internminne. Derfor brukes ikke denne metoden lenger i MOSART.

Vi har utviklet en alternativ metode som går gjennom populasjonen, det vil si en kohort etter kjønn, alder og reg.status, på en tilfeldig måte. Ved å bruke første person som innfrir kravene vil man ha funnet en tilfeldig person, samtidig som man unngår å korrumpere rekkefølgen i personlistene. Metoden initialiseres ved:

[24] Det er høyst utilstrekkelig å omplassere den personen man har funnet, blant annet fordi alle personene forut for den funne personen er et skjevt utvalg ved at de ikke tilfredsstiller kravene til den man leter etter. I stedet for å finne alle slike subtile avhengigheter, er det trolig mer effektivt og betryggende å sortere hele personlisten på nytt.

population.restore_random_list(*reg.status*, *kjønn*, *fødselsår*);

Neste tilfeldige person hentes ut ved:

population.next_random_person(*reg.status*, *kjønn*, *fødselsår*);

Når man har gått gjennom hele lista returneres 'null'. Det koster noe å initialisere metoden, slik at hvis man i en operasjon leter etter flere personer med samme karakteristika initialiserer man ikke nødvendigvis metoden mellom hver gang man henter ut en person.

4.9. Justering av overgangssannsynligheter

I en del sammenhenger er det relevant og viktig at modellen klarer å treffe eksogene skranker på bestemte størrelser. Spesielt gjelder dette de årene av framskrivingen som går gjennom historiske år. I tidligere modellversjoner med utdaterte utgangspopulasjoner var dette spesielt viktig for at simuleringen skulle være mest mulig riktig i det brukerne oppfatter som startåret for simuleringen. Dette krever (blant annet) at antall simulerte begivenheter stemmer med antall observerte begivenheter for historiske år, for eksempel antall fødsler per år. Forventet antall simulerte begivenheter vil være gitt som summen av de individuelle overgangssannsynlighetene, og det betyr at disse sannsynlighetene må justeres slik at det blir samsvar. Følgende betingelse må dermed være oppfylt, hvor p_i er de individuelle sannsynlighetene, og Q er skranken:

(4.9.1) $\sum_i p_i = Q$

Problemet med å justere sannsynligheter er at disse må innfri disse kravene (det andre kravet faller bort når det bare er to tilstander å velge mellom, det vil si binomiske sannsynligheter):

(4.9.2) $0 \leq p_i \leq 1$

(4.9.3) $\sum_i p_{ij} \equiv 1$ hvor *j* omfatter de ulike tilstandene man kan velge mellom

Justeringer som bryter disse kravene vil ikke bare skape problemer med fortolkningen, de vil også gi tekniske problemer. Med den ordinære trekkemetoden vil sannsynligheter utenfor det lovlige området (0, 1) i effekt blir erstattet med enten 0 eller 1, avhengig av hvilken vei feilen går, som gjør at justeringen blir feil. Med den nye variansreduserende trekkemetoden, se avsnitt 4.3, vil (restverdiene av) sannsynligheter utenfor det lovlige området (0, 1) bli overført til neste person i simuleringen, og det kan føre til svært vilkårlige endringer i sannsynlighetene. Med den forrige variansreduserende metoden, se avsnitt 4.4, vil også her de ulovlige sannsynlighetene bli overført til de etterfølgende personene, og i tillegg er det en viss fare for divisjon med null i helt spesielle tilfeller. I de tilfellene forenklede justeringsmetoder gir ulovlige verdier, så anbefales det å korrigere sannsynlighetene til intervallet (0, 1) før man anvender trekkemetoden, slik at ulovlige verdier ikke blir overført til etterfølgende personer, og slik at de ulike trekkemetodene opererer på samme måte.

Enkle justeringsmetoder

En **additiv justering** av sannsynlighetene, det vil si at man legger til en lik justeringsfaktor til hver sannsynlighet, vil i nesten alle tilfeller fungere dårlig. Er noen sannsynligheter nær 0 eller 1, sammenliknet med behovet for å justere, vil garantert noen sannsynligheter falle utenfor det lovlige

intervallet. Videre vil en additiv justering gi en svært urimelig fordeling av justeringen hvis sannsynlighetene er heterogene.

En **relativ justering** av sannsynlighetene, det vil si at man ganger alle sannsynlighetene med en felles justeringsfaktor, kan i en del tilfeller fungere bra. Dette gjelder spesielt der ingen sannsynligheter er store (nær 1) sett i forholdet til behovet for justering[25], og dette er ofte innfridd. Et typisk eksempel kan være fødselssannsynligheter, hvor størst sannsynlighet av interesse er 40 prosent i MOSART for øyeblikket. En relativ justering gir en relativt rimelig fordeling av justeringen, og er effektiv i den betydning at man kun trenger å summere initialverdiene av sannsynlighetene én gang.

Logistisk justering

En mer allment robust metode er å ta utgangspunkt i en logistisk formulering av sannsynlighetene, hvor p_i^0 er sannsynligheten forut for justeringen, det vil si:

(4.9.4) $$p_i^0 = \frac{e^{X_i\beta}}{1+e^{X_i\beta}}$$

Hvor X_i er en vektor av personkjennetegn/forklaringsvariabler og β er en vektor av parametere. En justering av sannsynlighetene kan foretas ved å legge en konstant til uttrykket $X\beta$, og deretter forenkle uttrykket noe[26]:

(4.9.5) $$p_i^{justert} = \frac{e^{X_i\beta+\delta}}{1+e^{X_i\beta+\delta}} = p_i^0 \frac{1+p}{(1+p_i)p} \qquad r = e^{\delta} - 1$$

Styrken med en slik logistisk justering er at de nye sannsynlighetene vil ligge i intervallet (0, 1) og at justeringen kan fortolkes innenfor en logit-modell for sannsynligheter (noe mange av sannsynlighetene er estimert på grunnlag av). Spesielt vil justeringsfaktoren δ inngå i formelen på samme måte som enkle årsdummier. Fordelingen av justeringen vil også bli rimelig målt i relative termer, gitt at man ser det i forhold til den residuale sannsynligheten (1-p) for p>0,5. Med p=0 og p=1, det vil si at utfallet er gitt, blir det ingen justering, som det også kan argumenteres for at er rimelig.

Merk at de andre forklaringsvariablene $X_i\beta$ faller ut etter forenklingen av uttrykket, noe som gjør at (4.9.5) relativt uproblematisk kan brukes selv om sannsynligheten opprinnelig er estimert på en annen måte.

Problemet består i å finne justeringsfaktoren δ, noe som krever en iterativ prosess. I MOSART bruker vi taylor-polynomer med den 1. deriverte av sannsynlighetene med hensyn på justeringsfaktoren. Først beregnes initialverdien for sannsynlighetene, p_i^0, noe som forkorter kjøretiden når p_i er en logit-funksjon eller et annet komplisert oppslag. Dernest gjennomføres en iterasjon som grovt beskrevet ser slik ut:[27]

[25] Er alle sannsynlighetene tilstrekkelig nærme én, og dermed vesentlig større enn null, kan man i stedet justere tvillingen (1-p) relativt.

[26] En relativ justering av $X_i\beta$ faller ikke heldig ut, fordi fortegnet på endringen av p da vil avhenge av fortegnet på $X_i\beta$, og fordi sannsynligheter nær 0,5 ikke vil bli justert i det hele tatt ($X_i\beta$ = 0).

[27] Oppsettet brukes med litt variasjoner over denne malen. I noen tilfeller settes startverdien for δ til en kvalifisert gjetning, for eksempel sluttverdien fra året før. Det er heller ikke gitt at man skal justere δ etter at man har bestemt seg for å avslutte iterasjonen ved tilstrekkelig konvergens; man har da formelt ikke sjekket at dette gir bedre konvergens, selv om det nesten alltid vil være tilfelle (når det gjøres likevel skyldes det at tallene ligger der nesten 'gratis' til bruk).

(4.9.6)

Iterasjoner er nummerert ved *v*, personer er referert til ved *i*. $\delta^0 = 0$.

$$p^v = e^{\delta^{v-1}} - 1$$
$$p_i^v = p_i^0 \frac{1 + p^v}{(1 + p_i^0)p^v}$$
$$p^v = \sum_i p_i^v$$
$$\frac{dp^v}{d\delta} = \sum_i p_i^v (1 - p_i^v)$$
$$\delta^v = \delta^{v-1} + \frac{Q - p^v}{\frac{dp^v}{d\delta}}$$

Begrens justeringen av δ til maksimalt □3 *per* iterasjon, dette forhindrer divergens
Avslutt iterasjonen hvis absoluttavviket (Q-p^v) er mindre enn en angitt grense
Avbryt iterasjonen hvis antall iterasjoner overstiger en angitt grense
Avbryt iterasjonen hvis $\frac{dp^v}{d\delta}$ blir for liten, spesielt vs. p^v

Denne iterasjonsprosessen krever noen ti-talls linjer i kildekoden og øker kjøretidene noe, men ikke avskrekkende. Med moderate justeringer og ikke for rigide krav til konvergens kommer prosessen normalt fram til en løsning i løpet av en til fire iterasjoner. Maksimalt antall iterasjoner kan styres gjennom parameteren 'max_no_of_iterations' i styrefilen 'parameters.con'. Konvergenskravet styres med parameteren ‘convergence_criteria’ i ‘parameters.con’, og er antall personer tillatt avvik (tallet kan angis som desimaler). Vil man ha et strengere (eller slakkere) konvergenskrav for en modul er det lagt til rette for at dette kan hardkodes. Merk at konvergenskravet ganges med utvalgsstørrelsen før det brukes i simuleringen.

Videre vil de fleste justeringene kunne styres gjennom parametere (i 'parameters.con') med navn av typen, hvor '*' er et navn på hendelsen (for eksempel 'fertility'):

- 'exogenous_*' Siste år justeringen blir gjennomført
- 'adjustment_*' Justeringsfaktoren som brukes etter dette, vanligvis δ-verdiene

Justeringene rapporteres til filer med navn av typen 'sim.adjust.*'. Objektet med felles variabler og hjelpemetoder (g), se vedlegg K, har noen metoder (‘g.adjust_prob’) som utfører logistisk justering på sannsynligheter.

Det er tre klasser i ‘iteration_parameters.cs’ (‘t_iteration_parameters_*navn*’) som pakker inn de viktigste variablene i oppsettet over (4.9.6). Se for eksempel ‘adjust_couple_formation_probablities’ og ‘adjust_wedding_probabilities’ i ‘sim_couples.cs’, samt justering av ekteskap i ‘read_simulated_population.cs’, for det binomiske tilfellet (det er nyanser rundt hvordan underliggende tellevariabler håndteres). Se ‘adjust_probabilities’ i henholdsvis ‘sim_disability.cs’ for det trinomiske tilfellet og ‘sim_sector.cs’ for det generelle multinomiske tilfellet. Variabelen ‘Q’ over er navngitt som ‘target’ og justeringsfaktoren ‘r’ er navngitt som ‘rho’.

Det er også en nyanseforskjell at der det er mulig, så blir den justerte sannsynligheten lagt tilbake i personobjektet i hver runde, og metoden jobber da med endringen (increment) i justeringsfaktorene. Dette krever trolig lite (ingen) regneressurser, og man slipper å legge inn denne tilbakeleggingen i en egen runde til slutt. Det gir identiske resultater (med unntak for avrundingsforskjeller i 16nde desimal).

Det er metoden 'compute_delta' som gjør hovedtyngden av jobben her. Ved å legge til to (valgfrie) parametere i metodekallet vil hver runde i iterasjon bli logget til 'sim.errors', se deklarasjonen i 'iteration_parameters.cs'. Det kan oppstå konvergensproblemer i enkelte metoder, og da vil en mer omfattende rapportering lette feilsøket.

Problemer med logistisk justering

Problemet med iterasjonsprosessen ovenfor er at kraftige justeringer kan få metoden til å bryte sammen. Spesielt må den eksogene skranken *Q* være slik at justeringen har mulighet for å konvergere (merk at det er strenge ulikheter):

N_{gitt} = Antall personer med $p_i^0 = 1$

N_{just} = Antall personer med $0 < p_i^0 < 1$

$0 < Q - N_{gitt} < N_{just}$

Gitt at alle personer har ekte sannsynligheter (ingen med 0 eller 1), betyr det at skranken *Q* må være større enn null og mindre enn antall personer i populasjonen. Dette er et faremoment blant annet i justeringen av utdanningstilbøyeligheter, hvor andelen som studerer i enkelte aldersgrupper er nær én og hvor det eksogene anslaget på elever og studenter kan bygge på andre befolkningsforutsetninger enn simuleringen. Justeringsfaktoren vil da gå mot pluss eller minus uendelig, og simuleringen vil raskt stoppe opp hvis man ikke legger inn en test der dette er et problem. En tilstrekkelig test vil være når $\frac{dp^v}{d\delta}$ faller under en gitt grense nær null (gjerne målt relativt til P_v), da er det ikke mer å hente ved videre justering av δ (dette ligger inne i 't_iteration_parameters_*type*').

Hvis initialverdien for justeringsfaktoren er for langt unna sluttverdien, der sluttverdien eksisterer, er det også en fare for at den foreslåtte inkrementelle justeringen av justeringsfaktoren, $\delta^v = \delta^{v-1} + \ldots$, faktisk bringer justeringsfaktoren lengre unna sluttverdien og at hele prosessen dermed divergerer. Dette kan løses ved å kontrollere at den nye justeringsfaktoren bringer prosessen nærmere skranken, og hvis ikke, halvere den inkrementelle justeringen inntil dette er innfridd. Alternativt kan man sette en øvre grense for hvor mye δ^v skal kunne endres per iterasjon, og ±3 er utfra erfaring tilstrekkelig for å sikre rask konvergens, samtidig som det forhindrer divergens (dette ligger også inne i 'iteration_parameters_*type*').

I noen tilfeller er også en grense på ±3 for mye, og det kan da bli oscillasjon. I det binomiske tilfellet er det også lagt inn ytterligere en brems i metoden som sjekker for oscillasjon, og demper justeringen slik at iterasjonen konvergerer. I det trinomiske og multinomiske tilfellet har ikke dette oppstått, og man kan da eventuelt hardkode en lavere grense enn ±3 ('max_increment').

Multinomiske fordelinger

Metoden som er skissert foran berører binomiske sannsynligheter. Det er i stor grad de samme kravene som stilles ved justering av multinomiske sannsynligheter, men metodene blir nødvendigvis noe mer komplisert fordi det er flere sannsynligheter (utfall) som skal justeres på en gang. Ved logistisk justering er det nødvendig med en tillempning av iterasjonsprosessen. Normalt vil det være nødvendig å ta hensyn til den kryssderiverte, spesielt for å sikre rask konvergens, og i noen tilfeller konvergens overhodet.

(4.9.7)

Iterasjon nummeres ved *v*, personer ved *i* og utfall ved *j* (og *k*), med *U* utfall. $\delta_u^0 = 0$.
Systemet må normeres, for eksempel ved at et av utfallene utelates fra justeringen, for eksempel det siste, som vil si at $\delta_N^j \equiv 0$.

$\rho_j^v = e^{\delta_j^{v-1}} - 1$ hvor $\rho_U^v = 0$
$C_i^v = 1 + \rho_j^v \sum_{j<U} p_{ij}^0$
$\rho_{ij}^v = p_{ij}^0 \frac{1+\rho_j^v}{C_j^v}$
$P_j^v = \sum_i \rho_{ij}^v$

$\frac{\partial p^v}{\partial i} = \sum_i p_{ij}^v \left(1 - p_{ij}^v\right)$
$\frac{\partial^2 p^v}{\partial j \partial k} = -\sum_i p_{ij}^v p_{ik}^v$
$a_i^v = Q_i - P_i^v$

Løs ligningssystemet med (*U*-1) ligninger/variabler mhp. Δ:
$D\Delta = A$
Hvor *D* er matrisen av deriverte ($\frac{\partial p^v}{\partial i}$) og kryssderiverte ($\frac{\partial^2 p^v}{\partial j \partial k}$)
Hvor Δ er vektoren av nye justeringsfaktorer (δ_j^v)
Hvor A er vektoren av avvik (a_j^v)
Begrens justeringen av hver δ_j^v til maksimalt □3 per iterasjon, dette forhindrer divergens
Avslutt iterasjonen hvis alle absoluttavviket (Q_j-P_j^v) er mindre enn en angitt grense
Avbryt iterasjonen hvis antall iterasjoner overstiger en angitt grense
Avbryt iterasjonen hvis (alle) $\frac{\partial p^v}{\partial i}$ er nær 0

Med tre utfall er det mulig (hensiktsmessig) å beregne den ene kryssderiverte, og løse ut den neste runden av justeringsfaktorer direkte. Se (for eksempel) på justeringen av sannsynlighetene for attføring og uførhet (‘adjust_probabilities’ i 'sim_disability.cs'.

Med mere enn tre utfall blir dette veldig komplisert, og en generell tilnærming med *n* lineære ligninger er da å anbefale. Se (for eksempel) på justeringen av sektoroverganger (‘adjust_probabilities’ i 'sim_sector.cs').

Ved mange tilstander kan man også prøve å beregne justeringsfaktorer ved å multiplisere hver sannsynlighet med forholdet mellom skranken og summen av sannsynlighetene, og deretter justere slik at summen blir lik én:

(4.9.8)

$$d_i = \begin{cases} \alpha \dfrac{\sum_i p_{ij}^0}{Q_i} & hvis\ utfall\ j\ skal\ beskrankes \\ 1 & hvis\ utfall\ j\ ikke\ skal\ beskrankes \end{cases}$$

$$\alpha = \begin{cases} \dfrac{N - \sum_i \sum_b p_{ij}^0}{N - \sum_b Q_b} & hvis\ minst\ ett\ utfall\ ikke\ beskrankes \\ 1 & hvis\ alle\ utfall\ beskrankes \end{cases}$$

$$p_{ij}^{\text{justert}} = \frac{p_{ij}^0 d_j}{\sum_j p_{ij}^0 d_j}$$

Hvor *i* refererer til person, *j* til utfall, *b* omfatter utfall som skal beskrankes, Q_b er skranken for utfall *b* og *N* er antall personer

Ved å gjenta metoden over vil summen av sannsynlighetene konvergere mot sine skranker med samme resultat som ved en logistisk justering med bruk av taylorpolynomer, men metoden bruker (mange) flere iterasjoner for å oppnå tilsvarende konvergens, spesielt ved strenge krav til konvergens. Det er viktig at summen av skrankene er konsistente med antall personer som skal simuleres, det vil si at summen av skrankene er mindre enn antall personer *N* som er tilgjengelig. I det tilfellet at alle utfallene skal beskrankes må summen av skrankene være identisk lik med antall personer som er tilgjengelig. Metoden trenger i tillegg noen sikringer mot at ikke alle skranker kan nås, det vil si at iterasjonen stoppes for de av utfallene hvis d_i går mot null eller mot uendelig, og at kravene til skrankene justeres på bakgrunn av dette. Se valg av fagfelt i simuleringen av utdanning for et eksempel på bruk av metoden.

5. Evaluering av IT-løsninger

Dette kapitlet drøfter forsøksvis noen punkter for evaluering av IT-løsningene i modellen, med hovedvekt på valg av programmeringsspråk og -tilnærminger, og hvordan unngå skjulte feil i applikasjonene som utvikles.

Utvikling og bruk av IT-løsninger i en modell som MOSART består av mange elementer som er vanskelig å vurdere opp mot hverandre. Dette gjelder blant annet arbeidsinnsats, investeringer i hardware og software, kjøretider for simuleringsmodellen og kvaliteten på det endelige produktet. De ulike kostnadselementene er til en viss grad substitutter, og det er et vesentlig poeng å få en balanse mellom innsatsen på de ulike områdene. Den viktigste kvalitetsegenskapen for IT-løsningene vil være gitt ved risikoen for såkalte skjulte feil, alle andre feil skal bli rettet opp etter hvert som de oppdages. Et så stort modellsystem som MOSART vil alltid inneholde feil som ennå ikke er avdekket, og arbeidet bør rettes inn mot å bringe risikoen for skjulte feil av vesentlig betydning ned på et akseptabelt nivå. Det er vanskelig å vurdere de andre utviklingskostnadene opp mot de konsekvensene mulige feil i kildekoden har for bruksverdien av modellen. Spesielt må dette ses på bakgrunn av at de forutsetningene som modellen bygger også vil inneholde svakheter som kan skape feil som er større enn det feil i kildekoden er opphav til.

5.1. Vurdering av teknologisk plattform

Brukergrensesnittet i MOSART er enkelt, men for en øvet bruker er modellen relativt lett å eksekvere, og modellen brukes per i dag kun innenfor prosjektgruppen som utvikler MOSART. Kjøretiden for modellen er om lag 1-2 timer med hele befolkningen fram til år 2100, og selv store testutvalg har kjøretider ned mot 5-15 minutter. Dette er akseptabelt for de fleste formål med modellen.

Vi ser ikke for oss at eksterne brukere skal få mulighet til å eksekvere MOSART. Grunnen er dels knyttet opp mot personvern ved at datagrunnlaget for modellen inneholder personopplysninger som kan danne grunnlag for bakveisidentifikasjon. Viktigere er det likevel at ekstern eksekvering forutsetter en langt bedre testing og dokumentasjon av modellen enn det vi har ressurser til, samt at eventuelle brukere må ha svært god kjennskap til modellen for å kunne gi en rimelig fortolkning av resultatene. Det er derfor mer effektivt at vi utfører oppdrag og formidler resultater ut fra hver enkelt oppdragsgivers forutsetninger.

Arbeidsinnsats ved drift er sterkt knyttet til brukergrensesnittet. MOSART har som nevnt et enkelt brukergrensesnitt basert på redigering av tekstfiler med parametere og direkte eksekvering av skreddersydde shellscript. Dette har de klare fordelene at brukergrensesnittet blir lett å utvikle og lett å vedlikeholde, noe som er svært viktig da modellen skal være åpen for et vidt spekter av virkningsberegninger. Videre er den effektive arbeidstiden man bruker på å eksekvere simuleringen normalt helt uvesentlig sammenliknet med tiden man bruker på å vurdere simuleringsresultatene. Kostnaden ved et enkelt brukergrensesnitt er at brukeren må ha en viss kompetanse på modellen, noe som de som utvikler modell er nødt til å ha uansett, men ikke nødvendigvis eksterne brukere. Videre blir det et gap mellom modellens IT-løsninger og modellens dokumentasjon som kunne vært unngått i et mer brukervennlig og automatisk system. Spesielt vil en teknisk dokumentasjon, for eksempel dette notatet, fort bli utdatert.

5.2. Kvalitetssikring

Når man utvikler en ny modellvariant, endrer man kildekoden på ett eller flere steder. Disse endringene kan inneholde feil som gjør at den nye modellvarianten gjør noe annet enn det som var intensjonen. Disse feilene kan deles inn i fire nivåer:

- Syntaksfeil; feil som gjør at kompilatoren ikke forstår instruksjonene i kildekoden
- Logiske feil; feil som gjør at simuleringen bryter sammen, for eksempel forsøk på å åpne en fil som ikke eksisterer
- Simuleringsmodellen gjør noe annet enn tiltenkt som gir åpenbare feil i resultatene
- Simuleringsmodellen gjør noe annet enn tiltenkt, men feilene er ikke åpenbare

Det siste punktet er hva vi kaller **skjulte feil** andre steder i notatet, og det er letingen etter skjulte feil som utgjør kvalitetssikringen. Egentlig vil det være en gradvis overgang mellom de to siste punktene. Det er normalt at kvalitetssikringen tar mer tid enn resten av utviklingsarbeidet til sammen. Ved syntaksfeil og logiske feil vil kompilatoren gi en feilmelding som i vekslende grad klargjør hva som er problemet og hvor i kildekoden problemet ligger, slik at man kan jobbe målrettet og effektivt inn mot å rette opp feilen. I nettopp dette arbeidet er det et miljø som tilbys av monodevelop kommer til sin rett. Spesielt vil eksekveringen ved logiske feil da avbrytes på det stedet i koden der feilen oppsto, og man kan enkelt inspisere innholdet av de variablene som forårsaket feilen. Også ved syntaksfeil vil monodevelop avbryte kompilering og sette markøren på det sted i koden som er årsaken til avbruddet, med en tilhørende forklaring.

Ved åpenbare feil i resultatene må man lete selv, men med god kjennskap til modellen vil store avvik ofte peke mot hvor feilen ligger. De skjulte feilene er vanskelige å oppdage, og når man har identifisert et lite avvik, kan det være vanskelig å finne årsaken(e) til avviket. Merk at feil som gir små avvik på nivåtallene, faktisk kan ha stor betydning når man utfører virkningsberegninger. Inspeksjon av variable som benyttes til mellomlagring, ved hjelp av en rekke ulike debug-verktøy i monodevelop, kan ofte være nyttig i dette arbeidet.

Det er en utfordring å vite når man bør avslutte letingen eter skjulte feil. Tar man i betraktning at simuleringsmodellen MOSART består av 139 000 linjer, blir det en uoverkommelig arbeidsoppgave å gjøre systemet helt feilfritt. Det er heller ikke med en arbeidsinnsats som står i rimelig forhold til hva disse ressursene alternativt kunne vært brukt til (økt kvalitet på dataene, parameterne og forutsetningene man ellers legger inn i simuleringen). Jobben består i å bringe risikoen ned på et akseptabelt nivå for at modellen inneholder skjulte feil av betydning.

Det er heller ikke snakk om å sette inn vesentlig mer ressurser i kvalitetssikringen, men snarere å få en best mulig fordeling mellom de ulike måtene å søke etter feil og sikre framgang i arbeidet med å få vekk eller aller helst unngå skjulte feil. Det som kanskje savnes mest for øyeblikket er noe mer systematikk i feilsøkingen, for eksempel at man systematisk kontrollerer et sett av tabeller/-resultater. Videre er arkiveringen og bearbeidingen av vanskelige feil mangelfull, det vil si at man tenker gjennom hvorfor feilen oppstod og bruker dette til å unngå samme eller lignende feil i framtiden.

Arkivsystemer

En grunnleggende forutsetning for å avdekke feil i et modellsystem er at man har noe å sammenlikne nye resultater med, og i denne sammenheng vil tidligere modellversjoner være til stor hjelp. Det er i denne sammenheng viktig å ha god dokumentasjon, samt metoder for å kunne finne tilbake til tidligere versjoner av modeller, data og andre parametere. Ved hjelp av 'diff' kan man effektivt finne forskjellene, gitt at man tar vare på tidligere modellversjoner og -varianter. Shellscriptet 'diffcon' kan også være til hjelp, se vedlegg C. Filene 'model.documentation', 'sim.documentation' og 'sim.input' er til stor hjelp når man skal finne kildene til en modellvariant og/eller simulering. Arkivering av modellversjoner og -varianter og simuleringer og sammenhengen mellom dem har trolig kommet opp på et tilfredsstillende nivå.

Det kan være hensiktsmessig å supplere nye versjoner med informasjon i vanlig språk om hva som er intensjonen bak hver ny modell eller simulering, men dette er opp til den enkelte bruker/utvikler.

Man kan for eksempel bruke filene 'sim.documentation' og 'model.documentation' som følger henholdsvis hver simulering og hver modellvariant. Endringskommentarer inne i kildekoden har den ulempen at de akkumuleres, og ved manglende vedlikehold av kommentarene kan de forvirre mer enn de opplyser.

Kontinuitet i arbeidet

Avbrudd i arbeidet med en ny modellvariant er ikke bare skadelig for effektiviteten, det gir også en betydelig risiko for at man fortsetter arbeidet på feil sted når man begynner igjen etter en uke eller flere. Planene for større modellendringer bør derfor dokumenteres, slik at man kan kontrollere at man har gjort det som er planlagt.

I denne prosessen kan monodevelop være til stor hjelp. Ved aktiv og fornuftig bruk av Tasks tilbys rike og gode muligheter for å strukturere pågående arbeid, samt markering av steder i koden man skal komme tilbake til. En oppramsing av slike Tasks visualiseres i et eget vindu, med mulighet for å hoppe direkte til den aktuelle lokasjonen.

Lesbar kildekode

En effektiv metode for å dokumentere en IT-løsning er at kildekoden er lesbar, det vil si at man kan forstå hva applikasjonen gjør ved hjelp av navn på og innholdet i variabler, klasser og metoder. C# ligger godt til rette for å oppnå lesbarhet, men det krever oppfølgning fra den som utvikler kildekoden. Dokumentasjon gjennom lesbare variabelnavn har den fordelen at dokumentasjonen i større grad oppdateres automatisk etter hvert som modellen utvikles. Notater som dokumenterer simuleringsmodellen eldes ofte raskt, og kommentarlinjer kan bli liggende helt uendret og dermed bidra til forvirring.

Lesbarhet av en omfattende applikasjon krever i tillegg til intuitive variabelnavn at man også bryter applikasjonen opp i hensiktsmessige moduler som er så små at modulen i seg selv er lesbar. Moduliseringen må ofte gå i flere trinn for å oppnå en høy grad av lesbarhet. Videre må hver modul ha et så klart innhold at man kan skjønne hva modulen gjør ut fra navnet på modulen. Til dette kan man bruke metoder eller klasser, se avsnitt 3.1. Noen ganger kan oppdelingen i moduler gjøre applikasjonen mer tunglest sett fra én synsvinkel, ved at man må forholde seg til flere moduler spredd over tilsvarende mange filer. Imidlertid vil arbeidet med simuleringsmodellen ha mange innfallsvinkler, og nærhet i én dimensjon gjør applikasjonen mer tunglest i en annen dimensjon. For eksempel kunne man samle alt om temaet utdanning på én fil, både innlesing av parametere og utgangspopulasjon og simulering av kjennetegnet. Det ville imidlertid gjort vedlikeholdet av simuleringsmodellen mer tungdrevet hvis man for eksempel skal gjøre noe generelt med innlesingen av parametere.

Omfattende moduler (mange linjer) bør inneholde noen kommentarer, da gjerne som generell omtale i innledningen til modulen. Overskrifter kan være hensiktsmessige for at man umiddelbart skal kunne se hvor en modul inne i en fil starter, men da må overskriften gå over flere linjer (tekst*boks*). Mer omfattende kommentarer inne i modulen bidrar imidlertid til å forlenge kildekoden og gjør den dermed mindre lesbar (det er ofte lettere å få oversikt når alt kommer inn på en skjermside). Kommentarer har også en tendens til å degenerere, og spesielt kan én-linjes kommentarer ofte inkorporeres i navn på variabler og/eller metoder.

Standardisering av arbeidsoppgaver

En oppsplitting av applikasjonen i metoder og klasser har også den fordelen at man kan standardisere arbeidsoppgavene, og dermed bruke felles hjelpemetoder til å utføre bestemte arbeidsoppgaver. Det kreves en viss arbeidsinnsats for å oppnå slik generalisering, men det gir en rekke fordeler. Det gir som nevnt økt lesbarhet, og har man lært en slik metode vil man umiddelbart

kjenne igjen hva en applikasjon gjør når metoden blir kalt på. Videre vil en velprøvd felles metode som virker ha nettopp den egenskapen at den virker, i motsetning til nyskrevet kildekode som nesten alltid inneholder feil.

Eksterne moduler

Oppsplittingen av applikasjonen i metoder og klasser kan gå lengre ved at man legger hver enkel modul ut på egne eksterne filer, se avsnitt 3.1. Dette har en kostnadsside ved at applikasjonen blir vanskeligere å redigere (på tvers) ved at kildekoden er splittet på ulike filer (med de redigeringsverktøyene vi har til rådighet). Derimot blir den enkelte modul lettere å håndtere, spesielt når man gjør feil med klammeparenteser ({...}). Videre vil bruk av moduler på eksterne filer ha den fordelen at all kommunikasjon med omverdenen går gjennom modulens parametere (eller moderklasse). Dette klargjør hvor eksterne variabler blir hentet fra, og bryter alle usynlige bånd mellom modulen og variabler som er globale på det nivået som modulen deklareres.

Stoppe simuleringen

Fordelen med syntaksfeil og logiske feil er at de stopper simuleringen og tvinger fram en oppretting. Spissformulert er det bedre å ha en programmeringsstil som gjør at man får syntaksfeil og logiske feil, hvis dette *reduserer faren* for feil som simuleringsmodellen godtar. Dette kan delvis oppnås ved å legge inn tester på at simuleringen forløper riktig, og deretter stoppe simuleringen ved å kalle metoden 'Error' i 'global.cs' ved avvik simuleringen ikke bør godta[28]. Feilen blir da synliggjort og man har et konkret sted man kan starte feilsøkingen. Metoden 'Error' blir til en viss grad benyttet ved innlesing av utgangspopulasjonen og simuleringsparametere, men kan trolig med hell benyttes langt flere steder i modellen, for eksempel når statusvariabler tilordnes illegale verdier.

Innlesing av utgangspopulasjonen og simuleringsparametere

MOSART er enkel når det gjelder innlesing fra (og utskrift til) eksterne filer, og det er fort gjort å få feil som er vanskelige å oppdage hvis den eksterne filen inneholder mer numerisk informasjon enn det simuleringen har bruk for. Slike feil oppstår spesielt i filer man endrer etter at man har utviklet en ny modellvariant, for eksempel når man lager nye versjoner av filer med overgangssannsynligheter. Ved bruk av grunnklassen for innlesing av filer ('read_base.cs', se vedlegg H) kan mange av disse feilene elimineres og avdekkes på en langt mer effektiv måte.

Det som savnes er et generelt opplegg som tester om det som er lest inn er i samsvar med de filene som brukes. Dette kan gjelde sammenlikning av innleste personer med filene med utgangs-populasjonen. Videre, og mer viktig, at de sannsynlighetene som objektene med overgangs-sannsynligheter leverer til simuleringen er i samsvar med de relasjonene som ligger på filene. Fraværet av slike generelle rutiner kan begrunnes/forsvares med at overgangssannsynlighetene er svært heterogent utformet, og at det er vanskelig å utforme en generell testrutine. Noen sannsynlig-heter foreligger som tabeller, andre som logit-funksjoner, og det er stor variasjon i hva slags forklaringsvariabler som brukes.

Bruk av modellen

Uansett programmeringsstil vil det være nødvendig å gå systematisk gjennom modellens resultater for å få risikoen for skjulte feil ned på et rimelig nivå. Problemet er å få til en systematisk gjennom-gang av alle modellens egenskaper, og vi foreslår i neste avsnitt noen kontrollpunkter avhengig av modellendringens art. Dokumentasjon av modellen, både teknisk og faglig, kan være et slikt grunnlag for en systematisk gjennomgang av alle modellens sider. En aktiv bruk av modellen vil

[28] Svakheter i forutsetningene eller modellspesifikasjonen kan gi resultater man definerer som feil, men det er ikke opplagt at simuleringen bør stanses i slike tilfeller.

bidra til å få ut feil, spesielt hvis man først søker etter feil hver gang et merkverdig resultat dukker opp snarere enn å konstruere en mulig forklaring på fenomenet. I dette ligger at man til den forklaringen man setter fram, utleder hvilke øvrige resultater dette bør lede til, og sjekker at dette faktisk stemmer i den simuleringen man ser på, både kvalitativt og helst også kvantitativt.

5.3. Elementer i en kontrollrutine

Vi omtaler i dette avsnittet noen punkter som bør eller kan inngå i en systematisk kontroll av en ny modellversjon eller en ny simulering. Kvalitetssikring av en simulering vil naturlig avhenge av hvilke endringer denne inkluderer sammenliknet med tidligere simuleringer, og vi vektlegger dette i de ulike punktene. Videre kan det være hensiktsmessig å være flere om jobben, fordi en enkelt person ofte kan slutte å være oppmerksom på enkelte typer feil, spesielt når man har jobbet lenge med et bestemt tema. Vedlegg F gir en oversikt over resultatfilene. Det er en fordel at den simuleringen man sammenlikner med går helt fram til år 2200, da det faktisk skjer en del modellbetingede endringer i populasjonen helt fram mot år 2080. Skal man ha fulle simulerte livsløp forutsetter det at simuleringen går minst 100 år forbi dette punktet. De fleste resultatfilene har en tidsdimensjon, enten det gjelder årstall, alder eller fødselsår, og det er en forutsetning at man har tilgang på et regneark med grafikk for å oppdage feilene på en sikker og effektiv måte. Om et avvik ett enkelt år er et tilfeldig avvik eller noe systematisk framgår ofte raskt av en figur. Videre vil ekstreme observasjoner avsløres raskt, spesielt gjelder dette tastefeil i tidsserietabellene. Etter hvert som feilene man søker blir mindre, må man sette en grense for når man tror at et avvik skyldes simuleringsstøy eller en svak modellspesifikasjon, snarere enn feil i kildekoden.

Dokumentasjon av simuleringen

En vesentlig forutsetning for kvalitetssikringen er at man sammenlikner simuleringsresultatene med det riktige og/eller relevante alternativet. Filene 'sim.documentation' og 'model.documentation' viser hvor simuleringen eller modellvarianten opprinnelig er kopiert fra, og brukergrensesnittet (*.con) viser hvilke filer og andre simuleringsparametere som er benyttet. Er man i tvil om filene allikevel er de samme, vil filen 'sim.input' fra de to simuleringene inneholde dato og klokkeslett for de versjonene av filene som er benyttet i de respektive simuleringene. Videre bør ingen av filene i brukergrensesnittet være endret etter at simuleringen er eksekvert. Shellscriptet 'diffcon' kan være til hjelp for å finne irregulariteter av denne typen, se vedlegg C. Feilmeldingsfilen 'sim.errors' er ofte tom, men kan inneholde advarsler om irregulariteter ('WARNING:'). De øvrige feilmeldingsfilene 'sim.errors.*' kan inneholde meldinger uten at simuleringen av den grunn inneholder feil.

Utskrift direkte fra simuleringen

Ved store modellendringer er det nødvendig å skrive ut variabelverdier direkte fra simuleringen, enten til egne filer eller til skjerm (helst via 'g.write_error'), hvor de fanges opp av 'sim.errors' for å se at modellen gjør det man forventer. Et element som bør undersøkes på denne måten er nye overgangssannsynligheter, ved at man bruker de metodene man har etablert til å skrive ut overgangssannsynligheter som så sammenliknes med et tilfeldig utvalg manuelt utregnete sannsynligheter fra de samme relasjonene.

Uttak av livshistorier ved store modellendringer

Når modellen utvides med nye kjennetegn eller nye statusverdier for bestemte variabler er det formålstjenlig å ta ut livshistorier for å avdekke enten illegale/unormale eller manglende bevegelser mellom ulike tilstander. Dette kan gjøres effektivt ved hjelp av modellpopulasjonen, ved å simulere et av testutvalgene og skape en modellpopulasjon sortert på individ og år. Modellpopulasjonen skrives også ut for basisåret, slik at en sammenlikning med utgangspopulasjonen manuelt eller maskinelt kan avsløre henholdsvis større og mindre feil i innlesingen.

Livsløpstabellene ('rl_*.prn') viser gjennomsnittlig utvikling over livet for en angitt gruppe årskull, og avdekker også grove feil i simuleringen. Livsløpstabellene vil også ha samsvarende tabeller for basisåret, hvor tilsvarende tall er rapportert for et tverrsnitt av befolkningen. Med en nettoinnvandring på 40.000 personer i året kan det ofte være hensiktsmessig å ta ut disse tabellene avgrenset til de som har bodd hele livet i Norge.

Det er også viktig at nye personkjennetegn trekkes gjennom alle mulighetene for dokumentasjon i simuleringen, spesielt gjelder dette at kjennetegnet legges inn i modellpopulasjonen (vedlegg I, 'model_population_file.cs', 'model_population_file.con') og i tabelluttaket (vedlegg G, 'table_base.cs').

Kohorttabeller
Uventede forskjeller mellom påfølgende årskull er egnet til å avdekke svakheter i modellspesifikasjonen, men kan også påpeke feil i innlesingen av utgangspopulasjonen. Kohorttabellene 'rc_*.prn' viser utviklingen i noen sentrale kjennetegn på gitte alderstrinn etter kjønn og fødselsår.

Framskrivinger
Simuleringen produserer et omfattende sett av tidsserietabeller som alene er egnet til å avdekke store feil, og med et egnet sammenligningsgrunnlag er de også egnet til å avdekke mindre feil.

Ved avvik, spesielt sammenliknet med tidligere framskrivinger, er det en fordel å starte med de mer grunnleggende faktorene. Det vil si at man først bør sjekke om befolkningstallene er de samme ('r_vital_statistics.prn' og 'r_person_by_age.prn'), deretter arbeidstilbudet, og til slutt pensjonsytelsene. Det kan hjelpe mye å bryte ned tallene på alder.

Reestimering av overgangssannsynligheter
En krevende kontroll av simuleringsresultatene er å bruke modellpopulasjonen til å reestimere de overgangssannsynlighetene man har lagt inn. Med begrensede ressurser er dette neppe hensikts-messig, men metoden vil avdekke svakheter i spesifikasjon og eventuelle feil i hvordan man håndterer forklaringsvariablene i simuleringsmodellen. Subsidiært kan man se på hvordan de sannsynlighetene som beregnes av simuleringsmodellen samsvarer med observerte antall begivenheter i basisåret for simuleringen (i den grad simuleringsmodellen regner ut dette ved å tilbakedatere kjennetegn), eller ved å se på hvor mye sannsynlighetene må justeres når simuleringen går gjennom estimeringsperioden (se vedlegg 4.9).

5.4. Svakheter i C#

Alle programmeringsspråk har svakheter som gjør at man får feil som er vanskelige å oppdage. Noen av de fellene vi har erfart gjennom bruk av C# omtales her (som ikke allerede er nevnt i kapittel 3).

Divisjon av heltall med heltall
Når to heltall divideres med hverandre, og resultatet tilordnes en variabel som er av typen desimaltall, vil resultatet trunkeres. Det vil si at desimalene blir strøket, som i dette eksempelet:

```
int a = 5;
int b = 4;
double c = a / b;
```

Her vil verdien av variabelen *c* nå være 1.0, ikke 1.25 som forventet. Løsningen er å benytte en såkalt 'cast' på en eller flere av høyresidevariablene:

```
int a = 5;
int b = 4;
double c = (double) a / b;
```

Dette gir 1.25 som verdi på *c*, da det er eksplisitt angitt at minst en av variablene som inngår i divisjonen er et desimaltall slik at resultatet også må være det.

Divisjon med null

I motsetning til en del andre programmeringsspråk vil ikke C# stoppe eksekveringen dersom en divisjon inneholder tallet 0 i nevneren, som i dette eksempelet:

```
double a = 5.0;
double b = 0.0;
double c = (double) a / b;
```

Her vil eksekveringen fortsette etter divisjonen. Verdien av variabelen c vil være 'Infinity', eller 'NaN' («Not a Number») dersom også telleren er 0. Det samme vil være tilfelle ved en del andre umulige operasjoner, som for eksempel å ta logaritmen av et negativt tall.

Metodekall med mange parametere

Enkelte metoder i simuleringsmodellen MOSART har mange parametere, og det er lett å forveksle parametere ved et metodekall, så lenge de parameterne man forveksler er av samme type. Noen av disse parameterne består av et endelig antall kategorier, definerer man parameteren som et heltall ('int') og tilordner hver kategori en tallverdi, så er det åpent for å forveksle den med andre parametere som også er heltallsvariabler. Deklarerer man parameteren som en 'enum', blir parameteren straks mer unik, og det blir tilnærmet umulig å forveksle parametere ved et uhell. Se også modulen 'enumerators.cs'.

Problemer med opprydning av internminne og parallellitet

Simuleringsmodellen MOSART har fortsatt enkelte feil knyttet til bruk av parallellitet og blant annet opprydning av internminne. Enkelte ganger oppstår det avvik i simuleringen, det vil si at simuleringen til tross for at den har samme forutsetninger og random-seed likevel gir andre resultater. Normalt skyldes dette feil i simuleringsmodellen, men vi står igjen med noen avvik som vi ikke finner ut av og som vi er ganske sikre på er svakheter utenfor kildekoden til MOSART. Et slikt tilfelle er når simuleringer går med parallellitet, og opprydning av internminnet iverksettes samtidig som tabelluttaket teller opp populasjonen. Da settes simuleringsmodellen på vent, men de trådene som var i ferd med å hente en variabelverdi fra en person via et metodekall, får returverdien 0 (eller 'false') uavhengig av hva verdien var for personen. Vi ser dette tydelig ved at simuleringen ikke endrer forløp, og ved at andre tabeller som teller opp denne personen samme år har riktige verdier. Det betyr at enkelte tabeller i enkelte celler kan ha avvikende verdier, men i en simulering med hele befolkningen blir avviket på +/- 1 - én - person, der det er personer som telles opp.

Referanser

Andreassen, Leif og Dennis Fredriksen (1991): MOSART - en mikrosimuleringsmodell for utdanning og arbeidsstyrke, Økonomiske analyser nr.2/91, Statistisk sentralbyrå.

Andreassen, Leif, Truls Andreassen, Dennis Fredriksen, Gina Spurkland og Yngve Vogt (1993): *Framskriving av arbeidsstyrke og utdanning*, Rapporter 93/6, Statistisk sentralbyrå.

Andreassen, Leif, Dennis Fredriksen og Olav Ljones (1996): The Future Burden of Public Pension Benefits, in Ann Harding (redaktør): *MicroSimulation and Public Policy*, Amsterdam: North Holland.

Andreassen, Leif og Inger Texmon (2000): Using Dynamic MicroSimulation Models for Policy Analysis and Research, in Gupta, Anil og V.Kapur (redaktører): *MicroSimulation in Government Policy and Forecasting*, North Holland.

Andreassen, Leif, Dennis Fredriksen, Hege Marie Gjefsen, Elin Halvorsen og Nils Martin Stølen (2020): The dynamic cross-sectional microsimulation model MOSART, International Journal of Microsimulation; 13(1); 92-113. DOI: 10.34196/IJM.00214.

Brinch, Christian N., Dennis Fredriksen og Ola Vestad (2015): Life Expectancy and Claiming Behaviour in a Flexible Pension System, The Scandinavian Journal of Economics, https://doi.org/10.1111/sjoe.12271.

Fredriksen, Dennis (1992a): Datagrunnlaget for trygdemodellen MOSART-T, Interne notater 92/7, Statistisk sentralbyrå.

Fredriksen; Dennis (1992b): MOSART 2 - en mikrosimuleringsmodell for alders- og uføretrygd fra Folketrygden, paper presentert på Velferdstatsprogrammets programseminar 16.-17. november 1992, Drammen.

Fredriksen, Dennis (1993a): MOSART - teknisk dokumentasjon, Notater 93/41, Statistisk sentralbyrå.

Fredriksen, Dennis (1993b): Dokumentasjon av input til MOSART, Notater 93/42, Statistisk sentralbyrå.

Fredriksen, Dennis (1995): MOSART, Teknisk dokumentasjon, Notater 95/5, Statistisk sentralbyrå.

Fredriksen, Dennis (1996): Datagrunnlaget for modellen MOSART, 1993, Notater 96/9, Statistisk sentralbyrå.

Fredriksen, Dennis (1998): *Projections of Population, Education, Labour Supply and Public Pension Benefits - Analyses with the Dynamic MicroSimulation Model MOSART*, Sosiale og økonomiske studier 101, Statistisk sentralbyrå.

Fredriksen, Dennis og Pål Knudsen (2015): MOSART 5.7 - Teknisk dokumentasjon, Interne dokumenter 2015/19, Statistisk sentralbyrå.

Fredriksen, Dennis og Gina Spurkland (1993): *Framskriving av alders- og uføretrygd fra ved hjelp av mikrosimuleringsmodellen MOSART*, Rapporter 93/7, Statistisk sentralbyrå.

Fredriksen, Dennis og Nils Martin Stølen (2011a): Utforming av ny alderspensjon i folketrygden, Rapporter 22/2011, Statistisk sentralbyrå.

Fredriksen, Dennis og Nils Martin Stølen (2011b): Pensjonsreformen - økte utgifter til alderspensjon motvirkes av sterkere vekst i arbeidsstyrken, Økonomiske analyser 6/2011, Statistisk sentralbyrå.

Fredriksen, Dennis og Nils Martin Stølen (2014): Pensjonsreformen - stort omfang av tidliguttak øker pensjonsutgiftene på kort sikt, Økonomiske analyser 6/2014, Statistisk sentralbyrå.

Fredriksen, Dennis og Nils Martin Stølen (2018): Reform av offentlig tjenestepensjon, Rapporter 2018/33, Statistisk sentralbyrå.

Fredriksen, Dennis og Halvorsen, Elin (2019). Beregninger av pensjonsformue. Rapporter 2019/29, Statistisk sentralbyrå.

Fredriksen, Dennis og Nils Martin Stølen (2021): MOSART og analyser av AFP i privat sektor, Vedlegg 2 i Utredning av en mulig reformert AFP-ordning i privat sektor, Hovedrapport, LO/NHO.

Fredriksen, Dennis, Kim Massey Heide, Erling Holmøy og Nils Martin Stølen (2003): Makroøkonomiske virkninger av endringer i pensjonssystemet. Rapporter 2003/13, Statistisk sentralbyrå.

Fredriksen, Dennis, Trude Gunnes og Nils Martin Stølen (2008): Oppdaterte framskrivinger av arbeidsstyrke, pensjonsutgifter og finansieringsbyrde. Økonomiske analyser 4/2008, Statistisk sentralbyrå.

Fredriksen, Dennis, Herman Kruse og Nils Martin Stølen (2022): Dynamisk justering av aldersgrensene i pensjonssystemet, Rapporter 2022/22, Statistisk sentralbyrå.

Gjefsen, Hege Marie (2013): Educational behavior in the dynamic micro-Simulation model MOSART, Rapporter 15/2013, Statistisk sentralbyrå.

NOU 1992:1: *Trygghet - Verdighet - Omsorg*. Norges Offentlige Utredninger 1992:1, Sosialdepartementet.

NOU 1992:26: *En nasjonal strategi for økt sysselsetting i 1990-årene*. Norges Offentlige Utredninger 1992:21, Finansdepartementet.

NOU 1993:11: *Mindre til overføringer - mer til sysselsetting*. Norges Offentlige Utredninger 1993:11, Finans- og Tolldepartementet.

NOU 1994:2: *Fra arbeid til pensjon*. Norges Offentlige Utredninger 1994:2, Sosial- og helsedepartementet.

NOU 1998:10: *Fondering av folketrygden?* (Molandutvalget). Norges Offentlige Utredninger 1998:10, Finans- og tolldepartementet.

NOU 1998:19: *Fleksibel pensjonering*. Norges Offentlig Utredninger 1998:19, Finans- og tolldepartementet.

NOU 2000:8 *Arveavgift*. Norges Offentlige Utredninger 2000:8, Finansdepartementet.

NOU 2000:14: *Frihet under ansvar, om høyere utdanning og forskning i Norge* (Mjøsutvalget). Norges Offentlig Utredninger 2000:14, Kirke-, undervisnings- og forskningsdepartementet.

NOU 2000:21: *En strategi for sysselsetting og verdiskapning*. Norges Offentlige Utredninger 2000:21, Finansdepartementet.

NOU 2004:1: *Modernisert folketrygd. Bærekraftig pensjon for framtida*. Norges Offentlige Utredninger 2004:1, Finansdepartementet og Sosialdepartementet.

NOU 2022:7: *Et forbedret pensjonssystem*. Norges Offentlige utredninger 2022:7, Arbeids- og inkluderingsdepartementet.

Pensjonskommisjonen (2002): Mål, prinsipper og veivalg for pensjonssystemet. Foreløpig rapport fra Pensjonskommisjonen. Finansdepartementet og Sosialdepartementet.

St.meld. nr. 4 (1992-1993): *Langtidsprogrammet 1994-97*. Finans- og tolldepartementet.

St.meld. nr. 35 (1994-95): *Velferdsmeldingen*. Sosial- og helsedepartementet.

St.meld. nr. 4 (1996-1997): *Langtidsprogrammet 1998-2001*. Finans- og tolldepartementet.

St.meld. nr. 30 (2000-2001): *Langtidsprogrammet 2002-2005*. Finansdepartementet.

St.meld. nr. 12 (2004-2005): *Pensjonsreform - trygghet for pensjonene*. Finansdepartementet.

St.meld. nr. 5 (2006-2007): *Opptjening og uttak av alderspensjon i folketrygden*. Arbeids- og inkluderingsdepartementet.

Stensnes, Kyrre, Nils Martin Stølen og Inger Texmon (2007): Pensjonsreformen, virkninger på statsfinanser, effektivitet og fordeling, Rapporter 2007/11.

Vedlegg A: Katalogoversikt

Alle IT-løsningene som inngår i driften av MOSART er samlet i et arkivsystem hvor alle katalog- og filnavn har '/mosart' som stamme. Når det refereres til kataloger videre i vedleggene er dette underkataloger av '/mosart' (med mindre annet sies).

Alle IT-løsningene som inngår i driften av MOSART er samlet i et arkivsystem hvor alle katalog- og filnavn har '/mosart' som stamme. Når det refereres til kataloger videre i vedleggene er dette underkataloger av '/mosart' (med mindre annet sies).

Katalog	***Innhold***
/mosart/bin	Katalog med ulike shellscript som støtter driften av MOSART, se vedlegg C
/mosart/prog	Katalog hvor hver modellvariant har sin egen underkatalog, og hver underkatalog har et fullstendig sett av den kildekoden inngår i simuleringsmodellen, samt en kopi av brukergrensesnittet
/mosart/prog/mo75_1b[29]	Basisversjonen av MOSART som omtales i dette notatet
/mosart/input	Katalog som inneholder alle eksterne filer med overgangssannsynligheter og utgangspopulasjoner, utover det som ligger i brukergrensesnittet og kildekoden til simuleringsmodellen. Hver versjon av MOSART har en egen underkatalog med et fullstendig sett av filer. Hver underkatalog for en versjon er igjen delt opp i kataloger med ulike tema. Vedlegg D omtaler temaer og filer brukt i MOSART 7.5
/mosart/input/v75	De versjonene av eksterne filer som er brukt i MOSART 7.5
/.../population	Utgangspopulasjonen, hvert utvalg har sin egen underkatalog
/.../befinn	Befolkningsframskrivinger
/.../befreg	Regional dimensjon i befolkningsframskrivingene
/.../migration	Inn- og utvandring
/.../mortality	Dødelighet
/.../fertility	Fruktbarhet
/.../household	Endringer i husholdningsstatus, det vil si barns flytting til og fra foreldrehjemmet

[29] Jobbene er kjørt med /MOSART/prog/mo75_1, denne versjonen inneholder noen små opprettinger, hvorav en nødvendig, som ikke har tilbakevirkning på disse kjøringene (en feil i model_population_file.cs, to språklige korreksjoner),

/.../couples	Endringer i samliv og ekteskapelig status
/.../region	Flytting
/.../education	Skolegang og innvirkning på utdanningsnivå
/.../pension	Satser med mer i pensjonsreglene
/.../public_occupational_pension	Satser med mer i pensjonsreglene i offentlig tjenestepensjon
/.../private_occupational_pension	Satser med mer i pensjonsreglene i privat tjenestepensjon
/.../afp	Satser med mer i pensjonsreglene for AFP (avtalefestet pensjon)
/.../social_security	Overganger i trygdestatus
/.../disability	Uføretrygd
/.../labour_supply	Yrkesdeltaking gitt ved AKU-tall
/.../labour_income_recipients	Yrkesdeltaking gitt ved pensjonsgivende inntekt, sannsynlighet for å ha slik inntekt
/.../labour_income_level*i*	Yrkesdeltaking gitt ved pensjonsgivende inntekt, nivået på denne inntekten. Postskript '*i*' viser til ulike versjoner
/.../sector	Overganger mellom ulike sektorer i arbeidsmarkedet (fem sektorer: ikke ansatte, privat ansatt uten AFP, privat ansatt med AFP, medlem i SPK, andre offentlig ansatte (KLP)
/.../income	Relasjoner for sparing og boligetterspørsel
/.../common_support	Diverse felles støtte
/mosart/res	Simuleringsresultater, hvor hver simulering har sin egen underkatalog, og hvor grupper av simuleringer kan være samlet i egne mellomkataloger
/mosart/population	Grunnlagsdata for modellen MOSART
/mosart/population/v2022	Grunnlagsdata som er brukt i MOSART 7.5
/mosart/population/adapt_population_2022	Kildekode for tilrettelegging av utgangspopulasjoner med utgangspunkt i forannevnte grunnlagsdata

/mosart/support	Katalog med ulike ettermodeller til MOSART, ikke vedlikeholdt
/ssb/stamme04/mostry	Område som er tilgjengelig fra sl-mosart-01, som blant annet kan brukes til å flytte filer mellom MOSART og resten av SSB
/mosart/wapi	Katalog for midlertidige filer som brukes av mono, kjøresystemet for C#.

Vedlegg B: Historikk

Nedenfor følger en kort beskrivelse av sammenhengen mellom de ulike versjonene av MOSART, med referanser til tekniske navn opp mot innhold (endringer), aktualitet, anvendelser, dokumentasjon og eventuelle avvik i filreferanser. Gamle versjoner av MOSART må kompileres på nytt for å kunne eksekveres under nyeste versjon av operativsystemet på Linux. Versjoner skrevet i Simula lar seg ikke lenger eksekvere (til og med MOSART 3.5). Dato-angivelse for modellversjoner angir når referansebanen ble eksekvert for denne versjonen (oppstart), enkelte av modellene kan ha fått tilført nye elementer senere, da som oftest forhold som ikke påvirker simuleringen (referansebanen), for eksempel utvidelser av tabelluttaket, rekompilering etter oppdateringer i operativsystemet eller alternative utforminger av pensjonsreglene. Endringer og utvidelser mellom modellversjoner er gitt egne versjonsnavn, disse er ikke omtalt her, og innhold som har nedfelt seg i modellen er omtalt i den etterfølgende modellversjonen.

Teknisk navn/dato	***Beskrivelse***
MOSART 1	Den første versjonen av MOSART inneholdt demografiske kjennetegn, utdanning og arbeidstilbud. Datagrunnlaget for utgangspopulasjonen omfattet demografiske kjennetegn og utdanning, og hadde 1987 som basisår
MOSART 1.1	
Oktober 1990	Den første versjonen av MOSART 1 ble utviklet på en såkalt stormaskin (Comparex) i Statistisk sentralbyrå, og baserte seg på befolknings-framskrivingen fra 1990 og øvrige forutsetninger i stor grad fra 1987. Ulike versjoner ble benyttet i NOU 1992:1 (Omsorgsutvalget) og NOU 1992:26 (Sysselsettingsutvalget), og en kort dokumentasjon finnes i Andreassen og Fredriksen (1991)
MOSART 1.2	
Juli 1992	MOSART som en modell for skolegang og arbeidstilbud ble sluttført med Andreassen med flere (1993). Forutsetninger utover demografi ble oppdatert til 1991. Modellversjonen ble benyttet blant annet i St.meld. nr. 4 (1992-1993) (Langtidsprogrammet 1994-97). Modellen ble overført fra stormaskin til en server med Unix som operativsystem. Kildekoden og deler av de andre filene ligger på katalogen '/mosart/prog/mo12', men på grunn av svake arkiveringsrutiner i starten av prosjektet mangler flere filer
MOSART 2	MOSART ble utvidet til å omfatte pensjoner fra folketrygden og pensjons-givende inntekt. Datagrunnlaget for utgangspopulasjonen ble oppdatert og utvidet til å omfatte tidsserier på demografiske kjennetegn, utdanning, trygd, og pensjonsgivende inntekt. Utgangspopulasjonen fikk 1989 som basisår og er dokumentert i Fredriksen (1992a)
MOSART 2.0	
September 1992	En foreløpig versjon ble brukt i St.meld. nr. 4 (1992-1993) (Langtids-programmet 1994-97) og NOU 1993:11 (Sysselsettingsutvalget), og er dokumentert i Fredriksen (1992b)
MOSART 2.1	
Mars 1993	Utvidelsen av MOSART til å omfatte trygd ble sluttført med Fredriksen og Spurkland (1993). Befolkningsutviklingen er basert på befolknings-framskrivingen fra 1990, øvrige forutsetninger er fra årene 1989-1991.

	Modellversjonen ble benyttet blant annet i Andreassen med flere (1996) og NOU 1994:2 (Fra arbeid til pensjon). Teknisk dokumentasjon finnes i Fredriksen (1993a) og Fredriksen (1993b)
MOSART 2.2	
April 1994	MOSART fikk sin første variansreduserende trekkemetode
MOSART 2.3	
Desember 1994	MOSARTs IT-løsninger ble vesentlig forbedret (modulisering, kortere kjøretider og språkvask/oversettelse til engelsk) og det ble etablert et opplegg for simulering av historiske data. Modellversjonen har 1993 som basisår og ble benyttet blant annet i St.meld. nr. 35 (1994-1995) (Velferdsmeldingen). MOSART fikk også sin første omfattende engelskspråklige dokumentasjon med Fredriksen (1998). Teknisk dokumentasjon finnes i Fredriksen (1995)
MOSART 3	Utgangspopulasjonen ble oppdatert og fikk 1993 som basisår, samtidig som metodene for trekking av utvalget ble forbedret, se Fredriksen (1996) for dokumentasjon av det nye datagrunnlaget. Oppdateringen av utgangs-populasjonen var også motivasjonen for nytt basisnummer
MOSART 3.1	
November 1996	MOSART ble utvidet til å omfatte husholdningsstatus og -tilhørighet. Uføreovergangene ble utvidet med attføring og reestimert. Befolknings-utviklingen er basert på befolkningsframskrivingen fra 1996, øvrige forutsetninger er fra årene 1993-1996. Modellversjonen ble brukt i St.meld. nr. 4 (1996-1997) (Langtidsprogrammet 1998-2001), og en kort dokumentasjon finnes i Andreassen og Texmon (2000)
MOSART 3.2	
Desember 1999	MOSART ble gjennom flere trinn utvidet til å omfatte skatt, sparing, bolig og arv, samtidig ble sentrale overgangssannsynligheter reestimert/remodellert og oppdatert. Befolkningsutviklingen er basert på befolknings-framskrivingen fra 1996, mens øvrige forutsetninger i hovedsak er fra 1997. Stort arbeidspress og små innholdsmessige endringer i befolknings-framskrivingene fra 1996 til 1999 var årsaken til at siste befolknings-framskriving ikke ble benyttet. Ulike modellversjoner har blitt benyttet i NOU 1998:10 (Fondsutvalget), NOU 1998:19 (Førtidspensjoneringsutvalget) og NOU 2000:8 (Arveutvalget). I en spesialversjon (mo32_sus) som ble utviklet i NOU 2000:21 (Sysselsettingsutvalget), ble innvandrings-forutsetningene fra 1999 lagt inn i tillegg til ulike antakelser om innvandreres kjennetegn
MOSART 3.3	
September 2000	I hovedsak en oppdatering med 1999 som basisår. Modellversjonen ble benyttet i St.meld. nr. 30 (2000-2001) (Langtidsprogrammet 2002-2005), Pensjonskommisjonen (2002) og Fredriksen med flere (2003). MOSART har etter dette blitt benyttet intensivt i pensjonsreformen, spesielt til analyser av alderspensjonen i folketrygden, og nedenfor omtales bare noen av anvendelsene

MOSART 3.4	
April 2003	Overgangssannsynligheter for arbeidsinntekter ble reestimert/forbedret, og simuleringen av noen andre kjennetegn ble justert på bakgrunn av dette. Befolkningsutviklingen er basert på befolkningsframskrivingen fra 2002, øvrige forutsetninger er i hovedsak fra 2001. En spesialversjon (mo34_pk) har blitt benyttet av NOU 2004:1 (Pensjonskommisjonen). En revidert versjon med oppretting av noen mindre feil (mo34_pk_op) ble benyttet i St.meld. nr. 12 (2004-2005) (oppfølging av Pensjonskommisjonen)
MOSART 3.5	
Juni 2005	Arbeidet med å lage en historisk simulering på MOSART 3.4 ledet fram til såpass mange (mindre) modifikasjoner/opprettinger, at en ny modellversjon ble etablert. Innholdsmessige endringer går på glatting av arbeidstilbudsfunksjonene for de over 60 år, fruktbarhet blant studenter, fruktbarhetshistorie for innvandrere og på justeringsmekanismene for fruktbarhet, utdanningsvalg og arbeidstilbud. Videre ble en rekke virkningsberegninger inkorporert i hovedmodellen. Dette gjelder simuleringen av historiske data, likestilling og levealderseffekter på arbeidstilbudet. En typehusholdsmodell ble etablert. I tillegg ble IT-løsningene i omfattende grad renskrevet og gitt mere intern dokumentasjon. Utover dette vil MOSART 3.5 ha samme forutsetninger som MOSART 3.4, og vil i de fleste henseende gi samme resultater. Modellversjonen ble benyttet i Stensnes med flere (2007) og St.meld. nr. 5 (2006-2007)
MOSART 3.6	
Høsten 2006	Den første versjonen av MOSART skrevet i C#. Funksjonaliteten er stort sett identisk med MOSART 3.5, men tilgangen på regnekraft økte radikalt ved overgangen fra gamle Solaris/Unix-servere til nye Linux-servere
MOSART 3.7	
Våren 2008	Oppdatert våren 2008 med nye befolkningsframskrivinger. Mulighet for å beregne flere versjoner av pensjonsytelser fra de samme simulerte livsløpene, men med ulik utforming av regelverket ('pension_system' og 'ps' i kildekoden). Dette eliminerer i sterk grad virkningen av simuleringstøyen vs. *virkningen* av regelendringene. En kort dokumentasjon finnes i Fredriksen med flere (2008)
MOSART 3.8	
August 2009	Oppdatert våren 2009 med nye befolkningsframskrivinger. Lagt inn multithreading som reduserer kjøretiden betydelig
MOSART 4	Datagrunnlaget for modellen og utgangspopulasjonene ble oppdatert, og omfattet hele befolkningen til og med 2008. Vi gikk over til en mer inkrementell oppdatering av datagrunnlaget, med hensyn til både aktualitet og nye kjennetegn. Kildekoden ble revidert med tanke på at modellen skal fungere effektivt og stabilt på store datamengder. I dataene inn til modellen blir alle kronebeløp angitt nominelt, mens alle data ut av modellen blir lønnsdeflatert til det angitte basisåret for priser
MOSART 4.2	
Juni 2010	Oppdatert våren 2010 med nye befolkningsframskrivinger. Ny utgangspopulasjon ikke tatt i bruk. Benyttet i Fredriksen og Stølen (2011a)

MOSART 4.3	
Juni/Sept 2011	Oppdatert våren 2011 med nye befolkningsframskrivinger og som standard utgangspopulasjon har modellen hele befolkningen per 31.12.2007. Ny utdanningsmodell lagt inn basert på endringer i utdanningssystemet, utdanningsstatistikken og med nyere data. Gammel utdanningsmodell kan fortsatt brukes, men blir ikke vedlikeholdt utover denne modellversjonen. Benyttet i Fredriksen og Stølen (2011b)
MOSART 4.4	Mellomversjon, brukt blant annet til analyse av etterlattepensjon
MOSART 5	Brutto inn- og utvandring ble lagt inn i modellen
MOSART 5.1	
Juli/August 2012	Oppdatert våren 2012, utgangspopulasjon for 2008, nye befolkningsframskrivinger og basisår 2011. Brutto inn- og utvandring tatt i bruk. Oppdaterte utdanningsoverganger
MOSART 5.2	
Våren 2013	Mellomversjon, utvikling av massivt repeterte simuleringer (av et angitt årskull, for å kunne beregne forventningsverdier på individnivå), ny dødelighetsrelasjon, elementer av en regional dimensjon, tilrettelegging for overføring av data til Lotte-trygd
MOSART 5.3	
Juni 2013	Oppdatert våren 2013, utgangspopulasjon for 2010, basisår 2012. Resirkulering av internminne og nytt mer fleksibelt tabelluttak. Kildekoden har vært gjennom en renskrivning (forkortelser, engelsk og bedre navnsetting)
MOSART 5.4	
Våren 2014	Mellomversjon, inkludering av nye utdanningsoverganger med innvandring som kjennetegn
MOSART 5.5	
Juni 2014	Oppdatert våren 2014, nye befolkningsframskrivinger, basisår 2013. Regional dimensjon (flytting) inkludert. Benyttet i Fredriksen og Stølen (2014)
MOSART 5.6	
	Mellomversjon for utvikling av offentlig tjenestepensjon. Omfattende renskrivning av kildekoden. Første oppsett for tilbakegående tabeller innarbeidet. Repeterte simuleringer og ny dødelighetsrelasjon er benyttet i Brinch med flere (2017)
MOSART 5.7	
Juni 2015	Oppdatert med ny utgangspopulasjon med 2014 (ID-numre) og 2013 (siste år med data som brukes) som basisår, ellers sammenfallende med MOSART 5.6. Første versjon hvor MOSART looper hvert enkelt historisk år som om det var en del av simuleringen (beregning av pensjonsytelser, overgangssannsynligheter, tabellprogram med mere)

MOSART 5.8	
April 2016	Lagt inn befolkningsframskrivinger fra 2016, i bruk fra og med MOSART 5.8.-5 (juni 2016). Lagt inn første versjon hvor enkelte kjennetegn kan simuleres for historiske år. Innarbeidet ny simulering av sektoroverganger. Tatt i bruk ny utgangspopulasjon med 2015 (ID-numre) og 2014 (siste år med data som brukes) som basisår
MOSART 6	MOSART blir anvendelig for to større kommende pensjonsreformer (OFTP, AFP)
MOSART 6.0	
Oktober 2017	Skisse til ny offentlig tjenestepensjon og mulig ny AFP i privat sektor blir lagt inn. Lagt inn første opplegg for å rekalibrere alderseffekter etter kjønn x ettårig alder for enkelte sentrale overgangssannsynligheter. Anvendt på ny offentlig tjenestepensjon i Fredriksen og Stølen (2018)
MOSART 6.1	
Juli 2018	Lagt inn befolkningsframskrivinger fra 2018
MOSART 6.2	
November 2018	Ny server med 1 Tb RAM og oppdatering av kompilator. Utvidelse av opplegget for resirkulering av internminne. Lagt inn nytt forbedret opplegg for repeterte simuleringer for pensjonsformueprosjektet (Fredriksen og Halvorsen, 2019)
MOSART 6.3	
Mai 2019	Tatt i bruk ny utgangspopulasjon med 2018 som basisår. Første versjon hvor simuleringen starter der dataene slutter for det enkelte kjennetegnet
MOSART 6.4	
September 2019	Revidert pensjonsformueberegning
MOSART 6.5	
Februar 2020	Rekalibrering av modellen
MOSART 6.6	
Juni 2020	Lagt inn befolkningsframskrivinger 2020 og rekalibrert modellen
MOSART 6.7	
November 2020	Rekalibrering av modellen. Anvendt på mulig ny AFP privat sektor i Fredriksen og Stølen (2021). Parameterfilen 'change_parameters.con' legges inn som håndterer flere pensjonssystemer på en enklere måte
MOSART 7	Monodevelop blir tilgjengelig for MOSART. Kildekoden revideres omfattende gjennom flere runder, blant annet tas maksimal linjelengde ned til 115 tegn per linje
MOSART 7.0	
April 2021	Første versjon av MOSART 7
MOSART 7.1	
Oktober 2021	Ny utgangspopulasjon med 2020 som basisår. Anvendt i Fredriksen med flere (2022) og NOU 2022:7

MOSART 7.2	
Mai 2022	Befolkningsframskrivingene fra 2022 blir lagt inn
MOSART 7.3	
April 2023	Ny utgangspopulasjon med 2022 som basisår. Første versjon av privat tjenestepensjon er lagt inn. Første forsøk med å legge estimeringsrutiner inn som separate moduler
MOSART 7.4	
Juni 2024	Befolkningsframskrivingene fra 2024 blir lagt inn. Omfattende opprydning
MOSART 7.5	
November 2024	Generasjonsskifte. Full revisjon av modulene for fødsler og husholdning

Vedlegg C: Driftsstøtte

På katalogen '/mosart/bin' finnes en rekke shellscript som skal forenkle driften av MOSART, her dokumenteres bruken av disse. Noen av shellscriptene kan ta en del tid, spesielt de som søker gjennom hele katalogstrukturen. Hvis katalogen '/mosart/bin' inngår i ens egen *path* er det nok å skrive navnet på shellscriptet for å få det eksekvert, uansett hvor man måtte være i katalogsystemet. Path er de katalogene som gjennomsøkes for å finne programvare, og kataloger kan legges til din egen path i filen '.cshrc', eventuelt '.bashrc', på hjemmekatalogen.

Noen av shellscriptene bruker midlertidige arbeidsfiler med navnestruktur '/mosart/bin/*shellscript-navn*.*dine-initialer*.*'. Når jobben er fullført, vil disse arbeidsfilene normalt bli slettet enten umiddelbart eller neste gang shellscriptet blir eksekvert. For de shellscriptene som er avhengige av arbeidsfiler, kan man bare eksekvere én jobb om gangen på hver type shellscript uten tap av informasjon og/eller risiko for feil i utførelsen. All relevant informasjon blir normalt bare skrevet til skjerm (standard output), og det er opp til bruker å styre informasjonen til fil.

Felles shellscript

diffcon	Sammenlikner brukergrensesnittet i to simuleringer
diffcon2	Avgrenser sammenligningen til forskjeller i parameterverdier som kan påvirke simuleringen (utelater tabeller, modellpopulasjon og kommentarer)
fkat	Finner en katalog for en simulering på grunnlag av siste navneledd
fkatall	Finner alle kataloger med angitte siste navneledd
mosdok	Gir en oversikt over disse shellscriptene
newjob	Lager oppsett for ny simulering
newmodel	Lager oppsett for ny modellversjon
newsolution	Lager en .csproj-fil slik at modellversjonen kan åpnes i monodevelop
removestopfile	Sletter 'stopfile' hvis simuleringen er slutt eller avbrutt
delete_backup	Sletter backup av tidligere simuleringer
delete_simulation	Sletter fullstendig en simulering eller modellvariant
delete_repeated_simulations delete_failed_repeated_simulations	Sletter repeterte simuleringer

Shellscript/opsjoner	***Virkning***
diffcon *kat1 kat2*	'*diffcon*' sammenlikner 'job.exe' og '*.con' i katalogene *kat2* og *kat1*. Utelates *kat2* benyttes gjeldende katalog, for øvrig må *kat1* og *kat2* angis tilpasset 'fkat'. Alle forskjeller skrives til skjerm
fkat *kat*	'*fkat*' søker etter første katalog med 'job.exe' som i denne rekkefølgen matcher '*kat*' (fullt navn), '/mosart/prog/*kat*', '*kat*' (lokalt navn), første '*kat*'

som finnes som en underkatalog i gjeldende katalog, deretter søkes rekursivt i moderkatalogene. Man må ikke skrive hele siste ledd i katalognavnet, men jo mer man skriver, jo sikrere blir treffet, bruk eventuelt 'fkatall' (se under) for å teste for tvetydighet. Det forutsettes at *kat* er en katalog og at 'job.exe' ligger i katalogen. '*fkat*' skriver det fulle navnet på katalogen *kat* til skjerm, hvis den ikke finnes skrives feilmeldingen 'fkat *feilmelding*'. Utelates *kat* brukes den katalogen man startet '*fkat*' fra. '*fkat*' brukes av flere av de andre shellscriptene

fkatall *kat*

'*fkatall*' søker etter samme kataloger som 'fkat', og skriver alle treffene fortløpende til skjerm, inkludert de katalogene som mangler 'job.exe'

mosdok *shellscript-navn*

'*mosdok*' lister ut kommentarlinjene på toppen av '*shellscript-navn*', gitt at det er et av de shellscriptene som inngår i dette vedlegget. Utelates '*shellscript-navn*' listes i stedet ut mulige shellscript

newjob *kat1 kat2* -group *gruppe-navn* –r

'*newjob*' oppretter en katalog for å utføre en simulering. Katalogen får navnet '/mosart/res/*eksisterende-gruppe*/*kat2*', hvor katalogen som opprettes ikke må eksistere fra før, og *kat2* må heller ikke eksistere som en modellvariant eller som en simulering rett på 'res'-katalogen. *kat2* skal kun angi siste ledd i katalognavnet. Standard plassering (*eksisterende*-gruppe) blir i samme katalog som simuleringen kopieres fra (*kat1*), denne kan eventuelt overstyres av '-group *gruppe-navn*', se nedenfor
Alle filer hentes fra katalogen *kat1* som '*newjob*' finner ved å bruke 'fkat'. Brukergrensesnittet (job.exe, *.con) etableres ved å kopiere over de nødvendige filene. Samtidig lages en fil med navn 'sim.documentation' med referanse til *kat1*. Tilslutt skrives til skjerm navnet på den nye katalogen (*kat2*) *og* eventuelle tvetydigheter om hvor denne er hentet fra. Ved feil i parameterne skrives en feilmelding til skjerm

De øvrige parameterne er valgfrie, og kan komme i vilkårlig rekkefølge. '*-group*' styrer simuleringen til en underkatalog under 'res' med navnet *gruppe-navn*, som da må eksistere på forhånd, og som kan være et flerleddet navn. Parameteren '-r' nullstiller random-seed

newmodel *kat1 kat2*

'*newmodel*' oppretter en katalog med navnet '/mosart/prog/*kat2*', hvor katalogen som opprettes ikke må eksistere fra før. '*newmodel*' kopierer over 'job.exe', '*.con', kompilator og '*.cs' fra katalogen '/mosart/prog/*kat1*'. Samtidig lages en fil med navn 'model.documentation' med referanse til *kat1*. Parameteren '-r' nullstiller random-seed. Til slutt skrives til skjerm navnet på den nye katalogen. Ved feil i parameterne skrives i stedet ut '/mosart/bin' og man finner en feilmelding på '/mosart/bin/newmodel.*dine-initialer*'

newsolution

'*newsolution*' oppretter en .csproj-fil som omfatter alle '*.cs'- og '*.con'-filene som eksisterer i katalogen. Denne filen åpnes fra monodevelop slik at modellen kan redigeres, og eventuelt eksekveres, der

removestopfile *kat*

'*removestopfile*' sjekker om *kat* går eller er skrivebeskyttet, og hvis ikke (med rimelig sikkerhet), så slettes 'stopfile' slik at simuleringen kan startes på

nytt. Hvis *kat* utelates, brukes gjeldende katalog, for øvrig må *kat* angis tilpasset 'fkat'

delete_backup '*delete_backup*' finner alle kataloger som ligger under gjeldende katalog og har et navn som starter med 'backup_' og som har skrivetillatelse for den som har eksekvert shellscriptet, og sletter deretter alt innholdet i disse katalogene inkludert katalogene selv

delete_simulation '*delete_simulation*' sletter en simulering eller en modellvariant, inkludert alle underliggende kataloger (kravet er at job.exe og parameters.con finnes i katalogen)

delete_repeated_simulations
'*delete_repeated_simulations*' sletter alle underliggende kataloger som har navn som at det er en repetert simulering

delete_failed_repeated_simulations
'*delete_failed_repeated_simulations*' sletter alle underliggende kataloger som har navn som at det er en repetert simulering og som ikke har blitt simulert ferdig

Vedlegg D: Filer som inngår i en simulering

Mye av informasjonen som inngår i en simulering med MOSART er organisert på (felles) eksterne filer. Grunnen til å bruke eksterne filer er at det da blir lettere å endre på informasjonen som inngår i en simulering. Grunnen til å bruke felles filer er at mye av informasjonen vil være lik fra simulering til simulering, og ved å bruke referanser til felles filer vil man spare plass på disken (gjelder spesielt utgangspopulasjonen) og det vil være lettere å kontrollere to simuleringer opp mot hverandre. Den eksterne informasjonen er spredd på flere filer på grunn av heterogeniteten i type og tema for informasjonen, og dette gjør det enklere å endre den normalt lille biten av informasjon man endrer fra simulering til simulering. Videre er filene med ekstern informasjon organisert på ulike kataloger i hovedsak etter tema for å gjøre det lettere å finne fram, og med unntak av simuleringsmodellen og brukergrensesnittet er disse katalogene samlet på en felles katalog med navn av typen '/mosart/input/v*ii*/*tema*' hvor '*ii*' refererer til versjonsnummer og '*tema*' til tema.

Nedenfor er en oversikt over hvilke typer filer som inngår i en simulering med MOSART 7.5. Parameternavn i 'input.con' og katalognavn er angitt under noen av temaene. Hver linje ellers angir en fil, hvor første kolonne i tabellen angir navnet på standardversjonen av filen(e), mens andre kolonne gir en kort beskrivelse av innholdet. Filene skal i utgangspunktet inneholde en viss dokumentasjon av sitt eget innhold, ved siden av å være dokumentert selvstendig andre steder. Ett unntak er populasjonsfilene, og disse er omtalt i vedlegg L og M.

Emner

- Simuleringsmodell
- Brukergrensesnitt
- Utgangspopulasjon
- Øvrige tema

Simuleringsmodellen

Simuleringsmodellen hentes inn ved parameteren 'simulation_model' i input.con, og versjonen som dokumenteres her ligger på '/mosart/prog/mo75_1b'. Under følger en kort beskrivelse av enkelte av modulene, i alfabetisk rekkefølge. Fila 'komp.modules' under hver modellvariant vil ha uttømmende (og oppdatert) liste for alle filene med kildekode ('*.cs')

*_base.cs	Moduler som er felles grunnlag for andre moduler (grunnklasser)
already_earned_pension_entitlements.cs	Klasse som håndterer individuelle data om allerede opparbeidede pensjonsrettigheter
befinn.cs befreg.cs	To moduler som utfører aggregeringsrollen i Befinn og Befreg, slik at alle mulige kombinasjoner av forutsetningene i befolkningsframskrivingene er tilgjengelige i simuleringsmodellen MOSART
binary_tree.cs	Klasse som håndterer personer i et binærtre, en metode egnet for dynamisk sortering av personer
cohort_list.cs	Klasse som håndterer en kohort av personer med samme kjønn, fødselsår og reg.status
contribution_rates.cs	Tabelluttak av lønnssum, pensjonsutgifter og bidragsrater

control_demographic_data.cs
control_non_demographic_data.cs
: Sjekker personobjektene for inkonsistenser i data ved utvalgte punkter

create_type_persons.cs
: Lager typepersoner for bruk i pensjon- og skatteberegninger, neppe operativ

draw_*.cs
: Klasser for trekkemetoder, alle er avledet av 'draw_base.cs'

enumerators.cs
: Deklarasjon av kategorivariabler slik at de blir unike

estimate_*.cs
: Supplerende moduler som estimerer/beregner/tabellerer overgangssannsyligheter og andre parametere som brukes i simuleringen. Alle har 'estimate_base.cs' som grunnklasse

find_median_equivalence_income.cs
: Finner median ekvivalensinntekt for populasjonen hvert år

freelist.cs
: Håndterer objekter (også arrays) som skal resirkuleres

generate_output_variables
: Shellscript som støtter kompileringen med å finne variabler som inngår i tabelluttaket ('table_base.cs') og legger dette ut på 'output.variables'

global.cs
: Klasse med blant felles referanser til andre moduler, noen felles konstanter og mindre hjelpemetoder. Objektet navngis som 'g' i kildekoden

iteration_parameters.cs
: Tre klasser som understøtter justering av sannsynligheter

komp
: Shellscript som kompilerer simuleringsmodellen

komp.modules
: Liste med alle C#-filer som inngår i kildekoden

linear_equations.cs
: Klasse som løser lineære ligninger

loop_historical_data.cs
: Går gjennom populasjonen for de historiske årene som om det var en simulering, men erstatter simuleringen med faktiske data

model.cs
: Hovedprogrammet i simuleringsmodellen

model.exe
: Den kompilerte versjonen av simuleringsmodellen

model.documentation
: Tekstfil med referanse til hvor modellen er kopiert fra, med muligheter for å legge inn egne kommentarer

model_name.cs
: Klasse som inneholder navnet på modellvarianten, da navnet *alltid* er unikt for modellvarianten har vi lagt navnet på en egen fil, slik at forskjeller i de andre modulene da vil skyldes reelle endringer og ikke bare endringen i navn

model_population_file.cs

	Modul som skriver ut modellpopulasjonen til fil
modelbackup/original_files	Kopi av kildekoden slik den var da simuleringsmodellen ble opprettet
outfile_base.cs	Liten klasse som støtter opp om utskrifter av *enkle* filer innenfor rammen av en simulering
pension_debt.cs	Beregner nåverdien av ulike pensjonsytelser på individnivå av hittil opptjente pensjonsrettigheter. Output håndteres i stor grad gjennom 'table_pension_debt.cs' og 'model_population_file.cs'
person.cs	Klasse som definerer alle kjennetegn for en person
population_list.cs	Klasse som håndterer populasjonen av personer i kohortlister etter kjønn, fødselsår og reg.status. Det tilhørende objektet navngis som 'population' i kildekoden
present_value_computations.cs	Beregner nåverdier på individnivå for et utvalg av variabler
ps.cs	Klasse som understøtter beregninger på flere pensjonssystemer i samme simulering
read_*.cs	I de fleste tilfeller moduler som leser inn ulike tema av overgangssannsynligheter og har 'read_base.cs' som grunnklasse (unntakene er 'read_parameters.cs', og 'read_initial_population.cs'). Navnene skal være selvforklarende og har som oftest en tvilling i 'sim_*.cs'. Mange av modulene har en egen '*.con'-fil
read_initial_population.cs	Leser inn den ordinære utgangspopulasjonen
read_parameters.cs	Klasse som leser inn filer i brukergrensesnittet og håndterer dette på en robust måte
read_simulated_initial_populasjon.cs	Simulerer en utgangspopulasjon i 1960 basert på aggregert statistikk og overgangssannsynligheter
sim_*.cs	Moduler som simulerer ulike temaer, og har 'sim_base.cs' som grunnklasse, og uten unntak en tvilling i 'read_*.cs'
sim_control.cs	Logger tidsbruk med mere for hver modul for hvert år
sim_information.cs	Skriver ut grunnleggende egendokumentasjon
table.cs	Organiserer tabelluttaket
table_base.cs	Henter ut informasjon fra personobjektene og støtter opp om utskrift til tabeller og annen type output
table_*.cs	Moduler for ulike typer tabelluttak, har 'table_base.cs' som grunnklasse

update.cs Oppdaterer modellpopulasjonen ved starten av hvert år

work_distributor.cs Klasser som forenkler organiseringen av multithreading

Brukergrensesnitt

Brukergrensesnittet følger hver modellversjon og simulering og referansebasen som omtales her finnes på '/mosart/res/mo75_1/reference'

job.exe Shellscript som eksekverer simuleringen på riktig måte

jobr.exe Shellscript som produserer repeterte simuleringer

input.con Navn på filer med simuleringsmodell, utgangspopulasjon og overgangssannsynligheter

parameters.con Valgbare parametere i simuleringen som ikke inngår i neste blokk

education_parameters.con
labour_income_level.con
pension_parameters.con
actuarial_tables.con
public_occupational_pension.con
private_occupational_pension.con
afp.con
toiler_pension.con
income.con
tax_parameters.con
Valgbare parametere med mere for ulike temaer av simuleringen. Ulike årganger av skatteregler ligger lagret som 'tax_parameters.v*iiii*.con'

change_parameters.con
Fil med valgfritt innhold for å håndtere endringer når man bruker flere pensjonssystemer, et ryddigere alternativ til å opprette separate filer av for eksempel typen 'pension_parameters_2.con'. Fungerer også når man bare har ett pensjonssystem

output.con
output.*.con Valg av egendokumentasjon og tabeller med aggregerte resultater

model_population_file.con
Valg av personer, år og variabler i modellpopulasjonen

pension_debt.con Beregning av pensjonsformue/forpliktelser basert på hittil opptjente pensjonsrettigheter

present_value_computations.con
Nåverdiberegninger av utvalgte pensjonsytelser med mere

estimate.con Beregninger/estimeringer/tabelluttak av overgangssannsynligheter og andre parametere. Husk å skru denne av i ordinære simuleringer

simulated_initial_population.con

Enkelte av parameterne som brukes når man simulerer en forenklet utgangspopulasjon med 1960 som startår, se også 'simulated_population' i 'input.con'

create_type_persons.con
Konstruerte personer for bruk i typehusholdsmodellen, ikke vedlikeholdt

Utgangspopulasjonen

Se vedlegg L for en nærmere omtale av utgangspopulasjonen.

Simulert utgangspopulasjon

Den simulerte utgangspopulasjonen er en alternativ utgangspopulasjon (se 'initial_population' i 'parameters.con'), både for å teste MOSART mot historiske data og for å ha en utgangspopulasjon uten faktiske data (og dermed ikke være bundet til linux-serverne i Statistisk sentralbyrå). Dataene hentes inn ved parameteren 'simulated_initial_population' i 'input.con'. Hovedtyngden av dataene ligger her, men det er også en fil som følger simuleringen, 'simulated_initial_population.con' (se over), med noen *simuleringsparametere*. Den versjonen som er tilgjengelig her ligger på '/mosart/input/v75/population/simulated_population_1960', og innlesingen skjer i 'read_simulated_population.cs'.

base_year	Basisår for utgangspopulasjonen
population	Antall personer etter kjønn og alder
marital_status	Andelen i ulike ekteskapelige statuser etter kjønn og alder
marital_age	Aldersfordeling for inngåelse av ekteskap
children	Kumulert fruktbarhet for kvinner etter kjønn og fødselsår
education	Andelen med ulike utdanninger i 1971 etter kjønn og fødselsår

Øvrige tema

Øvrig input til simuleringen ligger i denne versjonen på '/mosart/input/v75/*', med en katalog for hvert tema. Dette vil omfatte blant annet overgangssannsynligheter, tidsserier som MOSART skal kalibrers mot og pensjonsregler. Katalognavnene skal være relativt selvforklarende, det samme gjelder også til en viss grad filnavnene i hver katalog. Dette er filer som er i stadig utvikling og har et omfang som vi ikke prøver å dokumentere her. Parameterfilen 'input.con' vil være en nøkkel til hvordan disse filene leses inn av simuleringsmodellen, hvor første kolonne angir et navn i kildekoden, og andre kolonne navnet på fila som leses inn. Underkatalogen 'population' er et unntak her, da den inneholder utgangspopulasjonen som leses inn på en annen måte. Underkatalogene 'labour_income_level2' og 'labour_income_level3' er utviklingsprosjekter som ikke er fullført.

Vedlegg E: Simuleringsparametere

En simulering med MOSART kan styres gjennom en rekke parametere som inngår i styrefilene i brukergrensesnittet ('*.con'). Nedenfor følger en oversikt over noen av de parameterne som inngår i 'parameters.con'. Første kolonne i tabellen nedenfor gjengir parameternavnet slik det inngår som styrevariabel. Andre kolonne av styrefilen ('*.con') inneholder parameterens tallverdi. Noen parametere kan ha flere kolonner med parametere. For boolske variabler vil tallverdi større enn null, for eksempel én, normalt angi at handlingen skal eksekveres. For årstall vil tallverdier mindre enn null normalt settes til evigheten.[30] Kun noen utvalgte parametere i pensjonssystemet er omtalt her, det vises til egen dokumentasjon av disse sidene av modellen. De øvrige styrefilene vil ha en lignende oppbygning som 'parameters.con', men noen av disse kan anta forskjellige verdier i ulike pensjonssystemer. Styrefilen 'input.con' inneholder en (delvis) filreferanse i andre kolonne, 'output*.con' har en annen oppbygning.

Generelle forhold

sim_end — Sluttåret for simuleringen. Merk at startåret for simuleringen ikke er en parameter, da denne er uvilkårlig knyttet til utgangspopulasjonen

partial_start_year_*navn*
Angir at simuleringen skal starte i et tidligere år for dette temaet (*navn*). Merk at dette også utløser et tidligere startår for andre temaer som kommer lenger ned på lista

no_of_pension_systems
Bestemmer antall pensjonssystemer som skal simuleres

Utgangspopulasjon (parameters.con)

no_of_model_population_files
Antall modellpopulasjoner som skal skrives ut. Ved mer enn én modellpopulasjon må 'model_population_file.con' kopieres/redigeres og gis navn 'model_population_file_*i*.con', hvor '*i*' angir populasjonsnummer ($i \geq 2$)

no_of_extra_model_population_files
Antall ekstra populasjoner som skal skrives ut mellom hver simulering av hvert kjennetegn. Krever endringer i kildekoden, søk opp 'g.write_extra_initial_population(1)' i 'model.cs'. Ekstra populasjoner kan legges inn hvor som helst *mellom* moduler, men det øker kjøretiden betraktelig. Dette brukes blant annet ved tunge feilsøk knyttet til feil i trekkemetodene.

Utgangspopulasjon

initial_population — Valg av type utgangspopulasjon,

0: ingen utgangspopulasjon,
1: ordinær utgangspopulasjon, jamfør vedlegg L,
2: simulert utgangspopulasjon,
3: typehushold, bruker da 'type_persons' som utgangspopulasjon, samt at 'sim_end' blir det eneste året som blir simulert

[30] I MOSART er dette variabelen 'eternity' i 'global.cs', definert som 'sim_end+max_age+1',

sample_size — Ved simulert utgangspopulasjon bestemmer denne størrelsen på utvalget, med ordinær utgangspopulasjon bestemmes utvalgsstørrelsen av det utvalget som er valgt

select_cohorts
selected_cohorts — 'select_cohorts' gjør det mulig å avgrense simuleringen til utvalgte årskull, 'selected_cohorts' er en liste med de årskullene som skal simuleres

largest_cohort — Antatt største kohort i historiske data, gir en mer *effektiv* innlesing av utgangspopulasjonen da tilstrekkelig plass for hver List i kohortlistene kan angis på forhånd

Repeterte simuleringer

no_of_repeated_simulations
Brukes av 'jobr.exe' til å bestemme antall ganger en simulering skal repeteres

no_of_parallel_repeated_simulations
Lar 'jobr.exe' eksekvere repeterte simuleringer i parallell. Settes denne til en tallverdi større enn én bør man vurdere å redusere antall tråder i 'no_of_threads' og 'maxdegreeofparallelism'

Tekniske parametere

no_of_threads — Antall tråder som skal benyttes ved multithreading, 0: Ingen ekstra tråder, 1+: Simuleringen eksekveres på antall angitte tråder, mens 'control_households_after_demography' og utskrift av tabeller legges ut som ekstra tråder i bakgrunnen

no_of_threads_control_demographic_data
Tilsvarende kontroll av kontrollrutinen for demografiske data som kjøres hvert år som en bakgrunnsjobb når simuleringen av demografiske data er ferdige det året

maxdegreeofparallelism
Antall tråder som benyttes i alternativ metode for multithreading (Parallell), -1: Ingen grense, 1+: Simuleringen eksekveres på (inntil) angitt antall tråder

recycle_objects — Bruker resirkulering av objekter, reduserer kjøretiden betraktelig og øker stabiliteten

limited_simulation — Valg av type simulering

0: Ordinær simulering, ingen begrensninger
1: read_only_parameters. Modellen leser kun inn overgangssannsynligheter og andre parametere, samt setter opp tabeller med videre, for uttesting av innlesings- og utskrivingsmetodene
2: Ingen simulering, skriver ut historiske år

largest_cohort — Største forventede kohort etter kjønn og alder, utvidelse av kohortlister krever mye regneressurser

Trekkemetoder (parameters.con)

random_seed — Startverdi for randomgenerator. Hvis 'random_seed' > 0 brukes den angitte verdien, men denne må være mindre enn 2.147.483.647. Hvis 'random_seed' er mindre eller lik 0 brukes tidsangivelsen omregnet til et heltall (som foran), og dette kan oppfattes som en tilnærmet tilfeldig startverdi. Settes verdien til 0, så vil simuleringen ved første eksekvering skrive inn i 'parameters.con' den random-seeden som er brukt. Settes verdien lavere enn 0, så endres ikke 'parameters.con'. Forøvrig logges startverdien for randomgeneratoren alltid til filen 'sim.information', se vedlegg F

random_seed_change_year
random_seed_changed_value — Random-seed settes på nytt ved starten til dette året til angitt/vilkårlig verdi. Dette skaper et bruddpunkt i simuleringen, hvor flere simuleringer (med identiske forutsetninger) kan ha et felles forløp opp til bruddpunktet, og deretter ulike forløp frembrakt av simuleringsstøy alene

random_seed_z_mortality
draw_cumulative_mortality — Alternativ metode for å trekke dødelighet som brukes av repeterte simuleringer, ellers av liten interesse

Stratified_binomial — Valg av trekkemetode for binomiske begivenheter, se avsnitt 4.3

stratified_multinomial — Valg av trekkemetode for multinomiske begivenheter, se avsnitt 4.6

stratified_homogeneous — Valg av trekkemetode for multinomiske begivenheter med homogene sannsynligheter, se avsnitt 4.6

stratified_continuous — Variansreduserende trekkemetoder for uniforme fordelinger skal benyttes, se avsnitt 4.7

stratified_restore — Nullstill alle parametere i trekkemetodene ved inngangen til nytt år

homogeneous_subjects — Trekkemetode spesifikk for valg av fagfelt

override_*identitet* — For multinomiske trekninger kan man overstyre det generelle valget gitt ved 'stratified_multi' for en gruppe av trekninger, ved å bruke roten av gruppens '*identitet*', og deretter angi hvilken trekkemetode dette objektet skal bruke

Generelt om øvrige parametere

exogenous_*navn* — Angir siste år for kalibrering mot en ekstern skranke

adjustment_*navn* — Angir en justering av samme kjennetegn etter dette året, typisk en delta-verdi

Vedlegg F: Resultatfiler

En simulering/kjøring/jobb med modellen MOSART produserer mange ulike filer som samles på den katalogen som simuleringen er eksekvert fra. Nedenfor følger en oversikt over de filene som kan produseres av eller er spesifikt tilknyttet en simulering. De fleste resultatfilene kan velges bort/inn ved hjelp av styrefilene for valg av resultatfiler ('output.con', 'output.*.con' og 'model_population_file.con'). Styrevariabelen er eventuelt angitt i overskriftene til hver enkelt gruppe av resultatfiler, eller under hver enkelt resultatfil. Parameteren 'protect_output' i styrefilen for valg av resultatfiler ('output.con') gir samtlige filer og resultatkatalogen skrivebeskyttelse hvis simuleringen fullføres på normal måte med et utvalg på minst én prosent.

Tabeller med aggregerte tall ('r*_*.prn') er laget for å kunne lastes inn i regneark på en overkommelig måte (for eksempel Excel). De fleste av disse tabellene er bygd opp med kjønn som sideinndeling, en av variablene år, fødselsår eller alder som forspalte og diverse aggregerte opplysninger som hodespalte. Tabellen 'extract.prn' har et noe annet format og er et utdrag av utvalgte år for sentrale variabler i pensjonssystemet. Egendokumentasjonen og tabellene skrives ut fortløpende fra simuleringen. Tabeller som beregnes for ulike pensjonssystemer gis nummering 'r*_*_ps*i*.prn'. Ikke alle tabeller blir produsert, da enkelte tabeller avhenger av at det er alternative pensjonssystemer og/eller pensjonsreform.

Vedlegg G går nærmere gjennom hvordan tabeller kan tas ut ved hjelp av 'output.con' og 'output.*.con'. Tilsvarende går vedlegg H gjennom produksjon av modellpopulasjonen ved hjelp av 'model_population_file.con'.

Emner

- Sikkerhetskopi
- Brukergrensesnitt, se tilsvarende punkt i vedlegg D
- Egendokumentasjon
- Modellpopulasjon
- Modellpopulasjon for repeterte simuleringer
- Utdragstabell
- Ordinære tabeller med aggregerte tall

Sikkerhetskopi

backup — Underkatalog for sikkerhetskopi av brukergrensesnittet (job.exe, *.con) og 'stopfile'. Brukergrensesnittet kopieres i det simuleringen starter, 'stopfile' kopieres både når den er nyopprettet og når simuleringen eventuelt avsluttes på normal måte. I tillegg flyttes 'sim.errors' hit idet simuleringen eventuelt startes på nytt

backup/backup_*dato_angivelse* — Ved omkjøring blir alle resultatfilene og sikkerhetskopien av brukergrensesnittet, 'stopfile' og 'sim.errors' *flyttet* ned i en ny underkatalog, hvor '*dato-angivelsen*' i navnet refererer til når simuleringen ble startet. Disse kopiene kan lett slettes med 'delete_backup', jamfør vedlegg C

Egendokumentasjon

execution.log — Hver gang simuleringen er startet og eventuelt avsluttet, så logges disse opplysningene normalt til 'execution.log'

sim.control	Informasjon som skrives ut fortløpende fra simuleringen med opplysninger om hvor langt simuleringen har kommet, kjøretid, forbruk av CPU-tid og internminne, samt enkelte andre nøkkelparametere. 'sim.control' er en nøkkelfil som brukes av mange av shellscpriptene i vedlegg C for å kartlegge status for simuleringen, og bør av den grunn ikke endres. Av den grunn opprettes 'sim.control' tidlig i startfasen av simuleringen, og det er også årsaken til at 'sim.control' sikkerhetskopieres før de andre resultatfilene, slik at 'sim.control' kan opprettes tidlig
sim.report.*navn*	Dokumentasjon av ulike sider ved simuleringen
sim.report.summarize_sim_control	Aggregert informasjon av modulene som inngår i sim.control, skrevet ut etter at simuleringen er ferdig
sim.report.memory	Rapporterer bruk av internminne slik Linux registrerer det. NB: Tidkrevende tabell, bør normalt ikke brukes
sim.documentation	Tekstfil med referanse til hvilken simulering brukergrensesnittet er hentet fra. Skapes av 'newjob', egnet sted for å legge inn tilleggskommentarer manuelt
sim.errors	Når man eksekverer modellen via 'job.exe' vil 'sim.errors' fange opp alle meldinger som skrives til skjerm (standard output) av simuleringen og alle feilmeldinger fra Linux vedrørende manglende filer og problemer med simuleringen. Filen er viktig fordi den bidrar til arkivering av feilmeldinger som ellers bare ville blafret forbi på skjermen, og spesielt ville feilmeldinger fra simuleringer som går videre etter at man har logget seg ut gått fullstendig tapt. Øvrige filer av typen 'sim.errors.*' rapporterer irregulariteter i simuleringen som aksepteres. Ved omkjøring kopieres 'sim.errors' først til underkatalogen 'backup', og flyttes så videre med resten av filene til 'backup/backup_*dato*'
sim.errors.*navn*	Øvrige feilmeldinger etter tema
sim.information	Inneholder viktig nøkkelinformasjon om simuleringen, spesielt kan nevnes anvendt startverdi for random-generator og forhold knyttet til tidsforbruk
sim.report.input	Dokumentasjon av filer som er benyttet i simuleringen, med dato/klokkeslett for siste endring i filene på det tidspunktet filene ble benyttet
stopfile	Fil som etableres i det simuleringen starter som forhindrer at simuleringen kan startes på nytt før simuleringen er avsluttet. Avsluttes simuleringen på normal måte flyttes 'stopfile' til underkatalogen 'backup', i motsatt fall blir den liggende. Første linje angir starttidspunkt, deretter følger separate linjer med eier, server, modellnavn, pid-nummer, C#-tid og Linux-tid. Hvis simuleringen avsluttes på normal måte av simuleringsmodellen, eller hvis 'job.exe' fullføres, vil siste linje angi når og hvordan simuleringen stanset. 'pid-nummer' er et identifikasjonsnummer som tildeles jobben, og som også brukes av blant annet i 'ps -alf' og 'top' (jobbovervåkning) og 'kill'

(stanse jobben). Merk at 'pid-nummeret' i 'stopfile' i sjeldne tilfeller kan være forvekslet med en annen jobb

unix_*dato* Temporær fil som brukes til å formidle informasjon fra Linux til simuleringsprogammet. Slettes fortløpende etter bruk

sim.*kategori*.*navn* Gruppe av tabeller som gjennomgående telles opp i og skrives ut fra modulene for simulering, styres av parametere, ofte med *report* i navnet, i filen 'output.documentation_and_special_tables.con'

sim.adjust.*navn* Justering av overgangssannsynligheter med videre

sim.table.*navn* Tabeller som skrives ut fra simuleringen

sim.estimate.*navn* Estimering/beregning/tabulering av overgangssannsynligheter, tidsserier og andre parametere som brukes av simuleringen. Se 'estimate.con', bør kjøres som separat simulering

sim.acturial.*navn* Tabeller med delingstall, forholdstall med videre

sim.draw.*navn* Rapportering fra trekkemetodene

sim.pension_debt.*navn*
Tabeller med nåverdien av framtidige utbetalinger av hittil opptjente pensjonsrettigheter. Styres ved 'pension_debt' i 'output.con'. Merk at tabellene krever mye regneressurser, og at simuleringen må fortsette i 113 år utover siste år i tabellen (maksimal levealder minus laveste alder for pensjonsopptjening)

Modellpopulasjonen (styres gjennom 'model_population_file.con', se vedlegg H)

population En tekstfil med person × år som enhet, med utvalgte personer, år og variabler. Ved oppsplitting på flere filer vil de få 'population_' som grunnstamme, etterfulgt av ulike angivelser (år, type og ekstra utskrifter)

population.record Feltbeskrivelse for hver linje i filen '*population*'

population.sas Et oppsett for å bruke sas til å ta ut tabeller fra filen '*population*'

Tabell med nøkkelinformasjon for pensjonssystemet

extract.prn Nøkkeltall for pensjonssystemet for utvalgte år, styres av 'extract' og 'extract_years' i 'output.documentation_and_special_tables.con'

Ordinære tabeller med aggregerte tall

r_*navn*.prn Standardtabeller med en kort dokumentasjon og forklaring på toppen, sideinndeling etter alle-menn-kvinner, år i forspalten, og utvalgte variabler i hodespalten. Tabellene er tekstformat, med rette kolonner, en eller flere blanke som kolonneseparator og anførselstegn (") som tekstkvalifikatorer. Dette gjør det relativt enkelt å laste inn tabellene som tekst i for eksempel Excel

rl_*navn*.prn

rb_*navn*.prn	Tilsvarende filer med alder i forspalten. 'rl_*navn*.prn' omfatter en nærmere valgt gruppe av årskull, og tabellen viser dermed livsforløpet til disse årskullene (**l**ifecycle, **b**ase_year). 'rb_*navn*.prn' gir tilsvarende opplysninger for et tverrsnitt av befolkningen i basisåret for simuleringen
rc_*navn*.prn	Tilsvarende tabeller med fødselsår i forspalten, med opplysninger om hvert årskull/hver gruppe av årskull (**c**ohort) ved en angitt alder
rp_*navn*.prn	Tabeller med detaljerte opplysninger om (en angitt gruppe) pensjonister og deres pensjonsytelser i hodespalten
rd_*navn*.prn	Tabeller med tall for fordeling (**d**istribution) av en angitt regulær variabel i hodespalten
ry_*navn*.prn	Tabeller med år (**y**ear) som sideinndeling, og med annen valgfri forspalte

I tillegg er det filer som retrospektive, men disse er kompliserte å bruke, med lav grad av kvalitetssikring på det som kommer ut. Her anbefales det heller å skrive ut en modellpopulasjon, hvor man kan hente ut slike tall for eksempel via Stata, med større grad av kontroll og fleksibilitet.

Vedlegg G: Tabelluttak

Av hensyn til kjøretider og visse andre hensyn til effektivitet er det hensiktsmessig å ha et tabelluttak integrert i simuleringsmodellen slik at aggregerte tall kommer ut samtidig med at simuleringen er ferdig. Tabellproduksjonen er i størst mulig grad forsøkt holdt isolert fra resten av simuleringsmodellen. MOSART har et tabelluttak som er relativt fleksibelt og lett å vedlikeholde og hvor nye tabeller i stor grad kan spesifiseres fra brukergrensesnittet ('output.con' og 'output.*.con'). Dette går noe utover kjøretidene, men denne kostnaden er her vurdert som mindre viktig siden man nå med en relativt beskjeden arbeidsinnsats kan få ut tabeller etter behov uten å kunne programmering. Det er fortsatt en betydelig terskel for å lære seg å bruke tabellprogrammet.

Merk at tabellprogrammet hvert år går gjennom alle personene, og teller opp disse personene ut fra sine kjennetegn til enhver tid. Det legger skranker på hva som kan telles opp og hvordan utskriften vil bli. Dette gjelder blant annet prospektive tabeller, det vil si hvordan det går *framover* med en gitt gruppe personer (retrospektive tabeller var et forsøk på å få til dette).

Det er to alternative tilnærminger til det tabellprogrammet som ligger inne. Man kan hardkode hver enkelt tabell, det krever gjennomgående mindre regnekraft (kjøretid) og har større fleksibilitet. Det krever imidlertid gode programmeringsferdigheter, tar (mye) lenger tid å utvikle og vil etterlate seg enorme mengder kildekode. Vi har forlatt dette som et hovedspor, men bruker det for enkelte faste tabeller (jamfør 'sim.table.*navn*'). Det andre alternativet er å skrive ut relevante personer × kjennetegn, og ta ut tabellene i for eksempel Stata. For tabeller man vet bare skal brukes en gang eller for en konkret analyse er dette et godt alternativ. For tabeller som skal brukes jevnlig over lang tid er det derimot svært uheldig.

Emner

- Dokumentasjon
- Styrefiler for uttak av tabeller og egendokumentasjon
- Uttak av tabeller og egendokumentasjon
- Egendokumentasjon og spesielle tabeller
- Format med mere
- Ulike typer tabeller
- Enkelttabeller
- Svakheter ved parallellitet
- Svakhet: Navnsetting av tabeller
- Bibliotek av tabeller
- Vedlikehold av tabellprogram
- Livsløpsfil

Dokumentasjon

Med hver simulering følger det med noen filer som også dokumenterer bruken av disse filene.

output.doc	Omtaler kort hvordan man lager tabeller
output.variables	En liste med alle variable (uttak av personkjennetegn) som inngår i tabellprogrammet. Denne vedlikeholdes automatisk, i den forstand at 'komp' produserer denne filen fra 'table_base.cs' i det programmet blir kompilert, og følger med alle simuleringer som blir avledet fra denne NB! Bytter man modellversjon manuelt i en simulering, må man passe på å bytte ut denne fila med den relevante

Styrefiler for uttak av tabeller og egendokumentasjon (output.con, output.*.con)

Produksjonen av tabeller og egendokumentasjon styres gjennom filene 'output.con' og 'output.*.con'. Tidligere var 'output.con' én fil, men har over tid blitt så stor og heterogen at en oppdeling var nødvendig. Behov for å modulisere tabelluttaket taler også for en oppdeling, hvor oppdelingen gjør det mulig å ha ulike bibliotek av tabelluttak som lett kan kobles inn og ut fra hovedfilen ('output.con'). I simuleringen blir alle filene av typen 'output.*.con' som er referert i 'output.con' slått sammen med 'output.con' under innlesingen, og lest inn som om det er én fil.

Uttak av tabeller og egendokumentasjon (output.con)

Produksjonen av samtlige tabeller og egendokumentasjonen styres fra 'output.con'. Fila er redusert i omfang, og omfatter noen felles parametere for innholdet i tabellene, samt hovedbrytere (s:*i*) for de ulike tabellmodulene og referanser til andre filer med tabellangivelser (output.*.con). Ordinære parametere i 'output.con' vil ha en linje med 2-3 kolonner, i første kolonne et parameternavn/styrevariabel, i andre kolonne en parameterverdi og deretter en eventuell kommentar. Parametere som inngår:

protect_output	Gir skrivebeskyttelse til katalogen og resultatfilene
navn-på-gruppe	Hovedbrytere som angir om en gruppe tabeller skal med i tabelluttaket, hvor hver *navn-på-gruppe* er omtalt lenger ned
external_parameters	Disse linjene angir filer med andre parametere og/eller oppsett for (en eller) flere enkelttabeller. Parameterverdien er mellomnavnet i 'output.*.con', det vil si 'output.*parameter-verdi*.con'. Noen av disse eksterne modulene er obligatoriske, det er i tilfelle angitt under. Ved å skrive en '*' (eller visuelt bedre, '* ') på begynnelsen av linja, utelates disse eksterne parameterne. De fleste av disse filene hører sammen med en *navn-på-gruppe*, men det er ingen hindringer for at en slik fil kan inneholde flere typer tabeller

Hovedbrytere er parametere som utelater grupper av tabeller, disse linjene vil også bestå av 2-3 kolonner, men hvor andre kolonne har en parameter av formatet 's:*i*', hvor '*i*' antar verdiene 0; utelatt, 1: inkludert, 2: inkludert med mer omfattende informasjon (gjelder bare noen få tabeller). De fleste hovedbryterne styrer produksjonen av en tabell, og bare noen få har underordnete parametere. I overskriftene under er hovedbryter og standard versjon av fil med eksterne parametere angitt i parenteser.

Egendokumentasjon og spesielle tabeller

Parameterne som styrer egendokumentasjonen og spesielle tabeller er lagt til styrefilen 'output.documentation_and_special_tables.con', og denne *referansen* er obligatorisk å ha med i 'output.con'. En nullet versjon ligger på 'output.documentation_and_special_variables2.con'. De spesielle tabellene er i første rekke tabeller som produseres i og skrives ut fra modulene for simulering (sim.*), herunder justeringsalgoritmer og beregning av aktuariske parametere. I tillegg kommer to tabeller som tidligere har vært mye brukt, men som (ennå) ikke er tatt inn de fleksible tabellene:

report_extract report_extract_years	Tabell med navn 'extract.prn'', tar utdrag av noen angitte tabeller, som må være med, og med riktig navn, 'r_labour_supply_16_74.prn', 'sim.tab.contribution.rates' og 'rp_*.prn'. Modulen

'table_extract_information.cs' produserer tabellen, og man kan angi hvilke år som skal være med i tabellen

Formater med mere

Enkelte felles formater og parametere er lagt ut på egne filer

format_min_no_of_observations
format_missing_code
: Hvis det er for få observasjoner i en tabellcelle (format_min_no_of_observations), så skrives i stedet ut en kode (format_missing_code). Dette er viktig der man ellers risikerer å skape bakveisidentifikasjon av utgangspopulasjonen fordi det blir for få observasjoner i en celle, noe som lett oppstår hvis man for eksempel kombinerer ettårig alder og bostedskommune. Ligger p.t. på 'output.format.con'

format_year
format_age
: Formattering (aggregering) av henholdsvis årstall og alder, er p.t. samlet på 'output.format.con'

base_year_marker*
: En gruppe av parametere som brukes til å markere en populasjon i et basisår, og selektere på fødselsår (min-max), via personkjennetegnet 'included_in_base_year', som fanges opp av både modellpopulasjonen og tabellprogrammet. Planen er å utvide settet av seleksjonskriterier. Er p.t. samlet på 'output.base_year_markers.con'

Ulike tabelltyper

Under følger en beskrivelse av de ulike typene tabeller som kan tas ut. Lenger ned følger en nærmere felles beskrivelse av parameterne som styrer tabelluttaket, da dette i stor grad er likt for alle tabelltypene (ikke alle parametere er tilgjengelig i alle tabeller). I parentes i hver tittel står navnet på hovedbryteren i 'output.con', sammen med navnet/navnene på tabelltypen slik det angis for hver enkelt tabell i filene som hentes inn.

Standard fleksible tabeller (standard_tables; standard_table)

Standardtabellene i MOSART har sideinndeling etter alle-menn-kvinner og i forspalten en av variablene år, alder eller fødselsår. Alle øvrige variabler kommer i hodespalten. Spesielt forspalten med tid (år/alder/fødselsår) er tilpasset at MOSART er en dynamisk mikrosimuleringsmodell som simulerer livsløp og populasjonen over lang tid. Tabellene produseres av 'table_standard_tables.cs'.

Pensjonstabeller (pension_tables; pension_table, pension_table_original_benefits, pension_table_reformed_benefits)

En av de mest sentrale anvendelsene av MOSART omfatter framskrivinger og analyser av folketrygden og pensjonssystemet ellers. Modellen har derfor en egen gruppe tabeller 'rp_*.prn' som i større detalj tar ut tabeller med antall pensjonister og tilhørende gjennomsnittlig pensjonsytelser. Modulen 'table_pensions.cs' produserer tabellen. Flere av parameterne under de fleksible tabellene er gyldige her, blant annet tabellbrytere, årslister, aldersavgrensning, reg.status, labler og subset (se standard fleksible tabeller nedenfor). De to sistnevnte tabelltypene angir pensjonsytelser beregnet etter folketrygden henholdsvis med reglene før og etter pensjonsreformen av 2011.

Fordelingstabeller (distribution_tables; distribution_table)

MOSART kan produsere tabeller med tall for fordeling (rd_*.prn), med antall personer, en analysevariabel med median, gjennomsnitt, Gini-koeffisient, desilgjennomsnitt og prosentilgrenser. Hver tabell inneholder som nevnt bare en analysevariabel, men dette kan være en hvilken som helst av alle de regulære variablene. Disse tabellene krever en del regneressurser. Modulen 'table_distribution.cs' produserer tabellen. De fleste parameterne under de fleksible tabellene er gyldige her, med unntak for kategorivariablene i tabellhodet og forspalten (se standard fleksible tabeller nedenfor).

Kohorttabeller (cohort_tables; cohort_table)

En undergruppe av standardtabellene (se under) er kohorttabeller (rc_*.prn), hvor forspalten er fødselsår (eller grupper av fødselsår), og hvor hver linje er avgrenset til en bestemt alder (eller aldersgruppe). Tabellene produseres sammen med standardtabellene i 'table_standard_tables.cs'. Alle parameterne med unntak for for 'year_list' kan brukes, se under. I tillegg kommer parametere som kan gruppere sammen fødselskull, se 'output.cohort_tables.con'.

Livsløpstabeller (lifecycle_tables; lifecycle_table)

En undergruppe av standardtabellene (se nedenfor) er livsløpstabeller (rl_*.prn), hvor forspalten er alder i stedet for år, og hvor hver tabell er avgrenset til et sett påfølgende fødselskull. Hver tabell viser dermed livsløpet til en (simulert) gruppe av populasjonen, avgrenset til fødselsår og eventuelle andre kjennetegn. Til hver livsløpstabell produseres en helt tilsvarende tabell med et tverrsnitt av populasjonen i startåret for simuleringen (rb_*.prn), hvor man kan sammenlikne simuleringen med historien (for et tverrsnitt over generasjoner, ikke for et gitt fødselskull). Tabellene produseres sammen med standardtabellene i 'table_standard_tables.prn'. Alle parameterne med unntak for 'year_list' kan brukes, se nedenfor. I tillegg kommer parametere som avgrenser hvilke fødselskull som skal inngå i tabellene, hvor det kan være flere sett av fødselskull som grupperes sammen, se 'output.lifecycles_tables.con'.

Årstabeller (year_tables; year_table)

Standardtabellene i MOSART har tid i forspalten, og presser dermed mye informasjon inn i hodespalten. Noen ganger er det bruk for en mer balansert tilnærming, og vi har derfor også lagt inn et opplegg for årstabeller (ry_*.prn) med år som sideinndeling og en valgfri forspalte. Modulen 'table_year_tables.cs' produserer tabellene. Alle parameterne som inngår i de fleksible tabellene er tilgjengelige her, i tillegg er det en egen kategori for 'front_row' som er helt analog med 'head_row' (se standard fleksible tabeller nedenfor)

Tabeller for AWG (awg_tables; awg_table)

Dette er spesialtabeller med navn 'r_awg*.prn' laget for *age working group*, en arbeidsgruppe under EU/EØS som hvert tredje år lager en rapport om pensjoner og økonomiske konsekvenser av en aldrende befolkning i Europa. Det er Finansdepartementet som står for den videre bearbeidingen og teksten som sendes til AWG, men det er høyt verdsatt at denne tabellen beholder samme format. Derfor har den blitt hengende igjen som en spesialtabell. Modulen 'table_awg.cs' produserer tabellen(e). Enkelte av parameterne under de fleksible tabellene er gyldige her, blant annet tabellbrytere, årslister, aldersavgrensning og reg.status.

Retrospektive tabeller (retrospective_tables; retrospecitive_year_table + retrospective_year_tables; retrospective_year_table)

Dette er to typer tabeller som teller opp hva som har skjedd bakover i tid for en gitt gruppe personer i året man teller opp for. Tabellsettet har blitt lite brukt, og er trolig for komplisert for denne bruken.

Produksjon av enkelttabeller

Produksjonen av en tabell styres av en linje per tabell, og inneholder informasjon i to faste og ellers valgfrie kolonner. Første kolonne vil angi type tabell (som omtalt foran, men også lenger ned), og er obligatorisk. Programmet vil deretter betrakte alle faktiske linjer i fila fram til neste linje med en tabelltype i første kolonne (altså en ny linje), og lese dette inn som om det er en linje. Mange tabeller går fint inn på en linje som kan leses på skjerm, mens enkelte tabeller har så mye input at de må deles.

I andre kolonne kommer en bryter ('t:*i*), hvor '*i*' angir om tabellen skal produseres (i=1) eller ikke (i=0). De øvrige kolonnene leter opp et nøkkelord (' *navn*: ", se lenger ned), og alt fram til neste nøkkelord vil være parametere til dette nøkkelordet. Noen nøkkelord kan ha bare en parameter, mens andre kan ha flere, og til dels mange (variabellister). Disse nøkkelordene/kolonnene er valgfrie og kan komme i en vilkårlig rekkefølge, og med ingen nøkkelord (for standardtabeller) vil man for hvert år få antall bosatte personer i alt i Norge.

Nøkkelordene her er omtalt som 'table_keywords' i 'table_base.cs', i motsetning til de nøkkelordene som allerede er omtalt (herunder tabelltyper og hovedbryter) som går under navnet 'output_con_major_keywords' i 'global.cs') Nøkkelordene omfatter blant annet manuelt angitt tittel, variabelliste, grupperingsvariabler, betingelser, pensjonssystem og diskonteringsrente.

title:	
full_title:	MOSART produserer et navn på hver tabell av typen 'r*t_navn-etter-innhold*.prn'. På enkle tabeller fungerer dette ofte veldig godt, men på sammensatte tabeller kan navnet bli urimelig langt og/eller ikke fange opp innholdet på noen god måte. Man kan da bruke ett av alternativene 'title:' eller 'full_title:', og angi et eget navn etter ønske, hvor den første i større grad hektes på annen grunnleggende informasjon om innholdet (betingelsesvariabler blant annet). Ved for lange navn blir filnavnet trunkert, med oppfordring i 'sim.errors' om å bruke 'full_title:'. Ved navnelikhet for filnavnet legges det til et appendiks av typen '_ex_*nummer*'
table_explanation:	Legger til denne kommentaren i tabellhodet
reg_status:	Angir hvilke kategorier av reg.status som skal telles opp. Utelates parameteren får man alle bosatte (resident), men ulike kombinasjoner av bosatte, døde og utvandrete kan brukes (resident, resident_and_abroad, resident_and_dead, abroad, abroad_and_dead, dead and all_reg_statuses, se også 'output.doc')
min_age:	Minste alder som skal telles opp, default er 0 år
max_age:	Høyeste alder for de skal telles opp, -1 angir at det er ingen øvre aldersgrense, som også er default

subset:	Øvrige betingelsesvariabler (i tillegg til reg.status og alder). Her er det stor fleksibilitet, i prinsippet er alle variabler i tabellprogrammet tilgjengelig, og de kan kombineres, se 'output.doc'
year_list:	Angir hvilke år man skal telle opp, default er alle år. Man angir her navnet på en årsliste som på forhånd er definert i 'output.formats.con'
group_cohorts:	Brukes av kohorttabellene for å gruppere kohorter, etter et lignende mønster som 'year_list:', se 'output.cohort.con' for detaljer
head_row:	Angir hvilke variabler som tabellen skal kryssgrupperes etter i hodespalten
front_row:	I årstabeller angir denne de variable som tabellen skal kryssgrupperes etter i forspalten
pension_systems:	Angi hvilket pensjonssystem som skal skrives ut, med en fil for hvert valgt pensjonssystem. Med 'all_pension_systems' får man alle pensjonssystemer, se for øvrig 'output.doc'. I 'pension_tables' og 'awg_tables' blir alle pensjonssystemer automatisk skrevet ut
interest_rate_numbers:	Velg hvilken nettorente som skal skrives ut etter lignende format som for pensjonssystemer, se 'present_value_computations.con' og 'output.doc'
variable_list: extra_variables:	Angir hvilke variabler det skal rapporteres tall for, enten i form av antall personer, andeler av populasjonen eller sum eller gjennomsnitt av regulære variabler. Ekstravariabler blir bare rapportert for kategorien 'alle' i hodespalten

Variabler og variabeltyper

Det finnes tre typer variabler i tabelluttaket:

regulære variabler	Dette er variabler som angir størrelsen på noe, og hvor det er meningsfylt å beregne for eksempel gjennomsnitt. Lønnsinntekt kan være et godt eksempel.
kategorivariabler	Dette er variabler med et endelig antall tilstander, men fortrinnsvis flere enn to, og hvor tallverdien som er tilordnet hver tilstand nødvendigvis ikke har noen numerisk betydning. Pensjonsstatus kan være et godt eksempel.
boolske variabler	Dette er variabler som antar verdien sann eller usann, og disse er det mange av i simuleringsmodellen MOSART.

I utgangspunktet så er regulære variabler tilordnet 'variable_list:' og 'extra_variables:', kategorivariabler er tilordnet 'head_row:' og 'front_row:', mens boolske variabler er tilordnet 'subset:'. De ulike variablene kan likevel brukes om hverandre, om enn med noen hjelpevariabler.

Boolske variabler kan uten videre brukes som regulære variabler (da som 0-1) og kategorivariabler, og kan også settes sammen til sammensatte boolske variabler. Regulære variabler kan brukes som kategorivariabler (via 'var') og som boolske variabler (via 'has', 'equal', 'below', 'above'). Kategorivariabler kan brukes som boolske variabler (via 'equal'). Noen regulære variabler angir enten år for begivenhet, alder ved begivenhet eller varighet siden begivenhet, disse kan

reformateres til å angi en av de andre mulighetene (AsAge, AsYear, AsTime), og spesielt kan alle brukes til å identifisere om begivenheten har skjedd i inneværende år via 'equal_current_age'. Noen få variabler er av praktiske grunner definert som både regulær variabel og kategorivariabel (alder).

Filen 'output.variables' (som følger hver simulering) angir hvilke variabler som inngår og hva slags type denne er.

Opsjoner og spesielle variabler

Under omtales noen opsjoner og noen spesielle variabler

variabel-navn:a — For regulære variabler, ta gjennomsnitt

variabel-navn:not — For boolske variabler, reverser hva som er sann

variabel-navn:ps=*i* — For variabler med pensjonssystem, overstyr valg av pensjonssystem til pensjonssystem '*i*'

var=*største-verdi*:*variabel-navn*:den=*tall*:min=*tall*

Regulære variabler kan representeres som kategorivariabler ved variabeltypen 'var=', hvor '*største-verdi*' angir hvor langt opp man skal gruppere, '*variabel-navn*' angir en regulær variabel, og de to siste valgfrie opsjonene angir henholdsvis hvor stor intervaller hver gruppe skal ha ('den', denominator, 1 er default) og eventuell minsteverdi ('min', 0 er default)

equal=*verdi*:*variabel-navn*

Regulære variabler og kategorivariabler kan representeres som boolske variabler ved variabeltypen 'equal='

has:*variabel-navn*
below=*verdi*:*variabel-navn*
below_or_equal=*verdi*:*variabel-navn*
above=*verdi*:*variabel-navn*
above_or_equal=*verdi*:*variabel-navn*

Regulære variabler kan representeres som boolske variabler ved variabeltypene som er listet opp over.

variabel-navn=AsAge
variabel-navn=AsYear
variabel-navn=AsTime

Enkelte variabler angir enten alder når, årstall når eller varighet siden en begivenhet inntraff. Disse variablene kan angis som en hvilken som av disse tre typene (banal omregning), og for disse variablene skjer dette ved å velge en av tilleggsparameterne som nevnt foran (AsAge, AsYear, AsTime). Variabler merket med 'D' (date) i 'output.variables' er slike dato-variabler

equal_current_age:*variabel-navn*=AsAge

Dato-variabler kan representeres som boolske variabler ved variabeltypen 'equal_current_age', som er sann hvis '*variabel-navn*' har samme verdi som alder på samme tidspunkt

variabel-navn:AtAge=*x*
variabel-navn:AtYear=*x*

variabel-navn:AtTime=*x*

Enkelte variabler kan man hente fram verdier for på et tidligere tidspunkt ved å legge til en av parameterne 'AtAge=*x*', 'AtYear=*x*' eller 'AtTime=*x*', hvor '*x*' angir henholdsvis alder, år eller varighet siden. Disse variablene er merket med 'L' (longitudinal) i 'output.variables'

Sammensatte boolske variabler

Under 'subset:' vil boolske variabler adskilt av et (eller flere) mellomrom bli håndtert som separate variabler bundet sammen av 'og' (det vil si at alle variablene må være oppfylt for kravet som helhet skal være oppfylt). Ellers (også under 'head_row:', 'front_row:', 'variables:' og 'extra_variables:') kan boolske variabler hektes sammen av vanlige parenteser, '(' og ')', samt tegn for 'og', '&', og 'eller', '|'. Blanke kan ikke inngå.

Svakheter ved parallellitet

Opptellingen i tabelluttaket skjer på slutten av året, etter at simuleringen er avsluttet for inneværende år, og før simuleringen går videre til neste år. Bruker man parallellitet, så skjer opptellingen ved hjelp av multithreading, med hver enkelt tabell som arbeidsenhet. Det er en svakhet her, at når *garbage collector* aktiveres mens tabelluttaket går, så kan det komme feil i opptellingen, ved at verdien for den ene variabelen som skulle vært hentet inn på det tidspunktet blir satt til default (0, falsk) uavhengig av dens faktiske verdi. Dette skjer ikke andre steder i simuleringen, og hadde det skjedd ville det utløst et skred av etterfølgende endringer, det skjer ikke her. Med simulering på store utvalg så blir dette dermed en skjønnhetsfeil, i noen sammenhenger irriterende da man ofte vil ha *eksakt* likhet, i andre sammenhenger bekymringsfullt fordi hvis dette skjer i simuleringen, så kan vi ikke bruke multithreading.

Ved parallellitet vil utskriften av tabellen skje i bakgrunnen, slik at simuleringen kan fortsette.

Bibliotek av tabeller

Det er ikke helt enkelt å skape nye tabeller, med mindre man har en lignende tabell å ta utgangspunkt i, og feilmeldingene kan være kryptiske i slike tilfeller. Dette gjør det litt vanskelig å vrake oppsettet til en tabell man har tatt ut (men på langt nær så vanskelig som å vrake en full tabellmodul skrevet i C#). Samtidig vil det føre til at det samler seg opp mange tabeller med perifer interesse. Moduliseringen av 'output.con' åpner for at slike tabeller kan samles på egne filer (bibliotek), som kan hentes inn senere ved å inkludere en linje med referanse til fila i 'output.con'. Vi er i starten av dette arbeidet. Merk at mellomnavnet i 'output.*.con' kan bestå av mange ledd, som åpner opp for systematikk i navnsettingen.

Vedlikehold av tabelluttak

Vedlikeholdet av tabelluttaket vil bestå i å legge til nye variabler i 'table_base.cs' *hver gang* man legger til eller endrer et personkjennetegn. Dette gjøres enklest ved å følge gangen til en lignende variabel som allerede ligger inne i systemet, og aller helst en variabel som er mest mulig lik. I enten 'find_regular_variable', 'find_category_variable' eller 'find_criterion' må man legge til to påfølgende linjer som først tester om denne teksten refererer til denne variabelen, og deretter oppretter en slik variabel (det vil si et objekt av denne klassen). Lenger ut i modulen opprettes klassen med dette navnet, og det som er spesifikt til klassen er hvilket tall som skal rapporteres, og eventuelle tilpasninger av forkortelser, navn med videre slik de skal inngå i tabellene. Boks I.1 gir strukturen for kildekoden. Shellscriptet 'generate_output_variables' som kalles av både 'komp', vedlikeholder dokumentasjonsfilen 'output.variables'. Legger man til en kommentar på slutten av linja som deklarerer klassen for den nye variabelen, så kommer denne kommentaren med i 'output.variables'.

Boks I.1. Vedlikehold av tabelluttak

```
// Ny regulær variabel
else if (same_root(base_parameters.label, "navn-på-ny-variabel"))
        new t_regular_variable_navn-på-ny-variabel(base_parameters);

// Klasse for ny regulær variabel
public class t_regular_variable_navn-på-ny-variabel : base t_base_regular_variable
{
        public t_regular_variable_navn-på-ny-variabel
                (t_base_parameters base_parameters) : base base_parameters
        {
                long_label = "fullt navn";
                short_labels.Add("kort-navn-max-åtte-tegn");
                eventuelle-andre-instruksjoner;
                add_labels(base_parameters);
        }

        public override double x(person p)
        {
                return p.relevant-kjennetegn;
        }

        public override eunum_longitudinal_information longitudinal_information ()
        {
                return enum_longitudinal_information.no;
        }
}
```

Vedlegg H: Modellpopulasjon

En måte å ta ut resultater fra en simulering med modellen MOSART er å produsere modell-populasjoner som filer, og deretter produsere tabeller og/eller ta ut livshistorier fra disse filene. Nedenfor følger en kort beskrivelse av modulen som produserer modellpopulasjoner og hvordan man begrenser produksjonen av en modellpopulasjon gjennom styrefilen 'model_population_file.con' (model_population_file_*i*.con). En modellpopulasjon vil bestå av filen 'population' (population_*i*) som er en tekstfil med en linje for hver person × år med utvalgte personer, år og variabler. I tillegg produseres to filer med henholdsvis feltbeskrivelse (population.record, population_*i*.record) og standard oppsett for tabellproduksjon i sas (population.sas, population_*i*.sas). Om angitt, vil modellpopulasjonen sorteres på år og person etter at simuleringen er avsluttet.

Fra 'parameters.con' er det en egen parameter 'no_of_model_population_files' som angir hvor mange modellpopulasjoner man vil skrive ut. Produksjonen av hver modellpopulasjon styres gjennom enten 'model_population_file.con' (populasjon 1) eller 'model_population_file_*i*.con' (for populasjon 2+, hvor '*i*' angir populasjonsnummer). For boolske variabler vil generelt en tallverdi være større enn null, for eksempel én, normalt gi at handlingen eksekveres. Modellpopulasjonen kan begrenses til hvilke personer, hvilke år og hvilke variabler man ønsker å ta med.

Det er mulig å skrive ut modellpopulasjonen mellom simuleringen av hvert kjennetegn med noe modifikasjon av kildekoden. Antall ekstra populasjoner man ønsker å skrive ut spesifiseres med parameteren 'no_of_extra_model_population_files' i 'parameters.con'. Utskriften av hver av disse ekstra populasjonene legges inn i kildekoden mellom simuleringen av hvert kjennetegn i 'model.cs', der man ønsker en ekstra utskift (mellom hvert metodekall). Søk opp *tekststrengen* 'g.write_extra_model_population_files(1)' i 'model.cs,, denne er kommentert ut i standardversjonen, men kan lett hentes inn og eventuelt kopieres videre nedover (huske da å erstatte '1' med økende tallverdier nedover). Formålet er i første rekke tung feilsøking, spesielt ved feil i trekkemetodene og/eller multithreadingen.

Emner

- Nye variabler
- Generelle parametere
- Valg av personer
- Valg av år
- Relaterte personer
- Valg av variabler

Nye variabler

Etter hvert som modellen utvides legges det til nye personkjennetegn, og man må da også oppdatere modulen som produserer modellpopulasjonen. Dette gjøres enklest ved å følge gangen for et personkjennetegn av lignende type, enten det gjelder heltall, flyttall, referanser eller boolske variabler. Produksjonen av modellpopulasjonen er organisert i ett objekt (model_population_file.cs, class t_model_population_file). Nye personkjennetegn kan legges til ved å gjøre følgende tre ting:

- Deklarere en ny klasse som håndterer det angitte personkjennetegnet i 'model_population_file.cs'
- Opprette et objekt av denne klassen i 'model_population_file.cs'
- Legge kjennetegnet til i styrefilen 'model_population_file.con'

Klassen som håndterer personkjennetegnet må være en avledet fra mellomklassene for utskrift av personkjennetegn (t_base_variable_*type*, *type*: double, int, single_digit, bool), og det eneste man må deklarere i klassen er en ny versjon av metoden 'x' (override). For variabler som er i kroner, det vil si underklasser av 't_base_variable_nominal_amount', er det metoden 'x2' som skal redeklares, ikke 'x', og da etter samme mal (kompilatoren skal fange opp feil her).

Konstruktøren er felles for alle variablene og angitt i de grunnleggende klassene, og er derfor ikke med i klassedeklarasjonen i eksempelet under.

Under er et eksempel for å skrive ut en sannsynlighet avledet fra 't_base_variable_double'. Linjene i hel kursiv gjelder ikke alle variabler. Metoden 'set' definerer format på utskriften (bredde, antall desimaler) der det ikke automatisk følger av variabeltypen (bool, single_digit). Søk opp 'standard_format_' for hvilke formater som er tilgjengelig, eventuelt definer et nytt format. Er det satt av for få felter til å få plass til ønsket format på utskriften, skrives variabelen automatisk ut med et mindre plasskrevende format (færre desimaler og/eller eksponentielt), eventuelt gis en feilmelding til 'sim.errors' , og om ønskelig stoppes simuleringen (se 'stop_at_warning_write_number' i 'parameters.con').

Metoden 'information_is_available' brukes der en variabel nødvendigvis ikke er tilgjengelig for de historiske årene. Søk opp '_available = ' for grupper av tilgjengelighet. Metoden 'x' kan være av annen type enn double, og da enten int eller bool.

```
class t_navn-på-ny-variabel : t_base_variable_double
{
        public override void set() {f = mpf.standard_format_probabilities;}

        public override double x(person p)
        {
                return p.ny-variabel;
        }

        public override bool information_is_available(person p)
          {return probabilities_available(p);}
}
```

Linjen hvor det returneres en verdi kan gjøres på andre måter, det vesentlige er at 'x' er en metode med returverdi double, med 'p' som parameter, og at verdien av den relevante variabelen for person p blir returnert. I eksekveringsdelen må man legge inn en linje hvor den nye variabelen kan bli opprettet ved følgende instruksjon:

```
add_variable t_navn-på-ny-variabel(rp, variable.navn-på-tilgjengelighet);
```

Den andre parameteren ('variable') er valgfri, og utelates den går programmet utfra at variabelen er tilgjengelig hele tiden, men kun med gjeldende verdi og uten å variere over pensjonssystemer. Se 'enum variable' i 'model_population_file.cs' for hvilke tilgjengeligheter som kan brukes, dette går på om simuleringen husker bakover i tid hva variabelen har vært, og om den varierer over pensjonssystemer.

Programmet konverter type-navnet 't_*navn-på-ny-variabel*' til en tekststreng '*navn-på-ny-variabel*', og denne må legges inn på akkurat samme måte i styrefilen 'model_population_file.con'. Glemmer man

dette, dukker ikke feilen opp før man faktisk bruker modellpopulasjonen. Det kan derfor hope seg opp en del slike feil. Dette gjelder også hvis man bytter en modellversjon i en simulering, og den andre modellversjonen har andre variabler i sin modellpopulasjon.

Generelle parametere

model_population_file
: Skal modellpopulasjonen produseres, 1: Ja, 2: Bare for den første av repeterte simuleringer. Merk at man må også sette verdier for 'no_of_model_population_files' (og 'no_of_extra_model_population_files') i 'parameters.con' for å få ut modellpopulasjon(er)

sort_model_population
: 1: Modellpopulasjonen sorteres på individ og år, 2: Modellpopulasjonen sorteres på begge deler. Sorteringsvariablene må være inkludert i modellpopulasjonen

column_separator
: Angir valg av tegn som kolonneseparator, 0: ingen, 1: minst en blank, slik at filen får rette kolonner, 2: gjør at fila får et såkalt csv-format. Det første formatet gjør fila leservennlig på skjerm, det andre formatet forenkler innlesingen for blant annet Stata. Csv-formatet vil her si ingen ekstra blanke, hver kolonne separert av et komma (','), første linje inneholder variabelnavnet til sin kolonne og fila navnes om til 'population.csv'

split_by_year
: Hvert år får sin egen fil

abbreviate_variable_label
: Gjør at dokumentasjonen ('population.record', 'population.sas' og første linje i csv-formatet) bruker forkortelser for variabelnavnene (som i enkelte tilfeller er veldig lange, da de er avledet av typenavnet). Fila 'population.abbreviations' inneholder en kobling mellom forkortelsen og det fulle navnet

Valg av personer

resident
: Angir om bosatte personer skal med i modellpopulasjonen

dead
: Angir om årets døde personer skal med i modellpopulasjonen

dead_previous_years
: Angir om personer som har dødd i tidligere år skal med i modellpopulasjonen

abroad
: Angir om personer bosatt i utlandet skal med i modellpopulasjonen

all_lists
: Angir om alle personer etter kjønn og alder skal med i modellpopulasjonen, hvis ja så overstyres etterfølgende parametere

which_sex
: 0: Begge kjønn, 1: Menn, 2: Kvinner

min_age
: Laveste alder i modellpopulasjonen

max_age
: Høyeste alder i modellpopulasjonen

min_birth_year
: Eldste årskull som skal med i modellpopulasjonen, 0: Ingen nedre grense

max_birth_year Yngste årskull som skal med i modellpopulasjonen, < 0: Ingen øvre grense

select_birth_years Liste med fødselsår som skal med

Valg av personer (select_on_status)

select_on_status Ta kun med personer som tilhører samtlige av de aktiviserte kriteriene under. Er det ingen grupper med skjer ingen seleksjon, er ikke hovedbryteren aktivert skjer heller ingen seleksjon

select_pensioners
select_old_age_pensioners
select_new_old_age_pensioners
Henholdsvis pensjonister, alderspensjonister og nye alderspensjonister

select_students Studenter og elever

select_head_of_household
Ta med personer som hovedpersonen i sitt hushold. Nyttig hvis man vil bruke husholdninger som telleenhet

De fire seleksjonskriteriene under har til dels lik funksjonalitet og i stor grad (dessverre) overlappende navnsetting, men er fire uavhengige systemer

select_included_in_base_year
Ta med personer hvor 'included_in_base_year' i 'person.cs' er satt til sann, og brukes spesielt ved repeterte simuleringer, hvor man ønsker å følge populasjonen fra bruddpunktet i simuleringen (året hvor simuleringene slutter å være like på grunn av trekningene)

select_marked Ta med personer hvor 'select_this_person' i 'person.cs' er satt til sann. Dette er en hjelpevariabel i simuleringen av (stort sett) husholdningsoverganger

select_reported_persons
Ta med personer som angitt under 'report_persons' i 'parameters.con', som er en liste med personer som skal følges nøyere opp. I modellpopulasjonen kommer de automatisk, men kan også brukes til å skrive ad hoc kildekode andre steder i programmet

select_person Ta med angitt liste med personer (identifikasjonsnumre), angitt fra kolonne 3 og utover.

Valg av år

include_pre_base_year_data
Ta med år med historiske data

all_years Ta med alle år, overstyrer øvrige variabler nedenfor som velger år

interval Antall år mellom hvert år som skal skrives ut, 0 utelater denne opsjonen

first_year Første år av årene som skal skrives ut med faste mellomrom (*interval*), gitt at '*interval*' er større enn null, 0: Ingen nedre grense

last_year — Siste år av årene som skal skrives ut med faste mellomrom (*interval*), gitt at '*interval*' er større enn null, < 0: Ingen øvre grense

select_years — Ta med utvalgte år, hvert enkelt år må skrives fra tredje kolonne og utover, kan kombineres med '*interval*'/'*first_year*'

Valg av pensjonssystemer

print_pension_systems
0: Alle, 1+: Angitt system

Relaterte personer

Det er mulig å skrive ut informasjon for relaterte personer. Disse personene må først inkluderes via parameterne 'include_*relasjon*', hvor relasjonene/forkortelsene er 'mother/m', 'father/f', 'spouse/s', 'marital_spouse/ms', 'partner/p', 'co_parent/cp', 'youngest_child/y' og 'head_of_household/h'. Deretter inkluderes de ulike variablene ved å legge til parametere til hver variabel som skal med, se under.

Valg av variabler

Hver parameter angir en variabel som normalt er skrevet tilnærmet lik skrivemåten i 'person.cs'. Noen variabler inkluderes automatisk, disse er omtalt her. I tillegg omtales en del variabler som ikke er en direkte utskrift av et personkjennetegn.

all_variables — Ta med alle variabler, overstyrer øvrige valg av variabler. NB! Gir i denne modellversjonen utskrift av minst 282 variabler med 1762 felter per linje med én blank som kolonnedeler. Med flere pensjonssystemer så økes antallet variabler

Noen av variablene kan ha tilleggsparametere, hvor man etter kolonne (men før kommentaren) skriver et nøkkelord etterfulgt av parametere. Nøkkelordene kan komme i vilkårlig rekkefølge, men tilhørende parametere må komme suksessivt. Følgende tilleggsparametere er tilgjengelig, nøkkelordet i første kolonne:

rp — Relaterte personer skrives ut, se lenger opp, etterfulgt av en liste med personer som skal med (m, f, s, ms, p, cp, y og h), eventuelt kan man også legge til 'se', som gjør at personen selv ikke skrives ut for dette kjennetegnet

i — Noen variabler er en vektor (for eksempel sammensetningen av pensjonsytelser) og som default skrives da alle elementene ut. Man kan velge hvilke elementer som skal skrives ut med nøkkelordet 'i', etterfulgt av tallverdien for hvilke elementer som skal med. Det kan kreve litt fikling å finne tallverdien (C# har nullbaserte vektorer som default, men noen ganger lar vi det første elementet i vektoren stå tom). Aktuelle variabler skal være merker med 'index' i kommentarfeltet, noen ganger med gyldighetsområdet for indekseringen

d — Noen variabler huskes bakover i tid (longitudinelle), og man kan skrive ut tidligere verdier på årets record, etterfulgt av en liste av *tidspunkter* man vil ha med. Har man inkludert nøkkelordet 'd' må man huske å be om året verdi (som ellers utelates), enten ved å skrive 'c' eller '256', gjerne umiddelbart etter 'd'. Alle heltallsverdier godtas, og rapporteres med en '.' hvis tidspunktet ikke finnes for personen (er i framtiden eller før personen

	ble født/simuleringen startet. Negative verdier vil angi antall år tilbakedatering fra året som skrives ut ('-1' er da fjoråret). Verdier større en '256' angir årstall. Verdier '[0, 255>' angir alder for personen. Aktuelle variabler skal være merket med 'LV' i kommentarfeltet
ps	Denne overstyrer valg av pensjonssystemer lenger opp, og skriver ut valgte pensjonssystemer. Aktuelle variabler skal være merket med 'PSD' i kommentarfeltet
i	Alle inndataene i utgangspopulasjonen er i løpende kroner, hvor simuleringsmodellen jobber på lønnsdeflaterte størrelser. I utskriften til tabeller og modellpopulasjonen er også alt som default i lønnsdeflaterte størrelser. Med nøkkelordet 'i' kan overstyre dette og skrive ut kronebeløpene med deflatorene 'W' (lønnsdeflatert, også default), 'P' (prisdeflatert) og/eller 'N' (løpende kroner). Aktuelle variabler skal være merket med 'NA' i kommentarfeltet
AsAge AsYear AsTime	Noen variabler angir datering av en hendelse, enten som alder ved hendelsen, året for hendelsen eller varighet siden hendelsen. Defaultverdien for de ulike variablene varierer ut fra behovet i simuleringen, og noen ganger kan det være ønskelig å transformere til en annen verdi. Det er veldig tydelig for kategorivariabler i tabellprogrammet, men også nyttig i modellpopulasjonen for å øke lesbarheten. Dette gjøres ved å legge til en av de tre omtalte nøkkelordene 'AsAge', 'AsYear' ***eller*** 'AsTime'

Noen av variablene er bare tilgjengelige som historiske data, og skal her være merket med 'BYSV' i kommentarfeltet (Base Year Support Variables).

Vedlegg I: Innlesing av overgangssannsynligheter

Moduler i C# som håndterer eksterne filer blir fort lange og tunge å vedlikeholde. I tillegg ligger det også hele tiden en stor risiko for feil enten i kildekoden eller når man gjør endringer i filene med overgangssannsynlighetene. For å bøte på dette er det lagt inn en felles innlesingsklasse (read_base.cs, class t_par) i MOSART som håndterer innlesing på en noe mer effektiv måte, hvor vi har lagt stor vekt på robust innlesing og effektiv avlevering av sannsynligheter. Se slutten av vedlegget for detaljer om metodene i klassen, de er ellers referert i teksten nedenfor.

Emner

- Gjeldende fil
- Kontrollmetoder
- Tidsserier
- Oppslag i tabeller
- Kvalitetssikring
- Innlesingsklassen

Gjeldende fil (current_file)

Innlesingsklassen er bygd opp omkring to referanser til en gjeldende fil (FileStream fs, StreamReader sr), og inneholder noen generelle metoder for håndtering av denne gjeldende filen. Det er lagt inn metoder som åpner gjeldende fil, laster inn informasjonen, finner relevante linjer, henter ut parameterverdier og lukker filen. I motsetning til standard metoder i C#, så vil disse metodene være mer sammensatte (gjøre flere grunnleggende metodekall i ett felles kall) og samtidig legge sterke føringer på hva man får og ikke får lov til å gjøre. I den grad metodene som håndterer gjeldende fil avdekker irregulariteter i filnavn og/eller -innhold stoppes simuleringen med en forhåpentligvis klar feilmelding. Disse stoppene kan være irriterende i en utviklingsfase, men sparer ressurser når trivielle feil i innlesingen må avdekkes senere med andre og langt mer arbeidskrevende metoder (i verste fall blir de aldri oppdaget). Irregularitetene går på overflødig informasjon, manglende informasjon, tvetydig informasjon og feil plassering av informasjon. Nesten alt dette er feil som C# aksepterer og som kan være svært vanskelig å oppdage for den som utvikler og/eller bruker modellen.

Kontrollmetoder

Filene med overgangssannsynligheter er forutsatt bygd opp omkring et relativt enkelt og generelt format. En egenskap er at alle kommentarlinjer i de eksterne filene starter med en '*', og normalt kan disse komme i et vilkårlig antall overalt i filen. Under innlesingen blir alle kommentarlinjer og blanke linjer utelatt fra videre behandling. Det legges opp til at øvrige linjer er organisert i kolonner separert av en eller flere blanke og/eller tabulatorer, eventuelt at dette gjelder fra og med en fast posisjon (det er da satt av et fast antall tegn til å gi tekstlig beskrivelse av innholdet på linja). For å gå videre til neste linje forutsettes det at all informasjon på linja er håndtert, se under. Dette sikrer at man unngår noen alvorlige feil som er vanskelig å oppdage, at informasjon er forskjøvet sideveis og at man leser inn fra midt i ett tall/kolonne, og at man leser inn feil kolonner (det er feilaktig lagt inn en eller flere kolonner for mye, eller man glemmer å hoppe over en kolonne). Den motsatte feilen, at det er for lite informasjon, vil C# normalt varsle om. En mulighet for feil som gjenstår er at man har feil rekkefølge på linjene i innlesingen og tilordningen av parametere. Denne faren kan reduseres ved å lese inn linjer ved å søke på en identifiserende tekststreng på begynnelsen av hver linje med parameteren (next_label).

Tidsserier

Noen av parameterne som skal leses inn består av tidsserier med antall begivenheter eller lignende som simuleringen skal kalibreres mot. Disse tidsseriene ligger på en egen type filer med navnestruktur 'time_series_*navn*.*versjon*', med noen kommentarlinjer i innledningen, og deretter en kompakt tabell med årstall som første kolonne og deretter en kolonne for hver strøm av antall begivenheter eller lignende. Tidsseriene leses inn med metoden 'read_time_series()' eller 'read_multiple_time_series()', som kontrollerer at alle år er med for den perioden man skal simulere. Blir tidsserien avsluttet med linja 'REPEAT', så repliseres siste år med data ut resten av framskrivingsperioden. Se metodebeskrivelsen nedenfor og følg gangen i en eksisterende bruk av 'read_time_series'.

Oppslag i tabeller

Det er mange ulike typer overgangssannsynligheter i modellen, og informasjonen om disse er heterogent organisert, noe som gjør det nødvendig med tilpassede metoder som leser inn filene og som leverer de relevante sannsynlighetene inn i simuleringen avhengig av ulike personkjennetegn. Samtidig vil noen av sannsynlighetene/parameterne bli brukt i simuleringen av flere kjennetegn, og det gjør det ønskelig at innlesings- og oppslagsmetodene ligger atskilt og tilgjengelig for alle modulene som simuler populasjonen. I tillegg vil omfanget av mange av innlesings- og beregningsmetodene være så store at de vil gjøre modulene som simulerer kjennetegnene uoversiktlige. En isolering av innlesings- og beregningsmetodene er oppnådd ved å legge overgangssannsynlighetene for hvert tema til hvert sitt objekt med innlesingsklassen 't_par' som moderklasse. Innad i hvert objekt er det metoder som gjør oppslag i tabellene, og i noen tilfeller er det lokale klasser/objekter som håndterer parameterne. Det gjelder spesielt der man har gjennomgående forklaringsvariabler, for eksempel at en logit-modell er estimert uavhengig for menn og kvinner med stort sett de samme forklaringsvariablene. Man trenger da kun å anvende variabelen kjønn én gang for hver beregning av én sannsynlighet, i stedet for én gang per parameter i sannsynlighetsuttrykket. Dette vil både forenkle (forkorte) kildekoden og redusere kjøretidene.

Kvalitetssikring

Selv med vekt på robuste innlesingsmetoder kommer man neppe utenom å teste at oppslagsmetodene faktisk leverer riktige beregnede sannsynligheter til simuleringen. Det er en svakhet ved MOSART at vi ikke har noe systematisk (og automatisk) opplegg for å teste at sannsynlighetene blir levert riktig til simuleringen. En vesentlig grunn er kanskje at disse testrutinene må tilpasses hver enkelt tabell og at testen bare blir brukt under utvikling av modellen. En mulighet er å bruke modellpopulasjonen der denne kan skrive ut sannsynligheter og andre relevante parametere.

Innlesingsklassen (read_base.cs, class t_par)

Innlesingsklassen inneholder felles metoder for å sikre en effektiv og mest mulig sikker innlesing av tabeller med overgangssannsynligheter. Oppdages irregulariteter under de forholdene som kontrolleres og er omtalt nedenfor, så stoppes simuleringen med en forhåpentligvis klar feilmelding. Under angis de metodene som brukes av de andre modulene.

open_file(fil-ident) Finner filnavnet knyttet til styrevariabelen '*fil-ident*' i 'input.con', sender dokumentasjon av denne filen 'sim.input', åpner filen, laster inn alle linjer med informasjon og legger denne i liste (List) av tekst (table) og nullstiller alle hjelpevariabler som håndterer denne listen

open_file_by_full_name(fil-navn)
Gjør det samme som 'open_file', men bruker '*fil-navn*' direkte

open_multiple_file(ps, fil-ident)
: Gjør det samme som 'open_file', men håndterer dette som at det kan være ulike filer for ulike pensjonssystemer (*ps*). For pensjonssystem større enn 1, søker metoden på styrevariabelen '*fil-ident_ps*', og finner den ingen fil så benyttes '*fil-ident*', men da med en *advarsel* om at den ikke fant en separat fil for dette pensjonssystemet. Alternative filer kan også legges inn via 'change_parameters.con'

close_file()
: Sjekker at all informasjon i gjeldende fil er håndtert og lukker deretter gjeldende fil

close_open_file()
: Lukker gjeldende fil uten krav om at all informasjon er håndtert. Metoden bør bare brukes der det gir mening at det er overflødige linjer, hvor hver linje har en identifiserende kolonne (jamfør next_label), og det er kompliserende å laste inn disse overflødige linjene. Et eksempel kan være tidsserier identifisert med årstall i første kolonne og hvor lengden under innlesingen kan variere på grunn av simuleringshorisonten

next_line()
next_line(sjekk)
next_line(antall-kolonner)
next_line(antall-kolonner, sjekk, med-labler)
: Sjekker først at inneværende linje er behandlet hvis parameteren '*sjekk*' er satt til sann (anbefales), flytter så informasjonsbehandlingen til neste linje med informasjon. Parameteren '*antall-kolonner*' angir hvor mange kolonner som skal finnes (dette sjekkes, med mindre tallverdien er mindre enn null). '*med-labler*' betyr at én blank blir håndtert som en del av teksten, og ikke som en kolonneseparator. I metodekall med færre parametere er default-verdier (-1, sann, usann)

next_label(tall-verdi)
next_label(tall-verdi, antall-kolonner)
next_label(linjenavn)
next_label(linjenavn, antall-kolonner)
next_label(tall-verdi, antall-kolonner, ingen-mellom-linjer)
next_label(linjenavn, antall-kolonner, ingen-mellom-linjer)
: Sjekker først at inneværende linje er behandlet, og leter så etter neste linje som begynner med parameteren '*linjenavn*' (+ en blank), hvor innledende blanke på linja blir ignorert. I metodekall med parameteren '*tall-verdi*' vil denne parameteren (heltall) bli konvertert til en tekststreng, og er mer anvendelig når identifikasjonen av en linje er for eksempel et årstall, en alder eller en annen tallangivelse. Hvis parameteren '*ingen-mellom-linjer*' er satt til sann, så godtas ingen mellomliggende linjer. Deretter leses linja inn fra etter '*linjenavn*' og utover. Parameteren '*antall-kolonner*' angir hvor mange kolonner som skal finnes eksklusive '*linjenavn*' (dette sjekkes, med mindre tallverdien er mindre enn null). I metodekall med færre parametere er '*antall-kolonner*' satt til -1 og '*ingen-mellom-linjer*' er satt til sann

repeated_label(linjenavn)
: Sjekker om noen av de gjenstående linjene starter med parameteren '*linjenavn*' tilsvarende som i 'next_label'

skip_rest_of_line() Setter 'current_column' til siste kolonne pluss 1, slik at 'next_line' og 'next_label' kan gå videre til neste linje selv om ikke all informasjon er behandlet. Dette er nødvendig blant annet der linjene avsluttes med kommentarfelt, og hensiktsmessig der linja skal hoppes over

table Er en liste (List) av tekststrenger som omfatter alle linjer med informasjon som ligger i gjeldende fil slik den er lest inn av 'open_file' eller 'open_file_by_full_name'

current_line_number
column
current_column Dette er variabler som håndterer den informasjonen som er lest inn av enten 'next_line' eller 'next_label'. '*Current_line_number*' refererer til hvilket linjenummer man er på i '*table*', hvor første linje er gitt nummeret 0. '*Column*' er en vektor av tekststrenger, en for hver kolonne. '*Current_column*' er den kolonnen som er aktiv for øyeblikket, hvor første kolonne er gitt nummeret 1 (merk forskjellen)

getint()
getint(k) Konverterer teksten fra kolonne '*k*' i '*column*' til ett heltall, og øker '*current_column*' med 1. Hvis ingen parameter er angitt bruker metoden '*current_column*' som verdi for '*k*

getdouble()
getdouble(k) Konverterer teksten fra kolonne '*k*' i '*column*' til ett desimaltall (double), og øker '*current_column*' med 1. Hvis ingen parameter er angitt bruker metoden '*current_column*' som verdi for '*k*'

get_first_column(l) Henter ut heltallverdien fra første kolonne i linjenummer '*l*'

next_line2
dcolumn Metode som sikrer raskere innlesing av (store mengder) data, hvor '*dcolumn*' er en liste av tall (List <double>)

year_time_series Variabel i read_base og angir årstall under innlesing av tidsserier

read_time_series(fil-ident, startår, sluttår, antall-kolonner, navn-på-lese-metode)
Dette er en metode som leser inn tidsserier på et bestemt format, det vil si med kommentarer på toppen, og deretter med år i forspalte og en tall-størrelse i hver kolonne. Parameteren '*fil-ident*' angir styrevariabelen i 'input.con', '*lengde*' angir linjelengde. Parameterne '*startår*' og '*sluttår*' angir over hvilken periode tidsserien skal leses inn. Parameteren '*antall-kolonner*' angir hvor mange kolonner som skal være i filen utover forspalten for år (er den mindre enn null overspringes sjekken). Parameteren '*navn-på-lese-metode*' er navnet på en metode som må defineres lokalt og leser inn de variablene som er relevante i tidsseriene. Finn en slik metode som allerede er i bruk for å lettere forstå gangen. Metoden har en parameter, og denne avgjør om man skal lese inn eller repetere siste linje. Ved innlesing refererer man til årstall i *'navn-på-lese-metode*' med variabelen '*year_time_series*'. Metoden '*read_time_series*' kontrollerer at alle relevante år kommer med og at linjene ikke inneholder overflødig informasjon. Derimot er det tillatt å ha med ekstra år før og etter den perioden som leses inn. Avsluttes tidsserien med 'REPEAT' repliseres siste år fra tidsserien til de resterende årene i

framskrivingsperioden. Det er da en forutsetning at metoden som brukes til innlesing (navn-på-lese-metode) har fått dette tillegget i kildekoden, og man bruker da variabelen 'year_repeat_time_series' for å angi år

read_multiple_time_series(ps, fil-ident, startår, sluttår, antall-kolonner, navn-på-lese-metode)
Metoden gjør i hovedsak det samme som '*read_time_series*', men har en tilleggsparameter '*ps*' som gjør at hvert pensjonssystem kan ha sine spesifikke filer. Ved gjennomsøk av 'input.con' vil metoden først forsøke å finne en styreparameter av typen '*fil-ident_ps*', og deretter bruke denne filen. Mislykkes dette logges en advarsel, og simuleringen benytter da filen tilknyttet '*fil-ident*'

check_probabilities(sannsynligheter, minste-verdi, største-verdi, identitet)
Boolsk metode (sann hvis feilfri) som kontrollerer og retter opp vektoren '*sannsynligheter*' som skal inneholde en multinomisk fordeling over heltallene (*minste-verdi*, *største-verdi*). '*Identitet*' er en vilkårlig tekststreng som kan brukes til å identifisere eventuelle feilmeldinger som blir skrevet ut av metoden

Vedlegg J: Felles klasser og objekter i simuleringsmodellen

Modellen MOSART har en rekke felles klasser som skal forenkle kildekoden. Nedenfor følger en oversikt over noen av disse klassene og deres innhold. Objektet 'global.cs' med felles konstanter og enklere hjelpemetoder er omtalt i vedlegg K. Overskrifter under angir klassens/metodens navn og filreferanse, etterfulgt av en kort forklaring. Første kolonne i hver linje angir en variabel/metode som tilhører klassen, med format på eventuelle klasseparametere, mens andre kolonne gir en kort forklaring av innhold og/eller virkning.

Klasser

- Trekkemetoder
- Personer
- Kohortliste
- Populasjonsliste
- Binærtrær
- Innlesing av brukergrensesnitt
- Trekkemetoder

Trekkemetoder (draw_base.cs, class t_draw_base)

Alle trekkemetodene i MOSART er klasser avledet fra grunnklassen for trekkemetoder (class t_draw_base), som har en del felles metode for håndtering av trekninger. Standard parametervalg styres gjennom det globale objektet. Trekkeobjekter som skal brukes over flere år bør inn i lista med trekkeobjekter (g.draw), og hvert enkelt trekkeobjekt vil da bli automatisk vedlikeholdt på begynnelsen av hvert år i 'update.cs', blant annet med hensyn til valg av nullstilling av parametere. Skal trekkeobjektet inn i lista er det et krav at trekkeobjektet får en unik 'identitet', denne har som formål blant annet å kunne spore feilmeldinger tilbake til kilden til avviket.

De angitte trekkemetodene inneholder metoder som kan redusere variansen betydelig, se kapittel 4 for en nærmere beskrivelse. Trekkeklassene ligger i 'draw_*.cs', og omfatter binomiske begivenheter (class t_draw_binomial), multinomiske begivenheter (class t_draw_multinomial), multinomiske begivenheter med homogene sannsynligheter (class t_draw_homogeneous), uniforme fordelinger (class t_draw_uniform) og par av uniforme fordelinger (class t_draw_bi_uniform). Valg av ulike trekkemetoder gjøres ved hjelp av parametere i 'parameters.con', se vedlegg E. Disse valgene kan overstyres ved å spesifisere navnet på trekkeobjektet (identitet) som en styrevariabel i filen 'parameters.con', og som parameterverdi angi den spesielle trekkemetoden som skal benyttes i denne metoden. Feilmeldinger logges til 'sim.errors.draw'.

På grunn av multithreading så må hvert trekkeobjekt kun brukes av enhver *potensiell* tråd, og rekkefølgen trekkeobjektene eksekveres kan heller ikke tillates å påvirke simuleringen. Dette gjelder også uavhengig av hvor mange tråder man har spesifisert. Det betyr at de kriteriene som inngår i inndelingen av tråder/jobber, også må inngå i inndelingen av trekkeobjektene. Normalt vil dette si kjønn og ettårig aldersinndeling. Man bør derfor være tilbakeholden med ytterligere inndeling av trekkeobjektene.

For 't_draw_multinomial' må man angi antall utfall i sannsynlighetsvektoren (største verdi), og eventuelt en tekststreng (sg_) som i 'input.con' identifiserer valg av sortering av utfallsrommet. I 't_draw_homogeneuous' må man i tillegg angi sannsynlighetsvektoren som skal benyttes.

control_and_restore	Virtuell metode som kalles av control_all som eksekveres på begynnelsen av hvert år. Metoden 'control_and_restore' gis innhold i hver av de avledete klassene som definerer de ulike trekkemetodene, hvor formålet er å

kontrollere for feil og opphopninger av data i metodene, samt hvis parameteren 'stratified_restore' er sann så skal alle trekke-parametere nullstilles eller re-initialiseres

't_draw_binomial': an_event(sannsynlighet)
Boolsk metode som returnerer med sant hvis en begivenhet skal inntreffe, og dette skjer med den angitte sannsynligheten

't_draw_multinomial': state(sannsynlighetsvektor)
Returner hvilket utfall som inntreffer (heltalls-variabel) med angitte sannsynlighetsfordeling

't_draw_homogeneous: state()
Returnerer hvilket utfall som inntreffer (heltalls-variabel) med de sannsynlighetene som ble brukt da metoden ble opprettet

't_draw_uniform': z()
Returner et tall med uniform fordeling (0,1)

't_draw_bi_uniform': rv(ref z1, ref z2)
Returnerer verdier på '*z1*'og '*z2*', begge med uniform fordeling (0,1) og uavhengig av hverandre

Personer (person.cs, class person)

Personobjektet inkluderer alle personkjennetegnene. Se filen 'person.cs' for detaljer innen hvert enkelt tema. Personobjektet vil definere behovet for internminne i simuleringer med store utvalg (i tillegg til utvalgsstørrelsen). Effektiv lagring av data er derfor spesielt viktig i denne klassen.

Arbeidslister (work_distributor.cs, class t_work_distributor)

Organiseringen av multithreading krever relativt mye kildekode som er tilnærmet likt fra modul til modul. Vi har organisert denne kildekoden i en egen fil (work_distributor.cs), med en klasse som håndterer multithreadingen (class t_work_distributor), og to hjelpeklasser (class t_work_list, class WorklistComparer). Ved utviklingen av nye moduler er det enklest å ta utgangspunkt i bruken av 'work_distributor' i en modul som er mest mulig lik den man utvikler. Se også avsnitt 3.6.

Boks J.1

```
class t_kategori
{
        // Konstruktør
        public t_kategori( ... )
        {
                wd_work_distributor = new t_work_distributor(parametere);
        }

        // Simulering av et kjennetegn
        public void modul-navn()
        {
                wd_modul-navn.update(parametere);
                wd_modul-navn.execute(this.worker_modul-navn);
        }

        // Arbeidsfordeler
        private t_work_distributor wd_modul-navn;

        // Arbeidsutfører
        public void worker_modul-navn(parametere)
        {
                Instruksjoner for overføring av parametere;

                t_work_list wl = wd_modul-navn.next();
                while (wl != null)
                {
                        Instruksjoner for simulering av en kohort;
                        wl = wl.next();
                }
        }
}
```

Boks J.1 gir strukturen for bruk av 'work_distributor' for et vilkårlig tema (for eksempel trygd) navngitt som *kategori* og et vilkårlig kjennetegn kalt *modul-navn* (for eksempel alderspensjonering). I konstruktøren til klassen '*kategori*' opprettes en arbeidsfordeler for denne modulen. I 'model.cs' som simulerer populasjonen, kalles simuleringen av dette kjennetegnet på samme måte som uten multithreading (*kategori.modul-navn*()). I den første linja av 'public void *modul-navn*() { ... }' nullstilles arbeidsfordeleren. I den andre linja eksekveres arbeidsfordeleren.

Arbeidsutføreren (worker) er en metode som fortløpende får tildelt en kohort etter kjønn, alder og reg.status, og som simulerer denne kohorten på ellers vanlig måte. Referansen 'wl' (work list) vil inneholde opplysninger om hvilken kohort arbeidsutføreren for øyeblikket jobber på.

Det som kompliserer bruken er spesielt når informasjon må overføres mellom hovedprogrammet og hver enkelt tråd, uansett hvilken vei informasjonen går. I tillegg vil de enkelte arbeidsfordelerne ha ulike avgrensninger på alder, kjønn og reg.status. Det er her man bør finne en modul som ligger tettest mulig opp til den man utvikler.

Kohortliste (cohort_list.cs, class t_cohort_list)

Hver kohortliste omfatter alle personer av et bestemt kjønn, en bestemt årsklasse og en bestemt reg.status i en modellpopulasjon. Kohortlister brukes bare i 'class t_population', og inneholder en del opplysninger og metoder som kun brukes her. Personene er lagret i en liste (List) som gir tilgang til en rekke effektive sorterings- og søkemetoder i C#.

Populasjonsliste (population_list.cs, class t_population)

Populasjonsliste inkluderer en matrise over kjønn, fødselsår og reg.status med kohortlister, samt en del metoder for å håndtere kohortlistene. Endringer av listetilhørighet, plassering i liste og tilhørende referanse skal kun skje gjennom de angitte metodene under. Referansen til populasjonslista ligger i hovedprogrammet med navnet 'population', og alle klasser/metoder som skal bruke populasjonslista må få denne referansen overført som parameter.

list_of_persons(reg.status, kjønn, fødselsår)
: Referanse til en List med alle personer i en kohort etter reg.status, kjønn og fødselsår

sort_cohort_list(reg.status, kjønn, fødselsår)
: Sorterer angitte kohort i en tilfeldig rekkefølge

resort_live_population()
: Sorterer alle levende personer i en tilfeldig rekkefølge innenfor sin kohort

no_of_persons(reg.status, kjønn, fødselsår)
: Antall personer totalt i populasjonen med disse kjennetegnene

count()
: Antall personer total i populasjonen

into_last_place(person)
: Plasserer *person* sist i sin kohortliste

remove_last_person(person)
: Fjerner siste person i kohortlista til *person*, krever at *person* faktisk er siste person i kohortlista

last_person(reg.status, kjønn, fødselsår)
: Returnerer siste person angitte kohortliste

restore_random_list(reg.status, kjønn, fødselsår)
no_of_remaining_persons(reg.status, kjønn, fødselsår)
next_random_person(reg.status, kjønn, fødselsår)
: Metoder som finner en tilfeldig person i angitte kohortliste, se avsnitt 4.8

Binærtrær (t_binary_tree(g), binary_tree.cs)

Binærtrær er en metode som sorterer objekter fortløpende, og som regnes med blant effektive sorteringsmetoder. Klassen 't_binary_tree' er en implementering av denne tilnærmingen i MOSART. Sammenliknet med standardmetoden i C# med 'List' og 'Sort', vil dette binærtreet trolig etterlate seg mindre søppel

new_element(sorteringsvariabel, p)

Tilordner en ny person '*p*' til binærtreet på sortert plass, angitt ved verdien på '*sorteringsvariabel*'

t_node build_sorted_list()
: Tilordner verdier til 'next_element' som i neste runde gjør det enklere å gå gjennom binærtreet i sortert rekkefølge. Returverdien er en referanse til første element i den sorterte lista

t_node next_element()
: Returner en referanse til neste person/element i den sorterte lista

Enumerators (enumerators.cs)

Dette er en liste med enumarators (enum), som er en variabeltype som kan bidra til at man i mindre grad risikerer å forveksle variabler i kildekoden. Hvis en konstant for henholdsvis ekteskapelig status og pensjonsstatus begge er definert som en heltallsvariabel (eller enda verre, angitt som faktiske tall i kildekoden), så er det lett å forveksle disse i for eksempel et metodekall (kompilatoren godtar forvekslingen). Definert som 'enum' får hvert kjennetegn sine unike koder, og utilsiktet forveksling er ikke mulig. Bruken av enumerators er i en startfase i MOSART, i første rekke for parametere som styrer bruken av metoder og klasser, samt innlesing av regler i pensjonssystemet.

Innlesing av brukergrensesnittet (read_parameters.cs, t_read_parameters)

Dette er en klasse som leser inn filer i brukergrensesnittet, og henter ut informasjonen på en robust måte. Der denne brukes i 'global.cs' har disse filene fått unike navn, mens der den brukes lokalt blant annet ved innlesing av pensjonsregler kan disse objektene godt få et generisk navn. Se for eksempel anvendelsen i 'read_pension.cs' og 'read_education.cs'.

Robustheten går blant annet på at innlesingsmetodene sjekker at parameteren forekommer en og bare en gang i brukergrensesnittet (som er en feil som veldig vanskelig å oppdage, med mindre den gir helt absurde resultater).

Følgende metoder brukes generelt, men de kan fores med litt forskjellige argumenter, avhengig av om de avhenger av pensjonssystem, om man vil tilordne tallverdien til variabelen lokalt (x=read_parameter.as_int(ident)) eller la funksjonen gjøre dette (read_parameter.as_int(out x,ident)), og om 'x' er skalar, vektor eller matrise. Argumentene vil typisk være i denne rekkefølgen (ps, ident) eller (ps, out x, ident), hvor ps er pensjonssystem, x er størrelsen som skal tilordnes en verdi og ident er tekststrengen i første kolonne i fila som parameterverdien skal hentes fra.

as_int(...)
: Returnerer/tilordner en heltallsverdi (int). Forutsetter at det som leses også er heltall

as_double(...)
: Returnerer/tilordner et flyttall (double)

as_bool(...)
: Returnerer/tilordner en boolsk variabel, 0: Usann, >0: Sann

as_string(...)
: Returnerer/tilordner parameterverdien som en tekststreng

as_year(...)
: Returnerer/tilordner som et heltall (int), men hvor negative verdier settes til 'eternity', det vil si (sim_end+max_age+1)

as_rule(...)
: Returnerer/tilordner som en nærmere spesifisert enum, og som default vil enum-navnet omdannes til *ident*. Er det ikke mulig kan man legge inn *ident*

eksplisitt i metodekallet. Denne metoden brukes spesielt på pensjonsregler, og sikrer i langt større grad at man leser inn lovlige verdier, og gir ved feil en oversikt over hvilke regler som er tilgjengelig

as_filestream(...) Returnerer/tilordner parameterverdien som en fil (FileStream)

found(ident) Sjekker om '*ident*' finnes i parameterfila

find(...) Sjekker om og leter seg fram til linja hvor *ident* ligger, brukes når det skal leses inn flere verdier fra samme linje, for eksempel en liste med år

as_directory_name(ident)
as_multiple_directory_name(ps, ident)
Leser inn og sjekker at parameteren er en katalog, brukes der inputfilene er en gruppe av filer som skal leses inn sammen, gjelder valg av fagfelt i utdanning og (under utvikling) nye overgangssannsynligheter for inntektsnivå

Trekkemetoder (draw.cs, draw_base.cs)

Trekkemetodene, spesielt random-seed, er organisert i et objekt kalt 'draw' som ligger 'global.cs'. Selve trekkemetodene deklareres og initieres rundt i de ulike modulene som simulerer kjennetegn ('sim_*navn*.cs', 'read_initial_population.cs'), og dette systemet er ganske fragilt hvis man skal opprettholde eksakt reproduserbarhet. Multithreading og asymmetriske startår for simuleringen forsterker dette problemet.

Det er derfor helt avgjørende at alle trekkemetoder initieres ***før*** både simuleringen starter, men også den egentlige innlesingen av utgangspopulasjonen. I simuleringsrutinene sikres dette ved å bruke metoden 'public override void initialize_drawing_procedures()'. Man fyller denne med innhold, og en bakenforliggende struktur sørger for at disse blir eksekvert i tur og orden i en fast rekkefølge. Av samme grunn skal man unngå å legge trekkemetoder (t_draw_...) til andre moduler, herunder de som leser inn overgangssannsynligheter (det er andre grunner også, trekkemetodene er gjennomgående mer synlige i simuleringsmodulene, men kan bli veldig bortgjemt i lange og anonyme innlesingsmoduler).

Legger man initieringen til andre steder er det fort gjort at maskinstøy, det vil si den rekkefølgen operativsystemet utfører jobben på via ulike tråder, kan påvirke rekkefølgen, og dermed hele simuleringsstøyen.

stratified_binomial
stratified_multinomial
stratified_homogeneous
stratified_continuous
stratified_restore
Fem variabler henter sine verdier fra 'parameters.con' og som angir hvilke trekkemetoder som skal benyttes. De fire første en enum, og velger trekkemetode. Den siste er bool, og angir om parametere i trekkemetodene skal nullstilles ved inngangen til hvert år.

Vedlegg K: Felles variable og hjelpemetoder

I simuleringsmodellen er de fleste felles hjelpemetoder og noen globale variable samlet i hva vi har kalt det globale objektet (global.cs, class t_global, g). Man får tilgang til disse hjelpemetodene og variablene ved å ha en referanse til det globale objektet i modellen, typisk kalt 'g', og deretter skrive 'g.' foran den metoden/variabelen man ønsker å bruke. Vær oppmerksom på at flere av metodene returner en referanse til et objekt, for eksempel en fil. Første kolonne angir variabelens/metodens navn, med format på eventuelle argumenter (flyttall hvis ikke annet er nevnt), mens andre kolonne gir en kort forklaring av innhold og/eller virkning. Kun metoder med en viss allmenn bruk er tatt med.

Emner

- Systemrelasjoner
- Jobbdokumentasjon
- Utvalget
- Tidsangivelser
- Trekkemetoder
- Brukerdefinerte funksjoner
- Konstanter
- Innlesing av brukergrensesnittet

Systemrelasjoner

establish_and_check_external_relations
: En av de første instruksjonene som eksekveres i simuleringen. Metoden kontrollerer at simuleringen kan startes, etablerer katalogreferanser, dokumenterer deler av simuleringen og sletter/sikkerhetskopierer eventuelle tidligere resultater. Samtidig sperres det for at simuleringen kan startes på nytt så lenge simuleringen pågår

terminate_simulation
: Avslutter simuleringen med dokumentasjon og filbehandling som ligger utenfor simuleringsmodellen, blant annet gjelder dette eventuell skrivebeskyttelse for resultatfilene og flytting av stopfile. Metoden er en av de siste som eksekveres i simuleringen, og etter dette metodekallet er det mulig å eksekvere simuleringen på nytt

unix(kommando til Linux)
: '*Unix*' er en metode som gjør at simuleringsmodellen kan eksekvere andre programmer på kommandolinja i Linux. '*kommando-til-Linux*' er en tekst som skal ha samme format som én linje i et shellscript. Simuleringen venter med å fortsette til kommandoen er ferdig

model_name
: Tekststreng med navnet på modellen, hentes inn via den eksterne modulen 'model_name.cs'

work_area
: Tekststreng med navn på katalogen som resultatfiler skal legges ut på

heading
: Tekststreng med navn på simuleringen (alt som følger etter 'res' eller 'prog' i katalognavnet)

job_date
: Tekststreng med dato og klokkeslett for når simuleringen ble startet

check_stopfile — Kontrollerer fortløpende at simuleringen er unik på sitt arbeidsområde, og stopper eventuelt simuleringen ved irregulariteter

Error()
Error(*identitet*)
Error(*identitet*, p)
Error(*identitet*, p, *melding*)

Stanser simuleringen, og skriver til skjerm (standard output) teksten '*identitiet*', andre '*meldinger*' og eventuelt også all informasjon om personen '*p*'. '*Identitet*' gjør at man kan finne tilbake til der feilmeldingen kommer fra i kildekoden, i de tilfeller man mister denne informasjonen fra systemet selv (dette var et større problem i tidligere versjoner av C# og Simula). '*Melding*' er en liste (List) med feilmeldinger som også skrives ut, og kan ta mange formater, se 'global.cs'

delete_file(fil-navn)
copy_file(fil-navn, katalog-navn)
move_file(fil-navn, katalog-navn)

Sletter/kopierer/flytter filen med navn '*fil-navn*' til '*katalog-navn*', så lenge begge ligger i arbeidsområdet

readable_file(fullt-fil-navn)
writable_directory(katalog-navn)

To boolske metoder som returnerer sann hvis filen/katalogen er henholdsvis lesbar/skrivbar. '*Readable_file*' må ha hele filnavnet, '*writable_directory*' angir en katalog ***i*** arbeidsområdet

new_file(fil-navn,overskrift)
existing_outfile(fil-navn,lengde)

To metoder som åpner utfiler, '*fil-navn*' refererer til fysisk navn på resultat-katalogen. '*Overskrift*' er en tekststreng som skrives inn i hodet på filen når den åpnes første gang, og da av '*new_outfile*' eller '*new_file*'. '*Lengde*' refererer til hvor lange linjene skal være. '*New-file*' skiller seg fra de to andre ved at den lukker filen som opprettes, mens man for de to andre må være påpasselig med å gjøre dette selv. '*Old_outfile*' åpner filen slik at nye meldinger *legges* til filen

first_line(fil-navn) — Returnerer første linje i '*fil-navn*' som en tekst

Jobbdokumentasjon

sim_control.write(*ident*, *melding*)

Metoden skriver en melding til filen 'sim.control' med teksten '*ident*' som forspalte og eventuelt teksten '*melding*' som eventuell tillegg på slutten av linja. Samtidig rapporteres tidsbruk og CPU-tid siden forrige gang metoden ble kalt, og løpende bruk av internminne. Man kan få supplerende opplysninger om regnekraft, internminne og aktivitet i garbage collector, men da rapportert til 'sim.garbage_collector' og 'sim.memory', dette styres av parametere i 'output.documentation_and_special_tables.con'. Se også 'sim_control.cs' for mer detaljer og varianter av 'write'

sim_information	Objekt som skriver ut relevant nøkkelinformasjon til filen 'sim.information' i starten av simuleringen, noe informasjon utover i simuleringen, og i avslutningen av simuleringen. Se 'sim_information.cs' for detaljer
sim_report_input	Objekt/fil for å skrive dato-opplysninger for filer som leses inn til 'sim.report.input'
write_error(beskjed)	Skriver '*beskjed*' til skjerm (standard output), som 'job.exe' omdirigerer til 'sim.errors'. Brukes enten sammen med 'Error' for å rapportere årsaken til/begrunnelsen for at simuleringen blir stoppet, eller for å rapportere mindre alvorlige avvik (linjer som starter med 'WARNING: '). '*Beskjed*' kan ha flere formater, se 'global.cs' for detaljer

Utvalget

sample_size	Utvalgsstørrelsen
sample_weight	Den inverse av utvalgsstørrelsen hvis denne er større enn null, ellers én
read_only_parameters	Leser da kun inn overgangssannsynlighetene og andre parametere, utelater innlesingen av utgangspopulasjonen og selve simuleringen, men tar med avslutningen. Brukes ved utvikling av nye innlesingsrutiner, da dette reduserer kjøretiden
initial_population	Valg av type utgangspopulasjon, 0: ingen utgangspopulasjon og utvalgsstørrelsen settes arbitrært til 1,0 (brukes blant annet for uttesting av nye tabellprogram da det reduserer kjøretiden, men gir tabellutskrifter), 1: Ordinær utgangspopulasjon (se vedlegg L), 2: simulert utgangspopulasjon. Se også 'enum_initial_population' i 'enumerators.cs'
new_id_number	Returnerer et ubrukt identifikasjonsnummer fortløpende fra første hele million som er (ekte) større enn største identifikasjonnummer i utgangspopulasjonen, 1 hvis simulert utgangspopulasjon. Startverdien er lavere for eldre versjoner av MOSART

Tidsangivelser

sim_year	Årstall i simuleringen
base_year	Siste året med data i utgangspopulasjonen, og normalt det siste året som ikke blir simulert
partial_start_year_*navn*	Startår for simuleringen av tema *navn*, normalt base_year+1, men enkelte kjennetegn går ikke til og med base_year, og det er mulig å starte simuleringen generelt før dataene slutter. Se 'parameters.con'
last_year_without_simulation	Siste året uten noen form for simulering
sim_end	Sluttåret for simuleringen

eternity	Antatt evigheten i simuleringen, definert som sim_end[31]+max_age+1. Dette er en størrelse som brukes mest i deklarasjonen av mange tidsserier med videre for å sikre at en person alltid kan hente ut en tallverdi (for eksempel ett grunnbeløp) for et år som kan være relevant i denne personens (for eksempel) pensjonsberegninger (det vil si mulige levetid)
first_birth_year	Antatt eldste mulige årskull i utgangspopulasjonen, for tiden minimum av (1850, base_year - max_age)
current_first_birth_year	Faktisk eldste årskull i simuleringen, inkludert døde personer som fortsatt ligger i populasjonslistene
base_year_prices	Basisåret for priser og lønninger, det vil si at alle nominelle størrelser blir deflatert med lønnsnivået i dette året, jamfør 'time_series_wages' i 'input.con'. Merk at grunnbeløpet har sitt eget basisår, men generelt bør disse to basisårene ha samme verdi. Grunnbeløpet går ofte lenger fram i tid (er kjent per 1.mai), men det er viktig at priser, lønninger og grunnbeløp er konsistente, og at det som da er lagt inn av pris- og lønnsvekst for det året man mangler faktisk er konsistent med det som ligger til grunn for reguleringen av grunnbeløpet
start_year_time_series	Første år i alle tidsseriene for ulike begivenheter, for tiden 1960
max_age	Høyeste tillatte alder i simuleringen, for tiden 130 år. Kan ikke endres uten omfattende endringer i filene som leses inn

Trekkemetoder

draw	Objekt som håndterer random-seed med videre, se vedlegg J
logistic_distribtuion(z)	'*logistic_distribution*' returner et tall med logistisk fordeling med forventning null og standardavvik én hvis '*z*' er uniformt fordelt (0,1). Halene er kuttet av ved (0,00001,0,99999) for å unngå ekstreme verdier iblant annet simuleringen av inntekt
normal_distribution(z)	'*normal_distribution*' returner et tall med normalfordeling med forventning null og standardavvik én hvis '*z*' er uniformt fordelt (0,1). Halene er kuttet av ved (0,00001,0,99999) for å unngå ekstreme verdier iblant annet simuleringen av inntekt

Justeringsmetoder

max_no_of_iterations	Maksimalt tillatt antall iterasjoner hvor man justerer sannsynlighetene iterativt, må implementeres av bruker
convergence_criteria	Maksimalt tillatt avvik i justeringsalgoritmene. Ganges med utvalgsstørrelsen før bruk i simuleringen. Kan være et desimaltall

[31] I de tilfellene sim_end er før base_year, brukes i stedet base_year her. Det skjer veldig sjelden, men er teknisk mulig.

adjust_prob(p,r)
adjust_prob(ref p, r)
adjust_prob(ref p1, ref p2, r1, r2)
adjust_prob(ref pv, rv)

Fire metoder som justerer sannsynligheten(e) '*p*' analogt med å justere konstantleddet i et logit-uttrykk, se avsnitt 4.9. De to første justerer den binomiske sannsynligheten '*p*' relativt opp med '*r*', den siste med tilbakelegging. Den tredje metoden justerer to av sannsynlighetene i en trinomisk fordeling (med tilbakelegging), hvor det implisitt er antatt at r3=0 (normering). Den siste justerer sannsynlighetene i en multinomisk fordeling (med tilbakelegging). Ingen av utfallene er normert, og det er opp til bruker å ivareta dette (er for eksempel *alle* r-verdiene lik 2, da blir det ingen justering)

Brukerdefinerte funksjoner

Noen av disse funksjonene har paralleller i C#, men har blitt hengende igjen av historiske grunner, eller C#-varianten er enten knotete utformet og/eller returnerer med feil type, det vil at svaret er et heltall (og burde vært int), men kommer tilbake som et flyttall, rett nok uten desimaler, men som 'double'.

average(x1, x2)
average(x1, x2, x3) Beregner gjennomsnittet av disse tallene

convert_int_to_int(x) Returner desimaltallet x som heltall (int), under forutsetning av at det ikke er desimaler i x. Noen heltall blir deklarert som flyttall, og dette er en robust metode for å få det over til type 'int'

entier(x) Returnerer heltallverdien av x rundet av nedover. Gjør det samme som 'Math.Floor', men returverdien er et ordinært heltall (int)

minmax(nedre-grense, x, øvre-grense)
Avgrenser '*x*' til intervallet ['*nedre-grense*', '*øvre-grense*'], '*nedre-grense*' må være mindre eller lik '*øvre-grense*'. Er alle tallene heltall (int), blir også returverdien heltall (int)

mod(dividend, divisor)
Returner restverdien som blir igjen når *dividend* er delt med *divisor*

rest(x) Returnerer tallverdien av '*x*' minus heltallsverdien av '*x*'

round(x)
round_as_ushort(x)
round_as_byte(x) Avrunder *x* etter de reglene som C# bruker, og returner med angitt type, default er vanlig heltall (int)

upside(x) Returner *x* avrundet oppover. Gjør det samme som 'Math.Ceiling', men returverdien er et ordinært heltall (int)

standard_date() Henter tidsangivelse ved 'DateTime'

relative_time(date) Returnerer antall sekunder siden 1.1.2018 angitt ved tekststrengen '*date*', som er hentet ut ved 'standard_date'

Konstanter

male, female	Konstanter for henholdsvis menn og kvinner
opposite_sex(sex)	Returnerer tall-verdien for det motsatte kjønn av '*sex*'

De fleste personvariablene har konstanter som angir statusverdiene, samt en variabel av typen 'nos_*variabel-navn*' som angir antall statuser for denne variabelen.

Innlesing av brukergrensesnitt

Deler av brukergrensesnittet leses inn og håndteres av 'global.cs', mens blant annet pensjonsregler, skatteregler, utdanning og arbeidsinntekt er lagt til sine respektive moduler. Se vedlegg J for en nærmere omtale av hvordan dataene hentes ut.

read_parameter	Håndterer 'parameters.con', det vil si felles parametere
read_input	Håndterer 'input.con', det vil si inputfiler
read_income	Håndterer 'income.con', fordi lønnsdeflateringen hentes herfra og er fundamental for all håndtering av nominelle størrelser, må denne leses inn her på et tidlig og forutsigbart tidspunkt
read_output	Håndterer 'output.con' og 'output.*.con'

Vedlegg L: Utgangspopulasjonen

Et særtrekk ved modellen MOSART er at utgangspopulasjonen henter inn (et utvalg av) befolkningen med faktiske opplysninger omkring disse personene både på starttidspunktet for simuleringen og med bakoverskuende registerdata. Dette vedlegget gir en oversikt over hvordan disse dataene er organisert på ulike filer.

Dataene hentes inn ved parameteren 'initial_population' i 'input.con', og versjonen som omtales her er lagret på '/mosart/input/v75/population/c_2022'. Dataene leses inn av 'read_initial_population.cs'.

Filformater og organisering av utvalg

Samtlige av filene er tekstfiler, filer med persondata har en linje per person (per år) og fast felt-beskrivelse for hele filen. Alle kolonner i alle filer er separert med minst en blank, og ellers organisert som rette kolonner (lesbar på skjerm). Følgende en modellversjon er det flere utgangspopulasjoner, med variasjoner over basisår og utvalgsstørrelse. Hver utgangspopulasjon har sin katalog med fullt sett av filer, hvor basisår og utvalgsstørrelse normalt inngår i navnet, for eksempel:

/mosart/input/v75/population/c_2022
/mosart/input/v75/population/c_2022_s01
: Førstnevnte er hele populasjonen i 2022 den andre er et 1 prosent utvalg, hvor s01 angir andelen, det vil si 0,01

Noen av variablene er nærmere beskrevet i vedlegg M.

Innlesing av utgangspopulasjonen

Modulen som leser inn utgangspopulasjonen (read_initial_population.cs) har blitt svært omfattende, og er kanskje moden for en tydeligere modulisering. Ved å lese eksekveringsdelen i begynnelsen av modulen får man likevel en viss oversikt. Merk at når man legger til nye kjennetegn i utgangspopulasjonen, så holder modulen automatisk rede på *endringer* i kolonneplassering av kjennetegnene, men man må eventuelt bare legge til de nye kjennetegnene.

Filer som inngår

control
: Dokumentasjon av jobben som produserte utgangspopulasjonen

adapt_population_6.cs
: Kopi av kildekoden som produserte denne utgangspopulasjon

parameters.con
: Parametere for å produsere utgangspopulasjonen

persons.record
persons_*i*_*j*
: Status for utgangspopulasjonen i basisåret[32] (2022) for kjønn *i* og alder *j*

longitudinal_data.record
longitudinal_data_*i*_*j*
: Forløpsdata for utgangspopulasjonen fra og med 1966/alder=0 og til og med basisåret/dødsåret, for kjønn *i* og alder *j*

spouse_data.record

[32] Det siste året vi for øyeblikket har demografiske data for, her 2022.

spouse_data_*i*_*j*	Data over parforhold, med type, start og slutt, for kjønn *i* og alder *j*
start_year_longitudinal_data	Startår for longitudinelle data
last_year_with_data	Siste år med data for utgangspopulasjonen for de ulike temaene, basisåret er parameterverdien for 'demography'
sample_size	Andelen som utvalget utgjør av totalpopulasjonen
highest_id_number	Største id-nummer i utgangspopulasjonen
private_occupational_pension_parameters	Parametere i tilretteleggingen av data for privat tjenestepensjon
MOSART_lotte_map.record	
MOSART_lotte_map	Koblingsnøkkel av ID-numre MOSART-Lotte

Supplerende data:

enterprises.record	
enterprises	Data for hver enkelt virksomhet (bedrift)
establishments.record	
establishments	Data for hvert enkelt foretak
highest_org_id	Største organisasjonsnummer (kryptert) i utgangspopulasjonen
adjust_children	Litt dokumentasjon på simuleringen av manglende fruktbarhetshistorier for de eldste, det vil si kvinner (potensielle mødre) født 1928 eller tidligere for barn som er født 1948 og tidligere (som potensielt ikke har blitt fanget opp av Folke- og Boligtellingen 1960)
table_lme	Tabell med pensjonsgivende inntekt, brukes fortsatt i kalibreringen av MOSART
table_mortality	Tabell med dødelighet etter kjønn, alder og år, brukes fortsatt i kalibreringen av MOSART
table_*navn*	Andre tabeller for kontroll av innholdet i utgangspopulasjonen
errors	Meldinger om irregulariteter i tilretteleggingen av data
errors2	Hjelpefil som kopierer feilmeldingsfila fra jobben som tilrettela dataene, omfatter 'errors' over, samt noen andre meldinger
backup_*dato*	Enkelte av filene over hvis utgangspopulasjonen har vært forsøkt produsert flere ganger

Vedlegg M: Variabelbeskrivelser

Nedenfor følger en forklaring av noen av variablene som er tildelt egne koder med videre i datagrunnlaget for MOSART. Filen 'global.cs' i simuleringsmodellen inneholder også dokumentasjon av innholdet i mange av variablene.

Identifikasjonsnummer (id_number)

Vilkårlig heltall som alene én-entydig identifiserer hver person i utvalget. Verdiene går fortløpende fra 1 og oppover, og gir ingen informasjon om personene, heller ikke kjønn og fødselsår. For eksempel vil de *n* laveste ID-numrene være et tilfeldig utvalg av befolkningen, uansett verdi for *n*. Når identifikasjonsnummer for partner, ektefelle, barn og foreldre er satt til '0' betyr det at opplysningen mangler eller ikke kan finnes. Det er satt av 8 felter til identifikasjonsnummer, da det muliggjør samme type utfiler fra simuleringen hvor nye identifikasjonsnummer fort kan overskride 9.999.999). Tilordning av identifikasjonsnummer i simuleringen starter med høyeste identifikasjonsnummer i utgangspopulasjonen, avrundet oppover til nærmeste hele million, for tiden 11.000.000. Bruker man en simulert utgangspopulasjon starter tilordningen på verdien 1.

Kjønn

1: Mann, 2: Kvinne, 0 vil i en del sammenhenger betegne begge kjønn (tabeller med videre)

Årstall

Årstall er angitt med fire siffer. '0' angir at årstallet er ukjent, for årstall fra personregisteret vil dette normalt tilsvare begivenheter for ekteskap og inn- og utvandring før 1964.

Reg.status

1: Bosatt i Norge, 2: Død, 3: Bosatt i utlandet, 4: Død i tidligere år, 0: Ukjent

Ekteskapelig status

1: Ugift, 2: Gift, 3: Enke/enkemann, 4: Skilt, 5: Separert, 0: Ukjent

Ektefelle, partner

Identifikasjonsnummer for ektefelle viser til nåværende ektefelle for gifte og siste ektefelle for før gifte. Identifikasjonsnummer for partner viser til den personen vedkommende bor sammen med, hvis de enten er gift eller har barn sammen, samt i en del andre tilfeller også til husholdninger med to personer av motsatt kjønn og uten andre familiære relasjoner.

Antall barn, alder yngste barn

Der opplysninger mangler er '0' satt inn.

Status for skolegang

1: Ingen utdanningsaktiviteter, 2: Er under utdanning, 3: Har begynt utdanning i år, 4: Kandidat, 5: Har avbrutt utdanning i år.

Trygdestatus

1: Ikke pensjonist, 2: Under attføring ved årets slutt, 3: Uførepensjonist, 4: Alderspensjonist, 5: Etterlattepensjonist, 6: AFP-pensjonist

Kronebeløp
Alle kronebeløp er i utgangspopulasjonen oppgitt i løpende nominelle kroner, men regnes om til lønnsdeflaterte størrelser i simuleringen og (som standard) også alt som kommer ut av simuleringen.

Yrkesdeltaking gitt ved pensjonsgivende inntekt, ny definisjon
I arbeid vil si at man har en arbeidsinntekt over en gitt grense (definert til om lag ett grunnbeløp i 1993 og deretter lønnsindeksert). Om man har begynt eller sluttet i løpet av såret avhenger på tilsvarende måte som før av om man har hatt tilstrekkelig arbeidsinntekt året etter og året før, men med visse tilleggsbetingelser for å rette opp åpenbare misforhold mellom arbeidsinntekt og yrkesdeltaking (gjelder blant annet etterbetaling av feriepenger). Personer med status 4 og en arbeidsinntekt over 1000 Nkr (1993-priser, deretter lønnsindeksert) blir definert som å ha lav arbeidsinntekt.

1: Er i arbeid, 2: Begynner å arbeide i løpet av året, 3: Slutter å arbeide i løpet av året, 4: Ikke vært i arbeid i løpet av året

Yrkesdeltaking gitt ved pensjonsgivende inntekt
J - har pensjonsgivende inntekt større enn 1200 Nkr (2001-priser), N - Ellers, x - Vilkårlig hvilken verdi.

J/N/x-koder i parentes angir inntekt året før, inneværende år og neste år.

1: Er i arbeid (JJJ), 2: Begynner å arbeide (NJJ) 3: Slutter å arbeide (JJN), 4: Ikke i arbeid (xNx), 5: Tilfeldig inntekt (NJN).

Antall år i siste status for yrkesdeltaking
Angir antall år i siste status for yrkesdeltaking, hvor kode 1 og 2 er slått sammen (yrkesaktiv) og kode 3, 4 og 5 er slått sammen (yrkespassiv), og hvor første år i ny status starter med verdi 1.

Vedlegg N: Monodevelop

Det anbefales sterkt å benytte et moderne integrert utviklingsmiljø, en såkalt IDE (Integrated Development Environment), når man skal jobbe med koden i MOSART. Dette har mange fordeler. Et slik miljø er både en kompilator og en svært kompetent editor, i tillegg til at det er mulig å eksekvere modellen inne i miljøet. Men det er for feilsøking miljøet virkelig skinner, idet en rekke nyttige verktøy tilbys til hjelp i dette arbeidet.

Utvikling, testing og debugging bør derfor foretas i monodevelop, men det er fortsatt mulig å utføre dette med manuelle metoder. Kildekoden befinner seg fremdeles i aktuell prog-katalog og kan oppdateres på tvers. Monodevelop er ikke spesielt godt egnet til bruk av multithreading. Dette begrenser muligheten til produksjonskjøringer, men er godt nok til andre formål. Produksjonskjøringer utføres derfor best som tidligere, i tilhørende **res-katalog**. Pass da på at modellen er kompilert (ved komp) først. En kort innføring i bruk av monodevelop er tilgjengelig fra https://www.monodevelop.com.

Klargjøring

Før modellen kan hentes inn i Monodevelop må det produseres en prosjektfil for den aktuelle modellvarianten. Dette forutsetter at en slik foreligger, avsnitt 3.4 beskriver hvordan denne opprettes. Et shell-skript som produserer prosjektfilen, 'newsolution', eksekveres slik:

```
/mosart/prog/navn-på-modellvariant/>newsolution [↵]
```

Dette resulterer i en ny fil, *navn-på-modellvariant.csproj*, som kan åpnes i monodevelop:

```
/mosart/prog/navn-på-modellvariant/>monodevelop navn-på-modellvariant & [↵]
```

Det er selvsagt også mulig å åpne filen manuelt fra menyvalgene i monodevelop. Så snart prosjektfilen åpnes i monodevelop opprettes det automatisk en solution-fil, *.sln*, som bør benyttes videre. Denne blir også med i listen over nylig åpnete løsninger i monodevelop.

Bruk av monodevelop

Når modellvarianten er åpnet i monodevelop vises Solution Pad helt til venstre. Her framkommer de filene som inngår i modellvarianten. Ved å dobbeltklikke på de åpnes de i editoren og kan redigeres. Filer med parametere (*.con) vises med et rødt kryss foran seg. Det angir at de ikke tas med i kompileringen, de kan likevel redigeres og blir lagret automatisk ved kompilering.

Progresjonen kan følges i sim.control som vanlig, men denne vises ikke i monodevelop. I vinduet External console skrives året som simuleres ut i forkant av dette (det skrives også til sim.errors). Alle log-filer blir skrevet ut til aktuell katalog som ved eksekvering ved job.exe. Siden monodevelop fungerer dårlig sammen med multithreading anbefales det å slå av dette, og også at det benyttes et så lite utvalg som mulig.

Resultatfiler produseres av monodevelop på samme måte som ved manuell eksekvering, men disse er i utgangspunktet ikke tilgjengelige fra Solution Pad. De kan imidlertid enkelt åpnes gjennom menyvalget «File – Open».

Vedlegg O: Introdusere nye kjennetegn

Dette vedlegget omtaler kort hvorledes nye personkjennetegn introduseres og hvordan modellen kan nyttiggjøre seg de. Beskrivelsen her gjelder for MOSART versjon 7.5, i senere versjoner kan endringer i arkitekturen innebære at noe av det som omtales er forandret. Prosessen belyses best ved å anvende et konkret eksempel. Det vil derfor vises hvordan det kan innlemmes at en person har fullført eller på et tidspunkt gått på sykepleiestudiet. Et fornuftig navn må oppgis for den nye variabelen, den vil derfor hete «has_attended_nursing_school». En datatype for variabelen må også bestemmes, det er naturlig at den er av type boolean (bool).

Endringer i person.cs

Aller først må den nye variabelen inkluderes i personobjektet. Dette er deklarert i klassen person i filen person.cs. Et stort antall personkjennetegn er allerede definert i denne klassen, slik at det først vil være fornuftig å lete opp en liknende variabel. Deretter defineres den nye variabelen på mer eller mindre samme måte som den eksisterende, blant annet ved å søke etter alle forekomster av den gamle variabelen. Dette gjøres enklest ved hjelp av Monodevelop. I MOSART-sjargong kalles dette å «haike» med variabelen.

Alle variable som omhandler et bestemt tema er i stor grad gruppert på samme sted i kildekoden. I eksempelet benyttes en utdanningsvariabel, slik at det bør søkes opp hvor i filen disse er deklarert. Deretter velges en av de for etterlikning, en opplagt kandidat her er attended_higher_education. Datatypen er bool, og som navnet antyder vil denne være sann dersom personen på et eller annet tidspunkt har vært under høyere utdanning. Dette er en variabel som er enkel, ikke har noe forløp og er uavhengig av pensjonssystem.

Den nye variabelen deklareres tilsvarende som for den eksisterende, og gjerne på linjen under:

```
public bool attended_higher_education; // Eksisterende variabel.
public bool has_attended_nursing_school; // Ny variabel.
```

Variabelen er nå deklarert, og da er det viktig at den blir nullstilt[33] ved resirkulering av personobjektet. Dette gjøres på samme måte som for de øvrige variable i metoden final_clear() i den blokken som tilsvarer datatypen (bool):

```
has_attended_nursing_school = false;
```

MOSART har en innebygget feilrapporteringsrutine[34] som skriver ut personkjennetegn, og den nye variabelen bør også innlemmes i denne. For oversiktlighetens skyld er det også her fornuftig å gruppere variable etter tema. Utdanningsvariable er sosioøkonomiske variable og skrives ut av

[33] Nullstilling handler om resirkulering av allokert internminne, og er en risikosport som kompliserer programmeringen, men som er (i det minste har vært) helt nødvendig for å kunne kjøre MOSART på hele populasjonen – og det går både på kjøretider (som ellers ville vært mange uker (ingen overdrivelse)) og stabilitet (simuleringen bryter sammen fordi operativsystemet / garbage collector ikke henger med). Glemmer man å nullstille en variabel, kan personen som overtar objektet risikere å arve verdiene for dette kjennetegnet, aka «sjelevandring». Resirkulering kan i stor grad kobles ut med recycle_objects i parameters.con, men kun for å kontrollere at resirkuleringen ikke påvirker resultatene.

[34] Feilrapporteringsrutinen består i at all informasjon om personen skrives til sim.errors, og er nyttig ved tung feilsøking.

metoden write_sosioeconomic_variables(). Altså bør den nye variabelen skrives ut av nettopp denne metoden og gjerne sammen med liknende variable, i dette tilfellet den variabelen vi «haiker» med:

```
message += Report(candidate_this_year, "candidate_this_year");
message += report(continues, "continues");
message += report(attended_higher_education, "attended_higher_education");
message += report(has_attended_nursing_school, "has attended a nursing school");
```

Noter her forskjellen mellom metodene «report» og «Report». Førstnevnte skriver ut tegnene «, » før variabelbetegnelsen (has attended a nursing school) etterfulgt av verdien av variabelen (true/false). Sistnevnte gjør ikke dette og er derfor å foretrekke i begynnelsen av linjen. Legg også merke til at metodene report/Report har to parametere, der den første er den variabelen hvis verdi skal skrives ut mens den andre er den teksten som skal skrives ut til feilrapporten. Den behøver ikke være det samme som variabelnavnet, og kan være hva som helst, men det kompliserer gjenfinning dersom teksten er noe annet enn dette.

For rapporteringsformål bør det om mulig[35] opprettes en metode som returnerer hvorvidt en person hadde gått på sykepleieskole ved en gitt alder. Dette er nyttig i både utskriften av modellpopulasjonen og i tabellprogrammet, da man kan hente ut data for tidligere år. En rekke slike metoder eksisterer allerede og er gruppert etter tema:

```
public bool has_attended_nursing_school_at(int actual_age)
{
   if (use_current_status(actual_age) || !has_attended_nursing_school)
      return has_attended_nursing_school;
   else if (use_retrospective_data(actual_age, g.standard_hfu_age))
   {
      bool returnvalue = false;
      int age_i = actual_age;
      while (!returnvalue && age_i >= g.standard_graduation_age_secondary_school)
      {
         if (hfu == g.bachelor_nursing || igu == g.bachelor_nursing)
            returnvalue = true;
         else
            age_i--;
      }
      return returnvalue;
   }
   else
      return false;
}
```

[35] MOSART tar ikke vare på historikken for alle kjennetegn.

Tilordning av verdi

Når variabelen er etablert må den også tilordnes en verdi. I eksemplet benyttes en variabel som ikke har noe forløp, den kan være sann allerede i utgangsåret, bli det i løpet av simuleringsperioden eller aldri være det. Aller først må kjennetegnet tilordnes dersom noen i populasjonen tilfredsstiller betingelsen.

Utgangspopulasjonen leses inn av read_initial_population.cs, og et søk etter den variabelen som etterliknes (attended_higher_education) avdekker at den tilordnes i metoden backdate_longitudinal_data_on_education(). Den nye variabelen tilordnes derfor rett etter:

```
if (g.education[p.igu].level > g.level_secondary_school ||
    g.education[p.hfu].level > g.level_secondary_school)
    p.attended_higher_education = true;

if (p.hfu == g.bachelor_nursing || p.igu == g.bachelor_nursing)
    p.has_attended_nursing_school = true;
```

Videre søk etter variabelen som etterliknes viser at utdanningsvariable også tilordnes av metoden subworker_hfu() i loop_historical_data.cs, slik at tilordningen vist over også må etableres der. Her defineres bare fullførte utdanninger slik at uttrykket blir enklere:

```
if (p.hfu == g.bachelor_nursing)
    p.has_attended_nursing_school = true;
```

Igangværende utdanninger tilordnes av metoden subworker_subjects i samme fil:

```
if (p.igu == g.bachelor_nursing)
    p.has_attended_nursing_school = true;
```

Personer kan også oppnå en utdanning i løpet av simuleringen. En ny fullført utdanning tilordnes av metoden assign_new_hfu_version_3() i read_education.cs:

```
if (new_hfu == g.bachelor_nursing)
    p.has_attended_nursing_school = true;
```

Det må også fanges opp når personer starter på en utdanning som tilfredsstiller kriteriene i simuleringsperioden. Dette tilordnes i metoden assign_new_igu_version_3(), også i read_education.cs:

```
if (new_igu == g.bachelor_nursing)
    p.has_attended_nursing_school = true;
```

Et siste sted personkjennetegn tilordnes er i read_simulated_population.cs. Den metoden der det er aktuelt å legge inn det nye kjennetegnet heter initialize_education(). Et søk etter

attended_higher_education avdekker at den blir tilordnet to steder i denne metoden, det nye kjennetegnet må tilordnes der på samme måte.

Endringer i table_base.cs

Det vil være ønskelig å ha muligheten til å inkludere nye personkjennetegn i tabellproduksjonen, og filen som håndterer nettopp dette heter table_base.cs. Enkle kriteria defineres her i metoden find_single_criterion(). Også i dette tilfellet er det fordelaktig å etterlikne en eksisterende variabel, for så å kopiere deklarasjonen av denne for den nye. Deretter oppdateres navnene:

```
else if (parameter_root == "attended_higher_education")
   new t_criterion_attended_higher_education(base_parameters);
else if (parameter_root == "has_attended_nursing_school")
   new t_criterion_has_attended_nursing_school(base_parameters);
```

Definisjonen som nå er opprettet peker mot en klasse som enda ikke eksisterer, t_criterion_has_attended_nursing_school. Den må derfor også opprettes. Klassedeklarasjoner er gruppert etter tema, en av de er utdanning, og deklarasjonen av den nye klassen bør plasseres her:

```
public class t_criterion_has_attended_nursing_school : t_base_criterion
{
   public t_criterion_has_attended_nursing_school(t_base_parameters base_parameters) :
base(base_parameters)
   {
      set_criterion_labels("has/is attended/ing nursing school", "AttNuSc");
      filename_label = preceding_not + "attended_nursing_school";
   }
   public override bool include(person p)
   {
      if (p.has_attended_nursing_school_at(corrected_age(p)))
         return actual_true;
      else
         return actual_false;
   }
   public override enum_availability availability()
   {
      return enum_availability.longitudinal_data;
   }
}
```

Dersom koden kopieres fra den klassen det haikes med er det viktig å endre alle tekster slik at den nye variabelen ikke forveksles med denne. Merk at andre parameter til metodekallet set_criterion_labels er et kortnavn på 7 tegn som beskriver variabelen. Kortnavnet bør være unikt,

og benyttes om nødvendig som kolonneoverskrift i tabellene som skrives ut, men da supplert med en forklaring i toppen av tabellen[36].

Endringer i model_population_file.cs og model_population_file.con

Det vil som regel være ønskelig å inkludere den nye variabelen i modellpopulasjonen dersom det er valgt at denne skal skrives ut. Aller først må den opprettes som en parameter i model_population_file.con:

```
attended_higher_education                0        ! LV
has_attended_nursingschool               1        ! LV
```

Deretter må det gjøres endringer i model_population_file.cs som håndterer denne. Først deklareres det en klasse som kan håndtere parameteren i metoden initialize_variables():

```
class t_has_attended_nursing_school : t_base_variable_bool
{
   public override bool x(person p)
   {
      return p.has_attended_nursing_school_at(corrected_age(p));
   }
}
```

Deretter initialiseres et objekt for den nye variabelen i metoden initialize_variables():

```
add_variable <t_attended_higher_education> (rp, variable.is_longitudinal);
add_variable <t_has_attended_nursing_school> (rp, variable.is_longitudinal);
```

Endringer i control_demographic_data.cs og control_non_demographic_data.cs

Dersom en personvariabel håndteres mange steder, flere variable beskriver ulike fasetter av et kjennetegn eller det er vanskelig å oppdage feil, vil det ofte være fornuftig å kontinuerlig monitorere verdiene av disse. Det er etablert metoder som håndterer dette i control_demographic_data.cs og control_non_demographic_data.cs. Disse metodene eksekveres som en del av simuleringsløkken, det vil si hvert eneste simuleringsår.

Variabelen anvendt i eksemplet er en ikke-demografisk variabel slik at den kontrolleres best i sistnevnte fil. Metoden som benyttes til dette er worker_control_non_demographic_data, og i denne legges det til en kontroll på at det ikke eksisterer personer under 18 år som er utdannet sykepleier i populasjonen. Dette plasseres helst sammen med de andre kontrollene som gjøres for utdanningsvariable:

```
if (p.age < 18 && p.hfu == g.bachelor_nursing)
   report_error(p, p.hfu, p.age, "Person is too young to have this education.");
```

[36] Noen ganger er det ikke hensiktsmessig å ha et kortnavn på maksimalt 7 tegn, men da finnes det andre oppsett for dette.

Dette vil si at dersom det i løpet av simuleringsperioden skulle forekomme personer som tilfredsstiller de kriteria som er spesifisert, her sykepleiere yngre enn 18 år, så vil en stor mengde informasjon om vedkommende person skrives ut til filen sim.errors.

Merk at rapporteringsmetoden som kalles, report_error(), kan ha inntil 4 parametere inkludert en forklarende tekst. Den første skal alltid være objektet til den personen informasjonen skrives ut for og den siste den tilhørende teksten. De øvrige kan være hva som helst bare de uttrykker en tallverdi.

Endringer i update.cs

Denne filen inneholder noen metoder for årlig nullstilling av variable. Dersom det er ønskelig å få utført dette for nye variable plasseres dette best her. Eksemplet som benyttes i dette vedlegget er en irreversibel variabel, og er derfor uaktuell her.

Endringer i sim_migration.cs

I denne tilordnes utdanning for immigranter og de som re-immigrerer i metoden assign_education(). Altså kan det også her være nødvendig å oppdatere utdanningsvariable. Dette gjøres tilsvarende som for backdate_longitudinal_data_on_education() i read_initial_population.cs.

Vedlegg P: Legge til nye pensjonsregler

Dette vedlegget viser hvordan nye pensjonsregler innlemmes i MOSART, konkretisert ved å ta utgangspunkt i det nye personkjennetegnet («has_attended_nursing_school») som ble introdusert i Vedlegg O. Dette kan følges opp via flere linjer. En er at sykepleiere har særaldersgrense som er/vil bli avviklet og erstattet av noe annet. Dette kan gjøres ved å innføre et slitertillegg i form av en særskilt pensjonsprosent for (her) sykepleiere, til erstatning for dagens sats på 18,1 prosent i ny folketrygd.

Akkurat som for det å legge inn et nytt kjennetegn vil det enkleste være å finne en eksisterende pensjonsregel som ligger nærmest på den nye regelen, både i tema, omfang og hva den skal gjøre, for deretter å «haike» med denne regelen eller settet av regler.

Endringer i pension_parameters.con

Den mest aktuelle regelen å haike med er «accrual_percent_adjusted» i «pension_parameters.con», som allerede er en regel som kan erstatte den omtalte pensjonsprosenten for alle fra et gitt år (og eventuelt med et høyere opptjeningstak). Det er ingen grunn til å lage en ny parameterfil for denne pensjonsregelen, slik andre regler er separert ut i afp.con, public_occupational_pension.con og private_occupational_pension.con. Det er likevel grunn til å omtale muligheten her[37].

I pension_parameters.con søkes det etter 'accrual_percent_adjusted', og på linjene etter legges følgende tre regler inn[38]:

```
toiler_pension                  0       ! 0: No, 1: Nurses
toiler_pension_first_birth_year 1964    !
toiler_pension_accrual_percent  0.22    ! Replaces ordinary accrual_percent
```

Den første regelen er en hovedbryter. Man kan slippe unna denne hovedbryteren ved å pakke den inn i en av de andre reglene, for eksempel ved å sette 'toiler_pension_first_birth_year' til et så høyt tall at regelen aldri kommer til anvendelse. Det er likevel to store fordeler med å ha en hovedbryter; den første er at man slipper å nullstille/overskrive veldig spesifikke verdier på de andre reglene. Man kan da lettere hente fram igjen disse verdiene ved at de står på linjene under. Den andre fordelen er man lettere kan legge til nye versjoner av dette settet av regler uten å måtte gjøre strukturelle endringer i parameterfilen, det vil si legge til nye parametere og/eller gjøre endringer i eksisterende parametere som gjør filen uleselig for andre modellversjoner (man kan lettere hente inn brukergrensesnittet/oppsettet fra tidligere kjøringer).

Den andre regelen angir fra (og med) hvilket årskull regelen skal gjelde.

Den tredje regelen angir hvilken pensjonsprosent som skal gjelde for den angitte gruppen (personer som har gått på sykepleiestudiet).

Innlesing

Dette må så legges inn i 'read_pension.cs', i tre runder. Første runde er deklarasjon, initiering og innlesing av reglene, som er relativt generell på tvers av alle (pensjons)regler i MOSART. Den andre

[37] Dersom nye parameterfiler opprettes, bør dette reflekteres ved å oppdatere dokumentasjonen i 'change_parameters.con' med kortnavnet på den nye filen.

[38] «Sliter» her oversatt til det engelske ordet «toiler». Det er ikke lett å finne et fullgodt ord for sliter, det skyldes trolig at «sliter» på norsk har vokst til å bli et sterkt ladet (positivt) normativ begrep, som «alle vet hva betyr», men ingen kan gi en definisjon på, og som ikke har noe motsvar i den anglo-amerikanske verdenen.

runden er hvordan regelen anvendes i kildekoden, som er litt mer spesifikk for hvert sett av regler. Den tredje runden er kvalitetssikring.

Det første som gjøres er å søke etter forekomster av den regelen det haikes med, 'accrual_percent_adjusted' i 'read_pension.cs'. Første treff er deklarasjonen av reglene, og på linjene etter dette legges de nye variablene fra 'pension_parameters.con' inn:

```
public double [] accrual_percent_adjusted;
public enum enum_toiler_pension { no = 0, nurses = 1 };
public enum_toiler_pension [] toiler_pension;
public int [] toiler_pension_first_birth_year;
public double [] toiler_pension_accrual_percent;
```

Alle reglene er deklarert som arrays (vektorer) med (minst) en dimensjon; pensjonssystemer. Hovedbryteren her kan deklareres som «bool», som er litt enklere både i deklarasjonen og senere i håndteringen. Det beste er likevel å deklarere den som en særskilt deklarert «enum», da det lettere gir mulighet for å legge til nye varianter senere. De to andre reglene er deklarert som henholdsvis heltall (int, fødselsår) og flyttall (double, pensjonsprosenten).

Deretter skal de nye reglene initieres. Søk etter variabelen det haikes med avdekker at dette gjøres i metoden 'initialize_pension_reform_of_2010()'. Reglene er vektorer, og de må opprettes som sådan:

```
initialize_rule_array(out accrual_percent_adjusted);
initialize_rule_array(out toiler_pension);
initialize_rule_array_as_year(out toiler_pension_first_birth_year);
initialize_rule_array(out toiler_pension_accrual_percent);
```

Metoden 'initialize_rule_array' oppretter de ulike reglene som en vektor (array), hvor første dimensjon er antall pensjonssystemer (no_of_pension_systems+1 siden pensjonssystem 1 har tallverdien 1, og ikke 0, som vanligvis er standard i C# (null-basert). En grunn til det er at ps=0 er reservert for andre formål). Flere dimensjoner i en pensjonsregel er litt mer komplisert, men det er mange eksempler på dette i MOSART, se spesielt avtalefestet pensjon/afp. Merk at 'toiler_pension_first_birth_year' her initieres som '_as_year', som betyr at default-verdi settes til 'eternity', som for praktiske formål i MOSART er sim_end+max_age+1.

Verdiene som er spesifisert for de nye reglene skal leses inn fra 'pension_parameters.con'. Videre søk etter variabelen det haikes med avdekker at dette skjer i metoden 'read_pension_reform_of_2010()'. Her legges følgende inn:

```
read_parameter.as_double(ps, accrual_percent_adjusted, "accrual_percent_adjusted");
read_parameter.as_rule(ps, toiler_pension);
read_parameter.as_year(ps, toiler_pension_first_birth_year, "toiler_pension_first_birth_year");
read_parameter.as_double(ps, toiler_pension_accrual_percent,
"toiler_pension_accrual_percent") ;
```

Det er ulike metoder for å lese inn parameterne avhengig av hva innholdet er. Det er lagt inn en del sjekker for feil, blant annet er det er krav at alle parametere forekommer en og bare en gang i sin parameterfil. Det er også mulig å lese inn parameteren som:

```
toiler_pension_accrual_percent[ps] = read_parameter.as_double(ps,
"toiler_pension_accrual_percent");
```

Merk at innlesingen av alle pensjonsregler skal ha 'ps' som første argument. Det finnes andre versjoner av disse innlesingsmetodene som er uten dimensjonen pensjonssystem, for eksempel 'parameters.con', men disse versjonene av metodene skal ikke være mulig å få brukt ved en feiltakelse her. Glemmer man å skrive 'ps, ' så får man dessverre en veldig kryptisk feilmelding ved kompilering.

Hovedbryteren 'toiler_pension' leses inn som 'as_rule', som betyr at hvis den har (eksakt) samme navn i 'pension_parameters.con', så slipper man å angi navnet som tredje parameter. En fordel med å lese den inn som 'as_rule', er at hvis man velger en ulovlig verdi får man en litt mer intelligent feilmelding, blant annet et forslag om hvilke lovlige verdier som er tilgjengelig, med litt forklaring.

De to siste kallene på 'read_parameter' benytter en tredje parameter. Det er overkommelig å avlede navnet på variabelen fra denne spesifikke typen (enum_...) den er deklarert som (ved å gi typen (nesten) samme navn som variabelen). Dette er ikke mulig for generiske typer som int, double og bool, som betyr at man må gjenta variabelnavnet to ganger, først som variabelen, og deretter pakket inn i "" som en tekststreng.

Regelen 'toiler_pension_first_birth_year' leses inn som 'as_year', som betyr at hvis verdien er negativ (typisk -1), så settes parameterverdien til omtalte 'eternity'. Regelen 'toiler_pension_accrual_percent' leses inn som 'as_double', de andre standardvariantene er 'as_int' (heltall, utenom årstall), 'as_bool' (boolske variabler) og 'as_string' (tekst).

Anvendelse

Regelen 'accrual_percent_adjusted' anvendes bare ett sted i modellen, og det er fordi den er pakket inn i metoden 'read_accrual_percent' i 'read_pension.cs' som kalles fra fire ulike steder i kildekoden ellers. Det betyr at man slipper unna med å endre kildekoden i denne spesifikke metoden, og så sørger metodekallene for at dette blir distribuert til resten av beregningene.

Sliterpensjonen skal gjelde for personer som tilfredsstiller det nye kjennetegnet som ble innlemmet ('has_attended_nursing_school') og er født i et år som omfattes av pensjonsordningen. Begge disse egenskapene er definert i personobjektet 'p'. Metoden 'read_accrual_percent' har i sin nåværende form ikke tilgang på personobjektet, slik at metodesignaturen må endres slik at objektet blir med som parameter i denne. I dette tilfellet er det mest hensiktsmessig å utvide denne til også å omfatte personobjektet dersom det er tilgjengelig fra der metoden kalles, i motsatt fall settes parameteren til 'null'. Parameteren blir dermed valgfri[39]. Metoden endres derfor til følgende:

[39] Valgfrie parametere defineres ved å tilordne de en verdi når metoden deklareres. Denne verdien blir benyttet dersom metoden kalles uten parameteren som argument. Den opprinnelige definisjonen til metoden her er read_accrual_percent(int ps, int year). Denne utvides til read_accrual_percent(int ps, int year, person p = null). Det innebærer at metoden fortsatt kan kalles uten å angi et personobjekt, men dette vil da være tomt (= null) når det som er implementert av metoden eksekveres.

```
public double read_accrual_percent(int ps, int year, person p = null)
{
   if (toiler_pension[ps] == enum_toiler_pension.nurses &&
      p != null &&
      p.birth_year >= toiler_pension_first_birth_year[ps] &&
      p.has_attended_nursing_school)
      return toiler_pension_accrual_percent[ps];
   else if (year > accrual_adjustment_year[ps])
      return accrual_percent_adjusted[ps];
   else
      return accrual_percent[ps];
   }
```

Da gjenstår det å kvalitetssikre det nye personkjennetegnet og den nye pensjonsregelen. Se Vedlegg Q for anbefalinger om dette.

Vedlegg Q: Kvalitetssikring av resultater

Etter at det er foretatt modellendringer må dette kvalitetssikres. Det gjøres i tre trinn, der leting etter feil utgjør en vesentlig del.

Leting etter åpenbare feil

På jakt etter åpenbare (og hyppige) feil kan man skrive ut modellpopulasjonen med relevante variabler, og deretter liste ut personer for å sjekke om livsløpene er riktige. Et lite shellscript 'skriv_person', som følger med hver simulering, viser en og en person på skjermen, og gjør det mulig å selektere og bearbeide filen med lite overhead, se vedlegg. Det er også enkelt å få til dette med Monodevelop.

Det kan være aktuelt å regne på resultatene i for eksempel Excel, der det er en klar sammenheng mellom for eksempel inntekt og senere pensjon. NB! Det er ikke lov å transportere ID-numre ut av linux-sonen, og unngå for all del lagring av slike data andre steder.

Leting etter subtile feil

På jakt etter åpenbare og sjeldne feil kan også modellpopulasjonen komme til anvendelse, men da må man ha noen antakelser om hvem feilen treffer, slik at man kan få selektert ut disse personene.

Neste trinn blir å ta ut aggregerte tabeller, og der det er mulig å anvende ulike versjoner (pensjonssystemer) av de nye reglene i samme kjøring. Man skal da få ut skarpe anslag på hvordan de nye reglene slår ut. Det er høyst ønskelig at alle reglene spesifiseres med sine ulike verdier, eventuelt noen ulike verdier på kontinuerlige variabler.

Det første man kan sjekke er timingen. Kommer endringen i det året den skal; effekter som (kvalitativt) kommer for tidlig (eller sent) indikerer FEIL, og bør (må) oppklares. Man kan også sjekke om innfasingen kommer i forventet tidsrom.

Man kan sjekke om endringen kommer på riktige grupper; kjønn er en grei dimensjon, men andre variabler er også aktuelle.

Til slutt kan man også ha noen forventninger om styrken i effektene.

Formidling

De gangene MOSART er riktig i utgangspunktet og man treffer på nykodingen, så vil feilsøkingen og kvalitetssikringen være en relativt enkel oppgave. Det er imidlertid en (negativ) og lang og tung hale når dette ikke stemmer.

Når man kjenner MOSART så er det ofte overraskende lett å koke sammen en forklaring som forklarer nesten hva som helst som ramler ut av modellen. Det er så mange effekter som trekker i ulike retninger at det lar seg gjøre. Dette er trolig den verste feilen man kan gjøre; bortforklare en (mulig) feil som noen i departementet har påpekt, og som man deretter har avfeid med en feilaktig forklaring.

Har man et kontraintuitivt resultat, så er det viktig å ha et åpent sinn på at dette både kan være feil/svakheter i modellen og/eller en reell effekt man ikke ser umiddelbart. På jakten etter en forklaring er det viktig å forsøke å kvantifisere de faktorene man trekker inn, og faktisk sjekke at framskrivingen faktisk er det man påstår (økende utdanningsnivå kan være en slik faktor noen ganger, sjekk da at utdanningsnivået faktisk øker i det tidsrommet/årskullene man omtaler, og at økningen er sterk nok til å forklare det som skjer).

Metodisering

Det kan ikke overdrives hvor viktig det er å bryte ned funksjonalitet i mindre metoder for vedlikeholdet og utvidelsene av MOSART. Den soleklart enkleste måten å legge inn noe nytt er å kopiere over kildekode fra et annet sted; man har da både kildekoden der man står og jobber nå, og man står helt fritt til å endre kildekoden uten å være bekymret for hva som skjer der man kopierte kildekoden fra. Ofte er det nødvendig å ta denne omveien, fordi utvikling av ny kildekode er i seg selv krevende. Det er da viktig å gå tilbake senere å sjekke om det man har gjort kan (og bør) samles i en felles metode. Hvis man ikke gjør det vil man i senere runder måtte sjekke rundt mange steder i kildekoden hvis man skal revidere en pensjonsregel, ofte med fare for at man overser steder man må sjekke ting.

Det er svært mange grunner til at man bør bruke metoder, og dette er i for liten grad gjennomført i MOSART.

Vedlegg: Skrive ut populasjonsfilen på lesbar form

```
#!/bin/sh

awk '{if (f1 != $1)
    {
      if (med == 1)
      {
        print f1;
        for (i = 0; i < t; i++) print g[i];
        print "\f";
      }
      f1 = $1;
      med = t = 0;
      g[t] = "      ID aar <Kjennetegn1> <Kjennetegn2> ...";
      t++;
    }
    g[t] = $0;
    t++;
    if (<Kriterier for å bli med i utskriften>)
    {
      med = 1;
    }
   }' population
```

Vedlegg R: Vedlikehold av parametere

En rekke av parameterne som inngår i MOSART er estimert på grunnlag av utgangspopulasjonen i modellen og ofte med hjelp av modellen selv. Mange av disse estimeringene ligger i 'adapt_population', i spesialuttak av tall i awk fra modellpopulasjoner eller i eldre spesialversjoner av MOSART. Noen av estimeringene ligger permanent i kildekoden, men bidrar da ofte til å gjøre kildekoden til den egentlige simuleringen mer uoversiktlig (på et nivå at det er et alvorlig problem).

Det kan være et poeng å samle alle disse rutinene permanent i MOSART og samtidig på siden av selve simuleringsmodellen. Modulen 'estimate_base.cs' og underliggende filer av typen 'estimate_*navn*.cs' er et forsøk på å få til dette. I første omgang er det bare de permanente estimeringene i modellen som er lagt over hit, men tanken er å hente inn alle tidligere eksterne estimeringer til dette oppsettet. Over tid er det også en ambisjon å få inn mer avanserte estimeringsmetoder (logistisk regresjon via R), slik at alle (de fleste) parameterne i MOSART kan vedlikeholdes fra modellen selv.

Under følger en kort beskrivelse av endringene i modellen ellers og litt mer på selve estimeringsmodulene. Det er trolig nyttig å se på de angitte filene for å forstå vedlegget.

Kompileringsskriptet 'komp' er nå splittet i to, hvor det er laget en liste 'komp.modules' hvor hver linje inneholder en modul i MOSART. Grunnen er at antall moduler nå kommer til å vokse betydelig og det blir umulig å holde oversikten hvis alle filene skal være angitt på én linje. Det er mulig å legge til kommentarlinjer i 'komp.modules' og filene kan komme i en vilkårlig rekkefølge (selv om de nå står i alfabetisk rekkefølge).

Det er et mål at *estimate* skal etterlate seg minst mulig kildekode inne i resten av modellen, men noen spor må det bli. Den ene forekomsten er at *estimate* må deklareres i hovedprogrammet 'model.cs', og grupper av estimeringsmoduler må kalles på egnede steder i simuleringsløkken. Det er mest hensiktsmessig å legge estimeringsrutinene rett etter der de anvendes i modellen. MOSART bruker sekvensiell simulering i en på forhånd fastlagt rekkefølge. Ved å legge estimeringene hit vil modellen ha hentet inn alle data som angår dette estimatet, samtidig vil ikke de andre kjennetegnene ha blitt oppdatert ennå. Man får da med seg utfallet og riktig status på forklaringsvariablene, men må i noen tilfeller tilbakestille verdien for det kjennetegnet man ser på.

Den andre forekomsten av *estimate* i kildekoden ellers er at noen ganger er dette en reestimering av enkelte deler av en fil som allerede inngår i simuleringen, og da må estimate-modulen ha tilgang til navnet på filen som er brukt og det kan også være nødvending å skru av justeringsalgoritmene i simuleringen. Dette krever litt interaksjon mellom *estimate* og kildekoden ellers.

Filen 'estimate_base.cs' er moderklassen i dette systemet og samtidig også utgangspunktet for '*estimate*' slik den inngår i hovedprogrammet 'model.cs'. Herfra styres arbeidet med estimeringene, og de ulike modulene er samlet i grupper. Hver enkelt modul kunne vært kalt direkte fra 'model.cs', men en gruppevis kall i et mellomsjikt gjør det lettere å vedlikeholde dette opplegget *utenfor* simuleringsmodellen. Videre inneholder 'estimate_base.cs' en rekke mindre og litt større felles metoder for å håndtere estimere og skrive ut estimatene.

Spesifikt er lineær regresjon i en variabel lagt inn, og det vil være en relativt triviell oppgave å legge lineær regresjon i flere variabler. Denne regresjonen er imidlertid avgrenset til punktestimater for parameterverdiene, og det vil være en langt større jobb å rapportere standardavvik, signifikansnivåer og mer avanserte mål på problemer i det at man bruker lineær regresjon. Logistisk regresjon, spesielt med flere utfall enn to og med mange forklaringsvariabler, ligger trolig utenfor det som er fornuftig at vi prøver å programmere selv. Det er her R kommer inn.

Estimeringene styres gjennom filen 'estimate.con'. På toppen er en hovedbryter 'include_estimation' som gjør at alle estimeringene utelates. Hver fil som skal estimeres har en bryter som heter kun *navn* på formatet 's:*i*', hvor 'i' kan anta verdiene 0:ingen estimering, 1:estimer og 2:estimer-og-anvend-estimatene. Fordelen med 's:' er at det blir enklere å søke opp alle brytere, og spesielt kan man nullstille alle brytere med standard søk-erstatt som inngår i nesten alle redigeringsverktøyer (uten 's:' vil det være vanskelig å skille ut hva som er brytere og hva som er andre boolske parametere). I noen tilfeller er det mulig å la simuleringen anvende de nye estimatene direkte i samme simulering (s:2). Det er to andre obligatoriske parametere, '*navn*_start_year' og '*navn*_end_year', som styrer hvilke år man skal estimere over. Det er ingen kontroll på om '*navn*_end_year' er mindre eller lik siste år med historiske data, som betyr at man kan bruke modellen til å estimere på simulerte data. De andre parameterne i 'estimate.con' er tilpasset hver enkelt modul, ofte en avgrensning til relevante aldre og/eller behov for glatting der hendelsene blir så sjeldne at støyen overdøver all struktur.

Ut av estimeringen kommer en (eller flere) filer av typen 'sim.estimate.*navn*'. Hver av disse filene innledes med standard dokumentasjon av viktige egenskaper så som navn på kjøring, når dette ble gjort, av hvem, modellnavn, utgangspopulasjon, opprinnelig filnavn og estimeringsperiode. Filene skal normalt være helt klare til bruk, og kan av den grunn også leses inn av simuleringsmodellen i samme kjøring (se s:2 over).

Det er lagt opp til av hver estimeringsmodul skal håndtere en (eller en liten gruppe av like filer). Dette gir mange moduler, men til gjengjeld blir hver modul på noen hundre linjer, mer homogen og vil være radikalt enklere å håndtere med de redigeringsverktøyene vi har til rådighet. Det er noen estimeringer som allerede er lagt over.

I noen tilfeller er parameterne estimert med banale metoder som tabeller med overgangssannsynligheter (eller lineær regresjon) og dette kan allerede hentes inn i *estimate*. Et (og foreløpig eneste) eksempel er 'estimate_fertility.cs' som estimerer fruktbarhetsratene som brukes på individnivå i simuleringen (antall fødsler bestemmes fortsatt av befolkningsframskrivingene). Modulen 'estimate_household.cs' står for tur og er inntil videre et tomt skall.

En annen gruppe er reestimering av alderseffekten i spesielt viktige overgangssannsynligheter (uførhet, arbeidstilbud, kobling arbeidsstyrke-inntektstakere (lfpr_by_lme), ha arbeidsinntekt og inntektsnivå, hvor sektor står for tur). Modellen reestimerer dummyene for alder (x kjønn) under forutsetning av at de andre parameter-estimatene er korrekte. Man kan da bruke beregnede overgangssannsynligheter, beregne nødvendig korreksjon i konstantleddet med en liten iterasjon, og legge denne korreksjonen tilbake i aldersdummyen. Tidligere ble disse korreksjonen skrevet ut til en egen fil ('adjust_age_effect_...'), og lest inn ved siden av det opprinnelige estimatet, gjerne etter en komplisert etterjustering i et regneark. Nå blir den opprinnelige filen skrev ut på nytt, hvor aldersdummyene er erstattet, og hvor *nye* kommentarlinjer starter med '*> ', slik at neste reestimering kan skrelle vekk de nye kommentarene. Det er mulig at disse reestimeringene kan bygges ut til flere variabler, men det blir kanskje for komplisert.

En tredje gruppe er uttak av rådata til videre bearbeiding, dels fordi ikke alt er ferdig, dels fordi det av og til ligger skjønn bak reestimeringen som er vanskelig å operasjonalisere i en algoritme, se for eksempel 'estimate_retirement.cs'. Merk at en del av parameterne hentes ut via standard tabellprogram, og det er trolig mest hensiktsmessig at vi fortsetter med det (se for eksempel 'output.time_series.doc').

Vedlegg S: Finne feil i anvendelsen av trekkemetoder

Dette vedlegget omtaler kort prosessen for feilsøk i trekkemetodene i MOSART, i første rekke knyttet til privat tjenestepensjon. Mye av utfordringene omkring trekkemetodene i modellen knytter seg til at vi samtidig benytter parallell prosessering.

Først litt C#-matematikk

En påstand i (ordinær) matematikk er at faktorenes orden er likegyldig når det kommer til addisjon og multiplikasjon. Dette gjelder ikke for flyttall i C#. Ta et enkelt regnestykke:

a x b x c

Poenget er at C# her først multipliserer sammen a og b, og må runde av resultatet før den multipliserer med c og runder av på nytt. Effekten av avrundingen avhenger av rekkefølgen, og vil normalt være liten, som oftest mikroskopisk.[40] Problemet er at MOSART bruker stokastisk simulering på diskrete hendelser som påvirker hverandre. Disse små forskjellene vil før eller senere påvirke en hendelse den ene eller andre veien. Og når det først har skjedd, vil alt som kommer etterpå bli annerledes, litt slik virkeligheten også fungerer.[41]

Parallell prosessering

Fram til begynnelsen av dette århundret ble datamaskinene raskere fordi prosessorene ble mindre og mer effektive, men dette stoppet opp fordi man nådde fysiske grenser for hvor tynne en elektrisk ledning kan bli før den slutter å overføre et signal på en robust måte[42]. Når datamaskiner blir raskere er det fordi tunge oppgaver løses av flere prosessorer i parallell. Vi har implementert slike metoder i modellen, med et visst hell, men det er samtidig utfordrende å få dette til å virke på en slik måte at det ikke påvirker resultatet av simuleringen. Nedenfor følger en beskrivelse av opplegget i MOSART, slik at man kan forstå implikasjonene for trekkemetodene.

C# har to opplegg for parallell prosessering, 'Thread' og 'Parallel'. 'Thread' krever mye kildekode, men er i MOSART pakket inn i hjelpeklasse som bidrar til å gjøre det adskillig enklere, men fortsatt litt knotete. 'Parallel' har en langt enklere syntaks, men er vanskeligere å kontrollere og trolig mindre effektiv enn 'Thread' (dette er en generell egenskap i C#, språket har mange fiffige metoder som gjør ting som ellers ville krevd masse kildekode, men som er negativt for kjøretidene og stabiliteten i tunge simuleringer). Moduler[43] som brukes sjelden benytter ofte 'Parallel', ellers er det 'Thread' som er hovedregelen.

I en modul med parallell prosessering blir det opprettet ***tråder*** som for alle praktiske formål er egne jobber på maskinen, som blir tildelt ***oppgaver*** den beregner. Når oppgaven er avsluttet, kan tråden blir tildelt en ny oppgave, eller bli avsluttet hvis det i denne runden ikke er flere oppgaver. En oppgave kan være alt fra å simulere alle menn (hvor en annen tråd tar alle kvinner), til å beregne én sannsynlighet for én person. En tråd vil få tildelt tid på en kjerne (prosessor), og dette skjer genuint stokastisk avhengig av hvordan de ulike trådene går i veien for hverandre og konkurrerer om kjøretid og andre knappe faktorer i maskinen.

[40] Standard flyttall i C# (double) har 15-16 reelle desimaler.
[41] Trolig bare enda sterkere; mannen som stod ved siden av Hitler under ølkuppet i 1924 døde angivelig av et geværskudd fra langt hold under kaotiske forhold, som tilsier at det like gjerne kunne har truffet hovedpersonen.
[42] Dette fenomenet er i faget informatikk kjent som «The power wall».
[43] Modul er her et steg i simuleringen som går gjennom (hele) befolkningen og beregner/simulerer en egenskap, før den går videre til neste modul/steg.

Det er også mulig å dele ut oppgavene helt rigid (for eksempel at en tråd tar alle menn, den andre tråden tar alle kvinner), eller fleksibelt hvor hver tråd plukker opp neste oppgave i køen når den er ferdig med den forrige. 'Parallel' har fleksibel tildeling, og måten vi har lagt inn 'Thread' er også slik. Grunnen er at ved en rigid tildeling, kan en tråd av ulike grunner ha blitt nedprioritert, og dermed bli ferdig lenge etter de andre trådene. Og siden simuleringen ***må*** vente til alle trådene er ferdige før den kan fortsette videre til neste modul/steg i simuleringen, blir hele simuleringen forsinket. Det tar tid å tildele en oppgave, og det blir derfor en avveining hvor stor en oppgave bør være; store oppgaver øker risikoen for at en tråd blir sent ferdig, eller at det ikke er oppgaver til alle (mulige) tråder; små oppgaver vil kreve mye tid i form av administrasjon. I MOSART ligger alle personene i lister etter

reg.status[44] x kjønn x ettårig alder

Dette gir minst 150 oppgaver i jevnbyrdig størrelse (hvert årskull av bosatte personer etter kjønn i alderen 16-91 år), og en mengde mindre oppgaver (alle de andre). Dette er derfor normalt utgangspunktet for parallell prosessering i MOSART, en dynamisk tildeling av personer etter reg.status x kjønn x alder, hvor de største årskullene deles ut først. Dette opplegget fungerer når individet er gjenstand for simulering, andre tilnærminger må velges når det interaksjoner mellom individer (endringer i husholdsammensetning er i så måte utrolig vanskelig/umulig).

Her benyttes et annet begrep, ***ressurs***, som er en bit av minnet til simuleringen, for eksempel en person, en tellevariabel eller en trekkemetode. Det er helt avgjørende at ulike tråder ikke jobber *samtidig* på *samme* ressurs, for eksempel bruke samme tellevariabel når man teller opp antall personer med et gitt kjennetegn. Det finnes metoder i C# for å håndtere dette ('lock'), men disse har ofte en negativ effekt på kjøretidene (alt annet stoppes), og løser heller ikke det mer grunnleggende problemet at faktorenes orden faktisk betyr noe. Deler to tråder en ressurs samtidig uten lock, vil en av trådene bli ignorert, som vil lede til en (liten) regnefeil. I verste fall mister simuleringen kontroll over sitt eget minne, og hele simuleringen stopper opp. Det enkleste er derfor at hver tråd jobber utelukkende på sine egne ressurser. For eksempel at en tellevariabel representeres med en array over reg.status x kjønn x alder, som i etterkant summeres i en fastlagt rekkefølge.

Parallell prosessering reduserer samlet kjøretid i MOSART med en faktor på minst 6. Det alene taler for at metoden bør bli i modellen. Det er trolig et potensial for å øke denne faktoren radikalt. Det er 64 kjerner i serveren 'sl-mosart-01', som antyder at det skulle være mulig å redusere kjøretiden med en faktor på 50+ med den teknologien vi allerede har.

Trekkemetoder

MOSART trekker om en person opplever en begivenhet gitt ved sannsynligheten for at denne begivenheten skal skje, og en eller annen form for (pseudo)stokastikk. Metoden vi i hovedsak bruker er å summere sannsynligheter, og hver gang summen passerer et heltall lar vi begivenheten skje. Brukt med litt varsomhet gir dette en forventningsrett simulering, med betydelig variansreduksjon. Hovedalternativet er å tildele personen et tilfeldig tall med uniform fordeling (0,1), og hvis tallet er mindre enn (den individuelle) sannsynligheten, lar vi begivenheten skje (standard tilfeldig trekning).

Poenget her er at rekkefølgen av personer inn i trekkemetoden i begge måter vil påvirke hvem det er som opplever begivenheten, og en slik endring kan/vil spre seg gradvis utover og til slutt vil hele simulering ha tatt en annen vei.

[44] Bosatte, utvandret, død (i år), død i tidligere år. Siste gruppe (spøkelser) er der for å huske (indirekte) relasjoner mellom personer, for eksempel barnebarn eller søsken.

Det er derfor helt avgjørende at ulike tråder ikke deler trekkemetoder[45], og at alle personer kommer inn i sin trekkemetode i en rekkefølge som er upåvirket av parallell prosessering. Dette sikrer vi ved å definere trekkemetoder som vektorer, om nødvendig med alle disse argumentene:

pensjonssystem x reg.status x kjønn x ettårig alder

Tildeling av random-seed til de ulike trekkemetodene er spesielt sensitivt, men dette skjer helt i begynnelsen av simuleringen, uten parallell prosessering.

Reproduserbarhet

Det er ingen ekstraordinære individer eller hendelser i MOSART, så feil i anvendelsen av trekkemetodene, som nevnt over, vil bli stokastisk støy rundt en forventet bane. Fordi MOSART har variansreduserende metoder, stort 'utvalg' og beregning av pensjonsytelser på samme livsløp ('pension systems'), blir denne støyen nesten uten betydning for forståelsen av resultatene. Det er likevel en helt sentral egenskap at man retter opp i slike feil, for ellers blir det umulig å reprodusere en simulering:

- Det kan ikke tas ut nye tabeller på samme kjøring uten at det kommer inn stokastisk støy
- Den variansreduserende effekten av pension systems kan bli ødelagt
- Feil i anvendelsen av en pensjonsregel kan gjemme seg bak stokastisk støy
- Det er vanskelig å formidle stokastisk støy, og oppdragsgiver liker det gjerne ikke
- Og viktigst; tunge feil krever at man finner det *punktet* i simuleringen feilen oppstår, men når dette flytter rundt på seg, blir det helt umulig (og det er også dette som ofte gjør det så vanskelig å finne disse feilene)

Generelt om feilsøking

Det er noen standardmetoder for å sjekke for slike feil:

- Kjøre samme simulering flere ganger med identisk oppsett, inkludert random-seed.
- Kjøre en av disse simuleringene uten parallell prosessering (og resirkulering av minne).
- Kjøre en simulering med flere identiske pensjonssystemer.

Det neste trinnet er å prøve å identifisere i hvilken modul avviket første gang oppstår, og deretter gå etter denne. I den grad avviket oppstår i flere moduler i samme år, vil det typisk være den modulen som kommer først i rekkefølgen av kjennetegn som har problemet. Utskrift av modellpopulasjoner med kritiske kjennetegn kan hjelpe, og det ligger en struktur hvor man (på ad hoc basis) kan legge inn utskrifter før og etter en modul (noen ganger kan en feil på slutten av året, der tabellprogrammet og modellpopulasjonen normalt skrives ut, kunne ha sin rot i flere moduler). I andre tilfeller må man gå inn i modulen for å skrive ut informasjon fortløpende til 'sim.errors'. Merk at feil i trekkemetodene ofte dukker opp stabilt også på små utvalg, som gjør denne prosessen raskere.

Konkret om feilen i privat tjenestepensjon

Simuleringen av privat tjenestepensjon har hatt en feil i trekkemetoden siden starten, men har blitt liggende av en viktig grunn; i motsetning til andre moduler har privat tjenestepensjon (slik den er nå) ingen tilbakevirkning på simuleringen ellers. Det er derfor et mindre problem, som i tillegg kunne ha

[45] En trekkemetode er her et objekt som har sin egen sekvens av tilfeldige tall og/eller summevariabel for en hendelse og sum av sannsynligheter.

blitt håndtert ved å koble ut modulen helt. Dette er gjort i noen av de beregningene som er levert. Det var to feil i modulen:

Den første feilen var at historiske data ble bare beregnet for pensjonssystem én, og på ett punkt var det informasjon som ikke ble distribuert til underliggende pensjonssystemer. Dette var en ordentlig feil som påvirket forventningsverdiene.

Den andre feilen var at ved uttak av alderspensjon simuleres det om den private tjenestepensjonen skal være over et fast antall år (dvs. færrest mulig) eller være en livslang annuitet. Dette er en egenskap som også simuleres for *ikke-bosatte*, som er ganske sjeldent i MOSART. Det ble oversett at reg.status skulle inngått i trekkemetoden, og bosatte og ikke-bosatte kom derfor inn i en (genuint) tilfeldig rekkefølge i trekkemetoden, og dette påvirket stokastikken (men ikke forventningsverdien).

Merknad: Normalt defineres arrays til den størrelsen de trenger å ha, men ikke større. Det er flere grunner til det, på individnivå handler det om å redusere forbruket av internminnet, som en nødvendighet. På aggregert nivå (tellevariabler, trekkemetoder) handler det litt om estetikk, kjøretider, men også det å fange opp feil; en for vid definisjon av en dimensjon i arrayet gjør at simuleringen aksepterer en ulovlig verdi på forklaringsvariabelen. Men her gjør det at man overser en feil; en utelatt forklaringsvariabel (reg.status) i trekkemetoden blir ignorert.

Dårlig haiking

En effektiv måte for å legge inn nye ting i MOSART er å følge mønsteret fra en lignende egenskap som allerede er lagt inn, gjerne omtalt som ***haiking***. Når man kommer med en ny ytelse, er det veldig enkelt å putte denne inn i en eksisterende modul. Man slipper å etablere en hel struktur for å loope gjennom befolkningen, og noen ganger vil det være helt absurd å etablere en egen modul for sammenfallende ytelser (det er for eksempel en modul som håndterer alle skatter, ikke en modul for hver skattetype).

Andre ganger kan en modul bli komplisert, uoversiktlig og overbelastet av å håndtere flere ting på en gang, det vil si dårlig haiking. Beregningen av pensjon skjer på par-nivå (samordning av minsteytelsen i gammelt system), beregningen av skatt skjer på husholdsnivå. Disse to modulene har derfor et litt annet oppsett av hvordan befolkningen loopes. Det var likevel naturlig å putte beregningen (simuleringen) av privat tjenestepensjon inn i modulen for beregningen av pensjon, men legge hele regelverket til en egen modul, 'read_private_occupational_pension.cs'. Problemet er at det inngår elementer av stokastikk her, både på innskuddspremien og i utbetalingsplanen (fast antall år vs. livslang annuitet).

MOSART er bygd opp slik at 'read_*'-modulene håndterer alt som er trivielt og voluminøst, som innlesing og beregning av overgangssannsynligheter og regler for skatt og pensjon. 'sim_*'-modulene er forsøkt holdt kortfattet, slik at det skal være mulig å lese strukturen ut av kildekoden. Dette inkluderer å håndtere stokastikken og trekkemetodene. Her feilet det ved at trekkemetodene ble lagt inn i 'read_private_occupational_pension.cs'. Dermed forsvant muligheten for å ha oversikt.

Privat tjenestepensjon har nå fått sin egen 'sim_*'-modul, og en struktur som ligner på de andre modulene når det gjelder arbeidsdelingen mellom 'read_*' og 'sim_*'.

Vedlegg T: Filaksess

I MOSART er det helt sentralt å arbeide med filer, både ved innlesing av utgangspopulasjonen og utskrift av resultatene. For å forenkle arbeidet med lesing av filer tilbyr modellen en innkapsling av de ulike C#-kommandoene som benyttes for filaksess. Kommandoene er beskrevet i Vedlegg I, disse omfatter ulike nyttige og robuste metoder for lesning av filer. Et utvalg av de blir illustrert i dette vedlegget. Under feilsøking er det ofte ønskelig å skrive ut resultater av mellomregninger og liknende for nærmere inspeksjon og verifisering. Her tilbyr ikke MOSART ferdige metoder som for innlesning, slik at ordinære C#-metoder i stedet må benyttes.

Definere en ny fil i input.con

Som eksempel i dette vedlegget illustreres hvordan en fil som angir månedlig gjennomsnittsinntekt i ulike sysselsettingssektorer kan leses inn i MOSART. Filen er på forhånd lastet ned fra Statistikkbanken og lagt inn i katalogen 'input/v75/income' med navnet 'averageincome.csv'. I modellen defineres filene som inngår i utgangspopulasjonen i 'input.con'. Her er de gruppert etter tema. Siden dette er en fil som inneholder inntektstall defineres den nye filen i denne bolken:

```
* Income
*
housing                                 income/housing.bas
savings                                 income/savings.bas
*
time_series_wages                       income/time_series_wages.w2023
time_series_consumer_price_index        income/time_series_consumer_price_index.w2023
time_series_projected_real_wage_growth  income/time_series_projected_real_wage_growth.bas
time_series_projected_consumer_price_index income/time_series_projected_consumer_price_index.2pros
*
average_income                          income/averageincome.csv
```

Påfølgende referanser til filen kan nå enkelt gjøres ved å benytte kortnavnet i første kolonne.

Lesing av filer ved bruk av innebygde metoder

I MOSART leses filer som regel i egne metoder som heter 'read_*'. Derfor deklareres en metode 'read_average_earnings' i 'read_income.cs' sammen med de andre 'read_*'-metodene:

```
read_housing(ps);
read_saving(ps);
read_average_earnings(ps);
```

Her bør det angis at dette skal leses inn for alle pensjonssystemer ('ps'). Før det kan gjøres noen som helst operasjoner på en fil må den åpnes og rettigheter settes. Som det er beskrevet i Vedlegg I er det metoden 'open_multiple_file' som da bør benyttes.

I MOSART er det som regel slik at første kolonne i filen benyttes som identifikator. Slik er det også i 'averageincome.csv', og i eksemplet under er det gjennomsnittsinntekt for personer i «Local government» og «Central government» det skal leses inn tall for. Derfor benyttes metoden 'next_label' med tilhørende parametere.

Verdiene som skal leses inn er desimaltall og er på forhånd deklarert som variable av type double. Tallene leses inn fra filen med metoden 'getdouble' med kolonnenummeret (null-basert) som parameter.

For å redusere minnebruken er det svært viktig å lukke filer når de er ferdig lest eller skrevet slik at minnet de opptar blir frigjort. Til dette benyttes i dette tilfellet metoden 'close_open_file'.

```
double LocalEarnings;
double CentralEarnings;

public void read_average_earnings(int ps) {
   open_multiple_file(ps, "average_income");

   next_label("Local government", 3, enum_ignore_intermediary_lines.yes);
   LocalEarnings = getdouble(2);
   skip_rest_of_line();

   next_label("Central government", 3, enum_ignore_intermediary_lines.yes);
   CentralEarnings = getdouble(2);

   close_open_file();

   write_average_earnings();
}
```

Filen er nå lest inn i de deklarerte variablene 'LocalEarnings' og 'CentralEarnings'. Dette er et lite eksempel, og det er kanskje ikke hensiktsmessig å undersøke om tallene er korrekt lest inn, men for å illustrere hvordan variabler og annet kan skrives ut til en fil er det lagt inn et kall på metoden 'write_average_earnings' som tar seg av dette[46].

Skriving til filer

Det er selvsagt ikke nødvendig å definere egne metoder for å skrive ut noe, det kan like gjerne gjøres direkte fra det stedet dette ønskes rapportert. Her vises likevel alle kommandoene som kreves samlet i én skriv-metode. MOSART tilbyr ikke samme ferdig definert funksjonalitet som for innlesning, så her må C#-kommandoer benyttes direkte. Merk at dette eksemplet bruker de samme metodene for aksess av filer som i modellen ellers. Det er de samme objektene som .Net-biblioteket kunne tilby i 2006. Det har senere kommet til enklere og trolig mer effektiv funksjonalitet, men dette er ikke undersøkt nærmere.

Det må først opprettes to objekter fra klassene 'FileStream' og 'StreamWriter'. Det første av disse håndterer aksess til filen mens det andre tar seg av utskriften av denne. Begge disse objektene tilbyr et rikt sett av funksjonalitet som ikke illustreres i dette lille eksempelet. Siden et objekt som heter 'fs' allerede er i bruk i klassen 't_par'[47] må dette få et annet navn, her er 'fs2' brukt. Dette objektet er igjen parameter ved opprettelsen av det andre objektet, som er det som tar seg av selve utskriften.

[46] I akkurat dette eksemplet er det mye enklere å benytte MonoDevelop for å inspisere aktuell(e) verdi(er) av variable(r).
[47] MOSART benytter nesten alltid 'fs' som navn på objekter av klassen 'FileStream' og 'sw' for 'StreamWriter'.

For å unngå tap av data og korrumperte data må det passes på at strømmen av data som ligger klar til å bli skrevet ut tømmes før filen lukkes. Det gjøres ved kommandoen 'Flush' Og også her er det viktig å lukke objektene etter bruk.

```
public void write_average_earnings() {
        FileStream fs2 = new FileStream("EarningsDifference.txt", FileMode.Create);
        StreamWriter sw = new StreamWriter(fs2);

        sw.WriteLine("This is customized output from the MOSART model.\n");
        sw.Write("The average earnings in central government was ");
        sw.Write(Math.Round(CentralEarnings / LocalEarnings, 2));
        sw.WriteLine(" times higher than in local government.");

        sw.Flush();
        sw.Close();
        fs2.Close();
}
```

Denne framgangsmåten for å skrive til filer kan også benyttes ved lesning av filer, da benyttes klassen 'StreamWriter' ut med 'StreamReader' og metodene 'Write'/'WriteLine' med 'Read'/'ReadLine'.

Utskrift til sim.errors

Noen ganger ønsker man kanskje bare å skrive ut noen enkle mellomregninger et fåtall antall ganger og få det gjort så enkelt som mulig. Siden det alltid opprettes en fil 'sim.errors' kan det være hensiktsmessig å benytte denne ved å bruke metodekallet 'write_error'. Dette er en del av 'global'-objektet, slik at dette i så fall må være tilgjengelig. En ulempe med denne prosedyren er at én og samme fil fylles opp fortløpende, og det er kanskje ikke gunstig.

```
g.write_error("This is customized output from the MOSART model.\n");
g.write_error("The average earnings in central government was " +
        Convert.ToString(Math.Round(CentralEarnings / LocalEarnings, 2)) +
        " times higher than in local government.");
```