Lab 2 - Exercise

Antoine Godin (Kingston University)

Agent-based and stock-flow consistent modelling: theory and applications - Paris - July 17

Using pksfc

To show how to use the package PKSFC, we will see the various step used in the lecture to update the model PC. First we need to load the package

```
library(PKSFC)
```

Then, you need to download the two attached 'SIM.txt' and 'SIMEX.txt' file and save it in the folder of your choice. Make sure to set the working directory where you saved the downloaded file. In command line this looks like this but if you use Rstudio, you can use the graphical interface as well (Session>Set Working Directory>Choose Directory)

```
setwd("pathToYourDirectory")
```

Loading the model

The first thing to do is to load the model and check for completeness.

```
pc<-sfc.model("Models/PC.txt",modelName="Portfolio Choice Model")
pc<-sfc.check(pc,fill=FALSE)
```

We are now ready to simulate the model

```
datapc<-simulate(pc)
```

Expectations and random shocks

The first of experiment is meant to observe the buffer role of money, in case of random shocks applied to expected disposable income. To do these, we need to modify slightly model pc and change the equations determining consumption, demand for bonds and money, and the expectations on income and wealth. We will call the new model pcRand.

```
pcRand<-sfc.editEqus(pc,list(list(var="cons",eq="alpha1*yde+alpha2*v(-1)"),
      list(var="b_h",eq="ve*(lambda0 + lambda1*r - lambda2*(yde/ve))")))
pcRand<-sfc.addEqus(pcRand,list(
  list(var="yde",eq="yd(-1)*(1+rnorm(1,sd=0.1))",
         desc="Expected disposable income depending on random shocks"),
  list(var="h_d",eq="ve-b_h",desc="Money demand"),
  list(var="ve",eq="v(-1)+yde-cons",desc="Expected disposable income")))
pcRand<-sfc.editVar(pcRand,var="yd",init=86.48648)
pcRand<-sfc.check(pcRand)
```

You probably have noticed that the `yde` equation contains the R function `rnorm` which allows you to extract a random number from a normal distribution centered around 0, with standard variation of 0.1. The package indeed allows you to use any R function such as `min`, `max` or `rnorm`. However, because of the way the Gauss-Seidel algorithm works, we are facing a problem. Indeed, as the `rnorm` function extract a number in each iteration, this will mean that the algorithm cannot converge since the difference between each iteration

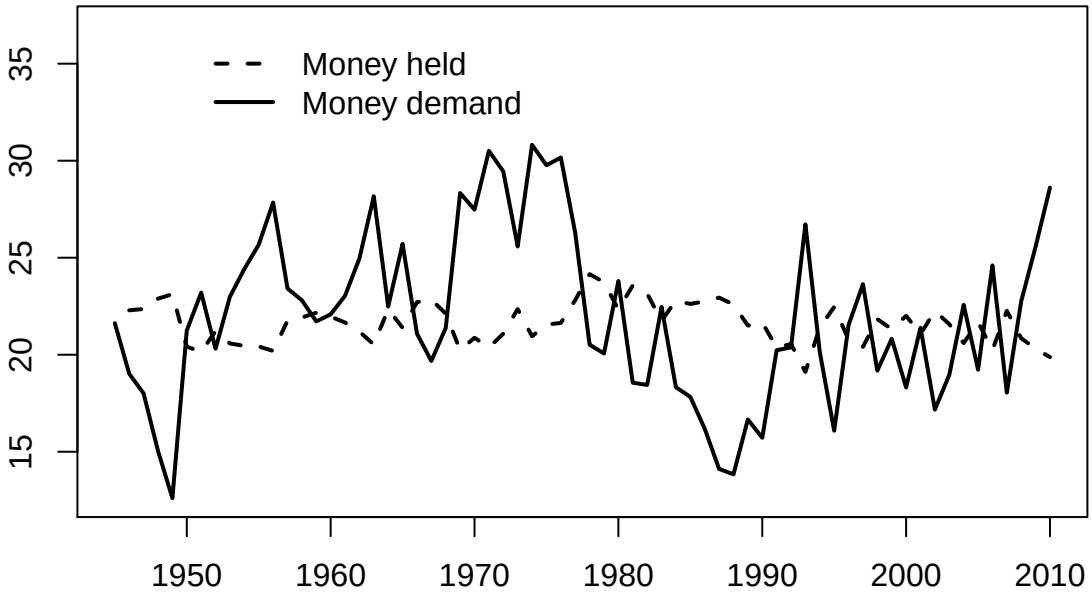
of the algorithm depends of the difference between each extraction. This is why we need to restrict the number of iteration of the Gauss-Seidel.

Let's now simulate the model.

```
datapcRand<-simulate(pcRand,maxIter=2)
```

This replicates figure 4.1, page 110

```
plot(pcRand$time,datapcRand$baseline[,"h_h"],type="l",xlab="",ylab="",lty=1,lwd=2,
     ylim=c(1*min(datapcRand$baseline[,c("h_h","h_d")],na.rm=T),
            1.2*max(datapcRand$baseline[,c("h_h","h_d")],na.rm=T)))
lines(pcRand$time,datapcRand$baseline[,"h_d"],lty=2,lwd=2)
legend(x=1950,y=1.2*max(datapcRand$baseline[,c("h_h","h_d")],na.rm=T),
       legend=c("Money held","Money demand"),lty=c(2,1),lwd=2,bty="n")
```



Endogenous propensities to consume

We change the propensities to consume to incorporate the fact that they might be impacted by interest rates and thus have

$$\alpha_1 = \alpha_1 0 - \iota \cdot r_{-1} \tag{4.32}$$

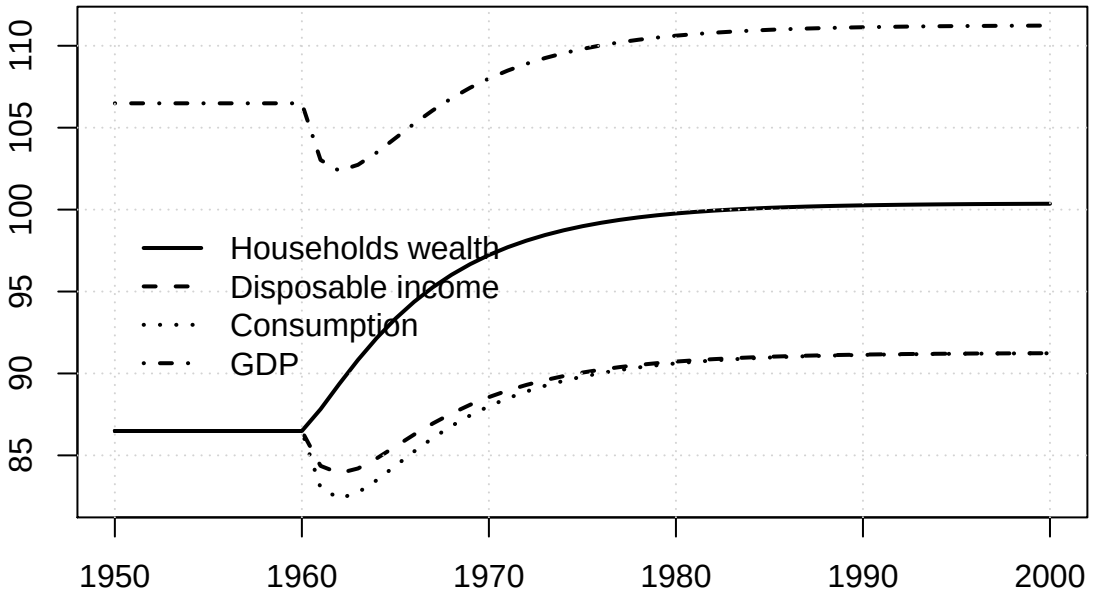
This piece of code shows how to implement this

```
pc_end<-sfc.editEqus(pc,list(
  list(var="cons",eq="alpha1*yde+alpha2*v(-1)"),
  list(var="b_h",eq="ve*(lambda0 + lambda1*r - lambda2*(yde/ve))")))
pc_end<-sfc.addEqus(pc_end,list(
  list(var="yde",eq="yd(-1)",desc="Expected disposable income"),
  list(var="h_d",eq="ve-b_h",desc="Money demand"),
  list(var="alpha1",eq="alpha10-iota*r(-1)"),
  list(var="ve",eq="v(-1)+yde-cons",desc="Expected disposable income")))
pc_end<-sfc.addVars(pc_end,list(
  list(var="alpha10",init="0.7",
       desc="Endoegnous propensity to consume - autonomous term"),
```

```
  list(var="iota",init="4",
         desc="Endoegnous propensity to consume - interest rate impact")))
pc_end<-sfc.editVar(pc_end,var="yd",init=86.48648)
pc_end<-sfc.check(pc_end)
datapc_end<-simulate(pc_end)
init = datapc_end$baseline[56,]
pc_end<-sfc.addScenario(model=pc_end,vars="r_bar",values=0.035,
                                                  inits=1960,ends=2010,starts=init)
datapc_end<-simulate(pc_end)
```

This replicates plot figure 4.9. p. 123

```
time2=c(1950:2000)
matplot(time2,
     datapc_end$scenario_1[as.character(time2),c("v","yd","cons","y")],
     type="l",xlab="",ylab="",lty=1:4,lwd=2,col=1)
grid()
legend(x=1950,y=100,legend=c("Households wealth","Disposable income",
            "Consumption","GDP"),lty=1:4,lwd=2,bty="n")
```



Adding Maastricht to model PC.

Your homework: I want you to add the golden rule to model PC by using various policies.

1. Assume a rational government: it knows the debt to GDP ratio in the steady state is given by some parameters it cannot control but also that the tax rate and thus select the tax rate that is optimal.
2. Assume a somewhat rational government which on top of using the optimal tax rule compensate the increase in taxes by an increase in government expenditure growing at 3% for 20 periods.
3. Assume a more rational government which does not use the optimal tax rule but instead gradually increases it such that the optimal rule is reached within 20 periods. The growth rate of 3% of government expenditure is maintained.

For all the different cases, plot the GDP, debt to GDP, Consumption, Disposable income and Deficit. Compare the results.