Setting up the environment

This tutorial is meant to reproduce the graphs and tables of chapter 3 of Godley and Lavoie (2007, G&L for now on) and at the same time discover some of the tools provided by the package PK-SFC.

Before doing any modelling, we need to load the package in the R environment.

```
library(PKSFC)
```

```
## Loading required package: expm
## Loading required package: Matrix
##
## Attaching package: 'expm'
##
## The following object is masked from 'package:Matrix':
##
##     expm
##
## Loading required package: igraph
```

The, 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 comand line this looks like this but if you use Rstudio, you can use the graphical interface as well (Session>Set Working Dirctory>Choose Directory)

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

Loading the model

The first thig to do is to load the model anc check for completeness.

```
simex<-sfc.model("SIMEX.txt",modelName="SIMplest model")
simex<-sfc.check(simex,fill=FALSE)
```

We are now ready to simulate the model

```
datasimex<-simulate(simex)
```

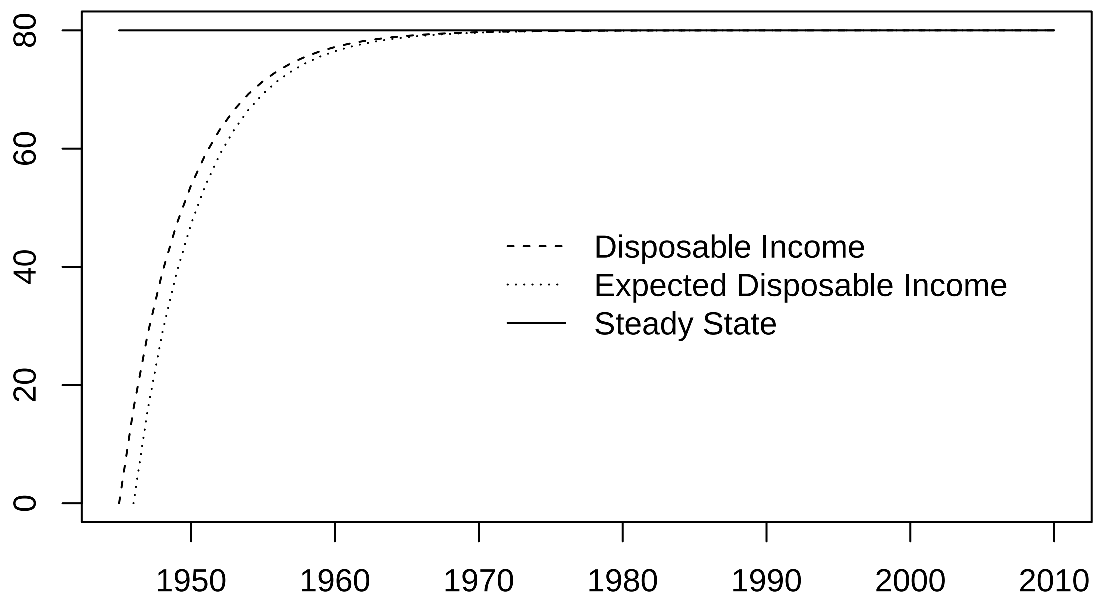
This replicates table 3.6 page 81

```
round(t(datasimex$baseline[c(1,2,3,66),c("G_d","Y","T_s","Yd","Yd_e","C_d","H_s","H_h")]),
      digits=1)
```

```
##      1945 1946 1947 2010
## G_d    20   20 20.0   20
## Y      NA   20 36.0  100
## T_s    NA    4  7.2   20
## Yd      0   16 28.8   80
## Yd_e   NA    0 16.0   80
## C_d    NA    0 16.0   80
## H_s     0   16 28.8   80
## H_h     0   16 28.8   80
```

This replicates figure 3.5 page 82

```r
plot(simex$time,datasimex$baseline[,"Yd"],type="l",xlab="",ylab="",lty=2)
lines(simex$time,datasimex$baseline[,"Yd_e"],lty=3)
lines(simex$time,vector(length=length(simex$time))+datasimex$baseline["2010","Yd"])
legend(x=1970,y=50,legend=c("Disposable Income","Expected Disposable Income","Steady State"),
       lty=c(2,3,1),bty="n")
```



Gauss-Seidel and solving a PK-SFC model with the package

If you have run the SIM model, you probably have noted the difference in time needed to obtain the results for model SIM and model SIMEX. We can confirm this impression by computing the simulation time. We can thus load the SIM model and then compare both timing.

```r
sim<-sfc.model("SIM.txt",modelName="sim")

#Simulation SIM
ptm <- proc.time()
data1<-simulate(sim)
paste("Elapsed time is ",proc.time()[3]-ptm[3],"seconds")
```

```
## [1] "Elapsed time is  7.358 seconds"
```

```r
#Simulation SIMEX
ptm <- proc.time()
dataex<-simulate(simex,tolValue = 1e-10)
paste("Elapsed time is ",proc.time()[3]-ptm[3],"seconds")
```

```
## [1] "Elapsed time is  0.109 seconds"
```

The reason behind these differences in time is based on the way of how the model are simulated. In a nutshell, the package solves a system of euquation for each period of the simulation. The solver used for the resolution of the system is based on the Gauss-Seidel (GS) algorithm.

The GS allows to solve a linear system of equation which can be represented by $Ax = b$, $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ via an iterative algorithm, where each iteration can be represented by $Lx^{k+1} = b - Ux^k$, $A = L + U$. The pseudo-code for the GS is the following:

```
Select initial values $x^0$
While $k < maxIter$ & $\delta < tolValue$
For each $i = 1, ..., n$:
```

$$x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right)$$

Compute δ:

$$\delta = \frac{x^{k+1} - x^k}{x^k}$$

where the convergence of the GS towards the solution of the system is being captured by the difference between δ and $tolValue$ and where $maxIter$ allows to limit the number of iterations performed in cases where the convergence is not observed (note that converegence is ensured for a linear system of equations).

The two parameters, i.e. the convergence parameter and the maximum number of iterations can be specified in the `simulate` function. This obviously impacts the time needed to simulate the model:

```r
#Simulation 1
ptm <- proc.time()
data1<-simulate(sim)
paste("Elapsed time is ",proc.time()[3]-ptm[3],"seconds")
```

```
## [1] "Elapsed time is  6.928 seconds"
```

```r
#Simulation 2
ptm <- proc.time()
data2<-simulate(sim,tolValue = 1e-3)
paste("Elapsed time is ",proc.time()[3]-ptm[3],"seconds")
```

```
## [1] "Elapsed time is  1.168 seconds"
```

```r
#Simulation 3
ptm <- proc.time()
data3<-simulate(sim, maxIter=10)
paste("Elapsed time is ",proc.time()[3]-ptm[3],"seconds")
```

```
## [1] "Elapsed time is  0.199999999999999 seconds"
```

In this package, the GS algorithm has been improved in order to account for the fact that you might have various blocks of independent systems of equations (where a system might actually be composed of only one equation). If this is the case, you reduce significantly the number of iterations needed for the Gauss-Seidel algorithm to converge. You can obtain the various blocks of equations that the package finds in the model by printing the summary of a model.

```r
summary(sim)
```

```
## Model: sim
## Equations:
## Block: 1
##  g_s = g_d #
## Block: 2
##  c_s = c_d # 1. EQUATIONS
##  t_s = t_d #
##  n_s = n_d #
##  t_d = theta*w*n_s #
##  c_d = alpha1*yd+alpha2*h_h_1 #
##  h_s = h_s_1+g_d-t_d #
##  y = c_s+g_s #
##  n_d = y/w #
##  dh_s = h_s-h_s_1 # VARIABLE USED FOR PLOTS
##  dh_h = h_h-h_h_1 #
##  yd = w*n_s-t_s #
##  h_h = h_h_1+yd-c_d #
## Initial Values:
##  c_s = NA # 1. EQUATIONS
##  g_s = NA #
##  t_s = NA #
##  n_s = NA #
##  yd = NA #
##  t_d = NA #
##  c_d = NA #
##  h_s = 0 #
##  h_h = 0 #
##  y = NA #
##  n_d = NA #
##  dh_s = NA # VARIABLE USED FOR PLOTS
##  dh_h = NA #
## Exogenous Values:
##  alpha1 = 0.6 # 2. PARAMETERS
##  alpha2 = 0.4 #
##  theta = 0.2 #
##  g_d = 20 # EXOGENOUS
##  w = 1 #
```

```
summary(simex)
```

```
## Model: SIMplest model
## Equations:
## Block: 1
##  G_s = G_d #
##  Yd_e = Yd_1 #
## Block: 2
##  C_d = alpha1*Yd_e+alpha2*H_h_1 #
## Block: 3
##  C_s = C_d # 1. EQUATIONS
## Block: 4
##  Y = C_s+G_s #
## Block: 5
##  N_d = Y/W #
## Block: 6
##  N_s = N_d #
## Block: 7
##  T_d = theta*W*N_s #
## Block: 8
##  T_s = T_d #
##  H_s = H_s_1+G_d-T_d #
## Block: 9
##  Yd = W*N_s-T_s #
## Block: 10
##  H_h = H_h_1+Yd-C_d #
## Initial Values:
##  C_s = NA # 1. EQUATIONS
##  G_s = NA #
##  T_s = NA #
##  N_s = NA #
##  Yd = 0 #
##  T_d = NA #
##  C_d = NA #
##  H_s = 0 #
##  H_h = 0 #
##  Y = NA #
##  Yd_e = NA #
##  N_d = NA #
## Exogenous Values:
##  alpha1 = 0.6 # 2. PARAMETERS
##  alpha2 = 0.4 #
##  theta = 0.2 #
##  G_d = 20 # EXOGENOUS
##  W = 1 #
```

You observe that by changing the consumption function, the simex model has a radical block structure. Indeed, the model is not a system of simultaneous equation anymore while the model sim is. This allows the package to reduce significantly the number of iterations in the GS algorithm. You can observe these number of iteration per block in the datastruture generated by the simulations under the colomn `iter block x` where `x` is the number of the block in the block structure observed when looking at the summary of the model. We can thus compare the number of iteratins needed to obtain convergence in the case of the SIM and SIMEX simulations

```r
#Observing the results of the three simulations
round(t(data1$baseline[c(1,2,20,40,66),c("iter block 1","iter block 2")]),digits=3)
```

```
##              1945 1946 1964 1984 2010
## iter block 1    0    2    1    1    1
## iter block 2    0  358  322  322  322
```

```r
round(t(dataex$baseline[c(1,2,20,40,66),
                        c("iter block 1","iter block 2","iter block 3",
                          "iter block 4","iter block 5","iter block 6")]),digits=3)
```

```
##              1945 1946 1964 1984 2010
## iter block 1    0    2    2    2    2
## iter block 2    0    2    2    2    2
## iter block 3    0    2    2    2    2
## iter block 4    0    2    2    2    2
## iter block 5    0    2    2    2    2
## iter block 6    0    2    2    2    2
```

Back to simulations: Expectations mistake

Section 3.7 of G&L analyses the impacts of a constant expectation on disposable income. In order to do that, we need to change the expectation equation in the model. This can be done with the `sfc.editEqu` function. We also need to set a value for the parameter

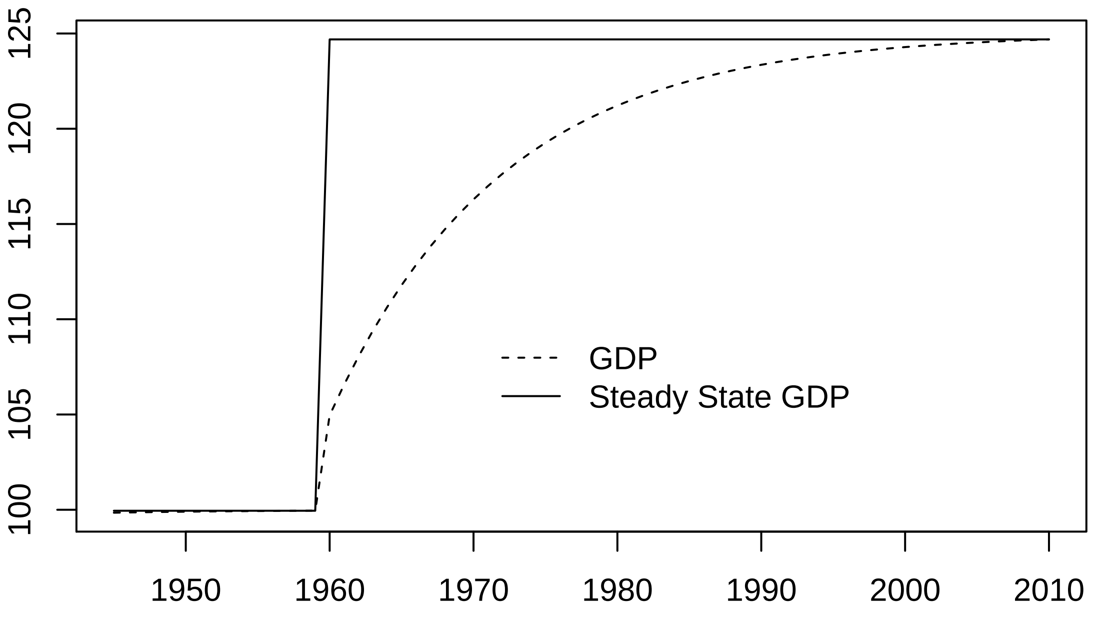
```
simex_b<-sfc.editEqu(simex,var="Yd_e",eq="Yd_fixed")
simex_b<-sfc.addVar(simex_b,var="Yd_fixed",init=80,desc="Constant expected disposable income")
simex_b<-sfc.check(simex_b,fill=F)
```

The model can be simulated and a scenario can be added as well, as done in Godley and Lavoie

```
datasimex_b<-simulate(simex_b)
init = datasimex_b$baseline[66,]
simex_b<-sfc.addScenario(simex_b,"G_d",25,1960,2010,init)
datasimex_b<-simulate(simex_b)
```

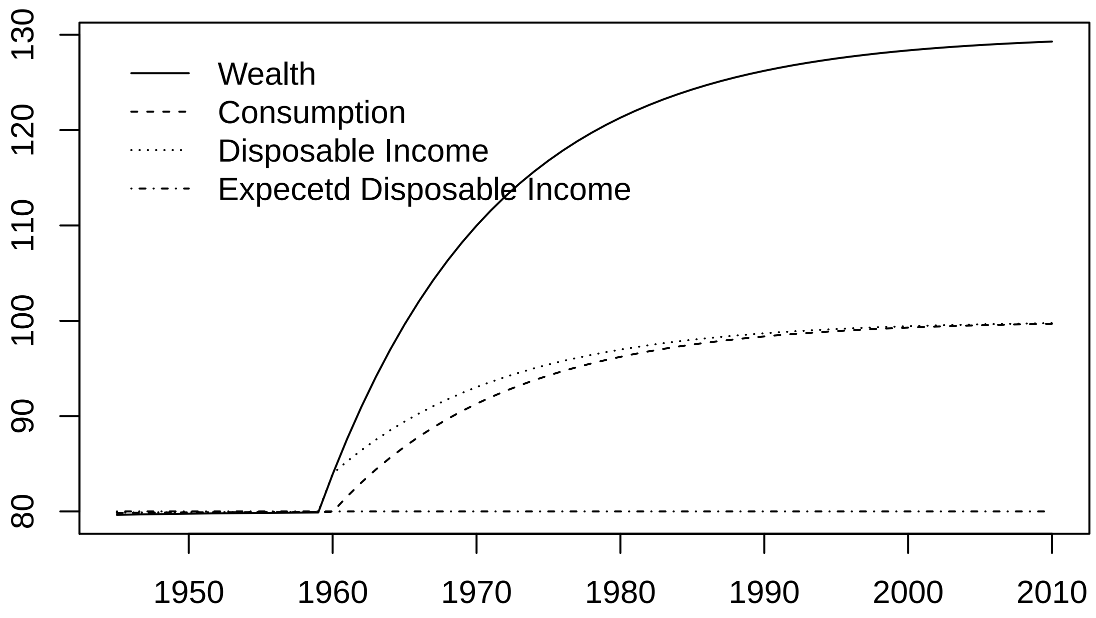
This replicates figure 3.6 page 84

```
plot(simex_b$time,datasimex_b$scenario_1[,"Y"],type="l",xlab="",ylab="",lty=2)
lines(simex_b$time,(simex_b$time>1959)*
        (datasimex_b$scenario_1["2010","Y"]-datasimex_b$scenario_1["1958","Y"])
      +datasimex_b$scenario_1["1958","Y"])
legend(x=1970,y=110,legend=c("GDP","Steady State GDP"),lty=c(2,1),bty="n")
```



This replicates figure 3.7 page 84

```r
plot(simex_b$time,datasimex_b$scenario_1[,"H_s"],type="l",xlab="",ylab="")
lines(simex_b$time,datasimex_b$scenario_1[,"C_d"],lty=2)
lines(simex_b$time,datasimex_b$scenario_1[,"Yd"],lty=3)
lines(simex_b$time,datasimex_b$scenario_1[,"Yd_e"],lty=4)
legend(x=1944,y=130,legend=c("Wealth","Consumption","Disposable Income",
                             "Expecetd Disposable Income"),lty=c(1,2,3,4),bty="n")
```


Wealth
Consumption
Disposable Income
Expecetd Disposable Income