CT561: Systems Modelling and Simulation

Week 7: Functions, Data Frames and deSolve

https://github.com/JimDuggan/CT561

Dr. Jim Duggan,
Information Technology,
School of Engineering & Informatics
NUI Galway
OÉ Gaillimh

Functions

- A function is a group of instructions that:
  - takes input,
  - uses the input to compute other value, and
  - returns a result (Matloff 2009).
- Users of R should adopt the habit of creating simple functions which will make their work more effective and also more trustworthy (Chambers 2008).
- Functions are declared:
  - using the **function** reserved word
  - are objects

Example

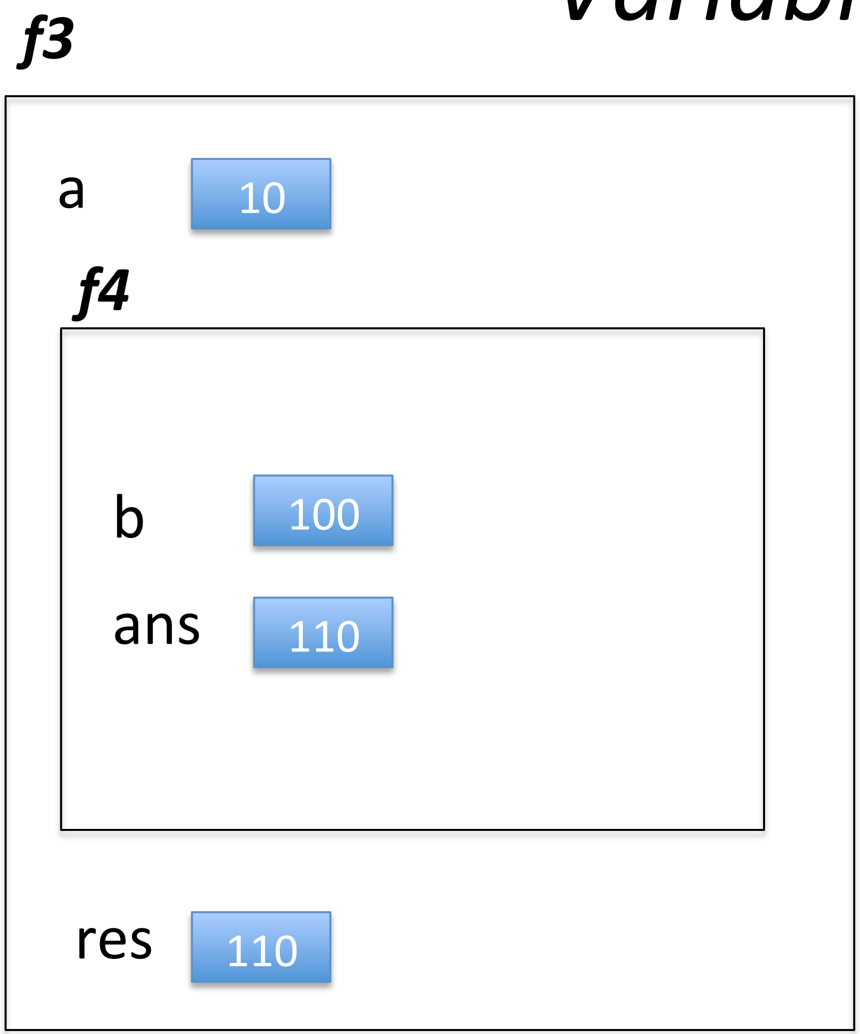
- Function declared as an object
- Takes arguments
- Local scope for function variables
- Generates result
- return() explicit
- Implicit – last expression evaluated.

```
convC2F<-function(celsius)
{
  fahr<-celsius*9/5 + 32.0
  return(fahr)
}
```

```
convC2F(celsius)
```

```
>
>
> x<-convC2F(100)
> x
[1] 212
```

Nested Functions: *Variable Scope*




```r
f3<-function(a){
  f4<-function(b){
    ans<-a+b
  }
  res<-f4(a^2)
}
```

```r
> f3(10)
> r<-f3(10)
> r
[1] 110
```

Function objects and arguments

- Functions are first-class objects, can be passed to other functions
- Arguments can be named directly in the call
- In that scenario, order of arguments not an issue

```
convert<-function(func, input)
{
   ans<-func(input)
   return (ans)
}
```

```
> convert(func=convC2F,input=100)
[1] 212
> convert(input=100,func=convC2F)
[1] 212
```

Challenge 7.1

- Write a function that takes in a vector and returns the number of odd numbers
- Write a function that takes in a vector and returns a vector of even numbers
- Write a function that returns the unique values in a vector

`duplicated()` determines which elements of a vector or data frame are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements (rows) are duplicates.

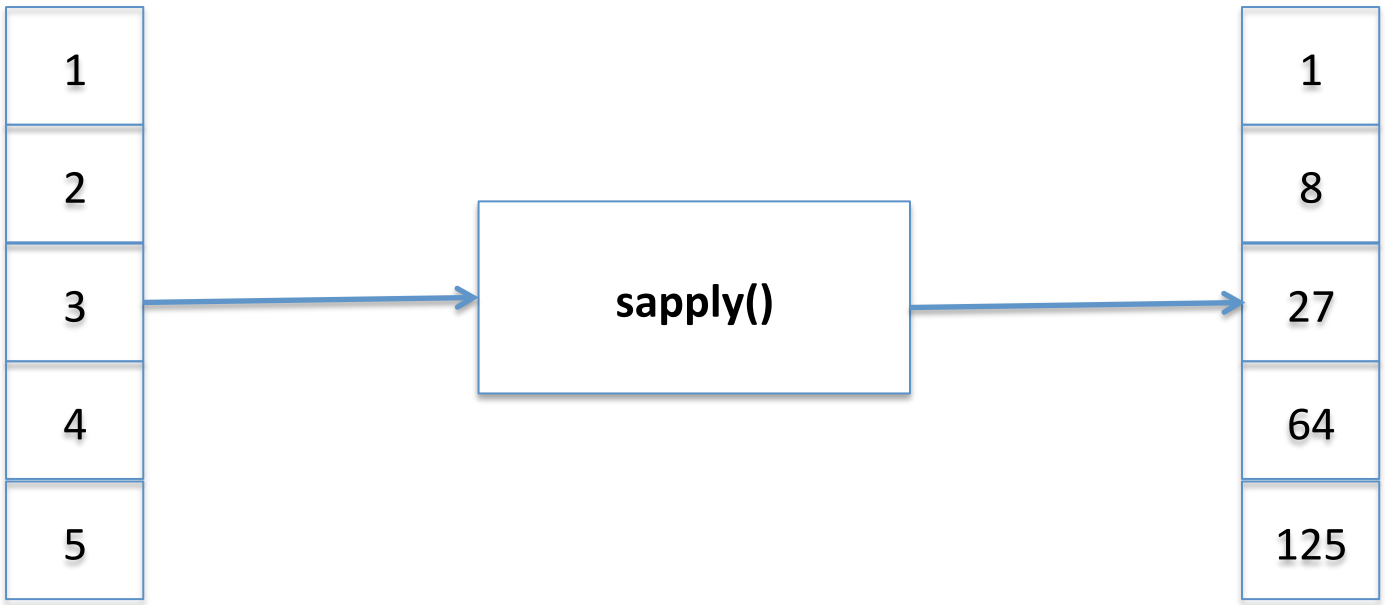
```
> duplicated(c(1,2,3,4,5,6,1))
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

sapply()

- Another use of user-defined functions in R is as parameters to the *apply* family of functions, which are one of the most famous and used features of R (Matloff 2009).
- The general form of the **sapply(x,f,fargs)** function is as follows:
  – **x** is the target vector or list
  – **f** is the function to be called and applied to each element
  – **fargs** are the optional set of arguments that can be applied to the function **f**.

Sample Problem

- To get the cube of numbers in a vector



Method

- Embed function in call

```
> sapply(v1,function(x){x^3})
[1]     1    64   729  4096 15625
>
```

```
> v1
[1]  1  4  9 16 25
> sapply(v1,function(x,y){x^3+y},10)
[1]    11    74   739  4106 15635
>
```

Challenge 7.2

- Use sapply() for the following task
  - transform a vector according to the following function
    - $f(x) = 3x^2+5x+10$

Data Frames

- A data frame has a two-dimensional rows and columns structure.
- Can be viewed as a set of vectors, organised into a column format

```
> ids<-c("1234567","1234568")
> fNames<-c("Jane","Matt")
> sNames<-c("Smith","Johnson")
> ages<-c(21,25)
> ids
[1] "1234567" "1234568"
> fNames
[1] "Jane" "Matt"
> sNames
[1] "Smith"   "Johnson"
> ages
[1] 21 25
```

Creating a Data Frame

```
s<-data.frame(ID=ids,FirstName=fNames,Surname=sNames,
              Age=ages,stringsAsFactors=FALSE)
```

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> str(s)
'data.frame':	2 obs. of  4 variables:
 $ ID       : chr  "1234567" "1234568"
 $ FirstName: chr  "Jane" "Matt"
 $ Surname  : chr  "Smith" "Johnson"
 $ Age      : num  21 25
```

Accessing Row Data

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> s[1,]
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
> s[1:2,]
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> s[-1,]
       ID FirstName Surname Age
2 1234568      Matt Johnson  25
```

Accessing Column Data

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> s[,"Age"]
[1] 21 25
> s[,1]
[1] "1234567" "1234568"
> s[,2]
[1] "Jane" "Matt"
> s[,3]
[1] "Smith"   "Johnson"
> s[,4]
[1] 21 25
```

Accessing Column Data (using tags)

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> s$ID
[1] "1234567" "1234568"
> s$FirstName
[1] "Jane" "Matt"
> s$Age[1:2]
[1] 21 25
> max(s$Age)
[1] 25
```

subset() function

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> sub<-subset(s,s$Age>21)
> sub
       ID FirstName Surname Age
2 1234568      Matt Johnson  25
```

Adding extra information to a data frame - Vectorized

```
> s
       ID FirstName Surname Age
1 1234567      Jane   Smith  21
2 1234568      Matt Johnson  25
> s$Discount<-ifelse(s$Age<=21,0.25,0.10)
> s
       ID FirstName Surname Age Discount
1 1234567      Jane   Smith  21     0.25
2 1234568      Matt Johnson  25     0.10
```

Merging Data Frames

- For data analytics, opportunities often arise by merging different data sets into a single data frame, and the **merge()** function facilitates this
- In our student example, we could have a second data frame that stores examination results.

```
> ids<-c("1234567","1234568")
> subjects<-c("CT111","CT111")
> grade<-c(80,80)
> res<-data.frame(ID=ids,Subject=subjects,Grade=grade,stringsAsFactors=FALSE)
> res
       ID Subject Grade
1 1234567   CT111    80
2 1234568   CT111    80
```

Merge Function

```
> s
       ID FirstName Surname Age Discount
1 1234567      Jane   Smith  21     0.25
2 1234568      Matt Johnson  25     0.10
> res
       ID Subject Grade
1 1234567   CT111    80
2 1234568   CT111    80
> new<-merge(s,res,by="ID")
> new
       ID FirstName Surname Age Discount Subject Grade
1 1234567      Jane   Smith  21     0.25   CT111    80
2 1234568      Matt Johnson  25     0.10   CT111    80
```

deSolve package

Journal of Statistical Software

February 2010, Volume 33, Issue 9. *http://www.jstatsoft.org/*

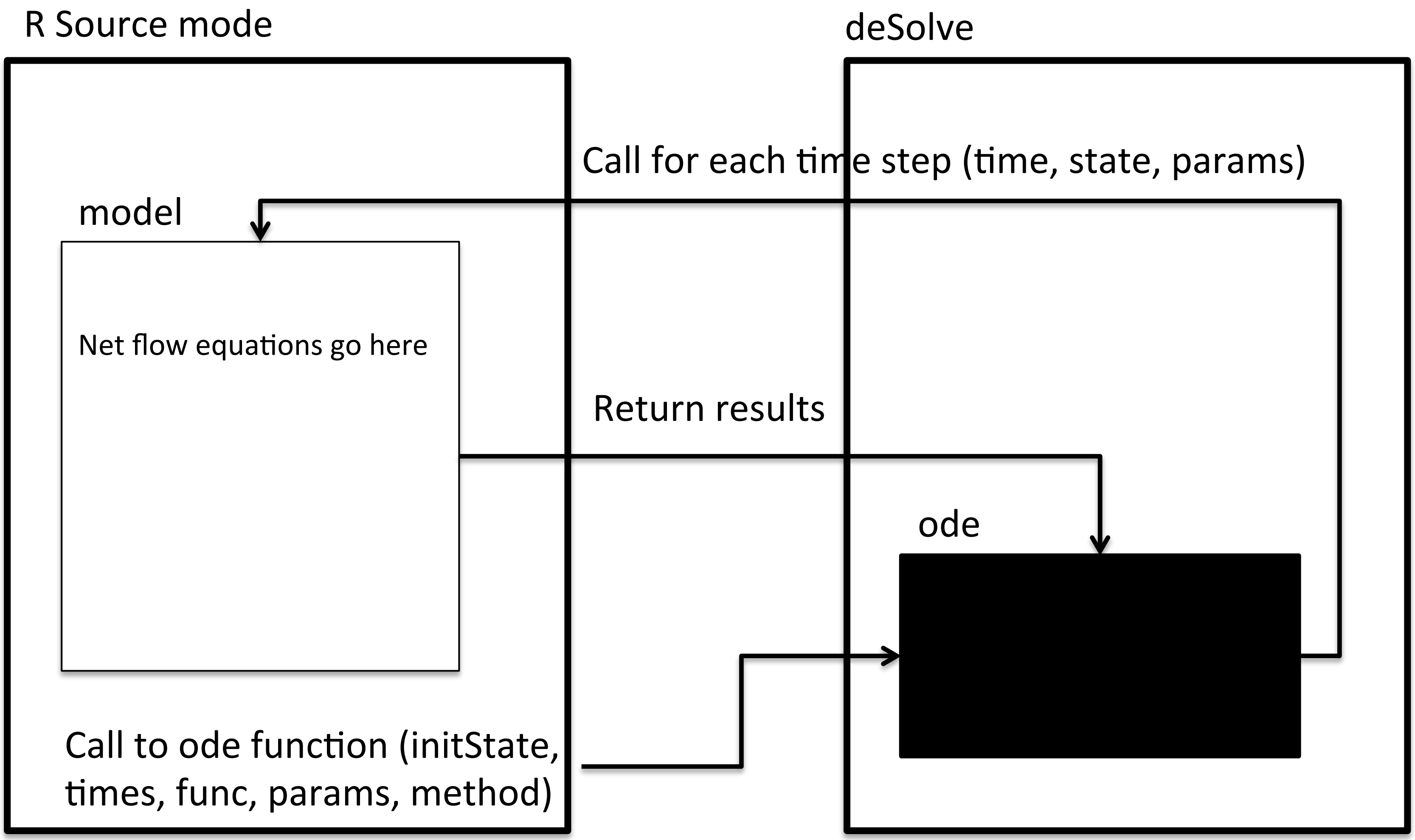
Solving Differential Equations in R: Package deSolve

Karline Soetaert
Netherlands Institute of Ecology

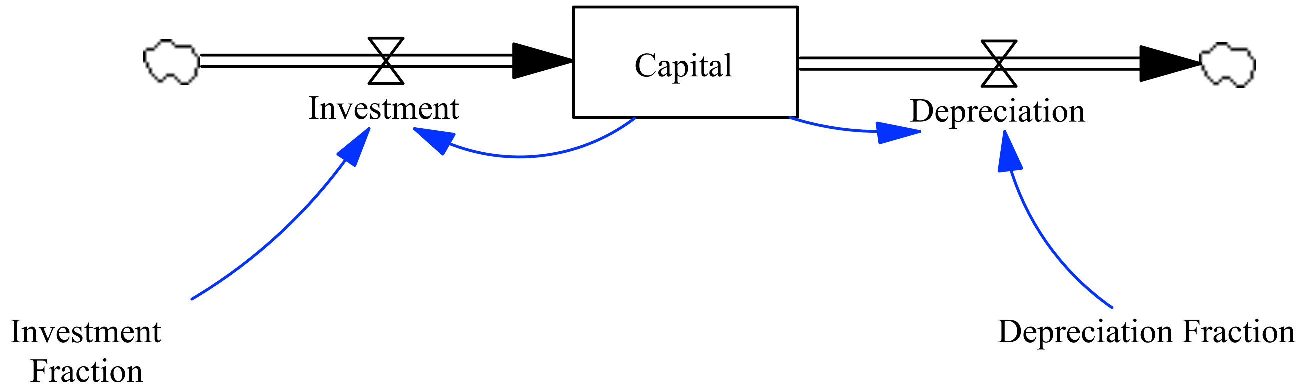
Thomas Petzoldt
Technische Universität Dresden

R. Woodrow Setzer
US Environmental Protection Agency

Overall Idea



Capital Model



Capital= INTEG (Investment-Depreciation,10000)

Investment Fraction = 0.08

Investment=Capital*Investment Fraction

Depreciation Fraction=0.05

Depreciation=Capital*Depreciation Fraction

R Implementation

```r
library(deSolve)  # include the deSolve library for odes
library(ggplot2)  # include ggplot for displaying graphs

# setup simulation start and finish times, and time step
START<-2015; FINISH<-2035; STEP<-0.25;

# create a simulation time vector, needed by deSolve
simtime <- seq(START, FINISH, by=STEP)
```

```
> head(simtime)
[1] 2015.00 2015.25 2015.50 2015.75 2016.00 2016.25
> tail(simtime)
[1] 2033.75 2034.00 2034.25 2034.50 2034.75 2035.00
```

The model function (1/2)

```
# create model function that contains all the equations
# this function receives
#   (1) the current time (time)
#   (2) a vectors of stocks (stocks)
#   (3) the auxiliary parameters (auxs)

model <- function(time, stocks, auxs)
{
  # with(as.list) makes it easy to reference variable
  with(as.list(c(stocks, auxs)),{
    # Calculate the flows
    fInvestment <- sCapital * aInvestmentFraction
    fDepreciation <- sCapital * aDepreciationFraction

```

The model function (2/2)

```
    # Calculate the net flow
    dC_dt  <- fInvestment - fDepreciation

    # return calculations as a list,
    # first value is a vector of net flows
    return (list(c(dC_dt),
                 Investment=fInvestment,
                 Depreciation=fDepreciation,
                 InvFraction=aInvestmentFraction,
                 DepFraction=aDepreciationFraction))
  })
}
```

Create stocks and auxiliaries.
Call solver, wrap in data frame

```
# create the vector of stocks and their initial values
stocks  <- c(sCapital=10000)

# create the vector of auxiliaries
auxs    <- c(aInvestmentFraction=0.08,
             aDepreciationFraction=0.05)

# call the function ode, results returned as a data frame
o<-data.frame(ode(y=stocks, simtime, func = model,
                  parms=auxs, method="euler"))
```

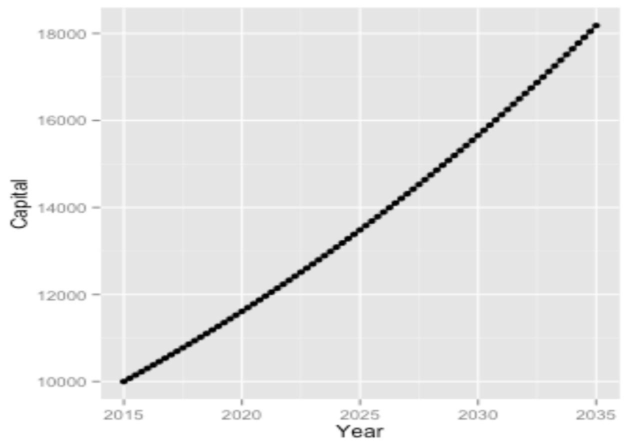
Simulation Results (in data frame)

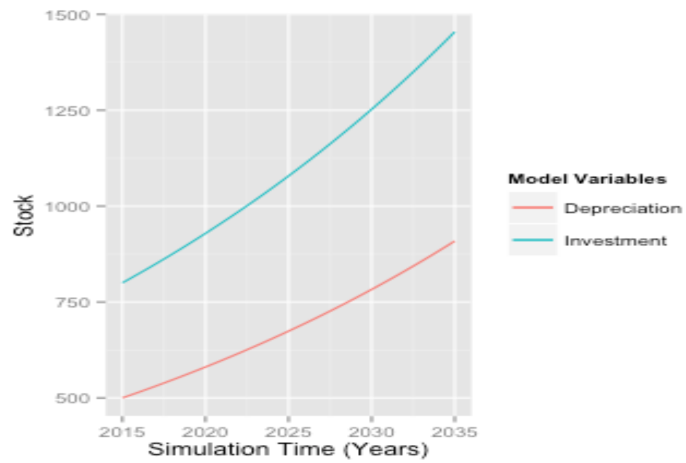
```
> str(o)
'data.frame':   81 obs. of  6 variables:
 $ time         : num  2015 2015 2016 2016 2016 ...
 $ sCapital     : num  10000 10075 10151 10227 10303 ...
 $ Investment   : num  800 806 812 818 824 ...
 $ Depreciation: num  500 504 508 511 515 ...
 $ InvFraction : num  0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.08 ..
 $ DepFraction : num  0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 ..
> head(o)
     time sCapital Investment Depreciation InvFraction DepFraction
1 2015.00 10000.00   800.0000     500.0000        0.08        0.05
2 2015.25 10075.00   806.0000     503.7500        0.08        0.05
3 2015.50 10150.56   812.0450     507.5281        0.08        0.05
4 2015.75 10226.69   818.1353     511.3346        0.08        0.05
5 2016.00 10303.39   824.2714     515.1696        0.08        0.05
6 2016.25 10380.67   830.4534     519.0334        0.08        0.05
```

Plot results

```
# plot the results
qplot(data=o,x=o$time,y=o$sCapital,
      geom=c("line","point"),xlab="Year",ylab="Capital")

ggplot() +
  geom_line(data = o, aes(x = time, y = Investment, color = "Investment")) +
  geom_line(data = o, aes(x = time, y = Depreciation, color = "Depreciation")) +
  xlab('Simulation Time (Years)') +
  labs(color="Model Variables")+
  ylab('Flows')
```





Challenge 7.3

- Build an R model for the stock adjustment structure
- Assume S_{INIT}=0
- Assume S^* starts at 100 and increases to 200 after 10 time units
- Let AT=1
- Run the simulation for 20 time units with DT=0.125