

Computer class

Econometrics of Financial Asset Pricing Models

Florian Ielpo

March 21, 2010

The purpose of this document is to introduce the very basic elements you need to complete your project. These elements are: programming a function in R and optimizing this function with respect to one or several arguments. The applications for this class are: computing implied volatility from option prices, sampling a GARCH process and estimating a GARCH process's parameters.

1 Basics of R

1.1 Syntax for functions

```
f<-function(input){  
  ...  
  commands  
  ...  
  return(output)  
}
```

Example: BS option pricer

```
bs<-function(sigma,T,K,S,r){  
  d1=log(S/(K*exp(-r*T)))/(sigma*sqrt(T))+1/2*sigma*sqrt(T)  
  d2=d1-sigma*sqrt(T)  
  P=S*pnorm(d1)-K*exp(-r*T)*pnorm(d2)  
  return(P)  
}
```

Basically, functions are to be jointly used with a **script**. For the BS example:

```
# Initial settings  
rm(list=ls())  
options(warn=-1)  
try(dev.off())
```

```

# Initial parameter settings
mu=0
r=0.05
sigma=.2
para=c(mu,sigma)
dt=1/250
T=5
n=100
S=100
K=30

# "Sourcing" of the function
source("bs.R")

# Display of the result
print(bs(sigma,T,K,S,r))

```

1.2 How to request help for an existing R function

- Get the arguments used as inputs

```
args(...)
```

- Get the html help for the function "rnorm()"

```
?rnorm
```

- More details in the help task bar.

1.3 Using the R optimizer

Know-hows around the R optimizer are essential for numerical finance. An easy way to step in: `optim` for several variable optimization and `optimize` for single variable optimization.

Example: inverting the BS formula numerically (**building on the previous script and functions**)

```

invert_bs<-function(sigma,P,T,K,S,r){
# require bs.R
spread=P-bs(sigma,T,K,S,r)
return(spread^2)
}

```

```

P=bs(sigma,T,K,S,r)
optimize(f = invert_bs, interval =c(0,1) , lower = 0,
        upper = 1, maximum = FALSE,
        tol = .Machine$double.eps^0.25, P,T,K,S,r)

```

1.4 Sampling a GARCH process

Important point: think of the recursive way to build the process. The model:

$$r_t = \log \frac{S_t}{S_{t-1}} = \mu + \sqrt{h_t} \epsilon_t \quad (1)$$

$$h_t = \omega_0 + \alpha(r_t - \mu)^2 + \beta h_{t-1} \quad (2)$$

Example from the class:

```

w0=10e-5
alpha=.05
beta=.8

n=1000
h=w0/(1-alpha-beta)
eps=rnorm(n,0,1)
x=eps[1]*sqrt(h)
for (i in 2:n){
x=rbind(x,eps[i]*sqrt(w0+alpha*x[i-1]^2+beta*h[i-1]))
h=rbind(h,w0+alpha*x[i-1]^2+beta*h[i-1])
}

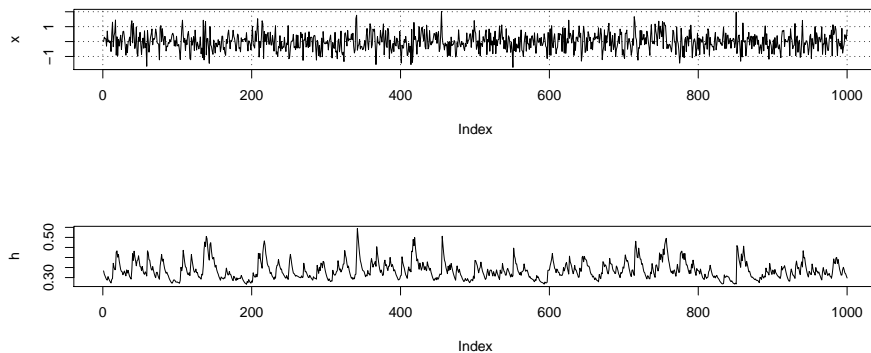
layout(matrix(1:2,2,1))
plot(x,type="l")
grid(col=1)
plot(h,type="l")

```

1.5 Estimating via ML a GARCH process

ML estimates of GARCH models are straightforward to obtain. Log likelihood of the model is obtained by computing

$$L(\theta|\underline{r}_t) = f(r_1) \sum_{t=2}^n f^\theta(r_t|r_{t-1}) \quad (3)$$



ML estimates are obtained by maximizing the latter expression with respect to θ the vector of the parameters. In the GARCH case, use copy and paste the following code:

```
garch_loglik<-function(para,x,mu){
omega0=para[1]
alpha=para[2]
beta=para[3]
loglik=0
h=w0/(1-alpha-beta)
  for (i in 2:length(x)){
    h=omega0+alpha*(x[i-1]-mu)^2+beta*h
    loglik=loglik+dnorm(x[i],mu,sqrt(h),log=TRUE)
  }
  return(-loglik)
}
```

```
mu=0
w0=10e-5
alpha=.1
beta=.5
```

Sampling of a GARCH process stored into x: to be done by the students

```
para=c(w0,alpha,beta)
optim(para, garch_loglik, gr = NULL,
      method = c("BFGS"),
      lower = -Inf, upper = Inf,
      control = list(trace=1), hessian = FALSE, x,mu)
```

1.6 Computing the price of an option through Monte Carlo method

An interesting way to gain insights from a stochastic volatility model is to compute implied volatilities obtained from its dynamics. The following function is a rough vanilla option pricing model based on a GARCH dynamics of volatility. Use it to compute option prices matching the following specifications:

- $S=100$
- strikes ranging from 80 to 120
- Maturities from 10 days to 200 days
- a risk free rate equal to 5%
- and using the parameters used for the previous GARCH model

Use the following function jointly with the implied volatility function obtained previously to compute the implied volatility surface and plot it using the function `persp`.

```
pricer<-function(para,n,strike,maturity,S,r){  
  
  # Parameters settings  
  w0=para[1]  
  alpha=para[2]  
  beta=para[3]  
  mu=para[4]  
  mu=mu-1/2*w0/(1-alpha-beta)  
  
  # Sampling of prices trajectories  
  prices=numeric(n)  
  
  for (nsim in 1:n){  
  
    h=w0/(1-alpha-beta)  
    x=numeric(maturity)  
    x[1]=rnorm(1)*sqrt(h)+mu  
  
    for (i in 2:maturity){  
      h=w0+alpha*(x[i-1]-mu)^2+beta*h  
      returns=rnorm(1)*sqrt(h)+mu  
      x[i]=returns  
    }  
  }  
}
```

```
temp=S*exp(sum(x))
prices[nsim]=temp
}
```

```
prices=pmax(prices-strike,0)
option_price=mean(prices)*exp(-r/365*maturity)
return(option_price)
}
```

1.7 How to load datasets from .csv files

To load a .csv file, provided that you set the right directory in R, you simply need to run:

```
X=read.csv("cac.csv")
```

To handle the dates, just use as.Dates:

```
dates=X[,1]
dates=as.Dates(dates,format="%d/%m/%Y")
```

An example with cac.csv that handles multiple windows graphics and the computation of returns:

```
layout(matrix(1:2,1,2))
```

```
X=read.csv("cac.csv",sep=";")
dates=X[,1]
dates=as.Date(dates,format="%m/%d/%Y")
par(bty="n")
plot(dates,X[,2],type="l",main="CAC",xlab="Time",ylab="Price",col="blue")
grid(nx=50)
```

```
rend=log(X[,2])
rend=diff(rend,1)
```

```
plot(dates[2:length(dates)],rend,type="l",main="CAC",
xlab="Time",ylab="Returns",col="red")
grid(nx=50)
```

