Estimating parameters for Garch(1,1) using the Optimization package

Reference: Haerdle et al, http://www.quantlet.com/mdstat/scripts/sfm/html/sfmframe118.html (in German)
AVt, Sep 2004

```
> restart; kernelopts(version); Digits:=14;
```

Maple 9.52, IBM INTEL NT, Dec 17 2004 Build ID 175529

Digits := 14

Given a time series x one wants to model its conditional variance according
to the following model which is Garch(1,1):

```
> #assume(0 <= a0): assume(0 <= a1):assume(0 <= b1): additionally(a1+b1 < 1);
  h(n+1) = a0 + a1*x(n)^2 + b1*h(n);
  h(1) = a0/(1-a1-b1); # =h1
  ``;
  epsilon(n) = x(n)/sqrt(h(n));
  epsilon = N(0,1), iid;
```

$$h(n+1) = a0 + a1\, x(n)^2 + b1\, h(n)$$

$$h(1) = \frac{a0}{1 - a1 - b1}$$

$$\varepsilon(n) = \frac{x(n)}{\sqrt{h(n)}}$$

$$\varepsilon = N(0, 1), \text{ iid}$$

To estimate the parameters a0, a1, b1 one has to maximize the maximum Likelihood LME

```
> LME = - 1/2*(N*ln(2*Pi) + Sum(ln(h(i)),i=1..N) + Sum(x(i)^2/h(i),i=1..N) );
```

$$LME = -\frac{1}{2} N \ln(2\,\pi) - \frac{1}{2}\left(\sum_{i=1}^{N} \ln(h(i))\right) - \frac{1}{2}\left(\sum_{i=1}^{N} \frac{x(i)^2}{h(i)}\right)$$

Then $\sqrt{h(n)}$ is the conditional variance.

A typical application is to estimate a (varying) variance (or better: volatility) for
stock prices.

Here the data series x are the *logarithmic* returns (which are almost normal
distributed, but not really) and one wants to have the annualized volatility as
$\text{volatility} = \sqrt{\text{variance}}\,\sqrt{252}$ (the usual year has about 252 trading days)

some Data: DAX closings

One does not work with prices, but with (logarithmic) returns (thus starting the day after)

```
> Returns:=[seq(evalf(ln(Closings[i]/Closings[i-1])),i=2..nops(Closings))]:
  i:='i':
  N:=nops(Returns);
```

N := 1197

statistics for the data series: for Garch(1,1) certain relations for the moments should be fullfilled
(Ref: Kierkegaard? Haerdle?):

```
> TS:=Returns:
  Mean:= stats[describe, mean](TS);          # should be 0
  Var := stats[describe, variance](TS);      # a0/(1-a1-b1)
  Skew:= stats[describe, skewness](TS);      # should be 0
  Kurt:= stats[describe, kurtosis](TS);      # should be
  6*a1^2/(1-b1^2-2*a1*b1-3*a1^2)
  TS:='TS':
```

$$\text{Mean} := -0.00046499982677338$$

$$\text{Var} := 0.00035376720909099$$

$$\text{Skew} := -0.00073269328851270$$

$$\text{Kurt} := 4.6161870555263$$

So at least correct it to have mean=0 (which is ok, as variance should be estimated) and work with that new time series (using hardware floats):

```
> Timeseries:= hfarray(1..N);
  for i from 1 to N do
    Timeseries[i]:=Returns[i]-Mean:
  end do: i:='i':
```

$$\text{Timeseries} := \begin{bmatrix} \text{1..1197 1-D Array} \\ \text{Data Type: float[8]} \\ \text{Storage: rectangular} \\ \text{Order: C\_order} \end{bmatrix}$$

Now take any more or less reasonable (historical) guess for the parameters

```
> A0 := 0.000;
  A1 := 0.1;
  B1 := 0.7;
```

$$A0 := 0.$$

$$A1 := 0.1$$

$$B1 := 0.7$$

and correct them according to the mentioned relation - where B1 is kept fix:

First change A1 to give the statistical kurtosis:

```
> theA1:='theA1':
  'Kurt=6*theA1^2/(1-B1^2-2*theA1*B1-3*theA1^2)';
  theA1:=fsolve(%,theA1);
```

$$\text{Kurt} = \frac{6\ \text{theA1}^2}{1 - \text{B1}^2 - 2\ \text{theA1}\ \text{B1} - 3\ \text{theA1}^2}$$

$$\text{theA1} := 0.21813969384208$$

Then change A0 to give the statistical variance:

```
> theA0:='theA0':
  'Var=theA0/(1-theA1-B1)'; theA0:=fsolve(%,theA0);
```

$$\text{Var} = \frac{\text{theA0}}{1 - \text{theA1} - \text{B1}}$$

$$\text{theA0} := 0.000028959492044822$$

We do not want to have this values as neccessary results (as we do not know whether the data actually are Garch(1,1)) and ignore skewness

```
> '[theA0, theA1, B1]': '%'=%;
  ``;
  'theA0/(1-theA1-B1)': '%'=%;
  '6*theA1^2/(1-B1^2-2*theA1*B1-3*theA1^2)': '%'=%;
```

$$[\text{theA0}, \text{theA1}, \text{B1}] = [0.000028959492044822, 0.21813969384208, 0.7]$$

$$\frac{\text{theA0}}{1 - \text{theA1} - \text{B1}} = 0.00035376720909100$$

$$\frac{6\ \text{theA1}^2}{1 - \text{B1}^2 - 2\ \text{theA1}\ \text{B1} - 3\ \text{theA1}^2} = 4.6161870555264$$

Hence for this selection we start with the 'right' variance and kurtosis for that N data.

We need to compute the maximum Likelihood:

```
> # maximium likelihood

  ML:=proc(dataseries, n, a0, a1, b1)
  local loc_a0, loc_a1, loc_b1, i,
  h0,h1,logML;

  loc_a0:=evalhf(a0);
  loc_a1:=evalhf(a1);
  loc_b1:=evalhf(b1);

  #calculate log Likelihood

  #h0 := loc_a0 + loc_a1 * 0 + loc_b1 * var;
  h0 := evalf( loc_a0 / (1 - loc_a1 - loc_b1) );

  logML:=0;
  for i from 2 to n do
    h1 :=    evalhf(loc_a0 + loc_a1 * dataseries[i - 1]^2 + loc_b1 * h0):
    logML := evalhf(logML - ln(h1) - dataseries[i] ^ 2 / h1):
    h0:= evalhf(h1);
  end do: i:='i':
  logML := evalhf(logML / 2 - ln(2 * Pi) * n / 2):
  return logML;
  end proc:
```

Check what happens for the new parameters:

```
> ML(Timeseries, N, A0, A1,B1);       # first guess
  ML(Timeseries, N, theA0, theA1,B1); # improvement
```

2187.11274345863376

3193.13570898078934

so i want to start working with that new guess.

```
> A0:=theA0; A1:=theA1; B1:=B1;
```

A0 := 0.000028959492044822

A1 := 0.21813969384208

B1 := 0.7

For fitting lots of evaluations are need and ML consumes a lot of time in Maple:

```
> st:=time():
  for i from 1 to 100 do
    ML(Timeseries, N, A0,A1,B1+0.01/i);
  end do:
  `seconds needed`=time()-st; i:='i':
```

seconds needed = 4.374

As this does not become better if gradients are need i coded it
up in an external C program (DLL) to be called from

```
> currentdir(): theDLL:=cat(%,`\\Garch.dll`);
  # the DLL is assumed to be in the directory of this worksheet
```

theDLL := "C:_Work\Maple_Work\Finance\Garch\Garch.dll"

```
> extML := define_external(
    'MLE',
    'C',
    'x_array'::ARRAY(1..n,float[8],NO_COPY),
    'nX'::integer[4],
    'g_a0'::float[8],
    'g_a1'::float[8],
    'g_b1'::float[8],
    'RETURN'::float[8] ,
    LIB=theDLL):
```

which gives the same values, but is much faster

```
> extML(Timeseries, N, A0,A1,B1);
  ML(Timeseries, N, A0,A1,B1);
  ``;
  st:=time():
  for i from 1 to 100 do
    extML(Timeseries, N, A0,A1,B1+0.01/i);
  end do:
  `seconds needed`=time()-st; i:='i':
```

3193.13570898078796

3193.13570898078934

seconds needed = 0.025

The DLL contains the numerical gradient as well

```
> extgradML := define_external(
    'gradMLE',
    'C', # <--- calling without a wrapper
    'x_array'::ARRAY(1..n,float[8],NO_COPY),
    'nX'::integer[4],
    'g_a0'::float[8],
    'g_a1'::float[8],
    'g_b1'::float[8],
    'dy_array'::ARRAY(1..3,float[8],NO_COPY),
    'RETURN'::float[8] ,
    LIB=theDLL):
```

For proper use the storage for dy_array has to be provided in Maple,

the external function writes the values to it

```
> X:=hfarray([seq(0,i=1..3)]);  # 3 parameters/variables
  Y:=hfarray([seq(0,i=1..1)]);  # value in R^1
  dY:=hfarray([seq(0,i=1..3)]); # 3 real valued gradients
```

$$X := [0., 0., 0.]$$

$$Y := [0.]$$

$$dY := [0., 0., 0.]$$

As this functions are hardware floats and do not accept variables i want
to apply the optimization in *Matrix* form where they can be wrapped.

We need

1. the objective function

```
> objFunction:=proc(x)
  global Timeseries, N;
  extML(Timeseries, N, evalf[18](x[1]), evalf[18](x[2]), evalf[18](x[3]));
  return(-%); # <--- the package minimizes, so change sign
  end proc:
```

2. the numerical gradient

```
> objGradient := proc (x, dy)
  local res;
  global Timeseries, N, dY;
  res:=extgradML(Timeseries, N, x[1], x[2], x[3], dY);
  dy[1]:=-evalf[18](dY[1]); dy[2]:=-evalf[18](dY[2]);
  dy[3]:=-evalf[18](dY[3]);
  return (-res); # <--- the package minimizes, so change all signs
  end proc:
```

3. a starting point

```
> initParam:=[A0,A1,B1];
  initVector:=convert(initParam,Vector);
  #gradTest:=hfarray([seq(1,i=1..3)]);
  #objGradient(initVector, gradTest);: gradTest;
```

$$initParam := [0.000028959492044822, 0.21813969384208, 0.7]$$

$$initVector := \begin{bmatrix} 0.000028959492044822 \\ 0.21813969384208 \\ 0.7 \end{bmatrix}$$

4. linear constraints

```
> A := Matrix([0,1,1], datatype=float):
  b := Vector([1-(1e-20)], datatype=float):
  linearConstraint := [A, b]:
```

Then let it run ...

```
> infolevel[Optimization]:=3:
  st:=time():
  sol:=Optimization[NLPSolve](
    3,
    objFunction,
    linearConstraint,
    objectivegradient=objGradient,
    initialpoint=initVector,
    assume=nonnegative,
    optimalitytolerance=1e-14
```

```
  );

  ``; `time used[seconds]`=time()-st;
  infolevel[Optimization]:=0:


NLPSolve:   calling NLP solver
SolveGeneral:   calling solver for constrained nonlinear problems
SolveGeneral:   number of problem variables   3
SolveGeneral:   number of nonlinear inequality constraints   0
SolveGeneral:   number of nonlinear equality constraints   0
SolveGeneral:   number of general linear constraints   1
SolveGeneral:   feasibility tolerance set to   .1053671213e-7
SolveGeneral:   optimality tolerance set to   .1e-13
SolveGeneral:   iteration limit set to   50
SolveGeneral:   infinite bound set to    NAG default
SolveGeneral:   trying evalhf mode
SolveGeneral:   trying evalf mode
E04UCA:   iterates did not converge, but first-order conditions have been met
E04UCA:   number of major iterations taken   17
```

$$\text{sol} := \left[-3221.95100471500018,\ \begin{bmatrix} 0.305184918006717378\ 10^{-5} \\ 0.0961758879818718771 \\ 0.896612487709952054 \end{bmatrix} \right]$$

$$\text{time used[seconds]} = 0.116$$

The result is given almost immediate, the estimated parameters are
.305184918006717378e-5 , .961758879818718771e-1, .896612487709952054

where the (maximum) likelihood is 3221.95100471500018

```
> estimatedParameters:=convert(sol[2],list);
  estA0:=estimatedParameters[1];
  estA1:=estimatedParameters[2];
  estB1:=estimatedParameters[3];
```

$$\text{estimatedParameters} := [0.305184918006717378\ 10^{-5},\ 0.0961758879818718771,\ 0.896612487709952054]$$

$$\text{estA0} := 0.305184918006717378\ 10^{-5}$$

$$\text{estA1} := 0.0961758879818718771$$

$$\text{estB1} := 0.896612487709952054$$

Check the restriction:

```
> 'estA1+estB1': '%'=%;
```

$$\text{estA1} + \text{estB1} = 0.99278837569182$$

and the 'statistics':

```
> 'estA0/(1-estA1-estB1)': '%'=%; 'Var': '%'=%;
  '6*estA1^2/(1-estB1^2-2*estA1*estB1-3*estA1^2)': '%'=%;
  'Kurt': '%'=%;
```

$$\frac{\text{estA0}}{1 - \text{estA1} - \text{estB1}} = 0.00042318471534985$$

$$\text{Var} = 0.00035376720909099$$

$$\frac{6\ \text{estA1}^2}{1 - \text{estB1}^2 - 2\ \text{estA1}\ \text{estB1} - 3\ \text{estA1}^2} = -13.443300686010$$

$$\text{Kurt} = 4.6161870555263$$

while variance is more or less ok the 'kurtosis' is strange,
the formula holds iff the following relation is be satisfied:

```
> '(estB1^2+2*estA1*estB1+3*estA1^2)<1'; %;
```

$$\text{estB1}^2 + 2\,\text{estA1}\,\text{estB1} + 3\,\text{estA1}^2 < 1$$

$$1.0041283617670 < 1$$

So i should have to use a non-linear constraint for the Kurtosis ... but can not
find out how Maple wants to have that and to have it working :-(

```
> # Optimization[MatrixForm]
> b1^2+2*a1*b1+3*a1^2-1;
  subs(a0=V[1],a1=V[2],b1=V[3],%);
```

$$b1^2 + 2\,a1\,b1 + 3\,a1^2 - 1$$

$$V_3^{\,2} + 2\,V_2\,V_3 + 3\,V_2^{\,2} - 1$$

```
> nlc := proc(V, W, needc)
            if needc[1] > 0.0 then
              W[1] := V[3]^2+2*V[2]*V[3]+3*V[2]^2-(1-1e-14);
            end if:
        end proc:
> #initParam:=[0, 0.4, 0.4];
  sol2:=Optimization[NLPSolve](
    3,
    objFunction,
  1,
  nlc,
    linearConstraint,
    objectivegradient=objGradient,
    initialpoint=initVector,
    assume=nonnegative,
    optimalitytolerance=1e-14
  );
Error, (in Optimization:-NLPSolve) no improved point could be found; consider
increasing optimality tolerance and checking correctness of objective and
nonlinear constraint functions
```

so try it with a dump linearization for the additional constraint

```
> b1^2+2*a1*b1+3*a1^2-1;
  #estB1*b1+2*estA1*b1+3*estA1*a1-1;   evalf[3](%);
  0.9*b1+2*a1*0.9+3*0.1*a1-1;
  subs(a0=V[1],a1=V[2],b1=V[3],%);
  c1:=1.0*coeff(%,V[1]);
  c2:=1.0*coeff(%%,V[2]);
  c3:=1.0*coeff(%%%,V[3]);
```

$$b1^2 + 2\,a1\,b1 + 3\,a1^2 - 1$$

$$0.9\,b1 + 2.1\,a1 - 1.$$

$$0.9\,V_3 + 2.1\,V_2 - 1.$$

$$c1 := 0.$$

$$c2 := 2.10$$

$$c3 := 0.90$$

set up the linear constraint

```
> A := Matrix([0,1,1], datatype=float):
  b := Vector([1-(1e-20)], datatype=float):
  A2 := Matrix([c1,c2,c3], datatype=float):
  b2 := Vector([0.999], datatype=float):
  linearConstraint2 := [A, b, A2, b2]:
```

and let it run

```
> sol3:=Optimization[NLPSolve](
    3,
    objFunction,
    linearConstraint2,
    objectivegradient=objGradient,
    initialpoint=initVector,
    assume=nonnegative,
    optimalitytolerance=1e-14
  );
```

$$\text{sol3} := \left[-3221.80865149600004, \begin{bmatrix} 0.310166685456576986\ 10^{-5} \\ 0.0889988337829700214 \\ 0.902336054506403307 \end{bmatrix} \right]$$

```
> EstimatedParameters:=convert(sol3[2],list);
  EstA0:=EstimatedParameters[1];
  EstA1:=EstimatedParameters[2];
  EstB1:=EstimatedParameters[3];
```

$$\text{EstimatedParameters} := [\,0.310166685456576986\ 10^{-5},\ 0.0889988337829700214,\ 0.902336054506403307\,]$$

$$\text{EstA0} := 0.310166685456576986\ 10^{-5}$$

$$\text{EstA1} := 0.0889988337829700214$$

$$\text{EstB1} := 0.902336054506403307$$

This time it is ok:

```
> 'EstA1+EstB1': '%'=%; ``;
  'EstA0/(1-EstA1-EstB1)': '%'=%; 'Var': '%'=%; ``;
  '6*EstA1^2/(1-EstB1^2-2*EstA1*EstB1-3*EstA1^2)': '%'=%;
  'Kurt': '%'=%; ``;
  '(EstB1^2+2*EstA1*EstB1+3*EstA1^2)<1'; %;
```

$$\text{EstA1} + \text{EstB1} = 0.99133488828937$$

$$\frac{\text{EstA0}}{1 - \text{EstA1} - \text{EstB1}} = 0.00035794885953528$$

$$\text{Var} = 0.00035376720909099$$

$$\frac{6\ \text{EstA1}^2}{1 - \text{EstB1}^2 - 2\ \text{EstA1}\ \text{EstB1} - 3\ \text{EstA1}^2} = 33.620746008342$$

$$\text{Kurt} = 4.6161870555263$$

$$\text{EstB1}^2 + 2\ \text{EstA1}\ \text{EstB1} + 3\ \text{EstA1}^2 < 1$$

$$0.99858644556916 < 1$$

In many cases i played with that model the condition for the kurtosis is violated.

And if i enforce it to hold ... the estimated volatility is 'not realistic' for extreme situations. But who ever said that this is the right model or higher moments do

exist.

Anyway ... let's have some nice plots.

```
> myA0:=EstA0; myA1:=EstA1; myB1:=EstB1;
```

$$myA0 := 0.310166685456576986\ 10^{-5}$$

$$myA1 := 0.0889988337829700214$$

$$myB1 := 0.902336054506403307$$

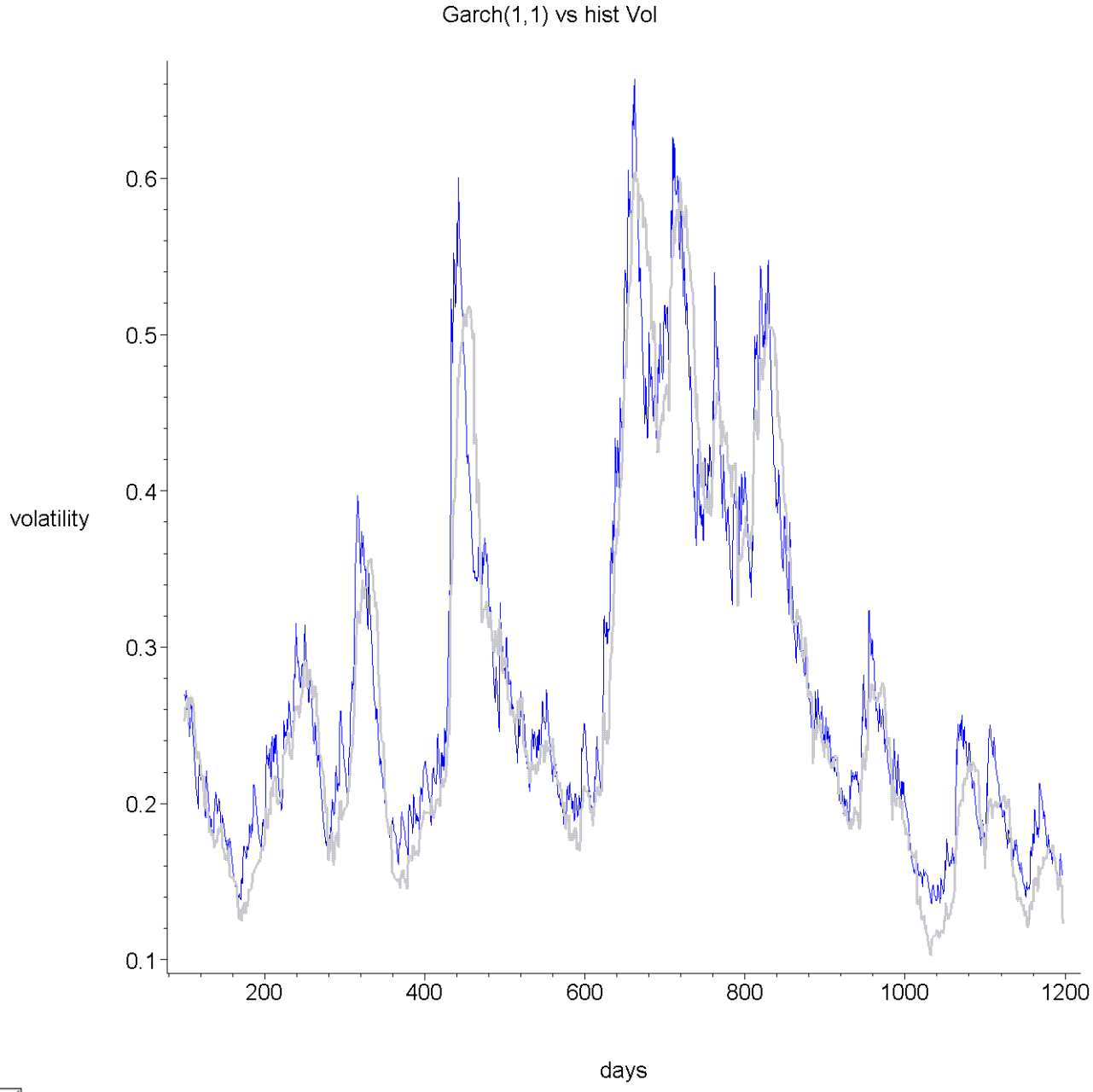
```
> condVar:=hfarray(1..N):
  condVar[1]:= evalf( myA0 / (1 - myA1 - myB1) ):
  for i from 2 to N do
  condVar[i]:= evalf( myA0 + myA1*Returns[i-1]^2 + myB1*condVar[i-1] );
  end do: i:='i':

> Vola:=map( x->sqrt(x)*sqrt(252.0),condVar):

> #P1:=plots[listplot]([seq([i+1,Spots[i+1]],i=1..N)], color=red,
  #  title = "Vola vs Index",axes=NONE):
  #P2:=plots[listplot]([seq([i,8000+10000*Vola[i]],i=1..N)], color=blue,
  axes=NONE):
  #plots[display]({P1,P2});
```

and see how historical volatility is reacting too slow (even for a 30 day frame):

```
> for i from 1 to 30 do
    MA30[i]:=0:
  end do: i:='i':
  for i from 31 to N do
    MA30[i]:=sqrt(252.0 * stats[describe, variance]([op(i-29..i,Returns)]) );:
  end do: i:='i':
> P4:=plots[listplot]([seq([i,MA30[i]],i=100..N)], color=grey,thickness=3):
  P3:=plots[listplot]([seq([i,Vola[i]],i=100..N)], color=blue,
    title="Garch(1,1) vs hist Vol", labels=["days", "volatility"]):
  plots[display]({P3,P4});
```



code for the DLL