A brute way to get a RND from option prices

AVt, Mar 2006

```
> restart;
> with(avtbslib):
  Digits:=14;
```

Digits := 14

Data for the Example

Data are for ODAX Settlement, 22 Sep 2005, vola is averaged to satisfy P/C parity and cut off on the down side (kicking off put prices below 1 Euro). At the upsite one can not cut much to have at least some right wing. But to have a chance for some simple solution I will remove calls with repeated 'prices' of 10 Cents.

Data

```
> dateData:=  "22 Sep 2005";
  dateExpiry:="200512";
```

dateData := "22 Sep 2005"

dateExpiry := "200512"

```
> Days:=85 - 7/24:
  Rates:=0.021600:
  RatesEuribor:=0.021578:
  Spot:=4865.076:
  Future:=4889.500:
  ATMVola:=0.15753:
```

```
> Time := Days/365.0;
  Fwd := Future;
  Rates:=RatesEuribor;
  Spot:= exp(-Rates*Time)*Fwd;
```

Time := 0.23207762557078

Fwd := 4889.500

Rates := 0.021578

Spot := 4865.0757103402

```
> # excercise, price call, price put, volatility #
  arrData:= [
  [3500.00,  1384.20,  1.50,  28.925]
  ,[3600.00,  1285.20,  2.10,  27.771]
  ,[3700.00,  1186.50,  2.90,  26.696]
  ,[3800.00,  1088.10,  4.00,  25.629]
  ,[3900.00,  990.00,  5.40,  24.484]
  ,[3950.00,  941.20,  6.40,  23.976]
  ,[4000.00,  892.60,  7.50,  23.457]
  ,[4050.00,  844.10,  8.80,  22.913]
  ,[4100.00,  795.90,  10.40,  22.406]
  ,[4150.00,  748.10,  12.20,  21.907]
  ,[4200.00,  700.50,  14.50,  21.414]
  ,[4250.00,  653.50,  17.10,  20.935]
  ,[4300.00,  606.90,  20.30,  20.462]
  ,[4350.00,  561.00,  24.10,  20.007]
  ,[4400.00,  515.80,  28.70,  19.565]
  ,[4450.00,  471.50,  34.10,  19.129]
  ,[4500.00,  428.10,  40.60,  18.700]
  ,[4550.00,  386.00,  48.20,  18.284]
  ,[4600.00,  345.20,  57.10,  17.867]
  ,[4650.00,  306.00,  67.60,  17.466]
  ,[4700.00,  268.50,  79.90,  17.070]
```

```
   ,[4750.00,  233.00,  94.20,  16.685]
   ,[4800.00,  199.60,  110.60,  16.297]
   ,[4850.00,  168.80,  129.50,  15.931]
   ,[4900.00,  140.60,  151.00,  15.574]
   ,[4950.00,  115.30,  175.50,  15.246]
   ,[5000.00,  93.10,  203.10,  14.953]
   ,[5050.00,  74.10,  233.80,  14.701]
   ,[5100.00,  58.10,  267.60,  14.492]
   ,[5150.00,  45.00,  304.20,  14.325]
   ,[5200.00,  34.50,  343.40,  14.205]
   ,[5250.00,  26.10,  384.80,  14.112]
   ,[5300.00,  19.60,  428.10,  14.061]
   ,[5350.00,  14.60,  472.80,  14.026]
   ,[5400.00,  10.80,  518.80,  14.025]
   ,[5450.00,  7.90,  565.60,  14.013]
   ,[5500.00,  5.70,  613.20,  14.016]
   ,[5600.00,  2.90,  709.90,  14.038]
   ,[5800.00,  0.70,  906.70,  14.209]
   ,[6000.00,  0.20,  1105.20,  14.840]
   ,[6200.00,  0.10,  1304.20,  16.485]
   ,[6400.00,  0.10,  1503.20,  18.463]
   ]:
> nData:=nops(arrData);
```

$$nData := 42$$

```
> arrData:=select( p -> is(p[1] < 6400), arrData):;
  nData:=nops(arrData);
```

$$nData := 41$$

```
> arrayStrikes:=map( x -> x[1], arrData):  # strikes
  arrayCalls:=map( x -> x[2], arrData):    # settlement prices calls
  arrayPuts:=map( x -> x[3], arrData):     # settlement prices puts
  arrayVola:=map( x -> x[4]/100, arrData): # averaged settlement vola

> arrayMoneyness:= map( strike -> ln(strike/Fwd), arrayStrikes):
  #arrayVariance:= map( vola -> vola^2*Time, arrayVola): # not used
```

We will fit between the minimal and the maximal strikes

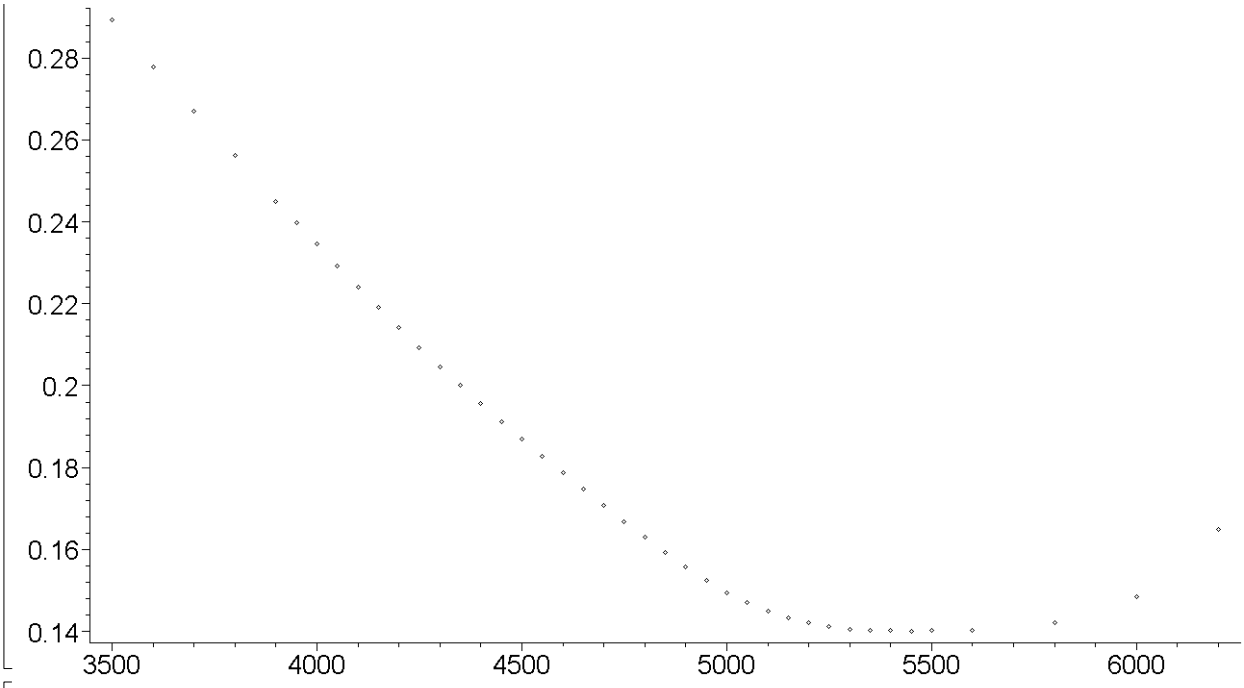
```
> data2fit:= [seq([arrayStrikes[i],arrayVola[i]], i=1..nData)]:
  minK:=min(op(arrayStrikes));
  maxK:=max(op(arrayStrikes));
```

$$minK := 3500.00$$

$$maxK := 6200.00$$

Looking at the data already indicates some problems: the right wing almost does not exist ("penny values") and we can not really identify a minimum, the data are jumpy at the right wing - and we have some broader valley (and not a unique minimum) while the downside looks pretty simple:

```
> plots[pointplot](data2fit); pVar := %:
```



But even if that is not quite realistic (settlement ...) it is a good data set for a test.

>

>

Roughly fitting by a polynom

I prefer an even degree, at least it should be 4 and to avoid overfitting there should be '4 - 6 data for each indeterminate'. Since we have to differentiate twice a high degree may cause troubles at the boundaries, so it may be better to demand for more data for each parameter. If there are less data then some judging is needed for the wings.

```
> (theDegree + 1) = nData/6;
  2*round(solve(%,theDegree)/2):
  myDegree:=max(%, 4);
```

$$\text{theDegree} + 1 = \frac{41}{6}$$

$$\text{myDegree} := 6$$

```
> poly:= xi -> a[0] + sum( a[i]*xi^i, i=1..myDegree); #indets(%,atomic):
  indets(%);
```

$$\text{poly} := \xi \rightarrow a_0 + \left(\sum_{i=1}^{\text{myDegree}} a_i \, \xi^i \right)$$

There is no need to use monomials as basis for fitting, but it is a good thing to scale data befor fitting (Maple would moan if not, Excel would not) and this is automatically the case if one works over (logarithmic) moneyness.

Afterwards switch back to the actual coordinates over the strikes.

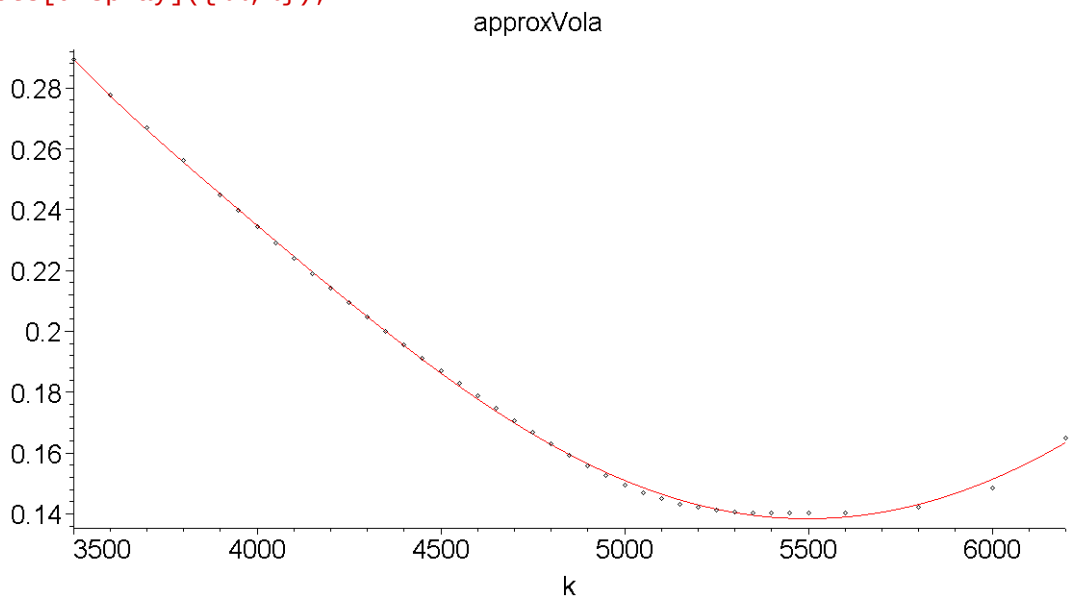
```
> Statistics:-Fit(poly(xi), arrayMoneyness, arrayVola, xi):
  approx_mny:=unapply(sort(%),xi);
  ``;
  approxVola:= K -> approx_mny( ln(K/Fwd));
```

$$\text{approx\_mny} := \xi \rightarrow 6.81982693112829441\ \xi^6 - 3.64185000489218513\ \xi^5 - 0.728466884175203888\ \xi^4$$

$$+ 2.17236630928862872\,\xi^3 + 0.898182373840632154\,\xi^2 - 0.292590568990153954\,\xi + 0.157020397214651498$$

$$\text{approxVola} := K \to \text{approx\_mny}\left(\ln\left(\frac{K}{\text{Fwd}}\right)\right)$$

```
> plots[pointplot](data2fit):
  plot(approxVola(k), k=minK..maxK, title="approxVola"):
  plots[display]({%%,%});
```

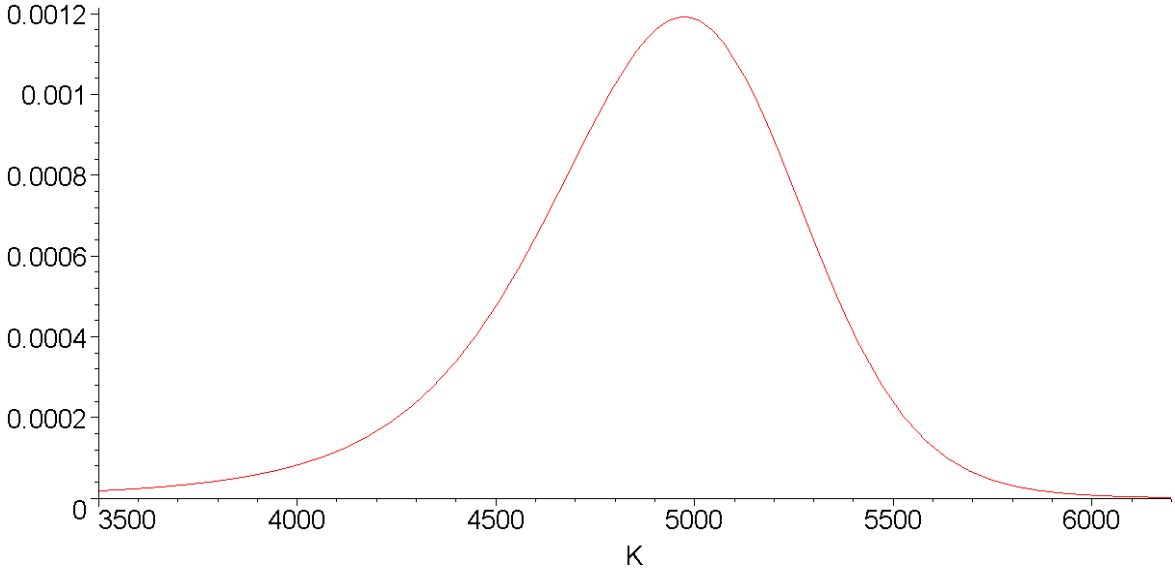


That simple way can not catch the valley and at the upside one can not expect to much for the data taken. But if they are good (say: from a desk) that should be ok.

The lame un-discounted risk neutral density follows from Breeden-Litzenberger and looks as expected:

```
> RND:= K -> exp(Rates*Time)*diff(BSCall(Spot,K,Time,Rates,approxVola(K)),
  K$2); evalf(%):

  plot(RND(K), K=minK..maxK);
```

$$\text{RND} := K \to \mathbf{e}^{(\text{Rates Time})}\left(\frac{\partial^2}{\partial K^2}\,\text{BSCall}(\text{Spot, K, Time, Rates, approxVola}(K))\right)$$



If I would not have kicked off strike = 6400 (with 10 Cents) the right side would indicate to do just that. It will never give something reasonable picture beyond the observable range, ie the tails are missing.

The above can be easily done in Excel as well by just coding using the following formula:

```
> 'diff(BSCall(s,e,t,r,v(e)),e$2)' =
  'exp(-r*t)*diffN(dTwo(s,e,t,r,v(e)))/v(e) *
    (1/e/sqrt(t) +2*dOne(s,e,t,r,v(e))*diff(v(e),e) +
    dOne(s,e,t,r,v(e))*dTwo(s,e,t,r,v(e))*e*sqrt(t)*(diff(v(e),e))^2 +
  e*sqrt(t)*diff(v(e),e$2)*v(e))';
  expand(%): simplify(%) assuming 0<s, 0<e: is(%) assuming 0<s, 0<e;
```

$$\frac{\partial^2}{\partial e^2}\,\mathrm{BSCall}(s, e, t, r, v(e)) = \mathbf{e}^{(-r\,t)}\,\mathrm{diffN}(\mathrm{dTwo}(s, e, t, r, v(e)))\left(\frac{1}{e\sqrt{t}} + 2\,\mathrm{dOne}(s, e, t, r, v(e))\left(\frac{d}{de}v(e)\right)\right.$$
$$\left. + \mathrm{dOne}(s, e, t, r, v(e))\,\mathrm{dTwo}(s, e, t, r, v(e))\,e\sqrt{t}\left(\frac{d}{de}v(e)\right)^2 + e\sqrt{t}\left(\frac{d^2}{de^2}v(e)\right)v(e)\right) / v(e)$$

true

[>

Norming: fwd = 1, time = 1, rates = 0

It is worth to norm the situation to time = 1, rates = 0 and fwd = 1 (please ignore that I use the future)

```
> fwd:=exp(r*t)*s;
  'exp(r*t)*BSCall(s,e,t,r,v(e)) = fwd*BSCall(1, e/fwd,t,0,v(e))';
  simplify(%,symbolic): is(%);
  'exp(r*t)*BSCall(s,e,t,r,v(e)) = fwd*BSCall(1, e/fwd,1,0,v(e)*sqrt(t))';
  simplify(%,symbolic): is(%);
```

$$\mathrm{fwd} := \mathbf{e}^{(r\,t)}\,s$$

$$\mathbf{e}^{(r\,t)}\,\mathrm{BSCall}(s, e, t, r, v(e)) = \mathrm{fwd}\,\mathrm{BSCall}\left(1, \frac{e}{\mathrm{fwd}}, t, 0, v(e)\right)$$

true

$$\mathbf{e}^{(r\,t)}\,\mathrm{BSCall}(s, e, t, r, v(e)) = \mathrm{fwd}\,\mathrm{BSCall}\left(1, \frac{e}{\mathrm{fwd}}, 1, 0, v(e)\sqrt{t}\right)$$

true

and to work with re-scaled data

```
> arrayStrikes:='map( x -> x[1]/Fwd, arrData)';
  arrayCalls:='map( x -> exp(Rates*Time)*x[2]/Fwd, arrData)';
  arrayPuts:='map( x -> exp(Rates*Time)*x[3]/Fwd, arrData)';
  arrayVola:='map( x -> sqrt(Time)*x[4]/100, arrData)';
  ``;
  arrayMoneyness:= 'map( strike -> ln(strike/1), arrayStrikes)';
  #arrayVariance:= 'map( vola -> vola^2, arrayVola)';
  ``;
  data2fit:= [seq([arrayStrikes[i],arrayVola[i]], i=1..nData)]:

  minK:=min(op(arrayStrikes));
  maxK:=max(op(arrayStrikes));
  minMny:=ln(minK);
  maxMny:=ln(maxK);
```

$$\text{arrayStrikes} := \text{map}\left(x \to \frac{x_1}{\text{Fwd}}, \text{arrData}\right)$$

$$\text{arrayCalls} := \text{map}\left(x \to \frac{\mathbf{e}^{(\text{Rates Time})}\, x_2}{\text{Fwd}}, \text{arrData}\right)$$

$$\text{arrayPuts} := \text{map}\left(x \to \frac{\mathbf{e}^{(\text{Rates Time})}\, x_3}{\text{Fwd}}, \text{arrData}\right)$$

$$\text{arrayVola} := \text{map}\left(x \to \frac{1}{100}\sqrt{\text{Time}}\, x_4, \text{arrData}\right)$$

$$\text{arrayMoneyness} := \text{map}(\text{strike} \to \ln(\text{strike}), \text{arrayStrikes})$$

$$\text{minK} := 0.71581961345741$$

$$\text{maxK} := 1.2680233152674$$

$$\text{minMny} := -0.33432708027482$$

$$\text{maxMny} := 0.23745924328085$$

Remark: to judge fitting it would be better to work with re-calculated prices from the vola matrix, as fitting is done through vola and not directly through prices.

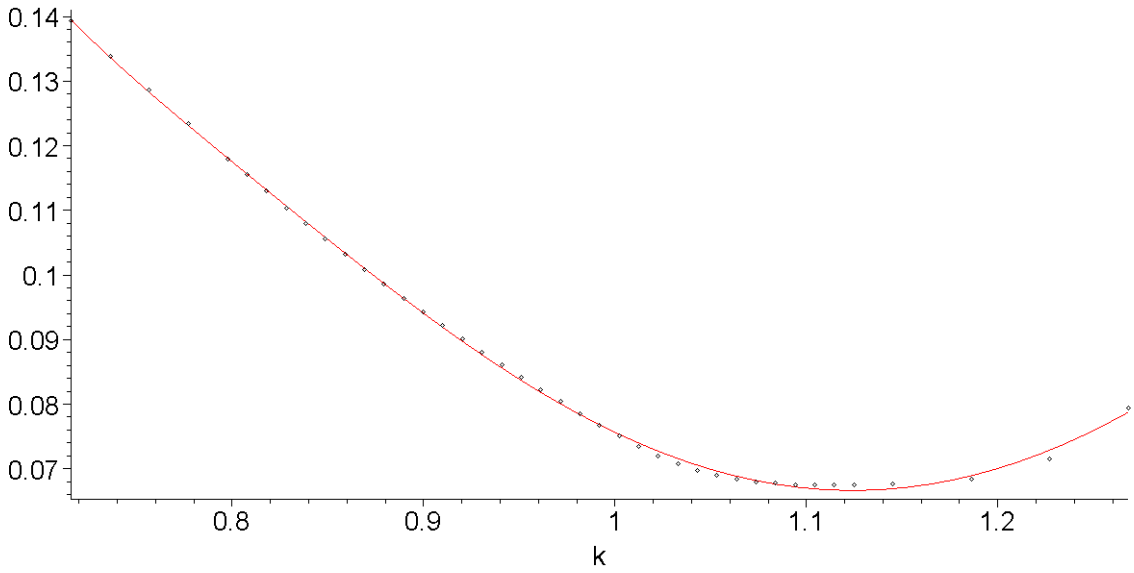
Fitting is done the same way

```
> poly:= xi -> a[0] + sum( a[i]*xi^i, i=1..myDegree);

  Statistics:-Fit(poly(xi), arrayMoneyness, arrayVola, xi):
  approx_mny:=unapply(sort(%),xi);

  approx:= K -> approx_mny( ln(K/1));

  plots[pointplot](data2fit):
  plot(approx(k), k=minK..maxK):
  plots[display]({%%,%});
```

$$\text{poly} := \xi \to a_0 + \left(\sum_{i=1}^{\text{myDegree}} a_i\, \xi^i\right)$$

$$\begin{aligned}\text{approx\_mny} := \xi \to\ & 3.28541314033208298\, \xi^6 - 1.75444068914613616\, \xi^5 - 0.350934810769839400\, \xi^4 \\ & + 1.04652521099237306\, \xi^3 + 0.432694290219424738\, \xi^2 - 0.140953855543503676\, \xi \\ & + 0.0756436903033729790\end{aligned}$$

$$\text{approx} := K \to \text{approx\_mny}(\ln(K))$$

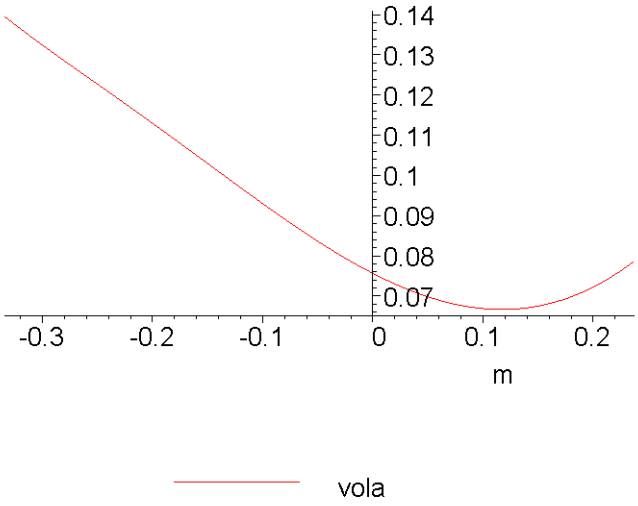


Vola is the same up to scaling by $\sqrt{\text{time}}$, so one works with standard deviation (no annualisation).

```
> approx_vola := m -> approx_mny(m); #*sqrt(Time);
```

$$\text{approx\_vola} := m \to \text{approx\_mny}(m)$$

```
> plot(approx_vola(m), m=minMny..maxMny, legend="vola"); ;
```




```
> 
```

```
> 
```

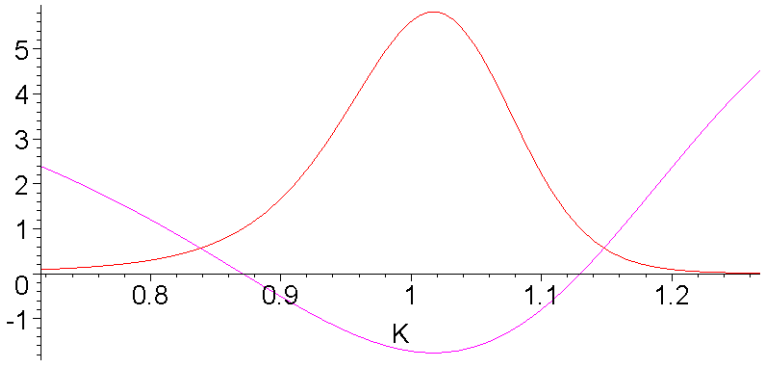
Lazy RND and PDF for the normed situation

Naming convention: PDF = density in log space, RND = density over strikes (both un-discounted).

While vola is the same up to scaling by $\sqrt{\text{time}}$, the RND differs by a factor, which is the forward:

```
> RND:= K -> diff(BSCall(1,K,1,0,approx(K)), K$2);
  RND:=unapply(diff(BSCall(1,K,1,0,approx(K)), K$2), K):
  plot(RND(K), K=minK..maxK, legend="     RND(K)"):
  plot(-ln(RND(K)), K=minK..maxK, color= magenta, legend="- ln(RND(K))"):
  plots[display](%%,%);
```

$$\mathrm{RND} := \mathrm{K} \to \frac{d^2}{dK^2}\,\mathrm{BSCall}(1, \mathrm{K}, 1, 0, \mathrm{approx}(\mathrm{K}))$$

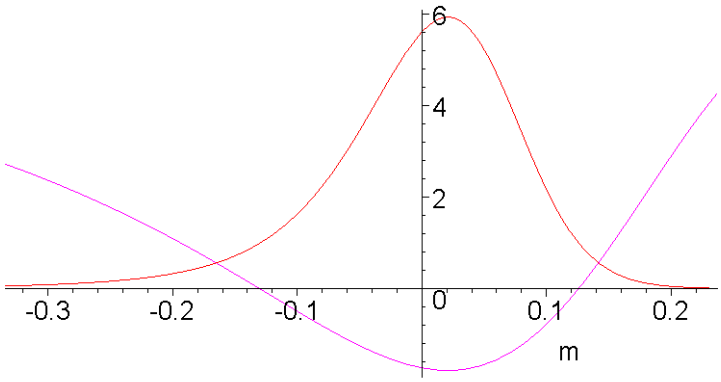


RND(K)

- ln(RND(K))

```
> PDF:= m -> exp(m)*RND(exp(m));
  exp(m)*RND(exp(m)) assuming m::real:
  PDF:= unapply(%, m):

  plot(PDF(m), m=ln(minK)..ln(maxK), legend="     PDF(m)"):
  plot(-ln(PDF(m)), m=ln(minK)..ln(maxK), color= magenta, legend="-
  ln(PDF(m))"):
  plots[display](%%,%);
```

$$\mathrm{PDF} := \mathrm{m} \to \mathbf{e}^{\mathrm{m}}\,\mathrm{RND}(\mathbf{e}^{\mathrm{m}})$$



PDF(m)

- ln(PDF(m))

At least at this point one would check whether extreme wings should be shortend and whether there is an over-fitting (polynomial's degree is too high).

```
>
```

```
>
```

Extending to the full range

The idea is to extend by the 'usual BS case' through appropriate glueing: of course that does not guarantee all theoretical needs - but it is relatively simple and fast enough for computations.

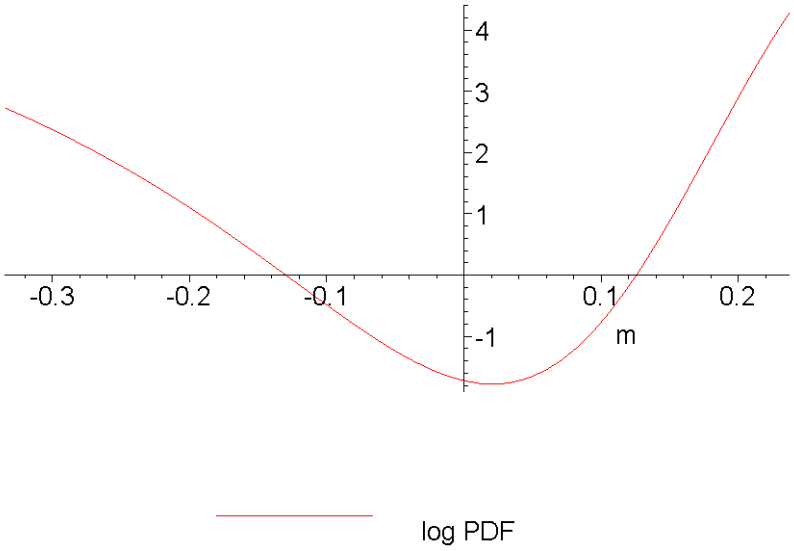
The pdf for Black-Scholes in the normed case is given as

```
> Q:= (m,v) -> exp(-1/2*(m/v+1/2*v)^2)/v/sqrt(2*Pi);
```

$$Q := (m, v) \to \frac{e^{\left(-1/2\left(\frac{m}{v}+1/2\,v\right)^2\right)}}{v\sqrt{2\,\pi}}$$

Instead of immediately working with the PDF it is better to look at its logarithm (times -1):

```
> plot(-ln(PDF(m)), m=minMny..maxMny, legend="log PDF");
```



The simpliest is to continue by choosing appropriate vola at the left and right:

```
> '-ln(PDF(minMny)) = -ln(Q(minMny,v))';
    fsolve(%,v=approx_vola(minMny)): v_lower :=%;

  '-ln(PDF(maxMny)) = -ln(Q(maxMny,v))';
    fsolve(%,v=approx_vola(maxMny)): v_upper :=%;
```

$$-\ln(\mathrm{PDF}(\mathrm{minMny})) = -\ln(Q(\mathrm{minMny}, v))$$

$$\mathrm{v\_lower} := 0.11645302118353$$

$$-\ln(\mathrm{PDF}(\mathrm{maxMny})) = -\ln(Q(\mathrm{maxMny}, v))$$

$$\mathrm{v\_upper} := 0.069008806237042$$

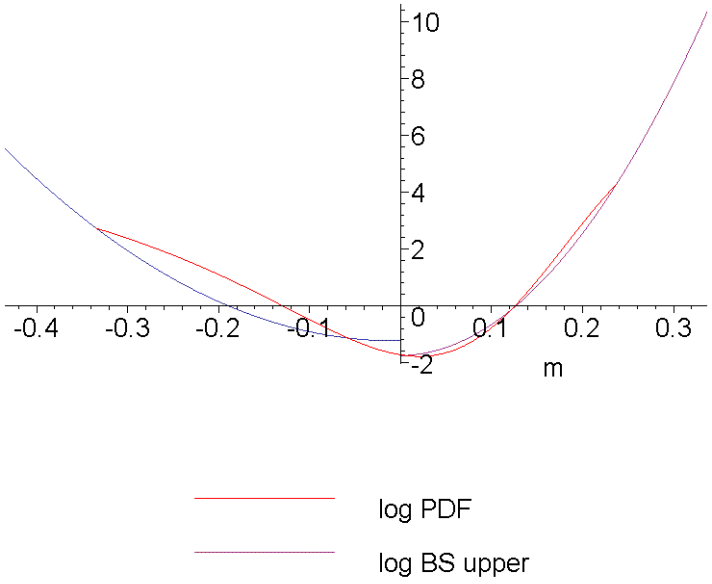
```
> fullPDF:= m -> piecewise(
    m < minMny, Q(m,v_lower),
    maxMny < m, Q(m,v_upper),
    PDF(m));
```

$$\mathrm{fullPDF} := m \to \mathrm{piecewise}(m < \mathrm{minMny}, Q(m, \mathrm{v\_lower}), \mathrm{maxMny} < m, Q(m, \mathrm{v\_upper}), \mathrm{PDF}(m))$$

which at first glance looks not bad

```
> plot(-ln(PDF(m)), m=minMny..maxMny, legend="log PDF"):
  plot(-ln(Q(m,v_upper)), m=0..maxMny + 0.1, color = maroon,legend="log BS
  upper"):
  plot(-ln(Q(m,v_lower)), m=-0.1 + minMny..0, color = navy, legend="log BS
  lower"):
```

```
plots[display](%%%,%%,%);
```



To be an actual probability function it has to integrate to 1 over the Reals. Due to the simple construction
that can be done by splitting into the used ranges, where for the wings explicite solutions exists:

```
> 'Int(PDF(m), m= minMny..maxMny)': '%'= evalf(%);
  ``;
  'Int(Q(mu-m0,v), mu= -infinity..m)' = 'N((m-m0)/v+v/2)';
  value(%) assuming 0 < v: is(%);

  'Int(Q(mu-m0,v), mu= m..infinity)' = '1-N((m-m0)/v+v/2)';
  value(%) assuming 0 < v: is(%);
  ``;
  'Int( fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int(PDF(m), m= minMny..maxMny)' +
    'eval(N(m/v_lower+1/2*v_lower), m=minMny)' +
    'eval(1-N(m/v_upper+1/2*v_upper), m=maxMny)';
  '%%'= evalf(rhs(%));
```

$$\int_{minMny}^{maxMny} PDF(m)\,dm = 0.99548819922415$$

$$\int_{-\infty}^{m} Q(\mu - m0, v)\,d\mu = N\left(\frac{m - m0}{v} + \frac{v}{2}\right)$$

true

$$\int_{m}^{\infty} Q(\mu - m0, v)\,d\mu = 1 - N\left(\frac{m - m0}{v} + \frac{v}{2}\right)$$

true

$$\int_{-\infty}^{\infty} fullPDF(m)\,dm =$$

$$\int_{\text{minMny}}^{\text{maxMny}} \text{PDF}(m)\,dm + N\left(\frac{m}{v\_lower} + \frac{v\_lower}{2}\right)\Bigg|_{m=\text{minMny}} + \left(1 - N\left(\frac{m}{v\_upper} + \frac{v\_upper}{2}\right)\right)\Bigg|_{m=\text{maxMny}}$$

$$\int_{-\infty}^{\infty} \text{fullPDF}(m)\,dm = 0.99819958457115$$

One can not simply re-norm that to 1, since in the central range the PDF is given through the desired volatility smile. Instead one has to glue better - by shifting the Black-Scholes pdf away from 0 using some $Q(m - m0, v)$.

One way is to use first variant the extremum of log PDF (which can be computed numerically) for the right wing (or left in other cases) and determine the rest in a way that gives the desired result.

But I prefer a 'smooth' extension: due to fitting errors and bad boundary behaviour one should not sit at the exact boundary, so go a bit into the interior, say 95%:

```
> '-ln(PDF(0.95*maxMny)) = -ln(Q(0.95*maxMny-mU,vU))';
  'eval(diff(-ln(PDF(m)),m),m=0.95*maxMny) =
  eval(diff(-ln(Q(m-mU,vU)),m),m=0.95*maxMny)';

  fsolve({%,%%},{mU=0,vU=approx_vola(maxMny)},fulldigits);

  select(has,%,mU): op(%): m_upper:= rhs(%);
```

$$-\ln(\text{PDF}(0.95\ \text{maxMny})) = -\ln(Q(0.95\ \text{maxMny} - mU, vU))$$

$$\left(\frac{d}{dm}(-\ln(\text{PDF}(m)))\right)\Bigg|_{m=0.95\ \text{maxMny}} = \left(\frac{\partial}{\partial m}(-\ln(Q(m - mU, vU)))\right)\Bigg|_{m=0.95\ \text{maxMny}}$$

$$\{\, vU = 0.091059301163105,\ mU = -0.068284883763726 \,\}$$

$$m\_upper := -0.068284883763726$$

Then it is smoothly extended to the right - but I do not want to use the vola need to have enough freedoms to make it an actual pdf, for which we need the total integral or 'weight':

```
> 'Int( fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int(PDF(m), m= minMny..maxMny)' +
    'eval(N((m-mL)/vL+1/2*vL), m=minMny)' +
    'eval(1-N((m-m_upper)/vU+1/2*vU), m=maxMny)';
  ``;
  theWeight:=evalf(rhs(%%)); #indets(%,atomic): indets(%);
```

$$\int_{-\infty}^{\infty} \text{fullPDF}(m)\,dm =$$

$$\int_{\text{minMny}}^{\text{maxMny}} \text{PDF}(m)\,dm + N\left(\frac{m - mL}{vL} + \frac{vL}{2}\right)\Bigg|_{m=\text{minMny}} + \left(1 - N\left(\frac{m - m\_upper}{vU} + \frac{vU}{2}\right)\right)\Bigg|_{m=\text{maxMny}}$$

$$theWeight := 1.9954881992242$$
$$+ 0.50000000000000\ \text{erf}\left(\frac{0.70710678118655\ (-0.33432708027482 - 1.\ mL)}{vL} + 0.35355339059328\ vL\right)$$
$$- 0.50000000000000\ \text{erf}\left(\frac{0.21619374554118}{vU} + 0.35355339059328\ vU\right)$$

Now one can set up the neccessary conditions to get the indeterminates,

the logarithms should fit together and the weight has to be 1:

```
> eqs:='{theWeight=1,
    -ln(PDF(minMny)) = -ln(Q(minMny-mL,vL)),
    -ln(PDF(maxMny)) = -ln(Q(maxMny-m_upper,vU))}'; #indets(%,atomic):
  indets(%);
  ``;
  initials:= '{mL=0,vL=approx_vola(minMny),vU=approx_vola(maxMny)}';
```

$$eqs := \{-\ln(PDF(minMny)) = -\ln(Q(minMny - mL, vL)),\ theWeight = 1,\ -\ln(PDF(maxMny)) = -\ln(Q(maxMny - m\_upper, vU))\}$$

$$initials := \{mL = 0,\ vL = approx\_vola(minMny),\ vU = approx\_vola(maxMny)\}$$

After providing some initial guess one can use a numerical solver and hope for results:

```
> sols:=fsolve(eqs,initials,fulldigits);:

  select(has,sols,vU): op(%): v_upper:= rhs(%);
  select(has,sols,vL): op(%): v_lower:= rhs(%);
  select(has,sols,mL): op(%): m_lower:= rhs(%);
```

$$sols := \{vL = 0.18736393503912,\ vU = 0.091312973356008,\ mL = 0.17752923576271\}$$

$$v\_upper := 0.091312973356008$$

$$v\_lower := 0.18736393503912$$

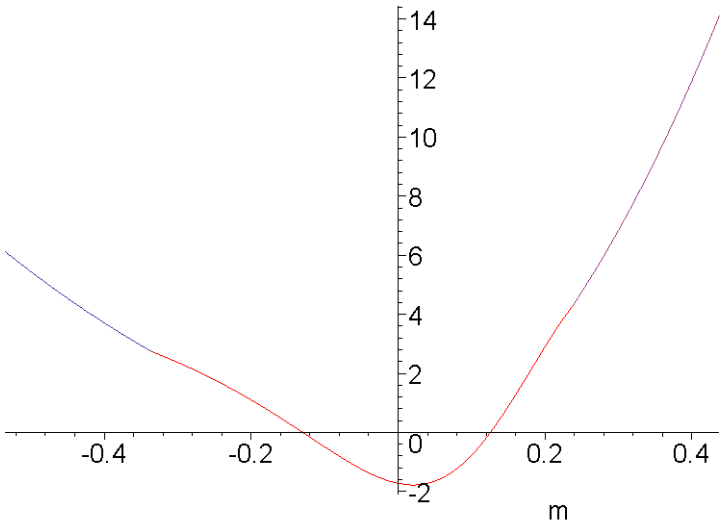
$$m\_lower := 0.17752923576271$$

which then is taken as the extension to the Reals

```
> fullPDF:= m -> piecewise(
    m < minMny, Q(m-m_lower,v_lower),
    minMny <= m and m <= maxMny, PDF(m),
    maxMny < m, Q(m-m_upper,v_upper)
    ); ``;
```

$$fullPDF := m \rightarrow piecewise(m < minMny,\ Q(m - m\_lower, v\_lower),\ minMny \le m \textbf{ and } m \le maxMny,\ PDF(m),\ maxMny < m,\ Q(m - m\_upper, v\_upper))$$

```
> plot(-ln(PDF(m)), m=minMny..maxMny, legend="log PDF"):
  plot(-ln(Q(m-m_upper,v_upper)), m=maxMny..maxMny + 0.2, color =
  maroon,legend="log BS upper"):
  plot(-ln(Q(m-m_lower,v_lower)), m=-0.2 + minMny..minMny, color = navy,
  legend="log BS lower"):
  plots[display](%%%,%%,%);
```




```
> 'Int( fullPDF(m),m=-infinity..infinity,method = _NCrule)':
  '%'= evalf(%); #evalf(subs(infinity=2,%));
```

$$\text{Int}(\text{fullPDF}(m), m = -\infty .. \infty, \text{method} = \text{\_NCrule}) = 1.0000000000028$$

```
> 'Int( exp(m)*fullPDF(m),m=-infinity..infinity,method = _NCrule)':
  '%'= evalf(%); #evalf(subs(infinity=2,%));
```

$$\text{Int}(e^{m}\,\text{fullPDF}(m), m = -\infty .. \infty, \text{method} = \text{\_NCrule}) = 1.0001422631766$$

Actually the last value should by exactly 1 (a call with strike 0) - but I do not have a way to reach it with that
brute way (and there is no theoretical reason for that). So live with it, see what it gives and what troubles
will occure - see below.

```
>
```

```
>
```

Pricing with the PDF

To judge the prices we will need the vola, it was

```
> volaFct_mny:= m -> approx_mny(m);
```

$$\text{volaFct\_mny} := m \rightarrow \text{approx\_mny}(m)$$

```
> volaFct:= K -> approx(K/Fwd)/sqrt(Time);
```

$$\text{volaFct} := K \rightarrow \frac{\text{approx}\left(\frac{K}{\text{Fwd}}\right)}{\sqrt{\text{Time}}}$$

i e: scale the strike to the normed situation, take the standard deviation over the moneyness in log space and annualize.

For integrating pay-offs against the PDF it is convenient to to use cut-offs where the integrand will be small enough for numerical needs:

```
> cutLower:= -1.6;
  cutUpper:= 0.8;
  ``;
  'fullPDF(cutLower)': '%'= evalf(%,3);
  'fullPDF(cutUpper)': '%'= evalf(%,3);
```

$$\text{cutLower} := -1.6$$

$$\text{cutUpper} := 0.8$$

$$\text{fullPDF}(\text{cutLower}) = 0.112\ 10^{-18}$$

$$\text{fullPDF}(\text{cutUpper}) = 0.622\ 10^{-19}$$

This is wide enough to price a wide range of calls (in the original situation):

```
> 'ln(K/Fwd)=cutLower': 'cutLower_strike'= evalf(fsolve(%,K),2);
  'ln(K/Fwd)=cutUpper': 'cutUpper_strike'= evalf(fsolve(%,K),2);
```

$$\text{cutLower\_strike} = 1000.$$

$$\text{cutUpper\_strike} = 11000.$$

For pricing with the PDF in the normed situation one uses

```
> C_num:= K -> Int((exp(mu)-K)*fullPDF(mu),mu = ln(K) .. infinity);
  C_num:= K ->
    evalf(Int((exp(mu)-K)*fullPDF(mu),mu = max(cutLower,ln(K)) ..
  cutUpper,method = _d01ajc));
```

$$\text{C\_num} := K \to \int_{\ln(K)}^{\infty} (e^{\mu} - K)\ \text{fullPDF}(\mu)\ d\mu$$

$$\text{C\_num} := K \to \text{evalf}(\text{Int}((e^{\mu} - K)\ \text{fullPDF}(\mu), \mu = \max(\text{cutLower}, \ln(K)) .. \text{cutUpper}, \text{method} = \_d01ajc))$$

The latter re-definition means to integrate within the cut-offs only (and that Maple should use a purely numerical routine as method).

This is for lazyness only and likewise one makes use of the simple extension at the tails, which does admitt explicte pricing formulae in that range like the Black-Scholes formula:

```
> 'Int((exp(mu)-exp(m))*Q(mu-mU,vU),mu = m0 .. infinity)':
  '%'='exp(mU)*N((mU-m0)/vU+1/2*vU)-exp(m)*N((mU-m0)/vU-1/2*vU)';
  collect(normal(value(%) assuming 0 < vU),exp):
  is(%);
```

$$\int_{m0}^{\infty} (e^{\mu} - e^{m})\ Q(\mu - mU, vU)\ d\mu = e^{mU}\ N\left(\frac{mU - m0}{vU} + \frac{vU}{2}\right) - e^{m}\ N\left(\frac{mU - m0}{vU} - \frac{vU}{2}\right)$$

$$\text{true}$$

For actual pricing (we started with almost real data) one has to scale back:

```
> Call_num:= strike ->
    exp(-Rates*Time)*Fwd*Int((exp(mu)-strike/Fwd)*fullPDF(mu),mu =
  ln(strike/Fwd) .. infinity);
  Call_num:= strike -> Spot *
    evalf(Int((exp(mu)-strike/Fwd)*fullPDF(mu),mu =
```

```
  max(ln(strike/Fwd),cutLower) .. cutUpper,
    method = _d01ajc));
```

$$\text{Call\_num} := \text{strike} \to \mathbf{e}^{(-\text{Rates Time})}\,\text{Fwd}\int_{\ln\left(\frac{\text{strike}}{\text{Fwd}}\right)}^{\infty}\left(\mathbf{e}^{\mu}-\frac{\text{strike}}{\text{Fwd}}\right)\text{fullPDF}(\mu)\,d\mu$$

$$\text{Call\_num} := \text{strike} \to \text{Spot}$$

$$\text{evalf}\left(\text{Int}\left(\left(\mathbf{e}^{\mu}-\frac{\text{strike}}{\text{Fwd}}\right)\text{fullPDF}(\mu),\ \mu=\max\left(\ln\left(\frac{\text{strike}}{\text{Fwd}}\right),\text{cutLower}\right)..\ \text{cutUpper},\ \text{method}=\_\text{d01ajc}\right)\right)$$

As a first and trivial check look at a call with strike almost 0 - that should give the spot:

```
> 'Call_num(1e-16)': '%'= %;
  'Spot': '%'= evalf(%);
```

$$\text{Call\_num}(0.1\ 10^{-15}) = 4865.7678317843$$

$$\text{Spot} = 4865.0757103402$$

Take a strike close below the forward and look at the original data: that is quite close.

```
> kTst:= evalf(floor(Fwd/100)*100,6);
  'Call_num(kTst)': '%'= evalf(%);
  ``;
  select( p -> is(p[1] = kTst), arrData): op(%);:
  kTstPrice:= %[2];
  kTstVola:= %%[4]/100;
```

$$\text{kTst} := 4800.$$

$$\text{Call\_num}(\text{kTst}) = 199.30762125933$$

$$[4800.00,\ 199.60,\ 110.60,\ 16.297]$$

$$\text{kTstPrice} := 199.60$$

$$\text{kTstVola} := 0.16297000000000$$

Now compare 2 things: one is what BS would give for the vola of 16.297 % (any difference is due to consistency problems on data input) and the other is what BS gives for the fitted vola (on which the PDF repends, so problems of fitting the vola came in):

```
> 'BSCall(Spot,kTst,Time,Rates,kTstVola)': '%'= evalf(%);
  'BSCall(Spot,kTst,Time,Rates,volaFct(kTst))': '%'= evalf(%);
```

$$\text{BSCall}(\text{Spot},\ \text{kTst},\ \text{Time},\ \text{Rates},\ \text{kTstVola}) = 199.62918790903$$

$$\text{BSCall}(\text{Spot},\ \text{kTst},\ \text{Time},\ \text{Rates},\ \text{volaFct}(\text{kTst})) = 199.40276237171$$

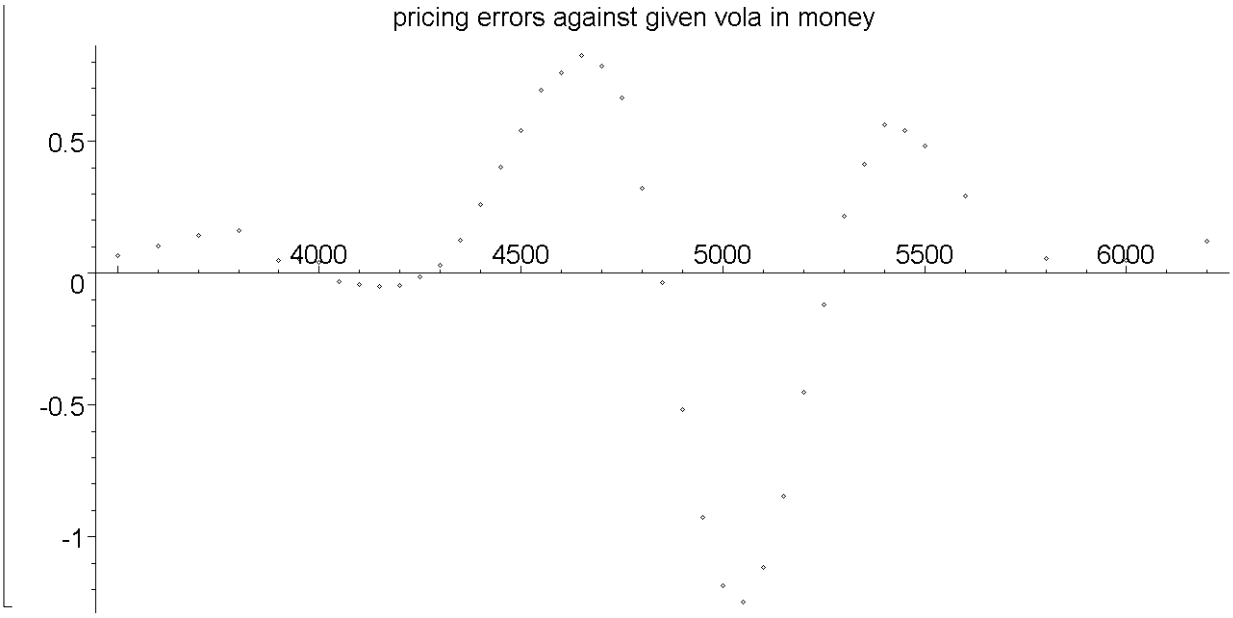
So even that almost matches the original price there is an error of 10 Cents.

```
[ >
```

Pricing errors

The pricing errors in Euro and Cents look as follows:

```
> pricingErrors:=[seq([arrData[i][1],
    BSCall(Spot,arrData[i][1],Time,Rates,arrData[i][4]/100) -
    Call_num(arrData[i][1])], i=1..nData)]:
  plots[pointplot](%, title="pricing errors against given vola in money");
```

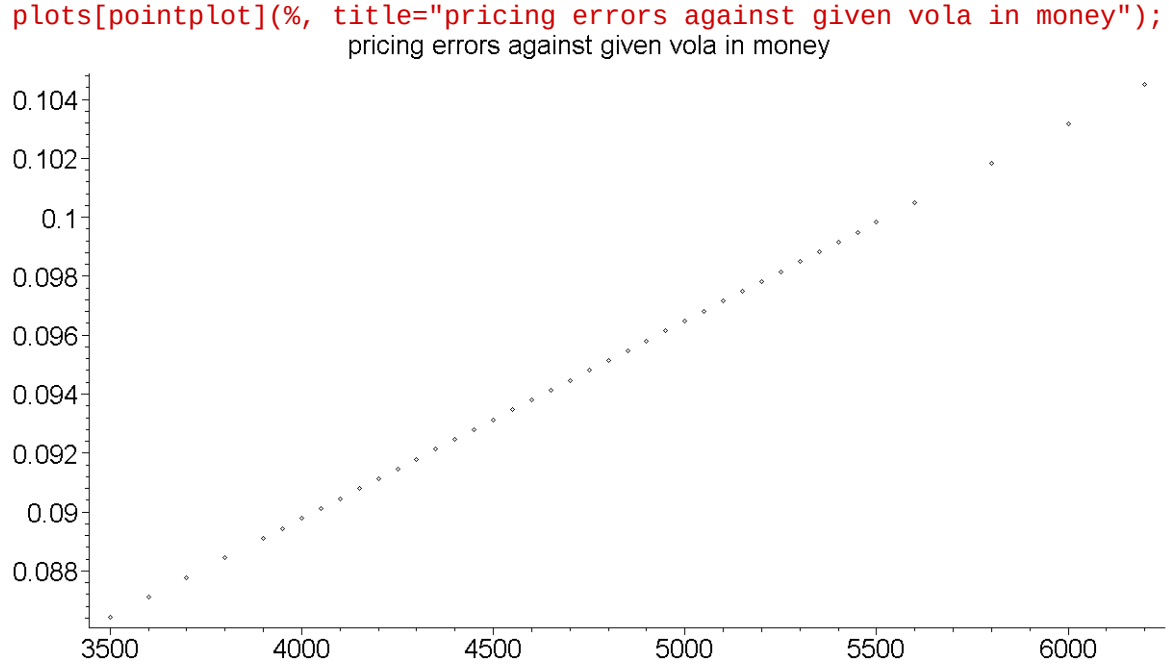


Not that bad for a brute approach (even if the relative error is large at the upside where prices are small).

Comparing that with the fitted vola one sees, that the largest error occure where the vola is not fitted well.

So compare again - but against the fitted vola function (to judge the PDF):

```
> arrK:=[seq(arrData[i][1], i=1..nData ) ]:
> pricingErrors:=evalf([seq(
    [arrK[i],  BSCall(Spot,arrK[i],Time,Rates,volaFct(arrK[i])) -
  Call_num(arrK[i])],
    i=1..nData)]):
  plots[pointplot](%, title="pricing errors against given vola in money");
```



It would be easy to control that error by (linear) regression:

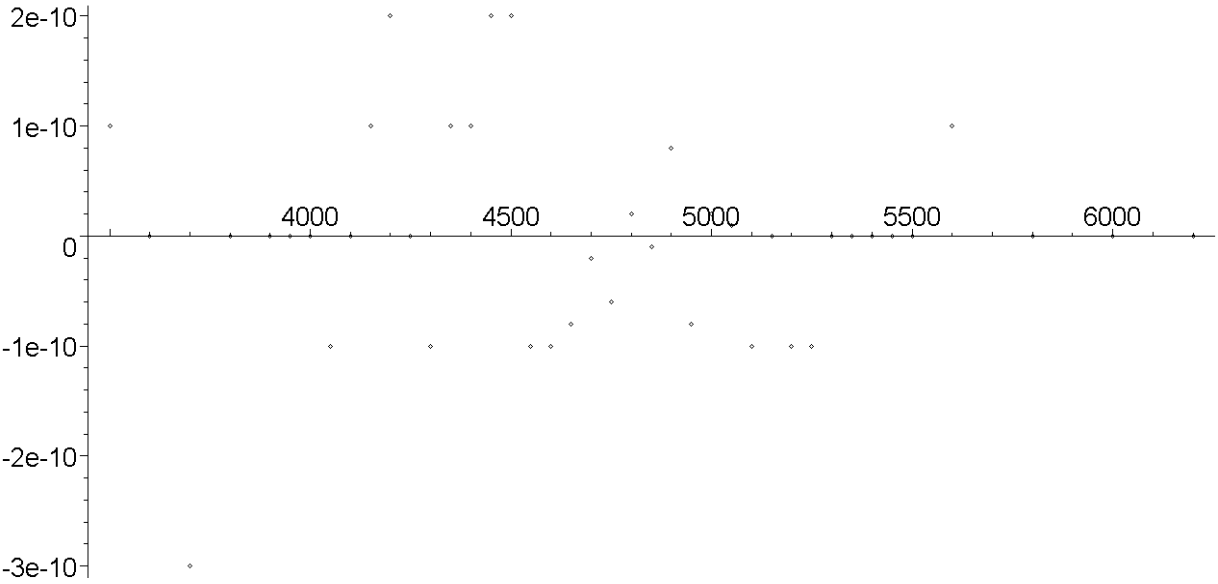
```
> tmpX:= map( x -> x[1], pricingErrors): tmpY:= map( x -> x[2],
```

```
  pricingErrors):
  Statistics[Fit]( error0 + error1*K, tmpX,tmpY, K):
  errorCorrection:=unapply(%,K);
  tmpX:='tmpX': tmpY:='tmpY':
```

$$\text{errorCorrection} := K \rightarrow 0.6297497490105147\,26 + 0.670127863717997842\ 10^{-5}\ K$$

With that correction one gets a result of 'numerical error = 0':

```
> pricingErrors0:=evalf([seq(
    [arrK[i],  BSCall(Spot,arrK[i],Time,Rates,volaFct(arrK[i])) -
  Call_num(arrK[i])
                - errorCorrection(arrK[i])],
    i=1..nData)]):
  plots[pointplot](%, title="pricing errors against given vola in money");
```



Since we work with 14 decimal places of exactness (through the integrator used) and have 3 - 4 leading digits one can not expect more exactness as shown in the graphic.

But be aware that the above error correction can not directly applied for other options: since the correction is linear in the strikes it will vanish differentiating twice and does not change the PDF.

Additionally there are numerical errors present, which may make theoretical identities numerically invalid (and need their own corrections).

```
>
```

```
>
```

Statistics for that PDF

Even if the approach theoretical is really questionable it allows to make some reasonable assertions about the statics for the PDF (thus derived from 'prices' via a vola fit).

Prepare the values for the tails first:

```
> 'PDF_M0_tail'=
  'Int(Q(mu-m_lower,v_lower), mu= -infinity..minMny) +
   Int(Q(mu-m_upper,v_upper), mu= maxMny .. infinity)';
  rhs(%): value(%)  assuming 0 < vDownside, 0 < vUpside: simplify(%):
```

```
collect(%,exp):
PDF_M0_tail:=%:
``;
'Int( fullPDF(m),m=-infinity..infinity)':
'%'=
  'Int(PDF(m), m= minMny..maxMny)' +'PDF_M0_tail';
'%%'= evalf(rhs(%));
```

$$\text{PDF\_M0\_tail} = \int_{-\infty}^{\text{minMny}} Q(\mu - \text{m\_lower}, \text{v\_lower})\, d\mu + \int_{\text{maxMny}}^{\infty} Q(\mu - \text{m\_upper}, \text{v\_upper})\, d\mu$$

$$\int_{-\infty}^{\infty} \text{fullPDF}(m)\, dm = \int_{\text{minMny}}^{\text{maxMny}} \text{PDF}(m)\, dm + \text{PDF\_M0\_tail}$$

$$\int_{-\infty}^{\infty} \text{fullPDF}(m)\, dm = 0.99999999999995$$

which says: it is a pdf (ok, some formal things are missing ...).

```
> 'PDF_M1_tail'=
  'Int(mu*Q(mu-m_lower,v_lower), mu= -infinity..minMny) +
   Int(mu*Q(mu-m_upper,v_upper), mu= maxMny .. infinity)';
  rhs(%): value(%)  assuming 0 < vDownside, 0 < vUpside: simplify(%):
  collect(%,exp):
  PDF_M1_tail:=%:
```

$$\text{PDF\_M1\_tail} = \int_{-\infty}^{\text{minMny}} \mu\, Q(\mu - \text{m\_lower}, \text{v\_lower})\, d\mu + \int_{\text{maxMny}}^{\infty} \mu\, Q(\mu - \text{m\_upper}, \text{v\_upper})\, d\mu$$

```
> 'Int( m * fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int(m^1*PDF(m), m= minMny..maxMny)' +
    'PDF_M1_tail';
  '%%'= evalf(rhs(%));

  PDF_Mean:=rhs(%);
```

$$\int_{-\infty}^{\infty} m\, \text{fullPDF}(m)\, dm = \int_{\text{minMny}}^{\text{maxMny}} m\, \text{PDF}(m)\, dm + \text{PDF\_M1\_tail}$$

$$\int_{-\infty}^{\infty} m\, \text{fullPDF}(m)\, dm = -0.0031280202660232$$

$$\text{PDF\_Mean} := -0.0031280202660232$$

So we have a mean clsoe to zero (astonishing enough) and for the higher sadistics:

```
> 'PDF_M2_tail'=
  'Int((mu-PDF_Mean)^2*Q(mu-m_lower,v_lower), mu= -infinity..minMny) +
   Int((mu-PDF_Mean)^2*Q(mu-m_upper,v_upper), mu= maxMny .. infinity)';
  rhs(%): value(%)  assuming 0 < vDownside, 0 < vUpside: simplify(%):
  collect(%,exp):
  PDF_M2_tail:=%:

  'PDF_M3_tail'=
  'Int((mu-PDF_Mean)^3*Q(mu-m_lower,v_lower), mu= -infinity..minMny) +
   Int((mu-PDF_Mean)^3*Q(mu-m_upper,v_upper), mu= maxMny .. infinity)';
  rhs(%): value(%)  assuming 0 < vDownside, 0 < vUpside: simplify(%):
  collect(%,exp):
  PDF_M3_tail:=%:
```

```
  'PDF_M4_tail'=
  'Int((mu-PDF_Mean)^4*Q(mu-m_lower,v_lower), mu= -infinity..minMny) +
   Int((mu-PDF_Mean)^4*Q(mu-m_upper,v_upper), mu= maxMny .. infinity)';
  rhs(%): value(%)  assuming 0 < vDownside, 0 < vUpside: simplify(%):
  collect(%,exp):
  PDF_M4_tail:=%:
```

$$\text{PDF\_M2\_tail} = \int_{-\infty}^{\text{minMny}} (\mu - \text{PDF\_Mean})^2\, Q(\mu - \text{m\_lower}, \text{v\_lower})\, d\mu + \int_{\text{maxMny}}^{\infty} (\mu - \text{PDF\_Mean})^2\, Q(\mu - \text{m\_upper}, \text{v\_upper})\, d\mu$$

$$\text{PDF\_M3\_tail} = \int_{-\infty}^{\text{minMny}} (\mu - \text{PDF\_Mean})^3\, Q(\mu - \text{m\_lower}, \text{v\_lower})\, d\mu + \int_{\text{maxMny}}^{\infty} (\mu - \text{PDF\_Mean})^3\, Q(\mu - \text{m\_upper}, \text{v\_upper})\, d\mu$$

$$\text{PDF\_M4\_tail} = \int_{-\infty}^{\text{minMny}} (\mu - \text{PDF\_Mean})^4\, Q(\mu - \text{m\_lower}, \text{v\_lower})\, d\mu + \int_{\text{maxMny}}^{\infty} (\mu - \text{PDF\_Mean})^4\, Q(\mu - \text{m\_upper}, \text{v\_upper})\, d\mu$$

```
> 'Int( (m-PDF_Mean)^2* fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int((m-PDF_Mean)^2*PDF(m), m= minMny..maxMny)' +
    'PDF_M2_tail';
  '%%'= evalf(rhs(%));

  PDF_Var:=rhs(%);
  PDF_Stdv:=sqrt(PDF_Var);
```

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^2\, \text{fullPDF}(m)\, dm = \int_{\text{minMny}}^{\text{maxMny}} (m - \text{PDF\_Mean})^2\, \text{PDF}(m)\, dm + \text{PDF\_M2\_tail}$$

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^2\, \text{fullPDF}(m)\, dm = 0.0067303643999051$$

$$\text{PDF\_Var} := 0.0067303643999051$$

$$\text{PDF\_Stdv} := 0.082038798137863$$

```
> 'Int( (m-PDF_Mean)^3* fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int((m-PDF_Mean)^3*PDF(m), m= minMny..maxMny)' +
    'PDF_M3_tail';
  '%%'= evalf(rhs(%));

  PDF_Skew:=rhs(%)/PDF_Stdv^3;
```

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^3\, \text{fullPDF}(m)\, dm = \int_{\text{minMny}}^{\text{maxMny}} (m - \text{PDF\_Mean})^3\, \text{PDF}(m)\, dm + \text{PDF\_M3\_tail}$$

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^3\, \text{fullPDF}(m)\, dm = -0.00060058937459551$$

$$\text{PDF\_Skew} := -1.0877266683138$$

```
> 'Int( (m-PDF_Mean)^4* fullPDF(m),m=-infinity..infinity)':
  '%'=
    'Int((m-PDF_Mean)^4*PDF(m), m= minMny..maxMny)' +
```

```
  'PDF_M4_tail';
'%%'= evalf(rhs(%));

PDF_Kurt:=rhs(%)/PDF_Stdv^4;
```

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^4 \, \text{fullPDF}(m) \, dm = \int_{\text{minMny}}^{\text{maxMny}} (m - \text{PDF\_Mean})^4 \, \text{PDF}(m) \, dm + \text{PDF\_M4\_tail}$$

$$\int_{-\infty}^{\infty} (m - \text{PDF\_Mean})^4 \, \text{fullPDF}(m) \, dm = 0.00026632848602339$$

$$\text{PDF\_Kurt} := 5.8795009225055$$

Compare it with the option prices: in the normed situation time = 1 vola is the standard deviation and a strike at forward (=1) in the used notation is volaFct_mny(0), which is reasonable close in size to the calculated standard deviation (there is absolutely no reason for them to be the same):

```
> 'volaFct_mny(0)': '%'=%;
  'PDF_Stdv': '%'=%;
```

$$\text{volaFct\_mny}(0) = 0.075643690303373$$

$$\text{PDF\_Stdv} = 0.082038798137863$$

One reason not to completely ignore that approach: one can use it as an initial guess to fit against sound models, if they allow to describe their parameters by their moments.

That brute approach also works for data not taken from option prices, but for time series of the stock.

```
> 
> 
```