SABR:
Patrick S. Hagan et al, Managing Smile Risk, Wilmott magazine 021118_smile.pdf
(siehe auch PatSmile.mnb)

approximation of implied volatility, (2.17), p 89

```
> restart;
  #assume(0<f): # forward
  #assume(0<K): # strike
  #assume(0<t): # remaining time
```

The Approximation (SABR volatility)

```
> z:= nu/alpha*(f*K)^((1-beta)/2)*ln(f/K);
  xi:=zeta -> ln((sqrt(1-2*rho*zeta+zeta^2)+zeta-rho)/(1-rho));
  S2:='z/xi(z)';
```

$$z := \frac{\nu\,(f\,K)^{\left(1/2-\frac{\beta}{2}\right)}\ln\left(\frac{f}{K}\right)}{\alpha}$$

$$\xi := \zeta \to \ln\left(\frac{\sqrt{1-2\,\rho\,\zeta+\zeta^2}+\zeta-\rho}{1-\rho}\right)$$

$$S2 := \frac{z}{\xi(z)}$$

```
> (f*K)^((1-beta)/2)*( 1 + (1-beta)^2*ln(f/K)^2/24 +
  (1-beta)^4*ln(f/K)^4/1920 ):
  S1:=alpha/%;
```

$$S1 := \frac{\alpha}{(f\,K)^{\left(1/2-\frac{\beta}{2}\right)}\left(1+\frac{1}{24}(1-\beta)^2\ln\left(\frac{f}{K}\right)^2+\frac{1}{1920}(1-\beta)^4\ln\left(\frac{f}{K}\right)^4\right)}$$

```
> (1-beta)^2*alpha^2/(f*K)^(1-beta)/24 +
  alpha*beta*rho*nu/(f*K)^((1-beta)/2)/4
  + (2-3*rho^2)*nu^2/24:;
  S3:= 1 +%*t*(+1);
```

$$S3 := 1+\left(\frac{(1-\beta)^2\,\alpha^2}{24\,(f\,K)^{(1-\beta)}}+\frac{\alpha\,\beta\,\rho\,\nu}{4\,(f\,K)^{\left(1/2-\frac{\beta}{2}\right)}}+\frac{(2-3\,\rho^2)\,\nu^2}{24}\right)t$$

```
> S1*S2*S3:
  SABR_vol:='unapply(S1*S2*S3, f,K,t,alpha,beta,rho,nu)';
```

$$SABR\_vol := unapply(S1\;S2\;S3, f, K, t, \alpha, \beta, \rho, \nu)$$

Code in C and Visual Basic

```
> 'SABR_vol(fwd,K,t,alpha,beta,rho,nu)';
  codegen[makeproc](%,[fwd,K,t,alpha,beta,rho,nu]):
  SABR:=codegen[optimize](%):
```

$$SABR\_vol(fwd, K, t, \alpha, \beta, \rho, \nu)$$

```
> CodeGeneration[VisualBasic](SABR,
  declare=[fwd::float,K::float,t::float,alpha::float,beta::float,rho::float,
  nu::float]);
Imports System.Math

Public Module CodeGenerationModule
  Public Function SABR( _
    ByVal fwd As Double, _
    ByVal K As Double, _
    ByVal t As Double, _
    ByVal alpha As Double, _
    ByVal beta As Double, _
```

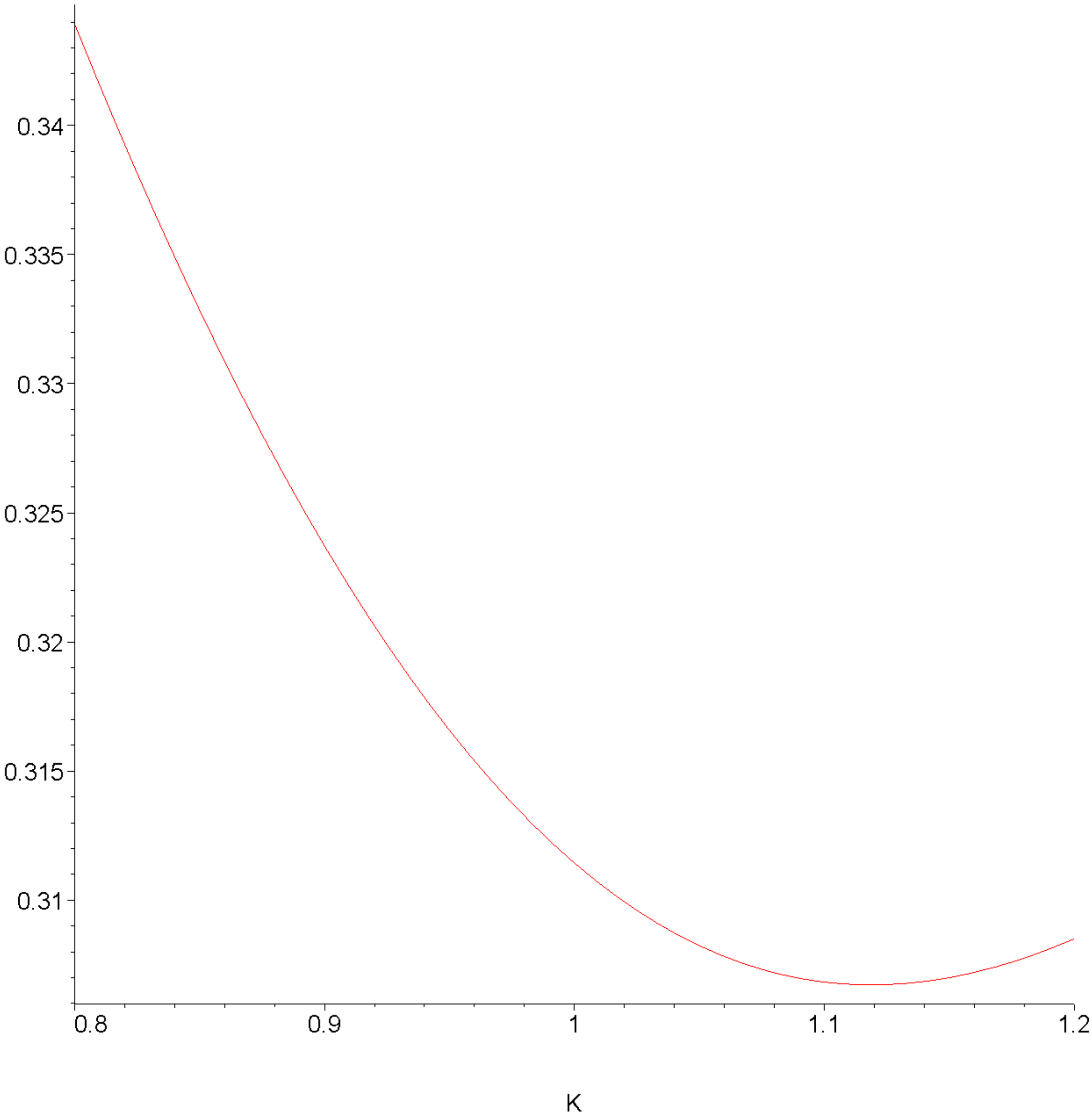
```
    ByVal rho As Double, _
    ByVal nu As Double) As Double
    Dim t1 As Double
    Dim t2 As Double
    Dim t5 As Double
    Dim t6 As Double
    Dim t9 As Double
    Dim t10 As Double
    Dim t16 As Double
    Dim t17 As Double
    Dim t18 As Double
    Dim t20 As Double
    Dim t25 As Double
    Dim t26 As Double
    Dim t29 As Double
    Dim t33 As Double
    Dim t41 As Double
    Dim t45 As Double
    Dim t54 As Double
    t1 = 0.1E1 - beta
    t2 = t1 * t1
    t5 = Log(fwd / K)
    t6 = t5 * t5
    t9 = t2 * t2
    t10 = t6 * t6
    t16 = rho * nu
    t17 = 0.1E1 / alpha
    t18 = K * fwd
    t20 = Pow(t18, t1 / 0.2E1)
    t25 = nu * nu
    t26 = alpha * alpha
    t29 = t20 * t20
    t33 = Sqrt(0.1E1 - 0.2E1 * t16 * t17 * t20 * t5 + t25 / t26 * t29 * t6)
    t41 = Log((t33 + nu * t17 * t20 * t5 - rho) / (0.1E1 - rho))
    t45 = Pow(t18, t1)
    t54 = rho * rho
    Return 0.1E1 / (0.1E1 + t2 * t6 / 0.24E2 + t9 * t10 / 0.1920E4) * nu * t
5 / t41 * (0.1E1 + (t2 * t26 / t45 / 0.24E2 + alpha * beta * t16 / t20 / 0.4
E1 + (0.2E1 - 0.3E1 * t54) * t25 / 0.24E2) * t)
  End Function
End Module
> CodeGeneration[C](SABR,
  declare=[fwd::float,K::float,t::float,alpha::float,beta::float,rho::float,
  nu::float]);
#include <math.h>

double SABR (
  double fwd,
  double K,
  double t,
  double alpha,
  double beta,
  double rho,
  double nu)
{
  double t1;
  double t2;
  double t5;
  double t6;
  double t9;
  double t10;
  double t16;
```

```
    double t17;
    double t18;
    double t20;
    double t25;
    double t26;
    double t29;
    double t33;
    double t41;
    double t45;
    double t54;
    t1 = 0.1e1 - beta;
    t2 = t1 * t1;
    t5 = log(fwd / K);
    t6 = t5 * t5;
    t9 = t2 * t2;
    t10 = t6 * t6;
    t16 = rho * nu;
    t17 = 0.1e1 / alpha;
    t18 = K * fwd;
    t20 = pow(t18, t1 / 0.2e1);
    t25 = nu * nu;
    t26 = alpha * alpha;
    t29 = t20 * t20;
    t33 = sqrt(0.1e1 - 0.2e1 * t16 * t17 * t20 * t5 + t25 / t26 * t29 * t6);
    t41 = log((t33 + nu * t17 * t20 * t5 - rho) / (0.1e1 - rho));
    t45 = pow(t18, t1);
    t54 = rho * rho;
    return(0.1e1 / (0.1e1 + t2 * t6 / 0.24e2 + t9 * t10 / 0.1920e4) * nu * t5
  / t41 * (0.1e1 + (t2 * t26 / t45 / 0.24e2 + alpha * beta * t16 / t20 / 0.4e1
  + (0.2e1 - 0.3e1 * t54) * t25 / 0.24e2) * t));
  }
```

Examples

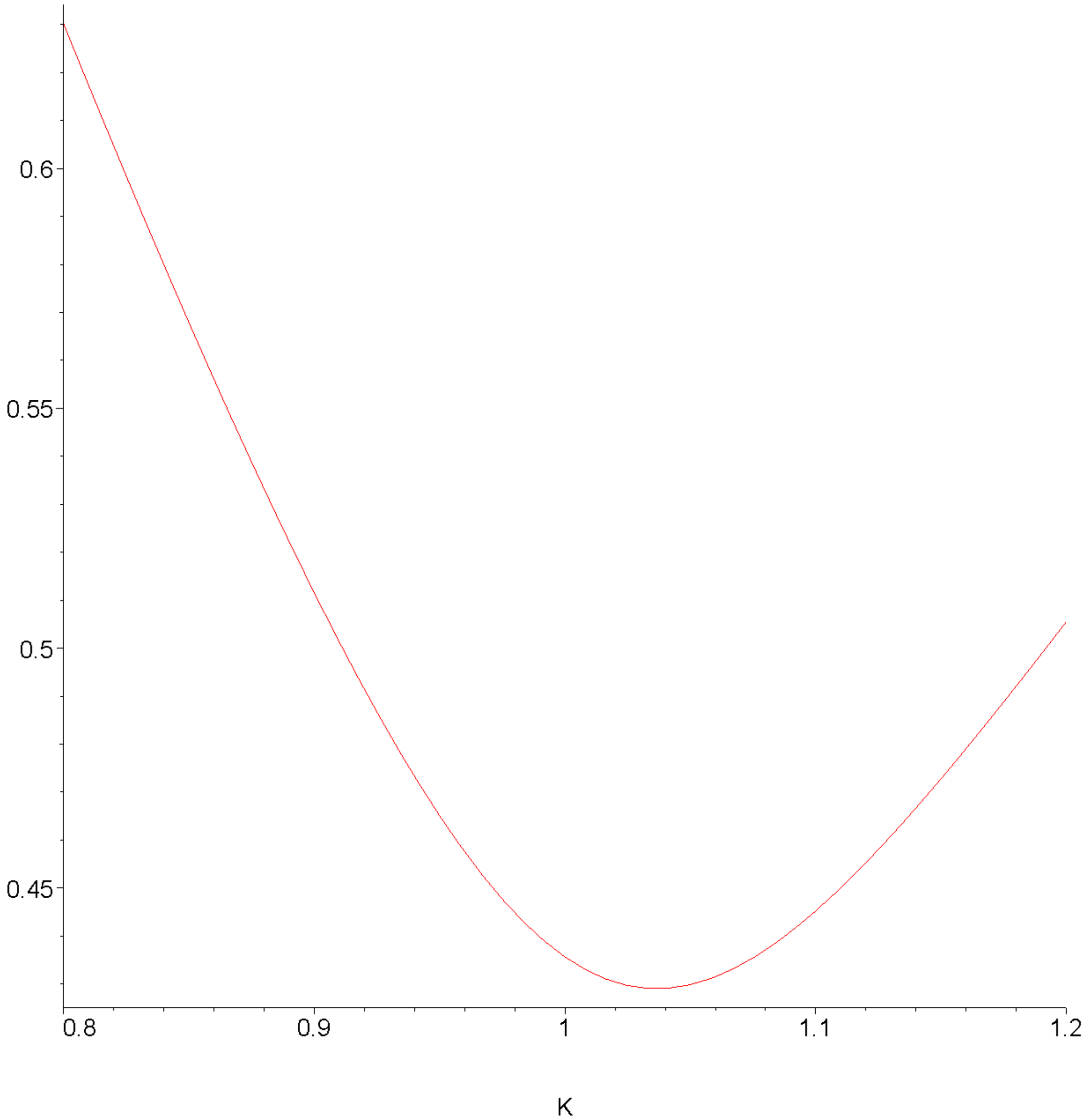
various data settings

```
> myData:=[f=1.0,K=K,t=1.0,alpha=0.30,beta=1.0,rho=-0.2,nu=0.8]:
  eval(SABR_vol(f,K,t,alpha,beta,rho,nu),myData): evalf(%):
  plot(%,K=0.8..1.2); myData; ``;
```



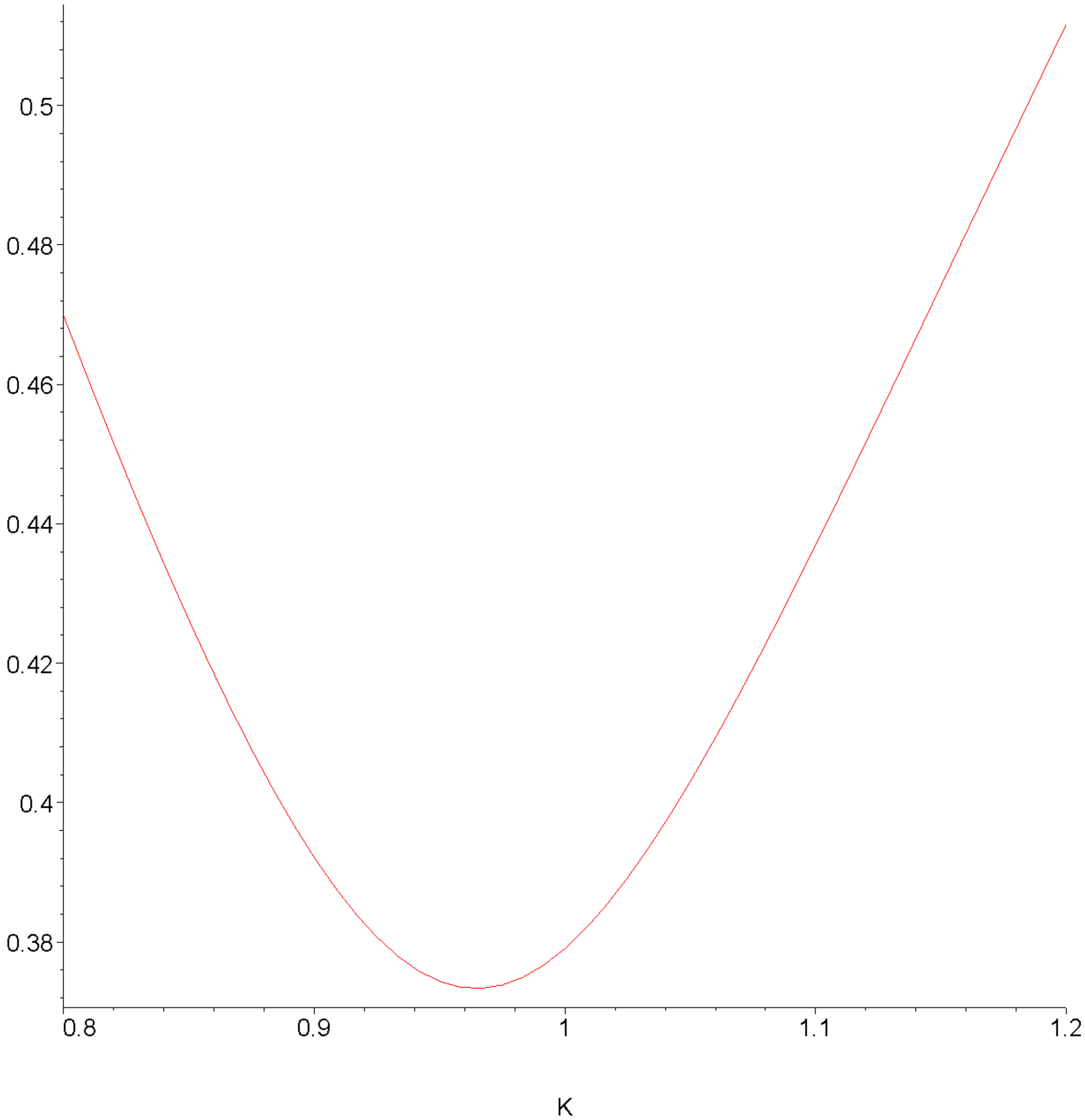
[f = 1.0, K = K, t = 1.0, α = 0.30, β = 1.0, ρ = -0.2, ν = 0.8]

```
> myData:=[f=1.0,K=K,t=1.0,alpha=0.30,beta=1.0,rho=-0.2,nu=2.5]:
  eval(SABR_vol(f,K,t,alpha,beta,rho,nu),myData): evalf(%):
  plot(%,K=0.8..1.2); myData; ``;
```



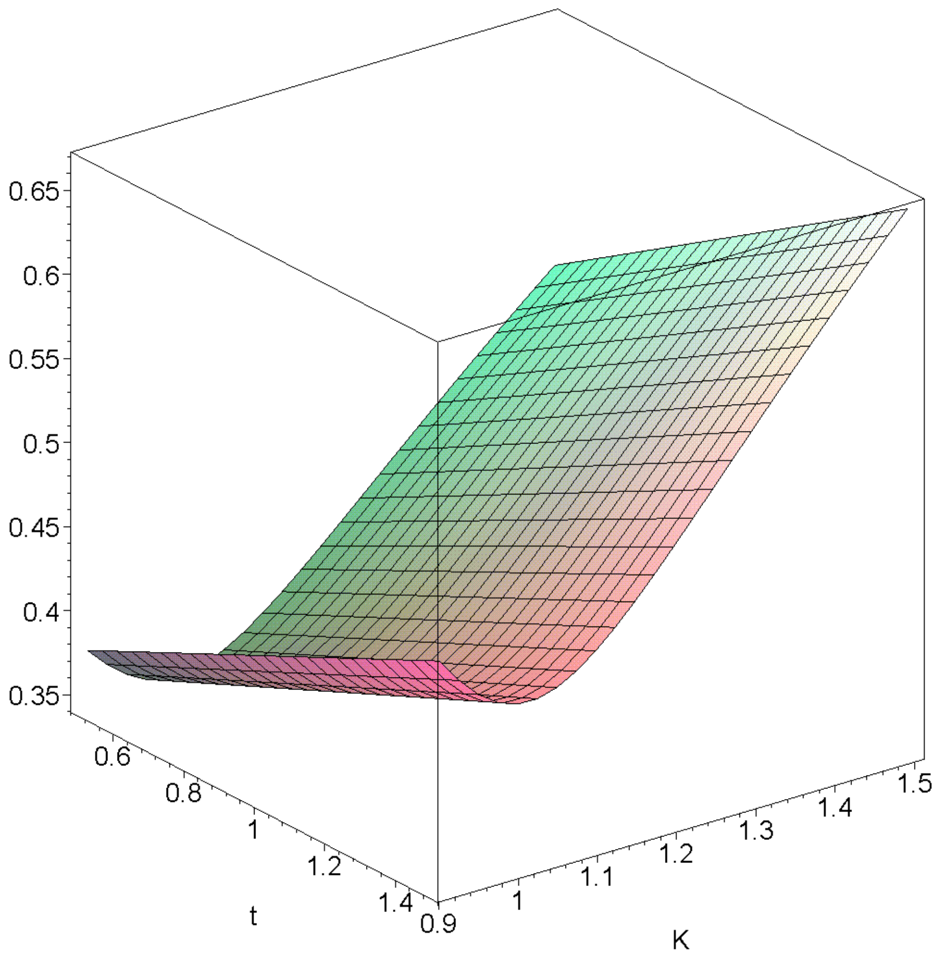
[f = 1.0, K = K, t = 1.0, α = 0.30, β = 1.0, ρ = -0.2, ν = 2.5]

```
> myData:=[f=1.0,K=K,t=0.5,alpha=0.30,beta=1.0,rho=-0.2,nu=-2.5]:;
  eval(SABR_vol(f,K,t,alpha,beta,rho,nu),myData): evalf(%):
  plot(%,K=0.8..1.2); myData; ``;
```



[f = 1.0, K = K, t = 0.5, α = 0.30, β = 1.0, ρ = -0.2, ν = -2.5]

```
> myData:=[f=1.0,K=K,t=t,alpha=0.30,beta=1.0,rho=-0.1,nu=2.0]:
  eval(SABR_vol(f,K,t,alpha,beta,rho,nu),myData): myExpr:=evalf(%):
  plot3d(%,t=0.5..1.5,K=.91...1.5,axes=boxed,orientation=[-37,67]); myData;
```



$$[f = 1.0, K = K, t = t, \alpha = 0.30, \beta = 1.0, \rho = -0.1, \nu = 2.0]$$

it seems that the minimum (at each fixed time) does not depend on K for the simple SABR model
but do not forget: f = forward is time dependend

```
> myData:=[f=exp(0.02*t)*1.0,K=K,t=t,alpha=0.30,beta=1.0,rho=-0.1,nu=2.0]:
  eval(SABR_vol(f,K,t,alpha,beta,rho,nu),myData): myExpr:=evalf(%):
  'myExpr'=myExpr; # from above
  diff(myExpr,K): eval(%,t=1.0): fsolve(%=0,K=1.01..1.5);
  diff(myExpr,K): eval(%,t=2.0): fsolve(%=0,K=1.01..1.5);
```

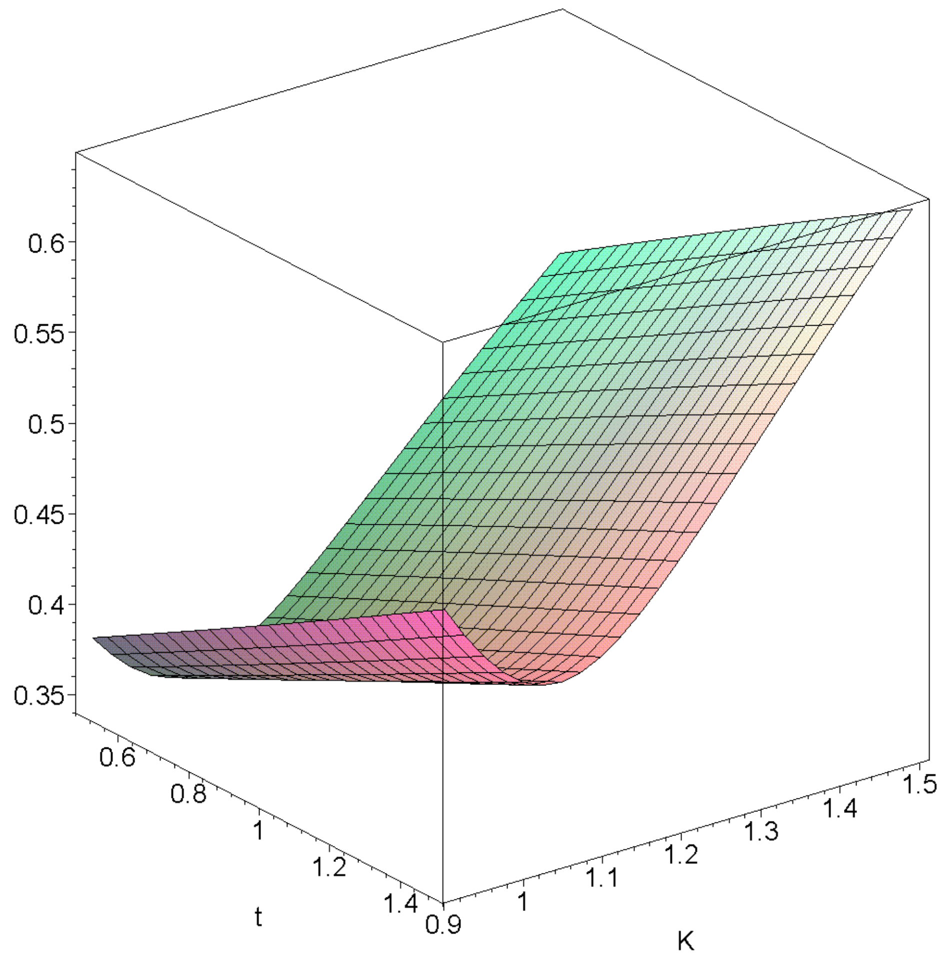
$$myExpr = 2.0 \ln\left(\frac{1.0\, \mathbf{e}^{(0.02\, t)}}{K}\right)(1. + 0.3133333333\, t) \Big/ \ln\Big(0.9090909091 \sqrt{1. + 1.333333333 \ln\left(\frac{1.0\, \mathbf{e}^{(0.02\, t)}}{K}\right) + 44.44444444 \ln\left(\frac{1.0\, \mathbf{e}^{(0.02\, t)}}{K}\right)^2} + 6.060606061 \ln\left(\frac{1.0\, \mathbf{e}^{(0.02\, t)}}{K}\right) + 0.09090909091 \Big)$$

$$1.043407203$$

$$1.064485427$$

anyway it is somewhat rigid doing it this way (--> dynamic SABR):

```
> plot3d(myExpr,t=0.5..1.5,K=.91...1.5,axes=boxed,orientation=[-37,67]);
  myData;
```



$$[f = 1.0\, \mathbf{e}^{(0.02\, t)}, K = K, t = t, \alpha = 0.30, \beta = 1.0, \rho = -0.1, \nu = 2.0]$$

Limits

```
> myData:=[f=1.0,K=K,t=1.0,alpha=0.30,beta=1.0,rho=-0.2,nu=2.5];
  eval(SABR_vol(f,f*exp(moneyness),t,alpha,beta,rho,nu),myData): evalf(%):
  limit(%,moneyness=-infinity);
```

$$myData := [f = 1.0, K = K, t = 1.0, \alpha = 0.30, \beta = 1.0, \rho = -0.2, \nu = 2.5]$$

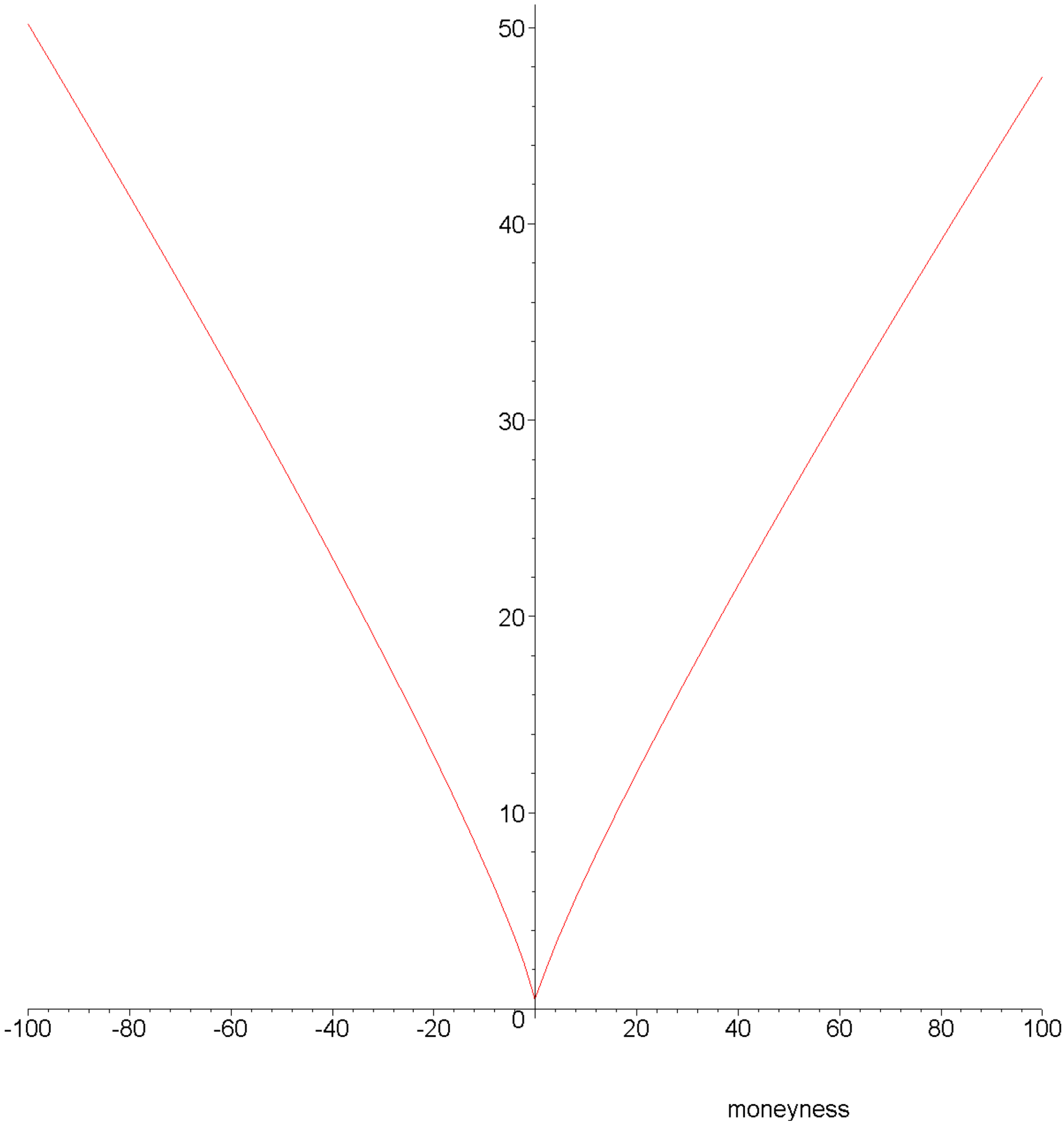
$$\text{Float}(\infty)$$

```
> myData:=[f=1.0,K=K,t=1.0,alpha=0.30,beta=1.0,rho=-0.2,nu=2.5];
  eval(SABR_vol(f,f*exp(moneyness),t,alpha,beta,rho,nu),myData): evalf(%):
  simplify(%,symbolic);
  #moneyness/ln(moneyness);
  plot(%,moneyness=-100..100);
  asympt(%%,moneyness,2);
```

$$myData := [f = 1.0, K = K, t = 1.0, \alpha = 0.30, \beta = 1.0, \rho = -0.2, \nu = 2.5]$$

$- 3.630208332$ moneyness / ln(0.00001666666667

$$\sqrt{0.2500000000\ 10^{10} - 0.8333333335\ 10^{10}\ \text{moneyness} + 0.1736111111\ 10^{12}\ \text{moneyness}^2} - 6.944444444\ \text{moneyness} + 0.1666666667\ )$$



$$-\frac{3.630208332\ \text{moneyness}}{-20.03011866 + 1.\ \ln(\text{moneyness})} + O\left(\frac{1}{\text{moneyness}}\right)$$

>

>