

Example for using the Fourier transform method for option pricing

Ref: Peter Carr and Dilip B. Madan, Option valuation using the fast Fourier transform

AVt, Aug 04.

```
> restart;
```

Set exactness to 14 digits, load libraries (for integral transformations and Black-Scholes pricing) and allow to use some predefined Maple variables:

```
> Digits:=14: with(inttrans): with(avtbslib):  
  unprotect(Psi);
```

Fourier transforms have various definitions, so check which one is used in

Maple - and what is meant in the papers ...

```
> fourier(f(x),x,z): %= convert(%,Int);  
  2*Pi*invfourier(f(x),x,z): %= convert(%,Int);
```

$$\text{fourier}(f(x), x, z) = \int_{-\infty}^{\infty} f(x) e^{(-ixz)} dx$$

$$2\pi \text{invfourier}(f(x), x, z) = \int_{-\infty}^{\infty} f(x) e^{(xzI)} dx$$

Hence for using Maple we need the *inverse* Fourier transform with scaling 2π .

Next we have the density f for BS. By homogeneity we can assume spot = 1

and f is given as normal density with

$$\text{mean} = \left(\text{rates} - \frac{\text{volatility}^2}{2} \right) \text{time} \text{ and}$$

$$\text{variance} = \text{volatility}^2 \text{ time} :$$

```
> '(exp((- (x-mean)^2)/(2*std_dev^2)))/(std_dev*  
sqrt(2*Pi))':  
'eval(%, [mean=(r-v^2/2)*t, std_dev=v*sqrt(t)]]':  
'%:= simplify(%,symbolic);  
f:=unapply(rhs(%),x);
```

$$\text{eval} \left(\frac{e^{\left(-\frac{(x - \text{mean})^2}{2 \text{std_dev}^2} \right)}}{\text{std_dev} \sqrt{2 \pi}}, \left[\text{mean} = \left(r - \frac{v^2}{2} \right) t, \text{std_dev} = v \sqrt{t} \right] \right) =$$

$$\frac{1}{2} \frac{e^{\left(-\frac{(2x - 2tr + tv^2)^2}{8tv^2} \right)} \sqrt{2}}{v \sqrt{t} \sqrt{\pi}}$$

$$f := x \rightarrow \frac{1}{2} \frac{e^{\left(-1/8 \frac{(2x - 2tr + tv^2)^2}{tv^2} \right)} \sqrt{2}}{v \sqrt{t} \sqrt{\pi}}$$

We need the characteristic function ϕ for f , which is given through the inverse Fourier transform (with scaling):

```
> 'f(x)'=f(x); rhs(%):
2*Pi*invsfourier(% ,x,xi) assuming (0<v, 0<t) ;
# write it in a nicer way ...
expand(%): combine(% ,exp):
algsbms(-v^2*t/2=tmp^2,%): completesquare(% ,tmp):
subs(tmp^2=-v^2*t/2,%):
phi:=unapply(% ,xi);
```

$$f(x) = \frac{1}{2} \frac{e^{\left(-\frac{(2x - 2tr + tv^2)^2}{8tv^2} \right)} \sqrt{2}}{v \sqrt{t} \sqrt{\pi}}$$

$$e^{\left(-\frac{t\xi(-2Ir + v^2I + \xi v^2)}{2} \right)}$$

$$\phi := \xi \rightarrow e^{(-1/2(\xi^2 + \xi I)tv^2 + t\xi rI)}$$

Now write down the characteristic function for a damned 'damped' Call with factor α :

```
> Psi:= u ->
exp(-r*t)*phi(u-(alpha+1)*I)/(alpha^2+alpha-u^2+I*
(2*alpha+1)*u);
```

$$\Psi := u \rightarrow \frac{e^{(-tr)} \phi(u - (\alpha + 1)I)}{\alpha^2 + \alpha - u^2 + (2\alpha + 1)uI}$$

Then a Call with strike K is given by undamping the Fourier transform:

```
> C:= K ->
exp(-alpha*ln(K))/Pi*Int(Re(exp(-I*u*ln(K))*Psi(u)
),u=0..infinity);
#indets(C(K),atomic): indets(%);
```

$$C := K \rightarrow \frac{e^{(-\alpha \ln(K))}}{\pi} \int_0^{\infty} \Re(e^{(-Iu \ln(K))} \Psi(u)) du$$

Let's look at some test values: volatility should be constant for any strike and time:

```
> tstData:=[S=1,t=1.0,r=0.02,v=0.2, alpha=0.75];
``;
K:=0.8;
eval(C(K),tstData): `price`=evalf(%); rhs(%):
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):
`vola`=evalf(%);
``;
K:=1.0;
eval(C(K),tstData): `price`=evalf(%); rhs(%):
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):
`vola`=evalf(%);
``;
K:=1.1;
eval(C(K),tstData): `price`=evalf(%); rhs(%):
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):
`vola`=evalf(%);
K:='K':
```

$\text{tstData} := [S = 1, t = 1.0, r = 0.02, v = 0.2, \alpha = 0.75]$

K := 0.8

price = 0.22542853157066

vola = 0.200000000000007

K := 1.0

price = 0.089160372785721

vola = 0.199999999999999

K := 1.1

price = 0.049438669572302

vola = 0.199999999999996

```
> tstData := [S=1, t=0.1, r=0.01, v=0.20, alpha=0.75];  
``;  
K:=0.8;  
eval(C(K),tstData): `price`=evalf(%); rhs(%):  
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):  
`vola`=evalf(%);  
``;  
K:=1.0;  
eval(C(K),tstData): `price`=evalf(%); rhs(%):  
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):  
`vola`=evalf(%);  
``;  
K:=1.1;  
eval(C(K),tstData): `price`=evalf(%); rhs(%):  
BSCallVolaNum(S,K,t,r,%):eval(%,tstData):  
`vola`=evalf(%);  
K:='K':
```

tstData := [S = 1, t = 0.1, r = 0.01, v = 0.20, α = 0.75]

K := 0.8

price = 0.20080237185902

vola = 0.20000000002967

K := 1.0

price = 0.025717414155455

vola = 0.199999999999998

K := 1.1

price = 0.0019817304457830

vola = 0.200000000000004

For K = 0.8 a 'larger' error is coming up and one may again look in the paper for better dampings for OTM options ...