

# Simulating (other) Lévy processes

---

Tony Ware  
'Lunch at the Lab' - 24th April, 2008

# Compound Poisson

```
function [JumpTimes,N] = PoissonJumpTimes(T,lambda,M);

if nargin<3, M=1; end
if nargin<4, algorithm = 2; end

JumpTimes = cell(M,1);
% Algorithm 6.2 from Cont&Tankov, P.174
NN = randraw('po',lambda*T,[1,M]);
ind = [0 cumsum(NN)];
U = rand(1,sum(NN));
for m = 1:M
    JumpTimes{m} = T*sort( U( (ind(m)+1):ind(m+1)) );
    N{m} = NN(m);
end

if M==1, JumpTimes=JumpTimes{1}; N = N{1}; end
```

# Compound Poisson

```
function CP = CompoundPoisson(T,lambda,M,JumpSize,varargin)
[JumpTimes,N] = PoissonJumpTimes(T,lambda,M);
if M>1
    y = cell(size(JumpTimes));
    for m = 1:M
        y{m} = cumsum(JumpSize([1,N{m}],varargin{:}));
    end
    for m=1:M
        CP{m}=struct('N',N{m},'JumpTimes',JumpTimes{m},...
            'Values',y{m},'T',T,'lambda',lambda);
    end
else
    y = cumsum(JumpSize([1,N],varargin{:}));
    CP=struct('N',N,'JumpTimes',JumpTimes,...
        'Values',y,'T',T,'lambda',lambda);
end
```

# Jump-diffusion

```
function JD = JumpDiffusion(CP,N,Diffusion,varargin);
% CP is a simulation of a compound Poisson processes
% N is the number of timesteps
tt = linspace(0,CP.T,N+1);
yJump = [0 CP.Values];
tJump = [0 CP.JumpTimes CP.T];
Values = zeros(1,CP.N+N+1);
t = Values;
yStart = 0;
pos = ones(1,CP.N+2);
% CP.N is the number of jumps
for n = 1:CP.N+1
    yStart = yStart+yJump(n);
    m = 1+[ceil(tJump(n)*N/CP.T) floor(tJump(n+1)*N/CP.T)];
    myt = [ tJump(n) tt(m(1):m(2)) tJump(n+1) ];
    y = Diffusion(yStart,myt,varargin{:});
    yStart = y(end);
    myN = length(myt);
    pos(n+1)=pos(n)+myN;
    ind = [pos(n):pos(n+1)-1];
    Values(ind)=y;
    t(ind)=myt;
end
JD=struct('Values',Values,'t',t,'pos',pos,'N',N,'CP',CP,'Diffusion',Diffusion);
```

# Variance Gamma

$$X_t = \sigma W_{G_t} + \theta G_t$$

```
function [x,g]=VarianceGammaSubordinated(t,sigma,theta,kappa,M)
% X_t = \sigma W_{G_t} + \theta G_t
% G_t is a gamma process with parameter \kappa

if nargin<5,M=1;end
dt = diff(t);
N = length(dt);
w = randn(M,N);
%assume dt is constant
a = dt(1)/kappa;
dg = kappa*randdraw('Gamma',a,[M,N]);
dx = sigma*w.*sqrt(dg)+theta*dg;
x = cumsum([repmat(0,M,1) dx],2);
g = cumsum([repmat(0,M,1) dg],2);
```

$$p(x) = \frac{x^{a-1}}{\Gamma(a)} e^{-x}$$

# Symmetric alpha-stable

$\Phi_X(z) = \exp(\sigma^\alpha |z|^\alpha)$  ← characteristic function

$$\Delta X_i = \Delta t_i^{\frac{1}{\alpha}} \frac{\sin \alpha u_i}{(\cos u_i)^{\frac{1}{\alpha}}} \left( \frac{\cos((1-\alpha)u_i)}{w_i} \right)^{\frac{1-\alpha}{\alpha}} \quad u_i \sim U \left( \left[ -\frac{\pi}{2}, \frac{\pi}{2} \right] \right)$$

```
function x=SymmetricAlphaStable(t,alpha,M)
```

```
if nargin<3,M=1;end
```

```
dt = diff(t);
```

```
N = length(dt);
```

```
u = (2*rand(M,N)-1)*pi/2;
```

```
w = randdraw('exp',1,[M,N]);
```

```
dx = log(repmat(dt,M,1))/alpha;
```

```
dx = dx - log(cos(u))/alpha;
```

```
dx = dx + ( log(cos((1-alpha)*u)) - log(w) )*(1-alpha)/alpha;
```

```
dx = real(dx);
```

```
dx = exp(dx).*sin(alpha*u);
```

```
x = cumsum([repmat(0,M,1) dx],2);
```