Jump Diffusions with Mathematica 10

Jump diffusions provide practical extension of the theory of random processes in various dimensions. They bring additional source of randomness and provide extra control for complex data handling that do not tend to follow particular parametric pattern. Therefore, jump diffusions have gained popularity in finance to model future behaviour and distributions of financial data. Why? The standard option theory in finance assumes that the logarithm of asset price is normally distributed. However, in practice, the observed distributions are not normal – they exhibit 'fatter tails', i.e. the probability of very large moves in either direction is larger than allowed by normal distribution. Therefore, the jump-diffusion method has been brought to provide plausible mechanism for explaining why fat tails exists and what are their consequences.

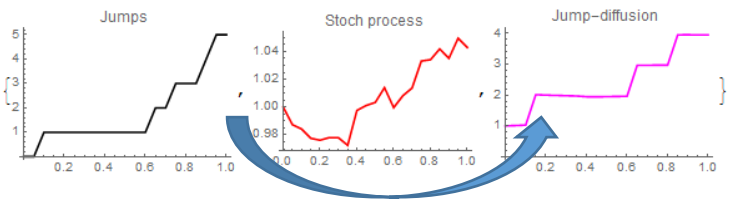
Jumps generally complicate the matter and the mathematics of diffusion process. We have two sources of randomness but only one tradable asset. A solution that still makes the model arbitrage-free is to assume that jumps are independent from the underlying and hence can be eliminated through diversification. We demonstrate the power of **Mathematica 10** for jump diffusion analysis and show how one can easily price options on assets, which follow this process.

A standard one-dimensional diffusion process in finance solves $dy = \mu(t)\,dt + \sigma(t)\,dw$. The jump-diffusion process solves (most time) the same differential equation, but solution occasionally jumps. The jumps can be bi-directional (positive and negative). To formalise the jump-diffusion process we therefore need to define the jump statistics.

Jumps are generally modelled with the *Poisson process* with rate λ which defines the jump frequency. Jumps are therefore independent of each other and the mean waiting time for the jump to occur is $1/\lambda$. The higher the rate, the more likely occurrence of a jump. When jump occurs, the underlying process increases by the jump size.

Consider the case – GBM process with $x0 = 1, \mu = 0.5\%, \sigma = 5\%$ and the Poisson process with $\lambda = 3$.

```
jProc = PoissonProcess[3];
dProc = GeometricBrownianMotionProcess[0.005, 0.05, 1];
jdProc = TransformedProcess[w[t] + j[t], {w ≈ dProc, j ≈ jProc}, t];
trun = {0, 1, 0.05};
jsim = RandomFunction[jProc, trun];
dsim = RandomFunction[dProc, trun];
jdsim = RandomFunction[jdProc, trun];
```

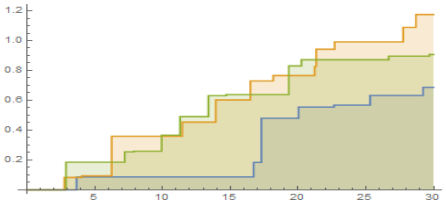


As the graphs indicate, jumps transform the original process (the middle graph) into very different path.

Although the Poisson process provides tractable representation of jumps, it has its limitation due to the fixed jump size = 1. Therefore, practitioners use its variation – the so-called *Compound Poisson process* that enables further definition of the jump statistics of with a specific jump-size distribution.

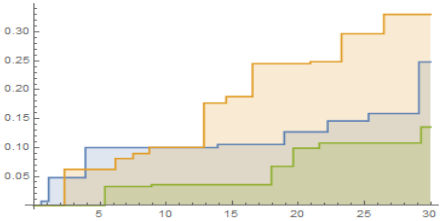
1) Exponential jumps:

```
jcProc1 = CompoundPoissonProcess[0.25, ExponentialDistribution[12]];
RandomFunction[jcProc1, {30}, 3];
ListLinePlot[%, InterpolationOrder → 0, Filling → Axis]
```



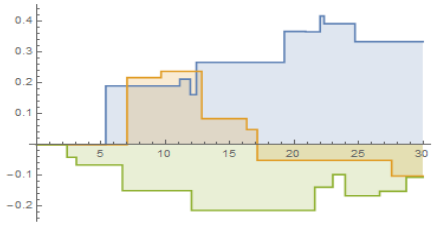
2) Gamma-distributed jumps:

```
jcProc2 = CompoundPoissonProcess[0.25, GammaDistribution[1.5, 0.02]];
RandomFunction[jcProc2, {30}, 3];
ListLinePlot[%, InterpolationOrder → 0, Filling → Axis]
```



Both *Exponential* and *Gamma* jumps are plausible choices for the magnitude control, however their jumps are unidirectional (positive). Since some financial markets exhibit jumps in both directions (such as credit), we can define a compound Poisson process with normally-distributed jump sizes in both directions:

```
jcProc3 = CompoundPoissonProcess[0.25, NormalDistribution[0.02, 0.09]];
RandomFunction[jcProc3, {30}, 3];
ListLinePlot[%, InterpolationOrder → 0, Filling → Axis]
```



Let's apply the jump theory to a real=case:

Consider the following case- *iBoxx USD High Yield Corporate Bond Index* with the following profile:

We want to price an option on the index and consider jump-diffusion model as alternative assumption for the process dynamics. Given the nature of the market (credit), we select the CIR process for this task and calibrate it to the historical data:

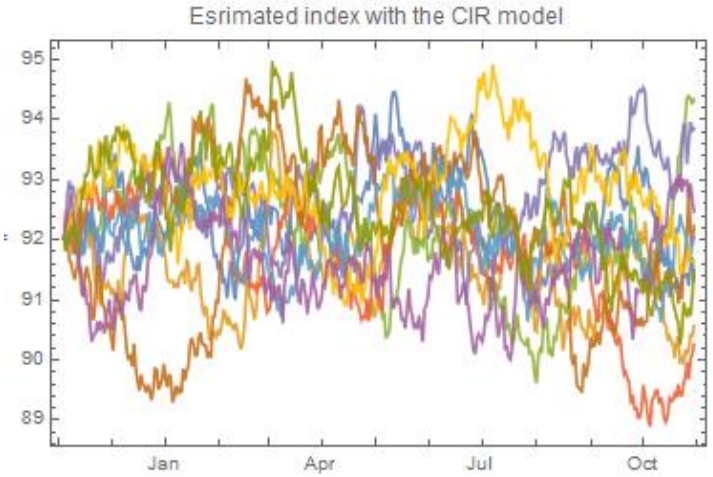
```
tswData = TimeSeriesWindow[iData, {{2014, 1, 1}, Automatic}];

estCIR = EstimatedProcess[tswData["Values"],
   CoxIngersollRossProcess[a, b, c, tswData[dLastDate]]]

CoxIngersollRossProcess[91.9462, 0.0254402, 0.0312705, 92.03]
```

The sample simulation will look as follows:

```
simCIR = RandomFunction[estCIR, {0, 360, 1}, 1000];
objCIR = TemporalData[simCIR["ValueList"], {dLastDate}];
DateListPlot[objCIR["Path", Range[10]], PlotStyle → Opacity[0.9],
 PlotLabel → "Esrimated index with the CIR model"]
```



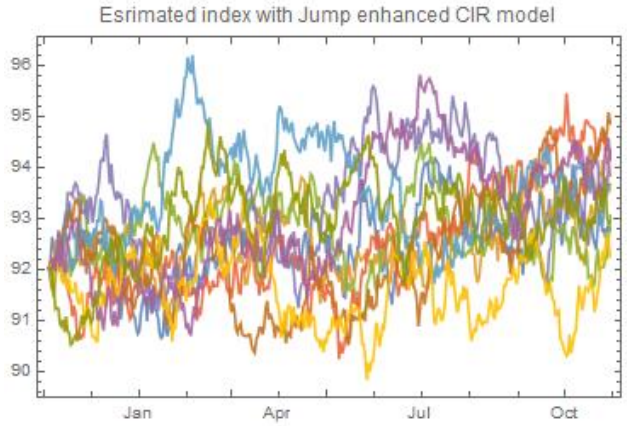
The process is well behaved and the CIR model fits the market data well.

We now want to add jumps to this process and explore an alternative specification with presence of extra randomness. We choose low-frequency **Compound Poisson Process** with normally distributed jumps with $\lambda = 5\%$ and jump size mean = 7.5% and the jump size volatility 9%. We use newly defined *TransformedProcess* function to define the jump diffusion.

```
estJCIR = TransformedProcess[x[t] + y[t],
   {x ≈ estCIR, y ≈ CompoundPoissonProcess[0.05, NormalDistribution[0.075, 0.1]]}, t];
```

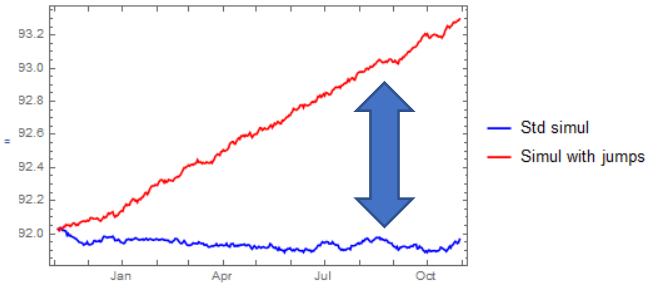
We can now visualise the simulation:

```
simJCIR = RandomFunction[estJCIR, {0, 360, 1}, 1000];
objJCIR = TemporalData[simJCIR["ValueList"], {dLastDate}];
DateListPlot[objJCIR["Path", Range[10]], PlotStyle → Opacity[0.9],
 PlotLabel → "Esrimated index with Jump enhanced CIR model"]
```



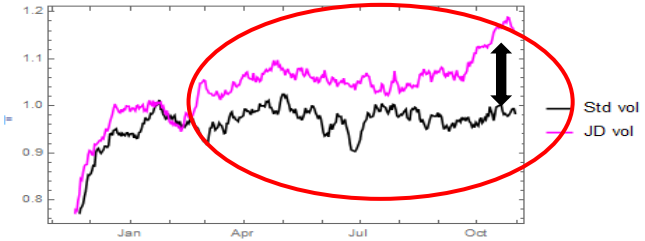
Although the graph does not look much different from the standard CIR simulation, we can compare the expected values for both processes to deduce the change:

```
DateListPlot[{meanCIR, meanJCIR}, PlotLegends → {"Std simul", "Simul with jumps"},
 PlotStyle → {Blue, Red}]
```



Means for both processes are now *quite different* and we can see that jumps produce very different distributional pattern.

We can further verify the jump impact through higher distributional moments such as volatility:

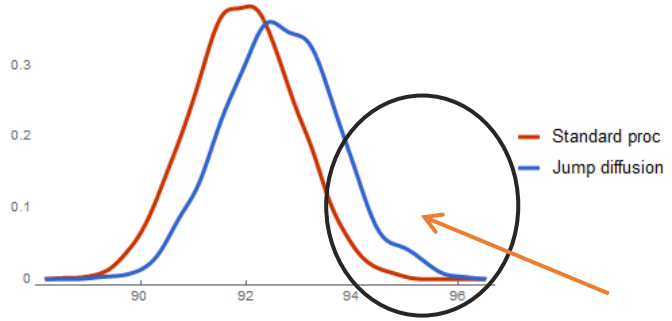


Jumps have clearly altered the HYG index distribution into the format with 'fatter tails'. We can investigate this further via histogram:

We fix the stationary distribution at 6M interval – 10th May 2015 and examine the patterns:

```
sd1 = objCIR["SliceData", "May 10, 2015"];
sd2 = objJCIR["SliceData", "May 10, 2015"];

SmoothHistogram[{sd1, sd2}, Automatic, "PDF", PlotTheme → "Web",
 PlotLegends → {"Standard proc", "Jump diffusion"}]
```



Jumps cause the distribution mass to move to the right and make the RHS tail fatter.

Option pricing: with generated paths, the option pricing is easy. 6M call option on the HYG index will produce very different prices under the CIR and Jump-CIR model:

Strikes	CIR model opt	CIR Jump model opt
93.	0.0686794	0.268486
93.25	0.0390809	0.187144
93.5	0.0215775	0.126729
93.75	0.0109663	0.0839942
94.	0.00537699	0.0539599
94.25	0.00238591	0.0349053
94.5	0.00087686	0.0220671
94.75	0.000068813	0.0126378
95.	0.	0.00617307
95.25	0.	0.00268216
95.5	0.	0.00105169
95.75	0.	0.000539478
96.	0.	0.0000707279

The option prices are quite different under the jump-diffusion model and the fatter tails make options more valuable. This pattern is particularly well observed in the OTM territory with higher strikes.