1 Assumptions of the Levy Functions Repository

We combine the work by Carr and Wu (2004) and the work of Duffie, Pan, and Singleton (2000) to create a very general framework for jump diffusions and stochastic volatility. Due to specification and calibration issues, we restrict our attention to jump-diffusions and do not consider processes with jumps of infinite activity.

1.1 Specification of stochastic volatility

Following Carr and Wu, we specify the stochastic time change rather than directly specifying the volatility. The time change is assumed to take the following form:

$$\tau = \int_0^t v_s ds$$

$$v_t = v_0 + \int_0^t a(1 - kv_s)ds + \int_0^t \eta\sqrt{v_s}dW_s^2 + \delta\sum_{j=1}^{N_t} z_j$$

Where N_t follows a Poisson process and the z_j are independent draws from an almost surely positive distribution. In this paper, we assume $z_j \sim \exp(q)$ and that it is independent of every other source of randomness. Following Carr and Wu, we set the parameters such that the long run expectation of v_t is 1. Since the the time change impacts the frequency of jumps linearly, the frequency of jumps can be modeled by λv_s. Hence to adjust the drift to make the long run expectation of v_t be 1, we adjust the drift as follows: $a\left(1 - \left(\frac{\delta\lambda\mathbb{E}[z_j]}{a} + 1\right)v_s\right)$ where for simplicity we let $k = \frac{\delta\lambda\mathbb{E}[z_j]}{a} + 1$.

1.2 Specification of the log asset price

The log asset price is assumed to follow the following form:

$$x_t = \log\left(\frac{S_t}{S_0}\right) = \left(\alpha - \frac{\sigma^2}{2}\right)t + \sigma W_t^1 + \sum_{j=1}^{N_t} y_j$$

Here $dW_t^1 dW_t^2 = \rho dt$ and N_t is the same jump process as the one in the stochastic time change dynamics. y_j is assumed to be Gaussian and independent of every other source of randomness.

1.3 Risk Neutral log asset price

Following Carr and Wu, the risk neutral log price can be modeled as follows (note that the market is incomplete):

$$x_t = \log\left(\frac{S_t}{S_0}\right) = rt - \left(\frac{\sigma^2}{2} + \lambda\left(e^{\mu_y + \frac{\sigma_y^2}{2}} - 1\right)\right)t + \sigma \tilde{W}_t^1 + \sum_{j=1}^{N_t} y_j$$

1.4 Analytical Characteristic Function

Following Carr and Wu, the full time changed x_τ has the following characteristic function:

$$\phi_x(u) = \hat{\mathbb{E}}\left[e^{uirt} e^{\tau\psi(u)}\right]$$

Where

$$\psi(u) = \lambda\left(e^{iu\mu_y - \frac{u^2\sigma_y^2}{2}} - 1\right) - \frac{\sigma^2}{2}u^2 - \left(\frac{\sigma^2}{2} + \lambda\left(e^{\mu_y + \frac{\sigma_y^2}{2}} - 1\right)\right)ui$$

Under $\hat{\mathbb{P}}$, v_s has the following dynamics:

$$v_t = v_0 + \int_0^t a\left(1 - \left(k - \frac{iu\rho\sigma\eta}{a}\right)v_s\right)ds + \int_0^t \eta\sqrt{v_s}d\hat{W}_s^2 + \delta\sum_{j=1}^{\hat{N}_t} z_j$$

Where $\hat{N}_t$ has jump frequency $v_s\lambda e^{iuy}$. By Duffie, Pan, and Singleton (2000), such a characteristic function has a semi-analytical solution.

1.5 ODE for Characteristic Function

1.5.1 General case

Consider the following functions:

$$\mu(x) = K_0 + K_1 x,\ \sigma^2(x) = H_0 + H_1 x,\ \lambda(x) = l_0 + l_1 x,\ R(x) = \rho_0 + \rho_1 x$$

By Duffie, Pan, and Singleton (2000), for processes X_t defined as

$$X_t = X_0 + \int_0^t \mu(X_s)ds + \int_0^t \sigma(X_s)dW_s + \sum_j^{N_t} Y_j$$

with jump frequency $\lambda(X_s)$, the following holds:

$$g(u, x, t, T) := \mathbb{E}\left[e^{-\int_t^T R(X_s)ds} e^{cX_T}\right]$$

has solution

$$e^{\alpha(t)+\beta(t)x}$$

where

$$\beta'(t) = \rho_1 - K_1\beta(t) - \frac{\beta^2(t)H_1}{2} - \int_\Omega l_1 \left(e^{\beta(t)z} - 1\right) \mu(dz)$$
$$\alpha'(t) = \rho_0 - K_0\beta(t) - \frac{\beta^2(t)H_0}{2} - \int_\Omega l_0 \left(e^{\beta(t)z} - 1\right) \mu(dz)$$

with $\beta(T) = c$, $\alpha(T) = 0$.

1.5.2 Application to the Analytical Characteristic Function

The process v_t under $\hat{\mathbb{P}}$ has this same structure with the following parameters:

$$K_0 = a,\ K_1 = -a\left(k - \frac{iu\rho\sigma\eta}{a}\right)$$
$$H_0 = 0,\ H_1 = \eta^2$$
$$l_0 = 0,\ l_1 = \lambda e^{iuy}$$
$$\rho_0 = 0,\ \rho_1 = -\psi(u)$$
$$c = 0$$

Substituting and simplifying yields the following ODEs:

$$\beta'(t) = -\psi(u) + (a + \delta\lambda - iu\rho\sigma\eta)\,\beta(t) - \frac{\beta^2(t)\eta^2}{2} - \int_\Omega \left(e^{\beta(t)\delta z} - 1\right) \lambda e^{iuy}\mu(dz, dy)$$
$$\alpha'(t) = -a\beta(t)$$

with $\beta(T) = 0$, $\alpha(T) = 0$.

1.5.3 Solution to the ODEs

2 Methodology for Option Pricing

The methodology for option pricing uses the Carr-Madan framework, the Fourier Space Time Step framework, and the Fang-Oosterlee framework. The code for these frameworks are in the FFTOptionPricing repo.

3 Methodology for Calibration

The methodology for calibration uses the Belomestny and Reiss framework. For more details and tests, see the FFTOptionPricing calibration documentation.

4 Simulation

To check that our option pricing methodology is implemented appropriately, we perform a Monte Carlo simulation:

```
> set.seed(42)
> r=.03
> sig=.2
> sigL=.1
> muL=-.05
> rho=-.5
> lambda=.5 #one jumps every two years on average
> a=.3
> eta=.2
> v0=.9
> s0=50
> k=50
> delta=.1
> n=100000 #number of options to simulate
> m=10000 #number of items per path
> t=1
> dt=t/(m)
> simulateExpJump=function(numJumps){
+   if(numJumps>0){
+     return(sum(delta*rexp(numJumps, 1.0))) #average of 1
+   }
+   else{
+     return(0)
+   }
+
+ }
> simulateGaussJump=function(numJumps){
+   if(numJumps>0){
+     return(sum(rnorm(numJumps, muL, sigL)))
+   }
+   else{
+     return(0)
+   }
+ }
> generatePricePath=function(m, type){
+   s=s0
+   v=v0
+  # sPath=c(s)
+   w2=rnorm(m)
+   w1=w2*rho+rnorm(m)*sqrt(1-rho*rho)
+   for(j in c(1:m)){
```

```
+      numJ=rpois(1, v*lambda*dt)
+      s=s*exp(r*dt-sig*sig*.5*v*dt-lambda*v*dt*(exp(muL+.5*sigL*sigL)-1)+sqrt(abs(v)*dt)*sig
+      v=v+a*(1-(delta*lambda/a+1)*v)*dt+eta*sqrt(abs(v)*dt)*w2[j]+simulateExpJump(numJ)
+     # sPath=c(sPath, s)
+    }
+    #plot(sPath, type='l')
+    if(type=='option'){
+      if(s>k){
+        return(s-k)
+      }
+      else{
+        return(0)
+      }
+    }
+    else{
+      return(s)
+    }
+
+ }
> optionPrices=sapply(c(1:n), function(index){
+   return(generatePricePath(m, 'option'))
+ })
> stockPrices=sapply(c(1:n), function(index){
+   return(generatePricePath(m, 'stock'))
+ })
> price=exp(-r*t)*mean(optionPrices)
> bounds=qnorm(.95)*sd(optionPrices)/sqrt(n-1)
> priceLow=price-bounds
> priceHigh=price+bounds
> mean(stockPrices)*exp(-r*t)# should be 50 if a martingale
>
```

This simulation creates bounds that are used to ensure that the numerical implementation of the characteristic function is accurate. For more details, see the integration tests inside the levy-functions repo.