**Lecture 5 (Exotics Markets) Assignment, MTH 9865**

Due start of class, October 14, 2015

**Question 1 (4 marks)**

In the stochastic spot/vol correlation model there are three correlations: the spot/vol correlation (which is stochastic); the spot/correlation correlation ; and the vol/correlation correlation .

In the formulation discussed in class, the latter two correlations are given as constant parameters. That 3x3 correlation matrix is not guaranteed to be positive definite, which is a problem for the model.

Show that, if the vol/correlation correlation is chosen as , the 3x3 correlation matrix is always positive definite.

The correlation matrix is a 3x3 matrix

We can calculate the determinant of the matrix as

If we then substitute , we get

Since and both are between -1 and +1, the determinant is always positive, and the correlation matrix is always positive definite.

**Question 2 (4 marks)**

For the Heston stochastic volatility model we showed that the characteristic function f needed to satisfy the following partial differential equation in two variables, x (the log of spot) and v (the instantaneous volatility squared):

and the initial condition for f(T=t) is

We guessed a form for f

Plug that form into the PDE above and show that it reduces the PDE to the two simultaneous ODEs

where and , the new time variable , and initial conditions are .

First we need to calculate the partial derivatives of f that we need to plug into the PDE:

In the last equation I defined , where is the time remaining to the expiration time T.

If we plug these into the PDE above we get

The first way to simplify this: every term is multiplied by f, so we can divide through by f to get rid of it (assuming f is non-zero, which is fine).

The next step: we can break this single equation into two simultaneous equations by recognizing that it needs to hold true for every value of v. That means we can group the terms with a “v” factor in front, and separately group the terms with no “v” factor, and set them both equal to zero independently. We can do this because the only place v comes into the equation is through the “v” factors; A and B are not functions of v, only of .

This gives us the answer we wanted:

**Question 3 (4 marks)**

Consider a Merton jump diffusion model:

where is the log of spot, is the log-drift (a constant), is the diffusive volatility of spot (a constant), and is a Poisson process with frequency . That is, in the infinitesimal time period , is equal to 1 with probability and equal to 0 otherwise. is the jump that happens if the Poisson process fires; it is normally distributed with mean and variance . Jumps are independent of each other, independent of the Poisson process, and independent of the Brownian motion .

We’ll look at finding the characteristic function of for some future time , because with that we can calculate vanilla option prices with a single integration.

We can apply Ito’s Lemma to f, including the possibility of jumps, and get

The last term is the one that comes from the jump , which of course is random (drawn from its normal distribution).

As with the Heston model, let’s try a form

The question: substitute that form into the expression for and solve for . You will need to use the fact that because the characteristic function is a martingale: but don’t forget that the jump term has a non-zero contribution to that expectation.

You will need to use one of the most important identities in finance: for a normally distributed variable , with mean and variance ,

Your final answer should be

If we start with the guessed form for f, we can calculate the derivatives we need:

Then when we plug them into the expression for df, along with the diffusive terms for dx, we get

For this, notice that we when substituted for “dx” in the SDE, we only included the diffusive terms, not the jump terms; the jump is included in the final term.

We know that the characteristic function f is a martingale, so the expected value of df has to be zero. If there were no jumps this fact would mean all the terms with a “dt” factor would sum to zero. Including the jumps, however, adds another term:

Since the jumps are independent of the Poisson process, we can split that final expectation into the product of two expectations. Also note that “x” here is not a random value: it is the (known) current value of x. The only random part is the size of the jump, J, which is normally distributed. That leads to

E[dq] is the expected value of the jump signal, which is 1 with probably and zero with probability . So .

For the term, we know that J is normally distributed, so we can use the identity above, and we get

Now we’re close to the solution. All the terms above are multiplied by f dt, so we can divide by that to get an ordinary differential equation for A:

All the terms on the right hand side of that equation are constant; and we know that A(0)=0 so that the characteristic function has the correct final condition. So we can trivially integrate this ODE to get the answer we wanted

**Question 4 (10 marks)**

Implement pricing for a jump diffusion model as in Question 3.

You should have two functions: one that prices a vanilla option using the characteristic function you derived in Question 3; and one that prices using conditional expectations, as defined below.

The first thing you need to do for either approach is figure out what drift to use to hit the forward to the expiration time T. For this, we need to calculate the value of a forward in the model; that is, the expected value of spot at time T.

For this we will use conditional expectations. That is, conditioned on exactly N jumps happening by time T, x is the sum of the normally-distributed Brownian motion plus the sum of the N normally-distributed jumps; so, conditional on N jumps, x is normally distributed, and we can calculate the conditional expected value of spot:

Complete this calculation. In calculating this expression, remember that the Brownian motion and all the N jumps are independent of each other.

For the next step, what we want is the unconditional expected value of , which we can identify with the forward (and then use to solve for the value of that makes the model match the forward). For this we can average over the number of jumps, weighting each by the probability of realizing that number of jumps. For a Poisson distribution, the probability of realizing N jumps by time T is given by

So to get the unconditional expected value of spot, equal to the forward, we can write

Calculate that quantity, which will determine the correct value of to use in your functions.

The second function should price the vanilla option using the same conditional expectation approach we just used to calculate the forward. That is, conditioned on N jumps, the distribution of spot is lognormal, and we can calculate the conditional expected value and the conditional variance of log(spot). That means, conditioned on N jumps you can calculate the conditional option price using the Black-Scholes formula. Then you can average over the number of jumps, as above, and get the unconditional option price.

You should implement the two functions and make sure that they give the same price for the same option. (Don’t forget to discount future cashflows!) For the numerical integration, use the scipy numerical integration package (eg scipy.integrate.quad).

Then, using either function, you should generate a plot of implied volatility for the following market: spot = 1, time to expiration = 0.5y, forward points = 0.03, risk neutral discount rate 5%, = 7%, = -4%, = (15%)2, = 3/year.

Discuss the impact the four model parameters (, , , and ) have on ATM vol, risk reversal, and butterfly.

First we need to calculate the forward under the model, so that we can figure out how to calibrate the drift parameter.

As above, we can calculate the expected value of spot conditioned on exactly N jumps happening by T as

is a normally-distributed variable with mean 0 and variance T. All the jumps are also normally distributed, with mean and variance . The jumps are all independent of each other, and all independent of . That lets us write

Again we can use the identity for expected value of the exponent of a normally-distributed variable to reduce this to

This is still the conditional expected value of S(T) – conditioned on there being exactly N jumps by T. We want the unconditional expected value of S(T), since that’s what we can match to the forward F(T). The unconditional expectation comes from averaging over the number of jumps:

We can substitute P(N) given in the question and the expression for the conditional expected value of S(T) to get

We can simplify this to

Then we can use a nice identity to get rid of the sum:

This delivers us the expression for the forward,

We can then invert that to get an expression for the drift we need in the model to hit the market forward price F(T):

Now we know what value of to use whenever we price options in this model.

The next step is to calculate the price of a vanilla option using conditional expectations. We’ll use the same approach as for the forward: recognizing that the conditional distribution of spot (conditioned on exactly N jumps by T) is lognormal. The mean is the conditional mean we just calculated; the conditional variance (of log(spot)) is the variance from the Brownian motion plus the N variances of the jumps, since all those stochastic pieces are independent. That is,

Whenever we have a lognormal asset price, we can use the Black vanilla price formula to price it. The “Black” model is the pricing model for options on forwards – ie when we know the forward price of the asset rather than the spot. It’s just the Black-Scholes formula, expect that for the spot argument you pass in the forward, and both interest rates equal the denominated discount rate.

In this case, the “forward” is the conditional expected value of spot.

For the implied volatility argument to the Black formula, we require that implied volatility squared multiplied by time to expiration equals the variance of log spot – in this case, the conditional variance above.

That lets us write the conditional option price as

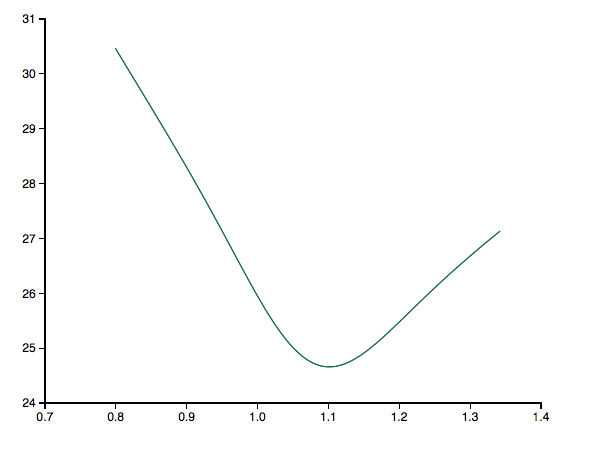
where is the Black-formula option price for forward F, implied volatility , and time to expiration T. (Of course it also takes arguments for the option type (call or put), the strike, and the denominated discount rate, but I’ve left those off for the sake of brevity.)

Then we can calculate the unconditional option price – the actual model price of the option – by summing over the jumps:

where P(N) is, as above, the probability of exactly N jumps by T.

There is no nice closed-form expression for this, so you need to do the sum numerically.

My implementation for this is included below. Generating implied volatilities under both models, I get



The qualitative impact of the model parameters:

* ATM volatility is determined by a combination of all three parameters. The lower bound is the parameter, and adding jump mean or jump variance increases it. The impact of the jumps is larger for larger tenors, so the ATM volatility curve is upward sloping, approaching for very short expirations.
* The risk reversal (skew) is determined mostly by the jump mean. A positive jump mean leads to positive risk reversal, and negative jump mean leads to negative risk reversal. In the example above the mean jump is negative, and the implied volatilities show a negative risk reversal.
* The butterfly (smile) is determined mostly by the jump variance: a larger jump variance leads to a larger butterfly.

The impact of the parameters on ATM vol, risk reversal, and butterfly is not quite orthogonal (ATM vol depends on all three), but because risk reversal is mostly a function of jump mean, and butterfly is mostly a function of jump variance, root-finding with this model is very efficient.

Python code:

import cmath

import math

import scipy.integrate

import scipy.optimize

import wst.core.analytics.bs as bs

def diffusive\_drift(spot, fwd, texp, vol, jump\_freq, jump\_mean, jump\_sd):

'''Returns the diffusive drift mu of the log(spot) SDE such that, once jumps are

included, the expected value of spot on time texp equals the forward fwd

spot: current spot of the asset

fwd: current forward price of the asset for settlement at texp

texp: time to settlement

vol: diffusive spot volatility

jump\_mean: mean jump size in log(spot)

jump\_sd: standard deviation of the jump size in log(spot)

'''

return 1 / texp \* math.log(fwd / spot) - jump\_freq \* (math.exp(jump\_mean + jump\_sd \* jump\_sd /2. ) - 1) - vol \* vol / 2.

def opt\_price\_merton\_charfn(is\_call, spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd):

'''Calculates an option price under a Merton jump diffusion model, integrating over

the characteristic function of ending log(spot).

is\_call: True, a call option; False, a put option

spot: current spot of the asset

strike: strike price of the option

texp: time to expiration of the option

vol: diffusive volatility of the Merton process

rd: denominated discount rate

rf: asset discount rate

jump\_freq: Poisson frequency of the jump process

jump\_mean: mean jump size in log(spot)

jump\_sd: standard deviation of the jump size in log(spot)

'''

# calculate the forward, then figure out the diffusive drift needed to hit that forward

fwd = spot \* math.exp((rd - rf) \* texp)

mu = diffusive\_drift(spot, fwd, texp, vol, jump\_freq, jump\_mean, jump\_sd)

# define the characteristic function of log(spot) at time texp - a function of the Fourier variable theta

i = 1j

jump\_var = jump\_sd \* jump\_sd

x = math.log(spot)

def char\_fn(theta):

A = i \* theta \* mu \* texp - theta \* theta \* vol \* vol \* texp / 2. \

+ jump\_freq \* texp \* (cmath.exp(i \* theta \* jump\_mean - theta \* theta \* jump\_var / 2.) - 1)

return cmath.exp(i \* theta \* x + A)

# define the integrand we'll integrate over to get a call option price

strike\_ratio = math.log(strike / spot)

def integrand(theta):

f = char\_fn(theta)

val = f \* cmath.exp(-i \* theta \* strike\_ratio) / (theta \* theta + i \* theta)

return val.real

# integrate over the integrand - for this we need to know what to use for "infinity" for theta. The real part of

# theta is something like exp(-theta^2 vol^2 T/2), where "vol" here means some representative volatility. So

# theta >> sqrt(2/vol^2/T) is something like an upper limit. Let's just set the scale of theta based on the

# diffusive volatility and keep stepping out until the contribution to the integral is zero.

theta\_scale = math.sqrt(2 / vol / vol / texp)

integ = 0

theta\_lo = 0

theta\_hi = theta\_scale

count = 0

while True:

integ\_piece = scipy.integrate.quad(integrand, theta\_lo, theta\_hi)[0]

integ += integ\_piece

if abs(integ\_piece) < 1e-10: break # the integral is unitless, something like option price/strike

# otherwise continue on to the next piece

theta\_lo = theta\_hi

theta\_hi += theta\_scale

# if we've cycled too many times, there's something wrong

count += 1

if count > 1000: raise ValueError('Numerical integration not converging')

# get the call option price from the formula we had from class (no discounting yet)

price = fwd - strike /2. - strike / math.pi \* integ

# if it's a put, flip through put/call parity

if not is\_call:

price -= fwd - strike

# apply discounting and we're done

price \*= math.exp(-rd \* texp)

return price

def opt\_price\_merton\_condexp(is\_call, spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd):

'''Calculates an option price under a Merton jump diffusion model, summing over conditional

expectations conditioned on number of jumps.

is\_call: True, a call option; False, a put option

spot: current spot of the asset

strike: strike price of the option

texp: time to expiration of the option

vol: diffusive volatility of the Merton process

rd: denominated discount rate

rf: asset discount rate

jump\_freq: Poisson frequency of the jump process

jump\_mean: mean jump size in log(spot)

jump\_sd: standard deviation of the jump size in log(spot)

'''

# calculate the forward, then figure out the diffusive drift needed to hit that forward

fwd = spot\*math.exp((rd - rf) \* texp)

mu = diffusive\_drift(spot, fwd, texp, vol, jump\_freq, jump\_mean, jump\_sd) + vol \* vol / 2.

# sum over number of jumps - need to approximate sum to infinity

price = 0

n\_jumps = 0

fact = 1

elt = math.exp(-jump\_freq \* texp)

jump\_var = jump\_sd \* jump\_sd

while True:

prob\_n\_jumps = math.pow(jump\_freq \* texp, n\_jumps) / fact \* elt

fwd\_cond = spot \* math.exp(mu \* texp + n\_jumps \* (jump\_mean + jump\_var / 2))

var\_cond = vol \* vol \* texp + n\_jumps \* jump\_var

vol\_cond = math.sqrt(var\_cond / texp)

price\_cond = bs.opt\_price(is\_call, fwd\_cond, strike, texp, vol\_cond, 0, 0)

price\_piece = price\_cond \* prob\_n\_jumps

price += price\_piece

if abs(price\_piece / strike) < 1e-14: break

fact \*= n\_jumps + 1 # keeping track of N!

n\_jumps += 1

# apply discounting and return

price \*= math.exp(-rd \* texp)

return price

def imp\_vol\_merton\_charfn(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd):

'''Implied volatility under the Merton jump diffusion model, pricing by integrating

over the characteristic function of ending log(spot).

spot: current spot of the asset

strike: strike price of the option

texp: time to expiration of the option

vol: diffusive volatility of the Merton process

rd: denominated discount rate

rf: asset discount rate

jump\_freq: Poisson frequency of the jump process

jump\_mean: mean jump size in log(spot)

jump\_sd: standard deviation of the jump size in log(spot)

'''

fwd = spot\*math.exp((rd - rf) \* texp)

is\_call = strike >= fwd

price = opt\_price\_merton\_charfn(is\_call, spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

vol = bs.imp\_vol(is\_call, spot, strike, texp, rd, rf, price, 3 \* vol)

return vol

def imp\_vol\_merton\_condexp(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd):

'''Implied volatility under the Merton jump diffusion model, pricing by summing over

conditionally-expected option prices conditioned on number of jumps.

spot: current spot of the asset

strike: strike price of the option

texp: time to expiration of the option

vol: diffusive volatility of the Merton process

rd: denominated discount rate

rf: asset discount rate

jump\_freq: Poisson frequency of the jump process

jump\_mean: mean jump size in log(spot)

jump\_sd: standard deviation of the jump size in log(spot)

'''

fwd = spot \* math.exp((rd - rf) \* texp)

is\_call = strike >= fwd

price = opt\_price\_merton\_condexp(is\_call, spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

vol = bs.imp\_vol(is\_call, spot, strike, texp, rd, rf, price, 3 \* vol)

return vol

def test():

import wst.util.plot as plot

spot = 1

pnts = 0.03

texp = 0.5

vol = 0.08

rd = 0.05

jump\_freq = 3

jump\_mean = -0.04

jump\_sd = 0.

# get the asset discount rate from the forward

fwd = spot + pnts

rf = rd - 1 / texp \* math.log(fwd / spot)

# solve for the limiting plot strikes on either side

delta\_limit = 0.10

def arg\_func(strike):

imp\_vol = imp\_vol\_merton\_charfn(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

return bs.opt\_delta(False, spot, strike, texp, imp\_vol, rd, rf) + delta\_limit

strike\_lo = scipy.optimize.newton(arg\_func, fwd \* math.exp(-vol \* math.sqrt(texp)))

def arg\_func(strike):

imp\_vol = imp\_vol\_merton\_charfn(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

return bs.opt\_delta(True, spot, strike, texp, imp\_vol, rd, rf) - delta\_limit

strike\_hi = scipy.optimize.newton(arg\_func, fwd \* math.exp(vol \* math.sqrt(texp)))

# plot out the vols

nstrikes = 100

dstrike = (strike\_hi - strike\_lo) / (nstrikes - 1)

strikes, vols\_charfn, vols\_condexp = [], [], []

for i in range(nstrikes):

strike = strike\_lo + i\*dstrike

strikes.append(strike)

vol\_charfn = imp\_vol\_merton\_charfn(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

vols\_charfn.append(vol\_charfn \* 100)

vol\_condexp = imp\_vol\_merton\_condexp(spot, strike, texp, vol, rd, rf, jump\_freq, jump\_mean, jump\_sd)

vols\_condexp.append(vol\_condexp \* 100)

plot.plotxy(strikes, [vols\_charfn, vols\_condexp])

if \_\_name\_\_=="\_\_main\_\_":

test()