**Lecture 3 (Options Markets) Assignment, MTH 9865**

Due start of class, September 30, 2015

**Question 1 (4 marks)**

Derive the expression for the ATM strike, given the ATM volatility and other market parameters, in the case where the market convention premium currency is the asset currency. In that case the price of the portfolio is

where is the price of the portfolio (which varies with spot S), is the price of the option (which varies with spot S), S is the spot, is the initial spot (at the time of the trade), and is the initial price of the option (equal to ). Note that the option here might be a call or might be a put.

Calculate the derivative of with respect to S to get the delta of the portfolio (taking at spot S=), and then solve for the strike that makes the delta of a call portfolio equal to the negative of the delta of a put portfolio (using Black-Scholes formulas for the call and put prices and deltas).

The delta of the portfolio of option minus premium is

and from here on we’ll assume S=S0, so will drop the “0” on v and S.

The premium v is just the Black-Scholes price, so we can plug in the formula for that, along with the Black-Scholes delta formula, to write down the deltas of a call and of a put:

The ATM strike is the one where the call delta equals the negative of the put delta, or where

Since we know that , we can rewrite to

which means that :

and we can then rewrite that as

So the ATM strike when the premium convention in the asset currency looks a lot like the one for premium in the denominated currency, but the opposite sign in front of the adjustment vs the forward.

**Question 2 (4 marks)**

Assume an FX market where the spot is 1, time to expiration is 0.5y, forward points are +0.0040, and the denominated discount rate is 1.75%. What is the strike corresponding to a 25-delta call option when its implied volatility is 8.75%, using market-convention delta? Assume the market convention premium currency is the denominated currency.

For this we’ll use the equation on slide 10 from the lecture 3 deck:

What are the arguments here? F, the forward, is just the spot plus the forward points, 1.0040. is the implied volatility for this strike, which we’re given as 8.75%. T is the time to expiration, 0.5. is the delta of the option, which we’re given as 0.25.

We need to calculate the asset interest rate q, however; for that we’ll use the definition of the forward in terms of the interest rates:

where r is the denominated currency interest rate (given as 1.75%) and q is the asset currency interest rate, which we want. We can rearrange that formula to solve for q=0.952%.

So then we just plug those numbers into the formula for the strike and get .

**Question 3 (2 marks)**

Describe the risk reversal “beta”.

The risk reversal beta is a measure of the covariance between moves in risk reversal and spot returns. A number like “0.1” means that if spot moves up 1%, the expected move in the risk reversal is 0.1 vols.

The risk reversal beta can be determined by running a historical regression of daily moves in risk reversal (in vols) against daily spot returns (in %).

Note that the risk reversal beta is a curve – you need to specify what tenor of risk reversal you are regressing against spot. Typically risk reversal betas are larger for shorter tenors.

**Question 4 (2 marks)**

Explain the two arbitrage conditions that should be avoided when interpolating in the strike direction, and the one (weak) arbitrage condition to avoid when interpolating in the time direction.

The two arbitrages in the strike direction to avoid are call price increasing with strike, and negative curvature of call price with respect to strike. That is,

both imply arbitrages, in the sense that you can construct a portfolio of vanilla options that you are net paid premium to enter into, but that at expiration either pays nothing or pays positive value.

The arbitrage in the time direction is weaker: it is hard to define an explicit portfolio that is a true arbitrage when there is skew and smile in the market (you can do it in a pure Black-Scholes world, however).

There you must guarantee that there is no “negative forward variance” – that the implied variance as a function of time to expiration T,

must be always increasing with T. Here, is the implied volatility to time T.

**Question 5 (10 marks)**

Implement a cubic spline interpolation for implied volatility vs strike which has non-standard boundary conditions to give more intuitive volatility extrapolation.

Assume you are given five implied volatilities, through , for five known strikes, through , for some known time to expiration .

Set up the boundary condition of the cubic spline such that implied volatility reaches a constant value a certain distance beyond the edge points. The distance is defined by a parameter, the **cubic spline extrapolation parameter, F**. On the left side, for strikes less than the minimum marked strike , implied volatility reaches a constant value at strike :

and on the right side, for strikes greater that the maximum marked strike , implied volatility reaches a constant value at strike :

Start with the standard method for defining a cubic spline (eg here: <http://www.aip.de/groups/soe/local/numres/bookcpdf/c3-3.pdf>), but add in two extra points at and where you know that the slope of vol vs strike goes to zero, and the second derivative of vol vs strike goes to zero, but where you do not know the value of volatility (that comes out of the solution for the cubic spline). Of course the first and second derivatives of vol vs strike should be continuous across and .

Implement a Python class that is initialized with the five strikes, five implied volatilities, time to expiration, and cubic spline extrapolation parameter. As part of initialization it will need to solve a linear system to generate the cubic spline parameters, for which you can use the functions in the scipy or numpy packages. The class should include a “volatility” method that takes a strike and returns an interpolated volatility.

Finally, generate plots of implied volatility vs strike for F=0.01 and F=10 for the following market: T=0.5y; ATM volatility of 8%, 25-delta risk reversal of 1%, 10-delta risk reversal of 1.8%, 25-delta butterfly of 0.25%, and 10-delta butterfly of 0.80%; spot=1; forward points=0; denominated discount rate=0%. For this you can use the plotxy function in the WST library wst/util/plot.py; you should return one chart with the two curves on it, showing implied volatility vs strike for all strikes between the 1-delta put and 1-delta call.

Give some intuition behind the shape of the two plots in terms of the value of F.

First we need to figure out how to represent and build the cubic spline.

We’ve got five input strikes and implied volatilities, and we’re going to add one point before the first strike () and one point after the last strike (). So that’s seven points, and six intervals.

In each interval i=0->5 we’ll define the volatility as a function Yi which is a function of strike x, like

That is, just a cubic form. Each piece has 4 parameters, and there are 6 intervals, so we have 4x6=24 parameters to solve for to fully define the spline.

First we can make sure the spline matches at the input point yi, that is

Those hold for i=1->5, that is, our five input points where we know the level of volatility. (Remember, we don’t know the level of volatility for the two points we added at the edges – that’ll come out of the fit.) That gives us 5 equations for our 24 parameters.

Next we can make sure that the spline’s value, slope, and second derivative match at the end of each interval i=0->4, which gives us another 15 equations:

So now we’re up to 20 equations for our 24 parameters; we need four more. The final set of constraints comes from how we’ve defined our extrapolation: we want to slope and second derivative to go to zero at and :

And that’s 24 linear equations for 24 parameters. We can bung those all into a linear system solver and get the parameters; then once we have the parameters we can calculate the implied volatility for any strike.

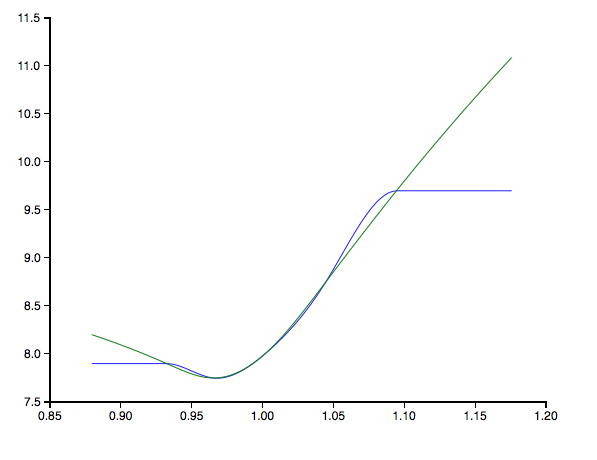
We also need to know how to get the vols and strikes we need for inputs.

First, let’s get the vols from the ATM, risk reversal, and butterfly values we’re given. We can invert the definition of risk reversal and butterfly to show that, for the 25d vols,

where is the 25-delta call volatility and is the 25-delta put volatility. We can get the 10-delta call and put volatilities with the same equations, just plugging in the 10-delta risk reversal and butterfly instead.

For strikes, we’ll use the ATM strike formula to get the ATM strike, and the strike-from-delta formula for the OTM strikes.

We can then chart the implied volatility vs strike for F=0.01 and F=10:



The blue line shows the interpolated volatility for F=0.01 and the green line corresponds to F=10.

Note that the vols for F=0.01 flatten out very quickly outside the marked points – that’s what we expect, since F represents the number of standard deviations beyond the marked points that vols flatten out. With F=10 the vol goes roughly linear, since the points where vol flattens are 10 standard deviations away from the marked points – far away indeed.

Python code:

import bisect

import math

import scipy

def \_fitted\_spline(strikes, vols, texp, extrap\_fact):

'''Function that takes the input strikes, vols, time to expiration, and

extrapolation factor and constructs the spline parameters'''

strike\_min = strikes[0] \* math.exp(-extrap\_fact \* vols[0] \* math.sqrt(texp))

strike\_max = strikes[-1] \* math.exp( extrap\_fact \* vols[-1] \* math.sqrt(texp))

all\_strikes = [strike\_min] + strikes + [strike\_max]

# there are six intervals for the spline where the function takes a cubic

# form, like

#

# y\_i(x) = A[i] + B[i] x + C[i] x^2 + D[i] x^3

#

# for i=0 through 5. x[0] == strike\_min, and x[6] == strike\_max.

#

# The goal is to solve for those 24 parameters. The constraints: first,

# that the volatilities match at the input points, for i=1->5:

#

# y[i] = A[i] + B[i] x[i] + C[i] x[i]^2 + D[i] x[i]^3

#

# That gives us 5 equations of the 24 we need. Next, we know that the

# value, slope, and second derivative must match at the end of each interval

# for i=0 to 4, like

#

# A[i] + B[i] x[i+1] + C[i] x[i+1]^2 + D[i] x[i+1]^3 = A[i+1]+ B[i+1] x[i+1] + C[i+1] x[i+1]^2 + D[i+1] x[i+1]^3

# B[i] + 2 C[i] x[i+1] + 3 D[i] x[i+1]^2 = B[i+1] + 2 C[i+1] x[i+1] + 3 D[i+1] x[i+1]^2

# 2 C[i] + 6 D[i] x[i+1] = 2 C[i+1] + 6 D[i+1] x[i+1]

#

# That gives us an addition 3\*5 = 15 equations, taking us to a total of 20

# equations so far for the 24 parameters.

#

# The final set of equations is where our extrapolation comes in. We require

# that the slope and second derivative go to zero at x[0] and x[6], which are the

# points we added beyond the marked points, based on the extrapolation factor parameter:

#

# B[0] + 2 C[0] x[0] + 3 D[0] x[0]^2 = 0

# 2 C[0] + 6 D[0] x[0] = 0

# B[5] + 2 C[5] x[6] + 3 D[5] x[6]^2 = 0

# 2 C[5] + 6 D[5] x[6] = 0

#

# And that gives us an additional four equations, bring us to 24 equations for our

# 24 parameters. So we can construct a linear system representing those equations

# and invert it to solve for the parameter values.

a = scipy.matrix(scipy.zeros((24, 24)))

b = scipy.matrix(scipy.zeros((24, 1)))

xs = all\_strikes

x2s = [x \* x for x in xs]

x3s = [x \* x \* x for x in xs]

# first five rows correspond to the five equations relating function values to the input vols

for i in range(5):

a[i, 4 \* (i + 1)] = 1

a[i, 4 \* (i + 1) + 1] = xs[i + 1]

a[i, 4 \* (i + 1) + 2] = x2s[i + 1]

a[i, 4 \* (i + 1) + 3] = x3s[i + 1]

b[i] = vols[i]

# next require the value to match at the end of each interval for interval=0->4

for i in range(5):

a[i + 5, 4 \* i] = 1

a[i + 5, 4 \* i + 1] = xs[i + 1]

a[i + 5, 4 \* i + 2] = x2s[i + 1]

a[i + 5, 4 \* i + 3] = x3s[i + 1]

a[i + 5, 4 \* (i + 1)] = -1

a[i + 5, 4 \* (i + 1) + 1] = -xs[i + 1]

a[i + 5, 4 \* (i + 1) + 2] = -x2s[i + 1]

a[i + 5, 4 \* (i + 1) + 3] = -x3s[i + 1]

b[i + 5] = 0

# next require the slopes to match

for i in range(5):

a[i + 10, 4 \* i + 1] = 1

a[i + 10, 4 \* i + 2] = 2 \* xs[i + 1]

a[i + 10, 4 \* i + 3] = 3 \* x2s[i + 1]

a[i + 10, 4 \* (i + 1) + 1] = -1

a[i + 10, 4 \* (i + 1) + 2] = -2 \* xs[i + 1]

a[i + 10, 4 \* (i + 1) + 3] = -3 \* x2s[i + 1]

b[i + 10] = 0

# next the 2nd derivs

for i in range(5):

a[i + 15, 4 \* i + 2] = 2

a[i + 15, 4 \* i + 3] = 6 \* xs[i + 1]

a[i + 15, 4 \* (i + 1) + 2] = -2

a[i + 15, 4 \* (i + 1) + 3] = -6 \* xs[i + 1]

b[i + 15] = 0

# then the final four equations forcing 1st and 2nd derivs to go

# to zero and the edge points we added in

a[20, 1] = 1

a[20, 2] = 2 \* xs[0]

a[20, 3] = 3 \* x2s[0]

b[20] = 0

a[21, 2] = 2

a[21, 3] = 6 \* xs[0]

b[21] = 0

a[22, 21] = 1

a[22, 22] = 2 \* xs[6]

a[22, 23] = 3 \* x2s[6]

b[22] = 0

a[23, 22] = 2

a[23, 23] = 6 \* xs[6]

b[23] = 0

# then solve the equation

sol = a.I \* b

cs\_params = [sol[i, 0] for i in range(24)]

return all\_strikes, cs\_params

class vol\_spline:

'''Represents a cubic spline fit to five implied volatilities/strikes, with

boundary conditions set such that vols flatten out a certain number of standard

deviations away from the outside strikes on either side'''

def \_\_init\_\_(self, strikes, vols, texp, extrap\_fact):

'''Initializes the spline and calculates the spline parameters internally.

strikes: list of five strikes (must be monotonically increasing)

vols: implied volatilities for the strikes

texp: time to expiration

extrap\_fact: cubic spline extrapolation factor, defining number of standard

deviations after the outside strikes that vols turn flat

'''

# validate that there are exactly five strikes and five vols, and that strikes

# are increasing

if len(strikes) != 5: raise ValueError('There should be 5 strike values')

if len(vols) != 5: raise ValueError('There should be 5 volatility values')

if strikes != sorted(set(strikes)): raise ValueError('Strikes should be monotonically increasing')

# remember the inputs

self.strikes = strikes

self.vols = vols

self.texp = texp

self.extrap\_fact = extrap\_fact

# calculate the spline parameters

self.all\_strikes, self.cs\_params = \_fitted\_spline(strikes, vols, texp, extrap\_fact)

def volatility(self, strike):

'''Interpolates a volatility for the given strike'''

# if it's asking for a vol for a strike outside the region

# where vols are flat, use the edges

if strike < self.all\_strikes[0]:

strike = self.all\_strikes[0]

if strike > self.all\_strikes[-1]:

strike = self.all\_strikes[-1]

# interpolate a vol from the spline

ind = bisect.bisect\_left(self.strikes, strike)

a = self.cs\_params[4 \* ind]

b = self.cs\_params[4 \* ind + 1]

c = self.cs\_params[4 \* ind + 2]

d = self.cs\_params[4 \* ind + 3]

return a + b \* strike + c \* strike \* strike + d \* strike \* strike \* strike

def test():

'''Test it out with some strikes & vols as per the assignment question'''

import scipy.stats

# note the market inputs from the question

spot = 1

atm = 0.08

rr25 = 0.01

rr10 = 0.018

bf25 = 0.0025

bf10 = 0.0080

texp = 0.5

# turn the RR and BF values into vols for the OTM strikes

vol25c = atm + rr25 / 2. + bf25

vol25p = atm - rr25 / 2. + bf25

vol10c = atm + rr10 / 2. + bf10

vol10p = atm - rr10 / 2. + bf10

# figure out the strikes (note that forward==spot since rates are zero)

atm\_strike = spot \* math.exp(atm \* atm \* texp /2.)

strike25c = spot \* math.exp(vol25c \* vol25c \* texp / 2. - vol25c \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.25))

strike25p = spot \* math.exp(vol25p \* vol25p \* texp / 2. + vol25p \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.25))

strike10c = spot \* math.exp(vol10c \* vol10c \* texp / 2. - vol10c \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.10))

strike10p = spot \* math.exp(vol10p \* vol10p \* texp / 2. + vol10p \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.10))

strikes = [strike10p, strike25p, atm\_strike, strike25c, strike10c]

vols = [vol10p, vol25p, atm, vol25c, vol10c]

# for a bunch of extrapolation factor values, generate the spline and get vol vs strike values

for extrap\_fact in [0.01, 10]:

# generate the spline

sp = vol\_spline(strikes, vols, texp, extrap\_fact)

# figure out the range of strikes for the plot; we'll use 1-delta on either side. We'll

# approximate the 1d vols with the 10d vols for the purpose of calculating the strikes

strike\_min = spot \* math.exp(vol10p \* vol10p \* texp / 2. + vol10p \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.01))

strike\_max = spot \* math.exp(vol10c \* vol10c \* texp / 2. - vol10c \* math.sqrt(texp) \* scipy.stats.norm.ppf(0.01))

nstrikes = 100

dstrike = (strike\_max - strike\_min) / (nstrikes - 1)

for i in range(nstrikes):

strike = strike\_min + i \* dstrike

vol = sp.volatility(strike) \* 100 # convert to % for display

# print out the values in comma-separated form to make it easy to paste into Excel for plotting

print(str(extrap\_fact) + ',' + str(strike) + ',' + str(vol))

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

test()