**Lecture 1 (Spot Markets) Assignment**

Due start of class, Wednesday September 11, 2019

**Question 1 (2 marks)**

Describe the four factors that contribute to the bid and ask prices a market maker will show to a client during voice trading?

* The inter-dealer market. This gives the market maker prices at which he knows he can hedge.
* Current risk position. If a market maker is long, she will tend to reduce her offer, and perhaps her bid as well, to make it more likely that a client will buy from her; if she is short, she will tend to increase her bid, and perhaps her offer as well, to make it more likely that a client will sell to her. In both cases she is incentivizing trading that will reduce her risk position while she still gets paid part of a bid/ask spread (and avoids having to pay a spread to hedge in the inter-dealer market).
* Market views. If she thinks the market is going up she will try to incentivize customers to sell to her by raising her bid price, and perhaps raising her offer price as well. The reverse if she thinks the market is going down.
* Client behavior. Does the client tend to always buy or always sell? Is the client typically right about market direction after their trade over time frames comparable to the market maker’s risk holding period?

**Question 2 (2 marks)**

Why has the daily turnover in the FX markets increased so much in the past fifteen years? Give some statistics.

Spot market turnover has grown from $0.6T/day in 1998 to $2.0T/day in 2013, mostly because of the move to electronic trading in the FX markets. Electronic trading leads to price transparency, which leads to lower bid/ask spreads, which incentivizes more day trading, which leads to higher volumes.

**Question 3 (4 marks)**

Describe the OTC market structure and the different roles involved in executing a trade. Describe the steps involved in executing a trade for voice trading and then for electronic trading.

Roles:

* Client: a market taker who asks dealers (either on the phone for voice trading, or via an application for electronic trading) to get bid and ask prices.
* Salesperson: works at a dealer, and talks to clients. In voice dealing they take orders from clients and mediate the client trades.
* Trader: works at a dealer, and has two functions: making prices to clients for new trades; and managing the portfolio risk resulting from taking the opposite side of client trades. In voice dealing these are humans; in electronic trading these functions are performed by computer programs, and human “traders” spend their day monitoring those programs.

Execution steps for a voice trade:

1. Client calls a dealer and asks for bid and ask prices for a specific currency in a specific notional.
2. Salesperson takes the client call, and yells over to the spot trader for that currency, asking for the two-way price in the size the client wants, and tells the trade the client name as well.
3. Trader looks at the inter-dealer market, his risk position, his views on the short-term market direction, and historical trading behavior of the client, and shouts bid and ask prices back to the salesperson.
4. The salesperson relays the prices back to the client, who decides whether she wants to buy at the offer or sell to the bid. If so, she tells the salesperson so, and the trade is done.

Execution steps for an electronic trade:

1. Client opens a trading application for a specific dealer on their computer, finds the screen for the currency she wants to trade and enters the notional she wants to trade.
2. The price-making engine at the dealer recognizes the request for quote and looks at the inter-dealer electronic market, the risk position of the electronic book, includes any short-term market direction bias (either manually-specified or determined algorithmically), and includes any bias due to historical trading behavior of the client. It calculates bid and ask prices and streams them back to the client, ticking them as those four inputs change through time.
3. The client sees the streaming prices in their application and, if she wants to trade, clicks to buy from the offer or sell to the bid.
4. The dealer receives the trade request and executes the trade (retaining the option to reject the trade request if the market has moved sufficiently far).

**Question 4 (4 marks)**

Today is October 27th, 2015. Tomorrow (October 28th) is a good business day for all three currencies. October 29th is a JPY currency settlement holiday. October 30th (a Friday) is a USD settlement holiday and November 2nd (a Monday) is a EUR settlement holiday. November 3rd is a good business day for all three currencies.

The EURUSD mid-market spot rate is 1.1300 (the price of a EUR in USD) and the USDJPY mid-market spot rate is 120.00 (the price of a USD in JPY). The USD interest rate is 0.25%, the EUR interest rate is 0.50%, and the JPY interest rate is 0.10%.

What are the spot dates for EURUSD, for USDJPY, and for EURJPY? What is the EURJPY mid-market spot rate implied from the triangle arbitrage?

The rule for determining a spot date: step ahead one business day, avoiding holidays for the two currencies in the pair; then step ahead one more business day, avoiding holidays for the two currencies in the pair *and* holidays for USD (if neither of the currencies in the pair are USD).

For EURUSD: 1d ahead goes to Oct 28, which is a good day for everything. 1d more goes to Oct 29, which is a holiday for JPY, but that doesn’t matter for EURUSD, so the EURUSD spot date is Oct 29.

For USDJPY: 1d ahead goes to Oct 28, a good day for everything. 1d more goes to Oct 29, which is a JPY holiday, so we skip over it. 1d more is Oct 30, which is a USD holiday, so we skip that too. That moves us to Nov 2, which is a EUR holiday, but that doesn’t matter for USDJPY; so the USDJPY spot date is Nov 2.

For EURJPY: 1d ahead goes to Oct 28. 1d more goes to Oct 29, which is a JPY holiday, so we skip it. 1d more goes to Oct 30, which is a USD holiday. Because this is the second of the two business day steps, and EURJPY is a cross, we also need to skip the USD holiday. That moves us to Nov 2, but that’s a EUR holiday, so we skip ahead one more day to Nov 3, which is good for everything. So the EURJPY spot date is Nov 3.

The midmarket spot for EURJPY is calculated via the triangle based on the FX forward rates for EURUSD and USDJPY to the EURJPY spot date, Nov 3. Due to the quoting conventions, the EURJPY spot is the product of the USDJPY and EURUSD forwards to Nov 3.

The USDJPY spot is 120.00, but that is for value date Nov 2. To get the forward to Nov 3 we need to use the expression for the forward rate

F = S exp( (R – Q) T)

where F is the forward, S is the spot, R is the denominated currency interest rate (in this case, the JPY interest rate, due to the quoting convention), and Q is the asset currency interest rate (in this case the USD interest rate). T is the calendar time from spot date to forward date – in this case, 1/365, since there is one calendar date from Nov 2 or Nov 3, and we’re assuming continuously-compounded actual/365 rates.

So that gives us a USDJPY forward of

F(USDJPY) = 120.00 exp( (0.001 – 0.0025) \* 1/365 ) = 119.9995.

Similarly for the EURUSD forward; here spot date is Oct 29, four calendar days before the EURJPY spot date:

F(EURUSD) = 1.13 exp( (0.0025 – 0.005) \* 5/365 ) = 1.129961.

The equivalent EURJPY rate is then 1.129961 \* 119.9995 = 135.5948. Note that it’s a product and not a ratio because of the market-convention quoting directions: EURUSD quotes the price of a EUR in USD, USDJPY quotes the price of a USD in JPY, and EURJPY quotes the price of a EUR in JPY.

**Question 5 (2 marks)**

Same market as Question 4. Assume zero bid/ask spread in interest rates.

Take the bid/ask for EURUSD as 1.1299/1.1301, and the bid/ask for USDJPY as 119.99/120.01. What is the bid/ask for EURJPY implied from the triangle arbitrage?

The EURJPY bid would use the bid spot for USDJPY and the bid forward for EURUSD, adjusted for the difference in spot date like with the last question. Similarly for the offer.

EURJPY bid = 1.1299 \* exp((0.0025 – 0.0050) \* 5/365) \* 119.99 \* exp((0.0010-0.0025) \* 1/365) = 135.5715

EURJPY ask = 1.1301 \* exp((0.0025-0.0050) \* 5/365) \* 120.01 \* exp((0.0010-0.0025) \* 1/365) = 135.6181

**Question 6 (10 marks)**

In Python, implement a variation of the “toy simulation algorithm” we discussed in class. Model parameters to assume:

* Spot starts at 1
* Volatility is 10%/year
* Poisson frequency  for client trade arrival is 1 trade/second
* Each client trade that happens delivers a position of either +1 unit of the asset or -1 unit of the asset, with even odds
* Bid/ask spread for client trades is 1bp
  + Receive PNL equal to 1bp\*spot\*50% on each client trade (since client trades always have unit notional in this simulation)
* Bid/ask spread for inter-dealer hedge trades is 2bp
  + Pay PNL equal to 2bp\*spot\*hedge notional\*50% on each hedge trade
* A delta limit of 3 units before the algorithm executes a hedge in the inter-dealer market.

Use a time step t equal to 0.1/ and assume that only a single client trade can happen in each time step (with probability equal to ). Use 500 time steps and a number of simulation runs to give sufficient convergence.

When converted between seconds and years, assume 260 (trading) days per year.

The variation in the algorithm: when the algorithm decides to hedge, it can do a partial hedge, where it trades such that the net risk is equal to the delta limit (either positive or negative depending on whether the original position was above the delta limit or below -1\*delta limit); or it can do a full hedge, like in the algorithm we discussed in class, where the net risk is reduced to zero.

Use the Sharpe ratio of the simulation PNL distribution to determine which of those two hedging approaches is better.

Your solution should deliver the Python script that implements the simulation, and you should explain your answer by giving numerical results from the simulation as well as some qualitative intuition behind the result. Include data that shows that you have used sufficient Monte Carlo simulation runs to show that your final result is not affected by statistical noise.

Marks will be given for both the numerical results generated from the simulation as well as the quality of your Python code. Remember to include lots of explanatory comments in your code and use variable names that are meaningful. Use external packages like numpy/scipy where applicable rather than rolling your own low-level numerical functions like random number generators. For top marks, use only vectorized operations across the Monte Carlo paths to speed up execution.

Results for hedging to zero:

* Sharpe ratio: 1.485 +/- 0.001
* Mean PNL 0.00084, standard deviation of PNL 0.00056
* Mean # of client trades: 47.6
* Mean # of hedge trades: 5.1

Results for hedging to edge of the band:

* Sharpe ratio: 3.338 +/- 0.002
* Mean PNL 0.00176, standard deviation of PNL 0.00053
* Mean # of client trades: 47.6
* Mean # of hedge trades: 6.2

These use 1,000,000 Monte Carlo runs. The standard error estimates on the Sharpe ratios come from running with different seeds seven times each and calculating the standard deviation of the results.

From the perspective of Sharpe ratio, it is better to hedge to the edge of the band than all the way to zero. The intuition: while the strategy results in slightly more frequency hedging (6.3 trades on average, vs 5.1 for hedging to zero), each hedge trade is smaller notional (usually 1 unit, vs 3 units) and the spreads paid to other dealers are smaller.

My code for this is attached. Note that it “vectorizes” all the calculations across paths, which means many of the computations happen at the C++ level – much faster than doing the same computations in Python. That script also includes a function “remote\_simulation\_fn\_slow” which performs those calculations without vectorizing them, and you can try that to see how it performs on a relative basis. I see the vectorized version running about 7x faster than the “slow” non-vectorized version.

Python code:

"""

Copyright: Copyright (C) 2015 Washington Square Technologies - All Rights Reserved

Description: Functions for running a simulation of electronic trading. Model is:

\* Spot follows a geometric Brownian motion with zero drift

\* Client trades happen according to a Poisson process

\* Client trades pay a fee of half the client bid/ask spread, defined

as a fraction of current spot

\* Client trades are always of unit notional, with even odds of delivering

a long or short position

\* Hedging happens when the net position reaches a delta threshold. Two

types of hedging supported: one where it rebalances the position to

zero whenever it trades; and one where it rebalances the position to

the delta threshold.

"""

from math import exp, sqrt

import scipy

def run\_simulation\_fast(vol, lam, sprd\_client, sprd\_dealer, delta\_lim, hedge\_style, dt, nsteps, nruns, seed):

'''Runs a Monte Carlo simulation and returns statics on PNL, client trades, and hedge trades.

"\_fast" because it uses vectorized operations.

vol: lognormal volatility of the spot process

lam: Poisson process frequency

sprd\_client: fractional bid/ask spread for client trades. eg 1e-4 means 1bp.

sprd\_dealer: fractional bid/ask spread for inter-dealer hedge trades. eg 1e-4 means 1bp.

delta\_lim: the delta limit at or beyond which the machine will hedge in the inter-dealer market

hedge\_style: 'Zero' or 'Edge', defining the hedging style. 'Zero' means hedge to zero position,

'Edge' means hedge to the nearer delta limit.

dt: length of a time step

nsteps: number of time steps for each run of the simulation

nruns: number of Monte Carlo runs

seed: RNG seed

'''

scipy.random.seed(seed)

trade\_prob = 1 - exp(-lam\*dt)

sqrtdt = sqrt(dt)

spots = scipy.zeros(nruns) + 1 # initial spot == 1

posns = scipy.zeros(nruns)

trades = scipy.zeros(nruns)

hedges = scipy.zeros(nruns)

pnls = scipy.zeros(nruns)

for step in range(nsteps):

dzs = scipy.random.normal(0, sqrtdt, nruns)

qs = scipy.random.uniform(0, 1, nruns)

ps = scipy.random.binomial(1, 0.5, nruns) \* 2 - 1 # +1 or -1 - trade quantities if a trade happens

# check if there are client trades for each path

indics = scipy.less(qs, trade\_prob)

posns += indics\*ps

trades += scipy.ones(nruns) \* indics

pnls += scipy.ones(nruns) \* indics \* sprd\_client \* spots / 2.

# check if there are hedges to do for each path

if hedge\_style == 'Zero':

indics = scipy.logical\_or(scipy.less\_equal(posns, -delta\_lim), scipy.greater\_equal(posns, delta\_lim))

pnls -= scipy.absolute(posns) \* indics \* sprd\_dealer \* spots / 2.

posns -= posns \* indics

hedges += scipy.ones(nruns) \* indics

elif hedge\_style == 'Edge':

# first deal with cases where pos>delta\_lim

indics = scipy.greater(posns, delta\_lim)

pnls -= (posns - delta\_lim) \* indics \* sprd\_dealer \* spots / 2.

posns = posns \* scipy.logical\_not(indics) + scipy.ones(nruns) \* indics \* delta\_lim

hedges += scipy.ones(nruns) \* indics

# then the cases where pos<-delta\_lim

indics = scipy.less(posns, -delta\_lim)

pnls -= (-delta\_lim - posns) \* indics \* sprd\_dealer \* spots / 2.

posns = posns \* scipy.logical\_not(indics) + scipy.ones(nruns) \* indics \* (-delta\_lim)

hedges += scipy.ones(nruns) \* indics

else:

raise ValueError('hedge\_style must be "Edge" or "Zero"')

# advance the spots and calculate period PNL

dspots = vol \* spots \* dzs

pnls += posns \* dspots

spots += dspots

return {'PNL': (pnls.mean(), pnls.std()), 'Trades': (trades.mean(), trades.std()), 'Hedges': (hedges.mean(), hedges.std())}

def run\_simulation\_slow(vol, lam, sprd\_client, sprd\_dealer, delta\_lim, hedge\_style, dt, nsteps, nruns, seed):

'''Runs a Monte Carlo simulation and returns statics on PNL, client trades, and hedge trades.

"\_slow" because it does not use vectorized operations.

vol: lognormal volatility of the spot process

lam: Poisson process frequency

sprd\_client: fractional bid/ask spread for client trades. eg 1e-4 means 1bp.

sprd\_dealer: fractional bid/ask spread for inter-dealer hedge trades. eg 1e-4 means 1bp.

delta\_lim: the delta limit at or beyond which the machine will hedge in the inter-dealer market

hedge\_style: 'Zero' or 'Edge', defining the hedging style. 'Zero' means hedge to zero position,

'Edge' means hedge to the nearer delta limit.

dt: length of a time step

nsteps: number of time steps for each run of the simulation

nruns: number of Monte Carlo runs

seed: RNG seed

'''

scipy.random.seed(seed)

pnls = []

trades = []

hedges = []

trade\_prob = 1 - exp(-lam \* dt)

sqrtdt = sqrt(dt)

for run in range(nruns):

x = 1

rs = scipy.random.normal(0, sqrtdt, nsteps)

qs = scipy.random.uniform(0, 1, nsteps)

ps = scipy.random.binomial(1, 0.5, nsteps)

pos = 0

pnl = 0

n\_trades = 0

n\_hedges = 0

for step in range(nsteps):

# check if there's a client trade in the window. If so, assume it happens at the start of the window

if qs[step] < trade\_prob:

if ps[step] == 0:

pos -= 1

else:

pos += 1

pnl += sprd\_client \* x / 2. # get paid the bid/ask spread on the unit notional of the trade

n\_trades += 1

# check whether we need to hedge; if so, hedge back to zero position

if hedge\_style == 'Zero':

if abs(pos) >= delta\_lim:

pnl -= abs(pos) \* sprd\_dealer \* x /2. # pay the bid/ask spread

pos = 0

n\_hedges += 1

elif hedge\_style == 'Edge':

if abs(pos) > delta\_lim:

if pos > delta\_lim:

pnl -= (pos - delta\_lim) \* sprd\_dealer \* x / 2.

pos = delta\_lim

else: # pos<-delta\_lim

pnl -= (-delta\_lim - pos) \* sprd\_dealer \* x/ 2.

pos = -delta\_lim

n\_hedges += 1

else:

raise ValueError('hedge\_style should be Zero or Edge')

# step forward the spot

dS = vol \* x \* rs[step]

pnl += pos \* dS

x += dS

pnls.append(pnl)

trades.append(n\_trades)

hedges.append(n\_hedges)

pnls = scipy.array(pnls)

trades = scipy.array(trades)

hedges = scipy.array(hedges)

return {'PNL': (pnls.mean(), pnls.std()), 'Trades': (trades.mean(), trades.std()), 'Hedges': (hedges.mean(), hedges.std())}

def test\_simulation():

'''Little test function that demonstrates how to run this'''

vol = 0.1\*sqrt(1/260.) # 10% annualized vol, converted to per-day vol using 260 days/year

lam = 1\*60\*60\*24 # Poisson frequency for arrival of client trades: 1 per second, converted into per-day frequency to be consistent with vol

sprd\_client = 1e-4 # fractional full bid/ask spread for client trades

sprd\_dealer = 2e-4 # fractional full bid/ask spread for inter-dealer hedge trades

hedge\_style = 'Zero' # 'Zero' means "hedge to zero position", or 'Edge' means "hedge to delta limit"

delta\_lim = 3. # algorithm hedges when net delta reaches this limit

dt = 0.1/lam # time step in simulation. Only zero or one client trades can arrive in a given interval.

nsteps = 500 # number of time steps in the simulation

nruns = 10000 # number of Monte Carlo runs

seed = 1 # Monte Carlo seed

res = run\_simulation\_fast(vol,lam,sprd\_client,sprd\_dealer,delta\_lim,hedge\_style,dt,nsteps,nruns,seed)

print('PNL Sharpe ratio =',res['PNL'][0]/res['PNL'][1])

print('PNL mean =',res['PNL'][0])

print('PNL std dev =',res['PNL'][1])

print('Mean number of client trades =',res['Trades'][0])

print('SD number of client trades =',res['Trades'][1])

print('Mean number of hedge trades =',res['Hedges'][0])

print('SD number of hedge trades =',res['Hedges'][1])

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

import time

t1 = time.time()

test\_simulation()

t2 = time.time()

print('Elapsed time =',t2-t1,'seconds')