Niall O'Connor

Developer, Engineer, Musician

Monte Carlo simulations in python

You will require matplotlib for python.

```
import
from            import
```

Monte-Carlo simulations are based on random numbers. The simplest example is as follows. You roll a dice which will give a value between 1-6. You can either add or subtract that value to a running total. You flip a coin to decide whether you add or subtract, you decide if heads means + and tails means -.

The coin flip can be implemented as follows.

```
     = lambda :        .       ([1, -1])
```

This will return a 1 or -1 which we can multiply into the dice roll to give us a signed value.

The dice roll.

```
     = lambda :        .        (1,6)
```

I don't need to wrap this in a lambda but it might make things more readable later.

Lets give these functions a test run.

```
>>>     ()
-1
>>>     ()
1
>>>     ()
1

>>>     ()
2
>>>     ()
4
>>>     ()
4
>>>     ()
5
```

So the idea is that we toss the dice several times while flipping the coin to see should we add or subtract.

```
>>> [    () *     () for   in      (10)]
[-1, 1, -2, 6, -5, -5, 6, -4, -5, 2]
```

The resulting list is the movements we need to make. We don't just want to jump up and down between -6 and 6. We need to trend these movements if we want to produce a graph. Starting at 0 or any random number we add the first flip-dice value which is -1. From -1 we add 1 giving 0. From 0 we subtract 2 giving -2. -2+6 = 4. 4-5 =-1. -1-5 = -6. -6+6 = 0. 0-4 = -4. -4-5 = -9. -9+2 = -7. We can use a trend function to build a trend.

```
def      (    ):
         = [0] # we need a starting point
    for   in     :
              .      (      [-1] +  )
    return
```

There are probably better ways to implement that in numpy, but this function will work for now.

So if we pass in a list of the following values [-1, 1, -2, 6, -5, -5, 6, -4, -5, 2] we will expect the following result [0, -1, 0, -2, 4, -1, -6, 0, -4, -9, -7]

```
>>>      ([-1, 1, -2, 6, -5, -5, 6, -4, -5,
2])
[0, -1, 0, -2, 4, -1, -6, 0, -4, -9, -7]
```

Ok, we are ready to do a Monte-Carlo simulation. If we roll the dice 80 times then we have 80 data points. In finance we sometimes use future dates as our data points when we forward simulate. 1Day, 2days, 3Days, 1Week, 2Weeks, 1Month, 2Month, 3Month, 4Month, 5Month, 6Month, 9Month, 1year, 18Month, 2year, 3year, 5year, 10year, 15year, 20year, 25year, 30year could be a set of dates representing interest rates or yields.

For our test lets do 80 data points. Now, running one set of dice rolls is usually not enough to get a feel for what may happen when we forward simulate. You may want to run many simulations and take a sample of all the runs. We call each set of 80 rolls a path an In our example we will do 200 paths. That means we will roll the dice and flip the coin 80 times per path and we will repeat for 200 paths. lets code something using our dice, flip and trend functions.

```
          >>>       = [      (    )  for      in
                       [
                           [    ()  *     ()  for    in
               (80)]
                           for   in       (200)
                       ]
                  ]
```

What a mouth full… lets break it down.

firstly we want to trend(path) for every path because instead of the actual dice rolls we want the movement they represent.

We are getting the path variable for every path in our 200 paths. That is the for p in range(200) bit. range(200) just gives us 200 numbers to iterate over. We don't use any of those numbers we just want 200 iterations. for t in range(80) gives us 80 iterations for dice and flip. dice() gives a random number between 1-6 and when we multiply the number with the result of flip() it gives a – or plus sign on our dice roll.

For easy reading here is another implementation.

```
                =     ()
          for   in      (200):
                        =     ()
              for   in      (80):
                     =      ()  *      ()
                          .       (         )
                    .       (            )
```

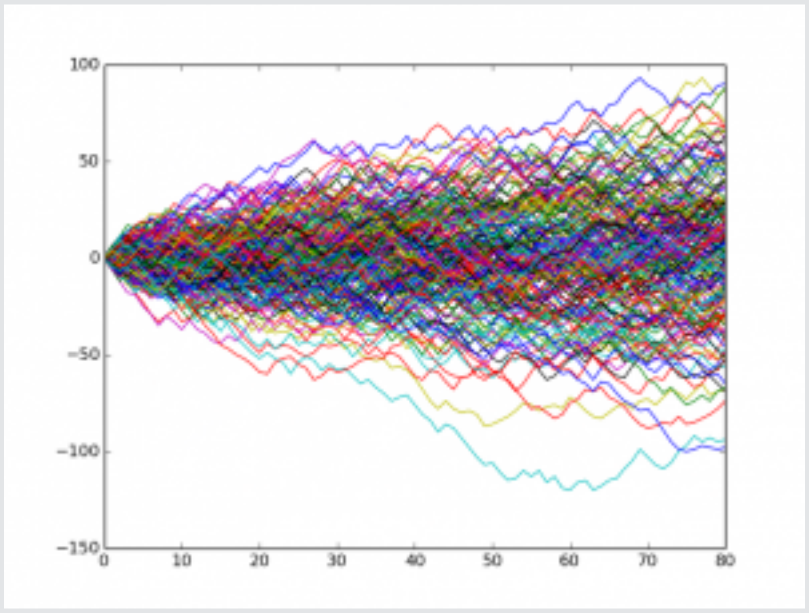
Paths is a list containing 200 lists again, or 200 paths. Each path has 80 data points which represents a trend. They represent a simulated movement over 80 iterations. There are 200 such simulated movements or paths.

We can graph this now 🙂

```
   from               import
   for   in         :
         .     (  )


         .     ()
```

Behold!!!



monte carlo sim results

Each of the 200 paths are shown in this wonderful image that I’ve entitled “What happens when you roll a dice and toss a coin 80 times in a row, 200 times over!”.

Before we run off to the stock market to lose our money its worth noting a few things.

1. The random number generators used in this example are not nearly sophisticated enough. They only give the illusion of a random pattern. You need to invest heavily in any random number generator.

2. The flip or dice function is too simple. It does not factor in the many things you need to factor when pricing options or forwards. Its not a serious simulation :-). You need precise formula (Black-Scholes, stochastic e.t.c) that can handle weighting and number correlations. Still the concept is sound. Its a black box that tells you how far you must move between each iteration or data point you wish to simulate.

3. To fully connect this code with what happens in the world of finance is a big jump. But I will attempt to do so.

Taking the current price of an asset like potatoes. If I agree to buy a million potatoes in one years time for 1 dollar a potato, then that is what we call a forward contract, call it forward for short. As the price of potatoes moves up and down it affects my forward.

Current market value of potatoes = $2 — I can buy $2million of potatoes for only $1million!!!!

I can convince a bank to loan me some money for the deal because, the security used against the loan they offer will be potatoes, that I buy for a dollar, but are worth 2 dollars! (that is a very generous move in the potato market). My "exposure" is good here.

However… the small print… if the price of potatoes should drop to 50cent, then, I'm contractually obliged, by the forward, to stump up the full price of 1 dollar a potato. The bank will be very reluctant offer me that loan.

Current market value of potatoes = 50c   — I can buy $500,000 of potatoes for $1million. Sounds dumb!

When we simulate, using sophisticated formula and consider relevant factors, we can chart scenarios and try to discover what factors will play against us. For example we could buy options to buy or sell potatoes at different prices. A stock option is the option but not the obligation to buy an asset or security at a price on a date. So if the price of potatoes drops the value of the stock options we own could increase. Its like betting some on red as well as black. This is called hedging your bets.

In conclusion… I'm not a financial expert and simulations are only as good the use you put them too. But if you want to begin simulation in python this post should help.

Categorized in: Coding, Fun

Posted on February 9, 2015 by admin