**Question 1: Finance**

* 1. Write a 500-word explanation of Bitcoin stock-to-flow model and make an argument for why it is a bad model?
  2. (Please show your workings). Yara Inc is listed on the NYSE with a stock price of $40 - the company is not known to pay dividends. We need to price a call option with a strike of $45 maturing in 4 months. The continuously-compounded risk-free rate is 3%/year, the mean return on the stock is 7%/year, and the standard deviation of the stock return is 40%/year. What is the Black-Scholes call price?

1. Bitcoin stock-to-flow model

A stock is measured at one specific time, and represents a quantity existing at that point in time (say, December 31, 2004), which may have accumulated in the past. A flow variable is measured over an interval of time. Therefore, a flow would be measured per unit of time (say a year).

The Stock to Flow model measures the relationship between the currently available stock of a resource and its production rate. PlanB@100trillionUSD is a Dutch institutional investor with a legal and quantitative finance background. He created the bitcoin Stock-to-Flow (S2F) model where he uses scarcity to quantify bitcoin value. S2F model is not only applicable to bitcoin but also to gold, silver and other assets.

Bitcoin (BTC) Stock-to-Flow (S2F) model was published in March 2019. The original BTC S2F model is a formula based on monthly S2F and price data. Since the data points are indexed in time order, it is a time series model. This model has activated quantitative analysts around the world.

In order to examine any model critically, there is a need to go through the theoretical assumptions of that particular model. This model is a bad model because;

1. Most of the price models depend upon some uncertain assumptions, and they should not be given that much importance as they are getting in the community. It is highly important for any price model to be based upon real facts and not plain assumptions. Only then, will it be able to hold some credibility.
2. The Bitcoin (BTC) Stock-to-Flow (S2F) model defies physics. In 2140, which is when we cannot mine new bitcoins. At that point, the S2F model predicts that the price of bitcoin will go to infinity. For this to happen;

Firstly, Bitcoin’s price would have to double every year, on average, for the next 30 years. That’s 30 doublings. No asset has ever come close to such a performance. Maybe pre-IPO Microsoft or Google or Walmart had such a rise for 10 years. But doublings become extremely difficult once an asset becomes large.

Secondly, we would need to invent nuclear fusion reactors and become a Type 1 Civilization. Bitcoin consumes vast amounts of energy. The higher the price goes, the more it consumes.

1. Not everyone agrees on what is gold’s stock-to-flow ratio, gold’s stock-to-flow isn’t fixed, gold’s stock-to-flow does not drive its price. Some metals with extremely low stock-to-flow ratios are worth more than gold, S2F doesn’t explain the prices of other crypto currencies, S2F assumes that Bitcoin’s demand continues to grow exponentially.
2. Given that;

Stock Price = S = 40

Strike Price = k = 45

Time to expiration = T = 40/12 = 3.3

Volatility = б = 0.4

Risk Free Rate = 0.03

C = call option value =?

Solution:

d1 = ln(S/k) + (r)T + 0.5бsqrtT

бsqrtT

= (ln(40/45)+ (0.03)(3.3)+0.5(0.4)sqrt3.3) / 0.4sqrt3.3

= 0.3374

d2 = d1 – бsqrtT

= 0.3374 – 0.4sqrt3.3

= - 0.3892

Nd1 = N(0.3374) = N(0.33) + 0.74[N(0.34) – N(0.33)]

= 0.6293 + 0.74(0.6331 – 0.6293)

= 0.6321

Nd2 = N(-0.3892) = N(-0.38) - 0.92[N(-0.38) – N(-0.39)]

=0.3520 – 0.92(0.3520 – 0.3483)

=0.3486

C = C = SN(d1) – Ke-rT N(d2)

C =40 \* 0.6321-45\*2.7182(-0.03\*3.3)\*0.3486

= 25.284 – 14.2077

= 11.0763.

**Question 2: Computer Science**

1. Why is it a bad idea to use recursion method to find the fibonacci of a number?
2. Write a function that takes in a Proth Number and uses Proth's theorem to determine if said number is prime? You can write this in any programming language but C/C++/Golang are preferred.

Answers:

1. In mathematics, the Fibonacci numbers, commonly denoted Fn, form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1. The Fibonacci sequence of numbers is as follows: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, etc. The key Fibonacci ratio of 61.8% is found by dividing one number in the series by the number that follows it. For example, 21 divided by 34 equals 0.6176, and 55 divided by 89 equals about 0.61798. Applications of Fibonacci numbers include computer algorithms such as the Fibonacci search technique and the Fibonacci heap data structure, and graphs called Fibonacci cubes used for interconnecting parallel and distributed systems.

Fibonacci retracements are often used as part of a trend-trading strategy. In this scenario, traders observe a retracement taking place within a trend and try to make low-risk entries in the direction of the initial trend using Fibonacci levels. Fibonacci retracement levels are horizontal lines that indicate where support and resistance are likely to occur. They are based on Fibonacci numbers. The Fibonacci retracement levels are 23.6%, 38.2%, 61.8%, and 78.6%. Fibonacci retracements can also be applied to stocks that are falling, in order to identify the levels up to which the stock can bounce back.

Recursion is the technique of making a function call itself. This technique provides a way to break complicated problems down into simple problems which are easier to solve. With respect to a programming function, recursion happens when a function calls itself within its own definition. It calls itself over and over again until a base condition is met that breaks the loop. Recursion can reduce time complexity. Recursion adds clarity and reduces the time needed to write and debug code. Recursion uses a lot memory, because the function has to add to the stack with each recursive call and keep the values there until the call is finished. Recursion can be slow if not implemented correctly (with memoization). The reason that recursion is slow is that it requires the allocation of a new stack frame. The time of computing the nth Fiboancci number is exponential for a recursive algorithm. The recursive implementation of Fibonacci is one of the slowest possible implementations. It is slow since its complexity is exponential in most popular languages. A language with built-in memoization is able to optimize it to linear complexity, but C, C++, python, Java, and JavaScript implementations do not have this optimization. The recursion is preferred for pedagogical purposes, since it is much simpler than the alternatives.

If the goal is to compute the Fibonacci sequence or the nth number in the Fibonacci sequence, naive recursion, which recomputes the same value repeatedly, it is one of the least efficient ways to do it, but you’ll often see computation of the nth number in the Fibonacci sequence used to illustrate recursion, not because it’s a good way to compute numbers in the Fibonacci sequence, it isn’t, but because it’s a good way to demonstrate recursion. Recursively computing numbers in the Fibonacci sequence is elegant and readable, very clearly illustrates recursion, and it leads neatly into a discussion about the potential costs of using recursion. Doing it iteratively is usually more efficient, but tends to be less elegant and less readable, but recursion with implicit (or even explicit) memoisation i.e catching previously-computed values is a nice compromise between efficiency and readability.

1. #using python language

import math as mt

prime = [0 for i in range(1000000)]

# Calculate all primes upto n.

def SieveOfEratosthenes(n):

# Initialize all entries it as true.

# A value in prime[i] will finally

# false if i is Not a prime, else true.

for i in range(1, n + 2):

prime[i] = True

prime[1] = False

for p in range(2, mt.ceil(n\*\*(0.5))):

# If prime[p] is not changed,

# then it is a prime

if (prime[p] == True):

# Update all multiples of p

# greater than or equal to

# the square of it numbers

# which are multiple of p and are

# less than p^2 are already been marked.

for i in range(p \* p, n + 1, p):

prime[i] = False

# Utility function to check power of two

def isPowerOfTwo(n):

return (n and (n & (n - 1)) == False)

# Function to check if the Given

# number is Proth number or not

def isProthNumber(n):

k = 1

while (k < (n // k)):

# check if k divides n or not

if (n % k == 0):

# Check if n/k is power of 2 or not

if (isPowerOfTwo(n // k)):

return True

# update k to next odd number

k = k + 2

# If we reach here means there

# exists no value of K such

# that k is odd number and n/k

# is a power of 2 greater than k

return False

# Function to check whether the given

# number is Proth Prime or Not.

def isProthPrime(n):

# Check n for Proth Number

if (isProthNumber(n - 1)):

# if number is prime, return true

if (prime[n]):

return True

else:

return False

else:

return False

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

# Driver Code

n = int(input("Please kindly enter a number:"))

# if number is proth number,

# calculate primes upto n

SieveOfEratosthenes(n)

for i in range(1, n + 1):

# Check n for Proth Prime

if isProthPrime(i) == True:

print(i)

**Question 3: Mathematics**

* (Please show your workings). Over all real numbers, find the minimum value of a positive real number, y such that

y = sqrt((x+6)^2 + 25) + sqrt((x-6)^2 + 121)

y = sqrt(x+6)2 + sqrt25 + sqrt(x-6)2 + sqrt121

y= x+6 + 5 + x – 6+ 11

y= 2x + 16

2x +16 = 0

2x = -16

x = -8

y = 2x +16

y = 2(-8) +16

y = -16 +16

y = 0, this is the minimum value of y.

(x, y) = (-8, 0), 0 is the minimum value of y.