COMP0043 Numerical Methods:

Truncation, discretization and grids (basics):  
Have continuous time, need to discretize this by introducing N intervals where:

Change x=(b-a)/N

A refers to lower truncation limit (ie starting value of interval) and upper truncation b, which refers to:

Np.arange(a,b,(a-b)/N) in python

Or

Np.linspace(a,b,N) in python

Xn=a+n\*change x, then you get grid going overtime.

Step time=change x=dt

These equispaced grids where step change x is constant.

Typically, intervals a and b are centered around 0 hence a=-b 0 as grid mid-point. Errors relating to calc:

* Truncation error: in change x, make a and b go to infinity.
* Discretization error: in change x, make N massive, change x goes to 0.

Will always have one of these errors in some cases both will occur.

Machine accuracy- machine epsilon smallest positive floating-point number that when added to 1, produces floating-point result different to 1. Round errors accumulate with an increase number of calculations.

For 32-bit double precision values, E has 11 bits and F has 52 bits.

Objective of numerical analysis- design stable algos to minimize truncation and discretization errors through use of CPU time and memory.

Histograms:

Plt.hist(dataset,bins,density=True) in python. Don’t touch np.histogram() as In don’t design your own from scratch!

Normal Distribution:

Defined pdf with 2 parameters of mu and sigma. CDF requires library function and provide numerical solution as well.

Normal.m script- defined mu and sigma alongside truncations a and b variable. Can use formulas or library functions from scipy.stats. Defined grid as N to derive change in x above to then feed into linspace across ranges, this is used as x-mu/sigma. Hence used to chart pdf and plot derived pdf and cdf against it.

Second part of script consists of generating random numbers via mu+sigma\*rand(nsample,1) where its replicating mean+sigma\*Dw where dw is random numbers.

Rand() derive uniform distribution across 0 and 1 to derive probability used which is then translated using inverse norm to derive the occurrence of the probability of occurring.

Pdf and cdf of lognorm, provided on distributions relevance in finance. Its pdf has an extra x term provided. Use library function for it.

Square distribution- chi-square distribution, if you take sum of squares up to N- alongside number of degrees of freedom.

Non central chi squared: pdf expression, need to use library function for it, ncx2 in python.

Exponential distribution: too simple as shown by pdf and exponent, so gamma e power -gamma\*. Too simple.

Look at scripts a1q1 and a1q2 (matrices post)

Simulating random variables:

Can use rand() derives value from 0 to 1 from uniform distribution- take the inverse transformation of the CDF> ie the value at x axis of probability provided by rand in y axis.

Check normal.m- generate random numbers can either:

* Use rand then take norminv of rand and multiply by sigma
* Use icdf so inverse cdf of normal rand directly
* Sigma\*randn() already provides rand adjusted by inverse cdf.

Exercise 1, look at exponential.m and overplot theoretical exponent values with library packaged modules such as pdf exponential and cdf.

Cdf of lognormal, difficult so not able to derive via single line but can derive normal distribution via mu+sigma\*randn(nsample,1) then take the exponential of it:

Exp(mu+sigma\*randn(nsample,1)) to derive lognormal sample distribution to generate the pdf.

Normal distribution comes with a challenge with the cdf cannot be expressed analytically.

Exercise 2, derive the cdf formula, so analytical solution for normal distribution.

Exercise 3, derive the pdf and cdf formula for non-chi squared distribution-need to figure out how to derive the Bessel function in python of the first kind, assumed spicy.special.iv but unclear on inputs.

Assignment 1 convert continuous problem to discrete problem implying facing truncation and discretization error (using limited change x).

Do all integrals have both discretization and truncation error- no because if integral goes from a and b- both finite, no discretization. Allocated integration weights, which is where trapz comes in.

-Fourier transform methods- need to compute an integral of characteristic function=trapz.

<https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4405779>

Random Number Generator:

Uniform deviates: choose a,b M and a seed where the algorithm, given N0 is chosen does:

Ni=mod(aNi-1+b, M)

Where first step implies mod(aN0+b, M) equivalent to saying x%y.

Seed refers to initial random number used in to commence the nsample you want to run.

Ui=is defined as Ni/M, equivalent in saying normalize all values generated by M (get values between 0 and 1).

These generate sudo random numbers, not fully random but hard to predict.

N- largest integer that you can use/represent in your processor (64 processor, 2^64 CPU- feed as N).

Characteristics:

* Ni goes from 0 to M-1 (eg 43 mod 20, its 3, while 19 mod 20, its 19). But when doing N/M=U, never 1 because 20 mod 20, 0- get 0 but not the 1? Apply condition to replace 0 with 1.
* N sequence periodic with period p<M – due to deterministic. Hence pseudo random number.

A and b cannot be chosen at random- make sure does not define closed loop.

Good random numbers?

P must be large so M must be large, where M=2^l, given 64 processor, use 2^64. Higher value, higher the loop/sequence.

Numbers must be distributed as intended, confirm empirical density of the kth intervals. But simple step is design scatter plot, see visually how the values are dispersed.

Values provided shows it going wrong- plotting show same directional move of line- if lines occur means high level of predictability- not ideal, remove randomness due to high correlation.

Fibonacci Generator:  
First generate linear congruential generator, then based on the derived sequence:

Mod((Ni-v)-(Ni-u),M)/M

Where v is taken as the 17 and u is 5, two arbitrary numbers (not 17). Show with simple modification how it improves the random generator.

Random numbers for other distributions:

Transform from 1 distribution to another via applying inverse function of a cdf.

Or can use scalar transformation that can be used.

Or see the formula used for exponential distribution- see conversion of values.

Here approximate CDF and ICDF.

Method Box and Muller:

Utilize second dimension, two-dimension transform. Takes:

Y1=sqrt(-2logx1)cos2pix2=h1(x1,x2)

Then take inverse for it.

The logic of its algorithm is based on:

Generate 2 random distribution samples, U1, U2 with uniform distribution [0,1].

Then rho=2piU2 and pho=sqrt(-2\*log(U1))

Z1=pho\*cos(rho) with N(0,1)

Z2=pho\*sin(rho) with N(0,1)

In the script, then you do Z1+Z2= standard random numbers

Then chart N vs random numbers via rand()\*pdf of N with mean=0,variance=1. See distribution of random numbers.

Acceptance-Rejection Method:

You can obtain distribution that you want by taking ratio between an area uniformly covered with dots and the area that is below your target distribution. Want to have larger area as close as possible to area you want to simulate.

Try to find function that covers majority of the pdf.

Logic is based on defining:

N number of samples, c as sqrt(2\*exp(1)/pi) where:

ucg(x)<=f(x)

Generate random numbers: via rand where these are sampled by:

x=log(2\*random numbers)\*(random numbers below<0.5)-log(2\*(1-random numbers))\*(random numbers>=0.5)

Then use the laplace-density where:

g(x)=0.5\*exp(-abs(x))

fx=1/sqrt(2pi)\*exp(-L\*\*2/2) which is the pdf logic used where L is x.

Then derive a list of values where x vales satisfy this condition: ucg(x)<=f(x), ie will generate true and false list and isolate those values that meet the conditions.

To find maximimum value of f(x)/g(x) over all xs, given f(x) normal density is symmetrical to origin, need to consider positive values of x.

F(x)/g(x) is maximized at the mode of standard normal, where x=0:

F(0)/g(0)=(1/sqrt(2\*pi)\*exp(0)/(0.5\*exp(0))=1/sqrt(2\*pi)/0.5, hence c=0.5?

According to script, c=sqrt(2\*exp(1)/pi).

Improve Box Muller by combining it with Acceptance rejection using Variant of Marsaglia:

Reason? Don’t want to compute cosine and sine.

Sample within unit square so -1 to 1 and down sample to unit disc- then obtain angles to ger a cosine and sine. No need to go indepth.

Multiplications more expensive than sums or subtractions= and trigonometrical functions are more expensive than multiplications including logs and exponents, hence this algorithm is cheaper, ie easier to run.

When computing monte carlo when you need to derive cosine and sine for 100000 numbers then it starts to matter.

The logic is based on:  
compute v1=2\*random values sample-1, v2=2\*random values sample-1. Then from it do square of both (v1\*\*2+v2\*\*2= get unit circle) then added and only isolate the values below 1, ie sum below 1. Normalize so that they can stay within -1 to 1 range.

Then based on this sequence=WI, do:

Rho=sqrt(-2\*log(WI)/WI)=Z1, then:

Ni=Rho\*v1+Rho\*v2

Vert similar to Box muller but replicate the use of cosines and sines for multiplications so should derive faster values. Can use tic and toc for it.

Correlated Normal Random Variates:

Are negatively correlated- want to achieve covariance matrix and then derive a target density function but have an additional metric sigma. You replicate Box Muller with sigma matrix. Check

Chat-gpt solution provides solution where we ask:  
“if the sum is equal to the matrix [sigma1 squared, psigma1sigma2,psigma1sigma2,sigma2squared] which is the equivalent covariance between two random variables which replicates the logic of a correlated normal random variates process, show that the solution is: sigma1Z1 and sigma2pZ1+sigma2\*sqrt(1-p squared)\*Z2 where p=correlation value”

Quasi random numbers:

Construct points similar random numbers but avoid clustering- tend to be self-avoiding.

Need uniform coverage but it also needs to be homogenous. Hence derive sequences to aht self-avoiding.

The accuracy scales as the sqrt(n) if using plain monte carlo, using quasi numbers, n-2/3.

Exam- generate random numbers and plot them in 2D and 3D scatter plots using methods above.

Stochastic differential equation:

Dx(t)/dt=mu\*X(t)+sigma\*epsilon(t), which the stochastic differential equation would be represented as:

dX(t)=mu\*(X(t),t)dt+sigma(X(t),t)dW(t)

where dW is wiener process. Drift-diffusion process, the unknown is the stochastic process known as X(t).

Itos formula:

dW squared=dt, y satisfies:

dY=dy/dw\*dW+1/2\*dy2/dw2\*dt

then derive itos lemma IV! Hence, dY=(…+a\*…+1/2sigma squared…)dt+sigmadW

Bachelier set the variable x in the diffusion equation= stock price S, which could be negative as such he modelled the log-price so X=log(S/So) with an arithematic Brownian motion and derived S=So\*exp(x) with geometric Brownian motion.

Black-scholes merton equation (1973):  
dS=mu\*S\*dt+sigma\*S\*dW

Where you then know the contract V(S,t) is defined as Itos lemma IV so:

Dv=(all info)dt+ sigma \*S\*dv/ds\*dW

Delta hedge so portfolio=dV-deltaS and you know that dP=rPdt

Hence derive the free of arbitrage black scholes formula.

Then shift formula to get dv/dt=…. See similar logic to kalgorov equation.

Consider other greeks: dV/dS=delta; dV2/dS2=Gamma; etc

Can include dividend payments by considering dP=dV-delta\*dS-rdelta\*S\*dt

Cum is not paid dividends and ex is already paid.

Solution for BSE is then always: C(S,t)=Sexp(-q(T-t)N(d1)-Kexp(-r(T-t))N(d2) for call and for put its negative -d1 and -d2 and inverse logic.

Can derive the CDF for each and derive the auxiliaries for d1 and d2.

Fair forward price:

F(S,t)=Sexp((r-q)(T-t))

And moneyness which measures position of F relative to K:

m=log(F/K)/sigma\*sqrt(T-t)=0.5\*(d1+d2) hence:

d1,2=m±0.5\*sigma\*sqrt(tau)

Monte Carlo Methods:

Scalar SDE for dX=a(x,t)dt+b(X,t)dW, use discretization over each timestep hence apply a logic of:

Y=y-1+a(y,t)\*delta\*t+b(y,t)\*delta\*W

Which formula wise It would be St=So\*exp(1+rate\*mean\*dt+sqrt(dt)\*sigma\*rand())

Hence use euler Maruyama logic.

Need to derive the approximation error= E[Xt-yt] hence expected value of difference between the analytic solution of Xt and the approximation value which is then known as the absolute error. Then find out dW is represented as sqrt(dt).

Larger gammas=quicker convergence so want large gammas.

Milstein method: Euler maruama + ½\*bb\*{delta\*W squared- delta\*t}

Better approximation then euler’s method.

Same correction in euler’s method is the application of analytic moments which is not a new approach but simply a correction.

Abm.m script:  
where use euler-maruyama: mu\*dt+sigma\*randn(npaths,nsteps)\*sqrt(dt)

Where dt is T/nsteps, hence time step across 0 to expiry. The logic specifies cumulative sum overtime given npaths start at 0!

The expected path= mu\*t each time step, vs simply taking a mean at each time step.

Variance, refers to the mean square deviation which is the equivalent of saying sigma\*\*2\*t and comparing it to variance overtime.

Consider also mean absolute deviation, meaning mean at each time step of the abs of X-EX.

The compare that with sigma\*sqrt(2t/pi).

To derive the pdf of each time step, just isolate a time step and derive the histogram for it and compare it to the normal pdf of it given the mean for that time step and sigma for that time step!!!!

Std overtime of ABM looks similar or identical to mean absolute deviation.

It is not a stationary process= the probability of BM overtime becomes larger.

Solution of fockker-planck equation:

Refers to deriving the pdf at each time step in a 3d surface area.

Where to derive the pdf at each time step, we need to consider:

Pdf=1/2\*sqrt(pi\*D\*t)\*exp(-(x-mu\*t)\*\*2/(4\*D\*t))

Where x is arrange going from -1 to 1 in increments of 0.02

T2 values going from 0.1 to 1 in increments of 0.025

And D refers to the diffusion coefficient, which refers to sigma\*\*2/2

Replicate pdf logic so 1/2sqrt(tpi)\*exp(-x\*\*2/2)

|  |  |  |
| --- | --- | --- |
| Deterministic | Stochastic | Name |
| dx=mu\*dt | dX=mu\*dt+sigma\*dW | Arithmetic Brownian Motion |
| Dx=mu\*x\*dt | dX=mu\*X\*dt+sigma\*X\*dW | Geometric Brownian Motion |
| Dx=alpha\*(mu-x)\*dt | dX=alpha(mu-X)dt+sigma\*dW  dX=alpha(mu-X)dt+sigma\*sqrt(X)\*dW  dX=alpha(mu-X)dt+sigma\*X\*dW | Ornstein-Uhlenbeck (Vasicek)  Square root (Cox-ingersoll)  Lognormal mean-reversion |
|  | dX=mu\*Xdt+sqrt(v)\*X\*dW  dv=alpha(mu-v)dt+k\*sqrt(v)\*dW | Stochastic Volatility Heston model |

Also cover Brownian Bridge: dX=((b-X)/(T-t))dt+dW, only model with dependency on time.

For Ornstein, square root and lognormal, its drift term is -X, implying pricing does not grow as much as geometric Brownian motion. There is a non-deterministic force pulling it towards the mean.

Stochastic volatility- Heston its sigma terms vary with time, vs fixed sigma of GBM.

Hull-White similar to GBM.

All models do not depend on t, only depend on x which means they indirectly depend on t.

Mean reverting models, ideal for fx, or interest rates, not really stocks.

Brownian bridge, we always know the final state which is something that is not clear for all other processes.

Gbp.m script:

Derive the arithmetic Brownian motion then apply simply a So\*exp(X). Meaning after computing abm increments and taking the cumsum overtime, derive So\*exp(X).

Done

Vasicek mean-reverting process:

When interest rates and inflation rates tend towards lower levels they are followed by high levels and inverse. This is known as mean-reversion which can be modelled via Mean-reverting process which matches with is known as Uhlenbeck or Vasicek model.

The logic is taking GBM and then applying a loop for each time step, not to different given that both models have a fluctuating drift term but diffusion remains fixed.

Alpha refers to the mean reverting speed.

The paths remain more together and converge to the dashed line which is the long-term average. If you look at the different densities at different time steps, eventually they converge into the same value.

When looking at variance of OH model, beginning it behaves like arithmetic Brownian motion but then converges on the long-term.

It is defined as quasi stationary becomes overtime it aligns to a stationary process.

Hypothetically, increasing time steps should lead to better convergence of auto correlation.

If there is a dependence on x, then always need to do loop over timesteps!!

Cox-ingersoll:

The issue with uhlenbeck’s model is that values can be negative, hence its not ideal to use this logic- where better approach is to use this model where they use square root of X. If X gets closer to 0, then noise term becomes 0. Get closer to 0, diffusion becomes 0 meaning aligns closer to the drift time so mu term. Guarantee does not become negative values.

If you want to add a lower boundary which is not 0, you need to do:

dX=alpha(mu-X)dt+sigma\*sqrt(X-c)\*dW

where c is the lower boundary and should align overtime to that value defined.