FMO6 — Web: https://tinyurl.com/ycaloqk6 Polls: https://pollev.com/johnarmstron561

Lecture 11 - Improvements and Revision

Dr John Armstrong

King's College London

August 22, 2020

What's coming up

Last Week:

- Instructions + video on submitting coursework.
- I have posted solutions to the May 2016 exam.

Today:

- Improving numerical methods
- Time to complete survey
- Revision

Improving numerical methods

We'll discuss

- Richardson Extrapolation which can be used to improve many numerical methods
- Four methods of improving Monte Carlo methods:
  - Antithetic sampling (which we've seen already)
  - Importance sampling
  - Control variate method.
  - Quasi Monte Carlo

Richardson Extrapolation

- Suppose we have a numerical method to compute y^* using some function y depending on a parameter ϵ.

$$y^* = \lim_{\epsilon \to 0} y(\epsilon)$$

- Suppose moreover we know that

$$y(\epsilon) = y^* + C\epsilon^n + O(\epsilon^{n+1})$$

  for some constant C.
- By comparing two estimates for y^* and taking a linear combination we can get the ϵ^n term to cancel giving us a new method with faster convergence:
-

$$\begin{aligned} y^k(\epsilon) &:= \frac{k^n y(\epsilon) - y(k\epsilon)}{k^n - 1} \\ &= y^* + O(\epsilon^{n+1}) \end{aligned}$$

Proof.

$$
\begin{aligned}
y(\epsilon) &= y^* + C(\epsilon^n) + O(\epsilon^{n+1}) \\
y(k\epsilon) &= y^* + C((k\epsilon)^n) + O(\epsilon^{n+1})
\end{aligned}
$$

Subtracting k^n copies of the first equation to one copy of the second

$$
\begin{aligned}
k^n y(\epsilon) - y(k\epsilon) &= (k^n - 1)y^* + O(\epsilon^{n+1}) \\
\frac{k^n y(\epsilon) - y(k\epsilon)}{k^n - 1} &= y^* + O(\epsilon^{n+1})
\end{aligned}
$$

$\square$

Example

- We wish to compute an integral by the trapezium rule. $f : [a, b] \longrightarrow \mathbb{R}$.
- Define $\epsilon = (b - a)/N$ where N is the number of steps.
- Compute estimate for the integral I_1 using N steps, $\epsilon_1 = (b - a)/N$.
- Compute estimate for the integral I_2 using $2N$ steps, $\epsilon_2 = \frac{1}{2}\epsilon_1$.
- So take $k = \epsilon_2/\epsilon_1 = \frac{1}{2}$ in Richardson method.
- Trapezium rule converges at rate $n = 2$.
- New estimate is:

$$I_R = \frac{(\frac{1}{2}^2 I_1 - I_2)}{\frac{1}{2}^2 - 1} = -\frac{1}{3}I_1 + \frac{4}{3}I_2$$

MATLAB implementation of Richardson Extrapolation

```
% Perform intergration by the trapezium rule then apply richardson
% extrapolation to obtain estimates with improved convergence
function richardsonEstimate = integrateRichardson( f, a, b, N )
h1 = (b-a)/N;
h2 = (b-a)/(2*N);
estimate1 = integrateTrapezium( f,a,b,N);
estimate2 = integrateTrapezium( f,a,b,2*N);
k = h2/h1;
n = 2;
richardsonEstimate = (k^n*estimate1 - estimate2)/(k^n-1);

end
```

Interpretation

$$I_R = -\frac{1}{3}I_1 + \frac{4}{3}I_2$$

I_2 is computed using trapezium rule with $2N$ steps

$$I_2 = \frac{h}{2}(f(x_0)+2f(x_1)+2f(x_2)+\ldots+2f(x_{N-2})+2f(x_{N-1})+f(x_N))$$

I_1 is computed using N steps

$$I_1 = \frac{h}{2}(f(x_0) \qquad\qquad +2f(x_2)+\ldots+2f(x_{N-2}) \qquad\qquad +f(x_N))$$

So I_R is given by:

$$I_R = \frac{h}{3}(f(x_0)+4f(x_1)+2f(x_2)+\ldots+2f(x_{N-2})+4f(x_{N-1})+f(x_N))$$

So Richardson extrapolation in this example is equivalent to Simpson's rule.

Importance of Richardson's rule

- Richardson's rule can be applied to many numerical methods e.g. integration methods and finite difference methods.
- You can generalize it to cancel higher orders, or simply apply it more than once.
- Unfortunately it cannot be used for Monte Carlo pricing as the error term is not a *constant* multiple of $\epsilon^{\frac{1}{2}}$

Revision: Antithetic Sampling

- Suppose we have a Monte Carlo pricer based on drawing n normally distributed random numbers ϵ_i
- It is often better to compute the price using a sample based on ϵ_i and $-\epsilon_i$ rather than to use a sequence of $2n$ independent random variables.
- Theory: If X_1 and X_2 are random variables with $E(X_1) = E(X_2)$ then

$$E(X_1) = E\left(\frac{X_1 + X_2}{2}\right)$$

But

$$\text{Var}\left(\frac{X_1 + X_2}{2}\right) = \frac{1}{4}(\text{Var}(X_1) + \text{Var}(X_2) + 2\,\text{Cov}(X_1, X_2))$$

Let X_1 be estimate based on the n variables ϵ_i. Let X_2 be estimate based on $-\epsilon_i$. We will often have $\text{Cov}(X_1, X_2)$ is negative.

MATLAB implementation of Antithetic sampling

```
% Price a call option by antithetic sampling
function [price,errorEstimate] = callAntithetic( K,T, ...
                                  S0,r,sigma, ...
                                  nPaths )

logS0 = log(S0);
epsilon1 = randn( nPaths/2,1 );
epsilon2 = -epsilon1;
logST1 = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon1;
logST2 = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon2;
ST1 = exp( logST1 );
ST2 = exp( logST2 );
discountedPayoffs1 = exp(-r*T)*max(ST1-K,0);
discountedPayoffs2 = exp(-r*T)*max(ST2-K,0);
price = mean(0.5*(discountedPayoffs1+discountedPayoffs2));
errorEstimate = std(0.5*(discountedPayoffs1+discountedPayoffs2))/sqrt(nPaths/2)
end
```

Antithetic Sampling Results

- Parameters: $S0 = 100$, $K = 100$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $N = 10000$
- Results:

Method	Price	Standard error estimate
Black–Scholes Formula	3.0679	
Naive Monte Carlo	3.0794	0.197
Antithetic Sampling	3.0771	0.054

- Conclusion: Antithetic sampling is easy to implement and often rather effective.

Importance Sampling

- Monte Carlo pricing is an integration method.
- You can use substitution to change one integral to another integral by re-parameterizing
- Equivalently you can change the distribution from which you draw your samples so long as apply appropriate weights to correct for this.
- Monte Carlo integration is exact when the price function is constant
- If we can re-parameterize so the price function is nearer to being constant, we will have reduced the variance of the Monte Carlo algorithm.

Importance Sampling Example

- Suppose we want to price a far out of the money knock out call option
- Suppose that for 99% of price paths the option will end out of the money
- This means that 99% of price paths in the Monte Carlo calculation will give us no information.
- Instead: find a way to generate the 1% of price paths where the option ends up in the money; compute the expectation for these paths; re-weight by multiplying by 100.
- For simplicity, let's do this for a vanilla call option to see how it improves upon ordinary Monte Carlo.

Calculation

- Generate stocks prices at time T using the formula:
$$\log(S_T) = \log(S_0) + \left(r - \frac{1}{2}\sigma^2\right) T + \sigma\sqrt{T} N^{-1}(u)$$
where u is uniformly distributed on $[0, 1]$.
- Option is in the money only if $\log(S_T) \geq \log(K)$. Equivalently only if:
$$u \geq u_{\min} := N\left(\frac{\log(K) - \log(S_0) - (r - (1/2)\sigma^2) * T}{\sigma\sqrt{T}}\right)$$
- So only generate values u on the interval $[u_{\min}, 1]$, then multiply resulting expectation by $1 - u_{\min}$ to account for the fact that we have only generated $1 - u_{\min}$ of the possible samples.
- We know the other samples would have given 0 for the option payoff.

MATLAB implementation of Importance Sampling

```
function [price,error] = callImportance( K,T, ...
                          S0,r,sigma, ...
                          nPaths )

logS0 = log(S0);
% Generate random numbers u on the interval [lowestU,1]
lowestU = normcdf( (log(K)-logS0 - (r-0.5*sigma^2)*T)/(sigma*sqrt(T)) );
u = rand(nPaths,1)*(1-lowestU)+lowestU;

% Now generate stock paths using norminv( u ). lowestU was chosen
% so that the lowest possible stock price obtained is K. Note that
% we are only considering a certain proportion of possible stock prices
logST = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*norminv(u);
ST = exp( logST );
discountedPayoff = exp(-r*T)*(ST-K);

% Since we only simulate a certain proportion of prices, the true
% epectation of the final option value must be weighted by proportion
proportion = 1-lowestU;
price = mean(discountedPayoff)*proportion;
error = std(discountedPayoff)*proportion/sqrt(nPaths);
```

Importance Sampling Results

- Parameters: $S_0 = 100$, $K = 200$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $n = 1000$.
- Note that this is far out of the money, so naive Monte Carlo will perform badly.
- Results:

Method	Price	Standard Error
Black–Scholes Formula	0.02241	
Naive Monte Carlo	0.05960	0.03469
Importance Sampling	0.02122	0.00066

- Conclusions: Importance Sampling is more difficult to implement than antithetic sampling, but can produce excellent improvement for far out of the money options

The Control Variate Method - Idea

- Suppose that we wish to price a Knock Out option
- We have an analytic formula for the price of a Call Option with the same strike.
- Maybe, rather than pricing a Knock Out option directly, it would be a better idea to estimate the difference between the price of a Knock Out option and the price of the Call Option using Monte Carlo instead.

$$\begin{aligned}\text{Price of Knockout Option} \approx \; & \text{Price of Call Option} \\ & + \text{Estimate of difference} \qquad (1)\end{aligned}$$

- Because the difference is probably smaller than the price we're trying to estimate, the variability in a Monte Carlo estimate of the difference is probably lower than the variablility in a Monte Carlo estimate of the price.

Control Variate - Example that proves it can work

- Consider the extreme case of pricing a knock out option where the barrier is so high it will very rarely be hit.
- In the control variate method, we will estimate that the difference between the call price and the knock-out option price is zero even if we use a tiny sample (e.g. a sample of one).
- The control variate method will converge to the exact answer immediately.
- The naive method will be no more accurate than pricing a call by Monte Carlo, so only converges slowly.

The Control Variate method

- Suppose we have a random variable M with $E(M) = \mu$ and wish to find μ.
- Suppose we have another random variable T with $E(T) = \tau$ with τ known.
- Define $M^* = M + c(T - \tau)$. $E(M^*) = \mu$ too for any $c \in \mathbb{R}$. Our previous example was the special case when $c = -1$.
- $\operatorname{Var}(M^*) = \operatorname{Var}(M) + c^2 \operatorname{Var}(T) + 2c \operatorname{Cov}(M, T)$
- Choose c to minimize this
$$c = \frac{-\operatorname{Cov}(M, T)}{\operatorname{Var}(T, T)}$$
-
$$\operatorname{Var}(M^*) = (1 - \rho^2) \operatorname{Var}(M)$$
where ρ is the correlation between M and T.

Control Variate method, worked example

- Let us price a Call Option by Monte Carlo
- We expect the price of a Call Option to be correlated with the price of the stock, so let's use the stock price as our control variate.

```
function [price,errorEstimate, c] = callControlVariate( K,T, ...
                         S0,r,sigma, ...
                         nPaths, ...
                         c)

% Usual pricing code
logS0 = log(S0);
epsilon = randn( nPaths,1 );
logST = logS0 + (r-0.5*sigma^2)*T + sigma*sqrt(T)*epsilon;
ST = exp( logST );
discountedPayoffs = exp(-r*T)*max(ST-K,0);

% Standard formula for control variate method
m = discountedPayoffs;
t = exp(-r*T)*ST;
tau = S0;
covMatrix = cov(m,t);
if nargin<7
    c = -covMatrix(1,2)/covMatrix(2,2);
end
mStar = m + c*(t-tau);

% Result
price = mean(mStar);
errorEstimate = std(mStar)/sqrt(nPaths);
```

Control Variate Results

- Parameters: $S0 = 100$, $K = 100$, $\sigma = 0.2$, $r = 0.14$, $T = 1$, $n = 1000$.
- Results:

Method	Result	Standard Error
Black–Scholes Formula	15.721	
Naive Monte Carlo	16.263	0.564
Control variate	15.723	0.137

- Note, to compute the error I fixed c and then re-ran to compute the same error as I was concerned using the same data to find c and estimate error may lead to bias.
- Conclusions: The control variate technique is easy to implement. It can produce significant improvements in the Monte Carlo price.

Low Discrepancy Sequences

- In one dimension, evenly distributed sample points give better performance than Monte Carlo sampling (i.e. the rectangle rule is faster than Monte Carlo).
- In high dimensions there are better sets of sample points available.
- Given an N and a dimension d, you can generate a "low discrepancy sequence" of N points in $[0,1]^d$ so that if you wish to estimate a function $f : [0,1]^d \longrightarrow \mathbb{R}$ you will get a better estimate by sampling at the points in the low discrepancy sequence than you would be Monte Carlo.
- Using a low discrepancy sequence rather than pseudo-random numbers is called *quasi-Monte Carlo*. Low discrepancy sequences are also called quasi-random numbers.
- Quasi Monte Carlo converges faster than Monte Carlo — but you have to be careful: you can only guarantee better results as N tends to infinity and you want good results for low N.
- See Joshi "More Mathematical Finance" (or many other sources) for details on how to use low discrepancy sequences in practice.

Generating Low discrepancy sequencey in MATLAB

```
d = 2;
s = sobolset(d);
points = net(s,1000);
```

- You can replace `sobolset` with `haltonset` for another low discrepancy sequence.
- Try plotting a `scatter` plot of the results and comparing with the halton version and genuine random numbers.
- Aside: this is an example of object oriented MATLAB code. We are creating a complex data object and then calling special functions to work with that kind of data object

Summary of improvements to numerical methods

- With little effort you can use techniques such as Richardson extrapolation to improve the convergence of a numerical method.
- There are various “variance reduction” techniques available for Monte Carlo. If you need to speed up your Monte Carlo pricer why not try all of them at once?

Feedback

- Please fill in the online feedback for the module. There are separate paper forms to rate your classes.

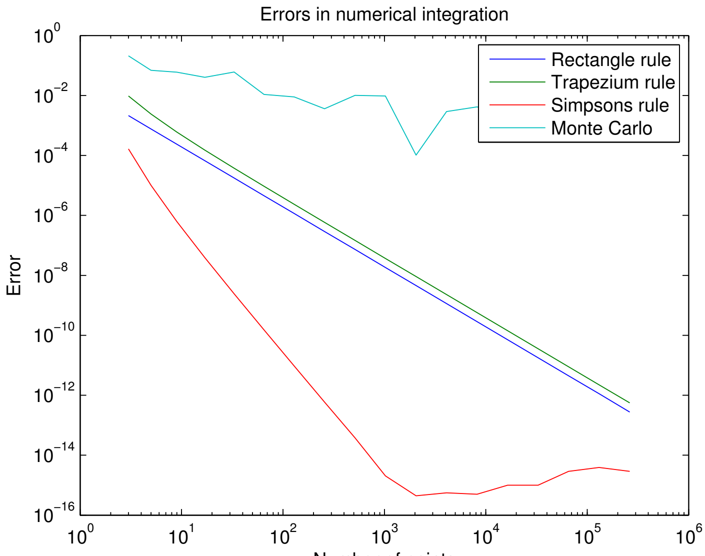
Question: Numerical intergration

- How many integration rules can you name?
- What are their formulae and rate of convergence?
- Can you draw a log-log plot indicating their convergence?

Revision: Numerical integration

- h = (b-a)/n;
- Rectangle rule:
$$x_i = a + \left(i - \frac{1}{2}\right) h$$
$$\int_a^b f = h\left(f(x_1) + f(x_2) + \ldots + f(x_n)\right)$$
Error $O(h^2)$.
- Trapezium rule:
$$x_i = a + ih$$
$$\int_a^b f = \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + \ldots 2f(x_{n-1}) + f(x_n)\right)$$
Error $O(h^2)$.
- Simpson's rule. n is even:
$$x_i = a + ih$$
$$\int_a^b f = \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots 4f(x_{n-1}) + f(x_n)\right)$$
Error $O(h^4)$.

Plot of errors for integration rules



Question: Numerical intergration

- What has numerical integration got to do with pricing derivatives?
- What has numerical integration got to do with Monte Carlo pricing?

Revision: Relevance of numerical integration

- Risk neutral prices are just expectations in the risk neutral measure
- If we $q_T(S)$ is the probability density of S_T in the $\mathbb{Q}$ measure,

$$\text{price} = e^{-rT} \int_{\mathbb{R}} q_T(S) \text{payoff}(S) \, \mathrm{d}S$$

  for options whose payoff only depends on S_T. $q_T(S)$ is the *pricing kernel*.
- You can perform 1 dimensional integrals by Monte Carlo.
  - Chose random points x on the interval $[a, b]$
  - Compute the average of $f(x)$ and multiply by $(b - a)$.

  Convergence $O(N^{-\frac{1}{2}})$.
- For high dimensional integrals, Monte Carlo is the only practical choice because minimum number of sample points required for other integration rules grows exponentially with dimension.

Question: Simulating random variables

- What is the recipe for converting a stochastic differential equation into its Euler approximation?

Revision: Simulating random variables

- To simulate a stochastic differential equation, first write down its Euler approximation:
  - Replace $\mathrm{d}t$ with δt.
  - Replace $\mathrm{d}X_t$ with δX.
  - Replace $\mathrm{d}W_t^{(i)}$ with $\delta W_t^{(i)}$.
- Example:

$$\mathrm{d}s = \left(\mu - \frac{1}{2}\sigma^2\right)\mathrm{d}t + \sigma \mathrm{d}W_t$$

becomes:

$$s_{i+1} = s_i + \left(\mu - \frac{1}{2}\sigma^2\right)\delta t + \sigma \delta W_t$$

- Now replace δW_t with $\sqrt{t}\epsilon_t^{(i)}$ with standard normally distributed ϵ.

$$\mathrm{d}s = \left(\mu - \frac{1}{2}\sigma^2\right)\mathrm{d}t + \sigma\sqrt{\delta t}\epsilon_t$$

- The Euler scheme is exact for Brownian motion. When simulating the log of the stock price, you can use just one time step if desired.

Revision: generateBSPaths

```
% Generate random price paths according to the black scholes model
% from time 0 to time T. There should be nSteps in the path and
% nPaths different paths
function [ S, times ] = generateBSPaths( ...
    T, S0, mu, sigma,nPaths, nSteps  )

dt = T/nSteps;
logS0 = log( S0);
eps = randn( nPaths, nSteps );
dlogS = (mu-0.5*sigma^2)*dt + sigma*sqrt(dt)*eps;
logS = logS0 + cumsum( dlogS, 2);
S = exp(logS);
times = dt:dt:T;

end
```

Revision: Correlated random variables

- A pseudo square root of a positive definite symmetric matrix A is a matrix B with BB^T.
- The Cholesky decomposition of A is the unique lower triangular pseudo-square root, L, with positive diagonal. $A = LL^T$.
- If x is a vector of independent $N(0,1)$ variables, then Lx is multivariate normal with mean 0 and covariance matrix A.

Question: Simulations

- What can we usefully do with our simulations?

Revision: Using simulations

We can use our simulations for:

- Pricing
- Testing strategies
- Optimization
- Risk management

Question: Monte Carlo Pricing

- How do you use Monte Carlo methods to price an option?
- Give one way of computing the delta by Monte Carlo?
- What kinds of option can you / can't you price by Monte Carlo?
- How fast is Monte Carlo?

Revision: Monte Carlo Pricing

- Generate paths in the *risk neutral measure*
- The discounted sample mean is an estimate for the price
- The sample standard deviation divided by the square root of the number of paths is an estimate for the error
- Use the same random numbers if estimating the delta by comparing two nearby Monte Carlo prices.
- Seeding the random number generator is one way to do this.
- You can price (discrete time) Knock Out, Knock In and Asian options.
- You can price European path-independent options, though 1-d integration is faster.
- You can't price American options.
- Convergence is of order $O(N^{-\frac{1}{2}})$

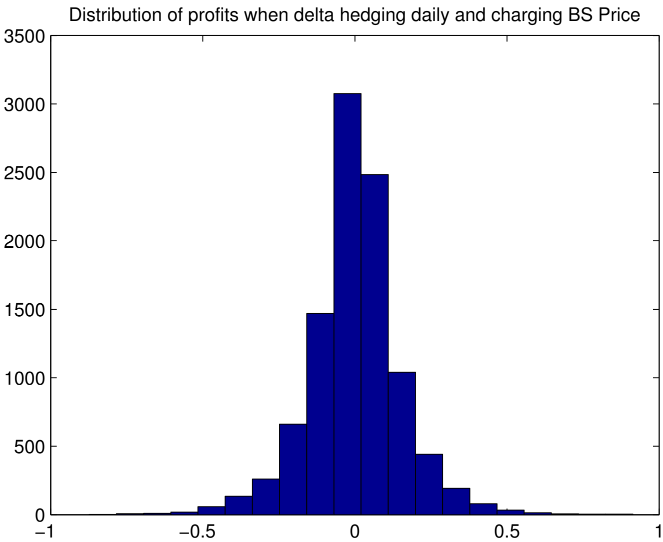
Question: Testing strategies

- If a trader decides to write a call option and then delta hedge to ensure they can fulfil their obligation what is their bank balance at each time?

Revision: Delta hedging

- b_t is bank balance at time t.
- P is amount charged.
- $$b_0 = P - \Delta_0 S_0$$
- $$b_t = e^{r\delta t} b_{t-1} - (\Delta_t - \Delta_{t-1}) S_t$$
- $$b_n = e^{r\delta t} b_{n-1} + (\Delta_{n-1}) S_n - \max\{S_n - K, 0\}$$
- Standard deviation of final bank balance is of order $(\delta t)^{\frac{1}{2}}$.
- To simulate delta hedging simulate $\mathbb{P}$-measure stock prices and then use the above equations to simulate the bank balance of a delta hedger.

Delta hedging results



Question: Optimization

- What is meant by a utility function? Give an example.
- How can you find good strategies by combining other strategies?
- What is the indifference price?

Revision: Optimization

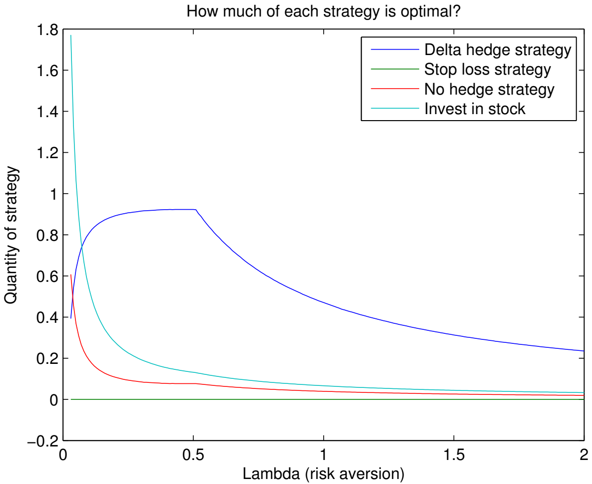
- A utility function is a mapping from $\mathbb{R}$ to $\mathbb{R}$ that ascribes a subjective value to any particular payoff. Utility is usually increasing and concave.
- We wish to maximize expected utility.
- Given n strategies $S_1, S_2, \ldots S_n$ we can form a linear combination $\sum_i \alpha_i S_i$.
- Generate M scenarios. Let p_i^m be the payoff of strategy i in scenario m
- Use `fmincon` to minimize the expected disutility:

$$-\frac{1}{M}\sum_m u(\sum_i \alpha_i p_i^m)$$

.

- The indifference price for a financial product P and strategy S is the amount you could pay P so that your expected utility when following strategy S remains the same whether or not you buy P.
- The indifference price for P is the infimum of the indifference prices over all strategies.

Revision: $u(x) = 1 - e^{-\lambda x}$



Question: Risk Management

- Name three methods of computing/approximating V@R
- What's good/bad about each?

Revision: V@R

- Monte Carlo VaR
  - Good Points: Accurate answers assuming model is correct
  - Bad Points: Slow, choice of model is subjective
- Parametric VaR
  - Good Points: Fast
  - Bad Points: Innaccurate for nonlinear products. Choice of model is subjective.
- Historic VaR
  - Good Points: Not subjective
  - Bad Points: Limited by availability of data and claim that future is same as past.

Revision: VaR implementation details

- For Monte Carlo V@R and parameteric V@R you need to estimate model parameters
  - Choice of drift is not so important e.g. choose so that *logS* is driftless.
  - Take EWMA for volatility:

$$\overline{\sigma}^2 = 365 \times \frac{1-\lambda}{1-\lambda^{m+1}} \sum_{j=0}^{m} \lambda^j r_{i-j}^2$$

    r contains daily log returns.
  - Use smaller λ for shorter time horizons.
- For historic V@R use historic data to find series of historic daily log returns. Scale by $\sqrt{n}$ to generate n-day returns. Otherwise same as Monte Carlo.

Revision: Parametric V@R

- If P^a are the risk factors
- $p^{(a)} = \log(P^{(a)})$
- $\Sigma = \mathrm{cov}(\mathrm{p})$ is the covariance matrix
- d is number of days
- V is the security we're trying to calculate VaR for. Define

$$\delta^{(a)} = P^{(a)} \frac{\partial V}{\partial P^{(a)}}$$

- Parametric V@R is

$$\mathrm{VaR} \approx N^{-1}\left(\frac{p}{100}\right)\sqrt{\frac{d\delta^T \Sigma \delta}{365}}$$

VaR versus CVaR etc.

- Good: VaR is easy to understand, VaR estimates are easy to back test. Banks have already implemented VaR systems.
- Bad: VaR is not sub-additive, hence not a coherent risk measure.
- Good: CVar is a coherent risk measure. It is convex so good for optimizations.
- Bad: CVar is hared to calculate than VaR. It is harder to test.
- Bad for both: may lead to herd behaviour, false sense of security.

Question: Finite difference methods

- What can you price with finite difference methods?
- What can't you price with finite difference methods? (as taught in this course)

Revision: finite difference methods

- Approximate PDE with finite difference method and work backwards in time from final payoff to current price.
- Precise method depends on choice of PDE and choice of stencil.
- Black Scholes PDE

$$V_t + \frac{1}{2}\sigma^2 S^2 V_{SS} + rSV_S - rV =$$

- Negative time heat equation

$$W_t = -\frac{1}{2}\sigma^2 W_{xx}$$

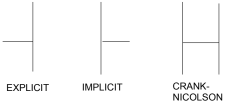
-

$$W = e^{-rt} V$$

$$x = -(r - \frac{1}{2}\sigma^2)t + \log(S)$$

Revision: stencils

- Forward difference
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$
- Backward difference
$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$
- Central difference
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$
- Second derivative
$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$



Revision: Simplest case of finite difference

- Take the heat equation and use the explicit scheme.

$$W_{i-1,j} = \lambda W_{i,j+1} + (1-2\lambda)W_{i,j} + \lambda W_{i,j-1}$$

$$\lambda = \frac{1}{2}\sigma^2 \frac{\delta t}{\delta x^2}$$

- Only stable if $(1-2\lambda) > 0$. Unstable means that small changes in W due to rounding errors result in wildly changing values in W in earlier time steps.
- Interpretation: only stable if we can see this as a trinomial tree with λ, $(1-2\lambda)$ and λ being probability of moving up, staying same or moving down.

Revision: Boundary conditions

- For a call option, on top boundary call option is well approximated by a portfolio of:
  - one stock (value at time t is S_t)
  - obligation to pay strike at (value at time t is $e^{-r(T-t)}K$)

  Hence $V \approx S_t - e^{-r(T-t)}K$ on top boundary.
- On bottom boundary, call option is well approximated by 0.
- What are boundary conditions for put? What about when using the heat equation?

Models other than Black Scholes

- Heston model and jump diffusion are two models we have seen
- Use Euler method to simulate Heston
- Calibrate $\mathbb{Q}$ measure model to smile by using `fmincon` to minimize mean square distance.
- You can hedge exotics by hedging using underlying and options.

Summary

- That's not the whole course, but it's a lot of it on only a few slides.
- This is how you should revise: write super-condensed summary notes, then learn them.

School's Out!

- Don't forget PTES and module feedback!
- Good luck!

School's Out!