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

Lecture 9

Dr John Armstrong

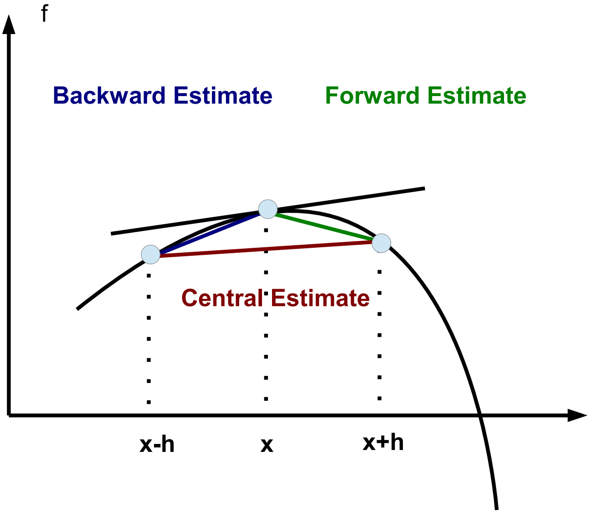
King's College London

August 22, 2020

Estimating derivatives

- There are many formulae for estimating derivatives numerically.
- For the first derivative alone we have
  - 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
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$
  - Higher order estimates, e.g.
$$f'(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} + O(h^4)$$

Graphical representation



Remarks

- The forward difference and backward difference are only accurate to first order.
- The central difference is accurate to second order (essentially because the formula is exact for quadratics)
- You can create schemes with arbitrary convergence if f is sufficiently smooth and you are willing to perform sufficiently many evaluations of f.

Other finite difference schemes

- We wish to solve:

$$\frac{\partial W}{\partial t} = -\frac{\sigma^2}{2}\frac{\partial^2 W}{\partial x^2}$$

  with final boundary condition given at time T and appropriate boundary conditions for large and small W.
- For the *explicit* method we took the backwards estimate for the time derivative (and used the simplest estimate for the second derivative term)
- For the *implicit* method, take the forward estimate for the time derivative
- For the *Crank-Nicolson* method use a central estimate.

Explicit and implicit difference equations

Recall we have discretized the t and x coordinates so $W_{i,j}$ is the value of W at time point i and space point j. We have N time steps and M space steps.

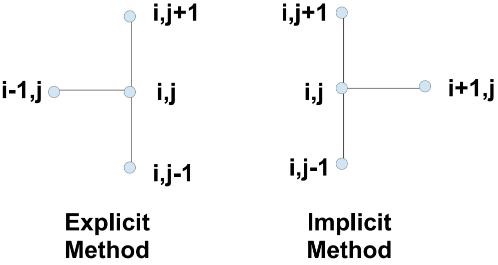
- Explicit method: take the backward difference in time and the simplest estimate in x.

$$\frac{W_{i,j} - W_{i-1,j}}{\delta t} = -\frac{\sigma^2}{2}\left(\frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2}\right)$$

- Implicit method: take the forward difference in time and the simplest estimate in x.

$$\frac{W_{i+1,j} - W_{i,j}}{\delta t} = -\frac{\sigma^2}{2}\left(\frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2}\right)$$

Stencils



These pictures are called *stencils*. They summarize how we use the values of W to estimate the various derivatives.

Implicit method

To price a call option using the implicit method for the heat equation, we have the following conditions:

- A difference equation

$$\frac{W_{i+1,j} - W_{i,j}}{\delta t} = -\frac{\sigma^2}{2}\left(\frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2}\right)$$

- Boundary conditions:

$$W_{i,j_{\min}} = 0$$

$$W_{i,j_{\max}} = e^{-\frac{1}{2}\sigma^2 t_i + x_{j_{\max}}} - e^{rT}K$$

$$W_{i_{\max},j} = \max\{e^{-\frac{1}{2}\sigma^2 T + x_j} - e^{rT}K, 0\}$$

- Note: we calculated the boundary conditions last week when we transformed the Black–Scholes PDE to the heat equation

Remarks

- The *explicit* method gave us a formula for $W_{i-1,j}$ in terms of the value of W at time i.
- The *implicit* method gives us $M+1$ linear equations in the $M+1$ unknowns $W_{i-1,j}$ in terms of the values of W at time i.
- (Recall we have N time steps, M space steps and so $N+1$ time points and $M+1$ space points)
- We can solve these linear equations to compute the values at time $i-1$.
- The method is called *implicit* because we don't get an explicit formula for $W_{i,j}$, instead we calculated $W_{i,j}$ as the value implied by a set of simultaneous equations.

Solving the linear equations

- To solve linear equations in MATLAB one writes them in matrix form $Ax = b$.
- The solution is then given by `x = A \ b`. i.e. we divide both sides by `A` on the left".
- Our difference equation is

$$\frac{W_{i+1,j} - W_{i,j}}{\delta t} = -\frac{\sigma^2}{2}\left(\frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2}\right)$$

- Rewriting:

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

where:

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

The simultaneous equations

- For $j \in \{j_{\min+1}, j_{\min+2}, \ldots, j_{\max-1}$ we have

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

- Boundary conditions

$$W_{i,j_{\min}} = \mathrm{bottom}_i = 0$$
$$W_{i,j_{\max}} = \mathrm{top}_i = e^{-\frac{1}{2}\sigma^2 t_i + x_{j_{\max}}} - e^{rT}$$

- This gives a total of $M+1$ linear equations in M unknowns.

Matrix form

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -\lambda & 1+2\lambda & -\lambda & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & -\lambda & 1+2\lambda & -\lambda & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & -\lambda & 1+2\lambda & -\lambda & \dots & 0 & 0 & 0 \\ \vdots & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & -\lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & 1+2\lambda & -\lambda & 0 \\ 0 & 0 & 0 & 0 & 0 & \dots & -\lambda & 1+2\lambda & -\lambda \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} W_{i,j_{\min}} \\ W_{i,j_{\min}+1} \\ W_{i,j_{\min}+2} \\ W_{i,j_{\min}+3} \\ W_{i,j_{\min}+4} \\ \vdots \\ W_{i,j_{\max}-2} \\ W_{i,j_{\max}-1} \\ W_{i,j_{\max}} \end{pmatrix} = \begin{pmatrix} \mathrm{bottom}_i \\ W_{i+1,j_{\min}+1} \\ W_{i+1,j_{\min}+2} \\ W_{i+1,j_{\min}+3} \\ \vdots \\ W_{i+1,j_{\max}-3} \\ W_{i+1,j_{\max}-2} \\ W_{i+1,j_{\max}-1} \\ \mathrm{top}_i \end{pmatrix}$$

- We call this large tri-diagonal matrix A.
- We write a MATLAB helper function `createTridiagonal` which creates a tridiagonal matrix given three vectors containing the three non-zero diagonals.

MATLAB implementation

First we initialize variables such as the vectors x and t precisely as we did for the explicit method.

```
x0 = log( S0 );
xMin = x0 - nSds*sqrt(T)*sigma;
xMax = x0 + nSds*sqrt(T)*sigma;

dt = T/N;
dx = (xMax-xMin)/M;

iMin = 1;
iMax = N+1;
jMin = 1;
jMax = M+1;
x = (xMin:dx:xMax)';
t = (0:dt:T);

lambda = 0.5*sigma^2 * dt/(dx)^2;
```

The changed code

```
currW=max(exp(-0.5*sigma^2*T + x) - exp(-(r*T))*K,0);
A = createTridiagonal( [0 ; -lambda*ones(M-1,1) ; 0], ...
                       [1 ; (1+2*lambda)*ones(M-1,1) ; 1], ...
                       [0 ; -lambda*ones(M-1,1) ; 0] );

bottom = zeros(N+1,1);
top=exp(-0.5*sigma^2 * t + x(jMax))- exp(-r*T)*K;

for i=iMax-1:-1:iMin
    vector= [ bottom(i); currW((jMin+1):(jMax-1)); top(i) ];
    currW= A \ vector;
end

price = currW(jMin+M/2);
```

- `currW` stores the value of `W` at time point i, we do not need to store the entire matrix of values for `W`
- Note that writing `[a; b; c]` concatenates matrices vertically
- Writing `[a b c]` concatenates matrices horizontally.

Advantages of the implicit method

- Suppose we fix λ. Choosing δx then determines δt.
- The implicit scheme is stable irrespective of λ
- The explicit scheme is stable only if $(1 - 2\lambda) > 0$.
- The error of the implicit scheme is $O(\delta t)$ just as is the explicit scheme.
- For the explicit scheme, for moderately δx you are forced to have a tiny value for δt to ensure stability.
- For the implicit scheme we can choose δx and δt independently. So we can get good answers with a comparatively small number of time steps.

Solving the linear equations

- To implement the implicit scheme, we need to solve a linear equation
$$Aw = v$$
where A is a symmetric, tri-diagonal matrix.
- If we wrote a general-purpose linear equation solver using Gaussian elimination this would not take advantage of the simple form.
- Let us see how to solve the equations efficiently

Gaussian elimination by hand

A tridiagonal system of equations can be written:

$$
\begin{array}{llllll}
b_1 x_1 & + c_1 x_2 & & & & = d_1 \quad [1] \\
a_2 x_1 & + b_2 x_2 & + c_2 x_3 & & & = d_2 \quad [2] \\
 & + a_3 x_2 & + b_3 x_3 & + c_3 x_4 & & = d_3 \quad [3] \\
 & & + a_4 x_3 & + b_4 x_4 & + c_4 x_5 & = d_4 [4] \\
\ldots
\end{array}
$$

Take b_1 times equation [2] and subtract a_2 times equation [1] x_1. This gives the new equation:

$$(b_1 b_2 - c_1 a_2)x_2 + b_1 c_2 x_3 = b_1 d_2 - a_2 d_1$$

This equation together with equations $[3], [4], \ldots$ gives a new tridiagonal system in $x_2, x_3, \ldots x_n$.

Thomas algorithm

- 1 dimensional tridiagonal problems are trivial to solve. $x_1 = d_1/b_1$.
- Assume for induction that we have developed the Thomas algorithm for problems of dimension n.
- For dimension $n+1$ use the previous slide to find a tridiagonal system in x_2, x_3, ..., x_n
- Solve this system by the Thomas algorithm (we can do so by induction)
- Now use the equation

$$b_1 x_1 + c_1 x_2 = d_1$$

to solve for x_1.
- Therefore we can solve a tridiagonal system of equations with only $O(n)$ multiplication and addition operations.
- A naive implementation of Gaussian elimination will take $O(n^3)$ steps

Getting MATLAB to use the Thomas algorithm

- We'd like MATLAB to use the Thomas algorithm
  - One option is to implement it ourselves
  - Another option is to use MATLAB's built in support for the algorithm
- MATLAB will automatically use the Thomas algorithm to solve $Ax = b$ if it detects that A is tri-diagonal.
- In general checking if an arbitrary matrix is tri-diagonal will take $O(n^2)$ steps so we need to give MATLAB a hint.

Sparse matrices

- A *sparse matrix* is a matrix where most of the entries are zero.
- To store a sparse matrix it is more efficient to store a list of the rows and columns that are non-zero and the values at those rows and columns than to store a large block of memory most of which is zero.
- In general, the linear algebra algorithms one should use for sparse matrices are very different from the ones one uses with full (i.e. non-sparse) matrices.
- We can create a sparse matrix in MATLAB using the command `sparse`.
- When you solve the problem $Ax = b$ in MATLAB with A a sparse matrix, it will automatically check to see whether using the Thomas algorithm is the best approach.

Creating a sparse matrix in MATLAB

- Suppose that a matrix A has non zero entries a_{r_i,c_i} where r_1, r_2, $\ldots r_n$ and c_1, c_2, $\ldots c_n$ are some sequences of indices.
  - Create a vector `rows` containing $r_1, r_2, \ldots, r_n$.
  - Create a vector `columns` containing $c_1, c_2, \ldots, c_n$.
  - Create a vector `values` containing $a_{r_1,c_1}, a_{r_2,c_2}, \ldots, a_{r_n,c_n}$.
  - Create a spare matrix A using the command

```
A = sparse( rows , columns , values );
```

- In general MATLAB tries to intelligently select the best available algorithm, therefore you should always use a sparse matrix to store matrices which are mostly zero so that MATLAB has a hint as to how to proceed.

The createTridiagonal function

```
%CREATETRIDIAGONAL Create a sparse tri-diagonal matrix contianing
%  the given upper, diagonal and lower entries.
%  Each of these should be a vector of length N, the first entry of
%  lower should be zero, the last entry of upper should be zero.
function A= createTridiagonal( lower, diagonal, upper )

N = length( diagonal );

rowsUpper = (1:N-1)';
colsUpper = (2:N)';
rowsDiagonal = (1:N)';
colsDiagonal = (1:N)';
rowsLower = (2:N)';
colsLower = (1:N-1)';
allRows = [rowsUpper ; rowsDiagonal ; rowsLower ];
allCols = [colsUpper ; colsDiagonal ; colsLower ];
allVals = [ upper(rowsUpper) ; diagonal ; lower(rowsLower)];

A = sparse( allRows, allCols, allVals );

end
```

The solveTridiagonal function

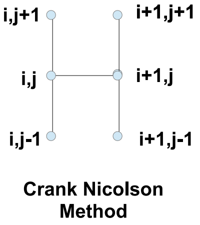
```
function [ x ] = solveTridiagonal( a,b,c,d )

if (length(a)==1)
    x = d(1)/(b(1));
else
    nextB = b(2:end);
    nextB(1) = b(1)*b(2)-c(1)*a(2);
    nextC = c(2:end);
    nextC(1) = b(1)*c(2);
    nextD = d(2:end);
    nextD(1) = b(1)*d(2)-d(1)*a(2);

    xRemainder = solveTridiagonal(a(2:end),nextB,nextC,nextD);

    x1 = (d(1)-c(1)*xRemainder(1))/b(1);
    x = [x1 ; xRemainder];
end
end
```

Recursion

- You can write functions in MATLAB that call themselves
- Writing functions in this way is called recursion
- This gives an easy implementation of `solveTridiagonal` that matches are inductive definition.
- Its not written as efficiently as it could be because we keep creating new vectors unnecessarily
- It isn't hard to replace the recursion with a for loop if preferred to get a fully efficient implementation. There isn't much point in running through the details since we can use sparse matrices to achieve the same result.

The Crank-Nicolson method

For the Crank-Nicolson method one uses the stencil:



$$\frac{\partial W}{\partial t} \approx \frac{W_{i+1,j} - W_{i,j}}{\delta t}$$

$$\frac{\partial^2 W}{\partial x^2} \approx \frac{1}{2} \times \frac{W_{i+1,j+1} - 2W_{i+1,j} + W_{i+1,j-1}}{\delta x^2} + \frac{1}{2} \times \frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2}$$

Crank-Nicolson difference equations

- The Crank-Nicolson method uses the average of the estimates for the second derivative at times i and $i+1$.
- Just as for the implicit method, when we include boundary conditions, at each time i we will get a system of $M+1$ equations in the $M+1$ unknowns W_i, j in terms of the values of W at time $i+1$.
- For j not at the boundary.

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

- Once again this is a tridiagonal system.

Benefits of Crank-Nicolson scheme

- It is always stable irrespective of choice of λ
- Convergence is $O(\delta t^2)$.
- It is an exercise for you to implement this method.

Pricing American options by the implicit method

- One of the main selling points of the explicit finite difference method is that we can use it to price American options.
- We have just seen how the implicit and Crank-Nicolson methods can be used to improve the stability and convergence of finite difference methods.
- Can these techniques be applied to improve the pricing of American options?

Recap

- To price an American option A by the explicit method, one assumes we can compute the price at time $i+1$.
- We can then use the explicit method to compute the expected value of a new option $\tilde{A}_i$ at time i that is not-exercisable at time i but can be exercises at any time from $i+1$ onwards.
- The price of the American option is then estimated as the maximum of the immediate exercise price and the price of option $\tilde{A}_i$.
- We can now proceed to time $i-1$.
- Notice that this argument uses expectations and financial logic: we haven't actually derived it from the Black Scholes PDE. It is really a "tree pricing" algorithm rather than a PDE algorithm.

What PDE does an American put option satisfy

- An American option does not obey the Black Scholes PDE at times when early exercise is optimal. At these points it satisfies:

$$V = K - S$$

$$\frac{\partial V}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV < 0$$

- At times when early exercise is not optimal, it obeys the Black Scholes PDE and also the condition

$$V > K - S$$

$$\frac{\partial V}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0$$

Boundary conditions

- At the boundary between the two regions, V and the delta are both continuous

$$V = \max\{K - S, 0\}$$

$$\frac{\partial V}{\partial S} = -1$$

- This is called a free boundary problem.
- We haven't proved that these differential inequalities hold, but you can convince yourself using a no-arbitrage argument. Since they are differential inequalities you will need to use continuous time stochastic calculus to prove things rigorously.

Complimentarity problem

- The following inequalities hold everywhere

$$V - K + S \geq 0$$

$$-\frac{\partial V}{\partial t} - \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} - rS \frac{\partial V}{\partial S} + rV \geq 0$$

Moreover we must have equality for at least one condition.

- The condition that $x \geq 0$ and $y \geq 0$ and one of x and y vanishes can be written as $x \geq 0$, $y \geq 0$ and $xy = 0$.

Differential inequalities for American options

- So we have that at all times

$$V - K + S \geq 0$$

$$-\frac{\partial V}{\partial t} - \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} - rS \frac{\partial V}{\partial S} + rV \geq 0$$

and

$$(V - K + S) \left(\frac{\partial V}{\partial t} + \frac{\sigma^2}{2} S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV \right) = 0$$

- We can now find discrete approximations to these inequalities using our choice of stencil and attempt to solve associated discrete problems.
- It then seems reasonable to hope that this will lead to a finite difference scheme for pricing American options with convergence properties similar to those seen for European options.

Moving to the heat equation

- Define $W = e^{-rt}V$ is the discounted price.
  $x = -(r - \frac{\sigma^2}{2})t + \log(S)$ as usual.
- Boundary conditions are exactly the same as for a pricing a European put by the heat equation:
  - Top boundary condition: $W(t, x_{\max}) = 0$
  - Bottom boundary condition: $W(t, x_{\min}) = e^{-rt}(K - S(x_{\min}))$
  - Final boundary condition: $W(t, x_{\max}) = E(t, x)$.

Transformed differential inequalities

The equations transform to:

$$\left(\frac{\partial W}{\partial t}+\frac{\sigma^2}{2}\frac{\partial^2 W}{\partial x^2}\right)(W-E(t,x))=0$$

$$-\frac{\partial W}{\partial t}-\frac{\sigma^2}{2}\frac{\partial^2 W}{\partial x^2}\geq 0$$

$$W-E(t,x)\geq 0$$

Here $E(t,x)=e^{-rt}\max\{K-S(x),0\}$ is the discounted early exercise price.

Discretize

Let's discretize using the implicit scheme, but you could use Crank Nicolson too.

$$\left(-\left(\frac{W_{i+1,j}-W_{i,j}}{\delta t}\right)-\frac{\sigma^2}{2}\left(\frac{W_{i,j+1}-2W_{i,j}+W_{i,j-1}}{\delta x^2}\right)\right)(W_{i,j}-E_{i,j})=0$$

$$\left(-\left(\frac{W_{i+1,j}-W_{i,j}}{\delta t}\right)-\frac{\sigma^2}{2}\left(\frac{W_{i,j+1}-2W_{i,j}+W_{i,j-1}}{\delta x^2}\right)\right)\geq 0$$

$$W_{i,j}-E_{i,j}\geq 0$$

How on earth do you solve such a system of inequalities?

Linear complimentarity problem

- The linear complementarity problem is the problem of solving

$$x.y = 0$$

$$x \geq 0$$

$$y \geq 0$$

$$Ax = b + y$$

For vectors x and y given a vector b and a matrix A. We'll assume that A is symmetric and positive definite.

- It is called "linear" because the last condition is linear
- It is called "complementarity" because x and y are complimentary vectors: for each index j either x_j or y_j is zero.

Rewriting

At time i take x to be the vector with components

$$x_j = W_{i,j} - E_{i,j}$$

x to be the vector

$$y_j = \left(- \left(\frac{W_{i+1,j} - W_{i,j}}{\delta t} \right) - \frac{\sigma^2}{2} \left(\frac{W_{i,j+1} - 2W_{i,j} + W_{i,j-1}}{\delta x^2} \right) \right)$$

So the equations earlier imply $xy = 0$ and $x \geq 0$, $y \geq 0$.
But these expressions for x and y are not independent as they both involve the same unknowns $W_{i,j}$. This establishes a linear relation between x and y of the form $y = Ax + b$

Identifying A and b

- Write W_i for the vector with components $W_{i,j}$. Similarly E_i.
- The definition of x tells us that $W_i = x + E_i$.
- The definition of y tells us that $y = -W_{i+1} + AW_i$ for an appropriate A (which will in fact be the tri-diagonal matrix found in the European case).
- Hence $y = -W_{i+1} + A(x + E_i) = Ax + (AE_i - W_{i+1})$
- Define $b = AE_i - W_{i+1}$ and we have shown $Ax = b$.
- Therefore x and y are solutions of a linear complimentarity problem.

Remarks

- The matrix A is the same tridiagonal matrix that occurred in the implicit method for European options.
- The formulae I've explicitly written only apply for j away from the boundary — as for European options, we have a 1 in the top left and the bottom right of A to account for the boundary conditions.

Initialization

```
x0 = log( S0 );
xMin = x0 - nSds*sqrt(T)*sigma;
xMax = x0 + nSds*sqrt(T)*sigma;

dt = T/N;
dx = (xMax-xMin)/M;

iMin = 1;
iMax = N+1;
jMin = 1;
jMax = M+1;
x = (xMin:dx:xMax)';
t = (0:dt:T);

lambda = 0.5*sigma^2 * dt/(dx)^2;
```

Boundary conditions

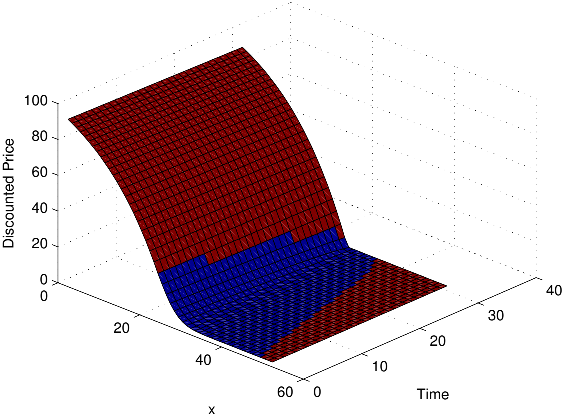
```
% Use boundary condition to create vector currW
currW=max(exp(-r*T)*K-exp(-0.5 *sigma^2 * T + x),0);
A = createTridiagonal( [0 ; -lambda*ones(M-1,1) ; 0], ...
                       [1 ; (1+2*lambda)*ones(M-1,1) ; 1], ...
                       [0 ; -lambda*ones(M-1,1) ; 0] );

bottom = exp(-r*T)*K- exp(-0.5*sigma^2 * t + x(jMin));
top=zeros(1,N+1);
```

Iteration

```
exercised = zeros(N+1,M+1);
W = zeros(N+1, M+1);
W(iMax,:)=currW;
for i=iMax-1:-1:iMin
    wIPlus1 = [ bottom(i); currW((jMin+1):(jMax-1)); top(i) ];
    if (american)
        % e = immediate exercise value
        e = max(exp(-r*t(i))*K-exp(-0.5 *sigma^2 * t(i) + x),0);
        b = A*e - wIPlus1;
        omega = 1.5;
        [xSol,ySol] = solveLCP(A, b, wIPlus1, omega, 10 );
        currW =  xSol + e;
        exercised(i,:)=currW<=(e);
    else
        currW = A \ wIPlus1;
    end
    W(i,:)=currW;
end
```

Results

Red region is where early exercise has taken place. (Note graph is given in terms of x not S.)



How to solve the linear complimentarity problem

- The short answer is lookup in the literature how this can be solved numerically
- We'll give a run-through of the ideas that lead to the standard numerical solution used to price American options:
  - Solving the equation $Ax = b$ iteratively.
    - Jacobi method
    - Gauss–Seidel method
    - Successive over relaxation (SOR))
  - Solving the linear complimentarity problem by SOR.

Jacobi method

- The Jacobi method is a numerical method for solving the equation $Ax = b$.
- Let us write $A = D + R$

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}$$

- D diagonal part
- R remainder

Idea

- D is easy to invert because it is diagonal.
- $(D+R)x = b$ implies $Dx = b - Rx$ which implies $x = D^{-1}(b - Rx)$.
- Pick an initial guess x_0.
- Define sequence $x_n = D^{-1}(b - Rx_{n-1})$.
- If this converges to a limit it will satisfy the equation $Ax = b$.

Recursion

- Consider the sequence
  - x_0
  - $x_1 = D^{-1}b - D^{-1}Rx_0$
  - $x_2 = D^{-1}b - D^{-1}RD^{-1}b + D^{-1}RD^{-1}Rx_0$
  - $x_3 = D^{-1}b - D^{-1}RD^{-1}b + D^{-1}RD^{-1}b - D^{-1}RD^{-1}RD^{-1}Rx_0$
  - ...
- So long as the spectral radius of $D^{-1}R$ is less than 1, this will converge.
- If the matrix is strictly diagonally dominant — i.e.

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \quad \forall i$$

the sequence will converge

Applications

- For sparse matrices we can perform the multiplication by $D^{-1}(R)$ reasonably quickly due to sparseness.
- The convergence of contractions is rapid, so we will only need a few iterations to get a good estimate.
- If we have a good guess for the initial value it will be more rapid still.
- Thus for diagonally dominant sparse matrices where we have a good idea of the initial value the Jacobi method will perform well.
- Example: for appropriate choices of λ the matrix in the implicit method for European options is diagonally dominant. We have a good first guess for the price vector at time i, it is presumably close to the price vector at time $i + 1$.

Gauss-Seidel method

- The Jacobi method is a numerical method for solving the equation $Ax = b$.
- Let us write $A = L_* + U$

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

- L_* lower triangular part
- U strictly upper triangular part

Algorithm

- Again, L_* is easy to invert because it is lower triangular.
- A solution to $Ax = b$ satisfies $x = L_*^{-1}(b - Ux)$.
- Pick initial guess $x^{(0)}$ and define $x^{(n)} = L_*^{-1}(b - Ux^{(n-1)})$
- Use the fact that L is lower triangular to write down the following relationship:

$$x_i^{(n+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(n+1)} - \sum_{j>i} a_{ij} x_j^{(n)} \right)$$

- This formula contains $x_j^{(n+1)}$ terms on both sides, but only terms for $j < i$ on the right. So long as we proceed by calculating in the order $i = 1, 2, \ldots, n$ this will give an explicit formula for x_i.

When does this converge

Gauss-Seidel converges:

- If A is symmetric and positive definite
- If A is strictly diagonally dominant

(It may converge under other circumstances too)

Successive over relaxation

- Write $A = L + D + U$ where L is strictly lower triangular, D is diagonal and U is upper triangular.
- $Ax = b$ can be rewritten:

$$(D + \omega L)\mathbf{x} = \omega \mathbf{b} - [\omega U + (\omega - 1)D]\mathbf{x}$$

- ω is some choice of parameter called the "relaxation" factor.
- It is a mash-up of Jacobi method and Gauss-Seidel method.
- It converges if A is positive definite and $0 < \omega < 2$.
- Hope is that for some $\omega > 1$ convergence should speed up, we won't discuss how to choose a good value of ω.

Motivation

- If we have a recursive system $x_{n+1} = f(x_n)$
- System $x'_{n+1} = (1-\omega)x'_n + \omega f(x'_n)$ gives another process which, if they both converge will have the same limit.
- Low values of ω slow rate of change of x_n (in limiting case $\omega = 0$, the sequence remains constant).
- High values of ω increase rate of change, so may speed convergence (or may cause oscillations or convergence to breakdown if ω is too high).

Explicit formulae for SOR

- Because $(D + \omega L)$ is lower triangular we can use forward substitution to write down explicit formulae as for Guass-Seidel

$$x_i^{(n+1)} = (1-\omega)x_i^{(n)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij} x_j^{(n+1)} - \sum_{j>i} a_{ij} x_j^{(n)} \right)$$

- Note this formula fits the general pattern given on the previous slide.
- You can use this to solve the linear equations that occur when pricing European options using the implicit or Crank-Nicolson schemes.

Solving the linear complimentarity problem

- The linear complimentarity problem is to solve

$$y = Ax + b, \quad x \geq 0, \quad y \geq 0, \quad x.y = 0$$

  for vectors x, and y.
- Note that in the special case when we have a solution with $y = 0$ this reduces to $Ax = -b$ and $y = 0$ everywhere.
- For example when applied to pricing American options $y = 0$ is saying that the Black–Scholes PDE is satisfied everywhere. The equations $Ax = -b$ are then just the equations that occur in pricing a European option.
- Idea: perhaps if we take an iterative method for solving $Ax = -b$ but at each stage we insist that $x \geq 0$ we will get a solution to the linear complimentarity problem?

Solving linear complimentarity by successive over-relaxation

- To solve $x \geq 0, y \geq 0, xy = 0, y = Ax + b$
- Take an initial guess x^0
- Define $x^{(n)}$ by:
$$x_i^{(n+1)} = \max\left\{(1-\omega)x_i^{(n+1)} + \frac{\omega}{a_{ii}}\left(-b_i - \sum_{j<i} a_{ij}x_j^{(n+1)} - \sum_{j>i} a_{ij}x_j^{(n)}\right), 0\right\}$$
- So long as A is positive semi-definite and $0 < \omega < 2$ this converges. "The Solution of a Quadratic Programming Problem Using Systematic Overelaxation", C Cryer, 1971
- I note that he calls it "Systematic Overrelaxation" while everyone else calls it "Successive Overrelaxation" so presumably everyone finds the terminology a little odd!

Summary

- We have shown how to write American option pricing using differential inequalities
- This gives rise to a finite difference problem where each time step is a linear complimentarity problem.
- The linear complimentarity problem can be solved in practice using a successive over-relaxation technique.
- (Claim) this converges to the true American option price.
- Thus the finite difference method does give a good approach to American option pricing, but it does involve quite a few new ideas.
- Pricing American options using the implicit and Crank-Nicolson finite difference methods is therefore non-examinable. The explicit method IS examinable.