ME2610
Engineering Mathematics and Programming

10th November 2020

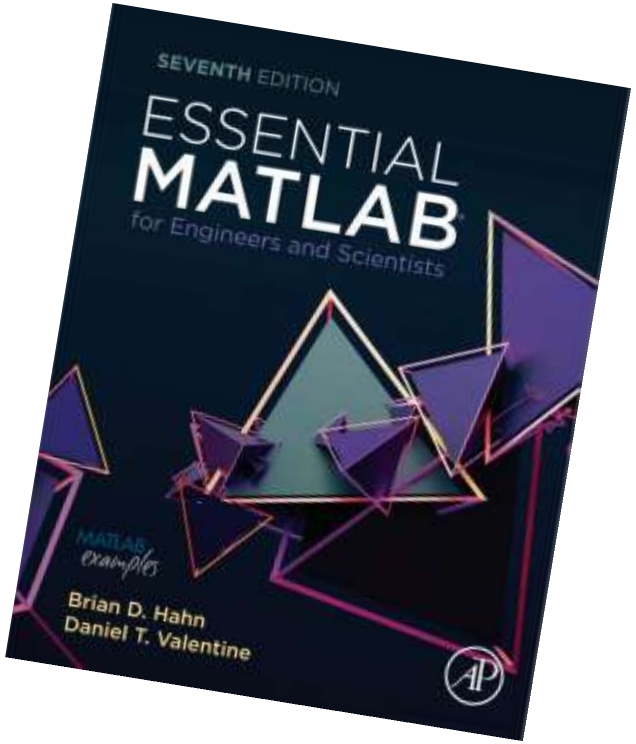
Dr Edward Smith

Room 105

Howell Building

Summary

- Brief recap of differential equation GRADER
- Second order differential equations
    - Initial value
    - Boundary value (iteration)
- Implicit vs Explicit solutions
- Partial Differential Equations



Essential Matlab (EM)
http://tinyurl.com/yy53shga

This session will be recorded

Learning Aims

- **LO2:** Understanding how to employ programming to solve basic engineering computational problems.
- **LO4:** Applying best-practice programming techniques to solve Mathematical models of Engineering problems.
- **LO5:** Understanding the usefulness of programming techniques in the process of solving Engineering problems.
- **LO6:** Presenting computational results in a clear and concise manner including validation and verification.

Registration and Questions

https://brunel.onlinesurveys.ac.uk/feedback_me2610

- Use the QR code to go to feedback
- You can ask questions or make comments at any time, either linked to your name (if you put it in) or anonymously (if you don't)

I'm finding the grader test 5 really challenging. I'm not sure on where to start and it doesn't make sense even though I have watched all relevant lectures

Wiseflow the ME2610 assignment does not have a brief

Is there another book that you could recommend other than essential MATLAB?

would like more examples when learning new Matlab language and concept. Could we have another book recommendation? The Essential Matlab doesn't go in-depth enough to help me out when I'm stuck on a question.

1. **Do the tutorial sheet!**
2. EM Section 12.3 – Solving Newton's law (initial value problem)
3. EM Section 14.4 – Differential Equations (14.4.2 bacteria/exponential case)
4. EM 14.6.2 – Lorentz Equations

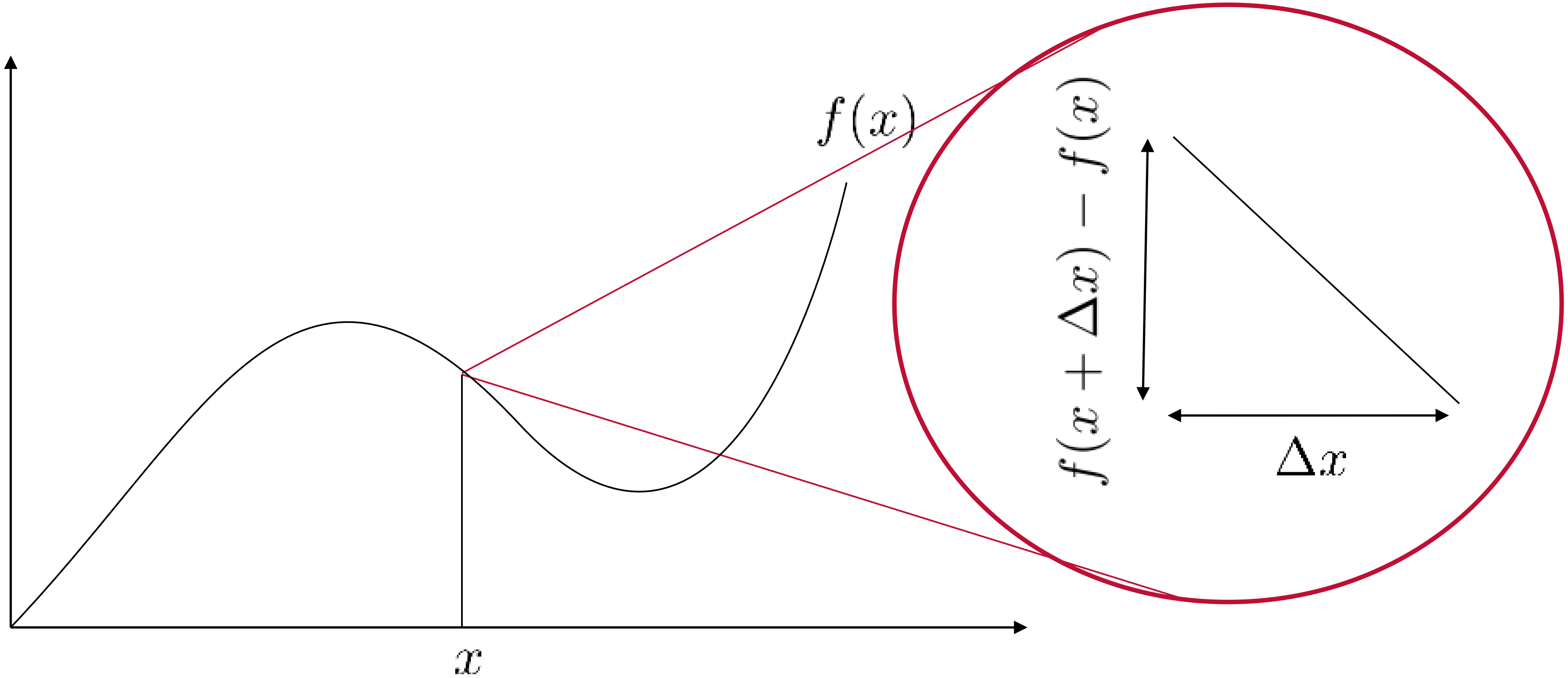
Plan for Course

Week	ecture N	ount M/	Lecture Content	Tutorial	Deadline	Date
1	1	1	Interpolation methods			29/09/2020
1	2	1	Introduction			29/09/2020
1	3	2	Interpolation methods			01/10/2020
1	4	2	Data types, matrices and arrays	TEST Matlab		01/10/2020
2	5	3	Interpolation methods	Basic arrays		06/10/2020
2	6	3	For and if statements	Matrices and simulatanous		06/10/2020
2	7	4	Root Finding	Interfaces and tests	↓	08/10/2020
2	8	4	Functions and Interfaces	For and if statements	**TEST Matlab**	08/10/2020
						15/10/2020
4	9	5	Root Finding	Interpolation		20/10/2020
4	10	5	Interpolation Numerics	Interpolation		20/10/2020
4	11	6	Root Finding	Root finding	↓	22/10/2020
4	12	6	Root Finding Numerics	Root finding	**Functions**	22/10/2020

Plan for Course

5	13	7	Integration Methods	Trapizum rule	|	27/10/2020
5	14	7	Integration Numerics	Simpson Rule	|	27/10/2020
5	15	8	Integration methods	Gauss integration	↓	29/10/2020
5	16	9	Gauss integration	Gauss integration	**Root/interpolation**	29/10/2020
6	17	10	Diff. equations (integrating factors, order)	Basic Finite difference	|	03/11/2020
6	18	8	Intro Finite Difference	Basic Finite difference	|	03/11/2020
6	19	11	Diff. equations (integrating factors, order)	Basic Finite difference	↓	05/11/2020
6	20	9	Explicit + 2nd order Finite Difference	Basic Finite difference	**Integration**	05/11/2020
7	21	12	Diff. equations (integrating factors, order)	1D ODE	|	10/11/2020
7	22	10	Implict Finite Difference	1D ODE	|	10/11/2020
7	23	13	2D unsteady convection from 1st principles	SIR Equation	↓	12/11/2020
7	24	11	2D Finite Difference	SIR Equation	**1D ODE**	12/11/2020
8	25	14	Vector functions/ Jacobian Newton-Raphson 2D	2D PDE	|	17/11/2020
8	26	12	Validation and Verification	2D PDE	|	17/11/2020
8	27	15	Vector functions/ Jacobian Newton-Raphson 2D	2D PDE	|	19/11/2020
8	28	16	Vector functions/ Jacobian Newton-Raphson 2D	2D PDE	|	19/11/2020
9	29	17	Laplace Transforms	Assignment help	|	24/11/2020
9	30	18	Laplace Transforms	Assignment help	|	24/11/2020
9	31	19	Laplace Transforms	Assignment help	↓	26/11/2020
9	32	20	Laplace Transforms	Assignment help	**Assignment 2D PDE**	26/11/2020

Definition of a Derivative



A derivative is just a placeholder for this (unreachable) $\Delta x \rightarrow 0$ limit

$$\frac{df}{dx} = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Differential Equations

- Equations which include derivatives are differential equations, e.g.

$$\frac{df}{dx} = 0 \qquad \frac{d^2 f}{dx^2} = 0 \qquad \frac{d^2 f}{dx^2} + f = 0$$

- These are the same as any other equation, for example the equation for a line or Newton's law

$$y = mx + c \qquad F = ma$$

- Which can also be written as differential equations

$$y = \frac{dy}{dx}x + c \qquad F = m\frac{d^2 r}{dt^2}$$

Differential Equations

- Equations which include derivatives are differential equations, e.g.

$$\frac{df}{dx} = 0 \qquad \frac{d^2 f}{dx^2} = 0 \qquad \frac{d^2 f}{dx^2} + f = 0 \qquad \frac{d^3 f}{dx^3} + \frac{df}{dx} + 4$$

- Order of equation is highest derivative, here 1, 2, 2 and 3
- Equations can be linear or non-linear. Roughly speaking, any equation which contains a product of unknown function or it's derivatives (here f) is non-linear, e.g.

$$f \frac{d^2 f}{dx^2} + x \frac{df}{dx} = 0 \qquad \frac{d^4 f}{dx^4} + f^2 = 0 \qquad \left(\frac{df}{dx} \right)^2 + x^2 = 0$$

Solutions to Differential Equations

- Some differential equations, especially if they are linear, can be solved exactly. For example:

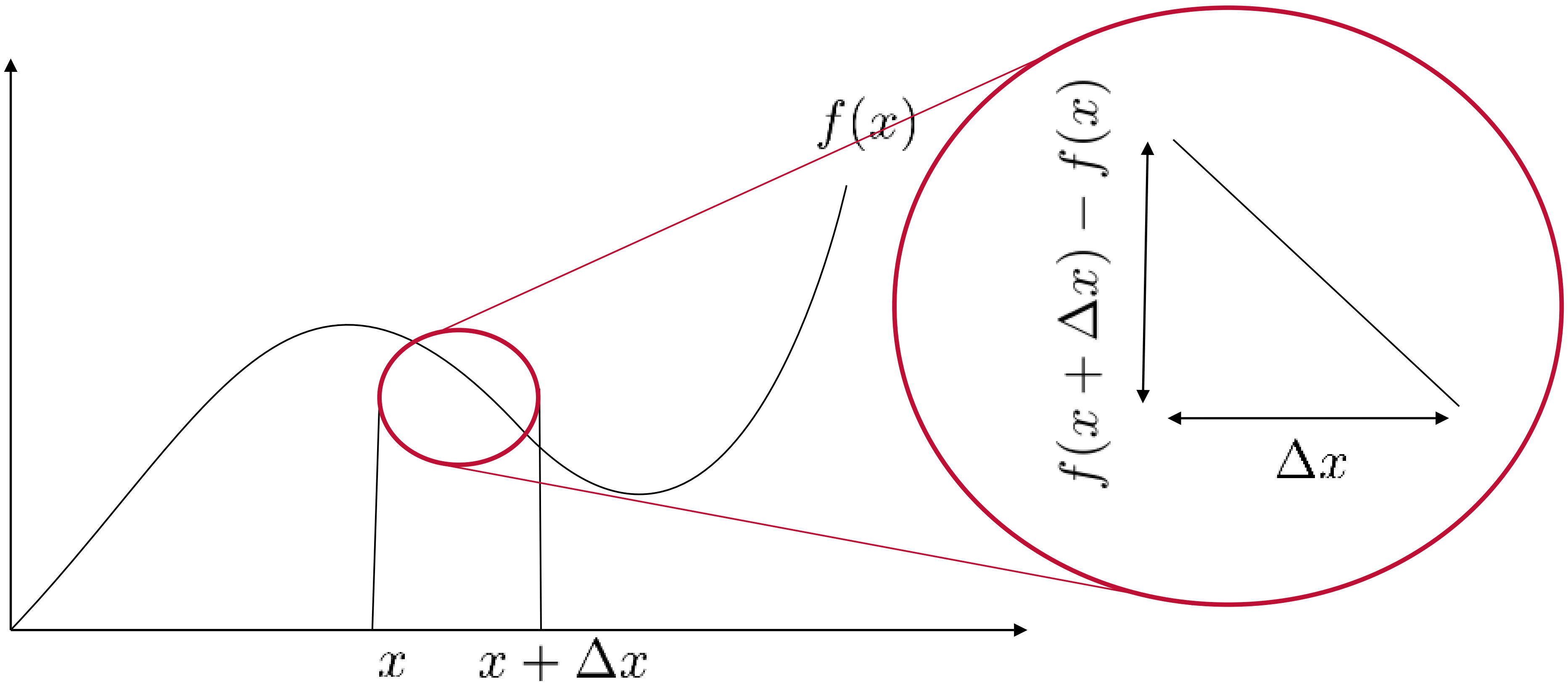
$$\frac{df}{dx} = a \qquad\qquad f(x) = ax + C_1$$

- This is integrated to give f as a function of x with an arbitrary constant of integration C_1. Similarly for second order equations,

$$\frac{d^2 f}{dx^2} = b \qquad\qquad \frac{df}{dx} = bx + C_2 \qquad\qquad f(x) = bx^2 + C_2 x + C_3$$

- You can solve with integrating factor, separation of variables, substitutions, etc, etc

Approximating a Derivative



So instead we approximate by not taking the limiting case

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

Schemes can have Different Accuracies (Taylor series)

- An expansion of a function about x+Δx

$$f(x+\Delta x) = f(x) + \frac{df}{dx}\Delta x + \frac{d^2 f}{dx^2}\frac{\Delta x^2}{2!} + \dots$$

neglecting >Δx^2 so error=$O(\Delta x^2)$

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

- An expansion of a function about x-Δx

$$f(x-\Delta x) = f(x) - \frac{df}{dx}\Delta x + \frac{d^2 f}{dx^2}\frac{\Delta x^2}{2!} - \dots$$

neglecting >Δx^2 so error=$O(\Delta x^2)$

$$\frac{df}{dx} \approx \frac{f(x) - f(x-\Delta x)}{\Delta x}$$

- Subtracting the two and rearranging

$$f(x+\Delta x) - f(x-\Delta x) = 2\frac{df}{dx}\Delta x + \dots$$

terms Δx^2 cancel so error=$O(\Delta x^3)$

$$\frac{df}{dx} = \frac{f(x+\Delta x) - f(x-\Delta x)}{2\Delta x} + \dots$$

Numerical Solutions to 1st and 2nd Order Terms

- First order derivatives

$$\frac{df}{dx} \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

- Second order derivatives

$$\frac{d^2 f}{dx^2} \approx \frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2}$$

- We introduce the same short-hand notation for both

$$\frac{f(x+\Delta x) - f(x)}{\Delta x} \equiv \frac{f_{i+1} - f_i}{\Delta x}$$

$$\frac{f(x+\Delta x) - 2f(x) + f(x-\Delta x)}{(\Delta x)^2} \equiv \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2}$$

Numerical Solutions to 1st and 2nd Order Terms

- First order derivatives

$$\frac{df}{dx} \approx \frac{f_{i+1} - f_i}{\Delta x}$$

- Second order derivatives

$$\frac{d^2 f}{dx^2} \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{(\Delta x)^2}$$

- We write as code in the same way (rearranged to get i+1 value)

$$\frac{df}{dx} = a$$

```
f(i+1) = f(i) + a*dx
```

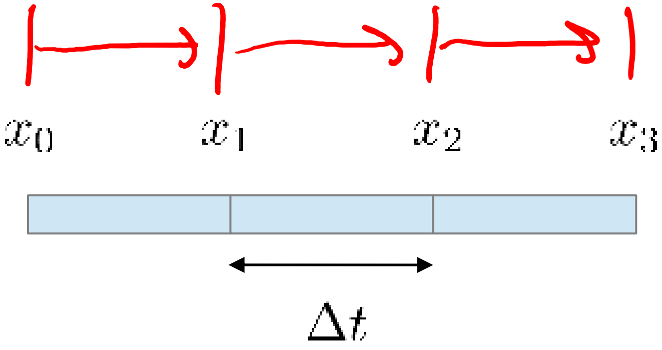
$$\frac{d^2 f}{dx^2} = b$$

```
f(i+1) = 2*f(i) - f(i-1) + b*dx^2
```

Numerical Solutions to 1st Order Equations

- First order derivatives $$\frac{df}{dx} \approx \frac{f_{i+1} - f_i}{\Delta x}$$
- We can write to get i+1 value from the previous, recall we worked out how a train moves in time step by step

$$\frac{dx}{dt} = v_0 \rightarrow \frac{x_{i+1} - x_i}{\Delta t} = V_0 \rightarrow x_{i+1} = x_i + \Delta t\, V_0$$



$$x_1 = x_0 + v_0 \Delta t \qquad x_1 = 0.5 + (0.05)(2)$$

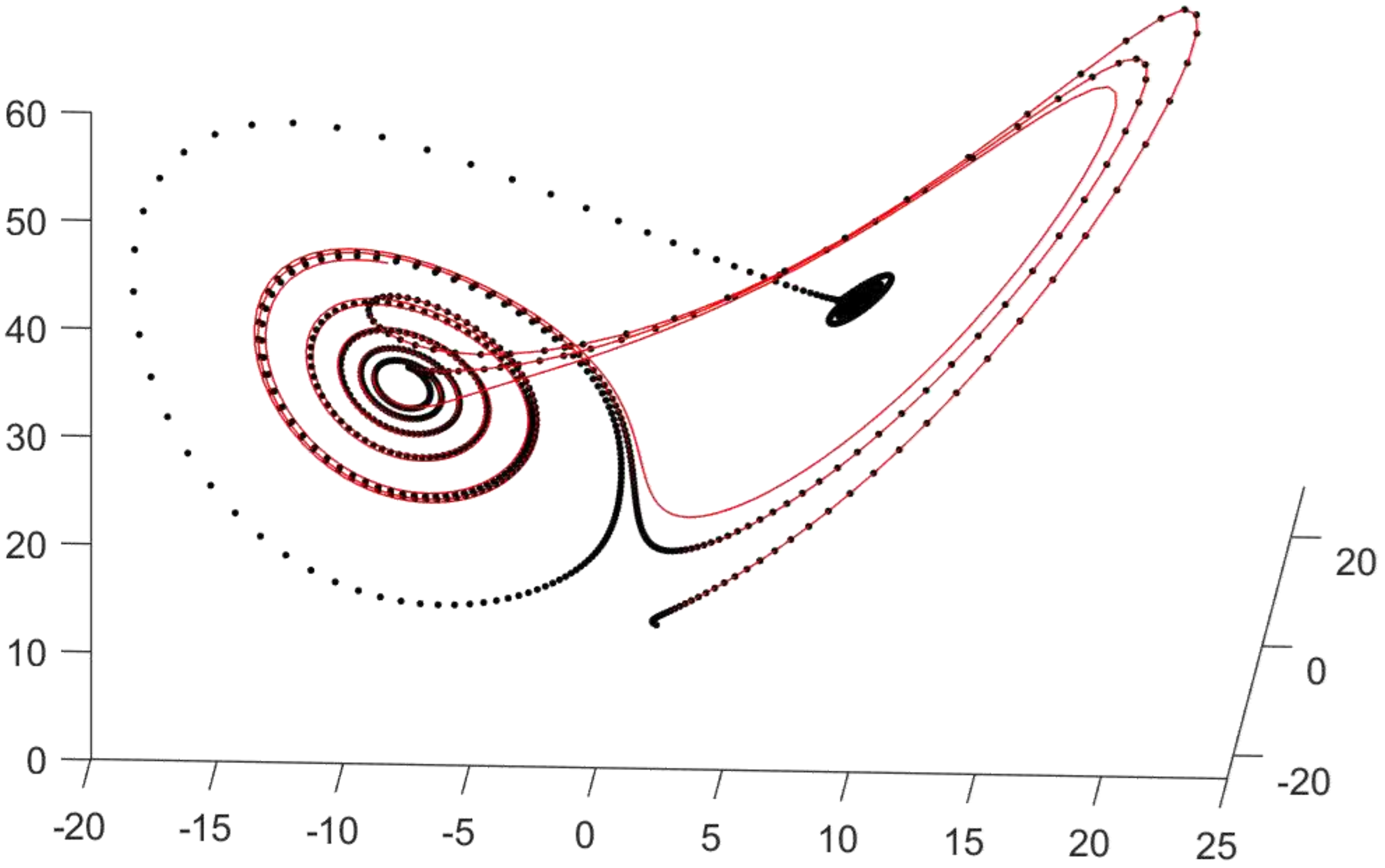
$$x_2 = x_1 + v_0 \Delta t \qquad x_2 = 0.6 + (0.05)(2)$$

$$x_3 = x_2 + v_0 \Delta t \qquad x_3 = 0.7 + (0.05)(2)$$

Lorentz Equations

- Coupled non-linear ordinary differential equations

$$\frac{dx}{dt} = 10(y - x),$$
$$\frac{dy}{dt} = -xz + 28x - y,$$
$$\frac{dz}{dt} = xy - 8z/3.$$



Susceptible Infected Recovered Model (SIR)

- Coupled first order ordinary differential equations

$$\frac{dS}{dt} = -\frac{\beta IS}{N},$$

```
S(i+1)=S(i)-dt*beta*I(i)*S(i)/N(i);
```

$$\frac{dI}{dt} = \frac{\beta IS}{N} - \gamma I,$$

```
I(i+1)=I(i)+dt*(beta*I(i)*S(i)/N(i)...
                - gamma*I(i));
```

$$\frac{dR}{dt} = \gamma I,$$

```
R(i+1) = R(i) + dt*gamma * I(i);
```

Susceptible Infected Recovered Model (SIR)

- Coupled first order ordinary differential equations

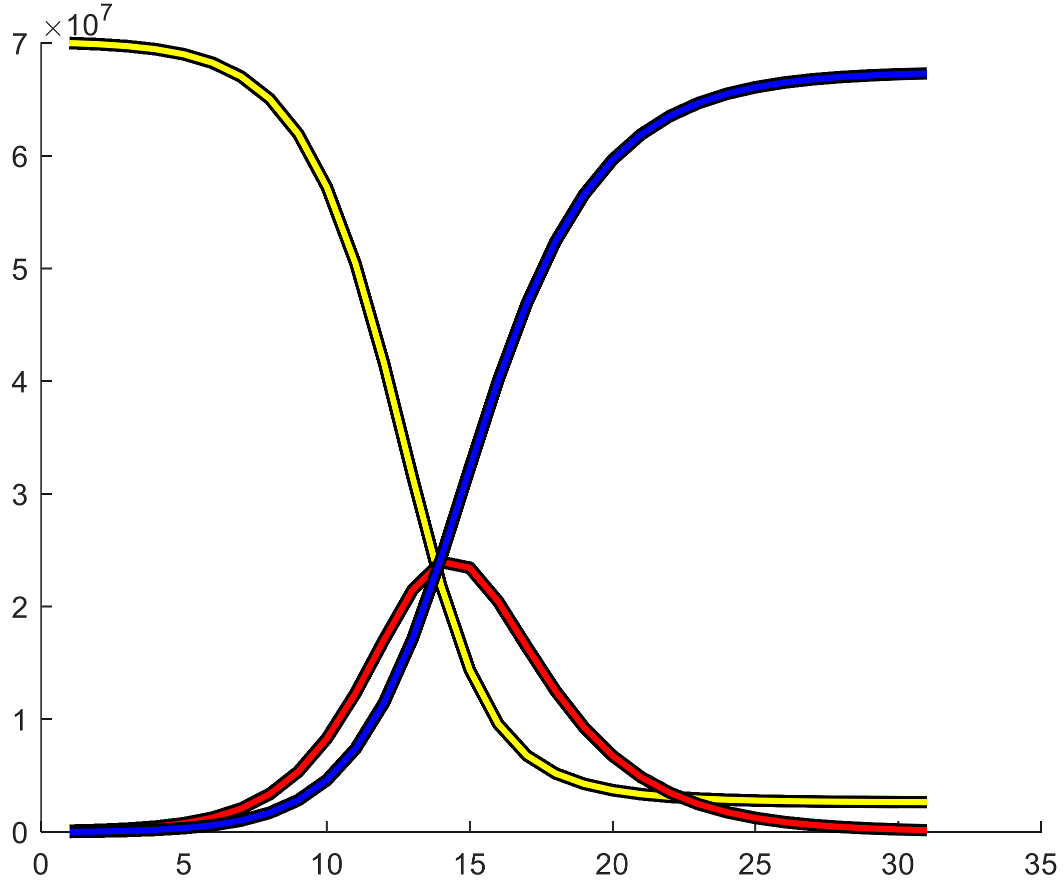
```
%Setup initial conditions
steps = 30;
population = 1000;
S(1) = population;
I(1) = 10;
R(1) = 0;

%? is the average number of
% contacts per person per time
beta = 1.0;

% Gamma is 1/infectious period
gamma = 1/3;

%Print reproduction rate
R0 = beta/gamma;
disp(["R0 = ", R0]);
```

```
%loop and solve equations
dt = 1.0;
for i=1:steps
    N(i) = S(i) + I(i) + R(i);
    S(i+1) = S(i) - dt*beta*I(i)*S(i)/N(i);
    I(i+1) = I(i) + dt*(beta*I(i)*S(i)/N(i) ...
                        - gamma*I(i));
    R(i+1) = R(i) + dt*gamma * I(i);
    plot(S, 'y-', "LineWidth", 4)
    hold all
    plot(I, 'r-', "LineWidth", 4)
    plot(R, 'b-', "LineWidth", 4)
    pause(0.001)
    hold off
end
```

Susceptible Infected Recovered Model (SIR)


```
steps = 30;
population = 70e6;
S(1) = population;
I(1) = 100000;
R(1) = 0;

%Timestep (days, weeks)
dt = 1.0;

%beta is the average number of
contacts per person per timestep
beta = 1.0;

% Gamma is one over infectious period
in timesteps
gamma = 1/3;

%reproduction rate
R0 = beta/gamma;
```

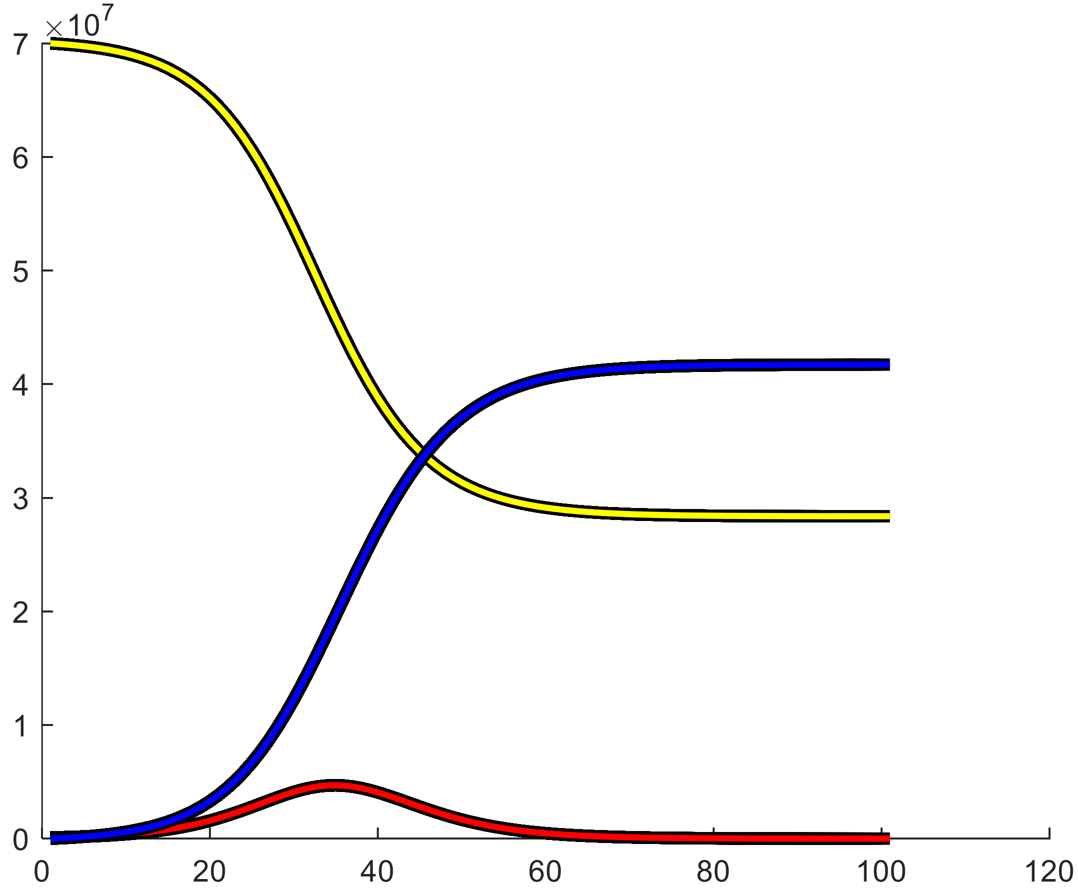
R0=3 is uncontrolled estimate for Covid



Susceptible Infected Recovered Model (SIR)


```
steps = 100;
population = 70e6;
S(1) = population;
I(1) = 100000;
R(1) = 0;

%Timestep (days, weeks)
dt = 1.0;

%beta is the average number of
contacts per person per timestep
beta = 0.5;

% Gamma is one over infectious period
in timesteps
gamma = 1/3;

%reproduction rate
R0 = beta/gamma;
```

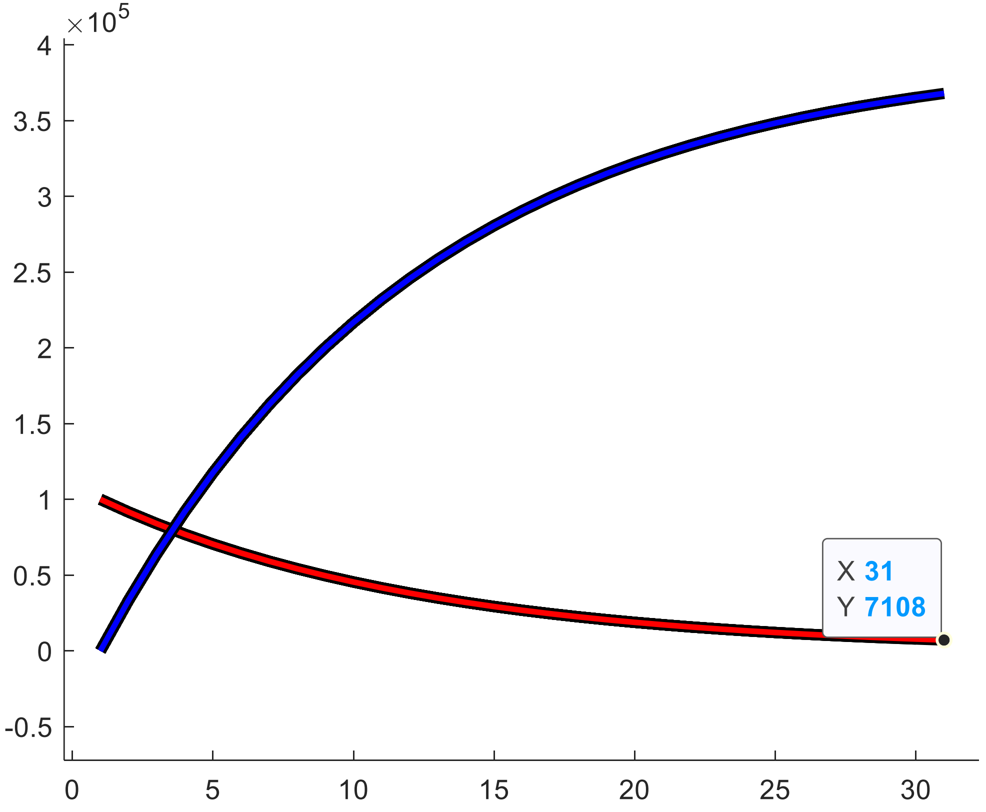
R0=1.5 is current estimate for Covid



Susceptible Infected Recovered Model (SIR)


```
steps = 30;
population = 70e6;
S(1) = population;
I(1) = 100000;
R(1) = 0;

%Timestep (days, weeks)
dt = 1.0;

%beta is the average number of
contacts per person per timestep
beta = 0.25;

% Gamma is one over infectious period
in timesteps
gamma = 1/3;

%reproduction rate
R0 = beta/gamma;
```

R0=0.75 is a lockdown estimate for Covid



Second Order - Initial Value Problem (Newton's laws)

- For an object with mass 70kg falling under gravity (g=9.81) with an initial velocity of v_0=2m/s and an initial position of x_0=100m, numerically integrate to get x=0 and work out the time this happens. Assume air resistance is negligible
  - So, Newton's law $$\frac{d^2x}{dt^2} = F$$
  - Discretisation $$\frac{d^2x}{dt^2} \approx \frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2}$$

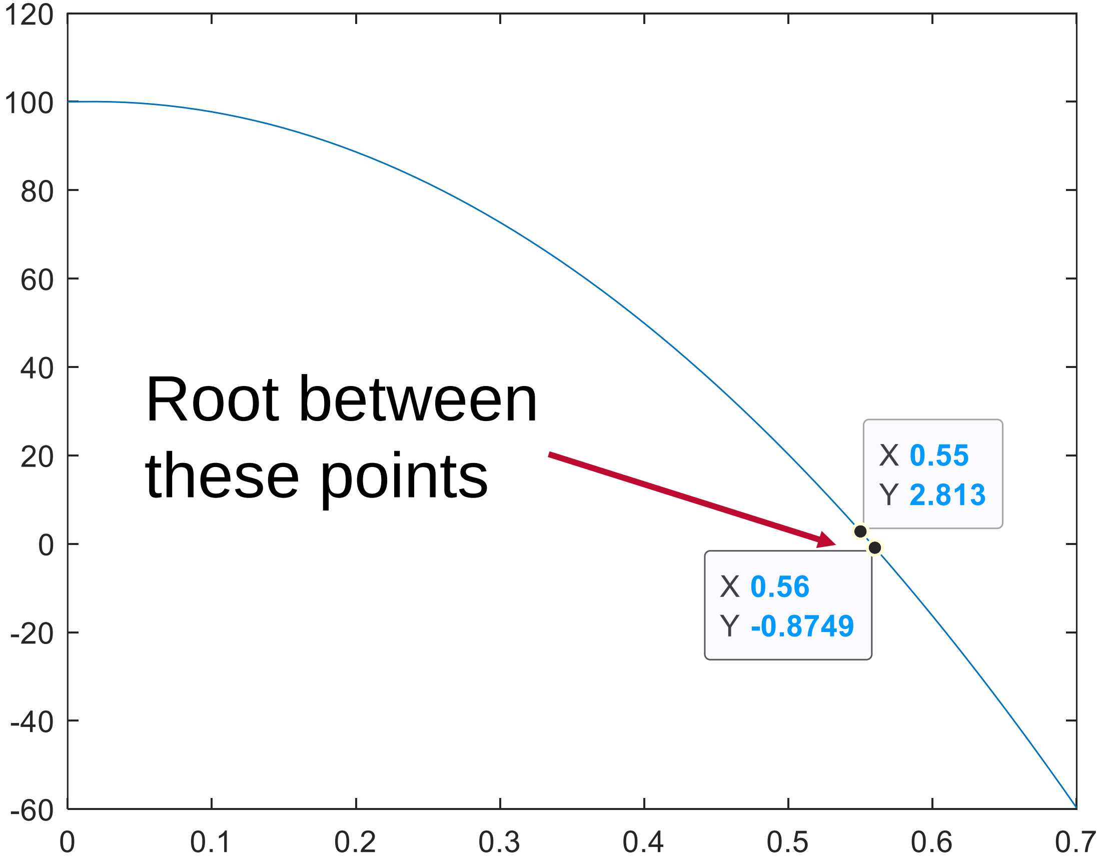
$$\frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2} = F$$

  - Numerical implementation `x(i+1) = 2*x(i) - x(i-1) + F*dt^2`

Second Order - Initial Value Problem (Newton's laws)

- For an object with mass 70kg falling under gravity (g=9.81) with an initial velocity of v_0=2m/s and an initial position of x_0=100m, numerically integrate to get to x=0.

```
m = 70; g = -9.81; F = m*g;
v0 = 2; x0 = 100;
M = 70; dt = 0.01; t(1) = 0;
x(1) = x0;
x(2) = v0*dt + x(1);
for i=2:M
    t(i) = i*dt
    x(i+1) = 2*x(i) - x(i-1) + F*dt^2 ;
end
plot(t, x(1:end-1))
```



Second Order - Initial Value Problem (Spring Mass)

- Recall the spring mass system you studied in last years dynamics lab

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

Mass damping spring constant

$$m\frac{x_{i+1} - 2x_i + x_{i-1}}{\Delta t^2} + c\frac{x_i - x_{i-1}}{\Delta t} + kx_i = 0$$

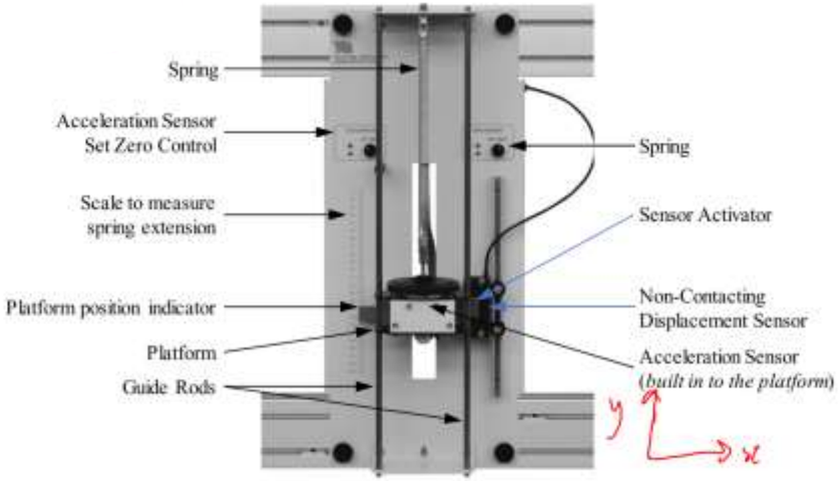


Figure 2 – Free vibration of a mass-spring system shown fitted to the free vibration test frame.

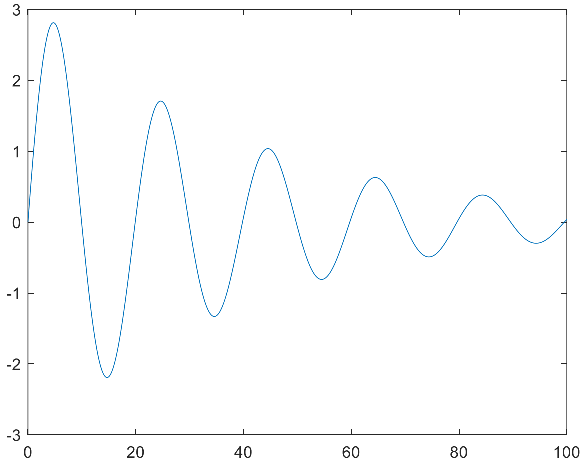
- Put discrete forms in equations and rearrange to get x_{i+1} (using x0 and v0) and defining mass m, spring constant k and damping (rate oscillation decreases) c.

$$x_{i+1} = 2x_i - x_{i-1} - \frac{c}{m}\Delta t(x_i - x_{i-1}) - \Delta t^2\frac{k}{m}x_i$$

Second Order - Initial Value Problem (Spring Mass)

- Put discrete forms in equations and rearrange to get x_{i+1} (using x0 and v0) and defining mass m, spring constant k and damping (rate oscillation decreases) c.

```
m = 1; %Mass
c = 0.05; %Damping
k = 0.1; %Spring constant
x0 = 0; %Initial position
v0 = 1; %Initial velocity
x(1) = x0;
x(2) = v0*dt + x(1);
M = 1000; dt=0.1;
for i =2:M
    x(i+1) = 2*x(i) - x(i-1) - dt*(c/m)*(x(i) - x(i-1)) - dt^2*(k/m)*x(i);
end
plot(0:dt:M*dt,x)
```



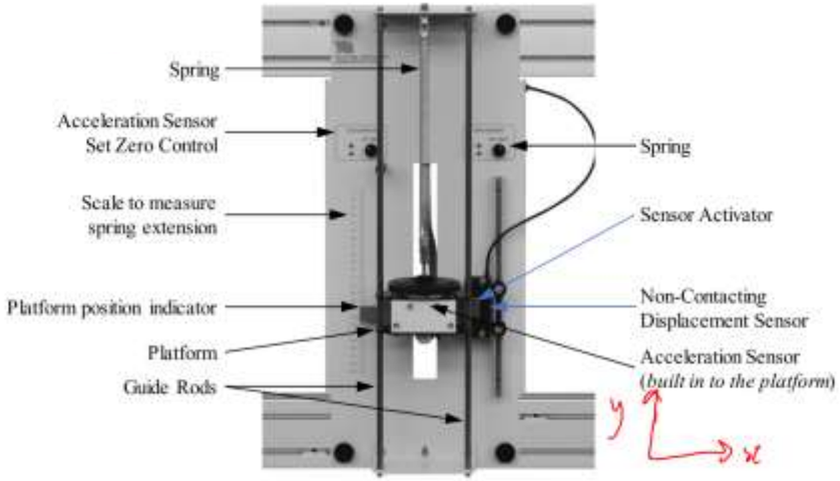
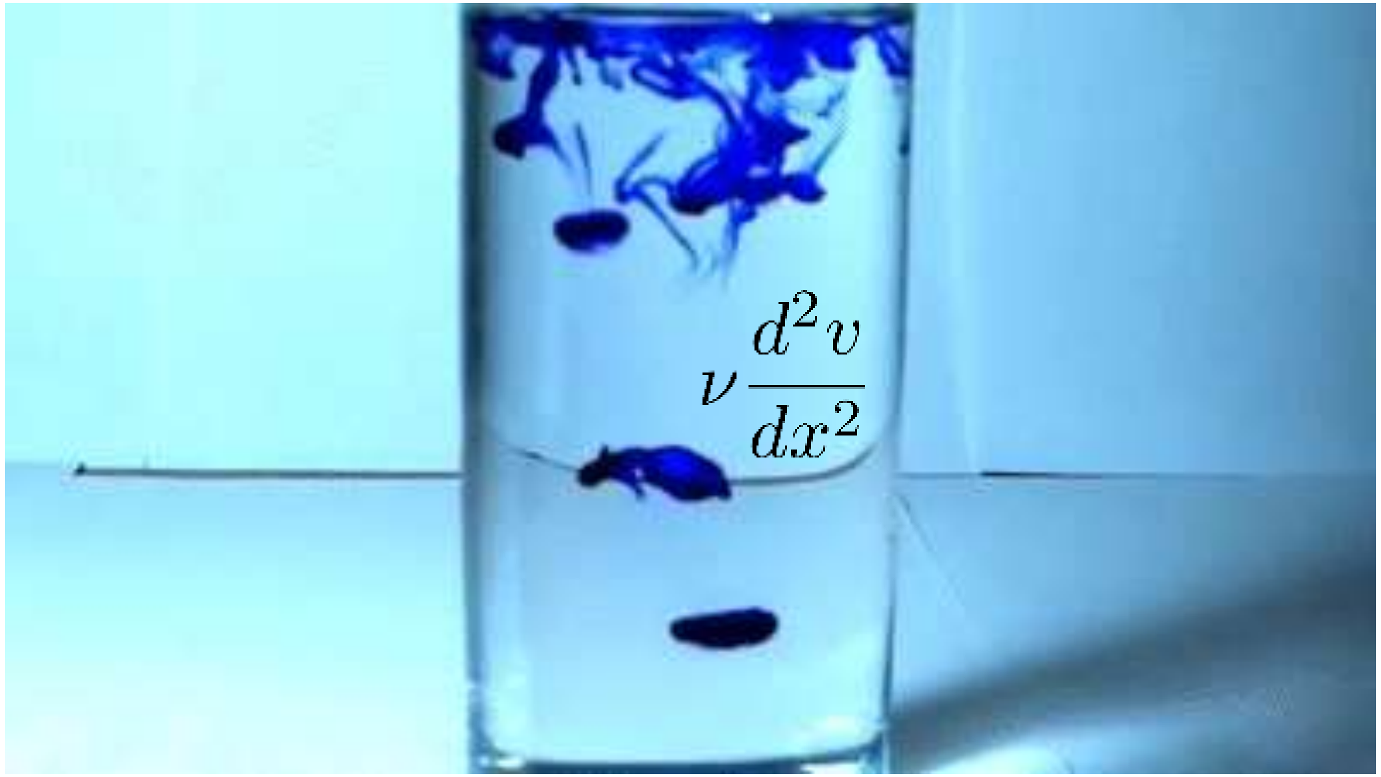


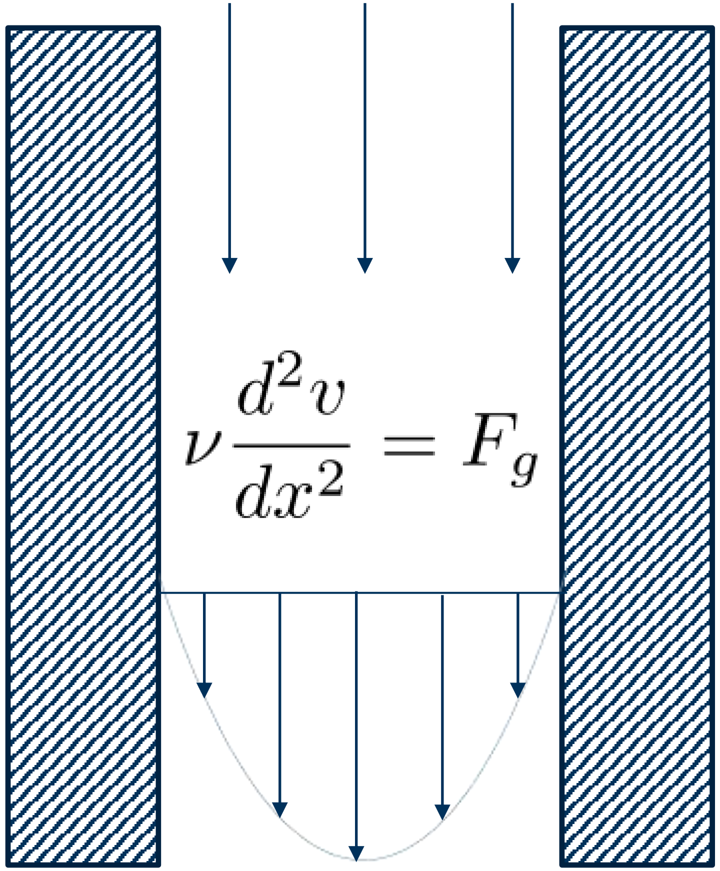
Figure 2 – Free vibration of a mass-spring system shown fitted to the free vibration test frame.

$$x_{i+1} = 2x_i - x_{i-1} - \frac{c}{m}\Delta t(x_i - x_{i-1}) - \Delta t^2 \frac{k}{m} x_i$$

Boundary Value Problem (flow between 2 walls)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - 2nd derivative models fluid diffusion with viscosity *v* (Greek nu)





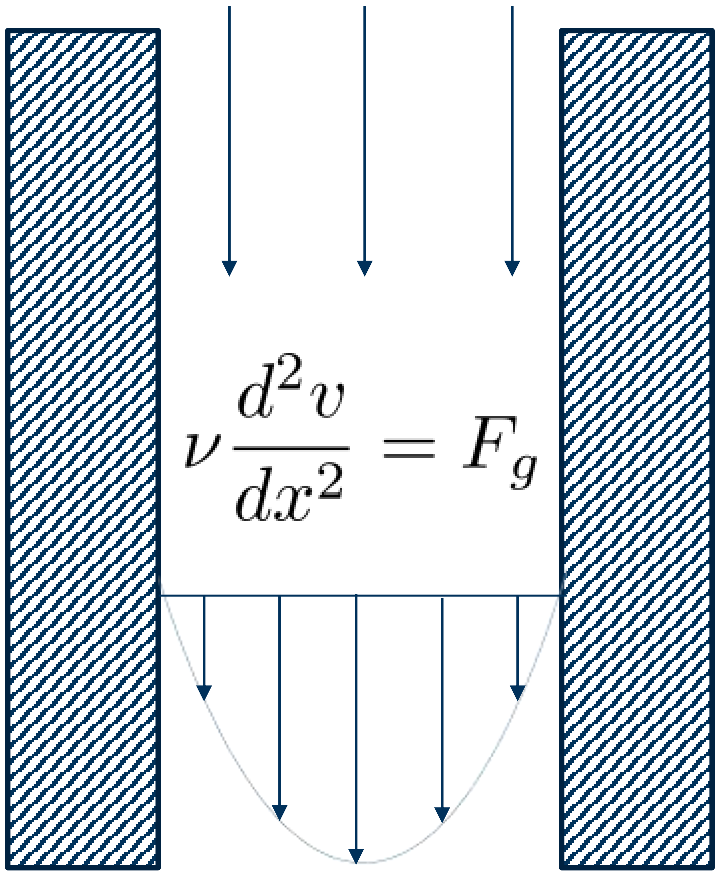
Boundary Value Problem (flow between 2 walls)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - 2^{nd} derivative models fluid diffusion with viscosity *v* (Greek nu)
  - Identical equation to before, discretised as before

```
v(i+1) = 2*v(i) - v(i-1) + (Fg/nu)*dx^2
```

  - However, this no longer model a change from an "initial value" but instead requires us to **iterate** until the solution agrees with the boundary values (flow is zero at **BOTH** walls)
  - N.B solvable exactly by integrating twice (C_2=0, C_1 using x=L)

$$\frac{dv}{dx} = \frac{F_g}{\nu}x + C_1 \qquad v(x) = \frac{F_g}{2\nu}x^2 + C_1 x + C_2$$

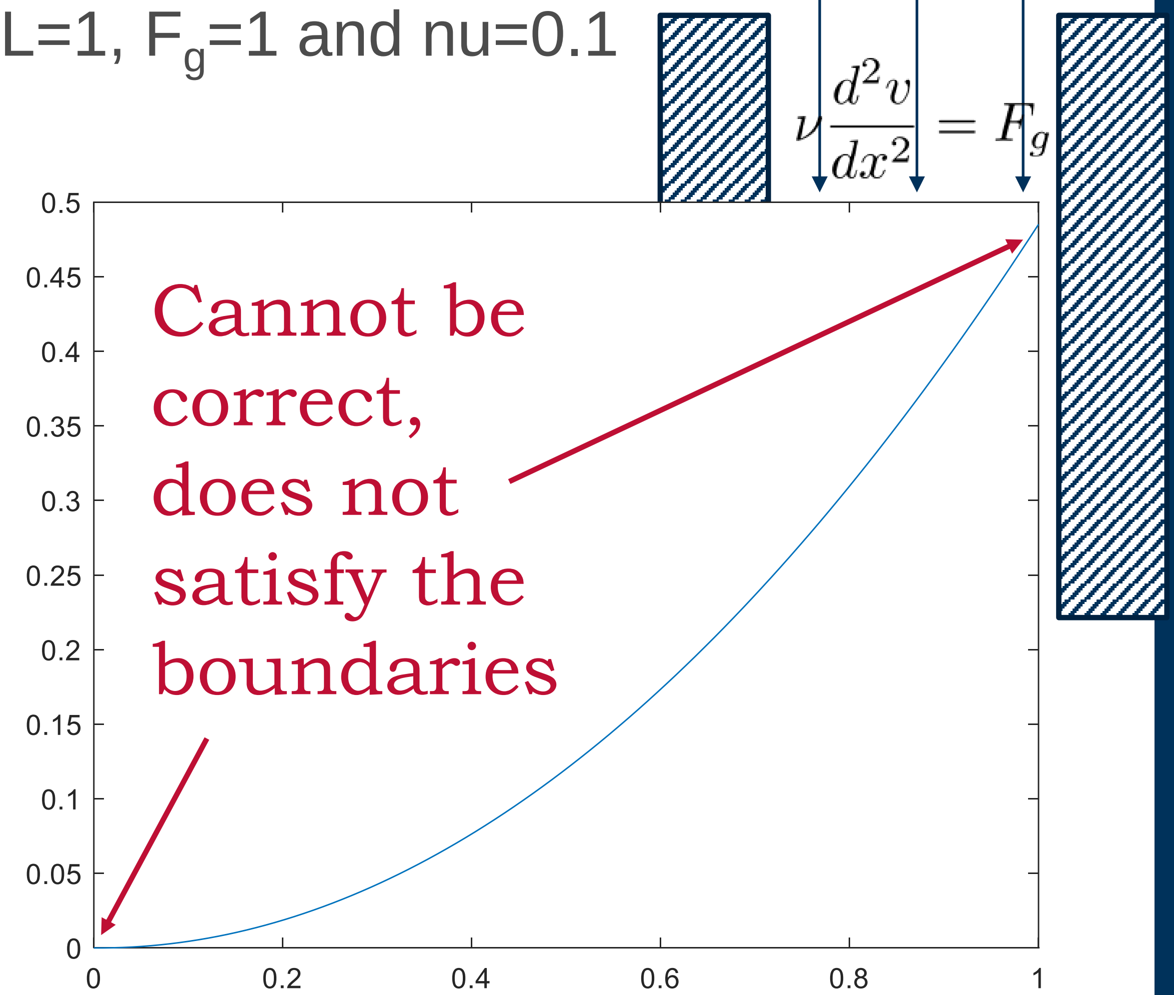


$$\nu \frac{d^2 v}{dx^2} = F_g$$

Boundary Value Problem (flow between 2 walls)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1

```
Fg = 1; nu=0.1; L=1;
M = 100; dx = L/M; x = linspace(0,L,M);
v(1) = 0; v(M) = 0;
for i=2:M-1
    v(i+1) = 2*v(i) - v(i-1) + Fg/nu*dx^2 ;
end
plot(x,v)
```



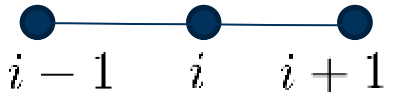
Boundary Value Problem (flow between 2 walls)

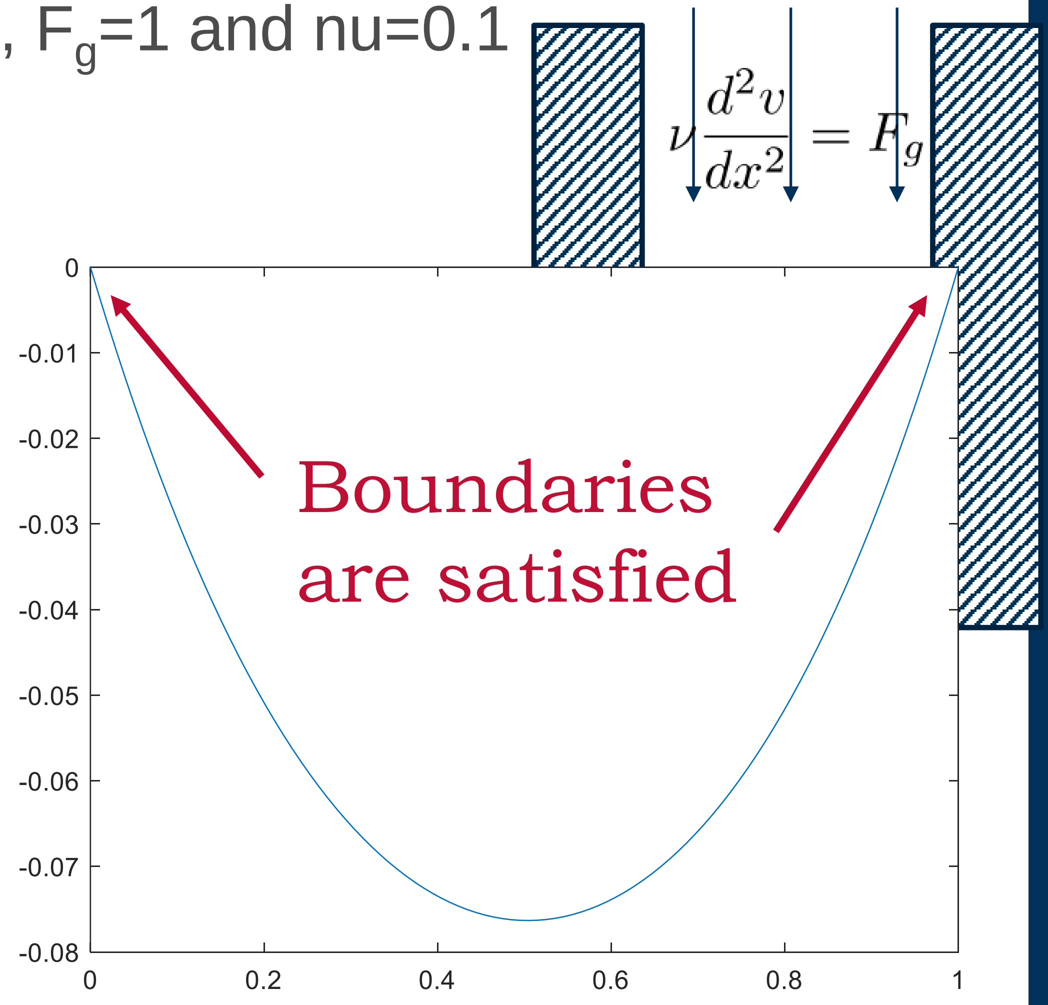
- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0) =0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1

```
Fg = 1; mu=0.1; L=1;
M = 100; dx = L/M;
x = linspace(0,L,M);
for iter=1:1000
    v(1) = 0; v(M) = 0;
    for i=2:M-1
        v(i) = (v(i+1) + v(i-1) - Fg*dx^2)/2;
    end
end
plot(x,v)
```

Iterate until v profile stops changing (will need more than 1000 here)

Note we rearrange so we get v(i) from above and below



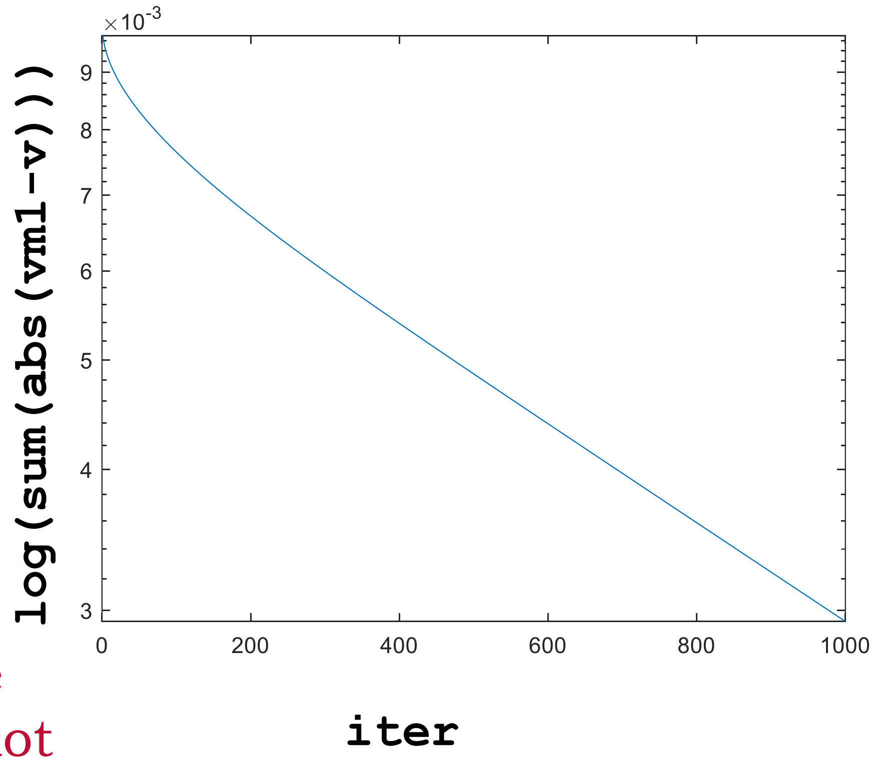


Boundary Value Problem (flow between 2 walls)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0) =0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1

```
Fg = 1; mu=0.1; L=1;
M = 100; dx = L/M;
x = linspace(0,L,M);
v = zeros(M,1); vm1=v;
for iter=1:10000
    v(1) = 0; v(M) = 0;
    for i=2:M-1
        v(i) = (v(i+1) + v(i-1) - Fg*dx^2)/2;
    end
    res(iter) = sum(abs(vm1-v));
    vm1=v;
end
semilogy(res)
```

How may iterations do we need?

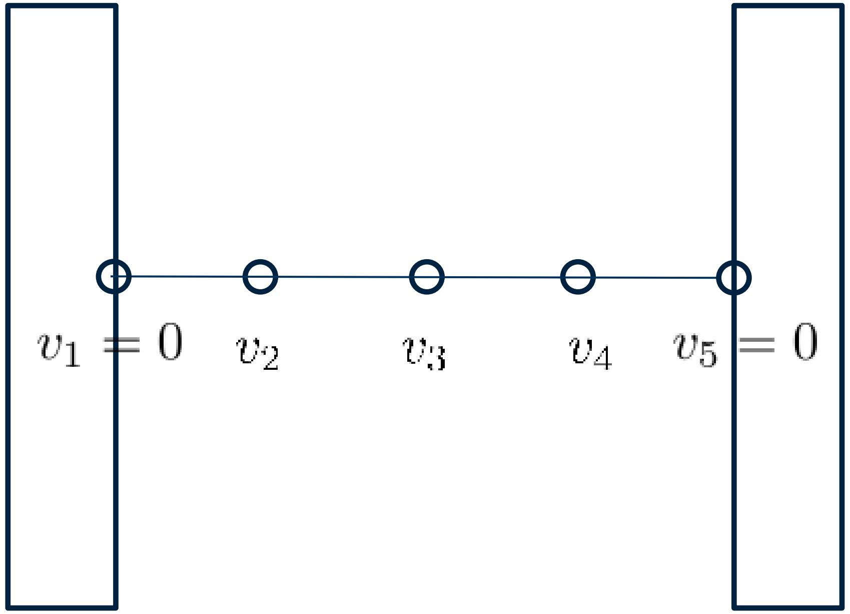


Collect sum of absolute change each iter and plot on log y axis

Boundary Value Problem (flow between 2 walls)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - To understand, let's simplify to include just 3 points in channel

$$\nu \frac{d^2 v}{dx^2} = F_g$$




```
Fg = 1; nu=0.01; L=1; M = 5; dx = L/M;
x = linspace(0,L,M); v = zeros(M,1);
for iter=1:100
    v(1) = 0; v(M) = 0;
    for i=2:M-1
        v(i)=(v(i+1)+v(i-1)-Fg*dx^2/nu)/2;
        plot(x,v,'r-o’); hold on
        plot([x(i-1),x(i),x(i+1)], ...
             zeros(3,1),'k-o', ...
             "LineWidth", 5, ...
             "MarkerSize", 10)
        hold off;  pause(0.5)
    end
end
```

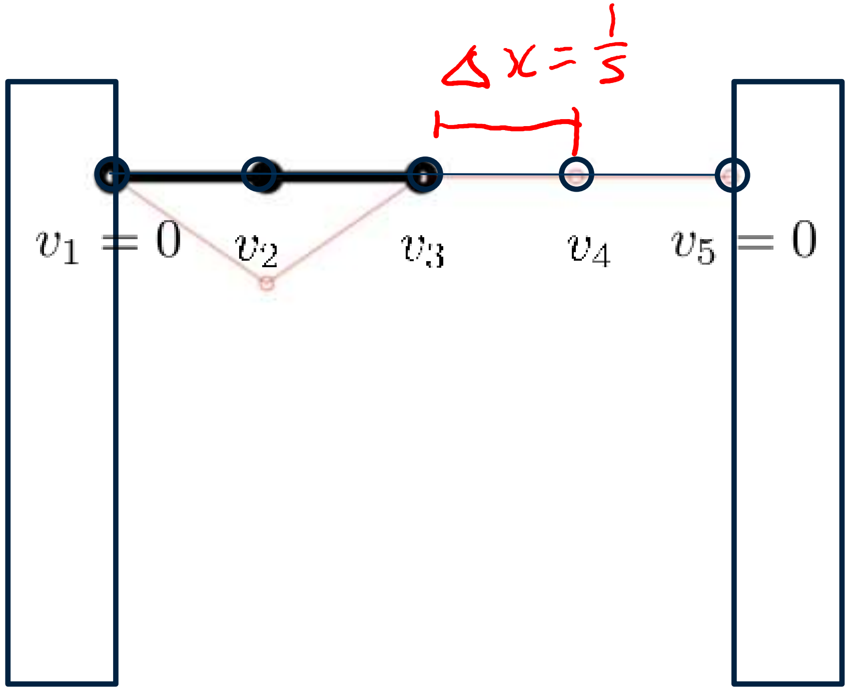
Boundary Value Problem (Explicit Solution)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - To understand, let's simplify to include just 3 points in channel

$$\frac{d^2v}{dx^2} = \frac{F_g}{\nu} = \frac{1}{0.1} = 10$$

$$\Delta x^2 = \frac{1}{25}$$

Hand calculation



① $V_1 - 2V_2 + V_3 - \Delta x^2 F_g/\nu = 0$

② $V_2 - 2V_3 + V_4 - \Delta x^2 F_g/\nu = 0$

③ $V_3 - 2V_4 + V_5 - \Delta x^2 F_g/\nu = 0$

for ① $V_2 = \frac{1}{2}\left[V_1 + V_3 - \Delta x^2 F_g/\nu\right]$

$= \frac{1}{2}\left[0 + 0 - \left(\frac{1}{25}\right)(10)\right]$

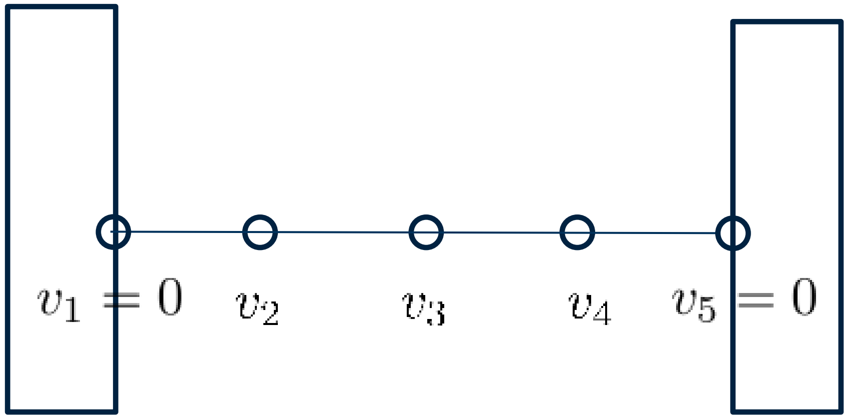
$= 0.2$ → then do ② etc

Boundary Value Problem (Explicit Solution)

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1

$$\frac{d^2 v}{dx^2} = \frac{F_g}{\nu}$$

  - To understand, let's simplify to include just 3 points in the channel
  - We see simply iterating until the results stop changing is a very inefficient way of solving these 3 simultaneous equations
  - This is known as an explicit method



① $V_1 - 2V_2 + V_3 - \Delta x^2 F_g/\nu = 0$

② $V_2 - 2V_3 + V_4 - \Delta x^2 F_g/\nu = 0$

③ $V_3 - 2V_4 + V_5 - \Delta x^2 F_g/\nu = 0$

Recall Solving Problems in Terms of Matrices

- We can solve simulatanous equations by forming matrices

$$2x + 3y = 7 \qquad (1)$$
$$x + 4y = 1 \qquad (2)$$

$$\begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 7 \\ 1 \end{bmatrix}$$

$$\to x = 1 - 4y$$

Sub into (1)

$$2(1-4y) + 3y = 7$$
$$2 - 8y + 3y = 7$$
$$-5y = 5 \rightsquigarrow y = -1$$

So from (2) $x = 5$

- Which can be written in the follow matrix form

$$\underbrace{\begin{pmatrix} 2 & 3 \\ 1 & 4 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} x \\ y \end{pmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{pmatrix} 7 \\ 1 \end{pmatrix}}_{\boldsymbol{b}} \to \begin{matrix} 2x + 3y = 7 \\ 1x + 4y = 1 \end{matrix}$$

- In Matlab code, solving **Ax**=**b** is done as follows,

```
A = [2, 3 ; 1, 4]
b = [7; 1]
x = A^(-1)*b   → ans = [5; -1]
```

Implicit Solution

- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - An implicit method solves these simultaneous equations directly

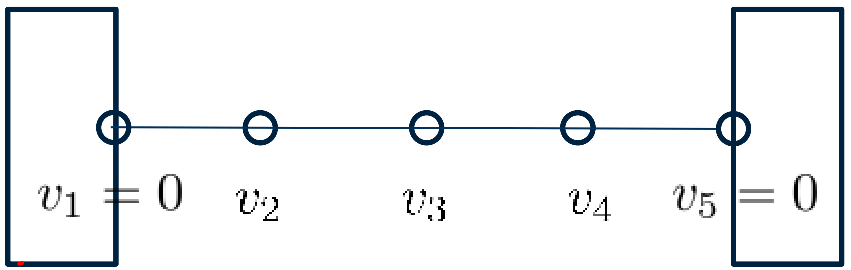
$$\frac{d^2 v}{dx^2} = \frac{F_g}{\nu}$$

① $V_1 - 2V_2 + V_3 - \Delta x^2 F_g/\nu = 0$

② $V_2 - 2V_3 + V_4 - \Delta x^2 F_g/\nu = 0$

③ $V_3 - 2V_4 + V_5 - \Delta x^2 F_g/\nu = 0$

- Taking the coefficient of each term and writing in a matrix
- Inverting the matrix will solve the system of equations



$$\underbrace{\begin{pmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} v_2 \\ v_3 \\ v_4 \end{pmatrix}}_{\boldsymbol{v}} = \underbrace{\begin{pmatrix} \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \end{pmatrix}}_{\boldsymbol{f}}$$

Implicit Solution

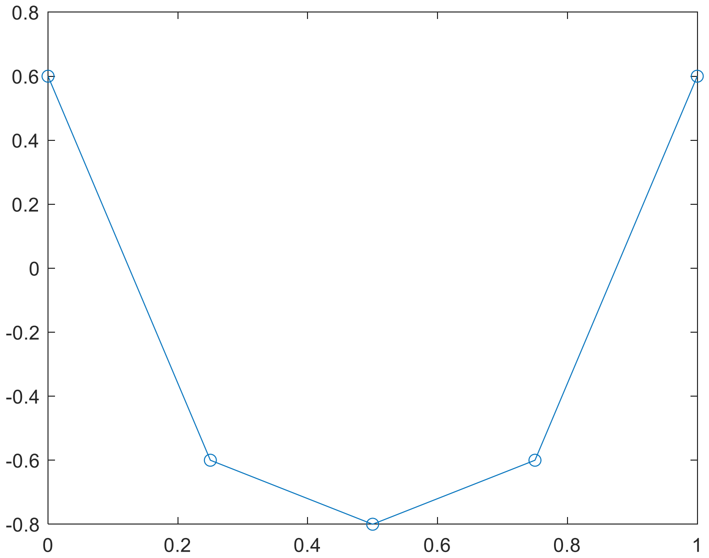
- Flow between two walls drive by gravity, v=0 at the walls. We define wall positions v(x=0)=0 and v(x=L)=0 with L=1, F_g=1 and nu=0.1
  - Defining the matrix in MATLAB

$$\frac{d^2v}{dx^2} = \frac{F_g}{\nu} = \frac{1}{0.1} = 10$$

$$\Delta x^2 = \frac{1}{25}$$

```
%Define coefficients
Fg = 1; nu=0.1; L=1;
M = 5; dx = L/M;
x = linspace(0,L,M);
%Form matrix
A = [ -2 1 0 ;
      1 -2 1 ;
      0 1 -2 ];
RHS = dx^2*Fg/nu;
f = [RHS RHS RHS];
%Solve Implicit Equation
v = f/A;
plot(x(2:end-1), v, 'rs-');
```

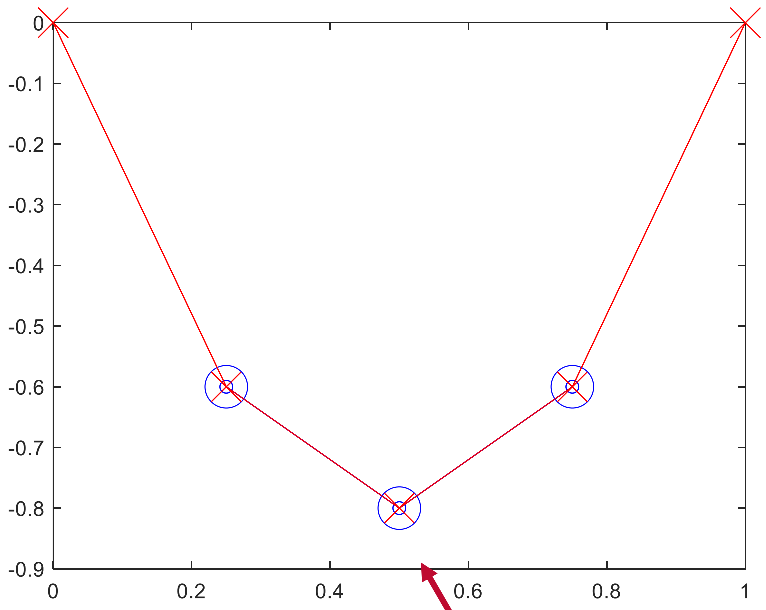
$$\underbrace{\begin{pmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix}}_{\mathbf{A}} \underbrace{\begin{pmatrix} v_2 \\ v_3 \\ v_4 \end{pmatrix}}_{\mathbf{v}} = \underbrace{\begin{pmatrix} \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \end{pmatrix}}_{\mathbf{f}}$$



Explicit vs Implicit

- Explicit Method

```
%Define coefficients
Fg = 1; mu=0.1; L=1;
M = 100; dx = L/M;
x = linspace(0,L,M);

%Iterate until converged
for iter=1:1000
    v(1) = 0; v(M) = 0;
    for i=2:M-1
        v(i)=(v(i+1)+v(i-1) ...
                - Fg*dx^2)/2;
    end
end
plot(x,v, ‘bo-’)
hold on
```



- Implicit Method

```
%Form matrix
A = [ -2 1 0 ;
      1 -2 1 ;
      0 1 -2 ];
RHS = dx^2*Fg/nu;
f = [RHS RHS RHS];
%Solve Implicit Equation
v = f/A;
plot(x(2:end-1), v, 'rs-');
```

Same results from both methods (blue circles and red crosses) but implicit much faster

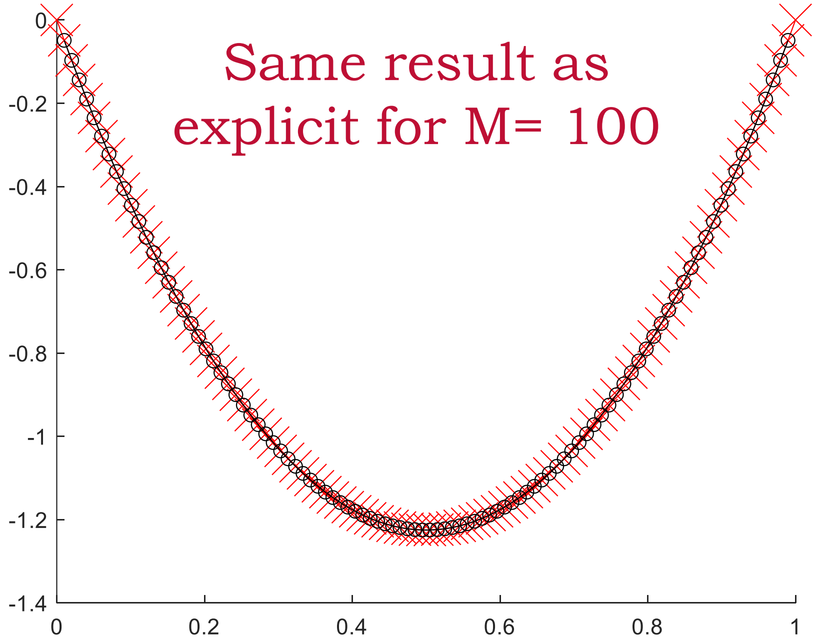
Implicit for a general matrix

- We can generate the implicit matrix for any number of elements by observing diagonal components are -2 and off diagonal are 1 while RHS is f

$$\underbrace{\begin{pmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix}}_{\boldsymbol{A}} \underbrace{\begin{pmatrix} v_2 \\ v_3 \\ v_4 \end{pmatrix}}_{\boldsymbol{v}} = \underbrace{\begin{pmatrix} \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \\ \Delta x^2 F_g/\nu \end{pmatrix}}_{\boldsymbol{f}}$$

```
M = 100; dx = L/M;
x = linspace(0,L,M);
RHS = dx^2*Fg/nu;
for i=1:M-2
    for j=1:M-2
        if (i == j)
            A(i,j) = -2;
        elseif (i+1 == j || i-1 ==j)
            A(i,j) = 1;
        else
            A(i,j) = 0;
        end
    end
    f(i) = RHS;
end
v = f/A;
```

There might be a more elegant way of doing this, but this works



Same result as explicit for M= 100

Recap

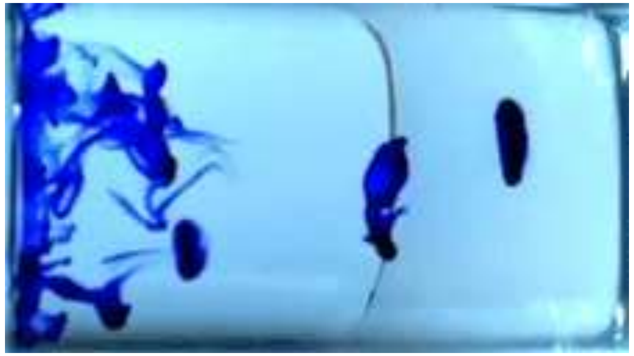
- Because in a bounded value problem, both top and bottom boundaries must be satisfied, the equation must iterate until both are correctly applied
- An explicit method solves term by term, looping/moving between the two boundaries until the solution stops changing
- An implicit solution recognises the terms are simultaneous equations and puts them in a matrix Ax = b form which can be solved
- Implicit solutions are much more efficient, especially with many points which take longer to converge for explicit methods
- However, explicit methods can be understood further in the context of time evolving partial differential equations

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}$$

Time Evolving Equations

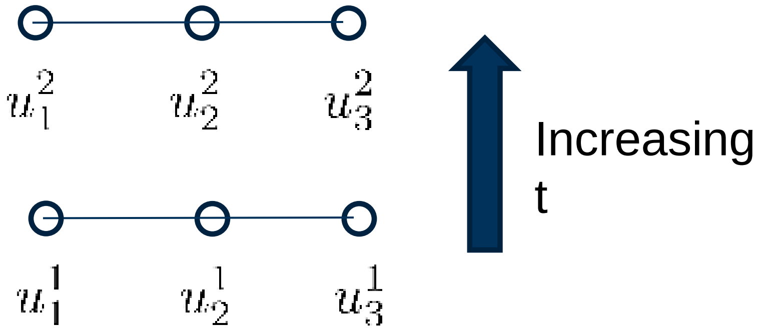
- We will cover partial differential equations next lecture but introduce the concept now
  - We have a time evolving term on the left
  - We have a spatial diffusion term on the right

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}$$



- We discretise this using the same formulas we have seen already
  - However, we denote time as a superscript
  - Spatial components are subscripts as previously

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \nu \frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{\Delta x^2}$$

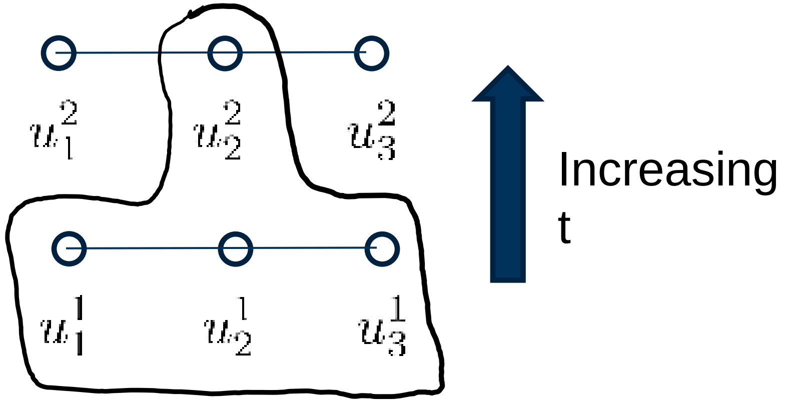


Time Evolving Equations

- We discretise this using the same formulas we have seen already
  - However, we denote time as a superscript
  - Spatial components are subscripts as previously

$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \nu \frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{\Delta x^2}$$

```
%Define coefficients
nu=0.1; L=1; M = 100; dx = L/M; dt=0.0001;
x = linspace(0,L,M); u = zeros(M,1); u(end/2) = 1;
utp1 = u;
%Iterate in time
for t=1:1000
    u(1) = 0; u(M) = 0;
    for i=2:M-1
        utp1(i)=u(i)+dt*nu*(u(i+1)-2*u(i)+u(i-1))/dx^2;
    end
    plot(x,utp1); pause(0.1)
    u = utp1;
end
```

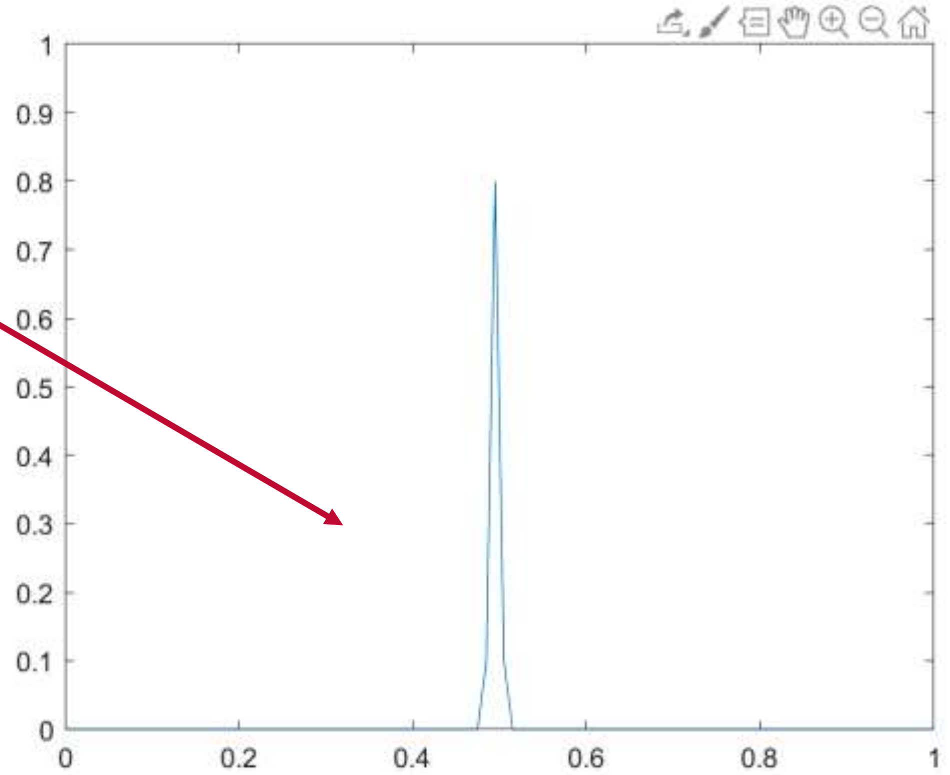


Time Evolving Equations

- This models time evolving diffusion – the explicit iteration can now be though of as evolving the system in time

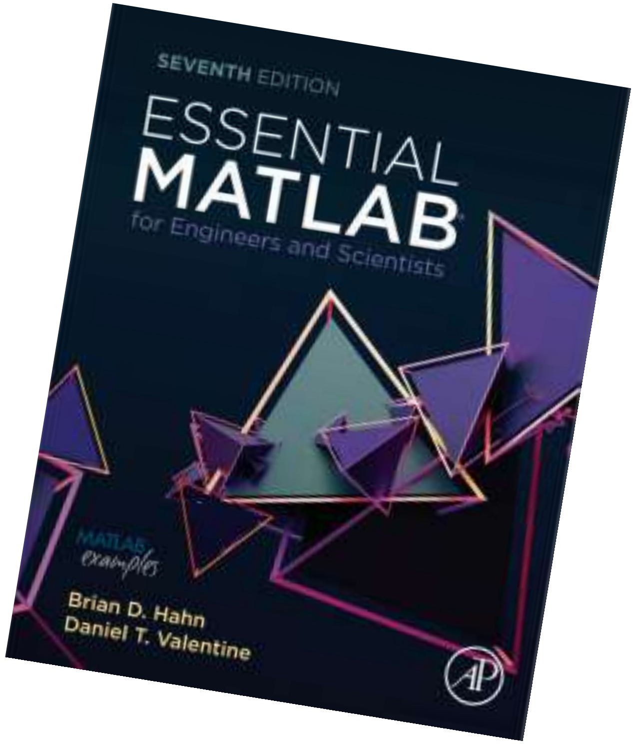
```
%Define coefficients
nu=0.1; L=1; M = 100; dx = L/M; dt=0.0001;
x = linspace(0,L,M); u = zeros(M,1);
u(end/2) = 1; utp1 = u;
%Iterate in time
for t=1:1000
    u(1) = 0; u(M) = 0;
    for i=2:M-1
        utp1(i)=u(i)+dt*nu*(u(i+1) ...
                -2*u(i)+u(i-1))/dx^2;
    end
    plot(x,utp1); pause(0.1)
    u = utp1;
end
```

Initial value of 1 in the middle



Summary

- Brief recap of differential equation GRADER
- Second order differential equations
  - Initial value
  - Boundary value (iteration)
- Implicit vs Explicit solutions
- Partial Differential Equations



Essential Matlab (EM)
http://tinyurl.com/yy53shga