Brunel University London

ME2610
Engineering Mathematics and Programming

17th November 2020

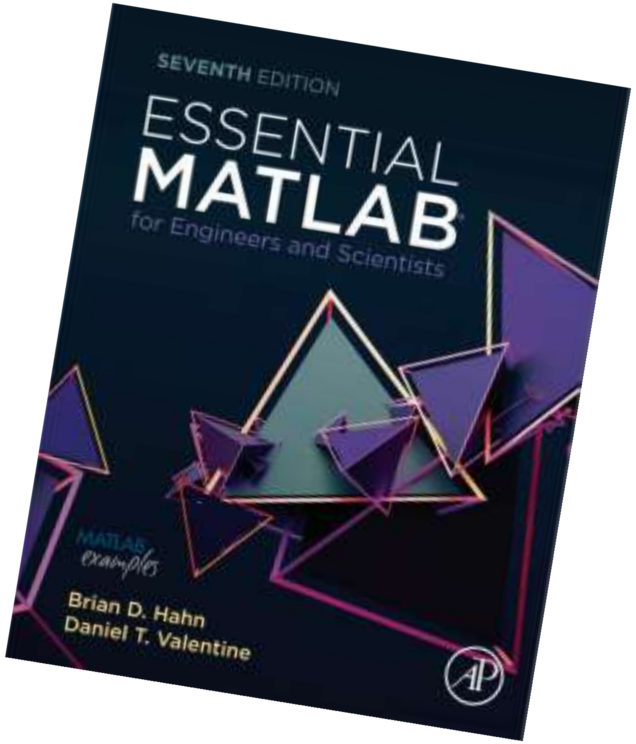
Dr Edward Smith

Room 105

Howell Building

Summary

- Recap of Partial Differential Equation
  - Temporal-spatial and spatial in two dimensional
  - Some boundary and initial conditions
- Combining both for the assessment exercise
  - A two dimensional time evolving field
  - Boundary conditions along the 4 sides
- Summary for Concepts needed for Assessment
  - Recap of functions, arrays and error checking
  - Best practice advice
  - Validation and verification



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

Partial derivative example quite complex in EM as they use implicit

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)

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

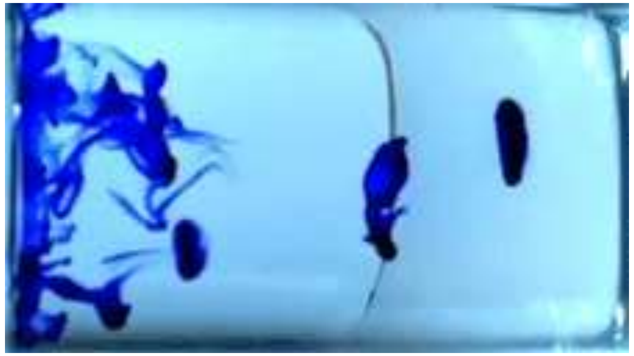
5	13	7	Integration Methods	Trapizum rule	|	27/10/2020
5	14	7	Integration Numerics	Simpson Rule	|	27/10/2020
5	15	8	Integration methods	Gauss integration	↓	29/10/2020
5	16	9	Gauss integration	Gauss integration	**Root/interpolation**	29/10/2020
6	17	10	Diff. equations (integrating factors, order)	Basic 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

Assignment Deadline 27th Nov 23:59 on WiseFlow **and** GRADER

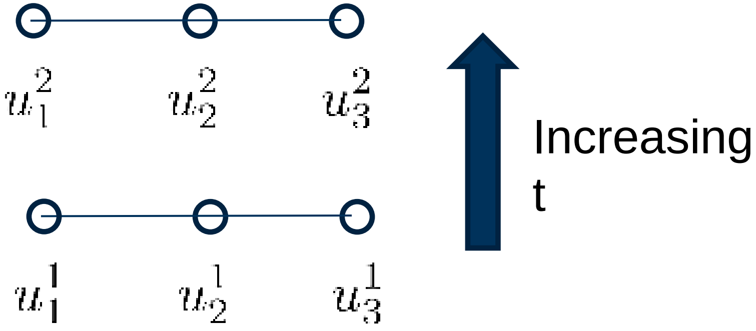
Recap 1D Time Evolving Equations

- Partial differential equations can include changes in time and 3 dimensions in space – we will start with 1D and varying in time
  - 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



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

Recap 1D 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

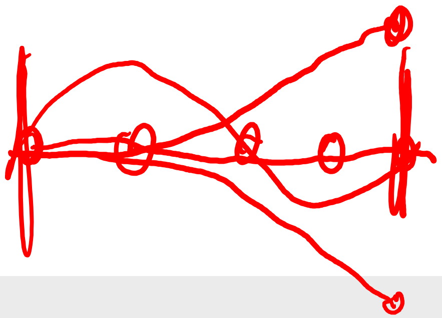
$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial x^2}$$

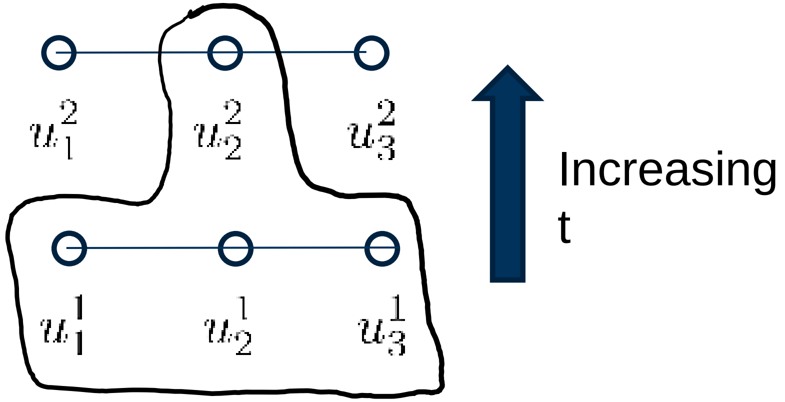
$$\frac{u_i^{t+1} - u_i^t}{\Delta t} = \nu \frac{u_{i+1}^t - 2u_i^t + u_{i-1}^t}{\Delta x^2}$$

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

```
%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
```

Note BC set before loop



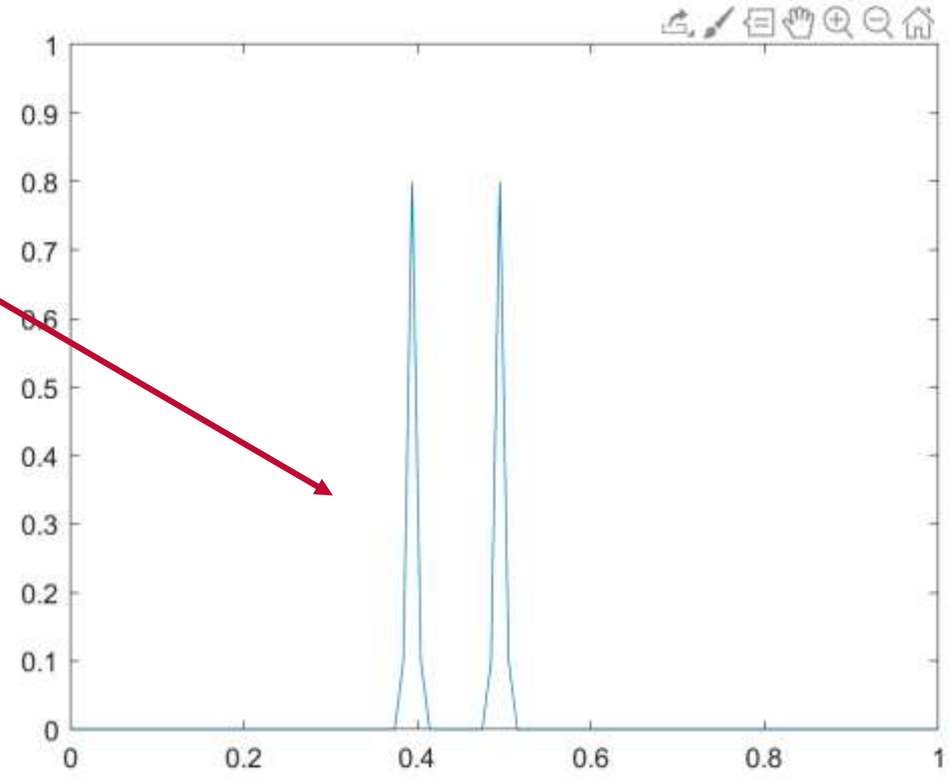


Recap - 1D Time Evolving Equations initial conditions

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

```
%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(4*end/10)=1; u(5*end/10)=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
```

2 values of 1 in the middle

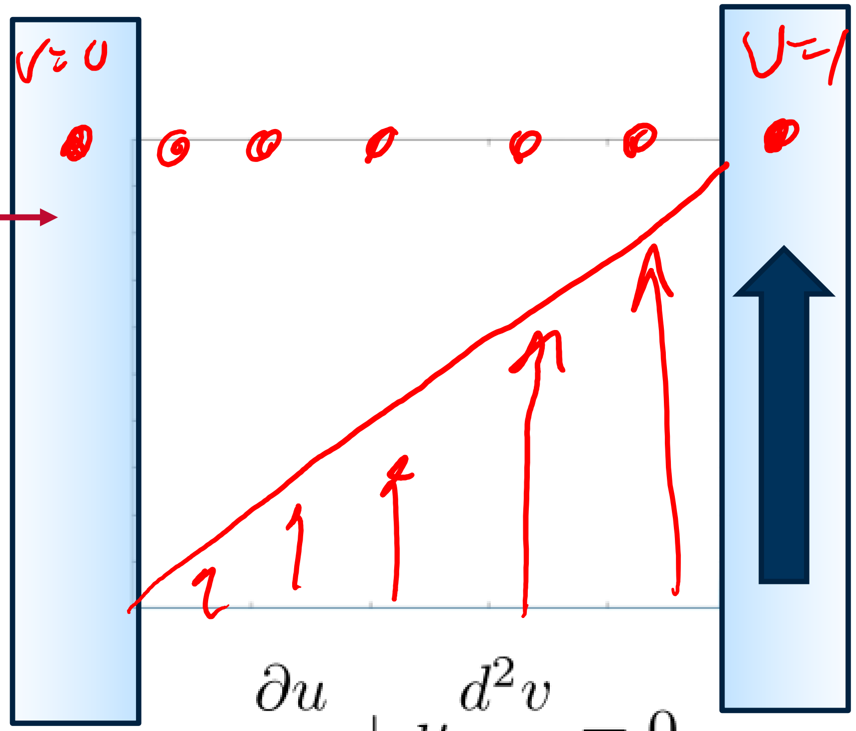


Recap - 1D Moving Wall Boundary Conditions

- A channel with a moving wall (e.g. inside a bearing, the gap between an engine piston head and wall or a fluid film)

```
%Define coefficients
nu=0.1; L=1; M = 100; dx = L/M; dt=0.0001;
x = linspace(0,L,M); u = zeros(M,1);
utp1 = u;
%Iterate in time
for t=1:1000
    u(1) = 1; u(M) = 1;
    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
```

Wall velocity set to 1



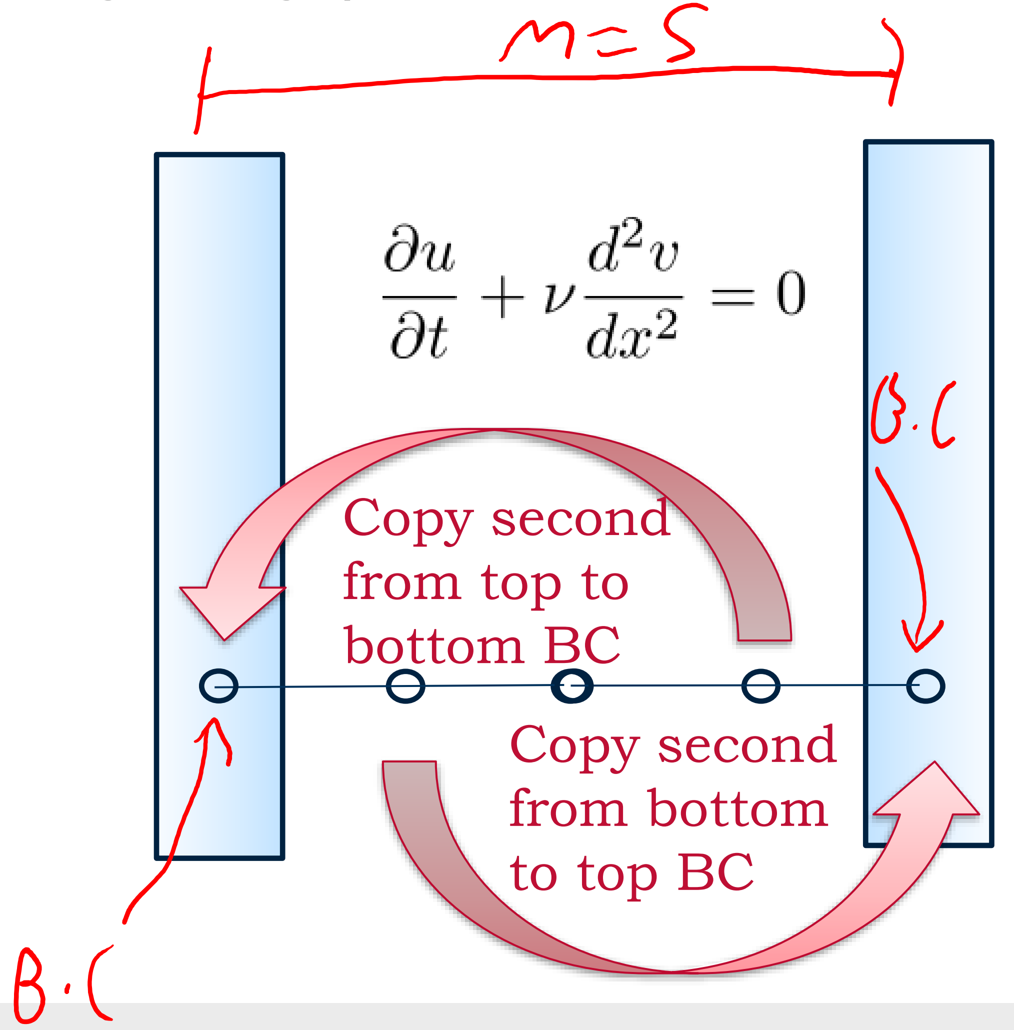
$$\frac{\partial u}{\partial t} + \nu \frac{d^2 v}{dx^2} = 0$$

Recap - Periodic Boundary Conditions (BC)

- A channel with a moving wall (e.g. inside a bearing, the gap between an engine piston head and wall or a fluid film)

```
%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(2) = 1; utp1 = u;
%Iterate in time
for t=1:1000
    u(1) = u(M-1); u(M) = u(2);
    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
```

Copy one side to other

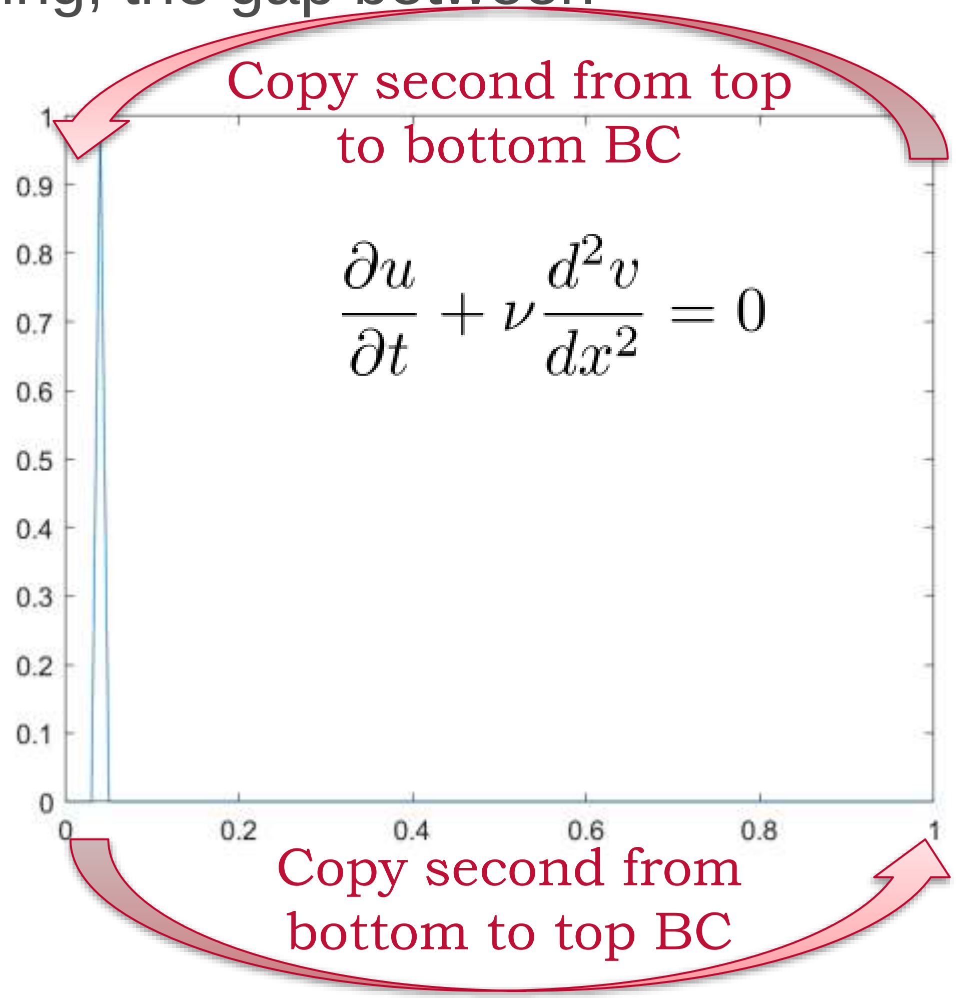


Recap - Periodic Boundary Conditions (BC)

- A channel with a moving wall (e.g. inside a bearing, the gap between an engine piston head and wall or a fluid film)

```
%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(5) = 1; utp1 = u;
%Iterate in time
for t=1:1000
    u(1) = u(M-1); u(M) = u(2);
    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 near the edge



$$\frac{\partial u}{\partial t} + \nu \frac{d^2 v}{dx^2} = 0$$

Recap - Laplace's Equation

- To describe the change in fields, we use partial differential equations which vary in space (2D here), for example Laplace's Equation:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

- Often written using other notation,

$$\nabla^2 f = 0 \text{ or } \Delta f = 0 \text{ where } \nabla^2 \equiv \Delta \equiv \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

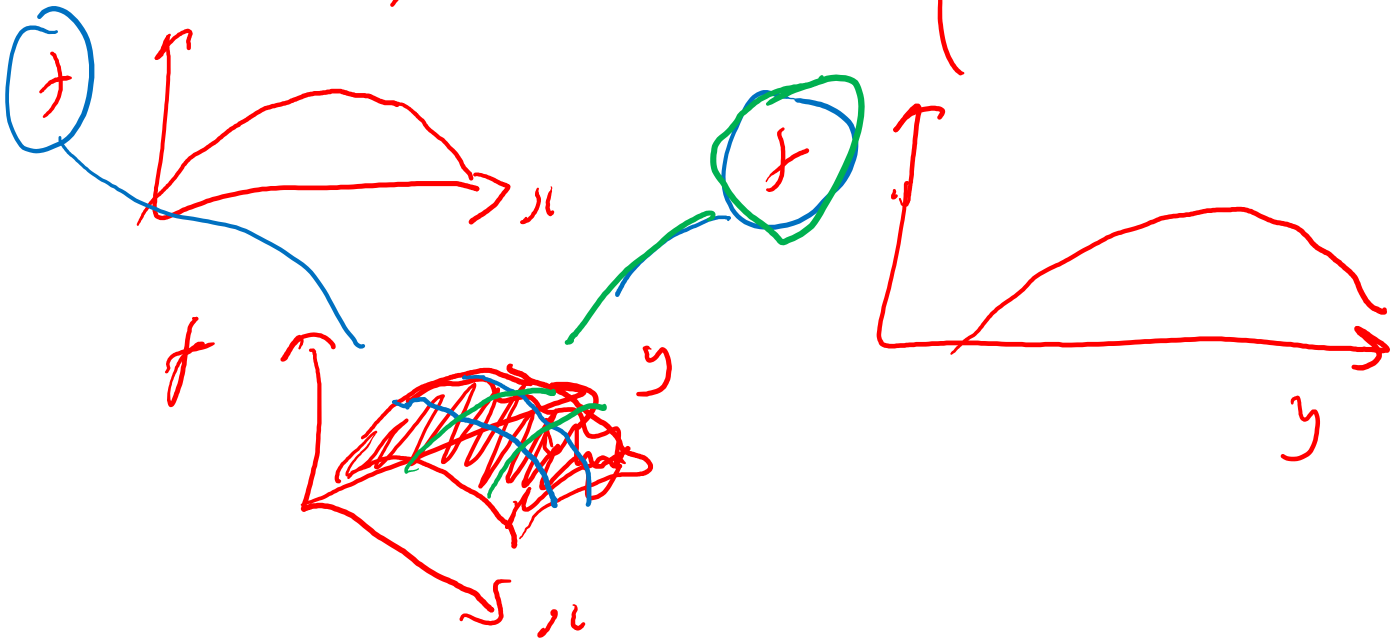
$$f_{xx} + f_{yy} = 0 \text{ where subscripts denote derivatives}$$

Two Dimensional Field

- Consider the example field described by an x-y polynomial

$$f(x,y) = ax^2 + bx + cy^2 + dy + exy + f$$

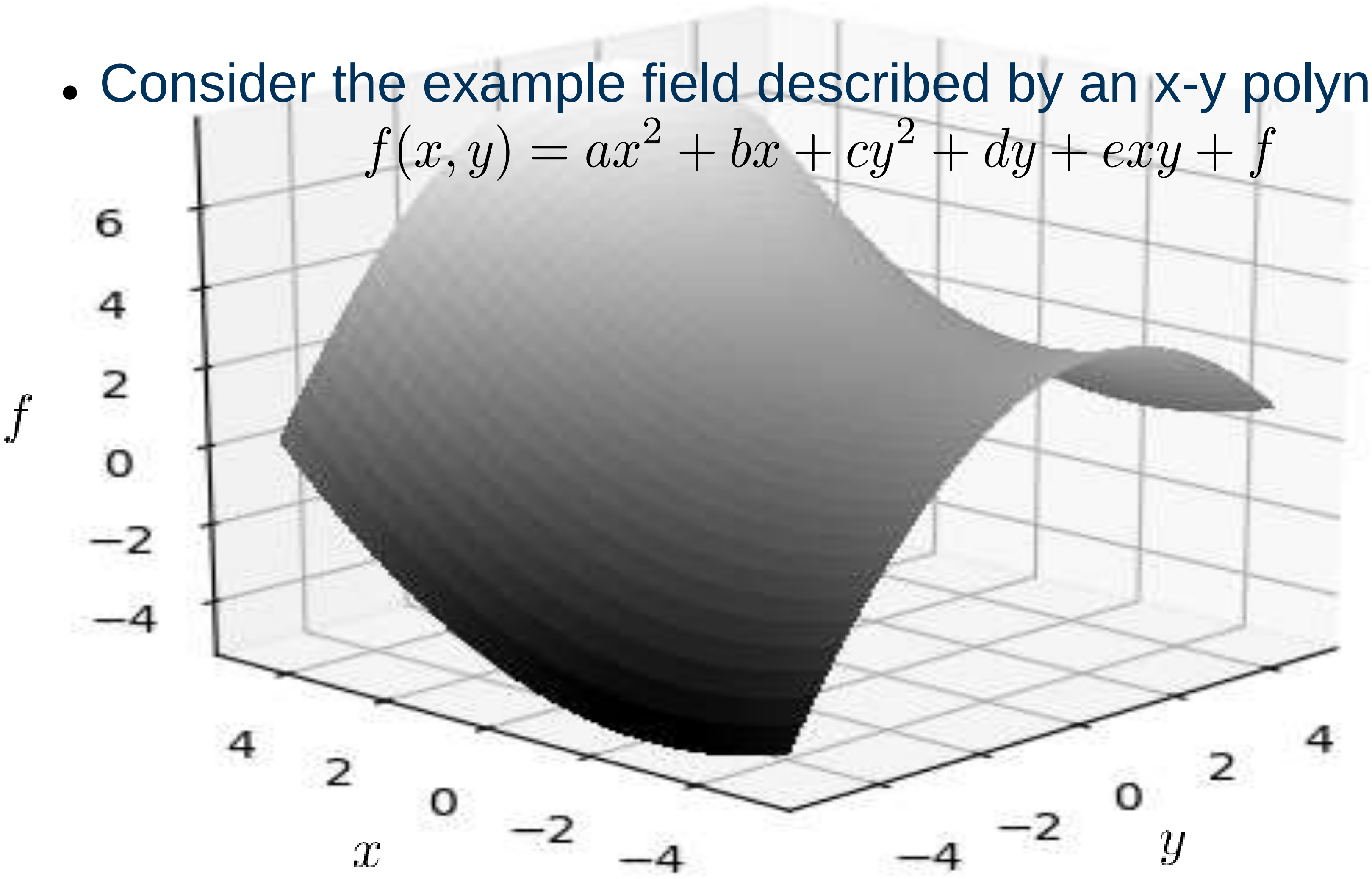
cross term



Two Dimensional Field

- Consider the example field described by an x-y polynomial

$$f(x,y) = ax^2 + bx + cy^2 + dy + exy + f$$



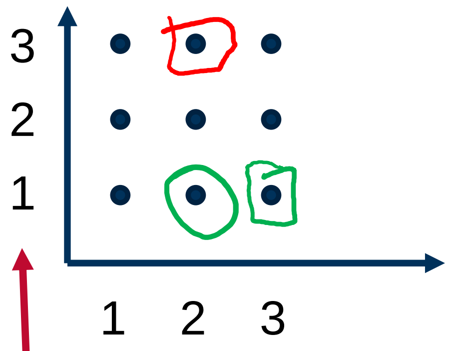
- A 2D field is a function of two variables

$$f = f(x,y)$$

- Show here in 3D for visualisation
- Assumed to be a continuous function

Plotting a 2D Field in Excel

- Plotted in Excel – create a grid of x and y values then plot f(x,y)



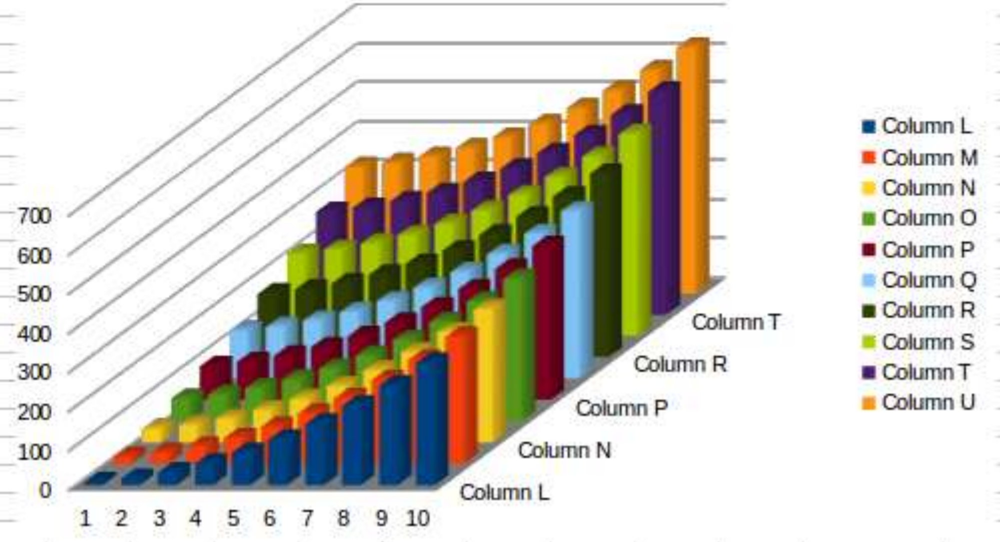
X values at each point in a grid

Corresponding Y values at each point in a grid

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	x											f									
2	1	2	3	4	5	6	7	8	9	10		7	18	35	58	87	122	163	210	263	322
3	1	2	3	4	5	6	7	8	9	10		17	28	45	68	97	132	173	220	273	332
4	1	2	3	4	5	6	7	8	9	10		33	44	61	84	113	148	189	236	289	348
5	1	2	3	4	5	6	7	8	9	10		55	66	83	106	135	170	211	258	311	370
6	1	2	3	4	5	6	7	8	9	10		83	94	111	134	163	198	239	286	339	398
7	1	2	3	4	5	6	7	8	9	10		117	128	145	168	197	232	273	320	373	432
8	1	2	3	4	5	6	7	8	9	10		157	168	185	208	237	272	313	360	413	472
9	1	2	3	4	5	6	7	8	9	10		203	214	231	254	283	318	359	406	459	518
10	1	2	3	4	5	6	7	8	9	10		255	266	283	306	335	370	411	458	511	570
11	1	2	3	4	5	6	7	8	9	10		313	324	341	364	393	428	469	516	569	628

	A	B	C	D	E	F	G	H	I	J
12										
13	y									
14	1	1	1	1	1	1	1	1	1	1
15	2	2	2	2	2	2	2	2	2	2
16	3	3	3	3	3	3	3	3	3	3
17	4	4	4	4	4	4	4	4	4	4
18	5	5	5	5	5	5	5	5	5	5
19	6	6	6	6	6	6	6	6	6	6
20	7	7	7	7	7	7	7	7	7	7
21	8	8	8	8	8	8	8	8	8	8
22	9	9	9	9	9	9	9	9	9	9
23	10	10	10	10	10	10	10	10	10	10
24										
25		coefficients								
26		a	3							
27		b	2							
28		c	3							
29		d	1							
30		e	0							
31		f	-2							
32										
33										
34										

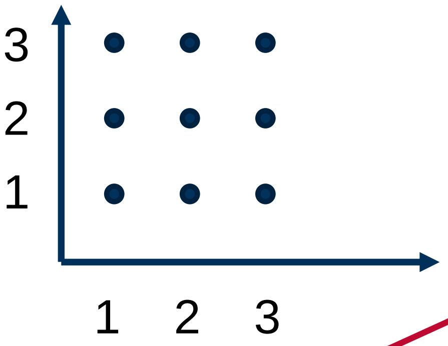
$$f(x,y) = ax^2 + bx + cy^2 + dy + exy + f$$



Plotting a 2D Field in MATLAB

- Plotted in MATLAB – using meshgrid for x and y values then plot f(x,y)

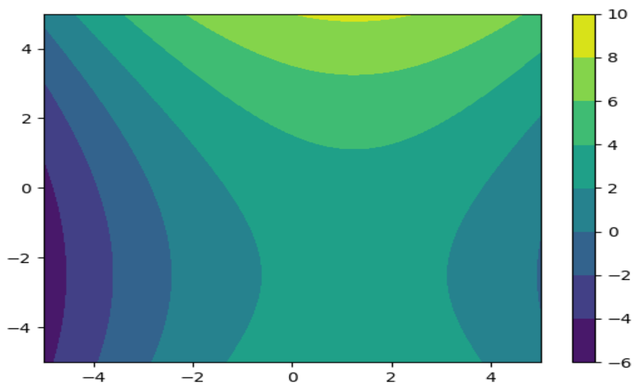
```
x = linspace(-5, 5., 100);
y = linspace(-5, 5., 100);
[X, Y] = meshgrid(x, y);
```



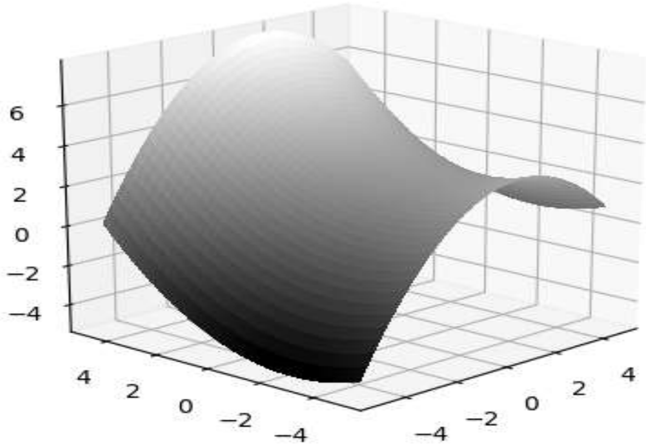

```
a = -0.2; b = 0.5; c=0.1;
d=0.5; e=0.; f=3.

f = a*X.^2 + b*X + c*Y.^2 + d*Y + e*X.*Y + f;

contourf(X, Y, f)
colorbar
```



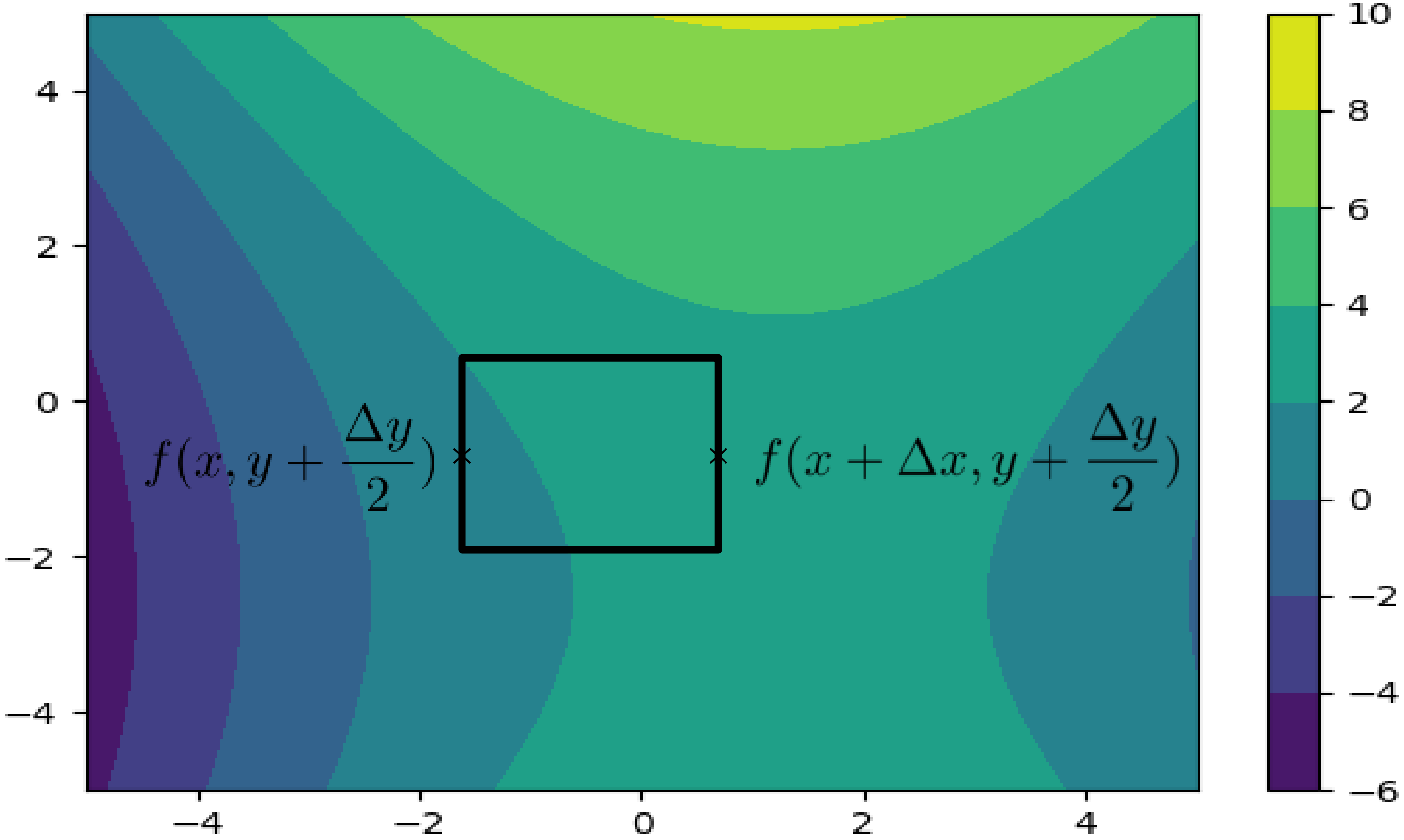

```
surf(X, Y, f) %To get 3D like plot seen previously
```



Defining an element of a 2D Field

- Contour plot

$$f(x,y) = ax^2 + bx + cy^2 + dy + exy + f$$



- Limit is a continuous function
- Here a function of two variables

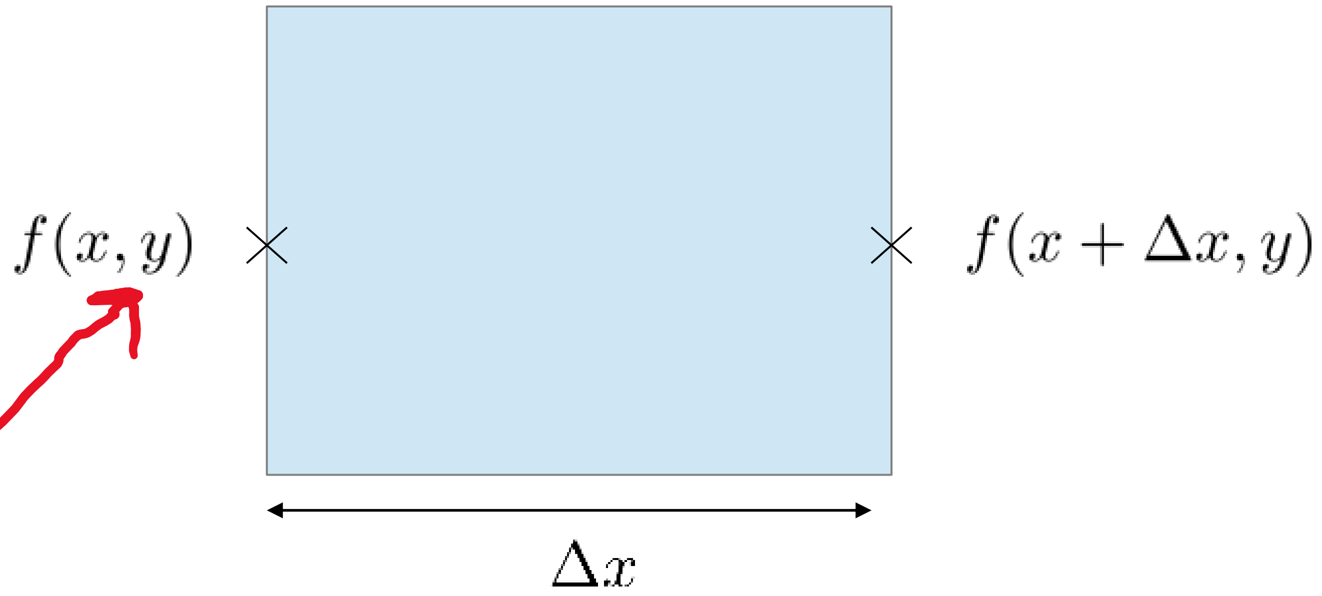
$$f = f(x,y)$$

- As we move in either x or y direction the value of f changes

Two Dimensions and Partial Derivatives

- Change in x keeping y constant (taken half way up element)

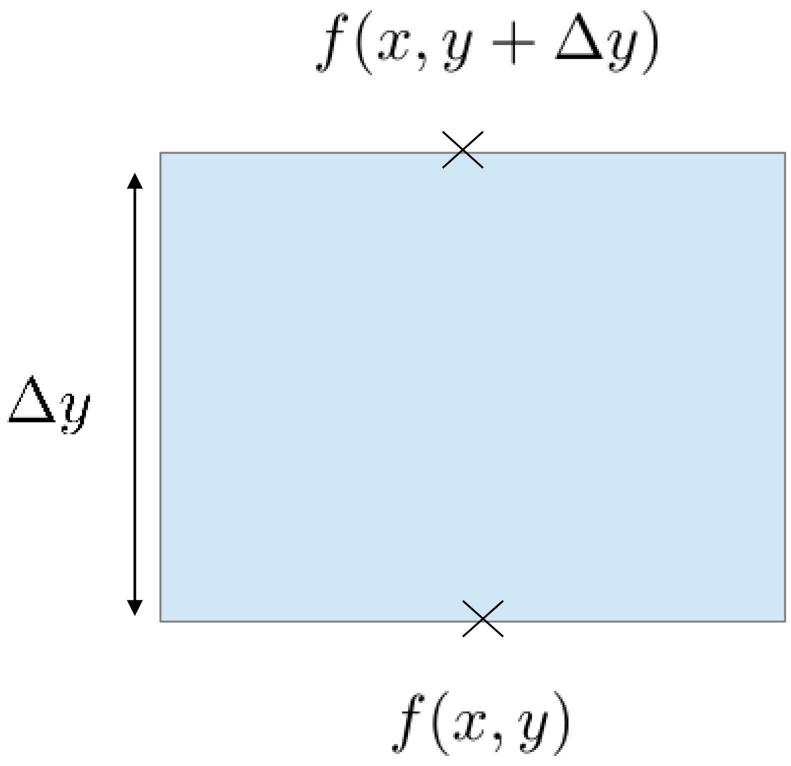
Should be
half way up
$+\frac{\Delta y}{2}$
but dropped
for simplicity



- Note we have dropped the $\Delta y/2$ terms for simplicity

$$\left.\frac{\partial f}{\partial x}\right|_{y\ \text{constant}} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x,y)-f(x,y)}{\Delta x}$$

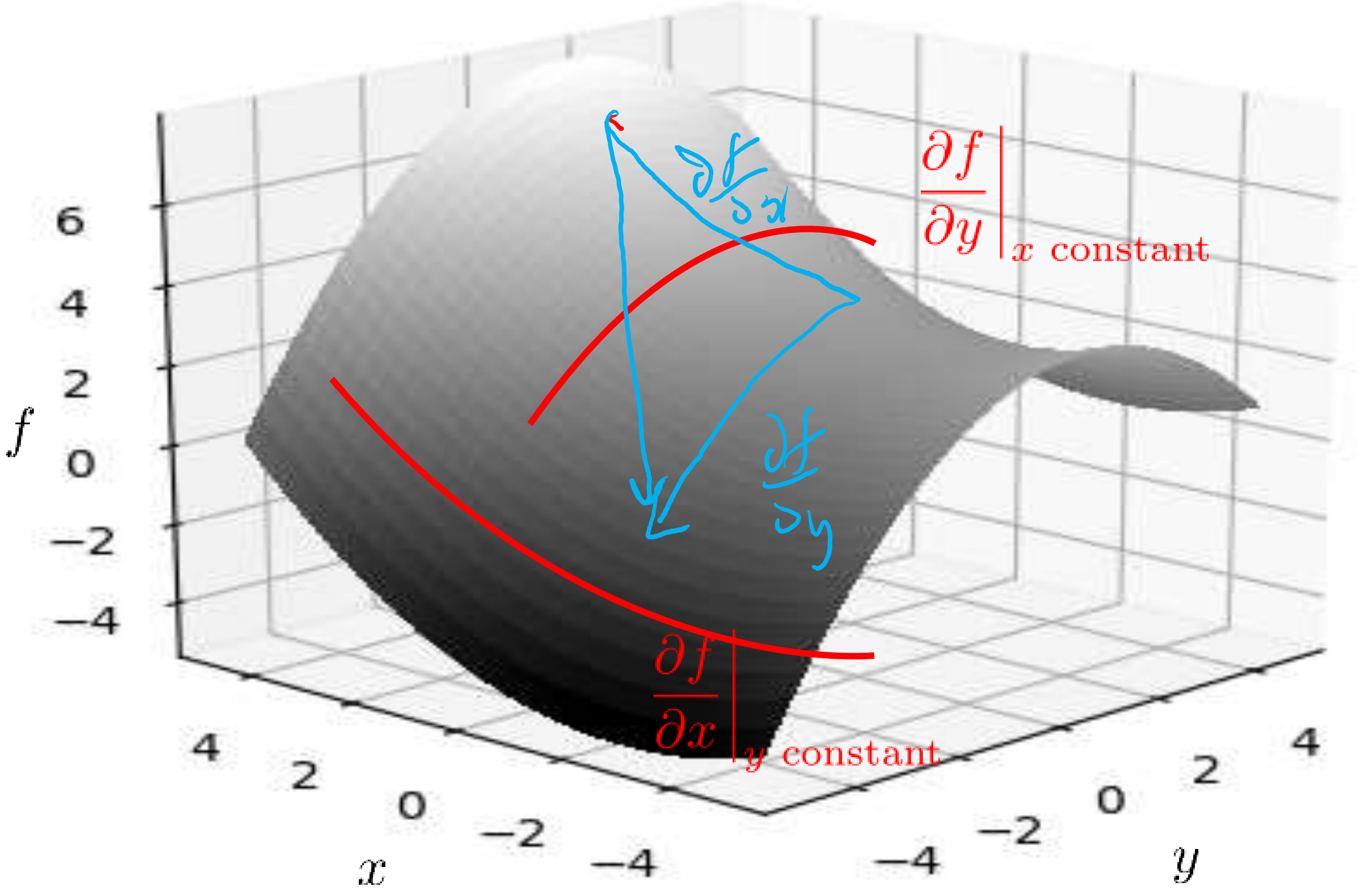
Two Dimensions and Partial Derivatives



- Note we have dropped the $\Delta x/2$ terms for simplicity

$$\left.\frac{\partial f}{\partial y}\right|_{x \text{ constant}} = \lim_{\Delta y \to 0} \frac{f(x, y + \Delta y) - f(x, y)}{\Delta y}$$

Two Dimensions and Partial Derivatives



Consider a function of two variables

$$f = f(x, y)$$

Change in x $\frac{\partial f}{\partial x}\Big|_{y \text{ constant}}$

Change in y $\frac{\partial f}{\partial y}\Big|_{x \text{ constant}}$

$$f(x, y) = ax^2 + bx + cy^2 + dy + exy + f$$

Partial Derivatives Numerical Solutions

- Previously we saw the first derivative for f=f(x) could be obtained from

$$\frac{df}{dt} \approx \frac{f(t+\Delta t) - f(t)}{\Delta t} = \frac{f^{t+1} - f^{t}}{\Delta t}$$

- We calculate the derivatives numerically in the same way as used in ordinary derivatives (note the superscript for time and subscript notation for space)

$$\frac{\partial f}{\partial t} \approx \frac{f(x, t+\Delta t) - f(x,t)}{\Delta t} = \frac{f_i^{t+1} - f_i^{t}}{\Delta t}$$

$t+1$

t

$$\frac{\partial f}{\partial x} \approx \frac{f(x+\Delta x, t) - f(x,t)}{\Delta x} = \frac{f_{i+1}^{t} - f_i^{t}}{\Delta x}$$

i $i+1$

Second Order Numerical Partial Derivatives

- Previously we saw the second derivative for f=f(x) could be obtained from

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

- The numerical approximation for partial second derivative of f=f(x,y) is obtained in the same way,

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f(x+\Delta x, y) - 2f(x,y) + f(x-\Delta x, y)}{(\Delta x)^2} = \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2}$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{f(x, y+\Delta y) - 2f(x,y) + f(x, y-\Delta y)}{(\Delta y)^2} = \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2}$$

Assessment Exercise – Combining Both

- Last Week you saw two examples of Partial Differential Equations
  - You assessed exercise is to combine them and provide a 2D finite difference solver

- Time Evolving in 1D

- 2D Spatial Equation

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

Combine

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

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

$$\frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2} + \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2} = 0$$

Assessment Exercise – Combining Both

- Last Week you saw two examples of Partial Differential Equations
  - You assessed exercise is to combine them and provide a 2D finite difference solver

- Time Evolving in 2D Spatial Equation

$$\frac{\partial u}{\partial t} = \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right]$$

Diffusion Equation

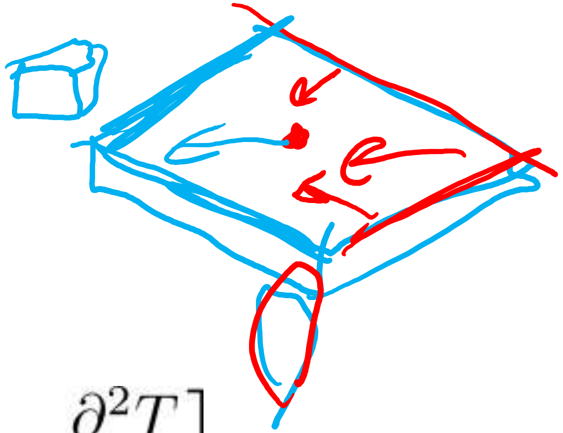
- Models the 2D diffusion as time evolves due to the viscosity coefficient using the sum of second order partial derivatives in x and y :

$$\frac{\partial u}{\partial t} = \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right]$$

- Note we have dropped the x=constant, y=constant for notational conciseness, but they are always implied by partial derivatives
- This equation describes the final state for the process of diffusion of a substance, such as ink in water, temperature in a metal, stress state in a material or concentration of a chemical in a mixture. It can also be solved to determine electromagnetic fields, potential fluid flow or gradients of pressure

Diffusion Equation

- Models the 2D diffusion as time evolves due to the viscosity coefficient using the sum of second order partial derivatives in x and y :



$$\frac{\partial u}{\partial t} = \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right]$$

- This equation describes
  - Spread of ink in water,
  - Temperature in a metal
  - Concentration of a chemical in a mixture

$$\frac{\partial T}{\partial t} = k \left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right]$$

$$\frac{\partial C}{\partial t} = k \left[\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2} \right]$$

Assessment Exercise – Combining Both

- Last Week you saw two examples of Partial Differential Equations
  - You assessed exercise is to combine them and provide a 2D finite difference solver

- Time Evolving in 2D Spatial Equation

$u(x, y, t+\Delta t)$

$$\frac{\partial u}{\partial t} = \nu \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right]$$

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = \nu \left[\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{\Delta x^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{\Delta y^2} \right]$$

Assessment Exercise – extend this to 2D

- Time Evolving diffusion in 2D

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = \nu \left[\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{\Delta x^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{\Delta y^2} \right]$$

- Rearrange to get next timestep from previous

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \nu \Delta t \left[\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{\Delta x^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{\Delta y^2} \right]$$

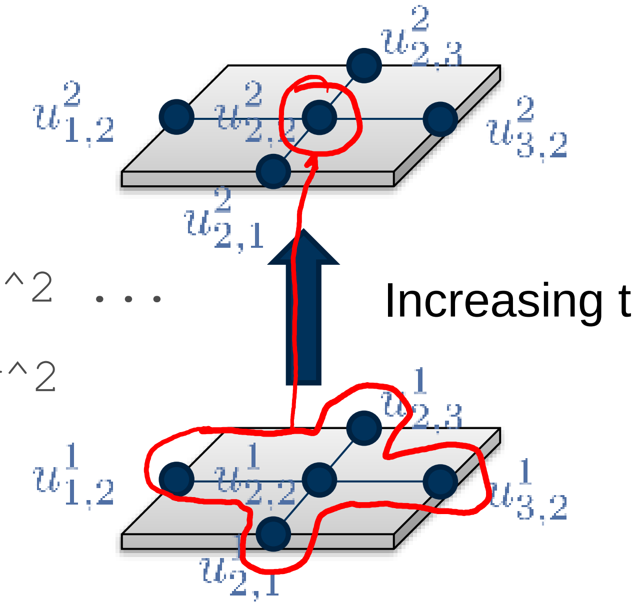
Assessment Exercise – extend this to 2D

- Rearrange to get next timestep from previous

$$u_{i,j}^{t+1} = u_{i,j}^{t} + \nu\Delta t\left[\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{\Delta x^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{\Delta y^2}\right]$$

- Expressed in code

```
utp1(i,j) = u(i,j) + nu*dt*( (u(i+1,j)-2*u(i,j)+u(i-1,j))/dx^2 ...
                            +(u(i,j+1)-2*u(i,j)+u(i,j-1))/dy^2
```

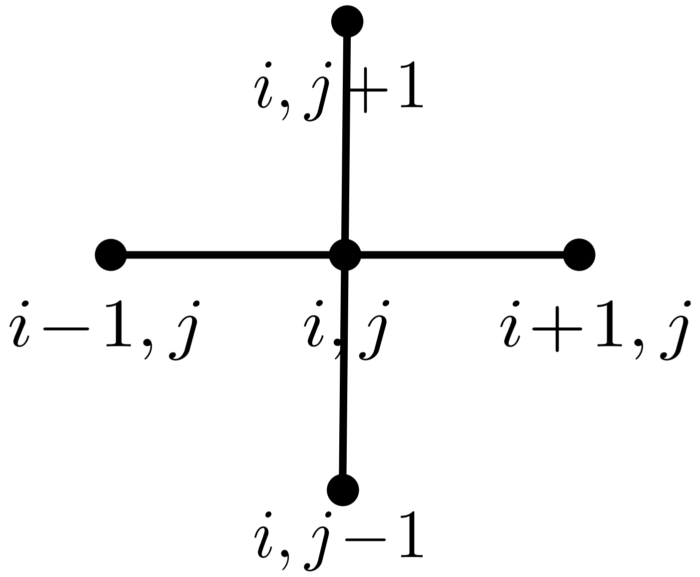


Second Order Numerical Partial Derivatives

- Using cell indices, derivatives in each direction can be seen to use what is called a five point “stencil”

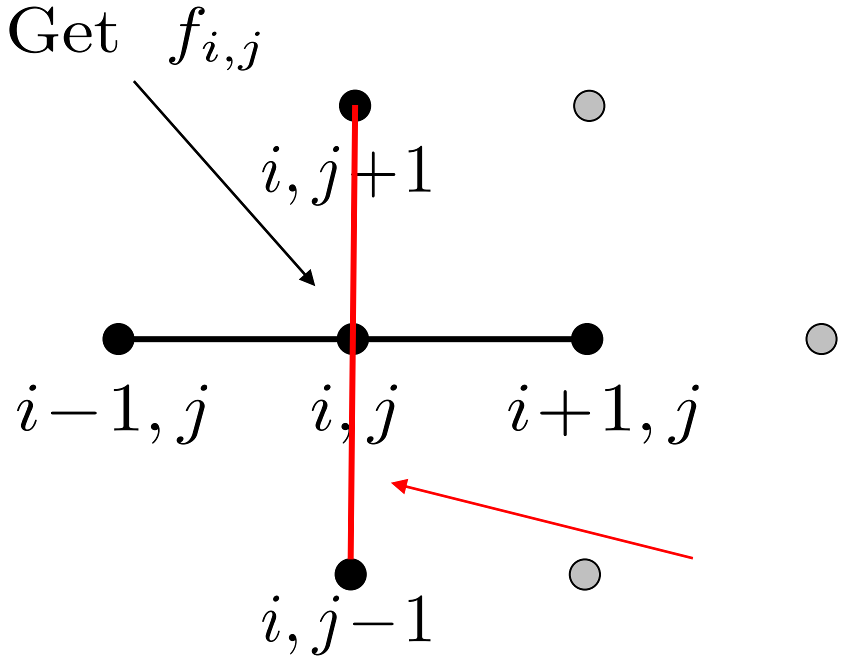
$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2}$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2}$$

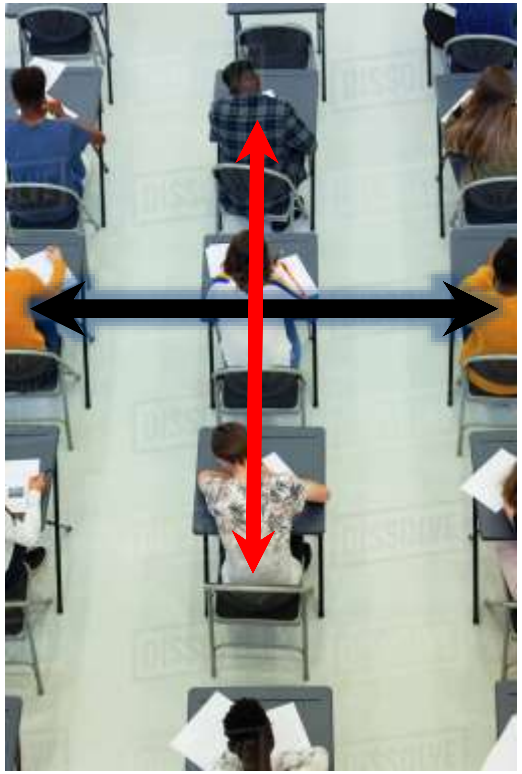


Solving 2D Finite Differences

- For a 2D grid we exchange points in both x and y

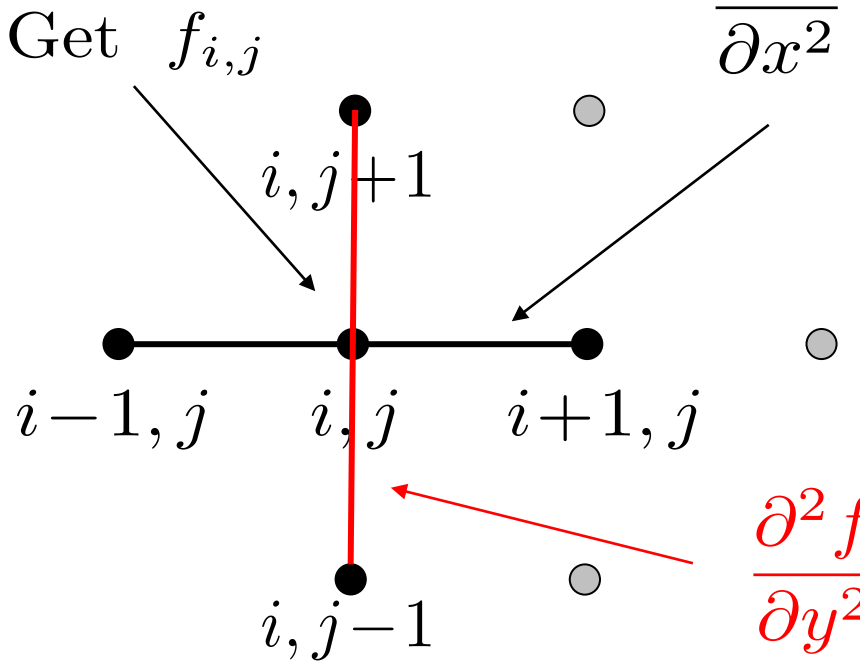


The first ever "CFD" used students to do the calculations and pass the results in 4 directions



Solving 2D Finite Differences

- For a 2D grid we exchange points in both x and y



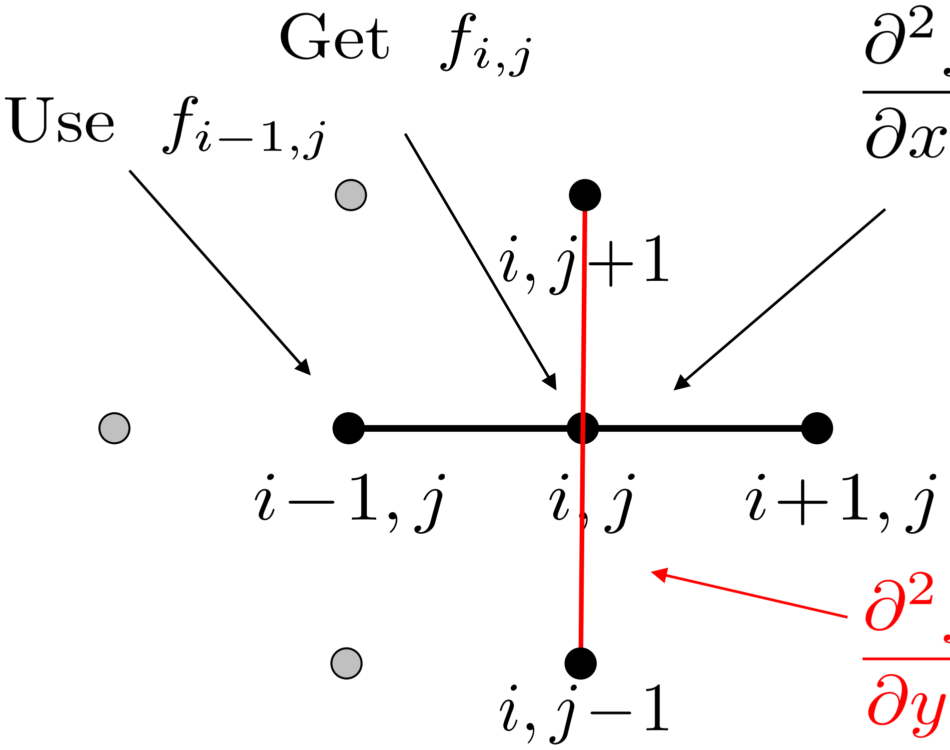
$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2}$$

Now we do in on a computer looping over i and j **(here i=1)**

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2}$$

Solving 2D Finite Differences

- Then we move to the next point



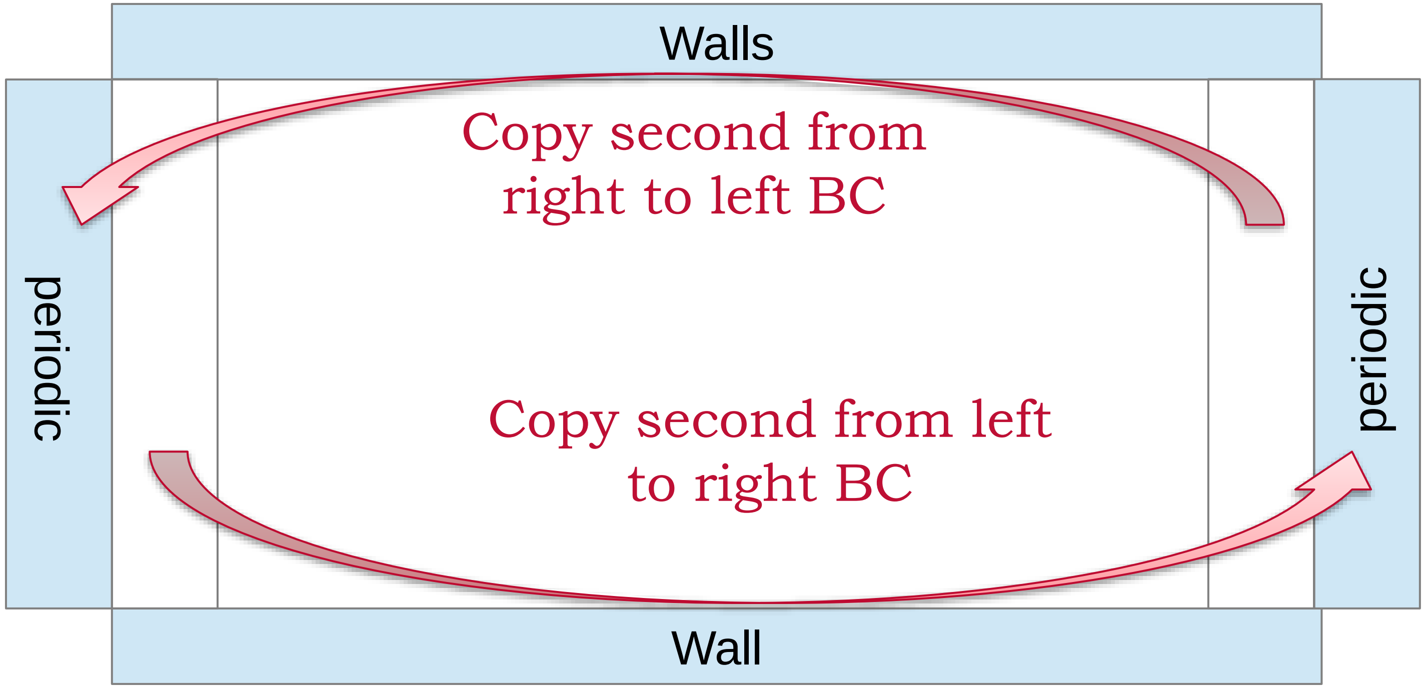
$$\frac{\partial^2 f}{\partial x^2} \approx \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\Delta x)^2}$$

Now we do in on a computer looping over i and j **(now i=2)**

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\Delta y)^2}$$

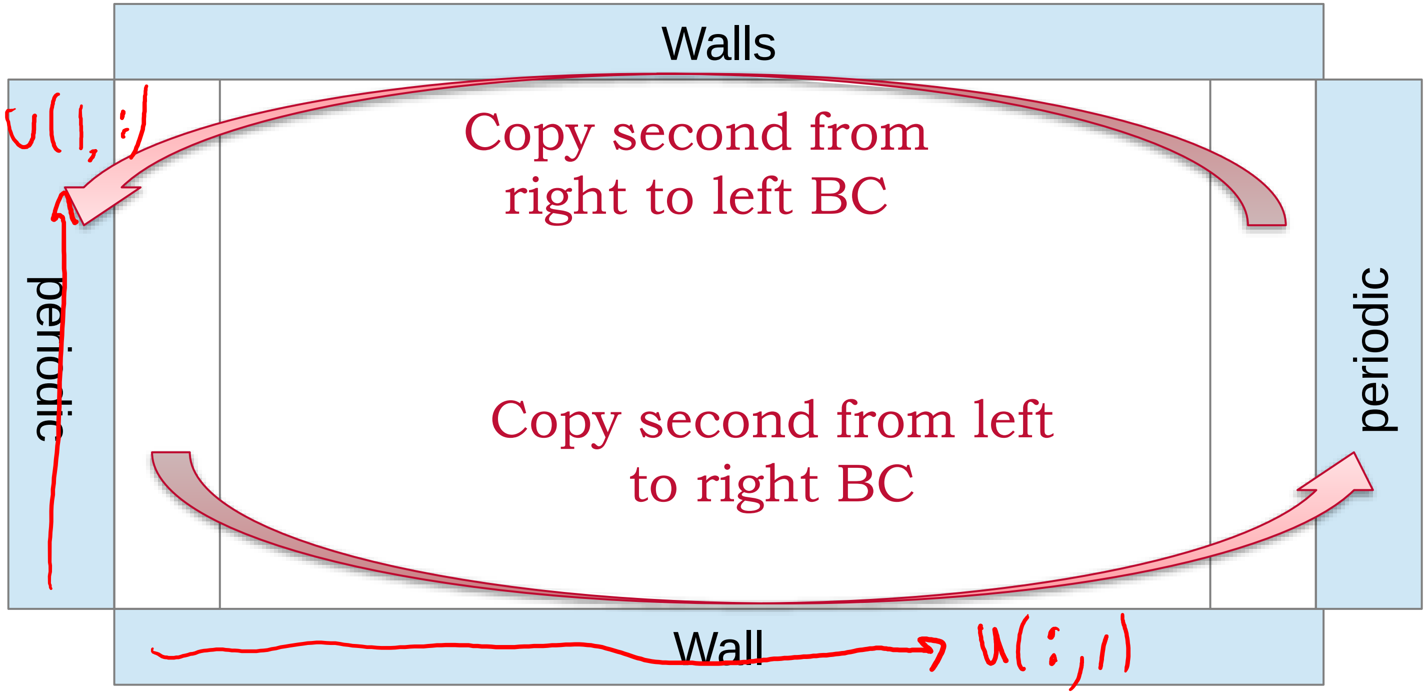
Boundary Conditions

- We need to choose values for the 4 boundaries to model different physical cases, for example an infinite fluid between two plates



Boundary Conditions – Infinite Channel

- We need to choose values for the 4 boundaries to model different physical cases, for example an infinite fluid between two plates

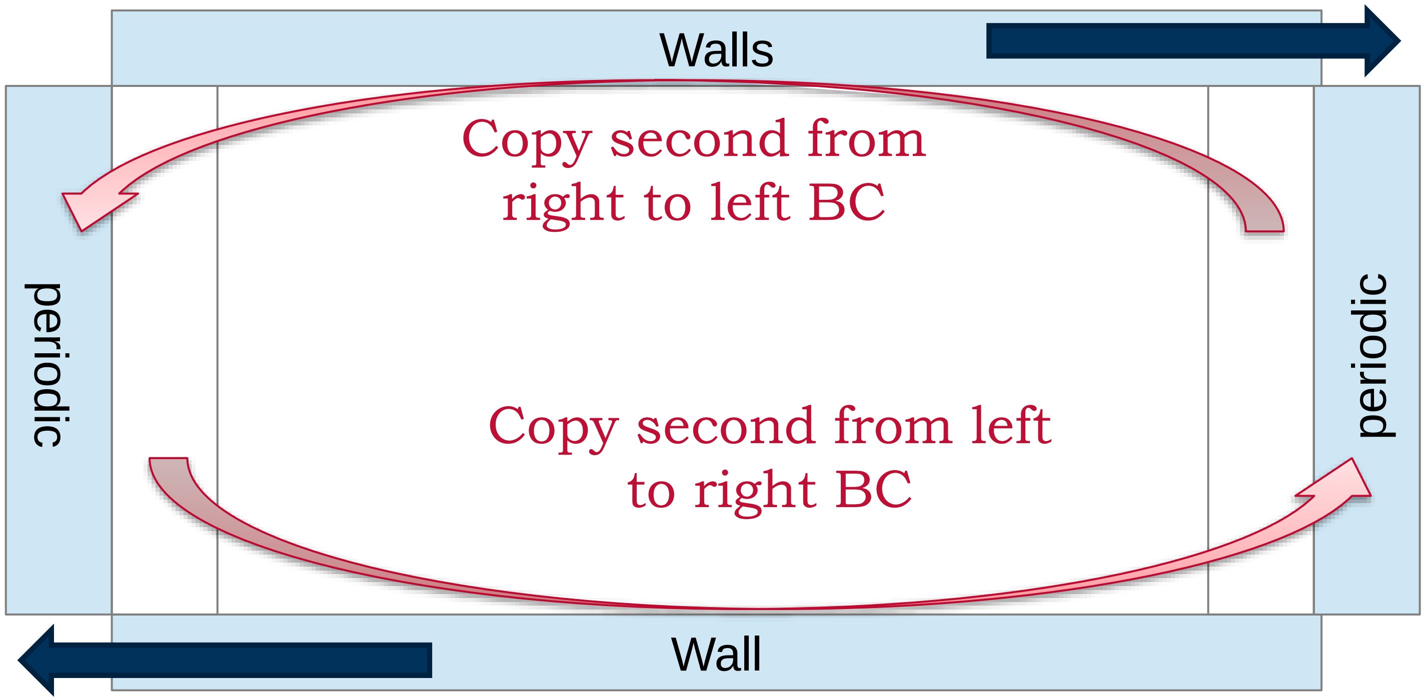



```
%Enforce Boundary Condition

%Bottom Wall Boundary
u(:,1) = 0;
%Left periodic BC
u(1,:) = u(end-1,:);
%Right periodic BC
u(end,:) = u(2,:);
%Top Wall Boundary
u(:,end) = 0;
```

Boundary Conditions – Counter-Sliding Walls

- We need to choose values for the 4 boundaries to model different physical cases

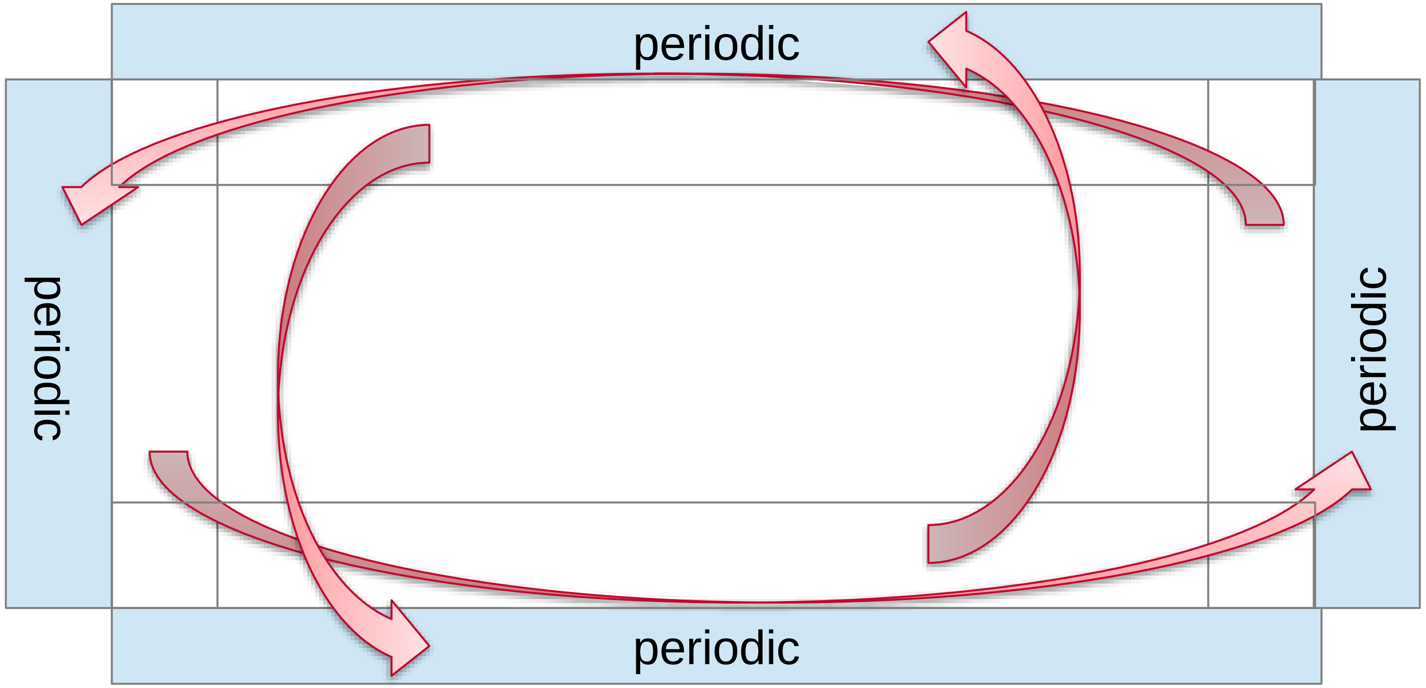



```
%Enforce Boundary Condition

%Bottom Wall Boundary
u(:,1) = -1;
%Left periodic BC
u(1,:) = u(end-1,:);
%Right periodic BC
u(end,:) = u(2,:);
%Top Wall Boundary
u(:,end) = 1;
```

Boundary Conditions – All Periodic

- We need to choose values for the 4 boundaries to model different physical cases



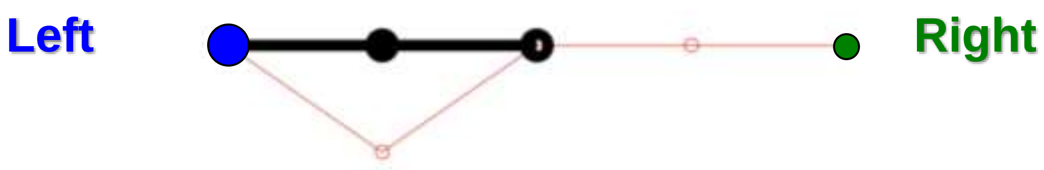

```
%Enforce Boundary Condition

%Bottom Wall Boundary
u(:,1) = u(:,end-1);
%Left periodic BC
u(1,:) = u(end-1,:);
%Right periodic BC
u(end,:) = u(2,:);
%Top Wall Boundary
u(:,end) = u(:,2);
```

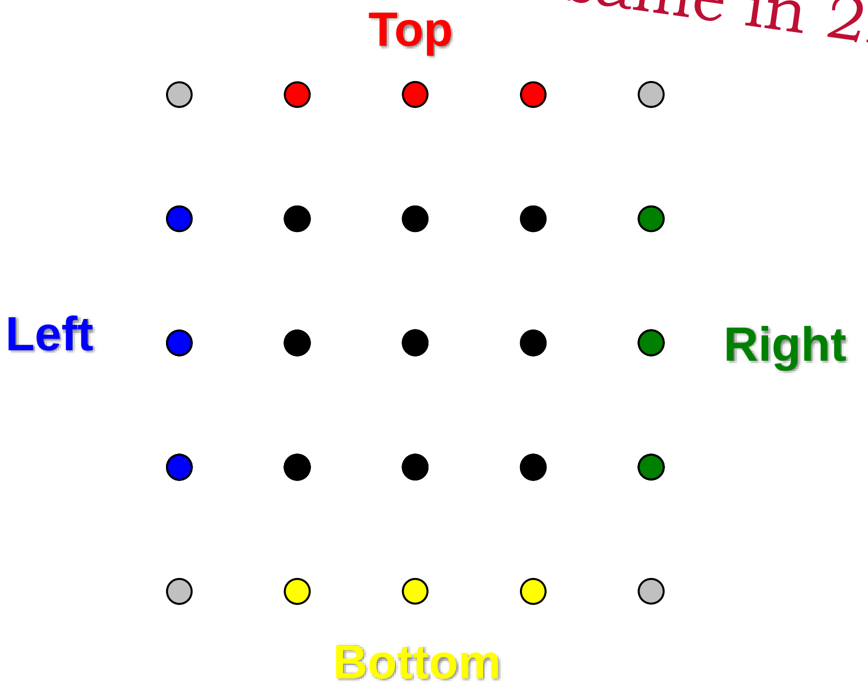
Now Evolving in Time

- Recall in 1D example, each time step `for t=1:100` we move from left to right. Then repeat at the next timestep.

Example in 1D



We do the same in 2D



Now Evolving in Time

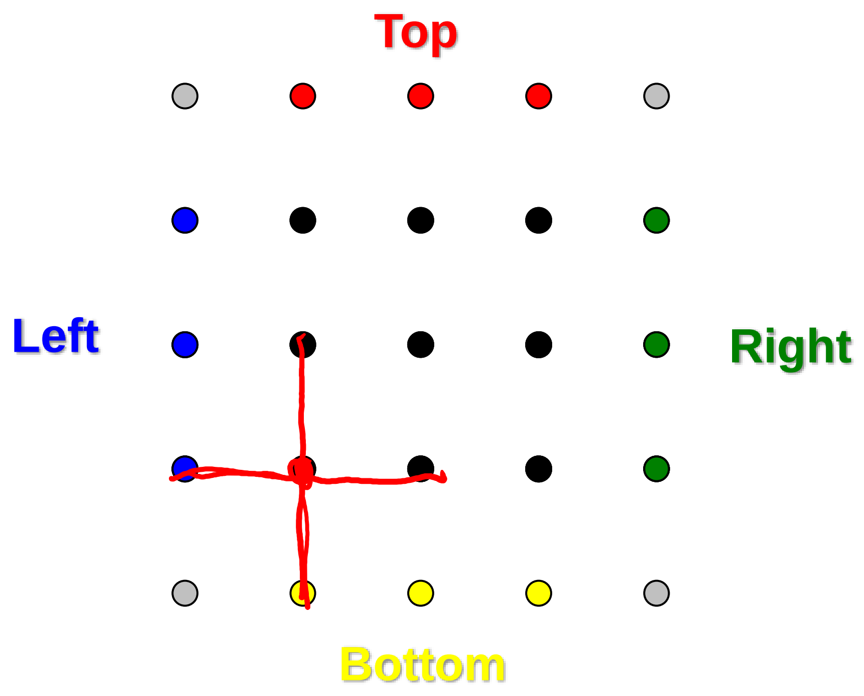
- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1

    end
end
```

i = 2

j = 2



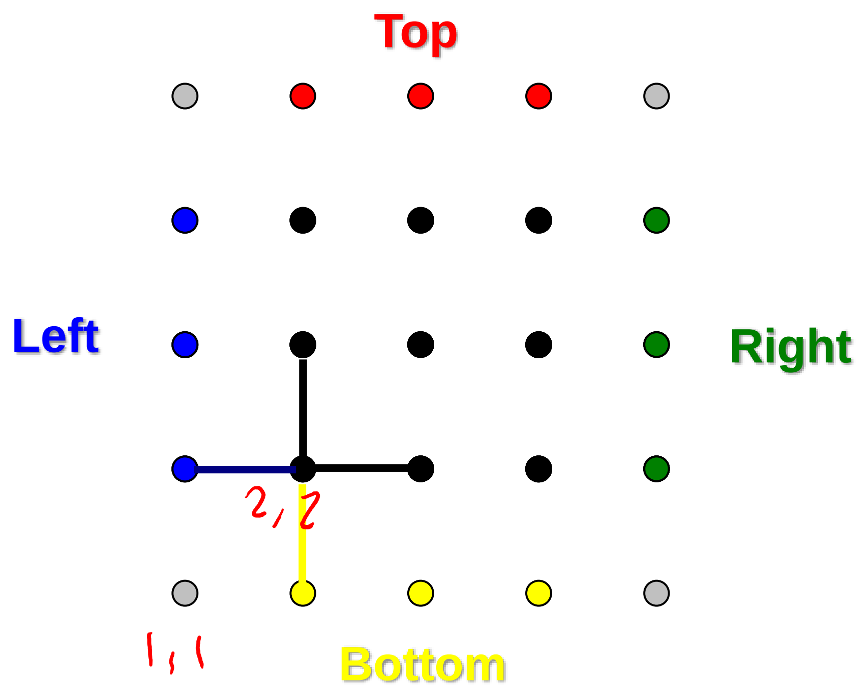
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1

        i=2
        j=2

    end
end
```



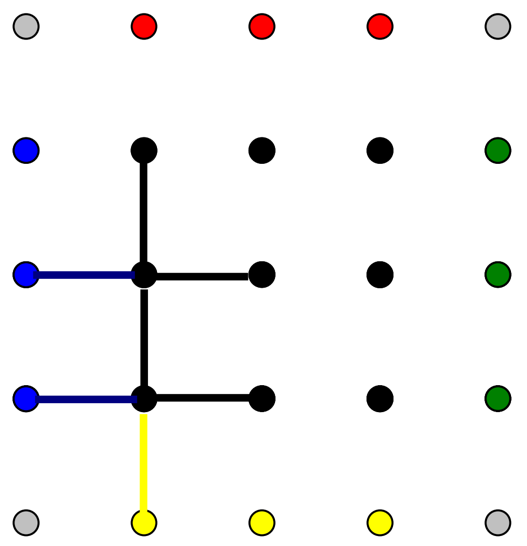
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1

        j=3
        i=2

    end
end
```



Now Evolving in Time

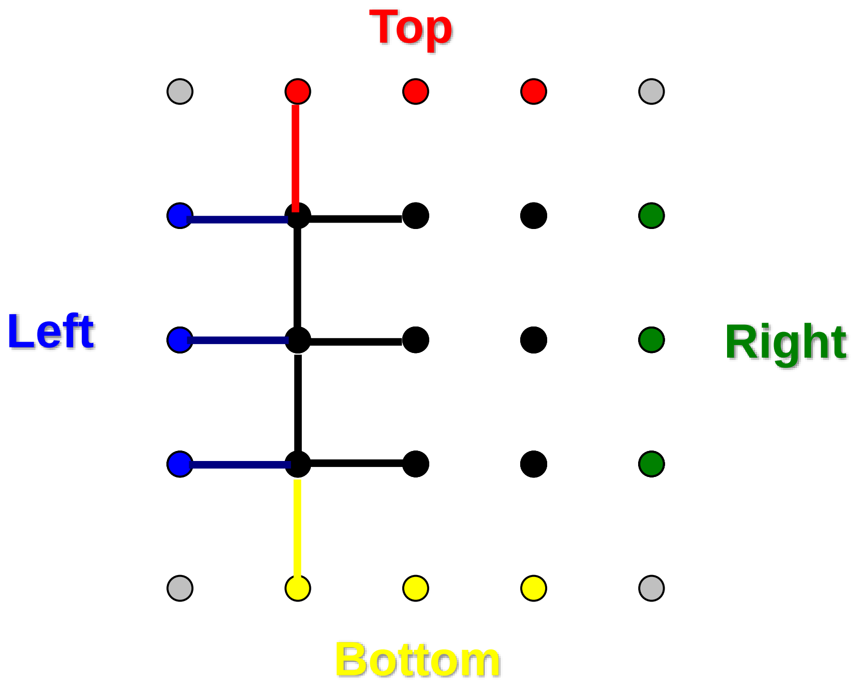
- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1

    end
end
```

j=4

j=2



Now Evolving in Time

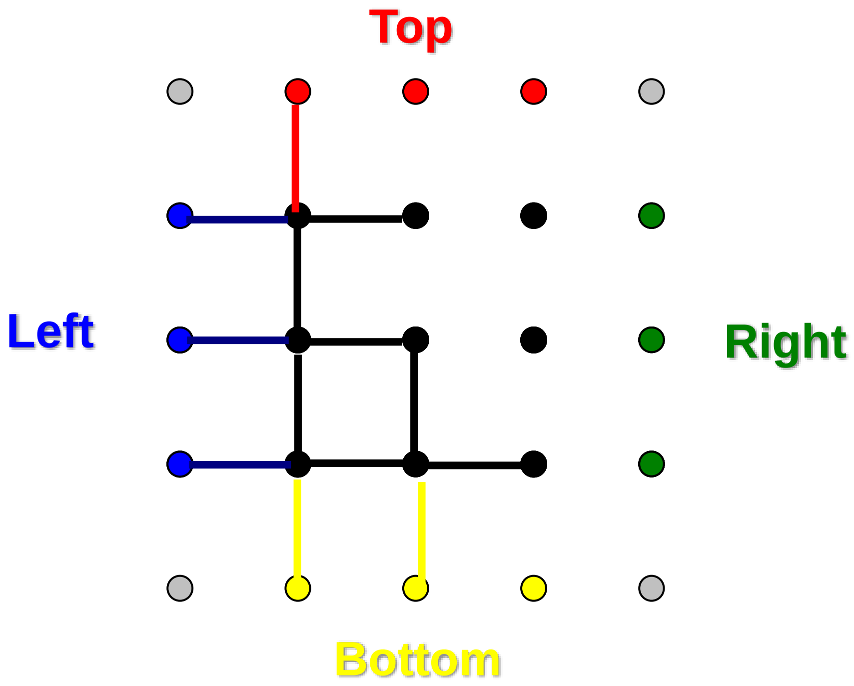
- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1

        j=2

         i=3

    end
end
```



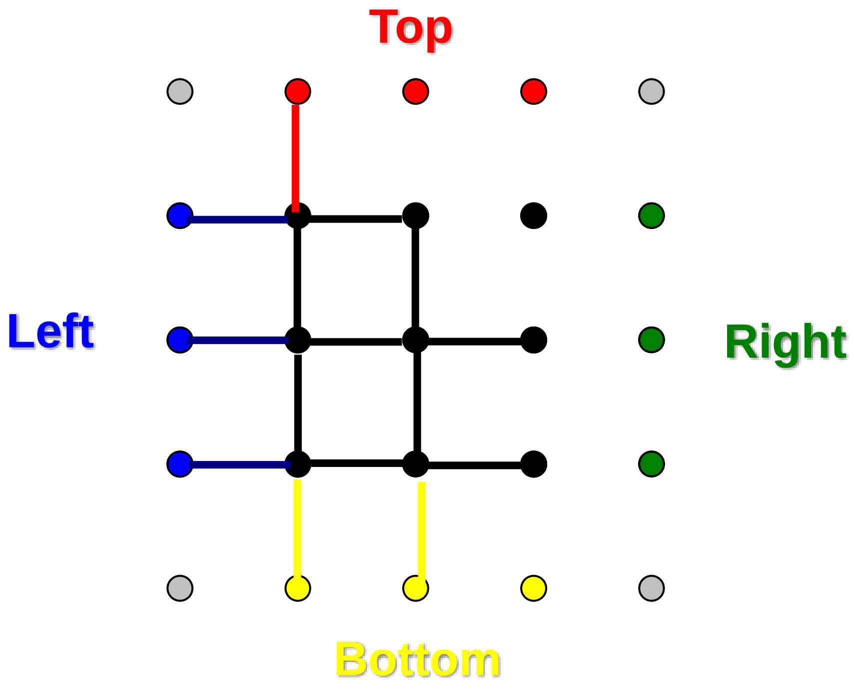
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1



    end
end
```



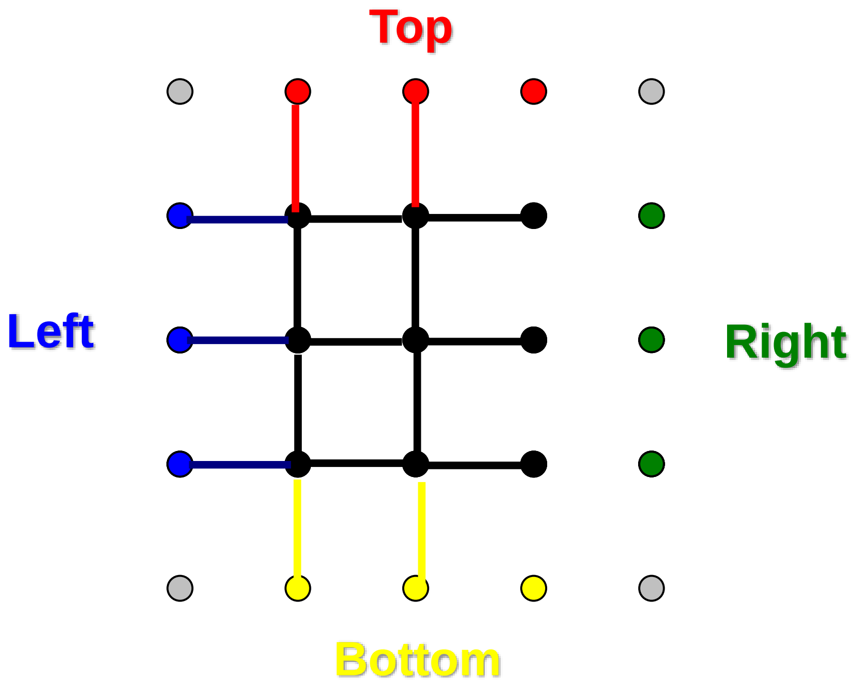
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1



    end
end
```



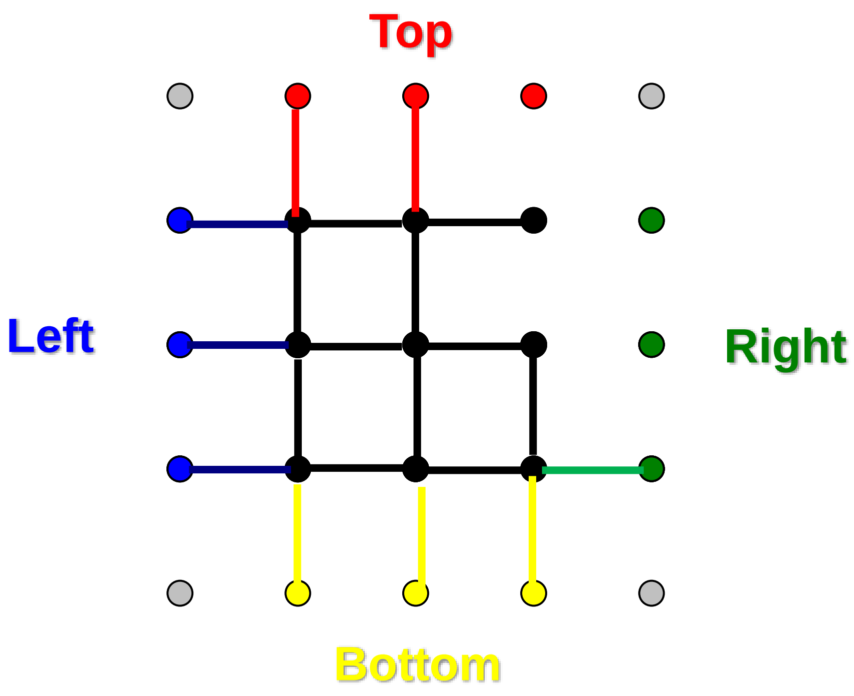
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1



    end
end
```



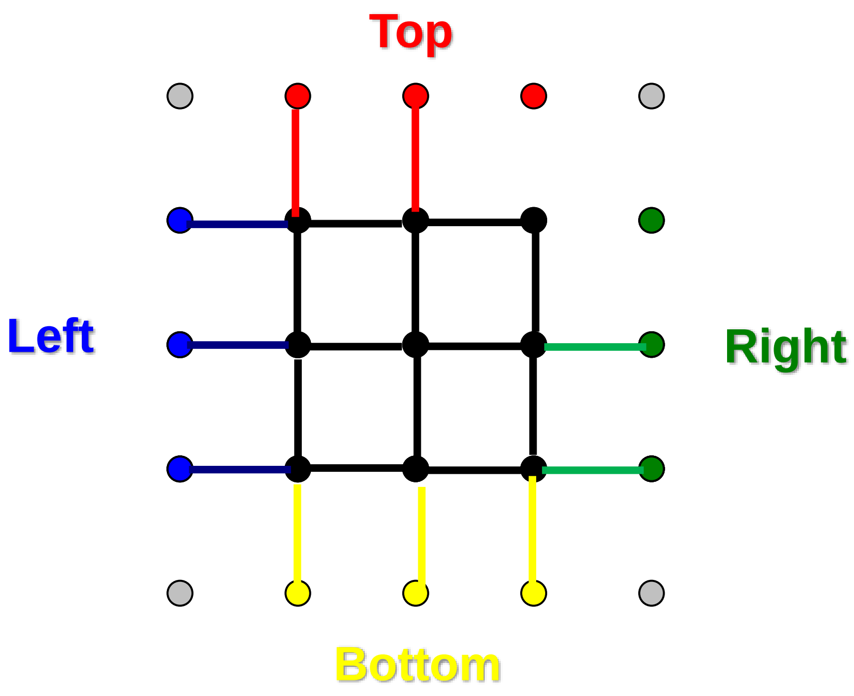
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1




    end
end
```



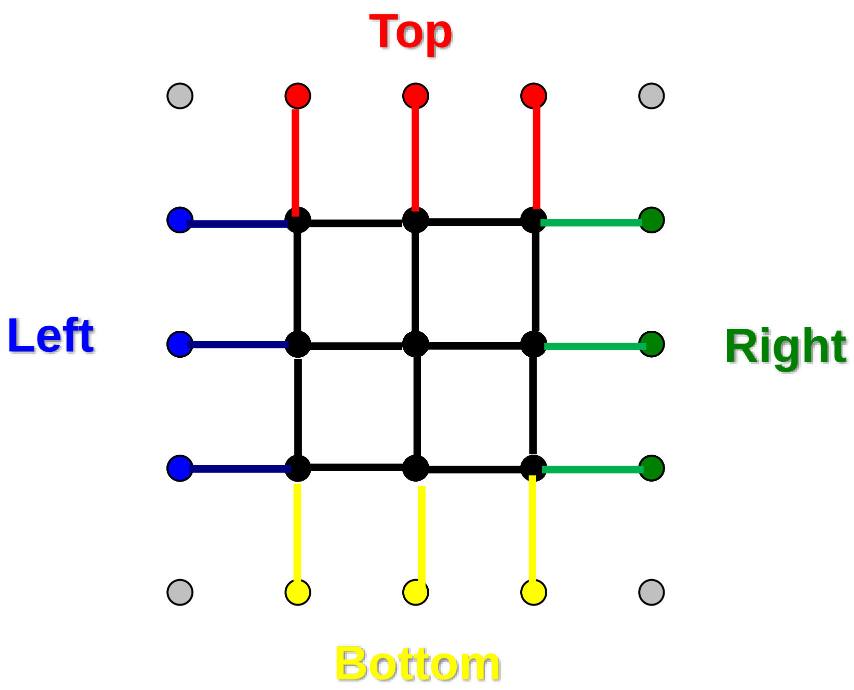
Now Evolving in Time

- Notice we use points either side so start from the edge of our domain (boundary)
- Boundary conditions based on what we know about the flow

```
for i=2:M-1
    for j=2:M-1



    end
end
```



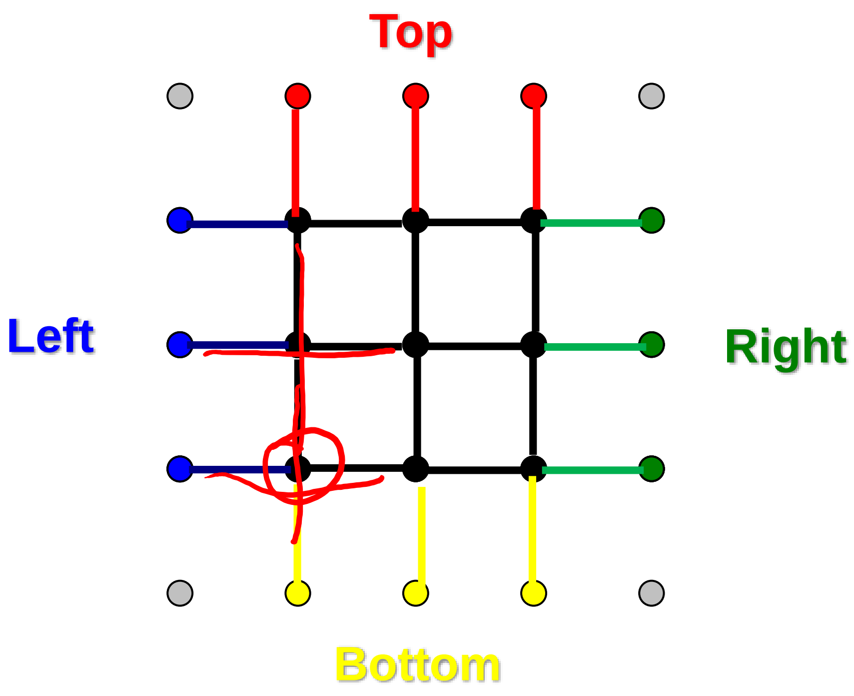
Now Evolving in Time

- Now we move on to the next timestep (t=2)

```
for t=1:100
    for i=2:M-1
        for j=2:M-1
```

t=2
i=2
j=2

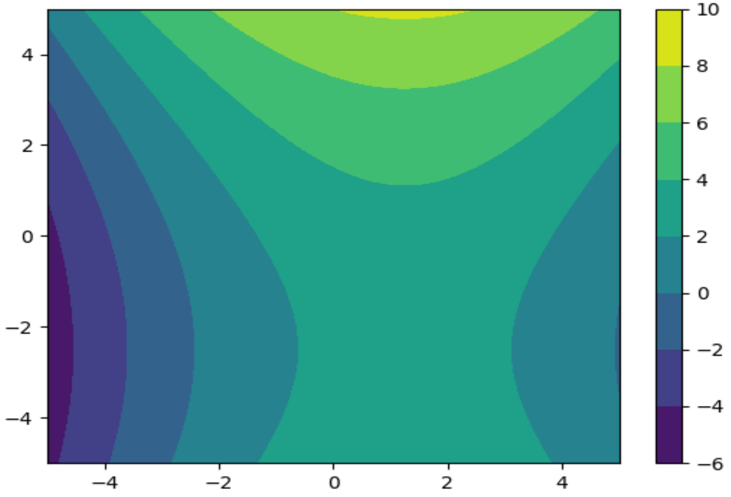
```
        end
    end
end
```



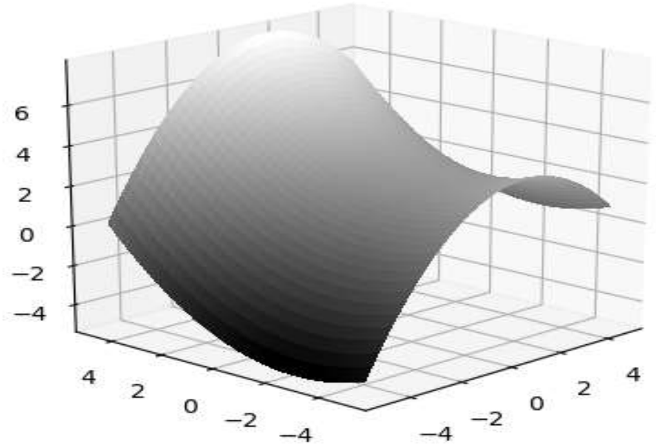
Plotting a 2D Field in MATLAB

- As the fields evolve, you should plot to make sure they look correct

```
contourf(u)
colorbar
```




```
surf(u)
```



Best Practice
and
Some Hints for the Assignment

Recap Functions and Interfaces

- Think of a function like a contract with the user – if you give me something, I will return something else

  - User provides number N to sum over
  - Function returns the sum

```
function a = sumtoN(N)
    a = 0;
    for i=1:N
        a = a + i;
    end
end
```

- An engineered part– has a defined role/interface, we aim to consider all possible uses and design so no need to change

Testing Functions

- It is good practice to write tests for functions to check expected behaviour

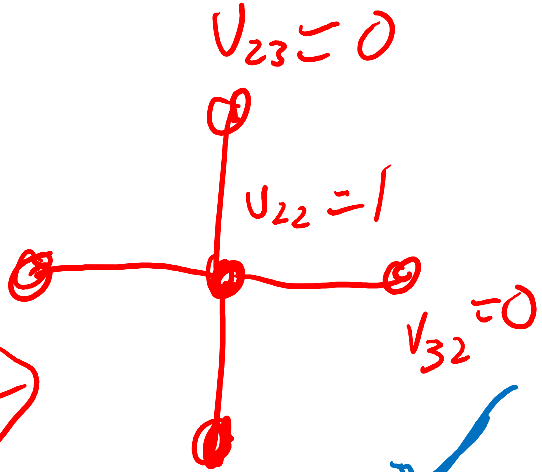
```
assert(divide(1,2) == 0.5)
assert(divide(1,3) == 0.3333333333333333)
```

- As a engineer, this concept should be familiar as a form of quality assurance
  - These tests should be automatic in code, so every function is ensured to work whenever any software development is done
  - Also important to catch potential problems or misuse of a function
- These can be inside the function (assertions) or outside as unit tests
  - Inside is similar to a diagnostic light (designed for a trained technician or the user)
  - Outside is closer to quality control tests on the product

Functions for Assessment

- The 2D solver will live in a function `solve_unsteady_diff`

```
function u = solve_unsteady_diff(...
        uinitial, Lx, Ly, Mx, My, ...
        nu, maxIter, dt, ...
        xperiodic, yperiodic, ...
        tBC, bBC, lBC, rBC, ...
        showplot)
```



$u_{i+1,j} - 2u_{i,j} + u_{i-1,j}$

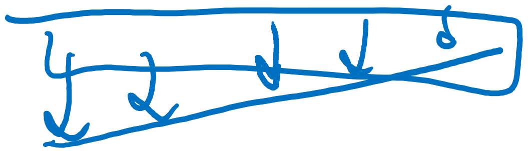
- The inputs aim to provide all needed functionality
- Testing can be applied using different inputs and checking how returned u changes, e.g. `u(5,5)=1; maxIter=1; dx=dy=nu=1` against a hand calculation.

$dx = 1$ $dy = 2$

Validation and Verification

- Verification is the process of making sure your code is free from basic errors
  - This involves checking that the mathematics is implemented correctly with hand calculations or similar
  - Making sure that the possible use cases give the expected answer
  - All code contains bugs, this attempts to limit them by splitting code into lots of pieces (e.g. with functions) and testing every bit of the code in these pieces using unit tests
- Validation

$T = 1$



$T = 0$

  - A test on the whole software product
  - Ensure that the software meets the requirement of the user and gives the required result
  - In engineering, this could be a test to ensure the computer model agrees with the real world (e.g. through an experiment or against analytical theory)

Test Driven Development (TDD)

- This is the “gold standard” of software development, you would write test first before writing the code the pass the test
  - Very useful in software development teams, to ensure a manager can specify all the bits and how they fit together, and ensure they work correctly
  - This is how MATLAB grader works, the required functionality is specified by the tests and you must write a function which passes these tests
- A simple example, a function to square a number could be required to satisfy two test,

```
assert(square(2)==4)

assert(square(3)==9)
```

```
function y = square(x)
    y = x^2;
end
```

- Very similar to engineering where we would develop the specification before we start designing a product – you have been given this specification in grader

Version Control

- A shared file system for code which keep track of the changes
  - The most popular is Git (and Github hosting) but also mecurial and subversion
  - Every change is linked to the person who made it
  - It is possible to go back to working or “stable” branches
- Can be linked to automatic testing so every change is checked when submitted to ensure it does not break the code,
  - e.g. if I accidently change square function

```
function y = square(x)        assert(square(2)==4) → Will raise an
    y = x^1;                                           Assertion failed.
end
```

General Modelling Guidelines

- Keep it simple – more complex model, more chance of errors
  - Can you model just a small part of the problem?
  - Can you use a 1D to validate the 2D model?
- Build up the complexity, adding more terms and features but always checking the solution is correct at each stage
  - Look at the solution, does it make physical sense?
  - Does the solution change when you add more elements (change mesh resolution)?
  - Can you match to analytical results, other modelling methods or previous solutions?
  - Can you match to experimental results?

Types of Error

- Common types of errors, in order of frequency
  - Coding Error – mistyped variable, unexpected function (note very common in your own code, very rare in commercial and open-source projects)
  - Input Errors – Wrong or incompatible boundary conditions, wrong magnitude (units) for coefficients
  - Numerical Errors – Poor mesh resolution, quality or shape, badly conditioned stiffness matrix
  - Judgement error – Inappropriate model of reality, missing important term

Recall in GRADER 2a – Array Slicing

- You defined a 2D matrix, extracted elements and multiplied them

```
A = [1 1 2 3 5 8;
       0 2 4 6 8 10;
     -1 -3 -5 -7 -9 -11;
       2 4 8 16 32 64;
     13 21 34 55 89 144];

%Multiply 1st column & last column of A
D = A(:,1) .* A(:,end);
```

Variables - A

A

5x6 double

	1	2	3	4	5	6
1	1	1	2	3	5	8
2	0	2	4	6	8	10
3	-1	-3	-5	-7	-9	-11
4	2	4	8	16	32	64
5	13	21	34	55	89	144

- You need to be able to get and set the minimum and maximum rows/columns of a 2D array for boundary conditions

Recall in GRADER 2b - Ensuring Inputs are Correct

- For a quadratic equation solver, check inputs and document

```
function out = quadratic(a, b, c)
    validateattributes([a,b,c],{'numeric'},{'size',[1,3]})
    D = b^2 - 4*a*c;
    out(1) = (-b + D^0.5)/(2*a);
    out(2) = (-b - D^0.5)/(2*a);
end
```

Recall in GRADER 3b - Ensuring Errors Raised

- One number of loops exceeded, raise an error

```
% Iteration limit is exceeded.
error('findroots:IterationLimitExceeded','The iteration limit was exceeded.')
```

Recap – Error Checking

- You can check the inputs are correct with ValidateAttributes

```
classes = {'numeric’}; attributes = {'size',[1,2]};
validateattributes([Lx, Ly], classes, attributes);
```

- You should use if statements to check array size against the Mx and My values and raise an error Statements as follows uinitial(Mx, My)

```
error("solve_unsteady_diff:DomainSizeError", ...
       "uninitial should be array of size Mx by My")
```

- Recall the form of the error statement (first string) must be exactly as above

Recap – Logical Statements

- Conditional statements
  - Check logical statements, e.g. a==3
  - Can also directly check a flag, if a variable is true of false
- Used to set boundaries to periodic

```
 %Enforce Boundary Condition
if (xperiodic)

else

end
```

1 0

true or false

Recap – Showing plot in a Function

- Pass in showplot logical argument

```
if (showplot)
        contour(u')
        ...
```

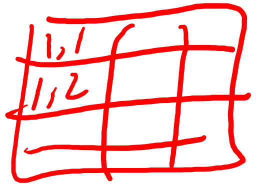
pause(0.01)

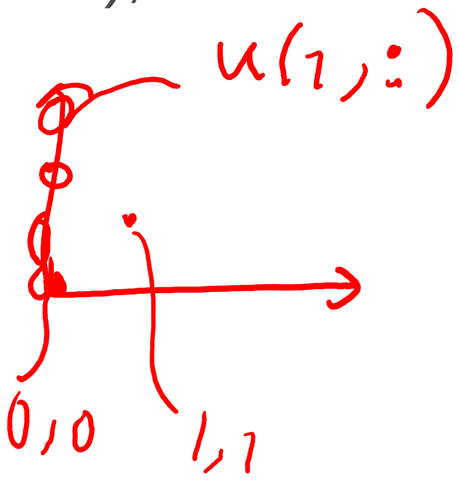
- Can be a bit more clever and specify frequency of plot with modulo arithmetic (not that showplot=false is pretty much the same as showplot=0 so stops plots)

```
if (mod(t,showplot) == 0)
        contour(u')
        ...
```

Recap – Array Conventions

- In mathematics, we start from the bottom left as zero (by convention), in MATLAB, we start from the top left.





- Creating an array (setting to filled with zeros) is done with
  - zeros(Nx, Ny) where Nx is number of x points and Ny is number in y
  - However, MATLAB assumes zeros(no. of rows, no. of columns)

zeros(4,2) →

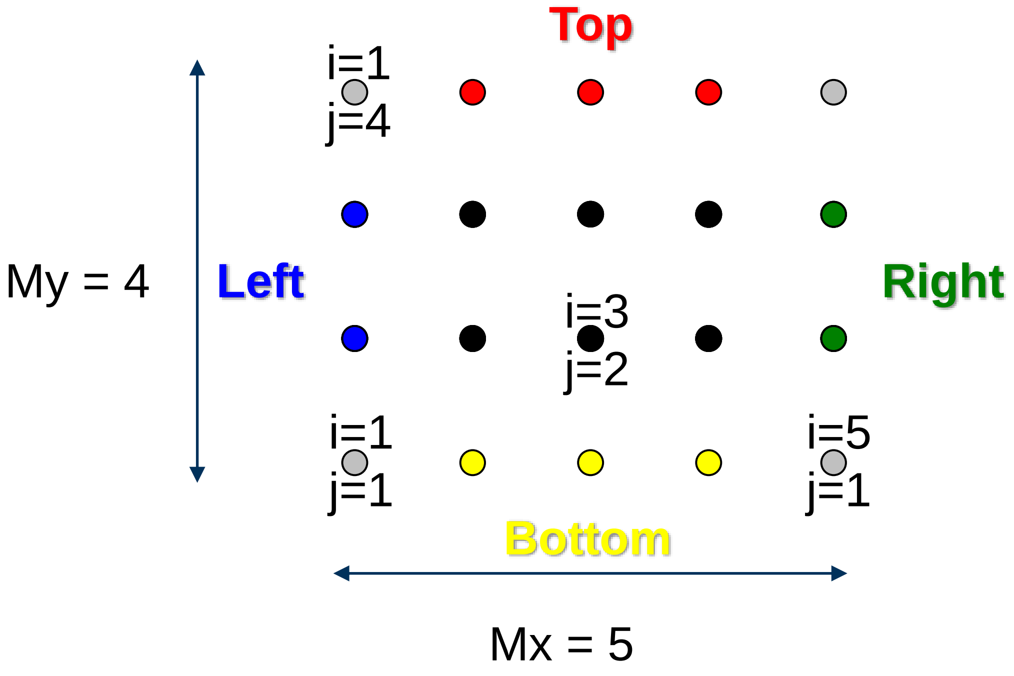
- We could work through and use flipud but instead we transpose at the end

Recap – Array Conventions

- In mathematics, we start from the bottom left as zero (by convention)
- MATLAB top left (and zeros defines rows in first argument, then columns)

```
Mx = 5; My = 4;
u = zeros(Mx,My)
```

```
%Define some example values
u(1,1) = 1; u(1,4) = 2;
u(3,2) = 3; u(5,1) = 4
contourf(u'); colorbar; %Use ' for transpose
```

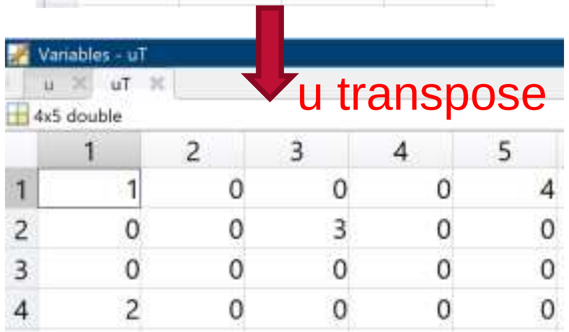



Variables - u

u

5x4 double

	1	2	3	4
1	1	0	0	2
2	0	0	0	0
3	0	3	0	0
4	0	0	0	0
5	4	0	0	0

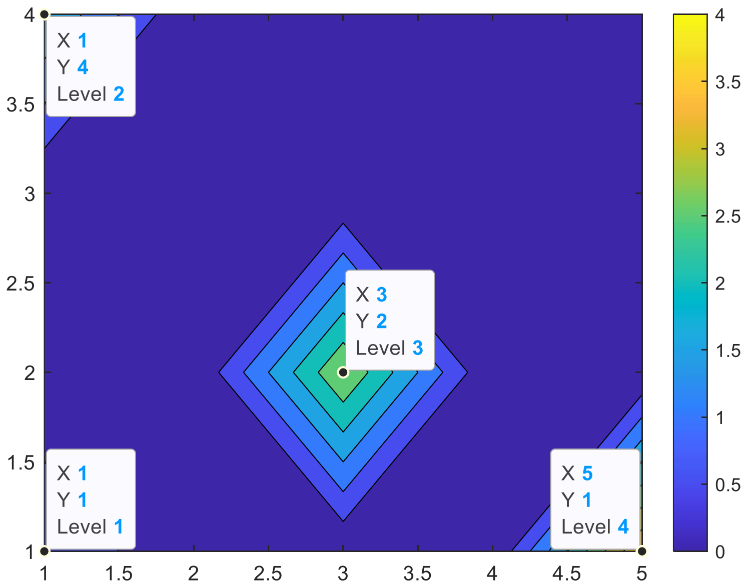

Variables - uT

u uT

4x5 double

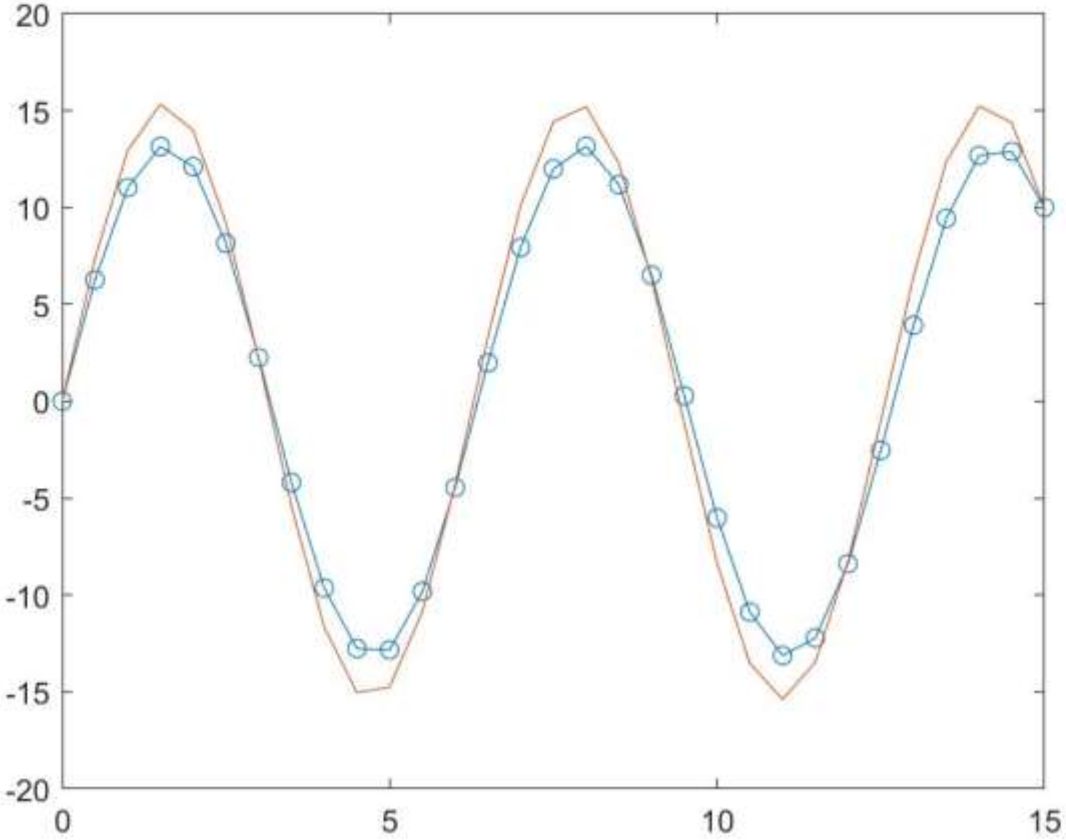
u transpose

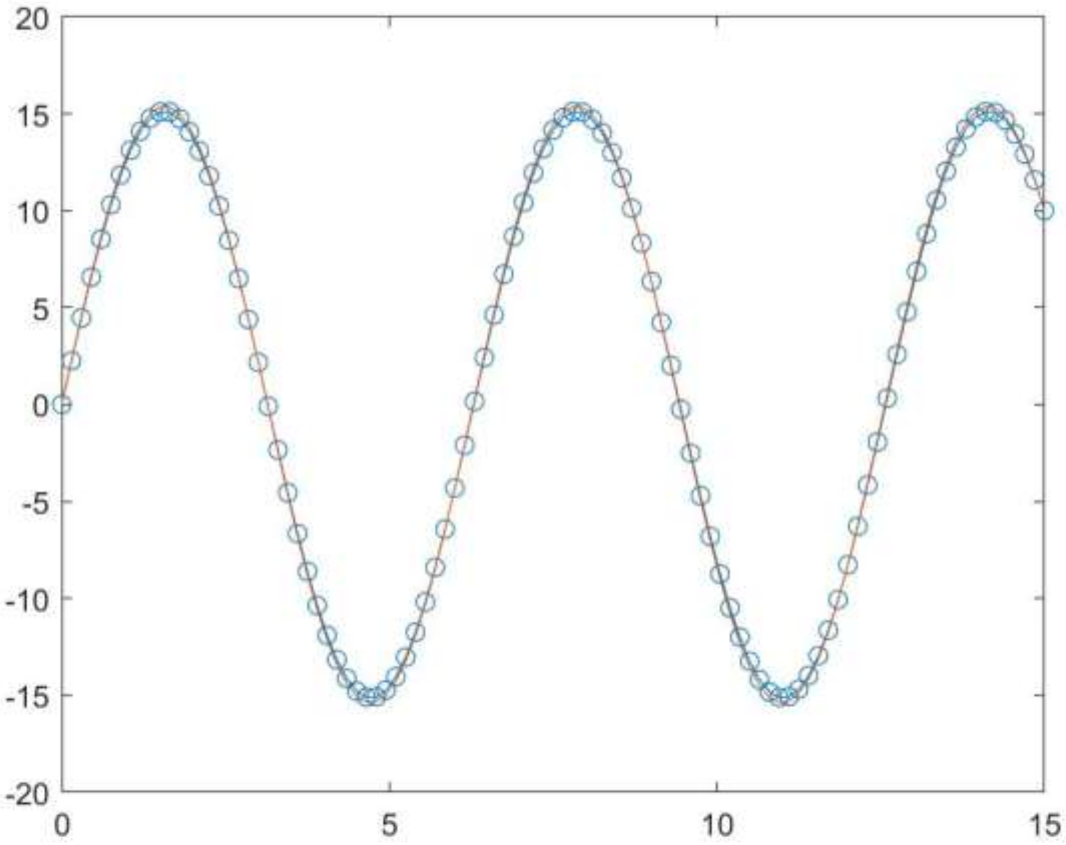
	1	2	3	4	5
1	1	0	0	0	4
2	0	0	3	0	0
3	0	0	0	0	0
4	2	0	0	0	0



Extra Ideas - Mesh Resolution

- As we use more points, we get closer to the “true” solution



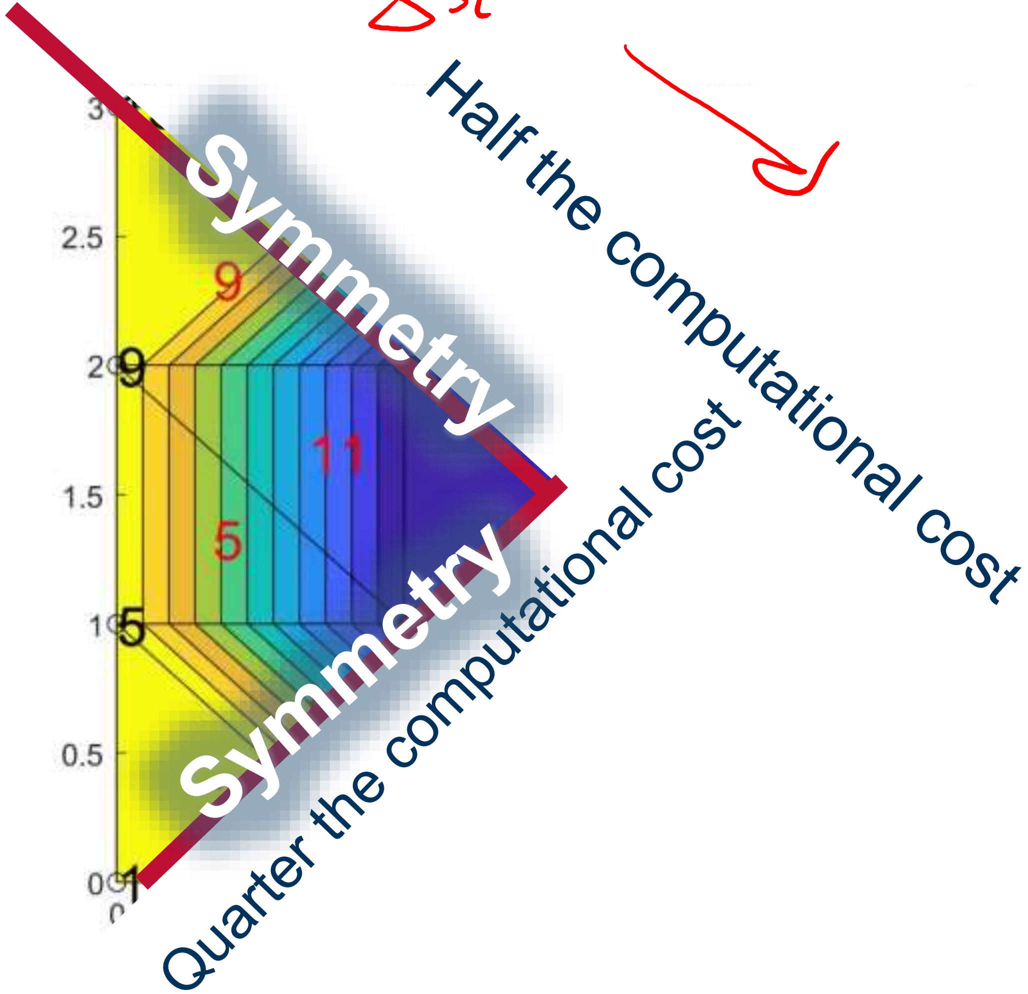


Extra Ideas - Other Boundary Conditions

- Direct (Dirichlet)
- Fluxes (Neumann)
- Mixed (Robin)
- Symmetry
- Periodic
  - Go out one side
  - Come back in other

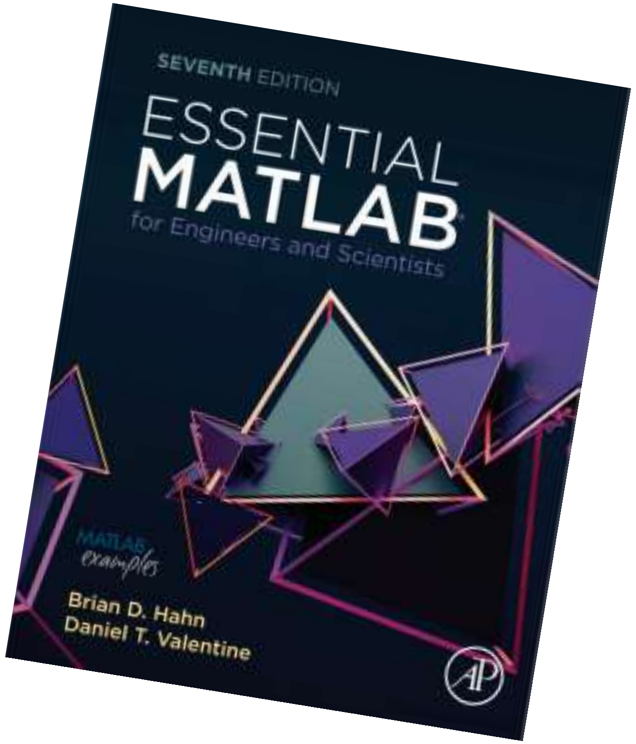
$$\frac{\partial u}{\partial x} = a$$

$$\frac{u(i+1) - u(i)}{\Delta x} = a$$



Summary

- Recap of Partial Differential Equation
  - Temporal-spatial and spatial in two dimensional
  - Some boundary and initial conditions
- Combining both for the assessment exercise
  - A two dimensional time evolving field
  - Boundary conditions along the 4 sides
- Summary for Concepts needed for Assessment
  - Recap of functions, arrays and error checking
  - Best practice advice
  - Validation and verification



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

Partial derivative example quite complex in EM as they use implicit