Data-Driven Methods for Engineers (MECH0107) 2025 - 2026

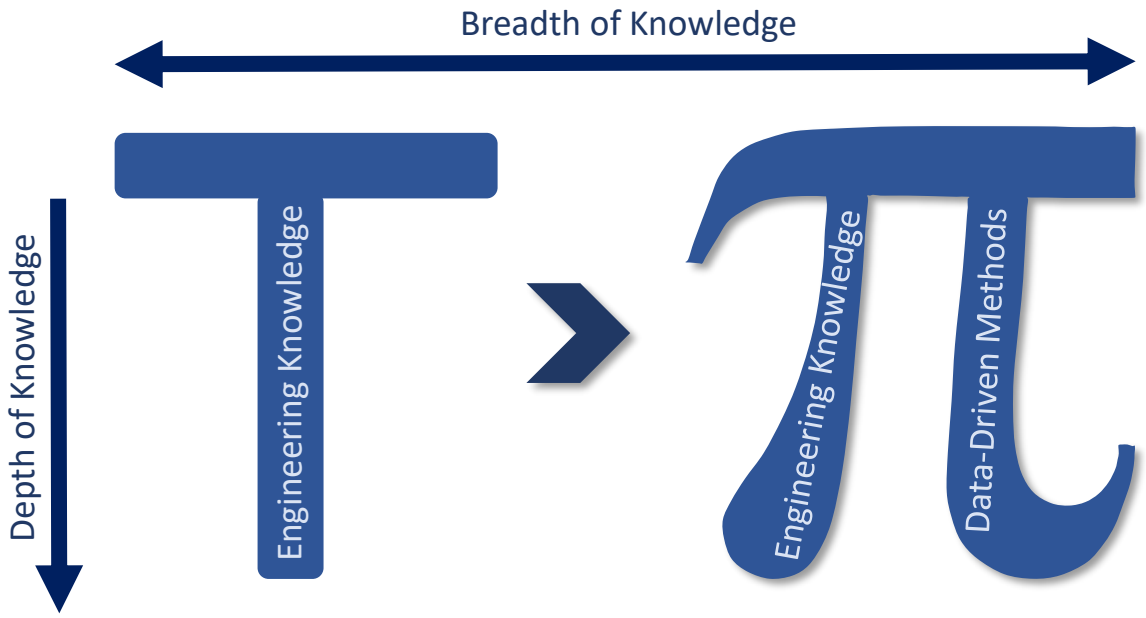
Dr Lama Hamadeh
Module Lead
Office 429 | Roberts Building
Mechanical Engineering Department
l.hamadeh@ucl.ac.uk

Dr Llewellyn Morse
Module Tutor
Office 503D| Roberts Building
Mechanical Engineering Department
l.morse@ucl.ac.uk

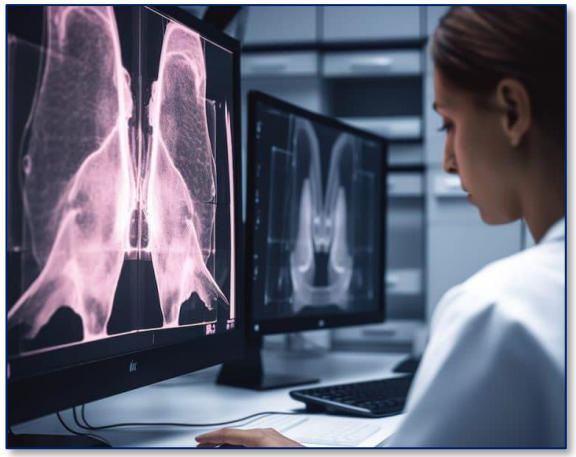
Introduction

- In the past, data on which science and engineering are based was scarce and frequently obtained by experiments proposed to verify a given hypothesis. Each experiment yielded only minimal data. Today, data is abundant and abundantly collected in each single experiment at a minimal cost.

- Data-driven modelling and scientific discovery is a change of paradigm on how many problems, both in science and engineering, are addressed. These methods have the additional advantage of testing correlations between different variables and observations, learning un-foreseen patterns in nature, and allowing us to discover new scientific laws.



Introduction – Data-Driven World!



Data-driven cancer detection uses large datasets of medical scans, patient histories, and clinical outcomes to train machine-learning models capable of identifying early signs of cancer. These algorithms learn subtle patterns in imaging data that may be difficult for humans to detect. By comparing new patient scans to thousands of past examples, the system can flag potential tumours earlier and more accurately, supporting radiologists and improving early diagnosis rates.

Renewable power output fluctuates due to changing weather conditions. Data-driven forecasting models learn patterns from historical and real-time sensor data to predict future energy production. Accurate forecasts help grid operators balance supply and demand, reduce reliance on fossil-fuel backup plants, schedule maintenance more efficiently, and increase overall grid stability.

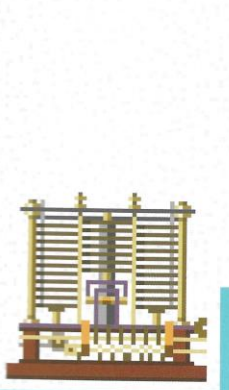
Supply chain and inventory optimisation: Data from sensors, logistics systems, and historical demand are used to predict delivery times, stock needs, and production rates. Financial models determine the cost of storage, delays, and shortages. Optimisation algorithms then suggest the most cost-effective inventory levels. This improves operational efficiency while reducing unnecessary financial expenditure.

Wearable health devices continuously collect physiological data like heart rate, movement, sleep patterns, and blood oxygen levels. These data are analysed using machine-learning algorithms that detect patterns, track long-term trends, and identify abnormalities in real time. This enables early warnings for issues such as irregular heart rhythms, stress, or declining activity levels, supporting preventive healthcare and personalised wellness monitoring.

Introduction – Historical Timeline

Computing

Before 1935, the word "computer" referred to a person whose job it was to do mathematical calculations. Today, a computer is a machine that takes input, stores and processes data, and gives output. Computers were initially built from mechanical components such as levers and gears, until electronic parts were used in the 20th century. Early computers were so huge they filled entire rooms. Now, the computing power of the smartphone in your pocket is just as powerful.



1822

Babbage's Difference Engine

British inventor Charles Babbage draws up plans for his Difference Engine – a mechanical device that can do complicated calculations. He imagines that this machine will store data in the future, anticipating the computer technology that is to come. Construction begins on the engine but is never completed.

1843

Ada Lovelace invents programming

Babbage shows maths whizz Ada Lovelace his idea for a mechanical computer. She works out how the computer can manipulate letters and symbols as well as numbers. Her idea was decades ahead of its time. Historians consider her the first computer programmer.

1890

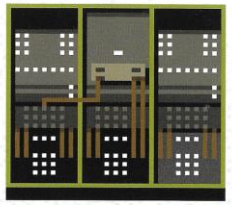
Punch card

American inventor Herman Hollerith designs a machine to calculate the size of the population in the 1890 US census. It uses electricity to read a pattern of holes punched in a card. His invention saves both time and millions of dollars.

1936

Turing machine

British computer scientist Alan Turing has an idea for a machine that can solve any problem that is solvable, leading to the development of the modern computer. During World War II, he devises a range of codebreaking tools to break Nazi codes.

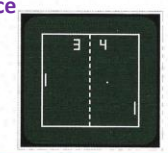


1946

ENIAC

John Mauchly and J. Presper Eckers, scientists at the University of Pennsylvania in the US, build one of the first general-purpose computers, called ENIAC. This massive 30-tonne computer glowing with 18,000 tubes is intended to make more calculations than all of humanity has up to this point.

Artificial intelligence (AI)



1956

1958

First computer game

Hundreds of people line up to play in New York in the US, when physicist William Higinbotham serves up what is said to be the first ever computer game. Two players use buttons and dials to bat a blob of light across a virtual court.

1959

Computer language

While working for the US Navy, maths expert Grace Hopper develops the first computer language, which becomes known as COBOL. This language is still in use today.

1971

Floppy disk

Engineers at tech company IBM invent the floppy disk, a small plastic envelope housing a bendy mylar (plastic) disk that lets users share data quickly. Billions of disks a year are sold.

1976

Cray supercomputer

The world's fastest computer is built. Its "C" shape reduces the time signals take to travel through the machine. It takes a year to assemble and costs about nine million dollars.

1994

First smartphone

IBM's Simon Personal Communicator is the world's first smartphone, created years before there is a name for the device. A touchscreen allows users to make calls, send and read emails, and manage diaries.

1984

Mac attack

Apple launches its Macintosh, the first successful computer controlled with a mouse. A pull-down menu makes it simple to use. Microsoft releases a mouse-friendly Windows operating system the next year.

1982

Rise of the PC

The Commodore 64 makes personal computers popular. Thousands of software programs, such as word-processors, spreadsheets, and games, are released for the machine, which is one of the highest-selling computers ever.

1976

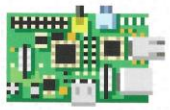
Apple I and Apple II

Steve Wozniak, Ron Wayne, and Steve Jobs create and sell one of the first desktop computers, Apple I, as a kit for computer enthusiasts. Apple II is created a year later, and appeals to a wider audience with its keyboard, game controllers, and the video game *Breakout*.

2010

First tablet

Microsoft releases the first tablet computer The device features a touchscreen, and car access games, music, email, and the internet, putting the power of a much larger machine in the user's hands.



2012

Raspberry Pi

This tiny and inexpensive computer introduces a generation of kids to computer programming through fun and easy projects that young coders upload and share with each other online.

2016

AR/VR goes mainstream

Smartphone users can add digital elements to a view on their camera with augmented reality (AR) technology, and can immerse themselves in artificial digital environments with virtual reality (VR).

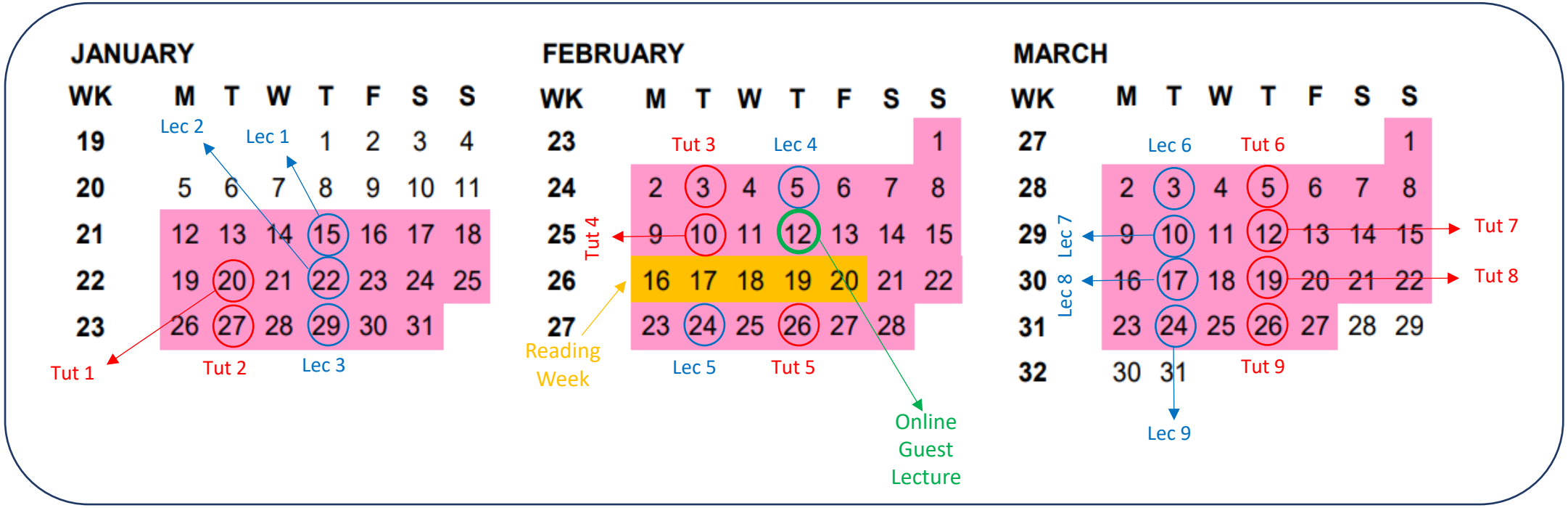
"The Timeline of Everything: From Woolly Mammoths to World Wars", Published by DK, 2018

Introduction – Learning Outcomes

- **Identify** the foundational concepts of artificial intelligence, machine learning, and data-driven modelling and their applications in science and engineering systems.
- **Develop** and refine various supervised and unsupervised algorithms and appreciate their underlying mathematical backgrounds.
- **Recognise** the value of data, **know** how to ask and answer data-driven questions, and **examine** the reliability and robustness of data-driven models.
- **Extract** features and patterns from data and discover new knowledge from it.
- **Identify** the need to use neural networks and deep learning algorithms in some applications, **compare** their efficiency with machine learning algorithms, and **interpret** their predictions

Module Structure/Timetable

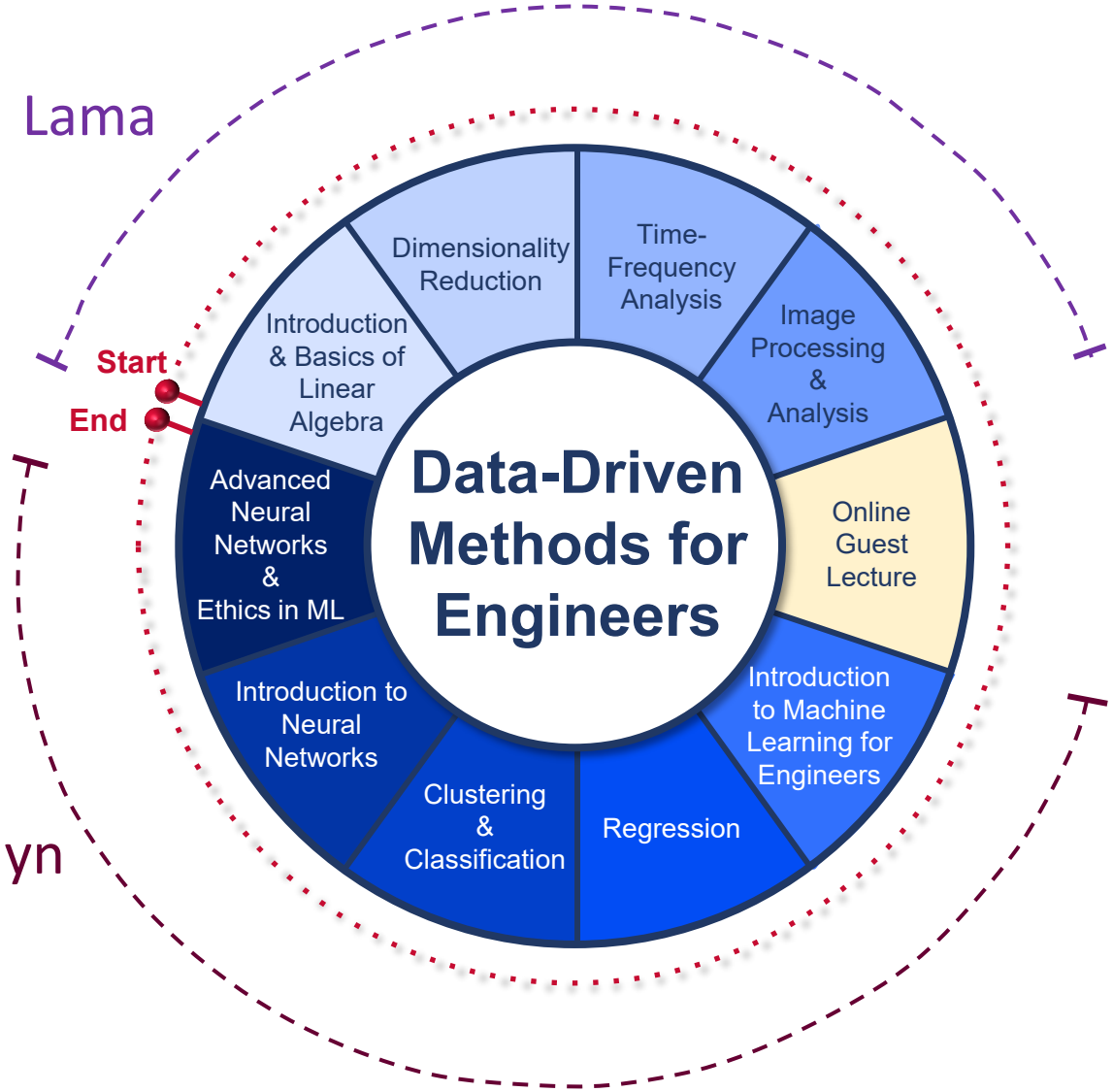
Always check your timetable!



9 Lectures with one online guest lecture **9 Tutorials** **2 Pieces of Coursework**

Module Lectures

Please have MATLAB &/or Jupyter Notebook ready to use on your laptop.



Please bring your laptop with you to the lecture in case you need to code along.

Module Tutorials

Go through the lecture notes before coming to the tutorial so that it will be easier to solve and understand the questions.

Download the tutorial sheet before coming in to the tutorial. You can find it on Moodle.

Please bring your laptop with you to the tutorial as it will be necessary to solve the questions.

There will be 4 PGTAs helping you with the questions.

Please have both MATLAB & Jupyter Notebook ready to use on your laptop.

Module Assessment

JANUARY

WK	M	T	W	T	F	S	S
19				1	2	3	4
20	5	6	7	8	9	10	11
21	12	13	14	15	16	17	18
22	19	20	21	22	23	24	25
23	26	27	28	29	30	31	

FEBRUARY

WK	M	T	W	T	F	S	S
23							1
24	2	3	4	5	6	7	8
25	9	10	11	12	13	14	15
26	16	17	18	19	20	21	22
27	23	24	25	26	27	28	

CW1 Release

MARCH

WK	M	T	W	T	F	S	S
27							1
28	2	3	4	5	6	7	8
29	9	10	11	12	13	14	15
30	16	17	18	19	20	21	22
31	23	24	25	26	27	28	29
32	30	31					

CW1 Deadline

CW2 Release

APRIL

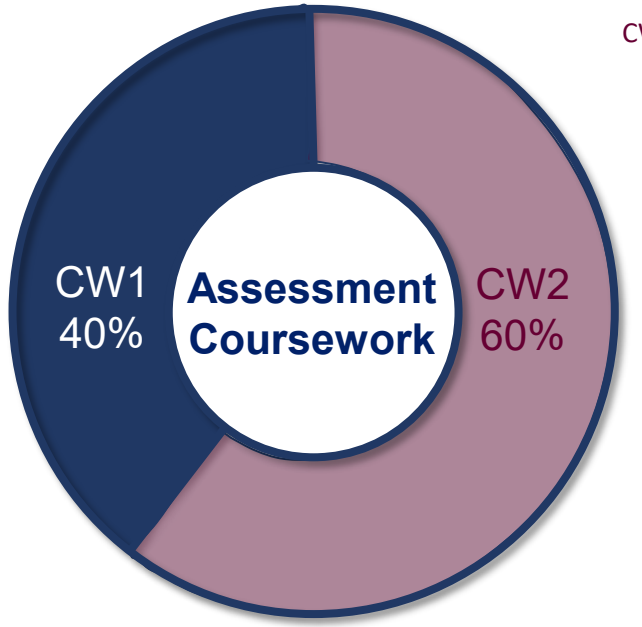
WK	M	T	W	T	F	S	S
32			1	2	3	4	5
33	6	7	8	9	10	11	12
34	13	14	15	16	17	18	19
35	20	21	22	23	24	25	26
36	27	28	29	30			

CW2 Deadline

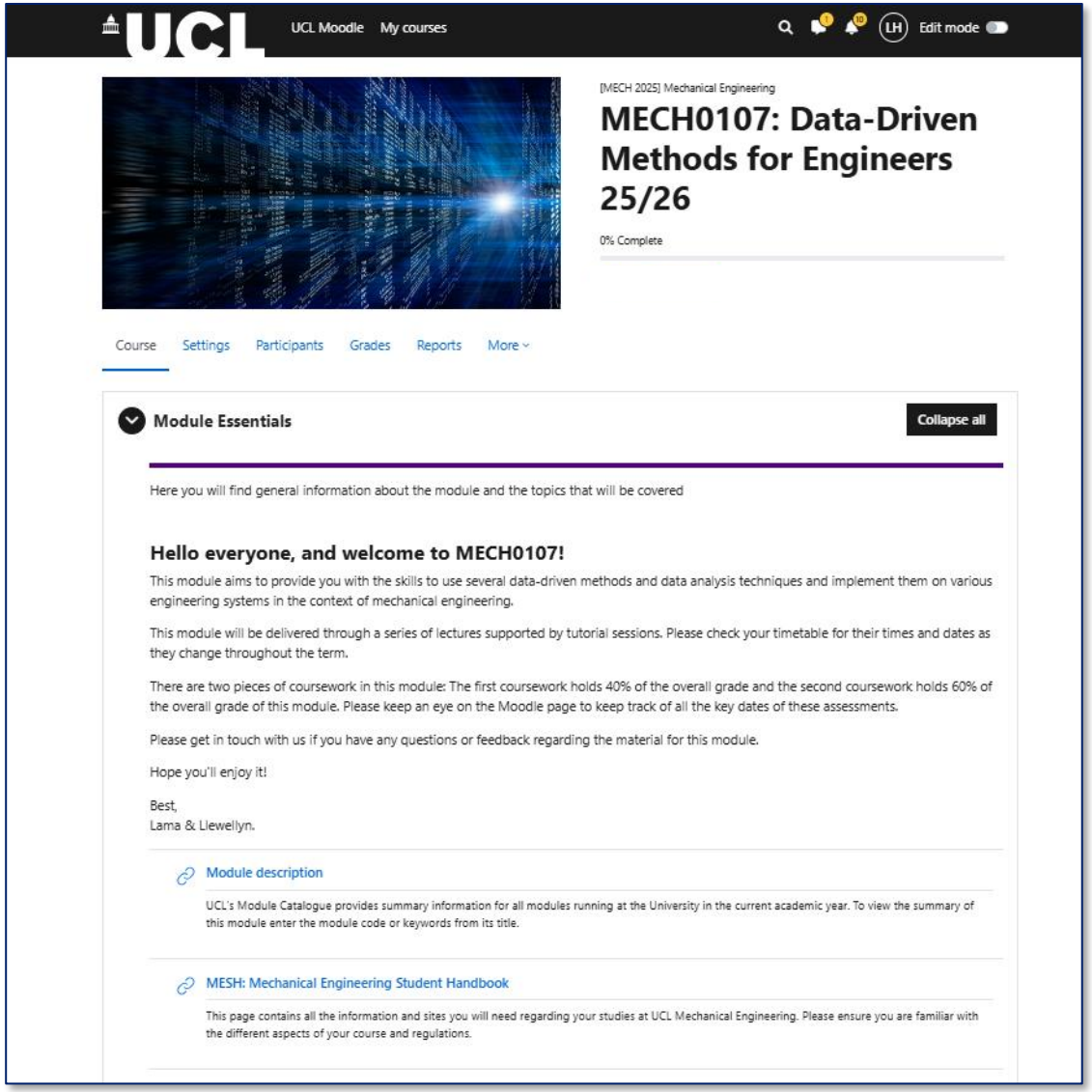
You can use either in your CW

MATLAB

jupyter python™



Module Moodle Page

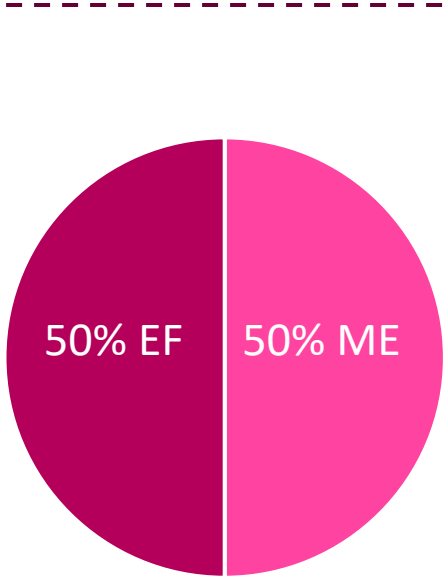


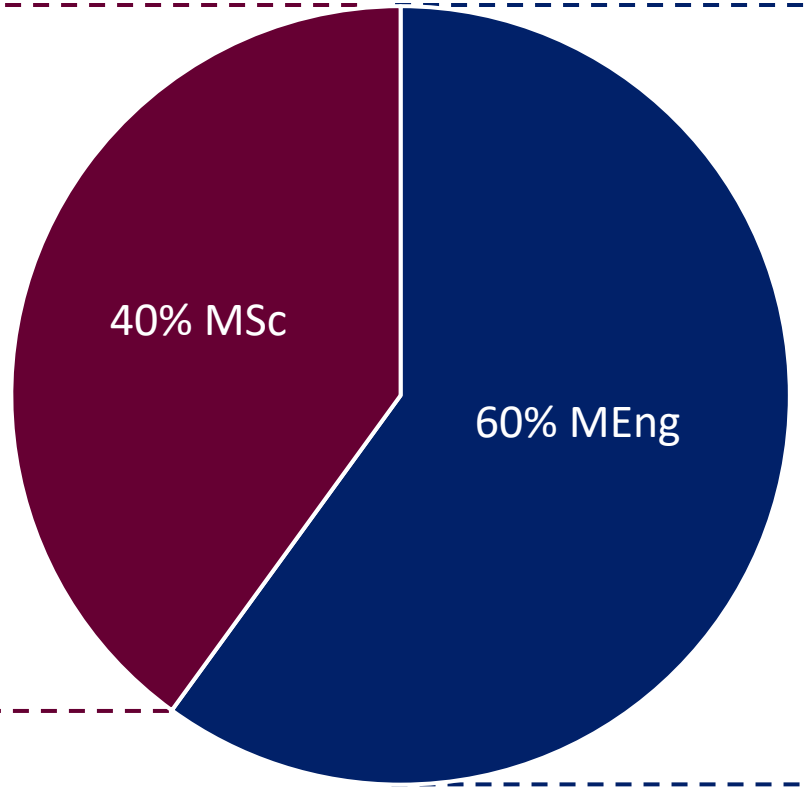
Everything you need about this modules in this page!

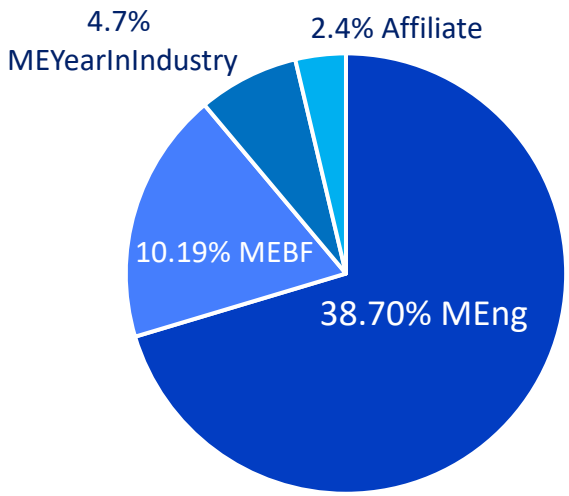
- Lectures
- Tutorials
- Assessment Information
- Assessment Submission Portals
- Q&A Forums
- Further Reading Resources

Module Cohort | 2025/2026

90 Students!



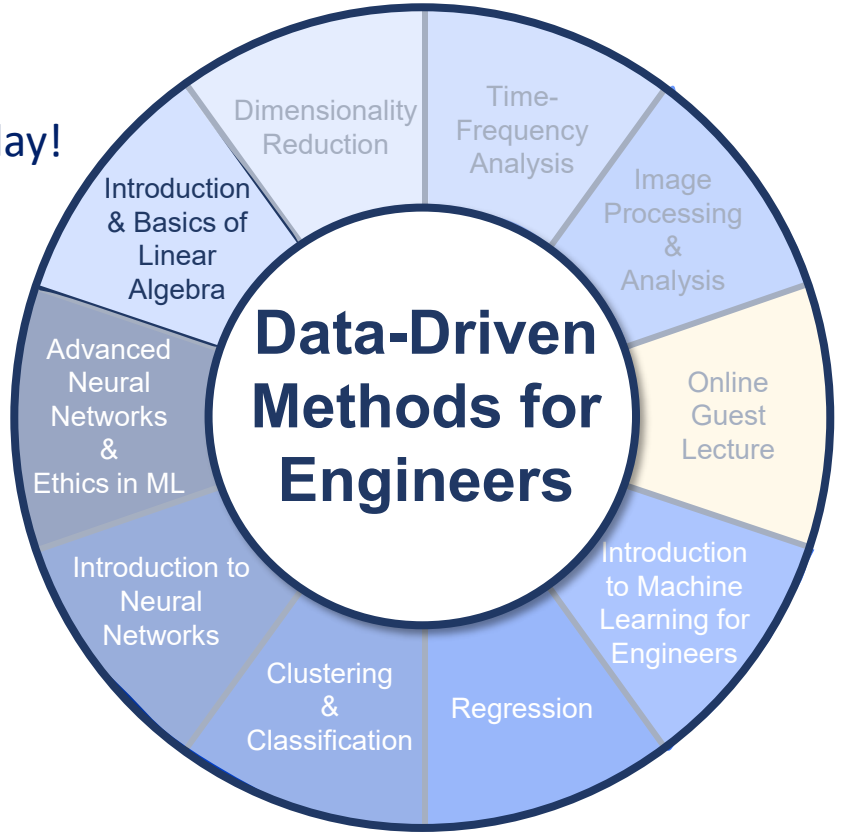




Any Questions before we start?

Module Lectures

Today!



Basics of Linear Algebra

- **Vectors**
  - Magnitude of a Vector
  - Dot Product
  - Cross Product
  - Vector Norms
- **Matrices**
  - Matrices Operations
  - Main Types of Matrices
  - Eigen decomposition
- **Tensors**
- **System of Linear Equations**
  - Direct Solution Methods for $\boldsymbol{Ax = b}$
    - Gaussian Elimination
    - Inverse of a matrix $\boldsymbol{A^{-1}}$
    - LU Decomposition
  - Iterative Solution Methods for $\boldsymbol{Ax = b}$
    - Jacobi's method
    - Gauss-Seidel's method
- **Gradient Descent for $\boldsymbol{Ax = b}$**
- **Numerical Basic Statistics**
  - Mean
  - Median
  - Mode
  - Standard Deviation

Iterative Solution Methods for $Ax = b$

Jacobi's method

- The Jacobi method is one of the simplest iterative method for solving a system of linear equations of the form $\boldsymbol{Ax} = \boldsymbol{b}$. It works by successively approximating the solution vector $\boldsymbol{x}$, updating each variable independently using values from the previous iteration. Jacobi's method is simple to implement and works best when the coefficient matrix $\boldsymbol{A}$ is **diagonally dominant**.

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
\vdots \qquad\quad \vdots \qquad\qquad\quad & \quad \vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

$$
\boldsymbol{A}\ \boldsymbol{x} = \boldsymbol{b}
$$

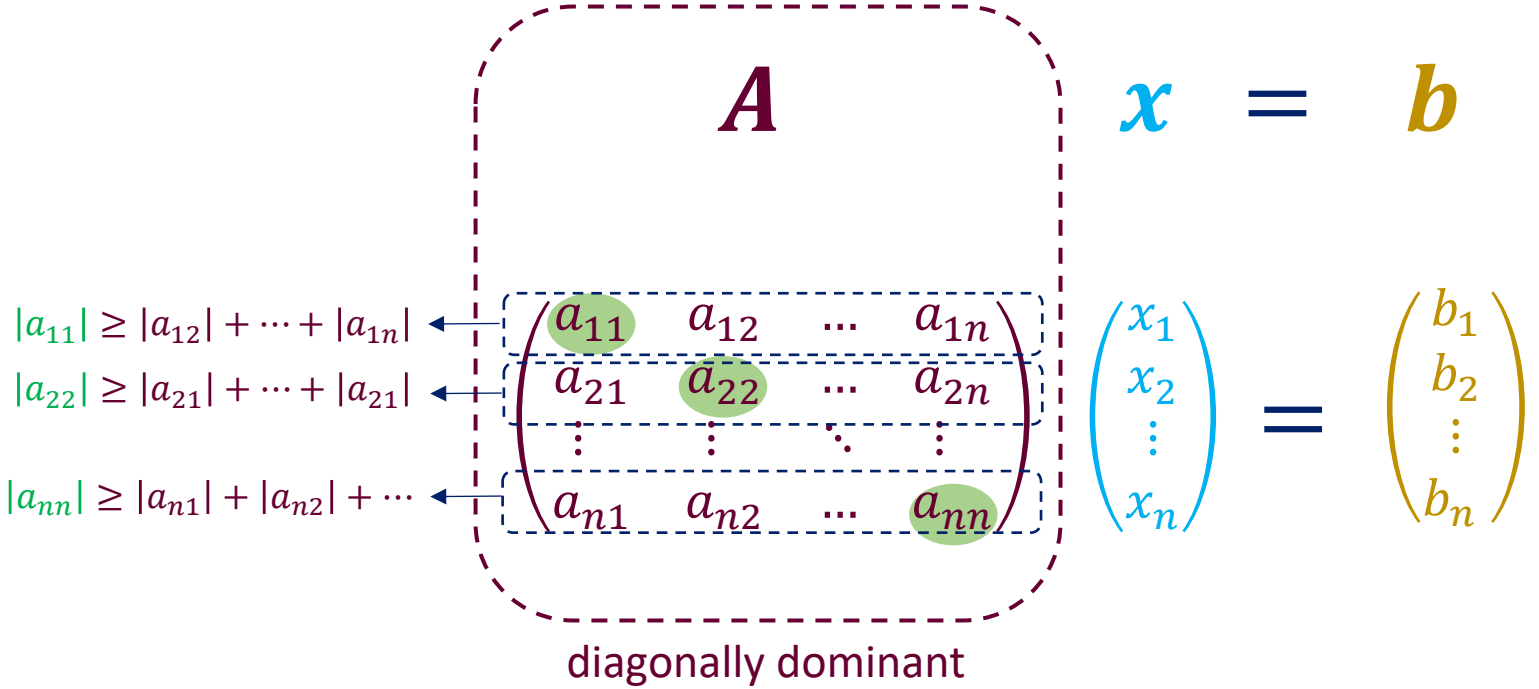
$$
\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}
=
\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}
$$

diagonally dominant

Iterative Solution Methods for $Ax = b$

Jacobi's method

- In mathematics, a square matrix is said to be diagonally dominant if, for every row of the matrix, the magnitude/absolute value of the diagonal entry in a row is larger than or equal to the sum of the magnitudes/absolute values of all the other (non-diagonal) entries in that row.



Iterative Solution Methods for $Ax = b$

Jacobi's method

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

$$A = \begin{pmatrix} 4 & -1 & 1 \\ 4 & -8 & 1 \\ -2 & 1 & 5 \end{pmatrix}$$

$$|4| \geq |-1| + |1|$$

$$|-8| \geq |4| + |1|$$

$$|5| \geq |-2| + |1|$$

diagonally dominant

Solve for x: $x = \frac{7 + y - z}{4}$

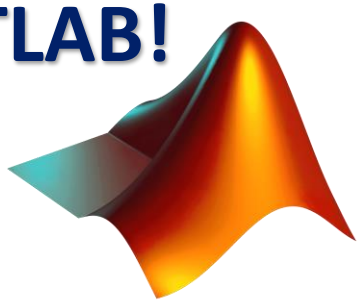
Solve for y: $y = \frac{21 + 4x + z}{8}$

Solve for z: $z = \frac{15 + 2x - y}{5}$

For an iteration process:

$$x_{i+1} = \frac{7 + y_i - z_i}{4}$$
$$y_{i+1} = \frac{21 + 4x_i + z_i}{8}$$
$$z_{i+1} = \frac{15 + 2x_i - y_i}{5}$$

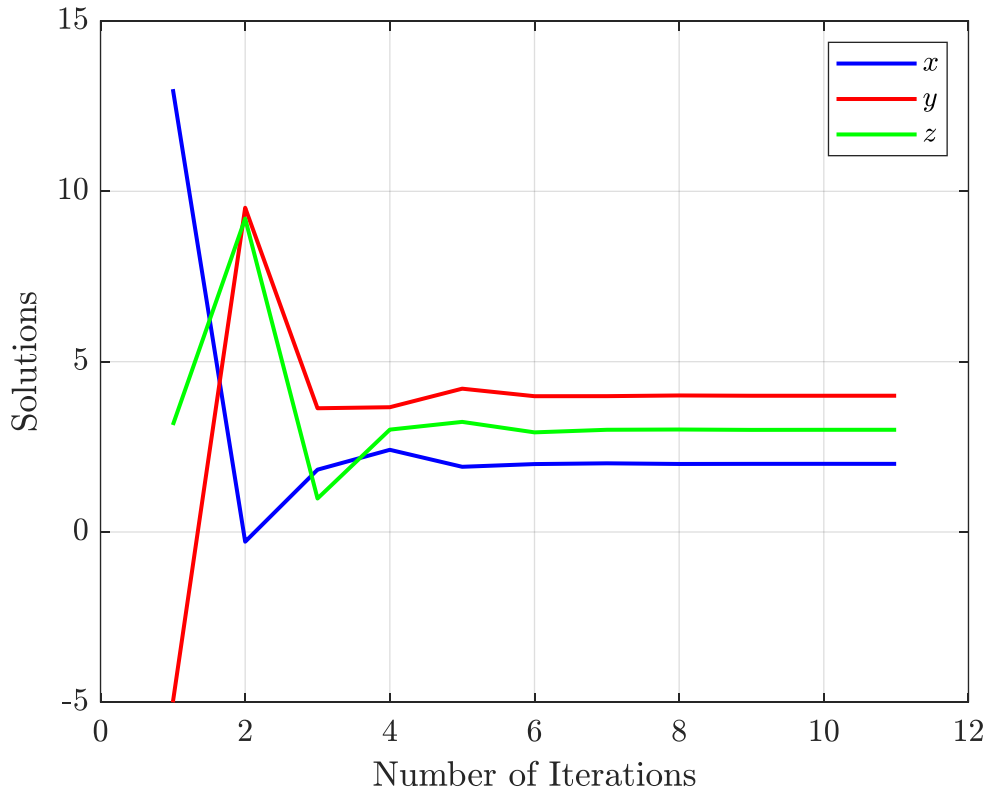
Let's move on to MATLAB!



Iterative Solution Methods for $Ax = b$

Jacobi's method

```
close all;
clear all;
clc

%Equations to solve (Strictly Dominant Diagonal A Matrix (SDD))
% 4x-y+z=7
% 4x-8y+z=-21
% -2x+y+5z=15

%Define initial guess for the three variables
x1(1) = 13;
y1(1) = -5;
z1(1) = pi;

%go through iterations
for j = 1:10
    x1(j+1) = (7+y1(j)-z1(j))/4;
    y1(j+1) = (21+4*x1(j)+z1(j))/8;
    z1(j+1) = (15+2*x1(j)-y1(j))/5;
end

%plot the solution
figure;
subplot(2,1,1)
plot(1:11,x1,'b',1:11,y1,'r',1:11,z1,'g','LineWidth',1.5)
xlabel('Number of Iterations','Interpreter','latex')
ylabel('Solutions','Interpreter','latex')
legend('$x$','$y$','$z$','Interpreter','latex')
% title('Matrix $A$ is a Strictly Dominant Diagonal Matrix (SDD)','Interpreter','latex')
set(gca,'TickLabelInterpreter','latex','FontSize',12)
grid on
hold on
```



Iterative Solution Methods for $Ax = b$

Jacobi's method

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

$$-2x + y + 5z = 15$$
$$4x - 8y + z = -21$$
$$4x - y + z = 7$$

$$\boldsymbol{A} = \begin{pmatrix} -2 & 1 & 5 \\ 4 & -8 & 1 \\ 4 & -1 & 1 \end{pmatrix}$$

No Convergence

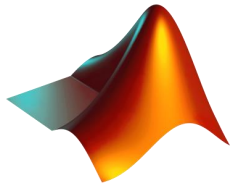
$|-2| \not\geq |1| + |5|$

NOT diagonally dominant

$|-8| \geq |4| + |1|$

$|1| \not\geq |4| + |-1|$

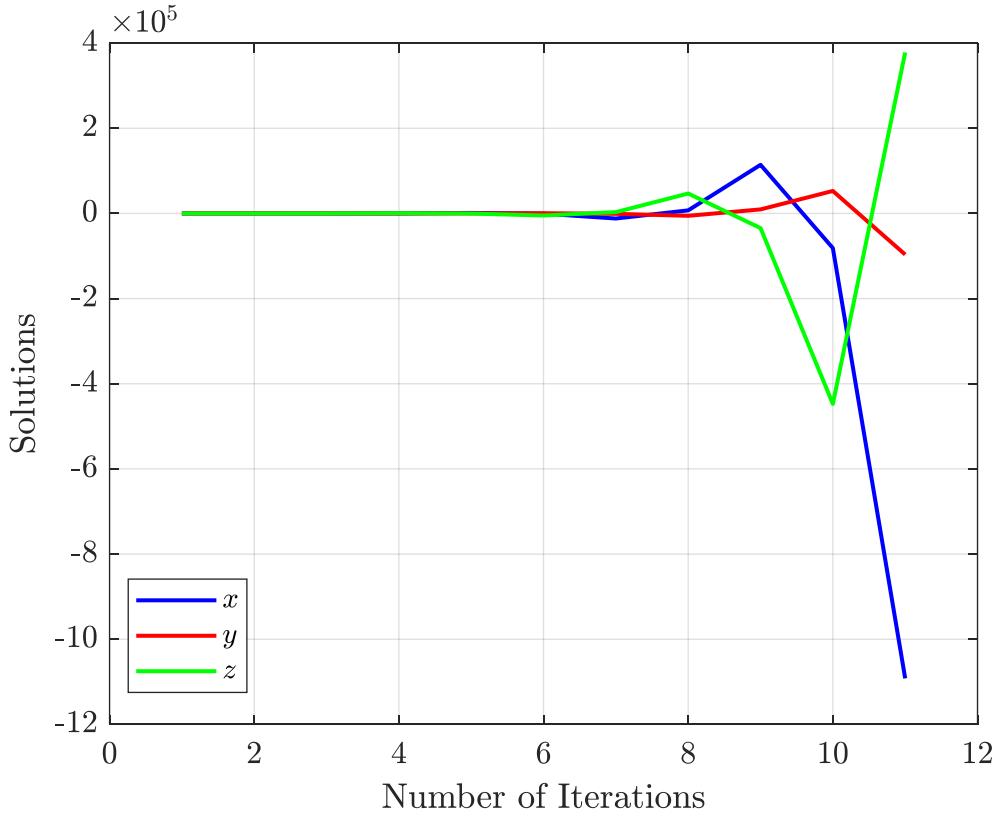
Let's move on to MATLAB!



Iterative Solution Methods for $Ax = b$

Jacobi's method

```
%Equatons to solve (NOT Strictly Dominant Diagonal A Matrix (SDD))
% -2x+y+5z=15
% 4x-8y+z=-21
% 4x-y+z=7

%Define initial guess for the three variables
x2(1) = 13;
y2(1) = -5;
z2(1) = pi;

%go through iterations
for j = 1:10
    x2(j+1) = (-15+y2(j)+5*z2(j))/2;
    y2(j+1) = (21+4*x2(j)+z2(j))/8;
    z2(j+1) = 7-4*x2(j)+y2(j);
end

%plot the solution
subplot(2,1,2)
plot(1:11,x2,'b',1:11,y2,'r',1:11,z2,'g','LineWidth',1.5)
xlabel('Number of Iterations','Interpreter','latex')
ylabel('Solutions','Interpreter','latex')
legend('$x$','$y$','$z$','Interpreter','latex')
% title('Matrix $A$ is not a Strictly Dominant Diagonal Matrix (SDD)','Interpreter','latex')
set(gca,'TickLabelInterpreter','latex','FontSize',12)
grid on
%------------------------
```



Iterative Solution Methods for $Ax = b$

Gauss-Seidel's method

- The Gauss-Seidel method is an improvement over the Jacobi method. With the Jacobi method, **the values obtained in the ith iteration remain unchanged until the entire ith iteration has been calculated**. With the Gauss-Seidel method, **we use the new values as soon as they are known**. This reduces the number of iterations and hence the Gauss-Seidel method converges much faster to the solution.

$$4x - y + z = 7 \xrightarrow{\text{Solve for } x} x = \frac{7 + y - z}{4}$$

$$4x - 8y + z = -21 \xrightarrow{\text{Solve for } y} y = \frac{21 + 4x + z}{8}$$

$$-2x + y + 5z = 15 \xrightarrow{\text{Solve for } z} z = \frac{15 + 2x - y}{5}$$

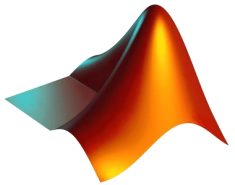
For an iteration process

$$x_{i+1} = \frac{7 + y_i - z_i}{4}$$

$$y_{i+1} = \frac{21 + 4\cancel{x_i}^{\,x_{i+1}} + z_i}{8}$$

$$z_{i+1} = \frac{15 + 2\cancel{x_i}^{\,x_{i+1}} - \cancel{y_i}^{\,y_{i+1}}}{5}$$

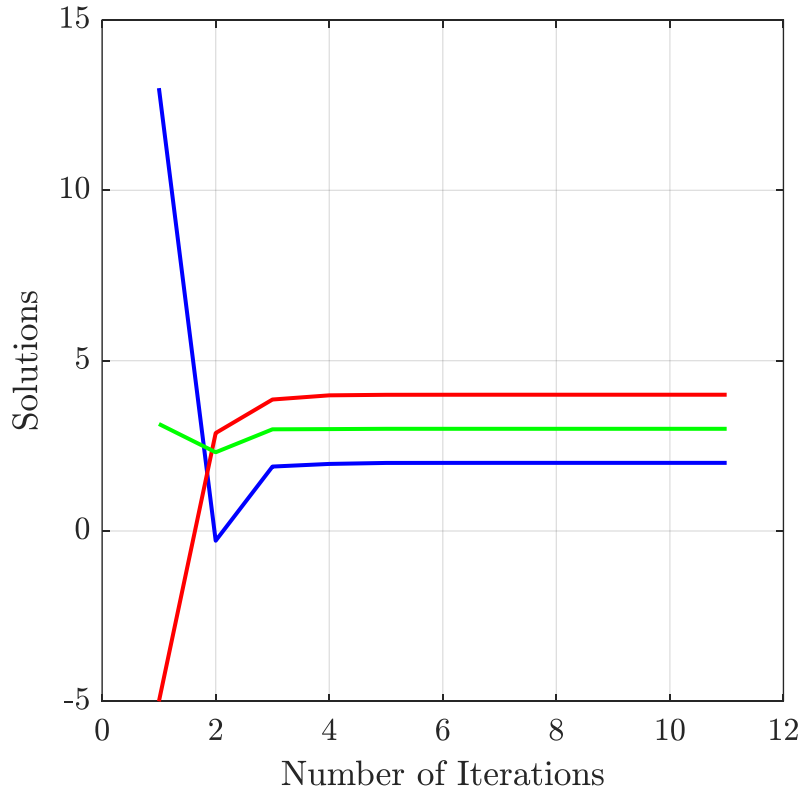
Let's move on to MATLAB!



Iterative Solution Methods for $Ax = b$

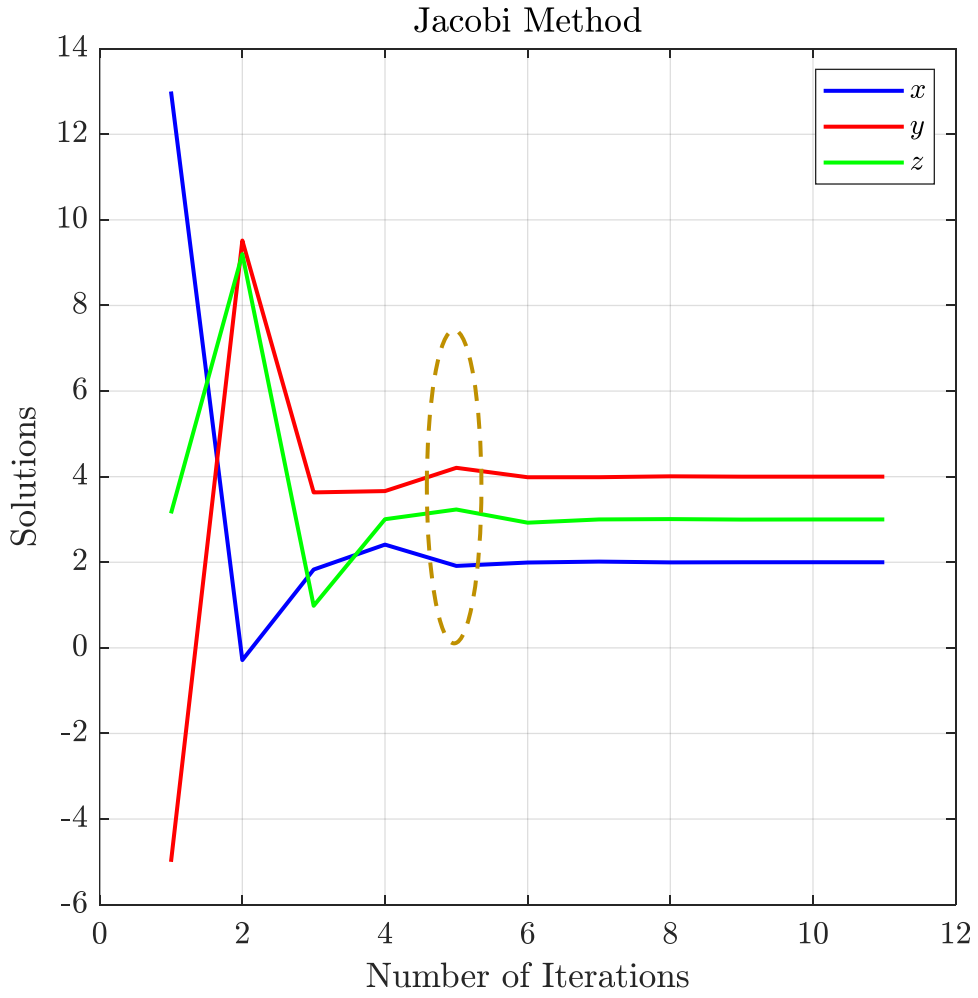
Gauss-Seidel's method

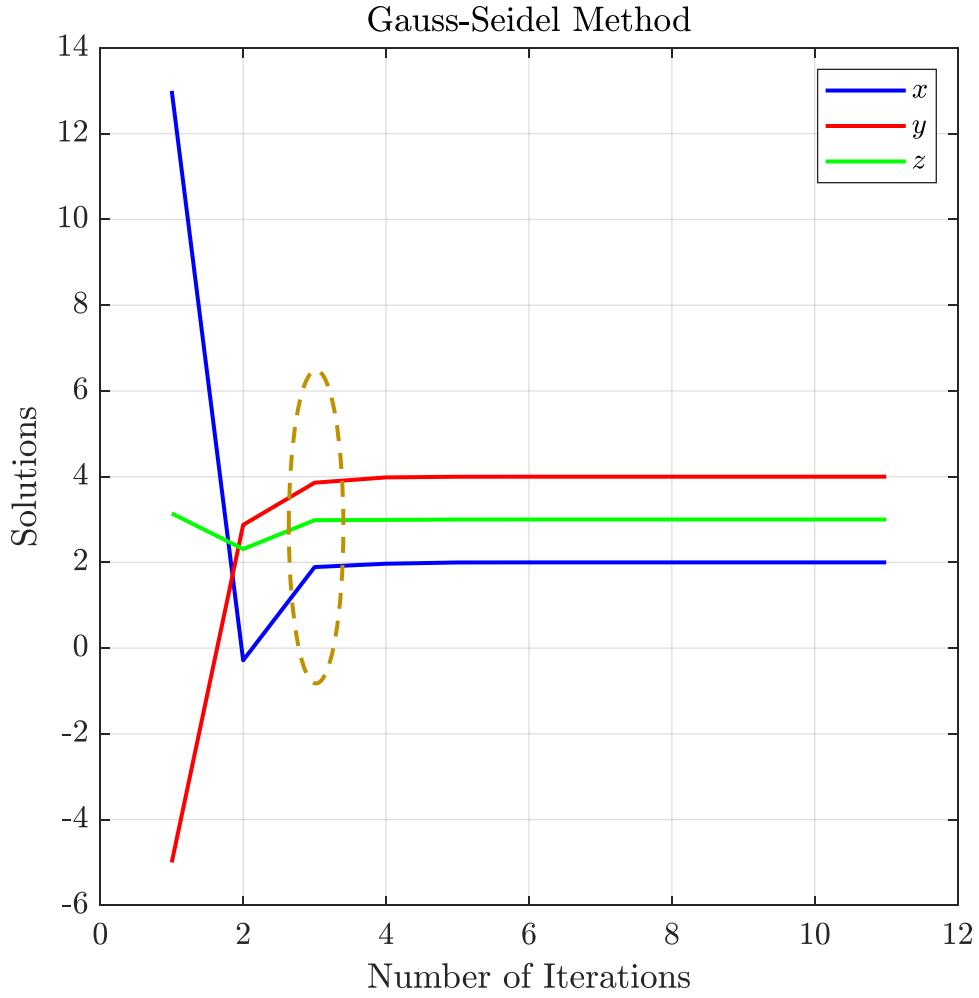
```
%Equations to solve (Strictly Dominant Diagonal A Matrix (SDD))
% 4x-y+z=7
% 4x-8y+z=-21
% -2x+y+5z=15

%Define initial guess for the three variables
x3(1) = 13;
y3(1) = -5;
z3(1) = pi;

%go through iterations
for j = 1:10
    x3(j+1) = (7+y3(j)-z3(j))/4;
    y3(j+1) = (21+4*x3(j+1)+z3(j))/8;
    z3(j+1) = (15+2*x3(j+1)-y3(j+1))/5;
end

%plot the solution
figure;
% subplot(1,2,1)
plot(1:11,x3,'b',1:11,y3,'r',1:11,z3,'g','LineWidth',1.5)
xlabel('Number of Iterations','Interpreter','latex')
ylabel('Solutions','Interpreter','latex')
% title('Matrix $A$ is SDD','Interpreter','latex')
set(gca,'TickLabelInterpreter','latex','FontSize',12)
axis square
grid on
hold on
%-----------------------
```



Iterative Solution Methods for $Ax = b$

Jacobi's method vs Gauss-Seidel's method





Gradient Descent

Gradient Descent (in linear systems of equations)

- Suppose you want to solve a system like

$$\boldsymbol{Ax} = \boldsymbol{b}$$

One way is to define a function that measures how far a guess $\boldsymbol{x}$ is from being a true solution, for example:

$$f(\boldsymbol{x}) = \|\boldsymbol{Ax} - \boldsymbol{b}\|^2$$

If $\boldsymbol{Ax} = \boldsymbol{b}$, then $f(\boldsymbol{x}) = 0$

- Gradient descent is a method where you start with a guess for $\boldsymbol{x}$, then repeatedly adjust it a little in the opposite direction of the gradient (the slope) of $f(\boldsymbol{x})$. Step by step, this moves your guess closer to the true solution.

Gradient Descent (in machine learning)

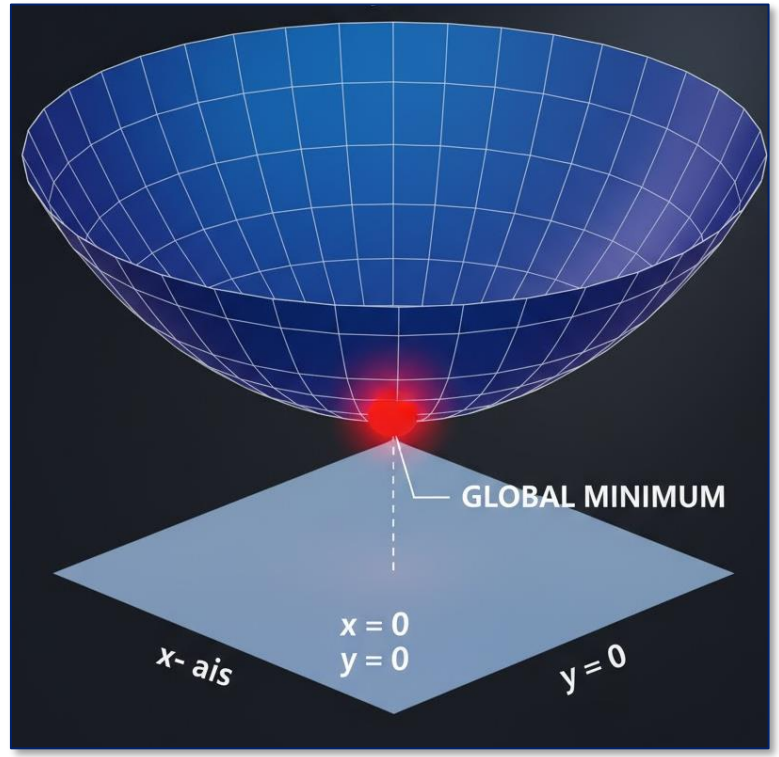
- In machine learning, we often want to train a model (e.g. a neural network) by adjusting parameters (weights).
- We measure the model's error using a **loss function** (how badly predictions differ from actual values).
- Gradient descent is used to reduce this error by computing the gradient of the loss function with respect to the parameters, and moving the parameters slightly in the opposite direction of the gradient.
- Repeating this makes the model's predictions better.

Gradient Descent

What does it mean to “find a minimum”?

- A function can be thought of as a landscape with hills and valleys.
- Finding a minimum means finding the lowest point (valley) of that landscape.
- In **linear systems**, the “minimum” corresponds to the point where equations are satisfied.

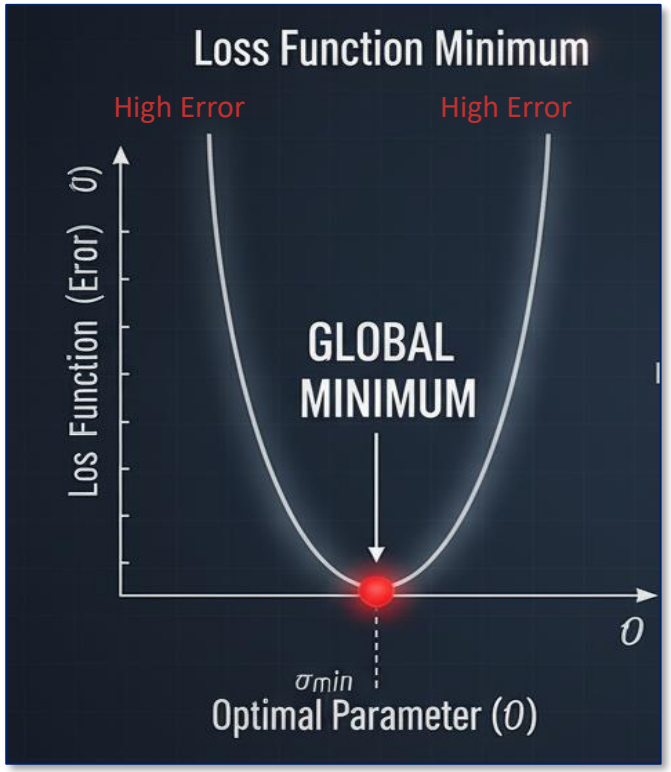
$f(x, y) = x^2 + y^2$



Gradient Descent

What does it mean to "find a minimum"?

- A function can be thought of as a landscape with hills and valleys.
- Finding a minimum means finding the lowest point (valley) of that landscape.
- In **machine learning**, a 'minimum' refers to the set of model parameters that minimise a chosen loss (or cost) function, and therefore provide the best fit to the data under that modelling assumption. In practice, this means finding parameter values (such as weights) that make the prediction error as small as possible.



Gradient Descent for Solving $Ax = b$

let's consider the simple 2×2 system of linear equations:

$$2x + y = 0$$

$$-x + 6y = 0$$

This system can be written in the matrix format as follows:

$$\boldsymbol{A}\,\mathbf{x} = \boldsymbol{b} \quad \Rightarrow \quad \begin{bmatrix} 2 & 1 \\ -1 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Gradient Descent for Solving $Ax = b$

Step 1: Use of quadratic form (Find what we want to minimize)

To begin with the gradient decent algorithm, this system should be converted into its quadratic form:

$$
\begin{aligned}
f(\mathbf{x}) &= \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x} \\
&= \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 2 & 1 \\ -1 & 6 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} 0 & 0 \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} 2x + y \\ -x + 6y \end{bmatrix} - 0 \\
&= x^2 + 3y^2
\end{aligned}
$$

Step 2: Compute the gradient of $f(x, y)$

Let's now compute the gradient of this scalar function:

$$
\begin{aligned}
\nabla f(x, y) &= \frac{\partial f}{\partial x}\,\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\,\hat{\mathbf{y}} \\
&= 2x\,\hat{\mathbf{x}} + 6y\,\hat{\mathbf{y}} \\
&= \begin{bmatrix} 2x \\ 6y \end{bmatrix}
\end{aligned}
$$

Step 3: Build an iteration Process

The next step now is to build the iteration process (or the update rule).

Future vector; Opposite to Gradient direction; Step size (or learning rate)

$$
\begin{aligned}
\mathbf{x}_{k+1}(\tau) &= \mathbf{x}_k - \tau\nabla f(\mathbf{x}) \\
&= \begin{bmatrix} x \\ y \end{bmatrix} - \tau\begin{bmatrix} 2x \\ 6y \end{bmatrix} \\
&= (1 - 2\tau)x\,\hat{\mathbf{x}} + (1 - 6\tau)y\,\hat{\mathbf{y}}
\end{aligned}
$$

τ determines convergence speed and stability.

- If τ is **too large**, the algorithm will **diverge** (overshoot the minimum).
- If τ is **too small**, convergence will be **very slow**.

Our job now, is to get an expression for τ…

Gradient Descent for Solving $Ax = b$

Step 4: Construct $F(\tau)$: study how the loss changes with step size

This expression is used to compute:

$$f(\mathbf{x}) = x^2 + 3y^2$$

$$\Rightarrow$$

The function of the future vector of values

$$\begin{aligned} F(\tau) &= f(\mathbf{x}_{k+1}(\tau)) \\ &= [(1-2\tau)x]^2 + 3[(1-6\tau)y]^2 \\ &= (1-2\tau)^2x^2 + 3(1-6\tau)^2y^2 \end{aligned}$$

Step 5: Set $\dot{F}(\tau) = 0$. This ensures we take the best possible step at each iteration

Setting $\dot{F}(\tau) = 0$ (finding the minima!)

$$\begin{aligned} \frac{\partial F}{\partial \tau} &= \frac{\partial}{\partial \tau}\left[(1-2\tau)^2x^2 + 3(1-6\tau)^2y^2\right] \\ &= 8x^2\tau - 4x^2 + 216y^2\tau - 36y^2 \\ &= 0 \end{aligned}$$

Step 6: Find the step size (τ) expression

Solving this equation for τ, gives:

$$\tau = \frac{x^2 + 9y^2}{2x^2 + 54y^2}$$

controls how far you move in the direction *opposite* to the gradient at each iteration.

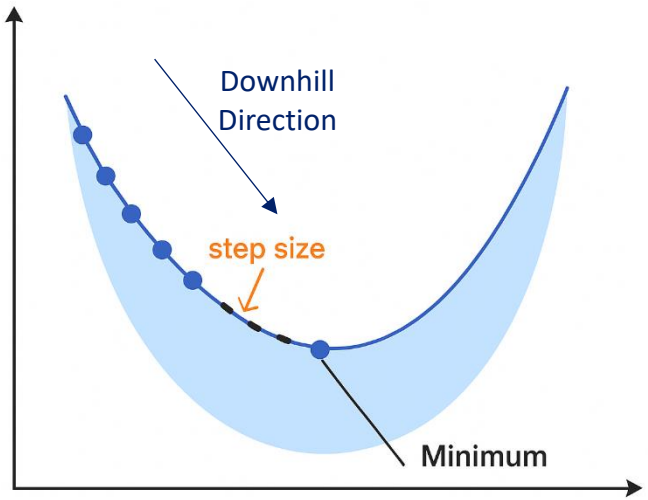
Line Search Approach

Gradient Descent for Solving $Ax = b$

why the step size τ moves in the opposite direction to the gradient at each iteration?

$$\tau = \frac{x^2 + 9y^2}{2x^2 + 54y^2}$$

- The **gradient** of a function points in the direction of the **steepest increase** — that is, the direction in which the function's value grows most rapidly.
- Since in optimization (and in solving systems or training models) we usually want to **minimize** a function — such as a cost or loss function — we must move in the **opposite direction** of the gradient to reduce its value.



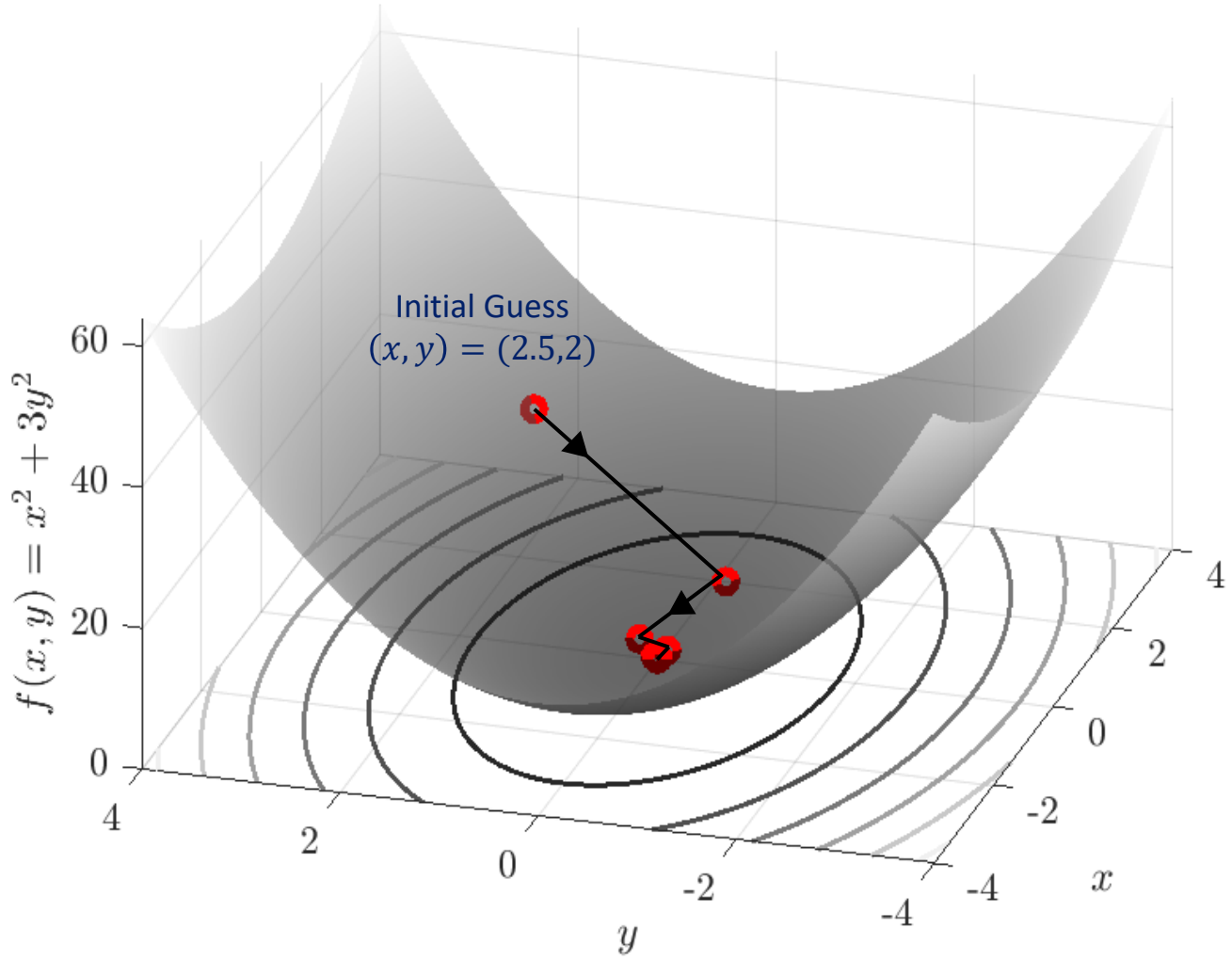
The **negative sign** ensures we move **downhill**, toward smaller values of $f(\mathbf{x})$

Mathematically: $\mathbf{x}_{k+1}(\tau) = \mathbf{x}_k - \tau \nabla f(\mathbf{x})$

step size or learning rate

gradient at the current point (direction of steepest ascent)

Gradient Descent for Solving $Ax = b$




```
>> GradientDescent

ans =

   1.5415e-04  = x


ans =

   1.2332e-04  = y
```

To wrap up with a few reminders…

- Go through the first chapter to refresh your Basics in Linear Algebra. You'll need it!
- **Your first tutorial will be on Tuesday 20th** . Four PGTAs will be with you during the session. Don't spare any question! Ask them and they will be happy to help.
- All Lecture Material will be uploaded to Moodle later this day, along with the questions of the first tutorial .
- See you next Thursday 22nd !