

Descartes, Cartesian Coordinates and Linear Algebra

Gavin Conran

I think it is worth while pointing out the interweaving thread found between mathematics and philosophy, and so I want to high-light the contribution ***Rene Descartes*** made, not only to mathematics, with his Cartesian Coordinates, but also to philosophy with the observation ‘Cogito Ergo Sum’. Probably the main reason to include Descartes is that Geometry and, and many would argue, Linear Algebra, would not have been possible without the invention of Cartesian Coordinates. There is, of course, the fact that he is viewed, together with his rationalist peers, Spinoza and Leibniz, as one of the greatest mathematicians and philosophers of all time.

Through out the Fourier, Gauss, Newton and Euler papers linear algebra, in different guises, makes an appearance. This is because Linear Algebra is probably the most popular tool used in the many strains of computational mathematics discussed to date. This paper will look under the bonnet, with code examples, of the different linear algebra topics introduced elsewhere in my mathematical adventures in Ma Ma Land:

Fourier paper:

- ***Electronics/Electrical Circuits:***
 - Kirchoff & Ohm’s Laws
- ***Signal Processing:***
 - Fourier Series
 - Fast Fourier Transform (FFT)
 - Complex Matrices:
 - Hermitian (symmetric)
 - Unitary (orthogonal)
- ***Differential Equations:***
 - Ordinary Differential Equations
 - Eigenvalues / Eigenvectors
 - Partial Differential Equations
 - FDM Numerical Methods
 - Heat Equation
 - Iterative solvers
 - Jacobi, Gauss-Seidel, SOR & Conjugate Gradient

Gauss paper:

- ***Data Science / Machine Learning:***
 - Principle Component Analysis (PCA) computed from:
 - Spectral Theorem, Diagonalisation & Eigenvalue / Eigenvector
 - Diagonalisation & Singular Value Decomposition (SVD)
 - (Weighted) Least Squares / Projections / Error
 - Linear Algebra
 - Positive Definite Matrices
 - Probability
 - Maximum Likelihood (ML) / Maximum A Posteriori (MAP)
 - Dimension reduction
 - Singular Value Decomposition (SVD)

Euler paper:

- ***Optimisation:***
 - Relationship between matrix Elimination and Graphs
 - Reducing Graph problems to ***Linear Programming***
 - Simplex algorithm

Additional Linear Algebra Applications:

- ***Image Processing***
 - Filtering
 - Compression
- ***Neural Networks***
 - Tensorflow
 - Caffe
 - ArtNet

There are many more areas in which linear algebra plays a key role, two of which being robotics and cryptography. Although I have studied both topics, I have not included them in this discussion. At a later date, I may update the document to include one or both of these subjects.

Networks / Graphs and Symmetric Matrices

There a direct relationship between every Matrix Elimination step and a Graph:

- **Rows** are **Dependent** if the corresponding **Edges** contain a **Loop**
- At the **end of Elimination** we have a **full set of r Independent Rows**
 - These **r edges form a tree** – a Graph with no loops, i.e. acyclic
 - called a **Spanning Tree**, a tree that ‘spans’ all the nodes of the Graph
 - A Spanning Tree has **$n-1$ edges** if the Graph is **connected**
 - **$n-1$** is the **Rank, r** , of the **Incidence matrix**
 - Row Space $C(A^T)$ and Column Space $C(A)$ have dimensions $n-1$
 - A Spanning Tree from Elimination gives a **Basis for the Row Space**
 - **Each Edge in the Tree corresponds to a Row in the Basis**

Fundamental Equations of Equilibrium

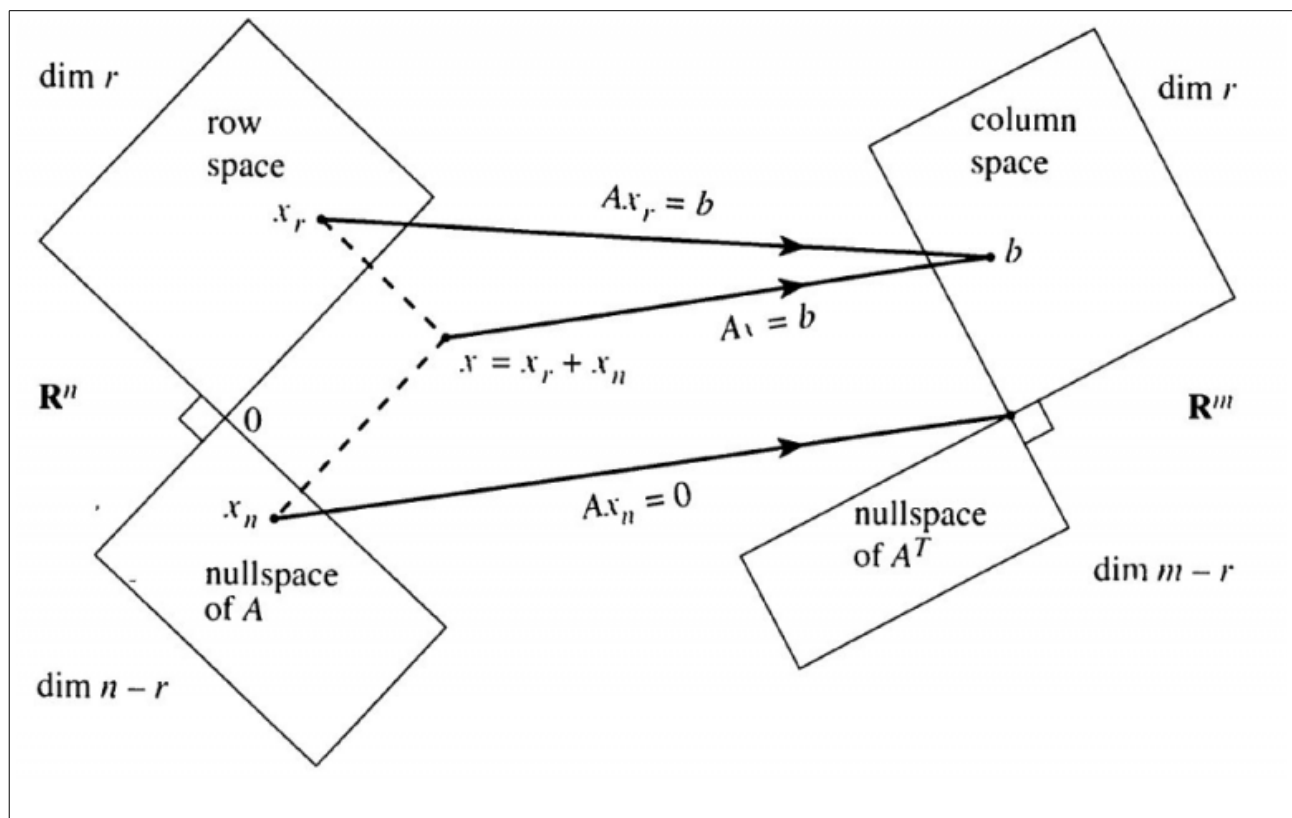


Illustration 1: Big Picture of the Fundamental Theorem of Linear Algebra

The Fundamental Theorem of Linear Algebra connects the Dimensions of Subspaces:

- NullSpace $N(A)$
- Column Space $C(A)$
- Row Space $R(A)$ or $C(A^T)$

- Left NullSpace $N(A^T)$

Part 1:

- The Column Space $C(A)$ and Row Space $C(A^T)$ both have dimensions r , the Rank
- The Null Spaces have dimensions $(n-r)$ for $N(A)$ and $(m-r)$ for $N(A^T)$

Part 2:

- Vectors in $C(A)$ and $N(A^T)$ are perpendicular
- Vectors in $C(A^T)$ and $N(A)$ are perpendicular

which leads to ***Euler's Formula***:

$$(\# \text{ of nodes}) - (\# \text{ of edges}) + (\# \text{ of loops}) = (n) - (m) + (m-n+1) = 1$$

Where Nodes are in 0 dimensions

Edges are in 1 dimensions

Loops are in 2 dimensions

An example of a Network → Electronics

- A Graph becomes a Network when a Diagonal Matrix C is assigned to Edges.
- C reflects 'material properties' in contrast to the Incidence Matrix, A , which gives info about the connections.
- On Edge i , the conductance is C_i and the resistance is $\frac{1}{C_i}$

We need three laws to complete our Framework:

1. **Ohm's Law:** current y_i through the resistor is proportional to the Voltage drop e_i
 - Ohm's law: $y_i = C_i e_i$
 - Alternative Form: $E = IR$
 - Vector Form: $y = Ce$
 - With an external voltage source, e.g. a battery, Ohm's law is given by $e = b - Ax$ across a Resistor
 - The drop in Potential = (battery in Edge i of strength b_i) - Ax
 - where x_i represents the potentials at the Nodes
$$y = C(b - Ax) \quad \text{or} \quad C^{-1}y + Ax = b$$

2. **Kirchoff Voltage Law (KVL):** The Voltage drop around each loop adds to Zero
 - Allows us to assign Potentials x_1, \dots, x_n to the Nodes
 - Differences around a loop give a sum in which everything cancels

$$(x_2 - x_1) + (x_3 - x_2) + (x_1 - x_3) = 0$$

3. **Kirchoff Current Law (KCL):** The Currents y_i (and f_i) into each node equals Zero
 - Asks us to add the currents into each Node by the multiplication $A^T y$
 - If there are no external sources of currents, KCL is

$$A^T y = 0$$

The Fundamental Equations of equilibrium combine Ohm and Kirchoff into a central problem of Applied Mathematics. These equations appear everywhere:

**Equilibrium
Equations**

$$C^{-1}y + Ax = b$$

$$A^T y = 0$$

Ohm's law
KCL

This is a Linear Symmetric System, from which e has disappeared. The columns are the currents, y , and the potentials, x .

**Block
Form**

$$\begin{bmatrix} C^{-1} & A \\ A^T & 1 \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix}$$

For Block Elimination:

- the pivot is C^{-1}

- the multiplier is $A^T C$
- subtraction knocks out A^T below the pivot:

$$\text{After Block Elimination} \quad \begin{bmatrix} C^{-1} & A \\ 0 & -A^T C A \end{bmatrix} \begin{bmatrix} y \\ x \end{bmatrix} = \begin{bmatrix} b \\ f - A^T C b \end{bmatrix}$$

The equation for x alone is in the bottom row.

$$\text{Fundamental Equation} \quad A^T C A x = A^T C b - f$$

Back substitution in the first equation produces y, i.e. substitute $y = C(b - Ax)$ into $A^T y = f$ to give **The Fundamental Equation**.

Notes

- One potential must be fixed in advance, $x_n = 0$
- The n^{th} node is Grounded, & the n^{th} column of the original Incidence matrix is removed
- The resulting Matrix is what we now mean **A**:
 - with n-1 independent columns
- The square matrix $A^T C A$, which is key to solving the Fundamental Equation for x, is an invertible matrix of order n-1:

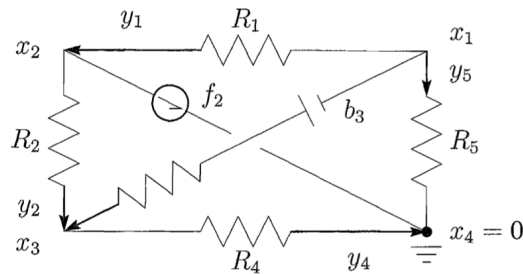
$$\begin{array}{cccc} (n-1) \text{ by } m & m \text{ by } m & m \text{ by } (n-1) & (n-1) \text{ by } (n-1) \\ \left[\begin{array}{c} A^T \end{array} \right] & \left[\begin{array}{c} C \end{array} \right] & \left[\begin{array}{c} A \end{array} \right] & \left[A^T C A \right] \end{array}$$

- $A^T C A$ is symmetric
- It has +ive pivots and comes from the Basic Framework of Applied Maths
- In mechanics, x and y become displacements and stresses
- In fluids, the unknowns are pressure and flow rate
- In statistics, e is the error and x is the best least-squares fit to the data → Least Squares

Above describes an example of when $Ax = b$ can be solved, i.e. there are the same number of equations as unknowns and b can be found in the column space. This is not always the case, e.g. when there are many more equations than unknowns and b is not in the column space. This will be the topic of the next section on Projection and Least Squares.

Example: Kirchoff and Ohm's Laws

Example 1 Suppose a battery b_3 and a current source f_2 (and five resistors) connect four nodes. Node 4 is grounded and the potential $x_4 = 0$ is fixed.



The first thing is the current law $A^T y = f$ at nodes 1, 2, 3:

$$\begin{aligned} -y_1 - y_3 - y_5 &= 0 \\ y_1 - y_2 &= f_2 \\ y_2 + y_3 - y_4 &= 0 \end{aligned} \quad \text{has} \quad A^T = \begin{bmatrix} -1 & 0 & -1 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 \end{bmatrix}.$$

No equation is written for node 4, where the current law is $y_4 + y_5 + f_2 = 0$. This follows from adding the other three equations.

The other equation is $C^{-1}y + Ax = b$. The potentials x are connected to the currents

Illustration 2: Electrical Circuit

Given the circuit in Illustration 2, with values for the battery, b_3 , the input current f_2 , and the resistances (or C), R_1, R_2, R_3, R_4, R_5 . Once the potentials are solved by using the Fundamental Equation, $A^T C A x = A^T C b - f$, the currents can be computed by using Ohm's law, $A^T C A x = A^T C b - f$. The computed potentials and currents can be found in Table 1.

| Potentials (by solving $A^T C A x = A^T C b - f$) | Currents (by solving $y = C(b - Ax)$) |
|--|--|
| $x_1 = -0.68$ volts | $y_1 = 3.98$ amps |
| $x_2 = -4.66$ volts | $y_2 = -11.02$ amps |
| $x_3 = 0.85$ volts | $y_3 = 4.42$ amps |
| | $y_4 = 3.40$ amps |
| | $y_5 = -3.40$ amps |
| with $b_3 = 3V$ $f_2 = 5A$ $C = [1, 2, 3, 4, 5] \Omega^{-1}$ | |

Table 1: Results from above Circuit using $A^T C A x = A^T C b - f$ and $y = C(b - Ax)$

Positive Definite Matrices

The problem solved by positive definite matrices is to recognise a minimum point, a major problem found in optimisation. The mathematical problem is to move the second derivative test $F'' > 0$ into n dimensions, usually with the introduction of a Hessian matrix, see page 40.

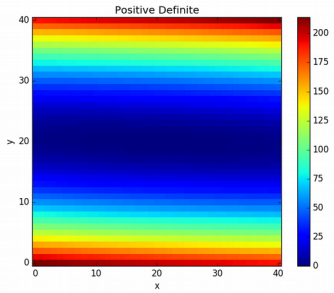
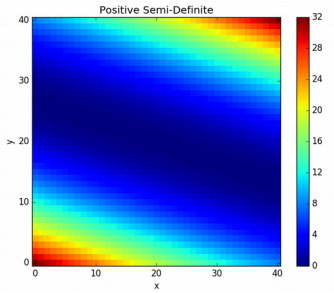
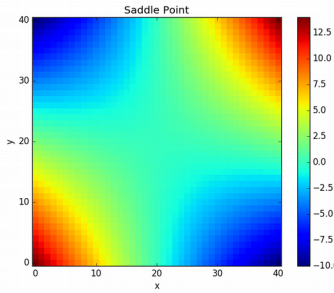
| <i>Positive Definite</i> | <i>Positive Semi-Definite</i> | <i>Saddle Point</i> |
|--|---|---|
|  |  |  |
| $A = \begin{bmatrix} 2 & 6 \\ 6 & 200 \end{bmatrix}$ | $A = \begin{bmatrix} 2 & 6 \\ 6 & 18 \end{bmatrix}$ | $A = \begin{bmatrix} 2 & 6 \\ 6 & 0 \end{bmatrix}$ |
| $\forall \lambda = 0 \quad \& \quad x^T A x > 0$ | $\exists \lambda = 0 \quad \& \quad x^T A x = 0$ | $x^T A x < 0$ |
| <p>If $A = \begin{bmatrix} a & b \\ b & c \end{bmatrix}$</p> <p>Then $x^T A x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a & b \\ b & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = a x_1^2 + 2 b x_1 x_2 + c x_2^2$</p> | | |

Illustration 3: Left: Positive Definite; Middle: Positive Semi-Definite; Right: Saddle Point

Test for a minimum:

- The conditions $a > 0$ and $ac > b^2$ are just right. They guarantee $c > 0$. $a x_1^2 + 2 b x_1 x_2 + c x_2^2$ is positive, and we have found a minimum.

Test for a maximum:

- Since $a x_1^2 + 2 b x_1 x_2 + c x_2^2$ has a maximum whenever $-a x_1^2 + 2 b x_1 x_2 + c x_2^2$ has a minimum, we just reverse the signs of a , b , and c . This actually leaves $ac > b^2$ unchanged. The quadratic form is negative definite iff $a < 0$ and $ac > b^2$.

Singular case $ac = b^2$:

- The second term in $a x_1^2 + 2 b x_1 x_2 + c x_2^2$ disappears to leave only the first square – which is either **positive semidefinite**, when $a > 0$, or **negative semidefinite**, when $a < 0$. The prefix *semi* allows the possibility that $a x_1^2 + 2 b x_1 x_2 + c x_2^2$ can equal zero, as it will at the point $x = b, y = -a$.

Saddle point $ac < b^2$:

- A stationary point that is neither a maximum or a minimum.

Orthogonality, Projection & Least Squares

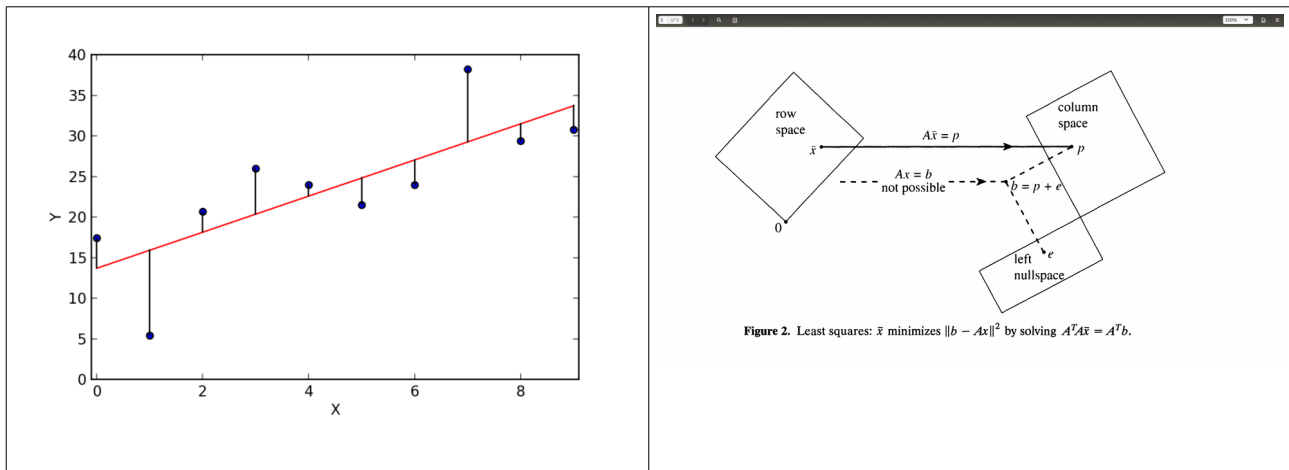


Illustration 4: Regression (Line Fitting) \Leftrightarrow Least squares: Projection and Error

Problem:

- If b is not in the column space, $Ax = b$ cannot be solved
 - but we still have to come up with a solution
- It is extremely common to have more equations than unknowns
 - i.e. more output data than input controls or more measurements than parameters to describe them.
- The data may lie close to a straight line, $b = C + Dt$
 - A parabola $C + Dt + Tt^2$ would come closer
- Whether we use polynomials or sines & cosines or exponentials
 - the problem is still linear in the coefficients C, D, E :

$$C + Dt_1 = b_1$$

•

•

•

$$C + Dt_m = b_m$$

or

$$C + Dt_1 + Et_1^2 = b_1$$

•

•

•

$$C + Dt_m + Et_m^2 = b_m$$

- There are $n = 2$ or $n = 3$ unknowns, and $m \gg n$
- There is no $x = (C, D)$ or $x = (C, D, E)$ that satisfies all the equations
- $Ax = b$ has a solution only when the points lie exactly on a line or a parabola
 - then b is in the column space of the m -by-2 or m -by-3 matrix A

Solution:

- The solution is to **make the error $b - Ax$ as small as possible**
- Since Ax can never leave the column space:
 - choose the closest point to b in that subspace. This point is the **projection p**
 - Then the **error vector $e = b - p$** has **minimal length**

- The solution is shown in Illustration 4 below:
 - the RHS image is the picture for least squares which shows the action over on the right hand side \rightarrow the splitting of b into $p + e$.
 - The LHS is an equivalent image showing, graphically, the data points projected on to a straight line with the associated error, e , the vertical difference between each data point, b , and the projected point, p .
- In short, the best combination $p = A\hat{x}$ is the projection of b onto the column space.
- The error e is perpendicular to that subspace.
 - Therefore $e = b - A\hat{x}$ is in the left nullspace:

$$A^T(b - A\hat{x}) = 0$$

or

$$A^T A \hat{x} = A^T b$$

- Calculus reaches the same linear equations by minimising the quadratic $\|b - Ax\|^2$.
- The chain rule multiplies both sides of $Ax = b$ by A^T to give the ‘normal equations’:

Normal Equations:

$$A^T A \hat{x} = A^T b$$

- They illustrate what is almost invariable true:
 - applications starting with a rectangular matrix A end up computing with the square symmetric matrix $A^T A$
- This matrix is **invertible** provided A has **independent columns** which means that the nullspace of A contains only $x = 0$.

Positive Definite

Not only is $A^T A$ symmetric but it is also Positive Definite if A is full rank (the n columns of A are linearly independent) and $A^T A$ passes the Positive Definite test: $x^T A^T A x > 0 \Rightarrow \text{True}$

If the Positive Definite Test passes then $x^T A^T A x = (Ax)^T (Ax) = \|Ax\|^2 > 0$ and we have a local minimum which, in this case, asserts that our projections, p , in the **column space** is positioned as close as possible to b with the minimum error, e , which is placed in the **null space**.

Example: Predict House Prices

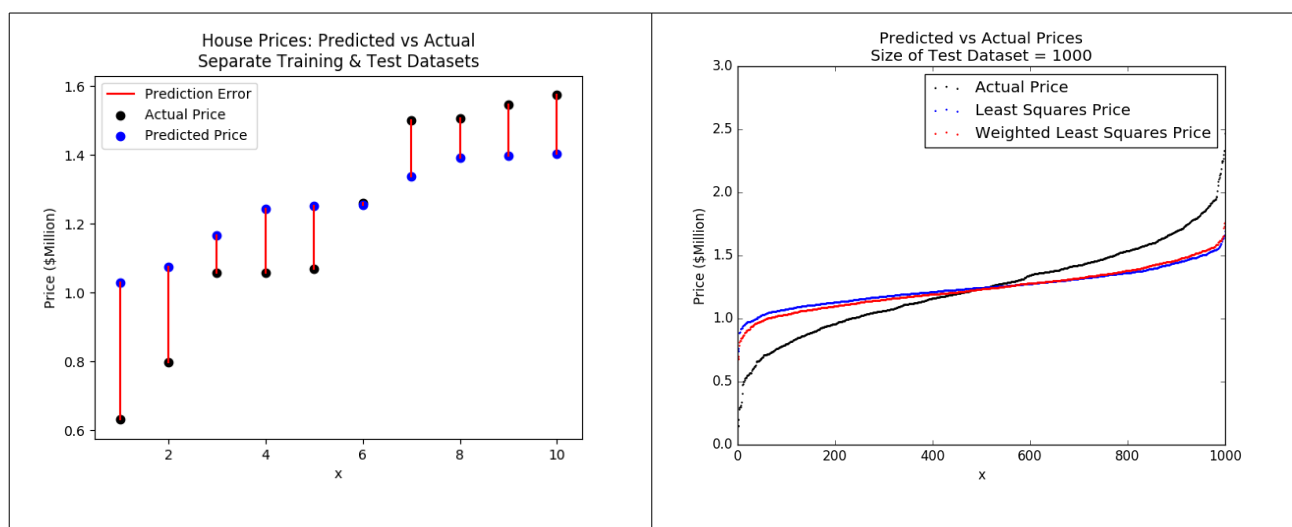


Illustration 5: Graphic of Predicted vs Actual Prices for
LHS: Test Dataset = 10; RHS: Test Dataset = 1000

(Weighted) Least Squares: Linear Algebra vs Probability

The key difference between the linear algebra and probability approaches to Least Squares is in what we are trying to optimise, as detailed in Table2. In the case of Least Squares (LS) the goal is to minimise the error. This, as we know, is a key property of semi-definite matrices because if the Positive Definite Test passes then we have a local minimum which, in this case, asserts that our projection, p , is positioned as close as possible to b with a minimum error, e .

| Linear Algebra | Probability |
|--|---|
| Least Squares (ls) | Maximum Likelihood (ml) |
| We want to Minimise the Error: $w_{ls} = \arg \min_w \ y - Xw\ ^2$ | We want to Maximise the Likelihood: $w_{ml} = \arg \max_w \ln p(y Xw, \sigma^2)$ |
| Which gives the Normalisation Equation : $X^T X w_{ls} = X^T y$ | We assume: <ul style="list-style-type: none"> • Data, X: x_1, \dots, x_n is iid Gaussian • Likelihood: $p(y) \sim N(X w_{ml}, \sigma^2 I)$ • Prior: $p(w_{ml}) \sim N(0, \lambda^{-1})$ • Error: $\epsilon_i = y_i - x^T w_{ml}$ |
| Solving for w_{ls} : $w_{ls} = (X^T X)^{-1} X^T y$ | Solving for w_{ml} : $E[w_{ml}] = w ; \text{Var}[w_{ml}] = \sigma^2 (X^T X)^{-1}$ |
| To make a prediction: $y_{new} = (X^T X) w_{ls}$ Histogram of y 's reveals a Gaussian distribution | To make a prediction: $y_{new} = (X^T X) w_{ml}$ with a confidence level dictated by σ^2 |
| Summary: $w_{ls} = \arg \min_w \ y - Xw\ ^2 \Leftrightarrow w_{ml} = \arg \max_w \ln p(y Xw, \sigma^2)$ | |
| Weighted Least Squares (Ridge Regression) | Maximum A Posteriori (Bayesian Modeling) |
| We want to Minimise the Regularised Error: $w_{rr} = \arg \min_w \ y - Xw\ ^2 + \lambda \ w\ ^2$ Ridge Regression (l_2): $\lambda \ w\ ^2$ Lasso Regression (l_1): $\lambda \ w\ $ | We want to Maximise the Bayesian Likelihood: $w_{map} = \arg \max_w \ln p(y Xw, \sigma^2) + \ln p(w)$ |
| Summary: $w_{rr} = (\lambda I + X^T X)^{-1} X^T y \Leftrightarrow w_{map} = (\lambda \sigma^2 I + X^T X)^{-1} X^T y$ | |
| λ is the regularisation term used to impose a penalty for large values of w <ul style="list-style-type: none"> • In a sense we are imposing a Prior Belief on what we think good values of w <ul style="list-style-type: none"> ◦ This is Baye's Rule. • $\lambda = 1$ for Least Squares, i.e no penalty is applied for large values of w σ^2 Is a Covariance matrix | |

Table2: (Weighted) Least Squares: Linear Algebra vs. Probability

In the case Maximum Likelihood (ML), we want to maximise the probability, or likelihood, that the price prediction, y , is generated from a Gaussian distribution governed by a mean and variance, as shown in Illustration 6. If we can estimate these distribution parameters, the mean and variance, from the input data, y and X , using Maximum Likelihood Estimation (MLE), then we can predict the price from test features. Once we have a predicted price we can estimate the error as the

difference between the actual price and the predicted price; Recall $e = b - p$. The errors are also drawn from a Gaussian distribution, as shown in Illustration 7.

What appears as remarkable at first glance, but logical and obvious from a mathematical perspective, is that Least Squares and Maximum Likelihood as well as Weighted Least Squares (WLS) and Maximum A Posteriori (MAP) are equivalent. The regularisation term in WLS is equivalent to adding a Prior belief to MAP that guides the mathematics towards the best solution, w.

Probability Distributions

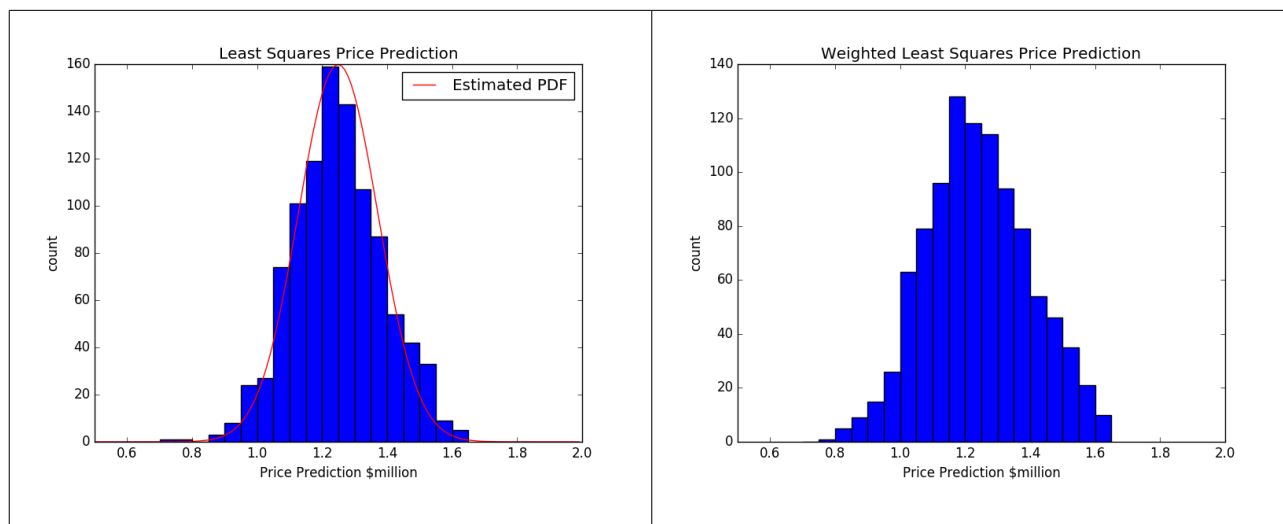


Illustration 6: Prediction Probability Distributions: Least Squares & Weighted Least Squares

In Illustration 6 above we have the prediction (or projection) probability distributions. On the LHS we have a histogram of the least squares predicted prices from the test data. Superimposed is the estimated probability distribution function (PDF), created by estimating the sample mean and variance of the prices associated with the training data. On the RHS is a histogram of the weighted least squares predicted prices. It is wider, due to higher variance, than the LS distribution, which is more reflective of the training data and thus makes for reduced error, which is mirrored in the error histograms in Illustration 7.

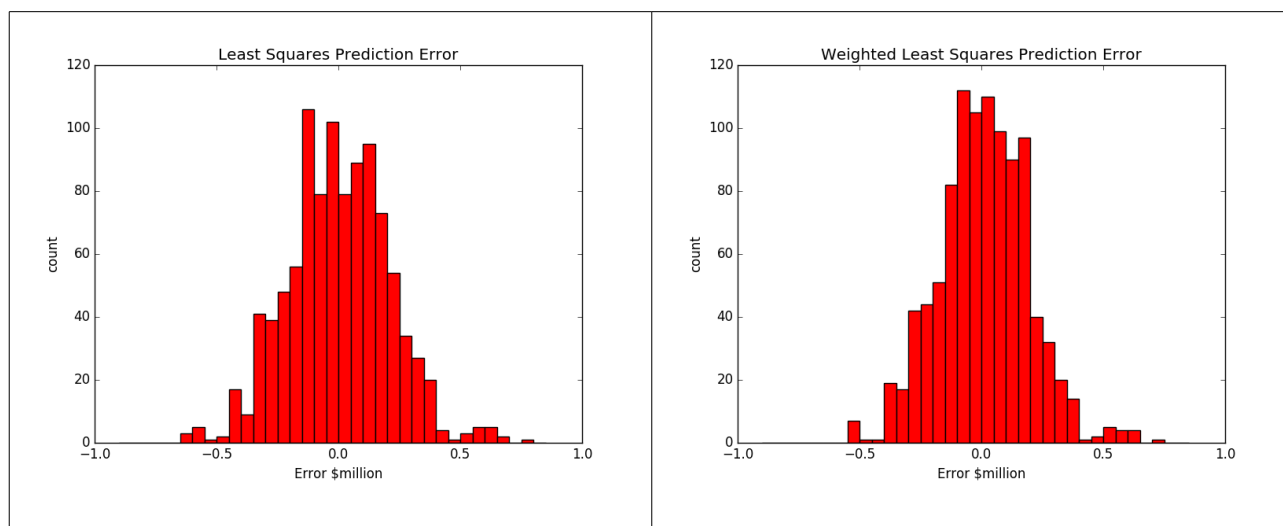


Illustration 7: Error Probability Distributions: Least Squares & Weighted Least Squares

Illustration 7 shows that the variance of the Weighted Least Squares prediction error is less than that of the Least Squares error, leading to better price predictions. As mentioned, adding the regularisation factor, or if you prefer, the prior probability of what we think makes a good solution, reduces the error and improves the performance of the prediction capability of the model.

Dimension Reduction

We can use Principle Component Analysis (PCA) or Singular Value Decomposition (SVD) to reduce the dimension of our data set and thus make the training of our model faster.

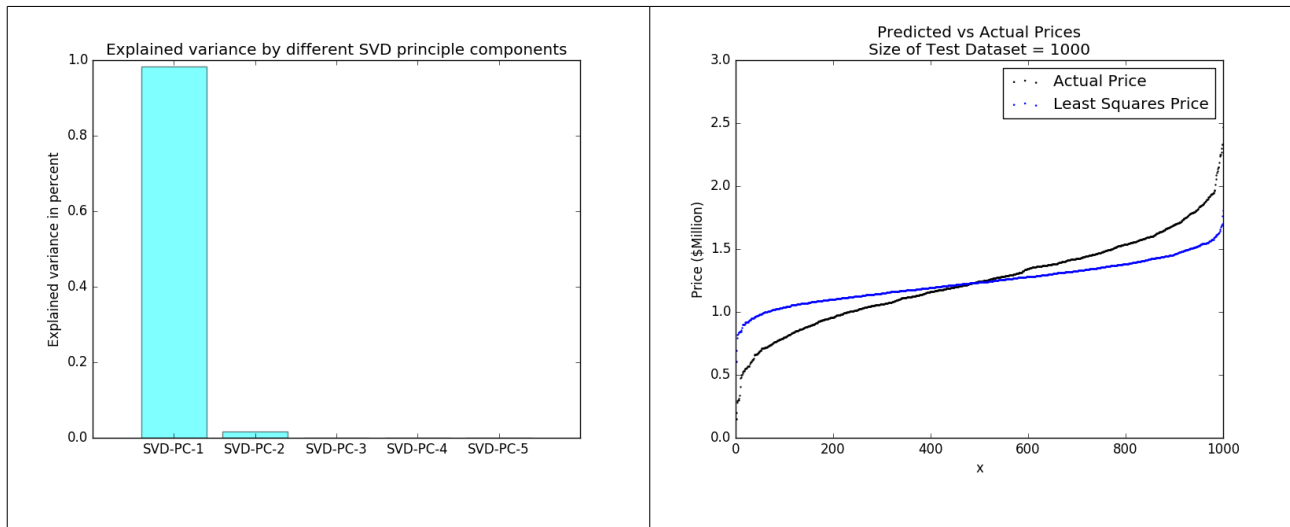


Illustration 8: Dimension Reduction using the SVD

The left image of Illustration 8 explains the variance in the system data between different SVD principle components and it is clear that most of the variance is due to a single principle component. It is important to note that the SVD components are projections of the eigenvectors in the original data matrix. So our dominant principle component could potentially consist of tranches made up from all of the original eigenvectors. The right hand side image shows the graph of the actual vs. predicted prices. The performance is on par with the weighted least squares model without dimension reduction. This topic is discussed further in the section of Diagonalisation.

Orthonormal Bases and the QR Factorisation

We prefer that our subspaces have Orthonormal bases:

- Column vectors are perpendicular to each other
- Each Column vector is of Unit Length

We can use the **Gram-Schmidt** process to convert a matrix with independent column vectors, say A , with columns a, b, c , to an Orthonormal matrix, Q , with columns q_1, q_2, q_3 .

The matrices A and Q are related through the matrix R , which is an upper triangular matrix, through the factorisation $A = QR$.

The actual mechanics of Gram-Schmidt and QR factorisation are straight forward and I refer the reader to Strang's Linear Algebra text book if they are intrested in going deeper into this topic.

Key property of orthogonal matrices, and their complex cousins, unitary matrices:

$$\text{Orthonormal: } Q^T Q = I \text{ or } Q^T = Q^{-1} \quad \Leftrightarrow \quad \text{Unitary: } U^H U = I \text{ or } U^H = U^{-1}$$

Matrix Diagonalisation

Spectral Theorem

A Spectral Theorem is a result about when a linear operator or matrix can be diagonalised. As is shown in Illustration 9 below, this is extremely useful because computations involving a diagonalisable matrix can often be reduced to much simpler computations involving the corresponding diagonal matrix.

The key equation is $\mathbf{Ax} = \lambda\mathbf{x}$. Most vectors, \mathbf{x} , will not satisfy such an equation. They change direction when multiplied by \mathbf{A} , so that \mathbf{Ax} is not a multiple of \mathbf{x} . This means that only certain special numbers λ are eigenvalues, and only certain special vectors \mathbf{x} are eigenvectors. We can watch the behaviour of each eigenvector, and then combine these '**normal modes**' to find the solution. To say the same thing in another way, ***the underlying (coupling) matrix can be diagonalised.***

Eigenvalue / Eigenvector Decomposition

The concept of eigenvalues and eigenvectors is critical to understanding many areas of applications. Two of the most important areas where it plays a role is in understanding the ***powers of matrices*** and ***differential equations***.

Powers of A

The evaluation of the powers of a matrix, \mathbf{A} , can be performed with eigenvalues & eigenvectors:

$$\mathbf{A}^M$$

where M is a large integer.

Knowing the eigenvalues and eigenvectors of \mathbf{A} allows for a significant ease in computational expense.

Assuming we have all the eigenvalues and eigenvectors of \mathbf{A} , then

$$\begin{aligned}\mathbf{A} \mathbf{x}_1 &= \lambda_1 \mathbf{x}_1 \\ \mathbf{A} \mathbf{x}_2 &= \lambda_2 \mathbf{x}_2 \\ &\vdots \\ \mathbf{A} \mathbf{x}_n &= \lambda_n \mathbf{x}_n\end{aligned}$$

This collection of eigenvalues and eigenvectors gives the matrix system:

$$\mathbf{AS} = \mathbf{S}\mathbf{\Lambda}$$

where the columns of the matrix, \mathbf{S} , are the ***eigenvectors of A***:

$$\mathbf{S} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)$$

And $\mathbf{\Lambda}$ is a matrix whose ***diagonals*** are the corresponding eigenvalues:

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}$$

By multiplying $\mathbf{AS} = \mathbf{S}\mathbf{\Lambda}$ on the right by \mathbf{S}^{-1} , the matrix \mathbf{A} can be rewritten as:

$$\mathbf{A} = \mathbf{S} \mathbf{\Lambda} \mathbf{S}^{-1} .$$

The final observation comes from:

$$\mathbf{A}^2 = (\mathbf{S} \mathbf{\Lambda} \mathbf{S}^{-1})(\mathbf{S} \mathbf{\Lambda} \mathbf{S}^{-1}) = (\mathbf{S} \mathbf{\Lambda}^2 \mathbf{S}^{-1}) .$$

This then generalises to:

$$\mathbf{A}^M = \mathbf{S} \mathbf{\Lambda}^M \mathbf{S}^{-1}$$

where the matrix $\mathbf{\Lambda}^M$ is easily calculated by:

$$\mathbf{\Lambda}^M = \begin{bmatrix} \lambda_1^M & 0 & \cdots & 0 \\ 0 & \lambda_2^M & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & \cdots & 0 & \lambda_n^M \end{bmatrix}$$

Since raising the diagonal terms to the M^{th} power is easily accomplished, the matrix \mathbf{A}^M can then be easily calculated by multiplying the three matrices, \mathbf{S} , $\mathbf{\Lambda}^M$ and \mathbf{S}^{-1} together.

Differential Equations

Consider the system of differential equations:

$$\frac{d\mathbf{u}}{dt} = \mathbf{A} \mathbf{u}$$

For some vector $\mathbf{u}(t)$ representing a dynamical system of variables, where the matrix \mathbf{A} determines the (coupled) interaction among these variables

Assuming a solution of the form

$$\mathbf{u} = \mathbf{x} \exp(\lambda t)$$

results in the eigenvalue problem:

$$\mathbf{A} \mathbf{x} = \lambda \mathbf{x}$$

How do we find the eigenvalues and eigenvectors?

To consider this problem we rewrite the eigenvalue problem as

$$\mathbf{A} \mathbf{x} - \lambda \mathbf{I} \mathbf{x} = (\mathbf{A} - \lambda \mathbf{I}) \mathbf{x} = \mathbf{0}$$

As we are only interested in non-trivial solutions, the solution $\mathbf{x} = \mathbf{0}$ is not sufficient.

If the determinant of the matrix $(\mathbf{A} - \lambda \mathbf{I})$ is zero then the matrix is **singular** and its inverse,

$(\mathbf{A} - \lambda \mathbf{I})^{-1}$, cannot be found. Although there is no longer a guarantee that there is a solution, it is the only scenario which allows for the possibility $\mathbf{x} \neq \mathbf{0}$. It is this condition which allows for the construction of eigenvalues and eigenvectors.

Indeed, we choose the **eigenvalues**, λ , so that this condition holds and the matrix is **singular**. In other words, the column matrix of **eigenvectors**, \mathbf{S} , lies in the **null space**.

Example

$$\text{Differential equation:} \quad \frac{d\mathbf{u}}{dt} = \mathbf{A} \mathbf{u} = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} \mathbf{u}$$

The first step is always to find the eigenvalues (-1 & -3) and the eigenvectors:

$$A \begin{bmatrix} 1 \\ 1 \end{bmatrix} = (-1) \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{and} \quad A \begin{bmatrix} 1 \\ -1 \end{bmatrix} = (-3) \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

The general solution is a combination of pure exponential solutions. These are solutions of the special form $c e^{\lambda t} x$, where λ is an eigenvalue of A and x is its eigenvector. In this 2 x 2 example, there are two pure exponentials to be combined:

| | |
|---|--|
| <u>Calculus</u> | <u>Linear Algebra</u> |
| <p>Solution</p> $u(t) = c_1 e^{\lambda_1 t} x_1 + c_2 e^{\lambda_2 t} x_2 \quad \text{or}$ | $u = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} e^{-t} & \\ & e^{-3t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$ |

At time 0, when the exponentials are $e^0 = 1$, $u(0)$ determines c_1 and c_2 :

Initial Condition

$$u(0) = c_1 x_1 + c_2 x_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = S c$$

Substituting back into the solution equation, the complete solution is:

$$u(t) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} e^{-t} & \\ & e^{-3t} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = S \begin{bmatrix} e^{-t} & \\ & e^{-3t} \end{bmatrix} S^{-1} u(0)$$

Therefore, $S e^{\Lambda t} S^{-1} u(0)$ solves the differential equation:

$$u(t) = S e^{\Lambda t} S^{-1} u(0) \quad \text{with} \quad \Lambda = \begin{bmatrix} -1 & \\ & -3 \end{bmatrix} \quad \text{and} \quad e^{\Lambda t} = \begin{bmatrix} e^{-t} & \\ & e^{-3t} \end{bmatrix}$$

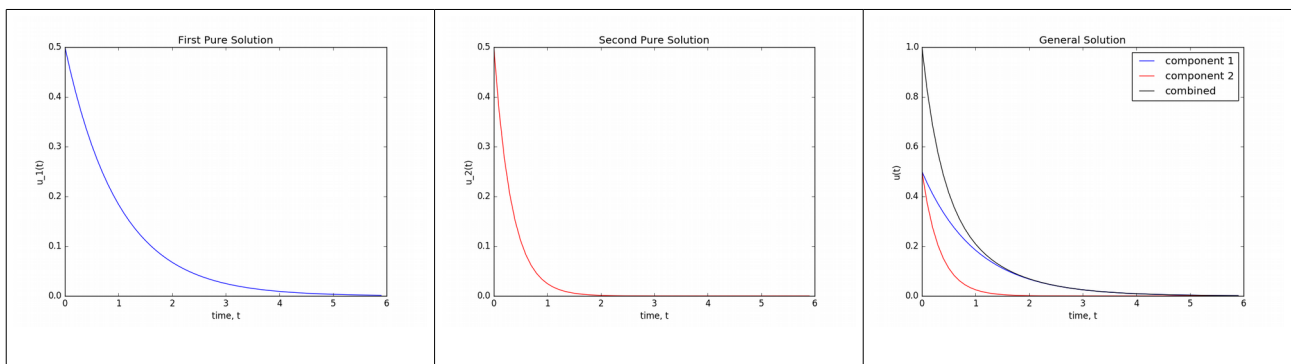


Illustration 9: Graphical Solution: two pure solutions combine make the general solution, $u(t)$

If $A^T = A$ then the eigenvectors, q_i , of A are orthonormal / unitary so that

$$A q_i = \lambda_i q_i$$

and

$$A = Q \Lambda Q^T$$

Every Symmetric Matrix is a combination of perpendicular Projection Matrices:

$$A = Q \Lambda Q^T = \begin{bmatrix} | & | & \dots & | \\ q_1 & q_2 & & \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \end{bmatrix} \begin{bmatrix} - & q_1^T & - \\ - & q_2^T & - \\ & \vdots & \end{bmatrix} = \lambda_1 q_1 q_1^T + \lambda_2 q_2 q_2^T$$

where $q_1 q_1^T$ and $q_2 q_2^T$ are **Projection Matrices**.

Stability of Differential equations

Stability is governed by the factors $e^{\lambda_i t}$. If they all approach zero, then $u(t)$ approaches zero; if they are all bounded, then $u(t)$ stays bounded; if one of them blows up, then except for very special starting conditions the solution will blow up. Furthermore, the size of $e^{\lambda_i t}$ depends only on the **real part of λ** . It is only the real parts of the eigenvalues that govern stability. The **imaginary part** is producing oscillations, but the amplitude comes from the **real part**.

If $\lambda = a + ib$, then:

$$e^{\lambda t} = e^{at} e^{ibt} = e^{at} (\cos bt + i \sin bt) \quad \text{and the magnitude is } |e^{\lambda t}| = e^{at} \quad \text{as } |e^{ibt}| = 1$$

This decays for $a < 0$, it is constant for $a = 0$, and it explodes for $a > 0$.

In summary, the differential equation $\frac{du}{dt} = \mathbf{A}u$ is:

- **Stable** and $e^{At} \rightarrow 0$ whenever all $\Re \lambda_i < 0$
- **Neutrally stable** when $\Re \lambda_i \leq 0$ and $\Re \lambda_1 = 0$
- **Unstable** and e^{At} is unbounded if any eigenvalue has $\Re \lambda_1 > 0$

The stability tests are:

- $\Re \lambda_i < 0$: **The trace $a + d$ must be negative**
- $\Re \lambda_i < 0$: **The determinant $ad - bc$ must be positive**

Singular Value Decomposition (SVD)

A singular value decomposition (SVD) is a factorisation of a matrix into a number of constitutive components and is closely associated with the eigenvalue / eigenvector factorisation, $Q\Lambda Q^T$, of a positive definite matrix. The SVD is essentially a transformation that stretches / compresses and rotates a given set of vectors. The Singular Value Decomposition (SVD) of a *rectangular matrix*, A :

$$A = U\Sigma V^T = (\text{orthogonal})(\text{diagonal})(\text{orthogonal})$$

$$A = U \Sigma V^T = \begin{bmatrix} | & | & \cdots \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \end{bmatrix} \begin{bmatrix} - & v_1^T & - \\ - & v_2^T & - \\ & \vdots & \end{bmatrix} = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T$$

Illustration 10: SVD matrix factorisation

Remark 1: For a *positive definite matrix*; Σ is Λ and $U\Sigma V^T$ is identical to $Q\Lambda Q^T$

For a *complex matrix*, Σ remains real but U and V become *unitary*

Remark 2: U and V give orthonormal bases for all fundamental subspaces:

| | | | |
|-------|---------|------------------|------------------------------|
| First | r | columns of U : | Column space of A |
| Last | $m - r$ | columns of U : | Left nullspace of A |
| First | r | columns of V : | Row space of A |
| Last | $n - r$ | columns of V : | Nullspace of A |

Remark 3: Eigenvectors of AA^T and $A^T A$ must go into the columns of U and V

- The r singular values, σ_i , on the diagonal Σ (m by m), are the square roots of the non-zero eigenvalues of both AA^T and $A^T A$

Remark 4: Matrices can be diagonalised either by an eigenvalue or an SVD decomposition, but:

- The SVD performs the diagonalisation using two different bases, U and V , while the eigenvalue method uses a single basis, X
- The SVD method uses an orthonormal basis while the basis vectors in X , while linearly independent, are not generally orthogonal
- Finally, the SVD is guaranteed to exist for any matrix A , while the same is not true, even for square matrices, for the eigenvalue decomposition

Diagonalisation

In the process of diagonalisation, the correct coordinates, or basis functions, are revealed that reduce the given system to its low-dimensional essence. The key idea behind diagonalisation is that there exists an ideal basis in which the Covariance matrix, C_x , can be written (diagonalised) so that in this basis, all redundancies have been removed, and the largest variances of particular measurements are ordered, i.e. the system has been written in terms of its *principle components*, or in a *proper orthogonal decomposition*. We compare diagonalisation using eigenvalue/eigenvector and SVD factorisation in Table 3.

Recall that the Covariance matrix, $C_x = \frac{1}{n-1} XX^T$

where the matrix X contains the experimental data of a system.

| <i>Eigenvectors & Eigenvalues</i> | <i>Singular Value Decomposition</i> |
|--|---|
| XX^T is a square, symmetric matrix meaning it has real & distinct eigenvalues. Therefore: | The SVD can diagonalise any matrix by working with the pair of bases, U & V : |
| $XX^T = SAS^T$ | $X = U\Sigma V^T$ |
| where S is a matrix of eigenvectors of XX^T arranged in columns. | Where U is the unitary transformation |
| Instead of working directly with X , we use the principle component basis: | Instead of working directly with X , we use the principle component basis: |
| $Y = S^T X$ | $Y = U^T X$ |
| For this new basis we consider its covariance: | For this new basis we consider its covariance: |
| $C_Y = \frac{1}{n-1} YY^T$ \vdots $C_Y = \frac{1}{n-1} \Lambda$ | $C_Y = \frac{1}{n-1} YY^T$ \vdots $C_Y = \frac{1}{n-1} \Sigma^2$ |
| In this basis, the <i>principle components</i> are the eigenvectors of XX^T with the interpretation that the j^{th} diagonal value of C_Y is the variance along x_j , the j^{th} column of S . | This makes explicit the connection between the SVD and the eigenvalue method: $\Sigma^2 = \Lambda$ |

Table 3: Diagonalisation: Eigenvalue/Eigenvector vs. SVD

Overall, the SVD method is more robust and should be used as we do so to compute the Principle Components of the Iris dataset.

Principal Component Analysis (PCA)

The PCA can be computed using Singular Value Decomposition (SVD) decomposition, as shown in Illustration 11 below.

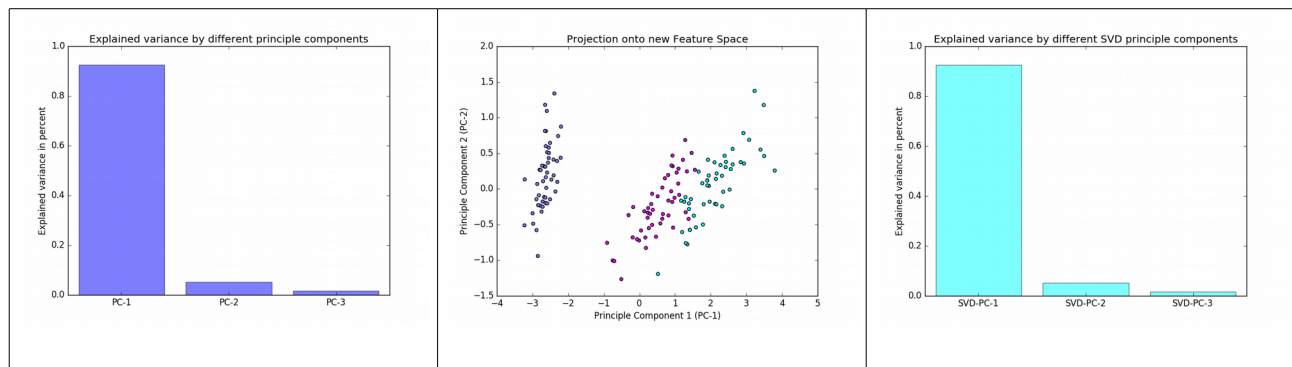


Illustration 11: PCA & SVD example with Iris Data-set

Principle components can also be computed using an eigenvalue/eigenvector decomposition but as mentioned above there is no guarantee that a matrix can be factorised this way but every matrix is guaranteed to have an SVD.

Direct method for Solving $Ax = b$

It is possible to find an efficient numerical solution to the 2nd order differential Heat Equation using an implicit Finite Difference Method and Linear Algebra as shown in Illustration 12.

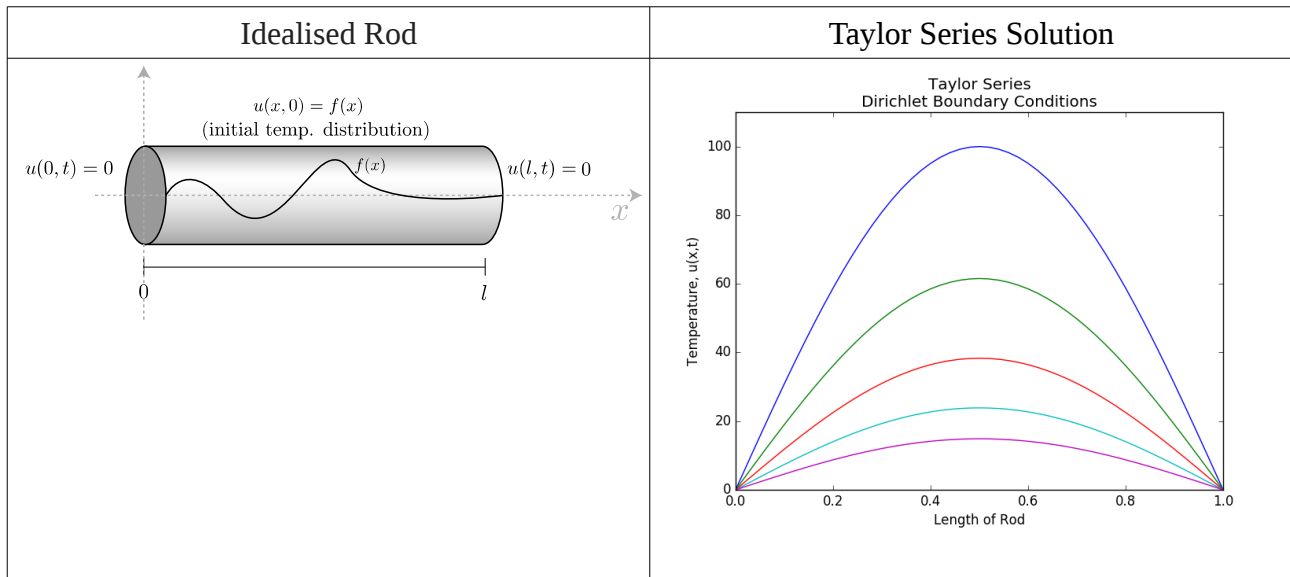


Illustration 12: Illustration 1: Graphical Solutions to the Heat Equation

This example illustrates the special properties that coefficient matrices frequently have. Large matrices almost always have a clear pattern, frequently a pattern of symmetry, and very many zero entries. Since a sparse matrix contains far fewer than n^2 pieces of information, the computations ought to be fast. We look at band matrices, to see how concentration near the diagonal speeds up elimination.

The continuous problem asks for $u(x)$ at every x , and a computer cannot solve it exactly. It has to be approximated by a discrete problem, the more unknowns we keep, the better will be the accuracy but the greater the expense. Let us consider the heat equation for a one space variable which could be used to model heat conduction in a rod. From [1], the equation is:

$$-\nabla^2 u = f(x, t) \quad , \quad 0 \leq x \leq 1 \quad [1]$$

where $u = f(x, t)$ is a function of two variables x and t . Here

- x is the space variable, so $x \in [0, 1]$, where 1 is the length of the rod.
- t is the time variable, so $t \geq 0$.

We assume the following initial condition:

$$u(x, 0) = f(x, 0) \quad \forall x \in [0, 1] \quad [2]$$

where the function, f , is given, and the Dirichlet boundary conditions are:

$$u(0, t) = 0 = u(1, t) \quad \forall t > 0 \quad [3]$$

The result is a two-point boundary-value problem, describing not a transient but a steady-state phenomenon, the temperature distribution in a rod with fixed ends at 0° and with a heat source $f(x, 0)$.

Our goal is to produce a discrete problem, i.e. a problem in linear algebra. For that reason we can only accept a finite amount of information about $f(x, t)$, say its value at n equally spaced points $x = h, x = 2h, \dots, x = nh$. We compute approximate values u_1, \dots, u_n for the true solution u at these same points. At the ends $x = 0$ and $x = 1 = (n + 1)h$, the boundary values are u_0 and $u_{n+1} = 0$.

The first task is to replace the derivative, $\frac{d^2 u}{dx^2}$, carried out in a two step process:

Step 1: The first derivative can be approximated by stopping $\frac{\Delta u}{\Delta x}$ at a finite step size, and not permitting h (or Δx) to approach zero. The difference Δu can be forward, backward, or centered:

| | | |
|---|---|---|
| $\frac{\Delta u}{\Delta x} = \frac{u(x+h) - u(x)}{h}$ | $\frac{\Delta u}{\Delta x} = \frac{u(x) - u(x-h)}{h}$ | $\frac{\Delta u}{\Delta x} = \frac{u(x+h) - u(x-h)}{h}$ |
| Forward | Backward | Centered |

The centered version is symmetric about x and is the most accurate.

Step 2: For the 2nd derivative there is just one combination that uses only the values x and $x \pm h$:

Second Difference:

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

This also has the merit of being symmetric about x . To repeat, the RHS approaches the true value of

$$\frac{d^2 u}{dx^2} \text{ as } h \rightarrow 0, \text{ but we have to stop at a +ive } x \text{ and}$$

- At each mesh point $x = jh$, the equation $-\frac{d^2 u}{dx^2} = f(x)$ is replaced by its discrete analogue.
- We multiplied through by h^2 to reach n equations $\mathbf{A} \mathbf{u} = \mathbf{b}$ giving:

Difference Equation:

$$-u_{j+1} + 2u_j - u_{j-1} = h^2 f(jh) \text{ for } j = 1, \dots, n$$

- The first and last equations ($j = 1$ and $j = n$) include u_0 and $u_{n+1} = 0$, i.e the BCs

The structure of these n equations can be better visualised in matrix form. If we choose $h = 1/6$, to get a 5 by 5 matrix \mathbf{A} :

Matrix equation:

$$\begin{bmatrix} 2 & 1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = h^2 \begin{bmatrix} f(h) \\ f(2h) \\ f(3h) \\ f(4h) \\ f(5h) \end{bmatrix}$$

The matrix \mathbf{A} possesses many special properties, and three of these properties are fundamental:

- The matrix \mathbf{A} is **Tridiagonal**:
 - All non-zero entries lie on the main diagonal and the two adjacent diagonals.
 - Outside this band all entries are $a_{ij} = 0$

- These zeros will bring a tremendous simplification to Gaussian elimination
- The matrix is **Symmetric**:
 - each entry a_{ij} equals its mirror image a_{ji} , so that $A^T = A$
 - upper triangular U will be transpose of lower triangular L and $A = LDL^T$
 - symmetry of A reflects the symmetry of $\frac{d^2 u}{dx^2}$
 - an odd derivative like $\frac{du}{dx}$ or $\frac{d^3 u}{dx^3}$ would destroy the symmetry
- The matrix is **Positive Definite**
 - this extra property says that the pivots are positive
 - row exchanges are unnecessary in theory and in practice

We return to the fact that A is tridiagonal:

The first stage of the elimination process produces zeros below the first pivot. Compared with a general 5 by 5 matrix, that step displays two major simplifications:

1. There was only one non-zero entry below the pivot
2. The pivot row was very short

In addition:

- The new pivot came from a single multiplication-subtraction
- The tridiagonal pattern is preserved

The final result is the $LDU = LDL^T$ **factorisation** of A .

$$A = \begin{bmatrix} 1 & & & & \\ -\frac{1}{2} & 1 & & & \\ & -\frac{2}{3} & 1 & & \\ & & -\frac{3}{4} & 1 & \\ & & & -\frac{4}{5} & 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{1} & & & & \\ & \frac{3}{2} & & & \\ & & \frac{4}{3} & & \\ & & & \frac{5}{4} & \\ & & & & \frac{6}{5} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & & & \\ & 1 & -\frac{2}{3} & & \\ & & 1 & -\frac{3}{4} & \\ & & & 1 & -\frac{4}{5} \\ & & & & 1 \end{bmatrix}$$

- The L and U factors of a tridiagonal matrix are bidiagonal
- L and U are transposes of one another, as expected from symmetry
- The pivots are all positive and their product is the determinant of A , $\det = 6$
- the pivots are converging to 1, as n gets large

These sparse factors L and U completely change the usual operation count.

- Elimination on each column needs only two operations, as above, and there are n columns
- In place of $\frac{n^3}{3}$ operations we need only $2n$

- Tridiagonal systems $Ax = b$ can be solved almost instantly
- The cost of solving a tridiagonal system is proportional to n

Key point:

- In solving $Ax = b$, we are actually worse off knowing A^{-1} than knowing L and U
- Multiplying A^{-1} by b to give $A^{-1}b$ takes n^2 steps whereas
- Two triangular steps are better:

$$x = A^{-1}b \text{ separates into } Lc = b \text{ and } Ux = c$$

- Meaning, $4n$ are sufficient for the forward elimination and back-substitution that produces

$$x = U^{-1}c = U^{-1}L^{-1}b = A^{-1}b$$

Iterative methods for Solving $Ax = b$

Gaussian elimination can be used to solve $Ax = b$ in a finite number of steps but for large matrices we may have to settle for an approximation of x that can be obtained more quickly. In this section there is an outline of methods that start from any initial guess x_0 , and produce an improved approximation x_{k+1} from the previous x_k .

- Jacobi
 - For a large matrix A , there is a very practical difficulty. The Jacobi iteration requires us to keep all components of x_k until the calculation of x_{k+1} is complete.
- Gauss-Seidel
 - Simple tweak to Jacobi method. Use updated values of the solution as soon as they are available instead of waiting for the values in the whole grid to be updated.
- Successive Over-Relaxation (SOR)
 - Improves on the Gauss-Seidel method by using in the update a linear combination of the previous and the current solution.
- Conjugate Gradient method
 - It is direct rather than iterative, but unlike elimination, it can be stopped part way.

Fourier Series and Fourier Transform

Fourier Series

Hilbert space:

- After studying \mathbf{R}^n , it is natural to think of the space \mathbf{R}^∞
- It contains all vectors $v = (v_1, v_2, v_3, \dots)$ with an infinite sequence of components
 - This space is too big when there is no control on the size of the components v_i
- A better idea is to keep the familiar definition of length, using a sum of squares, but
 - *to include only those vectors that have a **finite length**:*

$$\text{Length squared: } \|v\|^2 = v_1^2 + v_2^2 + v_3^2 + \dots$$

- The infinite series must converge to a finite sum
- Vectors with finite length can be added & multiplied by scalars so they form a vector space
 - This is the celebrated **Hilbert Space**
- Hilbert space is the natural way to let the **number of dimensions become infinite**, and at the same time to keep the **geometry of ordinary Euclidean space**
- In Hilbert space, **'vectors' can turn into functions**

Lengths and Inner Products

- For the length of a function, summation is replaced by **integration**:

$$\text{Length } \|f\| \text{ of function } \|f\|^2 = \int_0^{2\pi} (f(x))^2 dx = \int_0^{2\pi} (\sin x)^2 dx = \pi$$

- Our Hilbert space has become a **function space**:
 - the vectors are functions
 - we can measure their length
 - and the space contains all those functions that have a finite length
- To find the **inner product** of two functions we use integration instead of summation
- If $f(x) = \sin(x)$ and $g(x) = \cos(x)$, then their **inner product** is:

$$(f, g) = \int_0^{2\pi} f(x)g(x) dx = \int_0^{2\pi} \sin x \cos x dx = 0$$

- This is exactly like the vector inner product, $f^T g$
- Of course, with the inner product equal to zero, $\sin(x)$ and $\cos(x)$ are **orthogonal**

The Fourier Series

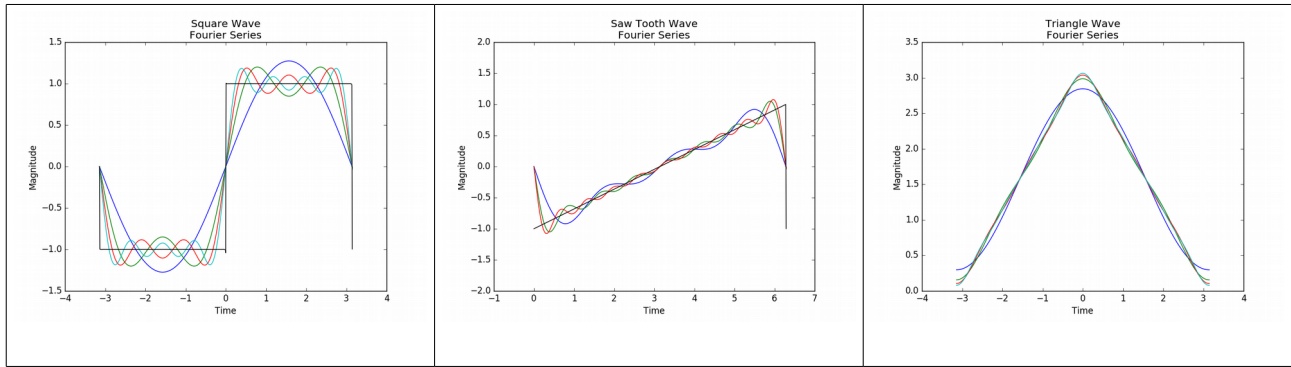


Illustration 13: Waveforms and their corresponding Fourier Series(taken from Fourier paper / Discrete Fourier Series section)

- The Fourier Series of a function is an expansion into sines and cosines:

$$f(x) = a_0 + a_1 \cos x + b_1 \sin x + a_2 \cos 2x + b_2 \sin 2x + \dots$$

- To compute a coefficient like b_1 take the inner product of both sides with $\sin x$:

$$\int_0^{2\pi} f(x) \sin x \, dx = a_0 \int_0^{2\pi} \sin x \, dx + a_1 \int_0^{2\pi} \cos x \sin x \, dx + b_1 \int_0^{2\pi} (\sin x)^2 \, dx + \dots$$

- On RHS, every integral is zero except one – the one in which $\sin x$ multiplies itself
 - i.e. the sines and cosines are **mutually orthogonal**
- Therefore, b_1 is the LHS divided by that one non-zero integral:

$$b_1 = \frac{\int_0^{2\pi} f(x) \sin x \, dx}{\int_0^{2\pi} (\sin x)^2 \, dx} = \frac{(f, \sin x)}{(\sin x, \sin x)}$$

- a_1 would have $\cos x$ in place of $\sin x$, and a_2 would use $\cos 2x$
- The key point is to see the analogy with **projections**:
 - The component of the vector b along the line spanned by a is $b^T a / a^T a$
 - A Fourier Series is projecting $f(x)$ on to $\sin x$**
 - Its component p in this direction is exactly $b_1 \sin x$
 - b_1 is the **least squares solution** of the inconsistent equation $b_1 \sin x = f(x)$
 - This brings $b_1 \sin x$ as close as possible to $f(x)$
 - All the terms in the series are projections onto a sine or cosine
 - Since the sines and cosines are orthogonal:

the Fourier Series gives the coordinates of the ‘vector’ $f(x)$ wrt a set of perpendicular axes

Fourier Transform

The Fourier series is linear algebra in infinite dimensions. The ‘vectors’ are functions $f(x)$; they are projected onto the sines and cosines; that produces the Fourier coefficients a_k and b_k . From this infinite sequence of sines and cosines, multiplied by a_k and b_k , we can construct $f(x)$. That is the classical case, which Fourier dreamt about, but in actual calculations it is the **discrete Fourier transform** that we compute. Fourier still lives, but in finite dimensions.

This section begins with w and its properties, moves to F^{-1} , and ends with the **Fast Fourier Transform**, the **FFT**. As discussed in the Fourier paper, the great application is *filtering*, and the key to its success is the *convolution rule*.

Complex roots of unity

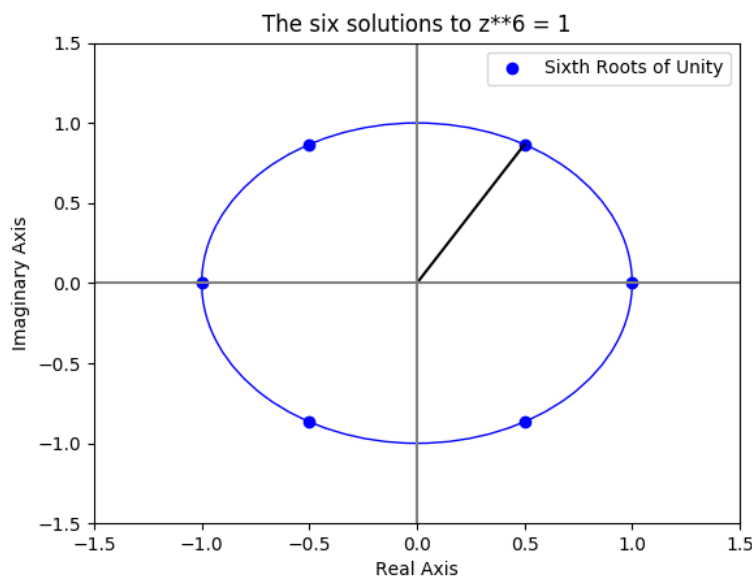


Illustration 14: The six solutions to $z^6 = 1$ are $1, w, w^2, \dots, w^5$

We want to compute the roots of $w^n = 1$, where $w_n = \exp(in\theta) = \exp(i2\pi/n) = \cos(2\pi/n) + i\sin(2\pi/n)$. The n roots are known as the n^{th} roots of unity and are evenly spaced around the Unit Circle. We use $w_n = \exp(i \cdot 2\pi/n)$ rather than $w_n = \cos(2\pi/n) + i \cdot \sin(2\pi/n)$ because $w_n = \exp(i \cdot 2\pi/n)$ is the easier form when working with powers.

The solutions to $w^6 = 1$ are:

- $w_6^0 = (\exp(i \cdot 2\pi/6))^0$
- $w_6^1 = (\exp(i \cdot 2\pi/6))^1$
- $w_6^2 = (\exp(i \cdot 2\pi/6))^2$
- $w_6^3 = (\exp(i \cdot 2\pi/6))^3$
- $w_6^4 = (\exp(i \cdot 2\pi/6))^4$
- $w_6^5 = (\exp(i \cdot 2\pi/6))^5$

The **sum of sixth roots**, $1 + w_6 + w_6^2 + \dots + w_6^5$, must equal zero.

Fourier matrix and Its Inverse

In the continuous case, the Fourier series can produce $f(x)$ over a whole interval. It uses infinitely many sines and cosines (or exponentials). In the discrete case, with only n coefficients, c_0, \dots, c_{n-1} to choose, we only ask for *equality at n points*. That gives n equations for $Fc = y$.

$$F_n c = y \rightarrow \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w^2 & w^4 & \cdots & w^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{2(n-1)} & \cdots & w^{(n-1)^2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

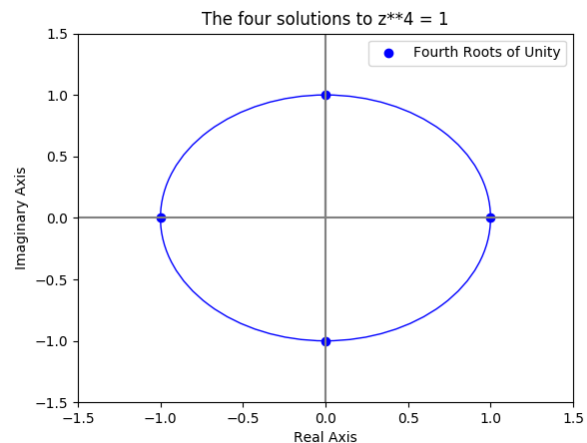
For every n , the matrix connecting y to c can be inverted. The inverse matrix is built from the powers of $w^{-1} = 1/w = \bar{w}$:

$$F_n^{-1} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w^{-1} & w^{-2} & \cdots & w^{-(n-1)} \\ 1 & w^{-2} & w^{-4} & \cdots & w^{-2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{-(n-1)} & w^{-2(n-1)} & \cdots & w^{-(n-1)^2} \end{bmatrix} = \frac{\bar{F}}{n}$$

Example: $n = 4$:

- Compute the 4th roots of unity:

- $w_4^0 = w_4^4 = (\exp(i \cdot 2\pi/4))^0 = (-i) \cdot i = 1$
- $w_4^1 = (\exp(i \cdot 2\pi/4))^1 = i$
- $w_4^2 = (\exp(i \cdot 2\pi/4))^2 = i^2 = -1$
- $w_4^3 = (\exp(i \cdot 2\pi/4))^3 = -1 \cdot i = -i$



- Complete the Fourier Matrix

$$\circ F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

- Key properties of the Fourier Matrix:
 - Columns are orthogonal, making it a Hermitian matrix

- Inner product of any two columns is equal to zero
- To make the columns orthonormal, divide by 2
 - The matrix is now Unitary because
 - $F_4^H F_4 = I$ where $F^H = F^{-1}$
 - meaning that F^{-1} is easy to compute

Fast Fourier Transform

Fourier Analysis is the best way to analyze a waveform into its frequencies and taking a signal apart. The reverse process takes it back. Each frequency component goes its own way, as an eigenvector, and then recombine into the solution. The analysis and synthesis of signals is central to scientific computing.

We want to show that $Fc = y$ and $F^{-1}y = c$ can be done quickly; rather than $O(n^2)$ for the DFT the **FFT executes in $O(n \log n)$** . The key is in the relation of F_4 to F_2 – or rather two copies of F_2 , which go into a matrix F_4^* . The same relation appears between F_{64} to F_{32} matrices. An F_{64} matrix has Roots of Unity, $w^{64}=1$, and the first root goes 1 / 64 round the Unit Circle. An F_{32} matrix has complex roots of unity, $w^{32}=1$, and the first root goes 1/32 times around the Unit Circle, i.e. twice as far as the 1st root of F_{64} .

Key Point: $w_{64}^2 = w_{32}$

To show the improvement in performance of the FFT over the DFT here are experimental times to transform an array with $\text{len}(x) = 10240$:

DFT: 12.976 seconds # DFT by matrix multiplication
FFT: 0.0723 seconds # FFT by recursion and matrix factorisation
fft: 0.0013 seconds # FFT by numpy.fft library

Matrix Factorisation

$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & -1 & \\ & & & -i \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ & 1 & 1 \\ & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} = W * F * P$$

where

- **W** consists of W_4 values
- **F** consists of two copies of F_2
- **P** is Permutation matrix that divides x into even and odd vectors

If we started with F_8 , the middle matrix, F , would contain two copies of F_4 , each of which would be split as above. Thus the **FFT** amounts to a giant **Matrix Factorisation of the Fourier Matrix**.

FFT Butterfly

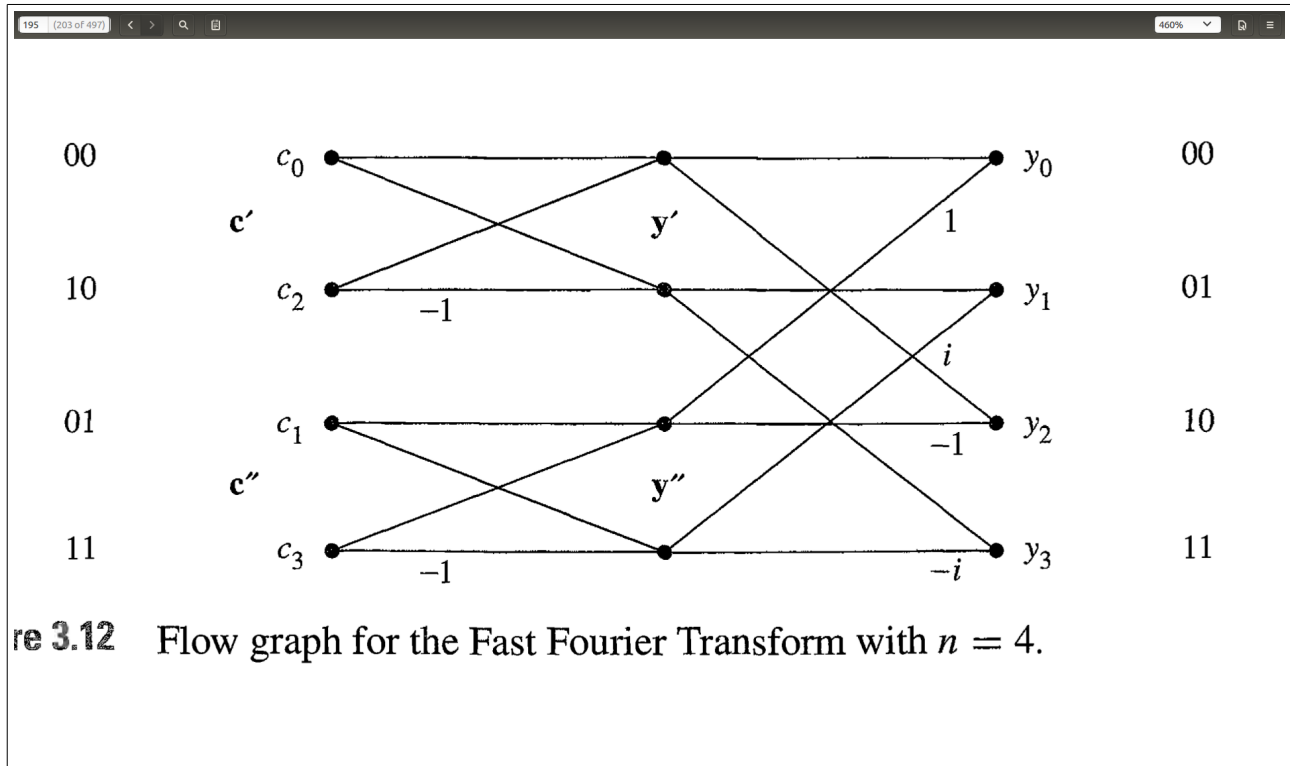


Figure 3.12 Flow graph for the Fast Fourier Transform with $n = 4$.

Illustration 15: Flow graph for the Fast Fourier Transform with $n = 4$

To compute $Fc = y$ using the FFT follow these steps:

- Convert F_n into 2 F_m matrices where $m = n/2$
- Split c into odd and even vectors:
 - $c' = [c_0, c_2, \dots, c_{n-2}]$ and $c'' = [c_1, c_3, \dots, c_{n-1}]$
- Transform them by F_m into y' and y'' :
 - $y' = F_m c'$ and $y'' = F_m c''$
- Reconstruct y from:
 - $y_j = y'_j + w_n^j y''_j$ where $j = 0, \dots, m-1$
 - $y_{j+m} = y'_j - w_n^j y''_j$ where $j = 0, \dots, m-1$

The computation for $n = 4$ is shown graphically in Illustration 15 above. The first step of the FFT changes multiplication by F_n to two multiplications of F_m . The even numbered components (c_0, c_2) are transformed separately from (c_1, c_3). For $n = 8$, the key idea is **to replace each F_4 box by two F_2 boxes**. The new factor $w_4 = i$ is the square of the old factor $w = w_8 = \exp(2\pi i/8)$. The flow graph shows the order that the c 's enter the FFT and the $\log_2 n$ stages that take them through it. Every stage needs $1/2 n$ multiplications so the final count is $\frac{1}{2} n \log n$.

Linear Transformations and Image Processing

A linear transformation is a mapping $V \rightarrow W$ between two vector spaces, possibly of a lower dimension, that preserves the operations of addition and scalar multiplication. Linear transformations are often represented by matrices, and simple examples include rotation and reflection. When $V = W$ the transformation is called a linear operator.

Filtering

Illustration 16 Shows the noise filtering of an image using filtering and the Fast Fourier Transform. The image goes through a linear transformation in the form of an FFT operation. Once in the frequency domain, the new basis, the image is filtered and then returned to the original basis via an inverse FFT operation.

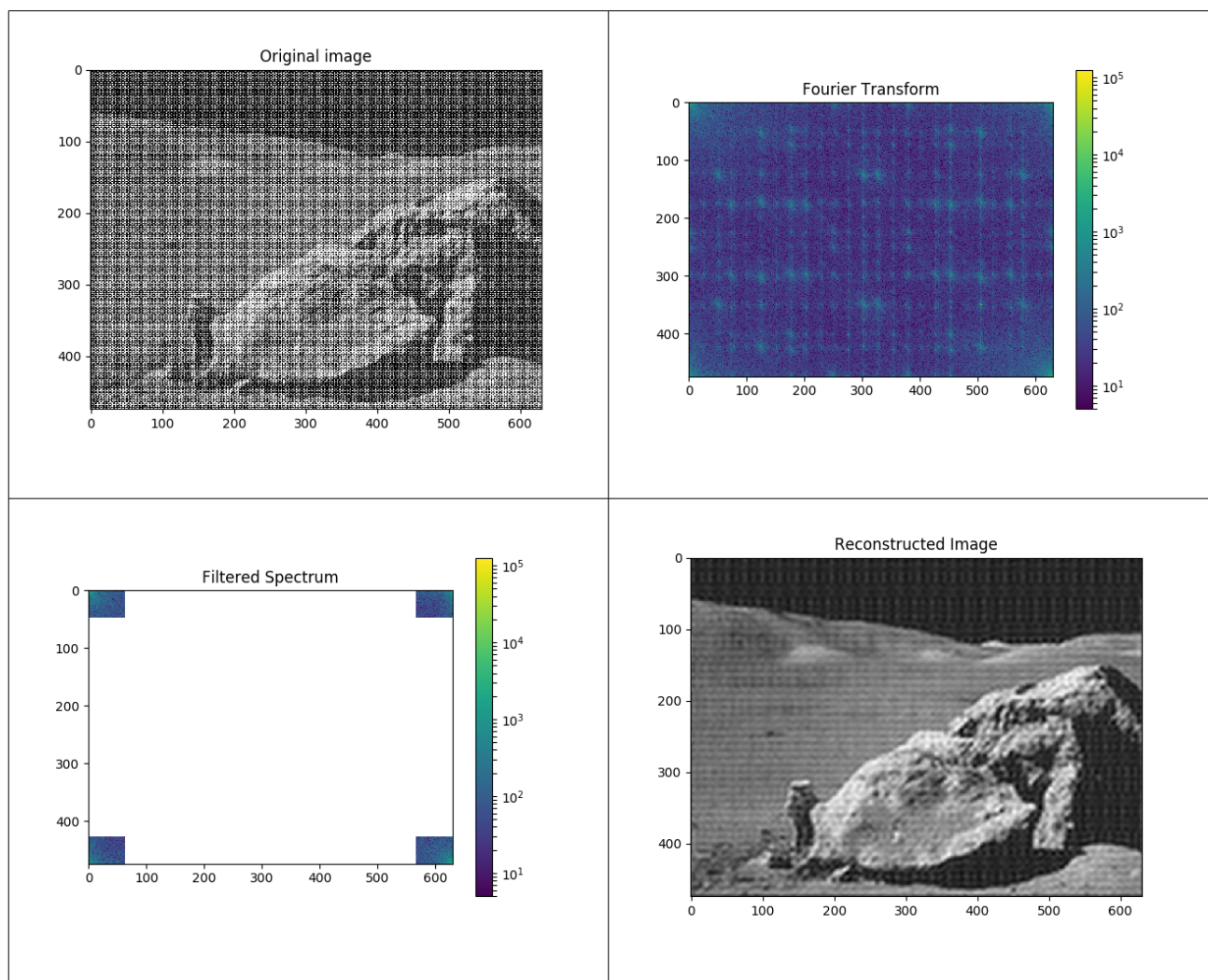


Illustration 16: Image Filtering using the FFT

It is worth noting that image filtering can be carried out directly via Gaussian filtering, which is mathematically equivalent to applying a Diffusion process, i.e. the solution to the heat equation, to the image. The solution of the heat equation illustrates a key and critical concept: the wavenumbers (spatial frequencies) decay according to a Gaussian function. Thus linear filtering with a Gaussian is equivalent to a linear diffusion of the image for periodic boundary conditions. The results of

Gaussian filtering is shown in Illustration 17 below. Note that there is no need to apply a linear transformation, like the FFT, to the image before filtering.

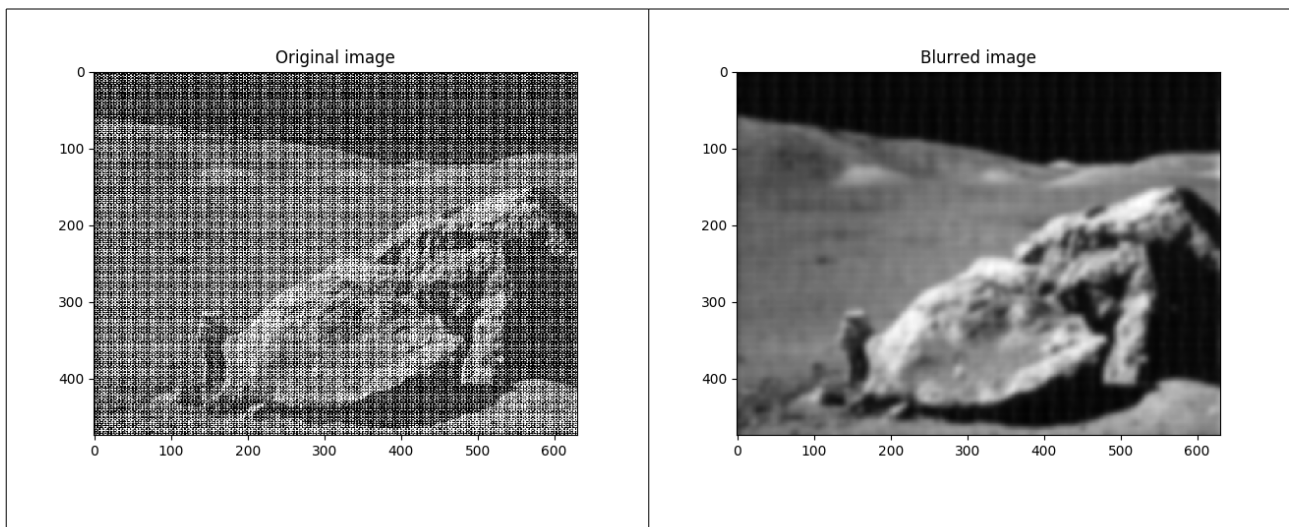


Illustration 17: Gaussian filtering of an image

Compression

As discussed in the section on SVD factorisation, a matrix can be decomposed into its ‘perfect’ bases coupled with its singular values. The results of the SVD for lossy compression are shown in Illustration 18. Finding the ideal bases, by using the SVD, for an image is computationally expensive which is an overhead making this technique inefficient for real world image compression applications, especially video.

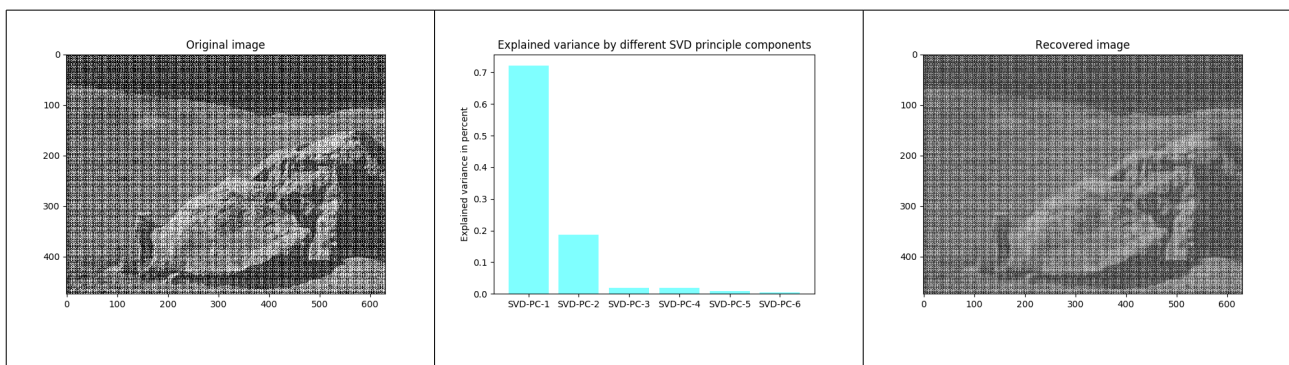


Illustration 18: Using the SVD to compress and recover an image

Image compression applications are usually based on one of the main image compression standards, two of which being the well known JPEG and JPEG-2000 standards. A key technique behind JPEG and JPEG-2000 is to provide an ideal basis for all images. In the case of JPEG, the ideal basis contains the Fourier modes. For JPEG-2000, the basis are wavelets, which have been designed to capture the dynamics of images better than the Fourier modes.

As a result it is best to stick to the tried and trusted compression JPEG standards for image storage and transmission rather than, however much fun it is, coding your own compression applications, unless, of course, you are the FBI or NASA.

We are only scratching the surface with image processing as this is a huge field that has revolutionised many industries, e.g. in medicine: ultrasound & MRIs; entertainment: computer graphics & digital photography, and for defence: radar and sonar imagery.

Linear Programming

Standard Form:

| | Primal | Dual |
|----------------------------|---------------------------------------|---------------------------------------|
| Objective Function: | $\underset{x}{\text{maximise}} C^T x$ | $\underset{y}{\text{minimise}} b^T y$ |
| Constraints: | subject to $Ax = b, x \geq 0$ | Subject to $A^T y = c, y \geq 0$ |

where C is the cost vector
 A is a m -by- n matrix
 b is a column vector with n components
 x is the optimal vector of unknowns

Three ways to approach the underlying mathematics of Linear Algebra:

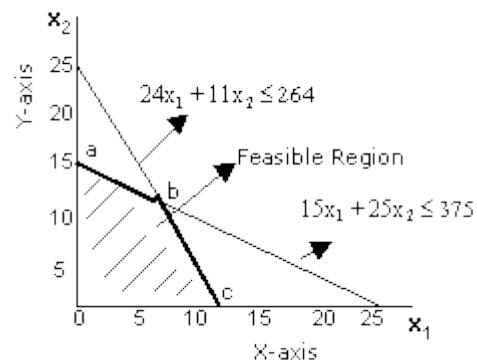
1) Geometry (intuitive)

IF an LP has a finite optimum value and the feasible region has at least one corner point

THEN there is a corner point optimal solution

key features of graph opposite:

- Feasible region
- Optimum value: A, B or C



2) Simplex method (computational) → Dantzig

At a high level the Simplex Algorithm is trying to find an optimal split of columns of A into Basic and non-Basis columns such that it gives a feasible solution and the solution has the highest possible objective value.

Phase I: finds one **Basic Feasible Solution (BFS)**:

- Not a trivial step
- Linear Program (LP) may not be feasible and so no BFS
- The corners of the feasible set are the basic feasible solutions of $Ax = b$
- A solution is basic when n of $m + n$ components are zero and
- it is feasible when it satisfies $x \geq 0$

$$z = \underset{x}{\text{maximise}} C_B^T x_B + C_N^T x_N$$

$$\text{subject to } Bx_B + Nx_N = b \text{ and } x_B, x_N \geq 0$$

where $A = [B \ N]$

B is the Basis and has m columns. $\text{Rank}(A) = m \rightarrow$ linearly independent columns!!!

N is a matrix of non-basis $(n - m)$ columns

x_B are basis variables

x_N are non-basis variables

Basic Feasible Solution (BFS):

$$x_B = B^{-1} b, x_N = 0$$

Phase II: search by moving step by step to the optimal x^*

- To complete a step, the **tableau** is used to fit A , b , c into a large matrix
- In each iteration, move to a neighbouring BFS where the objective value improves
- Stop when objective value can no longer be improved

Initially, the tableau looks like:

$$\begin{aligned} z - C_B^T x_B - C_N^T x_N &= 0 \\ Bx_B + Nx_N &= b \end{aligned}$$

We can re-write the tableau as:

$$\begin{aligned} z - C_B^T x_B - C_N^T x_N &= 0 \\ Bx_B + B^{-1}Nx_N &= B^{-1}b \end{aligned}$$

Reformulation of the tableau gives:

$$\begin{aligned} z - (C_N^T - C_B^T B^{-1}N) x_N &= (C_B^T B^{-1} b) \rightarrow \text{current objective value} \\ x_B + B^{-1}Nx_N &= B^{-1}b \rightarrow \text{value of BFS} \end{aligned}$$

IF $C_N^T - C_B^T B^{-1}N \leq 0$

THEN the current **BFS** is optimal, i.e. $x^* = (x_B = B^{-1} b, x_N = 0)$

3) Duality (algebraically) \rightarrow von Neumann

There are two ideas fundamental to duality theory:

- The dual of a dual LP is the original primal LP
- Every feasible solution for a LP gives a bound on the optimal value of the objective function of its dual

Weak Duality: the objective function value of the dual at any feasible solution is always greater than or equal to the objective function value of the primal at any feasible solution

Strong Duality: if the primal has an optimal solution, x^* , then the dual also has an optimal solution, y^* , and $C^T x^* = b^T y^*$

Applications

Electronic Circuits

Signal Processing

Differential Equations

Data Science / Machine Learning

Quantum Mechanics

Economics

Optimisation / Linear Programming

Image Processing

Computer Vision / Graphics

Neural Networks

Robotics

Cryptography

Logistics

References

Papers / Books

Although not directly referenced in the paper, the following texts were used extensively:

Graduate:

Oppenheim, A. and Schaffer, W. (2013). Discrete-Time Signal Processing

Kutz, N. (2013). Data-Driven Modeling & Scientific Computation: Complex Systems & Big Data

Bishop, M. (2006). Pattern Recognition and Machine Learning

Hastie, T., Tibshirani, R. and Friedman, J. (2008). The Elements of Statistical Learning

LeVeque, R. (2013). High Performance Scientific Computing

Barba, L. A. (2014). Practical Numerical Methods with Python

LeCun, Y., Bottou, L., Bengio, Y. and Haffner. (1998). Gradient-Based Learning Applied to Document Recognition

Turek, D. (2004) Design of Efficient Digital Interpolation Filters for Integer Upsampling

Jha, R.G. (2012). On the Numerical Simulations of Feynman's Path Integrals using Markov Chain Monte-Carlo with Metropolis-Hastings Algorithms

Undergraduate:

Calculus:

Fowler, J. and Snapp, B. (2014). MOOCulus

Bonfert-Taylor, P. (2015). Complex Analysis

Blanchard, P. and Devaney, R. (2011). Differential Equations

Linear Algebra:

Klein, P. (2013). Coding the Matrix: Linear Algebra through Applications to Computer Science

Strang, G. (2006). Linear Algebra and Its Applications, 4th Addition

Probability & Statistics:

Conway, A. (2012). Statistics One

Johns Hopkins Specialisation (2014). Data Science

Physics:

Krauth, W. (2006). Statistical Mechanics: Algorithms and Computation

Aspuru-Guzik, A. (2017). The Quantum World

Economics & Financial Mathematics:

Zivot, E. (2016). Computational Finance and Financial Econometrics

Cvitanic, J. and Zapatero, F. (2004). Intro to the Economics and Mathematics of Financial Markets

Rangel, A. (2016). Principles of Economics with Calculus

Electronics:

Prandoni, P. and Vetterli, M. (2008). Signal Processing for Communications

Agarwa, A. and Lang, J. (2005). Foundations and Digital Electronic Circuits

Computation and Algorithms:

Guttag, J. (2013). Introduction to Computation and Programming Using Python

Sedgewick, R. and Wayne, K. (2011). Algorithms

Additional:

Wikipedia (multiple links)

Linear Algebra (scipy.linalg) <https://docs.scipy.org/doc/numpy/reference/routines.linalg.html>

Signal Processing (scipy.signal) <https://docs.scipy.org/doc/scipy/reference/signal.html>

Machine Learning: https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_iris.html

Data Analysis: <https://pandas.pydata.org/>

Graphs / Networks: <https://networkx.github.io/>

Simulated Annealing: <https://www.fourmilab.ch/documents/travelling/anneal/>

Codes

The following codes were used to generate the various plots in the paper.

01_FundamentalEquation_Kirchoff_SYMMETRIC.py

02a_POSITIVE_DEFINITE_tests.py

02b_POSITIVE_DEFINITE_plots.py

03a_Least_Squares_Projection_Error_SYMMETRIC_POSITIVE_DEFINITE.py

03b_Weighted_Least_Squares_Projection_Error_SYMMETRIC_POSITIVE_DEFINITE.py

03c_Weighted_Least_Squares_SVD_DIMREDUCTION_DIAGONALISATION.py

04a_Powers_Fibonacci_Diagonalisation_EIGEN_SPECTRAL_THEOREM.py

04b_ODE_Diagonalisation_EIGEN_SPECTRAL_THEOREM.py

04c_Iris_PCA_SVD_DIAGONALISATION.py

05a_PDE_HeatEquation_TaylorSeries_TRIDIAGONAL_POSITIVE_DEFINITE.py

05b_PDE_2DLaplace_Jacobi_ITERATIVE_METHOD.py

06a_FourierSeries_SquareWave.py

06b_FourierSeries_SawToothWave.py

06c_FourierSeries_TriangleWave.py

07a_Fourier_Transform_Complex_Roots_of_Unity.py

07b_Fast_Fourier_Transform_UNITARY.py

08a_image_filtering_FFT_LINEAR_TRANSFORMATION.py

08b_image_filtering_GAUSSIAN_FILTERING.py

08c_image_compression_SVD.py

Appendix I: Properties of Eigenvalues & Eigenvectors

| | | |
|--|--|--|
| Symmetric: $A^T = A$ | Real λ 's | Orthogonal $x_i^T x_j = 0$ |
| Orthogonal: $Q^T = Q^{-1}$ | All $ \lambda = 1$ | Orthogonal $\bar{x}_i^T x_j = 0$ |
| Skew-symmetric: $A^T = -A$ | Imaginary λ 's | Orthogonal $\bar{x}_i^T x_j = 0$ |
| Complex Hermitian: $\bar{A}^T = A$ | Real λ 's | Orthogonal $\bar{x}_i^T x_j = 0$ |
| Positive Definite: $x^T A x > 0$ | All $\lambda > 0$ | Orthogonal |
| Similar matrix: $B = M^{-1} A M$ | $\lambda(B) = \lambda(A)$ | $x(B) = M^{-1} x(A)$ |
| Projection: $P = P^2 = P^T$ | $\lambda = 1; 0$ | Column space; nullspace |
| Reflection: $I - 2uu^T$ | $\lambda = -1; 1, \dots, 1$ | $u; u^{\text{perpendicular}}$ |
| Rank-1 matrix: uv^T | $\lambda = v^T u; 0, \dots, 0$ | $u; v^{\text{perpendicular}}$ |
| Inverse: A^{-1} | $1 / \lambda(A)$ | Eigenvectors of A |
| Shift: $A + cI$ | $\lambda(A) + c$ | Eigenvectors of A |
| Stable powers: $A^n \rightarrow 0$ | All $ \lambda < 1$ | |
| Stable exponential: $e^{At} \rightarrow 0$ | All $\text{Re } \lambda < 0$ | |
| Markov: $m_{ij} > 0, \sum_{i=1}^n m_{ij} = 1$ | $\lambda_{\max} = 1$ | Steady state $x > 0$ |
| Cyclic permutation: $p^n = I$ | $\lambda_k = e^{2\pi i k/n}$ | $X_k = (1, \lambda_k, \dots, \lambda_k^{n-1})$ |
| Diagonalisable: $S \Lambda S^{-1}$ | Diagonal of Λ | Columns of S are independent |
| Symmetric: $Q \Lambda Q^T$ | Diagonal of Λ (real) | Columns of Q are orthonormal |
| Jordan: $J = M^{-1} A M$ | Diagonal of J | Each block gives 1 eigenvector |
| Every matrix: $A = U \Sigma V^T$ | $\text{rank}(A) = \text{rank}(\Sigma)$ | Eigenvectors of $A^T A, A A^T$ in V, U |

Appendix II: Real versus Complex Matrices

| | | |
|--|-------------------|--|
| \mathbf{R}^n (n Real components) | \Leftrightarrow | \mathbf{C}^n (n Complex components) |
| Length: $\ x\ ^2 = x_1^2 + \dots + x_n^2$ | \Leftrightarrow | Length: $\ x\ ^2 = x_1 ^2 + \dots + x_n ^2$ |
| Transpose: $A_{ij}^T = A_{ji}$ | \Leftrightarrow | Hermitian transpose: $A_{ij}^H = \bar{A}_{ji}$ |
| $(AB)^T = B^T A^T$ | \Leftrightarrow | $(AB)^H = B^H A^H$ |
| Inner product: $x^T y = x_1 y_1 + \dots + x_n y_n$ | \Leftrightarrow | $x^H y = \bar{x}_1 y_1 + \dots + \bar{x}_n y_n$ |
| $(Ax)^T = x^T (A^T y)$ | \Leftrightarrow | $(Ax)^H = x^H (A^H y)$ |
| Orthogonality: $x^T y = 0$ | \Leftrightarrow | Orthogonality: $x^H y = 0$ |
| Symmetric matrices: $A^T = A$ | \Leftrightarrow | Hermitian matrices: $A^H = A$ |
| $A = Q \Lambda Q^{-1} = Q \Lambda Q^T$ (real Λ) | \Leftrightarrow | $A = U \Lambda U^{-1} = U \Lambda U^H$ (real Λ) |
| Skew-symmetric: $K^T = -K$ | \Leftrightarrow | Skew-Hermitian: $K^H = -K$ |
| Orthogonal: $Q^T Q = I$ or $Q^T = Q^{-1}$ | \Leftrightarrow | Unitary: $U^H U = I$ or $U^H = U^{-1}$ |
| $(Qx)^T (Qy) = x^T y$ and $\ Qx\ = \ x\ $ | \Leftrightarrow | $(Ux)^H (Uy) = x^H y$ and $\ Ux\ = \ x\ $ |

Appendix III: Jacobian and Hessian Matrices

The Jacobian

(Matrix of 1st Order Partial Derivatives of a **Vector**-valued function)

Suppose $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a function which takes as input the vector $x \in \mathbb{R}^n$ and produces, as output, the vector $f(x) \in \mathbb{R}^m$ then the Jacobian Matrix, J , of f is an $m \times n$ matrix:

$$J = \left[\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad \text{Component wise:} \quad J_{ij} = \frac{\partial f_i}{\partial x_j}$$

- If the function is differentiable at point X
- Then the Jacobian Matrix defines a linear map, $\mathbb{R}^n \rightarrow \mathbb{R}^m$, which is the best point wise linear approximation of the function, f , near the point X . This linear map is thus the generalisation of the usual notion of a derivative and is called the **Derivative** or the **Differential of f at X** .
- if $m = n$ the Jacobian Matrix is a square matrix and its Determinant is the **Jacobian Determinant of f** . It carries important information about the local behaviour of f .
- f in the neighbourhood of point X has an Inverse Function that is differentiable iff the **Jacobian Determinant** is non-zero at X .

Example: Consider the function, $f: \mathbb{R}^2 \rightarrow \mathbb{R}^2$, given by:

$$f(x, y) = \begin{bmatrix} x^2 y \\ 5x + \sin(y) \end{bmatrix}$$

Then we have

$$f_1(x, y) = x^2 y$$

and

$$f_2(x, y) = 5x + \sin(y)$$

And the Jacobian Matrix, J , of f is

$$J_f(x, y) = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \dots & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \dots & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2xy & \dots & x^2 \\ 5 & \dots & \cos(y) \end{bmatrix}$$

And the Jacobian Determinant is

$$\det(J_f(x, y)) = 2xy \cos(y) 5x^2$$

The Hessian

(Square Matrix of 2nd Order Partial Derivatives of a **Scalar**-valued function)

Describes the Local Curvature (remember it is a 2nd order derivative) of a function of many variables.

Suppose $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a function taking as input a vector, $x \in \mathbb{R}^n$, and outputting a scalar, $f(x) \in \mathbb{R}$; if all 2nd Partial Derivatives of f exist and are continuous over the domain of the function, then the Hessian Matrix, H , of f , H_f , is a square $n \times n$ matrix, usually defined and arranged as follows:

$$H_f(X) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

or component wise

$$H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

The Hessian Matrix is related to the Jacobian Matrix by:

$$H_f(X) = J(\nabla f(X))^T$$

Note: H_f is a matrix with functions as entries and it is meant to be evaluated at some point, X , this is sometimes called a Matrix-valued Function.

Bordered Hessian

Used for the 2nd derivative test in certain Constrained Optimisation problems. Given a function, f , but adding a constraint function, g , such as $g(x) = c$, the bordered Hessian of the Lagrange Function,

$L(x, \lambda) = f(x) + \lambda[g(x) - c]$ is:

$$H(L) = \begin{bmatrix} \frac{\partial^2 \lambda}{\partial \lambda^2} & \frac{\partial^2 \lambda}{\partial \lambda \partial x} \\ \left(\frac{\partial^2 \lambda}{\partial \lambda \partial x}\right)^T & \frac{\partial^2 \lambda}{\partial x^2} \end{bmatrix} = \begin{bmatrix} 0 & \frac{\partial g}{\partial X} \\ \left(\frac{\partial g}{\partial X}\right)^T & \frac{\partial^2 L}{\partial X^2} \end{bmatrix}$$

$$= \begin{bmatrix} 0 & \frac{\partial g}{\partial x_1} & \frac{\partial g}{\partial x_2} & \dots & \frac{\partial g}{\partial x_n} \\ \frac{\partial g}{\partial x_1} & \frac{\partial^2 L}{\partial x_1^2} & \frac{\partial^2 L}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 L}{\partial x_1 \partial x_n} \\ \frac{\partial g}{\partial x_2} & \frac{\partial^2 L}{\partial x_2 \partial x_1} & \frac{\partial^2 L}{\partial x_2^2} & \dots & \frac{\partial^2 L}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g}{\partial x_n} & \frac{\partial^2 L}{\partial x_n \partial x_1} & \frac{\partial^2 L}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 L}{\partial x_n^2} \end{bmatrix}$$

- If there are m Constraints Then the Zero in the upper-left corner is an $m * m$ block of zeros
- 2nd derivative test consists here of sign restrictions of the determinants of a certain set of $n-m$ submatrices of the bordered Hessian
 - Normal 2nd order derivative test rules cannot apply here since a bordered Hessian can neither be -ive / +ive definite

Application: Quadratic Approximations of Multi-Variate Functions

Hessian Matrices are used in large scale Optimisation problems within Newton type methods because they are the Coefficient of the Quadratic Term of a local Taylor Expansion of a function, that is:

$$y = f(x + \Delta x) \approx f(x) + \nabla f(x)^T \Delta x + \frac{1}{2!} \Delta x^T H(x) \Delta x$$

where ∇f is the gradient $\left(\frac{\partial f}{\partial x_1} \dots \frac{\partial f}{\partial x_N}\right)$

Application: Image Processing

The Hessian Matrix is commonly used for expressing Image Processing Operators in field of Computer Vision.

Appendix IV: Scientists, Artists & Thinkers

Classical Antiquity (~1000 BC – 476 AD)

Homer (~750 BC). The Iliad, The Odyssey

Hesiod (~750 BC). Theogony

Solon of Athens (638 – 588 BC): Seven Sages

Thales (624 – 548 BC): Seven Sages

Pythagoras (570 - 495 BC)

Buddha (563 – 483 BC) – for reference

Confucius (551 – 479 BC) – for reference

Aeschylus (525 – 456 BC). The Oresteia

Sophocles (497 – 405 BC). Oedipus Rex

Socrates (470 -399 BC)

Plato (428 – 348 BC)

Aristotle (367 - 347 BC)

Euclid (c. 350 – 250 BC)

Epicurus (341 – 270 BC)

Archimedes (287 – 212 BC)

Jesus and his disciples (0 – 100 AD)

Ptolemy (100 – 170)

St. Augustine of Hippo (354 – 430)

Middle Ages (476 - 1453)

Muhammad (570 - 632)

Muḥammad ibn Mūsā al-Khwārizmī (780-850)

Anselm of Canterbury (1033 - 1109)

Maimonides (1135 - 1204)

Fibonacci, L. (1170 – 1250)

Francis of Assisi (1182 – 1226)

Thomas Aquinas (1225 – 1274)

William of Ockham (1285 - 1347)

Early Modern History – Renaissance & Reformation (1453 - 17th Century)

Leonardo da Vinci (1452 – 1519)

Erasmus (1466 - 1536)

Machiavelli, N. (1469 – 1527)

Michelangelo (1475 - 1564)

Copernicus, N. (1473 – 1543). *De revolutionibus orbium coelestium* (1532)

Raphael (1483 - 1520)

Luther, M. (1483 – 1546)

Zwingli, H. (1484 – 1531)

Ignatius of Loyola (1491 - 1556)

Calvin, J. (1509 – 1564)

Knox, J. (1513 - 1572)

Late Modern History - Enlightenment & Revolution (~17th – mid 20th Century)

Montaigne (1533 - 1592)

Galilei, G. (1564 – 1642)

Caravaggio (1571 - 1610)

Hobbes, T. (1588 – 1674)

Descartes, R. (1596 - 1650). *Principles of Philosophy* (1644)

Velasquez (1599 – 1660)

Rembrandt (1606 - 1669)

Spinoza, B. (1632 - 1677). *Ethics*

Locke, J. (1632 – 1704). *Liberalism*

Newton, I. (1643 - 1727). *Philosophiae Naturalis Principia Mathematica* (1687)

Leibniz, G. (1646 - 1716). *Nova Methodus pro Maximis et Minimis* (1684)

Bernoulli, J. (1667 – 1748). *Commercium philosophicum et mathematicum* (1745)

De Moivre, A. (1667 - 1754)

Taylor, B. (1685 – 1731)

Montesquieu (1689 – 1755). *The Spirit of the Laws* (1748)

Voltaire (1694 - 1778)

Bayes, T. (1701 – 1761)

Wesley, J. (1703 - 1791)

Buffon, Comte de. (1707 - 1788)

Euler, L. (1707 - 1783). Institutiones Calculi Differentialis (1748)

Hume, D. (1711 – 1776)

Rousseau, J.J. (1712 – 1778)

Diderot, D. (1713 - 1784)

Smith, A. (1723 - 1790)

Kant, I. (1724 – 1804)

Lagrange, J. (1736 - 1813). Mécanique Analytique (1788)

Laplace, P.S. (1749 – 1827)

Fourier, J. (1768 - 1830). Théorie analytique de la chaleur (1822)

Hegel, G. (1770 - 1831)

Brown, R. (1773 - 1858)

Gauss, F. (1777 - 1855). Disquisitiones Arithmeticae (1801)

Ohm, G. (1789 – 1854)

Faraday, M. (1791 - 1867)

Babbage, C. (1791 - 1871)

Jacobi, C. (1804 -1851)

Dirichlet, P. (1805 - 1859)

Hamilton, W. (1805 - 1865). Second Essay on a General Method in Dynamics (1835)

Darwin, C. (1809 – 1882). On the Origin of Species (1859)

Hesse, O. (1811 – 1874)

Kierkegaard, S. (1813 – 1855)

Wagner, R. (1813 – 1883): various operas including Tristan and Isolde (1857 - 1864)

Lovelace, A. (1815 – 1852). Sketch of the Analytical Engine (1843)

Boole, G. (1815 – 1864)

Marx, K. (1818 - 1883)

Kirchoff, G. (1824 – 1887)

Reimann, B. (1826 - 1866)

Maxwell, J. (1831 – 1879)

Neitzsche, F. (1844 – 1900): The Birth of Tragedy (1886)

Boltzman, L.(1844 – 1906)

Freud, S. (1856 - 1939)

Runge, C. (1856 – 1927)

Planck, M. (1858 – 1947)

Hilbert, D. (1862 – 1943)

Curie. M. (1867 - 1934)

Kutta, M. (1867 – 1944)

Rutherford, E. (1871 – 1937)

Mondrain, P. (1872 - 1944)

Russell, B. (1872 – 1970)

Schoenberg, A. (1874 - 1951)

Einstein, A. (1879 – 1955). The Foundation of the General Theory of Relativity (1916)

Eddington, A. (1882 – 1944)

Stravinsky, I. (1882 - 1971)

Bohr, N. (1885 – 1962)

Schrödinger, E.(1887 – 1961)

Friedman, A. (1888 - 1925)

Hubble, E. (1889 – 1953)

Heidegger, M. (1889 - 1976)

Weiner, N. (1894 – 1964)

Lemaitre, G. (1894 – 1966)

Heisenberg, W. (1901 – 1976)

Contemporary History (mid 20th Century - present)

Popper, K. (1902 – 1994)

de Kooning, W. (1904 - 1997)

von Neumann, J. (1903 – 1957)

Rothko, M. (1903 - 1970)

Satre, J.P. (1905 - 1980)

Turing, A. (1912 – 1954)

Pollock, J. (1912 - 1956)

Camus, A. (1913 - 1960)

Dantzig, G. (1914 – 2005). Simplex algorithm

Tukey, J.(1915 - 2000). FFT Algorithm (1965)

Ito, K. (1915 - 2008)

Feynman, R.(1918 – 1988)

Bernstein, L. (1918 - 1990)

Codd, E. (1923 – 2003). Relational Model for Databases

Backus, J. (1924 – 2007). Fortran

Cooley, J. (1926 – 2016). FFT Algorithm (1965)

Markowitz, H. (1927 – present)

Chomsky, N. (1928 – present)

Moore, G. (1929 – present). Moore’s law

Dijkstra, E. (1930 – 2002). Dijkstra’s Algorithm for shortest path in a graph

Penzias, A. (1933 – present): CMB

Hoare, T. (1934 – present). Quicksort

Wirth, N. (1934 – present). Pascal & Modula

Karp, R. (1935 – present). NP-completeness

Wilson, R. (1936 – present): CMB

Pearl, J. (1936 – present). Bayesian Networks

Glass, P. (1937 - present)

Black, F. (1938 – 1995). The Pricing of Options & Corporate Liabilities (1973)

Knuth, D. (1938 – present). The Art of Programming

Kahn, R. (1938 – present). TCP/IP

Liskov, B. (1939 – present). Substitution Principle

Richie, D. (1941 – 2011). C

Hawking, S. (1942 – 2018). A brief history of time

Thompson, K. (1943 – present). Unix

Cerf, V. (1943 – present). TCP/IP

Stonebraker, M (1943 – present). Relational Databases

Scholes, M. (1941 - present). The Pricing of Options & Corporate Liabilities (1973)

Diffie, B. (1943 – present). Public-key cryptography

Hellman, M. (1945 – present). Public-key cryptography

Adelman, L. (1945 – present). RSA algorithm (public-key)

Schama, S. (1945 – present). Historian

Rivest, R. (1947 – present). RSA algorithm (public-key)

Patterson, D. (1947 – present) RISC architecture

Hinton, J. (1947 – present). Deep Learning (Neural Networks)

Guth, A. (1948 – present). Cosmological Inflation

Holtzman, G. (1951 – present). Promela and SPIN model checker

Shamir, A. (1952 – present). RSA algorithm (public-key)

Hennessey, J. (1952 – present). RISC architecture

Jobs, S. (1955 – 2011). Founder of Apple

Gates, W. (1955 – present). Founder of Microsoft

Berners-Lee, T. (1955 – present). Word Wide Web (WWW)

LeCun, Y. (1960 – present). Deep Learning (Neural Networks)

Edelman, A. (1963 – present). Inventor of Julia and mates with Gil Strang ;-)

Bengio, Y (1964 – present). Deep Learning (Neural Networks)