SIF3012 Computational Physics

Handwritten notes by Juan Carlos Algaba

Topics:

- Initial Value Problems
- Boundary Value Problems
- Matrices
- Fourier Analysis
- Wavelet Analysis

INITIAL VALUE PROBLEM

The behaviour of a dynamical system can be described as a set of 1st order differential equations:

$$\frac{d\vec{y}}{dt} = \vec{g}(\vec{y},t)$$

where $\vec{y} = (y_1, y_2, y_3 \ldots)$ is the dynamical variable vector and $\vec{g}(\vec{y},t) = [g_1(\vec{y},t), \vec{g}_2(\vec{y},t) \ldots]$ is the generalized velocity vector.
In principle, we can give a solution of the equation if the initial condition $y(t=0) = y_0$ is given and the solution exists.
In general, most higher order differential equations can be transformed into a set of coupled first order differential equations.

Example:

Consider a particle moving in one dimension due to an elastic force that is governed by Newton's equation $f = m\vec{a}$. Then

$$y_1 = x$$
$$y_2 = v = \frac{dx}{dt}$$
$$g_1 = v = y_2 \qquad \rightarrow \qquad \frac{dy_1}{dt} = y_2$$
$$g_2 = f/m = -kx/m = -ky_1/m \qquad \frac{dy_2}{dt} = -\frac{k}{m} y_1$$

If the initial position $y_1(0)$ and the initial velocity $y_2(0) = v(0)$ are given, this is an initial value problem and can be solved.

Euler Method

We can try to numerically solve the differential equation step-by-step since we know the initial value and can infer the value in the following step in the iteration. The derivative can be approximated by

$$\frac{dy}{dt} \simeq \frac{y_{i+1} - y_i}{t_{i+1} - t_i} \simeq g(y_i, t_i)$$

If we rearrange terms and use $g_i = g(y_i, t_i)$ to simplify the notation we obtain the simplest algoritm for initial value problems:

$$y_{i+1} = y_i + \tau g_i + \mathcal{O}(\tau^2)$$

where $\tau = t_{i+1} - t_i$ is a fixed time step. We can reach the same result by considering a Taylor approximation of y_{i+1} at t_i and keeping the terms up to first order.
At the end of the calculation, the error accumulated is of the order of $n\mathcal{O}(\tau^2) \simeq \mathcal{O}(\tau)$, meaning that the accuracy of this algorithm is relatively low.

Picard Method

We can rewrite the first order differential equation $\frac{dy}{dt} = g(y,t)$ in integral form:

$$y_{i+j} = y_i + \int_{t_i}^{t_{i+j}} g(y,t)\,dt$$

which is the solution for the differential form. Since the integral may not be analytic, we have to approximate it.

In the most simple case, $j=1$ and we approximate $g(y,t) \simeq g_i$ in the integral, we recover the Euler algorithm.

Alternatively, we can choose $j=1$ and use the trapezoid rule for the integral to obtain

$$y_{i+1} = y_i + \frac{\tau}{2}(g_i + g_{i+1}) + O(\tau^3)$$

Note that, in the right hand side of the equation we have $g_{i+1} = = g(y_{i+1}, t_{i+1})$ which contains y_{i+1}, which can be provided adaptively. For example, we can take the solution of the previous step as a first guess for the current time $y^{(0)}_{i+1} = y_i$ and then iterate the solution in the equation; e.g. $y^{(k+1)}_{i+1} = y_i + \frac{\tau}{2}(g_i + g^{(k)}_{i+1})$.

Alternatively, we can apply a less acurate algorithm to predict the next value y_{i+1} first (for example, using the Euler algorithm) and then apply a better algoritm to improve the next value.

This kind of methodoloy where we first predict the next value and then iterate with a different algorithm to get a better approximation is called the predictor - corrector method.

Another method to improve the Picard algorithm is to increase the number of mesh points in the integral form. For example, if we take $j=2$, the integral will be

$$y_{i+2} = y_i + \int_{t_i}^{t_{i+2}} g(y,t)$$

If now we use the linear interpolation scheme to approximate $g(y,t)$, we obtain

$$g(y,t) = \frac{t-t_i}{\tau} g_{i+1} - \frac{t-t_{i+1}}{\tau} g_i + O(\tau^2)$$

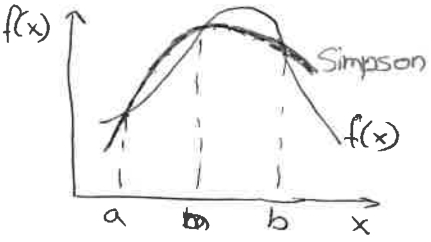
Substituting in the integral we obtain

$$y_{i+2} = y_i + 2\tau g_{i+1} + O(\tau^3)$$

which has an accuracy one order higher than the Euler algorithm. However we need the values of the first two points to start this algorithm.

Of course, we can always include more points in the integral (i,e, we can search for $j=3,4,\dots$) to obtain algorithms with apparently higher accuracy, but we will need the values of more points in order to start the algorithm. In practice, it may happen that the errors accumulated from the approximation of the first few points will eliminate the apparently high accuracy of the algorithm.

Alternatively, we can make the accuracy higher by using a better quadrature. For example, we can take $j=2$ and apply Simpson's rule.



In Simpson's rule, we approximate each step by a parabola that goes through the points a, b, and an intermediate m. Then the approximation to any integral will be

$$\int_a^b f(x)\,dx = \left[\frac{b-a}{6} \quad f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] =$$

$$= \frac{1}{3}h\left[f(a)+4f(a+h)+b(B)\right]$$

Applying this to our integral for $j=2$; ie $y_{i+2}=y_i+\int_{t_i}^{t_{i+2}} g(y,t)\,dt$, we will have:

$$y_{i+2} = y_i + \frac{\tau}{3}\left(g_{i+2}+4g_{i+1}+g_i\right) + \mathcal{O}(\tau^5)$$

This implicit algorithm can be used as a corrector if our previous algorithm $y_{i+2}=y_i+2\tau g_{i+1}$ is used as the predictor.

The Runge-Kutta Method

The accuracy of the methods discussed so far can be improved only by including more points, which is not always practical. We can improve the calculations by the Runge-Kutta method, which is based on two different ways of seeing the Taylor series approximation.

Formally, we can write $y(t+\tau)$ as:

$$y(t+\tau) = y + \tau y' + \frac{\tau^2}{2} y'' + \frac{\tau^3}{3!} y''' + \dots$$

$$= y + \tau g + \frac{\tau^2}{2}(g_t + g g_y) + \frac{\tau^3}{6}(g_{tt} + 2 g g_{ty} + g^2 g_{yy} + g g_y^2 + g_t g_y) + \dots$$

where the subscript denote partial derivatives: for example, $g_{yt} = \partial^2 g / \partial y \partial t$

We can also write the expansion of $y(t+\tau)$ as

$$y(t+\tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2 + \alpha_3 c_3 + \dots + \alpha_m c_m$$

Comparing the α parameters, we see that

$$c_1 = \tau g(y,t)$$

$$c_2 = \tau g(y + \nu_{21} c_1, t + \nu_{21} \tau)$$

$$c_3 = \tau g(y + \nu_{31} c_1 + \nu_{32} c_2, t + \nu_{31}, \tau + \nu_{32} \tau)$$

$$\vdots$$

$$c_m = \tau g\left(y + \sum_{i=1}^{m-1} \nu_{mi} c_i, t + \sum_{i=1}^{m-1} \nu_{mi}\right)$$

Where α_i and ν_{ij} are parameters to be determined. A set of equations for α_i and ν_{ij} is obtained by keeping the coefficients of the terms with the same power on τ on both sides equal. By truncating the expansion to the term $O(\tau^m)$ we obtain m equations and $m + m(m-1)/2$ parameters (α_i, ν_{ij}) to be determined, thus leaving some room to choose these parameters.

Example:

R-K truncated to $O(\tau^2)$:

The Taylor expansion will become:

$$y(t+\tau) = y + \tau g + \frac{\tau^2}{2}(g_i + gg_y)$$

The expansion of second order for the coefficients will be:

$$y(t+\tau) = y(t) + \alpha_1 c_1 + \alpha_2 c_2$$

$$c_1 = \tau g(y,t)$$

$$c_2 = \tau g(y + \nu_{21} c_1 \, , \, t + \nu_{21}\tau)$$

Now, if we perform the Taylor expansion for c_2 up to $O(\tau^2)$,

$$c_2 = \tau g + \nu_{21}\tau^2(g_t + gg_y)$$

Substituting c_1 and the expansion of c_2 in the expression above:

$$y(t+\tau) = y(t) + (\alpha_1+\alpha_2)\tau g + \alpha_2\tau^2(g_t + gg_y)\nu_{21}$$

If we compare this expression with the original Taylor expansion and equal the terms with the same order in τ, we obtain:

$$\alpha_1 + \alpha_2 = 1$$

$$\alpha_2\nu_{21} = 1/2$$

As discussed, we have $m=2$ equations and $m+m(m-1)/2=3$ parameters to be derived, thus we do not have a unique solution. We can use this as flexibility to make our problem easier, to increase the numerical acuracy or to adjust to the particular problem.

For this case, we can choose, for example

$$\alpha_1 = \alpha_2 = \frac{1}{2}, \quad \nu_{21} = 1 \quad \text{or}$$

$$\alpha_1 = \frac{1}{3}, \quad \alpha_2 = \frac{2}{3}, \quad \nu_{21} = 3/4$$

Example:

R-K truncated to $O(\tau^4)$ is the most widely used case. The algorithm is then given by:

$$y(t+\tau) = y(t) + \frac{1}{6}(c_1 + 2c_2 + 2c_3 + c_4)$$

$$c_1 = \tau g(y,t)$$

$$c_2 = \tau g\left(y + \frac{c_1}{2}, t + \frac{\tau}{2}\right)$$

$$c_3 = \tau g\left(y + \frac{c_2}{2}, t + \frac{\tau}{2}\right)$$

$$c_4 = \tau g(y + c_3, t + \tau)$$

Boundary Value Problems

A class of problems in physics requires the solving of differential equations with the values of the physical quantities or their derivates given at the boundary of a given region

The Shooting Method

In general, problems in physics are usually given as a second-order differential equation $y'' = f(y, y', x)$. We first convert this second order differential equation into two first-order differential equations by defining $y_1 = y$ and $y_2 = y'$, namely

$$\frac{dy_1}{dx} = y_2$$

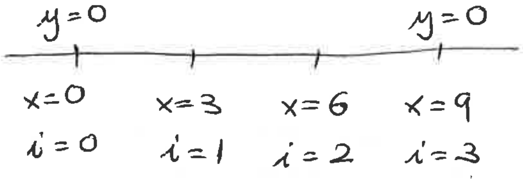
$$\frac{dy_2}{dx} = f(y_1, y_2, x)$$

The key here is to make the problem look like an initial value problem by introducing an adjustable parameter. The solution is then found for that parameter and then the parameter is adjusted until the solution found agrees with the boundary conditions within a given tolerance.

Example:

Let's consider the ODE $\frac{d^2y}{dx^2} - 2y = 8x(9-x)$ with boundary conditions $y(0) = 0$ and $y(9) = 0$.

For the initial guessing we will use the Euler method with step size $h = 3$.



We are given that $y(0) = 0$ for our initial condition. As our adjustable parameter, let's choose $y'(0) = 4$ as our initial guess for the other initial condition for our shooting method. Converting our 2nd order ODE into two 1st order ODES:

$$\frac{dy}{dx} = z = f_1(x, y, z) \qquad [y(0) = 0 \text{ given initial condition}]$$

$$\frac{dz}{dx} = 2y + 8x(9-x) = f_2(x, y, z) \qquad [z(0) = \frac{dy}{dx}(0) = 4 \text{ given by shooting initial guessing parameter}]$$

For the integration method assuming these initial conditions, we can use the Euler technique for simplicity:

$$y_{i+1} = y_i + f_1(x_i\ y_i\ z_i)h$$

$$z_{i+1} = z_i + f_2(x_i, y_i, z_i)h$$

We will use $h = 3$ from $i=0$ to 3: $\left\{ \begin{array}{l} i=0 \Rightarrow x=0 \\ i=1 \Rightarrow x=3 \\ i=2 \Rightarrow x=6 \\ i=3 \Rightarrow x=9 \end{array} \right.$

Step by step:

$\underline{i=0}$ $(x_0=0,\ y_0=0,\ z_0=4)$

$$y_1 = y_0 + f_1(x_0, y_0, z_0) \rightarrow y_1 = 0 + f_1(0,0,4)\times 3 = 0 + 4\times 3 = 12$$

$$z_1 = z_0 + f_2(x_0, y_0, z_0) \rightarrow z_1 = 0 + f_2(0,0,4)\times 3 = 4 + [2\times 0 + 8\times 0(9-0)] = 4$$

$\underline{i=1}$ $(x_1=3,\ y_1=12,\ z_1=4)$

$$y_2 = y_1 + f_1(x_1, y_1, z_1) \rightarrow y_2 = 12 + f_1(3,12,4) = 12 + 3\times 4 = 24$$

$$z_2 = z_1 + f_2(x_1, y_1, z_1) \rightarrow z_2 = 4 + [2\times 12 + 8\times 3\times(9-3)]\times 3 = 508$$

$\underline{i=2}$ $(x_2=6,\ y_2=24,\ z_2=508)$

$$y_3 = y_2 + f_1(x_2, y_2, z_2) \rightarrow y_3 = 24 + f_1(6,24,508)\times 3 = 1548$$

$$z_3 = z_2 + f_2(x_2, y_2, z_2) \rightarrow \dots$$

Therefore, we find that $y_3 \simeq y(x_3) = y(9) = 1548$. This means that using the initial guessing parameter $y'(0)=4$ we do not get the required $y(9)=0$ by the contour conditions. Therefore, we need to guess another $y'(0)$ [give it another shot] that leads us closer to the required value.

Let us choose now a different guess $y'(0)=-24$. Using the same steps as before, but putting the outcomes in table form:

i	x_i	y_i	z_i
0	0	0	-24
1	3	-72	-24
2	6	-144	-24
3	9	-216	

Therefore, with our initial guess $y'(0)=-24$ we obtain that $y_3 \simeq y(x_3) = y(9) = -256$. This is still not the required $y_3=0$ but since we have now two guesses, we can hopefully interpolate to obtain a more reasonable initial guess.

In the hopes that our outcome value from the initial guess follows a somewhat smooth relationship, we aim to interpolate using a straight line. Let's call p_i to our initial guesses $i=1,2$ and q_i our outputs. Then:

$y'(0)$	$y_3 \simeq y(9)$
$p_0 = 4$	$q_0 = 1548$
$p_1 = -24$	$q_1 = -216$

$$p = p_0 + \frac{p_1 - p_0}{q_1 - q_0}(q - q_0) \quad q_0 \leq q \leq q_1$$

$$p = 4 + \frac{-24-4}{-216-1548}(q - 1548)$$

Based on this interpolation, if we look for the desired value $q=0$, this happens for $p=-20.57$. We can use this new value of p as our newest initial guess. If we do so,

i	x	y	z
0	0	0	-20.57
1	3	-61.7	-20.57
2	6	-123.42	41.17
3	9	0.09	

Finally, our final estimate leads to $y_3 \simeq y(9) \sim 0.09$ which is very close to the desired $y(9)=0$. Depending on the accuracy that is required, we may stop here, or include more decimals in our calculations, or interpolate to a much better initial guess. If we are happy with this result, the values obtained for x, y, z at the various steps are the solution for the ODE we were looking for.

Finite Difference Method

In the finite difference method, the differential equation is solved by approximating the derivatives with finite differences. Both the spatial and time domains (if applicable) are discretized, or broken into a series of a finite number of intervals. In particular, if we have a function $u(r)$,

$$u(r_0+\Delta r) = u(r_0) + \left.\frac{du}{dr}\right|_{r_0} h + O(r^2) \rightarrow \left.\frac{du}{dr}\right|_{r_0} \simeq \frac{u(r_0+\Delta r) - u(r_0)}{\Delta r}$$

Considering the discrete steps, this transforms into

$$\left.\frac{du}{dr}\right|_i = \frac{u_{i+1} - u_i}{\Delta r}$$

Similarly:

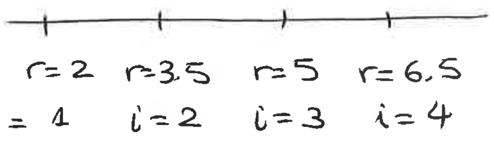
$$\left.\frac{d^2u}{dr^2}\right|_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta r)^2}$$

By substituting the derivatives of the differential equation by their finite differences counterpart at each node (step), we can obtain a system of equations that, once solved, provides us with the ODE solution.

Example:

We consider $\frac{d^2u}{dr^2} + \frac{1}{r}\frac{du}{dr} - \frac{u}{r^2} = 0$ with contour conditions $u(2) = 0.008$ and $u(6.5) = 0.003$.

We will use 4 nodes to solve the problem. Therefore $\Delta r = \frac{6.5-2}{3} = 1.5$



We can write the ODE in terms of the finite differences:

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta r^2} + \frac{1}{r_i}\frac{u_{i+1} - u_i}{\Delta r} - \frac{u_i}{r_i^2} = 0$$

Now we can substitute for each of the nodes $i=1,\dots 4$:

Node $i=1$:

$$u_1 = u(2) = 0.008 \quad \text{(Equation 1)}$$

Node $i=2$

$$\frac{u_3-2u_2+u_1}{(\Delta r)^2} + \frac{1}{r_2}\frac{u_3-u_2}{\Delta r} - \frac{u_2}{r_2^2} = 0 \Rightarrow \frac{u_3-2u_2+u_1}{(1.5)^2} + \frac{1}{3.5}\frac{u_3-u_2}{(1.5)^2} - \frac{u_2}{(3.5)^2} = 0$$

$$\Rightarrow 0.444\,u_1 - 1.1610\,u_2 + 0.63442\,u_3 = 0 \quad \text{(Equation 2)}$$

Node $i=3$

$$\frac{u_4-2u_3+u_2}{(\Delta r)^2} + \frac{1}{r_3}\frac{u_4-u_3}{\Delta r} - \frac{u_3}{r_3^2} = 0 \Rightarrow \frac{u_4-2u_3+u_2}{(1.5)^2} + \frac{1}{5}\frac{u_4-u_3}{1.5} - \frac{u_3}{5^2} = 0$$

$$\Rightarrow 0.444\,u_2 - 1.0622\,u_3 + 0.5778\,u_4 = 0 \quad \text{(Equation 3)}$$

Node $i=4$

$$u_4 = u(6.5) = 0.003 \quad \text{(Equation 4)}$$

With this, we have 4 equations with 4 unknowns. We can aim to solve this system of equations to obtain the values of u at each of the nodes. For example, writing the system of equations in matrix form, we have:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0.444 & -1.1610 & 0.63492 & 0 \\ 0 & 0.444 & -1.0622 & 0.5778 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} 0.008 \\ 0 \\ 0 \\ 0.003 \end{bmatrix}$$

We can solve this by e.g. backward substitution (see later), with the following results:

$u_1 = 0.008$

$u_2 = 0.005128$

$u_3 = 0.003778$

$u_4 = 0.003$

MATRICES

Many problems in physics and maths can be written in the form of a matrix, and the way of solving these problems involve operating with these matrices. Some examples include:

Example 1: Solution to a linear equation set

A system of equations with various incognitas can be written in matrix form and typically solved when the number of equations is equal to the number of unknowns by operating, in this case, the square matrix.

Example 2: Inverse of a matrix

The inverse of a matrix can be obtained as the solution for the linear equation given by $[A][x]=[B]$ with $b_{ij}=\delta_{ij}$ the elements of $[B]$. We simply need to take each column of $[x]$ as an independent problem: $A^{-1}_{ij}=x_{ij}$

Example 3: Zeroes of multivariable functions

Consider the equation $\vec{f}(\vec{x})=0$ with $\vec{f}=(f_1, f_2, \_ f_n)$ and $\vec{x}=(x_1 .. x_n)$

If the equation has at least one zero, then we have a solution $\vec{x}_r$ and we can perform a Taylor expansion around its neighbourhood:

$\vec{f}(\vec{x}_r)=\vec{f}(\vec{x})+\Delta\vec{x}\ \vec{\nabla}\vec{f}(\vec{x})+\partial(\Delta x)^2 \simeq 0$

This can be considered a linear equation and put in the form

$[A]\Delta x=[B]$ where $A_{ij}=\frac{\partial f_i(\vec{x})}{\partial x_j}$ and $B_i=-f_i$.

We can now set the root of the non-linear equation iteratively by finding the solutions at every point of the iteration. For this, we can use

- The Newton method: $x_{k+1}=x_k+\Delta x_k$
- The two point formula: $A_{ij}=\frac{f_i(x+h_jx_j)-f_i(x)}{h_j}$

Example 4: Extremes of a multivariable function

The extremes of a function $f(\vec{x})$ are the solutions for the non-linear equation set

$\vec{g}(\vec{x})=\vec{\nabla}f(\vec{x})=0$

To find the solutions we only need to find the zeroes of $g(\vec{x})$.

Gauss Elimination Method

Let's consider we want to solve $[A]x = c$. The idea here is to make $[A]$ an upper diagonal matrix, which is easy to solve. For this we will use two steps:

1.- Forward elimination (to diagonalize the matrix)

2.- Backward substitution (to obtain the solutions bottom-to-top).

Example:

Consider the system of equations given by $\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 114 & 12 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$

1.- Forward elimination leads to $\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ -96.21 \\ 0.735 \end{bmatrix}$

2. With backward substitution we solve bottom to top, first x_3, then x_2, and finally x_1. In this case

$0.7x_3 = 0.735$ → we obtain x_3

$-4.8x_2 - 1.56x_3 = -96.21$ → we obtain x_2, and so on.

For the general case, we consider

$$\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots & a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots & a_{2n}x_n = b_2 \\ \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \cdots & a_{mn}x_n = b_m \end{bmatrix}$$

We want to eliminate a_{21}. Therefore, for the second row: $\left(a_{21} - \frac{a_{21}}{a_{11}}a_{21}\right)$ will give a new $a'_{21} = 0$. We operate the same for the rest of the rows in the matrix, until we are left with

$$\begin{bmatrix} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots & a_{1n}x_n = b_1 \\ 0 \quad a'_{22}x_2 + a'_{13}x_3 + \cdots & a'_{2n}x_n = b'_2 \\ \vdots \quad\quad \vdots & \\ 0 \quad a'_{m2}x_2 + a'_{m3}x_3 + \cdots & a'_{mn}x_n = b'_m \end{bmatrix}$$

We now proceed in a similar way for the second column, and then for the third, until we obtain the diagonal matrix. The properties of matrices ensure that this combination of sums and multiplications does not affect the result of the system of equations.

When numerically implementing the Gauss elimination method into a code, we may run into some problems:

Example: Division by zero

$$\begin{bmatrix} 0 & 5 & 6 \\ 4 & 5 & 7 \\ 0 & 2 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 11 \\ 16 \\ 15 \end{bmatrix}$$

Here, unless especified, the algorithm would not automatically recognize that the third rows has its first element already zeroed, and when trying to do the Gauss elimination it would find an error.

Example: Division by zero

$$\begin{bmatrix} 5 & 6 & 7 \\ 10 & 12 & 3 \\ 20 & 17 & 19 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ 25 \\ 56 \end{bmatrix} \longrightarrow \begin{bmatrix} 5 & 6 & 7 \\ 0 & 0 & -11 \\ 0 & -7 & -9 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 18 \\ -4 \\ -16 \end{bmatrix}$$

As we see here, the problem of a division by zero can effectively happen at any step of the calculation. This means that, at every step, we need to let the algorithm check if this happens or not. A possible solution when this happens is to allow the algorithm to pivot the various rows.

Example: Round off error

Consider diagonalizing the following system:

$$\begin{bmatrix} 20 & 15 & 10 \\ -3 & 2.249 & 7 \\ 5 & 1 & 3 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 1.751 \\ 9 \end{bmatrix} \longrightarrow \begin{bmatrix} 20 & 15 & 10 \\ 0 & 0.001 & 8.5 \\ 0 & 0 & 23375 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 45 \\ 8.501 \\ 23374 \end{bmatrix}$$

Back substitution leads to

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.625 \\ 1.5 \\ 0.99995 \end{bmatrix}$$

But the actual solution to the initial problem is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This happens due to round off errors, where significant digits were cut from the calculations, leading to a wrong solution. To prevent this happening, we have to select the correct amount of significant digits that we want to keep.

Matrix LU Decomposition

LU decomposition factors a square matrix [A] into the product of a lower triangular matrix [L] and an upper triangular matrix [U] such that [A] = [L][U]. The lower triangular matrix [L] has ones on its diagonal and the multiplication factors from the Gauss elimination process in the off-diagonal elements, while the upper triangular matrix [U] contains the pivots from the elimination process.

Example:

Decomposition of the matrix $\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}$

* 1st Step:

Multiplication of the second row by a factor $\left(\frac{64}{25}\right) = 2.56$

$\Rightarrow L_{21} = 2.56$ (first column, second row)

Multiplication of the third row by a factor $\left(\frac{144}{25}\right) = 5.76$

$\Rightarrow L_{31} = 5.76$ (first column, third row)

With this, we have transformed the matrix into $\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & -16.8 & -4.76 \end{bmatrix}$

* 2nd Step:

Multiplication of the second row by a factor $\left(\frac{-16.8}{-4.8}\right) = 3.5$

$\Rightarrow L_{32} = 3.5$ (second column, third row)

With the Gauss substitution method, we finally obtain the upper matrix $[U] = \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$

Now, for the [L] lower diagonal matrix, the diagonal elements will be ones, and the off-diagonal in the lower region will be the L_{ij} found above

$$[L] = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix}$$

It is easy to show by simple multiplication that, indeed:

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 114 & 12 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \times \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

As with the Gauss decomposition, the LU decomposition can also be useful to solve equations in matrix form. In particular; if we have $[A] = [L][U]$, then we can substitute in the equation as follows:

$$[A][x] = [c] \rightarrow [L][U][x] = [c] \rightarrow [L]^{-1}[L][U][x] = [L]^{-1}[c]$$

$$\Rightarrow [U][x] = [L]^{-1}[c] \longrightarrow [U][x] = [Z]$$

Where in the second step we just multiplied to the left by $[L]^{-1}$ and in the last step we defined $[L]^{-1}[c] = [Z]$ (or, if we rearrange terms, we have $[L][Z] = [C]$.).

Therefore, to solve the system of equations, we proceed as follows:

1.- Forward elimination to find $[L][U]$

2- Forward substitution to operate on $[L][Z] = [C]$

3.- Backward substitution to operate on $[U][x] = [Z]$

Example: let's consider our $[A]$ above and $[c] = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$

$$[L][Z] = [C] \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix} \xrightarrow[\text{(forward solving)}]{} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ -96.208 \\ 0.76 \end{bmatrix}$$

$$[U][x] = [Z] \rightarrow \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 106.8 \\ -96.208 \\ 0.76 \end{bmatrix} \xrightarrow[\text{(backward solving)}]{} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0.29048 \\ 19.691 \\ 1.0857 \end{bmatrix}$$

One may wonder how an LU decomposition is better than the Gauss method if we are adding additional steps. There are indeed cases where, once the LU decomposition is done, the number of steps is smaller. Consider the problem of solving n sets of equations with the same LHS but different RHS for each. If we define C as the time to perform a multiply/divide FLOP, the computational time will be:

Forward elimination time: $Cn^3/3$

Forward substitution time: $Cn^2/2$

Backward substitution time: $Cn^2/2$

With this:

→ Gauss total time: $n\left(\frac{Cn^3}{3} + \frac{Cn^2}{2}\right)$

→ LU total time: $\frac{Cn^3}{3} + \left(\frac{Cn^2}{2} + \frac{Cn^2}{2}\right)n \simeq \frac{4Cn^3}{3}$

For large n, the factor between the two is $\frac{n}{4}$, meaning that, in fact, Gauss elimination would take 4 times more.

Example: Finding the inverse matrix using LU decomposition

The problem is simply $[A]_{n\times m}[B]_{n\times m} = [I]_{n\times n}$ where $[B]=[A]^{-1}$, and for the right hand side we can take each of the columns of the $[B]$ matrix for each of the operations. For the first row:

$$[A]\begin{bmatrix} b_{11} \\ b_{24} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \longrightarrow [L][U]\begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \longrightarrow \underset{\substack{\downarrow \\ \text{solving for z}}}{[L][Z] = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}}, \underset{\substack{\downarrow \\ \text{solving for b.}}}{[U]\begin{bmatrix} b_{11} \\ b_{21} \\ \vdots \\ b_{n1} \end{bmatrix} = \begin{bmatrix} Z \end{bmatrix}}$$

where $[B]=[A]^{-1}=\begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$

Let's consider $A = \begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix}\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$

Calculating the first column:

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 2.56 & 1 & 0 \\ 5.76 & 3.5 & 1 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \underset{\substack{\downarrow \\ \text{(forward substitution)}}}{\longrightarrow} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -2.56 \\ 3.2 \end{bmatrix}$$

$$\begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}\begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ -2.56 \\ 3.2 \end{bmatrix} \rightarrow \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 0.04762 \\ -0.9524 \\ 4.571 \end{bmatrix}$$

Calculating the second column:

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}\begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \rightarrow (\ldots) \longrightarrow \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} 0.08333 \\ 1.417 \\ -5.000 \end{bmatrix}$$

And calculating the third column:

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix}\begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \rightarrow (\ldots) \longrightarrow \begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0.0357 \\ -0.4643 \\ 1.429 \end{bmatrix}$$

Therefore:

$$[B] = [A]^{-1} = \begin{bmatrix} 0.04762 & 0.08333 & 0.0357 \\ -0.9524 & 1.417 & -0.4643 \\ 4.571 & -5.000 & 1.429 \end{bmatrix}$$

Eigenvalues

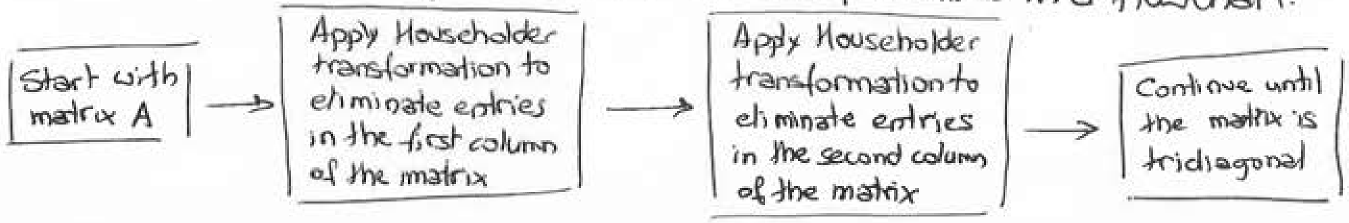
Consider the usual problem $[A][x] = \lambda[x]$ where λ is the eigenvalue for the vector $\vec{x}$ of the matrix $[A]$. We can normally determine these eigenvalues by calculating the solutions of the characteristic polynomia defined by $|A-\lambda I| = 0$.
In physics most matrices are hermitian $A^{+} = A$. In this case, the eigenvalues are all real and the eigenvectors can be made orthonormal. A complex $n \times n$ Hermitian matrix is the same as a $2n \times 2n$ symmetric matrix: $[A] = [B] + i[C]$ with $B_{ij} = B_{ji}$ and $C_{ij} = -C_{ji}$. If we decompose the eigenvector z in a similar manner, $z = x + iy$, then the original expression becomes

$$(B+iC)(x+iy) = \lambda(x+iy) \leftrightarrow \begin{pmatrix} B & -C \\ C & B \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix}.$$

In general, computing eigenvectors and eigenvalues numerically is done by first tridiagonalizing the original matrix. The tridiagonal form reduces a dense symmetric matrix to one with non-zero elements only on the main diagonal and the first diagonals above and below it. This makes the matrix easier to handle computationally. For example, computing eigenvalues of an $n \times n$ symmetric tridiagonal matrix can typically be done in $\mathcal{O}(n^2)$ operations, compared to $\mathcal{O}(n^3)$ for a full dense matrix. Furthermore, the eigenvalues of a symmetric tridiagonal matrix are guaranteed to be real and distinct, which ensures more stable and accurate algorithms.
Mathematically speaking, the tridiagonalization is achieved via a similarity transformation; i.e, $A \rightarrow T = Q^T A Q$, where Q is an orthogonal matrix.
The first algorithms for tridiagonalization were developed in the 1950s and 1960s, with the introduction of the Householder transformation and Givens rotation. Since then, various algorithms have been developed and refined, including the Faddeev - Le Verrier or the Lanczos algorithms. We will discuss these here.

The Householder Method

The basic idea in this method is to apply a series of orthogonal transformations to the matrix A to eliminate the entries outside the tridiagonal band. The Householder transformation is defined as $H = I - 2ww^t$, where w is a unit vector. The transformation is applied to the matrix A as HAH. By carefully choosing the vector w, we can eliminate the desired entries in the matrix A. The process of Householder tridiagonalization can be represented in a flowchart:


Start with matrix A
Apply Householder transformation to eliminate entries in the first column of the matrix
Apply Householder transformation to eliminate entries in the second column of the matrix
Continue until the matrix is tridiagonal

For any nxn matrix, the tridiagonalization is achieved after $(n-2)$ transformations.

The algorithm is as follows:

```
If dim(A) = n , for k = 1,2,-- n-2:
    $\alpha = -\text{sign}(a_{k+1,k})\left(\sum_{i=k+1}^{n} a_{1k}^2\right)^{1/2}$
    $r = \frac{1}{2}\alpha^2 - \frac{1}{2}\alpha\, a_{k+1,k}$
    $\omega_1^{(k)} = \omega_2^{(k)} = \dots = \omega_k^{(k)} = 0 \;;\quad \omega_{k+1} = \frac{a_{k+1,k-\alpha}}{2r}$
    for $j = k+2, \dots n \quad \omega_j = \frac{a_{jk}}{2r}$
    $H = I - 2\omega\omega^t$
    $A^{(k)} = H^{(k)} A^{(k-1)} H^{(k)}$
```

<u>Example</u>: consider the matrix $A = \begin{pmatrix} 1 & 4 & 3 \\ 4 & 1 & 2 \\ 3 & 2 & 1 \end{pmatrix}$. $A = A^t$ and $n = 3$, so only one step is needed, $k = 1$.

$$\alpha = -\text{sign}\, a_{21}\,(a_{21}^2 + a_{31}^2)^{1/2} = (-)(+)\,(4^2 + 3^2)^{1/2} = -5$$

$$r = \frac{1}{2}\alpha^2 - \frac{1}{2}\alpha\, a_{21} = \left[\frac{1}{2}(-5)^2 - \frac{1}{2}(-5)(-4)\right]^{1/2} = \frac{3\sqrt{10}}{2}$$

$$w_1 = 0 \;; \quad w_2 = \frac{a_{21} - \alpha}{2r} = \frac{4-(-5)}{2 \times 3\sqrt{10}/2} = \frac{3\sqrt{10}}{10} \;, \quad w_3 = \frac{a_{31}}{2r} = \frac{3}{2 \times 3\sqrt{10}/2} = \frac{\sqrt{10}}{10}$$

$$ww^t = \begin{bmatrix} w_1^2 & w_1w_2 & w_1w_3 \\ w_2w_1 & w_2^2 & w_2w_3 \\ w_3w_1 & w_3w_2 & w_3^2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9/10 & 3/10 \\ 0 & 3/10 & 1/10 \end{bmatrix} \rightarrow H = I - 2ww^t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -4/5 & -3/5 \\ 0 & -3/5 & 4/5 \end{bmatrix}$$

$$A^{(1)} = H^{(1)}A^{(0)}H^{(1)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -4/5 & -3/5 \\ 0 & -3/5 & 4/5 \end{bmatrix} \begin{bmatrix} 1 & 4 & 3 \\ 4 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -4/5 & -3/5 \\ 0 & -3/5 & 4/5 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & -5 & 0 \\ -5 & -2.92 & -0.56 \\ 0 & -0.56 & -0.92 \end{bmatrix}$$

It is clear that the resulting matrix $A^{(1)}$ is already tridiagonal after $k = n-2 = 3-2 = 1$ steps.
From this tridiagonal matrix we can now find the characteristic polynomia. We note that the secular equation $|A - \lambda I| = 0$ is equivalent to the polynomial equation $p_n(\lambda) = 0$. Because of the simplicity of the symmetric tridiagonal matrix, the polynomial can be generated recursively by

$$p_k(\lambda) = (a_k - \lambda)\, p_{k-1}(\lambda) - b_{k-1}^2\, p_{k-2}(\lambda)$$

where a_k are the diagonal elements $a_k = A_{kk}$ and b_k are the sub/super-diagonal elements $b_k = A_{k,k+1} = A_{k+1,k}$. The recursion is started with $p_0(\lambda) = 1$ by convention and $p_1(\lambda) = a_1 - \lambda$.

Example: From our matrix $A^{(1)}$ we identify

$a_1 = 1$ $\quad a_2 = -2.92$ $\quad a_3 = -0.92$ $\quad b_1 = -5$ $\quad b_2 = -0.56$

The characteristic polynomia will be of order $n=3$. Following the recursive recipe, we have:

$$p_0(\lambda) = 1$$
$$p_1(\lambda) = 1 - \lambda$$
$$p_2(\lambda) = (a_2 - \lambda)\, p_1(\lambda) - b_1^2\, p_0(\lambda) = (-2.92 - \lambda)(1 - \lambda) - (-5)^2 (1) =$$
$$= \lambda^2 + 1.92\lambda - 27.92$$
$$p_3(\lambda) = (a_3 - \lambda)\, p_2(\lambda) - b_2^2\, p_1(\lambda) = (-0.92 - \lambda)(\lambda^2 + 1.92\lambda - 27.92) -$$
$$- (-0,56)^2 (1 - \lambda) =$$
$$= -\lambda^3 - 2.84\lambda^2 + 26.4672\,\lambda + 25.3728$$

This polynomia can be now used to calculate the eigenvalues, For example, if we use the Newton-Raphson formula $x_{i+1} = x_i - \dfrac{f(x_i)}{f'(x_i)}$ we obtain:

$\lambda_1 = 7.07467$

$\lambda_2 = -3.18788$

$\lambda_3 = -0.886791$

The Givens Method

Givens rotations are another popular method for tridiagonalization. The idea is to apply a series of rotations to the matrix A to eliminate the entries outside the tridiagonal band. The rotation is applied to the matrix A as GAG^t. By carefully choosing the rotation angle θ, we can eliminate the desired entries in the matrix A.

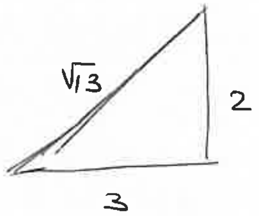
Let's consider the 3x3 matrix $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$ which we want to convert into tridiagonal form.

To achieve this, we need to transform it in such a way that $a'_{13} = 0$.

Let's consider for this the rotation matrix $R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$. The condition to make $a'_{13} = 0$ reads $tg\theta = \frac{a_{13}}{a_{12}}$.

Example: Take the matrix $A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & -1 \\ 3 & -1 & 1 \end{bmatrix}$

In this case, we see that $tg\,\theta = \frac{a_{13}}{a_{12}} = \frac{3}{2}$. By looking at the triangle on the right, with sides of lenght 2, and 3, we find that $\sin\theta = \frac{3}{\sqrt{13}}$, $\cos\theta = \frac{2}{\sqrt{13}}$ therefore:



$$A' = R^t A R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & +\sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & -1 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \\ 0 & \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & -1 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{2}{\sqrt{13}} & -\frac{3}{\sqrt{13}} \\ 0 & \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \end{bmatrix} = \begin{bmatrix} 1 & \sqrt{13} & 0 \\ \sqrt{13} & \frac{1}{13} & \frac{5}{13} \\ 0 & \frac{5}{13} & \frac{25}{13} \end{bmatrix}$$

As can be seen, we have obtained the tridiagonal form.

For the eigenvalues, the characteristic polynomia can be found to be

$$P_3(\lambda) = (1-\lambda)(\lambda^2 - 2\lambda) - 25 + 13\lambda = 0$$

$$P_3(\lambda) = \lambda^3 - 3\lambda^2 - 11\lambda + 25 = 0$$

$$\Rightarrow \lambda_1 = 4.20191$$

$$\lambda_2 = -3.11309$$

$$\lambda_3 = 1.91118$$

The Faddeev-LeVerrier Method

The F-L algorithm is a recursive numerical method to find the coefficients of the square matrix characteristic polynomia using the traces of matrix powers rather than determinants, which is computationally more efficient, and simultaneously yields the inverse matrix A^{-1}, all without explicit symbolic determinant calculation.

The algorithm generates a series of matrices $\{B_k\}_{k=1}^{n}$ and uses their traces to compute the coefficients of the characteristic polynomia $p(\lambda)$.

$$\begin{aligned}
B_1 &= A && \text{and } p_1 = Tr(B_1)\\
B_2 &= A(B_1 - p_1 I) && \text{and } p_2 = (1/2)\,Tr(B_2)\\
&\vdots && \vdots\\
B_k &= A(B_{k-1} - p_{k-1} I) && \text{and } p_k = (1/k)\,Tr(B_k)\\
&\vdots && \vdots\\
B_n &= A(B_{n-1} - p_{n-1} I) && \text{and } p_n = (1/n)\,Tr(B_n)
\end{aligned}$$

Then, the characteristic polynomia is given by

$$p(\lambda) = \lambda^n - p_1 \lambda^{n-1} - p_2 \lambda^{n-2} - \dots - p_{n-1}\lambda - p_n$$

In addition, the inverse matrix is given by $A^{-1} = \frac{1}{p_n}(B_{n-1} - p_{n-1} I)$

Example: Consider the matrix $A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{bmatrix}$. Using the algorithm,

$$B_1 = A = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{bmatrix} ; \quad p_1 = Tr(B_1) = 6$$

$$B_2 = A(B_1 - p_1 I) = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{bmatrix}\left(\begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{bmatrix} - 6\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) = \begin{pmatrix} -6 & 1 & -3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{pmatrix}$$

$$p_2 = \frac{1}{2} Tr(B_2) = \frac{1}{2}(-22) = -11$$

$$B_3 = A(B_2 - p_2 I) = \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & -1 & 2 \end{bmatrix}\left(\begin{bmatrix} -6 & 1 & -3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{bmatrix} - (-11)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}\right) = \begin{pmatrix} 6 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 6 \end{pmatrix}$$

$$p_3 = \frac{1}{3} Tr(B_3) = \frac{1}{3}(18) = 6$$

The characteristic polynomial will then be $p(\lambda) = -6 + 11\lambda - 6\lambda + \lambda^2$

The inverse matrix is

$$A^{-1} = \frac{1}{p_3}(B_2 - p_2 I) = \frac{1}{6}\left(\begin{bmatrix} -6 & 1 & -3 \\ 3 & -8 & -3 \\ -1 & 1 & -8 \end{bmatrix} - (-11)\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right) = \frac{1}{6}\begin{bmatrix} 5 & 1 & -3 \\ 3 & 3 & -3 \\ -1 & 1 & 3 \end{bmatrix}$$

The Lanczos Algorithm

The Lanczos Algorithm is an iterative method used to find the extreme eigenvalues (largest or smallest) and corresponing eigenvectors of a very large, sparse (with many zeros) Hermitian or symmetric matrix. If efficiently reduces the given matrix to a much smaller tridiagonal matrix whose eigenvalues approximate the original matrix extreme eigenvalues. This algorithm is especially useful for very large matrices where full eigenvalue decomposition is computationally expensive.

Consider as an example a map of size 1024×1024 containing information of the air quality of temperature (or any weather conditions) of a city and its surroundings. The matrix from the map will be very sparse, as we will have data only on few hundreds of measuring stations and, for statistical and properties analysis, of course we will need only the eigenvalues giving the maximal information from only those 100s of weather stations, not the thousands arising from the entire size of the matrix. With the Lanczos algorithm we can create a 100×100 matrix that will contain the interesting eigenvalues, rather than computing the 1024 of the original matrix.

Steps of the Lanczos Algorithm:

We start choosing an initial vector v_1 with unit norm.

For $k = 1, 2, \dots n$

1. Compute the matrix vector $w'_k = A v_k$
2. Compute the diagonal elements $\alpha_k = v_k^T w'_k$
3. Compute the residual vector $r_k = w'_k - \alpha_k v_k - \beta_{k-1} v_{k-1}$
4. - Compute the off-diagonal elements $\beta_k = \|r_k\|$ (the norm of r_k)

   Check for termination. If $\beta_k = 0$ then the process has finished and the final matrix contains the eigenvalues of the subspace
5. - Compute the next Lanczos vector $v_{k+1} = r_k / \beta_k$ otherwise.

After this, we can construct the tridiagonal matrix as:

$$T_{k \times k} = \begin{bmatrix} \alpha_1 & \beta_2 & \ddots & & 0 \\ \beta_2 & \alpha_2 & \ddots & & \\ & \ddots & \alpha_3 & \ddots & \\ & 0 & \ddots & \alpha_{k-1} & \beta_k \\ & & & \beta_k & \alpha_k \end{bmatrix}$$

Example: Consider the matrix $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ with initializing vector $v = (1,0,0)$ and $\beta_0 = 0$.

$k=1$: 1. Calculate $\omega_1 = A v_1 = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1 \\ 0 \end{bmatrix}$

2. Calculate $\alpha_1 = v_1^T \omega_1 = [1,0,0][2,-1,0] = 2$

3. Calculate $r_1 = \omega_1 - \alpha_1 v_1 - \beta_0 v_0 = [2,-1,0] - 2[1,0,0] - 0[0,0,0]$
$= [0,-1,0]$

4. Calculate $\beta_1 = |r_1\| = \sqrt{0^2 + (-1)^2 + 0^2} = 1$

5. Calculate $v_2 = r_1 / \beta_1 = [0,-1,0]$

At this point, we have obtained our first 1×1 tridiagonal matrix $T_1 = [\alpha_1] = [2]$. The eigenvalue of this matrix is 2, and the corresponding eigenvector is $v_1 = [1,0,0]$. This is still a poor approximation but will improve in the next steps.

$k=2$ 1.- Calculate $\omega_2 = A v_2 = [1,-2,1]$

2.- Calculate $\alpha_2 = v_2^T \omega_2 = 2$

3. Calculate $r_2 = \omega_2 - \alpha_2 v_2 - \beta_1 v_1 = [0,0,1]$

4. Calculate $\beta_2 = |r_2| = 1$

5. Calculate $v_2 = r_2/\beta_2 = [0,0,1]$

The $(2\times2]$ tridiagonal matrix is $T_2 = \begin{bmatrix} \alpha_1 & \beta_1 \\ \beta_1 & \alpha_1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$. The eigenvalues of T_2 are given by $\lambda^2 - 4\lambda + 3 = 0 \rightarrow (\lambda-1)(\lambda-3) = 0$ so the values are $\lambda = 3$, $\lambda = 1$, which are still not converging. Let's check the next step.

$k=3$: 1. Calculate $\omega_3 = A v_3 = [0,-1,2]$

2.- Calculate $\alpha_3 = v_3^T \omega_3 = 2$

3. Calculate $r_3 = \omega_3 - \alpha_3 v_3 - \beta_2 v_2 = [0,0,0]$

The residual is zero, which means we have found all the eigenvalues and vectors.

The (3×3) tridiagonal matrix is $T_3 = \begin{bmatrix} \alpha_1 & \beta_1 & 0 \\ \beta_1 & \alpha_2 & \beta_2 \\ 0 & \beta_2 & \alpha_3 \end{bmatrix} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$

The eigenvalues of this matrix are $\lambda_1 = 2$, $\lambda_{2,3} = 2 \pm \sqrt{2}$, which are the same than the eigenvalues of the original matrix A. This is reasonable since the matrix A is small and the Lanczos process exactly spans its space. We note that, if the initial vector v_1 had been an eigenvector of A, the resulting matrix T_3 would have been also equal to the original matrix A.

Fourier Analysis

Consider a vector expressed in its basis and components

$\vec{v} = v_1\vec{e}_1 + v_2\vec{e}_2 + \vec{v}_3\vec{e}_3 + \ldots v_n\vec{e}_n$

In general, we can find each of the components of the vector $\vec{v}$ by multiplying by each of the basis, since by ortogonality only that basis vector will be left and the other products will be zero: $v\vec{e}_i = v_i$. For example:

$\vec{v}\vec{e}_2 = v_1(\vec{e}_1\cdot\vec{e}_2) + v_2(\vec{e}_2\cdot\vec{e}_2) + v_3(\vec{e}_3\cdot\vec{e}_2) + \ldots v_n(\vec{e}_n\cdot e_2) = v_2$

In the context of vector spaces, the same methodology can be applied to other mathematical objects, like for example, functions. For example, if we can find a basis of "vector" functions, we can write any function in terms of such basis.

Example:

The functions $\{1, x, x^2, x^3, \ldots x^n\}$ form a basis. Therefore any function can be written as a combination of that basis. This is nothing else than the Taylor expansion:

$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots$

Fourier Series

Since $\{(\sin nx), (\cos nx)\}$ form a basis, if $f(x)$ is periodic and piecewise smooth, then it can be written as a Fourier series, a sum of sines and cosines of increasing frequency

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left(a_k\cos(kx) + b_k\sin(kx)\right)$$

The coefficients a_k, b_k can be obtained using the same trick as before, multiplying by the basis elements, but considering in this case that the basis dimension is infinite, which eventually leads to an integration over all frequencies:

$$a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx = \frac{1}{|\cos(kx)|^2}\langle f(x)|\cos(kx)\rangle$$

$$b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(kx)dx = \frac{1}{|\sin(kx)\|^2}\langle f(x)|\sin(kx)\rangle$$

Since we are expanding in terms of sines and cosines, we can also use the Euler formula $e^{ikx} = \cos(kx) + i\sin(kx)$ to write the series in complex form

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{ikx}$$

Fourier Transform

The Fourier series is defined for periodic functions. The Fourier transform integral is essentially the limit of a Fourier series as the lenght of the domain goes to infinity.
Let's consider the Fourier series on a domain $x \in [-L, L[$, and then let $L \to \infty$. On this domain, the Fourier series is

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left[a_k \cos\left(\frac{k\pi x}{L}\right) + b_k \sin\left(\frac{k\pi x}{L}\right)\right] = \sum_{k=-\infty}^{\infty} c_k e^{ik\pi x/L}$$

with $c_k = \frac{1}{2L}\int_{-L}^{L} f(x) e^{-ik\pi x/L}\, dx$.

The discrete set of frequences $\omega_k = k\pi/L$ will become a continuous range of frequences with $\Delta\omega = \pi/L$ and taking the limit, we will have:

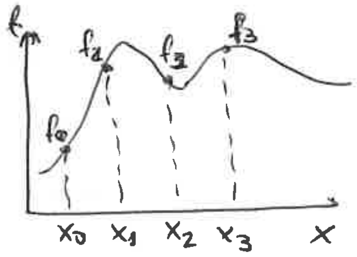
$$f(x) = \lim_{\Delta\omega \to 0} \sum_{k=-\infty}^{\infty} \underbrace{\frac{\Delta\omega}{2\pi}\int_{-\pi/\Delta\omega}^{\pi/\Delta\omega} f(\xi) e^{-ik\Delta\omega\xi}\, d\xi}_{c_k}\, e^{ik\Delta\omega x}$$

When we take the limit, the expression for c_k will become the Fourier transform of $f(x)$, denoted as $\hat{f}(\omega)$, resulting in

$$\hat{f}(\omega) = F[f(x)] = \int_{-\infty}^{\infty} f(x) e^{-i\omega x}\, dx$$

$$f(x) = F^{-1}[\hat{f}(\omega)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} \hat{f}(\omega) e^{i\omega x}\, d\omega$$

Discrete Fourier Transform



In reality, whenever we have an experiment, we will be sampling data at specific locations $(x_0, x_1, x_2 \ldots)$, so what we have is a vector of data $(f_1, f_2, f_3 \ldots)$ that we understand arises from an underlying function. In this case, the Fourier Transform will be a vector of the frequency components

$$\hat{f}_k = \sum_{j=0}^{n-1} f_j e^{-i2\pi jk/n} \qquad f_k = \sum_{j=0}^{n-1} \hat{f}_j e^{i2\pi jk/n} \qquad \{f_1, f_2, f_3 \ldots\} \Leftrightarrow \{\hat{f}_1, \hat{f}_2, \hat{f}_3 \ldots\}$$

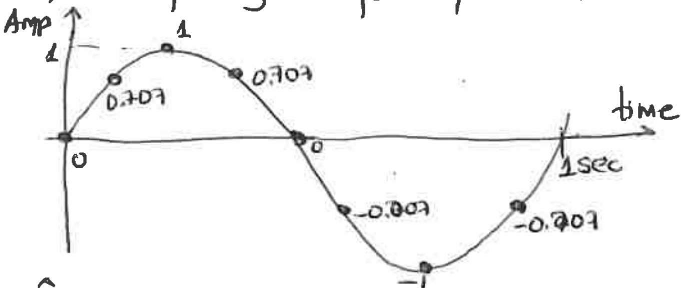
However, we don't want to calculate each of those coefficients individually. We can make it much more simple if we realise that we can put this in matrix form. Considering the fundamental frequency $w_n = e^{-2i\pi/n}$, we can write:

$$\begin{bmatrix} \hat{f}_1 \\ \hat{f}_2 \\ \hat{f}_3 \\ \vdots \\ \hat{f}_n \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_n & w_n^2 & \cdots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \cdots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & & \\ 1 & w_n^{n-1} & w^{2(n-1)} & \cdots & w_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_n \end{bmatrix}$$

We note that the resulting matrix is a complex one (i.e, it has an imaginary and a real part), and so will be the coefficients of the Fourier transform $\hat{f}(\omega)$.
Also, in general, to obtain the values of $\hat{f}$ we need to proceed with N complex multiplications and N-1 complex additions (for each k), so we will need $O(N^2)$ computations for the direct DFT.

Example:

Let's consider a sine wave with amplitude 1 and frequency 1Hz. The sampling frequency will be 8Hz, giving us 8 samples in total



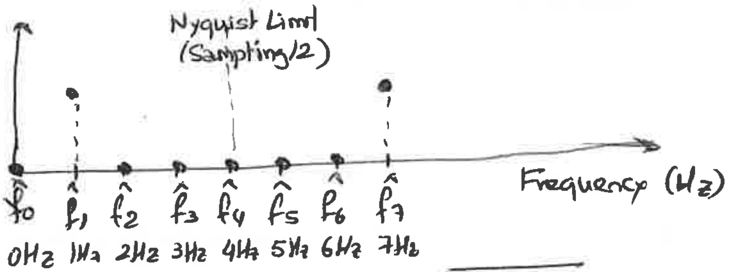
$x_0 = 0$ $\quad$ $x_4 = 0$
$x_1 = 0.707$ $\quad$ $x_5 = -0.707$
$x_2 = 1$ $\quad$ $x_6 = -1$
$x_3 = 0.707$ $\quad$ $x_7 = -0.707$

$$\hat{f}_0 = 0 + 0.707 + 1 + 0.707 + 0 + (-0.707) + (-1) + (-0.707) = 0$$

$$\begin{aligned}
\hat{f}_1 &= 0e^{-\frac{i2\pi(0)(0)}{8}} + 0.707\,e^{-j\frac{2\pi(1)(1)}{8}} + 1.\,e^{-j\frac{2\pi(1)(2)}{8}} + \dots \\
&= 0 + 0.707\left[\cos\left(-\frac{\pi}{4}\right) + i\sin\left(-\frac{\pi}{4}\right)\right] + 1\left[\cos\left(-\frac{\pi}{2}\right) + i\sin\left(-\frac{\pi}{2}\right)\right] + \dots \\
&= 0 + (0.5-0.5i) + (-i) + (-0.5-0.5j) + (0.5-0.5i) + (-1) + (-0.5-0.5i) \\
&= -4i
\end{aligned}$$

If we continue the process, we find that the coefficients are:

$\hat{f}_0 = 0$
$\hat{f}_1 = 0-4i$
$\hat{f}_2 = 0$
$\hat{f}_3 = 0$
$\hat{f}_4 = 0$
$\hat{f}_5 = 0$
$\hat{f}_6 = 0$
$\hat{f}_7 = 0-4i$



$$\text{Magnitude} = \sqrt{A_k^2 + B_k^2} = \sqrt{0^2 + (-4)^2} = 4$$

We see that we have a value at $\hat{f}_1 = f(1\,\text{Hz}) = 4$, which is natural, because our signal is of 1Hz. What is $\hat{f}_7$? This is beyond our Nyquist limit, so what we do is to get rid of everything above this limit and duplicate the values to get a single sided DFT. With this, we have an amplitude of 8, which, averaged over the number of samples (i,e, normalized), gives us the value of 1, the original amplitude. Furthermore, the phase $\theta = \frac{3\pi}{2}$ (i.e, in the direction of $(-i)$ which we know is also correct since, if we shift the cosine wave by $\theta = 3\pi/2$ we obtain the original sine wave signal.

Now, we want to factor the values that do not depend on r

$$X[k] = \sum_{r=0}^{N/2-1} x[2r] \left(W_N^2\right)^{kr} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x[2r+1] \left(W_N^2\right)^{kr}$$

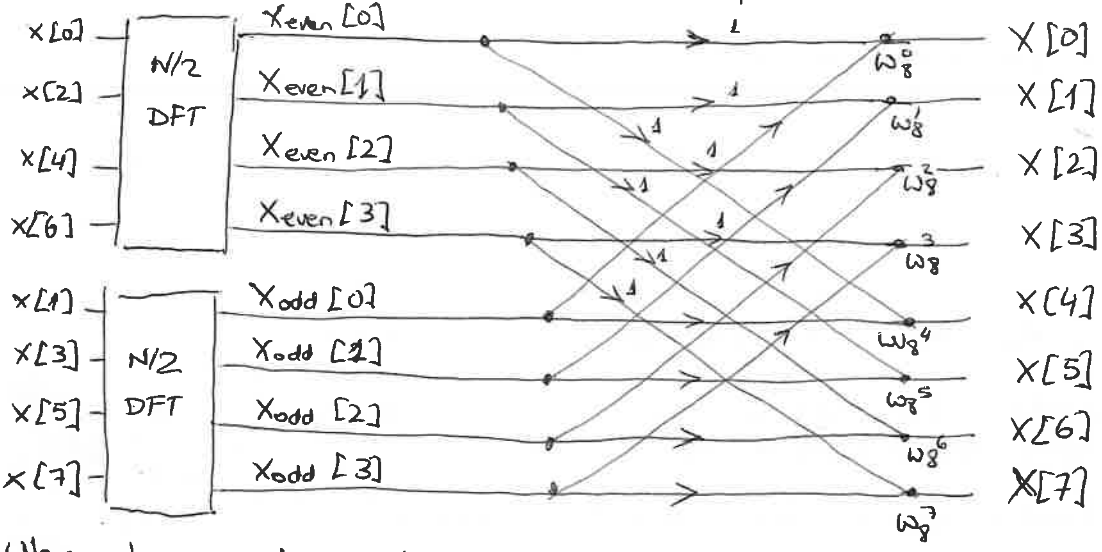
But $W_N^2 = e^{-i\frac{2\pi 2}{N}} = e^{-i\frac{2\pi}{(N/2)}} = W_{N/2}$, so we can write the expression above in terms of quantities that run only to up to $N/2$:

$$X[k] = \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r] W_{N/2}^{kr}}_{\frac{N}{2}\text{ DFT of even samples } X_{even}[k]} + W_N^k \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x[2r+1] W_{N/2}^{kr}}_{\frac{N}{2}\text{ DFT of odd samples } X_{odd}[k]}$$

Therefore we have the sum of 2 $\frac{N}{2}$ points DFT:

$$X[k] = X_{even}[k] + W_N^k X_{odd}[k]$$

If we write the diagram of this equation, what we see is as follows:



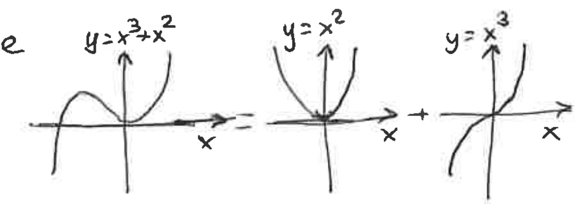
When two nodes get together, that is summation. For example, we see X[0], which has a line coming from x[0] (multiplied by 1), and another line coming from Xodd[0] (multiplied by W_8^0), which are the two terms in the sum to obtain X[0] indeed.

If we write this in the DFT matrix form that we saw before:

$$\hat{f} = F_N f = \begin{bmatrix} I_{N/2} & -D_{N/2} \\ I_{N/2} & -D_{N/2} \end{bmatrix} \begin{bmatrix} F_{N/2} & 0 \\ 0 & F_{N/2} \end{bmatrix} \begin{bmatrix} f_{even} \\ f_{odd} \end{bmatrix}$$

where $D_{N/2} = \begin{bmatrix} 1 & & & 0 \\ & \omega & & \\ & & \omega^2 & \\ 0 & & & \ddots \\ & & & & \omega^{\frac{N}{2}-1} \end{bmatrix}$, I is the identity matrix, and $F_{N/2}$ is the DFT matrix, now for only half of the operations. The left matrix is almost trivial (identity + diagonal) while the right matrix is half difficult because we have all the zeros to operate with.

Fast Fourier Transform



[Interlude]. Imagine you want to obtain the values of a given function $f(x)$. In general, this may take some time. But if we are smart, we can make use of the properties of the function to save some calculations.

For example, if the function is even, then $f_{even}(x) = f_{even}(-x)$. So we only have to calculate half of the points. Similarly, for an odd function, we have $f_{odd}(x) = -f_{ood}(-x)$, with another saving. For a general function, we may be able to split it into its even and its odd parts $f(x) = f_{even}(x) + f_{odd}(x)$ and operate accordingly.

Example:

We can split $f(x) = x^3+x^2$ into its even $f_{even}(x) = x^2$ and odd $f_{odd}(x) = x^3$ parts which are symmetric and antisymmetric respectively. Given their cyclic propeaties, the function $f(x) = e^{\pm ikx}$ also has symilar symmetry properties.

The FFT algorithm exploits the symetries of $e^{-i\frac{2\pi kn}{N}}$ which we have seen before is a key part in the calculation of the Fourier transform. Again, if we define $W_n = e^{-i2\pi/N}$, we have

i) Complex conjugate symmetry : $W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^*$

ii) Periodicity in n and k: $W_N^{kn} = W_N^{k(N+n)} = W_N^{(k+N)n}$

Let's assume the number of points to be $N = 2^m$ a power of 2, for reasons that we will see later.

$$X[k] = \sum_{n=0}^{N-1} x[x]\, e^{-i\frac{2\pi kn}{N}} \qquad k = 0, 1, \dots N-1$$

This method was invented first by Gauss in 1805 to make calculations in his head, but never published it. It was re-discovered in 1965 by Cooley & Tukey, based on the work of many people before them. We separate $X[n]$ into even and odd subsequences:

$$X[k] = \sum_{n=0}^{N-1} x[n]\, W_N^{kn} = \sum_{n \text{ even}} X[n]\, W_N^{kr} + \sum_{n \text{ odd}} X[n]\, W_N^{kr}$$

Now, we can write our new indices as:

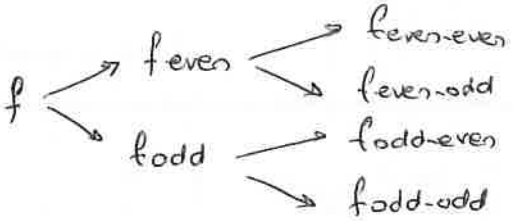
$$\left.\begin{array}{l}\text{even}\\ \text{odd}\end{array}\right\} \text{indices} \begin{cases} n = 2r \\ n = 2r+1 \end{cases}, \quad r = 0, 1, \dots \frac{N}{2}-1$$

and we can express the previous separation as a sum over r:

$$X[k] = \sum_{r=0}^{N/2-1} x[2r]\, W_n^{k2r} + \sum_{r=0}^{N/2-1} X[2r+1]\, W_n^{(2r+1)k}$$

In terms of computational power, we note that now we have to perform $2\times\left(\frac{N}{2}\right)^2$ operations (the two boxes for the $N/2$ DFT) plus N additional operation (the additions in the nodes of the previous diagram). This leaves us $2\left(\frac{N}{2}\right)^2+N \simeq \frac{N^2}{2}+N$ operations, which is (almost) a factor of half compared with the original N^2 operations of the DFT.

The beauty of this algorithm is that we do not need to stop here. On the contrary, once we have these $N/2$ elements, we can play the same trick again. For example, we can focus on the odd elements and consider the separation of the odd-odd and the odd-even. Same for the even values, which can be split between the even-odd and even-even. With this, we can keep making our DFT matrix smaller and smaller:



$$F_N \rightarrow F_{N/2} \longrightarrow F_{N/4} \longrightarrow \cdots$$

$$F_{1024} \rightarrow F_{512} \longrightarrow F_{256} \longrightarrow \cdots$$

$$\begin{matrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{matrix} \Rightarrow \begin{bmatrix} f_0 = f_0^{\text{even}} \\ f_2 = f_1^{\text{even}} \\ f_4 = f_2^{\text{even}} \\ f_6 = f_3^{\text{even}} \end{bmatrix} \Rightarrow \begin{bmatrix} f_0^{\text{even}} = f_0 = f_0^{\text{even-even}} \\ f_2^{\text{even}} = f_4 = f_1^{\text{even-even}} \end{bmatrix} \begin{bmatrix} f_1^{\text{even}} = f_2 = f_0^{\text{odd-odd}} \\ f_3^{\text{even}} = f_6 = f_1^{\text{even-odd}} \end{bmatrix}$$

$$\begin{bmatrix} f_1 = f_0^{\text{odd}} \\ f_3 = f_1^{\text{odd}} \\ f_5 = f_2^{\text{odd}} \\ f_7 = f_3^{\text{odd}} \end{bmatrix} \Rightarrow \begin{bmatrix} f_0^{\text{odd}} = f_1 = f_0^{\text{odd-even}} \\ f_2^{\text{odd}} = f_5 = f_1^{\text{odd-even}} \end{bmatrix} \begin{bmatrix} f_1^{\text{odd}} = f_3 = f_0^{\text{odd-odd}} \\ f_3^{\text{odd}} = f_7 = f_1^{\text{odd-odd}} \end{bmatrix}$$

We can keep splitting until our final F matrix has only a single element. How many times is this? We go from $N \rightarrow N/2 \rightarrow \frac{N}{4}, \ldots \frac{N}{2^{m-1}} \rightarrow \frac{N}{2^m} = 1$. A total of $p = \log_2 N$ times. This is why we initially were good with a number of points to be a power of 2, so we can optimize this division into breaking down to smaller matrices problem.

What is the total computing cost of doing this? Let's break it down for every time we split the computation in half (even/odd) of the points:

$$1.-\ N/2 \rightarrow \text{Cost} = 2\left(\frac{N}{2}\right)^2 + N \simeq \frac{N^2}{2} + N$$

$$2.-\ N/4 \rightarrow \text{Cost} = 2\left[2\left(\frac{N}{4}\right)^2 + \frac{N}{2}\right] + N = \frac{N^2}{4} + 2N$$

$$3.-\ N/8 \rightarrow \text{Cost} = 2\left\{2\left[2\left(\frac{N}{8}\right)^2 + \frac{N}{4}\right] + \frac{N}{2}\right\} + N = \frac{N^2}{8} + 3N$$

$$\vdots$$

$$p: \frac{N}{2^p} = 1 \rightarrow \text{Cost} = \frac{N^2}{2^p} + pN = \frac{N^2}{N} + N\log_2 N \simeq O(N\log_2 N)$$

(for large N)

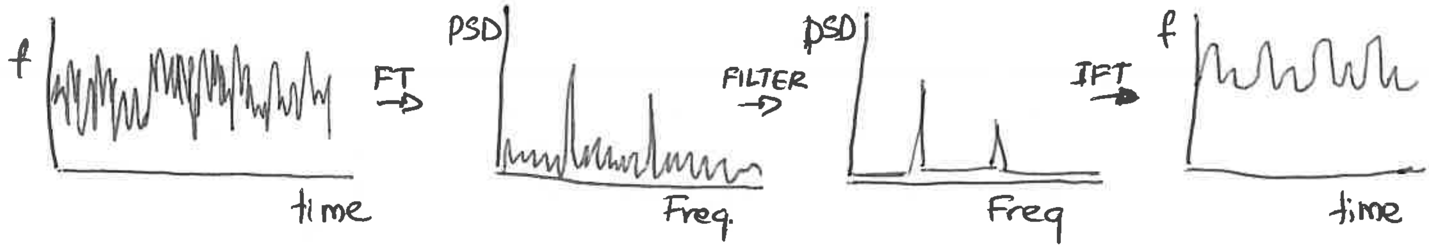
Hence, in terms of computational cost, we have moved from $O(N^2)$ to $O(N\log_2 N)$. This may not be a big difference for small N, but as this number increases, it plays a big role: Let's consider the following table:

N	N^2	$N\log_2 N$
1000	10^6	10^4
10^6	10^{12}	20×10^6
10^9	10^{18}	30×10^9

Now think that the computer can do one operation each nano second. If we had 10^9 data points (that's just a couple dozens of 8K images; or a one-second movie at 30 FPS) it would take 10^{18} ns ($\simeq 31.2$ years) to do a DFT, compared with a mere 30×10^9 ns (30 sec) of the FFT. Considering you want to dowload and watch these movies in your monitor (which does some Fourier to convert the bits into photograms), you definitely need the FFT algorithm.

Example: Noise Filtering

Consider we want to transmit a pure signal with various pure tones $f(t) = \sin(2\pi f_1 t) + \sin(2\pi f_2 t)$. However, in the process, the received signal has a very large amount of gaussian white noise added (coming, for example from interferences, bad processing, etc). White in the normal space $f(t)$ it would be very difficult to remove the noise, in the Fourier space this is very straightforward, since the signal will have two strong peaks at f_1, f_2 while the noise would not have a defined frequency and its power in the Fourier space will be much smaller. If we zero out the components in frequency that have a power below a certain threshold, we can remove the noise from the signal. After that, we just inverse-FT to recover the filtered signal.



Example:

Other cases where the Fourier transform is useful are

- Music Equalizer (boosting the chosen frequencies in the FT)
- Autotune (to select pure tones for the vocals)
- Karaoke (to select tones from the music and filter frequencies from the voice)
- Signal multiplexing (to carry different information in each band but select which frequencies we are interested in: radio 95.7 fm or 102.5 fm? Internet wifi 2.5 GHz or 5.0 GHz?, etc)
- Data compression (those frequencies with much less power need the same amount of storage, but can be easily neglected to save space without affecting the data quality: WAV → MP3)

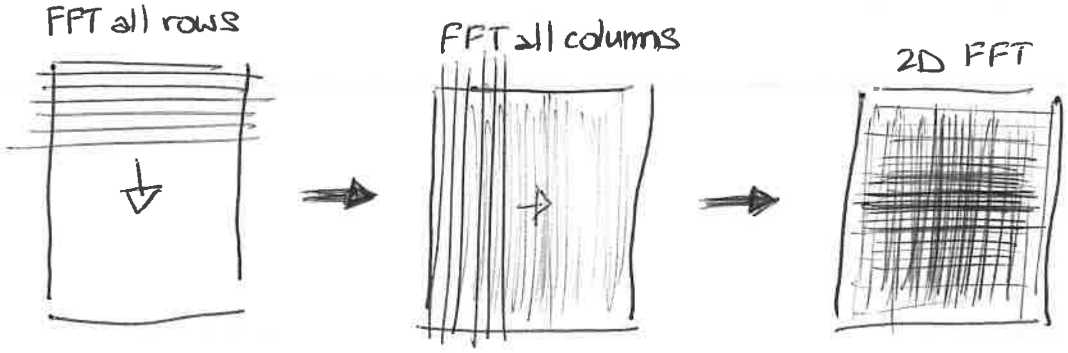
2D Fourier Transform

Extension of the FT to higher dimensions is straightforward if we note that we can transform each coordinate independently. Let's assume data in rectangular form (e.g an image) formed by a mesh with N_1 points in one direction and N_2 points in the other. The discrete Fourier transform is given by

$$g_{jk} = \frac{1}{\sqrt{N}} \sum_{l=0}^{N_1-1} \sum_{m=0}^{N_2-1} f_{lm} \exp\left[-i2\pi\left(jl/N_1 + km/N_2\right)\right] =$$

$$= \frac{1}{\sqrt{N_1}} \sum_{l=0}^{N_1-1} \exp\left[-i2\pi jl/N_1\right] \frac{1}{\sqrt{N_2}} \sum_{m=0}^{N_2-1} f_{lm} \exp\left[-i2\pi km/N_2\right]$$

Thus, we can obtain the transform first for all the terms under m with fixed l and then for all terms under index l with fixed k.



Of course, this has many applications in imaging processing. For example, we can obtain a compressed image using only 0.2% of the largest Fourier coefficients without losing too much of the information. A typical screen Full HD image 1920 × 1080 $\simeq$ 2MB can be easily converted into a mere 4 KB.

WAVELET ANALYSIS

Wavelet transformations aim to analyse spectral information at different scales and locations of the data stream. This multi-resolution approach enables different time/frequency fidelities in different bands, which is useful in decomposing signals that arise from multiscale processes such as are found in climatology, neuroscience, finance or turbulence. Audio and image signals are also amenable to wavelet analysis, which is currently the leading method for compression

Windowed Fourier Transform

A windowed FT can be formulated to select the information of the data at a specific location. We can define

$$g(\omega,\tau) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t)\, w(t-\tau)\, e^{i\omega t}\, dt$$

as the windowed FT of the function $f(t)$ under the window function $w(t-\tau)$. This window function is used to extract data in the neighborhood of $t=\tau$. The window function is commonly chosen to be a real, even, normalized function that satisfies $\int_{-\infty}^{\infty} w^2(t)\, dt = 1$. Typical window functions include:

- Triangular window: $w(t) = \begin{cases} \frac{1}{\sqrt{N}}\left(1-\frac{|t|}{\sigma}\right) & \text{if } |t|<\sigma \\ 0 & \text{elsewhere} \end{cases}$

- Gaussian function: $w(t) = \frac{1}{\sqrt{N}} e^{-t^2/2\sigma^2}$

where σ is a measure of the window width, and N is the normalization factor.

From the definition, we can recover the original data (function) from the Fourier coefficients via the inverse transform:

$$f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega,\tau)\, w(\tau-t)\, e^{-i\omega t}\, d\omega\, d\tau$$

Also, as the original FT, this also keeps the power spectrum:

$$\int_{-\infty}^{\infty} |f(t)|^2\, dt = \int_{-\infty}^{\infty} |g(\omega,\tau)|^2\, d\omega\, d\tau$$

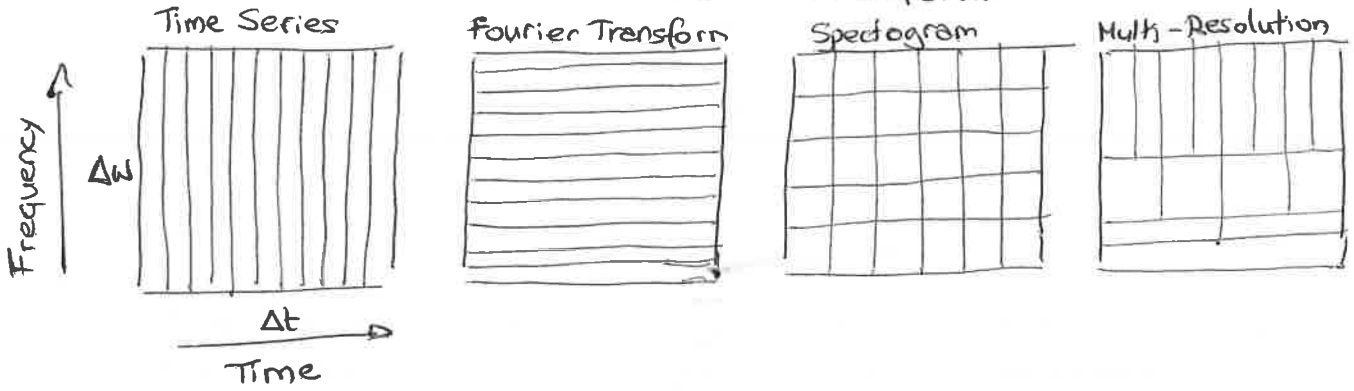
Wavelet transform

The windowed FT can provide information at a proper location in time but fails to provide the data with specific scale at the selected location, and thus is unable to distinguish structures of the data at different scales.

The basic idea of wavelet analysis is to start with a function $\psi(t)$ known as the *mother wavelet* and generate a family of scaled and translated versions of the function:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}}\,\psi\left(\frac{t-b}{a}\right)$$

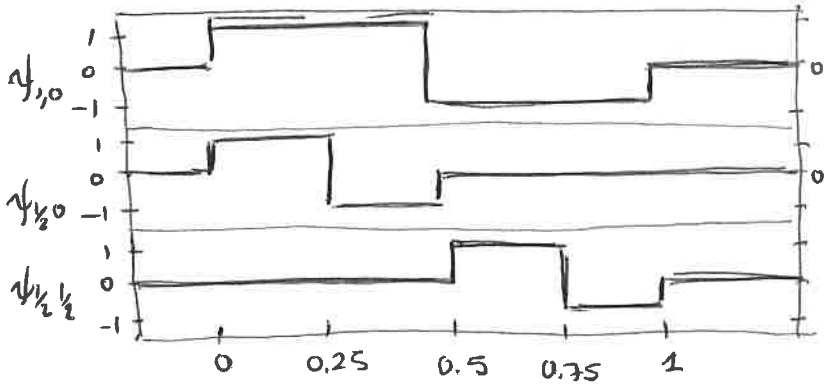
The parameters a, b are responsible for scaling and translating the function ψ. If these functions are orthogonal, then the basis may be used for projection, as in Fourier transform



For example, one can imagine trying to study in a multi-scale manner the data shown in the figure, spliting (or focusing on) the differently sized regions both in time and frequency domains. We could choose a function $\psi_{a,b}(t)$ with a, b so that the scaled an translated function fits in each of the segments of the multi-resolution.

The simplest example of a wavelet is the Haar wavelet

$$\psi(t) = \begin{cases} 1 & 0 \le t \le 1/2 \\ -1 & 1/2 \le t < 1 \\ 0 & \text{otherwise} \end{cases}$$



The three Haart wavelets $\psi_{1,0}$ $\psi_{\frac12 0}$, $\psi_{\frac12 \frac12}$ are represented on the right, showing the two layers of the multi-resolution in the figure above. Note that by choosing each higher frequency layer as a bisection of the next layer down, the resulting Haar wavelets are orthogonal, providing a hierarchical basis for a signal.

The continuous wavelet transform will be given by

$$W_\psi(f)(a,b) = \langle f, \psi_{a,b} \rangle = \int_{-\infty}^{\infty} f(t)\, \psi^{*}_{a,b}(t)\, dt$$

where $\psi^{*}_{a,b}$ denotes the complex conjugate of $\psi_{a,b}$

The inverse continuous wavelet transform is given by

$$f(t) = \frac{1}{C_\psi} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} W_\psi(f)(a,b)\, \psi_{a,b}(t)\, \frac{1}{a^2}\, da\, db$$

When computing the wavelet transform on data, it is necessary to introduce the discretized version

$$W_\psi(f)(j,k) = \langle f, \psi_{j,k} \rangle = \int_{-\infty}^{\infty} f(t)\, \psi^{*}_{j,k}(t)\, dt$$

where $\psi_{j,k}(t)$ are the discrete family of wavelets

$$\psi_{j,k}(t) = \frac{1}{a^j}\, \psi\left(\frac{t-kb}{a^j}\right)$$

If the family of wavelets is orthogonal (e.g the Haart wavelets), it is possible to expand the function $f(t)$ in this basis:

$$f(t) = \sum_{j,k=-\infty}^{\infty} \langle f(t), \psi_{j,k}(t) \rangle\, \psi_{j,k}(t)$$

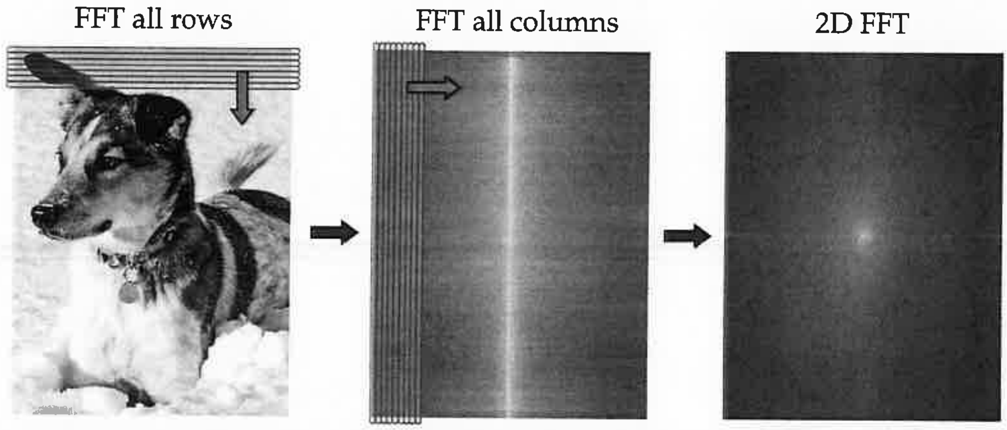
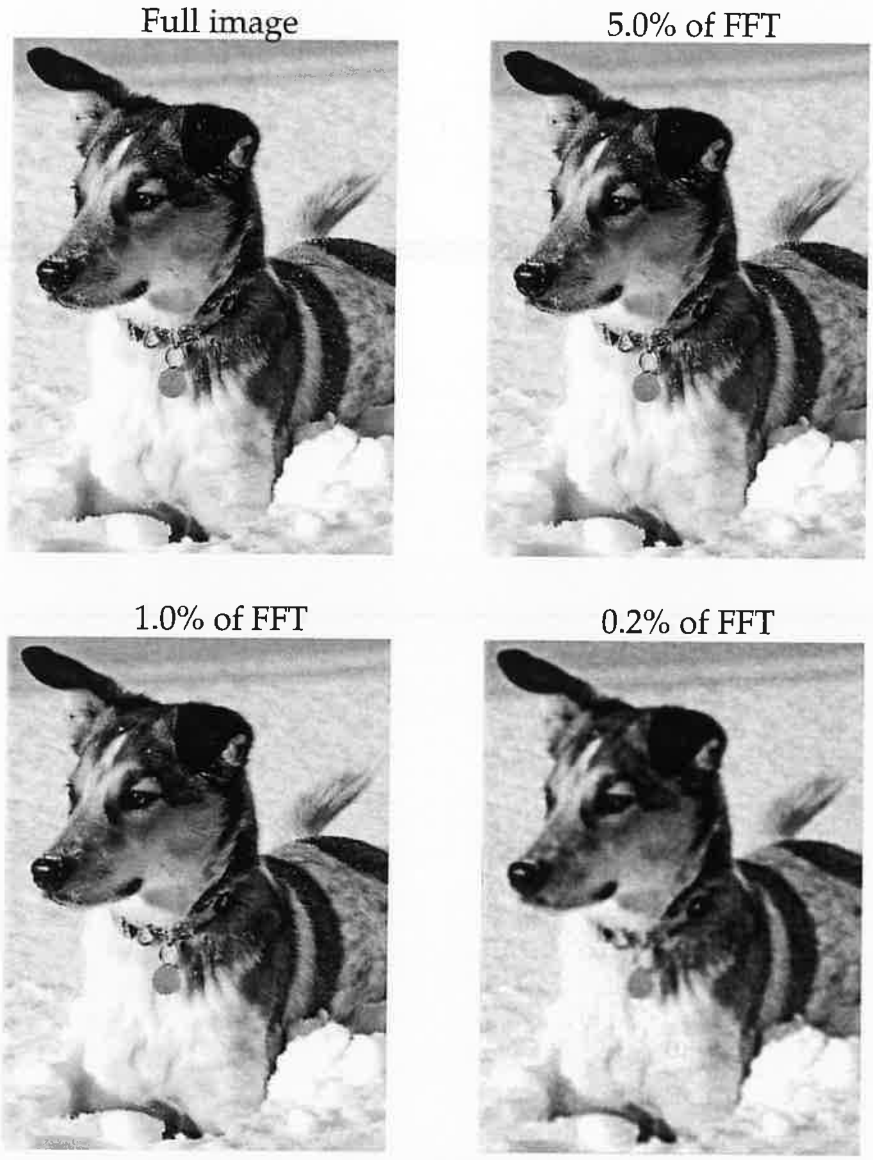


Figure 2.25 Schematic of 2D FFT. First, the FFT is taken of each row, and then the FFT is taken of each column of the resulting transformed matrix.



Full image

5.0% of FFT

1.0% of FFT

0.2% of FFT

Figure 2.26 Compressed image using various thresholds to keep 5%, 1%, and 0.2% of the largest Fourier coefficients.

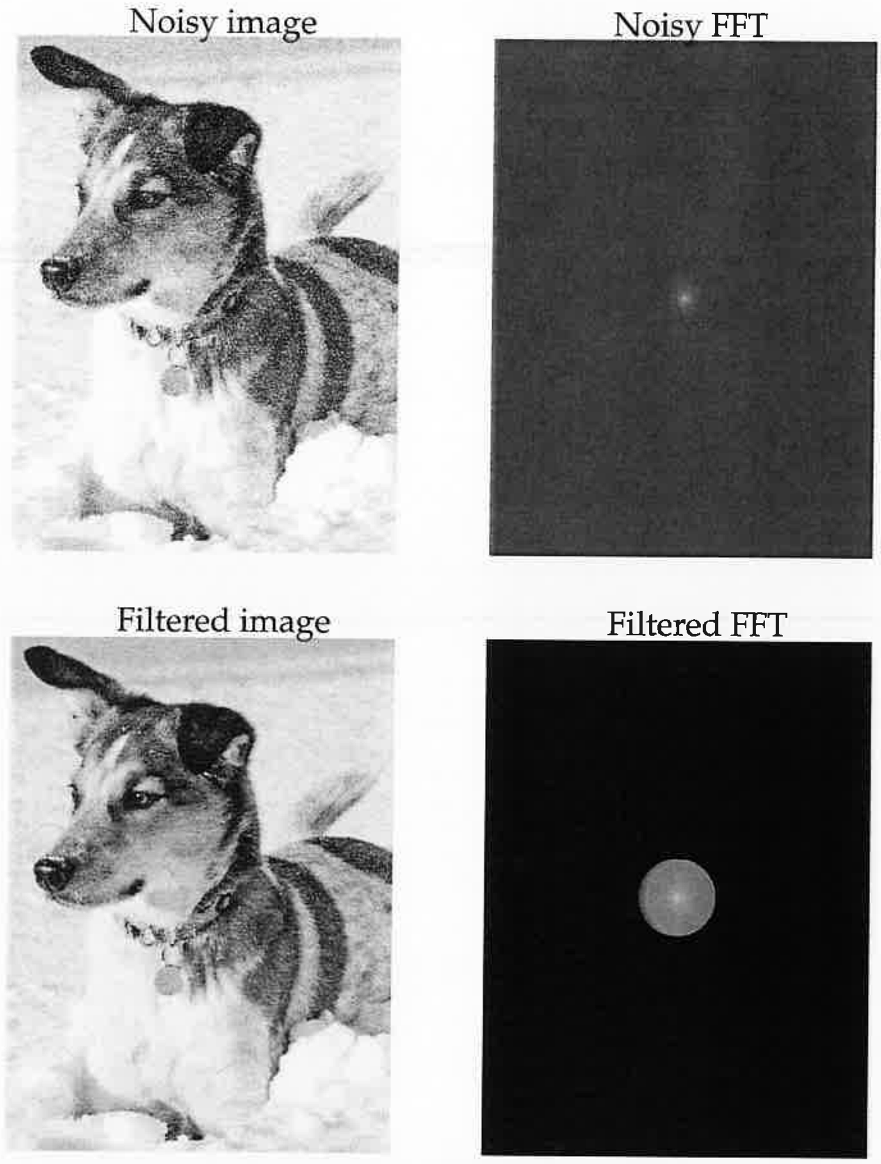


Figure 2.27 Denoising image by eliminating high-frequency Fourier coefficients outside of a given radius (bottom right).

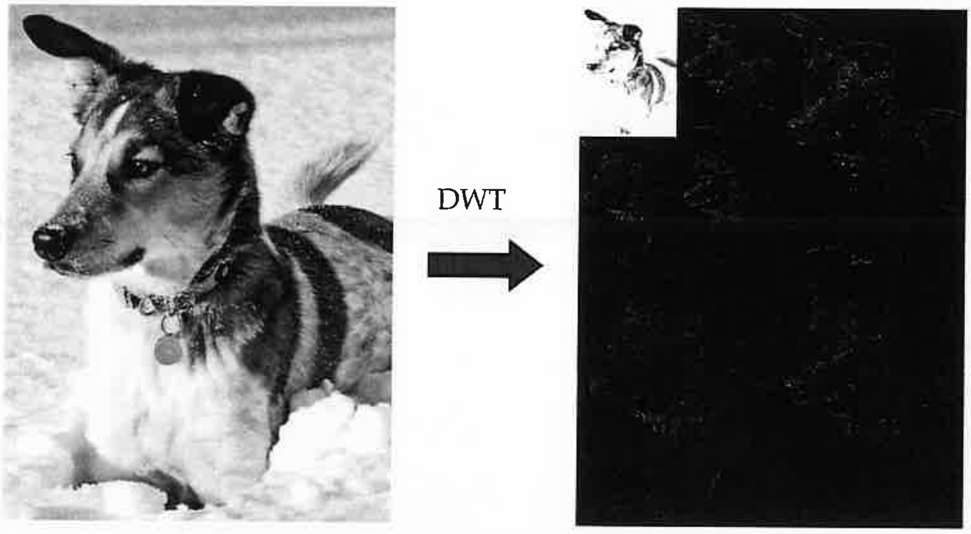


Figure 2.28 Illustration of three level discrete wavelet transform.

three levels are illustrated in Fig. 2.28. In this figure, the hierarchical nature of the wavelet decomposition is seen. The upper left corner of the DWT image is a low-resolution version of the image, and the subsequent features add fine details to the image.

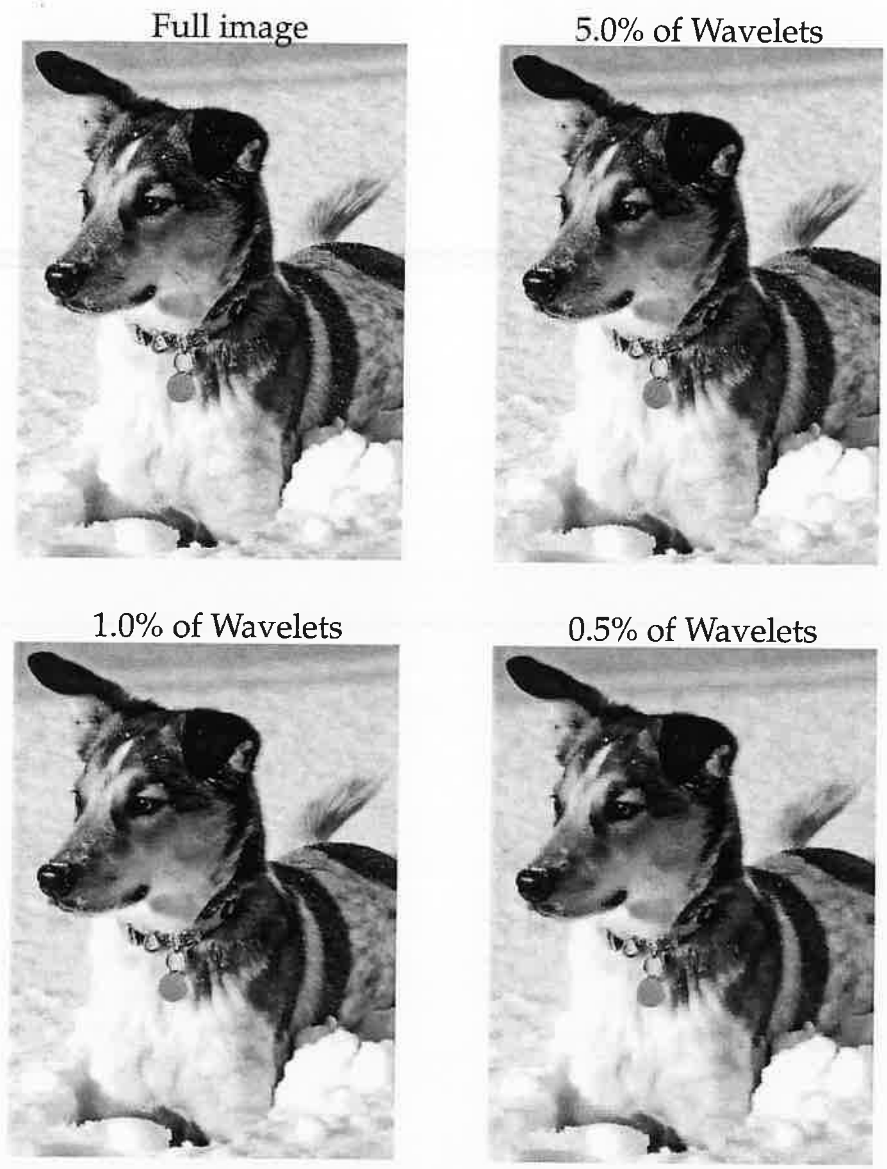


Figure 2.29 Compressed image using various thresholds to keep 5%, 1%, and 0.5% of the largest wavelet coefficients.