Lecture 10:

Fast Fourier Transforms

Fourier Transforms in simulations:

- A powerful tool for solving PDEs (basis-set discretization)

$$f(t) = \sum_{n=-\infty}^{\infty} c_n \mathrm{e}^{in\omega_0 t}$$

[similar expansions could be used in multiple dimensions, e.g. $f(x, y, z) = \sum_i \sum_j \sum_n c_{i,j,n} \exp[i(ik_x x + jk_y y + nk_z z)]$]

Note: in some algorithms, such forward- and backward- transformations are needed for each iterative step, and there will be many such iterations.

- An arbitrary function $f(t)$ => a set of coefficients c_n
- A set of coefficients c_n => the corresponding function $f(t)$

- A general signal processing tool (obtain and manipulate with frequency contents of a signal)

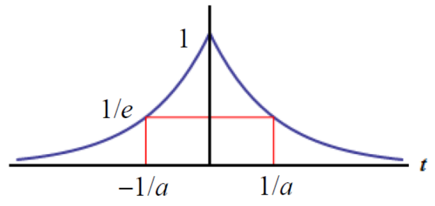
- A key tool in many algorithms for image/sound compression and processing

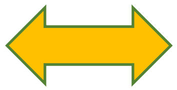
Fourier Transforms vs Fourier Series vs Discrete Fourier Transforms

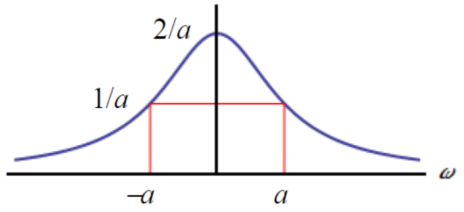
- Fourier Transforms (FT):

continuous $f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)\exp(i\omega t)\,d\omega$

continuous $F(\omega) = \int_{-\infty}^{+\infty} f(t)\exp(-i\omega t)\,dt$



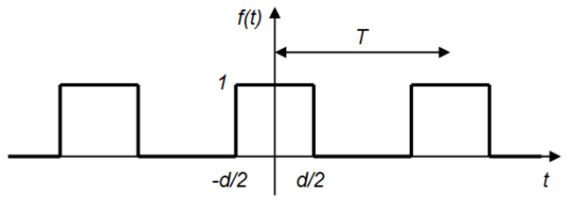


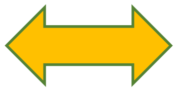


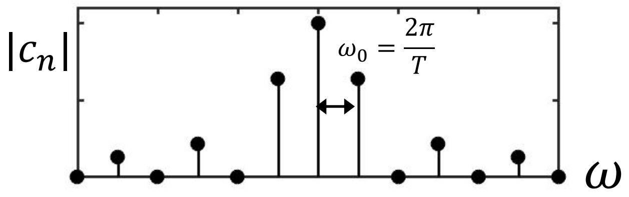
- Fourier Series (FS):

continuous periodic $f(t) = \sum_n c_n \exp(in\omega_0 t)$

discrete $c_n = \frac{1}{T}\int_0^T f(t)\exp(-in\omega_0 t)\,dt$



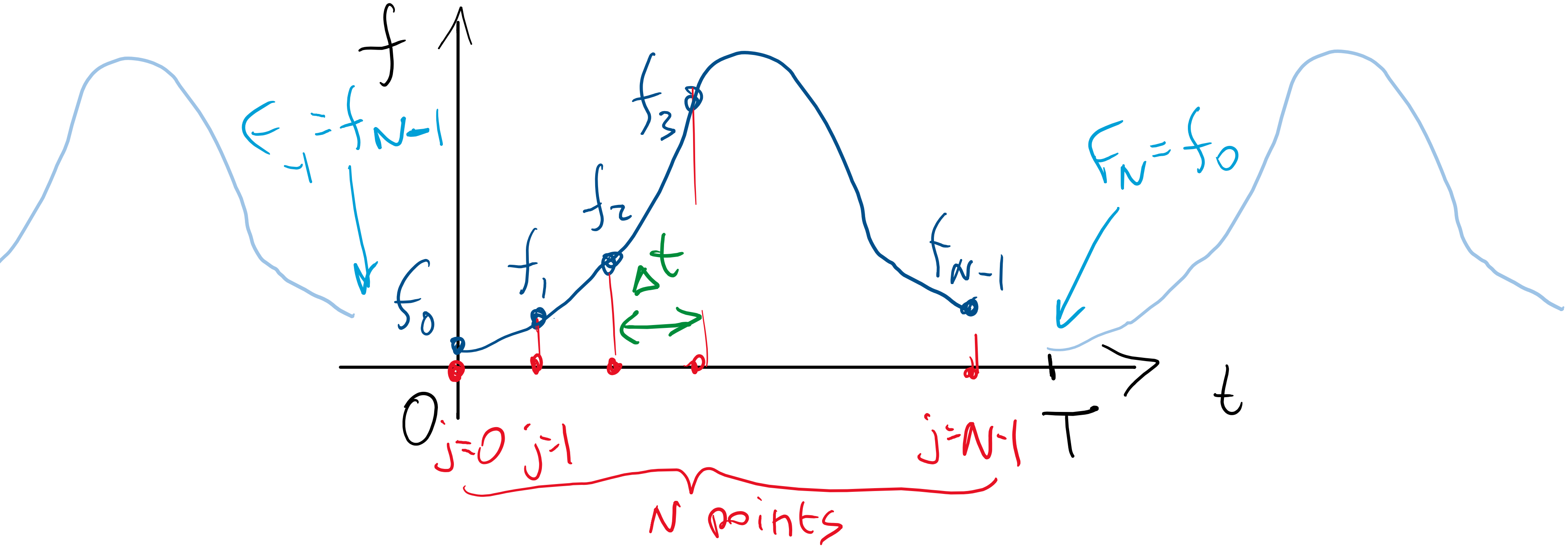




- Deal with a numerical representation of an ideal (continuous and infinite) signal:
  - The numerical function is only available in a finite window
  - The numerical function is only available at a discrete set of points

Calculating Fourier Series

- Suppose you work with a regular grid, such that you know values of a function $f(t)$ at regular intervals $f_j = f(j\Delta t), j = 0,1,2, \dots N-1$



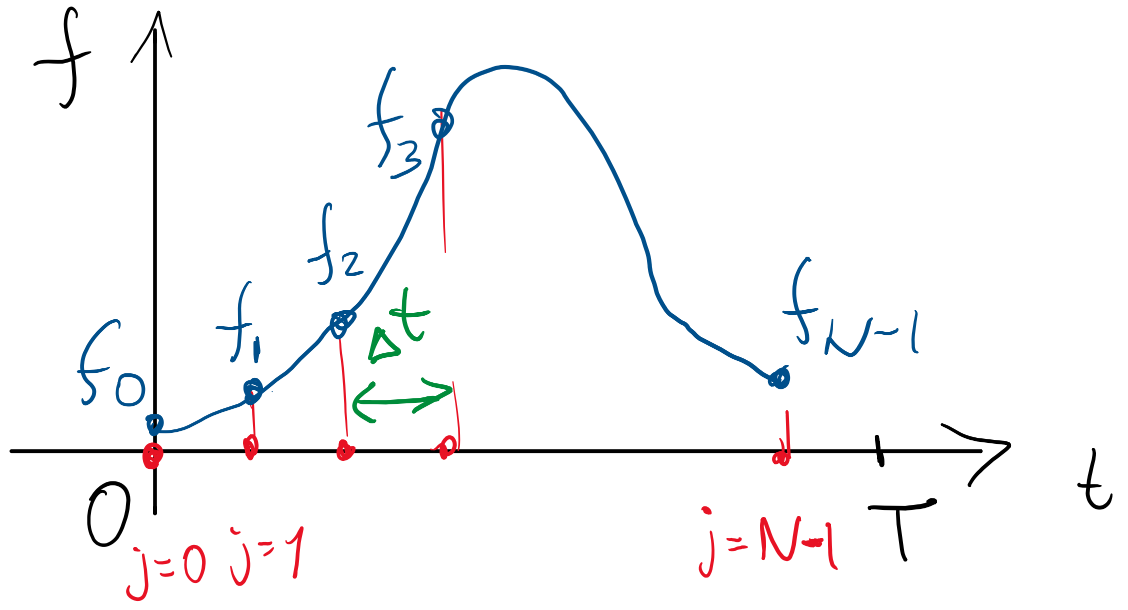
- It is convenient to introduce a periodic extension of this function, which satisfies:

$$f(t) = f(t+T), \; T = N\Delta t \qquad \text{i.e.} \quad \boldsymbol{f_{N+n} = f_n}$$

Calculating Fourier Series

$$f(t) = f(t+T)$$

$$T = N\Delta t$$



- Periodic function => can expand in Fourier series:

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]$$

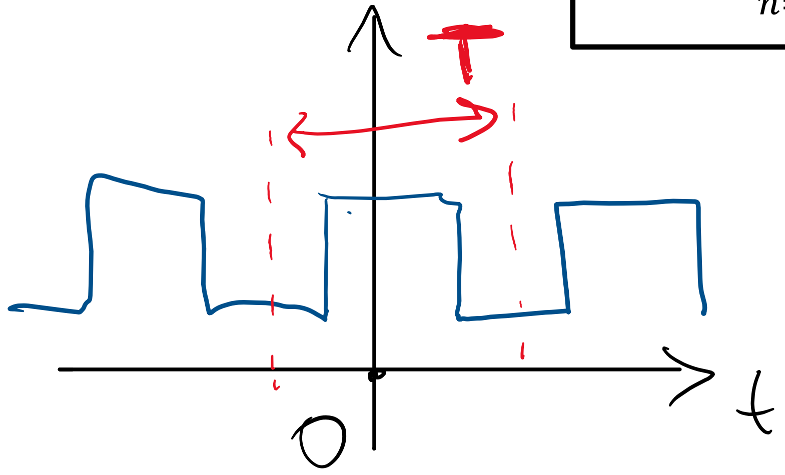
where $c_n = \frac{1}{T}\int_0^T f(t)\exp\left[-in\left(\frac{2\pi}{T}\right)t\right]dt$

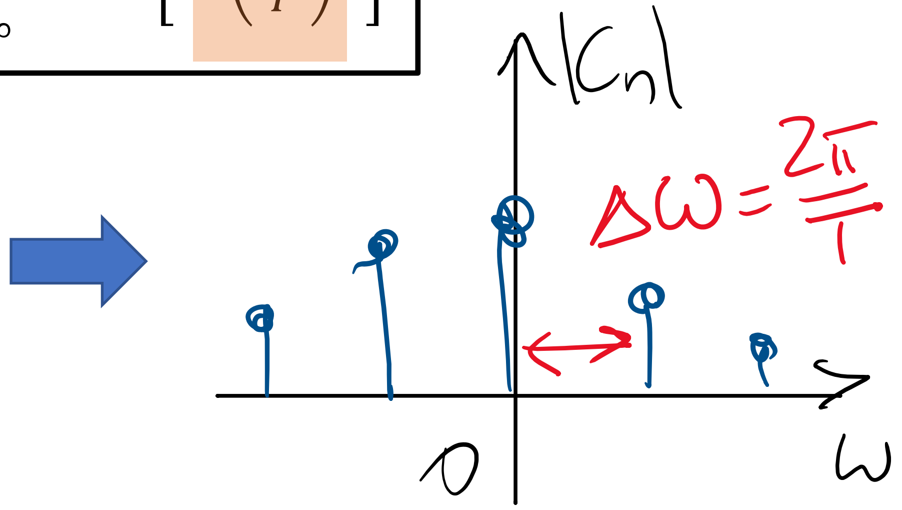
NOTE: FS instead of FT

- A periodic function has a discrete spectrum (Fourier Series)

$$\boxed{f(t) = \sum_{n=-\infty}^{+\infty} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]}$$

$\omega_n = n \cdot (2\pi/T)$



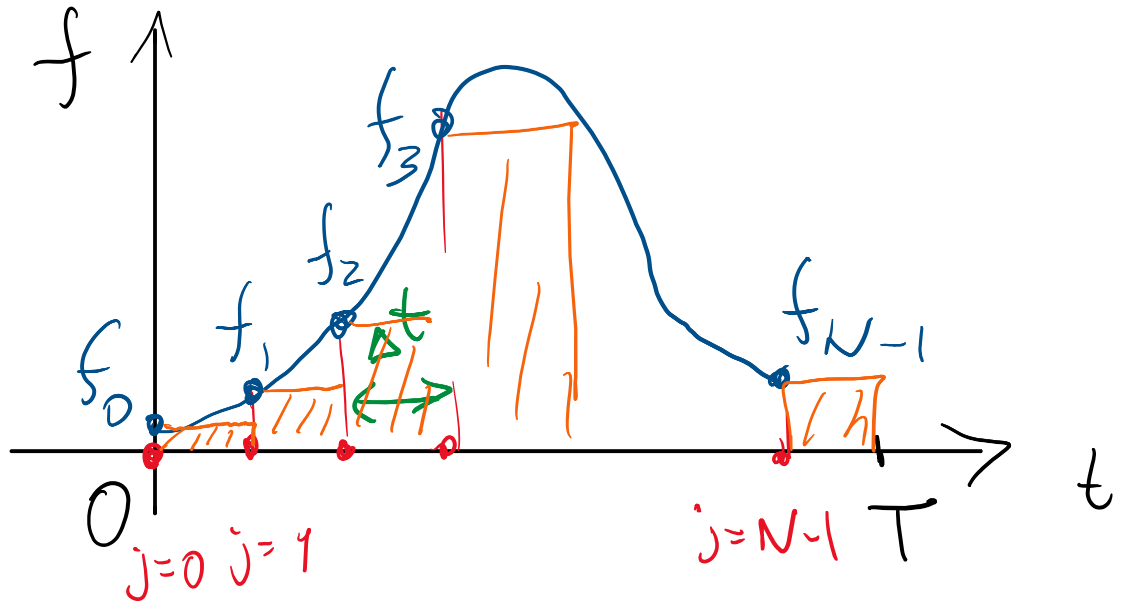


- If the actual signal is not periodic, by introducing a period T you are defining the resolution in frequency domain.

If you need better resolution in frequency, use larger time window T

Calculating Fourier Series

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]$$



- Need to compute:

$$c_n = \frac{1}{T}\int_0^T f(t)\exp\left[-in\left(\frac{2\pi}{T}\right)t\right]dt$$

$$t_j = j \cdot \Delta t$$

- Can use the trapezoid rule:

$$c_n \approx \frac{1}{T}\sum_{j=0}^{N-1} f_j \exp\left[-in\left(\frac{2\pi}{T}\right)j\Delta t\right]\cdot \Delta t$$

Calculating Fourier Series

$$\boxed{c_n \approx \frac{1}{T} \sum_{j=0}^{N-1} f_j \exp\left[-in\left(\frac{2\pi}{T}\right) j\Delta t\right] \cdot \Delta t}$$

- Use $T = N \cdot \Delta t$

$$c_n \approx \frac{1}{N \cdot \cancel{\Delta t}} \sum_{j=0}^{N-1} f_j \exp\left[-in\left(\frac{2\pi}{N \cdot \cancel{\Delta t}}\right) j\cancel{\Delta t}\right] \cdot \cancel{\Delta t}$$

$$\Rightarrow \boxed{c_n \approx \frac{1}{N} \sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right) jn\right]}$$

Note: the time step Δt and time interval T have completely disappeared!

Calculating Fourier Series

$$\boxed{c_n \approx \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]}$$

- How many coefficients to calculate?

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]$$

- Formally requires an infinite set of coefficients

BUT

$$\exp\left[-i\left(\frac{2\pi}{N}\right)j(n+N)\right] = \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]\underbrace{\exp[-i2\pi j]}_{=1} = \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]$$

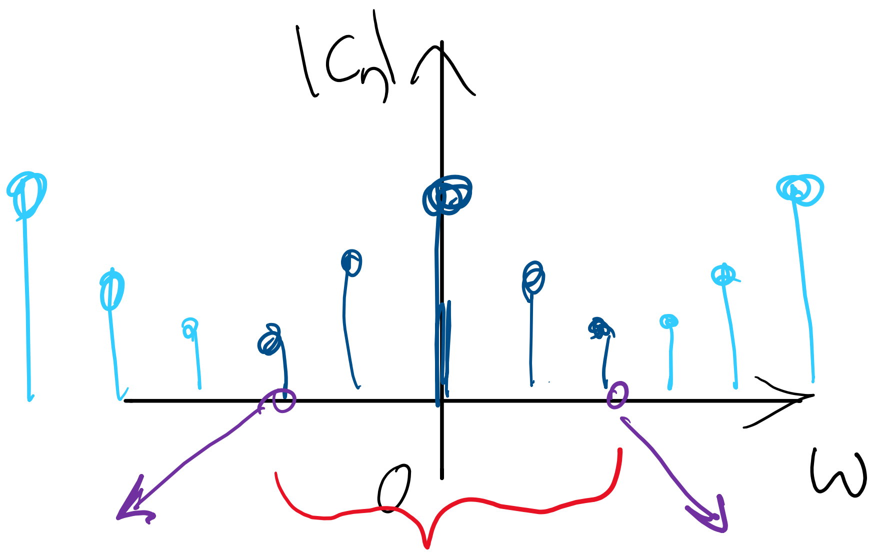
$$\Rightarrow \boxed{c_{n+N} = c_n}$$

NOTE: Mind aliasing!

$$\boxed{c_{n+N} = c_n}$$

$$\boxed{c_n \approx \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]}$$

$$\omega_n = n\left(\frac{2\pi}{T}\right) = n\left(\frac{2\pi}{N\Delta t}\right)$$



$$\omega_{-N/2} = -\frac{N}{2}\left(\frac{2\pi}{N\Delta t}\right) = -\frac{2\pi}{\Delta t}$$

N coeffs.

$$\omega_{N/2-1} = \frac{N-1}{N}\frac{2\pi}{\Delta t}$$

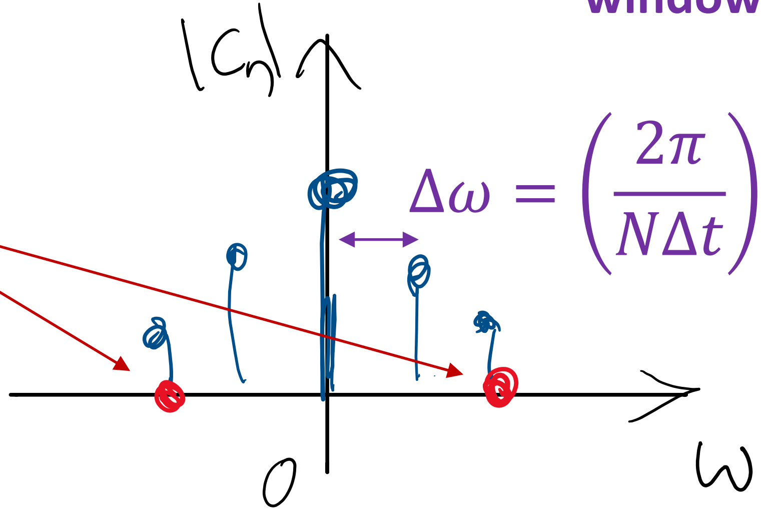
- Any spectral information for $|\omega| > 2\pi/\Delta t$ will be lost!

  (this is known as aliasing)

NOTE: Managing conflicting demands

- **If you need wider spectral window, use smaller sampling interval $\boldsymbol{\Delta t}$**
- **If you need better resolution in frequency, use larger time window $\boldsymbol{T = N\Delta t}$**

$$|\omega_{max}| = \frac{2\pi}{\Delta t}$$



$$\Delta\omega = \left(\frac{2\pi}{N\Delta t}\right)$$

- If you need both, the only way to go forward is to increase the number of sampling points N

➡ Can face a tough choice of accuracy vs computation time

Discrete Fourier Transforms (DFT)

$$\boxed{c_n \approx \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]}$$

- **Obtain N numbers**
  $\{\mathbf{c_n}\} = \boldsymbol{c_0, c_1, c_2, \dots c_{N-1}}$

- **Take N numbers**
  $\{\mathbf{f_j}\} = \boldsymbol{f_0, f_1, f_2, \dots f_{N-1}}$

- Such transformation of a set of numbers is known as Discrete Fourier Transform:

$$X_n = DFT[\{x_j\}] = \sum_{j=0}^{N-1} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]$$

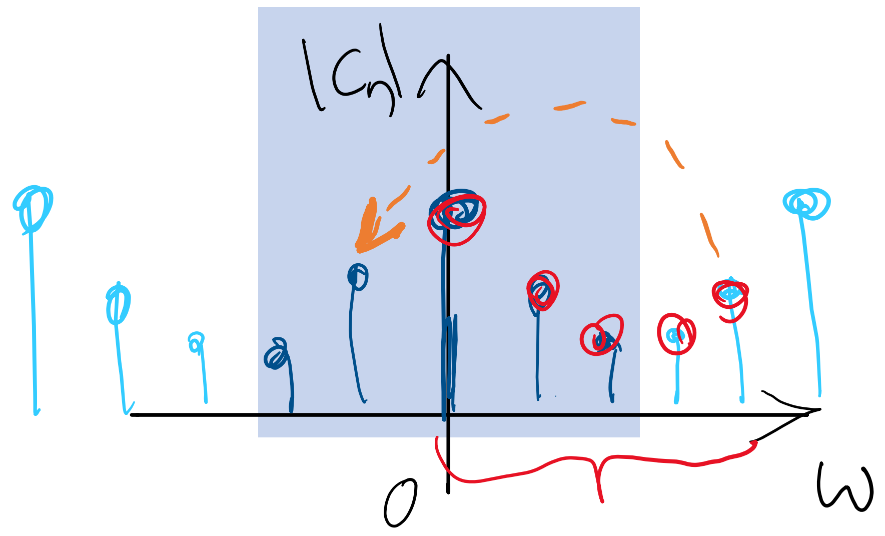
- And therefore:

$$\boldsymbol{c_n \approx \frac{1}{N} DFT[\{f_j\}]\ \text{for}\ n = 0, 1, 2, \dots N-1}$$

...and $\boldsymbol{c_{n+N} = c_n}$ (in particular, $c_{-1} = c_{N-1}, c_{-2} = c_{N-2}$ and so on)

Replacing FT/FS with DFT

- Need to obtain spectrum of $f(t)$ from its N-point discretization $f_j = f(j\Delta t)$
  - Spectrum will be discrete because of the finite-size window, $\Delta\omega = 2\pi/(N\Delta t)$
  - Spectrum will be finite because of the time discretisation, $|\omega_{max}| = 2\pi/\Delta t$



You can move DFT coefficients to the negative frequency range by using $c_{n-N} = c_n$

DFT will give you these values

$$c_n \approx \frac{1}{N} DFT[\{f_j\}] \text{ for } n = 0, 1, 2, \dots N-1$$

Note: the approximation here is due to replacing integrals with discrete sums. The quality of discretization can be assessed by checking aliasing features (presence of any noticeable signal close to the spectral window edges $|\omega_{max}| = 2\pi/\Delta t$ is an indicator of a possible aliasing!)

Inverse Fourier transform

$$\boxed{c_n \approx \frac{1}{N}\sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right) jn\right]}$$

$$\boxed{c_{n+N} = c_n}$$

- Only need to calculate N Fourier coefficients: $\boldsymbol{n = 0, 1, \ldots N-1}$
- But what about the inverse Fourier transform?

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]$$

$$\Rightarrow \quad f(t) \approx \sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right] \quad ?$$

- It is possible to show that $f_j = f(j\Delta t) = \sum_{\boldsymbol{n=0}}^{\boldsymbol{N-1}} c_n \exp\left[in\left(\frac{2\pi}{T}\right)t\right]$

[see Appendix slides for more details!]

Discrete Fourier Transform (DFT) and Inverse Fourier Transform (IDFT)

- Working with sets of complex numbers $\{f_j\}, j = 0,1,..,N-1$

[A discretized function f(t) on an N-point regular grid: $f_j = f(j\Delta t)$]

- Forward transform:

$$F_n = DFT\left[\{f_j\}\right] = \sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right], \qquad n = 0,1,\ldots,N-1$$

Spectrum:

FS: $f(t) = \sum_n c_n \exp(in\omega_0 t)\,,\ \omega_0 = 2\pi/(N\Delta t)$

$c_n \approx (1/N)F_n$ $[n = 0,1,2,\ldots N/2]$

$c_n \approx (1/N)F_{n+N}$ $[n = -N/2+1,\ldots,-2,-1]$

FT: $f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)\exp(i\omega t)\,d\omega$

$F(\omega = n\omega_0) = Tc_n \approx \Delta t F_n$

$F(\omega = n\omega_0) \approx \Delta t F_{n+N}$

- Inverse transform:

$$f_n = IDFT\left[\{F_j\}\right] = \frac{1}{N}\sum_{j=0}^{N-1} F_j \exp\left[i\left(\frac{2\pi}{N}\right)jn\right], \qquad n = 0,1,\ldots,N-1$$

- IDFT is the inverse operation of DFT: $\{f_n\} = IDFT[DFT[\{f_n\}]]$

Fast Fourier Transforms (FFT)

- Formally, DFT requires $O(N^2)$ operations:

$$X_n = DFT[\{f_j\}] = \sum_{j=0}^{N-1} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right] \qquad n = 0,1,\dots,N-1$$

- But it can be computed **much faster!**

- **If N is an even number**

$$X_n = \underbrace{\sum_{j=0,2,\dots}^{N-2} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]}_{\text{even terms}\ \frac{N}{2}\text{ in total}} + \underbrace{\sum_{j=1,3,\dots}^{N-1} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]}_{\text{odd terms}\ \frac{N}{2}\text{ in total}}$$

Fast Fourier Transforms (FFT)

$$X_n = \sum_{j=0,2,\ldots}^{N-2} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right] + \sum_{j=1,3,\ldots}^{N-1} x_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right]$$

- Change notation $\boldsymbol{j = 2k}$ in the first sum, and $\boldsymbol{j = 2k+1}$ in the second sum

$$X_n = \underbrace{\sum_{k=0}^{N/2-1} x_{2k} \exp\left[-i\left(\frac{2\pi}{N/2}\right)kn\right]} + \exp\left[-i\left(\frac{2\pi}{N}\right)n\right]\sum_{k=0}^{N/2-1} x_{2k+1} \exp\left[-i\left(\frac{2\pi}{N/2}\right)kn\right]$$

$$S_n^{(even)} = \sum_{k=0}^{N/2-1} x_{2k} \exp\left[-i\left(\frac{2\pi}{N/2}\right)kn\right] = \begin{cases} X_n^{(e)}, & 0 \le n \le \frac{N}{2}-1 \\ X_{n-N/2}^{(e)}, & \frac{N}{2} \le n \le N-1 \end{cases}$$

where $\mathrm{X}_\mathrm{n}^{(\mathrm{e})} = DFT[\{x_0, x_2, x_4, .., x_{N-2}\}]$ – nth coefficient of Discrete Fourier Transform of the **sub-set of the initial data points with even indexes**

Fast Fourier Transforms (FFT)

$$X_n = \sum_{j=0,2,\ldots}^{N-2} x_j \exp\left[-i\left(\frac{2\pi}{N}\right) jn\right] + \sum_{j=1,3,\ldots}^{N-1} x_j \exp\left[-i\left(\frac{2\pi}{N}\right) jn\right]$$

- Change notation $j = 2k$ in the first sum, and $j = 2k + 1$ in the second sum

$$X_n = \sum_{k=0}^{N/2-1} x_{2k} \exp\left[-i\left(\frac{2\pi}{N/2}\right) kn\right] + \exp\left[-i\left(\frac{2\pi}{N}\right) n\right] \underbrace{\sum_{k=0}^{N/2-1} x_{2k+1} \exp\left[-i\left(\frac{2\pi}{N/2}\right) kn\right]}$$

$$S_n^{(odd)} = \sum_{k=0}^{N/2-1} x_{2k+1} \exp\left[-i\left(\frac{2\pi}{N/2}\right) kn\right] = \begin{cases} X_n^{(o)}, & 0 \le n \le \frac{N}{2} - 1 \\ X_{n-N/2}^{(o)}, & \frac{N}{2} \le n \le N - 1 \end{cases}$$

where $X_n^{(o)} = DFT[\{x_1, x_3, x_5, .., x_{N-1}\}]$ – nth coefficient of Discrete Fourier Transform of the **sub-set of the initial data points with odd indexes**

Fast Fourier Transforms (FFT)

- Instead of computing $\{X_n\} = DFT[\{x_j\}]$ - Requires $\sim N^2$ operations
- You can compute $\{X_n^{(e)}\} = DFT[\{x_{2j}\}]$ $\sim(N/2)^2$

  $\{X_n^{(o)}\} = DFT[\{x_{2j+1}\}]$ $\sim(N/2)^2$

  $\Big\}$ $\sim N^2/2$ operations in total
- **… And there is no reason to stop there!**
- **If $N/2$ is an even number,** you could do the same trick with the subsequent DFT calculations of even and odd subsets!
- **The best efficiency is achieved if N is a power of 2**: $N = 2^k$. The computation time **scales as $\sim \boldsymbol{N \log_2 N}$**. This is much faster than N^2 for large N!

<u>This algorithm is known as Fast Fourier Transform (FFT)</u>

FFT routines are available in literally every numerical library!

Summary:

- **If you need to compute:**

  - *A Fourier Series* $c_n = \frac{1}{T}\int_0^T f(t)\exp\left[in\left(\frac{2\pi}{T}\right)t\right]dt$

  - *A Fourier Transform* $F(\omega) = \int_{-\infty}^{\infty} f(t)\exp[i\omega t]\,dt$

with **N data points** $f(t_j) = f_j$ on a regular grid of **step Δt** (such that $T = N\Delta t$)

1) Make sure you use $N = 2^k$ *(i.e. 1024 points is much better than 1000!)*

2) Calculate $F_n = DFT[\{f_j\}]$ using an FFT routine

3) $$c_n \approx \begin{cases} (1/N)F_n, & n = 0,1,2,\dots\frac{N}{2} \\ (1/N)F_{n+N}, & n = -1,-2,\dots-\frac{N}{2}+1 \end{cases}$$ **(Fourier Series coefficients)**

(Fourier Transform) $$F(n\cdot 2\pi/(N\Delta t)) \approx \begin{cases} \Delta t\cdot F_n, & n = 0,1,2,\dots\frac{N}{2} \\ \Delta t\cdot F_{n+N}, & n = -1,-2,\dots-\frac{N}{2}+1 \end{cases}$$

4) Similarly for the inverse transforms using IDFT

Appendix: Inverse Discrete Fourier Transform

- Consider the set of coefficients:

$$F_n = \sum_{j=0}^{N-1} f_j \exp\left[-i\left(\frac{2\pi}{N}\right)jn\right] \left(\approx \boldsymbol{c_n} \cdot \boldsymbol{N}\right)$$

- Multiply both sides by $\exp\left[i\left(\frac{2\pi}{N}\right)kn\right]$ and sum over n:

$$\sum_{n=0}^{N-1} F_n \exp\left[i\left(\frac{2\pi}{N}\right)kn\right] = \sum_{n=0}^{N-1}\sum_{j=0}^{N-1} f_j \exp\left[i\left(\frac{2\pi}{N}\right)n(k-j)\right]$$

Note: this is analogous to the "closure" procedure we used to do in the basis-set discretisation procedure… only in the discrete world we have summations instead of integrations

- Change the summation order in the right-hand side:

$$\sum_{n=0}^{N-1} F_n \exp\left[i\left(\frac{2\pi}{N}\right)kn\right] = \sum_{j=0}^{N-1} f_j \left\{\sum_{n=0}^{N-1} \exp\left[i\left(\frac{2\pi}{N}\right)n(k-j)\right]\right\}$$

Appendix: Inverse Discrete Fourier Transform

$$\sum_{n=0}^{N-1} F_n \exp\left[i\left(\frac{2\pi}{N}\right)kn\right] = \sum_{j=0}^{N-1} f_j \left\{\sum_{n=0}^{N-1} \exp\left[i\left(\frac{2\pi}{N}\right)n(k-j)\right]\right\}$$

- The sum of exponents:

$$\sum_{n=0}^{N-1} \exp\left[i\left(\frac{2\pi}{N}\right)n(k-j)\right] = \exp(0) + \exp\left(\frac{i2\pi}{N}(k-j)\right) + \ \ldots + \exp\left(\frac{i2\pi\cdot(N-1)}{N}(k-j)\right)$$

This is the sum of the geometric series

$$\sum_{n=0}^{N-1} \exp\left[i\left(\frac{2\pi}{N}\right)n(k-j)\right] = \frac{1-\exp[i2\pi(k-j)]}{1-\exp\left[i\left(\frac{2\pi}{N}\right)(k-j)\right]} \quad = \begin{cases} 0, & j \neq k \\ N, & j = k \end{cases}$$

$$\Rightarrow \quad \sum_{n=0}^{N-1} F_n \exp\left[i\left(\frac{2\pi}{N}\right)kn\right] = \sum_{j=0}^{N-1} f_j N \delta_{k,j} \quad = N f_k$$

Lecture 11:

Simultaneous Linear Equations: Exact Methods

- Numerical solutions of ODEs and PDEs often reduce to standard Linear algebra problems, such as:

$\hat{M}\vec{x} = \vec{b}$ - Simultaneous linear equations

$\hat{M}\vec{x} = \lambda\vec{x}$ - Eigenvalue problem

- Also, we often deal with:

$\vec{F}(\vec{x}) = \vec{b}$ - Simultaneous nonlinear equations

- Forward- and backward- Fourier transforms

- All such problems are resource demanding (scale badly with the system size N). Must think hard about the efficiency!

- Many algorithms exist in well developed libraries. Documentation often assumes you speak the language.

Simultaneous linear equations

$$\widehat{M}\vec{x} = \vec{b}$$

- Tempting to find the inverse matrix $\widehat{M}^{-1}$ [i.e. $\widehat{M}^{-1}\widehat{M} = \hat{I}$], such that the solution is given by:

$$\vec{x} = \widehat{M}^{-1} \cdot \vec{b}$$

- But usually it is **faster** (and more reliable) **to use either direct or relaxation** methods to find $\vec{x}$

- Also not every matrix can be inverted…

Simultaneous linear equations

- A **direct (exact)** method allows you to obtain an answer for $\vec{x}$, the accuracy of which is only limited by the numerical precision (truncation error)

  (In other words, you have a formula/algorithm to calculate $\vec{x}$ for any given values of the matrix elements $\widehat{M}$ and the right-hand side vector $\vec{b}$)

  Improving the accuracy is only the matter of using a higher precision format for numerical values, e.g. switching from single- (32bit) to double-precision (64bit)

- **In relaxation** methods you find $\vec{x}$ by successive iterations, starting from some initial guess $\overrightarrow{x_0}$. At each next iteration you obtain (ideally) a better approximation for the actual solution.

  To improve the accuracy, you would need to make further iterations (assuming they are converging)

In this lecture we will discuss exact methods

Gaussian elimination

$$\hat{M}\vec{x} = \vec{b}$$

- This is an exact method, based on the two properties:

  - $\vec{x}$ is unchanged if two rows in the matrix $\hat{M}$ and the two corresponding elements in $\vec{b}$ are swapped simultaneously;

  - $\vec{x}$ is unchanged if a row is replaced by a linear combination of itself and another row.

- The idea is to reduce $\hat{M}$ to the triangular form

$$\begin{pmatrix} m'_{11} & m'_{12} & \cdots & m'_{1N} \\ 0 & m'_{22} & \cdots & \cdots \\ \vdots & 0 & \ddots & \cdots \\ 0 & \cdots & 0 & m'_{NN} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_N \end{pmatrix}$$

- Then obtain all elements of $\vec{x}$ by back substitution, starting with x_N

Gaussian elimination

$$\hat{M}\vec{x} = \vec{b}$$

An example:

$$\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}$$

Gaussian elimination

$$\hat{M}\vec{x} = \vec{b}$$

- Computational time scales as N^3
- Problematic if there are small or zero elements along the diagonal

example:

$$\begin{pmatrix} 10^{-5} & 2 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}$$

$R_1 \cdot (-2) \cdot 10^5 + R_2$

$$\begin{pmatrix} 10^{-5} & 2 \\ 0 & -4 \cdot 10^5 + 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ -16 \cdot 10^5 + 3 \end{pmatrix}$$

Can face an issue with the numerical precision!

With standard double precision, $1 + 10^{-16} = 1$ *(!!!)*

- An error is quickly propagated across the entire matrix
  => this is a numerical instability!

Gaussian elimination

$\hat{M}\vec{x} = \vec{b}$

- Computational time scales as N^3
- Problematic if there are small or zero elements along the diagonal

$\Rightarrow$ The modified elimination algorithm:

1) Identify the largest by modulus element in the first column;
2) Swap the first row and the row with the largest element;
3) Proceed with the elimination;
4) Proceed to the next column and repeat

- The additional swapping of rows is known as **"pivoting".**

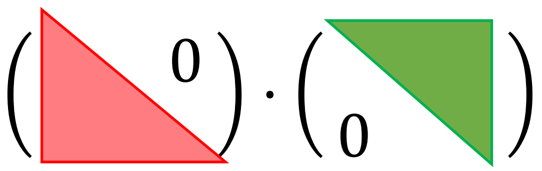
This modification requires even more time to solve the problem.

LU decomposition

$$\hat{M}\vec{x} = \vec{b}$$

- Represent matrix $\hat{M}$ as the product of two matrices: one ($\hat{L}$) has non-zero elements only in the lower triangle, and the other ($\hat{U}$) – only in the upper triangle:

$$\hat{M} = \hat{L} \cdot \hat{U} =$$



Redundancy with diagonal elements - can choose $l_{ii} = 1$

- Solution is then simple via an intermediate solution $\vec{y}$:

$$\hat{L}\vec{y} = \vec{b}$$ - obtain $\vec{y}$ via forward substitution

$$\hat{U}\vec{x} = \vec{y}$$ - obtain $\vec{x}$ via backward substitution

LU decomposition

$\hat{M}\vec{x} = \vec{b}$

An example: $\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}$

Decompose: $\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}$

$$4 = u_{11} \longrightarrow u_{11} = 4$$

$$2 = l_{21}u_{11} \longrightarrow l_{21} = 1/2$$

$$2 = u_{12} \longrightarrow u_{12} = 2$$

$$3 = l_{21}u_{12} + u_{22} \longrightarrow u_{22} = 2$$

$$\Rightarrow \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1/2 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix}$$

LU decomposition

$$\widehat{M}\vec{x} = \vec{b}$$

An example: $\begin{pmatrix} 1 & 0 \\ 1/2 & 1 \end{pmatrix} \cdot \boxed{\begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}$

$\widehat{U}\vec{x} = \vec{y}$ $\quad \begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$

$\widehat{L}\vec{y} = \vec{b}$ $\quad \begin{pmatrix} 1 & 0 \\ 1/2 & 1 \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 8 \\ 3 \end{pmatrix}$ $\longrightarrow$ $y_1 = 8$

$\longrightarrow$ $4 + y_2 = 3$ $\quad y_2 = -1$

$\begin{pmatrix} 4 & 2 \\ 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 8 \\ -1 \end{pmatrix}$ $\longrightarrow$ $4x_1 - 1 = 8$ $\quad \boldsymbol{x_1 = 9/4}$

$\longrightarrow$ $\boldsymbol{x_2 = -1/2}$

LU decomposition

$$\hat{M}\vec{x} = \vec{b}$$

- It may seem that we are replacing one problem of solving N coupled equations:

$$\hat{M}\vec{x} = \vec{b} \quad \Rightarrow \quad \sum_k m_{jk} x_k = b_j \, , \qquad j = 1,2, \ldots N$$

with another problem of N^2 coupled equations for the coefficients of $\hat{L}$ and $\hat{U}$ matrices:

$$\hat{M} = \hat{L} \cdot \hat{U} \quad \Rightarrow \quad \sum_k l_{ik} u_{kj} = m_{ij} \, , \qquad i,j = 1,2, \ldots N$$

Does this make any sense???

LU decomposition

$$\hat{M}\vec{x} = \vec{b}$$

- YES IT DOES: the solution is known, and there is a well-defined algorithm to obtain all l_{ij} and u_{ij}:

For $j = 1,2,\dots,N$ compute:

$$u_{1j} = m_{1j}$$

$$u_{ij} = m_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \qquad i = 2,\dots,j$$

$$l_{ij} = \frac{1}{u_{jj}}\left(m_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj}\right) \qquad i = j+1,\dots,N$$

$$\begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}$$

$$u_{11} = 4$$

$$u_{12} = 2$$

$$u_{22} = 3 - \frac{1}{2} \cdot 2 = 2$$

$$l_{21} = \frac{1}{4}(2-0) = \frac{1}{2}$$

LU decomposition

$\hat{M}\vec{x} = \vec{b}$

- Computational time scales as N^3
- May still require pivoting

Generally, a square matrix $\hat{M}$ can be decomposed as: $\hat{M} = \hat{P} \cdot \hat{L} \cdot \hat{U}$

where $\hat{P}$ is a permutation matrix which re-orders rows of $(\hat{L} \cdot \hat{U})$

- *(apart from possible pivoting)* LU decomposition does not require manipulations with the right-hand side $\vec{b}$.

=> MUCH more efficient than Gaussian elimination in problems which require iterative solutions of $\hat{M}\vec{x} = \vec{b}_i$ with different right-hand sides $\vec{b}_i$

LU decomposition $\widehat{M} = \hat{L} \cdot \widehat{U}$

- A useful shortcut:

If $\widehat{M}$ is a symmetric *positive-definite* matrix (i.e. all eigenvalues are positive) then it can be decomposed as:

$$\widehat{M} = \hat{L} \cdot \hat{L}^*$$

where $\hat{L}^*$ is the conjugate transpose of $\hat{L}$

This is known as **Cholesky decomposition**

Works much faster than LU decomposition

André-Louis Cholesky
1875-1918

https://en.wikipedia.org/wiki/Andr%C3%A9-Louis_Cholesky

Example: stationary states in a quantum well

$$-\frac{d^2\Psi}{d\xi^2} + V(\xi)\Psi = E\Psi,$$

Shifting the bottom of the well to $E = 0$ would make the operator positive-definite!

Tri-diagonal matrices: Recursion Method

$$\hat{M}\vec{x} = \vec{b}$$

- naturally occur when using CDA formulae for derivatives

$$\begin{pmatrix} \beta_1 & \gamma_1 & & & & 0 \\ \alpha_2 & \beta_2 & \gamma_2 & & & \\ & \alpha_3 & \beta_3 & \ddots & & \\ & & \ddots & \ddots & \gamma_{N-2} & \\ & 0 & & \alpha_{N-1} & \beta_{N-1} & \gamma_{N-1} \\ & & & & \alpha_N & \beta_N \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{N-1} \\ b_N \end{pmatrix}$$

- Can be solved by substitution $x_{i+1} = g_i x_i + h_i\,, \qquad i = 1, \dots, N-1$

- Obtain all g_i and h_i by down-recursion

$$g_{N-1} = -\frac{\alpha_N}{\beta_N}\,, \qquad h_{N-1} = \frac{b_N}{\beta_N}$$

$$g_{i-1} = \frac{-\alpha_i}{\beta_i + \gamma_i g_i}\,, \qquad h_{i-1} = \frac{\beta_i - \gamma_i h_i}{\beta_i + \gamma_i g_i}, i = N-1, .., 2$$

- Then obtain solution by up-recursion

$$x_1 = \frac{b_1 - \gamma_1 h_1}{\beta_1 + \gamma_1 g_1}\,,$$

$$x_{i+1} = g_i x_i + h_i\,, \qquad i = 1, \dots, N-1$$

- Computational time scales as N

Summary:

- Gaussian elimination and LU decomposition methods allow to solve simultaneous linear equations exactly *(up to the numerical precision)*
- LU decomposition is more efficient for problems which require multiple (recursive) solutions with different right-hand sides but the same matrix
- Also used to obtain the inverse of the matrix, and determinant of the matrix
- Computational time for Gaussian and LU methods both scale as N^3. Need more efficient methods for large systems!
- Tri-diagonal problems can be solved by recursion much faster! Computational time scales as N

Lecture 12:

Simultaneous Linear Equations: Relaxation Methods

Simultaneous linear equations

$$\hat{M}\vec{x} = \vec{b}$$

- Exact methods (Gaussian elimination, LU decomposition, or direct matrix inversion) are computationally expensive (computational time ~ N^3)
- For large system sizes we need more efficient schemes
- Also, if $\hat{M}$ is near singular (has eigenvalues close to zero), all direct methods can suffer from numerical instabilities
- Iterative (relaxation) methods offer a useful alternative

Relaxation methods

- The idea of a relaxation method is to re-arrange the initial problem

$$\widehat{M}\vec{x} = \vec{b}$$

in the form

$$\vec{x} = \hat{J}\vec{x} + \vec{c}$$

- Starting with an initial guess $\vec{x} = \vec{x}_0$, carry out iterations

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

- If iterations converge, you should observe the difference $\Delta\vec{x} = \vec{x}_{k+1} - \vec{x}_k$ gradually decays to zero, and $\vec{x}_k$ gradually approaches the true solution

Relaxation methods – will this even work???

$$\vec{x} = \hat{J}\vec{x} + \vec{c}$$

- Let $\vec{x}^{(e)}$ be the exact solution, which satisfies

$$\vec{x}^{(e)} = \hat{J}\vec{x}^{(e)} + \vec{c}$$

- Solution at k-th iteration can be written as $\vec{x}_k = \vec{x}^{(e)} + \vec{\epsilon}_k$

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c} \quad \Rightarrow \quad \vec{x}^{(e)} + \vec{\epsilon}_{k+1} = \hat{J}\vec{x}^{(e)} + \hat{J}\vec{\epsilon}_k + \vec{c}$$

- Hence the relationship between the error at $(k+1)$-th iteration and the error at the previous step is:

$$\boxed{\vec{\epsilon}_{k+1} = \hat{J}\vec{\epsilon}_k}$$

Relaxation methods – will this even work???

$$\boxed{\vec{\epsilon}_{k+1} = \hat{J}\vec{\epsilon}_k}$$

- The iterations will converge if the error $\vec{\epsilon}_k$ is reducing at each iteration step

- This will happen if and only if all the eigenvalues of $\hat{J}$ are below 1 by modulus: $|\lambda_i| < 1 \; \forall \; i$

- The maximal modulus of the eigenvalues is known as the spectral radius of the matrix $\hat{J}$

$$\rho(\hat{J}) = \max(|\lambda_i|)$$

- It determines the rate of convergence: the smaller is the spectral radius, the faster is the convergence rate.

(Roughly speaking, the rms error $|\vec{\epsilon}_k|$ at each subsequent iteration is multiplied by the spectral radius ρ.)

Relaxation methods

$$\hat{M}\vec{x} = \vec{b} \quad \Rightarrow \quad \vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

- Different relaxation methods essentially use different options for $\hat{J}$ and $\vec{c}$
- Convergence will strongly depend on the actual problem – as usual, there is no “one for all” method.
- At each iteration, the matrix product $\hat{J}\vec{x}_k$ is calculated => computational times scales as N^2

This is better than with direct schemes (~N^3)

Relaxation methods: convergence *(i.e. when shall we stop?)*

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

- Could monitor the actual error at each step, i.e.

$$\epsilon_0 = \left|\widehat{M}\vec{x}_k - \vec{b}\right| = |\widehat{M}\epsilon_k|$$

Requires an extra calculation $\widehat{M}\vec{x}_k$ at each step – a bit expensive

- Or could monitor the correction $|\vec{x}_{k+1} - \vec{x}_k|$ at each step

- As (and if) iterations converge to the exact solution, $|\vec{x}_{k+1} - \vec{x}_k|$ should become smaller and smaller

$$\epsilon_1 = \frac{|\vec{x}_{k+1} - \vec{x}_k|}{|\vec{x}_k|} \approx |\vec{\epsilon}_k| \cdot \frac{|1-\rho|}{\left|\vec{x}^{(e)}\right|} \sim \epsilon_0$$

Richardson Relaxation

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

- Re-arrange $\widehat{M}\vec{x} = \vec{b}$ as

$$\vec{x} = (\hat{I} - \widehat{M})\vec{x} + \vec{b}$$

- In other words:

$$\hat{J} = (\hat{I} - \widehat{M}), \qquad \vec{c} = \vec{b}$$

- Spectral radius of $\hat{J}$ depends on the properties of $\widehat{M}$ - will not work for any matrix!

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Richardson Relaxation

$$\hat{J} = (\hat{I} - \widehat{M}), \qquad \vec{c} = \vec{b}$$

An example:

$$\begin{pmatrix} 2 & -0.1 \\ 1 & 0.5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

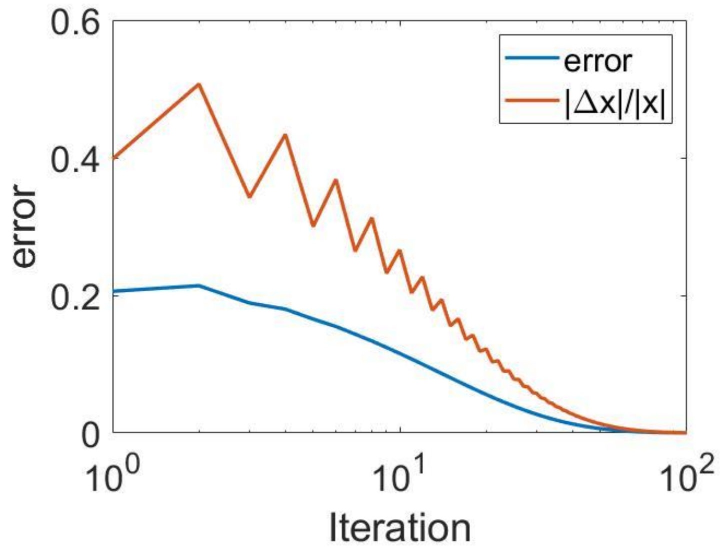
Exact solution: $\vec{x}^{(e)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$\hat{J} = \begin{pmatrix} -1 & 0.1 \\ -1 & 0.5 \end{pmatrix}$$

Eigenvalues: $\lambda_1 \approx -0.93 \quad \lambda_2 \approx 0.43$

Spectral radius: $\rho \approx 0.93 < 1$

Use $\vec{x}_0 = \begin{pmatrix} 1.2 \\ 0.2 \end{pmatrix}$



$$\vec{x}_1 = \begin{pmatrix} 0.82 \\ -0.1 \end{pmatrix} \quad \vec{x}_2 = \begin{pmatrix} 1.17 \\ 0.13 \end{pmatrix} \quad \vec{x}_3 = \begin{pmatrix} 0.84 \\ -0.11 \end{pmatrix} \quad \cdots \quad \vec{x}_{100} = \begin{pmatrix} 1.0001 \\ 0.0001 \end{pmatrix}$$

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Richardson Relaxation

$$\hat{J} = (\hat{I} - \widehat{M}), \qquad \vec{c} = \vec{b}$$

Another example:

$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Exact solution: $\vec{x}^{(e)} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

$$\hat{J} = \begin{pmatrix} -2 & -1 \\ -2 & -3 \end{pmatrix}$$

Eigenvalues: $\lambda_1 = -1 \qquad \lambda_2 = -4$

Spectral radius: $\rho = 4 > 1$

This is not going to work!

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

(Modified) Richardson Relaxation

- If $\widehat{M}\vec{x} = \vec{b}$ then $\frac{1}{a}\widehat{M}\vec{x} = \frac{1}{a}\vec{b}$ for any $a \neq 0$

$$\vec{x} = \left(\hat{I} - \frac{1}{a}\widehat{M}\right)\vec{x} + \frac{1}{a}\vec{b}$$

$$\Rightarrow \quad \hat{J} = \left(\hat{I} - \frac{1}{a}\widehat{M}\right), \qquad \vec{c} = \frac{1}{a}\vec{b}$$

- Usually can tune a to push the spectral radius of $\hat{J}$ below 1

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

(Modified) Richardson Relaxation

$$\hat{J} = \left(\hat{I} - \frac{1}{a}\widehat{M}\right), \qquad \vec{c} = \frac{1}{a}\vec{b}$$

Another example:

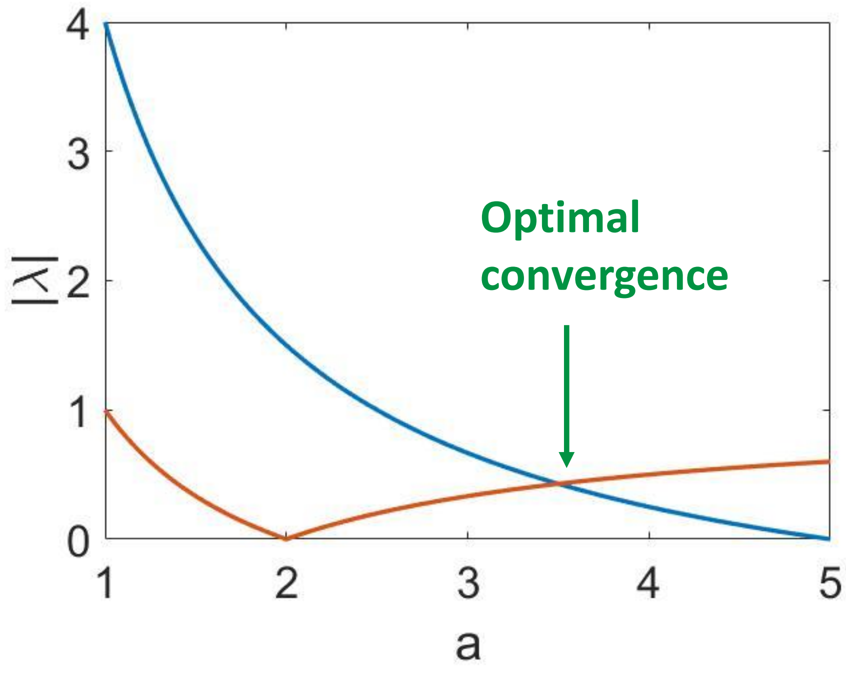
$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \qquad \hat{J} = \begin{pmatrix} 1 - 3/a & -1/a \\ -2/a & 1 - 4/a \end{pmatrix}$$

Eigenvalues: $\lambda_1 = 1 - \frac{5}{a}$

$$\lambda_2 = 1 - \frac{2}{a}$$

Minimal spectral radius: $a = 3.5$

$$\rho = \frac{3}{7} \approx 0.43$$



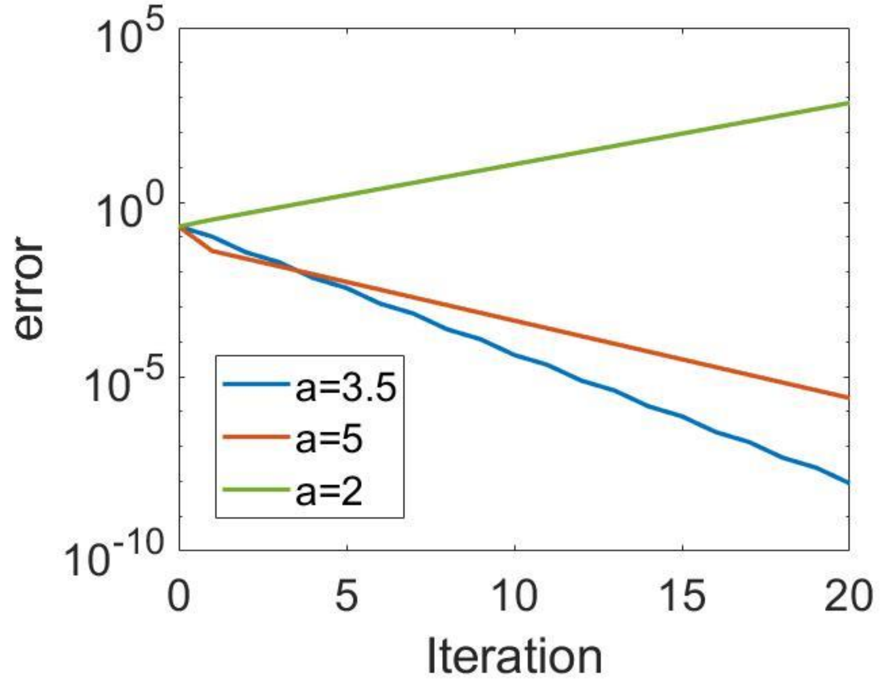
$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

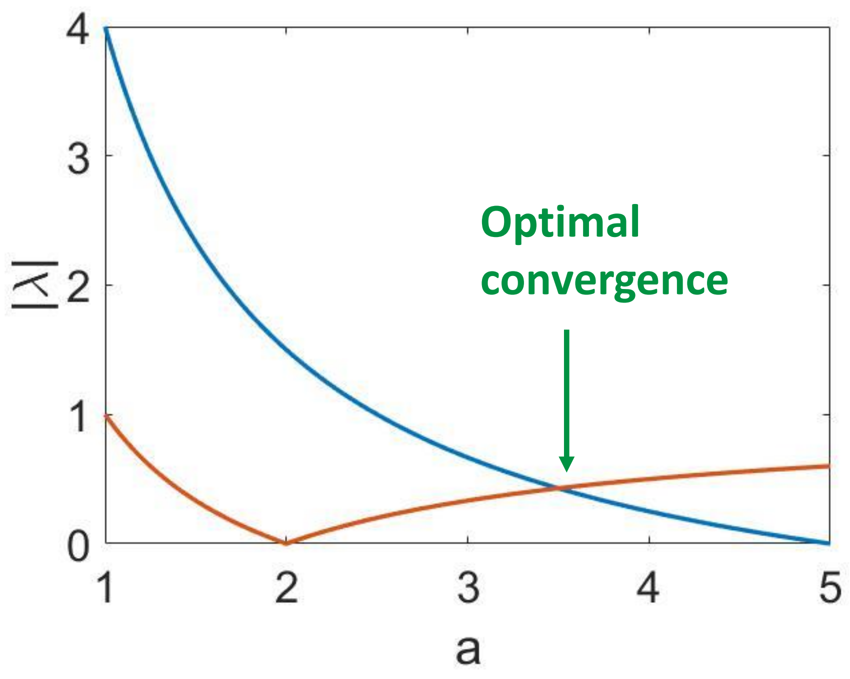
(Modified) Richardson Relaxation

$$\hat{J} = \left(\hat{I} - \frac{1}{a}\widehat{M}\right), \qquad \vec{c} = \frac{1}{a}\vec{b}$$

Another example:

$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix} \qquad \hat{J} = \begin{pmatrix} 1 - 3/a & -1/a \\ -2/a & 1 - 4/a \end{pmatrix}$$



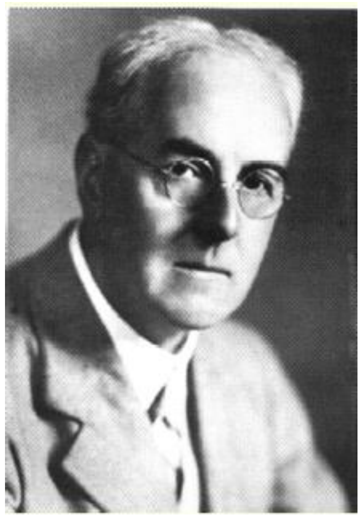


(Modified) Richardson Relaxation

- Has pioneered modern math methods of weather forecasting

- In his *“Weather Prediction by Numerical Process”* in 1922 he said:

  *“A myriad **computers** are at work upon the weather of the part of the map where each sits, but each computer attends only to one equation or part of an equation.”*



Lewis Fry Richardson
1881-1953
https://en.wikipedia.org/wiki/Lewis_Fry_Richardson

- By “computers” he meant human-beings!
  (a person who does the calculation)

- In 1950 the first weather forecast was made by an actual computer (ENIAC)

  The first calculation of a 24-hour weather forecast took ENIAC….. ~24hours

Jacobi Relaxation

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

$$\hat{M}\vec{x} = \vec{b}$$

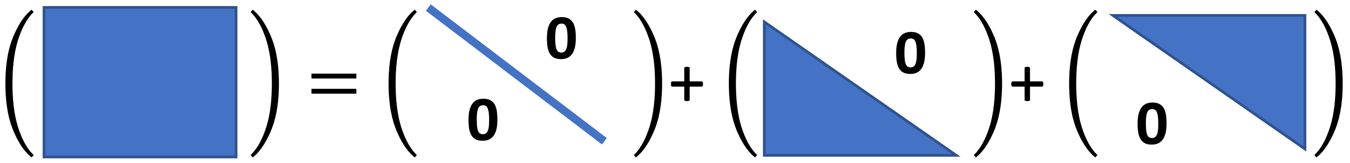
- Split $\hat{M}$ into diagonal and off-diagonal parts:

Carl Gustav Jacob Jacobi

1804-1851

https://en.wikipedia.org/wiki/Carl_Gustav_Jacob_Jacobi

$$\hat{M} = \hat{D} + \hat{L} + \hat{R}$$



E.g. $\hat{M} = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} = \begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$

Note: $\hat{L}$ and $\hat{R}$ matrices have nothing to do with the LU decomposition!

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Jacobi Relaxation

$$\widehat{M}\vec{x} = \vec{b}$$

- Split $\widehat{M}$ into diagonal and off-diagonal parts:

$$\widehat{M} = \widehat{D} + \hat{L} + \hat{R}$$

$$\Rightarrow \quad (\widehat{D} + \hat{L} + \hat{R})\vec{x} = \vec{b}$$

- Re-arrange:

$$\widehat{D}\vec{x} = -(\hat{L} + \hat{R})\vec{x} + \vec{b}$$

$$\Rightarrow \quad \boxed{\vec{x} = -\widehat{D}^{-1}(\hat{L} + \hat{R})\vec{x} + \widehat{D}^{-1}\vec{b}}$$

Note: $\widehat{D}$ is a diagonal matrix. The inversion is trivial, but requires no zero elements on the diagonal!

Jacobi Relaxation

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

$$\vec{x} = -\hat{D}^{-1}\left(\hat{L} + \hat{R}\right)\vec{x} + \hat{D}^{-1}\vec{b}$$

$$\Rightarrow \quad \hat{J} = -\hat{D}^{-1}\left(\hat{L} + \hat{R}\right), \qquad \vec{c} = \hat{D}^{-1}\vec{b}$$

An example:

$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

$$\hat{M} = \begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} = \underbrace{\begin{pmatrix} 3 & 0 \\ 0 & 4 \end{pmatrix}}_{\hat{D}} + \underbrace{\begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix}}_{\hat{L}} + \underbrace{\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}}_{\hat{R}}$$

$$\hat{J} = -\begin{pmatrix} 1/3 & 0 \\ 0 & 1/4 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 2 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1/3 \\ -1/2 & 0 \end{pmatrix} \qquad \vec{c} = \begin{pmatrix} 1/3 & 0 \\ 0 & 1/4 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}$$

Eigenvalues: $\lambda_{1,2} = \pm\frac{1}{\sqrt{6}} \approx \pm 0.41$

Spectral radius: $\rho = 0.41$

Iterations:

$$\vec{x}_{k+1} = \begin{pmatrix} 0 & -1/3 \\ -1/2 & 0 \end{pmatrix} \cdot \vec{x}_k + \begin{pmatrix} 1 \\ 1/2 \end{pmatrix}$$

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Jacobi Relaxation

$$\vec{x} = -\hat{D}^{-1}\left(\hat{L} + \hat{R}\right)\vec{x} + \hat{D}^{-1}\vec{b}$$

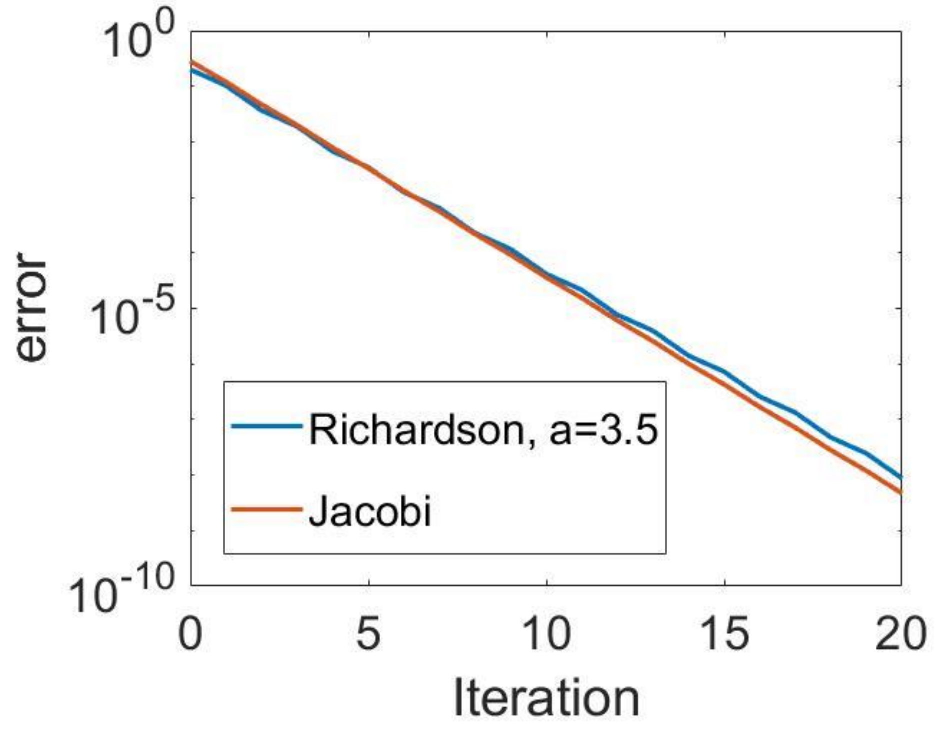
$$\Rightarrow \quad \hat{J} = -\hat{D}^{-1}\left(\hat{L} + \hat{R}\right), \qquad \vec{c} = \hat{D}^{-1}\vec{b}$$

An example:

$$\begin{pmatrix} 3 & 1 \\ 2 & 4 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$$

Jacobi spectral radius: $\rho = 0.41$

Richardson spectral radius (minimal): $\rho \approx 0.43$



$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Jacobi Relaxation

$$\vec{x} = -\widehat{D}^{-1}\left(\hat{L} + \hat{R}\right)\vec{x} + \widehat{D}^{-1}\vec{b}$$

$$\Rightarrow \quad \hat{J} = -\widehat{D}^{-1}\left(\hat{L} + \hat{R}\right), \qquad \vec{c} = \widehat{D}^{-1}\vec{b}$$

- Converges best for diagonally dominated matrix problems (i.e. where elements along the main diagonal are the largest)

Another example:

$$\begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \end{pmatrix}$$

$$\widehat{M} = \begin{pmatrix} 1 & 3 \\ 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 4 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 3 \\ 0 & 0 \end{pmatrix}$$

$$\hat{J} = -\begin{pmatrix} 1 & 0 \\ 0 & 1/2 \end{pmatrix} \cdot \begin{pmatrix} 0 & 3 \\ 4 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -3 \\ -2 & 0 \end{pmatrix}$$

Eigenvalues: $\lambda_{1,2} = \pm\sqrt{6} \approx \pm 2.45$

$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Gauss-Seidel Relaxation

- Part way through iteration $k+1$ we already know some new (better?) values of $\vec{x}$ components. Why not use them?

$$x_{k+1}^{(1)} = J_{11}x_k^{(1)} + J_{12}x_k^{(2)} + \cdots + J_{1N}x_k^{(N)} + c^{(1)}$$

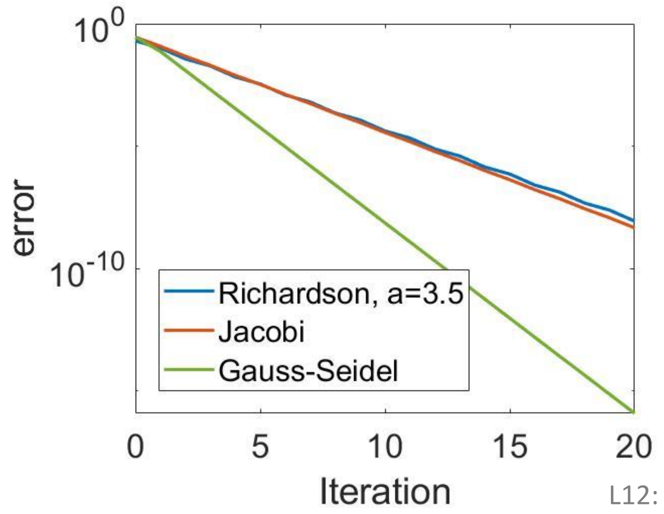
$$x_{k+1}^{(2)} = J_{21}\boldsymbol{x}_{\boldsymbol{k}}^{(\boldsymbol{1})} + J_{22}x_k^{(2)} + \cdots + J_{2N}x_k^{(N)} + c^{(2)}$$

$\updownarrow$ Replace with the new value!

$$x_{k+1}^{(2)} = J_{21}\boldsymbol{x}_{\boldsymbol{k+1}}^{(\boldsymbol{1})} + J_{22}x_k^{(2)} + \cdots + J_{2N}x_k^{(N)} + c^{(2)}$$

- The convergence rate is better:

$$\rho_{GS} \approx \rho_J^2$$



$$\vec{x}_{k+1} = \hat{J}\vec{x}_k + \vec{c}$$

Successive Over-Relaxation (SOR)

- If you know that where you are heading is better, why not going a bit faster?

Instead of: $\vec{x}_{k+1}^{(GS)} = \vec{x}_k + \Delta\vec{x}$ $\longrightarrow$ $\boldsymbol{\Delta\vec{x} = \vec{x}_{k+1}^{(GS)} - \vec{x}_k}$

try: $\vec{x}_{k+1}^{(SOR)} = \vec{x}_k + \omega\Delta\vec{x} = \omega\vec{x}_{k+1}^{(GS)} + (1-\omega)\vec{x}_k$

- Relaxation parameter: $\boldsymbol{\omega = 1}$ - initial GS scheme

  $\boldsymbol{1 < \omega < 2}$ - over-relaxation

  $\omega = \dfrac{2}{1+\sqrt{1-\rho^2}}$ - optimal

- Setting large ω at the start can lead to over-shooting

  Gradual increase of ω to the optimal value is known as **“Chebysheff acceleration”**

Summary:

- Relaxation methods can save computational effort with large-size systems (computational time scales as $\sim N^2$).

- Convergence can be very different for different schemes.

- Solution of $\widehat{M}\vec{x} = \vec{b}$ problems is in the cornerstone of many computational physics (and not only physics) problems. A lot of research has been done on this topic. Literature search will help you to identify a suitable method for your particular problem.

Lecture 13:

Eigenvalue problem

- The eigen-value problem plays crucial role in physics

- Analysis of the natural modes and frequencies of the system

E.g. eigen-states of the Schrödinger equation $-\frac{\hbar^2}{2m}\frac{d^2\Psi}{dx^2} + V(x)\Psi = E\Psi,$

- Orbital states of an atom

- Electron band structure of solids

$\Rightarrow$ ***Define all physical properties of the system!***

- Eigen-states of differential operators can be used for basis-set expansion => a powerful technique to solve PDEs

E.g. wave equation $\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$ has eigen-states

$$\boldsymbol{u_k = \exp(ikx)\exp(-ickt)}$$

An arbitrary initial condition $u(x, t = 0) = f(x)$ evolves as:

$$u(x,t) = \int c_k \exp(ikx - ickt)dk, \qquad c_k = (1/2\pi)\int f(x)\exp(-ikx)dx$$

- Stationary analysis

- Often the starting point is a PDE with time derivatives

$$-\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2}+V(x)\Psi=\mathrm{i}\hbar\frac{\partial\Psi}{\partial t}$$

- Searching for stationary solutions

$$\Psi(t)\sim\exp\left(-\frac{i}{\hbar}Et\right)$$

- Arrive at the eigenvalue problem

$$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+V(x)\right)\Psi=E\Psi$$

E and Ψ are both unknown

- Eigenvalue problem $$\left(-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2}+V(x)\right)\Psi = E\Psi$$

- Apply discretisation (grid or basis-set), formulate as the matrix eigen-value problem:

$$\boxed{\hat{M}\vec{\mathrm{x}} = \lambda\vec{\mathrm{x}}}$$

- Usually deal with large size matrices

  e.g. grid discretisation of a 1D quantum well problem will typically require $N \sim 10^2 - 10^4$

- Calculation time of all eigenvalues and eigenvectors typically scales as $\sim N^3$ - can be expensive!

- Often only few eigenvalues are needed *(e.g. only localized states)*

➡ *More efficient method can be used – the so-called Power method*

Finding the largest eigenvalue: the Power method

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- Eigen-vectors of an $N \times N$ matrix form a basis-set: any N −component vector can be represented as a linear superposition of eigen-vectors:

$$\vec{x} = \sum_i^N a_i \vec{e}_i$$

- Calculate $\hat{M}\vec{x} = \hat{M}\left(\sum_i^N a_i \vec{e}_i\right) = \sum_i^N a_i \hat{M}\vec{e}_i = \sum_i^N a_i \lambda_i \vec{e}_i$

- Repeat many (R) times: $\hat{M}^R \vec{x} = \sum_i^N a_i (\lambda_i)^R \vec{e}_i$

Finding the largest eigenvalue: the Power method

$$\hat{M}\vec{e}_i = \lambda_i\vec{e}_i$$

$$\hat{M}^R\vec{x} = \sum_i^N a_i(\lambda_i)^R\vec{e}_i$$

- As $R \to \infty$ the term with the largest (by modulus) eigenvalue λ_m will dominate in this sum:

$$|(\lambda_m)^R| \gg |(\lambda_i)^R|$$

- And therefore, for large R: $\boxed{\hat{M}^R\vec{x} \approx a_m(\lambda_m)^R\vec{e}_m}$

➡ Any arbitrary initial guess $\vec{x}$ should rotate towards the eigenvector $\vec{e}_m$ as $R \to \infty$
(as long as it has a non-zero projection onto $\vec{e}_m$, i.e. $a_m \neq 0$)

Finding the largest eigenvalue: the Power method

$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$

$$\boxed{\hat{M}^R \vec{x} \approx a_m (\lambda_m)^R \vec{e}_m}$$

- BUT for $R \to \infty$:

$|(\lambda_m)^R| \to \infty$ If $|\lambda_m| > 1$

$|(\lambda_m)^R| \to 0$ If $|\lambda_m| < 1$

Will have problems, as $\hat{M}^R \vec{x}$ will either diverge, or decay to zero!

Can implement normalization at each iteration $\vec{x} \to \vec{x}/|\vec{x}|$

Finding the largest eigenvalue: the Power method

$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$

$$\boxed{\hat{M}^R \vec{x} \approx a_m (\lambda_m)^R \vec{e}_m}$$

- To find the largest eigen-value (and the eigen-vector):

  1. Start with an arbitrary guess $\overrightarrow{x_1}$

     (the closer your guess is to the actual eigen-vector, the faster the scheme will converge)

  2. Calculate $\overrightarrow{x_2'} = \hat{M}\vec{x}_1$
  3. Perform normalization $\vec{x}_2 = \vec{x}_2'/|\vec{x}_2'|$
  4. Repeat steps 2 and 3 for R iterations

For large R, $\vec{x}_R$ **will asymptotically approach the eigen-vector $\vec{e}_m$,** and $\vec{x}'_{R+1} = \hat{M}\vec{x}_R \to \lambda_m \vec{x}_R$

Can obtain the eigenvalue as $\boxed{\lambda_m = \vec{x}'_{R+1} \cdot \vec{x}_R}$

Note: since $\vec{x}_R$ is normalised, $\vec{x}_R \cdot \vec{x}_R = 1$

Finding the largest eigenvalue: the Power method

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- **An example:**

The stability matrix of the Leapfrog iteration scheme for decay equation (as derived in Lecture 8) is

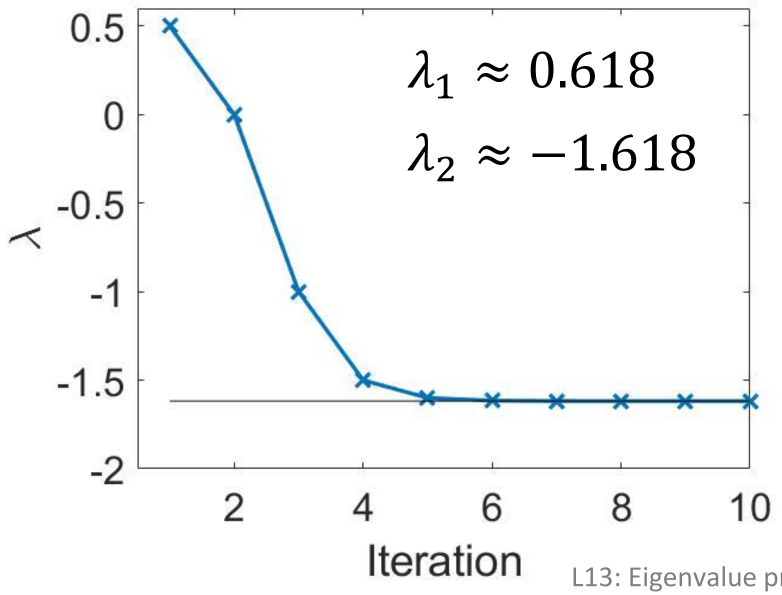
$$\hat{M} = \begin{bmatrix} -2\lambda a & 1 \\ 1 & 0 \end{bmatrix}$$

The eigenvalues are: $\lambda_{1,2} = -\lambda a \pm \sqrt{(\lambda a)^2 + 1}$

$\boldsymbol{\lambda a = 0.5}$

Let's obtain the largest eigenvalue via iterations!

Start iterations with $\vec{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$



Finding the largest eigenvalue: the Power method

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- **An example:**

The stability matrix of the Leapfrog iteration scheme for decay equation (as derived in Lecture 8) is

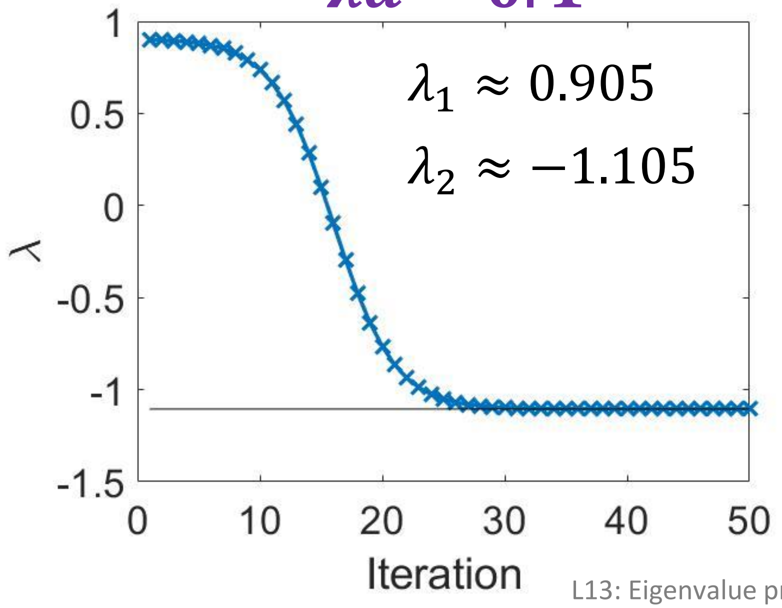
$$\hat{M} = \begin{bmatrix} -2\lambda a & 1 \\ 1 & 0 \end{bmatrix}$$

The eigenvalues are: $\lambda_{1,2} = -\lambda a \pm \sqrt{(\lambda a)^2 + 1}$

Let's obtain the largest eigenvalue via iterations!

Start iterations with $\vec{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

$\boldsymbol{\lambda a = 0.1}$



Finding an eigenvalue: the Power method

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- Finding the largest eigenvalue is not always useful. But with little modifications, the Power method can be used to find an eigenvalue closest to an arbitrary given number λ_0

- For this, we can construct the matrix

$$\hat{M}' = \hat{M} - \lambda_0 \hat{I}$$

- All eigenvalues of $\hat{M}'$ will be related to eigenvalues of the original matrix $\hat{M}$ as

$$\lambda_i' = \lambda_i - \lambda_0$$

- In particular, the eigenvalue λ_c closest to λ_0 will correspond to **the smallest (by magnitude) eigenvalue of $\hat{M}'$**

The Power method: Inverse iterations

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- If we apply similar iterations

$$(\hat{M}')^R \vec{x} = \sum_i^N a_i (\lambda_i - \lambda_0)^R \vec{e}_i$$

The term with the closest to λ_0 eigenvalue will decay faster than anything else… this is not very useful!

- But if we make iterations with the inverse matrix, the situation will be the exact opposite

$$\left[\left(\hat{M}'\right)^{-1}\right]^R \vec{x} = \sum_i^N a_i \frac{1}{(\lambda_i - \lambda_0)^R} \vec{e}_i$$

Now, the term with the closest to λ_0 eigenvalue will be the largest!

The Power method: Inverse iterations

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

$$\boxed{\left[\left(\hat{M} - \lambda_0 \hat{I}\right)^{-1}\right]^R \vec{x} \approx a_c (\lambda_c - \lambda_0)^{-R} \vec{e}_c}$$

- To find the eigenvalue λ_c closest to a given value λ_0:

1. Start with an arbitrary guess $\overrightarrow{x_1}$
2. Calculate the iteration matrix $\hat{J} = \left(\hat{M} - \lambda_0 \hat{I}\right)^{-1}$

   Matrix inversion can be resource-demanding, but you only need to do it once

3. Calculate $\overrightarrow{x_2'} = \hat{J}\vec{x}_1$
4. Perform normalization $\vec{x}_2 = \vec{x}_2'/|\vec{x}_2'|$
5. Repeat steps 3 and 4 for R iterations

For large R, **$\vec{x}_R$ will asymptotically approach the eigen-vector $\vec{e}_c$,**

Can obtain the eigenvalue as

$$\boxed{\lambda_c = \lambda_0 + \frac{1}{\vec{x}'_{R+1} \cdot \vec{x}_R}}$$

The Power method: Inverse iterations

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

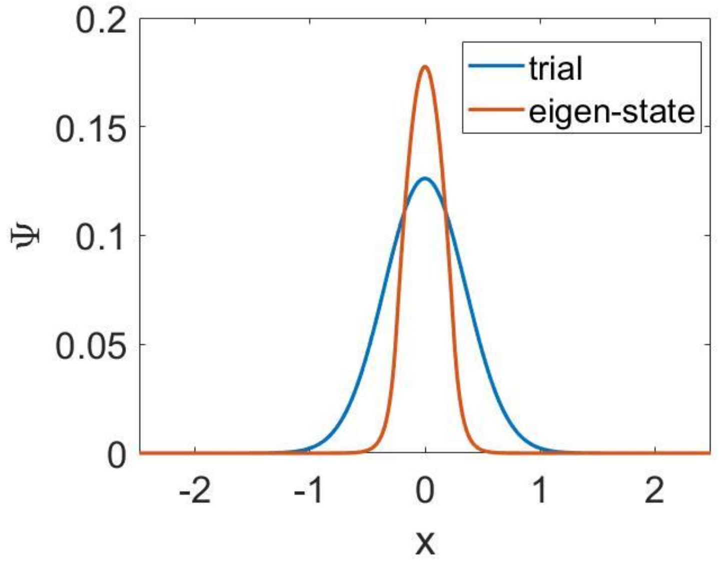
- **An example:** Localized levels in a quantum well (discussed on Lecture 6)

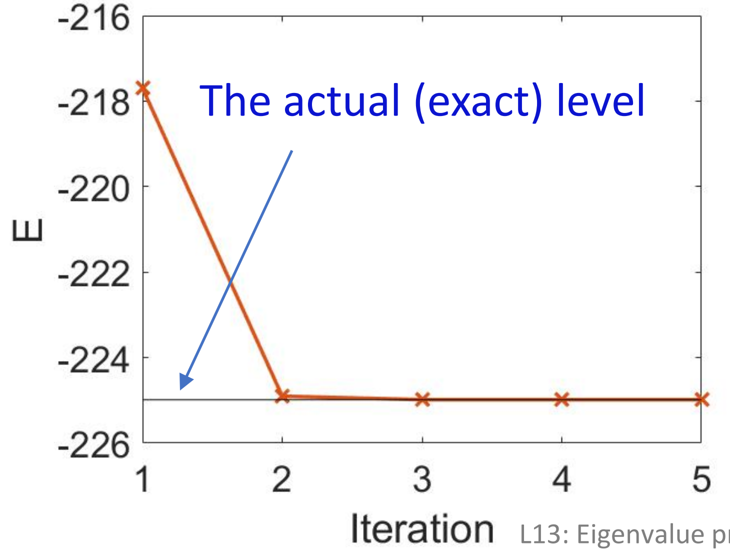
original

discretized

$$-\frac{d^2\Psi}{d\xi^2} + V(\xi)\Psi = E\Psi, \quad \Rightarrow \quad -\frac{1}{a^2}(\Psi_{j+1} + \Psi_{j-1} - 2\Psi_j) + V_j\Psi_j = E\Psi_j,$$

- For the fundamental mode (lowest eigen-value) can try to search for a level near the bottom of the quantum well ($E_0 = V_0 = -250$)
- To speed up, can start with a localized trial function





The Power method: Inverse iterations

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- **An example:** Localized levels in a quantum well (discussed on Lecture 6)

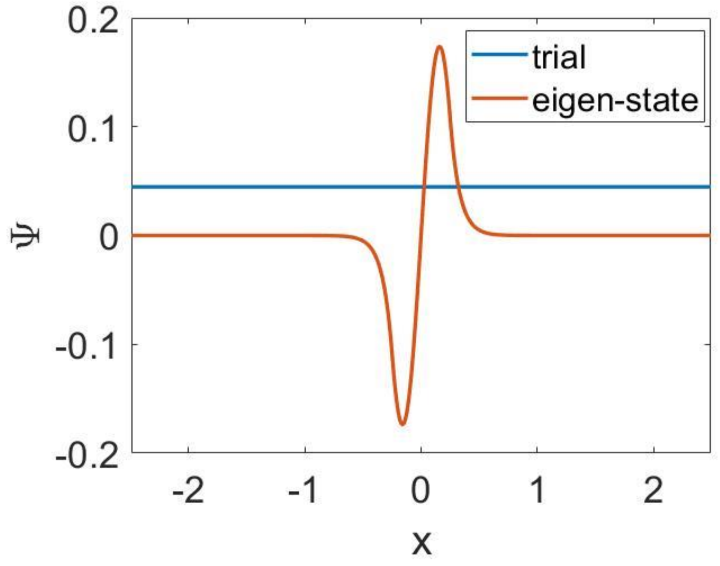
original

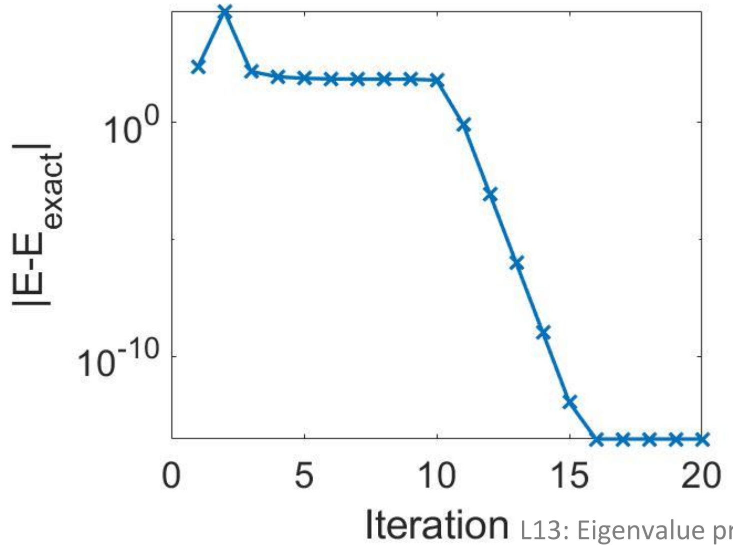
$$-\frac{d^2\Psi}{d\xi^2} + V(\xi)\Psi = E\Psi, \quad \Rightarrow$$

discretized

$$-\frac{1}{a^2}(\Psi_{j+1} + \Psi_{j-1} - 2\Psi_j) + V_j\Psi_j = E\Psi_j,$$

- Can try to find the next excited level… (use $E_0 = -150$)
- Will converge even with a simple trial function $\psi_{ini}(x) = 1$





The Power method

$$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- Works well when you only need to find a few eigen-values (and corresponding eigen-vectors)

- Requires iterations with matrix multiplication. For a general matrix the computational time scales as ~ $O(N^2)$.

- Often we deal with **sparce matrices** (e.g. tri-diagonal, due to CDA and FDA/BDA approximations). Matrix products are **computed much faster** in such cases.

- If you require all eigenvalues and/or eigenvectors (e.g. for basis-set expansions, or detailed stability analysis), you will have to use other (much slower) algorithms.

Full eigenvalue problem

$$\widehat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- If only eigenvalues are required, can try to solve the secular equation

$$\left|\widehat{M} - \lambda\hat{I}\right| = 0$$

  - Arrive to the problem of finding all roots of an Nth order polynomial
  - Only feasible if N is small

- More general approach is to find a transformation which diagonalises the matrix $\widehat{M}$, i.e. to represent $\widehat{M}$ as:

$$\widehat{M} = \hat{S}^{-1}\widehat{D}\hat{S}$$

where $\widehat{D}$ is a diagonal matrix

Full eigenvalue problem: Diagonalisation

$$\boxed{\hat{M}\vec{e}_i = \lambda_i \vec{e}_i}$$

- A diagonal matrix

$$\hat{D} = \begin{pmatrix} \lambda_1 & & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_N \end{pmatrix}$$

has eigenvalues $\lambda_1, \lambda_2, \ldots \lambda_N$, and corresponding N eigen-vectors:

$$\vec{d}_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ \vdots \\ 0 \end{pmatrix}, \vec{d}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \vec{d}_N = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}$$

$\hat{M}\vec{e}_i = \lambda_i \vec{e}_i$

Full eigenvalue problem: Diagonalisation

- For a generic $N \times N$ matrix: $\boxed{\hat{M} = \hat{S}^{-1}\hat{D}\hat{S}}$
- Eigenvalue problem: $\hat{S}^{-1}\hat{D}\hat{S}\vec{e}_i = \lambda_i \vec{e}_i$
- Multiply both sides by $\hat{S}$: $\hat{D}(\hat{S}\vec{e}_i) = \lambda_i(\hat{S}\vec{e}_i)$

➡ 1) All eigen-values λ_i of the diagonal matrix $\hat{D}$ are the same as of the original matrix $\hat{M}$

2) Eigen vectors of the diagonal matrix are $\hat{S}\vec{e}_i$. In other words:

$$\vec{e}_i = \hat{S}^{-1}\vec{d}_i = \hat{S}^{-1} \cdot \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

i-th element is 1, the rest are zero

i-th row of the $\hat{S}^{-1}$ matrix is the i-th eigen vector $\vec{e}_i$ of the original matrix $\hat{M}$

Full eigenvalue problem: Diagonalisation

$$\widehat{M}\vec{e}_i = \lambda_i \vec{e}_i$$

- For a generic $N \times N$ matrix: $\boxed{\widehat{M} = \hat{S}^{-1}\widehat{D}\hat{S}}$
- If we find the corresponding matrices $\hat{S}$ and $\widehat{D}$, we have full information about the eigen-values and eigen-vectors
- Various algorithms have been developed and implemented in linear algebra libraries, available for different languages
- Your knowledge of properties of $\widehat{M}$, can significantly speedup the process!

E.g. find all 200 eigenvalues of a real symmetric (dense) matrix using NAG library:

F02AKF	complex	all λ, $\underline{x}$	1.0
F02AJF	complex	λ only	0.461
F02AXF	hermitian	all λ, $\underline{x}$	0.539
F02AWF	Hermitian	λ only	0.256
F02ABF	real symm	all λ, $\underline{x}$	0.246
F02AAF	real symm	λ only	0.051

Summary:

- The computational time for the general problem of finding all eigenvalues and eigenvectors scales as $\sim N^3$ - expensive!

- For typical physics problems, the matrix is Hermitian or real symmetric – this saves a lot of effort.

- In many cases you only need few eigenvalues. The Power method is very efficient in such situations. Computational time scales as

  $\sim N^2$ for generic matrices
  $\sim N$ for sparse matrices

Lecture 14:

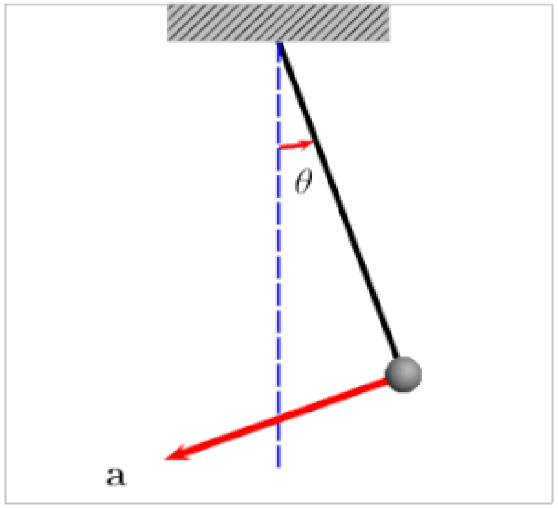
Simultaneous nonlinear algebraic equations

An example: Pendulum equation

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0$$

De-dimensionalize: $t = \sqrt{l/g}\,\tau$

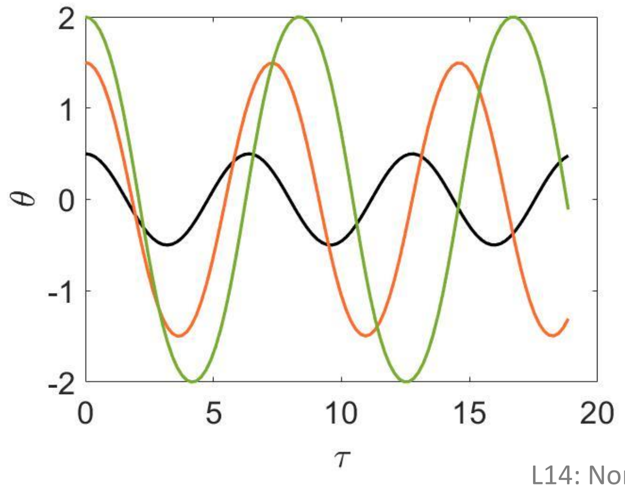
$$\Rightarrow \boxed{\frac{d^2\theta}{d\tau^2} + \sin\theta = 0}$$



https://en.wikipedia.org/wiki/Pendulum_(mathematics)

- Can solve as Initial Value Problem (IVP) using e.g. initial condition $\theta(0) = \theta_0,\ \dot\theta(0) = 0$

$$\begin{cases} \dot\theta = \omega \\ \dot\omega = -\sin(\theta) \end{cases}$$
$$\theta(0) = \theta_0, \omega(0) = 0$$



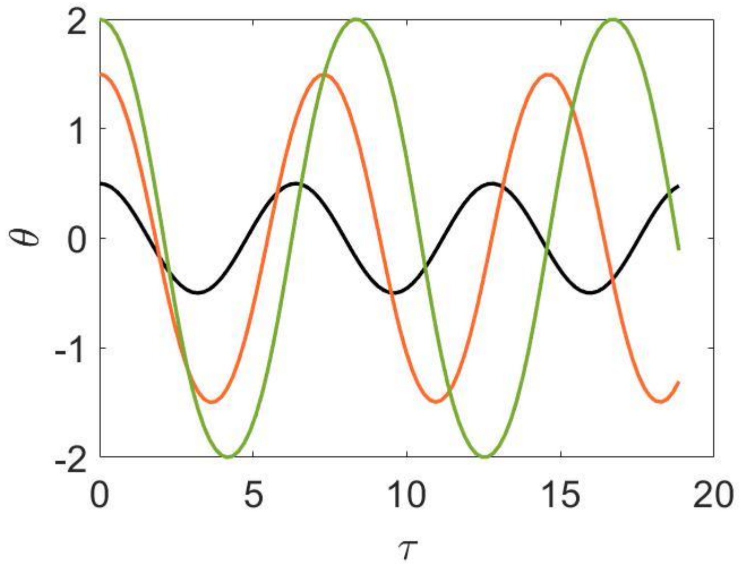
An example: Pendulum equation

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

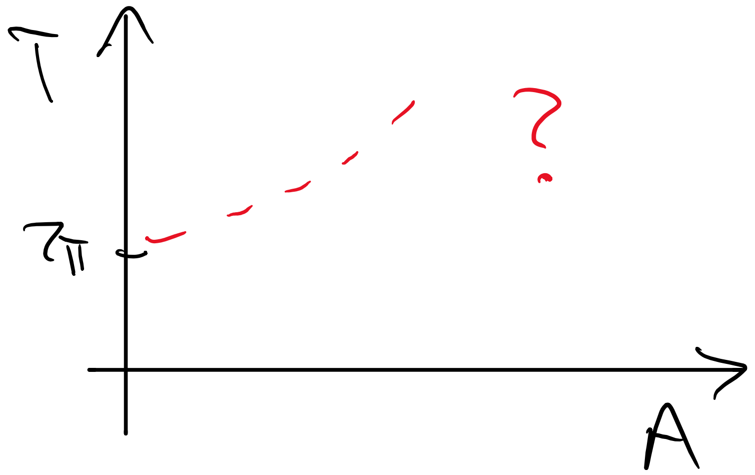
- Small-amplitude limit:

$$|\theta| \ll 1 \quad \Rightarrow \quad \sin\theta \approx \theta$$

$$\ddot{\theta} + \theta = 0$$
$$\theta(\tau) = A\cos\tau\,,$$
$$\theta(\tau + 2\pi) = \theta(\tau)$$



- An observation: as amplitude increases, the period seem to be increasing as well

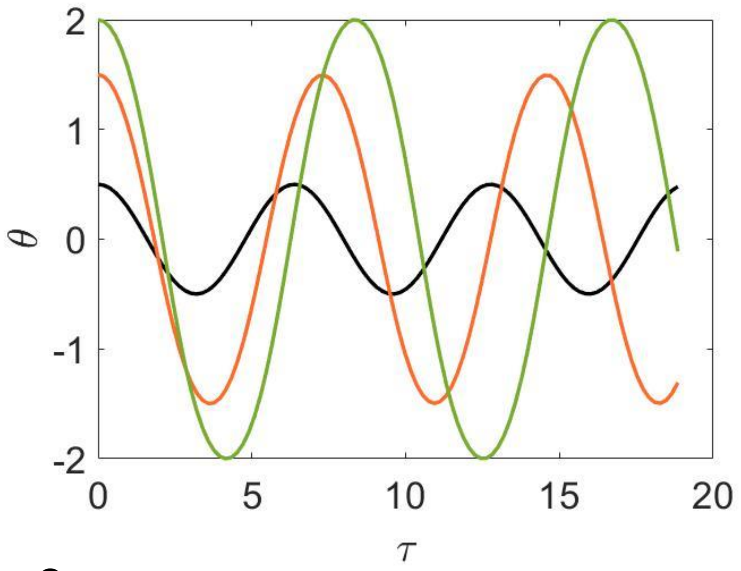


- Can try to obtain a relationship between the amplitude and the period $T(A)$?

An example: Pendulum equation

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

- Can attempt to solve the Boundary Value Problem (BVP) to obtain $\theta(\tau)$ profile, assuming a periodic function.



1) Discretise:

$$\frac{1}{a^2}(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \sin\theta_j = 0,$$

2) Fix period T, solve for $\theta(\tau)$

=> Obtain A(T), but this can be easily reverted for T(A)

3) Boundary conditions?

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

An example: Pendulum equation

Variant 1: solve in the **full period window**, $0 \le \tau \le T$
assume **periodic boundary conditions**

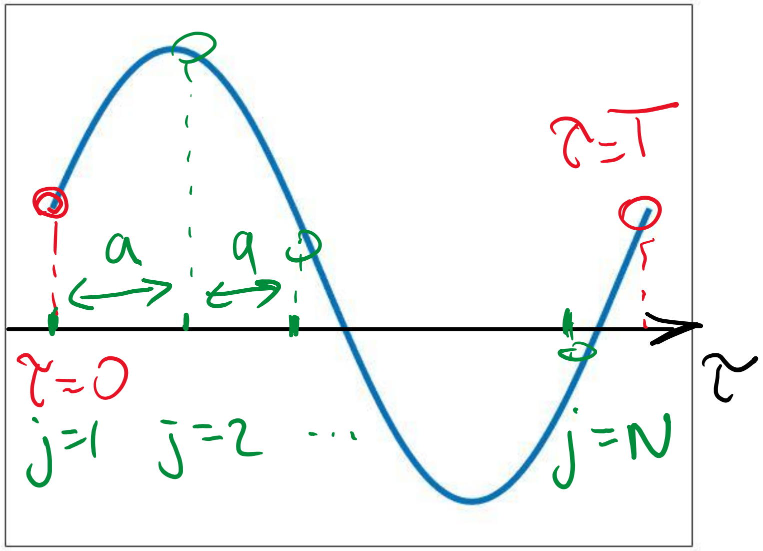
$$\theta(T) = \theta(0),$$

Discretise:

$\tau = 0$	$j = 1$
$\tau = a$	$j = 2$
…	
$\tau = T - a$	$j = N$

$$\boxed{a = T/N}$$

Note: *do not need $\tau = T$ point because of periodic BC!*



Full set of equations:

$$\frac{1}{a^2}(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \sin\theta_j = 0, \qquad j = 2,3,\ldots N-1$$

$$\frac{1}{a^2}(\theta_2 + \boldsymbol{\theta_N} - 2\theta_1) + \sin\theta_1 = 0, \qquad j = 1$$

$$\frac{1}{a^2}(\boldsymbol{\theta_1} + \theta_{N-1} - 2\theta_N) + \sin\theta_N = 0, \qquad j = N$$

An example: Pendulum equation

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

Variant 2: solve in the **full period window**, $0 \le \tau \le T$

assume **fixed $\theta = 0$ at $\tau = 0$**

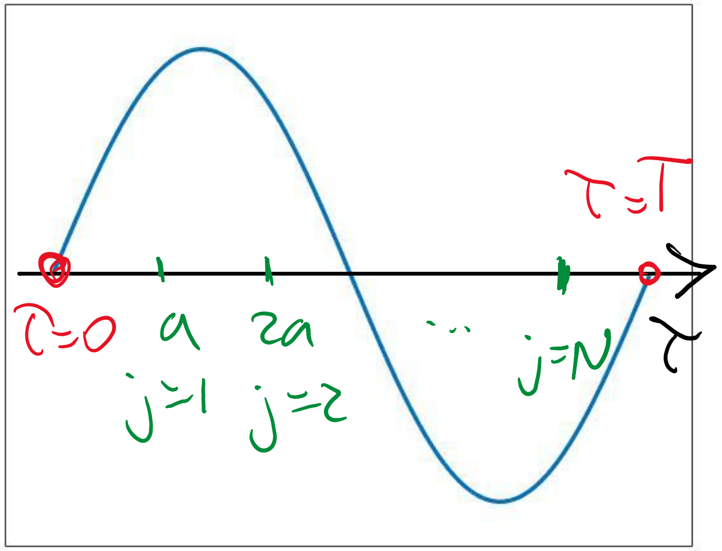
$$\theta(T) = \theta(0) = 0$$

Discretise:

$\tau = a$	$j = 1$
$\tau = 2a$	$j = 2$
…	
$\tau = T - a$	$j = N$

$$\boxed{a = T/(N+1)}$$

Note: *do not need $\tau = 0$ and $\tau = T$ points!*



Full set of equations:

$$\frac{1}{a^2}(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \sin\theta_j = 0, \qquad j = 2,3,\dots N-1$$

$$\frac{1}{a^2}(\theta_2 + \mathbf{0} - 2\theta_1) + \sin\theta_1 = 0, \qquad j = 1$$

$$\frac{1}{a^2}(\mathbf{0} + \theta_{N-1} - 2\theta_N) + \sin\theta_N = 0, \qquad j = N$$

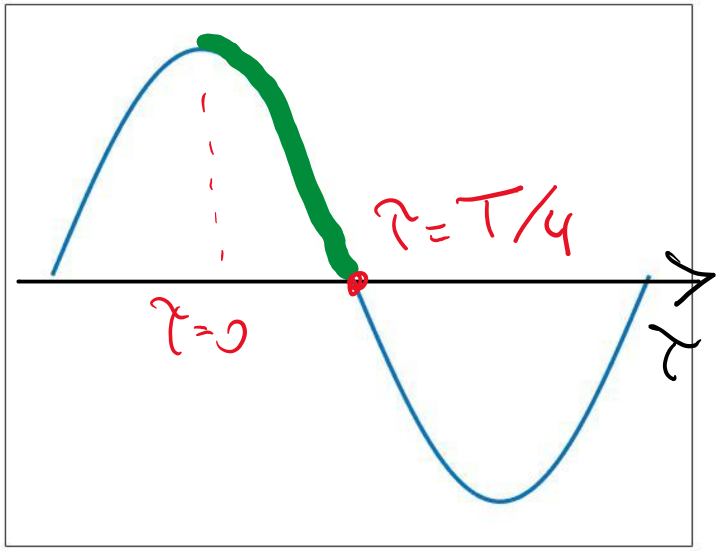
$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

An example: Pendulum equation

Variant 3: solve in the **quarter-period window**, $0 \le \tau \le T/4$

assume **max θ at $\tau = 0$** and **$\theta = 0$ at $\tau - T/4$**

$$\dot{\theta}(0) = 0, \qquad \theta(T/4) = 0$$



Discretise:

$\tau = 0$	$j = 1$
$\tau = a$	$j = 2$
…	
$\tau = T/4 - a$	$j = N$

$$\boxed{a = T/(4N)}$$

Note: *do not need $\tau = T/4$ point*

Full set of equations:

$$\frac{1}{a^2}(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \sin\theta_j = 0, \qquad j = 2,3, \dots N-1$$

$$\frac{1}{a^2}(\theta_2 + \boldsymbol{\theta_2} - 2\theta_1) + \sin\theta_1 = 0, \qquad j = 1 \quad \leftarrow \frac{\partial\theta}{\partial\tau}(0) = 0$$

$$\frac{1}{a^2}(\mathbf{0} + \theta_{N-1} - 2\theta_N) + \sin\theta_N = 0, \qquad j = N \quad \leftarrow \theta(T/4) = 0$$

An example: Pendulum equation

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

Need to solve:

$$\begin{cases} (2\theta_2 - 2\theta_1) + a^2 \sin\theta_1 = 0, & j = 1 \quad = \boldsymbol{F_1(\theta_1, \theta_2, \dots, \theta_N)} \\ (\theta_3 + \theta_1 - 2\theta_2) + a^2 \sin\theta_2 = 0, & j = 2 \quad = \boldsymbol{F_2(\theta_1, \theta_2, \dots, \theta_N)} \\ \dots & \\ (\theta_{j+1} + \theta_{j-1} - 2\theta_j) + a^2 \sin\theta_j = 0, & \\ \dots & \\ (\theta_{N-1} - 2\theta_N) + a^2 \sin\theta_N = 0. & j = N \quad = \boldsymbol{F_N(\theta_1, \theta_2, \dots, \theta_N)} \end{cases}$$

This is a system of coupled nonlinear algebraic equations

$$\boxed{\vec{F}(\vec{x}) = 0}$$

where $\vec{x} = [\theta_1, \theta_2, \dots, \theta_N]^T$ - vector of unknowns

$\vec{F}(\vec{x})$ - vector function, contains all left-hand sides of the equations

Systems of nonlinear equations

$$\boxed{\vec{F}(\vec{x}) = 0}$$

- There are no exact methods, can only use iterative schemes

- Can have no solution

E.g. for the pendulum there is no solution for $T < 2\pi$

Knowledge of the physics of the system is a useful guide

(Or you could try to prove mathematically that a solution exists…)

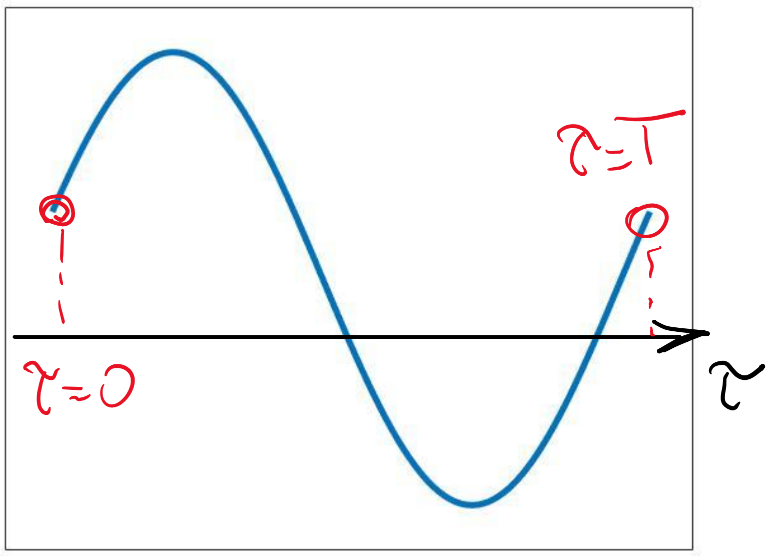
- Can have multiple solutions

E.g. for periodic BC $\theta(t+T) = \theta(t)$

If a solution $\theta^{(0)}(t)$ exists

then any $\theta^{(1)}(t) = \theta^{(0)}(t+\alpha)$ is also a solution!

➡ ***will have problems with convergence!***



Systems of nonlinear equations

$$\boxed{\vec{F}(\vec{x}) = 0}$$

- There are no exact methods, can only use iterative schemes
- Can have no solution

  Knowledge of the physics of the system is a useful guide

  (Or you could try to prove mathematically that a solution exists…)

- Can have multiple solutions

  - **need to be aware of possible degeneracies** and setup the problem accordingly

  again, knowledge of the physics helps you here

  - **need a good initial guess $\vec{x}_0$**, such that iterations converge to the desired solution

  e.g. start with parameter values where an approximate solution is known

E.g. for the pendulum small amplitude oscillations have $T \approx 2\pi$

Systems of nonlinear equations: simple iterations

$\vec{F}(\vec{x}) = 0$

- Could try to re-arrange the system in the form $\vec{x} = \vec{G}(\vec{x})$
- Use iterations $\boxed{\vec{x}_{k+1} = \vec{G}(\vec{x}_k)}$
- Convergence?

$$\vec{x}_k = \vec{x}^{(e)} + \vec{\epsilon}_k \quad \Rightarrow \quad \vec{x}^{(e)} + \vec{\epsilon}_{k+1} = \vec{G}\left(\vec{x}^{(e)} + \vec{\epsilon}_k\right)$$

- Need to use Taylor expansion to obtain an equation for $\vec{\epsilon}$

$$\vec{G}\left(\vec{x}^{(e)} + \vec{\epsilon}_k\right) \approx \vec{G}\left(\vec{x}^{(e)}\right) + \hat{J}_G\left(\vec{x}^{(e)}\right) \cdot \vec{\epsilon}_k$$

N-component vector

$N \times N$ matrix $\hat{J}_G$ is called Jacobian

N-component vector

Jacobian (vector-by-vector derivative)

$$\vec{F}(\vec{x}) = \begin{bmatrix} F_1(x_1, x_2, \dots, x_N) \\ F_2(x_1, x_2, \dots, x_N) \\ \dots \\ F_N(x_1, x_2, \dots, x_N) \end{bmatrix} \longrightarrow \hat{J}_F(\vec{x}) = \frac{\partial \vec{F}}{\partial \vec{x}} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \dots & \frac{\partial F_1}{\partial x_N} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \dots & \frac{\partial F_2}{\partial x_N} \\ \dots & \dots & \dots & \dots \\ \frac{\partial F_N}{\partial x_1} & \frac{\partial F_N}{\partial x_2} & \dots & \frac{\partial F_N}{\partial x_N} \end{bmatrix}$$

- Some useful identities:

  - if $\hat{A}$ is a constant matrix (i.e. its elements are numbers, not functions of x)

  then:

1) $$\frac{\partial(\hat{A}\vec{x})}{\partial \vec{x}} = \hat{A}$$

2) $$\frac{\partial\left(\hat{A}F(\vec{x})\right)}{\partial \vec{x}} = \hat{A}\frac{\partial \vec{F}}{\partial \vec{x}} = \hat{A} \cdot \hat{J}_F$$

Systems of nonlinear equations: simple iterations

$$\vec{F}(\vec{x}) = 0$$

- Could try to re-arrange the system in the form $\vec{x} = \vec{G}(\vec{x})$
- Use iterations $\boxed{\vec{x}_{k+1} = \vec{G}(\vec{x}_k)}$
- Convergence?

$$\vec{x}_k = \vec{x}^{(e)} + \vec{\epsilon}_k \quad \Rightarrow \quad \vec{x}^{(e)} + \vec{\epsilon}_{k+1} = \vec{G}\left(\vec{x}^{(e)} + \vec{\epsilon}_k\right)$$

$$\boxed{\vec{\epsilon}_{k+1} \approx \underset{\uparrow}{\hat{J}_G\left(\vec{x}^{(e)}\right)} \cdot \vec{\epsilon}_k}$$

- The Jacobian is a function of the exact solution $\vec{x}^{(e)}$, which is unknown
- Could try to estimate convergence using the trial solution $\vec{x}_0$ (assuming it is reasonably close to the exact solution)

Simple iterations: Version 1

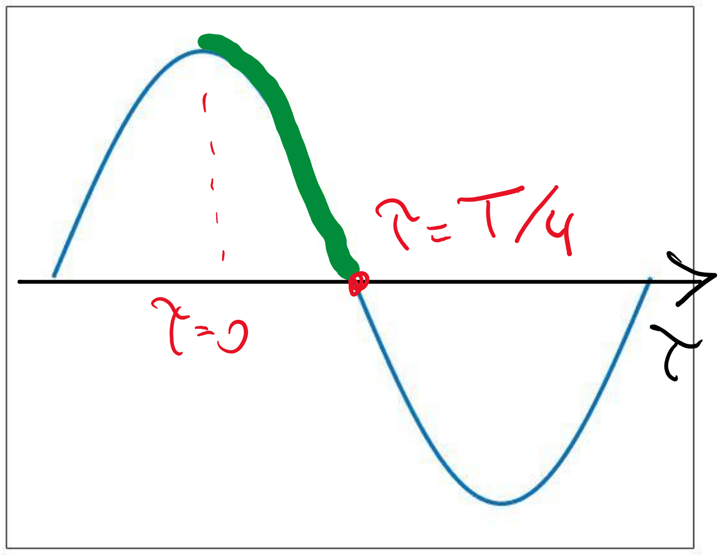
$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

$$(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \mathrm{a}^2 \sin\theta_j = 0$$

- Can re-arrange as:

$$\theta_j = \frac{1}{2}(\theta_{j+1} + \theta_{j-1} + \mathrm{a}^2 \sin\theta_j)$$

- Need a good initial guess



- Solve on the interval $0 < \tau < T/4$ with the boundary conditions

$$\dot\theta(0) = 0, \qquad \theta(T/4) = 0$$

- Could try $\theta_0(\tau) = A\cos(\tau \cdot 2\pi/T)$

Simple iterations: Version 1

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

- Iterations:

$$\theta_j = \frac{1}{2}(\theta_{j+1} + \theta_{j-1} + \mathrm{a}^2 \sin\theta_j) \Rightarrow \boxed{\vec{\theta}_{k+1} = \widehat{M_0}\vec{\theta}_k + \frac{a^2}{2}\sin(\vec{\theta}_k)}$$

$$\theta_0(\tau) = A\cos(\tau \cdot 2\pi/T)$$

$$\Rightarrow \quad \vec{\theta}_0 = 2\cos(\vec{\tau} \cdot 2\pi/T)$$

$$\boldsymbol{i.e.} \quad \vec{\theta}_0 = \begin{bmatrix} A\cos(0) \\ A\cos(2\pi a/T) \\ A\cos(4\pi a/T) \\ \cdots \\ A\cos(2\pi N a/T) \end{bmatrix}$$

- Convergence?

$$\hat{J} = \widehat{M}_0 + \frac{a^2}{2}\frac{\partial\left(\sin\vec{\theta}\right)}{\partial\vec{\theta}}$$

$$\sin\vec{\theta} = \begin{bmatrix} \sin(\theta_1) \\ \sin(\theta_2) \\ \cdots \\ \sin(\theta_N) \end{bmatrix} \qquad \frac{\partial\sin\vec{\theta}}{\partial\vec{\theta}} = \begin{bmatrix} \cos(\theta_1) & 0 & \cdots & 0 \\ 0 & \cos(\theta_2) & 0 & \cdots \\ \cdots & \cdots & \ddots & 0 \\ 0 & \cdots & 0 & \cos(\theta_N) \end{bmatrix}$$

Simple iterations: Version 1

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

- Iterations: $$\boxed{\vec{\theta}_{k+1} = \widehat{M_0}\vec{\theta}_k + \frac{a^2}{2}\sin(\vec{\theta}_k)}$$

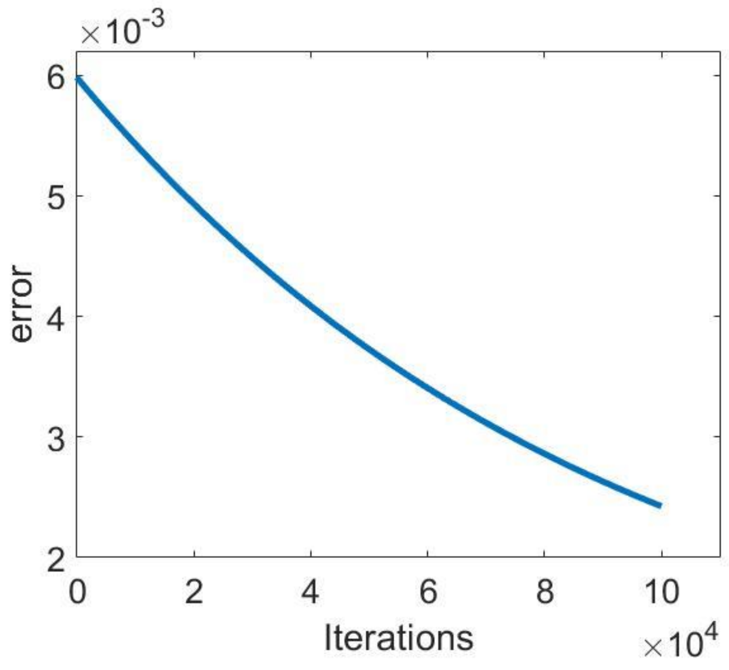
- Initial guess: $$\vec{\theta}_0 = \begin{bmatrix} A\cos(0) \\ A\cos(2\pi a/T) \\ A\cos(4\pi a/T) \\ \cdots \\ A\cos(2\pi N a/T) \end{bmatrix}$$

- **Convergence:**

$$\hat{J} = \widehat{M}_0 + \frac{a^2}{2}\frac{\partial(\sin\vec{\theta})}{\partial\vec{\theta}}$$

$$\rho = \max(|\lambda|) \approx 0.9999991$$

…is very poor!



Indeed!

Simple iterations: Version 2

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

$$(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + \mathrm{a}^2 \sin\theta_j = 0$$

- Can re-arrange

$$(2\theta_j - \theta_{j+1} - \theta_{j-1}) = \mathrm{a}^2 \sin\theta_j \quad \Rightarrow \quad \widehat{\mathrm{M}}_L \vec{\theta} = \mathrm{a}^2 \sin(\vec{\theta})$$

- Try iterations: $\boxed{\vec{\theta}_{k+1} = \mathrm{a}^2 \widehat{\mathrm{M}}_{\mathrm{L}}^{-1} \cdot \sin(\vec{\theta}_k)}$

- Convergence?

$$\hat{J} = a^2 \widehat{M}_L^{-1} \frac{\partial(\sin\vec{\theta})}{\partial\vec{\theta}} \qquad \rho = \max(|\lambda|) \approx 1.1 \cdot 10^{-6}$$

Should be very good!

Simple iterations: Version 2

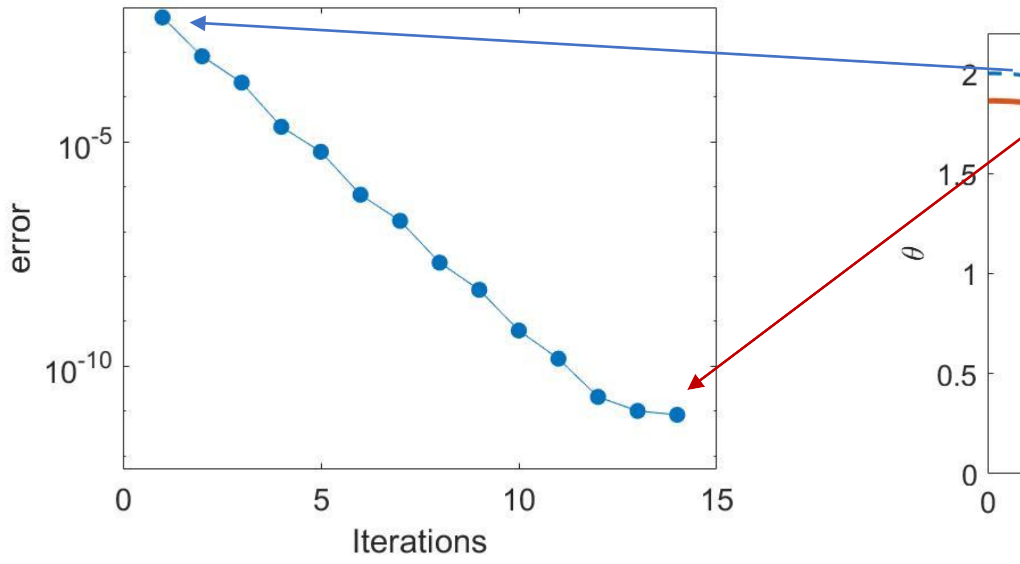
$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

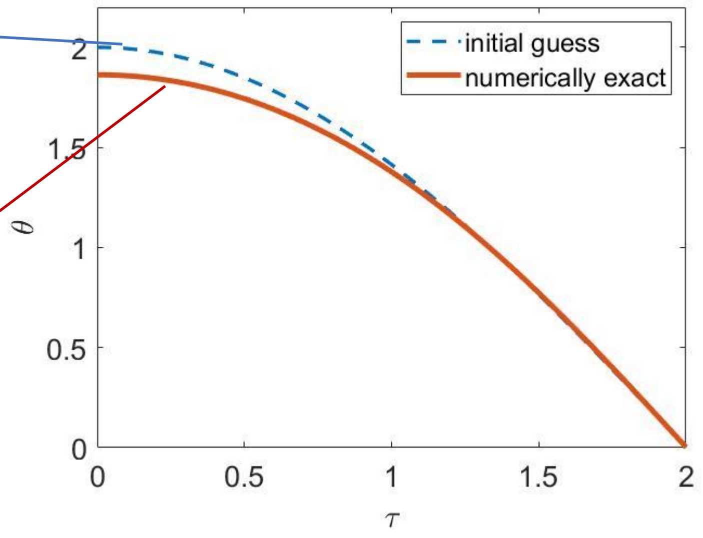
$$(\theta_{j+1} + \theta_{j-1} - 2\theta_j) + a^2 \sin\theta_j = 0$$

- Can re-arrange

$$(2\theta_j - \theta_{j+1} - \theta_{j-1}) = a^2 \sin\theta_j \Rightarrow \hat{M}_L \vec{\theta} = a^2 \sin(\vec{\theta})$$

- Try iterations: $\boxed{\vec{\theta}_{k+1} = a^2 \hat{M}_L^{-1} \cdot \sin(\vec{\theta}_k)}$





Indeed, convergence is good!

Newton-Raphson iterations

$\vec{F}(\vec{x}) = 0$

- At k-th iteration we have $\vec{F}(\vec{x}_k) \neq 0$, looking for a correction such that

$$\vec{F}(\vec{x}_{k+1} = \vec{x}_k + \Delta\vec{x}_k) = 0$$

- Assuming, we are close to the true solution (i.e. $|\Delta\vec{x}_k| \ll |\vec{x}_k|$), use Taylor expansion

$$\vec{F}(\vec{x}_k + \Delta\vec{x}_k) \approx \vec{F}(\vec{x}_k) + \hat{J}_F(\vec{x}_k) \cdot \Delta\vec{x}_k = 0$$

$$\Rightarrow \quad \hat{J}_F(\vec{x}_k) \cdot \Delta\vec{x}_k = -\vec{F}(\vec{x}_k)$$

This is a linear matrix problem for $\Delta\vec{x}_k$.

- Iteration scheme: $\boxed{\vec{x}_{k+1} = \vec{x}_k - \hat{J}_F{}^{-1}(\vec{x}_k)\vec{F}(\vec{x}_k)}$

Note: *Each derivative in the Jacobian is evaluated at* $\vec{x} = \vec{x}_k$
=> Need to update Jacobian at each iteration

- **Fast convergence:** $|\vec{\epsilon}_{k+1}| \sim |\vec{\epsilon}_k|^2$

Newton-Raphson iterations

$\vec{F}(\vec{x}) = 0$

$$\boxed{\vec{x}_{k+1} = \vec{x}_k - \hat{J}^{-1}(\vec{x}_k)\vec{F}(\vec{x}_k)}$$

- Iteration procedure:

  1) Start with an initial guess $\vec{x}_0$ (need to be reasonably close to the true solution)

  2) Calculate $\vec{\mathrm{F}}(\vec{x}_0)$

  3) Calculate $\hat{J}(\vec{x}_0)$

  4) Solve $\hat{J}(\vec{x}_0) \cdot \Delta\vec{x} = -\vec{F}(\vec{x}_0)$

  Note: *This is a standard matrix problem, matrix inversion is not necessarily the best approach. Can use different methods – see previous lecture*

  5) Calculate next value $\vec{x}_1 = \vec{x}_0 + \Delta\vec{x}$

  6) Repeat steps 2-5 until a desirable accuracy is achieved

- Solution of matrix problem is needed at each iteration. This is expensive, **but the convergence is extremely good!** You will normally need not more than 3-5 iterations…

Newton-Raphson iterations: the pendulum example

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

$$\boxed{\vec{x}_{k+1} = \vec{x}_k - \hat{J}^{-1}(\vec{x}_k)\vec{F}(\vec{x}_k)}$$

- Construct the Jacobian:

$$\begin{cases} (2\theta_2 - 2\theta_1) + a^2 \sin\theta_1 = 0, & j = 1 \quad = \boldsymbol{F_1(\theta_1, \theta_2, \dots, \theta_N)} \\ (\theta_3 + \theta_1 - 2\theta_2) + a^2 \sin\theta_2 = 0, & j = 2 \quad = \boldsymbol{F_2(\theta_1, \theta_2, \dots, \theta_N)} \\ \dots & \\ (\theta_{j+1} + \theta_{j-1} - 2\theta_j) + a^2 \sin\theta_j = 0, & \\ \dots & \\ (\theta_{N-1} - 2\theta_N) + a^2 \sin\theta_N = 0. & j = N \quad = \boldsymbol{F_N(\theta_1, \theta_2, \dots, \theta_N)} \end{cases}$$

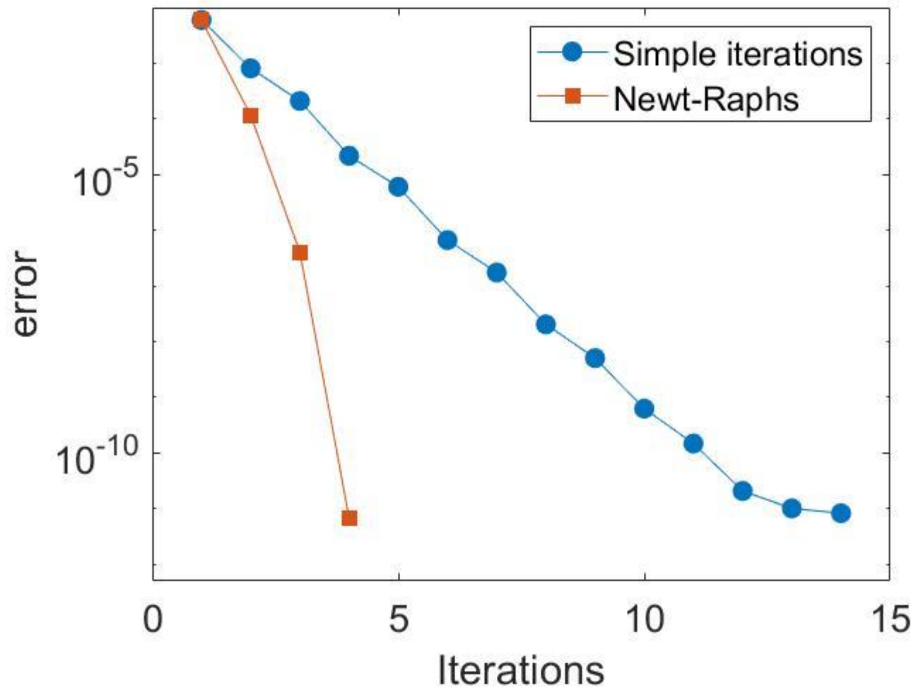
$$\Rightarrow \quad \hat{J}(\vec{\theta}) = \begin{bmatrix} -2 + a^2\cos\theta_1 & 2 & 0 & \cdots & \cdots & 0 \\ 1 & -2 + a^2\cos\theta_2 & 1 & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & 0 & 1 & -2 + a^2\cos\theta_N \end{bmatrix}$$

Newton-Raphson iterations: the pendulum example

$$\frac{d^2\theta}{d\tau^2} + \sin\theta = 0$$

$$\boxed{\vec{x}_{k+1} = \vec{x}_k - \hat{J}^{-1}(\vec{x}_k)\vec{F}(\vec{x}_k)}$$

- Results:



Summary:

- When dealing with nonlinear equations, a special care need to be taken to account for:

  existence of a solution;

  any possible degeneracies (such as time-shift or phase-shift symmetries);

  a good initial guess (e.g. a parameter regime where an approximate solution is known).

- Simple iterative schemes can work, but need to consider convergence.

- Often the default method would be Newton-Raphson

  Very good convergence;

  But slow: requires calculation of Jacobian and linear matrix problem solution at each step!

  If you have a (semi-)analytical form for the Jacobian, this can speed up the process significantly!