Numerical Techniques

Amal Medhi

School of Physics
IISER Thiruvananthapuram

Floating point numbers

- In computers, numbers are represented in binary format.
- The *floating point numbers* in computer are used to represent the real number system in mathematics.
- But since computer allocates only a *finite number of bits* to represent a number, it introduces an inherent approaximation.
- As a result, the floating point number system has certain peculiarities not present in real number system.
- The rounding error that is inevitable in representing a real number as a floating point number is generally small. *But successive operations on such numbers can magnify microscopic error to macroscopic size*.
- Here we will discuss some of the subleties associated with this approaximate representation of real numbers in computers.

IEEE format for floating point numbers

- Floating point representation of real numbers is similar to the *scientific notation* of real numbers.
- In the past, several formats were used for representation of floating point numbers in computers.
- The IEEE format is now a standard adopted by all computers today.

IEEE format for floating point numbers

- Floating point representation of real numbers is similar to the *scientific notation* of real numbers.
- In the past, several formats were used for representation of floating point numbers in computers.
- The IEEE format is now a standard adopted by all computers today.
- In the standard, format of a ***normalized*** floating point number is

$$\pm 1.b_1 b_2 b_3 \ldots \times 2^E$$

- The standard define three levels of precision for floating point numbers: ***single precision, double precision, and extended double precision***.
- Number of bits allocated for these three levels are 32 bits, 64 bits, and 80 bits respectivley.

IEEE format for floating point numbers

- The bits are divided among parts as follows:

precision	sign	exponent	mantissa
single	1	8	23
double	1	11	52
long double	1	15	64

IEEE format for floating point numbers

- The bits are divided among parts as follows:

precision	sign	exponent	mantissa
single	1	8	23
double	1	11	52
long double	1	15	64

The machine epsilon ϵ_M

- Consider the ***single precision*** numbers. Here number of bits for the exponent & the mantissa are $M = 8$ and $N = 23$, respectively.
- The **normalized** representation of the number 1.0_{10} is

$$+1.\boxed{000\ 000\ 000\ 000\ 000\ 000\ 000\ 00} \times 2^0$$

- The next floating point number **greater than** 1.0_{10} is

$$+1.\boxed{000\ 000\ 000\ 000\ 000\ 000\ 000\ 01} \times 2^0$$

which is $1.0 + 2^{-23}$.

IEEE format for floating point numbers

The machine epsilon ϵ_M

- The *machine epsilon* ϵ_M is defined as the distance between 1.0 and the smallest floating point number greater than 1.0.
- For *single precision* ($N = 23$) numbers: $\epsilon_M = 2^{-23}$.
- For *double precision* ($N = 52$) numbers: $\epsilon_M = 2^{-52}$.
- For *long double precision* ($N = 64$) numbers: $\epsilon_M = 2^{-64}$.

IEEE format for floating point numbers

The machine epsilon ϵ_M

- The *machine epsilon* ϵ_M is defined as the distance between 1.0 and the smallest floating point number greater than 1.0.
- For *single precision* ($N = 23$) numbers: $\epsilon_M = 2^{-23}$.
- For *double precision* ($N = 52$) numbers: $\epsilon_M = 2^{-52}$.
- For *long double precision* ($N = 64$) numbers: $\epsilon_M = 2^{-64}$.

Rounding

- Consider the binary representation of number 9.4.
  $9.4_{10} = 1001.0110011001100110\ldots\ldots$
  In **normalized** form,

$$9.0_{10} = +1.\boxed{001\ 0110\ 0110\ 0110\ 0110\ 0110}\ 0110\ 0110\ldots \times 2^3$$

- In *single precision*, we have space for only 23 bits (inside the box) for the mantissa. The integral part, '1' is not stored as it is understood that it is there.

Rounding error

- In such cases as above, we have to either discard all the bits from 24th onwards or do some rounding.
- The IEEE standard is to round the number *to its nearest vlaue* as follows.
  - Add 1 to the bit-23 if the bit-24 is 1 (round up), do nothing (simply truncate) if bit-24 is 0 (round down).
- Applying the *round-to-the-nearest* rule, the number $x = 9.4$ in single precision would be represented by the number $fl(x)$ given by

$$fl(9.4) = +1.\boxed{001\ 0110\ 0110\ 0110\ 0110\ 0110} \times 2^3$$

- The rounding error involved in this case is,

$$fl(9.4) - 9.4 = -0.\overline{0110} \times 2^{-23} \times 2^3 = -0.4 \times 2^{-20}$$

- The *absolute rounding error* is therefore,

$$|fl(9.4) - 9.4| = 0.4 \times 2^{-20}$$

- Thus we see that floating point number representation of a real number x may not be exactly x though very close.

IEEE format for floating point numbers

- A useful quantity to define is the *relative rounding error* given by

$$\delta = \frac{|fl(x) - x|}{|x|}, \quad \text{if } x \neq 0$$

- It turns out, the relative rounding error $\delta \leq \frac{1}{2}\epsilon_M$.

IEEE format for floating point numbers

- A useful quantity to define is the *relative rounding error* given by

$$\delta = \frac{|fl(x) - x|}{|x|}, \quad \text{if } x \neq 0$$

- It turns out, the relative rounding error $\delta \leq \frac{1}{2}\epsilon_M$.

Machine representation details

- Let us examine the actual machine representation of a normalized floating point number.

$$\pm 1.b_1 b_2 b_3 \ldots \times 2^E$$

- The bit field of a single precision number has the form

$$\boxed{s \big| a_1 a_2 \ldots a_8 \big| b_1 b_2 \ldots b_{23}}$$

- The sign bit s is 0 for +vs numbers and 1 for −ve numbers.

Machine representation

- Next 8 bits are used to represent the exponent value E.
  - This field does not actually store E but store $(b+E)$ where b is called the *exponent bias*. The bias for single precision numbers is $b = 2^{10} - 1 = 127$.
  - For normalized numbers, E can have values from -126 to +127. These are 254 values. The rest 2 possible values are reserved for special numbers.
  - Thus for the normalized numbers, possible values of $(b+E)$ are 1 to 254. The special values will have $(b+E) = 0$ and 255.
  - The special value 255 for $(b+E)$ is used to represent ∞ if the mantissa bit string are all zeros and *NaN* (not a number) otherwise.
  - The special value 0 for $(b+E)$ is used to represent the number 0 (***the most important number***) if the mantissa bit string are all zeros. But in this case E is intepreted as -126 not -127. More on this next.

Machine representation

- Next 8 bits are used to represent the exponent value E.
  - This field does not actually store E but store $(b+E)$ where b is called the *exponent bias*. The bias for single precision numbers is $b = 2^{10} - 1 = 127$.
  - For normalized numbers, E can have values from -126 to +127. These are 254 values. The rest 2 possible values are reserved for special numbers.
  - Thus for the normalized numbers, possible values of $(b+E)$ are 1 to 254. The special values will have $(b+E) = 0$ and 255.
  - The special value 255 for $(b+E)$ is used to represent ∞ if the mantissa bit string are all zeros and *NaN* (not a number) otherwise.
  - The special value 0 for $(b+E)$ is used to represent the number 0 (***the most important number***) if the mantissa bit string are all zeros. But in this case E is intepreted as -126 not -127. More on this next.

Subnormal numbers

- For the special exponent $(b+E) = 0$, with E interpreted as -126 (instead of -127), numbers have the following form called ***subnormal*** numbers,

$$\pm 0.b_1 b_2 \ldots b_2 3 \times 2^{-126}$$

Machine representation

- The reason E is intepreted this way is because it allows to represent numbers closer (on lower side) to the lowest +ve normalized numbers.
- This scheme of machine representation of single precision numbers is illustrated in the following table.
- Thus the **smallest normalized single precision** +ve number is $1.00000000000000000000000 \times 2^{-126}$ which is $\approx 1.2 \times 10^{-38}$. For double precision, it is $\approx 2.2 \times 10^{-308}$.
- Thus the **smallest subnormal single precision** +ve number is $0.00000000000000000000001 \times 2^{-126}$ which is $\approx 5.9 \times 10^{-39}$. For double precision, it is $\approx 1.1 \times 10^{-308}$.
- Note that this smallest number is different from machine precision ϵ_M. Numbers much smaller that ϵ_M can be represented in a computer, even though adding them to 1 may have no effect.

Machine representation

- This scheme of single precision machine represntation of floating point numbers is illustrated in the following table.

If exponent bitstring $a_1 \dots a_8$ is	Then numerical value represented is
$(00000000)_2 = (0)_{10}$	$\pm(0.b_1b_2b_3\dots b_{23})_2 \times 2^{-126}$
$(00000001)_2 = (1)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{-126}$
$(00000010)_2 = (2)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{-125}$
$(00000011)_2 = (3)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{-124}$
$\downarrow$	$\downarrow$
$(01111111)_2 = (127)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{0}$
$(10000000)_2 = (128)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{1}$
$\downarrow$	$\downarrow$
$(11111100)_2 = (252)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{125}$
$(11111101)_2 = (253)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{126}$
$(11111110)_2 = (254)_{10}$	$\pm(1.b_1b_2b_3\dots b_{23})_2 \times 2^{127}$
$(11111111)_2 = (255)_{10}$	$\pm\infty$ if $b_1 = \dots = b_{23} = 0$, NaN otherwise

Root finding

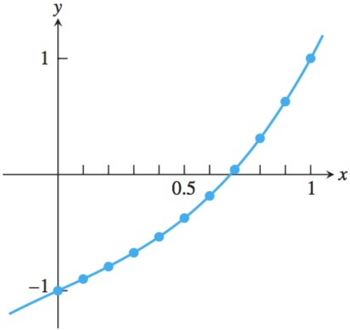
- **For a function $f(x)$, *find* r for which $f(r) = 0$.**

Root finding

- **For a function $f(x)$, *find* r for which $f(r) = 0$.**
- Does a root exist?
- **Theorem:** Let $f(x)$ be a *continous* function on $[a, b]$, satisfying $f(a)f(b) < 0$. Then there exist a number $r \in (a, b)$ such that $f(r) = 0$.

Root finding

- **For a function $f(x)$, *find* r for which $f(r) = 0$.**
- Does a root exist?
- **Theorem:** Let $f(x)$ be a *continous* function on $[a, b]$, satisfying $f(a)f(b) < 0$. Then there exist a number $r \in (a, b)$ such that $f(r) = 0$.
- Bracketing a root.

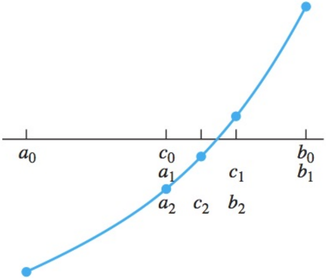


- Here root is bracketed in $[-1, 1]$ as $f(1)f(-1) < 0$.

Bisection method

Bisection method

- First find an initial interval $[a, b]$ which bracket the root.
- Pick the midpoint $c = \frac{a+b}{2}$. That is *bisect the interval*.
  - If $f(c) = 0 \rightarrow$ stop.
  - If $f(a)f(c) \rightarrow$ new bracket $[a, c]$.
  - If $f(c)f(b) \rightarrow$ new bracket $[c, b]$.



- Bisection Method

Bisection method

- **Accuracy:** A solution is **correct with p decimal places** if the error is less than 0.5×10^{-p}.
- For the bisection method, after n iterations:
  - The interval $[a_n, b_n]$ has the length $(b-a)/2^n$.
  - Best estimate for the solution r is $x_c = (a_n + b_n)/2$.
  - Solution error = $|x_c - r| = \frac{b-a}{2^{n+1}}$.
  - Function evaluations = $n + 2$.

Bisection method

- **Accuracy:** A solution is **correct with p decimal places** if the error is less than 0.5×10^{-p}.
- For the bisection method, after n iterations:
  - The interval $[a_n, b_n]$ has the length $(b-a)/2^n$.
  - Best estimate for the solution r is $x_c = (a_n + b_n)/2$.
  - Solution error = $|x_c - r| = \frac{b-a}{2^{n+1}}$.
  - Function evaluations = $n + 2$.
- **Question:** Find the root of $f(x) = (\cos x - x)$ in $[0, 1]$ to within **six** correct decimal places.
  - Error after n steps is $(b-a)/2^{n+1} = 1/2^{n+1}$.
  - We require $1/2^{n+1} < 0.5 \times 10^{-6}$. This means $n > \frac{6}{\log_{10} 2} \approx 19.9$.
  - We need $n = 20$ iterations to achieve the accuracy.

Forward error and backward error

- Consider finding the root of the following equation:

$$f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27} = 0$$

- The analytical answer is $r = 2/3 = 0.66666666\ldots$.
- Suppose numerically we want to find r to **six** significant digits.

Forward error and backward error

- Consider finding the root of the following equation:

$$f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27} = 0$$

- The analytical answer is $r = 2/3 = 0.66666666\ldots$.
- Suppose numerically we want to find r to **six** significant digits.
- A typical bisection calculation would stop iterating and declare the root to be $r = 0.6666641$ becuase numerically $f(0.6666641) = 0$.
- So we never get the answer correct to six decimal place in this case.

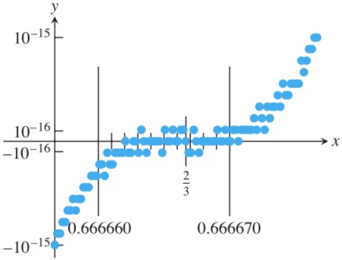
Forward error and backward error

- Consider finding the root of the following equation:

$$f(x) = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27} = 0$$

- The analytical answer is $r = 2/3 = 0.66666666\ldots$.
- Suppose numerically we want to find r to **six** significant digits.
- A typical bisection calculation would stop iterating and declare the root to be $r = 0.6666641$ becuase numerically $f(0.6666641) = 0$.
- So we never get the answer correct to six decimal place in this case.
- The reason is the nature of $f(x)$ near the root $r = 2/3$. There are many numbers near $r = 2/3$ where numerically $f(x) = 0$.
- Such things occur if the root is of *higher order.*

Forward error and backward error



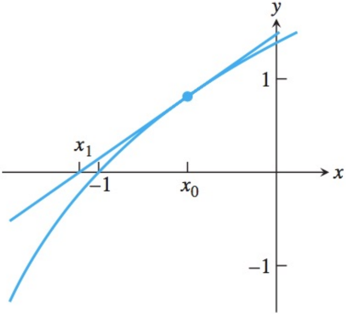
- Suppose r is the correct root and x_r is the numerical root.
- **Backward error** $= |f(x_r)|$ is the error in function value.
- **Forward error** $)x_r - r|$ is the error in the root value.
- A stopping criteria can be on either of these two.

Newton-Raphson method

- A method that needs derivative information. Usually converges faster than bisection.

Newton-Raphson method

- A method that needs derivative information. Usually converges faster than bisection.



- In Newton's method, to find the root of $f(x) = 0$, we start with guess solution x_0.
- Draw a tangent line to $f(x)$ at $x = x_0$. The tangent line will follow the $f(x)$ down to the axis towards the root.
- The intersection point of the line with the x-axis is the approximate root.
- The steps are repeated to get more closer to the answer.

Newton-Raphson method

- The equation of the tangent line is $y - f(x_0) = f'(x_0)(x - x_0)$.
- The intersection point with x-axis is obtained by putting $y = 0$.
- The next guess for the root is

$$x \equiv x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

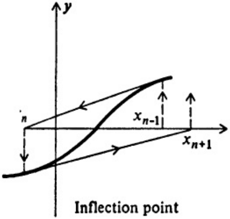
- One can repeat the steps and hope for convergence.

Newton-Raphson method

- The equation of the tangent line is $y - f(x_0) = f'(x_0)(x - x_0)$.
- The intersection point with x-axis is obtained by putting $y = 0$.
- The next guess for the root is

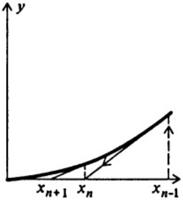
$$x \equiv x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- One can repeat the steps and hope for convergence.
- Potential problems:



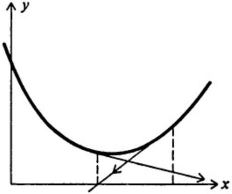
Inflection point

Can cycle and never converge



Multiple zero

Slow approach with $f' \to 0$ and trouble in division step



Local minimum

Risks being sent very far away for next approximation

Modified Newton method

- A better strategy for faster convergence is as follows.
- Lets $\Delta x_i = -f(x_i)/f'(x_i)$ be the step size in i-th iteration.
- Calculate the $f(x)$ at $x = x_i = x_i - \Delta x_i$.
- Check if $|f(x_{i+1})| < |f(x_i)|$. If yes accept, else halve the step.
- Keep halving the steps till $|f(x_{i+1})| < |f(x_{i+1})|$.

Modified Newton method

- A better strategy for faster convergence is as follows.
- Lets $\Delta x_i = -f(x_i)/f'(x_i)$ be the step size in i-th iteration.
- Calculate the $f(x)$ at $x = x_i = x_i - \Delta x_i$.
- Check if $|f(x_{i+1})| < |f(x_i)|$. If yes accept, else halve the step.
- Keep halving the steps till $|f(x_{i+1})| < |f(x_{i+1})|$.
- DEMO: Modified Newton Method

Modified Newton method

- A better strategy for faster convergence is as follows.
- Lets $\Delta x_i = -f(x_i)/f'(x_i)$ be the step size in i-th iteration.
- Calculate the $f(x)$ at $x = x_i = x_i - \Delta x_i$.
- Check if $|f(x_{i+1})| < |f(x_i)|$. If yes accept, else halve the step.
- Keep halving the steps till $|f(x_{i+1})| < |f(x_{i+1})|$.
- DEMO: Modified Newton Method
- **Convergence:** Let e_i be the error after i_{th} iteration. The iteration is quadratically convergent if

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} \text{ is finite.}$$

- Let r be a root of the function $f(x)$. Let we are at x_i in i-th iteration.
- By Taylor expansion,

$$f(r) \approx f(x_i) + (r - x_i)f'(x_i) + \frac{(r - x_i)^2}{2} f''(c_i)$$

c_i is between x_i and r.

Newton-Raphson method

- Since $f(r) = 0$,

$$-\frac{f(x_i)}{f'(x_i)} = r - x_i + \frac{(r - x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)}$$

- Assuming $f'(x_i) \neq 0$,

$$\begin{aligned} x_i - \frac{f(x_i)}{f'(x_i)} - r &= \frac{(r - x_i)^2}{2}\frac{f''(c_i)}{f'(x_i)} \\ x_{i+1} - r &= e_i^2 \frac{f''(c_i)}{2f'(x_i)} \end{aligned}$$

- Since $c_i \to r$ as $i \to \infty 0$,

$$\lim_{i\to\infty} \frac{e_{i+1}}{e_i^2} = \left| \frac{f''(r)}{2f'(r)} \right|$$

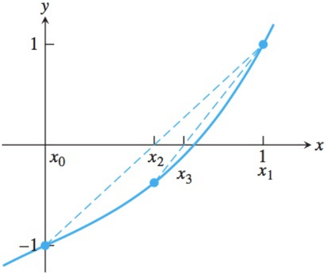
- Hence Newton's method is convergent quadratically if $f'(r) \neq 0$.

Secant method

- A method without derivative.

Secant method

- A method without derivative.



- If x_{i-1} and x_i are the last two guesses, it replaces the derivative by the approaximation

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

- The iteration step is:

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}, \quad i = 1, 2, 3, \ldots$$

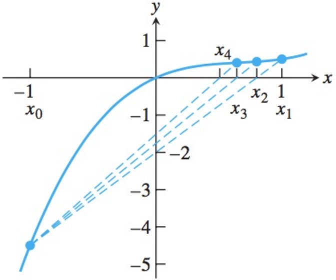
- Example: find the root $f(x) = x^3 + x - 1$ with starting guesses 0 and 1.

The method of false position (*regula falsi*)

- Similar to bisection method, but the midpoint is repaced by a secant like approaximation.

The method of false position (*regula falsi*)

- Similar to bisection method, but the midpoint is repaced by a secant like approaximation.

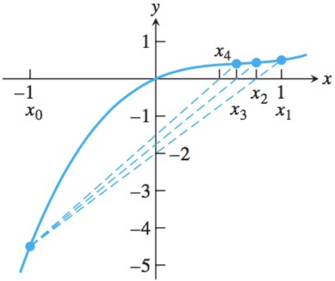


- Given an interval $[a, b]$ that brackets a root, the next point is

$$c = a - \frac{f(a)(a-b)}{f(a)-f(b)} = \frac{bf(a) - af(b)}{f(a)-f(b)}$$

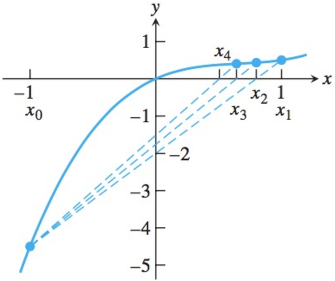
- The new point c is guaranteed to be in $[a, b]$ since the points $(a, f(a))$ and $(b, f(b))$ lie on separate side of the x-axis.

The method of false position (*regula falsi*)



- In the above, we approach the zero from one side. But this can be slow.
- We can improve by approacing from both sides.
- If the other side is at x_0, then we simply replace $f(x_0) \to f(x_0)/2$ in each iteration.

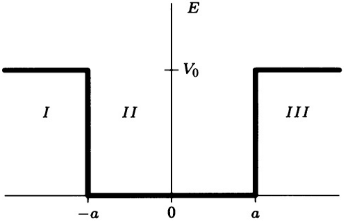
The method of false position (*regula falsi*)



- In the above, we approach the zero from one side. But this can be slow.
- We can improve by approacing from both sides.
- If the other side is at x_0, then we simply replace $f(x_0) \to f(x_0)/2$ in each iteration.
- DEMO: Regula Falsi

The Quantum Well problem

- Consider a quantum particle in a *finite 1D* potential well.



- The Schrodinger equation is

$$-\frac{\hbar^2}{2m}\frac{\partial^2\Psi}{\partial x^2} + V_0\Psi = E\Psi$$

- We can write it as

$$\frac{\partial^2\Psi}{\partial x^2} - \beta^2\Psi = 0, \quad \beta = \sqrt{2m(V_0 - E)/\hbar^2}$$

The Quantum Well problem

- The general solutions are:

$$\Psi_I(x) = Ce^{\beta x}$$

$$\Psi_{II}(x) = A\sin\alpha x + B\cos\alpha x, \quad \alpha = \sqrt{2mE/\hbar^2}$$

$$\Psi_{III}(x) = De^{-\beta x}$$

The Quantum Well problem

- The general solutions are:

$$\begin{aligned}\Psi_I(x) =& Ce^{\beta x}\\ \Psi_{II}(x) =& A\sin\alpha x + B\cos\alpha x, \quad \alpha = \sqrt{2mE/\hbar^2}\\ \Psi_{III}(x) =& De^{-\beta x}\end{aligned}$$

- Boundary conditions are: $\Psi(x)$ and $\Psi'(x)$ are continous at $x = \pm a$.
- BC at $x = -a$ gives

$$\begin{aligned}-A\sin\alpha a + B\cos\alpha a =& Ce^{-\beta a}\\ A\alpha\cos\alpha a + B\alpha\sin\alpha a =& \beta Ce^{-\beta a}\end{aligned}$$

The Quantum Well problem

- The general solutions are:

$$\Psi_I(x) = Ce^{\beta x}$$
$$\Psi_{II}(x) = A\sin\alpha x + B\cos\alpha x, \quad \alpha = \sqrt{2mE/\hbar^2}$$
$$\Psi_{III}(x) = De^{-\beta x}$$

- Boundary conditions are: $\Psi(x)$ and $\Psi'(x)$ are continous at $x = \pm a$.
- BC at $x = -a$ gives

$$-A\sin\alpha a + B\cos\alpha a = Ce^{-\beta a}$$
$$A\alpha\cos\alpha a + B\alpha\sin\alpha a = \beta Ce^{-\beta a}$$

- BC at $x = a$ gives

$$A\sin\alpha a + B\cos\alpha a = De^{\beta a}$$
$$A\alpha\cos\alpha a - B\alpha\sin\alpha a = -\beta De^{\beta a}$$

The Quantum Well problem

- We get the following two equations:

$$
\begin{aligned}
2B\cos(\alpha a) =& (C + D)e^{-betaa} \\
2A\alpha\cos(\alpha a) =& \beta(C - D)e^{-betaa}
\end{aligned}
$$

The Quantum Well problem

- We get the following two equations:

$$
\begin{aligned}
2B\cos(\alpha a) =& (C + D)e^{-betaa} \\
2A\alpha\cos(\alpha a) =& \beta(C - D)e^{-betaa}
\end{aligned}
$$

- There are two classes of solutions:

$$
\begin{aligned}
A = 0, \quad B \neq 0, \quad C = D \Rightarrow \alpha\tan(\alpha a) = \beta, \quad \text{Even states} \\
A \neq 0, \quad B = 0, \quad C = -D \Rightarrow \alpha\cot(\alpha a) = -\beta, \quad \text{Odd states}
\end{aligned}
$$

- So we need to solve these two transcendental equations to find the wave function and the energy eigen values.

The Quantum Well problem

- Let us take the first equation.

$$f(E) = \alpha \tan(\alpha a) - \beta$$

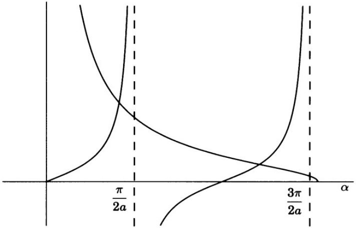
- We need to solve for E such that $f(E) = 0$.

The Quantum Well problem

- Let us take the first equation.

$$f(E) = \alpha \tan(\alpha a) - \beta$$

- We need to solve for E such that $f(E) = 0$.
- For numerical solution, we need to make an initial guess. How?
- Fix the constants. Let's say: $V_0 = 10$ eV, $a = 3$Å, $m = m_e$.
- First, we expect $0 < E < V_0$.
- Next we can make plots of $\tan(\alpha a)$ and β/α versus α.



The Quantum Well problem

- We find that there is a solution between

$$0 < E < \frac{\pi^2 \hbar^2}{8ma^2} \approx 1.045eV \approx 1.045eV$$

The Quantum Well problem

- We find that there is a solution between

$$0 < E < \frac{\pi^2\hbar^2}{8ma^2} \approx 1.045eV \approx 1.045eV$$

- We can write the equation in the form,

$$f(E) = \beta\cos(\alpha a) - \alpha\sin(\alpha a) = 0$$

- Finally, we should use *natural units*: $m_e = 1$, $a = 3$Å and $\hbar^2 = 7.609097\ m_e(eV)$Å^2.

The Quantum Well problem

- We find that there is a solution between

$$0 < E < \frac{\pi^2 \hbar^2}{8ma^2} \approx 1.045eV \approx 1.045eV$$

- We can write the equation in the form,

$$f(E) = \beta \cos(\alpha a) - \alpha \sin(\alpha a) = 0$$

- Finally, we should use *natural units*: $m_e = 1$, $a = 3\text{Å}$ and $\hbar^2 = 7.609097\, m_e(eV)\text{Å}^2$.

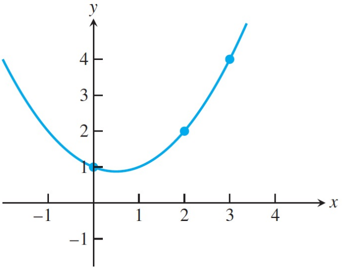
- DEMO: Quantum Well

Interpolation

- A function $y = P(x)$ **interpolates** a set of data points $(x_1, y_1), \ldots, (y_n, y_n)$ if $P(x_i) = y_i$ for $i = 1, \ldots, n$, i.e. it passes through all the points.

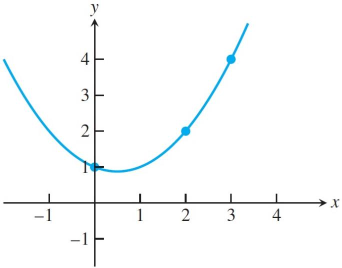
Interpolation

- A function $y = P(x)$ **interpolates** a set of data points $(x_1, y_1), \ldots, (y_n, y_n)$ if $P(x_i) = y_i$ for $i = 1, \ldots, n$, i.e. it passes through all the points.
- Example: Parabola $P(x) = x^2/2 - x/2 + 1$ interpolates $(0, 1)$, $(2, 2)$ and $(3, 4)$.



Interpolation

- A function $y = P(x)$ **interpolates** a set of data points $(x_1, y_1), \ldots, (y_n, y_n)$ if $P(x_i) = y_i$ for $i = 1, \ldots, n$, i.e. it passes through all the points.
- Example: Parabola $P(x) = x^2/2 - x/2 + 1$ interpolates $(0, 1)$, $(2, 2)$ and $(3, 4)$.



- Interplation can be viewd as a way of *data compression*.
- The numerical problem is - given a set of data points, find the interpolating function.

Polynomial Interpolation

- Consider data points (x_i, y_i), $i = 1, \ldots, n$. We would like to find an interpolating polynomial.

Polynomial Interpolation

- Consider data points (x_i, y_i), $i = 1, \ldots, n$. We would like to find an interpolating polynomial.
- **Lagrange interpolating formula**: Let $n = 3$. Construct the polynomial of degree $d = n - 3 = 2$ as follows:

$$P_2(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

- It passes through the points (x_1, y_1), (x_2, y_2) and (x_3, y_3), and called *Lagrange interpolating polynomial*.

Polynomial Interpolation

- Consider data points (x_i, y_i), $i = 1, \dots, n$. We would like to find an interpolating polynomial.
- **Lagrange interpolating formula**: Let $n = 3$. Construct the polynomial of degree $d = n - 3 = 2$ as follows:

$$P_2(x) = y_1 \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} + y_2 \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} + y_3 \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

- It passes through the points (x_1, y_1), (x_2, y_2) and (x_3, y_3), and called *Lagrange interpolating polynomial*.
- For the general case of n points, first define the degree $n - 1$ polynomial:

$$L_k(x) = \frac{(x - x_1) \dots (x - x_{k-1})(x - x_{k+1}) \dots (x - x_n)}{(x_k - x_1) \dots (x_k - x_{k-1})(x_k - x_{k+1}) \dots (x_k - x_n)}$$

- Then the Lagrange interpolating polynomial is:

$$P_{n-1}(x) = y_1 L_1(x) + \cdots + y_n L_n(x)$$

Polynomial Interpolation

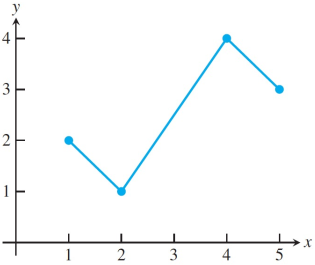
- Lagrange Interpolation Demo

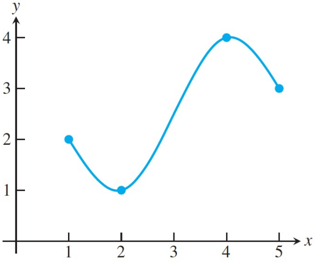
Cubic Splines

- In interpolation, a single formula is used to meet all data points.
- In splines, we use several formulas, each a low degree polynomial, to pass through successive sets of data points.

Cubic Splines

- In interpolation, a single formula is used to meet all data points.
- In splines, we use several formulas, each a low degree polynomial, to pass through successive sets of data points.
- Consider n data points - (x_i, y_i) with $x_1 < x_2 < \ldots < x_n$.
- A linear spline consists of $n - 1$ line segments that are drawn between neighbouring points.
- A cubic spline replaces linear functions between the data points by degree 3 polynomial.





Cubic Splines

Construction

Cubic Splines

Construction

- Consider n data points - (x_i, y_i) with $x_1 < x_2 < \ldots < x_n$.
- A **cubic spline** through the data points is a set of cubic polys

$$
\begin{aligned}
S_1(x) =& y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 \quad \text{on } [x_1, x_2] \\
S_2(x) =& y_2 + b_2(x - x_2) + c_2(x - x_2)^2 + d_2(x - x_2)^3 \quad \text{on } [x_2, x_3] \\
\vdots =& \quad \vdots \\
S_{n-1}(x) =& y_{n-1} + b_{n-1}(x - x_{n-1}) + c_{n-1}(x - x_{n-1})^2 + d_{n-1}(x - x_{n-1})^3 \\
\text{on}&[x_{n-1}, x_n]
\end{aligned}
$$

Cubic Splines

- The polynomials have the following properties.
  1. $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for $i = 1, 2, \ldots n-1$. Interpolates at the points.

Cubic Splines

- The polynomials have the following properties.
  1. $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for $i = 1, 2, \ldots n-1$. Interpolates at the points.
  2. $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 2, 3, \ldots n-1$. Slopes match at interior points - *smoothness*.

Cubic Splines

- The polynomials have the following properties.
  1. $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for $i = 1, 2, \ldots n-1$. Interpolates at the points.
  2. $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 2, 3, \ldots n-1$. Slopes match at interior points - *smoothness*.
  3. $S''_{i-1}(x_i) = S''_i(x_i)$ for $i = 2, 3, \ldots n-1$. Second derivatives match at interior points - *curvature matches*.

Cubic Splines

- The polynomials have the following properties.
  1. $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$ for $i = 1, 2, \ldots n-1$. Interpolates at the points.
  2. $S'_{i-1}(x_i) = S'_i(x_i)$ for $i = 2, 3, \ldots n-1$. Slopes match at interior points - *smoothness*.
  3. $S''_{i-1}(x_i) = S''_i(x_i)$ for $i = 2, 3, \ldots n-1$. Second derivatives match at interior points - *curvature matches*.
- There are a total of $3n-5$ equations.
- But $3n-3$ unknowns a_i, b_i, c_i.
- Set two more constraints - *natural splines*:

$$S''_1(x_1) = 0 \quad \text{and} \quad S''_{n-1}(x_n) = 0$$

Cubic Splines

- Property-1 generates $n-1$ equations:

$$
\begin{aligned}
y_2 &= y_1 + b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 + d_1(x_2 - x_1)^3 \\
&\vdots \\
y_n &= y_{n-1} + b_{n-1}(x_n - x_{n-1}) + c_{n-1}(x_n - x_{n-1})^2 + d_{n-1}(x_n - x_{n-1})^3
\end{aligned}
$$

Cubic Splines

- Property-1 generates $n-1$ equations:

$$y_2 = y_1 + b_1(x_2 - x_1) + c_1(x_2 - x_1)^2 + d_1(x_2 - x_1)^3$$
$$\vdots$$
$$y_n = y_{n-1} + b_{n-1}(x_n - x_{n-1}) + c_{n-1}(x_n - x_{n-1})^2 + d_{n-1}(x_n - x_{n-1})^3$$

- Property-2 generates $n-2$ equations:

$$\begin{aligned}
0 =& S_1'(x_2) - S_2'(x_2) = b_1 + 2c_1(x_2 - x_1) + 3d_1(x_2 - x_1)^2 - b_2 \\
&\vdots \\
0 =& S_{n-2}'(x_{n-1}) - S_{n-1}'(x_{n-1}) = b_{n-2} + 2c_{n-2}(x_{n-1} - x_{n-2}) \\
&+ 3d_{n-2}(x_{n-1} - x_{n-2})^2 - b_{n-1}
\end{aligned}$$

Cubic Splines

- Property-3 generates $n-2$ equations:

$$
\begin{aligned}
0 =& S_1''(x_2) - S_2''(x_2) = 2c_1 + 6d_1(x_2 - x_1) - 2c_2 \\
& \vdots \\
0 =& S_{n-2}''(x_{n-1}) - S_{n-1}''(x_{n-1}) = 2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2})^2 - 2c_{n-1}
\end{aligned}
$$

Cubic Splines

- Property-3 generates $n-2$ equations:

$$
\begin{aligned}
0 =& S_1''(x_2) - S_2''(x_2) = 2c_1 + 6d_1(x_2 - x_1) - 2c_2 \\
&\vdots \\
0 =& S_{n-2}''(x_{n-1}) - S_{n-1}''(x_{n-1}) = 2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2})^2 - 2c_{n-1}
\end{aligned}
$$

- Property-4 generates 2 equations:

$$
\begin{aligned}
S_1''(x_1) =& 0 \Rightarrow 2c_1 = 0 \\
S_1''(x_{n-1}) =& 0 \Rightarrow 2c_n = 0
\end{aligned}
$$

Cubic Splines

- Property-3 generates $n-2$ equations:

$$
\begin{aligned}
0 =& S_1''(x_2) - S_2''(x_2) = 2c_1 + 6d_1(x_2 - x_1) - 2c_2 \\
&\vdots \\
0 =& S_{n-2}''(x_{n-1}) - S_{n-1}''(x_{n-1}) = 2c_{n-2} + 6d_{n-2}(x_{n-1} - x_{n-2})^2 - 2c_{n-1}
\end{aligned}
$$

- Property-4 generates 2 equations:

$$
\begin{aligned}
S_1''(x_1) =& 0 \Rightarrow 2c_1 = 0 \\
S_1''(x_{n-1}) =& 0 \Rightarrow 2c_n = 0
\end{aligned}
$$

- Define $\delta_i = x_{i+1} - x_i$ and $\Delta_i = y_{i+1} - y_i$.

Cubic Splines

- Determine c_i-s from the following equation:

$$
\begin{bmatrix}
1 & 0 & 0 & & & \\
\delta_1 & 2\delta_1 + 2\delta_2 & \delta_2 & & & \\
0 & \delta_2 & 2\delta_2 + 2\delta_3 & \delta_3 & & \\
 & \ddots & \ddots & \ddots & \ddots & \\
 & & & \delta_{n-2} & 2\delta_{n-2} + 2\delta_{n-1} & \delta_{n-1} \\
 & & & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
c_1 \\ \\ \vdots \\ \\ c_n
\end{bmatrix}
=
\begin{bmatrix}
0 \\
3(\frac{\Delta_2}{\delta_2} - \\
\vdots \\
3\frac{\Delta_{n-1}}{\delta_{n-1}} - \\
0
\end{bmatrix}
$$

Cubic Splines

- Determine c_i-s from the following equation:

$$\begin{bmatrix} 1 & 0 & 0 & & & \\ \delta_1 & 2\delta_1 + 2\delta_2 & \delta_2 & & & \\ 0 & \delta_2 & 2\delta_2 + 2\delta_3 & \delta_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & & \delta_{n-2} & 2\delta_{n-2} + 2\delta_{n-1} & \delta_{n-1} \\ & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \\ \vdots \\ \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3(\frac{\Delta_2}{\delta_2} - \\ \vdots \\ 3\frac{\Delta_{n-1}}{\delta_{n-1}} - \\ 0 \end{bmatrix}$$

- Determine d_i-s from the following equations:

$$d_i = \frac{c_{i+1} - c_i}{3\delta_i}, \quad i = 1, 2, \ldots n - 1$$

Cubic Splines

- Determine c_i-s from the following equation:

$$\begin{bmatrix} 1 & 0 & 0 & & & \\ \delta_1 & 2\delta_1 + 2\delta_2 & \delta_2 & & & \\ 0 & \delta_2 & 2\delta_2 + 2\delta_3 & \delta_3 & & \\ & \ddots & \ddots & \ddots & \ddots & \\ & & & \delta_{n-2} & 2\delta_{n-2} + 2\delta_{n-1} & \delta_{n-1} \\ & & & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ \\ \vdots \\ \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 3(\frac{\Delta_2}{\delta_2} - \\ \vdots \\ 3\frac{\Delta_{n-1}}{\delta_{n-1}} - \\ 0 \end{bmatrix}$$

- Determine d_i-s from the following equations:

$$d_i = \frac{c_{i+1} - c_i}{3\delta_i}, \quad i = 1, 2, \ldots n - 1$$

- Determine b_i-s from the following equations:

$$b_i = \frac{\Delta_i}{\delta_i} - \frac{\delta_i}{3}(2c_i + c_{i+1}), \quad i = 1, 2, \ldots n - 1$$

Least Squares

- **Inconsistent systems of equations**: Consider the following system of linear equations

$$\begin{aligned} x_1 + x_2 &= 2 \\ x_1 - x_2 &= 1 \\ x_1 + x_2 &= 3 \end{aligned} \quad \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \quad \Rightarrow A\mathbf{x} = \mathbf{b}$$

- It has no solution. In general m equations of n unknowns with $m > n$ has no solutions and is called inconsistent.

Least Squares

- **Inconsistent systems of equations**: Consider the following system of linear equations

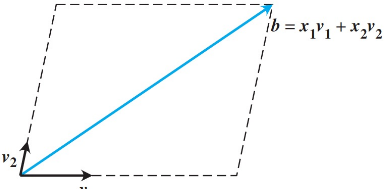
$$\begin{aligned} x_1 + x_2 &= 2 \\ x_1 - x_2 &= 1 \\ x_1 + x_2 &= 3 \end{aligned} \qquad \Rightarrow \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \qquad \Rightarrow A\mathbf{x} = \mathbf{b}$$

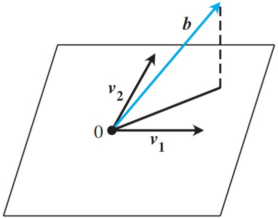
- It has no solution. In general m equations of n unknowns with $m > n$ has no solutions and is called inconsistent.
- Still we want to find an **approaximate** solution to it. How can we do it?
- Write the equations in the following form:

$$x_1 \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} + x_2 \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$
$$\Rightarrow x_1\mathbf{v}_1 + x_2\mathbf{v}_2 = \mathbf{b}$$

Least Squares

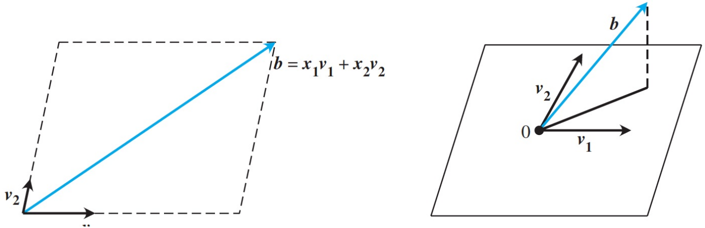
- We can interpret $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{b}$ as three 3D vectors. We want to find x_1 and x_2 that satisfies the vector equation $x_1\mathbf{v}_1 + x_2\mathbf{v}_2 = \mathbf{b}$.
- But if $\mathbf{b}$ lies outside the plan containing $\mathbf{v}_1$, $\mathbf{v}_2$, then there is no solution.





Least Squares

- We can interpret $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{b}$ as three 3D vectors. We want to find x_1 and x_2 that satisfies the vector equation $x_1\mathbf{v}_1 + x_2\mathbf{v}_2 = \mathbf{b}$.
- But if $\mathbf{b}$ lies outside the plan containing $\mathbf{v}_1$, $\mathbf{v}_2$, then there is no solution.



- Consider dropping a perpendicular from the tip of $\mathbf{b}$ on the plan containing $\mathbf{v}_1$, $\mathbf{v}_2$. Let the point be $\bar{x}_1\mathbf{v}_1 + \bar{x}_2\mathbf{v}_2 = A\bar{\mathbf{x}}$.
- The residual vector $\mathbf{b} - A\bar{\mathbf{x}}$ is $\perp$ to the plane.
- $\bar{\mathbf{x}}$ is the **best possible** solution to the inconsistent solutions.

Least Squares

- Now since the vector $A\mathbf{x}$ is $\perp$ to $\mathbf{b} - A\bar{\mathbf{x}}$, their dot product vanishes,

$$
\begin{aligned}
&(A\mathbf{x})^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \quad \forall \mathbf{x} \\
\Rightarrow &\mathbf{x}^T A^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \quad \forall \mathbf{x} \\
\Rightarrow &A^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \\
\Rightarrow &A^T A\bar{\mathbf{x}} = A^T\mathbf{b}
\end{aligned}
$$

Least Squares

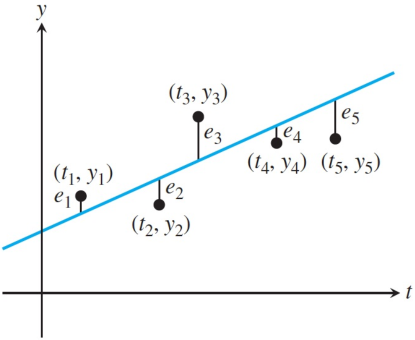
- Now since the vector $A\mathbf{x}$ is $\perp$ to $\mathbf{b} - A\bar{\mathbf{x}}$, their dot product vanishes,

$$
\begin{aligned}
&(A\mathbf{x})^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \quad \forall \mathbf{x} \\
\Rightarrow &\mathbf{x}^T A^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \quad \forall \mathbf{x} \\
\Rightarrow &A^T(\mathbf{b} - A\bar{\mathbf{x}}) = 0 \\
\Rightarrow &A^T A\bar{\mathbf{x}} = A^T\mathbf{b}
\end{aligned}
$$

- The last equation is called the **normal equation**. The solution $\bar{\mathbf{x}}$ is the least squares solutions of $Ax = b$.

Least Squares

- **Data fitting problem**: Consider a set of data points (t_i, y_i) where $i = 1, 2, \ldots, m$.
- We want to **fit** the data with a linear model, e.g. $y = c_1 + c_2 t$.
- The model need not pass through the points (t_i, y_i). The error is defined as $e_i = y_i - (c_1 + c_2 t_i)$.



- We want to find c_1, c_2 such that the rms error $\sigma = \sqrt{\sum_i e_i^2/m}$ is minimized.

Least Squares

- This problem of minimizing rms error is equivalent to finding the least square solution to a normal equation.
- We first choose the model, such as $y = c_1 + c_2 t$.
- Next substitute the data points into the model. Each data point create an equation with unknowns c_1 and c_2. This results in a system $A\mathbf{x} = \mathbf{b}$.
- Solve the normal equation $A^T A x = A^T \mathbf{b}$.

Least Squares

- This problem of minimizing rms error is equivalent to finding the least square solution to a normal equation.
- We first choose the model, such as $y = c_1 + c_2 t$.
- Next substitute the data points into the model. Each data point create an equation with unknowns c_1 and c_2. This results in a system $A\mathbf{x} = \mathbf{b}$.
- Solve the normal equation $A^T A x = A^T \mathbf{b}$.
- Least Squares Demo

Numerical differentiation

Finite Difference Formulae

- We need to evaluate

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

- Taylor's theorm *with Reminder*: If a function $f(x)$ is $k+1$ times continously differentiable between x and $x+h$, *then there exist a number* $c \in [x, x+h]$ such that

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \ldots + \frac{h^k}{k!} f^{(k)}(x) + \frac{h^{k+1}}{(k+1)!} f^{(k+1)}(c)$$

  The last term is called **Taylor reminder**.

- Limiting k to $k=2$

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2} f''(c)$$

Numerical differentiation

- Thus we get,

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \frac{h^2}{2} f''(c)$$

- **Two point forward difference formula**: Treating the last term as an error

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \rightarrow \textit{formula used numerically}$$

- The error is $O(h)$, that is $\propto h$. Hence the above is called a first order formula.

Numerical differentiation

- Thus we get,

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \frac{h^2}{2} f''(c)$$

- **Two point forward difference formula**: Treating the last term as an error

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \rightarrow \textit{formula used numerically}$$

- The error is $O(h)$, that is $\propto h$. Hence the above is called a first order formula.
- **Example**: Consider derivative of $f(x) = \frac{1}{2}$ at $x = 2$. Taking $h = 0.1$,

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} = -0.2381$$

- Exact $f'(x) = -0.25$. Exact Error = $-0.2381 - (-0.25) = 0.0119$.
- **Estimated error**: c is between 2 and 2.1. Hence estimated error $hf''(c)/2$ is between 0.0125 and 0.0108.

Numerical differentiation

- **A second order formula**: Consider the Taylor expansions

$$
\begin{aligned}
f(x+h) =& f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(c_1), \quad x < c_1 < x+h \\
f(x-h) =& f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(c_2), \quad x-h < c_2 < x
\end{aligned}
$$

- Subtracting we get the centered difference formula,

$$
f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(c_1) - \frac{h^2}{6}f'''(c_2)
$$

  The two error term can be combined to get,

$$
f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}f'''(c), \quad x-h < c < x+h
$$

- **Example:** Derivative of $f(x) = \frac{1}{x}$ at $x = 2$ give -0.2506. An improvement over the first order formula.

Numerical differentiation

- **Three point centered difference formula for Second Derivative**:

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12} f^{(4)}(c), \quad x-h < c < x+h$$

Numerical differentiation

- **Three point centered difference formula for Second Derivative**:

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12} f^{(4)}(c), \quad x - h < c < x + h$$

- **Rounding error**: Consider the function $f(x) = e^x$. Find $f'(x)$ at $x = 0$ with smaller and smaller h. What do we expect about the error?

Numerical differentiation

- **Three point centered difference formula for Second Derivative**:

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} - \frac{h^2}{12} f^{(4)}(c), \quad x-h < c < x+h$$

- **Rounding error**: Consider the function $f(x) = e^x$. Find $f'(x)$ at $x = 0$ with smaller and smaller h. What do we expect about the error?

h	First Order Formula	error	Second Order Formula	error
10^{-1}	1.05170918075648	−0.05170918075648	1.00166750019844	−0.00166750019844
10^{-2}	1.00501670841679	−0.00501670841679	1.00001666674999	−0.00001666674999
10^{-3}	1.00050016670838	−0.00050016670838	1.00000016666668	−0.00000016666668
10^{-4}	1.00005000166714	−0.00005000166714	1.00000000166689	−0.00000000166689
10^{-5}	1.00000500000696	−0.00000500000696	1.00000000001210	−0.00000000001210
10^{-6}	1.00000049996218	−0.00000049996218	0.99999999997324	0.00000000002676
10^{-7}	1.00000004943368	−0.00000004943368	0.99999999947364	0.00000000052636
10^{-8}	0.99999999392253	0.00000000607747	0.99999999392253	0.00000000607747
10^{-9}	1.00000008274037	−0.00000008274037	1.00000002722922	−0.00000002722922

Numerical Integration

- Given a function $f(x)$, how can we evaluate the following numerically?

$$I = \int_a^b f(x)$$

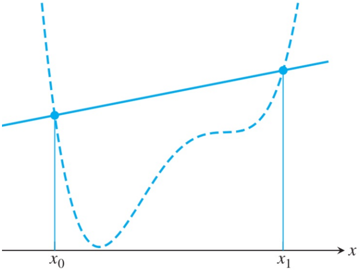
- The idea is to **approximate** the function by an **interpolating** polynomial or by a **least square** polynomial. Once this is done, we can analytically integrate these polynomials.

Numerical Integration

- Given a function $f(x)$, how can we evaluate the following numerically?

$$I = \int_a^b f(x)$$

- The idea is to **approximate** the function by an **interpolating** polynomial or by a **least square** polynomial. Once this is done, we can analytically integrate these polynomials.
- **Trapezoid Rule:** Consider a function $f(x)$ in the interval $[x_0, x_1]$. Let $y_0 = f(x_0)$ and $y_1 = f(x_1)$.
- The simplest thing we can do is to approximate the function by a degree-1 interpolating polynomial.



Numerical Integration

- Using the Lagrange method,

$$f(x) \approx P_1(x) = y_0 \frac{x - x_1}{x_0 - x_1} + y_1 \frac{x - x_0}{x_1 - x_0}$$

- Define the interval $h = x_1 - x_0$. The integration is

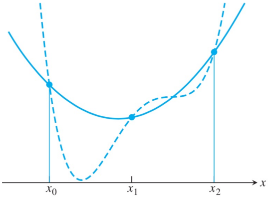
$$\begin{aligned} I = \int_{x_0}^{x_1} f(x)\,dx \approx & y_0 \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1}\,dx + \int_{x_0}^{x_1} y_1 \frac{x - x_0}{x_1 - x_0}\,dx \\ = & y_0 \frac{h}{2} + y_1 \frac{h}{2} \end{aligned}$$

- Therefore

$$I \approx \frac{h}{2}(y_0 + y_1) \rightarrow \text{Trapezoid rule}$$

Numerical Integration

- **Simpson's Rule**: Approaximate the function by a 2nd order polynomial (parabola):

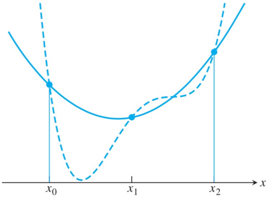


- We have to find $I = \int_{x_0}^{x_2}$. Take a point $x_1 = (x_0 + x_2)/2$ in the middle. Define $h = x_1 - x_0 = x_2 - x_1$. The polynomial is

$$f(x) \approx P_2(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Numerical Integration

- **Simpson's Rule**: Approaximate the function by a 2nd order polynomial (parabola):



- We have to find $I = \int_{x_0}^{x_2}$. Take a point $x_1 = (x_0 + x_2)/2$ in the middle. Define $h = x_1 - x_0 = x_2 - x_1$. The polynomial is

$$f(x) \approx P_2(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

- The integration is

$$I = \int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}(y_0 + 4y_1 + y_2) \rightarrow \text{Simpson's 1/3 Rule}$$

Numerical Integration

- **Simpson's 3/8 Rule**: By using a degree-3 polynomial, we get the 3/8 rule

$$I = \int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}\,(y_0 + 3y_1 + 3y_2 + y_3)$$

$h = x_1 - x_0 = x_2 - x_1 = x_3 - x_2.$

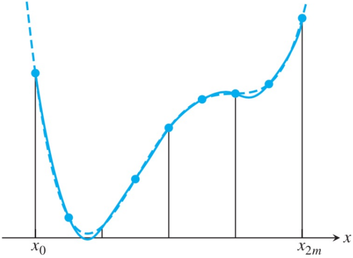
Numerical Integration

- **Simpson's 3/8 Rule**: By using a degree-3 polynomial, we get the 3/8 rule

$$I = \int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}\left(y_0 + 3y_1 + 3y_2 + y_3\right)$$

$h = x_1 - x_0 = x_2 - x_1 = x_3 - x_2$.

- **Composite Newton-Cotes formulas**: One way to improve the accuracy is to divide the interval $[a, b]$ into n-number of equal sub-intervals and then apply the closed Newton-Cotes formula to it.



Numerical Integration

- **Composite Trapezoid Rule**: We have $I = \int_a^b f(x)\,dx$.
- Divide the interval $[a, b]$ into the following evenly spaced grid

$$a = x_0 < x_1 < x_2 < \ldots < x_{m-1} < x_m = b$$

- Define the sub-interval width $h = x_{i+1} - x_i$. On each subinterval use the trapezoid rule

$$\int_{x_i}^{x_{i+1}} = \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

Numerical Integration

- **Composite Trapezoid Rule**: We have $I = \int_a^b f(x)\,dx$.
- Divide the interval $[a, b]$ into the following evenly spaced grid

$$a = x_0 < x_1 < x_2 < \ldots < x_{m-1} < x_m = b$$

- Define the sub-interval width $h = x_{i+1} - x_i$. On each subinterval use the trapezoid rule

$$\int_{x_i}^{x_{i+1}} = \frac{h}{2}\left(f(x_i) + f(x_{i+1})\right)$$

- Adding the contributions from all the sub-intervals gives the *composite trapezoid rule*

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=1}^{m-1} f(x_i)\right]$$

Numerical Integration

- **Composite Simpson's Rule**: Divide the interval $[a, b]$ into the following evenly spaced grid

$$a = x_0 < x_1 < x_2 < \ldots < x_{2m-2} < x_{2m-1} < x_{2m} = b$$

- Define $h = x_{i+1} - x_i$. On each interval of width $2h$, i.e. $[x_{2i}, x_{2i+2}]$, $i = 0, 1, \ldots, m-1$, apply Simpson's rule.

$$\int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \approx \frac{h}{3}\left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2})\right]$$

- Adding up all the sub-intervals,

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(a) + f(b) + 4\sum_{i=1}^{m} f(x_{2i-1}) + 2\sum_{i=1}^{m-1} f(x_{2i})\right]$$

Numerical Integration

- **Composite Simpson's Rule**: Divide the interval $[a, b]$ into the following evenly spaced grid

$$a = x_0 < x_1 < x_2 < \ldots < x_{2m-2} < x_{2m-1} < x_{2m} = b$$

- Define $h = x_{i+1} - x_i$. On each interval of width $2h$, i.e. $[x_{2i}, x_{2i+2}]$, $i = 0, 1, \ldots, m-1$, apply Simpson's rule.

$$\int_{x_{2i}}^{x_{2i+2}} f(x)\, dx \approx \frac{h}{3} \left[f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2}) \right]$$

- Adding up all the sub-intervals,

$$\int_a^b f(x)\, dx \approx \frac{h}{3} \left[f(a) + f(b) + 4 \sum_{i=1}^{m} f(x_{2i-1}) + 2 \sum_{i=1}^{m-1} f(x_{2i}) \right]$$

- **Example**: write a program to find $\int_0^\pi \sin^2(x)\, dx$ using composite Simpson's rule.

Numerical Integration

- **Degree of Precision (DOP)**: It is the greatest integer k for which all degree-k or less polynomials are integrated exactly by the method. Examples: DOP of
  - Trapezoid rule is 1.
  - Simpson's 1/3 rule is 3. (Not 2 as expected due a special property).
  - Simpson's 3/8 rule is 3.

Numerical Integration

- **Degree of Precision (DOP)**: It is the greatest integer k for which all degree-k or less polynomials are integrated exactly by the method. Examples: DOP of
  - Trapezoid rule is 1.
  - Simpson's 1/3 rule is 3. (Not 2 as expected due a special property).
  - Simpson's 3/8 rule is 3.
- These rules make $n+1$ function evalulation to give a DOP of n or $n+1$.
- **Question**: is it possible to have same DOP with fewer number of function evaluations?

Numerical Integration

- **Degree of Precision (DOP)**: It is the greatest integer k for which all degree-k or less polynomials are integrated exactly by the method. Examples: DOP of
  - Trapezoid rule is 1.
  - Simpson's 1/3 rule is 3. (Not 2 as expected due a special property).
  - Simpson's 3/8 rule is 3.
- These rules make $n + 1$ function evalulation to give a DOP of n or $n + 1$.
- **Question**: is it possible to have same DOP with fewer number of function evaluations?
- The answer is **yes**. There is a **Gaussian Quadrature** method which give DOP of $(2n + 1)$ using $(n + 1)$ function evaluations.

Gaussian Quadrature

- Consider the set of orthogonal polynomials $p_0, p_1, \ldots, p_n$ on the interval $[a, b]$. Degree of polynomial p_i is i.

$$\int_a^b p_m(x)p_n(x)\,dx = \begin{cases} 0 & m \neq n \\ \neq 0 & m = n \end{cases}$$

- The set form a vector space of polynomials of degrees upto n.
- A basis polynomial $p_i(x)$ has i distict roots in the interval (a, b).

Gaussian Quadrature

- Consider the set of orthogonal polynomials $p_0, p_1, \ldots, p_n$ on the interval $[a, b]$. Degree of polynomial p_i is i.

$$\int_a^b p_m(x) p_n(x)\, dx = \begin{cases} 0 & m \neq n \\ \neq 0 & m = n \end{cases}$$

- The set form a vector space of polynomials of degrees upto n.
- A basis polynomial $p_i(x)$ has i distict roots in the interval (a, b).
- Now consider the set of **Legendre polynomials** for $0 \leq i \leq n$:

$$p_i(x) = \frac{1}{2^i i!} \frac{d^i}{dx^i} \left[(x^2 - 1)^i \right]$$

- The Legendre polynomials are orthogonal on $[-1, 1]$.

Gaussian Quadrature

- Let's say we need to evaluate $I = \int_{-1}^{1} f(x)\, dx$.
- Consider the order-n Legendre polynomial $p_n(x)$. Let the roots of $p_n(x)$ be $x_1, x_2, \ldots, x_n$, with each $x_i \in (a, b)$.
- Now we divide the interval $[a, b]$ at NOT equally spaced points but at the grid points $x_1, x_2, \ldots, x_n$.

Gaussian Quadrature

- Let's say we need to evaluate $I = \int_{-1}^{1} f(x)\,dx$.
- Consider the order-n Legendre polynomial $p_n(x)$. Let the roots of $p_n(x)$ be $x_1, x_2, \ldots, x_n$, with each $x_i \in (a, b)$.
- Now we divide the interval $[a, b]$ at NOT equally spaced points but at the grid points $x_1, x_2, \ldots, x_n$.
- We can find an interpolating polynomial $Q(x)$ that passes through the points $(x_1, f(x_1)),\ (x_2, f(x_2))\ \ldots (x_n, f(x_n))$. Using the Lagrange's formula

$$f(x) \approx Q(x) = \sum_{i=1}^{n} L_i(x) f(x_i), \quad L_i(x) = \frac{(x - x_1)\cdots\overline{(x - x_i)}\cdots(x - x_n)}{(x_i - x_1)\cdots\overline{(x_i - x_i)}\cdots(x_i - x_n)}$$

Gaussian Quadrature

- Let's say we need to evaluate $I = \int_{-1}^{1} f(x)\,dx$.
- Consider the order-n Legendre polynomial $p_n(x)$. Let the roots of $p_n(x)$ be $x_1, x_2, \ldots, x_n$, with each $x_i \in (a, b)$.
- Now we divide the interval $[a, b]$ at NOT equally spaced points but at the grid points $x_1, x_2, \ldots, x_n$.
- We can find an interpolating polynomial $Q(x)$ that passes through the points $(x_1, f(x_1)),\ (x_2, f(x_2))\ \ldots (x_n, f(x_n))$. Using the Lagrange's formula

$$f(x) \approx Q(x) = \sum_{i=1}^{n} L_i(x) f(x_i), \quad L_i(x) = \frac{(x - x_1)\cdots\overline{(x - x_i)}\cdots(x - x_n)}{(x_i - x_1)\cdots\overline{(x_i - x_i)}\cdots(x_i - x_n)}$$

- Integrating both sides,

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} f(x_i) \int_{-1}^{1} L_i(x)\,dx = \sum_{i=1}^{n} c_i f(x_i)$$

Gaussian Quadrature

- We have

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} c_i f(x_i), \quad (\mathbf{DOP} = 2n-1)$$

- The coefficients c_i-s are universal. We can evalute it once with great accuracy and store. Example:

n	roots x_i	coefficients c_i
2	$-\sqrt{1/3} = -0.57735026918963$	$1 = 1.00000000000000$
	$\sqrt{1/3} = 0.57735026918963$	$1 = 1.00000000000000$
3	$-\sqrt{3/5} = -0.77459666924148$	$5/9 = 0.55555555555555$
	$0 = 0.00000000000000$	$8/9 = 0.88888888888888$
	$\sqrt{3/5} = 0.77459666924148$	$5/9 = 0.55555555555555$
4	$-\sqrt{\frac{15+2\sqrt{30}}{35}} = -0.86113631159405$	$\frac{90-5\sqrt{30}}{180} = 0.34785484513745$
	$-\sqrt{\frac{15-2\sqrt{30}}{35}} = -0.33998104358486$	$\frac{90+5\sqrt{30}}{180} = 0.65214515486255$
	$\sqrt{\frac{15-2\sqrt{30}}{35}} = 0.33998104358486$	$\frac{90+5\sqrt{30}}{180} = 0.65214515486255$
	$\sqrt{\frac{15+2\sqrt{30}}{35}} = 0.86113631159405$	$\frac{90-5\sqrt{30}}{180} = 0.34785484513745$

Gaussian Quadrature

- **Example:** Evaluate

$$\int_{-1}^{1} e^{-\frac{x^2}{2}}\, dx$$

using Gaussian quadrature. (Exact answer=1.71124878378430).

- The $n = 2$ approximation is

$$c_1 f(x_1) + c_2 f(x_2) = 1f(-1/\sqrt{3}) + 1f(\sqrt{3}) \approx 1.69296344978123$$

- The $n = 3$ approximation is

$$\frac{5}{9} f(-3/\sqrt{5}) + \frac{8}{9} f(0) + \frac{5}{9} f(3/\sqrt{5})) \approx 1.71202024520191$$

Gaussian Quadrature

- Why Gaussian quadrature using degree-n Legendre polynomial on $[-1, 1]$ has DOP equal to $2n - 1$?
- Consider a polynomial $P(x)$ of degree $2n - 1$. GQ should integrate it exactly.
- We can write $P(x)$ as

$$P(x) = S(x)p_n(x) + R(x)$$

$S(x)$ and $R(x)$ are polynomial of degree less than n.

Gaussian Quadrature

- Why Gaussian quadrature using degree-n Legendre polynomial on $[-1, 1]$ has DOP equal to $2n - 1$?
- Consider a polynomial $P(x)$ of degree $2n - 1$. GQ should integrate it exactly.
- We can write $P(x)$ as

$$P(x) = S(x)p_n(x) + R(x)$$

  $S(x)$ and $R(x)$ are polynomial of degree less than n.
- At the roots x_i of $p_n(x)$, $P(x_i) = R(x_i)$ since $p_n(x_i) = 0$.
- The integral

$$\int_{-1}^{1} P(x)\,dx = \int_{-1}^{1} S(x)p_n(x)\,dx + \int_{-1}^{1} R(x)\,dx$$

- Since $S(x)$ is of degree $n - 1$, the first term on rhs is zero due to orthogonality.
- Hence $\int_{-1}^{1} P(x)\,dx = \int_{-1}^{1} R(x)\,dx$.

Gaussian Quadrature

- Why Gaussian quadrature using degree-n Legendre polynomial on $[-1, 1]$ has DOP equal to $2n-1$?
- Consider a polynomial $P(x)$ of degree $2n-1$. GQ should integrate it exactly.
- We can write $P(x)$ as

$$P(x) = S(x)p_n(x) + R(x)$$

  $S(x)$ and $R(x)$ are polynomial of degree less than n.
- At the roots x_i of $p_n(x)$, $P(x_i) = R(x_i)$ since $p_n(x_i) = 0$.
- The integral

$$\int_{-1}^{1} P(x)\,dx = \int_{-1}^{1} S(x)p_n(x)\,dx + \int_{-1}^{1} R(x)\,dx$$

- Since $S(x)$ is of degree $n-1$, the first term on rhs is zero due to orthogonality.
- Hence $\int_{-1}^{1} P(x)\,dx = \int_{-1}^{1} R(x)\,dx$.
- Gaussian qadrature of $R(x)$ is exact because it is an polynomial of degree less than n. Hence GQ of $P(x)$ is also exact.

Differential Equation

Ordinary differential equation (ODE)

- Consider the **first order** differential equation of the form

$$y'(t) = f(t, y(t))$$

- Let's assume $t \in [a, b]$ and the **initial value** is specified, $y(a) = y_a$.
- The problem is to find the value of $y(t)$ at a given $t \in [a, b]$.

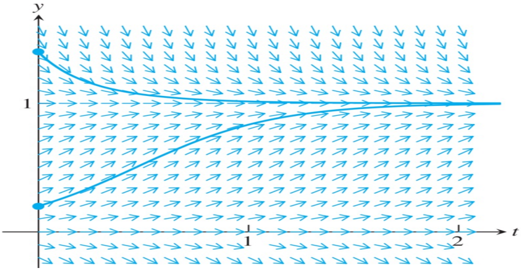
Differential Equation

Ordinary differential equation (ODE)

- Consider the **first order** differential equation of the form

$$y'(t) = f(t, y(t))$$

- Let's assume $t \in [a, b]$ and the **initial value** is specified, $y(a) = y_a$.
- The problem is to find the value of $y(t)$ at a given $t \in [a, b]$.
- We can visualize $y'(t)$ by drawing a **slope field** or **direction field**.
- Starting with an initial point, we can follow the *arrows* to the *solution* at specified t.



ODE

- **Euler's Method**:
  - Discretize the t-axis into $n+1$ equidistant grid points $t_0 < t_1 < t_2 < \ldots < t_n$. Let h be the step size.
  - Start at the initial point $y_0(t_0) = w_0$. Change in y as t changes from t_0 to t_1 is $hy'(t_0) = hf(t_0, w_0)$.

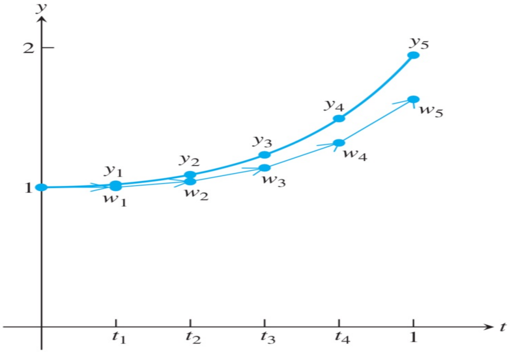
$$y(t_1) = w_1 = w_0 + hf(t_0, w_0)$$

  - For the i-th grid point, **Euler's formula**:

$$w_{i+1} = w_i + hf(t_i, w_i)$$

ODE

- **Euler's Method**:
- **Example:** $y'(t) = ty + t^3,\ y(0) = 1,\ t \in [0, 1]$.
- Take $h = 0.2$. Grid points $0, 0.2, 0.4, 0.6, 0.8, 1.0$. Do the Euler steps.



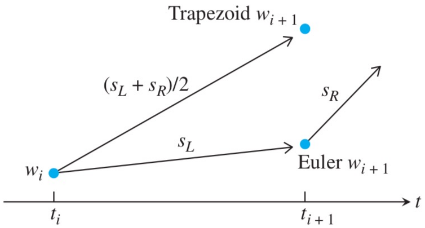
- Also shown are the true value $y(t)$ at each steps.
- Error in each step is $e_i = |y_i - w_i|$. Error $\propto h$.

ODE

- **Improved Euler' (Trapezoid) method**:
  - In Euler method, for interval $[t_i, t_{i+1}]$, the slope $y'(t)$ is taken at t_i.
  - Instead, we can take the slope to be an average of its end-point values.

$$w_{i+1} = w_i + \frac{h}{2}\left[f(t_i, w_i) + f(t_i + h, hf(t_i, w_i))\right] \rightarrow \text{Impoved Euler step.}$$

  - Error in this improved method goes as $\propto h^2$.
  - Schematic representation:

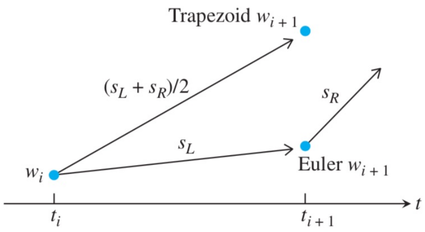


ODE

- **Improved Euler' (Trapezoid) method**:
  - In Euler method, for interval $[t_i, t_{i+1}]$, the slope $y'(t)$ is taken at t_i.
  - Instead, we can take the slope to be an average of its end-point values.

$$w_{i+1} = w_i + \frac{h}{2}\left[f(t_i, w_i) + f(t_i + h, hf(t_i, w_i))\right] \rightarrow \text{Impoved Euler step.}$$

  - Error in this improved method goes as $\propto h^2$.
  - Schematic representation:



- **Example**: Apply the Euler methods to $y'(t) = -4t^3y^2$, $y(-10) = 1/10001$, $t \in [-10, 10]$. Take $h = 10^{-1}$, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}.

ODE

- **Higher order ODE**: Consider the general n-th ODE,

$$y^{(n)} = f(t, y, y', y'', \ldots, y^{(n-1)})$$

ODE

- **Higher order ODE**: Consider the general n-th ODE,

$$y^{(n)} = f(t, y, y', y'', \dots, y^{(n-1)})$$

- To solve numerically, first define new variables

$$\begin{aligned} y_1 &= y \\ y_2 &= y' \\ &\vdots \\ y_n &= y^{(n-1)} \end{aligned}$$

- The orginal equation now becomes,

$$y_n' = f(t, y_1, y_2, \dots, y_n)$$

ODE

- Thus instead of the original n-th order ODE, we get a system of 1st order ODE:

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
y_3' &= y_4 \\
&\vdots \\
y_{n-1}' &= y_n \\
y_n' &= f(t, y_1, y_2, \ldots, y_n)
\end{aligned}
$$

ODE

- Thus instead of the original n-th order ODE, we get a system of 1st order ODE:

$$
\begin{aligned}
y_1' &= y_2 \\
y_2' &= y_3 \\
y_3' &= y_4 \\
&\vdots \\
y_{n-1}' &= y_n \\
y_n' &= f(t, y_1, y_2, \ldots, y_n)
\end{aligned}
$$

- We can think of $y_1, y_2, \ldots$ as components of a vector $\mathbf{y}$ and write the equations as a vector equation,

$$\mathbf{y}' = \mathbf{f}(t, \mathbf{y})$$

- For solution, apply Euler's method to each component independently.

ODE

- **Example:** Consider the equation of a damped Simple Pendulum,

$$\frac{d^2\theta(t)}{dt^2} + b\frac{d\theta(t)}{dt} + c\sin\theta(t) = 0$$

ODE

- **Example:** Consider the equation of a damped Simple Pendulum,

$$\frac{d^2\theta(t)}{dt^2} + b\frac{d\theta(t)}{dt} + c\sin\theta(t) = 0$$

- We can write it as $\theta'' = -c\sin\theta - b\theta'$.
- Convert it into a system of 1st order differential equation. Define $y_1 = \theta, y_2 = \theta'$. Then we have

$$\begin{aligned} y_1' =& y_2 \\ y_2' =& -c\sin\theta - b\theta' = -c\sin y_1 - by_2 \end{aligned}$$

- Take the initial conditions to be $\theta(0) = y_1(0) = \pi$, $\theta'(0) = y_2(0) = 0$.
- The i-th Euler step is given by

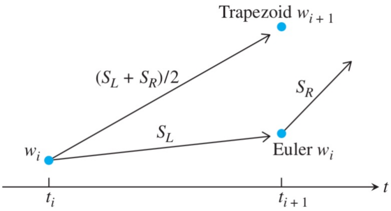
$$\begin{aligned} w_{i+1,1} =& w_{i,1} + hw_{i,2} \\ w_{i+1,2} =& w_{i,2} + h\left(-c\sin w_{i,1} - bw_{i,2}\right) \end{aligned}$$

- DEMO: ODE

ODE

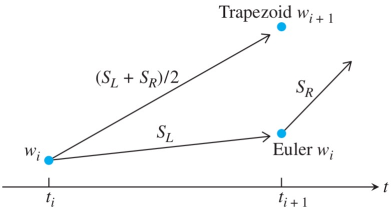
- **Runge-Kutta (RK) method**:

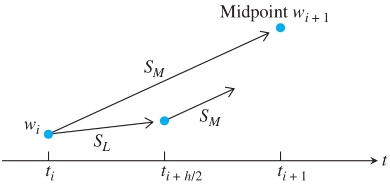
- **Runge-Kutta (RK) method**:
  - The iteration step in Euler method was $w_{i+1} = w_i + hf(t_i, w_i)$ and that for trapezoid method $w_{i+1} = w_i + \frac{h}{2}\left[f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i))\right]$. Pictorially



ODE

- **Runge-Kutta (RK) method**:
  - The iteration step in Euler method was $w_{i+1} = w_i + hf(t_i, w_i)$ and that for trapezoid method $w_{i+1} = w_i + \frac{h}{2}\left[f(t_i, w_i) + f(t_i + h, w_i + hf(t_i, w_i))\right]$. Pictorially





  - In **2nd order RK (RK2)** method, the iteration step is,

$$w_{i+1} = w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2}f(t_i, w_i)\right)$$

  - Error goes as h^2.

ODE

- **RK method of order 4 (RK4)**: Iteration step is given by,

$$w_{i+1} = w_i + \frac{h}{6}\left(s_1 + 2s_2 + 2s_3 + s_4\right)$$

  where

$$\begin{aligned}
s_1 =& f(t_i, w_i) \\
s_2 =& f(t_i + \frac{h}{2}, w_i + \frac{h}{2} s_1) \\
s_3 =& f(t_i + \frac{h}{2}, w_i + \frac{h}{2} s_2) \\
s_4 =& f(t_i + h, w_i + h s_3)
\end{aligned}$$

- **RK method of order 4 (RK4)**: Iteration step is given by,

$$w_{i+1} = w_i + \frac{h}{6}\left(s_1 + 2s_2 + 2s_3 + s_4\right)$$

where

$$\begin{aligned}
s_1 =& f(t_i, w_i) \\
s_2 =& f(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_1) \\
s_3 =& f(t_i + \frac{h}{2}, w_i + \frac{h}{2}s_2) \\
s_4 =& f(t_i + h, w_i + hs_3)
\end{aligned}$$

- Very accurate method. Error goes as h^4.

Partial Differential Equation (PDE)

- Consider differential equations of following forms

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{1}{c^2}\frac{\partial^2 u(x,t)}{\partial t^2}, \rightarrow \text{Wave equation}$$

$$\frac{\partial u(\mathbf{r},t)}{\partial t} = D\nabla^2 u(\mathbf{r},t) \rightarrow \text{Heat equation}$$

- We can use the 'finite' difference method to solve.

PDE

- Consider the heat equation in 1D,

$$\frac{\partial u(x,t)}{\partial t} = D\frac{\partial^2 u(x,t)}{\partial x^2} \Rightarrow u_t = Du_{xx}$$

- There are 2 independant variable x and t. Let the domain be $a \leq x \leq b$, $t \geq 0$.
- The problem is fully defined by specifying the initial & the boundary conditions:

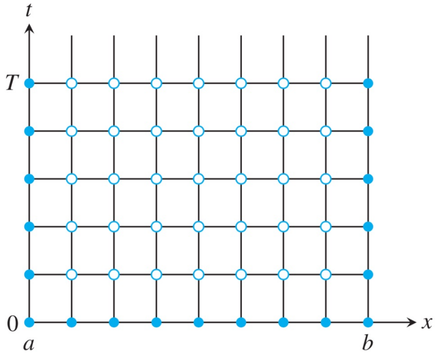
$$\begin{aligned} u(x,0) =& f(x) \quad a \leq x \leq b \\ u(a,t) =& g_1(t) \quad t \geq 0 \\ u(b,t) =& g_2(t) \quad t \geq 0 \end{aligned}$$

PDE

- **Finite difference method:**

PDE

- **Finite difference method:**
  - First we take integration domain and discretize it.



  - The closed circles are points where $u(x, t)$ is already known from initial and boundary conditions.
  - Open circles are point where we will calculate $u(x, t)$.

PDE

- The grid points are (x_i, t_j), $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. The step sizes $h = (b-a)/M$ and $k = T/N$ along x and t.

PDE

- The grid points are (x_i, t_j), $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. The step sizes $h = (b - a)/M$ and $k = T/N$ along x and t.
- Consider a point (x_i, t_j) in the mesh. At this point, let the exact solution be $u(x_i, t_)$ and approaximate solution be w_{ij}.
- Finite difference formular for 2nd derivative wrt x is

$$u_{xx}(x, t) \approx \frac{u(x + h, t) - 2u(x, t) + u(x - h, t)}{h^2}$$

- For the 1st derivative wrt time,

$$u_t(x, t) \approx \frac{u(x, t + k) - u(x, t)}{k}$$

PDE

- The grid points are (x_i, t_j), $i = 1, 2, \ldots, M$ and $j = 1, 2, \ldots, N$. The step sizes $h = (b - a)/M$ and $k = T/N$ along x and t.
- Consider a point (x_i, t_j) in the mesh. At this point, let the exact solution be $u(x_i, t_)$ and approaximate solution be w_{ij}.
- Finite difference formular for 2nd derivative wrt x is

$$u_{xx}(x,t) \approx \frac{u(x+h,t) - 2u(x,t) + u(x-h,t)}{h^2}$$

- For the 1st derivative wrt time,

$$u_t(x,t) \approx \frac{u(x,t+k) - u(x,t)}{k}$$

- The heat equation at point (x_i, t_j),

$$\begin{aligned} Du_{xx}(x,t) &= u_t(x,t) \\ D\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} &\approx \frac{w_{i,j+1} - w_{i,j}}{k} \end{aligned}$$

PDE

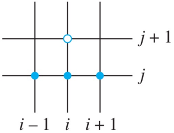
- Initial conditions specify w_{i0}, $i = 0, \ldots, M$.
- Boundary conditions specify w_{0j} and w_{Mj}, $j = 0, \ldots, N$.

PDE

- Initial conditions specify w_{i0}, $i = 0, \ldots, M$.
- Boundary conditions specify w_{0j} and w_{Mj}, $j = 0, \ldots, N$.
- We can solve the discrete version by stepping forward in time.

$$
\begin{aligned}
w_{i,j+1} =& w_{ij} + \frac{Dk}{h^2} \left(w_{i+1,j} - 2w_{i,j} + w_{i-1,j} \right) \\
=& \sigma w_{i+1,j} + (1 - 2\sigma) w_{ij} + \sigma w_{i-1,j}, \quad \sigma = Dk/h^2.
\end{aligned}
$$

- The **stencil** for the method.



- The above **forward difference** method is **explicit**.

PDE

- We can write the equation in terms of matrix notation,

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = \begin{bmatrix} 1-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 1-2\sigma & \sigma & \cdots & \vdots \\ 0 & \sigma & 1-2\sigma & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \sigma \\ 0 & \cdots & 0 & \sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{m,j} \end{bmatrix} + \sigma \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

$$m = M - 1$$

PDE

- We can write the equation in terms of matrix notation,

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = \begin{bmatrix} 1-2\sigma & \sigma & 0 & \cdots & 0 \\ \sigma & 1-2\sigma & \sigma & \cdots & \vdots \\ 0 & \sigma & 1-2\sigma & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \sigma \\ 0 & \cdots & 0 & \sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{m,j} \end{bmatrix} + \sigma \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

$$m = M - 1$$

- **Example:** Solve the heat equation for $D = 1$, initial conditions $f(x) = \sin^2 \pi x$ and boundary conditions $u(0,t) = u(1,t) = 0$ for all t. Take $h = 0.1$, $k = 0.004$.

PDE

- **Stability:** In the above example, try taking $k = 0.005$. Unstability due to error magnetification appears.
- **Von Neumann stability criteria** for the heat equation:
    - If $D > 0$, the forward difference method is stable if $\frac{Dk}{h^2} < \frac{1}{2}$.

PDE

- **Stability:** In the above example, try taking $k = 0.005$. Unstability due to error magnetification appears.
- **Von Neumann stability criteria** for the heat equation:
  - If $D > 0$, the forward difference method is stable if $\frac{Dk}{h^2} < \frac{1}{2}$.
- **Backward difference method:**
  - This time, take the backward difference formula for time derivative.

$$u_t(x,t) \approx \frac{u(x,t) - u(x,t-k)}{k}$$

PDE

- **Stability:** In the above example, try taking $k = 0.005$. Unstability due to error magnetification appears.
- **Von Neumann stability criteria** for the heat equation:
  - If $D > 0$, the forward difference method is stable if $\frac{Dk}{h^2} < \frac{1}{2}$.
- **Backward difference method:**
  - This time, take the backward difference formula for time derivative.

$$u_t(x,t) \approx \frac{u(x,t) - u(x,t-k)}{k}$$

  - Now the heat equation becomes

$$\begin{aligned} \frac{w_{ij} - w_{i,j-1}}{k} =& D\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} \\ &\Rightarrow -\sigma w_{i+1,j} + (1+2\sigma)w_{ij} - \sigma w_{i-1,j} = w_{i,j-1} \end{aligned}$$

PDE

- **Backward difference method**: The iteration equation is ($m = M - 1$)

$$\begin{bmatrix} 1-2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 1-2\sigma & -\sigma & \cdots & \vdots \\ 0 & -\sigma & 1-2\sigma & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{m,j} \end{bmatrix} = \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

- The solution:

$$w_j = A^{-1} w_{j-1} + b$$

PDE

- **Backward difference method**: The iteration equation is ($m = M - 1$)

$$\begin{bmatrix} 1-2\sigma & -\sigma & 0 & \cdots & 0 \\ -\sigma & 1-2\sigma & -\sigma & \cdots & \vdots \\ 0 & -\sigma & 1-2\sigma & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & -\sigma \\ 0 & \cdots & 0 & -\sigma & 1-2\sigma \end{bmatrix} \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{m,j} \end{bmatrix} = \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma \begin{bmatrix} w_{0,j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

- The solution:

$$w_j = A^{-1} w_{j-1} + b$$

- The backward difference method is **unconditionally stable** - stable for any h and k.

PDE

- **Boundary condition:**
  - **Dirichlet BC:** Set the value of $u(x, t)$ at the boundary. For the heat equation, fix temperatures at the boundary.

PDE

- **Boundary condition:**
  - **Dirichlet BC:** Set the value of $u(x, t)$ at the boundary. For the heat equation, fix temperatures at the boundary.
  - **Neumann BC:** Set the value of **first derivative** of $u(x, t)$ at the boundary. For the heat equation, setting $u_x(x, t) = 0$ at the boundaries mean a *insulating* or no flux boundary.

PDE

- **Boundary condition:**
  - **Dirichlet BC:** Set the value of $u(x,t)$ at the boundary. For the heat equation, fix temperatures at the boundary.
  - **Neumann BC:** Set the value of **first derivative** of $u(x,t)$ at the boundary. For the heat equation, setting $u_x(x,t) = 0$ at the boundaries mean a *insulating* or no flux boundary.
- **Example:** Solve the heat equation with homogeneous neumann BC,

$$
\begin{aligned}
u_t =& u_{xx}, \quad 0 \le x \le 1, 0 \le t \le 1 \\
u(x,0) =& \sin^2 2\pi x, \quad 0 \le x \le 1 \\
u_x(0,t) =& 0, \quad 0 \le t \le 1 \\
u_x(1,t) =& 0, \quad 0 \le t \le 1
\end{aligned}
$$

PDE

- **The wave equation:**

$$u_{tt}(x,t) = c^2 u_{xx}(x,t), \quad a \le x \le b, \quad t \ge 0$$

- Initial & boundary conditions,

$$\begin{aligned} u(x,0) =& f(x) \quad a \le x \le b \\ u_t(x,0) =& g(x) \quad a \le x \le b \\ u(a,t) =& l(t) \quad t \ge 0 \\ u(b,t) =& r(t) \quad t \ge 0 \end{aligned}$$

PDE

- **The wave equation:**

$$u_{tt}(x,t) = c^2 u_{xx}(x,t), \quad a \le x \le b, \quad t \ge 0$$

- Initial & boundary conditions,

$$\begin{aligned} u(x,0) =& f(x) \quad a \le x \le b \\ u_t(x,0) =& g(x) \quad a \le x \le b \\ u(a,t) =& l(t) \quad t \ge 0 \\ u(b,t) =& r(t) \quad t \ge 0 \end{aligned}$$

- Discretize the variables

$$x_i = a + ih, \quad i = 0, 1, \ldots, M t_i = jk, \quad i = 0, 1, \ldots, N$$

- Wave equation in terms of centered finite difference formula,

$$\frac{w_{i,j+1} - 2w_{ij} + w_{i,j-1}}{k^2} - c^2 \frac{w_{i+1,j} - 2w_{ij} + w_{i-1,j}}{h^2} = 0$$

PDE

- Write $\sigma = ck/h$ and we get the iteration step,

$$w_{i,j+1} = (2 - 2\sigma^2)w_{ij} + \sigma^2 w_{i-1,j} + \sigma w_{i+1,j} - w_{i,j-1}$$

PDE

- Write $\sigma = ck/h$ and we get the iteration step,

$$w_{i,j+1} = (2 - 2\sigma^2)w_{ij} + \sigma^2 w_{i-1,j} + \sigma w_{i+1,j} - w_{i,j-1}$$

- **Problem at first time step $j = 1$:** we need values for both $j = 0$ and $j = -1$ which we do not have.
- To get around, use the three point formula for first derivative wrt t,

$$u_t(x_i, t_j) \approx \frac{w_{i,j+1} - w_{i,j-1}}{2k}$$

- We have IC, $u_t(x_i, t_0) = g(x_i)$. Hence

$$g(x_i) = u_t(x_i, t_0) \approx \frac{w_{i1} - w_{i,-1}}{2k} \Rightarrow w_{i,-1} \approx w_{i1} - 2kg(x_i)$$

PDE

- Write $\sigma = ck/h$ and we get the iteration step,

$$w_{i,j+1} = (2 - 2\sigma^2)w_{ij} + \sigma^2 w_{i-1,j} + \sigma w_{i+1,j} - w_{i,j-1}$$

- **Problem at first time step** $j = 1$**:** we need values for both $j = 0$ and $j = -1$ which we do not have.
- To get around, use the three point formula for first derivative wrt t,

$$u_t(x_i, t_j) \approx \frac{w_{i,j+1} - w_{i,j-1}}{2k}$$

- We have IC, $u_t(x_i, t_0) = g(x_i)$. Hence

$$g(x_i) = u_t(x_i, t_0) \approx \frac{w_{i1} - w_{i,-1}}{2k} \Rightarrow w_{i,-1} \approx w_{i1} - 2kg(x_i)$$

- Using this equation, we can solve for $w_{i,1}$ to get

$$w_{i1} = (1 - \sigma^2)w_{i0} + kg(x_i) + \frac{\sigma^2}{2}(w_{i-1,0} + w_{i+1,0})$$

PDE

- To write in matrix notation, define

$$A = \begin{bmatrix} 2-2\sigma^2 & \sigma^2 & 0 & \cdots & 0 \\ \sigma^2 & 2-2\sigma^2 & \sigma^2 & \cdots & 0 \\ 0 & \sigma^2 & 2-2\sigma^2 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \sigma^2 \\ 0 & \cdots & 0 & \sigma^2 & 2-2\sigma^2 \end{bmatrix}$$

PDE

- To write in matrix notation, define

$$A = \begin{bmatrix} 2-2\sigma^2 & \sigma^2 & 0 & \cdots & 0 \\ \sigma^2 & 2-2\sigma^2 & \sigma^2 & \cdots & 0 \\ 0 & \sigma^2 & 2-2\sigma^2 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \sigma^2 \\ 0 & \cdots & 0 & \sigma^2 & 2-2\sigma^2 \end{bmatrix}$$

- Equation for $j = 1$ time step is,

$$\begin{bmatrix} w_{11} \\ \vdots \\ w_{m1} \end{bmatrix} = \frac{1}{2} A \begin{bmatrix} w_{10} \\ \vdots \\ w_{m0} \end{bmatrix} + k \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_m) \end{bmatrix} + \frac{1}{2}\sigma^2 \begin{bmatrix} w_{00} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,0} \end{bmatrix}$$

PDE

- Equation for $j > 1$ time steps is,

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = A \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} - \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma^2 \begin{bmatrix} w_{0j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

- **Stability condition**: $\sigma = ck/h < 1$ assuming $c > 0$.

PDE

- Equation for $j > 1$ time steps is,

$$\begin{bmatrix} w_{1,j+1} \\ \vdots \\ w_{m,j+1} \end{bmatrix} = A \begin{bmatrix} w_{1j} \\ \vdots \\ w_{mj} \end{bmatrix} - \begin{bmatrix} w_{1,j-1} \\ \vdots \\ w_{m,j-1} \end{bmatrix} + \sigma^2 \begin{bmatrix} w_{0j} \\ 0 \\ \vdots \\ 0 \\ w_{m+1,j} \end{bmatrix}$$

- **Stability condition**: $\sigma = ck/h < 1$ assuming $c > 0$.
- **Example:** Solve the wave equation with $c = 2, f(x) = \sin \pi x$, $g(x) = l(x) = r(x) = 0$. $0 \leq x \leq 1$, $t \leq 0 \leq 1$.

- **Stability**

Thank You.