If we choose to replace $u^{(k)}(\bar{x})$, the k^{th} derivative of u with a linear combination of function values

$$D_k u(\bar{x}) = c_1 u(x_1) + c_2 u(x_2) + \cdots + c_n u(x_n)$$

$$= c_1\left(u(\bar{x}) + u^{(1)}(\bar{x})(x_1-\bar{x}) + \frac{1}{2!}u^{(2)}(\bar{x})(x_1-\bar{x})^2 + \cdots + \frac{1}{k!}u^{(k)}(\bar{x})(x_1-\bar{x})^k + \cdots\right)$$
$$+ c_2\left(u(\bar{x}) + u^{(1)}(\bar{x})(x_2-\bar{x}) + \frac{1}{2!}u^{(2)}(\bar{x})(x_2-\bar{x})^2 + \cdots + \frac{1}{k!}u^{(k)}(\bar{x})(x_2-\bar{x})^k + \cdots\right)$$
$$+ \cdots +$$
$$+ c_k\left(u(\bar{x}) + u^{(1)}(\bar{x})(x_k-\bar{x}) + \frac{1}{2!}u^{(2)}(\bar{x})(x_k-\bar{x})^2 + \cdots + \frac{1}{k!}u^{(k)}(\bar{x})(x_k-\bar{x})^k + \cdots\right)$$
$$+ \cdots +$$
$$+ c_n\left(u(\bar{x}) + u^{(1)}(\bar{x})(x_n-\bar{x}) + \frac{1}{2!}u^{(2)}(\bar{x})(x_n-\bar{x})^2 + \cdots + \frac{1}{k!}u^{(k)}(\bar{x})(x_n-\bar{x})^k + \cdots\right)$$

This assumes we will use n-points $\{x_1, x_2, \ldots, x_n\}$ and the Taylor series of $u(x)$ exists with a remainder for the n^{th} term. Then rewrite the approximation by reordering

$$D_k u(\bar{x}) = u(\bar{x})\left(c_1 + c_2 + c_3 + \cdots + c_n\right)$$
$$+ u^{(1)}(\bar{x})\left(c_1(x_1-\bar{x}) + c_2(x_2-\bar{x}) + \cdots + c_k(x_k-\bar{x}) + \cdots + c_n(x_n-\bar{x})\right)$$
$$+ \frac{1}{2!}u^{(2)}(\bar{x})\left(c_1(x_1-\bar{x})^2 + c_2(x_2-\bar{x})^2 + \cdots + c_k(x_k-\bar{x})^2 + \cdots + c_n(x_n-\bar{x})^2\right)$$
$$+ \cdots +$$
$$+ \frac{1}{k!}u^{(k)}(\bar{x})\left(c_1(x_1-\bar{x})^k + c_2(x_2-\bar{x})^k + \cdots + c_k(x_k-\bar{x})^k + \cdots + c_n(x_n-\bar{x})^k\right)$$
$$+ \cdots +$$
$$+ \frac{1}{n!}u^{(n)}(\bar{x})\left(c_1(x_1-\bar{x})^n + c_2(x_2-\bar{x})^n + \cdots + c_k(x_k-\bar{x})^n + \cdots + c_n(x_n-\bar{x})^n\right)$$

$$+\frac{1}{(n+1)!}\left(c_1 \cdot u^{(n+1)}(\xi_1)(x_1-\bar{x})^{n+1} + c_2 u^{(n+1)}(\xi_2)(x_2-\bar{x})^{(n+1)} + \cdots + c_n u^{(n+1)}(\xi_n)(x_n-\bar{x})^{(n+1)}\right) \quad ②$$

n - error terms at same level

We can write the terms to match the terms by enforcing the following conditions. The coefficients $c_1, c_2, \ldots, c_n$ need to be chosen so that

$$D_k(u(\bar{x})) = u^{(k)}(\bar{x}) + \text{error}$$

This indicates we need

$i=0 \Rightarrow$ no dependence on $u(\bar{x})$

$\Rightarrow c_1 + c_2 + \cdots + c_n = 0$

$i=1 \Rightarrow$ no dependence on $u^{(1)}(\bar{x})$ (unless $k=1$)

$\Rightarrow c_1(x_1-\bar{x}) + c_2(x_2-\bar{x}) + \cdots + c_n(x_n-\bar{x}) = 0$

$i=2 \Rightarrow$ no dependence on $u^{(2)}(\bar{x})$ (unless $k=2$)

$\Rightarrow \frac{1}{2!}c_1(x_1-\bar{x})^2 + \frac{1}{2!}c_2(x_2-\bar{x})^2 + \cdots + \frac{1}{n!}c_n(x_n-\bar{x})^n = 0$

$\vdots$

$i=k \Rightarrow$ we need this to $= u^{(k)}(\bar{x})$

$\Rightarrow \frac{1}{k!}c_1(x_1-\bar{x})^k + \cdots + c_n(x_n-\bar{x})^k = 1$

$\vdots$

$i=n \Rightarrow \frac{1}{n!}c_1(x_n-\bar{x})^n + \cdots + \frac{1}{n!}c_n(x_n-\bar{x})^n = 0$

$\Rightarrow n > k$

So, we can write

$$\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ (x_1-\bar{x}) & (x_2-\bar{x}) & & & \\ \vdots & & & & \\ (x_1-\bar{x})^n & \cdots & & & (x_n-\bar{x})^n \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow k^{th} \text{ entry}$$

$$A \cdot \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = A^{-1} * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

There is a need for a solution routine. Matlab makes this a black box.

Code: What things should be considered?

1. Building a set of coefficients

Input: $\begin{cases} u(x) \text{ at } n \text{ points}, x_1, x_2, \ldots x_n \\ x_1, x_2, \ldots x_n \\ \bar{x} \\ k - \text{the order of the derivative} \quad k > 0. \end{cases}$

Initialize: Build A using data

$$A_{ij} = (x_j - \bar{x})^{i-1}$$

$$b_i = \delta_{i,k} = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases}$$

Kronecker delta function

Matlab to Python ⟶ import numpy as np

function c = fdcoeffV (k, xbar, x)   def fdcoeffV(k, x bar

A = ones(n,n);   A = [[1 for i in range(n)] for j in range(n)]

for i in range(n-1):

Write a code:

```
A = [[1.0 for i in range(n)] for j in range(n)]

for j in range(n):
    factor = x[j] - xbar
    for i in range(n-1):
        A[i+1][j] = A[i+1][j] * (x[j] - xbar)
b = [0.0 for i in range(n)]
b[k+1] = 1.0
```

Compute:

$c = A \backslash b$ → matlab ⇒

Simple elimination

elim

```
for k in range(n-1):
    for i in range(k+1):
        fac = A[i][k] / A[k][k]
        for j in range(k+1):
            A[i][j] = A[i][j] + fac * A[k,j]
        b[i] = b[i] - fac * b[k]
```

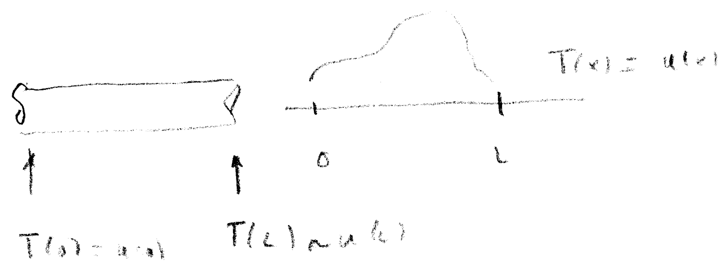
```
c = [0.0 for i in range(n)]

c[n-1] = b[n-1] / A[n-1][n-1]

for i in range(n-1)
```

?

Chapter 2:

2.1 The heat equation:

Consider the flow of heat in a rod made out of some heat conducting material. Given an initial distribution of heat in the rod we want to model how heat will flow.



The DE that can be used to model the heat flow is

$$\frac{\partial u}{\partial t}(x,t) = \frac{\partial}{\partial x}\left(K(x)\frac{\partial u}{\partial x}\right) + \psi(x,t)$$

Sources and sinks

$$\Rightarrow \boxed{u_t = (k u_x)_x + \psi}$$

This can also be used to model any dissipative phenomenon. As a first case, assume the conductivity is constant

$$\Rightarrow \quad u_t = k u_{xx}$$

We need conditions on the boundary

$$u(L,t) = g(t)$$
$$u(0,t) = h(t)$$

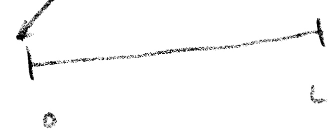
and

$u(x,0) = u_0(x) \rightarrow$ init dist.

Steady state $\Rightarrow \frac{\partial u}{\partial t} = 0$ and

$$\begin{cases} u''(x) = f(x) & 0 < x < 1 \\ u(0) = \alpha & \\ u(1) = \beta & \end{cases}$$

This is a 2-pt. boundary value problem.



Note: Having BCs does not guarantee a solution exists.

A Simple F.D. method

$$u''(x) = f(x)$$
$$u(0) = \alpha$$
$$u(1) = \beta$$

We can easily compute solution using

$$u' = \int f(x)\,dx = F(x) + C_1$$

$$\boxed{u = \int F(x)\,dx + C_1 x + C_2}$$

Fit the function to the BCs using C_1, C_2

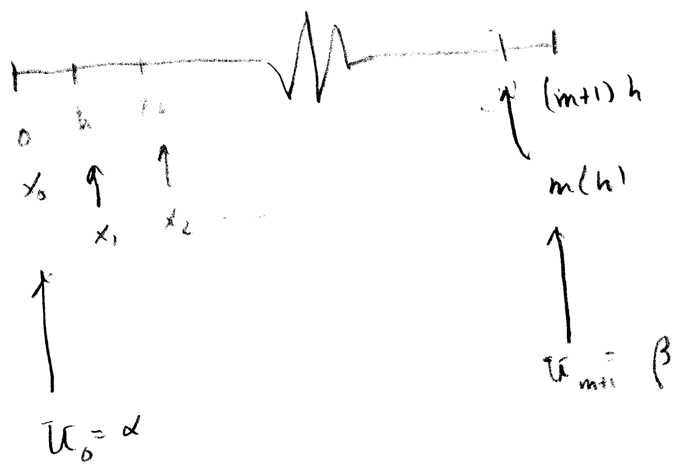
In a finite difference method we will approximate $u''(x)$ with

$$D^2 u(\bar{x}) \approx u''(\bar{x})$$

1 dim.

Suppose we assume $U_0 \approx u(x_0)$, $U_1 \approx u(x_1)$, $\dots$, $U_m \approx u(x_m)$ where $x_j = jh$

for $h = 1/(m+1)$ = mesh width



So, we want some way to compute U_j, $j = 1, 2, \dots, m$

use [illegible]

$$u'' \approx D_2^2 U_j = \frac{1}{h^2}\left(U_{j-1} - 2U_j + U_{j+1}\right)$$

$$= f(x_j)$$

Write out eqn. for each point in the mesh.

$j=1 \Rightarrow \frac{1}{h^2}(U_0 - 2U_1 + U_2) = f(x_1)$ $\Rightarrow$ [illegible]

$j=2 \Rightarrow \frac{1}{h^2}(U_1 - 2U_2 + U_3) = f(x_2)$

$j=m \Rightarrow \frac{1}{h^2}(U_{m-1} - 2U_m + U_{m+1}) = f(x_m)$