Computer Methods (MAE 3403)

Systems of Linear Equations

Motivation

- Many engineering problems can be described or approximated by linear relationships, e.g., combine resistors, small deformations of rigid structures
- Important and fundamental in numerical methods
- *n* linear, algebraic equations with *n* unknowns

Systems of linear equations

- Linear equations: unknown variables appear linearly.

$$3x_1 + 4x_2 - 3 = -5x_3$$

$$\frac{-x_1+x_2}{x_3} = 2$$

$$x_1 x_2 + x_3 = 5$$

- A system of linear equations

$$\begin{aligned} 4x + 3y - 5z &= 2 \\ -2x - 4y + 5z &= 5 \\ 7x + 8y &= -3 \\ x + 2z &= 1 \\ 9 + y - 6z &= 6 \end{aligned}$$

General m equations with n unknowns

$$
\begin{array}{rcrcccrcrcl}
a_{1,1}x_1 & + & a_{1,2}x_2 & + & \dots & + & a_{1,n-1}x_{n-1} & + & a_{1,n}x_n & = & b_1, \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & \dots & + & a_{2,n-1}x_{n-1} & + & a_{2,n}x_n & = & b_2, \\
 & & & & \dots & & \dots & & & & \\
a_{m-1,1}x_1 & + & a_{m-1,2}x_2 & + & \dots & + & a_{m-1,n-1}x_{n-1} & + & a_{m-1,n}x_n & = & b_{m-1}, \\
a_{m,1}x_1 & + & a_{m,2}x_2 & + & \dots & + & a_{m,n-1}x_{n-1} & + & a_{m,n}x_n & = & b_m.
\end{array}
$$

$$
\underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}}_{A}
\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}}_{x}
=
\underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \dots \\ b_m \end{bmatrix}}_{b}
$$

Solutions to systems of linear equations

- Given the equation $Ax = b$ (m equations, n unknowns), we have three cases
  - No solution for x, if $\text{rank}(A) + 1 = \text{rank}([A, b])$
  - Unique solution for x, if $\text{rank}([A, b]) = \text{rank}(A)$ & $\text{rank}(A) = n$
  - Infinite number of solutions for x, if $\text{rank}([A, b]) = \text{rank}(A)$ & $\text{rank}(A) < n$

How do we solve for x?

- Assume a unique solution exists for Ax = b

$$\underbrace{\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_m \end{bmatrix}}_{b}$$

- Gauss elimination
- Gauss Seidel iterative method

Gauss elimination

$$Ax = y$$

$$\begin{aligned} 4x_1 + 3x_2 - 5x_3 &= 2 \\ -2x_1 - 4x_2 + 5x_3 &= 5 \\ 8x_1 + 8x_2 &= -3 \end{aligned}$$

$$\begin{bmatrix} 4 & 3 & -5 \\ -2 & -4 & 5 \\ 8 & 8 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ -3 \end{bmatrix}$$

$$[A, y] = \begin{bmatrix} 4 & 3 & -5 & 2 \\ -2 & -4 & 5 & 5 \\ 8 & 8 & 0 & -3 \end{bmatrix}$$

Then eliminate the elements in the matrix.

- Choose a pivot equation
- Eliminate elements in other equations

Next

$$[A, y] = \begin{bmatrix} 4 & 3 & -5 & 2 \\ -2 & -4 & 5 & 5 \\ 8 & 8 & 0 & -3 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 8 & 8 & 0 & -3 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 0 & 2 & 10 & -7 \end{bmatrix} \Rightarrow \begin{bmatrix} 4 & 3 & -5 & 2 \\ 0 & -2.5 & 2.5 & 6 \\ 0 & 0 & 12 & -2.2 \end{bmatrix}$$

- Solve for $x_3 = -2.2/12$
- Substitute x_3 to the 2nd equation and solve for x_2
- Substitute x_2, x_3 to the 1st equation and solve for x_1

Code your own Gauss elimination method (Ax = b)

- Elimination Phase

$$
\begin{array}{cl}
\left[\begin{array}{cccccccccc|c}
A_{11} & A_{12} & A_{13} & \cdots & A_{1k} & \cdots & A_{1j} & \cdots & A_{1n} & b_1 \\
0 & A_{22} & A_{23} & \cdots & A_{2k} & \cdots & A_{2j} & \cdots & A_{2n} & b_2 \\
0 & 0 & A_{33} & \cdots & A_{3k} & \cdots & A_{3j} & \cdots & A_{3n} & b_3 \\
\vdots & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots \\
0 & 0 & 0 & \cdots & A_{kk} & \cdots & A_{kj} & \cdots & A_{kn} & b_k \\
\hline
\vdots & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots \\
0 & 0 & 0 & \cdots & A_{ik} & \cdots & A_{ij} & \cdots & A_{in} & b_i \\
\vdots & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots \\
0 & 0 & 0 & \cdots & A_{nk} & \cdots & A_{nj} & \cdots & A_{nn} & b_n
\end{array}\right]
&
\begin{array}{l}
\\ \\ \\ \\ \leftarrow \text{pivot row} \\ \\ \leftarrow \text{row being transformed} \\ \\ \\
\end{array}
\end{array}
$$

Elimination of row i below row k

$$A_{ij} \leftarrow A_{ij} - \lambda A_{kj}, \quad j = k, k+1, \ldots, n$$

$$b_i \leftarrow b_i - \lambda b_k$$

Range of i and k?

```
for k in range(0,n-1):
    for i in range(k+1,n):
```

Back substitution

$$\left[\mathbf{A} \mid \mathbf{b}\right] = \left[\begin{array}{ccccc|c} A_{11} & A_{12} & A_{13} & \cdots & A_{1n} & b_1 \\ 0 & A_{22} & A_{23} & \cdots & A_{2n} & b_2 \\ 0 & 0 & A_{33} & \cdots & A_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & A_{nn} & b_n \end{array}\right]$$

$$x_k = \left(b_k - \sum_{j=k+1}^{n} A_{kj} x_j\right) \frac{1}{A_{kk}}, \quad k = n-1, n-2, \ldots, 1$$

Pseudo code (no pivoting)

— *Gaussian Elimination* —

for $k = 1$ to $n-1$ do
  for $i = k+1$ to n do
    $a_{ik} = a_{ik}/a_{kk}$
    for $j = k+1$ to n do
      $a_{ij} = a_{ij} - a_{ik}a_{kj}$
    endfor
  endfor
endfor

— *Forward Elimination* —

for $k = 1$ to $n-1$ do
  for $i = k+1$ to n do
    $b_i = b_i - a_{ik}b_k$
  endfor
endfor

— *Backward Solve* —

for $i = n$ downto 1 do
  $s = b_i$
  for $j = i+1$ to n do
    $s = s - a_{ij}x_j$
  endfor
  $x_i = s/a_{ii}$
endfor

Iterative method: Gauss Seidel

- Always use the latest estimated value for each elements in x.
  - Assume initial values of $x_2, \ldots, x_n$ and then solve for x_1
  - Using the calculated x_1 and the rest of x to solve for x_2
  - Repeat the process until convergence.

Correction: Gauss Seidel for Ax = b

- In Gauss-Seidel, the computation of $x^{(k+1)}$ uses the elements of $x^{(k+1)}$ that have already been computed and only the elements of $x^{(k)}$ that have not been computed in the (k+1)-th iteration.

Input: initial values for all x_i, i=1,…,n, $x^{(0)}$

for k in range(0, max_iter):

for i in range(0, n):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

#check convergence

if converged:

break

New slide: Gauss Jacobi

- In Gauss-Seidel, the computation of $x^{(k+1)}$ uses only the elements of $x^{(k)}$ in the (k+1)-th iteration.

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n.$$

- Code up your own Jacobi method

Code your own Gauss-Seidel algorithm

- Test on the previous example

$$
\begin{aligned}
4x_1 + 3x_2 - 5x_3 &= 2 \\
-2x_1 - 4x_2 + 5x_3 &= 5 \\
8x_1 + 8x_2 &= -3
\end{aligned}
$$

- NOTE: Convergence of Gauss-Seidel requires specific conditions. A sufficient (but not necessary) condition is that A is **diagonally dominant.**

Python implementations

- Numerous and SIMPLE ways to solve systems of linear equations in Python using the numpy module
- numpy.linalg.solve (LU decomposition)

```
import numpy as np
A = np.array([[4, 3, -5],
              [-2, -4, 5],
              [8, 8, 0]])
y = np.array([2, 5, -3])
x = np.linalg.solve(A, y)
print(x)
```

- Matrix inverse:

```
A_inv = np.linalg.inv(A)
x = np.dot(A_inv, y)
```