Linear Systems

Gaussian Elimination

CSE 541
Roger Crawfis

Solving Linear Systems

- Transform *Ax* = *b* into an equivalent but simpler system.
- Multiply on the left by a nonsingular matrix: *MAx* = *Mb*:

$$x = (MA)^{-1}Mb = A^{-1}M^{-1}Mb = A^{-1}b$$

- Mathematically equivalent, but may change rounding errors

Gaussian Elimination

- Finding inverses of matrices is *expensive*
- Inverses are *not necessary* to solve a linear system.
- Some system are much easier to solve:
  - Diagonal matrices
  - Triangular matrices
- Gaussian Elimination transforms the problem into a *triangular* system

Gaussian Elimination

- Consists of 2 steps
  1. Forward Elimination of Unknowns.

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 25 & 5 & 1 \\ 0 & -4.8 & -1.56 \\ 0 & 0 & 0.7 \end{bmatrix}$$

  2. Back Substitution

Gaussian Elimination

- Systematically ***eliminate*** unknowns from the equations until only a equation with only one unknown is left.
- This is accomplished using three operations applied to the linear system of equations:
  - A given equation can be multiplied by a non-zero constant and the result substituted for the original equation,
  - A given equation can be added to a second equation, and the result substituted for the original equation,
  - Two equations can be transposed in order.

Gaussian Elimination

- Uses these elementary row operations
  - Adding a multiple of one row to another
  - Doesn't change the *equality* of the equation
    - Hence the solution does not change.
- The sub-diagonal elements are zeroed-out through elementary row operations
  - In a specific order (next slide)

Order of Elimination

$$\begin{matrix} ? & ? & ? & ? \\ 1 & ? & ? & ? \\ 2 & 4 & ? & ? \\ 3 & 5 & 6 & ? \end{matrix} = \begin{matrix} ? \\ ? \\ ? \\ ? \end{matrix}$$

Gaussian Elimination in 3D

$$\begin{matrix} 2x & + & 4y & - & 2z & = & 2 \\ 4x & + & 9y & - & 3z & = & 8 \\ -2x & - & 3y & + & 7z & = & 10 \end{matrix}$$

- Using the first equation to eliminate *x* from the next two equations

Gaussian Elimination in 3D

$$\begin{aligned} 2x + 4y - 2z &= 2 \\ y + z &= 4 \\ y + 5z &= 12 \end{aligned}$$

- Using the second equation to eliminate *y* from the third equation

Gaussian Elimination in 3D

$$\begin{aligned} 2x + 4y - 2z &= 2 \\ y + z &= 4 \\ 4z &= 8 \end{aligned}$$

- Using the second equation to eliminate *y* from the third equation

Solving Triangular Systems

- We now have a triangular system which is easily solved using a technique called ***Backward-Substitution***.

$$\begin{array}{rcrcrcl} 2x & + & 4y & - & 2z & = & 2 \\ & & y & + & z & = & 4 \\ & & & & 4z & = & 8 \end{array}$$

Solving Triangular Systems

- If A is upper triangular, we can solve $Ax = b$ by:

$$x_n = b_n / A_{nn}$$

$$x_i = \left(b_i - \sum_{j=i+1}^{n} A_{ij} x_j \right) / A_{ii}, \quad i = n-1, \ldots, 1$$

Backward Substitution

- From the previous work, we have

$$\begin{aligned} 2x + 4y - 2z &= 2 \\ y + z &= 4 \\ z &= 2 \end{aligned}$$

- And substitute z in the first two equations

Backward Substitution

$$\begin{aligned} 2x + 4y - 4 &= 2 \\ y + 2 &= 4 \\ z &= 2 \end{aligned}$$

- We can solve *y*

Backward Substitution

$$\begin{aligned} 2x + 4y - 4 &= 2 \\ y \qquad &= 2 \\ z &= 2 \end{aligned}$$

- Substitute to the first equation

Backward Substitution

$$\begin{aligned} 2x + 8 - 4 &= 2 \\ y \qquad &= 2 \\ z &= 2 \end{aligned}$$

- We can solve the first equation

Backward Substitution

$$\begin{aligned} x \qquad\qquad\qquad &= -1 \\ y \qquad\qquad &= 2 \\ z &= 2 \end{aligned}$$

Robustness of Solution

- We can measure the *precision* or *accuracy* of our solution by calculating the ***residual***:
  - Calling our computed solution x*…
  - Calculate the ***distance*** Ax* is from b
    - |Ax* – b|
- Some matrices are ***ill-conditioned***
  - A tiny change in the input (the coefficients in A) drastically changes the output (x*)

C# Implementation

```
//convert to upper triangular form
for (int k=0; k<n-1; k++)  {
      try {
            for (int i=k+1; i<n; i++) {
                  float s = a[i,k] / a[k,k];
                  for(int j=k+1; j<n; j++)
                        a[i,j] -= a[k,j] * s;
                  b[i]=b[i]-b[k] * s;
            }
      }
      catch (DivideByZeroException e)
      {
            Console.WriteLine(e.Message);
      }
}
```

```
// back substitution
b[n-1]=b[n-1] / a[n-1,n-1];
for (int i=n-2; i>=0; i--) {
      sum = b[i];
      for (int j=i+1; j<n; j++)
            sum -= a[i,j] * x[j];
      x[i] = sum / a[i,i];
}
```

Computational Complexity

- Forward Elimination

For i = 1 to n-1 { // for each equation

For j = i+1 to n { // for each target equation below the current

$$M_{ji} = \frac{A_{ji}}{A_{ii}}, A_{ji} = 0$$

$\sum_{i=1}^{n-1}(n-i) = \frac{n^2}{2}$ divisions

For k = i+1 to n { // for each element beyond pivot column

$$A_{jk} \leftarrow A_{jk} - M_{ji} A_{ik}$$

$\sum_{i=1}^{n-1}(n-i)^2 \approx \frac{2}{3}n^3$ multiply-add's

}

}

}

$O(n^3)$

Computational Complexity

- Backward Substitution

```
For i = n-1 to 1 {          //  for each equation
    For j = n  to i+1 {     //  for each known variable
        sum = sum – A_ij * x_j
    }
}
```

$\sum_{i=1}^{n-1}(n-i)=\frac{n^2}{2}$ multiply-add's

$O(n^2)$