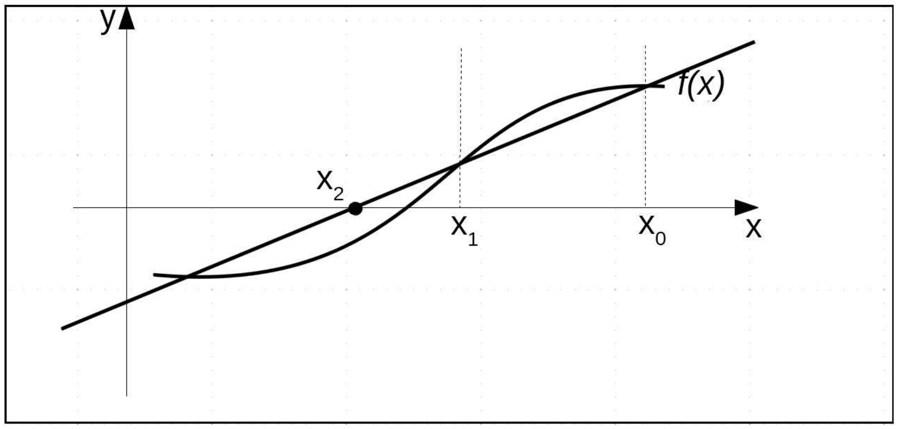
CS323 LECTURE NOTES - LECTURE 7

1 Secant Method

Newton's method has the disadvantage that we need to know the derivative $f'(x)$ in order to find a solution to $f(x) = 0$. Since the slope can be computed given two points, it should be possible to start with two initial points close to the solution, and use them to compute the line that passes through those points, as we did with the tangent in the case of Newton's Method. So start by giving two points $(x_0, f(x_0)), (x_1, f(x_1))$ and using them to find the equation of the line that passes through them. This line is a secant of $f(x)$.



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

To find the x intercept of this line we need to find the value x_2 such that $f(x_2) = 0$, so we have

$$0 - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_1)$$

And solving for x_2 we get

$$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$$

1.1 Secant Algorithm

Using the previous formula we can write an algorithm that keeps computing new values each iteration. The idea is to use the points $x_0, \ x_1$ to compute the next approximation x_2 and set the new x_0 and x_1 to x_1 and x_2 respectively to be used in the next iteration.

Secant Algorithm

```
INPUT: function $f : \mathbb{R} \to \mathbb{R}$
          $x_0, \ x_1, \ \epsilon \in \mathbb{R}, N \in \mathbb{N}$
$i \leftarrow 0$
repeat
    $x_2 \leftarrow x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)}$
    $i \leftarrow i + 1$
    $e \leftarrow |x_2 - x_1|$
    $x_0 \leftarrow x_1$
    $x_1 \leftarrow x_2$
until $e \leq \epsilon$ or $i > N$
if $i \leq N$
    return $x_1$
else
    halt with error (no solution found)
```

The convergence rate of this algorithm is given by

$$e_n = c \, e_{n-1}^{1.618}$$

Which makes it faster than bisection but slower than Newton's method.

Example:

Solve the equation

$$x^2 - 2 = 0$$

using the Secant Method with $x_0 = 1$, $x_1 = 2$, $\epsilon = 10^{-5}$, $N = 10$

x0	x1	f(x0)	f(x1)	x2	error	Error < epsilon ?
1.000000	2.000000	-1.000000	2.000000	1.333333	0.666667	
2.000000	1.333333	2.000000	-0.222222	1.400000	0.066667	
1.333333	1.400000	-0.222222	-0.040000	1.414634	0.014634	
1.400000	1.414634	-0.040000	0.001190	1.414211	0.000423	
1.414634	1.414211	0.001190	-0.000006	1.414214	0.000002	yes

2 Systems of Linear Equations

A system of linear equation is a collection of m linear equations with m unknowns. In general it can be written as

$$\begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n & = & b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n & = & b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n & = & b_3 \\ \vdots & \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n & = & b_n \end{array}$$

For example

$$\begin{array}{rcl} 3x_1 + 2x_2 & = & 5 \\ 6x_1 - x_2 & = & 1 \end{array}$$

is a system of two linear equations with two unknowns.

2.1 Matrix Representation

Any system of linear equations can be represented in matrix form as:

$$A\bar{x} = \bar{b}$$

where A is an $m \times n$ matrix containing the coefficients of the system of equations, $\bar{b}$ is the vector of independent terms and $\bar{x}$ is the solution of the system.

In the example given above:

$$\begin{pmatrix} 3 & 2 \\ 6 & -1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

Where

$$A = \begin{pmatrix} 3 & 2 \\ 6 & -1 \end{pmatrix}$$

and

$$\bar{b} = \begin{pmatrix} 5 \\ 1 \end{pmatrix}$$

A linear system has 3 possible solution sets:

- unique solution
- no solution
- infinite number of solutions

Geometrically, each equation represents a $n-1$ dimension hyperplane in a vector space of n dimensions. The solution corresponds to the points where all the hyperplanes intersect.

Therefore, when writing our algorithms to solve systems of linear equations, it is important to identify the possible solution sets. In this course we will be interested only in those cases where a unique solution might be found, therefore we will only consider the case where the number of equations is equal to the number of unknowns.

2.2 Matrix Row Operations (Elementary Operations)

In order to solve a linear system we can perform some operations on the equations that do not modify the solution (linear algebra course):

1. Multiply an equation by a constant
2. Add an equation multiplied by a constant to another equation
3. Swap two equations (now equation i is equation j, and equation j is equation i)

We can mimic these operations using the matrix representation described above. Notice that these operations must be performed on the entire equation, including the independent term. So it is useful, from the algorithmic point of view, to represent the system of equations as an **augmented matrix**:

$$\left(\begin{array}{ccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} & b_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} & b_n \end{array}\right)$$

and perform the following operations on rows corresponding to the cases above:

1. $R_i = cR_i$
2. $R_i = R_i + cR_j$
3. $R_i \leftrightarrow R_j$

Since the result of applying matrix row operation yields an equivalent augmented matrix (one with the same solution as the original) we will use the following notation to represent the application of the operations applied to matrix A_1 and producing an equivalent augmented matrix A_2

$$A_1 \overrightarrow{operation} A_1$$

For example, we add row 1 times -4 to row 2 of the first matrix to get the second one:

$$\left(\begin{array}{cc|c} 1 & 5 & -1 \\ 4 & 3 & 2 \end{array}\right) \overrightarrow{R_2 = R_2 + (-4)R_1} \left(\begin{array}{cc|c} 1 & 5 & -1 \\ 0 & 17 & 6 \end{array}\right)$$

2.3 Gaussian Elimination

2.3.1 Strategy

Notice that if we are able to transform the original augmented matrix into an upper triangular matrix, i.e. all entries below the main diagonal are equal to 0, it would be equivalent to solving the following system of equations

$$\begin{array}{ccccccccc} a_{1,1}x_1 & + & a_{1,2}x_2 & + & \dots & + & a_{1,n}x_n & = & b_1 & \text{...........} & (1) \\ & & a_{2,2}x_2 & + & \dots & + & a_{2,n}x_n & = & b_2 & \text{...........} & (2) \\ & & & & \vdots & & & & \vdots & & \\ & & & & & & a_{n,n}x_n & = & b_n & \text{...........} & (\text{n}) \end{array}$$

Notice that equation n can be easily solved to obtain x_n, and the x_n is substituted in equation $n-1$ to obtain x_{n-1}, and so on. This process is called **backward substitution**.

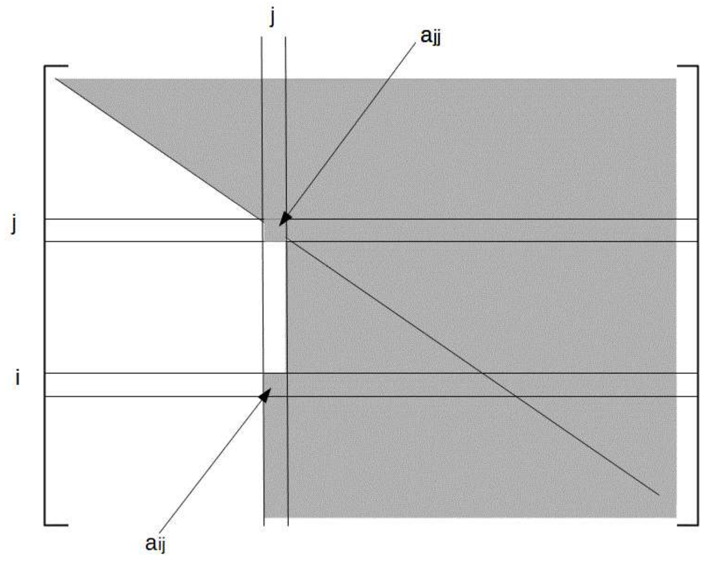
So far our method to solve a linear system involves two steps:

1. Use Row Matrix Operators to transform the original augmented matrix into an upper triangular matrix
2. Use the backward substitution method to find the solution

2.4 Gaussian Elimination 1.0

The idea to transform the original augmented matrix into an upper triangular matrix is the following:

```
For every column
    Perform the Row Matrix Operations necessary
    to make all entries below the main diagonal equal to 0
```



The figure represents a matrix while being processed. Suppose that all columns before column j have 0 in all entries under the main diagonal (are not shaded in the figure), and that we have successfully converted to 0 all entries under the main diagonal of column j and above row i. Our goal is to perform a row matrix operation in order to make element a_{ij} equal to 0. Notice that since row i has 0's in all columns from 1 to i, the following operation will do the trick!

$$R_i = R_i - \left(\frac{a_{i,j}}{a_{j,j}} \right) R_j$$

So we have our algorithm we just have to proceed in order, noticing that we do not need to make 0's in the last column of the original matrix.

Algorithm: Gaussian Elimination,
Input: Augmented matrix a_{ij} $(1 \le i \le n,\ \ 1 \le j \le n+1)$

```
for $j = 1$ to $n-1$
    for $i = j+1$ to $n$
        // Do Matrix Row operation (affects entire row, including column $n+1$)
        $R_i = R_i - \left( \frac{a_{i,j}}{a_{j,j}} \right) R_j$
```

Since the number of additions/subtractions/products required for each row operation is in $O(n)$, the total running time of this algorithm is in $O(n^3)$.

2.4.1 Pivoting

Before we continue with the next step (backward substitution) notice that it is possible for a_{jj} to be equal to zero, in which case we would get an error. In this case what we can do is look for a row k below j ($k > j$) where $a_{kj} \neq 0$ so that we can swap row k with row j and continue with the algorithm. If no such row exists, that means that column j has only 0's from row i to row n. In this case we could skip to the next column, and start making 0's, until eventually we would end up with a matrix where the the last row has only 0's.

From linear algebra we recall that if this is the case, then one of the original equations must be a linear combination of the other ones, and depending on the value of the independent terms, the system will have either no solution or an infinite number of solutions, but it will not have a unique solution.

Since in our case we are only interested in finding unique solutions, we will just throw an exception: `not_unique_solution`.

There is another problem, it might happen that the non-zero value that is found to swap with, is very small, might even be 0 but due to floating point errors, it is not exactly equal to 0. Using this very small value, will cause a loss of significance error, since we will be diving by it and adding it to another value!

The solution to this problem is not just to find the first value that is not zero under the main diagonal, but to find the value with the maximum absolute value, i.e. we will find a row k such that

$$|a_{k,j}| = max\{|a_{j,j}|, |a_{j+1,j}|, \ldots, |a_{n,j}|\}$$

and then swap row j with row k.

This technique is called *partial pivoting* or *maximum pivoting*. We can now put everything together and write version 2 of our Gaussian Elimination algorithm, including backward substitution.

Algorithm: Gaussian Elimination,
Input: Augmented matrix a_{ij} $(1 \leq i \leq n,\ \ 1 \leq j \leq n+1)$
Output: Solution vector x_i $(1 \leq i \leq n)$

```
for $j = 1$ to $n - 1$
    // find the matrix with largest absolute value under main diagonal
    let $k$ be such that
        $|a_{k,j}| = max\{|a_{j,j}|, |a_{j+1,j}|, \ldots, |a_{n,j}|\}$
    Swap Row $j$ with row $k$
    if $(a_{j,j} = 0)$
        Throw not_unique_solution exception
    for $i = j + 1$ to $n$
        // Do Matrix Row operation (affects entire row, including column $n + 1$)
        $R_i = R_i - \left(\frac{a_{i,j}}{a_{j,j}}\right) R_j$
// Backward Substitution
$x_n = \frac{a_{n,n+1}}{a_{n,n}}$
for $i = n - 1$ down to 1
    $x_i = \frac{a_{i,n+1} - \sum_{k=i+1}^{n} a_{i,k} x_k}{a_{i,i}}$
return $(x_1, x_2, \ldots, x_n)$
```