Iterative Algorithms I:
Elementary Iterative Methods and the Conjugate Gradient Algorithms

By:- Nitin Kamra
Indian Institute of Technology, Delhi
Advisor:- Prof. Ulrich Reude

Outline

1. Introduction to Linear Systems
   I. Cramer's Rule

2. Direct Methods to solve Linear Systems
   I. Gaussian Elimination Method (GEM)
   II. LU Factorization

3. Linear Stationary Iterative Methods
   I. Jacobi Method
   II. Gauss-Siedel Method
   III. Over-Relaxation Methods (JOR) and (SOR)

4. Non-stationary Iterative Methods
   I. Method of Steepest Descent
   II. Conjugate Gradient Algorithm

Introduction to Linear Systems

- Aim:- To solve for x in the linear system which is of the form:

$$Ax = b$$

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Order of the system = n

- Matrix A should be non-singular for unique solution to exist.

Cramer's Rule

- Solution provided by Cramer's Rule:

$$x_i = \Delta_i \,/\, \det(A)$$

  Δ_i = Determinant of the matrix obtained by substituting the i-th column of A by the vector b.
- Essentially equivalent to inverting the matrix A and getting $x = A^{-1}b$.

- Computational effort: O((n+1)!) flops as determinants are evaluated recursively.
- A computer able to perform 10^9 flops per second would need 9.6×10^{47} years to solve a linear system of order 50.

- Need alternate methods for solving linear systems.

Direct Methods to solve Linear Equations

- Solve a set of linear equations in a finite number of steps.
- Based around triangular matrices and the fact that any system of the form:

$$\begin{bmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

  can be solved using Forward Substitution as follows:

$$x_1 = \frac{b_1}{l_{11}}; \qquad x_i = \frac{1}{l_{ii}}\left(b_i - \sum_{j=1}^{i-1} l_{ij}x_j\right) \quad for\ i = 2,3,\dots,n$$

- The argument can be extended for upper triangular matrices with Backward Substitution.
- Takes n^2 flops of computation.

Gaussian Elimination Method (GEM)

- Aims at reducing the original system Ax = b to an equivalent system (having the same solution x) of the form Ux = β, where U is an upper-triangular matrix and β is the updated right hand side matrix.
- New system can be solved with Backward Substitution.
- Let the original system be $A^{(1)}x = b^{(1)}$,

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & a_{1n}^{(1)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1}^{(1)} & a_{n2}^{(1)} & \dots & a_{nn}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ \vdots \\ b_n^{(1)} \end{bmatrix}$$

- Introduce the multipliers m_{i1},

$$m_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, 3, \dots, n,$$

Gaussian Elimination Method (GEM)

- Now define,

$$a_{ij}^{(2)} = a_{ij}^{(1)} - m_{i1}a_{1j}^{(1)}, \quad i,j = 2,\ldots,n,$$

$$b_{i}^{(2)} = b_{i}^{(1)} - m_{i1}b_{1}^{(1)}, \quad i = 2,\ldots,n,$$

- We get the new system, $A^{(2)}x = b^{(2)}$ as shown:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \ldots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{bmatrix}$$

- Now x_1 has been removed from all the equations except the first one.
- Similar procedure can be repeated for x_2, …, x_n and the left hand matrix can be reduced to an equivalent upper-triangular matrix.

Gaussian Elimination Method (GEM)

- The final upper-triangular system $A^{(n)}x = b^{(n)}$ OR $Ux = \beta$ looks like:

$$\begin{bmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & \vdots \\ 0 & & & \ddots & \vdots \\ 0 & & & & a_{nn}^{(n)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1^{(1)} \\ b_2^{(2)} \\ \vdots \\ \vdots \\ b_n^{(n)} \end{bmatrix}$$

$m_{ik} = \dfrac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \dots, n.$	$a_{ij}^{(k+1)} = a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)}, \quad i, j = k+1, \dots, n$ $b_i^{(k+1)} = b_i^{(k)} - m_{ik} b_k^{(k)}, \quad i = k+1, \dots, n.$

- Now we can solve the above system with Back Substitution Technique.

- Cost of computation: $(2n^3/3 + n^2) + n^2$ flops = $O(n^3)$

Gaussian Elimination Method (GEM): Applicability

- Applicable only if all the coefficients m_{ik} are defined i.e. $a_{ii}^{(k)}$ should be non-zero. This can be guaranteed for:
  1) Matrices diagonally dominant by rows.
  2) Matrices diagonally dominant by columns.
  3) Symmetric and Positive Definite Matrices.

- Positive Definiteness: Matrix A ϵ $C^{n \times n}$ is said to be positive definite in C^n iff for any non-zero vector x, (Ax, x) is real and positive.
- Diagonally Symmetric by rows:

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^{n} |a_{ij}|, \quad \text{with } i = 1, \ldots, n$$

- Diagonally Symmetric by columns:

$$|a_{ii}| \geq \sum_{j=1, j \neq i}^{n} |a_{ji}|, \quad \text{with } i = 1, \ldots, n$$

LU Factorization

- Write A = LU, where L = Lower triangular matrix with diagonal entries all equal to 1, and U = Upper triangular matrix.

$$Ax = b \quad \rightarrow \quad LUx = b$$

$$\begin{bmatrix} 1 & 0 & \dots & 0 \\ l_{21} & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

- Similar to GEM and the entries of L and U can be obtained while carrying out GEM.

- Now assume Ux = w ➔ Lw = b and solve for w using Forward Substitution. Next solve for x from Ux = w using Backward Substitution.
- Computational complexity same as of GEM.
- Useful for system of equations where matrix A remains same and solution is needed for different vectors b.

Linear Iterative Methods

- Basic idea: To form a sequence of vectors $x^{(k)}$ that enjoys the convergence:

$$x = \lim_{k \to \infty} x^{(k)}$$

- Aim: To employ vectors $x^{(k)}$, which should be cheaply generated per iteration and should quickly converge to x in a small number of iterations.

- Formally, iterative methods give the solution of the equation Ax = b after infinite iterations, but in practice they are stopped for minimum value of n such that $|| x^{(n)} - x || < \epsilon$, where ϵ is a fixed tolerance.

- If we denote $e^{(k)} = x^{(k)} - x$, the error at the k^{th} step of iteration, then the condition for convergence amounts to the requirement: $0 = \lim_{k \to \infty} e^{(k)}$ for any choice of the initial guess $x^{(0)}$.

Linear Stationary Iterative Methods

- Linear iterative methods: $x^{(k+1)} = Bx^{(k)} + f, \ k \geq 0$
- B (n×n matrix): called the 'iteration matrix' and f: vector obtained from b.

- In order to obtain B and f, we generally split A into two parts:
  A = P – N, where P is called the Pre-conditioning matrix.

➔ $Ax = b$

➔ $Px^{(k+1)} = Nx^{(k)} + b$

➔ $x^{(k+1)} = P^{-1}Nx^{(k)} + P^{-1}b$

➔ $B = P^{-1}N$ and $f = P^{-1}b$.

- P should be non-singular and easy to invert.

Jacobi Method

- Write A as the sum of D(diagonal elements) and R(remainder)

$$A = D + R \text{ such that } D = \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} \text{ and } R = \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}$$

- Then $x^{(k+1)} = D^{-1}(b - Rx^{(k)})$

- The algorithm can be stated as:
  1. Choose $x^{(0)}$.
  2. Repeat till convergence {

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

  }

Gauss – Siedel Method

- Write A as the sum of D(Diagonal matrix), L*(Lower-triangular matrix) and U*(strictly Upper-triangular matrix)

$$A = D + L^* + U^* \text{ where}$$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \quad L^* = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix} \quad U^* = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

- Then $x^{(k+1)} = (D + L^*)^{-1}(b - U^* x^{(k)})$

- Differs from Jacobi method only in the fact that it uses the already calculated elements of $x^{(k+1)}$ to update the remaining elements per iteration.

Gauss – Siedel Method

- The algorithm can be stated as:
  1. Choose $x^{(0)}$.
  2. Repeat till convergence {

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

  }

- Pros:
  - In general, results in faster convergence compared to Jacobi Method.
  - No need to store previous values of $x_i^{(k)}$ after calculating $x_i^{(k+1)}$, so saves memory.
- Cons:
  - Components of $x^{(k)}$ can't be updated in parallel as in Jacobi Method.

Over-Relaxation Methods (JOR) & (SOR)

- JOR = Jacobi Over-Relaxation and SOR = Successive Over-Relaxation
- Generalizations of Jacobi and Gauss-Siedel Methods respectively.

- Define a relaxation parameter = ω (>0)
- We have ωAx = ωb

- For JOR, put A = D + R,
- The update relation gets modified as:

$$x^{(k+1)} = D^{-1}(\omega b - \omega R x^{(k)} - (\omega - 1) D x^{(k)})$$

- Element – wise:

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

Over-Relaxation Methods (JOR) & (SOR)

- This can be expressed as: $x_{JOR}^{(k+1)} = x^{(k)} + \omega(x_J^{(k+1)} - x^{(k)})$

- Similarly for SOR, put A = D +L*+ U*,
- The update relation gets modified as:

$$x^{(k+1)} = (D + L^*)^{-1}(\omega b - \omega U^* x^{(k)} - (\omega - 1)(D + L^*)x^{(k)})$$

- Element – wise:

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

- This can be expressed as: $x_{SOR}^{(k+1)} = x^{(k)} + \omega(x_{GS}^{(k+1)} - x^{(k)})$

- So, in both JOR and SOR, ω scales the correction term to be added in each update and thereby controls the rate of convergence.
- ω>1 means over-relaxation and ω<1 means under-relaxation.

Convergence Results for Jacobi, Gauss-Siedel and Over-Relaxation Methods

- If A is a strictly Diagonally Dominant Matrix by rows, the Jacobi and Gauss-Siedel methods are convergent.
- If A and 2D – A are symmetric and positive definite matrices then Jacobi Method is convergent.
- If A is symmetric and positive definite, Gauss-Siedel method is monotonically convergent w.r.t. the norm $||\cdot||_A$.
- If the Jacobi method is convergent then the JOR method converges if $0<\omega\leq 1$.
- For SOR method to converge, $0<\omega<2$.

Non-Stationary Iterative Methods

- Stationary Iterative Methods: Jacobi, Gauss – Siedel, GOR, SOR. These had a relaxation (acceleration) parameter ω, independent of the current iteration.

- Non-stationary Iterative Methods involve acceleration parameters which change every iteration.

- Examples:-
  - Method of Steepest Descent (Gradient Method),
  - Conjugate Gradient Algorithm

- These are best suited for large matrices with many null entries per row (Sparse Matrices).
- They might be the only methods available for Non-Linear systems.

Method of Steepest Descent

- We'll study the methods for a symmetric, positive definite matrix A i.e. $A = A^T$ and $x^TAx>0$ for all non-zero vectors x.
- Consider the quadratic form energy function: $f(x) = \frac{1}{2}x^T A\,x - b^T x + c$
- The gradient of the function is given by: $f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b = Ax - b$
- The extremum of this function occurs when: $f'(x) = 0 \Rightarrow Ax = b$
- We'll use this example: $A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}, c = 0$ which has solution $x = \begin{bmatrix} 2 \\ -2 \end{bmatrix}$
- Consider any arbitrary point p and $x = A^{-1}b$. Then for symmetric matrix A,

$$f(p) = f(x) + \frac{1}{2}(p-x)^T A(p-x)$$

- Further, for positive definite matrix A, this says that $f(p) \geq f(x)$, hence x is the minima.
- So, to solve Ax = b, we have to minimize the energy function f(x).

Method of Steepest Descent

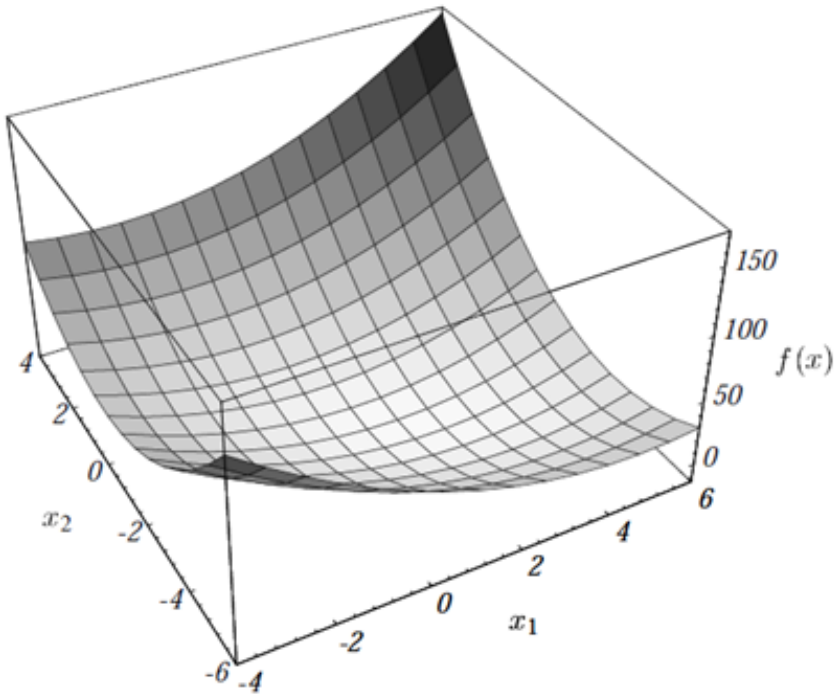

Fig 1: Graph of quadratic form f(x). The minima is the solution to Ax = b

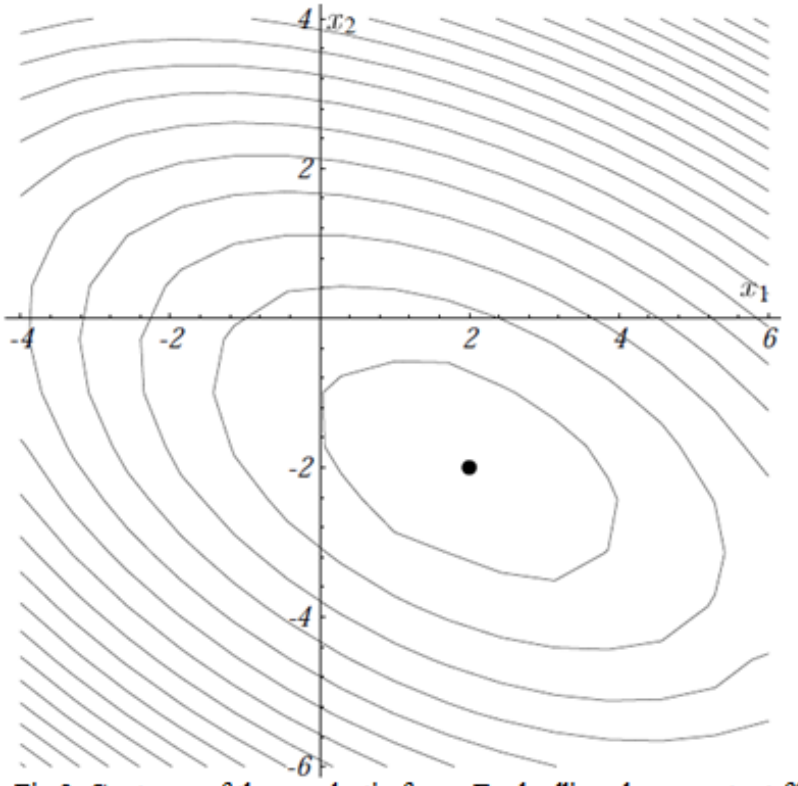

Fig 2: Contours of the quadratic form. Each ellipse has constant f(x).

Method of Steepest Descent

- We start at an arbitrary point $x^{(0)}$ and move towards the direction of steepest descent.
- At any point, *error* $e^{(i)} = x^{(i)} - x$.
- Direction of steepest descent at any point $x^{(i)}$ is given by:

  r*esidual* $r^{(i)} = - f'(x^{(i)}) = b - Ax^{(i)} = -Ae^{(i)}$.
- Step size = $\alpha_{(i)}$. It is variable and decided every iteration to minimize error by moving in the direction of steepest descent. For minimizing error, we have

$$\frac{\partial f\left(x^{(i+1)}\right)}{\partial \alpha} = 0 \text{ which leads to the condition } f'\left(x^{(i+1)}\right) \text{is normal to } r^{(i)}.$$

$$\text{Using } f'\left(x^{(i+1)}\right) = -r^{(i+1)} \text{ and solving for } \alpha^{(i)}, \text{ we get } \alpha^{(i)} = \frac{r^{(i)T} r^{(i)}}{r^{(i)T} A r^{(i)}}$$

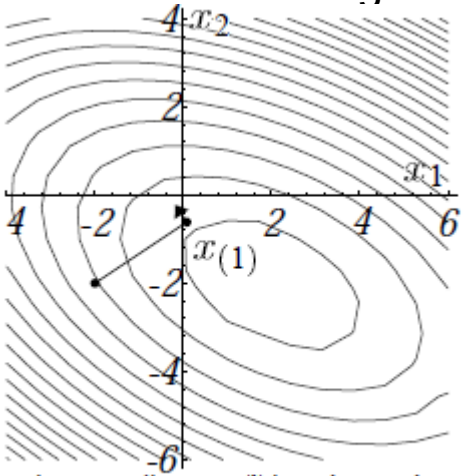

Fig 3: Gradient at $x^{(1)}$ is orthogonal to gradient at $x^{(0)}$.

Method of Steepest Descent

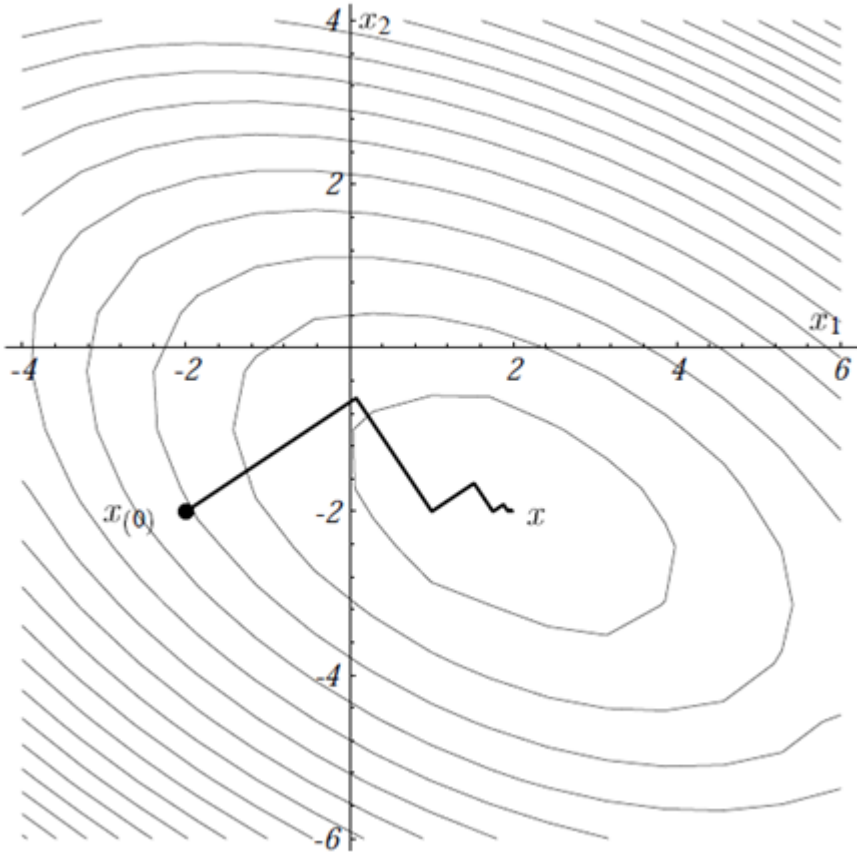

Fig 4: Steepest Descent starts at $[-2.-2]^T$ and converges at $[2,-2]^T$.

Algorithm:-

1) Choose $x^{(0)}$.
2) Find $r^{(0)} = b - Ax^{(0)}$
3) repeat till convergence {

$$\alpha^{(i)} = \frac{r^{(i)T} r^{(i)}}{r^{(i)T} A r^{(i)}}$$

$x^{(i+1)} = x^{(i)} + \alpha^{(i)} r^{(i)}$

$r^{(i+1)} = r^{(i)} - \alpha^{(i)} A r^{(i)}$

}

Motivation for Conjugate Gradient

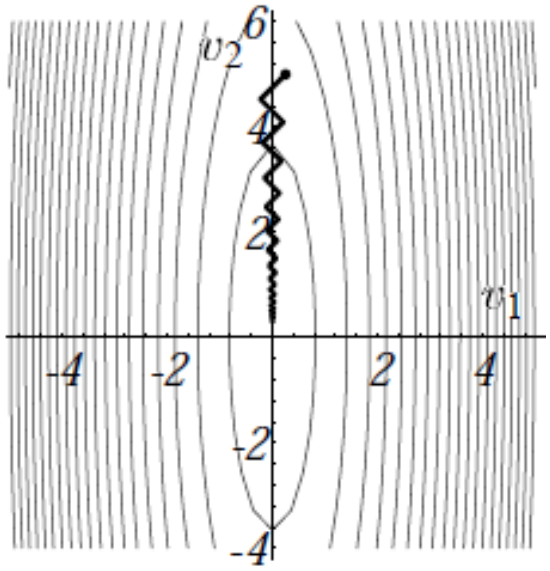

Fig 5: Oscillations in Steepest Descent

- Method of Steepest Descent can take too many iterations to converge.
- As the sequence $x^{(k)}$ approaches x, the step-size $\alpha^{(k)}$ decreases and the convergence slows down.

- **Motivation for conjugate Gradient**: Instead of probing repeatedly in the same direction, if we could identify ‘n’ orthogonal directions in which we could move only once each the correct amount and end up at the right position, convergence would be very quick.

Motivation for Conjugate Gradient

- Let the orthogonal search directions be $d^{(0)}$, $d^{(1)}$, …, $d^{(n-1)}$.
- Then $x^{(i+1)} = x^{(i)} + \alpha^{(i)} d^{(i)}$. For each step $e^{(i+1)}$ is perpendicular to $d^{(i)}$.
- Using this condition

$$d^{(i)T} e^{(i+1)} = 0$$

$$d^{(i)T} \left(e^{(i)} + \alpha^{(i)} d^{(i)} \right) = 0$$

$$\alpha^{(i)} = -\frac{d^{(i)T} e^{(i)}}{d^{(i)T} d^{(i)}}$$

- But this implies that to know $\alpha^{(i)}$, we have to know $e^{(i)}$, which is like already knowing the solution x.

Conjugate Gradient Algorithm: Method of Conjugate Directions

- Solution is to make the search directions A-orthogonal or conjugate i.e. for any two directions $d^{(i)}$ and $d^{(j)}$, $d^{(i)T}Ad^{(j)} = 0$.

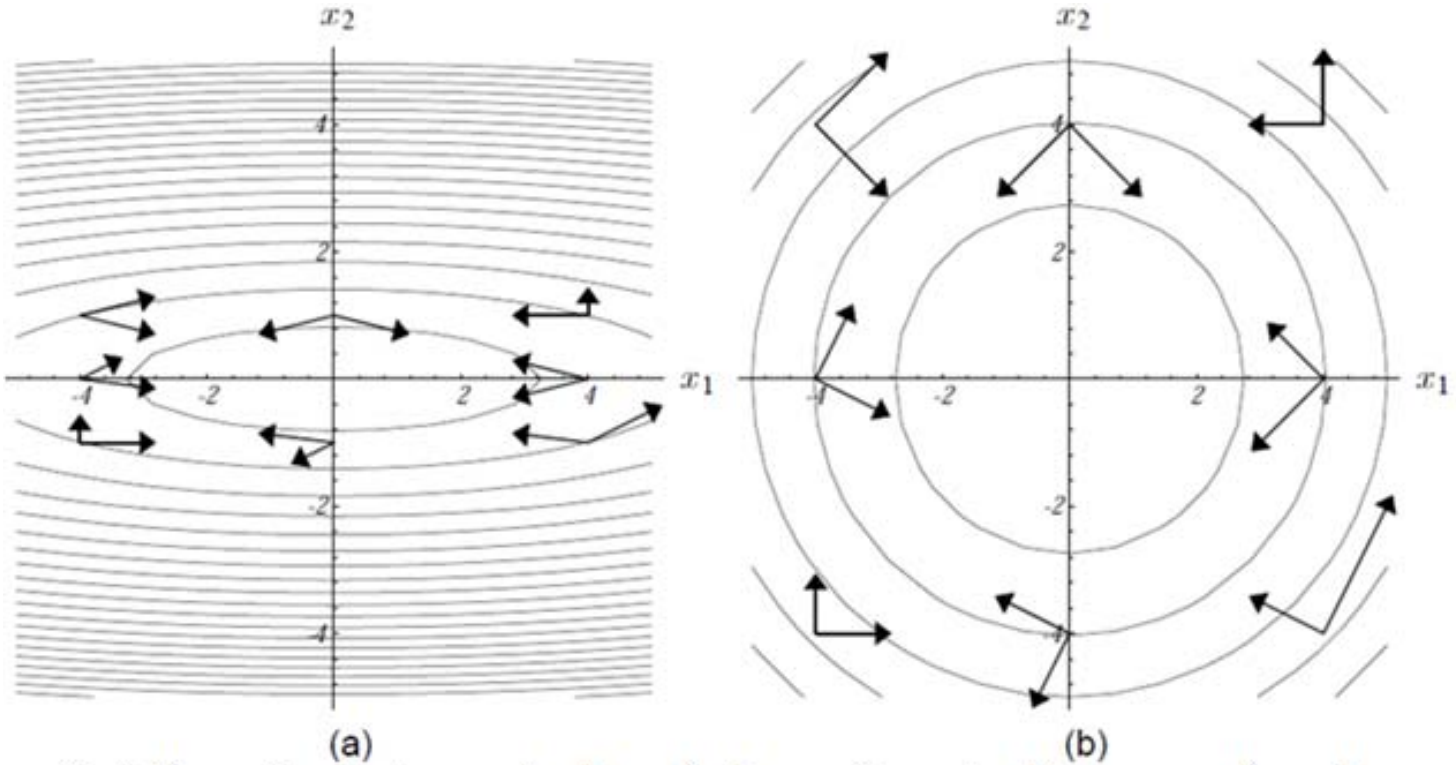

Fig 6: These pairs of vectors are A-orthogonal ... because these pairs of vectors are orthogonal.

Conjugate Gradient Algorithm: Method of Conjugate Directions

- Our new requirement is $e^{(i+1)}$ be A-orthogonal to $d^{(i)}$, which is equivalent to finding the minimum point along the search direction $d^{(i)}$.

- Using the above condition, we solve for $\alpha^{(i)}$,

$$\alpha^{(i)} = -\frac{d^{(i)T} r^{(i)}}{d^{(i)T} A d^{(i)}}$$

- With the above value of $\alpha^{(i)}$, it can be proved that if initial error $e^{(0)}$:

$$e^{(0)} = \sum_{j=0}^{n-1} \delta_j d^{(j)}$$

then $\alpha^{(i)} = -\delta^{(i)}$ which means that each step cancels a component of error in one of the search directions $d^{(i)}$. Hence after n-steps all the components of error have been nulled down and so we achieve exact convergence to x.

Conjugate Gradient Algorithm: Gram Schmidt Conjugation

- How to generate the A-orthogonal directions $\{d^{(i)}\}$?
- Gram Schmidt Process generates $\{d^{(i)}\}$ from a set of 'n' linearly independent vectors: $\{u^{(0)}, u^{(1)}, \ldots, u^{(n-1)}\}$. (For now the co-ordinate axes will do.)

$$d^{(0)} = u^{(0)}$$

- To construct $d^{(i)}$, take $u^{(i)}$ and subtract out any components that are not A-orthogonal to the previous (i-1) d-vectors.

$$d^{(i)} = u^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)}$$

Conjugate Gradient Algorithm: Gram Schmidt Conjugation

- B_{ik} can be found by ensuring A-orthogonality of each $d^{(i)}$ with each $d^{(j)}$ as:

$$\beta_{ij} = -\frac{u^{(i)T} A d^{(j)}}{d^{(j)T} A d^{(j)}} \text{ for } i > j$$

- Problem: Takes $O(n^3)$ computation. Can turn into gaussian elimination. Hence the method's true power remained dormant till the CG algorithm was discovered.

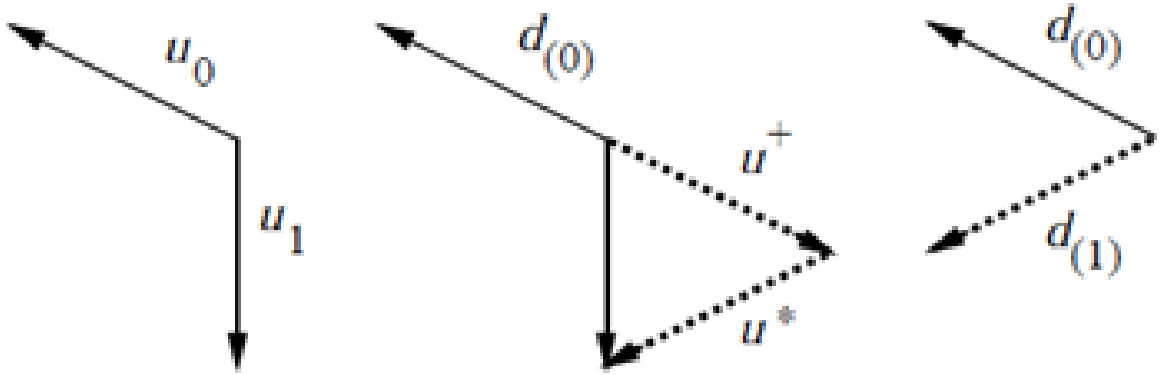

Fig 7: Gram-Schmidt conjugation of two vectors

Method of Conjugate Directions: Example

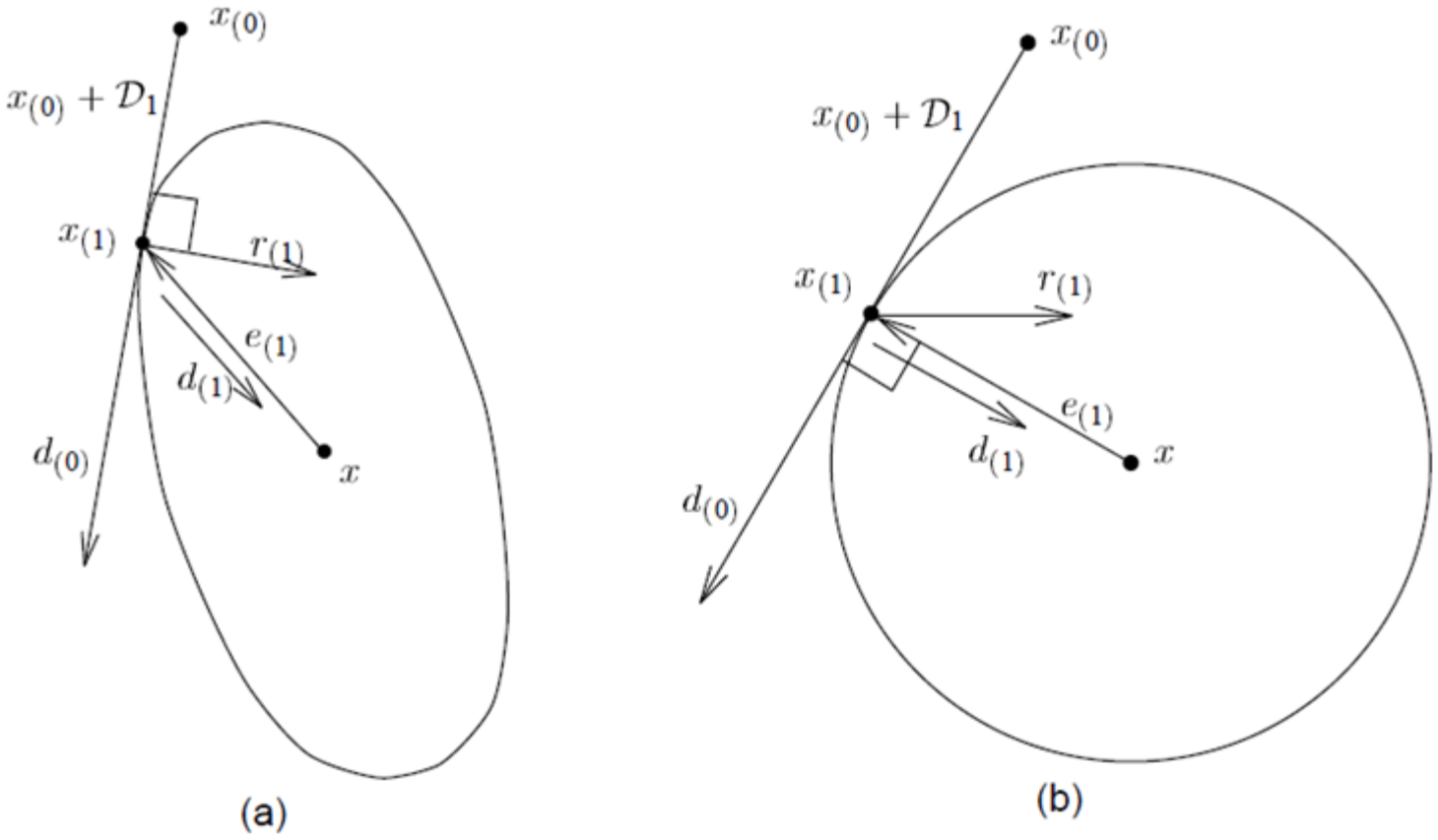

Fig 8: A visualization of the method of Conjugate-Directions

Method of Conjugate Directions: Observations

1. Residual at the j-th iteration is orthogonal to all the previous directions.

   $d^{(i)T} r^{(j)} = 0 \; for \; i < j$

   As $r^{(j)} = -Ae^{(j)}$, and $e^{(j)}$ is A-orthogonal to d(i) for all i<j.

2. As span {d(0), d(1), … d(j-1)} = span {u(0), u(1), … , u(j-1)}, Residual at the j-th iteration is orthogonal to all previous u(i)'s as well.

   $u^{(i)T} r^{(j)} = 0 \; for \; i < j$

3. $d^{(i)T} r^{(i)} = u^{(i)T} r^{(i)}$ due to the way Gram-Schmidt process works.

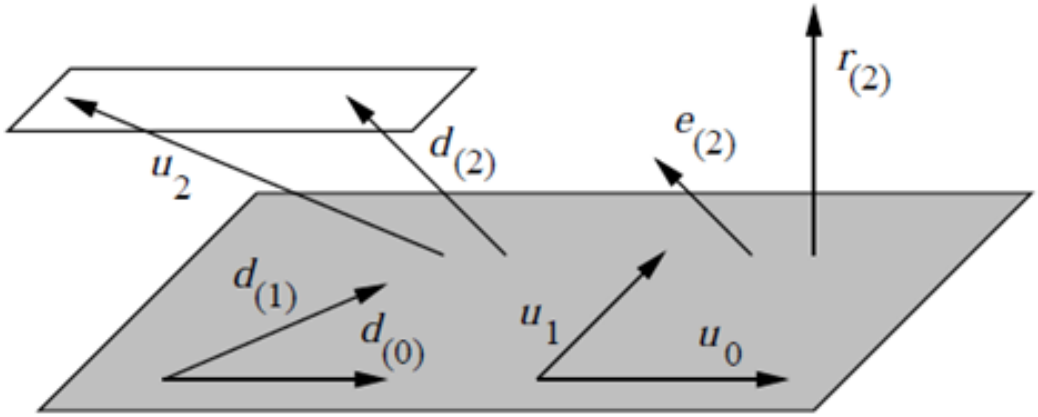

Fig 9: Summary of the observations from Method of Conjugate Directions

Conjugate Gradient Algorithm

- It is simply the method of Conjugate Directions with the directions constructed from the conjugation of the residuals i.e. $u^{(i)} = r^{(i)}$.
- Advantages:-
  - As $r^{(i)}$ is orthogonal to all previous search directions, its guaranteed to always produce a new linearly independent search direction, unless $r^{(i)} = 0$ in which case the problem has already been solved.
  - As $r^{(i)}$ is orthogonal to all previous $u^{(j)}$'s, its orthogonal to all previous residuals now.
- Now observe that: $r^{(i+1)} = -Ae^{(i+1)} = -A\left(e^{(i)} + \alpha^{(i)} d^{(i)}\right) = r^{(i)} - \alpha^{(i)} Ad^{(i)}$
- So, $r^{(i+1)}$ is a linear combination of $r^{(i)}$ and $Ad^{(i)}$.

  $r^{(i)}$ is a linear combination of $r^{(i-1)}$ and $Ad^{(i-1)}$

  $r^{(i-1)}$ …..
- As $r^{(i+1)}$ is orthogonal to all previous residues, this means that it is A-orthogonal to all previous search directions except $d^{(i)}$.

Conjugate Gradient Algorithm

- This simplifies the β_{ij} coefficients tremendously as now, there exists only 1 such non-zero coefficient for every iteration. So, β_{ij} is non-zero only for j=i-1.

- Hence we call it $\beta_{(i)} = \beta_{i,i-1}$ which is now given by:

$$\beta_{(i)} = \frac{r^{(i)T} r^{(i)}}{r^{(i-1)T} r^{(i-1)}}$$

ALGORITHM:

Choose $x^{(0)}$.

$$d^{(0)} = r^{(0)} = b - Ax^{(0)}$$

Repeat for $i = (0: n-1)$ {

$$\alpha^{(i)} = -\frac{r^{(i)T} r^{(i)}}{d^{(i)T} A d^{(i)}}$$

$$x^{(i+1)} = x^{(i)} + \alpha^{(i)} d^{(i)}$$

$$r^{(i+1)} = r^{(i)} - \alpha^{(i)} A d^{(i)}$$

$$\beta_{(i+1)} = \frac{r^{(i+1)T} r^{(i+1)}}{r^{(i)T} r^{(i)}}$$

$$d^{(i+1)} = r^{(i+1)} + \beta_{(i+1)} d^{(i)}$$

}

Summary

1. Understood what a linear system is, and Cramer's rule to solve it.

2. Cramer's rule computationally very expensive - O((n+1)!), so we studied Direct Solvers like Gaussian Elimination or LU factorization – $O(n^3)$.

3. Next we explored stationary iterative techniques like the Jacobi method and Gauss-Siedel methods, also their generalized versions JOR and SOR.

4. Further, learnt about non-stationary iterative techniques like Method of Steepest Descent and the most popular Conjugate gradient algorithm, which gives exact solution within 'n' iterations.

Thank you!!!

Questions ???

References

1. Numerical Mathematics – Alfio Quarteroni, Riccardo Sacco, Fausto Saleri.
2. An Introduction to the Conjugate Gradient Method without the Agonizing Pain, Edition 1(1/4) – Jonathan Richard Shewchuk, August 4, 1994.

Figures:-

All the figures have been taken from:

1. An Introduction to the Conjugate Gradient Method without the Agonizing Pain, Edition 1(1/4) – Jonathan Richard Shewchuk, August 4, 1994.