Iterative Methods for Linear Systems

Iterative methods begin with an initial estimate for the solution and successilvely improve it until the solution is within a desired level of accuracy. The following are some advantageous to iterative methods:

1) Only require n^2 operations compared to n^3 for direct methods.
2) Algorithms are designed to reduce rounding errors and at each iteration the computational error can be improved upon.

Stationary Iterative Methods

- most iterative methods come under the class of stationary iterative methods.
- Here the method has the form $x_{k+1} = Gx_k + c$ for solving $Ax = b$. Here G and c are constants throughout all iterations and thus is said to be "stationary".
- The choice of matrix G and vector c determines the type of iterative method.
- To obtain a suitable matrix G, the matrix A can be split into

$$A = M - N \quad , \text{ where } M \text{ is nonsingular}$$

and thus $G = M^{-1}N$ and $c = M^{-1}b$. since it satisfies the following equation.

$$Ax = b$$

$$(M-N)x = b.$$

$$(M-N)x = Mx - Nx = b.$$

From $x_{k+1} = G x_k + C = M^{-1}N x_k + M^{-1}b$

$$M^{-1}x_{k+1} = N x_k + b$$

Then at iteration $(k+1)$,

$$M x_{k+1} - N x_{k+1} = b$$

$$(N x_k + b) - N x_{k+1} = b$$

$$N x_k - N x_{k+1} = 0$$

$$N(x_k - x_{k+1}) = 0$$ since $N \neq 0$, then as $k \to \infty$, $x_k - x_{k+1} \to 0$.

- $G = M^{-1}N$ is also known as the Jacobian matrix.
- The method is convergent if $\rho(G) < 1$ where $\rho(G)$ is the spectral radius of matrix G, where the spectral radius is the largest $|$eigenvalue$|$. — largest absolute eigenvalue.

Jacobi method

- If $M = D$ and $N = -(L+U)$, we have the Jacobi method. Here D is a diagonal matrix.
- Then $x_{k+1} = -D^{-1}(L+U)x_k + D^{-1}b$
- L and U are the strict lower and upper triangular portions of A and not the LU factorization of A.
- In algorithm form, for each component of x

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

- The method requires double the storage since $x_i^{(k+1)}$ and $x_i^{(k)}$ must be stored.
- convergence rate is unacceptably low.

Gauss-Seidel method

- The Gauss-Seidel method is similar to Jacobi but takes into consideration of the new information available
- Here M is defined as $M = D+L$ and $N = -U$
- The algorithm can be written as,

$$x^{(k+1)} = -(D+L)^{-1}Ux^{(k)} + (D+L)^{-1}b$$

OR in component form,

$$x_i^{(k+1)} = \frac{b_i - \sum_{j<i} a_{ij}x_j^{(k+1)} - \sum_{j>1} a_{ij}x_j^{(k)}}{a_{ii}}, \quad i = 1, \ldots, n$$

- The advantageos of the Gauss-Seidel approach : first, faster convergence and duplicate storage is not ~~allowed~~ needed, since the most latest information available for $x^{(k+1)}$ is employed.
- The disadvantage would be the updating $x^{(k+1)}$ must be done successively from i to i+1 row since the latest informat is used. This is unlike the Jacobi method, where each row can be updated simultaneously using parallel calculation

Successive Over-Relaxation (SOR)

- The convergence rate of the Gauss-Seidel method can be accelerated by a technique called successive over-relaxation (SOR)

- The algorithm can be written as,

$$x^{(k+1)} = x^{(k)} + \omega \left(x_{GS}^{(k+1)} - x^{(k)} \right)$$

$$= (1-\omega) x^{(k)} + \omega x_{GS}^{(k+1)}$$

  - The method amounts to taking a weighted average of the current iterate and the next Gauss-Seidel iterate.
  - ω is the relaxation parameter and is chosen to accelerate the convergence. $\omega > 1$ provides over-relaxation, $\omega < 1$ gives under-relaxation, and $\omega = 1$ gives the original Gauss-Seidel method.

- In matrix notation,

$$M = D + \omega L \quad , \quad N = (1-\omega) D - \omega U$$

and can be written as,

$$x^{(k+1)} = x^{(k)} + \omega \left[D^{-1} (b - L x^{(k+1)} - U x^{(k)}) - x^{(k)} \right]$$

- To solve and update one ~~the~~ just sweeps (forward) from i=1 through n, repeatly.

Symmetric SOR (SSOR)

- Here at each iteration, a forward and then a backward sweep through the unknowns are performed.
- This method can be slightly faster depending on the problem.

Optimization Approach to solving Linear systems

- The previous methods are based on the stationary iterative approach. In this section we discuss the optimization approach to solving linear systems.

- First let us review, the basics of quadratic functions and then we will illustrate the conjugate gradient approach.

- Quadratic Functions

  → Solving optimization problems require approximating objective functions using linear or higher-order polynomials.

  → Linear Functions are simple to use but do not have second derivatives

  → Quadratic functions is the next simplest function and it has continuous second derivatives.

  → Function is in quadratic form if it is written as

  : $f(\vec{x}) = \vec{x}^T A \vec{x} = \sum_{i,j=1}^{n} a_{ij} x_i x_j$

  where we assume that A is a symmetric square matrix.

  : A is = positive-definite, if $f(\vec{x}) > 0$, $x \in \mathbb{R}^n$ except zero

  positive-semidefinite, if $f(\vec{x}) \geq 0$

  negative-definite, if $f(\vec{x}) < 0$, $x \in \mathbb{R}$ except zero

  → An arbitrary symmetric matrix is positive definite if and only if each of its principal submatrices has a positive determinant. This is knowns as the Sylvester Criterion

→ Example if $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, then

$$f(\vec{x}) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \text{where } a_{12} = a_{21}$$

$$= \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_{11}x_1 + a_{12}x_2 \\ a_{21}x_1 + a_{22}x_2 \end{bmatrix}$$

$$= a_{11}x_1^2 + a_{12}x_1x_2 + a_{21}x_1x_2 + a_{22}x_2^2$$

$$= a_{11}x_1^2 + 2a_{12}x_1x_2 + a_{22}x_2^2$$

→ A quadratic function with n-variables is generally given in the form of,

$$f(\vec{x}) = \frac{1}{2}\vec{x}^T A \vec{x} + b^T \vec{x} + c$$

- where A is a symmetric matrix, b is a vector containing the coefficients of x_i, and c is a scalar.
- the gradient or first derivative of $f(\vec{x})$ is $\nabla f(\vec{x}) = A\vec{x} + b$
- the Hessian (second partial derivative) is

$$H(\vec{x}) = A \quad \text{or} \quad \nabla^2 f(\vec{x}) = H(\vec{x}) = A$$

$$H(\vec{x}) = \begin{bmatrix} \frac{\partial^2 f(\vec{x})}{\partial x_1^2} & \cdots & \frac{\partial^2 f(\vec{x})}{\partial x_1 \partial x_n} \\ \vdots & & \vdots \\ \frac{\partial^2 f(\vec{x})}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(\vec{x})}{\partial x_n \partial x_n} \end{bmatrix}$$

→ A Taylor series expansion about the point $\vec{x}_0$ is

$$f(\vec{x}) = f(\vec{x}_0) + \nabla^T f(\vec{x}_0)(\vec{x} - \vec{x}_0) + \frac{1}{2}(\vec{x}-\vec{x}_0)^T H(\vec{x}_0)(\vec{x}-\vec{x}_0) + \dots\dots$$

If the third- and higher-order terms are neglected, the Taylor series expansion of any function with a continuous second derivative is similar to a quadratic function.

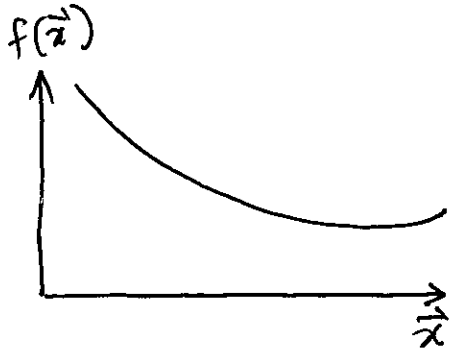
The implication of this is that any function behaves as a quadratic function within a small domain.

→ If any two vectors, $\vec{x}_1$ and $\vec{x}_2 \in \mathbb{R}^n$ and

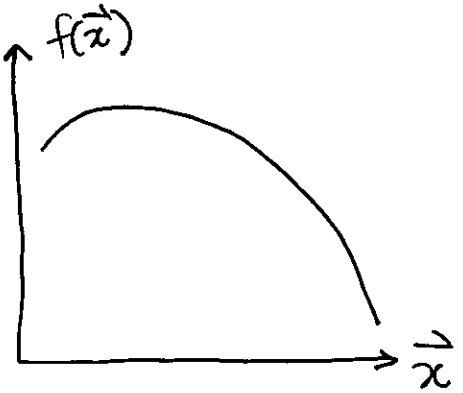
$$f(\theta\vec{x}_1 + (1-\theta)\vec{x}_2) \leq \theta f(\vec{x}_1) + (1-\theta) f(\vec{x}_2)$$

, where θ is a scalar with range $0 \leq \theta \leq 1$, then $f(\vec{x})$ is said to be convex over the set I

- $-f(\vec{x})$ is concave if $f(\theta\vec{x}_1 + (1-\theta)\vec{x}_2) \geq \theta f(\vec{x}_1) + (1-\theta)f$
- If $H(\vec{x})$ is positive-definite or -semi-definitive, then $f(\vec{x})$ is convex.

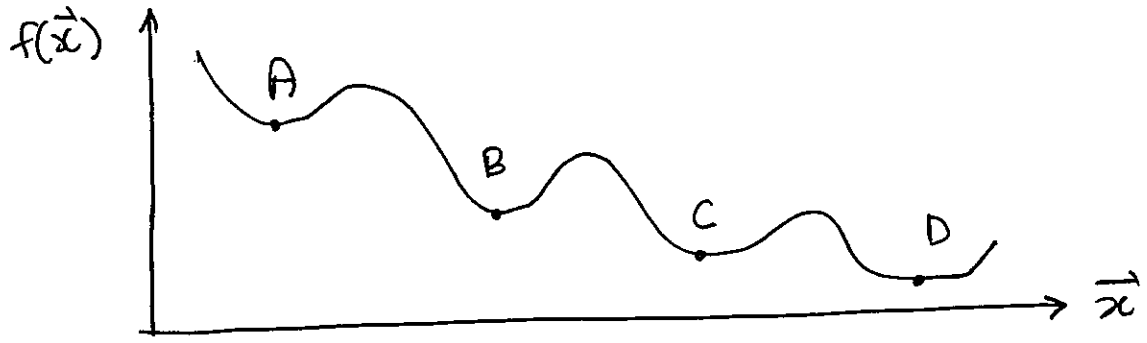


convex function



concave function

- local and global minimum or maximum



- $f(A)$, $f(B)$, $f(C)$, and $f(D)$ are <u>local</u> minimums within a specified range.

- $f(A) > f(B) > f(C) > f(D)$, therefore D is a <u>global</u> minimum point.

- Necessary and Sufficient Conditions for Unconstrained Minimum

- Definition : <u>strong minimum point</u>

$f(\vec{x})$ has a strong minimum at $\vec{x}^*$ if there exists a scalar $\delta > 0$ such that $f(\vec{x}^*) < f(\vec{x}^* + \Delta\vec{x})$ and for all $\Delta\vec{x}$, $\quad 0 < \| \Delta\vec{x} \| \leq \delta$.

- Definition : <u>weak minimum point</u>

$\vec{x}^*$ is a weak minimum point if $f(\vec{x}^*) \leq f(\vec{x}^* + \Delta\vec{x})$

- Consider a Taylor series expansion of $f(\vec{x})$ about $\vec{x} + \Delta\vec{x}$

$$f(\vec{x}^* + \Delta\vec{x}) = f(\vec{x}^*) + \Delta\vec{x}^T \nabla f(\vec{x}^*) + \frac{1}{2} \Delta\vec{x}^T \nabla^2 f(\vec{x}^*) \Delta\vec{x} + \dots$$

→ To have a weak minimum at the point $\vec{x}^*$, we at least need $f(\vec{x}^*) = f(\vec{x}^* + \Delta\vec{x}^*)$

→ Therefore from the Taylor series expansion, for $f(\vec{x}^* + \Delta\vec{x}) = f(\vec{x}^*)$, we at least need

$\Delta\vec{x}^T \nabla f(\vec{x}^*) = 0$, which implies $\nabla f(\vec{x}^*) = 0$

→ For a strong minimum $f(\vec{x}^* + \Delta x) > f(\vec{x}^*)$, we require $\Delta\vec{x}^T \nabla^2 f(\vec{x}^*) \Delta\vec{x} > 0$

since $\nabla f(\vec{x}^*) = 0$

This implies that $\nabla^2 f(\vec{x}^*) > 0$.

- We can now state, the "Necessary and Sufficient conditions for an unconstrained minimum",

Theorem: If a point $\vec{x}^*$ is to be a local minimum of a function $f(\vec{x})$, which is differentiable at $\vec{x}^*$, then

① A necessary condition is that $\vec{x}^*$ be a stationary point, $\nabla f(\vec{x}^*) = 0$
(also called first-order optimality condition)

② A sufficient condition is that the Hessian matrix is positive-definite at $\vec{x}^*$, $\nabla^2 f(\vec{x}^*) > 0$
(also called second-order optimality condition)

How to choose Δx?

There are two potential strategies for unconstrained optimization problems:
1) Line Search methods 2) Trust Region methods.

Line Search methods

In order to locate the local minimizer x^*, a line search method update the current position at x_k via the following simple formula

$$x_{k+1} = x_k + \alpha_k p_k \quad \text{where} \quad \alpha_k p_k = \Delta x_k$$

and p_k is the search direction, also known as descent direction, α_k is the step length.

→ $p_k = -\nabla f_k$ is the most obvious choice for a descent direction. At point x_k, $\nabla f_k(x_k)$ is the gradient or for a linear or quadratic function it is easy to imagine that ∇f_k is the slope at that point. Therefore if $p_k = -\nabla f_k$, then you are moving x_k downwards to x_{k+1} along the slope by a factor α_k. The larger α_k, the larger the descent. In short p_k tells you the direction of descent and α_k provides the magnitud

→ From the Taylor series expansion

$$f(x_k + \Delta x_k) = f(x_k + \alpha_k p_k) = f(x_k) + \nabla f(x_k)^T \alpha_k p_k + \frac{1}{2}\alpha_k^2 p_k^T \nabla^2 f p_k$$

If $\alpha_k = 1$, then

$$f(x_k + p_k) = f(x_k) + \nabla f_k^T p_k + \dots .$$

if $p_k = -\nabla f_k$, then $\nabla f_k^T p_k = p_k^T \nabla f_k = -\|\nabla f\|^2$

Therefore $f(x_k + p_k) < f(x_k)$ The function is guranteed to be reduced.

In general, p_k can be written as, $p_k = -B_k^{-1} \nabla f_k$

If where B_k is a symmetric and non singular matrix

If $B_k = I$, the identity matrix, then $p_k = -\nabla f_k$
This is the most basic form of p_k and the method is called line searches.

If $B_k = \nabla^2 f_k$, then $p_k = -\dfrac{\nabla f_k}{\nabla^2 f_k}$

and $x_{k+1} = x_k - \alpha_k \dfrac{\nabla f_k}{\nabla^2 f_k}$

This is called the Newton method.

We can still show that $p_k = -B_k^{-1} \nabla f_k$ is a descent direction by repeating the above step.

$$p_k^T \nabla f_k = -(B_k^{-1} \nabla f_k)^T \nabla f_k = -\nabla f_k^T B_k^{-1} \nabla f_k$$

since B is positive definite, then $-\nabla f_k^T B_k^{-1} \nabla f_k < 0$.

Therefore $f(x_k + \alpha_k p_k) < f(x_k)$.

Steepest Descent Method

- This method is the simplest of all the schemes, where the search direction, p_k is set to the negative of the gradient, $\nabla f(x_k)$
- Each search direction is orthogonal to the previous value. $p_{k+1} \perp p_k$. Therefore the method "zigzags" in the design space. and is rather inefficient.

$$\frac{df(x_{k+1})}{d\alpha} = 0 \qquad \text{(Find } \alpha \text{ such that } f(x_k + \alpha p_k) \text{ is minimized).}$$

$$\frac{\partial f(x_{k+1})}{\partial x_{k+1}} \cdot \frac{\partial x_{k+1}}{\partial \alpha} = 0 \qquad x_{k+1} = x_k + \alpha p_k$$

$$\frac{\partial x_{k+1}}{\partial \alpha} = p_k$$

$$\nabla f(x_{k+1})^T \cdot p_k = 0$$

since $p_k = -\nabla f(x_k)$, then $\quad -\nabla f(x_{k+1})^T \nabla f(x_k) = 0$

The product of the two vectors is zero, so the seach direction at the k+1 iterate is orthogonal to the value at the previous iterate, ~~k~~ k.

- The algorithm is guaranteed to converge p_k is the same direction of the gradient. The angle between the two vectors are zero. However it may take an infinite number of iterations.
- The rate of convergence is linear.

Conjugate Gradient Method (CG)

- Suppose we want to find the minimum of a convex quadratic function

$$f(x) = \frac{1}{2}x^T Q x - b^T x$$

where Q is an $n \times n$ matrix that is symmetric and positive definite. The gradient of the function $f(x)$ can then be written as,

$$\nabla f(x) = Qx - b \equiv r(x)$$

- If we solve the linear system, $Qx = b$, then $\nabla f(x) = 0$ and the function will be minimized.

- Thus the conjugate gradient method is an iterative method for solving linear systems.

- An important property of this method is its ability to generate a set of vectors with a property known as conjugacy.

A set of nonzero vectors $\{p_0, p_1, \ldots, p_{n-1}\}$ is conjugate with respect to matrix Q if

$$p_i^T Q p_j = 0, \text{ for all } i \neq j.$$

- Any set of vectors that satisfy this property is linearly independent.

- Given a starting point $x_0 \in \mathbb{R}^n$ and a set of conjugate directions $\{p_0, p_1, \ldots, p_{n-1}\}$, let us generate the sequence $\{x_k\}$ by setting

$$x_{k+1} = x_k + \alpha p_k$$

where α_k is the one-dimensional minimizer of the quadratic function $f(x)$ along $x_k + \alpha p_k$, given explicitly by

$$\alpha_k = - \frac{r_k^T p_k}{p_k^T Q p_k}.$$

for any x_0, the sequence $\{x_k\}$ generated by the CG algorithm converges to the solution of the linear system in at most n steps.

- Since the vectors p_k are linearly independent and span the whole space n, we can write the following,

$$x^* - x_0 = \sigma_0 p_0 + \sigma_1 p_1 + \ldots\ldots + \sigma_{n-1} p_{n-1}.$$

Here x^* is the solution and x_0 the initial solution point. σ_k are some choice of scalars.

We can solve for σ_k by premultiplying the above equation by $p_k^T Q$, $0 \leq k < n$, to obtain

$$p_k^T Q (x^* - x_0) = \sigma_k p_k^T Q p_k$$

$$\sigma_k = \frac{p_k^T Q (x^* - x_0)}{p_k^T Q p_k}$$

From the equation $x_{k+1} = x_k + \alpha_k p_k$,

we can write x_k as a function of all previous Δx by the following equation,

$$x_k = x_0 + \alpha_0 p_0 + \cdots + \alpha_{k-1} p_{k-1}$$

premultiplying by $p_k^T Q$,

$$p_k^T Q (x_k - x_0) = \alpha p_k^T Q p_i = 0$$

(the right hand-side is equal to 0 by the conjugacy property)

Therefore $p_k^T Q (x^* - x_0) - p_k^T Q (x_k - x_0)$

$$= p_k^T Q x^* - p_k^T Q x_k$$
$$= p_k^T Q (x^* - x_k)$$
$$= p_k^T (Q x^* - Q x_k)$$
$$= p_k^T (b - Q x_k)$$
$$= -p_k^T r_k$$

Dividing the equations by $p_k^T Q p_k$ we get,

$$\frac{p_k^T Q (x^* - x_0)}{p_k^T Q p_k} = - \frac{p_k^T r_k}{p_k^T Q p_k}$$

$$\sigma_k = \alpha_k \quad \text{(from previous page).}$$

The CG Algorithm

- The algorithm computes the search directions which are conjugate directions as the algorithm progresses. At each step, the search direction is computed as a linear combination of the previous direction and the current gradient and thus satisfies the condition that all the directions are mutually conjugate.

- The algorithm:

① set $k=0$, select initial point $x^{(0)}$.

② $g^{(0)} = \nabla f(x^{(0)})$.
If $g^{(0)}=0$, then stop.
else set $p^{(0)} = -g^{(0)}$.

③ $$\alpha_k = \frac{-r^{(k)T} p^{(k)}}{p^{(k)T} Q p^{(k)}}.$$

④ $$x^{k+1} = x^k + \alpha_k p^{(k)}.$$

⑤ $r^{(k+1)} = \nabla f(x^{k+1})$. If $r^{(t+1)} = 0$, then stop.

⑥ $$\beta_k = \frac{r^{(k+1)T} Q p^{(k)}}{p^{(k)T} Q p^{(k)}} = \frac{r^{T(t+1)} r^{(k+1)}}{r^{(k)T} r^{(k)}}$$

⑦ $$p^{(k+1)} = -r^{(k+1)} + \beta_k p^{(k)}.$$

Example

Minimize, $f(x_1, x_2, x_3) = \frac{3}{2}x_1^2 + 2x_2^2 + \frac{3}{2}x_3^2 + x_1x_3 + 2x_2x_3 - 3x_1 - x_3$.

We can represent f as

$$f(x) = \frac{1}{2}x^T Q x - x^T b.$$

$$Q = \begin{bmatrix} 3 & 0 & 1 \\ 0 & 4 & 2 \\ 1 & 2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} 3 \\ 0 \\ 1 \end{bmatrix}$$

$$r(x) = \nabla f(x) = Qx - b = \begin{bmatrix} 3x_1 + x_3 - 3 \\ 4x_2 + 2x_3 \\ x_1 + 2x_2 + 3x_3 - 1 \end{bmatrix}$$

Cycle/Step A: $r^{(0)} = \begin{bmatrix} -3 \\ 0 \\ -1 \end{bmatrix}$, $x^{(0)} = 0$.

$$p^{(0)} = -g^{(0)}$$

$$\alpha_0 = -\frac{r^{(0)T} p^{(0)}}{p^{(0)T} Q p^{(0)}} = \frac{10}{36}$$

$$x^{(1)} = x^{(0)} + \alpha_0 p^{(0)} = \begin{bmatrix} 0.8333 \\ 0 \\ 0.2778 \end{bmatrix}$$

$$r^{(1)} = \nabla f(x^1) = \begin{bmatrix} -0.2222 \\ 0.5556 \\ 0.6667 \end{bmatrix}$$

$$\sigma_0 = \frac{r^{(1)T} Q p^{(0)}}{p^{(0)T} Q p^{(0)}} = 0.08025.$$

$$p^{(1)} = -r^{(1)} + \sigma_0 p^{(0)} = \begin{bmatrix} 0.4630 \\ -0.5556 \\ -0.5864 \end{bmatrix}.$$

Cycle B : $\alpha_1 = 0.2187$

$$x^{(2)} = x^{(1)} + \alpha_1 p^{(1)} = \begin{bmatrix} 0.9346 \\ -0.1215 \\ 0.1495 \end{bmatrix}$$

$$r^{(2)} = \begin{bmatrix} -0.04673 \\ -0.1869 \\ 0.1402 \end{bmatrix}$$

Compute $\sigma_1 = ?$

$p^{(2)} = ?$

Cycle C : $\alpha_2 = ?$

$x^{(3)} = ?$

$g^{(3)} = \nabla f(x^{(3)}) = 0.$ $\Rightarrow$ function is minimized.

Non-linear CG Method

- In the previous section, we employed the conjugate gradient method to minimize a quadratic function whose Hessian is positive definite. In such a case it takes n steps to converge where n is the number of variables.
- In this section, we will explore on how to extend the method for general nonlinear functions and how to develop new algorithms that do not require the Hessian if the system is too complex and computationally intensive.
- A CG algorithm can be extended by employing a Taylor-series expansion of the objective function. Truncating the expansion, at the second-order term results in a quadratic approximation of the objective function.

Disadvantageous of the CG method

- Although its a significant improvement over steepest descent, the conjugate gradient algorithm can still converge very slowly if the matrix A or Q is ill-conditiond.

- Convergence can be improved by multiplying A by M^{-1}, where M is a preconditioner and is a matrix for which systems of the form $Mz = y$ are easily solved, and whose inverse approximates that of A, so that $M^{-1}A$ is relatively well-conditioned.

- Choices of preconditioner, depend on the trade-off between the gain in the convergence rate and the increased cost per iteration that results from applying the preconditioner.

- Types of preconditioner

  1) Diagonal (also called Jacobi): M is taken to be a digonal matrix with diagonal entries equal to those of A.

  2) SSOR. Using a matrix splitting form $A = L + D + L^T$ we can take $M = (D+L)D^{-1}(D+L)^T$ or

  $$M(\omega) = \frac{1}{2-\omega}\left(\frac{1}{\omega}D + L\right)\left(\frac{1}{\omega}D\right)^{-1}\left(\frac{1}{\omega}D + L\right)^T$$ if

  a relaxation parameter ω is introduced.

  3) Incomplete Factorization; If one uses a Cholesky factorization to solve the system, then $A = LL^T$,

then one may use an approximate factorization $A \approx \hat{L}\hat{L}^T$, then use $M = \hat{L}\hat{L}^T$ as a preconditioner

4) Polynomial : M^{-1} is taken to be a polynomial in A that approximates A^{-1}.

- The CG method can only be applied to symmetric positive-definite systems. If A is indefinite or nonsymmetric, then the method will breakdown

- In recent years a number of related algorithms have been formulated for solving nonsymmetric linear systems such as :

  1) GMRES
  2) QMR
  3) CGS
  4) Bi CG
  5) Bi - CGSTAB.

Rate of Convergence

- The total cost of solving the problem depends on both the cost per iteration and the number of iterations required for convergence, and there is a trade-off between the two.

- To compare the effectiveness of iterative methods, we need to characterize their convergence rates.

  → Let us denote the error at iteration k by $e_k = |x_k - x^*|$ where x_k is the approximate solution at iteration k and x^* is the true solution

Then a method converges with rate r if

$$\lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C$$

→ If $r=1$ and $C<1$, the convergence rate is linear

If $r>1$, the convergence rate is superlinear

If $r=2$, the convergence rate is quadratic.

→ A linearly convergent sequence gains a constant number of digits of accuracy per iteration. However a superlinearly convergent sequence gains an increasing number of digits of accuracy with each iteration.

→ A linearly convergent sequence gains $\log_\beta C$ base-β digits per iteration. A supelinear sequence would gain r times that per iteration. For quadratic covergence rates, the number of digits of accuracy doubles per iteration.

- For stationary iterative methods, where $x_{k+1} = Gx_k + C$, and $G = M^{-1}N$ is the Jacobian matrix, and $\rho(G)$ is the spectral radius, then $R = -\log_{10}(\rho(G))$ serves as a useful quantitative measure of the speed of convergence.

Project 1

Consider the Helmholtz equation, which is an elliptic partial differential equation in 2D (two-dimensional),

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \lambda u = f(x,y)$$

$$u_{xx} + u_{yy} + \lambda u = f(x,y)$$

Special cases of this equation are :

1) Poisson Equation ($\lambda = 0$)

$$u_{xx} + u_{yy} = f(x,y)$$

2) Laplace Equation ($\lambda = 0$ and $f = 0$)

$$u_{xx} + u_{yy} = 0$$

The possible boundary conditions for the Helmholtz equation are :

1) Dirichlet : u is specified.
2) Neumann : The derivatives u_x or u_y are specified :
3) Mixed : A combination of Dirichlet or Neumann.

Consider the Laplace equation on a unit square,

$$u_{xx} + u_{yy} = 0,$$

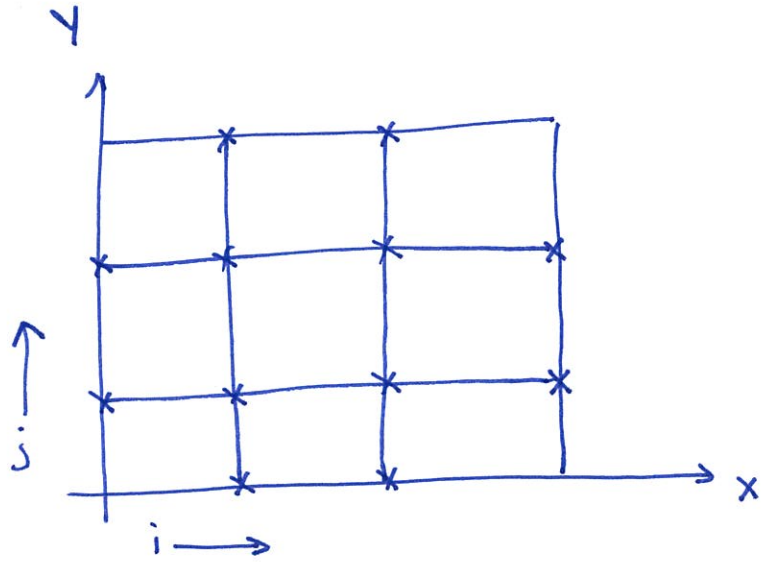
with boundary conditions $u(x,0) = 0$, $u(x,1) = 1$, $u(0,y) = 0$, $u(1,y)$

→ Using a finite difference approach, discretize u_{xx} and u_{yy} as

$$u_{xx} = \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2}$$

and $u_{yy} = \dfrac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h^2}$

On a 4×4 grid,



$$h = \frac{1}{(n+1)} = \frac{1}{2+1} = \frac{1}{3}.$$

where $n = 2$.

$$(1,1): \quad \frac{u_{2,1} - 2u_{11} + u_{0,1}}{(\frac{1}{3})^2} + \frac{u_{1,2} - 2u_{11} + u_{10}}{(\frac{1}{3})^2} = 0$$

$$u_{21} - 4u_{11} + u_{01} + u_{12} + u_{10} = 0$$

$$4u_{11} - u_{01} - u_{21} - u_{10} - u_{12} = 0$$

$$(2,1): \quad 4u_{21} - u_{11} - u_{31} - u_{20} - u_{22} = 0$$

$$(1,2): \quad 4u_{12} - u_{02} - u_{22} - u_{11} - u_{13} = 0$$

$$(2,2): \quad 4u_{22} - u_{12} - u_{32} - u_{21} - u_{23} = 0$$

In matrix form,

$$\begin{bmatrix} 4 & -1 & -1 & 0 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{bmatrix} = \begin{bmatrix} u_{01} + u_{10} \\ u_{31} + u_{20} \\ u_{02} + u_{13} \\ u_{32} + u_{23} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

By a direct method, this $Ax = b$ problem, can be solved to yield,

$$\begin{bmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{bmatrix} = \begin{bmatrix} 0.125 \\ 0.125 \\ 0.375 \\ 0.375 \end{bmatrix}$$