A COMPARATIVE STUDY OF NON LINEAR CONJUGATE GRADIENT METHODS

Subrat Pathak

Thesis Prepared for the Degree of

MASTER OF ARTS

UNIVERSITY OF NORTH TEXAS

August 2013

APPROVED:

Jianguo Liu, Major Professor
Joseph Iaia, Committee Member
Kai-Sheng Song, Committee Member
Su Gao, Chair of the Department of Mathematics
Mark Wardell, Dean of the Graduate School

Pathak, Subrat. *A comparative study of non linear conjugate gradient methods.* Master of Arts (Mathematics), August 2013, 34 pp., 11 numbered references.

We study the development of nonlinear conjugate gradient methods, Fletcher Reeves (FR) and Polak Ribiere (PR). FR extends the linear conjugate gradient method to nonlinear functions by incorporating two changes, for the step length α_k a line search is performed and replacing the residual, r_k (r_k=b-Ax_k) by the gradient of the nonlinear objective function. The PR method is equivalent to FR method for exact line searches and when the underlying quadratic function is strongly convex. The PR method is basically a variant of FR and primarily differs from it in the choice of the parameter $\beta_{k.}$ On applying the nonlinear Rosenbrock function to the MATLAB code for the FR and the PR algorithms we observe that the performance of PR method (k=29) is far better than the FR method (k=42). But, we observe that when the MATLAB codes are applied to general nonlinear functions, specifically functions whose minimum is a large negative number not close to zero and the iterates too are large values far off from zero the PR algorithm does not perform well. This problem with the PR method persists even if we run the PR algorithm for more iterations or with an initial guess closer to the actual minimum. To improve the PR algorithm we suggest finding a better weighing parameter β_k, using better line search method and/or using specific line search for certain functions and identifying specific restart criteria based on the function to be optimized.

A COMPARATIVE STUDY OF NON LINEAR CONJUGATE GRADIENT METHODS

Introduction

Optimization originated from the study of calculus of variations, a study which started with the famous Brachistochrone problem concerning the line of steepest descent. In calculus of variations we study optimization of mappings of functions to real numbers. Optimization is an iterative process which is initiated by an initial guess and followed by improving the solution in subsequent steps and finally terminating the algorithm by some stopping criteria such as tolerance or bound on the number of steps. Optimization essentially is the process of maximizing or minimizing a given objective function.

If we optimize the function f(x) subject to certain conditions or constraints then it is called constrained optimization. In unconstrained optimization, an objective function f(x) of real variables is maximized or minimized without restriction on the underlying variables.

There are two basic methods to update the current iterate x_k, the line search method and the trust region method.

In the line search methods we follow a search direction p_k and compute an associated step length α_k. The updated iterate is given by $x_{k+1} = x_k + \alpha_k p_k$

Before we can use the optimization algorithms, we need to bracket the point within a given interval at which the function needs to be optimized. This bracketing phase is used to find the interval which contains optimum step lengths. This is followed by interpolation to find an appropriate step length within the particular interval.

The bracketing process consists of starting with an initial guess, x_0, and descending downhill and computing f(x) at the iterates x_1, x_2, x_3, x_4, …… respectively until we reach some iterate x_n, for which the value of the objective function f(x) increases for the first time.

The minimum point is then bracketed in the interval, (x_{n-2}, x_n).We subsequently generate a telescoping sequence of intervals to a point within a given error tolerance denoted by ϵ to find a minimizer.

Background

For an $n \times n$ matrix, A and a vector b of dimension n, the sequence $\{A^0b, A^1b, A^2b, A^3b, A^4b, \ldots\ldots, A^{m-1}b\}$ is called a Krylov sequence. A Krylov subspace of order m, generated by matrix A and vector b, is a linear subspace spanned by the set $\{A^0b, A^1b, A^2b, A^3b, A^4b, \ldots\ldots, A^{m-1}b\}$.

The Krylov subspace methods are methods for solving large systems of linear equations or for finding the eigenvalues of sparse matrix. These methods involve a repeated pre multiplication of b by a matrix, A.

Definition - a sparse matrix is primarily a matrix composed of zeros; these matrices usually show up in the solution of partial differential equations.

Definition - a matrix A is symmetric if $A=A^T$.

Definition - a symmetric matrix A is said to be positive definite if the quadratic form $x^T A x > 0$.

Some of the Krylov subspace methods are: Arnoldi, Lanczos, GMRES (generalized minimum residuals) and conjugate gradients.

The Krylov subspace methods provide intuition to solve a large system of linear equations because for a non-singular system, Ax=b, suppose

$m(x) = x^k - \sum_0^{k-1} \alpha_j x^j$

is the minimum polynomial of b relative to A [6].

$\Leftrightarrow (A^k - \sum_0^{k-1} \alpha_j A^j)\, b=0$

$\Leftrightarrow A^k b - \sum_0^{k-1} \alpha_j A^j b=0 \Leftrightarrow A\,(A^{k-1}b - \alpha_{k-1} A^{k-2}b - \ldots\ldots\ldots - \alpha_1 b) + \alpha_0 b = 0.$

$\Leftrightarrow A\,[(A^{k-1}b - \alpha_{k-1} A^{k-2}b - \ldots\ldots\ldots - \alpha_1 b)/\alpha_0] = b,\ \alpha_0 \neq 0.$

This implies that the solution to the linear system exists within the Krylov subspace itself.

The conjugate gradient method is a Krylov method to solve symmetric positive definite system of matrices, i.e., for, Ax=b, where A is an n×n matrix, the minimum polynomial of b relative to A is $x^k - \sum_0^{k-1} \alpha_j x^j$, so, the solution lies within the Krylov space. The quadratic function, $f(x) = (1/2)\, x^T A x - b^T x$, has the gradient $\nabla f(x) = Ax - b$ and consequently we observe that finding the minimizer of the function f is equivalent to solving the linear system Ax=b.

Theorem [2] - $A \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix and let $b \in \mathbb{R}^{n\times 1}$. If $f(x) = (1/2)x^T A x - x^T b$, then the minimizer x of f(x), is the solution of A z=b.

Proof: $f(x) = (1/2)\, x^T A x - x^T A z = (1/2)x^T A x - x^T A z + (1/2)z^T A z - (1/2)z^T A z$

$= (1/2)\,(x-z)^T A(x-z) - (1/2)\, z^T A z$; noting that $x^T A z = z^T A x$; since, $-(1/2)z^T A z$, is a constant, f(x) will be minimized if x=z.

Definition - a matrix is positive definite if all its principal minors are positive.

Definition - the unique annihilating polynomial for $A \in \mathbb{C}^{n\times n}$ of minimal degree is called the minimum polynomial.

Definition - a symmetric matrix A is positive definite or positive semi definite if and only if all its eigenvalues are positive or non-negative.

The conjugate gradient method can be derived from Lanczos method since both methods use repeated multiplication by the underlying matrix to generate the Krylov subspace method.

The aim is to minimize the objective function $f(x) = (1/2)\, x^T A x - b^T x$, in n variables that is, $x \in \mathbb{R}^n$, the partial differential of the above equation with respect to, x_i, is $\frac{\partial f}{\partial xi} = -b(i) + \sum_j A(ij)x(j)$, with the equivalent vector form, $\nabla f = Ax - b$; where ∇f represents the gradient of the function f. Now, as the vector x lies in the Krylov subspace, it may turn out to be useful to optimize x, over the Krylov subspace.

The conjugate gradient method is an improvement over the steepest descent method but does not perform as well as the Newton's methods. The conjugate gradient method has the following advantages:

- It solves the quadratic function in n variables in n steps.
- It does not require the evaluation and storage of the Hessian matrix.
- It does not require the evaluation of matrix inverse.

Definition - If A is a real, positive definite, symmetric $n \times n$ matrix, then, $\{p_0, p_1, p_2, \ldots\}$ is a mutually conjugate set of vectors, sometimes called A-conjugate with respect to a symmetric positive definite matrix, A, if $p_i^T A p_j = 0$; $i \neq j$. In addition if the directions $p_0, p_1, p_2, \ldots, p_m \in \mathbb{R}^n$, $m \leq n-1$ are non-zero and A-conjugate then the set $\{p_0, p_1, p_2, \ldots, p_m\}$ is linearly Independent.

Reason - for α_k scalar let $\sum \alpha_k p_k = 0 \Leftrightarrow p_j^T A(\sum \alpha_k p_k) = 0$ (pre multiplying)

$$\Leftrightarrow \quad \alpha_k p_k^T A p_k = 0 \text{ (by conjugacy property)}$$

$$\Leftrightarrow \quad \alpha_k = 0 .$$

Hence the set $\{p_0, p_1, p_2, \ldots, p_m\}$ is linearly independent.

To find the A-conjugate vectors we may make use of the Gram Schmidt orthogonalization process from linear algebra to transform the basis of $\mathbb{R}^n$ into an orthonormal basis for $\mathbb{R}^n$.

The conjugate gradient method is a technique using the gradient of the objective function to find the unconstrained minimizer, that is, the gradient of the objective function is used to determine the search direction. In the linear conjugate gradient algorithm the search direction at each iteration is a linear combination of the previous search directions and the current gradient with the added condition that the search directions are mutually A-conjugate.

It is noteworthy that conjugate gradient algorithm is a conjugate direction method that minimizes a positive definite quadratic function in n variables in at most n steps because at most we could have n linearly independent conjugate directions which could form an orthogonal basis for $\mathbb{R}^n$.

We may use either of the following techniques for finding the descent direction to minimize the above quadratic problem, the steepest descent method or the Newton's method.

Steepest Descent method - this is a line search method where the algorithm chooses the descent direction, p_k (where $p_k = -\nabla f_k$) at the current iterate, x_k, and the appropriate step length is an approximation to the solution of the following one-dimensional minimization problem;

$\min f(x_k + \alpha p_k,)$

To ensure sufficient decrease in the function value without taking unreasonable short steps, the step length α_k can be chosen using the Wolfe conditions, the Goldstein condition or the Armijo conditions.

The steepest descent method is advantageous as it does not involve evaluation of the second derivative. But it is quite possible that the convergence in the steepest descent may not be quick if the ratio of the eigenvalues, $\lambda_{(max)}/ \lambda_{(min)}$, also known as condition number, is disproportionately large and the resulting surface may be very uneven. As a consequence, it may turn out that the direction of the negative gradient r_j may not necessarily be a descent direction.

Newton direction - we could also obtain the quadratic approximation by using the truncated Taylor series.

$$f(x_k+p) \approx f(x_k) + \{\nabla f(x_k)\}^T p +(1/2)\, p^T \nabla^2 f(x_k)p$$

by finding a vector p which minimizes a quadratic model function, $m(x_k)$, where $m(x_k)$ is an approximation to the actual function near the current iterate x_k. The basic idea is, given an initial guess, we construct a quadratic function which closely approximates the objective function and the first and second derivatives at that point. We then use the minimizer of this new function we constructed as the initial point for the next iteration. We can employ the Newton direction method only if $\nabla^2 f(x_k)$ is positive definite since the inverse of the Hessian matrix might not even exist. Though the convergence rate of the method is usually quadratic the disadvantage of Newton's method is the need to evaluate and store $\nabla^2 f (x_k)$.

This issue could be sidestepped by using the specific search directions, $\{p_0, p_1, p_{2\ldots}\}$, with the property such that $p_i^T A\ p_j=0$; $i \neq j$, in place of the residual, r_j. These specific search directions are conjugates or A-conjugates as seen before.

The basic idea is to initiate the process starting from the point x_0 with the initial descent direction being the steepest descent direction, $p_0 = - \nabla f(x_0)$ with $x_1= x_0 + \alpha_0 p_0$, where $\alpha_0 = - \{ \nabla f(x_0)\}^T p_0/(p_0)^T A p_0$; we then update the direction at the next step to $p_1= -\nabla f(x_1)_+ \beta_0 p_0$, where β_0 is such that it forces $p_1^T A p_0=0$. The next updated iterate is $x_2= x_{1+} \alpha_1 p_1$. We continue on with this process.

The linear conjugate gradient method is an algorithm to find the numerical solution for a symmetric, positive definite system of linear equations. This technique is especially useful in solving large linear system of equations.

The linear conjugate gradient method was proposed by Magnus Hestenes and Eduard Stiefel in 1952 [5].

The linear conjugate gradient method is an alternative to the Newton's method in the sense that it is an improvement over the Newton's method since it does not require the second derivative to be calculated and also in contrast to the secant updating methods the conjugate gradient method does not require the Hessian to be stored in memory.

The linear conjugate gradient method uses the gradient but unlike the steepest descent it updates the gradient at each step by removing the components from the previous search directions. The sequence of search directions is thus obtained with the terms being called conjugates. These search directions preserve the information about the Hessian matrix as well.

The linear conjugate gradient method being an iterative method to solve linear system with large and sparse (matrix primarily composed of zeros) positive definite matrices (i.e Aϵ $\mathbb{R}^{n\times n}$ and $x^T A\, x>0$) is a viable alternative to Gaussian elimination and is perfect for large problems.

For the quadratic function $f(x) = (1/2)\, x^T A\, x - b^T x$, where A is a symmetric positive definite matrix, $\nabla f(x) = Ax-b$, then the minimizer of function f is also the solution to $Ax=b$; which suggests that the methods such as the steepest descent, Newton, quasi Newton or the secant updating methods could be applied to get the solution of the corresponding system of linear equations.

An outstanding feature of quadratic optimization is the residual vector is the negative gradient, i.e. $-\nabla f(x) = b-Ax=r$. We note that since in particular the minimum over α_k occurs when the new residual is orthogonal to the search direction, a suitable value for α_k could be ascertained by analysis alone specifically by having $(d/d\alpha)\, f(x_{k+1})=0$, hence we do not need to perform a line search because the new residual depends on the old residual and the search direction and we can then solve for α.

We utilize the above outlined features to obtain the linear conjugate gradient method for solving a linear system which is both symmetric and positive definite.

Definition - a subset of $S\subseteq \mathbb{R}^n$ is convex if it contains the line segment between any two points x, yϵ S, $\{\alpha x+ (1-\alpha)\, y: 0\leq\alpha\leq 1\} \subseteq S$.

Definition - a function $f: S\subseteq \mathbb{R}^n \rightarrow \mathbb{R}$ is convex on a convex set S if for any two points x and y in S, $f(\alpha x +(1-\alpha)y)\leq \alpha\, f(x)+(1-\alpha)\, f(y)$

Definition - a function f is strictly convex if for x ≠ y,

f(αx +(1-α)y)<α f(x)+(1-α) f(y) , 0≤α≤1.

The linear conjugate gradient method is quite efficient because only one matrix vector multiplication operation is performed at each iteration besides the evaluation of Euclidean dot products thereby requiring little memory space.

Since the linear conjugate gradient method generates Krylov subspace by multiplying by matrix A over and over. The linear conjugate gradient method may not have desirable convergence if the matrix is ill conditioned. The convergence may also be affected by the distribution of the eigenvalues of the matrix of coefficients.

Definition - The rate of convergence of a descent method is measured by the limiting value of the ratio {ln f (x_k)} /{ln f (x_{k+1})}; as k approaches infinity. This limiting value is known as the order of convergence of an algorithm. Specifically, quadratic convergence means that the order of convergence is 2.

We also compare the convergence rates of iterative methods by the formula

$\lim k \to \infty \frac{\|x(k+1)-x*\|}{\|x(k)-x*\|^r}$ = C (where C is some finite positive number)

if r=1 and C<1 then the convergence rate is linear.

if r>1, the convergence rate is superliner.

One criterion to measure the convergence is to consider a descent method good if it could find the minimum of a symmetric positive definite quadratic function in a finite number of steps.

The usual stopping criterion is when the relative change $\frac{\|x(k+1)-x(k)\|}{\|x(k)\|}$ does not vary sufficiently, which implies that the approximate solution is not changing sufficiently on performing more iteration. This then signals the termination of the iterative process.

Convergence

The Krylov method: conjugate gradient, gives the solution to Ax=b, where A is n × n matrix and b is an n vector, in at most n steps. But, in actual implementation, it might turn out that, n, is a very large number. To avoid facing the problem of encountering a large n, we may use a preconditioned matrix, $M^{-1}Ax= M^{-1}b$. Here we note that preconditioning means changing the variables to get a new equation whose coefficient matrix has better eigenvalue distribution.

This is helpful in arriving at a relatively close approximation to the minimizer in just a few iterations. Although we note that the convergence is also dependent on the distribution of eigenvalues in the system.

Definition - a matrix P is called a preconditioner of another matrix A, if the condition number of the matrix $P^{-1}A$ is smaller than the condition number of matrix A.

We could improve the convergence of the conjugate gradient method by preconditioning the linear system. A preconditioner matrix M of a matrix A is such that the condition number of M^{-1} A is less than the condition number of matrix A.

Linear stationary Iterative methods split the matrix A, A=M-N. The iteration function, $x_{(k)} =H x_{(k-1)}+d$ where $H=M^{-1} N$, $d=M^{-1} b$ and the Jacobian matrix(i.e. H) are effective methods to find an easily invertible matrix M and to replace the system Ax=b by $M^{-1}Ax= M^{-1}b$ [4]. We could obtain desirable convergence by restricting the spectral radius, $\rho (M^{-1} N) < 1$. The idea is to

precondition the matrix, A, by pre multiplying it by the inverse of a matrix P for some system Px=y which can be easily solved and where P^{-1} approximates A^{-1} where the matrix $P^{-1}A$, has a smaller condition number relative to A.

Various descent methods differ from each other in their respective convergence rates. The convergence of non-stationary conjugate methods is more tricky and complicated.

We require that a preconditioned system has better spectral properties. An ideal preconditioning matrix M must fulfill the following properties:

- It must be a good approximation to the original matrix under consideration and must not be expensive to construct from the point of number of operations involved.
- The preconditioned system must be easier to solve compared to the original system.

For a linear system Ax=b, if a preconditioner M is used to solve the preconditioned system, $M^{-1}Ax= M^{-1}b$, then M is called a left preconditioner.

In this case, using the Krylov subspace method, we would construct an orthonormal basis for the Krylov subspace. $K (M^{-1}A, r_0) = \text{span} \{r_0, M^{-1}A r_0, \ldots , (M^{-1}A)^{n-1} r_0\}$; with $r_0= M^{-1}(b-A x_0)$. M is called a right preconditioner, if it solves: $AM^{-1}y=b$ where y= M x [2].

Definition - If an n×n matrix A the condition number with respect to the matrix norm II .II is ƙ (A) =IIAII $IIA^{-1}II$

Definition - A matrix is ill-conditioned if it has a large condition number and the matrix is singular if it is infinite.

Definition - a matrix is symmetric iff $v^TAW=W^TAv$.

We note that though the introduction of the preconditioner increases the convergence rate but it also increases the number of evaluations per iteration.

Some of the most common types of preconditioners are listed [4]:

- Jacobi - M is taken to be a diagonal matrix with entries equal to the corresponding entries in A.
- Block Jacobi – the indices 1, 2, 3,…., n are partitioned into mutually disjoint subsets, with, $m_{ij}= a_{ij}$ if i and j belong to the same subset otherwise, m_{ij},=0.This can be achieved by partitioning along lines or planes in a grid.
- Gauss-Seidel method - this method is an improvement over Jacobi in the sense that it incorporates the new evaluations immediately in the computation process besides requiring less storage. Here we split A= (D-L)-U, where –L and –U contain the entries above and below the main diagonal of A and D is diagonal.
- Successive Over Relaxation(SOR) - this method is an improvement over Gauss-Seidel. It uses a real number $\omega \neq 0$ as relaxation or correction parameter and we write $A= [\omega^{-1} D-L]-[(\omega^{-1} -1) D+U]$.
- Symmetric Successive Over Relaxation (SSOR) - we split matrix A, as $A= L+D+L^T$, where D is diagonal matrix and L is lower triangular matrix and then express, $M= (D+L) D^{-1}(D+L)^T$.
- Polynomial – we try to find a polynomial matrix of low degree with nicer properties and we approximate A^{-1} by taking M^{-1}as a polynomial in A.

- Approximate Inverse – here we use the optimization algorithms to minimize the residual, $I-A M^{-1}$, in some norm with restriction to have a pre-determined pattern for the non-zero entries.
- Incomplete Cholesky factorization – here we compute the approximate Cholesky factorization, $A \approx LL^T$, with the non-zero entries of L restricted to positions as those in the lower triangle in A. This method is suitable for the conjugate gradients.

The Linear Conjugate Gradient Method

This is an iterative method for solving a linear system of equations Ax=b; where A is an $n \times n$ symmetric positive definite matrix. The idea of linear conjugate gradient method is to obtain the new search direction which is orthogonal to all the previous search directions. This is achieved by restricting the search directions to a set of conjugates which are linearly independent which in turn guarantees that after n steps we would have the exact solution since n linearly independent vectors span $\mathbb{R}^n$.

We recall that solving the above system is equivalent to minimizing the quadratic function, $f(x) = (1/2)\, x^T Ax - x^T b$. The equivalence between the linear system and the convex minimization problem allows us to visualize the linear conjugate gradient method both as an algorithm for solving linear systems and as a technique for minimizing convex quadratic functions.

An outstanding feature of the linear conjugate gradient method is the ability to generate a set of vectors with the conjugacy property. The importance of conjugacy lies in the

fact that function f could be minimized in n steps by successively minimizing it along the individual directions in a conjugate set. Also, the gradient of function f given by $f(x) = (1/2)\, x^T Ax - b^T x$, equals the residual of the linear system Ax=b, that is, $\nabla f(x) = Ax - b = r$. In particular the residual vectors, $r_k = A x_k - b$ are orthogonal, that is, $r_k^T r_j = 0 \forall k > j$.

Theorem [9] - Suppose the k th iterate generated by the conjugate gradient method is not the solution x*.Then $r_k^T r_i = 0$ for $0 \leqq i \leqq k-1$.

span $\{ r_0, r_1, r_{2,\ldots\ldots,\ldots\ldots}, r_{k-1}\}$ = span $\{A^0 b, A^1 b, A^2 b, A^3 b, \ldots., A^{k-1} b\}$

span $\{ p_0, p_1, p_{2,\ldots\ldots,\ldots\ldots}, p_{k-1}\}$= span $\{A^0 b, A^1 b, A^2 b, A^3 b, \ldots., A^{k-1} b\}$

$p_k^T A\, p_i = 0$; $0 \leqq I \leqq k-1$. Therefore the sequence $\{x_k\}$ converges to x* in at most n steps.

Theorem [2] - The conjugate gradient algorithm converges in n steps.

Proof - We know that r_n, is orthogonal to $r_0, r_1, r_{2\ldots} r_{n-1}$ and from the above Krylov space properties, evidently $r_0, r_1, r_{2,\ldots\ldots,\ldots\ldots}, r_{n-1}$ being linearly independent ; form a basis for $\mathbb{R}^n$. Also since r_n, is orthogonal to the preceding residuals, $r_0, r_1, r_{2,\ldots\ldots,\ldots\ldots}, r_{n-1}$ we have that $r_n = 0$.

The linear conjugate gradient method can be modified to solve nonlinear optimization problems.

Basic Properties of Linear Conjugate Gradient Method

The conjugate gradient method is a conjugate direction method with additional property that in generating the set of conjugate vectors, it can compute a new vector p_k by using only the previous vector p_{k-1}. Thus it requires little storage. Also, the method does not require the calculation of second partial derivatives.

In the conjugate gradient method the direction p_k ,at each iteration is chosen to be a linear combination of the negative residual $-r_k$,which is the steepest descent direction for the function f and the preceding direction p_{k-1}, given by, $p_k = - r_k + \beta_k p_{k-1}$ with the requirement that the A-conjugacy property of the vectors, p_k and p_{k-1} help in determining the scalar, β_k .

The first search direction p_0 is chosen to be the steepest descent direction at the initial point x_0.While executing the conjugate direction method one dimensional minimizations is performed successively along each of the search directions.

Since $\alpha_k = \min f(x_k + \alpha p_k)$ and we minimize over α, we note that the matrix A appears in computations only while updating α_k and β_k . Consequently, we could replace α_k by using the line search methods. In a similar approach, for each β_k, where $\beta_k = (\nabla f_{k+1})^T A \nabla f_{k+1} / (\nabla f_{k+1})^T A \nabla f_k$; we rearrange the formula so that the matrix, A does not show up in the formula. Then at each iteration, the computation will depend only on the objective function and the gradient of the function.

We look into ways to modify the conjugate gradient algorithm such that we do not require the Hessian to be evaluated at each iteration but at the same time the gradient and the value of the objective function is available.

MATLAB Code for Linear Conjugate Gradient Algorithm

```
function x=cg_2(A, b , x, tol)
%x = [2.00; 1.00];
%b = [1.00; 2.00];
%A = [4.00, 1.00; 1.00, 3.00];
```

```
r = (A * x(:,1)) - b;
p = -1. * r(:,1);
k = 1;
while (r(:,k) ~= 0) & (k < tol) %tol is the max number of k values that the iteration can go
upto
   alpha = (r(:,k)' * r(:,k)) / (p(:,k)' * A * p(:,k));%(:,k)
%is the k th col
   x = [x , (x(:,k) + alpha * p(:,k))];%this calculates the k+1 th iteration and appends that
value as the k+1 th column
   r = [r , (r(:,k) + alpha * A * p(:,k) )];
   beta = (r(:,k+1)' * r(:,k+1)) / (r(:,k)' *
r(:,k));%here beta is being updated at each step
p = [p , ( (-1.*r(:,k+1)) + (beta * p(:,k)) )];
   k = k+1;
end
display(r); display(x);
end
```

Nonlinear Conjugate Gradient Method

The linear conjugate gradient method can be modified to solve nonlinear optimization problem also. We recall that the linear conjugate gradient method can be viewed as the minimization algorithm for convex quadratic function f, given by, min $f(x) = (1/2)x^TAx - b^Tx$.

Intuitively we could apply the conjugate gradient algorithm to nonlinear functions as well by visualizing the quadratic function,

$f(x) = (1/2)x^TAx - b^Tx$ as a Taylor series approximation of the objective function as the end behavior of the nonlinear functions near the solution is similar to that of the quadratic functions.

Fletcher Reeves Method

Fletcher Reeves (FR) extends the linear CG method to nonlinear functions by incorporating two changes:

- For the step length α_k, (which minimizes f along the search direction p_k), we perform a line search that identifies the approximate minimum of the nonlinear function f along the search direction p_k.

Note - to find the appropriate step length effecting sufficient decrease we could choose from various method such as the Armijo, the Goldstein or the Wolfe's conditions.

- The residual r_k (r_k=b-Ax_k), which is the gradient of function f has to be replaced by the gradient of the nonlinear objective function.

Note - If f is a strongly convex quadratic function and α_k is the exact minimizer of the function f, then the FR algorithm becomes specifically the linear conjugate gradient algorithm.

Definition [8] - A continuously differentiable function f is called strongly convex on $\mathbb{R}^n$ if there exists some constant $\mu > 0$ such that $\forall x, y \in \mathbb{R}^n$,

$f(y) - f(x) \geqq \langle f'(x), y-x \rangle + (1/2)\,\mu \|y-x\|^2$

MATLAB Code for Fletcher Reeves Algorithm

```
function [fmin,xmin,ymin,k,finalX,finalY,finalZ] = FR_20Jan2013( f,x0,y0 )%RHS input
%finalX and finalY return variables were added to obtain the x and y
%cordinates. these two are vectors
%FR [x,y,fmin,n] = FR( f,x0,y0 )
%as example FR_11('z^2+z^3',5)
%x0: initial guess
tol=10^-4; % our tolerance
x(1)=x0;y(1)=y0;%matlab starts vectors at index 1
%f is a function of x and y, k:number of steps limited to 200
gradf=[diff(f,sym('x'));diff(f,sym('y'))]; %the gradient of f
%p(1)=-subs(subs(f,'x',x0),'y',y0) %by this we mean p0=-gradf(x0)
p(:,1)=-subs(subs(gradf,'x',x0),'y',y0); %by this we mean p0=-gradf(x0)
%:,1 inside p means the first col in all the rows in the matrix
k=1;%matlab starts counting at 1
finalX = x(1) ; %initialize the vector
finalY = y(1) ;
finalZ = subs(subs(f,'x',x(1)),'y',y(1));
while and(norm(subs(subs(gradf,'x',x(k)),'y',y(k)))>tol,k<500)
   rho=0.5;c=0.1;
   alp=armijo(f,rho,c,x(k),y(k),gradf,p(:,k)); % alp is updated using the other function
   x(k+1)=x(k)+alp*p(1,k);
```

```
        %updating x(k);p(1,k)means row 1 &col k
           y(k+1)=y(k)+alp*p(2,k);
        %updating y(k);p(2,k)means row 2 &col k
        bet(k+1)=subs(subs(gradf,'x',x(k+1)),'y',y(k+1))'*subs(subs(gradf,'x',x(k+1)),'y',y(k+1))/((s
ubs(subs(gradf,'x',x(k)),'y',y(k))'*subs(subs(gradf,'x',x(k)),'y',y(k))));
           p(:,k+1)=-subs(subs(gradf,'x',x(k+1)),'y',y(k+1))+bet(k+1)*p(:,k);
        %p(:,k+1)takes existing matrix and adds a col
           finalX = [finalX; x(k+1)];
        %each calculated value is appended into the
           finalY = [finalY; y(k+1)];
           finalZ= [finalZ; subs(subs(f,'x',x(k+1)),'y',y(k+1))];
           k=k+1
        end
           k=k
           fmin=subs(subs(f,'x',x(k)),'y',y(k))
           xmin=x(k)
           ymin=y(k)
           finalX = [finalX; xmin]; %each calculated value is append into the
           finalY = [finalY; ymin];
           finalZ = [finalZ; fmin];%vertical descent down along the Z direction
        %axis([x(k)-10 x(k)+10 y(k)-20 y(k)+20]);trying to set the axis to an area
        %near the final solution
```

```
plot3(finalX,finalY,finalZ);%plot of the iterarates x,y and z i.e the path to the solution
hold on
   %plot(finalX,finalY);this was for the 2D projection that we are not
   %using now
  [X,Y]= meshgrid(x(k)-10:1:x(k)+10,y(k)-20:1:y(k)+20);%
   % Z=subs(subs(f,'x','X.'),'y','Y.');
   IJ=size(X);%getting the matrix dimension for the mesh grid
   Z=zeros(size(X));%initializing Z
   for i=1:IJ(1)
      for j=1:IJ(2)
         Z(i,j)=subs(subs(f,'x',X(i,j)),'y',Y(i,j));%evaluating the function on the grid
      end
   end
   mesh(X,Y,Z)%plotting the surface
   title('Subrats Pics'),xlabel('x'),ylabel('y')
end


function [alp]=armijo(f,rho,c,xk,yk,gradf,p)%alp0 is the initial step size,
%rho is (0,1),xk is the current iterate
%c is in (0,1);
%DEBUG - put gradf_val back in and compare with the p values, to try and
```

```
%see why our alphas are so small/silly
gradf_val=subs(subs(gradf,'x',xk),'y',yk);
alp=0.5;% initial step
fkk=subs(subs(f,'x',xk+alp*p(1)),'y',yk+alp*p(2));%fkk is the potential new min
fk=subs(subs(f,'x',xk),'y',yk);
   while and(fkk>fk+c*alp*(gradf_val)'*p, alp>10^-6);
   alp=rho*alp;
   fkk=subs(subs(f,'x',xk+alp*p(1)),'y',yk+alp*p(2));%fkk is the potential new min
   end
end
```

Fletcher Reeves algorithm applied to nonlinear Rosenbrock function:

$f(x,y) = 100(y-x^2)^2+(1-x)^2 + .01$

FR ('$100*(y-x^2)^2+ (1-x)^2 + .01$',0,0)

Initial guess $x_0 = 0$, $y_0 = 0$

k = 42(number of iterations)

fmin = 0.0100

xmin = 0.9999

ymin = 0.9999

ans = 0.0100

Fletcher Reeves algorithm applied to nonlinear function [7]:

$f(x,y) = (x^2/4)+(y^2/10)-0.8\ x – y - 0.3\ x\ y-3$

FR ('$(x^2/4) + (y^2/10)-0.8*x-y-0.3*x*y-3$',0,0)

Initial guess $x_0 =0$, $y_0=0$

k = 115

fmin = -58.4000

xmin = 45.9966

ymin = 73.9945

Fletcher Reeves algorithm applied to nonlinear function [7]:

$f(x,y) = (x^2/4)+(y^2/11)-0.8\ x – y - 0.3\ x\ y-3$

FR ('$(x^2/4) + (y^2/11)-0.8*x-y-0.3*x*y-3$',0,0)

Initial guess $x_0 =0$, $y_0=0$

k = 389

fmin = -606.0000

xmin = 489.9623

ymin = 813.9374

The outstanding features of the FR method are:

- Suitability for large nonlinear problems because of the only requirements at each iteration being, evaluation of objective function and the gradient.

- No need to perform matrix operations (matrix-vector or matrix-matrix multiplication) for each step computation.
- Requirement of very little storage space.

The Polak Ribiere(PR) is another nonlinear conjugate gradient method similar/ equivalent to FR for exact line searches and when the underlying quadratic function is strongly convex. Recalling the identities from Krylov spaces; if the k^{th} iterate from the CG method is not the solution, then the successive gradients being orthogonal (i.e. $r_k^T * r_j = 0 \; \forall \; j<k$) we have $\beta^{PR}_{k+1} = \beta^{FR}_{k+1}$.

PR is basically a variant of FR and primarily differs from it in the choice of the parameter β_k.

But when we apply the FR and the PR to nonlinear functions using inexact line search, we find that the PR algorithm is more stable and efficient (this is highlighted in our examples when we input nonlinear Rosenbrock function into the MATLAB code for FR and PR algorithms we find that PR performs the task in fewer steps and with better approximation).

We could also have other choices for β_{k+1}, as well but in particular for quadratic functions with the exact line search, using the Hestenes-Stiefel (HS) formula,

$$\beta^{HS}_{k+1} = (\nabla f_{k+1})^T[\nabla f_{k+1} - \nabla f_k]/(\nabla f_{k+1} - \nabla f_k)^T p_k$$

gives an algorithm that is quite similar to the PR.

If the line search method is not accurate it is better to use the Hestenes- Stiefel to generate the β_k.

Extension to Non-Quadratic Functions: Restart

A modification that the nonlinear conjugate gradient method makes to linear conjugate gradient method is that it restarts the iterations after every n steps. The restart is needed because the Hessian keeps changing at each iteration and we may not obtain convergence after n steps. The restart is executed by choosing the descent direction as the steepest descent step, which is achieved by setting $\beta_k=0$. This re initialization deletes the unnecessary information from memory thereby increasing the efficiency of the algorithm. This is also the basic difference between the PR and the FR algorithms.

Also, if the algorithm is converging for a function, f, which is not quadratic anywhere but only convex and quadratic in the neighborhood of the solution, then, it is certain that the iterates will enter the neighborhood of the solution at some point. At this point the algorithm will be restarted and would behave as the linear conjugate gradient method [9].

Also, in case, if the function, f, is not quadratic in neighborhood of the solution, the Taylor's theorem:

Theorem [9] - Suppose that f: $\mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable and that $p \epsilon \mathbb{R}^n$. Then we have,

$f(x+p) = f(x)+ \nabla f(x+tp)^T$ for some $t \epsilon (0,1)$.Moreover, if f is twice continuously differentiable, we have that ,

$\nabla f(x+tp) = \nabla f(x) + \int \nabla^2 f(x+tp)p \, dt$ (the limits of integration are t=0 to t=1 And that,

$f(x+p) = f(x)+ \nabla f(x)^T p+(1/2)\, p^T \nabla^2 f(x+tp)p$, for some $t \epsilon (0,1)$.

tells us that a smooth function f can be approximated by a quadratic function.

One of the most popular algorithm restart strategy is executed when the consecutive gradients are not orthogonal and is determined by the relation,

$(|\nabla f_k^T * \nabla f_{k-1}|/ \ \nabla f_k \ ) \geqq v$, where v is typically 0.1 [9]

If the FR algorithm generates a bad search direction and very small step length, it is quite probable that the steps that follow the updated search directions and step lengths are just as under achieving.

As we know, $\cos \theta_k = (-(\nabla f_k)^T * p_k) / \|\nabla f_k\| * \|p_k\|$

where θ_k is the angle between the search direction, p_k, and the steepest descent direction $-\nabla f_k$.

In particular if θ_k is such that $\cos \theta_k \approx 0$, for some iterate k, and the immediate next step turns out to be tiny, that is, $x_{k+1} \approx x_k$ then it is an indicator that the algorithm is stuck in a sequence of under achieving iterations. Consequently the FR algorithm will take a large number of tiny steps to get a close approximation to the solution.

In contrast the PR method is quite efficient in comparison to FR. If the search direction, p_k, satisfies the condition, $\cos \theta_k \approx 0$,and,in case the immediate next step is too small, then plugging , $\nabla f_k \approx \nabla f_{k+1}$, in

$\beta_{k+1}^{PR} = (\nabla f_{k+1})^T [\nabla f_{k+1} - \nabla f_k] / \|\nabla f_k\|^2$

$\Rightarrow \beta_{k+1} \approx 0$,then as the updated descent direction p_{k+1}, is

$p_{k+1} = -\nabla f_{k+1} + \beta_{k+1}^{PR} * p_k$, the updated search direction, $p_{k+1} = -\nabla f_{k+1}$ which is almost the steepest descent direction, $-\nabla f_{k+1}$.This is an example of the restart that the PR method undertakes on encountering a bad search direction with insignificant reduction in function value [9].

Here in the formula for β_k , i.e. $\beta_k=(\nabla f_{k+1})^T Ap_k/((\nabla p_k)^T Ap_k)$; replacing

Ap_k by $(\nabla f_{k+1} -\nabla f_k)/ \alpha_k$; where $\alpha_k=-(\nabla f_k)^T p_k /((p_k)^T Ap_k)$

Now since, $x_{k+1}= x_k+\alpha_k p_k$. Pre multiplying with A gives, $Ax_{k+1}= A x_k +A\alpha_k p_k$

$\Leftrightarrow \nabla f_{k+1}=\nabla f_k+ \alpha_k A p_k$ (as $\nabla f_{k+1} =A x_k -b$)

$\Leftrightarrow A p_k = (\nabla f_{k+1}-\nabla f_k)/ \alpha_k$; plugging back this value,

$\beta_k =(\nabla f_{k+1})^T[\nabla f_{k+1}-\nabla f_k]/(p_k)^T [\nabla f_{k+1}-\nabla f_k]$

$= (\nabla f_{k+1})^T[\nabla f_{k+1}-\nabla f_k]/(p_k)^T [\nabla f_{k+1}- \nabla f_k]$

But by conjugacy property [1]

$(p_k)^T \nabla f_{k+1}=0$; and $p_k=-\nabla f_k+\beta_{k-1*} p_{k-1}$;

$\Leftrightarrow (\nabla f_k)^T p_k=-(\nabla f_k)^T* \nabla f_k+\beta_{k-1*}(\nabla f_k)^T * p_{k-1}$;

$=-(\nabla f_k)^T* \nabla f_k$

Hence, $\beta_{k+1}= ((\nabla f_{k+1})^T*[\nabla f_{k+1}-\nabla f_k])/((\nabla f_k)^T* \nabla f_k)$;

This is the formula for Polak Ribiere [1].

Thus we see that the PR method differs from the FR conjugate gradient method in the choice of the parameter.

Also, it is noteworthy that the two algorithms are identical when the function is convex and quadratic and the exact line search method is used. In contrast when the two algorithms are applied to general nonlinear functions with inexact line search methods then the PR algorithm is more stable and efficient.

It is worth noting that it is not always the case that the PR algorithm is more efficient than the FR method. Also PR method requires storing one extra vector in comparison to FR method.

MATLAB Code for Polak Ribiere Algorithm

```
function [fmin,xmin,ymin,k,finalX,finalY,finalZ] = FR_20Jan2013Copy( f,x0,y0 )%RHS
input

%THIS IS THE POLAK RIBIERE

%finalX and finalY return variables were added to obtain the x and y

%cordinates. these two are vectors

%FR [x,y,fmin,n] = FR( f,x0,y0 )

%as example FR_11('z^2+z^3',5)

%x0: initial guess

tol=10^-4; % our tolerance

x(1)=x0;y(1)=y0;%matlab starts vectors at index 1

%f is a function of x and y, k:number of steps limited to 200

gradf=[diff(f,sym('x'));diff(f,sym('y'))]; %the gradient of f

%p(1)=-subs(subs(f,'x',x0),'y',y0) %by this we mean p0=-gradf(x0)

p(:,1)=-subs(subs(gradf,'x',x0),'y',y0); %by this we mean p0=-gradf(x0)

%:,1 inside p means the first col in all the rows in the matrix

k=1;%matlab starts counting at 1

finalX = x(1) ; %initialize the vector

finalY = y(1) ;

finalZ = subs(subs(f,'x',x(1)),'y',y(1));

while and(norm(subs(subs(gradf,'x',x(k)),'y',y(k)))>tol,k<500)

   rho=0.5;c=0.1;
```

```
    alp=armijo(f,rho,c,x(k),y(k),gradf,p(:,k)); % alp is updated using the other function
    x(k+1)=x(k)+alp*p(1,k);
%updating x(k);p(1,k)means row 1 &col k
    y(k+1)=y(k)+alp*p(2,k);
%updating y(k);p(2,k)means row 2 &col k
bet(k+1)=subs(subs(gradf,'x',x(k+1)),'y',y(k+1))'*(subs(subs(gradf,'x',x(k+1)),'y',y(k+1))-
subs(subs(gradf,'x',x(k)),'y',y(k)))/((subs(subs(gradf,'x',x(k)),'y',y(k))'*subs(subs(gradf,'x',x(k)),'y',y
(k))));
    p(:,k+1)=-subs(subs(gradf,'x',x(k+1)),'y',y(k+1))+bet(k+1)*p(:,k);
%p(:,k+1)takes existing matrix and adds a col
    finalX = [finalX; x(k+1)];
%each calculated value is appended into the
    finalY = [finalY; y(k+1)];
    finalZ= [finalZ; subs(subs(f,'x',x(k+1)),'y',y(k+1))];
    k=k+1
end
    k=k
    fmin=subs(subs(f,'x',x(k)),'y',y(k))
    xmin=x(k)
    ymin=y(k)
    finalX = [finalX; xmin]; %each calculated value is append into the
    finalY = [finalY; ymin];
```

```
    finalZ = [finalZ; fmin];%vertical descent down along the Z direction
%axis([x(k)-10 x(k)+10 y(k)-20 y(k)+20]);trying to set the axis to an area
%near the final solution
 plot3(finalX,finalY,finalZ);%plot of the iterarates x,y and z i.e the path to the solution
hold on
    %plot(finalX,finalY);this was for the 2D projection that we are not
    %using now
     [X,Y]= meshgrid(x(k)-10:1:x(k)+10,y(k)-20:1:y(k)+20);%
    % Z=subs(subs(f,'x','X.'),'y','Y.');
    IJ=size(X);%getting the matrix dimension for the mesh grid
    Z=zeros(size(X));%initializing Z
    for i=1:IJ(1)
        for j=1:IJ(2)
            Z(i,j)=subs(subs(f,'x',X(i,j)),'y',Y(i,j));
%evaluating the function on the grid
        end
    end
    mesh(X,Y,Z)%plotting the surface
    title('Subrats Pics'),xlabel('x'),ylabel('y')
end
 function [alp]=armijo(f,rho,c,xk,yk,gradf,p)%alp0 is the initial step size,
%rho is (0,1),xk is the current iterate
```

```
%c is in (0,1);
%DEBUG - put gradf_val back in and compare with the p values, to try and
%see why our alphas are so small/silly
gradf_val=subs(subs(gradf,'x',xk),'y',yk);
alp=0.5;
% initial step
fkk=subs(subs(f,'x',xk+alp*p(1)),'y',yk+alp*p(2));
%fkk is the potential new min
fk=subs(subs(f,'x',xk),'y',yk);
   while and(fkk>fk+c*alp*(gradf_val)'*p, alp>10^-6);
   alp=rho*alp;
   fkk=subs(subs(f,'x',xk+alp*p(1)),'y',yk+alp*p(2));
%fkk is the potential new min
   end
end
```

Polak Ribiere algorithm applied to Rosenbrock function:

PR ('100*(y-x^2)^2+(1-x)^2 +.01',0,0)

Initial guess x_0 =0, y_0=0

k = 29

fmin = 0.0100

xmin = 1.0001

ymin = 1.0002

ans = 0.0100

Polak Ribiere algorithm applied to [7]:

f(x,y) = $(x^2/4)+(y^2/10)-0.8\ x - y- 0.3\ x\ y-3$

PR ('$(x^2/4) + (y^2/10)-0.8*x-y-0.3*x*y-3$',0,0)

k = 500

fmin = -58.3608

xmin = 44.7819

ymin = 72.0290

ans = -58.3608

Polak Ribiere algorithm applied to [7]:

f(x,y) = $(x^2/4)+(y^2/11)-0.8\ x - y-0.3\ x\ y-3$

PR ('$(x^2/4) + (y^2/11)-0.8*x-y-0.3*x*y-3$',0,0)

k = 500

fmin = -296.3405

xmin = 139.0078

ymin = 230.5760

Applications and Research

One of the most researched areas in computational biology is to determine the genome sequence of organisms and finding new protein structures. The tools required for such an exercises are software to build protein models* and fitting maps etc. which make use of Polak Ribiere variant of BFGS (a secant updating method for solving nonlinear equations) conjugate gradient technique to minimize a multiple variable function [3].

The nonlinear conjugate gradient methods are increasingly used in imaging and image restoration because of low evaluation and storage cost [10].

MATLAB is used in computed tomography (CT scans) for three dimensional modeling of forward scattering using the conjugate gradient together with fast linear adjoint approximation. The nonlinear conjugate gradient method is used in three dimensional diffraction tomography [11].

Observation

We observe that the performance of PR method (k=29) is far better than the FR method (k=42) when the minimum of the nonlinear function is close to zero and the iterates have small magnitude as well; this is evident from the application of the MATLAB code to the Rosenbrock function.

But when the function minimum is a large negative number and the iterates are large values far off from zero the PR algorithm does not perform well. This is verified by applying the MATLAB codes to general nonlinear functions [7]. Although, it is worth noting that the FR method still works better in comparison to the PR method in this situation.

There could be more than one reason for this inaccuracy of the PR method. One reason could be that the general condition for restart is not met for particular termination conditions such as Wolfe or Armijo. This, if true highlights another drawback of the PR method namely the restart conditions have to be appropriate and suitable on the case by case basis which then would depend on the function being considered.

Another reason could be that we may need to use specific inexact line search condition(Wolfe , Strong Wolfe or another criteria) for certain specific functions to obtain better convergence. This would need more analysis of the function before we have a particular variant of PR available which gives better results.

Suggestions for Improvement of PR Nonlinear Conjugate Gradient Method

- Finding a better weighing parameter β_k.
- Devising better line search method to find the optimum step length descent direction and/or using specific line search for certain functions.
- Specific restart criteria need to be identified based on the function to be optimized.

If we incorporate the above features, it may make the PR algorithm more robust and accurate for iterates and solution bounded away from zero. This could also make it cost effective for operation count, storage and algorithm run time.

REFERENCES

[1] An Introduction to Optimization- Chong Edwin K P and Zak; Second Edition, Wiley.

[2] Numerical Linear Algebra and Applications- Datta Biswa Nath, Second Edition, SIAM 2010.

[3] Coot: Model- building tools for molecular graphics-Biological Crystallography: ISSN 0907-4449, Emslev and Cowtan.

[4] Scientific Computing: An Introductory Survey- Heath Michael T, International Edition, McGraw Hill.

[5] Journal of Research of the National Bureau of Standards Vol. 49, No. 6, December 1952 Research Paper 2379. Methods of Conjugate Gradients for Solving Linear Systems 1, Magnus R. Hestenes 2 and Eduard Stiefel

[6] Matrix Analysis and Applied Linear Algebra- Meyer Carl D, SIAM

[7] Optimization Foundations and Applications- Miller Ronald E, Wiley

[8] Introductory lectures on Convex Optimization-Nesterov and Nesterov; Springer

[9] Numerical Optimization-Nocedal and Wright, page 14, Second Edition, Springer 2000.

[10] Smoothing nonlinear Conjugate Gradient Method for Image Restoration Using Non smooth Non convex Minimization; *SIAM J. Imaging Sci.*, 3(4), 765–790.

[11] AWARD NUMBER: W81XWH-04-1-0042; Transurethral Ultrasound Diffraction Tomography; Principal Investigators: Matthias C. Schabel, Ph.D., Dilip Ghosh Roy, Ph.D. Altaf Khan ; University of Utah Salt Lake City, Utah 84112-9351.