4.5.3 Variants of Newton's Method

One important aspect of the algorithm in Figure 4.49 is the step (1), which involves solving a linear system $\nabla^2 f(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \nabla f(\mathbf{x}^{(k)})$. The system can be easy to solve if the Hessian is a 100×100 sparse matrix, but it can get hairy if it is a larger and denser matrix. Thus it can be unfair to claim that the Newton's method is faster than the gradient descent method on the grounds that it takes a fewer number of iterations to converge as compared to the gradient descent, since each iteration of the Newton's method involves inverting the hessian to solve a linear system, which can take time[26] $O(n^3)$ for dense systems. Further, the method assumes that the hessian is positive definite and therefore invertible, which might not always be so. Finally, the Newton's method might make huge-uncontrolled steps, especially when the hessian is positive semi-definite (for example, if the function is flat along the direction corresponding to a 0 or nearly 0 eigenvalue). Due to these disadvantages, most optimization packages do not use Newton's method.

There is a whole suite of methods called *Quasi-Newton methods* that use approximations of the hessian at each iteration in an attempt to either do less work per iteration or to handle singular hessian matrices. These methods fall in between gradient methods and Newton's method and were introduced in the 1960's. Work on quasi-Newton methods sprang from the belief that often, in a large linear system, most variables should not depend on most other variables (that is, the system is generally sparse).

We should however note that in some signal and image processing problems, the hessian has a nice structure, which allows one to solve the linear system $\nabla^2 f(\mathbf{x}^{(k)})\Delta\mathbf{x}^{(k)} = \nabla f(\mathbf{x}^{(k)})$ in time much less than $O(n^3)$ (often in time comparble to that required for quasi Newton methods), without having to explicitly store the entire hessian. We next discuss some optimization techniques that use specific approximations to the hessian $\nabla^2 f(\mathbf{x})$ for specific classes of problems, by reducing the time required for computing the second derivatives.

4.5.4 Gauss Newton Approximation

The Gauss Newton method decomposes the objective function (typically for a regression problem) as a composition of two functions[27] $f = l \circ \mathbf{m}$; (i) the vector valued model or regression function $\mathbf{m} : \Re^n \rightarrow \Re^p$ and (ii) the scalar-valued loss (such as the sum squared difference between predicted outputs and target outputs) function l. For example, if m_i is $y_i - r(\mathbf{t}_i, \mathbf{x})$, for parameter vector $\mathbf{x} \in \Re^n$ and input instances $(y_i, \mathbf{t}_i)$ for $i = 1, 2, \ldots, p$, the function f can be written as

$$f(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^{p}(y_i - r(\mathbf{t}_i, \mathbf{x}))^2$$

[26] $O(n^{2.7})$ to be precise.

[27] Here, n is the number of weights.

An example of the function r is the linear regression function $r(\mathbf{t}_i, \mathbf{x}) = \mathbf{x}^T\mathbf{t}_i$. Logistic regression poses an example objective function, which involves a cross-entropy loss.

Cross entropy loss for logistic regression

$$f(\mathbf{x}) = -\sum_{i=1}^{p} \left(y_i \log\left(\sigma(\mathbf{x}^T\mathbf{t}_i)\right) + (1 - y_i) \log\left(\sigma(-\mathbf{x}^T\mathbf{t}_i)\right)\right)$$

where $\sigma(k) = \frac{1}{1+e^{-k}}$ is the logistic function.

The task of the loss function is typically to make the optimization work well and this gives freedom in choosing l. Many different objective functions share a common loss function. While the sum-squared loss function is used in many regression settings, cross-entropy loss is used in many classification problems. These loss functions arise from the problem of maximizing log-likelihoods in some reasonable way.

The Hessian $\nabla^2 f(\mathbf{x})$ can be expressed using a matrix version of the chain rule, as

$J_m(x) = [\nabla m_1(x)\ \nabla m_2(x) \ldots \nabla m_q(x)]$

$$\nabla^2 f(\mathbf{x}) = \underbrace{J_{\mathbf{m}}(\mathbf{x})^T \nabla^2 l(\mathbf{m}) J_{\mathbf{m}}(\mathbf{x})}_{G_f(\mathbf{x})} + \sum_{i=1}^{p} \nabla^2 m_i(\mathbf{x})(\nabla l(\mathbf{m}))_i$$

By virtue of chain rule expected to → 0 near optimal pt

where $J_{\mathbf{m}}$ is the jacobian[28] of the vector valued function $\mathbf{m}$. It can be shown that if $\nabla^2 l(\mathbf{m}) \succeq 0$, then $G_f(\mathbf{x}) \succeq 0$. The term $G_f(\mathbf{x})$ is called the Gauss-Newton approximation of the Hessian $\nabla^2 f(\mathbf{x})$. In many situtations, $G_f(\mathbf{x})$ is the dominant part of $\nabla^2 f(\mathbf{x})$ and the approximation is therefore reasonable. For example, at the point of minimum (which will be the critical point for a convex function), $\nabla^2 f(\mathbf{x}) = G_f(\mathbf{x})$. Using the Gauss-Newton approximation to the hessian $\nabla^2 f(\mathbf{x})$, the Newton update rule can be expressed as

Certainly a reasonable approximation in the quadratically convergent phase

$$\Delta\mathbf{x} = -(G_f(\mathbf{x}))^{-1}\nabla f(\mathbf{x}) = -(G_f(\mathbf{x}))^{-1} J_{\mathbf{m}}^T(\mathbf{x}) \nabla l(\mathbf{m})$$

where we use the fact that $(\nabla f(\mathbf{x}))_i = \sum_{k=1}^{p} \frac{\partial l}{\partial m_k} \frac{\partial m_k}{\partial x_i}$, since the gradient of a composite function is a product of the jacobians.

For the cross entropy classification loss or the sum-squared regression loss l, the hessian is known to be positive semi-definite. For example, if the loss function is the sum of squared loss, the objective function is $f = \frac{1}{2}\sum_{i=1}^{p} m_i(\mathbf{x})^2$ and $\nabla^2 l(\mathbf{m}) = I$. The Newton update rule can be expressed as

$$\Delta\mathbf{x} = -(J_{\mathbf{m}}(\mathbf{x})^T J_{\mathbf{m}}(\mathbf{x}))^{-1} J_{\mathbf{m}}(\mathbf{x})^T \mathbf{m}(\mathbf{x})$$

Recall that $(J_{\mathbf{m}}(\mathbf{x})^T J_{\mathbf{m}}(\mathbf{x}))^{-1} J_{\mathbf{m}}(\mathbf{x})^T$ is the Moore-Penrose pseudoinverse $J_{\mathbf{m}}(\mathbf{x})^+$ of $J_{\mathbf{m}}(\mathbf{x})$. The Gauss-Jordan method for the sum-squared loss can be interpreted as multiplying the gradient $\nabla l(\mathbf{m})$ by the pseudo-inverse of the jacobian of $\mathbf{m}$

[28] The Jacobian is a $p \times n$ matrix of the first derivatives of a vector valued function, where p is arity of $\mathbf{m}$. The $(i,j)^{th}$ entry of the Jacobian is the derivative of the i^{th} output with respect to the j^{th} variable, that is $\frac{\partial m_i}{\partial x_j}$. For $m = 1$, the Jacobian is the gradient vector.

instead of its transpose (which is what the gradient descent method would do). Though the Gauss-Newton method has been traditionally used for non-linear least squared problems, recently it has also seen use for the cross entropy loss function. This method is a simple adoption of the Newton's method, with the advantage that second derivatives, which can be computationally expensive and challenging to compute, are not required.

4.5.5 Levenberg-Marquardt

Like the Gauss-Newton method, the Levenberg-Marquardt method has its main application in the least squares curve fitting problem (as also in the minimum cross-entropy problem). The Levenberg-Marquardt method interpolates between the Gauss-Newton algorithm and the method of gradient descent. The Levenberg-Marquardt algorithm is more robust than the Gauss Newton algorithm - it often finds a solution even if it starts very far off the final minimum. On the other hand, for well-behaved functions and reasonable starting parameters, this algorithm tends to be a bit slower than the Gauss Newton algorithm. The Levenberg-Marquardt method aims to reduce the uncontrolled step size often taken by the Newton's method and thus fix the stability issue of the Newton's method. The update rule is given by

$$\Delta\mathbf{x} = -\left(G_f(\mathbf{x}) + \lambda \operatorname{diag}(G_f)\right)^{-1} J^T_{\mathbf{m}}(\mathbf{x})\nabla l(\mathbf{m})$$

where G_f is the Gauss-Newton approximation to $\nabla^2 f(\mathbf{x})$ and is assumed to be positive semi-definite. This method is one of the work-horses of modern optimization. The parameter $\lambda \geq 0$ adaptively controlled, limits steps to an elliptical model-trust region[29]. This is achieved by adding λ to the smallest eigenvalues of G_f, thus restricting all eigenvalues of the matrix to be above λ so that the elliptical region has diagonals of shorter length that inversely vary as the eigenvalues (*c.f.* page 3.11.3). While this method fixes the stability issues in Newtons method, it still requires the $O(n^3)$ time required for matrix inversion.

4.5.6 BFGS

The Broyden-Fletcher-Goldfarb-Shanno[30] (BFGS) method uses linear algebra to iteratively update an estimate $B^{(k)}$ of $\left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1}$ (the inverse of the curvature matrix), while ensuring that the approximation to the hessian inverse is symmetric and positive definite. Let $\Delta\mathbf{x}^{(k)}$ be the direction vector for the k^{th} step obtained as the solution to

$$\Delta\mathbf{x}^{(k)} = -B^{(k)}\nabla f(\mathbf{x}^{(k)})$$

The next point $\mathbf{x}^{(k+1)}$ is obtained as

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)}\Delta\mathbf{x}^{(k)}$$

[29] Essentially the algorithm approximates only a certain region (the so-called trust region) of the objective function with a quadratic as opposed to the entire function.

[30] The the 4 authors wrote papers for exactly the same method at exactly at the same time.

where $t^{(k)}$ is the step size obtained by line search. Let $\Delta \mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)})$. Then the BFGS update rule is derived by imposing the following logical conditions:

1. $\Delta \mathbf{x}^{(k)} = -B^{(k)} \nabla f(\mathbf{x}^{(k)})$ with $B^{(k)} \succ 0$. That is, $\Delta \mathbf{x}^{(k)}$ is the minimizer of the convex quadratic model

$$Q^{(k)}(\mathbf{p}) = f(\mathbf{x}^{(k)}) + \nabla^T f(\mathbf{x}^{(k)})\mathbf{p} + \frac{1}{2}\mathbf{p}^T \left(B^{(k)}\right)^{-1} \mathbf{p}$$

2. $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)} \Delta \mathbf{x}^{(k)}$, where $t^{(k)}$ is obtained by line search.

3. The gradient of the function $Q^{(k+1)} = f(\mathbf{x}^{(k+1)}) + \nabla^T f(\mathbf{x}^{(k+1)})\mathbf{p} + \frac{1}{2}\mathbf{p}^T \left(B^{(k+1)}\right)^{-1} \mathbf{p}$ at $\mathbf{p} = \mathbf{0}$ and $\mathbf{p} = -t^{(k)} \Delta \mathbf{x}^{(k)}$ agrees with gradient of f at $\mathbf{x}^{(k+1)}$ and $\mathbf{x}^{(k)}$ respectively. While the former condition is naturally satisfied, the latter need to be imposed. This quasi-Newton condition yields

$$\left(B^{(k+1)}\right)^{-1} \left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)}).$$

   This equation is called the *secant equation.*

4. Finally, among all symmetric matrices satisfying the secant equation, $B^{(k+1)}$ is closest to the current matrix $B^{(k)}$ in some norm. Different matrix norms give rise to different quasi-Newton methods. In particular, when the norm chosen is the Frobenius norm, we get the following BGFS update rule

$$B^{(k+1)} = B^{(k)} + R^{(k)} + S^{(k)}$$

   where,

$$R^{(k)} = \frac{\Delta \mathbf{x}^{(k)} \left(\Delta \mathbf{x}^{(k)}\right)^T}{\left(\Delta \mathbf{x}^{(k)}\right)^T \Delta \mathbf{g}^{(k)}} - \frac{B^{(k)} \Delta \mathbf{g}^{(k)} \left(\Delta \mathbf{g}^{(k)}\right)^T \left(B^{(k)}\right)^T}{\left(\Delta \mathbf{g}^{(k)}\right)^T B^{(k)} \Delta \mathbf{g}^{(k)}}$$

   and

$$S^{(k)} = \mathbf{u} \left(\Delta \mathbf{x}^{(k)}\right)^T B^{(k)} \Delta \mathbf{x}^{(k)} \mathbf{u}^T$$

   with

$$\mathbf{u} = \frac{\Delta \mathbf{x}^{(k)}}{\left(\Delta \mathbf{x}^{(k)}\right)^T \Delta \mathbf{g}^{(k)}} - \frac{B^{(k)} \Delta \mathbf{g}^{(k)}}{\left(\Delta \mathbf{g}^{(k)}\right)^T B^{(k)} \Delta \mathbf{g}^{(k)}}$$

   We have made use of the Sherman Morrison formula that determines how updates to a matrix relate to the updates to the inverse of the matrix.

Any operation involving n^3/n^2 basic operations is multiplication or addition.

The approximation to the Hessian is updated by analyzing successive gradient vectors and thus the Hessian matrix does not need to be computed at any stage. The initial estimate $B^{(0)}$ can be taken to be the identity matrix, so that the first step is equivalent to a gradient descent. The BFGS method has a reduced complexity of $O(n^2)$ time per iteration. The method is summarized

LBFGS: Limited memory BFGS .. by restricting search space of $B^{(k+1)}$ further. . .

Tradeoff: Computation/memory per iteration Vs. total # iterations

Find a starting point $\mathbf{x}^{(0)} \in \mathcal{D}$ and an approximate $B^{(0)}$ (which could be I).
Select an appropriate tolerance $\epsilon > 0$.
repeat
1. Set $\Delta\mathbf{x}^{(k)} = -B^{(k)}\nabla f(\mathbf{x}^{(k)})$.
2. Let $\lambda^2 = \nabla^T f(\mathbf{x}^{(k)}) B^{(k)} \nabla f(\mathbf{x}^{(k)})$.
3. If $\frac{\lambda^2}{2} \leq \epsilon$, **quit**.
4. Set step size $t^{(k)} = 1$.
5. Obtain $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)}\Delta\mathbf{x}^{(k)}$.
6. Compute $\Delta\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k+1)}) - \nabla f(\mathbf{x}^{(k)})$.
7. Compute $R^{(k)}$ and $S^{(k)}$.
8. Compute $B^{(k+1)} = B^{(k)} + R^{(k)} + S^{(k)}$.
6. Set $k = k + 1$.

until

Figure 4.50: The BFGS method.

in Figure 4.50 The BFGS [**?**] method approaches the Newton's method in behaviour as the iterate approaches the solution. They are much faster than the Newton's method in practice. It has been proved that when BFGS is applied to a convex quadratic function with exact line search, it finds the minimizer within n steps. There is a variety of methods related to BFGS and collectively they are known as Quasi-Newton methods. They are preferred over the Newton's method or the Levenberg-Marquardt when it comes to speed. There is a variant of BFGS, called LBFGS [**?**], which stands for "Limited memory BFGS method". LBFGS employs a limited-memory quasi-Newton approximation that does not require much storage or computation. It limites the rank of the inverse of the hessian to some number $\gamma \in \Re$ so that only $n\gamma$ numbers have to be stored instead of n^2 numbers. For general non-convex problems, LBFGS may fail when the initial geometry (in the form of $B^{(0)}$) has been placed very close to a saddle point. Also, LBFGS is very sensitive to line search.

Recently, L-BFGS has been observed [**?**] to be the most effective parameter estimation method for Maximum Entropy model, much better than improved iterative scaling [**?**] (IIS) and generalized iterative scaling [**?**] (GIS).

4.5.7 Solving Large Sparse Systems

In many convex optimization problems such as least squares, newton's method for optimization, *etc.*, one has to deal with solving linear systems involving large and sparse matrices. Elimination with ordering can be expensive in such cases. A lot of work has gone into solving such problems efficiently[31] using iterative

[31] Packages such as LINPack (which is now renamed to LAPACK), EiSPACK, MINPACK, *etc.*, which can be found under the netlib respository, have focused on efficiently solving large linear systems under general conditions as well as specific conditions such as symmetry or positive definiteness of the coefficient matrix.

methods instead of direct elimination methods. An example iterative method is for solving a system $A\mathbf{x} = \mathbf{b}$ by repeated multiplication of a large and sparse matrix A by vectors to quickly get an answer $\widehat{\mathbf{x}}$ that is sufficiently close to the optimal solution $\mathbf{x}^*$. Multiplication of an $n \times n$ sparse matrix A having k non-zero entries with a vector of dimension n takes $O(kn)$ time only, in contrast to $O(n^3)$ time for Gauss elimination. We will study three types of methods for solving systems with large and sparse matrices:

1. *Iterative Methods.*
2. *Multigrid Methods.*
3. *Krylov Methods.*

The most famous and successful amongst the Krylov methods has been the *conjugate gradient method*, which works for problems with positive definite matrices.

Iterative Methods

The central step in an iteration is

$$P\mathbf{x}_{k+1} = (P - A)\mathbf{x}_k + \mathbf{b}$$

where $\mathbf{x}_k$ is the estimate of the solution at the k^{th} step, for $k = 0, 1, \ldots$. If the iterations converge to the solution, that is, if $\mathbf{x}_{k+1} = \mathbf{x}_k$ one can immediatly see that the solution is reached. The choice of matrix P, which is called the *preconditioner*, determines the rate of convergence of the solution sequence to the actual solution. The initial estimate $\mathbf{x}_0$ can be arbitrary for linear systems, but for non-linear systems, it is important to start with a good approximation. It is desirable to choose the matrix P reasonably close to A, though setting $P = A$ (which is referred to as perfect preconditioning) will entail solving the large system $A\mathbf{x} = \mathbf{b}$, which is undesirable as per our problem definition. If $\mathbf{x}^*$ is the actual solution, the relationship between the errors $\mathbf{e}_k$ and $\mathbf{e}_{k+1}$ at the k^{th} and $(k+1)^{th}$ steps respectively can be expressed as

$$P\mathbf{e}_{k+1} = (P - A)\mathbf{e}_k$$

where $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$. This is called the *error equation*. Thus,

$$\mathbf{e}_{k+1} = (I - P^{-1}A)\mathbf{e}_k = M\mathbf{e}_k$$

Whether the solutions are convergent or not is controlled by the matrix M. The iterations are stationary (that is, the update is of the same form at every step). On the other hand, Multigrid and Krylov methods adapt themselves across iterations to enable faster convergence. The error after k steps is given by

$$\mathbf{e}_k = M^k\mathbf{e}_0 \tag{4.96}$$

Using the idea of eigenvector decomposition presented in (3.101), it can be proved that the error vector $\mathbf{e}_k \rightarrow \mathbf{0}$ if the absolute values of all the eigenvalues of M are less than 1. This is the *fundamental theorem of iteration*. In this case, the rate of convergence of $\mathbf{e}_k$ to $\mathbf{0}$ is determined by the maximum absolute eigenvalue of M, called the spectral radius of M and denoted by $\rho(M)$.

Any iterative method should attempt to choose P so that it is easy to compute $\mathbf{x}_{k+1}$ and at the same time, the matrix $M = I - P^{-1}A$ has small eigenvalues. Corresponding to various choices of the preconditioner P, there exist different iterative methods.

1. *Jacobi:* In the simplest setting, P can be chosen to be a diagonal matrix with its diagonal borrowed from A. This choice of A is correponds to the *Jacobi* method. The value of $\rho(M)$ is less than 1 for the Jacobi method, though it is often very close to 1. Thus, the Jacobi method does converge, but the convergence can be very slow in practice. While the residual $\widehat{\mathbf{r}} = A\widehat{\mathbf{x}} - \mathbf{b}$ converges rapidly, the error $\overline{\mathbf{x}} = \widehat{\mathbf{x}} - \mathbf{x}^*$ decreases rapidly in the beginning, but the rate of decrease of $\overline{\mathbf{x}}$ reduces as iterations proceed. This happens because $\overline{\mathbf{x}} = A^{-1}\widehat{\mathbf{r}}$ and A^{-1} happens to have large condition number for sparse matrices. In fact, it can be shown that Jacobi can take upto n^β iterations to reduce the error $\overline{\mathbf{x}}$ by a factor β.

   We will take an example to illustrate the Jacobi method. Consider the following $n \times n$ tridiagonal matrix A.

$$A = \begin{bmatrix} 2 & -1 & 0 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ -1 & 2 & -1 & \ldots & 0 & 0 & 0 & \ldots & 0 \\ . & . & . & \ldots & . & . & . & \ldots & . \\ . & . & . & \ldots & . & . & . & \ldots & . \\ 0 & 0 & 0 & \ldots & -1 & 2 & -1 & \ldots & 0 \\ 0 & 0 & 0 & \ldots & 0 & -1 & 2 & \ldots & 0 \\ . & . & . & \ldots & . & . & . & \ldots & . \\ . & . & . & \ldots & . & . & . & \ldots & . \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & 2 \end{bmatrix} \tag{4.97}$$

   The absolute value of the i^{th} eigenvalue of M is $\cos\frac{j\pi}{n+1}$ and its spectral radius is $\rho(M) = \cos\frac{\pi}{n+1}$. For extremely large n, the spectral radius is approximately $1 - \frac{1}{2}\left(\frac{\pi}{n+1}\right)^2$, which is very close to 1. Thus, the Jacobi steps converge very slowly.

2. *Gauss-Seidel:* The second possibility is to choose P to be the lower-triangular part of A. The method for this choice is called the *Gauss-Siedel* method. For the example tridiagonal matrix A in (4.97), matrix

$P - A$ will be the strict but negated upper-triangular part of A. For the Gauss-Seidel technique, the components of $\mathbf{x}_{k+1}$ can be determined from $\mathbf{x}_k$ using back-substitution. The Gauss-sidel method provides only a constant factor improvement over the *Jacobi* method.

3. *Successive over-relaxation:* In this method, the preconditioner is obtained as a weighted composition of the preconditioners from the above two methods. It is abbreviated as SOR. In history, this was the first step of progress beyond Jacobi and Gauss-Seidel.

4. *Incomplete LU:* This method involves an incomplete elimination on the sparse matrix A. For a sparse matrix A, many entries in its LU decomposition will comprise of nearly 0 elements; the idea behind this method is to treat such entries as 0's. Thus, the L and U matrices are approximated based on the tolerance threshold; if the tolerance threshold is very high, the factors are exact. Else they are approximate.

Multigrid Methods

Multigrid methods come very handy in solving large sparse systems, especially differential equations using a hierarchy of discretizations. This approach often scales linearly with the number of unknowns n for a pre-specified accuracy threshold. The overall multi-grid algorithm for solving $A_h\mathbf{u}_h = \mathbf{b}_h$ with residual given by $\mathbf{r}_h = \mathbf{b} - A\mathbf{u}_h$ is

1. **Smoothing:** Perform a few (say 2-3) iterations on $A_h\mathbf{u} = \mathbf{b}_h$ using either Jacobi or Gauss-sidel. This will help remove high frequency components of the residual $\mathbf{r} = \mathbf{b} - A_h\mathbf{u}$. This step is really outside the core of the multigrid method. Denote the solution obtained by $\mathbf{u}_h$. Let $\mathbf{r}_h = \mathbf{b} - A_h\mathbf{u}_h$.

2. **Restriction:** Restrict $\mathbf{r}_h$ to coarse grid by setting $\mathbf{r}_{2h} = R\mathbf{r}_h$. That is, $\mathbf{r}_h$ is downsampled to yield $\mathbf{r}_{2h}$ Let $k < n$ characterize the coarse grid. Then, the $k \times n$ matrix R is called the restriction matrix and it takes the residuals from a finer to a coarser grid. It is typically scaled to ensure that a vector of 1's on the fine mesh gets transformed to a vector of 1's on a coarse mesh. Calculations on the coarse grid are way faster than on the finer grid.

3. Solve $A_{2h}\mathbf{e}_{2h} = \mathbf{r}_{2h}$ with $A_{2h} = RA_hN$, which is a natural construction for the coarse mesh operation. This could be done by running few iterations of Jacobi, starting with $\mathbf{e}_{2h} = \mathbf{0}$.

4. **Interpolation/Prolongation:** This step involves interpolating the correction computed on a coarses grid to a finer grid. Interpolate back to $\mathbf{e}_h = N\mathbf{e}_{2h}$. Here N is a $k \times n$ interpolation matrix and it takes the residuals from a coarse to a fine grid. It is generally a good idea to connect N to R by setting $N = \alpha R^T$ for some scaling factor α. Add $\mathbf{e}_h$ to $\mathbf{u}_h$. The

analytical expression for $\mathbf{e}_h$ is

$$\mathbf{e}_h = N(A_{2h})^{-1}RA_h(\mathbf{u}-\mathbf{u}_h) = \underbrace{\big(N(RAN)^{-1}RA_h(\mathbf{u}-\mathbf{u}_h)\big)}_{S}(\mathbf{u}-\mathbf{u}_h)$$

A property of the $n\times n$ matrix S is that $S^2 = S$. Thus, the only eigenvalues of S are 0 and 1. Since S is of rank $k < n$, k of its eigenvalues are 1 and $n-k$ are 0. Further, the eigenvectors for the 1 eigenvalues, which are in the null space of $I-S$ form the coarse mesh (and correspond to low frequency vectors) whereas the eigenvectors for the 0 eigenvalues, which are in the null space of S form the fine mesh (and correspond to high frequency vectors). We can easily derive that k eigenvalues of $I-S$ will be 0 and $n-k$ of them will be 1.

5. Finally as a post-smoothing step, iterate $A\mathbf{u}_h = \mathbf{b}_h$ starting from the improved $\mathbf{u}_h + \mathbf{e}_h$, using Jacobi or Gauss-Sidel.

Overall, the error $\mathbf{e}^k$ after k steps will be of the form

$$\mathbf{e}_k = (M^t(I-S)M^t)\mathbf{e}_0 \tag{4.98}$$

where t is the number of Jacobi steps performed in (1) and (5). Typically t is 2 or 3. When you contrast (4.98) against (4.96), we discover that $\rho(M) \geq\geq \rho(M^t(I-S)M^t)$. As t increases, $\rho(M^t(I-S)M^t)$ further decreases by a smaller proportion.

In general, you could have multiple levels of coarse grids corresponding to $2h$, $4h$, $8h$ and so on, in which case, steps (2), (3) and (4) would be repeated as many times with varying specifications of the coarseness. If A is an $n\times n$ matrix, multi-grid methods are known to run in $O(n^2)$ floating point operations (flops). The multi-grid method could be used an iterative method to solve a linear system. Alternatively, it could be used to obtain the preconditioner.

Linear Conjugate Gradient Method

The conjugate gradient method is one of the most popular Krylov methods. The Krylov matrix K_j, for the linear system $A\mathbf{u} = \mathbf{b}$ is given by

$$K_j = \begin{bmatrix}\mathbf{b} & A\mathbf{b} & A^2\mathbf{b} & \dots & A^{j-1}\mathbf{b}\end{bmatrix}$$

The columns of K_j are easy to compute; each column is a result of a matrix multiplication A with the previous column. Assuming we are working with sparse matrices, (often symmetric matrices such as the Hessian) these computations will be inexpensive. The Krylov space $\mathcal{K}_j$ is the column space of K_j. The columns of K_j are computed during the first j steps of an iterative method such as Jacobi. Most Krylov methods opt to choose vectors from $\mathcal{K}_j$ instead of a fixed choice of the j^{th} column of K_j. A method such as *MinRes* chooses a vector

$\mathbf{u}_j \in \mathcal{K}_j$ that minimizes $\mathbf{b} - A\mathbf{u}_j$. One of the well-known Krylov methods is the *Conjugate gradient* method, which assumes that the matrix A is symmetric and positive definite and is faster than MinRes. In this method, the choice of u_j is made so that $\mathbf{b} - A\mathbf{u}_j \perp \mathcal{K}_j$. That is, the choice of $\mathbf{u}_j$ is made so that the residual $\mathbf{r}_j = \mathbf{b} - A\mathbf{u}_j$ is orthogonal to the space $\mathcal{K}_j$. The conjugate gradient method gives an exact solution to the linear system if $j = n$ and that is how they were originally designed to be (and put aside subsequently). But later, they were found to give very good approximations for $j << n$.

The discussions that follow require the computation of a basis for $\mathcal{K}_j$. It is always prefered to have a basis matrix with low condition number[32], and an orthonormal basis is a good choice, since it has a condition number of 1 (the basis consisting of the columns of K_j turns out to be not-so-good in practice). The *Arnoldi* method yields an orthonormal Krylov basis $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_j$ to get something that is numerically reasonable to work on. The method is summarized in Figure 4.51. Though the underlying idea is borrowed from Gram-Schmidt at every step, there is a difference; the vector $\mathbf{t}$ is $\mathbf{t} = A\mathbf{Q}_j$ as against simply $\mathbf{t} = \mathbf{q}_j$. Will it be expensive to compute each $\mathbf{t}$? Not if A is symmetric. First we note that by construction, $AQ = QH$, where $\mathbf{q}_j$ is the j^{th} column of Q. Thus, $H = Q^T AQ$. If A is symmetric, then so is H. Further, since H has only one lower diagonal (by construction), it must have only one higher diagonal. Therefore, H must be symmetric and tridiagonal. If A is symmetric, it suffices to subtract only the components of $\mathbf{t}$ in the direction of the last two vectors $\mathbf{q}_{j-1}$ and $\mathbf{q}_j$ from $\mathbf{t}$. Thus, for a symmetric A, the inner 'for' loop needs to iterate only over $i = j - 1$ and $i = j$.

Since A and H are similar matrices, they have exactly the same eigenvalues. Restricting the computation to a smaller number of orthonormal vectors (for some $k << n$), we can save time for computing Q_k and H_k. The k eigenvalues of H_k are good approximations to the first k eigenvalues of H. This is called the *Arnoldi-Lanczos* method for finding the top k eigenvalues of a matrix.

As an example, consider the following matrix A.

$$A = \begin{bmatrix} 0.5344 & 1.0138 & 1.0806 & 1.8325 \\ 1.0138 & 1.4224 & 0.9595 & 0.8234 \\ 1.0806 & 0.9595 & 1.0412 & 1.0240 \\ 1.8325 & 0.8234 & 1.0240 & 0.7622 \end{bmatrix}$$

[32] For any matrix A, the condition number $\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$, where $\sigma_{max}(A)$ and $\sigma_{min}(A)$ are maximal and minimal singular values of A respectively. Recall from Section 3.13 that the i^{th} eigenvalue of $A^T A$ (the gram matrix) is the square of the i^{th} singular value of A. Further, if A is normal, $\kappa(A) = \left|\frac{\lambda_{max}(A)}{\lambda_{min}(A)}\right|$, where $\lambda_{max}(A)$ and $\lambda_{min}(A)$ are eigevalues of A with maximal and minimal magnitudes respectively. All orthogonal, symmetric, and skew-symmetric matrices are normal. The condition number measures how much the columns/rows of a matrix are dependent on each other; higher the value of the condition number, more is the linear dependence. Condition number 1 means that the columns/rows of a matrix are linearly independent.

Set $\mathbf{q}_1 = \frac{1}{||\mathbf{b}||}\mathbf{b}$. //The first step in Gram schmidt.
for $j = 1$ to $n-1$ **do**
  $\mathbf{t} = A\mathbf{q}_j$.
  for $i = 1$ to j **do**
    //If A is symmetric, it will be $i = \max(1, j-1)$ to j.
    $H_{i,j} = \mathbf{q}_i^T\mathbf{t}$.
    $\mathbf{t} = \mathbf{t} - H_{i,j}\mathbf{q}_i$.
  end for
  $H_{j+1,j} = ||\mathbf{t}||$.
  $\mathbf{q}_{j+1} = \frac{1}{||\mathbf{t}||}\mathbf{t}$.
end for
$\mathbf{t} = A\mathbf{q}_n$.
for $i = 1$ to n **do**
  //If A is symmetric, it will be $i = n-1$ to n.
  $H_{i,n} = \mathbf{q}_i^T\mathbf{t}$.
  $\mathbf{t} = \mathbf{t} - H_{in}\mathbf{q}_i$.
end for
$H_{j+1,j} = ||\mathbf{t}||$.
$\mathbf{q}_{j+1} = \frac{1}{||\mathbf{t}||}\mathbf{t}$.

Figure 4.51: The Arnoldi algorithm for computing orthonormal basis.

and the vector $\mathbf{b}$

$$\mathbf{b} = \begin{bmatrix} 0.6382 & 0.3656 & 0.1124 & 0.5317 \end{bmatrix}^T$$

The matrix K_4 is

$$K_4 = \begin{bmatrix} 0.6382 & 1.8074 & 8.1892 & 34.6516 \\ 0.3656 & 1.7126 & 7.5403 & 32.7065 \\ 0.1124 & 1.7019 & 7.4070 & 31.9708 \\ 0.5317 & 1.9908 & 7.9822 & 34.8840 \end{bmatrix}$$

Its condition number is 1080.4.

The algorithm in Figure 4.51 computed the following basis for the matrix K_4.

$$Q_4 = \begin{bmatrix} 0.6979 & -0.3493 & 0.5101 & -0.3616 \\ 0.3998 & 0.2688 & 0.2354 & 0.8441 \\ 0.1229 & 0.8965 & 0.1687 & -0.3908 \\ 0.5814 & 0.0449 & -0.8099 & -0.0638 \end{bmatrix}$$

The coefficient matrix H_4 is

$$H_4 = \begin{bmatrix} 3.6226 & 1.5793 & 0 & 0 \\ 1.5793 & 0.6466 & 0.5108 & 0 \\ 0 & 0.5108 & -0.8548 & 0.4869 \\ 0 & 0 & 0.4869 & 0.3459 \end{bmatrix}$$

and its eigenvalues are 4.3125, 0.5677, -1.2035 and 0.0835. On the other hand, the following matrix H_3 (obtained by restricting to K_3) has eigenvalues 4.3124, 0.1760 and -1.0741.

The basic conjugate gradient method selects vectors in $\mathbf{x}_k \in \mathcal{K}_k$ that approach the exact solution to $A\mathbf{x} = \mathbf{b}$. Following are the main ideas in the conjugate gradient method.

1. The rule is to select an $\mathbf{x}_k$ so that the new residual $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$ is orthogonal to all the previous residuals. Since $A\mathbf{x}_k \in \mathcal{K}_{k+1}$, we must have $\mathbf{r}_k \in \mathcal{K}_{k+1}$ and $\mathbf{r}_k$ must be orthogonal to all vectors in $\mathcal{K}_k$. Thus, $\mathbf{r}_k$ must be a multiple of $\mathbf{q}_{k+1}$. This holds for all k and implies that

   $$\mathbf{r}_k^T \mathbf{r}_i = 0$$

   for all $i < k$.

2. Consequently, the difference $\mathbf{r}_k - \mathbf{r}_{k-1}$, which is a linear combination of $\mathbf{q}_{k+1}$ and $\mathbf{q}_k$, is orthogonal to each subspace $\mathcal{K}_i$ for $i < k$.

3. Now, $\mathbf{x}_i - \mathbf{x}_{i-1}$ lies in the subspace $\mathcal{K}_i$. Thus, $\Delta\mathbf{r} = \mathbf{r}_k - \mathbf{r}_{k-1}$ is orthogonal to all the previous $\Delta\mathbf{x} = \mathbf{x}_i - \mathbf{x}_{i-1}$. Since $\mathbf{r}_k - \mathbf{r}_{k-1} = -A(\mathbf{x}_k - \mathbf{x}_{k-1})$, we get the following 'conjugate directions' condition for the updates

   $$(\mathbf{x}_i - \mathbf{x}_{i-1})^T A(\mathbf{x}_k - \mathbf{x}_{k-1}) = 0$$

   for all $i < k$. This is a necessary and sufficient condition for the orthogonality of the new residual to all the previous residuals. Note that while the residual updates are orthogonal in the usual inner product, the variable updates are orthogonal in the inner product with respect to A.

The basic conjugate gradient method consists of 5 steps. Each iteration of the algorithm involves a multiplication of vector $\mathbf{d}_{k-1}$ by A and computation of two inner products. In addition, an iteration also involves around three vector updates. So each iteration should take time upto $(2+\theta)n$, where θ is determined by the sparsity of matrix A. The error $\mathbf{e}_k$ after k iterations is bounded as follows.

$$||\mathbf{e}_k||_A = (\mathbf{x}_k - \mathbf{x})^T A(\mathbf{x}_k - \mathbf{x}) \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}\right)^k ||\mathbf{e}_0||$$

The 'gradient' part of the name *conjugate gradient* stems from the fact that solving the linear system $A\mathbf{x} = \mathbf{b}$ is corresponds to finding the minimum value

Basic idea: $\min_x f(x) = \frac{1}{2}x^T Ax - b^T x$ (say $A \succ 0$)

Consider the innerproduct (& norm) induced by A

$$\langle x, y \rangle_A = x^T A y$$

Consider a Gram Schmidt orthogonalization process to obtain A-orthogonal vectors $\{d_1, d_2 .. d_k\}$

$$\langle d_i, d_j \rangle_A = 0 \quad \forall \; i \neq j$$

Let $v_0 \cdots v_{n-1}$ be Linearly independent vectors

$$d_0 = v_0$$
$$\vdots$$
$$d_k = v_k - \sum_{i=0}^{k-1} \frac{v_k^T A d_i}{d_i^T A d_i} d_i$$

Let $x = \sum_{i=0}^{n-1} \alpha_i d_i \Rightarrow f(x) = \frac{1}{2}\|x\|_A^2 - b^T x$

$$= \sum_{i=0}^{n-1} \underbrace{\left(\frac{1}{2}\alpha_i^2 \|d_i\|_A^2 - \alpha_i b^T d_i \right)}_{\phi_i(\alpha_i)}$$

$$= \sum_{i=0}^{n-1} \phi_i(\alpha_i)$$

separable in α

$$\Rightarrow \min_x f(x) = \sum_{i=0}^{n-1} \min_{\alpha_i} \phi_i(\alpha_i)$$

Conjugate directions method $\Rightarrow$ Start with $x^{(0)}$

$\Rightarrow x = x^{(0)} + \sum_{i=0}^{n-1} \alpha_i d_i$ will still have separability in α

Expanding manifold property [ASIDE]

Conjugate gradient satisfies expanding manifold property: $G^k = \left\{ x^{(0)} + \sum_{i=0}^{k} d_i d_i \right\}$

Affine subspace at k^{th} iteration

Claim k^{th} conjugate direction step:

$$x^{(k)} = \underset{x \in G^k}{\operatorname{argmin}} f(x)$$

Claim: $\nabla f(x^{(k)})$ will be orthogonal to $d_0, d_1 \ldots d_{k-1}$ (ie all previous directions)

Conjugate gradient algo: (for Quadratic opt)

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

$$\nabla f(x^k) = Qx^{(k)} - b$$

Initial: $d_0 = -\nabla f(x^{(0)}) = -Qx^{(0)} - b$

k^{th} step: $x^{(k)} = x^{(k-1)} + \alpha_{(k-1)} d_{(k-1)}$ .. Exact line search

$$d_k = -\nabla f(x^{(k)}) + \sum_{i=0}^{k-1} \frac{\nabla^T f(x^k) A d_i}{d_i^T A d_i}$$

← By Gram Schmidt

... Gram Schmidt for A-orthogonality of $\{d_0, .. d_k\}$

Further Simplifications

(a) $d_{(k-1)} = \frac{1}{\alpha_{(k-1)}}(x^{(k)} - x^{(k-1)}) \Rightarrow A d_{k-1} = \frac{1}{\alpha_{k-1}} A(x^{(k)} - x^{(k-1)})$

$$= \frac{1}{\alpha_{k-1}}\left(\nabla f(x^{(k)}) - \nabla f(x^{(k-1)})\right)$$

(b) By expanding manifold property:

$\nabla f(x^{(k)}) \perp \nabla f(x^{(k-1)}), \nabla f(x^{(k-2)}) \ldots \nabla f(x^{(0)})$

$$\Rightarrow d_k = -\nabla f(x^{(k)}) + \frac{\nabla^T f(x^k)\left(\nabla f(x^k) - \nabla f(x^{(k-1)})\right)}{d_{k-1}^T A d_{k-1}}$$

} Polak Riebiere method

$\mathbf{x}_0 = \mathbf{0}$, $\mathbf{r}_0 = \mathbf{b}$, $\mathbf{d}_0 = \mathbf{r}_0$, $k = 1$.
repeat
1. $\alpha_k = \frac{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}{\mathbf{d}_{k-1}^T A \mathbf{d}_{k-1}}$. //Step length for next update. This corresponds to the entry $H_{k,k}$.
2. $\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k \mathbf{d}_{k-1}$.
3. $\mathbf{r}_k = \mathbf{r}_{k-1} - \alpha_k A \mathbf{d}_{k-1}$. //New residual obtained using $\mathbf{r}_k - \mathbf{r}_{k-1} = -A(\mathbf{x}_k - \mathbf{x}_{k-1})$.
4. $\beta_k = \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{r}_{k-1}^T \mathbf{r}_{k-1}}$. //Improvement over previous step. This corresponds to the entry $H_{k,k+1}$.
5. $\mathbf{d}_k = \mathbf{r}_k + \beta_k \mathbf{d}_{k-1}$. //The next search direction, which should be orthogonal to the search direction just used.

$k = k + 1$.
until $\beta_k < \theta$.

Fletcher Reeves update

Figure 4.52: The conjugate gradient algorithm for solving $A\mathbf{x} = \mathbf{b}$ or equivalently, for minimizing $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$.

of the convex (for positive definite A) energy function $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} = \mathbf{r}$ by setting its gradient $A\mathbf{x} - \mathbf{b}$ to the zero vector. The steepest descent method makes a move along at the direction of the residual $\mathbf{r}$ at every step but it does not have a great convergence; we land up doing a lot of work to make a little progress. In contrast, as reflect in the step $\mathbf{d}_k = \mathbf{r}_k + \beta_k \mathbf{d}_{k-1}$, the conjugate gradient method makes a step in the direction of the residual, but only after removing any component β_k along the direction of the step it just took. Figures 4.53 and 4.54 depict the steps taken by the steepest descent and the conjugate descent techniques respectively, on the level-curves of the function $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$, in two dimensions. It can be seen that while the steepest descent technique requires many iterations for convergence, owing to its oscillations, the conjugate gradient method takes steps that are orthogonal with respect to A (or are orthogonal in the transfomed space obtained by multiplying with A), thus taking into account the geometry of the problem and taking a fewer number of steps. If the matrix A is a hessian, the steps taken by conjugate gradient are orthogonal in the local Mahalonobis metric induced by the curvature matrix A. Note that if $\mathbf{x}^{(0)} = \mathbf{0}$, the first step taken by both methods will be the same.

The conjugate gradient method is guaranteed to reach the minimum of the energy function E in exactly n steps. Further, if A has only r distinct eigenvalues, then the conjugate gradient method will terminate at the solution in at most r iterations.

4.5.8 Conjugate Gradient

We have seen that the Conjugate Gradient method in Figure 4.52 can be viewed as a minimization algorithm for the convex quadratic function $E(\mathbf{x}) =$

Suppose Conjugate Gradient is applied to solving $\min_x x^T A x - b^T x$

Then: $$\frac{\|x^* - x^{(k)}\|_A}{\|x^* - x^{(0)}\|_A} \leq \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \quad \text{for} \quad k \geq 0$$

$$\kappa = \text{cond}(A) = \frac{\lambda_{max}}{\lambda_{min}}$$

$$\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k \to 1 - 2\sqrt{\frac{\lambda_{min}}{\lambda_{max}}}$$

For gradient descent, rate was $1 - 2\left(\frac{\lambda_{min}}{\lambda_{max}}\right)$

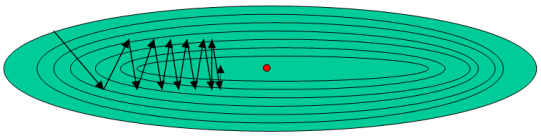

Figure 4.53: Illustration of the steepest descent technique on level curves of the function $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$.

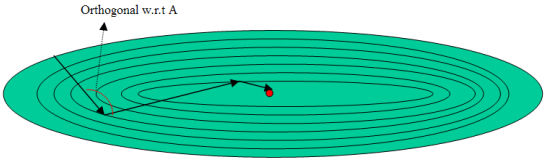

Figure 4.54: Illustration of the conjugate gradient technique on level curves of the function $E(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$.

$\frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$. Can the approach be adapted to minimize general nonlinear convex functions? Nonlinear variants of the conjugate gradient are well studied [**?**] and have proved to be quite successful in practice. The general conjugate gradient method is essentially an incremental way of doing second order search.

Fletcher and Reeves showed how to extend the conjugate gradient method to nonlinear functions by making two simple changes[33] to the algorithm in Figure 4.52. First, in place of the exact line search formula in step (1) for the step length α_k, we need to perform a line search that identifies an approximate minimum of the nonlinear function f along $\mathbf{d}^{(k-1)}$. Second, the residual $\mathbf{r}^{(k)}$, which is simply the gradient of E (and which points in the direction of decreasing value of E), must be replaced by the gradient of the nonlinear objective f, which serves a similar purpose. These changes give rise to the algorithm for nonlinear optimization outlined in Figure 4.55. The search directions $\mathbf{d}^{(k)}$ are computed by Gram-Schmidt conjugation of the residuals as with linear conjugate gradient. The algorithm is very sensitive to the line minimization step and it generally requires a very good line minimization. Any line search procedure that yields an α_k satisfying the strong Wolfe conditions (see (4.90) and (4.91)) will ensure that all directions $\mathbf{d}^{(k)}$ are descent directions for the function f, otherwise, $\mathbf{d}^{(k)}$ may cease to remian a descent direction as iterations proceed. We note that each iteration of this method costs on $O(n)$, as against the Newton or quasi-newton methods which cost atleast $O(n^2)$ owing to matrix operations. Most often, it yields optimal progress after $h << n$ iterations. Due to this property, the conjugate gradient method drives nearly all large-scale optimization today.

[33] We note that in the algorithm in Figure 4.52, the residuals $\mathbf{r}^{(k)}$ in successive iterations (which are gradients of E) are orthogonal to each other, while the corresponding update directions are orthogonal with respect to A. While the former property is difficult to enforce for general non-linear functions, the latter condition can be enforced.

Select $\mathbf{x}^{(0)}$, Let $f_0 = f(\mathbf{x}^{(0)})$, $\mathbf{g}_0 = \nabla f(\mathbf{x}^{(0)})$, $\mathbf{d}^{(0)} = -\nabla \mathbf{g}_0$, $k = 1$.
repeat
1. Compute α_k by line search.
2. Set $\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + \alpha_k \mathbf{d}^{(k-1)}$.
3. Evaluate $\mathbf{g}^{(k)} = \nabla f(\mathbf{x}^{(k)})$.
4. $\beta_k = \frac{\left(\mathbf{g}^{(k)}\right)^T \mathbf{g}^{(k)}}{\left(\mathbf{g}^{(k-1)}\right)^T \mathbf{g}^{(k-1)}}$.
5. $\mathbf{d}_k = -\mathbf{g}^{(k)} + \beta_k \mathbf{d}^{(k-1)}$.

$k = k + 1$.
until $\frac{||\mathbf{g}^{(k)}||}{||\mathbf{g}^{(0)}||} < \theta$ OR $k > maxIter$.

Figure 4.55: The conjugate gradient algorithm for optimizing nonlinear convex function f.

It revolutionalized optimization ever since it was invented in 1954.

Variants of the Fletcher-Reeves method use different choices of the parameter β_k. An important variant, proposed by Polak and Ribiere, defines β_k as

$$\beta_k^{PR} = \frac{\left(\mathbf{g}^{(k)}\right)^T \left(\mathbf{g}^{(k)} - \mathbf{g}^{(k-1)}\right)}{\left(\mathbf{g}^{(k)}\right)^T \mathbf{g}^{(k)}}$$

The Fletcher-Reeves method converges if the starting point is sufficiently close to the desired minimum. However, convergence of the Polak-Ribiere method can be guaranteed by choosing

$$\beta_k = max\left\{\beta_k^{PR}, 0\right\}$$

Using this value is equivalent to restarting[34] conjugate gradient if $\beta_k^{PR} < 0$. In practice, the Polak-Ribiere method converges much more quickly than the Fletcher-Reeves method. It is generally required to restart the conjugate gradient method after every n iterations, in order to get back conjugacy, *etc.*

If we choose f to be the strongly convex quadratic E and α_k to be the exact minimizer, this algorithm reduces to the linear conjugate gradient method, Unlike the linear conjugate gradient method, whose convergence properties are well understood and which is known to be optimal (see page 321), nonlinear conjugate gradient methods sometimes show bizarre convergence properties. It has been proved by Al-Baali that if the level set $\mathcal{L} = \left\{\mathbf{x} | f(\mathbf{x}) \leq f(\mathbf{x}^{(0)}\right\}$ of a convex function f is bounded and in some open neighborbood of $\mathcal{L}$, f is Lipshitz continuously differentiable and that the algorithm is implemented with a line search that satisfies the strong Wolfe conditions, with $0 < c_1 < c_2 < 1$, then

$$\lim_{k \to \infty} \inf \ ||\mathbf{g}^{(k)}|| = 0$$

[34]Restarting conjugate gradient means forgetting the past search directions, and start it anew in the direction of steepest descent.

In summary, quasi-Newton methods are robust. But, they require $O(n^2)$ memory space to store the approximate Hessian inverse, and so they are not directly suited for large scale problems. Modificationsof these methods called Limited Memory Quasi-Newton methods use $O(n)$ memory and they are suited for large scale problems. Conjugate gradient methods also work well and are well suited for large scale problems. However they need to be implemented carefully, with a carefully set line search. In some situations block coordinate descent methods (optimizing a selected subset of variables at a time) can be very much better suited than the above methods.

4.6 Algorithms for Constrained Minimization

The general form of constrained convex optimization problem was given in (4.20) and is restated below.

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{array} \tag{4.99}$$

For example, when f is linear and g_i's are polyhedral, the problem is a linear program, which was stated in (4.83) and whose dual was discussed on page 289. Linear programming is a typical example of constraint minimization problem and will form the subject matter for discussion in Section 4.7. As another example, when f is quadratic (of the form $\mathbf{x}^T Q\mathbf{x} + \mathbf{b}^T\mathbf{x}$) and g_i's are polyhedral, the problem is called a quadratic programming problem. A special case of quadratic programming is the least squares problem, which we will take up in details in Section 4.8.

4.6.1 Equality Constrained Minimization

The simpler form of constrained convex optimization is when there is only the equality constrained in problem (4.99) and it turns out to be not much different from the unconstrained case. The equality constrained convex problem can be more explicitly stated as in (4.100).

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & Ax = b \end{array} \tag{4.100}$$

where f is a convex and twice continuously differentiable function and $A \in \Re^{p\times n}$ has rank p. We will assume that the finite primal optimal value p^* is attained by f at some point $\widehat{\mathbf{x}}$. The following fundamental theorem for the equality constrained convex problem (4.100) can be derived using the KKT conditions stated