On Preconditioning the Linearized Conjugate Gradient method for Sparse Nonlinear Optimization

(Without computing the Hessian matrix)

By Ehsan Ganjidoost

Supervisor: Thomas F. Coleman

Research essay for Computational Mathematics

Department of Computational Mathematics

University of Waterloo

Spring 2015

Contents:

Abstract

In many practical nonlinear optimization problems, the objective function has sparsity structure in the corresponding Hessian matrices. Even with sparsity, many optimization methods involve computing, or approximating the Newton step at each iteration. This in turn involves the calculation of the matrix of second derivatives, the Hessian matrix (at each iteration).

Here we propose a method that induces the solution of the Newton step but avoids calculating the Hessian matrix. Instead we compute a sparser approximation used as a preconditioner for a conjugate gradient process; the true Hessian matrix is never computed.

The preconditioner we compute is an approximation of the Hessian matrix using a subset of the nonzero elements of the Hessian matrix. The approximation is obtained based on the knowledge of the sparsity structure, graph coloring techniques, and automatic differentiation.

1. Introduction

One practical methodology in continuous optimization is minimization through the conjugate gradient method with preconditioning. The conjugate gradient method is an efficient iterative method to solve symmetric positive definite linear systems; hence we use it at each iteration of an optimization algorithm to compute an approximation of Newton step.

In large-scale problems, the computation cost becomes crucial. Fortunately, in many large-scale optimization problems, there is sparse structure that we can take advantage of. For example, we can exploit sparsity to just think about solving nonzero elements, which by itself reduce computation cost considerably.

To gain efficiency, instead of the sparse Hessian, we can use a preconditioner for particular problems. Indeed, no single preconditioner is suitable for all types of problems. Here we propose to search for a suitable preconditioner by using a subset of the nonzero elements of the Hessian matrix. The idea is to compute just these elements and avoid computing the entire Hessian matrix.

In this research, we consider optimization problems with sparse Hessian structure, H_S. Given the location of nonzero elements of the Hessian, and using graph coloring methods, we a find the thin matrix V (if possible), which consists of columns vectors, to compute HV by either the finite differencing method or the automatic differentiation method, and then we deduce the desired subset of nonzero elements of H efficiently.

In this way we compute M , an estimation to the Hessian matrix H , without computing the Hessian matrix. Finally, we use M in the preconditioned conjugate gradient algorithm to help find a solution for the nonlinear optimization problem. Here is the proposed scheme of the process:

$$H_S \xrightarrow{compute} \underset{V=\left[d^1,\ldots,d^p\right]}{V} \rightarrow \left\{\begin{array}{c} FD_{method}\left(\nabla f(x),x\right) \\ or \\ AD_{method}\left(f(x),x\right)\end{array}\right\} \xrightarrow{compute} HV \xrightarrow{extract} M \xrightarrow{factorize} \underbrace{M \approx LL^T}_{\substack{Cholesky \\ or \\ \text{modified } Cholesky}}$$

2. Conjugate Gradient for nonlinear optimization

The conjugate gradient method CG, is a suitable tool for solving symmetric positive definite, SPD, linear systems, often, in an iterative method. This method has advantageous compared to other direct methods of solving sparse and large-scale systems. The conjugate gradient method could be applied on a sparse and large nonlinear optimization problem.

Considering the fact that, at each iteration of Newton step we approximated the twice differentiable objective function $f(x)$ by a quadratic function $\min_x \left(\frac{1}{2} x^T Hx + g^T x \right)$ around the optimization point at nth iteration x_n to get the new optimization point x_{n+1}. Thus, $\min_x f(x)$ results in solving $\nabla f(x) = 0$ which leads us to solve the linear system $\nabla^2 f(x) s_N + \nabla f(x) = 0$. Then, by finding Newton step, and updating $x_{n+1} = x_n + s_N$ we will take a step towards the min/max of the quadratic function.

Next, we should approximate the objective function by a quadratic function with the Hessian for x_{n+1} and then solve it for Newton step and so on. Therefore, the main part of the approach in such a nonlinear optimization is to approximate around the earlier optimization point and get the new one by solving for Newton step.

To approximate the objective function, which is often large and sparse, by a quadratic function, we already know the conjugate gradient method which has comparative advantageous over the other methods for solving such problems. The conjugate gradient method itself is a well-studied and well-known method for solving SPD linear systems. Besides, we could compute a preconditioner to approximate the Hessian in order to avoid expensive computing of it. Thus, the focus of this work is on preconditioned CG method applied to Newton systems in the nonlinear optimization context.

2.1 Preliminary Conjugate Gradient

The CG technique can be used in solving large-scale nonlinear optimization problems in the following way. Let us assume our optimization problem is $\min_x \{f(x)\}$ where $f(x)$ is a twice differentiable function, and the gradient $\nabla f(x)$ can be computed. The gradient of $f(x)$ can be identified as the residual in linear systems (i.e. $\nabla f(x) = b - Ax$).

First we take a look at the CG algorithm to find a solution for a SPD linear system which is a finite iterative method. Suppose we have a quadratic function to solve.

$$\min f(x) = \frac{1}{2}x^T Hx - g^T x \ \Rightarrow \ \text{solve}\left(Hx = g\right) \text{ where } H : \nabla^2 f(x) \text{ and } g : \nabla f(x)$$

Newton's method is going to find the roots the derivatives (i.e. $\frac{\partial}{\partial x} f(x) = 0$) . It uses curvature information to take a more direct route for minimizing $f(x)$. By solving a sequence of quadratic problems in Newton system, we can get linear approximation of nonlinear problem. Note that if $f(x)$ is a quadratic function, we can get the exact solution in one step.

Table 2.1: Preliminary conjugate gradient

```
Conjugate gradients for linear Positive Definite systems:
Inputs:
        $f$ : Objective function $f(x)$
        $x = x_0$ : starting value
        $i_{\max}$ : Maximum number of CG iterations
        $\varepsilon < 1$ : CG error tolerance
Output:
        $x^*$, $d$ : Optimization point and the direction of positive (or negative) curvature
--------------------------------------------------------------------------------
$i = 0$
$r = -\nabla f(x)$, $d = r$
$\delta_{new} = r^T r$, $\delta_0 = \delta_{new}$
$tol = \varepsilon^2 \delta_0$
While $i < i_{\max}$ and $\delta > tol$
        $\gamma = Hd$
        If $d^T \gamma < 0$ then
                $return(x, d)$ (it gives direction in negative curvature)
        else
                $\alpha = \frac{\delta_{new}}{d^T \gamma}$
                $x = x + \alpha d$
                $r = -\nabla f(x)$
                $\delta_{old} = \delta_{new}$
                $\delta_{new} = r^T r$
                $\beta = \frac{\delta_{new}}{\delta_{old}}$
                $d = r + \beta d$
                $i = i + 1$
end
```

Notes:

- α is chosen to minimize $f(x)$ along d because:

  $\frac{\partial}{\partial \alpha} f(x+\alpha d) \approx \left[\nabla f(x)\right]^T d + \alpha d^T \left[\nabla^2 f(x)\right] d$ and setting it to 0, gives $\alpha = \frac{-J^T d}{d^T H d}$
- r is residual which is $r = -\nabla f(x)$
- Directions $\left(d_i, d_j\right)$ are H conjugate: $d_i^T H d_j = 0$ where $i \neq j$
- k^{th} iteration produces x; which is minimizer of $f(x)$ on a k dimensional subspace spanned by k conjugate directions generated.
- The recent note implies finite $n-steps$ convergence; but in practice due to finite precision expanding subspace property is lost. In other words, Separate eigenvalues results in poor performance.
- Stop criteria is either when the number of iterations exceeds $i_{\max}$ or when $\left\| r_{(i)} \right\| \leq \varepsilon \left\| r_{(0)} \right\|$
- Fast and inexact line search can be done by a small $i_{\max}$ or approximating the Hessian with its diagonal.
- Why is it important to $return(x,d)$ in some condition?(i.e. $if \;\; d^T \gamma < 0 \Rightarrow return(x,d)$)
  I. It is important to have x since it is an optimization point.
  II. It is important to know d because we will have the direction of down-hill and, as a result, the optimization becomes better and better.
- Computational cost contains:
  I. Matrix-vector product.
  II. Inner product of vectors.
  III. Three vector sums.
- Benefits:
  I. The CG method is beneficial for large-scale problems (otherwise Gaussian and the other methods are better) since it is less sensitive to rounding errors.
  II. Unlike factorization, it does not change the coefficient matrix H.
  III. The CG method sometime approaches the solution very quickly.

2.2 Preconditioned Conjugate Gradient

The performance of the CG method is correlated with the distribution of eigenvalues of the iterative matrix. Using an appropriate preconditioner, a desired clustered distribution of eigenvalues, can be achieved. This results in improvement of convergence.

Our idea is to use a sparse preconditioner M, where M consists of a subset of nonzero elements of the Hessian matrix. Note that $M = M(x)$ and it is calculated directly at each outer iteration before starting to find the Newton step; but our technique avoids computing $H(x)$.

The other thing worthwhile to note is that the preconditioner M should always be positive definite. After finding M, if it is not positive definite, we can factorize it by incomplete Cholesky (also known as modified Cholesky) to make sure positive definiteness of M. Thus, using incomplete Cholesky will give back lower triangular matrix L, which means that $M \approx LL^T$ is definitely positive definite. Considering $Md = r$ in the PCG algorithm, we can solve for d by following steps:

$$\begin{cases} Md = r \\ M \approx LL^T \end{cases} \Rightarrow L \underbrace{L^T d}_{y} \simeq r \Rightarrow y = L \backslash r \Rightarrow L^T d = y \Rightarrow d = L^T \backslash y$$

Table 2.2: Preconditioned conjugate gradient

```
Preconditioned conjugate gradient:
Inputs:
    $f$ : Objective function $f(x)$
    $x = x_0$ : starting value
    $i_{\max}$ : Maximum number of CG iterations
    $\varepsilon < 1$ : CG error tolerance
Output:
    $x^*,\ d$ : optimization point and the direction of positive (or negative) curvature
-----------------------------------------------------------------
$i = 0$
$r = -\nabla f(x)$
Calculating preconditioner $M \approx \nabla^2 f(x)$, and $M \approx LL^T$ using $\begin{cases} M \text{ is PD} & \mathit{cholesky} \\ M \text{ is not PD} & \text{modified } \mathit{cholesky} \end{cases}$
$d \approx M^{-1} r$
(which can be calculated by: $L \overbrace{L^T d}^{y} \simeq r \Rightarrow \begin{cases} y = L \backslash r \\ d = L^T \backslash y \end{cases}$
$\delta_{new} = r^T d$, $\delta_0 = \delta_{new}$
$tol = \varepsilon^2 \delta_0$
While $i < i_{\max}$ and $\delta > tol$
    $\gamma = Hd$ (by FD or the AD methods)
    If $d^T \gamma < 0$ then
        $return(x, d)$ (it gives direction in negative curvature)
    else
```

```
        $\alpha = \frac{\delta_{new}}{d^T \gamma}$
        $x = x + \alpha d$
        $r = r - \alpha \gamma$
        $s \approx M^{-1} r$ (can be calculated by: $L \underbrace{L^T s}_{y} = r \Rightarrow y = L \backslash r \Rightarrow s = L^T \backslash y$)
        (Which avoids factorization again by using $L$ from previous calculation to calculate $s$)
        $\delta_{old} = \delta_{new}$
        $\delta_{new} = r^T s$
        $\beta = \frac{\delta_{new}}{\delta_{old}}$
        $d = s + \beta d$
        $i = i + 1$
end
```

Notes:

- In addition to notes in table 2.2 there are some more notes for the preconditioner CG
- To be efficient, $M \approx LL^T$ must be fast.
- Choosing M is a hard problem by itself for two reasons:
  - I. M should approximate H well.
  - II. $M \approx LL^T$ should be inexpensive.

  Examples of preconditioners are:
  1. Diagonal Matrix M
  2. Banded Approximation
  3. Incomplete Cholesky
- Preconditioner M must be always positive-definite in order to use in the CG method. For this purpose if M was not positive definite, by using modified Cholesky factorization we will get $M \approx LL^T$ and then it is positive definite which is fine.
- $\|x\|_M$ increases at each iteration.
- If $M \approx \nabla^2 f(x)$, then $\nabla^2 f(x) d$ estimates finite difference $\nabla f(x)$ along direction d
- To calculate $s = M^{-1} r$ it does not need to use Cholesky factorization again since we already know the factor L.

3: The Hessian Determination and Graph Coloring

Computation partial derivatives often represent the majority of the computing time in solving optimization problems. The good news is that there are methods that could exploit large-scale sparsity to reduce computation time, such as sparse Finite Differencing (FD) and Automatic Differentiation (AD).

Considering the sparse structure of the Hessian, it is enough to calculate only nonzero elements of matrices in large-scale problems which results in saving on cost. We can further improve computation cost by approximating the Hessian by a subset of its nonzero elements.

First, let us introduce the Jacobian matrix as a straight-forward concept before we move on to the Hessian matrix definition next. To define the Jacobian matrix, suppose we have m functions of x construct $F(x)$ which is an m dimensional vector of functions, $F:\mathbb{R}^n \to \mathbb{R}^m$, over n dimensional vector space, $x \in \mathbb{R}^n$; so the Jacobian matrix defined as derivatives of $F(x)$ with respect to x will result in $J \in \mathbb{R}^{m\times n}$. Making these definitions more clear, we can show $F(x)$, and the Jacobian matrix, and the formula for each element of the matrix as follows:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},\ F = \begin{bmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{bmatrix},\ J = \frac{dF}{dx} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix},\ J_{ij} = \frac{\partial f_i}{\partial x_j}$$

For the Hessian, we can define the Hessian matrix, $H \in \mathbb{R}^{n\times n}$, as the second derivatives of a scalar valued function $f:\mathbb{R}^n \to \mathbb{R}^1$ over the n dimensional vector space $x \in \mathbb{R}^n$. Note that the gradient $\nabla f(x)$ is a $n-by-1$ column vector. The formulations are as follows:

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},\ f = f(x_1,\ldots,x_n),\ \nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix},\ H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix},\ H_{ij} = \frac{\partial^2 f}{\partial x_i x_j}$$

3.1 Finite Differencing (FD) method

To approximate the Jacobian and the Hessian matrices by the FD method along the direction d (which is normalized), we could use Taylor expansion as follows:

$F(x+hd) \approx F(x) + hJ(x)^T d$, which results in: $Jd \approx \dfrac{F(x+hd)-F(x)}{h}$. For the scalar valued function $f(x)$, the Taylor approximation is: $f(x+\alpha d) \approx f(x) + \alpha \nabla f(x)^T d + \frac{1}{2}\alpha^2 d^T Hd$; and if we write the Taylor expansion to the first term approximation for the gradient then we have: $\nabla f(x+\alpha d) \approx \nabla f(x) + \alpha \nabla^2 f(x) d$ which results in: $Hd \approx \dfrac{\nabla f(x+\alpha d) - \nabla f(x)}{\alpha \|d\|}$

Clearly, in the finite differencing method, to approximate the Jacobian and gradient, we need the objective functions $F(x)$ and $f(x)$ respectively; while to estimate the Hessian matrix, the gradient, $\nabla f(x)$, is required. Although the Hessian could be approximated directly from the objective function, but the drawback is poor accuracy.

When we estimate the gradient from the objective function we use Taylor approximation which intrinsically has error, so doing one more approximation on the gradient using Taylor expansion to estimate the Hessian will magnify the error as a result of compounding effect. For this reason, we need to have the gradient precisely in order to estimate the Hessian matrix. Therefore, the gradient is required as an input rather than being approximated, unless we use the automatic differentiation method to get the Hessian matrix directly and precisely from the objective function $f(x)$.

3.2 Automatic Differentiation (AD) method

An alternative method to calculate the Hessian matrix is automatic differentiation which can directly calculate the Hessian by having vector x and the objective function as input. The idea of AD method is built on the chain rule.

Let us assume we want to compute the differentiable function $z = F(x)$ where $F : \mathbb{R}^n \to \mathbb{R}^m$, and m, n are positive integers. We can evaluate $F(x)$ by intermediate variables $y = (y_1, \ldots, y_p)$ which $p \gg m, n$ and obviously each y_k is an output of the atomic function on one or two previous intermediate or original variables i.e. $y_k = y_i \begin{pmatrix} function \\ elements \end{pmatrix} y_j$. In other words, we can write every nonlinear function as a partially ordered sequence of atomic functions. Therefore we can decompose any function $F(x)$ into intermediate variables and functions as:

$$\begin{array}{cc}
solve(y_1): & F_1^E(x, y_1) = 0 \\
solve(y_2): & F_2^E(x, y_1, y_2) = 0 \\
\vdots & \vdots \\
solve(y_p): & F_p^E(x, y_1, y_2, \ldots, y_p) = 0 \\
solve(z): & z - \bar{F}_{p+1}^E(x, y_1, y_2, \ldots, y_p) = 0
\end{array}$$

Viewing $F(x)$ as a partially ordered sequence of atomic functions, we can differentiate it with respect to the original independent variables and the intermediate variables, which results in the $(p+m)-by-(n+p)$ sparse matrix J_{giant}.

$$J_{giant} = \begin{array}{cc} \begin{bmatrix} A & L \\ B & M \end{bmatrix} & \begin{matrix} \}p \\ \}m \end{matrix} \\ \begin{matrix} n & \quad p \end{matrix} & \end{array}, \quad \begin{matrix} AD_{fwd} : J = B - M[L^{-1}A] & \text{and } w_{fwd}(J) \propto n \times w(F) \\ AD_{rev} : J = B - [ML^{-1}]A & \text{and } w_{rev}(J) \propto m \times w(F) \end{matrix}$$

To calculate the gradient, we can use the AD method to get it precisely than approximating it by the FD method. The other fact is that the gradient is a special case of the Jacobian computation when $f : \mathbb{R}^n \to \mathbb{R}^1$ is differentiable and we need to compute the gradient $\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$. So as it can be seen the gradient is like the Jacobian with $m = 1$as a special case of the Jacobian. If we assume the work (floating point operations) for evaluating the objective function is $w(f)$, [1] showed that calculating the gradient in reverse-mode AD_{rev} is $\propto w(f)$ while it cost $n \times w(f)$ in forward mode AD_{fwd}, which takes the same time to calculate the gradient as the FD method.

So far we have considered the gradient computation. Next step is computing second derivatives and the Hessian matrix which is useful in optimization problems i.e. $\min_x f(x)$ where $f : \mathbb{R}^n \to \mathbb{R}^1$ and $f(x)$ is twice continuously differentiable. Finally, we need $\{ f(x),\ \nabla f(x),\ \nabla^2 f(x) \}$ at each iteration x, in the CG iterative method. The goal here is to obtain $\nabla^2 f(x)$ along directions, for given $f(x)$, by AD without computing the $\nabla^2 f(x)$.

First, suppose we want to obtain $\nabla^2 f(x)$. In order to calculate $\nabla^2 f(x)$, one way is to compute the gradient $\nabla f(x)$ from $f(x)$ by the AD method which has discussed briefly; and by the same method we computed the gradient we can find $\nabla^2 f(x)$, since it is the Jacobian of $\nabla f(x)$. However, the forward mode needs less space than the reverse mode while both computing the Hessian in time $n \times w(\nabla f)$.

However, practically, it is slower to compute the Hessian when we have $\nabla f(x)$ rather than when we have $f(x)$. Thus, we assumed the objective function $f(x)$ is given, not $\nabla f(x)$. As a result, the computing cost for the Hessian matrix by the AD method, in general, is:

$$w(\nabla^2 f) \sim n \times w(\nabla f) \sim n \times w(f)$$

The Hessian matrix product can be produced by the AD method directly without requiring the determination of the Hessian matrix itself. The same claim can be made for the Jacobian matrix products. For more clarification, let us suppose for the given differentiable mapping $F:\mathbb{R}^n \to \mathbb{R}^m$, thin matrix $V_{n\times t_V}$, and thin matrix $W_{m\times t_W}$, we could have the products JV and $W^T J$ by the forward mode and the reverse mode of AD respectively. Here are the formulas and the works related to computing each of the products.

$$JV = BV - M[L^{-1}AV],\ w(JV) \sim t_V \times w(F)$$

$$W^T J = W^T B - [W^T M L^{-1}]A,\ w(W^T J) \sim t_w \times w(F)$$

When the number of columns of V (or W) is small compared to the column (or row) dimension, these works substantially cost less than the cost of computing the Jacobian first and then multiplying to get JV and $W^T J$.

The same argument can be used for the Hessian matrix product, Hx. The constrained optimization is one application of product determination, which we work with the reduced gradient and Hessian matrices. We have choices whether to use AD to determine Hx directly and then multiply the result by x^T to get $x^T Hx$ or not. The decision depends on the sparsity of the Hessian matrix. In this research we assumed the Hessian matrix has sparse structure.

While we know sparsity of the Jacobian (or Hessian) matrix how we could find nonzero elements of it. If we could determine a thin matrix V and/or a thin matrix W, and determine JV and/or $W^T J$ by AD forward and reverse mode respectively, then we could extract nonzero elements of the Jacobian matrix from these products. Similarly, if we could determine a thin matrix V, which $V = [d^1, \ldots, d^p]$, and determine HV by the AD method or the FD method, then we could determine the Hessian matrix [3].

In order to calculate HV matrix, the AD method has some benefits over the FD method as follows:

- The AD Method offers more accuracy since it does not have truncation error due to using Taylor expansion in the FD method.

- The FD Method needs knowledge of the structure of the Jacobian or the Hessian while AD can preprocess sparsity pattern.
- The AD Method is much less sensitive to dense rows than the FD method.

To sum up, we already know how to calculate products JV, W^TJ, and HV by either AD or the FD method. The other parts of the puzzle are determining thin matrices V or W for the Jacobian matrix and V for the Hessian matrix, still remain unsolved. Next, we are going to explain how to get those matrices such that they are thin and all nonzero elements could be extracted from those products.

3.3 Graph Coloring

Now we have to find thin matrices in order to recover all nonzero elements of the Jacobian matrix from JV, and W^TJ, and of the Hessian matrix from the HV product. To have a better intuition of thin matrices let us consider the following examples. Suppose $F:\mathbb{R}^n \to \mathbb{R}^n$ is differentiable and the Jacobian of F has the following structures:

Example 1:

$$J = \begin{pmatrix} \alpha_1 & & & & \\ \alpha_2 & \beta_1 & & & \\ \alpha_3 & & \beta_2 & & \\ \alpha_4 & & & \beta_3 & \\ \alpha_5 & & & & \beta_4 \end{pmatrix} \quad V = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \quad JV = \begin{pmatrix} \alpha_1 & 0 \\ \alpha_2 & \beta_1 \\ \alpha_3 & \beta_2 \\ \alpha_4 & \beta_3 \\ \alpha_5 & \beta_4 \end{pmatrix}$$

In this example, by having such V and then computing JV by the AD_{fwd} mode, we can recover all nonzero elements of J. As it can be seen, by the first column of JV, the first column of J can be recovered and by the second column of JV, the rest of diagonal elements of J can be extracted. Note that similar steps can be done for the Hessian matrix.

Example 2:

$$J = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \\ & \beta_1 & & & \\ & & \beta_2 & & \\ & & & \beta_3 & \\ & & & & \beta_4 \end{pmatrix} \quad W = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \end{pmatrix}^T$$

$$W^TJ = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \\ 0 & \beta_1 & \beta_2 & \beta_3 & \beta_4 \end{pmatrix}$$

In the second example, similarly, by having W and then computing W^TJ by the AD_{rev} mode, all nonzero elements of J can be recovered. In this example, because of the dense first row we get poor performance from the AD_{fwd} mode. Similarly, in the previous example the AD_{fwd} mode because of the first dense column has the same situation.

Example 3:

$$J = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \\ \gamma_2 & \beta_1 & & & \\ \gamma_3 & & \beta_2 & & \\ \gamma_4 & & & \beta_3 & \\ \gamma_5 & & & & \beta_4 \end{pmatrix} \quad V = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \quad JV = \begin{pmatrix} \alpha_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \\ \gamma_5 \end{pmatrix} \quad W = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$W^T J = \begin{pmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 & \alpha_5 \\ 0 & \beta_1 & \beta_2 & \beta_3 & \beta_4 \end{pmatrix}$$

In the third example, because of the first dense row, we cannot use the AD_{fwd} mode; because of the first dense column, the AD_{rev} mode could not be a good option as well. However, we can take advantage of both AD_{fwd} and the AD_{rev} modes to extract all nonzero elements of the Jacobian. It is clear that the first column of the Jacobian could be obtained from the AD_{fwd} mode and the rest of nonzero elements could be obtained from AD_{rev} mode.

Now, we know how to get nonzero elements of the Jacobian by having V, W, and obtaining JV, and $W^T J$ from both modes of the AD method as we discussed so far. Similarly, we know getting nonzero elements of the Hessian by having V and obtaining HV by the AD method. However, we still need to determine those thin matrices (i.e. V, W) as missing parts.

Suppose we could partition columns (or rows) of the Jacobian matrix into groups named e_i where $1 \le i \le p$. For example, if we have the Jacobian matrix as below, and x, y, z represent nonzero elements, one possible partition for the Jacobian matrix is $V = (e_1, e_2, e_3)$.

$$J = \begin{pmatrix} x & y & & & & & \\ x & y & z & & & & \\ & y & z & x & & & \\ & & z & x & y & & \\ & & & x & y & z & \\ & & & & y & z & x \\ & & & & & z & x \end{pmatrix}, \quad \begin{matrix} e_1 = (1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1)^T ;\; e_1 \to d^1 \\ e_2 = (0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0)^T ;\; e_2 \to d^2 \\ e_3 = (0 \;\; 0 \;\; 1 \;\; 0 \;\; 0 \;\; 1 \;\; 0)^T ;\; e_3 \to d^3 \end{matrix}$$

Note that each element (index) of the vector e_i corresponds to a column (or row) of the Jacobian (or Hessian) matrix, which presents presence of that columns (or row) in that group.

Therefore, we can formulate partitioning of the Jacobian (or Hessian) matrix into V such that: $V = (e_1, \ldots, e_p)\Big|_{i \in [1,p]} \forall (col_k, col_l) \in e_i;\ (\overrightarrow{col_k} \bullet \overrightarrow{col_l}) = \vec{0}$ for nonzero positions; and if we map $e_1 \to d^1$ for example, we could say $d_i^1 = 1 \Leftrightarrow col(i) \in e_1$ as well.

So far we know that the Jacobian (or Hessian) matrix can be divided into the groups $e_1, \ldots, e_p$ which $d^i \big|_{i=1,\ldots,p}$ can be used to calculate Jd^i for the Jacobian products (or similarly $d^i \big|_{i=1,\ldots,\tilde{p}}$ for Hd^i(s)). Considering the fact that there are different combinations to partition the Jacobian (or Hessian) matrix, the work for calculating JV by the AD_{fwd} method, for example, is $p \times w(F)$. Therefore, our strategy to minimize $w(JV)$ turns into finding the smallest p as possible i.e. the thinnest V (or W) matrix.

Let us define a bi-partition of a matrix J gives a row partition of a subset of rows of J called G_R and a column partition of a subset of columns of J called G_C [2]. If we chose pairs (G_R, G_C) such that $|G_R| + |G_C|$ is the smallest possible partitions, where $|G_R|$ and $|G_C|$ represent the number of groups in G_C and G_R respectively; matrices $V_{n \times |G_C|}$ and $W_{m \times |G_R|}$ could be constructed such that J could be directly determined from $W^T J$, and JV. However, by substitution, the work required to evaluate the nonzero elements of J can be reduced further.

For calculating the Hessian matrix H of a scalar value function $f : \mathbb{R}^n \to \mathbb{R}$ in addition to sparsity we could exploit the symmetric property of the Hessian matrix. In the case of the Hessian matrix has the arrowhead structure, for example, it requires n groups if we ignore symmetry; while it just needs two groups if we take into account the symmetric property.

Example: the Hessian matrix with the arrowhead structure

$$H_{n\times n} = \begin{bmatrix} x & \cdots & x \\ \vdots & \ddots & \\ x & & x \end{bmatrix},\ V_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{bmatrix}_{n\times 2},\ V_2 = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}_{n\times n}, \text{ thus } |V_1| = 2, \text{ and } |V_2| = n$$

We deal with the direct method determination and the substitution method determination as combinatorial problems by exploiting Graph theory to approximate the Jacobian and the Hessian matrices. Let us define the general notation of a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ which has V vertices and E edges; then we color vertices of the graph such that any two adjacent vertices cannot have the same color. In other words, two neighbors in a graph cannot be in a same group; since they have different colors, which is the main idea of graph coloring.

Let $\mathrm{G}_{adj}(\mathrm{H}) = (\mathrm{V}, \mathrm{E})$ be adjacency graph of H, and $|\mathrm{V}| = n$ which v_i corresponds to H_{ii} and if $H_{j,k} \neq 0 \wedge j \neq k \Rightarrow (j,k) \in E$. Assigning P colors to vertices such that for every path in G_{adj} of length of four distinct vertices use at least three colors, which is called the path p-coloring of G_{adj}. Although we can say the general direct method based on the path p-coloring idea exploits symmetry, it does not exploit symmetry to the fullest. We need a way to handle direct asymmetric method when, for example, it applied to a symmetric band matrix. In other words, we are looking for a method which relaxes the restriction that every element of the Hessian should be determined directly.

Let us assume a symmetric matrix which we can find an ordering to determine nonzero elements of the matrix such that they can be solved by using symmetry and previously solved elements. We can see the substitution method as a symmetric tri-diagonal of the Hessian matrix which can be determined by two of the gradient differences with substitution. Substitution method for symmetric matrices is based on the cyclic coloring. Mapping vertices into P colors is a cyclic p-coloring if this mapping uses at least three colors in every cycle of G_{adj}.

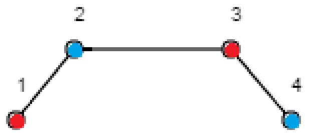


Figure 3.1: Path coloring

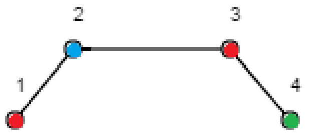


Figure 3.2: Path p-coloring

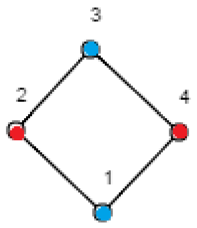


Figure 3.3: Regular Coloring

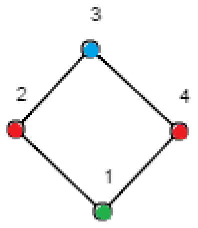


Figure 3.4: Cyclic Coloring

Figure 3.1 and figure 3.2 show path coloring and oath p-coloring which the difference is observable. Also, figure 3.3 shows normal coloring for a cycle of a graph while figure 3.4 shows path cyclic coloring for the same loop of the graph. Since cyclic coloring usually uses fewer colors than path coloring, we need fewer gradient finite differences for the substitution method compared to the direct method. However, the vulnerability to round off error growth due to the substitution method leads us to choose between cost and accuracy.

Overall, there are three different approaches in order to determine the Hessian matrix by finite differencing of the gradient function:

I. The direct method based on coloring of the intersection graph G_I of H, ignoring symmetry.
II. The direct method based on path coloring of the adjacency graph G_{adj} of symmetric H.
III. The Indirect method based on cyclic coloring of the adjacency graph G_{adj} of symmetric H.

Table 3.1 Different coloring approaches determining the Hessian matrix

First Approach	Second Approach	Third Approach
Color $G_I(H) \to 1,\dots,p_I$	Color $G_{adj}(H) \to 1,\dots,p_\alpha$	Color $G_I(H_L) \to 1,\dots,p_\sigma$

Table 3.1 shows how coloring happens in all three approaches. As it can be seen in the third approach, coloring applied on the lower triangular part of the matrix H, which is named here H_L. The other fact is that coloring of the intersection graph of lower triangular, $G_I\left(H_L\right)$ corresponds to a substitution process for nonzero elements of H [4]. Suffice it to say that to implement the cyclic coloring, which is an $NP-Hard$ problem, it is enough to color $G_I\left(H_L\right)$, which has well-known heuristic solutions. Tables 3.2 to 3.4 summarized all three approached supposed $f:\mathbb{R}^n \to \mathbb{R}^1$, $H:\mathbb{R}^{n\times n}$, and $H_{ij} = \dfrac{\partial^2 f}{\partial x_i x_j}$

Table 3.2: First approach to Determinate the Sparse Hessian by FD method

```
Direct, Sparse estimation of $H$:
    -   Color $G_I(H) \to 1,\dots,p_I$
    -   $\forall k: color(k) \to group(e_H) \to d^k$
    -   $for(j=1:p_I)$
            $y^j \simeq \dfrac{\nabla f(x+\alpha d^j) - \nabla f(x)}{\alpha}$
$Y = \left[ y^1 \quad \dots \quad y^{p_I} \right]_{n\times p_I}$
```

Table 3.3: Second approach to Determine the Sparse Hessian by FD method

```
Direct, Sparse estimation of $H$:
    -   Color $G_{adj}(H) \to 1,\dots,p_\alpha$
    -   $\forall k: color(k) \to group(e_H) \to d^k$
    -   $for(j=1:p_\alpha)$
            $y^j \simeq \dfrac{\nabla f(x+\alpha d^j) - \nabla f(x)}{\alpha}$
$Y = \left[ y^1 \quad \dots \quad y^{p_\alpha} \right]_{n\times p_\alpha}$
```

Table 3.4: Third approach to Determinate the Sparse Hessian by FD method

```
InDirect, Sparse estimation of $H$:
    -   Color $G_I(H_L) \to 1,\dots,p_\sigma$
    -   $\forall k: color(k) \to group(e_H) \to d^k$
    -   $for(j=1:p_\sigma)$
            $y^j \simeq \dfrac{\nabla f(x+\alpha d^j) - \nabla f(x)}{\alpha}$
            $Y = \left[ y^1 \quad \dots \quad y^{p_\sigma} \right]_{n\times p_\sigma}$
```

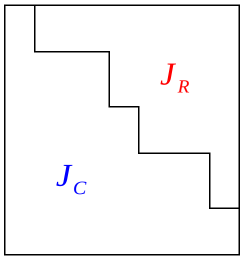
4. Recovering Designated Subset of Nonzero elements

In previous section, in fact, we learned how to do graph coloring and consequently find thin matrices. In order to use graph coloring idea for the purpose of finding thin matrices, and finally extracting nonzero elements of the Jacobian and/or the Hessian matrices, we need to construct related graphs first. In this section, the algorithms are used to construct graphs for extracting nonzero elements of the Jacobian and the Hessian matrices will be explored [5].

4.1 Recovering Nonzero Elements of the Jacobian Matrix

A) Determining J (non-zero elements), by the direct, and 2-sided method.

Let us assume J, after a permutation, partitioned into J_C, and J_R which is called bi-coloring [2]. Therefore, determining J turns to determine J_C, and J_R.



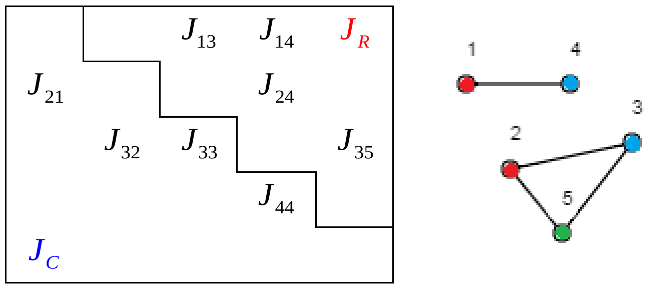
Two sub problems are:

1. Elements in J_C determined by JV ; how to get V ?
2. Elements in J_R determined by $W^T J$; how to get W ?

Algorithm 4.1: The pseudo code for method A

```
To get $V$ :
    1. Construct $G_I(J_C)$ as follows
        ▪ $V = \{1, \dots, n\}$
        ▪ $if\,(k,i) \in nnz$ and $(k,i) \in J_C$
              $for(j = 1:n)$
                  $if\,(k,j) \in nnz \Rightarrow (i,j) \in E(G_I^{J_C})$
    2. Color$\left(G_I^{J_C}\right) \rightarrow V$
To get $W$ :
-   Similarly, construct $G_I^{J_R^T}$, then color it, finally $J_R^T \rightarrow W$
To get $J$ :
-   Extract $J_C$ from $JV$
-   Extract $J_R$ from $W^T J$
```

Example 4.1: We want to determine elements of J_C (similarly J_R) in order to construct $G_I(J_C)$. As it can be followed from the algorithm and the figure 4.1, there should be an edge between column 1 and 4 i.e. $(1,4) \in G_I(J_C)$ because they cannot be in the same group. Similarly, considering other pair of columns we can see $\{(1,4),\ (2,3),\ (2,5),\ (3,5)\} \subseteq E(G_I^{J_C})$. By applying coloring algorithm on the graph, we can get matrix V as a set of column vectors each of which belong to one color. By having JV and V we can get non-zeros of J_C by diagonal solver. Similarly, by having $W^T J$, and W we can get J_C.



$$V = \begin{array}{c} \\ \end{array}\begin{array}{ccc} R & B & G \\ \end{array}$$

$$V = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 4.1: The Jacobian matrix structure, corresponding colored graph, and thin matrix

B) Determining J (non-zero elements), by the substitution, and 2-sided method.

Algorithm 4.2: The pseudo code for method B

```
To get $V$:
    1. Construct $G_I(J_C)$ as follows
        ▪ $V = \{1,\dots,n\}$
        ▪ $if\,(k,i) \in nnz$ and $(k,i) \in J_C$
              $for(j = 1:n)$
                  $if\,(k,j) \in nnz$ and $(k,j) \in J_C \Rightarrow (i,j) \in E(G_I^{J_C})$
    2. Color$\left(G_I^{J_C}\right) \to V$
To get $W$:
-   Similarly, construct $G_I^{J_R^T}$, then color it, finally $J_R^T \to W$
To get $J$:
-   Extract nonzero elements using substitution from $W^T J$, and $JV$
```

Example 4.2: There are two possibilities for any pair of non-zeros we should consider:

1. Both nonzero elements belong to J_C, which correspond to an edge in $G_I^{J_C}$.
2. Some nonzero elements (in one row) belong to J_R, which we have to solve them first and substitute back to solve for J_C.

As it shown in the figure 4.2, by solving J_{34} and J_{35} from J_R; then we can solve J_{33} in J_C. In other words, substitute back from solving J_R to J_C.

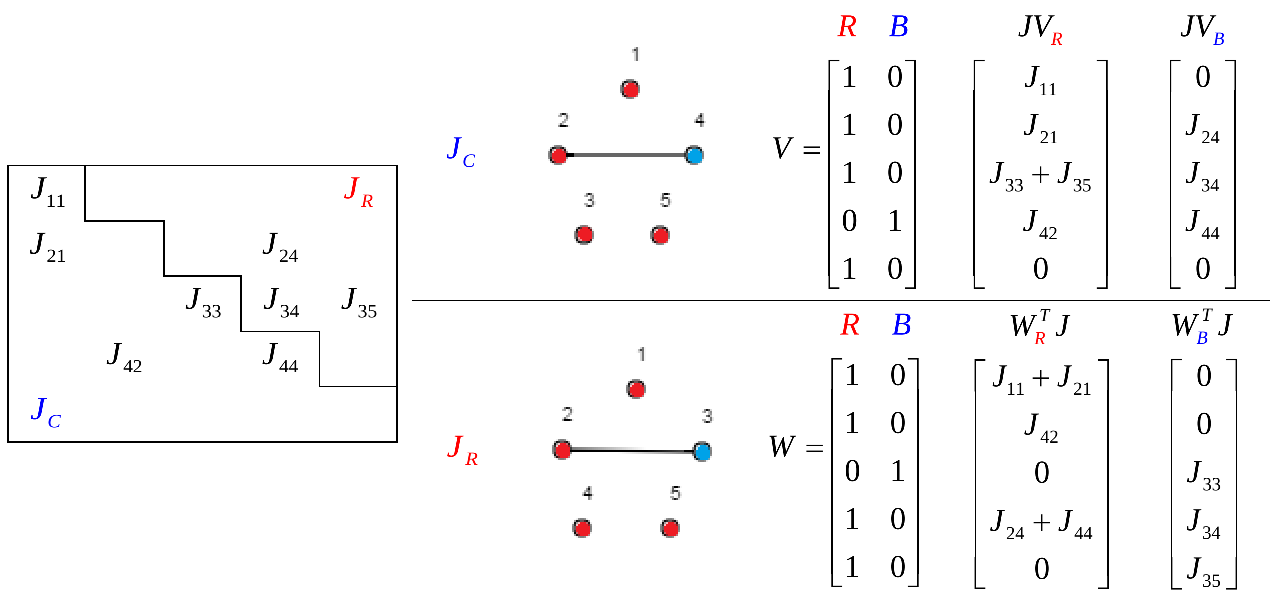


Figure 4.2: The Jacobian matrix structure, J_R and J_C corresponding colored graphs, and thin matrices

4.2 Recovering Designated Subset of Nonzero Elements of the Jacobian Matrix

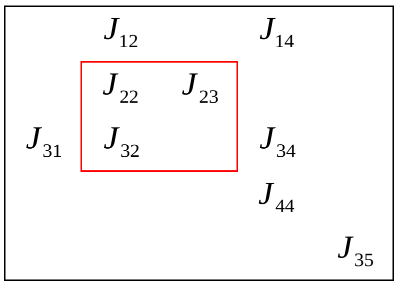
C) Determining designated subset $U \subseteq nnz(J)$, by the direct and 1-sided method.

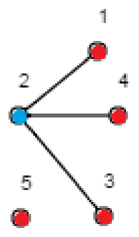
Algorithm 4.3: The pseudo code for method C

```
To get $V$ :
        1.  Construct $G_U$ as follows
                ▪  $V = \{1, \ldots, n\}$
                ▪  $if\,(k,i) \in nnz$ and $(k,i) \in U$
                        $for(j = 1:n)$
                            $if\,(k,j) \in nnz \Rightarrow (i,j) \in E(G_U)$
        2.  Color$(G_U) \to V$
To get $U$ :
-   Extract nonzero elements using diagonal solver.
```

Example 4.3: Here, we should consider all possibilities for any pair of non-zeros:

1. If both nonzero elements do not belong to U, leave it as is and do nothing.
2. If both nonzero elements are in U, there is an edge between them.
3. When one of them is in U and the other is out of U, there is an edge between them.





$$V = \begin{array}{cc} \color{red}{R} & \color{blue}{B} \\ \hline 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{array} \qquad JV_R = \begin{bmatrix} \cancel{J}_{14} \\ J_{23} \\ \cancel{J}_{31} + \cancel{J}_{34} \\ \cancel{J}_{44} \\ \cancel{J}_{35} \end{bmatrix} \qquad JV_B = \begin{bmatrix} \cancel{J}_{12} \\ J_{22} \\ J_{32} \\ 0 \\ 0 \end{bmatrix}$$

Figure 4.3: The Jacobian matrix structure, corresponding colored graph, and thin matrix

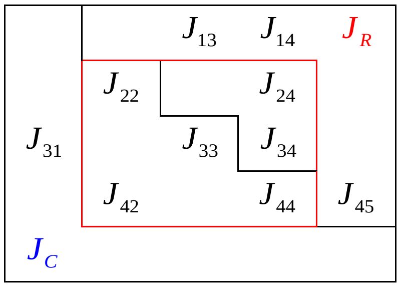
Here we are looking to determine nonzero elements of U which can be solved by diagonal solver. As in the figure 4.3, since J_{31}, J_{34} are non-zeros out of U, they are in the same color, otherwise $(1,4) \in E(G_U)$. Crossed out elements e.g. J_{12} are not interested to calculate.

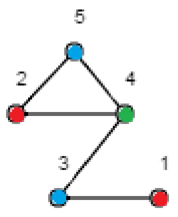
D) Determining designated subset $U \subseteq nnz(J)$, by the direct and 2-sided method.

Algorithm 4.4: The pseudo code for method D

```
To get $V$ :
    1.  Construct $G_U^{J_C}$ as follows
            ▪  $V = \{1, \dots, n\}$
            ▪  $if\,(k,i) \in nnz$ and $(k,i) \in (J_C \cap U)$
                   $for(j = 1:n)$
                       $if\,(k,j) \in nnz \Rightarrow (i,j) \in E(G_U^{J_C})$
    2.  Color$\left(G_U^{J_C}\right) \to V$
To get $W$ :
-   Similarly, construct $G_U^{J_R^T}$, Color $\left(G_U^{J_R^T}\right) \to W$
To get $U$ :
-   Extract nonzero elements of $U$ using diagonal solver from $W^T J$, and $JV$.
```

Example 4.4: As in the figure 4.4, we are solving for non-zero elements of U. For this purpose, every nonzero element out of U could be disregarded.





$$V = \begin{array}{ccc} \color{red}{R} & \color{blue}{B} & \color{green}{G} \\ \hline 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{array} \qquad JV_R = \begin{bmatrix} 0 \\ J_{22} \\ \cancel{J}_{31} \\ J_{42} \\ 0 \end{bmatrix} \qquad JV_B = \begin{bmatrix} \cancel{J}_{13} \\ 0 \\ J_{33} \\ \cancel{J}_{45} \\ 0 \end{bmatrix} \qquad JV_G = \begin{bmatrix} \cancel{J}_{14} \\ J_{24} \\ J_{34} \\ J_{44} \\ 0 \end{bmatrix}$$

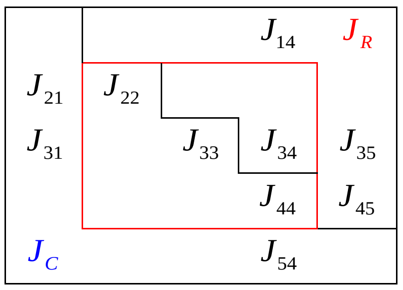
Figure 4.4: The Jacobian matrix structure, corresponding colored graph, and thin matrix

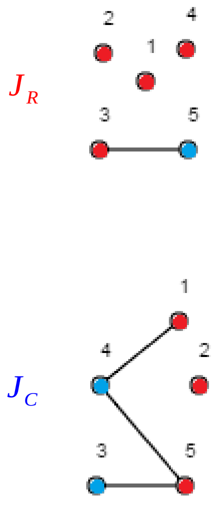
E) Determining designated subset $U \subseteq nnz(J)$, by the substitution, and 2-sided method.

Algorithm 4.5: The pseudo code for method E

```
To get $V$:
    1. Construct $G_U^{J_C}$ as follows
        ▪ $V = \{1,\dots,n\}$
        ▪ $if\,(k,i) \in nnz$ and $(k,i) \in (J_C \cap U)$
              $for(j = 1:n)$
                  $\left\{\begin{array}{c} if\,(k,j) \in nnz \text{ and } (k,j) \in J_C \\ or \\ if\,(k,j) \in nnz \text{ and } (k,j) \in J_R - U \end{array}\right. \Rightarrow (i,j) \in E(G_U^{J_C})$
    2. Color$(G_U^{J_C}) \to V$
To get $W$:
 -  Similarly, construct $G_U^{J_R^T}$, Color $(G_U^{J_R^T}) \to W$
To get $U$:
 -  Extract nonzero elements of $U$ by Substitution from $W^T J$ back to $JV$.
```

Example 4.5: As in the figure 4.5, by solving J_{44} in JV_R we can solve for J_{34} from $W_R^T J$, and then J_{33} in JV_R although in can be solved from $W_R^T J$ directly. Note that to solve for J_{44} we should use $W_R^T J$. If we want to extract it from JV_R we have to determine unnecessary nonzero elements which is, clearly, not efficient.





$$W = \begin{array}{cc} R & B \\ \hline 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{array} \qquad W_R^T J = \begin{bmatrix} \cancel{J_{21}} + \cancel{J_{31}} \\ J_{22} \\ J_{33} \\ \cancel{J_{14}} + J_{34} + J_{44} \\ \cancel{J_{35}} + \cancel{J_{45}} \end{bmatrix} \qquad W_B^T J = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \cancel{J_{54}} \end{bmatrix}$$

$$V = \begin{array}{cc} R & B \\ \hline 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{array} \qquad JV_R = \begin{bmatrix} 0 \\ \cancel{J_{21}} + J_{22} \\ \cancel{J_{31}} + \cancel{J_{35}} \\ \cancel{J_{45}} \\ 0 \end{bmatrix} \qquad JV_R = \begin{bmatrix} \cancel{J_{14}} \\ 0 \\ J_{33} + J_{34} \\ J_{44} \\ \cancel{J_{54}} \end{bmatrix}$$

Figure 4.5: The Jacobian matrix structure, J_R and J_C corresponding colored graphs, and thin matrices

4.3 Recovering Nonzero Elements of the Symmetric Hessian Matrix

In order to recover nonzero elements of the Hessian matrix, we could take advantage of the symmetry property of the matrix. In contrary to the previous algorithms for evaluating the Jacobian matrix, here we do not need to consider coloring in two different sides, columns and rows. In fact, even if we do that we get same thin matrices [5].

Let us assume the Hessian matrix, is permuted symmetrically in order to add as few as possible edges to graph when we construct the graph for it [6], which basically improves the performance as well.

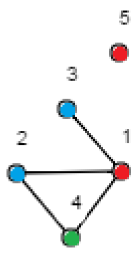
F) Determining nonzero elements of symmetric H, by the direct, and 1-sided method

Algorithm 4.6: The pseudo code for method F

```
To get $V$ :
    1. Permute $H$
    2. Construct $G_A$ as follows
        ▪ $V = \{1, \ldots, n\}$
        ▪ $if\,(k,i) \in nnz$ where $k > i$
              $(k, j) \in nnz \Rightarrow (i, j) \in E(G_A)$
    3. Color$(G_A) \rightarrow V$
To get $H$ :
-   Non-zeros can be directly extracted from $HV \Leftrightarrow V$ corresponds to a path-coloring of
    $G_A^H$
```

Example 4.6: In this example the Hessian matrix is symmetric. Thus, if we solve for the nonzero elements of lower (or upper) triangular of the matrix, we literally have solved for all nonzero elements of the Hessian matrix. Thus, we look at lower triangular part of the Hessian.

$$\begin{array}{|ccccc|} & H_{12} & & H_{14} & H_{15} \\ H_{21} & & H_{23} & H_{24} & \\ & H_{32} & & & \\ H_{41} & H_{42} & & H_{44} & \\ H_{51} & & & & \end{array}$$



$$V = \begin{array}{ccc} R & B & G \\ \hline 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array} \quad \begin{array}{c} HV_R \\ \hline H_{15} \\ H_{21} \\ - \\ H_{41} \\ H_{51} \end{array} \quad \begin{array}{c} HV_B \\ \hline H_{12} \\ H_{23} \\ H_{32} \\ H_{42} \\ - \end{array} \quad \begin{array}{c} HV_G \\ \hline H_{14} \\ H_{24} \\ - \\ H_{44} \\ - \end{array}$$

Figure 4.6: The Hessian matrix structure, corresponding colored graph, and thin matrix

In the figure 4.6, although column 5 is in red group, it could be in another group as well. Dash lines of the Hessian in directions mean the value of those elements is zero. Indeed, we can recover the Hessian just by the elements of our concern.

It is worth it to mention that, when nonzero elements H_{21} extracted from a Hessian product HV_R, the nonzero element H_{12} already solved in HV_B. In this example imagine if we had H_{31} in the structure, it would appear on third row of HV_R and also in HV_B in first row as $H_{12}+H_{13}$ which already solved. Thus, we could pretend that nonzero elements in strict upper triangular are zero (i.e. red elements in the HV). Indeed, by solving lower triangular elements using the symmetry property, we solved for all nonzero elements of the Hessian matrix.

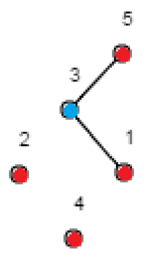
G) Determining nonzero elements of symmetric H, by the substitution, and 1-sided method

Algorithm 4.7: The pseudo code for method G

```
To get $V$:
    1. Permute $H$
    2. Construct $G_A = G_I(H_L)$ as follows
        ▪ $V = \{1, \dots, n\}$
        ▪ $if\ (k,i) \in nnz$ where $k > i$
              $(k, j) \in nnz$ where $k \ge j \Rightarrow (i, j) \in E(G)$
    3. Color$(G_A) \rightarrow V$
To get $H$:
-   Nonzero elements can be extracted by substitution from $HV \Leftrightarrow V$ corresponds to a
    cyclic-coloring
```

Example 4.7: In this example we used the symmetry property of the Hessian. Similar to the previous problem, we look at the lower triangular part of the Hessian. In the figure 4.7 by solving for the lower triangular from bottom to top, we can solve first for H_{53} (i.e. H_{35}) from HV_B and then solving H_{31} when substitute H_{35} back to HV_R. The benefit of this approach is fewer colors are required equal to fewer Hessian products. Note that column 4 could be in Blue.

$$\begin{array}{|ccccc|} & & H_{13} & & \\ & H_{22} & & & \\ H_{31} & & H_{33} & & H_{35} \\ & & & H_{44} & \\ & & H_{53} & & H_{55} \end{array}$$



$$V = \begin{array}{cc} {\color{red}R} & {\color{blue}B} \\ \hline 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 0 \end{array} \qquad \begin{array}{c} HV_{\color{red}R} \\ \hline - \\ H_{22} \\ H_{31}+H_{35} \\ H_{44} \\ H_{55} \end{array} \qquad \begin{array}{c} HV_{\color{blue}B} \\ \hline {\color{red}H_{13}} \\ - \\ H_{33} \\ - \\ H_{53} \end{array}$$

Figure 4.7: The Hessian matrix structure, corresponding colored graph, and thin matrix

4.4 Recovering Designated Subset of Nonzero Elements of the Hessian Matrix

H) Determining designated subset $U \subseteq nnz(H)$, by the direct and 1-sided method

Algorithm 4.8: The pseudo code for method H

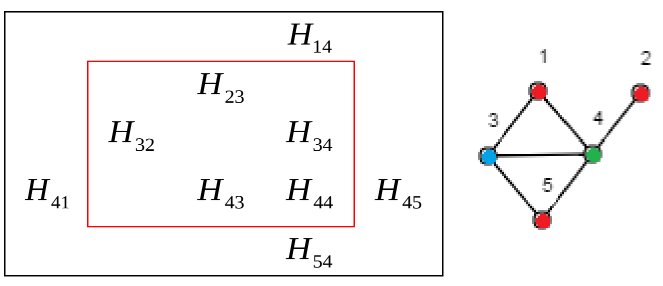
```
To get $V$ :
        1. Construct $G_A$ as follows
                ▪ $V = \{1, \ldots, n\}$
                ▪ $if\,(k,i) \in U$ where $k > i$
                        $if\,(k,j) \in nnz \Rightarrow (i,j) \in E(G_A)$
        2. Color$(G_A) \rightarrow V$
To get $U$ :
-   Non-zeros can be extracted from $HV \Leftrightarrow V$ directly
```

Example 4.8: Considering the symmetry property of H, as in the figure 4.8, by following the algorithm we can color the graph. In this example, the column 2 could be in either blue or red direction.

Like before, we want to solve for the lower triangular elements of the Hessian. Besides, we should consider elements belong to the designated subset on nonzero elements of the Hessian matrix U.

Considering these two limitations, solving the nonzero elements like H_{41}, H_{45}, and consequently H_{14}, H_{54} are not interested any more. Moreover, the upper triangular of the nonzero elements, no matter if they belong to U, is not our mission.

Therefore, in this example by solving for the three nonzero elements H_{32}, H_{43}, and H_{44} we could claim that we determined all nonzero elements of the designated subset of nonzero elements of the Hessian matrix.



$$V = \begin{array}{ccc} R & B & G \\ \hline 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{array} \quad HV_R = \begin{bmatrix} - \\ - \\ H_{32} \\ \cancel{H_{41}} \\ \cancel{H_{45}} \end{bmatrix} \quad HV_B = \begin{bmatrix} - \\ H_{23} \\ - \\ H_{43} \\ - \end{bmatrix} \quad HV_G = \begin{bmatrix} \cancel{H_{14}} \\ - \\ H_{34} \\ H_{44} \\ \cancel{H_{54}} \end{bmatrix}$$

Figure 4.8: The Hessian matrix structure, corresponding colored graph, and thin matrix

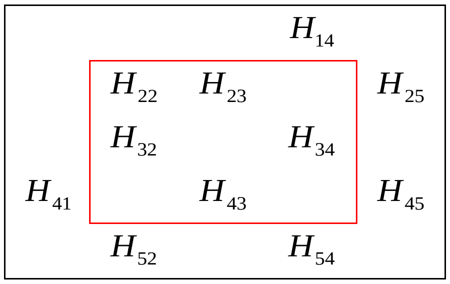
I) Determining designated subset $U \subseteq nnz(H)$, by 1-sided substitution method

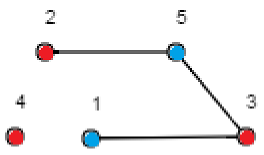
Algorithm 4.9: The pseudo code for method I

```
To get $V$ :
        3. Construct $G_A$ as follows
                ▪ $V = \{1, \ldots, n\}$
                ▪ $if\,(k,i) \in U$ where $k > i$
                        $\left\{\begin{array}{c} (k,j) \in nnz \text{ where } k \ge j \\ or \\ (k,j) \in (nnz - U) \end{array}\right. \Rightarrow (i,j) \in E(G)$
        4. Color$(G_A) \rightarrow V$
To get $U$ :
-   Non-zeros can be extracted from $HV \Leftrightarrow V$ substitution.
```

Example 4.9: Similar to the substitution method to determine all nonzero elements of the Hessian matrix we approach to find nonzero elements which are belong to U as a restriction. In the figure 4.9 using symmetry property of H, leads us to finding nonzero elements of the lower triangular of the Hessian.

Note that column 4 could be in either red or the blue color. The other thing we may noticed here is that we are not interested on solving for blue direction since they are not in the subset U. In order to solve for concerned nonzero elements, we move from bottom to the top i.e. first solve for the H_{43} , then substitute back in third row and get H_{32}, and substitute it in the second row and get H_{22}. Consider that we used symmetry property several times in this process of solving for nonzero elements of U by substitution.





$$V = \begin{array}{cc} B & R \\ \left[\begin{matrix} 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{matrix}\right] \end{array} \quad \begin{array}{c} HV_B \\ \left[\begin{matrix} - \\ \cancel{H}_{25} \\ - \\ \cancel{H}_{41} + \cancel{H}_{45} \\ - \end{matrix}\right] \end{array} \quad \begin{array}{c} HV_R \\ \left[\begin{matrix} - \\ H_{22} + H_{23} \\ H_{32} + H_{34} \\ H_{43} \\ - \end{matrix}\right] \end{array}$$

Figure 4.9: The Hessian matrix structure, corresponding colored graph, and thin matrix

5. Experimental Results

In this section we are going to use a quadratic problem, which approximate the nonlinear optimization problems. Then, we are going to compute a preconditioner for solving linear system in order to find the Newton step with the preconditioned conjugate gradient method.

To experiment with the idea we made up quadratic problems such that it has the Hessian matrix, H, and the Jacobian, g, for the problem size n. For the Hessian matrix it should be positive definite diagonal matrix which is zero free diagonal in order to use in the CG algorithm. The other things about the Hessian matrix we can take in consideration are the sparsity structure, the symmetry, the density of nonzero elements of the sparse matrix, and the condition number for the matrix.

After generating an $n-by-n$ hessian matrix H with above properties, next we should generate g, a column vector of size n, which completes constructing of a quadratic problem. All these matrices and vectors are generated randomly by MATLAB function. We should note that although both the Hessian and the Jacobian are generated, we just have the quadratic problem $f(x)$, and the gradient $\nabla f(x)$. In other words, we pretend we do not have H, and g. Therefore we can say that we just have:

$$\text{The problem is: } f(x) = \frac{1}{2} x^T H x + g^T x$$

$$\text{The gradient is: } \frac{\partial}{\partial x} f(x) = Hx + g$$

The next step after constructing the quadratic problem is finding the preconditioner. There are general purpose preconditioners such as symmetric successive over-relaxation, incomplete Cholesky, and banded preconditioner. We are using the banded in this experiment.

From previous sections, we understood the process of getting the Hessian matrix indirectly without actually computing it which reduces computation cost effectively. In brief, here is what we have discussed:

$$H_S \rightarrow \underset{V=\left[d^1,\ldots,d^p\right]}{V} \rightarrow \begin{Bmatrix} FD_{method}\left(\nabla f(x), x\right) \\ AD_{method}\left(f(x), x\right) \end{Bmatrix} \rightarrow HV \rightarrow M \rightarrow \underbrace{M \approx LL^T}_{\substack{Cholesky \\ or \\ \text{modified } Cholesky}}$$

In order to get the preconditioner M, we need to know the second derivatives in the directions, V, by the FD or AD methods which gives back HV. For the FD method we need the gradient and the input vector of variables. For the AD method we need the objective function and the input vector of variables.

By having the structure of the Hessian H_S, and the structure of the subset of nonzero elements of the Hessian matrix U we could get coloring information and construct V. After computing HV by both methods FD and AD, we computed the preconditioner for each of them. Now everything was ready to use preconditioned CG algorithm by getting an input random vector x_0. In this experience we tried different size of problem, several density of sparse structure, and varying on banded preconditioner by choosing diagonal or tri-diagonal elements as subset of the Hessian matrix.

The other experiment was measuring the cost of computing the Hessian completely and then finding the Hessian product Hd. We measured the computation cost for calculating the Hessian product Hd by the AD method (i.e. computed Hd instead of $H \times d$) as well. For this part we used $ADmath$ toolbox.

We observed that, although the computing preconditioner is notable, it is way cheaper than computing the whole Hessian for each iteration. Also should note that the PCG algorithm when the problem size becomes bigger performs better than the CG method. The other fact is, even if the computation cost for computing the preconditioner is not negligible, it just computed once.

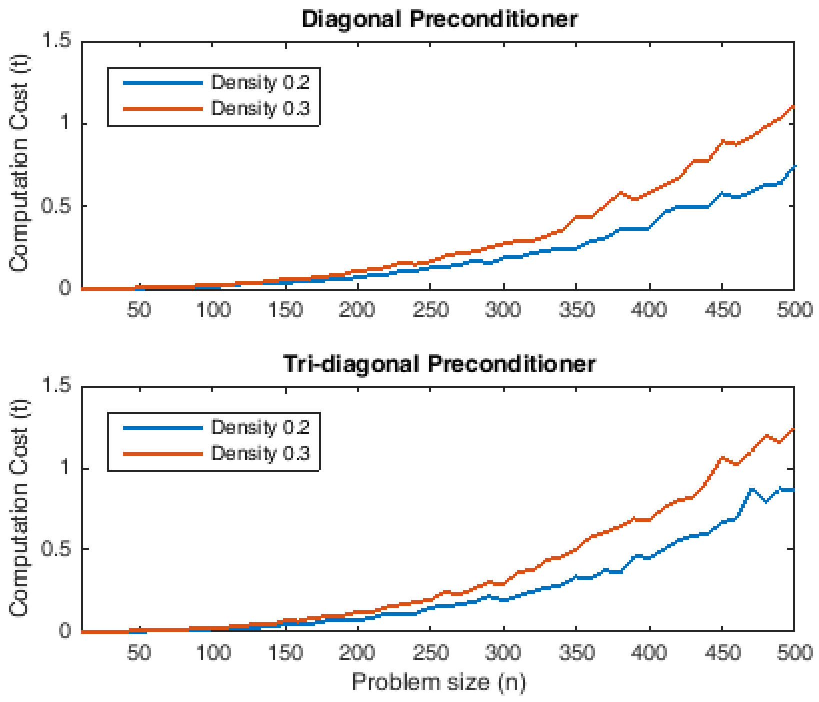


Figure 5.1: Cost of computing the preconditioner for subset of non-zeros U and different density of H

The figure 5.1 shows that the cost of computing the preconditioner increases by density of the Hessian matrix. It means if the given Hessian structure is less sparse, the costs we will pay increase. This will effect both situations no matter if we want diagonal or tri-diagonal nonzero elements of the Hessian. However, the cost of computing the preconditioner is higher for the tri-diagonal subset on non-zeros. The reason is because the cost of coloring, the cost of finding the Hessian products HV, and the cost of extracting nonzero elements will increase by both density and type of preconditioners.

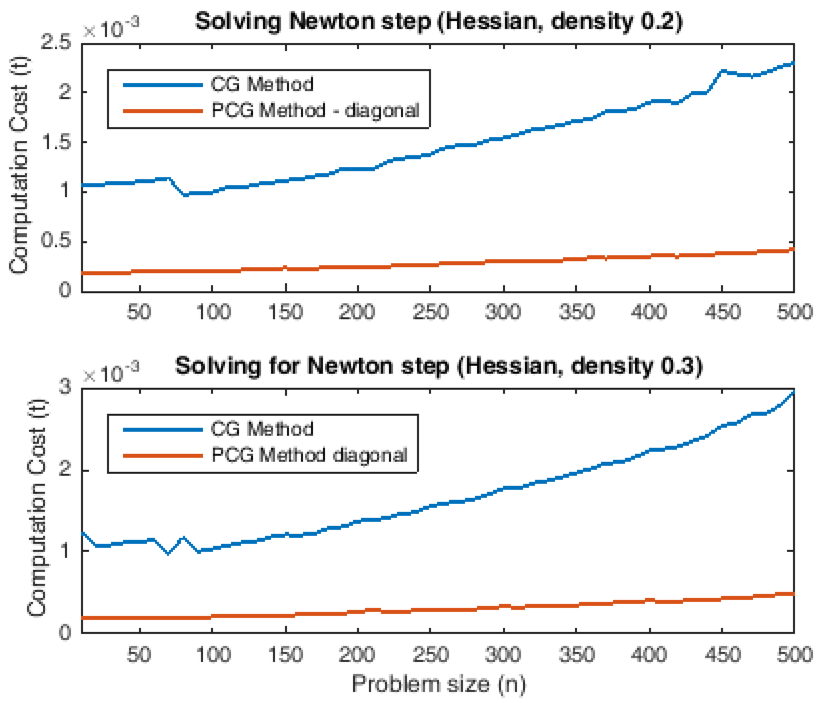


Figure 5.2: Performance of CG and PCG with diagonal preconditioner with different H

In the figure 5.2, the performance of solving the Newton step is compared for both CG and the PCG methods. In this experiment, we used diagonal subset of nonzero elements of the Hessian as the preconditioner. The performance of the PCG method is way better than the CG algorithm. Furthermore, when the problem size is small, the difference for performances is negligible, but when the problem size increase, PCG shows its comparative advantageous.

In the figure 5.3, same experiment repeated for tri-diagonal subset of nonzero elements of the Hessian matrix. The PCG method performs well even in denser Hessian structure. Although the cost of the algorithm increase by an increase in the size of problem for both methods, we experience more increase in the CG method compared to the PCG method.

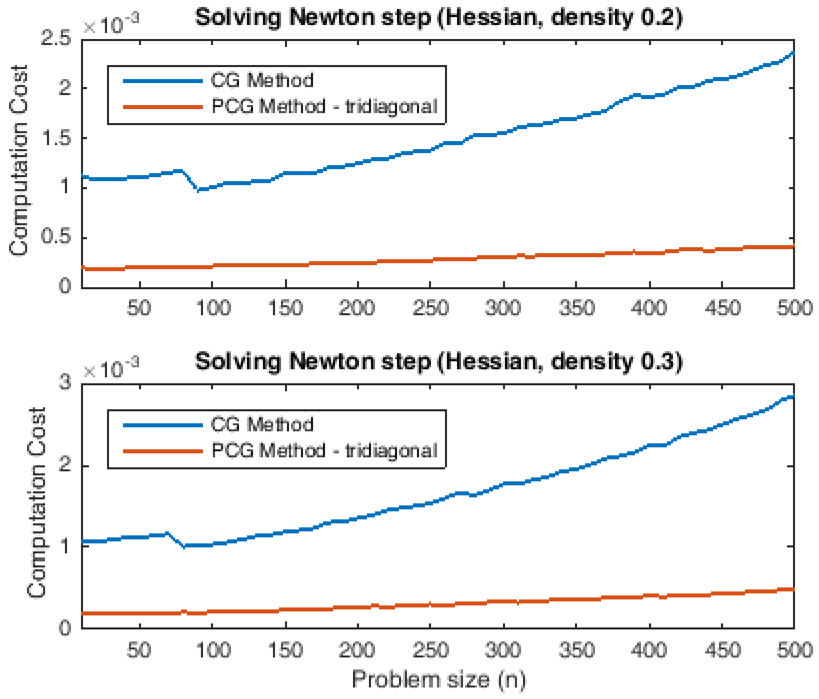


Figure 5.3: Performance of CG and PCG with tri-diagonal preconditioner with different H

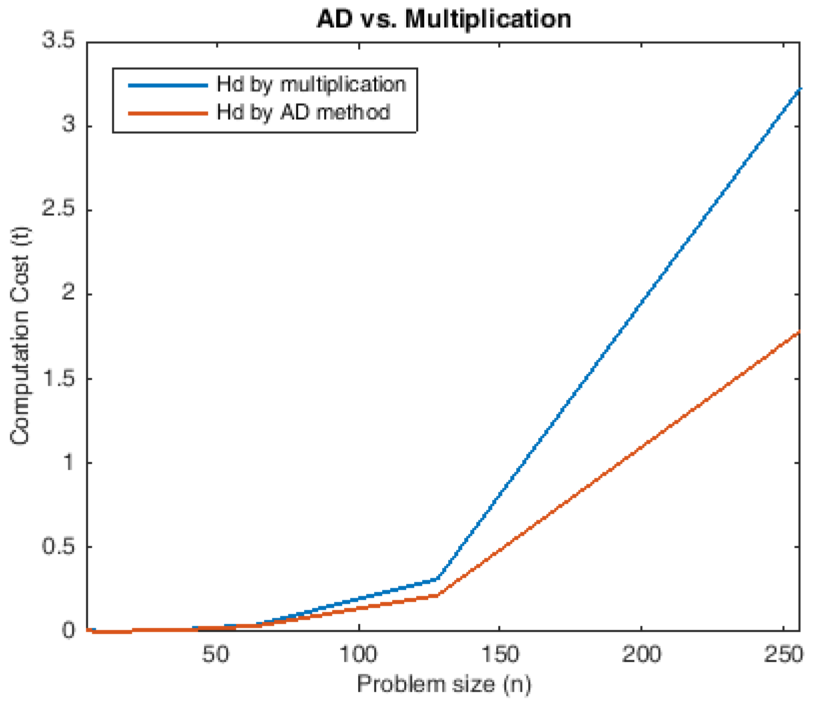


Figure 5.4: Cost of computing HV by direct and the AD method

In the figure 5.4, as we expected the cost of computing the Hessian and the calculating the Hessian products HV (i.e. $H \times V$) is more costly than calculating them by the AD method.

6. Conclusion

In many practical optimization problems, as well as nonlinear system problems, the objective function has the sparse structure in their Jacobian and/or Hessian matrices which can be used to great advantage when computing the Newton steps. The Newton steps for nonlinear optimization problems typically involve two major steps;

First step: evaluation of $f(x):\mathbb{R}^n \to \mathbb{R}^1,\ \nabla f(x) \in \mathbb{R}^n,\ \text{and}\ H = \nabla^2 f(x) \in \mathbb{R}^{n\times n}$
Second step: solving $\nabla^2 f(x) s_N = -\nabla f(x)$.

For the first step, since evaluation of the Hessian matrix H is often the most expensive part of evaluation, we avoid computing it. Indeed, we use an estimation of the Hessian matrix for the early stage of the Newton steps with lower computation cost. This approximation is deduced based on sparsity structure of the Hessian in addition to graph coloring techniques and the automatic differentiation method.

The second step infers $\min f(x) \Leftrightarrow \text{solve } \nabla f(x) = 0$; so, we can use the preconditioned conjugate gradient method for linear semi-positive-definite (SPD) systems to solve $Hs_N = -\nabla f(x)$ and update the minimizer. Thus, when the Newton step is not positive-definite, it is not allowed to go through negative curvature. Therefore, without computing the complete Hessian matrix, we could find the solution for the Newton steps for nonlinear optimization problems using the preconditioned conjugate gradient method. The hierarchy of the idea is:

$$\min f(x)$$
$$\Downarrow$$
$$\text{nonlinear solver } \nabla f(x) = 0$$
$$\Downarrow$$
$$\begin{pmatrix} \nabla^2 f(x) s_N + \nabla f(x) = 0 \\ x^* = x^* + s \end{pmatrix} \Leftrightarrow \underbrace{Hx + g = 0}_{CG \min\left(\frac{1}{2}x^T Hx + g^T x\right)}$$

Here, in this research, we showed that using the automatic differentiation combined with the graph coloring techniques can improve the computation cost by approximating the Hessian matrix. This improvement is proportional to the number of columns of the thin matrix. The preconditioned conjugate gradient method presents better performance when the problem size grows. It solves linear system for the Newton step in fewer iterations compared to the conjugate gradient method itself. All works we have discussed for Newton step is an inner part of the iterations for solving the nonlinear optimization problem.

References

[1] T.F. Coleman and G.F. Jonsonn, *The Efficient computation of structured gradients using automatic differentiation*, SIAM J. Sci. Comput., vol. 20, 1999, 1430-1437.

[2] T.F. Coleman, A. Verma, *Structured and Efficient Jacobian Calculation,* in Computational Diffrentiation: Techniques, application,and tool, M. Berz, C. Bischof, G. Corliss and A. Griewank (eds), SIAM, Philadelphia, (1996), pp. 149–159.

[3] T.F. Coleman and J.J. Moré, *Estimation of sparse Hessian matrices and graph coloring problems*, Math. Programming, Vol. 28 (1984), pp. 243–270.

[4] T.F. Coleman and J.Y. Cai, *The cyclic coloring problem and estimation of sparse Hessian matrices*, SIAM J. Algebraic Discrete Methods, Vol. 7 (1986), pp. 221–235.

[5] W. Xu , T.F. Coleman, *Efficient (Partial) Determination of Derivative Matrices via Automatic Differentiation*, SIAM J. Sci. Comput., Vol. 35(3) (2013), pp. 1398-1416.

[6] T.F. Coleman, A. Verma, *The efficient computation of sparse Jacobian matrices using automatic differentiation*, SIAM J. Sci. Comput., Vol. 19 (1998), pp. 1210–1233.

[7] J. Nocedal, S.J. Wright, *Numerical Optimization – 2nd Edition*, Springer Series in Operations Research, T.V. Mikosch, S.M. Robinson, S.I. Resnick, Springer Science + Business Media (2006)

[8] J.R. Shewchuk, *An Introduction to Conjugate Gradient Method Without the Agonizing Pain*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA (1994)

[9] MATLAB 2014b, www.mathworks.com, 2015

[10] ADMAT 2.0: Automatic Differentiation Toolbox, www.cayugaresearch.com, 2015

Appendix