From: http://trond.hjorteland.com/thesis/node1.html

Optimization

As discussed briefly in Section 4.1, the problem we are facing when searching for stationary values of the action given in equation (4.1) is optimization; i.e. trying to locate the stationary values. In literature treating the AVP, only minimum and saddle points have been searched for, indicating that there are no maximum points for the action. I will therefore only discuss minimum (both global and local) and saddle points in this thesis. The choice of optimizing methods is a reflection of this; with the exception of the Newton-Raphson method, all of the methods search for stationary values in a *descent direction*. This automatically excludes any possible maximum points. The Newton-Raphson method is capable of locating any stationary point, but, as we will see in the next chapter, there will solely be minimum and saddle points that are solutions to the action.

A general description of several optimizing methods will be given in this section, while in the next chapter, they will be evaluated by applying them to a system galaxies. The methods will be judged on the ground of effectiveness and ability to locate different solutions. The reason for doing this rather detailed testing is simply that there, as of today, is no method that is known to be better than any other for general use, and that it is hard to say from just looking at the problem which method is best suitable. A description of the methods presented can be found in textbooks like Press,Walsh,DS,GMW,Bert.

In the general treatment given in the remaining parts of this chapter, the function for which we try to find the minima are referred to as $f(\mathbf{x})$, where $\mathbf{x}$ are the unknown variables. An unconstrained optimization procedure starts by choosing a starting point; i.e. an initial guess for the values of the unknown parameters in $f(\mathbf{x})$, $\mathbf{x}_0$. A substantial amount of computing time can be saved, and the possibility of actually finding minima increased, by choosing $\mathbf{x}_0$ with some care. This means in practice using whatever information available on the behavior of $f(\mathbf{x})$, so that our initial guess is not too far from the stationary point(s).

Once the initial point is chosen, we must make two decisions before the next point can be generated: (i) we must first pick a direction along which the next point is to be chosen, and (ii) we must decide on a step size to be taken in that chosen direction. We then have the following iterative picture

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k \qquad k = 0, 1, \ldots, \tag{4.5}$$

where $\mathbf{d}_i$ is the direction and $|\lambda_i \mathbf{d}_i|$ is the step size. The different optimization methods presented differ in the choice of $\mathbf{d}_i$ and λ_i. Roughly speaking, we can classify these methods into three categories:

(1) The methods using only the functional values, calle
5plus2.5minus10plus4minus65plus2.5minus(2) The methods making use of first-order derivatives.
(3) The methods which also requires knowledge of secc

The two last categories are often also classified as gradient methods. In choosing a method, accuracy and especially computing time are of the essence. Category (1) will not be considered here; it is generally accepted that one should make use of first-order derivatives if they are available and not too elaborate to calculate. Category (3) will generally generate points that are sufficiently close to the minimum point in the least number of steps, but that does not necessarily mean that it is the most efficient. The computational costs of computing and handling the second derivatives can be so substantial, that the evaluation of the first derivative alone at several points would take less computing time. Methods of category (2) would then be a preferable choice.

For illustrative purposes, a quadratic function will be used in the presentation of these optimizing methods, much due to its simplicity, but mainly because several of these methods where originally designed to solve problems equivalent to minimizing a quadratic function. The *n*-dimensional quadratic functions used here is on the form

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \cdot \mathbf{Q} \cdot \mathbf{x} + \mathbf{b} \cdot \mathbf{x} + c, \tag{4.6}$$

where $\mathbf{Q}$ is a positive definite matrix, $\mathbf{x}$ and $\mathbf{b}$ are vectors, and *c* is a scalar constant. The illustrations presented are 2-dimensional representation of the quadratic, where the contours represent where the function values are the same.

Let us now take a closer look at some of these optimizing methods, starting with category (2).

Method of Steepest Descent

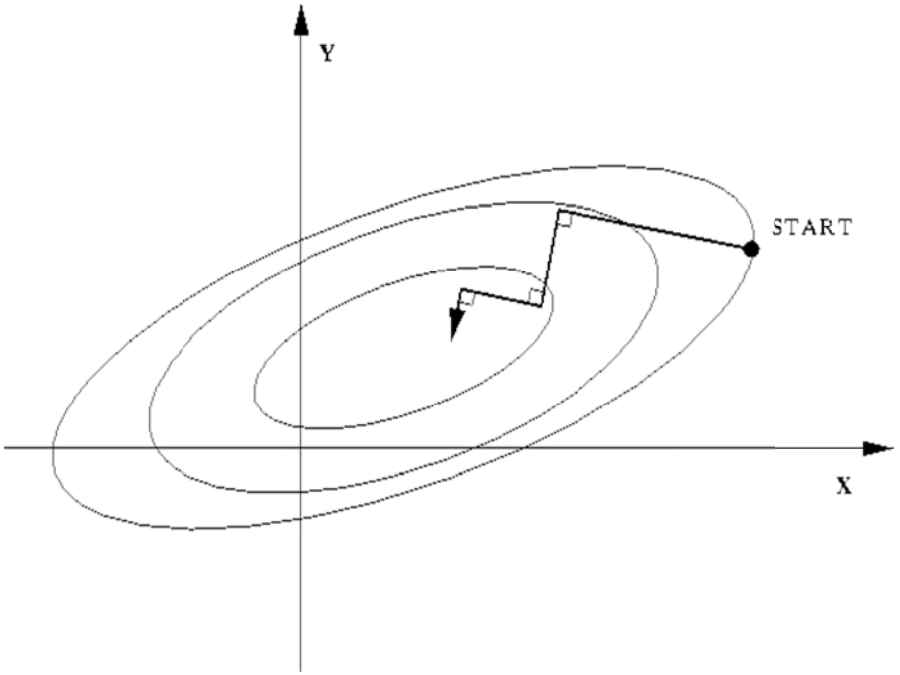

Figure 4.1: The method of Steepest Descent approaches the minimum in a zig-zag manner, where the new search direction is orthogonal to the previous.

The method of *Steepest Descent* is the simplest of the gradient methods. The choice of direction is where *f* decreases most quickly, which is in the direction opposite to $\nabla f(\mathbf{x}_i)$. The search starts at an arbitrary point $\mathbf{x}_0$ and then slide down the gradient, until we are close enough to the solution. In other words, the iterative procedure is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \nabla f(\mathbf{x}_k) = \mathbf{x}_k - \lambda_k \mathbf{g}(\mathbf{x}_k) , \qquad (4.7)$$

where $\mathbf{g}(\mathbf{x}_k)$ is the gradient at one given point. Now, the question is, how big should the step taken in that direction be; i.e. what is the value of λ_k? Obviously, we want to move to the point where the function *f* takes on a minimum value, which is where the directional derivative is zero. The directional derivative is given by

$$\frac{d}{d\lambda_k} f(\mathbf{x}_{k+1}) = \nabla f(\mathbf{x}_{k+1})^T \cdot \frac{d}{d\lambda_k} \mathbf{x}_{k+1} = -\nabla f(\mathbf{x}_{k+1})^T \cdot \mathbf{g}(\mathbf{x}_k) . \qquad (4.8)$$

Setting this expression to zero, we see that λ_k should be chosen so that $\nabla f(\mathbf{x}_{k+1})$ and $\mathbf{g}(\mathbf{x}_k)$ are orthogonal. The next step is then taken in the direction of the negative gradient at this new point and we get a zig-zag pattern as illustrated in Figure (4.1). This iteration continues until the extremum has been determined within a chosen accuracy ϵ.

What we have here is actually a minimization problem along a line, where the line is given by (4.7) for different values of λ_k. This is usually solved by doing a *linear search* (also referred to as a *line search*); i.e. searching for a minimum point along a line. A description of some of these methods is given by Press,Walsh,Shew. Hence, the search for a minimum of $f(\mathbf{x})$ is reduced to a sequence of linear searches. This implementation of the Steepest Descent method are often referred to as the *optimal gradient method*.

Alternatively, one can start out with a chosen value for λ_k, which, if necessary, will be modified during the iterations, making sure that the function decreases at each iteration. This is of course a lot simpler, and often works better, in cases where the calculation using a linear search is laborious. It will take many more iterations to reach the minimum, but each iteration will take much less time than by using a linear search.

The resultant iterative algorithm with a linear search is given in Algorithm 4.1.

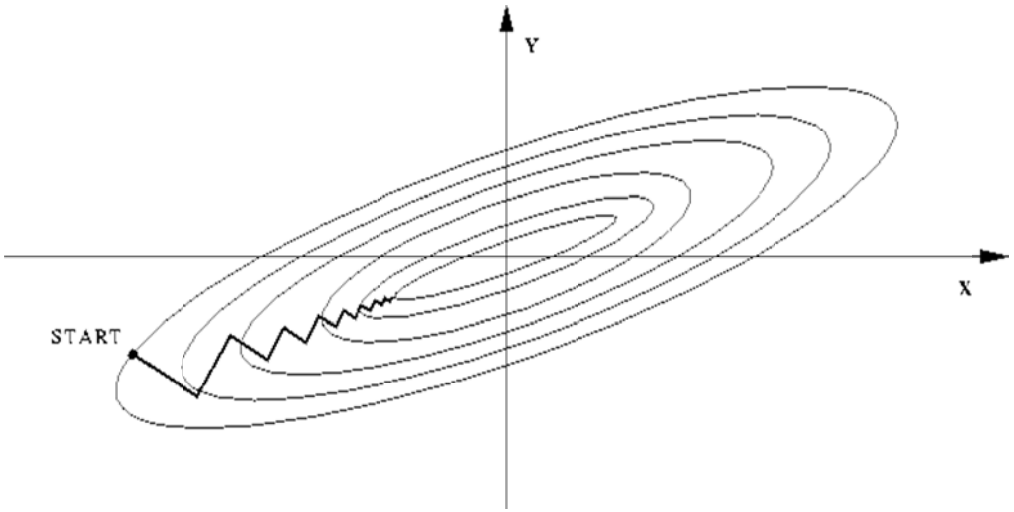

Figure 4.2: The convergence of the method of Steepest Descent. The step size gets smaller and smaller, crossing and recrossing the valley (shown as contour lines), as it approaches the minimum.

Initializing: $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$, $\mathbf{d}_0 = -\mathbf{g}_0$ 5plus2.5minus10plus4minus65plus2.5minus

ΔDetermine the step length λ_k: $\min_{\lambda_k > 0} f(\mathbf{x}_k + \lambda_k \mathbf{d}_k)$. Calculate the new point: $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$. Calculate the gradient: $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$. Set direction of search: $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1}$.

Algorithm 4.1: Method of Steepest Descent

As seen, the method of Steepest Descent is simple, easy to apply, and each iteration is fast. It also very stable; if the minimum points exist, the method is guaranteed to locate them after at least an infinite number of iterations. But, even with all these positive characteristics, the method has one very important drawback; it generally has slow convergence. For badly scaled systems; i.e. if the eigenvalues of the Hessian matrix at the solution point are different by several orders of magnitude, the method could end up spending an infinite number of iterations before locating a minimum point. It starts out with a reasonable convergence, but the

progress gets slower and slower as the minimum is approached Shew. This is illustrated in Figure 4.2, in the case of a quadratic function with a long, narrow valley. The method may converge fast for such badly scaled systems, but is then very much dependent on a good choice of starting point. In other words, the Steepest Descent method can be used where one has an indication of where the minimum is, but is generally considered to be a poor choice for any optimization problem. It is mostly only used in conjunction with other optimizing methods.

Method of Conjugate Gradients

As seen in the previous subsection, the reason why the method of Steepest Descent converges slowly is that it has to take a right angle turn after each step, and consequently search in the same direction as earlier steps (see Figure 4.1). The method of *Conjugate Gradients* is an attempt to mend this problem by ``learning" from experience.

``Conjugacy" means that two unequal vectors, $\mathbf{d}_i$ and $\mathbf{d}_j$, are orthogonal with respect to any symmetric positive definite matrix, for example **Q** in (4.6); i.e.

$$\mathbf{d}_i^T \cdot \mathbf{Q} \cdot \mathbf{d}_j = 0. \qquad (4.9)$$

This can be looked upon as a generalization of orthogonality, for which **Q** is the unity matrix. The idea is to let each search direction $\mathbf{d}_i$ be dependent on all the other directions searched to locate the minimum of $f(\mathbf{x})$ through equation (4.9). A set of such search directions is referred to as a Q-orthogonal, or conjugate, set, and it will take a positive definite n-dimensional quadratic function as given in (4.6) to its minimum point in, at most, n exact linear searches. This method is often referred to as *Conjugate Directions*, and a short description follows.

The best way to visualize the working of Conjugate Directions is by comparing the space we are working in with a ``stretched" space. An example of this ``stretching" of space is illustrated in Figure 4.3: (a) demonstrates the shape of the contours of a quadratic function in real space, which are elliptical (for $\mathbf{b} \neq \mathbf{0}$). Any pair of vectors that appear perpendicular in this space, would be orthogonal. (b) show the same drawing in a space that are stretched along the eigenvector axes so that the elliptical contours from (a) become circular. Any pair of vectors that appear to be perpendicular in this space, is in fact $\mathbf{Q}$-orthogonal. The search for a minimum of the quadratic functions starts at $\mathbf{x}_0$ in Figure 4.3(a), and takes a step in the direction $\mathbf{d}_0$ and stops at the point $\mathbf{x}_1$. This is a minimum point

along that direction, determined the same way as for the Steepest Decent method in the previous section: the minimum along a line is where the directional derivative is zero (equation (4.8)). The essential difference between the Steepest Descent and the Conjugate Directions lies in the choice of the next search direction from this minimum point. While the Steepest Descent method would search in the direction $\mathbf{r}_1$ in Figure 4.3(a), the Conjugate Direction method would chose $\mathbf{d}_1$. How come Conjugate Directions manage to search in the direction that leads us straight to the solution $\mathbf{x}$? The answer is found in Figure 4.3(b): In this stretched space, the direction $\mathbf{d}_0$ appears to be a tangent to the now circular contours at the point $\mathbf{x}_1$. Since the next search direction $\mathbf{d}_1$ is constrained to be $\mathbf{Q}$-orthogonal to the previous, they will appear perpendicular in this modified space. Hence, $\mathbf{d}_1$ will take us directly to the minimum point of the quadratic function $f(\mathbf{x})$.

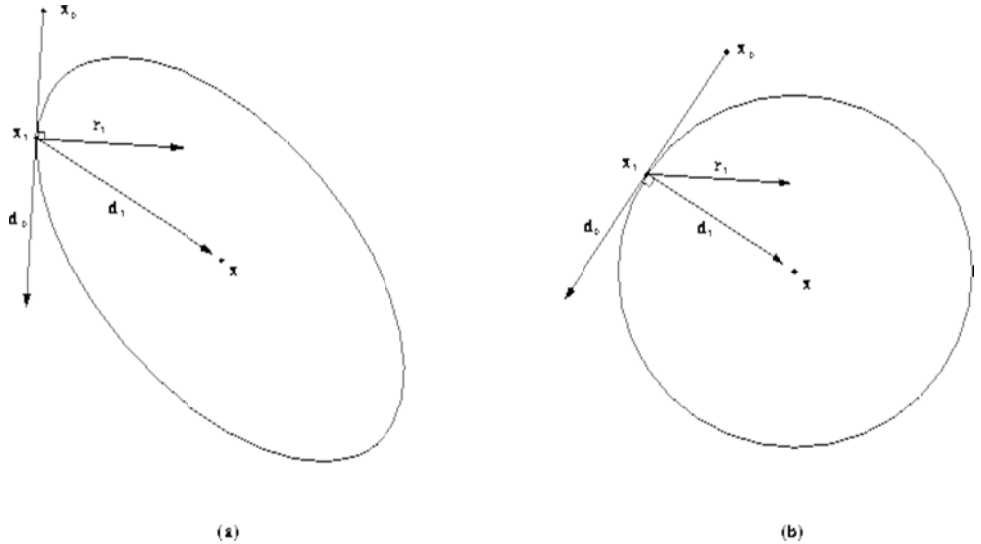

Figure 4.3: Optimality of the method of Conjugate Directions. (a) Lines that appear perpendicular are orthogonal. (b) The same problem in a ``stretched" space. Lines that appear perpendicular are $\mathbf{Q}$-orthogonal.

To avoid searching in directions that have been searched before, the Conjugate Direction guarantees that the minimization of $f(\mathbf{x}_k)$ along one direction does not ``spoil" the minimization along another; i.e. that after *i* steps, $f(\mathbf{x}_i)$ will be minimized over all searched directions. This is essentially what is stated in equation (4.9): A search along $\mathbf{d}_i$ has revealed where the gradient is perpendicular to $\mathbf{d}_i$, $\nabla f_i \cdot \mathbf{d}_i = 0$, and as we now move along some new direction $\mathbf{d}_j$, the gradient changes by $\delta(\nabla f) = \mathbf{Q} \cdot \mathbf{d}_j$ (see equation (4.6)). In order to not interfere with the minimization along $\mathbf{d}_i$, we require that the gradient remain perpendicular to $\mathbf{d}_i$; i.e. that the change in gradient itself be perpendicular to $\mathbf{d}_i$. Hence, we have equation (4.9). We see from Figure 4.3(b), where $\mathbf{d}_0$ and $\mathbf{d}_1$ appear perpendicular because they are $\mathbf{Q}$-orthogonal, that it is clear that $\mathbf{d}_1$ must point to the solution $\mathbf{x}$. A thorough description of the linear Conjugate Directions and Conjugate Gradients methods has been given by Shew.

The Conjugate Gradients method is a special case of the method of Conjugate Directions, where the conjugate set is generated by the gradient vectors. This seems to be a sensible choice since the gradient vectors have proved their applicability in the Steepest Descent method, and they are orthogonal to the previous search direction. For a quadratic function, as given in equation (4.6), the procedure is as follows.

The initial step is in the direction of the steepest descent:

$$\mathbf{d}_0 = -\mathbf{g}(\mathbf{x}_0) = -\mathbf{g}_0 \,. \qquad (4.10)$$

Subsequently, the mutually conjugate directions are chosen so that

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k \qquad k = 0, 1, \ldots , \qquad (4.11)$$

where the coefficient β_k is given by, for example, the so called Fletcher-Reeves formula:

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \cdot \mathbf{g}_{k+1}}{\mathbf{g}_k^T \cdot \mathbf{g}_k}. \qquad (4.12)$$

For details, see Walsh. The step length along each direction is given by

$$\lambda_k = \frac{\mathbf{d}_k^T \cdot \mathbf{g}_k}{\mathbf{d}_k^T \cdot (\mathbf{Q} \cdot \mathbf{d}_k)}. \qquad (4.13)$$

The resulting iterative formula is identical to (4.5).

When the matrix $\mathbf{Q}$ in (4.13) is not known, or is too computationally costly to determine, the stepsize can be found by linear searches. For the Conjugate Gradient method to converge in *n* iterations, these linear searches need to be accurate. Even small deviations can cause the search vectors to lose $\mathbf{Q}$-orthogonality, resulting in the method spending more than *n* iterations in locating the minimum point. In practice, accurate linear searches are impossible, both due to numerical accuracy and limited computing time. The direct use of equation (4.13) will most likely not bring us to the solution in *n* iterations either, the reason being the limited numerical accuracy in the computations which gradually will make the search vectors lose their conjugacy. It should also be mentioned that if the matrix $\mathbf{Q}$ is badly scaled, the convergence will be slowed down considerably, as it was for the Steepest Descent method.

Even though the Conjugate Gradients method is designed to find the minimum point for simple quadratic functions, it also does the job well for any continuous function $f(\mathbf{x})$ for which the gradient $\nabla f(\mathbf{x})$ can be computed. Equation (4.13) can not be used for non-quadratic functions, so the step length has to be determined by linear searches. The conjugacy of the generated directions may then progressively be lost, not only due to the inaccurate linear searches, but also due to the non-quadratic terms of the functions. An alternative to the Fletcher-Reeves formula that, to a certain extent, deals with this problem, is the Polak-Ribière formula Press:

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \cdot (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \cdot \mathbf{g}_k}. \qquad (4.14)$$

The difference in performance between these two formulas is not big though, but the Polak-Ribière formula is known to perform better for non-quadratic functions.

The iterative algorithm for the Conjugate Gradients method for non-quadratic functions using the Polak-Ribière formula is given in Algorithm 4.2.

Initializing: $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$, $\mathbf{d}_0 = -\mathbf{g}_0$ 5plus2.5minus10plus4minus65plus2.5minus

ΔDetermine the steplength λ_k: $\min_{\lambda_k > 0} f(\mathbf{x}_k + \lambda_k \mathbf{d}_k)$. Calculate the new point: $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$. Calculate the gradient: $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$. Determine the direction of search: $\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$,

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \cdot (\mathbf{g}_{k+1} - \mathbf{g}_k)}{\mathbf{g}_k^T \cdot \mathbf{g}_k}.$$

Algorithm 4.2: Method of Conjugate Gradients

Regardless of the direction-update formula used, one must deal with the loss of conjugacy that results from the the non-quadratic terms. The Conjugate Gradients method is often employed to problems where the number of variables *n* is large, and it is not unusual for the method to start generating nonsensical and inefficient directions of search after a few iterations. For this reason it is important to operate the method in cycles, with the first step being the Steepest Descent step. One example of a restarting policy is to restart with the Steepest Descent step after *n* iterations after the preceding restart.

Another practical issue relates to the accuracy of the linear search that is necessary for efficient computation. On one hand, an accurate linear search is needed to limit the loss of direction conjugacy. On the other hand, insisting on very accurate line search can be computationally expensive. To find the best possible middle path, trial and error is needed.

The Conjugate Gradients method is apart from being an optimization method, also one of the most prominent iterative method for solving sparse systems of linear equations. It is fast and uses small amounts of storage since it only needs the calculation and storage of the second derivative at each iteration. The latter becomes significant when *n* is so large that problems of computer storage arises. But, everything is not shiny; the less similar *f* is to a quadratic function, the more quickly the search directions lose conjugacy, and the method may in the worst case not converge at all. Although, it is generally to be preferred over the Steepest Descent method.

The Newton-Raphson Method

We now devote our attention to a method of category (3) from page . The method differs from the Steepest Descent and Conjugate Gradients method, which both are of category (2), in that the information of the second derivative is used to locate the minimum of the function $f(\mathbf{x})$. This results in faster convergence, but not necessarily less computing time. The computation of the second derivate and the handling of its matrix can be very time-consuming, especially for large systems.

The idea behind the *Newton-Raphson method* is to approximate the given function $f(\mathbf{x})$ in each iteration by a quadratic function, as given in equation (4.6), and then move to the minimum of this quadratic. The quadratic function for a point $\mathbf{x}$ in a suitable neighbourhood of the current point $\mathbf{x}_k$ is given by a truncated Taylor series:

$$f(\mathbf{x}) \approx f(\mathbf{x}_k) + (\mathbf{x} - \mathbf{x}_k)^T \cdot \mathbf{g}_k + \frac{1}{2}(\mathbf{x} - \mathbf{x}_k)^T \cdot \mathbf{H}_k \cdot (\mathbf{x} - \mathbf{x}_k), \qquad (4.15)$$

where both the gradient $\mathbf{g}_k$ and the Hessian matrix $\mathbf{H}_k$ are evaluated at $\mathbf{x}_k$. The derivative of (4.15) is

$$\nabla f(\mathbf{x}) = \mathbf{g}_k + \frac{1}{2}\mathbf{H}_k \cdot (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2}\mathbf{H}_k^T \cdot (\mathbf{x} - \mathbf{x}_k). \qquad (4.16)$$

The Hessian matrix is always symmetric if the function $\mathbf{x}_k$ is twice continuously differentiable at every point, which is the case here. Hence, (4.16) reduces to

$$\nabla f(\mathbf{x}) = \mathbf{g}_k + \mathbf{H}_k \cdot (\mathbf{x} - \mathbf{x}_k) . \qquad (4.17)$$

If we assume that $f(\mathbf{x})$ takes its minimum at $\mathbf{x} = \mathbf{x}^*$, the gradient is zero; i.e.

$$\mathbf{H}_k \cdot (\mathbf{x}^* - \mathbf{x}_k) + \mathbf{g}_k = 0 , \qquad (4.18)$$

which is noting else but a linear system. The Newton-Raphson method uses the $\mathbf{x}^*$ as the next current point, resulting in the iterative formula

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{H}_k^{-1} \cdot \mathbf{g}_k \qquad k = 0, 1, \ldots , \qquad (4.19)$$

where $-\mathbf{H}_k^{-1} \cdot \mathbf{g}_k$ is often referred to as the Newton direction.

If the approximation in (4.15) is valid, the method will converge in just a few iterations. For example, if the function to be optimized, $f(\mathbf{x})$, is a *n*-dimensional quadratic function, it will converge in only one step from any starting point.

Even though the convergence may seem to be fast, judged by the number of iterations, each iteration does include the calculation of the second derivative and handling of the Hessian. The performance of the method is therefore very dependent on certain qualities of

the Hessian. One of these qualities is positive definiteness. For the method to converge towards a minimum, the Newton direction needs to be a direction of descent. Thus, we require

$$\nabla f(\mathbf{x}_k) \cdot \mathbf{d}_k = \mathbf{g}_k^T \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) < 0\,, \qquad (4.20)$$

which, by inserting (4.19), gives

$$-(\mathbf{x}_{k+1} - \mathbf{x}_k)^T \cdot \mathbf{H}_k \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) < 0\,. \qquad (4.21)$$

This inequality is satisfied at all points for which $\mathbf{x}_{k+1} - \mathbf{x}_k \neq 0$ if $\mathbf{H}_k$ is positive definite. The further $\mathbf{x}_k$ is from the solution, the worse the quadratic approximation in (4.15) gets, which can result in the Hessian not being positive definite. The method is then no longer guaranteed to proceed toward a minimum; it may end up at any other critical point, whether it be a saddle point or a maximum point.

There are several ways to make sure that the Hessian is positive definite, e.g. by adding a large enough $\lambda \mathbf{I}$, where $\mathbf{I}$ is the unit matrix and λ is a positive scalar, or by diagonalizing the matrix by the use of eigenvalues. Descriptions of some of these methods can be found in textbooks like GMW,DS.

The size of the Hessian can also be crucial to the effectiveness of the Newton-Raphson method. For systems with a large number of dimensions, i.e. that the function $f(\mathbf{x})$ has a large number of variables, both the computation of the matrix, and calculations that includes it, will be very time-consuming. This can be mended by either just using the diagonal terms in the Hessian; i.e. ignoring the cross terms, or just not recalculate the Hessian at each iteration (can be done due to slow variation of the second derivative).

Another serious disadvantage of the Newton-Raphson method is that it is not necessarily globally convergent, meaning that it may not converge from any starting point. If it does converge, it is not unusual that it expend significant computational efforts in getting close enough to the solution where the approximation in (4.15) becomes valid, and the convergence is fast. An aid to this problem is to adjust the stepsize in the Newton direction, assuring the function value to decrease. Equation (4.19) is then as follows:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \mathbf{H}_k^{-1} \cdot \mathbf{g}_k \qquad k = 0, 1, \ldots . \qquad (4.22)$$

There are several methods for doing this, for example linear searches and fixed stepsize mentioned in Subsection 4.3.1, but it is recommended to use a backtracking scheme [e.g. by][]Press, where the full Newton step, $\lambda_k = 1$, is tried first. If this step fail to satisfy the criterion for the decrease of the function, one backtracks in a systematic way along the Newton direction. The advantage of doing this is that one will be able to enjoy the fast convergence of the Newton-Raphson method close to the solution. For details on the backtracking scheme, see for example [page 377]Press; [Section 6.3]DS.

In spite of these mentioned problems, the Newton-Raphson method enjoys a certain amount of popularity because of its fast convergence in a sufficiently small neighbourhood of the stationary value. The convergence is quadratic, which, loosely speaking, means that the number of significant digits double after each iteration. In comparison, the convergence of Steepest Descent method is at best linear and for the Conjugate Gradients method it is superlinear. The resultant algorithm for the Newton-Raphson method using backtracking is given in Algorithm 4.3.

5plus2.5minus10plus4minus65plus2.5minus

ΔCalculate the gradient: $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. Calculate the Hessian: $\mathbf{H}_k = \nabla^2 f(\mathbf{x}_k)$. Determine the direction of search: $\mathbf{d}_k = -\mathbf{H}_k^{-1} \cdot \mathbf{g}_k$ Determine the steplength λ_k by backtracking. Calculate the new point by: $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$.

Algorithm 4.3: The Newton-Raphson Method

As shown in Subsection 4.3.1, the method of Steepest Descent start out having a rather rapid convergence, while Newton-Raphson has just the opposite quality; starts out slowly and ends with a very rapid convergence. It is too obvious to be ignored that these two methods can be combined, resulting in a very efficient method. One starts out with the Steepest Descent method, and switch to the Newton-Raphson method when the progress by the former gets slow, and enjoys the quadratic convergence of the latter. An example of this type of combination is discussed next (and in Section 5.3: the Hybrid method).

Secant Method

Comparing the iterative formula of the Steepest Descent (4.5) with (4.22) shows that the Steepest Decent method is identical to the Newton-Raphson when $\mathbf{H}_k^{-1} = \mathbf{I}$, $\mathbf{I}$ being the unit matrix. This is the basis for the *Secant method*, also known as the *Variable Metric method*.

As seen in the previous subsection, the Hessian matrix needed in the Newton-Raphson method can be both laborious to calculate and invert for systems with a large number of dimensions. The idea of the Secant method is not to use the Hessian matrix directly, but rather start of the procedure with an approximation to the matrix, which, as one gets closer to the solution, gradually approaches the Hessian.

The earliest Secant methods were formulated in terms of constructing a sequence of matrices $\mathbf{B}_k$ which builds up the *inverse* Hessian; that is,

$$\lim_{k \to \infty} \mathbf{B}_k = \mathbf{H}^{-1}. \qquad (4.23)$$

Even better if the limit is achieved after n iterations instead of infinitely many iterations. The iterative formula is then

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k \mathbf{B}_k \cdot \mathbf{g}_k \qquad k = 0, 1, \ldots, \qquad (4.24)$$

where $\mathbf{g}_k$ is the gradient. Later, methods for constructing the ordinary Hessian were presented. The inverse Hessian approximation might seem to offer an advantage in terms of the number of arithmetic operation required to perform an iteration, since one needs to inverse the matrix for the Hessian update. This seeming defect was eliminated by, instead of updating the Hessian, the Cholesky factors $\mathbf{L}^T\mathbf{L}$ of the Hessian were modified at each iteration GMW. The number of arithmetic operations is then the same for the two methods. Both methods are being used today, the choice being a matter of convenience and taste. The inverse Hessian update will be used in this thesis, and therefore only this method will be presented in the following. For details on the Hessian update, see GMW,DS.

For a n-dimensional quadratic function, the inverse Hessian will be achieved after at most n iterations, and the solution will then be located, from what we saw in Subsection 4.3.3, in only one iteration. Hence, the solution will be arrived at after at most n iterations. In other words, this method resembles the method of Steepest Descent near the starting point of the procedure, while near the optimal point, it metamorphose into the Newton-Raphson method.

The Secant method is quite similar to the Conjugate Gradients, in some senses. They both converge after *n* iterations for an *n*-dimensional quadratic function, they both accumulate information from successive iterations, and they both require the calculation of only the first derivative. (The latter makes the Secant method part of category (2) from page .) The main difference between the two methods is the way they store and update the information; the Secant method requires an $n \times n$ matrix while the Conjugate Gradients method only needs an *n*-dimensional vector. Usually, the storage of the matrix in the Secant method is no disadvantage, at least not for a moderate number of variables.

So how are the $\mathbf{B}_k$ being updated at each iteration? There are two main schemes GMW, namely Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS). Let us first consider the DFP algorithm. At a point $\mathbf{x}_k$, the approximated inverse Hessian at the subsequent point is given by Press

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{p}_k \times \mathbf{p}_k}{\mathbf{y}_k \cdot \mathbf{p}_k} - \frac{(\mathbf{B}_k \cdot \mathbf{y}_k) \times (\mathbf{B}_k \cdot \mathbf{y}_k)}{\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)}, \qquad (4.25)$$

where $\mathbf{p}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$ and $\mathbf{y}_k = \mathbf{g}_{k+1} - \mathbf{g}_k$, $\mathbf{g}$ being the gradient. This expression is constructed by forcing the $\mathbf{p}$-s to be mutually $\mathbf{Q}$-conjugate. Even though the deduction of equation (4.25) is in regard of a quadratic function, it basically applies for all kinds of equations/systems. Another property of (4.25) is that if $\mathbf{B}_0$ is positive definite, it will remain so for each *k*; i.e. $\mathbf{B}_k$ is a sequence of positive definite matrices. For proofs and a detailed deduction of (4.25), see Walsh.

Initializing: $\mathbf{g}_0 = \nabla f(\mathbf{x}_0)$, $\mathbf{d}_0 = -\mathbf{g}_0$, $\mathbf{B}_0 = \mathbf{I}$
5plus2.5minus10plus4minus65plus2.5minus

ΔDetermine the steplength λ_k by backtracking. Calculate the new point: $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{d}_k$. Calculate the gradient: $\mathbf{g}_{k+1} = \nabla f(\mathbf{x}_{k+1})$. Calculate the update to the inverse Hessian:

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{p}_k \times \mathbf{p}_k}{\mathbf{y}_k \cdot \mathbf{p}_k} - \frac{(\mathbf{B}_k \cdot \mathbf{y}_k) \times (\mathbf{B}_k \cdot \mathbf{y}_k)}{\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)} + [\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)]\, \mathbf{u} \times \mathbf{u},$$

$$\mathbf{u} \equiv \frac{\mathbf{p}_k}{\mathbf{y}_k \cdot \mathbf{p}_k} - \frac{(\mathbf{B}_k \cdot \mathbf{y}_k)}{\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)}$$

Determine the direction of search: $\mathbf{d}_{k+1} = -\mathbf{B}_{k+1} \cdot \mathbf{g}_{k+1}$

Algorithm 4.4: The Secant Method

The BFGS algorithm was later introduced to improved some unbecoming features of the DF**P** formula, concerning, for example, roundoff error and convergence tolerance. The BFGS updating formula is exactly the same, apart from one additional term Press

$$\ldots + [\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)]\, \mathbf{u} \times \mathbf{u}, \qquad (4.26)$$

where **u** is defined by

$$\mathbf{u} \equiv \frac{\mathbf{p}_k}{\mathbf{y}_k \cdot \mathbf{p}_k} - \frac{(\mathbf{B}_k \cdot \mathbf{y}_k)}{\mathbf{y}_k \cdot (\mathbf{B}_k \cdot \mathbf{y}_k)}. \qquad (4.27)$$

The BFGS formula is usually to be preferred GMW.

The step length λ_k in equation (4.24) is determined as for the Newton-Raphson method; i.e either by linear search, backtracking, or by deciding on $0 < \lambda \leq 1$ and gradually changing it to unity as the solution is being approached. The algorithm for the BFGS update with backtracking is given in Algorithm 4.4.

The Secant method has become very popular for optimization: it converges fast (at best superlinear), it is stable, and spends relatively modest computing time at each iteration. The method is often being preferred over the Conjugate Gradients method, but the latter will have an advantage over the former for systems with large number of dimensions (e.g. 10,000). Since both the DFP and the BFGS updating formulas are guaranteed to uphold the positive definiteness of the initial approximation of the inverse Hessian (assuming that there are no roundoff errors that results in loss of positive definiteness), the Secant method will only be able to locate minimum points.