# Notes for AMTH 370/371: Optimization Techniques

Santa Clara University
School of Engineering
Department of Applied Mathematics

Glenn Williams

June 4, 2018

# Contents

# 1 General Introduction

Root-finding (including $A\vec{x} = \vec{b}$) and optimization are the most common problems in numerical computing. Optimization problems are those in which we need to find the value(s) of an argument $x$ that makes some function $F(x)$ take on a value that is either a local minimum or maximum. Maximization and minimization are trivially related to each other (e.g., $\max F(x) \sim \min F(-x)$).

The function $F$ is typically called the *objective function*. Independent variables $x$ can be continuous, discrete, integer, or permutations. For the purpose of our work, we will consider continuous independent variables. If there are no limits places on $x$ within their domain, the optimization is called *unconstrained*. If there are limits placed on $x$, the optimization is called *constrained*.

## 1.1 Notation

Our $n$-dimensional working space will be denoted by $\Re^n$

Single-valued functions of multiple variables will be denoted by $f : \Re^n \to \Re$ or $f(\vec{x})$

The $i^{\text{th}}$ component of vector $\vec{x}$ will be denoted by $x^i$

The scalar, or dot, product will be denoted by $(x, y)$ or $\langle x, y \rangle$ or $x^T y$,

where dot product is defined as $(x, y) = \sum_{i=1}^{n} x^i y^j$

The gradient (vector of partial derivatives) will be denoted by $\nabla f$ or $f'$

The Hessian (matrix of second partial derivatives) will be denoted by $\nabla^2 f$ or $f''$

## 1.2 Problem Formulation

Typically, we will work with functions of more than 1 variable. We can formulate this problem as follows.

Given a set $X$ and a function $f : X \to \Re$, find $x^* \in X$ such that, for all $x \in X$, there holds $f(x) \geq f(x^*)$ (for minimizing the function) or $f(x) \leq f(x^*)$ (for maximizing the function).

We will consider only the case where $X$ is a subset of $\Re^n$, defined by inequality constraints $I$ and/or equality constraints $E$. Given a number $m_I + m_E$ of functions $c_j : \Re^n \to \Re$ for $j = 1, \ldots, m_I + m_E$, the optimization problem (minimization/maximization) can be stated as:

$$\begin{cases} \min/\max\ f(x) & x \in \Re^n \\ c_j(x) \leq 0 & j \in I \\ c_j(x) = 0 & j \in E \end{cases}$$

# 2 1D Searches

We will first consider functions of a single variable $f : \Re \to \Re$. Our ultimate goal is the study of optimization techniques, but included in that is the consideration of root-finding techniques (techniques for finding the point(s) $x$ such that $f(x) = 0$). Recall from a first course in Calculus that a method for finding the local minimum/maximum of a function of one variable, $f(x)$ is to find the critical points, or the roots of $f'(x)$. We will first examine the case where the desired point (min, max, or root) lies within a bounded, or bracketed, interval in $x$.

## 2.1 Bracketed Search

### 2.1.1 Root-Finding

A simple bisection search is one of the most basic search methods, which can be used to find the root of a function when it is known that the root lies within a given interval in $x$. We will say that a root is *bracketed* in the interval $(a, b)$ if $f(a)$ and $f(b)$ have opposite signs. By the intermediate value theorem, if the function is continuous, then at least one root must lie in that interval. If the function is discontinuous, but bounded, then instead of a root there might be a step discontinuity which crosses zero.

The idea behind the *bisection method* is simple. Over some interval, the function is known to pass through zero because it changes sign. Evaluate the function at the interval's midpoint and examine its sign. If the function at the midpoint is the same sign as the left endpoint of the interval, make the midpoint the new left endpoint. If the function at the midpoint is the same sign as the right endpoint of the interval, make the midpoint the new right endpoint. The following is pseudocode to describe this method:

INPUT: Function $f$, endpoint values $a$, $b$, $f_a$, $f_b$, tolerance $tol$, maximum iterations $maxniter$

CONDITIONS: $a < b$, either $f(a) < 0$ and $f(b) > 0$ or $f(a) > 0$ and $f(b) < 0$

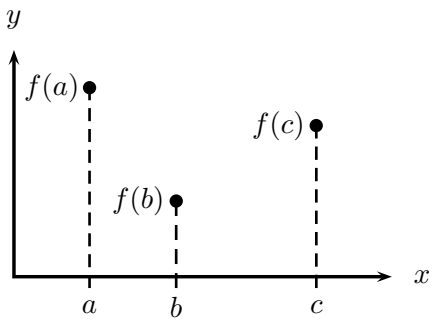
OUTPUT: Value which differs from a root of $f(x)$ by less than $tol$

function $[x_{zero}] = \text{bisection}(a, b, f_a, f_b, tol, maxniter)$

$niter = 1$
WHILE $niter \leq maxniter$
$c = (a + b)/2$
$f_c = f(c)$
IF $\text{abs}(b - a) < tol$
break
IF $\text{sign}(f_c) = \text{sign}(f_a)$
$a = c$
$f_a = f_c$
ELSE
$b = c$
$f_b = f_c$
$niter = niter + 1$
$x_{zero} = c$

### 2.1.2 Minimization

Finding the minimum of a function when we know it lies in an interval $(a, b)$ is similar to finding the root of a function when we know it lies in an interval $(a, b)$. The main difference is that a root of a function can be bracketed by 2 points, $a < b$, whereas a minimum of a function is known to be bracketed only when there is a triplet of points, $a < b < c$, such that $f(b)$ is less than both $f(a)$ and $f(c)$.




The analog of bisection for function minimization is to choose a new point $d$, either between $a$ and $b$ or between $b$ and $c$. It remains to decide on a strategy for choosing the new point $d$, given $a$, $b$, and $c$. Suppose that $b$ is a fraction $W$ of the way between $a$ and $c$; i.e.,

$$\frac{b-a}{c-a} = W \qquad \frac{c-b}{c-a} = 1 - W$$

Also suppose that the next trial point $d$ is an additional fraction $Z$ beyond $b$

$$\frac{d-b}{c-a} = Z$$

The next bracketing segment will either be from $a$ to $d$ or from $b$ to $c$, and thus will either be of length $W + Z$ relative to the current one or else of length $1 - W$. If we want to minimize the worst case possibility, then we will choose $Z$ to make these equal, namely

$$Z = 1 - 2W \tag{1}$$

Therefore, the new point $d$ is the symmetric point to $b$ in the interval, with $|b-a| = |d-c|$. This implies that the point $d$ lies in the larger of the two segments ($Z$ is positive only if $W < 1/2$). But where in the larger segment do we place $d$? If $Z$ was chosen to be optimal, then so was $W$ before it. The scale similarity implies that $d$ should be the same fraction of the way from $b$ to $c$ as $b$ was from $a$ to $c$, or

$$\frac{Z}{1-W} = W \tag{2}$$

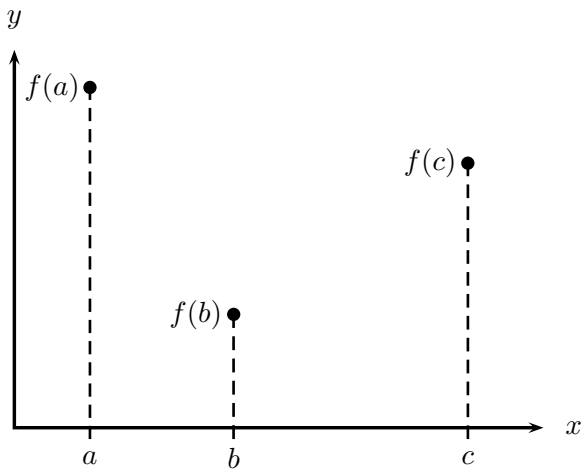
Substituting Equation (1) into Equation (2) gives

$$\begin{aligned} \frac{1-2W}{1-W} &= W \\ 1-2W &= W - W^2 \\ W^2 - 3W + 1 &= 0 \\ W &= \frac{3-\sqrt{5}}{2} \approx 0.38197 \end{aligned}$$

In other words, the optimal bracketing interval $a < b < c$ has its middle point $b$ a fractional distance 0.38197 from one end (say $a$) and 0.61803 from the other end (say $b$). These fractions are those of the so-called *golden mean* or *golden section*. This optimal method of function minimization, the analog of bisection method for finding zeros, is called the *golden section search*. The method can be summarized as follows:

Start with the interval $(a, c)$ that you know contains a local minimum. Choose point $b$ such that

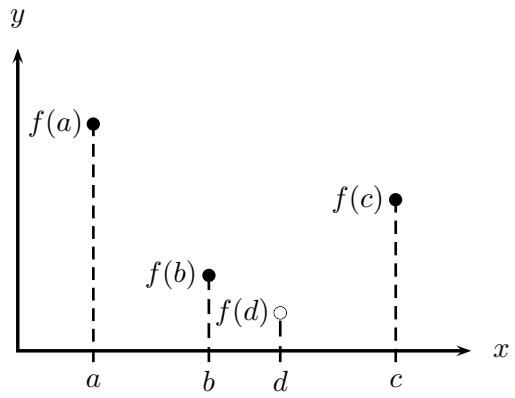
$$b = a + 0.38197 * (c - a)$$



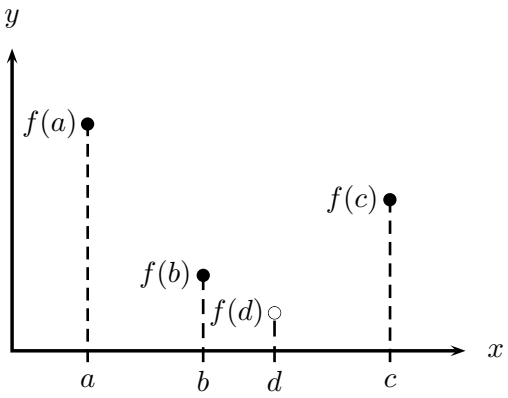

Note: $f(b)$ must be less than $f(a)$ and $f(c)$. If it is not, then replace one of the end points with $b$ and continue until you find a triplet $a < b < c$ such that $f(b) < f(a)$ and $f(b) < f(c)$.

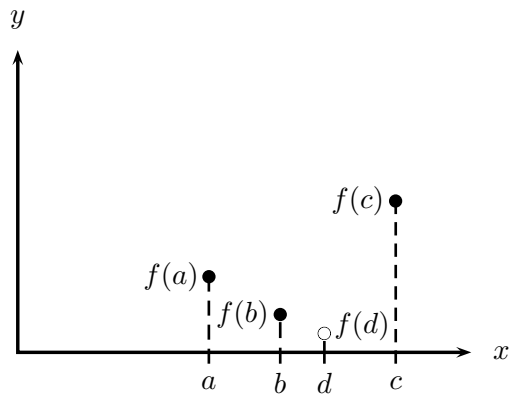
The algorithm will then proceed so that point $d$ is always chosen to be 0.38197 of the way from $b$ towards $c$. At each iteration of the algorithm, points $a$, $b$, and $c$ will be assigned based on whether $f(d) < f(b)$ or $f(d) > f(b)$. It could be that $c < b < a$ (the $(a, c)$ interval could become $(c, a)$), but $b$ will always be 0.38197 of the way from $a$ towards $c$. The following illustrations show the 4 possibilities of the first 2 iterations of the algorithm:

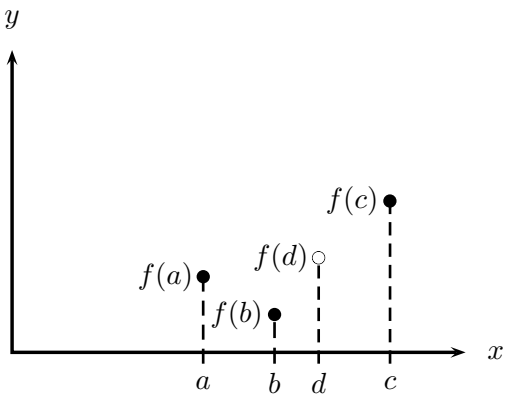
Iteration 1: $f(d) < f(b)$
Iteration 2: $f(d) < f(b)$

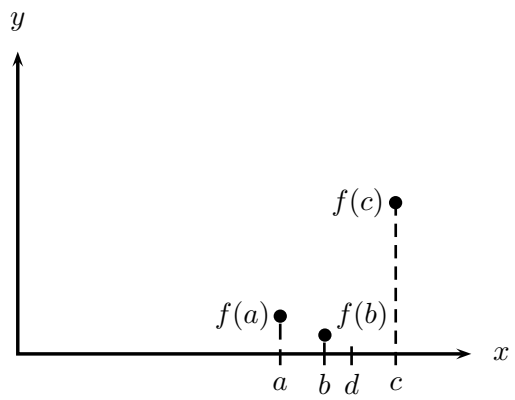


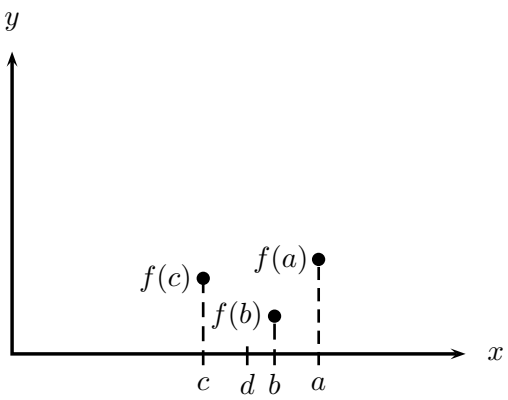

Iteration 1: $f(d) < f(b)$
Iteration 2: $f(d) > f(b)$



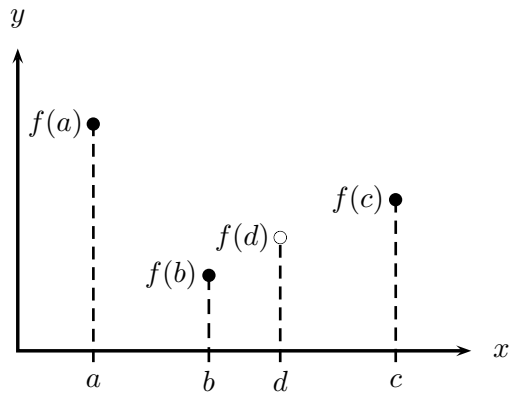



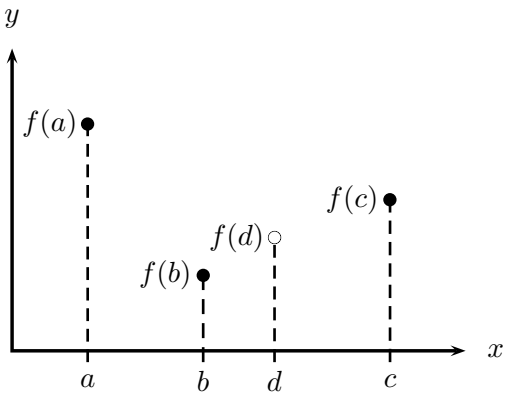



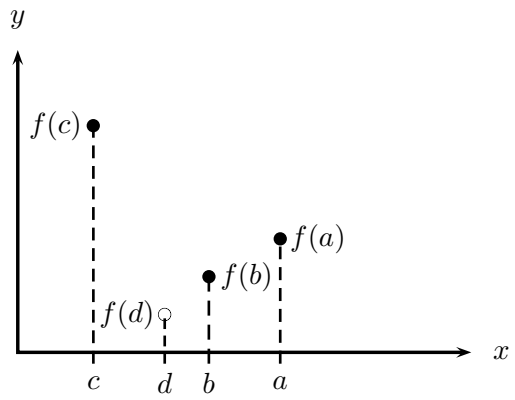



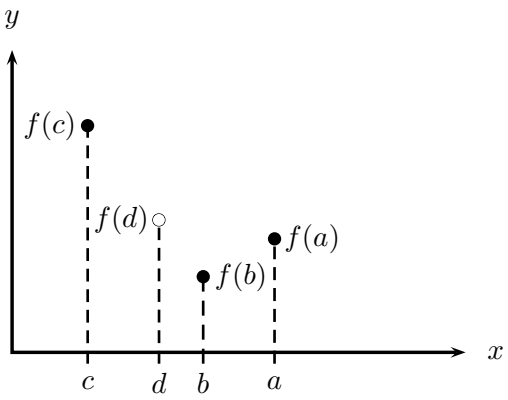
Iteration 1: $f(d) > f(b)$
Iteration 2: $f(d) < f(b)$

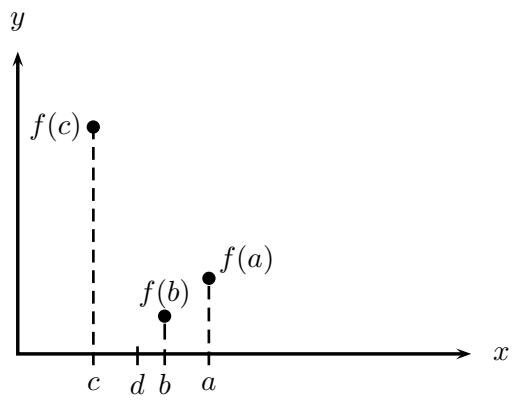


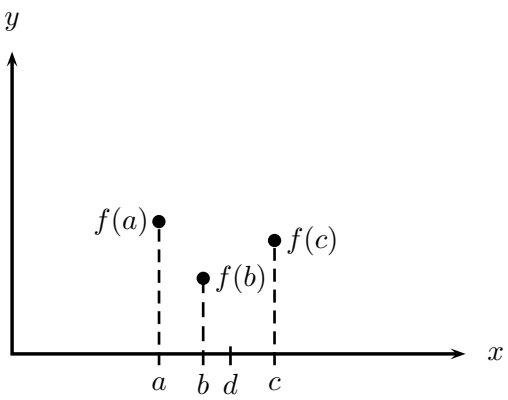

Iteration 1: $f(d) > f(b)$
Iteration 2: $f(d) > f(b)$



The following is pseudocode to describe the golden search method:

INPUT: Function $f$, values $a$, $b$, $c$, tolerance $tol$, maximum iterations $maxniter$

CONDITIONS: $a < b < c$, $f(b) < f(a)$, and $f(b) < f(c)$

OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, and the function value at that point, $f_{min}$

function $[x_{min},\ f_{min}]$ = goldenSearch$(a, b, c, tol, maxniter)$

$niter = 1$

WHILE $niter \leq maxniter$

IF abs$(c - a) < tol$

break

$d = b + 0.38197 * (c - b)$

IF $f(d) < f(b)$

$a = b$

$b = d$

ELSE

$c = a$

$a = d$

$niter = niter + 1$

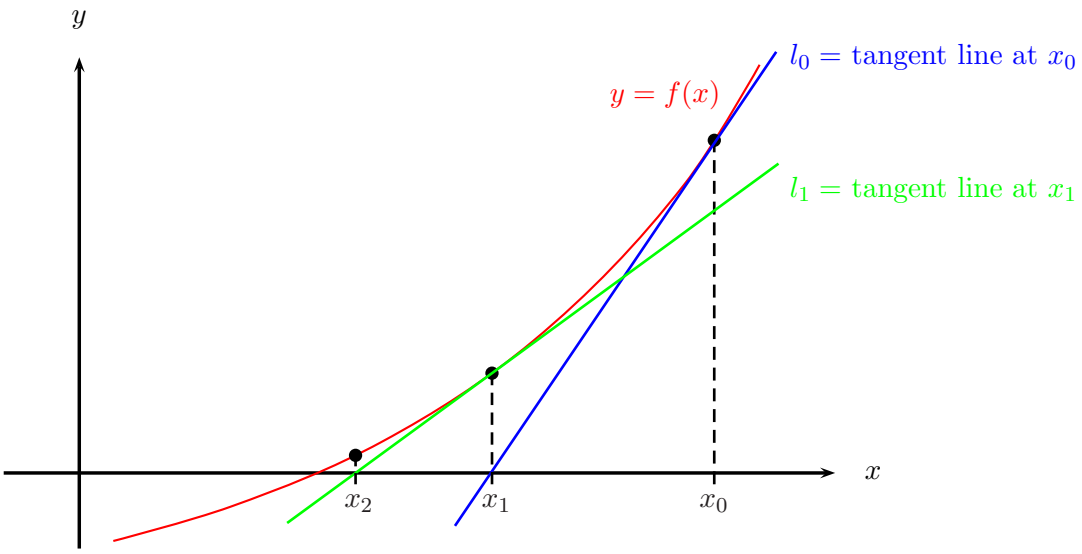
$x_{min} = b$

$f_{min} = f_b$

## 2.2 Newton's Method

### 2.2.1 Root-Finding

Newton's method for finding the root of a function is based on the idea of using the tangent line to the function at a given point as a linear approximation to the function at that point. We don't know where the function itself crosses the $x$-axis, but we can determine where the tangent line to the function at that point crosses the $x$-axis. We can then use this point where the tangent line crosses the $x$-axis as the next point in an iterative process. The result of this iterative process will be the point where the function crosses the $x$-axis, or the root of the function. The following diagram illustrates this process:




Newton's method can be viewed in one of two ways. It can be viewed as moving directly to the zero of the linear approximation, or as having the linear approximation determine a length and direction, $\Delta x$, and then multiplying $\Delta x$ by some factor $\lambda$ to determine the next point. The $\Delta x$ approach will be useful in making necessary adjustments for non-smooth or non-convex functions.

**2.2.1.1 Moving to Zero** Given the function $y = f(x)$, let $l_i(x) = a\,x + b$ be the tangent line at $x_i$. We can determine $a$ and $b$, and thus $l_i(x)$, as follows:

$$
\begin{aligned}
l_i'(x_i) &= f'(x_i) \equiv f_i' \\
a &= f_i' \\
l_i(x_i) &= f(x_i) \equiv f_i \\
f_i' x_i + b &= f_i \\
b &= f_i - f_i' x_i \\
l_i(x) &= f_i' x + f_i - f_i' x_i
\end{aligned}
$$

The point where $l_i(x)$ intersects the $x$-axis is found by setting $l_i(x) = 0$:

$$
\begin{aligned}
l_i(x) = f_i' x + f_i - f_i' x_i &= 0 \\
f_i' x &= f_i' x_i - f_i \\
x &= x_i - \frac{f_i}{f_i'}
\end{aligned}
$$

Therefore, the Newton iteration is simply $x_{i+1} = x_i - \frac{f_i}{f_i'}$

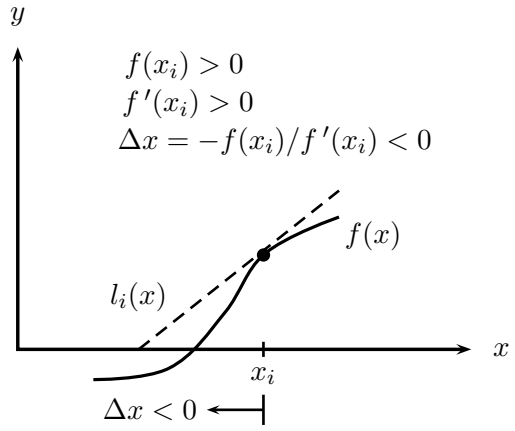
**2.2.1.2 $\Delta x$ Approach** The Newton iteration step above can be viewed as

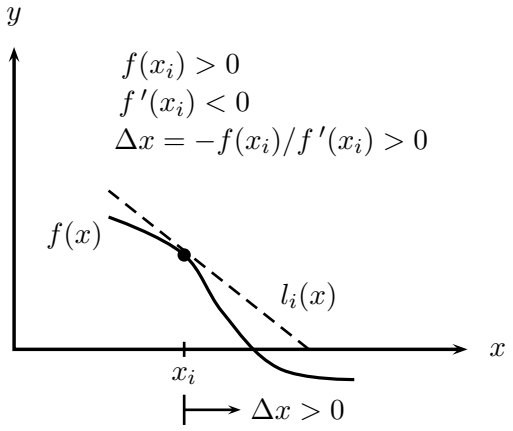
$$x_{i+1} = x_i + \Delta x_i \text{ , where } \Delta x_i = -\frac{f_i}{f_i'}$$

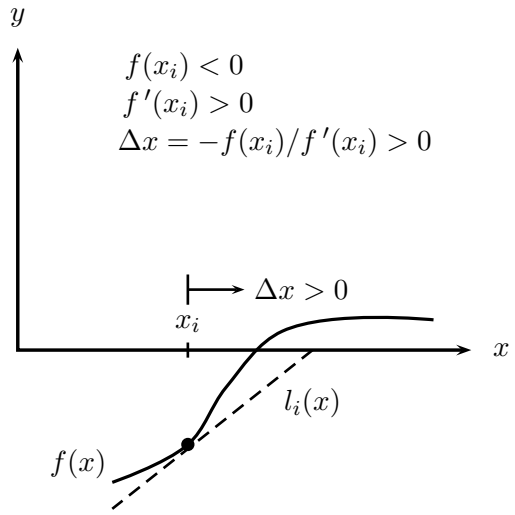
This might seem like a trivial distinction, but it becomes important for functions $f(x)$ where Newton's method does not converge nicely to a root. For these cases, an adjustment has to be made to attempt to achieve convergence. This adjustment can be made through a correction to the $\Delta x$ term; i.e.,

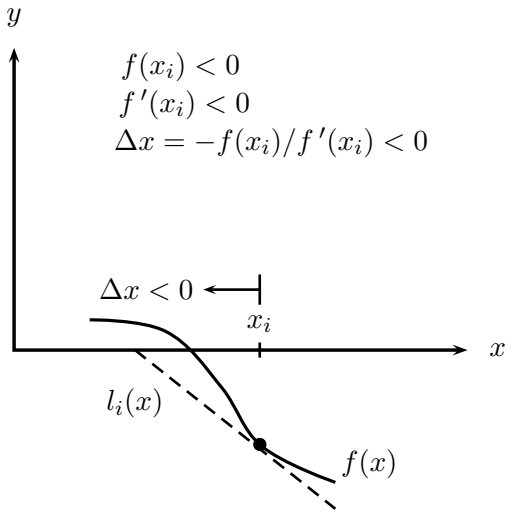
$$x_{i+1} = x_i + \lambda \Delta x_i \text{ , where } \Delta x_i = -\frac{f_i}{f_i'} \text{ and } 0 < a < 1$$

In the $\Delta x$ approach, a Newton iteration is basically split into 2 parts – 1) finding a search vector $\Delta x$, and 2) adjusting the magnitude of that vector through the scalar $\lambda$. When $f(x)$ is a function of 1 variable, the direction is simply positive or negative (right or left). But when we consider functions of more than 1 variable, $\Delta x$ will be a vector, and then we will perform a 1D search along that direction to find the optimal $\lambda$. See the following diagram regarding direction for 1D root finding.

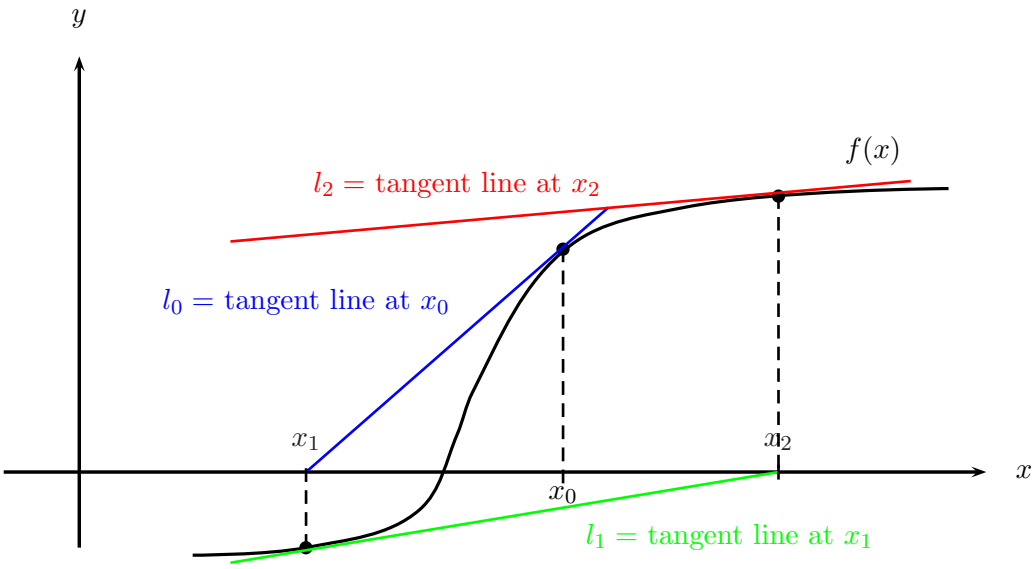

y
$f(x_i) > 0$
$f'(x_i) > 0$
$\Delta x = -f(x_i)/f'(x_i) < 0$
$f(x)$
$l_i(x)$
x
$x_i$
$\Delta x < 0$


y
$f(x_i) > 0$
$f'(x_i) < 0$
$\Delta x = -f(x_i)/f'(x_i) > 0$
$f(x)$
$l_i(x)$
x
$x_i$
$\Delta x > 0$

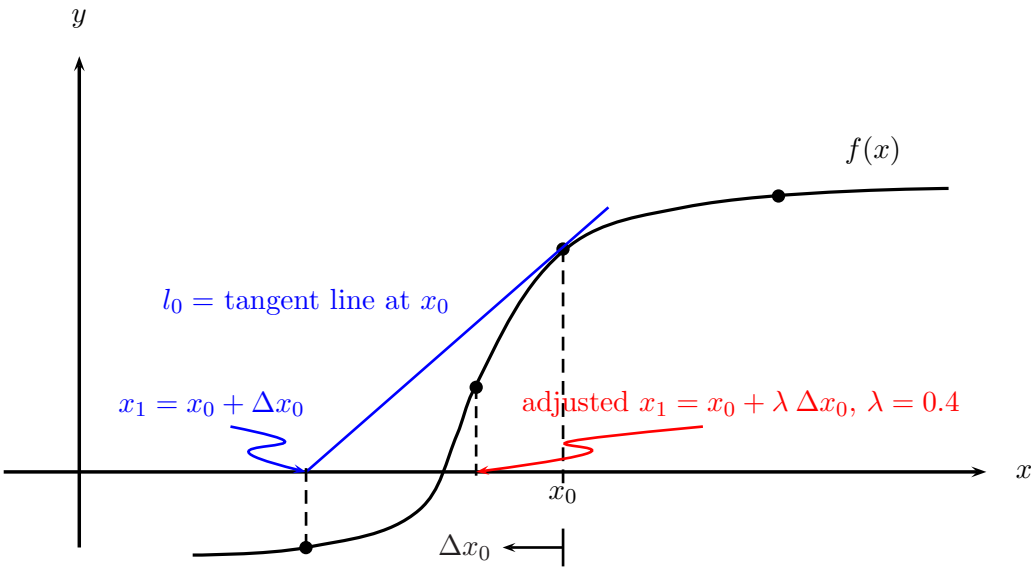

y
$f(x_i) < 0$
$f'(x_i) > 0$
$\Delta x = -f(x_i)/f'(x_i) > 0$
$\Delta x > 0$
$x_i$
x
$l_i(x)$
$f(x)$


y
$f(x_i) < 0$
$f'(x_i) < 0$
$\Delta x = -f(x_i)/f'(x_i) < 0$
$\Delta x < 0$
$x_i$
x
$l_i(x)$
$f(x)$

The following example shows where Newton's method may fail to find the root, and thus would need a $\lambda$ adjustment of $\Delta x$. A method for finding $\lambda$ will be discussed in Section 2.3.




We could adjust the length to move in the direction for $x_0$; i.e., $x_1 = x_0 + \lambda \Delta x_0$, $0 < \lambda < 1$.



The following is pseudocode to describe Newton's method for finding a root of the function $f(x)$, where $f : \Re \to \Re$.

INPUT: Functions $f(x)$ and $f'(x)$, starting value $x_0$, tolerance $tol$, maximum iterations $maxniter$

OUTPUT: Value which differs from a root of $f(x)$ by less than $tol$, and the number of iterations taken, $niters$

function [$x_{zero}$,$niters$] = newtonRoot($x_0, tol, maxniter$)

$iter = 1$

$x = x_0$

WHILE $iter \leq maxniter$

$xprev = x$

$\Delta x = -f(x)/f'(x)$

$x = x + \Delta x$

IF abs($x - xprev$) $< tol$

break

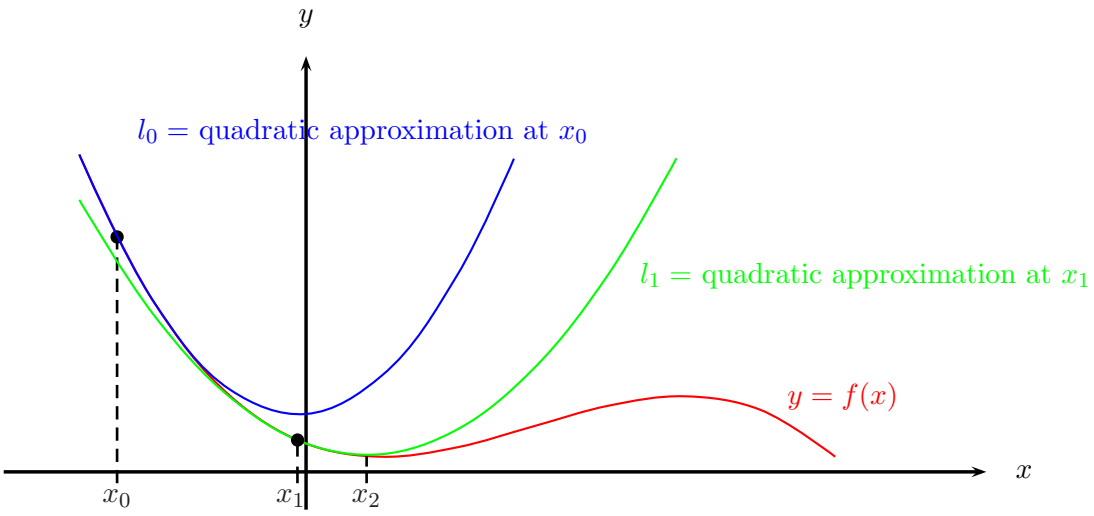
$iter = iter + 1$

$x_{zero} = x$

$niters = iter$

### 2.2.2 Minimization

Newton's method for finding the local minimum of a function is based on the idea of using the quadratic approximation to the function at a given point. We don't know where the function itself attains a minimum, but we can determine in one step where the quadratic approximation attains its minimum. We can then use this point where the quadratic approximation attains its minimum as the next point in an iterative process. The result of this iterative process will be the point where the function attains its local minimum. The following diagram illustrates this process:




Again, Newton's method can be viewed in one of two ways. It can be viewed as moving directly to the minimum of the quadratic approximation, or as having the quadratic approximation determine a length and direction, $\Delta x$, and then multiplying $\Delta x$ by some factor $\lambda$ to determine the next point. The $\Delta x$ approach will be useful in making necessary adjustments for non-smooth or non-convex functions.

**2.2.2.1 Moving to Minimum** Given the function $y = f(x)$, let $q_i(x) = a\,x^2 + b\,x + c$ be the parabola that is tangent to $f(x)$ at $x_i$. We can determine $a$, $b$, and $c$, and thus $q_i(x)$, as follows:

$$\begin{aligned} q_i''(x_i) &= f''(x_i) \equiv f_i'' \\ 2\,a &= f_i'' \\ a &= \frac{1}{2} f_i'' \end{aligned}$$

$$\begin{aligned} q_i'(x_i) &= f'(x_i) \equiv f_i' \\ 2\,a\,x_i + b &= f_i' \\ f_i''\,x_i + b &= f_i' \\ b &= f_i' - f_i''\,x_i \end{aligned}$$

$$\begin{aligned} q_i(x_i) &= f(x_i) \equiv f_i \\ \frac{1}{2} f_i''\,x_i^2 + \left(f_i' - f_i''\,x_i\right) x_i + c &= f_i \\ c &= f_i - \left(f_i' - f_i''\,x_i\right) x_i - \frac{1}{2} f_i''\,x_i^2 \\ &= f_i - f_i'\,x_i + \frac{1}{2} f_i''\,x_i^2 \end{aligned}$$

$$q_i(x) = \left(\frac{1}{2} f_i''\right) x^2 + \left(f_i' - f_i''\,x_i\right) x + f_i - f_i'\,x_i + \frac{1}{2} f_i''\,x_i^2$$

$q_i(x)$ attains its minimum value at $x = -b/2a$

$$\begin{aligned} x &= \frac{-b}{2a} \\ &= \frac{-\left(f_i' - f_i''\,x_i\right)}{2\left(\frac{1}{2} f_i''\right)} \\ &= \frac{f_i''\,x_i - f_i'}{f_i''} \\ x &= x_i - \frac{f_i'}{f_i''} \end{aligned}$$

Therefore, the Newton iteration is simply $x_{i+1} = x_i - \frac{f_i'}{f_i''}$

**2.2.2.2 $\Delta x$ Approach** The Newton iteration step above can be viewed as

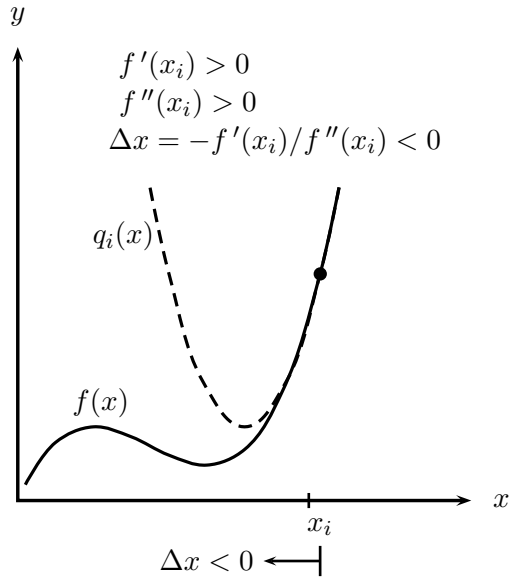
$$x_{i+1} = x_i + \Delta x_i \text{ , where } \Delta x_i = -\frac{f_i'}{f_i''}$$

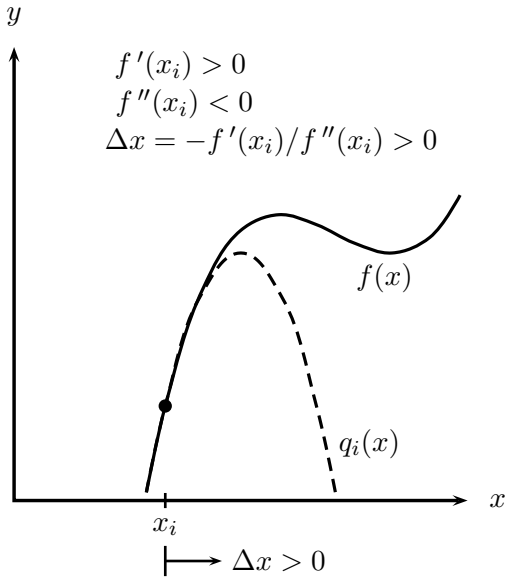
This might seem like a trivial distinction, but it becomes important for functions $f(x)$ where Newton's method does not converge nicely to a minimum. For these cases, an adjustment has to be made to attempt to achieve convergence. This adjustment can be made through a correction to the $\Delta x$ term; i.e.,

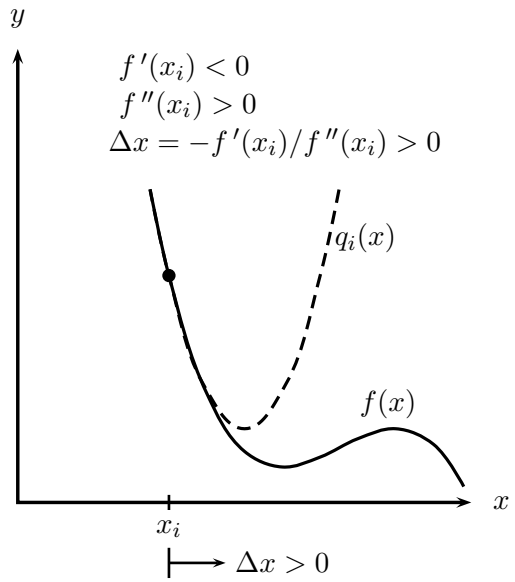
$$x_{i+1} = x_i + \lambda\, \Delta x_i \text{ , where } \Delta x_i = -\frac{f_i'}{f_i''} \text{ and } 0 < a < 1$$

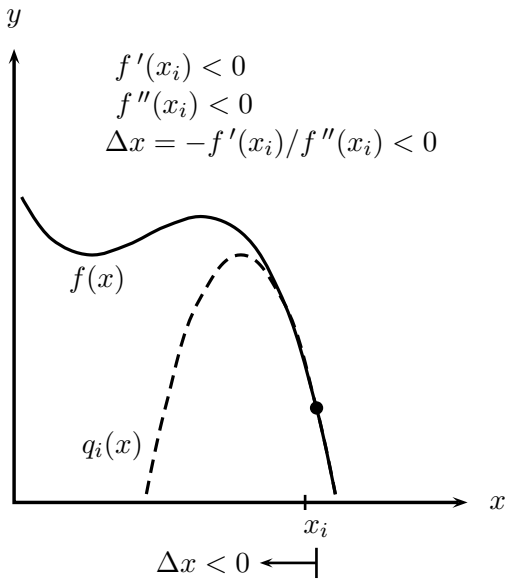
In the $\Delta x$ approach, a Newton iteration is basically split into 2 parts – 1) finding a search vector $\Delta x$, and 2) adjusting the magnitude of that vector through the scalar $\lambda$. When $f(x)$ is a function of 1 variable, the direction is simply positive or negative (right or left). But when we consider functions of more than 1 variable, $\Delta x$ will be a vector, and then we will perform a 1D search along that direction to find the optimal $\alpha$. See the following diagram regarding direction for 1D minimization/maximization.

Note that for the cases on the right, we are finding a maximum rather than a minimum. We will work mainly with minimization of functions. In these cases, we will require a movement in a descent direction; i.e., the derivative is negative in the direction of $\Delta x$ (directional derivative $< 0$). For the maximization problems shown, the derivative is positive in the direction of $\Delta x$.

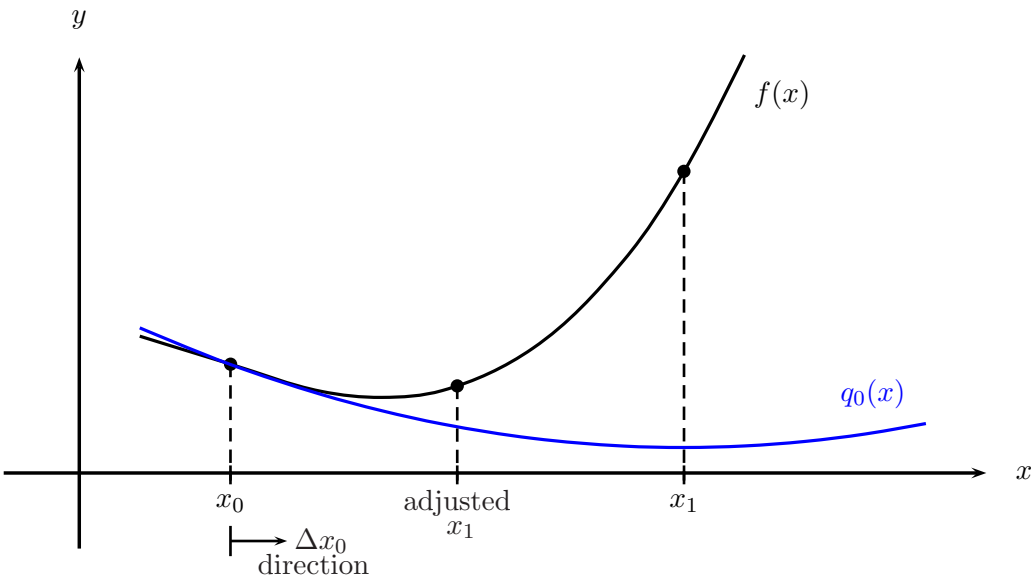

y
$f'(x_i) > 0$
$f''(x_i) > 0$
$\Delta x = -f'(x_i)/f''(x_i) < 0$
$q_i(x)$
$f(x)$
x
$x_i$
$\Delta x < 0$


y
$f'(x_i) > 0$
$f''(x_i) < 0$
$\Delta x = -f'(x_i)/f''(x_i) > 0$
$f(x)$
$q_i(x)$
x
$x_i$
$\Delta x > 0$


y
$f'(x_i) < 0$
$f''(x_i) > 0$
$\Delta x = -f'(x_i)/f''(x_i) > 0$
$q_i(x)$
$f(x)$
x
$x_i$
$\Delta x > 0$


y
$f'(x_i) < 0$
$f''(x_i) < 0$
$\Delta x = -f'(x_i)/f''(x_i) < 0$
$f(x)$
$q_i(x)$
x
$x_i$
$\Delta x < 0$

**Example 2.2.2.3.** Consider the following example where Newton's method may fail to find the minimum, or take many iterations to find the minimum.




Notice that the first iteration, $x_1 = x_0 + \Delta x_0$, takes us to a function value that is greater than the starting function value. We could adjust the length to move in the direction $\Delta x_0$ so that we end up with a function value that is below some maximum value; i.e., adjusted $x_1 = x_0 + \lambda\,\Delta x_0$, $0 < \lambda < 1$. A method for finding such a $\lambda$ will be discussed in the following section (Section 2.3).

The following is pseudocode to describe Newton's method for finding a minimum of the function $f(x)$, where $f : \Re \to \Re$.

INPUT: Functions $f(x)$, $f'(x)$, and $f''(x)$, starting value $x_0$, tolerance $tol$, maximum iterations $maxniter$

OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = newtonMin($x_0, tol, maxniter$)

$iter = 0$

$x = x_0$

WHILE $iter \leq maxniter$

$xprev = x$

$\Delta x = -f'(x)/f''(x)$

$x = x + \Delta x$

IF $x$ unacceptable (we will discuss this later)

Calculate $\lambda$

$x = x + \lambda \Delta x$

IF abs($x - xprev$) $< tol$

break

$iter = iter + 1$

$x_{min} = x$

$f_{min} = f(x)$

$niters = iter$

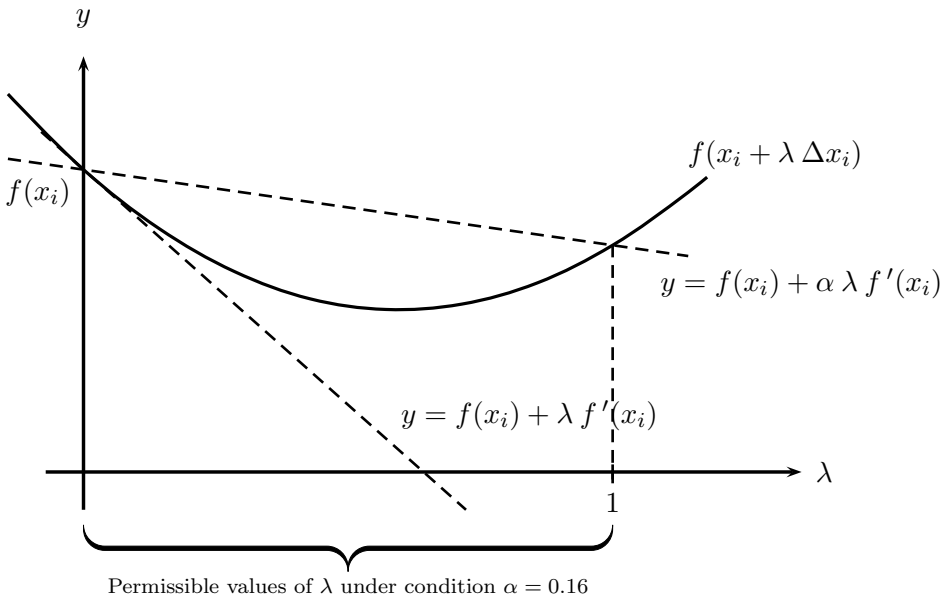
## 2.3 Backtracking Line Search (Polynomial Interpolation)

The idea of a line-search algorithm is simple: given a descent direction $\Delta x_i$, take a step in that direction that yields an "acceptable" $x_{i+1}$. That is, find $\lambda$ such that $x_{i+1} = x_i + \lambda \Delta x_i$ is acceptable. The common procedure is to first try the full Newton step; i.e., $\lambda = 1$. If that step fails to satisfy the criterion in use, then backtrack in a systematic way along the direction defined by that step. Computational experience has shown the importance of taking a full Newton step whenever possible. While no step acceptance rule will always be optimal, it does seem to be common sense to require that

$$f(x_{i+1}) < f(x_i)$$

This simple condition does not guarantee that $x_k$ will converge to a minimizer of $f$. In some cases, we can achieve very small decreases in $f$ relative to the length of the steps. We can fix this by requiring that the average rate of decrease from $f(x_i)$ to $f(x_{i+1})$ be at least some prescribed fraction of the initial rate of decrease in that direction; that is, we pick an $\alpha \in (0, 1)$ and choose $\lambda$ from among those $\lambda > 0$ that satisfy

$$f(x_{i+1}) \;=\; f(x_i + \lambda\, \Delta x_i) \;\leq\; f(x_i) \;+\; \alpha\, \lambda\, f\,'(x_i) \tag{3}$$

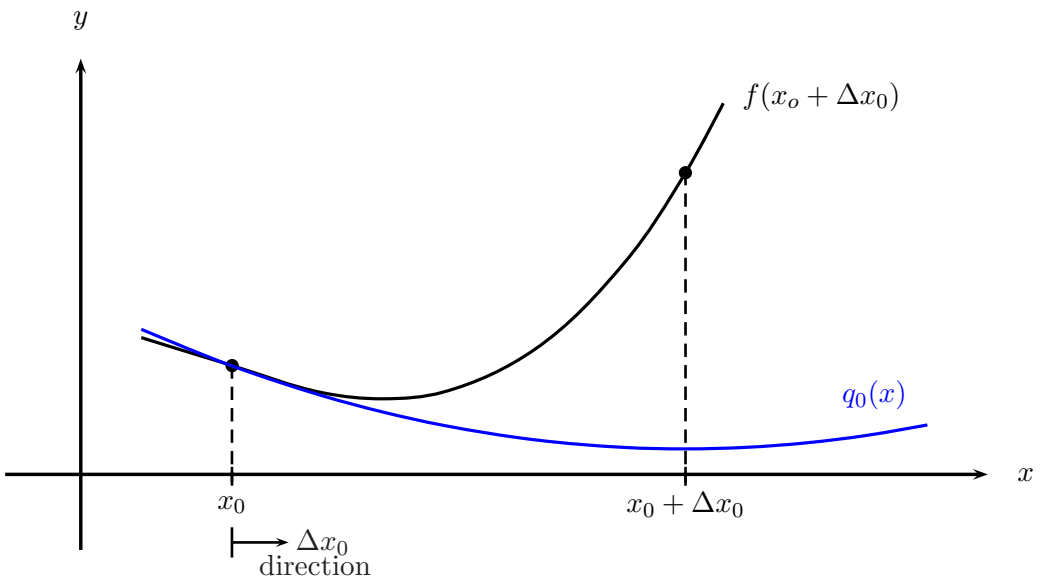


Thus, this line search is attempting to find $\lambda$ that will satisfy the criterion given by Equation (3). This criterion does guarantee that $f(x_{i+1})$ be less than $f(x_i)$, since $\alpha > 0$, $\lambda > 0$, and $f'(x_i) < 0$ (for descent direction). In practice, $\alpha$ is chosen to be very small, so that hardly more than a decrease in function value is required. A typical value is about $10^{-4}$.

For this method, we will assume that we only have the values of the function and its derivative at a given point. This type of 1D search is often used with methods that approximate second derivative information (Hessian matrices) from gradient information. Thus, we can calculate a first-order directional derivative using the gradient information at that point. Consider the previous example

Each cut of the line search is similar to an iteration of Newton's method for minimization in that it fits a polynomial in the given search direction and then moves directly to the minimum of that interpolating polynomial. The first cut of this line search will fit a quadratic function and then subsequent cuts, if necessary, will fit cubic functions. The $\lambda$ values found will be $< 1$, thus this line search method is called a backtracking line search.

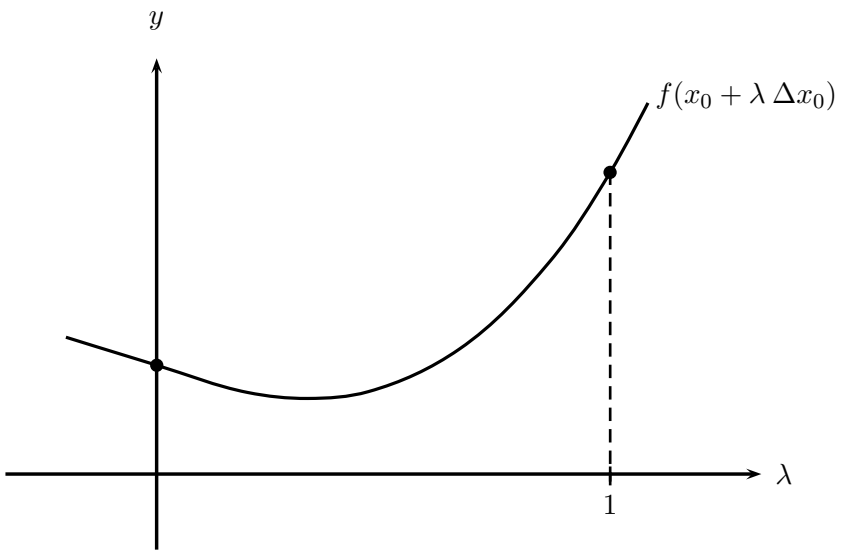
Recall Example 2.2.2.3. In that case, the function value from the full Newton iteration at the first step would not satisfy the criterion; i.e., $f(x_1) > f(x_0) + \alpha\,\lambda\,f'(x_0)$, $\lambda = 1$. Therefore, we could adjust $\lambda$ using our backtracking line search.



At this point, we have three pieces of information available regarding the function $f$

$$f(x_0)$$
$$f'(x_0)$$
$$f(x_0 + \Delta x_0)$$

Change the horizontal axis from $x$ to $\lambda$ because, for the purpose of this line search, the independent variable is $\lambda$.




These three values can be used to determine the quadratic function $m_q(\lambda)$ that fits $f(x_0 + \lambda\, \Delta x_0)$ such that

$$\begin{aligned} m_q(0) &= f(x_0) \equiv f_0 \\ m_q'(0) &= f'(x_0) \equiv f_0' \\ m_q(1) &= f(x_0 + \Delta x_0) \equiv f_1 \end{aligned}$$

Let $m_q(\lambda) = a\,\lambda^2 + b\,\lambda\, +\, c$. We can determine $a$, $b$, and $c$, and thus $m_q(\lambda)$, as follows:

$$\begin{aligned} m_q(0) &= c = f_0 \\ m_q'(0) &= b = f_0' \\ m_q(1) &= a + b + c = f_1 \\ a + f_0' + f_0 &= f_1 \\ a &= f_1 - f_0 - f_0' \\ m_q(x) &= \left(f_1 - f_0 - f_0'\right) x^2 + \left(f_0'\right) x + f_0 \end{aligned}$$

$m_q(\lambda)$ attains its minimum value at $\hat{\lambda} = -b/2a$

$$\begin{aligned} \hat{\lambda} &= \frac{-b}{2a} \\ \hat{\lambda} &= \frac{-f_0'}{2\left(f_1 - f_0 - f_0'\right)} \end{aligned} \tag{4}$$

Also

$$m_q''(\lambda) = 2\left(f_1 - f_0 - f_0'\right)$$

Since

$$f_1 > f_0 + \alpha\, f_0' > f_0 + f_0'$$

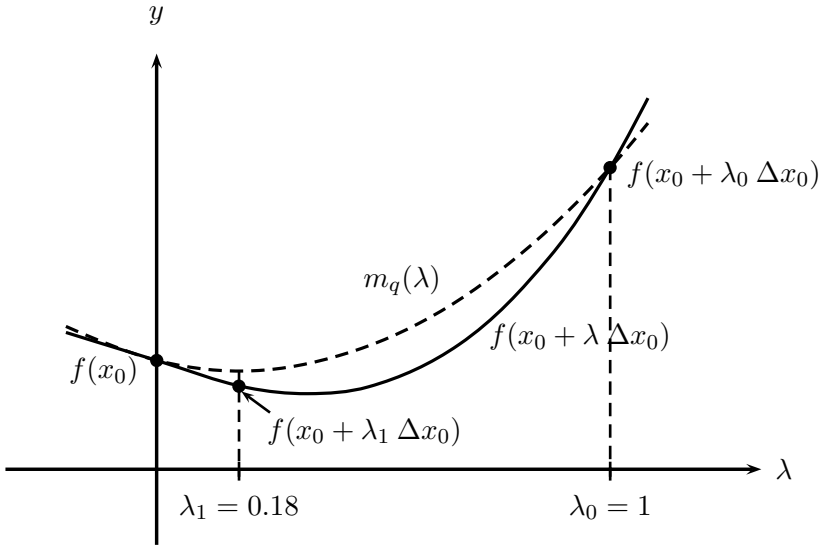
Then

$$m_q'' > 0$$

Therefore, $\hat{\lambda}$ minimizes $m_q(\lambda)$. Also, $\hat{\lambda} > 0$ because $f_0' < 0$. We take $\hat{\lambda}$ as our new value of $\lambda$. Note that since $f_1 > f_0 + \alpha f_0'$, then

$$\begin{aligned}
f_1 &> f_0 + \alpha f_0' \\
f_1 - f_0 &> \alpha f_0' \\
f_1 - f_0 - f_0' &> \alpha f_0' - f_0' \\
2\left(f_1 - f_0 - f_0'\right) &> 2 f_0' (\alpha - 1) \\
\frac{1}{2(f_1 - f_0 - f_0')} &< \frac{1}{2 f_0' (\alpha - 1)} \\
\frac{-f_0'}{2(f_1 - f_0 - f_0')} &< \frac{-f_0'}{2 f_0' (\alpha - 1)} \\
\hat{\lambda} &< \frac{1}{2(1 - \alpha)}
\end{aligned}$$

In fact, if $f_1 \geq f_0$, then $\hat{\lambda} \leq \frac{1}{2}$. Thus, Equation (4) gives a useful upper bound of $\approx \frac{1}{2}$ on the first cut of the line search. This means that if $\hat{\lambda} > \frac{1}{2}$, we set the new $\lambda_1 = \frac{1}{2}$. On the other hand, if $f_1$ is much larger than $f_0$, $\hat{\lambda}$ can be very small. In that case, we would not want to decrease $\lambda$ too much since it probably indicates that $f(\lambda)$ is poorly modeled by a quadratic in this region. Therefore, we impose a lower bound of $\frac{1}{10}$. This means that if $\hat{\lambda} < \frac{1}{10}$, we set the new $\lambda_1 = \frac{1}{10}$. Consider again the example, with $m_q(\lambda)$ drawn in:



Our new value of $\lambda_1 = 0.18$ yields an acceptable function value. The line search would stop at this point. This particular line search is not intended to find a minimum value within a given tolerance, but is only searching for a value that meets the criterion given in Equation (3).

If our $\lambda_1$ did not yield an acceptable value, we would continue with the line search. From this point on, we will use four values to fit a cubic function, $m_c(\lambda)$ and then move to the minimum of that cubic function. The four values we will have regarding the function $f$ are:

$$
\begin{aligned}
&f(x_0) \\
&f'(x_0) \\
&f(x_0 + \lambda_{prev}\, \Delta x_0) \\
&f(x_0 + \lambda_{2prev}\, \Delta x_0)
\end{aligned}
$$

These four values can be used to determine the cubic function $m_c(\lambda)$

$$
\begin{aligned}
m_c(0) &= f(x_0) \equiv f_0 \\
m_c'(0) &= f'(x_0) \equiv f_0' \\
m_c(\lambda_{prev}) &= f(x_0 + \lambda_{prev}\, \Delta x_0) \equiv f_{\lambda prev} \\
m_c(\lambda_{2prev}) &= f(x_0 + \lambda_{2prev}\, \Delta x_0) \equiv f_{\lambda 2prev}
\end{aligned}
$$

Let $m_c(\lambda) = a\,\lambda^3 + b\,\lambda^2 + c\,\lambda + d$. We can determine $a$, $b$, $c$, and $d$, and thus $m_c(\lambda)$, as follows:

$$m_c(0) = d = f_0$$

$$m_c'(0) = c = f_0'$$

$$
\begin{aligned}
m_c(\lambda_{prev}) &= a\,(\lambda prev)^3 + b\,(\lambda prev)^2 + f_0'\,(\lambda prev) + f_0 = f_{\lambda prev} \\
m_c(\lambda_{2prev}) &= a\,(\lambda 2prev)^3 + b\,(\lambda 2prev)^2 + f_0'\,(\lambda 2prev) + f_0 = f_{\lambda 2prev}
\end{aligned}
$$

The last two equations can be solved for $a$ and $b$

$$\begin{aligned} a\,(\lambda prev)^3 \;+\; b\,(\lambda prev)^2 \;&=\; f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ a\,(\lambda 2prev)^3 \;+\; b\,(\lambda 2prev)^2 \;&=\; f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{aligned}$$

This is the system of equations

$$A \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{bmatrix}$$

where

$$A \;=\; \begin{bmatrix} \lambda prev^3 & \lambda prev^2 \\ \lambda 2prev^3 & \lambda 2prev^2 \end{bmatrix}$$

The solution is then

$$\begin{bmatrix} a \\ b \end{bmatrix} \;=\; A^{-1} \begin{bmatrix} f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} \;=\; \frac{1}{\lambda prev^3 \lambda 2prev^2 - \lambda 2prev^3 \lambda prev^2} \begin{bmatrix} \lambda 2prev^2 & -\lambda prev^2 \\ -\lambda 2prev^3 & \lambda prev^3 \end{bmatrix} \begin{bmatrix} f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} \;=\; \frac{1}{\lambda prev^2\, \lambda 2prev^2\, (\lambda prev - \lambda 2prev)} \begin{bmatrix} \lambda 2prev^2 & -\lambda prev^2 \\ -\lambda 2prev^3 & \lambda prev^3 \end{bmatrix} \begin{bmatrix} f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{bmatrix}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} \;=\; \frac{1}{\lambda prev - \lambda 2prev} \begin{bmatrix} \frac{1}{\lambda prev^2} & \frac{-1}{\lambda 2prev^2} \\ \frac{-\lambda 2prev}{\lambda prev^2} & \frac{\lambda prev}{\lambda 2prev^2} \end{bmatrix} \begin{bmatrix} f_{\lambda prev} \;-\; f_0'\,(\lambda prev) \;-\; f_0 \\ f_{\lambda 2prev} \;-\; f_0'\,(\lambda 2prev) \;-\; f_0 \end{bmatrix}$$

$m_c(\lambda)$ attains its minimum value at

$$m_c' \;=\; 3\,a\,\hat{\lambda}^2 \;+\; 2\,b\,\hat{\lambda} \;+\; c \;=0$$

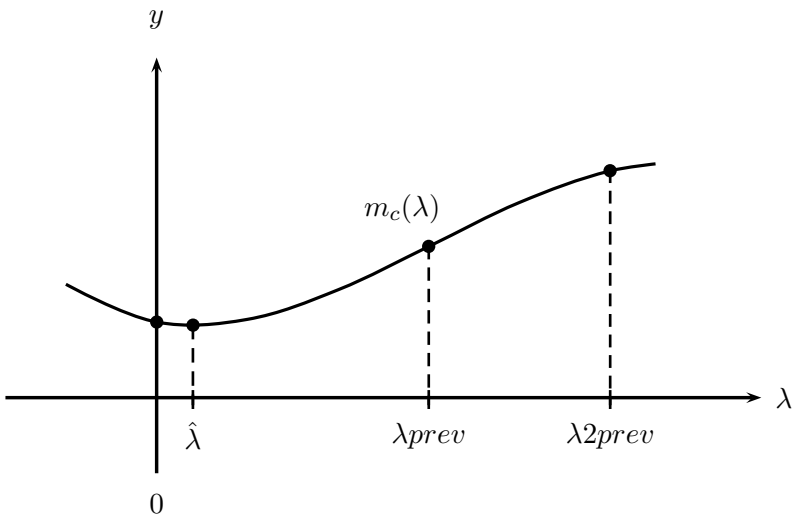
$$\begin{aligned}
\hat{\lambda} \;&=\; \frac{-2b \pm \sqrt{4b^2 - 12ac}}{6a} \\
\hat{\lambda} \;&=\; \frac{-2b \pm 2\sqrt{b^2 - 3ac}}{6a} \\
\hat{\lambda} \;&=\; \frac{-b \pm \sqrt{b^2 - 3ac}}{3a}
\end{aligned}$$

$\hat{\lambda}$ is chosen to be the point where $m_c''(\hat{\lambda}) \;>\; 0$. Then $\hat{\lambda}$ is the point where $m_c$ attains a local minimizes $m_q(\lambda)$.

It can be shown that if $f_{\lambda prev} \geq f_0$, then $\hat{\lambda} < \frac{2}{3}\,\lambda prev$. But this reduction is considered too small. Therefore, the upper bound of 0.5 is again imposed, which means that if $\hat{\lambda} > \frac{1}{2}\lambda prev$, we set the new $\lambda_k = \frac{1}{2}\,\lambda prev$. Also, since $\hat{\lambda}$ can be an arbitrarily small fraction of $\lambda prev$, the lower bound of 0.1 is used again, which means that if $\hat{\lambda} < \frac{1}{10}\,\lambda prev$, we set the new $\lambda_k = \frac{1}{10}\,\lambda prev$.

To illustrate the cubic portion of the line search, consider the following example where the full Newton step ($\lambda = 1$) and one cut of the line search (the quadratic function $m_q$) have been evaluated and the resulting function value is not acceptable. The full Newton step in this case will be the $\lambda 2prev$ and the first cut of the line search will be the $\lambda prev$. Assume the following values:

$$\begin{aligned}
f_0 \;&=\; 1 \\
f_0' \;&=\; -1 \\
\lambda 2prev \;&=\; 1 \qquad & f_{\lambda 2prev} \;&=\; 3 \\
\lambda prev\; 0.6 \qquad & & f_{\lambda prev} \;&=\; 2
\end{aligned}$$




To find $\hat{\lambda}$, we would make the following calculations:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\lambda prev - \lambda 2prev} \begin{bmatrix} \frac{1}{\lambda prev^2} & \frac{-1}{\lambda 2prev^2} \\ \frac{-\lambda 2prev}{\lambda prev^2} & \frac{\lambda prev}{\lambda 2prev^2} \end{bmatrix} \begin{bmatrix} f_{\lambda prev} - f_0'(\lambda prev) - f_0 \\ f_{\lambda 2prev} - f_0'(\lambda 2prev) - f_0 \end{bmatrix}$$

$$= \frac{1}{0.6 - 1} \begin{bmatrix} \frac{1}{(0.6)^2} & -1 \\ \frac{-1}{(0.6)^2} & 0.6 \end{bmatrix} \begin{bmatrix} 2 + 0.6 - 1 \\ 3 + 1 - 1 \end{bmatrix}$$

$$= 2.5 \begin{bmatrix} -2.\bar{7} & 1 \\ 2.\bar{7} & -0.6 \end{bmatrix} \begin{bmatrix} 1.6 \\ 3 \end{bmatrix} = \begin{bmatrix} -3.6\bar{1} \\ 6.6\bar{1} \end{bmatrix}$$

$$m_c(\lambda) = -3.6\bar{1}\,x^3 + 6.6\bar{1}\,x^2 - x + 1$$
$$m_c'(\lambda) = -10.8\bar{3}\,x^2 + 13.\bar{2}\,x - 1$$

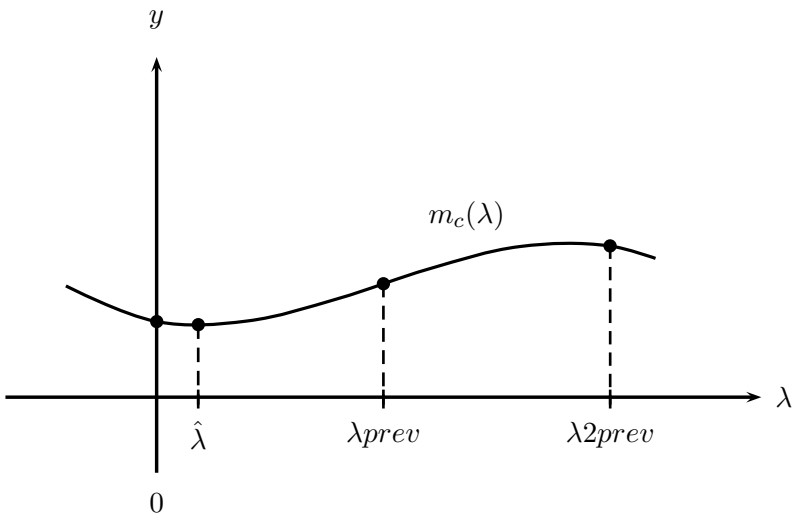
$m_c'(\lambda) = 0$ at

$$\hat{\lambda} = \frac{-13.\bar{2} \pm \sqrt{(13.\bar{2})^2 - 4\,(10.8\bar{3})}}{2\,(-10.8\bar{3})}$$
$$= \frac{-13.\bar{2} \pm 11.467}{2\,(-21.\bar{6})} = 0.081, 1.14$$

We would take the first value, 0.081, to be $\hat{\lambda}$ because $m_c''(0.081) > 0$. Note that this is the $\frac{-b+\sqrt{b^2-3ac}}{3a}$ value rather than the $\frac{-b-\sqrt{b^2-3ac}}{3a}$ value.

Consider one more example where the full Newton step ($\lambda = 1$) and one cut of the line search (the quadratic function $m_q$) have been evaluated and the resulting function value is not acceptable. As before, the full Newton step in this case will be the $\lambda 2prev$ and the first cut of the line search will be the $\lambda prev$. Assume the following values:

$$f_0 = 1$$
$$f_0' = -1$$
$$\lambda 2prev = 1 \qquad f_{\lambda 2prev} = 2$$
$$\lambda prev\, 0.5 \qquad f_{\lambda prev} = 1.5$$



To find $\hat{\lambda}$, we would make the following calculations:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \frac{1}{\lambda prev - \lambda 2prev} \begin{bmatrix} \frac{1}{\lambda prev^2} & \frac{-1}{\lambda 2prev^2} \\ \frac{-\lambda 2prev}{\lambda prev^2} & \frac{\lambda prev}{\lambda 2prev^2} \end{bmatrix} \begin{bmatrix} f_{\lambda prev} - f_0'(\lambda prev) - f_0 \\ f_{\lambda 2prev} - f_0'(\lambda 2prev) - f_0 \end{bmatrix}$$

$$= \frac{1}{0.5 - 1} \begin{bmatrix} \frac{1}{(0.5)^2} & -1 \\ \frac{-1}{(0.5)^2} & 0.5 \end{bmatrix} \begin{bmatrix} 1.5 + 0.5 - 1 \\ 2 + 1 - 1 \end{bmatrix}$$

$$= 2 \begin{bmatrix} -4 & 1 \\ 4 & -0.5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -4 \\ 6 \end{bmatrix}$$

$$\begin{aligned} m_c(\lambda) &= -4x^3 + 6x^2 - x + 1 \\ m_c'(\lambda) &= -12x^2 + 12x - 1 \end{aligned}$$

$m_c'(\lambda) = 0$ at

$$\begin{aligned} \hat{\lambda} &= \frac{-12 \pm \sqrt{(12)^2 - 4(12)}}{2(-12)} \\ &= \frac{-12 \pm 4\sqrt{6}}{-24} \\ &= \frac{-3 \pm \sqrt{6}}{-6} = 0.092, 0.908 \end{aligned}$$

We would take the first value, 0.092, to be $\hat{\lambda}$ because $m_c''(0.092) > 0$. Note that this is the $\frac{-b+\sqrt{b^2-3ac}}{3a}$ value rather than the $\frac{-b-\sqrt{b^2-3ac}}{3a}$ value. This is not a proof, but it turns out that the local minimizing point of $m_c$ is $\frac{-b+\sqrt{b^2-3af_0'}}{3a}$.

The following is pseudocode to describe the quadratic/cubic backtracking line search.

INPUT: $x_0$, $f(x_0) \equiv f_0$, $f'(x_0) \equiv f_0'$, $\Delta x$, $\alpha$, $maxncuts$, $ll$ (lower limit for $\hat{\lambda}$), $ul$ (upper limit for $\hat{\lambda}$).

CONDITIONS: $f'(x_i) < 0$

OUTPUT: Value for $\hat{\lambda}$ that satisfies $f(x_0 + \hat{\lambda}\,\Delta x) \leq f(x_0) + \alpha\,\hat{\lambda}\,f'(x_0)$, or a flag denoting that the condition was not satisfied and thus the line search failed

function [$\hat{\lambda}$, $success$,$ncuts$] = quadCubicLS($x_0, f_0, f_0', \Delta x, \alpha, maxncuts, ll, ul$)

$cut = 1$

$success = 0$

$\lambda = 1$

$\lambda_{prev} = \lambda$

$f_\lambda = f(x_0 + \lambda\,\Delta x)$

$f_{\lambda prev} = f_\lambda$

WHILE $cut \leq maxncuts$

IF $f_\lambda \leq f_0 + \alpha\,\lambda\,f_0'$

$success = 1$

break

IF $cut = 1$

$\lambda = \frac{-f_0'}{2(f_\lambda - f_0' - f_0)}$

ELSE

Calculate $a$ and $b$

$\lambda = \frac{-b + \sqrt{b^2 - 3af_0'}}{3a}$

IF $\lambda > ul * \lambda prev$

$\lambda = ul * \lambda prev$

IF $\lambda < ll * \lambda prev$

$\lambda = ll * \lambda prev$

$f_\lambda = f(x_0 + \lambda\,\Delta x)$

$\lambda 2prev = \lambda prev$

$\lambda prev = \lambda$

$f_{\lambda 2prev} = f_{\lambda prev}$

$f_{\lambda prev} = f_{\lambda}$

$cut = cut + 1$

IF $success \quad \hat{\lambda} = \lambda$

ELSE $\quad \hat{\lambda} = 0$

$ncuts = cut$

## 2.4 Evaluating Derivatives

For some functions, a closed-form expression might not exist for the derivative. For others, it can be computationally expensive to evaluate the closed-form expression for the derivative. In these cases, we will examine two options for the approximation of the derivative; the finite difference approximation and the secant approximation.

### 2.4.1 Numerical Differentiation

Formulas for numerical approximation of derivatives are typically derived from a Taylor series expansion of a function $u(x)$ about a point $x$. First order approximations are derived as follows

$$u(x + h) = u(x) + h \frac{du}{dx}(x) + \frac{h^2}{2} \frac{d^2u}{dx^2}(\xi) , \quad \xi \in (x, x+h)$$

$$\frac{du}{dx}(x) = \frac{u(x + h) - u(x)}{h} - \frac{h}{2} \frac{d^2u}{dx^2}(\xi)$$

This is referred to as the **Forward Difference Approximation** of $\frac{du}{dx}(x)$; i.e.,

$$\frac{du}{dx}(x) \approx \frac{u(x + h) - u(x)}{h} \qquad \text{with error of} \quad -\frac{h}{2} \frac{d^2u}{dx^2}(\xi)$$

Since the error is on the order of $h$, the forward difference approximation of $\frac{du}{dx}(x)$ is said to be first-order accurate.

If we start with a slightly different Taylor series expansion, we can derive a backward difference approximation of $\frac{du}{dx}(x)$.

$$u(x - h) = u(x) - h\frac{du}{dx}(x) + \frac{h^2}{2}\frac{d^2u}{dx^2}(\xi) , \quad \xi \in (x-h, x)$$

$$\frac{du}{dx}(x) = \frac{u(x) - u(x - h)}{h} + \frac{h}{2}\frac{d^2u}{dx^2}(\xi)$$

Thus, the **Backward Difference Approximation** of $\frac{du}{dx}(x)$ is

$$\frac{du}{dx}(x) \approx \frac{u(x) - u(x - h)}{h} \qquad \text{with error of} \quad \frac{h}{2}\frac{d^2u}{dx^2}(\xi)$$

Since the error is on the order of $h$, the backward difference approximation of $\frac{du}{dx}(x)$ is said to be first-order accurate.

If we combine two higher-order Taylor series expansions, we can derive a centered difference approximation of $\frac{du}{dx}(x)$.

$$u(x + h) = u(x) + h\frac{du}{dx}(x) + \frac{h^2}{2}\frac{d^2u}{dx^2}(x) + \frac{h^3}{6}\frac{d^3u}{dx^3}(\xi_1) , \quad \xi_1 \in (x, x+h)$$

$$u(x - h) = u(x) - h\frac{du}{dx}(x) + \frac{h^2}{2}\frac{d^2u}{dx^2}(x) - \frac{h^3}{6}\frac{d^3u}{dx^3}(\xi_2) , \quad \xi_2 \in (x-h, x)$$

Subtracting the second equation from the first gives

$$u(x + h) - u(x - h) = 2h\frac{du}{dx}(x) + \frac{h^3}{6}\frac{d^3u}{dx^3}(\xi_1) + \frac{h^3}{6}\frac{d^3u}{dx^3}(\xi_2)$$

$$\frac{du}{dx}(x) = \frac{u(x + h) - u(x - h)}{2h} + \frac{h^2}{3}\frac{d^3u}{dx^3}(\xi_1) + \frac{h^2}{3}\frac{d^3u}{dx^3}(\xi_2)$$

Thus, the **Centered Difference Approximation** of $\frac{du}{dx}(x)$ is

$$\frac{du}{dx}(x) \approx \frac{u(x+h) - u(x-h)}{2h} \qquad \text{with error of} \quad \frac{h^2}{3}\frac{d^3u}{dx^3}(\xi_1) + \frac{h^2}{3}\frac{d^3u}{dx^3}(\xi_2)$$

Since the error is on the order of $h^2$, the backward difference approximation of $\frac{du}{dx}(x)$ is said to be second-order accurate.

If we combine two even higher-order Taylor series expansions, we can derive a centered difference approximation of the second derivative $\frac{d^2u}{dx^2}(x)$.

$$u(x+h) = u(x) + h\frac{du}{dx}(x) + \frac{h^2}{2}\frac{d^2u}{dx^2}(x) + \frac{h^3}{6}\frac{d^3u}{dx^3}(x) + \frac{h^4}{24}\frac{d^4u}{dx^4}(\xi_1) \ , \quad \xi_1 \in (x, x+h)$$

$$u(x-h) = u(x) - h\frac{du}{dx}(x) + \frac{h^2}{2}\frac{d^2u}{dx^2}(x) - \frac{h^3}{6}\frac{d^3u}{dx^3}(x) + \frac{h^4}{24}\frac{d^4u}{dx^4}(\xi_2) \ , \quad \xi_2 \in (x-h, x)$$

Adding the second equation to the first gives

$$u(x+h) + u(x-h) = 2u(x) + h^2\frac{d^2u}{dx^2}(x) + \frac{h^4}{24}\frac{d^4u}{dx^4}(\xi_1) + \frac{h^4}{24}\frac{d^4u}{dx^4}(\xi_2)$$

$$\frac{d^2u}{dx^2}(x) = \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} + \frac{h^2}{24}\frac{d^4u}{dx^4}(\xi_1) + \frac{h^2}{24}\frac{d^4u}{dx^4}$$

Thus, the **Centered Difference Approximation** of $\frac{d^2u}{dx^2}(x)$ is

$$\frac{d^2u}{dx^2}(x) \approx \frac{u(x+h) - 2u(x) + u(x-h)}{h^2} \qquad \text{with error of} \quad \frac{h^2}{24}\frac{d^4u}{dx^4}(\xi_1) + \frac{h^2}{24}\frac{d^4u}{dx^4}$$

Since the error is on the order of $h^2$, the centered difference approximation of $\frac{d^2u}{dx^2}(x)$ is said to be second-order accurate.

Note that this formula for the approximation of $\frac{d^2u}{dx^2}(x)$ can also be derived using the difference of two centered difference approximations of $\frac{du}{dx}(x)$.

$$\frac{du}{dx}\left(x - \frac{h}{2}\right) \approx \frac{u(x) - u(x-h)}{h}$$

$$\frac{du}{dx}\left(x + \frac{h}{2}\right) \approx \frac{u(x+h) - u(x)}{h}$$

$$\frac{d^2u}{dx^2}(x) \approx \frac{\frac{du}{dx}\left(x + \frac{h}{2}\right) - \frac{du}{dx}\left(x - \frac{h}{2}\right)}{h}$$

$$\frac{d^2u}{dx^2}(x) \approx \frac{\left[\frac{u(x+h)-u(x)}{h}\right] - \left[\frac{u(x)-u(x-h)}{h}\right]}{h}$$

$$\frac{d^2u}{dx^2}(x) \approx \frac{u(x+h) - 2\,u(x) + u(x-h)}{h^2}$$

### 2.4.2 Finite Difference Newton's Method for Root Finding

When using Newton's method to find a root of a function, we will need to evaluate the first derivative at a given point. We can use the numerical methods described above to calculate this derivative. In the context of Newton's method, we would choose some $\epsilon$, typically a fairly small value, and then evaluate the derivative at a point $x_i$ using a centered difference approximation as follows:

$$f'(x_i) \equiv f_i' = \frac{f(x_i+\epsilon) - f(x_i-\epsilon)}{2\epsilon}$$

The $\Delta x$ at each iteration would be

$$\Delta x_i = -f_i/f_i'$$
$$\Delta x_i = \frac{2\,\epsilon\,f_i}{f(x_i+\epsilon) - f(x_i-\epsilon)}$$

### 2.4.3 Finite Difference Newton's Method for Minimization

When using Newton's method to find a minimum of a function, we will need to evaluate both the first and second derivatives at a given point. We can use the numerical methods described above to calculate these derivatives. In the context of Newton's method, we would choose some $\epsilon$, typically a fairly small value, and then evaluate the derivatives at a point $x_i$ using centered difference approximations as follows:

$$\begin{aligned} f'(x_i) \equiv f_i' &= \frac{f(x_i+\epsilon) - f(x_i-\epsilon)}{2\epsilon} \\ f''(x_i) \equiv f_i'' &= \frac{f(x_i+\epsilon) - 2\,f(x_i) + f(x_i-\epsilon)}{\epsilon^2} \end{aligned}$$

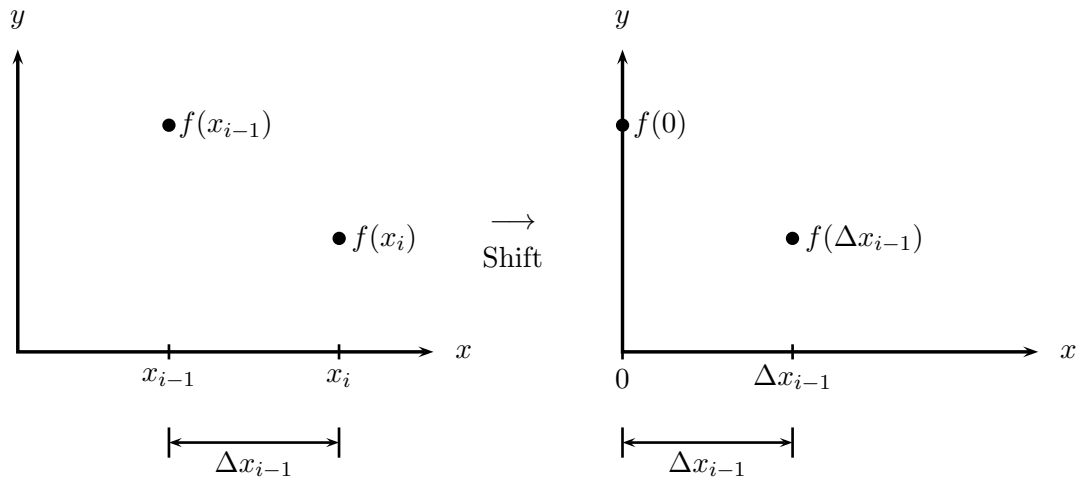
The $\Delta x$ at each iteration would be

$$\begin{aligned} \Delta x_i &= -f_i'/f_i'' \\ \Delta x_i &= \frac{\epsilon\,[f(x_i+\epsilon) - f(x_i-\epsilon)]}{2\,[f(x_i+\epsilon) - 2\,f(x_i) + f(x_i-\epsilon)]} \end{aligned}$$

### 2.4.4 Secant Newton's Method for Root Finding

Finite difference methods can be advantageous in that they allow us to approximate derivatives using only function evaluations. Another such "derivative-free" method is the *secant method*. This is also a way of approximating derivatives using only function evaluations, but instead of functions evaluated at some small $\epsilon$ distance from $x_i$, we use functions evaluated at previous $x_k$ where $k < i$. We can find expressions for these approximations using backward difference formulae.

First consider Newton's method for root-finding. At iteration $i$, we initially have $x_i$ and the information from the previous iteration; i.e., $x_{i-1}$, $\Delta x_{i-1}$, and $f(x_{i-1})$. We can also calculate $f(x_i)$. We need to estimate $f'(x_i)$ using this available information. We can do so as follows:




We want to find an expression for $f'(x_i)$ as a function of $f(x_{i-1})$ and $f(x_i)$

$$f'(x_i) \;=\; a\, f(x_{i-1}) \;+\; b\, f(x_i)$$

which is equivalent to

$$f'(\Delta x_{i-1}) \;=\; a\, f(0) \;+\; b\, f(\Delta x_{i-1})$$

To solve for $a$ and $b$, we can enforce the condition that the approximation be exact for polynomials up to degree 1. Therefore, it must be exact for the functions $f(x) = 1$ and $f(x) = x$.

$$\begin{aligned} f(x) = 1: \qquad f'(\Delta x_{i-1}) &= a\, f(0) \;+\; b\, f(\Delta x_{i-1}) \\ 0 &= a\,(1) \;+\; b\,(1) \\ 0 &= a \;+\; b \end{aligned} \tag{5}$$

$$\begin{aligned} f(x) = x: \qquad f'(\Delta x_{i-1}) &= a\, f(0) \;+\; b\, f(\Delta x_{i-1}) \\ 1 &= a\,(0) \;+\; b\,(\Delta x_{i-1}) \\ 1 &= b\,(\Delta x_{i-1}) \\ b &= \frac{1}{\Delta x_{i-1}} \end{aligned} \tag{6}$$

Substituting Equation (6) into Equations (5) gives

$$a \;=\; \frac{-1}{\Delta x_{i-1}}$$

Therefore

$$f'(\Delta x_{i-1}) \;=\; \left(\frac{-1}{\Delta x_{i-1}}\right) f(0) \;+\; \left(\frac{1}{\Delta x_{i-1}}\right) f(\Delta x_{i-1})$$

$$=\; \frac{f(\Delta x_{i-1}) \;-\; f(0)}{\Delta x_{i-1}}$$

$$f'(x_i) \;=\; \frac{f(x_i) \;-\; f(x_{i-1})}{\Delta x_{i-1}}$$

This results seems fairly obvious, since it is simply the slope of the secant line formed by the points $f(x_i)$ and $f(x_{i-1})$. But this procedure will be useful when considering higher order approximations.

We can now construct a formula for the Newton root-finding iteration when using the secant approximation for the derivative.

$$x_{i+1} \;=\; x_i \;-\; \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} \;=\; x_i \;-\; \frac{f(x_i)}{\frac{f(x_i) - f(x_{i-1})}{\Delta x_{i-1}}} \;=\; x_i \;-\; \frac{\Delta x_{i-1}\, f(x_i)}{f(x_i) \;-\; f(x_{i-1})}$$

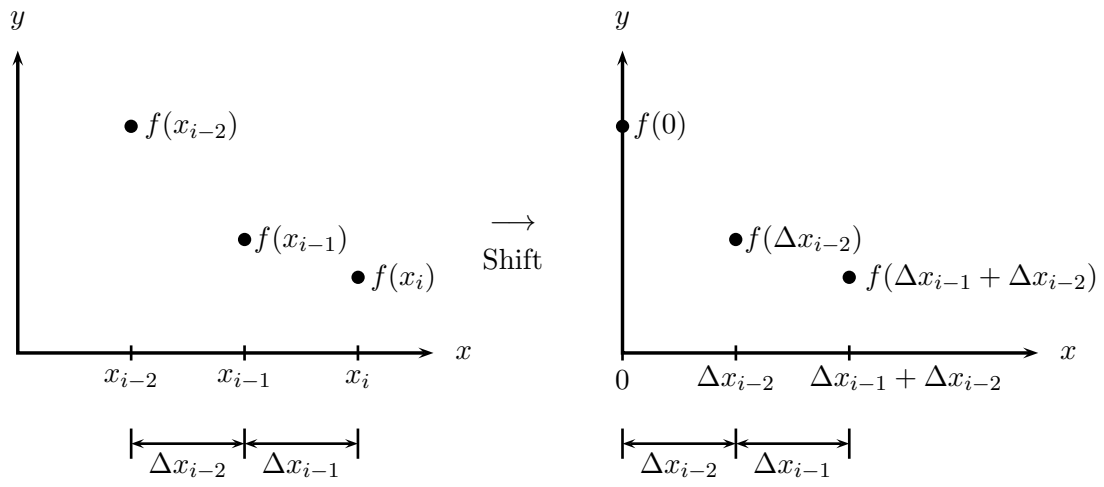
or

$$x_{i+1} \;=\; x_i \;+\; \Delta x_i$$

$$\text{where} \qquad \Delta x_i \;=\; -\Delta x_{i-1}\left[\frac{f(x_i)}{f(x_i) \;-\; f(x_{i-1})}\right] \;=\; \Delta x_{i-1}\left[\frac{f(x_i)}{f(x_{i-1}) \;-\; f(x_i)}\right]$$

### 2.4.5 Secant Newton's Method for Minimization

Now consider Newton's method for minimization. At iteration $i$, we initially have $x_i$ and the information from the previous two iterations; i.e., $x_{i-1}$, $\Delta x_{i-1}$, $f(x_{i-1})$, $x_{i-2}$, $\Delta x_{i-2}$, $f(x_{i-2})$. We can also calculate $f(x_i)$. We need to estimate $f'(x_i)$ and $f''(x_i)$ using this available information. We can do so as follows:




We want to find expressions for $f'(x_i)$ and $f''(x_i)$ as a functions of $f(x_{i-2})$, $f(x_{i-1})$ and $f(x_i)$

$$\begin{aligned} f'(x_i) &= a\, f(x_{i-2}) + b\, f(x_{i-1}) + c\, f(x_i) \\ f''(x_i) &= d\, f(x_{i-2}) + e\, f(x_{i-1}) + g\, f(x_i) \end{aligned}$$

which are equivalent to

$$\begin{aligned} f'(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ f''(\Delta x_{i-1} + \Delta x_{i-2}) &= d\, f(0) + e\, f(\Delta x_{i-2}) + g\, f(\Delta x_{i-1} + \Delta x_{i-2}) \end{aligned}$$

To solve for $a$, $b$, and $c$, we can enforce the condition that the approximation be exact for polynomials up to degree 2. Therefore, it must be exact for the functions $f(x) = 1$, $f(x) = x$, and $f(x) = x^2$.

$$f(x) = 1: \qquad \begin{aligned} f'(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 0 &= a\,(1) + b\,(1) + c\,(1) \\ 0 &= a + b + c \end{aligned} \tag{7}$$

$$f(x) = x: \qquad \begin{aligned} f'(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 1 &= a\,(0) + b\,(\Delta x_{i-2}) + c\,(\Delta x_{i-1} + \Delta x_{i-2}) \\ 1 &= b\,(\Delta x_{i-2}) + c\,(\Delta x_{i-1} + \Delta x_{i-2}) \end{aligned} \tag{8}$$

$$f(x) = x^2: \qquad \begin{aligned} f'(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 2\,(\Delta x_{i-1} + \Delta x_{i-2}) &= a\,(0) + b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 \\ 2\,(\Delta x_{i-1} + \Delta x_{i-2}) &= b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 \end{aligned} \tag{9}$$

Considering Equation (8) multiplied by $\Delta x_{i-2}$, and Equation (9) gives

$$\begin{aligned} b\,(\Delta x_{i-2})^2 + c\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2}) &= \Delta x_{i-2} \\ b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 &= 2\,(\Delta x_{i-1} + \Delta x_{i-2}) \end{aligned}$$

---

$$\begin{aligned} c\left(\Delta x_{i-1}^2 + 2\,\Delta x_{i-1}\,\Delta x_{i-2} + \Delta x_{i-2}^2 - \Delta x_{i-1}\,\Delta x_{i-2} - \Delta x_{i-2}^2\right) &= 2\,\Delta x_{i-1} + \Delta x_{i-2} \\ c\,\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2}) &= 2\,\Delta x_{i-1} + \Delta x_{i-2} \\ c &= \frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})} \end{aligned}$$

From Equation (8)

$$\begin{aligned} b &= \frac{1 - c\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-2}} = \frac{1 - \frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}(\Delta x_{i-1} + \Delta x_{i-2})}\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-2}} \\ &= \frac{1 - \frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}}}{\Delta x_{i-2}} = \frac{\Delta x_{i-1} - (2\,\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-1}\,\Delta x_{i-2}} = \frac{-\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-1}\,\Delta x_{i-2}} \end{aligned}$$

From Equation (7)

$$a = -b - c = \frac{\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,\Delta x_{i-2}} - \frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

$$= \frac{(\Delta x_{i-1} + \Delta x_{i-2})\,(\Delta x_{i-1} + \Delta x_{i-2}) - \Delta x_{i-2}\,(2\,\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-1}\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

$$= \frac{\Delta x_{i-1}^2 + 2\,\Delta x_{i-1}\,\Delta x_{i-2} + \Delta x_{i-2}^2 - 2\,\Delta x_{i-1}\,\Delta x_{i-2} - \Delta x_{i-2}^2}{\Delta x_{i-1}\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

$$= \frac{\Delta x_{i-1}}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

Therefore

$$f'(\Delta x_{i-1} + \Delta x_{i-2}) = \left[\frac{\Delta x_{i-1}}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(0) - \left[\frac{\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,\Delta x_{i-2}}\right] f(\Delta x_{i-2}) + \left[\frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(\Delta x_{i-1} + \Delta x_{i-2})$$

$$f'(x_i) = \left[\frac{\Delta x_{i-1}}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(x_{i-2}) - \left[\frac{\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,\Delta x_{i-2}}\right] f(x_{i-1}) + \left[\frac{2\,\Delta x_{i-1} + \Delta x_{i-2}}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(x_i)$$

We can now do the same for $f''(x_i)$

$$f''(x_i) = d\,f(x_{i-2}) + e\,f(x_{i-1}) + g\,f(x_i)$$
$$f''(\Delta x_{i-1} + \Delta x_{i-2}) = d\,f(0) + e\,f(\Delta x_{i-2}) + g\,f(\Delta x_{i-1} + \Delta x_{i-2})$$

To solve for $d$, $e$, and $g$, we again enforce the condition that the approximation be exact for polynomials up to degree 2. Therefore, it must be exact for the functions $f(x) = 1$, $f(x) = x$, and $f(x) = x^2$.

$$f(x) = 1: \qquad \begin{aligned} f''(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 0 &= a\,(1) + b\,(1) + c\,(1) \\ 0 &= a + b + c \end{aligned} \tag{10}$$

$$f(x) = x: \qquad \begin{aligned} f''(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 0 &= a\,(0) + b\,(\Delta x_{i-2}) + c\,(\Delta x_{i-1} + \Delta x_{i-2}) \\ 0 &= b\,(\Delta x_{i-2}) + c\,(\Delta x_{i-1} + \Delta x_{i-2}) \end{aligned} \tag{11}$$

$$f(x) = x^2: \qquad \begin{aligned} f''(\Delta x_{i-1} + \Delta x_{i-2}) &= a\, f(0) + b\, f(\Delta x_{i-2}) + c\, f(\Delta x_{i-1} + \Delta x_{i-2}) \\ 2 &= a\,(0) + b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 \\ 2 &= b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 \end{aligned} \tag{12}$$

Considering Equation (11) multiplied by $\Delta x_{i-2}$, and Equation (12) gives

$$\begin{aligned} b\,(\Delta x_{i-2})^2 + c\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2}) &= 0 \\ b\,(\Delta x_{i-2})^2 + c\,(\Delta x_{i-1} + \Delta x_{i-2})^2 &= 2 \end{aligned}$$

---

$$\begin{aligned} c\left(\Delta x_{i-1}^2 + 2\,\Delta x_{i-1}\,\Delta x_{i-2} + \Delta x_{i-2}^2 - \Delta x_{i-1}\,\Delta x_{i-2} - \Delta x_{i-2}^2\right) &= 2 \\ c\,\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2}) &= 2 \\ c &= \frac{2}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})} \end{aligned}$$

From Equation (11)

$$b = \frac{-c\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-2}} = \frac{\frac{-2}{\Delta x_{i-1}(\Delta x_{i-1} + \Delta x_{i-2})}\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-2}} = \frac{-2}{\Delta x_{i-1}\,\Delta x_{i-2}}$$

From Equation (10)

$$a \;=\; -b \;-\; c \;=\; \frac{2}{\Delta x_{i-1}\,\Delta x_{i-2}} \;-\; \frac{2}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

$$=\; \frac{2\,(\Delta x_{i-1} \;+\; \Delta x_{i-2}) \;-\; 2\,\Delta x_{i-2}}{\Delta x_{i-1}\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})} \;=\; \frac{2}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

Therefore

$$f''(\Delta x_{i-1} + \Delta x_{i-2}) \;=\; \left[\frac{2}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(0) \;-\; \left[\frac{2}{\Delta x_{i-1}\,\Delta x_{i-2}}\right] f(\Delta x_{i-2}) + \left[\frac{2}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(\Delta x_{i-1} + \Delta x_{i-2})$$

$$f''(x_i) \;=\; \left[\frac{2}{\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(x_{i-2}) \;-\; \left[\frac{2}{\Delta x_{i-1}\,\Delta x_{i-2}}\right] f(x_{i-1}) + \left[\frac{2}{\Delta x_{i-1}\,(\Delta x_{i-1} + \Delta x_{i-2})}\right] f(x_i)$$

We can now construct a formula for the Newton minimization iteration when using the secant approximation for the derivatives.

$$x_{i+1} \;=\; x_i \;-\; \frac{f'(x_i)}{f''(x_i)} \;=\; x_i \;+\; \Delta x_i$$

where

$$\Delta x_i \;=\; \frac{-f'(x_i)}{f''(x_i)} \;=\; \frac{-f'(x_i)}{f''(x_i)} \cdot \frac{\Delta x_{i-1}\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}{\Delta x_{i-1}\,\Delta x_{i-2}\,(\Delta x_{i-1} + \Delta x_{i-2})}$$

$$=\; \frac{-\left[\Delta x_{i-1}^2\right] f(x_{i-2}) \;+\; \left[(\Delta x_{i-1} + \Delta x_{i-2})^2\right] f(x_{i-1}) \;-\; \left[\Delta x_{i-2}\,(2\,\Delta x_{i-1} + \Delta x_{i-2})\right] f(x_i)}{[2\,\Delta x_{i-1}]\,f(x_{i-2}) \;-\; [2\,(\Delta x_{i-1} + \Delta x_{i-2})]\,f(x_{i-1}) \;+\; [2\,\Delta x_{i-2}]\,f(x_i)}$$

The following is pseudocode to describe the secant Newton's method for finding a minimum of the function $f(x)$, where $f : \Re \to \Re$.

INPUT: Functions $f(x)$, $f'(x)$, and $f''(x)$, starting value $x_0$, tolerance $tol$, maximum iterations $maxniter$

OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = newtonSec($x_0, tol, maxniter$)

$iter = 1$

$x = x_0$

$xprev = x$

$fcurr = f(x)$

$fprev = fcurr$

$\Delta x = 0.0$

$\Delta x_{prev} = \Delta x$

WHILE $iter \leq maxniter$

$\Delta x_{2prev} = \Delta x_{prev}$

$\Delta x_{prev} = \Delta x$

IF $iter < 3$

$\Delta x \ = \ -f'(x)/f''(x)$

ELSE

$numer = \Delta x_{prev}^2 \, f2prev - (\Delta x_{prev} + \Delta x_{2prev})^2 \, fprev$
$+\Delta x_{2prev}(2 \, \Delta x_{prev} + \Delta x_{2prev}) fcurr$

$denom = 2\,\Delta x_{prev}\, f2prev - 2\,(\Delta x_{prev} + \Delta x_{2prev})\, fprev + 2\,\Delta x_{2prev}\, fcurr$

$\Delta x \ = \ -numer/denom$

$xprev = x$

$x \ = \ x \ + \ \Delta x$

IF $x$ unacceptable

Line search to calculate $\lambda$

$x = x + \lambda \Delta x$

IF abs$(x - xprev) < tol$

break

$f2prev = fprev$

$fprev = fcurr$

$fcurr = f(x)$

$iter = iter + 1$

$x_{min} = x$

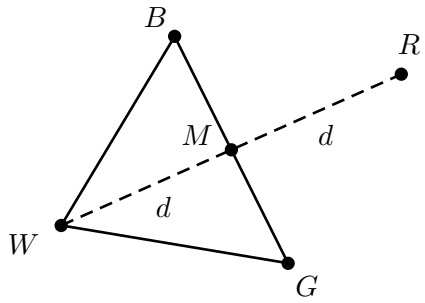
$f_{min} = f(x)$

$niters = iter$

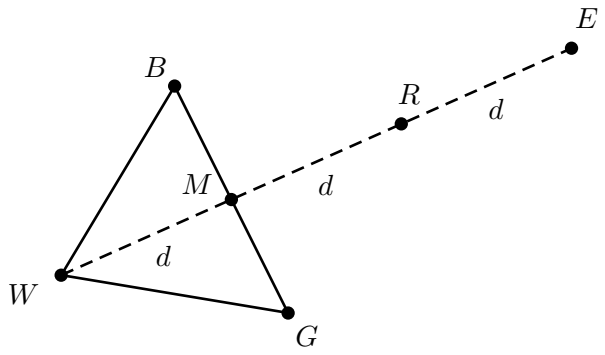
# 3 Simplex Methods: Nelder-Mead Algorithm

The Nelder-Mead algorithm is a *simplex method* for finding a local minimum of several variables. We will consider the problem of two independent variables. For two variables, a simplex is a triangle, and the method is a pattern search that compares function values at the three vertices of a triangle. The worst vertex, where $f(x, y)$ is largest, is rejected and replaced with a new vertex. A new triangle is thus formed and the search is continued. The process generates a sequence of triangles (which might have different shapes), for which the function values at the vertices get smaller and smaller. The size of the triangles is reduced and the coordinates of the minimum point are found.

Let $f(x, y)$ be the function that is to be minimized. To start, we are given three vertices of a triangle. The function is then evaluated at each of the three points. We will use the notation $B$, $G$, and $W$ to denote the best, good (next to best), and worst points, respectively, in terms of their function values ($f_B \leq f_G \leq f_W$). The first move will be to try to reflect the worst point through the side formed by the other two points. Let this new **reflection** point be $R$. If $M$ is the midpoint between $B$ and $G$, then $R = M + (M - W) = 2\,M - W$.

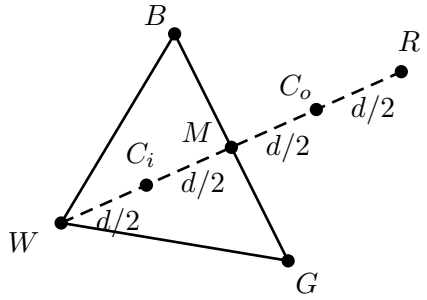



If the function value at $R$ is smaller than the function value at $W$, then we have moved in the correct direction toward the minimum. If the function value at $R$ is less than the function value at $G$ but greater than the function value at $B$, then we change $G$ to $W$ and $R$ to $G$. If the function value at $R$ is less than the function value at $B$, then we might not have moved far enough in that direction and we create an **expansion** point $E$, with the distance $d$ between $M$ and $R$ being the same as the distance between $R$ and $E$, or $E = R + (R - M) = 2\,R - M$.

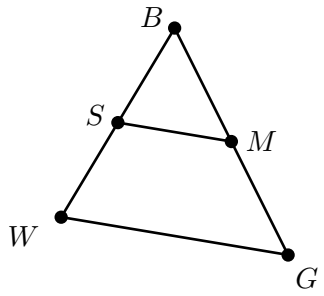



If the function value at $E$ is less than the function value at $R$, we change $G$ to $W$, $B$ to $G$, and $E$ to $B$ (accept $E$). Otherwise we change $G$ to $W$, $B$ to $G$, and $R$ to $B$ (accept $R$).

If the function value at $R$ is greater than the function value at $G$, then we examine the inner and outer **contraction** points $C_i$ and $C_o$.




Choose point $C$ to be the lower function value of $C_i$ and $C_o$. If the function value at $C$ is less than the function value at $G$, then we accept $C$ (and assign new W, G, B based on function values). If the function value at $C$ is greater than the function value at $G$, then we **shrink** the simplex towards $B$, forming the new triangle $S - M - B$, where $S$ is the midpoint between $W$ and $B$.




The following is pseudocode to describe the Nelder-Mead simplex algorithm.

INPUT: Function $f(\vec{x}) : \Re^2 \to \Re$, 3 points (vectors) $(x_1, y_1)$, $(x_2, y_2)$, and $(x_3, y_3)$, tolerance $tol$

OUTPUT: Value which differs from a local minimum, $f_{min}$, of $f(\vec{x})$ by less than $tol$, and the $\vec{x}$ value, $\vec{x}_{min}$, at that local minimum.

function $[\vec{x}_{min},\ f_{min}]$ = nelderMead($(x_1, y_1)$, $(x_2, y_2)$, $(x_3, y_3)$, $tol$)

Evaluate $f(x_1, y_1)$, $f(x_2, y_2)$, $f(x_3, y_3)$

Define lowest $f$ value to be $f_B$, middle to be $f_G$, and highest to be $f_W$

Define $\vec{x}_B$ to be the $\vec{x}$ corresponding to $f_B$, $\vec{x}_G$ to be the $\vec{x}$ corresponding to $f_G$, and $\vec{x}_W$ to be the $\vec{x}$ corresponding to $f_W$

$\bar{f} = \frac{1}{3}\left(f_W + f_G + f_B\right)$

$error = \sqrt{\frac{1}{3}\left[\left(f_W - \bar{f}\right)^2 + \left(f_G - \bar{f}\right)^2 + \left(f_B - \bar{f}\right)^2\right]}$

WHILE $error > tol$

$M$ = midpoint between $B$ and $G$

$R = 2\,M - W$ (reflection point)

Evaluate $f_R$

IF $f_B < f_R < f_G$ (accept $f_R$)

$\vec{x}_W = \vec{x}_G$, $f_W = f_G$

$\vec{x}_G = \vec{x}_R$, $f_G = f_R$

ELSE

IF $f_R < f_B$

$E = 2\,R - M$ (expansion point)

IF $f_E < f_R$ (accept $f_E$)

$\vec{x}_W = \vec{x}_G$, $f_W = f_G$

$\vec{x}_G = \vec{x}_B$, $f_G = f_B$
$\vec{x}_B = \vec{x}_E$, $f_B = f_E$
ELSE (accept $f_R$)
$\vec{x}_W = \vec{x}_G$, $f_W = f_G$
$\vec{x}_G = \vec{x}_B$, $f_G = f_B$
$\vec{x}_B = \vec{x}_R$, $f_B = f_R$
ELSE (will have $f_G < f_R$)
$C_i = \frac{1}{2}(M + W)$ (inner contraction point)
Evaluate $f_{C_i}$

$C_o = \frac{1}{2}(M + R)$ (outer contraction point)
Evaluate $f_{C_o}$
IF $f_{C_i} < f_{C_o}$
$C = C_i$
$f_C = f_{C_i}$
ELSE
$C = C_o$
$f_C = f_{C_o}$
IF $f_C < f_G$ (accept $f_C$)
$B$, $G$, and $C$ form new triangle, assign new $W$, $G$, and $B$
ELSE (shrink triangle)
$S$ = midpoint between $W$ and $B$
$S$, $M$, and $B$ form new triangle, assign new $W$, $G$, and $B$

$$error = \sqrt{\frac{1}{3}\left[\left(f_W - \bar{f}\right)^2 + \left(f_G - \bar{f}\right)^2 + \left(f_B - \bar{f}\right)^2\right]}$$

(this *error* calculation is in the WHILE loop but outside of all the IF/ELSE statements)

# 4 Direction Set Methods

We have seen methods for minimizing a function of one variable. If we start at a point $x^k$ in $n$-dimensional space and proceed from there in some direction $p^k$, then any function of $n$ variables $f(x)$ can be minimized along the line $p^k$ by one of our 1D minimization methods. We can then try to construct multidimensional minimization methods that consist of sequences of these line minimizations. Different methods will differ only by how they choose the next direction $p^k$ to try.

## 4.1 1D Minimizer

### 4.1.1 Quadratic Form

For a function that is of quadratic form, we can derive an exact expression for and thus move directly to the 1D minimizer along a search direction, $p_k$.

Consider moving from the point $x_k$ to the point $x_{k+1}$ by taking the step $x_{k+1} = x_k + \lambda_k p_k$. We can express the function value at $x_{k+1}$ using a three-term Taylor series. Because the function is quadratic, this expression is exact.

$$f(x_{k+1}) \;=\; f(x_k + \lambda_k\, p_k) \;=\; f(x_k) \;+\; \lambda_k\, \nabla f(x_k)^T p_k \;+\; \frac{1}{2}\, \lambda_k^2\, p_k^T\, \nabla^2 f(x_k)\, p_k$$

We can find the $\lambda_k$ that minimizes $f(x_{k+1})$ by setting $\frac{\partial f(x_{k+1})}{\partial \lambda_k} = 0$ and solving for $\lambda_k$.

$$\frac{\partial f(x_{k+1})}{\partial \lambda_k} \;=\; \nabla f(x_k)^T p_k \;+\; \lambda_k\, p_k^T\, \nabla^2 f(x_k)\, p_k \;=\; 0$$

$$\lambda_k \;=\; \frac{-\nabla f(x_k)^T\, p_k}{p_k^T\, \nabla^2 f(x_k)\, p_k} \tag{13}$$

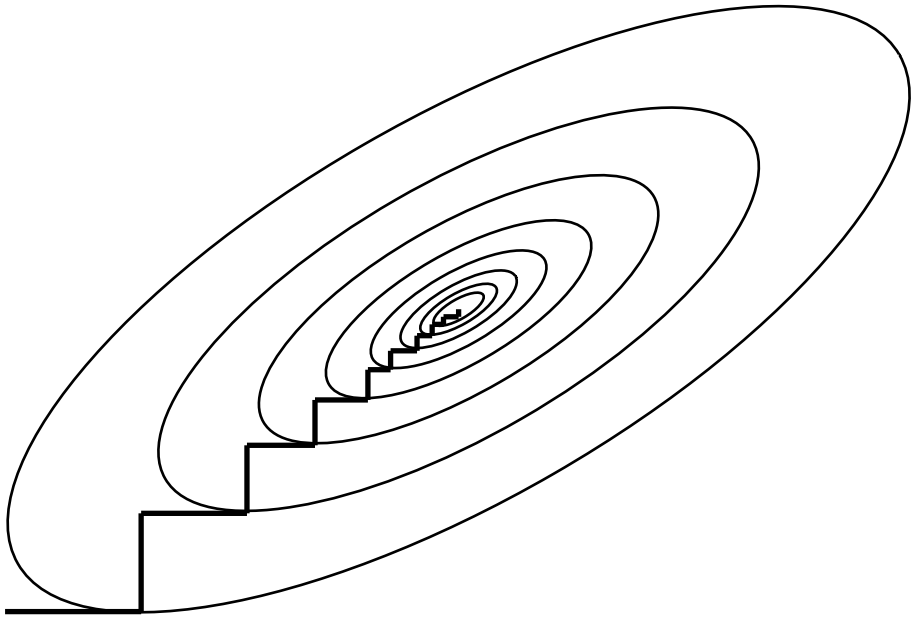
### 4.1.2 General Nonlinear Form

For general convex or general nonlinear functions $f$, we have to find $\lambda_k$ in a way that ensures that our $p_k$'s are descent directions. The conditions that we imposed on our backtracking line search will ensure this. Therefore, we will use the backtracking line search (Section 2.3) to determine $\lambda_k$. This search will not necessarily move to the minimum function value along the search direction, but it will move us to a point that is acceptable in terms of both function value and maintaining descent directions.

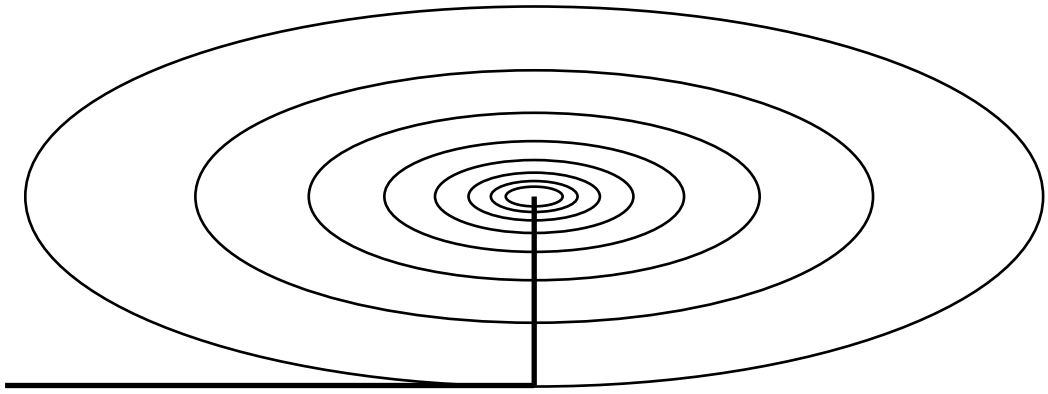
## 4.2 Alternating Variable Method

A simple method for choosing the set of directions is to use the unit vectors along the coordinate directions. This is called the *alternating variable* method. While the alternating variable method may be sufficient for some functions, it is not very efficient for most

functions that we will encounter. Why is the alternating variable method not always very efficient? Consider the case of a long, narrow valley, as shown in the following figure. The ellipses represent the contour lines of $f(x, y)$, and the straight lines show the search lines along the coordinate directions. Each search line starts off in the local gradient direction, perpendicular to the contour lines, and traverses a straight line until a local minimum is reached, where the traverse if parallel to the local contour lines (we will prove these later).



This is not a very efficient search, unless the valley is optimally oriented along coordinate directions, as in the following figure. In that case, we reach the minimum in two steps.



The following is pseudocode to describe the Alternating Variable method.

iter INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniters$
OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = alternateVar($n$,$x_0$,$tol$,$maxniters$)
    $x = x_0$
    $iter = 0$
    WHILE $iter \leq maxniters$
        $p$ = zeros($n$)
        $p[iter\%n] = 1$
        Find $\lambda$ to minimize $f(x + \lambda\, p)$  (see Section 4.1)
        $x = x + \lambda\, p$
        $iter = iter + 1$
        IF $\|\nabla f(x)\| < tol$
            break
    $x_{min} = x$
    $f_{min} = f(x)$
    $niters = iter$

## 4.3 Directional Derivatives and Gradients

iter

First consider $f : \Re^2 \to \Re$, or $f(x, y)$. $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$ represent derivatives in the direction of the $x$ and $y$ axes, respectively. What about derivatives in some arbitrary direction that is not along a coordinate axis? The derivative of $f(x, y)$ in the direction of the vector $p = (p_x, p_y)$ is called the *directional derivative* and is denoted by

$$\frac{\partial f}{\partial p}(x, y) = \lim_{h \to 0} \frac{f(x + h\, p_x, y + h\, p_y) - f(x, y)}{h}$$

We wish to derive a formula for $\frac{\partial f}{\partial p}(x,y)$ in terms of the gradient of $f$, where gradient of $f$ is defined as

$$\nabla f \;=\; \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \;\equiv\; \begin{bmatrix} f_x \\ f_y \end{bmatrix}$$

To do so, we define a new function of a single variable

$$g(z) = f(x_0 + z\,p_x, y_0 + z\,p_y)$$

where $x_0, y_0, p_x, p_y$ are some fixed numbers.

By the definition of a derivative for functions of a single variable

$$\begin{aligned} g'(z) \;&=\; \lim_{h\to 0} \frac{g(z+h) - g(z)}{h} \\ g'(0) \;&=\; \lim_{h\to 0} \frac{g(h) - g(0)}{h} \\ g'(0) \;&=\; \lim_{h\to 0} \frac{f(x_0 + h\,p_x, y_0 + h\,p_y) - f(x_0,y_0)}{h} \;=\; \frac{\partial f}{\partial p}(x_0,y_0) \end{aligned} \tag{14}$$

Now consider this from a different perspective. $g(z) = f(x,y)$ where $x = x_0 + z\,p_x$, $y = y_0 + z\,p_y$. Using the chain rule, we get

$$\begin{aligned} g'(z) \;&=\; \frac{dg}{dz} \;=\; \frac{\partial f}{\partial x}\frac{\partial x}{\partial z} \;+\; \frac{\partial f}{\partial y}\frac{\partial y}{\partial z} \;=\; f_x(x,y)\,p_x \;+\; f_y(x,y)\,p_y \\ g'(0) \;&=\; f_x(x_0,y_0)\,p_x \;+\; f_y(x_0,y_0)\,p_y \end{aligned} \tag{15}$$

Equating (14) and (15) gives

$$\begin{aligned} \frac{\partial f}{\partial p}(x_0,y_0) \;&=\; f_x(x_0,y_0)\,p_x \;+\; f_y(x_0,y_0)\,p_y \;=\; \nabla f(x_0,y_0)^T\,p \;=\; \nabla f(x_0,y_0)\cdot p \\ \frac{\partial f}{\partial p}(x,y) \;&=\; f_x(x,y)\,p_x \;+\; f_y(x,y)\,p_y \;=\; \nabla f(x,y)^T\,p \;=\; \nabla f(x,y)\cdot p \end{aligned}$$

Extending to $n$ dimensions gives

$$\frac{\partial f}{\partial p}(x) \;=\; \nabla f(x)^T\, p \;=\; \nabla f(x) \cdot p \tag{16}$$

where $x \in \Re^n$, $p \in \Re^n$, $f : \Re^n \to \Re$

Similarly for second directional derivatives

$$\frac{\partial^2 f}{\partial p^2}(x) \;=\; p^{\,T}\, \nabla^2 f(x)^T\, p$$

where $\nabla^2 f(x)$ is the $n \times n$ matrix of second derivatives.

## 4.4 Steepest Descent Method

Assume that $f : \Re^n \to \Re$. From Equation (16) we see that $\frac{\partial f}{\partial p}(x) = \nabla f(x) \cdot p = \|\nabla f(x)\|\, \|p\| \cos\theta$, where $\theta$ is the angle between $\nabla f(x)$ and $p$. Therefore, $\frac{\partial f}{\partial p}(x)$ is maximized when $\theta = 0$. This says that the direction of greatest increase in $f$ is the gradient of $f$. In order to minimize a function, we want to move in the direction of greatest decrease in $f$, or $-\nabla f$.

The steepest descent method chooses its direction from $x_k$ ($x_k \in \Re^n$) to be the opposite of the gradient vector at that point, or $-\nabla f(x_k)$. If $\vec{p}$ is the direction tangent to a contour line, the function value does not change, or $\frac{\partial f}{\partial p}(x) = 0$. $\frac{\partial f}{\partial p}(x)$ is 0 when cos $\theta$ is 90°. Therefore, the steepest descent direction is orthogonal to the contours of the function. Also, any direction that makes an angle of $< \pi/2$ with $-\nabla f(x_k)$ is a *descent direction*. Any descent direction is guaranteed to produce a decrease in $f$, provided that the step length is sufficiently small.

The steepest descent method suffers from the same type of inefficiency we saw with the alternating variable method; i.e., it may perform many small steps in going down a long narrow valley, even if the valley is a perfect quadratic form. The reason for this is that the new gradient at the minimum point of any line minimization is $\perp$ to the direction just traversed. With the steepest descent method, you make right angle turns which does not, in general, take you to the minimum.

The following is pseudocode to describe the Steepest Descent method.

INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniters$
OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = steepestDesc($n$,$x_0$,$tol$,$maxniters$)
    $x = x_0$
    $iter = 0$
    WHILE $iter \leq maxniters$
        $p = -\nabla f(x)$
        Find $\lambda$ so that $f(x + \lambda\, p)$ is acceptable
        $x = x + \lambda\, p$
        $iter = iter + 1$
        IF $\|\nabla f(x)\| < tol$
            break
    $x_{min} = x$
    $f_{min} = f(x)$
    $niters = iter$

## 4.5 Conjugate Directions

Suppose we have moved along some direction $p_k$ to a minimum and propose to move along some new direction $p_{k+1}$. The condition that motion along $p_{k+1}$ not spoil the minimum along $p_k$ is just that the change in gradient stay $\perp$ to $p_k$.

$$0 = p_k\, \delta(\nabla f) = p_k\, \nabla^2 f(x_k)\, p_{k+1}$$

When this condition holds for two vectors,they are said to be *conjugate*. When it holds for all members of a set of vectors, they are said to be a *conjugate set*. If you do successive line minimizations along a conjugate set of directions, then you will not have to redo any of those directions.

There is a simple interpretation of the properties of conjugate directions. Assume that $f$ is quadratic form. Therefore,

$$f(x) \;=\; f(0) \;+\; \nabla f(x)^T x \;+\; \frac{1}{2}\, x^{\,T}\, \nabla^2 f(x)\, x$$

If the matrix $\nabla^2 f(x)$ is diagonal, the contours of $f(x)$ are ellipses whose axes are aligned with the coordinate axes. We can find the minimizer of this function by performing 1D minimizations along the coordinate directions. Successive minimizations along coordinate directions will find the minimum in $n$ iterations (as we demonstrated for the Alternating Variable method).

When $\nabla^2 f(x)$ is not diagonal, the contours of $f(x)$ are still elliptical but usually not aligned with the coordinate directions. Successive minimization along coordinate directions no longer leads to the solution in $n$ iterations.

We can recover the nice behavior of the aligned case by transforming the problem to make $\nabla^2 f$ diagonal and then minimizing along the transformed coordinate directions. We can transform the problem by defining a new variable $\hat{x}$ as $\hat{x} \;=\; S^{-1}\, x$, where $S$ is the $n \times n$ matrix defined by $S \;=\; [p_1 \quad p_2 \quad \cdots \quad p_n]$ with $\{p_1, p_2, \cdots, p_n\}$ being a set of conjugate directions w.r.t. $\nabla^2 f$ .

The quadratic $f$ now becomes

$$\begin{aligned} f(x) \;&=\; f(S\,\hat{x}) \;=\; f(0) \;-\; \nabla f(S\,\hat{x})^T \hat{x} \;+\; \frac{1}{2}\,(S\,\hat{x})^{\,T}\; \nabla^2 f(\hat{x})\,(S\,\hat{x}) \\ f(\hat{x}) \;&=\; f(0) \;-\; \nabla f(S\,\hat{x})^T \hat{x} \;+\; \frac{1}{2}\,\hat{x}^T \left(S^T\, \nabla^2 f(\hat{x})\, S\right) \hat{x} \end{aligned}$$

Looking at the third term

$$S^T\, \nabla^2 f(\hat{x})\, S \;=\; \begin{bmatrix} — & p_1 & — \\ & \vdots & \\ — & p_n & — \end{bmatrix} \left[\nabla^2 f(\hat{x})\right] \begin{bmatrix} | & & | \\ p_1 & \cdots & p_n \\ | & & | \end{bmatrix}$$

$$S^T \nabla^2 f(\hat{x})\, S = \begin{bmatrix} p_1^T \nabla^2 f(\hat{x})\, p_1 & p_1^T \nabla^2 f(\hat{x})\, p_2 & \cdots & p_1^T \nabla^2 f(\hat{x})\, p_n \\ \vdots & & & \vdots \\ p_n^T \nabla^2 f(\hat{x})\, p_1 & p_n^T \nabla^2 f(\hat{x})\, p_2 & \cdots & p_n^T \nabla^2 f(\hat{x})\, p_n \end{bmatrix}$$

$$= \begin{bmatrix} p_1^T \nabla^2 f(\hat{x})\, p_1 & 0 & \cdots & 0 \\ \vdots & & & \vdots \\ 0 & 0 & \cdots & p_n^T \nabla^2 f(\hat{x})\, p_n \end{bmatrix} \quad \begin{array}{l} \text{because of conjugacy} \\ p_i^T \nabla^2 f(\hat{x})\, p_j \,, i \neq j \end{array}$$

Therefore, $S^T \nabla^2 f(\hat{x})\, S$ is a diagonal matrix and $f(\hat{x})$ is a quadratic form aligned with the coordinate $\hat{x}$ axes. We can find the minimum of $f(\hat{x})$ by performing $n$ 1D minimizations along the coordinate directions of $\hat{x}$.

What are the coordinate directions of $\hat{x}$ in $x-$space?

$$\text{if } \hat{x} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \Rightarrow \quad \vec{x} = S\,\hat{x} = \begin{bmatrix} | & & | \\ p_1 & \cdots & p_n \\ | & & | \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = p_1$$

$$\text{if } \hat{x} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \Rightarrow \quad \vec{x} = S\,\hat{x} = \begin{bmatrix} | & & | \\ p_1 & \cdots & p_n \\ | & & | \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} = p_2$$

$$\vdots$$

Therefore, the coordinate directions of $\hat{x}$ are simply the $p_i$ directions in $x-$space.

Therefore, we can find the minimum by performing $n$ 1D minimizations along the $n$ $p_i$ conjugate directions, and it will terminate in at most $n$ steps.

A goal for a direction set method is to come up with a set of $n$ linearly independent mutually conjugate directions, then one pass of $n$ line minimizations will put it exactly at the minimum of a quadratic form. For functions that are not exactly quadratic forms, we won't be exactly at the minimum, but repeated cycles of the $n$ line minimizations will in due course converge quadratically to the minimum.

There are several ways to choose such a conjugate set of vectors; e.g.,

1. The eigenvectors of $\nabla^2 f$ are mutually orthogonal as well as conjugate w.r.t. $\nabla^2 f$. But this is not practical to compute for large-scale applications.

2. Could modify the Gram-Schmidt process to produce a set of conjugate directions rather than a set of orthogonal directions. But this is expensive and requires the storage of the entire direction set.

3. The **Conjugate Gradient** method, which has the special property that, in generating its set of conjugate vectors, it can compute a new vector $p_k$ using only the previous vector $p_{k-1}$. This approach requires very little storage and computation.

## 4.6 Conjugate Gradient Method

The conjugate gradient method is a way of constructing a conjugate set of direction vectors. Each direction vector is chosen to be a linear combination of the steepest descent direction, $-\nabla f(x_k) \equiv -\nabla f_k$, and the previous direction $p_{k-1}$. Thus, we can start with the relationship

$$p_k \;=\; -\nabla f_k \;+\; \beta_k \, p_{k-1} \tag{17}$$

where $\beta_k$ is determined by the requirement that $p_{k-1}$ and $p_k$ be conjugate w.r.t $\nabla^2 f(x_k) \equiv \nabla^2 f_k$. Therefore, $p_{k-1}^T \, \nabla^2 f_k \, p_k = 0$.

Premultiplying Equation (17) by $p_{k-1}^T \nabla^2 f_k$ gives

$$p_{k-1}^T \nabla^2 f_k \, p_k \;=\; -p_{k-1}^T \nabla^2 f_k \, \nabla f_k \;+\; p_{k-1}^T \nabla^2 f_k \, \beta_k \, p_{k-1} \;=\; 0$$

$$\beta_k \;=\; \frac{\nabla {f_k}^T \nabla^2 f_k \, p_{k-1}}{p_{k-1}^T \nabla^2 f_k \, p_{k-1}} \tag{18}$$

It makes intuitive sense to choose the first direction, $p_0$ to be the descent direction at the initial point $x_0$; i.e., $\beta_0 = 0$.

The following is pseudocode to describe the Conjugate Gradient method.

INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, second partial derivative functions $\frac{\partial^2 f}{\partial x_i \, \partial x_j}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniters$
OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = conjGrad($n$,$x_0$,$tol$,$maxniters$)
    $x = x_0$
    $g \;=\; \nabla f(x)$
    $iter = 0$
    WHILE $iter \leq maxniters$
        $H \;=\; \nabla^2 f(x)$  (if $f$ is quadratic form, this can come out of WHILE loop)
        IF $iter = 0$
            $p \;=\; -g$
        ELSE
            $\beta \;=\; \frac{g \cdot H \, p}{p \cdot H \, p}$
            $p \;=\; -g \;+\; \beta \, p$
        Find $\lambda$ so that $f(x + \lambda \, p)$ is acceptable
        $x = x + \lambda \, p$
        $g \;=\; \nabla f(x)$
        $iter = iter + 1$
        IF $\|g\| < tol$

break

$x_{min} = x$

$f_{min} = f(x)$

$niters = iter$

## 4.7 Secant Conjugate Gradient

We can simplify our calculation for $\beta_k$ to avoid the computation of $\nabla^2 f_k$. Consider the case where $f$ is of quadratic form. We can using a Taylor series expression for $f(x_{k-1})$ to get an exact expression for $\nabla^2 f_k$.

$$
\begin{aligned}
f(x_{k-1}) &= f(x_k - \lambda_{k-1}\, p_{k-1}) = f(x_k) - \lambda_{k-1}\, \nabla f_k{}^T\, p_{k-1} + \frac{1}{2}\lambda_{k-1}^2\, p_{k-1}^T\, \nabla^2 f_k\, p_{k-1} \\
f_{k-1} &= f_k - \lambda_{k-1}\, \nabla f_k{}^T\, p_{k-1} + \frac{1}{2}\lambda_{k-1}^2\, p_{k-1}^T\, \nabla^2 f_k\, p_{k-1} \\
\nabla f_{k-1} &= \nabla f_k - \lambda_{k-1}\, \nabla^2 f_k\, p_{k-1} + 0 \\
\lambda_{k-1}\, \nabla^2 f_k\, p_{k-1} &= \nabla f_k - \nabla f_{k-1} \\
\nabla^2 f_k\, p_{k-1} &= \frac{1}{\lambda_{k-1}} (\nabla f_k - \nabla f_{k-1})
\end{aligned}
$$

Substituting this approximation into the equation for $\beta_k$, Equation (18), gives

$$
\beta_k = \frac{\nabla f_k{}^T\, \nabla^2 f_k\, p_{k-1}}{p_{k-1}^T\, \nabla^2 f_k\, p_{k-1}}
$$

$$
\beta_k = \frac{\nabla f_k{}^T \left(\frac{1}{\lambda_{k-1}}\right) (\nabla f_k - \nabla f_{k-1})}{\vec{p}_{k-1}^T \left(\frac{1}{\lambda_{k-1}}\right) (\nabla f_k - \nabla f_{k-1})}
$$

$$
\beta_k = \frac{\nabla f_k{}^T\, (\nabla f_k - \nabla f_{k-1})}{p_{k-1}^T\, (\nabla f_k - \nabla f_{k-1})}
$$

This expression for $\beta_k$ avoids the compuation of the full $\nabla^2 f_k$ matrix.

The previous expression for $\beta_k$ is exact when $f$ is of quadratic form. If $f$ is not of quadratic form, we can still use this expression as an approximation.

The following is pseudocode to describe the Secant Conjugate Gradient method. iter

INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniters$
OUTPUT: Value which differs from the $x$ value, $x_{min}$, of a local minimum of $f(x)$ by less than $tol$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = secConjGrad($n$,$x_0$,$tol$,$maxniters$)
$x = x_0$
$g = \nabla f(x)$
$iter = 0$
WHILE $iter \leq maxniters$
IF $iter = 0$
$p = -g$
ELSE
$\beta = \frac{g \cdot (g - gprev)}{p \cdot (g - gprev)}$
$p = -g + \beta\, p$
Find $\lambda$ to minimize $f(x + \lambda\, p)$ (see Section 4.1)
$x = x + \lambda\, p$
$gprev = g$
$g = \nabla f(x)$
$iter = iter + 1$
IF $\|g\| < tol$
break
$x_{min} = x$
$f_{min} = f(x)$
$niters = iter$

For functions that are not of quadratic form, we can still use the conjugate gradient to find a minimum, but it will not necessarily converge to the minimum in $n$ iterations. There are two ways to look at the conjugate gradient method applied to general nonlinear functions: 1) we could simply proceed as if the function were of quadratic form, choosing the $p_k$ direction at each iteration as we did before, or 2) we could approximate the general linear function at each iteration by a function that is of quadratic form, and move to the minimum of that approximating function in $n$ iterations. Approach 2 above would add to the complexity of the algorithm because we would be taking $n$ iterations at each iteration; i.e., each quadratic function approximation of the general linear function. Approach 1) would not necessarily maintain the conjugacy properties that we have for quadratic functions. This is because the $p_k$ directions would be conjugate to $\nabla^2 f_k$, but $\nabla^2 f_k$ would no longer be a matrix of constant real numbers and would instead be changing at each iteration.

If we are going to approximate a quadratic function at each iteration, it would make sense to use a Newton-type approach, as we did for functions of one variable. That is, we can construct a quadratic approximation to $f$ at each iteration, using first and second derivative information at that point in the iteration ($x_i$). We can then move to the minimum of that quadratic function in one iteration.

## 4.8 Newton's Method

Newton's Method for multivariate functions is analogous to that of single variable functions. We will construct a quadratic approximation to $f$ at each iteration, using first and second derivative information at that point in the iteration ($x_i$). As we did for the single variable case, we can derive an expression for the approximating quadratic function at $x_i$ as a function of the $f$ value and its first and second derivatives – in this case, $f_k$, $\nabla f_k$, and $\nabla^2 f_k$.

Let $q_i(x) = \frac{1}{2} x^T A\, x + b^T x + c$ be the approximating quadratic form function at $x$, where $q_i : \Re^n \to \Re$, $x \in \Re^n$ $A \in \Re^{n \times n}$, $b \in \Re^n$, and $c \in \Re$. We can determine $A$, $b$, and $c$, and thus $q_i$, by enforcing the following conditions:

$$\begin{aligned}
\nabla^2 q_i(x_i) &= \nabla^2 f(x_i) \equiv \nabla^2 f_i \\
\nabla q_i(x_i) &= \nabla f(x_i) \equiv \nabla f_i \\
q_i(x_i) &= f(x_i) \equiv f_i
\end{aligned}$$

$$\begin{aligned}
\nabla^2 q_i(x_i) &= \nabla^2 f_i \\
A &= \nabla^2 f_i
\end{aligned}$$

$$\begin{aligned}
\nabla q_i(x_i) &= \nabla f_i \\
A\, x_i + b &= \nabla f_i \\
\nabla^2 f_i\, x_i + b &= \nabla f_i \\
b &= \nabla f_i - \nabla^2 f_i\, x_i
\end{aligned}$$

$$\begin{aligned}
q_i(x_i) &= f_i \\
\frac{1}{2}\, x_i^T \nabla^2 f_i\, x_i + \left(\nabla f_i - \nabla^2 f_i\, x_i\right) x_i + c &= f_i \\
c &= f_i - \left(\nabla f_i - \nabla^2 f_i\, x_i\right) x_i - \frac{1}{2}\, x_i^T \nabla^2 f_i\, x_i \\
c &= f_i - \nabla f_i\, x_i + \frac{1}{2}\, x_i^T \nabla^2 f_i\, x_i
\end{aligned}$$

$$q_i(x) = \frac{1}{2}\, x^T \nabla^2 f_i\, x + \left(\nabla f_i - \nabla^2 f_i\, x_i\right) x + f_i - \nabla f_i\, x_i + \frac{1}{2}\, x_i^T \nabla^2 f_i\, x_i$$

$q_i(x)$ attains its minimum value when $\nabla q_i(x) = 0$, or

$$\begin{aligned}
\nabla^2 f_i\, x + \nabla f_i - \nabla^2 f_i\, x_i &= 0 \\
\nabla^2 f_i\, x &= \nabla^2 f_i\, x_i - \nabla f_i \\
x &= \left[\nabla^2 f_i\right]^{-1} \left(\nabla^2 f_i\, x_i - \nabla f_i\right) \\
x &= x_i - \left[\nabla^2 f_i\right]^{-1} \nabla f_i
\end{aligned}$$

This is equivalent to $x_{i+1} = x_i + \Delta x_i$ where

$$\begin{aligned}
\Delta x_i &= -\left[\nabla^2 f_i\right]^{-1} \nabla f_i \\
\left[\nabla^2 f_i\right] \Delta x_i &= -\nabla f_i
\end{aligned}$$

Therefore, we find $\Delta x_i$ at each iteration by solving the $n \times n$ linear system of equations

$$\left[\nabla^2 f_i\right] \Delta x_i \;=\; -\nabla f_i$$

The following is pseudocode to describe Newton's method for multivariable minimization.

INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, second partial derivative functions $\frac{\partial^2 f}{\partial x_i^2}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniter$
OUTPUT: The $x$ value, $x_{min}$, of a local minimum of $f(x)$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = newtonMult($n$,$x_0$,$tol$,$maxniter$)
    $x = x_0$
    $g \;=\; \nabla f(x)$
    $iter = 0$
    WHILE $iter \leq maxniter$
        $H \;=\; \nabla^2 f(x)$
        Solve $H\,p \;=\; -g$ for $p$
        Find $\lambda$ so that $f(x + \lambda\,p)$ is acceptable
        $x = x + \lambda\,p$
        $g \;=\; \nabla f(x)$
        $iter = iter + 1$
        IF $\|g\| < tol$
            break
    $x_{min} = x$
    $f_{min} = f(x)$
    $niter = iter$

## 4.9 Descent Directions

In choosing a direction $p_k$, we want to make sure that we are moving in a direction such that $f$ is decreasing; i.e., the derivative in the direction of $p_k$ is less than 0, or a *descent direction*. This is equivalent to $\frac{\partial f}{\partial p_k}(x_k) < 0$. Recall that $\frac{\partial f}{\partial p_k}(x_k) \;=\; \nabla f_k^T\, p_k$. Now consider the directional derivative in the context of the steepest descent, conjugate gradient, and Newton's methods.

### 4.9.1 Steepest Descent

$$\frac{\partial f}{\partial p_k}(x_k) \;=\; \nabla f_k^{\,T}\, p_k \;=\; -\nabla f_k^{\,T}\, \nabla f_k \;=\; -\|\nabla f_k\|^2 \;<\; 0$$

Thus, the steepest descent method guarantees descent directions.

### 4.9.2 Conjugate Gradient

Here we have to be more precise about the choice of the step length parameter $\lambda_{k-1}$ in the update formula $x_k \;=\; x_{k-1} \;+\; \lambda_{k-1}\, p_{k-1}$. The search direction $p_k$ may fail to be a descent direction unless $\lambda_{k-1}$ satisfies certain conditions.

$$\begin{aligned}
\frac{\partial f}{\partial p_k}(x_k) \;&=\; \nabla f_k^{\,T}\, p_k \\
&=\; \nabla f_k^{\,T}\, (-\nabla f_k \;+\; \beta_k\, p_{k-1}) \\
&=\; -\|\nabla f_k\|^2 \;+\; \beta_k\, \nabla f_k^{\,T}\, p_{k-1}
\end{aligned}$$

If the step length $\lambda_{k-1}$ leads to a local minimizer of $f$ along $p_{k-1}$, then

$$\nabla f_k^{\,T}\, p_{k-1} \;=\; 0 \qquad \Rightarrow \qquad \frac{\partial f}{\partial p_k}(x_k) \;=\; -\|\nabla f_k\|^2 \;<\; 0$$

If the step length $\lambda_{k-1}$ does not lead to a local minimizer of $f$ along $p_{k-1}$, then the $\beta_k\, \nabla f_k^{\,T}\, p_{k-1}$ term may dominate the $-\|\nabla f_k\|^2$ term and make $\frac{\partial f}{\partial p_k}(x_k) > 0$ (ascent direction). We can avoid this by requiring that the step length $\lambda_{k-1}$ satisfy the *strong Wolfe conditions.*

$$\begin{aligned}
f_k \;&\leq\; f_{k-1} \;+\; c_1\, \lambda_{k-1}\, \frac{\partial f}{\partial p_{k-1}}(x_{k-1}) \\
f_k \;&\leq\; f_{k-1} \;+\; c_1\, \lambda_{k-1} \nabla f_{k-1}^{\,T}\, p_{k-1} \\
&\qquad\qquad \text{and} \\
|\nabla f_k^{\,T}\, p_{k-1}| \;&\leq\; c_2\, |\nabla f_{k-1}^{\,T}\, p_{k-1}| \\
&\qquad\qquad\qquad\qquad \text{where } 0 < c_1 < c_2 < 0
\end{aligned}$$

### 4.9.3 Newton's Method

$$\begin{aligned} p_k &= -\left[\nabla^2 f_k\right]^{-1} \nabla f_k \\ \nabla f_k &= -\left[\nabla^2 f_k\right] p_k \end{aligned}$$

$$\begin{aligned} \frac{\partial f}{\partial p_k}(x_k) &= \nabla f_k^T \, p_k \\ &= \left(-\left[\nabla^2 f_k\right] p_k\right)^T p_k \\ &= -p_k^T \left[\nabla^2 f_k\right]^T p_k \\ &= -p_k^T \left[\nabla^2 f_k\right] p_k \qquad \text{because } \nabla^2 f_k \text{ is symmetric} \end{aligned}$$

If $\nabla^2 f_k$ is *positive definite*, then, by definition, $p_k^T \nabla^2 f_k p_k > 0$, and then $-p_k^T \left[\nabla^2 f_k\right] p_k < 0$. Thus, Newton's method guarantees descent directions when the Hessian is positive definite.

## 4.10 Secant Newton's Method (Quasi-Newton Methods)

For functions of one variable, we examined a secant approximation for the first and second derivatives of $f$. This allowed us to calculate $\Delta x$ using only function value information. We can do something similar for functions of more than one variable, however, we will use use exact gradient information and approximate only the Hessian matrix.

First consider the 1D case. We can make the following backward difference approximation for $f''(x_k)$ in terms of $f'(x_k - 1)$ and $f'(x_k)$

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}$$

or

$$a_k \left(x_k - x_{k-1}\right) = f'(x_k) - f'(x_{k-1})$$

where $a_k \in \Re$ is an approximation to $f''(x_k)$.

For functions of more than one variable, this becomes

$$A_k\,(x_k\ -\ x_{k-1})\ =\ \nabla f(x_k)\ -\ \nabla f(x_{k-1}) \tag{19}$$

where $A_k \in \Re^{n\times n}$ is an approximation to $\nabla^2 f(x_k)$. We will refer to Equation (19) as the *secant equation*.

The difficulty with the secant equation is, we cannot simply solve for $A_k$, as we did for $a_k$ in the 1D case. The reason is, the secant equation represents a problem involving $n^2$ unknowns and $n$ equations, which gives an infinite number of solutions for $n > 1$. Constructing a successful secant approximation consists of selecting a good way to choose $A_k$ among the infinite number of possibilities.

Consider the function $m_k(x)$ that is a linear model of the gradient function between $x_{k-1}$ and $x_k$; i.e,

$$\begin{aligned} m_k(x) &= \nabla f_k\ +\ A_k\,(x\ -\ x_k) \\ m_k(x_k) &= \nabla f_k \\ m_k(x_{k-1}) &= \nabla f_{k-1} \\ m_{k-1}(x) &= \nabla f_{k-1}\ +\ A_{k-1}\,(x\ -\ x_{k-1}) \end{aligned}$$

An approach that leads to a good secant approximation to $\nabla^2 f_k$ is to try to minimize the change in $m(x)$ functional forms from $x_{k-1}$ to $x_k$, subject to the secant equation $A_k s_k = y_k$, where $s_k\ =\ x_k\ -\ x_{k-1}$ and $y_k\ =\ \nabla f_k\ -\ \nabla f_{k-1}$.

$$\begin{aligned} m_k(x)\ -\ m_{k-1}(x) &= \nabla f_k\ +\ A_k\,(x\ -\ x_k)\ -\ \nabla f_{k-1}\ -\ A_{k-1}\,(x\ -\ x_{k-1}) \\ &= \nabla f_k\ -\ \nabla f_{k-1}\ -\ A_k\,(x_k\ -\ x_{k-1}) \\ &\quad +\ A_k\,(x_k\ -\ x_{k-1})\ +\ A_k\,(x\ -\ x_k)\ -\ A_{k-1}\,(x\ -\ x_{k-1}) \\ &= 0\ +\ A_k\,(x_k\ -\ x_{k-1})\ +\ A_k\,(x\ -\ x_k)\ -\ A_{k-1}\,(x\ -\ x_{k-1}) \\ &= A_k\,(x\ -\ x_{k-1})\ -\ A_{k-1}\,(x\ -\ x_{k-1}) \\ &= (A_k\ -\ A_{k-1})\,(x\ -\ x_{k-1}) \end{aligned}$$

For any $x \in \Re^n$, express $x - x_{k-1}$ as a combination of a vector in the direction of $x_k - x_{k-1}$ and a vector $\perp$ to $x_k - x_{k-1}$.

$$\begin{aligned} x \;-\; x_{k-1} \;&=\; \alpha\,(x_k - x_{k-1}) \;+\; t\,, \qquad \text{where } t \;\perp\; x_k - x_{k-1} \\ x \;-\; x_{k-1} \;&=\; \alpha\, s_k \;+\; t\,, \qquad \text{where } s_k^T\, t = 0 \end{aligned}$$

$$\begin{aligned} m_k(x) \;-\; m_{k-1}(x) \;&=\; (A_k \;-\; A_{k-1})\,(\alpha\, s_k \;+\; t) \\ &=\; \alpha\,(A_k \;-\; A_{k-1})\, s_k \;+\; (A_k \;-\; A_{k-1})\, t \end{aligned}$$

We have no control over the first term, but we can make the second term 0 for all $x$ by choosing $A_k$ such that $A_k \;-\; A_{k-1} \;=\; u\, s_k^T$, where $u \in \Re^n$. This gives

$$(A_k \;-\; A_{k-1})\, t \;=\; u\, s_k^T\, t \;=\; 0 \qquad \text{since } s_k^T\, t = 0$$

We now have that $A_k \;-\; A_{k-1} \;=\; u\, s_k^T$. But this holds for any $u \in \Re^n$, and thus does not uniquely determine $A_k \;-\; A_{k-1}$. We can solve for $u$ by adding the secant equation condition $A_k\, s_k \;=\; y_k$.

$$\begin{aligned} A_k \;-\; A_{k-1} \;&=\; u\, s_k^T \\ A_k\, s_k \;-\; A_{k-1}\, s_k \;&=\; u\, s_k^T\, s_k \\ y_k \;-\; A_{k-1}\, s_k \;&=\; u\, s_k^T\, s_k \\ u \;&=\; \frac{y_k \;-\; A_{k-1}\, s_k}{s_k^T\, s_k} \end{aligned}$$

$$\begin{aligned} A_k \;-\; A_{k-1} \;&=\; u\, s_k^T \\ A_k \;-\; A_{k-1} \;&=\; \left(\frac{y_k \;-\; A_{k-1}\, s_k}{s_k^T\, s_k}\right) s_k^T \\ A_k \;-\; A_{k-1} \;&=\; \frac{(y_k \;-\; A_{k-1}\, s_k)\, s_k^T}{s_k^T\, s_k} \end{aligned}$$

which is **Broyden's Update** $\qquad \boxed{A_k \;=\; A_{k-1} \;+\; \dfrac{(y_k \;-\; A_{k-1}\, s_k)\, s_k^T}{s_k^T\, s_k}}$

The problem with Broyden's update is that even if $A_{k-1}$ is symmetric, $A_k$ will not be symmetric unless $y_k - A_{k-1}\, s_k$ is a multiple of $s_k$. In approximating Hessians, we would like to preserve symmetry. Therefore, we seek an $A_k$ that obeys $A_k = A_k^T$ as well as $A_k\, s_k = y_k$. Such an $A_k$ can be constructed using a series of Frobenius norm projections of $A_k$ into the subspace $S$ of symmetric matrices, which gives the **Powell-Symmetric-Broyden (PSB) Update**

$$\boxed{A_k = A_{k-1} + \frac{(y_k - A_{k-1}\, s_k)\, s_k^T + s_k\, (y_k - A_{k-1}\, s_k)^T}{s_k^T\, s_k} - \frac{\left[(y_k - A_{k-1}\, s_k)^T\, s_k\right] s_k\, s_k^T}{\left(s_k^T\, s_k\right)^2}}$$

There is also a **symmetric, rank 1 update (SR1)**

$$\boxed{A_k = A_{k-1} + \frac{(y_k - A_{k-1}\, s_k)\, (y_k - A_{k-1}\, s_k)^T}{(y_k - A_{k-1}\, s_k)^T\, s_k}}$$

There are other symmetric, positive definite updates:

**Davidson, Fletcher, and Powell (DFP)**

$$\boxed{A_k = A_{k-1} + \frac{(y_k - A_{k-1}\, s_k)\, y_k^T + y_k\, (y_k - A_{k-1}\, s_k)^T}{y_k^T\, s_k} - \frac{\left[(y_k - A_{k-1}\, s_k)^T\, s_k\right] y_k\, y_k^T}{\left(y_k^T\, s_k\right)^2}}$$

**Broyden, Fletcher, Goldfarb, and Shanno (BFGS)**

$$\boxed{A_k = A_{k-1} + \frac{y_k\, y_k^T}{y_k^T\, s_k} - \frac{A_{k-1}\, s_k\, s_k^T\, A_{k-1}}{s_k^T\, A_{k-1}\, s_k}}$$

We can also directly update approximations to the inverse of the Hessian, using the Sherman-Morrison formula. For the case of the BFGS update, this gives

$$\boxed{A_k^{-1} = \left(I - \frac{s_k\, y_k^T}{y_k^T\, s_k}\right) A_{k-1}^{-1} \left(I - \frac{y_k\, s_k^T}{y_k^T\, s_k}\right) + \frac{s_k\, s_k^T}{y_k^T\, s_k}}$$

The following is pseudocode to describe the Quasi-Newton method for multivariable minimization.

INPUT: Function $f(x) : \Re^n \to \Re$, partial derivative functions $\frac{\partial f}{\partial x_i}(x) : \Re^n \to \Re$, dimension $n$, starting point $x_0$, tolerance $tol$, maximum number of iterations, $maxniter$
OUTPUT: The $x$ value, $x_{min}$, of a local minimum of $f(x)$, the function value at that point, $f_{min}$, and the number of iterations taken, $niters$

function [$x_{min}$, $f_{min}$,$niters$] = quasiNewt($n$,$x_0$,$tol$,$maxniter$)
    $x = x_0$
    $g = \nabla f(x)$
    $iter = 1$
    WHILE $iter \leq maxniter$
        IF $iter = 1$
            $A = eye(n)$
        ELSE
            $A = A +$ (function of $A, x, xprev, g, gprev$)  (quasi Newton update for $A$)
        Solve $A\, p = -g$ for $p$
        Find $\lambda$ to minimize $f(x + \lambda\, p)$  (see Section 4.1)
        $xprev = x$
        $x = x + \lambda\, p$
        $gprev = g$
        $g = \nabla f(x)$
        IF $\|g\| < tol$
            break
        $iter = iter + 1$
    $x_{min} = x$
    $f_{min} = f(x)$
    $niter = iter$

# 5 Least Squares Problems

Often times, an objective function to be minimized is formulated in terms of the sum of squares of residuals. This is particularly common when fitting a function (model) to data. For example, if you have a set of data $(t_i, y_i)\,;\ i = 1, \ldots, m$ and want to fit the model $M(x,t)$ to that data, where $x$ represents the parameters of the model. You could find the optimal parameter values by minimizing the residuals, where a residual $r_i(x)$ defined as $r_i(x) = M(x,t_i) - y_i$.

Mathematically, this least squares problem can be represented as

$$\min_{x \in \Re^n} f(x) = \frac{1}{2} \sum_{i=1}^{m} r_i(x)^2$$

where $m$ is the number of data points, $n$ is the number of parameters to be optimized, and $m > n$.

Define the residual function $R(x)\,, R : \Re^n \to \Re^m$, as

$$R(x) = \begin{bmatrix} r_i(x) \\ \vdots \\ r_m(x) \end{bmatrix}$$

We can then express the minimization problem as

$$\min_{x \in \Re^n} f(x) = \frac{1}{2} \sum_{i=1}^{m} r_i(x)^2 = \frac{1}{2} R(x)^T R(x)$$

If each $r_i(x)$ is linear in $x$, then it is called a *linear least squares* problem. If each $r_i(x)$ is nonlinear in $x$, then it is called a *nonlinear least squares* problem.

## 5.1 Linear Least Squares

In this case, each $r_i(x)$ is linear in $x$, therefore we can express each $r_i(x)$ as

$$r_i(x) \;=\; a_{i1}\,x_1 \;+\; a_{i2}\,x_2 \;+\; \cdots \;+\; a_{in}\,x_n \;+\; b_i\;, \qquad i = 1, \ldots, m$$

Therefore

$$R(x) \;=\; \underbrace{\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}}_{A}_{m \times n} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}}_{x}_{n \times 1} + \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}}_{b}_{m \times 1}$$

We will define the $m \times n$ Jacobian matrix as

$$J \;=\; \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix}$$

We also know that

$$\begin{aligned} \frac{\partial r_1}{\partial x_1} \;&=\; a_{11} \qquad \cdots \qquad \frac{\partial r_1}{\partial x_n} \;=\; a_{1n} \\ &\vdots \\ \frac{\partial r_m}{\partial x_1} \;&=\; a_{m1} \qquad \cdots \qquad \frac{\partial r_m}{\partial x_n} \;=\; a_{mn} \end{aligned}$$

Therefore, $J = A$, and $R(x) \;=\; J\,x \;+\; b$.

To find the minimum, we will set the gradient of $f$ equal to 0 and solve for $x$

$$f(x) \;=\; \frac{1}{2}\sum_{i=1}^{m} r_i(x)^2$$

$$\nabla f(x) \;=\; \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$= \frac{1}{2}\begin{bmatrix} 2\,r_1(x)\,\frac{\partial r_1}{\partial x_1} \;+\; 2\,r_2(x)\,\frac{\partial r_2}{\partial x_1} \;+\; \cdots \;+\; 2\,r_m(x)\,\frac{\partial r_m}{\partial x_1} \\ \vdots \\ 2\,r_1(x)\,\frac{\partial r_1}{\partial x_n} \;+\; 2\,r_2(x)\,\frac{\partial r_2}{\partial x_n} \;+\; \cdots \;+\; 2\,r_m(x)\,\frac{\partial r_m}{\partial x_n} \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial r_1}{\partial x_n} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix}}_{J^T} \underbrace{\begin{bmatrix} r_1(x) \\ \vdots \\ r_m(x) \end{bmatrix}}_{R(x)}$$

$$n \times m \qquad\qquad m \times 1$$

Then

$$\nabla f(x) \;=\; J^T\,R(x) \;=\; J^T\,[J\,x \;+\; b] \;=\; 0$$

$$J^T\,J\,x \;+\; J^T\,b \;=\; 0$$

$$J^T\,J\,x \;=\; -J^T\,b$$

This represents a system of linear equations, known as the *Normal Equations*. Our minimum point is then the solution to the Normal Equations.

## 5.2 Nonlinear Least Squares

Consider the general problem of finding the solution to a system of $m$ nonlinear equations in $n$ unknowns, or finding $x$ that solves $F(x) = 0$, where $F : \Re^n \to \Re^m$. This problem can be expressed in matrix-vector form as

$$F(x) \;=\; \begin{bmatrix} a_{11}(x) & \cdots & a_{1n}(x) \\ \vdots & & \vdots \\ a_{m1}(x) & \cdots & a_{mn}(x) \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

If $m = n$, we can use multi-dimensional root-finding methods to solve for $x$ (Picard iteration, Newton-Raphson iteration, etc).

If $m > n$, there may or may not be a unique solution (root). In this case, think of each equation as being a residual, $r_i(x)$, expressed as

$$a_{i1}(x)\, x_1 \;+\; a_{i2}(x)\, x_2 \;+\; \cdots \;+\; a_{in}(x)\, x_n \;=\; r_i(x)\;, \qquad i = 1, \ldots, m$$

In this case, instead of trying to find $x$ that makes all of the $r_i(x) = 0$ (since this may not exist), we would attempt to somehow minimize the residuals. We would not want to simply sum the residuals in order to minimize, but we would rather attempt to minimize the sum of the squares of the residuals. This gives us the least squares problem we saw before, but now it is nonlinear.

$$\min_{x \in \Re^n} f(x) \;=\; \frac{1}{2} \sum_{i=1}^{m} r_i(x)^2$$

where each $r_i(x)$ is nonlinear in $x$.

If we let

$$R(x) \;=\; \begin{bmatrix} r_1(x) \\ \vdots \\ r_m(x) \end{bmatrix}$$

then

$$\sum_{i=1}^{m} r_i(x)^2 \;=\; \left[\begin{array}{ccc} r_i(x) & \cdots & r_m(x) \end{array}\right] \left[\begin{array}{c} r_1(x) \\ \vdots \\ r_m(x) \end{array}\right] \;=\; R(x)^{\,T}\, R(x)$$

Therefore, the nonlinear least squares problem is

$$\min_{x \in \Re^n} \; \frac{1}{2}\, R(x)^{\,T}\, R(x)$$

#### 5.2.0.1 Newton's Method for Nonlinear Least Squares

We have

$$f(x) \;=\; \frac{1}{2}\left(r_1^2 \,+\, \cdots \,+\, r_m^2\right) \;=\; \frac{1}{2}\, R(x)^{\,T}\, R(x)$$

$$\nabla f(x) \;=\; \left[\begin{array}{c} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{array}\right] \;=\; \left[\begin{array}{c} \frac{\partial r_1}{\partial x_1}\, r_1 \,+\, \cdots \,+\, \frac{\partial r_m}{\partial x_1}\, r_m \\ \vdots \\ \frac{\partial r_1}{\partial x_n}\, r_1 \,+\, \cdots \,+\, \frac{\partial r_m}{\partial x_n}\, r_m \end{array}\right]$$

$$= \left[\begin{array}{ccc} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_1} \\ \vdots & & \\ \frac{\partial r_1}{\partial x_n} & \cdots & \frac{\partial r_m}{\partial x_n} \end{array}\right] \left[\begin{array}{c} r_1 \\ \vdots \\ r_m \end{array}\right]$$

$$= \left[\begin{array}{ccc} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{array}\right]^T \left[\begin{array}{c} r_1 \\ \vdots \\ r_m \end{array}\right]$$

$$\nabla f(x) \;=\; J(x)^{\,T}\, R(x)$$

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

$$= \left[\begin{array}{c|c|c} \left(\frac{\partial r_1}{\partial x_1}\right)^2 + r_1 \frac{\partial^2 r_1}{\partial x_1^2} + & \cdots & \left(\frac{\partial r_1}{\partial x_1}\right)\left(\frac{\partial r_1}{\partial x_n}\right) + r_1 \frac{\partial^2 r_1}{\partial x_1 \partial x_n} + \\ \cdots + \left(\frac{\partial r_m}{\partial x_1}\right)^2 + r_m \frac{\partial^2 r_m}{\partial x_1^2} & & \cdots + \left(\frac{\partial r_m}{\partial x_1}\right)\left(\frac{\partial r_m}{\partial x_n}\right) + r_m \frac{\partial^2 r_m}{\partial x_1 \partial x_n} \\ \hline \vdots & & \vdots \\ \hline \left(\frac{\partial r_1}{\partial x_n}\right)\left(\frac{\partial r_1}{\partial x_1}\right) + r_1 \frac{\partial^2 r_1}{\partial x_n \partial x_1} + & \cdots & \left(\frac{\partial r_1}{\partial x_n}\right)^2 + r_1 \frac{\partial^2 r_1}{\partial x_n^2} + \\ \cdots + \left(\frac{\partial r_m}{\partial x_n}\right)\left(\frac{\partial r_m}{\partial x_1}\right) + r_m \frac{\partial^2 r_m}{\partial x_n \partial x_1} & & \cdots + \left(\frac{\partial r_m}{\partial x_n}\right)^2 + r_m \frac{\partial^2 r_m}{\partial x_n^2} \end{array}\right]$$

$$= \underbrace{\begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial r_1}{\partial x_n} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix}}_{J(x)^T} \underbrace{\begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix}}_{J(x)}$$

$$+ \underbrace{\left[\begin{array}{c|c|c} r_1 \frac{\partial^2 r_1}{\partial x_1^2} + \cdots + r_m \frac{\partial^2 r_m}{\partial x_1^2} & \cdots & r_1 \frac{\partial^2 r_1}{\partial x_1 \partial x_n} + r_m \frac{\partial^2 r_m}{\partial x_1 \partial x_n} \\ \hline \vdots & & \vdots \\ \hline r_1 \frac{\partial^2 r_1}{\partial x_n \partial x_1} + \cdots + r_m \frac{\partial^2 r_m}{\partial x_n \partial x_1} & \cdots & r_1 \frac{\partial^2 r_1}{\partial x_n^2} + r_m \frac{\partial^2 r_m}{\partial x_n^2} \end{array}\right]}_{\equiv S(x)}$$

$$\nabla^2 f(x) \;=\; J(x)^T J(x) \;+\; S(x)$$

The Newton step is $x_{k+1} = x_k + \Delta x_k$, where $\Delta x_k = -\left[\nabla^2 f(x_k)\right]^{-1} \nabla f(x_k)$. Therefore, the full Newton step is described by

$$\boxed{x_{k+1} \;=\; x_k \;-\; \left[J(x_k)^T J(x_k) \;+\; S(x_k)\right]^{-1} J(x_k)^T R(x_k)}$$

In practice, you would solve the $n \times n$ system $\left[J(x_k)^T J(x_k) + S(x_k)\right] \Delta x_k = -J(x_k)^T R(x_k)$ for $\Delta x_k$, and then make the update $x_{k+1} = x_k + \Delta x_k$.

### 5.2.1 Gauss Newton Method for Nonlinear Least Squares

The problem with the full Newton step is that $S(x_k)$ is usually either unavailable or inconvenient to obtain. By omitting $S(x_k)$, you get the *Gauss Newton method*.

$$\boxed{x_{k+1} \;=\; x_k \;-\; \left[J(x_k)^{\,T}\, J(x_k)\right]^{-1} J(x_k)^{\,T}\, R(x_k)}$$

In practice, you would solve the $n \times n$ system $\left[J(x_k)^{\,T}\, J(x_k)\right] \Delta x_k \;=\; -J(x_k)^{\,T}\, R(x_k)$ for $\Delta x_k$, and then make the update $x_{k+1} \;=\; x_k \;+\; \Delta x_k$.

The Gauss-Newton method tends to work better on small- or zero-residual problems.

### 5.2.2 Levenberg-Marquardt Method

The *Levenberg-Marquardt (LM) method*, also known as damped least-squares method, is used to solve nonlinear least squares problems. The LM method interpolates between the Gauss-Newton method and the steepest descent method. The LM method is more robust than the Gauss-Newton method, which means that in many cases it finds a solution even if it starts very far off the final minimum. For well-behaved functions and reasonable starting positions, the LM method tends to be a bit slower than the Gauss-Newton method.

The LM update can be written as

$$\boxed{x_{k+1} \;=\; x_k \;-\; \left[J(x_k)^{\,T}\, J(x_k) \;+\; \gamma I_n\right]^{-1} J(x_k)^{\,T}\, R(x_k)}$$

When $\gamma = 0$, the LM method is the Gauss-Newton method. The LM method approaches the steepest descent method as $\gamma$ gets large.

# 6 Scaling

Independent variables may vary greatly in magnitude, and can have a significant effect on the optimization algorithm. For example, in the optimization problem,

$$\min_{x \in \Re^2} f(x) \,, \quad \text{where} \quad x \;=\; \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \qquad x_1 \in [100, 1000] \qquad x_2 \in [10^{\,-7}, 10^{\,-6}]$$

the $x_2$ variable will be virtually ignored.

The obvious remedy is to rescale the individual variables; i.e., change their units by multiplying by a scaling matrix

$$\hat{x} \;=\; D_x \, x$$

$$\text{where} \qquad D_x \;=\; \begin{bmatrix} 10^{\,-3} & 0 \\ 0 & 10^{\,6} \end{bmatrix}$$

Then

$$\hat{x} \;=\; \begin{bmatrix} \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} \qquad \hat{x}_1 \in [0.1, 1] \qquad \hat{x}_2 \in [0.1, 1]$$

The question then is, how will this affect our search algorithms?

Define $\hat{x} = T\,x$, $x \in \Re^n$, $\hat{x} \in \Re^n$, $T \in \Re^{n \times n}$.

$$\hat{f}(\hat{x}) \;=\; f(x) \;=\; f(T^{\,-1}\,\hat{x})$$

$$\text{let} \qquad T^{\,-1} \;=\; \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

$$x \;=\; T^{-1}\,\hat{x}$$

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \;=\; \begin{bmatrix} a_{11}\,\hat{x}_1 + a_{12}\,\hat{x}_2 + \cdots + a_{1n}\,\hat{x}_n \\ \vdots \\ a_{n1}\hat{x}_1 + a_{n2}\,\hat{x}_2 + \cdots + a_{nn}\hat{x}_n \end{bmatrix}$$

$$\nabla \hat{f}(\hat{x}) \;=\; \begin{bmatrix} \frac{\partial \hat{f}}{\partial \hat{x}_1} \\ \vdots \\ \frac{\partial \hat{f}}{\partial \hat{x}_n} \end{bmatrix}$$

$$\begin{aligned} \frac{\partial \hat{f}}{\partial \hat{x}_i} \;&=\; \frac{\partial f}{\partial x_1}\frac{\partial x_1}{\partial \hat{x}_i} \;+\; \frac{\partial f}{\partial x_2}\frac{\partial x_2}{\partial \hat{x}_i} \;+\; \cdots \;+\; \frac{\partial f}{\partial x_n}\frac{\partial x_n}{\partial \hat{x}_i} \\ &=\; \frac{\partial f}{\partial x_1}\,(a_{11}) \;+\; \frac{\partial f}{\partial x_2}\,(a_{21}) \;+\; \cdots \;+\; \frac{\partial f}{\partial x_n}\,(a_{n1}) \end{aligned}$$

$$\nabla \hat{f}(\hat{x}) \;=\; \underbrace{\begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ \vdots & & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix}}_{\left(T^{-1}\right)^T} \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\nabla \hat{f}(\hat{x}) \;=\; \left(T^{-1}\right)^T \nabla f(x)$$

Similarly,

$$\nabla^2 \hat{f}(\hat{x}) \;=\; \left(T^{-1}\right)^T \nabla^2 f(x)\, T^{-1}$$

Now consider the steepest descent and Newton's methods with respect to the new transformed gradient and Hessian. Let $S$ and $\hat{S}$ represent the steps taken with respect to the original $x$ space and the transformed $\hat{x}$ space, respectively.

## 6.1 Steepest Descent

$$\hat{S}^{SD} = -\nabla \hat{f}(\hat{x}) = -\left(T^{-1}\right)^T \nabla f(x)$$

$$\begin{aligned} S^{SD} &= T^{-1}\, \hat{S}^{SD} \\ &= -T^{-1} \left(T^{-1}\right)^T \nabla f(x) \\ &= -T^{-1} \left(T^T\right)^{-1} \nabla f(x) \\ &= -\left(T^T T\right)^{-1} \nabla f(x) \\ &\neq -\nabla f(x) \end{aligned}$$

Therefore, the steepest descent direction is changed by the scaling transformation.

## 6.2 Newton's Method

$$\begin{aligned} \hat{S}^N &= -\left[\nabla^2 \hat{f}(\hat{x})\right]^{-1} \nabla \hat{f}(\hat{x}) \\ &= -\left[\left(T^{-1}\right)^T \nabla^2 f(x)\, T^{-1}\right]^{-1} \left(T^{-1}\right)^T \nabla f(x) \\ &= -\left[\left(T^{-1}\right)^{-1} \left(\nabla^2 f(x)\right)^{-1} \left(\left(T^{-1}\right)^T\right)^{-1}\right] \left(T^{-1}\right)^T \nabla f(x) \\ &= -T \left[\nabla^2 f(x)\right]^{-1} \left(\left(T^T\right)^{-1}\right)^{-1} \left(T^T\right)^{-1} \nabla f(x) \\ &= -T \left[\nabla^2 f(x)\right]^{-1} T^T \left(T^T\right)^{-1} \nabla f(x) \\ &= -T \left[\nabla^2 f(x)\right]^{-1} \nabla f(x) \end{aligned}$$

$$\begin{aligned} S^N &= T^{-1}\, \hat{S}^N \\ &= -T^{-1}\, T \left[\nabla^2 f(x)\right]^{-1} \nabla f(x) \\ &= -\left[\nabla^2 f(x)\right]^{-1} \nabla f(x) \end{aligned}$$

Therefore, the Newton's method direction is unchanged by the scaling transformation.

# 7 Stopping Criteria

When the model Hessian is positive definite, $\nabla f(x) = 0$ is a necessary and sufficient condition for $x$ to be a local minimizer of $f$. To test whether $\nabla f(x) \approx 0$, a test such as $\|\nabla f(x)\| \leq \epsilon$ is inadequate because it is strongly dependent on the scaling of both $f$ and $x$. For example, if $\epsilon = 10^{-3}$ and $f \in \left[10^{-7}, 10^{-5}\right]$, almost any $x$ will satisfy the test. But if $f \in \left[10^{5}, 10^{7}\right]$, then this test is too stringent.

Instead, we could use a relative gradient of $f$ at $x$ as a test for stopping, where each component of this relative gradient is given by

$$\begin{aligned}
\text{relgrad}(x)_i &= \frac{\text{relative rate of change in } f}{\text{relative rate of change in } x_i} \\
&= \lim_{\delta \to 0} \frac{\frac{f(x+\delta\, e_i) - f(x)}{f(x)}}{\frac{\delta}{x_i}} \\
&= \lim_{\delta \to 0} \frac{f(x + \delta\, e_i) \;-\; f(x)}{\delta} \;\cdot\; \frac{x_i}{f(x)} \\
&= \frac{\partial f}{\partial x_i} \;\cdot\; \frac{x_i}{f(x)} \\
&= \frac{[\nabla f(x)]_i \; x_i}{f(x)}
\end{aligned}$$

We then test for

$$\begin{aligned}
\|\text{relgrad(x)}\| &\leq \epsilon \\
\left\| \frac{\nabla f(x) \cdot x}{f(x)} \right\| &\leq \epsilon
\end{aligned}$$

This can break down if $x_i$ or $f(x)$ happen to be near 0. We can fix this by making the following modificiation

$$\max_{1\leq i\leq n}\left|\frac{[\nabla f(x)]_i\ \max\{|x_i|\,,\text{typ }x_i\}}{\max\{|f(x)|\,,\text{typ }f\}}\right|\ \leq\ \epsilon$$

where typ $x_i$ and typ $f$ are user's estimate of typical magnitude of $x_i$ and $f$, respectively.

# 8 Constrained Optimization

## 8.1 The Lagrangian Function

In constrained optimization, we attempt to find the values of the independent variables that optimize an objective function, subject to constraints placed on the values of the independent variables. The general formulation is:

$$\min_{x\in\Re^n}\ f(x) \text{ subject to } \begin{cases} c_i(x)=0\ ,\ i\in E \\ c_i(x)\leq 0\ ,\ i\in I \end{cases}$$

where $E$ and $I$ are two sets of indices. The $c_i\ ,\ i\in E$ are called *equality constraints*, and the $c_i\ ,\ i\in I$ are called *inequality constraints*. A *feasible set* of points is one where all the $x$ satisfy the constraints; i.e., $\Omega\ =\ \{x\mid c_i(x)=0\ ,\ i\in E;\ c_i(x)\leq 0\ ,\ i\in I\}$. So we can write our problem as

$$\min_{x\in\Omega}\ f(x)$$

Inequality constraints are said to be *active* if $c_i(x)=0$ and *inactive* if $c_i(x)<0$.

Consider the problem of one constraint, min $x_1+x_2$, s.t. $x_1^2+x_2^2-2=0$. The solution, $x^*$, is (-1,-1). At $x^*$, the constraint normal $\nabla c_1(x^*)$ is parallel to $\nabla f(x^*)$. That is, there is a scalar $\lambda_1^*$ s.t.

$$\nabla f(x^*)\ =\ \lambda_1^*\,\nabla c_1(x^*) \tag{20}$$

We can derive (20) by examining 1st order Taylor series of the objective and constraint functions. To retain feasibility w.r.t. the function $c_1(x) = 0$, we require that $c_1(x+d) = 0$. Therefore,

$$0 \;=\; c_1(x+d) \;\approx\; c_1(x) \;+\; \nabla c_1(x)^T d \;=\; \nabla c_1(x)^T d$$

Similarly, a direction of improvement must produce a decrease in $f$, so that

$$0 \;>\; f(x+d) \;-\; f(x) \;\approx\; \nabla f(x)^T d$$

A necessary condition for optimality is that no direction $d$ exists that satisfies both

$$\begin{aligned} \nabla c_1(x)^T d \;&=\; 0 \\ \nabla f(x)^T d \;&<\; 0 \end{aligned}$$

The only way such a direction can not exist is if $\nabla c_1(x)$ and $\nabla f(x)$ are parallel. This is the same as saying that $\nabla f(x) = \lambda_1 \nabla c_1(x)$ holds at some $x$, for some scalar $\lambda_1$.

We will introduce the *Lagrangian* function

$$\mathcal{L}(x, \lambda_1) \;=\; f(x) \;-\; \lambda_1 \, c_1(x)$$

Note that

$$\nabla_x \mathcal{L}(x, \lambda_1) \;=\; \nabla f(x) \;-\; \lambda_1 \nabla c_1(x)$$

Then, at the solution $x^*$, there is a scalar $\lambda_1^*$ s.t. $\nabla_x \mathcal{L}(x^*, \lambda_1^*) = 0$, or $\nabla f(x^*) = \lambda_1^* \nabla c_1(x^*)$.

Therefore, we can search for solutions of the equality-constrained optimization problem by searching for stationary points of the Lagrangian function. The scalar quantity $\lambda_1$ is called the *Lagrange multiplier* for the constraint $c_1(x) = 0$.

The condition $\nabla f(x^*) \;=\; \lambda_1^* \, \nabla c_1(x^*)$ is necessary for a solution but not sufficient. For example, in our previous problem, at (1,1) the condition is also satisfied, but $\lambda_1 = 1/2$ at (1,1) whereas $\lambda_1 = -1/2$ at the solution (-1,-1).

But, for equality constraints, we cannot make this a sufficient condition simply by putting restrictions of the sign of $\lambda$. Consider changing $c_1(x)$ to $2 - x_1^2 - x_2^2 = 0$. The solution does not change but $\lambda_1^*$ changes from -1/2 to 1/2. For inequality constrained problems, the sign of $\lambda_1$ plays a significant role.

In general, for more than one equality constraint, the Lagrangian for the constrained optimization problem $\min_{x \in \Omega} f(x)$ is defined as

$$\mathcal{L}(x, \lambda) \;=\; f(x) - \sum_{i \in E \cup I} \lambda_i \, c_i(x) \;=\; f(x) - \lambda^T \, c(x)$$

The *active set* $A(x)$ at any feasible $x$ is the union of the set $E$ with the indices of the active inequality constraints; i.e.,

$$A(x) \;=\; E \;\cup\; \{i \in I \mid c_i(x) = 0\}$$

First-Order Necessary Conditions

If $x^*$ is a local solution and the set of active gradients $\{\nabla c_i(x^*)\ ,\ i \in A(x^*)\}$ is linearly independent, then there exists $\lambda^*$ with components $\lambda_i^*\ ,\ i \in E \cup I$ s.t. at $(x^*, \lambda^*)$

$$\begin{aligned}
\nabla_x \mathcal{L}(x^*, \lambda^*) \;&=\; 0 \\
c_i(x^*) \;&=\; 0 \text{ for all } i \in E \\
c_i(x^*) \;&\leq\; 0 \text{ for all } i \in I \\
\lambda_i^* \;&\geq\; 0 \text{ for all } i \in I \\
\lambda_i^* \, c_i(x^*) \;&=\; 0 \text{ for all } i \in E \cup I
\end{aligned}$$

These conditions are known as the Karush-Kuhn-Tucker (KKT) conditions. The last condition is known as the *complimentarity condition*, which says that for each index $i \in I$, exactly one of $\lambda_i^*$ and $c_i(x^*)$ is zero.

## 8.2 Equality Constraints

Consider the constrained minimization problem

$$\min_{x \in \Re^n} \; f(x) \text{ subject to } c_i(x) = 0 \; , \; i = 1, \ldots, m$$

$$f \; : \; \Re^n \to \Re \qquad c_i \; : \; \Re^n \to \Re$$

If we define the following Lagrangian function on $n + m$ variables

$$\mathcal{L}(x, \lambda) \; = \; f(x) - \sum_{i=1}^{m} \lambda_i \, c_i(x)$$

then the constraint optimization problem can be cast as one of finding the critical points of $\mathcal{L}$ in $\Re^{n+m}$. To find the critical points of $\mathcal{L}$, we need the gradient of $\mathcal{L}$. The gradient of the Laplacian can be written as

$$\nabla \mathcal{L} \; = \; \begin{bmatrix} \frac{\partial \mathcal{L}}{\partial x_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial x_n} \\ \frac{\partial \mathcal{L}}{\partial \lambda_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial \lambda_m} \end{bmatrix} \; = \; \begin{bmatrix} \frac{\partial f}{\partial x_1} - \lambda_1 \frac{\partial c_1}{\partial x_1} - \cdots - \lambda_m \frac{\partial c_m}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} - \lambda_1 \frac{\partial c_1}{\partial x_n} - \cdots - \lambda_m \frac{\partial c_m}{\partial x_n} \\ -c_1 \\ \vdots \\ -c_m \end{bmatrix}$$

$$= \; \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \; - \; \lambda_1 \begin{bmatrix} \frac{\partial c_1}{\partial x_1} \\ \vdots \\ \frac{\partial c_1}{\partial x_n} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \; - \; \cdots \; - \; \lambda_m \begin{bmatrix} \frac{\partial c_m}{\partial x_1} \\ \vdots \\ \frac{\partial c_m}{\partial x_n} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \; - \; \begin{bmatrix} 0 \\ \vdots \\ 0 \\ c_1 \\ \vdots \\ c_m \end{bmatrix}$$

$$\nabla \mathcal{L} = \begin{bmatrix} \nabla f \\ \\ 0 \end{bmatrix} - \lambda_1 \begin{bmatrix} \nabla c_1 \\ \\ 0 \end{bmatrix} - \cdots - \lambda_m \begin{bmatrix} \nabla c_m \\ \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \\ c_1 \\ \vdots \\ c_m \end{bmatrix}$$

$$= \begin{bmatrix} \nabla f - \sum_{i=1}^{m} \lambda_i \nabla c_i \\ -c_1 \\ \vdots \\ -c_m \end{bmatrix}$$

Therefore, we would like to find a point $(x^*, \lambda^*)$ such that

$$\nabla \mathcal{L}(x^*, \lambda^*) = \begin{bmatrix} \nabla_x \mathcal{L}(x^*, \lambda^*) \\ \nabla_\lambda \mathcal{L}(x^*, \lambda^*) \end{bmatrix} = \begin{bmatrix} \nabla f(x^*) - \sum_{i=1}^{m} \lambda_i^* \nabla c_i(x^*) \\ -c_1(x^*) \\ \vdots \\ -c_m(x^*) \end{bmatrix} = \vec{0}$$

The importance of this is that we have converted a constrained optimization problem into an unconstrained root-finding problem.

We could apply Newton-like techniques for finding $(x^*, \lambda^*)$. Recall that, for the function f(x), the Newton iteration update is

$$x_{k+1} = x_k - \nabla^2 f(x_k)^{-1} \nabla f(x_k)$$

We can apply the Newton iteration approach to the Lagrangian function rather than just the $f(x)$ function by

$$(x_{k+1}, \lambda_{k+1}) = (x_k, \lambda_k) - \nabla^2 \mathcal{L}(x_k, \lambda_k)^{-1} \nabla \mathcal{L}(x_k, \lambda_k)$$

In order to use the full Newton's method, we need the Hessian of $\mathcal{L}$. The Hessian of the Laplacian is

$$\nabla^2 \mathcal{L} = \left[\begin{array}{ccc|ccc} \frac{\partial^2 \mathcal{L}}{\partial x_1^2} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial x_n} & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial \lambda_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_1 \partial \lambda_m} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial^2 \mathcal{L}}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_n^2} & \frac{\partial^2 \mathcal{L}}{\partial x_n \partial \lambda_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial x_n \partial \lambda_m} \\ \hline \frac{\partial^2 \mathcal{L}}{\partial \lambda_1 \partial x_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial \lambda_1 \partial x_n} & \frac{\partial^2 \mathcal{L}}{\partial \lambda_1^2} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial \lambda_1 \partial \lambda_m} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial^2 \mathcal{L}}{\partial \lambda_m \partial x_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial \lambda_m \partial x_n} & \frac{\partial^2 \mathcal{L}}{\partial \lambda_m \lambda_1} & \cdots & \frac{\partial^2 \mathcal{L}}{\partial \partial \lambda_m^2} \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc} \frac{\partial^2 f}{\partial x_1^2} - \sum_{i=1}^m \lambda_i \frac{\partial^2 c_i}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} - \sum_{i=1}^m \lambda_i \frac{\partial^2 c_i}{\partial x_1 \partial x_n} & -\frac{\partial c_1}{\partial x_1} & \cdots & -\frac{\partial c_m}{\partial x_1} \\ \vdots & & \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} - \sum_{i=1}^m \lambda_i \frac{\partial^2 c_i}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} - \sum_{i=1}^m \lambda_i \frac{\partial^2 c_i}{\partial x_n^2} & -\frac{\partial c_1}{\partial x_n} & \cdots & -\frac{\partial c_m}{\partial x_n} \\ \hline -\frac{\partial c_1}{\partial x_1} & \cdots & -\frac{\partial c_1}{\partial x_n} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ -\frac{\partial c_m}{\partial x_1} & \cdots & -\frac{\partial c_r}{\partial x_n} & 0 & \cdots & 0 \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc} & & & -\frac{\partial c_1}{\partial x_1} & \cdots & -\frac{\partial c_m}{\partial x_1} \\ & \nabla^2 f - \sum_{i=1}^m \lambda_i \nabla^2 c_i & & & \vdots & \\ & & & -\frac{\partial c_1}{\partial x_n} & \cdots & -\frac{\partial c_m}{\partial x_n} \\ \hline -\frac{\partial c_1}{\partial x_1} & \cdots & -\frac{\partial c_1}{\partial x_n} & & & \\ & \vdots & & & 0 & \\ -\frac{\partial c_m}{\partial x_1} & \cdots & -\frac{\partial c_m}{\partial x_n} & & & \end{array}\right]$$

$$= \left[\begin{array}{c|c} \nabla^2 f - \sum_{i=1}^m \lambda_i \nabla^2 c_i & -J_c^T \\ \hline -J_c & 0 \end{array}\right]$$

where $J_c$ is the $m \times n$ Jacobian matrix of the $c_i$, $\quad J_c = \left[\begin{array}{ccc} \frac{\partial c_1}{\partial x_1} & \cdots & \frac{\partial c_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial c_m}{\partial x_1} & \cdots & \frac{\partial c_m}{\partial x_n} \end{array}\right]$

Therefore, the Hessian of the Lagrangian, $\nabla^2 \mathcal{L}$, is given by the $(m+n) \times (m+n)$ matrix

$$\nabla^2 \mathcal{L} \;=\; \left[\begin{array}{c|c} \nabla^2 f \;-\; \sum_{i=1}^m \lambda_i \nabla^2 c_i & -J_c^T \\ \hline -J_c & 0 \end{array}\right]$$

Therefore, the Newton iteration for the Lagrangian function proceeds by updating $x_k$ and $\lambda_k$ as

$$\begin{aligned} x_{k+1} \;&=\; x_k \;+\; \Delta x_k \\ \lambda_{k+1} \;&=\; \lambda_k \;+\; \Delta \lambda_k \end{aligned}$$

where $\Delta x_k$ and $\Delta \lambda_k$ are found by solving the $m+n$ linear system

$$\left[\begin{array}{c|c} \nabla^2 f(x_k) \;-\; \sum_{i=1}^m \lambda_{k_i} \nabla^2 c_i(x_k) & -J_c(x_k)^T \\ \hline -J_c(x_k) & 0 \end{array}\right] \left[\begin{array}{c} \Delta x_k \\ \Delta \lambda_k \end{array}\right] \;=\; -\left[\begin{array}{c} \nabla f(x_k) \;-\; \sum_{i=1}^m \lambda_{k_i} \nabla c_i(x_k) \\ -c(x_k) \end{array}\right]$$

$$\left[\begin{array}{c|c} \nabla^2 f(x_k) \;-\; \sum_{i=1}^m \lambda_{k_i} \nabla^2 c_i(x_k) & -J_c(x_k)^T \\ \hline -J_c(x_k) & 0 \end{array}\right] \left[\begin{array}{c} \Delta x_k \\ \Delta \lambda_k \end{array}\right] \;=\; \left[\begin{array}{c} -\nabla f(x_k) \;+\; \sum_{i=1}^m \lambda_{k_i} \nabla c_i(x_k) \\ c(x_k) \end{array}\right]$$

We can interpret the Newton step as the solutions of a linearized approximation of the optimality conditions. Thus, the system above is a linear system, even though the constraint functions might be nonlinear.

We can write this system in a slightly different form by considering the $\Delta x_k$ equation

$$\left[\nabla^2 f(x_k) \;-\; \sum_{i=1}^m \lambda_{k_i} \nabla^2 c_i(x_k)\right] \Delta x_k \;-\; J_c(x_k)^T \, \Delta \lambda_k \;=\; -\nabla f(x_k) \;+\; \sum_{i=1}^m \lambda_{k_i} \nabla c_i(x_k) \quad (21)$$

$$\sum_{i=1}^m \lambda_i \nabla c_i \;=\; \lambda_1 \left[\begin{array}{c} \frac{\partial c_1}{\partial x_1} \\ \vdots \\ \frac{\partial c_1}{\partial x_n} \end{array}\right] \;+\; \cdots \;+\; \lambda_m \left[\begin{array}{c} \frac{\partial c_m}{\partial x_1} \\ \vdots \\ \frac{\partial c_m}{\partial x_n} \end{array}\right]$$

$$= \left[\begin{array}{c} \frac{\partial c_1}{\partial x_1} \lambda_1 \;+\; \cdots \;+\; \frac{\partial c_m}{\partial x_1} \lambda_m \\ \vdots \\ \frac{\partial c_1}{\partial x_n} \lambda_1 \;+\; \cdots \;+\; \frac{\partial c_m}{\partial x_n} \lambda_m \end{array}\right] \left[\begin{array}{ccc} \frac{\partial c_1}{\partial x_1} & \cdots & \frac{\partial c_m}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial c_1}{\partial x_n} & \cdots & \frac{\partial c_m}{\partial x_n} \end{array}\right] \left[\begin{array}{c} \lambda_1 \\ \vdots \\ \lambda_m \end{array}\right] \;=\; J_c^T \, \lambda$$

Therefore, Equation (21) is

$$\left[\nabla^2 f(x_k) \; - \; \sum_{i=1}^{m} \lambda_{k_i} \nabla^2 c_i(x_k)\right] \Delta x_k \; - \; J_c(x_k)^T \, \Delta\lambda_k \; = \; -\nabla f(x_k) \; + \; \sum_{i=1}^{m} \lambda_{k_i} \nabla c_i(x_k)$$

$$\left[\nabla^2 f(x_k) \; - \; \sum_{i=1}^{m} \lambda_{k_i} \nabla^2 c_i(x_k)\right] \Delta x_k \; - \; J_c(x_k)^T \, (\lambda_{k+1} \; - \; \lambda_k) \; = \; -\nabla f(x_k) \; + \; J_c(x_k)^T \, \lambda_k$$

$$\left[\nabla^2 f(x_k) \; - \; \sum_{i=1}^{m} \lambda_{k_i} \nabla^2 c_i(x_k)\right] \Delta x_k \; - \; J_c(x_k)^T \, \lambda_{k+1} \; = \; -\nabla f(x_k)$$

Our $m + n$ system is then

$$\left[\begin{array}{c|c} \nabla^2 f(x_k) \; - \; \sum_{i=1}^{m} \lambda_{k_i} \nabla^2 c_i(x_k) & -J_c(x_k)^T \\ \hline -J_c(x_k) & 0 \end{array}\right] \left[\begin{array}{c} \Delta x_k \\ \lambda_{k+1} \end{array}\right] \; = \; \left[\begin{array}{c} -\nabla f(x_k) \\ c(x_k) \end{array}\right] \tag{22}$$

**Example 8.2.1.** Solve the following equality constrained problem:

$$\min_{(x_1, x_2)} \; f(x_1, x_2) \; = \; x_1^2 \; + \; x_2^2 \; - \; 8\, x_1 \; - \; 6\, x_2$$

subject to the constraint $x_1 + x_2 \; = \; 5$ , by taking partial derivatives of the Lagrangian function, setting them equal to 0, and solving the resulting system of linear equations.

$$\begin{aligned} \mathcal{L} \; &= \; f(x_1, x_2) \; - \; \lambda \, c(x_1, x_2) \\ &= \; x_1^2 \; + \; x_2^2 \; - \; 8\, x_1 \; - \; 6\, x_2 \; - \; \lambda \, (x_1 \; + \; x_2 \; - \; 5) \end{aligned}$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_1} \; &= \; 2x_1 \; - \; 8 \; - \; \lambda \; = \; 0 \\ \frac{\partial \mathcal{L}}{\partial x_2} \; &= \; 2x_2 \; - \; 6 \; - \; \lambda \; = \; 0 \\ \frac{\partial \mathcal{L}}{\partial \lambda} \; &= \; -x_2 \; - \; x_2 \; + \; 5 \; = \; 0 \end{aligned}$$

This gives the system of equations

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ -5 \end{bmatrix}$$

Which has the solution

$$\begin{bmatrix} x_1 \\ x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}$$

Therefore, the solution to the equality constrained problem is

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Note that the $\lambda$ value is less than 0. This is not important for the equality constrained problem, but will be more significant for the inequality constrained problem.

**Example 8.2.2.** Solve the following equality constrained problem:

$$\min_{(x_1,x_2)} f(x_1, x_2) = x_1^2 + x_2^2 - 8\,x_1 - 6\,x_2$$

subject to the constraint $x_1 + x_2 = 5$ , by using Newton iteration starting at the point

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\mathcal{L} = f(x_1, x_2) - \lambda\, c_1(x_1, x_2) \qquad c(x_1, x_2) = x_1 + x_2 - 5$$

$$\mathcal{L} = x_1^2 + x_2^2 - 8\,x_1 - 6\,x_2 - \lambda\, c(x_1, x_2)$$
$$\mathcal{L} = x_1^2 + x_2^2 - 8\,x_1 - 6\,x_2 - \lambda\,(x_1 + x_2 - 5)$$

$$\nabla f_x = \begin{bmatrix} 2x_1 - 8 \\ 2x_2 - 6 \end{bmatrix}$$

$$\nabla^2 f_x = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

$$J_c = \begin{bmatrix} \frac{\partial c}{\partial x_1} & \frac{\partial c}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\nabla^2 c = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

From Equation (22), we get the system of equations

$$\left[\begin{array}{c|c} \nabla^2 f(x_k) - \sum_{i=1}^{m} \lambda_{k_i} \nabla^2 c_i(x_k) & -J_c(x_k)^T \\ \hline -J_c(x_k) & 0 \end{array}\right] \begin{bmatrix} \Delta x_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ c(x_k) \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} -2x_1 + 8 \\ -2x_2 + 6 \\ x_1 + x_2 - 5 \end{bmatrix}$$

Iteration 1

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ -5 \end{bmatrix}$$

$$\begin{bmatrix} \Delta x_1 \\ \Delta x_2 \\ \lambda \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

Note that the $\lambda$ value is less than 0. This is not important for the equality constrained problem, but will be more significant for the inequality constrained problem.

## 8.3 Inequality Constraints: Active Set Method

For the case of inequality constraints, we can solve by performing a series of subproblems, each of which involves equality constraints. We have defined an active set to be the set of equality constraints and those inequality constraints that we are treating as equality constraints. The active set method proceeds by defining, at each step, a working set of constraints, $W^k$, composed of an active set. The equality constrained problem is then solved. The resulting point is then checked to see if it lies in a feasible region and if so, whether it minimized the function. If it is not feasible or does not minimize the function, then the working set is adjusted and the algorithm proceeds. The following pseudocode describes the algorithm.

1. Let $x = x^0$, $W = W^0$
2. Let $x^*_{EQP}$ and $\lambda^*_{EQP}$ solve the equality constrained problem defined by $W^0$
3. IF $x^*_{EQP} \in \mathcal{F}$ (feasible region)
   - IF all $\lambda^*_{EQP} \leq 0$, STOP ($x^*_{EQP}$ is optimal)
   - ELSE
     - Remove from $W$ the constraint with the highest $\lambda^*_{EQP}$ value
     - Set $x = x^*_{EQP}$
     - Go to Step 2

   ELSE

   - Let $\alpha_{max}$ solve the problem

     Find the maximum $\alpha$ such that $x + \alpha \left( x^*_{EQP} - x \right) \in \mathcal{F}$
   - Set $x^{new} = x + \alpha_{max} \left( x^*_{EQP} - x \right)$
   - Add to $W$ on constraint that is binding at $x^{new}$ and that is violated by $x^*_{EQP}$
   - Set $x = x^{new}$
   - Go to Step 2

Note, this pseudocode assumes that the constrained problem at each step is solved by setting the partial derivatives equal to 0 and solving the resulting system of equations. You could also solve the constrained problem at each step by implementing a full set of Newton iterations at each active set step.

**Example 8.3.1.** Consider the following inequality constrained problem:

$$\min_{(x_1,x_2)} f(x_1,x_2) = x_1^2 + x_2^2 - 8\,x_1 - 6\,x_2$$

subject to the constraints

$$\begin{aligned} x_1 &\geq 0 \quad (1)\\ x_2 &\geq 0 \quad (2)\\ x_1 + x_2 &\leq 5 \quad (3) \end{aligned}$$

$$\begin{array}{rcl} x_1 \geq 0 \\ x_2 \geq 0 \\ x_1 + x_2 \leq 5 \end{array} \quad \Rightarrow \quad \begin{array}{rcl} -x_1 \leq 0 \\ -x_2 \leq 0 \\ x_1 + x_2 - 5 \leq 0 \end{array} \quad \Rightarrow \quad \begin{array}{rcl} c_1(x_1,x_2) &=& -x_1 \\ c_2(x_1,x_2) &=& -x_2 \\ c_3(x_1,x_2) &=& x_1 + x_2 - 5 \end{array}$$

Form the Lagrangian function

$$\begin{aligned} \mathcal{L} &= f(x_1,x_2) - \sum_{i=1}^{3} \lambda_i\, c_i(x_1,x_2) \\ \mathcal{L} &= x_1^2 + x_2^2 - 8\,x_1 - 6\,x_2 - \lambda_1\,(-x_1) - \lambda_2\,(-x_2) - \lambda_3\,(x_1 + x_2 - 5) \end{aligned}$$

Take partial derivatives

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial x_1} &= 2x_1 - 8 + \lambda_1 - \lambda_3 \\ \frac{\partial \mathcal{L}}{\partial x_2} &= 2x_2 - 6 + \lambda_2 - \lambda_3 \\ \frac{\partial \mathcal{L}}{\partial \lambda_1} &= x_1 \\ \frac{\partial \mathcal{L}}{\partial \lambda_2} &= x_2 \\ \frac{\partial \mathcal{L}}{\partial \lambda_3} &= -x_1 - x_2 + 5 \end{aligned}$$

Therefore the system of equations, including all the constraints, is

$$\begin{bmatrix} 2 & 0 & 1 & 0 & -1 \\ 0 & 2 & 0 & 1 & -1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 0 \\ 0 \\ -5 \end{bmatrix}$$

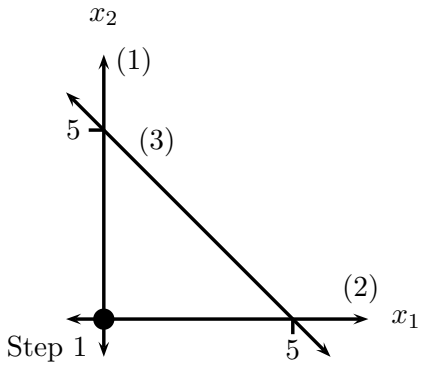
Step 1

Assume an initial active set $W^0 = \{1, 2\}$. The system of equations for these constraints is

$$\begin{bmatrix} 2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 0 \\ 0 \end{bmatrix}$$

The solution to this system of equations is $\begin{bmatrix} x_1 \\ x_2 \\ \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 8 \\ 6 \end{bmatrix}$

The point $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is feasible.



All of the $\lambda$'s are not $< 0$, therefore this point is not optimal. Drop constraint 1, which has the highest $\lambda$ value.

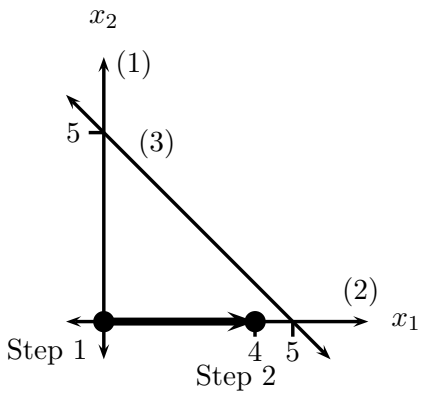
Step 2

The only active constraint is now (2); i.e., $W^2 = \{2\}$. The system of equations for these constraints is

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 0 \end{bmatrix}$$

The solution to this system of equations is $\begin{bmatrix} x_1 \\ x_2 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \\ 6 \end{bmatrix}$

The point $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$ is feasible.




All of the $\lambda$'s are not $< 0$, therefore this point is not optimal. Drop constraint 2, which has the highest $\lambda$ value.

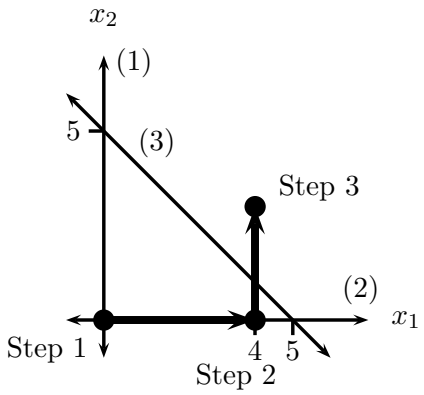
Step 3

There are no active constraints remaining; i.e., $W^3 = \{\emptyset\}$. The system of equations for the unconstrained problem is

$$\begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}$$

The solution to this system of equations is $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$

The point $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$ is not feasible.



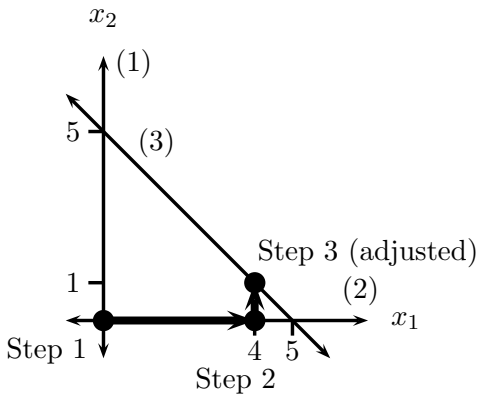

Therefore, we must adjust the point $\vec{x}^{\,3} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$. That is, we must find $\vec{x}^{\,3}_{adj} = \vec{x}^{\,2} + \alpha\left(\vec{x}^{\,3} - \vec{x}^{\,2}\right)$ where $\alpha$ is maximized. This will be the point where the vector $\left(\vec{x}^{\,3} - \vec{x}^{\,2}\right)$ intersects constraint (3); where $\vec{x}^{\,2} = \begin{bmatrix} 4 \\ 0 \end{bmatrix}$

$$\bar{x}^{\,3}_{adj} = \begin{bmatrix} x^3_{adj_1} \\ x^3_{adj_2} \end{bmatrix} = \bar{x}^2 + \alpha\left(\bar{x}^3 - \bar{x}^2\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + \alpha\left(\begin{bmatrix} 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 4 \\ 0 \end{bmatrix}\right)$$

$$\text{where} \quad x^3_{adj_1} + x^3_{adj_2} = 5 \qquad \text{(to satisfy constraint (3))}$$

$$\begin{aligned} 4 + \alpha(4 - 4) + 0 + \alpha(3 - 0) &= 5 \\ 4 + 3\alpha &= 5 \\ \alpha &= \frac{1}{3} \end{aligned}$$

$$\vec{x}^{\,3}_{adj} = \vec{x}^{\,2} + \alpha\left(\vec{x}^{\,3} - \vec{x}^{\,2}\right) = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + \frac{1}{3}\left(\begin{bmatrix} 4 \\ 3 \end{bmatrix} - \begin{bmatrix} 4 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$




Constraint (3) now becomes active; i.e., $W^3 = \{3\}$.

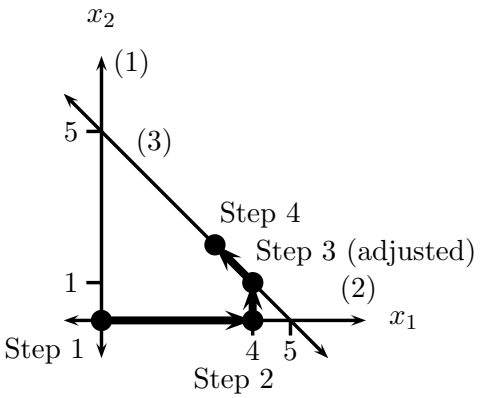
Step 4

For $W^3 = \{3\}$, the system of equations is

$$\begin{bmatrix} 2 & 0 & -1 \\ 0 & 2 & -1 \\ -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ -5 \end{bmatrix}$$

The solution to this system of equations is $\begin{bmatrix} x_1 \\ x_2 \\ \lambda_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ -2 \end{bmatrix}$

The point $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ is feasible.




All of the $\lambda$'s are $< 0$, therefore the point $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ is optimal subject to the given constraints.

## 8.4 Penalty Methods

We can attempt to minimize an objective function subject to constraints by adding a term to the objective function that prescribes a high cost for violation of the constraints. This approach is referred to as a *penalty method*.

Consider our constrained minimization problem $\min \{f(x) : x \in S\}$, where $f$ is a continuous function on $\Re^n$ and $S$ is a constraint set in $\Re^n$.

The idea of a penalty function method is to replace $f(x)$ with $f(x) + c\,P(x)$, where $c$ is a positive constant and $P$ is a function on $\Re^n$ satisfying

(i) $P(x)$ is continuous

(ii) $P(x) \geq 0$ for all $x \in \Re^n$

(iii) $P(x) = 0$ if and only if $x \in S$

Suppose $S$ is defined by a number of inequality constraints

$$S \;=\; \{x : g_i(x) \;\leq\; 0 \;,\; i = 1, \ldots, m\}$$

A very useful penalty function in this case is

$$P(x) \;=\; \frac{1}{2} \sum_{i=1}^{m} \left(\max\{0, g_i(x)\}\right)^2$$

which gives a quadratic augmented objective function denoted by

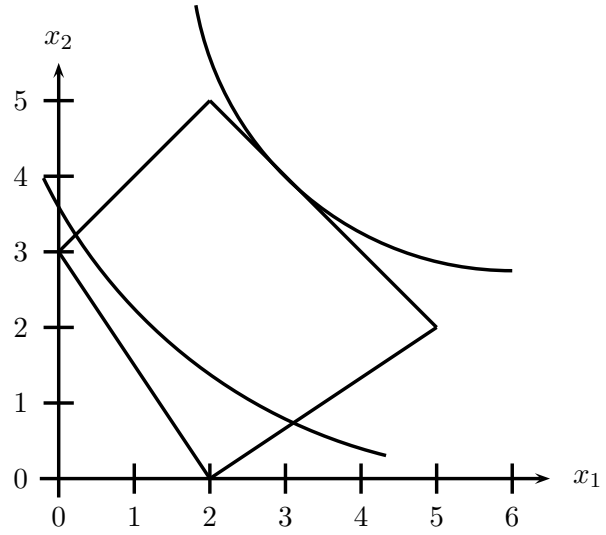
$$\Theta(c, x) \;=\; f(x) \;+\; c\, P(x)$$

Here, each unsatisfied constraint influences $x$ by assessing a penalty equal to the square of the violation. These influences are summed and multiplied by $c$, the *penalty parameter*. This influence is counterbalanced by $f(x)$. If the magnitude of the penalty term is small relative to the magnitude of $f(x)$, minimization of $\Theta(c, x)$ will almost certainly not result in an $x$ that is feasible with respect to the constraints. However, if the value of $c$ is made suitably large, the penalty term will exact such a heavy cost for any constraint violation that the minimization of the augmented objective function might not yield a feasible solution.

**Example 8.4.1.** Consider the following example

$$\begin{aligned}
\text{Minimize } f(x) \;&=\; (x_1 \;-\; 6)^2 \;+\; (x_2 \;-\; 7)^2 \\
\text{subject to } g_1(x) \;&=\; -3\, x_1 \;-\; 2\, x_2 \;+\; 6 \;\leq\; 0 \\
g_2(x) \;&=\; -x_1 \;+\; x_2 \;-\; 3 \;\leq\; 0 \\
g_3(x) \;&=\; x_1 \;+\; x_2 \;-\; 7 \;\leq\; 0 \\
g_4(x) \;&=\; \frac{2}{3}\, x_1 \;-\; x_2 \;-\; \frac{4}{3} \;\leq\; 0
\end{aligned}$$

The feasible region is shown graphically in the following figure, along with several isovalue contours for the objective function. The problem is a quadratic program and the isovalue contours are concentric circles centered at (6,7), the unconstrained minimum of $f(x)$.




Using the quadratic penalty function, the augmented objective function is

$$\begin{aligned}\Theta(c,x) \;=\; (x_1 \,-\, 6)^2 \,+\, (x_2 \,-\, 7)^2 \,+\, \frac{1}{2}\,c\,\Big[&(\max\{0, -3\,x_1 \,-\, 2\,x_2 \,+\, 6\})^2 \\ &+\, (\max\{0, -x_1 \,+\, x_2 \,-\, 3\})^2 \,+\, (\max\{0, x_1 \,+\, x_2 \,-\, 7\})^2 \\ &+\, (\max\{0, \frac{2}{3}\,x_1 \,-\, x_2 \,-\, \frac{4}{3}\})^2\Big]\end{aligned}$$

The first step in the solution process is to select a starting point. A good rule of thumb is to start at an infeasible point. By design then, we will see that every trial point, except the *last* one, will be infeasible (exterior to the feasible region).

A reasonable place to start is at the unconstrained minimum. So we set $x^0 = (6, 7)$. Since only constraint 3 violated at this point, we have

$$\Theta(c,x) \;=\; (x_1 \,-\, 6)^2 \,+\, (x_2 \,-\, 7)^2 \,+\, \frac{1}{2}\,c\,(\max\{0, x_1 \,+\, x_2 \,-\, 7\})^2$$

Assuming that in the neighborhood of $x^0$, the "max" operator returns the constraint if that constraint is violated, the gradient with respect to $x$ is

$$\nabla_x \Theta(c,x) \;=\; \begin{bmatrix} 2\,x_1 \;-\; 12 \;+\; c\,(x_1 \;+\; x_2 \;-\; 7) \\ 2\,x_2 \;-\; 14 \;+\; c\,(x_1 \;+\; x_2 \;-\; 7) \end{bmatrix}$$

Setting the elements of $\nabla_x \Theta(c,x)$ to zero and solving yields the stationary point

$$x_1^*(c) \;=\; \frac{3\,(c+2)}{c+1} \quad \text{and} \quad x_2^*(c) \;=\; \frac{4c+7}{c+1}$$

For any positive value of $c$, $\Theta(c,x)$ is a strictly convex function (the Hessian of $\Theta(c,x)$ is positive definite for all $c > 0$), so $x_1^*(c)$ and $x_2^*(c)$ determine a global minimum. It turns out for this example that the minima will continue to satisfy all but the third constraint for all positive values of $c$. If we take the limit of $x_1^*(c)$ and $x_2^*(c)$ as $c \to \infty$, we obtain $x_1^* = 3$ and $x_2^* = 4$, the constrained global minimum for the original problem.

Because the above approach seems to work so well, it is natural to conjecture that all we have to do is set $c$ to a very large number and then optimize the resulting augmented objective function $\Theta(c,x)$ to obtain the solution to the original problem. Unfortunately, this conjecture is not correct. First, "large" depends on the particular model. It is almost always impossible to tell how large $c$ must be to provide a solution to the problem without creating numerical difficulties in the computations. Second, in a very real sense, the problem is dynamically changing with the relative position of the current value of $x$ and the subset of the constraints that are violated.

The third reason why the conjecture is not correct is associated with the fact that large values of $c$ create enormously steep valleys at the constraint boundaries. Steep valleys will often present formidable if not insurmountable convergence difficulties for all preferred search methods unless the algorithm starts at a point extremely close to the minimum being sought.

Fortunately, there is a direct and sound strategy that will overcome each of the difficulties mentioned above. All that needs to be done is to start with a relatively small value of $c$ and an infeasible (exterior) point. This will assure that no steep valleys are present in

the initial optimization of $\Theta(c, x)$. Subsequently, we will solve a sequence of unconstrained problems with monotonically increasing values of $c$ chosen so that the solution to each new problem is "close" to the previous one. This will preclude any major difficulties in finding the minimum of $\Theta(c, x)$ from one iteration to the next.

To implement this strategy, let $\{c_k\}, k = 1, 2, \ldots$ be a sequence tending to infinity such that $c_k > 0$ and $c_{k+1} > c_k$. Now for each $k$ we solve the problem $\min \{\Theta(c_k, x) : x \in \Re^n\}$ to obtain $x^k$, the optimum. It is assumed that this problem has a solution for all positive values of $c_k$. This will be true, for example, if $\Theta(c, x)$ increases without bounds as $\|x\| \to \infty$.

A simple implementation known as the sequential unconstrained minimization technique (SUMT) is given below

*Initialization Step*: Select a growth parameter $\eta > 1$, a stopping parameter $\epsilon > 0$, and an initial value of the penalty parameter $c_0$.

Choose a starting point $x^0$ that violates at least one constraint and formulate the augmented objective function $\Theta(c_0, x)$. Let $k = 1$.

*Iterative Step*: Starting from $x^{k-1}$, use an unconstrained search technique to find the point that minimizes $\Theta(c_{k-1}, x)$. Call this point $x^k$ and determine which constraints are violated at this point.

*Stopping Rule*: If the distance between $x^{k-1}$ and $x^k$ is smaller than $\epsilon$ (i.e., $\|x^{k-1} - x^k\| < \epsilon$) or the difference between two successive objective function values is smaller than $\epsilon$ (i.e., $|f(x^{k-1}) - f(x^k)| < \epsilon$), stop with $x^k$ an estimate of the optimal solution. Otherwise, let $c_k = \eta\, c_{k-1}$, formulate the new $\Theta(c_k, x)$ based on which constraints are violated at $x^k$, let $k = k + 1$ and return to the iterative step.

Applying this algorithm to our previous example with $\eta = 2$ and $c_0 = 0.5$ for eight iterations yields the sequence of solutions given in the following table. The iterates $x^k$ are seen to approach the true minimum point $x^* = (3, 4)$.

| $k$ | $c$ | $x_1$ | $x_2$ | $g_3$ |
|---|---|---|---|---|
| 0 | — | 6.00 | 7.00 | 6.00 |
| 1 | 0.5 | 4.50 | 5.50 | 3.00 |
| 2 | 1 | 4.00 | 5.00 | 2.00 |
| 3 | 2 | 3.60 | 4.60 | 1.20 |
| 4 | 4 | 3.33 | 4.33 | 0.66 |
| 5 | 8 | 3.18 | 4.18 | 0.35 |
| 6 | 16 | 3.09 | 4.09 | 0.18 |
| 7 | 32 | 3.05 | 4.05 | 0.09 |
| 8 | 64 | 3.02 | 4.02 | 0.04 |

## 8.5 Barrier Methods

It should be apparent that the quadratic penalty function is only one of an endless set of possibilities. While the "sum of squared violations" is probably the best type of penalty function for an *exterior method*, it is not at all suitable if one desires to conduct a search through a sequence of feasible or *interior* points. Let us consider the following augmented objective function

$$B(r, x) \;=\; f(x) \;+\; r \sum_{i=1}^{m} \frac{-1}{g_i(x)} \tag{23}$$

where $r > 0$ is the barrier parameter. This function is valid only for interior points such that all constraints are strictly satisfied: $g_i(x) < 0$ for all $i$.

Equation (23) indicates that the closer one gets to a constraint boundary, the larger $B(r, x)$ becomes. $B(r, x)$ is not defined for points precisely on any boundary. $B(r, x)$ is often called a *barrier function* and, in a sense, is opposite to the kind of exterior penalty function introduced in the previous section.

The basic idea of interior point methods is to start with a feasible point and a relatively *large* value of the paramter $r$. This will prevent the algorithm from approaching the boundary of the feasible region. At each subsequent iteration, the value of $r$ is monotonically *decreased* in such a way that the resultant problem is relatively easy to solve if the optimal solution of its immediate predecessor is used as the starting point.

**Example 8.5.1.** Consider the following example

$$\begin{aligned} \text{Minimize } f(x) &= 2\,x_1^2 + 9\,x_2 \\ \text{subject to } g_1(x) &= -x_1 - x_2 + 4 \leq 0 \end{aligned}$$

Using (23), the augmented objective function is

$$B(r,x) = 2\,x_1^2 + 9\,x_2 + r\left(\frac{-1}{-x_1 - x_2 + 4}\right)$$

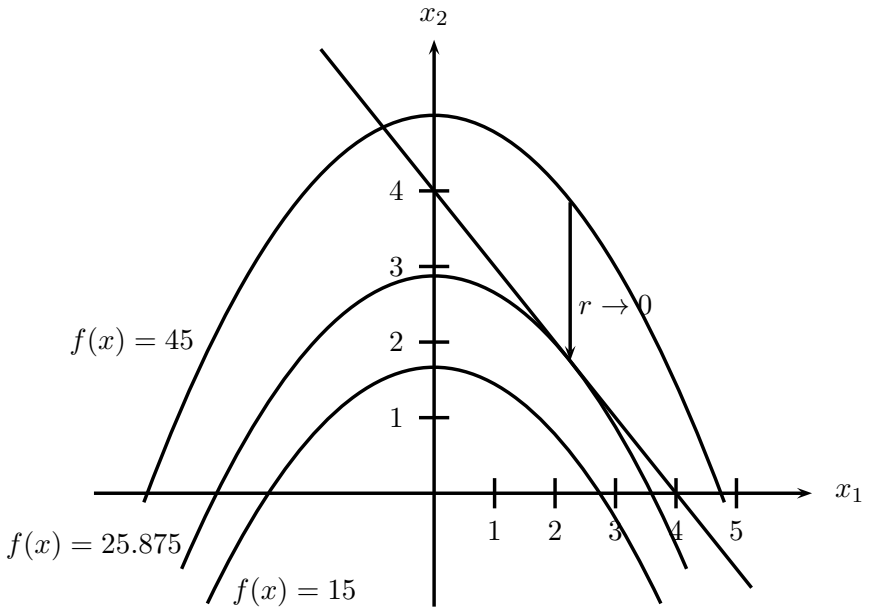
with gradient

$$\nabla_x B(r,x) = \begin{bmatrix} 4\,x_1 - r\,(-x_1 - x_2 + 4)^{-2} \\ 9 - r\,(-x_1 - x_2 + 4)^{-2} \end{bmatrix}$$

Setting the gradient vector to zero and solving for the stationary point yields

$$x_1(r) = 2.25 \quad \text{and} \quad x_2(r) = 0.333\sqrt{r} + 1.75 \quad \text{for all } r > 0$$

In the limit as $r$ approaches 0, these values become $x_1 = 2.25$ and $x_2 = 1.75$ with $f(x) = 25.875$ which is the optimal solution to the original problem. The following figure depicts the feasible region and several isovalue contours of $f(x)$.



## 8.6 Mixed Barrier-Penalty Methods

Equality constraints, though not discussed up until now, can be handled efficiently with a penalty function. Consider the following problem

$$\begin{aligned} &\text{Minimize } f(x) \\ &\text{subject to } h_i(x) \;=\; 0\ ,\ i = 1, \ldots, p \\ &\qquad\qquad g_i(x) \;\leq\; 0\ ,\ i = 1, \ldots, m \end{aligned}$$

The most common approach to implementing the sequential unconstrained minimization technique for this model is to form the *logarithmic-quadratic loss function*

$$LQ(x)(r_k, x) \;=\; f(x) \;-\; r_k \sum_{i=1}^{m} \ln(-g_i(x)) \;+\; \frac{1}{r_k} \sum_{i=1}^{p} h_i(x)^2$$

This algorithm finds the unconstrained minimizer of $LQ(r_k, x)$ over the set $\{x : g_i(x) < 0\ ,\ i = 1, \ldots, m\}$ for a sequence of scalar parameters $\{r_k\}$ strictly decreasing to zero. Note that for a given $r$, the stationarity condition is

$$\nabla f(x) \;-\; r_k \sum_{i=1}^{m} \frac{1}{g_i(x)} \nabla g_i(x) \;+\; \frac{2}{r_k} \sum_{i=1}^{p} h_i(x) \nabla h_i(x) \;=\; 0$$

This method will proceed iteratively, where the logarithmic-quadratic loss function at each iteration will contain only use the terms that correspond to violated constraints. It can be shown that as $r_k \to 0$, $\frac{r_k}{g_i(x)} \to \mu_i^*$, the optimal Lagrange multiplier for the $i$th inequality constraint, and $\frac{2\, h_i(x)}{r_k} \to \lambda_i^*$, the optimal Lagrange multiplier for the $i$th equality constraint. This result is suggested by the fractions in the above summations.

Penalty and barrier methods are among the most powerful class of algorithms available for attacking general nonlinear optimzation problems. This statement is supported by the fact that these techniques will converge to at least a local minimum in most cases, regardless of the convexity characteristics of the objective function and constraints. They work well even in the presence of cusps and similar anomalies that can hinder other approaches.

Of the two classes, the exterior methods are considered preferable. The primary reasons are

1. Interior methods cannot deal with equality constraints without cumbersome modifications to the basic approach.
2. Interior methods demand a feasible starting point. Finding such a point often presents formidable difficulties in and of itself.
3. Interior methods require that the search never leave the feasible region. This significantly increases the computational effort associated with the line search segment of the algorithm.

Although penalty and barrier methods met with great initial success, their slow rates of convergence due to ill-conditioning of the associated Hessian led researchers to pursue other approaches. With the advent of interior point methods for linear programming, algorithm designers have taken a fresh look at penalty methods and have been able to achieve much greater efficiency that previously thought possible.

# 9 Global Optimization

Many nonlinear optimization problems contain locally optimal states. Some locally optimal states can have relatively low error yet not the globally minimum error. Global optimization methods seek to find the global optimum state among all the locally optimal states, either by transforming the problem to remove some of the local minimum states, or by providing some sort of perturbation (often random or stochastic) to sample more of the search space. Examples of global optimization methods include smoothing and continuation, implicit filtering, branch and bound, genetic algorithms, and simulated annealing.

## 9.1 Smoothing and Continuation Methods

The idea behind smoothing and continuation methods is that by smoothing a rugged function profile, most or all of the local minima will disappear, and the remaining major

features of the profile only show a single minimizer. Types of smoothing include Gaussian smoothing and artificial diffusion. For example, artificial diffusion takes a function $f(x)$ and creates a $F(x)$ function which is the solution to the initial value diffusion problem

$$\begin{aligned} \frac{\partial^2 F}{\partial x^2}(x,t) &= \frac{\partial F}{\partial t}(x,t) \\ F(x,0) &= f(x) \end{aligned}$$

$F(x,t)$ gets smoother as $t$ gets larger. For large enough $t$, $F(x,t)$ is even unimodal, thus $F(x,t)$ can be minimized by local methods when $t$ is sufficiently larger. Using the minimizer at a given $t$ as the starting point for a local optimization at a smaller $t$, a sequence of local minimizers of $F(x,t)$ for successively smaller $t$ is obtained until finally, with $t = 0$, a minimizer of $f(x)$ is obtained. This process is known as *continuation*.

## 9.2 Implicit Filtering

Implicit filtering is a technique that is useful for generally convex functions containing high-frequency noise. It is similar to a continuation process, but does not smooth the function. Instead of a decreasing sequence of smoothing procedures, implicit filtering uses a decreasing sequence of step sizes to calculate finite difference gradients, which are then used in a gradient-based search procedure.

Starting with a relatively large step size, you would then perform an entire optimization using finite difference gradients and secant-based Hessians. Then you would decrease the step size and use the local minimum from the previous step size optimization as a starting point for an optimization using the new step size. The idea is that the earlier, larger step sizes will overlook relatively small noise and move close to the global minimum, then as the step size is decreased, the exact global minimum will be found, taking into account the local noise.

The entire algorithm can be restarted, using the results from the previous run as starting values for the next run. The following is pseudocode to describe the implicit filtering method with restarts. This pseudocode assumes that $\epsilon$ is decreased by diving by 2 at each

step. Locally optimal values of $x$ can be calculated by using Newton method, quasi-Newton methods, etc.

function implicitFilter($\Delta F_{tol}$,$maxnrestarts$,$\epsilon_{min}$,$\epsilon_{max}$)
    Generate starting $x$ values
    $F_0 = f(x)$
    FOR $restart = 0$ to $maxnrestarts$
        $\epsilon = \epsilon_{max}$
        DO
            Calculate locally optimal $x$ using $\epsilon$ for finite difference gradient
            $\epsilon = \epsilon$ / 2
        WHILE $\epsilon > \epsilon_{min}$
        $F = f(x)$
        $\Delta F = F - F_0$
        $F_0 = F$
        IF $|\Delta F| \leq \Delta F_{tol}$ break

## 9.3 Branch and Bound Methods

Branch and bound methods are general search methods where upper and lower bounds are calculated and the problem is divided into subregions with these bounds. The search terminates when the bounds match. The algorithm is applied recursively to the subregions, generating a tree of subproblems. If an optimal solution is found to a subproblem, it is a feasible solution to the full problem, but not necessarily globally optimal. A feasible solution can be used to prune the rest of the tree. The search continues until all nodes have been solved or pruned.

## 9.4 Genetic (Evolutionary) Algorithms

Genetic algorithms are based on a biological metaphor in that they view learning as a competition among a population of evolving candidate problem solutions. A fitness function evaluates each solution to decide whether it will contribute to the next generation of

solutions. Through operations analogous to gene transfer in reproduction, the algorithm creates a new population of candidate solutions. Genetic algorithms are a good way of sampling much of the search space in an efficient manner. They are a form of *machine learning*. The following outlines the genetic algorithm procedure:

1. Randomly generate an initial population, $M(0)$
2. Compute and save the fitness $u(m)$ for each individual $m$ in the current population $M(t)$
3. Define selection probabilities $p(m)$ for each individual $m$ in $M(t)$ so that $p(m)$ is proportional to $u(m)$
4. Generate $M(t+1)$ by probabilistically selecting individuals from $M(t)$ to produce offspring via genetic operators(reproduction, mutation, recombination, etc.)
5. Repeat from Step 2 until solution is obtained

## 9.5 Simulated Annealing

Simulated annealing is analogous to the cooling of metals (annealing). At high temperatures, the metal is disordered. As it is slowly cooled, maintaining approximate thermodynamic equilibrium, the system becomes more ordered and approaches a minimum energy state. If the initial temperature is too low or the cooling is done too quickly, the system may become quenched in a metastable state; i.e. trapped in a local minimum energy state.

The simulated annealing algorithm is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system. This method proceeds as follows

- Given an initial state with energy $E$ and temperature $T$
- Perturb the system and calculate the change in energy, $\Delta E$
- If $\Delta E < 0$, accept new state

- If $\Delta E > 0$, accept with probability $e^{-\Delta E/T}$
- Decrease $T$ and repeat

Note that the probability of accepting a higher energy state

- Decreases as $\Delta E$ increases
- Decreases as $T$ decreases

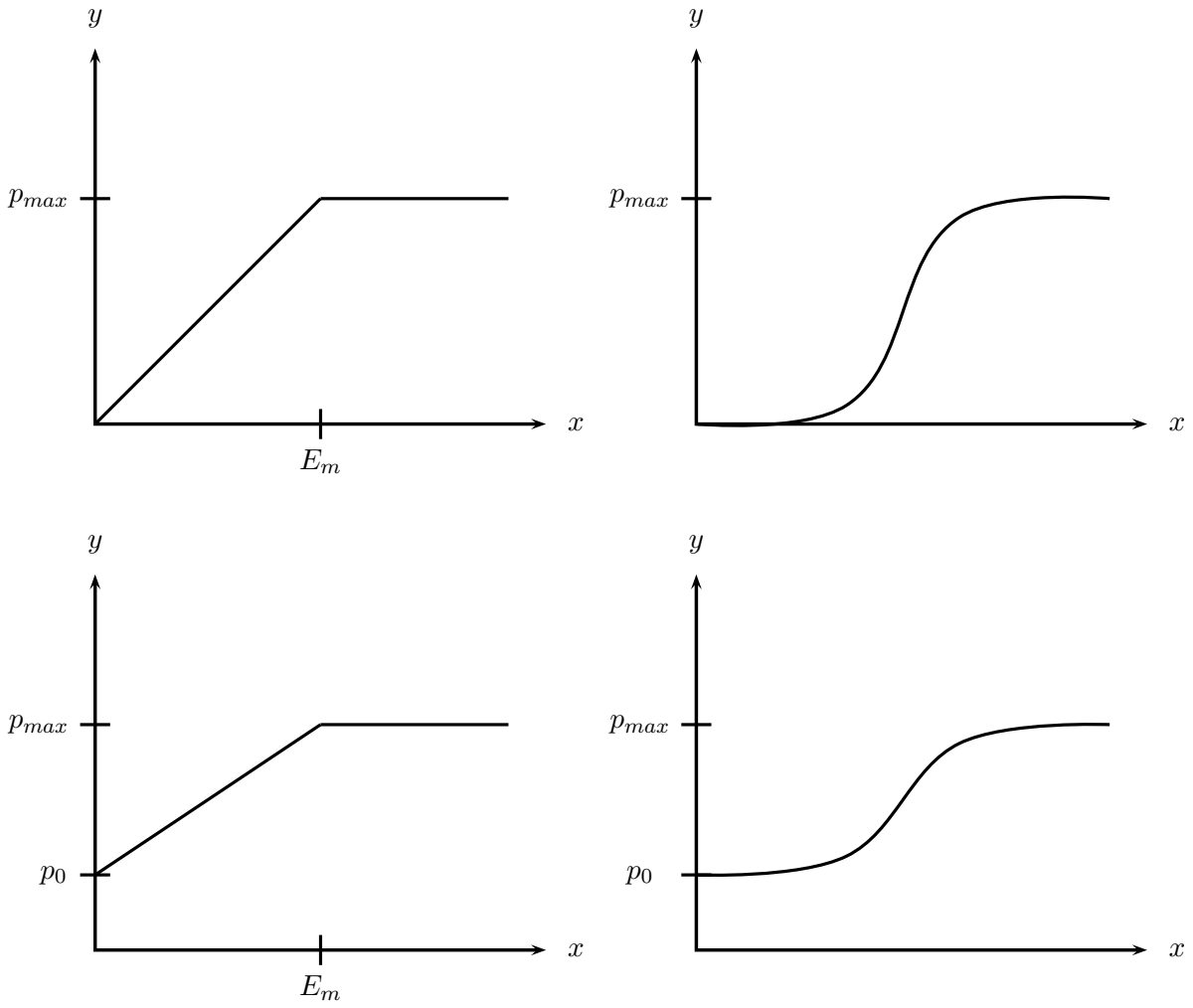
Application to Nonlinear Optimization

There are different ways to implement this within a nonlinear optimization code. The simulated annealing algorithm will essentially serve as a wrapper around a local optimization algorithm (quasi-Newton, etc.). It will introduce arbitrary parameters and requires the definition of two functions

- A perturbation function
- An annealing schedule (how $T$ is decreased)

Perturbation functions can be general, random perturbations, or can be based on relative error; e.g., if error can be attributed to each component of the system, then let the magnitude of the perturbation be directly related to attributed error.

Error-Based Perturbation Functions




Each component of the state vector, $x$, would be perturbed according to

$$x_i \;=\; x_i \;+\; r_i\, p$$

where $p$ is found from above and $r_i$ is a random number between -1 and 1.

Annealing Schedules

We want to decrease $T$ as the step number increases. The term "step" in this case will refer to a step of the simulated annealing process (i.e., each step could contain a number of nonlinear iterations). We could use any function that starts at some $T_0$ and decreases. Exponential-based functions are often used, such as

$$T = T_0\, e^{-\lambda(step)} \qquad \text{where } \lambda \text{ is some constant} > 0$$

Using the linear-based perturbation function, starting at $p_0$, and the above $T$ function, 5 arbitrary parameters are introduced: $p_0$, $p_{max}$, $E_m$, $T_0$, and $\lambda$. The following is pseudocode to describe the simulated annealing method

function SA($x, p_0, p_{max}, E_m, T_0, \lambda, maxnsteps$)
    $x = x + s^{QN}$ (quasi-Newton step update)
    $x_0 = x$
    $E_0 = \text{error}(x_0)$
    $E = E_0$
    WHILE $step \leq maxnsteps$
        $x = \text{perturbation}(x, E)$
        Calculate a locally optimal $x$ (using, for example, a quasi-Newton method)
        $E = \text{error}(x)$
        $\Delta E = \|E - E_0\|$
        IF $\Delta E < 0$
            $x_0 = x$
            $E_0 = E$
        ELSE
            $r =$ random number such that $0 < r < 1$
            $T = T_0\, e^{-\lambda(step)}$
            IF $e^{-\Delta E/T} > r$
                $x_0 = x$
                $E_0 = E$
            ELSE
                $x = x_0$
                $E = E_0$

The perturbation function in this case is

function perturbation$(x, E, E_m)$

FOR $i = 1 : n$ ($n$ = length of $x$, $E$)

IF $E_i < E_m$

$$p = p_0 + \left(\frac{p_{max} - p_0}{E_m}\right) E_i$$

ELSE

$$p = p_{max}$$

$r$ = random number such that $-1 < r < 1$

$x_i = x_i + r\,p$