# Unconstrained Optimization Review

## 1 Eigenvalues & Eigenvectors

Let $A$ be a real symmetric matrix:

**1.** If the eigenvalues of $A$ are all strictly positive $\lambda_i > 0$, then the matrix $A$ is called positive definite. This is also equivalent to demonstrating that $x^T Ax \geq \alpha x^T x$ for some $\alpha > 0$.

**2.** If the eigenvalues of $A$ are non-negative such that $\lambda_i \geq 0$, then the matrix $A$ is called positive semi-definite. Note that since we possibly have $\lambda_i = 0, A$ may be singular when $A$ is only positive semi-definite.

**3.** If the eigenvalues of $A$ are both positive and negative, then $A$ is called indefinite.

**4.** If $\lambda_i < 0$ then $A$ is negative definite.

**5.** If $\lambda_i \leq 0$ then $A$ is negative semi-definite.

General Note: If any eigenvalue of a matrix is zero ($\lambda_i = 0$ for some $i$), then the matrix is singular.

Example 1: A is a real, symmetric matrix: $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$

To get the eigenvalues, use: $det(A - \lambda I) = 0$

$$det \begin{bmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{bmatrix} = 0$$

$$\begin{gathered} (2-\lambda)(2-\lambda) - 1 = 2 \\ 4 - 4\lambda + \lambda^2 - 1 = 0 \\ \lambda^2 - 4\lambda + 3 = 0 \\ (\lambda - 3)(\lambda - 1) = 0 \\ \lambda_1 = 3 \quad \lambda_2 = 1 \end{gathered}$$

To get the eigenvectors, use: $(A - \lambda_i I)\, q_i = 0$

$$\begin{bmatrix} 2-\lambda_i & 1 \\ 1 & 2-\lambda_i \end{bmatrix} q_i = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

For $\lambda_1 = 3$, we get $q_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, which we normalize to $q_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$.
For $\lambda_2 = 1$, we get $q_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$, which we normalize to $q_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$.

## 2 Root Finding: Newton's Method

Newton's Method is an approach for finding the roots of **nonlinear equations**. It is significant because it is one of the most common root-finding algorithms due to its relative simplicity and speed. In optimization, we're interested in determining the x-values at which $g(x)$ crosses zero where we define $g(x) = f'(x)$. Therefore, solving for the roots of $g(x)$ gives the $x_{cr}$ points for the original function $f(x)$.

Newton's method is an iterative method that begins with an initial guess $x_0$ of the critical point(s). The method uses the derivative of the function $g'(x)$ and function $g(x)$, and thus only works when the derivative can be determined.

Newton's Method (single variable):

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)} \tag{1}$$

Need:

$$\frac{d}{dx}[f(x)] = 0 \tag{2}$$

Therefore, set:

$$g = \frac{d}{dx}[f(x)] \tag{3}$$

To use Newton's method, make an initial guess $x_0$ of the critical point. To make a good estimate, we can plot the function and look at the locations of the critical points to serve as initial guesses.

Example 1: Minimize transcendental function (find $x_{cr}$)

$$f(x) = \frac{1}{2}\cos\left(\frac{3\pi x}{2}\right) - 3x + 2\exp(x) - \frac{7}{2}$$

$$x_{n+1} = x_n - \frac{g\left(x_n\right)}{g'\left(x_n\right)}$$

$$g(x) = \frac{d}{dx}[f(x)] = -\frac{3\pi}{4}\sin\left(\frac{3\pi x}{2}\right) - 3 + 2\exp(x)$$

$$g'(x) = \frac{d^2}{dx^2}[f(x)] = -\frac{9\pi^2}{8}\cos\left(\frac{3\pi x}{2}\right) + 2\exp(x)$$

After plotting f(x), make initial guess $x_0 = -0.5$, using Newton's method get an $x_{cr} = -0.486$. But! If chose $x_0 = +0.5$ and using Newton's method, get an $x_{cr} = 0.605$.

Newton's Method (two variables):

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - \left(H(f)\left(\vec{x}^{(k)}\right)\right)^{-1}(\nabla f)\left(\vec{x}^{(k)}\right) \tag{4}$$

OR follow the steps:

1.) Find $\nabla[f(x)]$

2.) Set $\nabla[f(x)] = 0$ to solve for the critical points $x_{cr}$

3.) Select the critical point(s) for which the Hessian matrix, $H\left[f\left(x_{cr}\right)\right]$ is positive definite (negative definite for maximization). (Sometimes this step does not work - limitation!)

Example 2: Minimize 2-variable function (find $x_{cr}$)

Minimize $f\left(x_1, x_2\right) = \frac{3}{2}x_1^2 + \frac{1}{2}x_1x_2 + 2x_2^2 - \frac{1}{2}x_1 + \frac{1}{2}x_2 - 2$

$$\nabla[f(x)] = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix} = \mathbf{0} \Rightarrow \begin{array}{l} \frac{\partial f}{\partial x_1} = 3\mathrm{x}_1 + \frac{1}{2}x_2 - \frac{1}{2} = 0 \\ \frac{\partial f}{\partial x_2} = \frac{1}{2}x_1 + 4x_2 + \frac{1}{2} = 0 \end{array}$$

$$\begin{bmatrix} 3 & 1/2 \\ 1/2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ -1/2 \end{bmatrix} \Rightarrow \begin{bmatrix} x_{1,cr} \\ x_{2,cr} \end{bmatrix} \approx \begin{bmatrix} 0.19 \\ -0.15 \end{bmatrix}$$

Okay, we have one critical point.(How would we know if a critical point did not exist?) Let's now determine whether the critical point is a minimum.

To do so, first calculate the Hessian matrix:

$$H(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix} = \begin{bmatrix} 3 & 1/2 \\ 1/2 & 4 \end{bmatrix}$$

Next, calculate its characteristic polynomial:

$$det(\lambda I - H) = 0 \Rightarrow det \begin{bmatrix} \lambda - 3 & -1/2 \\ -1/2 & \lambda - 4 \end{bmatrix} = 0$$

$$det \begin{bmatrix} \lambda - 3 & -1/2 \\ -1/2 & \lambda - 4 \end{bmatrix} = 0 \quad \Rightarrow \quad \lambda^2 - 7\lambda + \frac{47}{4} = 0 \Rightarrow \quad \lambda \approx \begin{bmatrix} 4.21 \\ 2.79 \end{bmatrix}$$

Since both eigenvalues are positive, $H(x)$ is positive definite. The critical point is therefore a (local) minimum.

**Issues:** What if there are multiple local minima (e.g. periodic sin(x))? What if we're only interested in the global minimum? We can make a plot, but it might not be that simple due to added constraints, multiple variables ($x_1, ..., x_n$, higher order polynomial, or transcendental function.

## 3 Conditions for Optimality

### 3.1 First-Order Necessary Condition

$\nabla f(x^*) = 0$ is necessary for $x^*$ to be a local solution to the unconstrained minimization problem. In which:

$x^*$ is a (weak) local solution to the minimization problem if $f(x^*) \leq f(x)$ for all x in a neighborhood N around $x^*$.

or

$x^*$ is a strong local solution to the minimization problem if $f(x^*) < f(x)$ for all x in a neighborhood N around $x^*$.

## 3.2 Second-Order Optimality Conditions

1.) Necessary conditions: $\nabla f(x^*) = 0$ and $H(x^*)$ being positive semidefinite are *necessary* for $x^*$ to be a local solution to the unconstrained minimization problem.

2.) Sufficient conditions: $\nabla f(x^*) = 0$ and $H(x^*)$ being positive definite are *sufficient* for $x^*$ to be a strong local solution to the unconstrained minimization problem.

## 3.3 Convexity

The function $f(x)$ is said to be convex if and only if for every two points $x_1$ and $x_2$ in the domain of $f(x)$ the function satisfies:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

for all $\lambda \in [0, 1]$.

Let f(x) be convex. If $x^*$ is a local solution to the unconstrained minimization problem, then $x^*$ is a **global solution** to the unconstrained minimization problem. Therefore, for convex functions, local optimality implies global optimality.

Think of convex sets as shapes where any line joining 2 points in this set is never outside the set. This is called a convex set.

Visual Example:

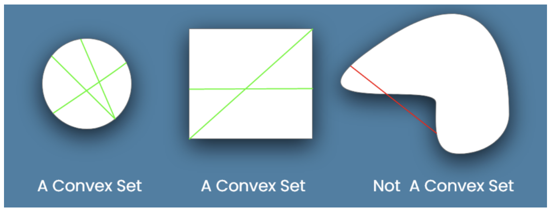

Figure 1: Understanding Convex Sets

A function f(x) is not convex if there exist two points x, y such that the line segment joining f(x) and f(y), is below the curve of the function f.
For the above non-convex $f(x)$, instead of converging to the global minimum, you can see that gradient descent would stop at the local minimum because the gradient at that point is zero (slope is 0) and minimum in the neighborhood.

How to tell if a function is convex?

Two main conditions:
1.) $f(y) \geq f(x) + (y-x)^T \nabla f(x)$ for any $x$ and $y$

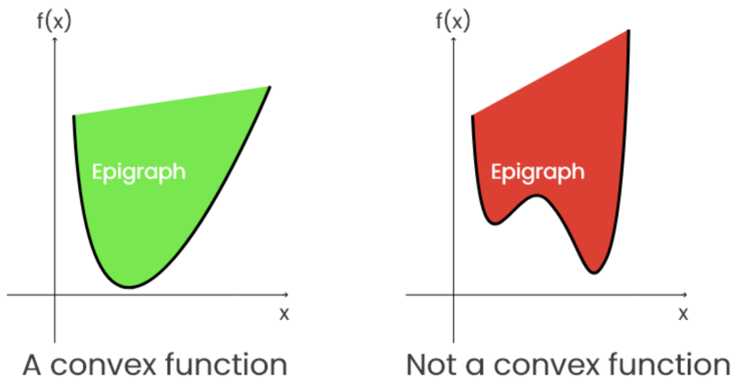

Figure 2: Understanding convex functions

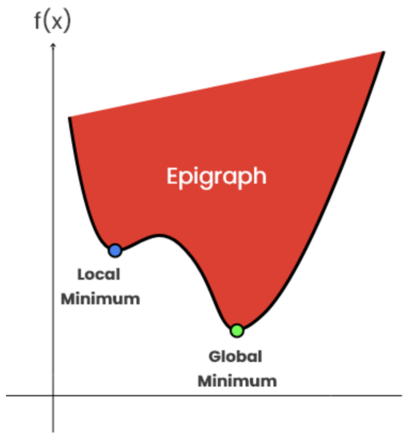

Figure 3: Gradient descent on a non-convex function

2.) $H(x) = \nabla^2 f$ is positive semi-definite for all $x$

**Issues:** It can be difficult to identify if a function is convex; it requires special problem structure.

## 3.4 Quadratic function

### 3.4.1 Converting quadratic function to standard quadratic form

If given quadratic in the form:

$$f(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2 + dx_1 + ex_2 + g$$

The first step is to convert it to standard quadratic form:

$$f(x) = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 2a & b \\ b & 2c \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} d \\ e \end{bmatrix} + g$$

In which $A = \begin{bmatrix} 2a & b \\ b & 2c \end{bmatrix}$ and B = $\begin{bmatrix} d \\ e \end{bmatrix}$ and C = g.

To condense into the standard quadratic form:

$$f(x) = \frac{1}{2}x^T Ax + x^T B + C.$$

Example 1: Convert to standard quadratic form with normalization

Given: $f(x_1, x_2) = 4x_1^2 + 6x_1x_2 + 2x_2^2 + x_1 + x_2$

Therefore: $a = 4, b = 6, c = 2, d = 1, e = 1$

Standard form: $f(x_1, x_2) = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 8 & 6 \\ 6 & 4 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

### 3.4.2 Minimization of a quadratic function

Minimization of a quadratic that takes the form: $f(x) : R^n \to R$

$$f(x) = \frac{1}{2}x^T Ax + x^T B + C$$

Transform to another basis: $x = Q\xi$ where $Q$ are a matrix of eigenvectors ($\lambda I$ aka $[q_1 ... q_n]$), $\xi \in R^n$

$Q \in R^{n\times n}$ satisfies $AQ = Q\Lambda$,
where $\Lambda = diag\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$
and $Q^TQ = I$.
Define $\bar{B} = Q^TB$.

$$f(\xi) = \frac{1}{2}\xi^T Q^T AQ\xi + \xi^T Q^T B + C = \frac{1}{2}\xi^T \Lambda\xi + \xi^T Q^T B + C$$

$$= \sum_{i=1}^{n}\left(\frac{\lambda_i \xi_i^2}{2} + \bar{B}_i\xi_i\right) + C$$

This is a separable function (sum of functions with no shared variables) Optimize for each, independent coordinate direction $\xi_i$ Once we find the optimal $\xi^*$ (if it exists) then we can find $x^* = Q\xi^*$.

Example: Consider the function $f(x) = 2x_1^2 + 2x_1x_2 + \frac{1}{2}x_2^2 + 2x_1 + x_2$

Standard quadratic form of this equation:

$$f(x) = \frac{1}{2}\begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

The $A$ matrix and $B$ vector are

$$A = \begin{bmatrix} 4 & 2 \\ 2 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

To get the standard quadratic form, need the eigenvalues of A:
To get the eigenvalues, use: $det(A - \lambda I) = 0$

$$det\begin{bmatrix} 4-\lambda & 1 \\ 2 & 1-\lambda \end{bmatrix} = 0$$

$$(4-\lambda)(1-\lambda) - 4 = 0$$
$$\lambda^2 - 5\lambda = 0$$
$$(\lambda - 5)(\lambda) = 0$$
$$\lambda_1 = 5 \quad \lambda_2 = 0$$

Therefore:

$$\Lambda = \begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}$$

Now can get the standard quadratic form:

$$f(\xi) = \frac{1}{2}\xi^T\begin{bmatrix} 5 & 0 \\ 0 & 0 \end{bmatrix}\xi + \xi^T Q^T\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \frac{5}{2}\xi_1^2 + \sqrt{5}\xi_1$$

### 3.4.3 Minimizer (critical point $x_{cr}^*$) of a quadratic function

With function:

$$f(x) = \frac{1}{2}x^T Ax + x^T B + C$$

Critical point located at $\nabla f = Ax + B = 0$

**1.** If $A$ is positive definite: Local and global minimizer at $x^* = -A^{-1}B$

**2.** If $A$ is non-singular and indefinite: Saddle point at $x = -A^{-1}B$
Note: only true if $A$ is greater than 2x2. If $A$ is 2x2, then just indefinite gives saddle point.

**3.** If $A$ is semi-definite with a single $\lambda_i = 0$, all other $\lambda_j > 0$. There is a direction such that $Aq = 0\,(q$ is the eigenvector for $\lambda_i = 0)$

- If $B^T q = 0:$ No change in the objective along $v$ direction. Infinite line of local minimizers: $x^* = x_0 + \alpha q$, where $\left(A + qq^T\right) x_0 = -\left(I - qq^T\right) B$

- If $B^T q \neq 0:$ Objective is unbounded - we can make $f(x)$ arbitrarily negative by moving along $q$. There is no minimizer.

**4.** If $A$ is semi-definite with multiple $\lambda_i = 0, AQ = 0, Q \in R^{n \times m}$

- If $B^T Q = 0:$ No change in the objective along any direction in $Q$. Infinite hyperplane of local minimizers: $x^* = x_0 + Qy$, for all $y \in R^m$ where $\left(A + QQ^T\right) x_0 = -\left(I - QQ^T\right) B$

- If $B^T Q \neq 0:$ Objective is unbounded - we can make $f(x)$ arbitrarily negative by moving along directions in $Q$ such that $B^T Qy < 0$, for some $y \in R^m$. There is no minimizer.

## 4 Types of Unconstrained Optimization Algorithms

**Direct search methods:**
- Methods for which derivatives are not used in the search
- Search direction is based on a sequence, pattern, or is random. E.g. "Search x1 direction first. Then search x2 direction,...)

**Line search methods:**
- Methods in which the search is conducted along successive lines in the design space
- Typically use derivatives in the search, but not always.

**Trust region methods**
- Methods in which the search is conducted by locally approximating the objective function as being quadratic or linear.
- The name "trust region" refers to how far from the current point this approximation is valid.

A subset of a design space for an optimization problem are usually 1.) search direction (these algorithms are “Line Search” techniques), 2.) neighborhood (these algoirthms are “Trust Region” techniques), or another method. But most techniques use a line search or trust region.

## 5 Direct Search Procedure

### 5.1 Univariate

The simplest approach to selecting a search direction is to move in one of the coordinate variable directions.

For univariate search method, we systematically select a different coordinate variable direction in each iteration.

#### 5.1.1 Steps for Univariate Search method

1.) Select an initial point, $x_0$

2.) Select a coordinate variable direction (typically $x_1$)

3.) Determine whether the $(+)$ or $(-)$ direction along that coordinate line corresponds to a descent direction. This can be accomplished with a “probe step.”

4.) Find $\alpha^*$ (scalar value indicating the distance we move along the line) along the coordinate line in the descent direction using a line search method that requires only $f$, not $df/d\alpha$.

5.) Move to the $x_n$ defined by $\alpha^*$ along the search direction.

6.) Select the next coordinate variable direction; if we reach the end of the list, start over with $x_1$.

7.) Repeat steps 3-6 until a convergence criterion has been met.

Example 1: An example progression of univariate search vectors for a 4-D problem is as follows:

$$s_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, s_2 = \begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \end{bmatrix}, s_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, s_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

Example 2:

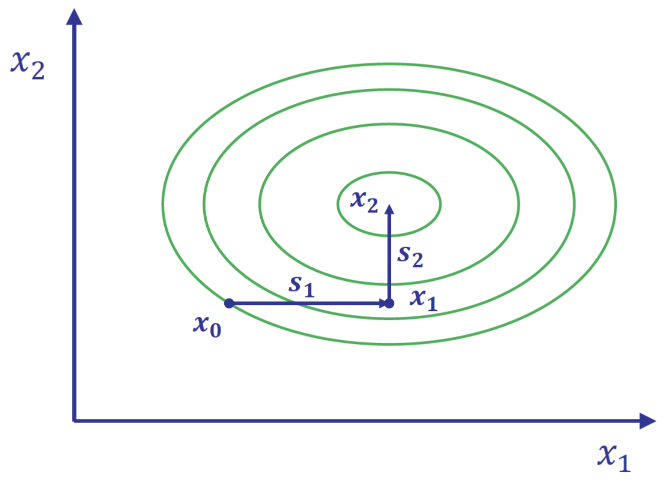

Figure 4: univariate example of non-rotated principal axes

The principal axes of the function contours were not rotated from the coordinate axes, so it would take less iterations (shown in Figure 4).

Example 3:

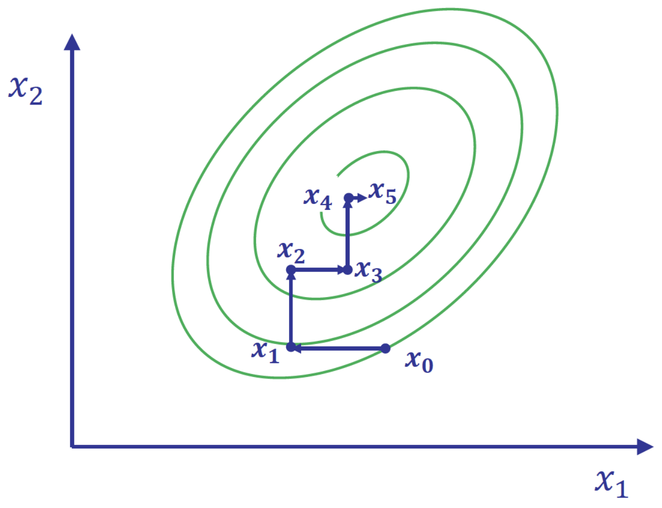

Figure 5: univariate example of rotated principal axes

The principal axes of the function contours were rotated from the coordinate axes, so it would take more iterations (shown in Figure 5).

### 5.1.2 Variations of Univariate

An alternative form of the univariate search is to select the next coordinate variable direction as the one at the current point for which $\frac{\partial f}{\partial x_i}$ is the greatest in magnitude.

The derivative information can be inferred by a "probe step" or from a gradient calculation.

If we use gradient information in the line search and the determination of the search direction, the univariate search becomes a first-order method.

## 5.2 Powell's Method

Powell's method is a modification of the univariate search based on **conjugate directions**.

The approach helps to "correct" (i.e. more efficient/less steps) the behavior of univariate search for cases in which the principal axes of the quadratic approximation of the function do not lie along a coordinate direction.

### 5.2.1 Powell's Method Procedure

1.) Select an initial point, $x_0$
2.) Begin with a set of $n$ coordinate (univariate) vectors $s_i$, $i = 1, \ldots, n$
3.) Perform one line search along each $s_i$ ($s_k = p_k =$ search directions) successively, moving to the minimum point $\alpha_i^*$ along the search direction each time.
4.) Create a conjugate direction by summing the $n$ vectors in the set involved in the iteration,

$$s_{n+\mathbf{1}} = \sum_{i=1}^{n} \alpha_i^* s_i$$

5.) Search along the conjugate direction to determine $\alpha_{n+1}^*$ and move to the minimum point.
6.) Update the set of vectors by discarding the $1^{st}$ element, shifting other elements one to the left, and introducing $s_{n+1}$ as the last element. This set of $n + 1$ line searches is called an iteration of Powell's method.
7.) Repeat steps 3 through 7 until a convergence criterion is met,resetting the set of vectors to the coordinate directions every $n + 1$ iterations to prevent the search directions from becoming increasingly parallel.

Example 1:

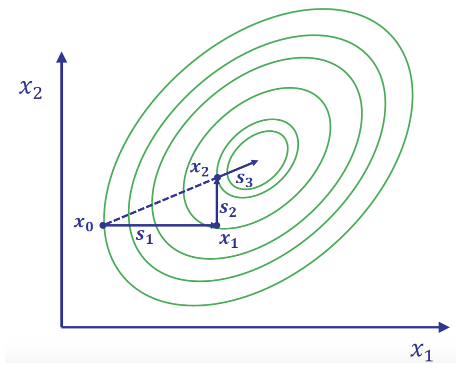

Figure 6: graphical interpretation of Powell's method

Search directions $s_1$ and $s_2$ were the 2 coordinate directions and $s_3$ was the first conjugate direction (Figure 6).

Example 2:

For a more complicated (non-quadratic) function, convergence is slower: (Figure 7).

#### 5.2.2 Convergence of Powell's method

In general, for an $n-D$ quadratic function with a minimum, Powell's method will converge to the minimum after $n$ conjugate directions have been formed, requiring $n^2$ line searches.

Powell's method is therefore said to demonstrate **quadratic convergence**.

#### 5.2.3 Limitations of Powell's method

For functions that are not positive definite quadratics, method will require more iterations or will not converge at all.

It will take additional iterations if the line search method does not find the "exact" minimum along each line.

If the conjugate directions become linearly dependent (parallel), the method can break down; hence, the need to reset the search vectors to the univariate directions every few steps.

### 5.3 Grid Search

Searches a full-factorial discretization of the entire space.

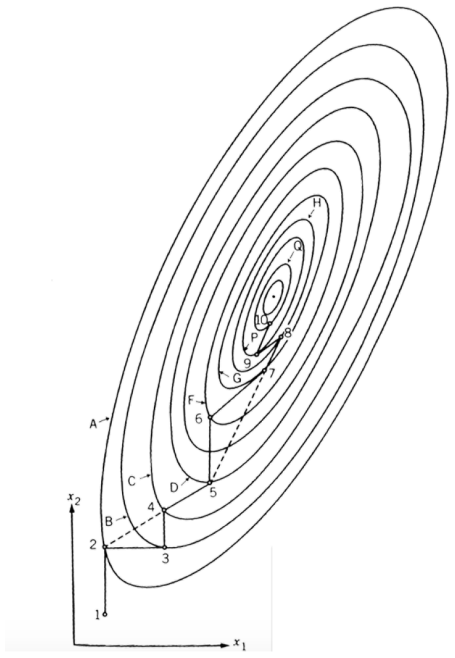

Figure 7: graphical interpretation of Powell's method for a complicated (non-quadratic) function

### 5.3.1 Grid Search Procedure

**1.** Discretize the space as a full-factorial "grid" in all of the design variables.
**2.** Evaluate $f$ at each point in the grid.
**3.**(Optional) Re-discretize the grid at a finer resolution in the region of the space with the smallest $f$ from step 2.
**4.** (Optional) Evaluate $f$ at each point in the finer grid.
**5.** (Optional) Repeat steps 3 through 4 until convergence.
**6.** Select the best point.

## 5.4 Random Search

Searches random points in a discretized design space.

### 5.4.1 Random Search Procedure

**1.** Discretize the space (even just to floating point precision).
**2.** Generate a set of points by sampling each design variable from a probability distribution (typically a uniform distribution) to generate each point.
**3.**Evaluate $f$ at the set of sampled points.

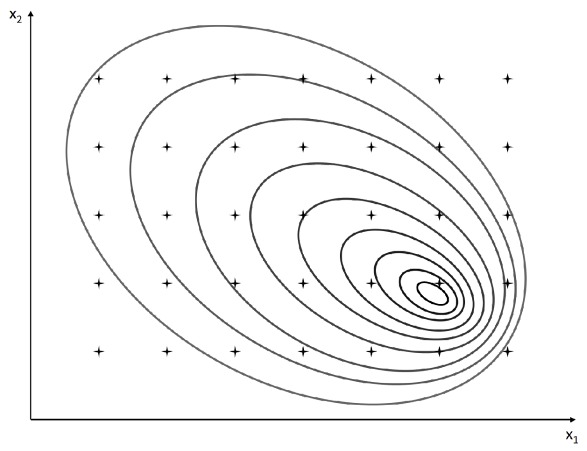

Figure 8: Grid Search

**4.** (Optional) Re-discretize the space at a finer resolution in the region with the smallest $f$ from step 3.
**5.** (Optional) Evaluate $f$ at each point in the finer discretization.
**6.** (Optional) Repeat steps 4 through 5 until convergence.

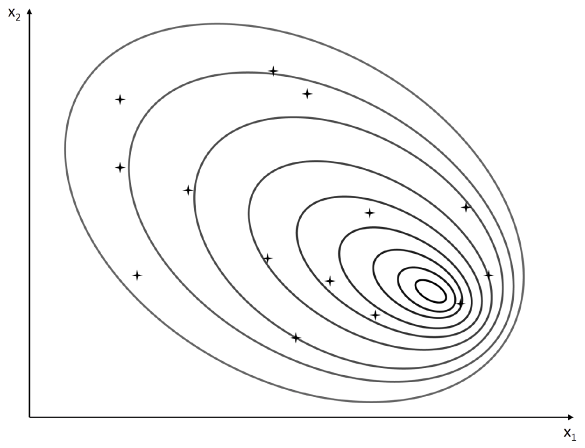

Figure 9: Random Search

### 5.4.2 Pros Cons of Random Search and Grid Search

**Pros:**
- Easy to implement/code

- Reasonable for low dimensional problems
- Works with discrete design variables
- Good chance of finding a point near the global minimum
- “Embarrassingly parallel”

**Cons:**
- Requires many function calls
- Especially inefficient for higher dimensional problems

## 5.5 Random Walk

Starts from an initial point and searches in random directions with a fixed step size, stepping to the first point with improvement.

### 5.5.1 Random Walk Procedure

**1.** Pick initial point and evaluate $f$.
**2.** Take trial step in random direction.
**3.** Evaluate $f$ at trial point.
**4.** If $f$ does not improve, test another random direction.
**5.** Move to first trial point with improved objective function value.
**6.** If several directions have been tested without improvement, halve step size and repeat from step 2.

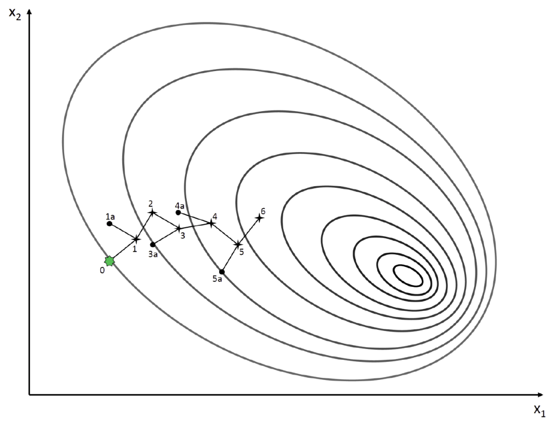

Figure 10: Random Walk

## 5.6 Compass

Starts from an initial point and searches in one univariate direction at a time with a fixed step size, stepping to the first point with improvement.

### 5.6.1 Compass Procedure

**1.** Pick initial point and evaluate $f$.
**2.** Take trial step in in first univariate direction, e.g. $x_1$.
**3.** Evaluate $f$ at trial point.
**4.** If $f$ does not improve, test next univariate direction, e.g. $x_2$. If $f$ does improve, retain search direction.
**5.** Move to first point with improved $f$ value, step again in retained search direction, and repeat from step 3.
**6.** If all directions have been tested without improvement, halve step size and repeat from step 2.

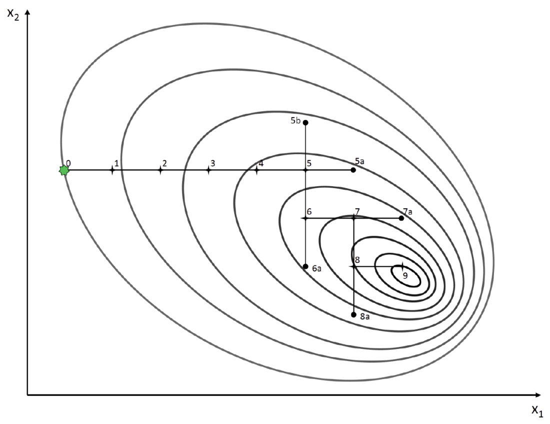

Figure 11: Compass Search

## 5.7 Coordinate Pattern

Starts from an initial point and searches in all univariate directions at the same time with a fixed step size, stepping to the point with most improvement.

### 5.7.1 Coordinate Pattern Procedure

**1.** Pick initial point and evaluate $f$.
**2.** Take trial step in all univariate directions, both (+) and (-), except for steps that would replicate points already evaluated.
**3.** Evaluate $f$ at trial point.

**4.** Move to point with most improvement in $f$ and repeat from step 2.
**5.** If all directions fail to improve, halve step size and repeat from 2.

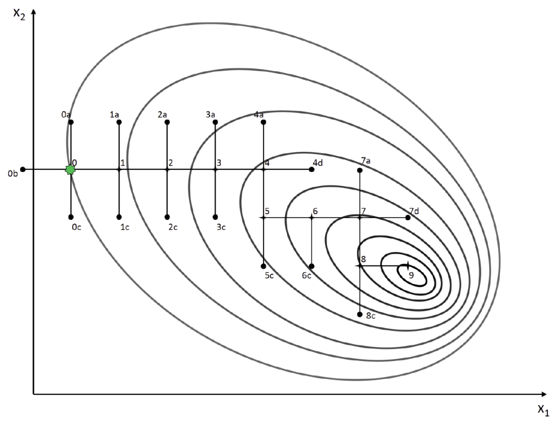

Figure 12: Coordinate Pattern

## 6 Line Search Methods

A line search involves two general steps:

**1.) Select a "search direction" ($s_k$ or $p_k$)**
- Discussed in subsection "Line Search Directions"

**2.) Select the distance ($\alpha^*$) to move along this direction**
- Discussed in subsection "Estimating distance $\alpha^*$".

**What is a line search?**
Line Search is a search method in which the search is conducted along successive lines in the design space.

Line search methods are based on an iterative procedure of the form:

$$x_k = x_{k-1} + \alpha^* s_k \tag{5}$$

where:
$x_k \equiv$ new design variable vector
$x_{k-\mathbf{1}} \equiv$ previous design variable vector
$s_k \equiv$ a vector in a **descent direction** that defines the line being searched. Finding $s_k$ is the **direction-finding problem**.

$\alpha^* \equiv$ a scalar indicating the distance we should move to (approximately) minimize the function along the line. Finding $\alpha^*$ is the **task of the line search itself**.

**General Steps of Line Search Method**

1.) Choose an initial point in the design space, $x_0$
2.) Select a descent direction, $s_k$, i.e. a direction along which the function value decreases.
3.) Determine the $\alpha^*$ that will minimize $f(x)$ (approximately) along the line defined by $s_k$
4.) Decide when the process has converged to an acceptable solution in order to stop iterating

**Types of Line Search Methods:**

Line search methods are categorized by the quality of the local approximation of the function that they use in the direction-finding problem to determine $s_k$:

**Zeroth-order method:**
- Use only values of the function $f(x)$ itself.
- Some consider zeroth-order methods to be direct search methods

**First-order methods:**
- Use values of $f(x)$ and $\nabla f(x)$
- May use approximations of $H(x)$ but are still of first-order accuracy

**Second-order methods:**
- Use values of $f(x), \nabla f(x)$, and $H(x)$

## 6.1 Estimating distance $\alpha^*$

Estimating $\alpha^*$ requires a search along the line defined by the direction $s_k$ to find an estimate of the minimum along that line.

**Advantage:** it is a 1-D search, which is much easier than an n-D search

**Two approaches discussed for finding $\alpha^*$:**
1.) Polynomial Approximation
2.) Golden Section

### 6.1.1 Polynomial Approximation

**What is it?**
Polynomial Approximation is used to approximate the function along the line defined by $s_k$ with a polynomial in $\alpha$:

$$f(x) \approx \tilde{f}(x) = b_0 + b_1\alpha + b_2\alpha^2 + \cdots + b_n\alpha^n$$

We are typically interested in "reasonable" but inexact approximations of the minimum along the search line. Therefore, the most common approximations are quadratics and cubics, which provide a nice balance between accuracy and efficiency.

**2-Point Quadratic Approximation**
Quadratic approximation in which we know information at two points:

$$f(x) = f\left(x_{k-\mathbf{1}} + \alpha s_k\right) \approx \tilde{f}(x) = b_0 + b_1\alpha + b_2\alpha^2$$

$$\tfrac{df(x)}{d\alpha} = \tfrac{df(x)}{dx}\tfrac{dx}{d\alpha} = \left[\nabla f\left(x_{k-1}\right)\right]^T s_k \approx \tfrac{d\tilde{f}(x)}{d\alpha} = b_1 + 2b_2\alpha$$

At point 1, corresponding to $\alpha_1$, we know both $\left.\tfrac{df}{d\alpha}\right|_1$ and $f_1$.
At point 2 , corresponding to $\alpha_2$, we know $f_2$.

Substituting the known values gives,

$$\begin{aligned} \left.\tfrac{df}{d\alpha}\right|_1 &= b_1 + 2b_2\alpha_1 \\ f_1 &= b_0 + b_1\alpha_1 + b_2\alpha_1^2 \\ f_2 &= b_0 + b_1\alpha_2 + b_2\alpha_2^2 \end{aligned}$$

With $\alpha_1$, $\left.\tfrac{df}{d\alpha}\right|_1$, $f_1, \alpha_2$, and $f_2$ known, we need to solve for the coefficients $b_0, b_1$, and $b_2$ to find the polynomial approximation.

With $\alpha_1$, $\left.\tfrac{df}{d\alpha}\right|_1$, $f_1, \alpha_2$, and $f_2$ known, we need to solve for the coefficients $b_0, b_1$, and $b_2$ to find the polynomial approximation.

Subtracting the two function values and dividing by $(\alpha_2 - \alpha_1)$,

$$\begin{aligned} f_2 - f_1 &= b_1\left(\alpha_2 - \alpha_1\right) + b_2\left(\alpha_2^2 - \alpha_1^2\right) \\ \tfrac{f_2 - f_1}{\alpha_2 - \alpha_1} &= b_1 + b_2\left(\alpha_2 + \alpha_1\right) \end{aligned}$$

But,

$$b_1 = \left.\frac{df}{d\alpha}\right|_1 - 2b_2\alpha_1$$

So,

$$\frac{f_2 - f_1}{\alpha_2 - \alpha_1} = \left.\frac{df}{d\alpha}\right|_1 + b_2(\alpha_2 - \alpha_1)$$

Solving for $b_2$ gives

$$b_2 = \frac{(f_2 - f_1)/(\alpha_2 - \alpha_1) - \left.\frac{df}{d\alpha}\right|_1}{(\alpha_2 - \alpha_1)}$$

With $b_2$ known, we can then find $b_1$ and $b_0$ directly as,

$$b_1 = \left.\tfrac{df}{d\alpha}\right|_1 - 2b_2\alpha_1$$
$$b_0 = f_1 - b_1\alpha_1 - b_2\alpha_1^2$$

The critical point of $f$ along the search line is then found as,

$$\tfrac{d\tilde{f}}{d\alpha} = b_1 + 2b_2\alpha = 0$$
$$\Rightarrow \quad \alpha_{cr} = \tfrac{-b_1}{2b_2}$$

If $b_2 > 0$, the critical point is a minimum.
If $b_2 < 0$, the critical point is a maximum.
If the critical point is a minimum, then $\alpha^* = \alpha_{cr}$.
since we search in a descent direction, we should hopefully find a minimum.

**3-Point Quadratic Approximation**
Quadratic approximation in which we know information at three points:

Consider again our quadratic approximation:

$$\tilde{f}(x) = b_0 + b_1\alpha + b_2\alpha^2$$
$$\tfrac{d\tilde{f}(x)}{d\alpha} = b_1 + 2b_2\alpha$$

Presume that instead of knowing the function value at 2 points and the derivative at 1 point, we know the function value at 3 points, i.e.

$$(\alpha_1, f_1), (\alpha_2, f_2), (\alpha_3, f_3)$$

We can then solve for the coefficients as,

$$b_2 = \frac{(f_3-f_1)/(\alpha_3-\alpha_1)-(f_2-f_1)/(\alpha_2-\alpha_1)}{(\alpha_3-\alpha_2)}$$
$$b_1 = \frac{(f_2-f_1)}{(\alpha_2-\alpha_1)} - b_2(\alpha_1+\alpha_2)$$
$$b_0 = f_1 - b_1\alpha_1 - b_2\alpha_2^2$$

Other polynomial approximations besides 2-point and 3-point approximations are presented in the Vanderplaats textbook.

**Selecting the Points for Defining the Polynomial**

Now, we know how to define a polynomial once we are given several points along a search direction (i.e. 2-points or 3-points to get polynomial approximations).

**But how do we select the points, i.e. values of $\alpha_1, \alpha_2$, etc.?**

We start the line search at the point $x_k$ at which $\alpha = 0$. It is convenient to associate $\alpha_1$ with this starting point. This is also the point at which we may know the gradient.

For 3-point methods, presuming that we number the $\alpha_i$ in order of increasing $\alpha$, we need to select $\alpha_2$ and $\alpha_3$ such that $f_2 < \min(f_1, f_3)$. This is called **bracketing the minimum**.

Presuming $\alpha_1$ is fixed and corresponds to the starting point $x_k$, we can use a bracketing algorithm to select points $\alpha_2$ and $\alpha_3$.

Let's begin by making the following presumptions:

a. The function is unimodal along the search direction.

b. $*\left.\frac{df}{d\alpha}\right|_1 < 0$, i.e. the search is in a descent direction.

These conditions imply that a minimum defined by $\alpha^*$ and $f^*$ exists along the search direction.

Presume that we select $\alpha_2$ and $\alpha_3$ and evaluate the corresponding values of $f_1, f_2$, and $f_3$. Three possibilities exist:

1. Minimum already bracketed: $f_2 < \min(f_1, f_3)$

2. Interval too small: $f_3 < \min(f_1, f_2)$

3. Interval too large: $f_1 < \min(f_2, f_3)$

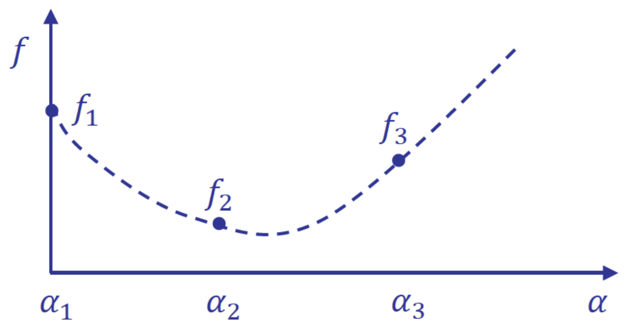

Figure 13: Minimum already bracketed: $f_2 < \min(f_1, f_3)$

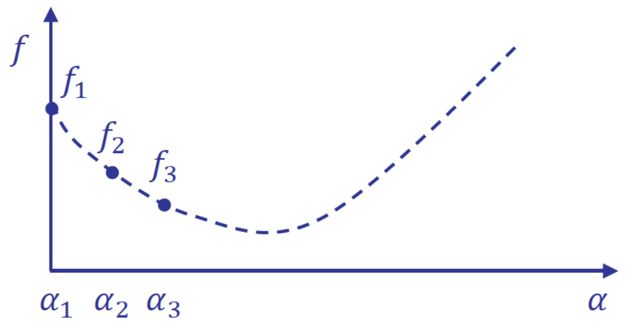

Figure 14: Interval too small: $f_3 < \min(f_1, f_2)$

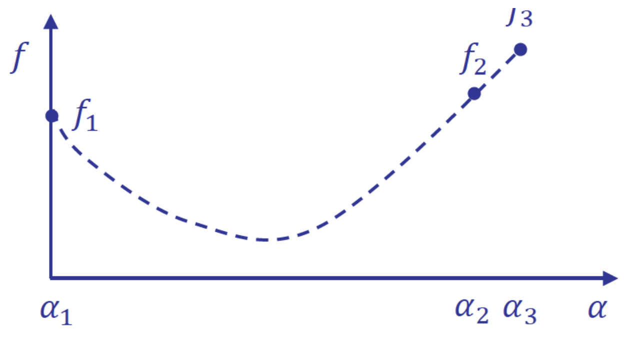

Figure 15: Interval too large: $f_1 < \min(f_2, f_3)$

Let's now presume that we work with a fixed interval width,

$$\Delta\alpha = \alpha_3 - \alpha_2 = \alpha_2 - \alpha_1$$

Here's an approach to adjusting $\Delta\alpha$ based on the initial points:

* If the minimum is already bracketed, $\Delta\alpha$ is fine as it is. Proceed.

* If the interval is too small, multiply $\Delta\alpha$ by 2 and try again.

* If the interval is too large, divide $\Delta\alpha$ by 2 and try again

When bracketing the minimum, you may need to shrink/grow the interval several times. However, if you start by shrinking it, you never need to grow it, and vice versa.

Note that approaches other than multiplying/dividing $\Delta\alpha$ by 2 are possible.

#### 6.1.2 Golden Section

The Golden Section Method is an approach that successively refines the bracketing of the minimum in order to converge to an estimate of the minimum.
**See Sarah's amazing notes for further details.**

### 6.2 Line Search Direction

#### 6.2.1 Wolfe Condition

**See Sarah's amazing notes for further details.**

#### 6.2.2 Steepest Descent

Steepest descent: $p_k = -\nabla f_k$, (very slow)

#### 6.2.3 Newton

Newton method: $p_k = -H(x_k)^{-1}\nabla f_k$, (slow if $H_k$ is computationally expensive, may not be a descent direction)

#### 6.2.4 Conjugate

Conjugate gradient: $p_k = -\nabla f_k + \beta p_{k-1}, \beta = \|\nabla f_k\|_2^2 / \|\nabla f_{k-1}\|_2^2$ (much better than steepest descent)

#### 6.2.5 Quasi-Newton (BFGS)

Quasi-Newton: $p_k = -B_k^{-1}\nabla f_k, B_k$ approximates the Hessian, formed by accumulating gradient information

# 7 Trust Region Methods

## 7.1 Line Search Method vs Trust Region Method

**Line search methods:**

- pick descent direction $p_k$
- pick stepsize $\alpha_k$ to "reduce" $f\left(x_k + \alpha p_k\right)$
- $x_{k+1} = x_k + \alpha_k p_k$

**Trust-region methods:**

- pick step $s_k$ to reduce "model" of $f\left(x_k + s\right)$
- accept $x_{k+1} = x_k + s_k$ if the decrease promised by the model is inherited
- otherwise set $x_{k+1} = x_k$ and improve the model.

The Trust Region Methods allow indefinite or semi-definite Hessian approximations; however, Line Search Methods do not.

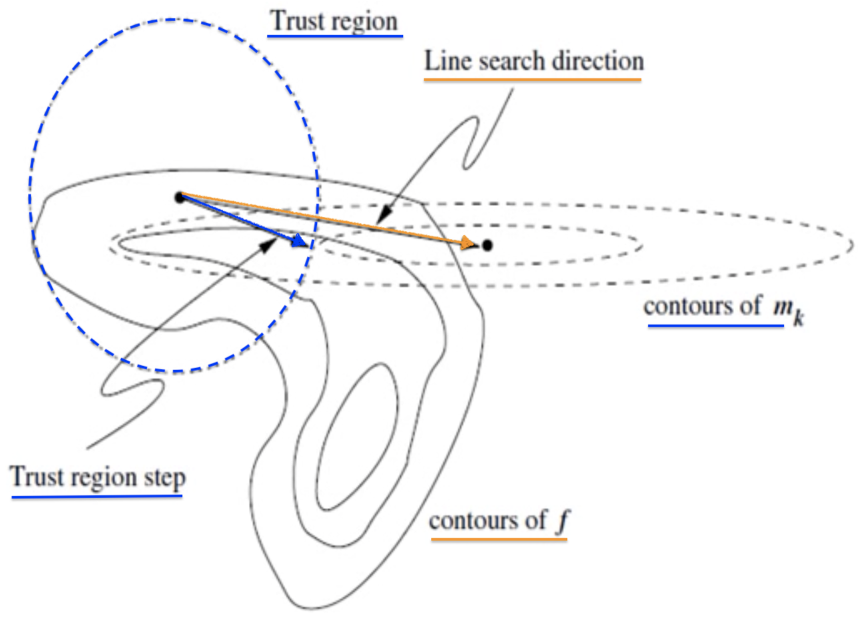

Figure 16: Line Search vs Trust Region

## 7.2 What are the Trust Region Methods?

Trust Region Methods are an iterative method that forms a model function $m_k(x_k + p)$ for some step inside or on the trust region radius ($\|p\| \leq \Delta_k$) in which $p$ is the search direction (the step taken by the algorithm), $\Delta_k$ is the trust region radius (the radius of the n-dimensional sphere of trust), and $x_k$ is the current search point.

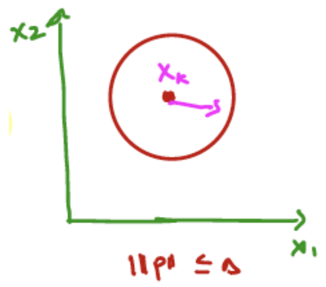

Figure 17: norm(p) <= trust region radius

The region radius is around the current search point $x_k$. This region radius is where the model for local minimization is "trusted" to be correct and steps are chosen to stay within this region. The size of the region is modified during the search, based on how well the model agrees with actual function evaluations.

The model $m_k(x_k + p)$ can be built in many different ways. We will build local models of the form:

$$m_k(x_k + p) = f(x_k) + g(x_k)^T p + \frac{1}{2} p^T B_k p \tag{6}$$

where:
$g(x_k) = \nabla f(x_k)$,
$B_k$ = Hessian or an approximation of the Hessian,
$p$ = step,
$x_k$ = current search point.

Note that this model is a quadratic obtained by a Taylor Series expansion of $f(x_k)$ around $x_k$ and is first-order consistent. The function and gradient of the model match the actual function and gradient at the origin $p = 0$.

For the course, we use quasi-Newton methods to form $B_k$.

## 7.3 Trust Region Basic Algorithm

**1.** Set $k = 1$, the initial trust region radius $\Delta_{k=1}$ and choose tolerance $\eta \in [0, 1/4)$ and $\Delta_{\max}$

**Note:** you can select the trust region radius - it will update each step. It is usually taken to be an ellipse such that $p^T D^2 p \leq \Delta^2$ where $D$ is a diagonal scaling (often taken from the diagonal of the approximate Hessian). The trust region radius $\Delta_k$ is modified dynamically each step - if the quadratic model is found to be a poor representation of the space the radius is resized in step 6.

**2.** Update the model function $m_k(x_k + p)$:

$$m_k(x_k + p) = f(x_k) + g(x_k)^T p + \frac{1}{2} p^T B_k p$$

**3.** Compute an approximate solution to: $p_k^* = \arg\min_p m_k(x_k + p)$,
such that $\|p\|_2 \leq \Delta_k$

**Note:** This is the **minimization subproblem** that can be solved using the Cauchy point algorithm (inexact solution), exact trust region algorithm, Steihaug-Toint algorithm, etc. This step will be shown in further detail in the next Section. Also, $\|.\|_2$ is "two-norm" and means the trust region is a circle. $\|p\|_2 \leq \Delta_k$ is the same as $p^T p \leq \Delta_k^2$.

**4.** Compute $f(x_k + p_k)$, and compute the ratio:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(x_k) - m_k(x_k + p_k)}$$

**Note:** The $\rho_k$ ratio is "actual improvement" over "inspected improvement".

**5.** Set the next point:
- If $\rho_k \geq \eta$, define $x_{k+1} = x_k + p_k$ (calculate new starting point because doesn't meet tolerance), otherwise define $x_{k+1} = x_k$ (keep the same starting point)

**6.** Set the new trust region radius:

- If $\rho_k < 0.25$, then set $\Delta_{k+1} = 0.25\Delta_k$ (shrink the trust region $\Delta_{k+1}$)

- Else If $\rho_k > 0.75$ and $\|p_k\|_2 == \Delta_k$, then set $\Delta_{k+1} = \min(2\Delta_k, \Delta_{\max})$ (increase the trust region because the quadratic model is found to be reasonable, so the radius is increased to allow larger, more efficient steps to be taken).

- Else, set $\Delta_{k+1} = \Delta_k$ (keep the same trust region).

**7.** Set $k = k + 1$, Go to step 2

## Trust Region Algorithm Outline:

```
Set the initial trust region radius $\Delta_1$ and choose $\eta \in [0, 1/4)$ and $\Delta_{max}$
Evaluate $f(x_1)$ and $g_1 = \nabla f(x_1)$
Set $B_1 = I$
while $||g_k||_2 = ||\nabla f(x_k)||_2 > \epsilon$ do
    if Use Cauchy point step then
        $p_k$ = cauchy_step($g_k$, $B_k$, $\Delta_k$)                    ▷ Solve the trust region problem
    else
        $p_k$ = trust_region_step($g_k$, $B_k$, $\Delta_k$)
    end if
    Compute $f(x_k + p_k)$ and evaluate the ratio:
        $$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{m_k(x_k) - m_k(x_k + p_k)}$$
    Compute $\nabla f(x_k + p_k)$
    Set $y_k = \nabla f(x_k + p_k) - \nabla f(x_k)$ and $s_k = p_k$ and update $B_k$ using the SR-1 formula:
        $$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{s_k^T(y_k - B_k s_k)}$$
    if $\rho_k \geq \eta$ then                                              ▷ Set the new point
        Set $x_{k+1} = x_k + p_k$
    else
        Set $x_{k+1} = x_k$
    end if
    if $\rho_k < 0.25$ then                                        ▷ Set the new trust region radius
        Set $\Delta_{k+1} = 0.25\Delta_k$
    else if $\rho_k > 0.75$ and $||p_k||_2 == \Delta_k$ then
        Set $\Delta_{k+1} = min(2\Delta_k, \Delta_{max})$
    else
        Set $\Delta_{k+1} = \Delta_k$
    end if
    Set $k = k + 1$, Goto step 2
end while
```

Figure 18: trust region algorithm

### 7.3.1 Trust Region Class of Algorithms (Needed for step 3 of Trust Region Basic Algorithm)

Remember, step 3 of the Trust Region Basic Algorithm is:

Compute an approximate solution to $p_k^* = \arg\min_p m_k\left(x_k + p\right)$,
such that $\|p\|_2 \leq \Delta_k$

Therefore, we need to find the solution of the constrained minimization problem:

$$\min_p g^T p + \frac{1}{2} p^T B p, \quad such that \|p\|_2 \leq \Delta$$

There are several algorithms to do this. The two main ones are the Cauchy point algorithm, which gives an estimate $p^*$, and the Exact trust region algorithm.

**Cauchy point algorithm (inexact)**
It turns out, we don't need an exact solution - we can estimate $p^*$. This will provide enough progress for convergence of the trust region method, as long as we make as much progress as the Cauchy point. In order to converge, our algorithm must make an improvement at every step.

Cauchy point is the minimum value of $m(x+p)$ subject to $|p\|_2 \leq \Delta$ along the steepest descent direction: $p = -\alpha g$ in which $\alpha = \tau \frac{\Delta}{\|g\|_2}$. $\tau$ is a factor that governs whether the curvature is along $g$ or not. The cauchy step has two options: 1.) step to the boundary (trust region radius) or 2.) step into a minimizer that's within the trust region radius.

Cauchy Point:

$$p = -\tau \tfrac{\Delta}{\|g\|_2} g \quad \tau = \begin{cases} 1 & g^T B g \leq 0 \\ \min\left(1, \|g\|_2^3 / \Delta g^T B g\right) & otherwise \end{cases}$$

in which $g^T B g \leq 0$ if the constraint bound is active.

***Cauchy Point Case 1:***
$B$ is positive definite & the unconstrained minimizer lies <u>outside</u> the trust region.

$$p = -\tau \frac{\Delta}{\|g\|_2} g, \tau = 1$$

$\tau = 1$ because $g^T B g \leq 0$

$$\longrightarrow p = -\frac{\Delta}{\|g\|_2} g$$

***Cauchy Point Case 2:***
$B$ is positive definite & the unconstrained minimizer lies <u>inside</u> the trust region.

$\tau = |g\|_2^3/\Delta g^T Bg$ because $g^T Bg \geq 0$

$$\longrightarrow p = -\tau \frac{\Delta}{\|g\|_2} g, \tau = \min\left(1, \|g\|_2^3/\Delta g^T Bg\right)$$

***Cauchy Point Case 3:***
$B$ is indefinite.

$g^T Bg{>}0$ therefore, inside because constraint

**<u>Exact trust region algorithm</u>**

The exact solution $p^*$ is characterized as follows:

1.) $(B + \lambda I)p^* = -g$
2.) $(\Delta - \|p^*\|_2)\,\lambda = 0$
3.) $\lambda \geq 0$
4.) $(B + \lambda I)$ positive, semi-definite

We have to find $p^*$ and $\lambda$ that will satisfy these conditions.

Complementarity constraint:

Either $\|p^*\|_2 \leq \Delta$ (aka inside trust region) and $\lambda = 0$,

or

$\lambda \neq 0$ and and $\|p^*\|_2 = \Delta$ (on trust region boundary).

***Exact trust region algorithm Case 1:***
$B$ is positive definite & $\|p\| \leq \Delta$.

Then: $\lambda = 0$
and $\Rightarrow p^* = p$

***Exact trust region algorithm Case 2:***
$B$ is positive definite & $\|p\| {>} \Delta$.

Then: solve for $\lambda$ with $\lambda \geq 0$

For $\lambda_1$ :

$$(B-\lambda_1 I)q_1 = 0$$

to get $q_1$

For $\lambda_2$ :

$$(B-\lambda_2 I)\, q_2 = 0$$

to get $q_2$

Use: $\lambda_1, \lambda_2, q_1, q_2, g, \Delta$

$\Rightarrow \|p(\lambda)\|_2^2 = \sum_{i=1}^{2} \frac{\left(q_i^\top g\right)^2}{(\lambda_i+\lambda)^2} \leq \Delta^2$ to get $\lambda$ multiplier $(\lambda \geq 0)$

Then: $(B+\lambda I)p = -g$
$\Rightarrow p$.

***Exact trust region algorithm Case 3:***
$B$ is indefinite Hessian.

Then: solve for $\lambda$ with 1.) $\lambda \geq 0$ and 2.) $\lambda \geq (-\lambda_1)$
For $\lambda_1$ :

$$(B-\lambda_1 I)q_1 = 0$$

to get $q_1$

For $\lambda_2$ :

$$(B-\lambda_2 I)\, q_2 = 0$$

to get $q_2$

Use: $\lambda_1, \lambda_2, q_1, q_2, g, \Delta$

$\Rightarrow \|p(\lambda)\|_2^2 = \sum_{i=1}^{2} \frac{\left(q_i^\top g\right)^2}{(\lambda_i+\lambda)^2} \leq \Delta^2$
to get $\lambda$ multiplier with constraints $\lambda \geq 0$ and $\lambda \geq (-\lambda_1)$

Then: plug $\lambda$ into: $(B+\lambda I)p = -g$
$\Rightarrow p$.

**Example (Cauchy point step $p_c$ vs Exact step $p^*$):**

**Cauchy point step $p_c$ :**

$m(p) = p^\top g + \frac{1}{2} p^\top B_\rho \Delta = 1$

g $= \begin{bmatrix} -1 \\ -1 \end{bmatrix} B = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Therefore: $\lambda_1 = -1, \lambda_2 = 1$

$g^\top B g = \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = 0$

Cauchy step will go all the way to the boundary (trust region radius).

$p^c = -\alpha g = \begin{bmatrix} \alpha \\ \alpha \end{bmatrix}$

$\|p^c\| = 1$

**Exact step $p^*$:**

$m(p) = p^\top g + \frac{1}{2} p^\top B_\rho \Delta = 1$

g $= \begin{bmatrix} -1 \\ -1 \end{bmatrix} B = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Therefore: $\lambda_1 = -1, \lambda_2 = 1$

$(B + \lambda I)p^* = -g$

$p_1^* = \frac{1}{\lambda - 1} \quad p_2^* = \frac{1}{1+\lambda}$

$\|p^*\|_2 = \Delta = 1$

$\|p^*\|_2^2 = \Delta^2$

$(p_1^*)^2 + (p_2^*)^2 = 1$

$\left(\frac{1}{\lambda-1}\right)^2 + \left(\frac{1}{\lambda+2}\right)^2 = 1$

$(\lambda - 1)^2 + (\lambda + 1)^2 = (\lambda + 1)^2 (\lambda - 1)^2$

$\lambda^2 - 2\lambda + 1 + \lambda^2 + 2\lambda + 1 = \left(\lambda^2 - 1\right)^2$

$= \lambda^4 + 4\lambda^2 - 1 = 0$

$\lambda^2 = 2 \pm \sqrt{5}$

$\lambda = \pm\sqrt{2 \pm \sqrt{5}}\ \lambda$

$=\pm\sqrt{2 - \sqrt{5}}$ is imaginery.

$\lambda = \pm\sqrt{2 + \sqrt{5}}, \quad \lambda < 0$ violates condition

So: $\lambda = \pm\sqrt{2+\sqrt{5}}$

And: $p^k = \begin{bmatrix} \frac{1}{\sqrt{2+\sqrt{5}}-1} \\ \frac{1}{\sqrt{2+\sqrt{5}}+1} \end{bmatrix}$