

Unconstrained Multivariate Optimization

Multivariate optimization means optimization of a scalar function of a several variables:

$$y = P(\mathbf{x})$$

and has the general form:

$$\min_{\mathbf{x}} \quad P(\mathbf{x})$$

where $P(\mathbf{x})$ is a nonlinear scalar-valued function of the vector variable \mathbf{x} .

Background

Before we discuss optimization methods, we need to talk about how to characterize nonlinear, multivariable functions such as $P(\mathbf{x})$. Consider the 2nd order Taylor series expansion about the point \mathbf{x}_0 :

$$P(\mathbf{x}) \approx P(\mathbf{x}_0) + \nabla_{\mathbf{x}} P|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_0} (\mathbf{x} - \mathbf{x}_0)$$

If we let:

$$\mathbf{a} = P(\mathbf{x}_0) - \nabla_{\mathbf{x}} P|_{\mathbf{x}_0} \mathbf{x}_0 + \frac{1}{2} (\mathbf{x}_0)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_0} \mathbf{x}_0$$

$$\mathbf{b}^T = \nabla_{\mathbf{x}} P|_{\mathbf{x}_0} - (\mathbf{x}_0)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_0}$$

$$\mathbf{H} = \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_0}$$

Unconstrained Multivariate Optimization

Then we can re-write the Taylor series expansion as a quadratic approximation for $P(\mathbf{x})$:

$$P(\mathbf{x}) \approx a + \mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x}$$

and the derivatives are:

$$\begin{aligned}\nabla_{\mathbf{x}} P(\mathbf{x}) &\approx \mathbf{b}^T + \mathbf{x}^T \mathbf{H} && \text{gradient} \\ \nabla_{\mathbf{x}}^2 P(\mathbf{x}) &\approx \mathbf{H} && \text{Hessian}\end{aligned}$$

We can describe some of the local geometric properties of $P(\mathbf{x})$ using its gradient and Hessian. In fact there are only a few possibilities for the local geometry, which can easily be differentiated by the eigenvalues of the Hessian (\mathbf{H}).

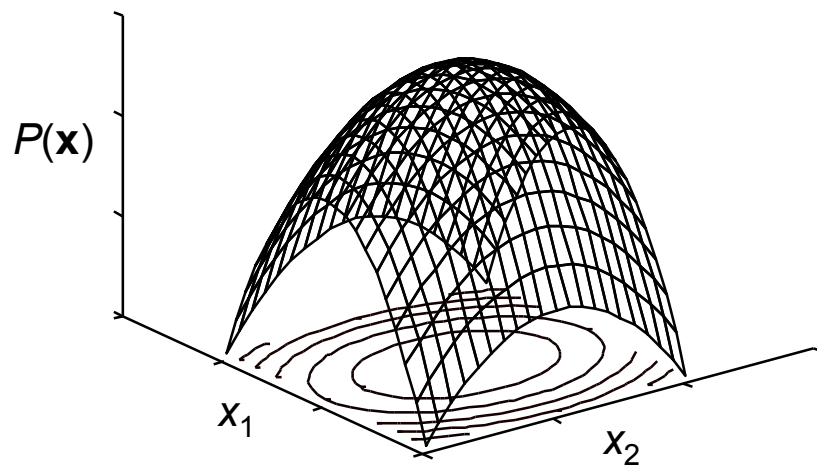
Recall that the eigenvalues of a square matrix (\mathbf{H}) are computed by finding all of the roots (λ_i) of its characteristic equation:

$$|\lambda \mathbf{I} - \mathbf{H}| = 0$$

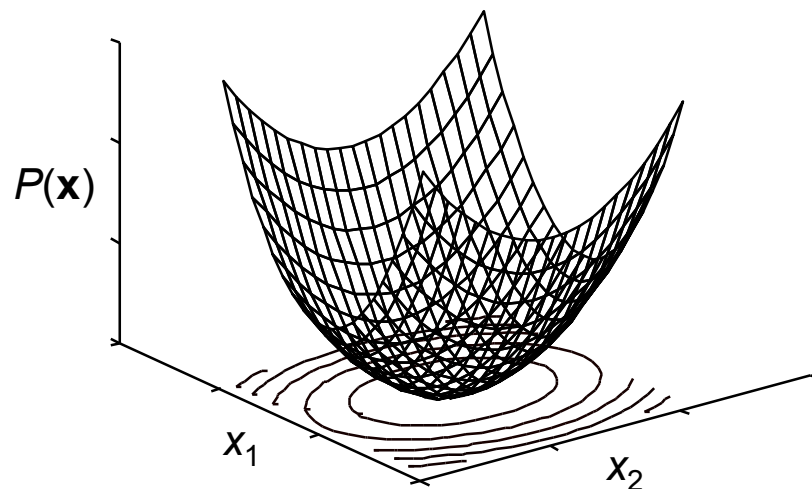
Unconstrained Multivariate Optimization

The possible geometries are:

- 1) if $\lambda_i < 0$ ($\forall i=1, \dots, n$), the Hessian is said to be negative definite. This object has a unique maximum and is what we commonly refer to as a hill (in three dimensions).

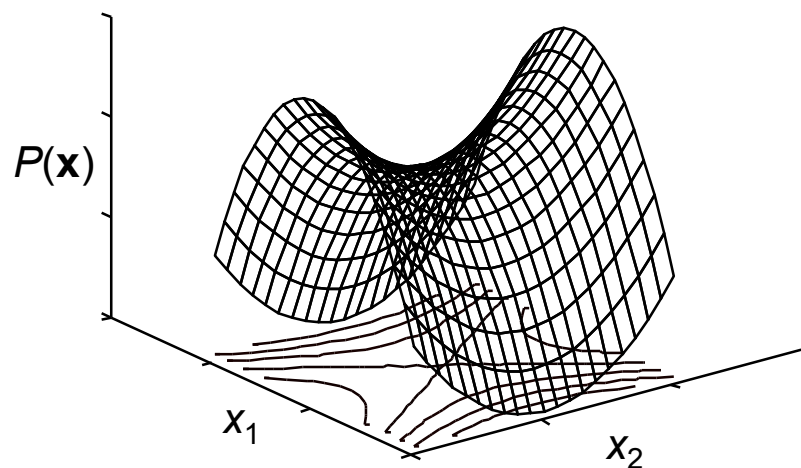


- 2) if $\lambda_i > 0$ ($\forall i=1, \dots, n$), the Hessian is said to be positive definite. This object has a unique minimum and is what we commonly refer to as a valley (in three dimensions).

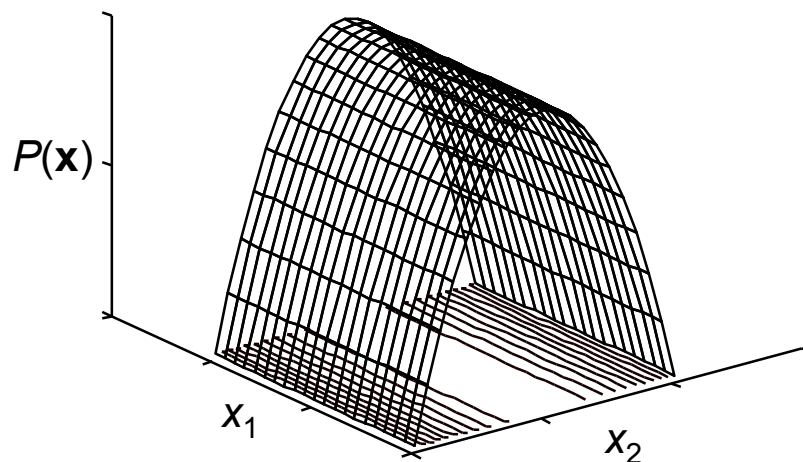


Unconstrained Multivariate Optimization

- 3) if $\lambda_i < 0$ ($\forall i=1, \dots, m$) and $\lambda_i > 0$ ($\forall i=m+1, \dots, n$), the Hessian is said to be indefinite. This object has neither a unique maximum or minimum, and is what we commonly refer to as a saddle (in three dimensions).

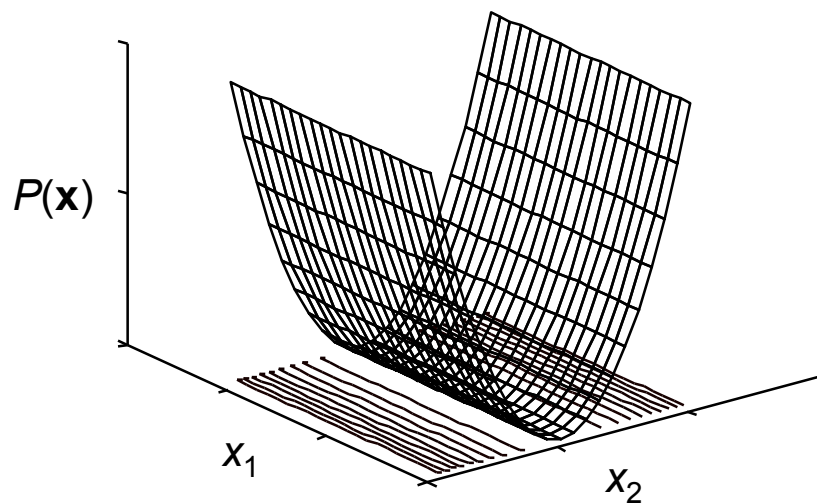


- 4) if $\lambda_i < 0$ ($\forall i=1, \dots, m$) and $\lambda_i = 0$ ($\forall i=m+1, \dots, n$), the Hessian is said to be negative semi-definite. This object does not have a unique maximum and is what we commonly refer to as a ridge (in three dimensions).



Unconstrained Multivariate Optimization

5) if $\lambda_i > 0$ ($\forall i=1, \dots, m$) and $\lambda_i = 0$ ($\forall i=m+1, \dots, n$), the Hessian is said to be positive semi-definite. This object does not have a unique minimum and is what we commonly refer to as a trough (in three dimensions).



A well posed problem has a unique optimum, so will limit our discussions to either problems with positive definite Hessians (for minimization) or negative definite Hessians (for maximization).

Further, we would prefer to choose units for our decision variables (\mathbf{x}) so that the eigenvalues of the Hessian all have approximately the same magnitude. This will scale the problem so that the profit contours are concentric circles and will condition our optimization calculations.

Unconstrained Multivariate Optimization

Necessary and Sufficient Conditions

For a twice continuously differentiable scalar function $P(\mathbf{x})$, a point \mathbf{x}^* is an optimum if:

$$\nabla_{\mathbf{x}} P|_{\mathbf{x}^*} = \mathbf{0}$$

and:

$\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}^*}$ is positive definite (a minimum)

$\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}^*}$ is negative definite (a maximum)

We can use these conditions directly, but it usually involves solving a set of simultaneous nonlinear equations (which is usually just as tough as the original optimization problem). Consider:

$$P(\mathbf{x}) = \mathbf{x}^T \mathbf{A} \mathbf{x} e^{\mathbf{x}^T \mathbf{A} \mathbf{x}}$$

Then:

$$\begin{aligned} \nabla P &= 2(\mathbf{x}^T \mathbf{A}) e^{\mathbf{x}^T \mathbf{A} \mathbf{x}} + \mathbf{x}^T \mathbf{A} \mathbf{x} (\mathbf{x}^T \mathbf{A}) e^{\mathbf{x}^T \mathbf{A} \mathbf{x}} \\ &= 2(\mathbf{x}^T \mathbf{A}) e^{\mathbf{x}^T \mathbf{A} \mathbf{x}} (1 + \mathbf{x}^T \mathbf{A} \mathbf{x}) \end{aligned}$$

and stationarity of the gradient requires that:

$$\nabla P = 2(\mathbf{x}^T \mathbf{A}) e^{\mathbf{x}^T \mathbf{A} \mathbf{x}} (1 + \mathbf{x}^T \mathbf{A} \mathbf{x}) = \mathbf{0}$$

This is a set of very nonlinear equations in the variables \mathbf{x} .

Unconstrained Multivariate Optimization

Example

Consider the scalar function:

$$P(\mathbf{x}) = 3 + x_1 + 2x_2 + 4x_1x_2 + x_1^2 + x_2^2$$

or:

$$P(\mathbf{x}) = 3 + \begin{bmatrix} 1 & 2 \end{bmatrix} \mathbf{x} + \mathbf{x}^T \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \mathbf{x}$$

where $\mathbf{x} = [x_1 \ x_2]^T$.

Stationarity of the gradient requires:

$$\nabla_{\mathbf{x}} P = \begin{bmatrix} 1 & 2 \end{bmatrix} + 2\mathbf{x}^T \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \mathbf{0}$$

$$\therefore \mathbf{x} = -\frac{1}{2} \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} \\ 0 \end{bmatrix}$$

Check the Hessian to classify the type of stationary point:

$$\nabla_{\mathbf{x}}^2 P = 2 \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix}$$

The eigenvalues of the Hessian are:

$$\left| \lambda \mathbf{I} - \begin{bmatrix} 2 & 4 \\ 4 & 2 \end{bmatrix} \right| = \left| \begin{bmatrix} \lambda - 2 & -4 \\ -4 & \lambda - 2 \end{bmatrix} \right| = (\lambda - 2)^2 - 16 = 0$$
$$\therefore \lambda_i = 6, -2$$

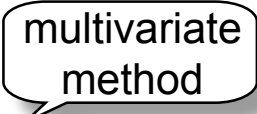
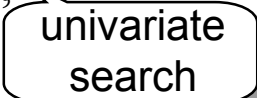
Thus the Hessian is indefinite and the stationary point is a saddle point. This is a trivial example, but it highlights the general procedure for direct use of the optimality conditions.

Unconstrained Multivariate Optimization

Like the univariate search methods we studied earlier, multivariate optimization methods can be separated into two groups:

- those which depend solely on function evaluations,
- those which use derivative information (either analytical derivatives or numerical approximations).

Regardless of which method you choose to solve a multivariate optimization problem, the general procedure will be:

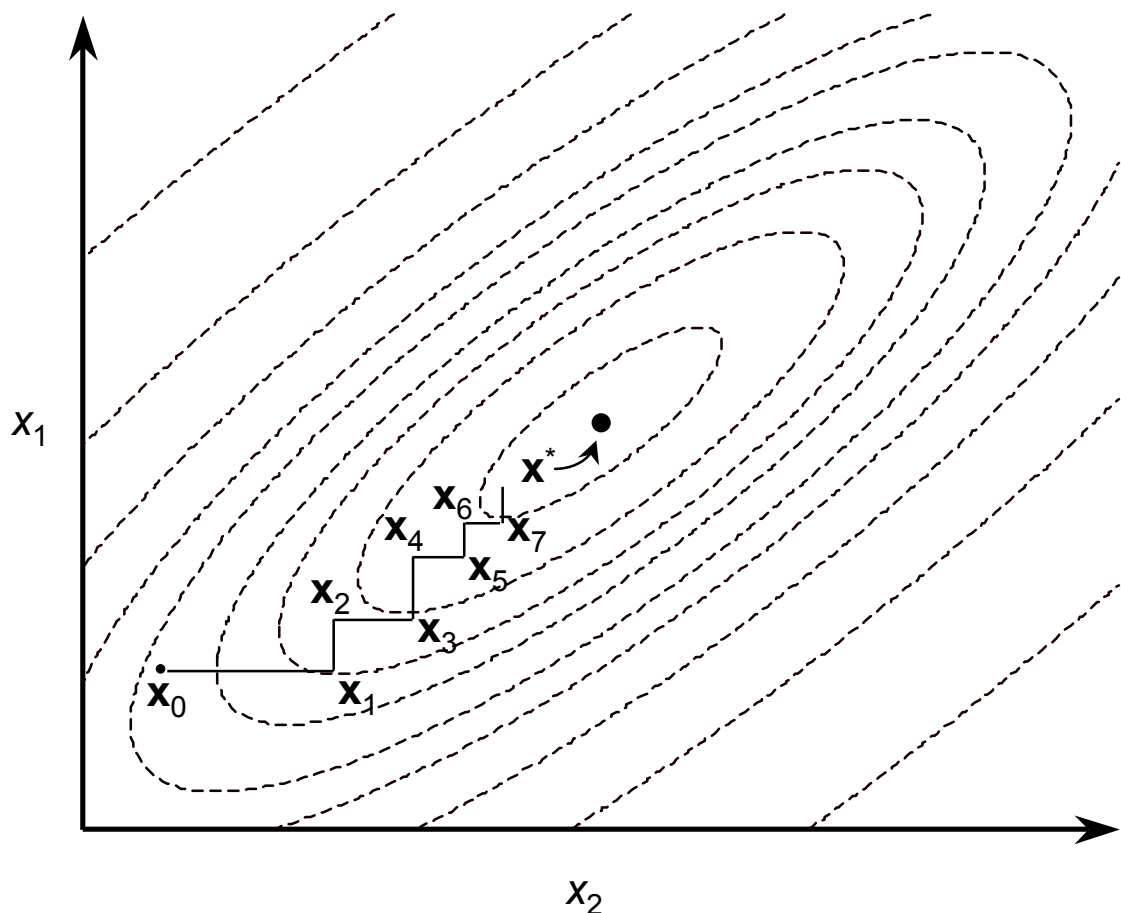
- 1) select a starting point(s), 
- 2) choose a search direction,
- 3) minimize in chosen search direction,
- 4) repeat steps 2 & 3 until converged. 

Also, successful solution of an optimization problem will require specification of a convergence criterion. Some possibilities include:

$$\begin{aligned}\| \mathbf{x}_{k+1} - \mathbf{x}_k \| &\leq \gamma \\ \| P(\mathbf{x}_{k+1}) - P(\mathbf{x}_k) \| &\leq \delta \\ \left\| \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \right\| &\leq \varepsilon\end{aligned}$$

Sequential Univariate Searches

Perhaps the simplest multivariable search technique to implement would be a system of sequential univariate searches along some fixed set of directions. Consider the two dimensional case, where the chosen search directions are parallel to the coordinate axes:

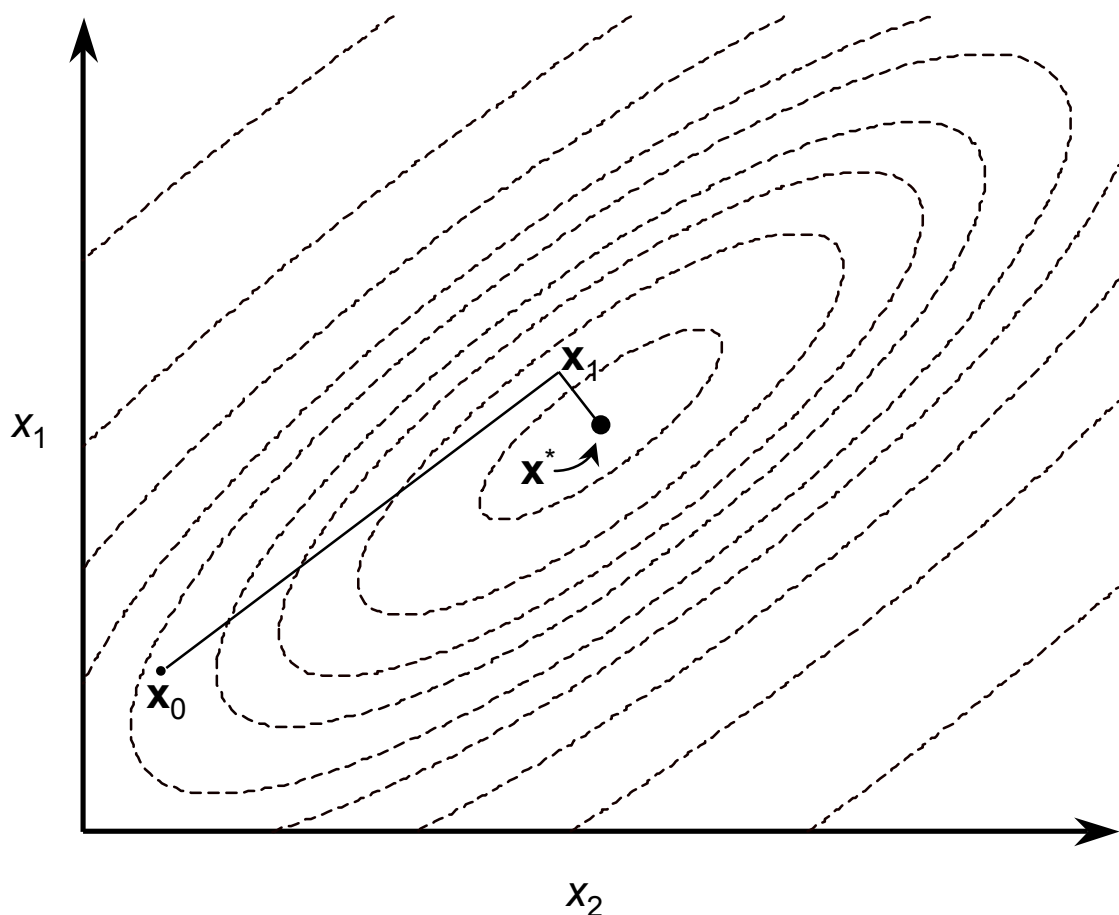


In this algorithm, you:

1. select a starting point \mathbf{x}_0 ,
2. select a coordinate direction (e.g. $\mathbf{s} = [0 \ 1]^T$, or $[1 \ 0]^T$),
3. perform a univariate search $\min_{\alpha} P(\mathbf{x}_k + \alpha \mathbf{s})$,
4. repeat steps 2 and 3 alternating between search directions, until converged.

Sequential Univariate Searches

The problem with this method is that a very large number of iterations may be required to attain a reasonable level of accuracy. If we knew something about the “orientation” of the objective function the rate of convergence could be greatly enhanced. Consider the previous two dimensional problem, but using the independent search directions $([1 \ 1]^T, [1 \ -1]^T)$.



Of course, finding the optimum in n steps only works for quadratic objective functions where the Hessian is known and each line search is exact. There are a large number of these optimization techniques which vary only in the way that the search directions are chosen.

Nelder-Meade Simplex

This approach has nothing to do with the SIMPLEX method of linear programming. The method derives its name from an n -dimensional polytope (and as a result is often referred to as the “polytope” method).

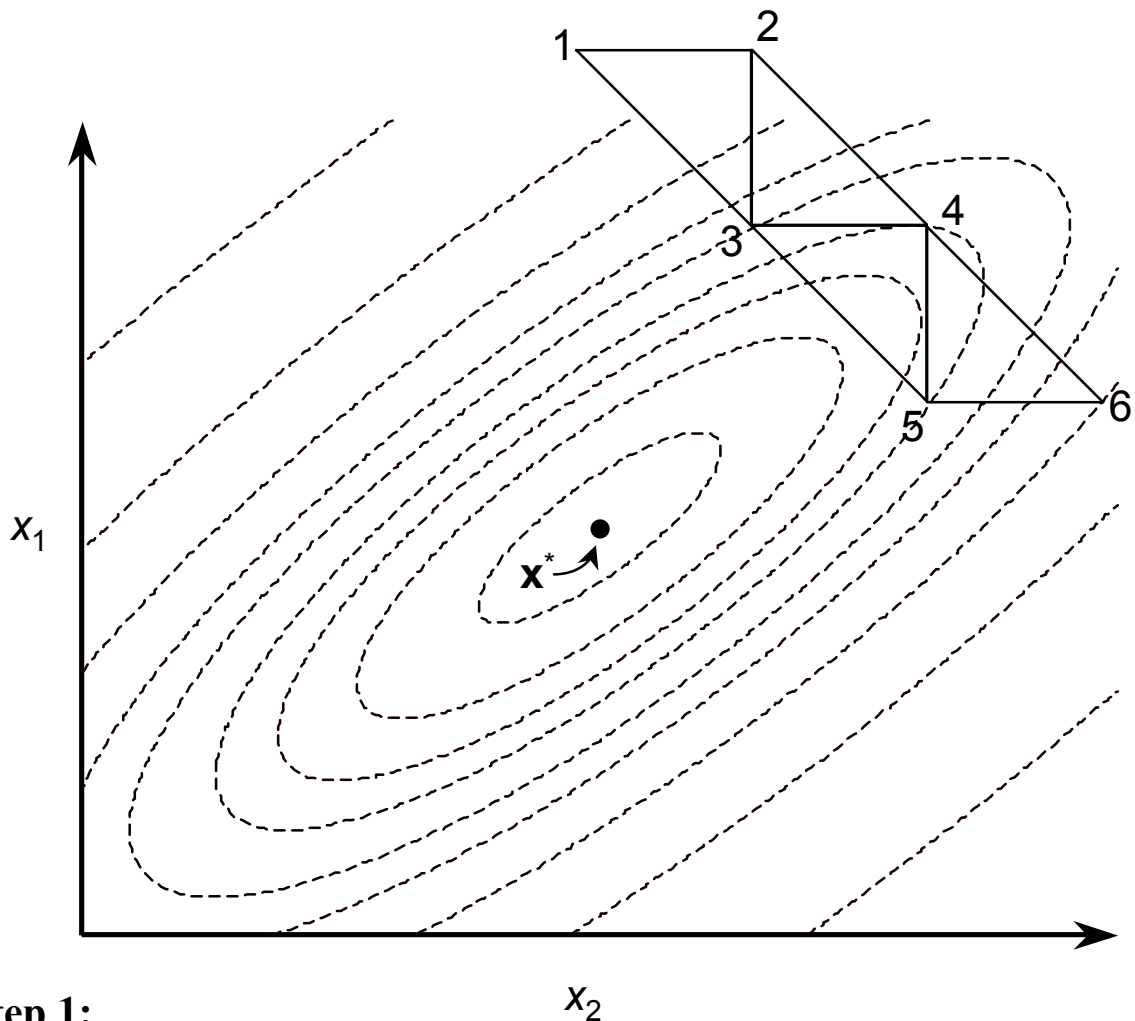
A polytope is an n -dimensional object that has $n+1$ vertices:

n	polytope	number of vertices
1	line segment	2
2	triangle	3
3	tetrahedron	4
\vdots	\vdots	\vdots

It is an easily implemented direct search method, that only relies on objective function evaluations. As a result, it can be robust to some types of discontinuities and so forth. However, it is often slow to converge and is not useful for larger problems (>10 variables).

Nelder-Meade Simplex

To illustrate the basic idea of the method, consider the two-dimensional minimization problem:



Step 1:

- $P_1 = \max (P_1, P_2, P_3)$, define P_4 as the reflection of P_1 through the centroid of the line joining P_2 and P_3 .
- $P_4 \leq \max (P_1, P_2, P_3)$, form a new polytope from the points P_2, P_3, P_4 .

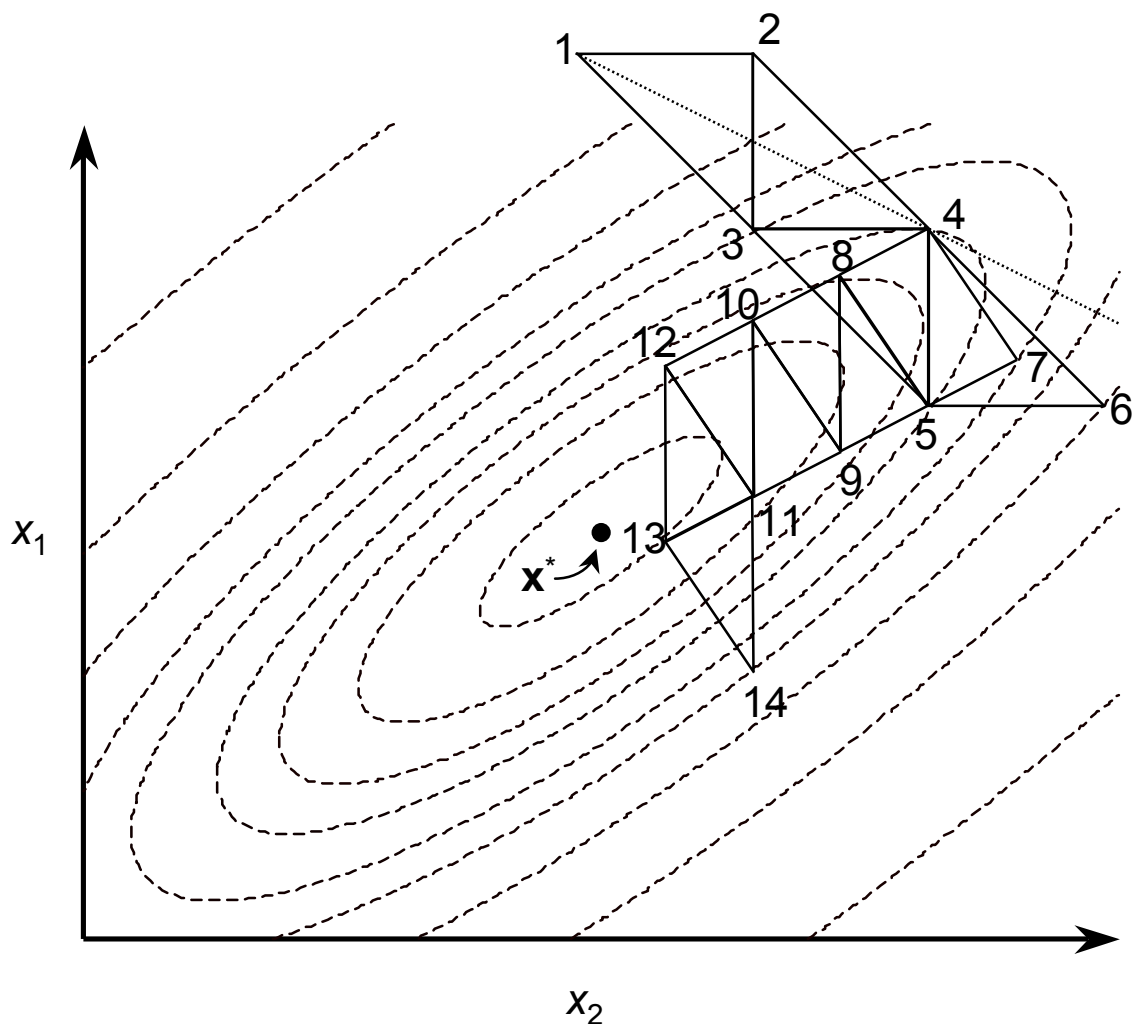
Steps 2 through 4:

- repeat the procedure in Step 1 to form new polytopes.

Nelder-Meade Simplex

We can repeat the procedure until the polytope defined by P_4 , P_5 , P_6 . Further iterations will cause us to flip between two polytopes. We can eliminate these problems by introducing two further operations into the method:

- **contraction** when the reflection step does not offer any improvement,
- **expansion** to accelerate convergence when the reflection step is an improvement.



Nelder-Meade Simplex

For minimization, the algorithm is:

- \mathbf{x}_h is given by $P(\mathbf{x}_h) = \max \{P(\mathbf{x}_1), \dots, P(\mathbf{x}_{n+1})\}$,
- \mathbf{x}_l is given by $P(\mathbf{x}_l) = \min \{P(\mathbf{x}_1), \dots, P(\mathbf{x}_{n+1})\}$,
- calculate the centroid of the vertices other than \mathbf{x}_h as:

$$\mathbf{x}_c = \frac{1}{n-1} \left[\left(\sum_{i=1}^{n+1} \mathbf{x}_i \right) - \mathbf{x}_h \right]$$

- calculate reflection step: $\mathbf{x}_r = 2 \mathbf{x}_c - \mathbf{x}_h$
- if $P(\mathbf{x}_r) < P(\mathbf{x}_l)$ then attempt expansion,

$$\mathbf{x}_e = \mathbf{x}_c + \gamma (\mathbf{x}_r - \mathbf{x}_c)$$

$\gamma > 1$ (usually 2~3, or
could do a line search)

if $P(\mathbf{x}_e) < P(\mathbf{x}_r)$ the expansion was successful,
form the new polytope replacing \mathbf{x}_h with \mathbf{x}_e .

else the expansion step failed,
form the new polytope replacing \mathbf{x}_h with \mathbf{x}_r .

end if

Nelder-Meade Simplex

else

if $P(\mathbf{x}_r) < \max \{P(\mathbf{x}_i) \mid \mathbf{x}_i \neq \mathbf{x}_h\}$ use the reflection,
form the new polytope replacing \mathbf{x}_h with \mathbf{x}_r .

else

the reflection step failed, try a contraction,

$$\mathbf{x}_s = \begin{cases} \mathbf{x}_c + \beta (\mathbf{x}_h - \mathbf{x}_c), & P(\mathbf{x}_r) \geq P(\mathbf{x}_h) \\ \mathbf{x}_c + \beta (\mathbf{x}_r - \mathbf{x}_c), & P(\mathbf{x}_r) < P(\mathbf{x}_h) \end{cases}$$

where $0 < \beta < 1$,
usually $\beta = 1/2$.

if $P(\mathbf{x}_s) < \min \{P(\mathbf{x}_h), P(\mathbf{x}_r)\}$ use contraction,
form the new polytope replacing \mathbf{x}_h with \mathbf{x}_s .

else

repeat contraction.

end if.

repeat the entire procedure until termination.

The Nelder-Meade Simplex method can be slow to converge, but it is useful for functions whose derivatives cannot be calculated or approximated (e.g. some non-smooth functions).

Nelder-Meade Simplex

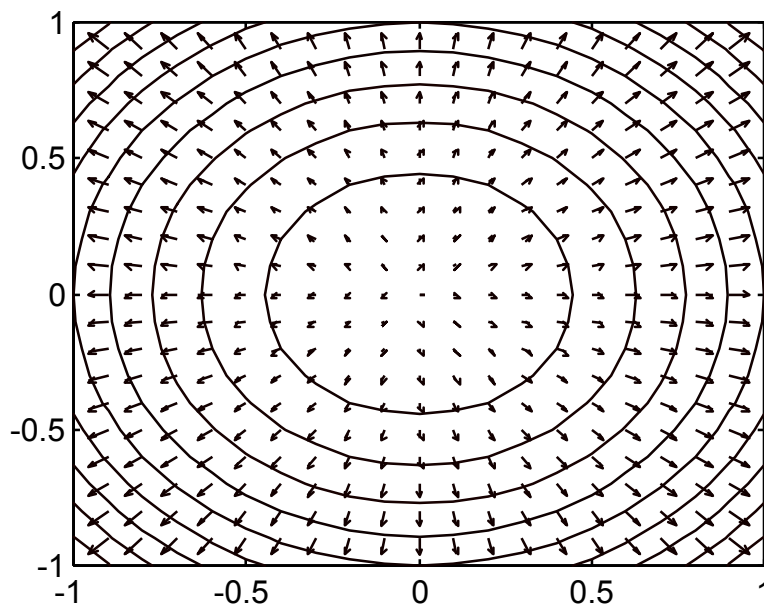
Example

$$\min_{\mathbf{x}} x_1^2 + x_2^2 - 2x_1 - 2x_2 + 2$$

	\mathbf{x}_1	\mathbf{x}_2	\mathbf{x}_3	P_1	P_2	P_3	\mathbf{x}_r	P_r	notes
0	$\begin{bmatrix} 3 \\ 3 \end{bmatrix}$	$\begin{bmatrix} 3 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 2 \end{bmatrix}$	8	5	2	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	1	expansion $\gamma=3$, failed
1	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 2 \end{bmatrix}$	1	5	2	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	0	expansion $\gamma=3$, failed
2	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 2 \\ 2 \end{bmatrix}$	1	0	2	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	1	expansion $\gamma=3$, failed
3	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	1	0	1	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	2	contraction $\beta = \frac{1}{2}$
4	$\begin{bmatrix} \frac{3}{2} \\ \frac{3}{4} \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 1 \\ 0 \end{bmatrix}$	$\frac{5}{16}$	0	1	$\begin{bmatrix} \frac{3}{2} \\ \frac{7}{4} \end{bmatrix}$	$\frac{13}{16}$	expansion $\gamma=3$, failed
5	$\begin{bmatrix} \frac{3}{2} \\ \frac{3}{4} \end{bmatrix}$	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} \frac{3}{2} \\ \frac{7}{4} \end{bmatrix}$	$\frac{5}{16}$	0	$\frac{13}{16}$			

Gradient-Based Methods

This family of optimization methods use first-order derivatives to determine a “descent” direction. (Recall that the gradient gives the direction of quickest increase for the objective function. Thus, the negative of the gradient gives the direction of quickest decrease for the objective function.)



Steepest Descent

It would seem that the fastest way to find an optimum would be to always move in the direction in which the objective function decreases the fastest. Although this idea is intuitively appealing, it is very rarely true; however, with a good line search every iteration will move closer to the optimum.

Gradient-Based Methods

The Steepest Descent algorithm is:

1. choose a starting point \mathbf{x}_0 ,
2. calculate the search direction:

$$\mathbf{s}_k = -\left[\nabla_{\mathbf{x}} P|_{\mathbf{x}_k}\right]$$

3. determine the next iterate for \mathbf{x} :

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k$$

by solving the univariate optimization problem:

$$\min_{\lambda_k} P(\mathbf{x}_k + \lambda_k \mathbf{s}_k)$$

4. repeat steps 2 & 3 until termination. Some termination criteria to consider:

$$\lambda_k \leq \varepsilon$$

$$\|\mathbf{x}_{k+1} - \mathbf{x}_k\| \leq \varepsilon$$

$$\left\| \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \right\| \leq \varepsilon$$

$$\|P(\mathbf{x}_{k+1}) - P(\mathbf{x}_k)\| \leq \varepsilon$$

where ε is some small positive scalar.

Gradient-Based Methods

Steepest Descent Example #1

$$\min_{\mathbf{x}} x_1^2 + x_2^2 - 2x_1 - 2x_2 + 2$$

$$\nabla_{\mathbf{x}} P^T = \begin{bmatrix} 2x_1 - 2 \\ 2x_2 - 2 \end{bmatrix}$$

start at $\mathbf{x}_0 = [3 \ 3]^T$

	\mathbf{x}_k	$\nabla_{\mathbf{x}} P^T _{\mathbf{x}_k}$	λ_k	$P(\mathbf{x}_k)$
0	$\begin{bmatrix} 3 \\ 3 \end{bmatrix}$	$\begin{bmatrix} 4 \\ 4 \end{bmatrix}$?	8
1	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

start at $\mathbf{x}_0 = [1 \ 2]^T$

	\mathbf{x}_k	$\nabla_{\mathbf{x}} P^T _{\mathbf{x}_k}$	λ_k	$P(\mathbf{x}_k)$
0	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 2 \end{bmatrix}$?	1
1	$\begin{bmatrix} 1 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

This example shows that the method of Steepest Descent is very efficient for well-scaled quadratic optimization problems (the profit contours are concentric circles). This method is not nearly as efficient for non-quadratic objective functions or objective functions with very elliptical profit contours (poorly scaled).

Gradient-Based Methods

Steepest Descent Example #2

Consider the minimization problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}$$

Then:

$$\nabla_{\mathbf{x}} P = \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} = [101x_1 - 99x_2 \quad -99x_1 + 101x_2]$$

To develop an objective function for our line search, substitute $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k$ into the original objective function:

$$\begin{aligned} P(\mathbf{x}_k + \lambda_k \mathbf{s}_k) &= \frac{1}{2} (\mathbf{x}_k - \lambda_k \nabla_{\mathbf{x}} P|_{\mathbf{x}_k})^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} (\mathbf{x}_k - \lambda_k \nabla_{\mathbf{x}} P|_{\mathbf{x}_k}) \\ &= \frac{1}{2} \left(\mathbf{x}_k - \lambda_k \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}_k \right)^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \left(\mathbf{x}_k - \lambda_k \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}_k \right) \end{aligned}$$

To simplify our work, let $\mathbf{H} = \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}$, which yields:

$$P(\mathbf{x}_k + \lambda_k \mathbf{s}_k) = \frac{1}{2} [\mathbf{x}_k^T \mathbf{H} \mathbf{x}_k - 2\lambda_k \mathbf{x}_k^T \mathbf{H}^2 \mathbf{x}_k + \lambda_k^2 \mathbf{x}_k^T \mathbf{H}^3 \mathbf{x}_k]$$

This expression is quadratic in λ_k , thus making it easy to perform an exact line search:

$$\frac{dP}{d\lambda_k} = -\mathbf{x}_k^T \mathbf{H}^2 \mathbf{x}_k + \lambda_k \mathbf{x}_k^T \mathbf{H}^3 \mathbf{x}_k = 0$$

Gradient-Based Methods

Solving for the step length yields:

$$\lambda_k = \frac{\mathbf{x}_k^T \mathbf{H}^2 \mathbf{x}_k}{\mathbf{x}_k^T \mathbf{H}^3 \mathbf{x}_k} = \frac{\mathbf{x}_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}^2 \mathbf{x}_k}{\mathbf{x}_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}^3 \mathbf{x}_k}$$

$$= \frac{\mathbf{x}_k^T \begin{bmatrix} 20,002 & -19,998 \\ -19,998 & 20,002 \end{bmatrix} \mathbf{x}_k}{\mathbf{x}_k^T \begin{bmatrix} 4,000,004 & -3,999,996 \\ -3,999,996 & 4,000,004 \end{bmatrix} \mathbf{x}_k}$$

Starting at $\mathbf{x}_0 = [2 \ 1]^T$:

	\mathbf{x}_k	$P(\mathbf{x}_k)$	$\nabla_{\mathbf{x}} P _{\mathbf{x}_k}^T$	λ_k
0	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	54.50	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	0.0050
1	$\begin{bmatrix} 1.4845 \\ 1.4854 \end{bmatrix}$	4.4104	$\begin{bmatrix} 2.8809 \\ 3.0591 \end{bmatrix}$	0.4591
2	$\begin{bmatrix} 0.1618 \\ 0.0809 \end{bmatrix}$	0.3569	$\begin{bmatrix} 8.3353 \\ -7.8497 \end{bmatrix}$	0.0050
3	$\begin{bmatrix} 0.1201 \\ 0.1202 \end{bmatrix}$	0.0289	$\begin{bmatrix} 0.2331 \\ 0.2476 \end{bmatrix}$	0.4591
4	$\begin{bmatrix} 0.0131 \\ 0.0065 \end{bmatrix}$	0.0023	$\begin{bmatrix} 0.6745 \\ -0.6352 \end{bmatrix}$	0.0050
5	$\begin{bmatrix} 0.0097 \\ 0.0097 \end{bmatrix}$	1.8915×10^{-4}	$\begin{bmatrix} 0.0189 \\ 0.0200 \end{bmatrix}$	0.4591
6	$\begin{bmatrix} 0.0011 \\ 0.0005 \end{bmatrix}$	1.5307×10^{-5}	$\begin{bmatrix} 0.0547 \\ -0.0514 \end{bmatrix}$	0.0050
7	$\begin{bmatrix} 7.868 \times 10^{-4} \\ 7.872 \times 10^{-4} \end{bmatrix}$	1.2387×10^{-4}	$\begin{bmatrix} 0.0015 \\ 0.0016 \end{bmatrix}$	0.4591

Gradient-Based Methods

Conjugate Gradients

Steepest Descent was based on the idea that the minimum will be found provided that we always move in the downhill direction. The second example showed that the gradient does not always point in the direction of the optimum, due to the geometry of the problem. Fletcher and Reeves (1964) developed a method which attempts to account for local curvature in determining the next search direction.

The algorithm is:

1. choose a starting point \mathbf{x}_0 ,
2. set the initial search direction as:

$$\mathbf{s}_0 = -\left[\nabla_{\mathbf{x}} P|_{\mathbf{x}_0}\right]$$

3. determine the next iterate for \mathbf{x} :

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k$$

by solving the univariate optimization problem:

$$\min_{\lambda_k} P(\mathbf{x}_k + \lambda_k \mathbf{s}_k)$$

4. calculate the next search direction as:

$$\mathbf{s}_{k+1} = -\nabla_{\mathbf{x}} P|_{\mathbf{x}_k}^T + \mathbf{s}_k \frac{\nabla_{\mathbf{x}} P|_{\mathbf{x}_{k+1}} \nabla_{\mathbf{x}} P|_{\mathbf{x}_{k+1}}^T}{\nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \nabla_{\mathbf{x}} P|_{\mathbf{x}_k}^T}$$

5. repeat steps 3 & 4 until termination.

Gradient-Based Methods

The manner in which the successive search directions are calculated is important. For quadratic functions, these successive search directions are conjugate with respect to the Hessian. This means that for the quadratic function:

$$P(\mathbf{x}) = a + \mathbf{b}^T \mathbf{x} + \frac{1}{2} \mathbf{x}^T \mathbf{H} \mathbf{x}$$

successive search directions satisfy:

$$\mathbf{s}_k^T \mathbf{H} \mathbf{s}_{k+1} = 0$$

As a result, these successive search directions have incorporated within them information about the geometry of the optimization problem.

Conjugate Gradients Example

Consider again the optimization problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}$$

We saw that Steepest Descent was slow to converge on this problem because of its poor scaling. As in the previous example, we will use an exact line search. It can be shown that the optimal step length is:

$$\lambda_k = - \frac{\mathbf{x}_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{s}_k}{\mathbf{s}_k^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{s}_k}$$

Gradient-Based Methods

Starting at $\mathbf{x}_0 = [2 \ 1]^T$:

	\mathbf{x}_k	$P(\mathbf{x}_k)$	$\nabla_{\mathbf{x}} P _{\mathbf{x}_k}^T$	\mathbf{s}_k	λ_k
0	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	54.50	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	0.0080
1	$\begin{bmatrix} 1.4845 \\ 1.4854 \end{bmatrix}$	4.4104	$\begin{bmatrix} 2.8809 \\ 3.0591 \end{bmatrix}$	$\begin{bmatrix} -2.9717 \\ -2.9735 \end{bmatrix}$	0.4888
2	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0			

Notice the much quicker convergence of the algorithm. For a quadratic objective function with exact line searches, the Conjugate Gradients method exhibits quadratic convergence.

The quicker convergence comes at an increased computational requirement. These include:

- a more complex search direction calculation,
- increased storage for maintaining previous search directions and gradients.

These are usually small in relation to the performance improvements.

Newton's Method

The Conjugate Gradients method showed that there was a considerable performance gain to be realized by incorporating some problem geometry into the optimization algorithm. However, the method did this in an approximate sense. Newton's method does this by using second-order derivative information.

In the multivariate case, Newton's method determines a search direction by using the Hessian to modify the gradient. The method is developed directly from the Taylor Series approximation of the objective function. Recall that an objective function can be locally approximated as:

$$P(\mathbf{x}) \approx P(\mathbf{x}_k) + \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} (\mathbf{x} - \mathbf{x}_k) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_k)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} (\mathbf{x} - \mathbf{x}_k)$$

Then, stationarity can be approximated as:

$$\nabla_{\mathbf{x}} P(\mathbf{x}) \approx \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} + (\mathbf{x} - \mathbf{x}_k)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} = \mathbf{0}$$

Which can be used to determine an expression for calculating the next point in the minimization procedure:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} \right)^{-1} \left(\nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \right)^T$$

Notice that in this case we get both the direction and the step length for the minimization. Further, if the objective function is quadratic, the optimum is found in a single (Newton Step) without a line search.

Newton's Method

At a given point \mathbf{x}_k , the algorithm for Newton's method is:

1. calculate the gradient at the current point $\nabla_{\mathbf{x}} P|_{\mathbf{x}_k}$,
2. calculate the Hessian at the current point $\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k}$,
3. calculate the Newton step:

$$\Delta \mathbf{x}_k = - \left(\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} \right)^{-1} \left(\nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \right)^T$$

4. calculate the next point:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$$

5. repeat steps 1 through 4 until termination.

Newton's Method Example

Consider once again the optimization problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}$$

The gradient and Hessian for this example is:

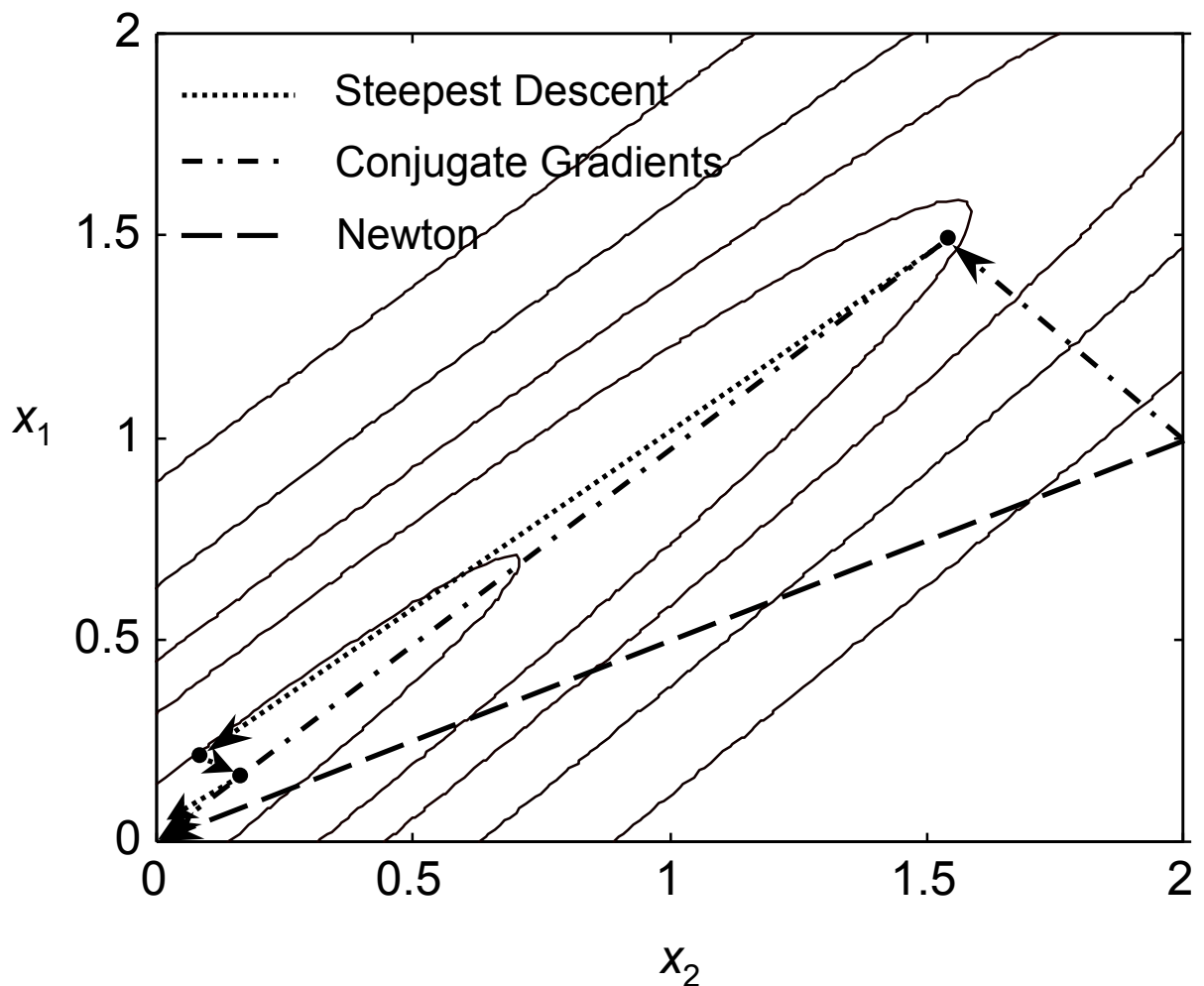
$$\nabla_{\mathbf{x}} P = \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}, \quad \nabla_{\mathbf{x}}^2 P = \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}$$

Newton's Method

Starting at $\mathbf{x}_0 = [2 \ 1]^T$:

	\mathbf{x}_k	$P(\mathbf{x}_k)$	$\nabla_{\mathbf{x}} P _{\mathbf{x}_k}^T$	$\nabla_{\mathbf{x}}^2 P _{\mathbf{x}_k}$
0	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	54.50	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	$\begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}$
1	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$	0		

Comparing the three methods on this problem:



Newton's Method

As expected Newton's method was very effective, even though the problem was not well scaled.

- poor scaling was compensated for by using the inverse of the Hessian.

Steepest Descent was very good for the first two iterations, when we were quite far from the optimum. But it slowed rapidly as:

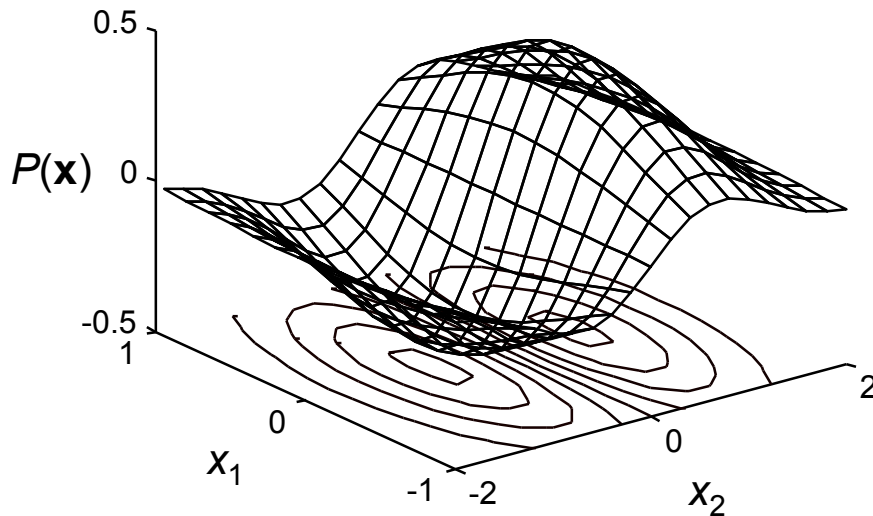
$$\mathbf{x}_k \rightarrow \mathbf{x}^* \quad \because \quad \nabla_x P|_{\mathbf{x}_k} \rightarrow \mathbf{0}$$

The Conjugate Gradients approach avoided this difficulty by including some measure of problem geometry in the algorithm. However, the method started out using the Steepest Descent direction and built in curvature information as the algorithm progressed to the optimum. For higher-dimensional and non-quadratic objective functions, the number of iterations required for convergence can increase substantially.

In general, Newton's Method will yield the best convergence properties, at the cost of increased computational load to calculate and store the Hessian. The Conjugate Gradients approach can be effective in situations where computational requirements are an issue. However, there are some other methods where the Hessian is approximated using gradient information and the approximate Hessian is used to calculate a Newton step. These are called the Quasi-Newton methods.

Newton's Method

As we saw in the univariate case, care must be taken when using Newton's method for complex functions:



Since Newton's method is searching for a stationary point, it can oscillate between such points in the above situation. For such complex functions, Newton's method is often implemented with an alternative to step 4.

4'. calculate the next point:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \Delta \mathbf{x}_k$$

where the step length is determined as:

- i) $0 < \lambda_k < 1$ (so we don't take a full step),
- ii) perform a line search on λ_k .

Generally, Newton and Newton-like methods are preferred to other methods, the main difficulty is determining the Hessian.

Approximate Newton Methods

As we saw in the univariate case the derivatives can be approximated by finite differences. In multiple dimensions this can require substantially more calculations. As an example, consider forward difference approximation of the gradient:

$$\nabla_{\mathbf{x}} P|_{\mathbf{x}_k} = \left[\frac{\partial P}{\partial x_i} \right]_{\mathbf{x}_k} \approx \frac{P(\mathbf{x}_k + \delta_i) - P(\mathbf{x}_k)}{\|\delta_i\|}$$

where δ_i is a perturbation in the direction of x_i . This approach requires 1 additional objective function evaluation per dimension for forward or backward differencing, and 2 additional objective function evaluations per dimension for central differencing.

To approximate the Hessian using only finite differences in the objective function could have the form:

$$\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} = \left[\frac{\partial^2 P}{\partial x_i \partial x_j} \right]_{\mathbf{x}_k} \approx \frac{[P(\mathbf{x}_k + \delta_i) - P(\mathbf{x}_k)] - [P(\mathbf{x}_k + \delta_j) - P(\mathbf{x}_k)]}{\|\delta_i\| \|\delta_j\|}$$

Finite difference approximation of the Hessian requires at least an additional 2 objective function evaluations per dimension. Some differencing schemes will give better performance than others; however, the increased computational load required for difference approximation of the second derivatives precludes the use of this approach for larger problems.

Approximate Newton Methods

Alternatively, the Hessian can be approximated in terms of gradient information. In the forward difference case the Hessian can be approximated as:

$$\nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} = \left[\frac{\partial P}{\partial x_i} \right]_{\mathbf{x}_k} \approx \mathbf{h}_i = \frac{\nabla_{\mathbf{x}} P|_{\mathbf{x}_k + \delta_i} - \nabla_{\mathbf{x}} P|_{\mathbf{x}_k}}{\|\delta_i\|}$$

Then, we can form a finite difference approximation to the Hessian as:

$$\tilde{\mathbf{H}} = [\mathbf{h}_i]$$

The problem is that often this approximate Hessian is non-symmetric. A symmetric approximation can be formed as:

$$\hat{\mathbf{H}} = \frac{1}{2}[\tilde{\mathbf{H}} + \tilde{\mathbf{H}}^T]$$

Whichever approach is used to approximate the derivatives, the Newton step is calculated from them. In general, Newton's methods based on finite differences can produce results similar to analytical results, providing care is taken in choosing the perturbations δ_i . (Recall that as the algorithm proceeds:

$$\mathbf{x}_k \rightarrow \mathbf{x}^* \quad \text{and} \quad \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} \rightarrow \mathbf{0}.$$

Then small approximation errors will affect convergence and accuracy.)

Some alternatives which can require fewer computations, yet build curvature information as the algorithms proceed are the *Quasi-Newton* family.

Quasi-Newton Methods

This family of methods are the multivariate analogs of the univariate Secant methods. They build second derivative (Hessian) information as the algorithm proceeds to solution, using available gradient information.

Consider the Taylor series expansion for the gradient of the objective function along the step direction \mathbf{s}_k :

$$\nabla_{\mathbf{x}} P|_{\mathbf{x}_{k+1}} = \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} + (\Delta \mathbf{x}_k)^T \nabla_{\mathbf{x}}^2 P|_{\mathbf{x}_k} + \dots$$

Truncating the Taylor series at the second-order terms and rearranging slightly yields the so-called *Quasi-Newton Condition*:

$$(\Delta \mathbf{x}_k)^T \hat{\mathbf{H}}_{k+1} = \nabla_{\mathbf{x}} P|_{\mathbf{x}_{k+1}} - \nabla_{\mathbf{x}} P|_{\mathbf{x}_k} = \Delta \mathbf{g}_k$$

Thus, any algorithm which satisfies this condition can build up curvature information as it proceeds from \mathbf{x}_k to \mathbf{x}_{k+1} in the direction $\Delta \mathbf{x}_k$; however, the curvature information is gained in only one direction. Then as these algorithms proceed from iteration-to-iteration, the approximate Hessians can differ by only a rank-one matrix:

$$\hat{\mathbf{H}}_{k+1} = \hat{\mathbf{H}}_k + \mathbf{v} \mathbf{u}^T$$

Combining the Quasi-Newton Condition and the update formula yields:

$$(\Delta \mathbf{x}_k)^T \hat{\mathbf{H}}_{k+1} = (\Delta \mathbf{x}_k)^T (\hat{\mathbf{H}}_k + \mathbf{v} \mathbf{u}^T) = \Delta \mathbf{g}_k$$

Quasi-Newton Methods

Which yields a solution for the vector \mathbf{u} of:

$$\mathbf{u}^T = \frac{1}{\Delta \mathbf{x}_k^T \mathbf{v}} \left[\Delta \mathbf{g}_k - \Delta \mathbf{x}_k^T \hat{\mathbf{H}}_k \right]$$

and a Hessian update formula:

$$\hat{\mathbf{H}}_{k+1} = \hat{\mathbf{H}}_k + \frac{1}{\Delta \mathbf{x}_k^T \mathbf{v}} \mathbf{v} \left[\Delta \mathbf{g}_k - \Delta \mathbf{x}_k^T \hat{\mathbf{H}}_k \right]$$

providing $\Delta \mathbf{x}_k$ and \mathbf{v}
are not orthogonal

Unfortunately this update formula is not unique with respect to the *Quasi-Newton Condition* and the step vector \mathbf{s}_k . We could add an additional term of $\mathbf{w}\mathbf{z}^T$ to the update formula, where the vector \mathbf{w} is any vector from the null space of $\Delta \mathbf{x}_k$. (Recall that if $\mathbf{w} \in \text{null}(\Delta \mathbf{x}_k)$, then $\Delta \mathbf{x}_k^T \mathbf{w} = 0$). Thus there are a family of such Quasi-Newton methods which differ only in this additional term $\mathbf{w}\mathbf{z}^T$.

Perhaps the most successful member of the family is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update:

$$\hat{\mathbf{H}}_{k+1} = \hat{\mathbf{H}}_k + \frac{\Delta \mathbf{g}_k \Delta \mathbf{g}_k^T}{\Delta \mathbf{g}_k^T \Delta \mathbf{x}_k} - \frac{\hat{\mathbf{H}}_k \Delta \mathbf{x}_k \Delta \mathbf{x}_k^T \hat{\mathbf{H}}_k}{\Delta \mathbf{x}_k^T \hat{\mathbf{H}}_k \Delta \mathbf{x}_k}$$

The BFGS update has several advantages including that it is symmetric and it can be further simplified if the step is calculated according to:

$$\hat{\mathbf{H}}_k \Delta \mathbf{x}_k = -\lambda_k \nabla_{\mathbf{x}} P|_{\mathbf{x}_k}^T$$

Quasi-Newton Methods

In this case the BFGS update formula simplifies to:

$$\hat{\mathbf{H}}_{k+1} = \hat{\mathbf{H}}_k + \frac{\Delta \mathbf{g}_k \Delta \mathbf{g}_k^T}{\Delta \mathbf{g}_k \Delta \mathbf{x}_k} + \frac{\lambda_k \nabla_{\mathbf{x}} P|_k^T \nabla_{\mathbf{x}} P|_k}{\nabla_{\mathbf{x}} P|_k \Delta \mathbf{x}_k}$$

General Quasi-Newton Algorithm

1. Initialization:

- choose a starting point \mathbf{x}_0 ,
- choose an initial Hessian approximation (usually $\hat{\mathbf{H}}_0 = \mathbf{I}$).

2. At iteration k :

- calculate the gradient $\nabla_{\mathbf{x}} P|_{\mathbf{x}_k}$,
- calculate the search direction:
$$\mathbf{s}_k = -(\hat{\mathbf{H}}_k)^{-1} (\nabla_{\mathbf{x}} P|_{\mathbf{x}_k})^T$$

- calculate the next iterate:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k$$

using a line search on λ_k .

- compute $\nabla_{\mathbf{x}} P|_{\mathbf{x}_k}$, $\Delta \mathbf{g}_k$, $\Delta \mathbf{x}_k$.
- update the Hessian ($\hat{\mathbf{H}}_{k+1}$) using the method of your choice (e.g. BFGS).

3. Repeat step 2 until termination.

Quasi-Newton Methods

BFGS Example

Consider once again the optimization problem:

$$\min_{\mathbf{x}} \quad \frac{1}{2} \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix} \mathbf{x}$$

Recall that the gradient for this example is:

$$\nabla_{\mathbf{x}} P = \mathbf{x}^T \begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}$$

As before starting at $\mathbf{x}_0 = [2 \ 1]^T$:

	\mathbf{x}_k	$P(\mathbf{x}_k)$	$\nabla_{\mathbf{x}} P _{\mathbf{x}_k}^T$	$\hat{\mathbf{H}}_k$	\mathbf{s}_k	λ_k
0	$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$	54.5	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 103 \\ -97 \end{bmatrix}$	0.0050
1	$\begin{bmatrix} 1.4845 \\ 1.4854 \end{bmatrix}$	4.4104	$\begin{bmatrix} 2.8809 \\ 3.0591 \end{bmatrix}$	$\begin{bmatrix} 100.5291 & -99.5 \\ -99.5 & 100.4681 \end{bmatrix}$	$\begin{bmatrix} -2.9717 \\ -2.9735 \end{bmatrix}$	0.4996
2	$\begin{bmatrix} 7.810 \times 10^{-4} \\ 7.815 \times 10^{-4} \end{bmatrix}$	1.2207×10^{-8}	$\begin{bmatrix} 1.516 \times 10^{-4} \\ 1.609 \times 10^{-4} \end{bmatrix}$	$\begin{bmatrix} 101 & -99 \\ -99 & 101 \end{bmatrix}$	$\begin{bmatrix} -7.810 \times 10^{-4} \\ -7.815 \times 10^{-4} \end{bmatrix}$	1.0000
3	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$					

Quasi-Newton Methods

It is worth noting that the optimum is found in 3 steps (although we are very close within 2). Also, notice that we have a very accurate approximation for the Hessian (to 7 significant figures) after 2 updates.

The BFGS algorithm was:

- not quite as efficient as Newton's method,
- approximately as efficient as the Conjugate Gradients approach,
- much more efficient than Steepest Descent.

This is what should be expected for quadratic objective functions of low dimension. As the dimension of the problem increases, the Newton and Quasi-Newton methods will usually out perform the other methods.

A cautionary note:

The BFGS algorithm guarantees that the approximate Hessian remains positive definite (and invertible) providing that the initial matrix is chosen as positive definite, in theory.

However, round-off errors and sometimes the search history can cause the approximate Hessian to become badly conditioned. When this occurs, the approximate Hessian should be reset to some value (often the identity matrix \mathbf{I}).

Convergence

During our discussions we have mentioned convergence properties of various algorithms. There are a number of ways with which convergence properties of an algorithm can be characterized. The most conventional approach is to determine the asymptotic behaviour of the sequence of distances between the iterates (\mathbf{x}_k) and the optimum (\mathbf{x}^*), as the iterations of a particular algorithm proceed. Typically, the distance between an iterate and the optimum is defined as:

$$\left\| \mathbf{x}_k - \mathbf{x}^* \right\|$$

We are most interested in the behaviour of an algorithm within some neighbourhood of the optimum; hence, the asymptotic analysis. The *asymptotic rate of convergence* is defined as:

$$\lim_{k \rightarrow \infty} \frac{\left\| \mathbf{x}_{k+1} - \mathbf{x}^* \right\|}{\left\| \mathbf{x}_k - \mathbf{x}^* \right\|^p} = \beta$$

with the *asymptotic order* p and *asymptotic error* β .

In general, most algorithms show convergence properties which are within three categories:

- linear ($0 \leq \beta < 1, p = 1$),
- superlinear ($\beta = 0, p = 1$),
- quadratic ($\beta \geq 0, p = 2$),

Convergence

For the methods we have studied, it can be shown that:

- 1) the Steepest Descent method exhibits superlinear convergence in most cases. For quadratic objective functions the asymptotic convergence properties can be shown to approach quadratic as $\kappa(\mathbf{H}) \rightarrow 1$ and linear as $\kappa(\mathbf{H}) \rightarrow \infty$.
- 2) the Conjugate Gradients method will generally show superlinear (and often nearly quadratic) convergence properties.
- 3) Newton's method converges quadratically.
- 4) Quasi-Newton methods can closely approach quadratic rates of convergence.

Unconstrained Multivariate Optimization

As in the univariate case, there are two basic approaches:

- direct search (Nelder-Meade Simplex, ...),
- derivative-based (Steepest Descent, Newton's ...).

Choosing which method to use for a particular problem is a trade-off among:

- ease of implementation,
- convergence speed and accuracy,
- computational limitations.

Unfortunately there is no 'best' method in all situations. For nearly quadratic functions Newton's and Quasi-Newton methods will usually be very efficient. For nearly flat or non-smooth objective functions, the direct search methods are often a good choice. For very large problems (more than 1000 decision variables), algorithms which use only gradient information (Steepest Descent, Conjugate Gradients, Quasi-Newton, etc.) can be the only practical methods.

Remember that the solution to a multivariable optimization problem requires:

- a function (and derivatives) that can be evaluated,
- an appropriate optimization method,
- a good starting point,
- a well chosen termination criterion.

Unconstrained Optimization Problems

1. For the Wastewater settling pond problem discussed in class, we found that the outlet concentration could be described as:

$$c_0(t) = c_i t e^{-\frac{t}{\tau}}$$

Assuming $\tau = 4$ hours and $c_i = 100$ grams/liter, please do each of the following:

- a) Find the maximum outlet concentration using Matlab's "fminbnd" and "fminunc" commands.
- b) Determine the sensitivity of the maximum outlet concentration and time at which the maximum occurs to the assumed value of τ .
- c) Determine the sensitivity of the maximum outlet concentration and time at which the maximum occurs to the assumed value of c_i .

Unconstrained Optimization Problems

2. The file “inhibit.mat” contains data from experiments that relate the consumption rate (μ) of substrate (hr^{-1}) in a wastewater treatment plant to the concentration of the substrate (S) in the feed (grams/litre) to the wastewater treatment plant. The relationship is expected to be:

$$\mu = \frac{\mu_{\max} S}{K_m + S + K_1 S^2}$$

Using the given data, please do each of the following:

- Estimate the model parameters μ , K_m , and K_1 using Matlab’s “lsqcurvefit” command.
- The last data point in the supplied data is expected to be an outlier. Determine how sensitive the parameters estimates are to the suspected outlier?

Constrained multivariable NLP

- Until now we have only been discussing NLP that do not have constraints. Most problems that we wish to solve will have some form of constraints.
- - equality constraints (material and energy balances)
 - inequality constraints (limits on operating conditions)
- These type of problems require techniques that rely heavily on the unconstrained methods.