Lecture 34
Minimization and maximization of functions

- Introduction
- Golden section search
- Parabolic interpolation
- Search with first derivatives
- Downhill simplex method

Introduction

In a nutshell, you are given a function f of one or more variables and you wish to determine the points where the function is maximum or minimum and evaluate the function there.

Obviously minimizing and maximizing can be made equivalent by changing f -> -f.

In many cases the computational cost resides in evaluating f. The best methods would therefore be the ones that find the maximum or minimum with less evaluations of the function.

Global extrema are considerably harder to find than local ones. Heuristically there are a couple of ways to proceed: 1) find local minima starting from widely separated initial guesses and choose the smallest local minima; 2) Perturb significantly the local minimum and see if one is driven to another local minima that is less than the previous one. *Simulated annealing* has proved successful in finding global extrema. We will discuss it soon.

This area of numerical analysis sometimes is called *optimization*.

There is no perfect optimization algorithm. So we will cover several methods. The general strategy is to try out more than one method to see which one works best for your particular problem. You may favor some methods over others depending on features of your problem.

For instance, you may choose between methods that only require evaluating the function and methods that also require evaluating derivatives of the function. In higher dimensions, the derivative is a gradient. In general methods involving derivatives work better. But not necessarily so much better to justify the extra computational cost of evaluating the derivative.

For one dimensional functions one uses bracketing and then *Brent's* method, which we will discuss. If your function has discontinuous second or lower derivatives then the *golden section search* is the way to go.

In multidimensions one must choose between methods that require storage of order N^2 and those that require order N, with N the number of dimensions. For lower values of N this is not much of an issue, but for larger values it becomes crucial.

Golden section in one dimension

Similar to finding roots of a function by bracketing. How to bracket a minimum? One needs three points a<b<c (or c<b<a). If f(b)<f(c) and f(b)<f(a) then there is a minimum in the interval [a,c].

The analog of the bisection method is to choose a fourth point x either between a and b or b and c. Let's say we choose the latter. Then if f(b)<f(x) then the new bracketing triplet of points is (a,b,x). On the other hand if f(b)>f(x) then the new bracketing triple is (b,x,c). In all cases the middle point of the triplet is the abscissa whose ordinate is the best approximation to a minimum achieved so far.

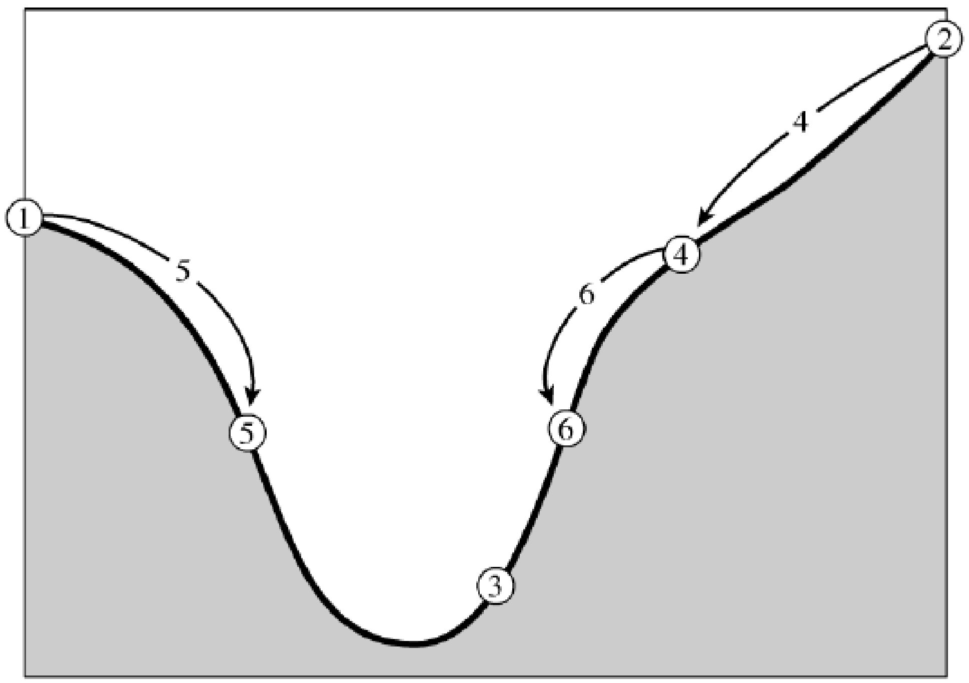


Figure 10.1.1. Successive bracketing of a minimum. The minimum is originally bracketed by points 1,3,2. The function is evaluated at 4, which replaces 2; then at 5, which replaces 1; then at 6, which replaces 4. The rule at each stage is to keep a center point that is lower than the two outside points. After the steps shown, the minimum is bracketed by points 5,3,6.

The process is continued until the distance between the two outer points is small enough.

What is "small enough"? For a minimum located at b, one would think that one will be able to bracket it between [(1-ε)b,(1+ε)b] with ε the machine's floating point precision (10^{-8} single precision. 10^{-15} in double precision). Not so!

In general close to a minimum your function f(x) will be given by Taylor's theorem by,

$$f(x) \approx f(b) + \frac{1}{2} f''(b)(x-b)^2$$

And if,

$$|x-b| < \sqrt{\epsilon}|b| \sqrt{\frac{2\,|f(b)|}{b^2 f''(b)}}$$

The second term will be a factor ε smaller than the first and therefore negligible. (The reason we multiply and divide by b in the right hand side is that for most functions the square root is of order one).

Therefore as a rule of thumb it is useless to try to bracket the root with a width less than the square root of the machine precision, i.e. 10^{-4} in single precision, 10^{-8} in double precision.

We need a strategy for choosing a new point x given (a,b,c). Suppose that b is a fraction w of the way between a and c, that is,

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

And our new point is a fraction z beyond b, $$\frac{x-b}{c-a} = z$$

Then our next bracketing segment will either be of length w+z relative to the current one or of length 1-w. If we want to minimize the worst case possibility we choose z to make them equal, that is z=1-2w. We see that the point is therefore symmetric to b in the original interval, that is |b-a|=|x-c|.

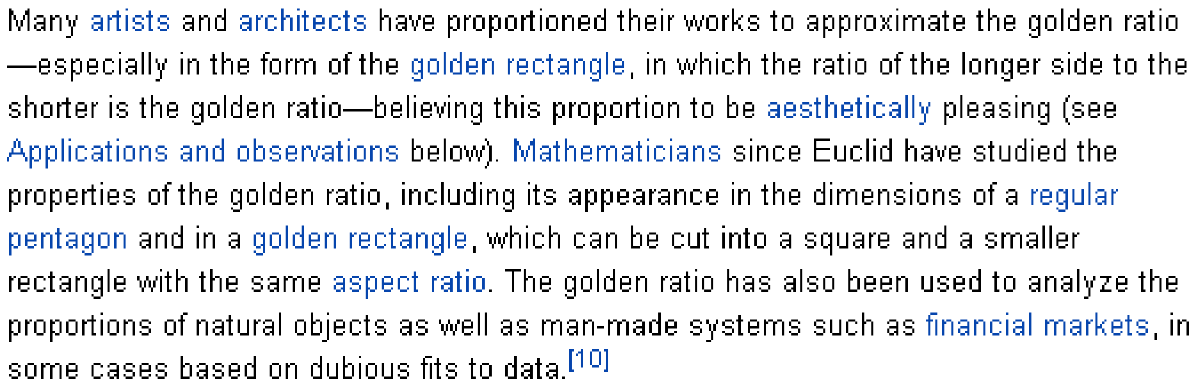
But where in the larger segment? Where do we get w? If we assume scale similarity, if z is chosen to be optimal, then so was w in the previous step. In other words,

$$\frac{z}{1-w} = w$$

Combining with z=1-2w this gives a quadratic equation with a positive root of 0.38197.

$$w = \frac{3-\sqrt{5}}{2} \approx 0.38197$$

Is a number related to the *golden mean* or *golden section* and it appears in many places in mathematics going back all the way to Pythagoras.


Many artists and architects have proportioned their works to approximate the golden ratio—especially in the form of the golden rectangle, in which the ratio of the longer side to the shorter is the golden ratio—believing this proportion to be aesthetically pleasing (see Applications and observations below). Mathematicians since Euclid have studied the properties of the golden ratio, including its appearance in the dimensions of a regular pentagon and in a golden rectangle, which can be cut into a square and a smaller rectangle with the same aspect ratio. The golden ratio has also been used to analyze the proportions of natural objects as well as man-made systems such as financial markets, in some cases based on dubious fits to data.[10]

Many of the proportions of the Parthenon are alleged to exhibit the golden ratio.

This proposal for function minimization is known as the golden section approach:

Given at each stage a bracketing triplet of points, the next point to be tried is that which a fraction of 038197 into the larger of the two intervals (measuring from the central point of the triplet). If your original triplet was not in a golden ratio, the procedure will converge towards the golden ratio.

The procedure will (after self replication is achieved) bracket the minimum just 0.61803 times the size of the preceding interval. This is comparable to the 0.5 that one has in finding roots by bisection. The convergence is linear in the sense that successive significant figures are won linearly with additional function evaluations.

Bracketing the initial guess for the minimum

Up to now we have assumed that somehow we had an initial bracket for the minimum. But this can be the trickiest part of finding the minimum. Some algorithms work without bracketing and in that case you do not have to worry about this step. However, it is far safer to work with algorithms that know for sure there is a minimum and those require a bracket.

One possibility is to “step downhill” with steps that are increasingly large. One can, for instance, parabolically exptrapolate from the previous stepsizes to guess the next one. We have the first and second point of the bracket, we just need to take a step long enough to stop the downhill trend and get a third point.

```
      FUNCTION golden(ax,bx,cx,f,tol,xmin)
      REAL golden,ax,bx,cx,tol,xmin,f,R,C
      EXTERNAL f
      PARAMETER (R=.61803399,C=1.-R)
         Given a function f, and given a bracketing triplet of abscissas ax, bx, cx (such that bx is
         between ax and cx, and f(bx) is less than both f(ax) and f(cx)), this routine performs
         a golden section search for the minimum, isolating it to a fractional precision of about
         tol. The abscissa of the minimum is returned as xmin, and the minimum function value
         is returned as golden, the returned function value.
         Parameters: The golden ratios.
      REAL f1,f2,x0,x1,x2,x3
      x0=ax                            At any given time we will keep track of four points, x0,x1,x2,x3.


      x3=cx
      if(abs(cx-bx).gt.abs(bx-ax))then        Make x0 to x1 the smaller segment,
          x1=bx
          x2=bx+C*(cx-bx)               and fill in the new point to be tried.
      else
          x2=bx
          x1=bx-C*(bx-ax)
      endif
      f1=f(x1)                          The initial function evaluations. Note that we never need to
      f2=f(x2)                              evaluate the function at the original endpoints.
1     if(abs(x3-x0).gt.tol*(abs(x1)+abs(x2)))then   Do-while loop: we keep returning here.
          if(f2.lt.f1)then              One possible outcome,
              x0=x1                     its housekeeping,
              x1=x2
              x2=R*x1+C*x3
              f1=f2
              f2=f(x2)                  and a new function evaluation.
          else                          The other outcome,
              x3=x2
              x2=x1
              x1=R*x2+C*x0
              f2=f1
              f1=f(x1)                  and its new function evaluation.
          endif
      goto 1                            Back to see if we are done.
      endif
      if(f1.lt.f2)then                  We are done. Output the best of the two current values.
          golden=f1
          xmin=x1
      else
          golden=f2
          xmin=x2
      endif
      return
      END
```

Parabolic interpolation and Brent's method in one dimension

Let's go into some more detail about the already mentioned parabolic interpolation.

The golden ratio is prepared to handle the worst scenario, hunting down the minimum and cornering "like a scared rabbit". But why assume the worse? Most functions near a minimum behave like a parabola. Therefore fitting a parabola through the given three points ought to take us in a single leap to the minimum, or very close to it.

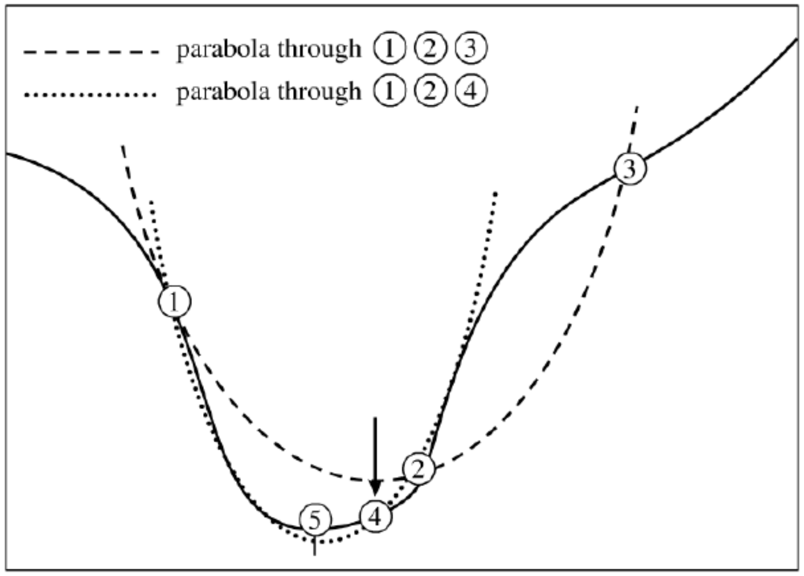


Figure 10.2.1. Convergence to a minimum by inverse parabolic interpolation. A parabola (dashed line) is drawn through the three original points 1,2,3 on the given function (solid line). The function is evaluated at the parabola's minimum, 4, which replaces point 3. A new parabola (dotted line) is drawn through points 1,4,2. The minimum of this parabola is at 5, which is close to the minimum of the function.

The formula for the abscissa that corresponds to the minimum of a parabola fitted through three points (a,f(a)),(b,f(b)),(c,f(c)) is easily derived:

$$x = b - \frac{1}{2}\,\frac{(b-a)^2[f(b)-f(c)] - (b-c)^2[f(b)-f(a)]}{(b-a)[f(b)-f(c)] - (b-c)[f(b)-f(a)]}$$

Notice, however, that this formula is happy jumping to a parabolic maximum as to a minimum. To rely exclusively on the formula is not good practice.

In practice one wishes a scheme that operates on something guaranteed to succeed, like the golden section method, combined with a parabolic approximation when one is sure that the function does not present problems. The devil as usual is in the details, there is complicated bookkeeping as well as trying to minimize the number of times one evaluates the function.

Brent's method implements these ideas in practice. Let us comment on some of its aspects. It is based on keeping track of six function points, a, b, u, v, w, and x, defined as follows: the minimum is bracketed between a and b; x is the point with the very least function value found so far (or the most recent one in the case of a tie); w is the point with the second least function value; v is the previous value of w;u is the point at which the function was evaluated most recently. Also appearing in the algorithm is xm, the midpoint of a and b.

Parabolic interpolation is attempted and evaluated with x, v, w. It is considered good if it a) falls in the interval [a,b] and b) it implies a movement from the best current x that is less than half the movement of the step before it (avoid bouncing around). If the parabolic step is not acceptable, then perform a golden section try. The method also avoids evaluating the function at a point at a distance less than TOL of a previous evaluation (as we discussed before) (TOL<sqrt of the machine precision).

```
      FUNCTION brent(ax,bx,cx,f,tol,xmin)
      INTEGER ITMAX
      REAL brent,ax,bx,cx,tol,xmin,f,CGOLD,ZEPS
      EXTERNAL f
      PARAMETER (ITMAX=100,CGOLD=.3819660,ZEPS=1.0e-10)
         Given a function f, and given a bracketing triplet of abscissas ax, bx, cx (such that bx is
         between ax and cx, and f(bx) is less than both f(ax) and f(cx)), this routine isolates
         the minimum to a fractional precision of about tol using Brent's method. The abscissa of
         the minimum is returned as xmin, and the minimum function value is returned as brent,
         the returned function value.
         Parameters: Maximum allowed number of iterations; golden ratio; and a small number that
         protects against trying to achieve fractional accuracy for a minimum that happens to be
         exactly zero.
      INTEGER iter
      REAL a,b,d,e,etemp,fu,fv,fw,fx,p,q,r,tol1,tol2,u,v,w,x,xm
      a=min(ax,cx)                      a and b must be in ascending order, though the input
      b=max(ax,cx)                          abscissas need not be.
      v=bx                              Initializations...
      w=v
      x=v
      e=0.                              This will be the distance moved on the step before last.
      fx=f(x)
      fv=fx
      fw=fx
      do 11 iter=1,ITMAX                Main program loop.
        xm=0.5*(a+b)
        tol1=tol*abs(x)+ZEPS
        tol2=2.*tol1
        if(abs(x-xm).le.(tol2-.5*(b-a))) goto 3        Test for done here.
        if(abs(e).gt.tol1) then                        Construct a trial parabolic fit.
          r=(x-w)*(fx-fv)
          q=(x-v)*(fx-fw)
          p=(x-v)*q-(x-w)*r
          q=2.*(q-r)
          if(q.gt.0.) p=-p
          q=abs(q)
          etemp=e
          e=d

          if(abs(p).ge.abs(.5*q*etemp).or.p.le.q*(a-x).or.
     *          p.ge.q*(b-x)) goto 1
            The above conditions determine the acceptability of the parabolic fit. Here it is o.k.:
          d=p/q                         Take the parabolic step.
          u=x+d
          if(u-a.lt.tol2 .or. b-u.lt.tol2) d=sign(tol1,xm-x)
          goto 2                        Skip over the golden section step.
        endif
        d=CGOLD*e                         Take the golden section step.
2       if(abs(d).ge.tol1) then           Arrive here with d computed either from parabolic fit, or
            u=x+d                             else from golden section.
        else
            u=x+sign(tol1,d)
        endif
        fu=f(u)                           This is the one function evaluation per iteration,
        if(fu.le.fx) then                 and now we have to decide what to do with our function
            if(u.ge.x) then                   evaluation. Housekeeping follows:
                a=x
            else
                b=x
            endif
            v=w
            fv=fw
            w=x
            fw=fx
            x=u
            fx=fu
        else
            if(u.lt.x) then
                a=u
            else
                b=u
            endif
            if(fu.le.fw .or. w.eq.x) then
                v=w
                fv=fw
                w=u
                fw=fu
            else if(fu.le.fv .or. v.eq.x .or. v.eq.w) then
                v=u
                fv=fu
            endif
        endif                             Done with housekeeping. Back for another iteration.
    enddo
    pause 'brent exceed maximum iterations'
3   xmin=x                                Arrive here ready to exit with best values.
    brent=fx
    return
    END
```

One dimensional search with first derivatives

We will now attempt the same as before, to isolate a functional minimum bracketed by (a,b,c) but using knowledge of the derivatives of the function.

In principle one could think that one could simply use a routine to search for the zero of the derivative. That is not a good idea. To begin with it cannot distinguish maxima from minima. Moreover, how to proceed when the derivative at the endpoints points to a zero outside the bracket?

We don't want to give up on the strategy of maintaining the minimum bracketed. Therefore the only way to proceed is to use a technique that uses function (not derivative) information. The derivative information can only be used to help find trial points within the bracket.

Some propose fitting a high order polynomial using all the information one has. But that gives up on bracketing.

A more conservative strategy is to use the derivative in the midpoint b to decide that the minimum is in [a,b] or [b,c]. The secant is extrapolated to zero and we impose the same conditions on this guess as in Brent's method. You can see the modified Brent routine in the book.

Downhill simplex method in multidimensions

This method is due to Nelder and Mead and uses only function evaluations, not derivatives. It is not the most economical. But it is the method of choice if one wishes something quickly and the function is not too costly to evaluate. It has a natural geometric interpretation that makes it easy to picture.

A simplex is the geometrical figure, in N dimensions, of N+1 points (or vertices) and all their interconnecting line segments. In two dimensions, it is a triangle. In three dimensions a tetrahedron. In general we are interested in simplexes that are non degenerate, that is, have non-zero volume.

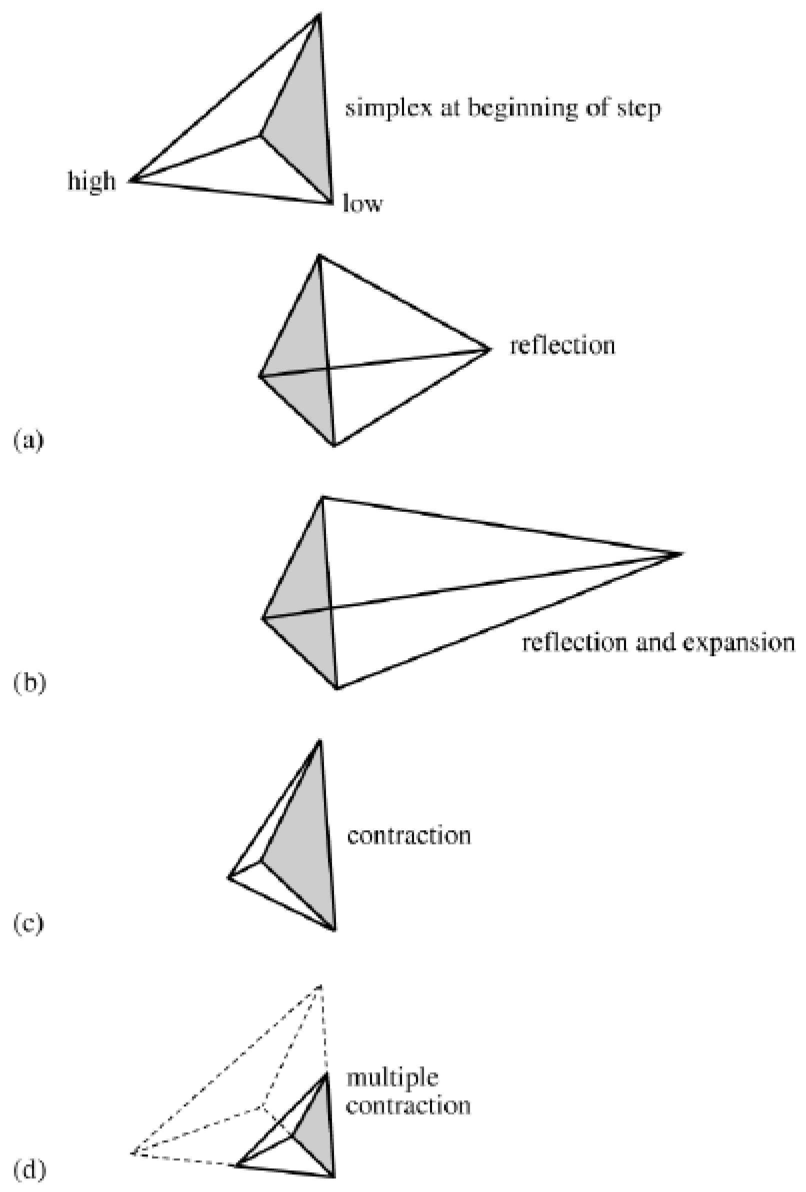
In higher dimensions it is not practical to bracket. The best we can do is to give the algorithm an initial N vector of independent variables as first try. The algorithm will then have to find its way down through the unimaginable complexity of an N Dimensional topography until it encounters a (local) minimum.

The simplex method is started with N+1 points. If you take one of these initial points (it doesn't matter which) as your initial point, then the other N points are given by,

$$\mathbf{P}_i = \mathbf{P}_0 + \lambda \mathbf{e}_i$$

with e_i unit vectors.

The method takes a series of steps, moving the point of the simplex where the function is the largest through the opposite face of the simplex to a lower point ("reflections"). The volume of the simplex is preserved to preserve its non-degeneracy. When it reaches a valley floor, the method contracts itself in the transverse direction and tries to flow down the valley. If there is a situation where the simplex is trying to pass through "the eye of a needle" it contracts itself in all directions. For this reason the routine that implements it is called "amoeba"



```
      FUNCTION amotry(p,y,psum,mp,np,ndim,funk,ihi,fac)
      INTEGER ihi,mp,ndim,np,NMAX
      REAL amotry,fac,p(mp,np),psum(np),y(mp),funk
      PARAMETER (NMAX=20)
      EXTERNAL funk
C     USES funk
         Extrapolates by a factor fac through the face of the simplex across from the high point,
         tries it, and replaces the high point if the new point is better.
      INTEGER j
      REAL fac1,fac2,ytry,ptry(NMAX)
      fac1=(1.-fac)/ndim
      fac2=fac1-fac
      do 11 j=1,ndim
        ptry(j)=psum(j)*fac1-p(ihi,j)*fac2
      enddo 11
      SUBROUTINE amoeba(p,y,mp,np,ndim,ftol,funk,iter)
      INTEGER iter,mp,ndim,np,NMAX,ITMAX
      REAL ftol,p(mp,np),y(mp),funk,TINY
      PARAMETER (NMAX=20,ITMAX=5000,TINY=1.e-10)   Maximum allowed dimensions and func-
      EXTERNAL funk                                   tion evaluations, and a small num-
C     USES amotry, funk                               ber.
```

Multidimensional minimization of the function `funk(x)` where `x(1:ndim)` is a vector in `ndim` dimensions, by the downhill simplex method of Nelder and Mead. The matrix `p(1:ndim+1,1:ndim)` is input. Its `ndim+1` rows are `ndim`-dimensional vectors which are the vertices of the starting simplex. Also input is the vector `y(1:ndim+1)`, whose components must be pre-initialized to the values of `funk` evaluated at the `ndim+1` vertices (rows) of `p`; and `ftol` the fractional convergence tolerance to be achieved in the function value (n.b.!). On output, `p` and `y` will have been reset to `ndim+1` new points all within `ftol` of a minimum function value, and `iter` gives the number of function evaluations taken.

```
      INTEGER i,ihi,ilo,inhi,j,m,n
      REAL rtol,sum,swap,ysave,ytry,psum(NMAX),amotry
      iter=0
1     do 12 n=1,ndim                  Enter here when starting or have just overall contracted.
        sum=0.                        Recompute psum.
        do 11 m=1,ndim+1
          sum=sum+p(m,n)
        enddo 11
        psum(n)=sum
      enddo 12
2     ilo=1                           Enter here when have just changed a single point.
      if (y(1).gt.y(2)) then          Determine which point is the highest (worst), next-highest,
        ihi=1                           and lowest (best),
        inhi=2
      else
        ihi=2
        inhi=1
      endif
      do 13 i=1,ndim+1                by looping over the points in the simplex.
        if(y(i).le.y(ilo)) ilo=i
        if(y(i).gt.y(ihi)) then
          inhi=ihi
          ihi=i
        else if(y(i).gt.y(inhi)) then
          if(i.ne.ihi) inhi=i
```

```
    endif
enddo 13
rtol=2.*abs(y(ihi)-y(ilo))/(abs(y(ihi))+abs(y(ilo))+TINY)
  Compute the fractional range from highest to lowest and return if satisfactory.
if (rtol.lt.ftol) then        If returning, put best point and value in slot 1.
    swap=y(1)
    y(1)=y(ilo)
    y(ilo)=swap
    do 14 n=1,ndim
        swap=p(1,n)
        p(1,n)=p(ilo,n)
        p(ilo,n)=swap
    enddo 14
    return
endif
if (iter.ge.ITMAX) pause 'ITMAX exceeded in amoeba'
iter=iter+2
  Begin a new iteration. First extrapolate by a factor −1 through the face of the simplex across
  from the high point, i.e., reflect the simplex from the high point.
ytry=amotry(p,y,psum,mp,np,ndim,funk,ihi,-1.0)
if (ytry.le.y(ilo)) then
  Gives a result better than the best point, so try an additional extrapolation by a factor 2.
    ytry=amotry(p,y,psum,mp,np,ndim,funk,ihi,2.0)
else if (ytry.ge.y(inhi)) then
```

The reflected point is worse than the second-highest, so look for an intermediate lower point, i.e., do a one-dimensional contraction.

```
    ysave=y(ihi)
    ytry=amotry(p,y,psum,mp,np,ndim,funk,ihi,0.5)
    if (ytry.ge.ysave) then      Can't seem to get rid of that high point. Better contract
        do 16 i=1,ndim+1              around the lowest (best) point.
            if(i.ne.ilo)then
                do 15 j=1,ndim
                    psum(j)=0.5*(p(i,j)+p(ilo,j))
                    p(i,j)=psum(j)
                enddo 15
                y(i)=funk(psum)
            endif
        enddo 16
        iter=iter+ndim           Keep track of function evaluations.
        goto 1                   Go back for the test of doneness and the next iteration.
    endif
else
    iter=iter-1                  Correct the evaluation count.
endif
goto 2
END
```

Summary

- The golden section is akin to bisection and “assumes the worse”.
- Brent’s method assumes parabolic behavior but corrects the most egregious problems of such assumption.
- The simplex method is like “an amoeba crawling through the function”.