Lecture 35
Minimization and maximization of functions

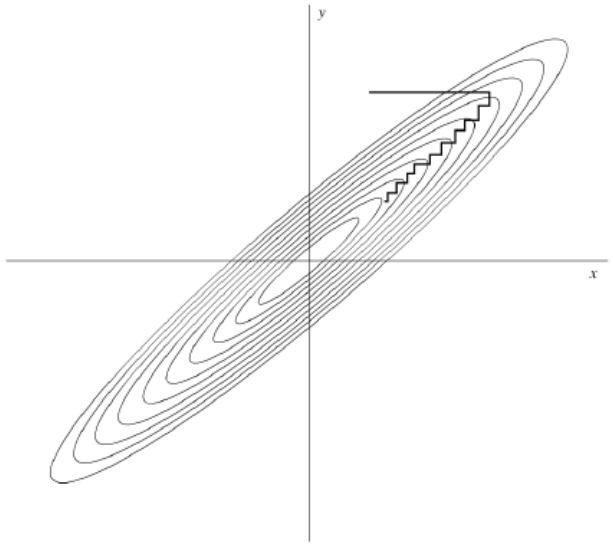
- Powell's method in multidimensions
- Conjugate gradient method.
- Annealing methods.

We know how to minimize functions in one dimension. If we start at a point P in and N dimensional space and proceed from there along some vector direction n, then any function of N variables f(P) can be minimized along the direction n using the one dimensional methods. One can construct in this way several minimization schemes in multidimensions, whose main difference will be how they choose the direction n. We will discuss a few.

In all our discussions we will assume we have a black box sub-algorithm, which we will call `linmin` whose definition can be taken as:

> `linmin`: Given as input the vectors $\mathbf{P}$ and $\mathbf{n}$, and the function f, find the scalar λ that minimizes $f(\mathbf{P}+\lambda\mathbf{n})$. Replace $\mathbf{P}$ by $\mathbf{P}+\lambda\mathbf{n}$. Replace $\mathbf{n}$ by $\lambda\mathbf{n}$. Done.

The first possibility is to take a basis of directions e_1, e_2,…,e_N in your space, and successively minimize along each basis element, and repeat when you used them all until you eventually reach the minimum. For many functions this method is not bad.



But in some cases it does not work so well. It is the case of functions that have narrow valleys that do not align with the basis vectors. That will force the algorithm to take many tiny steps before it finds the minimum. This condition is not that unusual, especially in higher dimensions.

To deal with this, we obviously need better guesses for the directions.

The better guess can come in two different fashions: a) The chosen direction will take us along a narrow valley; b) The directions chosen are “non interfering”, that is, minimizing along one of them will not be spoiled by then minimizing along another.

Conjugate directions

First notice that if we minimize a function along a given direction, the gradient vector will be perpendicular to that direction (otherwise there would be a non-zero directional derivative along the direction you supposedly minimized). Next, take some particular point P as the origin of the coordinate system with coordinates x. Any function can be approximated by its Taylor series.

$$f(\mathbf{x}) = f(\mathbf{P}) + \sum_i \frac{\partial f}{\partial x_i} x_i + \frac{1}{2} \sum_{i,j} \frac{\partial^2 f}{\partial x_i \partial x_j} x_i x_j + \cdots$$

$$\approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x}$$

where

$$c \equiv f(\mathbf{P}) \qquad \mathbf{b} \equiv -\nabla f|_{\mathbf{P}} \qquad [\mathbf{A}]_{ij} \equiv \left. \frac{\partial^2 f}{\partial x_i \partial x_j} \right|_{\mathbf{P}}$$

In this approximation the gradient is given by: $\nabla f = \mathbf{A} \cdot \mathbf{x} - \mathbf{b}$

And the change of the gradient as we move along some direction is:

$$\delta(\nabla f) = \mathbf{A} \cdot (\delta \mathbf{x})$$

Now, suppose you have moved along some direction **u** to a minimum and now propose to move along a new direction **v**. The condition that motion along **v** not spoil our minimization along u is just that the gradient stay perpendicular to **u**, i.e. that the change of the gradient be perpendicular to **u**. Given the equation we just discussed, this implies that,

$$0 = \mathbf{u} \cdot \delta(\nabla f) = \mathbf{u} \cdot \mathbf{A} \cdot \mathbf{v}$$

When an equation of this sort holds for two vectors **u**,**v**, they are said to be *conjugate*. If you minimize along conjugate directions, then you do not have to re-do the directions.

An ideal situation is to come up with N conjugate directions. Then one pass on each will do the job. If the function were exactly quadratic, it will put you exactly at the minimum. For more general functions it will come close, and converge quadratically to the minimum in terms of the number of steps.

Powell's quadratically convergent method

Powell discovered a direction set that produces N mutually conjugate directions.

Initialize your set of directions $\mathbf{u}_i$ to the basis vectors, $\mathbf{u}_i=\mathbf{e}_i$ i=1,…,N

Now repeat the following sequence of steps,

- Save your starting position as $\mathbf{P}_0$.
- For $i = 1, \ldots, N$, move $\mathbf{P}_{i-1}$ to the minimum along direction $\mathbf{u}_i$ and call this point $\mathbf{P}_i$.
- For $i = 1, \ldots, N-1$, set $\mathbf{u}_i \leftarrow \mathbf{u}_{i+1}$.
- Set $\mathbf{u}_N \leftarrow \mathbf{P}_N - \mathbf{P}_0$.
- Move $\mathbf{P}_N$ to the minimum along direction $\mathbf{u}_N$ and call this point $\mathbf{P}_0$.

Powell showed that, for a quadratic function, k iterations of this procedure produce a set of directions $\mathbf{u}_i$ whose last k members are mutually conjugate. Therefore, N iterations, amounting to N(N+1) line minimizations in all will exactly minimize a quadratic functions.

Sketch of proof (Brent, “Algorithms for minimizations without derivatives”).

Theorem: Given $f(\mathbf{x})=\mathbf{x}^T A\ \mathbf{x}-2\ \mathbf{b}^T\ \mathbf{x}+c$ if its minimum along the direction u from $\mathbf{x}^*_i$ is at $\mathbf{x}_i$ for i=0,1, then x_1-x_0 is conjugate to **u.**

Proof:

For i=0,1, $$\frac{\partial}{\partial\lambda} f(x_i+\lambda u)=0$$

Particularizing for f(x), $$u^T(Ax_i-b)=0$$

Subtracting for i=0,1, $u^T A(x_1-x_0)=0$, next to last step in Powell's proposal.

Unfortunately there is a problem with Powell's algorithm. The procedure of throwing away, at each stage, $\mathbf{u}_1$ in favor of $\mathbf{P}_N$-$\mathbf{P}_0$ tends to produce sets of directions that "fold up on each other" and become linearly dependent. Once this happens, the procedure finds the minimum in a subspace of the N dimensional space. That is, it produces the wrong answer.

There are several ways of fixing this:

1. You reinitialize the set of directions back to the $\mathbf{e}_i$'s after N or N+1 iterations of the basic procedure.

2. The set of directions can equally be reset to the columns of any orthogonal matrix. Rather than throw away the information on conjugate directions already built up, reset the direction to calculated principal directions of the matrix A.

3. You can give up quadratic convergence in favor of a more heuristic scheme, which tries to find a few good directions along narrow valleys instead of N necessarily conjugate directions. This is the method that Numerical Recipes implements.

Shall we be so quick to abandon quadratic convergence? That depends on the function. Some problems produce functions with long, twisty valleys. A quadratic method tries to extrapolate the minimum along the long direction with a parabola that is not there yet whereas the twists spoil the conjugacy of the N-1 transverse directions.

The basic idea of the modified Powell method is still to take $\mathbf{P}_N$-$\mathbf{P}_0$ as a new direction; it is, after all, the average direction moved after trying all N possible directions. For a valley whose long direction is twisting slowly, this direction is likely to give a good run along the long direction. The change is to discard the old direction along which the function f made its largest decrease. This seems surprising, since that direction was the best direction of the previous iteration. However, it has a big chance of being a major component of the new direction that we are adding so by dropping we avoid building up linear dependence.

There are a couple of exceptions to this basic idea. Sometime it is better not to add a new direction at all. Define

$$f_0 \equiv f(\mathbf{P}_0) \qquad f_N \equiv f(\mathbf{P}_N) \qquad f_E \equiv f(2\mathbf{P}_N - \mathbf{P}_0)$$

With f_E the value of the function at an "extrapolated" point further along the proposed new direction. Also define Δf the magnitude of the maximum decrease along one particular direction.

Then:

1. If $f_E \geq f_0$, then keep the old set of directions for the next basic procedure, because the average direction $\mathbf{P}_N - \mathbf{P}_0$ is all played out.

2. If $2\,(f_0 - 2f_N + f_E)\,[(f_0 - f_N) - \Delta f]^2 \geq (f_0 - f_E)^2 \Delta f$, then keep the old set of directions for the next basic procedure, because either (i) the decrease along the average direction was not primarily due to any single direction's decrease, or (ii) there is a substantial second derivative along the average direction and we seem to be near to the bottom of its minimum.

The following routine implements Powell's method in the version just described. In the routine, `xi` is the matrix whose columns are the set of directions $\mathbf{n}_i$; otherwise the correspondence of notation should be self-evident.

```
      SUBROUTINE powell(p,xi,n,np,ftol,iter,fret)
      INTEGER iter,n,np,NMAX,ITMAX
      REAL fret,ftol,p(np),xi(np,np),func,TINY
      EXTERNAL func
      PARAMETER (NMAX=20,ITMAX=200,TINY=1.e-25)
C     USES func,linmin
```

Minimization of a function `func` of `n` variables. (`func` is not an argument, it is a fixed function name.) Input consists of an initial starting point `p(1:n)`; an initial matrix `xi(1:n,1:n)` with physical dimensions `np` by `np`, and whose columns contain the initial set of directions (usually the `n` unit vectors); and `ftol`, the fractional tolerance in the function value such that failure to decrease by more than this amount on one iteration signals doneness. On output, `p` is set to the best point found, `xi` is the then-current direction set, `fret` is the returned function value at `p`, and `iter` is the number of iterations taken. The routine `linmin` is used.

Parameters: Maximum value of `n`, maximum allowed iterations, and a small number.

```
      INTEGER i,ibig,j
      REAL del,fp,fptt,t,pt(NMAX),ptt(NMAX),xit(NMAX)
      fret=func(p)
      do 11 j=1,n                          Save the initial point.
        pt(j)=p(j)
      enddo 11
      iter=0
1     iter=iter+1
      fp=fret
      ibig=0
      del=0.                               Will be the biggest function decrease.
      do 13 i=1,n                          In each iteration, loop over all directions in the set.
        do 12 j=1,n                        Copy the direction,
          xit(j)=xi(j,i)
        enddo 12
```

```
        fptt=fret
        call linmin(p,xit,n,fret)       minimize along it,
        if(fptt-fret.gt.del)then        and record it if it is the largest decrease so far.
            del=fptt-fret
            ibig=i
        endif
    enddo 13
    if(2.*(fp-fret).le.ftol*(abs(fp)+abs(fret))+TINY)return     Termination criterion.
    if(iter.eq.ITMAX) pause 'powell exceeding maximum iterations'
    do 14 j=1,n                         Construct the extrapolated point and the average di-
        ptt(j)=2.*p(j)-pt(j)                rection moved. Save the old starting point.
        xit(j)=p(j)-pt(j)
        pt(j)=p(j)
    enddo 14
    fptt=func(ptt)                      Function value at extrapolated point.
    if(fptt.ge.fp)goto 1                One reason not to use new direction.
    t=2.*(fp-2.*fret+fptt)*(fp-fret-del)**2-del*(fp-fptt)**2
    if(t.ge.0.)goto 1                   Other reason not to use new direction.
    call linmin(p,xit,n,fret)           Move to the minimum of the new direction,
    do 15 j=1,n                         and save the new direction.
        xi(j,ibig)=xi(j,n)
        xi(j,n)=xit(j)
    enddo 15
    goto 1                              Back for another iteration.
    END
```

Conjugate gradient method in multidimensions

We now consider the case where you are able to calculate, at a given N-dimensional point **P**, not just the value of the function f(**P**) but also the gradient $\nabla f(\mathbf{P})$

Let us assume that the function can be approximated by a quadratic form, as before,

$$f(\mathbf{x}) \approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2} \mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x}$$

Then the number of unknown parameters in f is equal to the number of free parameters in **A** and **b**, which is N(N+1)/2, so it is of order N^2. Changing any of these parameters will move the location of the minimum. Therefore we should not expect to be able to find the minimum until we have collected an equivalent information content, of the order of N^2 numbers.

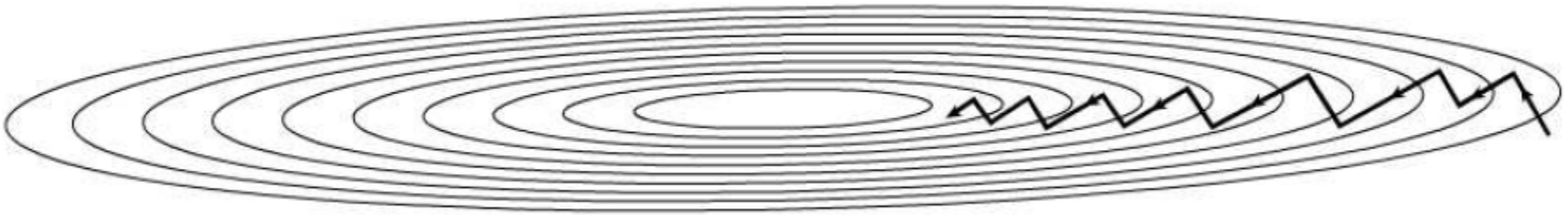
In the direction set methods we talked about one collected the information by making of the order of N^2 minimizations. Here we can expect to do less, since we will be using information about the gradient. It is not clear that computationally one gains much, since computing the gradient requires N operations.

It also matters how one uses the information of the gradient.

For instance a not too good use of the gradient information leads to the steepest descent method:

> Steepest Descent: Start at a point $\mathbf{P}_0$. As many times as needed, move from point $\mathbf{P}_i$ to the point $\mathbf{P}_{i+1}$ by minimizing along the line from $\mathbf{P}_i$ in the direction of the local downhill gradient $-\nabla f(\mathbf{P}_i)$.

The problem with this method (already discussed by Cauchy) is similar to the one we encountered before with narrow valleys. The method will perform many steps descending along the narrow valley, even if it is perfectly quadratic and one could cover it in one step.



As you see at each step it is doing its job, but overall it is not doing so well.

What we want is not to go down the new gradient, but rather a direction that is conjugate to the old gradient. Such methods are called conjugate gradient methods.

We discussed conjugate gradient methods in the context of solving linear algebraic equations by minimizing a quadratic form. That formalism can also be applied to the problem of minimizing a function approximated by a quadratic form.

Starting from an initial vector g_0, and letting $h_0=g_0$, the conjugate gradient method constructs two sequences of vectors through the recurrence.

$$\mathbf{g}_{i+1} = \mathbf{g}_i - \lambda_i \mathbf{A} \cdot \mathbf{h}_i \qquad \mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i \qquad i = 0, 1, 2, \ldots$$

With

$$\lambda_i = \frac{\mathbf{g}_i \cdot \mathbf{g}_i}{\mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_i} = \frac{\mathbf{g}_i \cdot \mathbf{h}_i}{\mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_i}$$

$$\gamma_i = \frac{\mathbf{g}_{i+1} \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i}$$

The vectors satisfy the orthogonality and conjugacy conditions,

$$\mathbf{g}_i \cdot \mathbf{g}_j = 0 \qquad \mathbf{h}_i \cdot \mathbf{A} \cdot \mathbf{h}_j = 0 \qquad \mathbf{g}_i \cdot \mathbf{h}_j = 0 \qquad j < i$$

So if we knew A, this procedure would provide successively conjugate directions along which to minimize.

But we don't know A…

Here's a remarkable theorem to save the day: suppose that we set g_i equal to the gradient at $\mathbf{P}_i$. We now proceed from $\mathbf{P}_i$ along the direction $\mathbf{h}_i$ to the local minimum of f located at some point $\mathbf{P}_{i+1}$. We then set $\mathbf{g}_{i+1}$ equal to the gradient at that point. Then this $\mathbf{g}_{i+1}$ is the same we would have constructed with the construction we outlined in the previous slide, except that we did it without knowledge of A.

Proof: Given the gradient of a quadratic function, we have that, $\mathbf{g}_i = -\mathbf{A} \cdot \mathbf{P}_i + \mathbf{b},$

And: $\mathbf{g}_{i+1} = -\mathbf{A} \cdot (\mathbf{P}_i + \lambda \mathbf{h}_i) + \mathbf{b} = \mathbf{g}_i - \lambda \mathbf{A} \cdot \mathbf{h}_i$

with l chosen to take us to the line minimum. But there $\mathbf{h}_i \cdot \nabla f = -\mathbf{h}_i \cdot \mathbf{g}_{i+1} = 0.$

So combining with the above equation we get the expression for λ on the previous slide.

The above proposal is due to Fletcher and Reeves.
Polak and Ribiere proposed a slightly different version:

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

Which would be completely equivalent for quadratic functions given the orthogonality condition. It seems to work better.

```
SUBROUTINE frprmn(p,n,ftol,iter,fret)
INTEGER iter,n,NMAX,ITMAX
REAL fret,ftol,p(n),EPS,func
EXTERNAL func
PARAMETER (NMAX=50,ITMAX=200,EPS=1.e-10)
```
USES dfunc,func,linmin

Given a starting point p that is a vector of length n, Fletcher-Reeves-Polak-Ribiere minimization is performed on a function func, using its gradient as calculated by a routine dfunc. The convergence tolerance on the function value is input as ftol. Returned quantities are p (the location of the minimum), iter (the number of iterations that were performed), and fret (the minimum value of the function). The routine linmin is called to perform line minimizations.

Parameters: NMAX is the maximum anticipated value of n; ITMAX is the maximum allowed number of iterations; EPS is a small number to rectify special case of converging to exactly zero function value.

```
INTEGER its,j
REAL dgg,fp,gam,gg,g(NMAX),h(NMAX),xi(NMAX)
fp=func(p)                              Initializations.
call dfunc(p,xi)
do 11 j=1,n
    g(j)=-xi(j)
    h(j)=g(j)
do 12 j=1,ncom
    df1dim=df1dim+df(j)*xicom(j)
enddo 12
return
END
```

Simulated annealing methods

These methods are especially suited for situations where the dimensionality of space is large and there is a global minimum hidden among many false local minima.

The idea is to draw an analogy with thermodynamics, specifically with the way liquids freeze and crystallize or metals cool and "anneal". The idea is that at high temperature the molecules of a liquid move freely with respect to one another. If one slowly lowers the temperature eventually a solid with a crystalline structure develops and this configuration is the lowest energy configuration for the system. It is amazing that nature is able to find this configuration provided one "cooks slowly". If one cools quickly ("quenching") then one ends up with an amorphous or policrystalline state that is not the minimum of energy.

Nature's own minimization algorithm is based on the Boltzmann probability distribution.

$$\text{Prob}\,(E) \sim \exp(-E/kT)$$

Metropolis et al. decided to use this distribution to solve minimization problems.

The idea is that given two configurations E_2 and E_1 a simulated thermodynamic system changes its configuration from energy E_1 to E_2 with probability

$$p = \exp[-(E_2 - E_1)/kT]$$

Notice that this number can be bigger than one. In that case we set it to one and the system actually transitions for sure. Otherwise it transitions with the respective probability. Just like in the Metropolis algorithm one takes a trial step and decides to keep it or not based on the probability.

An example where this method has been used to in practice solve a problem is the traveling salesperson problem. Given a series of cities characterized by coordinates (x1,x2) in a map, which is the trajectory that minimizes the total traveled distance. In that example th

$$E = L \equiv \sum_{i=1}^{N} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}$$

This problem is what is known as an “NP-complete” problem, which means that that the computation time for the exact solution goes as exp(N) with N the number of cities. A closely related problem is that of the design of integrated circuits, in which one wishes to minimize the interference among connecting wires.

Summary

- Powell's method allows to construct conjugate directions easily. One needs to "reset" it every once in a while to prevent "folding".
- Conjugate gradient is an alternative way of finding conjugate directions.
- The annealing method uses thermodynamical analogies and is statistical in nature, akin to the Monte Carlo method.