1.3 Numerical Solution with Gradient Methods

Unless the relations for $L(y)$ and $f(y)$ in Section 1.2 are quite simple, numerical methods must be used to determine the parameter vector y that minimizes $L(y)$ with $f(y) = 0$.

POP—A Gradient Algorithm for Parameter OPtimization with Equality Constraints

A straightforward numerical method is to guess an initial y and then take small steps Δy in the direction of $-H_y^T$ = the direction of *steepest descent*. This will lead to a local minimum if one exists. If there are several local minima, this method may not find the global minimum. Choosing the step size requires some experience. If the steps are too large, the minimum may be overshot; if the steps are too small, it takes many steps to reach the minimum.

We first state the algorithm and then give a brief derivation.

Enter data

- Guess y.
- Choose $k > 0$ for a minimum, $k < 0$ for a maximum.
- Choose η so that $|\Delta y|$ is not too large, $0 < \eta \leq 1$.
- Choose stopping tolerance tol.

Find L, f, L_y, f_y, f_y^*, and H_y

$(*) L = L(y),\ f = f(y),\ L_y = L_y(y),\ f_y = f_y(y)$.

- $f_y^* = f_y^T (f_y f_y^T)^{-1}$.
- $\lambda^T = -L_y f_y^*$.
- $H_y = L_y + \lambda^T f_y$.

Find improved value of y

- $dy = -\eta f_y^* f - k H_y^T$.
- If $\max(|f|,\ |dy|/\sqrt{p}) < tol$, then end.
- Replace y by $y + dy$ and go to $(*)$.

This algorithm may be interpreted as finding a small change Δy, to minimize a linear approximation to ΔL subject to a quadratic penalty on Δy and a linear approximation to coming closer to satisfying the constraints $f(y) = 0$:

$$\min_{\Delta y} \Delta J = L_y \Delta y + \frac{\Delta y^T \Delta y}{2k} , \tag{1.35}$$

subject to

$$f_y \Delta y = -\eta f , \tag{1.36}$$

where L_y, f, f_y are evaluated at the current estimate of y. *y should be normalized so that a change of one unit of each element of y is approximately of equal significance.* $\eta = 1$ asks for the constraints to be satisfied in one step, so $\eta < 1$ asks only to move part way toward satisfying the constraints. k is a scalar step-size parameter; we shall say more about choosing k below.

To solve this linear-quadratic problem, we adjoin the constraint (1.36) to the performance index with a Lagrange multiplier vector $\bar{\lambda}$:

$$\Delta J \triangleq L_y \Delta y + \frac{\Delta y^T \Delta y}{2k} + \bar{\lambda}^T (\eta f + f_y \Delta y) , \tag{1.37}$$

Assuming that L_y, f, f_y are fixed matrices, take a differential of ΔJ. For $d(\Delta J) = 0$ with arbitrary $d(\Delta y)$, it is clearly necessary that

$$L_y + \bar{\lambda}^T f_y + \frac{\Delta y^T}{k} = 0 , \tag{1.38}$$

from which

$$\Delta y^T = -k(L_y + \bar{\lambda}^T f_y) . \tag{1.39}$$

Substituting (1.39) into (1.36), we may solve for $\bar{\lambda}^T$:

$$\bar{\lambda}^T = \lambda^T + \frac{\eta}{k} f^T Q^{-1} . \tag{1.40}$$

where

$$Q = f_y f_y^T \ , \quad \lambda^T = -L_y f_y^* \ , \quad f_y^* \triangleq f_y^T Q^{-1} \ . \tag{1.41}$$

f_y^* is the weighted *generalized inverse* of f_y.

Finally, substituting (1.40) and (1.41) into (1.39) gives

$$\Delta y = -\eta f_y^* f - k H_y^T \ , \tag{1.42}$$

where

$$H_y \triangleq L_y + \lambda^T f_y \ . \tag{1.43}$$

When starting with a possibly poor guess, one can put $k = 0$ (i.e., ask for no improvement in L) and pick $\eta < 1$ to limit the size of $|dy|$ so that the linear approximation is reasonable. Over a few steps, gradually increase η to 1, and a feasible solution ($f \approx 0$) should be obtained, after which $\eta = 1$ can be used with $k \neq 0$ to minimize L.

k must be chosen by interpolation so that L decreases at a reasonable rate with each iteration. If $|k|$ is too large, each step will "overshoot" the minimum and L will actually increase; if $|k|$ is too small, many iterations will be required to reach the vicinity of the minimum.

Other gradient algorithms are available that do not require the user to interpolate k, and that converge more rapidly near the minimum (e.g., Refs. Gr, GMSW, and Ln). They use the successive values of H_y to approximate the Hessian H_{yy}; they are called *quasi-Newton* algorithms since they tend toward Newton-Raphson (NR) algorithms as the number of iterations increases. NR algorithms are second-order gradient algorithms (covered in the next section). These algorithms are necessarily more complicated than POP; obviously there is a trade-off between algorithm complexity and speed of convergence.

Table 1.1 lists a MATLAB function file that implements the POP algorithm.

TABLE 1.1 A MATLAB Code for POP

```
function [L,y,f]=pop(name,y,k,tol,eta,mxit)
% Parameter OPtimization using a generalized gradient algorithm; outputs L
% and y (p by 1) are optimum performance index & parameter vector; con-
% straints are f=0 where f is (n by 1) with n < p; user must supply a sub-
% routine 'name' that computes L,f,Ly,fy; input y is a guess; y should be
% normalized so that a change of one unit in each element of y has roughly
% the same significance; k is a scalar step size parameter; stopping cri-
% terion is max(fn,dyn)<tol, where fn=norm(f); dyn=norm(dy)/sqrt(p); eta=
% fraction of constraint violation to be removed; mxit=max no. iterations.
%
it=0; dyn=1; fn=1; p=length(y);
disp('     it        L        fn          dyn')
while max(fn,dyn) > tol
  [L,f,Ly,fy]=feval(name,y); fn=norm(f); fyi=fy'/(fy*fy'); lat=-Ly*fyi;
  Hy=Ly+lat*fy; dy=-eta*fyi*f-k*Hy'; dyn=norm(dy)/sqrt(p);
  disp([it L fn dyn]); y=y+dy; if it> mxit, break, end; it=it+1;
end
```

Example 1.3.1—Max Velocity of a Sailboat

This is a numerical solution of Problem 1.2.7. The subroutine containing L, f, L_y, f_y) is listed below.

```
function [L,f,Ly,fy]=slbt(y)
% Sailboat max velocity; L = V; y = [V,Wr,al,th,ps]',
%
V=y(1); Wr=y(2); al=y(3); th=y(4); ps=y(5); sa=sin(al); ca=cos(al);
st=sin(th); ct=cos(th); sp=sin(ps); cp=cos(ps); sat=sin(al+th);
cat=cos(al+th); L=V; Ly=[1 0 0 0 0];
f=[V^2-Wr^2*sa*st; Wr^2-V^2-1+2*V*cp; Wr*sat-sp];
fy=[2*V -2*Wr*sa*st -Wr^2*ca*st -Wr^2*sa*ct 0; -2*(V-cp) 2*Wr 0 0
    -2*V*sp; ... 0 sat Wr*cat Wr*cat -cp];
```

An edited diary is listed below that solves the problem using POP. It took 28 iterations to bring the norms of f and dy below .00005.

```
% Script e01_3_1.m for gradient solution of Ex. 1.3.1;
%
y=[.5 1 .5 .5 1.5]'; k=-1.2; tol=1e-5; eta=1; mxit=50;
[L,y,f]=pop('slbt',y,k,tol,eta,mxit);
  it       L          fn         dyn
   0     0.5000     0.2385     0.0733
   1     0.5793     0.0149     0.0263
   -        -          -          -
  27     0.5931     0.0000     0.0000
  28     0.5931     0.0000     0.0000
y'=[.5931 1.0057 .6307 .6307 1.2798]; f=1e-9*[.1183 .0627
    .0065]'
```

1.5 Numerical Solution with Newton-Raphson Methods

First-order gradient algorithms like those in Section 1.3, approximate the L and f functions locally with osculating hyperplanes. Newton-Raphson (NR) algorithms approximate the L and f functions locally with osculating *quadric surfaces*, i.e., they use the Hessian as well

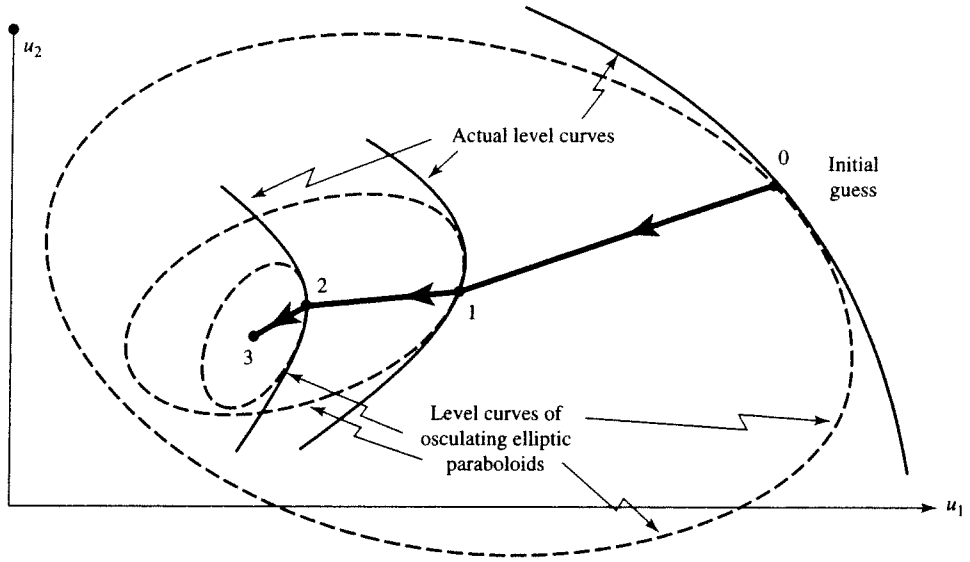

FIGURE 1.24 Typical Convergence of an NR Algorithm with Two Decision Parameters

as the gradient at the test point. Figure 1.24 is a sketch of the typical convergence of an NR algorithm with two decision parameters u_1 and u_2. Starting from an initial guess, each step fits an elliptic paraboloid to the local data and goes to the minimum of that surface (in this case, the center of the elliptical cross-section) for the next step.

Simple NR algorithms work only if the Hessian is positive definite (so that the quadratic hypersurface is a hyperelliptic paraboloid). This is a serious drawback, since, in many problems, it is difficult to find an initial guess where the Hessian is positive definite.

Another drawback is the extra time spent in determining and entering the second-derivative data into the algorithm, and the extra time necessary to compute the Hessian at each step.

However, the big advantage of NR algorithms over gradient algorithms is their rapid convergence to a precise solution *if the initial guess is such that the Hessian is positive-definite.*

Many algorithms are now available that are more efficient and more user friendly than the simple versions presented above and below, that combine the advantages of gradient and NR algorithms, e.g., NPSOL (Ref. GMSW) and the MATLAB Optimization Toolbox (Ref. Gr).

POPN—Parameter Optimization with Equality Constraints; a Newton-Raphson Algorithm

The NR derived here finds the parameter vector y to minimize or maximize $L(y)$ subject to $f(y) = 0$, where the dimension of f is less than the dimension of y.

Starting with an initial guess of y, a differential of (1.44) gives

$$d(\Delta H) \approx \left\{ \begin{bmatrix} H_y & f^T \end{bmatrix} + \begin{bmatrix} \Delta y^T & \Delta\lambda^T \end{bmatrix} \begin{bmatrix} H_{yy} & f_y^T \\ f_y & 0 \end{bmatrix} \right\} \begin{bmatrix} d(\Delta y) \\ d(\Delta\lambda) \end{bmatrix} . \tag{1.59}$$

At a minimum, $d(\Delta H) = 0$ for arbitrary $d(\Delta y)$ and $d(\Delta\lambda)$, which requires that

$$\begin{bmatrix} H_{yy} & f_y^T \\ f_y & 0 \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} H_y^T \\ f \end{bmatrix} . \tag{1.60}$$

Equation (1.60) is easily solved for Δy and $\Delta\lambda$.

The NR algorithm is summarized below.

- Guess y and set $it = 0$.
  $(*)$ $f = f(y),\ L = L(y)$.
- $f_y = f_y(y),\ L_y = L_y(y)$.
- If $it = 0$ then

  $$\lambda^T = -L_y \cdot f_y^* ,$$

  where

  $$f_y^* \triangleq f_y^T (f_y f_y^T)^{-1} ,$$

  which is the weighted *generalized inverse* of f_y.
- $H_y = L_y + \lambda^T f_y$.
- If norm$(g) <$ *tol* then end, where

  $$g \triangleq \begin{bmatrix} H_y^T \\ f \end{bmatrix} .$$

- $H_{yy} = H_{yy}(y) \triangleq L_{yy} + \sum_{i=1}^{n} \lambda_i \cdot f_{yy}^i$.
-

  $$\begin{bmatrix} \Delta y \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} H_{yy} & f_y^T \\ f_y & 0 \end{bmatrix}^{-1} \begin{bmatrix} H_y^T \\ f \end{bmatrix} .$$

- Replace y by $y + \Delta y$ and λ by $\lambda + \Delta\lambda$. Set $it = it + 1$.
- Go to $(*)$.

In the POPN algorithm above, if we replace H_{yy} by I/k, where k is a positive scalar, we obtain the POP algorithm of Section 1.3.

A MATLAB file is listed in Table 1.2 that implements this algorithm.

Example 1.5.1—Max Velocity of a Sailboat Using POPN

This is a numerical solution of Example 1.3.1 using POPN. The subroutine containing L, f and their derivatives with respect to y is listed below. The first part of this subroutine is identical to the subroutine used in the example for POP, so it could also be used with POP to get a convex approximation for use with POPN.

TABLE 1.2 POPN—A MATLAB Newton-Raphson Code

```
function [L,y,ev,evec]=popn(name,y,tol,mxit)
% Parameter OPtimization with equality constraints using a
% Newton-raphson algorithm. L is optimum performance index, output y is
% optimum parameter vector; fn = norm(f); Hyn=norm(Hy); (ev,evec)=
% (eigvals, eigvecs) of Hessian; mxit=max no. iterations input y is
% guess of parameter vector; stopping criterion is max(fn,Hyn) < tol;
% subroutine 'name' contains L,f,Ly,fy,Lyy,fyy.
%
format compact; it=0; fn=1; Hyn=1; p=length(y);
disp('        it       L        fn         Hyn')
while max([fn Hyn]) > tol
 [L,f,Ly,fy,Lyy,fyy]=feval(name,y); fn=norm(f); n=length(f);
 if it==0, lat=-Ly*fy'/(fy*fy'); end; Hy=Ly+lat*fy; Hyn=norm(Hy);
 Hyy=Lyy; for i=1:n, Hyy=Hyy+lat(i)*fyy([1+p*(i-1):i*p],:); end
 A=[Hyy fy'; fy zeros(n)]; dv=-A\[Hy';f]; y=y+dv([1:p]);
 lat=lat+dv([p+1:n+p])'; disp([it L fn Hyn]);
 if it>mxit, break, end; it=it+1;
end
%
% Finds generalized eigenvalues and eigenvectors of Hessian:
B=diag([ones(1,p) zeros(1,n)]); [X,D]=eig(A,B); ev=diag(D)';
ev=ev([1+2*n:n+p]); evec=real(X([1:p],[1+2*n:n+p]));
```

```
function [L,f,Ly,fy,Lyy,fyy]=slbt2(y)
% Subroutine for Example 1.5.1; sailboat max velocity using POPN,
   an NR
% code; L=V; y=[V Wr al th ps]'.
%
V=y(1); Wr=y(2); al=y(3); th=y(4); ps=y(5); sa=sin(al);
   ca=cos(al);
st=sin(th); ct=cos(th); p=sin(ps); cp=cos(ps); sat=sin(al+th);
cat=cos(al+th);
L=V; Ly=[1 0 0 0 0];
f=[V^2-Wr^2*sa*st; Wr^2-V^2-1+2*V*cp; Wr*sat-sp];
fy=[2*V -2*Wr*sa*st -Wr^2*ca*st -Wr^2*sa*ct 0; -2*(V-cp) 2*Wr 0
   0 ... -2*V*sp; 0 sat Wr*cat Wr*cat -cp];
Lyy=zeros(5);
f1=[2 0 0 0 0; 0 -2*st*sa -2*Wr*ca*st -2*Wr*sa*ct 0; 0 0 Wr^2*
   sa*st ... -Wr^2*ca*ct 0; 0 0 0 Wr^2*sa*st 0; 0 0 0 0 0];
fyy=f1+f1'-diag(diag(f1));
f2=[-2 0 0 0 -2*sp; 0 2 0 0 0; zeros(2,5); 0 0 0 0 -2*V*cp];
fyy([6:10],:)=f2+f2'-diag(diag(f2));
f3=[0 0 0 0 0; 0 0 cat cat 0; 0 0 -Wr*sat -Wr*sat 0; 0 0 0 -Wr*sat
   0; ... 0 0 0 0 sp];
fyy([11:15],:)=f3+f3'-diag(diag(f3));
```

An edited MATLAB diary solving the problem using POPN is shown below. It took only four iterations to bring the norms of f and of H_y below 10^{-8}. Note the two generalized eigenvalues are negative (indicating a maximum). The difficult part of using this algorithm is in selecting an initial guess for which the eigenvalues are negative. In an unfamiliar problem, it may be necessary to first use POP to get close to the maximum.

```
y=[.6 1 .6 .6 1.3]'; tol=1e-8; mxit=10;
[L,y,ev]=popn('slbt2',y,tol,mxit);
      it       L         fn         Hyn
       0    0.6000    0.0375    0.0050
  1.0000    0.5915    0.0007    0.0039
  2.0000    0.5931    0.0000    0.0005
  3.0000    0.5931    0.0000    0.0000
  4.0000    0.5931    0.0000    0.0000
y'=[.5931 1.0057 .6308 .6308 1.2797]; ev=-.2081 -.7075
```