2.2 Numerical Solution with Gradient Methods

Only very simple problems can be solved in terms of tabulated functions. Hence, in this section, we consider algorithms for numerical solution. We shall do this using the *Mayer formulation* of the problem instead of the *Bolza formulation* of (2.1) to (2.3). These two formulations are exactly equivalent, but the Mayer form yields simpler expressions, that makes it easier to code for numerical solutions.

In the Mayer formulation the state vector of (2.1) and (2.2) is augmented by one state $x_{n+1}(i)$ that is the cumulative sum of L to step i, i.e.,

$$x_{n+1}(i+1) = x_{n+1}(i) + L[x(i), u(i), i] \ , \ x_{n+1}(0) = 0 \ .$$

Thus the performance index (2.3) becomes

$$J = \phi[x(N)] + x_{n+1}(N) \triangleq \bar{\phi}[\bar{x}(N)] \ ,$$

where

$$\bar{x} \triangleq \begin{bmatrix} x \\ x_{n+1} \end{bmatrix} .$$

Dropping the bar on x and ϕ, the problem may be stated as finding a vector sequence $u(i), i = 0 \ , \ \cdots \ , \ N-1$, to minimize (or maximize)

$$\phi[x(N)] \ , \tag{2.44}$$

subject to

$$x(i+1) = f[x(i), u(i)] \ , \tag{2.45}$$

where u is $(n_c \times N)$, both f and x are $(n \times 1)$, and $x(0), \ N$ are specified.

We first describe an iterative gradient algorithm DOP0 (for Discrete OPtimization with 0 terminal constraints). It requires a subroutine that calculates $f, \ \phi, \ \phi_x, \ f_x$, and f_u and selection of a step-size parameter k. A MATLAB implementation of DOP0 is listed in Table 2.1.

Then we describe the use of the MATLAB command FMINU (for Function MINimization Unconstrained) to solve this problem. This code does *not* require the selection of a step-size parameter. Furthermore it does not require the analytical gradients $\phi_x, \ f_x, \ f_u$ since it will calculate them numerically. However the code will run faster for big problems if these gradients are provided.

Finally, in Section 2.5, we describe a *shooting algorithm,* which is a multiple interpolation algorithm for sequencing the EL equations backward to match the specified initial conditions. This algorithm yields a precise solution, but it converges only if the initial guess of the final state is very good; such a guess can be provided by DOP0 or FMINU.

DOP0—Discrete OPtimization with 0 Terminal Constraints

This is a simplified version of the POP algorithm of Section 1.3 since there are "no constraints" in the sense of that algorithm; simply replace L by ϕ and y by $u \triangleq [u(0) \cdots u(N-1)]^T$. This is a first-order gradient algorithm, which starts with an initial guess of $u(i)$, finds J and the gradient of J with respect to $u(i)$, then changes $u(i)$ by a small amount $du(i)$ in the direction of the negative gradient, and iterates until the magnitude of $u(i)$ becomes small. A more detailed description is given below.

- Enter $x(0), \ t_f, \ k, \ tol$, and a guess of $u(i), \ i = 0, \cdots, N-1$; choose $k > 0$ if minimizing, $k < 0$ if maximizing.

$(*)$ Sequence forward. Compute and store $x(i)$.

- Evaluate $\phi[x(N)]$ and set $\lambda^T(N) = \phi_x$.
- Sequence backward. Compute and store the pulse response sequence $H_u(i)$. For $i = N-1, \cdots, 0$:

$$H_u(i) = \lambda^T(i+1) f_u(i) \ , \tag{2.46}$$

$$\lambda^T(i) = \lambda^T(i+1) f_x(i) \ . \tag{2.47}$$

TABLE 2.1 A MATLAB Code for DOP0

```
function [u,s,la0]=dop0(name,u,s0,tf,k,tol,mxit)
% Discrete OPtim. with 0 term. Constraints, tf specified, nc controls.
% Inputs: name must be in single quotes; function file 'name' computes
% s(i+1)= f(s(i),u(i))for flg=1, (phi,phis) for flg=2, and (fs,fu)
% for flg=3; u(nc,N) = estimate of optimal u; s0(ns,1) = initial state;
% tf = final time; k = step size parameter; u should be normalized so
% that the elements of du are roughly the same size; stopping criterion
% is max(dua) < tol; mxit = max no iterations; outputs (u,s) = improved
% (u,s) histories; la0 = optimal lambda(0) for possible use with a
% shooting algorithm.
% BASIC version 1984; MATLAB version 1994; rev. 1/8/98
%
if nargin<7, mxit=10; end;
ns=length(s0); [nc,N]=size(u); s=zeros(ns,N+1); dum=zeros(nc,1);
la=zeros(ns,1); Hu=zeros(N,nc); dua=1; it=0; dt=tf/N; s(:,1)=s0;
disp('     Iter.        phi        dua');
while norm(dua)>tol,
 % Forward sequencing and store state histories x(:,i):
 for i=1:N, s(:,i+1)=feval(name,u(:,i),s(:,i),dt,(i-1)*dt,1); end
 % Performance index phi and b.c. for backward sequence phis:
 [phi,phis]=feval(name,dum,s(:,N+1),dt,N*dt,2); la=phis';
 % Backward sequencing and store Hu(i);
 for i=N:-1:1
    [fs,fu]=feval(name,u(:,i),s(:,i),dt,(i-1)*dt,3);
    Hu(i,:)=la'*fu; la=fs'*la;
 end; la0=la;
 % New u(i):
 for j=1:nc,
    du(j,:)=-k*Hu(:,j)'; dua(j)=norm(du(j,:))/sqrt(N);
 end; u=u+du;
 disp([it phi dua]);
 if it>mxit, break, end; it=it+1;
end
```

- Compute $\Delta u(i)$ and Δu_{avg}. For $i = 0, \cdots, N-1$

$$\Delta u(i) = -kH_u^T(i)\ , \tag{2.48}$$

$$\Delta u_{avg} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} \Delta u^T(i)\Delta u(i)}\ . \tag{2.49}$$

- If $\Delta u_{avg} < tol$ then stop.

- Compute new $u(i)$:

$$u(i) \leftarrow u(i) + \Delta u(i)\ , \tag{2.50}$$

- Go to $(*)$.

A code for implementing the DOP0 algorithm in MATLAB is listed in Table 2.1. The data for the problem being solved are put into a subroutine "name" that gives the f functions, the performance index ϕ and its gradient ϕ_s, and the derivatives f_s, f_u. Note that MATLAB starts sequences with $i = 1$ where we have used $i = 0$ above.

Example 2.2.1—DVDP for Max Range with Gravity

This is the same example solved analytically in Example 2.1.1. The subroutine used is listed below.

```
function [f1,f2]=dvdp0(ga,s,dt,t,flg)
% s=[v x]'; ga=gamma; t in units of tf, v in g(tf), x in g(tf)^2.
%
v=s(1); x=s(2);
if flg==1,
 f1=s+dt*[sin(ga); v*cos(u)+dt*sin(2*ga)/4];                    % f1 = f
elseif flg==2,
 f1=x; f2=[0 1];                                  % f1 = phi, f2 = phis
elseif flg==3,
 f1=[1 0; dt*cos(ga) 1];                                       % f1 = fs
 f2=dt*[cos(ga); -v*sin(ga)+dt*cos(2*ga)/2];                   % f2 = fu
end
```

An edited MATLAB diary using DOP0 is shown below. It took only five iterations to bring du_a below .0001. It takes a little practice to learn how to select the step-size parameter k. Start with small values and put mxit=3 so that the code runs only three steps and stops; gradually increase k until *dua* and ϕ increase instead of decreasing (for a minimization problem); k is then too large and you will overshoot the minimum; decrease k by a small amount so that *dua* and ϕ decrease and *dua* is a reasonable size (i.e., one could expect linear prediction from one step to the next to be fairly accurate).

```
% Script e02_2_1.m; DVDP for max xf with gravity;
%
ga=[1:-.25:0]; s0=[0 0]'; tf=1; k=-7; tol=5e-5;
[ga,s]=dop0('dvdp0',ga,s0,tf,k,tol);
    Iter.      phi        dua
        0    0.2780     0.3553
        -       -          -
   6.0000    0.3157     0.0000
ga=[1.4137 1.0996 .7854 .4712 .1571]
s=[0 .3566 .9836 1.8709 2.9486; 0 .4341 .8246 1.1245 1.2457]
```