Contents

1 Boundary value problems
   1.1 Reconstructing trajectories of particles moving in force fields . . . 2
   1.2 Deflection of a loaded beam . . . . . . . . . . . . . . . . . . . . . . . . 3

2 ODE Approach to Boundary-Value Problems: The Shooting Method 4

3 Linear-Algebra Approach to Boundary-Value Problems: The Finite-Difference Method 5
   3.1 Example: The beam equation . . . . . . . . . . . . . . . . . . . . . . . . 6
1 Boundary value problems

In our discussion of ODEs we considered initial value problems—that is, ODEs \( \frac{du}{dt} = f(t, u) \) in which we are given a vector \( u_0 \) specifying all components of the \( u \) vector at a single time point \( t_0 \). In such a situation, we are guaranteed (assuming \( f \) satisfies certain niceness conditions discussed in our unit on ODEs) the existence of a unique curve \( u(t) \) that satisfies the differential equation and runs through the point \( t_0, u_0 \).

An alternative type of ODE is the boundary value problem. In this case, we are given only partial data for the components of the \( u \) vector, but we are given these data for multiple time points \( t \). Such problems arise in many fields of science and engineering; for the purposes of numerical analysis they are interesting not only because they reveal the limitations of the ODE techniques we discussed previously, but also because they motivate the introduction of finite-difference solution techniques, which then extend immediately to higher-dimensional PDEs.

1.1 Reconstructing trajectories of particles moving in force fields

For example, suppose we are biologists observing under a microscope the motion of a bioparticle moving in a time-dependent force field \( F(t) = F(t)\hat{x} \). (For simplicity, we will consider here the case of 1D motion, although it is easy to extend the discussion to higher dimensions.) For example, if the bioparticle has charge \( q \) and the \( x \)-component of the electric field is \( E_x(t) \), then the force is \( F(t) = qE_x(t) \).

Suppose we observe that the position of the particle at time \( t_1 \) is \( x_1 \), while at some later time \( t_2 \) it is at position \( x_2 \). (Note that we do not observe the velocity of the particle.) We would like to reconstruct the trajectory that the particle followed between \( t_1 \) and \( t_2 \). We then have a boundary-value problem of the form

\[
\frac{d^2x}{dt^2} = \frac{1}{m} F(t), \quad x(t_1) = x_1, \quad x(t_2) = x_2. \tag{1}
\]

where \( m \) is the mass of the bioparticle. To phrase this equation in the language of first-order ODE systems, we define \( u_1 = x, u_2 = \dot{x} \) and obtain the ODE system

\[
\frac{du}{dt} = \frac{d}{dt} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = \begin{pmatrix} u_2 \\ F(t)/m \end{pmatrix} \tag{2}
\]

subject to the boundary conditions

\[
u(t_1) = \begin{pmatrix} x_1 \\ ? \end{pmatrix}, \quad u(t_2) = \begin{pmatrix} x_2 \\ ? \end{pmatrix}. \tag{3}\]

The point is that we don’t know the velocity of the particle at either endpoint, which means we don’t have an initial-value problem. This has at least two immediate implications:
(a) the nice existence and uniqueness theorems for initial-value problems go completely out the window; for a boundary-value problem like (2) there may be no solution, or there may be multiple solutions, and these things may be true even for perfectly nice \( f \) functions.

(b) Even assuming there is a solution curve \( u(t) \), we can’t use the ODE algorithms we discussed previously to find points on it, because all of those algorithms required that we start with a known point on the curve. In this case we don’t know all the coordinates of even a single point on the curve, so none of our ODE integrators can get started.

1.2 Deflection of a loaded beam

Another classic example of a boundary-value problem is the deflection of a beam of constant cross-section forced to support a position-dependent weight (mechanical engineers would say “subject to a position-dependent load”). The relevant equation here is the Euler-Bernoulli equation,

\[ \alpha \frac{d^4 h}{dx^4} = q(x) \]

where \( h(x) \) is the vertical deflection of the beam at position \( x \), \( q(x) \) is the position-dependent loading of the beam\(^1\), and \( \alpha \) is a material-dependent rigidity parameter describing the beam’s resistance to shearing. Suppose the beam is affixed rigidly to two supporting walls at positions \( x_1 \) and \( x_2 \). This means that both the beam’s deflection and slope are constrained to be 0 at both endpoints, or in other words

\[ h(x_1) = 0, \quad h'(x_1) = 0, \quad h(x_2) = 0, \quad h'(x_2) = 0. \]

If we proceed in the usual way to convert equation (10) to a first-order ODE system, we obtain

\[
\frac{d\mathbf{u}}{dt} = \frac{d}{dt} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} u_2 \\ u_3 \\ u_4 \\ q(u_1)/\alpha \end{pmatrix} \tag{5}
\]

subject to the boundary conditions

\[
\mathbf{u}(x_1) = \begin{pmatrix} 0 \\ 0 \\ ? \\ ? \end{pmatrix}, \quad \mathbf{u}(x_2) = \begin{pmatrix} 0 \\ 0 \\ ? \\ ? \end{pmatrix}. \tag{6}
\]

As before, we can’t simply use an ODE integrator to solve this equation because we don’t have any full point on the solution curve from which to start integrating.

\(^1\)For example, if the beam in question were a bookshelf, and there were heavier books near the center of the shelf and lighter books near its edges, then the function \( q(x) \) would be peaked near the center of the interval.


2 ODE Approach to Boundary-Value Problems: 
The Shooting Method

We noted above that our standard bag of ODE tricks for integrating initial-value problems (such as Euler's method or RK4) can't get started on a boundary-value problem like (2) or (5), because in order to use e.g. Euler's method we need to know a point on the solution curve. In a problem like (2) we only know "half" of a point on a solution curve at \( t_1 \) – we know the \( u_1 \) coordinate of the point, but not the \( u_2 \) coordinate.

There is, however, a way to remedy this difficulty. Starting at \( t = t_1 \), we 
guess a number for the \( u_2 \) coordinate. In the case of (2), this corresponds to 
guessing an initial velocity for the particle. Denote our guess by \( u_2^{\text{guess}} \). We now 
have the coordinates of one full point on a curve at time \( t_1 \), and we call this point \( u^{\text{guess}} \):

\[
u^{\text{guess}} = \begin{pmatrix} u_1 \\ u_2^{\text{guess}} \end{pmatrix}
\]

The existence and uniqueness theorems now guarantee that there exists a full curve \( u^{\text{guess}}(t) \) satisfying the differential equation and the condition \( u^{\text{guess}}(t_0) = u_0^{\text{guess}} \). So we can now use any ODE algorithm we like to integrate our equation to compute more points on this curve. In particular, we can integrate all the way from \( t_1 \) to \( t_2 \) and evaluate the value of \( u^{\text{guess}}(t_2) \). If this value equals \( x_2 \), we’re done! We have found our desired solution curve. If not, we have to go 
back and try a new value for \( u_2^{\text{guess}} \).

This method is known as the “shooting method,” for obvious reasons: integrating from \( t_1 \) to \( t_2 \) with initial position and velocity \( u_1, u_2^{\text{guess}} \) corresponds to 
“shooting” the particle from that position with that velocity, and if we guess the initial velocity just right then the particle will just pass through position \( u_2 \) at time \( u_2 \).

The difficulty is that we now have to solve a root-finding problem to compute \( u_2^{\text{guess}} \). Indeed, for each choice of \( u_2^{\text{guess}} \) at time \( t_1 \) we can integrate the resulting initial-value problem and compute the value it predicts for the coordinate \( u_1 \) at time \( t_2 \). Denote this value by \( u_1^{\text{integrated}}(u_2^{\text{guess}}, t_2) \). Choosing the correct value of \( u_2^{\text{guess}} \) then corresponds to finding a root of the nonlinear equation

\[
u_1^{\text{integrated}}(u_2^{\text{guess}}, t_2) - u_1^{\text{desired}}(t_2) = 0 \tag{7}
\]

where \( u_1^{\text{desired}}(t_2) \) is the given boundary-value at time \( t_2 \).

Equation (7), a nonlinear root-finding problem, is much more difficult to solve than standard initial-value ODE problems. Moreover, for a problem like 6 in which we are missing two or more necessary components from the initial-condition vector, we face the problem of finding a root of a multidimensional function, again much harder than simply integrating an ODE.
3 Linear-Algebra Approach to Boundary-Value Problems: The Finite-Difference Method

An alternative approach to boundary-value problems is to convert a differential equation like (2) or (5) into an algebraic equation—more specifically, a linear system of equations involving a matrix and two vectors—which we then solve using computational linear algebra. This is the idea behind the finite-difference method. It has several advantages over the shooting method outlined above, the most significant of which is that it readily extends to higher dimensions, where it constitutes one of the most widely used techniques for solving partial differential equations (PDEs).

The key idea here is something we discussed in our notes on numerical differentiation: when we work with finite-difference approximations to derivatives, the operation of differentiation is equivalent to the operation of matrix multiplication. More specifically, if we have a vector $f$ whose entries are samples of some function $f(x)$ at evenly-spaced sample points, then there exists a matrix $A$ such that the matrix-vector product $Af$ is a vector whose entries are samples of the second derivative of $f$, i.e. if we have an interval $[a, b]$ and we define the vectors

\[
\begin{align*}
\mathbf{f} &= \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_N \end{pmatrix}, & \mathbf{f}'' &= \begin{pmatrix} f''_1 \\ f''_2 \\ \vdots \\ f''_N \end{pmatrix}
\end{align*}
\]

where

\[
f_n \equiv f(a + nh), \quad f''_n \equiv f''(a + nh), \quad n = 1, \cdots, N, \quad h = \frac{b - a}{N + 1}
\]

then the vectors $\mathbf{f}$ and $\mathbf{f}''$ are related\(^3\) by

\[Af = \mathbf{f}''\quad (8)\]

where the matrix $A$ looks like

\[
A = \frac{1}{h^2} \begin{pmatrix}
-2 & 1 & 0 & \cdots & 0 & 0 \\
1 & -2 & 1 & \cdots & 0 & 0 \\
0 & 1 & -2 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & -2 & 1 \\
0 & 0 & 0 & \cdots & 1 & -2
\end{pmatrix}.
\]

\(^2\)Of course this technique is not limited to the second derivative; we could alternatively write down different matrices that, when applied to $f$, yield vectors of samples of its first derivative, its fourth derivative, etc.

\(^3\)Equation (8) assumes that $f(a) = f(b) = 0$. Implementation of nontrivial boundary conditions is discussed in our lecture notes on numerical differentiation.
(Equation (8) assumes that \( f \) satisfies the boundary conditions \( f(a) = f(b) = 0 \); other boundary conditions may be represented by adding suitable terms to the RHS.)

Of course, as soon as we write down equation (8) we can immediately proceed to invert that equation to find a relation predicting values of \( f \) from the values of \( f'' \):

\[
f = A^{-1}f''.
\]

(9)

The usefulness of this equation is that, in a boundary-value problem, we typically have a relation expressing \( f'' \) in terms of some known function. For example, in (1), the second derivative of the function we seek is related to the (known) force field \( F(x) \). Then all we have to do is replace \( f'' \) in (9) with the expression for the second derivative given by the differential equation in question, and we can immediately solve for samples of the function \( f(x) \).

3.1 Example: The beam equation

In this section we’ll work through a finite-difference method for solving the one-dimensional beam equation

\[
\frac{d^4f}{dx^4} = \frac{1}{\alpha} q(x)
\]

over an interval \([a, b]\) with boundary conditions

\[
f(a) = f'(a) = f(b) = f'(b) = 0.
\]

(11)

Finite-difference stencil for \( \frac{d^4}{dx^4} \)

It is easy to verify that a finite-difference stencil with stepsize \( h \) for the fourth derivative of a function \( f(x) \) at a point \( x \) is

\[
f^{(4)}_{FD}(h, x) = \frac{f(x - 2h) - 4f(x - h) + 6f(x) - 4f(x + h) + f(x + 2h)}{h^4}
\]

(12)

This stencil achieves second-order convergence, i.e. if \( f^{(4)}(x) \) is the exact fourth derivative of \( f \) at \( x \), then we have

\[
\left| f^{(4)}_{FD}(h, x) - f^{(4)}(x) \right| = O(h^2)
\]

Implementation of boundary conditions

When we attempt to apply (12) at points within 1 or 2 sites of the ends of the interval, we find that we need values for the quantities \( f_{-1}, f_0, f_{N+1}, f_{N+2} \). The values of \( f_0 \) and \( f_{N+1} \) are fixed by the boundary conditions (12) to be 0. This leaves unspecified the values of \( f_{-1} \) and \( f_{N+2} \), but the condition that \( f' = 0 \) at both endpoints winds up being equivalent to the requirement that \( f_{-1} = f_{N+2} = 0 \). (Less trivial boundary conditions could be handled using the method described in our lecture notes on numerical differentiation.)
The matrix $A$

In view of the above considerations, the finite-difference matrix we want is

$$A = \frac{1}{h^4} \begin{pmatrix} 6 & -4 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ -4 & 6 & -4 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & -4 & 6 & -4 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & -4 & 6 & -4 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & -4 & 6 & -4 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 6 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -4 & 6 \end{pmatrix}.$$  

This matrix operates on a vector of samples of $f$ to yield a vector of samples of $f^{(4)}$:

$$Af = f^{(4)}$$

where the $n$th elements of $f$ and $f^{(4)}$ are respectively

$$f_n = f(a + nh), \quad f^{(4)}_n = f^{(4)}(a + nh), \quad h = \frac{b - a}{N + 1}$$

and where we have assumed $f_{-1} = f_0 = f_{N+1} = f_{N+2} = 0$.

The solution

Inverting equation (13), we have

$$f = A^{-1} \cdot f^{(4)},$$

On the other hand, the differential equation (10) allows us to compute values of $f''$ here in terms of the loading function $q(x)$, i.e. we can put

$$f^{(4)} = \frac{1}{\alpha} q$$

where the elements of the vector $q$ are the values of the function $q(x)$ at the sample point $x_n$. Then equation (15) reads

$$f = A^{-1} \left( \frac{1}{\alpha} q \right)$$

We solve this equation numerically using the JULIA code reproduced below. The results, for a forcing function $q(x) = x^2$, are plotted in Figure 1.

```julia
# solve the beam equation on the interval [0:10] given a
# loading function q(x), a stiffness parameter Alpha, and a
# dimension N (where N is the dimension of the solution
# vector, so the stepsize is (b-a)/(N+1) )
```
Figure 1: Solution of beam equation with loading function $q(x) = x^2$. 
function SolveBeamEquation(q, Alpha, N)

b=10.0;
a=0.0;
h=(b-a)/(N+1);
h4=h^4;

# start by making A a diagonal matrix with 6s on the diagonal
A=6*eye(N,N) / h4;

# add the -4s on the first upper and lower sub-diagonals
for n=1:N-1
    A[n,n+1]=-4.0 / h4;
    A[n+1,n]=-4.0 / h4;
end

# add the +1s on the second upper and lower sub-diagonals
for n=1:N-2
    A[n,n+2]=+1.0 / h4;
    A[n+2,n]=+1.0 / h4;
end

# form the RHS vector
# (XVector is just a vector of the sample points)
# note we interpret q as the positive (upward-directed)
# loading, so for downward-directed loading we want -q
xVector = zeros(N);
RHSVector = zeros(N);
for n=1:N
    xVector[n] = a+n*h;
    RHSVector[n] = -q(xVector[n]) / Alpha;
end

# solve the system to obtain the solution vector y
yVector = A\RHSVector;
end