5.5 Exercises

1. Approximate the integrals, using $n = 2$ Gaussian Quadrature. Compare with the correct value, and give the approximation error.

$$\text{(a)} \int_{-1}^{1} (x^3 + 2x)\,dx \quad \text{(b)} \int_{-1}^{1} x^4\,dx \quad \text{(c)} \int_{-1}^{1} e^x\,dx \quad \text{(d)} \int_{-1}^{1} \cos \pi x\,dx$$

2. Approximate the integrals in Exercise 1, using $n = 3$ Gaussian Quadrature, and give the error.

3. Approximate the integrals in Exercise 1, using $n = 4$ Gaussian Quadrature, and give the error.

4. Change variables, using the substitution (5.46) to rewrite as an integral over $[-1, 1]$.

$$\text{(a)} \int_{0}^{4} \frac{x\,dx}{\sqrt{x^2 + 9}} \quad \text{(b)} \int_{0}^{1} \frac{x^3\,dx}{x^2 + 1} \quad \text{(c)} \int_{0}^{1} xe^x\,dx \quad \text{(d)} \int_{1}^{3} x^2 \ln x\,dx$$

5. Approximate the integrals in Exercise 4, using $n = 3$ Gaussian Quadrature.

6. Approximate the integrals, using $n = 4$ Gaussian Quadrature.

$$\text{(a)} \int_{0}^{1} (x^3 + 2x)\,dx \quad \text{(b)} \int_{1}^{4} \ln x\,dx \quad \text{(c)} \int_{-1}^{2} x^5\,dx \quad \text{(d)} \int_{-3}^{3} e^{-\frac{x^2}{2}}\,dx$$

7. Show that the Legendre polynomials $p_1(x) = x$ and $p_2(x) = x^2 - 1/3$ are orthogonal on $[-1, 1]$.

8. Find the Legendre polynomials up to degree 3 and compare with Example 5.13.

9. Verify the coefficients c_i and x_i in Table 5.1 for degree $n = 3$.

10. Verify the coefficients c_i and x_i in Table 5.1 for degree $n = 4$.

Reality Check 5

Motion Control in Computer-Aided Modeling

Computer-aided modeling and manufacturing requires precise control of spatial position along a prescribed motion path. We will illustrate the use of Adaptive Quadrature to solve a fundamental piece of the problem: equipartition, or the division of an arbitrary path into equal-length subpaths.

In numerical machining problems, it is preferable to maintain constant speed along the path. During each second, progress should be made along an equal length of the machine–material interface. In other motion planning applications, including computer animation, more complicated progress curves may be required: A hand reaching for a doorknob might begin and end with low velocity and have higher velocity in between. Robotics and virtual reality applications require the construction of parametrized curves and surfaces to be navigated. Building a table of small equal increments in path distance is often a necessary first step.

Assume that a parametric path $P = \{x(t), y(t) | 0 \le t \le 1\}$ is given. Figure 5.6 shows the example path

$$P = \begin{cases} x(t) = 0.5 + 0.3t + 3.9t^2 - 4.7t^3 \\ y(t) = 1.5 + 0.3t + 0.9t^2 - 2.7t^3 \end{cases},$$

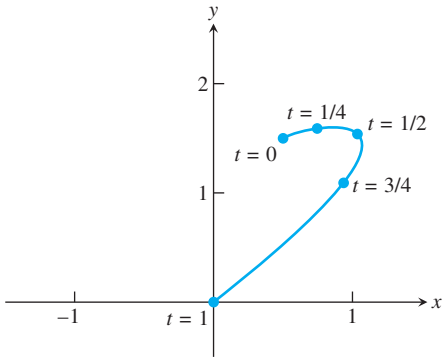

Figure 5.6 Parametrized curve given by Bézier spline. Typically, equal intervals of the parameter t do not divide the path into segments of equal length.

which is the Bézier curve defined by the four points $(0.5, 1.5)$, $(0.6, 1.6)$, $(2, 2)$, $(0, 0)$. (See Section 3.5.) Points defined by evenly spaced parameter values $t = 0, 1/4, 1/2, 3/4, 1$ are shown. Note that even spacing in parameter does not imply even spacing in arc length. Your goal is to apply quadrature methods to divide this path into n equal lengths.

Recall from calculus that the arc length of the path from t_1 to t_2 is

$$\int_{t_1}^{t_2} \sqrt{x'(t)^2 + y'(t)^2}\, dt.$$

Only rarely does the integral yield a closed-form expression, and normally an Adaptive Quadrature technique is used to control the parametrization of the path.

Suggested activities:

1. Write a MATLAB function that uses Adaptive Quadrature to compute the arc length from $t = 0$ to $t = T$ for a given $T \leq 1$.

2. Write a program that, for any input s between 0 and 1, finds the parameter $t^*(s)$ that is s of the way along the curve. In other words, the arc length from $t = 0$ to $t = t^*(s)$ divided by the arc length from $t = 0$ to $t = 1$ should be equal to s. Use the Bisection Method to locate the point $t^*(s)$ to three correct decimal places. What function is being set to zero? What bracketing interval should be used to start the Bisection Method?

3. Equipartition the path of Figure 5.6 into n subpaths of equal length, for $n = 4$ and $n = 20$. Plot analogues of Figure 5.6, showing the equipartitions. If your computations are too slow, consider speeding up the Adaptive Quadrature with Simpson's Rule, as suggested in Computer Problem 5.4.2.

4. Replace the Bisection Method in Step 2 with Newton's Method, and repeat Steps 2 and 3. What is the derivative needed? What is a good choice for the initial guess? Is computation time decreased by this replacement?

5. Appendix A demonstrates animation commands available in MATLAB. For example, the commands

```
set(gca,'XLim',[-2 2],'YLim',[-2 2],'Drawmode','fast',...
   'Visible','on');
cla
axis square
```

```
ball=line('color','r','Marker','o','MarkerSize',10,...
   'LineWidth',2, 'erase','xor','xdata',[],'ydata',[]);
```

define an object "ball" that is assigned position (x, y) by the following commands:

```
set(ball,'xdata',x,'ydata',y); drawnow;pause(0.01)
```

Putting this line in a loop that changes x and y causes the ball to move along the path in the MATLAB figure window.

Use MATLAB's animation commands to demonstrate traveling along the path, first at the original parameter $0 \le t \le 1$ speed and then at the (constant) speed given by $t^*(s)$ for $0 \le s \le 1$.

6. Experiment with equipartitioning a path of your choice. Build a design, initial, etc. of your choice out of Bézier curves, partition it into equal arc length segments, and animate as in Step 5.

7. Write a program that traverses the path P according to an arbitrary **progress curve** $C(s), 0 \le s \le 1$, with $C(0) = 0$ and $C(1) = 1$. The object is to move along the curve C in such a way that the proportion $C(s)$ of the path's total arc length is traversed between 0 and s. For example, constant speed along the path would be represented by $C(s) = s$. Try progress curves $C(s) = s^{1/3}, C(s) = s^2, C(s) = \sin s\pi/2$, or $C(s) = 1/2 + (1/2)\sin(2s - 1)\pi/2$, for example.

Consult Wang et al. [2003] and Guenter and Parent [1990] for more details and applications of reparametrization of curves in the plane and space. ✔

Software and Further Reading

The closed and open Newton–Cotes Methods are basic tools for approximating definite integrals. Romberg Integration is an accelerated version. Most commercial software implementations involve Adaptive Quadrature in some form. Classic texts on numerical differentiation and integration include Davis and Rabinowitz [1984], Stroud and Secrest [1966], Krommer and Ueberhuber [1998], Engels [1980], and Evans [1993].

Many effective quadrature techniques are implemented by Fortran subroutines in the public–domain software package Quadpack (Piessens et al. [1983]), available in Netlib (`www.netlib.org/quadpack`). The Gauss–Kronrod Method is an adaptive method based on Gaussian Quadrature. Quadpack provides nonadaptive and adaptive methods QNG and QAG, respectively, the latter based on Gauss–Kronrod. The programs in both IMSL and NAG are based on the Quadpack subroutines. The `quadrature` class in IMSL is the java implementation, for example.

MATLAB's `quad` command is an implementation of adaptive composite Simpson's Quadrature, and `dblquad` handles double integrals. MATLAB's Symbolic Toolbox has commands `diff` and `int` for symbolic differentiation and integration, respectively.

Integration of functions of several variables can be done by extending the one-dimensional methods in a straightforward way, as long as the integration region is simple; for example, see Davis and Rabinowitz [1984] and Haber [1970]. For some complicated regions, Monte Carlo integration is indicated. Monte Carlo is easier to implement, but converges more slowly in general. These issues are discussed further in Chapter 9.