AER336: Scientific Computing

Lecture Notes

Version 1.0 (Winter 2022)

Masayuki Yano

University of Toronto
Institute for Aerospace Studies

©2016–2022 Masayuki Yano, University of Toronto.

Acknowledgment

I have prepared this lecture notes for a third-year undergraduate course AER336 Scientific Computing taught at the University of Toronto. Much of the presentation was inspired by my previous work with Prof. Anthony Patera and James Penn at MIT on an undergraduate numerical methods course, 2.086 Numerical Methods for Mechanical Engineers; I would like to thank them for many fruitful discussions on pedagogy over the years. Special thanks also goes to Debbie Blanchard, who was instrumental in our 2.086 curriculum development effort at MIT.

I would also like to thank all of my past AER336 students who have provided many helpful feedback, directly or indirectly, and improved the quality of the notes.

The topics covered in the course are based on those covered by Profs. Clinton Groth and David Zingg at the University of Toronto in previous years.

The following lectures are influenced by the other teaching resources available online:

- A significant fraction of Lectures 8 and 9 on least-squares and regression were originally developed for MIT 2.086 and is also found in *Numerical methods for mechanical engineers* by Masayuki Yano, James Penn, and Anthony Patera, available on MIT OpenCourseWare.
- Lecture 17 on boundary value problems is inspired by the lecture notes on finite difference methods used in MIT 16.920 prepared by Profs. Anthony Patera, Jaime Peraire, and Jacob White, available on MIT OpenCourseWare.

Contents

Lecture 1

Polynomial interpolation

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

1.1 Motivation

Suppose we are given $n+1$ data points (x_i, y_i), $i = 1, \dots, n+1$, and wish to estimate the value y at some arbitrary point x which does not belong to the data points. One way to approach the problem is by *polynomial interpolation*. In this chapter, we focus on *global* polynomial interpolation based on a single, high-order polynomial; in the next chapter, we will consider *piecewise* polynomial interpolation based on multiple lower-order polynomials.

1.2 Polynomial interpolation

We can define the global polynomial interpolation problem as follows: given $n+1$ distinct data points (x_i, y_i), $i = 1, \dots, n+1$, over an interval $[a, b]$, find a degree-n polynomial

$$p_n(x) = \sum_{j=0}^{n} a_j x^j$$

such that

$$p_n(x_i) = y_i, \quad i = 1, \dots, n+1.$$

The polynomial p_n is called the *interpolant*. The points at which the interpolant matches the underlying data are called the *interpolation points*. An example of a polynomial interpolant is shown in Figure 1.1. A polynomial interpolant is uniquely characterized by two ingredients:

1. the polynomial degree n of the interpolant;
2. the location of the interpolation points.

We will see in this lecture how the choice of the ingredients influences the accuracy and computational cost of the interpolation process.

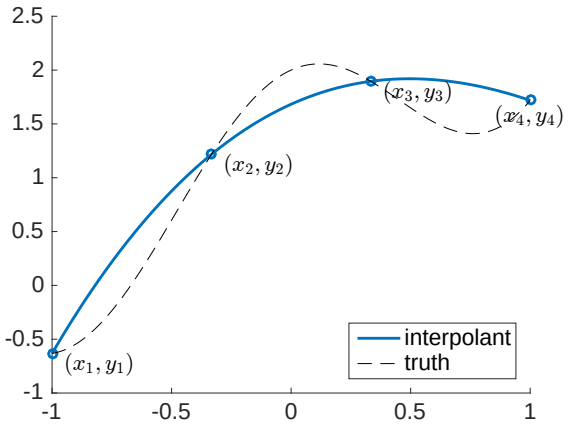


Figure 1.1: Interpolation of a function over $[-1, 1]$ by a degree-3 polynomial using equispaced interpolation points.

1.3 Construction of interpolants: Vandermonde's method

We now introduce Vandermonde's method, a systematic approach to construct polynomial interpolants.

Case $n = 1$ (linear interpolation). We first consider the case were we are given two data points, (x_1, y_1) and (x_2, y_2). We assume $x_1 \neq x_2$. Since $n = 2 - 1 = 1$, we consider a linear polynomial of the form $p_1(x) = a_0 + a_1 x$; our goal is to identify the two unknowns a_0 and a_1 to find the interpolant. The two interpolation conditions can be expressed as a system of linear equations in the unknowns a_0 and a_1:

$$\begin{aligned} y_1 &= p_1(x_1) = a_0 + a_1 x_1 \\ y_2 &= p_1(x_2) = a_0 + a_1 x_2. \end{aligned}$$

We could write equivalent conditions in the matrix form:

$$\begin{pmatrix} 1 & x_1 \\ 1 & x_2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}.$$

The matrix on the left hand side is called the *Vandermonde matrix*. We observe that for $x_1 \neq x_2$, the rows (or columns) of the matrix are linearly independent and the matrix is non-singular. We can hence find a unique solution (a_0, a_1). We can then evaluate the interpolant at any arbitrary point x by evaluating $p_1(x) = a_0 + a_1 x$.

Due to the small size of the system associated with linear interpolation, we can readily find explicit expressions for the linear interpolant:

$$\begin{aligned} p_1(x) &= \left(y_1 - \frac{y_2 - y_1}{x_2 - x_1} x_1 \right) + \left(\frac{y_2 - y_1}{x_2 - x_1} \right) x \\ &= y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \\ &= \left(1 - \frac{x - x_1}{x_2 - x_1} \right) y_1 + \left(\frac{x - x_1}{x_2 - x_1} \right) y_2. \end{aligned} \tag{1.1}$$

The first form allows us to readily identify the coefficients a_0 and a_1. The second form provides the slope-intercept interpretation. The third form groups together the terms with y_1 and y_2.

General case. We now consider the general case with $n+1$ data points, (x_i, y_i), $i = 1, \ldots, n+1$, where the interpolation points $x_1, \ldots, x_{n+1}$ are distinct. We consider an interpolant of the form $p_n(x) = a_0 + a_1 x + \cdots + a_n x^n = \sum_{j=0}^{n} a_j x^j$. We impose the interpolation conditions

$$y_i = p_n(x_i) = \sum_{j=0}^{n} a_j x^j \quad \forall i = 1, \ldots, n+1,$$

or, more explicitly,

$$\begin{aligned} a_0 + a_1 x_1 + \cdots + a_n x_1^n &= y_1 \\ a_0 + a_1 x_2 + \cdots + a_n x_2^n &= y_2 \\ &\vdots \\ a_0 + a_1 x_{n+1} + \cdots + a_n x_{n+1}^n &= y_{n+1}. \end{aligned}$$

An equivalent linear system in the matrix form is

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^n \\ 1 & x_2 & \cdots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & \cdots & x_{n+1}^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{pmatrix}. \tag{1.2}$$

For $n+1$ distinct interpolation points, the rows (or columns) of the matrix are linearly independent and the Vandermonde matrix is non-singular. We can hence find a unique solution to the linear system and the associated interpolant.

Cost. To construct the interpolant using the Vandermonde's method, we need to first populate the Vandermonde matrix and then solve the linear system. To populate the first row of the Vandermonde matrix, we must evaluate $1, x_1, x_1^2, \ldots, x_1^n$; the computation requires n multiplications. We must repeat the operation for each of the $n+1$ interpolation points; the total cost to populate the Vandermonde matrix is $\mathcal{O}(n^2)$. Once we populate the matrix, we need to solve the linear system; as we will see in a proceeding lecture, the solution of a $(n+1) \times (n+1)$ linear system requires $\mathcal{O}(n^3)$. Hence the total cost to construct the interpolant using the Vandermonde's method is $\mathcal{O}(n^3)$. (Recall that we only worry about the most dominant cost.)

To evaluate the interpolant $p_n(x) = \sum_{j=0}^{n} a_j x^j$, we first need to evaluate $1, x, x^2, \ldots, x^n$, multiply the terms with the respective coefficients $a_0, a_1, \ldots, a_n$, and then sum the terms together. Each of these three steps require $\mathcal{O}(n)$ operations. Hence the total cost to evaluate the interpolant for a given x is $\mathcal{O}(n)$.

Conditioning of the Vandermonde system. The Vandermonde's approach is a systematic approach to construct a degree n polynomial interpolant associated with $n+1$ data points. For $n+1$ distinct interpolation points, the solution is unique — at least in *exact arithmetic*. Unfortunately, in particular for a large n, the columns of the linear system (1.2) become nearly linearly dependent and the system becomes *ill-conditioned*. We will study the effect of ill-conditioning in a later lecture; for now, it suffices to know that we cannot accurately find the coefficients $a_0, \ldots, a_n$ that defines the interpolant for a large n using the Vandermonde's approach.

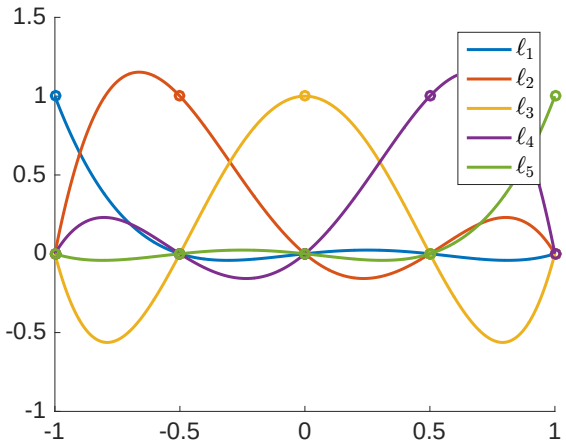


Figure 1.2: Lagrange basis for degree-$n = 4$ interpolation using equidistributed interpolation points over $[-1, 1]$, $(-1.0, -0.5, 0, 0.5, 1.0)$.

1.4 Construction of interpolants: Lagrange basis polynomials

In order to overcome the issue of ill-conditioning and to reduce the computational cost, we can consider a different construction of interpolant. Here we introduce the *Lagrange basis polynomials* associated with the $n+1$ interpolation points x_i, $i = 1, \ldots, n+1$. Lagrange basis polynomials are the set of $n+1$ degree-n polynomials ℓ_j, $j = 1, \ldots, n+1$, with the property

$$\ell_j(x_i) = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases} \tag{1.3}$$

Note that, for each j, the $n+1$ constraints define a unique degree n polynomial. The explicit form of the polynomial is given by

$$\ell_j(x) = \prod_{\substack{0 \leq k \leq n+1 \\ k \neq j}} \frac{x - x_k}{x_j - x_k}, \quad j = 1, \ldots, n+1.$$

An example of Lagrange basis is shown in Figure 1.2. Note that each basis function ℓ_j takes the value of 1 at the j-th interpolation points and vanishes at all other interpolation points.

Thanks to the interpolation property of the Lagrange basis (1.3), we can construct the interpolant for data points (x_i, y_i), $i = 1, \ldots, n+1$, in a straightforward manner:

$$p_n(x) = \sum_{k=0}^{n} y_k \ell_k(x). \tag{1.4}$$

We readily verify that $p_n(x_i) = \sum_{k=0}^{n} y_k \ell_k(x_i) = y_i$, since $\ell_k(x_i) = 0$ for all $k \neq i$.

We note that, for a given degree n and data points (x_i, y_i), $i = 1, \ldots, n+1$, the polynomial interpolant constructed using the Lagrange basis polynomials is identical to that constructed using the Vandermonde method. This is because a polynomial interpolant depends only on the polynomial degree n and the location of the interpolation points; it does not depend the particular basis

functions — monomial basis (for Vandermonde's method) or Lagrange basis — used to represent the polynomial.

Cost. Thanks to the interpolation property (1.3), we can directly construct the interpolant without forming and inverting the Vandermonde matrix. The computation of Lagrange basis at a point requires $2n$ subtractions and n divisions for the total cost of $\mathcal{O}(n)$. Since we must repeat the procedure for each of the $n+1$ basis functions to evaluate (1.4), the total cost is $\mathcal{O}(n^2)$.

1.5 Error analysis

We have introduced a few different approaches to construct and evaluate polynomial interpolants. We also know the associated computational cost. We now wish to assess the accuracy of our polynomial interpolant. The following theorem provides an answer to the question:

Theorem 1.1. Suppose the polynomial interpolant p_n interpolates a smooth function f at interpolation points $a = x_1 < x_2 < \cdots < x_{n+1} = b$. Then the error in the polynomial interpolant is bounded from above by

$$|f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \max_{s\in[a,b]} |f^{(n+1)}(s)|(b-a)^{n+1} \quad \forall x \in [a,b]. \tag{1.5}$$

Proof. We sketch the proof for $n = 2$ for notational simplicity; the proof for an arbitrary n follows the same argument. We first introduce an auxiliary function

$$g_2(s) = f(s) - p_2(s) - \left(\frac{f(x) - p_2(x)}{(x-x_1)(x-x_2)(x-x_3)}\right)(s-x_1)(s-x_2)(s-x_3).$$

We note that $g_2(x) = 0$ by construction. We also note that $g_2(x_i) = 0$, $i = 1, 2, 3$, because the polynomial p_2 interpolates f at these points. Hence, the function g has at least four roots over $[a, b]$. By Rolle's theorem, the function $g_2' = g_2^{(1)}$ has at least three roots over $[a, b]$. Invoking Rolle's theorem two more times, we deduce that $g_2^{(3)}$ has at least one root over $[a, b]$. Let ξ be one of these points: $g_2^{(3)}(\xi) = 0$. We now compute the third derivative of g_2:

$$g_2^{(3)}(s) = f^{(3)}(s) - \left(\frac{f(x) - p_2(x)}{(x-x_1)(x-x_2)(x-x_3)}\right) \cdot 3!.$$

Note that $\frac{d^3}{ds^3} p_2(s) = 0$ since p_2 is of degree 2. We now evaluate the expression at ξ to obtain

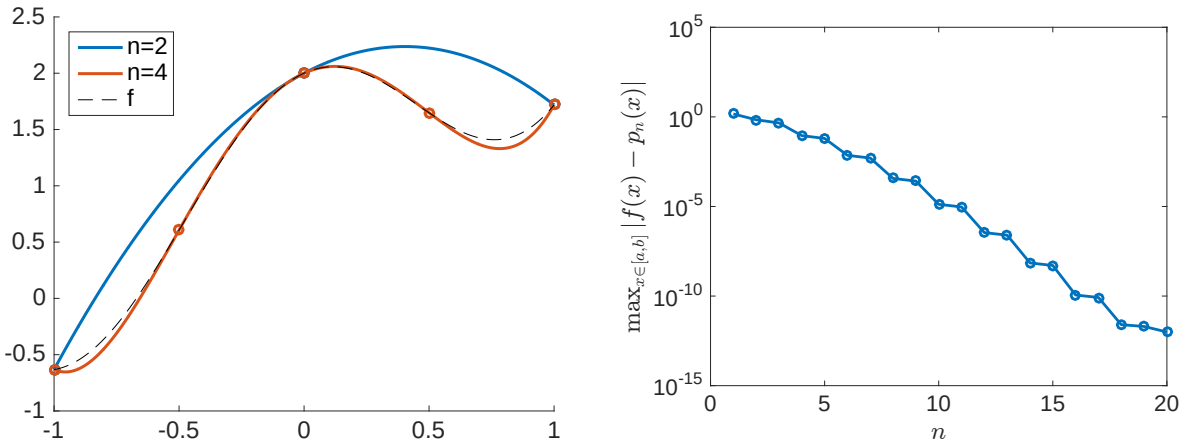
$$0 = f^{(3)}(\xi) - \left(\frac{f(x) - p_2(x)}{(x-x_1)(x-x_2)(x-x_3)}\right) \cdot 3!,$$

or, equivalently,

$$f(x) - p_2(x) = \frac{1}{3!} f^{(3)}(\xi)(x-x_1)(x-x_2)(x-x_3). \tag{1.6}$$

We finally note that $|f^{(3)}(\xi)| \leq \max_{s\in[a,b]} |f^{(3)}(s)|$ and $|x - x_i| \leq (b-a)$, $i = 1, 2, 3$, which yield the desired result for $n = 2$. The proof for a general n follows exactly the same procedure. □

We make a few key observations:



(a) interpolants

(b) interpolation error

Figure 1.3: Interpolation of $f(x) = \exp(x) + \cos(\pi x)$ using equispaced points. (Note: the maximum over $[a, b]$ is approximated using 1001 equidsitbuted points.)

1. The interpolation error depends on the $n+1$-st derivative of the underlying function, $f^{(n+1)}$. The smaller the $n+1$-st derivative, the more accurate the interpolant.

2. The interpolation error depends on the degree n of the interpolant. *If* $|f^{(n+1)}|$ grows slower than $(n+1)!$, *then* the interpolation error decreases exponentially with n.

3. As a special case, if the underlying function is a polynomial of degree $m \leq n$, then the interpolation error is zero everywhere (since $f^{(n+1)}(x) = 0$ for a degree $m \leq n$ polynomial). In other words, the interpolant exactly reproduces the underlying function.

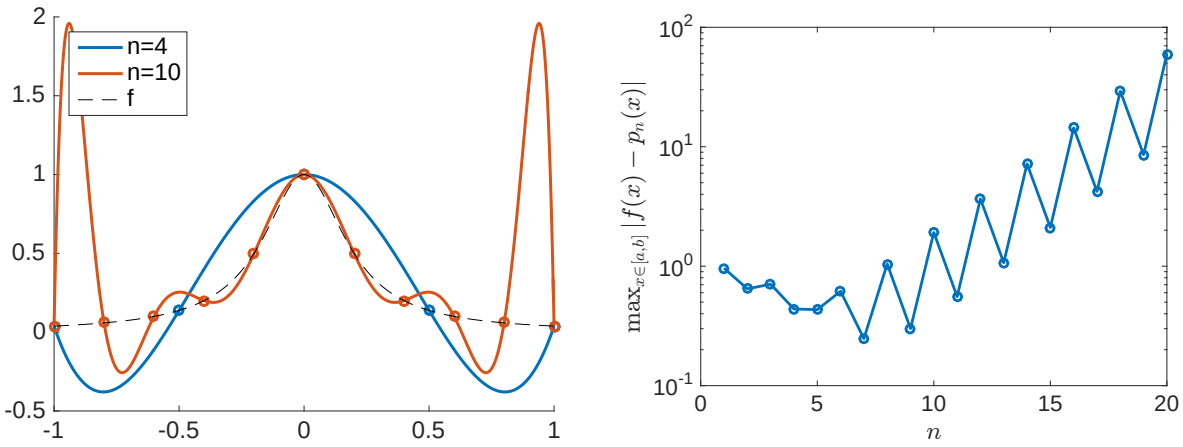
An example of interpolating the function $f(x) = \exp(x) + \cos(\pi x)$ is shown in Figure 1.3(a). Here, all the higher derivatives of the functions over $[-1, 1]$ are bounded by $|f^{(n)}(x)| \leq 1 + \pi^n$. Hence, we observe that

$$|f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \max_{s\in[-1,1]} |f^{(n+1)}(x)|(2)^{n+1} \leq \frac{(1+\pi^{n+1})2^{n+1}}{(n+1)!},$$

which decays exponentially with n. We indeed observe this exponential convergence in Figure 1.3(b). The exponential convergence is very powerful: we achieve a very small interpolation error of $\mathcal{O}(10^{-10})$ using just 18 interpolation points.

1.6 Runge's phenomena

Unfortunately, for some functions, increasing the polynomial degree p does not necessarily result in a smaller error. One such function is the *Runge function*, $f(x) = \frac{1}{1+25x^2}$. An example of interpolating the Runge function using a degree 10 polynomial is shown in Figure 1.4(a). For an interpolant based on equispaced points, we observe very large oscillation in the interpolant near the endpoints. Figure 1.4(b) shows that the interpolants with equispaced points do not converge to the to the Runge function as the degree increases; in fact the interpolation error increases



(a) interpolants

(b) interpolation error

Figure 1.4: Interpolation of $f(x) = \frac{1}{1+25x^2}$ using equispaced interpolation points.

exponentially with n. This increasingly oscillatory behavior of the interpolant with n is called the *Runge's phenomenon*.

This is a very undesirable result. However, this lack of convergence is in fact predicted by our interpolation error bound, Theorem 1.1: the maximum value of the higher derivatives, $\max_{s\in[-1,1]} |f^n(s)|$, increases very rapidly for the Runge function with n.

1.7 Selection of interpolation points: Chebyshev nodes

One approach to overcome the Runge's phenomenon is to consider a use of different interpolation points. For example, instead of equispaced points, we can use the *Chebyshev nodes*, given by

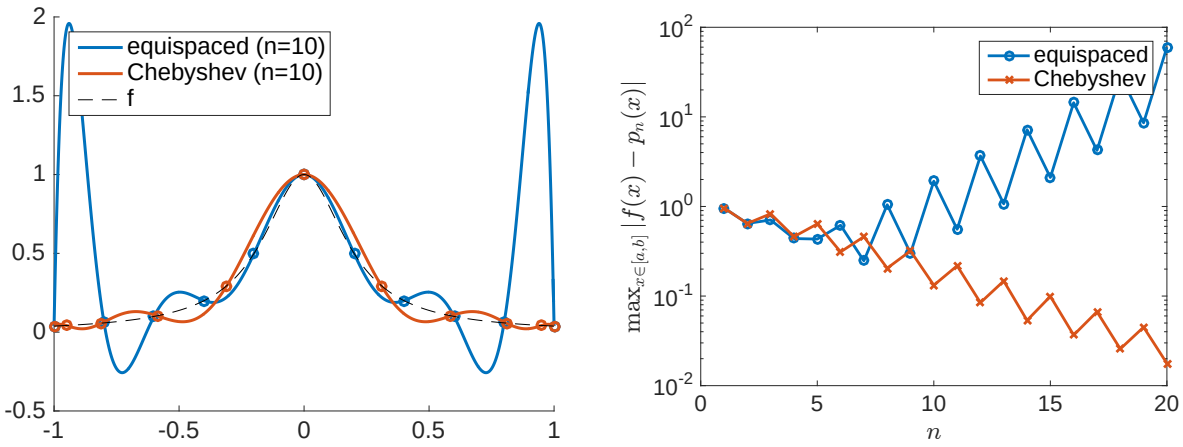
$$x_i = -\cos\left(\frac{i-1}{n}\pi\right), \quad i = 1, \ldots, n+1,$$

as the interpolation points. Note that these interpolation points are more clustered toward the endpoints. The points produce a more stable interpolant, as shown pictorially in Figure 1.5(a). Figure 1.5(b) also confirms that the interpolation error decreases with the polynomial degree n and in fact the error converges expoentially with n, albeit much less rapidly than for $f(x) = \exp(x)+\cos(\pi x)$ shown in Figure 1.3(b). (The above Chebyshev nodes are also called the *Chebyshev points of the second kind* or *Chebyshev-Lobatto points*; see the Appendix for details.)

To understand why Chebyshev nodes produce a more accurate interpolant than the equispaced nodes, we take a closer look at (a generalization of) the error expression (1.6) which arises in the proof of Theorem 1.1. We can show that for any $x \in [a, b]$ there exists $\xi \in [a, b]$ such that

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \prod_{j=1}^{n+1} (x - x_j).$$

Given that we in general do not know $f^{(n+1)}$, the best we can do to minimize the error is to choose



(a) interpolants

(b) interpolation error

Figure 1.5: Interpolation of $f(x) = \frac{1}{1+25x^2}$ using equispaced and Chebyshev interpolation points.

the node locations x_j, $j = 1, \ldots, n+1$, that minimize

$$\max_{x\in[a,b]} |\prod_{j=1}^{n+1}(x - x_j)|.$$

The Chebyshev nodes nearly minimizes this function. Hence, the upper bound of the interpolation error for Chebyshev nodes is smaller than that for equispaced nodes. (Note however that this is an *upper (i.e. worst-case) bound*; it is possible for equispaced-nodes to yield a smaller error than Chebyshev nodes for a specific f.)

In cases where we can pick the location of the data points (x_i, y_i), choosing Chebyshev nodes produces much more robust interpolant than using equispaced points. However, if the data is given to us, then we do not have the option to pick the interpolation points. In the following lecture, we consider a different strategies to interpolate the data points such that we can provide a convergent approximation for a wide range of underlying functions.

1.8 Summary

We summarize the key points of the lecture:

1. Given a set of $n+1$ data points, one approach to approximate the value at an intermediate point is by polynomial interpolation.
2. A polynomial interpolant is uniquely determined by the polynomial degree and the location of interpolation points.
3. One approach to systematically construct an interpolant is to use the Vandermonde's method.
4. The cost of finding the polynomial coefficients by the Vandermonde's method is $\mathcal{O}(n^3)$.
5. Another approach to systematically construct an interpolant is to use the Lagrange basis.

6. The error associated with a polynomial interpolant depends on the degree n of the interpolant and the $n+1$-st derivative of the underlying function.

7. The error decreases exponentially with n for functions with well-behaved higher derivatives.

8. If the higher derivatives of the underlying function increase very rapidly, then the the interpolant may exhibit oscillatory behavior and may not converge with n. This is called the Runge's phenomenon.

9. One way to overcome the Runge's phenomenon is to use interpolations points that are clustered towards the endpoints, for instance Chebyshev nodes.

1.9 Appendix: Chebyshev polynomial interpolation

In Section 1.7, we introduced the Chebyshev points (of the second kind),

$$x_j = -\cos\left(\frac{i-1}{n}\pi\right), \quad i = 1, \ldots, n+1,$$

which include the endpoints $\{-1, 1\}$. The closely related Chebyshev points of the first kind are given by

$$x_j = \cos\left(\frac{2i-1}{2(n+1)}\pi\right), \quad i = 1, \ldots, n+1,$$

which do not include the endpoints. These points are closely related to orthogonal polynomials called *Chebyshev polynomials*; the Chebyshev nodes of the first and second kind are the extrema and zeros, respectively, of a Chebyshev polynomial. It can be shown that, for most continuous functions f (or more precisely f for which the total variation of the first derivative is bounded), polynomial interpolant based on Chebyshev nodes converge: i.e., $\|f - p_n\|_\infty \to 0$ as $n \to \infty$. In addition, for the Chebyshev points of the first kind, it can be shown that the points minimizes $\max_{x\in[a,b]} |\prod_{j=1}^{n+1}(x-x_j)|$ and the quantity is bounded by $|\prod_{j=1}^{n+1}(x-x_j)| \le \frac{1}{2^n}\left(\frac{b-a}{2}\right)^{n+1}$, $\forall x \in [a,b]$. As a result, the error bound (1.5) sharpens for Chebyshev points of the first kind to

$$|f(x) - p_n(x)| \le \frac{1}{2^n(n+1)!} \max_{s\in[a,b]} |f^{(n+1)}(s)| \left(\frac{b-a}{2}\right)^{n+1} \qquad \forall x \in [a,b].$$

The proof of convergence relies on the properties of the Chebyshev polynomials and are beyond the scope of this course; we refer to *Approximation Theory and Approximation Practice* by Trefethen and *Numerical Analysis* by Burden and Faires for details.

Lecture 2

Piecewise polynomial interpolation

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

2.1 Motivation

We observed in the previous lecture that polynomial interpolation is an effective tool to approximate a function from a finite set of data points. However, we also observed that increasing the polynomial degree does not necessarily improve the accuracy for equispaced data points, and the interpolant might exhibit undesirable oscillatory behaviors (i.e. the Runge's phenomenon). We will introduce in this lecture a technique to improve the accuracy and robustness of interpolation based on equispaced points.

2.2 Piecewise linear interpolation

One approach to overcome the Runge's phenomenon is to consider piecewise interpolation of data points and construct a *piecewise polynomial interpolant*. We first consider *piecewise linear interpolation*. Here, we consider $n+1$ equispaced data points (x_i, y_i), $i = 1, \dots, n+1$, over $[a, b]$. Without loss of generality, we assume the points are ordered such that $a = x_1 < x_2 < \cdots < x_{n+1} = b$. We then introduce $N = n$ segments $S_k \equiv [x_k, x_{k+1}]$, $k = 1, \dots, N$, delineated by the $n+1$ points. We will denote the length of each segment by h: $h \equiv x_{i+1} - x_i = (b-a)/n$. We then construct a degree 1 polynomial interpolant over each segment S_k:

$$p_{h,1}^{(k)}(x) = a_0^{(k)} + a_1^{(k)} x, \quad x \in S_k \equiv [x_k, x_{k+1}],$$

such that

$$p_{h,1}^{(k)}(x_k) = y_k \quad \text{and} \quad p_1^{(k)}(x_{k+1}) = y_{k+1}.$$

Here, the first subscript, h, indicates the length of the segment, and the second subscript indicates the degree of the polynomial for each segment. An explicit expression for the interpolant in terms of the data (x_k, y_k) and (x_{k+1}, y_{k+1}) is given by

$$p_{h,1}^{(k)}(x) = y_k + \frac{y_{k+1} - y_k}{x_{k+1} - x_k}(x - x_k), \quad k = 1, \dots, N. \tag{2.1}$$

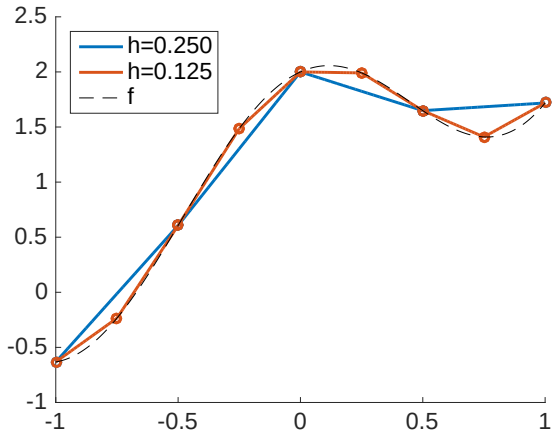


Figure 2.1: Piecewise linear interpolation of $\exp(x) + \cos(\pi x)$.

We readily verify that the interpolation conditions are satisfied: $p_{h,1}^{(k)}(x_k) = y_k$ and $p_{h,1}^{(k)}(x_{k+1}) = y_{k+1}$. Combining the segment-wise interpolant, our global interpolant is given by

$$p_{h,1}(x) = \begin{cases} p_{h,1}^{(1)}(x), & x \in S_1 \\ p_{h,1}^{(2)}(x), & x \in S_2 \\ \quad \vdots & \\ p_{h,1}^{(N)}(x), & x_n \in S_N. \end{cases}$$

An example of piecewise linear interpolation is shown in Figure 2.1. The interpolant simply estimate the value between any two data points by linear interpolation. Note in particular that the value of the interpolant at a point $x \in S_k$ depends only on the data at x_k and x_{k+1}.

Cost. In order to evaluate the interpolant for an arbitrary $x \in [a, b]$, we must first identify the segment k in which x lies in and then evaluate the expression $p_1^{(k)}(x)$. Since the segments are sorted, we can find the appropriate segment in $\mathcal{O}(\log_2(N))$ operations. The evaluation of the interpolant by (2.1) requires $\mathcal{O}(1)$ operations; i.e. the number of operations is independent of the number of interpolation points.

2.3 General degree-p piecewise interpolation

More generally, we can construct piecewise polynomial interpolant based on general degree p interpolant over each segment. We recall that construction of a degree p interpolant requires $p + 1$ interpolation points. Hence, to construct N interpolants we require $Np + 1$ data points, assuming each endpoint is shared by the two abutting segments. Then, for each segment k, we introduce an interpolant

$$p_{h,p}^{(k)}(x) = \sum_{j=0}^{p} a_j^{(k)} x^j, \quad x \in S_k$$

such that

$$p_{h,p}^{(k)}(x_i^{(k)}) = y_i^{(k)}, \quad i = 1, \ldots, p+1;$$

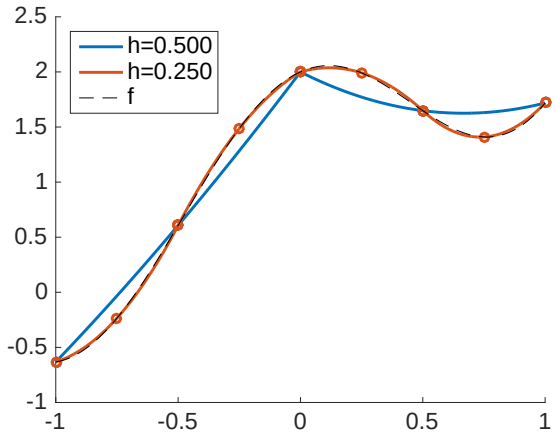


Figure 2.2: Piecewise quadratic interpolation of $\exp(x) + \cos(\pi x)$.

here $x_i^{(k)}$ is the i-th interpolation point in the k-th segment, the first subscript, h, indicates the length of the segment, and the second subscript, p, indicates the polynomial degree. As in the linear case, we combine the local interpolants to yield a global interpolant:

$$p_{h,p}(x) = \begin{cases} p_{h,p}^{(1)}(x), & x \in S_1 \\ p_{h,p}^{(2)}(x), & x \in S_2 \\ \vdots & \\ p_{h,p}^{(N)}(x), & x_n \in S_N. \end{cases}$$

Figure 2.2 shows an example of piecewise quadratic interpolant ($p = 2$); we note that the quadratic interpolant appears more accurate than the piecewise linear interpolant for this particular case for a same number of data points; we will soon confirm that this indeed is the case.

The interpolant over each segment may be constructed using, for instance, the Vandermonde's method or the Lagrange basis. The global, piecewise-polynomial interpolant is then given by combining the polynomial interpolant over each segment.

Cost. As in the case of piecewise linear interpolants, to evaluate the piecewise interpolant at an arbitrary x, we must first identify the segment k in which x lies and then construct and evaluate the interpolant over the segment. For a piecewise interpolant with N segments, the identification of the segment requires $\mathcal{O}(\log_2(N))$ operations. As discussed in the previous lecture, if we use the Vandermonde's method, then the construction requires $\mathcal{O}(p^3)$ operations and evaluation requires $\mathcal{O}(p)$ operations. If we use the Lagrange basis, then the evaluation requires $\mathcal{O}(p^2)$ operations.

2.4 Piecewise polynomial interpolation: error analysis and convergence rate

We now wish to assess the accuracy of our piecewise polynomial interpolants. Towards this end, we apply the interpolation error bound obtained in the previous lecture to individual segment.

Theorem 2.1. Suppose we interpolate a smooth function over $[a, b]$ using a piecewise polynomial interpolant with N segments. Suppose the segments are of the equal length $h \equiv (b - a)/N$ and

in each segment we employ a degree p polynomial. Then the error in the piecewise polynomial interpolant is bounded from above by

$$|f(x) - p_{h,p}(x)| \leq \frac{1}{(p+1)!} \max_{s\in[a,b]} |f^{(p+1)}(s)|h^{p+1}, \quad \forall x \in [a,b]. \tag{2.2}$$

Proof. Proof is an application of the error bound for global interpolants to each segment of the piecewise interpolants. We first note that the error over segment k is bounded from above by, for any $x \in S_k$,

$$|f(x) - p_{h,p}(x)| \leq \frac{1}{(p+1)!} \max_{s\in S_k} |f^{(p+1)}(s)|h^{p+1}.$$

We then note that the maximum interpolation error over $[a,b]$ is the largest of the individual-segment interpolation errors. Hence, for any $x \in [a,b]$,

$$|f(x) - p_{h,p}(x)| \leq \max_{1\leq k\leq N} \frac{1}{(p+1)!} \max_{s\in S_k} |f^{(p+1)}(s)|h^{n+1} = \frac{1}{(p+1)!} \max_{s\in[a,b]} |f^{(p+1)}(s)|h^{p+1},$$

which is the desired result. □

We make a few key observations:

1. The interpolation error depends on the $n+1$-st derivative of the underlying function, $f^{(n+1)}$.
2. The interpolation error depends on the degree p *and* the segment length h.
3. For a fixed segment length h, if $|f^{(p+1)}|$ grows slower than $(p+1)!$, then the interpolation error decreases exponentially with the degree p.
4. For a fixed polynomial degree p, the interpolation error decreases as h^{p+1} with the segment length h.

As in the global polynomial interpolant considered in the previous lecture, the interpolation error depends on the $p+1$-st derivative of the underlying function. However, we now have two means to control the error: increasing the degree *or* decreasing the segment length $h \equiv 1/N$. When we decrease h, the error converges as h^{p+1}; in words, a higher degree interpolant converges more rapidly to the true solution as h decreases. The order at which the error decreases with h — in this case $p+1$ — is called the *convergence rate*.

Figure 2.3 shows the convergence of the piecewise lienar, quadratic, and cubic interpolants applied to $f(x) = \exp(x) + \cos(\pi x)$. Note that the convergence rate of a given method can be empirically deduced from the log-lot plot of error vs h. Specifically, taking the logarithm of the error expression (2.2) yields

$$\underbrace{\log(|f(x) - p_{h,p}(x)|)}_{\text{error}} \leq \underbrace{\log(C)}_{\text{intercept}} + \underbrace{(p+1)}_{\text{slope}} \log(h)$$

for $C \equiv \frac{1}{(p+1)!} \max_{s\in[a,b]} |f^{(p+1)}(s)|$. Hence, the convergence rate is equal to the slope of the log-log plot, which, for piecewise polynomial interpolants, is $p+1$. We confirm in Figure 2.3 that the log-log plots of the linear, quadratic, and cubic interpolant have the slope of approximately 2, 3, and 4, respectively.

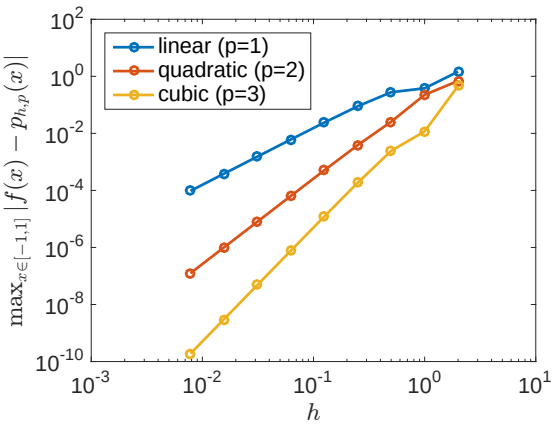


Figure 2.3: Convergence with h of the piecewise linear, quadratic, and cubic interpolants applied to $\exp(x) + \cos(\pi x)$ over $[-1, 1]$.

We also note that the piecewise polynomial interpolants are not as efficient as global polynomial interpolants for this smooth function. In the previous lecture, the global interpolant achieved the error of $\mathcal{O}(10^{-10})$ using just 18 data points. Achieving the same error level of $\mathcal{O}(10^{-10})$ using piecewise polynomial interpolation, even for the piecewise cubic interpolation, requires ≈ 500 data points. Hence, for (very) smooth functions, global interpolants are more efficient.

Runge's phenomenon. We now apply piecewise polynomial interpolant to Runge's function, for which the global high-order interpolants perform poorly. Figure 2.4(a) shows an example of piecewise quadratic ($p = 2$) interpolation applied to the Runge's function. We observe that the piecewise interpolants do not exhibit the oscillation associated with global high-order interpolant based on equispaced points. Figure 2.4(b) confirms that the error decay as the number of datapoints is increased for piecewise linear, quadratic, and cubic interpolants. Thus, piecewise polynomial interpolant provides more robust approximation even for functions, such as the Runge function, whose higher derivatives are ill-behaved.

2.5 Spline interpolation: cubic spline

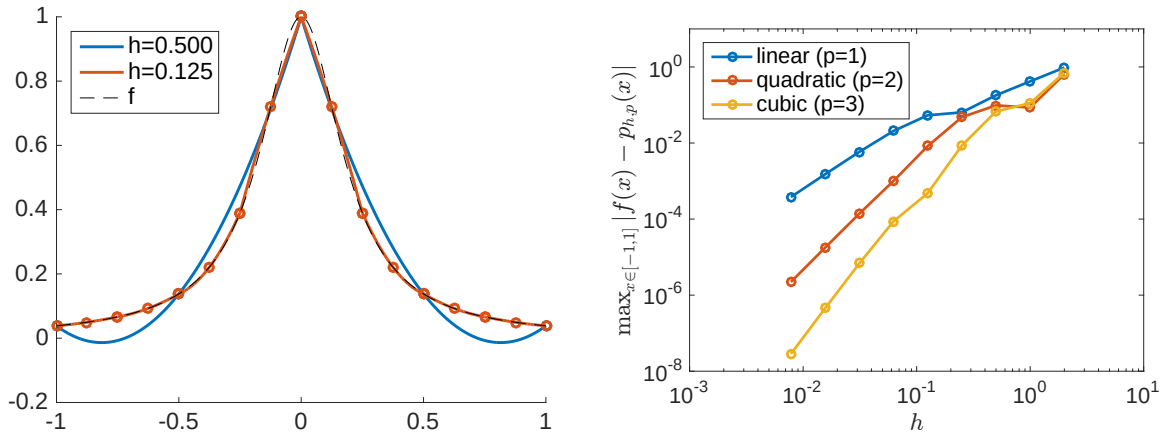
Spline interpolation is a variant of piecewise polynomial interpolation with additional constraints on the smoothness of the interpolant. We here introduce arguably the most well-known spline interpolation: *cubic spline interpolation*.

As in the case of piecewise polynomial interpolation, we first subdivide the domain $[a, b]$ into N segments deliniated by $N + 1$ equispaced interpolation points, $a = x_1 < x_2 < \cdots < x_{N+1} = b$. A cubic spline is a piecewise cubic polynomial

$$s_i(x) = a_i + b_i(x - x_i)^1 + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad i = 1, \ldots, N, \tag{2.3}$$

such that the values at the endpoints match the underlying function,

$$s_i(x_i) = y_i, \quad i = 1, \ldots, N \quad \text{(left value)}, \tag{2.4}$$
$$s_i(x_{i+1}) = y_{i+1}, \quad i = 1, \ldots, N \quad \text{(right value)}, \tag{2.5}$$



(a) quadratic interpolation

(b) error convergence

Figure 2.4: Piecewise polynomial interpolation applied to the Runge function $f(x) = 1/(1+25x^2)$ over $[-1, 1]$.

and the first and second derivatives are continuous across segments,

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \quad i = 1, \ldots, N-1, \quad \text{(first derivative)}, \tag{2.6}$$
$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}), \quad i = 1, \ldots, N-1, \quad \text{(second derivative)}. \tag{2.7}$$

Note that we use a polynomial representation shifted by x_i (instead of monomials) to simplify the identification of the coefficients later. Combining the local splines, our global cubic spline is given by

$$s(x) = \begin{cases} s_1(x), & x \in S_1 \\ s_2(x), & x \in S_2 \\ \vdots & \\ s_N(x), & x \in S_N. \end{cases}$$

We now compare the number of degrees of freedom and the number of constraints. We have $4N$ degrees of freedom because we have N segments and four degrees of freedom per segment. We have N left value constraints, N right value constraints, $N-1$ first-derivative continuity constraints, and $N-1$ second-derivative continuity constraints; the total number of constraints is $4N-2$. (Note that the number of first- and second-derivative are equal to the number of segment interfaces, which is one less than the number of segments.) We hence need two additional constraints to specify a unique cubic spline.

There are a number of different constraints we could introduce. A few common examples are the following:

- **Complete spline.** We specify the first derivative at the endpoints of the spline:

  $$s_1'(x_1) = g_L \quad \text{and} \quad s_N'(x_{N+1}) = g_R,$$

  where g_L and g_R are the derivatives specified. We must know the first derivatives at the two endpoints to use this approach.

- **Natural spline.** We require the second derivative at the endpoints of the spline to vanish:

$$s_1''(x_1) = 0 \quad \text{and} \quad s_N''(x_{N+1}) = 0.$$

  In words, it sets the curvature of the spline to zero at the endpoints to zero.

- **Not-a-knot spline.** We require the *third* derivative at the first and last interfaces to be continuous:

$$s_1'''(x_2) = s_2'''(x_2) \quad \text{and} \quad s_{N-1}'''(x_N) = s_N'''(x_N).$$

Construction of a cubic spline. Due to the presence of the first- and second-derivative continuity conditions at the segment interfaces, the value of a cubic spline, unlike the value of a piecewise polynomial interpolant, depends not only on the two endpoints delineating the particular segment but on all of the $N+1$ data points. We will hence solve a system of (globally coupled) $N+1$ equations to find the polynomial coefficients. A convenient variables to solve for are the derivative of the spline at the interpolation points; we will denote the derivatives by g_i, $i = 1, \ldots, N+1$.

We first impose the value conditions at the endpoints, (2.4) and (2.5), to obtain

$$s_i(x_i) = a_i = y_i, \quad i = 1, \ldots, N, \tag{2.8}$$

$$s_i(x_{i+1}) = a_i + b_i h + c_i h^2 + d_i h^3 = y_{i+1}, \quad i = 1, \ldots, N. \tag{2.9}$$

We then set the first derivative at the endpoints to our independent variables, g_i, $i = 1, \ldots, N+1$:

$$s_i'(x_i) = b_i = g_i, \quad i = 1, \ldots, N, \tag{2.10}$$

$$s_i'(x_{i+1}) = b_i + 2c_i h + 3d_i h^2 = g_{i+1}, \quad i = 1, \ldots, N; \tag{2.11}$$

note that the first derivative continuity condition (2.6) is implicitly enforced. We also note the second derivative continuity condition (2.7):

$$s_i''(x_{i+1}) = 2c_i + 6d_i h = c_{i+1} = s_{i+1}''(x_{i+1}), \quad i = 1, \ldots, N-1. \tag{2.12}$$

Solving (2.8)–(2.11) for a_i, b_i, c_i, and d_i in terms of y_i and g_i, we obtain

$$a_i = y_i, \quad i = 1, \ldots, N, \tag{2.13}$$

$$b_i = g_i, \quad i = 1, \ldots, N, \tag{2.14}$$

$$c_i = \frac{3}{h^2}(y_{i+1} - y_i) - \frac{2}{h} g_i - \frac{1}{h} g_{i+1}, \quad i = 1, \ldots, N, \tag{2.15}$$

$$d_i = -\frac{2}{h^3}(y_{i+1} - y_i) + \frac{1}{h^2} g_i + \frac{1}{h^2} g_{i+1}., \quad i = 1, \ldots, N. \tag{2.16}$$

We now substitute the expressions for c_i and d_i into (2.7) to obtain

$$g_i + 4g_{i+1} + g_{i+2} = \frac{3}{h}(y_{i+2} - y_i), \quad i = 1, \ldots, N-1,$$

these are the $N-1$ equations associated with any cubic splines.

The two additional equations are associated with the specific end condition for the spline.

- **Complete spline.** For a complete spline, the conditions are

$$s_1'(x_1) = g_1 = g_L \quad \text{and} \quad s_N'(x_{N+1}) = g_{N+1} = g_R.$$

The resulting system of equations in the matrix form is

$$\begin{pmatrix} 1 & & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 & \\ & & & & & 1 & \end{pmatrix} \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_N \\ g_{N+1} \end{pmatrix} = \begin{pmatrix} g_L \\ 3(y_3 - y_1)/h \\ 3(y_4 - y_2)/h \\ \vdots \\ 3(y_{N+1} - y_{N-1})/h \\ g_R \end{pmatrix}. \tag{2.17}$$

- **Natural spline.** For a natural spline, the conditions are

$$s_1''(x_1) = 2c_1 = 0 \quad \text{and} \quad s_N''(x_{N+1}) = 2c_N + 6d_N h = 0.$$

The substitution of the expressions (2.15) and (2.16) yields

$$2g_1 + g_2 = \frac{3}{h}(y_2 - y_1) \quad \text{and} \quad g_N + g_{N+1} = \frac{3}{h}(y_{N+1} - y_N).$$

$$\begin{pmatrix} 2 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 2 & \end{pmatrix} \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_N \\ g_{N+1} \end{pmatrix} = \begin{pmatrix} 3(y_2 - y_1)/h \\ 3(y_3 - y_1)/h \\ 3(y_4 - y_2)/h \\ \vdots \\ 3(y_{N+1} - y_{N-1})/h \\ 3(y_{N+1} - y_N)/h \end{pmatrix}. \tag{2.18}$$

The linear system is well posed and has a unique solution.

- **Not-a-knot spline.** For a not-a-knot spline, the conditions are

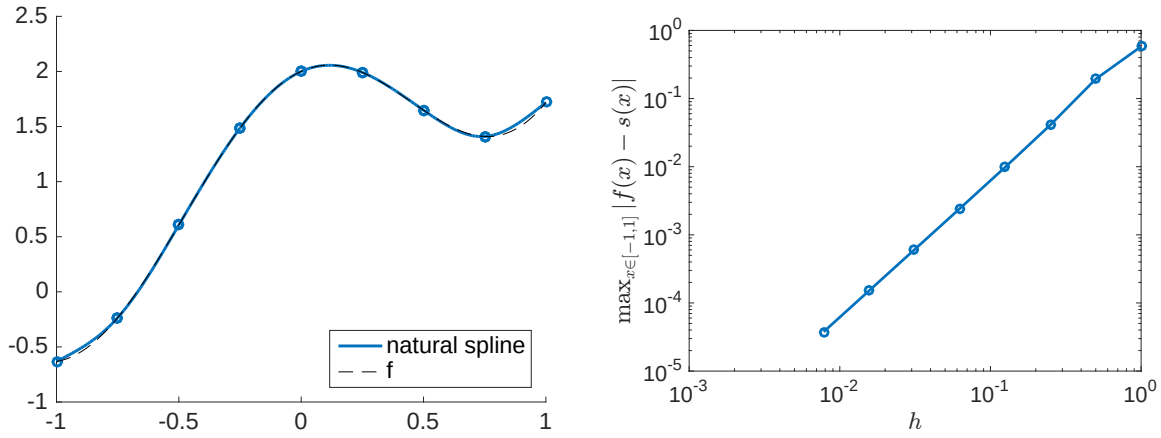
$$s_1'''(x_2) = 6d_1 = 6d_2 = s_2'''(x_2) \quad \text{and} \quad s_{N-1}'''(x_N) = 6d_{N-1} = 6d_N = s_N'''(x_N).$$

The substitution of (2.16) yields

$$g_1 - g_3 = \frac{2}{h}(-y_3 + 2y_2 - y_1) \quad \text{and} \quad g_{N-1} - g_{N+1} = \frac{2}{h}(-y_{N+1} + 2y_N - y_{N-1}).$$

The resulting linear system is

$$\begin{pmatrix} 1 & & -1 & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 & \\ & & & 1 & & -1 & \end{pmatrix} \begin{pmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_N \\ g_{N+1} \end{pmatrix} = \begin{pmatrix} 2(-y_3 + 2y_2 - y_1)/h \\ 3(y_3 - y_1)/h \\ 3(y_4 - y_2)/h \\ \vdots \\ 3(y_{N+1} - y_{N-1})/h \\ 2(-y_{N+1} + 2y_N - y_{N-1})/h \end{pmatrix}. \tag{2.19}$$



(a) $n = 8$ nautral spline

(b) error convergence

Figure 2.5: Natural cubic spline interpolation applied to $f(x) = \exp(x) + \cos(\pi x)$.

Each of the three $N + 1$-by-$N + 1$ linear systems associated with the three endpoint conditions are well-posed and has a unique solution. Thus, the construction (i.e. the identification of the polynomial coefficients) of a cubic spline is a two step procedure: we first assemble and solve a linear system (2.17), (2.18), or (2.19) associated with an appropriate additional constraints; we then evaluate the coefficients for each segment using (2.13)–(2.16). Then, to evaluate the interpolant at a given point x, we evaluate the cubic polynomial (2.3).

Examples. Figure 2.5(a) shows an example of a natural cubic spline applied to $f(x) = \exp(x)+\cos(\pi x)$. We observe estimates the underlying function well despite a small number of points used in approximation. (Compare, for instance, to the piecewise linear interpolation shown in Figure 2.1.) The error convergence plot shown in Figure 2.5(b) confirms that the error decays rapidly as we decrease the segment length.

Figure 2.6(a) shows an example of natural cubic spline applied to the Runge function. Despite using equispaced points, spline interpolants do not exhibit Runge's phenomenon. This is unlike the (global) polynomial interplant based on equispaced interpolation points.

2.6 Summary

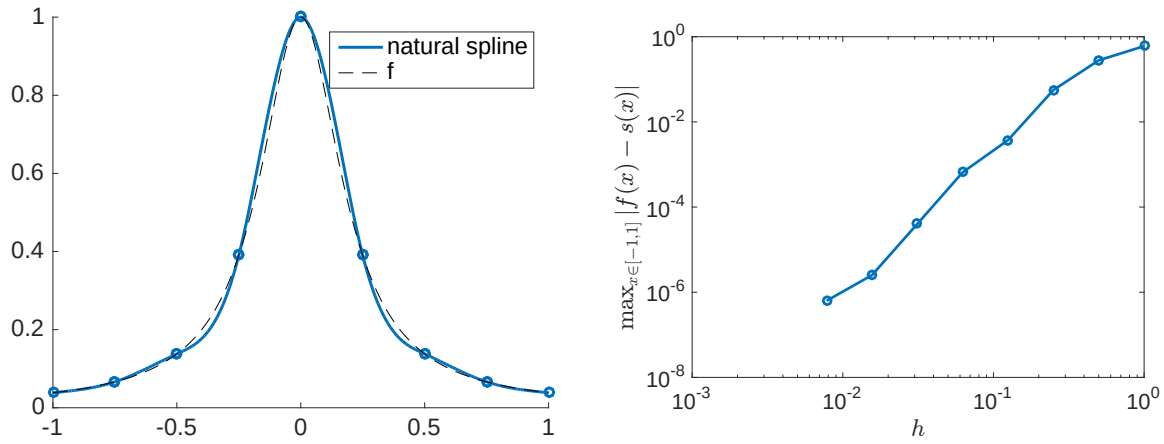
We summarize the key points of the lecture:

1. Piecewise polynomial interpolation divides the domain $[a, b]$ into N small segments and constructs polynomial interpolant in each segment.

2. Error of a piecewise polynomial interpolation depends on both the polynomial order and the segment length.

3. Piecewise polynomial interpolants based on a higher-degree polynomials converge at a higher rate as the segments are shortened.

4. The convergence rate of a given method can be deduced from the slope of the log-log plot of the error vs h.



(a) $n = 8$ natural spline

(b) error convergence

Figure 2.6: Natural cubic spline interpolation applied to the Runge function $f(x) = 1/(1 + 25x^2)$.

5. Cubic spline interpolation uses a cubic polynomial in each segment, but requires the continuity of the first- and second-derivative at the segment interface.

6. There are different types of cubic interpolants depending on the choice of the two additional constraints; common choices are complete spline, natural spline, and knot-a-not spline.

7. For spline interpolation, the derivative continuity conditions introduce coupling; as a result, the polynomial coefficients must be obtained by solving a global system.

8. Despite using equispaced data points, (low-order) piecewise polynomial interpolantion and spline interpolation do not suffer from the Runge's phenomenon.

Lecture 3

Numerical integration: Newton-Cotes rules

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

3.1 Motivation

In this lecture we consider numerical approximation of a definite integral

$$I = \int_a^b f(x)dx.$$

In particular, we consider the case where

1. we know f, but we cannot find an analytical expression for its antiderivative;
2. we can only evaluate f at a finite number of points.

A numerical approach to approximate the integral using a finite number evaluations of f is called *numerical integration* or *numerical quadrature*.

3.2 Newton-Cotes rules

We consider an approximation of the integral of the form

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^m w_i f(x_i);$$

here the points x_i, $i = 1, \dots, m$, are called *quadrature points*, the weights w_i, $i = 1, \dots, m$, are called *quadrature weights*, and the points and weights together define a *quadrature rule*.

Note: in some literature quadrature weights are normalized by the domain length, $b - a$; i.e. the quadrature rule is defined as $I \approx (b-a) \sum_{i=1}^m w_i f(x_i)$. In this note we follow the convention above without the normalization.

We can define various quadrature rules by considering different sets of quadrature points and weights. One approach to find a quadrature rule is to first approximate the integrand f by an interpolant p_n, then compute the integral of the interpolant, and finally identify the points and weights. This approach works well because i) the construction of the interpolant requires function values only at the interpolation points, and ii) polynomial can be integrated analytically.

The *m-point Newton-Cotes rule* is given by

$$Q_{\mathrm{NC}(m)} \equiv \int_a^b p_{m-1}(x)dx,$$

where $p_{m-1}(x)$ is the degree $m-1$ polynomial interpolant based on m equispaced interpolation points over $[a,b]$. This family of quadrature rules is called *Newton-Cotes rules*. More precisely, if both endpoints of $[a,b]$ are included as quadrature points, the rule is called a *closed Newton-Cotes rule*; if the endpoints are not included as quadrature points, the rule is called an *open Newton-Cotes rule*. We consider a few concrete (and popular) examples.

Midpoint rule. The midpoint rule is induced by the constant interpolant (i.e., polynomial of degree 0) with the midpoint $c \equiv (a+b)/2$ as the interpolation point. An explicit expression for the interpolant is given by

$$p_0(x) = f(c) = f\left(\frac{a+b}{2}\right).$$

Given our (very simple) approximation p_0 to f, we approximate the integral $I = \int_a^b f(x)dx$ by integrating p_0 over $[a,b]$:

$$I \approx Q_{\mathrm{midpoint}} \equiv \int_a^b f(c)dx = \underbrace{(b-a)}_{w_1} f(\underbrace{c}_{x_1}).$$

The midpoint rule is illustrated visually in Figure 3.1. The midpoint rule is a one-point rule with the quadrature point at $c = (a+b)/2$. The quadrature weight associated with the point c is $(b-a)$. Because the endpoints are not quadrature points, the midpoint rule is an open Newton-Cotes rule.

Trapezoid rule. Trapezoid rule is derived from the linear interpolant based on $m = 2$ interpolation points. We recall from the previous lecture that the linear interpolation for f over an interval $[a,b]$ can be expressed using Lagrange basis polynomials as

$$p_1(x) = \ell_1(x)f(a) + \ell_2 f(b) = \frac{x-b}{a-b}f(a) + \frac{x-a}{b-a}f(b).$$

Our approximation $Q_{\mathrm{NC}(2)}$ to $I = \int_a^b f(x)dx$ is given by the (exact) integration of $p_1(x)$ over $[a,b]$:

$$\begin{aligned} Q_{\mathrm{NC}(2)} &= \int_a^b p_1(x)dx = \int_a^b [\frac{x-b}{a-b}f(a) + \frac{x-a}{b-a}f(b)]dx \\ &= \left(\int_a^b \frac{x-b}{a-b}dx\right) f(a) + \left(\int_a^b \frac{x-a}{b-a}dx\right) f(b) = (b-a)\left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right). \end{aligned}$$

Hence, the trapezoid rule (or the 2-point closed Newton-Cotes rule) is given by

$$Q_{\mathrm{NC}(2)} = (b-a)\left(\frac{1}{2}f(a) + \frac{1}{2}f(b)\right).$$

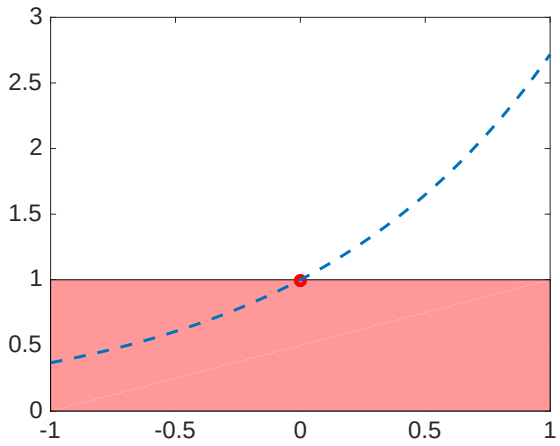


Figure 3.1: Midpoint rule applied to $I = \int_{-1}^{1} \exp(x)dx$. $I \approx 2.35$, $Q_{\text{midpoint}} = 2.00$.

The trapezoid rule is illustrated visually in Figure 3.2(a). We recognize that, for the trapezoid rule, the quadrature points are $x_1 = a$ and $x_2 = b$, and the quadrature weights are $w_1 = (b-a)/2$ and $w_2 = (b-a)/2$. In fact, we recognize that we can systematically find the quadrature weights from the Lagrange basis polynomials:

$$w_i = \int_a^b \ell_i(x)dx = \frac{1}{2}(b-a), \quad i = 1, 2.$$

This approach to identify the weights works for any Newton-Cotes rules. Because both endpoints a and b are quadrature points, the trapezoid rule is a closed Newton-Cotes rule.

Simpson's rule. Simpson's rule is induced by the quadratic interpolant based on $m = 3$ equispaced interpolation points, including the endpoints a and b. We will denote the midpoint by c: i.e., $c \equiv (a+b)/2$. The quadratic interpolant through the points a, c, and b is given by

$$\begin{aligned} p_2(x) &= \ell_1(x)f(a) + \ell_2(x)f(c) + \ell_3(x)f(b) \\ &= \frac{(x-b)(x-c)}{(a-b)(a-c)}f(a) + \frac{(x-a)(x-b)}{(c-a)(c-b)}f(c) + \frac{(x-a)(x-c)}{(b-a)(b-c)}f(b). \end{aligned}$$

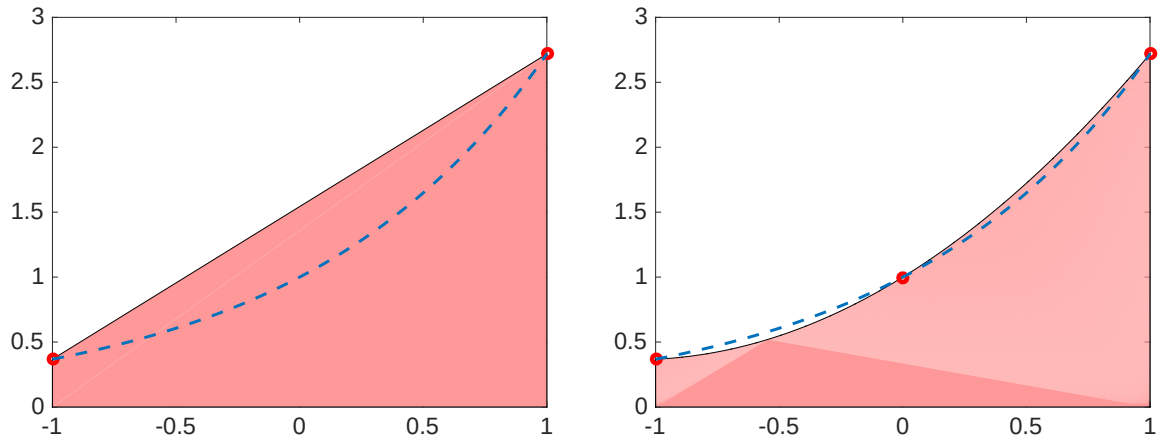
Our approximation Q to $I = \int_a^b f(x)dx$ is given by the integration of $p_2(x)$ over $[a, b]$. As in the case of the trapezoid rule, we can systematically identify the quadrature weights for the Simpson's rule by integrating the Lagrange basis polynomials:

$$w_1 = \int_a^b \ell_1(x)dx = \frac{1}{6}(b-a), \qquad w_2 = \int_a^b \ell_2(x)dx = \frac{4}{6}(b-a), \qquad w_3 = \int_a^b \ell_3(x)dx = \frac{1}{6}(b-a).$$

Hence the Simpson's rule (or the 3-point closed Newton-Cotes rule) is given by

$$Q_{\text{NC(3)}} = \int_a^b p_2(x)dx = (b-a)\left(\frac{1}{6}f(a) + \frac{4}{6}f(c) + \frac{1}{6}f(b)\right).$$

The Simpson's rule is illustrated visually in Figure 3.2(b).



(a) trapezoid rule

(b) Simpson's rule

Figure 3.2: The trapezoid rule and Simpson's rule applied to $I = \int_{-1}^{1} \exp(x)dx$.

General m-point closed Newton-Cotes rule. Using the same recipe as those used to construct the trapezoid rule and the Simpson's rule, m-point Newton cotes rule is given by

$$Q_{\mathrm{NC}(m)} = \int_a^b p_{m-1}(x)dx = \sum_{i=1}^{m} w_i f(x_i),$$

where the quadrature points x_i, $i = 1, \ldots, m$, are the m equispaced points over $[a, b]$,

$$x_i = a + \frac{b-a}{m-1}(i-1), \quad i = 1, \ldots, m,$$

and the quadrature weights w_i, $i = 1, \ldots, m$, are given by the integration of the associated Lagrange basis polynomials,

$$w_i = \int_a^b \ell_i(x)dx, \quad i = 1, \ldots, m.$$

Using the procedure, we can construct Newton-Cotes rules of an arbitrary degree in a systematic manner.

3.3 Newton-Cotes rules: error analysis

We now analyze the error associated with Newton-Cotes rules. We provide a full derivation for the midpoint rule, and then state the main results for general cases.

Midpoint rule. The following theorem summarizes the error associated with the midpoint rule:

Theorem 3.1. The integration error associated with the midpoint rule for the integral $I \equiv \int_a^b f(x)dx$ is bounded by

$$|I - Q_{\mathrm{midpoint}}| \leq \frac{1}{24} \max_{s \in [a,b]} |f''(s)|(b-a)^3.$$

Proof. We first recall the integral form of the Taylor series expansion

$$f(x) = f(c) + (x-c)f'(c) - \int_c^x (s-x)f''(s)ds.$$

We now evaluate directly the integration error:

$$\begin{aligned}
|I - Q_{\text{midpoint}}| &= |\int_a^b f(x)dx - (b-a)f(c)| \\
&= |\int_a^b f(c)dx + \int_a^b (x-c)f'(c)dx - \int_a^b \int_c^x (s-x)f''(s)dsdx - (b-a)f(c)| \\
&= |(\int_a^b f(c)dx - (b-a)f(c)) + f'(c)\int_a^b (x-c)dx - \int_a^b \int_c^x (s-x)f''(s)dsdx| \\
&= |\int_a^b \int_c^x (s-x)f''(s)dsdx| \leq \max_{s\in[a,b]} |f''(s)| |\int_a^b \int_c^x (s-x)dsdx| \\
&= \max_{s\in[a,b]} |f''(s)| \frac{1}{24}(b-a)^3,
\end{aligned}$$

which is the desired result. □

We make a few key observations:

1. The error depends on the second derivative of the underlying function. If the integrand is a linear function, then the midpoint rule is exact.
2. The error depends on the third power of the length of the interval. The shorter the interval, the more accurate the estimate.

General Newton-Cotes rules. We now analyze the error associated with general Newton-Cotes rules. We in particular consider a simplified analysis that builds on the interpolation error results we obtained in the previous chapters:

$$|f(x) - p_{m-1}(x)| \leq \frac{1}{m!} \max_{s\in[a,b]} |f^{(m)}(s)|(b-a)^m, \quad \forall x \in [a,b].$$

Using the interpolation error bound, we can construct an error bound for closed Newton-Cotes rules in a straightforward manner:

$$\begin{aligned}
|I - Q_{\text{NC}(m)}| &= |\int_a^b (f(x) - p_{m-1}(x))dx| \leq \int_a^b |f(x) - p_{m-1}(x)|dx \\
&\leq \int_a^b dx \max_{x\in[a,b]} |f(x) - p_{m-1}(x)| \leq (b-a)\frac{1}{m!} \max_{s\in[a,b]} |f^{(m)}(s)|(b-a)^m \\
&= \frac{1}{m!} \max_{s\in[a,b]} |f^{(m)}(s)|(b-a)^{m+1}.
\end{aligned}$$

As always, this is an upper bound of the integration error; however, this bound unfortunately is not sharp. In particular, this bound suggests that an m-point Newton-Cotes rule integrates exactly degree $m-1$ polynomials. In fact, when m is odd — as in the case of the midpoint rule or the Simpson's rule — m-point Newton-Cotes rule integrates exactly degree m polynomial. The result is summarized in the following theorem:

Theorem 3.2. Given a function f over domain $[a, b]$ and an integral $I \equiv \int_a^b f(x)dx$, the error associated with the m-point closed Newton-Cotes rule is bounded by

$$|I - Q_{\mathrm{NC}(m)}| \leq \begin{cases} C_m \max\limits_{s\in[a,b]} |f^{(m)}(s)|(b-a)^{m+1}, & m \text{ is even} \\ C_m \max\limits_{s\in[a,b]} |f^{(m+1)}(s)|(b-a)^{m+2}, & m \text{ is odd,} \end{cases}$$

where C_m is a constant independent of b, a, and f.

Proof. A proof for an even m is provided above; a proof for an odd m is provided in Appendix 3.6. □

We make a few observations:

1. When the number of points m is even, the error depends on the m-th derivative of the integrand. The quadrature rule is exact for degree $m - 1$ polynomials.

2. When the number of points m is odd, the error depends on the $m + 1$-st derivative of the integrand. The quadrature rule is exact for degree m polynomials.

We now elaborate the result for the two Newton-Cotes rules we have explicitly considered: trapezoid rule and Simpson's rule.

Trapezoid rule. The error associated with the trapezoid rule ($m = 2$) depends on the second derivative of the integrand, and the rule is exact for linear functions. The result is (perhaps) expected as the rule is based on a linear interpolant. The bound with an explicit expression for the constant C_m is

$$|I - Q_{\mathrm{NC}(2)}| \leq \frac{1}{12} \max_{s\in[a,b]} |f^{(2)}(s)|(b-a)^3.$$

We here omit the proof for brevity.

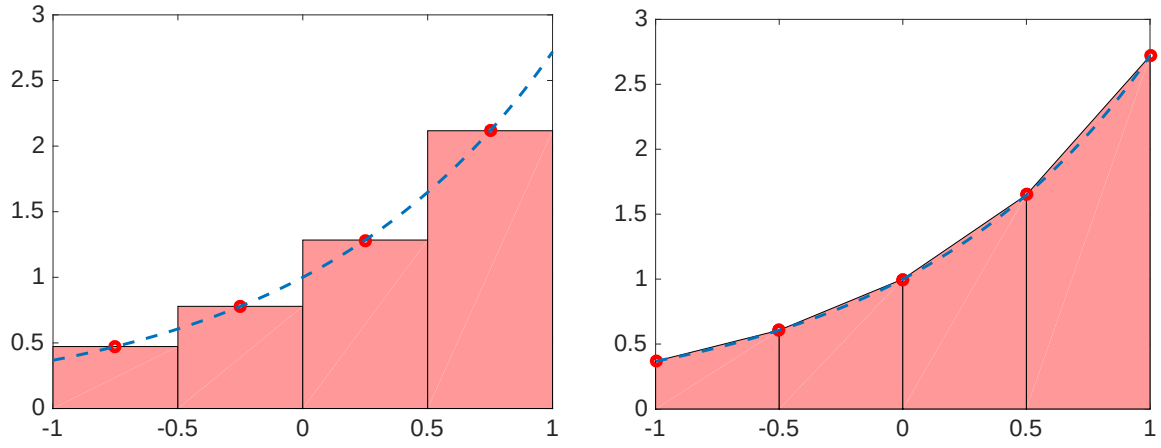
Simpson's rule. The error associated with the Simpson's rule ($m = 3$) depends on the *fourth* derivative of the integrand, and the rule is exact for *cubic* functions. The result is (perhaps) surprising because the rule is based on a *quadratic* interpolant. This rather surprising result is due to the fact that the quadrature points are symmetric and the interpolant associated with the quadrature rule is of even degree. (See Appendix 3.6 for a proof.) The bound with an explicit expression for the constant C_m is

$$|I - Q_{\mathrm{NC}(3)}| \leq \frac{1}{2880} \max_{s\in[a,b]} |f^{(4)}(s)|(b-a)^5.$$

We again omit the proof for brevity.

3.4 Composite rules

As in the case of interpolation, we can improve the accuracy of the integration rules by dividing the segment $[a, b]$ into smaller segments and by using a quadrature rule on each of the segments. Specifically, we first introduce equispaced points $a = s_1 < s_2 < \cdots < s_{N+1} = b$ over $[a, b]$. We then



(a) composite midpoint, $N = 4$

(b) composite trapezoid, $N = 4$

Figure 3.3: Composite midpoint and trapezoid rules applied to $\int_{-1}^{1} \exp(x)dx$.

introduce N segments $S_i \equiv [s_i, s_{i+1}]$ delineated by the $N+1$ points. The length of each segment is $h \equiv s_{i+1} - s_i = (b-a)/N$. We then consider *composite rules* of a form

$$Q^{(N)}_{\mathrm{NC}(m)} = \sum_{i=1}^{N} Q_{\mathrm{NC}(m)}(S_i),$$

where $Q_{\mathrm{NC}(m)}(S_i)$ is the closed Newton-Cotes rule over the segment S_i.

Composite trapezoid rule. As a concrete example, we consider the composite trapezoid rule. Dividing $[a, b]$ into N segments and applying the trapezoid rule to each segment yield

$$\begin{aligned} Q^{(N)}_{\mathrm{NC}(2)} &= \sum_{i=1}^{N} (s_{i+1} - s_i) \left(\frac{1}{2} f(s_i) + \frac{1}{2} f(s_{i+1}) \right) \\ &= h \left(\frac{1}{2} f(s_1) + \frac{1}{2} f(s_{N+1}) + \sum_{i=2}^{N} f(s_i) \right). \end{aligned}$$

The composite trapezoid rule for $N = 4$ is illustrated in Figure 3.3(b). Note that when a composite rule is constructed from a closed Newton-Cotes rule (i.e., a quadrature rule that includes the two endpoints as quadrature points), we can reduce the number of function evaluations by reusing the endpoint values for two neighboring segments. Thus, the number of function evaluations required for composite trapezoid rule with N segments is $N + 1$ (and not $2N$).

We now analyze the error associated with composite rules. Applying the Newton-Cotes *a priori* error bound in Theorem 3.2 to each segment, we obtain the following *a priori* error bound for composite rules.

Theorem 3.3. Given a function f over domain $[a, b]$ and an integral $I = \int_a^b f(x)dx$, the error associated with the N-segment composite rule based on the m-point closed Newton-Cotes rule for

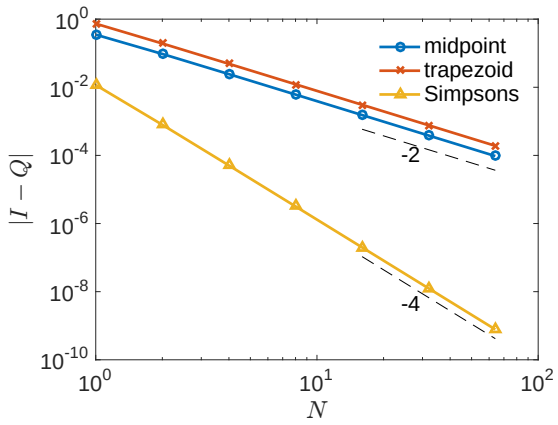


Figure 3.4: Convergence of a few composite Newton-Cotes rules with N applied to $I = \int_{-1}^{1} \exp(x)dx$.

each segment is bounded by

$$|I - Q^{(N)}_{\mathrm{NC}(m)}| \leq \begin{cases} C_m \max\limits_{s\in[a,b]} |f^{(m)}(s)|h^m(b-a), & m \text{ is even} \\ C_m \max\limits_{s\in[a,b]} |f^{(m+1)}(s)|h^{m+1}(b-a), & m \text{ is odd,} \end{cases}$$

where $h \equiv (b-a)/N$, and C_m is a constant independent of b, a, and f.

Proof. For an even m,

$$\begin{aligned} |I - Q^{(N)}_{\mathrm{NC}(m)}| &= |\sum_{i=1}^{N} \int_{S_i} f(x)dx - \sum_{i=1}^{N} Q_{\mathrm{NC}(m)}(S_i)| \leq \sum_{i=1}^{N} |\int_{S_i} f(x)dx - Q_{\mathrm{NC}(m)}(S_i)| \\ &\leq \sum_{i=1}^{N} \max_{s\in[s_i,s_{i+1}]} |f^{(m)}(s)|h^{m+1} \leq \max_{s\in[a,b]} |f^{(m)}(s)|h^m \sum_{i=1}^{N} h \\ &= \max_{s\in[a,b]} |f^{(m)}(s)|h^m(b-a). \end{aligned}$$

The proof for an odd m follows the same procedure. □

We make a few observations:

1. The integration error depends on the m-th (resp $m+1$-st) derivative of the underlying function for an even m (resp an odd m).

2. The integration error depends on the quadrature degree m (or $m+1$) and the segment length h.

3. For a fixed m, the integration error decreases as h^m (or h^{m+1}) with the segment length h.

An example of convergence plots for a few composite Newton-Cotes rules is shown in Figure 3.4.

3.5 Summary

We summarize key points of this lecture:

1. Quadrature rules allow us to estimate the value of a definite integral using function values at a finite number of points.

2. A quadrature rule is uniquely determined by a set of quadrature points and a set of quadrature weights.

3. Newton-Cotes rules are quadrature rules based on i) the construction of a polynomial interpolant using equispaced interpolation points and ii) the exact integration of the polynomial interpolant.

4. We can systematically identify the quadrature weights associated with a Newton-Cotes quadrature rule of arbitrary degree using Lagrange basis polynomials.

5. A m-point Newton-Cotes rule integrates exactly polynomials of degree $m-1$ if m is even. A m-point Newton-Cotes rule integrates exactly polynomials of degree m if m is odd.

6. Composite rules improve the accuracy of the integration rules by dividing the segment $[a, b]$ into smaller segments and by using quadrature rule on each of the segments.

7. Composite Newton-Cotes rules based on N intervals (and m quadrature points per segment) converges at the rate of $1/N^m$ if m is even and at the rate of $1/N^{m+1}$ if m is odd.

3.6 Appendix

We provide a proof of Theorem 3.2 for Newton-Cotes rules based on an odd number of points. (We have already provided a proof for m even in the main text.)

We first recall that m-point closed Newton-Cotes rules are based on m equispaced points x_i, $i = 1, \ldots, m$. We now introduce a degree-m polynomial q of the form

$$q^m(x) = \prod_{i=1}^{m} (x - x_i).$$

The polynomial q^m has two important properties: first, its m roots are the Newton-Cotes quadrature points; second, for m odd, q^m is an odd function about the midpoint of $[a, b]$ and hence $\int_a^b q^m(x)dx = 0$. We next introduce the usual degree $m-1$ polynomial interpolant p_{m-1} of f so that $p_{m-1}(x_i) = f(x_i)$, $i = 1, \ldots, m$. We then introduce a degree-m polynomial

$$z_m(x) \equiv p_{m-1}(x) + \frac{f(x_0) - p_{m-1}(x_0)}{q^m(x_0)} q^m(x),$$

where $x_0 \in [a, b]$ is some point that is distinct from the Newton-Cotes quadrature points. We now note that, for $i = 1, \ldots, m$, $q^m(x_i) = 0$ and hence

$$z_m(x_i) = p_{m-1}(x_i) = f(x_i),$$

and for $i = 0$,

$$z_m(x_0) = p_{m-1}(x_0) + \frac{f(x_0) - p_{m-1}(x_0)}{q^m(x_0)} q^m(x_0) = f(x_0).$$

Hence, the degree-m polynomial z_m interpolates f; its $m+1$ interpolation points are x_i, $i = 0, \ldots, m$. Recalling $\int_a^b q^m(x)dx = 0$ for an odd m, it follows that

$$\begin{aligned}
|I - Q_{\mathrm{NC}(m)}| &= |\int_a^b (f(x) - p_{m-1}(x))dx| \\
&= |\int_a^b (f(x) - p_{m-1}(x) - \frac{f(x_0) - p_{m-1}(x_0)}{q^m(x_0)} q^m(x))dx| \\
&= |\int_a^b (f(x) - z_m(x))dx| \leq \int_a^b dx \max_{x \in [a,b]} |f(x) - z_m(x)|.
\end{aligned}$$

Finally, invoking the interpolation error bound for the degree-m interpolant z_m (and not degree $m-1$ interpolant p_{m-1}) yields the desired result.

Lecture 4

Numerical integration: Clenshaw-Curtis, Gauss, and adaptive quadratures

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

4.1 Preliminary

In the previous lecture we introduced the Newton-Cotes rules for numerical integration and analyzed the error associated with the approximation. In this lecture we consider more advanced techniques that will provide a more accurate estimate of the integral for a given number of function evaluations.

Before we introduce the quadrature rules, we make one preliminary remark. Following the standard practice, we will develop quadrature rules for the interval $[-1,1]$; the quadrature rule over the interval will be given in terms of quadrature points $\{x_i\}_{i=1}^m$, each of which is in $[-1,1]$, and the quadrature weights $\{w_i\}_{i=1}^m$. The quadrature rule then approximates the integral of f over $[-1,1]$:

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{m} w_i f(x_i).$$

If we wish to integrate a function $\tilde{f}$ over an arbitrary interval $[a,b]$, we then need to transform the quadrature points and weights. The appropriate transformation is given by

$$\int_{a}^{b} \tilde{f}(\tilde{x})d\tilde{x} \approx \sum_{i=1}^{m} \tilde{w}_i \tilde{f}(\tilde{x}_i),$$

where

$$\tilde{x}_i = a + \frac{b-a}{2}(x_i+1), \quad i = 1,\dots,m,$$
$$\tilde{w}_i = \frac{b-a}{2} w_i, \quad i = 1,\dots,m.$$

To observe why this transformation works, we note that

$$\int_a^b \tilde{f}(\tilde{x})d\tilde{x} = \int_{-1}^{1} \tilde{f}(a + \frac{b-a}{2}(x+1))\frac{b-a}{2}dx \approx \sum_{i=1}^{m} w_i \tilde{f}(a + \frac{b-a}{2}(x_i+1))\frac{b-a}{2}$$
$$= \sum_{i=1}^{m} \underbrace{\frac{b-a}{2}w_i}_{\tilde{w}_i} \tilde{f}(\underbrace{a + \frac{b-a}{2}(x_i+1)}_{\tilde{x}_i}) = \sum_{i=1}^{m} \tilde{w}_i \tilde{f}(\tilde{x}_i).$$

We observe that the transformation provides a quadrature rule for $[a, b]$ based on a reference quadrature rule for $[-1, 1]$. For the rest of the lecture, we will hence focus on the development of quadrature rules for the reference domain $[-1, 1]$.

4.2 Clenshaw-Curtis quadrature

We recall that the (single-segment) Newton-Cotes rules are of the form

$$Q_{\mathrm{NC}(m)} = \sum_{i=1}^{m} w_i f(x_i),$$

where $\{x_i\}_{i=1}^m$ is a set of *equispaced* quadrature points, and $\{w_i\}_{i=1}^m$ is a set the quadrature weights that integrates exactly polynomials of degree at least $m-1$. Lower-order Newton-Cotes rules, and in particular composite Newton-Cotes rules, are commonly used in practice. However, Newton-Cotes rules have one major drawback: they do not converge for a general f as $m \to \infty$. This is closely related to the behavior we observed for polynomial interpolation: interpolants based on equispaced set of points is not convergent for all f as polynomial degree increases and in particular suffers from the Runge's phenomenon. The poor behavior of higher-order Newton-Cotes rules applied to the Runge's function, $\frac{1}{1+25x^2}$, $x \in [-1, 1]$, is shown in Figure 4.1.

In the context of interpolation, we have however also observed that we can use a set of interpolation points that are clustered towards the edges — for instance the Chebyshev nodes — to overcome the instability. We can use these more stable interpolants associated with the Chebyshev nodes also in the contest of numerical integration.

The *Clenshaw-Curtis rules* are based on the Chebyshev nodes. The quadrature rule is of the form

$$Q_{\mathrm{CC}(m)} = \sum_{i=1}^{m} w_i f(x_i),$$

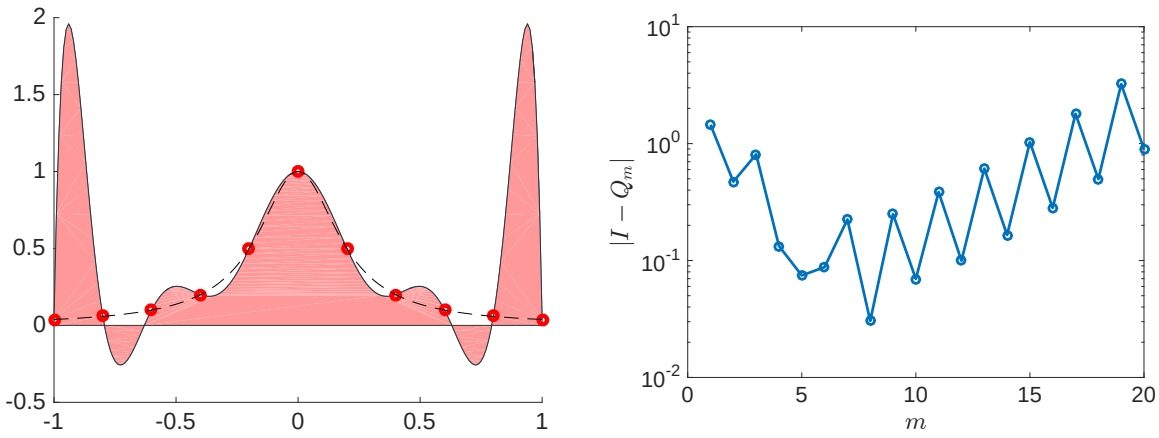
where $\{x_i\}_i$ are the Chebyshev nodes over $[-1, 1]$,

$$x_i = -\cos\left(\frac{(i-1)\pi}{m-1}\right), \quad i = 1, \ldots, m,$$

and $\{w_i\}_i$ are given by

$$w_i = \int_{-1}^{1} \ell_{\mathrm{Chebyshev},i}(x)dx, \quad i = 1, \ldots, m,$$

where $\ell_{\mathrm{Chebyshev},i}$ are the Lagrange basis polynomial associated with the Chebyshev nodes. As in the case of the Newton-Cotes rules, this choice of the quadrature points and weights can be



(a) illustration

(b) integration error

Figure 4.1: Integration of the Runge's function $f(x) = \frac{1}{1+25x^2}$ over $[-1, 1]$ using Newton-Cotes rules of various degree.

interpreted as integrating exactly a polynomial interpolant of degree $m-1$; specifically,

$$Q_{\mathrm{CC}(m)} = \int_{-1}^{1} p_{m-1}(x)dx,$$

where $p_{m-1}(x)$ is the polynomial interpolant for f constructed using the Chebyshev nodes. The quadrature points and weights associated with the $m = 1, \ldots, 6$ Clenshaw-Curtis quadrature rules are shown in Table 4.1.

Figure 4.2 shows the Clenshaw-Curtis rules applied to the Runge's function. We observe that the Clenshaw-Curtis rules converge to the exact value of the integral, unlike the Newton-Cotes rules which diverge.

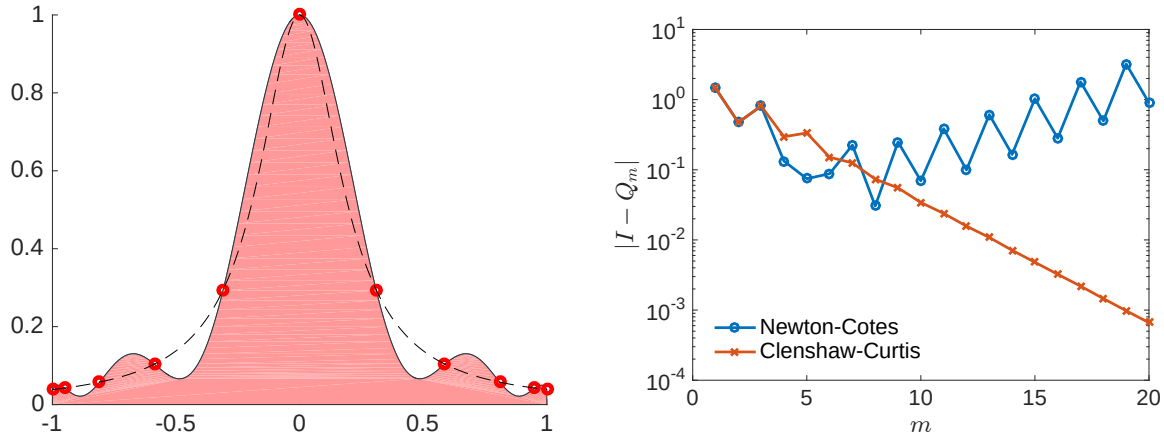
4.3 Clenshaw-Curtis quadrature: error analysis

Because the Newton-Cotes and Clenshaw-Curtis rules are both constructed from the approximation of the integrand by a polynomial interpolant and the integration of the polynomial interpolant, their error bounds have similar forms. The key difference between the Newton-Cotes and Clenshaw-Curtis rules is of course the choice of the quadrature points, which results in different stability behavior as m increases. This is reflected in the difference in the leading m-dependent constant in the error bound: specifically, we replace the constant C_m in the Newton-Cotes rules with the C_m^{CC} for the Clenshaw-Curtis rules. (A careful analysis of this constant is however beyond the scope of this lecture. We only note that C_m^{CC} for the Clenshaw-Curtis rules is smaller than C_m for the Newton-Cotes rules.) We summarize the result in the theorem below:

Theorem 4.1. Given a function f over domain $[-1, 1]$ and an integral $I \equiv \int_{-1}^{1} f(x)dx$, the error

m	x_i	w_i
1	0.0000000000000000	2.0000000000000000
2	±1.0000000000000000	1.0000000000000000
3	0.0000000000000000	1.3333333333333333
	±1.0000000000000000	0.3333333333333333
4	±0.5000000000000000	0.8888888888888889
	±1.0000000000000000	0.1111111111111111
5	0.0000000000000000	0.8000000000000000
	±0.7071067811865476	0.5333333333333333
	±1.0000000000000000	0.0666666666666666
6	±0.3090169943749475	0.5992569587999889
	±0.8090169943749475	0.3607430412000111
	±1.0000000000000000	4.0000000000000000

Table 4.1: The Clenshaw-Curtis quadrature rules for $m = 1$ to 6 points.



(a) illustration

(b) integration error

Figure 4.2: Integration of the Runge's function $f(x) = \frac{1}{1+25x^2}$ over $[-1, 1]$ using the Clenshaw-Curtis rules of various degree.

associated with the m-point Clenshaw-Curtis rule is bounded by

$$|I - Q_{\mathrm{CC}(m)}| \leq \begin{cases} C_m^{\mathrm{CC}} \max\limits_{s\in[-1,1]} |f^{(m)}(s)|, & m \text{ is even} \\ C_m^{\mathrm{CC}} \max\limits_{s\in[-1,1]} |f^{(m+1)}(s)|, & m \text{ is odd,} \end{cases}$$

where C_m^{CC} is a constant independent of f.

4.4 Gauss quadrature

The Newton-Cotes rules are based on equispaced points, and the Clenshaw-Curtis rules are based on Chebyshev nodes that are designed to minimize the interpolation error in the worst case. In this section we ask ourselves a question: if we could choose the integration weights *and* integration points strictly for the purpose of integration, then could we formulate a more accurate quadrature rule for a given number of function evaluations? It turns out the answer to the question is yes: one approach is called the *Gauss quadrature*. The Gauss quadrature provides the highest order of accuracy for a given number of function evaluations. In particular, a Gauss quadrature rule with m quadrature points integrates exactly polynomials of degree up to $2m-1$ (unlike $m-1$ (or m) for Newton-Cotes and Clenshaw-Curtis).

m = 1 point rule. As a concrete example, let us first construct a $m = 1$ Gauss quadrature rule. Our quadrature rule will be in the form

$$Q_{\mathrm{Gauss}(1)} = w_1 f(x_1);$$

we wish to identify the weight w_1 and the point x_1 that together provide the highest order of accuracy. Specifically, we aim to integrate exactly the polynomials of degree up to $2m-1 = 1$ using this quadrature rule. Choosing monomials $\{1, x\}$ as the basis functions for convenience, this exact integration condition translates to

$$\begin{aligned} f(x) = 1: &\qquad w_1 f(x_1) = w_1 = \int_{-1}^{1} 1 dx = 2, \\ f(x) = x: &\qquad w_1 f(x_1) = w_1 x_1 = \int_{-1}^{1} x dx = 0. \end{aligned}$$

We readily observe that $w_1 = 2$ and $x_1 = 0$ provide the desired result. Hence $m = 1$ Gauss quadrature rule is

$$Q_{\mathrm{Gauss}(1)} = 2f(0),$$

which is in fact the midpoint rule. The quadrature rule integrates exactly polynomials of degree two using just one point.

m = 2 point rule. We now consider a $m = 2$ Gauss quadrature rule. Our quadrature rule will be in the form

$$Q_{\mathrm{Gauss}(2)} = w_1 f(x_1) + w_2 f(x_2).$$

We wish to identify parameters x_1, x_2, w_1, and w_2 such that the rule integrates exactly the polynomials of degree $2m-1 = 3$. Choosing monomials $\{1, x, x^2, x^3\}$ as the basis functions for convenience,

m	x_i	w_i
1	0.0000000000000000	2.0000000000000000
2	±0.5773502691896257	1.0000000000000000
3	0.0000000000000000	0.8888888888888888
	±0.7745966692414834	0.5555555555555556
4	±0.3399810435848563	0.6521451548625461
	±0.8611363115940526	0.3478548451374538
5	0.0000000000000000	0.5688888888888889
	±0.5384693101056831	0.4786286704993665
	±0.9061798459386640	0.2369268850561891
6	±0.6612093864662645	0.3607615730481386
	±0.2386191860831969	0.4679139345726910
	±0.9324695142031521	0.1713244923791704

Table 4.2: Gauss quadrature rules for $m = 1$ to 6 points.

this exactly integration condition translates to

$$
\begin{aligned}
f(x) = 1: &\qquad w_1 f(x_1) + w_2 f(x_2) = w_1 + w_2 = \int_{-1}^{1} 1 dx = 2, \\
f(x) = x: &\qquad w_1 f(x_1) + w_2 f(x_2) = w_1 x_1 + w_2 x_2 = \int_{-1}^{1} x dx = 0, \\
f(x) = x^2: &\qquad w_1 f(x_1) + w_2 f(x_2) = w_1 x_1^2 + w_2 x_2^2 = \int_{-1}^{1} x^2 dx = \frac{2}{3}, \\
f(x) = x^3: &\qquad w_1 f(x_1) + w_2 f(x_2) = w_1 x_1^3 + w_2 x_2^3 = \int_{-1}^{1} x^3 dx = 0.
\end{aligned}
$$

After some algebraic manipulation, we find that the solution to the system of nonlinear algebraic equations is given by $x_1 = -1/\sqrt{3}$, $x_2 = 1/\sqrt{3}$, $w_1 = 1$, and $w_2 = 1$. Hence the $m = 2$ Gauss quadrature rule is

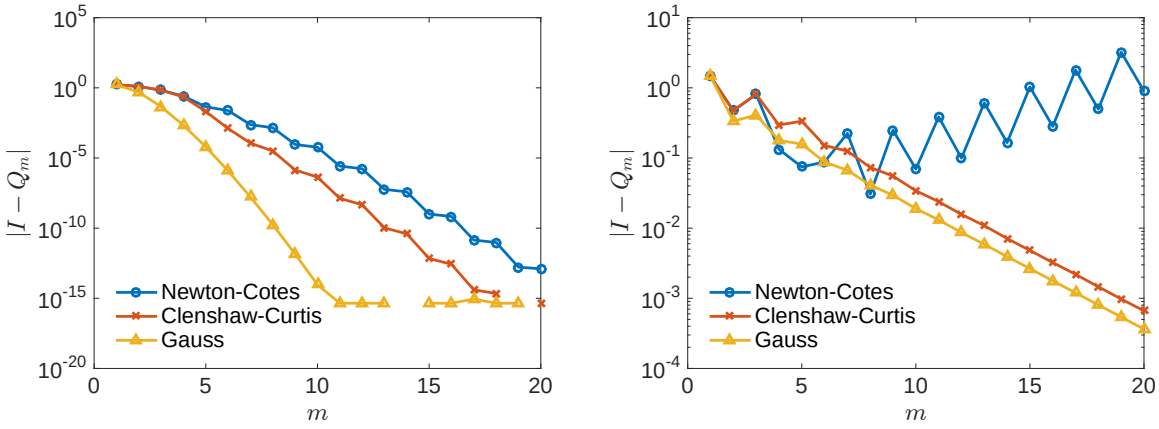
$$
Q_{\text{Gauss}(2)} = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right).
$$

The quadrature rule integrates exactly polynomials of degree three using just two points.

General m point rule. In general, m-point Gauss quadrature rule has the form

$$
Q_{\text{Gauss}(m)} = \sum_{i=1}^{m} w_i f(x_i)
$$

and integrates exactly the polynomial of degree $2m - 1$. We can generate these quadrature rules in a systematic manner. However, the construction relies on Legendre polynomials, which is beyond the scope of this lecture. We here simply provide the quadrature points and weights associated with several Gauss quadrature rules in Table 4.2. More precisely, these rules are called *Legendre-Gauss quadrature rules*, as the quadrature points are the roots of Legendre polynomials.



(a) $\int_{-1}^{1}(\exp(x) + \cos(\pi x))dx$ (b) $\int_{-1}^{1} \frac{1}{1+25x^2}dx$

Figure 4.3: Comparison of high-order Newton-Cotes, Clenshaw-Curtis, and Gauss quadrature rules.

4.5 Gauss quadrature: error analysis

The error in the m-point Gauss quadrature is bounded from above by

$$|\int_{-1}^{1} f(x)dx - Q_{\text{Gauss}(m)}| \leq \frac{(m!)^4}{(2m+1)((2m)!)^3} \max_{x\in[-1,1]} |f^{(2m)}(x)|.$$

We make a few key observations:

1. The approximation error depends on the $2m$-th derivative of the underlying function. In particular, the integration is exact for polynomials of degree up to and including $2m-1$.

2. The error decreases exponentially with m.

We now integrate a few functions using Gauss quadrature. Figure 4.3(a) shows the error associated with the integration of a smooth and well-behaved function $\exp(x) + \cos(\pi x)$ over $[-1, 1]$. We observe that the Gauss quadrature converges more rapidly than the high-order Newton-Cotes or Clenshaw-Curtis quadrature rules. Figure 4.3(b) shows the error associated with the integration of the Runge's function, $\frac{1}{1+25x^2}$, over $[-1, 1]$. Similar to the Clenshaw-Curtis rules, and unlike the Newton-Cotes rules, the Gauss quadrature rules converge to the correct value without exhibiting instability. However, we note that the convergence is slower than that observed for $\exp(x)+\cos(\pi x)$.

4.6 h vs p convergence: shorter segments or higher degrees?

We have so far seen two different means of reducing the quadrature error: the first is to increase the degree of polynomial used in each segment (assuming we use Clenshaw-Curtis or Guass quadrature such that we do not suffer from instability); the second is to use the composite rule and increase the number of segments. Increasing the quadrature degree or reducing the segment length requires additional function evaluations, and hence we might ask what might be a good strategy to increase the accuracy of the approximation for a given number of function evaluations. To compare the

different error convergence behaviors associated with increasing the degree and reducing the segment lengths, we apply three different quadrature rules — (single-segment) Guass quadrature, composite trapezoid, and composite Simpson's — to integrands of various smoothness.

We first consider the integration of a smooth function, $\exp(x) + \cos(\pi x)$. The convergence plot is shown in Figure 4.4(a). Because the function is smooth, the Gauss quadrature rules converge very rapidly as the quadrature degree increases; we in fact observe exponential convergence. The composite trapezoid rule and the composite Simpson's rule also converge as the number of segments increases; however, the convergence is algebraic (and not exponential) and is much slower than that observed for the Gauss quadrature. For instance, in order to achieve the accuracy of 10^{-10}, the Gauss quadrature only requires 9 function evaluations, whereas the Simpson's rule requires $\mathcal{O}(100)$ function evaluations; the trapezoid rule requires even more function evaluations to achieve the same error level. The convergence rate for the composite trapezoid and Simpson's rule are 2 and 4, respectively, as expected from our error analysis in the previous lecture.

We next consider an integrand that has a singularity at the end of the domain, $\sqrt{x+1}$. The convergence plot is shown in Figure 4.4(b). Due to the presence of the singularity, the Gauss quadrature does not converge as rapidly as it did for the smooth integrand. In fact, we no longer observe exponential convergence and instead observe an algebraic convergence. We also note that the convergence rate of the trapezoid and Simpson's rule are reduced to 1.5 for this less smooth function. While increasing the quadrature degree of the Gauss quadrature is still more efficient than using composite trapezoid or Simpson's rule, the difference is much smaller for $\sqrt{x+1}$ than for $\exp(x) + \cos(\pi x)$.

We finally consider an integrand that has a singularity in the domain, $|x+1/7|$. The convergence plot is shown in Figure 4.4(c). The convergence of the Gauss quadrature with the quadrature degree significantly deteriorates for this non-smooth function. The accuracy for a given number of function evaluations (hence the cost) is comparable to that for the composite trapezoid and Simpson's rules. We also note that the convergence of all methods are rather slow: achieving $\mathcal{O}(10^{-4})$ accuracy requires $\mathcal{O}(100)$ function evaluations for all of the methods.

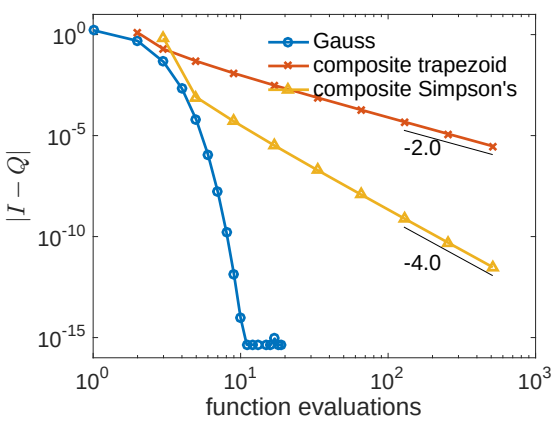
To conclude, we observe that, for smooth functions, the Gauss quadrature rules (or the Clenshaw-Curtis rules) provide very rapid convergence with the increase in the quadrature degree. For non-smooth functions, however, the convergence of the Gauss quadrature rules can be no better than the lower-order composite Newton-Cotes rules. In addition, the convergence can be rather slow for all of the methods.

4.7 Adaptive quadrature: adaptive Simpson's method

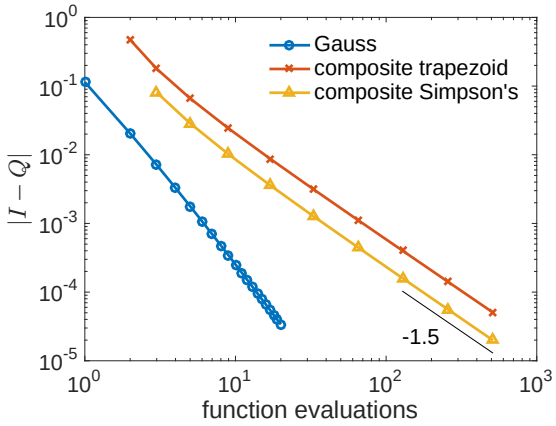
We have seen that Gauss quadrature provides much more accurate solution than composite Newton-Cotes rules for smooth integrands, but its performance is limited for non-smooth functions. In many cases, the functions we wish to integrate are not smooth. Integrands may be formally not smooth and the higher derivatives might not exist; consider for instance the integration of $\sqrt{x+1}$ over $[-1, 1]$. Or, integrands may be formally smooth but may vary significantly over a small region. In these cases, global (i.e. single segment) quadrature rules or composite rules with equispaced segments do not converge rapidly. In order to overcome the limitation, we consider *adaptive quadrature rules*.

The two key ingredients of adaptive quadrature rules are

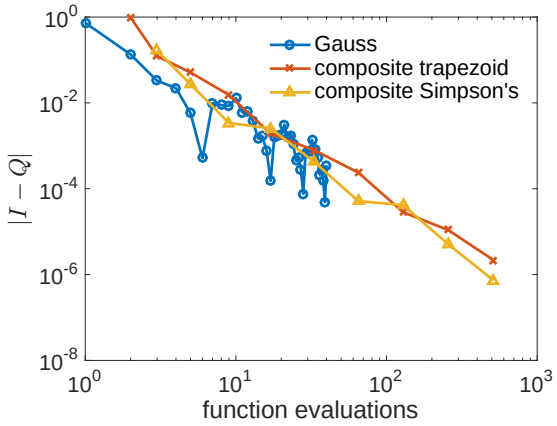
1. an error estimate which assesses the error associated with the current approximation;



(a) $\int_{-1}^{1}(\exp(x)+\cos(\pi x))dx$



(b) $\int_{-1}^{1}\sqrt{x+1}dx$



(c) $\int_{-1}^{1}|x+1/7|dx$

Figure 4.4: Comparison of Gauss quadrature (increasing degree) and composite low-order Newton-Cotes formulas (increasing number of segments).

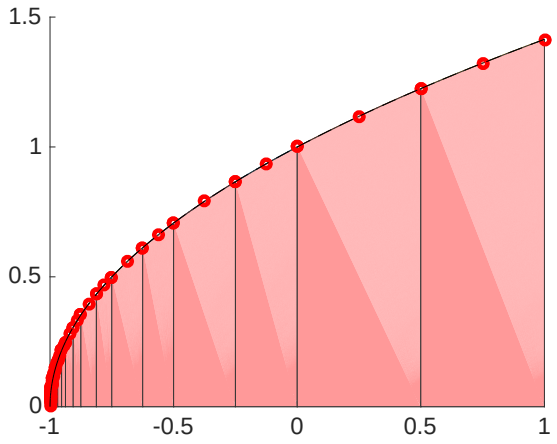


Figure 4.5: Approximation of $\int_{-1}^{1} \sqrt{x+1}dx$ using the adaptive Simpson's method.

2. an adaptive selection of segment lengths based on the error estimate.

As a concrete example, we introduce the adaptive Simpson's method. Given (not necessarily equispaced) points $a \equiv x_1 < \cdots < x_{N+1} \equiv b$ that delineate N intervals, the approximation of the integral based on the adaptive Simpson's rule is given by

$$I \equiv \int_a^b f(x)dx \approx \sum_{i=0}^{N} Q^{(2)}_{\mathrm{NC}(3)}([x_i, x_{i+1}]) = \sum_{i=0}^{N} \left(Q_{\mathrm{NC}(3)}([x_i, x_i^{\mathrm{mid}}]) + Q_{\mathrm{NC}(3)}([x_i^{\mathrm{mid}}, x_{i+1}]) \right),$$

where $x_i^{\mathrm{mid}} \equiv \frac{1}{2}(x_i + x_{i+1})$. We recall that $Q^{(2)}_{\mathrm{NC}(3)}([x_i, x_{i+1}])$ denotes the approximation of $\int_{x_i}^{x_{i+1}} f(x)dx$ using the two-segment composite Simpsons's rule, and $Q_{\mathrm{NC}(3)}([x_i, x_i^{\mathrm{mid}}])$ denotes the approximation of $\int_{x_i}^{x_i^{\mathrm{mid}}} f(x)dx$ using the (single-segment) Simpson's rule. Of course, the strength of the adaptive Simpson's method lies in the way it automatically identifies points $\{x_i\}_{i=1}^{N+1}$, to achieve a desired accuracy. An example of adaptive selection of the interval lengths is shown in Figure 4.5.

Error estimate. A key ingredient of any adaptive scheme is the error estimate. We here focus on the estimation of the error over a single interval; the error over all the intervals is simply the sum of individual contributions. The error estimate for the adaptive Simpson's rule is based on a comparison of the integral approximated using two different quadrature rules. Specifically, given the approximation of the integral over $[a, b]$ using a single interval,

$$I \equiv \int_a^b f(x)dx \approx Q^{(2)}_{\mathrm{NC}(3)}([a, b]) \equiv Q_{\mathrm{NC}(3)}([a, c]) + Q_{\mathrm{NC}(3)}([c, b])$$

for $c = (a + b)/2$, the adaptive Simpson's rule considers an error estimate,

$$E([a, b]) \equiv \frac{1}{15} |Q^{(2)}_{\mathrm{NC}(3)}([a, b]) - Q_{\mathrm{NC}(3)}([a, b])|.$$

We make two observations about this particular error estimate.

1. The first is regarding the cost. The three quadrature points $\{a, \frac{1}{2}(a + b), b\}$ for $Q_{\mathrm{NC}(3)}([a, b])$ is a subset of the five quadrature points $\{a, \frac{1}{4}(a + b), \frac{1}{2}(a + b), \frac{3}{4}(a + b), b\}$ for $Q^{(2)}_{\mathrm{NC}(3)}([a, b])$. Hence, we can evaluate both quadrature rules using just five function evaluations.

2. The second is regarding the accuracy. The error associated with the error estimate $E([a,b])$ is one order higher than the error in the integral itself. In other words,

$$|I - Q^{(2)}_{\mathrm{NC}(3)}([a,b])| \sim \mathcal{O}((b-a)^5)$$

whereas

$$||I - Q^{(2)}_{\mathrm{NC}(3)}([a,b])| - E([a,b])| \sim \mathcal{O}((b-a)^6).$$

A sketch of the proof is provided in the Appendix. In words, for a small interval length $b-a$ — which we will have in composite rules — the error in the error estimate will be smaller than the error itself, and hence the estimate serves as a good indicator of the error.

The error estimate for the adaptive Simpson's rule, $E([a,b])$, is an example of an *a posteriori error estimate*. The error estimate differs from *a priori* error estimates considered in the context of interpolation and integration, which take on the form $e \sim C_k h^k$. Specifically, unlike *a priori* error estimates, *a posteriori* error estimates are *computable*; it does not involve uncomputable terms such as $\|f^{(m)}\|_{L^\infty}$. Hence, *a posteriori* error estimates can be directly incorporated in a numerical method to inform the user the level of confidence they should have in the approximation.

Adaptive algorithm. Adaptive Simpson's method is most easily implemented as a recursive algorithm. An example implementation is shown in Algorithm 1. In each call to the adaptive Simpson's function, we compute the approximate integral as well as the error estimate. If the error tolerance is met for the current interval, then we simply return the current approximate value of the integral as well as the error estimate. If the error tolerance is not met, then we split the interval into two subintervals and invoke adaptive quadrature on each subinterval; we however set the error tolerance over each of the subinterval to be the half of the error tolerance over the whole interval, such that the error over the whole interval — which is the sum of the contributions from the two intervals — meets the desired error tolerance.

4.8 Adaptive Simpson's method: examples

We now apply the adaptive Simpson's method to compute the integral $\int_{-1}^{1} \cos((x+1.05)^{-1})dx$. As shown in Figure 4.6(a), the integral exhibits rapid oscillation near $x = -1$; the integrand however is formally smooth, i.e. in $C^\infty([-1,1])$.

Figure 4.6(b) illustrates the adaptive Simpson's method in action: we observe that shorter segments are used to resolve the rapidly varying feature near $x = -1$ and longer segments are used in the regions with relatively small function variation. Figure 4.6(c) shows the variation in the interval length more quantitatively.

Figure 4.6(c) shows the convergence of the adaptive Simpson's method. To generate the figure, the adaptive scheme was invoked for several different values of the error tolerance δ. We see that the adaptive Simpson's method requires fewer function evaluations than the non-adaptive Simpson's rule with a uniform interval spacing. We also observe that the error estimate $E([a,b])$ is an accurate indicator of the true error $|I - Q([a,b])|$, in particular for a tight error tolerance for which the intervals are small.

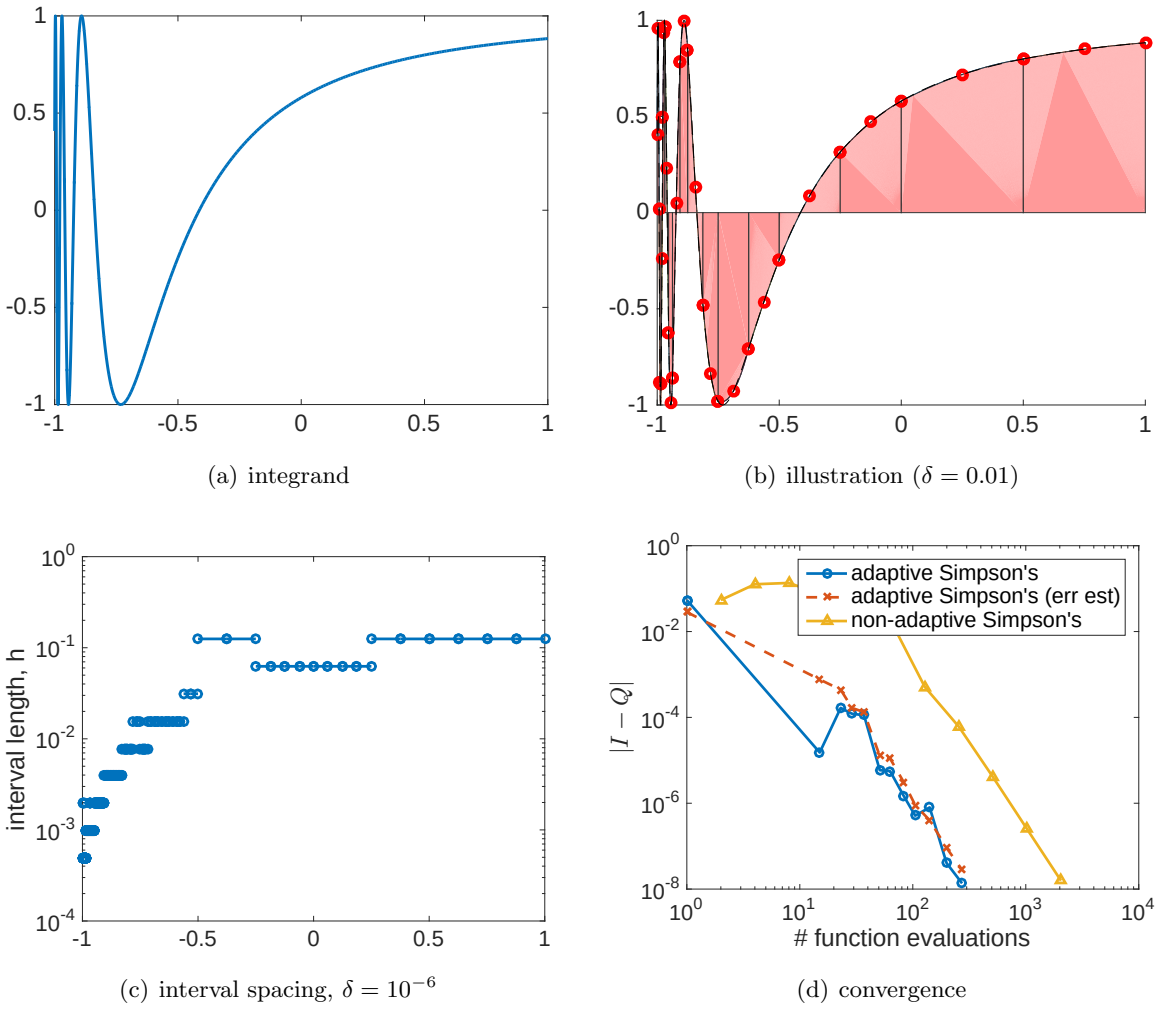
The adaptive Simpson's method is even more effective when the integrand is singular. As an example, we consider the approximation of $\int_0^1 \sqrt{x}dx$. Figure 4.7(a) shows the interval spacing selected by the adaptive Simpson's method for the error tolerance of $\delta = 10^{-6}$. Note that the

Algorithm 1: Adaptive Simpson's method

Input: f: integrand
a, b: lower and upper bound of integration
δ: error tolerance

Output: Q: integral approximation
E: error estimate

function $[Q, E] = \text{adaptive\_simpsons}(f, a, b, \delta)$
$c = (a + b)/2$
$Q_0 = Q_{\text{NC(3)}}([a, b])$ `// single interval approximation`
$Q_1 = Q_{\text{NC(3)}}([a, c]) + Q_{\text{NC(3)}}([c, b])$ `// two-interval approximation`
$E = |Q_1 - Q_0|/15$ `// error estimate`
if $E \leq \delta$ **then**
    `// tolerance is met; use current approximation`
    $Q = Q_1$
else
    `// tolerance is not met; split the interval and invoke adaptive quadrature`
    $[Q^{\text{L}}, E^{\text{L}}] = \text{adaptive\_simpsons}(f, a, c, \delta/2)$
    $[Q^{\text{R}}, E^{\text{R}}] = \text{adaptive\_simpsons}(f, c, b, \delta/2)$
    $Q = Q^{\text{L}} + Q^{\text{R}}$
    $E = E^{\text{L}} + E^{\text{R}}$
end



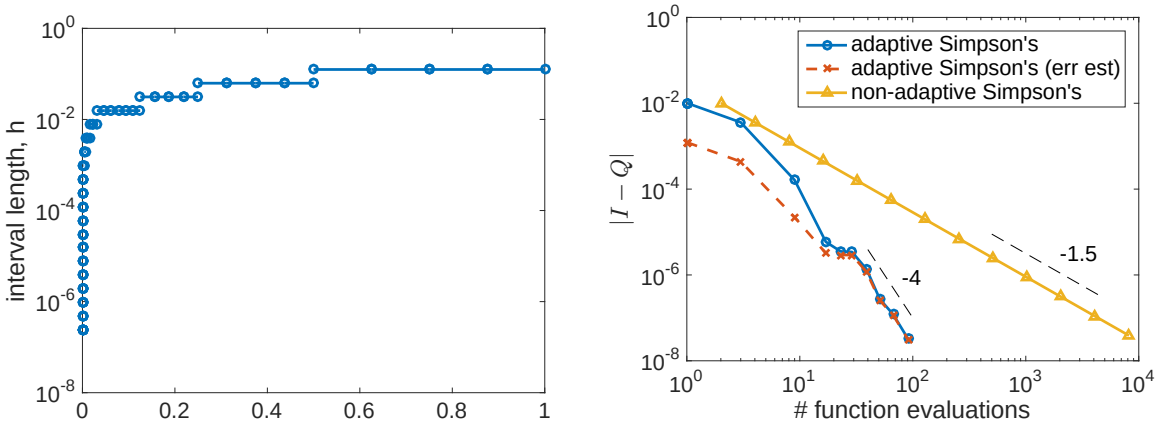
(a) integrand

(b) illustration ($\delta = 0.01$)

(c) interval spacing, $\delta = 10^{-6}$

(d) convergence

Figure 4.6: Adaptive Simpson's method applied to $\int_{-1}^{1} \cos((x+1.05)^{-1})dx$.



(a) interval spacing, $\delta = 10^{-6}$

(b) convergence

Figure 4.7: Adaptive Simpson's method applied to $\int_0^1 \sqrt{x}dx$.

intervals of a length less than 10^{-6} is employed in the vicinity of the singularity. Figure 4.7(b) shows the convergence result. We observe that, for the integrand with a singularity, the convergence rate of the non-adaptive Simpson's rule is limited to -1.5. On the other hand, the error of the adaptive method decays at the rate of -4 with respect to the number of function evaluations. This difference in the convergence rates for the non-adaptive and adaptive schemes result in a large difference in a number of function evaluations required to achieve a given error level, particularly for a small error tolerance.

4.9 Adaptive quadrature: modern methods

The adaptive Simpson's method we have presented is a "classical" adaptive quadrature method. Many of the more modern implementations of adaptive quadrature rules is based on Gauss-Kronrod quadrature — which builds on the Gauss quadrature we covered in the first half of the lecture — or Clenshaw-Curtis quadrature. As we have seen in the first half of this lecture, Gauss quadrature rules provide higher-order accuracy for a given number of points compared to Newton-Cotes rules; the higher-order accuracy is advantageous, especially when the formulation is augmented with an adaptive algorithm that selects the interval lengths in an intelligent manner. While many of the modern adaptive quadrature methods use a different set of quadrature rules, all adaptive quadrature rules are built on the same basic principles of error estimation and adaptive spacing selection, the principles we have illustrated in the context of adaptive Simpson's rule.

4.10 Summary

We provide a short summary of the lecture:

1. High-order Newton-Cotes rules, which are based on m equispaced quadrature points, may not converge with the integration order m due to the Runge's phenomenon.

2. Clenshaw-Curtis rules, which use the Chebyshev nodes as the quadrature points, do not suffer from the instability that the Newton-Cotes rules exhibit. For higher-order integration, we should hence consider the Clenshaw-Curtis rules instead of the Newton-Cotes rules.

3. Gauss quadrature rules select the quadrature weights *and* points to maximize the degree of polynomial integrated exactly for a given number of points.

4. The m-point Gauss quadrature rule exactly integrates polynomials of degree $2m-1$. This is higher than the m-point Newton-Cotes or Clenshaw-Curtis rule which integrate exactly polynomials of degree $m-1$ or m (depending on if m is even or odd).

5. Gauss (or Clenshaw-Curtis) quadrature rules converge very rapidly for smooth functions as the quadrature degree increases; however, the convergence can be slow for non-smooth functions.

6. Adaptive quadrature rules automatically evaluate the definite integral to desired tolerance.

7. Key ingredients of adaptive quadrature rules are an error estimate and an adaptive interval-selection scheme.

8. The integration error associated with the adaptive Simpson's rule converge rapidly even when the integrand exhibit a significant variation over a small region or the integrand is singular.

4.11 Appendix

In this appendix we prove that the error in the error estimate of the adaptive Simpson's method over a single interval $[a,b]$ is of $\mathcal{O}((b-a)^6)$: i.e.,

$$||I - Q^{(2)}([a,b])| - E([a,b])| = \mathcal{O}((b-a)^6).$$

To see this, we first note that the actual error is given by

$$|I - Q^{(2)}_{\mathrm{NC}(3)}([a,b])| = \frac{|f^{(4)}(\xi)|}{2880}\frac{1}{16}(b-a)^5,$$

for some $\xi \in [a,b]$. We then note that the difference between $Q^{(2)}_{\mathrm{NC}(3)}(a,b)$ and $Q_{\mathrm{NC}(3)}(a,b)$ is

$$|Q^{(2)}_{\mathrm{NC}(3)}([a,b]) - Q_{\mathrm{NC}(3)}([a,b])| = \frac{|f^{(4)}(\xi)|}{2880}\frac{15}{16}(b-a)^5 + \mathcal{O}((b-a)^6).$$

for the same ξ. It thus follows that

$$\begin{aligned} E(a,b) &\equiv \frac{1}{15}|Q^{(2)}_{\mathrm{NC}(3)}([a,b]) - Q_{\mathrm{NC}(3)}([a,b])| = \frac{|f^{(4)}(\xi)|}{2880}\frac{1}{16}(b-a)^5 + \mathcal{O}((b-a)^6) \\ &= |I - Q^{(2)}_{\mathrm{NC}(3)}([a,b])| + \mathcal{O}((b-a)^6), \end{aligned}$$

which is the desired result.

Lecture 5

Linear algebra: an interpretation

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

In this short note we provide various interpretations of techniques that are often introduced in an introductory course on linear algebra. The note assumes familiarity with basic concepts in linear algebra; the note is not complete in any sense.

5.1 Linear transformation

We consider the transformation of a vector $x \in \mathbb{R}^n$ by a matrix $A \in \mathbb{R}^{n\times n}$ and interpret it in a column-wise sense:

$$b = Ax = \begin{pmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = x_1 \begin{pmatrix} | \\ a_1 \\ | \end{pmatrix} + \cdots + x_n \begin{pmatrix} | \\ a_n \\ | \end{pmatrix},$$

where a_i is the i-th column of the matrix A and x_i is the i-th entry of the vector x. Note that our interpretation is

$$b = Ax\text{: linear combination of columns of } A \text{ with coefficients } x.$$

Visually, we may think of this multiplication as shown in Figure 5.1 for a 2×2 matrix,

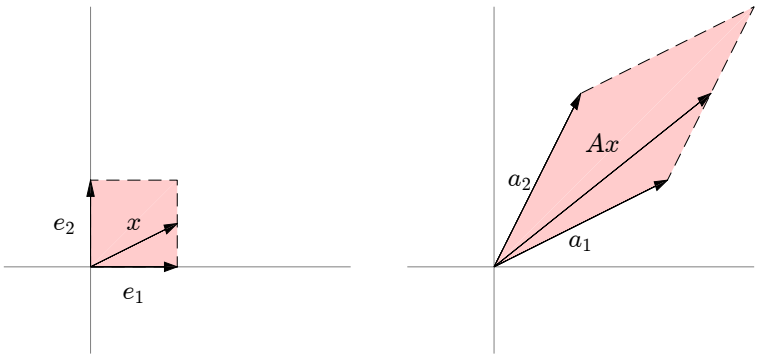
$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

We simply multiply the columns of A by the coefficients specified in x and add the resulting vectors. In a higher dimension, the operation can be thought of as a transformation of a hypercube.

Conversely, when we solve a linear system $Ax = b$ for the vector x, we may interpret the operation as follows:

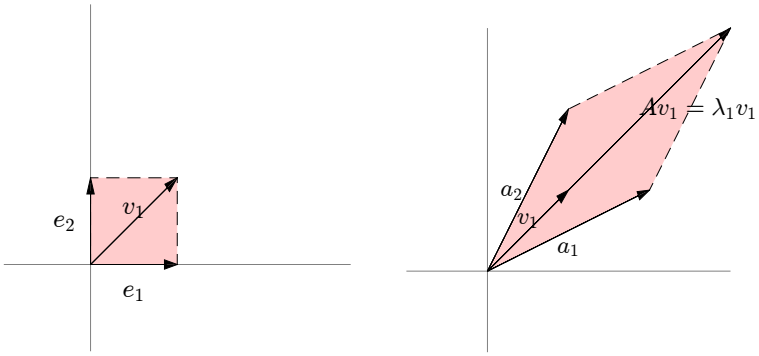
$$\text{Solve } Ax = b \text{ for } x\text{: find a vector of coefficients of the expansion of } b \text{ in the basis of columns of } A.$$

In other words, we are simply looking for coefficients for $\{a_1, \ldots, a_n\}$ such that the linear combination is equal to the right hand side b.



(a) x (b) $b = Ax$

Figure 5.1: Illustration of linear transformation for $A = [2, 1; 1, 2]$.



(a) x (b) $b = Ax$

Figure 5.2: Illustration of linear transformation applied to an eigenvector of $A = [2, 1; 1, 2]$. The eigenvector is $v_1 = [1; 1]$; the eigenvalue (i.e. "the scaling factor") is $\lambda_1 = 3$.

5.2 Eigenvalues and eigenvectors

Geometric interpretation. We call a vector $v \neq 0$ an *eigenvector* if $Av = \lambda v$ for some scalar λ; we then call this scalar λ the *eigenvalue*. In other words,

$$v \neq 0 \text{ is an eigenvector of } A \text{ if } Av \text{ is a scalar multiple of } v.$$

Figure 5.2 visually identifies one of the eigenvectors for the 2×2 matrix A introduced above. We note that the vector $v_1 = (1, 1)^T$ and Av_1 visually align; hence $v_1 = (1, 1)^T$ is an eigenvector. In addition, we note that the "scaling factor" is 3, i.e. Av_1 is 3 times longer than v_1; hence the associated eigenvalue is $\lambda_1 = 3$.

Algebraic interpretation (for completeness). Algebraically, the eigenvalue problem is the following: find $\lambda \in \mathbb{C}$ and non-zero $v \in \mathbb{C}^n$ such that

$$Av = \lambda v.$$

Note that this is equivalent to finding $v \neq 0$ such that

$$(A - \lambda I)v = 0.$$

We observe that in order for a non-zero v to satisfy the equation,

$$A - \lambda I \text{ must have a non-zero nullspace (aka kernel).}$$

This condition is equivalent to requiring that

$$\det(A - \lambda I) = 0.$$

The polynomial in λ in the left hand side, $p_A(\lambda) \equiv \det(A - \lambda I)$, is called the *characteristic polynomial*. Clearly, the characteristic polynomial is of degree n (since $A - \lambda I$ is $n \times n$). By the fundamental theorem of algebra, we can find n roots; we see that

$$\text{the } n \text{ roots of the characteristic polynomial } p_A(\lambda) \text{ are the } n \text{ eigenvalues of } A.$$

In general, these roots need not be distinct. Let us label the eigenvalues by superscripts: $\lambda_1, \lambda_2, \ldots, \lambda_n$. Once we find the eigenvalues, we can associate the i-th eigenvector v_i with the nullspace of $A - \lambda_i I$.

(Note: while the characterization of eigenvalues and eigenvectors by a characteristic polynomial and nullspace is useful for theoretical purposes, a typical numerical algorithm for solving eigenproblems is quite different from what is presented here.)

Example. We again look at

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

To find the eigenvalues (algebraically), we first find the roots of the characteristic polynomial

$$p_A(\lambda) = \det \begin{pmatrix} 2-\lambda & 1 \\ 1 & 2-\lambda \end{pmatrix} = (2-\lambda)^2 - 1 = \lambda_2 - 4\lambda + 3 = (\lambda - 3)(\lambda - 1) = 0;$$

we recognize that roots are 3 and 1. Without loss of generality, let us label them $\lambda_1 = 3$ and $\lambda_2 = 1$. To find eigenvectors (algebraically), we look for the nullspace of $A - \lambda_i I$:

$$A - \lambda_1 I = \begin{pmatrix} 2-3 & 1 \\ 1 & 2-3 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix} \quad \Rightarrow \quad v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$
$$A - \lambda_1 I = \begin{pmatrix} 2-1 & 1 \\ 1 & 2-1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \quad \Rightarrow \quad v_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

So, $v_1 = (1, 1)^T$ and $v_2 = (1, -1)^T$ are eigenvectors associated with λ_1 and λ_2, respectively. They are unique up to scaling.

Eigenvalue decomposition. We can now form a decomposition of a matrix A based on the eigenvectors and eigenvalues. To see this we observe that

$$A \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ \lambda_1 v_1 & \cdots & \lambda_n v_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix} \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}.$$

We now set

$$V = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix} \qquad \text{and} \qquad \Lambda = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix},$$

and observe that

$$AV = V\Lambda.$$

If V is invertible, we can multiply on the right by V^{-1} to obtain

$$A = V\Lambda V^{-1}.$$

Note that for V to be invertible, the eigenvectors that constitute V must be linearly independent. It follows that

the eigenvalue decomposition of A exists iff A has a linearly independent set of n eigenvectors.

We call such an A *diagonalizable*.

When does the eigendecomposition *not* work? Eigenvalue decomposition can fail for even a simple matrix. For instance, consider

$$B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

We observe

$$p_B(\lambda) = \det(B - \lambda I) = (1-\lambda)^2 = 0 \quad \Rightarrow \quad \lambda_1 = \lambda_2 = 1.$$

But, when we look at the nullspace of $B - \lambda_1 I$, we realize that

$$B - \lambda_1 I = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

We can only find one eigenvector, $v_1 = (1, 0)^T$. Hence, we cannot find $n = 2$ linearly independent eigenvectors: B is not diagonalizable.

5.3 Eigenvalues and eigenvectors for symmetric matrices

Properties. Symmetric matrices have very nice properties. In particular

1. eigenvalues are real
2. eigenvectors are orthogonal and span the entire $\mathbb{R}^n$.

As a consequence of the second property, symmetric matrices are always diagonalizable.

Sketch of proof of 1. Let $v \in \mathbb{C}^n$ and $\lambda \in \mathbb{C}$ be an eigenpair of $A \in \mathbb{R}^{n\times n}$. Then

$$v^*Av = v^*(Av) = v^*\lambda v = \lambda\|v\|^2,$$

where v^* denotes the conjugate transpose of v. We can take the conjugate of the scalar quantity (v^*Av) to obtain

$$(v^*Av)^* = \lambda^*\|v\|^2.$$

On the other hand,

$$(v^*Av)^* = v^*A^*v = v^*Av = v^*(Av) = \lambda v^*v = \lambda\|v\|^2,$$

where we have used $A^* = A^T = A$ for a real symmetric matrix. From these two equations, we see that

$$\lambda^*\|v\|^2 = \lambda^*(v^*Av)^* = \lambda\|v\|^2.$$

Since $\|v\| \neq 0$, it must be that $\lambda = \lambda^*$, which implies that λ is real.

Sketch of proof of 2. We consider case with distinct eigenvalues. (A proof for repeated eigenvalues is more involved.) Suppose (λ_i, v_i) and (λ_j, v_j) are eigenpairs of A and $\lambda_i \neq \lambda_j s$. Then

$$\lambda_i v_j^T v_i = v_j^T(\lambda_i v_i) = v_j^T A v_i = v_j^T A^T v_i = (Av_j)^T v_i = \lambda_j v_j^T v_i,$$

where the third equality follows from the symmetry of A. So

$$\lambda_i v_j^T v_i = \lambda_j v_j^T v_i;$$

since $\lambda_i \neq \lambda_j$, we must have $v_i^T v_j = 0$. The two eigenvectors are orthogonal.

Eigendecomposition. We now consider an eigendecomposition of a symmetric matrix A. Without loss of generality, we scale the eigenvectors such that $\|v_i\| = 1$, $\forall i$. Then,

$$A = V\Lambda V^{-1} \quad (= V\Lambda V^T),$$

where V is an orthogonal matrix of orthonormal eigenvectors. Before we provide a geometric interpretation of this decomposition, let us provide a geometric interpretation of orthogonal matrices.

5.4 Orthogonal matrices

Definition. We recall that a matrix

$$Q = \begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix}$$

is said to be *orthogonal* if the vectors $\{q_1, \ldots, q_n\}$ form an *orthonormal basis*. We recall that a set of vectors are orthonormal if

$$q_i^T q_j = \begin{cases} 1, & i = j \\ 0, & i \neq j. \end{cases}$$

Inverse. The inverse of an orthogonal matrix is particularly simple to compute:

$$Q^{-1} = Q^T.$$

To see this, we observe that

$$Q^TQ = \begin{pmatrix} - & q_1^T & - \\ & \vdots & \\ - & q_n^T & - \end{pmatrix}\begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix} = \begin{pmatrix} q_1^Tq_1 & \cdots & q_1^Tq_n \\ \vdots & & \vdots \\ q_n^Tq_1 & \cdots & q_n^Tq_n \end{pmatrix} = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix},$$

which implies $Q^T = Q^{-1}$.

Interpretation: change of basis. The interpretation of linear transformation we introduced in the context of a general A also applies to an orthogonal matrix Q. In particular,

$$Qx\text{: linear combination of columns of } Q \text{ with coefficients } x$$

and

$$Q^T b \ (= Q^{-1}b)\text{: vector of coefficients of the expansion of } b \text{ in the basis of columns of } Q.$$

In addition, due to the use of orthonormal basis, we can interpret the i-th component of $Q^T b$ — the i-th coefficient of the expansion of b in the basis of columns of Q — as

$$(Q^T b)_i = q_i^T b\text{: component of } b \text{ that lies in the direction of } q_i.$$

This allows us to represent the vector b in various basis. For instance, in the $n = 2$ case, the regular "b" should be interpreted as

$$b = \underbrace{b_1}_{\text{coeff.}} \underbrace{e_1}_{\substack{\text{basis}\\ \text{vector}}} + b_2 e_2 : \text{ vector } b \text{ expressed in canonical basis } \{e_1, e_2\}.$$

On the other hand we may also express b as

$$b = \underbrace{(q_1^T b)}_{\text{coeff.}} \underbrace{q_1}_{\substack{\text{basis}\\ \text{vector}}} + (q_2^T b) q_2 : \text{ vector } b \text{ expressed in basis } \{q_1, q_2\}.$$

(Proving the equivalence is straightforward:

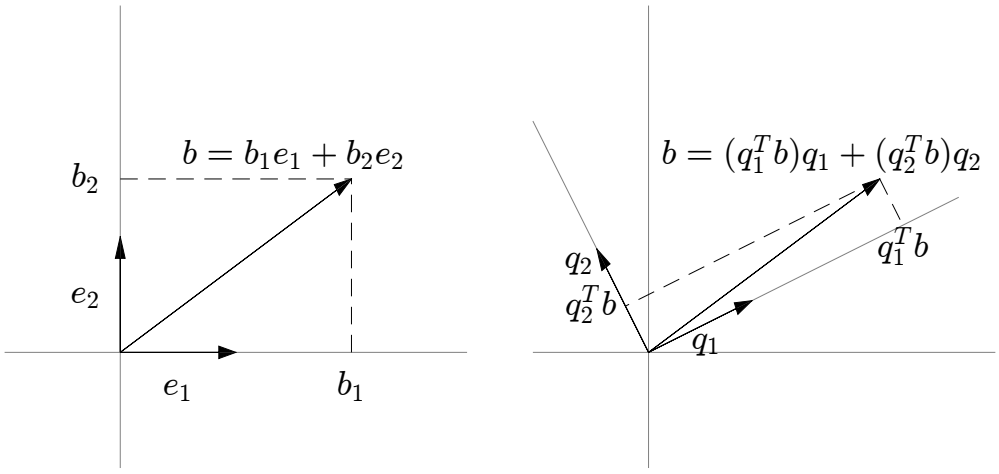
$$\begin{aligned} q_1(q_1^T b) + \cdots + q_n(q_n^T b) &= \begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix} \begin{pmatrix} q_1^T b \\ \vdots \\ q_n^T b \end{pmatrix} \\ &= \begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix} \begin{pmatrix} - & q_1^T & - \\ & \vdots & \\ - & q_n^T & - \end{pmatrix} b = QQ^T b = b. \ ) \end{aligned}$$

These two expression for the same vector b — one using canonical orthonormal basis $\{e_1, e_2\}$ and another using an arbitrary orthonormal basis $\{q_1, q_2\}$ — is illustrated in Figure 5.3. Again, we see b as a linear combination of columns of Q with appropriate coefficients. For an orthonormal basis, each coefficient is simply given by $(q_1^T b)$; note that b_1 for the canonical basis also arise in this fashion: $b_1 = (e_1^T b)$.

2-norm of Qx. The 2-norm of a vector Qx is the same as that of x. To see this, we observe

$$\|Qx\|^2 = (Qx)^T (Qx) = x^T Q^T Q x = x^T x = \|x\|^2,$$

where we have used $Q^T Q = I$.



(a) $b = b_1e_1 + b_2e_2$ (b) $b = (q_1^Tb)q_1 + (q_2^Tb)q_2$

Figure 5.3: Illustration of representing the vector b using two different orthonormal basis.

5.5 Eigenvalue decomposition of symmetric matrices

We now provide a geometric interpretation of linear operators to eigenvalue decomposition of symmetric matrices, which have orthogonal eigenvectors. Specifically, upon the decomposition $A = V\Lambda V^T$, the multiplication Ax can be thought of as $Ax = V(\Lambda(V^Tx))$, where

$$\begin{aligned} V^Tx &: \text{vector of coefficients of } x \text{ in basis } \{v_1, \ldots, v_n\} \\ &\equiv \text{vector of components of } x \text{ that lie in directions } \{v_1, \ldots, v_n\} \\ \Lambda V^Tx &: \text{the coefficients scaled by eigenvalues} \\ V\Lambda V^Tx &: \text{linear combination of } \{v_1, \ldots, v_n\} \text{ with the scaled coefficients.} \end{aligned}$$

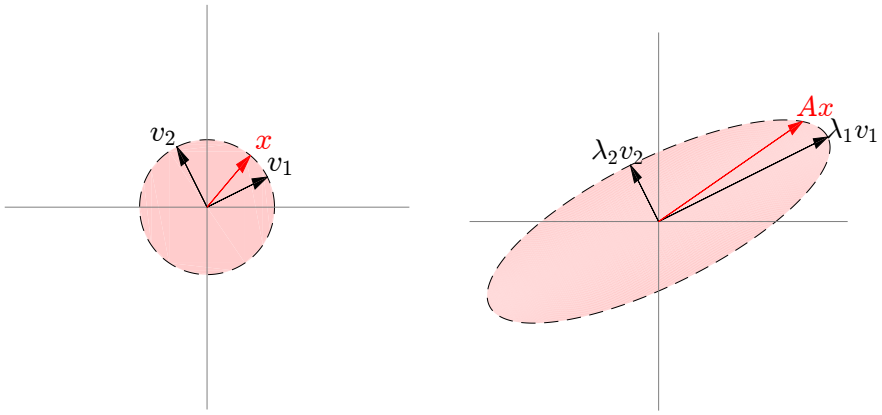
In other words,

$$Ax = \underbrace{(\lambda_1 \overbrace{(v_1^Tx)}^{\text{coeff}})}_{\text{scaled coeff}} \underbrace{v_1}_{\substack{\text{basis} \\ \text{vector}}} + (\lambda_2(v_2^Tx))v_2 + \cdots + (\lambda_n(v_n^Tx))v_n.$$

A visual representation of this geometric interpretation is shown in Figure 5.4. Given a vector x, we first express the vector in the orthonormal basis of eigenvectors $\{v_1, v_2\}$; i.e. $x = (v_1^Tx)v_1 + (v_2^Tx)v_2$. We then scale the coefficients v_1^Tx and v_2^Tx by respective eigenvalues to obtain $\lambda_1(v_1^Tx)$ and $\lambda_2(v_2^Tx)$. We finally consider a linear combination of $\{v_1, v_2\}$ with these scaled coefficients to obtain

$$Ax = \underbrace{(\lambda_1 \overbrace{(v_1^Tx)}^{\text{coeff}})}_{\text{scaled coeff}} \underbrace{v_1}_{\substack{\text{basis} \\ \text{vector}}} + (\lambda_2(v_2^Tx))v_2.$$

We see that λ_1 is the maximum "stretching" (or scaling) provided by the matrix A; λ_n ($= \lambda_2$) it the minimum "stretching" provided by the matrix A.



(a) (b)

Figure 5.4: Illustration of linear transformation interpreted in terms of eigenvalue decomposition.

Symmetric positive definite (SPD) matrices. A symmetric matrix A is said to be SPD if

$$x^T Ax > 0 \quad \forall x \neq 0.$$

An equivalent condition is

A is SPD: all eigenvalues of a symmetric matrix A are positive.

To see the equivalence, we observe that

$$x^T Ax = x^T V \Lambda V^T x = (V^T x)^T \Lambda (V^T x) = x' \Lambda x'$$

for the change of basis $x' = V^T x$. We observe that $x^T Ax$ is positive as long as all entries in $\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ are positive. Intuitively, "stretching" provided by a SPD matrix is all positive and finite.

Towards SVD. We can relate our geometric interpretation provided by eigendecomposition to various concepts in linear algebra, such as image, kernel, invertibility, operator norms, and conditioning. But, before we continue, let us introduce a more generalized decomposition which allows us to provide the same interpretation for non-symmetric matrices.

5.6 Singular Value Decomposition (SVD)

Arguably two limitations of the eigenvalue decomposition are

1. the decomposition only exists for diagonalizable matrices
2. eigenvectors are orthogonal only if A is symmetric.

The *singular value decomposition*, which exists for any matrix (in fact even for non-square matrices even though we will not consider them in this lecture), in some sense address these two issues. The singular value decomposition of $A \in \mathbb{R}^{n \times n}$ is

$$A = U \Sigma V^T,$$

where

$$U = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_n \\ | & & | \end{pmatrix}, \quad \Sigma = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{pmatrix}, \quad V = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_n \\ | & & | \end{pmatrix},$$

and U and V are orthogonal matrices. The terms of the diagonal, $\{\sigma_1, \ldots, \sigma_n\}$, are the *singular values*, the vectors $\{u_1, \ldots, u_n\}$ are the *left singular vectors*, and the vectors $\{v_1, \ldots, v_n\}$ are the *right singular vectors*. Without loss of generality, we order the singular values in non-increasing order

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

Singular values are non-negative by definition. The SVD has a structure very similar to the eigen-decomposition of a symmetric matrix A, except that it uses two different sets of orthonormal basis, $\{u_1, \ldots, u_n\}$ and $\{v_1, \cdots, v_n\}$. As noted, any matrix A admits a singular value decomposition. If the matrix A is symmetric positive definite, then the SVD of A has $U = V$, and hence the SVD is the same as the eigenvalue decomposition.

Given the SVD of A, $A = U\Sigma V^T$, the multiplication Ax can be thought of as $Ax = U(\Sigma(V^T x))$, where

$$\begin{aligned} V^T x &: \text{vector of coefficients of } x \text{ in basis } \{v_1, \ldots, v_n\} \\ &\equiv \text{ vector of components of } x \text{ that lie in directions } \{v_1, \ldots, v_n\} \\ \Sigma V^T x &: \text{the coefficients scaled by singular values} \\ U\Sigma V^T x &: \text{linear combination of } \{u_1, \ldots, u_n\} \text{ with the scaled coefficients.} \end{aligned}$$

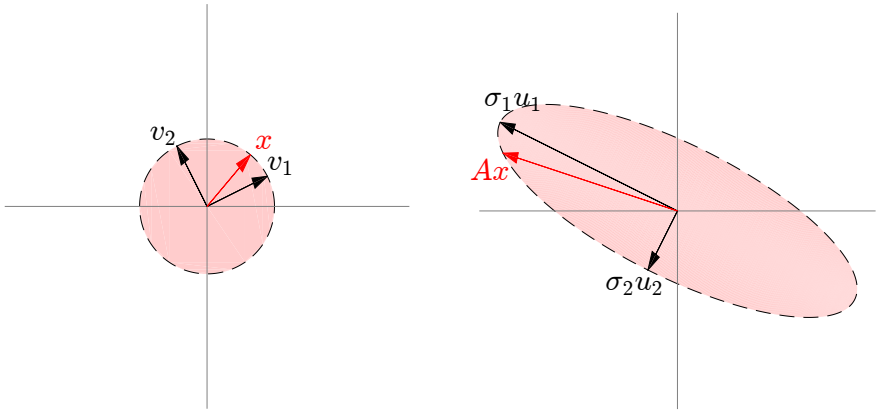
In other words,

$$Ax = \underbrace{(\sigma_1 \overbrace{(v_1^T x)}^{\substack{\text{coeff for} \\ \text{right basis}}})}_{\text{scaled coeff}} \underbrace{u_1}_{\substack{\text{left basis} \\ \text{vector}}} + (\sigma_2(v_2^T x))u_2 + \cdots + (\sigma_n(v_n^T x))u_n.$$

A visual representation of this geometric interpretation is shown in Figure 5.5. Given a vector x, we first express the vector in the "right" orthonormal basis $\{v_1, v_2\}$; i.e. $x = (v_1^T x)v_1 + (v_2^T x)v_2$. We then scale the coefficients $v_1^T x$ and $v_2^T x$ by respective singular values to obtain $\sigma_1(v_1^T x)$ and $\sigma_2(v_2^T x)$. We finally consider a linear combination of the "left" orthonormal basis $\{u_1, u_2\}$ with these scaled coefficients to obtain

$$Ax = \underbrace{(\sigma_1 \overbrace{(v_1^T x)}^{\substack{\text{coeff for} \\ \text{right basis}}})}_{\text{scaled coeff}} \underbrace{u_1}_{\substack{\text{left basis} \\ \text{vector}}} + (\sigma_2(v_2^T x))u_2.$$

Again, the interpretation is the same as that associated with the eigendecomposition of a symmetric matrix, except now we use two different set of orthonormal basis. We see that σ_1 is the maximum "stretching" (or scaling) provided by the matrix A; σ_n ($= \lambda_2$) it the minimum "stretching" provided by the matrix A.



(a) (b)

Figure 5.5: Illustration of linear transformation interpreted in terms of singular value decomposition.

5.7 SVD and properties of matrices

The singular value decomposition allows us to readily identify various matrix properties. [Note that due to the equivalence of SVD and eigendecomposition for symmetric positive definite matrices, the discussion here also applies to the eigendecomposition of SPD matrices.]

Preliminary. Recalling that singular values are set in non-increasing order and that all singular values are non-negative, we may denote the last non-zero singular value by σ_r. More explicitly,

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0,$$

and

$$\sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_n = 0.$$

(Note that if there is no zero singular value, then $r = n$.)

Image (or range) of A. We recall that the image of a matrix A is defined as the subspace $\mathbb{R}^n$ spanned by the columns of A:

$$\text{Img}(A) = \text{span}\{a_1, \ldots, a_n\}.$$

Equivalently, we can interpret Img(A) as

$$\text{Img}(A) : \text{ a space of vectors that can be expressed as } Ax \text{ for some coefficients } x.$$

Now we recall that for $A = U\Sigma V^T$, we can express the product $y = Ax$ as

$$\begin{aligned} y = Ax &= (\sigma_1(v_1^T x))u_1 + \cdots + (\sigma_r(v_r^T x))u_r + (\sigma_{r+1}(v_{r+1}^T x))u_{r+1} + \cdots + (\sigma_n(v_n^T x))u_n \\ &= (\sigma_1(v_1^T x))u_1 + \cdots + (\sigma_r(v_r^T x))u_r. \end{aligned}$$

Here, we have dropped the scaled coefficients associated with $\sigma_{r+1}, \ldots, \sigma_n$ because they are zero. Because $\{u_1, \ldots, u_r\}$ is orthonormal and $\sigma_1, \ldots, \sigma_r$ are non-zero, we readily observe that

$$\text{Img}(A) = \text{span}\{u_1, \ldots, u_r\}.$$

Again, if all singular values are non-zero, then $\text{Img}(A) = \text{span}\{u_1, \ldots, u_n\} = \mathbb{R}^n$.

Rank of A. The rank of A is the dimension of the image of A. Because $\{u_1, \ldots, u_r\}$ is orthonormal, we have

$$\text{Rank}(A) = \dim(\text{span}\{u_1, \ldots, u_r\}) = r.$$

Kernel (or nullspace) of A. The kernel of a matrix A is defined as

$$\text{Ker}(A) = \{x \in \mathbb{R}^n : Ax = 0\}.$$

Let us represent a vector x in the right singular basis $\{v_1, \ldots, v_n\}$: $x = Vc = v_1c_1 + \cdots v_nc_n$, where c is the vector of associated coefficients. Then we observe that

$$Ax = (\sigma_1(v_1^T Vc))u_1 + \cdots + (\sigma_n(v_n^T Vc))u_n = (\sigma_1 c_1)u_1 + \cdots + (\sigma_n c_n)u_n.$$

Because the basis $\{u_1, \ldots, u_n\}$ is orthonormal, all the scaled coefficients must be zero for $Ax = 0$. This is only possible if $c_1 = \cdots = c_r = 0$; on the other hand, $\{c_{r+1}, \ldots, c_n\}$ can be set to any values, since $\sigma_{r+1} = \cdots = \sigma_n = 0$. Hence all vectors in $\text{Ker}(A)$ can be expressed as $v_{r+1}c_{r+1} + \cdots + v_nc_n$ for some coefficients c. Equivalently,

$$\text{Ker}(A) = \text{span}\{v_{r+1}, \ldots, v_n\}.$$

Again, if all singular values are non-zero (and hence $r = n$), then $\text{Ker}(A) = 0$.

Invertiblity of A. A matrix A is invertible if it is full rank. From the argument above, we readily see that we must have $r = n$, which is equivalent to the following:

$$A \text{ is invertible: all singular values are non-zero.}$$

We can also provide a geometric interpretation. We recall that

$$\text{Solve } Ax = b \text{ for } x\text{: find a vector of coefficients of the expansion of } b \text{ in the basis of columns of } A.$$

In the context of SVD, we are looking for scaled coefficients $\sigma_1(v_1^T x), \ldots, \sigma_n(v_n^T x)$ for the basis $\{u_1, \ldots, u_n\}$ such that

$$b = (\sigma_1(v_1^T x))u_1 + \cdots + (\sigma_n(v_n^T x))u_n.$$

In order to represent any b in $\mathbb{R}^n$, we must be able to set the scaled coefficients freely. This is possible as along as all the singular values are non-zero; if a singular value is zero, then we can only set the associated scaled coefficient to zero.

More intuitively, we recall that singular values represent "stretching" (or scaling) in different directions. If there is a direction in which the "stretching" is zero — that is, if the component in a given direction vanishes after the application of A — then we have no means of deducing the component of x in that direction simply by looking at Ax because that information is lost by the zero scaling. Hence, all singular values must be non-zero — the "strething" must be finite — for a matrix to be invertible.

Inverse of A. Given a SVD $A = U\Sigma V^T$ with all non-zero singular values, the inverse of A is simply given by

$$A^{-1} = (U\Sigma V^T)^{-1} = (V^T)^{-1}\Sigma^{-1}U^{-1} = V\Sigma^{-1}U^T,$$

where we have used the fact that $V^{-1} = V^T$ for orthogonal matrices. Intuitively, the roles of the left and right singular vectors are switched, and we get reciprocal "stretchings".

Induced norm of A. We recall that the 2-norm of A is defined as

$$\|A\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|Ax\|_2}{\|x\|_2}.$$

But we readily observe that

$$\|A\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|Ax\|_2}{\|x\|_2} = \max_{x \in \mathbb{R}^n} \frac{\|U\Sigma V^T x\|_2}{\|x\|_2} = \max_{x' \in \mathbb{R}^n} \frac{\|U\Sigma x'\|_2}{\|Vx'\|_2} = \max_{x' \in \mathbb{R}^n} \frac{\|\Sigma x'\|_2}{\|x'\|_2} = \sigma_1.$$

Recalling that σ_1 is the maximum "stretching" (or scaling) induced by a matrix A, $\|A\|_2$ measures the maximum "stretching" induced by A.

Induced norm of A^{-1}. We similarly observe that

$$\|A^{-1}\|_2 = \max_{x \in \mathbb{R}^n} \frac{\|A^{-1}x\|_2}{\|x\|_2} = \max_{x \in \mathbb{R}^n} \frac{\|V\Sigma^{-1}U^T x\|_2}{\|x\|_2} = \max_{x' \in \mathbb{R}^n} \frac{\|\Sigma^{-1}x'\|_2}{\|x'\|_2} = \frac{1}{\sigma_n}.$$

Recalling that σ_n is the minimum "stretching" (or scaling) induced by a matrix A, $\|A^{-1}\|_2$ measures the (reciprocal of the) minimum "stretching" induced by A.

Lecture 6

Linear systems: LU factorization

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

6.1 Introduction

In this lecture, we consider numerical solution of linear systems. Specifically, we introduce systematic approaches to find a unique solution x to

$$Ax = b,$$

where A is an $n \times n$ non-singular matrix, b is an n-vector, and x is an n-vector. The approaches we consider are categorized as *direct methods* (as opposed to *iterative methods*) as they provide the solution in a finite number of steps.

6.2 Solution of lower triangular systems: forward substitution

We first consider cases where the matrix possesses a special structure which makes the solution of the linear system particularly simple. A matrix L is said to be *lower triangular* if all the entries above the diagonal are zero: i.e.,

$$l_{ij} = 0 \quad \forall j > i.$$

For instance, 3×3 lower triangular system is of the form

$$\begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

Note that L is non-singular if and only if the diagonal entries are non-zero. In this case, we can find the entries of the solution in a sequential manner:

$$\begin{aligned} x_1 &= b_1/l_{11} \\ x_2 &= (b_2 - l_{21}x_1)/l_{22} \\ x_3 &= (b_3 - l_{31}x_1 - l_{32}x_2)/l_{33}. \end{aligned}$$

This solution strategy is called *forward substitution*. Hence, (non-singular) lower-triangular system can be solved systematically using *forward substitution*.

More generally, for an $n \times n$ lower triangular system, the forward substitution algorithm is the following:

Algorithm 2: forward substitution

```
for i = 1, ..., n do
    x_i = b_i
    for j = 1, ..., i − 1 do
        x_i = x_i − l_ij x_j
    end
    x_i = x_i / l_ii
end
```

Operation count. We now analyze the cost of forward substitution. The inner loop of forward substitution requires two FLOPs: one multiplication and one subtraction. The total FLOP count for the forward substitution is hence

$$\sum_{i=1}^{n}(1+\sum_{j=1}^{i-1}2) \approx n^2.$$

The cost of forward substitution hence scales quadratically with the size of the linear system. For instance, the solution of a 10×10 system requires ≈ 100 operations, whereas the solution of a 100×100 system requires 10000 operations.

6.3 Solution of upper triangular systems: backward substitution

We can use a similar technique to solve an upper triangular system. A matrix U is said to be *upper triangular* if all the entries below the diagonal are zero: i.e.,

$$u_{ij} = 0 \quad \forall i > j.$$

For instance, 3×3 upper triangular system is of the form

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}.$$

The matrix U is non-singular if and only if the diagonal entries are non-zero. We can again find the entries of the solution in a sequential manner:

$$\begin{aligned} x_3 &= b_3/u_{33} \\ x_2 &= (b_2 - u_{23}x_3)/u_{22} \\ x_1 &= (b_1 - u_{12}x_2 - u_{13}x_3)/u_{33}. \end{aligned}$$

This solution strategy is called *backward substitution*. Hence, (non-singular) upper-triangular system can be solved systematically using *backward substitution*.

More generally, for an $n \times n$ upper triangular system, the backward substitution algorithm is the following:

Algorithm 3: backward substitution

```
for i = n, ..., 1 do
    x_i = b_i
    for j = i + 1, ..., n do
        x_i = x_i − u_ij x_j
    end
    x_i = x_i / u_ii
end
```

Operation count. We readily observe that the FLOP count for the backward substitution is identical to that for the forward substitution: $\approx n^2$. The cost of backward substitution hence scales quadratically with the size of the linear system.

While forward and backward substitution are efficient methods to solve lower- and upper-triangular systems, respectively, they cannot be used for general matrices. In the next section, we consider a systematic approach to decompose a general matrix A into a pair of lower- and upper-triangular matrices.

6.4 Gaussian elimination: 3×3 example

We first recall *Gaussian elimination*, a systematic procedure to reduce a general linear system to an upper triangular system. To illustrate the procedure, we consider a concrete 3×3 system:

$$\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 4 \end{pmatrix}.$$

Step 1. Eliminate entries in the first column below the diagonal:

$$\begin{array}{l} \\ -2 \times (\text{row1}) \\ -4 \times (\text{row1}) \end{array} \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 4 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

Step 2. Eliminate entries in the second column below the diagonal:

$$-3 \times (\text{row2}) \quad \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$$

$$\Downarrow$$

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 3 \end{pmatrix}$$

Step 3. Solve the upper triangular system using backward substitution:

$$\begin{aligned} x_3 &= 3/3 = 1 \\ x_2 &= (-1 - 1 \cdot x_3)/1 = (-1 - 1 \cdot 1)/1 = -2 \\ x_1 &= (1 - 1 \cdot x_2 - 1 \cdot x_3)/2 = (1 - 1 \cdot (-2) - 1 \cdot 1)/2 = 1. \end{aligned}$$

The Gaussian elimination reduces a matrix to an upper triangular form and also identifies the associated right hand side such that the solution is not modified.

6.5 LU factorization: 3×3 example

Gaussian elimination provides the solution of a linear system in a systematic manner. In fact, by recording the row manipulation operations used to generate the upper triangular system, we can provide a factorization of matrix A into a lower triangular matrix L and an upper triangular matrix U: $A = LU$. Once we have the factorization, we can solve the problem for any right hand side b using a two-step procedure: first solve using forward substitution

$$Lz = b$$

for z. Then solve using backward substitution

$$Ux = z$$

for x. The x is the solution to $Ax = b$, because the substitution of $z = Ux$ to the first equation yield $LUx = b$, which of course is equal to $Ax = b$.

We now present the LU factorization for the 3×3 matrix we introduced above:

Step 1. Eliminate entries in the first column below the diagonal, and record the operation in a lower triangular matrix M_1:

$$\underbrace{\begin{pmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{pmatrix}}_{M_1} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 6 \end{pmatrix}}_{M_1 A}$$

Step 2. Eliminate entries in the second column below the diagonal, and record the operation in a lower triangular matrix M_2:

$$\underbrace{\begin{pmatrix} 1 & & \\ & 1 & \\ & -3 & 1 \end{pmatrix}}_{M_2} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 3 & 6 \end{pmatrix}}_{M_1 A} = \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix}}_{M_2 M_1 A}$$

We now note that our upper triangular matrix is in fact given by $M_2 M_1 A = U$. In addition, we make the following critical observations:

Observation 1. The inverse of the matrices M_1 and M_2 are obtained by simply negating their off-diagonal entries:

$$M_1^{-1} = \begin{pmatrix} 1 & & \\ -2 & 1 & \\ -4 & & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & & 1 \end{pmatrix}$$

$$M_2^{-1} = \begin{pmatrix} 1 & & \\ & 1 & \\ & -3 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & \\ & 1 & \\ & 3 & 1 \end{pmatrix}.$$

Observation 2: The product $M_1^{-1} M_2^{-1}$ is a lower triangular matrix, and the product is obtained by simply "collecting" the off-diagonal entries:

$$M_1^{-1} M_2^{-1} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & 3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix}.$$

It follows that

$$A = (M_1^{-1} M_2^{-1})(M_2 M_1 A) = \underbrace{\begin{pmatrix} 1 & & \\ 2 & 1 & \\ 4 & 3 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ & 1 & 1 \\ & & 3 \end{pmatrix}}_{U}.$$

Hence, the off-diagonal entries of the matrix L simply records the factors by which the pivot row was multiplied to eliminate the entries below the pivot. We have now obtained an *LU factorization* of the matrix A such that $A = LU$.

6.6 LU factorization (without pivoting): $n \times n$

The LU factorization for $n \times n$ matrix is summarized in the algorithm below.

Algorithm 4: LU factorization (without pivoting)

```
1 U = A, L = I
2 for i = 1, ..., n − 1 do
3     for j = i + 1, ..., n do
4         l_ji = u_ji / u_ii
5         for k = i, ..., n do
6             u_jk = u_jk − l_ji u_ik
7         end
8     end
9 end
```

Operation count. We now analyze the cost of LU factorization. The most inner loop of LU factorization requires two FLOPs: one multiplication and one subtraction. The total FLOP count for the LU factorization is hence

$$\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \sum_{k=i}^{n} 2 \approx \frac{2}{3} n^3.$$

The LU factorization scales as the third power of the size of the linear system. For instance, the factorization of a 10×10 system requires $\approx \frac{2}{3} \cdot 10^3$ operations, whereas the factorization of a 100×100 system requires $\approx \frac{2}{3} \cdot 10^6$ operations.

Solution of a linear system. Given an LU factorization of A such that $A = LU$, we can find the solution to $Ax = b$ as follow (as also noted in the beginning). We first solve

$$Lz = b$$

using forward substitution. We then solve

$$Ux = z$$

using backward substitution.

6.7 Summary

We summarize the key points of this lecture:

1. A lower triangular system can be solved using forward substitution. The operation count for a $n \times n$ system is $\approx n^2$.

2. An upper triangular system can be solved using backward substitution. The operation count for a $n \times n$ system is $\approx n^2$.

3. LU provides a factorization of an $n \times n$ matrix A into a lower triangular matrix L and an upper triangular matrix U such that $A = LU$. The operation count for LU factorization is $\approx \frac{2}{3} n^3$.

Warning. We make one important note. The LU factorization presented in this lecture, while appearing reasonable, is missing one key ingredient: *pivoting*. In the next lecture we will study the conditioning of linear systems, the stability of LU factorization without pivoting, and the LU factorization with pivoting. As we will see, the pivoting is crucial for the stability of LU factorization.

Lecture 7

Linear systems: conditioning and stability

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

7.1 Introduction

In solving linear systems using MATLAB, you might have encountered the following message:

```
>> x = A\b;
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.000000e-16.
```

In this lecture, we will study the conditioning of linear systems which will shed light on the cause of the above warning message. We will then study the stability of LU factorization, and make modifications to the "vanilla" LU factorization introduced in the previous lecture to improve the stability.

7.2 Scientific notation

In a typical (digital) computer, numbers are represented as *floating-point numbers* using a finite number of bits. Before we provide a formal definition of floating-point arithmetic, we first relate floating-point numbers to something we are familiar: scientific notation. Real numbers expressed in scientific notation are of the form

$$\tilde{x} = \pm m \times 10^e,$$

where m is the *mantissa*, which is a real number in the range $[1, 10)$, and e is the *exponent*, which is an integer. (We treat "0" as a special case since it does not fit in the above form.) For instance, $+123.45$ is expressed as $+1.2345 \times 10^2$. We make two observations:

1. The number of digits in the mantissa m controls the *relative precision* of the number.

2. The range of the exponent e controls the range of numbers that can be represented.

We elaborate on the first observation: scientific notation controls the *relative* precision, not the *absolute* precision. To this end, we consider a concrete case where we use four digits to represent m. In this case, the sequence of distinct numbers expressed in scientific notation that begins with $0.1234 = 1.234 \times 10^{-1}$ are

$$1.234 \times 10^{-1}, \quad 1.235 \times 10^{-1}, \quad 1.236 \times 10^{-1}, \ldots;$$

we have the *absolute* precision of $0.001 \times 10^{-1} = 0.0001$. On the other hand, the sequence of distinct numbers expressed in scientific notation that begins with $123400 = 1.234 \times 10^5$ are

$$1.234 \times 10^5, \quad 1.235 \times 10^5, \quad 1.236 \times 10^5, \ldots;$$

we have the *absolute* precision of $0.001 \times 10^5 = 100$. Clearly, the *absolute* precision of the scientific notation varies depending on the value of the exponent; however, the *relative* precision is preserved regardless of the exponent. More precisely, for any real number x, which may have more than four significant digits, there exists $\tilde{x}$ expressed using four-digit scientific notation such that

$$\tilde{x} = x(1 + \epsilon)$$

for some $\epsilon < 0.001$.

7.3 Floating point arithmetic

We now formally introduce floating-point numbers. Floating-point numbers are of the form

$$\tilde{x} = \pm(m/\beta^t) \times \beta^e,$$

where $\beta \in \mathbb{Z}_{>0}$ is the *base*, $m \in \mathbb{Z}$ is in the range $[\beta^{t-1}, \beta^t - 1]$, $t \in \mathbb{Z}_{>0}$ is the *precision*, and $e \in \mathbb{Z}$ is the *exponent*. The quantity $\pm(m/\beta^t)$ is called the *mantissa*; note that $m/\beta^t \in [1/\beta, 1 - 1/\beta^t]$. We make a few observations:

1. The precision t determines the number of "significant figures" in the mantissa $\pm(m/\beta^t)$, which in turn controls the relative precision of floating-point numbers.

2. The range of the exponent e determines the range of numbers that can be represented.

We note that the scientific notation with four digits is a special case of floating-point number representation with $\beta = 10$ and $t = 4$. (With the above convention the mantissa takes on a value in $[0.1, 1)$, which is shifted by a factor of 10 compared to the standard scientific notation considered in Section 7.2.) The IEEE double-precision format uses $\beta = 2$, $t = 53$, and $e = 11$; for $\beta = 2$, there are precisely β^{t-1} integers in the range of m (i.e., $2^t - 2^{t-1} = 2^{t-1}$), and hence m can be stored in $t - 1 = 52$ bits. Hence, the IEEE double-precision format uses 52 bits for m, 11 bits for e, and 1 bit for the sign to store a floating-point number in 64 bits.

Similar to a real number represented in scientific notation, floating-point arithmetic controls the *relative* precision and not the *absolute* precision. More precisely, for any real number x, there exists $\tilde{x}$ in the floating-point number system such that

$$\tilde{x} = x(1 + \epsilon)$$

for some $\epsilon < \epsilon_{\text{machine}}$, where $\epsilon_{\text{machine}}$ is called the *machine precision*. The machine precision can be equivalently defined as the smallest number such that

$$1 \quad \text{and} \quad 1 + \epsilon_{\text{machine}}$$

are distinct in the given floating-point arithmetic. For the IEEE double precision format, the machine precision is $\epsilon_{\text{machine}} \approx 2^{-52} \approx 10^{-16}$. Hence, while the two numbers

$$1 \quad \text{and} \quad 1.0001 = 1 + 10^{-4}$$

are distinguishable, the numbers

$$1 \quad \text{and} \quad 1.00000000000000000001 = 1 + 10^{-20}$$

are not distinguishable; these two numbers are identical in double-precision floating-point arithmetic. This in turn means that any number expressed in a floating point representation in fact has a relative error of an order $\epsilon_{\text{machine}}$.

The double-precision floating-point arithmetic provides sufficient (relative) precision for most situations. However, poorly designed algorithm can be affected by rounding errors. In fact, being stable and not affected by rounding errors is one of the fundamental criteria of a usable algorithm.

7.4 Condition number

We now introduce the concept of *condition number* which characterizes the stability of a linear system. The condition number plays an important role in the characterization of algorithms that "approximately" solve a linear system $Ax = b$. This "approximation" might arise because we consider a solution method that is designed to solve the equation approximately (even in infinite-precision arithmetic). The "approximation" may also arise because all computations performed using floating point arithmetic is in the end an approximation to the exact arithmetic. In both cases, we wish to analyze how a small perturbation to a linear system can affect the accuracy of our solution.

Towards this end, given an original system $Ax = b$, we consider a perturbed system

$$A(x + \delta x) = b + \delta b.$$

Here, δb represents a small perturbation to the equation, and δx represents the associated perturbation in the solution; we observe $A\delta x = \delta b$ since $Ax = b$. We wish to compare the relative size of these two perturbations:

$$\frac{\|\delta x\|_2}{\|x\|_2} \quad \text{vs} \quad \frac{\|\delta b\|_2}{\|b\|_2}.$$

We note that the ratio of these two quantities is bounded by

$$\frac{\|\delta x\|_2/\|x\|_2}{\|\delta b\|_2/\|b\|_2} = \frac{\|\delta x\|_2}{\|\delta b\|_2}\frac{\|b\|_2}{\|x\|_2} = \frac{\|A^{-1}\delta b\|_2}{\|\delta b\|_2}\frac{\|Ax\|_2}{\|x\|_2} \leq \|A^{-1}\|_2\|A\|_2. \tag{7.1}$$

Hence, the ratio of the relative perturbation in the solution to the relative perturbation in the right hand side is bounded by $\|A^{-1}\|_2\|A\|_2$. This number is called the *condition number* and is denoted by $\kappa(A)$:

$$\kappa(A) = \|A^{-1}\|_2\|A\|_2.$$

Rearranging the expressions (7.1), we note that

$$\frac{\|\delta x\|_2}{\|x\|_2} \leq \kappa(A) \frac{\|\delta b\|_2}{\|b\|_2}.$$

In words, the condition number bounds the relative perturbation in the solution by the relative perturbation in the right hand side.

Before we investigate the importance of condition number in numerical methods, we now consider a few other characterizations of the condition number. We recall $\|A^{-1}\|_2 = \sigma_{\min}(A^{-1})^{-1}$ and $\|A\|_2 = \sigma_{\max}(A)$, where $\sigma_{\min}$ and $\sigma_{\max}$ denote the minimum and maximum singular value of the matrix, respectively. Hence, the condition number can also be expressed as

$$\kappa(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)};$$

we observe that since $\sigma_{\max}(A) \geq \sigma_{\min}(A)$ by definition, $\kappa(A) \geq 1$. For a SPD matrix, since the eigenvalues and the singular values are the same, we may express the condition number as

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

For an orthogonal matrix Q, the condition number is unity ($\kappa(Q) = 1$) because all of its singular values are 1.

7.5 Ill-conditioned systems

The condition number plays an important role in the numerical solution of linear systems. To illustrate its importance, let us consider a 2×2 system

$$\begin{pmatrix} 1+\delta & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

for some small nonzero number δ. (For a concrete illustration, consider for instance $\delta = 10^{-15}$.) Because the matrix A is non-singular, the problem is formally well-posed and it has a unique solution:

$$x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Note that that both the solution and the right hand side are of $\mathcal{O}(1)$.

Now let us consider a very slightly perturbed problem:

$$\begin{pmatrix} 1+\delta & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1+\delta \\ 1 \end{pmatrix} \equiv \tilde{b};$$

the right hand side has been perturbed by $\mathcal{O}(\delta)$. The solution to this problem is

$$\tilde{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

We note that the even though the perturbation $\|\tilde{b} - b\|_2 = \|(\delta, 0)^T\|_2$ is of order $\mathcal{O}(\delta)$ — very small — the solution perturbation $\|\tilde{x} - x\|_2 = \|(1, -1)\|_2$ is of order $\mathcal{O}(1)$. Hence, even a very small error

in the data b (vs $\tilde{b}$), can result in a completely different solution x (vs $\tilde{x}$). Problems that exhibit this sensitive behavior to data is said to be *ill-conditioned*.

The ill-conditioning of the system is reflected in the condition number. We can estimate the condition number of the system by computing its eigenvalues:

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)} = \frac{2 + \delta + \sqrt{\delta^2 + 4}}{2 + \delta - \sqrt{\delta^2 + 4}} \approx \frac{4}{\delta} \quad \text{for } \delta \ll 1.$$

A linear system with a large condition number is said to be ill-conditioned.

The condition number plays a particularly important role when the computation is carried out using floating-point arithmetic. In the above example of the ill-conditioned 2×2 system, the two right hand sides

$$b = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{and} \quad \tilde{b} = \begin{pmatrix} 1 + \delta \\ 1 \end{pmatrix}$$

become indistinguishable in floating point arithmetic when $\delta < \epsilon_{\text{machine}}$. However, we recall that a small perturbation of δ is sufficient to yield two completely different solutions:

$$x = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \text{and} \quad \tilde{x} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Hence, if the system is ill-conditioned, then even the very small error in floating point representation can be amplified to yield completely different solutions. Hence, regardless of the algorithm used to solve $Ax = b$, we cannot expect to obtain an accurate solution if the system x is ill-conditioned.

7.6 Stability of Gaussian elimination (without pivoting)

We have observed that, for ill-conditioned systems, we cannot expect to obtain an accurate solution regardless of the algorithm used. However, it turns out that the pure form of Gaussian elimination introduced in the previous lecture is unstable even if the system is well conditioned. In fact, the algorithm may simply not work in some cases. Consider a matrix

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 1 \end{pmatrix}.$$

This matrix is obviously non-singular *and* is well-conditioned; the condition number is approximately 2.6. However, we cannot apply our Gaussian elimination algorithm because the $(1, 1)$ entry is 0.

A more subtle problem arises for a matrix

$$B = \begin{pmatrix} 10^{-17} & 1 \\ -1 & 1 \end{pmatrix}.$$

This matrix is again non-singular *and* is well-conditioned; the condition number is again approximately 2.6. This time, we can apply our Gaussian elimination algorithm because the $(1, 1)$ entry is small but finite. The resulting LU decomposition is

$$B = \begin{pmatrix} 10^{-17} & 1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -10^{17} & 1 \end{pmatrix} \begin{pmatrix} 10^{-17} & 1 \\ 0 & 1 + 10^{17} \end{pmatrix}.$$

However, recall that the double-precision arithmetic provides $\epsilon_{\text{machine}} \approx 10^{-16}$. Hence, the number $1 + 10^{17}$ is represented in a computer by its closest floating point number, 10^{17}. Thus, Gaussian elimination performed on a computer yields a decomposition

$$\begin{pmatrix} 1 & 0 \\ -10^{17} & 1 \end{pmatrix} \begin{pmatrix} 10^{-17} & 1 \\ 0 & 10^{17} \end{pmatrix}.$$

Unfortunately, the product of the L and U yields

$$\begin{pmatrix} 1 & 0 \\ -10^{17} & 1 \end{pmatrix} \begin{pmatrix} 10^{-17} & 1 \\ 0 & 10^{17} \end{pmatrix} = \begin{pmatrix} 10^{-17} & 1 \\ -1 & 0 \end{pmatrix} \neq B.$$

Even though the numbers are represented to $\mathcal{O}(10^{-16})$ relative accuracy, the pure form of LU factorization has committed an $\mathcal{O}(1)$ error even if the system is well-conditioned. (Recall that the condition number is $\approx 2.6 = \mathcal{O}(1)$ for this matrix.) This is clearly undesirable.

7.7 Partial pivoting: motivational example

We can overcome the stability problem by introducing *partial pivoting*. The idea is very simple: in the k-th step of Gaussian elimination, we swap rows such that the pivot — the (k, k) entry — is the biggest entry (in magnitude).

For in stance, in the case of

$$B = \begin{pmatrix} 10^{-17} & 1 \\ -1 & 1 \end{pmatrix},$$

we first compare the entries in the first column: 10^{-17} and -1. We note that the $(2, 1)$ entry is bigger than the $(1, 1)$ entry in magnitude. We hence introduce a *permutation matrix* that swaps the first and second rows,

$$P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

and apply the matrix to B to obtain

$$PB = \begin{pmatrix} -1 & 1 \\ 10^{-17} & 1 \end{pmatrix}.$$

We then eliminate the $(2, 1)$ entry of the new matrix by subtracting -10^{-17} times the first row to the second row:

$$\begin{pmatrix} -1 & 1 \\ 10^{-17} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -10^{-17} & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 0 & 1 + 10^{-17} \end{pmatrix}.$$

As before, $1 + 10^{-17}$ is represented in a computer as 1. But, this time, we note that

$$\begin{pmatrix} 1 & 0 \\ -10^{-17} & 1 \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 10^{-17} & 1 - 10^{-17} \end{pmatrix} = PB + \begin{pmatrix} 0 & 0 \\ 0 & 10^{-17} \end{pmatrix}.$$

We have just introduced not the LU decomposition of the original matrix B but rather a LU decomposition the permuted matrix $PB = LU$. Unlike the LU decomposition (without pivoting)of B, which produces an inaccurate decomposition (i.e. $B \neq LU$ with $\mathcal{O}(1)$ error), the LU decomposition of PB is very accurate (i.e. $PB = LU$ up to machine precision). Hence, even though two matrices B and PB are closely related, one produces an unstable LU decomposition (without pivoting) while the other produces a stable decomposition.

7.8 LU factorization (with pivoting)

We now present the LU factorization (with pivoting) for the 3×3 matrix we have considered above,

$$A = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix}.$$

Step 1. Compare the magnitude of the entries in the first column. Since the third entry is the largest, swap the first and third rows:

$$\underbrace{\begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix}}_{P_1} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 4 & 3 & 3 \\ 2 & 1 & 1 \end{pmatrix}}_{P_1 A}.$$

Step 1'. Eliminate the subdiagonal entries in the first column (as in the non-pivoting LU):

$$\underbrace{\begin{pmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{1}{4} & & 1 \end{pmatrix}}_{M_1} \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 4 & 3 & 3 \\ 2 & 1 & 1 \end{pmatrix}}_{P_1 A} = \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 0 & -\frac{1}{2} & -2 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \end{pmatrix}}_{M_1 P_1 A}.$$

Step 2. Compare the magnitude of the entries in the second column. Since the third entry is the largest, swap the second and third rows:

$$\underbrace{\begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & \end{pmatrix}}_{P_2} \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 0 & -\frac{1}{2} & -2 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \end{pmatrix}}_{M_1 P_1 A} = \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \\ 0 & -\frac{1}{2} & -2 \end{pmatrix}}_{P_2 M_1 P_1 A}.$$

Step 2'. Eliminate the subdiagonal entries in the second column (as in the non-pivoting LU):

$$\underbrace{\begin{pmatrix} 1 & & \\ & 1 & \\ & -\frac{2}{3} & 1 \end{pmatrix}}_{M_2} \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \\ 0 & -\frac{1}{2} & -2 \end{pmatrix}}_{P_2 M_1 P_1 A} = \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ 0 & -\frac{3}{4} & -\frac{3}{2} \\ 0 & 0 & -1 \end{pmatrix}}_{M_2 P_2 M_1 P_1 A} \equiv U.$$

We observe that we have now created a decomposition $M_2 P_2 M_1 P_1 A = U$. We now re-write this decomposition as

$$(M_2)(P_2 M_1 P_2^{-1})(P_2 P_1) A = U.$$

We then make the third crucial observation (in addition to the two we made in the context of LU without pivoting):

$$P_2 M_1 P_2^{-1} = \begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & \end{pmatrix} \begin{pmatrix} 1 & & \\ -\frac{1}{2} & 1 & \\ -\frac{1}{4} & & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & \end{pmatrix} = \begin{pmatrix} 1 & & \\ -\frac{1}{4} & 1 & \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}:$$

the permuted matrix $P_2 M_1 P_2^{-1}$ retains the lower triangular structure, and in addition we obtain $P_2 M_1 P_2^{-1}$ by simply swapping the subdiagonal entries of M_1. In particular, because M_2 and $P_2 M_1 P_2^{-1}$ are both lower triangular, we can appeal to the two crucial observations we made earlier in the context of LU without pivoting to obtain

$$(P_2 M_1 P_2^{-1})^{-1} M_2^{-1} = \begin{pmatrix} 1 & & \\ -\frac{1}{4} & 1 & \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} 1 & & \\ & 1 & \\ & -\frac{2}{3} & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & & \\ \frac{1}{4} & 1 & \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & \frac{2}{3} & 1 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & & \\ \frac{1}{4} & 1 & \\ \frac{1}{2} & \frac{2}{3} & 1 \end{pmatrix} \equiv L,$$

which is a lower triangular matrix. We also note that

$$P_2 P_1 = \begin{pmatrix} 1 & & \\ & & 1 \\ & 1 & \end{pmatrix} \begin{pmatrix} & & 1 \\ & 1 & \\ 1 & & \end{pmatrix} = \begin{pmatrix} & & 1 \\ 1 & & \\ & 1 & \end{pmatrix} \equiv P,$$

which is a permutation matrix. It thus follows

$$\underbrace{\begin{pmatrix} & & 1 \\ 1 & & \\ & 1 & \end{pmatrix}}_{P} \underbrace{\begin{pmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 10 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & & \\ \frac{1}{4} & 1 & \\ \frac{1}{2} & \frac{2}{3} & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 8 & 7 & 10 \\ & -\frac{3}{4} & -\frac{3}{2} \\ & & -1 \end{pmatrix}}_{U}.$$

This is the LU factorization of A with partial pivoting. Because the partial pivoting is used universally in practice, this method is simply called LU facotrization. With partial pivoting, we note that the L matrix has subdiagonal entries that are all less than or equal to zero.

The Gaussian elimination with partial pivoting for a general $n \times n$ matrix may be expressed as follows:

Algorithm 5: LU factorization with partial pivoting

```
1  U = A, L = I, P = I
2  for i = 1, ..., n − 1 do
3      Find q ≥ i that maximizes |u_qk|
4      Exchange u_{i,i:n} ↔ u_{q,i:n}, l_{i,1:i−1} ↔ l_{q,1:i−1}, p_{i,:} ↔ p_{q,:}
5      for j = i + 1, ..., n do
6          l_ji = u_ji / u_ii
7          for k = i, ..., n do
8              u_jk = u_jk − l_ji u_ik
9          end
10     end
11 end
```

This is the algorithm used to solve linear systems by essentially any numerical solution package. One of the most popular numerical package that implements the algorithm is LAPACK, which in fact is also used as the backbone to solve linear systems in MATLAB. (The actual implementation often do not explicitly swap rows to perform partial pivoting, but achieve the same effect through careful bookkeeping.)

Solution of linear system. Given an LU factorization such that $PA = LU$, we can find the solution to $Ax = b$ as follows. We first permute the entries of b to obtain Pb. We then solve

$$Lz = Pb$$

using forward substitution. We finally solve

$$Ux = z$$

using backward substitution.

7.9 SPD systems: Cholesky factorization

If the matrix A is symmetric positive definite (SPD), we may take advantage of the symmetry (and also the positive definiteness) to construct a factorization of the form

$$A = R^T R,$$

where R is upper triangular. The factorization is called the *Cholesky factorization*. Because the algorithm takes advantage of the symmetry, the factorization can be computed in $\frac{1}{3}n^3$ operations (as opposed to $\frac{2}{3}n^3$ for LU factorization). In addition, Cholesky factorization does not require pivoting to achieve stability.

7.10 Summary

We summarize the key points of this lecture:

1. Real numbers are approximated using floating-point numbers in (digital) computers. The IEEE double-precision format provides a relative precision of $\epsilon_{\text{machine}} \approx 10^{-16}$.

2. The condition number of a matrix A characterizes the sensitivity of the solution x of $Ax = b$ to the perturbations in the right hand side b.

3. If a matrix A is ill-conditioned, the solution x may be inaccurate independent of the algorithm used to solve $Ax = b$.

4. Gaussian elimination without partial pivoting can be unstable. Even if the condition number of the matrix is $\mathcal{O}(1)$, the algorithm can stall or produce a factorization with an error of $\mathcal{O}(1)$.

5. Pivoting eliminates stability issues associated with Gaussian elimination. The resulting factorization is of the form $PA = LU$, where P is a permutation matrix.

6. If the matrix A is SPD, we can consider Cholesky factorization $A = R^T R$, where R is an upper triangular matrix. The operation count for Cholesky factorization is $\approx \frac{1}{3}n^3$, approximately half of the LU factorization.

7.11 Appendix. A review of vector and matrix norms

A *norm* $\|\cdot\|$ on a vector space $\mathbb{R}^n$ have the following properties: for all $\alpha \in \mathbb{R}$ and $w, v \in \mathbb{R}^n$,

1. $\|\alpha w\| = |\alpha| \|w\|$
2. $\|w + v\| \leq \|w\| + \|v\|$ (triangle inequality)
3. $\|w\| \geq 0$, and $\|w\| = 0$ if and only if $w = 0$.

While any function that satisfies the above three properties is a norm on $\mathbb{R}^n$, a family of norms that is particularly useful for us is the *p-norm*. Given a vector $v \in \mathbb{R}^n$, the p-norm of the vector is

$$\|v\|_p \equiv \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}$$

for $p \geq 1$. Some of the common p-norms are

$$\begin{aligned} \|v\|_1 &\equiv \sum_{i=1}^{n} |v_i| \quad \text{(1-norm)} \\ \|v\|_2 &\equiv \left(\sum_{i=1}^{n} |v_i|^2 \right)^{1/2} \quad \text{(2-norm)} \\ \|v\|_\infty &\equiv \max_i |v_i| \quad (\infty\text{-norm}). \end{aligned}$$

We can readily verify p-norms satisfy the required properties of norms.

A matrix norm is a vector norm on $\mathbb{R}^{m \times n}$. One family of norms that is particularly relevant for us is *induced norm*. Given $A \in \mathbb{R}^{m \times n}$ and a vector norm $\|\cdot\|$, the induced norm associated with $\|\cdot\|$ is

$$\|A\| \equiv \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|}{\|x\|}.$$

The induced norm associated with the p-norm is

$$\|A\|_p \equiv \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|_p}{\|x\|_p}, \qquad p \geq 1.$$

Lecture 8

Least-squares problems

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

Disclaimer: significant fraction of the section on QR factorization was originally developed for 2.086 taught at MIT and is also found in *Numerical methods for mechanical engineers* by Masayuki Yano, James Penn, and Anthony Patera.

8.1 Least-squares

We have so far considered non-singular $n \times n$ linear systems. These systems have unique solution because the number of unknowns (i.e. the number of columns) is equal to the number of independent equations (i.e. the number of rows). In this lecture we consider an overdetermined system, where the number of (independent) equations is greater than the number of unknowns.

A concrete example of such a system is

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 0 \end{pmatrix}.$$

Here, in order to satisfy the first equation, $1 \cdot x_1 + 0 \cdot x_2 = 2$, we need $x_1 = 2$. Similarly, in order to satisfy the second equation, $0 \cdot x_1 + 1 \cdot x_2 = 4$, we need $x_2 = 4$. However, with these choices of x_1 and x_2, the third equation is not satisfied: $1 \cdot x_1 + 1 \cdot x_2 \neq 0$. Hence the system is *overdetermined*, and there is no x that satisfies the three equations.

More generally, a linear system is overdetermined if the number of independent equations is greater than the number of unknowns. Specifically, a linear system $Ax = b$ associated with the $m \times n$ matrix A is overdetermined if the dimension of the image of A is greater than n; since $\dim(\text{img}(A)) \leq m$, we must have $m > n$ for $\dim(\text{img}(A)) > n$. In words, we are considering "tall-and-skinny" systems

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix},$$

where $m > n$.

For an overdetermined system, we forgo finding x that satisfies $Ax = b$ — since it does not exist — and instead look for an x that "satisfies the equation as well as possible" in some sense. Specifically, we first introduce the *residual*

$$r \equiv Ax - b,$$

which is an m vector. The residual measures the "misfit" between the left-hand side Ax and the right-hand side b. We then seek x which minimizes the residual in some norm. We may consider different norms — 1-norm, 2-norm, ∞-norm, etc — but in this lecture we choose the *2-norm*. Hence our goal is to find n-vector x such that

$$\|r\|_2 \equiv \|Ax - b\|_2$$

is minimized. We recall that the two norm of a vector is given by

$$\|r\|_2 \equiv \left(\sum_{i=1}^{m} r_i^2 \right)^{1/2}$$

Assuming the columns of the matrix A are linearly independent, the solution to the least-squares problem is unique.

8.2 Normal equation

The solution to the least-squares problem satisfies the *normal equation*,

$$A^T A x = A^T b.$$

This $n \times n$ linear system is non-singular and has a unique solution if the columns of A are linearly independent.

In order to derive the normal equation, we first note that the square of the 2-norm of the residual can be expressed as

$$\begin{aligned} r^2(x) \equiv \|Ax - b\|_2^2 = (Ax - b)^T (Ax - b) &= x^T A^T A x - b^T A x - x^T A^T b - b^T b \\ &= x^T A^T A x - 2 x^T A^T b - b^T b. \end{aligned}$$

If the columns of A are linearly independent, then $A^T A$ is symmetric positive definite, and the function $r^2(x)$ is convex. Hence, a unique minimum is obtained at x for which $\frac{\partial r^2}{\partial x} = 0$. An explicit expression for the condition is

$$\frac{\partial r^2}{\partial x} = 2A^T A x - 2A^T b = 0,$$

or equivalently $A^T A x = A^T b$, which is the normal equation.

Let us count the number of operations associated with the formation and solution of the normal equation. The computation of each entry of $A^T A$ requires a dot product of two m-vectors, which requires $2m$ operations. Using the symmetry of $A^T A$ to compute the $n(n+1)/2$ entries of $A^T A$ hence requires $\approx mn^2$ operations. The formulation of $A^T b$ on the hand requires mn operations. Hence the leading cost for forming the normal equation is mn^2. In addition, the solution of the

normal equation by Cholesky factorization (again taking advantage of the symmetry) requires $\approx n^3/3$ operations. Hence the total cost of solving the least-squares problem by normal equation is $mn^2 + n^3/3$ operations.

Unforatunatley, solving the normal equation by the normal equation is not most stable. In particular, the condition number of A^TA is square of the condition number of A: $\kappa(A^TA) = \kappa(A)^2$. We now introduce a more suitable method.

8.3 Reduced QR factorization

A standard approach to numerically solve least-squares problems is by *(reduced) QR factorization.* QR factorization is a decomposition of a matrix $A \in \mathbb{R}^{m\times n}$ into $Q \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$ such that

$$A = QR,$$

where the columns of the matrix $Q \in \mathbb{R}^{m\times n}$ are orthonormal and the matrix $R \in \mathbb{R}^{n\times n}$ is upper triangular:

$$\underbrace{\begin{pmatrix} | & & | \\ a_1 & \cdots & a_n \\ | & & | \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} | & & | \\ q_1 & \cdots & q_n \\ | & & | \end{pmatrix}}_{Q} \underbrace{\begin{pmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \end{pmatrix}}_{R}.$$

Note that a_j and q_j denote the j-th column of A and Q, respectively. The fact columns of Q are orthonormal implies that

$$Q^TQ = I \qquad (\text{of size } n \times n).$$

However, because $Q \in \mathbb{R}^{m\times n}$ for $m > n$, $QQ^T \neq I$ unless $m = n$. We also note that the fact R is an upper triangular matrix implies that the first j columns of Q must span the same space as the first j columns of A.

Using the QR factorization, we can greatly simplify the normal equation $A^TAx = A^Tb$. Substitution of QR factorization to the normal equation yields

$$A^TAx = A^Tb \quad \Rightarrow \quad R^TQ^TQRx = R^TQ^Tb \quad \Rightarrow \quad R^TRx = RQ^T \quad \Rightarrow \quad Rx = Q^Tb.$$

Here the second step follows from the fact that $Q^TQ = I$, and the last step follows from the fact that the upper triangular matrix R is invertible as long as the its diagonal entries are all nonzero, which will be the case for A with linearly independent columns.

8.4 Gram-Schmidt procedure

There are at least three classical methods to systematically find a QR factorization: the Gram-Schmidt procedure, the Householder transformation, and Givens rotation. Here we briefly discuss the Gram-Schmidt procedure. The idea behind the Gram-Schmidt procedure is to successively turn the columns of A into orthonormal vectors. Recall that we want $q_i^Tq_=\delta_{ij}$ (Kronecker-delta), and, in order to get an upper triangular R, we also require that

$$\text{span}\{a_1, \ldots, a_j\} = \text{span}\{q_1, \ldots, q_j\}, \quad \forall j = 1, \ldots, n.$$

The Gram-Schmidt procedure starts with a set which consists of a single vector, a_1. We construct an orthonormal set consisting of single vector q_1 that spans the same space as $\{a_1\}$. Trivially, we can take

$$q_1 = \frac{1}{\|a_1\|} a_1 \ .$$

Or, we can express a_1 as

$$a_1 = q_1 \|a_1\| \ ,$$

which is the product of a unit vector and an amplitude.

Now we consider a set which consists of the first two columns of A, $\{a_1, a_2\}$. Our objective is to construct an orthonormal set $\{q_1, q_2\}$ that spans the same space as $\{a_1, a_2\}$. In particular, we will keep the q_1 we have constructed in the first step unchanged, and choose q_2 such that (i) it is orthogonal to q_1, and (ii) $\{q_1, q_2\}$ spans the same space as $\{a_1, a_2\}$. To do this, we start with a_2 and first remove the component in the direction of q_1, i.e.

$$\tilde{q}_2 = a_2 - (q_1^{\mathrm{T}} a_2) q_1 \ .$$

Here, we recall the fact that the inner product $q_1^{\mathrm{T}} a_2$ is the component of a_2 in the direction of q_1. We readily confirm that $\tilde{q}_2$ is orthogonal to q_1, i.e.

$$q_1^{\mathrm{T}} \tilde{q}_2 = q_1^{\mathrm{T}} (a_2 - (q_1^{\mathrm{T}} a_2) q_1) = q_1^{\mathrm{T}} a_2 - (q_1^{\mathrm{T}} a_2) q_1^{\mathrm{T}} q_1 = q_1^{\mathrm{T}} a_2 - (q_1^{\mathrm{T}} a_2) \cdot 1 = 0 \ .$$

Finally, we normalize $\tilde{q}_2$ to yield the unit length vector

$$q_2 = \tilde{q}_2 / \|\tilde{q}_2\| \ .$$

With some rearrangement, we see that a_2 can be expressed as

$$a_2 = (q_1^{\mathrm{T}} a_2) q_1 + \tilde{q}_2 = (q_1^{\mathrm{T}} a_2) q_1 + \|\tilde{q}_2\| q_2 \ .$$

Using a matrix-vector product, we can express this as

$$a_2 = \begin{pmatrix} q_1 & q_2 \end{pmatrix} \begin{pmatrix} q_1^{\mathrm{T}} a_2 \\ \|\tilde{q}_2\| \end{pmatrix} .$$

Combining with the expression for a_1, we have

$$\begin{pmatrix} a_1 & a_2 \end{pmatrix} = \begin{pmatrix} q_1 & q_2 \end{pmatrix} \begin{pmatrix} \|a_1\| & q_1^{\mathrm{T}} a_2 \\ & \|\tilde{q}_2\| \end{pmatrix} .$$

In two steps, we have factorized the first two columns of A into an $m \times 2$ orthogonal matrix (q_1, q_2) and a 2×2 upper triangular matrix. The Gram-Schmidt procedure consists of repeating the procedure n times; let us show one more step for clarity.

In the third step, we consider a set which consists of the first three columns of A, $\{a_1, a_2, a_3\}$. Our objective it to construct an orthonormal set $\{q_1, q_2, q_3\}$. Following the same recipe as the second step, we keep q_1 and q_2 unchanged, and choose q_3 such that (i) it is orthogonal to q_1 and q_2, and (ii) $\{q_1, q_2, q_3\}$ spans the same space as $\{a_1, a_2, a_3\}$. This time, we start from a_3, and remove the components of a_3 in the direction of q_1 and q_2, i.e.

$$\tilde{q}_3 = a_3 - (q_1^{\mathrm{T}} a_3) q_1 - (q_2^{\mathrm{T}} a_3) q_2 \ .$$

Again, we recall that $q_1^{\mathrm{T}} a_3$ and $q_2^{\mathrm{T}} a_3$ are the components of a_3 in the direction of q_1 and q_2, respectively. We can again confirm that $\tilde{q}_3$ is orthogonal to q_1

$$q_1^{\mathrm{T}} \tilde{q}_3 = q_1^{\mathrm{T}}(a_3 - (q_1^{\mathrm{T}} a_3)q_1 - (q_2^{\mathrm{T}} a_3)q_2) = q_1^{\mathrm{T}} a_3 - (q_1^{\mathrm{T}} a_3)\cancelto{1}{q_1^{\mathrm{T}} q_1} - (q_2^{\mathrm{T}} a_3)\cancelto{0}{q_1^{\mathrm{T}} q_2} = 0$$

and to q_2

$$q_2^{\mathrm{T}} \tilde{q}_3 = q_2^{\mathrm{T}}(a_3 - (q_1^{\mathrm{T}} a_3)q_1 - (q_2^{\mathrm{T}} a_3)q_2) = q_2^{\mathrm{T}} a_3 - (q_1^{\mathrm{T}} a_3)\cancelto{0}{q_2^{\mathrm{T}} q_1} - (q_2^{\mathrm{T}} a_3)\cancelto{1}{q_2^{\mathrm{T}} q_2} = 0 \ .$$

We can express a_3 as

$$a_3 = (q_1^{\mathrm{T}} a_3)q_1 + (q_2^{\mathrm{T}} a_3)q_2 + \|\tilde{q}_3\| q_3 \ .$$

Or, putting the first three columns together

$$\begin{pmatrix} a_1 & a_2 & a_3 \end{pmatrix} = \begin{pmatrix} q_1 & q_2 & q_3 \end{pmatrix} \begin{pmatrix} \|a_1\| & q_1^{\mathrm{T}} a_2 & q_1^{\mathrm{T}} a_3 \\ & \|\tilde{q}_2\| & q_2^{\mathrm{T}} a_3 \\ & & \|\tilde{q}_3\| \end{pmatrix}.$$

We can see that repeating the procedure n times would result in the complete orthogonalization of the columns of A.

8.5 "Pure" and modified Gram-Schmidt procedures

The "pure" form of the Gram-Schmidt procedure described above is given in an algorithmic form as follows:

Algorithm 6: "pure" Gram-Schmidt procedure

```
for j = 1, ..., n do
    q̃_j = a_j
    for k = 1, ..., j − 1 do
        r_kj = q_k^T a_j
        q̃_j = q̃_j − r_kj q_k
    end
    r_jj = ‖q̃_j‖_2
    q_j = (1/r_jj) q̃_j
end
```

It turns out that this pure form of the Gram-Schmidt procedure is numerically unstable; i.e., it is susceptible to round off errors. Fortunately, the algorithm can be made stable with a minimal change. Specifically, we replace the inner loop for the "pure" procedure

$$\tilde{q}_j \leftarrow \tilde{q}_j - (q_k^T a_j) q_k,$$

with the inner loop for the "modified" procedure

$$\tilde{q}_j \leftarrow \tilde{q}_j - (q_k^T \tilde{q}_k) q_k.$$

A geometric interpretation for the "pure" procedure is that, in the k-th step, we compute the component of the *original vector* a_j that lies in the direction of q_k and subtract it from $\tilde{q}_j$; this is equivalent to *simultaneously* subtracting from a_j the components of a_j that lies in the directions of $q_1, \ldots, q_{j-1}$. A geometric interpretation for the modified procedure is that, in the k-th step, we compute the component of the *current vector* $\tilde{q}_j$ that lies in the direction of q_k and subtract it from itself; in other words, we *successively* subtract, in $j-1$ steps, from $\tilde{q}_j$ the component of $\tilde{q}_j$ that lies in the direction of q_k, $k = 1, \ldots, j-1$. When the computation is carried out in exact arithmetic, the pure and modified Gram-Schmidt yield identical results; however, in floating-point arithmetic, the modified procedure yields a smaller error than the pure procedure. The resulting algorithm, the *modified Gram-Schmidt procedure*, is the following:

Algorithm 7: modified Gram-Schmidt procedure

```
for j = 1, ..., n do
    q̃_j = a_j
    for k = 1, ..., j − 1 do
        r_kj = q_k^T q̃_j
        q̃_j = q̃_j − r_kj q_k
    end
    r_jj = ‖q̃_j‖_2
    q_j = (1/r_jj) q̃_j
end
```

As the modified algorithm results from a minimal change to the "pure" algorithm, we should always use the modified Gram-Schmidt procedure for the better numerical stability.

8.6 Computational cost of the Gram-Schmidt procedure

Let us count the number of operations of the Gram-Schmidt procedure. At j-th step, there are $j-1$ components to be removed, each requiring of $4m$ operations. Thus, the total operation count of the Gram-Schmidt procedure is

$$C^{\text{Gram-Schmidt}} \approx \sum_{j=1}^{n} (j-1)4m \approx 2mn^2 \ .$$

To find the least-squares solution, we then solve $Rx = Q^T b$; the formation of the right hand side $Q^T b$ requires $2mn$ operations, and the solution of the upper triangular system requires n^2 operations using backward substitution. Thus, the total cost for solving a least-squares problem by QR factorization is $2mn^2 + 2mn + n^2 \approx 2mn^2$. Note that the method based on Gram-Schmidt is approximately twice as expensive as the method based on normal equation for $m \gg n$. However, the superior numerical stability often warrants the additional cost.

8.7 Full QR factorization

We introduced in Section 8.3 a *reduced* QR factorization, which yields $A = QR$ where $A \in \mathbb{R}^{m\times n}$, $Q \in \mathbb{R}^{m\times n}$ consists of a set of orthonormal columns, and $R \in \mathbb{R}^{n\times n}$ is upper triangular. Another variant of QR factorization is a *full* QR factorization. A full QR factorization of A is given by

$$A = \tilde{Q}\tilde{R},$$

where $\tilde{Q} \in \mathbb{R}^{m\times m}$ is an orthogonal matrix and $\tilde{R} \in \mathbb{R}^{m\times n}$ is still upper triangular:

$$A = \underbrace{\begin{pmatrix} q_{11} & \cdots & q_{1n} & q_{1,n+1} & \cdots & q_{1,m} \\ \vdots & & \vdots & \vdots & & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ \vdots & & \vdots & \vdots & & \vdots \\ q_{m1} & \cdots & q_{mn} & q_{m,n+1} & \cdots & q_{m,m} \end{pmatrix}}_{\tilde{Q}} \underbrace{\begin{pmatrix} r_{11} & \cdots & r_{1n} \\ 0 & \ddots & \vdots \\ \vdots & \ddots & r_{nn} \\ \vdots & & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}}_{\tilde{R}}.$$

We note that the $\tilde{Q}$ matrix associated with the full QR factorization is square ($m \times m$) and orthogonal; this is unlike the Q matrix associated with the reduced QR factorization which is $m \times n$. The $\tilde{R}$ matrix is rectangular ($m \times n$) and its subdiagonal entries are all zero; this again is unlike the R matrix associated with the reduced QR factorization which is $n \times n$. Note that we can easily identify a reduced QR factorization of A from a full QR factorization of A: we use the first n columns of $\tilde{Q}$ forms Q, and the first n rows of $\tilde{R}$ forms R to form the reduced matrices.

The full QR factorization may be thought of as a "completion" of the reduced QR factorization such that $\text{Img}(\tilde{Q}) = \mathbb{R}^m$. In practice, the full QR factorization is computed using the *Householder reflections* or *Givens rotations*. The coverage of these two algorithms is beyond the scope of this lecture.

8.8 Summary

We summarize the key points of this lecture:

1. Given an overdetermined system, the least-squares solution is the solution that minimizes the 2-norm of the residual.

2. Given the columns of the matrix A is linearly independent, the least-squares solution exists and is unique.

3. Given an $m \times n$ matrix, the normal equation is a $n \times n$ system whose solution is the least-squares solution. The method allows simple by-hand calculation of least-squares solution; however, the method provides limited numerical stability.

4. Given a matrix $A \in \mathbb{R}^{m\times n}$, a (reduced) QR factorization is a decomposition of the matrix into an orthogonal matrix $Q \in \mathbb{R}^{m\times n}$ and an upper triangular matrix $R \in \mathbb{R}^{n\times n}$.

5. Given $A = QR$, the least squares solution is the solution to $Rx = Q^T b$.

6. Gram-Schmidt procedure is a systematic approach to find a QR factorization of a matrix $A \in \mathbb{R}^{m \times n}$ in approximately $2mn^2$ operations.

7. The "pure" Gram-Schmidt procedure can be numerically unstable; the modified Gram-Schmidt procedure resolves the stability issue.

8. There exist two variants of QR factorization: reduced and full. The factors Q and R of a reduced QR factorization are submatrices of the factors $\tilde{Q}$ and $\tilde{R}$ of a full QR factorization.

Lecture 9

Regression: statistical inference

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

Disclaimer: significant fraction of this lecture was originally developed for 2.086 taught at MIT and is also found in *Numerical methods for mechanical engineers* by Masayuki Yano, James Penn, and Anthony Patera.

9.1 Motivation

Regression is often used in two different contexts.

1. **Prediction.** Suppose we are given a *noisy dataset* consisting of m data points (x_i, Y_i), $i = 1, \ldots, m$. We then wish to predict the value of y for an arbitrary value of x.
2. **Inference.** Suppose we are given a parametrized model, but we do not know the value of the parameter. Given a *noisy dataset* consisting of m data points (x_i, Y_i), $i = 1, \ldots, m$, we wish to predict the value of the parameter that best describe the behavior.

Both the prediction and inference problems are ubiquitous in engineering and science. For instance, climate modeling is a challenging inference and prediction problem that incorporates sophisticated climate models and data (e.g. remote sensing). In this lecture we introduce fundamentals of regression.

9.2 Response model

We first propose a model for the relationship between the input x and output Y; the particular model we will consider is of form

$$Y = Y_{\text{model}}(x; \beta) + \epsilon(x),$$

where

1. x is the independent variable,
2. Y is the measured quantity, which in general is noisy.

3. Y_{model} is the predictive model with no noise. In linear regression, Y_{model} is by definition a linear function of the model parameter β.

4. ϵ is a noise, which is a random variable.

To simplify the presentation, we will first consider a simple two degrees-of-freedom model for Y_{model}:

$$Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x,$$

where β_0 and β_1 are the components of the model parameter $\beta \in \mathbb{R}^2$. We will later generalize this assumption (still in the context of linear regression).

We now introduce the key assumption: our model is *unbiased*. That is, in the absence of noise ($\epsilon = 0$), our underlying input-output relationship can be perfectly described by

$$y(x) = Y_{\text{model}}(x;\beta^{\text{true}})$$

for some true parameter β^{true}. In other words, our model includes the true functional dependency (but may include more generality than is actually needed).

We also introduce assumptions on the characteristics of the noise. These assumptions allow us to make quantitative (statistical) claims about the quality of our regression.

1. **Normality.** We assume the noise is normally distributed with zero mean, i.e., $\epsilon(x) \sim \mathcal{N}(0, \sigma^2(x))$.

2. **Homoscedasticity.** We assume that the distribution of the noise ϵ, as characterized by σ^2, does not depend on x.

3. **Independence.** We assume that $\epsilon(x_1)$ and $\epsilon(x_2)$, $x_1 \neq x_2$, are independent and hence uncorrelated.

These assumptions imply that $\epsilon(x) = \epsilon = \mathcal{N}(0, \sigma^2)$, where σ^2 is the single parameter that characterizes the noise for *all* instances of x. Figure 9.1 shows visually the regression process; note that the Y_{model} is an affine function of x and the noise is a normal distribution with a zero mean and a fixed standard deviation.

We note that because

$$Y(x) = Y_{\text{model}}(x;\beta) + \epsilon = \beta_0 + \beta_1 x + \epsilon$$

and $\epsilon \sim \mathcal{N}(0, \sigma^2)$, the deterministic model $Y_{\text{model}}(x;\beta)$ simply shifts the mean of the normal distribution. Thus, the measurement is a random variable with the distribution

$$Y(x) \sim \mathcal{N}(Y_{\text{model}}(x;\beta), \sigma^2) = \mathcal{N}(\beta_0 + \beta_1 x, \sigma^2).$$

9.3 Parameter estimation

Given a parametrized response model $Y_{\text{model}}(\cdot;\beta)$ and m noisy datapoints

$$(x_i, Y_i) \quad i = 1, \ldots, m,$$

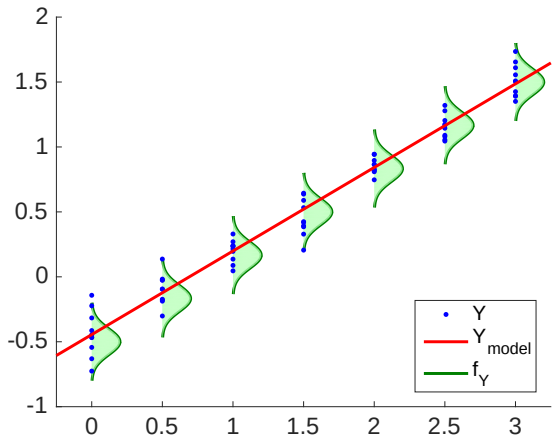


Figure 9.1: Illustration of the regression process for $Y_{\text{model}}(x;\beta) = \beta_0 + \beta_1 x$.

we wish to estimate the true parameter β^{true}. As noted, we assume that our noisy measurements satisfy

$$Y_i = Y_{\text{model}}(x_i;\beta) + \epsilon_i = \beta_0 + \beta_1 x_i + \epsilon_i.$$

We will in fact estimate both the true parameter β^{true} and the noise standard deviation σ from the data.

One way to estimate β^{true} is to consider the *maximum likelihood estimator* (MLE) — the most likely value of the parameter given the measurements. We will denote the MLE by $\hat{\beta}$. The MLE is given by the least-squares fit:

$$\hat{\beta} = \arg\min_{\beta\in\mathbb{R}^2} \|X\beta - Y\|_2,$$

where

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \quad \text{and} \quad Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix}.$$

We recall that the least-squares solution may be found by solving the normal equation,

$$(X^T X)\hat{\beta} = X^T Y,$$

or by QR factorization.

Once we estimate the unknown parameter β^{true} by $\hat{\beta}$, we may estimate the noise standard deviation σ by $\hat{\sigma}$ given by

$$\hat{\sigma} = \left(\frac{1}{m-2}\|\hat{Y} - Y\|_2^2\right)^{1/2} = \left(\frac{1}{m-2}\|X\hat{\beta} - Y\|_2^2\right)^{1/2}.$$

Note that $\|X\hat{\beta} - Y\|$ is the root mean square of the residual, which is minimized in the least squares formulation. The normalization factor $1/(m-2)$ comes from the fact that there are m data points and that our response model has $n = 2$ parameters.

9.4 Confidence intervals: individual

We will consider two different sets of *confidence intervals*. The first is the confidence interval associated with individual parameters: we will construct two confidence intervals I_0 and I_1 associated with the parameters β_0 and β_1, respectively. The confidence interval I_j is an interval such that the probability of the parameter β_j^{true} taking on a value with in the interval is equal to the confidence level γ: i.e.,

$$P(\beta_0^{\text{true}} \in I_0) = \gamma$$

and separately

$$P(\beta_1^{\text{true}} \in I_1) = \gamma.$$

To construct the confidence intervals, we first estimate the covariance of $\hat{\beta}$ by

$$\hat{\Sigma} \equiv \hat{\sigma}^2 (X^T X)^{-1}.$$

From our estimate of the covariance, we may construct the confidence interval for β_0 as

$$I_0 \equiv \left[\hat{\beta}_0 - t_{\gamma,m-2} \sqrt{\hat{\Sigma}_{11}},\ \hat{\beta}_0 + t_{\gamma,m-2} \sqrt{\hat{\Sigma}_{11}} \right]$$

and the confidence interval for β_1 as

$$I_1 \equiv \left[\hat{\beta}_1 - t_{\gamma,m-2} \sqrt{\hat{\Sigma}_{22}},\ \hat{\beta}_1 + t_{\gamma,m-2} \sqrt{\hat{\Sigma}_{22}} \right].$$

Here, the coefficient $t_{\gamma,m-2}$ depends on the confidence level, γ, and the degrees of freedom, $m-2$. Specifically, these two-parameter coefficients satisfy

$$\int_{-t_{\gamma,q}}^{t_{\gamma,q}} f_{T,q}(s) ds = \gamma,$$

where $f_{T,q}$ is the probability density function for the *Student's t-distribution* with q degrees of freedom.

In practice, it is convenient to relate $t_{\gamma,q}$ to the cumulative distribution function of the Student's t-distribution, $F_{T,q}$. By the symmetry of the t-distribution, the relationship is given by

$$t_{\gamma,q} = F_{T,q}^{-1} \left(\frac{1}{2} + \frac{\gamma}{2} \right).$$

Note that the inverse cumulative distribution function of the t distribution, $F_{T,q}^{-1}$ is readily available in MATLAB as `tinv`. (The function is also available in many other statistical analysis libraries). For convenience, we tabulate the coefficients for 95% confidence level for select values of degrees of freedom in Table 9.1.

9.5 Confidence intervals: joint

The second type of confidence interval we consider is the *joint confidence interval*. We will consider the interval I_0^{joint} and I_1^{joint} such that

$$P(\beta_0^{\text{true}} \in I_0^{\text{joint}} \text{ and } \beta_1^{\text{true}} \in I_1^{\text{joint}}) \geq \gamma,$$

(a) t-distribution

q	$t_{\gamma,q}\|_{\gamma=0.95}$
5	2.571
10	2.228
15	2.131
20	2.086
25	2.060
30	2.042
40	2.021
50	2.009
60	2.000
∞	1.960

(b) F-distribution

	$s_{\gamma,p,q}\|_{\gamma=0.95}$							
q	$p=1$	2	3	4	5	10	15	20
5	2.571	3.402	4.028	4.557	5.025	6.881	8.324	9.548
10	2.228	2.865	3.335	3.730	4.078	5.457	6.533	7.449
15	2.131	2.714	3.140	3.496	3.809	5.044	6.004	6.823
20	2.086	2.643	3.049	3.386	3.682	4.845	5.749	6.518
25	2.060	2.602	2.996	3.322	3.608	4.729	5.598	6.336
30	2.042	2.575	2.961	3.280	3.559	4.653	5.497	6.216
40	2.021	2.542	2.918	3.229	3.500	4.558	5.373	6.064
50	2.009	2.523	2.893	3.198	3.464	4.501	5.298	5.973
60	2.000	2.510	2.876	3.178	3.441	4.464	5.248	5.913
∞	1.960	2.448	2.796	3.080	3.327	4.279	5.000	5.605

Table 9.1: The coefficient for computing the 95% confidence interval from Student's t-distribution and F-distribution.

where γ is the confidence level. The joint confidence intervals are given by

$$I_0^{\text{joint}} = \left[\hat{\beta}_0 - s_{\gamma,2,m-2}\sqrt{\hat{\Sigma}_{11}}, \hat{\beta}_0 + s_{\gamma,2,m-2}\sqrt{\hat{\Sigma}_{11}}\right]$$

and

$$I_1^{\text{joint}} = \left[\hat{\beta}_1 - s_{\gamma,2,m-2}\sqrt{\hat{\Sigma}_{11}}, \hat{\beta}_1 + s_{\gamma,2,m-2}\sqrt{\hat{\Sigma}_{11}}\right].$$

Note that the coefficient $t_{\gamma,m-2}$ in the individual confidence interval is replaced by the coefficient $s_{\gamma,2,m-2}$ in the joint confidence interval.

Here, $s_{\gamma,k,q}$ is related to γ-quantile of the F-distribution, $g_{\gamma,k,q}$ by

$$s_{\gamma,k,q} = \sqrt{k g_{\gamma,k,q}} = \sqrt{k F^{-1}_{F,k,q}(\gamma)},$$

where $F_{F,k,q}$ is the cumulative distribution function of the F-distribution. Again, the inverse cumulative distribution function of the F-distribution is readily available in MATLAB.

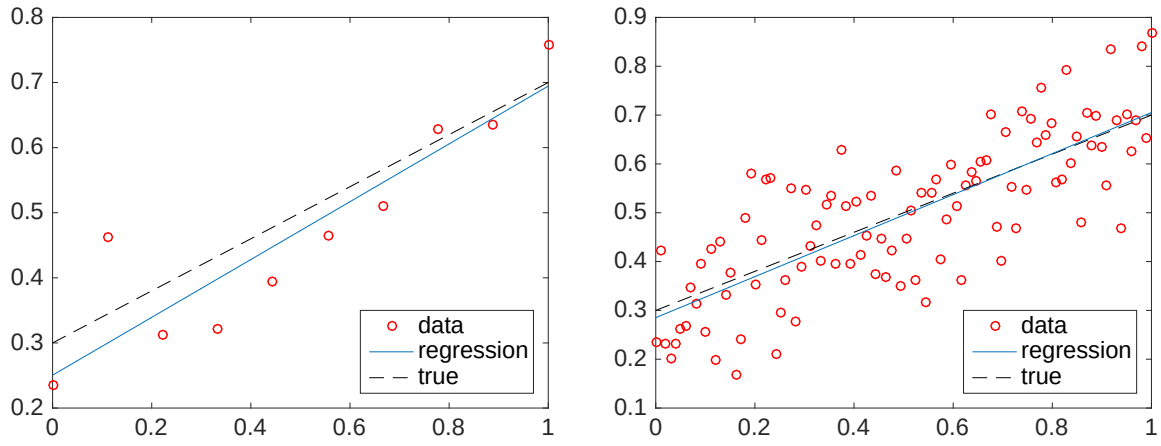
9.6 Linear regression with a linear predictive model: example

We provide a concrete example of the regression process for

$$Y = \beta_0 + \beta_1 x + \epsilon,$$

where $x \in [0,1]$, $\beta_0^{\text{true}} = 0.4$, $\beta_1^{\text{true}} = 0.3$, and $\sigma = 0.1$. The data used in this example is generated synthetically using a pseudo-random number generator.

Given a data vector $Y \in \mathbb{R}^m$, we first solve the 2×2 normal equation, $X^T X\hat{\beta} = X^T Y$, to obtain our MLE for β^{true}. Once we obtain $\hat{\beta}$, we may evaluate our response model $Y_{\text{model}}(x;\hat{\beta})$ for any value of x. An example of predictions provided by $m = 10$ and $m = 100$ are shown in Figure 9.2. Note that the prediction using $m = 100$ points is noticeably more accurate than that for $m = 10$.



(a) $m = 10$ (b) $m = 100$

Figure 9.2: Instances of linear regression using two different values of m. ($Y_{\text{model}} = 0.3 + 0.4x$ and $\sigma = 0.1$.)

We can also build the 95% confidence interval for our estimates of β^{true}. Figure 9.3 shows the confidence intervals for β_0 and β_1 that results from 100 different realizations of data for two different values of m. For $m = 10$, we observe that the confidence interval for β_0 and β_1 are both quite wide. We observe that 96/100 and 95/100 of the confidence intervals for β_0 and β_1, respectively,include the true value; we confirm that the 95% confidence intervals include the true value $\sim 95\%$ of the times as advertised.

For $m = 100$, we observe that the confidence intervals are tighter than the $m = 10$. The half-width of the confidence interval in particular scales as $\sim \sqrt{m}$. We observe that 94/100 and 95/100 of the confidence intervals for β_0 and β_1, respectively, include the true value; we again confirm that the 95% confidence intervals include the true value $\sim 95\%$ of the times as advertised.

9.7 General linear regression

So far we have considered a simple response model of the form

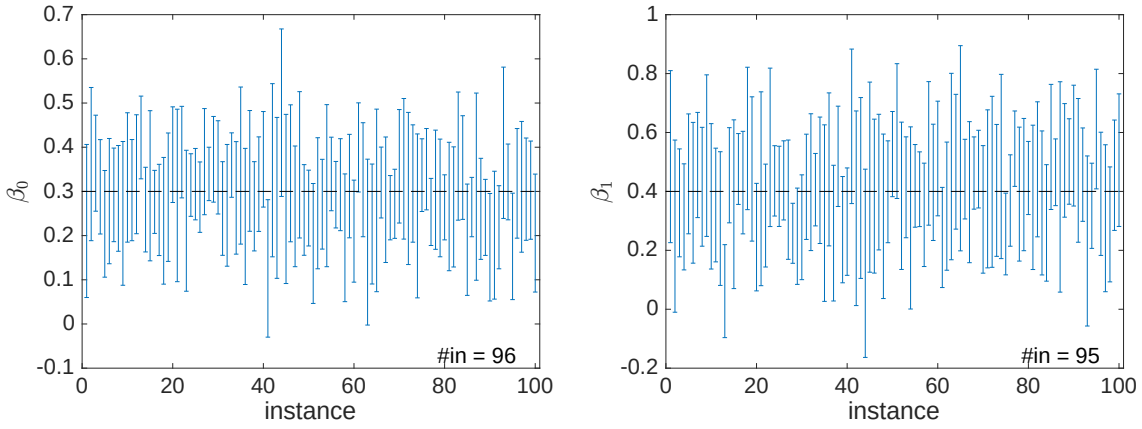
$$Y_{\text{model}}(x; \beta) = \beta_0 + \beta_1 x.$$

We may instead consider a more general model of the form

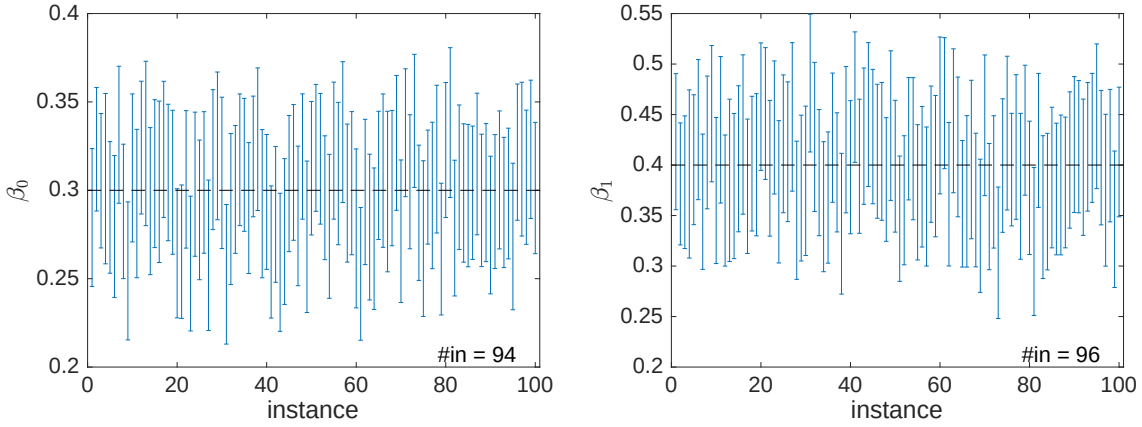
$$Y_{\text{model}}(x; \beta) = \beta_0 + \sum_{j=1}^{n-1} \beta_j h_j(x),$$

where h_j, $j = 0, \ldots, n-1$, are some basis functions and β_j, $j = 0, \ldots, n-1$, are the regression coefficients. As before, we assume that Y_{model} is sufficiently rich such that there exists a parameter $\beta^{\text{true}} \in \mathbb{R}^n$ with which $Y_{\text{model}}(\cdot; \beta^{\text{true}})$ perfectly describes the behavior of the noise-free model. We in addition assume that the noise is normal, homoscedastic, and independent. Our goal is to estimate the unknown parameter β^{true} based on a noisy dataset

$$(x_i, Y_i), \quad i = 1, \ldots, m,$$



(a) $m = 10$



(b) $m = 100$

Figure 9.3: Confidence intervals for β_0 and β_1 for 100 realizations of data. ($Y_{\text{model}} = 0.3 + 0.4x$ and $\sigma = 0.1$.)

for $m \geq n$. Note that even though we permit arbitrary basis functions h_j, $j = 1, \ldots, n-1$, the model Y_{model} still depends linearly on the regression coefficients β_j, $j = 1, \ldots, n$. Hence this is still a linear regression problem.

We as before estimate $\beta^{\text{true}} \in \mathbb{R}^n$ by the maximum likelihood estimator $\hat{\beta} \in \mathbb{R}^n$. The MLE is given by the least-squares problem

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^n} \|X\beta - Y\|_2,$$

where

$$X = \begin{pmatrix} 1 & h_1(x_1) & \cdots & h_{n-1}(x_1) \\ 1 & h_1(x_2) & \cdots & h_{n-1}(x_2) \\ \vdots & \vdots & & \vdots \\ 1 & h_1(x_m) & \cdots & h_{n-1}(x_m) \end{pmatrix} \quad \text{and} \quad Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{pmatrix}.$$

The problem maybe solved by the normal equation or by QR factorization.

We estimate the noise standard deviation σ by

$$\hat{\sigma} = \left(\frac{1}{m-n} \|X\hat{\beta} - Y\|_2^2 \right)^{1/2}.$$

The normalization factor of $1/(m-n)$ comes from the fact that we have m observations and n degrees of freedom. We then estimate the covariance of the parameter β by

$$\hat{\Sigma} = \hat{\sigma}^2 (X^T X)^{-1}.$$

We may then construct the individual confidence intervals

$$I_j = \left[\hat{\beta}_j - t_{\gamma, m-n} \sqrt{\hat{\Sigma}_{j+1,j+1}}, \hat{\beta}_j + t_{\gamma, m-n} \sqrt{\hat{\Sigma}_{j+1,j+1}} \right], \quad j = 0, \ldots, n-1,$$

where $t_{\gamma, m-n} = F_{T, m-n}^{-1}(1/2 + \gamma/2)$ is associated with the Student's t-distribution. (Note that the shifting of the covariance indices is due to the index of the parameter starting from 0.) Each of the individual confidence intervals satisfies

$$P(\beta_j^{\text{true}} \in I_j) = \gamma, \quad j = 0, \ldots, n-1,$$

where γ is the confidence level.

We can also construct the joint confidence intervals

$$I_j = \left[\hat{\beta}_j - s_{\gamma, n, m-n} \sqrt{\hat{\Sigma}_{j+1,j+1}}, \hat{\beta}_j + s_{\gamma, n, m-n} \sqrt{\hat{\Sigma}_{j+1,j+1}} \right], \quad j = 0, \ldots, n-1,$$

where $s_{\gamma, n, m-n} = \sqrt{n F_{F, n, m-n}^{-1}(\gamma)}$ is associate with the F-distribution. The joint confidence interval satisfies

$$P(\beta_0^{\text{true}} \in I_0^{\text{joint}} \quad \text{and} \cdots \quad \beta_{n-1}^{\text{true}} \in I_{n-1}^{\text{joint}}) \geq \gamma.$$

9.8 Summary

We summarize the key points of this lecture:

1. Given a parametrized predictive model and a set of noisy observations, we can estimate the parameter of the model using linear regression.
2. The key assumptions required in the regression process are i) the model is unbiased and ii) the noise is zero-mean normal, homoscedastic, and independent.
3. Under the above assumptions, the maximum-likelihood estimate of the parameter is found by solving a least-squares problem.
4. Under the above assumptions, we can provide confidence intervals for the parameters of the model.
5. The confidence interval scales as $1/\sqrt{m}$, where m is the number of observations: the convergence is rather slow.
6. Linear regression can be performed on predictive model whose output depends nonlinearly on the independent variables as long as the model is linear in the parameters.

Lecture 10

Nonlinear equations

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

10.1 Roots of scalar nonlinear equations

We first recall the definition of a root: given a function $f : \mathbb{R} \to \mathbb{R}$, $x^\star$ is a root (or zero) of f if

$$f(x^\star) = 0.$$

Then, the goal of a *root-finding problem* is to find a root of the function.

10.2 Bisection method

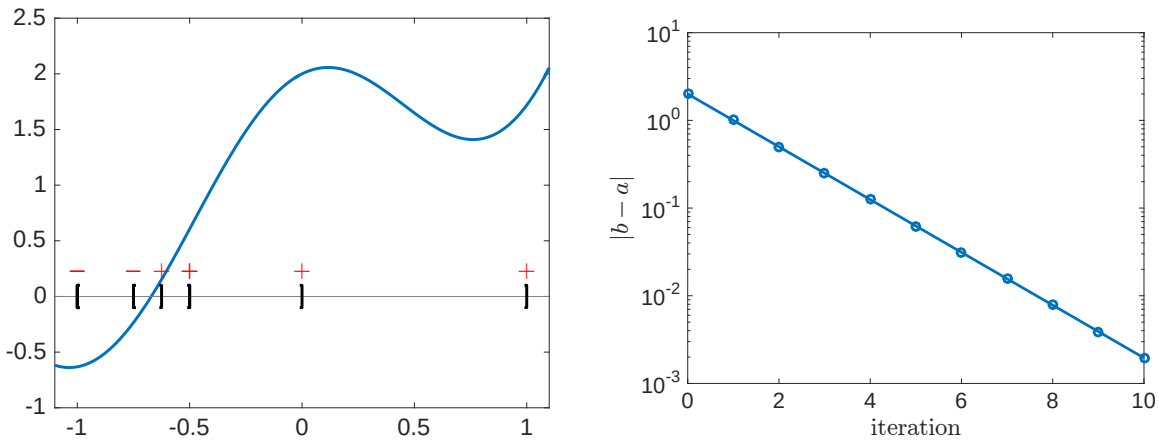
Bisection method is a root-finding method based on the idea of *bracketing* — the idea of iteratively refining an interval in which a root lies. The bisection method requires two initial guesses, a^0 and b^0, such that the signs of $f(a^0)$ and $f(b^0)$ are different. If f is continuous, then the different signs of $f(a^0)$ and $f(b^0)$ implies that there is a root in the interval $[a^0, b^0]$. The bisection method is based on an iterative refinement of this interval that is guaranteed to contain a root: a single step of the bisection method is the following:

1. set $c^i = (a^i + b^i)/2$ and evaluate $f(c^i)$
2. if $\text{sign}(f(c^i)) = \text{sign}(f(a^i))$, then set $a^{i+1} = c^i$ and $b^{i+1} = b^i$;
   otherwise, set $a^{i+1} = a^i$ and $b^{i+1} = c^i$

In the second step, if $\text{sign}(f(c^i)) \neq \text{sign}(f(a^i))$, then $\text{sign}(f(c^i)) = \text{sign}(f(b^i))$ since $f(a^i)$ and $f(b^i)$ have opposite signs. In addition, after the update (and hence in the next iteration), $f(a^{i+1})$ and $f(b^{i+1})$ are again guaranteed to have opposite signs since the midpoint (c^i) which replaces one the endpoints (a^i or b^i) have the same sign as the endpoint. Figure 10.1(a) illustrates the bisection method.

Termination criteria. Because the bisection method is an iterative algorithm, we must provide a termination criteria. There are two commonly used termination criteria. One is based on the length of the interval:

$$|b^i - a^i| \leq \delta^x_{\text{tol}}$$



(a) illustration

(b) convergence

Figure 10.1: The illustration and error convergence of the bisection method.

for some prescribed tolerance δ^x_{tol}. Note that this criterion guarantees that $|x^\star - a^i| \leq \delta^x_{\text{tol}}$ and $|x^\star - b^i| \leq \delta^x_{\text{tol}}$. The other is based on the function value at one of the approximations (say a^i):

$$|f(a^i)| \leq \delta^f_{\text{tol}}$$

for some prescribed tolerance δ^f_{tol}.

Convergence. In the bisection method, both of the errors $|x^\star - a^i|$ and $|x^\star - b^i|$ are guaranteed to be smaller than the length of the interval $|b^i - a^i|$. In addition, by construction the interval $[a^i, b^i]$ is guaranteed to contain at least one root. Because of the length of the interval is halved in each iteration, the length of the interval after the i-th iteration is

$$|b^i - a^i| = \frac{1}{2^i}|b^0 - a^0|.$$

Hence, the error converges exponentially with the number of iterations, as shown in Figure 10.1(b). As we will see shortly, the bisection method is one of few methods with a guaranteed convergence property.

Advantages and disadvantages. Perhaps the biggest advantage of the bisection method is that, for continuous functions, it is *guaranteed* to converge assuming the initial bracket $[a^0, b^0]$ can be found. Unfortunately, the convergence is not as some other methods, and the scheme does not extend naturally to higher dimensions.

10.3 Newton's method

Newton's method, which is also known as the Newton-Raphson method, is another iterative procedure to find a root of a nonlinear function. Given a initial guess x^0 for a root, Newton's method update the value according to

$$x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)},$$

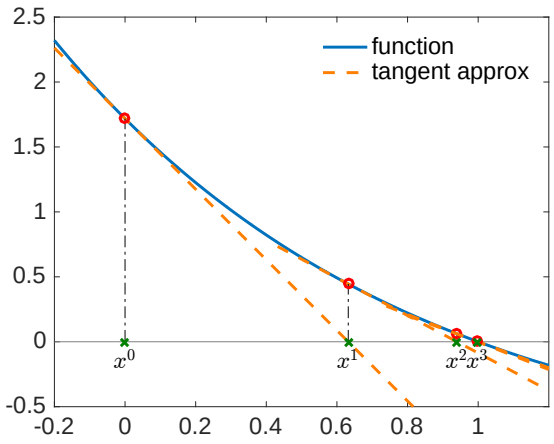


Figure 10.2: An illustration of Newton's method applied to $\exp(1-x)-1$ starting from $x^0 = 0$.

where $f'(x^i)$ is the derivative of f evaluated at x^i.

Derivation. The i-th step of Newton's method is motivated by a linear approximation of the nonlinear function about the current state x^i. To see this, we first note by the Taylor expansion

$$f(x) \approx f(x^i) + f'(x^i)(x - x^i) + \mathcal{O}(|x - x^i|^2).$$

We then neglect the $\mathcal{O}(|x - x^i|^2)$ and set x^{i+1} to be the root of the linear approximation:

$$0 = f(x^i) + f'(x^i)(x^{i+1} - x^i).$$

Assuming $f'(x^i) \neq 0$, we can readily solve for x^{i+1} to obtain

$$x^{i+1} = x^i - \frac{f(x^i)}{f'(x^i)},$$

which is precisely a single step of Newton's method. Figure 10.2 illustrates Newton's method as a method that sequentially solves linearized problems.

Termination criteria. There are a few different types of termination criteria commonly used in Newton's method. One approach is to compute the size of the update taken in a single step and to terminate if the update is small:

$$|x^{i+1} - x^i| \leq \delta^x_{\text{tol}}$$

for some prescribed tolerance δ^x_{tol}. Another approach is to ensure that the function value is sufficiently close to zero:

$$|f(x^i)| \leq \delta^f_{\text{tol}},$$

for some prescribed tolerance δ^f_{tol}.

Computational requirement. A single step of Newton's method requires the evaluation of $f(x^i)$ and $f'(x^i)$. Note that we must have means of evaluating both the function and the derivative to use Newton's method.

Convergence. The following theorem summarizes the convergence of Newton's method:

Theorem 10.1. Suppose the function $f : \mathbb{R}^n \to \mathbb{R}^n$ is twice differentiable on $I \equiv [x^\star - r, x^\star + r]$ and there exist constants α and β such that

1. $|f'(x)| \geq \alpha, \quad \forall x \in I$
2. $|f''(x)| \leq \beta, \quad \forall x \in I$
3. $r \leq \frac{\alpha}{\beta}$.

If $x^i \in I$, then Newton update x^{i+1} satisfies

$$|x^\star - x^{i+1}| \leq \frac{\beta}{2\alpha}|x^\star - x^i|^2 \leq \frac{1}{2}|x^\star - x^i|.$$

Proof. We first note that

$$0 = f(x^\star) = f(x^i) + f'(x^i)(x^\star - x^i) + \frac{1}{2}f''(\xi^i)(x^\star - x^i)^2$$

for some $\xi^i \in [x^i, x^\star]$ (if $x^i < x^\star$, or $[x^\star, x^i]$ if $x^\star < x^i$). We now add "0" to the equation and rearrange the expression

$$\begin{aligned} 0 &= f(x^i) + f'(x^i)(x^\star - x^i) + \frac{1}{2}f''(\xi^i)(x^\star - x^i)^2 - (\underbrace{f(x^i) + f'(x^i)(x^{i+1} - x^i)}_{=0 \,:\, \text{Newton's update step}}) \\ &= f'(x^i)(x^\star - x^{i+1}) + \frac{1}{2}f''(\xi^i)(x^\star - x^i)^2. \end{aligned}$$

Assuming $f'(x^i) \neq 0$, it follows

$$x^\star - x^{i+1} = -\frac{1}{2}\frac{f''(\xi^i)}{f'(x^i)}(x^\star - x^i)^2.$$

We now take the absolute value of the equation and invoke assumptions 1 and 2 to obtain

$$|x^\star - x^{i+1}| \leq \frac{\beta}{2\alpha}|x^\star - x^i|^2;$$

This proves the first inequality. In addition, for $x^i \in I \equiv [x^\star - r, x^\star + r] \equiv [x^\star - \alpha/\beta, x^\star + \alpha/\beta]$, we note that

$$|x^\star - x^i| \leq r = \frac{\alpha}{\beta}.$$

It thus follows that

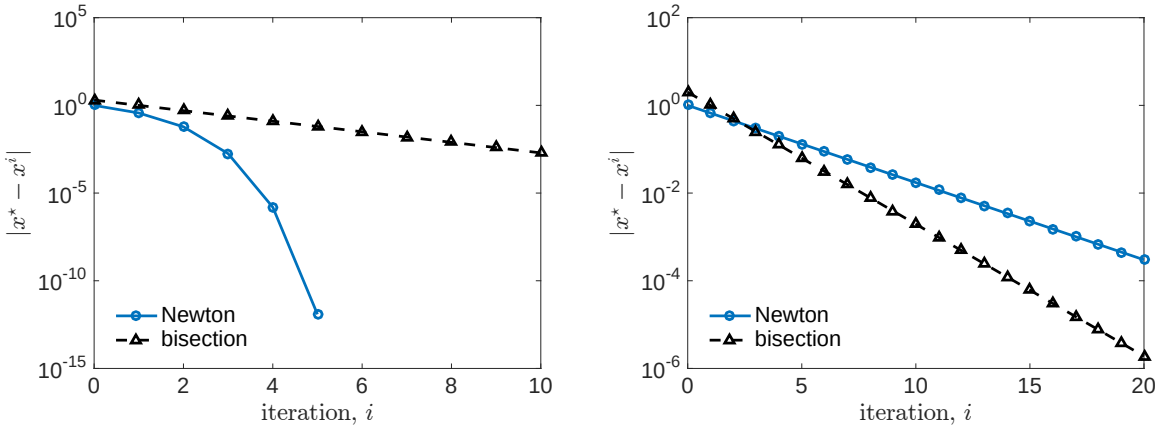
$$|x^\star - x^{i+1}| \leq \frac{1}{2}|x^\star - x^i|,$$

which is the second inequality. □

We now provide an interpretation of the inequalities. First, the "outer" inequality,

$$|x^\star - x^{i+1}| \leq \frac{1}{2}|x^\star - x^i|,$$

states that, *if* x^i is in the interval I, then the error decreases by a factor of at least 2. Hence, once x^i is in the interval I, Newton's method is guaranteed to converge.



(a) $f(x) = \exp(1-x) - 1$, $x^0 = 0$ (b) $f(x) = (x-1)^3$, $x^0 = 0$

Figure 10.3: Convergence of Newton's method for favorable and unfavorable cases.

Second, the "inner" inequality,

$$|x^\star - x^{i+1}| \leq \frac{\beta}{2\alpha}|x^\star - x^i|^2,$$

states that, under favorable conditions, Newton's method converge *quadratically*. Quadratic convergence is a very rapid convergence: for instance if the first two digits are correct in the i-th step of Newton, then the first four digits will be correct in the next step, and the first eight digits will be correct in the following step. In words, the number of correct digits doubles every iteration. Figure 10.3(a) shows the convergence of Newton's method for the function $f(x) = \exp(1-x) - 1$ starting from $x^0 = 0$. We observe that Newton's method converge very rapidly, reducing the error from $\mathcal{O}(10^{-3})$ to $\mathcal{O}(10^{-6})$ to $\mathcal{O}(10^{-12})$ in the last three steps.

On the other hand, if any of the three assumptions in the theorem is violated, then Newton's method might converge slowly (i.e. not quadratically) or might not converge to a root at all. Figure 10.3(b) shows the convergence behavior for the function $f(x) = (x-1)^3$, which violates the first condition as $f'(x^\star = 1) = 0$. We observe that Newton's method converge rather slowly, and in fact the convergence is slower than that observed for the bisection method.

In general, the convergence of Newton's method is highly dependent on the initial guess x^0. Figure 10.4 shows the convergence (or diverge) of Newton's method for $f(x) = \arctan(x)$. Newton's method converges for the initial guess of $x^0 = 0.5$ and $x^0 = 1.3$, but it diverges for the initial guess of $x^0 = 5.0$. (Note however that the conditions in the theorem are sufficient conditions and not necessary conditions; Newton's method may still converge rapidly even if the conditions are not met.)

10.4 Secant method: a quasi-Newton method

One limitation of Newton's method is that it requires the evaluation of the derivative $f'(x^i)$ in each step. The evaluation of the derivative may be expensive, or an expression for the derivative simply may not be available. In these cases, we may consider a *quasi-Newton method*, which does not

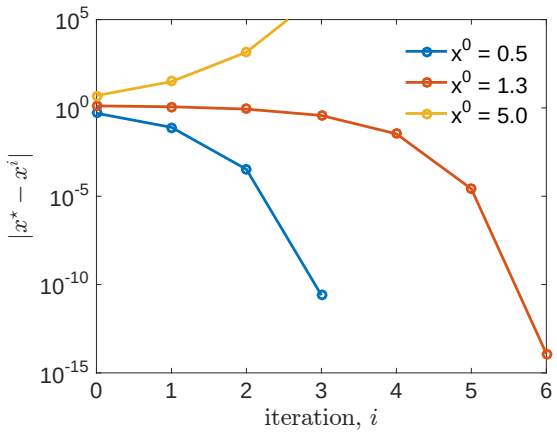


Figure 10.4: Convergence of Newton's method to the root $x^\star = 0$ of $f(x) = \arctan(x)$ using three different initial values.

require the evaluation of the derivative. An example of quasi-Newton method for scalar equations is the *secant method*. Given initial guesses x^{-1} and x^0, a single step of secant method is given by

$$x^{i+1} = x^i - \frac{x^i - x^{i-1}}{f(x^i) - f(x^{i-1})} f(x^i).$$

Note that, unlike Newton's method, the initialization of the secant method requires two guesses.

Derivation. The secant method is obtained from Newton's method by replacing $f'(x^i)$ with its approximation. Specifically, we approximate the derivative by a finite difference,

$$f'(x^i) \approx \frac{f(x^i) - f(x^{i-1})}{x^i - x^{i-1}}.$$

The substitution of the derivative approximation to Newton's method yields the secant method.

Computational requirement. Unlike Newton's method, the secant method only requires the evaluation of the function value $f(x^i)$ in the i-th step. (We assume the value of $f(x^{i-1})$ is stored from the previous step.) The fact that the method does not require the evaluation of the derivative $f'(x^i)$ is a significant computational advantage.

Convergence. Because the secant method uses an approximate derivative based on a finite difference as opposed to the exact derivative, it does not provide quadratic convergence. The instead converges as

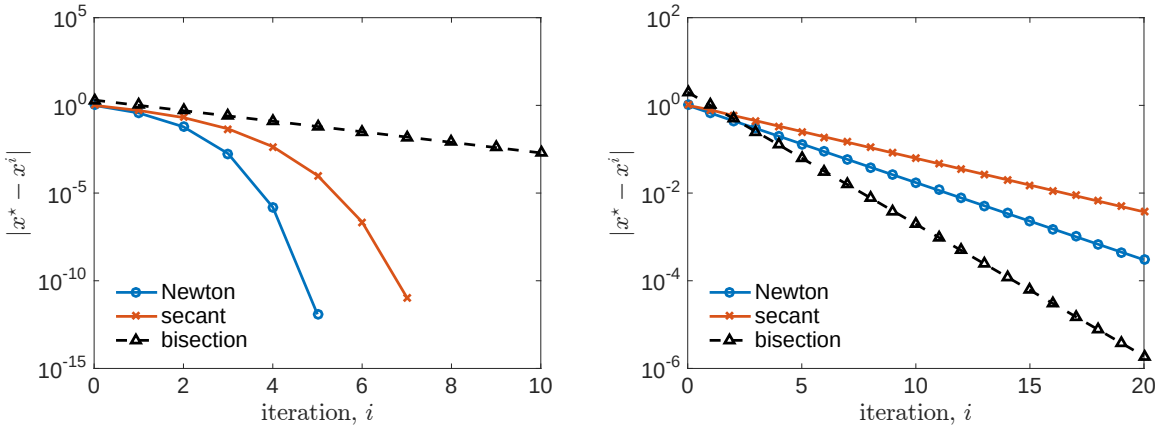
$$|x^\star - x^{i+1}| \leq C|x^\star - x^i|^k \qquad \text{for} \qquad k = (1 + \sqrt{5})/2 \approx 1.618.$$

(We here omit the proof for brevity.) Note that the convergence is still superlinear. Figure 10.5(a) shows the convergence behavior of the secant method applied to $\exp(1 - x) - 1$

10.5 System of nonlinear equations: Newton's method

Given a vector-valued nonlinear function $f : \mathbb{R}^n \to \mathbb{R}^n$, we wish consider a root finding problem: find $x \in \mathbb{R}^n$ such that

$$f(x) = 0.$$



(a) $f(x) = \exp(1-x) - 1$, $x^0 = 0$, $x^{-1} = -1/2$ (b) $f(x) = (x-1)^3$, $x^0 = 0$, $x^{-1} = -1/2$

Figure 10.5: Convergence of the secant method for favorable and unfavorable cases.

Note that here the right-hand side is a zero vector in $\mathbb{R}^n$. More explicitly (and visually), the system of nonlinear equations we wish to solve is

$$\begin{pmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{pmatrix} = 0,$$

where $f_i(x)$ is the i-th component of the vector-valued function $f(x)$ evaluated at x.

Before we introduce Newton's method for multivariate functions, we introduce the *Jacobian matrix*, $\nabla f(x) \in \mathbb{R}^{n \times n}$, associated with the function $f : \mathbb{R}^n \to \mathbb{R}^n$. We recall that the (k, l) entry of the Jacobian $\nabla f(x)$ is

$$(\nabla f(x))_{kl} = \frac{\partial f_k}{\partial x_l}(x),$$

which is the partial derivative of the k-th component of $f : \mathbb{R}^n \to \mathbb{R}^n$ with respect to the l-th component of the input. More explicitly (and visually), the Jacobian matrix is given by

$$\nabla f \equiv \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix},$$

which is a $n \times n$ matrix.

We can now generalize Newton's method that we considered for scalar equations to vector-valued functions in a straightforward manner. Given a initial guess $x^0 \in \mathbb{R}^n$, Newton's method update the vector according to

$$x^{i+1} = x^i - (\nabla f(x^i))^{-1} f(x^i),$$

where $\nabla f(x^i) \in \mathbb{R}^{n\times n}$ is the Jacobian matrix of f evaluated at x^i. More explicitly (and visually), the update is given by

$$\begin{pmatrix} x_1^{i+1} \\ x_2^{i+1} \\ \vdots \\ x_n^{i+1} \end{pmatrix} = \begin{pmatrix} x_1^{i} \\ x_2^{i} \\ \vdots \\ x_n^{i} \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}^{-1}_{x^i} \begin{pmatrix} f_1(x^i) \\ f_2(x^i) \\ \vdots \\ f_n(x^i) \end{pmatrix},$$

where the subscript x^i on the Jacobian matrix indicates the matrix is evaluated at $x = x^i$. Note that we do not explicitly form the inverse of the Jacobian matrix, but rather solve the linear system $\nabla f(x^i)\delta x = f(x^i)$ to compute the action of $\nabla f(x^i)^{-1}$ on $f(x^i)$.

Derivation. The i-th step of Newton's method is again based on a linear approximation of the nonlinear function about x^i. To see this, we first note by the Taylor expansion

$$f(x) \approx f(x^i) + \nabla f(x^i)(x - x^i) + \mathcal{O}(\|x - x^i\|^2).$$

We then neglect the $\mathcal{O}(\|x - x^i\|^2)$ term and set x^{i+1} to be the root of the linear approximation:

$$f(x^i) + \nabla f(x^i)(x^{i+1} - x^i) = 0;$$

assuming $\nabla f(x^i)$ is non-singular, the x^{i+1} is given by

$$x^{i+1} = x^i - (\nabla f(x^i))^{-1} f(x^i),$$

which is precisely a single step of Newton's method.

Convergence criteria. As in the case of scalar equations, one simple test to check for the convergence of the Newton's method is to monitor the size of the update. Specifically, we may consider the solution "converged" if the 2-norm of the update is smaller than some user specified tolerance: $\|x^{i+1} - x^i\|_2 \leq \delta^x_{\text{tol}}$. Alternatively, we may also assess the convergence based on the 2-norm of function value, $\|f(x^{i+1})\|_2 \leq \delta^f_{\text{tol}}$.

Procedure. We now identify the steps required for Newton's method for vector-valued functions:

0. Make an initial guess $x^0 \in \mathbb{R}^n$. Set $i = 0$.

1. Evaluate the function and Jacobian:

$$f(x^i) \in \mathbb{R}^n \quad \text{and} \quad \nabla f(x^i) \in \mathbb{R}^{n\times n}.$$

2. Solve a linear system for $\delta x^i \equiv x^{i+1} - x^i$:

$$\nabla f(x^i)\delta x^i = -f(x^i).$$

3. Update the solution

$$x^{i+1} = x^i + \delta x^i.$$

4. Check for convergence. If $\|\delta x^i\|_2 \leq \delta^x_{\text{tol}}$ then terminate; otherwise set $i \leftarrow i + 1$ and go back to Step 1.

Computational cost. The multi-dimensional version of Newton's method requires the evaluation of function value $f(x^i) \in \mathbb{R}^n$ and the Jacobian $\nabla f(x^i) \in \mathbb{R}^{n \times n}$ in each step. This implies that we must have a means of evaluating the Jacobian and compute all of its n^2 entries. In addition, to compute the state update, we must solve a $n \times n$ linear system

$$(\nabla f(x^i))\delta x^i = -f(x^i);$$

we can solve the system using, for instance, Gaussian elimination, which requires $\approx \frac{2}{3}n^3$ operations (for a dense linear system).

10.6 Quasi-Newton method for system of nonlinear equations

We have so far introduced the secant method as an example of quasi-Newton method for scalar equations. There also exists a number of quasi-Newton methods for a system of nonlinear equations. These multi-dimensional quasi-Newton methods approximate the Jacobian $\nabla f(x^i) \in \mathbb{R}^{n \times n}$ using a sequence of function vectors $f(x^i), f(x^{i-1}), \ldots$. One popular method is the Broyden-Fletcher-Goldfarb—Shanno (BFGS) algorithm. However, the coverage of these quasi-Newton methods is beyond the scope of this lecture.

10.7 Summary

We summarize the key points of this lecture:

1. Bisection method is a root-finding method that is based on iteratively refining a bracket in which a root lies.
2. Bisection method is guaranteed to converge for continuous functions, but does not extend naturally to higher dimensions.
3. Newton's method is a root-finding method that uses both the function value and gradient in each step.
4. Depending on the function and the initial guess, Newton's method may converge quadratically, converge linearly, or diverge.
5. Secant method is an example of quasi-Newton method, in which the gradient used in a Newton step is replaced by a finite-difference derivative.
6. Newton's method readily extends to higher dimensions; the generalization relies on the Jacobian of the vector-valued function.
7. Newton's method in higher dimension can be computationally expensive, as it requires the evaluation of a $n \times n$ Jacobian matrix and the solution of the associated $n \times n$ linear system.

Lecture 11

Optimization: (very) brief overview

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

11.1 Introduction

Optimization problems are ubiquitous in engineering and science. One example in aerospace engineering is aerodynamic shape optimization: here, we wish to optimize the shape of the aircraft such that we can minimize the drag at a given lift subject to structural constraints.

Mathematically, a general optimization problem can be stated as the following:

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} g(x) \\ \text{subject to } &c_1(x) \le 0 \\ &c_2(x) = 0. \end{aligned}$$

We identify different components of the statement:

$x \in \mathbb{R}^n$: design variable

$g : \mathbb{R}^n \to \mathbb{R}$: objective function which we wish to minimize

$c_1 : \mathbb{R}^n \to \mathbb{R}^p$: inequality constraints

$c_2 : \mathbb{R}^n \to \mathbb{R}^q$: equality constraints .

In words, we wish to find a n-vector of the design variable which minimizes the objective function subject to p inequality constraints and q equality constraints.

We now make a few remarks. By convention, optimization problems are expressed as a minimization problem; if we wish to maximize some objective, we can then minimize the negative of the objective. In addition, the inequality constraints are expressed as upper bound constraints; if we wish to impose a lower bound constraint, we can then impose an upper bound constraint on the negative of the constraint function. Furthermore, we may remove the equality constraints without loss of generality; we could express any equality constraint as two inequality constraints, one bounding the function from the above and the other bounding the function from the below. Finally, mathematical optimization is often also called *mathematical programming*. (Note this is unrelated to "programming" in the sense of "coding.")

11.2 Newton's method for unconstrained optimization

In this brief overview to mathematical optimization, we consider arguably the simplest class of optimization problems: we look for

1. local minimum
2. of smooth functions
3. without a constraint.

We recall from single-variable calculus for smooth functions that any local minimum must satisfy two conditions: the first derivative of the function must be zero; the second derivative of the function must be positive. For a smooth function of multiple variables, sufficient (but not necessary) conditions for $x^\star \in \mathbb{R}^n$ to be a local minimum are the following:

1. The gradient of the function vanishes:

$$\nabla g(x^\star) = 0,$$

   or, more explicitly,

$$\left(\begin{array}{c} \frac{\partial g}{\partial x_1} \\ \vdots \\ \frac{\partial g}{\partial x_n} \end{array} \right) \Bigg|_{x^\star} = 0.$$

2. The *Hessian* of the function is symmetric positive definite (SPD):

$$\nabla^2 g(x^\star) \text{ is SPD},$$

   where the Hessian may be expressed as

$$\nabla^2 g(x) = \left(\begin{array}{cccc} \frac{\partial^2 g}{\partial x_1^2} & \frac{\partial^2 g}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n} \\ \frac{\partial^2 g}{\partial x_2 \partial x_1} & \frac{\partial^2 g}{\partial x_2^2} & \cdots & \frac{\partial^2 g}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 g}{\partial x_n \partial x_1} & \frac{\partial^2 g}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 g}{\partial x_n^2} \end{array} \right).$$

We observe that the first condition is equivalent to seeking the root of a vector-valued function, $\nabla g : \mathbb{R}^n \to \mathbb{R}^n$. Hence, we can solve the problem using Newton's method for system of nonlinear equations. (Here, ∇g plays the role of $f : \mathbb{R}^n \to \mathbb{R}^n$ in the previous chapter.) We can then check if the Hessian is SPD at the stationary point to assess if the point is a local minimum.

Proof. We now show that the above two conditions are sufficient condition for $x^\star \in \mathbb{R}^n$ to be a local minimum. Towards this end, we consider the Taylor expansion of $g : \mathbb{R}^n \to \mathbb{R}$ about $x^\star \in \mathbb{R}^n$ and make two observations:

$$g(x) = g(x^\star) + \underbrace{(x - x^\star)^T \nabla g(x^\star)}_{= 0 \text{ by the first condition}} + \underbrace{\frac{1}{2}(x - x^\star)^T \nabla^2 g(x^\star)(x - x^\star)}_{> 0 \text{ by the second condition}} + \mathcal{O}(\|x - x^\star\|_2^3).$$

It follows that $g(x) > g(x^\star)$ for $\|x - x^\star\|_2$ sufficiently small. Hence $x^\star$ is a local minimum of g.

Derivation of Newton's method. To derive Newton's method to seek the stationary point $x^\star \in \mathbb{R}^n$ such that $\nabla g(x^\star) = 0$, we first note that

$$g(x) = g(x^i) + (x - x^i)^T \nabla g(x^i) + \frac{1}{2}(x - x^i)^T \nabla^2 g(x^i)(x - x^i) + \mathcal{O}(\|x - x^i\|^3).$$

We now take the gradient of the expression to obtain

$$\nabla g(x) = \nabla g(x^i) + \nabla^2 g(x^i)(x - x^i) + \mathcal{O}(\|x - x^i\|^2).$$

As before, we now neglect the second (and higher) order contributions and choose the point $x^{i+1} \in \mathbb{R}^n$ such that the x^{i+1} sets the linearized approximation of $\nabla g(x)$ equal to zero:

$$0 = \nabla g(x^i) + \nabla^2 g(x^i)(x^{i+1} - x^i).$$

More explicitly, we may express the iterate x^{i+1} as

$$x^{i+1} = x^i - [\nabla^2 g(x^i)]^{-1} \nabla g(x^i).$$

(Note: even though we write the update step here in terms of the inverse of the Hessian, $\nabla^2 g(x^i)$, we note that in practice the inverse is *never* explicitly computed; we instead solve the linear system for the update $\delta x^i \equiv x^{i+1} - x^i$ as described below.) This update process is repeated until a convergence criterion is met. Typical convergence criteria used are $\|x^{i+1} - x^i\|_2 \leq \delta_{\text{tol}}^x$ and/or $\|\nabla g(x^i)\|_2 \leq \delta_{\text{tol}}^{\nabla g}$.

Classification of a stationary point. For smooth functions, we can classify a stationary point (i.e. $x^\star$ such that $\nabla g(x^\star) = 0$) based on the eigenvalues of the Hessian $\nabla^2 g(x^\star) \in \mathbb{R}^{n \times n}$ as follows:

- All eigenvalues are (strictly) positive; i.e. $\nabla^2 g(x^\star)$ is SPD. The stationary point is a local minimum.
- At least one eigenvalue is (strictly) negative eigenvalue. The stationary point is *not* a local minimum, as there is at least one decent direction.
- All eigenvalues are non-negative and at least one of them is zero. The test is inconclusive; the point may or may not be a local minimum.

11.3 Computational procedure

We summarize the computational procedure:

0. Make an initial guess $x^0 \in \mathbb{R}^n$. Set $i = 0$.
1. Evaluate the gradient and Hessian:

$$\nabla g(x^i) \in \mathbb{R}^n \quad \text{and} \quad \nabla^2 g(x^i) \in \mathbb{R}^{n \times n}.$$

2. Compute the update $\delta x^i \equiv x^{i+1} - x^i$ by solving a linear system

$$\nabla^2 g(x^i) \delta x^i = -\nabla g(x^i).$$

3. Update the solution
$$x^{i+1} = x^i + \delta x^i.$$

4. Check for convergence to a stationary point. If $\|\delta x^i\|_2 \leq \delta_{\text{tol}}$ then terminate; otherwise go back to Step 1.

5. Classify the stationary point. Evaluate the eigenvalues of the Hessian and determine if the stationary point is a local minimum. If the point is not a local minimum, go back to Step 0 and make a different initial guess.

Computational cost. As in the Newton's method for vector-valued functions, the two computationally expensive steps to find a stationary point are i) the evaluation of the gradient and the Hessian in Step 1 and ii) the solution of the linear system in Step 2. In addition, in the context of optimization, the solution of the eigenproblem required to classify the stationary point is also computationally expensive.

11.4 Summary

We summarize the key points of this lecture:

1. There are many different classes of optimization problems. In this lecture we considered Newton's method to find a local minimum of smooth functions without any constraints.

2. A stationary point — a point $x^\star \in \mathbb{R}^n$ such that $g(x^\star) = 0$ — can be obtained by using Newton's method.

3. In some cases, the Hessian at the stationary point, $\nabla^2 g(x^\star)$, informs whether the point is a local minimum; however, the test may be inclusive.

Lecture 12

Numerical differentiation

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

12.1 Motivation

We have considered in earlier lectures numerical approximation of functions and integrals by interpolation and quadrature, respectively. In this lecture we consider numerical approximation of derivatives by finite difference formulas. Specifically, given function values $(x_i, f(x_i))$ evaluated at a finite set of points, we wish to estimate the derivative at one of those points. The finite difference techniques we introduce in this lecture play a key role in the development of numerical methods for ODEs and PDEs.

12.2 First derivative: forward difference

We first consider an approximation of

$$f_i' \equiv f'(x_i) \equiv \left.\frac{\partial f}{\partial x}\right|_{x\in x_i}$$

by

$$(x_i, f_i) \quad \text{and} \quad (x_{i+1}, f_{i+1}).$$

(For notational convenience, we denote $f(x_i)$ by f_i and $f'(x_i)$ by f_i'.) The *forward difference* approximation of the first derivative is given by

$$f'(x_i) \approx \frac{f_{i+1} - f_i}{\Delta x},$$

where $\Delta x \equiv x_{i+1} - x_i$. As the name implies, the derivative is approximated by function evaluated at points greater than or equal to x_i.

Error analysis. We may readily analyze the *truncation error* associated with the approximation by Taylor series. Towards this end, we note that a Taylor series expansion of f_{i+1} is given by

$$f_{i+1} = f_i + f_i'\Delta x + \frac{1}{2}f''(\xi)\Delta x^2$$

for some $\xi \in [x_i, x_{i+1}]$. The truncation error is hence bounded by

$$\epsilon \equiv |f_i' - \frac{f_{i+1} - f_i}{\Delta x}| = |f_i' - \frac{1}{\Delta x}\left(f_i + f_i'\Delta x + \frac{1}{2}f''(\xi)\Delta x^2 - f_i\right)| \leq \frac{1}{2}\max_{s\in[x_i,x_{i+1}]}|f''(s)|\Delta x.$$

The inequality follows from $|f''(\xi)| \leq \max_{s\in[x_i,x_{i+1}]}|f''(s)|$ for any $\xi \in [x_i, x_{i+1}]$. We note that the truncation error is bounded by

$$\epsilon \leq C\Delta x,$$

and hence the approximation is *first-order accurate*; i.e., the error is bounded by a constant times Δx^1.

Cost analysis. The cost associated with a given finite difference formula is characterizes in terms of its *stencil* — a set of points involved in the formula. Because the forward difference formula uses the function values at x_i and x_{i+1}, the formula is said to have a *two-point stencil*.

12.3 First derivatives: backward difference and higher-order approximations

There are many other ways to approximate the first derivative. We here introduce several alternatives.

Backward difference. The *backward difference formula* approximates f_i' based on the function values at x_i and x_{i-1} (as opposed to at x_{i+1} for the forward difference). The backward difference formula is

$$f'(x_i) \approx \frac{f_i - f_{i-1}}{\Delta x}.$$

The backward difference formula also has a two-point stencil; however the evaluation points are different.

We may readily analyze the truncation error following the same procedure as before. We first note that by a Taylor series expansion

$$f_{i-1} = f_i - f_i'\Delta x + \frac{1}{2}f''(\xi)\Delta x$$

for some $\xi \in [x_{i-1}, x_i]$. We then note that

$$\epsilon = f_i' - \frac{f_i - f_{i-1}}{\Delta x} \leq \frac{1}{2}\max_{s\in[x_{i-1},x_i]}|f''(s)|\Delta x.$$

The backward difference formula is first order accurate.

Central difference. The *central difference formula* approximates f_i' based on the function values at x_{i-1} and x_{i+1}. The formula is given by

$$f_i' \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x}.$$

Note that the formula uses a *three-point stencil*.

In order to analyze the truncation error, we consider Taylor series expansions for f_{i+1} and f_{i-1}:

$$f_{i+1} = f_i + f_i'\Delta x + \frac{1}{2}f_i''\Delta x^2 + \frac{1}{6}f'''(\xi_1)\Delta x^3$$
$$f_{i-1} = f_i - f_i'\Delta x + \frac{1}{2}f_i''\Delta x^2 + \frac{1}{6}f'''(\xi_2)\Delta x^3$$

for some $\xi_1 \in [x_i, x_{i+1}]$ and $\xi_2 \in [x_{i-1}, x_i]$. We then note that

$$\epsilon \equiv |f_i' - \frac{f_{i+1} - f_{i-1}}{2\Delta x}| = \frac{1}{12}(f'''(\xi_1)\Delta x^2 + f'''(\xi_2)\Delta x^2) \leq \frac{1}{6} \max_{s \in [x_{i-1}, x_{i+1}]} |f'''(s)|\Delta x^2.$$

The truncation error is bounded by $\epsilon \leq C\Delta x^2$; the central difference formula is *second-order accurate*.

Second-order forward difference. The *second-order forward difference formula* approximates f_i' based on the function values at x_i, x_{i+1}, and x_{i+2}. The formula is given by

$$f_i' \approx \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2\Delta x}.$$

Using Taylor series, we may readily show that the truncation error is bounded by

$$|f_i' - \frac{-f_{i+2} + 4f_{i+1} - 3f_i}{2\Delta x}| \leq \max_{s \in [x_i, x_{i+2}]} f'''(s)\Delta x^2.$$

The formula has a three-point stencil. As the name suggests, the formula is second-order accurate.

Second order backward difference. The *second-order backward difference formula* approximates f_i' based on the function values at x_i, x_{i-1}, and x_{i-2}. The formula is given by

$$f_i' \approx \frac{3f_i - 4f_{i-1} + f_{i-2}}{2\Delta x}.$$

The formula has a three-point stencil and is second-order accurate.

Fourth order central difference. The *fourth-order central difference formula* approximates f_i' based on the function values at x_{i-2}, x_{i-1}, x_{i+1}, and x_{i+2}. The formula is given by

$$f_i' \approx \frac{-f_{i+2} + 8f_{i+1} - 8f_{i-1} + f_{i-2}}{12\Delta x}.$$

The formula has a five-point stencil and is fourth-order accurate.

12.4 Second derivatives

We have so far considered the approximation of the first derivative. We may also approximate the second (or higher) derivative using finite difference formulas. For instance, the *second order central difference formula* approximates $f_i'' \equiv f''(x_i)$ by

$$f_i'' \approx \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}.$$

Note that the denominator scales with Δx^2 for the second derivative (as opposed to Δx for the first derivative); perhaps this is not too surprising if we consider the definition of the derivative and the limit.

We may analyze the truncation error using the same approach as those used to analyze the first derivative formulas. We express f_{i+1} and f_{i-1} using Taylor series expansions:

$$f_{i+1} = f_i + f_i'\Delta x + \frac{1}{2}f_i''\Delta x^2 + \frac{1}{6}f_i'''\Delta x^3 + \frac{1}{24}f^{(4)}(\xi_1)\Delta x^4$$
$$f_{i-1} = f_i - f_i'\Delta x + \frac{1}{2}f_i''\Delta x^2 - \frac{1}{6}f_i'''\Delta x^3 + \frac{1}{24}f^{(4)}(\xi_2)\Delta x^4$$

for some $\xi_1 \in [x_i, x_{i+1}]$ and $\xi_2 \in [x_{i-1}, x_i]$. We then note that

$$f_{i+1} - f_{i-1} = 2f_i + f_i'' \Delta x^2 + \frac{1}{24}[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)]\Delta x^4;$$

note in particular that the odd-order terms cancel due to the symmetry. We finally note

$$|f_i'' - \frac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}| = \frac{1}{24}[f^{(4)}(\xi_1) + f^{(4)}(\xi_2)]\Delta x^2 \leq \frac{1}{12} \max_{s \in [x_{i-1}, x_{i+1}]} |f^{(4)}(s)| \Delta x^2.$$

Note that the second-order central difference formula is *second-order accurate*.

12.5 Taylor tables

We can derive a finite difference formula of an arbitrary-order accuracy for an arbitrary-order derivative in a systematic manner using *Taylor tables*. To illustrate the idea, we consider the approximation of the second-order accurate approximation of the first derivative using the function values at f_{i-1}, f_i, and f_{i+1}. We hence consider an approximation of the form

$$f_i' \approx \frac{1}{\Delta x}(af_{i-1} + bf_i + cf_{i+1});$$

our goal is to find appropriate coefficients a, b, and c. We approach the problem as follows: we first rearrange our approximation problem as

$$f_i' \Delta x \approx af_{i-1} + bf_i + cf_{i+1};$$

we then consider the Taylor series expansions of f_{i-1}, f_i, and f_{i+1} about x_i; we finally match as many coefficients of f_i, f_i', f_i'', ... appearing in the left hand side and the right hand side. We may compactly record coefficients in a tabular form.

	f_i	$f_i' \Delta x$	$f_i'' \Delta x^2$	$f_i''' \Delta x^3$	$\cdots$	multiplier
f_{i-1}	1	-1	1/2	$-1/6$	$\cdots$	a
f_i	1	0	0	0	$\cdots$	b
f_{i+1}	1	1	1/2	1/6	$\cdots$	c
$f_i' \Delta x$	0	1	0	0	$\cdots$	

We then require that sum of each column yields the coefficient of the desired quantity (in our case the first derivative). This can be expressed as a system of equations

$$\begin{aligned}
1 \cdot a + 1 \cdot b + 1 \cdot c &= 0 \qquad \text{(1st column of Taylor table)} \\
-1 \cdot a + 0 \cdot b + 1 \cdot c &= 1 \qquad \text{(2nd column of Taylor table)} \\
\frac{1}{2} \cdot a + 0 \cdot b + \frac{1}{2} \cdot c &= 0 \qquad \text{(3rd column of Taylor table)}
\end{aligned}$$

or, equivalently, in a matrix form as

$$\begin{pmatrix} 1 & 1 & 1 \\ -1 & 0 & 1 \\ 1/2 & 0 & 1/2 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

The solution is $a = -1/2$, $b = 0$, and $c = 1/2$. The resulting approximation of f_i' is

$$f_i' \approx \frac{1}{\Delta x}(-\frac{1}{2}f_{i-1} + \frac{1}{2}f_{i+1}) = \frac{f_{i+1} - f_{i-1}}{2\Delta x}.$$

This of course is the central difference formula for the first derivative, which is second-order accurate.

We may also readily evaluate the truncation error using the Taylor table. Because we chose the coefficients a, b, and c to match the first three columns of the Taylor table, the leading truncation error associated with our finite difference formula is the fourth column. Specifically,

$$\begin{aligned}
\epsilon &= |f_i' - \frac{1}{\Delta x}(af_{i-1} + bf_i + cf_{i+1})| = \frac{1}{\Delta x}|f_i'\Delta x - [af_{i-1} + bf_i + cf_{i+1}]| \\
&= \frac{1}{\Delta x}|-\left(-\frac{1}{2}\right)\left(-\frac{1}{6}\right)f'''(\xi_1)\Delta x^3 + \left(\frac{1}{2}\right)\left(\frac{1}{6}\right)f'''(\xi_2)\Delta x^3| \\
&\leq \frac{1}{6}\max_{s\in[x_{i-1},x_{i+1}]}|f'''(s)|\Delta x^2;
\end{aligned}$$

note that this is consistent with our earlier analysis.

In general, we may construct the k-th order accurate approximation of m-th derivative using a $m + k$-point stencil. For instance, the second order approximation of the first derivative requires a three-point stencil (i.e. central difference). Hence we may construct a Taylor table with $m+k$ rows and $m + k$ columns (or $m + k + 1$ columns if we wish to analyze the truncation error), and identify the appropriate coefficients by solving a $(m + k) \times (m + k)$ linear system. However, the formula might achieve a higher order accuracy due to systematic cancellation of a higher order term, as we observed for the second-order central difference formula which yields the second order accurate approximation of second derivative using a three-point stencil.

Remark on accuracy and cost. In general, higher-order accurate formulas are desired because they provide faster convergence. However, higher-order accurate formulas also have wider stencils than lower-order formulas. Wider stencils arise because we must match more coefficients of Taylor series to achieve higher-order accuracy. Hence, we must balance the accuracy and cost in choosing an appropriate formula. In addition, we must consider the smoothness of the function f, because a higher-order accuracy cannot be achieved if the underlying function is irregular. (Note that our truncation error analysis based on Taylor series does not apply for irregular functions.) This consideration for accuracy, cost, and smoothness is similar to what we have already observed in the context of interpolation and integration.

12.6 Summary

We summarize the key points of this lecture:

1. A finite difference formula approximates the (first or higher) derivative of a function evaluated at x_i using a finite set of function evaluations near x_i.

2. The accuracy of a given finite difference formula is characterized by its order of accuracy, which can be analyzed using Taylor series. A finite difference formula is said to be p-th order accurate if the truncation error is bounded by $C\Delta x^p$.

3. The cost of a given finite difference formula is characterized by its stencil. A formula that involves n distinct points is said to have a n-point stencil.

4. If the function evaluation points are all greater than or equal to x_i, it is a forward difference formula. If the function evaluation points are all less than or equal to x_i, then it is a backward difference formula. If the stencil is symmetric, then it is a centered difference formula.

5. A finite difference formula of an arbitrary-order accuracy for an arbitrary-order derivative can be systematically constructed using a Taylor table.

Lecture 13

Initial value problems: Euler methods

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

13.1 Motivation

The objective of this lecture is twofold. The first objective is to introduce two methods to numerically approximate the solution of initial value problems (IVPs): the backward Euler method and the forward Euler method. The second objective is to introduce various concepts associated with numerical approximations of IVPs, which are used throughout our discussion of IVPs.

13.2 Model problem

Throughout this lecture (and the next lecture), we consider a model IVP of the form

$$\begin{aligned} \frac{du}{dt} &= f(u,t) \quad \text{for } 0 < t \leq t_f, \\ u(t=0) &= u_0, \end{aligned}$$

where $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the function that characterized the ODE, and $u_0 \in \mathbb{R}$ is some initial condition. We assume that f is continuous in the second arguments. In addition, we assume f is Lipschitz continuous in the first argument: there exists a constant L such that

$$|f(w,t) - f(v,t)| \leq L|w - v|, \quad \forall t \in (0, t_f],$$

where L is the Lipschitz constant. We recall that if f is continuously differentiable then

$$L = \max_{w,t} \left| \frac{\partial f(w,t)}{\partial w} \right|.$$

Under these assumptions, the IVP has a unique solution over some interval. (This follows from the Picard-Lindelöf theorem; we here omit technical discussions.)

13.3 Backward Euler method

In order to solve an IVP numerically, we first discretize the domain $(0, t_f]$ into J segments. The discrete time points are given by

$$t^j = j\Delta t, \quad j = 0, 1, \ldots, J \equiv t_f/\Delta t,$$

where $\Delta t = t_f/J$ is the time step, which we assume to be constant.

The *backward Euler method* is obtained by approximating the time derivative by the (first-order) backward difference formula:

$$\frac{d\tilde{u}}{dt}(t^j) \approx \frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t},$$

were $\tilde{u}^j = \tilde{u}(t^j)$ is the approximation to $u(t^j)$. The substitution of the backward difference formula to our model IVP yields

$$\begin{aligned} \frac{\tilde{u}^j - \tilde{u}^{j-1}}{\Delta t} &= f(\tilde{u}^j, t^j), \quad j = 1, \ldots, J, \\ \tilde{u}^0 &= u_0. \end{aligned}$$

or, equivalently,

$$\begin{aligned} \tilde{u}^j &= \tilde{u}^{j-1} + \Delta t f(u^j, t^j), \quad j = 1, \ldots, J, \\ \tilde{u}^0 &= u_0. \end{aligned}$$

This is an example of a *difference equation* (i.e. a recurrence relation) which arises from discretization of a *differential equation*.

We hope that our approximation $\tilde{u}^j$, $j = 1, \ldots, t_f/J$, converges to the true solution $u(t^j)$ as $\Delta t \to 0$ (or equivalently as $J \to \infty$). In order to ensure convergence, the difference equation must be consistent with respect to the IVP. We now analyze the consistency of the backward Euler method. (We will soon see that consistency is necessary, but not sufficient, condition for convergence.)

13.4 Consistency

Consistency ensures that the difference equation approximates the same process as the underlying differential equation in the limit of $\Delta t \to 0$. In order to formally define the notion of consistency, we first introduce the *local truncation error (LTE)*. In the context of numerical solution of IVPs, the truncation errors τ^j, $j = 1, \ldots, J$, are obtained by the substitution of the true solution u into the difference equation. For instance for the backward Euler method,

$$\tau^j \equiv u(t^j) - u(t^{j-1}) - \Delta t f(u(t^j), t^j), \quad j = 1, \ldots, J.$$

The truncation error is a measure of the extent to which the exact solution does not satisfy the difference equation. We are particularly interested in the largest truncation error,

$$\|\tau\|_\infty = \max_{j=1,\ldots,J} |\tau^j|.$$

A scheme is *consistent* with the ODE if the truncation error decays faster than the time step Δt: i.e.,

$$\frac{\|\tau\|_\infty}{\Delta t} \to 0 \quad \text{as} \quad \Delta t \to 0.$$

The difference equation for a consistent scheme approaches the differential equation as $\Delta t \to 0$.

Warning. The local truncation error is sometimes defined as $\hat{\tau}^j \equiv \tau^j/\Delta t$. If this definition of the local truncation error is used, then consistency requires that $\hat{\tau}^j \to 0$ as $\Delta t \to 0$. In this note we however use the convention for LTE τ^j (and not $\hat{\tau}^j$) defined above.

As an example, we analyze the truncation error of the backward Euler method and demonstrate the method is consistent. First, we recall by Taylor expansion

$$u(t^{j-1}) = u(t^j) - u'(t^j)\Delta t + \frac{1}{2}u''(\xi^j)\Delta t^2,$$

for some $\xi^j \in [t_{j-1}, t_j]$, $j = 1, \ldots, J$. We then note that

$$\begin{aligned}|\tau^j| &\equiv |u(t^j) - u(t^{j-1}) - \Delta t f(u(t^j), t^j)| = \left| \left(u'(t^j)\Delta t + \frac{1}{2}u''(\xi^j)\Delta t^2 \right) - \Delta t f(u(t^j), t^j) \right| \\ &= |\Delta t \underbrace{(u'(t^j) - \Delta t f(u(t^j), t^j))}_{=0 \text{ by ODE}} - \frac{1}{2}u''(\xi^j)\Delta t^2| = |-\frac{1}{2}u''(\xi^j)\Delta t^2| \le \frac{1}{2}\max_{s\in[t^{j-1},t^j]}|u''(s)|\Delta t^2.\end{aligned}$$

Hence the maximum truncation error is bounded by

$$\|\tau\|_\infty \le \frac{1}{2}\max_{s\in[0,t_f]}|u''(s)|\Delta t^2;$$

the local truncation error is second order accurate. We in addition note that

$$\lim_{\Delta t\to 0}\frac{\|\tau\|_\infty}{\Delta t} \le \lim_{\Delta t\to 0}\frac{1}{2}\max_{s\in[0,t_f]}|u''(s)|\Delta t = 0;$$

hence the backward Euler method is consistent.

13.5 Convergence

A scheme is *convergent* if the solution to the difference equation approaches the solution of the IVP as $\Delta t \to 0$ for all values of t. Formally, this means that

$$\tilde{u}^j \equiv \tilde{u}(t^j) \to u(t^j) \quad \text{for any fixed } t^j \text{ as } \Delta t \to 0.$$

Note that fixed time t^j means that the time index j must go to infinity as $\Delta t \to 0$ since $t^j = j\Delta t$.

We now prove that the backward Euler scheme is convergent. Towards this end, we first subtract the two equations

$$\begin{aligned}u(t^j) - u(t^{j-1}) - \Delta t f(u^j, t^j) &= \tau^j, \\ \tilde{u}^j - \tilde{u}^{j-1} - \Delta t f(\tilde{u}^j, t^j) &= 0,\end{aligned}$$

to obtain the equation for the error, $e^j \equiv u(t^j) - \tilde{u}^j$,

$$e^j - e^{j-1} - \Delta t[f(u^j, t^j) - f(\tilde{u}^j, t^j)] = \tau^j.$$

We now rearrange the equation as

$$e^j - \Delta t[f(u^j, t^j) - f(\tilde{u}^j, t^j)] = e^{j-1} + \tau^j.$$

We then take the absolute value of both sides and invoke triangle inequality:

$$|e^j| - \Delta t|f(u^j, t^j) - f(\tilde{u}^j, t^j)| \leq |e^{j-1}| + |\tau^j|.$$

We then appeal to the Lipschitz continuity of the function $f(\cdot, t^j)$ to obtain

$$|e^j| - \Delta t L|e^j| \leq |e^{j-1}| + |\tau^j|.$$

Assuming $\Delta t < 1/L$ such that $1 - \Delta t L > 0$, we obtain

$$\begin{aligned} |e^j| &\leq (1 - \Delta t L)^{-1}(|e^{j-1}| + |\tau^j|), \quad j = 1, \ldots, J, \\ e^0 &= 0. \end{aligned}$$

We observe the recurrence relationship

$$\begin{aligned} |e^0| &\leq 0, \\ |e^1| &\leq (1 - \Delta t L)^{-1}|\tau^1|, \\ |e^2| &\leq (1 - \Delta t L)^{-1}((1 - \Delta t L)^{-1}|\tau^1| + |\tau^2|), \\ &\vdots \\ |e^j| &\leq \sum_{k=1}^{j} (1 - \Delta t L)^{-(j-k+1)}|\tau^k|. \end{aligned}$$

We now note $\|\tau\|_\infty \equiv \max_{k=1}^J |\tau^j|$ to simply the bound to

$$|e^j| \leq \sum_{k=1}^{j} (1 - \Delta t L)^{-(j-k+1)}\|\tau\|_\infty = \sum_{k=1}^{j} (1 - \Delta t L)^{-k}\|\tau\|_\infty.$$

We now appeal to geometric series to evaluate the sum:

$$|e^j| \leq \frac{(1 - \Delta t L)^{-j} - 1}{\Delta t L}\|\tau\|_\infty.$$

We next appeal to $(1 - \Delta t L)^{-1} \leq 1 + 2\Delta t L \leq \exp(2\Delta t L)$ for $\Delta t L \leq 1/2$ to obtain

$$|e^j| \leq \frac{\exp(2j\Delta t L) - 1}{\Delta t L}\|\tau\|_\infty = \frac{\exp(2t^j L) - 1}{L}\frac{\|\tau\|_\infty}{\Delta t}.$$

We finally recall that $\|\tau\|_\infty/\Delta t \leq \frac{1}{2}\max_{s\in[0,t_f]} |u''(s)|\Delta t$ to obtain

$$|e^j| \leq \frac{\exp(2t^j L) - 1}{2L} \max_{s\in[0,t_f]} |u''(s)|\Delta t.$$

We make two observations:

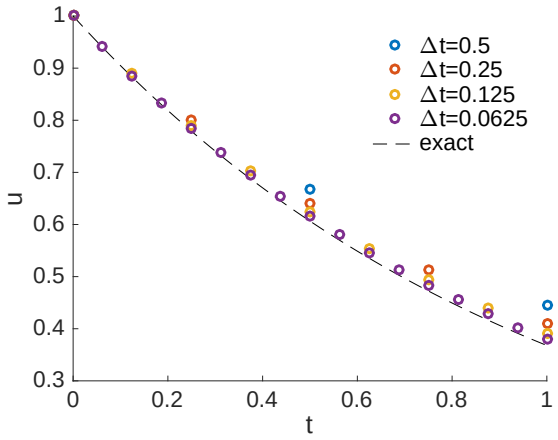


Figure 13.1: The backward Euler method applied to $\frac{du}{dt} = -u$ with the initial condition of $u_0 = 1$.

1. convergence: $|e^j| \to 0$ as $\Delta t \to 0$ for any t^j and hence the scheme is convergent;

2. order of accuracy: $|e^j| \leq C\Delta t$ and hence the scheme is first order accurate.

We see that the proof of convergence for even an arguably simple backward Euler scheme is still quite involved. Fortunately, there is a theorem — *Dahlquist equivalence theorem* — which shows that consistency and zero stability (which we discuss in the following lecture) are sufficient condition for any multistep scheme (which includes backward Euler) to be convergent. We will introduce this theorem in the context of more general multistep schemes in the next chapter.

Example. We solve the IVP,

$$\begin{aligned} \frac{du}{dt} &= -u \quad \text{for } t \in (0, 1], \\ u(t = 0) &= 1, \end{aligned}$$

using the backward Euler method and a few different time steps. The result of the computation is shown in Figure 13.1. We observe that the backward Euler approximation improves as the time step decreases. The error at the final time for the time steps of $\Delta t = 1/2$, $1/4$, $1/8$, and $1/16$ are $|u(t_f) - \tilde{u}(t_f)| = 0.077$, 0.042, 0.022, and 0.011, respectively. The error decreases by a factor of 2 when we decrease the time step by a factor of $2^1 = 2$; this is the expected behavior for the first-order accurate method.

13.6 Forward Euler method

We now introduce the *forward Euler scheme*. The forward Euler scheme is based on the (first-order) forward difference formula for the time derivative:

$$\frac{du}{dt}(t^j) \approx \frac{\tilde{u}^{j+1} - \tilde{u}^j}{\Delta t}.$$

The application of the scheme to our model equation yields the following difference equation:

$$\frac{\tilde{u}^{j+1} - \tilde{u}^j}{\Delta t} = f(\tilde{u}^j, t^j), \quad j = 0, \ldots, J-1,$$
$$\tilde{u}^0 = u_0.$$

We may shift the indices and rearrange the equations to obtain an equivalent difference equation:

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t f(\tilde{u}^{j-1}, t^{j-1}), \quad j = 1, \ldots, J,$$
$$\tilde{u}^0 = u_0.$$

Note that, unlike the backward Euler scheme, the right hand side of the forward Euler scheme does not contain a term with a time index j.

We may readily show that the forward Euler scheme is consistent. To see this we again substitute the exact solution to the difference equation and analyze the truncation error:

$$\begin{aligned}
|\tau^j| &\equiv |u(t^j) - u(t^{j-1}) - \Delta t f(u(t^{j-1}), t^{j-1})| \\
&= |u(t^{j-1}) + u'(t^{j-1})\Delta t + \frac{1}{2}u''(\xi^j)\Delta t^2 - u(t^{j-1}) - \Delta t u'(t^{j-1})| \\
&\leq \frac{1}{2} \max_{s \in [t^{j-1}, t^j]} |u''(s)| \Delta t^2.
\end{aligned}$$

It thus follows

$$\|\tau\|_\infty \leq \frac{1}{2} \max_{s \in [0, t_f]} |u''(s)| \Delta t^2.$$

Since $\|\tau\|_\infty / \Delta t \to 0$ as $\Delta t \to 0$, the scheme is consistent.

We can also show that the forward Euler method is convergent. More specifically, the error at time t^j is bounded by

$$|e^j| \leq \frac{\exp(t^j L) - 1}{2L} \max_{s \in [0, t_f]} |u''(s)| \Delta t.$$

As $\Delta t \to 0$, $|e^j| \to 0$ for any t^j; the forward Euler method is convergent. The expression also shows that the scheme is first order accurate.

Example. We solve an IVP, $\frac{du}{dt} = -u$ and $u_0 = 1$, using the forward Euler method and a few different time steps. The result of the computation is shown in Figure 13.2. As expected, the forward Euler approximation improves as the time step decreases. The error at the final time for the time steps of $\Delta t = 1/2$, $1/4$, $1/8$, and $1/16$ are $|u(t_f) - \tilde{u}(t_f)| = 0.118$, 0.052, 0.024, and 0.012, respectively. The error decreases by a factor of 2 when we decrease the time step by a factor of $2^1 = 2$; this is the expected behavior for the first-order accurate method. For a given time step, the error associated with the forward Euler method is comparable to that for the backward Euler method.

13.7 Absolute stability

We have seen that both backward Euler and forward Euler methods are convergent. Hence, as $\Delta t \to 0$, both methods produce the exact solution to the IVP: $\tilde{u}(t^j) \to u(t^j)$. However, the convergence does not inform us the behavior of the scheme for a finite Δt.

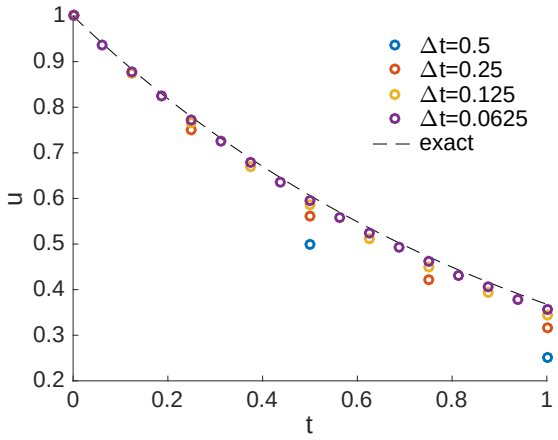


Figure 13.2: The forward Euler method applied to $\frac{du}{dt} = -u$ with the initial condition of $u_0 = 1$.

One of the important properties of a scheme that characterizes its finite-time-step behavior is the *absolute stability*. In order to study absolute stability, we consider a homogeneous IVP,

$$\frac{du}{dt} = \lambda u,$$
$$u(t = 0) = 1,$$

where $\lambda \in \mathbb{C}$. (We take $\lambda \in \mathbb{C}$ instead of in $\mathbb{R}$; we will see the importance of this generalization in the context of the solution of systems of equations.) Note that the solution to this IVP is

$$u(t) = \exp(\lambda t).$$

The solution grows with time for $\Re(\lambda) > 0$; the solution decays with time for $\Re(\lambda) < 0$; the solution is purely oscillatory if λ is purely imaginary.

A scheme is said to be *absolutely stable* for a given value of $\lambda\Delta t$ if the solution $\tilde{u}^j$, $j = 1, \ldots, J$, to the associated difference equation satisfies

$$|\tilde{u}^j| \leq |\tilde{u}^{j-1}|, \quad j = 1, \ldots, J.$$

Alternatively, we can introduce the *amplification factor*,

$$\gamma \equiv \frac{|\tilde{u}^j|}{|\tilde{u}^{j-1}|},$$

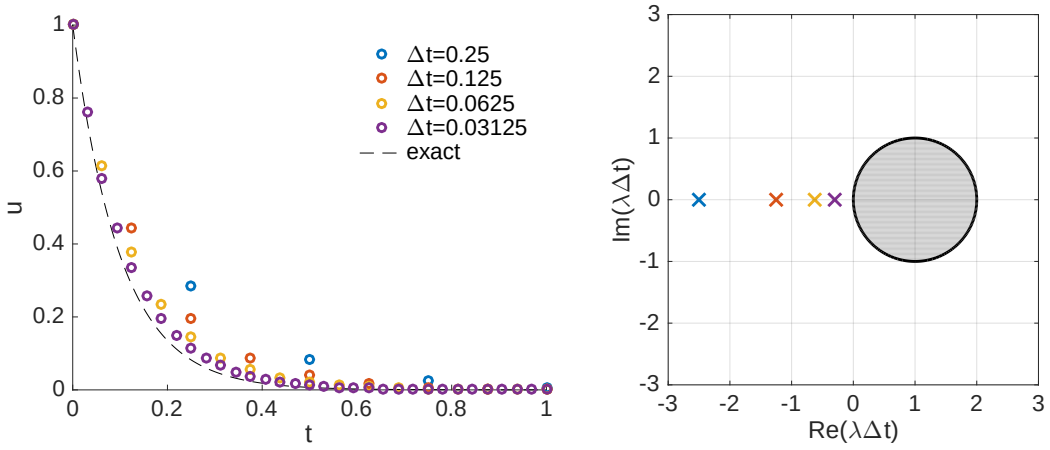
and define absolute stability as $\gamma \leq 1$ for all $j = 1, \ldots, J$.

We now analyze the absolute stability of the backward Euler method. Towards this end, we first obtain the difference equation for the homogeneous equation,

$$\tilde{u}^j = \tilde{u}^{j-1} + \lambda\Delta t\tilde{u}^j, \quad j = 1, \ldots, J,$$

and $u^{j=0} = 1$. The difference equation may be rearranged as

$$(1 - \lambda\Delta t)\tilde{u}^j = \tilde{u}^{j-1}.$$



(a) approximation (b) stability diagram

Figure 13.3: The backward Euler method applied to $\frac{du}{dt} = -10u$ with the initial condition of $u_0 = 1$. The scheme is absolutely stable in unshaded region; the scheme is unstable in the shaeded region. The marks on the absolute stability diagrams are associated with the values of $\lambda\Delta t$ for the time steps used in the approximation.

It follows that

$$|\tilde{u}^j| = \frac{1}{|1 - \lambda\Delta t|}|\tilde{u}^{j-1}|.$$

Hence the scheme is absolutely stable for any $\lambda\Delta t$ such that $|1 - \lambda\Delta t| \geq 1$. In particular, given $\Re(\lambda) \leq 0$ for which the exact solution decays with time, the scheme is absolutely stable for all Δt. For a given λ, a scheme that is stable for all Δt is said to be *unconditionally stable* (for that λ); the backward Euler scheme is unconditionally stable for all $\Re(\lambda) \leq 0$. A scheme that is unconditionally stable for all λ such that $\Re(\lambda) \leq 0$ is called *A-stable*; the backward Euler scheme is A-stable.

The absolute stability of a given scheme is often summarized in a *absolute stability diagram*. The absolute stability diagram for the backward Euler scheme is shown in Figure 13.3(b); the scheme is stable in the unshaded region and is unstable in the shaded region. We here also superimpose to the diagram $\lambda\Delta t(= -10\Delta t)$ associated with several different time steps by `x`; note that all `x` falls in the stable region of the stability diagram. As a result, all approximations shown in Figure 13.3(a) exhibit a stable behavior.

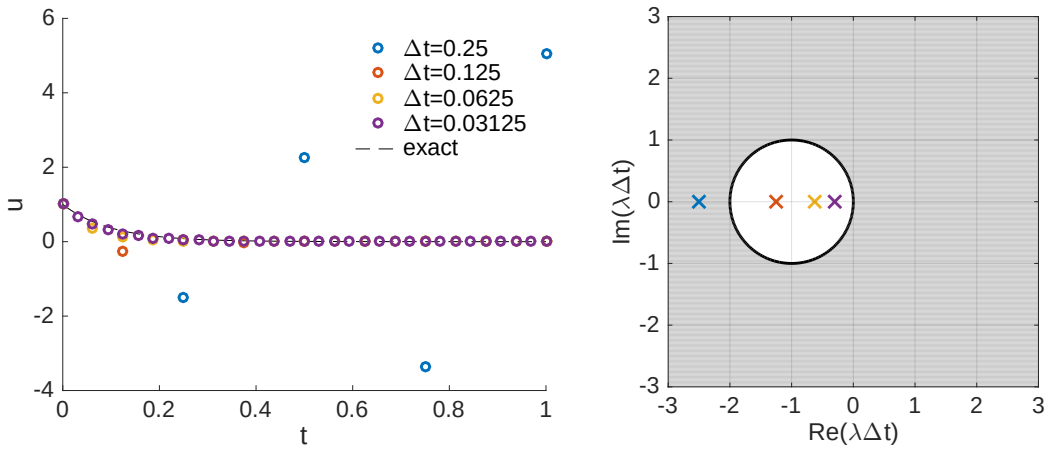
We can also analyze the absolute stability of the forward Euler method. We again first obtain the difference equation for the homogeneous equation,

$$\tilde{u}^j = \tilde{u}^{j-1} + \lambda\Delta t\tilde{u}^{j-1}, \quad j = 1, \ldots, J,$$

and $u^{j=0} = 1$. Taking the absolute value of both sides, this time we obtain

$$|\tilde{u}^j| = |1 + \lambda\Delta t||\tilde{u}^j|.$$

We observe that the scheme is absolutely stable for any $\lambda\Delta t$ such that $|1 + \lambda\Delta t| \leq 1$. Note that, for a real $\lambda < 0$, the scheme is absolutely stable for $\Delta t \leq t_{\text{cr}} = 2/|\lambda|$. Because of this time-step



(a) approximation

(b) stability diagram

Figure 13.4: The forward Euler method applied to $\frac{du}{dt} = -10u$ with the initial condition of $u_0 = 1$. The scheme is absolutely stable in unshaded region; the scheme is unstable in the shaeded region. The marks on the absolute stability diagrams are associated with the values of $\lambda\Delta t$ for the time steps used in the approximation.

restriction, the scheme is said to be *conditionally stable* (for $\lambda < 0$). Because the scheme is not unconditionally stable for all λ such that $\Re(\lambda) \leq 0$, the forward difference scheme is not A-stable.

Figure 13.4(b) shows the absolute stability diagram for the forward Euler method. We here also superimpose the $\lambda\Delta t(= -10\Delta t)$ associated with several different time steps. Note that the forward Euler method is stable only for $\lambda\Delta t$ such that $|1 + \lambda\Delta t| < 1$. Specifically, for the time steps considered, the scheme is unstable for $\Delta t = 0.25$ since $|1 + \lambda\Delta t| = |1 - 10 \cdot 0.25| = 1.5 > 1$. In the unstable case, we see in Figure 13.4(a) that the approximation *grows* exponentially in time, even though the exact solution *decays* exponentially in time. Consequently, with a conditionally stable scheme, we cannot take a time step greater than Δt_{cr} even when we do not need to find a very well-resolved solution; the largest Δt we can take is not governed by the accuracy requirement but rather by the stability consideration.

13.8 Implementation of the forward and backward Euler methods

We recall that the difference equation (or the recurrence relation) for the forward Euler method is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t f(\tilde{u}^{j-1}, t^{j-1}), \quad j = 1, \ldots, J.$$

Given the state $\tilde{u}^{j-1}$, we simply evaluate the right hand side of the equation to compute $\tilde{u}^j$. This simple update is possible because the right hand side of the difference equation does not contain the unknown $\tilde{u}^j$; a scheme which does not have $\tilde{u}^j$ in the right hand side — or equivalently a scheme which does not require the evaluation of $f(\cdot, \cdot)$ at $\tilde{u}^j$ and t^j — is called an *explicit* method. The explicit methods are very easy to advance in time.

We also recall that the difference equation for the backward Euler method is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t f(\tilde{u}^j, t^j), \quad j = 1, \ldots, J.$$

Note that, unlike in the forward Euler method, the backward Euler method requires the evaluation of the function f at $\tilde{u}^j$, which in fact is the unknown we with to evaluate. A scheme which requires the evaluation of $f(\tilde{u}^j, t^j)$ is called an *implicit* method. When a scheme is implicit, we cannot simply evaluate the right hand side of the equation to advance in time.

For an implicit scheme, in order to compute the state $\tilde{u}^j$ given $\tilde{u}^{j-1}$, we must solve a nonlinear equation

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t f(\tilde{u}^j, t^j),$$

where the unknown is $\tilde{u}^j$. More explicitly, we may express the residual of the nonlinear equation as

$$r(\tilde{u}^j) \equiv \tilde{u}^j - u^{j-1} - \Delta t f(\tilde{u}^j, t^j),$$

and find the root of the function $r : \mathbb{R} \to \mathbb{R}$. We may for example use Newton's method to solve the nonlinear equation. Note that the Jacobian of the residual equation is

$$\left.\frac{dr}{du}\right|_{\tilde{u}^j} = 1 - \Delta t \left.\frac{\partial f}{\partial u}\right|_{(\tilde{u}^j, t^j)}.$$

Hence, an efficient implementation of an implicit method often requires not only f, which defines the IVP, but also the derivative of f with respect to the state u.

In general, a single step of implicit method is computationally more expensive than a single step of an explicit method; the former requires a solution to the nonlinear equation involving f, whereas the latter only requires a simple evaluation of f. However, in general, implicit methods exhibit larger regions of absolute stability than explicit methods. Hence, implicit methods often allow us to take a larger time step without a stability limitation. The favorable stability characteristics of implicit methods play a crucial for time integration of stiff equations as we will see in a later lecture.

13.9 Summary

We summarize the key points of this lecture:

1. Discretization of an initial value problem (IVP) governed by an ODE yields a difference equation.

2. Local truncation error (LTE) of a given scheme at a given time step j, denoted by τ^j, is the remainder that results from the substitution of the exact solution to the difference equation. The order of LTE can be analyzed using Taylor series.

3. A scheme is said to be consistent if $\|\tau\|_\infty / \Delta t \to 0$ as $\Delta t \to 0$.

4. A scheme is said to be convergent if $\tilde{u}^j \to u(t^j)$ for any fixed t^j as $\Delta t \to 0$. Convergence analysis is complicated because the error at time step j depends on not only the truncation error τ^j but also all previous errors.

5. A scheme is said to be absolutely stable for a given $\lambda \Delta t \in \mathbb{C}$ if the solution amplitude decays with time for an IVP, $du/dt = \lambda u$ and $u(t = 0) = 1$. Absolute stability characterizes the stability of the scheme for a finite Δt.

6. A scheme is said to be A-stable if it is absolutely stable for all $\lambda \Delta t$ in the left-hand plane.

7. A scheme is said to be conditionally stable (for $\lambda < 0$) if it is absolutely stable for only a finite range of Δt. A scheme that is absolutely stable for any Δt is said to be unconditionally stable.

8. Implicit schemes require the evaluation of $f(\cdot, \cdot)$ at $\tilde{u}^j$; explicit schemes require evaluation of $f(\cdot, \cdot)$ at only previous time steps (for which the approximation is already known). For a general nonlinear f, implicit schemes require the solution of a nonlinear equation at each time step.

9. The backward Euler method is first-order accurate, implicit, and A-stable.

10. The forward Euler method is first-order accurate, explicit, and conditionally stable for any $\Re(\lambda) < 0$.

Lecture 14

Initial value problems: multistep schemes

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

14.1 Multistep schemes

A K-step linear multistep method for the initial value problem, $\frac{du}{dt} = f(u,t)$, is defined by the difference equation

$$\sum_{k=0}^{K} \alpha_k \tilde{u}^{j-k} = \Delta t \sum_{k=0}^{K} \beta_k f(\tilde{u}^{j-k}, t^{j-k}).$$

A particular set of coefficients α_k, $k = 0,\dots,K$, and β_k, $k = 0,\dots,K$, determines a unique multistep scheme. It is customary to set $\alpha_0 = 1$ to fix the scaling of the coefficients. A mutlistep scheme is said to be *implicit* when $\beta_0 \neq 0$, as the term involving $\tilde{u}^j$ appears on the right hand side; a scheme is said to be *explicit* when $\beta_0 = 0$.

14.2 Local truncation error and consistency

We recall that the local truncation error (LTE) is the residual we obtain upon the substitution of the exact solution to the difference equation. For multistep schemes, the local truncation error is given by

$$\tau^j \equiv \sum_{k=0}^{K} \alpha_k u(t^{j-k}) - \Delta t \sum_{k=0}^{K} \beta_k f(u(t^{j-k}), t^{j-k}).$$

We recall that a scheme is consistent if

$$\lim_{\Delta t \to 0} \frac{\|\tau\|_\infty}{\Delta t} = 0,$$

where $\|\tau\|_\infty \equiv \max_j |\tau^j|$.

14.3 Zero-stability

For the backward and forward Euler methods, we proved the convergence through a careful direct analysis. The analysis was quite involved even for the Euler methods, which are arguably the simplest multistep methods. In this lecture we introduce an alternative way to prove convergence that relies on the *Dahlquist equivalence theorem*. The theorem requires two ingredients: the first is the consistency, which we have already introduced; the second is the zero stability, which we introduce now.

The zero stability associated with the case of $\lambda\Delta t = 0$. A multistep scheme is said to be zero stable if the solution to the difference equation,

$$\sum_{k=0}^{K} \alpha_k \tilde{u}^{j-k} = 0, \quad j = 1, \ldots, J,$$

is bounded for all possible initial conditions. We may exploit the structure of the difference equation to obtain an equivalent condition known as the *root condition*. Specifically, we consider the roots of a degree-K polynomial,

$$p(x) \equiv \sum_{k=0}^{K} \alpha_k x^{K-k}.$$

Let us denotes the K roots of the polynomial p by z_k, $k = 1, \ldots, K$. We then consider two cases:

1. if all roots are distinct, then the multistep scheme is zero stable if the magnitude of each root is less than or equal to unity (i.e. $|z_k| \leq 1$, $\forall k$);

2. if there is a repeated root, then the multistep scheme is zero stable if the magnitude of each repeated root is (strictly) less than unity (i.e. $|z_k| < 1$, $\forall k$).

We provide a sketch which shows the equivalence of the definition of zero stability and the root condition for the case with distinct roots. First, suppose $\tilde{u}^j = \zeta^j$ for some number ζ, where the superscript j on ζ^j indicates the power (unlike the superscript on $\tilde{u}^j$, which is the time index). We plug the solution into the difference equation for $\Delta t = 0$ to obtain

$$\sum_{k=0}^{K} \alpha_k \zeta^{j-k} = 0.$$

We divide through by ζ^{j-K} to obtain

$$\sum_{k=0}^{K} \alpha_k \zeta^{K-k} = 0.$$

It follows that ζ must be a root of the polynomial associated with the root condition, $p(x) = \sum_{k=0}^{K} \alpha_k x^{K-k}$. Let us denote the roots by $\{z_k\}_{k=1}^{K}$. Then, $\tilde{u}^j = z_k^j$ (again the superscript on z_k^j is the power) satisfies the difference equation. Because any linear combination of z_k^j is also a solution to the difference equation, it follows that

$$\tilde{u}^j = c_1 z_1^j + c_2 z_2^j + \cdots + c_K z_K^j$$

for some constants $\{c_k\}_{k=1}^K$. (The constants $\{c_k\}_{k=1}^K$ are determined by the initial conditions.) In order for the solution $\tilde{u}^j$ to be bounded for any initial condition as $j \to \infty$, each of the functions z_k^j, $k = 1, \ldots, K$, must be bounded as $j \to \infty$. This requires that the magnitude of the roots to be less than or equal to one: $|z_k| \leq 1$, $k = 1, \ldots, K$. This is precisely the root condition for non-repeated roots.

14.4 Convergence: Dahlquist equivalence theorem

The relationship between consistency, zero stability, and convergence for multistep schemes is summarized in the *Dahlquist equivalence theorem*. The theorem states that consistency and stability are the necessary and sufficient condition for convergence: i.e.,

$$\text{consistency} \;+\; \text{zero stability} \;\Leftrightarrow\; \text{convergence} \;.$$

Thus, we need to show, and only need to show, that a scheme is consistent and zero stable to show that the scheme is convergent. (The proof of the theorem is quite involved; we here omit the proof.)

14.5 Order of accuracy

The Dahlquist equivalence theorem only shows whether a scheme converges as $\Delta t \to 0$; however, the theorem does not show how quickly the scheme converges to the true solution as Δt is reduced. We recall that a scheme is said to be q-th order accurate if

$$|u(t^j) - \tilde{u}(t^j)| \leq C\Delta t^q \quad \text{for any fixed } t^j = j\Delta t \text{ as } \Delta t \to 0.$$

In general, for a zero stable multistep scheme, if the truncation error is $q+1$-th order acccurate, then the scheme is q-th order accurate: i.e.,

$$\|\tau\|_\infty \leq C\Delta t^{q+1} \quad \Rightarrow \quad |u(t^j) - \tilde{u}(t^j)| \leq C\Delta t^q \quad \text{for a fixed } t^j = j\Delta t \text{ and } \Delta t \text{ sufficiently small.}$$

In other words, once we show the stability of a scheme, we only need to analyze the truncation error to understand its convergence rate.

14.6 Absolute stability

In order to analyze the absolute stability, we again consider a homogeneous ODE of the form $\frac{du}{dt} = \lambda u$. In particular, a multistep scheme is said to be absolutely stable for a given $\lambda\Delta t$ if the solution to the difference equation,

$$\sum_{k=0}^K \alpha_k \tilde{u}^{j-k} = \Delta t \sum_{k=0}^K \beta_k \lambda \tilde{u}^{j-k},$$

is bounded for all possible initial conditions. We may again exploit the root condition to analyze the stability of the scheme. Towards this end we consider a degree-K polynomial

$$\sum_{k=0}^K (\alpha_k - \lambda\Delta t\beta_k) x^{K-k} = 0.$$

Let z_k, $k = 1, \ldots, K$ be the roots of the polynomial. We then again consider two different cases:

1. if all roots are distinct, then the multistep scheme is absolutely stable if the magnitude of each root is less than or equal to unity (i.e. $|z_k| \leq 1$, $\forall k$);

2. if there is a repeated root, then the multistep scheme is absolutely stable if the magnitude of each repeated root is (strictly) less than unity (i.e. $|z_k| < 1$, $\forall k$).

14.7 Example: two-step Adams-Moulton

The two-step Adams-Moulton scheme (AM2) is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \left(\frac{5}{12} f(\tilde{u}^j, t^j) + \frac{2}{3} f(\tilde{u}^{j-1}, t^{j-1}) - \frac{1}{12} f(\tilde{u}^{j-2}, t^{j-2}) \right)$$

We note that AM2 is an implicit scheme, as the term $f(\tilde{u}^j, t^j)$ is on the right hand side.

Consistency. The local truncation error of the AM2 is

$$\tau^j \equiv u(t^j) - u(t^{j-1}) - \Delta t \left(\frac{5}{12} f(u(t^j), t^j) + \frac{2}{3} f(u(t^{j-1}), t^{j-1}) - \frac{1}{12} f(u(t^{j-2}), t^{j-2}) \right).$$

To simplify the expression, we consider the Taylor series expansion of each term:

$$\begin{aligned}
u(t^{j-1}) &= u(t^j) - u'(t^j)\Delta t + \frac{1}{2}u''(t^j)\Delta t^2 - \frac{1}{6}u'''(t^j)\Delta t^3 + \frac{1}{24}u^{(4)}(\xi^j)\Delta t^4, \\
f(u(t^j), t^j) &= u'(t^j), \\
f(u(t^{j-1}), t^{j-1}) &= u'(t^{j-1}) = u'(t^j) - u''(t^j)\Delta t + \frac{1}{2}u'''(t^j)\Delta t^2 - \frac{1}{6}u^{(4)}(\eta^j)\Delta t^3, \\
f(u(t^{j-2}), t^{j-2}) &= u'(t^{j-2}) = u'(t^j) - 2u''(t^j)\Delta t + 2u'''(t^j)\Delta t^2 - \frac{4}{3}u^{(4)}(\zeta^j)\Delta t^3.
\end{aligned}$$

The substitution of the expansions to the expression for the truncation error yields

$$\tau^j = -\frac{1}{24}u^{(4)}(\xi^j)\Delta t^4.$$

We note that the scheme is consistent because

$$\frac{\|\tau\|_\infty}{\Delta t} = \frac{1}{24} \max_{s \in [0, t_f]} |u^{(4)}(s)| \Delta t^3 \to 0 \quad \text{as} \quad \Delta t \to 0.$$

Zero stability. To analyze the zero stability using the root condition, we consider the roots of

$$\sum_{k=0}^{2} \alpha_k x^{2-k} = x^2 - x = x(x-1).$$

The two roots are given by $x = 0$ and $x = 1$. The two roots are distinct and each has a magnitude less than or equal to unity. Hence, the two-step Adams-Moulton scheme is zero stable.

Convergence. By Dahlquist's equivalence theorem, the AM2 scheme is convergent because it is consistent and zero stable. In addition, because the local truncation error is fourth-order accurate, the two-step Adams-Moulton scheme is third-order accurate.

Numerical example. We approximate the solution of the IVP,

$$\frac{du}{dt} = (\cos(10t) - u) - \sin(10t), \quad t \in (0,1], u(t=0) = 1,$$

using the two-step Adams-Moulton method. We will also solve the problem using the backward Euler method for the purpose of comparison. Figures 14.1(a) and 14.1(b) shows the results of approximating the IVP using the backward Euler and two-step Adams-Moulton methods, respectively. We observe that qualitatively the AM2 method produces more accurate approximation than the backward Euler method for a given time step. More quantitatively, Figure 14.1(c) shows the error at the final time as a function of the time step Δt; we observe that the error associated with the AM2 method converges rapidly — as $\mathcal{O}(\Delta t^3)$. We do note however that, despite being an implicit method, the AM2 scheme is not A-stable as shown in Figure 14.1(d); the region of absolute stability is nevertheless quite large.

14.8 Example: a consistent but unstable scheme

Consider a scheme given by

$$\tilde{u}^j - 3\tilde{u}^{j-1} + 2\tilde{u}^{j-2} = -\Delta t f(\tilde{u}^{j-2}, t^{j-2}).$$

We will see that this scheme is consistent but not convergent.

Consistency. An analysis of the truncation error reveals that

$$\begin{aligned}
|\tau^j| &= |\tilde{u}^j - 3\tilde{u}^{j-1} + 2\tilde{u}^{j-2} + \Delta t f(\tilde{u}^{j-2}, t^{j-2})| \\
&= |[u(t^{j-2}) + 2u'(t^{j-2})\Delta t + 2u''(\xi_1)\Delta t^2] - 3[u(t^{j-2}) + u'(t^{j-2})\Delta t + \frac{1}{2}u''(\xi_2)\Delta t^2] \\
&\quad + 2u(t^{j-2}) + \Delta t u'(t^{j-2})| \\
&= |2u''(\xi_1)\Delta t^2 - \frac{3}{2}u''(\xi_2)\Delta t^2| \\
&\leq \frac{7}{2} \max_{s \in [t^{j-2}, t^j]} |u''(s)| \Delta t^2.
\end{aligned}$$

Since $\|\tau\|_\infty / \Delta t \to 0$ as $\Delta t \to 0$, the scheme is consistent.

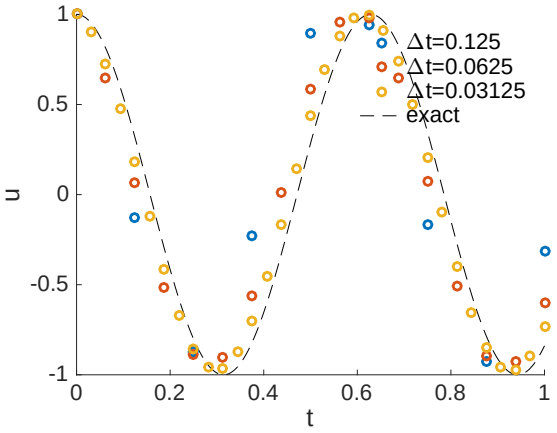
Zero stability. We now analyze the zero stability of the scheme. We check for the root condition:

$$\sum_{k=0}^{2} \alpha_k x^{2-k} = x^2 - 3x + 2 = (x-2)(x-1).$$

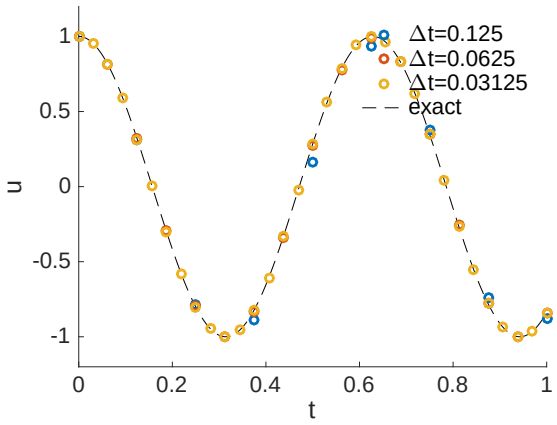
The roots are given by $x = 2$ and $x = 1$. Because the magnitude of one of the roots is greater than 1, this scheme is not zero stable.

Convergence. Because the scheme is not zero stable, the scheme is not convergent (even thought it is consistent).

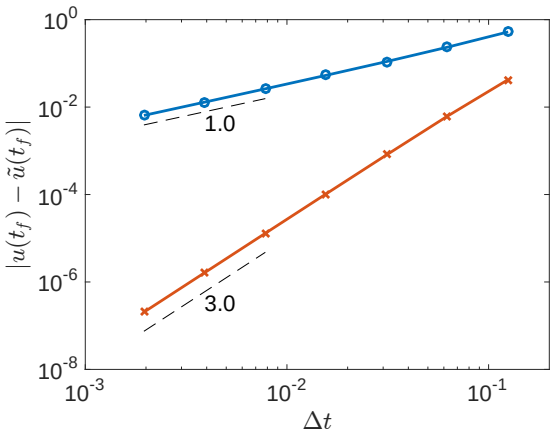
Numerical example. We (attempt to) approximate the solution of the IVP, $\frac{du}{dt} = -u$, $t \in (0,1]$, and $u(t=0) = 1$ using the consistent-but-unstable scheme. Figure 14.2 shows the result of the computation. As expected the approximation grows exponentially in time even though the exact solution decays exponentially in time. In fact the error increases (exponentially) as the time step decreases: the error at the final time for $\Delta t = 1/8$, $1/16$, $1/32$, and $1/64$ are $|u(t_f) - \tilde{u}(t_f)| = 4.3 \times 10^{-1}$, 1.5×10^2, 2.9×10^6, and 3.4×10^{15}, respectively.



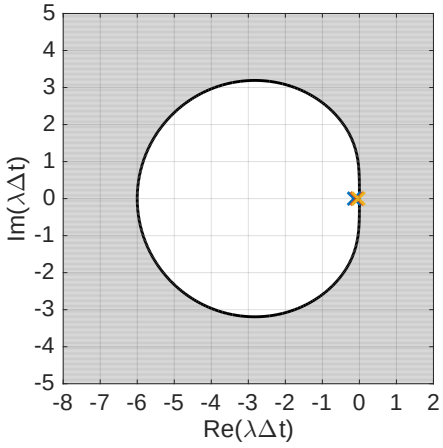
(a) backward Euler



(b) two-step Adams-Moulton



(c) final-time error convergence plot



(d) absolute stability diagram

Figure 14.1: Solution of $\frac{du}{dt} = (\cos(10t) - u) - \sin(10t)$, $t \in (0, 1]$, and $u_0 = 1$ using the backward Euler and two-step Adams-Moulton methods.

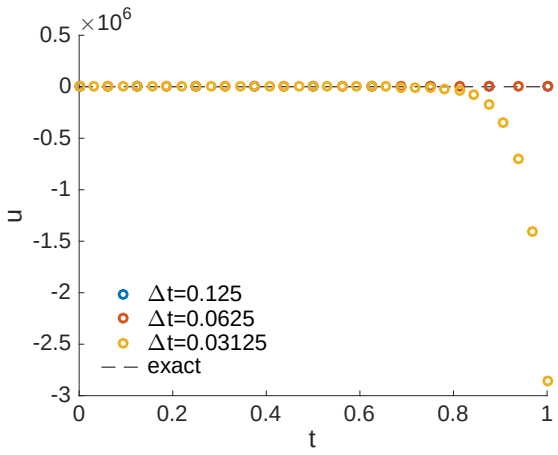


Figure 14.2: Approximation of $\frac{du}{dt} = -u$, $t \in (0, 1]$, and $u_0 = 1$ using the consistent but unstable scheme.

14.9 Construction: order matching conditions

We can construct multistep schemes of arbitrary order accuracy in a systematic manner. For instance, if we wish to construct a scheme of order M, we require that the scheme exactly solves IVPs whose solutions are polynomials of degree up to and including M:

$$\begin{aligned} \frac{d}{dt}1 &= 0, \\ \frac{d}{dt}t^m &= mt^{m-1}, \quad m = 1, \ldots, M. \end{aligned}$$

Without loss of generality, we chose $\Delta t = 1$ and write the system of equations associated with the IVPs:

$$\begin{aligned} \sum_{k=0}^{K} \alpha_k &= 0, \\ \sum_{k=0}^{K} \alpha_k (K-k+1)^m + \sum_{k=0}^{K} \beta_k m (K-k+1)^{m-1} &= 0, \quad m = 1, \ldots, M. \end{aligned}$$

For instance, the system of equations associated with $K = 2$ and $M = 4$ is

$$\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 2 & 1 & -1 & -1 & -1 \\ 9 & 4 & 1 & -6 & -4 & -2 \\ 27 & 8 & 1 & -27 & -12 & -3 \\ 81 & 16 & 1 & -108 & -32 & -4 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

We recall that the coefficients of a given multistep scheme is unique only up to scaling. We hence fix $\alpha_0 = 1$ and obtain

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 2 & 1 & -1 & -1 & -1 \\ 4 & 1 & -6 & -4 & -2 \\ 8 & 1 & -27 & -12 & -3 \\ 16 & 1 & -108 & -32 & -4 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = - \begin{pmatrix} 1 \\ 3 \\ 9 \\ 27 \\ 81 \end{pmatrix}.$$

The solution to this linear system would yield a two-step scheme whose local truncation error is of order $\mathcal{O}(\Delta t^{M+1}) = \mathcal{O}(\Delta t^5)$. *If* the scheme is also zero stable, then it would be an fourth-order scheme ($M = 4$).

The two-step Adam-Moultons scheme (which we have considered before) results from requiring $\alpha_2 = 0$ and setting $M = 3$. The associated linear system is

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & -1 & -1 & -1 \\ 4 & -6 & -4 & -2 \\ 8 & -27 & -12 & -3 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix} = - \begin{pmatrix} 1 \\ 3 \\ 9 \\ 27 \end{pmatrix}.$$

The solution to this linear system — $\alpha_1 = -1$, $\beta_0 = 5/12$, $\beta_1 = 2/3$, and $\beta_2 = -1/12$ — is the two-step Adams-Moulton scheme, which is third-order accurate (i.e. $M = 3$). The scheme exactly solves IVPs whose solutions are polynomials of degree up to and including $M = 3$.

Of course, the formulation only ensures that the method is consistent. In order to ensure convergence, we need to also ensure that the scheme is zero stable. Zero stability can be assessed by using the aforementioned root test.

14.10 Popular schemes

Forward and backward Euler. The backward and forward Euler schemes that we considered in the previous lecture is an example of multi-step — in fact single-step — schemes. The backward Euler scheme results from

$$\alpha_0 = 1, \quad \alpha_1 = -1, \quad \beta_0 = 1, \quad \text{and} \quad \beta_1 = 0.$$

The forward Euler scheme results from

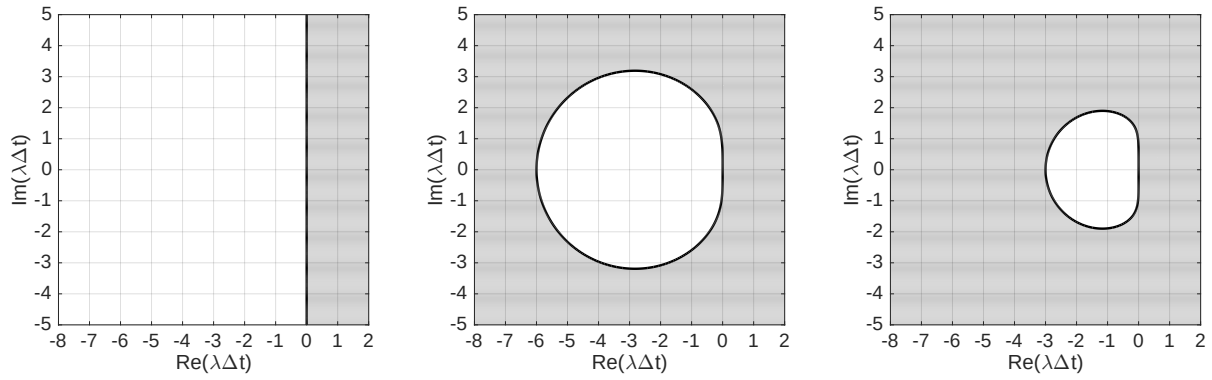
$$\alpha_0 = 1, \quad \alpha_1 = -1, \quad \beta_0 = 0, \quad \text{and} \quad \beta_1 = 1.$$

We recall both the backward and forward Euler schemes are first order accurate. We also recall that the schemes exhibit very different stability properties.

Crank-Nicolson. The Crank-Nicolson scheme is another popular implicit multistep method given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \frac{\Delta t}{2}(f(\tilde{u}^j, t^j) + f(\tilde{u}^{j-1}, t^{j-1})).$$

The stability diagram of the Crank-Nicolson scheme is shown in Figure 14.3(a); because the scheme is stable for the entire left-hand plane, the scheme is A-stable. The Crank-Nicolson scheme is second order accurate.



(a) Crank-Nicolson (AM1) (b) AM2 (c) AM3

Figure 14.3: Stability diagrams for Adams-Moulton schemes.

K	α_0	α_1	β_0	β_1	β_2	β_3
1	1	-1	1/2	1/2	-	-
2	1	-1	5/12	2/3	-1/12	-
3	1	-1	3/8	19/24	-5/24	1/24

Table 14.1: Coefficients for Adams-Moulton schemes.

Adams-Moulton. Adams-Moulton schems are implicit multistep schemes with the condition

$$\alpha_0 = 1, \qquad \alpha_1 = -1 \qquad \text{and} \qquad \alpha_k = 0 \text{ for } k = 2, \ldots, K. \tag{14.1}$$

The K-step Adams-Moulton scheme hence takes the form

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=0}^{K} \beta_k f(\tilde{u}^{j-k}, t^{j-k}).$$

The coefficients β_k, $k = 0, \ldots, K$, are chosen to achieve the highest order of accuracy. The Crank-Nicolson scheme is the $K = 1$ step Adams-Moulton scheme. Table 14.1 provides the multistep coefficients for the 1-, 2-, and 3-step Adams-Moulton schemes. The stability diagrams for the 1-, 2-, and 3-step Adams-Moulton schemes are shown in Figure 14.3; due to the implicit formulation, Adams-Moulton schemes provide relatively large stability regions. The K-step Adams-Moulton scheme is $K + 1$-st order accurate.

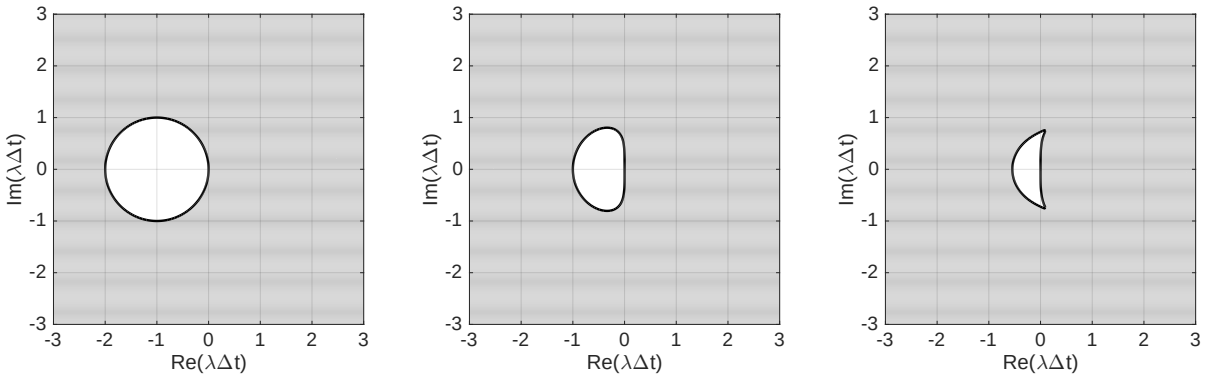
Adams-Bashforth. Adams-Bashforth schemes are explicit (i.e. $\beta_0 = 0$) multistep schemes with the same condition on the coefficient α_k as the Adams-Moulton schemes (i.e. equation (14.1)). The Adams-Bashforth family of schemes hence takes the form

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=1}^{K} \beta_k f(\tilde{u}^{j-k}, t^{j-k});$$

note that $\beta_0 = 0$ (and we hence sum from $k = 1$) since the scheme is explicit. The coefficients β_k, $k = 1, \ldots, K$, are chosen to achieve the highest order of accuracy. Table 14.2 provides the multistep

K	α_0	α_1	β_1	β_2	β_3
1	1	-1	1	-	-
2	1	-1	3/2	-1/2	-
3	1	-1	23/12	-4/3	5/12

Table 14.2: Coefficients for Adams-Bashforth schemes.



(a) forward Euler (AB1) (b) AB2 (c) AB3

Figure 14.4: Stability diagrams for Adams-Bashforth schemes.

coefficients for the 1-, 2-, and 3-step Adams-Bashforth schemes. The stability diagrams for 1-, 2-, and 3-step Adams-Bashforth schemes are shown in Figure 14.4; note that the stability regions of Adams-Bashforth schemes are considerably smaller than those of Adams-Moulton schemes. The forward Euler method is the $K = 1$ step Adams-Bashforth scheme. The K-step Adams-Bashforth scheme is K-th order accurate.

Backward differentiation formulas. Backward differentiation formulas (BDFs) are implicit schemes with the condition

$$\beta_k = 0, \quad k = 1, \ldots, K.$$

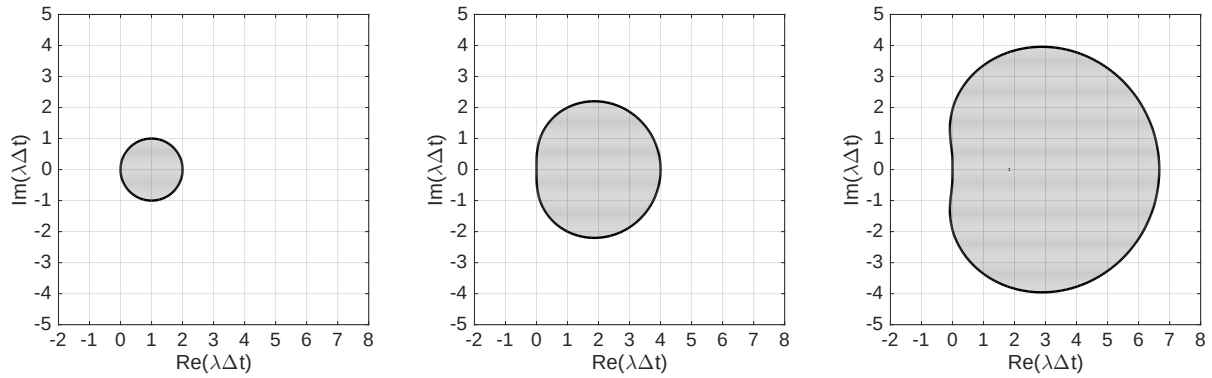
The backward differentiation formulas hence take the form

$$\sum_{k=0}^{K} \alpha_k \tilde{u}^{j-k} = \Delta t \beta_0 f(\tilde{u}^j, t^j).$$

The coefficients α_k, $k = 1, \ldots, K$, are chosen to achieve the highest order of accuracy. (Recall that we fix $\alpha_0 = 1$.) The backward Euler scheme is the backward differentiation formula for $K = 1$, which is often denoted BDF1. Table 14.3 provides the multistep coefficients for the 1-, 2-, and 3-step backward differentiation formulas. The backward differentiation formulas are well-known for their stability properties. The stability diagrams for BDF2 and BDF3 are shown in Figure 14.5. BDF1 (i.e. backward Euler) and BDF2 are A-stable — stable for all $\lambda \in \mathbb{C}$ such that $\Re(\lambda) < 0$. BDF3 is not quite A-stable; however, it nevertheless is stable for a large portion of the negative real half-plane. The K-step backward differentiation formula is K-th order accurate.

K	α_0	α_1	α_2	α_3	β_0
1	1	-1	-	-	1
2	1	-4/3	1/3	-	2/3
3	1	-18/11	9/11	-2/11	6/11

Table 14.3: Coefficients for backward differentiation formulas.



(a) backward Euler (BDF1) (b) BDF2 (c) BDF3

Figure 14.5: Stability diagrams for backward differentiation formulas.

14.11 Summary

We summarize the key points of this lecture:

1. A K-step multistep scheme approximates the state $\tilde{u}^j$ based on the state and the state derivative from K previous steps.

2. The local truncation error of a multistep scheme is the residual we obtain when the exact solution is substituted to the difference equation. The order of the truncation can be analyzed using Taylor series.

3. The zero-stability of a multistep scheme is determined by the solution behavior when applied to the ODE, $du/dt = 0$. The zero-stability can be equivalently analyzed using a root test.

4. The Dahlquist equivalence theorem states that a multistep scheme is convergent if and only if the scheme is consistent and zero-stable.

5. If the truncation error is bounded by $\|\tau\|_\infty \leq C\Delta t^{q+1}$, then the multistep scheme is q-th order accurate given the scheme is zero-stable.

6. Higher-order multistep schemes can be constructed by matching coefficients of the Taylor expansion of the difference equation.

7. A given multistep scheme can be assessed based on the following characteristics: the number of steps involved; whether it is implicit or explicit (which affects cost); the order of accuracy; and the absolute stability diagram.

8. Some of the popular multistep schemes include the following: forward and backward Euler; Crank-Nicolson; Adams-Moulton; Adams-Bashforth; and backward differentiation formulas (BDFs). Given a scenario, a suitable scheme can be selected based on the aforementioned criteria.

Lecture 15

Initial value problems: multistage methods

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

15.1 Introduction

We have so far considered multi*step* methods — a class of methods that incorporates the state at several previous time steps to advance the current state. We now introduce multi*stage* methods — a class of methods that only uses the state at the current time step to advance the solution but performs this update in several stages. By far the most popular family of multistage methods is *Runge-Kutta methods*. We will hence focus on Runge-Kutta methods in this section.

15.2 Two-stage (explicit) Runge-Kutta method

To introduce the Runge-Kutta family of schemes, we consider a concrete example: a two-stage Runge-Kutta method (RK2). The RK2 method advances the solution from $\tilde{u}^{j-1} = \tilde{u}(t^{j-1})$ to $\tilde{u}^j = \tilde{u}(t^j)$ by the following sequence of updates:

$$\begin{aligned} v_1 &= \tilde{u}^{j-1}, & F_1 &= f(v_1, t^{j-1}), \\ v_2 &= \tilde{u}^{j-1} + \frac{1}{2}\Delta t F_1, & F_2 &= f(v_2, t^{j-1} + \frac{1}{2}\Delta t), \\ \tilde{u}^j &= \tilde{u}^{j-1} + \Delta t F_2. \end{aligned}$$

A single step of RK2 has two stages and requires two evaluations of f to compute F_1 and F_2. Note RK2 is an explicit scheme; the arguments to the function f is always known (and hence we need not solve a nonlinear equation).

Local truncation error and consistency. In order to analyze the consistency of the RK2 method, we compute the *local truncation error*. As in the case of multistep methods, the local truncation error is defined as the remainder associated with the substitution of the exact solution of the IVP to the difference equation.

For the RK2 method, we first note that

$$\hat{F}_1 = f(v_1, t^{j-1}) = f(u(t^{j-1}), t^{j-1}) = u'(t^{j-1});$$

here $\hat{\cdot}$ indicates that the variable is associated with the exact solution u rather than the approximate solution $\tilde{u}$. We then note that

$$\hat{v}_2 = u(t^{j-1}) + \frac{1}{2}u'(t^{j-1})\Delta t,$$

and hence

$$\begin{aligned}
\hat{F}_2 &= f(u(t^{j-1}) + \frac{1}{2}u'(t^{j-1})\Delta t, t^{j-1} + \frac{1}{2}\Delta t) \\
&= \underbrace{f(u(t^{j-1}), t^{j-1})}_{u'(t^{j-1})} + \underbrace{\left.\frac{\partial f}{\partial u}\right|_{(u(t^{j-1}),t^{j-1})} \frac{1}{2}u'(t^{j-1})\Delta t + \left.\frac{\partial f}{\partial t}\right|_{(u(t^{j-1}),t^{j-1})} \frac{1}{2}\Delta t}_{\frac{1}{2}\frac{d}{dt}(f(u(t),t))\Delta t \equiv \frac{1}{2}u''(t^{j-1})\Delta t} + \mathcal{O}(\Delta t^2) \\
&= u'(t^{j-1}) + \frac{1}{2}u''(t^{j-1})\Delta t + \mathcal{O}(\Delta t^2).
\end{aligned}$$

It follows that the truncation error is

$$\begin{aligned}
\tau^j &\equiv u(t^j) - u(t^{j-1}) - \Delta t\hat{F}_2 \\
&= u'(t^{j-1})\Delta t + \frac{1}{2}u''(t^{j-1})\Delta t^2 + \mathcal{O}(\Delta t^3) - \Delta t(u'(t^{j-1}) + \frac{1}{2}u''(t^{j-1})\Delta t + \mathcal{O}(\Delta t^2)) \\
&= \mathcal{O}(\Delta t^3).
\end{aligned}$$

The local truncation error is $|\tau^j| < C\Delta t^3$ for some C independent of Δt. We also note that $\|\tau\|_\infty/\Delta t \to 0$ as $\Delta t \to 0$, and hence the method is consistent.

Convergence. Because all Runge-Kutta methods are single-step method, we can prove convergence using an argument similar to that used for Euler methods. The final result is that if

$$\|\tau\|_\infty \leq C\Delta t^{q+1},$$

then

$$|u(t^j) - \tilde{u}(t^j)| \leq \tilde{C}\Delta t^q \quad \text{for any fixed } t^j = j\Delta t \text{ as } \Delta t \to 0.$$

In words, if the truncation error of a Runge-Kutta scheme is bounded by $C\Delta t^{q+1}$, then the scheme is convergent and is q-th order accurate. For instance, the truncation error of the RK2 method is $\mathcal{O}(\Delta t^3)$, and hence the scheme is second-order accurate.

Proof of convergence (sketch). We now provide a sketch of the proof. We first note that we can abstract the update $\tilde{u}^j = \tilde{u}^{j-1} + \Delta t F_2$ as

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t\Phi(\tilde{u}^{j-1}, t^{j-1}; \Delta t);$$

the abstraction emphasizes the fact that the update F_2 is a function of only $\tilde{u}^{j-1}$, t^{j-1}, and Δt. We in addition note that Φ is Lipschitz continuous in the first argument:

$$|\Phi(w, t; \Delta t) - \Phi(v, t; \Delta t)| \leq L|w - v|$$

for some Lipschitz constant L. We now subtract the difference equation

$$\tilde{u}^j - \tilde{u}^{j-1} - \Delta t\Phi(\tilde{u}^{j-1}, t^{-1}j; \Delta t) = 0$$

from the definition of the truncation error

$$u(t^j) - u(t^{j-1}) - \Delta t \Phi(u(t^{j-1}), t^{j-1}; \Delta t) = \tau^j$$

to obtain

$$e^j - e^{j-1} - \Delta t[\Phi(u(t^{j-1}), t^{j-1}; \Delta t) - \Phi(\tilde{u}^{j-1}, t^{j-1}; \Delta t)] = \tau^j,$$

where $e^j \equiv u(t^j) - \tilde{u}^j$. We now rearrange the equations, appeal to the Lipchitz continuity of Φ, and invoke triangle inequality to obtain

$$|e^j| \leq |e^{j-1}| + \Delta t L |e^{j-1}| + |\tau^j| = (1 + \Delta t L)|e^{j-1}| + |\tau^j|.$$

Using a geometric series similar to that used for the analysis of Euler methods, we can readily show

$$|e^j| \leq \sum_{k=1}^{j} (1 + \Delta t L)^{j-k+1} |\tau^k| \leq \frac{(1 + \Delta t L)^j - 1}{\Delta t L} \|\tau\|_\infty \leq \frac{\exp(2t^j L) - 1}{L} \frac{\|\tau\|_\infty}{\Delta t}.$$

It follows that if $\|\tau\|_\infty \leq C \Delta t^{q+1}$, then

$$|e^j| \leq \frac{\exp(2t^j L) - 1}{L} C \Delta t^q = \tilde{C} \Delta t^q.$$

We observe that i) the scheme is convergent because $\max_j |e^j| \to 0$ as $\Delta t \to 0$, and ii) the scheme is q-th order accurate if the truncation error is order $q + 1$. (Note that a slight modification of the proof also allows us to treat implicit Runge-Kutta schemes, which are introduced shortly.)

Absolute stability. To analyze the absolute stability of the RK2 method, we again consider an IVP of the form

$$\frac{du}{dt} = -\lambda u,$$
$$u(t = 0) = 1.$$

The analytical solution to this IVP is stable if $\Re(\lambda) \leq 0$ and is unstable if $\Re(\lambda) > 0$. As before, we say a scheme is absolutely stable for a given $\lambda \Delta t$ if the difference equation associated with the IVP yields $|\tilde{u}^j| \leq |\tilde{u}^{j-1}|$. Figure 15.1 shows the absolute stability diagram of the RK2 scheme. We see that the stability region is similar to the forward Euler method. In particular, the region of absolute stability along the negative real axis is $|\lambda| \Delta t \leq 2$, or $t \leq t_{\text{cr}} \equiv 2/|\lambda|$.

15.3 Four-stage (explicit) Runge-Kutta method

Another popular Runge-Kutta method — and perhaps the most popular of all Runge-Kutta methods — is a four-stage Runge-Kutta method (RK4). (We note that there are a few different variants of RK4.) The update formula for a popular RK4 method is

$$\begin{aligned}
v_1 &= \tilde{u}^{j-1}, & F_1 &= f(v_1, t^{j-1}), \\
v_2 &= \tilde{u}^{j-1} + \frac{1}{2}\Delta t F_1, & F_2 &= f(v_2, t^{j-1} + \frac{1}{2}\Delta t), \\
v_3 &= \tilde{u}^{j-1} + \frac{1}{2}\Delta t F_2, & F_3 &= f(v_3, t^{j-1} + \frac{1}{2}\Delta t), \\
v_4 &= \tilde{u}^{j-1} + \Delta t F_3, & F_4 &= f(v_4, t^{j-1} + \Delta t), \\
\tilde{u}^j &= \tilde{u}^{j-1} + \Delta t(\frac{1}{6}F_1 + \frac{1}{3}F_2 + \frac{1}{3}F_3 + \frac{1}{6}F_4).
\end{aligned}$$

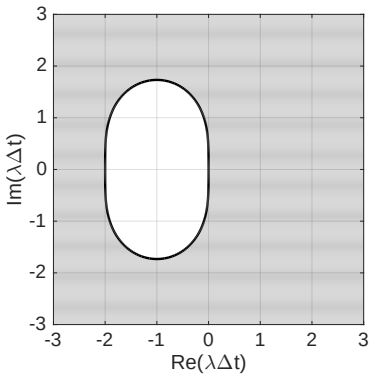


Figure 15.1: The absolute stability diagram of the RK2 scheme. The scheme is stable in the unshaded region; the scheme is unstable in the shaded region.

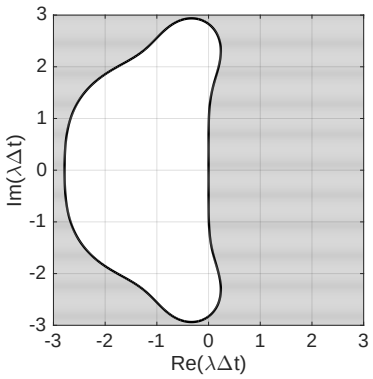


Figure 15.2: Absolute stability diagram for the RK4 method.

As the name suggest, a single step of RK4 has four stages and requires four evaluations of f to compute F_1, F_2, F_3, and F_4. We also note that RK4 is an explicit scheme: all the stage updates can be computed in an explicit manner without solving a nonlinear problem.

Local truncation error and consistency. Following the analysis similar to that used for RK2, we can show that the local truncation error of the RK4 scheme is fifth-order: $\|\tau\|_\infty \leq C\Delta t^5$. We here omit the proof.

Convergence. Even though the number of stages involved in RK4 is greater than that for RK2, the scheme still fit in the abstract form

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \Phi(\tilde{u}^{j-1}, t^{j-1}; \Delta t).$$

Hence, the convergence proof from the RK2 section still applies. Because the truncation error is $\mathcal{O}(\Delta t^5)$, the RK4 method is fourth-order accurate: $|u(t^j) - \tilde{u}(t^j)| \leq C\Delta t^4$ for any t^j.

Absolute stability. Figure 15.2 shows the absolute stability diagram for the RK4 scheme. Note that the region of absolute stability is larger than that for the RK2 scheme. This is unlike multistep schemes, whose stability region tends to shrink as the order increases.

15.4 Two-stage implicit Runge-Kutta method

The RK2 and RK4 methods are both explicit Runge-Kutta methods. There are also implicit Runge-Kutta methods. Implicit methods tend to enjoy better stability properties. As an example, we introduce a two-stage implicit Runge-Kutta (IRK2) method. The update formula for the IRK2 method is

$$v_1 = \tilde{u}^{j-1} + A_{11}\Delta t f(v_1, t^{j-1} + c_1\Delta t) + A_{12}\Delta t f(v_2, t^{j-1} + c_2\Delta t),$$
$$v_2 = \tilde{u}^{j-1} + A_{21}\Delta t f(v_1, t^{j-1} + c_1\Delta t) + A_{22}\Delta t f(v_2, t^{j-1} + c_2\Delta t)$$

where

$$A_{11} = \frac{1}{4}, \quad A_{12} = \frac{1}{4} - \frac{\sqrt{3}}{6}, \quad A_{21} = \frac{1}{4} + \frac{\sqrt{3}}{6}, \quad A_{22} = \frac{1}{4},$$
$$c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad \text{and} \quad c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}.$$

We then set the solution at the current time step equal to

$$\tilde{u}^j = \tilde{u}^{j-1} + \frac{1}{2}\Delta t[f(v_1, t^{j-1} + c_1\Delta t) + f(v_2, t^{j-1} + c_2\Delta t)].$$

We note that to compute v_1 and v_2, we need to solve a system of coupled nonlinear equations; hence the method is implicit. This is unlike explicit methods, whose states at all stages can be evaluated in an explicit manner.

The particular IRK2 scheme presented here is based on the Gauss-Legendre quadrature that we studied earlier. The stage state evaluation points, c_1 and c_2, are in fact the quadrature points associated with the Gauss quadrature rule. For this reason, the method is also called *Gauss-Legendre Implicit Runge-Kutta method*.

Convergence. Using the same analysis technique as before, we can show that the IRK2 scheme is fourth-order accurate. (This also follows from the fact that two-point Gauss quadrature rule integrates exactly polynomials of degree up to three; and hence the leading error is fourth order.)

Absolute stability. Figure 15.3 shows the absolute stability diagram of the IRK2 scheme. Thanks to the implicit construction, the IRK2 method is very stable; the scheme is in fact A-stable. In addition, the stability diagram matches exactly the stability diagram for the exact solution.

Computational cost. The IRK2 method requires the solution of fully coupled nonlinear system by, for instance, Newton's method. Hence the computational cost per step can be significantly higher than that for explicit Runge-Kutta schemes.

15.5 General form of Runge-Kutta methods and Butcher tableau

In general, the intermediate stage states of a Runge-Kutta method is governed by system of equations

$$v_k = \tilde{u}^{j-1} + \Delta t \sum_{l=1}^{K} A_{kl} F_l, \quad k = 1, \ldots, K,$$

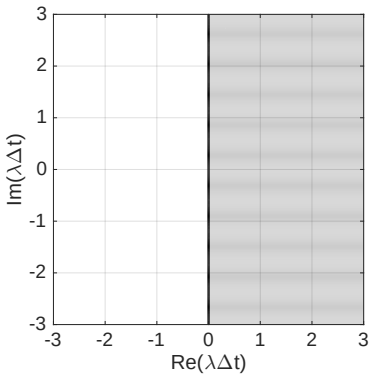


Figure 15.3: Absolute stability diagram for the IRK2 method.

where

$$F_k = f(v_k, t^{j-1} + c_k \Delta t),$$

and the final step update is given by

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=1}^{K} b_k F_k.$$

Here the $K \times K$ matrix A, K-vector c, and K-vector b are the coefficients that characterize the particular K-step Runge-Kutta scheme. These coefficients are often condensed in a *Butcher tableau*:

$$\begin{array}{c|ccc} c_1 & A_{11} & \cdots & A_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ c_K & A_{K1} & \cdots & A_{KK} \\ \hline & b_1 & \cdots & b_K \end{array}.$$

For instance, the tableau for the (explicit) RK2 scheme is

$$\begin{array}{c|cc} 0 & & \\ 1/2 & 1/2 & \\ \hline & 0 & 1 \end{array}$$

and the tableau for the (explicit) RK4 scheme is

$$\begin{array}{c|cccc} 0 & & & & \\ 1/2 & 1/2 & & & \\ 1/2 & 0 & 1/2 & & \\ 1 & 0 & 0 & 1 & \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

Note that, for explicit schemes, the diagonal and superdiagonal of the A matrix are always equal to zero. This triangular structure enables an explicit evaluation of each successive stage state.

The tableau for the (implicit) RK2 scheme is

$$\begin{array}{c|cc} 1/2-\sqrt{3}/6 & 1/4 & 1/4-\sqrt{3}/6 \\ 1/2+\sqrt{3}/6 & 1/4+\sqrt{3}/6 & 1/4 \\ \hline & 1/2 & 1/2 \end{array}$$

Note that, for implicit schemes, the diagonal and superdiagonal are in general non-zero. Hence, we must solve a fully coupled system to compute the stage updates.

15.6 Adaptive Runge-Kutta methods

So far, we have considered methods for IVPs which use a fixed time step Δt. We here provide a very brief introduction to adaptive Runge-Kutta schemes, which automatically select the time step depending on the user specified error tolerance. The idea of adaptive computing itself is very similar to what we have already seen in the context of numerical integration with adaptive quadrature rules. The key ingredients again are

1. error estimation (i.e. how big is the current error?)
2. adaptive selection of the step size given the error estimate.

We consider these two steps separately.

Error estimation. One convenient way to approximate the solution *and* estimate the error is to employ an *embedded Runge-Kutta* method. An embedded Runge-Kutta method constructs two approximations of different order of accuracy using a shared set of function evaluation points. The Butcher tableau is of the form

$$\begin{array}{c|ccc} c_1 & A_{11} & \cdots & A_{1K} \\ \vdots & \vdots & \ddots & \vdots \\ c_K & A_{K1} & \cdots & A_{KK} \\ \hline & b_1 & \cdots & b_K \\ & \hat{b}_1 & \cdots & \hat{b}_K \end{array}.$$

Here, following the standard RK procedure, the K stage states are given by

$$v_k = \tilde{u}^{j-1} + \Delta t \sum_{l=1}^{K} A_{kl} F_l, \quad k = 1, \ldots, K.$$

where

$$F_k = f(v_k, t^{j-1} + c_k \Delta t).$$

Then, as before, we construct the first approximation at the next time step using

$$\tilde{u}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=1}^{K} b_k F_k.$$

We next construct the second approximation at the next time step using

$$\tilde{\tilde{u}}^j = \tilde{u}^{j-1} + \Delta t \sum_{k=1}^{K} \hat{b}_k F_k.$$

We then use the difference between the approximations $\tilde{u}^j$ and $\tilde{\tilde{u}}^j$ as our error estimate:

$$E^j(\Delta t) \equiv |\tilde{u}^j - \tilde{\tilde{u}}^j|;$$

here we include Δt as the argument of E^j to emphasize the fact that the error is dependent on the time step.

There are a number of embedded Runge-Kutta methods, but one popular one is the Dormand-Prince method. The Butcher tableau for the Dormand-Prince method is

$$\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}.$$

(Source: Dormand and Prince, Journal of Computational and Applied Mathematics 6(1): 19–26, 1980.) For Dormand-Prince, the approximation $\tilde{u}^j$ is locally fifth order accurate, whereas the approximation $\tilde{\tilde{u}}^j$ is locally fourth order accurate. The MATLAB routine `ode45` implements the scheme.

Adaptive time stepping. Strategies typically used to adaptively select the time step is based on heuristic and best practice. The basic idea is the following: if the error at a given step is too large, we then reject the step and recompute the approximation using a smaller time step; if the error at a given step is too small, we then propose to use a larger time step for the next step. Specifically, if we wish to achieve the error tolerance of $\epsilon_{\text{tol}}(\Delta t)$ at the final time t_f, we then apportion to the current time step

$$\epsilon_{\text{tol}}^{\text{local}}(\Delta t) \equiv \epsilon_{\text{tol}} \frac{\Delta t}{t_f}.$$

The goal is to adaptively select the time step such that the error committed in each step is less than, but close to, $\epsilon_{\text{tol}}^{\text{lcoal}}$.

Suppose $E^j(\Delta t) > \epsilon_{\text{tol}}^{\text{local}}(\Delta t)$ in the j-th time step. Then we reject the update, reduce the time step, and recompute the approximation. One way to choose the new reduced time step Δt^{new} is to observe that, because the error estimate for Dormand-Prince is locally fifth-order accurate,

$$E^j(\Delta t^{\text{new}}) \approx E^j(\Delta t) \left(\frac{\Delta t^{\text{new}}}{\Delta t} \right)^5.$$

Hence, we set the new time step as the solution to

$$E^j(\Delta t^{\text{new}}) = \gamma \epsilon_{\text{tol}}^{\text{local}}(\Delta t^{\text{new}}),$$

where $\gamma \in (0, 1]$ is some scaling factor to make a conservative estimation. (Take, for instance, $\gamma = 0.8$.)

Now consider the opposite scenario. Suppose $E^j(\Delta t) < \epsilon_{\text{tol}}^{\text{local}}(\Delta t)$ and in fact the error E^j is considerably smaller than local error budget $\epsilon_{\text{tol}}^{\text{local}}$. (Take, for instance, by a factor of 2.) The small error relative to the error budget indicates that we could be taking larger time steps. Specifically, when we compute the solution of the next step, we can use a new increased time step that satisfies

$$E^j(\Delta t^{\text{new}}) = \gamma \epsilon_{\text{tol}}^{\text{local}}(\Delta t^{\text{new}}),$$

where $\gamma \in (0, 1]$ is again some scaling factor.

15.7 Summary

We summarize the key points of this lecture:

1. Multistage methods use the state at only the current time step to advance the solution but performs the update in several stages. This is unlike multistep methods which incorporate the state from a few previous time steps.

2. Runge-Kutta methods are examples of multistage methods. A K-stage RK method computes K intermediate states every time step.

3. For RK methods, if the truncation error is $\mathcal{O}(\Delta t^{q+1})$, then the scheme is q-th order accurate.

4. Explicit RK2 and RK4 are popular multistage methods that are second and fourth order accurate, respectively.

5. Implicit RK methods are more complicated to implement and are computationally expensive per time step, but they typically provide better stability than explicit RK methods.

6. A given RK method can be compactly described using a Butcher tableau.

7. Adaptive RK methods consist of two ingredients: error estimation and time step selection. A common approach to estimate the error is to use an embedded RK method, which provides approximation of two different order, and to compare the two approximations. The time step is then adjusted based on the error estimate and the expected rate of convergence.

Lecture 16

Initial value problems: systems of equations

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

16.1 Introduction

In this lecture we consider approximation of initial value problems (IVPs) governed by a system of ordinary differential equations (ODEs). Specifically, the IVPs that we consider are of the form

$$\begin{aligned} \frac{du}{dt} &= f(u,t), \quad t \in (0, t_f], \\ u(t=0) &= u^0, \end{aligned}$$

where $u(t) \in \mathbb{R}^n$ is the state vector, $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ is the function that defines the ODE, and $u_0 \in \mathbb{R}^n$ is the initial condition. We could also write the ODE more explicitly component-by-component to emphasize the system aspect:

$$\frac{d}{dt} \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix} = \begin{pmatrix} f_1(u_1, \dots, u_n; t) \\ \vdots \\ f_n(u_1, \dots, u_n; t) \end{pmatrix}.$$

Note that we will use a subscript to denote a component of the state u or function f.

(*Note.* In the previous lectures, we denoted the initial condition by u_0, with the subscript 0; from hereon, we will denote the initial condition by u^0, with the superscript 0, to reserve the subscript for the component index.)

16.2 Multistep method

The extension of the linear multistep method for scalar IVPs to vector IVPs is very simple. Because everything is essentially identical to the scalar case, we here concisely state the key points.

Formulation. Our difference equation is given by

$$\sum_{k=0}^{K} \alpha_k \tilde{u}^{j-k} = \Delta t \sum_{k=0}^{K} \beta_k f(\tilde{u}^{j-k}, t^{j-k}), \quad j = 1, \ldots, J,$$

where $\tilde{u}^j \in \mathbb{R}^n$ is our approximation of the state at time t^j; i.e. $\tilde{u}^j \equiv \tilde{u}(t^j) \approx u(t)$. We may write the difference equation more explicitly as

$$\sum_{k=0}^{K} \alpha_k \begin{pmatrix} \tilde{u}_1^{j-k} \\ \vdots \\ \tilde{u}_n^{j-k} \end{pmatrix} = \Delta t \sum_{k=0}^{K} \beta_k \begin{pmatrix} f_1(\tilde{u}_1^{j-k}, \ldots, \tilde{u}_n^{j-k}; t^{j-k}) \\ \vdots \\ f_n(\tilde{u}_1^{j-k}, \ldots, \tilde{u}_n^{j-k}; t^{j-k}) \end{pmatrix}, \quad j = 1, \ldots, J,$$

where $\tilde{u}_i^j$ denotes the i-th component of our approximation of the state at time t^j. As in the scalar case, a set of coefficients $\{\alpha_k\}$ and $\{\beta_k\}$ defines a particular scheme. For instance, the forward Euler scheme is given by $\alpha_0 = 1$, $\alpha_1 = -1$, $\beta_0 = 0$, and $\beta_1 = 1$. The scheme is said to be explicit if $\beta_0 = 0$; otherwise the scheme is implicit.

Local truncation error. The local truncation error is the residual associated with the substitution of the exact solution to the difference equation; i.e.,

$$\tau^j = \sum_{k=0}^{K} \alpha_k u(t^{j-k}) - \Delta t \sum_{k=0}^{K} \beta_k f(u(t^{j-k}), t^{j-k}), \quad j = 1, \ldots, J.$$

Note that $\tau^j \in \mathbb{R}^n$ is a vector; we may denote the i-th component of the local truncation error at time step j by τ_i^j.

Consistency. We first define the ∞-norm of τ as

$$\|\tau\|_\infty \equiv \max_{\substack{i=1,\ldots,n \\ j=1,\ldots,J}} |\tau_i^j|.$$

Then, a scheme is consistent if

$$\frac{\|\tau\|_\infty}{\Delta t} \to 0 \quad \text{as} \quad \Delta t \to 0;$$

in other words, $|\tau_i^j|/\Delta t$ vanishes as $\Delta t \to 0$ for all components i and all time index j.

Convergence. As in the scalar case, convergence follows from the Dahlquist equivalence theorem. Namely, a scheme is convergent if and only if it is consistent and zero stable:

$$\text{(consistency)} \ + \ \text{(zero stability)} \ \Leftrightarrow \ \text{(convergence)} \ .$$

16.3 Multistage methods

The extension of multistage methods to a system of first order ODEs is also straight forward: we simply replace a scalar state with a vector state. The consistency and convergence are extended in exactly the same manner as the multistep methods.

16.4 Higher-order ODEs

We can employ multistep and multistage methods for systems of ODEs to also solve higher-order ODEs. Consider a general order-q (scalar) ODE of the form

$$\begin{aligned}\frac{d^q u}{dt} &= g(u, \frac{du}{dt}, \ldots, \frac{d^{q-1}u}{dt}; t), \quad \text{for } t \in (0, t_f], \\ \frac{d^i u}{dt^i}(t=0) &= u^{(i),0}, \quad i = 0, \ldots, q-1,\end{aligned}$$

where $u^{(i),0}$, $i = 0, \ldots, q-1$, are the initial conditions. Note that for $q = 1$, we recover the first-order ODE we have already considered.

We can transform this system of a general order-q scalar ODE to a system of first order ODEs. Specifically, let us introduce auxiliary variables w_i, $i = 0, \ldots, q$, where

$$w_i = \frac{d^i u}{dt^i}, \quad i = 0, \ldots, q-1.$$

We can then rewrite the scalar high-order ODE as a system of q first-order equations:

$$\begin{aligned}\frac{dw_{i-1}}{dt} &= w_i, \quad i = 1, \ldots, q-1, \\ \frac{dw_{q-1}}{dt} &= g(w_0, w_1, \ldots, w_{q-1}; t).\end{aligned}$$

The initial conditions for the system of equations is

$$w_i(t=0) = u^{(i),0}, \quad i = 0, \ldots, q-1,$$

where $u^{(i),0}$, $i = 0, \ldots, q-1$, are the $q-1$ initial conditions for the original higher-order IVP. With this recasting of the higher-order ODE as a system of first-order ODEs, we can solve the first-order ODE using the multistep/multistage methods we have developed.

16.5 Analysis of a system of ODEs: stability and stiffness

To motivate the absolute stability analysis of a system of ODEs, let us first consider a linear IVP of the form

$$\begin{aligned}\frac{du}{dt} &= Au, \quad t \in (0, t_f], \\ u(t=0) &= u^0,\end{aligned}$$

where $A \in \mathbb{R}^{n \times n}$ is a time-invariant matrix. To analyze the behavior of the system of equations using the techniques we have developed for a scalar equation, we decouple the system using eigen-decomposition. Specifically, let us assume that A is diagonalizable and set

$$A = V \Lambda V^{-1},$$

where V is a non-singular matrix whose i-th column is the i-th eigenvector of A, and Λ is a diagonal matrix whose i-th diagonal entry is the i-th eigenvalue of A. Using the eigendecomposition, we may rewrite the system of ODEs as

$$\frac{d}{dt}u = V\Lambda V^{-1}u \quad \Rightarrow \quad \frac{d}{dt}V^{-1}u = \Lambda V^{-1}u.$$

If we define a new variable $z \equiv V^{-1}u$ (and $z^0 \equiv V^{-1}z^0$), then we obtain

$$\begin{aligned} \frac{dz}{dt} &= \Lambda z, \quad t \in (0, t_f], \\ z(t=0) &= z^0. \end{aligned}$$

The key observation here is that Λ is a diagonal matrix, and hence the ODEs are decoupled. More explicitly, each of the n components of the transformed variable z must satisfy the ODE

$$\begin{aligned} \frac{dz_i}{dt} &= \lambda_i z, \quad t \in (0, t_f], \\ z_i(t=0) &= z_i^0. \end{aligned}$$

The solution to each of these ODEs is given by $z_i = z_i^0 \exp(\lambda_i t)$; the i-th mode decays exponentially in time if $\Re(\lambda_i) < 0$, grows exponentially in time if $\Re(\lambda_i) > 0$, and is purely oscillatory if λ_i is purely imaginary. Using the transformation $u = Vz$, we note that the solution to the original system of IVPs is given by

$$u(t) = \sum_{i=1}^{n} v_i (z_i^0 \exp(\lambda_i t)),$$

where v_i is the i-th column of V.

(If the matrix A is not diagonalizable, we can transform A to a Jordan normal form. We can then write the solution associated with the Jordan block of size k as $c_1 \exp(\lambda_i t) + c_2 t \exp(\lambda_i t) + \cdots + c_k t^{k-1} \exp(\lambda_i t)$, where the coefficients $c_1, \ldots, c_k$ are deduced from the specific initial condition.)

Stability analysis. Suppose the IVP of interest is stable in the sense that all eigenvalues have non-positive real part. In order for our numerical approximation of the IVP to be stable, our approximation of every mode must be numerically stable; if even one of the modes grow in an unbounded manner, then the solution $u = Vz$ would also grow in an unbounded manner. In order to ensure that there is no unbounded growth, each of the n Δt-weighted eigenvalues, $\lambda_i \Delta t$, must lie in a stable region of the absolute stability diagram. It is important to note that this stability requirement must be met for every mode, even if the mode has a small contribution to the overall response of the system for the particular initial condition; even the mode with a very small z_i^0 at the initial time will eventually dominate the system behavior if it grows exponentially in time.

Stiffness. A system of ODEs is often described as stiff or non-stiff. While there is no mathematically rigorous definition, we will define stiffness based on the *stiffness ratio*,

$$r \equiv \frac{\max_{i=1,\ldots,n} |\lambda_i(A)|}{\min_{i=1,\ldots,n} |\lambda_i(A)|}.$$

Note that the this is the ratio of the time scale associated with the fastest and slowest modes of the system. If the stiffness ratio is large, we say the system is stiff; if the stiffness ratio is $\mathcal{O}(1)$, we say the the system is not stiff.

A very stiff system is difficult to solve using explicit methods, because the relatively small stability region of explicit methods means that we must employ very small time step if $\lambda_{\max}(A)$ is large. For explicit methods, the maximum time step we can take for stiff systems is almost always governed by the absolute stability concerns rather than the accuracy of the approximation. On the other hand, implicit schemes with a large stability region — and in particular A-stable schemes — allow us to take much larger time steps; hence, implicit schemes are more suited for stiff equations. (In fact some people define stiff equations as equations that cannot be solved at a reasonable cost using explicit methods.)

16.6 Example: mass-spring-damper system

We consider a mass-spring-damper system consists of n masses arranged in a line. Each mass is connected to its two neighbors (or a neighbor and a wall for the first and last masses) by a spring and a damper. We denote the mass of each mass by m, the stiffness of each spring by k, and the damping coefficient of each damper by c. The resulting system of second-order ODEs is

$$\begin{aligned}
\frac{d^2x_1}{dt^2} &= \frac{k}{m}(-2x_1 + x_2) + \frac{c}{m}(-2\frac{dx_1}{dt} + \frac{dx_2}{dt}),\\
\frac{d^2x_i}{dt^2} &= \frac{k}{m}(x_{i-1} - 2x_i + x_{i+1}) + \frac{c}{m}(\frac{dx_{i-1}}{dt} - 2\frac{dx_i}{dt} + \frac{dx_{i+1}}{dt}), \quad i = 2, \ldots, n-1,\\
\frac{d^2x_n}{dt^2} &= \frac{k}{m}(x_{n-1} - 2x_n) + \frac{c}{m}(\frac{dx_{n-1}}{dt} - 2\frac{dx_n}{dt}),
\end{aligned}$$

with initial displacement and velocity conditions

$$\begin{aligned}
x_i(t=0) &= x_i^0, \quad i = 1, \ldots, n,\\
x_i'(t=0) &= v_i^0, \quad i = 1, \ldots, n.
\end{aligned}$$

We can write the system more compactly using matrices and vectors as

$$\frac{d^2x}{dt^2} = Kx + C\frac{dx}{dt},$$

where

$$K = \frac{k}{m}\begin{pmatrix} -2 & 1 & 0 & \cdots & 0\\ 1 & -2 & 1 & & \vdots\\ 0 & \ddots & \ddots & \ddots & 0\\ \vdots & & 1 & -2 & 1\\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix}, \quad \text{and} \quad C = \frac{c}{m}\begin{pmatrix} -2 & 1 & 0 & \cdots & 0\\ 1 & -2 & 1 & & \vdots\\ 0 & \ddots & \ddots & \ddots & 0\\ \vdots & & 1 & -2 & 1\\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix},$$

with initial conditions $x(t=0) = x^0$ and $x'(t=0) = v^0$.

We now introduce an augmented state

$$u \equiv \begin{pmatrix} x\\ x' \end{pmatrix} \in \mathbb{R}^{2n},$$

and express the system of second-order ODEs as a (larger) system of first-order ODEs:

$$\frac{du}{dt} = Bu,$$

where

$$B = \begin{pmatrix} 0 & I_{n\times n} \\ K & C \end{pmatrix},$$

with the initial condition $u(t=0) = u^0 = ((x^0)^T, (v^0)^T)^T$. We now consider the solution of this IVP for two separate cases.

Undamped case: $c = 0$. We consider an undamped system ($c = 0$) of eight masses ($n = 8$), each of unit mass ($m = 1.0$), connected by spring of unit stiffness ($k = 1.0$). We consider the initial condition where the first mass is displaced by a unit length. The approximation of the IVP by RK4 for $\Delta t = 0.2$ is shown in Figure 16.1(a). We observe that each of the eight masses oscillate in time, and we observe qualitatively the amplitude of oscillation remains constant over time.

We also show in Figure 16.1(b) all $2n = 16$ eigenvalues of the matrix B superimposed on the stability diagram of RK4. We observe that all eigenvalues of B are purely imaginary, as the solution exhibits a purely oscillatory behavior. We also observe that eigenvalues lie on the boundary of the region of absolute stability for RK4; hence the discrete approximation preserves the oscillatory behavior of the solution without growth or dissipation.

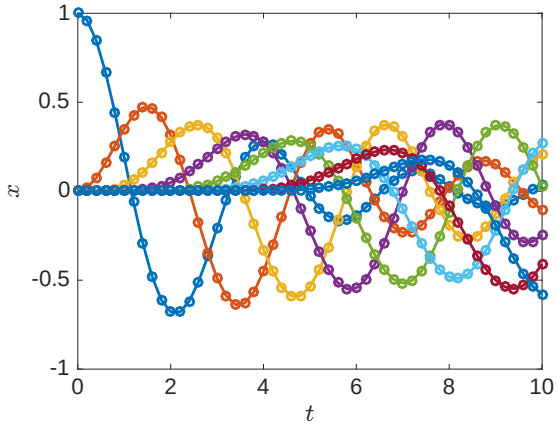
Figure 16.1(c) shows the approximation of the same undamped system by RK4 using a larger time step of $\Delta t = 2$ (as opposed to $\Delta t = 0.2$ for the previous case). We observe that this time the solution becomes unstable. Figure 16.1(d) shows the cause of the instability; some of the eigenvalues lie in the region of instability for RK4, and hence these modes grow exponentially in time.

Highly damped case: $c = 2$. We consider a highly damped system ($c = 2$) of eight masses ($n = 8$), each of unit mass ($m = 1.0$), connected by spring of unit stiffness ($k = 1.0$). We again consider the initial condition where the first mass is displaced by a unit length. Figure 16.2(a) shows the approximation by RK4 for $\Delta t = 0.2$. We observe that the displacement of the first mass decays (essentially) exponentially in time, and the system exhibits very limited oscillation. The eigenvalue distribution shown in Figure 16.2(b) confirms that most of the modes decay exponentially in time, as most of the eigenvalues lie along the negative real axis.

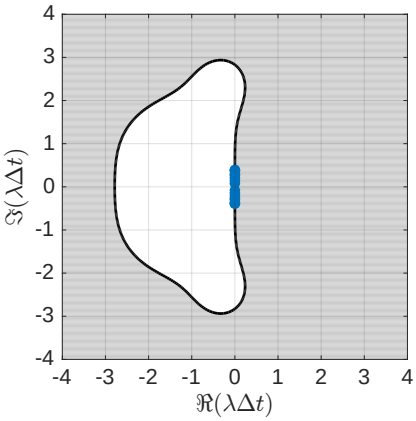
Figures 16.2(c) and 16.2(d) shows the RK4 approximation and the associated stability diagram for a larger time step of $\Delta t = 0.5$ (as opposed to $\Delta t = 0.2$ for the previous case). The maximum eigenvalue of B in magnitude is $|\lambda_{\max}| \approx 7.22$. Hence, for RK4, the critical time step is $t_{\text{cr}} \approx 2.8/|\lambda_{\max}| \approx 3.9$. As a result, the approximation is unstable for $\Delta t = 0.5$. The stiffness ratio for this problem is $\max_i |\lambda_i| / \min_i |\lambda_i| \approx 20.8$, which is moderately high. As a result, while the solution shows a simple behavior in time, we must use a small time step in order for the RK4 scheme to remain stable. An A-stable (implicit) scheme is arguably a better choice even for this moderately stiff problem.

16.7 Nonlinear equations: computation

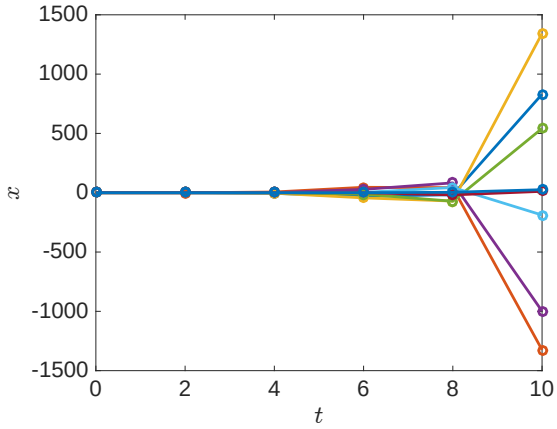
If the scheme is explicit, we simply evaluate f (or a set of f's) and evaluate the state at the next time step.



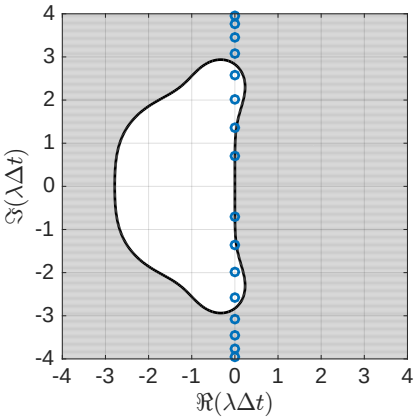
(a) RK4 approximation ($\Delta t = 0.2$)



(b) stability diagram ($\Delta t = 0.2$)

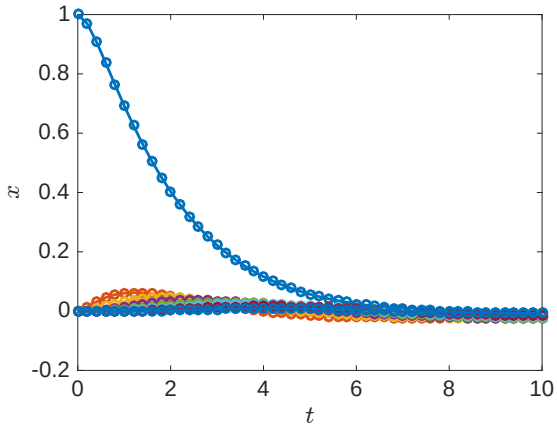


(c) RK4 approximation ($\Delta t = 2$)

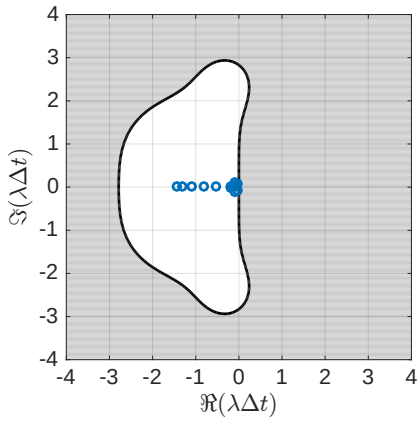


(d) stability diagram ($\Delta t = 2$)

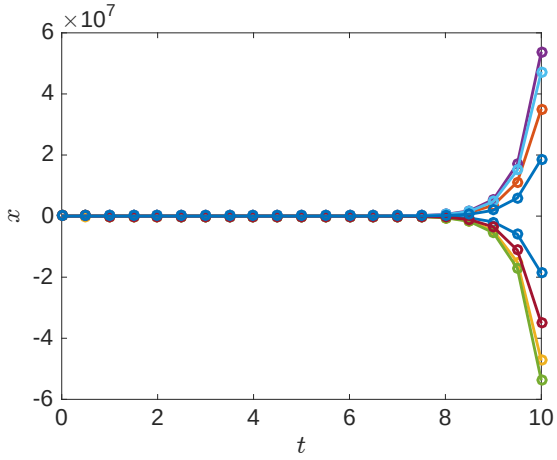
Figure 16.1: RK4 approximation and eigenvalue distribution associated with the mass-spring-damper system for $n = 8$, $m = 1$, $k = 1$, and $c = 0$.



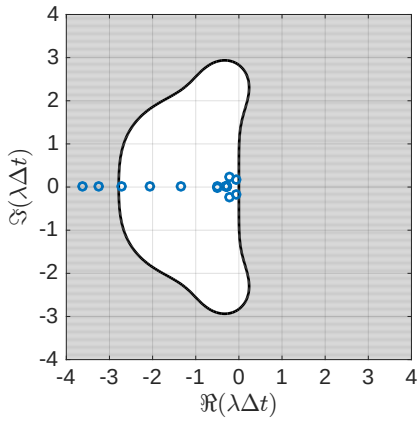
(a) RK4 approximation ($\Delta t = 0.2$)



(b) stability diagram ($\Delta t = 0.2$)



(c) RK4 approximation ($\Delta t = 0.5$)



(d) stability diagram ($\Delta t = 0.5$)

Figure 16.2: RK4 approximation and eigenvalue distribution associated with the mass-spring-damper system for $n = 8$, $m = 1$, $k = 1$, and $c = 2$.

If the scheme is implicit, we must solve a system of nonlinear equation in each time step. For concreteness, let us consider the backward Euler method. In each time step, we must solve

$$r(w) = w - \tilde{u}^{j-1} - \Delta t f(w, t^j) = 0,$$

where $r : \mathbb{R}^n \to \mathbb{R}^n$ is the residual form whose root is $\tilde{u}^j$. We may find the root using, for instance, Newton's method. Note that the Jacobian of the residual is given by

$$\nabla r(w) = I - \Delta t[\nabla f(w, t^j)],$$

where $\nabla f(w, t^j) \in \mathbb{R}^{n \times n}$ is the Jacobian of the function f with respect to the state evaluated about w. For a system of equations, the evaluation of the Jacobian, which has $n \times n$ entries, and the solution of the $n \times n$ linear system, which in general requires $\mathcal{O}(n^3)$ operations, can be computationally expensive. Thus, per time step, implicit schemes are significantly more expensive than explicit schemes, especially for a system of nonlinear equations; however, their favorable absolute stability allows us to take much a larger time step than implicit schemes.

16.8 Nonlinear equations: stability analysis

One approach to analyze the stability of a system of nonlinear equations is to linearize the problem about the solution trajectory and to analyze the eigenvalues of the Jacobian ∇f along the trajectory. While the approach is not entirely rigorous, it nevertheless works well in practice when the state does not change very rapidly in time. To motivate the analysis, we first decouple the solution $u(t) \in \mathbb{R}^n$ into a stationary state and a (presumably small) time-varying perturbation: $u(t) = \bar{u} + \delta u(t)$. We then note that the perturbation $\delta u(t) \in \mathbb{R}^n$ satisfies the following relationship:

$$\frac{d(\delta u)}{dt} = \frac{du}{dt} = f(u(t), t) = f(\bar{u} + \delta u(t), t) \approx f(\bar{u}) + \nabla f(\bar{u})\delta u(t) + \mathcal{O}(\|\delta u\|_2^2).$$

In other words, for $\|\delta u\|_2^2$ small, the perturbation obeys a linearized equation

$$\frac{d(\delta u)}{dt} \approx A\delta u(t) + b,$$

where $A \equiv \nabla f(\bar{u}) \in \mathbb{R}^{n \times n}$ and $b \equiv f(\bar{u}) \in \mathbb{R}^n$. Hence, if an eigenvalue of λ_i of $\nabla f(\bar{u}) \in \mathbb{R}^{n \times n}$ i) has a negative real part ($\Re(\lambda_i) < 0$) but ii) $\lambda_i \Delta t$ lies outside of the region of absolute stability of a given scheme, then the numerical approximation will likely be unstable.

In practice, there is no way to know in advance the eigenvalues associated with the Jacobian $\nabla f(u(t))$ since the solution is unknown. For a fixed Δt schemes, sometimes an appropriate time step must be found through trial and error. For an adaptive Δt scheme, the instability can be detected from the increase in the error and the scheme could reduce Δt. Of course, a more robust solution is to consider a use of an A-stable scheme, which is stable over the entire left hand plane; for these reasons, implicit schemes are often used (especially for stiff systems) despite the higher computational cost per step.

16.9 Summary

We summarize the key points of this lecture:

1. Both multistep and multistage methods can be extended to a system of equations in a straightforward manner.

2. A (system of) higher-order IVPs can be recast as a (larger) system of first-order IVPs.

3. The stability of a system of linear IVPs, $du/dt = Au$, is governed by the eigenvalues of the matrix A. In order for a numerical approximation to be stable, all eigenvalues of A must lie in the region of absolute stability of the scheme.

4. A system is said to be stiff if the ratio of the largest and smallest eigenvalues — and hence the range of time scales — is large.

5. Explicit methods are often ill-suited for stiff systems due to a (very) small critical time step to remain stable.

6. The solution of a system of nonlinear IVPs by an implicit method requires the solution of a system of nonlinear (algebraic) equations.

7. The stability of a system of nonlinear IVPs can be assessed in practice by analyzing the eigenvalues of the Jacobian matrix ∇f along the solution trajectory.

Lecture 17

Boundary value problems: 1d Poisson's equation

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

17.1 Introduction

We have so far considered initial value problems (IVPs), where the problem is defined by i) ordinary differential equations (ODEs) and ii) an initial condition. IVPs are solved by successively computing the state forward in time starting with the initial condition.

We now consider a different class of problems called *boundary value problems* (BVPs). BVPs are typically posed over a physical (spatial) domain and are governed by an ODE or a partial differential equation (PDE) and boundary conditions. In this lecture, we consider BVPs in one dimension.

17.2 Poisson's equation

As a canonical problem, consider Poisson's equation in one dimension,

$$\begin{aligned} -\frac{d^2u}{dx^2} &= f \quad \text{in } \Omega \equiv (0,1), \\ u(x=0) &= 0, \\ u(x=1) &= 0. \end{aligned}$$

Here, Ω is the physical domain of interest, $u \in C^2(\Omega)$ is the state, $f \in C^0(\Omega)$ is the forcing function, and $u(x=0)=0$ and $u(x=1)=0$ are the boundary conditions on the left and right boundaries, respectively. Note that unlike the IVPs we have considered in the previous lectures, BVPs have conditions imposed on both ends of the domain. We will see this (seemingly subtle) difference introduces a significant difference in the solution behavior as well as the associated numerical solution strategy.

Before we consider numerical solution of BVPs, we comment on the characteristics of the exact solution. First, for any $f \in C^0(\Omega)$, the solution is exists and is unique. Second, the solution is smoother than the forcing function; specifically, if $f \in C^m(\Omega)$, then $u \in C^{m+2}(\Omega)$.

17.3 Finite difference method

We now consider the approximation of Poisson's equation by a finite difference method. We first introduce a computational grid over the domain $\Omega \equiv [0,1]$. The gird points are given by

$$x_i = i\Delta x, \quad i = 0, \ldots, n+1,$$

where $\Delta x \equiv 1/(n+1)$ is the spacing between any two grid points. We then denote our approximation of the solution by

$$\tilde{u}_i \equiv \tilde{u}(x_i), \quad i = 0, \ldots, n+1.$$

We now apply the central difference formula to approximate the second derivative $\frac{d^2u}{dx^2}$; i.e.

$$\left.\frac{d^2u}{dx^2}\right|_i \approx \frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2}.$$

The substitution of the finite difference formula to Poisson's equation yields

$$-\frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2} = \tilde{f}_i, \quad i = 1, \ldots, n,$$

with boundary conditions

$$\tilde{u}_{i=0} = \tilde{u}_{i=n+1} = 0,$$

where $\tilde{f}_i = f(x_i)$. Note that we have n unknowns and n equations associated with the interior grid points $i = 1, \ldots, n$. These equations can be expressed in a matrix form:

$$A\tilde{u} = \tilde{f},$$

where

$$A = \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}, \qquad \tilde{u} = \begin{pmatrix} \tilde{u}_1 \\ \tilde{u}_2 \\ \vdots \\ \tilde{u}_{n-1} \\ \tilde{u}_n \end{pmatrix}, \qquad \tilde{f} = \begin{pmatrix} \tilde{f}_1 \\ \tilde{f}_2 \\ \vdots \\ \tilde{f}_{n-1} \\ \tilde{f}_n \end{pmatrix}.$$

We easily verify that the matrix A is symmetric. The matrix is also positive definite for any Δx:

$$\begin{aligned} v^T A v &= \frac{1}{\Delta x^2}[\sum_{i=1}^{n} 2v_i^2 - \sum_{i=1}^{n-1} 2v_i v_{i+1}] = \frac{1}{\Delta x^2}[v_1^2 + \sum_{i=1}^{n-1}(v_i^2 - 2v_i v_{i+1} + v_{i+1}^2) + v_n^2] \\ &= \frac{1}{\Delta x^2}[v_1^2 + \sum_{i=1}^{n-1}(v_i - v_{i+1})^2 + v_n^2] > 0 \quad \forall v \neq 0. \end{aligned}$$

It follows that the linear system is well-posed, and a unique solution $\tilde{u}$ exists for any f.

17.4 Truncation error

We define the *truncation error* at the i-th grid point as

$$\tau_i \equiv -\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} - f(x_i).$$

We recall from the lecture on numerical differentiation that the central difference formula is second order accurate, and more specifically,

$$\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} = \left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} + \frac{1}{12}u^{(4)}(\xi_i)\Delta x^2,$$

for some $\xi_i \in [x_{i-1}, x_{i+1}]$. It follows that the truncation error is given by

$$\tau_i = \underbrace{-\left.\frac{\partial^2 u}{\partial x^2}\right|_{x_i} - f(x_i)}_{=0 \text{ by ODE}} - \frac{1}{12}u^{(4)}(\xi_i)\Delta x^2 = -\frac{1}{12}u^{(4)}(\xi_i)\Delta x^2, \quad i = 1, \ldots, n.$$

It follows that

$$\|\tau\|_\infty \leq \frac{1}{12}\max_{s\in[0,1]} |u^{(4)}(s)|\Delta x^2,$$

where $\|\tau\|_\infty \equiv \max_{i=1,\ldots,n} |\tau_i|$. The truncation error for the central finite difference formula applied to Poisson's equation is second order.

17.5 Properties of A^{-1} and stability

We now note the two important properties of the matrix A^{-1}. For notational convenience, let us denote the i, j entry of A^{-1} by α_{ij}; in other words,

$$A^{-1} = \begin{pmatrix} \alpha_{11} & \cdots & \alpha_{1n} \\ \vdots & \ddots & \vdots \\ \alpha_{n1} & \cdots & \alpha_{nn} \end{pmatrix}.$$

We now provide two important properties of A^{-1}:

1. All entries of A^{-1} are positive: $\alpha_{ij} > 0$, $i, j = 1, \ldots, n$;
2. The infinity norm of the matrix A^{-1} is bounded for all Δx:

$$\|A^{-1}\|_\infty \equiv \max_{v\in\mathbb{R}^n} \frac{\|A^{-1}v\|_\infty}{\|v\|_\infty} = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |\alpha_{ij}| \leq \frac{1}{8}.$$

We now sketch the proofs.

Proof of property 1. To prove the first property, we first introduce a notational shorthand for convenience. Specifically, let us denote $v \geq 0$ if $v_i \geq 0$ for all $i = 1, \ldots, n$. Then, to prove the positivity of the entries of A^{-1}, it suffices to show that if $v = A^{-1}w$ and $w \geq 0$ then $v \geq 0$. (This

is true because we can chose w which is (positive) nonzero on just one of the entries, say the k-th entry; $v \geq 0$ then implies that the entries in the k-th column of A^{-1} are all positive.)

Now, let $i_{\min}$ be the index associated with the smallest component of v: $i_{\min} = \arg\min_i v_i$. We observe that $i_{\min}$ must be either the first grid point ($i_{\min} = 1$) or the last grid point ($i_{\min} = n$), as otherwise

$$-v_{i_{\min}-1} + 2v_{i_{\min}} - v_{i_{\min}+1} < 0,$$

which violates our assumption that $w \geq 0$. Now, if $i_{\min} = 1$, then $2v_1 - v_2 \geq 0$, which implies that $v_1 \geq v_2 - v_1 \geq 0$, and thus $v \geq 0$. Similarly, if $i_{\min} = n$, then $-v_{n-1} + 2v_n \geq 0$, which implies that $v_n \geq v_{n-1} - v_n \geq 0$, and thus $v \geq 0$. It follows that if $v = A^{-1}w$ and $w \geq 0$, then $v \geq 0$; in other words, all the entries of A^{-1} are positive.

Proof of property 2. To prove the second property, we note that the function $v(x) = \frac{1}{2}x(1-x)$ satisfies

$$-\frac{v(x_{i+1}) - 2v(x_i) + v(x_{i-1})}{\Delta x^2} = 1,$$

because $v'' = 1$ and $v^{(4)} = 0$, which implies that the truncation error is 0. This implies that a vector $\tilde{v}_i = v(x_i)$ and $\tilde{w}_i = 1$ satisfy the equation $A\tilde{v} = \tilde{w}$ or, equivalently, $\tilde{v} = A^{-1}\tilde{w}$. It follows that

$$\|A^{-1}\|_\infty \equiv \max_{i=1,\ldots,n} \sum_{j=1}^{n} |\alpha_{ij}| = \max_{i=1,\ldots,n} \sum_{j=1}^{n} \alpha_{ij} = \max_{i=1,\ldots,n} \tilde{v}_i \leq \max_{x\in[0,1]} v(x) = \frac{1}{8}.$$

Note that the second equality follows from the fact that all entries of A^{-1} are positive and hence $|\alpha_{ij}| = \alpha_{ij}$.

Implications. The first property implies that if all entries of $\tilde{f}$ are positive, then all entries of $\tilde{u}$ are positive. The second property implies that our finite difference approximation is stable in the sense that

$$\|\tilde{u}\|_\infty \leq \frac{1}{8}\|\tilde{f}\|_\infty.$$

In words, for any finite data $\tilde{f}$, the approximation solution $\tilde{u}$ is bounded.

17.6 Convergence

We now define the error at grid point i as

$$e_i \equiv u(x_i) - \tilde{u}_i.$$

We then appeal to the definition of the truncation error to obtain

$$-\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{\Delta x^2} = \tilde{f}_i + \tau_i;$$

we also recall the finite difference equation

$$-\frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2} = \tilde{f}_i.$$

Taking the difference of the two equations, we obtain the *error equation*,

$$-\frac{e_{i+1} - 2e_i + e_{i-1}}{\Delta x^2} = \tau_i, \quad i = 1, \ldots, n,$$

with boundary conditions $e_0 = e_{n+1} = 0$. The error equation in matrix form is

$$Ae = \tau.$$

It follows that $e = A^{-1}\tau$. We appeal to the stability of A^{-1} to obtain

$$\|e\|_\infty = \|A^{-1}\tau\|_\infty \leq \|A^{-1}\|_\infty \|\tau\|_\infty \leq \frac{1}{8}\|\tau\|_\infty.$$

The substitution of the upper bound for the truncation error yields an error bound

$$\|e\|_\infty \leq \frac{1}{96} \max_{s\in\Omega} |u^{(4)}(s)| \Delta x^2.$$

We observe that

1. the scheme is convergent: $\|e\|_\infty \to 0$ as $\Delta x \to 0$;
2. the scheme is second-order accurate: $\|e\|_\infty \leq C\Delta x^2$.

Note that, once we obtain the error equation, the rest of the convergence proof relies on *i*) the stability of A and *ii*) the decrease of the truncation error with Δx. This is similar to the numerical solution of IVPs, where convergence was provided by *i*) zero stability and *ii*) consistency.

17.7 Computational considerations: sparsity

Sparsity. One of the most computationally expensive part of solving the BVP by a finite difference method is the solution of the linear system,

$$A\tilde{u} = \tilde{f}.$$

Note that this is an $n \times n$ system, where n could be very large if Δx is very small. In fact in two and three dimensions, the size n is often in millions, if not billions.

While A could be very large, it is important to note that A has a very special structure to it. While A has n^2 entries, most of these n^2 entries are in fact zero; the number of nonzeros in our A for the central difference approximation in one dimension is $3n - 2$ and not n^2. If $n = 1{,}000$, then it has 2,998 nonzeros, not 1,000,000 nonzeros. A matrix whose entries are mostly 0 is called a *sparse matrix*. (There is no rigorous threshold on what "mostly" means.) Our matrix A is a sparse matrix. A matrix that is not sparse is called a *dense matrix*.

Storage. It is important to take advantage of the sparsity of the matrix, both to store the matrix and to solve the linear system. First, sparse matrix can be stored using $\mathcal{O}(n)$ storage by recording, for instance, the indices and values of nonzero entries, i.e., (i, j, a_{ij}). It would be inefficient to store a sparse matrix in the same manner as a regular matrix, i.e. by storing all of $\mathcal{O}(n^2)$ entries; if n is large, the dense format might not even fit in the memory.

Linear solve. We can also take advantage of the sparsity of the matrix to solve the linear system, $A\tilde{u} = \tilde{f}$. For instance, the LU factorization can be modified to only treat nonzero entries; the resulting algorithm, for our particular A, would compute the factors L and U in $\mathcal{O}(n)$ operations, instead of in $\mathcal{O}(n^3)$ operations. The resulting factors L and U are again sparse. The specific algorithm that we can apply to our *tridiagonal matrix* A — which has nonzero entries only along the main diagonal and the two subdiagonals — is called the *Thomas algorithm*.

Even if the matrix is not tridiagonal, we can still take advantage of the sparsity of the matrix to compute sparse factors L and U. However, in general, L and U are not as sparse as A; the number of nonzeros in L and U could be substantially larger than the number of nonzeros in A, though in general the number of nonzeros is still much smaller than n^2. These additional nonzero entries are called *fill-ins*. One important factor that determines the number of fill-ins is the ordering of equations; Matlab's backslash command for instance reorders the equations such that the number of fill-ins is small.

More generally, sparse linear systems are often solved using an *iterative method*, instead of by a direct method (e.g., LU factorization). A class methods that is often used to solve large sparse systems is called *Krylov space methods*. There are also other specialized methods, such as the *multigrid method* suited for Poisson's equation. Unfortunately, their coverage is beyond the scope of this lecture.

Do *not* form A^{-1}. Before we conclude this section, we make one important remark:

if the matrix A is sparse, then never explicitly compute the matrix A^{-1}.

It is important to note that even if A is sparse, A^{-1} is in general dense. For instance, the A matrix associated with our central difference approximation of Poisson's equation is sparse and has $3n-2$ nonzero entries, but the matrix A^{-1} is dense and has n^2 nonzero entries. Forming A^{-1} is inefficient, and, for a large n, the matrix might not even fit in the computer memory. So, we should never form A^{-1}. Instead, if we are using a direct method, we should perform LU factorization such that $A = LU$, where L and U are sparse, and solve two sparse systems $Lz = b$ and $Ux = z$ using forward and backward substitutions, respectively. ("Never explicitly form A^{-1}" applies for dense systems as well, but this is even more true for sparse systems.)

17.8 Numerical example

We consider Poisson's equation

$$
\begin{aligned}
-\frac{d^2u}{dx^2}(x) &= x^2\exp(x) \quad \forall x \in \Omega \equiv (0,1),\\
u(x=0) &= u(x=1) = 0,
\end{aligned}
$$

and solve it using the finite difference method introduced in this lecture. Figure 17.1 shows the result for $\Delta x = 1/16$. We see that the solution varies smoothly in space, is positive everywhere (since f is positive everywhere), and vanishes at the two boundary points. Table 17.1 shows the variation in the maximum error as a function of the grid spacing. We observe that decreasing Δx by a factor of 2 decreases the error by approximately a factor of 4, which is consistent with our expectation for the second-order accurate method.

17.9 Summary

We summarize the key points of this lecture:

1. Poisson's equation can be discretized using the second-order central finite difference formula to yield a difference equation. For any Δx, the associated matrix A is SPD and hence the linear system is well-posed.

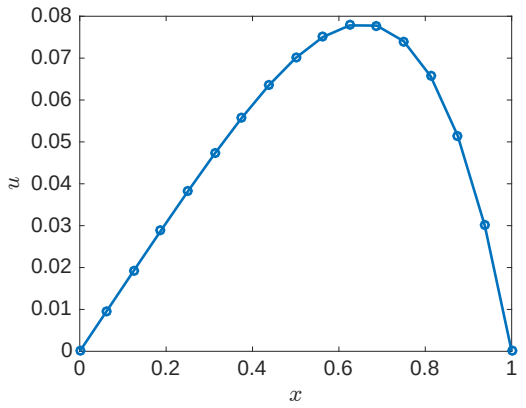


Figure 17.1: Approximation of Poisson's equation using the central finite difference method ($\Delta x = 1/16$).

Δx	$\|u - \tilde{u}\|_\infty$
$1/4$	4.88×10^{-3}
$1/8$	1.26×10^{-3}
$1/16$	3.17×10^{-4}
$1/32$	7.94×10^{-5}
$1/64$	1.98×10^{-5}

Table 17.1: Maximum error for Poisson's equation as a function of the grid spacing.

2. The truncation error associated with the second-order central difference discretization of Poisson's equation is second order: $\|\tau\|_\infty \leq C\Delta x^2$. Moreover, because $\|\tau\|_\infty \to 0$ as $\Delta x \to 0$, the scheme is consistent.

3. The ∞-norm of A^{-1} is bounded; specifically, $\|A^{-1}\|_\infty \leq 1/8$ for any Δx.

4. Convergence of the scheme follows from the consistency and stability. Moreover, scheme is second-order accurate.

5. The matrix A arising from the second-order finite difference discretization of Poisson's equation is sparse and, more specifically, tridiagonal. Sparse systems can be stored efficiency by storing only non-zero entries. The tridiagonal system may be solved using the Thomas algorithm in $\mathcal{O}(n)$ operations (instead of $\mathcal{O}(n^3)$ for dense systems).

6. Even though A is sparse, A^{-1} is dense. Hence, A^{-1} should never be formed. The rule to "never explicitly form A^{-1}" also applies to dense systems, but this is even more true for sparse systems.

Lecture 18

Boundary value problems: generalization

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

18.1 Introduction

In the previous lecture, we studied a finite difference approximation of one-dimensional Poisson's equation. In this lecture, we provide a general framework to solve wider classes of equations and in higher dimensions.

18.2 Finite difference framework for general equations

We abstract a linear differential equation as

$$\mathcal{L}u = f \quad \text{in } \Omega,$$

where $u \in C^m(\Omega)$ is the solution, $\mathcal{L} : C^m(\Omega) \to C^{m-k}(\Omega)$ is the differential operator, and $f \in C^{m-k}(\Omega)$ is the source term. We assume that boundary conditions are embedded in the definition of u such that the equation is well-posed. For instance, for Poisson's equation with homogeneous Dirichlet boundary conditions, $\mathcal{L} \equiv \frac{d^2}{dx^2}$ and $u(x=0) = u(x=1) = 0$.

To approximate the solution of the BVP, we first discretize the domain Ω into $n+1$ intervals delineated by grid points $\{x_i\}_{i=0}^n$. Then, we approximate the differential operator $\mathcal{L}$ with a finite difference formula to obtain a difference scheme

$$\tilde{\mathcal{L}}\tilde{u} = \tilde{f} \quad \text{in } \mathbb{R}^n,$$

where $\tilde{u} \in \mathbb{R}^n$ is our approximation of u at the grid points, $\tilde{\mathcal{L}} : \mathbb{R}^n \to \mathbb{R}^n$ is the difference operator, and $\tilde{f} \in \mathbb{R}^n$ is f evaluated at the grid points. We hope that $u(x_i) \approx \tilde{u}_i$, $i = 1, \ldots, n$.

Truncation error and consistency. Similar to our analysis of Poisson's equation, one of the two required ingredients for convergence is consistency. (The other is stability.) As before, we define the truncation error as the remainder that result from the substitution of the exact solution to the difference equation:

$$\tau \equiv \tilde{L}u(\{x_i\}) - \tilde{f} \in \mathbb{R}^n,$$

where $u(\{x_i\}) \in \mathbb{R}^n$ is the vector whose i-th entry is $u(x_i)$. We say a scheme is consistent if

$$\|\tau\|_\infty \to 0 \quad \text{as } \Delta x \to 0.$$

Moreover, if

$$\|\tau\|_\infty \leq C_\tau \Delta x^p \quad \text{as } \Delta x \to 0,$$

then the truncation error is p-th order.

Stability. For a scheme to be convergent, it must also be stable. As before, we define stability in ∞-norm. Specifically, a scheme is uniformly stable if, there exists a constant C_s such that

$$\|\tilde{\mathcal{L}}^{-1}\|_\infty \equiv \max_{v \in \mathbb{R}^n} \frac{\|\mathcal{L}^{-1} v\|_\infty}{\|v\|_\infty} \leq C_s \quad \text{for all } \Delta x.$$

This condition, for a general finite difference scheme, is often difficult to prove. A weaker condition is that a scheme is *asymptotically* stable: there exists a constant C_s such that

$$\|\tilde{\mathcal{L}}^{-1}\|_\infty \leq C_s \quad \text{for } \Delta x < \Delta x^\star,$$

where $\Delta x^\star$ is some critical grid spacing.

Convergence. To prove convergence, we first relate the solution error to the truncation error. Towards this end, we subtract our difference scheme $\tilde{L}\tilde{u} = \tilde{f}$ from the definition of the truncation error $\tilde{L}u(\{x_i\}) = \tilde{f} + \tau$ to obtain the error equation

$$\tilde{L}e = \tau,$$

where $e_i \equiv u(x_i) - \tilde{u}_i$, $i = 1, \ldots, n$. It also follows that $e = \tilde{\mathcal{L}}^{-1}\tau$. We then note that

$$\|e\|_\infty = \|\tilde{\mathcal{L}}^{-1}\tau\|_\infty \leq \|\tilde{\mathcal{L}}^{-1}\|_\infty \|\tau\|_\infty \leq C_s C_\tau \Delta x^p \to 0 \quad \text{as } \Delta x \to 0.$$

We conclude that the scheme is convergent and it is p-th order accurate. Hence, we again have the relationship

$$\text{(consistency)} + \text{(stability)} \Rightarrow \text{(convergence)},$$

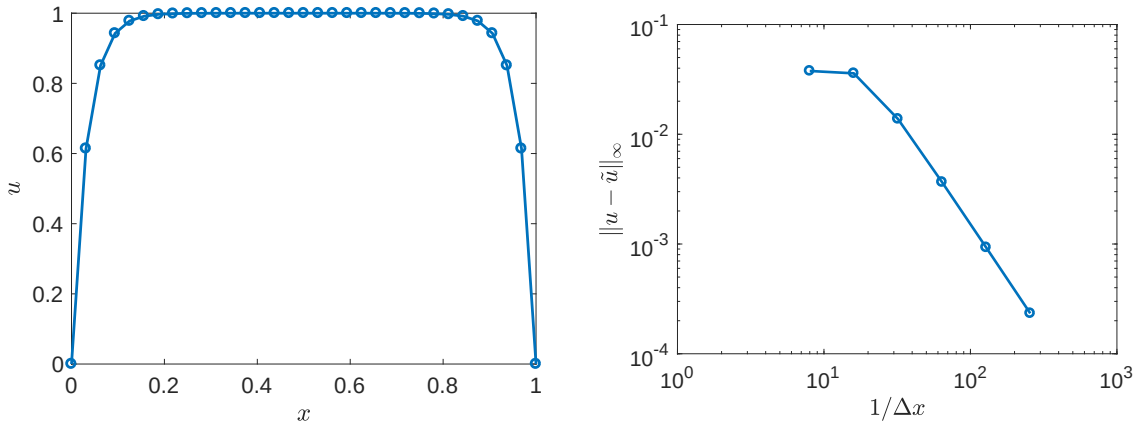
which we have seen several times in this course.

18.3 Example 1. Reaction-diffusion equation

Reaction-diffusion equation is a model equation for reactive processes, such as combustion and more general chemical reactions. The model equation is given by

$$\begin{aligned} -\epsilon \frac{d^2 u}{dx^2} + u &= f \quad \text{in } \Omega = (0,1), \\ u(x=0) = u(x=1) &= 0, \end{aligned}$$

where $\epsilon \in \mathbb{R}_{>0}$ is the diffusion coefficient and f is the source term. Without loss of generality, we take the reaction coefficient to be unity; when the coefficient is not unity, the equation can be scaled to provide an equivalent problem with a unity reaction coefficient and modified ϵ and f.



(a) approximation ($\Delta x = 1/32$)

(b) convergence

Figure 18.1: Solution of the reaction-diffusion equation for $\nu = 0.001$ and $f = 1$.

Finite-difference formulation. We discretize the operator d^2/dx^2 using the central difference formula and obtain the following difference equation:

$$-\epsilon \frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2} + \tilde{u}_i = \tilde{f}_i, \quad i = 1, \ldots, n,$$
$$\tilde{u}_{i=0} = \tilde{u}_{i=1} = 0.$$

The associated matrix form of the equation is

$$(\epsilon A + I)\tilde{u}(\{x_i\}) = \tilde{f},$$

where $A \in \mathbb{R}^{n \times n}$ is the finite difference matrix associated with the negative of the Laplacian operator introduced in the previous lecture. Because A is SPD, we readily observe that $(\epsilon A + I)$ is SPD. Moreover, the system is tridiagonal.

Consistency. With the central difference formula, we can readily show that

$$\|\tau\|_\infty \equiv \|(\epsilon A + I)u - \tilde{f}\|_\infty \leq C_\tau \Delta x^2 \quad \text{as } \Delta x \to 0.$$

The scheme is consistent, and the truncation error is second order.

Stability. Proving the stability of the reaction-diffusion system is beyond the scope of this lecture. We here just state that it is possible to show that, for any $\epsilon \in \mathbb{R}_{>0}$,

$$(\epsilon A + I)^{-1} \leq 1 \quad \text{for } \Delta x \text{ sufficiently small.}$$

Convergence. Because the scheme is consistent and stable, it is convergent. The order of accuracy of the scheme is two because the scheme is stable and $\|\tau\|_\infty \leq C_\tau \Delta x^2$.

Example. An example solution of the reaction-diffusion equation for $\nu = 0.001$ and $f = 1$ is shown in Figure 18.1. For the small diffusion coefficient, we observe boundary layers of thickness $\mathcal{O}(\sqrt{\nu})$. The scheme exhibit second-order convergence as expected.

18.4 Example 2. Convection-diffusion equation

Convection-diffusion equation is a model equation for transport processes, including flows governed by the Navier-Stokes equations. The model equation is given by

$$\begin{aligned} -\epsilon \frac{d^2u}{dx^2} + \frac{du}{dx} &= f \quad \text{in } \Omega = (0,1), \\ u(x=0) = u(x=1) &= 0, \end{aligned}$$

where $\epsilon \in \mathbb{R}_{>0}$ is the diffusion coefficient and f is the source term. Without loss of generality, we take the convection coefficient to be unity; when the coefficient is not unity, the equation can be scaled to provide an equivalent problem with a unity convection coefficient and modified ϵ and f.

Finite-difference formulation. We discrete the operators d^2/dx^2 and d/dx using the central difference formulas for second and first derivatives, respectively, to obtain

$$\begin{aligned} -\epsilon \frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2} + \frac{\tilde{u}_{i+1} - \tilde{u}_{i-1}}{2\Delta x} &= \tilde{f}_i, \quad i = 1, \ldots, n, \\ \tilde{u}_{i=0} = \tilde{u}_{i=1} &= 0. \end{aligned}$$

The associated matrix form of the equation is

$$(\epsilon A + B)\tilde{u} = \tilde{f},$$

where $A \in \mathbb{R}^{n \times n}$ and $B \in \mathbb{R}^{n \times n}$ are the finite difference matrices associated with the negative of the second-derivative and the first-derivative, respectively. The system is not symmetric, but it is tridiagonal.

Consistency. With the second-order central difference formulas for both the first and second derivatives, we can readily show that $\|\tau\|_\infty \leq C_\tau \Delta x^2$ as $\Delta x \to 0$.

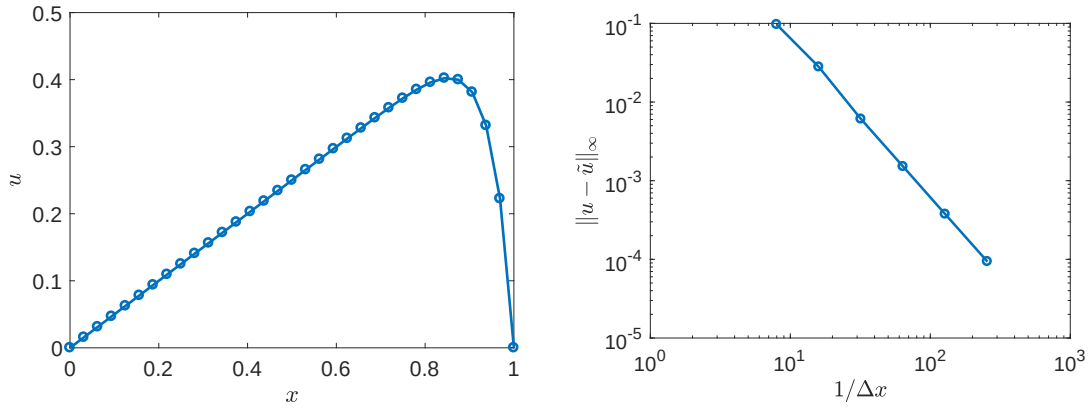
Stability. Proving the stability of the convection-diffusion system is beyond the scope of this lecture. We here just state that it is possible to show that, for any $\epsilon \in \mathbb{R}_{>0}$, the stability constant is bounded for Δx sufficiently small.

Warning. The particular discretization considered — which is based on the centered difference approximation of the first derivative — yields solutions that exhibit spurious oscillations when the grid spacing is larger than the length scale of the boundary layer: $\Delta x \gtrsim \epsilon$. The discussion of this phenomena is beyond the scope of this lecture; we only note that the issue can be overcome by adding grid-dependent "stabilization terms" or incorporating the idea of "upwinding".

Convergence. Because the scheme is consistent and stable, it is convergent. The order of accuracy of the scheme is two because $\|\tau\|_\infty \leq C_\tau \Delta x^2$.

Example. An example of solving a convection-diffusion equation is shown in Figure 18.1. For the small diffusion coefficient of $\nu = 0.1$, we observe boundary layers of thickness $\mathcal{O}(\nu)$. The scheme exhibit second-order convergence as expected.

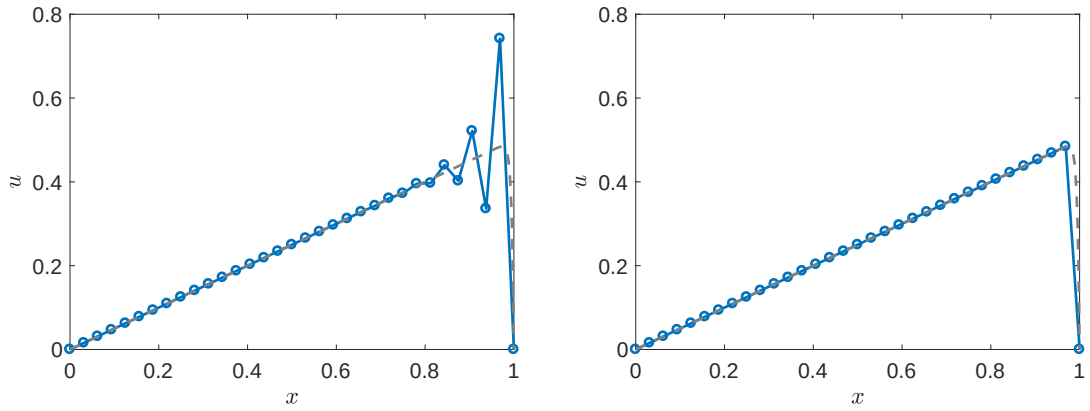
We also consider the case where the solution is underresolved in the sense that $\epsilon = 0.01 < \Delta x = 1/32$. Figure 18.3(a) shows that the scheme discussed above — based on centered difference of the convection term — is unstable and exhibits spurious oscillation in the vicinity of the boundary layer. Figure 18.3(b) shows that, with stabilization, the solution no longer exhibit oscillation, even though the boundary layer is underresolved because $\Delta x > \epsilon$. A simple stabilization used in this example is based on the idea of *artificial diffusion*; we modify the effective viscosity of the problem such that $\epsilon_{\text{eff}} = \min\{\epsilon, \Delta x\}$. The added artificial diffusion stabilizes the scheme when the boundary



(a) approximation ($\Delta x = 1/32$)

(b) convergence

Figure 18.2: Convection-diffusion equation for $\nu = 0.1$ and $f = 1$.



(a) no stabilization

(b) with stabilization

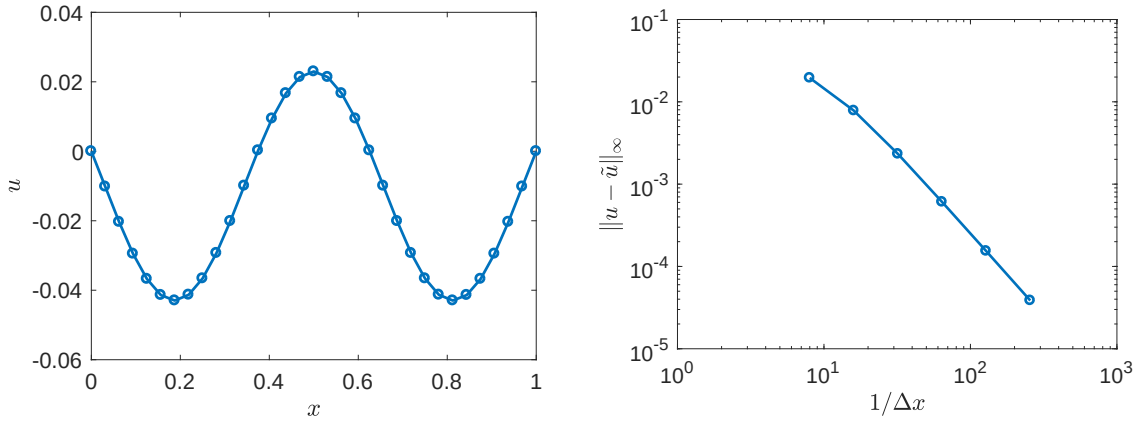
Figure 18.3: Convection-diffusion equation for $\epsilon = 0.01$, $f = 1$, and $\Delta x = 1/32$.

layer is underresolved. But, when the boundary layer is resolved, the artificial diffusion is removed from the scheme; i.e., we revert back to the original scheme discussed above, which works well for $\Delta x < \epsilon$.

18.5 Example 3. Helmholtz equation

Helmholtz equation is a model equation for the frequency-domain analysis of wave phenomena, including acoustics and elastodynamics. The model equation is given by

$$-\frac{d^2u}{dx^2} - k^2 u = f \quad \text{in } \Omega \equiv (0,1),$$
$$u(x=0) = u(x=1) = 0,$$



(a) approximation ($\Delta x = 1/32$)

(b) convergence

Figure 18.4: Helmholtz equation for $k = 10$ and $f = 1$.

where $k \in \mathbb{R}_{\geq 0}$ is the wave number and f is the source term. The Helmholtz equation is not well-posed when $k = i\pi$ for some integer i; physically, these values of k corresponds to the resonance frequencies. Hence, we require $k \neq i\pi$ for any integer i.

Finite-difference formulation. We discretize the operator d^2/dx^2 using the central difference formulas to obtain the following difference equation:

$$-\frac{\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1}}{\Delta x^2} - k^2\tilde{u}_i = \tilde{f}_i, \quad i = 1, \ldots, n,$$
$$\tilde{u}_{i=0} = \tilde{u}_{i=1} = 0.$$

The associated matrix form of the equation is

$$(A - k^2 I)\tilde{u} = \tilde{f},$$

where $A \in \mathbb{R}^{n\times n}$ is the finite difference matrix associated with the (negative of the) Laplacian operator. The matrix $(A - k^2 I) \in \mathbb{R}^{n\times n}$ is symmetric and tridigonal; however, it is not positive definite for a large k. Moreover, the equation is singular when k^2 is an eigenvalue of A. (We will study eigenproblems in the next lecture.)

Consistency. With the second-order central difference formulas for both the first and second derivatives, we can readily show that $\|\tau\|_\infty \leq C_\tau \Delta x^2$ as $\Delta x \to 0$.

Stability. Proving the stability of the Helmholtz system is beyond the scope of this lecture. We here only state that the stability constant C_τ is dependent on the wave number k, and it is not bounded when $k = i\pi$ for some integer i.

Convergence. Because the scheme is consistent and stable, it is convergent. The order of accuracy of the scheme is two because $\|\tau\|_\infty \leq C_\tau \Delta x^2$.

Example. An example of solving the Helmholtz equation for $k = 10$ and $f = 1$ is shown in Figure 18.4. The solution is oscillatory. The scheme exhibit second-order convergence as expected.

18.6 Higher dimensions: Poisson's equation in two dimensions

We now consider Poisson's equation in two dimensions,

$$\begin{aligned} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} &= f \quad \text{in } \Omega \equiv (0,1) \times (0,1), \\ u(x \in \partial\Omega) &= 0. \end{aligned}$$

Here, Ω is the physical domain of interest, u is the state, $f \in C^0(\Omega)$ is the forcing function, and $u(x \in \partial\Omega) = 0$ is the boundary condition. Note that for simplicity we consider the unit square domain and homogeneous Dirichlet boundary condition.

Finite-difference formulation. We now consider the approximation of the two-dimensional Poisson's equation by a finite difference method. We first introduce a computational grid over the domain Ω. The grid points are given by

$$(x_i, y_j) = (i\Delta x, j\Delta y), \quad i, j = 0, \ldots, n+1,$$

where $\Delta x = \Delta y = h = 1/(n+1)$ is the spacing between any two grid points. We then denote our approximation of the solution by

$$\tilde{u}_{i,j} \equiv \tilde{u}(x_{i,j}), \quad i, j = 0, \ldots, n+1.$$

We now apply the central difference formula to approximate the second derivatives $\frac{\partial^2 u}{\partial x^2}$ and $\frac{\partial^2 u}{\partial y^2}$,

$$\left.\frac{\partial^2 u}{\partial x^2}\right|_{i,j} + \left.\frac{\partial^2 u}{\partial y^2}\right|_{i,j} \approx \frac{\tilde{u}_{i+1,j} - 2\tilde{u}_{i,j} + \tilde{u}_{i-1,j}}{\Delta x^2} + \frac{\tilde{u}_{i,j+1} - 2\tilde{u}_{i,j} + \tilde{u}_{i,j-1}}{\Delta y^2};$$

because the approximation of the Laplacian at (i, j) involves the solution values at five points (i.e. four neighbors and itself), the finite difference approximation is said to have a *five-point stencil*. The substitution of the finite difference formula to Poisson's equation yields

$$-\frac{\tilde{u}_{i+1,j} - 2\tilde{u}_{i,j} + \tilde{u}_{i-1,j}}{\Delta x^2} - \frac{\tilde{u}_{i,j+1} - 2\tilde{u}_{i,j} + \tilde{u}_{i,j-1}}{\Delta y^2} = \tilde{f}_{i,j}, \quad i, j = 1, \ldots, n, \tag{18.1}$$

with boundary conditions

$$\begin{aligned} \tilde{u}_{0,j} = \tilde{u}_{n+1,j} &= 0, \quad j = 0, \ldots, n+1, \\ \tilde{u}_{i,0} = \tilde{u}_{i,n+1} &= 0, \quad i = 1, \ldots, n, \end{aligned}$$

where $\tilde{f}_{i,j} = f(x_{i,j})$. Note that we have n^2 equations associated with the grid points. The difference equation can again be expressed in a matrix form:

$$A\tilde{u} = \tilde{f};$$

for instance, if $n = 3$, the vectors $\tilde{u}$ and $\tilde{f}$ are given by

$$\tilde{u} = \begin{pmatrix} \tilde{u}_{11} \\ \tilde{u}_{21} \\ \tilde{u}_{31} \\ \hline \tilde{u}_{12} \\ \tilde{u}_{22} \\ \tilde{u}_{32} \\ \hline \tilde{u}_{13} \\ \tilde{u}_{23} \\ \tilde{u}_{33} \end{pmatrix}, \qquad \tilde{f} = \begin{pmatrix} \tilde{f}_{11} \\ \tilde{f}_{21} \\ \tilde{f}_{31} \\ \hline \tilde{f}_{12} \\ \tilde{f}_{22} \\ \tilde{f}_{32} \\ \hline \tilde{f}_{13} \\ \tilde{f}_{23} \\ \tilde{f}_{33} \end{pmatrix}$$

and the matrix A is given by

$$A = \frac{1}{h^2} \left(\begin{array}{ccc|ccc|ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array} \right).$$

Note that we have ordered the grid points such that x-index is the faster changing index and y-index is the slower changing index.

Consistency. Using exactly the same procedure we have used for the one-dimensional problem, we can readily prove the convergence of the finite difference method in two dimensions.

First, using Taylor series expansion, we can shows that the the truncation error at the grid point (i, j) is

$$\tau_{i,j} = \underbrace{-\frac{\partial^2 u}{\partial x^2}(x_{i,j}) - \frac{\partial^2 u}{\partial y^2}(x_{i,j}) - f(x_i)}_{=0 \text{ by PDE}} - \frac{1}{12}\frac{\partial^4 u}{\partial x^4}(\xi_{i,j})\Delta x^2 - \frac{1}{12}\frac{\partial^4 u}{\partial y^4}(\eta_{i,j})\Delta y^2 = \mathcal{O}(\Delta x^2, \Delta y^2).$$

It follows that

$$\|\tau\|_\infty = \mathcal{O}(\Delta x^2, \Delta y^2).$$

Hence, the method is consistent.

Stability. The matrix A associated with our two-dimensional finite difference approximation also satisfies the two properties:

1. All entries of A^{-1} are positive;

2. The infinity norm of A^{-1} is bounded: $\|A^{-1}\|_\infty \leq \frac{1}{8}$.

It follows that the finite difference approximation is stable in the sense that $\|\tilde{u}\|_\infty \leq \frac{1}{8}\|\tilde{f}\|_\infty$.

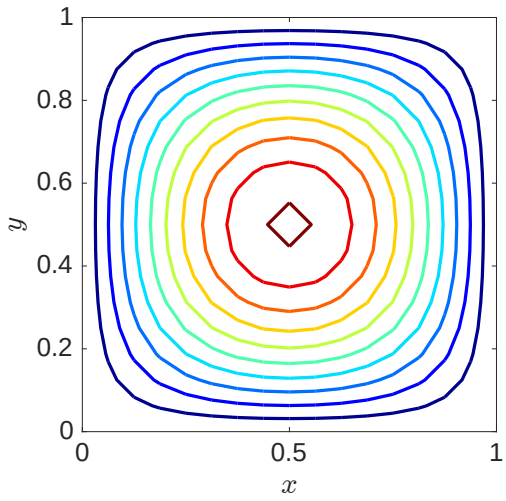


Figure 18.5: Approximation of the two-dimensional Poisson's problem using the central finite difference method ($\Delta x = \Delta y = 1/16$).

$\Delta x = \Delta y$	$\|u - \tilde{u}\|_\infty$
1/4	2.69×10^{-3}
1/8	6.56×10^{-4}
1/16	1.63×10^{-4}
1/32	4.07×10^{-5}
1/64	1.02×10^{-5}

Table 18.1: Maximum error for the two-dimensional Poisson's problem as a function of the grid spacing.

Convergence. To prove convergence, we again subtract the finite difference equation from the equation for the truncation error to obtain the error equation, $Ae = \tau$. By appealing to the stability of A and the bound of the truncation error, we obtain

$$\|e\|_\infty \leq \frac{1}{8}\|\tau\|_\infty = \mathcal{O}(\Delta x^2, \Delta y^2).$$

We observe that the scheme is convergent and is second-order accurate.

Example. We consider a two-dimensional Poisson's problem for $f = \sin(\pi x)\sin(\pi y)$. The second-order central finite difference approximation for $\Delta x = \Delta y = 16$ is shown in Figure 18.5. Note that the solution varies smoothly in space. Table 18.1 shows the maximum error over all grid points as a function of the grid spacing; we observe that the maximum error scales as $\mathcal{O}(\Delta x^2, \Delta y^2)$, which we is consistent with our expectation for the second-order method.

18.7 Treatment of various boundary conditions

We have so far considered homogeneous Dirichlet boundary conditions. We now consider the treatment of more general boundary conditions. To provide a concrete example, we consider the

two-dimensional Poisson's problem in Section 18.6; however, the procedure readily generalizes to other dimensions and equations.

1. *Dirichlet boundary.* Suppose we wish to impose a nonhomogeneous Dirichlet boundary condition on the $x = 0$ boundary:

$$u(x_{i=0}, y_j) = h(y_j).$$

This boundary condition is incorporated in the finite difference equation associated with $(x_{i=1}, y_j)$; i.e., one point inside the boundary. We substitute $u_{i=0,j} = h(y_j)$ to (18.1) evaluated at $(x_{i=1}, y_j)$,

$$-\frac{\tilde{u}_{2,j} - 2\tilde{u}_{1,j} + \tilde{u}_{0,j}}{\Delta x^2} - \frac{\tilde{u}_{1,j+1} - 2\tilde{u}_{1,j} + \tilde{u}_{1,j-1}}{\Delta y^2} = \hat{f}_{1,j},$$

to obtain

$$-\frac{\tilde{u}_{2,j} - 2\tilde{u}_{1,j} + h(y_j)}{\Delta x^2} - \frac{\tilde{u}_{1,j+1} - 2\tilde{u}_{1,j} + \tilde{u}_{1,j-1}}{\Delta y^2} = \hat{f}_{1,j}.$$

We move all known terms to the right hand side to obtain

$$-\frac{\tilde{u}_{2,j} - 2\tilde{u}_{1,j}}{\Delta x^2} - \frac{\tilde{u}_{1,j+1} - 2\tilde{u}_{1,j} + \tilde{u}_{1,j-1}}{\Delta y^2} = \hat{f}_{1,j} + \frac{h(y_j)}{\Delta x^2}.$$

Note that we have moved the "known" value to the right hand side of the equation so that the left hand side is linear in $\tilde{u}$.

2. *Neumann boundary.* Suppose we wish to impose a nonhomogeneous Neumann boundary condition on the $x = 0$ boundary:

$$-\frac{\partial u}{\partial x}(x_{i=0}, y_j) = h(y_j).$$

This boundary condition is incorporated in the finite difference equation associated with $(x_{i=0}, y_j)$. Note that, unlike the case of a Dirichlet boundary condition, the solution on the boundary $u(x_{i=0}, y_j)$ is unknown for a Neumann boundary. The finite difference approximation of the boundary condition is

$$-\frac{\tilde{u}_{1,j} - \tilde{u}_{-1,j}}{2\Delta x} = h(y_j),$$

which implies $\tilde{u}_{-1,j} = \tilde{u}_{1,j} + 2\Delta x h(y_j)$. We substitute the (approximation of the) boundary condition to (18.1) evaluated at $(x_{i=1}, y_j)$,

$$-\frac{\tilde{u}_{1,j} - 2\tilde{u}_{0,j} + \tilde{u}_{-1,j}}{\Delta x^2} - \frac{\tilde{u}_{0,j+1} - 2\tilde{u}_{0,j} + \tilde{u}_{0,j-1}}{\Delta y^2} = \hat{f}_{0,j},$$

to obtain

$$-\frac{\tilde{u}_{1,j} - 2\tilde{u}_{0,j} + \tilde{u}_{1,j} + 2\Delta x h(y_j)}{\Delta x^2} - \frac{\tilde{u}_{0,j+1} - 2\tilde{u}_{0,j} + \tilde{u}_{0,j-1}}{\Delta y^2} = \hat{f}_{0,j}.$$

We again move known quantities to the right hand side to obtain

$$-\frac{2\tilde{u}_{1,j} - 2\tilde{u}_{0,j}}{\Delta x^2} - \frac{\tilde{u}_{0,j+1} - 2\tilde{u}_{0,j} + \tilde{u}_{0,j-1}}{\Delta y^2} = \hat{f}_{0,j} + \frac{2h(y_j)}{\Delta x}.$$

We have again moved the term associated with the boundary data h to the right hand side of the equation so that the left hand side is linear in $\tilde{u}$.

We make a few remarks.

1. On a Dirichlet boundary, the solution value $\tilde{u}_{0,j}$ is *known* and becomes the right hand side $h(y_j)$. On a Neumann boundary, the solution value $\tilde{u}_{0,j}$ is *unknown* and stays on the left hand side.

2. The above treatment of the boundary conditions ensures that the truncation error is second-order accurate.

3. Assuming not all boundary is Neumann boundary, it can be shown that $\|A^{-1}\|_\infty$ is bounded, and hence the method is stable. (The continuous problem associated with fully Neumann boundary conditions is not a well posed because the solution can "float"; i.e., the solution can only be determined up to a constant. Our finite difference approximation inherits this property of the continuous problem in the fully Neumann case.)

4. Since the truncation error is second-order accurate and the method is stable, the finite difference approximation is convergent and is second-order accurate.

18.8 Summary

We summarize the key points of this lecture:

1. A general boundary value problem, $\mathcal{L}u = f$ with appropriate boundary conditions, can be approximated by applying a finite-difference formula to the differential operator $\mathcal{L}$. The resulting (discrete) equation is of the form $\tilde{\mathcal{L}}\tilde{u} = \tilde{f}$.

2. In order to prove a finite difference approximation is convergent, we need to show that the scheme is consistent — $\|\tau\|_\infty \equiv \|\tilde{\mathcal{L}}u(\{x_i\}) - \tilde{f}\|_\infty \to 0$ as $\Delta x \to 0$ — and that the inverse of the difference operator is bounded — $\|\mathcal{L}^{-1}\|_\infty \leq C_s$.

3. If the scheme is stable and $\|\tau\|_\infty = \mathcal{O}(\Delta x^p)$, then the scheme is p-th order accurate in the sense that $\|u(\{x_i\}) - \tilde{u}\|_\infty \leq C\Delta x^p$ as $\Delta x \to 0$.

4. Canonical BVPs — including the reaction-diffusion equation, convection-diffusion equation, and Helmholtz equation — can be approximated using the finite difference method. However, there are subtle issues — especially regarding the stability — for some equations.

5. Finite difference method can be extended to higher dimensions by applying the finite difference formula to the differential operator in each coordinate direction.

6. Finite difference method can readily treat nonhomogeneous Dirichlet and Neumann boundary conditions.

Lecture 19

Boundary value problems: eigenproblems

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

19.1 Introduction

Eigenproblems are ubiquitous in engineering. One example is the identification of resonance frequencies in structural analysis, which requires eigenvalues of the linear elasticity equations. Another example is the hydrodynamic stability analysis, which requires the eigenvalues of the (linearized) Navier-Stokes equations. In essentially all but simplest cases, the analytical solution is not known and the eigenproblems must be solved numerically. In this lecture, we consider numerical approximation of the eigenproblem associated with the Laplacian operator.

19.2 Model problem

A model eigenproblem we consider is the following: find (non-trivial) eigenpairs $(u^k, \lambda^k) \in C^m(\Omega) \times \mathbb{R}$ such that

$$\begin{aligned} -\frac{d^2u^k}{dx^2} &= \lambda^k u^k \quad \text{in } \Omega \equiv (0,1), \\ u^k(x=0) &= 0, \\ u^k(x=1) &= 0, \end{aligned}$$

for $k = 1, 2, \ldots$. Without loss of generality, we order the eigenpairs such that

$$\lambda^1 \leq \lambda^2 \leq \ldots.$$

Exact solutions to this eigenproblem are given by

$$\begin{aligned} u^k(x) &= \sin(\sqrt{\lambda^k}x), \\ \lambda^k &= k^2\pi^2, \end{aligned}$$

for $k = 1, 2, \ldots$; we can readily verify the result through a direct substitution. We recall that u^k is unique only up to scaling. We make a few observations:

1. Eigenvalues are positive; $\lambda^k > 0$, $\forall k$.
2. Eigenvectors $\{u^k\}$ are ortohgonal in the sense that
$$\int_0^1 u^k(x)u^l(x)dx = \delta_{kl} = \begin{cases} 1, & k = l \\ 0, & k \neq l \end{cases}$$
where δ_{kl} is the Kronecker delta. (We here assume an appropriate scaling of $\tilde{u}^k$.)
3. There are infinite number of eigenpairs. The minimum eigenvalue is bounded but the maximum eigenvalue is not.
4. Eigenvectors are more oscillatory for a higher k; we intuitively anticipate that the numerical approximation of high modes are more difficult than low modes.

The positiveness and orthogonality are consequences of the Laplacian operator being self-adjoint (i.e., "symmetric") and coercive (i.e., "positive definite").

19.3 Finite difference method

Following the recipe we introduced to solve BVPs, we first introduce a computational grid over the domain $\Omega \equiv [0, 1]$. The grid points are given by

$$x_i = i\Delta x, i = 0, \ldots, n + 1,$$

where $\Delta x \equiv 1/(n + 1)$ is the grid spacing. We then approximate the second derivative with a central difference formula to obtain the difference equation: find eigenpairs $(\tilde{u}^k, \tilde{\lambda}^k) \in \mathbb{R}^n \times \mathbb{R}$ such that

$$-\frac{\tilde{u}^k_{i+1} - 2\tilde{u}^k_i + \tilde{u}^k_{i-1}}{\Delta x^2} = \tilde{\lambda}^k \tilde{u}^k_i, \quad i = 1, \ldots, n,$$

with boundary conditions

$$\tilde{u}_{i=0} = \tilde{u}_{i=n+1} = 0.$$

The matrix form of the equation is

$$A\tilde{u}^k = \tilde{\lambda}^k \tilde{u}^k, \quad k = 1, 2, \ldots$$

where

$$A = \frac{1}{\Delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & -1 & 2 & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix} \in \mathbb{R}^{n \times n},$$

which is symmetric positive definite (SPD) as shown in the previous lecture. Our hope is that $\tilde{\lambda}^k \approx \lambda^k$ and $\tilde{u}^k \approx u^k(\{x_i\})$. (We recall that $u^k(\{x_i\})$ is our shorthand for a n vector whose i-th entry is $u^k(x_i)$.) Before we proceed with the error analysis, we make a few observations about the difference equation:

1. The eigenvalues are real and positive (since A is SPD).

2. Eigenvectors are orthogonal in the sense that $(v^k)^T(v^l) = \delta_{kl}$ (since A is symmetric).

3. We have n eigenvalues, though they may not be distinct.

The last observation implies that we can expect only a small subset of the eigenvalues of the original differential operator to be approximated, as the differential operator has infinite number of eigenvalues.

19.4 Error analysis

It turns out we can find the eigenpairs of the matrix $A \in \mathbb{R}^{n\times n}$ in closed form. Specifically, the n eigenpairs are given by

$$\begin{aligned}\tilde{u}_i^k &= \sin(k\pi x_i), \quad i = 1, \ldots, n,\\ \tilde{\lambda}^k &= \frac{2}{\Delta x^2}(1 - \cos(k\pi\Delta x)),\end{aligned}$$

for $k = 1, \ldots, n$. We can readily verify that $(\tilde{u}^k, \tilde{\lambda}^k)$ is an eigenpair through a direct substitution to the difference equation: the i-th row of $A\tilde{u}^k = \tilde{\lambda}^k\tilde{u}^k$ is

$$\begin{aligned}(A\tilde{u}^k)_i &= \frac{1}{\Delta x^2}(-\tilde{u}_{i+1}^k + 2\tilde{u}_i^k - \tilde{u}_{i-1}^k) = \frac{1}{\Delta x^2}(-\sin(k\pi(x_i + \Delta x)) + 2\sin(k\pi x_i) - \sin(k\pi(x_i - \Delta x)))\\ &= \frac{1}{\Delta x^2}(-2\sin(k\pi x_i)\cos(k\pi\Delta x) + 2\sin(k\pi x_i)) = \underbrace{\frac{2}{\Delta x^2}(1 - \cos(k\pi\Delta x))}_{\tilde{\lambda}^k}\underbrace{\sin(k\pi x_i)}_{\tilde{u}_i^k} = \tilde{\lambda}^k\tilde{u}_i^k.\end{aligned}$$

We also note that $\tilde{u}^k$ satisfies the boundary conditions: $\tilde{u}_{i=0}^k = \tilde{u}_{i=n+1}^k = 0$.

We now analyze the error in our eigenvalue approximation. For $k\pi\Delta x \ll 1$, the eigenvalues are

$$\begin{aligned}\tilde{\lambda}^k &= \frac{2}{\Delta x^2}[1 - (1 - \frac{k^2\pi^2\Delta x^2}{2} + \frac{k^4\pi^4\Delta x^4}{24} + \mathcal{O}(k^6\pi^6\Delta x^6))]\\ &= k^2\pi^2 - \frac{k^4\pi^4\Delta x^2}{12} + \mathcal{O}(k^6\pi^6\Delta x^4).\end{aligned}$$

It follows that, for $k\pi\Delta x \ll 1$, the *relative* error in the eigenvalues is

$$\frac{|\lambda^k - \tilde{\lambda}^k|}{|\lambda^k|} = \frac{k^2\pi^2\Delta x^2}{12} + \mathcal{O}(k^4\pi^4\Delta x^4).$$

We observe that our finite difference method approximates the true eigenvalues of low-modes (i.e. for k small in the sense that $k\pi\Delta x \ll 1$). More precisely, the approximate eigenvalues are second-order accurate: there exists a constant C such that $|\lambda^k - \tilde{\lambda}^k|/|\lambda^k| \le Ck^2\Delta x^2$.

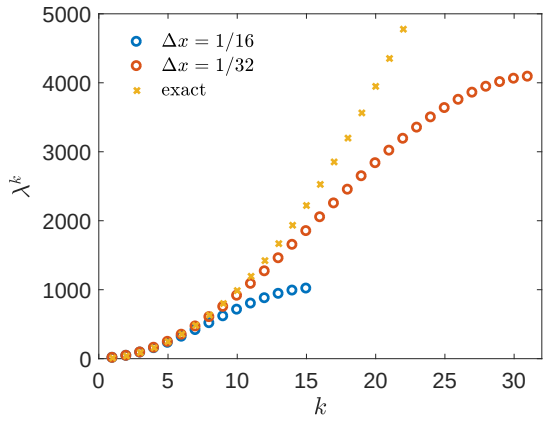


Figure 19.1: Approximation of eigenvalues for $\Delta x = 1/16$ and $\Delta x = 1/32$

19.5 Numerical results

We now solve the eigenproblem associated with the Laplacian operator over $\Omega \equiv (0, 1)$ using the finite difference method. Figure 19.1 illustrates how the range of eigenvalues that are well approximated changes with the grid spacing Δx. For a small value of $\Delta x = 1/16$, only the first several modes are approximated well; note that this is consistent with the theory which predicts the error to be reasonably small for $k\pi \lesssim 1/\Delta x$. We also observe that reducing the grid spacing by a factor of two to $\Delta x = 1/32$ roughly doubles the number of eigenvalues that are well approximated.

We now assess how well eigenvectors are approximated. Figures 19.2(a)–(c) show that the eigenmodes i) are exact at the evaluation points $\{x_i\}_{i=1}^n$ and ii) are well-represented in the sense that we can readily approximate the true eigenmodes, which are continuous functions, by (say) interpolating the values at the evaluation points. The exactness of the eigenmodes evaluated at $\{x_i\}_{i=1}^n$ is expected from the theory.

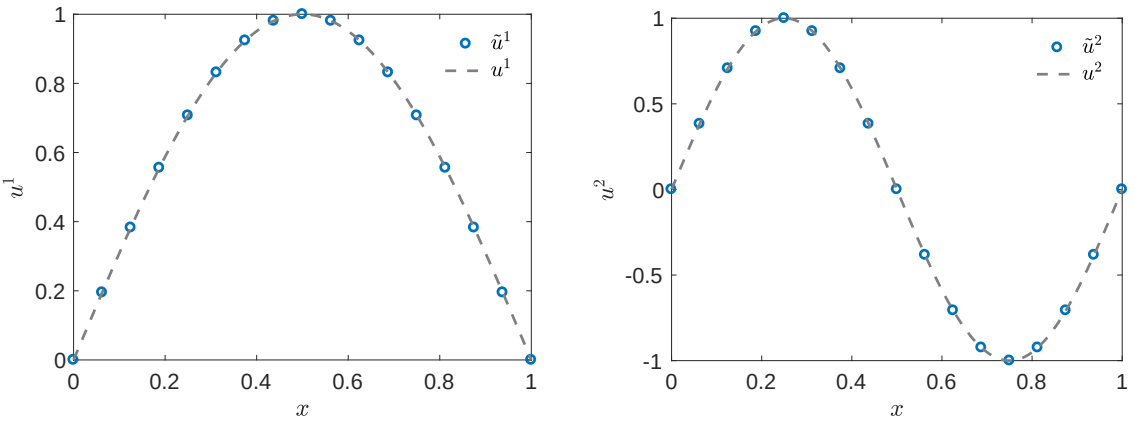
On the other hand, we observe in Figure 19.2(d) that the high-modes i) are exact at the evaluation points $\{x_i\}_{i=1}^n$ but ii) are *not* well-represented in the sense that we would not be able to approximate the eigenmodes by interpolating the values at the evaluation points. (Note that even though we show the true eigenmode u^{15} in the figure for the purpose of assessment, this solution would not be known in practice; otherwise, there is no reason to approximate the solution numerically.) Specifically, we suffer from *aliasing* because our grid spacing is too large to resolve the true eigenmode. In order to accurately resolve this mode, the grid spacing must be decreased.

19.6 Generalization: other equations and higher dimensions

As discussed in the introduction, eigenproblems are ubiquitous in engineering. The technique discussed in this lecture in the context of the Laplacian operator can be generalized to other equations; a general eigenproblem is of the form

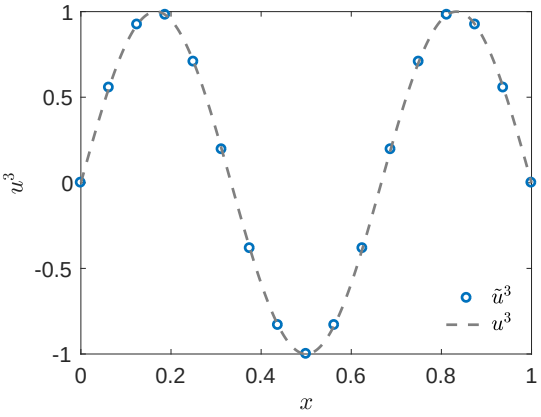
$$\mathcal{L}u = \lambda u,$$

where $\mathcal{L}$ is associated with the particular governing equation. For instance, in structural analysis, a structure can be modeled by the linear elasticity equations (i.e., Cauchy-Navier equations); the

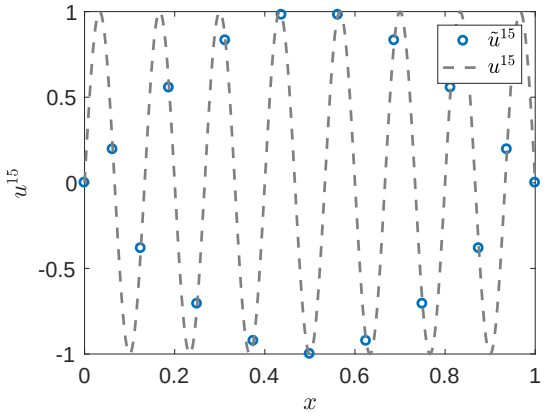


(a) $k = 1$

(b) $k = 2$



(c) $k = 3$



(d) $k = 15$

Figure 19.2: Approximation of eigenmodes of the Laplacian for $\Delta x = 1/16$.

identification of resource modes requires the solution of an eigenproblem where $\mathcal{L}$ is the Cauchy-Navier operator. As another example, in fluid dynamics, the analysis of hydrodynamics stability requires the solution of an eigenproblem where $\mathcal{L}$ is the linearized Navier-Stokes operator. While closed form solutions to these problems do not exist in general, we can numerically approximate the problem by solving

$$\tilde{\mathcal{L}}\tilde{u} = \tilde{\lambda}\tilde{u},$$

where $\tilde{\mathcal{L}}$ is an appropriate finite difference approximation of $\mathcal{L}$.

19.7 Summary

We summarize the key points of this lecture:

1. The negative of the Laplacian operator has an infinite number of eigenpairs. The eigenvaleus are positive, and eigenvectors are orthogonal.
2. We can approximate the eigenpairs of the negative of the Laplacian operator by approximating the operator using the second-order central difference formula. The associated eigenvalues are positive and orthogonal. However, there are only a finite number of eigenpairs.
3. The relative error in the k-th eigenvalue is $\mathcal{O}(k^2\Delta x^2)$. The (true) eigenvalues are well approximated assuming Δx is sufficiently small in the sense $k\Delta x \ll 1$.
4. Eigenmodes of low modes (i.e., $k\Delta x \ll 1$) are well represented. High modes are not well represented due to aliasing.
5. The finite difference method for eigenproblems may be extended to other equations and higher dimensions.

Lecture 20

Time-dependent PDEs

©2016–2022 Masayuki Yano. Prepared for AER336 Scientific Computing taught at the University of Toronto.

20.1 Motivation

We have so far considered initial value problems (IVPs) governed by a system of ordinary differential equations (ODEs) and steady boundary value problems. We will now consider problems governed by time-dependent partial differential equations (PDEs).

20.2 Heat equation

The heat equation over a one-dimensional domain $\Omega \equiv (0,1)$ and a time-interval $I \equiv (0, t_f]$ is given by

$$\begin{aligned} \frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} &= f \quad \text{in } \Omega \times I, \\ u &= 0 \quad \text{on } \partial\Omega \times I, \\ u &= u^0 \quad \text{on } \Omega \times \{t = 0\}; \end{aligned}$$

here κ is the thermal diffusivity, f is volume heat source, and u^0 is the initial condition.

We now obtain a *semi-discrete* equation of the PDE by discretizing the problem in space, but not in time. Specifically, following the presentation in the previous lectures, we introduce a one-dimensional grid with grid points

$$x_i = i\Delta x, \quad i = 0, \dots, n+1,$$

where $\Delta x = 1/(n+1)$ is the grid spacing. We then denote the approximate solution at the grid point for any $t \in (0, t_f]$ by

$$\tilde{u}_i(t) \equiv \tilde{u}(x_i, t), \quad i = 0, \dots, n+1.$$

We approximate the Laplacian using the second-order finite-difference formula to obtain the following semi-discrete equation:

$$\frac{d\tilde{u}_i}{dt} - \kappa[\frac{1}{\Delta x^2}(\tilde{u}_{i+1} - 2\tilde{u}_i + \tilde{u}_{i-1})] = \tilde{f}_i, \quad i = 1, \ldots, n, \quad t \in (0, t_f],$$
$$\tilde{u}_0 = \tilde{u}_{n+1} = 0, \quad t \in (0, t_f],$$
$$\tilde{u}_i(t = 0) = u_i^0, \quad i = 1, \ldots, n.$$

Note that we now have an IVP governed by a system of n ODEs. The equation is said to be *semi-discrete* because it is discrete in space but is continuous in time. We can write the equation more compactly using the matrix notation:

$$\frac{d\tilde{u}}{dt} + \kappa A\tilde{u} = \tilde{f}, \quad t \in (0, t_f];$$

here the matrix A is associated with the second-order central difference approximation of the negative of the Laplacian, which was introduced in the lecture on Poisson's equation.

20.3 Modal decomposition of A and the solution of the IVP

We can now solve the IVP governed by the system of ODEs using any time integration technique, for instance multistep or multistage methods. We recall that, for an IVP given by $du/dt = g(u, t)$, a good choice of an IVP integration scheme relies on the eigenvalues — and in particular the stiffness ratio — of the operator $\nabla_u g$, which for the heat equation is $-\kappa A$.

We recall from the previous lecture that the eigenvalues of the matrix $A \in \mathbb{R}^{n\times n}$ is given by

$$\tilde{\lambda}^k = \frac{2}{\Delta x^2}(1 - \cos(k\pi\Delta x)), \quad k = 1, \ldots, n.$$

The minimum eigenvalue is

$$\tilde{\lambda}^1 = \frac{2}{\Delta x^2}(1 - \cos(\pi\Delta x)) = \frac{2}{\Delta x^2}(\frac{1}{2}\pi^2\Delta x^2 + \mathcal{O}(\Delta x^4)) \approx \pi^2.$$

The maximum eigenvalue is

$$\tilde{\lambda}^n = \frac{2}{\Delta x^2}(1 - \cos(\pi n\Delta x)) \approx \frac{4}{\Delta x^2},$$

where the approximation follows from $n\Delta x = n/(n + 1) \approx 1$ and hence $\cos(\pi n\Delta x) \approx \cos(\pi) \approx -1$ for n sufficiently large (i.e. Δx sufficiently small). It follows that the stiffness ratio is

$$r \equiv \frac{\tilde{\lambda}^n}{\tilde{\lambda}^1} \approx \frac{4}{\pi^2\Delta x^2} \approx \frac{4n^2}{\pi^2}.$$

The stiffness ratio of the problem increases as $1/\Delta x^2$. Hence, when we employ a fine discretization in space (i.e. Δx small), the problem will be very stiff.

There are two plausible strategies to solve the IVP:

1. Use an explicit method and scale the time step according to $\Delta t \sim \Delta x^2$ such that $\lambda^n\Delta t$ remains in the stable region of the absolute stability diagram;

2. Use an A-stable method and choose time step without any stability considerations.

The first strategy does not scale well with Δx, as we must take smaller and smaller time step to remain stable. Hence, the recommended strategy is to employ an A-stable method. We may for example consider backward Euler, Crank-Nicolson, backward differentiation formulas (BDFs), or implicit Runge-Kutta schemes.

20.4 Error analysis

There are two difference sources of errors associated with the discretization of time-dependent PDEs. The first is associated with the spatial discretization of the (spatial) differential operator by finite difference; for instance, for the second-order central difference,

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = f \quad \Rightarrow \quad \frac{\partial \tilde{u}}{\partial t} + A\tilde{u} = \tilde{f}.$$

The second is associated with the temporal discretization of the (now finite-dimensional) ODEs; for instance, for backward Euler,

$$\frac{\partial \tilde{u}}{\partial t} = -A\tilde{u} + \tilde{f} \quad \Rightarrow \quad \frac{\tilde{\tilde{u}}^j - \tilde{\tilde{u}}^{j-1}}{\Delta t} = -A\tilde{\tilde{u}}^j + \tilde{f}^j.$$

We note that our error can be decomposed as

$$\|u - \tilde{\tilde{u}}\|_\infty \equiv \max_{\substack{i=1,\ldots,n \\ j=0,\ldots,J}} |u(x_i, t^j) - \tilde{u}_i^j| \leq \underbrace{\|u - \tilde{u}\|_\infty}_{\text{spatial error}} + \underbrace{\|\tilde{u} - \tilde{\tilde{u}}\|_\infty}_{\text{temporal error}}.$$

Moreover, it can be shown that, if we use a p-th order accurate discretization in space and q-th order accurate integration in time, we have

$$\|u - \tilde{\tilde{u}}\|_\infty \leq C\Delta x^p + C'\Delta x^q,$$

assuming the solution is sufficiently regular. For instance, if we discretize the heat equation using the second-order central difference in space and backward Euler (which is first order accurate) in time, then the error is $\mathcal{O}(\Delta x^2, \Delta t^1)$. In order to obtain an accurate approximation, we must control both the spatial discretization error and temporal discretization error. In particular, to achieve high efficiency, we need to balance the error due to the two sources; for instance, using a very small time stepping would be counterproductive if the error due to spatial discretization is large.

20.5 Example: heat equation

We now consider the heat equation with a source term $f(x) = x(1-x^3)\exp(x)$, a boundary condition $u|_{x\in\partial\Omega} = 0$, and an initial condition $u|_{t=0} = 0$. We use the second-order central difference formula for spatial discretization, and the backward Euler method for temporal integration. Figure 20.1 shows the approximate solution for $\Delta x = 1/16$ and $\Delta t = 1/16$. We observe that the solution starts with the initial state of $u(t = 0) = 0$ and grows until it reaches the steady state solution.

Table 20.1 shows the maximum error $\|u - \tilde{u}\|_\infty$ as a function of Δx and Δt. As discussed, for time dependent equations, the solution accuracy can be limited by either the lack of spatial

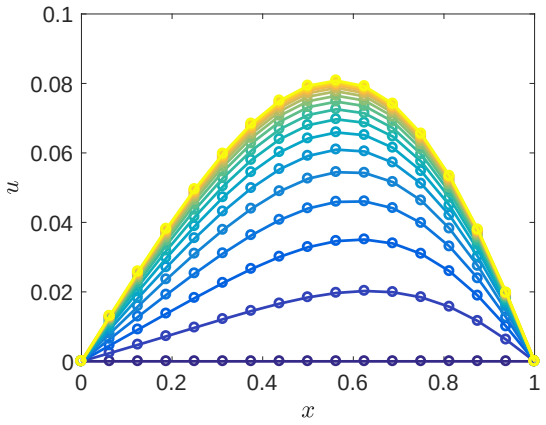


Figure 20.1: Approximation of the heat equation problem for $\Delta x = 1/16$ and $\Delta t = 1/16$.

	$\Delta t = 1/16$	$\Delta t = 1/32$	$\Delta t = 1/64$	$\Delta t = 1/128$	$\Delta t = 1/256$
$\Delta x = 1/4$	3.89×10^{-3}	4.12×10^{-3}	4.22×10^{-3}	4.27×10^{-3}	4.30×10^{-3}
$\Delta x = 1/8$	3.91×10^{-3}	2.02×10^{-3}	1.06×10^{-3}	1.10×10^{-3}	1.12×10^{-3}
$\Delta x = 1/16$	4.10×10^{-3}	2.14×10^{-3}	1.08×10^{-3}	5.30×10^{-4}	2.69×10^{-4}

Table 20.1: Maximum error for the heat equation problem as a function of the grid spacing Δx and time stepping Δt using the second-order central difference discretization in space and the backward Euler method in time.

resolution or the lack of temporal resolution. Hence, we must decrease both Δx *and* Δt to decrease the error. Specifically, we observe that if Δt is sufficiently small such that the accuracy is limited by Δx (i.e. $\Delta t = 1/256$), then the error converges as $\mathcal{O}(\Delta x^2)$, which is consistent with our expectation for the second-order accurate spatial discretization. Conversely, if Δx is sufficiently small such that the accuracy is limited by Δt (i.e. $\Delta x = 1/16$), then the error converges as $\mathcal{O}(\Delta t^1)$, which is consistent with the backward Euler method.

20.6 Wave equation

The wave equation over a one-dimensional domain $\Omega = (0, 1)$ and a time interval $I \equiv (0, t_f]$ is given by

$$\begin{aligned} \frac{\partial^2 \varphi}{\partial t^2} &= \frac{\partial^2 \varphi}{\partial x^2} \quad \text{in } \Omega \times I, \\ \varphi &= 0 \quad \text{on } \partial\Omega \times I, \\ \varphi &= x^0 \quad \text{on } \Omega \times \{t = 0\}, \\ \varphi' &= v^0 \quad \text{on } \Omega \times \{t = 0\}, \end{aligned}$$

where x^0 specifies the initial position, and v^0 specifies the initial velocity.

We now obtain a semi-discrete equation of the PDE by discretization the problem in space (and not in time). Applying the second-order central difference discretization of the Laplacian, we

obtain

$$\frac{d^2\tilde{\varphi}_i}{dt^2} = [\frac{1}{\Delta x^2}(\tilde{\varphi}_{i+1} - 2\tilde{\varphi}_i + \tilde{\varphi}_{i-1})] \quad \text{in } (0, t_f],$$

with boundary conditions $\tilde{\varphi}_0 = \tilde{\varphi}_n = 0$. The matrix form of the equation is

$$\frac{d^2\tilde{\varphi}}{dt^2} = -A\tilde{\varphi}, \quad t \in (0, t_f],$$

where A is associated with the central difference discretization of the negative of the Laplacian. The initial conditions are $\tilde{\varphi} = \tilde{x}^0$ and $\tilde{\varphi}' = \tilde{v}^0$.

We now have the second-order IVP. This IVP can be recast into a first-order form by introducing variables $\tilde{w}_0 = \tilde{\varphi}$ and $\tilde{w}_1 = \tilde{\varphi}'$. The resulting first-order IVP is

$$\frac{d}{dt}\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \end{pmatrix} = \begin{pmatrix} 0 & I \\ -A & 0 \end{pmatrix}\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \end{pmatrix}, \quad t \in (0, t_f],$$
$$\begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \end{pmatrix}_{t=0} = \begin{pmatrix} \tilde{x}^0 \\ \tilde{v}^0 \end{pmatrix}.$$

We could write the equation even more compactly by introducing

$$\tilde{u} = \begin{pmatrix} \tilde{w}_0 \\ \tilde{w}_1 \end{pmatrix} \in \mathbb{R}^{2n}, \quad \tilde{u}^0 = \begin{pmatrix} \tilde{x}^0 \\ \tilde{v}^0 \end{pmatrix} \in \mathbb{R}^{2n}, \quad \text{and} \quad B = \begin{pmatrix} 0 & I \\ -A & 0 \end{pmatrix} \in \mathbb{R}^{2n\times 2n};$$

we then have

$$\frac{d\tilde{u}}{dt} = B\tilde{u}, \quad t \in (0, t_f],$$
$$\tilde{u}(t=0) = \tilde{u}^0.$$

20.7 Modal decomposition of B and the solution of the IVP

The $2n$ eigenvalues of the matrix B are given by

$$\tilde{\lambda}^k = (-1)^k \frac{i}{\Delta x}\sqrt{2(1 - \cos(m_k\pi\Delta x))},$$

where $m_k = \lceil k/2 \rceil$, $k = 1, \ldots, 2n$. The associated eigenvectors can be expressed in a block form,

$$\tilde{v}^k = \begin{pmatrix} \tilde{p}^k \\ \tilde{q}^k \end{pmatrix},$$

where each of the block components are given by

$$\tilde{p}_i^k = \sin(m_k\pi x_i),$$
$$\tilde{q}_i^k = (-1)^k \frac{i}{\Delta x}\sqrt{2(1 - \cos(m_k\pi\Delta x))}\sin(m_k\pi x_i).$$

We observe that all $2n$ eigenvalues of B are purely imaginary, and they are n complex pairs. The maximum eigenvalue (in magnitude) is $|\tilde{\lambda}^{2n}| \approx 2/\Delta x \approx 2n$.

In order to preserve the energy of the oscillatory modes associated with purely imaginary eigenvalues, a recommended time integrator for the wave equation are those that have a relatively large stability boundary along the imaginary axis. For instance, the four-stage explicit Runge-Kutta method (RK4) or the Crank-Nicolson method are good choices.

	$\Delta t = 1/16$	$\Delta t = 1/32$	$\Delta t = 1/64$	$\Delta t = 1/128$	$\Delta t = 1/256$
$\Delta x = 1/16$	2.02×10^{-1}	1.25×10^{-1}	1.00×10^{-1}	9.38×10^{-2}	9.21×10^{-2}
$\Delta x = 1/32$	1.66×10^{-1}	6.84×10^{-2}	3.71×10^{-2}	2.84×10^{-2}	2.62×10^{-2}
$\Delta x = 1/64$	1.56×10^{-1}	5.20×10^{-2}	1.89×10^{-2}	9.68×10^{-3}	7.26×10^{-3}
$\Delta x = 1/128$	1.53×10^{-1}	4.80×10^{-2}	1.42×10^{-2}	4.86×10^{-3}	2.42×10^{-3}
$\Delta x = 1/256$	1.52×10^{-1}	4.70×10^{-2}	1.30×10^{-2}	3.62×10^{-3}	1.25×10^{-3}

Table 20.2: Maximum error for the wave equation problem as a function of the grid spacing Δx and time stepping Δt using the second-order central difference discretization in space and the Crank-Nicolson method in time.

20.8 Example: wave equation

We now consider the wave equation with a boundary condition $\varphi|_{x\in\partial\Omega} = 0$, and initial conditions $\varphi|_{t=0} = \exp(-32(x-1/2)^2)$ and $\varphi'|_{t=0} = 0$. We use the second-order central difference formula for the spatial discretization ($\Delta x = 1/32$), and the Crank-Nicolson method for temporal integration ($\Delta t = 1/32$). Figure 20.2 shows the evolution of the solution over time. As expected, the solution consists of two waves — one left-traveling and the other right-traveling — which are reflected at the boundaries.

Table 20.2 shows the maximum error $\|u - \tilde{u}\|_\infty$ as a function of Δx and Δt. As before, the solution accuracy can be limited by either the lack of spatial or temporal resolution; we must decrease both Δx *and* Δt to control the error.

20.9 Generalization: other equations and higher dimensions

The semi-discrete formulation for time-dependent problems readily extend to other equations and higher dimensions. Specifically, a time-dependent PDE that is first-order in time may be expressed as

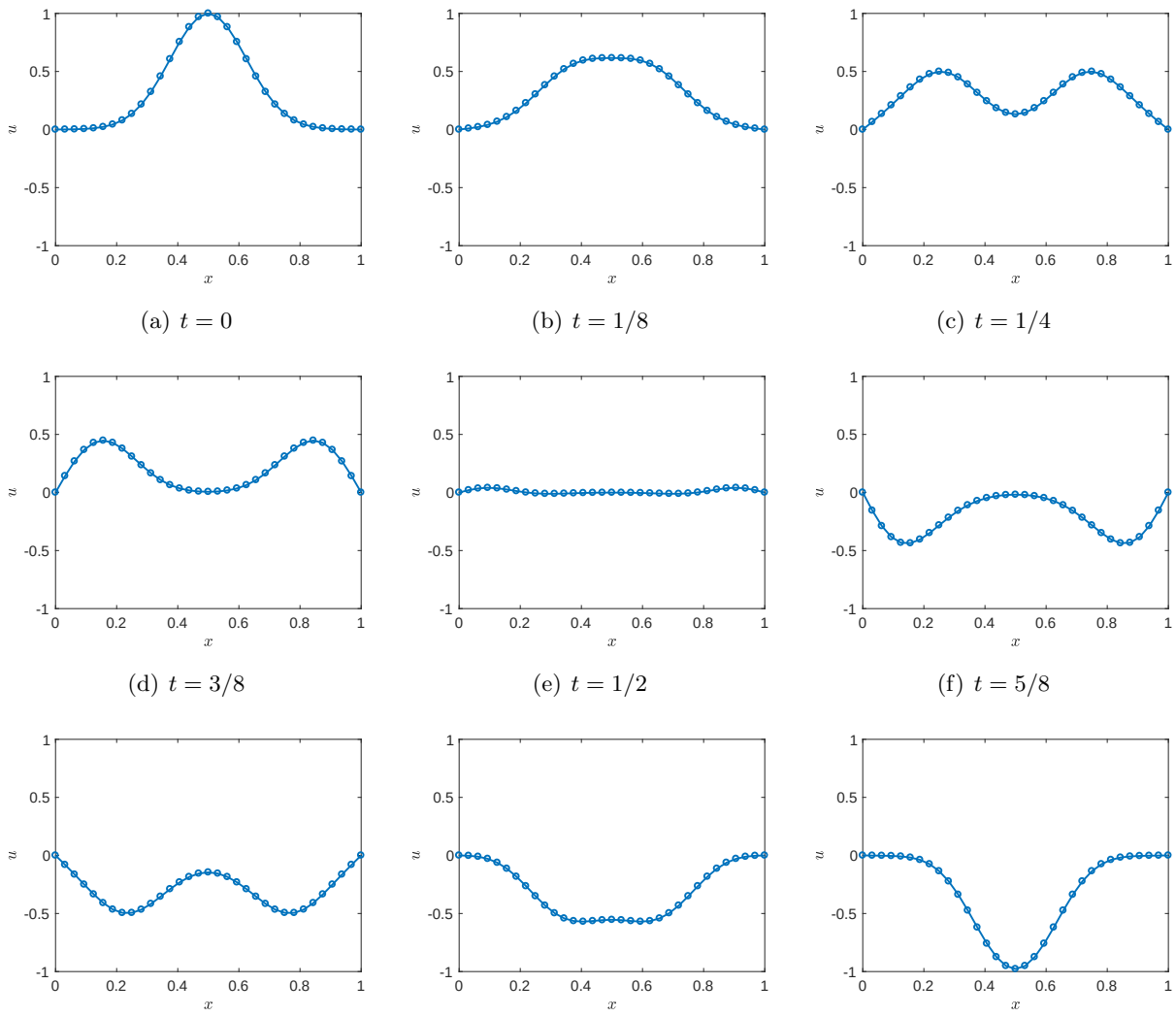
$$\begin{aligned} \frac{\partial u}{\partial t} + \mathcal{L}u &= f \quad \text{in } \Omega \times (0, t_f], \\ u &= u^0 \quad \text{on } \Omega \times \{t = 0\}, \end{aligned}$$

with appropriate boundary conditions. The semi-discrete form of the PDE is

$$\begin{aligned} \frac{\partial \tilde{u}}{\partial t} + \tilde{\mathcal{L}}\tilde{u} &= \tilde{f} \quad \text{in } (0, t_f], \\ \tilde{u}(t = 0) &= \tilde{u}^0, \end{aligned}$$

where $\tilde{\mathcal{L}}$ results from an appropriate finite difference approximation of the differential operator $\mathcal{L}$. We emphasize that $\mathcal{L}$ may represent any linear operator — for instance associated with advection-diffusion or reaction-diffusion — which may be defined over any spatial dimension. For instance, the heat equation with homogeneous Dirichlet boundary conditions in two dimensions would yield a semi-discrete equation

$$\frac{d\tilde{u}_{i,j}}{dt} - \kappa \left[\frac{\tilde{u}_{i+1,j} - 2\tilde{u}_{i,j} + \tilde{u}_{i-1,j}}{\Delta x^2} + \frac{\tilde{u}_{i,j+1} - 2\tilde{u}_{i,j} + \tilde{u}_{i,j-1}}{\Delta y^2} \right] = \tilde{f}_{i,j}, \quad i, j = 1, \ldots, n,$$



(a) $t = 0$ (b) $t = 1/8$ (c) $t = 1/4$

(d) $t = 3/8$ (e) $t = 1/2$ (f) $t = 5/8$

(g) $t = 3/4$ (h) $t = 7/8$ (i) $t = 1$

Figure 20.2: Approximation to the wave equation obtained using the second-order central difference discretization in space ($\Delta x = 1/33$) and the Crank-Nicolson method in time ($\Delta t = 1/32$).

with boundary conditions $\tilde{u}_{0,j} = \tilde{u}_{n+1,j} = 0$, $j = 0, \ldots, n+1$, and $\tilde{u}_{i,0} = \tilde{u}_{i,n+1} = 0$, $i = 1, \ldots, n$, and an initial condition $\tilde{u}_{i,j}(t=0) = u^0(x_i, y_j)$. More compactly, we can write the equations in matrix form as

$$\begin{aligned} \frac{d\tilde{u}}{dt} &= -\kappa A\tilde{u} \quad \text{in } (0, t_f], \\ \tilde{u}(t=0) &= \tilde{u}^0, \end{aligned}$$

where A is the central difference approximation of the negative of the Laplacian in two dimensions.

Similarly, a time-dependent PDE that is second-order in time may be expressed as

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} &= \mathcal{L}u + f \quad \text{in } \Omega \times (0, t_f], \\ u &= u^0 \quad \text{on } \Omega \times \{t = 0\}, \\ u' &= u'^0 \quad \text{on } \Omega \times \{t = 0\}. \end{aligned}$$

The semi-discrete form of the PDE is

$$\begin{aligned} \frac{\partial^2 \tilde{u}}{\partial t^2} &= \tilde{\mathcal{L}}\tilde{u} + \tilde{f} \quad \text{in } (0, t_f], \\ \tilde{u}(t=0) &= \tilde{u}^0, \\ \tilde{u}'(t=0) &= \tilde{u}'^0. \end{aligned}$$

The semi-discrete form can then be turned into a first-order form by introducing an auxiliary variable for $\tilde{u}'$. Again, $\mathcal{L}$ may represent any linear operator in any spatial dimension. For instance, the wave equation in two dimensions would yield a semi-discrete equation

$$\frac{d^2\tilde{\varphi}_{i,j}}{dt^2} = \frac{\tilde{\varphi}_{i+1,j} - 2\tilde{\varphi}_{i,j} + \tilde{\varphi}_{i-1,j}}{\Delta x^2} + \frac{\tilde{\varphi}_{i,j+1} - 2\tilde{\varphi}_{i,j} + \tilde{\varphi}_{i,j-1}}{\Delta y^2}, \quad i, j = 1, \ldots, n,$$

with boundary conditions $\tilde{\varphi}_{0,j} = \tilde{\varphi}_{n,j} = 0$, $j = 0, \ldots, n+1$, and $\tilde{\varphi}_{i,0} = \tilde{\varphi}_{i,n} = 0$, $i = 1, \ldots, n$, and initial conditions $\tilde{\varphi}_{i,j}(t=0) = x^0(x_i, y_j)$ and $\tilde{\varphi}'_{i,j}(t=0) = v^0(x_i, y_j)$.

20.10 Summary

We summarize the key points of this lecture:

1. As semi-discrete form of a time-dependent PDE is obtained by discretizing the problem in space but not in time.

2. A fully discrete form of a time-dependent PDE is obtained by applying a time-integration scheme to a semi-discrete form.

3. The choice of a time-integration scheme depends on the stiffness of the IVP, which can be determined by analyzing the eigenvalues of the spatial matrix.

4. The accuracy of a numerical approximation of a time-dependent PDE depends on two factors: the spatial discretization error and the time discretization error. Both errors must be controlled to control the accuracy of the approximation.

5. PDEs that are higher than first-order in time (e.g., wave equation) can be recast into a first-order form and solved using a standard time-marching scheme.

6. The two-step strategy, in which we first discreteize in space and then in time, applies to any time-dependent PDE in arbitrary dimensions.