Lecture 28

The Main Sources of Error

Truncation Error

Truncation error is defined as the error caused directly by an approximation method. For instance, all numerical integration methods are approximations and so there is error, even if the calculations are performed exactly. Numerical differentiation also has a truncation error, as will the differential equations methods we will study in Part IV, which are based on numerical differentiation formulas. There are two ways to minimize truncation error: (1) use a higher order method, and (2) use a finer grid so that points are closer together. Unless the grid is very small, truncation errors are usually much larger than roundoff errors. The obvious tradeoff is that a smaller grid requires more calculations, which in turn produces more roundoff errors and requires more running time.

Roundoff Error

Roundoff error always occurs when a finite number of digits are recorded after an operation. Fortunately, this error is extremely small. The standard measure of how small is called **machine epsilon**. It is defined as the smallest number that can be added to 1 to produce another number on the machine, i.e. if a smaller number is added the result will be rounded down to 1. In IEEE standard double precision (used by MATLAB and most serious software), machine epsilon is 2^{-52} or about 2.2×10^{-16}. A different, but equivalent, way of thinking about this is that the machine records 52 floating binary digits or about 15 floating decimal digits. Thus there are never more than 15 significant digits in any calculation. This of course is more than adequate for any application. However, there are ways in which this very small error can cause problems.

You can test that machine epsilon is 2^{-52}:

```
≫ format long
≫ (1 + 2^(-52)) - 1
≫ (1 + 2^(-53)) - 1
```

MATLAB has a command that produces machine epsilon:

```
≫ eps
```

To see an unexpected occurance of round-off try

```
≫ (2^52+1) - 2^52
≫ (2^53+1) - 2^53
```

Thus roundoff isn't always small! It is just small compared with the scale of the numbers you are calculating. A number of magnitude 10^p will have roundoff of magnitude about $10^p \cdot 10^{-16} = 10^{p-16}$.

Loss of Precision (also called Loss of Significance)

Suppose we had some way to compute π that effectively did the calculation $(e \cdot 10^9 + \pi) - e \cdot 10^8$. Rounding everything to 16 digits, we are computing

$$(2718281828.459045 + 3.141592653589793) - 2718281828.459045\,.$$

The addition is peformed first and the result rounded to 16 digits, giving

$$2718281831.600637 - 2718281828.459045\,.$$

Although roundoff caused some error, it is about a factor of 10^{-16} smaller than the numbers shown. In other words, all but perhaps the last digit shown is correct. Next the subtraction is performed, giving

$$0000000003.141592\,.$$

The leading zeros are not significant, so we lost 9 significant digits and have only 7 left. This type of error, where common leading significant digits cancel, is loss-of-precision (also called loss-of-significance) error. Computers will not display these leading zeros. Instead, the subtraction above yielded

$$3.141592025756836\,.$$

Although it is displayed with 16 digits, only the first 7 are correct. Usually, if you add two numbers of magnitude 10^p each with roundoff error 10^{p-16}, then the result is also of magnitude 10^p and the relative error due to roundoff is $10^{p-16}/10^p = 10^{-16}$. In this example, cancellation made the result of magnitude 10^{p-q}, so the relative error due to roundoff is $10^{p-16}/10^{p-q} = 10^{q-16}$, and so we lost $q = 9$ digits of precision.

This type of *loss of precision* can happen by accident, with catastrophic results, if you are not careful. For example in $f'(x) \approx (f(x+h) - f(x))/h$ you will lose precision when h gets too small. Try

```
>> format long
>> format compact
>> f = @(x) x^2
>> for i = 1:30
>>    h = 10^(-i)
>>    df = (f(1+h)-f(1))/h
>>    relerr = (2-df)/2
>> end
```

At first the relative error decreases since truncation error is reduced. Then loss of precision takes over and the relative error increases to 1. This happens because when $f(1)$ and $f(1+h)$ become close, the subtraction "cancels" digits.

Bad Conditioning

We encountered bad conditioning in Part II, when we talked about solving linear systems. Bad conditioning means that the problem is unstable in the sense that small input errors can produce large output errors. This can be a problem in a couple of ways. First, the measurements used to get the inputs cannot be completely accurate. Second, the computations along the way have roundoff errors. Errors in the computations near the beginning especially can be magnified by a factor close to the condition number of the matrix. Thus what was a very small problem with roundoff can become a very big problem.

It turns out that matrix equations are not the only place where condition numbers occur. In any problem one can define the condition number as the maximum ratio of the relative errors in the output versus input, i.e.

$$\text{condition \# of a problem} = \max\left(\frac{\text{Relative error of output}}{\text{Relative error of inputs}}\right).$$

An easy example is solving a simple equation

$$f(x) = 0.$$

Suppose that f' is close to zero at the solution x^*. Then a very small change in f (caused perhaps by an inaccurate measurement of some of the coefficients in f) can cause a large change in x^*. It can be shown that the condition number of this problem is $1/f'(x^*)$.

Summary

Error type:	Whose fault is it?	How to mitigate it?
Truncation	the method	higher-order method or finer grid
Round-off	the computer	usually okay, higher precision arithmetic
Loss of Precision	the programmer	avoid cancellation of significant digits
Bad Conditioning	the problem	check answers, redesign if possible

Table 28.1: A summary of the main sources of error.

Exercises

28.1 Identify the (main) source of error in each case and propose a way to reduce this error if possible.

(a) If we do $(\sqrt{3})^2$ we should get 3, but if we check

```
≫ mythree = (sqrt(3))^2
≫ mythree-3
```

we find the error is `ans = -4.4409e-16`.

(b) Since it is a quarter of a circle of radius 2, we should have $\int_0^2 \sqrt{4-x^2}dx = \frac{1}{4}\pi 2^2 = \pi$. We try to use `mytrap` from Lecture 21 and do

```
≫ x = 0:.2:2;
≫ y = sqrt(4-x.^2);
≫ mypi = mytrap(x,y)
≫ mypi-pi
```

and find the error is `ans = -0.0371`.

28.2 [1] The function $f(x) = (x-2)^9$ could be evaluated as written, or first expanded as $f(x) = x^9 - 18x^8 + \cdots$ and then evaluated. To find the expanded version, type

```
≫ syms x
≫ expand((x-2)^9)
≫ clear
```

To evaluate it without expansion, type

```
≫ f1 = @(x) (x-2).^9
≫ x = 1.92:.001:2.08;
≫ y1 = f1(x);
≫ plot(x,y1,'blue')
```

To do it with expansion, convert the symbolic expansion above to an anonymous function `f2` and then type

```
≫ y2 = f2(x);
≫ hold on
≫ plot(x,y2,'red')
```

Carefully study the resulting graphs. Should the graphs be the same? Which is more correct? MATLAB does calculations using approximately 16 decimal places. What is the largest error in the graph, and how big is it relative to 10^{-16}? Which source of error is causing this problem?

[1] From *Numerical Linear Algebra* by L. Trefethen and D. Bau, SIAM, 1997.

Review of Part III

Methods and Formulas

Polynomial Interpolation:

An exact fit to the data.
For n data points it is a $n-1$ degree polynomial.
Only good for very few, accurate data points.
The coefficients are found by solving a linear system of equations.

Spline Interpolation:

Fit a simple function between each pair of points.
Joining points by line segments is the most simple spline.
Cubic is by far the most common and important.
Cubic matches derivatives and second derivatives at data points.
Simply supported and clamped ends are available.
Good for more, but accurate points.
The coefficients are found by solving a linear system of equations.

Least Squares:

Makes a "close fit" of a simple function to all the data.
Minimizes the sum of the squares of the errors.
Good for noisy data.
The coefficients are found by solving a linear system of equations.

Interpolation vs. Extrapolation:

Polynomials, Splines and Least Squares are generally used for *Interpolation*, fitting between the data. *Extrapolation*, i.e. making fits beyond the data, is much more tricky. To make predictions beyond the data, you must have knowledge of the underlying process, i.e. what the function should be.

Numerical Integration:

Left Endpoint:

$$L_n = \sum_{i=1}^{n} f(x_{i-1})\Delta x_i$$

Right Endpoint:

$$R_n = \sum_{i=1}^{n} f(x_i)\Delta x_i.$$

Trapezoid Rule:

$$T_n = \sum_{i=1}^{n} \frac{f(x_{i-1}) + f(x_i)}{2}\Delta x_i.$$

Midpoint Rule:

$$M_n = \sum_{i=1}^{n} f(\bar{x}_i)\Delta x_i \qquad \text{where} \qquad \bar{x}_i = \frac{x_{i-1} + x_i}{2}.$$

Numerical Integration Rules with Even Spacing:

For even spacing: $\Delta x = \frac{b-a}{n}$ where n is the number of subintervals, then:

$$L_n = \Delta x \sum_{i=0}^{n-1} y_i = \frac{b-a}{n} \sum_{i=0}^{n-1} f(x_i),$$

$$R_n = \Delta x \sum_{i=1}^{n} y_i = \frac{b-a}{n} \sum_{i=1}^{n} f(x_i),$$

$$T_n = \Delta x\big(y_0 + 2y_1 + \ldots + 2y_{n-1} + y_n\big) = \frac{b-a}{2n}\big(f(x_0) + 2f(x_1) + \ldots + 2f(x_{n-1}) + f(x_n)\big),$$

$$M_n = \Delta x \sum_{i=1}^{n} \bar{y}_i = \frac{b-a}{n} \sum_{i=1}^{n} f(\bar{x}_i),$$

Simpson's rule:

$$\begin{aligned} S_n &= \Delta x\big(y_0 + 4y_1 + 2y_2 + 4y_3 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n\big) \\ &= \frac{b-a}{3n}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right). \end{aligned}$$

Area of a region:

$$A = -\oint_C y\,dx,$$

where C is the counter-clockwise curve around the boundary of the region. We can represent such a curve, by consecutive points on it, i.e. $\bar{x} = (x_0, x_1, x_2, \ldots, x_{n-1}, x_n)$, and $\bar{y} = (y_0, y_1, y_2, \ldots, y_{n-1}, y_n)$ with $(x_n, y_n) = (x_0, y_0)$. Applying the trapezoid method to the integral (21.4):

$$A = -\sum_{i=1}^{n} \frac{y_{i-1} + y_i}{2}\left(x_i - x_{i-1}\right)$$

Accuracy of integration rules:

Right and Left endpoint are $O(\Delta x)$
Trapezoid and Midpoint are $O(\Delta x^2)$
Simpson is $O(\Delta x^4)$

Double Integrals on Rectangles:

Centerpoint:

$$C_{mn} = \sum_{i,j=1,1}^{m,n} f(c_{ij}) A_{ij}.$$

where

$$c_{ij} = \left(\frac{x_{i-1} + x_i}{2}, \frac{y_{i-1} + y_i}{2} \right).$$

Centerpoint – Evenly spaced:

$$C_{mn} = \Delta x \Delta y \sum_{i,j=1,1}^{m,n} z_{ij} = \frac{(b-a)(d-c)}{mn} \sum_{i,j=1,1}^{m,n} f(c_{ij}).$$

Four corners:

$$F_{mn} = \sum_{i,j=1,1}^{m,n} \frac{1}{4} \left(f(x_i, y_j) + f(x_i, y_{j+1}) + f(x_{i+1}, y_i) + f(x_{i+1}, y_{j+1}) \right) A_{ij},$$

Four Corners – Evenly spaced:

$$\begin{aligned} F_{mn} &= \frac{A}{4} \left(\sum_{\text{corners}} f(x_i, y_j) + 2 \sum_{\text{edges}} f(x_i, y_j) + 4 \sum_{\text{interior}} f(x_i, y_j) \right) \\ &= \frac{(b-a)(d-c)}{4mn} \sum_{i,j=1,1}^{m,n} W_{ij} f(x_i, y_j). \end{aligned}$$

where

$$W = \begin{pmatrix} 1 & 2 & 2 & 2 & \cdots & 2 & 2 & 1 \\ 2 & 4 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 2 & 4 & 4 & 4 & \cdots & 4 & 4 & 2 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 2 & 4 & 4 & 4 & \cdots & 4 & 4 & 2 \\ 1 & 2 & 2 & 2 & \cdots & 2 & 2 & 1 \end{pmatrix}.$$

Double Simpson:

$$S_{mn} = \frac{(b-a)(d-c)}{9mn} \sum_{i,j=1,1}^{m,n} W_{ij} f(x_i, y_j).$$

where

$$W = \begin{pmatrix} 1 & 4 & 2 & 4 & \cdots & 2 & 4 & 1 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ 2 & 8 & 4 & 8 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ 2 & 8 & 4 & 8 & \cdots & 4 & 8 & 2 \\ 4 & 16 & 8 & 16 & \cdots & 8 & 16 & 4 \\ 1 & 4 & 2 & 4 & \cdots & 2 & 4 & 1 \end{pmatrix}.$$

Integration based on triangles:

– Triangulation: Dividing a region up into triangles.
– Triangles are suitable for odd-shaped regions.
– A triangulation is better if the triangles are nearly equilateral.
Three corners:

$$T_n = \sum_{i=1}^{n} \bar{f}_i A_i$$

where $\bar{f}$ is the average of f at the corners of the i-th triangle.
Area of a triangle with corners (x_1, y_1), (x_2, y_2), (x_3, y_3):

$$A = \frac{1}{2} \left| \det \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix} \right|.$$

Centerpoint:

$$C = \sum_{i=1}^{n} f(\bar{x}_i, \bar{y}_i) A_i, \quad \text{with}$$

$$\bar{x} = \frac{x_1 + x_2 + x_3}{3} \quad \text{and} \quad \bar{y} = \frac{y_1 + y_2 + y_3}{3}.$$

Finite Differences

Forward Difference:

$$f'(x_i) = y_i' \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i}.$$

Backward Difference:

$$f'(x_i) \approx \frac{y_i - y_{i-1}}{x_i - x_{i-1}}.$$

Central Difference:

$$f'(x_i) = y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h}.$$

Higher order central differences:

$$f''(x_i) = y_i'' \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2},$$
$$f'''(x_i) = y_i''' \approx \frac{1}{2h^3}\left[y_{i+2} - 2y_{i+1} + 2y_{i-1} - y_{i-2}\right],$$
$$f^{(4)}(x_i) = y_i^{(4)} \approx \frac{1}{h^4}\left[y_{i+2} - 4y_{i+1} + 6y_i - 4y_{i-1} + y_{i-2}\right].$$

Partial Derivatives: Denote $u_{i,j} = u(x_i, y_j)$.

$$u_x(x_i, y_j) \approx \frac{1}{2h}\left(u_{i+1,j} - u_{i-1,j}\right),$$
$$u_y(x_i, y_j) \approx \frac{1}{2k}\left(u_{i,j+1} - u_{i,j-1}\right),$$
$$u_{xx}(x_i, y_j) \approx \frac{1}{h^2}\left(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}\right),$$
$$u_{yy}(x_i, y_j) \approx \frac{1}{k^2}\left(u_{i,j+1} - 2u_{i,j} + u_{i,j-1}\right),$$
$$u_{xy}(x_i, y_j) \approx \frac{1}{4hk}\left(u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}\right).$$

Sources of error:

Truncation – the method is an approximation.
Roundoff – double precision arithmetic uses ≈ 15 significant digits.
Loss of precision – an amplification of roundoff error due to cancellations.
Bad conditioning – the problem is sensitive to input errors.
Error can build up after multiple calculations.

See Table 28.

Matlab

Data Interpolation:

Use the plot command `plot(x,y,'*')` to plot the data.
Use the Basic Fitting tool to make an interpolation or spline.
If you choose an $n-1$ degree polynomial with n data points the result will be the exact polynomial interpolation.
If you select a polynomial degree less than $n-1$, then MATLAB will produce a least squares approximation.

Functions of 2 Variables and Meshgrids:

`f = @(x,y) sin(x.*y)./sqrt(x+y)` ..... make an anonymous function of x and y.
`[X Y] = meshgrid(-1:.01:2, 0:.01:3)` ..... make a 2-d grid of points.

`Z = f(X,Y)` ..... evaluate the function at all points on the grid.
`mesh(X,Y,Z)` or `surf(X,Y,Z)` ..... plot the function on the grid.

Integration:

`integral(f,a,b)` ..... Numerical integral of $f(x)$ on $[a,b]$.
`integral2(f,a,b,c,d)` ..... Integral of $f(x,y)$ on $[a,b]\times[c,d]$.
Example:

```
≫ f = @(x,y) sin(x.*y)/sqrt(x+y)
≫ I = integral2(f,0,1,0,2)
```

MATLAB uses an advanced form of Simpson's method.

Integration over non-rectangles:

Redefine the function to be zero outside the region. For example:

```
≫ f = @(x,y) sin(x.*y).^3.*(2*x + y <= 2)
≫ I = integral2(f,0,1,0,2)
```

Integrates $f(x,y)=\sin^3(xy)$ on the triangle with corners $(0,0)$, $(0,2)$, and $(1,0)$.

Triangles:
MATLAB stores triangulations as a matrix of vertices `V` and triangles `T`.
`T = delaunay(V)` (or `delaunay(x,y)`) ..... Produces triangles from vertices.
`trimesh(T,x,y)`
`trimesh(T,x,y,z)`
`trisurf(T,x,y)`
`trisurf(T,x,y,z)`

Logical expressions

`(2*x + y <= 2)`, `(x.^2+y.^2<=1)` are examples of logical expressions.
If a logical expression is true it is given the value 1. If false, then it is assigned the value 0.

Part IV
Differential Equations

©Copyright, Todd Young and Martin Mohlenkamp, Department of Mathematics, Ohio University, 2021