Algorithms and Computational Mathematics

Ian T. Nabney

Aston University

http://www.ncrg.aston.ac.uk
Aston University, Birmingham, UK

Session Objectives

- Extend the theory of rounding error analysis from a single function to a complete algorithm consisting of a sequence of computations.
- Understand the concepts of numerical stability and numerical trustworthiness.
- Carry out algorithm rounding error analysis for simple algorithms.
- Carry out complete error analysis for numerical differentiation.
- Compute the interpolating polynomial for a set of support points.

Algorithm Analysis

$$x = x^{(0)} \to \phi^{(0)}(x^{(0)}) = x^{(1)} \to \phi^{(1)}(x^{(1)}) = x^{(2)} \to \cdots \to \phi^{(r)}(x^{(r)}) = x^{(r+1)} = y.$$

Each $\phi^{(i)}$ is an elementary step (is basically a single machine instruction, or a well defined group of machine instructions whose error analysis is simple: e.g. $\sqrt{x}$).

Each step introduces a new rounding error, and our analysis has to take account of the way this is affected by the subsequent steps of the algorithm.

- $\psi^{(i)}$, for $i = 0, \ldots, r$, is the function given by the last $r - i + 1$ steps:

$$\psi^{(i)} = \phi^{(r)} \circ \phi^{(r-1)} \circ \cdots \circ \phi^{(i)}.$$

- We also need $D\phi^{(i)}$ and $D\psi^{(i)}$ the Jacobian matrices of $\phi^{(i)}$ and $\psi^{(i)}$.

- At the ith step of the computation,

$$\mathsf{fl}(\phi_j^{(i)}(u)) = \mathsf{rd}(\phi_j^{(i)}(u)) = (1 + \epsilon_j)\phi_j^{(i)}(u)$$

for $j = 1, \ldots, n_i + 1$, where $|\epsilon_j| \leq \text{eps}$. The values ϵ_j represent the relative rounding errors when $\phi^{(i)}$ is evaluated in floating point arithmetic.

- We define α_{i+1}, the absolute rounding error introduced when $\phi^{(i)}$ is evaluated in floating point arithmetic, by

$$\alpha_{i+1} := \mathsf{fl}(\phi^{(i)}(\tilde{x}^{(i)})) - \phi^{(i)}(\tilde{x}^{(i)}) \doteq E_{i+1} x^{(i+1)}.$$

The matrix E_{i+1} is diagonal, with entries $\epsilon_1, \ldots, \epsilon_{n_i+1}$.

Putting it all together

The total absolute error in computing y with the given algorithm in floating point arithmetic is, to first order,

$$\Delta y \doteq D\phi(x)^T \Delta x + D\psi^{(1)}(x^{(1)})^T \alpha_1 + \cdots + D\psi^{(r)}(x^{(r)})^T \alpha_r + \alpha_{r+1}.$$

This is the sum of terms consisting of the rounding error introduced at the ith step magnified by the Jacobian of the remainder of the algorithm.

Worked Example

Calculate the total absolute error when computing

$$y = \phi(a, b) = a^2 - b^2$$

using the following algorithm:

```
eta1 = a * a;
eta2 = b * b;
eta  = eta1 - eta2;
```

Numerical Trustworthiness

The error $D\phi(x) \cdot \Delta x$ is inherent in the problem (it is the conditioning). So when we compare different algorithms for computing the same quantity, it is only relevant to consider the roundoff error

$$D\psi^{(1)}(x^{(1)})^T\alpha_1 + \cdots + D\psi^{(r)}(x^{(r)})^T\alpha_r + \alpha_{r+1}.$$

which is, in this example, equal to

$$|a^2\epsilon_1 - b^2\epsilon_2 + (a^2 - b^2)\epsilon_3| \leq (a^2 + b^2 + |a^2 - b^2|)\text{eps}.$$

An algorithm is numerically more trustworthy than another for $\phi(x)$ if, for a given input set D, the rounding error is smaller.

Numerical Stability

- For any algorithm, we can expect an inherent error of magnitude

$$\Delta^{(0)}(y) := [|D\phi(x)| \cdot |x| + |y|]\,\text{eps}.$$

- Roundoff errors in an algorithm are harmless if their contribution towards Δy is at most the same order of magnitude as $\Delta^{(0)}(y)$.
- If all roundoff errors are harmless, then we say that an algorithm is numerically stable.

In the worked example,

$$\Delta^{(0)}(y) = \left(2(a^2 + b^2) + |a^2 - b^2|\right)\text{eps}.$$

It is fairly easy to see that the total error adds at most $(a^2 + b^2)$eps to this, so the algorithm is numerically stable. However, it is ill-conditioned if $a^2 \approx b^2$.

Larger Algorithms

- This method of analysis works for small algorithms, but for large numbers of steps, the calculations rapidly become infeasible.
- Statistical approaches (with simplifying assumptions about independence of arguments at each step) have been used in such cases.
- Ideally, there would be a random walk of errors, so that the total relative error would have magnitude approximately $(\sqrt{n})$eps, but as we have seen, some steps can be very much worse than this.

Efficient Evaluation of Polynomials

If `c[0]`, ..., `c[N]` represent the coefficients of a polynomial p (with `c[i]` the coefficient of x^i), then we can use Horner's scheme to evaluate the polynomial $p(x)$ with n multiplies and $n+1$ additions:

```
p = c[N];
for (j = N-1; j >= 0; j--)
    p = p * x + c[j];
```

A similar algorithm is used to compute $p(x)$ and $p'(x)$ together:

```
p = c[N];
dp = 0.0;
for (j = N-1; j >= 0; j--) {
    dp = dp*x + p;
    p = p*x + c[j];
}
```

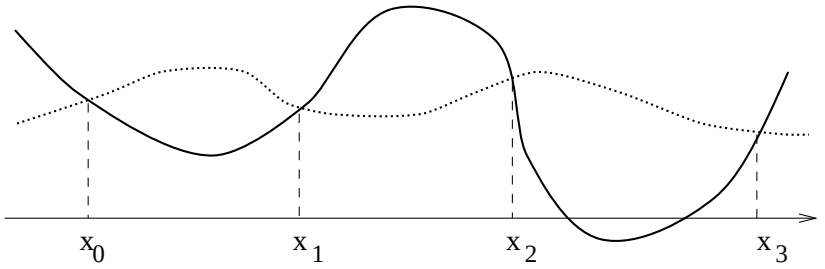
The correctness of these algorithms can be proved by induction.

Numerical Differentiation

- Often need function derivatives (e.g. for optimisation).
- Numerical algorithms can be used where analytic derivatives aren't available.
- Numerical derivatives can be used to check analytic derivatives (e.g. `gradchek` in Netlab).
- Nice example of error analysis.

Can it be done?

- If the values of a function f can be calculated at arbitrary points, can we calculate an estimate of the derivative $f'(c)$? The answer is a qualified 'yes'.
- Because the estimate of the derivative may be unreliable, we shall demand an upper bound on the size of the error, as well as the value of the derivative itself.



Difference Algorithms

- From the definition of the derivative, it follows that

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}. \tag{1}$$

- This formula gives rise to an obvious procedure for calculating $f'(x)$ which is almost guaranteed to give inaccurate results.
- We shall perform an experiment where h reduces to 0 through a sequence of positive values, and the approximations to $f'(x)$ are compared with the true value.
- Because the formula is exact in the limit as $h \to 0$, we expect the approximation to improve as h becomes smaller.

Experimental Results

- We shall select $f(x) = \tan^{-1}(x)$ and evaluate the numerical derivative at $x = \sqrt{2}$. The result should be

$$f'(x) = (x^2 + 1)^{-1} = (2 + 1)^{-1} = \frac{1}{3}.$$

- Carried out on a 32 bit computer. $f(x) = 0.95531660$. At each iteration we halve the value of h, starting from $h = 1$.

Iteration	h	$f(x+h)$	$f(x+h) - f(x)$	Result
4	0.62×10^{-1}	0.97555095	0.02023435	0.32374954
12	0.24×10^{-3}	0.95539796	0.00008136	0.33325195
20	0.95×10^{-6}	0.95531690	0.00000030	0.31250000
24	0.60×10^{-7}	0.95531666	0.00000006	1.00000000
26	0.15×10^{-7}	0.95531660	0.00000000	0.00000000

Conclusions

- Inspecting the last column in the table shows that the result is most accurate at iteration 12, and that after this point the result becomes significantly less accurate, until the answer is useless.
- The reason for this can be seen in the third and fourth columns: as h gets smaller, $f(x+h)$ becomes closer to $f(x)$ and with the limited precision at our disposal, the two values eventually become the same and the numerical derivative is 0.
- This phenonmenon is known as subtractive cancellation.

So what can we do about it?

Error Analysis

There are two main sources of error:

1. Truncation error from higher order terms in the Taylor series:

$$\frac{f(x+h)-f(x)}{h} = f' + \frac{1}{2}hf'' + \cdots$$

   We approximate the truncation error by the first omitted term $e_t \sim |hf''(x)|$. (We use $\sim$ to denote two quantities that are approximately equal, ignoring constants (such as $1/2$ in this case).)

2. Roundoff error. Analysis shows that the most important source of rounding error is the term h in the denominator of equation 1.

Rounding Error Analysis

- If x and $x + h$ are not exactly representable, then the error in h is of order $\epsilon_m x$.
- The relative (fractional) error is $\epsilon_m x/h$ which has the value 2×10^{-7} if $\epsilon_m \approx 2 \times 10^{-16}$ (which is appropriate for double precision arithmetic), $x = 10$ and $h = 1 \times 10^{-8}$. This gives rise to a similar (large) fractional error in the derivative.
- It is therefore important to ensure that x and $x + h$ differ exactly by a representable number.

```
temp = x + h;
h = temp - x;
```

- Once h is exact, the roundoff error e_r is equal to $\epsilon_f |f(x)/h|$, where ϵ_f is the relative error in f. If f is simple, then we may have $\epsilon_f \approx \epsilon_m$, but this will not be the case if f is complicated.

Optimising h

- We now choose h so as to minimise $e_r + e_t$.

$$\frac{\partial}{\partial h}(e_r + e_t) \sim |f''(x)| - \frac{\epsilon_f |f(x)|}{|h|^2}.$$

  Setting this quantity to zero implies that the optimal h is

$$h \sim \sqrt{\frac{\epsilon_f |f|}{|f''|}} \approx \sqrt{\epsilon_f} x_c,$$

  where $x_c \equiv (|f|/|f''|)^{1/2}$ is the curvature scale of the function.

- In the absence of any special information, away from zero we assume that $x_c = x$.

- If we substitute this value of h back in, we find that the relative accuracy is

$$\frac{(e_r + e_t)}{|f'|} \sim \sqrt{\epsilon_f}(|f f''|/(f')^2)^{1/2} \sim \sqrt{\epsilon_f}. \qquad (2)$$

More Accuracy

- The central difference formula for the derivative is

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

The rounding error is the same as before, but the truncation error is now of higher order: $e_t \sim h^2|f'''|$.

- The optimal value of h is

$$h \sim \left(\frac{\epsilon_f |f|}{|f'''|}\right)^{1/3} \sim (\epsilon_f)^{1/3} x_c.$$

The corresponding fractional error in $f'(x)$ is given by

$$\frac{(e_r + e_t)}{|f'|} \sim \frac{(\epsilon_f)^{2/3}|f|^{2/3}(|f'''|)^{1/3}}{|f'|} \sim (\epsilon_f)^{2/3}.$$

This is better than equation 2 by one order of magnitude or two orders of magnitude for single and double precision respectively.

Interpolation

- Classical interpolation is one-dimensional with noiseless inputs.
- We know the value of a function f at a set of points $x_0 < x_1 < x_2 < \cdots < x_N$, and want to calculate it at an arbitrary point x.
- In such circumstances, we approximate the value of f by drawing a smooth curve through the points (x_i, f_i) and compute the value of the curve at x. The pairs (x_i, f_i) are called support points.
- If x lies within $[x_0, x_N]$ this is called interpolation. If x lies outside this interval, then the procedure is called extrapolation and is likely to have much larger errors.
- In function approximation you can choose where the points x_i are, and you can improve the error bounds considerably.

Interpolation Schemes

We write $\Phi(x; a_0, \ldots, a_n)$ for a function which has $n+1$ parameters $a_0, \ldots, a_n$ and which interpolates the support points. There are three commonly used schemes that are linear in their coefficients:

Polynomials $a_0 + a_1 x + \cdots + a_n x^n$

Splines Φ is twice continuously differentiable and coincides with a cubic polynomial on each subinterval $[x_i, x_{i+1}]$

Trigonometic $a_0 + a_1 e^{ix} + \cdots + a_n e^{inx}$. This gives rise to the Fourier transform, and won't be discussed further in this course.

We also consider one scheme that is non-linear in its parameters.

Rational interpolation uses a function that is a ratio of two polynomials:

$$\Phi(x; a_0, \ldots, a_n, b_0, \ldots b_m) \equiv \frac{a_0 + \cdots + a_n x^n}{b_0 + \cdots + b_m x^m}.$$

Polynomial Interpolation

Lagrange's Theorem:

For $n+1$ arbitrary support points (x_i, f_i) where $x_i \neq x_k$ for $i \neq k$ and $i = 0, \ldots, n$, there exists a unique polynomial $P \in \Pi_n$ (the class of polynomials whose degree is at most n) with

$$P(x_i) = f_i. \tag{3}$$

Proof

Uniqueness: If P and Q both satisfy equation 3, then $P - Q$ has $n + 1$ zeroes and is a polynomial of degree at most n. This means that it must be identically zero and $P \equiv Q$.

Existence: Define polynomials $L_i \in \Pi_n$ with the property that $L_i(x_j) = \delta_{ij}$. Then

$$P(x) \equiv \sum_{i=0}^{n} f_i L_i(x)$$

has the property that

$$P(x_k) = \sum_{i=0}^{n} f_i L_i(x_k) = \sum_{i=0}^{n} f_i \delta_{ik} = f_k.$$

Practical Polynomial Interpolation

- Lagrange's construction is important theoretically, but is not as efficient or numerically trustworthy as other algorithms for practical purposes.
- In addition, if we add a new support point (x_{n+1}, f_{n+1}) then all the polynomials $L_i(x)$ have to be recomputed.
- Neville's algorithm directly evaluates the interpolating polynomial at the point of interest x.
- It starts at a support point close to x and then adding (usually decreasing) corrections until the information from all the other support points is incorporated. This typically takes $O(n^2)$ operations.
- The last correction is used as an informal error estimate.

Algorithm Details

Let $\{i_0, \ldots, i_k\}$ be a subset of $\{0, \ldots, n\}$. Let $P_{i_0 \ldots i_k} \in \Pi_k$ be the polynomial in Π_k for which $P_{i_0 \ldots i_k}(x_{i_j}) = f_{i_j}$ for $j = 0, \ldots, k$. These polynomials can be computed by the following recursion:

$$P_i(x) \equiv f_i \tag{4}$$

$$P_{i_0 \ldots i_k}(x) \equiv \frac{(x - x_{i_0}) P_{i_1 \ldots i_k}(x) - (x - x_{i_k}) P_{i_0 \ldots i_{k-1}}(x)}{x_{i_k} - x_{i_0}}. \tag{5}$$

	$k = 0$	1	2	3
x_0	$f_0 = P_0(x)$			
		$P_{01}(x)$		
x_1	$f_1 = P_1(x)$		$P_{012}(x)$	
		$P_{12}(x)$		$P_{0123}(x)$
x_2	$f_2 = P_2(x)$		$P_{123}(x)$	
		$P_{23}(x)$		
x_3	$f_3 = P_3(x)$			

- To improve the accuracy by reducing the effects of rounding errors, we define the differences between parents and children as:

$$C_{m,i} \equiv P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)}$$

$$D_{m,i} \equiv P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}.$$

  Then

$$C_{m+1,i} = \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}$$

$$D_{m+1,i} = \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}.$$

- The value that we want is $P_{0\ldots n}$, and this is computed as the sum of f_i and any set of C's and/or D's that form a path to the rightmost child from f_i.
- An implementation of this algorithm can be found on page 109–110 of Numerical Recipes.

Limitations of Polynomial Interpolation

- Error estimates for this approximation will be considered later.
- Contrary to what one might hope (expect?), the interpolant is not guaranteed to converge to the underlying function as the number of support points increases.
- Stoer and Bulirsch give the example of Runge's function

$$R(x) = \frac{1}{1 + 25x^2},$$

for which the behaviour of the interpolating polynomial becomes increasingly poor for $|x| > 0.7$ as the number of knots increases even if they partition the interval $[-1, 1]$ uniformly.