NUMERICAL COMPUTING CS2008

NATIONAL UNIVERSITY
of Computer & Emerging Sciences

Numerical Computing

Muhammad Shahid Ashraf

Errors, ALgorithms

Types of errors

- Errors in the formulation of the problem to be solved.
  - Errors in the mathematical model. Simplifications.
  - Error in input data. Measurements.
- Approximation errors
  - Discretization error.
  - Convergence error in iterative methods.
  - Discretization/convergence errors may be assessed by an analysis of the method used.
- Roundoff errors
  - Roundoff errors arise everywhere in numerical computation because of the finite precision arithmetic.
  - Roundoff errors behave quite erratic.

Discretization errors in action

Problem: want to approximate the derivative $f'(x_0)$ of a given smooth function $f(x)$ at the point $x = x_0$.

Example: Let $f(x) = \sin(x)$, $-\infty < x < \infty$, and set $x_0 = 1.2$. Thus, $f(x_0) = \sin(1.2) \approx 0.932\ldots$.

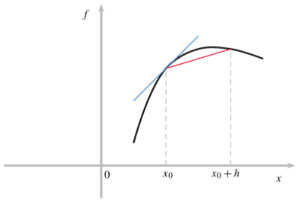
Discretization: Function values $f(x)$ are available only at a discrete number of points, e.g. at grid points $x_j = x_0 + jh$, $j \in \mathbb{Z}$.

Want to approximate $f'(x_0)$ by values $f(x_j)$.

Discretization errors in action (cont.)

Taylor's series gives us an algorithm to approximate $f'(x_0)$:

$$f'(x_0) \approx D_{x_0,h}(f) = \frac{f(x_0+h) - f(x_0)}{h}$$



Notes

Discretization errors in action (cont.)

Expanding $f(x)$ by a Taylor series around $x = x_0$ gives

$$\frac{f(x_0+h) - f(x_0)}{h} = f'(x_0) - \frac{h}{2} f''(\xi), \quad x_0 < \xi < x_0 + h.$$

So, we expect the error to decrease linearly with h.

$$\left| f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h} \right| = \frac{h}{2} |f''(\xi)| \approx \frac{h}{2} |f''(x_0)|$$

Or, using the big-O notation:

$$|f'(x_0) - D_{x_0,h}(f)| = \mathcal{O}(h).$$

Notes

Results

Try for $f(x) = \sin(x)$ at $x_0 = 1.2$.

(So, we are approximating $\cos(1.2) = 0.362357754476674\ldots$)

h	Absolute error
0.1	$4.71667 \cdot 10^{-2}$
0.01	$4.666196 \cdot 10^{-3}$
0.001	$4.660799 \cdot 10^{-4}$
10^{-4}	$4.660256 \cdot 10^{-5}$
10^{-7}	$4.619326 \cdot 10^{-8}$

These results reflect the discretization error as expected.

Note that $f''(x_0)/2 = -\sin(1.2)/2 \approx -0.466$.

Notes

Results for smaller h

The above results indicate that we can compute the derivative as accurate as we like, provided that we take h small enough.

If we wanted

$$\left| \cos(1.2) - \frac{\sin(1.2+h) - \sin(1.2)}{h} \right| < 10^{-10}.$$

We have to set $h \leq 10^{-10}/0.466$.

Notes

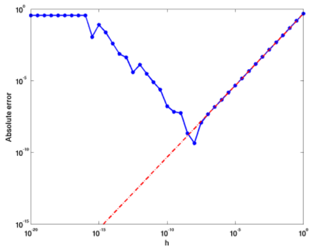
Results for smaller h

h	Absolute error
10^{-8}	$4.36105 \cdot 10^{-10}$
10^{-9}	$5.594726 \cdot 10^{-8}$
10^{-10}	$1.669696 \cdot 10^{-7}$
10^{-11}	$7.938531 \cdot 10^{-6}$
10^{-13}	$6.851746 \cdot 10^{-4}$
10^{-15}	$8.173146 \cdot 10^{-2}$
10^{-16}	$3.623578 \cdot 10^{-1}$

These results reflect both discretization and roundoff errors.

Notes

Results for all h



The solid curve interpolates the computed values of $|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}|$ for $f(x) = \sin(x)$ at $x_0 = 1.2$.

The dash-dotted straight line depicts the discretization error without roundoff error.

Notes

Algorithm properties

Performance features that may be expected from a good numerical algorithm.

- Accuracy
  Relates to errors. How accurate is the result going to be when a numerical algorithm is run with some particular input data.
- Efficiency
  - How fast can we solve a certain problem?
    Rate of convergence. Floating point operations (flops).
  - How much memory space do we need?
  - These issues may affect each other.
- Robustness
  (Numerical) software should run under all circumstances.
  Should yield correct results to within an acceptable error or should fail gracefully if not successful.

Notes

Complexity I

Complexity/computational cost of an algorithm $:\Longleftrightarrow$ number of elementary operators

Asymptotic complexity $\hat{=}$ "leading order term" of complexity w.r.t. *large* problem size parameters

The usual choice of problem size parameters in numerical linear algebra is the number of independent real variables needed to describe the input data (vector length, matrix sizes).

operation	description	#mul/div	#add/sub	
inner product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^H\mathbf{y}$	n	$n-1$	$\mathcal{O}(n)$
outer product tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$\mathcal{O}(mn)$
matrix product	$(A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}) \mapsto AB$	mnk	$mk(n-1)$	$\mathcal{O}(mnk)$

Notes

Big-O and Θ notation

For an error e depending on h we denote

$$e = \mathcal{O}(h^q)$$

if there are two positive constants q and C such that

$$|e| \leq C\,h^q \qquad \forall h > 0 \text{ small enough.}$$

Similarly, for $w = w(n)$ the expression

$$w = \mathcal{O}(n \log n)$$

means that there is a constant $C > 0$ such that

$$|w| \leq Cn \log n \qquad \text{as } n \to \infty.$$

Notes

Big-O and Θ notation

More abstract:
Class $\mathcal{O}(f)$ of functions is defined as

$$\mathcal{O}(f) = \{g \mid \exists c_1, c_2 > 0 : \forall N \in \mathbb{Z}^+ : g(N) \leq c_1 f(N) + c_2\}$$

The Θ notation signifies a stronger relation than the $\mathcal{O}$ notation: a function $\phi(h)$ for small h (resp., $\phi(n)$ for large n) is $\Theta(\psi(h))$ (resp., $\Theta(\psi(n))$) if ϕ is asymptotically bounded both above and below by ψ.

Example:
$\mathcal{O}(h^2)$ means at least "quadratic convergence" (see later). $\Theta(h^2)$ is exact quadratic convergence.

Notes

Complexity II

To a certain extent, the asymptotic complexity allows to predict the dependence of the runtime *of a particular implementation* of an algorithm on the problem size (for large problems). For instance, an algorithm with asymptotic complexity $\mathcal{O}(n^2)$ is likely to take $4\times$ as much time when the problem size is doubled.

One may argue that the memory accesses are more decisive for run times than floating point operations. In general there is a linear dependence among the two. So, there is no difference in the O notation.

Notes

Scaling

Scaling $\equiv$ multiplication with diagonal matrices (with non-zero diagonal entries) from left and/or right.

It is important to know the different effects of multiplying with a diagonal matrix from left or right:

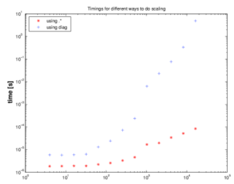
$$DA \quad \text{vs.} \quad AD \quad \text{with } \mathbb{R}^{n\times n} \ni A, D = \text{diag}(d_1, \ldots, d_n)$$

Scaling with $D = \text{diag}(d_1, \ldots, d_n)$ in MATLAB:

```
y = diag(d)*x;
```

or

```
y = d.*x;
```



Notes

Elementary matrices

Matrices of the form $A = I + \alpha \mathbf{u}\mathbf{v}^T$ are called elementary.
Again we can apply A to a vector $\mathbf{x}$ in a straightforward and a more clever way:

$$A\mathbf{x} = (I + \alpha \mathbf{u}\mathbf{v}^T)\mathbf{x}$$

or

$$A\mathbf{x} = \mathbf{x} + \alpha \mathbf{u}(\mathbf{v}^T\mathbf{x})$$

Cf. exercises.

Notes

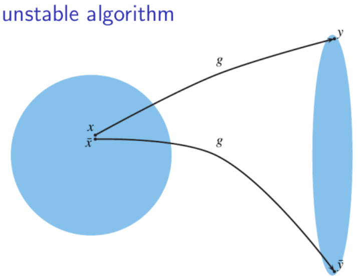
Problem conditioning and algorithm stability

Qualitatively speaking:

- The problem is ill-conditioned if a small perturbation in the data may produce a large difference in the result.
  The problem is well-conditioned otherwise.
- The algorithm is stable if its output is the exact result of a slightly perturbed input.

Notes

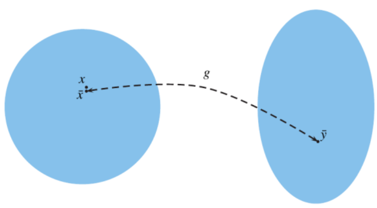
An unstable algorithm



Ill-conditioned problem of computing output values y from input values x by $y = g(x)$: when x is slightly perturbed to $\bar{x}$, the result $\bar{y} = g(\bar{x})$ is far from y.

Notes

A stable algorithm



An instance of a stable algorithm for computing $y = g(x)$: the output $\bar{y}$ is the exact result, $\bar{y} = g(\bar{x})$, for a slightly perturbed input, i.e., $\bar{x}$ which is close to the input x. Thus, if the algorithm is stable and the problem is well-conditioned, then the computed result $\bar{y}$ is close to the exact y.

Notes

Unstable algorithm

Problem statement: evaluate the integrals

$$y_n = \int_0^1 \frac{x^n}{x+10}\,dx, \quad \text{for } n = 0, 1, 2, \ldots, 30.$$

Algorithm development: observe that analytically, for $n > 0$,

$$y_n + 10y_{n-1} = \int_0^1 \frac{x^n + 10x^{n-1}}{x+10}\,dx = \int_0^1 x^{n-1}\,dx = \frac{1}{n}.$$

Also,

$$y_0 = \int_0^1 \frac{1}{x+10}\,dx = \log(11) - \log(10).$$

Algorithm:

- Evaluate $y_0 = \log(11) - \log(10)$.
- For $n = 1, 2, \ldots, 30$, evaluate $y_n = \frac{1}{n} - 10y_{n-1}$.

Notes

Unstable algorithm (cont.)

Roundoff error accumulation

- In general, if E_n is error after n elementary operations, cannot avoid linear roundoff error accumulation

$$E_n \simeq c_0 n E_0.$$

- Will not tolerate an exponential error growth such as

$$E_n \simeq c_1^n E_0, \qquad \text{for some constant } c_1 > 1.$$

This is an **unstable algorithm**.

Notes

Thankyou

Notes

Notes