Lectures on Scientific Computing

Lise-Marie Imbert-Gerard

8/30/18

1 Modeling Floating Point Error

Continuing from where we left off last time, review the example in the text. Remember that each operation generates its own value for error (ϵ_n) and that each individual error is less that the machine epsilon, as that is the upper bound of the of the relative error as a function of the machine precision.

In the end we can see that in the worst case the relative error is $\frac{\hat{x}-x}{x} - \frac{\epsilon_d}{x}$ i.e. the error is amplified by a function of $\frac{1}{x}$

In the particular case of the catastrophic cancellation that is produced by the function $\mathrm{fl}(1-\sqrt{1-\delta})$ where δ is small, we can use the substitution $x = 1-\sqrt{(1-\delta)} = \frac{\delta}{1+\sqrt{1-\delta}}$ which eliminated the problem substitution.

1.1 Exceptions

When a floating point operation attempts to calculate a non-zero number that is less than 2^{e_m} this causes an **Underflow** problem. In the calculation the use of denormalized numbers (i.e. $m = 0.d_1...d_{p-1}$) creates rounding problems. Rounding denormalized numbers creates **gradual underflow** and it leads to errors larger than ϵ_M.

A floating point operation generates an exception if the result of the operation cannot be encoded in the floating point system (i.e. is not a normalized floating point number). The IEEE standard has encodings for two exceptions:

1. $\pm\infty$
2. NotANumber (NaN)

∞ exceptions occur when the result is larger than the largest floating point number that is able to be expressed by the machine. This is called **Overflow**.

NaN exceptions occur when the floating point operation the machine is attempting to perform is invalid.

1.2 Truncation Error

Truncating is neglecting terms in an approximation formula. This is usually the main source of error in numerical integration or in solving PDEs.

example:
If we use the Taylor Expansion of a smooth function we can truncate it after the second term. i.e.

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

1.3 Iterative Methods

A **Direct Method** computes the exact answer to a problem assuming exact arithmetic. In this case the only error you will see is the round-off error, as the arithmetic is exact and only suffers errors produced by the precision of the numbers used.

An **Iterative Method** constructs a sequence A_n of approximate solutions to A (the exact solution) hoping that it converges to A. i.e. $A_n \to A$ as $n \to \infty$. In practice, some A_n for n large but finite is the approximated answer. An example of this is the Newton Method. Another example is the Fixed-Point method

Fixed Point Iteration is defined by $x_{n+1} = \Phi(x_n)$ to approximate $x = \Phi(x)$. The properties are such that the sequence must converge and the function must be continuous.

1.4 Error Propagation and Amplification

example:

Approximation of a derivative:

$f_1 := \hat{f(x)}$ $f_2 := \hat{f(x+h)}$
$e_1 = f_1 - f(x)$ $e_2 = f_2 - f(x+h)$
The derivative is defined by $\frac{f_2-f_1}{h}$.

$\frac{f_2-f_1}{h} = \frac{f(x+h)-f(x)}{h}(1 + \frac{e_2-e_1}{f(x+h)-f(x)})\frac{f(x+h)-f(x)}{h} =$
$f'(x)(1+\epsilon_t)$ Where epsilon t is the truncation error.

$\hat{f}' = \frac{f_2 - f_1}{h}(1+\epsilon_r) \to \hat{f}' = f'(x)(1+\epsilon_t)(1+\epsilon_p)(1+\epsilon_r)$

It is important to realize that the important factor in the total error is the relative sizes of the errors when doing floating point operations.

An algorithm is **unstable** if its error mainly comes from amplification. In scientific computing we use **Stability Theory** to study the propagation of a small error or small changes by a process, to search for error growth in computation. You may have already see stability analysis in use on fixed point iterations.

In a typical stability analysis we try to write a recurrance formula for the error and attempt to find where that formula converges. In other words, we focus on finding an error propagation equation such that $e_{k+1} = F(e_k)$ in order to show how e grows as k increases.

1.5 Condition Number and Ill-Conditioned Problems

The **Condition Number** of a problem measures the sensitivity of the answer with respect to small changes in the input. The larger the Condition Number, the more sensitive the function is. Condition Number is usually defined as κ. We expect that the relative error will be at lease $\kappa\epsilon_M$.

An algorithm is unstable if relative errors in the output are much larger that the relative errors in the input. An ill-conditioned problem is not going to be solved accurately unless it can be reformulated to improve the condition. For example, $1 - \sqrt{1-\delta} = \frac{\delta}{1+\sqrt{1-\delta}}$

If the output of your process ($\hat{A}$) is the exact answer to a problem that differs from the original problem only by round-off error in the input then we say that the algorithm is **Backwards Stable**. As a consequence, such algorithms cannot be more accurate than the condition number allows, as the error will be exactly dependent on the condition number.

example:

$A(x) = R\sin x \to \kappa(x) = |\cos(x)\frac{x}{\sin x}|$
$x \to 0,\ \kappa\ 1$
$x \to \pi,\ \kappa \to \infty$

Therefore, we see this problem is well conditioned around 0 but ill-conditioned around π.

There is no perfect definition of condition number for more complicated problems.