Scientific Computing
Sources of Error

Aleksandar Donev
Courant Institute, NYU[1]
donev@courant.nyu.edu

[1]Course MATH-GA.2043 or CSCI-GA.2112, Spring 2012

February 2nd, 2011

Outline

1. Sources of Error
2. Propagation of Roundoff Errors
3. Truncation Error
4. Conclusions
5. Example Homework Problem

Computational Error

Numerical algorithms try to control or minimize, rather then eliminate, the various **computational errors**:

Approximation error due to replacing the computational problem with an easier-to-solve approximation. Also called **discretization error** for ODEs/PDEs.

Truncation error due to replacing limits and infinite sequences and sums by a finite number of steps. Closely related to approximation error.

Roundoff error due to finite representation of real numbers and arithmetic on the computer, $x \neq \hat{x}$.

Propagated error due to errors in the data from user input or previous calculations in iterative methods.

Statistical error in stochastic calculations such as Monte Carlo calculations.

Conditioning

- A rather generic computational problem is to find a **solution** x that satisfies some condition for given **data** d:

$$F(x, d) = 0, \text{ for example } F(x, d) = x - \sqrt{d}$$

- **Well-posed** problem: Unique solution that depends continuously on the data.
- Otherwise it is an intrinsically **ill-posed** problem and no numerical method can help with that.
- A numerical algorithm always computes an **approximate solution** $\hat{x}$ given some **approximate data** d instead of the (unknown) exact solution x.
- An **ill-conditioned** problem is one that has a large condition number, $K \gg 1$.
- K is "large" if a given **target solution accuracy** of the solution cannot be achieved for a given **input accuracy** of the data.

Consistency, Stability and Convergence

Instead of solving $F(x, d) = 0$ directly, many numerical methods generate a sequence of solutions to

$$F_n(x_n, d_n) = 0, \text{ where } n = 0, 1, 2, \ldots$$

where for each n it is easier to obtain x_n given d.

- A numerical method is **consistent** if the approximation error vanishes as $F_n \to F$ (typically $n \to \infty$).
- A numerical method is **stable** if propagated errors decrease as the computation progresses (n increases).
- A numerical method is **convergent** if the numerical error can be made arbitrarily small by increasing the computational effort (larger n).
- Rather generally

$$\text{consistency} + \text{stability} \to \text{convergence}$$

Example: Consistency

- [*From Dahlquist & Bjorck*] Consider solving

$$F(x) = f(x) - x = 0,$$

where $f(x)$ is some non-linear function so that an exact solution is not known.

- A simple problem that is easy to solve is:

$$f(x_n) - x_{n+1} = 0 \quad \Rightarrow \quad x_{n+1} = f(x_n).$$

- This corresponds to choosing the sequence of approximations:

$$F_n\left(x_n,\, d_n \equiv x_{n-1}\right) = f(x_{n-1}) - x_n$$

- This method is consistent because if $d_n = x$ is the solution, $f(x) = x$, then

$$F_n\left(x_n,\, x\right) = x - x_n \quad \Rightarrow \quad x_n = x,$$

which means that the true solution x is a **fixed-point of the iteration**.

Example: Convergence

- For example, consider the calculation of square roots, $x = \sqrt{c}$.
- Warm up MATLAB programming: Try these calculations numerically.
- First, rewrite this as an equation:

$$f(x) = c/x = x$$

- The corresponding fixed-point method

$$x_{n+1} = f(x_n) = c/x_n$$

oscillates between x_0 and c/x_0 since $c/(c/x_0) = x_0$.
The error does not decrease and the method does not converge.

- But another choice yields an algorithm that converges (fast) for any initial guess x_0:

$$f(x) = \frac{1}{2}\left(\frac{c}{x} + x\right)$$

Example: Convergence

- Now consider the Babylonian method for square roots

$$x_{n+1} = \frac{1}{2}\left(\frac{c}{x_n} + x_n\right), \text{ based on choosing } f(x) = \frac{1}{2}\left(\frac{c}{x} + x\right).$$

- The relative error at iteration n is

$$\epsilon_n = \frac{x_n - \sqrt{c}}{\sqrt{c}} = \frac{x_n}{\sqrt{c}} - 1 \quad \Rightarrow \quad x_n = (1+\epsilon_n)\sqrt{c}.$$

- It can now be proven that the error will decrease at the next step, at least in half if $\epsilon_n > 1$, and quadratically if $\epsilon_n < 1$.

$$\epsilon_{n+1} = \frac{x_{n+1}}{\sqrt{c}} - 1 = \frac{1}{\sqrt{c}} \cdot \frac{1}{2}\left(\frac{c}{x_n} + x_n\right) - 1 = \frac{\epsilon_n^2}{2(1+\epsilon_n)}.$$

$$\text{For } n > 1 \text{ we have } \epsilon_n \geq 0 \quad \Rightarrow \quad \epsilon_{n+1} \leq \min\left\{\frac{\epsilon_n^2}{2}, \ \frac{\epsilon_n^2}{2\epsilon_n} = \frac{\epsilon_n}{2}\right\}$$

Example: (In)Stability

[From Dahlquist & Bjorck] Consider error propagation in evaluating

$$y_n = \int_0^1 \frac{x^n}{x+5} dx$$

based on the identity

$$y_n + 5y_{n-1} = n^{-1}.$$

- Forward iteration $y_n = n^{-1} - 5y_{n-1}$, starting from $y_0 = \ln(1.2)$, enlarges the error in y_{n-1} by 5 times, and is thus unstable.
- Backward iteration $y_{n-1} = (5n)^{-1} - y_n/5$ reduces the error by 5 times and is thus stable. But we need a starting guess?
- Since $y_n < y_{n-1}$,

$$6y_n < y_n + 5y_{n-1} = n^{-1} < 6y_{n-1}$$

and thus $0 < y_n < \frac{1}{6n} < y_{n-1} < \frac{1}{6(n-1)}$ so for large n we have tight bounds on y_{n-1} and the error should decrease as we go backward.

Beyond Convergence

- An algorithm will produce the correct answer if it is convergent, but...
- Not all convergent methods are equal. We can differentiate them further based on:

Accuracy How much computational work do you need to expand to get an answer to a desired relative error?
The Babylonian method is very good since the error rapidly decays and one can get relative error $\epsilon < 10^{-100}$ in no more than 8 iterations if a smart estimate is used for x_0 [see Wikipedia article].

Robustness Does the algorithm work (equally) well for all (reasonable) input data d? The Babylonian method converges for every positive c and x_0, and is thus robust.

Efficiency How fast does the **implementation** produce the answer? This depends on the algorithm, on the computer, the programming language, the programmer, etc. (more next class)

Propagation of Errors

- Assume that we are calculating something with numbers that are not exact, e.g., a rounded floating-point number $\hat{x}$ versus the exact real number x.
- For IEEE representations, recall that

$$\frac{|\hat{x} - x|}{|x|} \leq u = \begin{cases} 6.0 \cdot 10^{-8} & \text{for single precision} \\ 1.1 \cdot 10^{-16} & \text{for double precision} \end{cases}$$

- In general, the **absolute error** $\delta x = \hat{x} - x$ may have contributions from each of the different types of error (roundoff, truncation, propagated, statistical).
- Assume we have an **estimate or bound for the relative error**

$$\left| \frac{\delta x}{x} \right| \lessapprox \epsilon_x \ll 1,$$

based on some analysis, e.g., for roundoff error the IEEE standard determines $\epsilon_x = u$.

Propagation of Errors: Multiplication/Division

- *How does the relative error change (propagate) during numerical calculations?*
- For **multiplication and division**, the bounds for the **relative** error in the operands are added to give an estimate of the relative error in the result:

$$\epsilon_{x+y} = \left| \frac{(x+\delta x)(y+\delta y) - xy}{xy} \right| = \left| \frac{\delta x}{x} + \frac{\delta y}{y} + \frac{\delta x}{x}\frac{\delta y}{y} \right| \lessapprox \epsilon_x + \epsilon_y.$$

- This means that multiplication and division are **safe**, since operating on accurate input gives an output with similar accuracy.

Addition/Subtraction

- For **addition and subtraction**, however, the bounds on the **absolute** errors add to give an estimate of the absolute error in the result:

$$|\delta(x+y)| = |(x+\delta x)+(y+\delta y)-xy| = |\delta x+\delta y| < |\delta x|+|\delta y|\,.$$

- This is much more **dangerous** since the relative error is not controlled, leading to so-called **catastrophic cancellation.**

Loss of Digits

- Adding or subtracting two numbers of **widely-differing magnitude** leads to loss of accuracy due to roundoff error.
- If you do arithmetic with only 5 digits of accuracy, and you calculate

$$1.0010 + 0.00013000 = 1.0011,$$

only registers one of the digits of the small number!
- This type of roundoff error can accumulate when adding many terms, such as calculating infinite sums.
- As an example, consider computing the **harmonic sum** numerically:

$$H(N) = \sum_{i=1}^{N} \frac{1}{i} = \Psi(N+1) + \gamma,$$

where the digamma special function Ψ is *psi* in MATLAB.
We can do the sum in **forward** or in **reverse order**.

Growth of Truncation Error

```
% Calculating the harmonic sum for a given integer N:
function nhsum=harmonic(N)
    nhsum=0.0;
    for i=1:N
        nhsum=nhsum+1.0/i;
    end
end

% Single-precision version:
function nhsum=harmonicSP(N)
    nhsumSP=single(0.0);
    for i=1:N % Or, for i=N:-1:1
        nhsumSP=nhsumSP+single(1.0)/single(i);
    end
    nhsum=double(nhsumSP);
end
```

contd.

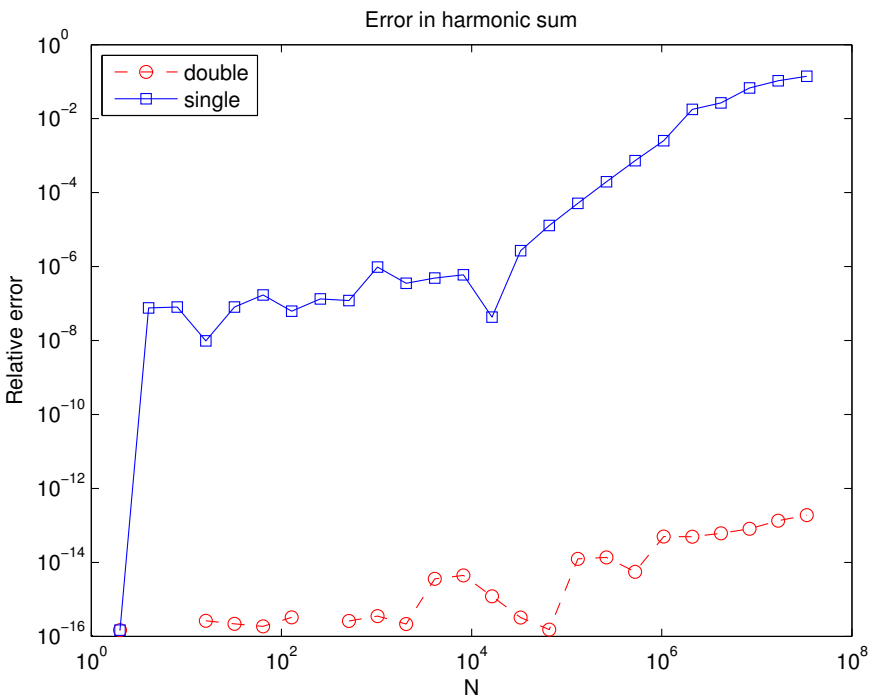
```
npts=25;
Ns=zeros(1,npts); hsum=zeros(1,npts);
relerr=zeros(1,npts); relerrSP=zeros(1,npts);
nhsum=zeros(1,npts); nhsumSP=zeros(1,npts);
for i=1:npts
    Ns(i)=2^i;
    nhsum(i)=harmonic(Ns(i));
    nhsumSP(i)=harmonicSP(Ns(i));
    hsum(i)=(psi(Ns(i)+1)-psi(1)); % Theoretical result
    relerr(i)=abs(nhsum(i)-hsum(i))/hsum(i);
    relerrSP(i)=abs(nhsumSP(i)-hsum(i))/hsum(i);
end
```

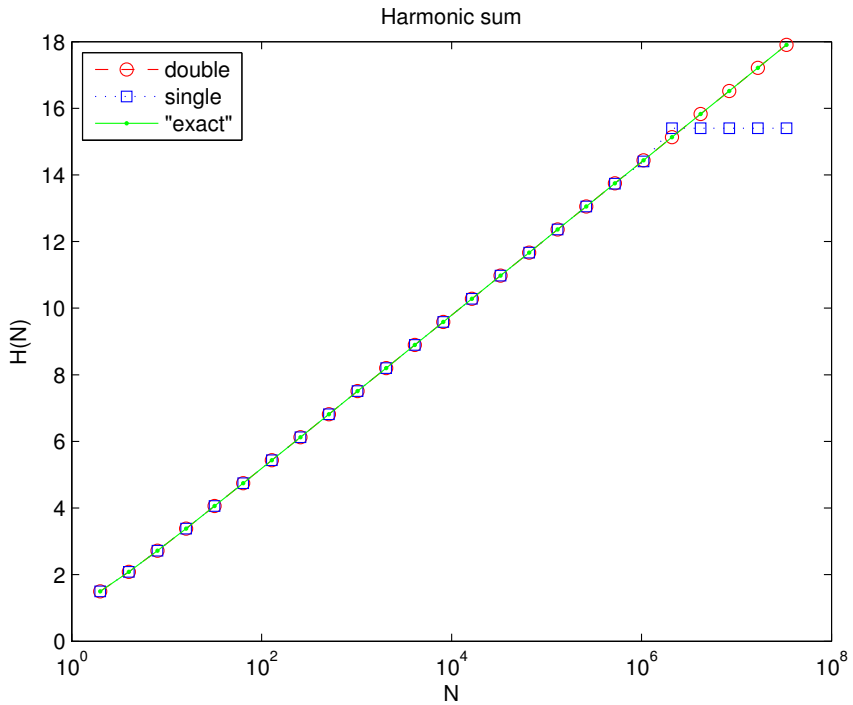
contd.

```
figure(1);
loglog(Ns,relerr,'ro--', Ns,relerrSP,'bs-');
title('Error in harmonic sum');
xlabel('N'); ylabel('Relative error');
legend('double','single', 'Location','NorthWest');

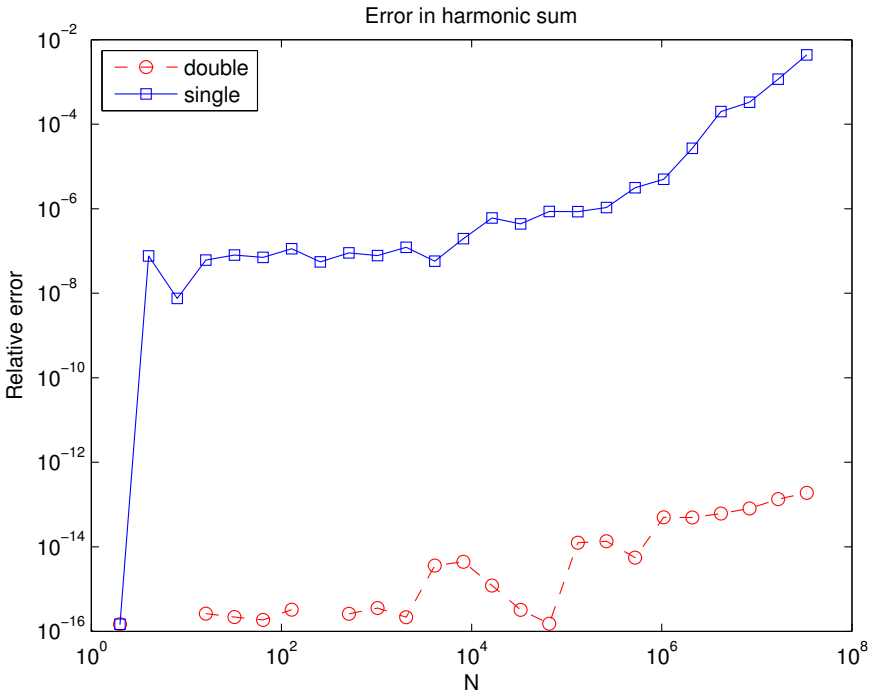
figure(2);
semilogx(Ns,nhsum,'ro--', Ns,nhsumSP,'bs:', Ns,hsum,'g.-');
title('Harmonic sum');
xlabel('N'); ylabel('H(N)');
legend('double','single','"exact"', 'Location','NorthWest');
```

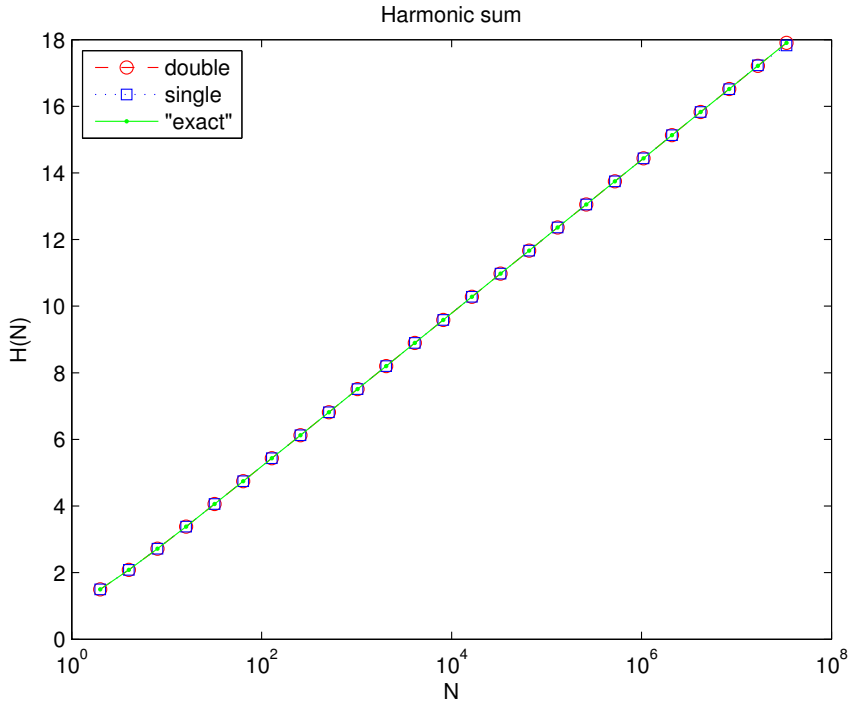
Results: Forward summation





Results: Backward summation





Numerical Cancellation

- If x and y are close to each other, $x - y$ can have reduced accuracy due to **catastrophic cancellation**.
  For example, using 5 significant digits we get

$$1.1234 - 1.1223 = 0.0011,$$

  which only has 2 significant digits!
- If gradual underflow is not supported $x - y$ can be zero even if x and y are not exactly equal.
- Consider, for example, computing the smaller root of the quadratic equation

$$x^2 - 2x + c = 0$$

  for $|c| \ll 1$, and focus on propagation/accumulation of **roundoff error**.

Cancellation example

- Let's first try the obvious formula

$$x = 1 - \sqrt{1 - c}.$$

- Note that if $|c| \leq u$ the subtraction $1 - c$ will give 1 and thus $x = 0$. How about $u \ll |c| \ll 1$.
- The calculation of $1 - c$ in floating-point arithmetic adds the absolute errors,

$$\mathrm{fl}(1 - c) - (1 - c) \approx |1| \cdot u + |c| \cdot u \approx u,$$

so the absolute and relative errors are on the order of the roundoff unit u for small c.

example contd.

- Assuming that the numerical *sqrt* function computes the root to within roundoff, i.e., to within relative accuracy of u.
- Taking the square root does not change the relative error by more than a factor of 2:

$$\sqrt{x+\delta x} = \sqrt{x}\left(1+\frac{\delta x}{x}\right)^{1/2} \approx \sqrt{x}\left(1+\frac{\delta x}{2x}\right).$$

- For quick analysis, we will simply ignore constant factors such as 2, and estimate that $\sqrt{1-c}$ has an absolute and relative error of order u.
- The absolute errors again get added for the subtraction $1-\sqrt{1-c}$, leading to the estimate of the relative error

$$\left|\frac{\delta x}{x}\right| \approx \epsilon_x = \frac{u}{x}.$$

Avoiding Cancellation

- For small c the solution is

$$x = 1 - \sqrt{1-c} \approx \frac{c}{2},$$

so the relative error can become much larger than u when c is close to u,

$$\epsilon_x \approx \frac{u}{c}.$$

- Just using the Taylor series result, $x \approx \frac{c}{2}$, already provides a good approximation for small c. Here we can do better!
- Rewriting in **mathematically-equivalent but numerically-preferred form** is the first try, e.g., instead of

$$1 - \sqrt{1-c} \text{ use } \frac{c}{1+\sqrt{1-c}},$$

which does not suffer any problem as c becomes smaller, even smaller than roundoff!

Local Truncation Error

- To recap: **Approximation error** comes about when we replace a mathematical problem with some easier to solve **approximation**.
- This error is separate and **in addition to** from any numerical algorithm or computation used to actually solve the approximation itself, such as **roundoff or propagated error**.
- Truncation error is a common type of approximation error that comes from replacing **infinitesimally** small quantities with finite **step sizes** and truncating **infinite** sequences/sums with finite ones.
- This is the most important type of error in methods for numerical interpolation, integration, solving differential equations, and others.

Taylor Series

- Analysis of local truncation error is almost always based on using Taylor series to approximate a function around a given point x:

$$f(x+h) = \sum_{n=0}^{\infty} \frac{h^n}{n!} f^{(n)}(x) = f(x) + hf'(x) + \frac{h^2}{2} f''(x) + \ldots,$$

  where we will call h the **step size**.
- This **converges** for finite h only for **analytic functions** (smooth, differentiable functions).
- We cannot do an infinite sum numerically, so we **truncate** the sum:

$$f(x+h) \approx F_p(x,h) = \sum_{n=0}^{p} \frac{h^n}{n!} f^{(n)}(x).$$

- What is the truncation error in this approximation?
  [Note: This kind of error estimate is one of the most commonly used in numerical analysis.]

Taylor Remainder Theorem

- The **remainder theorem** of calculus provides a formula for the error (for sufficiently smooth functions):

$$f(x+h) - F_p(x,h) = \frac{h^{p+1}}{(p+1)!} f^{(p+1)}(\xi),$$

  where $x \leq \xi \leq x+h$.
- In general we do not know what the value of ξ is, so we need to estimate it. We want to know what happens for **small step size** h.
- If $f^{(p+1)}(x)$ does not vary much inside the interval $[x, x+h]$, that is, $f(x)$ is sufficiently smooth and h is sufficiently small, then we can approximate $\xi \approx x$.
- This simply means that we **estimate the truncation error with the first neglected term**:

$$f(x+h) - F_p(x,h) \approx \frac{h^{p+1}}{(p+1)!} f^{(p+1)}(x).$$

The Big O notation

- It is justified more rigorously by looking at an **asymptotic expansion for small** h :
$$|f(x+h) - F_p(x,h)| = O(h^{p+1}).$$
- Here the **big** O **notation** means that for small h the error is of smaller magnitude than $\left|h^{p+1}\right|$.
- A function $g(x) = O(G(x))$ if $|g(x)| \leq C\,|G(x)|$ whenever $x < x_0$ for some finite constant $C > 0$.
- Usually, when we write $g(x) = O(G(x))$ we mean that $g(x)$ is of the same order of magnitude as $G(x)$ for small x,
$$|g(x)| \approx C\,|G(x)|\,.$$
- For the truncated Taylor series $C = \frac{f^{(p+1)}(x)}{(p+1)!}$.

Conclusions/Summary

- No numerical method can compensate for an **ill-conditioned problem**. But not every numerical method will be a good one for a **well-conditioned problem**.
- A numerical method needs to control the various **computational errors** (**approximation, truncation, roundoff, propagated, statistical**) while balancing computational cost.
- A numerical method must be **consistent** and **stable** in order to **converge** to the correct answer.
- The **IEEE standard** standardizes the **single** and **double precision floating-point formats**, their **arithmetic**, and **exceptions**. It is widely implemented but almost never in its entirety.
- Numerical overflow, underflow and cancellation need to be carefully considered and may be avoided.
  Mathematically-equivalent forms are not numerically-equivalent!

Example: Numerical Differentiation

[20 points] Numerical Differentiation

The derivative of a function $f(x)$ at a point x_0 can be calculated using finite differences, for example the first-order *one-sided difference*

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

or the second-order *centered difference*

$$f'(x = x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h},$$

where h is sufficiently small so that the approximation is good.

example contd.

1. [10pts] Consider a simple function such as $f(x) = \sin(x)$ and $x_0 = \pi/4$ and calculate the above finite differences for several h on a logarithmic scale (say $h = 2^{-m}$ for $m = 1, 2, \cdots$) and compare to the known derivative. For what h can you get the most accurate answer?
2. [10pts] Obtain an estimate of the *truncation error* in the one-sided and the centered difference formulas by performing a Taylor series expansion of $f(x_0 + h)$ around x_0. Also estimate what the *roundoff error* is due to cancellation of digits in the differencing. At some h, the combined error should be smallest (optimal, which usually happens when the errors are approximately equal in magnitude). Estimate this h and compare to the numerical observations.

MATLAB code (1)

```
format compact; format long
clear all;

% Roundoff unit / machine precision:
u=eps('double')/2;

x0 = pi/4;
exact_ans = -sin(x0);
h = logspace(-15,0,100);

%calculate one-sided difference:
f_prime1 = (cos(x0+h)-cos(x0))./h;
%calculate centered difference
f_prime2 = (cos(x0+h)-cos(x0-h))./h/2;

%calculate relative errors:
err1 = abs((f_prime1 - exact_ans)/exact_ans);
err2 = abs((f_prime2 - exact_ans)/exact_ans);
```

MATLAB code (2)

```
%calculate estimated errors
trunc1 = h/2;
trunc2 = h.^2/6;
round = u./h;

% Plot and label the results:
figure(1); clf;
loglog(h,err1,'or', h,trunc1,'--r');
hold on;
loglog(h,err2,'sb', h,trunc2,'--b');
loglog(h,round,'-g');

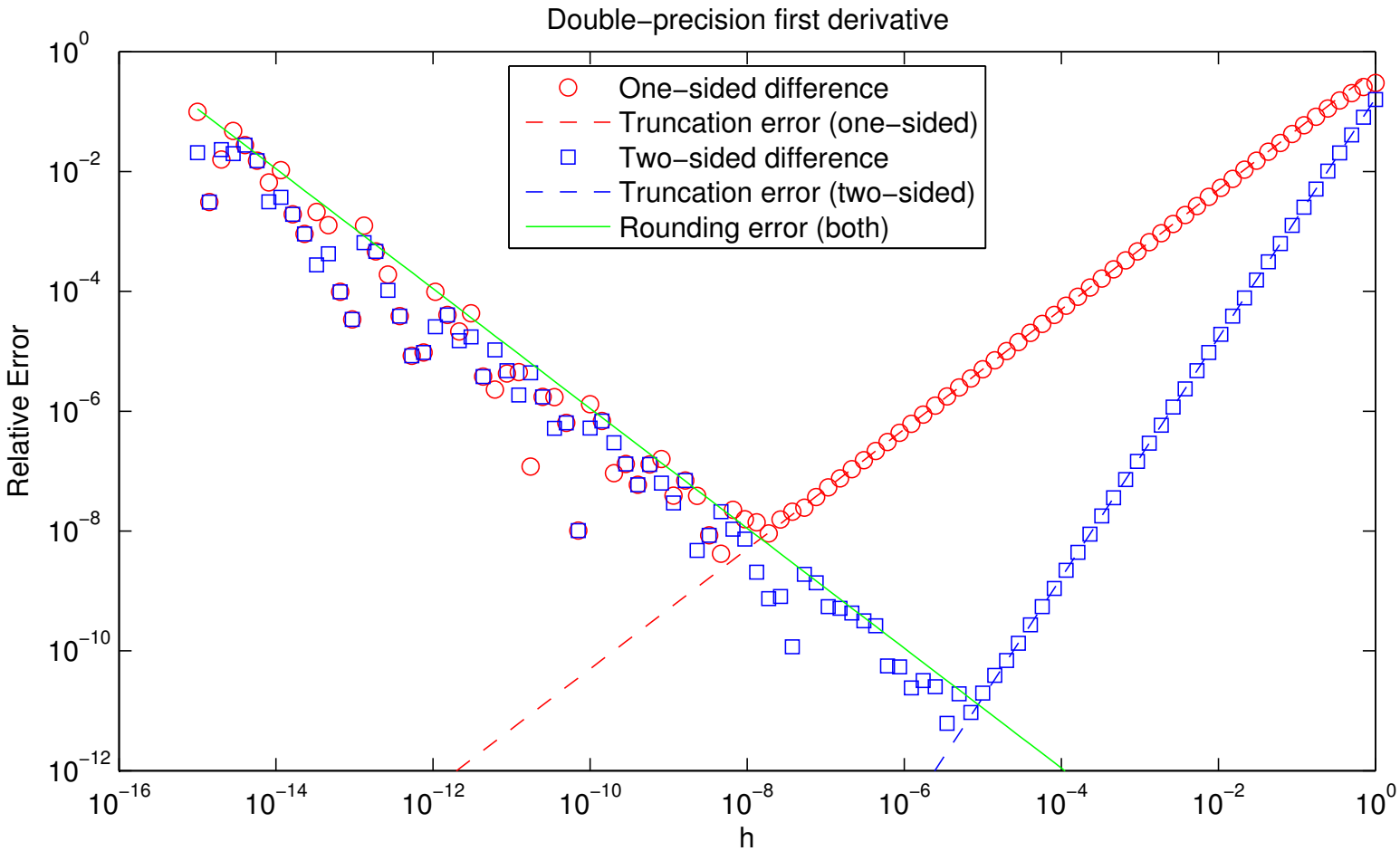
legend('One-sided difference','Truncation (one-sided)',...
       'Two-sided difference','Truncation (two-sided)',...
       'Rounding error (both)',     'Location','North');
axis([1E-16,1, 1E-12,1]);
xlabel('h'); ylabel('Relative Error')
title('Double-precision first derivative')
```

MATLAB code (2)

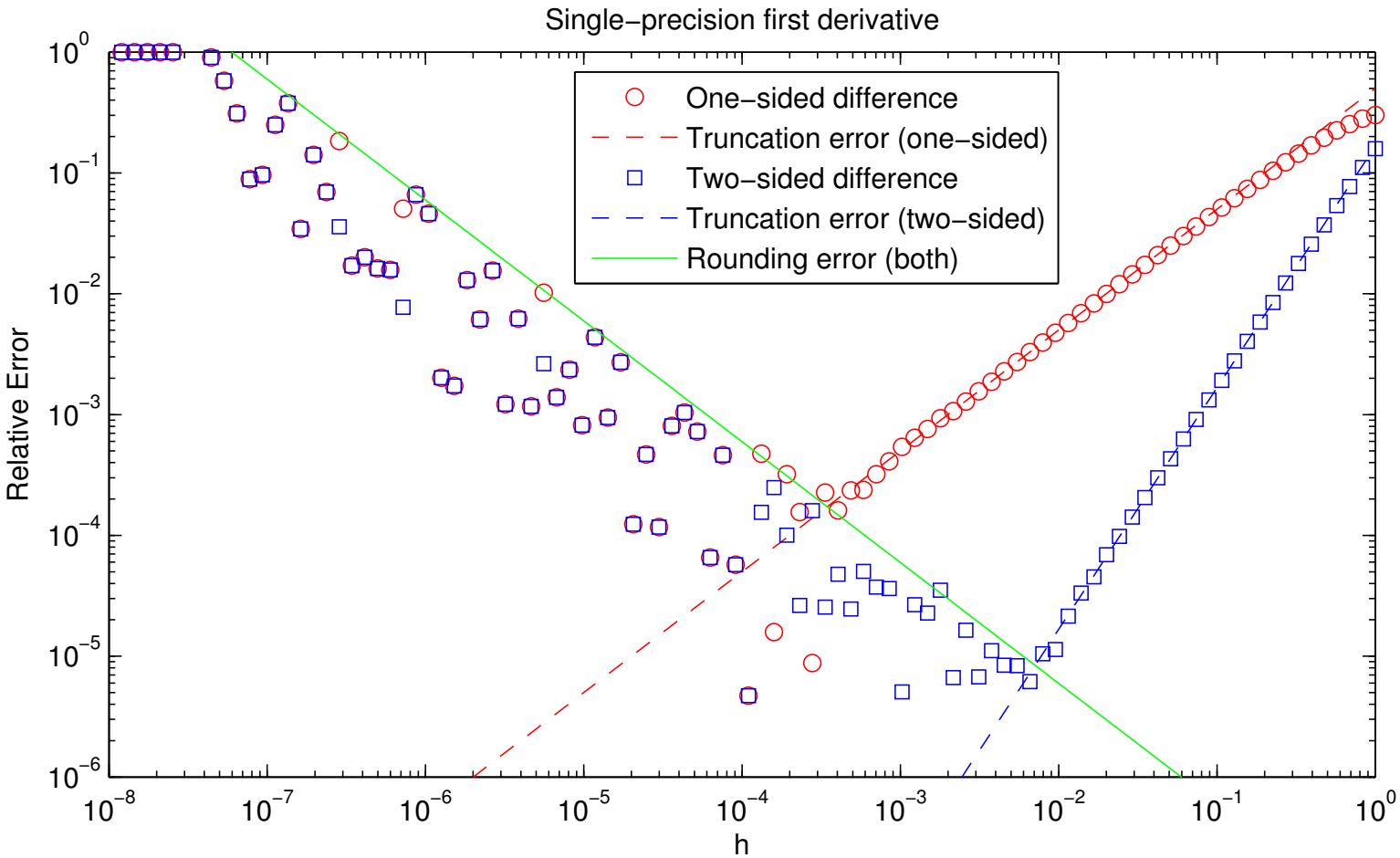
For single, precision, we just replace the first few lines of the code:

```
u=eps('single')/2;
...
x0 = single(pi/4);
exact_ans = single(-sqrt(2)/2);
...
h = single(logspace(-8,0,N));
...
axis([1E-8,1, 1E-6,1]);
```

Double Precision



Single Precision



Truncation Error Estimate

Note that for $f(x) = \sin(x)$ and $x_0 = \pi/4$ we have that

$$\left|f'(x_0)\right| = \left|f'(x_0)\right| = \left|f''(x_0)\right| = 2^{-1/2} \approx 0.71 \sim 1.$$

For the truncation error, one can use the next-order term in the Taylor series expansion, for example, for the one-sided difference:

$$f_1 = \frac{f(x_0+h) - f(x_0)}{h} = \frac{h\, f'(x_0) + h^2\, f''(\xi)/2}{h} = f'(x_0) - f''(\xi)\frac{h}{2},$$

and the magnitude of the relative error can be estimated as

$$|\epsilon_t| = \frac{|f_1 - f'(x_0)|}{|f'(x_0)|} \approx \frac{|f''(x_0)|}{|f'(x_0)|}\frac{h}{2} = \frac{h}{2}$$

and the relative error is of the same order.

Roundoff Error

Let's try to get a rough estimate of the roundoff error in computing

$$f'(x = x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}.$$

Since for division we add the relative errors, we need to estimate the relative error in the numerator and in the denominator, and then add them up.

Denominator: The only error in the denominator comes from rounding to the nearest floating point number, so the relative error in the denominator is on the order of u.

Numerator

The numerator is computed with absolute error of about u, where u is the machine precision or roundoff unit ($\sim 10^{-16}$ for double precision).
The actual value of the numerator is close to $h\,f'(x = x_0)$, so the magnitude of the relative error in the numerator is on the order of

$$\epsilon_r \approx \left| \frac{u}{h\,f'(x = x_0)} \right| \approx \frac{u}{h},$$

since $|f'(x = x_0)| \sim 1$.
We see that due to the cancellation of digits in subtracting nearly identical numbers, we can get a very large relative error when $h \sim u$.
For small $h \ll 1$, the relative truncation error of the whole calculation is thus dominated by the relative error in the numerator.

Total Error

The magnitude of the overall relative error is approximately the sum of the truncation and roundoff errors,

$$\epsilon \approx \epsilon_t + \epsilon_r = \frac{h}{2} + \frac{u}{h}.$$

The minimum error is achieved for

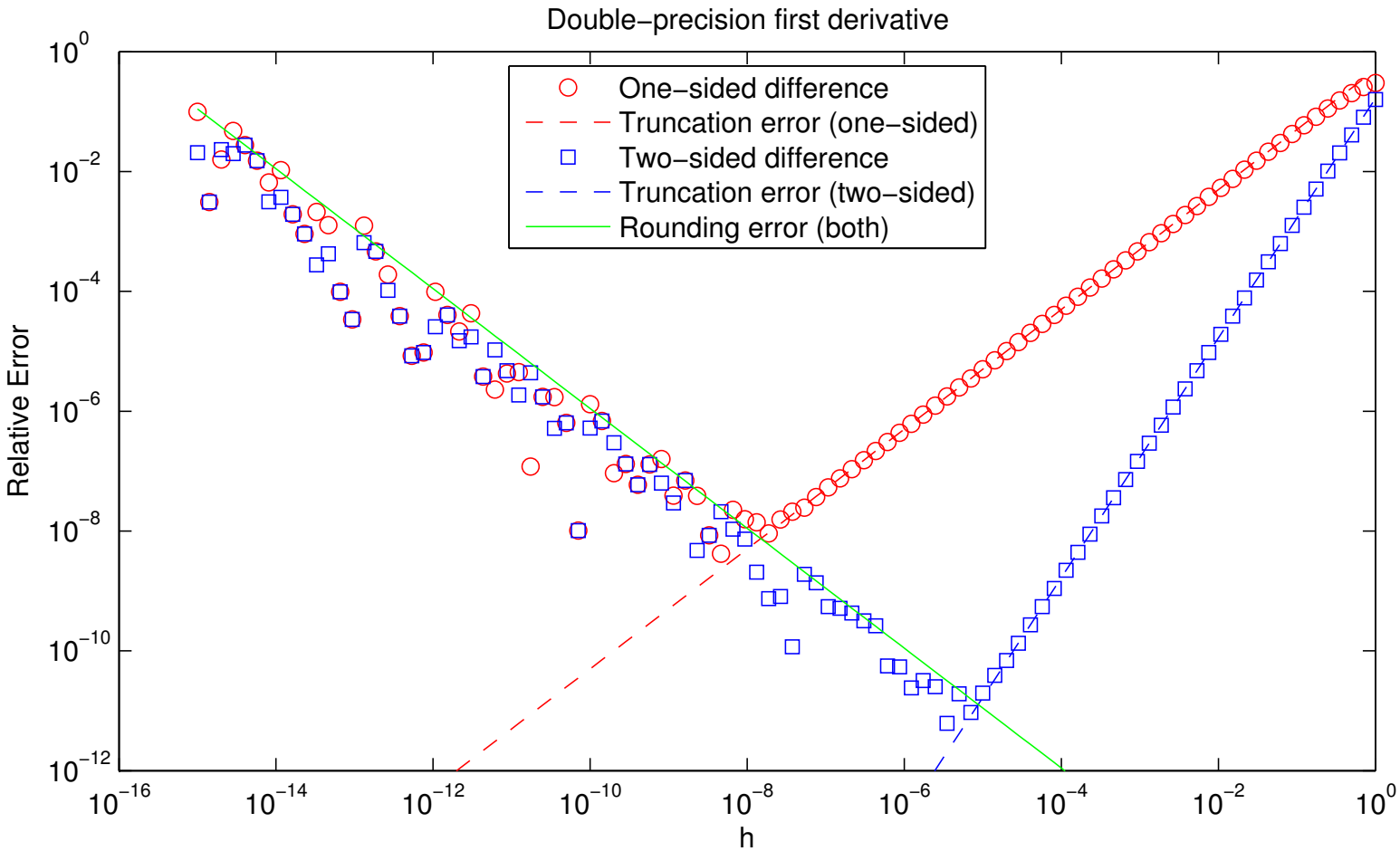
$$h \approx h_{opt} = (2u)^{1/2} \approx (2 \cdot 10^{-16})^{1/2} \approx 2 \cdot 10^{-8},$$

and the actual value of the smallest possible relative error is

$$\epsilon_{opt} = \frac{h_{opt}}{2} + \frac{u}{h_{opt}} = \sqrt{2u} \sim 10^{-8},$$

for double precision. Just replace $u \approx 6 \cdot 10^{-8}$ for single precision.

Double Precision



Two-Sided Difference

For the two-sided difference we get a much smaller truncation error, $O(h^2)$ instead of $O(h)$,

$$f'(x_0) - \frac{f(x_0+h) - f(x_0-h)}{2h} = -f'''(\xi)\frac{h^2}{6} \approx f'''(x_0)\frac{h^2}{6}.$$

The roundoff error analysis and estimate is the same for both the one-sided and the two-sided difference.
The optimal step is $h_{opt} \approx u^{1/3} \sim 10^{-5}$ for the centered difference, with minimal error $\epsilon_{opt} \sim 10^{-12}$.

Notes on Homework Submission

- Submit a **PDF of your writeup** and all of the source codes. Put all files in a **zip archive**.
- Do **not** include **top-level folders**/subfolders, that is, when we unpack the archive it should give a pdf file and **MATLAB sources**, not a directory hierarchy.
- Do not include MATLAB codes in the writeup, only the discussion and results.
- Submit only the archive file as a solution via **BlackBoard** (Assignments tab). Make sure to **press submit**.
- Homeworks must be submitted by **8am on the morning after** the listed due date (this will usually be 8am on a Monday).
- Name files **sensibly** (e.g., not "A.m" or "script.m" but also not "Solution of Problem 1 by Me.m"). Include **your name** in the filename and in the text of the PDF writeup.
- In general, one should be able to grade without looking at all the codes.

contd...

- For pen-and-pencil problems (rare) you can submit hand-written solutions if you prefer, or scan them into the PDF.
- In general, one should be able to grade without looking at all the codes. The reports should be mostly **self-contained**, e.g., the figures should be included in the writeup along with legends, explanations, calculations, etc.
- If you are using Octave, use **single quotes** for compatibility with MATLAB.
- Plot **figures** with thought and care! The plots should have axes labels and be easy to understand at a glance.
- A picture is worth a thousand words! However, a picture by itself is not enough: **explain** your method and findings.
- If you do print things, **use fprintf**.

Grading Standards

- **Points** will be **added** over all assignments (70%) and the take-home final (30%).
- **No makeup** points (solutions will be posted on BlackBoard).
- The actual grades will be rounded upward (e.g., for those that are close to a boundary), but not downward:
  - 92.5-max = A
  - 87.5-92.5 = A-
  - 80.0-87.5 = B+
  - 72.5-80.0 = B
  - 65.0-72.5 = B-
  - 57.5-65.0 = C+
  - 50.0-57.5 = C
  - 42.5-50.0 = C-
  - min-42.5 = F

Academic Integrity Policy

- If you use any external source, even Wikipedia, make sure you acknowledge it by **referencing all help**.
- It is encouraged to **discuss** with other students the mathematical aspects, algorithmic strategy, code design, techniques for debugging, and compare results.
- Copying of any portion of someone else's solution or allowing others to copy your solution is considered **cheating**.
- **Code sharing is not allowed**. You must type (or create from things you've typed using an editor, script, etc.) every character of code you use.
- Policy of financial math program:
  "A student caught cheating on an assignment may have his or her grade for the class reduced by one letter for a first offense and the grade reduced to F for a second offense."
- Submitting an **individual and independent final** is crucial and **no collaboration** will be allowed for the final.

Discussing versus Copying

- Common justifications for copying:
  - We are too busy and the homework is very hard, so we cannot do it on our own.
  - We do not copy each other but rather "work together."
  - I just emailed Joe Doe my solution as a "reference."
- Example 1: Produce an overflow.
- Example 2: Wikipedia article on Gauss-Newton method.
- Example 3: Identical tables / figures in writeup, or worse, identical codes.
- Copying work from others is not necessary and makes subsequent homeworks **harder** by increasing my expectations.
- You should talk to me if you are having a difficulty: Office hours every Tuesday 4-6pm or by appointment, or email.