1.4 Assess the conditioning of the problem of evaluating

$$g(x) = \tanh(cx) = \frac{\exp(cx) - \exp(-cx)}{\exp(cx) + \exp(-cx)}$$

near $x = 0$ as the positive parameter c grows.

Solution. Following the example in the text, let $|x| << 1$ and $\bar{x} = 0$ be a small perturbation of x. Clearly, we see that $g(\bar{x}) = 0$. Consider $g(x) - g(\bar{x})$. We have that $g(x) - g(\bar{x}) = g(x) \approx cx - \frac{(cx)^3}{3}$ by Taylor expansion of $g(x)$. Since $|x| << 1$, $cx - \frac{(cx)^3}{3} \approx cx = c(x - \bar{x})$. That is, the conditioning of this problem is linearly proportional to c. When c small, the problem is well-conditioned. However, when c becomes large, this evaluation is ill-conditioned *at* $x = 0$. See figure 1 for visual.

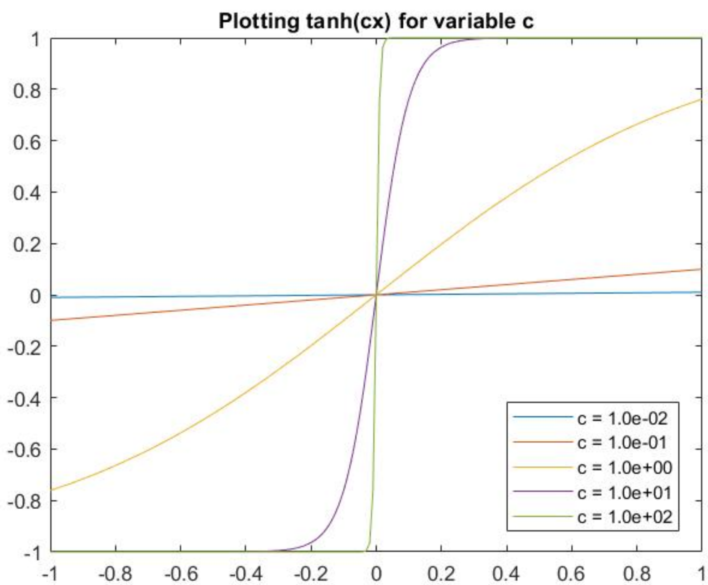


Figure 1: Conditioning of $g(x)$

1.5 (a) Derive a formula for approximately computing these integrals based on evaluating y_{n-1} given y_n.

Solution. Solving the previous equation for y_{n-1} gives us

$$y_{n-1} = \frac{1}{10}\left(\frac{1}{n} - y_n\right)$$

(b) Show that for any given value $\varepsilon > 0$ and positive integer n_0, there exists an integer $n_1 \geq n_0$ such that taking $y_{n_1} = 0$ as a starting value will produce integral evaluations y_n with an absolute error smaller than ε for all $0 < n \leq n_0$.

Proof. Fix $\varepsilon > 0$ and $n_0 \in \mathbb{N}$. Let the absolute error $|y_n - \tilde{y}_n|$ be denoted as ξ_n. Now computing ξ_{n-1}, we see that

$$y_{n-1} - \tilde{y}_{n-1} = \frac{1}{10}\left(\frac{1}{n} - y_{n-1}\right) - \frac{1}{10}\left(\frac{1}{n} - \tilde{y}_{n-1}\right) = -\frac{1}{10}\xi_n$$

So at each step, the absolute error decreases by a factor of 10. Now suppose we let $\tilde{y}_{n_1} = 0$. Then $\xi_{n_1} = y_{n_1}$. Applying this iterative process $n_1 - n_0$ times gives us

$$\xi_{n_0} = y_{n-1} \cdot \left(\frac{-1}{10}\right)^{n_1 - n_0}$$

Since y_{n-1} is fixed, it is clear that if we choose $n_1 > \log_{10}(\frac{y_{n-1}}{\varepsilon} + n_0)$, then $\xi_{n_0} < \varepsilon$. Also note that we can bound $y_n < \frac{1}{10}$ for all $n \in \mathbb{N}$. This means we further bound acceptable n_1 as $n_1 > \log_{10}(\frac{1}{\varepsilon} + n_0 - 1)$. $\square$

(c) Explain why your algorithm is stable

Solution. The algorithm is stable because at each step, the absolute error is decreased by a factor greater than 1. Thus, we can start arbitrarily high to obtain an arbitrarily small absolute error.

(d) Write a MATLAB function that computes the value of y_{20} within an absolute error of at most 10^{-5}. Explain how you chose n_1 in this case.

Solution. My $n_1 = 25$ was chosen directly from the bound previously mentioned. My code generated a true value of 0.0043470358, and approximated it to be 0.0043466709 which resulted in an absolute error of $3.649165e-07$, which was below the threshold given. See attached for code design.

2.4 Suppose a computer company is developing a new floating point system for use with their machines. They need your help in answering a few questions regarding their system.

(a) How many different nonnegative floating point values can be represented by this following point system?

Solution. The most significant digit has $\beta - 1$ different values it can take on while the other $t - 1$ digits have β. The exponent can be $U - L + 1$ different values. We also have the number 0. This results in $(\beta-1)(\beta)^{t-1}(U-L+1)+1$ different floating point values.

(b) Same question for actual choice $(\beta, t, L, U) = (8, 5, -100, 100)$.

Solution. Brute computation tells us that $7 \cdot 8^5 \cdot (201) + 1 = 45875201$.

(c) What is the approximate value of the largest and smallest positive numbers that can be represented by this floating point system.

Solution. The largest is $7.7777 \cdot 8^{100}$ and the smallest (because we are normalized) is $1.0000 \cdot 8^{-100}$.

(d) What is the rounding unit?

Solution. The rounding unit is $\eta = \frac{1}{2} \cdot 8^{(1-5)} = 2^{-13}$.

2.10 The function $f_1(x,\delta) = \cos(x+\delta) - \cos(x)$ can be transformed into another form, $f_2(x,\delta)$ using the trigonometric formula

$$\cos(\phi) - \cos(\psi) = -2\sin\left(\frac{\phi+\psi}{2}\right)\sin\left(\frac{\phi-\psi}{2}\right)$$

(a) Show that, analytically, $f_1(x,\delta)/\delta$ or $f_2(x,\delta)/\delta$ are effective approximations for $-\sin(x)$ for δ sufficiently small.

Proof. Let us consider the Taylor Expansion of $\cos(x+\delta)$ at x.

$$\begin{aligned} &\cos(x+\delta) = \cos(x) + \delta(-\sin(x)) + \mathcal{O}(\delta^2) \\ &\implies f_1(x,\delta)/\delta = (-\sin(x)) + \mathcal{O}(\delta) \\ &\implies f_1(x,\delta)/\delta \approx -\sin(x) \text{ for small } \delta \end{aligned}$$

□

(b) Derive $f_2(x,\delta)$.

Solution. Using the formula given, it is clear that

$$f_2(x,\delta) = -2\sin\left(\frac{2x+\delta}{2}\right)\sin\left(\frac{\delta}{2}\right)$$

(c) Write a MATLAB script that will calculate $g_1(x,\delta) = f_1(x,\delta)/\delta + \sin(x)$ and $g_2(x,\delta) = f_2(x,\delta)/\delta + \sin(x)$ for $x = 3$ and $\delta = 1e-11$.

Solution. See code.

(d) Explain the difference in the two calculations.

Solution. It should be noted that $g_1(x,\delta)$ and $g_2(x,\delta)$ are simply computing the absolute error for the approximations of $-\sin(x)$ by f_1, f_2. They have the same values in exact arithmetic, but we see in part (c) that they have different absolute errors. This stems from f_1 requiring the difference of $\cos(x+\delta) - \cos(x)$, two values very close in modulus. f_2 circumvents this problem by performing its equivalent $x + \delta - x$ (in the second sine) in exact arithmetic. Therefore, it makes sense that g_1 will be much greater than g_2.

2.13 Consider the linear system

$$\begin{bmatrix} a & b \\ b & a \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

with $a, b > 0$.

(a) If $a \approx b$, what is the numerical difficulty in solving this system?

Solution. When $a \approx b$, A is nearly a singular matrix and may not have a solution to $b = [1,0]'$.

(b) Suggest a numerically stable formula for computing $z = x + y$ given a and b.

Solution. Combining $ax + by = 1$ and $bx + ay = 0$ gives $z = x + y = \frac{1}{a+b}$.

(c) Determine whether the following statement is true or false, and explain why: "When $a \approx b$, the problem of solving the linear system is ill-conditioned but the problem of computing $x + y$ is not ill-conditioned."

Solution. Let $a \approx b$. When solving the linear system, it is clear that small changes in the matrix, i.e. small changes in a, b, will result in drastically different values for x, y. This is because both $x + y = \frac{1}{a+b}$ and $x - y = \frac{1}{a-b}$ must hold. The latter is particularly problematic for $a \approx b$. Solving for $z = x+y$ could also by ill-conditioned, but is not necessarily when $a \approx b$. $\frac{1}{a+b}$ is ill-conditioned when $a, b << 1$, but not otherwise. So the statement is neither true nor false. It depends on the modulus of a, b.

2.14 Consider the approximation to the first derivative

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

The truncation error for this formula is $\mathcal{O}(h)$. Suppose that the absolute error in evaluating the function f is bounded by ε.

(a) Show that the total computational error is bounded by

$$\frac{Mh}{2} + \frac{2\varepsilon}{h}$$

where M is a bound on $|f''(x)|$.

Proof. Let $\tilde{f}$ be the evaluation of f. We wish to show that the total computational error is bounded. Consider

$$\begin{aligned}
\left| \frac{\tilde{f}(x+h) - \tilde{f}(x)}{h} - f'(x) \right| &= \left| \frac{\tilde{f}(x+h) - \tilde{f}(x) + f(x+h) - f(x+h) + f(x) - f(x)}{h} - f'(x) \right| \\
&= \left| \frac{\tilde{f}(x+h) - f(x+h)}{h} \right| + \left| \frac{\tilde{f}(x) - f(x)}{h} \right| + \left| \frac{f(x+h) - f(x)}{h} - f'(x) \right| \\
&\leq \left| \frac{2\varepsilon}{h} \right| + \frac{|f''(\xi)|h}{2} + \mathcal{O}(h^2) \text{ by Taylor's approximation} \\
&\leq \frac{2\varepsilon}{h} + \frac{Mh}{2}
\end{aligned}$$

which is what we wanted to show. □

(b) What is the value of h for which the above is minimized?

Solution. Define $\xi(h) = \frac{Mh}{2} + \frac{2\varepsilon}{h}$. We see that $\xi'(h) = \frac{M}{2} - \frac{2\varepsilon}{h^2} = 0 \implies h^* = 2\sqrt{\frac{\varepsilon}{M}}$. Also because $\xi''(h^*) > 0$ this is indeed our global minimizer for a quadratic.

(c) The rounding unit we employ is approximately equal to 10^{-16}. Use this to explain the behavior of the graph in Example 1.3.

Solution. We can bound $\sin(x)$ by 1 so our $h^* = 2\sqrt{10^{-16}} = 2 \cdot 10^{-8}$. Before h^*, the $\frac{Mh}{2}$ term dominates the error as seen by the linear drop on the loglog plot. After h^*, roundoff error takes over and the error becomes significantly more erratic.

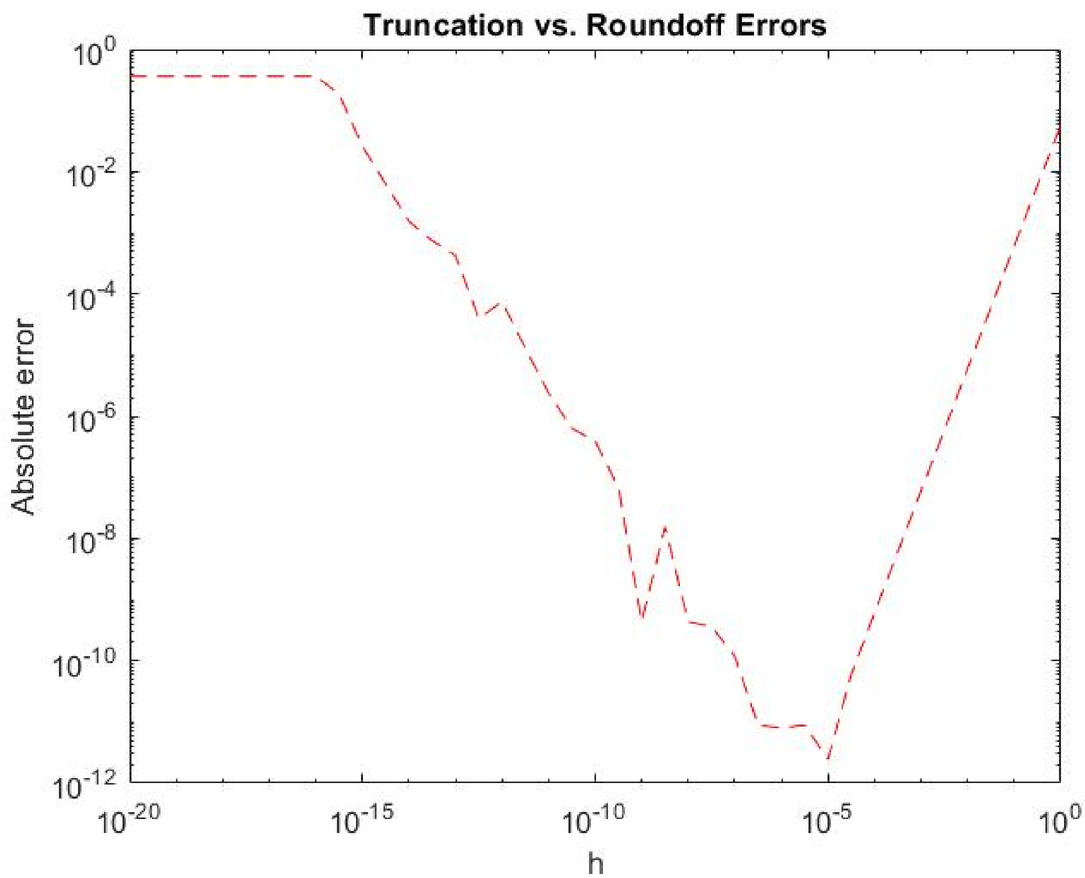
(d) It is not difficult to show using Taylor expansions that $f'(x)$ can be approximated more accurately by

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

For this approximation, the truncation error is $\mathcal{O}(h^2)$. Generate a graph similar to Figure 1.3. Explain the meaning of your results.

Solution. Figure 2 sides with our approximations. Because of the better truncation error, our error decreases at a much faster rate, but is minimized at a different h^*.

Figure 2: Two point approximation error



2.17 Write a MATLAB program that

(a) sums up $1/n$ for each $n = 1, 2, \ldots, 10,000$.

(b) rounds each number $1/n$ to 5 decimal digits and then sums them up in 5-digit decimal arithmetic for $n = 1, 2, \ldots, 10,000$.

(c) sums up the same rounded numbers (in 5-digit decimal arithmetic) in reverse order.

Solution. See code for the full MATLAB program. The three results (to 4 decimal places) were $9.7876, 9.7509$, and 9.7875 respectively. It is clear that reducing the number of digits of precision will worsen the sum; however, we found that summing the terms in reverse yielded better approximations. This is because in reverse order, we do not sum two different numbers that differ drastically in modulus, which **was** evident in the forward sum.