Numerical Applications to Differential Equations

Assignment 3

This assignment is largely intended to introduce you to the numerical accuracy and stability.

The error: Denote x an approximate solution and denote $\bar{x}$ the exact solution, thus $\bar{x} = x + \text{Error}$ and therefore

$$\text{Error} \equiv \delta x = \bar{x} - x$$

however we often interesting in only the magnitude or even just a % of the error:

$$\text{Absolute Error} \equiv |\delta x| = |\bar{x} - x| \quad \text{or} \quad \text{Relative Error} \equiv \frac{|\delta x|}{|\bar{x}|} = \frac{|\bar{x} - x|}{|\bar{x}|}$$

Source of Error: (1) Real numbers can only be approximated on computer, up to certain number of decimals. To solve it numbers either rounded (that is, 123.456 becomes 123.46.) or cut (that is, simply ignore extra digits, so that 123.456 becomes 123.45.). Thus, one get a round-off or cut-off errors. (2) Error from measurements. (3) Error from using results of prior approximations as an input.

Algorithm: Consider an algorithm as a "solver" to a given problem with an input and output.

Stability: A numerical problem may often be solved using more then one algorithm. A stable algorithm is an algorithm that satisfies

$$A(x + \delta x) = A(x) + \delta A \approx A(x),$$

i.e. a small difference in the input implies relatively small difference in the output.

1. Calculate absolute and relative errors in the following approximations for x by x_n:
   (a) $x = \pi, \quad x_n = 22/7$
   (b) $x = e, \quad x_n = 2.718$
   (c) $x = e/100, \quad x_n = 0.02718$
   (d) $x = 10^{\pi}, \quad x_n = 1400$

2. Consider that a real number is represented with only 4 digits after the decimal point (*a.k.a. 4 digits precision*), then
$$1 = \frac{1}{3} \cdot 3 \approx 0.3333 \cdot 3 = 0.9999 \neq 1$$
   (a) Suggest a solution to the problem described above.
   (b) Using Matlab, find the *smallest* n such that
$$\underbrace{\frac{1}{3} \times \frac{1}{3} \times ... \times \frac{1}{3}}_{n} \times \underbrace{3 \times 3 \times ... \times 3}_{n} \neq 1 \ .$$

3. (a) Solve the quadratic equation $x^2 + 111.11x + 1.2121 = 0$ using the formula
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
   with 5 digit precision. Note the loss of significant digits in x_1. Do this twice: First, *round* the result after each calculation step. (that is, 123.456 becomes 123.46.) Then repeat the calculations but *chop* the result after each step. (that is, simply ignore extra digits, so that 123.456 becomes 123.45.) Note that usually, rounding gives more accurate results than chopping.

(b) The loss of significant digits in x_1 in (a) can be avoided by using an alternative formula for x_1:

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}$$

Use this formula to calculate x_1 more accurately. *(this formula can be obtained from the original one by multiplying the enumerator and the denominator by: $-b - \sqrt{b^2 - 4ac}$.)*

4. Let $S_x(N) = \sum_{n=0}^{N} \frac{x^n}{n!}$. Then $\lim_{N\to\infty} S_x(N) = e^x$ for any $x \in \mathbb{R}$.

(a) Using Matlab, calculate $S_{x=10}(N)$ for N=[10:10:100]. You obtain a vector of numbers, $\vec{S}$. The elements $S(i)$ of $\vec{S}$ are partial sums that approximate the function e^x with different accuracies.

(b) Using Matlab's built-in function **exp(x)** you can calculate e^x "exactly" (i.e., with double precision accuracy). Using **exp(x)** plot the graph of relative errors

$$R_x(N) = \left| \frac{e^x - S_x(N)}{e^x} \right|$$

for $x = 10$ as a function of N.

(c) Repeat (a) and (b) with $x = -10$.

(d) Do the graphs show convergence for both values of x?

(e) Do the elements of the $\vec{S}$ converge to the correct value in both cases?

(f) Explain your answers to (d) and (e).

Hand in: answers to questions, a program used to answer the question, i.e. all m-files, and all the output it creates including plots (whenever relevant).

THE MATLAB DIGEST

Here are some additional functions in MATLAB and some useful tips:

- **if . . . elseif . . . else . . . end**. These are the MATLAB commands that perform conditional statements. Note that **elseif** is used as an **else + (new) if**, unlike C.

- ```
  while  expression
         operations
  end
  ```

  This is how MATLAB performs a loop with a conditional breaking point, i.e., the loop ends when the condition in **expression** is false, e.g., if **expression** is **1 < 0**.

- To perform finite number of steps loop use

  ```
  for index = values
       statements
  end
  ```

  the **values** above can be a range, e.g. **values=a:b**, but can also be any vector, e.g. **values = [2.5,5,17]**

- To avoid unnecessary loops which are often slow - 'vectorize' your funciton, so it can be called with a vector and return a value per each element in the input vector like for instant matlab's **sin(0:0.1:2*pi)** . To do so all relevant operations should become element-wise operators, e.g. use **.*** instead of ***.
  ```