Assignment 2

Bhishan Poudel

Sep, 2015

Contents

List of Figures

1 Care in evaluating functions

This program computes the equation

$$y = x - sinx \tag{1}$$

for a wide range of values of x.

- With a careful analysis of this function for values of x near zero, we used Taylor expansion of sine series near x equals 0.
- From the 'loss of precision theorem' we found that for $|x| < 1.9$ we need to use Taylor expansion and beyond that range we can use the general expression $xsinx$,so, we did like this.
- For $|x| < 1.9$, we used recurrence relation for the terms in the series expansion in order to avoid having to compute very large factorials.
- Here, we used single precision without Taylor expansion x near 0, and plotted the graph. Then, we used double precision and used Taylor expansion for $|x| < 1.9$.
  The compiler is bright enough that we do not see much difference between two graphs.
- The source code and outputs are following:
  for single precision : /assign2/qn1/ass2qn1Single.f90 and ass2qn1Single.dat
  for double precision : /assign2/qn1/ass2qn1Double.f90 and ass2qn1Double.dat

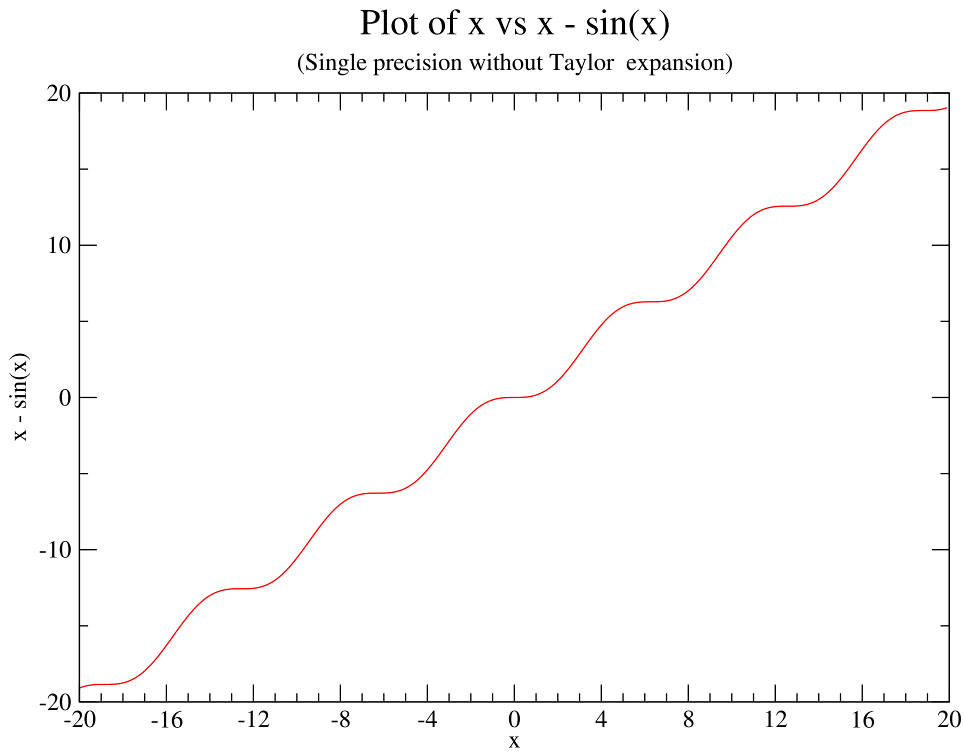

Figure 1: Plot of x vs $x - sin(x)$

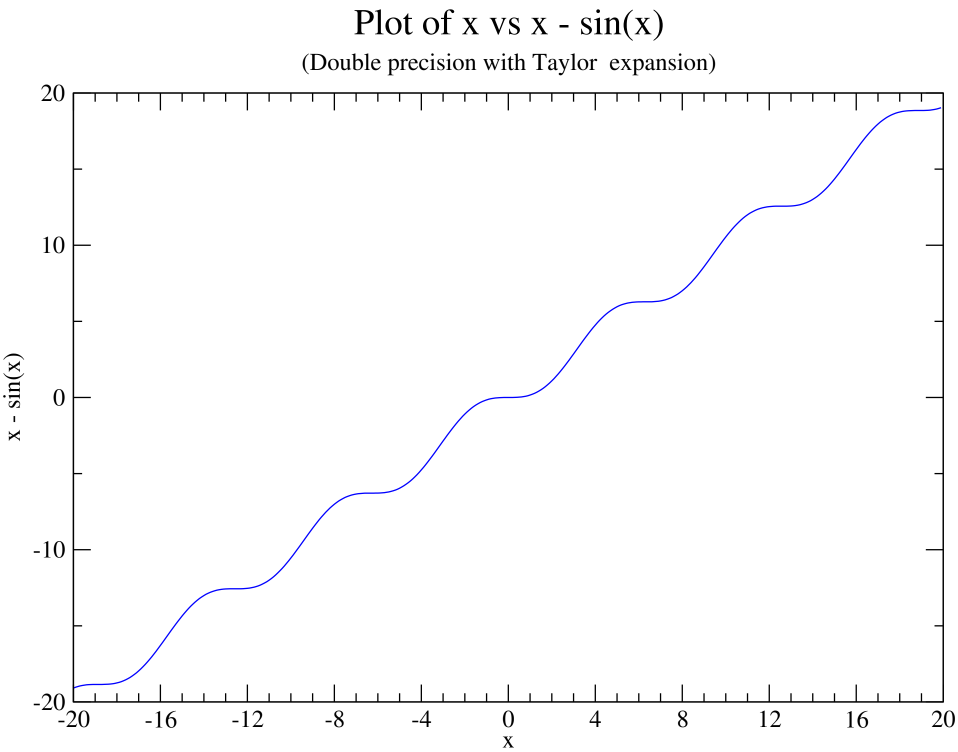

Figure 2: Plot of x vs $x - sin(x)$

2 Continued Fractions

Generally, when we encounter singular behaviors we rewrite the function in terms of a Taylor expansion. Another possibility is to use so-called continued fractions, which may be viewed as generalizations of a Taylor expansion. When dealing with continued fractions, one possible approach is that of successive substitutions. Let us illustrate this by this second order equation:

$$\begin{aligned} x^2 + 4x - 1 &= 0 \quad (2) \\ x &= 1/(4 + x) \quad (3) \\ x &= -4 + 1/x \quad (4) \end{aligned}$$

Here, second equation gives positive root and third equation gives negative root.
note: 4th iteration gives correct answer upto 5 digits after decimal
The source code and result are saved in the path:
assign2/qn2/ass2qn2.f90 and ass2qn2.dat

3 Bessel-functions via Recursion

In this question we calculated Bessel functions using upward and downward recursion relations.

3.1 part a

In this part we calculated $j_l(x)$ values for first 25 l values for x = 0.1,1.0,and 10.0. First we used single precision and this gave floating point exception beyond l=20. The code fragment is below:

```
do l = 1,19 !eg. l = 1 (when we reach l = 20, we get floating point exception)

        jup(l+1) = ( (2.d0*real(l)+ 1.d0)/x ) * jup(l)  - jup(l-1)
        write(kwrite,*) x,l+1,jup(l+1)
        end do
```

Then we used double precision. This time we did not get floating point exception. There were two options: upward and downward recursion; I chose downward recursion and normalize it in the end with known value of $j_0(x)$. The source code and results can be found here: single precision:
assign2/qn3/ass2qn3Single.f90
double precision: assign2/qn3/ass2qn3Double.f90

results for x = 0.1,1.0,and 10.0 are respectively at:
assign2/qn3/jdown_A.dat
assign2/qn3/jdown_B.dat
assign2/qn3/jdown_C.dat

3.2 part b

I tried both upward and downward recursion for spherical Bessel function. In the upward recursion relation with known value of $j_0(x)$ and $j_1(x)$, I calculated values upto $j_25(x)$ for x = 0.1. The true values for x = 0.1 can be found on Abramovitz Stegun table. I used double precision and took x = 0.1. For larger values of l, I found that downward recursion is better than upward recursion. I prepared a table like this:
x l jtrue(x) jup(x) jdown(x) rd_for_up rd_for_dn

here, jtrue(x=0.1) values are taken from Stegun book table.
rd means relative difference.
we can see the table in:
assign2/qn3/rd_wrt_true_A.dat

The source code is:
assign2/qn3/ass2qn3Double.f90

3.3 part c

In this part for downward recursion with large starting value of l, the fragment of source code looks like this:

```
jdown(51) = 2.d0 ! this value doesnot change ( if it is >= 19, we get floating point
     jdown(50) = 1.d0 ! this value doesnot change (Note: take diff values)

     do l = 50, 1,-1 !eg. l=50
         jdown(l-1) = ((2.d0*real(l)+1.d0)/x)*jdown(l)  - jdown(l+1) ! we get l-1 =
         write(2,130)x,l-1,jdown(l-1)

     end do
```

The source code and values for l = 0 to 25 for x = 0.1,1.0,10.0 respectively can be found in:
assign2/qn3/ass2qn3Double.f90
assign2/qn3/jdown_A.dat
assign2/qn3/jdown_B.dat
assign2/qn3/jdown_C.dat

3.4 part d

In this part I compared upward and downward recursion methods. I formed a table looking like this:
x l jup jdown relative difference
The table can be found in:
assign2/qn3/relative_diff_A.dat
assign2/qn3/relative_diff_B.dat
assign2/qn3/relative_diff_C.dat

Here, A,B,C are for x =0.1,1.0,and,10.0 respectively.

3.5 part e

Here, we found that the errors in the upward recursion depend on x.
Looking the results we found that:
when x = 0.1, jup and jdown values are approximately similar upto l = 4.
when x = 1.0, jup and jdown values are approximately similar upto l = 8.

when x = 10.0, jup and jdown values are approximately similar upto l = 25.

The tables can be found in:
assign2/qn3/relative_diff_A.dat
assign2/qn3/relative_diff_B.dat
assign2/qn3/relative_diff_C.dat

Here, A,B,C are for x =0.1,1.0,and,10.0 respectively. This can be explained like this: when x increases the values we take for initial values of $j_0(x)$ decreases,i.e. $j_0(0.1) > j_0(1.0) > j_0(10.0)$.

Now, the upward recursion relation is:

$$j_{l+1}(x) = ((2l+1)/x)j_l(x) - j_{l-1}(x) \tag{5}$$

Here we use recursion relation to calculate higher values of $j_l(x)$ by subtracting two terms as shown above.

There is larger error in subtracing small value from large number and smaller is the error when subtracting two similar numbers. Hence the result follows.