MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF MECHANICAL ENGINEERING
CAMBRIDGE, MASSACHUSETTS 02139

2.29 NUMERICAL FLUID MECHANICS— SPRING 2007

Problem Set 1

Posted 02/08/07, due Thursday 4 p.m. 02/23/07, from Lecture 1 to 3

Problem 1.1 (40 Points)

Solve the below problems from "Chapara and Canale" textbook:

- 1.11
- 2.13, 2.15: Print the program and the result
- 3.1, 3.7, 3.9
- 4.5, 4.8, 4.15, 4.19

Solution of Textbook Problem 1.11:

a)

Note with positive "v" as upward velocity:

$$\frac{dv}{dt} = -g(x) - \frac{c}{m}v = -(g_0 \frac{R^2}{(R+x)^2} + \frac{c}{m}v) \quad provided\ that\ v = \frac{dx}{dt} > 0$$

b)

For negligible drag force:

$$\frac{dv}{dt} = -g_0 \frac{R^2}{(R+x)^2}$$

$$\frac{dv}{dx}\frac{dx}{dt} = -g_0 \frac{R^2}{(R+x)^2}$$

$$\frac{dv}{dx}v = -g_0 \frac{R^2}{(R+x)^2}$$

c)

$$\frac{dv}{dx}v = -g_0 \frac{R^2}{(R+x)^2}$$

$$\int_{v_0}^{v} v dv = \int_{x_0}^{x} -g_0 \frac{R^2}{(R+x)^2} dx$$

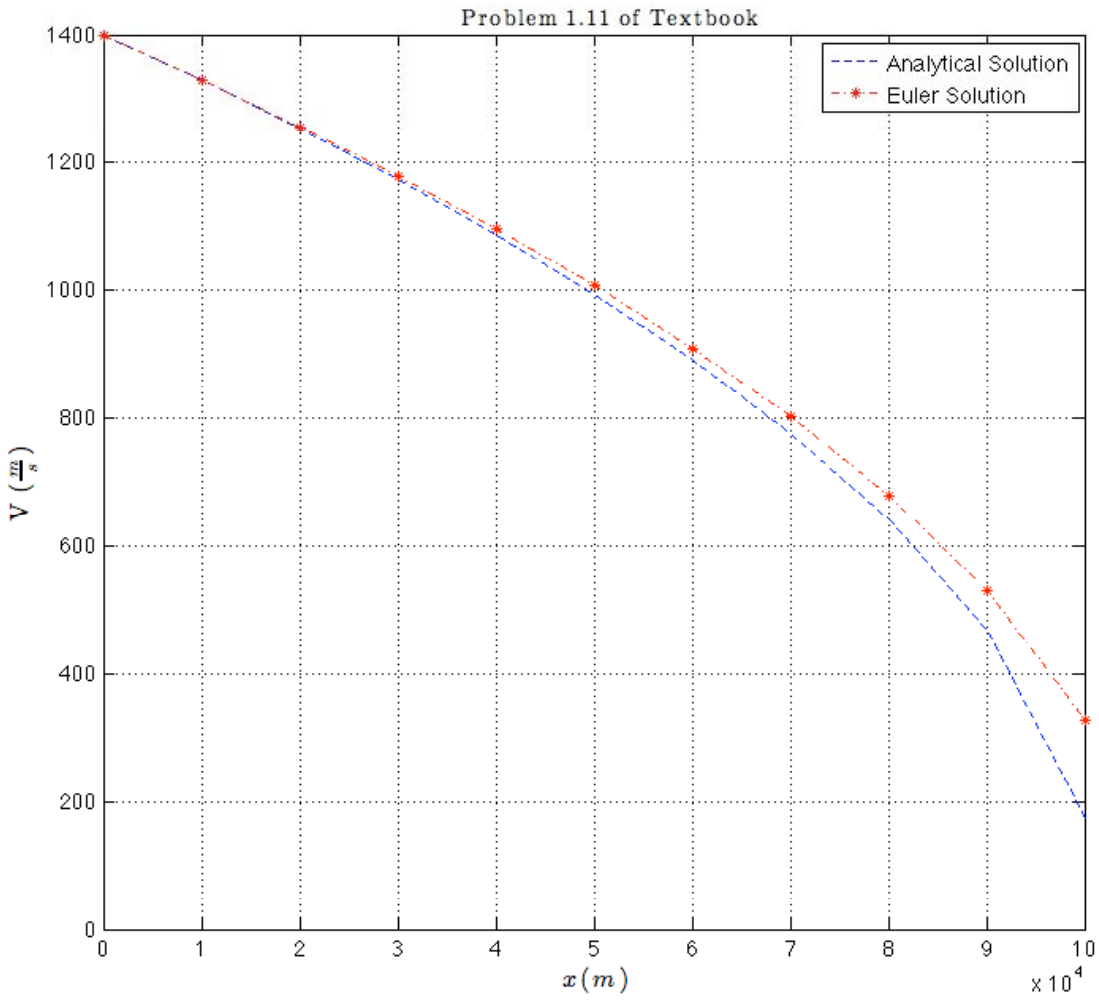
$$\frac{1}{2}(v^2 - v_0^2) = g_0 R^2 \left(\frac{1}{R+x} - \frac{1}{R+x_0}\right)$$

$$\frac{1}{2}(v^2 - v_0^2) = -g_0 R^2 \frac{x - x_0}{(R+x)(R+x_0)}$$

$$x_0 = 0 \Rightarrow v^2(x) = v_0^2 - 2g_0 R \frac{x}{R+x}$$

d)

The below plot is produced by attached file C2p29_PSET1_1p11.m; accordingly Euler's solution stands on top of analytical solution due to: $\frac{dv}{dx} = -\frac{g_0 R^2}{v(R+x)^2} < 0$ & $v(x)$: *downward convex function*. BTW, as we know and we expected the Euler's solution is only accurate for small x.



Solution of Textbook Problem 2.13:

Look at C2p29_PSET1_2p13.m file.

```
 Miami: T(0-59)          =16.214848

 Boston: T(180-242)      =22.249129
>>
```

Solution of Textbook Problem 2.15:

Look at C2p29_PSET1_2p15.m file.

Note that a logical switch exist which checks whether within each round, any numbers are swapped or not. If not, then the numbers are already sorted and an early termination can be done.

Also note after every round, we reduce the size of sort vector by one. This is because the last part of array is already sorted.

Solution of Textbook Problem 3.1:

a)

$$(101101)_2 = 1\times 2^5 + 0\times 2^4 + 1\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0$$

$$(101101)_2 = 1\times 2^5 + 1\times 2^3 + 1\times 2^2 + 1\times 2^0 = 45$$

b)

$$(101.101)_2 = 1\times 2^2 + 0\times 2^1 + 1\times 2^0 + 1\times 2^{-1} + 0\times 2^{-2} + 1\times 2^{-3}$$

$$(101.101)_2 = (101101)_2 \times 2^{-3} = \frac{45}{8} = 5.625$$

c)

$$\cos\frac{\pi}{3} = 0.5$$

$$(0.01101)_2 = \frac{13}{32} \approx 0.4062$$

Solution of Textbook Problem 3.7:

Look at C2p29_PSET1_3p7.m file.

```
Exact Value: 0.043053
Case      a: 0.050000          relative error percent 16.135925
Case      b: 0.047000          relative error percent 9.167770
```

Note that chopping should be done at any step including the summation and the powers. For example if chop3 is the function for this work then for case a and b:

```
f_a=chop3(chop3(chop3(chop3(x*chop3(x*x))-chop3(7*chop3(x^2)))+chop3(8*x))-0.35);
f_b=chop3(chop3(chop3(chop3(chop3(x-7)*x)+8)*x)-0.35);
```

Note that in general $chop(x^3) \neq chop(x \times chop(x^2))$ *or* $chop(a+b+c) \neq chop(chop(a+b)+c)$. For example:

$x = 1.37, x^2 = 1.8769, x^2 = 2.571353$

$chop3(x^3) = 2.57$

$chop3(x^2) = 1.88$

What we really get for x^3 : $chop3(chop3(x^2) \times x) = chop3(1.88 \times 1.37) = chop3(2.5756) = 2.58$

Apparently error for case "a" is higher (0.007 compared to 0.004). This can be due to the fact that we have smaller number of operations (and consequently smaller number of choppings) for case "b".

Solution of Textbook Problem 3.9:

We are interested to see that at which term, the accuracy of summation is reliable up to 8 significant digits. Since $0.3\pi \approx \frac{\pi}{3}$ and $\cos\frac{\pi}{3} = .5$, then previous condition is equivalent to saying where the summation of truncated terms is smaller than $0.5*10^{-9}$. The rigorous answer to this question comes from Taylor's remainder[1]:

$$f(x_0 + \Delta x) = f(x_0) + \sum_{i=1}^{\infty} f^{(i)}(x_0)\frac{\Delta x^i}{i!}$$

$$f(x_0 + \Delta x) = f(x_0) + \sum_{i=1}^{n} f^{(i)}(x_0)\frac{\Delta x^i}{i!} + R_n, \quad \text{where } R_n = f^{(n+1)}(\xi)\frac{\Delta x^{n+1}}{(n+1)!} \text{ for some } 0 \le \xi \le \Delta x$$

To find an upper bound for error we have to find an upper bound for $f^{(n+1)}(\xi)$. Now for f=cos(x), the series terms correspond to even n, so the (n+1) term which is zero is already included. As a result, if we have n=2k as the last term then the remainder is:

$$\text{where } R_{n=2k} = f^{(n+2)}(\xi)\frac{\Delta x^{n+2}}{(n+2)!} \text{ for some } 0 \le \xi \le \Delta x$$

By ignoring the sign of remainder, then we have to find an upper bound for $\cos(\xi)$, $0 \le \xi \le 0.3\pi$. This is when $\cos(\xi = 0) = 1$. So to find the right number of terms we have to solve the below equation:

[1] Here we use double with 15 significant digits and we only need 8 significant digits. So the error is mostly due to series truncation and predictable. However, if 8 were closer to 15 then we had comparable and non-formulabe errors due to numerical representation. In that case even if sum up to the right term in series we might not get the correct significant digits.

$$1\times\frac{x^{n+2}}{(n+2)!}\le 5\times10^{-9},\qquad where\ x=0.3\pi\ \ and\ n=2k=?$$

This can be solved with a few iterations as shown in the commandline outputs. So the n=10 have to be selected and the result is shown below. Now indeed if we compare the exact solution and series summation; we see that they at most match up to 8 significant digits for n=10 (when rounded to equal number of significant digits). However, for n=8 they at most match up to 6 significant digits which is not enough.

```
>> x=.3*pi;
>> n=6; R=1*x^(n+2)/factorial(n+2)

R =

     1.544004265784698e-05

>> n=10; R=1*x^(n+2)/factorial(n+2)

R =

     1.025454900563930e-09

>> n=8; R=1*x^(n+2)/factorial(n+2)

R =

     1.523871129688940e-07

>> % so the n=10 have to be selectd
>> power=[0:2:10]

power =

     0     2     4     6     8    10

>> Taylor_cos_x=sum(x.^(power).*(-1).^(power/2)./factorial(power))

Taylor_cos_x =

     5.877852512720046e-01

>> Exact_cos_x=cos(x)

Exact_cos_x =

     5.877852522924731e-01

>> % check n=8
>> power=[0:2:8]

power =

     0     2     4     6     8

>> Taylor_cos_x=sum(x.^(power).*(-1).^(power/2)./factorial(power))

Taylor_cos_x =

     5.877854036591176e-01
```

Exact: Rounded: 0.587,785,252

n=10, Rounded: 0.587,785,251
Accurate up to 8 significant digits

n=8, Rounded: 0.587,785,404
Accurate up to 6 significant digits

Alternatively, one might stop the summation when the last term is smaller than 5e-9. Here this approach works and ends exactly to the same as equation as above. However, as will be discussed in problem 4 as well, in general there is no warranty that when the last term is smaller than something, the summation of remaining term is also smaller than it. On the other hand, if we were restricted to rely on only 8 significant digits, then while the remaining term might contribute to the summation they will be all lost on a forward summation scheme.

Solution of Textbook Problem 4.5:

To compute the Taylor series note that:

$$f = \ln(x) \qquad f(1) = 0$$
$$f^{(1)} = \frac{1}{x} \qquad f^{(1)}(1) = 1$$
$$f^{(2)} = \frac{-1}{x^2} \qquad f^{(2)}(1) = -1$$
$$f^{(3)} = \frac{2}{x^3} \qquad f^{(3)}(1) = 2$$
$$f^{(4)} = \frac{-6}{x^4} \qquad f^{(3)}(1) = -6$$

So we can compute it around ln(1). Unfortunately as seen on the next page; the error is in fluctuation and indeed it is not converging. This is because the ln(1+x) Taylor expansion will converge only if abs(x)<1 and here (2.5-1)>1.

```
>> Coeff=[0 1 -1 2 -6];
>> dx=2.5-1;
>> for i=1:5
Order=[0:i-1];
Taylor(i)=sum(dx.^[Order].*Coeff(1:i)./factorial(Order));
end
>> Exact_Value=log(2.5)

Exact_Value =

    0.9163

>> Taylor

Taylor =

         0    1.5000    0.3750    1.5000    0.2344

>> format bank
>> Relative_Percent_Error=(Taylor/Exact_Value-1)*100

Relative_Percent_Error =

       -100.00         63.70        -59.07         63.70        -74.42
```

Solution of Textbook Problem 4.8:

```
>> syms g m c t; % Use Symbolic Toolbox
>> v=g*m/c*(1-exp(-c*t/m));
>> dv_dc=simple(diff(v,'c'));
>> pretty(dv_dc)
```

$$\frac{g\left(-m+m\exp\left(-\frac{c\,t}{m}\right)+t\exp\left(-\frac{c\,t}{m}\right)c\right)}{c^2}$$

```
>> m=50;g=9.8;t=6;c=12.5; % Parameter Values
>> V_eval=eval(v)

V_eval =

         30.45

>> dv_dc_eval=eval(dv_dc)

dv_dc_eval =

         -1.39

>> delta_c=1.5;
>> V_error_1st_Order=dv_dc_eval*delta_c

V_error_1st_Order =

         -2.08
```

Solution of Textbook Problem 4.15:

The results are displayed on the next page. Accordingly it is more sensitive to variation in "n: roughness". This is because both "n" and "S" appear as power terms but absolute value of "n" power is higher than "S" one (ratio of errors matches the ratio of powers).

```
>> syms n B H S;
>> Q=1/n*(B*H)^(5/3)/(B+2*H)^(2/3)*S^(1/2); pretty(Q)

                                  5/3  1/2
                               (B H)    S
                               --------------
                                    2/3
                               n (B + 2 H)
>> dQ_dS=simple(diff(Q,'S'));prettY(dQ_dS)

                                      5/3
                                  (B H)
                           1/2 --------------------
                                    2/3  1/2
                               n (B + 2 H)    S
>> dQ_dn=simple(diff(Q,'n'));pretty(dQ_dn)

                                    5/3  1/2
                                (B H)    S
                             - --------------
                                2           2/3
                               n  (B + 2 H)
>> B=20;H=.3;n=.03;S=.0003;
>> Q_eval=eval(Q),dQ_dn_eval=eval(dQ_dn),dQ_dS_eval=eval(dQ_dS)

Q_eval =

          1.52


dQ_dn_eval =

        -50.74


dQ_dS_eval =

       2536.85

>> delta_n=.1*n;delta_S=.1*S;
>> Error_S=dQ_dS_eval*delta_S, Error_n=dQ_dn_eval*delta_n

Error_S =

          0.08


Error_n =

         -0.15
```

Solution of Textbook Problem 4.19:

Note that:

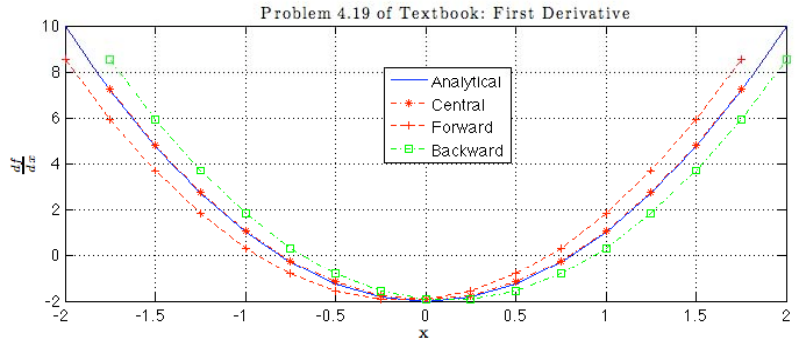
$$f = x^3 - 2x + 4$$
$$f' = 3x^2 - 2$$
$$f'' = 6x$$

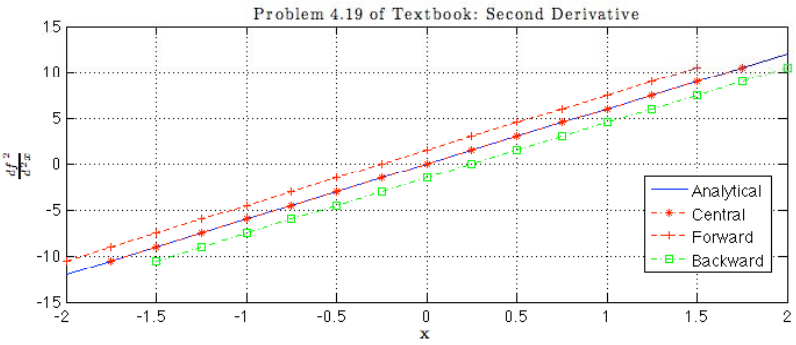
For the 2nd derivative we have (from tables on chapter 23):

$$f''(x_i) \approx \frac{f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)}{h^2} + O(h^2), \quad Central$$
$$f''(x_i) \approx \frac{f(x_{i+2}) + f(x_i) - 2f(x_{i+1})}{h^2} + O(h), \quad Forward$$
$$f''(x_i) \approx \frac{f(x_{i-2}) + f(x_i) - 2f(x_{i-1})}{h^2} + O(h), \quad Backward$$

The plot next page is generated by C2p29_PSET1_4p19.m file. As we see errors for central approximation is zero in 2nd derivatives. This relies on the fact that "f" is a 3rd order polynomial and consequently its second derivative has no 2nd order term to appear on $O(h^2)$. That's also why central 1st order derivative has very small error and also the 2nd order derivative by forward and backwards methods have constant errors.

It is also important to note that central and backward approximation curves are basically the same curve just shifted forward or backward.





Problem 1.2 (10 Points)

Review MATLAB help on these commands:

- realmin
- realmax
- eps

Now use the eps for both single and double accuracy levels and compute the below series by calling a RECURSIVE function. Compute the relative and absolute accuracy and compare it by errors associated with numerical representation in your computer.

$$2 = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + ...$$

Solution:

MATLAB HELP:

```
>> help realmax
 REALMAX Largest positive floating point number.
    x = realmax is the largest double precision floating point number
    representable on this computer.  Anything larger overflows.

    REALMAX('double') is the same as REALMAX with no arguments.

    REALMAX('single') is the largest single precision floating point number
    representable on this computer. Anything larger overflows to single(Inf).

    See also eps, realmin, intmax.


    Reference page in Help browser
       doc realmax

>> help realmin
 REALMIN Smallest positive floating point number.
    x = realmin is the smallest positive normalized double precision floating
    point number on this computer.  Anything smaller underflows or is an IEEE
    "denormal".

    REALMIN('double') is the same as REALMIN with no arguments.

    REALMIN('single') is the smallest positive normalized single precision
    floating point number on this computer.

    See also eps, realmax, intmin.


    Reference page in Help browser
       doc realmin
```

```
>> help eps
 EPS  Spacing of floating point numbers.
    D = EPS(X), is the positive distance from ABS(X) to the next larger in
    magnitude floating point number of the same precision as X.
    X may be either double precision or single precision.
    For all X, EPS(X) is equal to EPS(ABS(X)).

    EPS, with no arguments, is the distance from 1.0 to the next larger double
    precision number, that is EPS with no arguments returns 2^(-52).

    EPS('double') is the same as EPS, or EPS(1.0).
    EPS('single') is the same as EPS(single(1.0)), or single(2^-23).

    Except for numbers whose absolute value is smaller than REALMIN,
    if 2^E <= ABS(X) < 2^(E+1), then
       EPS(X) returns 2^(E-23) if ISA(X,'single')
       EPS(X) returns 2^(E-52) if ISA(X,'double')

    For all X of class double such that ABS(X) <= REALMIN, EPS(X)
    returns 2^(-1074).   Similarly, for all X of class single such that
    ABS(X) <= REALMIN('single'), EPS(X) returns 2^(-149).

    Replace expressions of the form
       if Y < EPS * ABS(X)
    with
       if Y < EPS(X)

    Example return values from calling EPS with various inputs are
    presented in the table below:

          Expression                  Return Value
         ==========================================
          eps(1/2)                    2^(-53)
          eps(1)                      2^(-52)
          eps(2)                      2^(-51)
          eps(realmax)                2^971
          eps(0)                      2^(-1074)
          eps(realmin/2)              2^(-1074)
          eps(realmin/16)             2^(-1074)
          eps(Inf)                    NaN
          eps(NaN)                    NaN
         ------------------------------------------
          eps(single(1/2))            2^(-24)
          eps(single(1))              2^(-23)
          eps(single(2))              2^(-22)
          eps(realmax('single'))      2^104
          eps(single(0))              2^(-149)
          eps(realmin('single')/2)    2^(-149)
          eps(realmin('single')/16)   2^(-149)
          eps(single(Inf))            single(NaN)
          eps(single(NaN))            single(NaN)

    See also realmax, realmin.
```

"eps" command is capable of estimating numerical round off errors, both absolute and relative one. Note that an upper bound for "round off relative error" due to numerical representation exists (also reflected in significant digits). This comes from limited number of bits associated with mantissa. Indeed for double and single precision it is 2^-52 and 2^-23 (for more information refer to http://www.hal-pc.org/~clyndes/computer-arithmetic/floats.html). Whenever we sum two numbers that their ratio is around eps, one of them will almost be totally lost.

On the other hand "realmax" and "realmin" commands show the maximum and minimum positive representable numbers with full possible significant digits. This takes into account the bits associated with exponent as well. Numbers greater than realmax be approximated infinity. Numbers below realmin start to lose their significant digits (the initial mantissa bits becomes zero). We have always to be careful if our numbers becomes around this limits. We can not exceed the realmax limit but we might be able to get non-zero numbers below realmin value (and as low as to eps*realmin, however they are suspicious values and their significant digits are questionable. The below run can better clarify this:

```
>> realmax
ans =
    1.797693134862316e+308
>> realmax*2
ans =
   Inf
>> realmin
ans =
    2.225073858507201e-308
>> realmin/2
ans =
    1.112536929253601e-308
>> realmin/2^52
ans =
    4.940656458412465e-324
>> realmin/2^53
ans =
     0
```

PROGRAM:

The attached file C2p29_PSET1_2.m provides the solution. The program adds terms as long as "S" changes due to new term. Then it continues until the last term is approximated by zero. In the program the series is represented as below:

$$S_n = \sum_{i=0}^{n} 2^{-i}, \quad \lim_{n\to\infty} S_n = 2$$

The program output is shown in the following pages. For both single and double precision the relative and absolute error reported is EXACTLY zero. However, this is only due to limited available significant digits. In other words, the error is zero only up to the significant digits. Indeed careful examination proves that for example for double precision, the summation is stationary right after 2^-53 (equal to eps('double')/2=eps(2)/4). However, the series terms are not zero and we can still have non zero represented a_i up to the 2^-1074 (note that 1074=52+1022 when 2^-1022=realmin('double') and 2^-1074=eps(0)). Similar statements can be made for single precision.

We might be interested in computation of truncation error (here fortunately it is rounded to zero). For that purpose we will use the below formula for summation of decreasing geometrical series:

$$S_n = \sum_{i=0}^{n} a_0 q^i$$

$$if\ |q| < 1 \Rightarrow \lim_{n\to\infty} S_n = \frac{a_0}{1-q}$$

Now for double precision the truncated terms begin with 2^-54. So we have:

$$Absolute\,Error = \frac{2^{-54}}{1-\frac{1}{2}} = 2^{-53}$$

$$\mathrm{Re}lative\,Error = \frac{2^{-53}}{2} = 2^{-54}$$

And as can be seen this numbers are comparable to eps('double')=2^-52. Similar calculations for single precision shows that relative and absolute errors due to truncation are 2^-25 and 2^-24.

```
  Name      Size            Bytes  Class     Attributes

  Sn        1x1                 8  double
  Sp        1x1                 8  double
  ai        1x1                 8  double

Stationary_Index

i =

    54


Sn =

     2

Final Index

i =

        1075


ai =

     0

relative_error =

     0

absolute_error =

     0
```

```
 | Name      Size            Bytes  Class     Attributes

   Sn        1x1                 4  single
   Sp        1x1                 4  single
   ai        1x1                 4  single

Stationary_Index

i =

    25


Sn =

     2

Final Index

i =

   150


ai =

     0


relative_error =

     0


absolute_error =

     0
```

Problem 1.3 (10 Points)

For a uniform inviscid flow passing a sphere with radius "R", the potential field is given by:

$$\phi(r,\theta) = U(r + \frac{R^3}{2r^2})\cos(\theta)$$

Here U is the far field velocity and the far field pressure is zero.

a) Compute the fluid velocity and plot it.
b) Discuss and evaluate the boundary conditions in infinity and in r=R.
c) Compute the pressure field and plot it.
d) Compute the drag force.

Solution:

a)

$$\vec{V} = \nabla\phi$$

$$V_r = \frac{\partial\phi}{\partial r} = U(1 - (\frac{R}{r})^3)\cos(\theta)$$

$$V_\theta = \frac{1}{r}\frac{\partial\phi}{\partial\theta} = -U(1 + \frac{1}{2}(\frac{R}{r})^3)\sin(\theta)$$

The attached file C2p29_PSET1_3.m generates a graph for fluid velocity and pressure contours. Figure is show on the next pages.

b)

Velocity at infinity (as expected to be a uniform field):

$$V_r(r \to \infty, \theta) = U\cos(\theta)$$
$$V_\theta(r \to \infty, \theta) = -U\sin(\theta)$$
$$\vec{V}(r \to \infty, \theta) = V_r e_r + V_\theta e_\theta = U e_z$$

Velocity at r=R:

$$V_r(r = R, \theta) = 0$$
$$V_\theta(r = R, \theta) = -\frac{3}{2}U\sin(\theta)$$

The radial velocity at sphere surface is zero. However, the tangential velocity is not zero because the flow is inviscid and we cannot include boundary layer effect and etc.

c)

Gravity term is ignored and steady flow with Bernoulli relation is utilized. It is taken into account that every streamline passes through infinity, and there it has zero pressure and U velocity.

$$const = \frac{1}{2}\left|\vec{V}\right|^2 + \frac{P}{\rho}$$

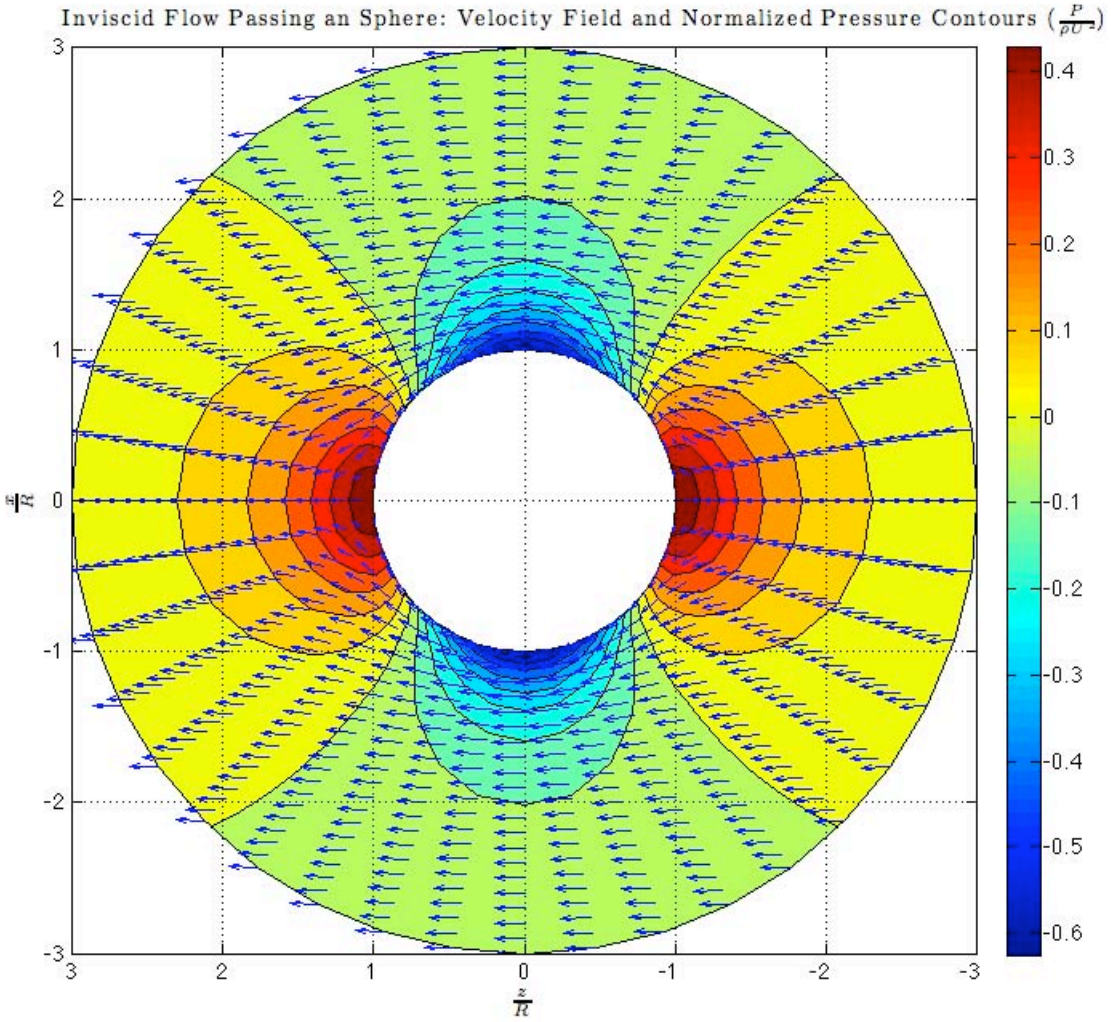
$$\frac{1}{2}U^2 + 0 = \frac{P}{\rho} + \frac{1}{2}\{[U(1 - (\frac{R}{r})^3)\cos(\theta)]^2 + [U(1 + \frac{1}{2}(\frac{R}{r})^3)\sin(\theta)]^2\}$$

$$P = \frac{1}{2}\rho U^2\{1-[(1- \quad (\frac{R}{r})^3)\cos(\theta)]^2 - [(1+\frac{1}{2}(\frac{R}{r})^3)\sin(\theta)]^2\}$$

The maximum pressure occurs at front and back stagnation points and is equal to $P(r=R,\theta=0 \text{ or } \pi)=\frac{1}{2}\rho U^2$. The minimum pressure occurs at top and bottom of sphere and is equal to $P(r=R,\theta=\pm\frac{\pi}{2})=-\frac{5}{8}\rho U^2$.

d)

The pressure has to be integrated in the sphere surface to compute the drag force. However, the net drag force will be zero due to symmetry.



Problem 1.4 (30 Points)

The sec(x) is defined as the inverse of cos(x):

$$\sec(x) = \frac{1}{\cos(x)}$$

a) Drive the Taylor series for both sec(x) and cos(x) around x=0. Note that the coefficients of sec(x)'s Taylor series are somehow complicated so you might need a program to evaluate them.

b) Write down the numerical value of 1st eight coefficients of sec(x)'s Taylor series.

c) Now compute the $\sec(\frac{\pi}{4})$ by its Taylor's series expansion. Sum up the series from the 1st term until the nth term; when the last term is smaller than "esp". Report "n" beside absolute and relative error.

d) Discuss whether the error is smaller than "eps" or not. Now repeat the part c; but stop the summation when for the first time, the last nonzero term is smaller than "esp" and is approximated by zero in your computer.

e) Repeat part c, but this time stop the summation when for the 1st time adding a nonzero term does not change the value of sec(x). Can you somehow approximate an upperbound for the error just from the numerical representation?

f) Repeat part c, d and e when sec(x) and series terms are represented by "single" data type (instead of default MATLAB data type, "double", which were used in previous parts).

g) Repeat part c, d and e; but this time, with known n, sum up the series from n to 1. Do you get better results? Discuss.

In practice the summation stops whenever either the condition e or the condition c (more conservatively d) is satisfied. From now on implement both conditions at the same time for computing both sec(x) and cos(x).

h) Compute sec(x) by inversing the cos(x). However, compute the cos(x) from its series expansion. Report "n" as well as absolute and relative errors.

i) Which way of computing sec(x) is better? Discuss. If we want to use a reverse summation scheme; can you estimate the n by which the summation should start?

j) Compare n and errors for computing sec(x) by its own series versus cos(x) series for cases when $x = \frac{\pi}{8}, \frac{3\pi}{8}, \frac{\pi}{16}, \frac{7\pi}{16}$.

Solution:

a)

The Taylor's series can be given by this formula:

$$f(x_0 + \Delta x) \approx f(x_0) + \sum_{i=1}^{\infty} f^{(i)}(x_0) \frac{\Delta x^i}{i!}$$

To compute the cos(x) Taylor's series around zero, we have to compute all order of its derivatives around zero. Fortunately this is easy for cos(x):

$$\frac{d\cos(x)}{dx} = +\sin(x)$$
$$\frac{d^2\cos(x)}{dx^2} = -\cos(x)$$
$$\frac{d^3\cos(x)}{dx^3} = -\sin(x)$$
$$\frac{d^4\cos(x)}{dx^4} = +\cos(x)$$
$$\vdots$$

Now if we evaluate above relation at x=0 we have:

$$f = \cos(x) \Rightarrow f^{(2k)}(0) = (-1)^k, f^{(2k+1)}(0) = 0 \quad k = 1,2,3,\ldots$$

Substitution of above values in the main equation leads to below Taylor expansion for cos(x). Note that convergence test proves that it will converge for all values of x.

$$\cos(x) \approx 1 + \sum_{i=1}^{\infty} (-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots \qquad for\ all\ x$$

We have to follow the same approach for finding Taylor expansion of sec(x) (which is a unique expansion). However, the evaluation of derivative of sec(x) is rather complicated. Instead here we use a recursive method by application of below derivative rule to ysec(x)=1. This seems to be one of the easiest direct ways to compute Taylor expansion of sec(x). Also, there are more advanced approaches to compute the coefficients and derive closed formula expansion for them, which we ignore due to advanced mathematic prerequisites for them.

$$f = f_1 f_2$$
$$f^{(1)} = f_1^{(1)} f_2 + f_1 f_2^{(1)}$$
$$f^{(2)} = f_1^{(2)} f_2 + 2 f_1^{(1)} f_2^{(1)} + f_1 f_2^{(2)}$$
$$f^{(3)} = f_1^{(3)} f_2 + 3 f_1^{(2)} f_2^{(1)} + 3 f_1^{(1)} f_2^{(2)} + f_1 f_2^{(3)}$$
$$\vdots$$

This formula relates the consecutive differentiation of a product to the Newton's polynomial expansion:

$$f = f_1 f_2$$
$$f^{(n)} = \sum_{r=0}^{n} \binom{n}{r} f_1^{(r)} f_2^{(n-r)}, \qquad where \binom{n}{r} = \frac{n!}{r!(n-r)!}$$

Application of above formula to 1=y*cos(x) leads to:

$$
\begin{aligned}
1 &= y\cos(x)\\
0 &= y^{(1)}\cos(x) - y\sin(x)\\
0 &= y^{(2)}\cos(x) - 2y^{(1)}\sin(x) - y\cos(x)\\
0 &= y^{(3)}\cos(x) - 3y^{(2)}\sin(x) - 3y^{(1)}\cos(x) + y\sin(x)\\
&\vdots
\end{aligned}
$$

Evaluation of above formula at x=0 leads to below relation. Note that we have omitted (x=0) from here forward:

$$
\begin{aligned}
1 &= y\\
0 &= y^{(1)}\\
0 &= y^{(2)} - y\\
0 &= y^{(3)}\\
0 &= y^{(4)} - 6y^{(2)} + y\\
0 &= y^{(5)}\\
0 &= y^{(6)} - 15y^{(4)} + 15y^{(2)} - y\\
&\vdots
\end{aligned}
$$

This leads to the recursive formula[2]:

$$
\begin{aligned}
y^{(2k+1)} &= 0\\
y^{(2k)} &= -\sum_{i=1}^{k}\binom{2k}{2i}(-1)^i y^{(2k-2i)}
\end{aligned}
$$

b)

The attached script C2p29_PSET1_4.m computes the coefficients. Please note that the odd coefficients are zero as we expected from the function being an even function.

$$
\begin{aligned}
&k = 0,1,2,3,\ldots\\
&y^{(2k)} = [1 \;\; 1 \;\; 5 \;\; 61 \;\; 1385 \;\; 50521 \;\; 2702765 \;\; \ldots]\\
&\frac{y^{(2k)}}{(2k)!} = [1 \;\; 1 \;\; 0.5 \;\; 0.208\overline{3} \;\; 0.0847\overline{2} \;\; 0.034350198412698 \;\; 0.013922233245150 \;\; \ldots]
\end{aligned}
$$

So the sec(x) Taylor's expansion can be computed:

$$
\sec(x) \approx \sum_{i=0}^{\infty} y^{(2i)}\frac{x^{2i}}{(2i)!} = 1 + \frac{x^2}{2!} + \frac{5x^4}{4!} + \frac{61x^6}{6!} + \frac{1385x^8}{8!}\cdots \qquad \textit{Converging for } |x| < \frac{\pi}{2}
$$

c)

We know the exact solution:

[2] Here we can define Euler number E_k where: $y^{(2k)} = (-1)^k E_k$. For more information refer to eg http://en.wikipedia.org/wiki/Trigonometric_function.

$$\sec(\frac{\pi}{4}) = \frac{1}{\cos(\frac{\pi}{4})} = \sqrt{2}$$

The program output is shown on the next page. For example in part c, the summation stops at 2k=56 and we have a relative error of order -4.71e-16. Please note that with modulation of script, it is rather easy to run various variations of it for incoming parts of problem.

Also please note that due to limited significant digits, we have some errors for combinatorial function when 2k>15 and also some error for factorial function when 2k>21.

d)

Assume that "f" is the value of function. Then in general the error could be larger than eps(f)[3], because while the last term is smaller than eps(f) the summation of truncated terms could be bigger than eps(f) (e.g. in previous problem the summation of truncated terms was two times the first truncated term). However, since the series usually converge very fast, the error will be of order eps(f) , where f is the value of function. Here eps(f)=2.22e-16 and absolute error is -6.66e-16. Also the relative error (here -4.71e-16) is comparable with eps=esp(1)=2.22e-16.

Since the series convergence is rather slow, we see that for last term to be approximated to zero; we have to go to 2k=174 (compare with case c: when 2k=56). Unfortunately the new terms (covering almost from 0.5*eps to realmin) do not bring any improvement because all of them are smaller than eps(f) and all of them will be truncated in the forward summation.

e)

Here we get similar results to part c. This is because the eps(1) is equal to eps(f) for f=1.41<1+1/2. However, in general the summation stops whenever, we do not have an improvement for the summation (the last term is smaller than 0.5*eps(f)) or the last term becomes smaller than realmin.

f)

The relative error in all three cases is equal to esp('single'). However, we need to compute fewer terms. For example in part c for double we need 2k=56 and here 2k=26.

g)

In all cases the relative error is divided by four while using the same number of terms. Indeed in general it is better to first sum over smaller number. However, we should be aware that:

- This needs higher amount of memory, because a lot of terms are computed with forward recursion (like factorial, power terms and derivativs) and then the summation is done with backward recursion. Here the memory constraint is ignorable for vectors with size about 50.
- The other concern is that where we have to start the backward recursion. An initial guess could be when the last term is of order eps(f). However, even then the order is not trivial to find. Here the order becomes 2k=56 from case c. As shown in the program output even if we start with 2k=76 we do not get any improvement.

[3] Since f=1.41 is very close to one, the value of f*eps=f*eps(1)=3.14e-16 is of the same order as eps(f)=2.22e-16=eps(1).

h)

With only 11 terms (2k=20), the cos(x) series converges and the absolute and relative errors are zero up to all 15 available significant digits.

i)

Clearly cos(x) expansion has a faster convergence (due to lower coefficients and alternating signs). Indeed as will be seen in the next part this is true for other values of x and especially when x gets close to $\frac{\pi}{2}$ (the convergence limit of sec(x) function). So if we ignore the computational cost of division (which is less than computing Euler's numbers!) the computation with 1/cos(x) is faster.

Also as discussed in part g, the inverse summation can improve the accuracy. However, it does not help cos(x) series, because its terms drop very quickly (by a $\frac{x^2}{(2k-1)(2k)}$ factor). On the other hand we see that the error for sec(x) is dropped by a factor of ¼ for backward summation.

In part g we discussed how we can find the last term (equal to realmin or eps(f)). In general computing the right n (here 2k) is difficult, but below relation can help for estimation of $n!$:

$$n! \approx \sqrt{2\pi n}\ (\frac{n}{e})^n$$

j)

Exact sec(x) value at $x = \frac{\pi}{8}, \frac{3\pi}{8}, \frac{\pi}{16}, \frac{7\pi}{16}$ can be computed from below relation:

$$\cos^2 x = \frac{1+\cos 2x}{2}$$

So we have:

$$\cos\frac{\pi}{8} = \sqrt{\frac{1+\cos 2\frac{\pi}{8}}{2}} = \sqrt{\frac{1+\cos\frac{\pi}{4}}{2}} = \sqrt{\frac{1+\frac{1}{\sqrt{2}}}{2}} = \sqrt{\frac{1+\sqrt{2}}{2\sqrt{2}}}$$

This can also be used to compute $\cos\frac{\pi}{16}$. Then we have:

$$\cos\frac{3\pi}{8} = \cos(\frac{\pi}{2} - \frac{\pi}{8}) = \sin\frac{\pi}{8} = \sqrt{1-\cos^2\frac{\pi}{8}}$$

$$\cos\frac{7\pi}{16} = \sqrt{1-\cos^2\frac{\pi}{16}}$$

The numerical results are shown on the next page. Accordingly sec(x) has a slower convergence compared with cos(x). When x gets close to $\frac{\pi}{2}$, the sec(x) series has too much error and still we need to compute a lot of terms. For example for $x = \frac{7\pi}{16}$ cos(x) converges with 2k=24 and 1.55e-15 as its relative error. On the other hand, sec(x) has 2k=174 and its relative error is 5 order of magnitude larger

and equal to -1.12e-10. More surprisingly if we use 2k=24 for sec(x), then we have unacceptable relative error of 3.29%.

```
Case -- 2k ----- f ---------------------- relative error ----------------- absolute error  -------

x=pi/4 Exact Value of Double f=1.414213562373095
x=pi/4 Exact Value of Single f=1.4142135

 c)     56      1.414213562373094       -4.710277376051325e-16          -6.661338147750939e-16
 d)     174     1.414213562373094       -4.710277376051325e-16          -6.661338147750939e-16
 e)     56      1.414213562373094       -4.710277376051325e-16          -6.661338147750939e-16
f+c)    26      1.4142137               +6.7179428e-08                  +9.5006058e-08
f+d)    38      1.4142137               +6.7179428e-08                  +9.5006058e-08
f+e)    28      1.4142137               +6.7179428e-08                  +9.5006058e-08
g+c)    56      1.414213562373095       -1.570092458683775e-16          -2.220446049250313e-16
g+c)    76      1.414213562373095       -1.570092458683775e-16          -2.220446049250313e-16
g+d)    174     1.414213562373095       -1.570092458683775e-16          -2.220446049250313e-16
g+e)    56      1.414213562373095       -1.570092458683775e-16          -2.220446049250313e-16
 h)     20      1.414213562373095       0.000000000000000e+00           0.000000000000000e+00

x=pi/8 Exact Value of Double f=1.082392200292394

sec j)  30      1.082392200292394       -4.102849315895826e-16          -4.440892098500626e-16
cos j)  16      1.082392200292394       +0.000000000000000e+00          +0.000000000000000e+00

x=3pi/8 Exact Value of Double f=2.613125929752753

sec j)  130     2.613125929752754       +5.098367493051679e-16          +1.332267629550188e-15
cos j)  22      2.613125929752753       +1.699455831017226e-16          +4.440892098500626e-16

x=pi/16 Exact Value of Double f=1.019591158208318

sec j)  20      1.019591158208318       -2.177780801034214e-16          -2.220446049250313e-16
cos j)  14      1.019591158208318       -2.177780801034214e-16          -2.220446049250313e-16

x=7pi/16 Exact Value of Double f=5.125830895483004

sec j)  174     5.125830894907060       -1.123610093856087e-10          -5.759437371466447e-10
cos j)  24      5.125830895483012       +1.554312234475219e-15          +7.993605777301127e-15
g)      24      4.957094715791831       -3.291879563172229e-02          -1.687361796911730e-01
>>
```