Numerical Computing in C & C++

REPORT

Waqil Rahman

Contents

Preface:

In order to avoid repetition of explanations within this report and comments on the .cpp files I have written below a series of explanations. These explanations are to explain parts of code which are ‘basic’ but is needed to understand in order to maintain a high level of accuracy when the each code is being run.

static_cast long double (‘expression’) – This particular bit of code allows to avoid accuracy loss when the expression takes a value in the form of int and needs to be redefined as a long double. This is because if we were carry out integer division we would lose the decimal point values as it wont be stored.

using namespace std cout cin

Const

#include <iostream>

#include <cmath>

exp

Question 1:

Code:

```
#include <iostream>
#include <cmath>
#include <iomanip>
#define EP 1E-15
//this defines the epsilon
using namespace std

int main

  long double        1.0
  //initial guess is defined

  for  int      0         2.0
  //for loop will continue until the value of x is more than 2(this won't occur due to the nature of the
function
    long double       exp
    //This outputs the result of the function
    long double                     1.0
    //This long double stores number of iterations until the end of the loop or if the loop breaks

    if  abs                  EP
    //This if statement sets the conditions for the final value of x
      cout      setprecision 18       "The final value of x: "              endl
      //This outputs to 18 digits the final value of x
      cout      "The number of iterations: "                        endl
      //This outputs the number iterations occurred stored from the double
      long double                          exp
      //This calculates the error
      cout      "The error in the transcendental equation is: "                        endl
      break
      //This breaks the entire for loop


    //This re-assignment of doubles allows for the recursive iteration to loop with the new value


  return 0
```

Output:

A) From the code output, we know the answer is

B) Since we implemented the long double variable ‘iterations’ which would increase by 1.0 every time the ‘*if*’ loop is executed. When the loop is broken from the *break* function the final value of the number of iterations is 61(as we output the long double ‘iterations’) which is shown in the code output.

C) We calculate the transcendental error by using the equation mentioned in the question by inputting our final value of x. The error is: 9.93247024847598220 8e-16, this is what I expected as I had programmed the ‘if’ loop to break when |x_new – x_old| is less than or equal to the stated epsilon. In this case, it will always be less than epsilon as it would be very improbable for x to be equal to 1e-15.

Question 2:

Code:

```
#include <iostream>
#include <cmath>
#include <valarray>
#include <iomanip>
using namespace std

long double dot const valarray  long double      const valarray  long double
    return         sum
    //This valarray calculation returns the sum of A*B

//This answers q2a

//The function cdot is defined in week10 notes - compensated summation.cpp - KahanSum function
long double cdot const valarray  long double
    long double           0.
    long double       0.
    long double
    long double
    for int      0         size





    return


class normal_cal
//using class allows to extract different components out the function
    int
public
    normal_cal int        m
    double operator    const valarray  long double       const
    //This initialises the valarray into the class and uses function object to call on the valarray
        long double                 0.
        //We let normal be zero as we are going to add to it
        for  int      0            size
        //the for loop will go through the entirety of the valarray
                          pow abs         m
            //This double computes summation part of the normal
         return pow static_cast  long double                 0.5
        //The function will now return the final value of the normal



int main
    cout     setprecision 20
    //We use the setprecision function to see the accuracy of the programme outputs
    long double       pow 10.0 6.0
    //Defined from question to 10^6 from library cmath
    valarray  long double
    //We initialise valarray a with size n
    for  double      1          size
        //This creates the valarray of the Euclidean n-valarray
            1  static_cast  long double
```

```
    cout      "Product of dot: "      dot          endl
    //Using the dot function we sum A and A together
    long double       3.14159265358979323851
    cout      "Difference in dot and the actual answer: "      dot          pow     2.0  6.0       endl
    //This answers q2b

    valarray  long double     0.1
    //This is the constant Euclidean constant n-valarray for the dot function
    valarray  long double           0.01
    //This is the constant Euclidean constant n-valarray for the dot function
    //We have to use 0.01 as that is c^2
    long double          10000.0
    // 10000 = (0.1^2)*(10^6)
    // We use the numerical exact value instead to avoid accuracy loss when the programme calculates
the large number

    cout      endl
    cout      "Product of dot: "      dot           endl
    //Using the dot function we sum x and x together to find the sum of nc^2 using the dot function
    cout      "Product of cdot: "      cdot               endl
    //Using the dot function we sum x_new to find the sum of nc^2 using the KahanSum method
    cout      "Difference in cdot and the actual answer: "      cdot                         endl
    //This answers q2c

    normal_cal         2
    //This uses the function object and defines the int m = 2
    long double
    //This uses the function object and defines the valarray object valarray a
    cout      endl      "The result of norm l2('valarray'a) is: "                      endl
    //This answers q2d

    return 0
```

Output:

A) I have defined the dot function before main() and the code explaining it is simply understood as it uses valarray multiplication with .sum() to return the final value.

B) I first create the valarray A within main() with size n as specified in the question. Then using the dot function I had outputted the final result and the exact value difference which is shown in the output. We get for dot: 1.6449330668477264373, with the difference being:

Since the cdot function already exists in the KahanSum function in week 10 notes (Compensated Summatation.cpp) I have just adjusted the valarray x to have the value c squared already instead of having to create two input values for cdot which reduces accuracy loss in the output. We also can see the difference in values from dot and cdot and the exact value. The result of cdot is :

D) Using the class normal_cal we can implement the double operator()… into the function in order to create a function object. The result of finding the $I_m(A)$ is:

https://www.symbolab.com/solver/step-by-step/%5Csqrt%7B%5Cfrac%7B%5Cpi%5E%7B2%7D%7D%7B6%7D%7D?or=input
$I_m(A)^2$ = dot(A,A) (Source: https://www.symbolab.com/solver/step-by-step/1.2825494403132093879%5E%7B2%7D?or=input) if we compute it within the .cpp file which is implied from the previous statement of square rooting the exact value.

Question 3

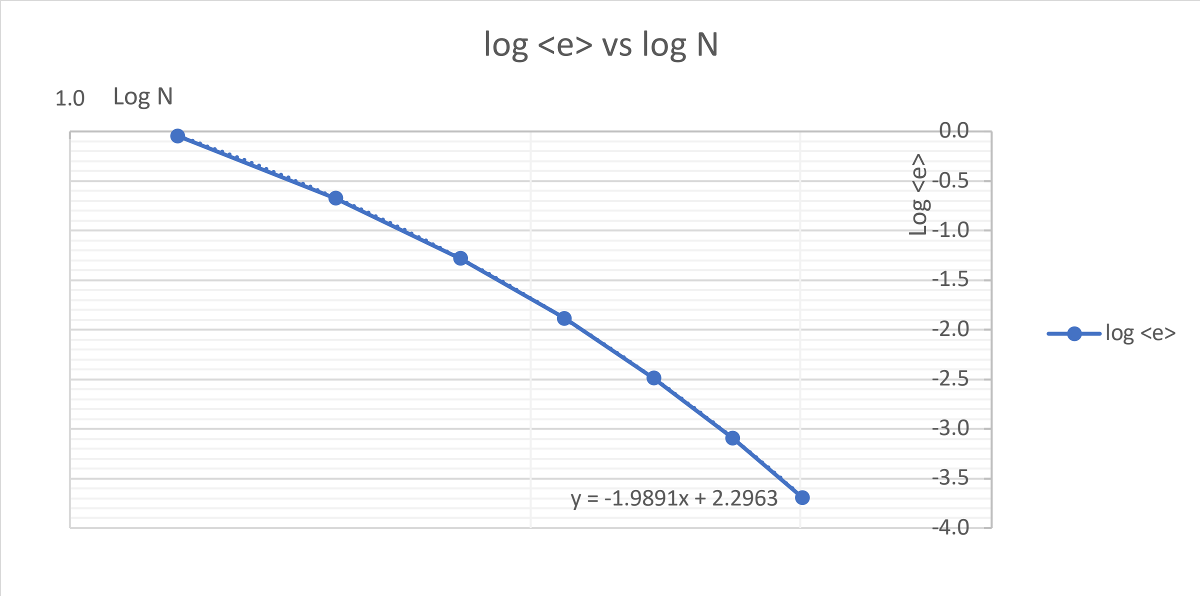
Code:

```
#include <iostream>
#include <cmath>
#include <valarray>
#include <iomanip>
using namespace std

long double normal_calculation int      valarray  long double
   //We have turned the normal_cal function object into a function so it's easier to use to avoid longer
calculation times and less re-defining of valarrays when the function is called several times
   long double            0
   for  int      0           size
                 pow abs

   long double                pow          1 static_cast  long double
   //This takes the normal formula from 2d in the form of a function instead of function object
   return


void fanalytical int
   valarray  long double                   1
   valarray  long double          1
   valarray  long double              1
   //This initialises the valarray to the correct size for given N

   long double                  pow 2.0   static_cast   long double       2.0
   //We calculate the long double here which is required to calculate other doubles

   for  int      0           1
                       2.0   double                     //xi formula
             exp   16.0     pow                2.0      //f(xi) formula
                 32.0      32.0            pow                  2.0
    //This series of for loops inputs the correct values into each respective valarray

   valarray   long double             1
   valarray   long double           1
   for  int     0        1
      if        0
               0             2    2.0          1            static_cast   long double
             0             0         0
        else if
                             2.0           1            2    static_cast   long double

        else
                           1    2.0                  1     static_cast   long double



   // Write header,in week 9 lab code
   #define SP << setw(30) << setprecision(10) <<     // save some repetition when writing
   cout      " i:" SP "xi:" SP "f(xi):" SP "fddx(xi):" SP "ei:"      endl
   for  int     0        1
      cout       SP                  SP       SP         SP            endl
    //This generates the headers when tabulating the outputs


long double e int
```

```
    valarray  long double                    1
    valarray  long double            1
    valarray  long double                1

    long double                 pow 2.0   static_cast   long double        2.0
    //We calculate the long double here which is required to calculate other doubles

    for  int      0           1
                              2.0    double                      //xi formula
                 exp   16.0     pow                    2.0      //f(xi) formula
                      32.0      32.0           pow                  2.0
     //This series of for loops inputs the correct values into each respective valarray

    cout      setprecision 10
    valarray   long double               1
    valarray   long double             1
    for  int     0         1
        if         0
                 0               2    2.0            1
               0             0          0
           else if
                                 2.0            1              2

          else
                              1    2.0                   1


      //The above is already explained but we need the previous values to calculate the error
    cout      setprecision 15
    long double              static_cast   long double             static_cast  long
double          1    normal_calculation 1
    //This long double calculates the final value error from the value of n inputted
    cout     "The result of (N^2)<e>: "                   " N = "              endl
    return



int main

    fanalytical 63

    int     15
    while     2048
        e
               2     1

    return 0
```

i:	xi:	f(xi):	fddx(xi):	ei:
0	-1.0000000000E+00	1.1253517470E-07	3.0325028050E-04	-1.9161538720E-04
1	-9.6825396830E-01	3.0582525730E-07	3.0325028050E-04	-1.9439510260E-05
2	-9.3650793650E-01	8.0473416630E-07	7.4090850990E-04	-4.3931269510E-05
3	-9.0476190480E-01	2.0503384740E-06	1.7485617630E-03	-9.5496127630E-05
4	-8.7301587300E-01	5.0581617430E-06	3.9853066720E-03	-1.9953322570E-04
5	-8.4126984130E-01	1.2082419050E-05	8.7701453610E-03	-4.0040497600E-04
6	-8.0952380950E-01	2.7945321210E-05	1.8629553820E-02	-7.7090936670E-04
7	-7.7777777780E-01	6.2583283910E-05	3.8187293640E-02	-1.4223185560E-03
8	-7.4603174600E-01	1.3570680330E-04	7.5510389870E-02	-2.5108658700E-03
9	-7.1428571430E-01	2.8493048890E-04	1.4397690970E-01	-4.2330422030E-03
10	-6.8253968250E-01	5.7925562040E-04	2.6459093940E-01	-6.7983992840E-03
11	-6.5079365080E-01	1.1402382870E-03	4.6839726220E-01	-1.0366788600E-02
12	-6.1904761900E-01	2.1732766480E-03	7.9822669370E-01	-1.4940336510E-02
13	-5.8730158730E-01	4.0107762770E-03	1.3084792130E+00	-2.0212618550E-02
14	-5.5555555560E-01	7.1669750380E-03	2.0611753540E+00	-2.5400518920E-02
15	-5.2380952380E-01	1.2400448030E-02	3.1163492080E+00	-2.9115670420E-02
16	-4.9206349210E-01	2.0774610570E-02	4.5153801800E+00	-2.9362383040E-02
17	-4.6031746030E-01	3.3699420810E-02	6.2573972800E+00	-2.3760098410E-02
18	-4.2857142860E-01	5.2930501930E-02	8.2715389850E+00	-1.0059826070E-02
19	-3.9682539680E-01	8.0497727160E-02	1.0391247510E+01	1.3058370220E-02
20	-3.6507936510E-01	1.1853736110E-01	1.2339938960E+01	4.5049998080E-02
21	-3.3333333330E-01	1.6901331540E-01	1.3738815290E+01	8.2718057280E-02
22	-3.0158730160E-01	2.3333539250E-01	1.4145669560E+01	1.1994722350E-01
23	-2.6984126980E-01	3.1191362430E-01	1.3127321060E+01	1.4830119450E-01
24	-2.3809523810E-01	4.0372170860E-01	1.0358281690E+01	1.5862304310E-01
25	-2.0634920630E-01	5.0596897830E-01	5.7267759290E+00	1.4343306840E-01
26	-1.7460317460E-01	6.1398775300E-01	-5.7970953320E-01	9.9533673940E-02
27	-1.4285714290E-01	7.2142229040E-01	-8.0392287710E+00	2.9969057300E-02
28	-1.1111111110E-01	8.2075480830E-01	-1.5832696630E+01	-5.5495213470E-02
29	-7.9365079370E-02	9.0413096780E-01	-2.2958712730E+01	-1.4184488810E-01
30	-4.7619047620E-02	9.6436909490E-01	-2.8408464310E+01	-2.1208606360E-01
31	-1.5873015870E-02	9.9597687240E-01	-3.1362817260E+01	-2.5148112580E-01
32	1.5873015870E-02	9.9597687240E-01	-3.1362817260E+01	-2.5148112580E-01
33	4.7619047620E-02	9.6436909490E-01	-2.8408464310E+01	-2.1208606360E-01
34	7.9365079370E-02	9.0413096780E-01	-2.2958712730E+01	-1.4184488810E-01
35	1.1111111110E-01	8.2075480830E-01	-1.5832696630E+01	-5.5495213470E-02
36	1.4285714290E-01	7.2142229040E-01	-8.0392287710E+00	2.9969057300E-02
37	1.7460317460E-01	6.1398775300E-01	-5.7970953320E-01	9.9533673940E-02
38	2.0634920630E-01	5.0596897830E-01	5.7267759290E+00	1.4343306840E-01

39	2.3809523810E-01	4.0372170860E-01	1.0358281690E+01	1.5862304310E-01
40	2.6984126980E-01	3.1191362430E-01	1.3127321060E+01	1.4830119450E-01
41	3.0158730160E-01	2.3333539250E-01	1.4145669560E+01	1.1994722350E-01
42	3.3333333330E-01	1.6901331540E-01	1.3738815290E+01	8.2718057280E-02
43	3.6507936510E-01	1.1853736110E-01	1.2339938960E+01	4.5049998080E-02
44	3.9682539680E-01	8.0497727160E-02	1.0391247510E+01	1.3058370220E-02
45	4.2857142860E-01	5.2930501930E-02	8.2715389850E+00	-1.0059826070E-02
46	4.6031746030E-01	3.3699420810E-02	6.2573972800E+00	-2.3760098410E-02
47	4.9206349210E-01	2.0774610570E-02	4.5153801800E+00	-2.9362383040E-02
48	5.2380952380E-01	1.2400448030E-02	3.1163492080E+00	-2.9115670420E-02
49	5.5555555560E-01	7.1669750380E-03	2.0611753540E+00	-2.5400518920E-02
50	5.8730158730E-01	4.0107762770E-03	1.3084792130E+00	-2.0212618550E-02
51	6.1904761900E-01	2.1732766480E-03	7.9822669370E-01	-1.4940336510E-02
52	6.5079365080E-01	1.1402382870E-03	4.6839726220E-01	-1.0366788600E-02
53	6.8253968250E-01	5.7925562040E-04	2.6459093940E-01	-6.7983992840E-03
54	7.1428571430E-01	2.8493048890E-04	1.4397690970E-01	-4.2330422030E-03
55	7.4603174600E-01	1.3570680330E-04	7.5510389870E-02	-2.5108658700E-03
56	7.7777777780E-01	6.2583283910E-05	3.8187293640E-02	-1.4223185560E-03
57	8.0952380950E-01	2.7945321210E-05	1.8629553820E-02	-7.7090936670E-04
58	8.4126984130E-01	1.2082419050E-05	8.7701453610E-03	-4.0040497600E-04
59	8.7301587300E-01	5.0581617430E-06	3.9853066720E-03	-1.9953322570E-04
60	9.0476190480E-01	2.0503384740E-06	1.7485617630E-03	-9.5496127630E-05
61	9.3650793650E-01	8.0473416630E-07	7.4090850990E-04	-4.3931269510E-05
62	9.6825396830E-01	3.0582525730E-07	3.0325028050E-04	-1.9439510260E-05
63	1.0000000000E+00	1.1253517470E-07	3.0325028050E-04	-1.9161538720E-04


log <e> vs log N
1.0
Log N
0.0
-0.5
-1.0
-1.5
-2.0
-2.5
-3.0
-3.5
-4.0
Log <e>
y = -1.9891x + 2.2963
log <e>

B)

Code:

```
#include <iostream>
#include <valarray>
#include <cmath>
#include <iomanip>
#include <random>
using namespace std

long double dot const valarray  long double      const valarray  long double
    return          sum
  //This function is recalled from question 2a

long double monteCarloEstimate long double                long double               long double
//Function to execute Monte Carlo integration on predefined function
    long double                   0.0
    const int       31
    //seed value
    mt19937_64
    //We are generating a seed with a real random value
    uniform_real_distribution  long double
    //This generates random values unifromally between the lower and upper bound

    for  int    0
        //Select a random number within the limits of integration
        long double
        //Add the f(x) value to the running sum using the correct function
                        pow    4.0                                0.5


    long double                                                   static_cast  long double
    //This calculates the final value of the Integration estimate using this method
    return


int main

    long double       63.0
    //This is the value of n defined in the question
    long double                static_cast  long double   4
    //We calculate delta x using the formula given, we use static_cast as this a fraction to avoid
accuracy loss

    valarray  long double                    1
    for  int      0                  size
                        double    4.0
      //This calculates the gridpoints for 4a/4b and inputs them into a valarray

    valarray  long double          1
    for  int     0          size
                pow    4.0                      0.5
      //This calculates the f(i) points for 4a/4b and inputs them into a valarray

    valarray   long double                      1
    for  int      0                   size
        long double           static_cast  long double             2.0
        if        0

           //This inputs the values into the valarray for when i is 0
```

```
else if
    //This inputs the values into the valarray for when i is 63
else
                    2.0
    //This inputs the values into the valarray
```

//This for loop inputs in the correct values for the weights creates the values inputted into the valarray

```
valarray  long double                  1
for  int     0                  size
    long double          static_cast  long double            48.0
    if       0
                    17.0
    //This inputs the values into the valarray for when i is 0
  else if       1
                    59.0
    //This inputs the values into the valarray for when i is 1
  else if       2
                    43.0
    //This inputs the values into the valarray for when i is 2
  else if       3
                    49.0
    //This inputs the values into the valarray for when i is 3
  else if           3
                    49.0
    //This inputs the values into the valarray for when i is 60
  else if           2
                    43.0
    //This inputs the values into the valarray for when i is 61
  else if           1
                    59.0
    //This inputs the values into the valarray for when i is 62
  else if
                    17.0
    //This inputs the values into the valarray for when i is 63
  else
                    48.0
    //This inputs the values into the valarray
```

//This for loop inputs in the correct values for the weights creates the values inputted into the valarray

```
long double      3.1415926535897932385

valarray   long double              1
for  int     0             size
               static_cast  long double
  //This calculates the theta values for 4c and inputs them into a valarray
```

```
valarray  long double                         1
for  int     0                        size
                                4.0   static_cast   double    4   cos                  2.0
  //This calculates the gridpoints for 4c and inputs them into a valarray

valarray  long double                1
for  int    0                size
               pow    4.0                                                   0.5
  //This calculates the f(i) for 4c and inputs them into a valarray


valarray  long double                1
for  int     0                size
    if      0
                     static_cast  double   2.0  pow    2.0
      //This inputs the values into the valarray for when i is 0

    else if
                     static_cast  double   2.0  pow    2.0
      //This inputs the values into the valarray for when i is 63

    else
      long double                   0
      for  double       1.0       32
                                  static_cast  double   2.0   cos 2.0                   4.0  pow    2.0   1.0
          //This generates the summation part of the CC equation

                     2.0      static_cast  double   2.0    1.0
      //This inputs the values for the valarray for the correct values of i

  //This for loop inputs in the correct values for the weights creates the values inputted into the
valarray

long double             2.0
long double                         dot
long double                       dot
long double               dot
//We use the function from 2a to calculate the porduct of two valarrays
long double             monteCarloEstimate 0.0  4.0  10000.0
//This calls the function defined to calculate the MC estimate for 4d

//The below outputs all the results for question 4
cout      setprecision 10
cout      "i_real: "                 endl      endl
cout      "i_trapezium: "                      endl
cout      "i_trapezium - i_real = "                                endl      endl
cout      "i_simpson: "                      endl
cout      "i_simpson - i_real = "                                endl      endl
cout      "i_CC: "                   endl
cout      "i_trapezium - i_real = "                          endl      endl
cout      "i_MC: "                   endl
cout      "i_MC - i_real = "                            endl

return 0
```

A) Using the dot function from 2a, we get the answer for 4a in the first two rows of the output past i_real which gives our I_{exact}. The error is what I expected as this method is not the most accurate method. This is shown by $I_{trapezium}$ = $I_{trapezium}$ I_{exact} =

Using the dot function from 2a, we get the answer for 4b in the second two rows of the output past i_real. The error is what I expected as this method is not the most accurate method but it is more accurate compared to the trapezium method. This is shown by $I_{Simpson}$ = 6.
$I_{Simpson}$ I_{exact} =

C) Using the dot function from 2a, we get the answer for 4c in the third two rows of the output past i_real. The error is what I expected as this method is the most accurate method. This is shown by $I_{ClenshawCurtis}$ = $I_{ClenshawCurtis}$ I_{exact} =
Extended Simpson method.

Using the dot function from 2a, we get the answer for 4d in the last two rows of the output past i_real. The error is what I expected as this method is a more accurate method compared to the Extended Simpson method. However, the accuracy is highly dependent on the seed value implemented into the random number generator in the function 'monteCarloEstimate'. $I_{MonteCarlo}$ = $I_{MonteCarlo}$ I_{exact} =

Question 5a, 5b

Code:

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <valarray>
using namespace std

valarray  long double   F const long double      const valarray  long double
   /* du/dt = f(t,u) = {  u0
                       3(u1)^2 + (u1)
    since a =0,c=1(given) we get the above
   */
   valarray  long double        size
   // u[0] = dq/dt, u[1] = dp/dt
   //Insert the relevant equations into the f valarray
     0       1
     1     3.0 pow static_cast   long double      0   2.0        0
   return
  //This function will return the correct values of dp/dt and dq/dt by using a valarray<valarray<... in
main()

valarray  long double   RK2 const long double     const valarray  long double         const long double
     valarray  long double     const long double    const valarray  long double
   unsigned long long           size
   //This allows for us to get the value of the size of the valarray we are using to stay consistent
   valarray  long double
   //This initialises valarrays of the correct size

   //Since we call on function f and don't declare any specific valarray we are able to manipulate the
function to our use different valarrays
                         0.5            0.5
   return
  //The above formula is used the lecture notes - Page 5 of ODEs

int main

   long double               0.
   long double              10.0
   int       100000
   long double                                      static_cast  long double
   valarray  long double          2     //[0],T[1],...,T[n],T[n+1]
   //Given from question

   valarray  valarray  long double           2
   valarray  long double          2
   valarray  long double          2

   //set initial data such that U(0) = {q(0),p(0)} = {-0.5,0}
     0       0.5  0.

   for int    0
                           static_cast  long double
      //given in question
           1     RK2                          F
      //We use the +1 as we already have been given the initial conditions, we do not want to overwrite
it - we also find the 0th value of each valarray without the need for 2 'for' loops
              static_cast  long double   0.5      j  1      j  1     static_cast  long
double      j  0      j  0      j  0     static_cast  long double   0.5      j  0      j  0
```

```
        //E(t) = 0.5(p^2) + (V(q)) = 0.5(p*p) - (q*q*q) -(0.5*q*q)
                j  0    static_cast  long double    0.5  cosh 0.5         cosh 0.5
        //error = q(t) - (-0.5sech^2(0.5*t) = q(t) - (-0.5/cosh^2(0.5*t))
        //We have to use cosh identity instead of sech as only cosh is cmath library


    // Write header
#define SP << setw(26) << setprecision(10) <<    // save some repetition when writing
    cout      "t" SP "q(t)" SP "p(t)" SP "E(t)" SP "e(t)"      endl

    int      0
    while
        cout           SP    i  0  SP    i  1  SP       SP           endl
              10000
        //We use a while function and this equation to output the only relevant time intervals for
i=0,i=1...i=10

    return 0
```

Output:

t	q(t)	p(t)	E(t)	e(t)
0	-5.0000000000E-01	0.0000000000E+00	0.0000000000E+00	0.0000000000E+00
1	-3.9322386600E-01	1.8171549550E-01	-5.2467299170E-12	5.2732825060E-10
2	-2.0998717020E-01	1.5992500170E-01	-2.0739801250E-11	6.0979269530E-10
3	-9.0353319320E-02	8.1783148900E-02	-2.5493658060E-11	1.3709330730E-10
4	-3.5325412760E-02	3.4054671400E-02	-2.5990213230E-11	-3.3179652270E-10
5	-1.3296114370E-02	1.3118134550E-02	-2.6023080210E-11	-1.0259183440E-09
6	-4.9330212880E-03	4.9086209670E-03	-2.6024900570E-11	-2.7049412680E-09
7	-1.8204495770E-03	1.8171181970E-03	-2.6024994840E-11	-7.2171302700E-09
8	-6.7049481720E-04	6.7000626130E-04	-2.6024999610E-11	-1.9475678440E-08
9	-2.4681149980E-04	2.4664508290E-04	-2.6024999850E-11	-5.2800230780E-08
10	-9.0935003930E-05	9.0640063020E-05	-2.6024999860E-11	-1.4338845530E-07

Question 5c

Code:

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include <valarray>
using namespace std

valarray  long double   F const long double      const valarray  long double
   /* du/dt = f(t,u) = {  u0
                     3(u1)^2 + (u1)
    since a =0,c=1(given) we get the above
   */
   valarray  long double        size
   // u[0] = dq/dt, u[1] = dp/dt
   //Insert the relevant equations into the f valarray
     0       1
     1     3.0 pow static_cast   long double       0   2.0        0
   return
  //This function will return the correct values of dp/dt and dq/dt by using a valarray<valarray<... in
main()

//We use the pre-defined function Hermit-2 as it uses the trapezium rule without having to alter the
function F values
valarray  long double   H2 const long double        const valarray  long double           const long double
     valarray  long double      const long double     const valarray  long double
   long double                    //This increases the time by delta_t starting from t_initial
   unsigned int           10  //This is the maximum number of self-consistent iterations (10) that will be
used
   valarray  long double                //This sets initial guess for the implicit ODE
   for int      0
      //This 'for' loop carries out the self-consistent iterations
                  0.5                                    //trapezium rule is now applied to the new values

   return


int main

   long double              0.
   long double             10.0
   int      100000
   long double                                        static_cast  long double
   valarray  long double          2     //[0],T[1],...,T[n],T[n+1]
   valarray  valarray  long double            2
   valarray  long double         2
   valarray  long double          2
   //Given from question

   // Write header
#define SP << setw(26) << setprecision(10) <<    // save some repetition when writing
   cout     "t" SP "q(t)" SP "p(t)" SP "E(t)" SP "e(t)"      endl

   //set initial data such that U(0) = {q(0),p(0)} = {-0.5,0}
     0       0.5  0.

   for int    0
                          static_cast  long double
      //given in question
```

```
        1    H2                    F
    //We use the +1 as we already have been given the initial conditions, we do not want to overwrite
it - we also find the 0th value of each valarray without the need for 2 'for' loops
                static_cast  long double   0.5      j  1      j  1      static_cast  long
double       j  0      j  0      j  0     static_cast  long double   0.5      j  0      j  0
    //E(t) = 0.5(p^2) + (V(q)) = 0.5(p*p) - (q*q*q) -(0.5*q*q)
                j  0    static_cast  long double    0.5  cosh 0.5        cosh 0.5
    //error = q(t) - (-0.5sech^2(0.5*t) = q(t) - (-0.5/cosh^2(0.5*t))
    //We have to use cosh identity instead of sech as only cosh is cmath library


  int     0
  while
    cout          SP    i  0  SP    i  1  SP      SP          endl
         10000
    //We use a while function and this equation to output the only relevant time intervals for
i=0,i=1...i=10

  return 0
```

Output:

t	q(t)	p(t)	E(t)	e(t)
0	-5.0000000000E-01	0.0000000000E+00	0.0000000000E+00	0.0000000000E+00
1	-3.9322386670E-01	1.8171549530E-01	1.0518493570E-11	-2.4935283190E-10
2	-2.0998717100E-01	1.5992500250E-01	4.1511998940E-11	-1.9208806810E-10
3	-9.0353319240E-02	8.1783149770E-02	5.1020583580E-11	2.2312316660E-10
4	-3.5325411610E-02	3.4054672630E-02	5.2013756370E-11	8.1606356960E-10
5	-1.3296111260E-02	1.3118137480E-02	5.2079493920E-11	2.0833425880E-09
6	-4.9330131700E-03	4.9086288420E-03	5.2083134830E-11	5.4129834200E-09
7	-1.8204279210E-03	1.8171396040E-03	5.2083323390E-11	1.4438908260E-08
8	-6.7043636860E-04	6.7006446620E-04	5.2083332920E-11	3.8972944540E-08
9	-2.4665303330E-04	2.4680331010E-04	5.2083333390E-11	1.0566625470E-07
10	-9.0504655690E-05	9.1070174610E-05	5.2083333420E-11	2.8695978550E-07