

```

// file: integ_test.cpp
//
// This is a test program for basic integration methods.
//
// Programmer: Dick Furnstahl  furnstahl.1@osu.edu
//
// Revision history:
// 04-Jan-2004 original version, for 780.20 Computational Physics
// 08-Jan-2005 changed functions to pass integrand
// 09-Jan-2011 updated functions
//
// Notes:
// * define with floats to emphasize round-off error
// * compile with: "g++ -Wall -c integ_test.cpp"
// * adapted from: "Projects in Computational Physics" by Landau and Paez
//   copyrighted by John Wiley and Sons, New York
//   code copyrighted by RH Landau
//
//*****
// include files
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cmath>
using namespace std;

#include "integ_routines.h" // prototypes for integration routines

float my_integrand (float x);

const double ME = 2.7182818284590452354E0; // Euler's number
//*****

int
main ()
{
    // set up the integration specification
    const int max_intervals = 501; // maximum number of intervals
    const float lower = 0.0; // lower limit of integration
    const float upper = 1.0; // upper limit of integration

    const double answer = 1. - 1. / ME; // the "exact" answer for the test
    float result = 0.; // approximate answer

    // open the output file stream
    ofstream integ_out ("integ.dat"); // save data in integ.dat
    integ_out << "# N trapezoid Simpsons Gauss " << endl;
    integ_out << "-----" << endl;

    // Simpson's rule requires an odd number of intervals
    for (int i = 3; i <= max_intervals; i += 2)
    {
        integ_out << setw(4) << i;

        result = trapezoid_rule (i, lower, upper, &my_integrand);
        integ_out << " " << scientific << fabs (result - answer);

        result = simpsons_rule (i, lower, upper, &my_integrand);
        integ_out << " " << scientific << fabs (result - answer);

        result = gauss_quadrature (i, lower, upper, &my_integrand);
        integ_out << " " << scientific << fabs (result - answer);

        integ_out << endl;
    }

    cout << "data stored in integ.dat\n";
}

```

```

integ_out.close ();

return (0);
}
//*****
// the function we want to integrate
float
my_integrand (float x)
{
    return (exp (-x));
}
//*****

```

```

// file: integ_routines.cpp
//
// These are routines for trapezoid, Simpson and Gauss rules
//
// Programmer: Dick Furnstahl  furnstahl.1@osu.edu
//
// Revision history:
// 04-Jan-2004  original version, for 780.20 Computational Physics
// 08-Jan-2005  function to be integrated now passed, changed names
// 09-Jan-2011  new names and rearranged; fixed old bug
//
// Notes:
// * define with floats to emphasize round-off error
// * compile with: "g++ -Wall -c integ_routines.cpp" or makefile
// * adapted from: "Projects in Computational Physics" by Landau and Paez
//   copyrighted by John Wiley and Sons, New York
//   code copyrighted by RH Landau
// * equation for interval h = (b-a)/(N-1) with x_min=a and x_max=b
//
// *****
//
// include files
#include <cmath>
#include "integ_routines.h" // integration routine prototypes
// *****
//
// Integration using trapezoid rule
float trapezoid_rule ( int num_pts, float x_min, float x_max,
                     float (*integrand) (float x) )
{
    float interval = ((x_max - x_min)/float(num_pts - 1)); // called h in notes
    float sum= 0.; // initialize integration sum to zero
    for ( int n=2; n<num_pts; n++) // sum the midpoint contributions
    {
        float x = x_min + interval * float(n-1);
        sum += interval * integrand(x);
    }
    // add in the endpoint contributions
    sum += (interval/2.) * (integrand(x_min) + integrand(x_max));
    return (sum);
}
// *****
//
// Integration using Simpson's rule
float simpsons_rule ( int num_pts, float x_min, float x_max,
                     float (*integrand) (float x) )
{
    float interval = ((x_max - x_min)/float(num_pts - 1)); // called h in notes
    float sum= 0.; // initialize integration sum to zero
    for ( int n=2; n<num_pts; n+=2) // loop for odd points
    {
        float x = x_min + interval * float(n-1);
        sum += (4./3.)*interval * integrand(x);
    }
    for ( int n=3; n<num_pts; n+=2) // loop for even points
    {
        float x = x_min + interval * float(n-1);
        sum += (2./3.)*interval * integrand(x);
    }
    // add in the endpoint contributions
    sum += (interval/3.) * (integrand(x_min) + integrand(x_max));
    return (sum);
}

```

```

// *****
//
// Integration using Gauss quadrature rule
float gauss_quadrature ( int num_pts, float x_min, float x_max,
                        float (*integrand) (float x) )
{
    float quadra = 0.;
    double weight[1000], x[1000]; // for points and weights
    gauss (num_pts, 0, x_min, x_max, x, weight); // returns Legendre
    for ( int n=0; n< num_pts; n++) // points and weights
    {
        quadra += integrand(x[n])*weight[n]; // calculating the integral
    }
    return (quadra);
}
// *****

```