PHYS4070 Worksheet. Week 3: Integration

Consider function:

$$f(x) = e^{-x}\ \sin(15x + 1/2)$$

- Over domain x = [0, 3]
- Exact value of the integral is: 0.059972633417316158....

1. Calculate the integral of the above function over domain [0,3] using

- Trapezoid rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{[f(x_0) + f(x_{N-1})]\delta x}{2} + \sum_{i=1}^{N-2} f(x_i)\,\delta x.$$

- Simpsons rule

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{\delta x}{3}\left[f(x_0) + f(x_{N-1}) * 2\sum_{\text{even } j} f(x_j) * 4\sum_{\text{odd } j} f(x_j)\right]$$

- k=5 Mixed-quadrature rule:

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{k-1} w_i\,(f(x_i) + f(x_{N-i-1}))\delta x * \sum_{i=k}^{N-k-1} f(x_i)\delta x$$

and compare the resulting error (difference between calculated and exact value). (p.s.: if you want to cheat, example code for these three methods was presented in the lecture notes!)

Do for:

- N = 11, 51, 101
- Note: N must be odd for Simpsons rule; and we must have N>2*k for mixed quadrature rule

The weights for a k=5 mixed quadrature rule are:

- w = {475.0/1440, 1902.0/1440, 1104.0/1440, 1586.0/1440, 1413.0/1440}

2. How many points do we need to use in the trapezoid rule for the error of the trapezoid rule to become smaller than that of the k=5 mixed-quadrature rule with n=100?

- Code a loop that calculates the integral using trapezoid rule with varying number of points, starting from n=100
- Continue until the error drops below that of the mixed-quadrature rule (with n=100)

3. [optional] Code an adaptive integration algorithm

- Adaptive methods need two things:
  - A method to integrate function over a given domain
  - A method to estimate the error of the integral
- They continuously sub-divide the integral into smaller-and-smaller sub-domains until the error for each drops below a given target
- Estimate the error by performing integral twice using a different number of points. For example, once using Simpsons rule with 3 points (n=3), and once with 7. (we don't need large n, since this will be taken into account by the recursive nature of algorithm)
- Estimate the error in the integral as the difference between these two values.
- If the error is too large, divide the integration region into two - and perform the same algorithm for each half
  - By continuing in this manner, we continuously divide sub-domains up until the error for the integral of each sub-domain drops below a specified value
- A rough pseudo-code algorithm is provided below; use it to guide your code
- This will be much easier if we use some concepts from Part A (functionals and recursive functions)
- Recursive functions can easily get out-of-hand - you may want to code in a counter that keeps track of the depth, and kills the function if the depth becomes too deep
- Test it by integrating f(x) as above
- If you don't want to code the algorithm, use my simple one (end of document)

```
Adaptive(Function, a, b, error_target):

 A1 = Integrate function (a,b) using Simpsons rule, n=3
 A2 = Integrate function (a,b) using Simpsons rule, n=7
 error = |A1-A2| (absolute value)

 if error is less than error_target:
   answer = A2
   return answer
   FINISHED

 otherwise:
   // call adaptive on sub-domains (a,m) and (m,b)
   // divide target_error by 2, since domain is half the size
   // (want the total error to be less than error_target)
   m = (a+b)/2
   answer = Adaptive(Function, a, m, error_target/2)
          + Adaptive(Function, m, b, error_target/2)
   return answer
   FINISHED
```

4. Use your (or my) adaptive method for 'tricky' function

Consider 'tricky' function

$$g(x) = \frac{f(x)}{x + 10^{-6}} = \frac{e^{-x}\sin(15x + 1/2)}{x + 10^{-6}}$$

Integrate this tricky below function over domain [0,1]:

- This function is very hard to integrate, due to near-singular
- Accurate value should be ~6.39019353...
- Integrate it using N=1001 using Simpsons and mixed-quadrature rules
  - You will notice the result is not even close to correct
- Now, integrate it using your adaptive method

5. Vacuum polarisation - Uehling potential

In quantum electrodynamics, the regular Coulomb interaction between two charges is perturbed by an effect called *vacuum polarisation*. It is caused by the creation of short-lived virtual electron-positron pairs out of the vacuum. This effect is largest in strong electric fields, and at very high energies. It must be taken into account for accurate calculations, including in atomic physics.

The Uehling potential, which describes the vaccuum polarisation correction to the regular Coulomb atomic potential, is (in atomic units):

$$V_{vp}(r) = \int_1^\infty \mathrm{d}t\, \frac{-2\,Z\alpha^2}{3\pi\, r} \frac{\sqrt{t^2+1}}{t^2}\left(1 + \frac{1}{2t^2}\right) e^{-2tr/\alpha}$$
$$\approx \int_1^{1/r} \mathrm{d}t\, \frac{-2\,Z\alpha^2}{3\pi\, r} \frac{\sqrt{t^2+1}}{t^2}\left(1 + \frac{1}{2t^2}\right) e^{-2tr/\alpha}$$
$$\text{where } \alpha \approx 1/137.036$$

The approximation in the second line is rough, but comes from fact that integrand becomes very small for t*r>1. This potential is largest at small r, where the electric field of the nucleus is extremely strong.

- The nuclear radius is on the order to ~1 fm (10^{-15} m) - which is around 10^-5 aB (10^-5 atomic units)
- Evaluate the Uehling potential at r = 1.0e-5 by performing the integral over dummy-variable t, using Simpsons rule with N=1001 and the adaptive method
- Accurate value should be ~-5.859605..
- To include the Uehling potential into calculations, we would need to evaluate it accurately at many points along r - the usefulness of adaptive methods in real-life examples should be clear

Example:
Simple function to integrate function f from [a,b], using adaptive method
Interactive version: https://godbolt.org/z/f865zExeY

```cpp
#include <cassert>
#include <cmath>
#include <functional>
#include <iostream>

// 'using' (alias) for functional std::function
using Function = std::function<double(double)>;
```

```cpp
// Function that integrates function f, over [a,b], using Simpsons rule.
// Note: n_pts must be odd (even sub-intervals)
double Simpsons(Function f, double a, double b, int n_pts) {
  assert(n_pts % 2 != 0 && "n_pts must be odd for Simpsons rule");
  const double dx = (b - a) / (n_pts - 1);
  double integral = (f(a) + f(b)) * dx / 3.0;
  for (int i = 1; i < n_pts - 1; ++i) {
    // ternary operator: w = condition ? val_if_true : val_if_false;
    // i % 2 == 0 means (i, mod 2) - is 0 if i is even
    const double w = (i % 2 == 0) ? (2.0 / 3) : (4.0 / 3);  // weight
    const double x = a + i * dx;
    integral += f(x) * dx * w;
  }
  return integral;
}
```

```cpp
// Integrates function f from [a,b], using adaptive method, until error drops
// below err_target. Recursive function.
// Note: This is very inefficient; I have tried to make it simple and clear at
// the expense of performance
double adaptive(Function f, double a, double b, double err_target, int depth = 1) {
  // Calculate integral twice, once with 3, once with 7 points
  // 3 is minimum for Simpsons rule
  const double integral_3 = Simpsons(f, a, b, 3);
  const double integral_7 = Simpsons(f, a, b, 7);
  // Error is difference between these
  const double err = std::abs(integral_3 - integral_7);
  if (err < err_target || depth > 100) {
    // if error is small, or depth exceeds limit, return best guess
    return integral_7;
  } else {
    const double m = (a + b) / 2.0;  // mid-point
    // divide target error by 2, since each domain is half the size
    // Increase depth counter as we recursively call function:
    return adaptive(f, a, m, err_target / 2.0, depth + 1) +
           adaptive(f, m, b, err_target / 2.0, depth + 1);
  }
}
```