# Computing at Scale
# Assignment 1

| | |
|---|---|
| Implementation | Monday, February 10th at 10:00pm |
| Code Review | Thursday, February 13th at 10:00pm |
| Reimplementation (based on code review) | Tuesday, February 18th at 10:00pm |
| Submit on Github | |

1. Numerical integration is a key component of many scientific computing applications in this assignment you will implement a routine to compute numerical integrals with user defined functions.

   All numerical integrals can be approximated by the following formula:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{1}$$

   where $w_i$ are the weights and $x_i$ are sampling locations.

   You must do the following in your implementation:

   (a) Create a pure virtual base class called `ScalarFunction` that the user can inherit from to define the function they want to integrate. This base class should have a call operator (`operator()`) that takes a single argument (the position) and returns a single value (the function value). Make sure to document any assumptions you make about the function (e.g., it is continuous on the interval $[a, b]$).

   (b) Allow the user to define a new integration rule by specifying the weights and sampling locations.

   (c) Use your mechanism for definining integration new integration rules to implement Gauss-Legendre quadrature with 1-4 points. And, Gauss-Lobotto with 3-5 points. You can find the weights and sampling locations here `https://en.wikipedia.org/wiki/Gaussian_quadrature#Gauss%E2%80%93Legendre_quadrature`.

   (d) Use your mechanism for definining integration new integration rules to implement the chebychev quadrature rule with an arbitrary number of points. You can find the weights and sampling locations here `https://en.wikipedia.org/wiki/Chebyshev%E2%80%93Gauss_quadrature`

   (e) You must use CMake to build your code. You should create a library for your integration routines and two executables: `unit_tests` and `polynomial_integrate` as described below.

   (f) The `unit_tests` executable should contain test cases that demonstrate the correctness of your implementation. If you are not using a testing framework, each test should be a separate function that returns 0 if the test passes and 1 if it fails. The `main` function in `unit_tests` should call each test and print a message indicating whether the test passed or failed and return the sum of the return values from the tests.

   (g) The `polynomial_integrate` should take an integrator type, a list of polynomial coefficients and the integration limits and return the integral of the polynomial. You may either read values from the command line or from a file.

   (h) Write a README file that explains how to use your code. Include build instructions.

   To submit your assignment you should create a branch assignment1 in your homework repository where you will commit your code. Once you are done, you should create a pull request to your main branch.

2. Please answer the following questions and add them to a markdown file called assignment0.md in your repository. Markdown is a simple markup language that is easy to read and write. You can find a guide to markdown here `https://daringfireball.net/projects/markdown/syntax`.

   (a) How did you test your code? What are the limitations of your testing?

(b) What approach did you take to document your functions and driver program?

(c) How did you handle errors? Whow did you test your error handling strategy?

(d) Is your code robust to errors in the input files? How did you test this?

(e) Are there any limitations in your implementation you are aware of? Do you have any ideas on how to address them?

(f) If you wanted to optimize the performance of your code, what approach would you take?

(g) How would you extend your code to two or more dimensions?

(h) Explain how you would handle a user defined function that takes additional parameters.

(i) Are there any ways you could handle arbitrary functions that are defined at runtime by the user for example in an input file?