Programming Techniques for Scientific Simulations

HS 2023
Dr. R. Käppeli,
V. Armegioiu,
I. Méndez Veiga, M. Herde,
F. Sasseli, N. Siegenheim

Exercise sheet 2

Due date: Monday, October 2, 2023, 23:59 CEST (midnight).

Problem 2.1 Static & dynamic arrays

Write a program which first reads in the number of values n. Then read in n values from standard input. Normalise the loaded sequence so that the sum is 1. Print out the normalised sequence in reverse order.

1. Set a maximum number of input values n_{max} and allocate a static array of length n_{max}.
2. Do the same using dynamic arrays so that the input size is not longer limited. One option to achieve this is to use `std::vector`.

Hint: if you use the standard input mechanism `std::cin`, the input sequence can be read from a file `input.txt` containing e.g. "3 1 2 3" with `./main < input.txt` or generated by either of the following lines:

```
./main < <(echo 3 1 2 3)
echo 3 1 2 3 | ./main
```

Problem 2.2 Simpson integration library using function pointers

1. Wrap the Simpson integration from the previous exercise into a function which takes as arguments a pointer to the integrand, the integration interval and the number of bins. Check the validity/correctness of input parameters using assertions. Put the function into a separate file.
2. Create a header file that declares the function. What are the preconditions and post-conditions? Document this file thoroughly.
3. Write a makefile that compiles the function for you. Make sure it only compiles the files that have changed.
4. Compile a library `libintegrate.a` that contains your Simpson integration function. Rewrite your makefile to link against it.

Problem 2.3 Formatted output

1. In your main function, iterate over an appropriately increasing discretisation resolution (number of bins) to investigate the improvement in accuracy.
2. Write your results into a file in the column format `<resolution> <integral>`, e.g.:

   ```
   3   2.001
   4   2.0003
   10  2.00001
   ...
   ```

   You can either use the standard file output mechanism `std::ofstream("result.txt")` or print it to `std::cout` and redirect the output into a file:

   ```
   ./main > result.txt
   ```

3. Update your makefile to generate the output file.

4. Optional: create a plot of the output file and integrate its generation into your makefile. You can use any plotting tool you prefer (gnuplot, a matplotlib script, Matlab, octave, ...). One simple method is this gnuplot one-liner:

   ```
   gnuplot < (echo "set terminal png; set output 'result.png'; plot 'result.txt'
    ↪  with lines")
   ```