

Problem 29

Introduction

This folder contain two C programs to calculate the moments

$$M_{2n} \equiv \frac{\int_{-\infty}^{+\infty} dx \frac{x^{2n}}{(2n)!} e^{-S(x)}}{\int_{-\infty}^{+\infty} dx e^{-S(x)}}, \quad \text{with } S(x) \equiv \frac{\alpha}{2}x^2 + \frac{\beta}{4}x^4, \quad (1)$$

in order to generate the data used in Figure 2.9. The program `direct.c` computes the moments directly from eq. (1) by numerical integration. The program `recursion.c` solves the set of equations

$$M_{n-2} = \alpha n M_n + \beta n(n+1)(n+2)M_{n+2} \quad (2)$$

(this is an exact recursion relation obeyed by the moments) with the boundary conditions $M_0 = 1$ and $M_N = 0$ where N is some large integer ($N = 5000$ in the code). More details about these two approaches are given below.

These programs can be compiled by doing

```
make direct
make recursion
```

The compilation requires the `FFTW3` library (the header files during the compilation phase, and the runtime libraries to run the code), and a compiler that supports quadruple precision floating point algebra, like `gcc`.

To run the programs, do

```
./direct
./recursion
```

(Note: the first one is very slow.) The output is made of three columns: the index $2n$ in the first column, the natural logarithm of the modulus $|M_{2n}|$ in the second column, and the phase of M_{2n} in the third one. These outputs are provided in the files `moments_direct.txt` and `moments_recursion-r**.txt`.

Direct computation of the integral

It is convenient to encapsulate all the moments into the following generating function:

$$M(z) \equiv \sum_{n=0}^{+\infty} M_n z^n = \frac{\int_{-\infty}^{+\infty} dx e^{zx} e^{-S(x)}}{\int_{-\infty}^{+\infty} dx e^{-S(x)}}. \quad (3)$$

From $M(z)$, one may in principle recover the moments from the derivatives at $z = 0$, but this becomes numerically unstable even for moderately large n . A more robust method is to calculate $M(z)$ on a circle of radius r in the complex plane, i.e., for $z = r e^{i\theta}$. Then, the moments are given by:

$$M_n = r^{-n} \int_0^{2\pi} \frac{d\theta}{2\pi} e^{-in\theta} M(re^{i\theta}). \quad (4)$$

The angular integral in eq. (4) is a Fourier integral, that can be approximated with very good accuracy by first evaluating $M(re^{i\theta})$ at equidistant values of the angle θ and then by performing a discrete Fourier transform (implemented with the FFTW3 library).

In order to compute the values of $M(z = r e^{i\theta})$ from the integral in eq. (3), we use the method of *double exponential quadrature*, based on the following change of variables:

$$x \equiv \frac{t}{1 - \exp(-k \sinh(t))}. \quad (5)$$

It may be shown¹ that this transformation, combined with a trapezoidal discretization of the resulting integral over t , leads to a sum that converges very quickly. With this method, implemented with *quadruple precision* floating point arithmetics², one may compute the moments up to very high orders.

An important parameter in the code is the variable `r0` that sets the radius of the circle used for computing $M(z)$. Loosely speaking, a given value of `r0` allows to calculate accurately moments whose orders are in a certain range (this range moves upwards as `r0` is increased). To cover all the orders n up to $n = 3700$ shown on Figure 2.9, the computation needs to be repeated with three distinct values of `r0`:

¹For more details on this approach and other related numerical integration methods, see L.N. Trefethen, J.A.C. Weideman, *The Exponentially Convergent Trapezoidal Rule*, SIAM Review vol. 56, no. 3, p 385.

²Quadruple precision floating point arithmetics can handle real numbers as small as 10^{-4932} before underflowing. In order to handle even smaller values, the code uses internally a rescaling to keep the moments in the representable range.

- r0=5: $n \leq 300$
- r0=15: $300 \leq n \leq 1500$
- r0=25: $1500 \leq n \leq 3700$

Computation from the recursion relation

The moments M_n obey the recursion relation (2). The set of all the sequences $\{M_n\}$ that satisfy this recursion is a linear space of dimension two. In addition, we have the two conditions $M_0 = 1$ and $M_\infty = 0$, that are sufficient to obtain a unique solution. In practice, we must replace $N = \infty$ by some large but finite N , chosen much larger than the maximal order of the moments one wishes to calculate.

Since the conditions that determine the solution are given at the two endpoints $n = 0$ and $n = N$, we can exploit the linear structure of the set of solutions of the recursion as follows. A basis of this two-dimensional linear space may be made of the following two sequences:

$$\begin{aligned} A_n, \quad & \text{defined by } A_0 = 1, A_2 = +1, \\ B_n, \quad & \text{defined by } B_0 = 1, B_2 = -1. \end{aligned} \tag{6}$$

Since the first two terms of the sequences $\{A_n\}$ and $\{B_n\}$ are known, it is straightforward to compute them for any n . The sequence $\{M_n\}$ we are looking for is the linear combination $M_n = aA_n + bB_n$, with $a + b = 1$ and $aA_N + bB_N = 0$ (this is a 2×2 system of linear equations that determine uniquely the coefficients a, b).

In order to tame the rapid decrease of the moments M_n (the goal being to avoid underflows), we use quadruple precision floating point arithmetics and we consider instead the rescaled moments $R_n \equiv E^n M_n$, where E is some constant. The R_n obey the following recursion relation:

$$E^2 R_{n-2} = \alpha n R_n + \beta n(n+1)(n+2)E^{-2} R_{n+2}. \tag{7}$$

The boundary conditions are unchanged ($R_0 = 1, R_N = 0$), and the solution $\{R_n\}$ is determined by the method described above. An appropriate value of the scaling constant E is determined by trial and error.