Elliptic Integrals

Elena Scibona

November 9, 2022

First exercise

The period T of the pendulum is obtained numerically from

$$T = 4\sqrt{\frac{L}{g}}\, F\left(\frac{\pi}{2}, k\right) \quad \text{where} \quad F\left(\phi, k\right) = \int_0^{\phi} \frac{1}{\sqrt{1 - k^2 \sin^2 u}}\, du\,,$$

with constants chosen so that $k = 0.8$ and $L/g = 1$ (from now on we will consider only these values and avoid specifying the dependence on k, e.g. $F(\phi, k) = F(\phi)$). In order to evaluate the integral, the Gaussian quadrature formula is applied in the range of integration $[0, \pi/2]$, with $N_g = 4$ gaussian points. The choice of the number N of sub-intervals is therefore determined by the required accuracy: $N = 6$ is obtained by successive evaluation of the integral with growing N until the difference between the result of the n^{th} step and the previous one is at most 10^{-8} s.
Here is the output of the first section:

```
Tolerance: 1e-08
Method: Gauss-Legendre quadrature (N = 6,  N_g = 4)
Period: T = 7.981211111s
```

Second exercise

The second part of the code tests how good the small-angle approximation formula[1]

$$T_{approx} = 4\sqrt{\frac{L}{g}}\frac{\pi}{2} = 2\pi$$

is compared to the T just calculated, which is closer to the real period by several significant digits and is therefore treated as the reference value.
The relative error

$$err = \left|\frac{T_{approx}}{T} - 1\right| \tag{1}$$

is higher than 20%.
The output of the code is displayed below.

```
Small-angle approximation: T_approx = 6.283185307s
Relative error: err = 0.212752899
```

Third exercise

Given $t = F(\phi)$ it is possible to get the dependence $\phi = \phi(t)$ by sampling the roots of the parametric function

$$Z(\phi, t_i) = F(\phi) - t_i \tag{2}$$

[1] T_{approx} is the leading term in the Taylor expansion of the expression for T

for a set of discrete values of the parameter t_i. Once t_i is set, as the integrand is always positive, the right term of the equation is increasing monotone. Moreover, subtracting t_i is a downward translation of the integral function. Knowing this, and noting that $Z(0,0) = 0$, we deduce that the roots of $Z(\phi, t_i)$ cannot be found on the negative ϕ-axis.
In order to extrapolate them, both the bisection and Newton methods are used 301 times on values t_i equally spaced between 0.0 s and 30.0 s. The latter is arbitrarily applied in the interval $[0, \pi/2]$ (first guess: $\pi/4$), where it performs well (tolerance $\sim 10^{-8}$ reached in no more than 10 steps). However, the first algorithm needs the function to be called over an interval in which $Z(\phi, t_i)$ changes sign. Due to what has already been pointed out, zero is chosen as the left boundary, while the right one is the first integer number $\widetilde{\phi}$ such that $Z(\widetilde{\phi}, 30) > 0$ (its value is $\widetilde{\phi} = 24$, previously calculated and employed in the code without showing the implementation, as not strictly necessary to solve the exercise). The requested tolerance is the same used in the numeric quadrature. The knowledge of ϕ_i allows us to apply Gauss method, with $N = 10$ and $N_g = 4$ fixed, to compute $\bar{t}_i = F(\phi_i)$. When compared to the exact value t_i, the highest relative error[2] is $\sim 10^{-8}s$ (the complete output of this process is not included here). The couples (t_i, ϕ_i) are shown in the following figure.

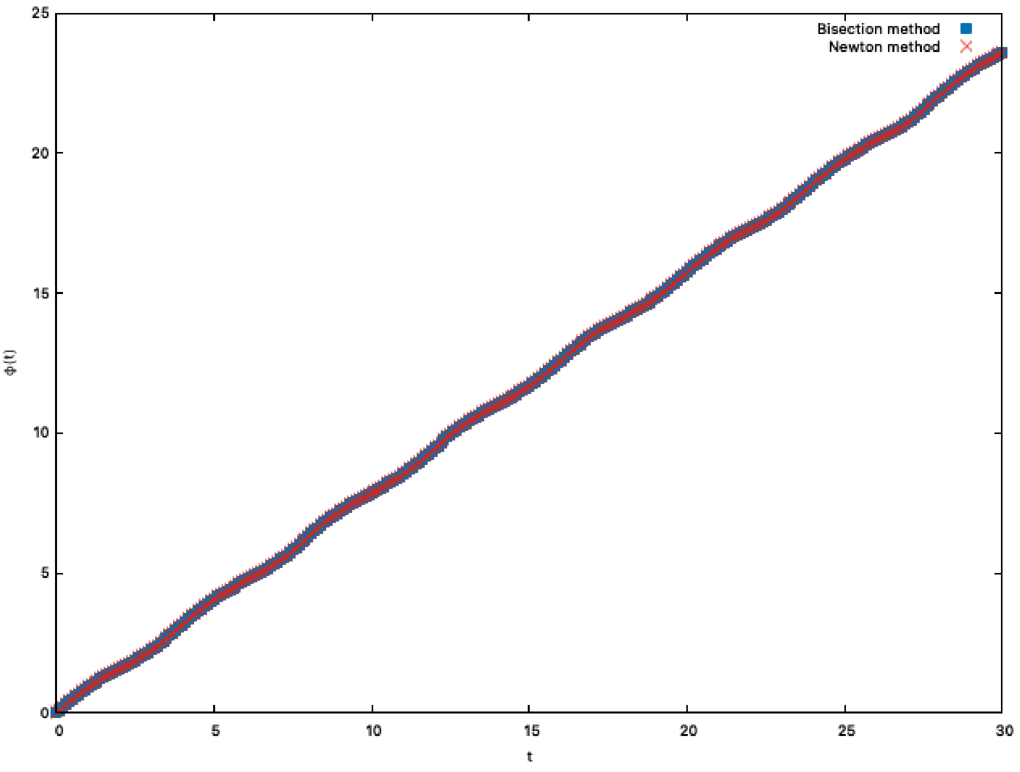

Figure 1: **Gnuplot plot from data in "dati.dat"**. Since both root solvers are given the same tolerance ($\sim 10^{-8}$), the outputs shown are overlapping better than the resolution power of the human eye (further confirmation is given by the check on $\bar{t}_i$).

1 Code

The header file *lib.h* of the local library contains the declaration of useful functions such as quadrature methods and root solvers, whereas functions specifically implemented for this homework are declared and defined in the main file: `Integrand(...)` and `Integral_Function(...)` have self-explanatory names, while `Func(...)` is $Z(\phi, t_i)$, whose roots are calculated to obtain $\phi(t)$.
Preprocessor directives are employed to set the value of k, the number of gaussian points, the starting value of N (number of sub-intervals for gaussian quadrature) and the tolerance.

[2]A branch in the loop prevents the calculation if $t_i = 0.0$

In the third section a stream to the output file “dati.dat” is opened. In “dati.dat” the roots ϕ_i and their respective t_i are stored.

```
#include "lib.h"

#define CONST_K 0.8
#define N_GAUSS 4
#define NUM_INTERVALS_0 4
#define TOL 1e-8

using namespace std;

static double g_time = 0.;

//Declaration
double Integrand(double);
double Integral_Function(double);
double Func(double);

/////////////////////////////////////////////////////////////////////////////////////////
int main(){
// -1
    double a = 0.;
    double b = M_PI * 0.5;
    double previous_integral, err;
    unsigned int n_intervals = NUM_INTERVALS_0;
    double integral = GaussQuadratureRule(Integrand, a, b, n_intervals, N_GAUSS);

    do{
        previous_integral = integral;
        n_intervals++;
        integral = GaussQuadratureRule(Integrand, a, b, n_intervals, N_GAUSS);
        err = fabs(integral - previous_integral);
    }while(err > TOL);

    integral *= 4;

    cout << "\n\nTolerance: " << TOL;
    cout << "\nMethod: Gauss-Legendre quadrature (N = " << n_intervals << ", N_g = " <<
    N_GAUSS << ")";
    cout << "\nPeriod: T = " << setprecision(10) << integral << "s\n";

// -2
    double small_angle = 2 * M_PI;
    cout << "\nSmall-angle approximation: T_approx = " << small_angle << "s";
    cout << "\nRelative error: err = " << setprecision(9) << fabs(small_angle / integral
    - 1);

// -3
    double zero_b, zero_n, rel_err_b, rel_err_n;
    double tmp_b, tmp_n;
    ofstream out;
    out.open("dati.dat");
    if(out.fail()){
        cerr << "\nError: stream to output file has failed.";
        exit(1);
    }

    for(int i = 0; i <= 300; i++){
        g_time = i * 0.1;
        Bisection(Func, a, 24, TOL, zero_b);
        Newton(Func, Integrand, 0., M_PI * 0.5, TOL, zero_n);
        out << setw(6) << g_time << setw(30) << setprecision(12) << zero_b;
        out << setw(30) << setprecision(12) << zero_n << "\n";
        tmp_b = Integral_Function(zero_b);
        tmp_n = Integral_Function(zero_n);
        if(i != 0){
            rel_err_b = fabs(tmp_b / g_time - 1);
            rel_err_n = fabs(tmp_n / g_time - 1);
            cout << "\n\nReal value t_i: " << g_time;
```

```cpp
            cout << "\nRelative error Bisection: " << rel_err_b;
            cout << "\nRelative error Newton: " << rel_err_n;
        }
    }

    out.close();

    return 0;
}

////////////////////////////////////////////////////////////////////////////////////////
//Definition
double Integrand(double x){
    return 1. / (sqrt(1. - CONST_K*CONST_K*sin(x)*sin(x)));
}

double Integral_Function(double x){
    return GaussQuadratureRule(Integrand, 0., x, 10, N_GAUSS);
}

double Func(double x){
    return Integral_Function(x) - g_time;
}
```