Elliptic Integrals

Marco Casari

November 9, 2022

1 Introduction

Same notation and formulas shown in file elliptic_integrals.pdf are used in this document. Whenever a new formula is derived, its form is general: it is always specified explicitally when a particular choice for some values is applied (e.g. assumption on value of parameter k).

The tollerance adopted for the numerical calculations is 10^{-7}. Whenever an integral is evaluated numerically, gaussian quadrature is used with 4 gaussian points and a number of subintervals chosen for each case.

2 Point 1

For the numerical integration of function $F(\frac{\pi}{2}, k)$ a number of subintervals $N = 5$ is set, because for greater N the difference between consecutive quadratures is less or equal than the chosen tollerance. The resulting value for oscillation period T is in equation 1.

$$T = 7.9812111 \ s \tag{1}$$

3 Point 2

Starting from the general formula for $F(\phi, k)$, the small angles approximation $\sin(u) \ll 1$ lets the integrand be rewritten using an expansion at first order. The resulting integral can be solved analytically, leading to equation 2.

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

Period T evaluated numerically in section 2 is considered the reference value, so that the accuracy of small angles approximation can be shown through the relative error in equation 3.

$$\sigma[T_{\text{approx}}] = \left|\frac{T_{\text{approx}} - T}{T}\right| = 2.127529 \ 10^{-1} \tag{3}$$

4 Point 3

The inverse function $\phi(t)$ is obtained by plotting points (t, ϕ) where $t = 0, 0.1, 0.2, \ldots, 30$. Since the integrand function is always positive in $\mathcal{R}$, function $h(\phi)$ as defined in equation 4 is monotone, hence it has a single zero in $\mathcal{R}$ for any given value of t and no bracketing procedure needs to be applied. Fixed a value of t, the corresponding ϕ is found by solving equation $h(\phi) = 0$.

$$h(\phi) = F(\phi, k) - t \tag{4}$$

Bisection and Newton-Raphson algorithms are used as root finders. Roots are searched in interval $[-1, 28]$, the same interval is chosen for both algorithms for ease. The lower bound is chosen negative to let the algorithms search for the root around the origin, thus without fixing point $(0, 0)$ which is known

analitically. The upper bound is an arbitrary choice made to include all possible values of ϕ given the maximum value of t: it is estimated by evaluating $F(\phi, k)$ in a loop where ϕ varies until the result is greater than $t = 30$. Both values are integers to avoid the representation error of floating point arithmetic.

The number of subintervals for the quadrature of $F(\phi, k)$ is fixed. It is selected arbitrarily to be greater than the maximum number of intervals obtained by varying the interval at each iteration of any root finder (cfr. method presented in section 2 for a single interval).

The resulting plot is shown in figure 1. Figure 2 is a magnification of a region of the plot which shows how roots finded by different algorithms at the same t can be at most equal to the chosen tollerance.

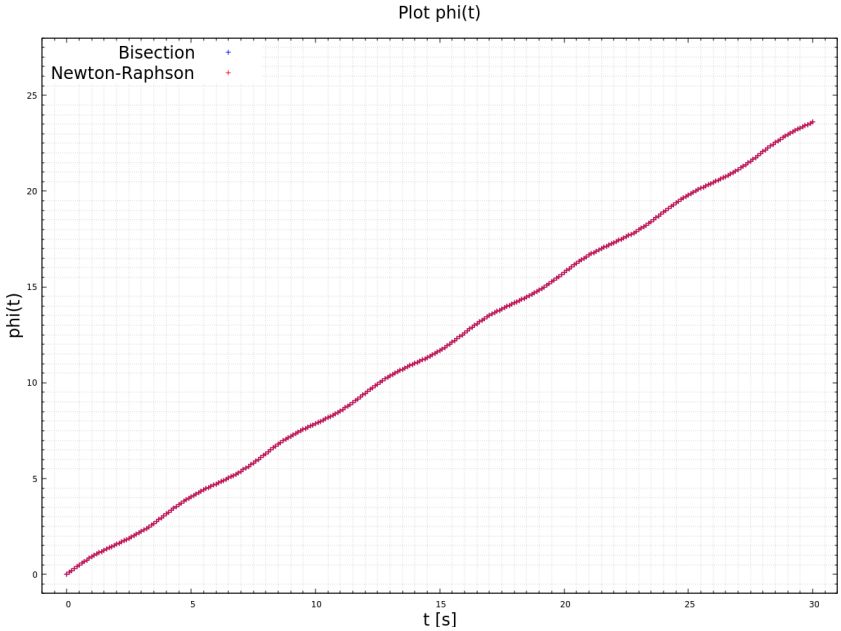

Figure 1: Plots of $\phi(t)$ for every root finding algorithm used.

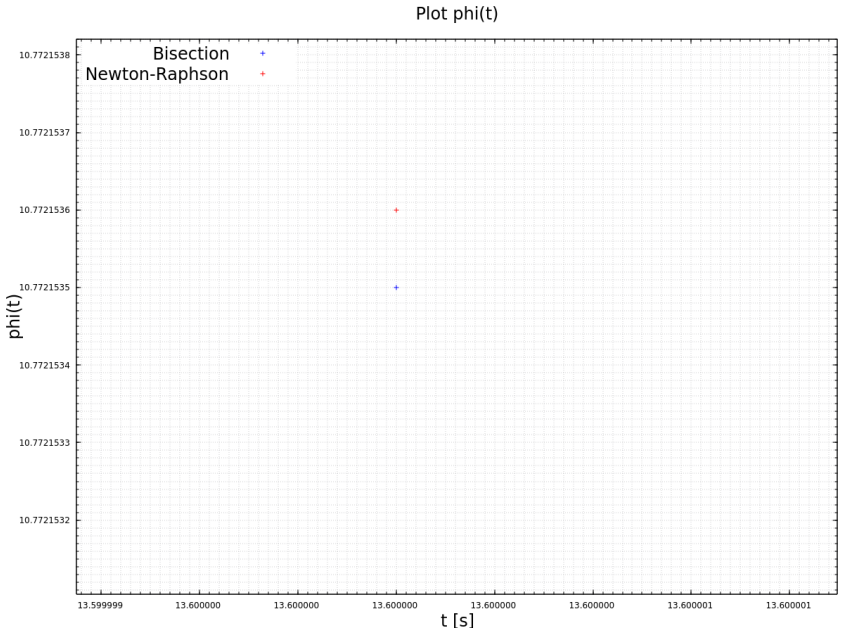

Figure 2: The difference between results of different algorithms can be at most the chosen tollerance.

5 Source code

```cpp
#include <cmath>
#include <fstream>
#include <iomanip>
#include <iostream>

#define HOMEWORK_NAME "elliptic_integrals"
#define N_PRECISION 8
#define TOLLERANCE 1e-7
#define N_GAUSSPOINTS 4

/* Global variables */
static double const global_k = 0.8;
static double global_t = 0.0;

/* Function prototypes */
double g(double u);
double F(double phi);
double h(double phi);

int main() {
  // Setup.
  using namespace std;
  cout << setprecision(N_PRECISION) << scientific;
  cout << "Homework " HOMEWORK_NAME << endl;

  // Point 1.
  cout << "Point 1" << endl;
  double T, a, b;
  a = 0.0;
  b = M_PI / 2.0;
  // Assume L / g = 1.
  T = 4.0 * gaussquad(g, a, b, 5, N_GAUSSPOINTS);
  cout << "Gauss: T = " << T << endl;

  // Point 2.
  cout << "\nPoint 2" << endl;
  double T_approx, sigma_T_approx;
  // Assume L / g = 1.
  T_approx = 2.0 * M_PI;
  cout << "T_approx = " << T_approx << endl;
  sigma_T_approx = fabs(T_approx / T - 1.0);
  cout << "Gauss: sigma[T_approx] = " << sigma_T_approx << endl;

  // Point 3.
  cout << "\nPoint 3" << endl;
  double phi_t, t_step;
  ofstream file_plot;
  file_plot.open(HOMEWORK_NAME ".dat");
  file_plot << setprecision(N_PRECISION) << scientific;
  file_plot << "t phi_bisection(t) phi_newton(t)";
  file_plot << endl;
  t_step = 0.1;
  // Conservative values to be able to invert the function to the minimum and maximum t
    requested.
  a = -1.0;
  b = 28.0;
  // Loop on t.
  for (int i = 0; i <= 300; ++i) {
    global_t = i * t_step;
    file_plot << global_t;
    phi_t = bisection(h, a, b, TOLLERANCE);
    file_plot << ' ' << phi_t;
    phi_t = newtonraphson(h, g, a, b, TOLLERANCE);
    file_plot << ' ' << phi_t;
    file_plot << endl;
  }
  cout << "Data written in file " HOMEWORK_NAME ".dat" << endl;
```

```
67
68    // Teardown.
69    file_plot.close();
70    return 0;
71  }
72
73  /* Function definitions */
74  double g(double u) {
75    return 1.0 / sqrt(1.0 - global_k*global_k * sin(u)*sin(u));
76  }
77
78  double F(double phi) {
79    double _return, phi_0;
80    phi_0 = 0.0;
81    _return = gaussquad(g, phi_0, phi, 64, N_GAUSSPOINTS);
82    if (phi < phi_0) {
83      _return = -_return;
84    }
85    return _return;
86  }
87
88  double h(double phi) {
89    return F(phi) - global_t;
90  }
```