N Body Problem

Ben McMurtry

11th November 2016

1 Introduction

The n-body problem refers to the problem of predicting the motion of a group of celestial bodies, interacting due to gravity. The Sun-Earth-Moon system is an example of such a problem.

A program was written to solve the N-body problem, for any number of celestial bodies, by reading their initial conditions from a text file, where each line represents a separate body, and then using either a Velocity Verlet method to calculate further positions or an algorithm from the GSL library for solving Ordinary Differential Equations. These methods then put the new position data into a file to be plotted, showing the trajectories of the bodies.

The code also calculates Kinetic Energy and Potential Energy at each stage??? to check how well they are conserved (Errors are expected due to the discrete timesteps and double arithmetic).

The accelerations of each body in each Cartesian coordinate direction were calculated as???

$$acceleration[x] = \frac{-GM\hat{x}}{r^3} \tag{1}$$

For a membrane of radius R, the vertical displacement u must be real, zero at the edge of the drum, and finite at the centre. Assuming that the initial displacement and velocity is axially symmetric, the solutions will involve Bessel functions of the first kind $J_k(x)$ and be of the form

$$u(r,t) = \sum_k J_0(\lambda_k r)(A_k cos(\lambda_k ct) + B_k(\lambda_k ct)) \tag{2}$$

where the allowed values of λ_k must satisfy:

$$J_0(\alpha_k) = 0 \quad where \quad \alpha_k = \lambda_k R \tag{3}$$

The values of A_k and B_k are the coefficients, which must be calculated by numerical integration of the Fourier-Bessel series that matches the initial displacement $u(r,0)$ and velocity $\dot{u}(r,0)$ conditions.

In order to answer questions about the time independent displacement of the drum $u(r)$, equation (3) above must be solved.

The program HW2/main.c was written to find the roots (technically an approximation to the roots defined by macro error tolerances for double arithmetic), of J_0 (or any other function provided that aMin and aMax themselves are not roots of that function, as this leads to a problem with finding whether aMin and aMax have opposite sign), for a range specified in the program macros (aMin and aMax).

The program takes the function of aMin and aMax, and checks if they have opposite sign, if they do not, then it takes the function of a random guess within the two, and checks that against each again. It does this until the functions of 2 values have opposite sign, and when this is the case, it narrows the range in which the root lies by either the bisection or secant method as specified by the user.

The bisection method narrows the region in which the root lies by iteratively taking a new value halfway between the initial guesses and checking which half the root lies in. This is repeated until the two values are within a certain range of each other, at which point the root and its uncertainty are calculated using those two values.

The secant method in the program narrows the region by finding the secant line between the two initial values, then finding the x-intercept of that line, and using that as a new value iteratively. The secant function in the program is technically not a secant method, but a "Regula Falsi" (or false position) method, since it then checks to see which side of the x-intercept the root lies in, which a normal secant method does not do (this can lead to the secant method not converging, and even leaving the initial range as shown in Figure **??**.

Having found a root, with its corresponding uncertainty, the program then stores the root, uncertainty and number of iterations required to find it in a linked list. It then deflates the current function at that root (x_n), by dividing the function by $(x - x_n)$. This is done iteratively until all roots are deflated and no roots can be found.

The roots, uncertainties and required iterations are then printed to the standard output, or to a .txt file.

2 Discussion

In looking at the output of the program in section 4, we can see that the bisection method proves to be robust, always finding the all 5 roots, to within the error tolerances, provided that the maximum number of iterations is above ≈50 iterations. The method appears to be accurate to ≈ 12 d.p. or higher when compared to sourced values in Figure **??**.

In contrast the secant method (or Regula Falsi method), is not as robust, finding all 5 roots, but often (not always) hitting the defined maximum number of iterations without converging to the root within the defined error tolerances, for one or more roots, depending on the seed of the random number generator srand(). In the case that it does converge to the root within the error tolerances, it does so much faster than the bisection method, ≈10-20 iterations, and with

an accuracy again of ≈ 12 d.p. or higher when compared to the sourced values in Figure **??**.

The secant method is known to not reliably converge if the initial values are not "close enough" to the root. In this case it can find a secant line which points it to an x-intercept which was not between the previous 2 bounds, as shown in Figure **??**.

With the regula falsi method, this is fixed by the check to see which side of the intercept the root lies in, however it can still be unable to reduce the bounds of the region in which the root lies to a small range.

If the initial values x_0 and x_1 are chosen such that $f(x_0)$ and $f(x_1)$ are of opposite signs, then at each step, one of the values will get closer to a root of the function. However, unlike the bisection method this does not force the range between the two values towards 0. And can in fact cause the function to bounce back and forth between two nearby values (line 69 of the code can allow you to watch this happen).

Another potential flaw in this program, is that the roots found have the potential to accumulate inaccuracy, as each deflated root is not perfectly accurate, this leads to small shifts in the whole function each time an inaccurate root is deflated. This also meant that a function had to be made to check each time a root was found, that it was not within the uncertainty of one of the other roots (i.e. a root being found twice due to innacurate deflation).

This could be fixed by processing the roots a second time, using the bisection or secant method again, but with a very small range created around the root, and using the undeflated original function.

3 Conclusions

This program can accurately find the roots of the Bessel function, robustly, and accurately, but (comparatively slowly with ≈50 iterations) using the bisection method to narrow the range of the root to an uncertainty of ≈1x10^{-15}. Or the program can find the roots of the Bessel function accurately and quickly (≈ 10 iterations) but less robustly using the secant (regula falsi) method, to an uncertainty of ≈1x10^{-15} unless the maxevals limit is hit, where the accuracy is lost, and the uncertainty can vary from the order of 1x10^{-14} up to order of 1. These conclusions are in good agreement with the lectures by CDHW.

The most useful implementation of these two methods would be some combination of the two (akin to Dekkers method or Brents method[5]) where secant is used to quickly narrow the range, with bisection taking place if secant would not effectively reduce the range.

4 Output

The results of compiling the program in Xcode are shown below. The maximum iteration number was set at 100 and the error tolerances were set at 1e-15. The

range in which to find roots was 0.0 to 15.0. The bisection method has been run three times and the secant method three times (some addional returns have been added here for better page formatting):

```
Bisection

Please enter 'b' for bisection or 's' for secant: b
5 roots were found in the range 0 to 15
Root----------------Uncertainty---------Iterations
5.520078110286313   4.88498e-15         50
2.404825557695772   3.10862e-15         49
14.93091770848778   1.15463e-14         46
11.79153443901428   1.15463e-14         49
8.653727912911005   6.21725e-15         50
Program ended with exit code: 0

Please enter 'b' for bisection or 's' for secant: b
5 roots were found in the range 0 to 15
Root----------------Uncertainty---------Iterations
5.520078110286311   4.88498e-15         50
2.404825557695773   2.22045e-15         50
14.93091770848779   1.24345e-14         46
11.79153443901429   1.24345e-14         49
8.653727912911005   6.21725e-15         50
Program ended with exit code: 0

Please enter 'b' for bisection or 's' for secant: b
5 roots were found in the range 0 to 15
Root----------------Uncertainty---------Iterations
5.520078110286307   4.44089e-15         50
2.404825557695774   2.44249e-15         50
14.93091770848777   1.24345e-14         46
11.79153443901427   1.15463e-14         49
8.653727912911005   6.21725e-15         50
Program ended with exit code: 0
```

```
Secant

Please enter 'b' for bisection or 's' for secant: s
5 roots were found in the range 0 to 15
```

```
Root----------------Uncertainty---------Iterations
6.624468405057883   1.10439             100
2.404825557695771   2.22045e-15         11
11.79153443901428   8.88178e-16         10
8.653727912911013   8.88178e-16         8
14.93091770848779   8.88178e-16         7
Program ended with exit code: 0

Please enter 'b' for bisection or 's' for secant: s
5 roots were found in the range 0 to 15
Root----------------Uncertainty---------Iterations
5.520078110286311   8.88178e-16         22
2.404825557695772   8.88178e-16         10
11.79153443901428   8.88178e-16         8
8.653727912911005   7.10543e-15         13
14.93091770848779   8.88178e-16         7
Program ended with exit code: 0

Please enter 'b' for bisection or 's' for secant: s
5 roots were found in the range 0 to 15
Root----------------Uncertainty---------Iterations
8.653727912911016   4.44089e-15         12
5.520078110286313   2.66454e-15         14
11.85946018639934   0.0679257           100
2.345858362161297   0.0589672           100
14.93091770848779   8.88178e-16         7
Program ended with exit code: 0
```

References

[1] http://mathworld.wolfram.com/BesselFunctionZeros.html

[2] https://en.wikipedia.org/wiki/Bisection_method

[3] https://en.wikipedia.org/wiki/Secant_method

[4] http://wwwal.kuicr.kyoto-u.ac.jp/www/accelerator/a4/besselroot.htmlx

[5] https://en.wikipedia.org/wiki/Brent\%27s_method