Physics 750: Exercise 6

Tuesday, September 14, 2017

1. Use the `curl` command to download from the class website everything you'll need for the lab.

```
$ WEBPATH=http://www.phy.olemiss.edu/~kbeach/
$ curl $WEBPATH/courses/fall2017/phys750/src/exercise6.tgz -O
$ tar xzf exercise6.tgz
$ cd exercise6
```

2. Consider a satellite orbiting the earth with period P whose distances of closest and farthest approach are r_1 and r_2. The satellite's path traces out an ellipse,

$$r(\theta) = \frac{a(1-e^2)}{1+e\cos(\theta)} = a(1 - e\cos E).$$

The formula above assumes polar coordinates (r, θ) in the plane of the orbit with the earth centred on one of the ellipse's foci. The ratio

$$e = \frac{r_2 - r_1}{r_1 + r_2} = \frac{r_2 - r_1}{2a}$$

defines the eccentricity of the orbit; a is the semi-major axis, and E is the so-called *eccentric anomaly*.

Kepler worked out the following procedure for determining the location of the satellite at time t, as measured from the moment of closest approach:

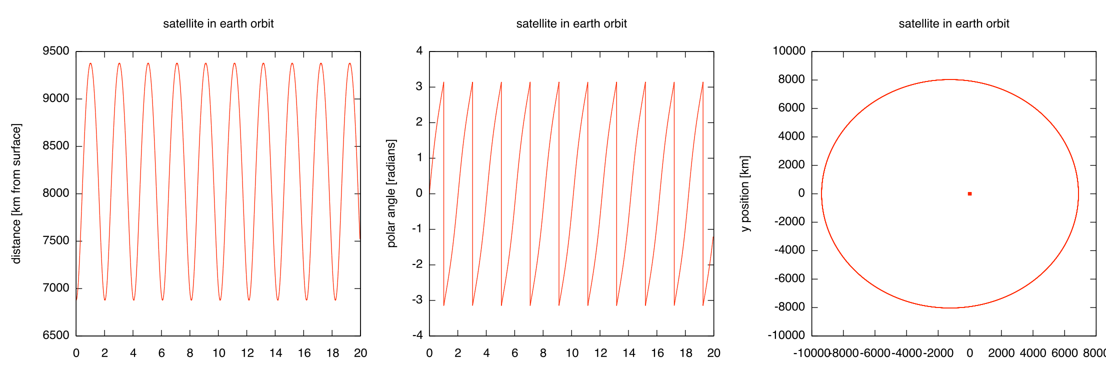
- Define the mean anomaly $M = 2\pi t/P$.
- Determine the eccentric anomaly E by solving Kepler's equation, $M = E - e\sin E$.
- Compute the true anomaly θ from the equation

$$\tan\frac{\theta}{2} = \sqrt{\frac{1+e}{1-e}}\tan\frac{E}{2}.$$

Kepler's equation is transcendental and has no closed-form solution. It has to be inverted numerically. In the file `orbit.cpp`, solve Kepler's equation by finding the root of the function $f(E) = E - M - e\sin E$ via Newton-Raphson iteration:

$$E_{n+1} := E_n - \frac{f(E_n)}{f'(E_n)} = E_n - \frac{E_n - M - e\sin E_n}{1 - e\cos E_n}.$$

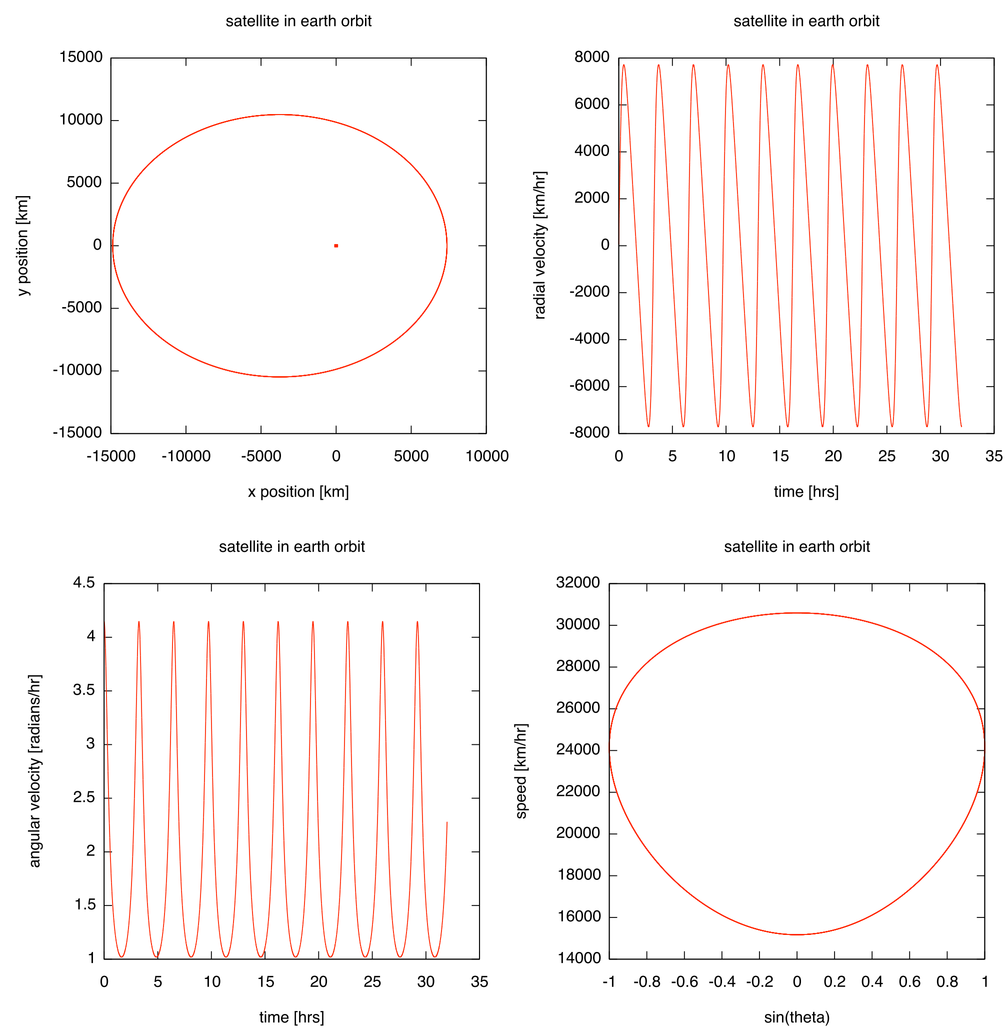
Use the initial guess $E_0 = 0$ at time $t = 0$, and for each subsequent time step use the previous step's converged E value as the initial guess. Make sure that a minimum number of iterations are always performed. The `view2.gp` script should give you the following plots.



3. Write a new program `orbit_vel.cpp` that functions identically to `orbit.cpp` from question 1 except that it outputs five columns of data: time t, radius r, angle θ, radial velocity $\dot{r}$, and angular velocity $\dot{\theta}$. The time derivatives $\dot{r}$ and $\dot{\theta}$ should be approximated as symmetric finite difference. Some care must be taken in computing $\dot{\theta}$ since θ is compact on $[0, 2\pi]$.

   In `orbit.cpp`, the closest- and farthest-approach values were set to $r_1 = R_e + 500\,\text{km}$ and $r_2 = R_e + 3000\,\text{km}$, where R_e is the radius of the earth. For `orbit_vel.cpp`, consider a more eccentric orbit with $r_1 = R_e + 1000\,\text{km}$ and $r_2 = R_e + 8500\,\text{km}$. Compose a gnuplot script `view3.gp` that plots the satellite's spatial trajectory, its radial velocity versus time, its angular velocity versus time, and its total speed versus $\sin\theta$, assuming that the data is in a file `ov.dat`. (Arrange these as four plots in succession, separated by a `pause -1` command.) In the last plot, be sure to compute the speed as the magnitude of the vector $\boldsymbol{v} = \dot{r}\hat{r} + r\dot{\theta}\hat{\theta}$.

```
$ make orbit_vel
$ ./orbit_vel > ov.dat
$ gnuplot -persist view3.gp
```



4. Recall that Newton's method is an iterative scheme for finding the zeros of an arbitrary function $f(x)$. It involves making an initial guess x_0 and then generating a sequence of improved estimates according to

$x_{n+1} := x_n - f(x_n)/f'(x_n)$. If we choose $f(x) = x^2 - a$, then finding the zeros of $f(x)$ is equivalent to computing the square root of a. The correct recurrence relation is

$$x_{n+1} := \frac{1}{2}\left(x_n + \frac{a}{x_n}\right).$$

The program `root.cpp` implements the recurrence relation shown above, starting from `x = 1`. The loop terminates when the next value in the sequence is sufficiently close to the old one.

```
$ make root
g++ -o root root.cpp -O2 -ansi -pedantic -Wall -lm
$ ./root
Returns the square root of the provided argument:
Usage: root # [--verbose]
./root 2
Newton's method value: 1.414213562373095
 C Math library value: 1.414213562373095
$ ./root 9
Newton's method value: 3
 C Math library value: 3
$ ./root 101010
Newton's method value: 317.8207041713928
 C Math library value: 317.8207041713928
```

In general, computing square roots via series expansion is much less reliable, but let's give it a try. The square root $\sqrt{a^2+b}$ can be expanded in powers of $b/4a^2$ as follows:

$$\begin{aligned}\sqrt{a^2+b} &= a + \frac{1}{2}\frac{b}{a} - \frac{1}{8}\frac{b^2}{a^3} + \frac{1}{16}\frac{b^3}{a^5} - \frac{5}{128}\frac{b^4}{a^7} + \cdots \\ &= a + \frac{b}{2a} + \sum_{n=1}^{\infty}(-1)^n C_n \frac{b^{n+1}}{(2a)^{2n+1}} \\ &= a + \frac{b}{2a}\left(1 + \sum_{n=1}^{\infty}(-1)^n C_n \frac{b^n}{(2a)^{2n}}\right).\end{aligned}$$

Here, $(C_n) = (1, 2, 5, 14, 42, 132, 429, 1430, \ldots)$ are the *Catalan numbers*. They are defined by

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}$$

and describe the number of ways a polygon with $n+2$ sides can be cut into n triangles. For large values of n, the factorials are too large to compute, so we should use the trick of computing each term from the previous one. The ratio of two consecutive terms is

$$\frac{(-1)^{n+1}C_{n+1}b^{n+1}}{(2a)^{2n+2}}\frac{(2a)^{2n}}{(-1)^n C_n b_n} = -\frac{b(2n+2)(2n+1)}{4a^2(n+1)(n+2)}.$$

Write a program `seriesroot.cpp` that computes the truncated N-term series expansion for a given list of N values. (In other words, `argc` can have any value greater than `3`, and the program should loop over all `N` assigned from `argv[3]`, `argv[4]`, ..., `argv[argc-1]`.) You should be able to generate the one- through ten-term approximations to $\sqrt{2} = \sqrt{1^2+1}$ and $\sqrt{3} = \sqrt{2^2-1}$ as follows. The script `view4.gp` illustrates the convergence rates for Heron's method and three different series approximations to $\sqrt{3}$.

```
$ make seriesroot
g++ -o seriesroot seriesroot.cpp -O2 -ansi -pedantic -Wall -lm
$ ./seriesroot
Computes the N-term series expansion of sqrt(a^2+b):
Usage: seriesroot a b N1 [N2 N3 N4 ...]
$ ./seriesroot 1 1 $(seq 10)
             1                    1
             2                  1.5
             3                1.375
             4               1.4375
             5            1.3984375
             6           1.42578125
             7         1.4052734375
             8        1.42138671875
             9    1.408294677734375
            10    1.419204711914062
$ ./seriesroot 2 -1 $(seq 10)
             1                    2
             2                 1.75
             3             1.734375
             4          1.732421875
             5     1.73211669921875
             6    1.732063293457031
             7    1.732053279876709
             8    1.732051312923431
             9    1.732050913386047
            10    1.732050830149092
$ gnuplot -persist view4.gp
```