PS7

Shihong Pan
https://github.com/PSH-hub24/phys-ga2000

October 30, 2024

Q1

(a) The gravitational force exerted on the Moon by the Earth is

$$F_{\text{moon}} = \frac{GMm}{R^2} := m\omega^2 R \tag{1}$$

so ω can be written as $\omega^2 = GM/R^3$. Now, the satellite has the same angular frequency at L1, so the centripetal force it experiences is

$$F_{\text{sate}} = m\omega^2 r. \tag{2}$$

The field strength at the satellite is the sum of the field strength due to the Earth and the Moon

$$g_{\text{sate}} = \frac{GM}{r^2} - \frac{Gm}{(R-r)^2} = \frac{F_{\text{sate}}}{m} = \omega^2 r = \frac{GMr}{R^3}. \tag{3}$$

To rescale the equation, substitute $m = m'M$ and $r = r'R$ into the equation above, such that

$$\frac{GM}{r'^2R^2} - \frac{Gm'M}{R^2(1-r')^2} = \frac{GMr'R}{R^3}. \tag{4}$$

Multiply R^2/M on both sides and rearrange to get

$$f(r') = \frac{G}{r'^2} - \frac{Gm'}{(1-r')^2} - Gr' = 0. \tag{5}$$

where m' is a constant depending on the masses of the two planets.

(b) The codes compute the root of $f(r')$ by first using a simple bracketing function to find an interval that includes a root, and use the mid-point of the interval as the initial guess for Newton's method (please see the codes for details). The derivative of $f(r')$ is obtained by using jax.grad.

Fig 1 shows the initial guesses and the r' values for all three cases (the Earth and the Moon, the Earth and the Sun, the Jupiter-mass planet and the Sun). Note that in the later two cases, r' measures the relative distance from the Sun to the other planet. So if we want to talk about the distance r from the Earth in all three cases, we have

- The Earth and the Moon: $r' \approx 0.8489$, $R = 3.844 \times 10^5$ km, so $r = 3.263 \times 10^5$ km.
- The Earth and the Sun: $r' \approx 1 - 0.9900 = 0.0100$, $R = 1.49 \times 10^8$ km, so $r = 1.49 \times 10^6$ km.
- The Jupiter-mass planet and the Sun: $r' \approx 1 - 0.9333 = 0.0667$, $R = 3.844 \times 10^5$ km, so $r = 2.5639 \times 10^4$ km.

Q2

For the parabolic approximation I used the one given in the root-finding lecture notes (section 4 Brent's method). Then I just followed the description given in the minimization lecture notes (section 4 Brent's 1d minimization). I set the termination condition by comparing the change between the previous minimum value and the current minimum value to the magnitude of the current minimum value times a tolerance (I set the tolerance to 1e-15). If the change is smaller than the tolerance, then the iterations terminate. Fig 2 shows the output of my Brent function and that of scipy.optimize.brent. It turns out that my number is closer to 0.3.

```
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun for more info.)
initial guess: 0.851
r/R value of L1 between the Earth and the Moon: 0.8489032983779907
initial guess: 0.9504999999999999
r/R value of L1 between the Sun and the Earth: 0.9900295734405518
initial guess: 0.9504999999999999
r/R value of L1 between the Sun and the Jupiter-mass planet (at the Earth distance): 0.9333252310752869
[Finished in 1.8s]
```

Figure 1: The r/R values and the initial guesses.

```
51  a = 0
52  c = 0.5
53  b = omega*(c-a) + a
54  tol = 1e-15
55  maxIter = 50
56  xMin = Brent(a, b, c, tol, maxIter)
57  print(f'My Brent function gives x_min = {xMin}.')
58
59  xScipyMin = scipy.optimize.brent(y)
60  print(f'Scipy\'s Brent function gives x_min = {xScipyMin}.')
61
62  xMinDiff = np.abs(xMin-0.3)
63  xScipyMinDiff = np.abs(xScipyMin-0.3)
64  print(f'Boolean value of whether my x_min is closer to 0.3 compared to scipy\'s x_min: {xMinDiff<xScipyMin}')
65
```

```
My Brent function gives x_min = 0.3000000000007488.
Scipy's Brent function gives x_min = 0.300000000023735.
Boolean value of whether my x_min is closer to 0.3 compared to scipy's x_min: True
[Finished in 3.0s]
```

Figure 2: Compare my Brent's 1d minimization to scipy's minimization.