CS 322: Assignment P3

Due: Friday, March 8, 2002, 4:30pm (in Upson 4130)[1]

You may work in pairs. Follow the course rules for the submission of assignments. Do not submit work unless you have adhered to the principles of academic integrity as descibed on the course website. Points will be deducted for poorly commented code, redundant computation that seriously effects efficiency, and failure to use features of MATLAB that are part of the course syllabus.

Part A (8 pts) Kepler's Second Law

Kepler's First Law states that the planets have elliptical orbits with the Sun at one focus. The size and elongation of the orbit are completely defined by the planet-Sun separation at perihelion and aphelion. Designate these values by P and A ($0 < P \leq A$). If we situate the Sun at (0,0), the perihelion point at $(P, 0)$, and the aphelion point at $(-A, 0)$, then the orbit is the set of points $\mathcal{E}(P, A) = \{(x(\tau), y(\tau)) \mid 0 \leq \tau \leq 2\pi\}$ where

$$\begin{aligned} x(\tau) &= \frac{P - A}{2} + \frac{P + A}{2}\cos(\tau) \\ y(\tau) &= \sqrt{AP}\sin(\tau). \end{aligned}$$

In polar coordinates $\mathcal{E}(P, A) = \{(\theta, r(\theta)) \mid 0 \leq \theta \leq 2\pi\}$ where

$$r(\theta) = \frac{2AP}{P(1 - \cos(\theta)) + A(1 + \cos(\theta))}.$$

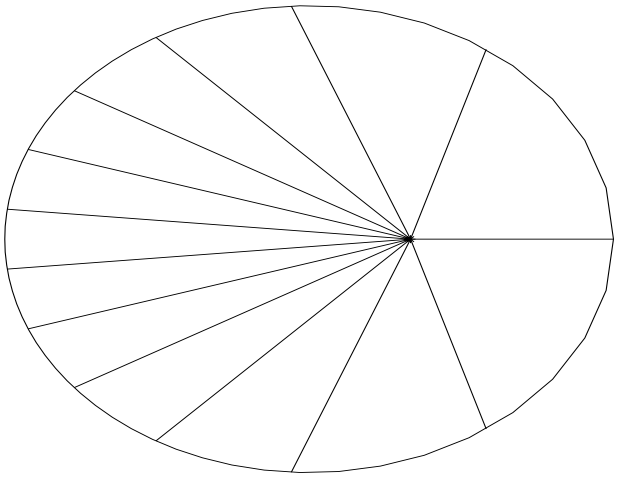
(Here, θ is the polar angle and $r(\theta)$ is the length of the radius vector.) The area "swept out" by the radius vector as it "rotates" from $\theta = \theta_1$ to $\theta = \theta_2$ is given by

$$\alpha(\theta_1, \theta_2) = \frac{1}{2}\int_{\theta_1}^{\theta_2} r(\theta)^2 d\theta.$$

Kepler's Second Law states that the radius vector sweeps out area at a constant rate. In particular, if T is the period of revolution, then

$$t(\theta_1, \theta_2) = \frac{\alpha(\theta_1, \theta_2)}{\alpha(0, 2\pi)}T$$

is the time it takes the planet to move from $(\theta_1, r(\theta_1))$ to $(\theta_2, r(\theta_2))$. In the picture, each of the 12 sectors has equal area. If the period of revolution is 360 days and the planet is at the perihelion point $(P, 0)$ at $t = 0$, then



[1] May also be submitted in lecture or section.

it would pass into a new sector every 30 days as it moved counterclockwise in its orbit.

You will produce an animation of the orbiting planet using the built-in MATLAB function `comet`. You first need to precompute planet locations as a function of time. At first glance it looks like we have some time-consuming numerical integrations to perform for each t-value. A way around this is to construct a cubic spline S with the property that $\theta = S(t)$ is a good approximation to the polar angle of the planet at time t. Here is how to construct S:

- Let $nSect$ be a positive integer ≥ 2 and set $\theta = linspace(0, 2\pi, nSect + 1)$. Our plan is to break up the area enclosed by the ellipse into $nSect$ sectors. The sectors will have unequal areas even though as "pizza slices" they have identical "pizza angles".
- For $k = 1{:}nSect$ use Simpson's rule to compute an estimate α_k of the true sector area $\alpha(\theta_k, \theta_{k+1})$.
- Define $t_1, \ldots, t_{n+1}$ by

$$t_k = \begin{cases} 0 & k = 1 \\ T\dfrac{\alpha_1 + \cdots + \alpha_{k-1}}{\alpha_1 + \cdots + \alpha_n} & k = 2{:}n+1 \end{cases}$$

  Kepler says that the planet has polar angle θ_k at time t_k. (Well, not quite since the α_k are Simpson rule estimates of the true sector areas.) We are assuming that at $t = 0$ the planet has polar angle 0, i.e., it is situated at the perihelion point $(x, y) = (P, 0)$.
- Let S be the cubic spline interpolant of $(t_1, \theta_1), \ldots, (t_{nSect+1}, \theta_{nSect+1})$.

Write a function `S = OrbitSpline(P,A,T,nSect)` that computes S as defined by this process. Use the MATLAB `spline` function. You do not have to exploit symmetry when computing the α_k. But vectorize!

Also write a function `[x,y] = OrbitVals(S,P,A,t)` where `S` is the cubic spline as produced by `orbitSpline` and `t` is a column vecctor made up of nonnegative values. It returns planet location at the times specified by the column vector t. In particular, `x` and `y` should be column vectors with the property that the planet is located at (`x(k)`,`y(k)`) at time `t(k)`. Note that S interpolates across $[0, T]$ so you'll have to use periodicity in order to locate the planet for t-values that are greater than T. E.g., $S(.23T)$ specifies the polar angle for the planet at time $3.23T$. You might want to make use of the `rem` function to handle the periodicity computations.

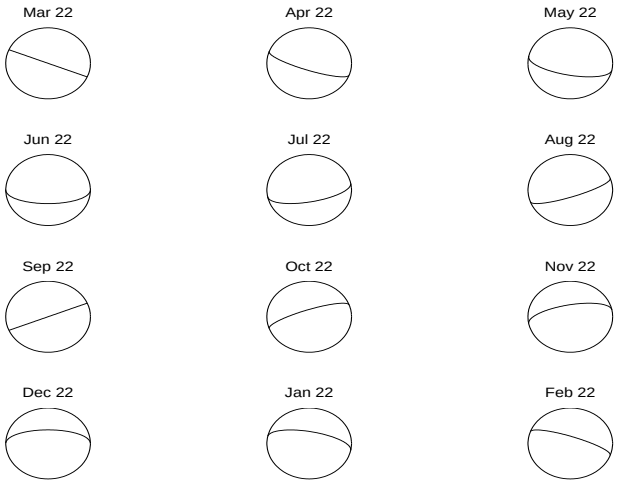
Test your implementations on the script `P3A` available on the website. It will display the orbit defined by $P = 10$, $A = 20$ and subdivide the area enclosed into 12 sectors of equal area. It will also run a simulation using `comet`. Submit a copy of the final figure that is produced and a listing of `OrbitSpline` and `OrbitVals`.

Part B (5 pts) A Computation that Supports the Second Law and Rejects Another

Despite the Earth's 23.5 degree axis tilt, the Northern and Southern hemispheres must receive equal amounts of solar energy over the course of a year. However, in light of Kepler's Second Law this is not obvious. Perihelion is in January, so when the Earth is closest to the Sun the Southern Hemisphere is enjoying its summer. The Southern hemisphere is presenting an "extra big face" to the Sun precisely when we are closer-than average to the Sun. In contrast, the Earth is furthest way from the Sun during its summer period. This suggests that the Southern hemispere might be capturing more than 50 percent of the total energy. On the other hand, Kepler's Second Law says that the Earth moves more quickly when it is closer to the Sun. This would tend to even out the energy balance.

But perhaps Kepler's Second Law is wrong. Maybe the Earth's radius vector rotates at a constant rate. Let's check out these alternative "theories" with a careful energy balance computation.

The figure below shows what the Earth looks like from the Sun at various times during the year. For any of these "snapshots", the fraction of solar energy received by the Northern Hemisphere equals the ratio of the area above the equator line to the total area of the disk. (Don't worry about the fact that the Earth is a spherical "target". Think about the fraction of photons that hit the earth above the equator.) This fraction is less than one-half during fall and winter and greater than one-half during spring and summer.



For the Earth, $A = 94.51$, $P = 91.41$, and $T = 365.25$. Assume that if $r(\theta)$ is the length of the radius vector when the polar angle is θ, then $E(\theta) = k/r(\theta)^2$ is the amount of solar energy that the Earth intercepts. This is just the usual inverse-square law. (Don't worry about the value of the constant k. You will find that it drops out of the computations below.) A recipe for $r(\theta)$ is given above. The amount of that energy received by the Northern Hemispere is given by

$$E_{North}(\theta) = E(\theta)\left(\frac{1+\sin(\theta-\theta_v)\sin(23.5\pi/180)}{2}\right).$$

Here, $\theta_v = 54.7\pi/180$ is the value of θ associated with the Vernal equinox (March 23 or so). The fraction in parenthesis can be derived by considering the "above equator" areas that you see in the figure.

Write a script `P3B` that uses the MATLAB integrator `Quad` to estimate the following ratios:

$$R_1 = \frac{\displaystyle\int_0^T E_{North}(S(t))dt}{\displaystyle\int_0^T E(S(t))dt}$$

and

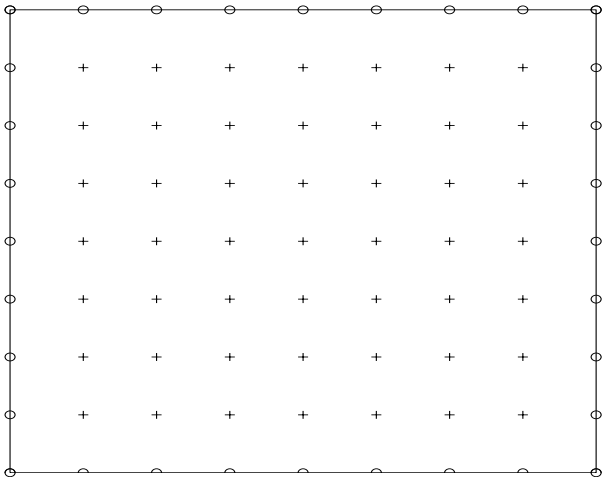
$$R_2 = \frac{\displaystyle\int_0^T E_{North}(2\pi t/T)dt}{\displaystyle\int_0^T E(2\pi t/T)dt}$$

Here, S is the spline produced `OrbitSpline(91.41,94.51,365.25,800)`. Thus, R_1 estimates the fraction of solar energy that is striking the Northern hemisphere during the course of a year assuming that the earth moves according to Kepler's Second Law. In contrast, R_2 estimates the same fraction under the assumption that polar angle rotates at a constant rate.

All integrals should be computed using `QUAD`. Your script should clearly report through ten decimal places the values of R_1 and R_2 that are obtained when the `QUAD` relative error tolernce is set to 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, and 10^{-8}. You will have to set up four calls to `QUAD` for each value of the tolerance. Submit output, a listing of `P3B`, and a listing of all the "integrand functions" that you needed to solve this problem. You will have to review how `QUAD` can integrate functions that have parameters.

Part C (7 pts) The Poisson Equation on a Square

Consider a square metalic plate with vertices at (0,0), (1,0), (1,1), and (0,1). Let $T(x,y)$ be the temperature at point (x,y) where $0 \leq x \leq 1$ and $0 \leq y \leq 1$. Assume that $T = 0$ along the left, right, and bottom edges but that $T(x,1) = 100\sin(\pi x)e^{-2x}$ for $0 \leq x \leq 1$. We wish to estimate the value of $T(x,y)$ for points (x,y) that satisfy $0 < x < 1$, $0 < y < 1$. In particular, for a given positive integer m we wish to estimate the temperature at (ih, jh) for all $1 \leq i \leq m$ and $1 \leq j \leq m$ where $h = 1/(m+1)$ The following figure depicts the situation when $m = 7$:



The points of known temperature are designated by 'o' and the points of unknown temperature by '+'. Thus, there are m^2 unknowns $t_1, \ldots, t_{m^2}$. We associate these unknowns with the '+' points in top-to-bottom, left-to-right order.

A reasonable model (whose details we suppress) is to assume that the temperature at a '+' point is the average of the temperatures at its four neighbors points. (The "north", "east", "south" and "west" neighbors.) Usually the four neighbors are '+' points. However, for a '+' point near the edge, one or two of the neighbors may be an 'o' point.

Let's look at the "averaging" equation at the 10th '+' point. This is the 3rd '+' point in the 2nd row (counting rows from the top). First, we figure out who the neighbors are:

- The north neighbor is the 3rd '+' point in the 1st row. (index = 3 = 10-7)
- The west neighbor is the 2nd '+' point in the 2nd row. (index = 9 = 10-1)
- The east neighbor is the 4th '+' point in the 2nd row. (index = 11 = 10+1)
- The south neighbor is the 3rd '+' point in the 3rd row. (index = 17 = 10+7)

Having done that, to say that the temperature at the 10th '+' point is the average of the four neighbor temperatures is to say that

$$-t_3 - t_9 + 4t_{10} - t_{11} - t_{17} = 0.$$

This is a linear equation in five unknowns. Equations associated with '+' points that are next to an edge are similar except that known edge temperatures are involved. For example, the equation at the 5th '+' point in the first row is given by

$$-t_4 + 4t_5 - t_6 - t_{12} = north_5,$$

where $north_5$ is the (known) temperature at the 'o' point located at $(5h, 1)$. This is a linear equation that involves four of the unknowns.

Thus, the vector of unknowns t solves an m^2-by-m^2 linear system of the form $At = b$. Note that for $i = 1{:}m^2$, $A(i,i) = 4$ and $A(i,:)$ has either two, three, or four -1's. Write a MATLAB function `t = Poisson(m)` that returns the solution to this system. Use the \ operator. Test your implementation by running the script `P3C` available on the website.