MAT 580 Programming Project

Diego G. A. Avalos Galvez

Fall 2017

Abstract

In this project, we look at the equations of position of the orbits of two planets and calculate the minimum distance between the planets at distinct times t_1 and t_2 using the method of steepest descent. We will look at various values of the maximum error ϵ implemented in steepest descent, and analyze how effective the method is with these values to conclude that it suffices to use $\epsilon = 0.01$ to obtain a good approximation of the minimum distance between the planets.

Introduction

Consider the orbit of planet 1, which is defined by the equation

$$\begin{bmatrix} x_1(t) \\ y_1(t) \end{bmatrix} = \begin{bmatrix} \cos(\varphi_1) & \sin(\varphi_1) \\ -\sin(\varphi_1) & \cos(\varphi_1) \end{bmatrix} \begin{bmatrix} \frac{P_1 - A_1}{2} + \frac{P_1 + A_1}{2}\cos(t) \\ \sqrt{P_1 A_1}\sin(t) \end{bmatrix}$$

where $A_1 = 10$, $P_1 = 2$, and $\varphi_1 = \pi/8$.

Also consider the orbit of planet 2, which is defined by the equation

$$\begin{bmatrix} x_2(t) \\ y_2(t) \end{bmatrix} = \begin{bmatrix} \cos(\varphi_2) & \sin(\varphi_2) \\ -\sin(\varphi_2) & \cos(\varphi_2) \end{bmatrix} \begin{bmatrix} \frac{P_2 - A_2}{2} + \frac{P_2 + A_2}{2}\cos(t) \\ \sqrt{P_2 A_2}\sin(t) \end{bmatrix}$$

where $A_2 = 4$, $P_2 = 1$, and $\varphi_2 = -\pi/7$.

In this project, we seek to minimize the the distance between the two planets over their time variables. In other words, we will minimize the function

$$dist(t_1, t_2) = \frac{1}{2}[(x_1(t_1) - x_2(t_2))^2 + (y_1(t_1) - y_2(t_2))^2]$$

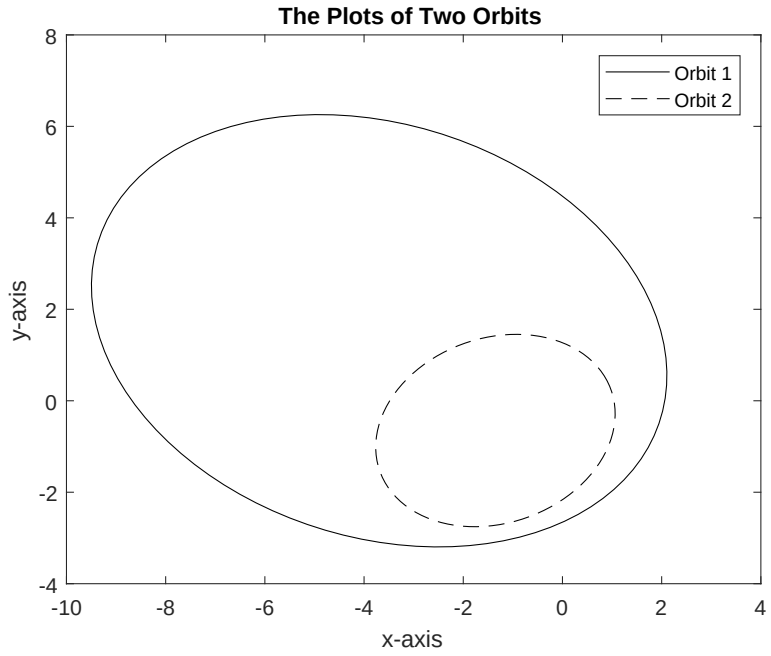
over t_1 and t_2 on the restricted domain $(t_1, t_2) \in D = [0, 2\pi] \times [0, 2\pi]$.

The method of steepest descent will take an initial fixed point $T_1 \in D \subset \mathbb{R}^2$ and a fixed maximum error number $\epsilon > 0$ such that if $||\nabla dist(T_1)|| < \epsilon$, then we accept T_1 as our value that minimizes $dist$ over D. Otherwise, we set $d_1 = -\nabla dist(T_1)$, and minimize $dist(T_1 + d_1\lambda)$ over $\lambda \in [0, \infty)$. If we let λ_1 be our minimizer, then set $T_2 = T_1 + \lambda_1 d_1$ and continue the process with our new fixed value T_2 and until we have k iterations such that $||\nabla dist(T_k)|| < \epsilon$.

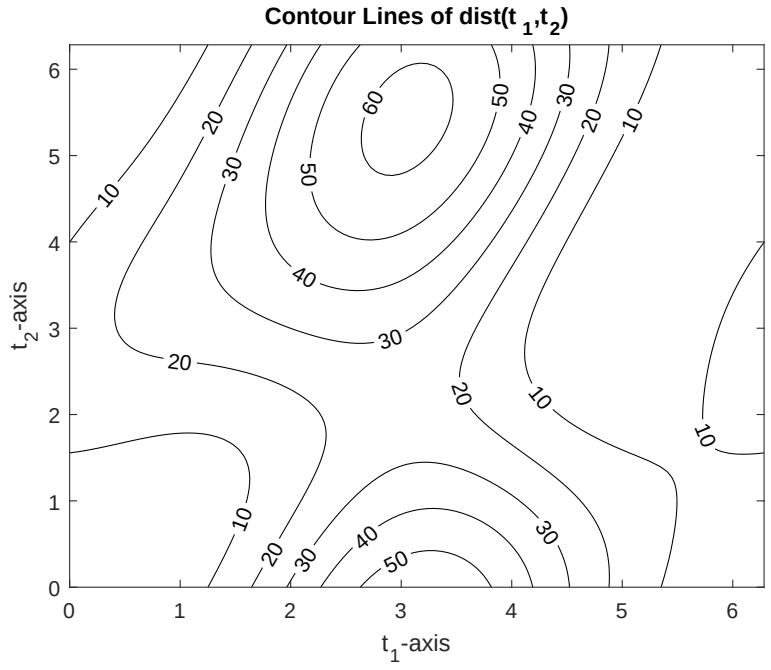
In this paper, we will analyze the method of steepest descent with the ϵ-values of 1, 0.5, 0.1, 0.01, and 0.001, and look at the number of iterations k and computation time to test the efficiency of our method and our code, which is implemented in Matlab.

Discussion and Results

The following figure generated in Matlab helps us to visualize the orbits of both planets, each on the time interval $t \in [0, 2\pi]$.



Now, we will use the contour plot of our dist function over $D = [0, 2\pi]^2$ in order to estimate a good starting point T_1 which we will implement in the steepest descent method.



It looks like a good starting place is in the region between two contour lines of value 10, namely, we will pick $T_1 = [5 \quad 3]^T$.

Let us see the output of our code when it is run with the epsilon values of 1, 0.5, 0.1, 0.01, and 0.001, respectively. The following italicized text are lines obtained when our code is run with the above mentioned ϵ-values.

With epsilon = 1.000, our t-values that minimize our dist function are t_1 = 5.14 and t_2 = 3.62. There were 8 iterations needed, and the computation time was 0.46 seconds. Our minimum distance is 0.74.

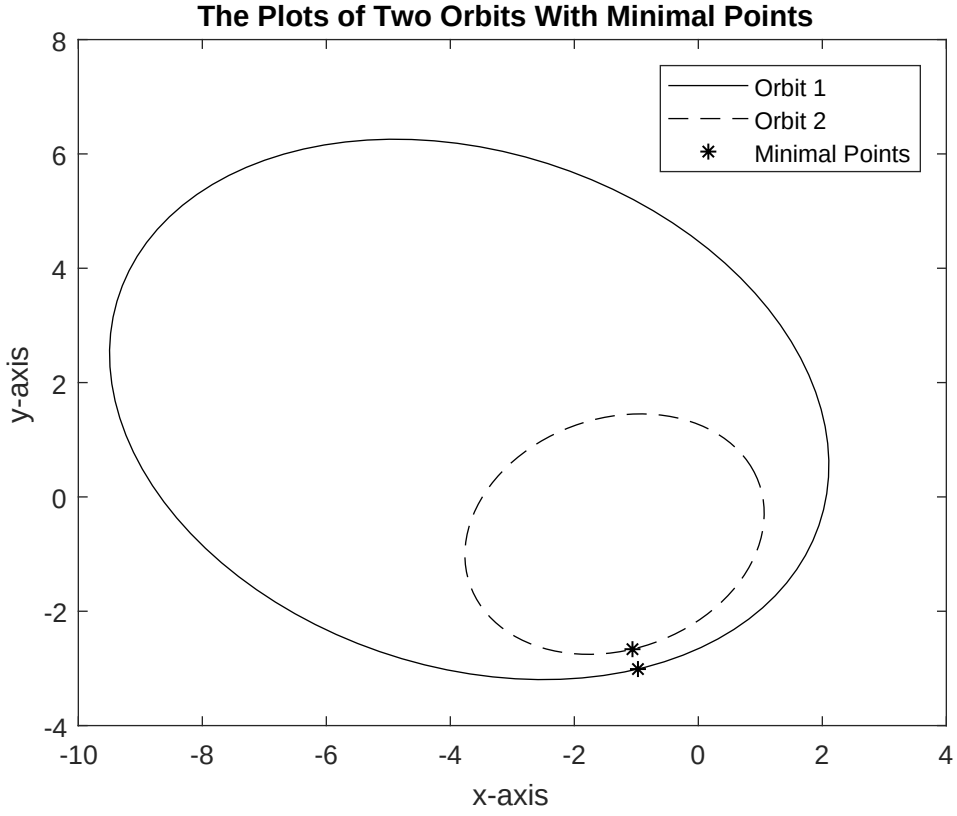
With epsilon = 0.500, our t-values that minimize our dist function are t_1 = 5.21 and t_2 = 3.85. There were 17 iterations needed, and the computation time was 0.99 seconds. Our minimum distance is 0.55.

With epsilon = 0.100, our t-values that minimize our dist function are t_1 = 5.39 and t_2 = 4.24. There were 65 iterations needed, and the computation time was 3.72 seconds. Our minimum distance is 0.38.

With epsilon = 0.010, our t-values that minimize our dist function are t_1 = 5.49 and t_2= 4.44. There were 199 iterations needed, and the computation time was 11.34 seconds. Our minimum distance is 0.36.

With epsilon = 0.001, our t-values that minimize our dist function are t_1 = 5.50 and t_2= 4.46. There were 351 iterations needed, and the computation time was 19.78 seconds. Our minimum distance is 0.36.

For instance, our results indicate that the t values that minimize the *dist* function are t_1 = 5.50 and t_2= 4.46 (even though $\epsilon = 0.01$ and $\epsilon = 0.001$ give us the same estimated minimum distance, we use the later because we trust our code gives us a better approximation of the t-values for the smaller espilon value). We check this by looking at the following plot of the two orbits, but this time we use stars to indicate the positions of our minimizing t-values.



From the figure above, we conclude that our results estimate the minimal distance between the planets very well. Observe that our actual minimum distance is 0.36 and it is computed by plugging in our t_1 and t_2 in our actual distance function

$$ActualDistance(t_1, t_2) = \sqrt{2 * dist(t_1, t_2)}.$$

Furthermore, our results show that, if all we care about is to find the minimum distance between the planets, it suffices to use $\epsilon = 0.01$.

In my opinion, the biggest advantage of my code is that I did not have to compute the gradient or any derivatives of *dist* because I used Matlab's Symbolic Toolbox to compute them symbolically. Please read the heavily commented source code in the next page to see how all the computations were carried out in this paper.

Source Code

```
%                       MAT 580 Project Source Code
%                                Diego Avalos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part 1
% In this part, we plot the two orbits X1 and X2 of the two planets.
% All parameters or constants labeled with j (j=1,2) are related to the
% orbit of planet j.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We first define our time points in our time vector.
time = linspace(0,2*pi);
% Next, we define some useful constants which define the shapes of the
% orbits.
A1 = 10; P1 = 2; theta1 = pi/8;
A2 = 4; P2 = 1; theta2 = -pi/7;
% We will use the following symbolic functions which will simplify the
% code:
% R(z) is a 2x2 rotation matrix,
% v(P,A,t) is a 2x1 vector.
syms z P A t
R(z) = [cos(z), sin(z); -sin(z), cos(z)];
v(P,A,t) = [0.5*(P-A)+0.5*(P+A)*cos(t); sqrt(P*A)*sin(t)];
% We now compute the orbits X1 and X2
X1 = R(theta1)*v(P1,A1,time); X2 = R(theta2)*v(P2,A2,time);
% We extract the inputs from X1 and X2 which represent the x and y
% positions of each planet.
x1 = X1(1,:); y1 = X1(2,:);
x2 = X2(1,:); y2 = X2(2,:);
% To conclude part 1, we plot both orbits on one graph.
figure(1)
plot(x1,y1,'k',x2,y2,'--k');
xlabel('x-axis'); ylabel('y-axis');
title('The Plots of Two Orbits');
legend('Orbit 1', 'Orbit 2');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part  2
% In this part we plot contour lines of our dist(t1,t2) function.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We first define the domain of our distance function that we seek to
% minimize. The domain is [0,2pi]^2.
[t1, t2] = meshgrid(time,time);
% Now we define our dist function in terms of t1 and t2. We use the
% functions x11, x22, y11, and y22 to simplify code.
x11 = cos(theta1)*(0.5*(P1-A1)+0.5*(P1+A1)*cos(t1))+...
    sin(theta1)*sqrt(P1*A1)*sin(t1);
x22 = cos(theta2)*(0.5*(P2-A2)+0.5*(P2+A2)*cos(t2))+...
    sin(theta2)*sqrt(P2*A2)*sin(t2);
y11 = -sin(theta1)*(0.5*(P1-A1)+0.5*(P1+A1)*cos(t1))+...
    cos(theta1)*sqrt(P1*A1)*sin(t1);
y22 = -sin(theta2)*(0.5*(P2-A2)+0.5*(P2+A2)*cos(t2))+...
    cos(theta2)*sqrt(P2*A2)*sin(t2);
dist = 0.5*( (x11-x22).^2+(y11-y22).^2 );
% To end part 2, we plot contour lines of dist(t1,t2).
figure(2)
contour(t1,t2,dist,'k','ShowText','on');
xlabel('t_1-axis'); ylabel('t_2-axis');
```

```
title('Contour Lines of dist(t_1,t_2)');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Part 3
% In this last part, we will minimize our dist function over its specified
% domain [0,2*pi]^2 using the steepest descent method.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% We first symbolically redefine our dist function (this time we call it
% 'Dist') in terms of the shorter functions X11, X22, Y11, Y22.
syms T1 T2
X11 = cos(theta1)*(0.5*(P1-A1)+0.5*(P1+A1)*cos(T1))+...
    sin(theta1)*sqrt(P1*A1)*sin(T1);
X22 = cos(theta2)*(0.5*(P2-A2)+0.5*(P2+A2)*cos(T2))+...
    sin(theta2)*sqrt(P2*A2)*sin(T2);
Y11 = -sin(theta1)*(0.5*(P1-A1)+0.5*(P1+A1)*cos(T1))+...
    cos(theta1)*sqrt(P1*A1)*sin(T1);
Y22 = -sin(theta2)*(0.5*(P2-A2)+0.5*(P2+A2)*cos(T2))+...
    cos(theta2)*sqrt(P2*A2)*sin(T2);
% Hence, the function we seek to minimize is the following
Dist(T1,T2) = 0.5*( (X11-X22).^2+(Y11-Y22).^2 );
% We now let Matlab compute the gradient of Dist symbolically.
GradDist(T1,T2) = [diff(Dist,T1); diff(Dist,T2)];
% This is the part where we implement the steepest descent method.
% Here we tweak our epsilon value. Some values that I will look closely are
% epsilon = 1, 0.5, 0.1, etc.
epsilon = 0.01;
% Now, from our contour plot in part 2, it seems that the following
% t-values are a good place to start our steepest descent.
minT1 = 5; minT2 = 3;
% We wish to count the number of iterations in our process and also monitor
% the computation time. The computation time may seem long, and it could be
% because I am making Matlab do a bunch of symbolic computations.
iterations=0;
tic;
% Let the steepest descent method begin! Observe that I am using the double
% function to make Matlab treat some values as doubles rather than symbolic
% values.
while norm(double(GradDist(minT1,minT2))) >= epsilon
% d is the negative of the gradient at the current point
    d = -double(GradDist(minT1,minT2));
% We seek to minimize the single-variable function of lambda
% f(minT+lambda*d) over lambda >= 0. We first define the function Dist_L
% symbolically.
    syms L
    Dist_L = Dist(minT1+L*d(1), minT2+L*d(2));
% In order to use Matlab's fminsearch function we need to convert our above
% function into a function handle.
    Dist_L = matlabFunction(Dist_L);
% We trust fminsearch to find our minimum lambda Lmin.
    Lmin = fminsearch(Dist_L,0);
% Update our point and iteration number.
    minT1 = minT1 + Lmin*d(1); minT2 = minT2 + Lmin*d(2);
    iterations = iterations+1;
end
% Keep track of the computation time.
TimeElapsed=toc;
% Our actual minimum distance is
minDist = double(sqrt(2*Dist(minT1,minT2)));
% We store useful information in the info vector
info = [epsilon, minT1, minT2, iterations, TimeElapsed, minDist];
```

```
% and state my conclusions as follows
fprintf(['With epsilon = %1.3f, our t-values that minimize our dist ',...
    'function are t_1 = %2.2f and t_2 = %2.2f. There were %d ',...
    'iterations needed, and the computation time was %3.2f seconds.'...
    ' Our minimum distance is %2.2f.\n'], info);
% We are done with part 3. However, I want to add an additional graph which
% includes the orbits of both planets and marks the locations of the two
% planets where they are the closest (the points are those computed using
% steepest descent). This will allows us to visualize how well steepest
% descent works with different epsilon values.
% Here are the positions of both orbits at minimal distance
XMin1 = double(R(theta1)*v(P1,A1,minT1));
XMin2 = double(R(theta2)*v(P2,A2,minT2));
% And here's our handsome plot!
figure(3)
plot(x1,y1,'k',x2,y2,'--k',[XMin1(1) XMin2(1)],[XMin1(2) XMin2(2)],'k*');
xlabel('x-axis'); ylabel('y-axis');
title('The Plots of Two Orbits With Minimal Points');
legend('Orbit 1', 'Orbit 2','Minimal Points');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                    Fin
```

The source code above may be downloaded from

https://github.com/mathydiego/mat580project/blob/master/mat580project.m