Midterm 1: Computational Physics

Pablo Lopez Duque

October 23^{rd}, 2020

1 Problem 1

True or False: When numerically solving a differential equation, it is always better to choose a smaller Δt. Explain why or why not.

This statement is **False**. Computers store float point variables with a limited precision due to memory size limitations. This means that decreasing the time step Δt will first reduce the round-off error.If we keep decreasing Δt, eventually, the round-off error will reach a minimum value and then start increasing again. The easiest way to show this is by using a counterexample. By convenience we can handpick problem 1 in HW1-2 as our counterexample: Consider the logistic equation for $a = b = 1$, and with $x_0 = 0.1$. Its exact solution is $x(t) = 1/(1 + 9e^{-t})$. If we compute $\delta(\Delta t) = |x'(t) - \frac{x(t+\Delta t)-x(t)}{|\Delta t}|$ for $t = 2$ and for different sizes of Δt we can see that the error first decreases until a certain value and then start increasing again. This trend was language independent for Matlab, Python and C++, all showing similar behavior. Minor differences are likely system dependent. Figure 1 depicts this behavior.

$\therefore$ **It is NOT always better to choose a smaller Δt when numerically solving differential equations.**

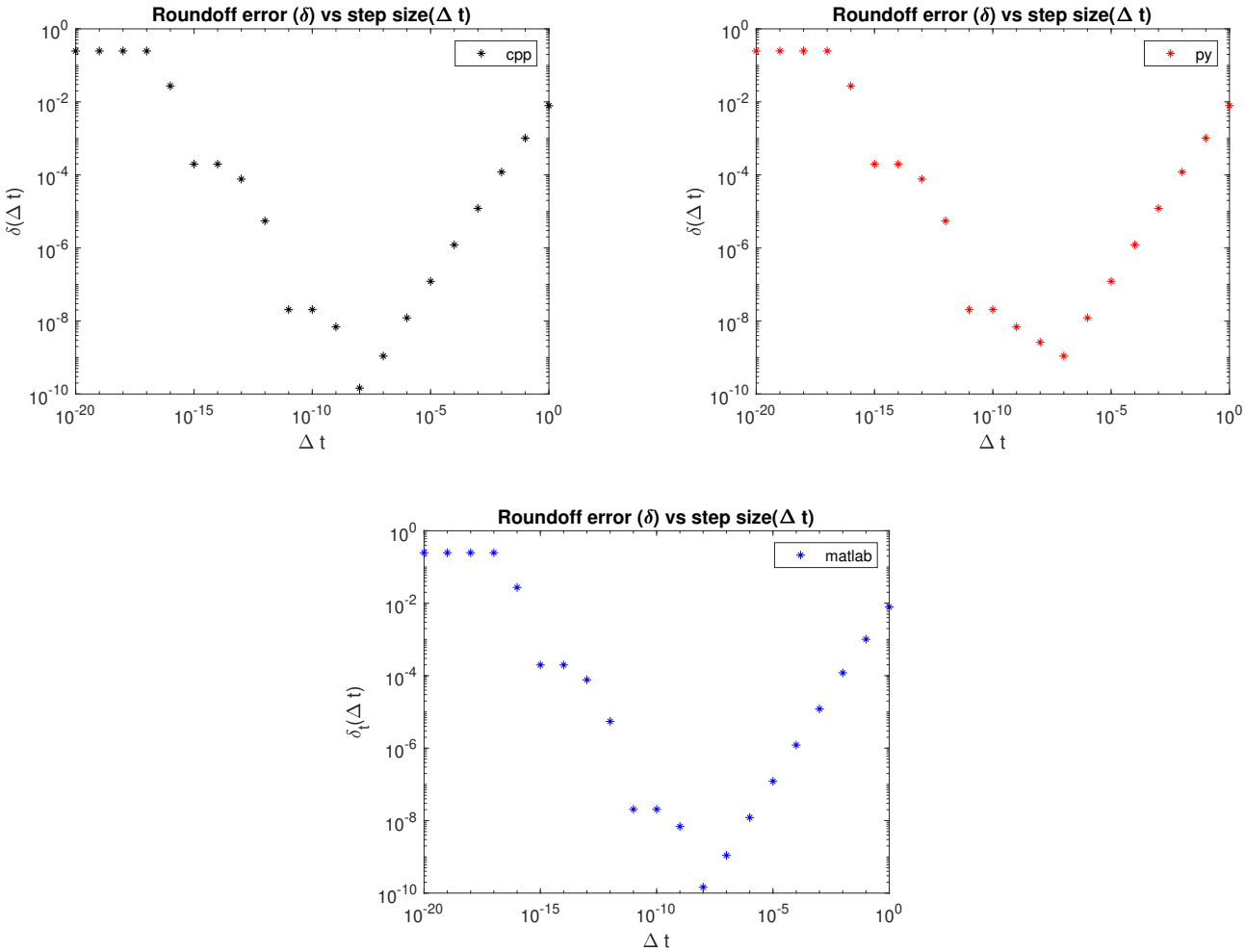

Figure 1: Absolute error vs step size(Δt). The minimum error is attained for a step size of 10^{-8}, but then increases due to round off effects.

2 Problem 2

Short answer: Describe the difference between a non-adaptive and adaptive numerical method. Under what conditions are adaptive methods generally useful. Are there any disadvantages to using an adaptive method?

A non-adaptive numerical method uses a fixed time step, or more generally, a fixed step for the independent variable, whereas an adaptive method actively tests and chooses a "right" step after each iteration (right defined in terms of some algorithm). Adaptive methods are most useful when we have stiff equations where there are abrupt changes in the function. They might be also useful whenever we are dealing with phase transitions.

The main disadvantage of adaptive methods is the extended computation time required with respect to non-adaptive ones. Depending on how the implementation of the step-size comparison and selection is made, this difference can be very significant or not so big.

3 Problem 3

This problem relates to the Discrete Fourier Transform. You are welcome to use the Fast Fourier Transform if you wish, but the data set is small enough there will be no appreciable difference between the two algorithms.

(a) **Import the data in data.csv, xi, Fourier transform it, and plot the resulting power spectrum.**

Importing the data to Matlab is extremely easy, one can just use the following command:

```
xdata=load('data.csv'); %load data from file 'data.csv'
```

Then, in order to take the Fourier Transform, we must define a time interval, a sampling rate that matches the size of our imported data and call the fft function that performs a Discrete Fourier Transform using a fast Fourier algorithm:

```
T=1;  %sample time
dt=1/128;  %sampling rate
tvals=(1:(T/dt))*dt;  %Creates a row vector with values from 0 to T with 1/dt ...
    divisions

transx=fft(xdata); %FFT of xdata
magx=abs(transx); %take the norm of the FFT of xdata
```

The data and the Fourier transform are shown in figure 2. It is clear that there are many frequencies contributing to the signal corresponding to the input data. From the amplitude of the signal at each contributing frequency, we can see that all contribute equally to the signal.

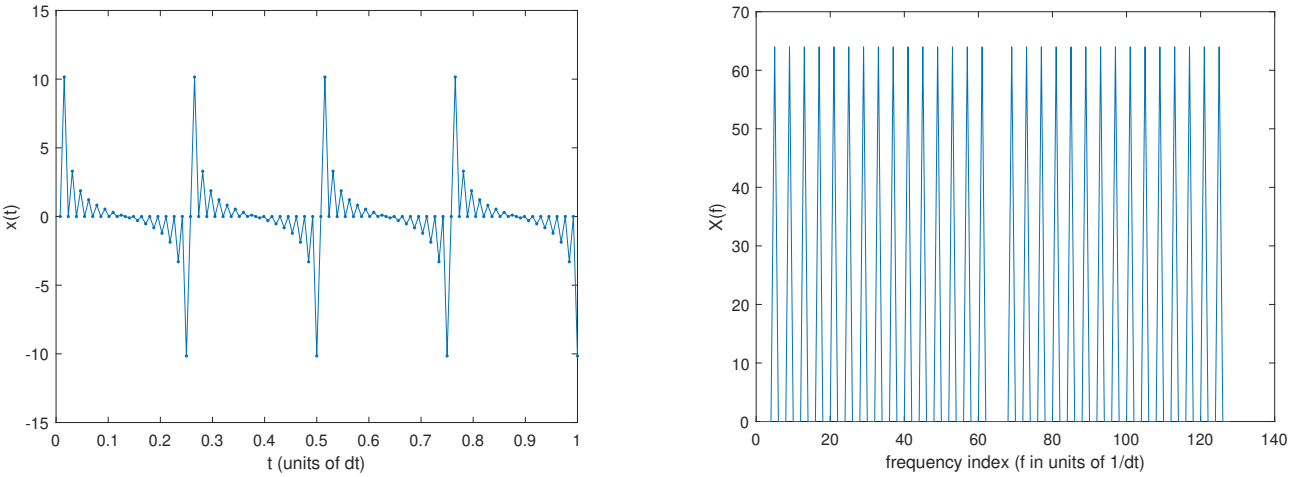

Figure 2: Raw signal(left) and its Fourier transform(right).

(b) **Define the sequence yi, with $y_i = (x_i + x_{i+1})/2$ and $y_0 = y_N$ (periodic boundary conditions). Fourier transform this data, and comment on similarities or differences with part (a). This averaging procedure is an example of a low-pass filter. Explain why that name is appropriate.**

In order to implement the given sequence yi, we can use a for loop, as follows:

```
for i=1:length(xdata)-1 %until N-1 due to the size limit in xdata
    yfiltrd(i)=(xdata(i)+xdata(i+1))/2;
```

```
3  end
4  yfiltrd(length(xdata))=yfiltrd(1); %periodic boundary conditions(BC)
```

In figure 3 the processed data and its Fourier transform are shown. It is clear that the signal looks smoother, but there are still some minor rough regions due to the fact that consecutive points have the same value for y(t). The Fourier transform shows the spectrum changed drastically. The behavior until the relative frequency index 60 is expected, with low frequencies passing almost unaltered while higher frequencies getting a considerable damping. However, after this value, we observe a symmetric reflection of the spectrum below frequency index of 60. We will explore this behavior below.

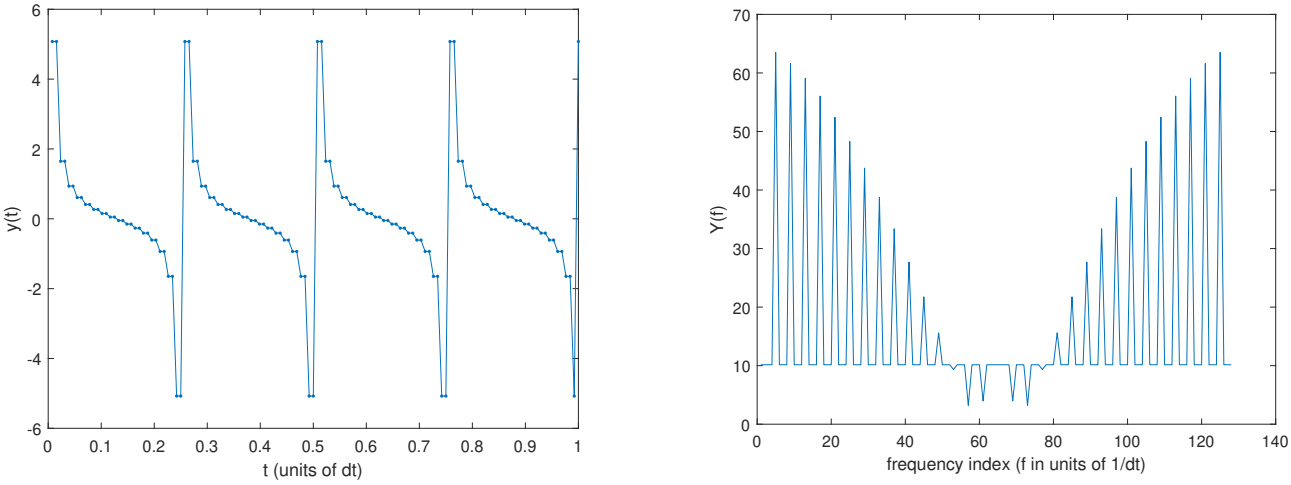

Figure 3: Processed signal(left) and its Fourier transform(right).

In order to understand what the averaging process is achieving, one can gain insight by plotting the raw and processed data together, as shown in figure 4. The averaging process reduces the high variation between consecutive data points by taking the average between them. This means that peaks in the raw signal will become smaller whereas valleys will become bigger, thus, reducing the variation in the signal. This means that we expect a reduction in the high frequencies contribution to the signal, since the rapid variations got smoothed out by the averaging process.

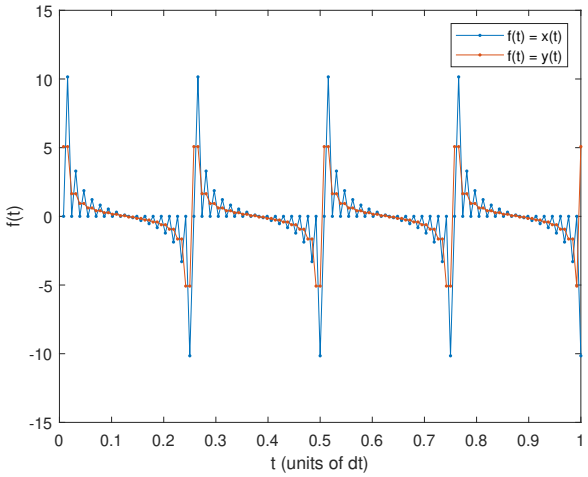

Figure 4: Raw and processed signal. The averaging process reduces the high variation between consecutive data points by taking the average between them.

Now, going back to the rightmost plot in figure 3, the Fourier transformed data show almost no decrease for low frequencies and start damping the amplitude for higher frequencies. This is exactly the low-pass filter expected behavior we wanted to observe. Recall that low-pass filters let low frequency signals pass undamped whereas they block high frequency ones. However, after some point, the values increase again. This is due to the symmetric behavior of the Fourier transform whenever a fully real signal is used and can be safely ignored for this analysis.

4 Problem 4

The logarithm of an $n \times n$ matrix $\mathbf{A}$ can be determined through a Taylor expansion, where I is the identity matrix:

$$\log(\mathbf{A}) = -\sum_{k=1}^{\infty} \frac{(\mathbf{I}-\mathbf{A})^k}{k}$$

(a) **What constraints are there on the matrix A for this sum to converge?**

$$\log(\mathbf{A}) = -\sum_{k=1}^{\infty} \frac{(\mathbf{I}-\mathbf{A})^k}{k} = \sum_{k=1}^{\infty}(-1)^{k+1}\frac{(\mathbf{A}-\mathbf{I})^k}{k}$$

This is an alternating series and converges when $|A-I| < 1$, which is explained in the following paragraph. If we recall, an alternating series $\sum_{k=1}^{\infty}(-1)^{k+1}a_n$, where a_n is positive or negative for all n, converges whenever the n^{th} term of the sum (a_n) is monotonically decreasing and $lim_{k\to\infty}a_n = 0$. In this particular case, to have convergence, we must impose the condition that the numerator is < 1 since x^n grows faster than n:

$$lim_{k\to\infty}\frac{x^n}{n} = lim_{k\to\infty}\frac{nx^{n-1}}{n} = lim_{k\to\infty}x^{n-1}$$

which converges only when $|x| < 1$.

$$\therefore |A-I| < 1$$

is the convergence condition. **This basically means that the matrix A should be sufficiently close to the identity matrix in order for the Taylor series to converge.**

(b) **What is the computational complexity of computing** $\log(\mathbf{A})$ **for large** n **using the Taylor expansion?**

Let's first analyze the kth term of the sum. The complexity of matrix multiplication is $O(n^3)$, since we are performing $k-1$ multiplications in the numerator, its computational complexity is $n^3(k-1)$. The complexity of division between two m-digit numbers is m^2, a factor of m for each digit. k has from 1 to 3 digits at most for almost all practical n's whereas we can assume a tolerance of 10^{-8} for this computation, which means that the division would cost around 8×3 operations at most for the k^{th} term, which is negligible compared to the matrix power for large n. The minus one exponentiation and multiplication also become negligible. We must finally account for the sum of k terms. Since we know that the highest order contributions would come from the matrix exponentiation for large n, to compute a fair estimate of the complexity we can safely neglect the other operations. So, we have a sum: $n^3+2n^3+3n^3+...+(k_{max}-1)n^3 = n^3(1+2+3+...+k_{max}-1) = n^3(k_{max}-1)(k_{max}-2)/2 \approx n^3k_{max}^2$. Hence, the computational complexity is $O(n^3k_{max}^2)$. This for n large and $n^3 >> k_{max}^2$ basically means that it scales as $O(n^3)$. Hence, as long as we don't let our iterator run very large, the scale will be dominated by n^3.

(c) **Write programs in C++, Matlab, and python (all three languages) implementing eq. (1) to compute** $\log(\mathbf{A})$**. You may not use predefined methods for computing the matrix log. Run this program on the matrix** $A_{ij} = (1+\delta_{ij})1/(n+1)$ **for** $n = 10$ **and print the result to a file.**

In this report, we will only include the discussion for a single language (Matlab), since the implementation is similar for all of them. First, we need to define the parameters and storage variables for our program:

```
clear;
tic %starts the time counter
n=200; %select matrix size
%nlist=[10,20,50,100,200,500,1000,2000]
d=1/(n+1);
Idm=eye(n);
```

```
A=d*ones(n)+d*eye(n);
check=logm(A); %to check if our program computes the same as built-in fun
tol=1e-8;%tolerance for our expansion, closer to 0 means better approx
lnm=0;%storing matrix result for logm(matrix)
lnm_old=0;
k=1;%iterator
```

Then, we will iterate over until we have reached the desired convergence condition through a tolerance. In this case, I choose to use a scaled Frobenius norm to check the convergence limit, so that the maximum difference between the entries of the matrices computed by our program and the built-in function are always within our tolerance.

```
while(1)
    lnm=lnm_old+(-1)^(k+1)*((A-Idm)^k)/k;
    if norm(n*(lnm-lnm_old),'fro')<tol
        break;
    end
    lnm_old=lnm;
    k=k+1;
end
toc %ends the time counter and prints time
```

Finally, we check that our desired result is within the given tolerance:

```
%check
if max(max(abs(lnm-check)))<tol %check how to estimate truncation error.
    fprintf("true\n")
else
    fprintf("false\n")
end
k %prints number of iterations

max(max(abs(lnm-check))) %prints max difference
writematrix(lnm,'MT3matrix_n10Matlab.csv','Delimiter',' ')
writematrix(lnm,'MT3Checkmatrix_n10Matlab.csv','Delimiter',' ')
```

The output file of the program is $MT3matrix\_n10Matlab.csv$. It is not included on this report due to space limitations and formatting. However, it is both included in the GitHub and Teams submission.

Note for C++ compilation:
For the cpp program, the Eigen library was used. Hence, in order to compile the program one has to make sure that the compiler includes the Eigen folder I am including on the include folder. Basically the compiler needs the ability to use files in the folders Midterm/eigen. Also, one must make sure that the folder Midterm/eigen/unsupported is included as a header for the cpp program.

(d) **Run your programs in C++, Matlab, and python to determine the time it takes (in seconds) to determine log(A) for $A_{ij} = 1 + \delta_{ij}$ with $n = 10, 20, 50, 100, 200, 500, 1000, 2000$. Plot the time as a function of n on log-log axes all in the same figure (that is, do not submit 3 separate graphs). Also plot your predicted complexity from (b) in the same figure.**

The output times are summarized in table 1. As we can see, Matlab and Python perform similarly, whereas C++ turns out very slow for larger matrices. The most probable cause is the usage of the Eigen class MatrixXd. The reason for this choice was that it allows an easy implementation of matrix objects into C++, especially for the kth power of a Matrix by the simple call: $(matrix).pow(k)$. To test this hypothesis one would need to implement the matrix multiplication by hand and compare performances. My expectation was that the Eigen implementation would perform better that a loop based program. However, the results show clearly that the performance is extremely poor for large matrices.

Table 1: Execution times for different matrix sizes and different languages

	matrix size	10	20	50	100	200	500	1000	2000
language									
Python	iterations	35	58	111	175	266	427	567	704
	time(s)	≈ 0	0.0156	0.0219	0.334	1.63	41.6	388	3060
Matlab	iterations	35	58	111	175	266	427	567	704
	time(s)	0.0919	0.0370	0.0423	0.119	1.22	36.7	358	3400
C++	iterations	65	133	349	726	1511			
	time(s)	0.0221	0.193	7.93	119	2340			

The running times for different languages and different matrix sizes are shown in figure 5. The prediction for the computational complexity discussed in part (b) was scaled by the number of processes per second of the CPU where the programs were compiled and run, a 2.2 GHz processor.

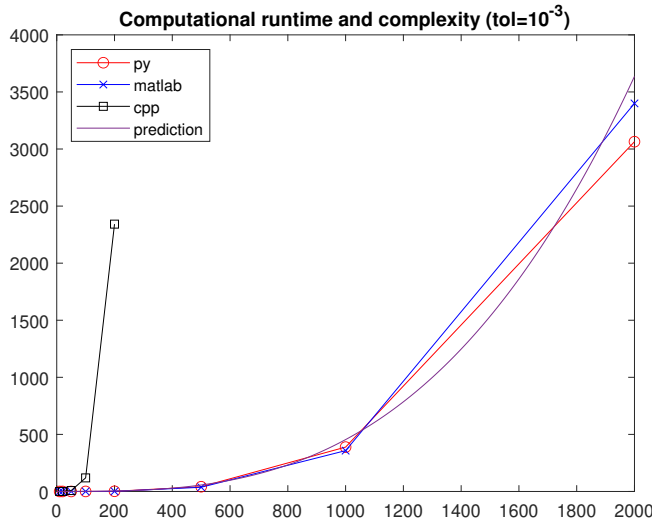

Figure 5: Computational runtimes for different programming languages and different matrix sizes showing how the time needed increases polynomially. However, C++ shows a different behavior possibly due to the use of the Eigen libraries. The prediction from b is plotted after scaling it with the number of processes of a 2.2Ghz CPU, which was used to compile and run the programs.

5 Problem 5

A system of 3 stars of mass $M = 1$ solar mass located at locations $\vec{R_1}$, $\vec{R_2}$ and $\vec{R_3}$ have a potential energy at any point $\vec{R}$:

$$U(\vec{R}, \vec{R_1}, \vec{R_2}, \vec{R_3}) = -GM^2 \left(\frac{1}{|\vec{R} - \vec{R_1}|} + \frac{1}{|\vec{R} - \vec{R_2}|} + \frac{1}{|\vec{R} - \vec{R_3}|} \right)$$

where G is the gravitational constant. Note that this three body problem has no known analytical solution for arbitrary initial conditions, but some initial conditions will give rise to periodic behavior (as you will show below). In this problem, you may use any programming language you like.

(a) **Show that if we choose an arbitrary length scale, the system can be non-dimensionalized in the form:**

$$\ddot{\vec{r_1}} = -\frac{\vec{r_1} - \vec{r_2}}{|\vec{r_1} - \vec{r_2}|^3} - \frac{\vec{r_1} - \vec{r_3}}{|\vec{r_1} - \vec{r_3}|^3} \qquad \ddot{\vec{r_2}} = -\frac{\vec{r_2} - \vec{r_1}}{|\vec{r_2} - \vec{r_1}|^3} - \frac{\vec{r_2} - \vec{r_3}}{|\vec{r_2} - \vec{r_3}|^3} \qquad \ddot{\vec{r_3}} = -\frac{\vec{r_3} - \vec{r_1}}{|\vec{r_3} - \vec{r_1}|^3} - \frac{\vec{r_3} - \vec{r_2}}{|\vec{r_3} - \vec{r_2}|^3}$$

What is a velocity of 1km/s in your units?

In order to find the differential equation for this problem, one must recall that:

$$\vec{F} = m\vec{a} = m\ddot{\vec{r}} = -\vec{\nabla} U(\vec{r}) \tag{1}$$

which applied to this system, yields:

$$\cancel{M}\ddot{\vec{R}} = -\left[-GM\cancel{^2}\vec{\nabla}_R \left(\frac{1}{|\vec{R} - \vec{R_1}|} + \frac{1}{|\vec{R} - \vec{R_2}|} + \frac{1}{|\vec{R} - \vec{R_3}|} \right) \right]$$

To recast this as a dimensionless equation, we can use the following change of variable: $\vec{r} = \frac{\vec{R}}{\rho} \Rightarrow d\vec{R} = \rho d\vec{r}$. Hence:

$$\rho\ddot{\vec{r}} = GM\frac{1}{\rho}\vec{\nabla}_r\frac{1}{\rho} \left(\frac{1}{|\vec{r} - \vec{r_1}|} + \frac{1}{|\vec{r} - \vec{r_2}|} + \frac{1}{|\vec{r} - \vec{r_3}|} \right)$$

where the last $1/\rho$ factor comes from the positions $\vec{R_i} = \rho\vec{r_i}$. Thus:

$$\ddot{\vec{r}} = \frac{GM}{\rho^3}\vec{\nabla}_r \left(\frac{1}{|\vec{r} - \vec{r_1}|} + \frac{1}{|\vec{r} - \vec{r_2}|} + \frac{1}{|\vec{r} - \vec{r_3}|} \right)$$

So, in order to have a dimensionless equation, $\rho = GM^{1/3}$. In order to further simplify this equation, we must resort to the following formula:

$$\vec{\nabla}\frac{1}{|\vec{r} - \vec{r_i}|} = -\frac{\vec{r} - \vec{r_i}}{|\vec{r} - \vec{r_i}|^3}$$

Therefore:

$$\ddot{\vec{r}} = \left(-\frac{\vec{r} - \vec{r_1}}{|\vec{r} - \vec{r_1}|^3} - \frac{\vec{r} - \vec{r_2}}{|\vec{r} - \vec{r_2}|^3} - \frac{\vec{r} - \vec{r_3}}{|\vec{r} - \vec{r_3}|^3} \right) \tag{2}$$

At this point, we have the equation of motion for any $\vec{r} \neq \vec{r_i}$. If we wanted to evaluate the equation of motion at a location of precisely one of the masses, we must discard the term corresponding to self interaction. This follows since there is no potential energy due to a mass at the location of that specific mass.

$$\therefore \ddot{\vec{r_1}} = -\frac{\vec{r_1} - \vec{r_2}}{|\vec{r_1} - \vec{r_2}|^3} - \frac{\vec{r_1} - \vec{r_3}}{|\vec{r_1} - \vec{r_3}|^3} \qquad \therefore \ddot{\vec{r_2}} = -\frac{\vec{r_2} - \vec{r_1}}{|\vec{r_2} - \vec{r_1}|^3} - \frac{\vec{r_2} - \vec{r_3}}{|\vec{r_2} - \vec{r_3}|^3} \qquad \therefore \ddot{\vec{r_3}} = -\frac{\vec{r_3} - \vec{r_1}}{|\vec{r_3} - \vec{r_1}|^3} - \frac{\vec{r_3} - \vec{r_2}}{|\vec{r_3} - \vec{r_2}|^3}$$

The velocity in this system is $\vec{v} = \frac{d\vec{R}}{dt}$, and the adimensional velocity is $\vec{\nu} = \frac{d\vec{r}}{dt}$. Hence:

$$\vec{v} = \frac{d\vec{R}}{dt} = \rho \frac{d\vec{r}}{dt} = (GM)^{1/3}\vec{\nu}$$

And the relationship between speeds is:

$$v = (GM)^{1/3}\nu$$

If we set $v = 1\ km/s = 10^3 m/s \Rightarrow \nu = \frac{1000 m/s}{(GM)^{1/3}} = 0.000196029$.

(b) **Analytically show that if the first star is initially stationary at the origin, and if the second and third stars satisfy $\vec{r_2} = -\vec{r_3}$ and with velocities $v_2 = -v_3$, that the first star will remain stationary for all times. Is this solution stable if r_1 is perturbed?**

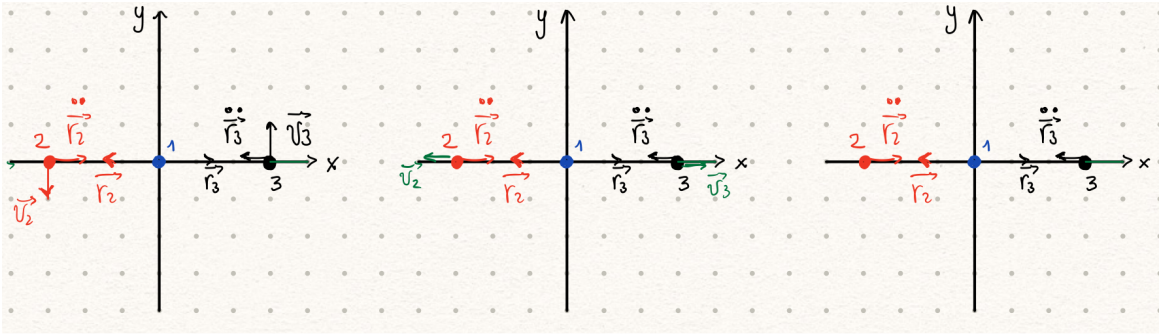

Figure 6: Left:Initial conditions that lead to periodic behavior with star 1 static at the origin. Center: Initial conditions which can lead to divergence of stars 2 and 3 towards infinity or collapse to the origin depending on the value of the velocities. Right: Initial condition that lead to a collapse of stars 2 and 3 at the origin.

In order to have a better understanding consider the diagram in figure 6. WLOG we can take all three stars collinear to the x-axis. The first star initially at the origin means $\vec{r_1}(t = 0) = \vec{0}$. If we choose $\vec{r_2} = -\vec{r_3}$ and evaluate every variable at $t = 0$, we have:

$$\ddot{\vec{r_1}} = -\frac{-\vec{r_2}}{|\vec{r_2}|^3} - \frac{\vec{r_2}}{|\vec{r_2}|^3} = \vec{0} \Rightarrow \cdot\vec{r_1} = constant = \dot{\vec{r_1}}(t = 0) = 0$$

$$\Rightarrow \vec{r_1} = constant = \vec{0}$$

Similarly:

$$\ddot{\vec{r_2}} = -\frac{\vec{r_2}}{|\vec{r_2}|^3} - \frac{\vec{r_2} + \vec{r_2}}{|\vec{r_2} + \vec{r_2}|^3} = -\frac{\vec{r_2}}{|\vec{r_2}|^3} - \frac{1}{4}\frac{\vec{r_2}}{|\vec{r_2}|^3} = -\frac{5}{4}\frac{\vec{r_2}}{|\vec{r_2}|^3}$$

and:

$$\ddot{\vec{r_3}} = +\frac{\vec{r_2}}{|\vec{r_2}|^3} - \frac{-\vec{r_2} - \vec{r_2}}{|-\vec{r_2} - \vec{r_2}|^3} = \frac{5}{4}\frac{\vec{r_2}}{|\vec{r_2}|^3}$$

$$\Rightarrow \ddot{\vec{r_3}} = -\ddot{\vec{r_2}}$$

This means that both accelerations for 2 and 3: $\ddot{\vec{r_3}}$ and $\ddot{\vec{r_2}}$, respectively, should point in anti-parallel directions and have the same magnitude. Hence, the force between the two will also be colinear. Thus, they will always be colinear unless they both collapse to the origin or diverge towards infinity. In any case, no matter what motion both stars 2 and 3 have, they will always stay equidistant to star 1. Hence, the latter will always stay stationary at the origin. This is true for any initial conditions that fulfill the given criteria.

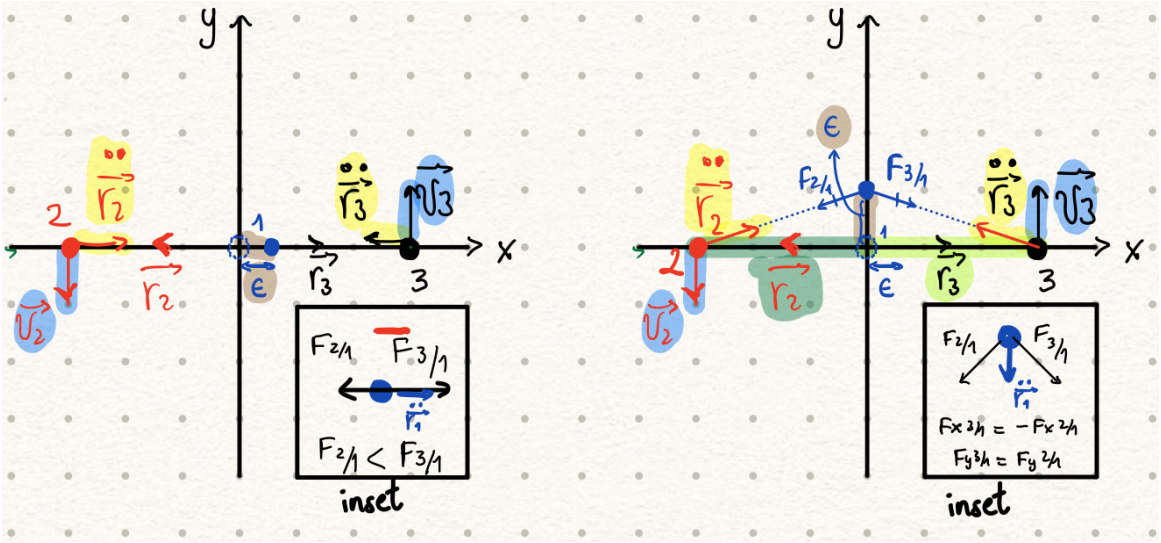

Figure 7: Left:Perturbation on star 1 initial position in the x direction. Right: Perturbation on star 1 initial position in the y direction.

Now, let's consider the stability of this solution if $\vec{r_1}$ is perturbed. It will suffice to show that there exists a case satisfying the given conditions in which the equilibrium is unstable for each direction x and y, since any linear combination of this will span the full $2-d$ plane. Consider the first case in figure 7: $\vec{r_1} = \vec{\epsilon} = \epsilon\hat{i}$ at $t = t_0$, which we can then consider our new starting point. Also, let $\vec{r_2} = -a\hat{i}$ and $\vec{r_3} = a\hat{i} = -\vec{r_2}$. Hence at $t = t_0$:

$$\ddot{\vec{r_1}} = -\frac{\vec{\epsilon}-\vec{r_2}}{|\vec{\epsilon}-\vec{r_2}|^3} - \frac{\vec{\epsilon}+\vec{r_2}}{|\vec{\epsilon}+\vec{r_2}|^3} = -\frac{\epsilon+a}{|\epsilon+a|^3} - \frac{\epsilon-a}{|\epsilon-a|^3} = -\frac{\epsilon+a}{|\epsilon+a|^3} + \frac{-(\epsilon-a)}{|\epsilon-a|^3} = -\frac{1}{|\epsilon+a|^2} + \frac{1}{|\epsilon-a|^2} > 0$$

where the second to last step comes from $|\epsilon - a| = -(\epsilon - a)$. Since, the initial perturbation was in the positive x direction and the initial acceleration also comes at the same direction, stars will eventually collapse due to the force unbalance.

Now, let's work out a perturbation in the y direction:

$$\ddot{\vec{r_1}} = -\frac{\vec{\epsilon}-\vec{r_2}}{|\vec{\epsilon}-\vec{r_2}|^3} - \frac{\vec{\epsilon}+\vec{r_2}}{|\vec{\epsilon}+\vec{r_2}|^3} = -\frac{\epsilon\hat{j}+a\hat{i}}{|\epsilon\hat{j}+a\hat{i}|^3} - \frac{\epsilon\hat{j}-a\hat{i}}{|\epsilon\hat{j}-a\hat{i}|^3} = \frac{-\epsilon\hat{j}-a\hat{i}-\epsilon\hat{j}+a\hat{i}}{(\epsilon^2+a^2)^{3/2}} = \frac{-2\epsilon\hat{j}}{(\epsilon^2+a^2)^{3/2}}$$

$$\ddot{\vec{r_2}} = -\frac{-a\hat{i}-\epsilon\hat{j}}{|-a\hat{i}-\epsilon\hat{j}|^3} - \frac{-a\hat{i}-a\hat{i}}{|-a\hat{i}-a\hat{i}|^3} = \frac{a\hat{i}+\epsilon\hat{j}}{|a^2-\epsilon^2|^{3/2}} + \frac{1}{4}\frac{a\hat{i}}{|a|^3} = \frac{\epsilon\hat{j}}{|a^2-\epsilon^2|^{3/2}} + \frac{a\hat{i}}{|a^2-\epsilon^2|^{3/2}} + \frac{1}{4}\frac{a\hat{i}}{|a|^3}$$

and:

$$\ddot{\vec{r_3}} = -\frac{a\hat{i}-\epsilon\hat{j}}{|a\hat{i}-\epsilon\hat{j}|^3} - \frac{a\hat{i}+a\hat{i}}{|a\hat{i}+a\hat{i}|^3} = \frac{a\hat{i}+\epsilon\hat{j}}{|a^2-\epsilon^2|^{3/2}} - \frac{1}{4}\frac{a\hat{i}}{|a|^3} = \frac{\epsilon\hat{j}}{|a^2-\epsilon^2|^{3/2}} - \frac{a\hat{i}}{|a^2-\epsilon^2|^{3/2}} - \frac{1}{4}\frac{a\hat{i}}{|a|^3}$$

From this last equation, it can be seen that: the acceleration for star 1 points towards the origin and depends on the distance from the origin to stars 2 or 3 and the value of the initial perturbation. However, accelerations for stars 2 and 3 will reduce their velocity components along the y-axis while giving them some extra component in the x-axis. Thus, one can see that eventually the system will collapse as well. Cases for velocities initial velocities along the x=axis will complete our proof. However, these cases are trivial. If the initial velocities point inwards, the system will clearly collapse very fast. Also, if the

velocities point outwards, stars 2 and 3 will eventually either diverge on collapse towards 1. The dynamics of the latter case are very complicated. So, simulating the result will be a very powerful argument for this case.

$\therefore$ **A initial perturbation to $\vec{r_1}$ will always lead to very complicated dynamics as we will show numerically below.**

(c) **Suppose (in two dimensions) that the initial conditions are $\vec{r_1} = (0;0)$, $\vec{r_2} = (0;1)$, $\vec{r_3} = (0;-1)$, $\vec{v_1} = (0;0)$, $\vec{v_2} = (1;0)$, and $\vec{v_3} = (-1;0)$ in your non-dimensional units. Numerically integrate the equations of motion using either Verlet, RK4, or an adaptive integration scheme. You are not required to use all three algorithms, just pick one. Your integration should run to the dimensionless time $T = 50$. You are free to choose your timestep Δt or tolerance ϵ, but be aware the solution should be smooth and periodic (so reduce your timestep or debug your code if it is not). For your solutions to this problem, you should submit (i) the code, (ii) a plot of the Kinetic, Potential, and Total energy of the system as a function of time, and (iii) the trajectories of $\vec{r_i}$ on the same plot (that is, $x1(t)$, $x2(t)$, and $x3(t)$ on the x-axis, and $y_1(t)$, $y_2(t)$, and $y_3(t)$ on the y-axis).**

The code was implemented in Matlab. First, we define the initial parameters for our system:

```
clear;
r1=[0.0;0.0];r2=[0;1.0];r3=[0;-1.0];
v1=[0.0;0.0];v2=[1.0;0];v3=[-1.0;0];
y0=[r1;v1;r2;v2;r3;v3];
```

Then, we define the simulation time and the options for the rk45 solver, including the differential equation to be solved as a function:

```
tspan = [0 100];
options = odeset('RelTol', 1.0e-10, 'AbsTol', 1.0E-10);
[t,y] = ode45(@(t,y) odefcn(t,y), tspan, y0, options);

%Odefcn.m
function dydt = odefcn(t,y)
%initial positions
r1=y(1:2);
r2=y(5:6);
r3=y(9:10);
%denominators for diff eqns
r12=norm(r1-r2)^3;
r13=norm(r1-r3)^3;
r23=norm(r2-r3)^3;
%system of diff eqns(linearized):
dydt = zeros(12,1);
dydt(1:2) = y(3:4);
% dydt(3:4) = -(r1-r2)/r12-(r1-r3)/r13;
dydt(5:6) = y(7:8);
dydt(7:8) = -(r2-r1)/r12-(r2-r3)/r23;
dydt(9:10) = y(11:12);
dydt(11:12) = -(r3-r1)/r13-(r3-r2)/r23;
dydt = dydt(:);
end
```

The energies are shown in figure 8. As it can be seen, there is an oscillatory exchange between potential and kinetic energy such that the total energy remains constant in time. The trajectories are shown in figure 9. The result shows circular trajectories that are centered at a point between the corresponding star and star 1. This is expected since the initial conditions fulfill a condition that lets the stars orbit in phase.

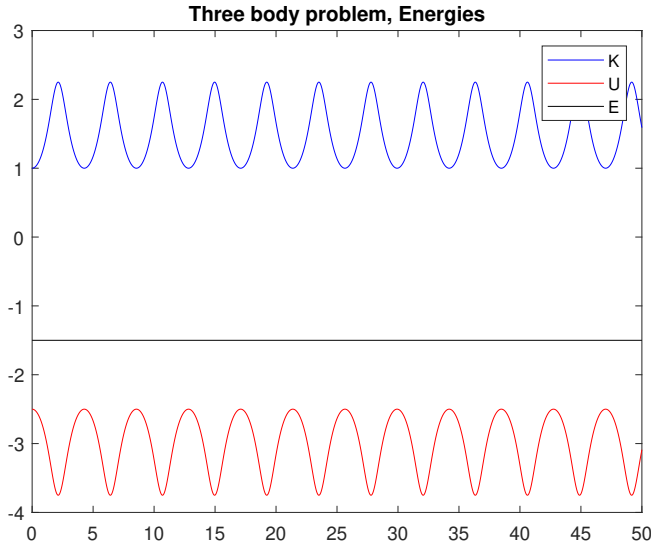

Figure 8: Path for the corresponding initial conditions showing circular orbits for stars 2 and 3 and equilibrium at the origin for star 1.

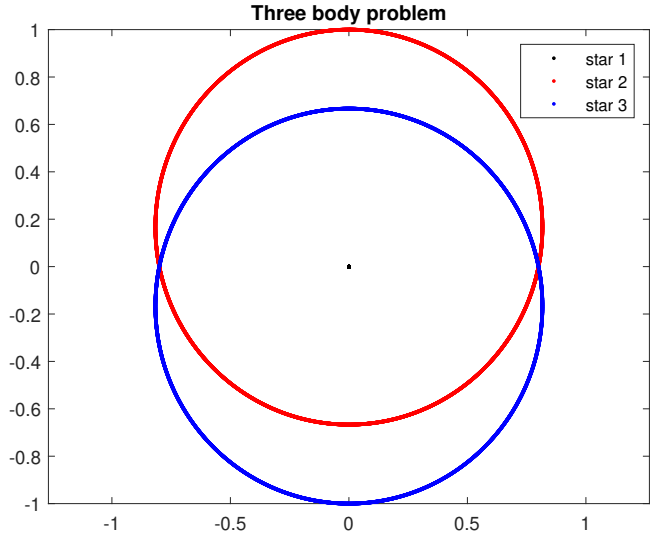

Figure 9: Path for the corresponding initial conditions showing circular orbits for stars 2 and 3 and equilibrium at the origin for star 1.

The code for calculating the kinetic and potential energies and plotting the figures is the following:

```
r1t=[y(:,1), y(:,2)];%star 1 position
r2t=[y(:,5), y(:,6)];%star 2 position
r3t=[y(:,9), y(:,10)];%star 3 position

plot ( r1t(:,1), r1t(:,2), 'k.');
hold on
plot(r2t(:,1), r2t(:,2), 'r.')
plot(r3t(:,1), r3t(:,2), 'b.')
axis equal;
legend('star 1','star 2','star 3')
```

```
title ( 'Three body problem' )
saveas(gcf,'MT5c','epsc');
hold off
r12t=((r1t(:,1)-r2t(:,1)).^2+(r1t(:,2)-r2t(:,2)).^2).^(1/2);
r13t=((r1t(:,1)-r3t(:,1)).^2+(r1t(:,2)-r3t(:,2)).^2).^(1/2);
r23t=((r2t(:,1)-r3t(:,1)).^2+(r2t(:,2)-r3t(:,2)).^2).^(1/2);
Ut=-1./r12t-1./r13t-1./r23t;

v1tsq=y(:,3).^2+y(:,4).^2;
v2tsq=y(:,7).^2+y(:,8).^2;
v3tsq=y(:,11).^2+y(:,12).^2;
Kt=0.5*(v1tsq+v2tsq+v3tsq);

f2=figure;
plot(t,Kt, 'b-')
hold on
plot(t,Ut,'r-')
plot(t,Ut+Kt,'k-')
legend('K','U','E')
title ( 'Three body problem, Energies' )
saveas(gcf,'MT5c_energy','epsc');
hold off
```

(d) **Rerun the simulation using the same timestep or tolerance as in (c), but changing the initial position of the first star to $\vec{r_1} = (0:01;1)$ (with all other initial conditions the same). You should provide code, a plot of the energy, and a plot of the trajectory as in (c). Do you find a periodic orbit? Is that consistent with your answer to (b)?**

The results are shown in figures 10 and 11. The kinetic energy and potential energy, again, change in the opposite way so that the total energy remains constant.

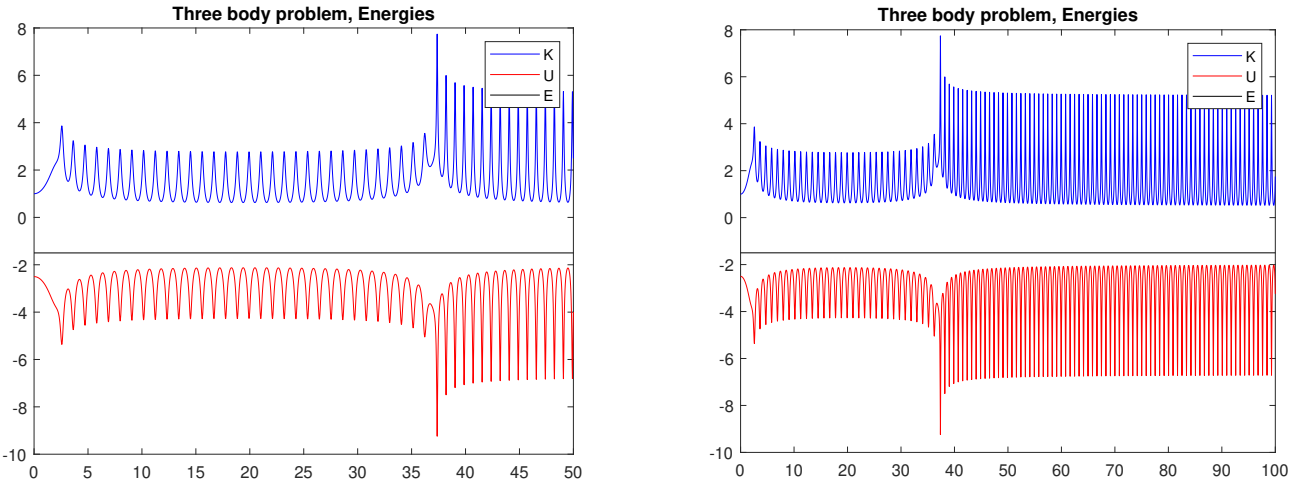

Figure 10: Path for the corresponding initial conditions showing circular orbits for stars 2 and 3 and equilibrium at the origin for star 1.

The trajectories are shown in figure 11. However, the trajectories are not periodic. Stars 1 and 2 couple and move together periodically, whereas star 2 travels around them at first, but eventually runs away in opposite direction. It can be seen that the system stabilizes after a larger time from the energy and path plots, however the situation is not periodic since star 3 will never return towards the origin.

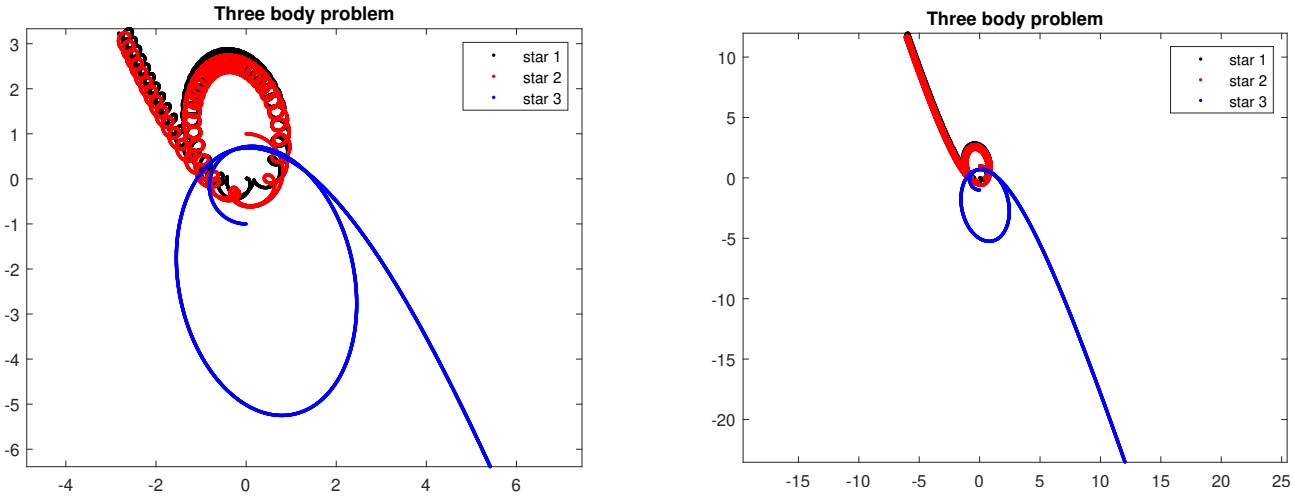

Figure 11: Path for the corresponding initial conditions showing circular orbits for stars 2 and 3 and equilibrium at the origin for star 1.

(e) **Rerun the simulation in (d) using a timestep or tolerance smaller by a factor of 10. Is your trajectory with the different timestep identical to the trajectory you found in (d)? If not, explain why that might be. Note that you do not need to correct this discrepancy to receive full credit, simply discuss its origin.**

The results are shown in figures 12 and 13. The kinetic energy and potential energy, again, change in the opposite way so that the total energy remains constant.

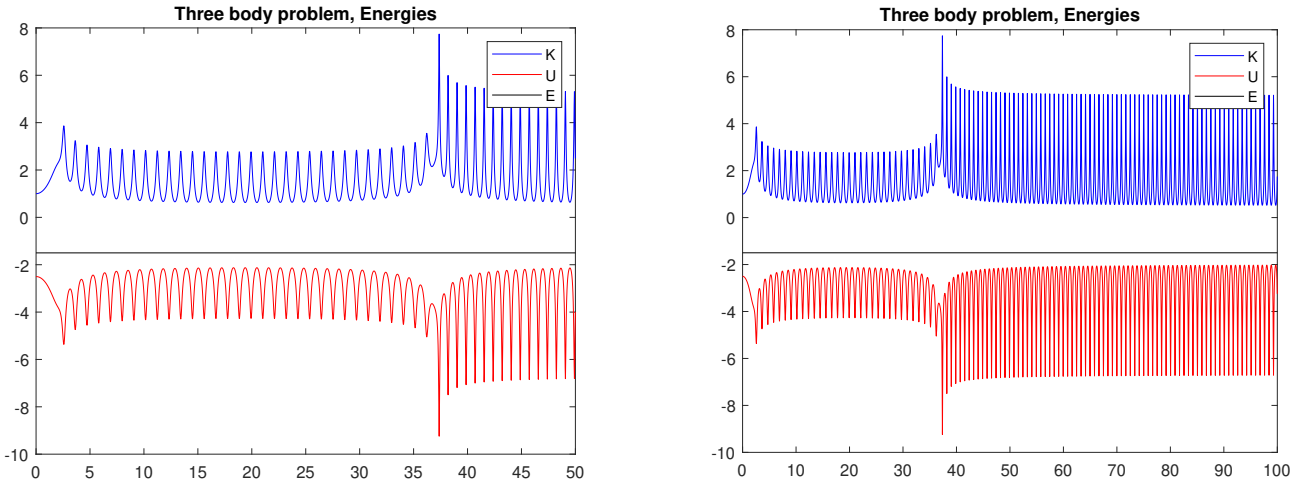

Figure 12: Path for the corresponding initial conditions showing non periodic orbits when a small perturbation is applied to star 1.

The trajectories are shown in figure 11. However, the trajectories are not periodic. Stars 1 and 2 couple and move together periodically, whereas star 2 travels around them at first, but eventually runs away in opposite direction. It can be seen that the system stabilizes after a larger time from the energy and path plots, however the situation is not periodic since star 3 will never return towards the origin.

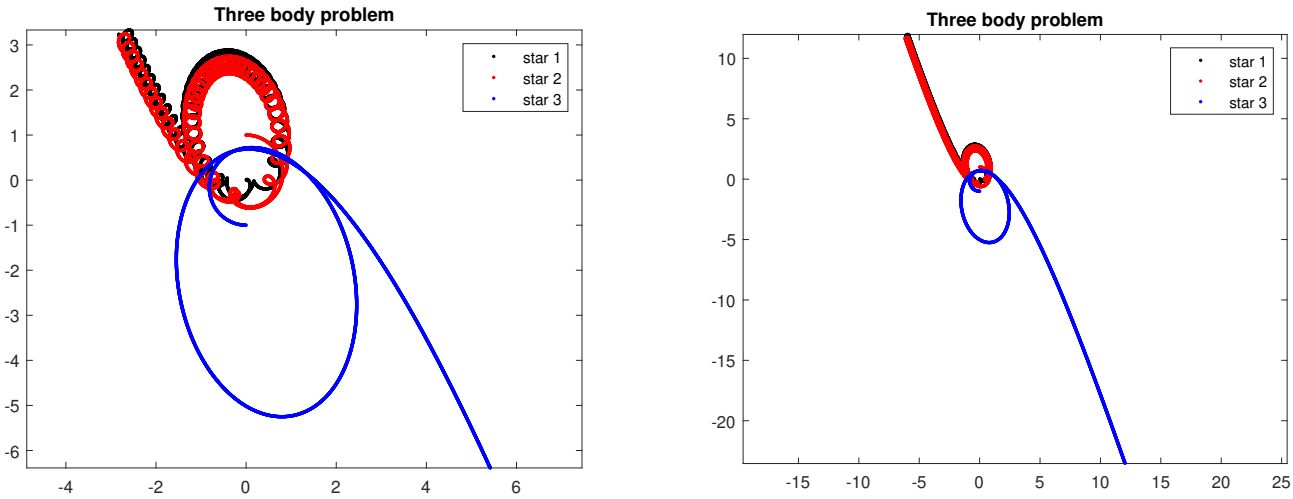

Figure 13: Path for the corresponding initial conditions showing non periodic orbits when a small perturbation is applied to star 1. Tolerance was decreased with respect to the previous part but results did not change.