An implementation of Backward Euler's method on spring system

Eleven (Shiyi) Chen*sc7825@nyu.edu

November 5, 2021

Contents

1 Introduction

In this project, Backward Euler's method and Forward Euler's method have been implemented and compared in the simulation of the spring system. Forward Euler's method is easier to implement than Backward Euler's method. However, when the stiffness of the spring becomes larger, the method breaks down. Although the Backward Euler's method is harder to implement, it provides ultra stability which can deal with stiffer springs.

We made several assumptions regard to our spring:

- The spring system contains a set of springs. Each spring connects two nodes. A node can be connected with multiple springs.
- The springs are weightless. However, there is weight on the nodes of the springs.
- The springs have non-zero rest length. This is important because the rest length will make the system non-linear as we will see in Section 3.

*The numerical method is introduced in Charles S. Peskin's lecture note as provided in the Appendices A.

- The spring is Hookean which means that the force on the spring is propositional to the difference between the length of the spring and the original length of the spring.

2 Mathematical description

In this project, the spring system is consists of Hookean springs, i.e. the spring has no damping coefficients and the force on the spring is only proportional to the difference between the length of the spring and the rest length of the spring. Consider a spring connects a node j and node k, the force node j exerts on node k is given by Equation 1.

$$T_{jk}(t) = S_{jk}(\|\mathbf{X}_j(t) - \mathbf{X}_k(t)\| - R^0_{jk}) \tag{1}$$

where S_{jk} denotes the stiffness of the spring and X_j and X_k denote the location of the node j and node k respectively. R^0_{jk} is the rest length of the spring. A good observation is that T_{jk} is symmetric, i.e. $T_{jk} = T_{kj}$. This is useful in the derivation of the numerical method.

Now define the direction vector $\mathbf{r}_{jk}$ and $\hat{\mathbf{r}}_{jk}$ as

$$\begin{aligned} \mathbf{r}_{jk} &= \mathbf{X}_j - \mathbf{X}_k \\ \hat{\mathbf{r}}_{jk} &= \frac{\mathbf{X}_j - \mathbf{X}_k}{\|\mathbf{X}_j - \mathbf{X}_k\|}. \end{aligned} \tag{2}$$

Now, apply Newton's second law to the node k :

$$M_k \frac{d\mathbf{U}_k}{dt} = \sum_{j \in N(k)} T_{jk}(t)\hat{\mathbf{r}}_{jk}(t) \tag{3}$$

where $N(k)$ denotes the neighboring nodes around node k. M_k is the mass of the node k.

Moreover, we have:

$$\mathbf{U}_k = \frac{d\mathbf{X}_k}{dt} = \frac{dX^1_k}{dt}\hat{\mathbf{e}}_1 + \frac{dX^2_k}{dt}\hat{\mathbf{e}}_2 + \frac{dX^3_k}{dt}\hat{\mathbf{e}}_3 \tag{4}$$

3 Numerical Methods

Our first step is to discretize Equation 3,4. Equation 3 can be discretized as

$$M_k \frac{\mathbf{U}_k(t) - \mathbf{U}_k(t - \Delta t)}{\Delta t} = \sum_{j \in N(k)} T_{jk}(t)\hat{\mathbf{r}}_{jk}(t). \tag{5}$$

Notice that on the right side of the Equation 5, we use the updated forces instead of $\sum_{j \in N(k)} T_{jk}(t - dt)\hat{\mathbf{r}}_{jk}(t - dt)$ because here we want to implement the Backward Euler's method.

Equation 4 can be discretized as

$$\mathbf{U}_k(t) = \frac{\mathbf{X}_k(t) - \mathbf{X}_k(t - \Delta t)}{\Delta t}. \tag{6}$$

Insert Equation 5 to Equation 6 we have:

$$\frac{M_k}{\Delta t^2}(\mathbf{X}_k(t) - \mathbf{Z}_k(t)) = \sum_{j \in N(k)} T_{jk}(t)\hat{\mathbf{r}}_{jk}(t) = \sum_{j \in N(k)} S_{jk}(\|\mathbf{X}_j(t) - \mathbf{X}_k(t)\| - R^0_{jk})\frac{\mathbf{X}_j - \mathbf{X}_k}{\|\mathbf{X}_j - \mathbf{X}_k\|} \tag{7}$$

As $\mathbf{Z}_k(t)$ defined as:

$$\mathbf{Z}_k(t) = \mathbf{X}_k(t - \Delta t) + \mathbf{U}_k(t - \Delta t)\Delta t \tag{8}$$

Equation 7 is the main equation we want to solve. $\mathbf{Z}_k(t)$ can be computed by the last time step. Thus, the main task is to get $\mathbf{X}_k(t)$ for all nodes in the system. Thus, we have $3N$ equations with $3N$ unknowns where N is the total number of the nodes. Notice that because the right-hand side of Equation 7 is non-linear and this is the main challenge of solving it. A great simplification to Equation 7 is to set the rest length R^0_{jk} 0 and this make the equation linear. However, we will discuss the situation when $R^0_{jk} \neq 0$ in this project.

3.1 Newton's Method

We will use Newton's Method to solve Equation 7. Newton's method is an iterative method to solve non-linear equations. The benefit of using Newton's method is its rate of convergence is quadratic, so we should be able to solve the equation in very few steps.

The following example uses Newton's method to solve a quadratic equation, the idea is similar when it comes to a larger system.

Suppose the equation we want to solve is

$$f(x) = 3x^2 + 6x + 1 = 0 \tag{9}$$

The goal here is to construct a sequence $x_0, x_1, \cdots, x_m, \cdots$ that converges to the exact solutions. Let δx_{m-1} be the difference between x_m and x_{m-1} :

$$x_m = x_{m-1} + \delta x_{m-1} \tag{10}$$

Writing out the Taylor expansion for $f(x_m)$ with respect to δx_{m-1} we have

$$\begin{aligned} f(x_m) &= f(x_{m-1} + \delta x_{m-1}) \\ &= f(x_{m-1}) + f'(x_{m-1})\delta x_{m-1} + \mathcal{O}(\delta x^2_{m-1}) \\ &= 3x^2_m + 6x_m + 1 + 6x_m\delta x_{m-1} + 6\delta x_{m-1} + \mathcal{O}(\delta x^2_{m-1}) \end{aligned} \tag{11}$$

Now we want the best δx_{m-1} to minimize $f(x_m)$. By setting $3x^2_m + 6x_m + 1 + 6x_m\delta x_{m-1} + 6\delta x_{m-1} = 0$, we have

$$\delta x_{m-1} = \frac{-3x^2_m - 6x_m - 1}{6(x_m + 1)}. \tag{12}$$

In each step step, we only need to calculate δx_{m-1} by Equation 12, then plug it in and calculate the new x_m by Equation 10. In the same spirit, we are going to derive Newton's method for the spring simulation.

For a system of equations, we are going to construct a sequence of $\mathbf{X}^{(m)}$ such that it converges the solution of Equation 7. Rewrite Equation 7, we define a new function $\mathbf{g}$ as:

$$\mathbf{g}(\mathbf{X}^{(m)}_k) = M_k(\mathbf{X}^{(m)}_k - \mathbf{Z}_k) - \Delta t^2 \sum_{j \in N(k)} T^{(m)}_{jk} \hat{\mathbf{r}}^{(m)}_{jk}. \tag{13}$$

Similar to Equation 11, we have:

$$\begin{aligned} \mathbf{g}(\mathbf{X}^{(m)}_k) &= \mathbf{g}(\mathbf{X}^{(m-1)}_k + \delta\mathbf{X}^{(m-1)}_k) \\ &= \mathbf{g}(\mathbf{X}^{(m-1)}_k) + J\mathbf{g}(\mathbf{X}^{(m-1)}_k)\delta\mathbf{X}^{(m-1)}_k + \mathcal{O}((\delta\mathbf{X}^{(m)}_k)^2) \end{aligned} \tag{14}$$

where $J\mathbf{g}(\mathbf{X}^{(m-1)}_k)$ denotes the Jacobian of $\mathbf{g}$ evaluated at $\mathbf{X}^{(m-1)}_k$. Setting Equation 14 to 0:

$$\mathbf{g}(\mathbf{X}^{(m-1)}_k) + J\mathbf{g}(\mathbf{X}^{(m-1)}_k)\delta\mathbf{X}^{(m-1)}_k = 0. \tag{15}$$

From Equation 15, in order to calculate $\delta\mathbf{X}^{(m-1)}_k$, we must calculate the Jacobian of $\mathbf{g}$. Provided we know $J\mathbf{g}(\mathbf{X}^{(m-1)}_k)$, $\delta\mathbf{X}^{(m-1)}_k$ can be solved with a linear solver.

The meaning of $J\mathbf{g}(\mathbf{X}_k^{(m-1)})\delta\mathbf{X}_k^{(m)}$ is that given a little perturbation in the direction $\delta\mathbf{X}_k^{(m-1)}$ at $\mathbf{X}_k^{(m-1)}$ how much $\mathbf{g}$ changes. Denote $J\mathbf{g}(\mathbf{X}_k^{(m-1)})\delta\mathbf{X}_k^{(m-1)}$ by $\delta\mathbf{g}(\mathbf{X}_k^{(m-1)})$. Apply δ on the both side of Equation 13, we have:

$$\delta\mathbf{g}(\mathbf{X}_k^{(m)}) = M_k\delta\mathbf{X}_k^{(m)} - \Delta t^2 \sum_{j\in N(k)} \delta(T_{jk}^{(m)}\hat{\mathbf{r}}_{jk}^{(m)}). \tag{16}$$

Thus, next step is to calculate $\delta(T_{jk}^{(m)}\hat{\mathbf{r}}_{jk}^{(m)})$.

$$\delta(T_{jk}^{(m)}\hat{\mathbf{r}}_{jk}^{(m)}) = \hat{\mathbf{r}}_{jk}^{(m)}\delta T_{jk}^{(m)} + T_{jk}^{(m)}\delta\hat{\mathbf{r}}_{jk}^{(m)}. \tag{17}$$

Next, we will derive $\delta T_{jk}^{(m)}$ and $\delta\hat{\mathbf{r}}_{jk}^{(m)}$.

$$\begin{aligned}\delta\hat{\mathbf{r}}_{jk} &= \delta\left(\frac{\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|}\right)\\ &= \frac{\delta\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|} - \frac{\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|^2}\delta\|\mathbf{r}_{jk}\|\\ &= \frac{\delta\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|} - \frac{\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|^2}\frac{\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|}\cdot\delta\mathbf{r}_{jk}\\ &= \frac{\delta\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|}\left(\mathbf{I}_3 - \hat{\mathbf{r}}_{jk}\hat{\mathbf{r}}_{jk}^T\right)\end{aligned} \tag{18}$$

Define $\hat{\mathbf{r}}_{jk}\hat{\mathbf{r}}_{jk}^T$ as $\mathbf{P}_{jk}$. Thus, $\delta\hat{\mathbf{r}}_{jk} = \dfrac{\delta\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|}\left(\mathbf{I}_3 + \mathbf{P}_{jk}\right)$.

Next, calculate δT_{jk}.

$$\begin{aligned}\delta T_{jk} &= S_{jk}\delta(\|\mathbf{r}_{jk}\| - R_{jk}^0)\\ &= S_{jk}\frac{\mathbf{r}_{jk}}{\|\mathbf{r}_{jk}\|}\cdot\delta\mathbf{r}_{jk}.\end{aligned} \tag{19}$$

Going back to Equation 17, we have:

$$\begin{aligned}\delta(T_{jk}^{(m)}\hat{\mathbf{r}}_{jk}^{(m)}) &= S_{jk}\mathbf{P}_{jk}^{(m)}\delta\mathbf{r}_{jk}^{(m)} + T_{jk}^{(m)}\frac{\delta\mathbf{r}_{jk}^{(m)}}{\|\mathbf{r}_{jk}^{(m)}\|}\left(\mathbf{I}_3 - \mathbf{P}_{jk}^{(m)}\right)\\ &= \delta\mathbf{r}_{jk}^{(m)}(S_{jk}\mathbf{P}_{jk}^{(m)} + \frac{T_{jk}^{(m)}}{\|\mathbf{r}_{jk}^{(m)}\|}\left(\mathbf{I}_3 - \mathbf{P}_{jk}^{(m)}\right))\end{aligned} \tag{20}$$

Plugging back to Equation 16, we have:

$$\delta\mathbf{g}(\mathbf{X}_k^{(m-1)}) = M_k\delta\mathbf{X}_k^{(m-1)} - \Delta t^2\sum_{j\in N(k)}\delta\mathbf{r}_{jk}^{(m-1)}(S_{jk}\mathbf{P}_{jk}^{(m-1)} + \frac{T_{jk}^{(m-1)}}{\|\mathbf{r}_{jk}^{(m-1)}\|}\left(\mathbf{I}_3 - \mathbf{P}_{jk}^{(m-1)}\right)) \tag{21}$$

Plugging Equation 21 back to Equation 15, we have:

$$M_k(\mathbf{X}_k^{(m-1)} - \mathbf{Z}_k) - \Delta t^2\sum_{j\in N(k)}T_{jk}^{(m-1)}\hat{\mathbf{r}}_{jk}^{(m-1)} + M_k\delta\mathbf{X}_k^{(m-1)} - \Delta t^2\sum_{j\in N(k)}\delta\mathbf{r}_{jk}^{(m-1)}(S_{jk}\mathbf{P}_{jk}^{(m-1)} + \frac{T_{jk}^{(m-1)}}{\|\mathbf{r}_{jk}^{(m-1)}\|}\left(\mathbf{I}_3 - \mathbf{P}_{jk}^{(m-1)}\right)) = 0 \tag{22}$$

Notice that Equation 22 is of form $\mathbf{A}\delta\widetilde{\mathbf{X}}^{(m-1)} = \mathbf{B}$. $\mathbf{A}$ is a $3N$ by $3N$ matrix, $\mathbf{B}$ is a $3N$ column vector, and $\delta\widetilde{\mathbf{X}}^{(m-1)}$ is simply rewrite $\delta\mathbf{X}^{(m-1)}$ into a $3N$ column vector. Plugging into Equation 22, we have:

$$\begin{aligned}
\mathbf{B}_k &= -M_k(\mathbf{X}_k^{(m-1)} - \mathbf{Z}_k) + \Delta t^2 \sum_{j\in N(k)} T_{jk}^{(m-1)}\hat{\mathbf{r}}_{jk}^{(m-1)} \\
\mathbf{A}_{j,k} = \mathbf{A}_{k,j} &= \Delta t^2 \sum_{j\in N(k)} \left(S_{jk}\mathbf{P}_{jk}^{(m-1)} + \frac{T_{jk}^{(m-1)}}{\|\mathbf{r}_{jk}^{(m-1)}\|}\left(\mathbf{I}_3 - \mathbf{P}_{jk}^{(m-1)}\right)\right) \quad k \neq j \\
\mathbf{A}_{k,k} &= -\sum_j \mathbf{A}_{j,k} + M_k\mathbf{I}_3
\end{aligned} \tag{23}$$

Thus, the Newton's method begins by calculating $\mathbf{Z}_k$, then calculates$\mathbf{A}$ and $\mathbf{B}$ to get $\delta\mathbf{X}^{(m-1)}$. Finally, we the result by $\mathbf{X}^{(m)} = \mathbf{X}^{(m-1)} + \delta\mathbf{X}^{(m-1)}$.

4 Computer Program and validation

4.1 Computer program

In the Matlab program, the location of nodes $\mathbf{X}(t)$ is stored in a N by 3 matrices with each row representing the x,y,z coordinate of the node. The code below shows the main loop for the backward simulation:

```
while t<Tfinal
    t = t + dt;
    Xold = X;
    Z = X+dt*U;

    for repeat = 1:repeats
        A = createA(X,R0,S,M3,jj,kk,dt);
        B = createB(X,R0,S,M3,jj,kk,dt,Z);
        dX = A\B;
        X = X+reshape(dX,3,[])';
    end

    % calculate U
    U = (X - Xold)./dt;
    locs = [locs, X(1,1)];
end
```

The matrix **A** is create `createA.m` function:

```
function A = createA(X,R0,S,M3,jj,kk,dt)
DX = X(jj,:) - X(kk,:);
R = sqrt(sum(DX.^2,2));
DXN = DX./[R,R,R];
T = S.*(R-R0);
TR = T./R;

A = diag(M3);

for l = 1:length(jj)
    P = (DXN(l,:))'*DXN(l,:);
    AA = -(dt^2)*(S(l)*P+TR(l)*(eye(3)-P));
    j = (3*jj(l)-2):(3*jj(l));
    k = (3*kk(l)-2):(3*kk(l));
    A(j,k) = AA;
    A(k,j) = AA;
    A(j,j) = A(j,j) - AA;
    A(k,k) = A(k,k) - AA;
end
end
```

The matrix **B** is create with `createB.m` function:

```
function B = create_B(X,R0,S,M3,jj,kk,dt,Z)
DX = X(jj,:) - X(kk,:);
```

```
R = sqrt(sum(DX.^2,2));
DXN = DX./[R,R,R];
T = S.*(R-R0);

B = -M3.*(reshape(X',[],1)-reshape(Z',[],1));

for l = 1:length(jj)
    k = (3*kk(l)-2):(3*kk(l));
    j = (3*jj(l)-2):(3*jj(l));
    B(k) = B(k)+(dt^2)*T(l)*DXN(l,:)';
    B(j) = B(j)-(dt^2)*T(l)*DXN(l,:)';
end
end
```

4.2 Validation

There are several checks that can validate our implementation. The first check is about Euler's method in general. Euler's method has the first order of convergence meaning that to eliminate the error 10 times smaller, one needs to make the time step 10 times smaller. Thus, it is useful to plot the relationship between the time step and the error. Here we are using the x coordinate of the first node as our reference, i.e. `X(1,1)`. To get the value of the error, we need to get the accurate value of `X(1,1)`. The idea here is to use a much smaller time step as our reference accurate value, and compare this value to the value we get using a much larger time step.

In the test, the test problem is shown in Figure 1. N equally weighted nodes were connected with springs with the same stiffness. The whole system is released with zero velocity at time zero. After simulating the system for 0.2 seconds, record the location of the first node. Then compare this value to the reference values. Doing this with dt from `1e-4` to `1e-1` and choose the standard value to be the result we get from `1e-6`, we get Figure 2.

The second test is to check the Newton's method is working. The convergence of the Newton's method is

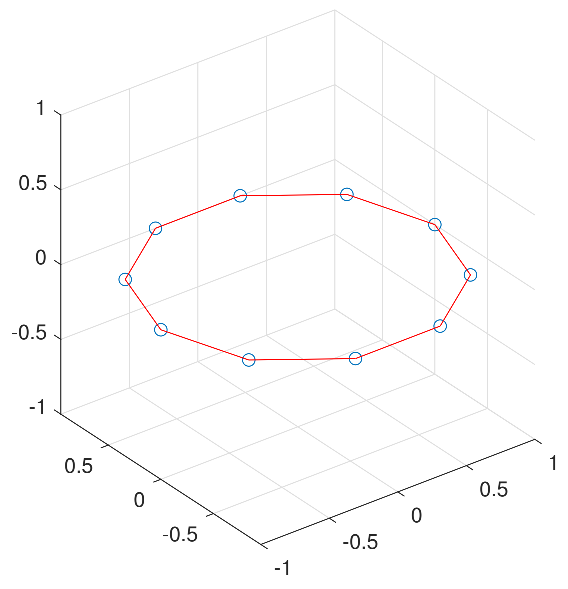


Figure 1: The test problem

quadratic. Looking at the digits of `X(1,1)`, suppose it gives us a corrects digits in the first iteration, it will give us $2a$ correct digits in the second iteration. Thus, by simply looking at `X(1,1)`, we will be able to check the Newton's method. Here is the value of `X(1,1)` in the Newton's iteration:

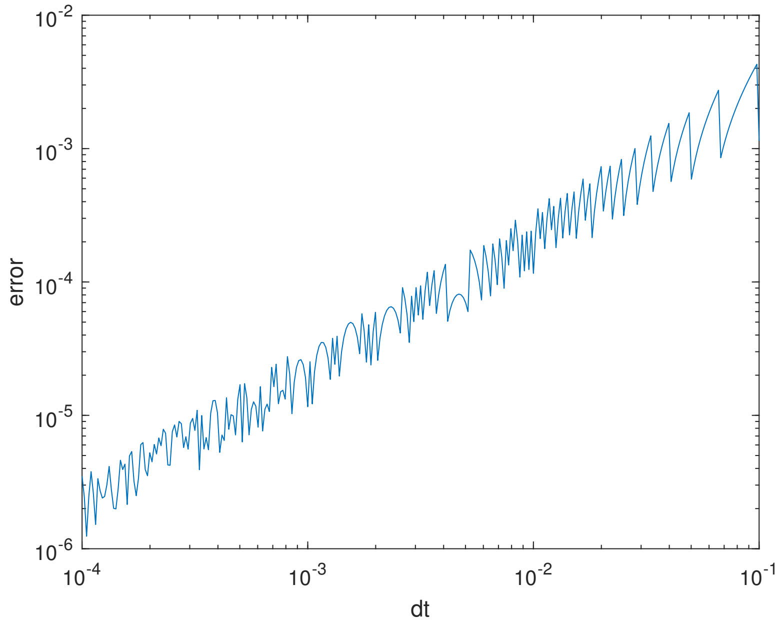


Figure 2: The relationship between the time step and the error

```
K>> x(1)
ans = 1.446037889959311
K>> x(1)
ans = 1.446043151276038
K>> x(1)
ans = 1.446043151275209
K>> x(1)
ans = 1.446043151275209
K>>
```

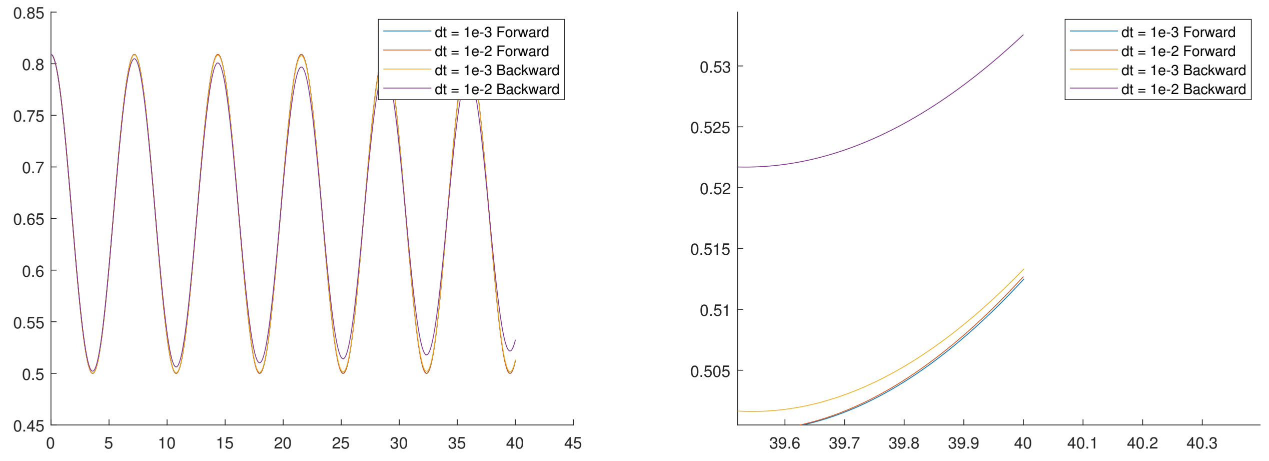
As we can see from the run above, the first iteration gives us 4 correct digits, the second one gives us 7 (almost eight) correct digits which agrees with the convergence of the Newton's method.

The final test is to check that we indeed solve the Equation 7 in each time step. The way to check it is to check whether two sides of Equation 7 are equal after we just finish Newton's method iterations. One notice is that how the right-hand side of Equation 7 agrees with the left-hand side of Equation 7 depends on how small *dt* we choose. Usually, the difference of entry on both sides hit below the machine precision, i.e. `1e-16`.

5 Results

The purpose of this project is to demonstrate the stability of Backward Euler's over the Forward Euler's method. We want to find a case where we use the same parameters for Forward and Backward Euler's method, the Forward version does not converge while the Backward version gives us the correct result.

By performing experiments on the test problem, the Forward Euler's method gives a more accurate result on the test problems by imposing the time steps on both methods are the same.
Figure 3 plots `X(1,1)` with the test problem. Under the same time step, the accuracy of Backward Euler's



(a) Backward Forward comparison with different dt (b) Zoomed around $t = 40s$

Figure 3: Backward and Forward Euler's method working on the test problem

method is lower than that of Forward Euler's method.

Although the Backward Euler's method has a lower accuracy on the test problem, it is a more stable solver when it comes to the stiffer springs. A gyroscope is created as shown in Figure 4. At the time 0, the

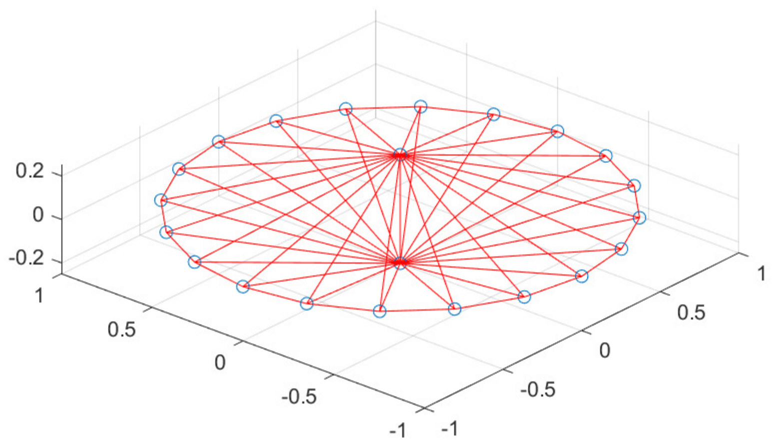


Figure 4: The gyroscope we created

gyroscope was given a initial angular velocity. There is no external forces applied to the gyroscope. Using `dt = 1e-4` and `s = 1e8` on the gyroscope, the Backward Euler's method is still working but the gyroscope

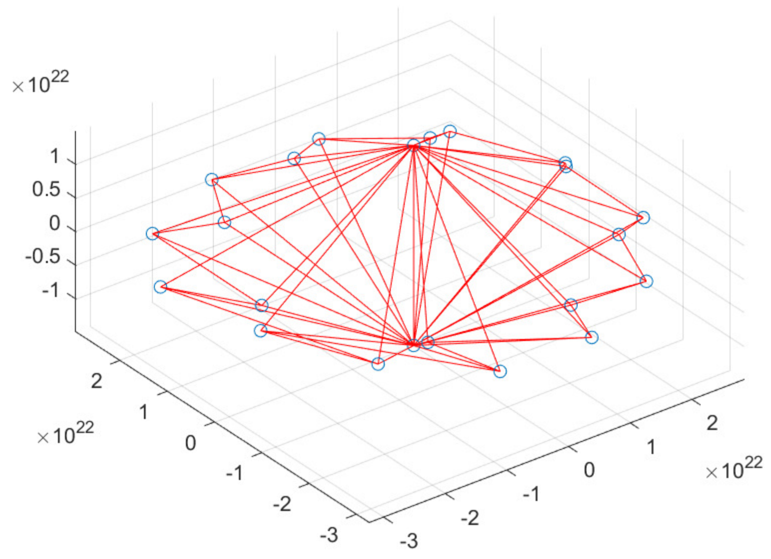


Figure 5: with `dt = 1e-4` and `s = 1e8`, the Forward Euler's method fails

split apart using Forward Euler's method as shown in Figure 5.

To conclude, the Forward Euler's method has advantage in the accuracy of solution while the Backward Euler's method can deal with stiffer springs.

6 Conclusion

As a summary. This project mainly focuses on the implementation of the Backward Euler's method. We build up the numerical method step by step in Section 3. In the result Section, we show the stability of the Backward Euler's method against the Forward Euler's method. Although the Backward Euler's method handles better on a stiffer spring systems, it generates a less accurate result compared to the Forward version.

Appendices

A Charles S. Peskin's lecture note

https://www.math.nyu.edu/~peskin/modsim_lecture_notes/backward_euler.pdf

B Runnable programs

https://github.com/Eleven7825/Backward-Euler-Spring

C Backward simulation generated by program

https://youtu.be/0as0sl2yQqY