Adding Loads to Dynamics

With the dynamics a bit more organized we want to take a look at a bit more complicated systems. Now we will add some external loads, no actuation yet, for now we will look at both gravity and viscosity. So, we will now have a $\bar{W}$ term to include in the dynamics and we will have to integrate g in the explicit Euler steps as well (left it out previously because it was unnecessary).

The load for gravity is:

$$\bar{W}_{grav} = \rho A \begin{bmatrix} 0 \\ R^T g_r \end{bmatrix}$$

where g_r is the gravitational acceleration vector

The load for viscosity is:

$$\bar{W}_{vis} = V\dot{\xi}$$

where $V = \mu * diag([3I, 3I, J, A, A, 3A])$ is the viscosity matrix.

First lets add in the integration of g in the explicit Euler stepping first.

```
def _integrate(self, prev, dt, xi0):
        self.xi[0, :] = xi0

        # integration over the body (don't need the initial point as the initial values are
        g_half = np.eye(4) # known initial condition
        for i in range(self.N - 1):
            # averaging over steps to get half step values
            xi_half = (self.xi[i, :] + prev.xi[i, :]) / 2
            eta_half = (self.eta[i, :] + prev.eta[i, :]) / 2

            # implicit midpoint approximation
            xi_dot = (self.xi[i, :] - prev.xi[i, :]) / dt
            eta_dot = (self.eta[i, :] - prev.eta[i, :]) / dt

            # spatial derivatives
            xi_der = np.linalg.inv(self.K) @ (
                    (self.M @ eta_dot) - (adjoint(eta_half).T @ self.M @ eta_half) + (
```

```
                    adjoint(xi_half).T @ self.K @ (xi_half - self.xi_ref)))
            eta_der = xi_dot - (adjoint(xi_half) @ eta_half)

            # explicit Euler step
            xi_half_next = xi_half + self.ds * xi_der
            eta_half_next = eta_half + self.ds * eta_der
            g_half = g_half @ expm(se(self.ds * xi_half))

            # determine next step from half step value
            self.xi[i + 1, :] = 2 * xi_half_next - prev.xi[i + 1, :]
            self.eta[i + 1, :] = 2 * eta_half_next - prev.eta[i + 1, :]

        # midpoint RKMK to step the g values
        for i in range(self.N):
            self.g[i, :] = flatten(unflatten(prev.g[i, :]) @ expm(se(dt * (self.eta[i, :] +
```

This is simply done by initializing the base value for g and then stepping it with each iteration.

Now to implement gravity and viscosity. Viscosity fits as a material property so we can extend the initialization to include it, but I think that gravity doesn't quite fit so it will be hardcoded for now and then looked at some abstraction later.

Adding viscosity to the construction is simple:

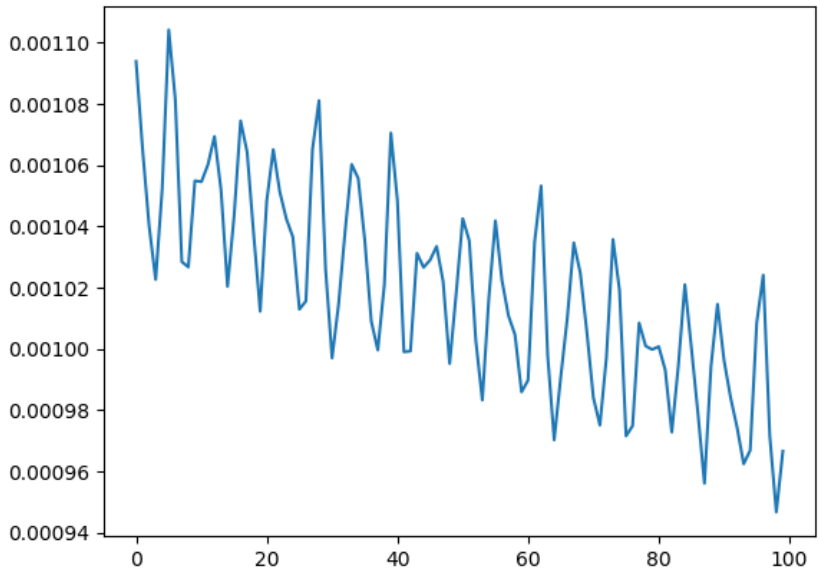
```
# add mu as an argument
self.V = mu * np.diag([3*I,3*I,J,A,A,3*A])
```

Now adding to the loads:

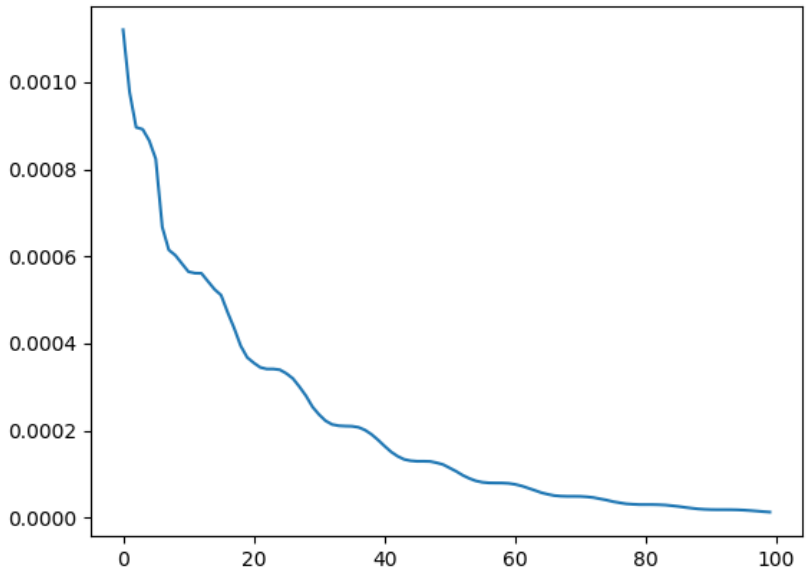
```
def _integrate(self, prev, dt, xi0):
        self.xi[0, :] = xi0

        # integration over the body (don't need the initial point as the initial values are
        g_half = np.eye(4) # known initial condition
        for i in range(self.N - 1):
            # averaging over steps to get half step values
            xi_half = (self.xi[i, :] + prev.xi[i, :]) / 2
            eta_half = (self.eta[i, :] + prev.eta[i, :]) / 2

            # implicit midpoint approximation
            xi_dot = (self.xi[i, :] - prev.xi[i, :]) / dt
            eta_dot = (self.eta[i, :] - prev.eta[i, :]) / dt

            # external loads
            W_bar = 0
            W_bar += self.V @ xi_dot
```

```
            # spatial derivatives
            xi_der = np.linalg.inv(self.K) @ (
                    (self.M @ eta_dot) - (adjoint(eta_half).T @ self.M @ eta_half) + (
                        adjoint(xi_half).T @ self.K @ (xi_half - self.xi_ref)) + W_bar)
            eta_der = xi_dot - (adjoint(xi_half) @ eta_half)

            # explicit Euler step
            xi_half_next = xi_half + self.ds * xi_der
            eta_half_next = eta_half + self.ds * eta_der
            g_half = g_half @ expm(se(self.ds * xi_half))

            # determine next step from half step value
            self.xi[i + 1, :] = 2 * xi_half_next - prev.xi[i + 1, :]
            self.eta[i + 1, :] = 2 * eta_half_next - prev.eta[i + 1, :]

        # midpoint RKMK to step the g values
        for i in range(self.N):
            self.g[i, :] = flatten(unflatten(prev.g[i, :]) @ expm(se(dt * (self.eta[i, :] +
```

Now we have the `W_bar` term. To test to see if this works as expected if we include some viscosity into the system and we use the previous conservative test setup we should see the energy gradually dissipate. For `mu=300`:



Then using a larger viscosity it should dissipate energy faster. For `mu=30000`:



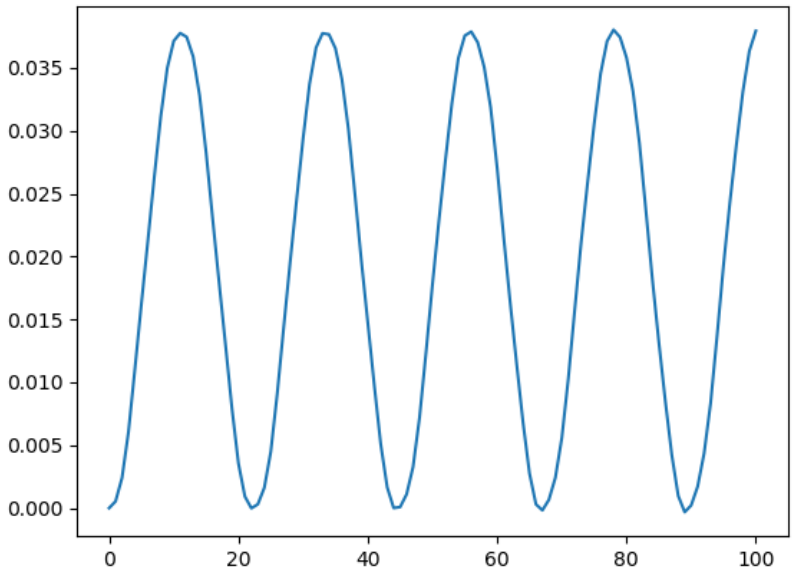
Now for gravity we can simply add it to the `W_bar` term (keeping gravitational acceleration hardcoded for now):

```
def _integrate(self, prev, dt, xi0):
        self.xi[0, :] = xi0
        grav = np.array([-9.81,0,0])

        # integration over the body (don't need the initial point as the initial values are
        g_half = np.eye(4) # known initial condition
        for i in range(self.N - 1):
            # averaging over steps to get half step values
            xi_half = (self.xi[i, :] + prev.xi[i, :]) / 2
            eta_half = (self.eta[i, :] + prev.eta[i, :]) / 2

            # implicit midpoint approximation
            xi_dot = (self.xi[i, :] - prev.xi[i, :]) / dt
            eta_dot = (self.eta[i, :] - prev.eta[i, :]) / dt

            # external loads
            W_bar = 0
            W_bar += self.V @ xi_dot
            R = g_half[:3,:3]
            W_bar += self.rho *self.A * np.concatenate([np.array([0,0,0]), R.T @ grav])
```

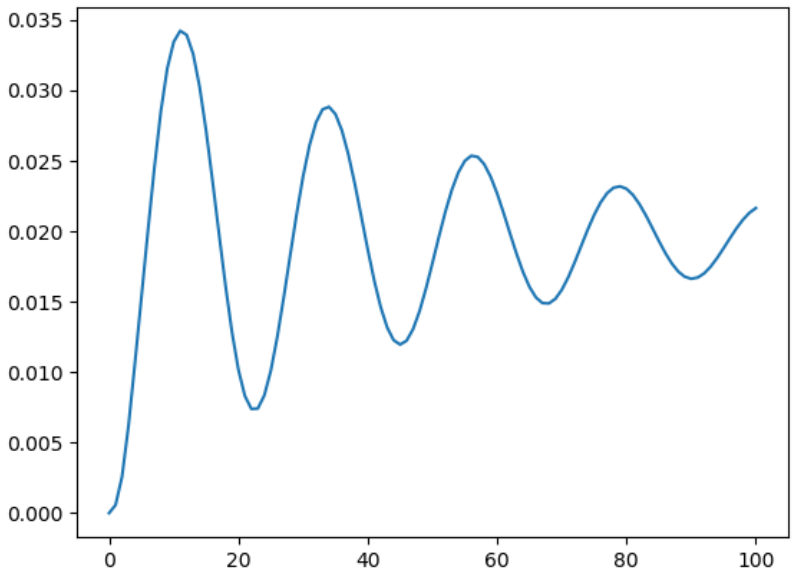
```
            # spatial derivatives
            xi_der = np.linalg.inv(self.K) @ (
                    (self.M @ eta_dot) - (adjoint(eta_half).T @ self.M @ eta_half) + (
                        adjoint(xi_half).T @ self.K @ (xi_half - self.xi_ref)) + W_bar)
            eta_der = xi_dot - (adjoint(xi_half) @ eta_half)

            # explicit Euler step
            xi_half_next = xi_half + self.ds * xi_der
            eta_half_next = eta_half + self.ds * eta_der
            g_half = g_half @ expm(se(self.ds * xi_half))

            # determine next step from half step value
            self.xi[i + 1, :] = 2 * xi_half_next - prev.xi[i + 1, :]
            self.eta[i + 1, :] = 2 * eta_half_next - prev.eta[i + 1, :]

        # midpoint RKMK to step the g values
        for i in range(self.N):
            self.g[i, :] = flatten(unflatten(prev.g[i, :]) @ expm(se(dt * (self.eta[i, :] +
```

To test to see if this makes sense we can have the rod act as a cantilever beam and track the tip position through time from an initially straight configuration. If we include viscosity it should start converging towards steady state, if viscosity is not included it should vibrate forever.

Without viscosity:



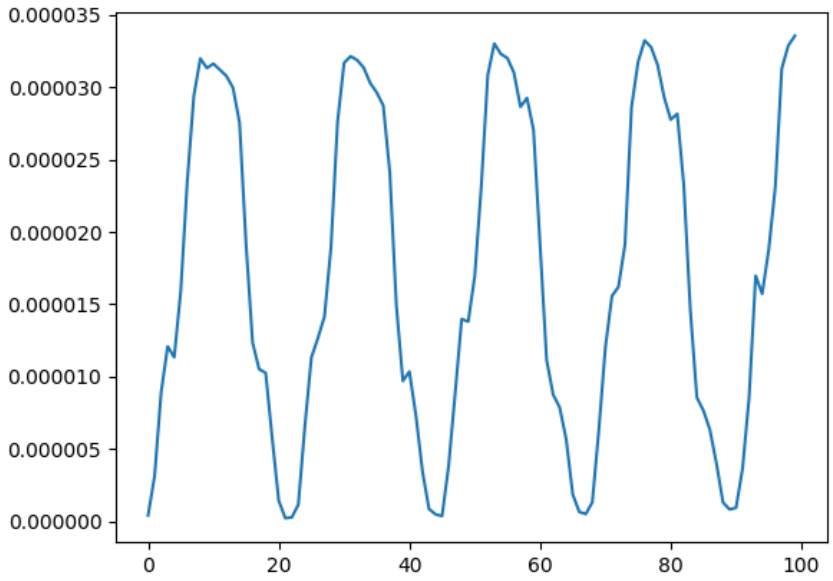
With viscosity:



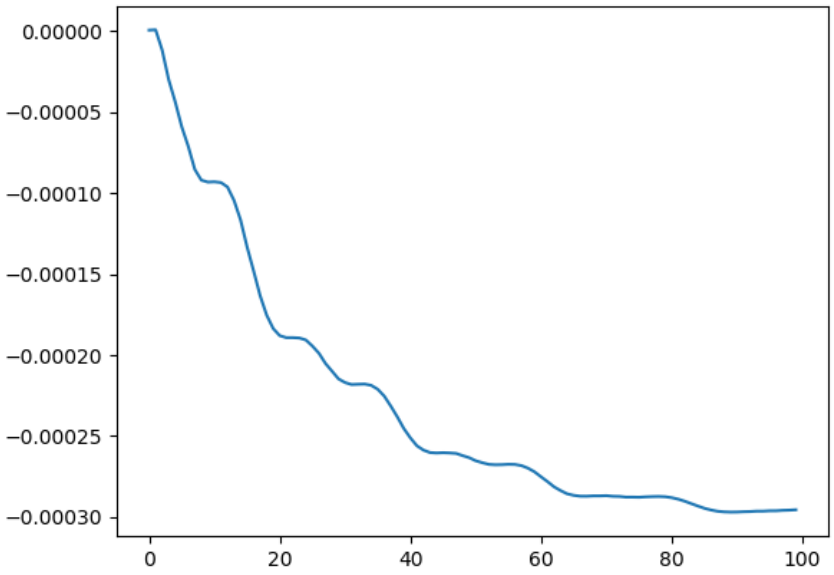
While this confirms our expectations it'd be nice to compute the total energy in the system considering gravity is a conservative force. So we will have to include gravity into the energy computation to see if it is conserved numerically. To do this we still have the same kinetic and strain energy parts and for gravity we integrate along the rod with $\Delta s \rho A x g$ where x is the x position of the cross section and g is the gravitational acceleration. So, with no viscosity we expect the energy to still be conserved.

```
def grav_energy(sys):
    H = sys.energy()
    for i in range(sys.N):
        H += -sys.ds*sys.rho*sys.A*sys.g[i,9]*9.81 # negative sign because it drops below th
    return H
```

The resulting energy oscillates slightly about 0 energy.



Then including viscosity as well we see the energy dissipates:



Here we saw the addition of gravity and viscosity into the system and see that symplecticity is still maintained with the right parameters.