# Multiple Segments

Now that a single segment has been defined and works it would be nice to work with multiple segments in series. This could be for having complex layouts of actuators or it could be for having point loads throughout the body. The major thing this helps with is point loads from external loads or from actuators being fixed to certain spots. Modeling a point load as a distributed force is relatively difficult as it needs to be defined by a dirac delta which would only be evaluated if a node is exactly at the right spot all of which is cumbersome. However, splitting into segments provides a simpler way of dealing with point loads as they act like tip conditions for their segment. So, what needs to happen is creating an interface for defining a series of segment that has the same interface as the single segment rods defined before and the conditions at the interface between segments needs to be handled properly.

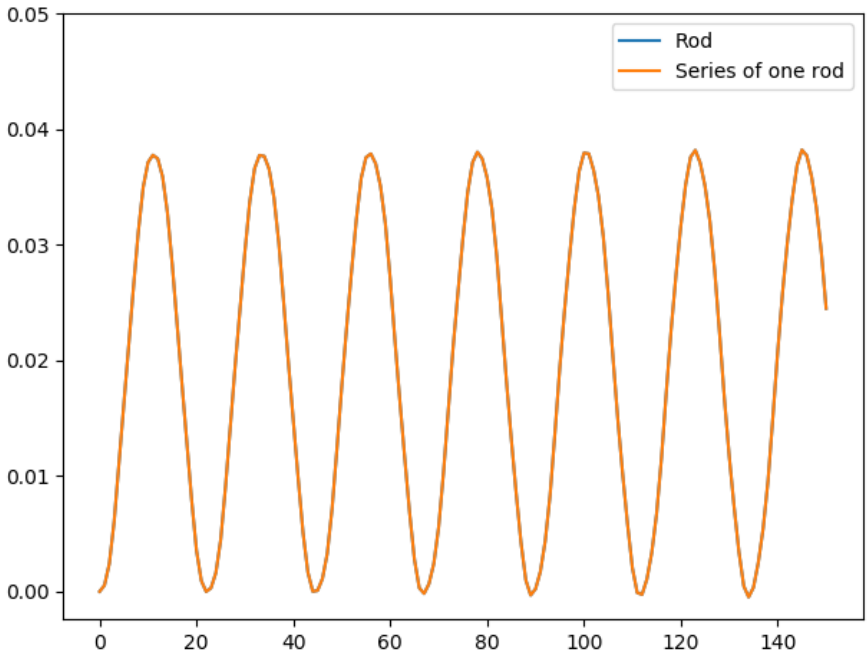
If we look at the initial definition for the rods and think about what it'd be like to put them in series we see that the tip node for one rod is the base node for the other and therefore should share the same state. So, what the integrator will have to do is make the state for these nodes consistent. This means that the system still only has one unknown initial condition, the one at the very base, and therefore can use the same scheme for solving as a single rod. Effectively doing this just guarantees that a node will be at the right location for a point load, but it also allows for different rod definitions to be easily joined.

This makes it pretty straight forward to implement, we can borrow the `_integrate` methods from the rods and modify them to take more initial conditions, then the condition to solve for is the tip condition for the final segment. There are some other minor tweaks, like for initializing the rods, but they are fairly minor.

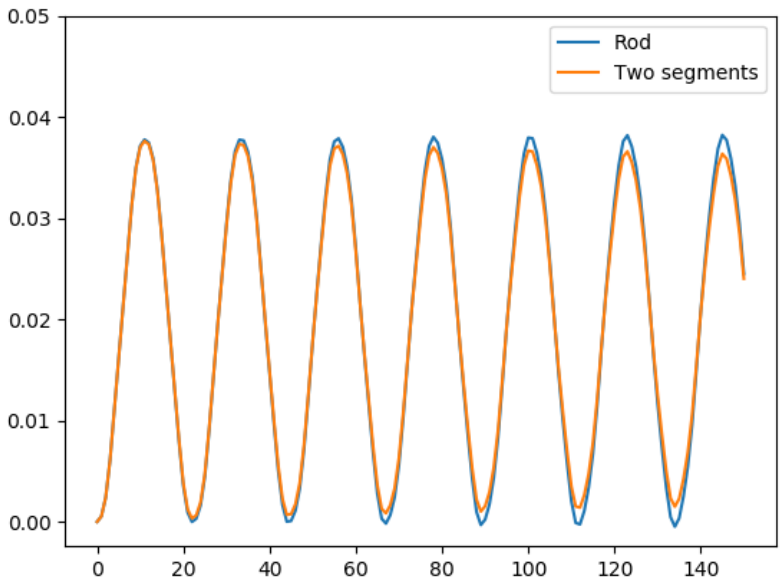
```
class Series():
    """
    A series of rods
    integrates and manages multiple rods
    """

    def __init__(self, rods):
        self.rods = rods
        self._initRods()
```

```
def _initRods(self):
    g0 = np.eye(4)
    for rod in self.rods:
        rod._initRod(g0)
        g0 = unflatten(rod.g[-1,:])

def plot(self, ax=None):
    for rod in self.rods:
        ax = rod.plot(ax)
    return ax

def energy(self):
    return sum([rod.energy() for rod in self.rods])

def step(self, dt, q):
    prev = copy.deepcopy(self)
    xi0 = fsolve(lambda x: self._condition(prev, dt, x, q), self.rods[0].xi[0, :])

def _condition(self, prev, dt, xi0, q):
    # same as before except just final rod
    self._integrate(prev, dt, xi0, q)

    # all tip loads
    W = 0
    # data
    rod = self.rods[-1]  # final segment
    g = unflatten(rod.g[-1, :])
    xi = rod.xi[-1, :]
    eta = rod.xi[-1, :]
    xi_dot = (rod.xi[-1, :] - rod.xi[-1, :]) / dt
    eta_dot = (rod.eta[-1, :] - rod.eta[-1, :]) / dt
    for load in rod.loads:
        W += load.tip_load(g, xi, eta, xi_dot, eta_dot, rod, q)

    return rod.K @ (rod.xi[-1, :] - rod.xi_ref) - W

    pass

def _integrate(self, prev, dt, xi0, q, g0=np.eye(4), eta0=np.array([0, 0, 0, 0, 0, 0])):
    for (i, rod) in enumerate(self.rods):
        rod._integrate(prev.rods[i], dt, xi0, q, g0, eta0)
        g0 = unflatten(rod.g[-1, :])
        xi0 = rod.xi[-1, :]
        eta0 = rod.eta[-1, :]
```

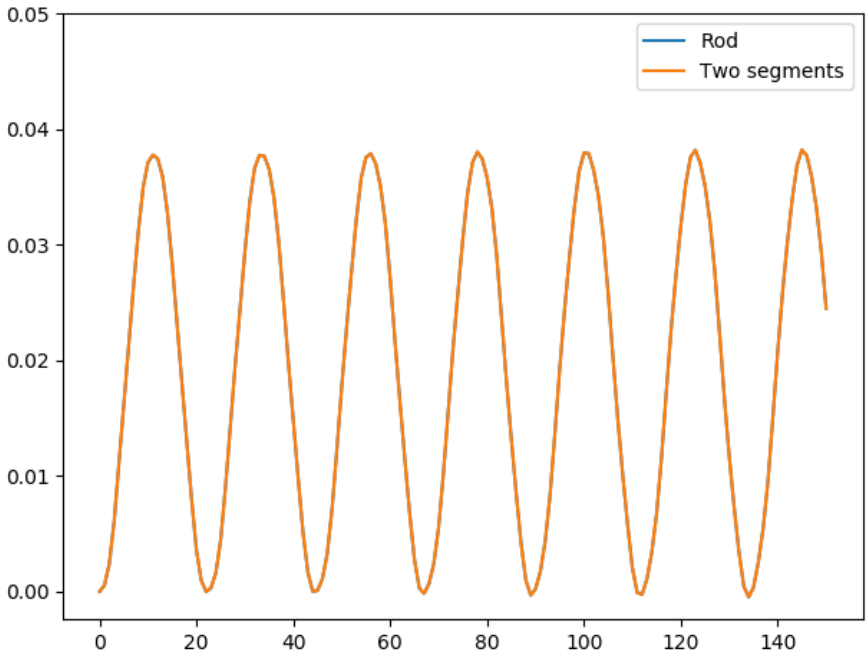
Now to make sure this work right we can compare with previous results. We know that a series of one rod should be identical to just the rod itself and if we were to just divide the original rod into some segments the result should also be the same. Here I check the cantilever rod responding to gravity.




Which shows that the responses are indistinguishable between the rod and a one rod series as expected. However, with the segments we see that there is some energy dissipation to investigate.

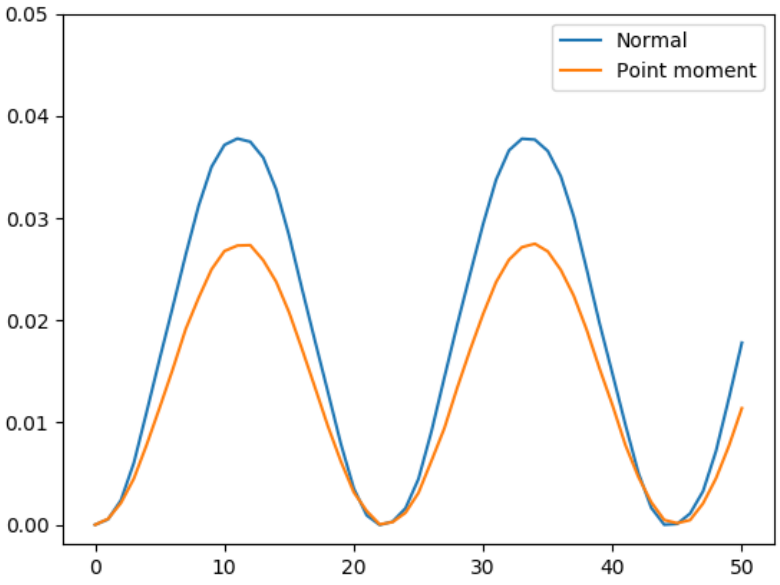



Running through several tests to see where the issue is with the integration of segments we eventually see that it was a subtle point. We need to start the next segments integration, `g0`, using the final `g_half` from the previous segment and not the final `g` because the integration is on the half time step and needs to be consistent. Making adjustments to accommodate for this we get identical behavior again as expected, also as the number of segments increases the results stay identical.




Now the main point of making multiple segments was to include point loads at locations other than the tip and this formulation doesn't quite do that yet. In order to include point loads they need to be included into the distributed load considerations when integrating. The simplest way to do this is to flag segments as being intermediate segments (not tip segments) or not. If they are intermediate segments apply tip loads at the final point in the segment. This works, but in my opinion is a bit messy and has room for some generalization in how segments are defined.

Here is an example of the cantilever rod response to gravity compared with a rod responding to gravity that also has a point moment applied 1/3 of the way along the rod.


Normal
Point moment
0.05
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
50