# More Complex Material and Geometric Properties

So far the considerations of the material properties and shape of the rod have been considered fairly simple. The rod is a cylinder, so its cross section is a constant circular shape, and the material is linear giving a simple form of the stiffness. Now there are many situations that would entail more complex considerations which we are going to look at now.

The primary place this considerations changes is the strain energy definition. So, now the energy will be defined in a more generic way as:

$$U = \int u(\Delta\xi)dV$$

where $u$ is the strain energy density function and therefore the strain energy is the integral of $u$ over the entire body.

Now we can start to look at how the variation of the strain energy, $\delta U$, changes with a more general material model. We now get:

$$\delta U = \langle \delta h, \int \nabla_{\Delta\xi} u dA \rangle - \int \langle \delta h, ad_{\xi}^{T} (\int \nabla_{\Delta\xi} u dA) - \frac{\partial}{\partial s}(\int \nabla_{\Delta\xi} u dA) \rangle ds$$

For convenience we define:

$$\Psi(s, \Delta\xi) = \int \nabla_{\Delta\xi} u dA$$

If we make this substitution the strong form of the variation of the strain energy is:

$$ad_{\xi}^{T}\Psi - \frac{\partial}{\partial s}\Psi$$

Using the product rule we can expand this to:

$$ad_\xi^T \Psi - \frac{\partial \Psi}{\partial \Delta\xi}\xi' - \Psi'$$

So, in our original model we have $\Psi = K\Delta\xi$. If we rearrange the dynamics equation to $\xi' = \ldots$ as needed by the simulation:

$$\xi' = (\frac{\partial \Psi}{\partial \Delta\xi} - \bar{A})^{-1}(M\dot{\eta} - ad_\eta^T M\eta + ad_\xi^T \Psi - \Psi' + \bar{B})$$

This shows that if we want to define more general material properties and geometric features we need to define $\Psi$, $\Psi'$, and $\frac{\partial \Psi}{\partial \Delta\xi}$ to complete the dynamics.

While it would be nice to be able to separate the geometric and material properties in the simulations, in general that won't be possible due to the definition of $\Psi$ depending on an integral over the cross sections. However, in certain forms we should be able to separate the definitions. So, we will look at some general definitions for the material and geometric properties and see when can we define special cases where the material and shape definitions can be separated.

For hyperelastic materials we can consider a general polynomial model of the material properties which is:

$$u = \sum_{i,j=0} C_{ij}(I_1 - 3)^i (I_2 - 3)^j + \sum_{k=1} D_k (J-1)^{2k}$$

where $C_{ij}$ and $D_k$ are material constants determined empirically and $I_1$, $I_2$, and $J$ are invariants of $F^T F$ where $F$ is the deformation gradient. Also we assume $C_{00} = 0$ because the energy should be 0 with no deformation. In this case invariants means independent of coordinate rotations. Using the previously defined $F$ we get that the invariants are:

$$I_1 = (\nu_y + \omega_z r_x x)^2 + (\nu_x - \omega_z r_y)^2 + (\nu_z - \omega_y r_x + \omega_x r_y + 1)^2 + 2 \tag{1}$$
$$I_2 = (\nu_y + \omega_z r_x)^2 + (\nu_x - \omega_z r_y)^2 + 2(\nu_z - \omega_y r_x + \omega_x r_y + 1)^2 + 1 \tag{2}$$
$$J = \nu_z - \omega_y r_x + \omega_x r_y + 1 \tag{3}$$
$$\tag{4}$$

where the components of $\Delta\xi$ have been broken up and the $\Delta$ dropped for simplicity, all the components are changes from the reference.

If we tried to do the symbolic manipulation to determine the form of $\Psi$ for arbitrary coefficients it does not appear possible and $\Psi$ would have to be determined for a specific case. However, if we restrict the polynomial for $u$ to only go upto a certain order then this could be possible.

Now we can start to implement the nonlinearity definitions into the system. First, we define the general interface for the body loads from $\Psi$.

```python
class Body(metaclass=ABCMeta):
    """
    General interface for the body definition

    has to implement the values associated with the material and geometric parameters
    Psi, Psi', and dPsi/dxi
    """

    @abstractmethod
    def Psi(self, xi, s, rod):
        return np.array([0, 0, 0, 0, 0, 0])

    @abstractmethod
    def Psi_prime(self, xi, s, rod):
        return np.array([0, 0, 0, 0, 0, 0])

    @abstractmethod
    def Psi_der(self, xi, s, rod):
        return np.zeros((6, 6))
```

Now the original rod needs to take a body argument and use the appropriate functions in the dynamics equation. However, it becomes apparent that the body should hold more information than just the wrench due to strain. It should define all the body dependent parameters like the inertia, $M$, viscosity, and length, $L$. So, the body should have more required information. Also, now the geometry is allowed to vary with $s$ and things are no longer just matrices so a few more change need to be made as well.

The first things to replace are the obvious parts in the rod dynamics and move them to the body definition. The body then includes inertia, viscosity, length, and $\Psi$ terms and the rod is defined by the body it is given rather than the material and geometric parameters. With no external loads these changes do not change the results of the simulations. However, with gravity we knew that it depends on both the density and the cross sectional area of the rods and we could pull that information from the rod definition, which is why the loads take the relevant rod as an argument, but now that information is not available and the gravitational load will not work correctly. To make gravity work again we have a few things to consider: we do not want to repeat ourselves when supplying data (i.e., define density once), gravity is a commonly used load but isn't always included, and want to keep all the interfaces pretty general. To rearrange how gravity is computed we need it to have access to the body definition and do not want to have to specify it for all systems. This makes gravity sit in a weird place it is most convenient to have it defined in the body as far as data goes, but we want it to be abstracted away from the body so if similar loads exist they can be defined similarly. So, gravity will have to access data from the body, but we do not want to need to define that data when gravity is not present as that just adds extra, unnecessary work. To deal with this gravity will try to call a method in the body to compute its needed values and this method will default to raising an unimplemented error this way it isn't absolutely necessary to make it work and it will alert the user when it needs to be defined. Seems like there could be a more abstracted way, but I'm not thinking of it or quite sure how I'd explain it in the first place, something like I want to keep the environment and the body cleanly separated, but gravity requires both and blurs the boundaries. Also, want the blurring of the boundaries to be generic which I haven't really achieved.

With the rambling out of the way, we have an interface for working with a body with generic material properties and shape. Now, since specific cases of the material properties can be handled in a generic way we can define a body with the material properties and geometry separate. The material case we look at is the polynomial model with only first order terms (I believe this is the incompressible Mooney-Rivlin model).

$$u = C_1(I_1 - 3) + C_2(I_2 - 3)$$

So, the material is fully defined by the coefficients $C_1$ and $C_2$. If we go through the process of computing $\Psi$ we get:

$$\Psi = \int \begin{bmatrix} (2C_1+4C_2)x^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & (2C_1+4C_2)y^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & (2C_1+2C_2)(x^2+y^2) & 0 & 0 & 0 \\ 0 & 0 & 0 & 2C_1+2C_2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2C_1+2C_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2C_1+4C_2 \end{bmatrix} \Delta\xi dA$$

Which can be simplified by substituting more familiar parameters $E = 2C_1+4C_2$ and $G = 2C_1 + 2C_2$. So, we see two parameters similar to the elastic, $E$, and shear, $G$, modulus. Then integrating we get:

$$\Psi = K\Delta\xi$$

where $K = \text{diag}([EI_x, EI_y, GJ, GA, GA, EA])$.

So, with the first order model we can easily define the body with the material and geometric properties decoupled. Implementing the same simulation as before works as expected.