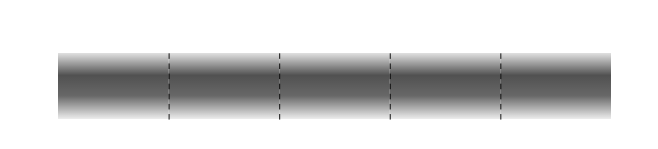
One-Dimensional Heat Transfer

Our previous discussion on [*State Space*](http://book.xogeny.com/behavior/arrays/state_space/#state-space) introduced matrices and vectors. The focus was primarily on mathematical aspects of arrays. In this section, we will consider how arrays can be used to represent something a bit more physical, the one-dimensional spatial distribution of variables. We’ll look at several features in Modelica that are related to arrays and how they allow us to compactly express behavior involving arrays.

Our problems will center around a simple heat transfer problem. Consider a one-dimensional rod like the one shown below:



Deriving Equations

Let us consider the heat balance for each discrete section of this rod. First, we have the thermal capacitance of the ith section. This can expressed as:

miCTi

where m is the mass of the ith section, C is the capacitance (a material property) and Ti is the temperature of the ith section. We can further describe the mass as:

mi=ρVi

where ρ is the material density and Vi is the volume of the ith section. Finally, we can express the volume of the ith section as:

Vi=AiLi

where Ai is the cross-sectional area of the ith section, which is assumed to be uniform, and Li is the length of the ith section. For this example, we will assume the rod is composed of equal size pieces. In this case, we can define the segment length, Li, to be:

Li=L/n

We will also assume that the cross-sectional area is uniform along the length of the rod. As such, the mass of each segment can be given as:

m=ρALi

In this case, the thermal capacitance of each section would be:

ρALiCTi

This, in turn, means that the net heat gained in that section at any time will be:

ρALiCdTi/dt

where we assume that A, Li and C don’t change with respect to time.

That covers the thermal capacitance. In addition, we will consider two different forms of heat transfer. The first form of heat transfer we will consider is convection from each section to some ambient temperature, Tamb. We can express the amount of heat lost from each section as:

qh=−hA(Ti−Tamb)

where h is the convection coefficient. The other form of heat transfer is conduction to neighboring sections. Here there will be two contributions, one lost to the i−1th section, if it exists, and the other lost to the i+1th section, if it exists. These can be represented, respectively, as:

qki→i−1=−kA(Ti−Ti−1)/Li

qki→i+1=−kA(Ti−Ti+1)/Li

Using these relations, we know that the heat balance for the first element would be:

ρALiCdT1/dt=−kA(T1−T2)/Li−hA(T1−Tamb)

Similarly, the heat balance for the last element would be:

ρALiCdTn/dt=−kA(Tn−Tn−1)/Li−hA(Tn−Tamb)

Finally, the heat balance for all other elements would be:

ρALiCdTidt=−kA(Ti−Ti−1)/Li−kA(Ti−Ti+1)/Li−hA(Ti−Tamb)

Implementation

We start by defining types for the various physical quantities. This will give us the proper units and, depending on the tool, allows us to do unit checking on our equations. Our type definitions are as follows:

**type** **Temperature**=Real(unit="K", min=0);

**type** **ConvectionCoefficient**=Real(unit="W/K", min=0);

**type** **ConductionCoefficient**=Real(unit="W.m-1.K-1", min=0);

**type** **Mass**=Real(unit="kg", min=0);

**type** **SpecificHeat**=Real(unit="J/(K.kg)", min=0);

**type** **Density**=Real(unit="kg/m3", min=0);

**type** **Area**=Real(unit="m2");

**type** **Volume**=Real(unit="m3");

**type** **Length**=Real(unit="m", min=0);

**type** **Radius**=Real(unit="m", min=0);

We will also define several parameters to describe the rod we are simulating:

**parameter** Integer n=10;

**parameter** Length L=1.0;

**parameter** Radius R=0.1;

**parameter** Density rho=2.0;

**parameter** ConvectionCoefficient h=2.0;

**parameter** ConductionCoefficient k=10;

**parameter** SpecificHeat C=10.0;

**parameter** Temperature Tamb=300 "Ambient temperature";

Given these parameters, we can compute the area and volume for each section in terms of the parameters we have already defined using the following declarations:

**parameter** Area A = pi\*R^2;

**parameter** Volume V = A\*L/n;

Finally, the only array in this problem is the temperature of each section (since this is the only quantity that actually varies along the length of the rod):

Temperature T[n];

This concludes all the declarations we need to make. Now let’s consider the various equations required. First, we need to specify the initial conditions for the rod. We will assume that T1(0)=200, Tn(0)=300 and the initial temperatures of all other sections can be linearly interpolated between these two end conditions. This is captured by the following equation:

**initial equation**

T = linspace(200,300,n);

where the linspace operator is used to create an array of n values that vary linearly between 200 and 300. Recall from our [*State Space*](http://book.xogeny.com/behavior/arrays/state_space/#state-space) examples that we can include equations where the left hand side and right hand side expressions are vectors. This is another example of such an equation.

Finally, we come to the equations that describe how the temperature in each section changes over time:

**equation**

rho\*V\*C\*der(T[1]) = -h\*(T[1]-Tamb)-k\*A\*(T[1]-T[2])/(L/n);

**for** i **in** 2:(n-1) **loop**

rho\*V\*C\*der(T[i]) = -k\*A\*(T[i]-T[i-1])/(L/n)-k\*A\*(T[i]-T[i+1])/(L/n);

**end** **for**;

rho\*V\*C\*der(T[**end**]) = -h\*(T[**end**]-Tamb)-k\*A\*(T[**end**]-T[**end**-1])/(L/n);

The first equation corresponds to the heat balance for section 1, the last equation corresponds to the heat balance for section n and the middle equation covers all other sections. Note the use of end as a subscript. When an expression is used to evaluate a subscript for a given dimension, end represents the size of that dimension. In our case, we use end to represent the last section. Of course, we could use n in this case, but in general, end can be very useful when the size of a dimension is not already associated with a variable.

Also note the use of a for loop in this model. A for loop allows the loop index variable to loop over a range of values. In our case, the loop index variable is iand the range of values is 2 through n−1. The general syntax for a for loop is:

**for** <var> **in** <range> **loop**

*// statements*

**end** **for**;

where <range> is a vector of values. A convenient way to generate a sequence of values is to use the range operator, :. The value before the range operator is the initial value in the sequence and the value after the range operator is the final value in the sequence. So, for example, the expression 5:10 would generate a vector with the values 5, 6, 7, 8, 9 and 10. Note that this **includes** the values used to specify the range.

When a for loop is used in an equation section, each iteration of the for loop generates a new equation for each equation inside the for loop. So in our case, we will generate n−2 equations corresponding to values of i between 2 and n-1.

Putting all this together, the complete model would be:

**model** **Rod\_ForLoop** "Modeling heat conduction in a rod using a for loop"

**type** **Temperature**=Real(unit="K", min=0);

**type** **ConvectionCoefficient**=Real(unit="W/K", min=0);

**type** **ConductionCoefficient**=Real(unit="W.m-1.K-1", min=0);

**type** **Mass**=Real(unit="kg", min=0);

**type** **SpecificHeat**=Real(unit="J/(K.kg)", min=0);

**type** **Density**=Real(unit="kg/m3", min=0);

**type** **Area**=Real(unit="m2");

**type** **Volume**=Real(unit="m3");

**type** **Length**=Real(unit="m", min=0);

**type** **Radius**=Real(unit="m", min=0);

**constant** Real pi = 3.14159;

**parameter** Integer n=10;

**parameter** Length L=1.0;

**parameter** Radius R=0.1;

**parameter** Density rho=2.0;

**parameter** ConvectionCoefficient h=2.0;

**parameter** ConductionCoefficient k=10;

**parameter** SpecificHeat C=10.0;

**parameter** Temperature Tamb=300 "Ambient temperature";

**parameter** Area A = pi\*R^2;

**parameter** Volume V = A\*L/n;

Temperature T[n];

**initial equation**

T = linspace(200,300,n);

**equation**

rho\*V\*C\*der(T[1]) = -h\*(T[1]-Tamb)-k\*A\*(T[1]-T[2])/(L/n);

**for** i **in** 2:(n-1) **loop**

rho\*V\*C\*der(T[i]) = -k\*A\*(T[i]-T[i-1])/(L/n)-k\*A\*(T[i]-T[i+1])/(L/n);

**end** **for**;

rho\*V\*C\*der(T[**end**]) = -h\*(T[**end**]-Tamb)-k\*A\*(T[**end**]-T[**end**-1])/(L/n);

**end** **Rod\_ForLoop**;

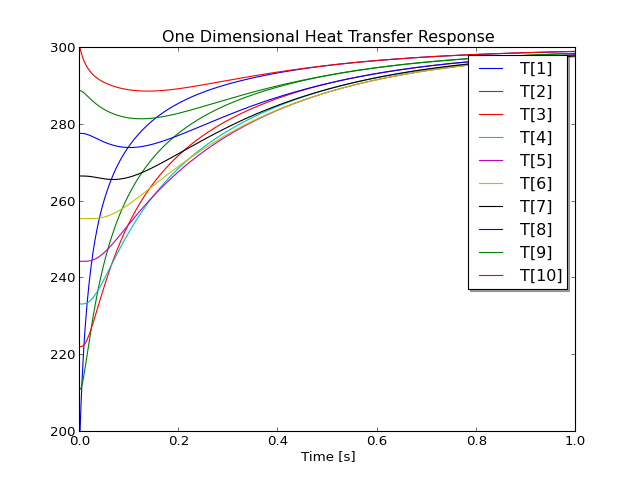
**Note**

Note that we’ve included pi as a literal constant in this model. Later in the book, we’ll discuss how to properly import common [*Constants*](http://book.xogeny.com/components/packages/msl/#constants).

Simulating this model yields the following solution for each of the nodal temperatures:

([Source code](http://book.xogeny.com/plots/RFL.py))

 Simulate Rod\_ForLoop in your browser



Note how the temperatures are initially distributed linearly (as we specified in our initial equation section).

Alternatives

It turns out that there are several ways we can generate the equations we need. Each has its own advantages and disadvantages depending on the context. We’ll present them here just to demonstrate the possibilities. The choice of which one they feel leads to the most understandable equations is up to the model developer.

One array feature we can use to make these equations slightly simpler is called an array comprehension. An array comprehension flips the for loop around so that we take a single equation and add some information at the end indicating that the equation should be evaluated for different values of the loop index variable. In our case, we can represent our equation section using array comprehensions as follows:

**equation**

rho\*V\*C\*der(T[1]) = -h\*(T[1]-Tamb)-k\*A\*(T[1]-T[2])/(L/n);

rho\*V\*C\*der(T[2:n-1]) = {-k\*A\*(T[i]-T[i-1])/(L/n)-k\*A\*(T[i]-T[i+1])/(L/n) **for** i **in** 2:(n-1)};

rho\*V\*C\*der(T[**end**]) = -h\*(T[**end**]-Tamb)-k\*A\*(T[**end**]-T[**end**-1])/(L/n);

We could also combine the array comprehension with some if expressions to nullify contributions to the heat balance that don’t necessarily apply. In that case, we can simplify the equation section to the point where it contains one (admittedly multi-line) equation:

**equation**

rho\*V\*C\*der(T) = {-h\*(T[i]-Tamb)

-(**if** i==1 **then** 0 **else** k\*A/(L/n)\*(T[i]-T[i-1]))

-(**if** i==n **then** 0 **else** k\*A/(L/n)\*(T[i]-T[i+1])) **for** i **in** 1:n};

Recall, from several previous examples, that Modelica supports vector equations. In these cases, when the left hand and right hand side are vectors of the same size, we can use a single (vector) equation to represent many scalar equations. We can use this feature to simplify our equations as follows:

**equation**

rho\*V\*C\*der(T[1]) = -h\*(T[1]-Tamb)-k\*A\*(T[1]-T[2])/(L/n);

rho\*V\*C\*der(T[2:n-1]) = -k\*A\*(T[2:n-1]-T[1:n-2])/(L/n)-k\*A\*(T[2:n-1]-T[3:n])/(L/n);

rho\*V\*C\*der(T[**end**]) = -h\*(T[**end**]-Tamb)-k\*A\*(T[**end**]-T[**end**-1])/(L/n);

Note that when a vector variable like T has a range of subscripts applied to it, the result is a vector containing the components indicated by the values in the subscript. For example, the expression T[2:4] is equivalent to {T[2], T[3], T[4}. The subscript expression doesn’t need to be a range. For example,T[{2,5,9}] is equivalent to {T[2], T[5], T[9]}.

Finally, let us consider one last way of refactoring these equations. Imagine we introduced two additional vector variables:

Heat Qleft[n];

Heat Qright[n];

Then we can write these two equations (again using vector equations) to define the heat lost to the previous and next section in the rod:

Qleft = {**if** i==1 **then** -h\*(T[i]-Tamb) **else** -k\*A\*(T[i]-T[i-1])/(L/n) **for** i **in** 1:n};

Qright = {**if** i==n **then** -h\*(T[i]-Tamb) **else** -k\*A\*(T[i]-T[i+1])/(L/n) **for** i **in** 1:n};

This allows us to express the heat balance for each section using a vector equation that doesn’t include any subscripts:

rho\*V\*C\*der(T) = Qleft+Qright;

Conclusion

In this section, we’ve seen various ways that we can use vector variables and vector equations to represent one-dimensional heat transfer. Of course, this vector related functionality can be used for a wide range of different problem types. The goal of this section was to introduce several features to demonstrate the various options that are available to the developer when working with vectors.