Transient One-Dimensional Heat Diffusion

Problem Definition

In this lesson, we explore the solution of the transient one-dimensional heat diffusion equation, given by:

$$\frac{\partial \left(\rho c_p T\right)}{\partial t} = k\nabla^2 T + S$$

For the current lesson we shall consider the density, ρ and specific heat capacity, c_p, to be constants.

Discretization

To discretize, we integrate the governing equation over space and also over time:

$$\int_{t_0}^{t_1}\int_V \frac{\partial \left(\rho c_p T\right)}{\partial t} dt dV = \int_{t_0}^{t_1}\int_V k\nabla^2 T dt dV + \int_{t_0}^{t_1}\int_V S dt dV$$

Let us first assume a timestep $\Delta t = t_1 - t_0$ and assume that the solution is stored at the time levels t and $t + \Delta t$. In this case, we can assume various profiles for the integrands as functions of time. Here we will examine the following:

- **Fully explicit**: evaluate integrands on the right side of the equation above at the initial time level, $t_0 = t$
- **Fully implicit**: evaluate integrands on the right side of the equaiton above at the final time level, $t_1 = t + \Delta t$
- **Crank-Nicolson**: assume a linear variation of integrands on the right side of the equation above over the time interval Δt (equivalent to trapezoid rule)

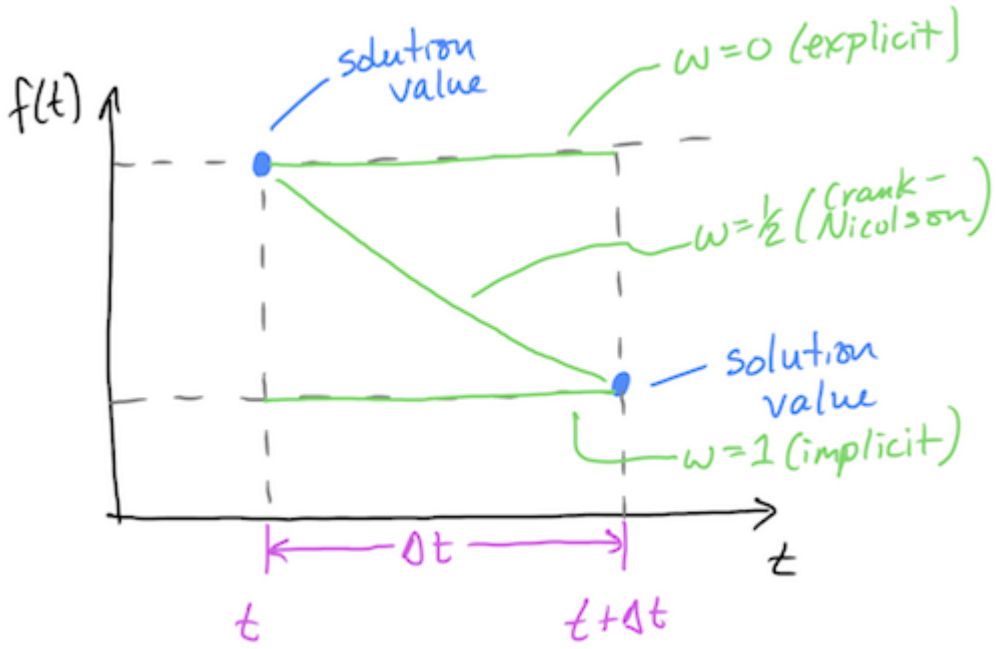
In this case, the left side of the equation can be integrated by interchanging the order of integration, which results in, for the control volume P:

$$\int_{t_0}^{t_1}\int_V \frac{\partial \left(\rho c_p T\right)}{\partial t} dV dt = \left(\rho c_p T_P V_P\right)^{t+\Delta t} - \left(\rho c_p T_P V_P\right)^{t}$$

The diffusion term can be integrated, similar to before, as:

$$\int_{t_0}^{t_1}\int_V k\nabla^2 T dt dV = -\left[\omega\left(F_e^d\right)^{t+\Delta t} + (1-\omega)\left(F_e^d\right)^{t}\right]\Delta t + \left[\omega\left(F_w^d\right)^{t+\Delta t} + (1-\omega)\left(F_w^d\right)^{t}\right]\Delta t$$

where ω is a weighting function that controls the assumed variation of the integrand over the timestep, as explained by the following figure.



Grouping the time levels together, we have:

$$\int_{t_0}^{t_1}\int_V k\nabla^2 T dt dV = \omega\left[F_w^d - F_e^d\right]^{t+\Delta t}\Delta t + (1-\omega)\left[F_w^d - F_e^d\right]^t \Delta t$$

Integration of the source term proceeds in a similar mannar, but for now we will assume $S = 0$.

> **Exercise:** Derive the discretized source term for the transient case considered.

In the case with $S = 0$, the discretized equation becomes:

$$\left(\rho c_p T_P V_P\right)^{t+\Delta t} - \left(\rho c_p T_P V_P\right)^t = \omega\left[F_w^d - F_e^d\right]^{t+\Delta t}\Delta t + (1-\omega)\left[F_w^d - F_e^d\right]^t \Delta t$$

Dividing both sides by Δt and assuming that ρ, c_p, and V_P are constant in time, we obtain:

$$\rho c_p \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t} V_P = \omega\left[F_w^d - F_e^d\right]^{t+\Delta t} + (1-\omega)\left[F_w^d - F_e^d\right]^t$$

It may be observed at this point that the fully implicit and fully explicit schemes are only first order accurate in time, while the Crank-Nicolson scheme is second order in time (based on the number of time points used in evaluating the integral). However, the Crank-Nicolson scheme can be less stable and may cause the solution to oscillate in time. Soon, we will consider how to develop a fully implicit scheme that is second-order accurate in time. But first, we will look into the properties of the schemes that we have derived so far.

Some general conclusions that can be drawn so far:

- The explicit solution method does not require the solution of a system of equations. In the last lesson, the system of equations was used because the current solution values were unknown so we had to linearize the problem. In the explicit method, all of the diffusive fluxes (and any other terms) are calculated using the solution values from the previous timestep which are all known.

- As in the previous lesson, the implicit method requires the solution of a system of linear equations for the values at the current timestep, since these are unknown when the discretized equations are assembled. The same is true for the Crank-Nicolson scheme, or any scheme where $0 < \omega \leq 1$

Analysis of Explicit Scheme

The explicit scheme can be re-written as:

$$\rho c_p \frac{T_P^{t+\Delta t}}{\Delta t} V_P = \left[F_w^d - F_e^d\right]^t + \rho c_p \frac{T_P^t}{\Delta t} V_P$$

From before we have:

$$F_e^d = -k \frac{T_E - T_P}{\Delta x_{PE}} A_e = -D_e \left(T_E - T_P\right)$$

$$F_w^d = -k \frac{T_P - T_W}{\Delta x_{WP}} A_w = -D_w \left(T_P - T_W\right)$$

where $D_e = kA_e/\Delta x_{PE}$ and $D_w = kA_w/\Delta x_{WP}$ are defined for convenience. In general, these coefficients could be considered to vary in time, but for the time being they are considered to be independent of time. In this notation, the explicit scheme can be expressed as:

$$\rho c_p \frac{T_P^{t+\Delta t}}{\Delta t} V_P = D_e T_E^t + D_w T_W^t + \left(\frac{\rho c_p V_P}{\Delta t} - D_e - D_w\right) T_P^t$$

In order to have the correct physical influence, the coefficient on T_P^t must be positive. This ensures that raising T_P^t will cause a rise in $T_P^{t+\Delta t}$. Similarly, this ensures that a drop in T_P^t will cause a a drop in $T_P^{t+\Delta t}$. Violation of this requirement may cause oscillations in the solution over time, as well as solution instability. In order for this coefficient to remain positive, the timestep must be selected such that:

$$\frac{\rho c_p V_P}{\Delta t} \geq D_e + D_w$$

Or:

$$\Delta t \leq \frac{\rho c_p V_P}{D_e + D_w} = \frac{1}{\frac{D_e}{\rho c_p V_P} + \frac{D_w}{\rho c_p V_P}}$$

If we assume that that $V_P = A\Delta x$, where A is the representative cross-sectional area of the domain at P and Δx is the grid spacing:

$$\frac{D_e}{\rho c_p V_P} \sim \frac{\frac{kA}{\Delta x}}{\rho c_p A \Delta x} = \frac{k}{\rho c_p} \frac{1}{\Delta x^2} = \frac{\alpha}{\Delta x^2}$$

We may interpret the quantity $\Delta x^2/\alpha$ as the timescale associated with conduction through the face (note that the units are indeed seconds).

Assuming a uniform grid spacing around P, the timestep restriction is:

$$\Delta t \leq \frac{1}{\frac{\alpha}{\Delta x^2} + \frac{\alpha}{\Delta x^2}} = \frac{\Delta x^2}{2\alpha}$$

Consider the case of an iron bar with $\alpha = 23.1 \times 10^{-6}$ [m^2/s] and a spatial grid resolution of $\Delta x = 0.01$ [m]. In this case the timestep must be less than 2.16 [s]. If the grid sizing is decreased to $\Delta x = 0.001$ [m], then the timestep restriction becomes 0.0216 [s]. This can result in a fairly significant timestep restriction that gets worse with increasing grid refinement. This is one of the reasons that implicit methods are more commonly used in practice. One exception is the calculation of turbulent flows using direct numerical simulaiton (DNS), which sometimes uses explicit methods, since the timescale must be small anyways to capture the turbulent fluctuations. In this case, explicit methods are a good choice since they are less expensive per timestep than implicit methods, since no linear system must be solved.

Analysis of Fully-Implicit Scheme

The fully-implicit scheme is obtained by setting $\omega = 1$, resulting in:

$$\rho c_p \frac{T_P^{t+\Delta t} - T_P^t}{\Delta t} V_P = \left[-D_w \left(T_P - T_W\right) + D_e \left(T_E - T_P\right)\right]^{t+\Delta t}$$

If we drop the superscripts $t + \Delta t$ for simplicity and denote 'old' values at t by the superscript 'o', we have:

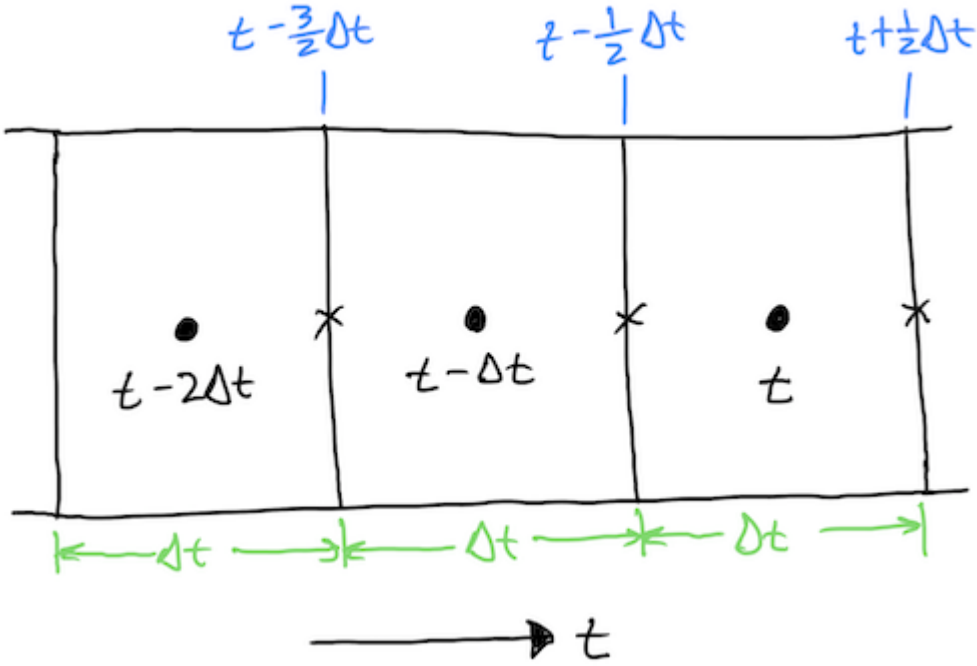
$$\left(\frac{\rho c_p V_P}{\Delta t} + D_w + De\right) T_P = D_w T_W + D_e T_E + \frac{\rho c_p V_P}{\Delta t} T_P^o$$

In this case it is impossible for any of the coefficients to become negative, thus there is no restriction on Δt. However, one must still ensure that the timestep is small enough to resolve all transient phenomena that one is interested in.

Similar to the fully-implicit scheme, the Crank-Nicholson scheme has no formal restriction on Δt, but can produce oscilaltory solutions for large Δt

Derivation of a Second-Order Implicit Scheme

A second-order implicit scheme can be derived by considering integration over a space-time control volume where the 'time faces' are located at times $t - \Delta t/2$ and $t + \Delta t/2$. The solution values are stored at times t, $t - \Delta t$, $t - 2\Delta t$, ..., which are in the centre of the time control volume. A schematic diagram of the time control volume is shown below:



As a result of the use of a space-time control volume, the right side of the discretized equation, evaluated at t, can be considered representative of the entire timestep. This avoids the need to assume a profile in time (i.e. piecewise constant for fully-implicit and fully-explicit and piecewise linear for Crank-Nicholson). This makes coding substantially simpler by avoiding storage of old flux values. Using this method, the interpolation comes when computing the time face values. If we assume piecewise constant, we get a first-order scheme. If we assume piecewise linear, we get a second order scheme.

To discretize using this method, we integrate the governing equation over the space-time control volume:

$$\int_{t-\Delta t/2}^{t+\Delta t/2} \int_V \frac{\partial \left(\rho c_p T\right)}{\partial t} dt dV = \int_{t-\Delta t/2}^{t+\Delta t/2} \int_V k \nabla^2 T dt dV + \int_{t-\Delta t/2}^{t+\Delta t/2} \int_V S dt dV$$

Carrying out the integration:

$$\left(\rho c_p T_P V_P\right)^{t+\Delta t/2} - \left(\rho c_p T_P V_P\right)^{t-\Delta t/2} = \left[F_w^d - F_e^d\right]^t \Delta t + S_P^t \Delta t V_P$$

Dividing through by Δt, expressing diffusive fluxes in terms of D_w and D_e, and dropping superscripts t, we have

$$\frac{\left(\rho c_p T_P V_P\right)^{t+\Delta t/2} - \left(\rho c_p T_P V_P\right)^{t-\Delta t/2}}{\Delta t} = -D_w \left(T_P - T_W\right) + D_e \left(T_E - T_P\right) + S_P V_P$$

We must then specify the values on the left side of the equation for the times $t - \Delta t/2$ and $t + \Delta t/2$

A first-order time integration scheme is obtained by interpolating the time face values assuming a piecewise constant distribution over each timestep, which is given as

$$T_P^{t-\Delta t/2} = T_P^{t-\Delta t}$$

$$T_P^{t+\Delta t/2} = T_P^t$$

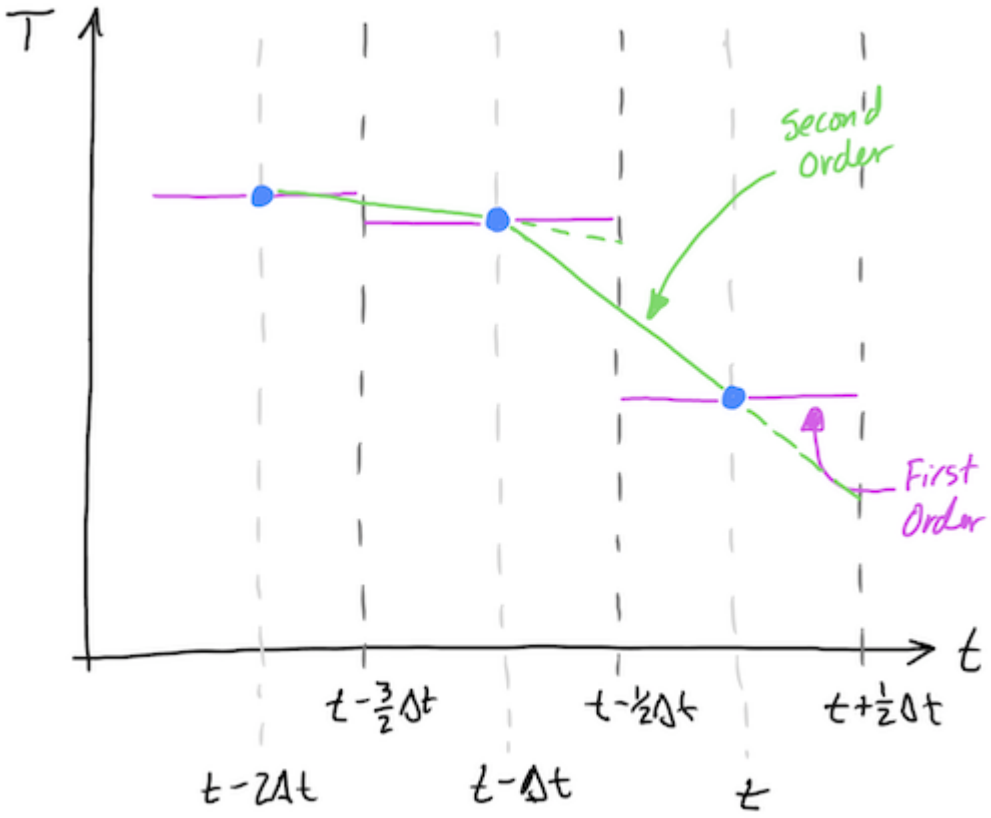
This time interpolation scheme is analogous to an upwind scheme for spatial interpolations (more on this later).

A second-order time integration scheme is obtained by interpolating the time face values by assuming a piecewise linear distribution between time values. This results in

$$T_P^{t-\Delta t/2} = T_P^{t-\Delta t} + \frac{1}{2}\left(T_P^{t-\Delta t} - T_P^{t-2\Delta t}\right)$$

$$T_P^{t+\Delta t/2} = T_P^t + \frac{1}{2}\left(T_P^t - T_P^{t-\Delta t}\right)$$

In this case, the slope is determined from the backwards direction, and this is used to interpolate to the forward face. This is shown schematically in the plot below:



Substituting these interpolated values into the integrated governing equation, for the first order scheme results in:

$$\frac{\left(\rho c_p T_P V_P\right)^{t+\Delta t/2} - \left(\rho c_p T_P V_P\right)^{t-\Delta t/2}}{\Delta t} = \rho c_p V_P \frac{T_P - T_P^o}{\Delta t}$$

where again, the superscript is dropped for current time values (i.e. at time t) and the superscript o is used for "old" time values (i.e. at time $t - \Delta t$). It can be readily seen that this is the same result as the fully implicit scheme derived previously. The advantage of this method comes when deriving the second order scheme, which is given as follows:

$$\frac{\left(\rho c_p T_P V_P\right)^{t+\Delta t/2} - \left(\rho c_p T_P V_P\right)^{t-\Delta t/2}}{\Delta t}$$
$$= \rho c_p V_P \frac{T_P + \frac{1}{2}\left(T_P - T_P^o\right) - T_P^o - \frac{1}{2}\left(T_P^o - T_P^{oo}\right)}{\Delta t}$$

where the superscript oo is used for the time value $t - 2\Delta t$. Simplifying the equation above results in:

$$\frac{\left(\rho c_p T_P V_P\right)^{t+\Delta t/2} - \left(\rho c_p T_P V_P\right)^{t-\Delta t/2}}{\Delta t} = \rho c_p V_P \frac{\frac{3}{2}T_P - 2T_P^o + \frac{1}{2}T_P^{oo}}{\Delta t}$$

This is a second-order fully implicit scheme. In contrast to the Crank-Nicholson scheme, flux values at previous timesteps do not need to be solved. Only the temperature values at the previous two timesteps need to be retained.

Other Transient Discretization Schemes

In addition to the transient discretization schemes considered above, there are several others that are occasionally used (we have discussed the most common). Some other schemes that can be higher than second order include:

- Adams-Bashforth (explicit)
- Adams-Moulton (implicit)
- Runge-Kutta (implicit or explicit)

These methods will not be discussed further in this course, but may be of interest when higher temporal accuracy is required.

Linearization

Recall from the previous lesson, that the cell residual for steady conduction was

$$r_P = D_w\left(T_P - T_W\right) - D_e\left(T_E - T_P\right) - S_P V_P$$

where

$$D_e = \frac{kA_e}{\Delta x_{PE}}$$

$$D_w = \frac{kA_w}{\Delta x_{WP}}$$

Adding the first order implicit transient term results in:

$$r_P = \rho c_p V_P \frac{T_P - T_P^o}{\Delta t} + D_w\left(T_P - T_W\right) - D_e\left(T_E - T_P\right) - S_P V_P$$

The linearization coefficients are therefore:

$$a_P = \frac{\partial r_P}{\partial T_P} = \frac{\rho c_p V_P}{\Delta t} + D_w + D_e - \frac{\partial S_P}{\partial T_P} V_P$$

$$a_W = \frac{\partial r_P}{\partial T_W} = -D_w$$

$$a_E = \frac{\partial r_P}{\partial T_E} - D_e$$

As before, we can form an algebraic system of equatons at each control volume, which is given as

$$a_P \delta T_P + a_W \delta T_W + a_E \delta T_E = -r_P$$

For the second order implicit temporal scheme, the a_P coefficient is modified to:

$$a_P = \frac{\partial r_P}{\partial T_P} = \frac{3}{2}\frac{\rho c_p V_P}{\Delta t} + D_w + D_e - \frac{\partial S_P}{\partial T_P} V_P$$

Exercise: Derive the linearized equations for the Crank-Nicholson transient discretization scheme.

Implementation

In order to implement the transient terms into our solution procedure, we begin by copying some of the code from the previous lesson into separate Python files. This allows us to simply import the necessary classes that were written previously without clutter this notebook.

These Python files are located in a directory called `Lesson3` to keep them separate from ones that are used in later lessons, since some of the files might need to be modified. Inside of the directory `Lesson3` there is an empty file called `__init__.py`. This simply indicated that this directory is to be considered as a Python package. This file can also contain code to help load differnt modules, but that is not required for this purpose. Each file within the package then becomes a module, which can contain different classes, functions, etc.

The modules included in the `Lesson3` package are:

- Grid: Contains the class `Grid`
- ScalarCoeffs: Contains the class `ScalarCoeffs`
- BoundaryConditions: Contains the classes `BoundaryLocation`, `DirichletBc`, `NeumannBc`.
- Models: Contains the classes `DiffusionModel`, `SurfaceConvectionModel`
- LinearSolver: Contains the function `solve`

In order to add the transient term, we add a class called `FirstOrderTransientModel`, which is partially shown below. You will need to complete this code. Note that the class interface is similar to those for the diffusion model and surface convection model. This makes the code easier to use. We can also create classes for other transient models, such as second order implicit or Crank-Nicholson.

Exercise: Complete the code for the `FirstOrderTransientModel` class.

In [ ]:
```
class FirstOrderTransientModel:
    """Class defining a first order implicit transient model"""

    def __init__(self, grid, T, Told, rho, cp, dt):
        """Constructor"""
        self._grid = grid
        self._T = T
        self._Told = Told
        self._rho = rho
        self._cp = cp
        self._dt = dt

    def add(self, coeffs):
        """Function to add transient term to coefficient arrays"""

        # Calculate the transient term

        # Calculate the linearization coefficient

        # Add to coefficient arrays

        return coeffs
```

The code below shows an example of how to solve a transient problem. The problem setup is similar to the one-dimensional steady conduction problem with external convection and Dirichlet/Neumann boundary conditions from Lesson 2. Currently, the transient code does not do anything, so the solution will be the same as the steady solution.

Exercise: After you have completed the code for the `FirstOrderTransientModel` class, solve the problem below. Consider the effects of the timestep size, the number of timesteps, and the thermophysical properties on the solution results.

In [ ]:
```
from Lesson3.Grid import Grid
from Lesson3.ScalarCoeffs import ScalarCoeffs
from Lesson3.BoundaryConditions import BoundaryLocation, DirichletBc, Neum
from Lesson3.Models import DiffusionModel, SurfaceConvectionModel
from Lesson3.LinearSolver import solve

import numpy as np
from numpy.linalg import norm

# Define the grid
lx = 1.0
ly = 0.1
lz = 0.1
ncv = 10
grid = Grid(lx, ly, lz, ncv)

# Set the timestep information
nTime = 10
dt = 1
time = 0

# Set the maximum number of iterations and convergence criterion
maxIter = 10
converged = 1e-6

# Define thermophysical properties
rho = 1000
cp = 1000
k = 100

# Define convection parameters
ho = 25
To = 200

# Define the coefficients
coeffs = ScalarCoeffs(grid.ncv)

# Initial conditions
T0 = 300

# Initialize field variable arrays
T = T0*np.ones(grid.ncv+2)

# Define boundary conditions
west_bc = DirichletBc(T, grid, 400, BoundaryLocation.WEST)
east_bc = NeumannBc(T, grid, 0, BoundaryLocation.EAST)

# Apply boundary conditions
west_bc.apply()
east_bc.apply()

# Create list to store the solutions at each timestep
# Note: If there are a lot of timesteps, this will cost a
#       lot of memory. It is suggested to set a parameter to
#       only store the solution every N timesteps.
T_solns = [np.copy(T)]
```

```
# Define the transient model
Told = np.copy(T)
transient = FirstOrderTransientModel(grid, T, Told, rho, cp, dt)

# Define the diffusion model
diffusion = DiffusionModel(grid, T, k, west_bc, east_bc)

# Define the surface convection model
surfaceConvection = SurfaceConvectionModel(grid, T, ho, To)

# Loop through all timesteps
for tStep in range(nTime):
    # Update the time information
    time += dt

    # Print the timestep information
    print("Timestep = {}; Time = {}".format(tStep, time))

    # Store the "old" temperature field
    # Note: do not use copy here because that creates a new object
    #       and we want the reference in the transient model to remain
    #       valid
    Told[:] = T[:]

    # Iterate until the solution is converged
    for i in range(maxIter):
        # Zero the coefficients and add each influence
        coeffs.zero()
        coeffs = diffusion.add(coeffs)
        coeffs = surfaceConvection.add(coeffs)

        # Compute residual and check for convergence
        maxResid = norm(coeffs.rP, np.inf)
        avgResid = np.mean(np.absolute(coeffs.rP))
        print("Iteration = {}; Max. Resid. = {}; Avg. Resid. = {}".format(
        if maxResid < converged:
            break

        # Solve the sparse matrix system
        dT = solve(coeffs)

        # Update the solution and boundary conditions
        T[1:-1] += dT
        west_bc.apply()
        east_bc.apply()

    # Store the solution
    T_solns.append(np.copy(T))
```

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt

i = 0
for T in T_solns:
    plt.plot(grid.xP, T, label=str(i))
    i += 1

plt.xlabel("x")
plt.ylabel("T")
plt.legend()
plt.show()
```

Next Steps

Exercise: For the problem above, implement a class called `SecondOrderTransientModel` that implements the second order implicit method derived in this lesson. Find a timestep that allows you to see the temporal variation as the solution goes from the initial condition to the steady state. Compare the results with the first order implict model.

Now that you have completed the lesson on transient diffusion, you are ready to move on the next lesson on Convection of a Scalar (4-ConvectionOfScalar.ipynb) where we will implement more code to solve this type of problem.

In [ ]:
```

```