The complete code, shown below, can also be found on [Github]()

```
import numpy as np
from scipy.integrate import quadrature as integrate
from shape_functions import shape_functions
from jacobian import jacobian

#
# A: area
# k: hydraulic conductivity
# l: length
# alpha: alpha term associated with pipe permeability
# num_elements: number of elements
# num_element_nodes: number of nodes per element
# bc_essential: ( dict ) essential boundary conditions
# bc_natural: ( dict ) natural boundary conditions
#
def solve_fe ( A, k, l, alpha, num_elements, num_element_nodes, bc_essential, bc_natura

    # Calculate number of nodes
    num_nodes = num_elements * ( num_element_nodes - 1 ) + 1

    # Get x-coordinate for each node
    x_coord = [ float(i) * ( float(l) / float( num_nodes - 1 ) ) for i in range( num_nc

    # Get the shape functions
    N, dN, xi = shape_functions( num_element_nodes )

    # Create IEN and ID functions
    def IEN ( element_number, local_node_number ):
        return ( num_element_nodes - 1 ) * element_number + local_node_number

    # Create global arrays
    K = np.zeros( ( num_nodes, num_nodes ) )
    M = np.zeros( ( num_nodes, num_nodes ) )
    F = np.zeros( ( num_nodes, 1 ) )

    # Element loop
    for element in range( num_elements ):

        # Create local matrices
        ke = np.zeros( ( num_element_nodes, num_element_nodes ) )
        me = np.zeros( ( num_element_nodes, num_element_nodes ) )
        fe = np.zeros( ( num_element_nodes, 1 ) )

        # Get globals
        nodes = [ IEN( element, i ) for i in range( num_element_nodes ) ]
        x = [ x_coord[ node ] for node in nodes ]

        # Coordinate transformation and jacobian for this element
        J = jacobian( x[ 0 ], x[ len( x ) - 1 ] )

        # Node loop
        for row in range( num_element_nodes ):

            # Check for natural boundary condition contribution
        if nodes[ row ] in bc_natural:

            fe[ row, 0 ] -= N[ row ]( xi[ row ] ) * bc_natural[ nodes[ row ] ] * A

        # Other node noop
        for col in range( num_element_nodes ):

            # Integrate to calculate ke term
            _k = integrate( lambda _xi: ( dN[row](_xi) / J )*( dN[col](_xi) / J ),
                            xi[0],
                            xi[len(xi)-1] )[ 0 ]

            _m = integrate( lambda _xi: N[ row ]( _xi ) * N[ col ]( _xi ),
                            xi[0],
                            xi[len(xi)-1] )[ 0 ]

            # Add the term to ke, converting to local coordinates
            ke[ row, col ] = _k * A * k * J
            me[ row, col ] = _m * alpha * J

    # Update global arrays with contributions from local
    for row in range( num_element_nodes ):

        # Get the global row
        r = IEN( element, row )

        # Add local f to global F
        F[ r, 0 ] += fe[ row, 0 ]

        for col in range( num_element_nodes ):

            # Get the global column
            c = IEN( element, col )

            # Add the local k to global K
            K[ r, c ] += ke[ row, col ]
            M[ r, c ] += me[ row, col ]

# Apply essential boundary conditions using elimination method
K_mask = np.ones( K.shape, dtype=bool )
F_mask = np.ones( F.shape, dtype=bool )
dof = num_nodes - len( bc_essential )
for node, value in bc_essential.iteritems():

    # Mask row and column in the K matrix and M matrix
    K_mask[ node, : ] = False
    K_mask[ :, node ] = False

    # Mask row in the F array
    F_mask[ node, : ] = False

    # Subtract from the F array
    for row in range( num_nodes ):

        _k = K[ row, node ]
        _m = M[ row, node ]
        F[ row, 0 ] -= _k * value
        F[ row, 0 ] += _m * value

# Apply the mask and reshape
K = np.reshape( K[ K_mask ], ( dof, dof ) )
M = np.reshape( M[ K_mask ], ( dof, dof ) )
F = np.reshape( F[ F_mask ], ( dof, 1 ) )
```

```python
    # Solve for unknowns
    d = np.linalg.solve( K-M, F )

    # Place essential boundary condition values into solution
    for node, value in bc_essential.iteritems():

        d = np.insert( d, node, value )

    return d, x_coord
```