Frontal polymerization MATLAB code documentation

Harshit Agarwal

Abstract

The document is prepared using LaTeX in an Elsevier Journal template.

Keywords:

1. Code description

The MATLAB code solves the frontal polymerization problem in axisymmetric co-ordinates. The domains are the wire at the center and the monomer microchannel surrounding the wire. The microchannel is surrounded by an inert polymer (PDMS) as seen in Fig. 1.

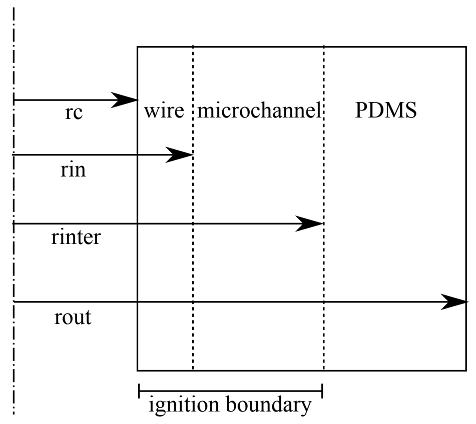

Figure 1: **The domain, the radii are indicated by variables used in code**

The numerical scheme implemented is as given in [1] and the cure kinetics are as given in [2].

The code takes (among other parameters) the following as input:

- Initial conditions
- Boundary conditions
- Material properties

- Time steps
- Time step size
- β value that decides if the scheme is explicit ($\beta = 0$) or implicit (second order for $\beta = 0.5$)
- Cure limit near ignition (curelimit)
- Intervals at which to record values (recordstep)
- Axial length
- Wire, microchannel and PDMS radius (as seen in Fig. 1)
- Number of elements in axial direction
- Number of elements in radial direction for wire and PDMS
- Geometric progression ratio (gpratio) for element length in PDMS.
- The ignition value q (in W/m^2)
- The resistive heating value Q (in W/m^3)

The code is such that the wire and microchannel are ignited from the bottom end to start the process. The code detects the average curing near the boundary at each time step. Once the cure reaches the limit set in the input file (curelimit), the ignition is switched off, after that the front self-propagates due to the heat released by polymerization.

The simulations sometime require large number of time steps (greater than 10,000). The temperature and cure fields in such case are recorded after certain time step intervals (defined by the recordstep variable in the input file). This spaced out recording is implemented so that the data files have a small disk space.

The code has a feature that if the cure values becomes complex, then the time marching is terminated. This is accomplished by checking the cure for complex values at each time step.

Since PDMS doesn't have much temperature variation, its element length in radial direction follows a geometric progression with element length increasing with increasing radius.

The code has provision for resistively heating the wire, the heating intensity (W/m^3) can be selected, the code detects the curing near the wire-monomer interface and switches off the resistive heating once the curing reaches the limit set in the input file (curelimit). Both ignition and resistive heating can be done together.

There is a restart file to add to the time marching, the file basically has the time marching loop copied.

2. Post Processing

The temperature and cure fields are stored as vectors. The ‘vectortomatrix-convert.m’ file converts these vectors to matrices. The temperature and cure fields can then be viewed using the ‘surf’ command at any time step.

The front position (FP) and velocity (FV) are the two important parameters of interest. They are calculated using the ‘postprocessor.m’ file.

2.1. Front position and velocity calculation

The file takes the radial node from the centre as input and outputs FP and FV along the axial line passing through the radial node.

The temperature derivative is calculated along each node on the chosen axial line (by specifying the radialnode variable in the file), and at all time steps. The position of the maxima of the temperature derivative at a given time step is taken as the front position. It is observed that as the front progresses, the derivative shows a distinct maxima. The front velocity is calculated by numerically differentiating the front position w.r.t. time.

3. Example

The procedure is:

1. Run inputfile_x.m (x = a,b,c)
2. Run unsteadysolver.m
3. Run vectortomatrixconvert.m
4. Run unsteadyplotanimated.m

unsteadyplotanimated.m code with slight modifications can be used to visualize different results, the quantities of interest are Tm and alpham. They contain the Temperature and Cure fields respectively at all time steps.

3.1. Example a: Ignition and resistive heating

See inputfile_a.m

3.2. Example b: Resistive heating only

See inputfile_b.m. It is to be noted that a very small value of ignition has been used to deactivate the constant temperature boundary condition.

3.3. Example c: Ignition only

See inputfile_c.m

4. For the developer

Most of the code is written in a fairly straightforward manner. A lot of helper functions have been used to calculate matrices such as C_c, C_T and K_T, and the vector P_c. The matrices are calculated element by element and then assembled.

The tricky part in the code are the boundary conditions (BCs). The code detects the state of cure near the ignition and switches off the ignition. This is basically accomplished by the changing the BCs. So matrices pertaining to both the pre and post-ignition BCs are stored and used by the code.

5. Drawbacks in the code

The code is not written in a modular fashion, it is not straightforward to apply more complex boundary conditions or have a more complex domain. The code was originally written for the explicit scheme, then modified to include the implicit scheme. After the modification, the code has not been tested for the explicit scheme. A better choice for variable names could have been made.

References

[1] Zhu, Q., Geubelle, P.H., Li, M., Tucker, C.L.. Dimensional accuracy of thermoset composites: simulation of process-induced residual stresses. Journal of composite materials 2001;35(24):2171–2205.

[2] Frulloni, E., Salinas, M., Torre, L., Mariani, A., Kenny, J.M.. Numerical modeling and experimental study of the frontal polymerization of the diglycidyl ether of bisphenol a/diethylenetriamine epoxy system. Journal of applied polymer science 2005;96(5):1756–1766.