Proceedings of Conference On Insights in mechanical Engineering (IME-19)
In Association with Novateur Publications IJIERT-ISSN No: 2394-3696
ISBN. No. 978-93-87901-04-9
March, 1st March 2019

Paper ID HP 06

THE REVIEW OF INJECTION MOLDING PROCESS SIMULATION USING OPEN FOAM SOFTWARE

Vishal A. Meshram,
Department of Mechanical Engineering, Indira College of Engineering & Management, Pune, India
* vishal.meshram@indiraicem.ac.in

Deepti Deshmukh,
Department of Mechanical Engineering, Indira College of Engineering & Management, Pune, India
deepti.deshmukh@indiraicem.ac.in

ABSTRACT
To solve the Injection molding case in OpenFOAM CFD tool, the code has been developed by Kristjan Krebelj and Janez Turk. Sample geometry and numerical methods has taken form the research paper to demonstrate using Open FOAM solver. This paper presented the tutorial example "open InjMoldSim" in stepwise manner to use this solver. This solver was modification of "compressible Inter Foam" which distributed with OpenFOAM package. The modified solver used for simulation of Injection molding filling, packing and cooling stages. Some test code also been given for the viscosity in the model adhere to the Cross-WLF model, The heat transfer code for solution of the heat diffusion equation, Solid shear modulus value of the solid phase and Equation of state for finding specific volume of melt.

Keywords: Injection molding, OpenFOAM, tutorial.

Introduction

This was an Open FOAM solver for simulation of injection molding filling, packing and cooling stages. It was a modification of the "compressibleInterFoam" solver distributed with OpenFOAM. The simulation was prohibitively inconvenient for typical industrial use. To make aware about this solver, the tutorial has presented in this paper. This was available online on jupyter notebook, supplied with openInjMoldSim. The demo example was presented here become starting point for developing injection molding simulations, i.e. changing the mesh and process parameters. It also presented an experimental validation of the predicted cavity pressure evolutions. The test examples was intended to numerically validate some important aspects of the solver. An experimental investigation [1] of the injection molding process was modeled. This simulation case was also described in a conference paper[2]. For a description of a similar experimental investigation with HDPE [3], has in the research paper. The HDPE investigation was not modeled in this repository.

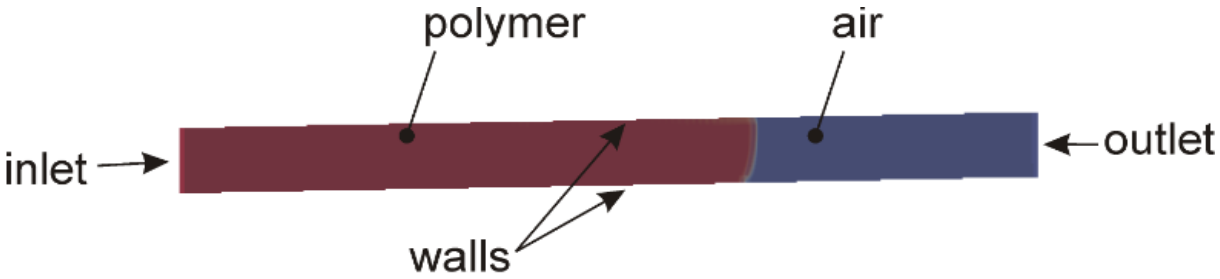

Figure 1: Figure represent various boundaries & material of numerical model.

Steps of tutorial

To solve this tutorial an OpenFOAM installation was required to run the code. This code was run on Linux Ubuntu. The master case openInjMoldSim can download from GitHub of reference [4].
Run Allwmake in the solver directory to compile the solver.
Run AllRun in the fill_pack directory to run the example simulation. Then run paraFoam in the case directory to view the results.

1) Geometry

The filling position in the model was not P0 (the feed system) but the P1 position which was in the cavity. This makes the domain simpler and excludes the modelling of the pressure drop at the gate, eliminating a possible source of uncertainty [4].

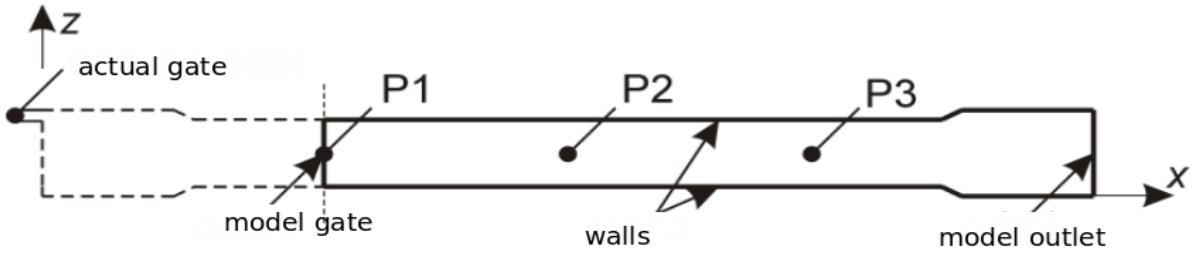

Figure 2: 2D model of mold cavity

The model uses a 2D description of the actual cavity and also assumes a uniform cavity thickness. Note, that the above image was schematic and the actual cavity was much more elongated. This allowed to create the mesh using the "blockMesh" utility with the simplest setup.

2) Changing the geometry

To use a different geometry, one way to use a general mesh was to import an Ansys Fluent mesh or modify the constant/polyMesh/blockMeshDict file. Note, not using the blockMesh utillity required removing the call to blockMesh from the AllRun script. An ./AllMesh call could be made from the AllRun script, where an AllMesh script could include a call to fluentMeshToFoam. Applying openInjMoldSim in practice might be feasible for an axi-symmetric model.

3) Setup

a) Material parameters

In the case openInjMoldSim, the polymer and the air are the material parameter. For the material parameters of either see the respective thermophysicalProperties files in the constant directory. Note, that the air viscosity was not realistic and neither was the air venting. The viscosity of the air was artificially increased to avoid the possible occurrence of turbulence and related convergence issues. The air viscosity was still set far lower than the viscosity of the polymer. The polymer thermal conductivity was described by the kappaTable and the specific heat by the cpTable (basic SI units are used with temperature in kelvin). The cpTable could have been supplied as pressure dependent by including data points for pressures other than "0", but this would only make sense in modeling a semi-crystalline polymer with lumped latent heat released at higher crystallization temperatures in case of higher pressure. The syntax for two (0 and 100 MPa -- probably not enough) pressure data points would have been:

```
(
(273.15((0 1377.715)(1e8 1477.715)))
(323.15((0 1482.72013)(1e8 1582.72013)))
(328.15((0 1493.23916)(1e8 1593.23916)))
.....
(390.15((0 1967.39298)(1e8 2067.39298)))
(392.15((0 1971.60316)(1e8 2071.60316)))
(623.15((0 2456.715)(1e8 2556.715)))
)
```

b) Boundary conditions

Boundary conditions was set together with the initial conditions in the "org0" directory. The thermal boundary conditions account for the "heat transfer coefficient" with the external Wall Heat Flux Temperature keyword. See the "org0/T" file. The keyword h sets the value of the heat transfer coefficient. The keyword Ta was the mold temperature. The keyword kappa was required for the boundary condition to obtain the material thermal conductivity (this way the temperature gradient can be used to calculate the heat flux). Since the solver uses a non-constant thermal conductivity (dependent on temperature and pressure) it makes this value available to the boundary condition by writing the field mojKappaOut. This field was not written to the results time folder because it was of little value for the post processing.

The inlet pressure at P1 was set to match the experiment by calling the external file "fim_01_01_F_00_00_01_08.txt" from "org0/p_rgh".

```
inlet
  {
    type uniformFixedValue;
    uniformValue table
      # include "$FOAM_CASE/fim_01_01_F_00_00_01_08.txt"
```

}

4) Procedure

The simulation procedure was set in the AllRun script. Venting the air out of the cavity requires designating an outlet for the filling phase and closing it for the packing phase.

Initial files are first set.

```
# Initial files
cp -r org0 0
cp system/controlDict0 system/controlDict
cp system/fvSolution0 system/fvSolution
blockMesh
setFields
```

This copies the initial and boundary conditions from "org0" to the "0" directory (where they may be modified by the code). The AllRun script runs the solver multiple times with different settings (e.g. each time extending the simulation time). Therefore, controlDict0 and fvSolution0 was the initial non-modified versions. blockMesh creates the mesh. setFields puts a small amount of material at the gate to make the simulation start better conditioned and prevent high injection velocities due to small inertia of the air-filled cavity.

The following runs the filling phase with the outlet open and renames the log file corresponding to the filling phase.

```
# fill
phase='_fill'
runApplication $application
mv 'log.'$application 'log.'$application$phase
```

The following code seals the outlet and runs the first portion of the packing phase.

```
# pack1
phase='_pack1'
sed -i -e 's/^deltaT\s.*;/deltaT 1e-10;/g' 0.135/uniform/time # change init deltaT
changeDictionary -latestTime -dict system/changeDictionaryDict.bcpack
changeDictionary -instance system -dict system/changeDictionaryDict.ctrl1
runApplication $application
mv 'log.'$application 'log.'$application$phase
```

The change in the boundary conditions implies a very sudden change in the solution development which was resolved by reducing the time step with the sed replacement in the last results dictionary. The keyword deltaT from the system/controlDict was ignored in an analysis starting from the latestTime and the initial time step in a restart analysis was read from the 0.135/uniform/time file. Using sed inside of AllRun was indeed cumbersome, since care must be taken to match the directory name.

RESULTS

The evolution of the fields can be viewed by calling paraFoam from within the case directory.

1) Filling progress

The alpha.poly field describes the phase distribution. If its value is 1, it means the space was occupied by the polymer. If it's 0, it's occupied by the air.

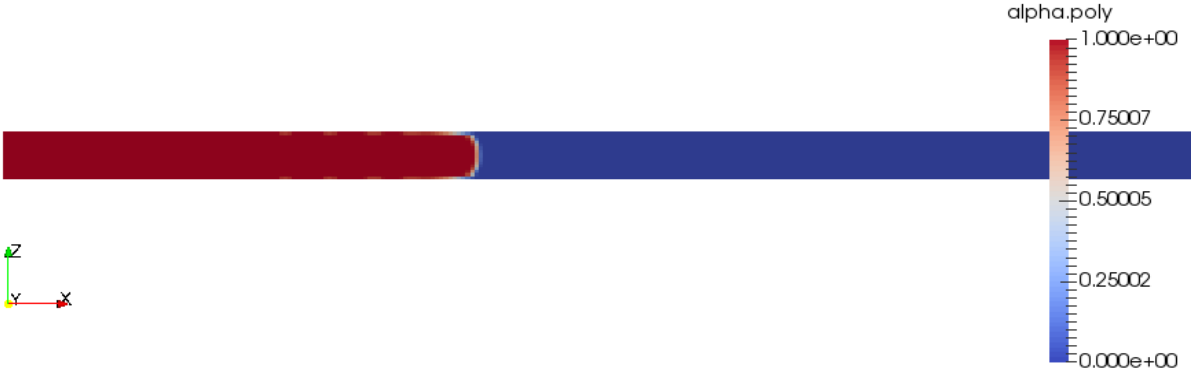

Figure 3: Contours of filling progress of alpha.poly

2) Filling pressure

Figure 4: Contours of filling pressure in mold model

3) Packing pressure

Figure 5: Contours of packing pressure at time 1.1 sec.

4) Packing temperature

t=1.1s

Figure 6: Contours of Packing Temperature at time 1.1 sec.

5) Pressure evolution

A steeper than measured pressure decrease during packing was expected due to the assumption of the rigid mold walls [2] as published in paper.

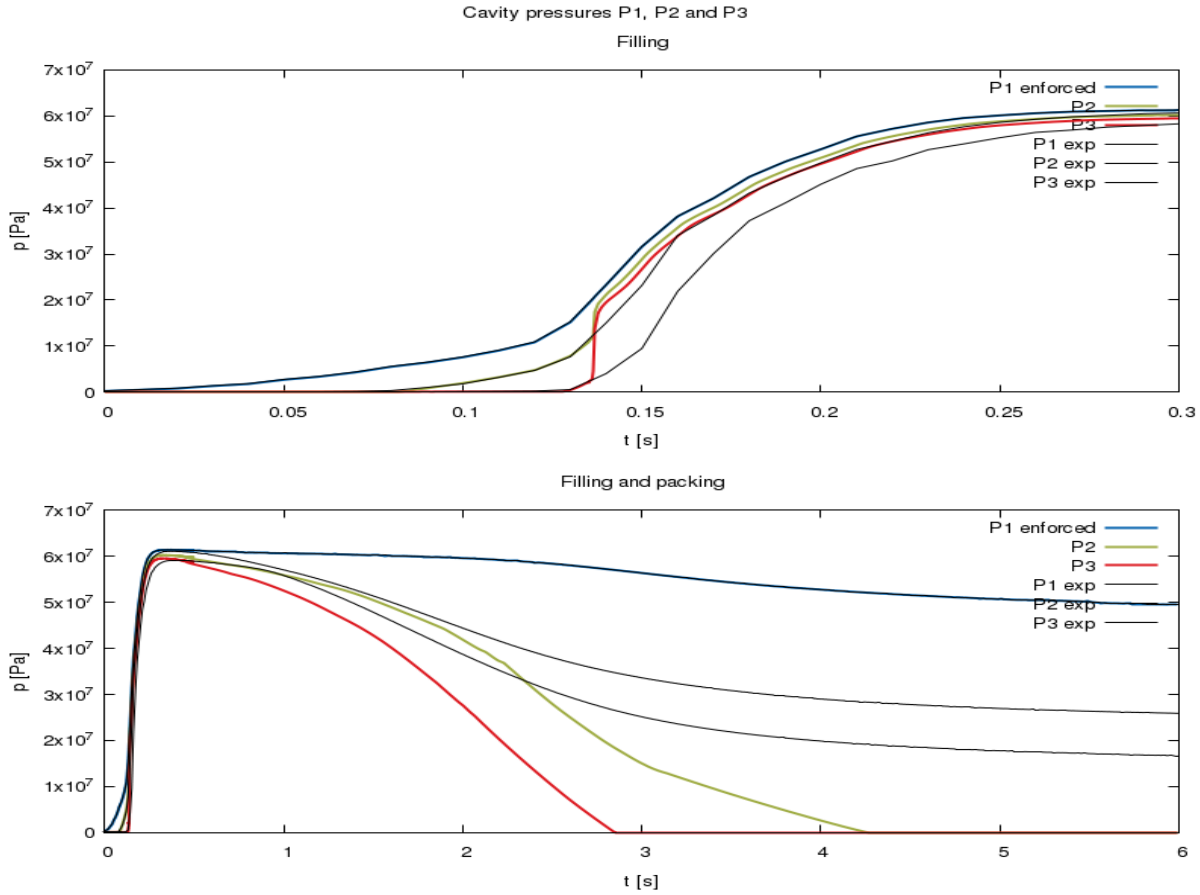

Figure 7: Validation of Pressure results with the experimental values [2].

Testing module

Some numerical validations was included.

1) **Viscosity** -- crossWLF: The viscosity in the model must adhere to the Cross-WLF model.
2) **Heat transfer** -- heatTranser: The heat transfer in the model mush be a proper solution of the heat diffusion equation.
3) **Solid shear modulus value** -- shearModulus: The value of the shear modulus of the solid phase.
4) **Equation of state** -- taitEq: The specific volume of the melt.

1) Viscosity: crossWLF

The viscosity in the model was calculated as if it were measured in an experiment and this value was verified to match the Cross-WLF model value, calculated in the supplied python script. The geometry was modified to a cuboid cavity to make the shear rate field constant in the whole domain (i.e. homogenious). An additional library mojLibForces (provided in the solver directory) needs to be compiled to enable the force calculation.

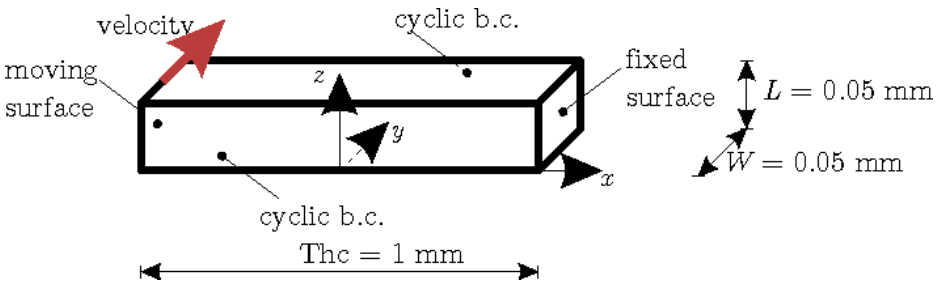

Figure 8: Geometry for viscosity FVM model [5]

a) Analysis

The force on the top face of the cuboid is the product of the constant shear stress ττ and the surface area A0:

$$F=A0\cdot\tau=A0\cdot\dot{\gamma}\,\eta=A0\cdot U/Thc\,\eta. \quad \text{--- (equation)}$$

Whereas, $\dot{\gamma}$ – shear rate,
η – Viscosity
U – Velocity
Thc – thickness

b) Shear rate field

The shear rate field was practically homogeneous and approximately 100.0 1/s.

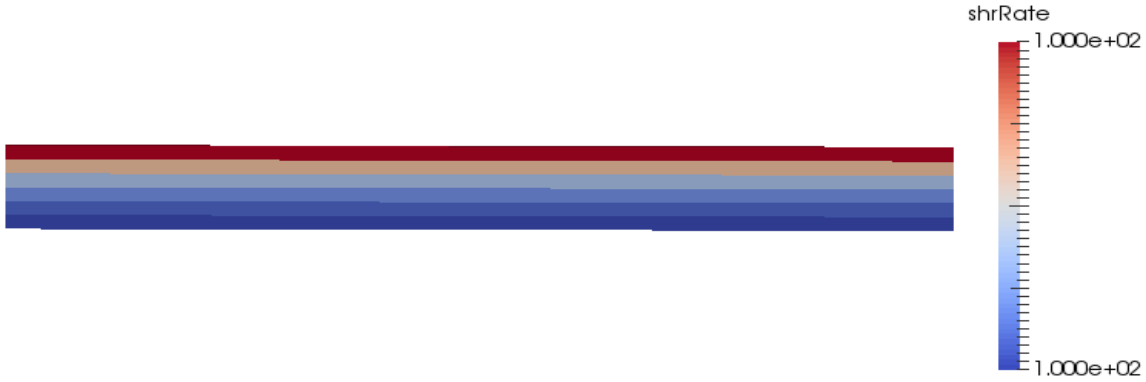

Figure 9: Contours of shear rate was homogeneous

c) Domain

The solution does not change along y. The domain could be made smaller by making the length L shorter.

2) Heat transfer: heatTransfer

A test of the energy equation solution was performed. The case was modified to match a 1D heat transfer case of a solid which was simulated using the finite element method. A more elegant way of doing this would be to calculate the temperature field evolution in a python script. The temperature evolutions in the wall center are compared. This example does not consider the possibility where density non-homogeneity would introduce a discrepancy.

a) Temperature field

Below was the temperature field at t = 4 s in Celsius. The material was being heated through the left- and right-hand side faces where contact to a hot mold was assumed.

Figure 10: Contours of Temperature field at time 4 sec.

3) Solid shear modulus vaule: shearModulus

Stress in the solid develops with the deformation of the solid. This test validates that the shearModulus setting in the "solidificationProperties" dictionary implies the value of the calculated stress in the solid.

a) Procedure

The simulation was composed of two openInjMoldSim calls.

The solidificationProperties was modified in between so the initial file was being kept under a different name. The controlDict likewise.

```
# Initial files
cp -r org0 0
cp system/controlDict0 system/controlDict
cp constant/solidificationProperties0 constant/solidificationProperties
blockMesh
```

During the first run no solid stress was present because the "solidificationProperties" prohibit it with value of the shrRateLimEl keyword. This could also be achieved by a very high value for viscLimEl. The solidificationProperties dictionary in the first run reads:

shearModulus 600e6;//Pa

shrRateLimEl -1.0;//1/s //elasticity appears at lower shear rates than this

viscLimEl -1.0;//1/s //elasticity appears at higher viscosities than this

The first run establishes a simple shearing field -- like in the crossWLF case.

The Allrun script then changes the shrRateLimEl keyword.

changeDictionary -instance constant -dict system/changeDictionaryDict.cons

The "system/changeDictionaryDict.cons" dictionary contains the following (to allow the elastic stress to develop):

```
dictionaryReplacement
{
  solidificationProperties
  {
    shrRateLimEl 1e10;//1/s
  }
}
```

b) Analysis

The second run with the development of the elastic stress simulates Δt =10 - 6Δt =10 - 6 s of material shearing at the shearing rate of γ̇ =100 γ̇ =100 1/s. The affected strain tensor component at the end was expected to be

τ = Gγ̇ Δt = 600·106·100·10 – 6 = 60τ = Gγ̇ Δt=600·106·100·10 – 6 = 60 kPa

This matches the calculated value inspected with paraView.

Figure 11: Contours of shear rate modulus.

c) Limitation

This test, unfortunately, does not validate whether the stress correctly acts on the material. Such a validation was not available with the same procedure as used in the case of crossWLF viscosity because of the way the wall force was calculated in the crossWLF case (view the mojForcesLib library in the solver directory). On the other hand, it can be verified that the solid stress does stop the material flow in the packing phase which was a strong indication that it correctly acts on the material (solid stress can be eliminated in the demo case by deleting or renaming the constant/solidificatinoProperties dictionary).

d) The use of the solidificationProperties **dictionary**

Elastic shear modulus was effected when the viscosity surpasses the viscosity threshold viscLimEl specified in the solidificationProperties dictionary. To control this according to temperature, change the TnoFlow temperature of the CrossWLF model in the constant/thermophysicalProperties.poly dictionary. Mind the etaMax value to be greater than viscLimEl to have it surpass the threshold. According to experience, elastic behavior should be limited to slow flow. This was effected with the shearRate limit.

4) Equation of state: taitEq

The specific volume in the model was verified to follow the tait equation. The material was fully contained and heated. Temperature and pressure both increase but density can only change due to redistribution of mass since the total mass and volume remain constant.

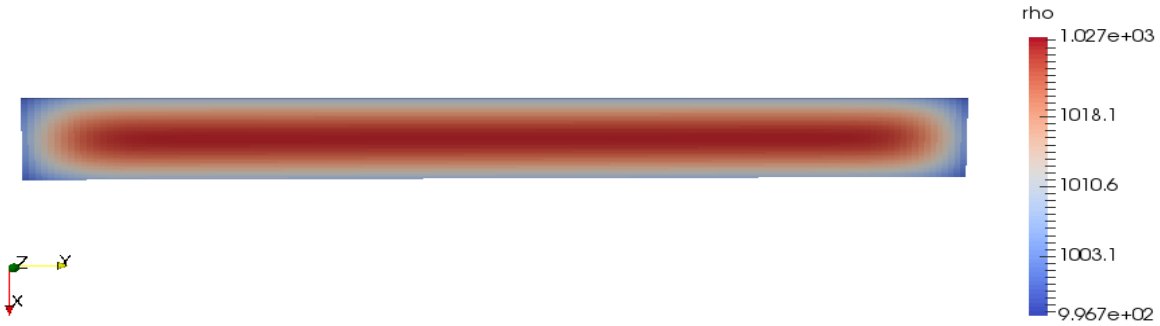

Figure 12: Contours of density field at time 1 sec.

ACKNOWLEDGEMENT
Author of this paper was thankful to **Kristjan Krebelj** for assembled & tested this code and **Janez Turk** for introducing OpenFOAM library.

CONCLUSION
This tutorial was used Compressible, non-isothermal, laminar cavity flow of injection mold of multiphase polymer & air. tait equation of state given temperature & pressure change in cavity volume, but density remain constant. Cross-WLF viscosity model given viscosity nature homogeneous for mold flow. Specific heat and thermal conductivity may depend on temperature and pressure in heat transfer test module. This model given elastic stress in the solid phase. Further modification and testing of code may help to become best suited for industrial use.

REFERENCES

[1] K. Krebelj, N. Mole and B. Štok, "Three-dimensional modeling of the stress evolution in injection," The International Journal of Advanced Manufacturing Technology, vol. 90, no. 5–8, p. 2363–2376, May 2017.

[2] N. Mole, K. Krebelj and B. Štok, "Validation of a numerical model for injection molding," in Kuhljevi dnevi 2017, Dobrna, 28 – 29. september 2017.

[3] N. Mole, K. Krebelj and B. Štok, "Injection molding simulation with solid semi-crystalline polymer mechanical behavior for ejection analysis," The International Journal of Advanced Manufacturing Technology, vol. 93, no. 9–12, p. 4111–4124, December 2017.

[4] K. Krebelj and J. Turk, "Open source injection molding simulation. A solver for OpenFOAM.," [Online]. Available: https://github.com/krebeljk/openInjMoldSim. [Accessed 2019].

[5] K. Krebelj, "openInjMoldSim tutorials," [Online]. Available: https://nbviewer.jupyter.org/github/krebeljk/openInjMoldSim/blob/master/tutorials/Tutorials.ipynb. [Accessed 2019].