

# BioFlow - README

Aleix Boquet Pujadas

## 1 Description

This Icy plug-in offers a tool capable of extracting internal cell pressure, velocity and forces from 2D and 3D image data sequences. The full method can be found in the paper[1] and the accompanying Supplementary Material. Briefly, it consists in tracking the cell over time while minimising a given functional describing its motion constrained to the weak formulation of a customised Stokes model (using a Finite Element Method and a variational data assimilation framework). That is, solving

$$\vec{u}, p, \vec{f}, \vec{g}, r = \operatorname{argmin} \left( J_{\text{data}}(\vec{u}\Delta t) + J_{\text{reg}}(\vec{f}, \vec{g}, r) \right), \text{ subject to } A(\vec{u}, p, \vec{f}, \vec{g}, r) = 0 \quad (1)$$

$$\begin{cases} J_{\text{data}}(\vec{u}\Delta t) &= \int_{\Omega} (\nabla I_2 \cdot \vec{u}\Delta t + (I_2 - I_1))^2 d\Omega, \\ J_{\text{reg}}(\vec{f}, \vec{g}, r) &= \alpha \int_{\Omega} \|\vec{f}\|^2 d\Omega + \gamma \int_{\Gamma} \|\nabla_{\Gamma} \vec{g}\|^2 d\Gamma + \eta \int_{\Omega} \|\nabla r\|^2 d\Omega, \end{cases}$$

where the model (in this case Stokes) is expressed in  $A$ :

$$A(\vec{u}, p, \vec{f}, \vec{g}, r) = \begin{cases} \nabla p - \mu \nabla^2 \vec{u} - \vec{f} & \text{in } \Omega \\ \nabla \cdot \vec{u} - r & \text{in } \Omega \\ \vec{u} - \vec{g} & \text{on } \Gamma. \end{cases} \quad (2)$$

In fact the equations are solved in a non-dimensional form and a multi-scale approach is used for the functional  $J_{\text{data}}$ . All of this is covered in detail in the Supplementary Material.

## 2 Introduction

After loading a video-microscopy sequence into the software, an initial contour is defined around the cell on the first frame (this manual step can be automated using any available segmentation technique). The main BioFlow script (written in Jython) is launched from Icy's script editor and starts by segmenting and tracking the cell boundary over time using an active contour method, then creates a multi-scale version of the image and contour for each frame. The contour-image pairs are subsequently transferred to a python script running the FEniCS Finite Element software, where the remainder of the computations take place: (a) the image and contour are converted into a Finite Element mesh using the Mshr (mesh generation) module; (b) the PDE system in 3D or 2D is then solved using the MUMPS solver; (c) finally, the functional  $J$  is minimised using the dolfin-adjoint module. This process is iterated until convergence and repeated from coarse to fine image scales using appropriate warping. To reduce computation time, each pair of frames in the video sequence is processed in parallel using all available processors on the workstation. The final results and figures were produced either within Icy or with the ParaView software. Notice that the results will depend on the accuracy of the model, in our case we are observing small filaments suspended within the cytoplasm and therefore our images do move like fluid.

## 3 Quick start guide

- Ensure Fenics ([www.fenics.org](http://www.fenics.org)) and dolfin-adjoint ([www.dolfin-adjoint.org](http://www.dolfin-adjoint.org)) are installed and can properly interface with Python. Or use the virtual machine provided. Or install docker.
- Open the Icy software and import the image sequence you are going to work with. If the resolution is too high you may want to scale it down, and if the background is useless you may want to crop it (so that multiscale does not spend time warping it at each step).
- Draw a ROI over the cell of interest on the first frame (or use the HK-Means plugin if the sequence is 3D).

- Install and launch the plug-in by typing *Biophysical Optical Flow* in Icy's search bar. This will install all source codes in a temporary folder on the disk and open the main script within Icy's *Script Editor*.
- Adjust the various options (output folder, plotting the results, parameters etc.) and click on the *Play* icon.
- If the code converges, the results will be saved in the output folder.
- There are two options to visualize the results:  
**Option 1: Icy** a) Load a video that was analyzed with BioFlow; b) Launch the *BioFlowDisplay* plug-in via Icy's search bar; c) Select the top-level folder (named with time and date) containing the results for this video; c) Click the *Play* button to load and display the pressure, velocity and force fields directly onto the raw data as extra layers; d) Go to the *layer* panel (in Icy's left-hand control panel) to show or hide the various layers; e) The scale factor and min-max ranges can be used to improve visualization; f) The camera icons on top of the sequence viewer can be used to save a screenshots of the final display.  
**Option 2: Paraview** a) Right-click in the Pipeline browser and click *open*; b) Select any of the top-level .pvd file (e.g. u.pvd for the velocity field, not u\_t000.pvd), then click *apply*; c) If the .pvd file contains a vector field (e.g. velocity or force fields), add the *glyph* filter to visualize the arrows; d) when working with 3D data, the *slice* filter may come in handy.

## 4 Parameters

- **savdir**: output folder (needs not exist). *string*. Example: "~/Documents/Bioflow/example"
- **parallel**: number of processors to be used (each pair of images in a sequence is parallelized). *integer* or None (as in not parallel).
- **plotting**: whether or not you want a pop-up plot of the velocity field. *boolean*.
- **phys param**: physical parameters of the problem, in this case the viscosity constant in *Pa s*. If unknown, use [1.]. In any case, the pressure and force will only be up to a constant and the velocity and divergence will be unaffected. *[float]*
- **fun param**: the parameters/constants in the adapted optical flow functional.  $\alpha, \gamma, \eta$  are associated to the penalisation of body forces, tangential velocity and gradient of divergence, respectively. *tol* is the convergence tolerance of the algorithm (with respect to  $10e-4$ ). *[ $\alpha, 1., \gamma, \eta, tol$ ]*. In 3D that is *[ $\alpha, 1., \gamma, tol$ ]* In our case, these parameters are optimised using Brent to try to predict the next image.
- **scale param**: the algorithm normally uses the metadata of the image (e.g. time between frames) to scale the problem to a unitless version, solve it and rescale it back. If the image does not inherently contain this information provide it as: *[time between frames, length of the y size of the image, [pixel size x/pixel size y]]* in seconds, microns and unitless ratio (e.g. micron divided by micron). For the 3D case: *[time between frames, length of the y size of the image, [pixel size x/pixel size y, pixel size z/pixel size y]]*
- **segment**: True if you want to run on the cell, False if you want BioFlow to run on the whole image. *boolean*.
- **segmentation param**: in order to segment the cell in the given image the algorithm uses the active contours model. Some of the parameters associated with the method include *[region sensitivity, contour resolution, convergence criterion]*. The default parameters ( $[1.5, 2., 1E - 2]$ ) should work, but if the user detects any faulty segmentation, they can adjust. Decreasing all the parameters (but *tol*) will give more strength to the actual fitting of the movement and may fix issues.
- **multiscale param**: in order to be able to detect large movements, the algorithm follows a multi-scale approach; first it detects the bigger displacements in a smoothed image and then progressively refines them until the original resolution. *[min pixel size, max pixel size, scale factor, mesh resolution]*. Respectively, the coarsest scale to be considered (the number of pixels that the image should have so that the displacements are one pixel or less); the finest scale (*None* for original image, smaller for less detailed results); the pixel ratio between two consecutive scales; the mesh resolution at the original scale (default is up to pixel size).

## 5 Output

The results are stored in the folder chosen as output. This folder contains a folder for each of the quantities  $u$  (velocity),  $p$  (pressure),  $r$  (divergence), mesh... Each of these folders contains a *.vtu* file for each time point storing the values of the quantity at each mesh point (and the mesh itself). In this folder the user can also find a *.pvd* both for each time point and for the whole sequence at once (e.g. *u.pvd*). *.vtu* contains the real data and *.pvd* are “pointers” to it. *u.pvd* will contain the whole sequence so you can play it like a movie while *u\_t003.pvd* (e.g.) will contain the flow between images 3 and 4.

These extensions are one of VTK’s file formats and can be read and imported to C++, Java, Python, etc. scripts through easily available packages after which you can play handle the data. Furthermore, they are accepted by many visualisation tools such as Paraview or Icy itself. One needs only to open the corresponding *.pvd* file (if the magnitude is vectorial the Glyph filter has also to be applied) of a time point or of the whole sequence.

The data folder will also contain a *aux.txt* file for each time point in the following format:

*timepoint maxu minu avgu maxf minf avgf maxp minp avgp*

so that they are readily accessible without a VTK import. These can be concatenated through a bash script and easily imported as text and plotted anywhere (e.g. R, Python, Excel...):

```
#!/bin/bash
SAVEIFS=$IFS
IFS=$(echo -en "\n\b") # otherwise spaces in folder names mess everything up
folder_list='ls -d */ | cut -f1 -d\'' # getting rid of the slash. OR echo */ , in case:
for f in ${folder_list};
do
    cat ${f}/data/t* > ${f}_results_uvp.txt;
done
IFS=$SAVEIFS # OR unset IFS if by default
in case you have many folders. Otherwise just:
cat *.txt > whatever_name.txt
```

## References

- [1] Boquet-Pujadas, A., Lecomte, T., Manich, M., Thibeaux, R., Labruyère, E., Guillén, N., Olivo-Marin, J.C. & Dufour, A. C. BioFlow: a non-invasive, image-based method to measure speed, pressure and forces inside living cells. *Scientific reports* **7**, 9178, doi:10.1038/s41598-017-09240-y (2017).