Supplementary information

Microscale hydrodynamic confinements: shaping liquids across length scales as a toolbox in life sciences

David P. Taylor[a, b, d†], Prerit Mathur[a, c†], Philippe Renaud[b] and Govind V. Kaigala[a*]

[a] *IBM Research - Europe, Säumerstrasse 4, 8803 Rüschlikon, Switzerland*

[b] *École Polytechnique Fédérale de Lausanne (EPFL), Microsystems Laboratory 4, 1015 Lausanne, Switzerland*

[c] *Dept. of Chemistry and Applied Biosciences, Eidgenössische Technische Hochschule (ETH), Vladimir-Prelog-Weg 1-5/10, 8093 Zurich, Switzerland*

[d] *Current address: Dept. of Mechanical and Process Engineering, ETH Zürich, Lab of Thermodynamics in Emerging Technologies, 8092 Zürich, Switzerland*
† *DT and PM made equal contributions*

* Corresponding author: gov@zurich.ibm.com

Most critical sections of the numerical MATLAB tool

The startup function, which is automatically called when the app is opened, plots an exemplary flow field for two apertures with given location and strength. This piece of code illustrates the entire workflow, which is also employed for later, user-defined computations. Variables shared across functions are stored in the namespace "app". The naming and calling of variables would look a little different if everything was implemented locally. For seeing this code, open the MATLAB App Designer, choose "open" and select the ".mlapp" version of our tool, then switch to "Code View"

```
% Code that executes after component creation
        function startupFcn(app)
            [X, Y] = meshgrid(0:app.xdim.Value/10,...
                0:1:app.ydim.Value/10);
            app.plane = complex(X,Y);
            app.pos.Data = [(app.xdim.Value/2-75), (app.ydim.Value/2), 1; ...
                (app.xdim.Value/2+75), (app.ydim.Value/2), -3];
            app.strx.Value = app.pos.Data(1,1);
            app.stry.Value = app.pos.Data(1,2);
            if app.first == 0
                app.omega = app.potflow(1)+app.potflow(2);
                Phi = real(app.omega);
                Phi(~isfinite(Phi)) = 0;
                contour(app.graph, Phi, 50)
                app.first = 1;
```

make two 2D matrices for x and y coordinates of each point (scaling of 1/10)

combine these matrices in one complex 2D matrix

define location and strength of 2 sources

call function "potflow" to calculate complex potential of each source and superimpose the potentials.

only use the velocity potential (real part)

```
..      [seedx, seedy] = meshgrid(app.strx.Value/10-app.strsz.Value/20
                                  :0.5:app.strx.Value/10+app.strsz.Value/20,...
                                      app.stry.Value/10-app.strsz.Value/20 ...
                                  :0.5:app.stry.Value/10+app.strsz.Value/20);
                                  [U, V] = gradient(Phi);
                                  streamline(app.graph, U, V, seedx, seedy)
            end

            xlim(app.graph, [0, app.xdim.Value/10]);
            ylim(app.graph, [0, app.ydim.Value/10]);
            app.graph.XTick = linspace(0, app.xdim.Value/10, 11);
            app.graph.YTick = linspace(0, app.ydim.Value/10, 11);
            app.graph.XTickLabel = linspace(0, app.xdim.Value, 11);
            app.graph.YTickLabel = linspace(0, app.ydim.Value, 11);
            axis(app.graph, 'manual')
            disableDefaultInteractivity(app.graph)
            app.checkphi.Value = 1;
            app.checkstream.Value = 1;
        end
```

seeding for streamlines

compute the local derivative for finding the flow velocity/discharge (this corresponds to the two final equations in tutorial box 2)

The rest is cosmetics on the plotting.

```
function field = potflow(app, n)
        source = app.pos.Data(n,:);
        field = (1E6*source(3)/60)/(2*pi)*...
            log(app.plane-complex(source(1)/10, source(2)/10));   % One unit
        % in the plot = 0.01 mm, “discharge” is computed in [0.01 mm2/sec]
    end
```

Computation of complex potential for each source, Q_m is converted from µL/min to SI units, scaling of 1/10 as above.