# Three ways to compute multiport inertance

G. D. McBain[1] Tom Gustafsson[2]
S. G. Mallinson[1] B. R. Brown[1]

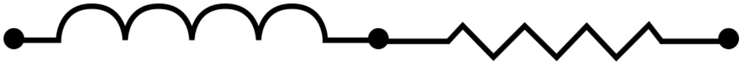
[1]Memjet

[2]VTT Technical Research Centre of Finland

Computational Techniques and Applications Conference
Newcastle, NSW, 2018

# Impulsively generated flow in liquids

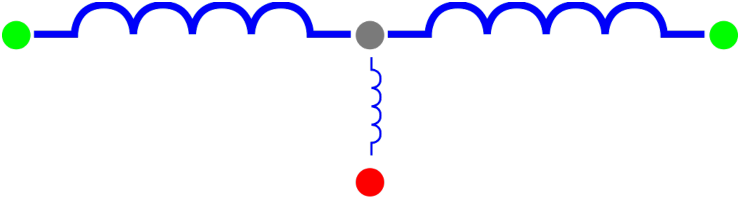
# Heaviside's electrical–hydraulic analogy



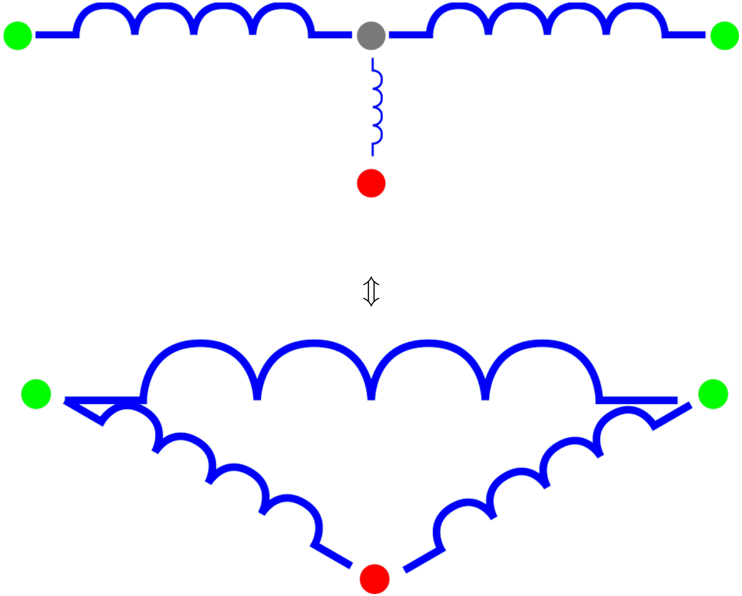
# Beasley's (1977) two-spoke network model

# Beasley uncouples forwards & backwards flows?

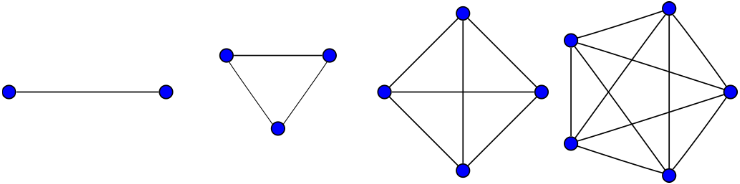
# Modified Beasley model



# Y–Δ transform



# Complete graph



# Early asymptotic Navier–Stokes equation

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u}$$

$$\nabla\cdot\mathbf{u} = 0$$

$$\Downarrow$$

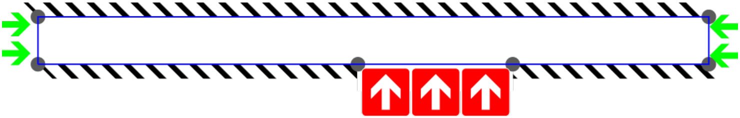
$$\rho\mathbf{U} \sim -\nabla\Pi$$

$$\nabla\cdot\mathbf{U} = 0$$

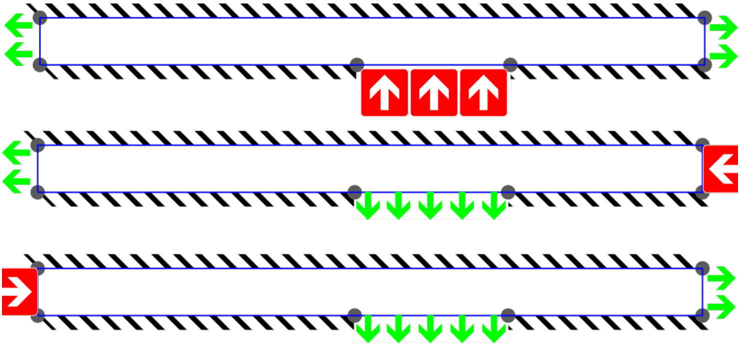
$$\Downarrow$$

$$-\nabla^2\Pi = 0$$

# Boundary conditions



# Multiport boundary conditions



# (Reciprocal) inertance is not a number, it's a matrix

The classical two-port law

$$q = \frac{\Delta\Pi}{L}$$

is really short for

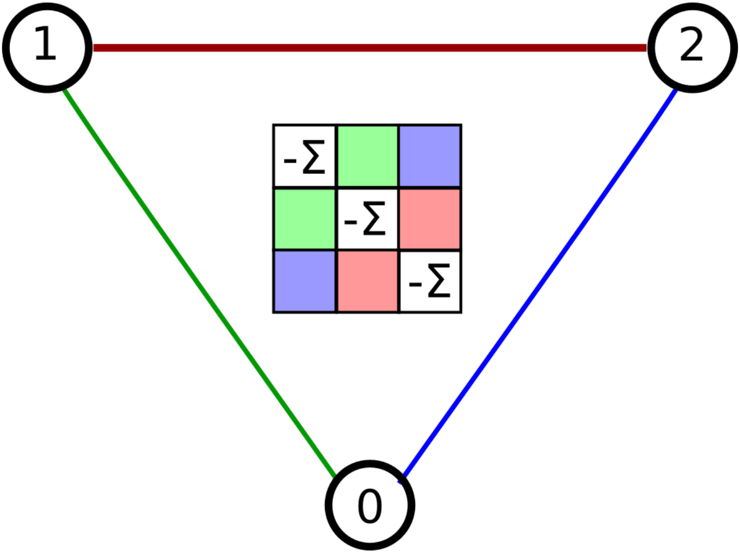
$$\begin{Bmatrix} q_0 \\ q_1 \end{Bmatrix} = \begin{bmatrix} L^{-1} & -L^{-1} \\ -L^{-1} & L^{-1} \end{bmatrix} \begin{Bmatrix} \Pi_0 \\ \Pi_1 \end{Bmatrix}$$

# The reciprocal inertance coefficients

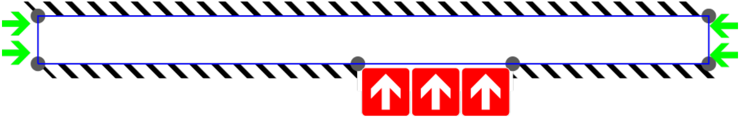
$$\left\{\begin{array}{c} q_0 \\ q_1 \\ \vdots \\ q_{n-1} \end{array}\right\} = \left[\begin{array}{cccc} s_{00} & s_{01} & \cdots & s_{0,n-1} \\ s_{10} & s_{11} & & \\ \vdots & & \ddots & \\ s_{n-1,0} & & \cdots & s_{n-1,n-1} \end{array}\right] \left\{\begin{array}{c} \Pi_0 \\ \Pi_1 \\ \vdots \\ \Pi_{n-1} \end{array}\right\}$$

$$s_{ij} \equiv \left[\mathbf{n}, \nabla \Pi^{(j)}\right]_{\Gamma_i}$$

# Three-port reciprocal inertance matrix



# First way to compute reciprocal inertance coefficients

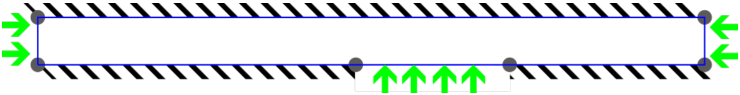


$$s_{ij} \equiv \left[\mathbf{n}, \nabla \Pi^{(j)}\right]_{\Gamma_i}$$

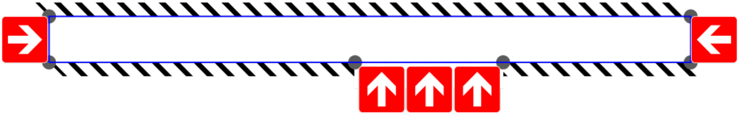
# Variational form of boundary value problems

$$
\begin{aligned}
-\nabla^2 \Pi^{(j)} &= 0 \\
\left\langle P, -\nabla^2 \Pi^{(j)} \right\rangle &= 0\,, \qquad \forall P \\
\left\langle \nabla P, \nabla \Pi^{(j)} \right\rangle &= \left[ P\mathbf{n}, \nabla \Pi^{(j)} \right] \\
&= \sum_i \left[ P\mathbf{n}, \nabla \Pi^{(j)} \right]_{\Gamma_i} \\
&= 0\,, \qquad \text{if } P = 0 \text{ on all ports } \Gamma_i
\end{aligned}
$$

# Property: zero column sums (incompressibility)



# Property: zero row sums (gauge pressure)



# Domain integrals for reciprocal inertance coefficients

$$
\begin{aligned}
\left\langle \Pi^{(i)}, -\nabla^2 \Pi^{(j)} \right\rangle &= 0 \\
\left\langle \nabla \Pi^{(i)}, \nabla \Pi^{(j)} \right\rangle &= \left[ \Pi^{(i)} \mathbf{n}, \nabla \Pi^{(j)} \right] \\
&= \left[ \mathbf{n}, \nabla \Pi^{(j)} \right]_{\Gamma_i} \\
&\equiv s_{ij}
\end{aligned}
$$

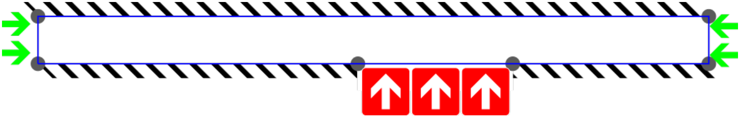
# Second way to compute reciprocal inertance coefficients

$$s_{ij} = \left\langle \nabla \Pi^{(i)}, \nabla \Pi^{(j)} \right\rangle$$

# Property: symmetry

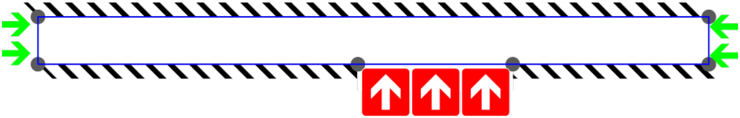
$$s_{ij} = \left\langle \nabla \Pi^{(i)}, \nabla \Pi^{(j)} \right\rangle$$

# Property: positive diagonal (driving point)



$$s_{ii} = \left\| \nabla \Pi^{(i)} \right\|^2 > 0$$

# Property: negative off-diagonals (transfer)



# Galerkin method

$$\left\langle \nabla P, \nabla \Pi^{(k)} \right\rangle = 0\,, \qquad \forall P : P = 0 \text{ on ports}$$

$$\sum_j \langle \nabla \phi_i, \nabla \phi_j \rangle \, \Pi_j^{(k)} = 0\,, \qquad \forall i$$

# Third way to compute reciprocal inertance coefficients

- Galerkin:

$$\sum_j \langle \nabla\phi_i, \nabla\phi_j \rangle \Pi_j^{(k)} = 0$$

$$\sum_j a_{ij} \Pi_j^{(k)} = 0$$

- Second way:

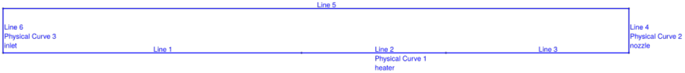
$$s_{ij} = \left\langle \nabla\Pi^{(i)}, \nabla\Pi^{(j)} \right\rangle$$

- Second way + Galerkin:

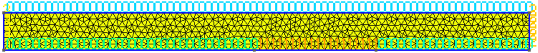
$$s_{ij} = \sum_{\ell m} \Pi_\ell^{(i)} a_{\ell m} \Pi_m^{(j)}$$

- Same matrix!
- In Python: `p.T @ L @ p`

# Implementation: defining geometry in Gmsh



# Implementation: meshing in Gmsh



# Implementation: finite elements with `scikit-fem`

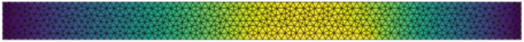
https://github.com/kinnala/scikit-fem

# Implementation: Python, single driven port

```
mesh = MeshTri.load('mesh.msh')
basis = InteriorBasis(mesh, ElementTriP1())

L = asm(laplace, basis)
ports = basis.get_dofs(mesh.boundaries)
dofs = basis.complement_dofs(ports)

p = zeros(basis.N)
p[ports['heater'].all()] = 1.
p[dofs] = solve(*condense(L, 0*p, p, dofs))

print(p.T @ L @ p)
mesh.plot(p).get_figure().savefig('potential.png')
```

# Output: single driven port

```
0.3612227893943555
```

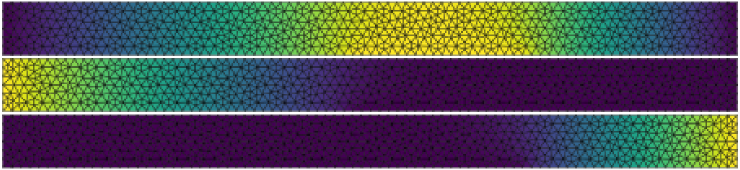


# Implementation: Python, multiports

```
mesh = MeshTri.load('mesh.msh')
basis = InteriorBasis(mesh, ElementTriP1())

L = asm(laplace, basis)
ports = basis.get_dofs(mesh.boundaries)
dofs = basis.complement_dofs(ports)

p = zeros((basis.N, len(mesh.boundaries)))
for j, port in enumerate(ports.values()):
    p[port.all(), j] = 1.
p[dofs] = solve(*condense(L, 0*p, p, dofs))

print(p.T @ L @ p)
for p, key in zip(p.T, ports.keys()):
    mesh.plot(p).get_figure().savefig(f'{key}.png')
```

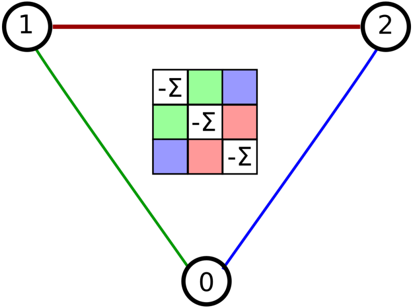
# Output: multiports

```
[[ 3.61222789e-01 -2.21070533e-01 -1.40152256e-01]
 [-2.21070533e-01  2.21170002e-01 -9.94689095e-05]
 [-1.40152256e-01 -9.94689095e-05  1.40251725e-01]]
```



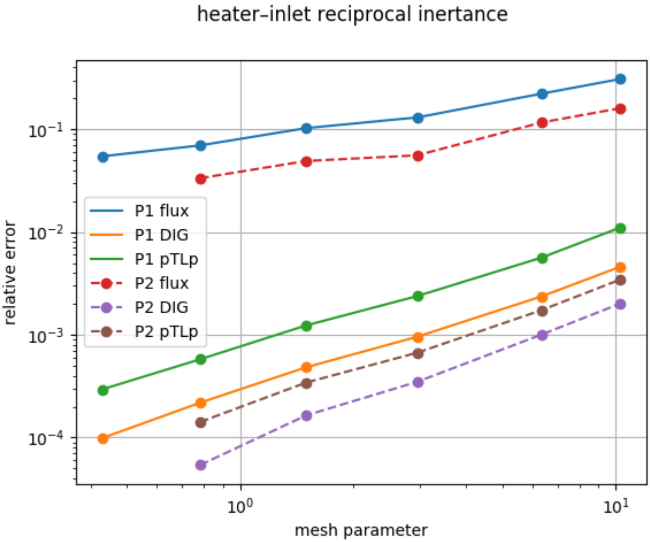
# Beasley revisited

```
[[ 3.61222789e-01 -2.21070533e-01 -1.40152256e-01]
 [-2.21070533e-01  2.21170002e-01 -9.94689095e-05]
 [-1.40152256e-01 -9.94689095e-05  1.40251725e-01]]
```

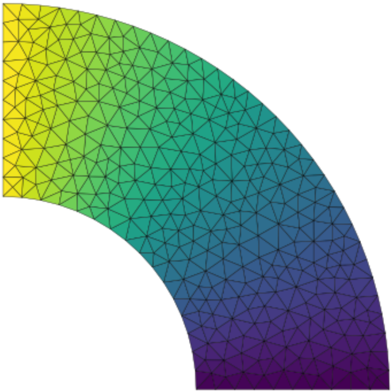



$$\begin{bmatrix} b+f & -b & -f \\ -b & b & 0 \\ -f & 0 & f \end{bmatrix}$$

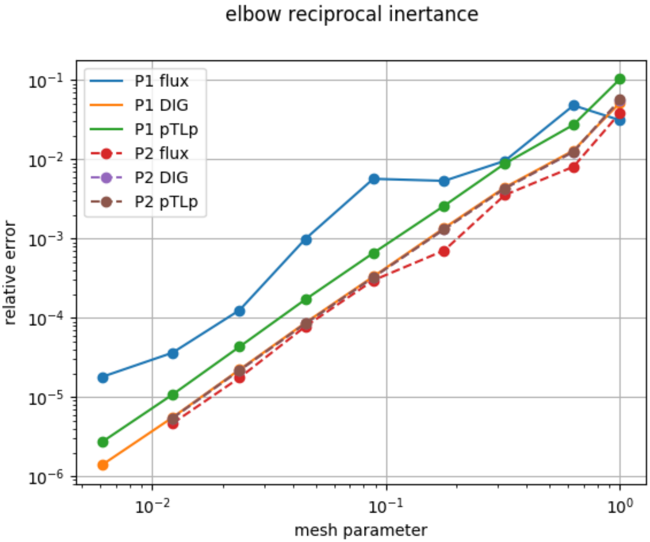
# Convergence of the three ways to compute inertance



# Example: right-angle circular bend



# Convergence for the right-angle circular bend



# Thank you

- *Memjet*
- Frédéric Hecht (Laboratoire J.-L. Lions, UPMC)
- also *Gmsh*, *pygmsh*, *meshio*, *Inkscape*, ...