Three-Dimensional Stokes Flow Using the Method of Fundamental Solutions in Python3

Josiah J.P. Jordan

May 2025

1 Background

This section contains the basic background information on the Method of Fundamental Solutions (MFS applied) to Stokes flow. *Skip to Section 2 for examples and explanation of how to use the* `pythonMFS` *code.* For a more detailed implementation, see [2], on which this guide is based; this Python code has been developed from MATLAB code from my PhD, I would like to thank my supervisor, Duncan Lockerby, for all of the work in this guide.

The MFS in `pythonMFS` solves the linear Stokes equations for vanishing Reynolds number flows, with a point forcing to the momentum conservation governing equation:

$$\boldsymbol{\nabla} \cdot \boldsymbol{v} = 0\,, \tag{1a}$$

$$\mu \boldsymbol{\nabla}^2 \boldsymbol{v} - \boldsymbol{\nabla} p = \boldsymbol{f}\delta(\boldsymbol{x})\,, \tag{1b}$$

where μ is the dynamic viscosity, $\boldsymbol{v}$ the fluid velocity, p the pressure, $\boldsymbol{f}$ the force and $\boldsymbol{x}$ the position at which the force is applied. We wish to solve these equations given the no-slip boundary condition:

$$\boldsymbol{v}(\boldsymbol{x}) = \boldsymbol{v}_{\text{wall}}(\boldsymbol{x}) \qquad \text{for} \quad \boldsymbol{x} \in \mathcal{S}\,, \tag{2}$$

where $\boldsymbol{v}_{\text{wall}}$ is the velocity of the fluid at point $\boldsymbol{x}$ on the particle's surface, $\mathcal{S}$, and that in the far field, the velocity disturbances vanish:

$$\boldsymbol{v}(\boldsymbol{x}) = 0 \qquad \text{for} \quad \boldsymbol{x} \to \infty \tag{3}$$

. The fundamental solution in 3D to the governing equations with the boundary conditions imposed is the Oseen tensor:

$$\mathbb{J}(\boldsymbol{x}) = \frac{1}{8\pi\mu}\left(\frac{\mathbb{I}}{|\boldsymbol{x}|} + \frac{\boldsymbol{x}\otimes\boldsymbol{x}}{|\boldsymbol{x}|^3}\right). \tag{4}$$

Given a single point force at point $\boldsymbol{x}^o$, the velocity at a point $\boldsymbol{x} \neq \boldsymbol{x}^o$ is given by:

$$\boldsymbol{v} = \boldsymbol{f} \cdot \mathbb{J}(\boldsymbol{x} - \boldsymbol{x}^o)\,. \tag{5}$$

1.1 Single Particle MFS

The problem to be solved is: given the velocity of a single particle relative to a fluid, find the resultant force. To do this, the surface of the particle can be discretised into N boundary nodes, onto which we impose our boundary conditions (velocities). We then distribute M points within the particle, outside the flow domain, where our fundamental solutions are located. Let $\boldsymbol{b}^n$ be the position of the n^{th} node, and $\boldsymbol{s}^m$ be the location of the m^{th} singularity site, and

$$\boldsymbol{r}^{bs} = \boldsymbol{b}^n - \boldsymbol{s}^m\,, \tag{6}$$

then the Oseen tensor of this node-site pair is:

$$\mathbb{J}(\boldsymbol{r}^{bs}) = \frac{1}{8\pi\mu}\left(\frac{\mathbb{I}}{|\boldsymbol{r}^{bs}|} + \frac{\boldsymbol{r}^{bs}\otimes\boldsymbol{r}^{bs}}{|\boldsymbol{r}^{bs}|^3}\right). \tag{7}$$

As the governing equations are linear, a superposition of the fundamental solutions provides a full solution. Given a $3N \times 1$ super-vector of all the velocities of the nodes ($\mathcal{V}$), we wish to find the $3M \times 1$ super-vector of all the forces on the site ($\mathcal{F}$). These are related by:

$$\mathcal{F} = \mathbb{A}^{+} \cdot \mathcal{V}\,, \tag{8}$$

where $\mathbb{A}^+$ is the pseudo-inverse of the matrix:

$$\mathbb{A} = \begin{bmatrix} \mathbb{J}(\boldsymbol{r}^{11}) & \dots & \mathbb{J}(\boldsymbol{r}^{1M}) \\ \vdots & \ddots & \vdots \\ \mathbb{J}(\boldsymbol{r}^{N1}) & \dots & \mathbb{J}(\boldsymbol{r}^{NM}) \end{bmatrix}, \tag{9}$$

and

$$\mathcal{V} = \begin{bmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^N \end{bmatrix}, \quad \text{and} \quad \mathcal{F} = \begin{bmatrix} \boldsymbol{f}^1 \\ \vdots \\ \boldsymbol{f}^M \end{bmatrix}. \tag{10}$$

Here, $\boldsymbol{f}^m$ is the force of the m^{th} site, $\boldsymbol{v}^n$ is the velocity of the n^{th} node, and $\mathbb{J}(\boldsymbol{r}^{nm})$ is the Oseen tensor of the n^{th} node/m^{th} site pair, with $\mathbb{J}$ being calculated by Equation 7 and $\boldsymbol{r}^{nm}$ by Equation 6. Note, to reduce the stiffness of the system, we shall always use $M < N$. If the body is rotating with an angular velocity $\boldsymbol{\omega}$, and translating with velocity $\boldsymbol{V}$, then the velocity of the n^{th} boundary node is:

$$\boldsymbol{v}^n = \boldsymbol{V} + \boldsymbol{r}^{bo} \times \boldsymbol{\omega} \tag{11}$$

where $\boldsymbol{r}^{bo} = \boldsymbol{b}^n - \boldsymbol{r}^o$, and $\boldsymbol{r}^o$ is the position of the centre of rotation.

The net force and torque, $\boldsymbol{\tau}$, on the particle are given by:

$$\boldsymbol{f}^{\text{net}} = \sum_{m=1}^{M} \boldsymbol{f}^m, \tag{12a}$$

$$\boldsymbol{\tau}^{\text{net}} = \sum_{m=1}^{M} \boldsymbol{r}^{mo} \times \boldsymbol{f}^m, \tag{12b}$$

where $\boldsymbol{r}^{mo}$ is the vector between the m^{th} site and the centre of rotation of the particle.

1.2 Preliminaries

The most basic example for a particle in Stokes flow in the force on a sphere. A sphere of radius R, moving at a velocity $-\boldsymbol{v}$ and rotating with angular velocity $\boldsymbol{\omega}$ submerged in an infinite, quiescent, Newtonian fluid with $Re = 0$ will experience a drag force and torque given by:

$$|\boldsymbol{f}| = 6\pi\mu R|\boldsymbol{v}|, \tag{13a}$$

$$|\boldsymbol{\tau}| = 8\pi\mu R^3|\boldsymbol{\omega}|. \tag{13b}$$

For non-spherical particles, the force and torque are related to the velocity and angular velocity by:

$$\begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} \mathbb{R} & \mathbb{C}^\dagger \\ \mathbb{C} & \mathbb{Q} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{14}$$

where $\mathbb{R}$ is the resistance tensor, $\mathbb{C}$ is the coupling tensor, and $\mathbb{Q}$ the rotational resistance tensor. This can be written more succinctly, as:

$$\mathfrak{F} = \mathcal{R} \cdot \mathcal{V}, \tag{15}$$

where $\mathcal{R}$ is known as the grand resistance tensor; far more information can be found in [1].

2 Code Examples and Usage

The two quantities of most interest are the net force/torque (Eq. 12), and the grand resistance tensor (Eq. 15). All the code requires `numpy`, and we shall `import pythonMFS as mfs`. The file `pythonMFS.py` contains all the functions necessary to run basic MFS calculations. The first step in all MFS calculations is creating a list of nodes (`rb`) and sites (`rs`). This can be done in multiple ways:

- `mfs.sphereMaker(200,1)` creates a 200 points around a sphere of radius 1
- `mfs.spheroidMaker(200,1,1,2)` creates 200 points around a spheroid with $R_z = 2R_y = 2R_x$
- Install and use `distmesh` and create shapes from implicit functions [3]

- Load points from a `.obj` file or similar (note, this can cause issues)

For a closed particle, `rs` must always lie within `rb`, *outside* the flow domain.

Set `example` to the number of the example you wish to run. The first two examples are simple force calculations, the second two are flow visualisation techniques.

2.1 Example 1: Force on a Sphere

This example calculates the force and torque on a unit sphere, printing the normalised result. Step-by-step:

1. Create `rb` – the unit sphere we wish to simulate. Give it N nodes with radius R.
2. Create `rs` – the sphere on which the sites are located. Give it $M(< N)$ nodes and radius $R^s(< R)$.
3. Define `V` and `omega` – the velocity and angular velocity of the body (Eq. 11).
4. `mfs.findForceAndTorque` is a function that calls two functions:
   (a) `mfs.matrixConstruct` constructs $\mathbb{A}$ (Eq. 9) from the given node and site positions (`rb` and `rs`).
   (b) `mfs.rhsConstruct` constructs $\mathcal{V}$ given $\boldsymbol{V}$ and $\boldsymbol{\omega}$.
   (c) Compute $\mathcal{F}$ using Eq. 8.
   (d) Sum all forces and torques to give the net force and torque, Eq. 12.
5. Print force and torque normalised by the analytical value, Eq. 13.

2.2 Example 2: Resistance Tensor of a Spheroid

Here, we calculate the force and torque in the exact same way as Example 1, however, this time for a spheroid with radii $a = b = \frac{1}{2}c$ in the x, y, and z directions. By imposing three orthogonal velocities/angular velocities, we can create $\mathcal{R}$:

1. Begin loop over three orthogonal directions:
   (a) The first row of $\mathcal{R}$ is $\begin{bmatrix}\boldsymbol{f}^{\text{net}} & \boldsymbol{\tau}^{\text{net}}\end{bmatrix}$ from a translational velocity in the x direction.
   (b) The fourth row of $\mathcal{R}$ is $\begin{bmatrix}\boldsymbol{f}^{\text{net}} & \boldsymbol{\tau}^{\text{net}}\end{bmatrix}$ from an angular velocity in the x-direction.
   (c) The second and fifth from a translational and angular velocity in the y-direction.
   (d) The third and sixth from a translational and angular velocity in the z-direction.
2. Ensure symmetry (remove small numerical errors) by $R = \frac{1}{2}(\mathcal{R} + \mathcal{R}')$.
3. print result.

The force and torques are normalised by a the analytical value for a sphere with radius $= a = b = \frac{1}{2}c$

2.3 Example 3: Manual Streamlines

This example plots streamlines calculated from a pre-defined starting position around an object. The concept of this example is that given the forces on each site in a system, at any point ($\boldsymbol{p}$) in the flow, the velocity at that point ($\boldsymbol{v}^p$) can be calculated by:

$$\boldsymbol{v}^p = \mathbb{A}^p \cdot \mathcal{F} \tag{16}$$

where $\mathbb{A}^p$ is the matrix of linear equations created between the point $\boldsymbol{r}$ and all of the sites, i.e.:

$$\mathbb{A}^p = \begin{bmatrix}\mathbb{J}(\boldsymbol{r}^{p1}) & \dots & \mathbb{J}(\boldsymbol{r}^{pM})\end{bmatrix} \tag{17}$$

where $\boldsymbol{r}^{pm} = \boldsymbol{p} - \boldsymbol{s}^m$; $\mathbb{A}^p$ will have size $(3 \times 3M)$ and $\mathcal{F}$ will have size $(3M \times 1)$, and $\boldsymbol{v}^p$ will therefore have size (3×1): the three components of velocity at that point in the x-, y-, and z-directions.

As the velocity at any point can be calculated, using a simple Euler time-stepping algorithm with time-step δ, we can calculate the trajectory of a mass-less tracer particle starting at position $\boldsymbol{p}^0$. The code follows the following procedure:

1. Calculate the force on all the sites, $\mathcal{F}$, from `rb` and `rs`, from a given global velocity $\boldsymbol{V}$.
2. Calculate the velocity at $\boldsymbol{p}^0$, $\boldsymbol{v}^0$ from Eq. 16.
3. Begin a loop until the streamline is at the desired finishing point:
   (a) Calculate the new (i^{th}) streamline position as: $\boldsymbol{p}^i = \boldsymbol{p}^{i-1} + \delta \cdot (\boldsymbol{V} - \boldsymbol{v}^i)$
   (b) Store $\boldsymbol{p}^i$ for plotting and calculating $\boldsymbol{p}^{i+1}$
4. After n iterations, the list of positions $\boldsymbol{p}^{0:n}$ describe the trajectory of the tracer particle starting at $\boldsymbol{p}^0$

Figure 1 shows examples of some streamlines plotted around different shapes using this algorithm and `mayavi`.

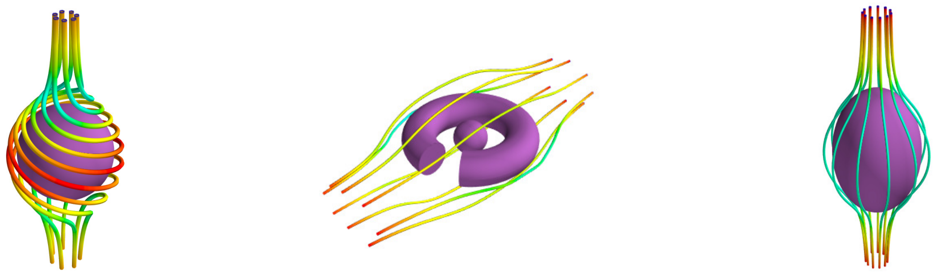
Figure 1: Example of manual streamlines plotted using `mayavi`

2.4 Example 4: Velocity Field Contour Plot

This example follows the same concept as Example 3, however this is a static view with no time-stepping. This example is arguably the simpler of the two. The procedure is as follows:

1. Calculate the force on all the sites, $\mathcal{F}$, from `rb` and `rs`, from a given global velocity $\boldsymbol{V}$.
2. Divide the area of flow to be visualised into a grid
3. Begin a loop over all the points:
   (a) If the point, $\boldsymbol{p}$, is outside the particle, then $\boldsymbol{v}^p = \mathbb{A}^p \cdot \mathcal{F}$ (c.f. 17)
   (b) If the point, $\boldsymbol{p}$, is inside the particle or on its surface, then $\boldsymbol{v}^p = 0$
4. The velocity field on the grid can now be plotted however you would like; `matplotlib`'s `contourf` and `streamplot` I particularly like.

Figure 2 shows an example of the velocity field visualised with this technique.

2.5 Example 5: Site Location

This example is for those interested in the numerics. The location of the sites is one of the most controversial parts of the MFS, and here we can see why. The absolute error of the MFS calculation against that of the analytical expression is given, in percent, by:

$$\epsilon = 100 \left| 1 - \frac{||\boldsymbol{f}^{\text{net}}||}{6\pi\mu R||\boldsymbol{v}||} \right| . \tag{18}$$

This, in general, is inversely proportional to the matrix condition number κ of $\mathbb{A}$, and:

$$\epsilon\kappa \approx \mathcal{O}(1) . \tag{19}$$

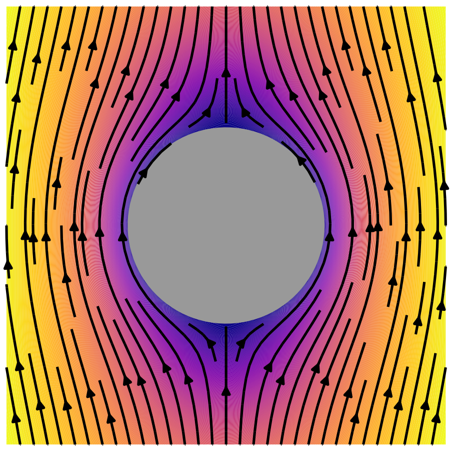

Figure 2: Velocity field of a sedimenting sphere, colours denote magnitude of velocity

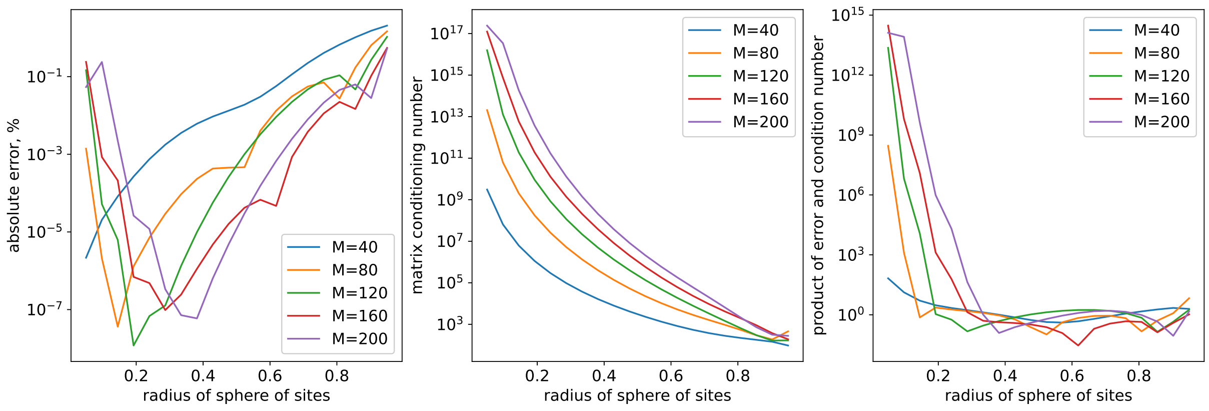

Figure 3: Error, conditioning number, and their product with site position relative to the node radius.

Figure 3 shows how the error, conditioning number, and value $\epsilon\kappa$ varies with the radius of the sphere on which the sites are located at different values of $M = \lfloor 0.8N \rfloor$.

Clearly, decreasing the size of the site-sphere radius increases the accuracy to a point, when the points are nearly overlapping the error starts to increase drastically – this is most noticeable at larger values of M, i.e., greater amount of overlap. With decreasing size of the site-sphere radius, the matrix conditioning number increases greatly, and this is also the case with increasing M. Care must therefore be taken when placing the sites relative to the nodes: too close, and the result will be inaccurate; too far, and the system of equations will be unstable.

$R^s = 0.5R^b$ and $N = 150 - 200$ seem to work well.

2.6 Caveats and Limitations

There are a few known issues with the MFS:

- **It does not handle slender bodies well**. This is due to the sites being too close to each other with increasing slenderness, decreasing the conditioning number of $\mathbb{A}$ to unstable levels.
- **It can only handle closed shapes**. This is because the singularity in the $\mathbb{J}$ tensor when $b^n - s^m = 0$, so there can be no flow evaluated at the singularity site.

- **The result is sensitive to the placement of the sites**. Too far, and you get a badly conditioned matrix; too close, and you get an inaccurate result.
- **The inversion of $\mathbb{A}$ may take a long time for large N.** Inverting $\mathbb{A}$ to get a result can take a long time for large numbers of particles/very fine meshes; SVD cost scales with $\mathcal{O}(MN^2)$. Anything over $N = 1000$ takes quite a while on my PC.

References

[1] J. Happel and H. Brenner. *Low Reynolds number hydrodynamics*. Springer Dordrecht, 1983. DOI: `doi.org/10.1007/978-94-009-8352-6`.

[2] Josiah J.P. Jordan and Duncan A. Lockerby. "The method of fundamental solutions for multi-particle Stokes flows: Application to a ring-like array of spheres". In: *Journal of Computational Physics* 520 (2025), p. 113487. DOI: `doi.org/10.1016/j.jcp.2024.113487`.

[3] P.-O. Persson and G. Strang. "a simple mesh generator in MATLAB". In: *SIAM Review* 46 (2 2004), pp. 329–345. DOI: `doi.org/10.1137/S0036144503429121`.