The Journal of Open Source Software

CNearest: A C++ toolkit to use the nearest-neighbour method of regularised stokeslets algorithm to solve viscous flow problems

Neil A. Butcher [1*], **James Tyrrell** [1*], **and David Smith** [2*]

1 Advanced Research Computing, University of Birmingham, Edgbaston, Birmingham, UK **2** School of Mathematics and Centre for Systems Modelling and Quantitative Biomedicine, University of Birmingham, Edgbaston, Birmingham, UK ***** These authors contributed equally.

DOI: 10.21105/joss.07605

Software

- Review
- Repository
- Archive

Editor: Philip Cardiff

Reviewers:

- @fabienevrard
- @Pecnut

Submitted: 01 August 2024
Published: 26 August 2025

Summary

This package implements a method for solving problems in very viscous fluid dynamics, in the limit where inertia is negligible relative to viscosity. This system is particularly relevant in microscale biological systems and colloids; one specific application is the calculation of the diffusion tensor of a specified structure. The underlying algorithm is the nearest-neighbour implementation (Smith, 2018b) of the method of regularised stokeslets (Cortez, 2001; Cortez et al., 2005); this approach enables efficient and accurate calculations requiring only surface point discretisations rather than surface or volumetric meshes and avoids singular integrals. It is a feature-complete reimplementation and further development of the key test cases in Smith (2018b). The package also serves as an extensible library that can be utilised with the user's own surface discretisation to calculate the force and moment on any given three dimensional body undergoing rigid body motion, and provides a building block to model flexible swimming bodies.

Statement of need

Microscale biological flow problems typically involve complex-shaped bodies undergoing large deformations, which renders analytical calculations or methods based on volumetric meshing (finite element, finite difference) challenging to implement. However, in the case where the fluid has Newtonian or approximately Newtonian properties, following non-dimensionalisation the flow can be modelled through the linear Stokes flow equations,

$$-\boldsymbol{\nabla} p + \nabla^2 \boldsymbol{u} = 0, \quad \boldsymbol{\nabla} \cdot \boldsymbol{u} = 0, \quad (1)$$

where $p(\boldsymbol{x})$ and $\boldsymbol{u}(\boldsymbol{x})$ are respectively pressure and velocity at the point $\boldsymbol{x}$.

One of the basic problems of Stokes flow is the calculation of the force and moment on a body immersed in a fluid and subject to no-slip, no-penetration boundary conditions, due to specified rigid body motion. Mathematically, for a body with surface B subject to translational velocity $\boldsymbol{U}$ and angular velocity $\boldsymbol{\Omega}$ about the origin, the problem is to solve equation (1) subject to the boundary condition,

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{U} + \boldsymbol{\Omega} \times \boldsymbol{x}, \quad \text{for all} \quad \boldsymbol{x} \in B, \quad (2)$$

and thereby calculate the force and moment

$$\boldsymbol{F} = \iint_B \boldsymbol{\sigma} \cdot \boldsymbol{n} \, dS(\boldsymbol{x}), \quad \boldsymbol{M} = \iint_B \boldsymbol{x} \times \boldsymbol{\sigma} \cdot \boldsymbol{n} \, dS(\boldsymbol{x}), \quad (3)$$

where the stress tensor is given in terms of the pressure and velocity by,

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \boldsymbol{\nabla u} + (\boldsymbol{\nabla u})^T. \quad (4)$$

The system (1–4) forms the basis for more complex problems in which the body motion is not prescribed, or the surface motion is non-rigid. Moreover due to the linearity and instantaneity of the equations, the solution for translational velocity $\boldsymbol{U}$ and angular velocity $\boldsymbol{\Omega}$ can be determined from a linear combination of the 'basic' modes corresponding to unit velocity along each Cartesian axis and unit angular velocity about each Cartesian axis; the basic functionality of CNearest is to perform these calculations. The 6×6 matrix of force and moment coefficients corresponding to each basic rigid body mode is referred to as the *grand resistance matrix* (Pozrikidis, 2002) or *frictional tensor* (Wegener, 1981). The matrix inverse can be used to calculate the diffusion tensor quantifying the translational and rotational Brownian motion of the body (Wegener, 1981).

Linearity means that (1-4) can be solved through techniques such as the boundary element method, which results in a linear algebra problem of the form,

$$[\mathsf{A}]\,\mathsf{f} = \mathsf{u},$$

where $[\mathsf{A}]$ is a $3N \times 3N$ matrix formed by evaluating or integrating fundamental solutions of (1), f is a $3N$-vector of unknown traction or force values at each degree of freedom, and u is a $3N$-vector of velocities at each point of the discretisation. For more details, see for example Pozrikidis (1992, 2002). However these techniques involve singular solutions which present challenges for implementation. Moreover, despite its high numerical efficiency, the boundary element method requires the user to supply a true surface mesh with connectivity table and local basis system. The method of regularised stokeslets (Cortez, 2001; Cortez et al., 2005) has achieved significant success in biological fluid dynamics (see for example the Special Issue edited by Cortez and Olson (Cortez & Olson, 2021)) by removing both the mathematically singular behaviour, and avoiding the need to construct a true mesh (see also Smith (2009)). This package implements an extension of the method of regularised stokeslets which utilises separate discretisations for the surface traction and quadrature. The method, previously implemented in Matlab (Smith, 2018a) is based on nearest-neighbour interpolation, and it removes the dependence of the linear system size on the quadrature error, greatly improving accuracy and efficiency. The method distinguishes the size of the force discretisation (N) from the larger sized quadrature discretisation (Q). For more details, including application to swimming problems and error analysis, see references (Gallagher et al., 2019; Gallagher & Smith, 2018).

Despite its proven effectiveness, a barrier to wider adoption of the nearest neighbour discretisation is that the existing implementation underpinning most of the references above is an undocumented research code implemented in Matlab. That implementation primarily uses built-in linear algebra operations throughout the workflow, from stokeslet evaluation through matrix assembly, and avoids the use of for loops. While this approach has some advantages - notably brief code and very simple exploitation of parallelism (Gallagher & Smith, 2020), it tends to result in the storage of large stokeslet matrices, which can be highly inefficient as the quadrature discretisation is refined. CNearest addresses these limitations.

In the wider setting of software packages and code for Stokes flow, we note several existing packages with related but distinct aims, which the present package is intended to complement. *pyStokes* (Singh & Adhikari, 2020) implements an efficient method for solving coupled Stokes and Laplace problems associated with phoretic swimmers, utilising a spectral approximation to the flow field around potentially large numbers of spherical particles. It is therefore particularly suited to problems involving many bodies, but is less focused on detailed resolution of swimmer morphology. Along similar lines, "Stokesian dynamics in Python" (Townsend, 2024) addresses the need for efficient approximation of long range interactions of many particles; an older package implementing a similar method is *StokeD* (Parker, 2016), also coded in Python.

Implementations of the method of regularised stokeslets include an undocumented parallelized code repository of the classical method (Copos, 2020), and a more recent package which enables the use of analytic surface integrals (Ferranti, 2024; Ferranti & Cortez, 2024). The latter method greatly improves the accuracy and efficiency of the method of regularised stokeslets, however it does require the generation of a true surface mesh rather than a point cloud discretisation. Taken together, we believe that the present package addresses a gap for an extensible, accessible and efficient methodology for Stokes flow problems, particularly those involving complex geometries.

To enable efficient and scalable use of computing resources, the package is implemented in C++, with the widely-used *eigen* package handling linear algebra operations (Guennebaud et al., 2010). The package also provides three specific algorithmic developments, which arose through the process of re-implementing the method in C++:

1. Tiebreaking: ensuring reproducibility of results when the nearest-neighbour mapping is non-unique.
2. Stokeslet accumulation: building the left hand side $3N \times 3N$ matrix by adding stokeslets one at a time, rather than forming the larger $3Q \times 3N$ stokeslet matrix and multiplying by a sparse projection matrix.
3. Rescaling by local stokeslet count: the degrees of freedom of the system are thereby the physical force associated with each discretisation point, thereby ensuring that the matrix and solution vector are convergent as the quadrature discretisation is refined.

Usage

The workflow is that the user will first define the problem in terms of physical bodies such as spheres or spheroids (implementation of other shapes and flexible motions is an area for future development) from which the force and quadrature discretisations are created, then finally defining the boundary conditions. For the rigid body resistance problem (1-4) with no-slip, no-penetration boundary conditions, the boundary velocity can be calculated from supplied values of the translational and rotational velocity of the body, from which the surface velocity components at each point are calculated.

The library then constructs a linear algebra problem by detecting nearest neighbours, managing degrees of freedom, assembling matrices from the regularised stokeslets, and assembling vectors from the boundary conditions. A linear solver (either the default solver based on LU-decomposition with pivoting or an alternative chosen by the user) is invoked to find the unknown force distribution over the nodes. Finally the same tools which constructed the full size vectors are used to decompose them back to the total physical force and moment on the body.

Validation

The results of the code have been compared with the results of the (established) Matlab codebase. This comparison goes to make one of our suites of unit tests. This code is also capable of reproducing the results tables published in Smith (2018b) for the force and moment on a translating and rotating sphere and prolate spheroid. This comparison is presented as examples 1 and 2 respectively (see *examples/example-1-sphere* and *examples/example-2-prolate-spheroid*).

Summary and areas for future development

Potential areas for future development include solution of model cells with flexible flagella, swimming problems (mobility due to a prescribed swimming stroke; (Gallagher & Smith, 2018)),

multiple bodies and parallelisation (Gallagher & Smith, 2020), boundary image systems (Ainley et al., 2008; Cortez & Varela, 2015), stokeslet lines (Cortez, 2018; Smith, 2009), stokeslets with improved far field behaviour (Zhao et al., 2019), integration with elastohydrodynamics (Hall-McNair et al., 2019), stokeslet rings (Tyrrell et al., 2019), Richardson extrapolation (Gallagher & Smith, 2021), implementation of the double layer potential, which is highly important for problems involving surface slip (Smith et al., 2021), modelling self-motile phoretic colloids (Montenegro-Johnson et al., 2015) and fast multipole acceleration for very large problems (Rostami & Olson, 2016).

Acknowledgements

This work was partially supported by the Engineering and Physical Sciences Research Council (EP/N021096/1).

We acknowledge Meurig Gallagher (Centre for Systems Modelling and Quantitative Biomedicine, University of Birmingham) for his continuing work on developing the nearest-neighbour method and associated Matlab implementations since 2018. We also acknowledge Research Software Engineer Simon Hartley (Advanced Research Computing, University of Birmingham), whose earlier engagement with this problem uncovered the risk of force points which were tied in for nearest neighbour.

References

Ainley, J., Durkin, S., Embid, R., Boindala, P., & Cortez, R. (2008). The method of images for regularized Stokeslets. *J. Comput. Phys.*, *227*(9), 4600–4616. https://doi.org/10.1016/j.jcp.2008.01.032

Copos, C. A. (2020). *Parallelized CUDA method of regularized stokeslets*. https://github.com/calinacopos/Parallelized-CUDA-Method-of-Regularized-Stokeslets

Cortez, R. (2001). The method of regularized Stokeslets. *SIAM J. Sci. Comput.*, *23*(4), 1204–1225. https://doi.org/10.1137/S106482750038146X

Cortez, R. (2018). Regularized stokeslet segments. *J. Comput. Phys.*, *375*, 783–796. https://doi.org/10.1016/j.jcp.2018.08.055

Cortez, R., Fauci, L., & Medovikov, A. (2005). The method of regularized Stokeslets in three dimensions: Analysis, validation, and application to helical swimming. *Phys. Fluids*, *17*(3). https://doi.org/10.1063/1.1830486

Cortez, R., & Olson, S. (2021). *20 years of regularized Stokeslets: Applications, computation and theory*. https://www.mdpi.com/journal/fluids/special_issues/regularized_stokeslets

Cortez, R., & Varela, D. (2015). A general system of images for regularized Stokeslets and other elements near a plane wall. *J. Comput. Phys.*, *285*, 41–54. https://doi.org/10.1016/j.jcp.2015.01.019

Ferranti, D. (2024). *Regularized stokeslet surfaces*. https://github.com/djferranti/RegularizedStokesletSurfaces

Ferranti, D., & Cortez, R. (2024). Regularized stokeslet surfaces. *J. Comput. Phys*, *508*, 113004. https://doi.org/10.1016/j.jcp.2024.113004

Gallagher, M. T., Choudhuri, D., & Smith, D. J. (2019). Sharp quadrature error bounds for the nearest-neighbor discretization of the regularized stokeslet boundary integral equation. *SIAM J. Sci. Comput.*, *41*(1), B139–B152. https://doi.org/10.1137/18M1191816

Gallagher, M. T., & Smith, D. J. (2018). Meshfree and efficient modeling of swimming cells. *Phys. Rev. Fluids*, *3*(5), 053101. https://doi.org/10.1103/PhysRevFluids.3.053101

Gallagher, M. T., & Smith, D. J. (2020). Passively parallel regularized stokeslets. *Phil. Trans. R. Soc. A*, *378*(2179), 20190528. https://doi.org/10.1098/rsta.2019.0528

Gallagher, M. T., & Smith, D. J. (2021). The art of coarse Stokes: Richardson extrapolation improves the accuracy and efficiency of the method of regularized stokeslets. *R. Soc. Open Sci.*, *8*(5), 210108. https://doi.org/10.1098/rsos.210108

Guennebaud, G., Jacob, B., & others. (2010). *Eigen v3*. http://eigen.tuxfamily.org

Hall-McNair, A. L., Montenegro-Johnson, T. D., Gadêlha, H., Smith, D. J., & Gallagher, M. T. (2019). Efficient implementation of elastohydrodynamics via integral operators. *Phys. Rev. Fluids*, *4*(11), 113101. https://doi.org/10.1103/PhysRevFluids.4.113101

Montenegro-Johnson, T. D., Michelin, S., & Lauga, E. (2015). A regularised singularity approach to phoretic problems. *Eur. Phys. J. E*, *38*, 1–7. https://doi.org/10.1140/epje/i2015-15139-7

Parker, J. (2016). *StokeD*. https://github.com/johnaparker/stoked

Pozrikidis, C. (1992). *Boundary integral and singularity methods for linearized viscous flow*. Cambridge University. https://doi.org/10.1017/CBO9780511624124

Pozrikidis, C. (2002). *A practical guide to boundary element methods with the software library BEMLIB*. CRC. https://doi.org/10.1017/S0022112004008201

Rostami, M. W., & Olson, S. D. (2016). Kernel-independent fast multipole method within the framework of regularized stokeslets. *J. Fluids Struct.*, *67*, 60–84. https://doi.org/10.1016/j.jfluidstructs.2016.07.006

Singh, R., & Adhikari, R. (2020). PyStokes: Phoresis and stokesian hydrodynamics in python. *Journal of Open Source Software*, *5*(50), 2318. https://doi.org/10.21105/joss.02318

Smith, D. J. (2009). A boundary element regularized Stokeslet method applied to cilia-and flagella-driven flow. *Proc. R. Soc. Lond. A*, *465*(2112), 3605–3626. https://doi.org/10.1098/rspa.2009.0295

Smith, D. J. (2018a). *Matlab/GNU octave code for the nearest neighbour regularised stokeslets method for stokes flow*. https://github.com/djsmithbham/NearestStokeslets

Smith, D. J. (2018b). A nearest-neighbour discretisation of the regularized stokeslet boundary integral equation. *J. Comput. Phys.*, *358*, 88–102. https://doi.org/10.1016/j.jcp.2017.12.008

Smith, D. J., Gallagher, M. T., Schuech, R., & Montenegro-Johnson, T. D. (2021). The role of the double-layer potential in regularised stokeslet models of self-propulsion. *Fluids*, *6*(11), 411. https://doi.org/10.3390/fluids6110411

Townsend, A. K. (2024). Stokesian Dynamics in Python. *Journal of Open Source Software*, *9*(94), 6011. https://doi.org/10.21105/joss.06011

Tyrrell, J., Smith, D. J., & Dyson, R. J. (2019). Regularized stokeslet rings: An efficient method for axisymmetric Stokes flow with application to the growing pollen tube. *Phys. Rev. Fluids*, *4*(6), 063102. https://doi.org/10.1103/PhysRevFluids.4.063102

Wegener, W. A. (1981). Diffusion coefficients for rigid macromolecules with irregular shapes that allow rotational-translational coupling. *Biopolymers*, *20*(2), 303–326. https://doi.org/10.1002/bip.1981.360200205

Zhao, B., Lauga, E., & Koens, L. (2019). Method of regularized stokeslets: Flow analysis and improvement of convergence. *Phys. Rev. Fluids*, *4*(8), 084104. https://doi.org/10.1103/PhysRevFluids.4.084104