The Journal of Open Source Software

NILT: Numerical Inverse Laplace Transform Methods

Rodolfo Oliveira [1,2¶]

1 Imperial College London, United Kingdom **2** Equinor, Norway ¶ Corresponding author

DOI: 10.21105/joss.10605

Software

- Review
- Repository
- Archive

Editor: Christoph Junghans

Reviewers:

- @Yurlungur
- @the-rccg

Submitted: 23 April 2026
Published: 10 August 2026

Summary

NILT is a C++ header-only library (with Python bindings) that numerically inverts Laplace transforms using three algorithms: the Gaver-Stehfest method (Stehfest, 1970), the fixed Talbot contour (Abate & Whitt, 2006), and the accelerated Fourier series of de Hoog et al. (de Hoog et al., 1982). Users pass any callable $F(s)$ and a time t, the library returns $f(t) = \mathcal{L}^{-1}\{F\}(t)$.

All three algorithms share the same three-argument interface in both C++ and Python:

Python

```
nilt.invert(F, t, method="Stehfest", options={...})
```

C++

```
nilt::invert(F, t, algorithm)
```

The Stehfest algorithm requires that $F(s)$ be real-valued while Talbot and de Hoog operate on complex-valued transforms. Each algorithm exposes tunable parameters (number of terms, tolerance, and contour shift) with defaults that work well for most problems.

The Python bindings are built with pybind11 and accept both scalars and NumPy array arguments. If using Python and evaluating over a list of scalars, it is more efficient to use NumPy arrays as arguments. The library was originally developed for continuous time random walk simulations of reactive transport in porous media (Oliveira et al., 2020, 2023; Oliveira, 2021).

Statement of need

Many problems in physics and engineering are easier to solve in the Laplace domain than in the time domain. Groundwater drawdown, heat conduction in semi-infinite solids, diffusion from spheres and cylinders, and viscoelastic creep are great examples that have closed-form Laplace-domain solutions that are difficult or impossible to invert analytically.

Existing tools are scattered:

- MATLAB's `ilaplace` implements an inverse Laplace transform but it has no access to individual methods or parameters within it, and does not offer an open-source license.
- Python's `mpmath.invertlaplace` provides all three families of methods (and the Cohen method as well), but it is written in pure Python with arbitrary-precision arithmetic. A Python-first implementation is far slower when you need to invert at thousands of points.
- The `ilt` package wraps a single algorithm and provides an implementation that is too tightly integrated to the application (transient spectroscopy).
- No other C++ library packages multiple algorithms behind a common interface.

NILT provides Stehfest, Talbot, and de Hoog in a dependency-free C++ header that compiles with any C++14 toolchain. The Python bindings expose the same compiled code for scripting and prototyping.

State of the field

NILT implements three families of numerical Laplace inversion.

The Stehfest algorithm (Stehfest, 1970) approximates $f(t)$ using a weighted sum of $F(s)$ evaluated at real points on the positive real axis. It is fast and simple but restricted to real-valued transforms. The accuracy of the method degrades for oscillatory or discontinuous $f(t)$.

The fixed Talbot contour, in the formulation of Abate and Whitt (Abate & Whitt, 2006), deforms the Bromwich integral into the complex plane. It converges rapidly for oscillatory functions but requires $F(s)$ to accept complex arguments.

de Hoog et al. (de Hoog et al., 1982) accelerate the Fourier series representation of the inverse transform using a quotient-difference algorithm. Like Talbot, it works with complex $F(s)$ and can handle discontinuities.

These methods differ in which transforms they accept and how they trade speed for accuracy. Packaging them behind one interface makes it easy to cross-check results or swap algorithms when one performs poorly on a given problem. NILT grew out of the author's PhD work on reactive transport modelling with continuous time random walks (Oliveira, 2021), in which different Laplace-domain kernels within the same simulation called for different inverters; the library packages that experience as a reusable component.

Software design

NILT is a header-only C++ library. Including `<nilt.hpp>` pulls in `stehfest.hpp`, `talbot.hpp`, and `dehoog.hpp`. There are no linking step or external dependencies. Each algorithm is a struct with tunable parameters and an `operator()` that accepts any callable $F(s)$ and a time t.

A *free* function `nilt::invert(F, t, algo)` provides a uniform entry point. The entire library is templated so there is no virtual dispatch and no heap allocation.

The Python layer is built with pybind11 and packaged via scikit-build-core. `Stehfest`, `Talbot`, and `DeHoog` mirror their C++ counterparts. The `invert` function accepts NumPy arrays, so evaluating $f(t)$ at a vector of time points is a single call. The Python binding tries to be as shallow as possible, with the NumPy arrays arguments being the main difference between each package.

The repository contains 10 verification functions drawn from Stehfest (Stehfest, 1970) and Abate and Whitt (Abate & Whitt, 2006), plus five worked physics examples from groundwater and transport phenomena:

- Theis well drawdown (Theis, 1935)
- pumping-injection dipole (Bear, 1979)
- sphere and cylinder diffusion (Crank, 1975)
- 2-D advection-diffusion plume (Bear, 1979)

Each example exists as both a C++ executable and a Python script that compare all three algorithms against the known analytical solution.

Installation and usage

The Python package is published on PyPI as `nilt-python`:

```
pip install nilt-python
```

The C++ headers install through CMake and can be consumed with `find_package(nilt REQUIRED)`; full instructions are in the repository README.

A minimal Python example inverts $F(s) = 1/(s+1)$ (analytical $f(t) = e^{-t}$) at a vector of times:

```python
import numpy as np
import nilt

F = lambda s: 1.0 / (s + 1.0)
t = np.linspace(0.1, 5.0, 50)
f = nilt.invert(F, t, method="Talbot", options={"N": 64})
```

Swapping method is a one-word change: `method="Stehfest"` or `method="DeHoog"`. The corresponding C++ call is `nilt::invert(F, t, nilt::Talbot{})`. More runnable examples (groundwater, transport, verification) are available in `examples/` and `verification/`.

Verification

The verification suite evaluates all three methods against 10 known Laplace transform pairs. Table 1 shows one of these functions, $f(t) = 1/\sqrt{\pi t}$, from Stehfest (1970). Figure 1 plots the inversions alongside the error. All three methods recover the analytical solution, but their error characteristics differ: Stehfest errors are around 10^{-6}, while Talbot and de Hoog reach 10^{-12} or better.

Table 1: Benchmark results for $f(t) = 1/\sqrt{\pi t}$, $F(s) = 1/\sqrt{s}$

t	f(t)	Stehfest	err	Talbot	err	de Hoog	err
1	5.642e-01	5.642e-01	2.17e-06	5.642e-01	4.63e-12	5.642e-01	1.73e-13
2	3.989e-01	3.989e-01	4.92e-06	3.989e-01	4.82e-12	3.989e-01	2.70e-14
5	2.523e-01	2.523e-01	4.24e-06	2.523e-01	4.87e-12	2.523e-01	5.06e-14
10	1.784e-01	1.784e-01	5.70e-06	1.784e-01	4.84e-12	1.784e-01	6.02e-14

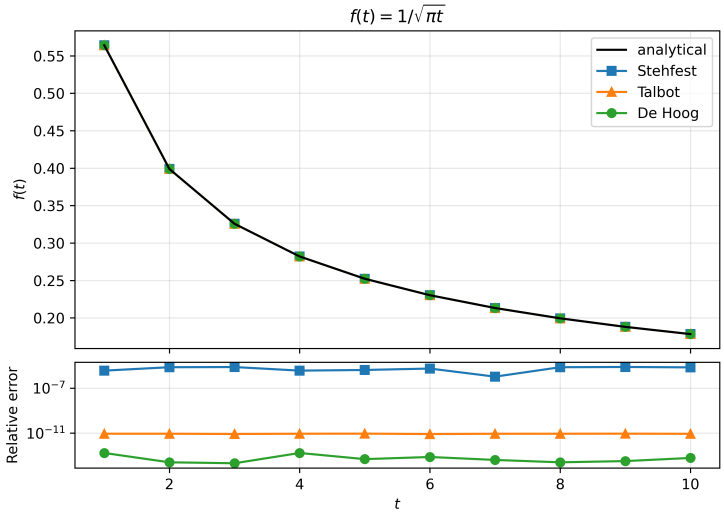

Figure 1: Verification of all three algorithms against $f(t) = 1/\sqrt{\pi t}$. Top: numerical inversions overlaid on the analytical solution. Bottom: relative error on a log scale.

Figure 2 shows timing per inversion call as a function of the algorithm parameter (N, n, or M) for $F(s) = 1/(s+1)$ at $t = 1$. Solid lines are C++, dashed lines are Python. The Python bindings add minimal overhead since the inversion itself runs in compiled C++. Stehfest is the cheapest at low parameter counts, de Hoog becomes the most expensive at high orders. Talbot sits in between.

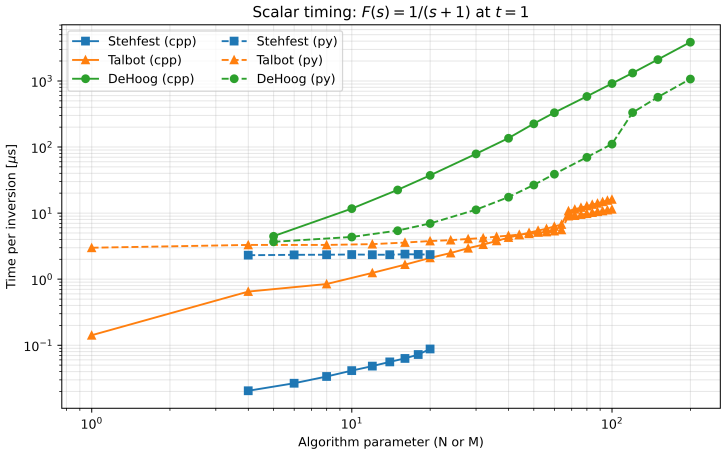

Figure 2: Time per inversion (μs) vs algorithm parameter for C++ (solid) and Python (dashed). The Python overhead is small because the inversion runs in compiled code.

Selected example: groundwater well dipole

The well dipole example (Figure 3, Figure 4) shows NILT applied to a 2-D groundwater problem with no radial symmetry. A pumping well ($Q_p = 0.010$ m^3/s) and an injection well ($Q_i = 0.007$ m^3/s) are placed at different locations in a confined aquifer ($T = 10^{-3}$ m^2/s, $S = 10^{-4}$). The Laplace-domain solution is a superposition of K_0 Bessel functions, inverted pointwise at each (x, y) grid node.

Figure 3 compares the numerical inversions against the analytical E_1 solution over time at a fixed observation point. All three methods track the analytical curve, Stehfest shows larger relative error at early times (around 10^{-1}) while Talbot and de Hoog stay below 10^{-10}.

Figure 4 shows the net drawdown field at $t = 2$ h. The pumping well (bottom-left) draws water down, the injection well (top-right) pushes it up, and the asymmetric contours reflect their different rates. This is the kind of spatial field that requires thousands of pointwise inversions, when a compiled backend matters.

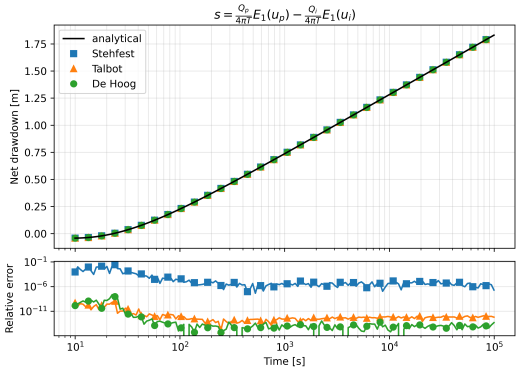

Figure 3: Net drawdown vs time for the well dipole. Top: all three algorithms overlaid on the analytical solution. Bottom: relative error.

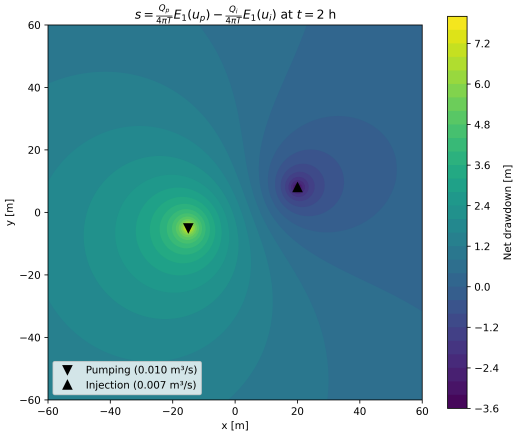

Figure 4: Spatial drawdown field at $t = 2$ h for the pumping-injection dipole. The pumping well (▼) and injection well (▲) produce an asymmetric drawdown pattern.

Research impact

NILT was developed during a PhD on reactive transport modelling using continuous time random walks (Oliveira, 2021) and derived works (Oliveira et al., 2020, 2023), where the Laplace inversion routines were used to compute first-passage time distributions and breakthrough curves in heterogeneous porous media. The library has since been extracted and released as standalone software.

AI usage disclosure

AI tools were used for code refactoring, and documentation editing. All scientific content, algorithm implementations, and mathematical formulations are the author's own work.

References

Abate, J., & Whitt, W. (2006). A unified framework for numerically inverting Laplace transforms. *INFORMS Journal on Computing*, *18*(4), 408–421. https://doi.org/10.1287/ijoc.1050.0137

Bear, J. (1979). *Hydraulics of groundwater*. McGraw-Hill.

Crank, J. (1975). *The mathematics of diffusion* (2nd ed.). Oxford University Press.

de Hoog, F. R., Knight, J. H., & Stokes, A. N. (1982). An improved method for numerical inversion of Laplace transforms. *SIAM Journal on Scientific and Statistical Computing*, *3*(3), 357–366. https://doi.org/10.1137/0903022

Oliveira, R. (2021). *Modelling of reactive transport in porous media using continuous time random walks* [PhD thesis]. https://doi.org/10.25560/92253

Oliveira, R., Bijeljic, B., Blunt, M. J., Colbourne, A., Sederman, A. J., Mantle, M. D., & Gladden, L. F. (2020). A continuous time random walk method to predict dissolution in porous media based on validation of experimental NMR data. *Chemical Engineering Science*. https://doi.org/10.1016/j.advwatres.2021.103847

Oliveira, R., Blunt, M. J., & Bijeljic, B. (2023). Impact of physical heterogeneity and transport conditions on effective reaction rates in dissolution. *Transport in Porous Media*, *146*, 113–138. https://doi.org/10.1007/s11242-022-01836-x

Stehfest, H. (1970). Algorithm 368: Numerical inversion of Laplace transforms [D5]. *Communications of the ACM*, *13*(1), 47–49. https://doi.org/10.1145/361953.361969

Theis, C. V. (1935). The relation between the lowering of the piezometric surface and the rate and duration of discharge of a well using ground-water storage. *Transactions of the American Geophysical Union*, *16*(2), 519–524. https://doi.org/10.1029/TR016i002p00519