HODLRlib Documentation

Release 3.1415

Sivaram Ambikasaran, Shyam Sundar Sankaran

Jul 16, 2020

Contents:

CHAPTER 1

About `HODLRlib`:

`HODLRlib` is a library consisting of fast matrix operations for matrices based on the Hierarchical Off-Diagonal Low-Rank (HODLR) structure. In the current version, the operations available are the matrix multiplication, solve, determinant computation and symmetric factorization.

This software is an optimized implementation and the extension of these articles [1] [2], with the current version showing a substantial increase in speed (a few orders of magnitude) over the timings reported in these articles. The solver has also been extended to matrices not necessarily arising out of kernels and higher dimensions. Low-rank approximation of the appropriate blocks is obtained using rook pivoting. The domain is subdivided based on a KDTree. The solver is fairly general, works with minimal restrictions and has been parallelized using OpenMP

The code is written in C++ and features an easy-to-use interface, where the user provides input through a `kernel` object which abstracts data of the matrix through a member function `getMatrixEntry(int i, int j)` which returns the entry at the i^{th} row and j^{th} column of the matrix.

The current release has the following capabilities:

- MatVecs: Obtains Ax at a cost of $\mathcal{O}\left(N \log N\right)$
- Factorization: Factors the matrix A into the desired form at a cost of $\mathcal{O}\left(N \log^2\left(N\right)\right)$
- Cholesky-like Symmetric Factorization: Obtains $A = WW^T$ at a cost of $\mathcal{O}\left(N \log^2\left(N\right)\right)$
- Solve: Solves linear systems $Ax = b$ at an additional cost of $\mathcal{O}\left(N \log\left(N\right)\right)$
- Determinant Computation: Additional Cost of $\mathcal{O}\left(N \log N\right)$

CHAPTER 2

Doc Contents

2.1 Installation and Building

2.1.1 Downloading the Source

`HODLRlib` is distributed using the git version control system, and is hosted on Github. The repository can be cloned using:

```
git clone https://github.com/sivaramambikasaran/HODLR.git --recursive
```

The `--recursive` flag is argument to ensure that all submodules are also checked out. The Python wrappers for the library make use of `pybind11` and would need to be checked out for Python functionality.

2.1.2 Dependencies

- Eigen Linear Algebra Library (get it here)
- (optional) An OpenMP enabled compiler (e.g. gcc4.2 or above) is required to use shared-memory parallelism.
- (optional) MKL libraries (`HODLRlib` has improved performance when compiled against MKL)

NOTE: On MacOS, the default compiler is *clang* which doesn't have OpenMP support. You will have to use g++ to make use of the speedups from OpenMP:

```
user@computer HODLR$ brew install g++-8
user@computer HODLR$ export CXX=g++
```

2.1.3 Installation

You can either install `HODLRlib` by using the provided install script provided or manually install and link the needed dependencies.

Install Script

The easiest way to get running is to install the needed dependencies by running the `install.sh` provided in the root level of this repository:

```
user@computer HODLR$ ./install.sh
```

The above command should create a folder `deps/` in the current directory with the needed dependencies. Additionally, the script should set the environment variables that would be needed during the build and execution stages. This only needs to be done once since the environment variables are automatically written to `.bash_profile`.

Manually Installing

First set the environment variable `HODLR_PATH` to the root level of this repository. This is needed by some of the routines in the plotting of the low-rank structure for the specific kernel. (**NOTE**: The plotting is carried out using python, and requires the matplotlib package to be installed in your python environment)

Then, set the environment variable `EIGEN_PATH` to the location of your Eigen installation. This is needed by the CMake script.:

```
user@computer HODLR$ export EIGEN_PATH=path/to/eigen/
```

Optionally: set the environment variable `MKLROOT` to take advantage of speedups from MKL.:

```
user@computer HODLR$ export MKLROOT=path/to/mkl/
```

Python Installation

Get the dependencies and set environment variables needed for the library through the Install Script of Manual Install routes. Once these have been setup, navigate to the `python` subdirectory from the root level of the repository. The Python wrapper for the library can be installed by running:

```
user@computer python$ python setup.py install
```

2.1.4 Testing

C++

Now, we need to ensure that all the functions of the libraries function as intended. For this purpose, we will be running the script `test/test_HODLR.cpp`. By default, during a build this file under `test/` gets compiled, and would show up under the `test/` directory in your build folder. To check this on your computer, run the following lines:

```
user@computer HODLR$ mkdir build && cd build
user@computer build$ cmake ..
user@computer build$ ./test/test_HODLR
```

For a succesful test, the final line of output for this run would read:"Reached End of Test File Successfully! All functions work as intended!".

Fortran

In order to ensure that the Fortran wrapper works as intended, run the Fortran file found under `fortran/test/test_HODLR.f90`. In order to run this and ensure correct functioning, type the following at the commandline:

```
user@computer HODLR$ cd fortran
user@computer fortran$ mkdir build && cd build
user@computer build$ cmake .. -DCMAKE_BUILD_TYPE=TESTING
user@computer build$ ./test_HODLR
```

For a succesful test, the final line of output for this run would read:"Reached End of Test File Successfully! All functions work as intended!".

Python

In order to ensure that the Python wrapper works as intended, run the Python file found under `python/test/test_HODLR.py`. This can be executed by running the following commands from the terminal:

```
user@computer HODLR$ cd python/test
user@computer test$ python test_HODLR.py
```

For a succesful test, the final line of output for this run would read:"Reached End of Test File Successfully! All functions work as intended!".

2.1.5 Building and Executing

C++

Key in the required `.cpp` to be used as input under `INPUT_FILE` in `HODLRlib/CMakeLists.txt`. Here you also set the name of the output executable under `OUTPUT_EXECUTABLE_NAME`. Then navigate to your build directory and run `cmake path/to/CMakeLists.txt` and run the generated `Makefile` to get your executable:

```
user@computer build$ cmake path/to/HODLR/
user@computer build$ make -j n_threads
user@computer build$ ./executable
```

Fortran

The compilation process for the fortran wrappers is similar to the process for C++: Key in the required `.f90` to be used as input under `INPUT_FILE` in `HODLRlib/fortran/CMakeLists.txt`. Here you also set the name of the output executable under `OUTPUT_EXECUTABLE_NAME`. Then navigate to your build directory and run `cmake path/to/CMakeLists.txt` and run the generated `Makefile` to get your executable:

```
user@computer build$ cmake path/to/HODLR/fortran/
user@computer build$ make -j n_threads
user@computer build$ ./executable
```

Python

The process of running Python files that make use of `pyhodlrlib` follow the standard procedure used to run a python file:

```
user@computer example$ python example.py
```

2.2 Tutorial

2.2.1 C++

For the sake of this tutorial, we are going to be using the `tutorial.cpp` file that is listed under `examples/` since it demonstrates all the features of this library. For the most part, comments in the file demonstrate intended functionality. However, we go over the main functions that may be of interest to a user on this page.

NOTE: It is assumed that you have already completed the installation process of getting the dependencies.

Setting Parameters in CMakeLists.txt

There are some variables that need to be set by the user at the top of the `CMakeLists.txt` file:

- `INPUT_FILE`: This is the input `.cpp` file that needs to be compiled. For this tutorial, it's going to be set to `examples/tutorial.cpp`.
- `OUTPUT_EXECUTABLE`: This is the name that the final build executable is given. Here we are just going to set is as `tutorial`.
- `DTYPE`: Datatype that is used in all the computations. Can be set to `float`, `double`, `complex32` and `complex64`. We are going to be using `double` for this tutorial.

Creating a Derived Class of `HODLR_Matrix`:

The matrix that needs to be solved for is abstracted through this derived class of `HODLR_Matrix`. For the sake of the tutorial, we are calling this derived class `Kernel`. The main method that needs to be set for this class is `getMatrixEntry` which returns the entry at the i^{th} row and j^{th} column of the matrix. For instance, for the Hilbert matrix of size $N \times N$, this would be set as:

```
class Kernel : public HODLR_Matrix
{
    public:

        Kernel(int N) : HODLR_Matrix(N){}

        dtype getMatrixEntry(int i, int j)
        {
            return (1. / (i + j + 1));
        }
}
```

Note that here `dtype` is set during compilation depending on `DTYPE` that was set in `CMakeLists.txt`.

In this tutorial, we have initialized a random set of points in $(-1, 1)$ which are then sorted to obtain a coordinate vector x. Using this, we compute the distance between the i^{th} point and j^{th} in x to obtain R which can then be used in different kernel functions:

```
class Kernel : public HODLR_Matrix
{
```

(continues on next page)

(continued from previous page)

```
    private:
        Mat x;

    public:

        // Constructor:
        Kernel(int N, int dim) : HODLR_Matrix(N)
        {
            // Getting the random distribution of points:
            x = (Mat::Random(N, dim)).real();
            // This is being sorted to ensure that we get optimal low rank structure:
            getKDTreeSorted(x, 0);
        };

        // In this example, we are illustrating usage of the gaussian kernel:
        dtype getMatrixEntry(int i, int j)
        {
            size_t dim = x.cols();

            // Value on the diagonal:
            if(i == j)
            {
                return 10;
            }

            // Otherwise:
            else
            {
                dtype R2 = 0;

                for(int k = 0; k < dim; k++)
                {
                    R2 += (x(i,k) - x(j,k)) * (x(i,k) - x(j,k));
                }

                return exp(-R2);
            }
        }
}
```

Creating the Instance of `HODLR_Tree`:

The main operations of this library are carried out through the `HODLR_Tree` class. The parameters that are taken for the constructor are the number of levels, tolerance for approximation and the earlier created instance of `Kernel`:

```
Kernel* K      = new Kernel(N, dim);
HODLR_Tree* T = new HODLR_Tree(n_levels, tolerance, K);
```

We will now proceed to demonstrate the individual methods available under this class.

`assembleTree`

We proceed to call the `assembleTree` method. This obtains the complete matrix for the leaf levels and the low-rank approximation for the off-diagonal blocks. Here we have used mentioned the fact that the matrix that we are constructing is both symmetric and positive-definite. Note that when we mention that the matrix is symmetric and positive-definite, the fast symmetric factorization method would be used. In all other cases the fast factorization method gets used:

```
bool is_sym = true;
bool is_pd = true;
T->assembleTree(is_sym, is_pd);
```

plotTree

This function is used to visualize the rank structure of the matrix encoded through the defined `Kernel` object. It's useful to build a visual understanding of the "low-rankness" of the sub-blocks of the matrix. This function takes the filename and extension of the output image as a string:

```
T->plotTree("plot.svg");
```

For instance, with the gaussian kernel with $N = 1000$, $M = 100$ and tolerance $\epsilon = 10^{-12}$, we obtain this image:

If consider the RPY Tensor of $\texttt{dim} = 1$ for the same parameters, we get this image:

It is easy to see that the gaussian kernel shows a much "stronger" low rank nature than the RPY tensor.

printTreeDetails

This is a function which is mainly used in the process of development to understand how the nodes are being assigned in the tree. It is a great utility function to understand all the details of the nodes in the tree. For instance, the gaussian kernel when using $N = 1000$, $M = 100$ and tolerance $\epsilon = 10^{-12}$ gives this output:

```
Level Number        :0
Node Number         :0
Start of Node       :0
Size of Node        :1000
Tolerance           :1e-12
Left Child:
Start of Child Node:0
Size of Child Node :500
Right Child:
Start of Child Node:500
Size of Child Node :500
Shape of U[0]       :500, 11
Shape of U[1]       :500, 11
Shape of V[0]       :500, 11
Shape of V[1]       :500, 11
Shape of K          :0, 0
=================================================================================================


Level Number        :1
Node Number         :0
Start of Node       :0
Size of Node        :500
Tolerance           :1e-12
```

(continues on next page)

(continued from previous page)

```
Left Child:
Start of Child Node:0
Size of Child Node :250
Right Child:
Start of Child Node:250
Size of Child Node :250
Shape of U[0]      :250, 8
Shape of U[1]      :250, 8
Shape of V[0]      :250, 8
Shape of V[1]      :250, 8
Shape of K         :0, 0
==========================================================================================================
Level Number       :1
Node Number        :1
Start of Node      :500
Size of Node       :500
Tolerance          :1e-12
Left Child:
Start of Child Node:500
Size of Child Node :250
Right Child:
Start of Child Node:750
Size of Child Node :250
Shape of U[0]      :250, 10
Shape of U[1]      :250, 10
Shape of V[0]      :250, 10
Shape of V[1]      :250, 10
Shape of K         :0, 0
==========================================================================================================


Level Number       :2
Node Number        :0
Start of Node      :0
Size of Node       :250
Tolerance          :1e-12
Left Child:
Start of Child Node:0
Size of Child Node :125
Right Child:
Start of Child Node:125
Size of Child Node :125
Shape of U[0]      :125, 6
Shape of U[1]      :125, 6
Shape of V[0]      :125, 6
Shape of V[1]      :125, 6
Shape of K         :0, 0
==========================================================================================================
Level Number       :2
Node Number        :1
Start of Node      :250
Size of Node       :250
Tolerance          :1e-12
Left Child:
Start of Child Node:250
Size of Child Node :125
Right Child:
```

(continues on next page)

(continued from previous page)

```
Start of Child Node:375
Size of Child Node :125
Shape of U[0]      :125, 8
Shape of U[1]      :125, 8
Shape of V[0]      :125, 8
Shape of V[1]      :125, 8
Shape of K         :0, 0
====================================================================================================
Level Number       :2
Node Number        :2
Start of Node      :500
Size of Node       :250
Tolerance          :1e-12
Left Child:
Start of Child Node:500
Size of Child Node :125
Right Child:
Start of Child Node:625
Size of Child Node :125
Shape of U[0]      :125, 8
Shape of U[1]      :125, 8
Shape of V[0]      :125, 8
Shape of V[1]      :125, 8
Shape of K         :0, 0
====================================================================================================
Level Number       :2
Node Number        :3
Start of Node      :750
Size of Node       :250
Tolerance          :1e-12
Left Child:
Start of Child Node:750
Size of Child Node :125
Right Child:
Start of Child Node:875
Size of Child Node :125
Shape of U[0]      :125, 9
Shape of U[1]      :125, 9
Shape of V[0]      :125, 9
Shape of V[1]      :125, 9
Shape of K         :0, 0
====================================================================================================


Level Number       :3
Node Number        :0
Start of Node      :0
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:0
Size of Child Node :62
Right Child:
Start of Child Node:62
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
```

(continues on next page)

(continued from previous page)

```
Shape of V[0]       :0, 0
Shape of V[1]       :0, 0
Shape of K          :125, 125
=====================================================================================================
Level Number        :3
Node Number         :1
Start of Node       :125
Size of Node        :125
Tolerance           :1e-12
Left Child:
Start of Child Node:125
Size of Child Node :62
Right Child:
Start of Child Node:187
Size of Child Node :63
Shape of U[0]       :0, 0
Shape of U[1]       :0, 0
Shape of V[0]       :0, 0
Shape of V[1]       :0, 0
Shape of K          :125, 125
=====================================================================================================
Level Number        :3
Node Number         :2
Start of Node       :250
Size of Node        :125
Tolerance           :1e-12
Left Child:
Start of Child Node:250
Size of Child Node :62
Right Child:
Start of Child Node:312
Size of Child Node :63
Shape of U[0]       :0, 0
Shape of U[1]       :0, 0
Shape of V[0]       :0, 0
Shape of V[1]       :0, 0
Shape of K          :125, 125
=====================================================================================================
Level Number        :3
Node Number         :3
Start of Node       :375
Size of Node        :125
Tolerance           :1e-12
Left Child:
Start of Child Node:375
Size of Child Node :62
Right Child:
Start of Child Node:437
Size of Child Node :63
Shape of U[0]       :0, 0
Shape of U[1]       :0, 0
Shape of V[0]       :0, 0
Shape of V[1]       :0, 0
Shape of K          :125, 125
=====================================================================================================
Level Number        :3
Node Number         :4
```

(continues on next page)

(continued from previous page)

```
Start of Node      :500
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:500
Size of Child Node :62
Right Child:
Start of Child Node:562
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
Shape of V[0]      :0, 0
Shape of V[1]      :0, 0
Shape of K         :125, 125
==========================================================================================================
Level Number       :3
Node Number        :5
Start of Node      :625
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:625
Size of Child Node :62
Right Child:
Start of Child Node:687
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
Shape of V[0]      :0, 0
Shape of V[1]      :0, 0
Shape of K         :125, 125
==========================================================================================================
Level Number       :3
Node Number        :6
Start of Node      :750
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:750
Size of Child Node :62
Right Child:
Start of Child Node:812
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
Shape of V[0]      :0, 0
Shape of V[1]      :0, 0
Shape of K         :125, 125
==========================================================================================================
Level Number       :3
Node Number        :7
Start of Node      :875
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:875
Size of Child Node :62
```

(continues on next page)

(continued from previous page)

```
Right Child:
Start of Child Node:937
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
Shape of V[0]      :0, 0
Shape of V[1]      :0, 0
Shape of K         :125, 125
===========================================================================================================
```

printNodeDetails

This function is useful if we want to find the details of a particular node in the tree. This function takes in the arguments of the level number and node number of the node you want to query. For instance if we call `T->printNodeDetails(3, 7)` for the above defined tree structure, we get:

```
Level Number       :3
Node Number        :7
Start of Node      :875
Size of Node       :125
Tolerance          :1e-12
Left Child:
Start of Child Node:875
Size of Child Node :62
Right Child:
Start of Child Node:937
Size of Child Node :63
Shape of U[0]      :0, 0
Shape of U[1]      :0, 0
Shape of V[0]      :0, 0
Shape of V[1]      :0, 0
Shape of K         :125, 125
```

matmatProduct

This function is used to obtain the matrix-matrix / matrix-vector product of the given matrix / vector x, with the matrix that is abstracted by the instance of `Kernel`:

```
b = T->matmatProduct(x);
```

factorize

Depends upon whether we intend to perform fast factorization or fast symmetric factorization:

- **Fast Factorization** - This function performs the factorizations such that the matrix is obtained as $K = K_\kappa K_{\kappa-1}...K_1K_0$ where K_i are block diagonal matrices with κ being the number of levels considered.
- **Fast Symmetric Factorization** - This function performs the factorizations such that the matrix is obtained as $K = \underbrace{K_\kappa K_{\kappa-1}...K_1K_0}_{W}\underbrace{K_0^TK_1^T...K_{\kappa-1}^TK_\kappa^T}_{W^T}$ where K_i are block diagonal matrices with κ being the number of levels considered.

For more details on this factorization refer to the articles [1] [2]

```
T->factorize();
```

solve

Applies the inverse of the matrix(abstracted by the `Kernel` object) on the given vector x. This must be called only after `factorize` has been called:

```
x = T->solve(b);
```

logDeterminant

Returns the log of the determinant of the matrix that has been described through the `Kernel` object:

```
dtype log_det = T->logDeterminant();
```

symmetricFactorProduct

If the matrix described through the `Kernel` object is a covariance matrix Q it can be expressed as $Q = WW^T$. If we create a random normal vector x i.e $\mathcal{N}(\mu = 0, \sigma = 1)$, then the random vector y with covariance matrix Q is given by $y = Wx$:

```
y = T->symmetricFactorProduct(x);
```

symmetricFactorTransposeProduct

This function returns the product of the transpose of the symmetric factor with the given vector x (i.e it returns $W^T x$):

```
y = T->symmetricFactorTransposeProduct(x);
```

getSymmetricFactor

Explicitly builds and returns the symmetric factor W:

```
W = T->getSymmetricFactor();
```

Running the Program:

For this particular tutorial, the problem parameters are passed to the executable during runtime. In the beginning of this file, we have the lines:

```
// Size of the Matrix in consideration:
int N             = atoi(argv[1]);
// Size of Matrices at leaf level:
int M             = atoi(argv[2]);
// Dimensionality of the problem:
```

(continues on next page)

(continued from previous page)

```
int dim            = atoi(argv[3]);
// Tolerance of problem
double tolerance  = pow(10, -atoi(argv[4]));
```

This means that the first argument would be the matrix size considered, the second one would be the size at the leaf level, the third one would be the dimensionality considered and the final argument is approximately the number of digits of accuracy we want. For instance, running `./tutorial 1000 100 1 12` would correspond to solving the problem with parameters $N = 1000, M = 100, \texttt{dim} = 1, \epsilon = 10^{-12}$.

2.2.2 Fortran

This is where details for the tutorial of the Fortran wrapper for HODLRlib will come in

2.2.3 Python

This is where details for the tutorial of the Python wrapper for HODLRlib will come in

2.3 Benchmarks

All the following benchmarks have been carried out on an i7-8750H(with OpenMP enabled, this is 12 threads), with Intel's icpc (ICC) 19.0.1.144 compiler and Eigen version 3.3.7, with `DTYPE` set to `double`. The compiler flags that were utilized are the same are those mentioned in the `CMakeLists.txt` file.

Presented below are the results as obtained when using different kernels:

2.3.1 Gaussian Kernel

The Gaussian Kernel is given by $K(i, j) = \sigma^2\delta_{ij}^2 + \exp(-||x_i - x_j||^2)$. For these benchmarks, we take $\sigma = 10$ with x being set as a sorted random vector $\in (-1, 1)$. Using the `plotTree` function of this library, we can look at the rank structure for this matrix. The following diagram is obtained with $N = 10000$, $M = 500$ and tolerance 10^{-12}

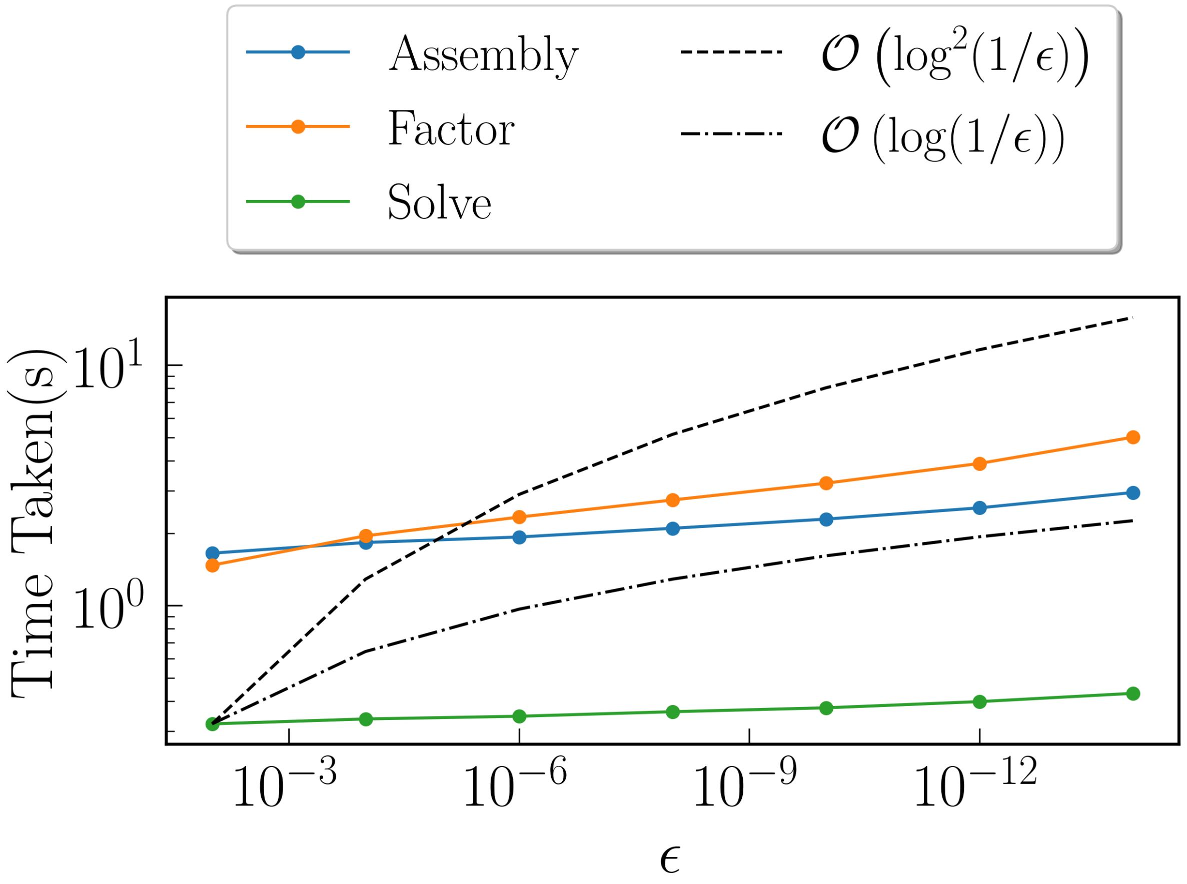
The green blocks are low-rank blocks. Their intensity of colour shows their degree of "low-rankness". Additionally, the rank has been displayed in each of these blocks. The red blocks are full-rank blocks and would have the rank of $M = 500$

Time Taken vs Tolerance

These benchmarks were performed for size of the matrix $N = 1000000$, with the size of the leaf node set to $M = 100$.

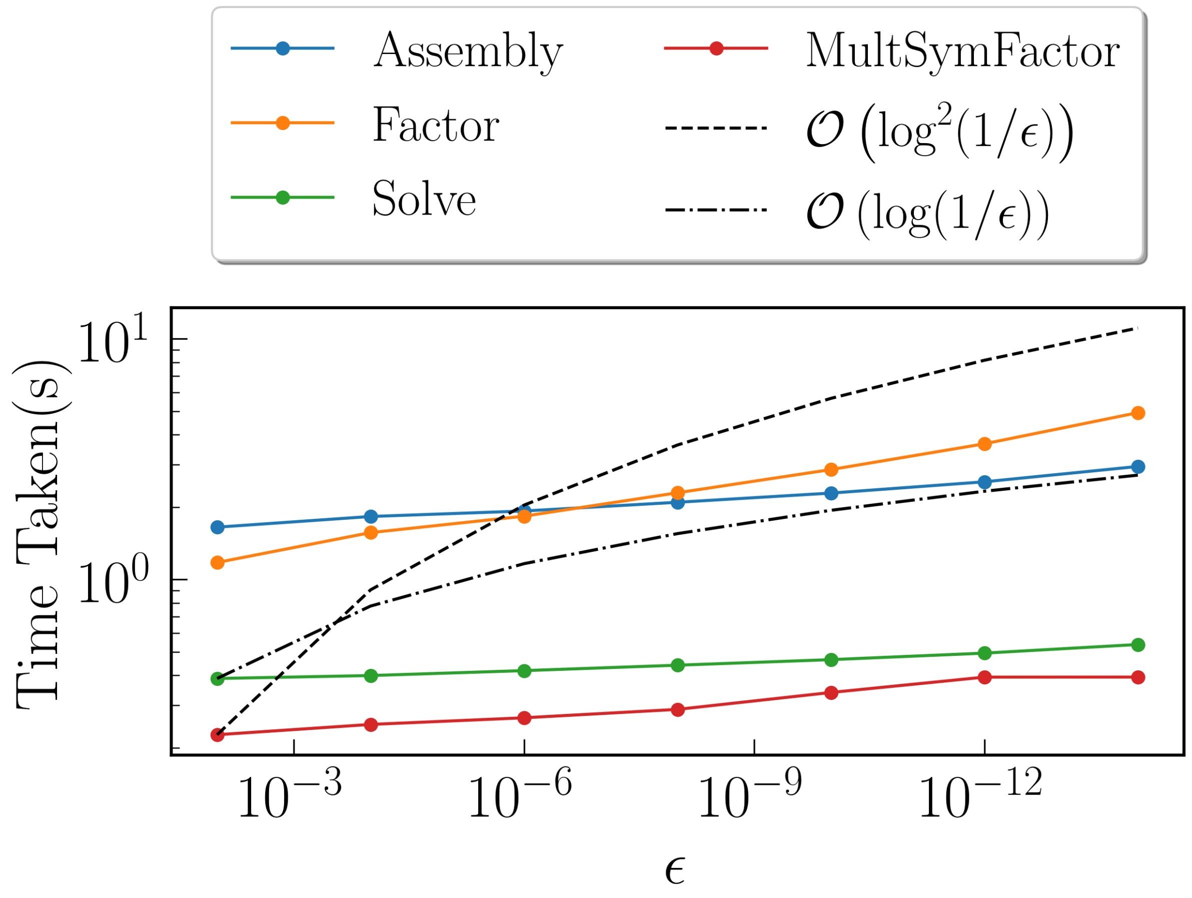
Fast Factorization

Tolerance	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)
10^{-2}	1.65059	11.4442	1.47112	0.321805	0.0300901
10^{-4}	1.82825	8.33693	1.94887	0.337377	0.03039
10^{-6}	1.92681	12.4077	2.33157	0.346198	0.0300648
10^{-8}	2.09475	11.5901	2.74718	0.361579	0.0338411
10^{-10}	2.28711	11.8123	3.22611	0.375279	0.0296249
10^{-12}	2.54764	11.2157	3.89779	0.398319	0.0305111
10^{-14}	2.95124	8.55489	5.01199	0.431082	0.0309851



Fast Symmetric Factorization

Tolerance	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)
10^{-2}	1.61076	11.4741	1.17726	0.387992	0.0366111	0.226509
10^{-4}	1.81511	8.08747	1.56692	0.399085	0.0328679	0.249969
10^{-6}	1.91956	12.3341	1.83361	0.418334	0.031215	0.266352
10^{-8}	2.07343	11.2653	2.29376	0.440591	0.0327439	0.288697
10^{-10}	2.24097	11.7877	2.86431	0.464687	0.0305729	0.339399
10^{-12}	2.57603	11.3027	3.66516	0.494536	0.031522	0.393104
10^{-14}	2.90611	7.89484	4.93738	0.537149	0.030225	0.393285

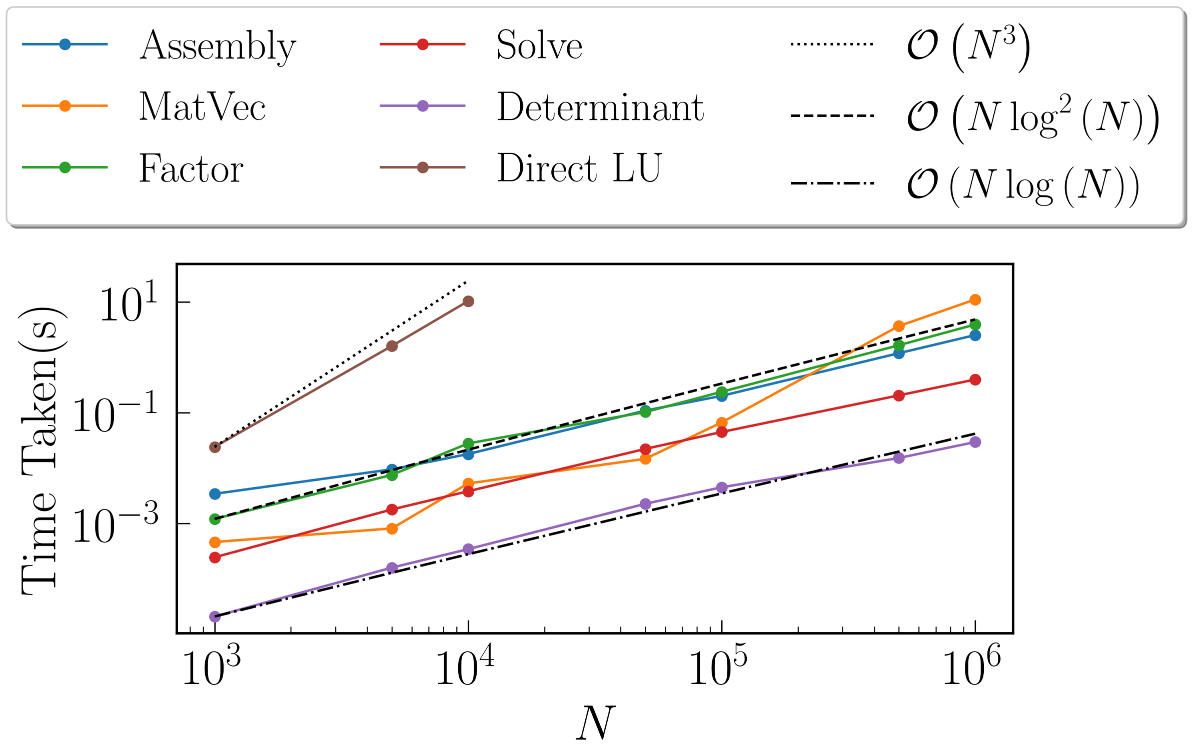


Time Taken vs Size of Matrix

For these benchmarks, the leaf size was fixed at $M = 100$, with tolerance set to 10^{-12}

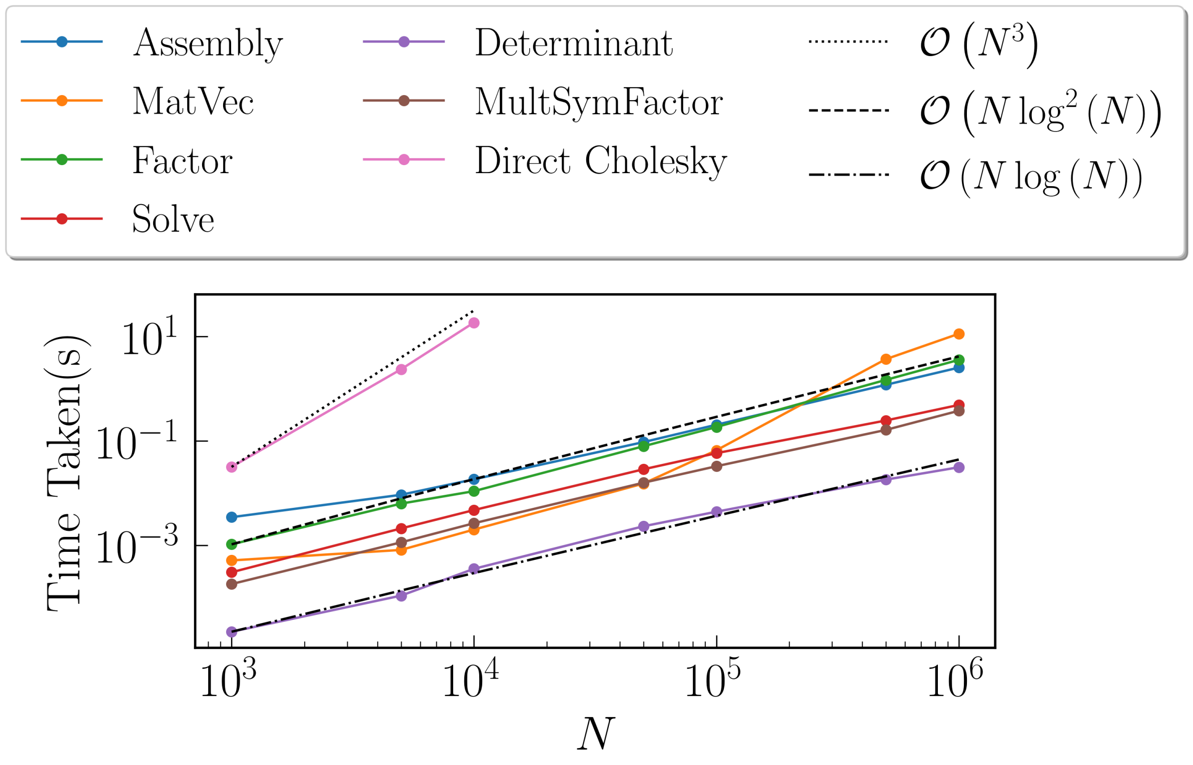
Fast Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	Direct LU(s)
10^3	0.00345016	0.000463963	0.00121403	0.000246048	2.09808e-05	0.024302
5×10^3	0.00954294	0.000818014	0.00755906	0.00179601	0.000159979	1.61282
10^4	0.0180159	0.00202203	0.103507	0.003834	0.000344992	10.4102
5×10^4	0.109816	0.0147851	0.103266	0.022316	0.00227404	N/A
10^5	0.202525	0.066885	0.239639	0.0450559	0.00451112	N/A
5×10^5	1.19365	3.68382	1.6615	0.206754	0.015748	N/A
10^6	2.53519	11.1435	3.93549	0.399695	0.0303771	N/A



Fast Symmetric Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)	Direct Cholesky(s)
10^3	0.00344396	0.000510931	0.00103807	0.000303033	2.19345e-05	0.000180006	0.0316679
5×10^3	0.00925708	0.000812054	0.00626493	0.002094033	0.000108004	0.00113392	2.35399
10^4	0.0183232	0.00199389	0.010865	0.00471711	0.000352859	0.00263691	18.5745
5×10^4	0.0946209	0.0151899	0.0787759	0.0285201	0.00230503	0.0157571	N/A
10^5	0.203769	0.0659761	0.183974	0.058074	0.00438595	0.03263	N/A
5×10^5	1.18639	3.67825	1.47418	0.245743	0.0180571	0.162066	N/A
10^6	2.53567	11.2973	3.56786	0.488049	0.0311899	0.377352	N/A



2.3.2 Matérn Kernel

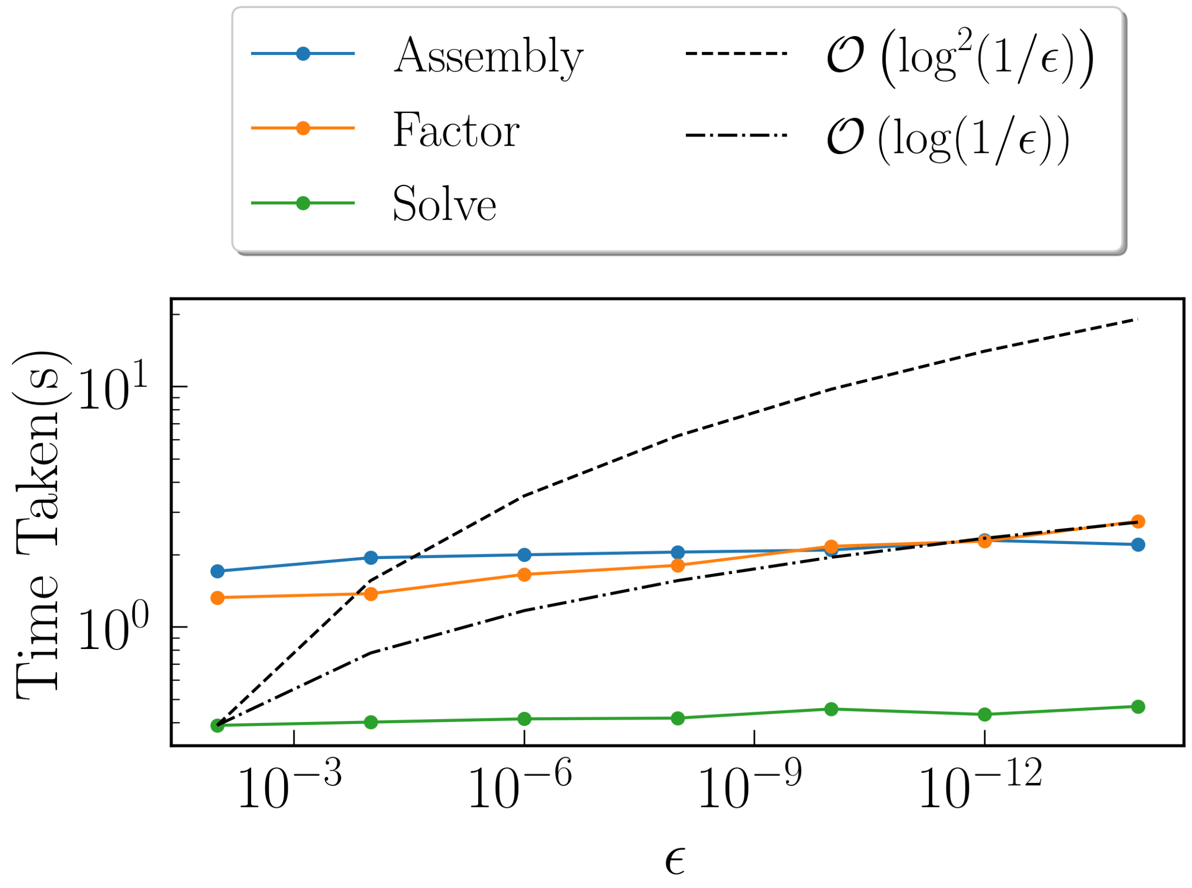
Kernel considered is given by $K(r) = \sigma^2\left(1+\frac{r\sqrt{5}}{\rho}+\frac{5r^2}{3\rho^2}\right)\exp\left(-\frac{r\sqrt{5}}{\rho}\right)$. For these benchmarks, we take $\sigma = 10$, $\rho = 5$, where $r = ||x_i - x_j||$ with x being set as a sorted random vector $\in (-1, 1)$. Using `plotTree` for $N = 10000$, $M = 500$ and tolerance 10^{-12}, we see this rank structure

Time Taken vs Tolerance

These benchmarks were performed for size of the matrix $N = 1000000$, with the size of the leaf node set to $M = 100$.

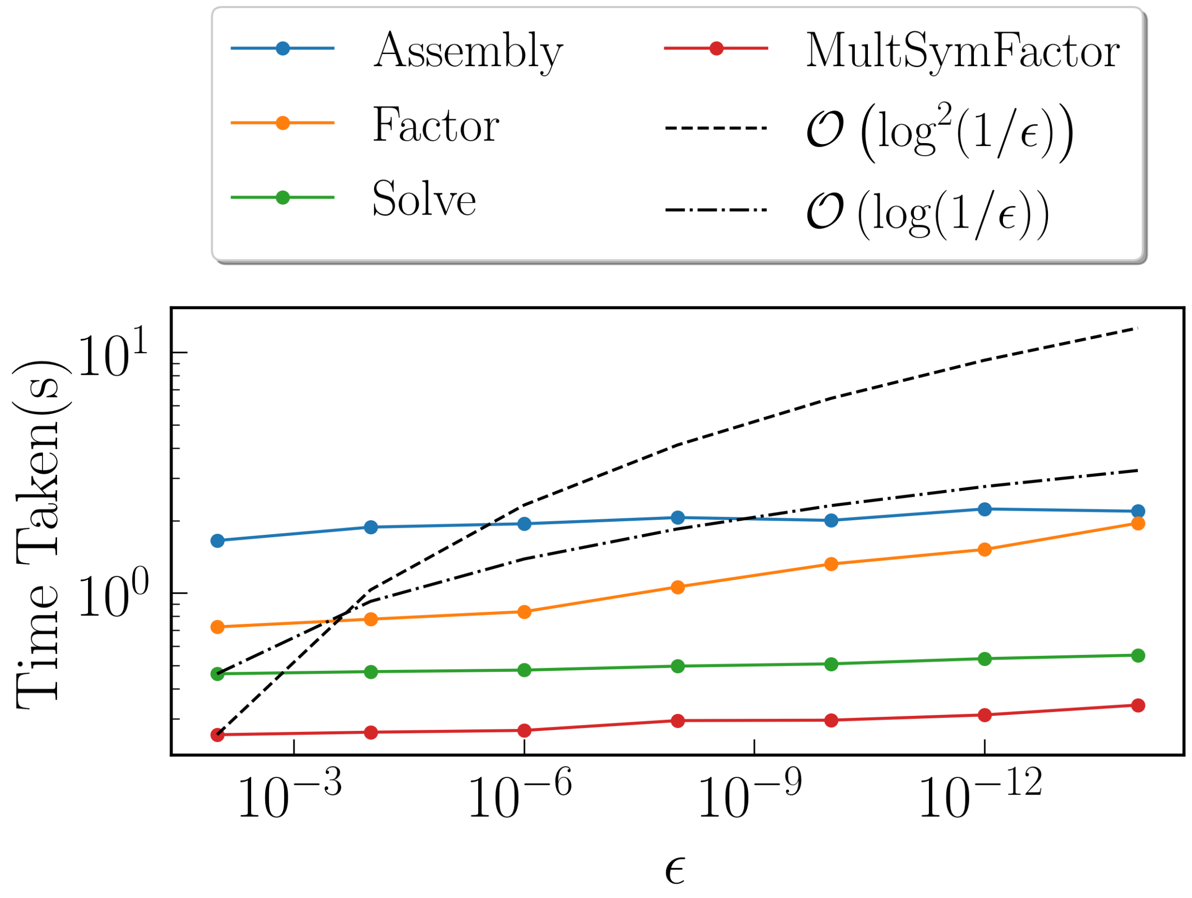
Fast Factorization

Tolerance	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)
10^{-2}	1.70237	13.8247	1.3231	0.388983	0.042177
10^{-4}	1.93746	14.0274	1.37327	0.401342	0.0430369
10^{-6}	1.99264	9.29146	1.6509	0.413971	0.0420959
10^{-8}	2.04502	13.6249	1.80135	0.417019	0.043962
10^{-10}	2.08538	14.7541	2.1616	0.455189	0.0420899
10^{-12}	2.28954	9.11655	2.27049	0.431815	0.043808
10^{-14}	2.19898	13.821	2.74798	0.466761	0.0431418



Fast Symmetric Factorization

Tolerance	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)
10^{-2}	1.65146	13.4722	0.722689	0.461396	0.0417881	0.257583
10^{-4}	1.87788	13.6014	0.778202	0.471056	0.041806	0.263908
10^{-6}	1.93905	8.81335	0.836078	0.478072	0.0427818	0.268437
10^{-8}	2.05821	13.4592	1.05975	0.496589	0.0437939	0.294927
10^{-10}	2.0032	14.3409	1.31922	0.507549	0.0424139	0.296023
10^{-12}	2.23442	8.84984	1.51609	0.533495	0.0427949	0.311331
10^{-14}	2.18632	13.6219	1.95092	0.551657	0.0439069	0.342182

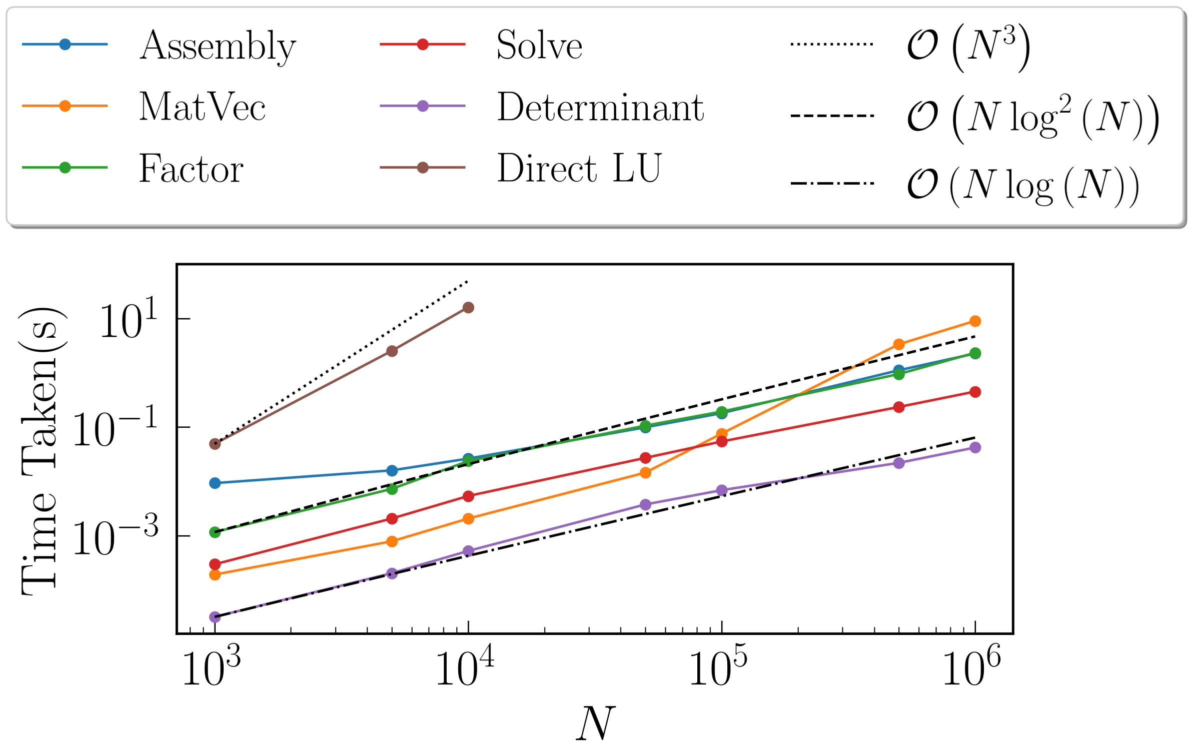


Time Taken vs Size of Matrix

For these benchmarks, the leaf size was fixed at $M = 100$, with tolerance set to 10^{-12}

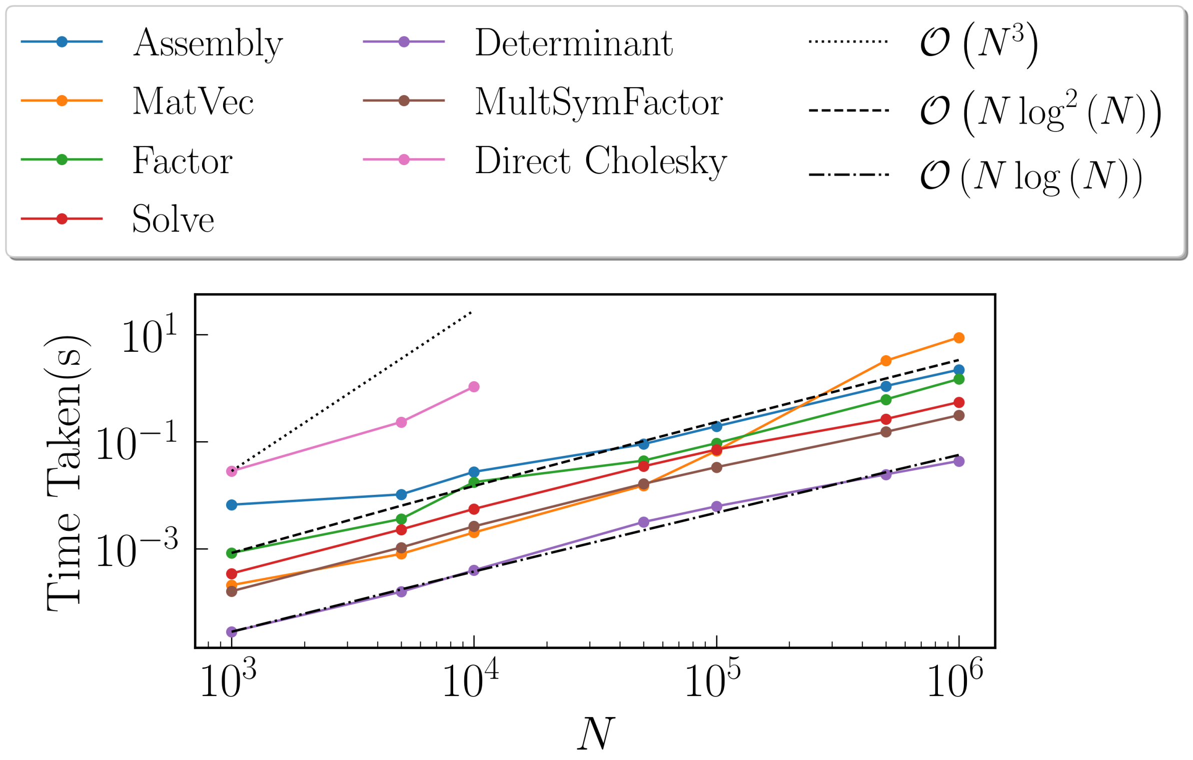
Fast Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	Direct LU(s)
10^3	0.00927687	0.0001921	0.0011642	0.000297	3.19481e-05	0.0489709
5×10^3	0.0159199	0.0007879	0.00726509	0.002069	0.000204086	2.52755
10^4	0.026196	0.0020630	0.0235729	0.005370	0.000522852	16.0086
5×10^4	0.098814	0.0144801	0.106045	0.027053	0.00375605	
10^5	0.180091	0.0756569	0.19264	0.054170	0.00687695	
5×10^5	1.10963	3.33762	0.943877	0.234129	0.0219009	
10^6	2.25833	9.01339	2.33021	0.450053	0.041976	



Fast Symmetric Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)	Direct Cholesky(s)
10^3	0.0066328	0.000208855	0.000833988	0.000344999	2.81334e-05	0.000160933	0.0281229
5×10^3	0.0103149	0.000798941	0.00359011	0.00228715	0.000156879	0.00105405	0.231569
10^4	0.02724	0.00200987	0.0175741	0.00552893	0.000396013	0.00261402	1.05882
5×10^4	0.08972	0.0151231	0.044107	0.034517	0.00314713	0.0162551	N/A
10^5	0.192696	0.067266	0.0933969	0.0709021	0.0061872	0.0332701	N/A
5×10^5	1.09055	3.2381	0.612783	0.263855	0.024405	0.151778	N/A
10^6	2.19711	8.79683	1.47177	0.545244	0.0434139	0.310443	N/A



2.3.3 RPY Tensor

The RPY Tensor is given by

$$K(i,j) = \begin{cases} \frac{k_BT}{6\pi\eta a}\left[\left(1-\frac{9}{32}\frac{r}{a}\right)\mathbf{I}+\frac{3}{32a}\frac{\mathbf{r}\otimes\mathbf{r}}{r}\right], & \text{if } r < 2a \\ \frac{k_BT}{8\pi\eta r}\left[\mathbf{I}+\frac{\mathbf{r}\otimes\mathbf{r}}{r^2}+\frac{2a^2}{3r^2}\left(\mathbf{I}-3\frac{\mathbf{r}\otimes\mathbf{r}}{r^2}\right)\right], & \text{if } r \geq 2a \end{cases}$$

where $r = ||\mathbf{r}_i - \mathbf{r}_j||$ with $\mathbf{r}$ being set as a sorted random matrix $\in (-1, 1)$ with the number of columns set equal to the dimension considered. For these benchmarks, we take $k_B = T = \eta = 1$. For a, we find the minimum of the interaction distances between all particles r_{min} and set $a = \frac{r_{min}}{2}$. This means that for the considered case, the RPY tensor simplifies to:

$$K(i,j) = \begin{cases} \frac{k_BT}{6\pi\eta a}\mathbf{I}, & \text{if } i = j \\ \frac{k_BT}{8\pi\eta r}\left[\mathbf{I}+\frac{\mathbf{r}\otimes\mathbf{r}}{r^2}+\frac{2a^2}{3r^2}\left(\mathbf{I}-3\frac{\mathbf{r}\otimes\mathbf{r}}{r^2}\right)\right], & \text{if } i \neq j \end{cases}$$

We have used `plotTree` to reveal the rank structure for the problems below when considering matrix size $N = 10000$, leaf size $M = 500$ and tolerance 10^{-12}.

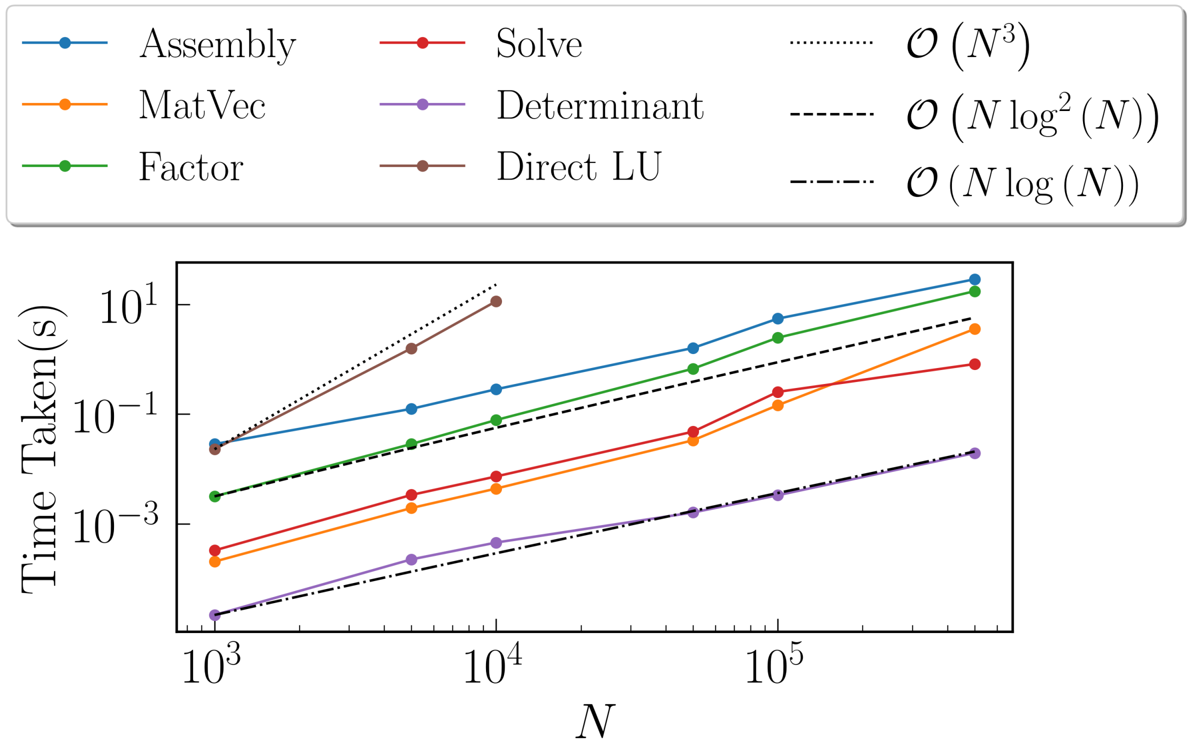
dim $= 1$

Time Taken vs Size of Matrix

For these benchmarks, the leaf size was fixed at $M = 100$, with tolerance set to 10^{-12}

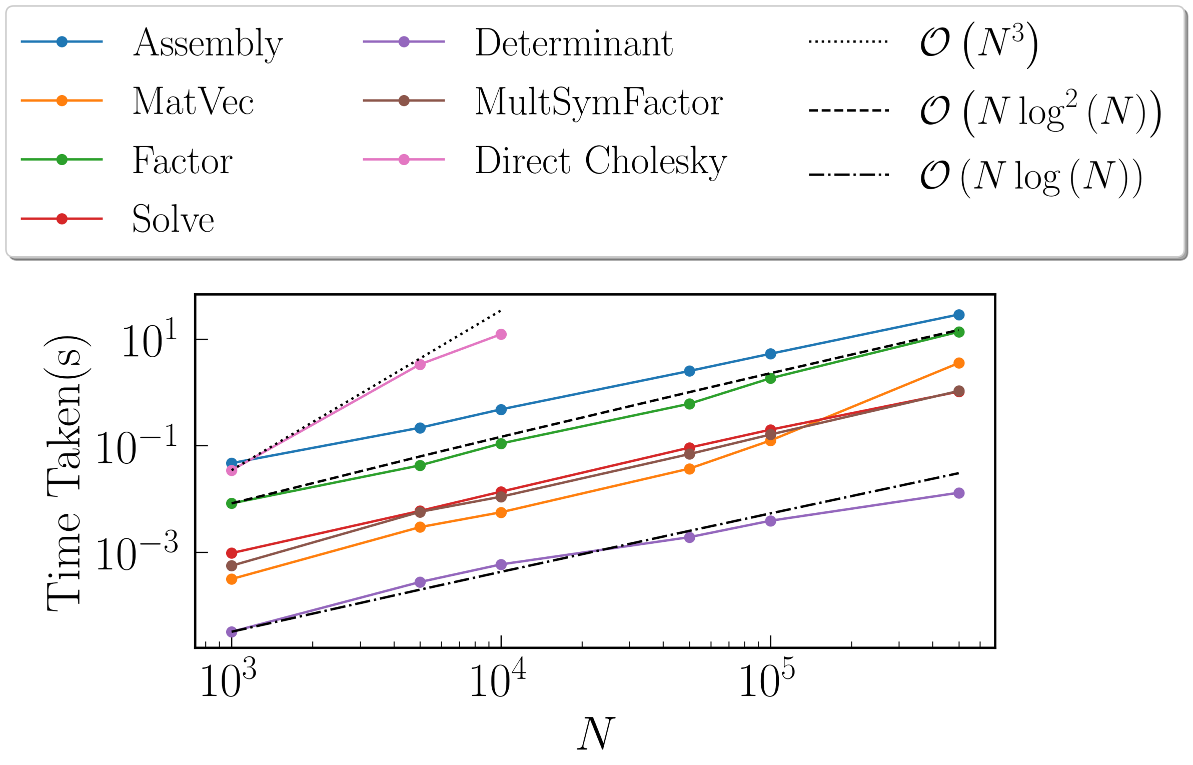
Fast Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	Direct LU(s)
10^3	0.0284998	0.0002059	0.00317788	0.000329	2.19345e-05	0.022975
5×10^3	0.124997	0.0019462	0.028585	0.003378	0.000226021	1.57937
10^4	0.284125	0.0044059	0.0781479	0.007328	0.000458002	11.3985
5×10^4	1.60538	0.033412	0.67361	0.047978	0.001616	
10^5	5.49457	0.145549	2.47014	0.254623	0.00333095	
5×10^5	28.6773	3.55899	17.4057	0.818555	0.0195651	



Fast Symmetric Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)	Direct Cholesky(s)
10^3	0.0468209	0.00031209	0.008219	0.00095796	3.19481e-05	0.0005548	0.034517
5×10^3	0.216226	0.00294399	0.042495	0.00592899	0.000274181	0.0056932	3.34734
10^4	0.47921	0.00559902	0.10963	0.0136352	0.00058794	0.0109739	12.3261
5×10^4	2.51609	0.0369091	0.609879	0.091403	0.00190592	0.069257	N/A
10^5	5.30011	0.124498	1.83744	0.198894	0.00388098	0.161215	N/A
5×10^5	28.9266	3.54814	13.6953	1.03255	0.0130229	1.06126	N/A



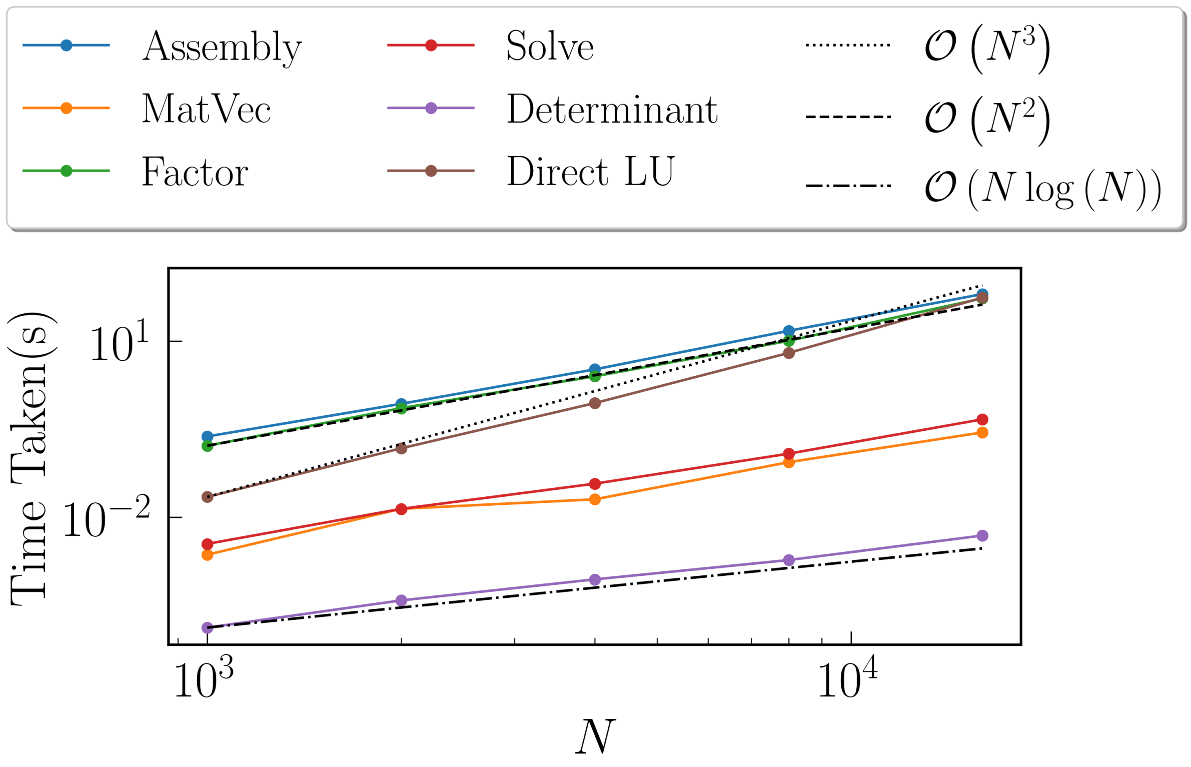
dim $= 2$

Time Taken vs Size of Matrix

For these benchmarks, the leaf size was fixed at $M = 100$, with tolerance set to 10^{-12}

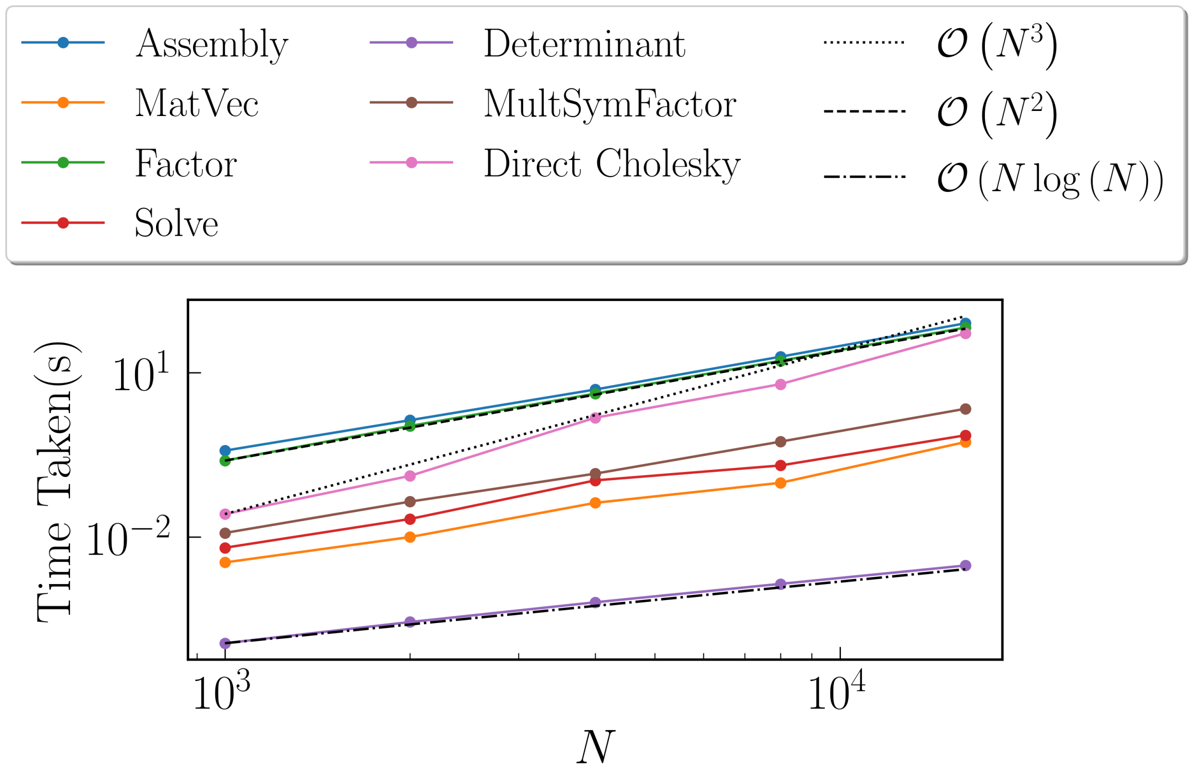
Fast Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	Direct LU(s)
10^3	0.237684	0.0022819	0.161561	0.003957	0.000130177	0.0220509
2×10^3	0.854779	0.0138352	0.728164	0.013024	0.000378847	0.148836
4×10^3	3.31121	0.0200999	2.52401	0.037282	0.000866175	0.88069
8×10^3	15.0432	0.0863769	10.1511	0.120667	0.00184798	6.3137
1.6×10^4	63.8282	0.278127	53.9138	0.46551	0.0048461	55.4166



Fast Symmetric Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)	Direct Cholesky(s)
10^3	0.375776	0.00342607	0.24704	0.00635099	0.000114918	0.0118291	0.026149
2×10^3	1.35015	0.00995803	1.05952	0.0212729	0.000280142	0.0441241	0.130154
4×10^3	4.89776	0.0418921	4.12168	0.107635	0.000642061	0.142907	1.49561
8×10^3	19.4326	0.0971079	16.3962	0.201411	0.00139117	0.547673	6.13806
1.6×10^4	79.2166	0.539779	66.3061	0.716309	0.00301003	2.18507	52.0411

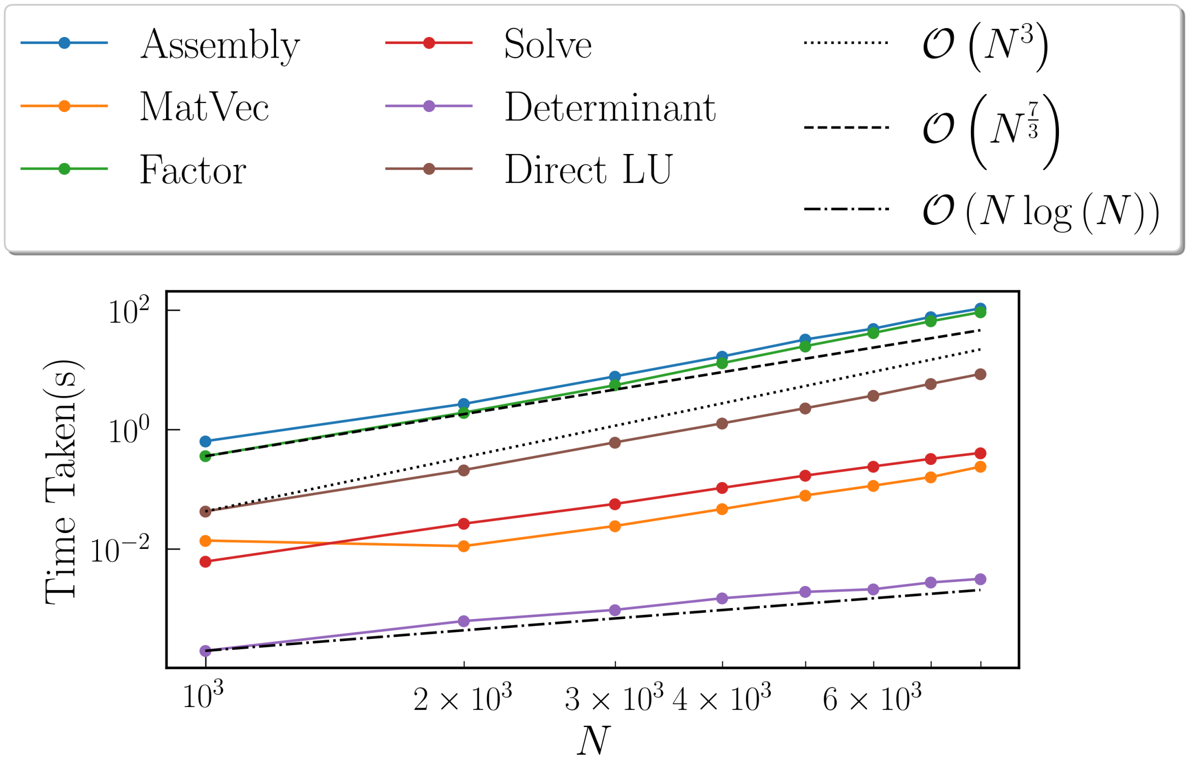


dim $= 3$

For these benchmarks, the leaf size was fixed at $M = 100$, with tolerance set to 10^{-12}

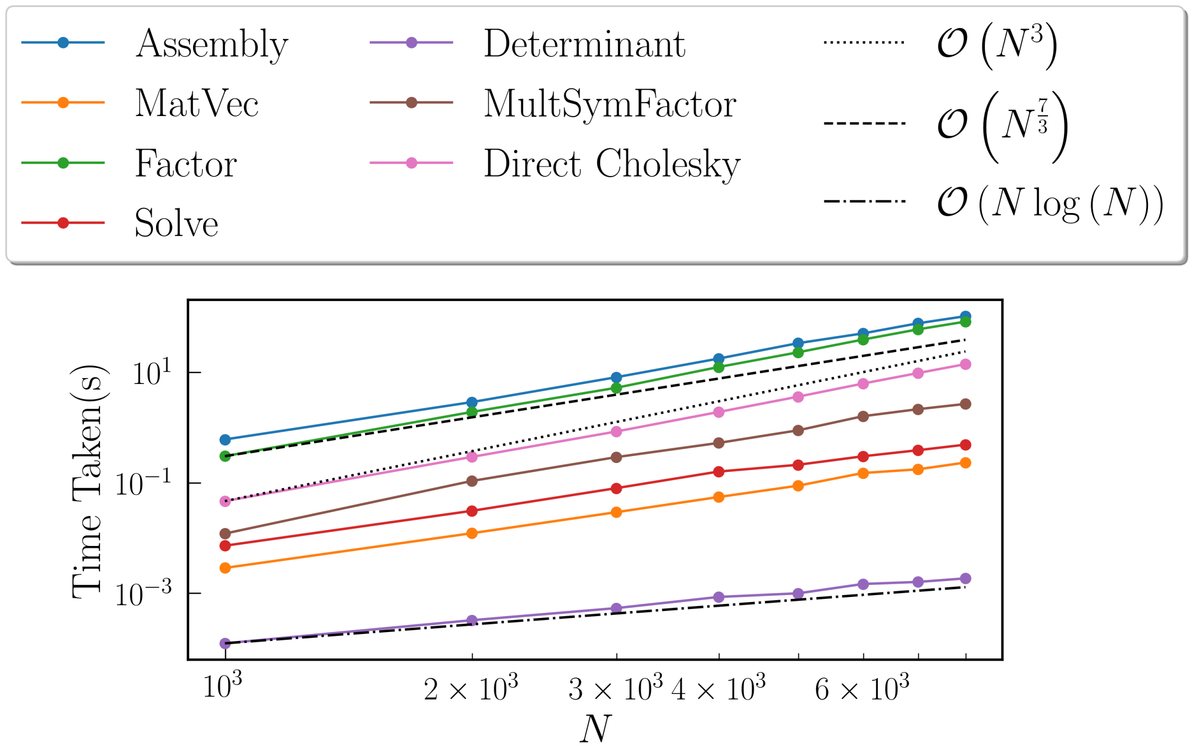
Fast Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	Direct LU(s)
999	0.637549	0.0138719	0.357686	0.006175	0.000198841	0.0427179
2×999	2.67525	0.0112109	1.915	0.026588	0.000624895	0.209663
3×999	7.72688	0.0243111	5.49087	0.056867	0.000961065	0.608226
4×999	16.6299	0.0466208	12.8754	0.105617	0.00150394	1.26595
5×999	31.9858	0.078845	24.7647	0.169478	0.00192094	2.26981
6×999	48.5977	0.114865	41.1307	0.241243	0.00212693	3.69254
7×999	76.1404	0.159873	64.5563	0.32299	0.00276279	5.80262
8×999	105.803	0.238746	91.5038	0.405407	0.00317407	8.46848



Fast Symmetric Factorization

N	Assembly(s)	MatVec(s)	Factorize(s)	Solve(s)	Determinant(s)	MultSymFactor(s)	Direct Cholesky(s)
999	0.60924	0.00287509	0.303544	0.00726318	0.000123978	0.012032	0.046663
2 × 999	2.88964	0.0122139	1.91556	0.0310731	0.000324965	0.107921	0.294039
3 × 999	8.15757	0.029355	5.2501	0.079915	0.000537872	0.291421	0.848032
4 × 999	17.7442	0.055275	12.4843	0.159855	0.000857115	0.527207	1.91446
5 × 999	33.741	0.08866	23.0541	0.211891	0.000997066	0.891842	3.62749
6 × 999	50.7127	0.150594	39.2222	0.302245	0.00147223	1.60647	6.26212
7 × 999	77.2103	0.175368	60.007	0.388988	0.00160313	2.15237	9.70271
8 × 999	103.913	0.232704	82.881	0.490437	0.00186086	2.67931	14.0804



Assembly
MatVec
Factor
Solve
Determinant
MultSymFactor
Direct Cholesky
$\mathcal{O}\left(N^3\right)$
$\mathcal{O}\left(N^{\frac{7}{3}}\right)$
$\mathcal{O}\left(N\log\left(N\right)\right)$
Time Taken(s)
10^1
10^{-1}
10^{-3}
10^3
2×10^3
3×10^3
4×10^3
6×10^3
N

CHAPTER 3

Other Links

Learn more about `HODLRlib` by visiting the

- Code Repository: http://github.com/sivaramambikasaran/HODLR
- Documentation: http://hodlrlib.rtfd.io