Refactoring and Optimization of the Python AM Solver using PETSc

Presenter: Chao Li
PI: Dr. Gianluca Lazzi
08/04/2022

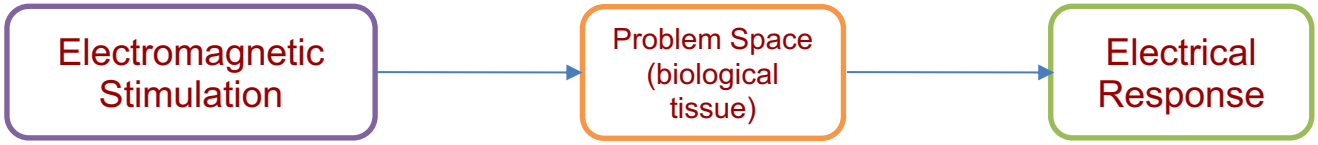
Introduction

- **Python AM (Admittance Method) Solver**: simulate the electrical stimulation of the biological tissue, and how the electric field propagates through the network of neurons.

- **Limitation**: The processing time of the AM solver ranges from hours to days given a medium to large input size

- **To address the limitation**: PETSc (pronounced, PET-see) library is more optimized for sparse matrix solving than the Scipy library

Literature Review



Real World

Computer Simulation

- (Cela, 2010)
- (Stang et al., 2019)

Research Question(s)

1. How to use PETSc4py?

1.1 Installation, environment setup

1.2 understanding PETSc4py APIs through PETSc Documentation

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

3. If so, how to apply PETSc4py in the Python AM Solver?

3.1 Installation, environment setup for PETSc4py on CARC cluster

1. How to use PETSc4py?

1.1 Installation, environment setup

- To ensure the reproducibility of my work, I started the documentation for the next version of the Python AM Solver
- For a successful setup: we need correct python version, package dependency, and environmental variables

Documentation on Refactoring Python AM Solver with PETSc4py

1. Package Installation and Local Environment Setup

*Note: the following setup steps are best suited to run on a **local environment** (your home OS terminal) instead of the environment in **PyCharm** or **Jupyter Notebook**.

1. Type in "**pip list**" in the terminal to check if (1) numpy, (2) petsc, (3) petsc4py packages are present. (4) matplotlib is also a useful package to have for plotting graphs.
2. If not present, do **pip3 install petsc** to install the petsc Package
3. Then do **pip3 install PETSc4py**, make sure you do this after you have petsc installed.
4. By the time you can see (1) numpy, (2) petsc, (3) petsc4py, and (4) matplotlib using the "pip list" command. Check your python version using "**python --version**", if the version is lower than python 3.0.0. You should install a newer version than python 3.0.0. You can check if you have a more recent version of python by typing "**python3 --version**"
5. Next, go into the folder that contains the test file and type for example "**python3 test.py**" to run test.py

Figure 1. Screenshot from the Google Doc titled "Documentation on Refactoring Python AM Solver with PETSc4py"

1. How to use PETSc4py?

1.2 understanding PETSc4py APIs through PETSc Documentation

- PETSc4py is poorly documented, we needed a sustainable workflow to learn the PETSc4py APIs

```
setPythonContext(self, context)
setPythonType(self, py_type)
setResidualNorm(self, rnorm)
setTolerances(self, rtol=None, atol=None, divtol=None, max_it=None)
setType(self, ksp_type)
setUp(self)
setUpOnBlocks(self)
```

Figure 2. Example of documentation of PETSc4py

- With both websites, one can learn most of the necessary PETSc4py functions

KSPSetTolerances

Sets the relative, absolute, divergence, and maximum iteration tolerances used by the default KSP convergence testers.

Synopsis

```
#include "petscksp.h"
#include "petscmat.h"
PetscErrorCode  KSPSetTolerances(KSP ksp,PetscReal rtol,PetscReal abstol,PetscR
```

Logically Collective on ksp

Input Parameters

- ***ksp*** **-** the Krylov subspace context
- ***rtol*** **-** the relative convergence tolerance, relative decrease in the (possibly preconditioned) residual norm
- ***abstol*** **-** the absolute convergence tolerance absolute size of the (possibly preconditioned) residual norm
- ***dtol*** **-** the divergence tolerance, amount (possibly preconditioned) residual norm can increase before KSPConvergedDefault() concludes that the method is diverging
- ***maxits*** **-** maximum number of iterations to use

Figure 3. Example of documentation of PETSc

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

• **Sample Problem (Implicit 1D Heat Diffusion):**	• **Problem to be solved by AM Solver:**
1. Solving ***Ax = b***	1. Solving ***Ax = b***
2. ***A*** is a sparse matrix	2. ***A*** is a sparse matrix
3. The construction of ***A*** only requires one for loop	3. The construction of ***A*** is complicated

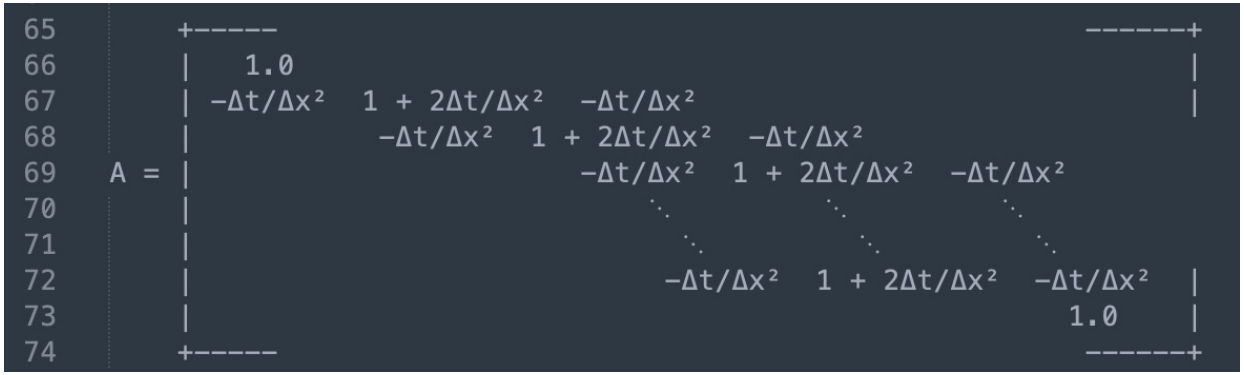


Figure 4. Illustration of the Matrix ***A*** in the Implicit 1D Heat Diffusion Problem

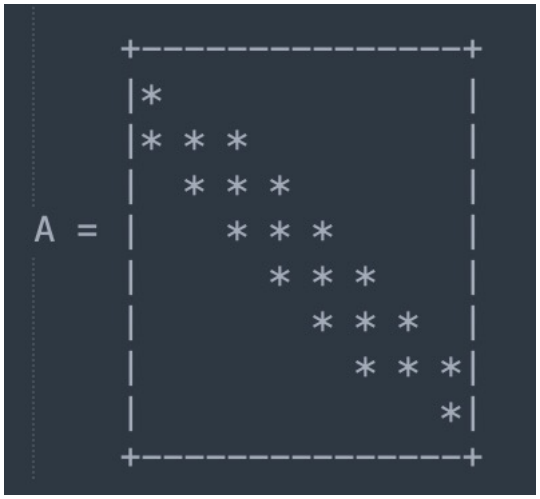
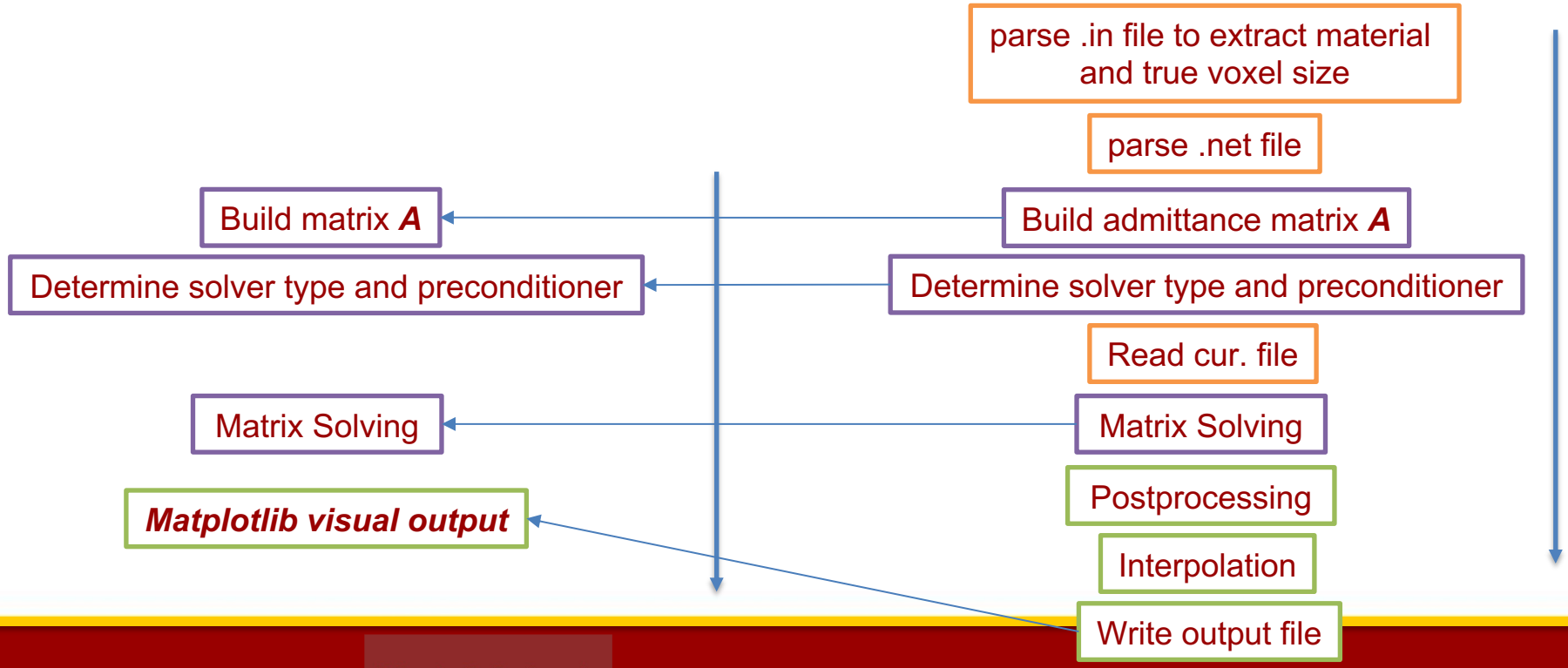


Figure 5. sparsity structure of Matrix ***A***

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

• **Sample Problem (Implicit 1D Heat Diffusion):**	• **Problem to be solved by AM Solver:**
1. Solving ***Ax = b***	1. Solving ***Ax = b***
2. ***A*** is a sparse matrix	2. ***A*** is a sparse matrix
3. The construction of ***A*** only requires one for loop	3. The construction of ***A*** is complicated



2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

PETSc4py Implementation

```
def main():
    element_length = 1.0 / (N_POINTS - 1)
    mesh = np.linspace(0.0, 1.0, N_POINTS)

    diagonal_entry = 1.0 + 2.0 * TIME_STEP_LENGTH / element_length**2
    off_diagonal_entry = - 1.0 * TIME_STEP_LENGTH / element_length**2
```

```
    # Create a new sparse PETSc matrix, fill it and then assemble it
    A = PETSc.Mat().createAIJ([N_POINTS, N_POINTS])
    A.setUp()

    A.setValue(0, 0, 1.0)
    A.setValue(N_POINTS-1, N_POINTS-1, 1.0)

    for i in range(1, N_POINTS - 1):
        A.setValue(i, i, diagonal_entry)
        A.setValue(i, i-1, off_diagonal_entry)
        A.setValue(i, i+1, off_diagonal_entry)

    A.assemble()
```

```
b = PETSc.Vec().createSeq(N_POINTS)
b.setArray(initial_condition); b.setValue(0, 0.0);
b.setValue(N_POINTS-1, 0.0)

# Allocate a PETSc vector storing the solution to the linear system
x = PETSc.Vec().createSeq(N_POINTS)
```

```
# Instantiate a linear solver: Krylow subspace linear iterative solver
ksp = PETSc.KSP().create(); ksp.setOperators(A); ksp.setFromOptions()

ksp.setType('cg'); ksp.getPC().setType('ilu') # solver & preconditioner

# plt.plot(mesh, initial_condition)
for iter in range(N_TIME_STEPS):
    ksp.solve(b, x)

    # Re-assemble the rhs to move forward in time
    current_solution = x.getArray()
    b.setArray(current_solution)
    b.setValue(0, 0.0)
    b.setValue(N_POINTS - 1, 0.0)
```

Creation of Matrix *A*

Creation of Matrix *b* And *X*

Actual sparse matrix solving

Scipy Implementation

```
def main():
    element_length = 1.0 / (N_POINTS - 1)
    mesh = np.linspace(0.0, 1.0, N_POINTS)

    diagonal_entry = 1.0 + 2.0 * TIME_STEP_LENGTH / element_length**2
    off_diagonal_entry = - 1.0 * TIME_STEP_LENGTH / element_length**2
```

```
    ones = off_diagonal_entry * np.ones(N_POINTS - 1)
    L = diagonal_entry * np.ones(N_POINTS)
    L[0] = 1.0
    L[-1] = 1.0
    A = sparse.diags([ones, L, ones], [-1, 0, 1], format = 'csc')
```

```
# Assemble the initial rhs to the linear system
b = initial_condition
b[0] = 0.0; b[-1] = 0.0

x = np.zeros(N_POINTS)
```

```
# preconditioner
iLU = sparse.linalg.spilu(A)
M = sparse.linalg.LinearOperator((N_POINTS, N_POINTS), iLU.solve)



for iter in range(N_TIME_STEPS):
    (x, info) = spla.cg(A, b, maxiter=None, M=M)
    # Re-assemble the rhs to move forward in time
    b = x
    b[0] = 0.0; b[-1] = 0.0
```

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

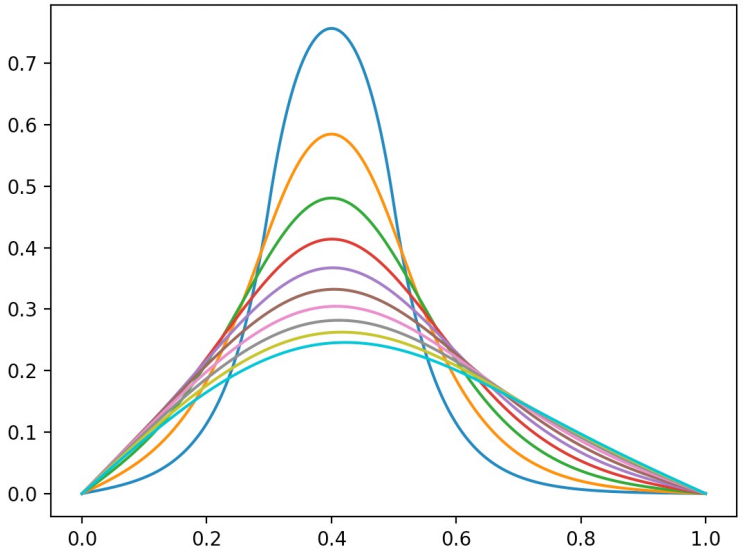


Figure 6. Solving results of **PETSc4py** implementation on the Implicit 1D Heat Diffusion Problem

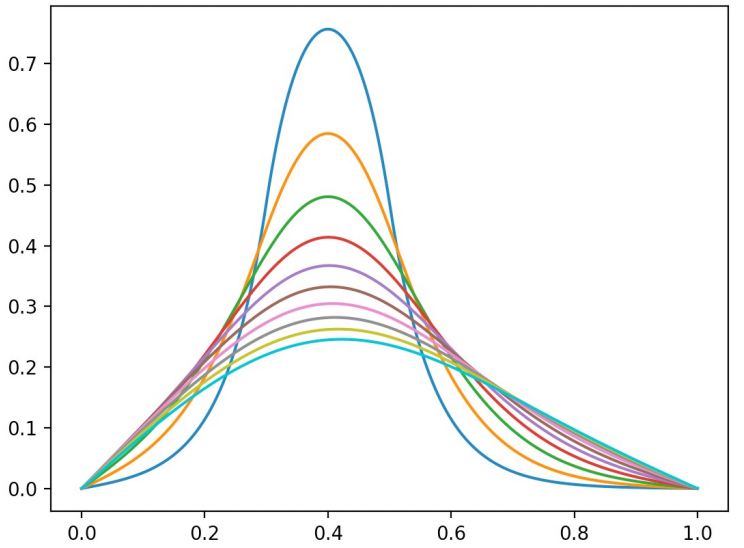


Figure 7. Solving results of **Scipy** implementation on the Implicit 1D Heat Diffusion Problem

The resulting ***x*** vector in ***Ax = b*** was examined and compared to make sure both gives the same and correct answer

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

- **Variable**: Matrix Dimension

- **Controlled** parameters:
- - **Solver Method**: Conjugate Gradient
- - **Preconditioner**: incomplete LU factorization (ILU)

- As the matrix dimension increases, Scipy consistently takes around x1.5 the time to finish running.

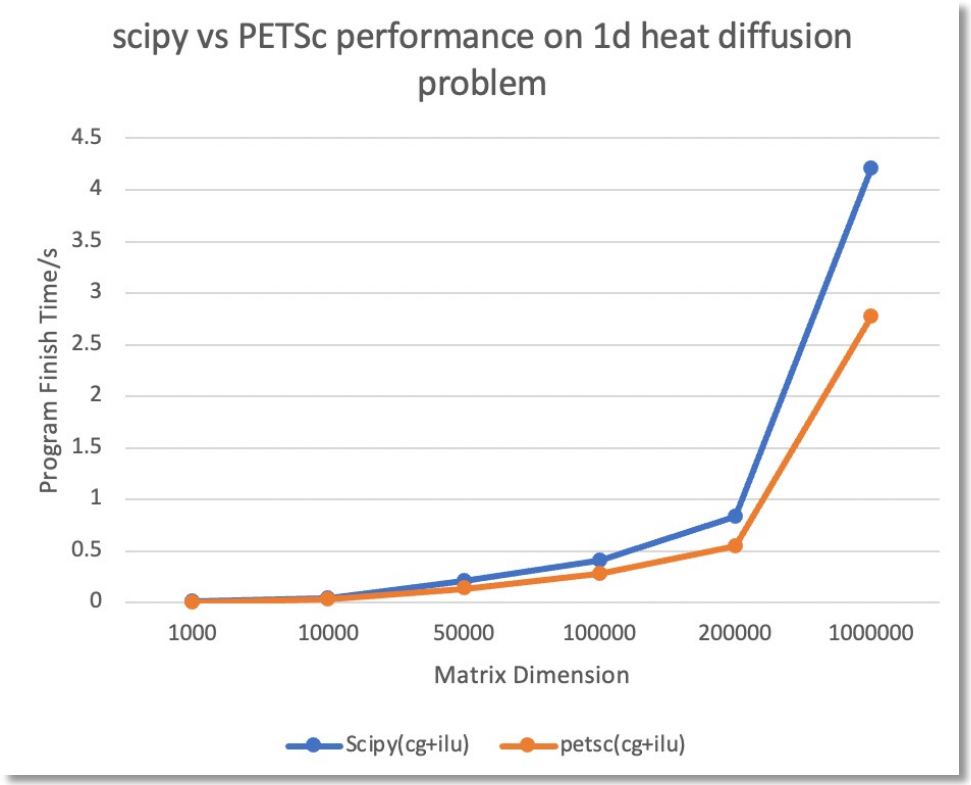


Figure 8. scipy vs PETSc performance on 1d heat diffusion problem

2. Does PETSc4py perform better in sparse matrix solving?

2.1 scipy vs PETSc4py benchmarking on small sample problems

CAVEAT

- Use a solver utilizes one of the **Krylov subspace methods** (e.g. generalized minimum residual, conjugate gradient)
- Don't forget **preconditioner**: removing the ILU preconditioner in Scipy, it becomes x45 slower at dimension 1000, and x475 slower at dimension 10000 comparing to itself with a preconditioner.
- Increasing CPU cores, did not improve performance since the program is single-threaded for now

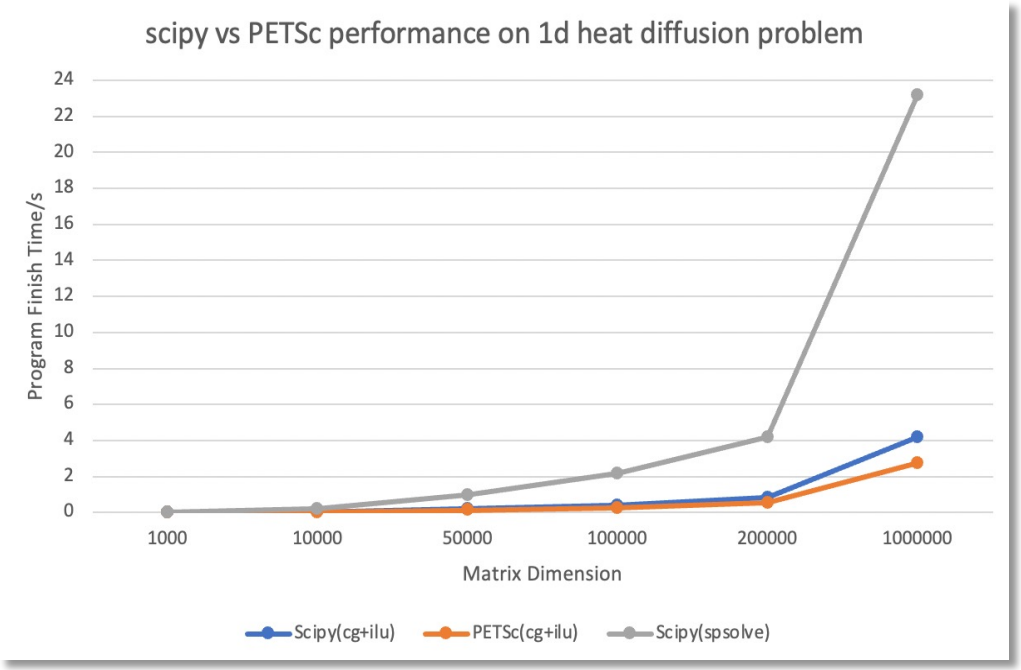


Figure 9. scipy vs PETSc vs scipy(spsolve) performance

3. If so, how to apply PETSc4py in the Python AM Solver?

3.1 Installation, environment setup for PETSc4py on CARC cluster

- Currently the CARC system, on which the Python AM Solver is used, does not have a PETSc4py as its module for quick installation.
- We need to download it, setup the environment, and compile it by ourselves.
- Step-up-step guide is provided on the documentation.

2. PETSc Installation Steps on CARC system

Currently, We don't have PETSc and PETSc4py installed and available as modules. I would recommend following these steps and installing them in your home directory:

```
cd $HOME
mkdir src
cd src
wget https://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-3.17.3.tar.gz
--no-check-certificate
tar -xvf petsc-3.17.3.tar.gz

cd petsc-3.17.3

module purge
module load gcc/11.3.0
module load openmpi/4.1.4
module load cmake/3.23.2
module load anaconda3/2021.05

export MPICC=$(which mpicc)
```

NOTE: Do one of the following step, if you are installing for the first time and only want one version of petsc4py, not both

Figure 1. Screenshot from the Google Doc titled "Documentation on Refactoring Python AM Solver with PETSc4py"

Discussion/ Conclusion

- PETSc is indeed a more optimized library, with a performance x1.5 faster to its Scipy counterpart. We expect to see a similar improvement in Python AM solver.

- Currently, the refactoring of the Python AM Solver is finished. Preliminary testings on larger input files results in a x1.3 ~ 1.6 speed up.

- Testing ran on the task named, “cyl_large_res”, finishes in 1 hour and 42 minutes comparing to 2 hours and 33 minutes on the previous stable version of the AM solver.

Future Research

- More testing on even larger files (ones that require days to run)
- Testing on various combinations of solver types and preconditioners
- Debugging of the PETSc implementation
- Parallelization / Multithreading of the PETSc code

References

1. Balay, S., Gropp, W., McInnes, L. C., & Smith, B. F. (1998). PETSc, the portable, extensible toolkit for scientific computation. *Argonne National Laboratory*, 2(17).

2. Cela, C. J. (2010). *A Multiresolution Admittance Method for Large-Scale Bioelectromagnetic Interactions* [Doctoral dissertation, North Carolina State University]. North Carolina State University ProQuest Dissertations Publishing. https://www.proquest.com/openview/b1889475b461ae38c3ca581554a2ee54/1?pq-origsite=gscholar&cbl=18750

3. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2

4. Stang, J., Lazzi, G., Chen, G., Kosta, P., Paknahad, J., Machnoor, M., Iseri, E., Du, J., Brizi, D., & Loizos, K. (2019). Recent Advances in Computational and Experimental Bioelectromagnetics for Neuroprosthetics. *2019 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, 1382. https://doi.org/10.1109/iceaa.2019.8878960

5. Virtanen P., Gommers R., Oliphant T. E., Haberland M., Reddy T., Cournapeau D., Burovski E., …, & SciPy 1.0 Contributors. (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17(3), 261-272.

Acknowledgements

- ***Dr. Lazzi and Dr. Ragusa**, for this summer research opportunity.*
- ***Dr. Bouteiller**, for my onboarding process and answering many of my questions when I needed directions.*
- ***Jinze Du**, the brilliant PhD student, my friend, whom I have been assisting this summer, thank you for all the resources you pointed me to and the ideas you've inspired me.*
- ***Iman Rahbari,** the CARC researcher support, who is critical in helping me writing out the installation steps of PETSc4py installation.*
- ***Jack Damiani and Simon To**, thank you for introducing me to the CARC system and the details of the current version of the Python AM Solver.*
- *Thank you to all the **CNL research experience organizers** and **members of the Dr. Lazzi's Lab**. I had a wonderful experience this summer.*