Leopold-Franzens-Universität

Master Thesis

Matrix-free Leja based exponential integrators in Python

Maximilian Samsinger

October 11, 2019

Supervised by Lukas Einkemmer and
Alexander Ostermann

Leopold-Franzens-Universität Innsbruck

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Stellen, die wörtlich oder inhaltlich den angegebenen Quellen entnommen wurden, sind als solche kenntlich gemacht.

Ich erkläre mich mit der Archivierung der vorliegenden Bachelorarbeit einverstanden.

Datum	Unterschrift

Matrix-free Leja based exponential integrators in Python

Abstract

1 Introduction

Consider the action of the matrix exponential function

$$e^A v, \quad A \in \mathbb{C}^{N\times N}, v \in \mathbb{C}^N.$$

It can be difficult or impossible to compute e^A in a first step and then the action $e^A v$ in a seperate step. This is especially true in applications where $N > 10000$ is not uncommon. Furthermore the matrix exponential of a sparse matrix is in general no longer sparse. Therefore it is more feasable to compute the action of the matrix exponential in a single step. This can be done by approximating the matrix exponential with a matrix polynomial p_n of degree n in A

$$e^A v \approx p_n(A)v.$$

This approach has many advantages. The cost of the computation of $p_n(A)v$ mainly depends on the calculation of n matrix-vector multiplications with A. Furthermore the explicit knowledge of A itself is no longer required. A can be replaced by a linear function, which can be more convenient and saves memory.

2 The Leja method

In this section we explore the core concepts of the Leja method for the exponential function. This serves as an introduction for the

Replacing the matrix exponential with a polynomial:

Exploiting the properties of the matrix exponential functions:

Evaluate the polynomial using precomputed Leja interpolation nodes:

3 Linear advection diffusion equation

Plot the eigenvalues of the matrix

4 Numerical experiments

For the first experiments we will discretize multiple one-dimensional advection-diffusion-reaction equations with hybrid difference schemes.[1] We will always choose an equidistant grid with grid size $h = \frac{1}{N}$, $N \in \mathbb{N}$ and grid points $x_i = ih$ for $i = 0 \ldots, N$ on the domain $\Omega = [0, 1]$. The resulting ordinary differential equations (ODEs) will be solved with four different integrators. Our goal is to investigate the respective computational costs of these methods while achieving a prescribed relative tolerance `tol`.

Crank-Nicolson method: We refer to the Crank-Nicolson method of order 2 as `cn2`. In our implementation of `cn2`, we used the SciPy[7] package `scipy.sparse.linalg.gmres` to solve linear equations. We set the relative tolerance to `tol`$/s$, where s is the total number of substeps taken for solving the ODE. This choice guarantees that the sum of errors made by `gmres` is always lower than our specified tolerance `tol`, since we have to solve exactly one linear equation per substep. No preconditioner was used for `gmres`. The Crank-Nicolson method is unconditionally stable and therefore does not have to satisfy the Courant-Friedrichs-Lewy (CFL) conditions imposed by the advective and diffusive part of the differential equations.

Exponential Rosenbrock-Euler method: We refer to the Exponential Rosenbrock-Euler method of order 2 as `exprb2`. The approximate the action of the matrix exponential with the Leja method. No hump reduction is used. The maximal interpolation degree is set to 100. Note that the total number of matrix-vector multiplication per time step can still exeed 100 since we have to compute a single matrix norm. This typically happens for $s = 1$.

Explicit midpoint method: We refer to the explicit midpoint method of order 2 as `rk2`.

Classical Runge kutta: We refer to the classical Runge-Kutta method of order 4 as `rk4`.

For our experiments we will often fix one of two different Péclet numbers

$$\mathtt{Pe} = \frac{b}{a}, \quad \mathtt{pe} = \frac{hb}{2a},$$

The Péclet numbers are dimensionless quantities representing the ratio of the advective velocity b to the diffusive velocity a. While `Pe` characterizes the original partial differential equation, the grid Péclet number `pe` is the dimensionless quantity for the resulting ODE after discretization. Note that by fixing `pe` for varying grid sizes, we have to change the original partial differential equantion. Unless otherwise noted we accomplish that by replacing b with $2b$ and a with ah.

[1] Need a source, https://en.wikipedia.org/wiki/Hybrid_difference_scheme

4.1 Experiment 1: Linear advection diffusion equation

Consider the one-dimensional advection-diffusion equation

$$\partial_t u = a\partial_{xx}u + b\partial_x u \quad a, b \geq 0$$
$$u_0(t) = e^{-80\cdot(t-0,45)^2} \quad t \in [0, 0.1]$$

with homogeneous Dirichlet boundary conditions on the domain $\Omega = [0, 1]$. For a fixed $N \in \mathbb{N}$ we approximate the diffusive part with second-order central differences on an equidistant grid with grid size $h = \frac{1}{N}$ and grid points $x_i = ih$, $i = 0 \ldots, N$.

$$\partial_{xx}u(x_i) = \frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} + \mathcal{O}(h^2)$$

In order to limit numerical instabilities we discretize the advective part with forward differences, similar to the upwind scheme.[2]

$$\partial_x u(x_i) = \frac{u(x_{i+1}) - u(x_i)}{h} + \mathcal{O}(h)$$

The resulting system of ordinary differential equation is given by

$$\partial_t u = Au.$$

Some eigenvalues of A can have an extremely large negative real part. Therefore, since no explicit Runge-Kutta method is A-stable, this imposes very stingend conditions on the time step size τ for rk2 and rk4.[3] We will refer to the Courant-Friedrich-Lewy (CFL) conditions imposed by the advective and diffusive part of A respectively by C_{adv} and C_{dif}.

$$C_{adv} = \frac{b\tau}{h} \leq 1, \quad C_{dif} = \frac{a\tau}{h^2} \leq \frac{1}{2}$$

In our case the problem is fully linear and therefore `exprb2` simplifies to the computation of the action of the matrix exponential funcion with the Leja method. We write `expleja` for the single precision Leja method approximation. Note that reference solution was computed with double precision and therefore uses different nodes.

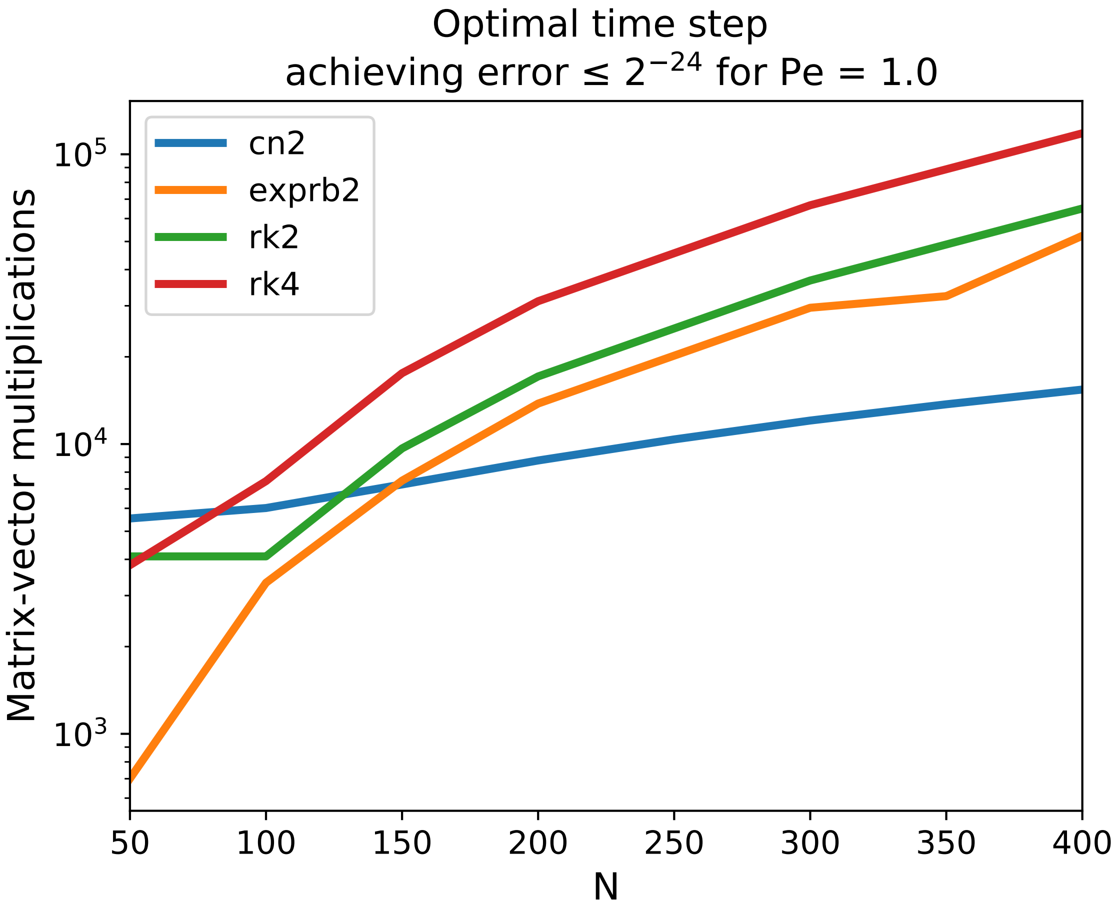
In order to keep the solution from vanishing, we only consider coefficients a, $b \in [0, 1]$. The advection-diffusion ratio scaled by the grid size h is represented by the grid Péclet number

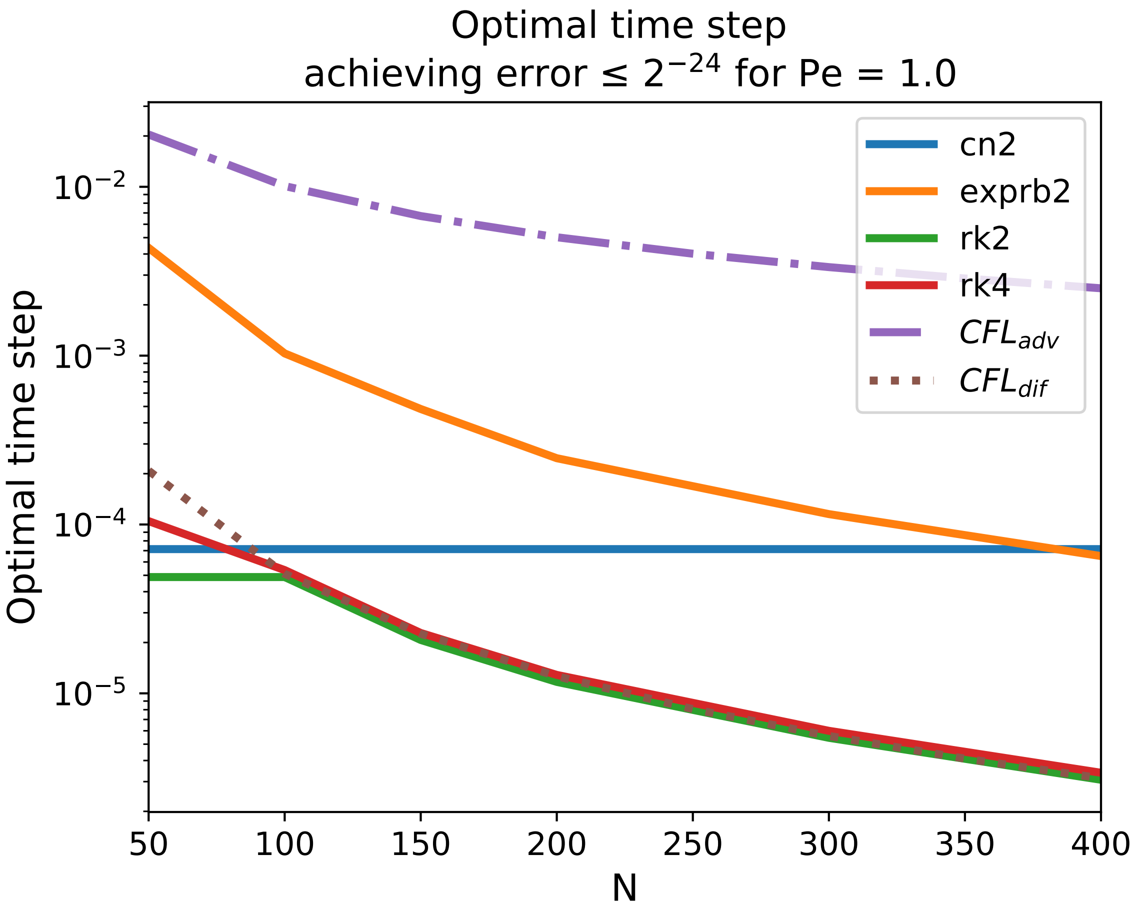
[2] Maybe create a seperate section on hybrid difference schemes? There we can also analyze the resulting matrix A itself and plot the eigenvalues. I need sources for that though.

[3] See section **??**

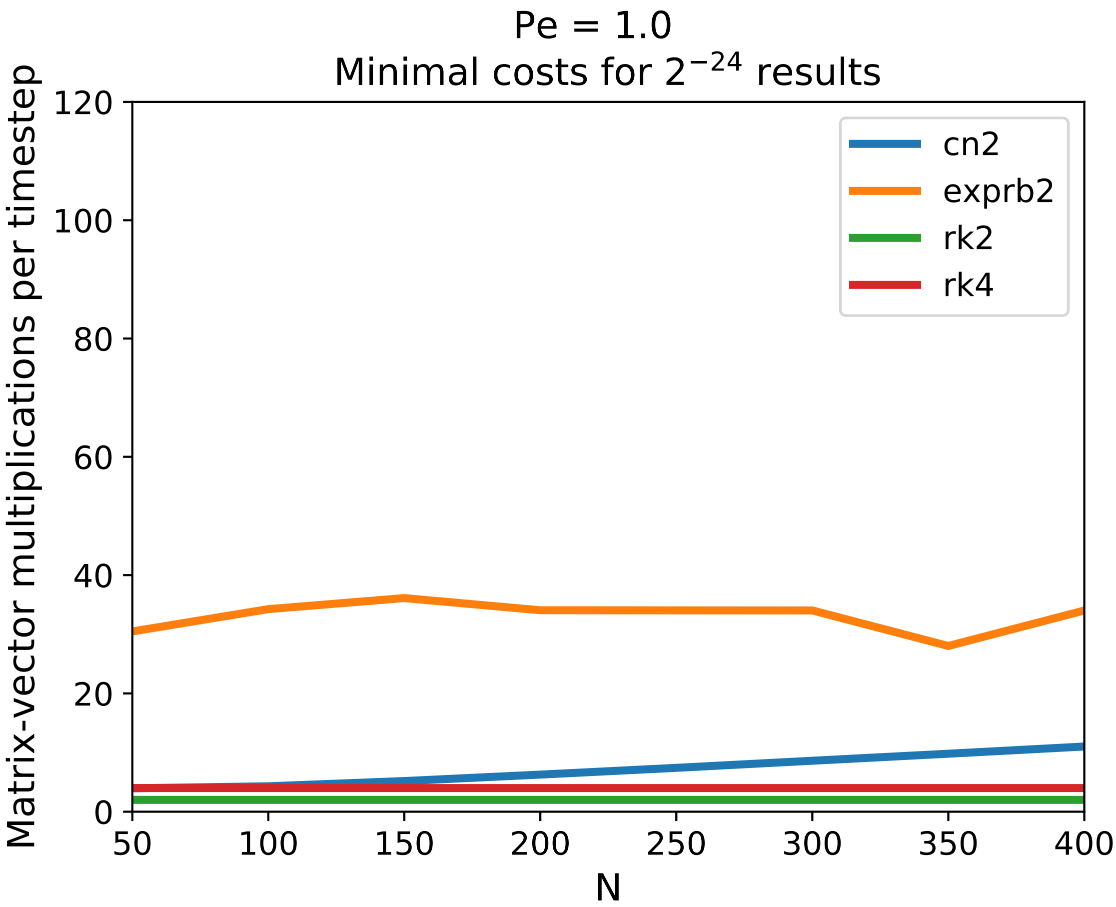
5 Appendix

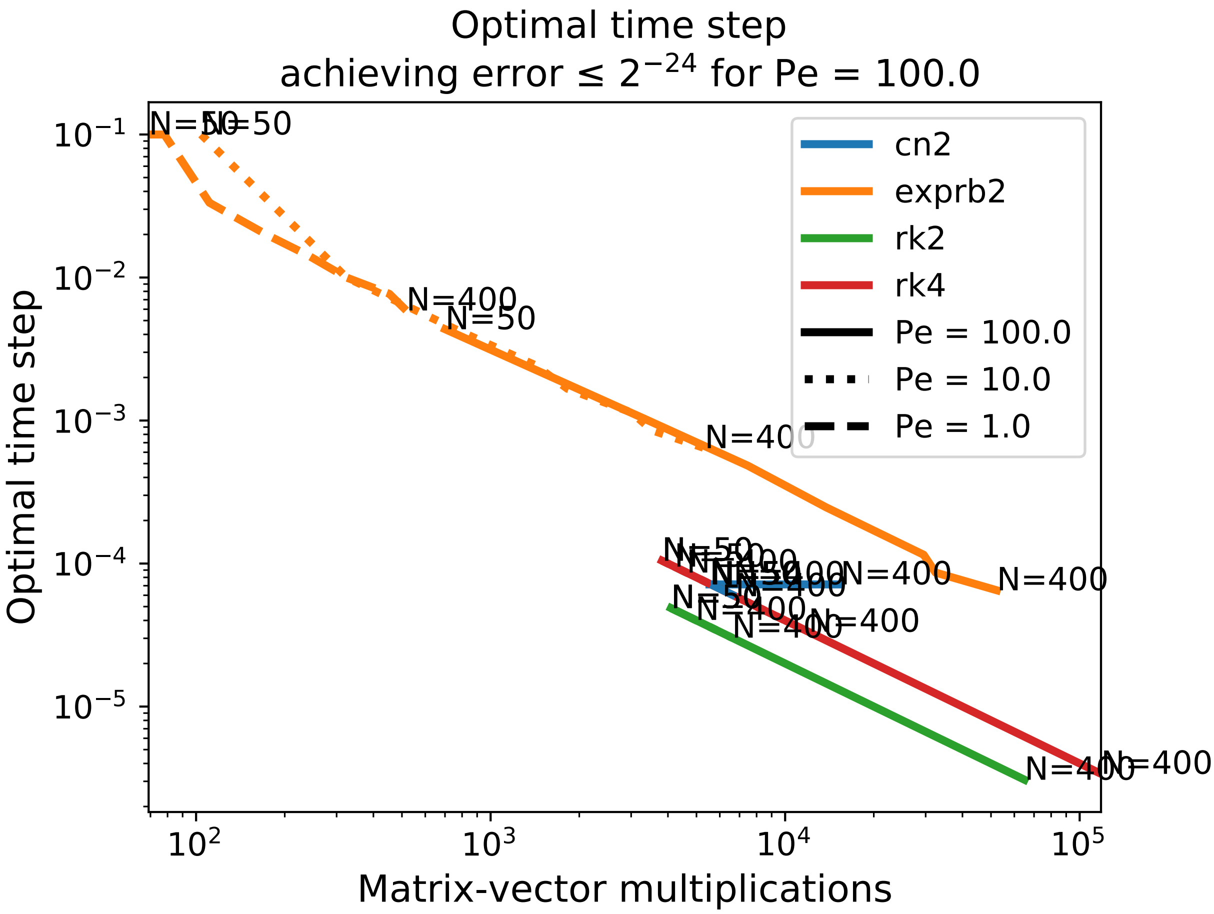
5.1 Experiment 1





Optimal time step
achieving error ≤ 2^−24 for Pe = 1.0
Optimal time step
N
10^−2
10^−3
10^−4
10^−5
50
100
150
200
250
300
350
400
cn2
exprb2
rk2
rk4
CFL_adv
CFL_dif


Pe = 1.0
Minimal costs for 2^{-24} results
Matrix-vector multiplications per timestep
0
20
40
60
80
100
120
50
100
150
200
250
300
350
400
N
cn2
exprb2
rk2
rk4



5.2 Experiment 1.5

In the matrix-free case the linear operator A is not explicitly given. In order to compute the matrix norm $\|A\|_2$ we use power iterations to estimate the absolutely largest eigenvalue of A. A priory it is not clear how many power iterations it are neccessary for a good approximation.

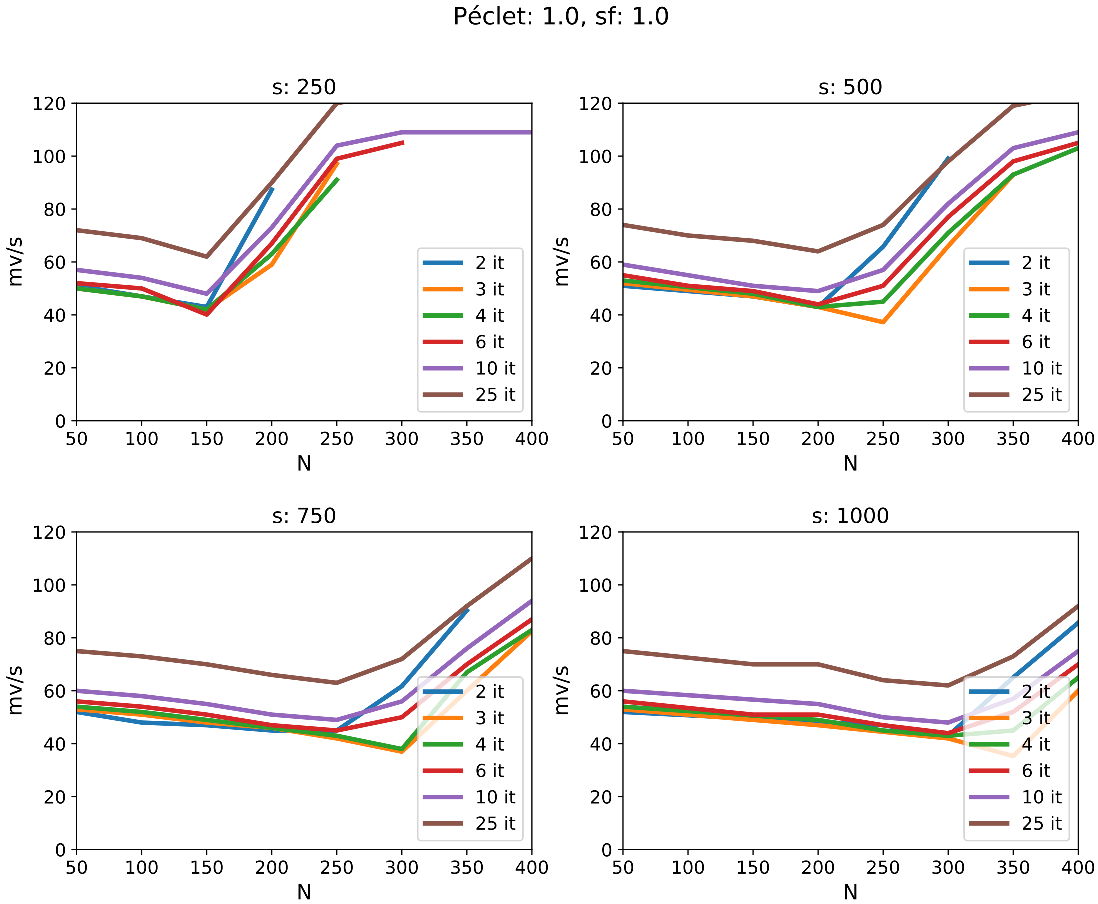


Figure 1: Space dimension N vs costs `mv` per timestep `s` for the exponential Rosenbrock method `exprb2`. Results are only shown if they achieve single precision.

References

[1] M. Caliari, A. Ostermann. Implementation of exponential Rosenbrock-type integrators, Applied Numerical Mathematics 59 (2009), 568-581.

[2] A. Al-Mohy, N. Higham. Computing the action of the matrix exponential, with an application to exponential integrators, SIAM Journal on Scientific Computing 33 (2011), 488-511.

[3] L. Reichel. Newton interpolation at Leja points, BIT Numerical Mathematics 30 (1990), 332-346.

[4] M. Caliari, M. Vianello, L. Bergamaschi. Interpolating discrete advection-diffusion propagators at Leja sequences, Journal of Computational and Applied Mathematics 172 (2004), 79-99.

[5] M. Caliari, P. Kandolf, A. Ostermann, S. Rainer. The Leja method revisited: backward error analysis for the matrix exponential, SIAM Journal on Scientific Computation, Accepted for publication (2016). arXiv:1506.08665.

[6] Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org. Manual at https://docs.python.org/2/. [Online; accessed 2015-12-14]

[7] E. Jones, E. Oliphant, P. Peterson, SciPy: Open Source Scientific Tools for Python, Available at http://www.scipy.org/. [Online; accessed 2015-12-14]