IMPLEMENTATION OF THE CHEBYSHEV RATIONAL APPROXIMATION METHOD FOR BURNUP CALCULATIONS INTO PYNE

MARISSA I. RAMIREZ ZWEIGER

Abstract. PyNE (Python for Nuclear Engineering) is an open-source software package containing tools for nuclear engineers. PyNE has the capacity to perform a variety of calculations of interest to nuclear engineers, including calculations of the consumption of fuel in a nuclear reactor, commonly referred to as burnup calculations. These calculations were previously performed by the ALARA method, but PyNE has now been updated to give users the option to perform the calculations by the Chebyshev Rational Approximation Method (CRAM), which can be both faster and more accurate than ALARA. This paper describes the implementation of CRAM in detail and provides numerical examples to illustrate performance.

Key words. Chebyshev Rational Approximation Method, Burnup Calculations, Python

1. Background.

1.1. The Burnup Equation. Nuclear energy is generated through fissioning of heavy nuclei, typically ^{235}U, in a nuclear reactor core. Fission and other processes such as neutron capture and radioactive decay cause the materials in the fuel to change over time. This process is referred to as "burnup." Burnup can be described mathematically by the following differential equation [1].

$$\frac{dN_i}{dt} = \sum_j \gamma_{ji}\sigma_{f,j}N_j\phi + \sigma_{c,i-1}N_{i-1}\phi + \sum_k \lambda_{ki}N_k - \sigma_{a,i}N_i\phi - \lambda_i N_i \tag{1.1}$$

Where

N_i : number density of nuclide i (atoms/cm^2)

γ_{ij} : yield of nuclide j from a fission of nuclide i

$\sigma_{f,j}$: microscopic fission cross section of nuclide j (cm^2)

ϕ: neutron flux (ns/cm$^2\cdot$ s)

$\sigma_{c,i-1}$: capture cross section of nuclide $i-1$ (cm^2)

λ_{ki}: decay constant of nuclide k to nuclide i

λ_i: decay constant of nuclide i

$\sigma_{a,i}$: absorption cross section of nuclide i (cm^2)

All variables except, N_i and ϕ are constant with respect to time. N_i and Φ are updated at each time step. This equation can be rewritten as

$$\frac{d\vec{N}}{dt} = \mathbf{A}\vec{N}, \tag{1.2}$$

where $\vec{N}$ is the vector of nuclide number densities and $\mathbf{A}$ is the burnup matrix. This gives the solution

$$\vec{N}(t) = e^{\mathbf{A}t}\vec{N}_0, \tag{1.3}$$

where $\vec{N}_0$ is the initial composition vector and the term $\exp(\mathbf{A}t)$ is known as the matrix exponential. The matrix exponential is formally defined by the convergent power series

$$e^{tA} = I + tA + \frac{t^2A^2}{2!} + ... \tag{1.4}$$

but no closed form solution is known [3].

Like many other differential equations, the burnup equation is difficult to solve analytically. Thus a number of numerical solvers for the Bateman Equation have been developed [1] [3] [4]. One approach is to use a time-step based ODE solver, such as the Runge-Kutta method. These approximations use an algebraic approximation of the derivative on the left side of eqn. (1.2). In order to produce an accurate solution, these methods must use sufficiently small time steps to ensure the number density of any single isotope does not change too much during the time step. Many burnup problems involve very short-lived isotopes, which require very small time steps, making the time-stepping methods too slow to be practical.

Another class of solution methods are known as linear chains. One of these methods is the Analytic Laplacian Adaptive Radioactivity Analysis (ALARA) method [4]. ALARA breaks the burnup information into chains, creating a smaller ODE system. In this system the smaller burnup matrices are bidiagonal, allowing use of the analytic solution to the Bateman equations [2]. This class of solutions is limited by their inability to model the loops that often occur in burnup calculations. When an isotope absorbs a neutron, it can form an isotope that appears earlier in the chain causing the decay chain to repeat.

The class of methods to which the Chebyshev Rational Approximation Method (CRAM), belongs is matrix exponential methods. These methods approximate the matrix exponential defined by eqn. (1.4), which is notoriously tricky [3]. The Padé method is one of the most commonly used matrix exponential methods; however, it only works for matrices with small norms. Because of the large variation in half lives, burnup matrices can have norms up to 10^{17}, which are too large for Padé to handle [6]. In many burnup codes, such as Origen v2.2, the matrix is split into smaller matrices with more manageable norms and then Padé or another matrix exponential method is applied.

1.2. PyNE. PyNE (Python for Nuclear Engineering) [5] is a C++/Cython/Python package with tools for completing tasks common in nuclear engineering. PyNE is rare in the world of nuclear engineering in that it is open source. Because of the sensitive nature of much of the data and calculations used by nuclear engineers, most software is closed source and tightly regulated, meaning that it can be difficult for those without proper academic or lab affiliations to access. PyNE does not supplant the need for these closed source software packages, instead it works with the software and data libraries already installed on the user's computer, providing tools to parse, process, and expand on those closed packages and data.

PyNE has the ability to perform a range of calculations, including burnup. PyNE has an implementation of the ALARA method to perform burnup calculations, but in the time since that method was implemented, CRAM was introduced. CRAM takes uses a rational function approximation, allowing the exponential of the entire burnup matrix to be calculated at once instead of having to break it into smaller matrices as previous algorithms have done. It has theoretically been shown to be faster, and those results have been confirmed in practice [6]. In recent years, CRAM has been widely adopted with implementations in both Serpent 1.1.7 and Origen-ARP. Serpent and Origen are closed source codes that are not always easily accessible. We wanted to create an open source version of CRAM in PyNE, giving engineers more direct access to the method. Our implementation of CRAM serves as a complement to, not replacement of, the ALARA method that is already in use. The derivation, implementation, and testing of CRAM in PyNE are detailed in the following sections.

2. Chebyshev Rational Approximation.

Consider the Cauchy integral formula.

$$f(\zeta) = \frac{1}{2\pi i}\int_{\Gamma}\frac{f(z)}{z-\zeta}dz \tag{2.1}$$

Where f is analytic on and inside Γ and has a winding number of one around the point ζ.

This formula can be extended to matrix functions.

$$f(\mathbf{A}) = \frac{1}{2\pi i}\int_{\Gamma} f(z)(z\mathbf{I}-\mathbf{A})^{-1}dz \tag{2.2}$$

Where f is still analytic and the eigenvalues of A are inside the curve Γ. This gives the following representation of the matrix exponential:

$$e^{At} = \frac{1}{2\pi i}\int_{\Gamma} e^{z}(z\mathbf{I}-\mathbf{A}t)^{-1}dz. \tag{2.3}$$

This integral could be solved using Gaussian quadrature with Chebyshev weights on Γ, but as is shown in reference [7], quadrature can be viewed as a rational function, and we can use that rational function to derive an error estimate,

$$Q - Q_N = \frac{1}{2\pi i}\int_{\Gamma'} (e^z - r(z))f(z)dz \tag{2.4}$$

Where Q is the matrix exponential, Q_N is the quadrature sum, and $r(z)$ is a rational function. To compute the contour integral, we select a rational function that minimizes the error.

Rational functions are functions of the form $r_{j,k} = p_j(z)/q_k(z)$ where p_j and q_k are polynomials of degree j and k, respectively. The rational function chosen in CRAM minimizes the error in equation (2.4), and is therefore the best rational approximation of the exponential function on the negative real axis [6]. It can be described as the unique rational function, $\hat{r}_{k,k}$, satisfying

$$\sup_{x\in\mathbb{R}_-} |\hat{r}_{k,k}(x) - e^x| = \inf_{r_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}_-} |r_{k,k}(x) - e^x| \right\}. \tag{2.5}$$

Because all the poles are simple, we may write the rational function in partial fraction decomposition form:

$$r_{k,k} = \alpha_0 + \sum_{j=1}^{k} \frac{\alpha_j}{z - \theta_j}, \tag{2.6}$$

where

$\alpha_0 =$ limit of the function $r_{k,k}$ at infinity

$\alpha_j =$ residues at the poles θ_j.

We choose even k, which results in the poles forming conjugate pairs. We can then rewrite the function to sum over the poles with positive imaginary parts, reducing computational cost.

$$r_{k,k}(x) = \alpha_0 + 2Re\left(\sum_{j=1}^{k/2} \frac{\alpha}{x - \theta_j}\right) \tag{2.7}$$

We finally rewrite the solution to the burnup equation given in (1.3) as

$$\vec{N} = \alpha_0\vec{N}_0 + 2Re\left(\sum_{j=1}^{k/2} \alpha_j(\mathbf{A}t - \theta_j\mathbf{I})^{-1}\vec{N}_0\right). \tag{2.8}$$

This approximation converges quickly, giving roughly k correct digits for sufficiently large approximation orders k [8].

```
def _rat_apprx_16(theta, alpha, alpha_0, A, t, n_0):
    """ CRAM of order 16
    Parameters
    ----
    theta : Poles of the rational function r
    alpha : Residues at theses poles
    alpha_0 : Limit of r at infinity
    A : Burnup matrix
    t : Time step
    n_0 : Inital composition vector
    """
    s = np.len(theta)
    A = A*t
    n = 0*n_0

    for j in range(s):
        n += np.linalg.solve(A - theta[j] * np.identity(np.shape(A)
           [0]), alpha[j]*n_0)

    n = 2*n.real
    n = n + alpha_0*n_0
    return n
```

Fig. 2.1. *A python script similar to that implemented in PyNE, showing how a rational function is used to approximate the matrix exponential solution in eqn. (1.3). It is modeled after the MATLAB script in ref. [6] .*

3. Implementation in PyNE. The difficult aspect of using CRAM is determining the rational coefficients for a given k. For the purposes of PyNE we used the precomputed coefficients for $k = 14$ and $k = 16$ published in [6]. These coefficients can be viewed in Appendix A. Once the coefficients are computed, computing the matrix exponential with CRAM is very straightforward. Solving eqn. (2.7) requires solving k/2 sparse linear systems, which we do using the Python script in Fig. 2.1. For the purposes of our research code, we used `np.linalg.solve`, a dense linear solver, as the matrices we are interested in are small enough for a dense solve. As we continue to improve the code, we plan to add in sparse storage capabilities and use a sparse linear solver.

In PyNE we implemented the CRAM matrix exponential method as well as a separate method to generate burnup matrices. Given an array of nuclides supplied by the user, the method uses the nuclear data capabilities in PyNE to retrieve the appropriate decay constants and assemble the matrix. That matrix, along with the desired time step and initial composition vector, are passed to the CRAM method with k of either 14 or 16, depending on user selection. The code for that method can be seen in Fig. 2.1. For now, the method only takes into account decay and does not include the flux terms from eqn. (1.1). For other types of reactions, users must still use ALARA; however, we would like to introduce more capability into our CRAM solver in the future.

4. Testing and Results.

4.1. The Accuracy of the Matrix Exponential Method. To ensure accuracy, we ran various tests on our implementation of CRAM. To test the matrix exponential method, we compared our solution to the Padé and Taylor methods in the SciPy package [9]. We calculated average relative error over 100 trials. Relative error is given by $e = \frac{2|x-y|}{x+y}$, where x and y are the results of the methods in question. These errors can be seen in Table 4.1. For small matrices the three methods were in close agreement. One hundred trials of randomly generated 4x4 matrices gave average relative error of 2.67e-06 for both Padé and Taylor compared to CRAM-16. CRAM-14 showed an average relative error of 9.96e-05. As the matrices increased in size, the difference between CRAM, Padé, and Taylor increased greatly. However, these results do not necessarily indicate anything about the accuracy of CRAM. Padé and Taylor approximations produced significantly different results than each other, so neither could be used as the "true" value for purposes of comparison. This could be a result of both Padé and Taylor methods' difficultly with large norms [6].

4.2. The Speed of the Matrix Exponential Method. To test speed, we generated a random sparse 1700x1700 matrix to simulate a burnup matrix. Then, using the *%timeit* magic function in IPython [10], we ran three loops of the CRAM method and took the average time of the three. This process was repeated two more times and the minimum average value was returned. The times can be seen in Table 4.2. Note, *%timeit* returns the minimum average value instead of just the average as it is a better measure of the lower bound for how fast a given machine can run the script. Larger times are often caused by other processes interfering with timing accuracy, not variability in Python's speed [11]. Our tests showed a time of 18.8 seconds per loop for CRAM-16 and 21.3 seconds per loop for CRAM-14. Padé showed 42.9 seconds per loop. The Taylor method is not often used in practice because of poor convergence properties and was not included in this test [9]. All tests were run on a 2015 MacBook Pro with a 2.8 GHz Intel Core i7 processor.

4.3. Accuracy of the Burnup Calculation. The CRAM method implemented in PyNE has two main components: the assembly of the burnup matrix and the computation of the matrix exponential. We have been assured of the accuracy of the exponential method by the tests above, so the following test was primarily to ensure the accuracy of the method as a whole. To test, we compared against the decay method in the materials class in PyNE, which has been benchmarked against Origen v2.2 [12]. As a preliminary test, we selected 16 different nuclides and calculated the relative error with the PyNE decay method. Both CRAM-14 showed a mean relative error of 4.91e-03, and CRAM-16 showed an error of 4.83e-03. The errors for each nuclide can be seen in Table 4.3. All tests are available in the test suite of PyNE hosted on GitHub (https://github.com/pyne/pyne).

Matrix Exponential Relative Errors			
CRAM-14		CRAM-16	
Padé	Taylor	Padé	Taylor
1.6309965E-06	1.630994E-06	8.454727E-05	8.454728E-05

Table 4.1
The relative errors between the CRAM, Padé, and Taylor matrix exponential methods. These errors are the mean of 100 trials of randomly generated 4x4 matrices.

Time Test		
CRAM-14	CRAM-16	Padé
21.3 s per loop	18.8 s per loop	42.9 s per loop

Table 4.2
Minimum average speed per loop for a randomly generated 1700x1700 sparse matrix.

Decay Tests		
Nuclides	CRAM-14 Mean Relative Error	CRAM-16 Mean Relative Error
H3	9.376389E-05	3.322047E-06
C14	9.645335E-05	5.998076E-06
V50	3.802240E-02	3.796358E-02
Rb87	9.301265E-05	1.747884E-06
In115	9.277929E-05	2.366546E-06
Sr90	1.355294E-04	1.350559E-05
Te123	9.445574E-05	3.047969E-07
Te130	1.008991E-04	1.050536E-05
I131	9.443619E-05	4.114517E-06
Cs137	9.506458E-05	4.850236E-06
La138	4.441119E-04	3.852850E-04
Nd144	1.420050E-02	1.411019E-02
Sm147	1.390944E-02	1.381905E-02
Lu176	9.409866E-05	3.643393E-06
Re187	9.474549E-05	1.504736E-08
Os186	1.097822E-02	1.088794E-02

Table 4.3
The average relative errors for decay chains of various nuclides as compared to the decay method in PyNE.

5. Future Work. We plan to create a more comprehensive benchmark study of CRAM, including comparisons with Origen-ARP, the latest version of Origen, which now uses a CRAM method to perform burnup calculations. We would also like to extend the capability of CRAM in PyNE to handle reactions other than decay, including fission, capture, and absorption.

Appendix A.

TABLE I

Partial Fraction Decomposition Coefficients for CRAM Approximation of Order 14

Coefficient	Real Part	Imaginary Part
θ_1	$-8.897\,773\,186\,468\,888\,819\,9 \times 10^{0}$	$+1.663\,098\,261\,990\,208\,530\,4 \times 10^{1}$
θ_2	$-3.703\,275\,049\,423\,448\,060\,3 \times 10^{0}$	$+1.365\,637\,187\,148\,326\,817\,1 \times 10^{1}$
θ_3	$-0.208\,758\,638\,250\,130\,125\,1 \times 10^{0}$	$+1.099\,126\,056\,190\,126\,091\,3 \times 10^{1}$
θ_4	$+3.993\,369\,710\,578\,568\,519\,4 \times 10^{0}$	$+6.004\,831\,642\,235\,037\,317\,8 \times 10^{0}$
θ_5	$+5.089\,345\,060\,580\,624\,506\,6 \times 10^{0}$	$+3.588\,824\,029\,027\,006\,510\,2 \times 10^{0}$
θ_6	$+5.623\,142\,572\,745\,977\,124\,8 \times 10^{0}$	$+1.194\,069\,046\,343\,966\,976\,6 \times 10^{0}$
θ_7	$+2.269\,783\,829\,231\,112\,709\,7 \times 10^{0}$	$+8.461\,737\,973\,040\,221\,401\,9 \times 10^{0}$
α_1	$-7.154\,288\,063\,589\,067\,285\,3 \times 10^{-5}$	$+1.436\,104\,334\,954\,130\,011\,1 \times 10^{-4}$
α_2	$+9.439\,025\,310\,736\,168\,877\,9 \times 10^{-3}$	$-1.718\,479\,195\,848\,301\,751\,1 \times 10^{-2}$
α_3	$-3.763\,600\,387\,822\,696\,871\,7 \times 10^{-1}$	$+3.351\,834\,702\,945\,010\,421\,4 \times 10^{-1}$
α_4	$-2.349\,823\,209\,108\,270\,119\,1 \times 10^{1}$	$-5.808\,359\,129\,714\,207\,400\,4 \times 10^{0}$
α_5	$+4.693\,327\,448\,883\,129\,304\,7 \times 10^{1}$	$+4.564\,364\,976\,882\,776\,079\,1 \times 10^{1}$
α_6	$-2.787\,516\,194\,014\,564\,646\,8 \times 10^{1}$	$-1.021\,473\,399\,905\,645\,143\,4 \times 10^{2}$
α_7	$+4.807\,112\,098\,832\,508\,890\,7 \times 10^{0}$	$-1.320\,979\,383\,742\,872\,388\,1 \times 10^{0}$
α_0	$+1.832\,174\,378\,254\,041\,275\,1 \times 10^{-14}$	$+0.000\,000\,000\,000\,000\,000\,0 \times 10^{0}$

TABLE II

Partial Fraction Decomposition Coefficients for CRAM Approximation of Order 16

Coefficient	Real Part	Imaginary Part
θ_1	$-1.084\,391\,707\,869\,698\,802\,6 \times 10^{1}$	$+1.927\,744\,616\,718\,165\,228\,4 \times 10^{1}$
θ_2	$-5.264\,971\,343\,442\,646\,889\,5 \times 10^{0}$	$+1.622\,022\,147\,316\,792\,730\,5 \times 10^{1}$
θ_3	$+5.948\,152\,268\,951\,177\,480\,8 \times 10^{0}$	$+3.587\,457\,362\,018\,322\,282\,9 \times 10^{0}$
θ_4	$+3.509\,103\,608\,414\,918\,097\,4 \times 10^{0}$	$+8.436\,198\,985\,884\,375\,082\,6 \times 10^{0}$
θ_5	$+6.416\,177\,699\,099\,434\,192\,3 \times 10^{0}$	$+1.194\,122\,393\,370\,138\,687\,4 \times 10^{0}$
θ_6	$+1.419\,375\,897\,185\,665\,978\,6 \times 10^{0}$	$+1.092\,536\,348\,449\,672\,258\,5 \times 10^{1}$
θ_7	$+4.993\,174\,737\,717\,996\,399\,1 \times 10^{0}$	$+5.996\,881\,713\,603\,942\,226\,0 \times 10^{0}$
θ_8	$-1.413\,928\,462\,488\,886\,211\,4 \times 10^{0}$	$+1.349\,772\,569\,889\,274\,538\,9 \times 10^{1}$
α_1	$-5.090\,152\,186\,522\,491\,565\,0 \times 10^{-7}$	$-2.422\,001\,765\,285\,228\,797\,0 \times 10^{-5}$
α_2	$+2.115\,174\,218\,246\,603\,090\,7 \times 10^{-4}$	$+4.389\,296\,964\,738\,067\,391\,8 \times 10^{-3}$
α_3	$+1.133\,977\,517\,848\,393\,052\,7 \times 10^{2}$	$+1.019\,472\,170\,421\,585\,645\,0 \times 10^{2}$
α_4	$+1.505\,958\,527\,002\,346\,752\,8 \times 10^{1}$	$-5.751\,405\,277\,642\,181\,997\,9 \times 10^{0}$
α_5	$-6.450\,087\,802\,553\,964\,659\,5 \times 10^{1}$	$-2.245\,944\,076\,265\,209\,605\,6 \times 10^{2}$
α_6	$-1.479\,300\,711\,355\,799\,971\,8 \times 10^{0}$	$+1.768\,658\,832\,378\,293\,790\,6 \times 10^{0}$
α_7	$-6.251\,839\,246\,320\,791\,889\,2 \times 10^{1}$	$-1.119\,039\,109\,428\,322\,848\,0 \times 10^{1}$
α_8	$+4.102\,313\,683\,541\,002\,127\,3 \times 10^{-2}$	$-1.574\,346\,617\,345\,546\,819\,1 \times 10^{-1}$
α_0	$+2.124\,853\,710\,495\,223\,748\,8 \times 10^{-16}$	$+0.000\,000\,000\,000\,000\,000\,0 \times 10^{0}$

Chebyshev rational function partial fraction decomposition coefficients used in PyNE [6].

REFERENCES

[1] D. Knott and A. Yamamoto, *Lattice Physics Computations*, in Handbook of Nuclear Engineering, Vol. 2 Reactor Design, D. Cacuci, ed., Springer Science+Business Media, New York, NY, 2010, pp. 913–1241.

[2] H. Bateman, *Solution of a System of Differential Equations Occurring in the Theory of Radio-active Transformations*, Cambridge Phil. Soc, 15 (1910).

[3] C. Moler and C. Van Loan, *Nineteen Dubious Ways to Compute the Exponential of a Matrix, Twenty-Five Years Later*, SIAM Review, 5 (2003), pp. 3-49.

[4] P. Wilson, *ALARA: Analytic and Laplacian Adaptive Radioactivity Analysis*, Ph.D. thesis, University of Wisconsin, Madison, WI, 1999.

[5] A. Scopatz, P. Romano, P. Wilson and K. Huff, *PyNE: Python for Nuclear Engineering*, Am. Nuc. Soc. Winter Meeting 2012, 107 (2012).

[6] M. Pusa, *Rational Approximations to the Matrix Exponential in Burnup Calculations*, Nuclear Science and Engineering, 169 (2011), pp. 155-167.

[7] L. N. Trefethen, J. A. C. Weideman, and T. Schmelzer, *Talbot Quadratures and Rational Approximations*, BIT, 46 (2006).

[8] A. A. Gonchar and E. A. Rakhmanov, *Equilibrium Distributions and Degree of Rational Approximation of Analytic Functions* Math. USSR Sb., 62 (1989)

[9] The SciPy community, *SciPy v0.16.0 Reference Guide* http://docs.scipy.org/ (24 Jul 2015).

[10] The IPython Development Team, *IPython Documentation* http://ipython.readthedocs.org

[11] Python Software Foundation, *Documentation, The Python Standard Library* https://docs.python.org/ (23 May 2015)

[12] A. Scopatz, *ORIGEN v2.2 Benchmark Study*, http://nbviewer.ipython.org/github/pyne/sand box/blob/master/origen-cmp.ipynb .