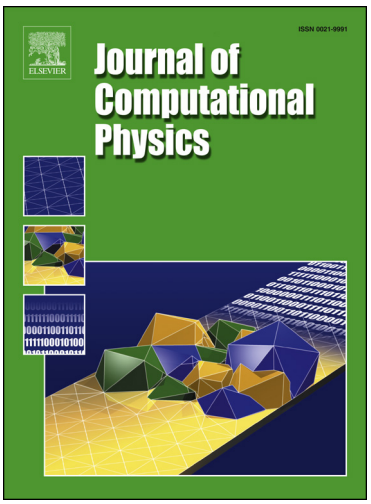
Accepted Manuscript



High order methods for the integration of the Bateman equations and other problems of the form of $y' = F(y,t)y$

C. Josey, B. Forget, K. Smith

PII: S0021-9991(17)30596-X
DOI: http://dx.doi.org/10.1016/j.jcp.2017.08.025
Reference: YJCPH 7527

To appear in: *Journal of Computational Physics*

Received date: 21 November 2016
Accepted date: 12 August 2017

Please cite this article in press as: C. Josey et al., High order methods for the integration of the Bateman equations and other problems of the form of $y' = F(y,t)y$, *J. Comput. Phys.* (2017), http://dx.doi.org/10.1016/j.jcp.2017.08.025

High Order Methods for the Integration of the Bateman Equations and Other Problems of the Form of $y' = F(y,t)y$

C. Josey[a,*], B. Forget[a], K. Smith[a]

[a] *Massachusetts Institute of Technology, 77 Massachusetts Ave, Cambridge, MA 02139*

Abstract

This paper introduces two families of A-stable algorithms for the integration of $y' = F(y,t)y$: the extended predictor-corrector (EPC) and the exponential-linear (EL) methods. The structure of the algorithm families are described, and the method of derivation of the coefficients presented. The new algorithms are then tested on a simple deterministic problem and a Monte Carlo isotopic evolution problem.

The EPC family is shown to be only second order for systems of ODEs. However, the EPC-RK45 algorithm had the highest accuracy on the Monte Carlo test, requiring at least a factor of 2 fewer function evaluations to achieve a given accuracy than a second order predictor-corrector method (center extrapolation / center midpoint method) with regards to Gd-157 concentration.

Members of the EL family can be derived to at least fourth order. The EL3 and the EL4 algorithms presented are shown to be third and fourth order respectively on the systems of ODE test. In the Monte Carlo test, these methods did not overtake the accuracy of EPC methods before statistical uncertainty dominated the error.

The statistical properties of the algorithms were also analyzed during the Monte Carlo problem. The new methods are shown to yield smaller standard deviations on final quantities as compared to the reference predictor-corrector method, by up to a factor of 1.4.

Keywords: Depletion, Bateman equations, Burnup calculations, Predictor-corrector, Exponential-linear

1. Introduction

In the study of nuclear reactors, it is often useful to know how the composition of fuel develops throughout the life of the reactor. This process as a whole is known as depletion. This evolution with time is governed by the Bateman equations [1]. In its most simple form, the Bateman equations can be written as $y' = F(y,t)y$, where y is a vector of nuclide quantities, and F is a matrix representing decay and reaction rates.

The function F can be calculated in a few ways, but the two main categories of interest are deterministic and stochastic methods. Deterministic solutions can be extremely quick, depending on how one approximates the physics. Conversely, a typical Monte Carlo function evaluation makes no approximations, but can take many hundreds of thousands of CPU hours to evaluate. Thus, when F is evaluated with Monte Carlo, the cost to integrate y in time is almost entirely due to the number of function evaluations.

This conflicts with the main difficulty in solving the Bateman equations. The equations are incredibly stiff. While the average engineering problem considers time scales of months, the eigenvalues of F can have values on the order of -10^{21} s^{-1} [2]. Section 2 of this paper focuses on how current methods solve this problem. The predictor and predictor-corrector family of methods are introduced, as well as the matrix exponent these algorithms rely on.

*Corresponding author

Email addresses: cjosey@mit.edu (C. Josey), bforget@mit.edu (B. Forget), kord@mit.edu (K. Smith)

The rest of the paper focuses on two new families of methods for integrating y that minimize the number of function evaluations so as to improve Monte Carlo performance. The first family is a simple extension of current predictor-corrector methods, and is detailed in Section 3. These methods are high order for single ODEs, but only second order for systems of ODEs. However, these methods can still be useful, as the leading coefficients of their temporal truncation error terms are quite a bit smaller than commonly used methods.

In order to get beyond second order, the Taylor series must be derived for both vector y and for the matrix exponent. These expansions are derived in Section 4. The results of this section are then used to derive a true third order algorithm and a true fourth order algorithm in Section 5.

Finally, a simple test case designed to maximize difficulty for the integrator is presented in Section 6, and the algorithms are tested against one another for accuracy and statistical properties in Section 7.

2. The Bateman Equations and Current Methods

First, it would be best to identify precisely why the Bateman equations are so hard to solve, so as to explain decisions made later. The general Bateman equation is shown as Equation (1) [1].

$$\begin{aligned}\frac{dN_i(t)}{dt} = &\sum_j \left[\int_0^\infty \sigma_{j\to i}(E,t)\phi(E,t)dE + \lambda_{j\to i}\right] N_j(t) \\ &- \sum_j \left[\int_0^\infty \sigma_{i\to j}(E,t)\phi(E,t)dE + \lambda_{i\to j}\right] N_i(t)\end{aligned} \tag{1}$$

Where:

$$\begin{aligned}N_i(t) &= \text{nuclide } i \text{ quantity at time } t \\ \sigma_{i\to j}(E,t) &= \text{microscopic cross section of a reaction} \\ &\quad \text{where nuclide } i \text{ generates } j \text{ at} \\ &\quad \text{neutron energy } E \text{ and time } t \\ \phi(E,t) &= \text{neutron flux at energy } E \text{ and time } t \\ \lambda_{i\to j} &= \text{decay constant of nuclide } i \text{ to } j\end{aligned}$$

This equation is strongly dependent on N in a linear way, and weakly dependent on N in a nonlinear way (via the change in flux as a function of time). As such, Equation (1) can be simplified into a matrix equation of the form in Equation (2).

$$y'(t) = F\left(y(t), t\right) y(t) \tag{2}$$

Where:

$$\begin{aligned}y(t) &= \text{vector of nuclide quantities} \\ F(y(t),t) &= \text{decay matrix at time } t \\ &\quad \text{for nuclide quantity } y\end{aligned}$$

This equation could in theory be solved with any common ODE solution method. The issue is that the eigenvalues of matrix F typically span $[-10^{21}, 0]$ s^{-1} [2]. In order for a normal ODE solver to stably integrate the ODE, the product of the span of eigenvalues and the time step must lie entirely within the stable region of a method. For example, explicit Euler has a stability disk which covers the real axis from $[-2, 0]$. The time step would then have to be on the order of 10^{-21} s to get a stable result.

As such, it is effectively impossible to perform depletion without an A-stable algorithm. Implicit methods may possibly be used, but the added cost of attempting to invert the transport operator would make that path also impractical. This lead to the development of the predictor and predictor-corrector methods. These methods leverage the analytical solution for a constant F to eliminate the stiffness. These are discussed in Section 2.1 and Section 2.2, respectively.

2.1. Predictor Method

The simplest A-stable solution for Equation (2) is the predictor method. It is derived by first noting that if F is assumed to be a constant matrix, then the ODE has an exact solution. This solution is given as Equation (3).

$$y(t) = e^{Ft}y(0) \tag{3}$$

So long as the matrix exponential can be numerically resolved, for which a great number of methods have been developed [2, 3], this equation can be evaluated.

In order to allow for an F that varies with y and t, F is then approximated as piecewise constant using an F evaluated at the beginning of the time step. For each constant region n, the integration can be performed as before. This method is shown in Equation (4), where h is the time span in which $F = F_n$.

$$y_{n+1} = e^{F_n h}y_n \tag{4}$$

The predictor method can be shown to converge $\mathcal{O}(h)$. This is often insufficient due to the number of F evaluations required, so the predictor-corrector methods were developed.

2.2. Predictor-Corrector

The idea behind predictor-corrector methods is that the method described earlier in Equation (4) only uses information at the beginning of the time step and can only provide a low quality prediction of y_{n+1}. Instead, predictor-corrector integrates to some intermediate step, evaluates F at that intermediate step, and combines the two evaluations of F in some way to integrate to the end of the step. The general structure of a two-stage predictor-corrector method is shown in Equation (5).

$$\begin{aligned} x &= e^{ha_{11}F(y_n,t_n)}y_n \\ y_{n+1} &= e^{h[a_{21}F(y_n,t_n)+a_{22}F(x,t_n+a_{11}h)]}y_n \end{aligned} \tag{5}$$

In the first equation, y is integrated using predictor to some intermediate step x at $t = t_n + a_{11}h$. F is then evaluated at x, and in the second equation mixed (using the coefficients a_{21} and a_{22}) with the original F to perform the final integration.

There are quite a few predictor-corrector methods available [4]. The most common of these methods are the so-called "CE/CM" and the "CE/LI" methods. CE/CM stands for constant extrapolation, constant midpoint, and is the algorithm used for depletion in MCNP6 [5]. In this algorithm, the predictor depletes to the midpoint, evaluates F, and uses only this new F as the average. This method is shown in Equation (6).

$$\begin{aligned} x &= e^{\frac{h}{2}F(y_n,t_n)}y_n \\ y_{n+1} &= e^{hF\left(x,t+\frac{h}{2}\right)}y_n \end{aligned} \tag{6}$$

CE/LI stands for constant extrapolation, linear interpolation, and is the default algorithm in Serpent [6]. The predictor integrates all the way to the end of the time step. The F from the end of the time step is then averaged with the F at the beginning, and this new function is used to deplete from the beginning to the end again. The algorithm is shown in Equation (7).

$$\begin{aligned} x &= e^{hF(y_n,t_n)}y_n \\ y_{n+1} &= e^{h\left[\frac{1}{2}F(y_n,t_n)+\frac{1}{2}F(x,t_n+h)\right]}y_n \end{aligned} \tag{7}$$

Both of these methods are $\mathcal{O}(h^2)$. A summary of the properties of the predictor and predictor-corrector methods is given in Table 1. Notably, the CE/CM algorithm has lower memory requirements, as a sum over multiple F does not occur.

There are also a few methods that take advantage of previous time steps. One such method is "LE/QI", the linear extrapolation, quadratic interpolation method. Unfortunately, this algorithm relies on F being continuous everywhere (not just within a time step), lowering its utility.

Method	Order	Stages	# F stored
Predictor	1	1	1
CE/CM	2	2	1
CE/LI	2	2	2

Table 1: Predictor and predictor corrector method properties

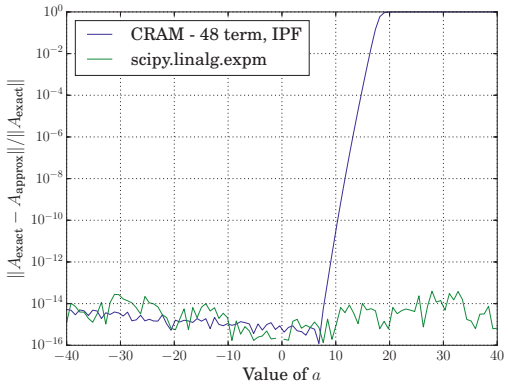


Figure 1: 48 term CRAM matrix exponential in incomplete partial fraction form and `scipy.linalg.expm` compared against an analytic solution

2.3. Matrix Exponentials

As mentioned earlier, there are quite a number of matrix exponential methods. In the case of depletion, we value two aspects in a matrix exponential. The first is that no dense matrices are formed. F is very sparse, so a dense matrix would increase memory consumption by orders of magnitude. The second is that the method has to be highly accurate for matrix eigenvalues spanning $[-10^{21}, 0]$. The Chebyshev rational approximation method (CRAM) [2] was specifically developed to incorporate both of these. As such, it will be the matrix exponential used elsewhere in this paper.

There is one weakness in the method, which will become important during the development of the exponential-linear methods in Section 5. While the method is valid for negative eigenvalues, error rapidly increases for positive ones. To demonstrate this issue, a simple problem was tested. A matrix M is formed as shown in Equation (8).

$$\begin{aligned} M(a) &= \begin{pmatrix} a & 1 \\ 1 & a \end{pmatrix} \\ A &= e^{M(a)} \\ A_{\text{exact}} &= \frac{1}{2}\begin{pmatrix} e^{a-1} + e^{a+1} & e^{a+1} - e^{a-1} \\ e^{a+1} - e^{a-1} & e^{a+1} + e^{a-1} \end{pmatrix} \end{aligned} \tag{8}$$

The eigenvalues of M are $\lambda = \{a - 1, a + 1\}$. We then take the exponent of this matrix, for which an analytical representation is known. In Figure 1, this analytic solution is compared against both CRAM and the method used in SciPy, which combines a Padé approximation with a scaling and squaring step [7, 8]. The downside of scaling and squaring is that it requires the formation of dense matrices to be efficient.

CRAM is extremely accurate for negative values of a. Once it reaches $a = 7$, roughly, the error rapidly increases. If positive eigenvalues are a possibility (such as in point kinetics, for example), or the problem does not require sparse matrices for memory reasons, the Padé method used in SciPy is a suitable alternative.

3. Improving on Predictor-Corrector

The first possible way to improve on predictor-corrector is to add more intermediate steps. The general method would take the form of Equation (9).

$$\begin{aligned} x_1 &= y_n \\ x_i &= e^{h\sum_{j=1}^{i-1} a_{ij} F(x_j, t_n + c_j)} y_n \\ y_{n+1} &= e^{h\sum_{j=1}^{s} b_j F(x_j, t_n + c_j)} y_n \end{aligned} \tag{9}$$

In the more common Runge-Kutta methods, the coefficients of a method are commonly arranged into a Butcher's tableau [9]:

$$\begin{array}{c|cccc} 0 & & & & \\ c_2 & a_{21} & & & \\ c_3 & a_{31} & a_{32} & & \\ \vdots & \vdots & & \ddots & \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss-1} \\ \hline & b_1 & b_2 & \dots & b_{s-1} \quad b_s \end{array}$$

The coefficients of Equation (9) also fit into such a tableau form. As such, we can convert the methods described in Section 2 into a tableau shorthand. For example, the predictor method is given by Equation (10), the CE/LI method is given by Equation (11), and CE/CM is given by Equation (12).

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array} \tag{10}$$

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \tag{11}$$

$$\begin{array}{c|cc} 0 & & \\ \frac{1}{2} & \frac{1}{2} & \\ \hline & 0 & 1 \end{array}. \tag{12}$$

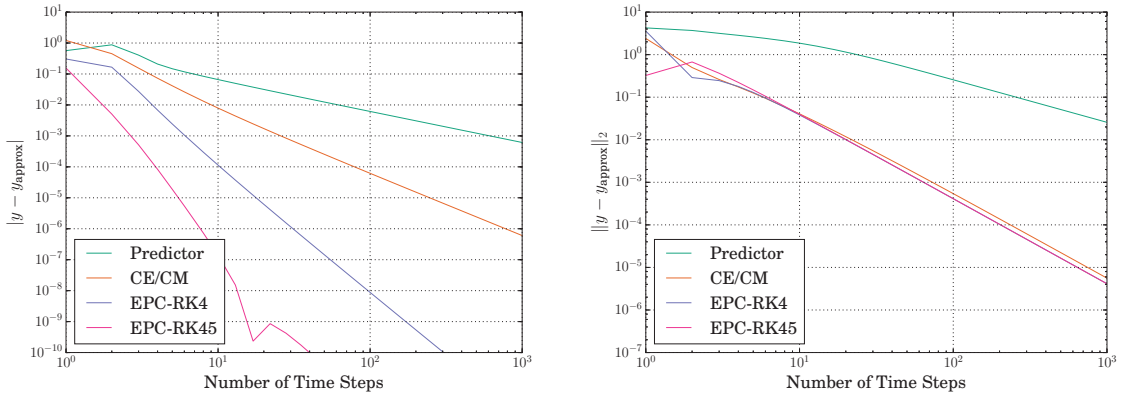
Every one of these methods have tableaux that match those of more conventional linear methods. The predictor method matches that of explicit Euler, and both CE/CM and CE/LI have valid 2nd order Runge-Kutta tableaux [10].

Since all of these methods have matching tableaux to normal Runge-Kutta methods, one might think that high order Runge-Kutta methods might yield superior results. This is the case to an extent. One of the easiest ways to show that a method has a given order is to assume y is a scalar and perform a Taylor series expansion of y and an approximate y. If, when subtracted from one another, all the terms $\mathcal{O}(h^m), m \leq n$ are zero, then the method is order n.

If Equation (9) with coefficients from an order n Runge-Kutta tableau are used as the approximate y, one will find that the resulting approximate y is also order n. However, it is incorrect to assume that this will still apply when y is a vector. For example, it is possible to form a Runge-Kutta method that is 5th order for a single ODE, and 4th order for a system [11].

In order to test this with this new family of methods, two simple problems were run. The first is an ODE in which y is scalar, shown in Equation (13).

$$\begin{aligned} y' &= \sin(y) y \\ y(0) &= 1 \\ y(1.5) &\approx 2.965401170854292 \end{aligned} \tag{13}$$



(a) Single ODE test

(b) System of ODEs test

Figure 2: The error for the extended predictor-corrector methods as a function of time step count

The second one is a simple system of ODEs, shown in Equation (14).

$$\begin{aligned} y_1' &= \sin(y_2)y_1 + \cos(y_1)y_2 \\ y_2' &= -\cos(y_2)y_1 + \sin(y_1)y_2 \\ y_1(0) &= 1 \\ y_2(0) &= 1 \\ y_1(1.5) &\approx 2.3197067076743316 \\ y_2(1.5) &\approx 3.1726475740397628 \end{aligned} \tag{14}$$

(15)

Each problem was integrated from $t = 0$ to $t = 1.5$. A variety of algorithms were tested. The first was the predictor method from Section 2.1, which should be first order. The second was the CE/CM method from Section 2.2, which should be second order. Then, the tableau for the traditional Runge-Kutta 4 method was used in Equation (9) (abbreviated as "EPC-RK4"). This method has a scalar order of 4. Finally, the Cash-Karp RK45 tableau [12] was used in Equation (9) (abbreviated as "EPC-RK45"). This method has a scalar order of 5.

The results for these tests are shown in Figure 2a and Figure 2b, for the single ODE and system of ODEs respectively. For the scalar problem, all algorithms converge at their expected rate. For the system, however, the EPC-RK4 and EPC-RK45 methods converge merely at second order.

As will be shown in Section 4, the reason for this is that y''' has fewer terms in scalar arithmetic than in the vector form. The resulting system of equations to generate a third order EPC method for scalar arithmetic can be solved without issue. The additional equations required for a system of ODEs generates mutually exclusive equations that cannot be solved simultaneously. As such, it is impossible to generate a true third order EPC method. To get around this, one needs a different form of integrator, such as the exponential-linear form presented in Section 5.

It is worth noting that this does not make the EPC-RK4 and EPC-RK45 methods useless. While they are not better than second order, the coefficients of many of the truncation terms are smaller than those of current methods. As will be shown in Section 7, this can yield significant performance advantages when moderate accuracy is needed. These new methods are summarized in Table 2. Although the EPC-RK45 method is 6 stages, some coefficients are zero such that fewer F need to be stored.

Method	Order	Stages	# F stored
EPC-RK4	2	4	4
EPC-RK45	2	6	5

Table 2: Extended predictor-corrector method properties

4. Series Expansions for Vector y

In the prior section, it was shown that a simple scalar Taylor series of y is not sufficient to derive a high-order method for a system of ODEs. This section introduces the mathematical framework required to derive such a method. First, the series for y is derived in Section 4.1, and then the series for the general exponent of sums of matrices is derived in Section 4.2.

4.1. Series Expansion of y

First, we expand the matrix multiply required for Equation (2) into the form shown in Equation (16).

$$y_i' = \sum_{j=1}^{N} f_{ij}(y) y_i \tag{16}$$

In this form, one can apply the chain rule to calculate y''. Since the first derivative of y is known, it can be substituted in as well.

$$\begin{aligned} y_i'' = &\sum_{j=1}^{N} f_{ij}(y) \sum_{k=1}^{N} f_{jk}(y) y_k \\ &+ \sum_{j=1}^{N} \sum_{k=1}^{N} \frac{\partial f_{ij}(y)}{\partial y_k} y_i \sum_{l=1}^{N} f_{kl}(y) y_l \end{aligned}$$

For compactness, it is useful to introduce a set of operators.

$$\begin{aligned} \mathbf{f}_u v &= \sum_{j=1}^{N} f_{ij}(u) v_j \\ \mathbf{f}_u{}'(v; w) &= \sum_{j,k=1}^{N} \frac{\partial f_{ij}(u)}{\partial u_k} v_j w_k \\ \mathbf{f}_u{}''(v; w, x) &= \sum_{j,k,l=1}^{N} \frac{\partial^2 f_{ij}(u)}{\partial u_k \partial u_l} v_j w_k x_l \end{aligned}$$

Here, u, v, w, and x are arbitrary vectors. This set of operators can be extended for any $\mathbf{f}_u{}^{(n)}$ as needed. These operators have a few useful properties. First, they are linear on each component.

$$\begin{aligned} \mathbf{f}_u{}'(v; w + \alpha x) &= \mathbf{f}_u{}'(v; w) + \alpha \mathbf{f}_u{}'(v; x) \\ \mathbf{f}_u{}'(v + \alpha x; w) &= \mathbf{f}_u{}'(v; w) + \alpha \mathbf{f}_u{}'(x; w) \\ \mathbf{f}_u{}'(v; 0) &= 0 \end{aligned}$$

Secondly, the first vector before the semicolon is index linked to f_{ij}, and the rest are index linked to the ∂u terms. As such, the vectors after the semicolon are interchangeable.

$$\begin{aligned} \mathbf{f}_u{}''(v; w, x) &= \mathbf{f}_u{}''(v; x, w) \\ \mathbf{f}_u{}''(v; w, x) &\neq \mathbf{f}_u{}''(w; v, x) \end{aligned}$$

Finally, taking derivatives of these operators is relatively easy.

$$\begin{aligned}\frac{\partial \mathbf{f}_u{}^{(n)}(v;w,x,\ldots)}{\partial t} &= \mathbf{f}_u{}^{(n+1)}(v;w,x,\ldots,u')\\ &+ \mathbf{f}_u{}^{(n)}(v';w,x,\ldots)\\ &+ \mathbf{f}_u{}^{(n)}(v;w',x,\ldots)\\ &+ \mathbf{f}_u{}^{(n)}(v;w,x',\ldots)+\ldots\end{aligned}$$

Using this new syntax, a much more compact form of the derivatives of y can be formed.

$$\begin{aligned}y' &= \mathbf{f}_y y\\ y'' &= \mathbf{f}_y{}'(y,\mathbf{f}_y y)+\mathbf{f}_y\mathbf{f}_y y\end{aligned}$$

Using this procedure, the third derivative can be written compactly as well.

$$\begin{aligned}y''' &= 2\mathbf{f}_y{}'(\mathbf{f}_y y;\mathbf{f}_y y)+\mathbf{f}_y\mathbf{f}_y{}'(y;\mathbf{f}_y y)+\mathbf{f}_y\mathbf{f}_y\mathbf{f}_y y\\ &+ \mathbf{f}_y{}''(y;\mathbf{f}_y y,\mathbf{f}_y y)+\mathbf{f}_y{}'(y;\mathbf{f}_y{}'(y;\mathbf{f}_y y))\\ &+ \mathbf{f}_y{}'(y;\mathbf{f}_y\mathbf{f}_y y)\end{aligned}$$

This can then be continued onwards to any order necessary. It is worth comparing this to the equivalent scalar forms.

$$\begin{aligned}y' &= f(y)y\\ y'' &= f'(y)f(y)y^2+f(y)^2y\\ y''' &= f''(y)f(y)^2y^3+4f'(y)f(y)^2y^2\\ &+ f'(y)^2f(y)y^3+f(y)^3y\end{aligned}$$

While the scalar y'' has the same number of terms and the same approximate structure as the vector y'', this is not the case for y'''. Here, all terms with a single $\mathbf{f}_y{}'$ collapse together. This reduces the number of constraint equations for a 3rd order method from 6 to 4, making it possible for a method to be 3rd order for scalar arithmetic and 2nd order for vector arithmetic.

4.2. Series Expansion of $e^{\sum F(x_i)}v$

A similar procedure can be performed for the matrix exponential. First, we replace the matrix exponential with an infinite sum.

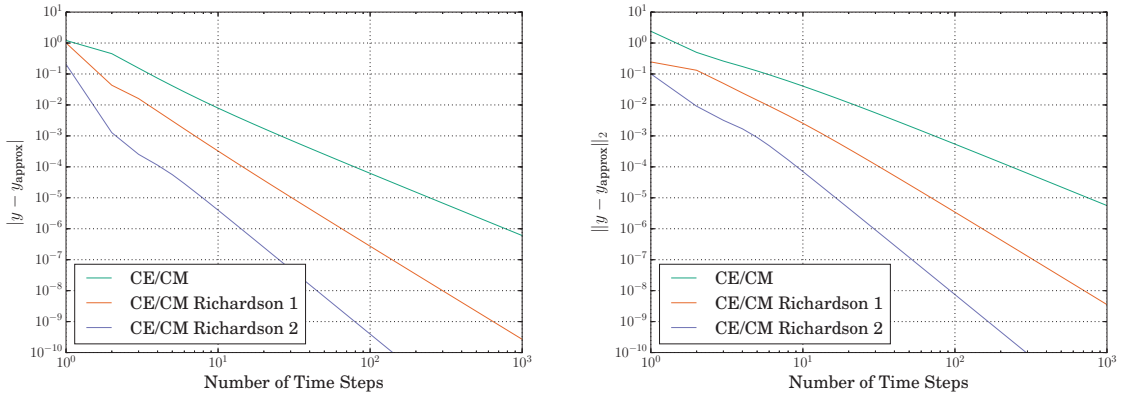
$$\begin{aligned}u &= e^{\sum_j ha_jF(x_j)}v\\ u &= \sum_{k=1}^{\infty}\frac{1}{k!}\left(\sum_j a_jh\mathbf{f}_{x_j}\right)^k v\end{aligned}$$

Let's then assume for simplicity that there is only two $\mathbf{f}_{x_j}$ in the above sum.

$$u=\sum_{k=1}^{\infty}\frac{1}{k!}\left(ha_1\mathbf{f}_{x_1}+ha_2\mathbf{f}_{x_2}\right)^k v$$

Truncating to second order yields:

$$\begin{aligned}u &= v+ha_1\mathbf{f}_{x_1}v+ha_2\mathbf{f}_{x_2}v\\ &+\frac{h^2}{2}a_1^2\mathbf{f}_{x_1}\mathbf{f}_{x_1}v+\frac{h^2}{2}a_1a_2\mathbf{f}_{x_1}\mathbf{f}_{x_2}v\\ &+\frac{h^2}{2}a_1a_2\mathbf{f}_{x_2}\mathbf{f}_{x_1}v+\frac{h^2}{2}a_2^2\mathbf{f}_{x_2}\mathbf{f}_{x_2}v+\mathcal{O}(h^3)\end{aligned}$$



(a) Single ODE test

(b) System of ODEs test

Figure 3: The error of CE/CM, CE/CM Richardson extrapolated once, and Richardson extrapolated twice as a function of time step count

To perform the Taylor series, one then calculates the derivatives of u as shown earlier. Then, one substitutes derivatives of x_1, x_2, and v as needed. Using these building blocks, one then constructs the series expansion of y_{approx}, which can then be used with the series expansion of y to derive a method.

5. Exponential-Linear Method

Now that the framework is fully derived, a formalism is needed such that high order is achievable. As mentioned earlier, simply extending predictor-corrector cannot achieve third order for systems of ODEs, so another approach must be taken.

One way to generate a high order method is to apply Richardson extrapolation to a working low order method, such as CE/CM. Each application should increase the order of a method by one. For example, Richardson extrapolation with CE/CM should create a 3rd order 5 term method. A double application on CE/CM should create a 4th order 12 term method.

Both of these Richardson extrapolated forms were tested on the same ODE test problems from Section 3. The results are shown in Figure 3a and Figure 3b for the single ODE and system of ODEs cases, respectively. As shown, the extrapolation yields methods that are truly higher order, yielding third and fourth order methods.

A single application of Richardson acceleration yields the algorithm in Equation (17).

$$\begin{aligned} x_1 &= e^{h/2F(y_n,t)}y_n \\ x_2 &= e^{h/4F(y_n,t)}y_n \\ x_3 &= e^{h/2F(x_2,t+h/4)}y_n \\ x_4 &= e^{h/4F(x_3,t+h/2)}x_3 \\ y_{n+1} &= \frac{4}{3}e^{h/2F(x_4,t+3h/4)}x_3 - \frac{1}{3}e^{hF(x_1,t+h/2)}y_n \end{aligned} \tag{17}$$

The two distinctions from extended predictor-corrector are that the matrix exponential does not operate on just y_n and terms can be sums of vectors. In general, such a method takes the form of Equation (18), where s is the number of intermediate stages.

$$
\begin{aligned}
x_1 &= y_n \\
x_{i+1} &= \sum_{j=1}^{i} d_{ij} e^{h \sum_{k=1}^{i} a_{ijk} F(x_k, t_n + c_k)} x_j \\
y_{n+1} &= x_{s+1}
\end{aligned} \tag{18}
$$

To calculate the correct values of the a, c and d coefficients, we need the consistency conditions and order conditions. For consistency, we just need to ensure that $\sum_j d_{ij} = 1$ for all i so that a constant F is treated correctly. An example set of order conditions are shown in Appendix A for $y - y_{\text{approx}}$ in a two-stage EL method. If we set all coefficients up to order n to zero, then the resulting method is order n. We note that the final equation for $\mathbf{f}_y{}'(y; \mathbf{f}_y{}'(y, \mathbf{f}_y y))$ cannot be set to zero in this example, so a third order method cannot be derived with a two-stage EL method. The maximum possible order of a method matches the number of stages, at least up to order 4.

The result is that there are more unknowns than constraint equations. This gives us flexibility. One use of this flexibility is to try to minimize as many order $n + 1$ terms as possible, to improve accuracy. In our case, we summed the squares of each $n + 1$ coefficient and minimized that function. The other use of the flexibility is to add a few more constraints that improve the stability of the method. These constraints are described in the next section.

5.1. Coefficient Considerations for Stability

In the ideal case, all coefficients a and d are non-negative. This ensures three things: the sums of reaction rates are non-negative, the decay constants are non-negative, and the vector sums are non-negative. During the search for a third and fourth order method, no method was found in which all terms were non-negative. As such, some level of compromise had to be made.

A negative reaction rate can occur if some a have negative values. To demonstrate why this is an issue, take the following simple ODE system:

$$
\begin{aligned}
y_1' &= -c y_1 \\
y_2' &= c y_1
\end{aligned}
$$

If c is negative, then y_1 is growing exponentially. y_2 is decreasing at a rate not proportional to its value. If integrated over too long a time span, then it becomes possible for y_2 to go negative. During testing, however, none of the algorithms used demonstrated this issue. As such, this constraint was not enforced during the derivation.

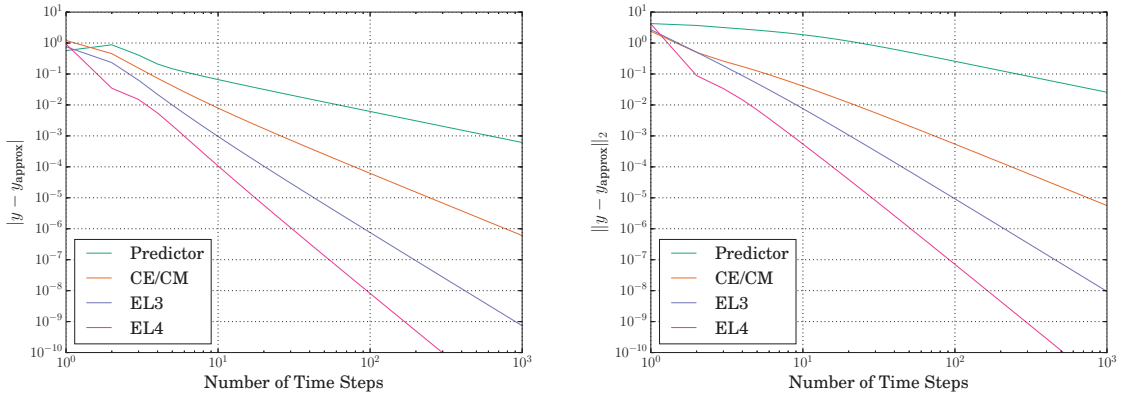
If the sum of the a coefficients used during a summation are negative, then it becomes possible for the decay constants themselves to go negative. This is a much worse circumstance, as the largest eigenvalues in the decay matrix are typically due to the decay constants. As mentioned in Section 2.3, if positive eigenvalues are too large, then CRAM will fail to calculate the correct matrix exponential. As such, the constraint listed in Equation (19) was always used.

$$
\sum_k a_{ijk} \geq 0 \tag{19}
$$

Finally, if any d is allowed to go negative, then the sum of positive vectors can become negative. As the vectors fluctuate over many orders of magnitude during depletion, even small negative d values can form negative x_i components. Where possible, then, the constraint in Equation (20) was used.

$$
d_{ij} \geq 0 \tag{20}
$$

There is a way around this issue if the constraint cannot be met. During vector summation, the vector can be clipped so that it can never go below a small, nearly infinitely dilute concentration.



(a) Single ODE test

(b) System of ODEs test

Figure 4: The error of the EL3 and EL4 algorithms as a function of time step count

For the third order method, a solution was found in which both of the constraints in Equation (19) and Equation (20) held. In the fourth order case, a method could be found in which Equation (19) held, but not Equation (20). For this method, the clipping workaround was used, with the cutoff set at 10 atoms per cell.

5.2. Solving for Coefficients

The end result is that we have a constrained minimization problem. We are minimizing the sum of the squares of the order $n + 1$ coefficients, with the constraint that the order n and under coefficients must be zero. Additionally, we have the constraints in Equation (19) and, when possible, Equation (20). Unfortunately, the resulting equations are so complex for anything beyond a two term second order method that non-numeric approaches are infeasible. As such, the coefficients are computed numerically. Specifically, the `NMinimize` function in Mathematica [13] was used to compute the coefficients.

The optimization process `NMiminize` uses has a statistical component to it, so several random sequences were run and the most optimal (by objective function value) was selected. A three stage, third order method and a four stage, fourth order method were derived, with properties summarized in Table 3. The resulting coefficients are listed in Appendix B. The ODE tests from earlier was run again with these new methods. The result for the single ODE is shown in Figure 4a, and the result for the system shown in Figure 4b. These methods converge at the anticipated order for both.

Method	Order	Stages	# F stored
EL3	3	3	3
EL4	4	4	4

Table 3: Exponential-linear method properties

6. Testing Methodology

In order to test a depletion problem, several components will need to be developed. The first is the depletion tool itself, along with its corresponding depletion chain. The second is the test geometry. The third is the reference solution. Each component will be discussed in the next few sections.

6.1. Depletion Tool and Depletion Chain

The depletion tool used is a wrapper for the OpenMC Python API [14]. This tool generates a geometry, runs an OpenMC simulation, extracts tallies, and carries out the actual depletion algorithm. Each simulation was run with the parameters listed in Table 4. As the number of neutrons is quite low, each simulation is computed many times with different random sequences and the average used. The reference solution was computed 50 times each, and all others were computed 75 times each. The discrepancy is due to the computational cost of the reference solution. This allows for analysis of the distribution of results as well.

Parameter	Value
Inactive Batches	40
Active Batches	60
Neutrons Per Batch	5000

Table 4: Simulation parameters

Matrix exponentiation is performed with a 48-term CRAM method, so as to minimize the possibility that exponentiation errors contribute to the results [2].

The decay chain utilized the ENSDF data [15], in which all nuclides with $Z < 92$ were included. All those above 92 are listed in Table 5. There were 3396 nuclides in total. The rationale behind including so many nuclides was to ensure that the span of eigenvalues of the problem was as wide as possible. Neutron capture branching ratios were not included, as the goal was merely to make a challenging problem so as to compare integrators.

Element	Isotope List
Uranium	234, 235, 236, 237, 238
Neptunium	237, 238
Plutonium	238, 239, 240, 241, 242

Table 5: Nuclides, $Z > 91$ in Decay Chain

As mentioned in Section 5.1, the 4th order exponential-linear method can possibly generate negative nuclide concentrations. For that algorithm only, nuclide concentrations are clipped during vector summation so that a nuclide cannot have fewer than 10 atoms in any cell.

6.2. Test Geometry

The test geometry is a simple 2D geometry containing 4 pins with reflective boundary conditions. All pins are 4.5 % enriched UO_2. One pin is segmented into five rings and additionally contains 2% ^{157}Gd by weight. A diagram of the geometry is shown in Figure 5. This geometry was chosen to exhibit the strong spatial self-shielding gadolinium has, which is particularly difficult to correctly deplete [16].

6.3. Time Steps Chosen

In order to generate a reference solution, the CE/CM algorithm was used with a time step of 6 hours. The end point of the simulation was chosen as the point in which the eigenvalue reaches maximum, at 108 days. This corresponds to when the gadolinium concentration in the center ring of the discretized pin is dropping, which is when the relative error is near its maximum. A plot of the relative error in ^{157}Gd concentration in the center ring between the 6 hour time step reference solution and a 3 day time step is shown in Figure 6. Both solutions were calculated using CE/CM. This figure indicates a steady rise in error until 108 days, at which point it begins to level off.

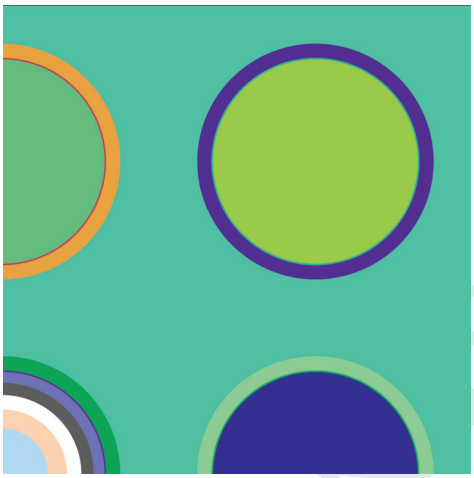


Figure 5: The geometry of the test problem, as plotted by OpenMC

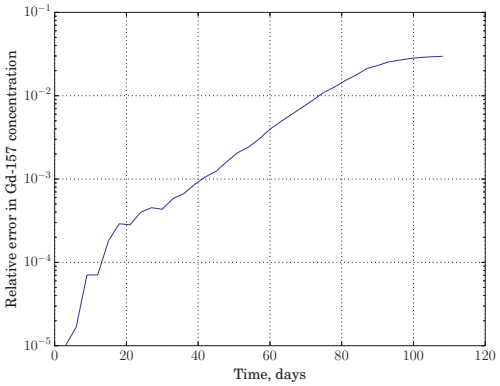


Figure 6: Relative error, Gd-157, between reference and 3 day time step, CE/CM

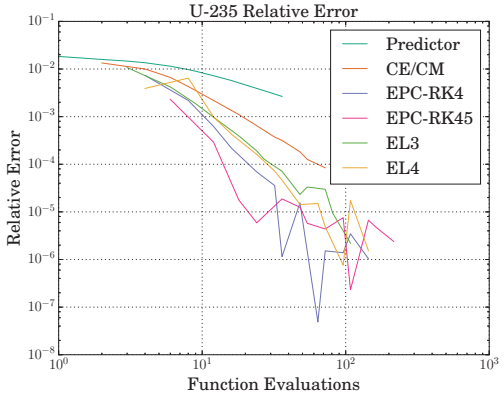


Figure 7: ^{235}U convergence vs. the total number of function evaluations used during the integration

7. Results

There are two topics of interest when it comes to depletion algorithms. The obvious one is whether or not a new method can achieve superior accuracy using the same number of function evaluations as the old method. The second one is with regards to how statistical uncertainty propagates throughout the time integrator. The first one is important for both deterministic and stochastic algorithms. The second is important for Monte Carlo.

7.1. Convergence

For this part, six algorithms were tested. First, from current methods, the predictor and CE/CM methods were tested. From the improved predictor-corrector methods, the EPC-RK4 and EPC-RK45 methods were tested. Finally, from the exponential-linear methods, the EL3 and EL4 methods were tested.

Each algorithm was used to integrate to 108 days, with a number of time steps from 1 to 36. The relative error compared to the reference solution was calculated for each algorithm and time step combination for the center cell of the segmented fuel pin.

For ^{235}U, the results are plotted in Figure 7. There are quite a number of interesting results in this figure. First, most of the algorithms begin to run into convergence issues by $\epsilon = 3 \times 10^{-5}$. As will be shown in Section 7.2, this is most likely due to statistics. The second is that the extended predictor-corrector algorithms appear to converge much better than 2nd order. As these methods have very small truncation coefficients below the scalar order of the method, this indicates that high order terms dominate in this region. If statistics were not an issue, these methods should eventually begin converging second order for extremely fine time steps.

Next, the exponential-linear methods converge roughly at the predicted order. However, the EL4 algorithm shows a much less uniform convergence. It is not clear why, but it is perhaps due to the stabilization added to ensure that nuclide concentrations cannot go negative. Disabling this feature to do a direct test would require a Monte Carlo solver that can support negative probability, a feature OpenMC does not currently have.

The exact same graph was also made for ^{157}Gd. This is shown in Figure 8. The results are much the same as that for ^{235}U, except that the EL4 algorithm converges a bit more smoothly.

Overall, the EPC-RK45 method has the best accuracy per unit cost in this regime. By the time EPC-RK45 is statistically limited, it is two orders of magnitude more accurate than CE/CM for a fixed number of function evaluations. When examined from a fixed error point of view, EPC-RK45 costs at most half as many function evaluations as CE/CM. This ratio improves as higher accuracy is desired. The exponential-linear methods should eventually overtake the EPC-RK45 method for short time steps if statistics are converged very tightly.

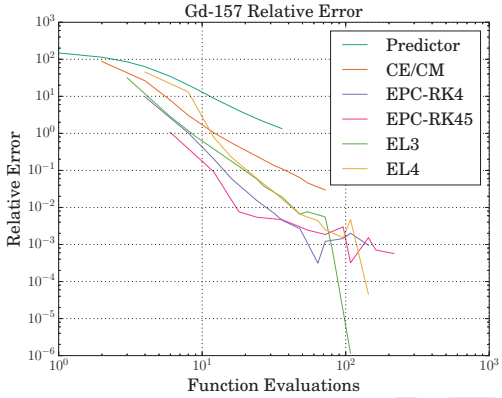


Figure 8: ^{157}Gd convergence vs. the total number of function evaluations used during the integration

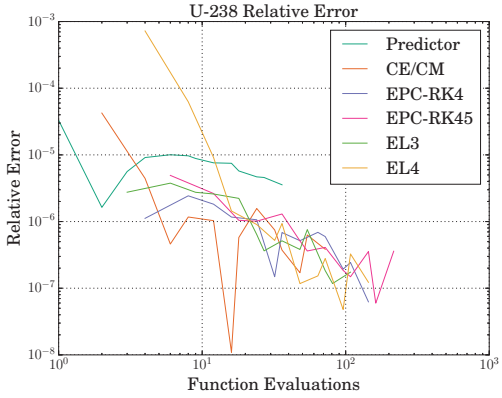


Figure 9: ^{238}U convergence vs. the total number of function evaluations used during the integration

Finally, the convergence of ^{238}U is plotted in Figure 9. In this particular case, ^{238}U is converged to within 0.1% for all algorithms and time steps. This will become useful later on during the statistical analysis, as it essentially eliminates convergence errors from affecting the uncertainty.

7.2. Statistics

As with anything involving Monte Carlo, it is very important to get a handle on the statistical properties of a method. As each method was run 75 times, it is possible to perform some statistical analysis on the distribution formed for each nuclide.

The standard deviation for ^{235}U, ^{157}Gd, and ^{238}U are shown in Figure 10-12 respectively. Most notably, for both ^{235}U and ^{238}U , convergence of the standard deviation is proportional to $1/\sqrt{N_f}$, N_f being the number of function evaluations. This is a useful fact, as more time steps not only improves the convergence of the algorithm, but it also improves the statistics. There is one point for EL4 which is significantly higher than the other values. This corresponds with an increase in error as seen in the previous section, and may also be caused by the stabilization required for the algorithm.

The ^{157}Gd plot is similar, except at small numbers of function evaluations. The likely reason for this is that the value the distribution represents is not temporally converged. The 1-step predictor solution has an error for ^{157}Gd of over two orders of magnitude. As such, an increase in standard deviation by an order of magnitude is not surprising.

In order to get a figure of merit for statistics between algorithms, we can fit Equation (21) to the data.

$$\sigma(N_f) \approx \frac{C}{\sqrt{N_f}} \tag{21}$$

^{238}U was used for the fitting, as this nuclide was highly converged for all runs. The ratio of the coefficient C to the C of predictor for ^{238}U is given in Table 6.

Method	$C_{\text{method}}/C_{\text{pred}}$
Predictor	1.000
CE/CM	1.422
EPC-RK4	1.005
EPC-RK45	1.268
EL3	1.041
EL4	1.177

Table 6: ^{238}U relative statistical performance

This indicates that the magnitude of the standard deviation is loosely sensitive to the method itself. As such, it is worthwhile to select an algorithm that not only minimizes the temporal truncation error, but also minimizes the statistical error propagation. Of the new methods tested, the predictor, EPC-RK4, and EL3 algorithms appear to have the best statistical properties.

8. Summary

In this paper, two new families of depletion algorithms, as well as a more rigorous way of testing high order methods was introduced.

The extended predictor-corrector methods (EPC-RK4 and EPC-RK45) are shown to be merely second order. However, the error coefficients for higher order terms are sufficiently small as to give substantial benefit in the moderate accuracy regime most nuclear analysis is performed in. The EPC-RK45 algorithm had the highest accuracy of all methods tested in this regime.

The exponential-linear methods (EL3 and EL4) are shown to be genuinely higher order. These methods test moderately well, beating the predictor and CE/CM algorithms. However, unless extreme accuracy is

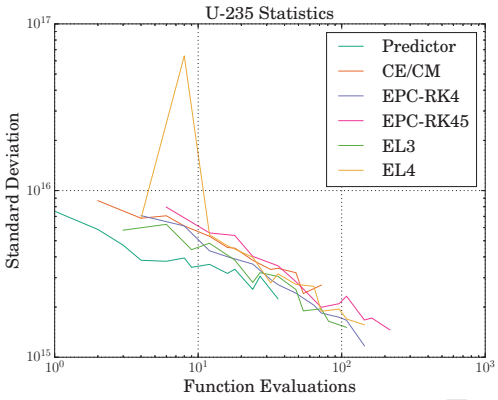


Figure 10: ^{235}U statistics

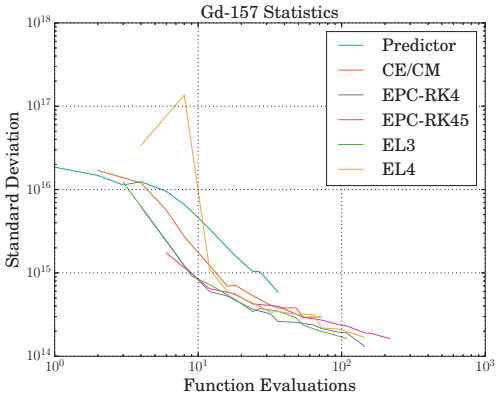


Figure 11: ^{157}Gd statistics

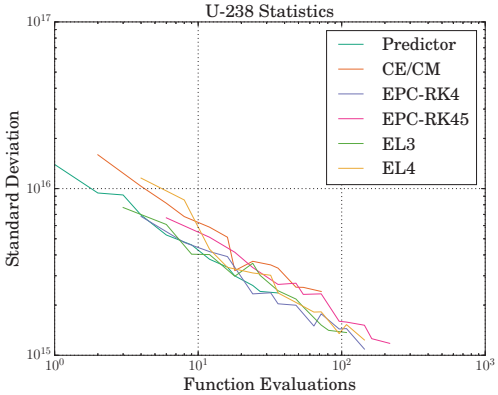


Figure 12: ^{238}U statistics

Method	Order	Stages	# F stored	Function Evals for Relerr: 10% ^{157}Gd	1% ^{157}Gd	Relative Std. Dev
Predictor	1	1	1	204*	949*	1.000
CE/CM	2	2	1	38	146*	1.422
EPC-RK4	2	4	4	14	28	1.004
EPC-RK45	2	6	5	12	17	1.267
EL3	3	3	3	20	45	1.041
EL4	4	4	4	21	49	1.177

Table 7: Overall comparison of methods (* extrapolated from last two datapoints)

desired, the extended predictor-corrector family has shown to be a cheaper alternative. Further, the EL4 algorithm has to be stabilized to ensure that the number density stays positive throughout the simulation.

The negative cost of using these new methods is the increased number of decay matrices that must be stored. The EPC-RK45 method requires the summation of 5 matrices to get the final value of y. If memory is a limiting consideration, these methods will have less utility.

With regards to statistics, two interesting properties were shown. First, the standard deviation is proportional to $1/\sqrt{N_f}$, N_f being the number of function evaluations. As such, reducing the time step will decrease statistical error and temporal truncation error simultaneously.

Finally, each algorithm has a slightly different statistical performance. The predictor, EPC-RK4, and EL3 algorithms all yield approximately the same standard deviation for a fixed number of neutrons. The CE/CM, EPC-RK45, and EL4 algorithms yield slightly higher standard deviations. If statistical uncertainty is the chief worry, these algorithms should be avoided.

These results are summarized in Table 7, in which each algorithm can be compared by accuracy, memory usage, and statistical performance.

9. Acknowledgements

This research was supported in part by the Consortium for Advanced Simulation of Light Water Reactors (CASL), an Energy Innovation Hub for Modeling and Simulation of Nuclear Reactors under U.S. Department of Energy Contract No. DE-AC05-00OR22725. The first author is also partially supported under a Department of Energy Nuclear Energy University Programs Graduate Fellowship. This research made use of the resources of the High Performance Computing Center at Idaho National Laboratory, which is supported by the Office of Nuclear Energy of the U.S. Department of Energy under Contract No. DE-AC07-05ID14517.

[1] P. J. Turinsky, Handbook of Nuclear Engineering, Springer US, Boston, MA, 2010, Ch. Core Isotopic Depletion and Fuel Management, pp. 1241–1312. doi:10.1007/978-0-387-98149-9_10.
URL http://dx.doi.org/10.1007/978-0-387-98149-9_10

[2] M. Pusa, Higher-order chebyshev rational approximation method and application to burnup equations, Nuclear Science and Engineering 182 (3).

[3] C. Moler, C. Van Loan, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM review 45 (1) (2003) 3–49.

[4] A. Isotalo, V. Sahlberg, Comparison of neutronics-depletion coupling schemes for burnup calculations, Nuclear Science and Engineering 179 (4) (2015) 434–459.

[5] J. T. Goorley, M. R. James, T. E. Booth, F. Brown, J. Bull, L. J. Cox, J. Durkee, J. Elson, M. Fensin, R. Forster, et al., Initial mcnp6 release overview-mcnp6 version 1.0, Los Alamos National Laboratory, Los Alamos, NM, LA-UR-13-22934 1.

[6] J. Leppänen, Serpent–a continuous-energy monte carlo reactor physics burnup calculation code, VTT Technical Research Centre of Finland 4.

[7] E. Jones, T. Oliphant, P. Peterson, et al., SciPy: Open source scientific tools for Python, [Online; accessed 2016-05-04] (2001–).
URL http://www.scipy.org/

[8] A. H. Al-Mohy, N. J. Higham, A new scaling and squaring algorithm for the matrix exponential, SIAM Journal on Matrix Analysis and Applications 31 (3) (2009) 970–989.

[9] J. C. Butcher, A history of runge-kutta methods, Applied numerical mathematics 20 (3) (1996) 247–260.

[10] J. C. Butcher, Runge–Kutta Methods, John Wiley & Sons, Ltd, 2016, Ch. 3, pp. 143–331. doi:10.1002/9781119121534.ch3. URL http://dx.doi.org/10.1002/9781119121534.ch3
[11] J. Butcher, On fifth and sixth order explicit runge–kutta methods: order conditions and order barriers, Canadian Applied Mathematics Quarterly 17 (3) (2009) 433–445.
[12] J. R. Cash, A. H. Karp, A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides, ACM Transactions on Mathematical Software (TOMS) 16 (3) (1990) 201–222.
[13] Wolfram Research, Inc., Mathematica 10.4. URL https://www.wolfram.com
[14] P. K. Romano, B. Forget, The openmc monte carlo particle transport code, Annals of Nuclear Energy 51 (2013) 274–281.
[15] N. ENSDF, online data service, ensdf database (2015).
[16] D. Lee, J. Rhodes, K. Smith, Quadratic depletion method for gadolinium isotopes in casmo-5, Nuclear Science and Engineering 174 (1) (2013) 79–86.

A. Formal Expansion of a Two-Stage Exponential Linear Method

Term	Order	Coefficient
y	0	$-1 + d_{21} + d_{11}d_{22}$
$\mathbf{f}_y y$	1	$-1 + a_{211}d_{21} + a_{212}d_{11}d_{21} + d_{11}(a_{111} + a_{221} + a_{222}d_{11})d_{22}$
$\mathbf{f}_y\mathbf{f}_y y$	2	$-1 + (a_{211} + a_{212}d_{11})^2 d_{21} + d_{11}(a_{111} + a_{221} + a_{222}d_{11})^2 d_{22}$
$\mathbf{f}_y{}'(y; \mathbf{f}_y y)$	2	$-1 + 2a_{111}d_{11}^2(a_{212}d_{21} + a_{222}d_{11}d_{22})$
$\mathbf{f}_y\mathbf{f}_y\mathbf{f}_y y$	3	$-1 + (a_{211} + a_{212}d_{11})^3 d_{21} + d_{11}(a_{111} + a_{221} + a_{222}d_{11})^3 d_{22}$
$\mathbf{f}_y(\mathbf{f}_y{}'(y; \mathbf{f}_y y))$	3	$-1 + 3a_{111}d_{11}^2(a_{212}(a_{211} + a_{212}d_{11})d_{21} + a_{222}d_{11}(a_{221} + a_{222}d_{11})d_{22})$
$\mathbf{f}_y{}''(y; \mathbf{f}_y y, \mathbf{f}_y y)$	3	$-1 + 3a_{111}^2 d_{11}^3(a_{212}d_{21} + a_{222}d_{11}d_{22})$
$\mathbf{f}_y{}'(y'; \mathbf{f}_y\mathbf{f}_y y)$	3	$-1 + 3a_{111}^2 d_{11}^2(a_{212}d_{21} + a_{222}d_{11}d_{22})$
$\mathbf{f}_y{}'(\mathbf{f}_y y; \mathbf{f}_y y)$	3	$-2 + 3a_{111}d_{11}^2(a_{212}(a_{211} + a_{212}d_{11})d_{21} + a_{222}d_{11}(2a_{111} + a_{221} + a_{222}d_{11})d_{22})$
$\mathbf{f}_y{}'(y; \mathbf{f}_y{}'(y; \mathbf{f}_y y))$	3	-1

Table 8: Formal series coefficients for $y - y_{\text{approx}}$ for a 2-stage exponential linear method to third order

B. Exponential-Linear Coefficients

B.1. EL3

Coeff.	Value
c_1	0.0
d_{11}	1.0
a_{111}	$4.546\,892\,904\,137\,023\,0 \times 10^{-1}$
c_2	$4.546\,892\,904\,137\,023\,0 \times 10^{-1}$
d_{21}	$4.917\,209\,126\,428\,904\,7 \times 10^{-1}$
a_{211}	$-9.357\,880\,632\,412\,118\,3 \times 10^{-2}$
a_{212}	$8.796\,663\,817\,251\,793\,8 \times 10^{-1}$
d_{22}	$5.082\,790\,873\,571\,095\,3 \times 10^{-1}$
a_{221}	$-5.901\,222\,142\,248\,917\,6 \times 10^{-1}$
a_{222}	$9.215\,207\,140\,261\,931\,5 \times 10^{-1}$
c_3	1.0
d_{31}	$2.037\,857\,322\,055\,807\,3 \times 10^{-2}$
a_{311}	$2.323\,856\,318\,306\,070\,0 \times 10^{-1}$
a_{312}	$1.815\,985\,521\,375\,668\,1 \times 10^{-1}$
a_{313}	$5.860\,142\,159\,064\,473\,0 \times 10^{-1}$
d_{32}	$5.023\,605\,076\,944\,110\,8 \times 10^{-1}$
a_{321}	$1.105\,777\,934\,011\,147\,9 \times 10^{-2}$
a_{322}	$2.782\,279\,660\,329\,436\,3 \times 10^{-2}$
a_{323}	$5.064\,301\,564\,868\,396\,1 \times 10^{-1}$
d_{33}	$4.772\,609\,190\,850\,308\,4 \times 10^{-1}$
a_{331}	$2.721\,242\,491\,737\,410\,7 \times 10^{-2}$
a_{332}	$-1.076\,902\,283\,649\,226\,7 \times 10^{-1}$
a_{333}	$2.943\,901\,631\,394\,099\,0 \times 10^{-1}$

Table 9: Coefficients for the exponential-linear 3rd order method EL3

Coeff.	Value
c_1	0.0
d_{11}	1.0
a_{111}	$2.638\,017\,781\,099\,526\,4 \times 10^{-1}$
c_2	$2.638\,017\,781\,099\,526\,4 \times 10^{-1}$
d_{21}	$4.714\,899\,766\,145\,780\,3 \times 10^{-1}$
a_{211}	$-1.096\,345\,914\,231\,227\,6 \times 10^{-1}$
a_{212}	$7.549\,479\,388\,690\,35 \times 10^{-1}$
d_{22}	$5.285\,100\,233\,854\,22 \times 10^{-1}$
a_{221}	$-8.139\,969\,413\,877\,527 \times 10^{-1}$
a_{222}	$1.195\,508\,497\,529\,188\,3$
c_3	$6.453\,133\,474\,459\,122\,4 \times 10^{-1}$
d_{31}	$2.333\,112\,759\,614\,89 \times 10^{-1}$
a_{311}	$2.432\,927\,685\,490\,108$
a_{312}	$-1.886\,991\,744\,360\,153\,8$
a_{313}	$4.540\,639\,985\,471\,296 \times 10^{-1}$
d_{32}	$5.526\,116\,522\,082\,521 \times 10^{-1}$
a_{321}	$1.440\,240\,011\,283\,619\,1$
a_{322}	$-1.999\,581\,093\,585\,001\,1$
a_{323}	$1.295\,539\,340\,166\,664$
d_{33}	$2.140\,770\,718\,302\,588\,4 \times 10^{-1}$
a_{331}	$-3.341\,457\,198\,009\,325\,5 \times 10^{-1}$
a_{332}	$-1.551\,927\,277\,833\,745$
a_{333}	$2.240\,759\,630\,039\,589$
c_4	1.0
d_{41}	$-2.540\,101\,046\,715\,893\,8 \times 10^{-2}$
a_{411}	$6.342\,361\,480\,700\,457 \times 10^{-1}$
a_{412}	$-1.426\,165\,912\,825\,637\,6$
a_{413}	$-7.209\,962\,986\,478\,266 \times 10^{-1}$
a_{414}	$2.512\,926\,068\,677\,481$
d_{42}	$2.913\,365\,964\,654\,815\,5 \times 10^{-1}$
a_{421}	$5.602\,130\,520\,260\,26 \times 10^{-1}$
a_{422}	$-1.036\,247\,635\,307\,391\,7$
a_{423}	$1.403\,357\,266\,739\,732\,5$
a_{424}	$-1.911\,244\,663\,312\,152\,1 \times 10^{-1}$
d_{43}	$6.387\,934\,650\,493\,379 \times 10^{-1}$
a_{431}	$1.138\,564\,243\,974\,421\,3 \times 10^{-1}$
a_{432}	$1.137\,278\,934\,630\,576\,9 \times 10^{-1}$
a_{433}	$-3.355\,485\,694\,559\,844\,4 \times 10^{-1}$
a_{434}	$4.626\,509\,125\,349\,493\,3 \times 10^{-1}$
d_{44}	$9.527\,094\,895\,233\,958 \times 10^{-2}$
a_{441}	$-1.138\,311\,740\,251\,085$
a_{442}	$4.998\,539\,153\,859\,359\,3 \times 10^{-1}$
a_{443}	$1.196\,593\,771\,894\,506\,6$
a_{444}	$-5.581\,359\,405\,254\,164 \times 10^{-1}$

Table 10: Coefficients for the exponential-linear 4th order method EL4