PUBLICATION IV

Solving linear systems with sparse Gaussian elimination in the Chebyshev Rational Approximation Method (CRAM)

SOLVING LINEAR SYSTEMS WITH SPARSE GAUSSIAN ELIMINATION IN THE CHEBYSHEV RATIONAL APPROXIMATION METHOD (CRAM)

MARIA PUSA AND JAAKKO LEPPÄNEN

ABSTRACT. The Chebyshev Rational Approximation Method (CRAM) has been recently introduced by the authors for solving the burnup equations with excellent results. This method has been shown to be capable of simultaneously solving an entire burnup system with thousands of nuclides both accurately and efficiently. The method was prompted by an analysis of the spectral properties of burnup matrices, and it can be characterized as the best rational approximation on the negative real axis. The coefficients of the rational approximation are fixed and have been reported for various approximation orders. In addition to these coefficients, implementing the method only requires a linear solver. This paper describes an efficient method for solving the linear systems associated with the CRAM approximation. The introduced direct method is based on sparse Gaussian elimination, where the sparsity pattern of the resulting upper triangular matrix is determined before the numerical elimination phase. The stability of the proposed Gaussian elimination method is discussed based on considering the numerical properties of burnup matrices. Suitable algorithms are presented for computing the symbolic factorization and numerical elimination in order to facilitate the implementation of CRAM and its adoption into routine use. The accuracy and efficiency of the described technique are demonstrated by computing the CRAM approximations for a large test case with over 1600 nuclides.

1. INTRODUCTION

The changes in material compositions of a reactor fuel must be taken into account in all reactor physics calculations, which is in practice handled by burnup calculation codes. An essential part of a burnup calculation is the solving of the burnup equations that describe the rates by which the concentrations of the various nuclides change. The burnup equations form a system of first-order linear differential equations that can be written in matrix notation as

$$\boldsymbol{n}' = \boldsymbol{A}\boldsymbol{n} \ , \quad \boldsymbol{n}(0) = \boldsymbol{n}_0 \ , \tag{1}$$

where $\boldsymbol{n}(t) \in \mathbb{R}^n$ is the nuclide concentration vector and $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is the burnup matrix containing the decay and transmutation coefficients of the nuclides under consideration. The matrix elements A_{ij} characterize the rates of neutron-induced reactions and spontaneous radioactive decay by which nuclide j is transformed to nuclide i, and they are assumed to be fixed constants.

The burnup equations can be formally solved by the matrix exponential method yielding the simple solution

$$\boldsymbol{n}(t) = e^{\boldsymbol{A}t}\boldsymbol{n}_0 \ , \tag{2}$$

Accepted for publication in the November 2013 issue of *Nuclear Science and Engineering*.

where the exponential of the matrix $\boldsymbol{A}t$ is defined as the power series expression

$$e^{\boldsymbol{A}t} = \sum_{k=0}^{\infty} \frac{1}{k!} \left(\boldsymbol{A}t\right)^k \ , \tag{3}$$

with the additional definition $\boldsymbol{A}^0 = \boldsymbol{I}$.

There are numerous algorithms for computing the matrix exponential, but because of the numerical properties of burnup matrices, most of them are ill-suited for solving the burnup equations. The decay constants of the nuclides vary extensively, which induces elements with both extremely small and large absolute values to the burnup matrix. Short-lived nuclides—corresponding to elements with the largest magnitude—are especially problematic since they increase the matrix norm and induce eigenvalues with absolute values up to the order of 10^{21}. Furthermore, the time steps used in burnup calculations can typically vary from a few days (10^5 seconds) to several months (10^7 seconds), and to even thousands of years, if only decay reactions are considered. Most of the established matrix exponential methods, such as the truncated Taylor series approach or the rational Padé approximation, are based on approximation near the origin and work well only when the matrix norm $\|\boldsymbol{A}t\|$ is sufficiently small. Consequently, these algorithms are prone to severe numerical problems when applied to the burnup equations, where this norm can be of the order of 10^{28} [1].

These difficulties have traditionally been solved by using simplified burnup chains or by treating the most short-lived nuclides separately when computing a matrix exponential solution. However, it was recently discovered by the authors that the eigenvalues of the burnup matrix are generally confined to a region near the negative real axis [1]. This observation combined with the fact that in the Chebyshev Rational Approximation Method (CRAM) the rational function $r(z)$ is chosen as the best rational approximation of the exponential function on the negative real axis $\mathbb{R}_-$, led to applying CRAM to solving the burnup equations. The rational approximation in CRAM can be formally defined as the unique rational function $\hat{r}_{k,k} = \hat{p}_k(x)/\hat{q}_k(x)$ satisfying

$$\sup_{x\in\mathbb{R}_-} |\hat{r}_{k,k}(x) - e^x| = \inf_{r_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}_-} |r_{k,k}(x) - e^x| \right\} \ , \tag{4}$$

where $\pi_{k,k}$ denotes the set of rational functions $r_{k,k}(x) = p_k(x)/q_k(x)$, where p_k and q_k are polynomials of order k. CRAM has been shown to give a robust and accurate solution to the burnup equations with a very short computation time [1, 2, 3].

For numerical reasons, it is generally advantageous to compute the matrix rational function in the partial fraction decomposition form. For a rational function $r_{k,k}(x)$ with simple poles, the decomposition takes the form

$$r_{k,k}(z) = \alpha_0 + \sum_{j=1}^{k} \frac{\alpha_j}{z - \theta_j} \ , \tag{5}$$

where α_0 is the limit of the function $r_{k,k}$ at infinity and α_j are the residues at the poles θ_j. The poles of a rational function with real-valued coefficients form conjugate pairs, so the computational cost can be reduced to half for a real variable x:

$$r_{k,k}(x) = \alpha_0 + 2\,\mathrm{Re}\left(\sum_{j=1}^{k/2} \frac{\alpha_j}{x-\theta_j}\right) \; . \tag{6}$$

The rational approximation to Eq. (2) can then be written

$$\boldsymbol{n} = \alpha_0 \boldsymbol{n}_0 + 2\,\mathrm{Re}\left(\sum_{j=1}^{k/2} \alpha_j (\boldsymbol{A}t - \theta_j \boldsymbol{I})^{-1} \boldsymbol{n}_0\right) \; . \tag{7}$$

Therefore, the computation of a CRAM approximation of order k requires solving $k/2$ linear systems of the form

$$(\boldsymbol{A}t - \theta_j \boldsymbol{I})\, \boldsymbol{x}_j = \alpha_j\, \boldsymbol{n}_0 \; . \tag{8}$$

Notice that the partial fraction coefficients for each CRAM approximation order are fixed. Therefore, implementing CRAM only requires a linear solver in addition to these coefficients. The approximation orders $1 \leq k \leq 13$ can be computed with high accuracy by using the approximative Carathéodory–Fejér method. A MATLAB script is provided for this purpose in [4]. In addition, revised sets of partial fraction coefficients for approximation orders $k = 14$ and $k = 16$ have been recently computed and reported [3, 5]. The development version of Serpent 2 features CRAM approximation of orders 6, 8, 10, 12, 14 and 16.

When no nuclides are excluded from the computation, the linear systems are large—the dimensions of the burnup matrix being in the order of a thousand—which complicates the computation of the CRAM approximation. Also, the numerical characteristics of burnup matrices may compromise the accuracy of widely used iterative solvers. The objective of this paper is to describe a method that can be used to solve these linear systems in order to compute the CRAM approximation in a burnup code. The proposed method is based on sparse Gaussian elimination and it has been implemented to the reactor physics code Serpent [6]. This paper is organized as follows. In Section 2, we briefly discuss the numerical properties of burnup matrices that are essential when selecting a numerical method for solving the linear systems according to Eq. (8). In Section 3 we review the theoretical background for sparse Gaussian elimination and introduce suitable algorithms for implementing the method. Finally, numerical results are presented in Section 4 and the accuracy and efficiency of the proposed method is demonstrated.

2. Numerical Properties of Burnup Matrices

In order to select a well-suited method for solving the linear systems of Eq. (8), it is necessary to consider the numerical characteristics of burnup matrices. First of all, they are both large and sparse. Depending on the nuclear data libraries used in the computation, the number of nuclides is generally between 1200 and 1600, if no nuclides are excluded from the computation. The density of the burnup matrix is a few percent, which corresponds to some tens of thousands of non-zero elements in a single matrix. The nuclides can naturally be indexed arbitrarily, which can be exploited when constructing the matrix. Fortunately, the burnup matrix becomes nearly upper triangular if the nuclides are indexed in ascending order with respect to their ZAI index, defined as $\mathrm{ZAI} = 10\,000Z + 10A + I$, where Z is the atomic number, A is the mass number and I is the isomeric state number. Due to the size of the problem, the method used to solve systems of Eq. (8) should either be iterative or exploit the sparsity pattern of the matrices.

The diagonal elements of a burnup matrix are non-positive and the element a_{jj} characterizes the total rate by which nuclide j is transformed to other nuclides. The off-diagonal elements, on the other hand, are non-negative and the element a_{ij} describes the rate by which nuclide j is transformed to nuclide i. When forming the burnup equations, it is possible to take into account the production of by-product nuclides. In this case, for example, the reaction rate for each (n,p) reaction contributes to the production rate of $^1\mathrm{H}$. Traditionally, the production of nuclides as by-products has been ignored in the burnup matrix [7, 8]. Therefore, we will use the term *augmented burnup matrix* to refer to the case, where the production of by-product nuclides has been taken into account when constructing the burnup matrix. In the development version of Serpent 2, the augmented burnup equations are solved by default.

The augmented burnup matrix elements satisfy the following relation:

$$-d_j\, a_{jj} = \sum_{i\neq j} a_{ij} \ , \quad j = 1,\ldots,n \ , \tag{9}$$

where d_j is the *average* number of nuclides produced in the transmutation and decay reactions of nuclide j, consisting of the daughter nuclide and the reaction by-products. Since at most four nuclides can result from a single transmutation or decay reaction, e.g., from the $(\mathrm{n},3\alpha)$ reaction, it holds $1 \leq d_j \leq 4$. Apart from fission, the reactions producing two or more nuclides are relatively improbable, however, and hence it generally holds $1 \leq d_j \leq 2$.

The half-lives of nuclides vary significantly, and as a result, the absolute values of burnup matrix elements can range from zero even up to the order of 10^{21}. As mentioned in Section 1, the extensive variations in the half-lives induce eigenvalues with extremely small and large magnitudes. The respective eigensystem is generally poorly conditioned with many nearly confluent eigenvalues. Many of the popular iterative methods, such as biconjugate gradient and generalized minimum residual method, are based on Krylov subspace techniques and their convergence is ultimately related to the spectral properties of the matrix at hand. In this context, it is particularly disadvantageous, if the matrix eigenvalues lie far apart from each other, which is the case for burnup matrices. Therefore, the numerical characteristics of burnup matrices may cause both convergence and round-off problems in these algorithms. For these reasons it was decided to implement a direct method to Serpent.

3. Sparse Gaussian Elimination

In order to avoid introducing any additional error to the matrix exponential solution, it would be favorable to solve the systems of Eq. (8) with a direct method instead of an iterative one. The sparsity pattern of burnup matrices makes Gaussian elimination a prominent candidate for this task and therefore the application of this method is considered in detail in this section.

As is well-known, the main complication in basic Gaussian elimination is related to round-off errors. It is easy to demonstrate that in the presence of finite precision arithmetics, this method may lead to totally erroneous results due to its instability. In addition, the triangular matrices (i.e., LU decomposition) generated during the process may be more ill-conditioned than the original matrix, which can in turn grow the round-off error in the forward and back substitution phases of the solution. This may be demonstrated by the following classical example matrix [9]

$$\boldsymbol{A} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 1 \end{bmatrix}, \tag{10}$$

which has the LU decomposition

$$\boldsymbol{L} = \begin{bmatrix} 1 & 0 \\ 10^{20} & 1 \end{bmatrix}, \quad \boldsymbol{U} = \begin{bmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{bmatrix}. \tag{11}$$

In floating point arithmetics the number $u_{22} = 1 - 10^{20}$ will be rounded off to roughly $\tilde{u}_{22} = -10^{20}$. Consequently, for respective floating point matrices $\widetilde{\boldsymbol{L}}$ and $\widetilde{\boldsymbol{U}}$ produced by the algorithm, we get the following result

$$\widetilde{\boldsymbol{L}}\widetilde{\boldsymbol{U}} = \begin{bmatrix} 10^{-20} & 1 \\ 1 & 0 \end{bmatrix}, \tag{12}$$

which is substantially different from $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$. Also, if we solve the system $\widetilde{\boldsymbol{L}}\widetilde{\boldsymbol{U}}\boldsymbol{x} = \boldsymbol{b}$, the solution will differ drastically from the true solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

The instability demonstrated in the previous example arises when the intermediate quantities computed during the algorithm are large relative to the elements of $\boldsymbol{A}$. In order to control this instability, Gaussian elimination is nearly always implemented with *partial pivoting.* In partial pivoting, the rows of the matrix are permuted in such manner that in each column the largest element of the lower triangular part of the matrix is chosen as the pivot element. This guarantees that the elements of the lower triangular matrix $\boldsymbol{L}$ fulfill $|L_{ij}| \le 1$. In this case the algorithm is stable if the elements of $\boldsymbol{U}$ are of the same order as the elements of $\boldsymbol{A}$. This stability can be monitored based on a quantity called the *growth factor*, defined as

$$\rho = \frac{\max_{i,j} |u_{ij}|}{\max_{i,j} |a_{ij}|}. \tag{13}$$

When ρ is of order 1, we can expect Gaussian elimination with partial pivoting to be stable. Gaussian elimination with partial pivoting is considered to be extremely stable in practice, although it is possible to construct matrices with large growth factors. When using partial pivoting, the elements of $\boldsymbol{U}$ can become large (compared to the elements of $\boldsymbol{A}$) only through the subtraction of rows in the Gaussian elimination. Therefore, it is relatively easy to show that the maximum value for the growth factor is $\rho_{\max} = 2^{n-1}$. In practice, however, for the growth factor to attain the value $\rho_{\max}$ would require very special matrices that never seem to arise in real applications [9].

Therefore, Gaussian elimination with partial pivoting is considered to be a reliable method for solving linear systems. There are, however, some difficulties in applying it efficiently to sparse systems. First of all, it would be sensible to utilize the sparsity pattern of the matrix under consideration and perform the elimination solely on the non-zero elements of the matrix. This requires employing a sparse matrix format where only the non-zero entries are stored together with their row and column information. During the Gaussian elimination phase, however, new non-zero elements are introduced in the matrix. Unfortunately, putting on new non-zero elements to the sparse matrix structure during the elimination may easily begin to dominate the computational cost. For this reason, an efficient implementation of a sparse Gaussian elimination requires that the non-zero structure of the resulting decomposition is determined in advance. This allows allocating storage for all non-zero entries before beginning the numerical elimination. Determining this sparsity pattern is called computing the *symbolic LU factorization* and the matrix containing the non-zero structure is called the fill-in matrix. It is noteworthy that when computing a CRAM approximation of order k, this sparsity pattern is similar for all $k/2$ matrix equations of Eq. (8) and it is therefore sufficient to compute the symbolic factorization only once for each CRAM approximation.

Unfortunately, this strategy fails if we wish to implement Gaussian elimination with partial pivoting. In this case the sparsity pattern of $\boldsymbol{U}$ depends on the employed permutations, which are determined during the numerical elimination. Hence, partial pivoting inhibits computing the symbolic LU factorization before the numerical elimination phase, which deteriorates the computational efficiency of this approach. Luckily, the special characteristics of burnup matrices actually allow using Gaussian elimination without partial pivoting. This is explained in Section 3.2

3.1. **Symbolic factorization.** This section describes an algorithm that can be used to compute the sparsity pattern of the fill-in matrix $\boldsymbol{F}$ resulting from the Gaussian elimination. The method presented here is essentially the algorithm FILL2 from [10] but we attempt to introduce it without the unnecessary use of graph-theoretical notation in order to facilitate its comprehensibility and implementation.

The column and row indices of $\boldsymbol{A}$ are referred to as nodes. When $a_{ij} \neq 0$, the notation $i \to j$ is used. We say that there is a *path* of length m from node i to node k, if there exists a sequence of non-zero nodes $[i = i_1, i_2, i_3 \ldots, i_m, i_{m+1} = k]$ such that $i_n \to i_{n+1}$ for $n = 1, \ldots, m+1$. The physical interpretation for this is that there exists a transmutation path of length m from nuclide k to nuclide i.

The sparsity pattern of the matrix $\boldsymbol{F}$ can be computed efficiently by studying the various paths related to the graph of $\boldsymbol{A}$. This is a direct consequence from Lemma 1 in [10], which states that $F_{ij} \neq 0$ if and only if there exists a path from node i to node j such that

$$i_n < \min(i,j) \qquad \text{for} \quad 2 \leq n \leq m \; . \tag{14}$$

Algorithm 1 can be used to compute the non-zero structure of $\boldsymbol{F}$ based on Lemma 1 from [10]. As a first step, the sparsity pattern of $\boldsymbol{F}$ is initialized with the sparsity pattern of $\boldsymbol{A}$. The algorithm then proceeds column-wise through the matrix. In each column, the upper triangular part of $\boldsymbol{F}$ is studied. If a non-zero element F_{ij} with $i < j$ is found, node i is added to the set of nodes Ω_j to be considered later. After examining the non-zero pattern of column j, the algorithm continues by deleting nodes one by one from the set Ω_j. After deleting node i from Ω_j, the respective column of matrix $\boldsymbol{F}$ is examined. If a non-zero element a_{ki} with $k > i$ is found, there exists a path $k \to i \to j$ such that $i < \min(k, j)$, and according to Lemma 1, it follows that F_{kj} is non-zero in the fill-in matrix. In addition, if $i < k < j$, there is a prospect for a longer path to node j through nodes

Algorithm 1 Symbolic factorization

```
Initialize $\boldsymbol{F}$ with the sparsity pattern of $\boldsymbol{A}$
for $j = 2, \ldots, n$ do
  $\Omega_j = \emptyset$
  $\boldsymbol{a} = \boldsymbol{0}$
  for each non-zero $F_{ij}$ in column $j$ do
    $a_i = 1$
    if $i < j$ then
      add $i$ to $\Omega_j$
    end if
  end for
  while $\Omega_j \neq \emptyset$ do
    Delete some node $i$ from $\Omega_j$
    for each non-zero $F_{ki}$ in column $i$ do
      if $k > i$ and $a_k = 0$ then
        add $F_{kj}$ to the non-zero structure of $\boldsymbol{F}$
        $a_k = 1$
        if $k < j$ then
          add $k$ to $\Omega_j$
        end if
      end if
    end for
  end while
end for
```

i and k, and therefore node k is added to the set Ω_j. Notice that since the matrix $\boldsymbol{F}$ is studied column-wise starting from the second column, the non-zeros F_{kj} with $k < i$ and $k < j$ can be ignored at this stage, since the columns with indices smaller than j have already been taken into account. The calculation continues this way until the set Ω_j is empty. The auxiliary vector $\boldsymbol{a}$ is used to keep track of the non-zero elements that have already been added to the sparsity structure of $\boldsymbol{F}$ in each column.

It is straightforward to estimate the computational cost of Algorithm 1. The first step in the algorithm is proportional to the number of non-zeros in the column j of $\boldsymbol{F}$. Then, for each non-zero F_{ij} in the upper triangular part of column j, the lower triangular part of column i is studied. We'll use the notation $|\cdot|$ to denote the number of non-zero entries in a matrix or a vector, and write the fill-in matrix in the following form:

$$\boldsymbol{F} = \boldsymbol{F}^L + \boldsymbol{D} + \boldsymbol{F}^U \ , \tag{15}$$

where $\boldsymbol{F}^L$ contains the lower triangular part, $\boldsymbol{D}$ the diagonal, and $\boldsymbol{F}^U$ the upper triangular part of the matrix. Using this notation, we can state that the computational cost of Algorithm 1 is proportional to

$$|\boldsymbol{F}| + \sum_{j=1}^{n} |\boldsymbol{F}_j^U||\boldsymbol{F}_j^L| \ .$$

This is computationally affordable, especially considering that the computational cost of basic Gaussian elimination is proportional to $\frac{2}{3}n^3$. The number of non-zero entries is generally of the order of $\frac{1}{100}n^2$ in burnup matrices. As already mentioned, it is also advantageous that we only need to compute this symbolic factorization once when computing a CRAM approximation.

It should be mentioned that Algorithm 1 can also be executed row-wise. The implementation in the current release of Serpent is row-based, whereas the column-based version, considered here, has been adopted to the development version of Serpent 2.

3.2. **Diagonal Dominance.** It was previously stated that the special characteristics of burnup matrices allow using Gaussian elimination without partial pivoting. This matter is explained in the following.

In Gaussian elimination, a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ is transformed to lower triangular form by a series of updates, which corresponds to multiplying the matrix by a sequence of lower-triangular matrices $\boldsymbol{L}_j$ from the left. Let $\boldsymbol{A}^{(j)}$ denote the matrix obtained after j updates, i.e. $\boldsymbol{A}^{(j)} = \boldsymbol{L}_j \cdots \boldsymbol{L}_2 \boldsymbol{L}_1 \boldsymbol{A}$. After $n-1$ updates, the matrix becomes upper-triangular:

$$\boldsymbol{L}_{n-1}\boldsymbol{L}_{n-2}\cdots\boldsymbol{L}_2\boldsymbol{L}_1\boldsymbol{A} = \boldsymbol{U} \ , \tag{16}$$

after which setting $\boldsymbol{L} = \boldsymbol{L}_1^{-1}\boldsymbol{L}_2^{-1}\cdots\boldsymbol{L}_{n-2}^{-1}\boldsymbol{L}_{n-1}^{-1}$ gives the decomposition $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$. During the elimination, the elements of $\boldsymbol{L}$ are computed according to

$$L_{ij} = -\frac{a_{ij}^{(j-1)}}{a_{jj}^{(j-1)}} \ , \quad i > j \ . \tag{17}$$

Therefore, $|L_{ij}| \leq 1$, if the absolute values of the diagonal elements remain greater than the absolute values of the elements below the diagonal during the updates. It is well-known that for column diagonally dominant matrices, diagonal dominance is preserved during Gaussian elimination. However, as mentioned in Section 2, due to reactions that produce two or more nuclides, burnup matrices are not generally diagonally dominant.

According to Lemma 1 in[10], the element a_{ij} is *updated* during Gaussian elimination, if there exists a path from $i \to j$ satisfying Eq. (14) with length $m \geq 2$. Physically this means that there exists a transmutation path from nuclide j to nuclide i through nuclides whose ZAI index is smaller than the indices of i and j. In particular, the element a_{lj} affects the element a_{ij} through the updates if $l \in \{i_n\}$.

In burnup matrices, diagonal dominance is violated by reactions that produce two or more nuclides. Let k denote the index of the first fissionable nuclide. Aside from fission, the only nuclides emitted as by-products are the hydrogen isotopes ^{1}H, ^{2}H and ^{3}H and the helium isotopes ^{3}He and ^{4}He. According to the previous Lemma, for the corresponding rows to affect other rows during Gaussian elimination would require the existence of transmutation paths originating from the by-product nuclides. However, this is generally impossible, since all these nuclides are stable aside from ^{3}H (which decays into ^{3}He) and they do not elicit neutron reactions that would produce nuclides outside this group. The violation of diagonal dominance in the submatrix $\boldsymbol{A}(k : n, k : n)$, on the other hand, would require a transmutation path from a fissile nuclide to another fissile nuclide through a fission product nuclide, which is extremely unlikely in reactor conditions.[1]

[1]Interestingly, paths from fission product nuclides to fissile nuclides are theoretically possible if data based on nuclear models rather than measurements is considered. For example, the nuclear data library `TENDL-2011` [11] produced with the nuclear reaction program Talys [12] contains data

Algorithm 2 Numerical elimination

```
$\boldsymbol{v} = \mathbf{0}$
for $i = 1, 2, \ldots, n-1$ do
  for $j \in F(i)$ do
    $v_j = a_{ij}$
  end for
  for $j \in F(i)$ do
    if $i > j$ then
      $c_i = c_i - \frac{F_{ij}}{F_{jj}} c_j$
      for $k \in F(j)$ do
        $v_k = v_k - \frac{F_{ij}}{F_{jj}} F_{jk}$
      end for
    end if
  end for
  for $j \in F(i)$ do
    $F_{ij} = v_j$
  end for
end for
```

3.3. **Numerical Elimination.** After computing the non-zero structure of the fill-in matrix $\boldsymbol{F}$, the numerical elimination can be efficiently performed on this symbolic factorization using Algorithm 2. It has been derived from the method CELIMINATE in [13]. CELIMINATE considers a system of the form of $\boldsymbol{xM} = \boldsymbol{c}$, where $\boldsymbol{x} \in \mathbb{R}^{1\times n}$, $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{x} \in \mathbb{R}^{1\times n}$, whereas we consider a system of the form of Eq. (8).

When using a sparse matrix format, it is most efficient to store the matrices and implement the numerical elimination row-wise. The notation $F(i)$ is used to denote the set of column indices of the non-zero entries on the i^{th} row of the fill-in matrix $\boldsymbol{F}$, i.e. $j \in F(i)$ if $F_{ij} \neq 0$. These column indices need to sorted in ascending order. Algorithm 2 proceeds row by row by introducing zeros to the left of the matrix diagonal. Since the non-zero structure of the resulting matrix has been determined before starting the numerical elimination, the actual elimination phase can be performed with a minimal computational cost. An auxiliary vector $\boldsymbol{v}$ is employed in order to make the subtracting of the matrix rows more efficient. Notice that the elements in each row need to be accessible in order by their column indices. In practice, this requires sorting the matrix structure after computing the fill-in by Algorithm 1. In Serpent, insertion sort is used for this purpose. It is also convenient to store the diagonal elements of the matrix to an auxiliary vector in order to allow for an easy access to them during the computation.

that enables these paths. However, since these transmutation paths are extremely unlikely, they are not further considered here.

TABLE 1. CPU time required for computing the CRAM approximation of different orders for the test case with 1606 nuclides.

Approximation order	CPU time (s)
16	0.108
14	0.094
12	0.086
10	0.079
8	0.070
6	0.063

4. Numerical Results

To evaluate the efficiency and accuracy of the described technique, it was applied to a burnup system that can be considered to represent the extreme case in terms of the matrix size. The test case represents a PWR pin-cell with fuel irradiated to 0.1 MWd/kgU burnup with the time step of 12.5 days. The number of nuclides present in the calculation is 1606, which is representative of the practical maximum encountered in burnup calculations. The absolute values of the matrix elements range from zero to the order of 10^{21}, the largest entries corresponding to the decay of ^{7}B whose half-life is $\sim 10^{-24}$ s.

The burnup matrix was formed by indexing the nuclides according to their ZAI index. The sparsity pattern of the resulting matrix is shown in Figure 1. The burnup matrix has 59 668 non-zero entries, which corresponds to a density of approximately 2.3%. Rows $2, \dots, 6$ correspond to hydrogen and helium isotopes. These rows are relatively dense because these nuclides are emitted as by-products in some neutron and decay reactions. The dense columns in the right part of the matrix result from fission reactions. The non-zero elements below the diagonal correspond to beta negative decay and the (n, γ) reaction.

The CRAM approximations of orders 6, 8, 10, 12, 14 and 16 were computed for this burnup system. The calculations were performed with the development version of Serpent 2 on a quad-core 3.00 GHz Intel Xeon X5450 CPU. The respective calculation times are shown in Table 1. These calculation times support the previous observation that in terms of computational efficiency, CRAM either matches or outperforms other established methods used in the context of burnup equations [2]. For each approximation order, the total calculation time is dominated by the numerical elimination phase. For the approximation of order 16, the numerical elimination takes about 60% of the total CPU time, as opposed to about 40% for the approximation of order 6. According to Eq. (7), the computational cost of this phase for approximation order k is proportional to $k/2$. The proportion of the total calculation time for performing the symbolic factorization phase, including the sorting of the row indices, increases from 13% to 23% when the approximation order is reduced from 16 to 6. The number of fill-in elements is 7 449 for this test case matrix.

In order to evaluate the accuracy and to study the stability of the chosen approach, a reference solution was computed with MATLAB's Symbolic Toolbox using

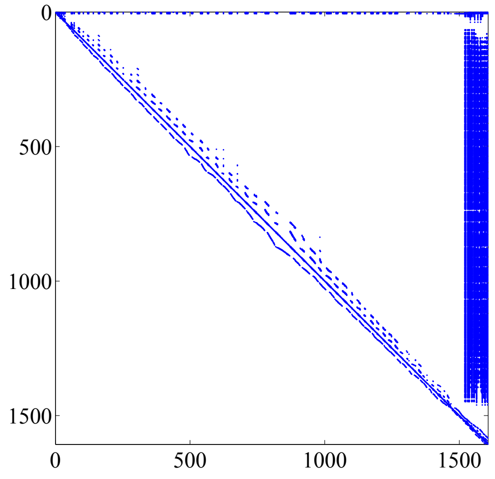

Figure 1. The non-zero structure of the test case burnup matrix.

high-precision arithmetics for approximation order 16. As round-off errors are accumulated with an increasing CRAM approximation order, the highest considered approximation order should provide the most conservative error estimate for the presented test case. The relative difference between the reference solution and the solution computed with Gaussian elimination using double precision is plotted in Figure 2. Recalling that we can expect about 15 correct digits with double precision arithmetics, it can be seen from the figure that, apart from ^{1}H, not many digits have been lost to round-off errors. As explained in Section 3, Gaussian elimination is stable, if the matrix growth factor ρ defined according to Eq. (13) is of the order of 1. For this test case, the growth factor is exactly one for each linear system, indicating that no amplification of the matrix entries takes place during the numerical elimination. However, the practicality of this observation is hindered by the fact that $\boldsymbol{A}$ contains entries of very large magnitude. Interestingly, the poorest relative accuracy, corresponding to about 12 correct digits, is obtained for the nuclide ^{1}H. In the burnup matrix, this nuclide corresponds to the first row, which does not change during the numerical elimination. Therefore, the round-off error in the result is solely attributable to the back substitution phase in the algorithm and reflects the characteristics of the original matrix. In particular, due to the decay of ^{7}B by proton emission, this first row in the matrix contains the largest element of the matrix.

To further illustrate the numerical characteristics of the linear systems encountered in CRAM, the same test case was solved with some standard iterative solvers available in Matlab. The considered iterative methods were the generalized minimum residual (GMRES) method and the biconjugate gradient (BCG) method, and they were used to compute the CRAM approximation of order 16. Figure 3 shows the relative difference between the solutions obtained with these iterative techniques

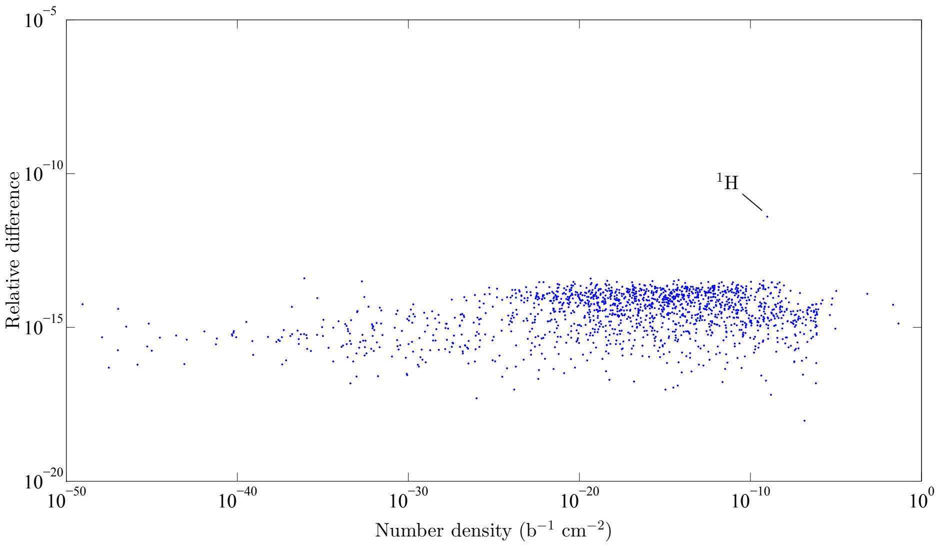

FIGURE 2. The relative difference between the test case solutions obtained using double and high-precision arithmetics for CRAM approximation of order 16.

and the reference solution based on Gaussian elimination with high-precision arithmetics. As can be seen from the figure, the accuracy of both iterative methods is questionable for this test case. This can be attributed to the methods stagnating before a reasonable accuracy was reached. With the GMRES method, the stagnation occurred between 23 and 42 iterations, with the norm of the relative residual being in the order of 10^{-4}. The BCG method converged slightly faster, with all relative residuals being in the order of 10^{-5} after less than 10 iterations when the method stagnated.

Both of these iterative methods are based on projecting the original problem to a lower dimensional Krylov subspace. Therefore, the observed convergence problems can ultimately be traced back to the spectral properties of the burnup matrix. In the GMRES method, the iteration is based on minimizing the 2-norm of the residual in a Krylov subspace, whose dimension increases with each iteration. In the BCG method, on the other hand, the same residual is forced orthogonal to another Krylov subspace. (For further details, see e.g., [9].) However, it should be kept in mind that these basic iterative methods can be substantially improved by preconditioning techniques. Therefore, this study does not imply that all Krylov-based iterative methods are ill-suited for solving these linear systems. However, it is clear that the numerical method for this purpose should be carefully selected and tested.

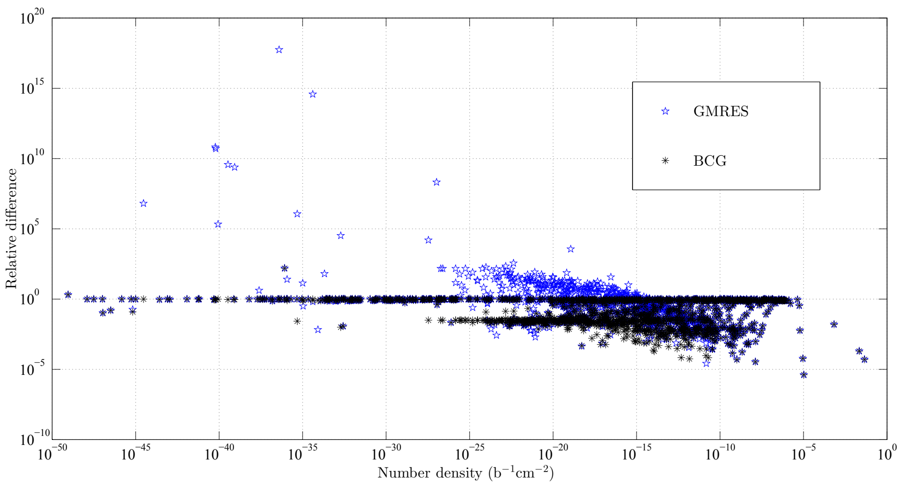

FIGURE 3. The relative difference between the reference solution and solutions obtained with the generalized minimum residual (GMRES) method and the biconjugate gradient (BCG) method for a CRAM approximation of order 16.

5. CONCLUSIONS

The computation of the matrix exponential in burnup calculations has been traditionally challenging due to the extreme stiffness of the problem resulting from the extensively varying magnitudes of the transmutation and decay constants. Based on the discovery that the eigenvalues of burnup matrices are generally confined to the vicinity of the negative real axis, the Chebyshev Rational Approximation Method (CRAM) was successfully proposed and applied to solving the burnup equations by the authors, effectively eliminating all previously encountered computational problems. The method can be described as the best rational approximation on the negative real axis, and it has been shown [1, 3] to give a robust and accurate solution to the burnup equations combined with high computational efficiency.

For numerical reasons it is generally advantageous to compute the CRAM approximation in its partial fraction form. Since these partial fraction coefficients are fixed for each approximation order, implementing CRAM in a burnup code is extremely straightforward—only a linear solver is required in addition to these coefficients. Unfortunately, the difficult numerical properties of burnup matrices may compromise the accuracy of some numerical methods used for solving the linear systems. In this paper, a direct method was considered. The introduced technique is based on sparse Gaussian elimination, where the non-zero structure of the resulting upper triangular matrix is determined before beginning the numerical elimination phase. The numerical properties of burnup matrices and the stability of Gaussian elimination were considered. The efficiency and accuracy of this approach were demonstrated by applying it to a large test case with over 1600 nuclides. This paper's contribution together with the recently reported, more accurate CRAM partial fraction coefficients [3, 5] are hoped to further facilitate the implementation of CRAM for burnup calculations.

Acknowledgment

This work was funded through the Finnish National Research Programme on Nuclear Power Plant Safety 2011–2014, SAFIR2014.

References

[1] M. Pusa and J. Leppänen, "Computing the Matrix Exponential in Burnup Calculations," *Nucl. Sci. Eng.*, **164**, 2, 140–150 (2010).

[2] A. Isotalo and P. A. Aarnio, "Comparison of depletion algorithms," *Ann. Nucl. Energy*, **38**, 2–3, 261–268 (2011).

[3] M. Pusa, "Rational Approximations to the Matrix Exponential in Burnup Calculations," *Nucl. Sci. Eng.*, **169**, 2, 155–167 (2011).

[4] T. Schmelzer, "Carathéodory–Fejér approximation", `http://www.mathworks.com/matlabcentral`, Matlab Central (Nov. 2008).

[5] M. Pusa, "Correction to Partial Fraction Decomposition Coefficients for Chebyshev Rational Approximation on the Negative Real Axis," `arXiv:1206.2880v1 [math.NA]` (2012).

[6] "PSG2/Serpent Monte Carlo Reactor Physics Burnup Calculation Code", `http://montecarlo.vtt.fi`, VTT Technical Research Centre of Finland (2012).

[7] J. Leppänen and M. Pusa, "Burnup calculation capability in the PSG2/Serpent Monte Carlo Reactor Physics Code," International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009), Springer-Verlag, Saratoga Springs, New York (2009), CD-ROM, 1662 - 1673, American Nuclear Society, LaGrange Park, IL.

[8] D. G. Cacuci, ed., *Handbook of Nuclear Engineering*, Vol. 2: Reactor Design, Springer (2010).

[9] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, SIAM (1997).

[10] D. J. Rose and R. E. Tarjan, "Algorithmic Aspects of Vertex Elimination on Directed Graphs," *SIAM J. Appl. Math*, **34**, 176–197 (1978).

[11] A. J. Koning and D. Rochman, "TENDL-2011, TALYS-based Evaluated Nuclear Data Library", Nuclear Research and Consultancy Group (NRG) (2011).

[12] A. Koning, S. Hilaire and S. Goriely, "TALYS-1.4, a Nuclear Reaction Program, User Manual", Nuclear Research and Consultancy Group (NRG) (2011).

[13] R. E. Tarjan, "Graph Theory and Gaussian Elimination." Tech. Rep. CS-TR-75-526, Stanford University, Department of Computer Science, Stanford, CA, USA (1975).

VTT Technical Research Centre of Finland, Nuclear Energy, P.O. Box 1000, FI-02044 VTT, Finland
E-mail address: `maria.pusa@vtt.fi`