Faculty of Sciences

Department of Applied Mathematics,

Computer Science and Statistics

# Implementation, validation and comparison of different algorithms to solve the Bateman equations for very large systems

Vranckx Maren

*Supervisors:* Prof. M. Van Daele (UGent)
Dr. ir. G. Van den Eynde (SCK-CEN)

Master thesis submitted to obtain the academic degree of master in Mathematics (Applied Mathematics)

Academic year 2015-2016

# Acknowledgements

I would like to thank the following people, for their contribution and support, which made it possible to create this master thesis.

First, my gratitude goes out to my supervisor prof M. Van Daele, for his suggestions, backing, patience and guidance in writing this master thesis.

I would like to thank my supervisor G. Van Den Eynde for the possibility to make this master thesis in cooperation with SCK-CEN (Studiecentrum voor kernenergie / Centre d'étude de l'énergie nucléaire), his support and advice.

I would also like to express my gratitude to helpdesk DICT UGent, Mathijs, Hilde and Annemieke for their practical support, vision and aid.

A very special thanks goes out to my parents, my brother and my animals for their constant support during my study years.

Finally, I would like to thank all the people, who helped me realize my dream, for their contribution, big or small.



June 2016
Maren Vranckx

# Contents

# List of Abbreviations

| | |
|---|---|
| ALEPH | A Monte Carlo Burnup Code |
| BLAS | Basic Linear Algebra Subprograms |
| CPU | Central processing unit |
| CRAM | Chebyshev Rational Approximation Method |
| CRS | Compressed Row Storage |
| HPC | High performance computing |
| MUMPS | MUltifrontal Massively Parallel sparse direct Solver |
| SCK-CEN | Studiecentrum voor kernenergie / Centre d'étude de l'énergie nucléaire (The Belgian nuclear research centre) |

# 1 Introduction

## 1.1 General introduction

Since 2004, SCK-CEN has been engaged in the development of the ALEPH code. The ALEPH code is used to determine the behaviour of nuclear reactor cores. In this code, two equations are important, namely the neutron transport equations and the Bateman equations. The neutron transport equations determine the neutron flux and the Bateman equations are used to describe the time evolution of the nuclide concentrations. The transport equations can be solved using stochastic methods, such as the Monte Carlo algorithms. In general, Monte Carlo methods can be used to look for numerical solutions equations with a probabilistic interpretation. A way of implementing Monte Carlo methods is through Monte Carlo N-particle code. The objective of this master thesis is to improve the way the Bateman equations are currently worked out. The Bateman equations will be solved with deterministic methods.
This master thesis is organized as follows. First, the Bateman equations with their properties will be described. Thereafter, the different algorithms which could possibly be used to solve the Bateman equations, will be discussed. These algorithms are

- calculating the exponential of the system matrix using the scaling and squaring algorithm
- Chebyshev Rational Approximation method
- RadauIIA method.

Finally, it will be examined which is the best algorithm to solve the Bateman equations.

## 1.2 The Bateman equations

The Bateman equations, which are also called the burnup equations, are used to describe the time evolution of nuclides in a nuclear system. This evolution consists of radioactive decay from one nuclide to another and the production of nuclides by fission, neutron capture, ... The Bateman equations are a set of first order differential equations of the form

$$n'(t) = Bn(t), \qquad n(0) = n_0\,, \tag{1.2.1}$$

with $n(t) \in \mathbb{R}^{\mathrm{m}}$ the nuclide concentration vector, $n_0 \in \mathbb{R}^{\mathrm{m}}$ the initial nuclide concentration and $B \in \mathbb{R}^{\mathrm{mxm}}$ the Bateman or burnup matrix.
The diagonal elements of the Bateman matrix $b_{ii}$ represent the rate by which a nuclide $i$ is transformed to other nuclides. The off-diagonal elements $b_{ij}$ describe the rate by which nuclide $j$ is converted into nuclide $i$ by a physical process. Throughout this master thesis, the Bateman matrix is assumed to have constant elements in a certain time step.
The Bateman equations can be solved exactly, giving the solution $n(t) = e^{Bt}n_0$. Here the exponential of the matrix $Bt$ is defined by

$$e^{Bt} = I + \sum_{k=1}^{\infty} \frac{(Bt)^k}{k!} \tag{1.2.2}$$

with $I$ the identity matrix.
This definition of the exponential of a matrix is based on the property that the exponential of a scalar can be defined as the Taylor expansion

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \quad \text{with} \quad x \in \mathbb{C}. \tag{1.2.3}$$

To transform from a scalar function $f(x)$ with scalar coefficients to a matrix function $f(M)$, one replaces, in the case of a polynomial or rational function, the argument $x$ through the matrix $M$ and the number one becomes the identity matrix. When a division is presented in the expression of the function, it will become the inverse of a matrix. To define a general matrix function, a definition will be given in the next section, based on the Jordan canonical form, the Hermite interpolation and the Cauchy integral formula.

## 1.3 Matrix functions

In this section, three ways to define a general matrix function will be described via

- the Jordan canonical form
- the Hermite interpolation
- the Cauchy integral formula.

The material is based on the book "Function of matrices" of N.J. Higham[1].
The matrix function via the Jordan canonical form uses the possibility to express a matrix $M \in \mathbb{C}^{\mathrm{mxm}}$ in the Jordan canonical form.

**Definition 1.3.1** (Jordan canonical form)
*Consider a matrix $M \in \mathbb{C}^{\mathrm{mxm}}$.*
*Let $\lambda_1, \ldots, \lambda_l$ be the different eigenvalues of M.*
*The Jordan canonical form of M is defined by a non-singular $Z \in \mathbb{C}^{\mathrm{mxm}}$ and the Jordan canonical blocks $J_k$ such that*

$$Z^{-1}MZ = J^o = diag(J_1, J_2, \ldots, J_p) \tag{1.3.1}$$

$$with\, J_k = J_k(\lambda_k) = \begin{bmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & \\ & & & 1 \\ & & & \lambda_k \end{bmatrix} \in \mathbb{C}^{\mathrm{m_k xm_k}}. \tag{1.3.2}$$

*Hereby is $m_1 + \ldots + m_p = m$ and $diag(f(J_k))$ represents a square matrix with $f(J_k)$ on the main diagonal.*

Before giving the definition of the matrix function via the Jordan canonical form, some terminology has to be introduced. The set of eigenvalues of a matrix $M \in \mathbb{C}^{\mathrm{mxm}}$, noted $\lambda$(M), is called the spectrum of $M$. For a matrix $M$ with distinct eigenvalues $\lambda_1$, ..., $\lambda_l$, it is said that a function $f$ is defined on the spectrum of $M$ if the values

$$f^{(j)}(\lambda_i), \qquad j = 0, \ldots, n_i - 1 \,\text{and}\, i = 1, \ldots, l \tag{1.3.3}$$

exist[1]. Hereby is $n_k$ the index of $\lambda_k$, which represents the order of the largest Jordan block in which $\lambda_k$ appears.

**Definition 1.3.2** (Matrix function via Jordan canonical form[1])
*Let f be defined on the spectrum of $M \in \mathbb{C}^{\mathrm{mxm}}$ and let M have the Jordan canonical form. Then*

$$f(M) := Zf(J^o)Z^{-1} = Zdiag(f(J_k))Z^{-1} \tag{1.3.4}$$

*where*

$$f(J_k) := \begin{bmatrix} f(\lambda_k) & f'(\lambda_k) & \cdots & \frac{f^{(m_k-1)}(\lambda_k)}{(m_k-1)!} \\ & f(\lambda_k) & \ddots & \vdots \\ & & \ddots & \\ & & & f'(\lambda_k) \\ & & & f(\lambda_k) \end{bmatrix}. \tag{1.3.5}$$

Another way to define a matrix function, is via the Hermite polynomial. This definition makes use of the minimal polynomial of $M$. The minimal polynomial of $M \in \mathbb{C}^{\mathrm{mxm}}$ is defined to be the unique monic polynomial $\psi$ of the lowest degree which satisfies $\psi(M)=0$[1]. A monic polynomial is characterized by the fact that the nonzero coefficient associated with the highest degree is one.

**Definition 1.3.3** (Matrix function via Hermite interpolation form[1])
*Let f be defined on the spectrum of $M \in \mathbb{C}^{mxm}$ and let $\psi$ be the minimal polynomial of M. The degree of $\psi$ will be denoted as $deg\,\psi$. Then f(M):=p(M), where p is the polynomial of degree less than*

$$\sum_{i=1}^{l} n_i = deg\,\psi \tag{1.3.6}$$

*that satisfies the interpolation conditions*

$$p^{(j)}(\lambda_i) = f^{(j)}(\lambda_i), \qquad j = 0, \ldots, n_i - 1\, and\, i = 1, \ldots, l\,. \tag{1.3.7}$$

*There is a unique such p and it is known as the Hermite interpolating polynomial.*

The last definition of a matrix function is based on a generalization of the Cauchy integral theorem.

**Definition 1.3.4** (Matrix function via Cauchy integral[1])
*For $M \in \mathbb{C}^{mxm}$,*

$$f(M) := \frac{1}{2\pi i} \int_{\Gamma} f(z)(zI - M)^{-1} dz \tag{1.3.8}$$

*where f is analytic on and inside a closed contour $\Gamma$ the encloses $\lambda$(M) (the spectrum of M).*

Finally, it has to be remarked that the definition via the Jordan canonical form and the definition via Hermite interpolation are equivalent. Furthermore, if $f$ is analytic, the three definitions of matrix functions (definitions 1.3.2, 1.3.3 and 1.3.4) are equivalent.

## 1.4 Examples of the Bateman equations

In this paragraph, the Bateman problems which were used for this master thesis will be explained.

### 1.4.1 The Polonium problem

A simple Bateman problem based on Polonium–210 is described in this section. Polonium–210 ($^{210}$Po) can be produced from neutron capture by irradiating Bismuth-209 ($^{209}$Bi) with neutrons in a nuclear reactor. This production can be expressed by the following chain

$$^{209}\mathrm{Bi} \xrightarrow{(n,\gamma)} {}^{210}\mathrm{Bi} \xrightarrow{5.013\mathrm{d}} {}^{210}\mathrm{Po} \xrightarrow{138.376\mathrm{d}} {}^{206}\mathrm{Pb}\ [2]\ .$$

In this chain, Bismuth-209 ($^{209}$Bi) forms Bismuth-210 ($^{210}$Bi) through neutron capture and consequently gamma decay ($\gamma$-decay). Hereafter Bismuth-210 decays to Polonium-210. The half-life of Bismuth-210 amounts to 5.013 days. Finally Polonium-210, which has a half-life of 138.376 days, decays to Lead-206 by emitting an alpha particle. Lead-206 ($^{206}$Pb) is a stable nuclide. The change in nuclide concentration can be expressed by the following Bateman equations

$$\begin{aligned} \tfrac{dn_{Bi209}}{dt} &= -d_{Bi209} n_{Bi209} \\ \tfrac{dn_{Bi210}}{dt} &= d_{Bi209} n_{Bi209} - d_{Bi210} n_{Bi210} \\ \tfrac{dn_{Po210}}{dt} &= d_{Bi210} n_{Bi210} - d_{Po210} n_{Po210}\,. \end{aligned} \tag{1.4.1}$$

Hereby represents $n_{nuclide}(t)$ the nuclide concentration, $d_{nuclide}$ the decay constant of the nuclide and $\frac{dn_{nuclide}}{dt}$ the change in nuclide concentration over time. The decay constant is made up from other fysical concepts, which will be explained furter.
Based on these equations, the nuclide concentration of Bismuth-209, Bismuth-210 and Polonium-210 can be calculated after a certain days of irradiation. The system of Bateman equations can also be expressed in matrix form as follows

$$\frac{d}{dt}\begin{bmatrix} n_{Bi209} \\ n_{Bi210} \\ n_{Po210} \end{bmatrix} = \begin{bmatrix} -d_{Bi209} & 0 & 0 \\ +d_{Bi209} & -d_{Bi210} & 0 \\ 0 & +d_{Bi210} & -d_{Po210} \end{bmatrix} \begin{bmatrix} n_{Bi209} \\ n_{Bi210} \\ n_{Po210} \end{bmatrix}. \tag{1.4.2}$$

The values of $d_{nuclide}$ are

$$\begin{aligned} d_{Bi209} &= 1.83163 \cdot 10^{-12}\ s^{-1} \\ d_{Bi210} &= 1.60035 \cdot 10^{-6}\ s^{-1} \\ d_{Po210} &= 5.79764 \cdot 10^{-8}\ s^{-1}. \end{aligned}$$

Therefor, the Bateman matrix looks like

$$\begin{bmatrix} -1,83163 \cdot 10^{-12} & 0 & 0 \\ +1,83163 \cdot 10^{-12} & -1,60035 \cdot 10^{-6} & 0 \\ 0 & +1,60035 \cdot 10^{-6} & -5,79764 \cdot 10^{-8} \end{bmatrix}. \tag{1.4.3}$$

The eigenvalues of this matrix are -1.8316·10$^{-12}$, -1.60035·10$^{-6}$ and -5.79764·10$^{-4}$ and the matrix 1-norm is 3.2007·10$^{-6}$.
The initial nuclide concentrations are

$$\begin{aligned} n_{Bi209,\,0} &= 6.95896{\cdot}10^{-4} \frac{at}{b \cdot cm} \\ n_{Bi210,\,0} &= 0\ \frac{at}{b \cdot cm} \\ n_{Po210,\,0} &= 0\ \frac{at}{b \cdot cm}\ , \end{aligned}$$

where at is the number of atoms and b represents barn, a unit of area ($10^{-28}$ m$^2$ or $10^{-24}$ m$^2$).
To explain in more detail where the decay constant of a nuclide $d_{nuclide}$ comes from, consider the equation

$$d_{nuclide} = \sigma_{n,\gamma\, nuclide} \phi\,, \tag{1.4.4}$$

were $\phi$ indicates the neutron flux and $\sigma_{n,\gamma\, nuclide}$ is the energy-averaged production cross section[3].
The neutron flux $\phi$ is the number of neutrons passing through a cross-sectional unit area per unit time. Further, the energy-averaged production cross section $\sigma$ is a measure to determine how likely it is to have an interaction between an incident particle and a target object. It is quantified in units of area (m$^2$) and typically barn ($10^{-28}$ m$^2$) is used. With the energy-averaged production cross section ($\sigma$) and the nuclide concentration ($n$), the macroscopic cross section ($\Sigma$) can by calculated by using the equality

$$\Sigma = \sigma n. \tag{1.4.5}$$

From this, the mean free path $\frac{1}{\Sigma}$ can be deduced. It stands for the mean distance a neutron can discard in a material between collisions. The higher $\Sigma$, the smaller $\frac{1}{\Sigma}$ and therefore the more chance atoms interact.
The system of Bateman equations (1.4.1) can be solved analytically. The concentration of the $k^{\text{th}}$ nuclide after time t[2] is

$$n_k(t) = \frac{n_1(0)}{d_k} \cdot \sum_{i=1}^{k} d_i \alpha_i e^{-d_i t} \tag{1.4.6}$$

with

$$\alpha_i = \prod_{j=1, j\neq i}^{k} \frac{d_i}{d_j - d_i}\,. \tag{1.4.7}$$

Based on these formulas, the nuclide concentration after 90 days of irradiation, calculated in Maple 2015.1[4], is

$$\begin{aligned} n_{Bi209}(90 \text{ days}) &= 0.0006958860886 \\ n_{Bi210}(90 \text{ days}) &= 7.964521967 \cdot 10^{-10} \\ n_{Po210}(90 \text{ days}) &= 7.451824964 \cdot 10^{-9}. \end{aligned}$$

Maple 2015.1 works, by default, with ten numbers of digits when doing calculations with floating-point numbers.

Figure 1, figure 2 and figure 3 show the evolution of the nuclide concentration over a period of one year for respectively $^{209}$Bi, $^{210}$Bi and $^{210}$Po.
Throughout this master thesis, this problem will be referred to as the Polonium problem.

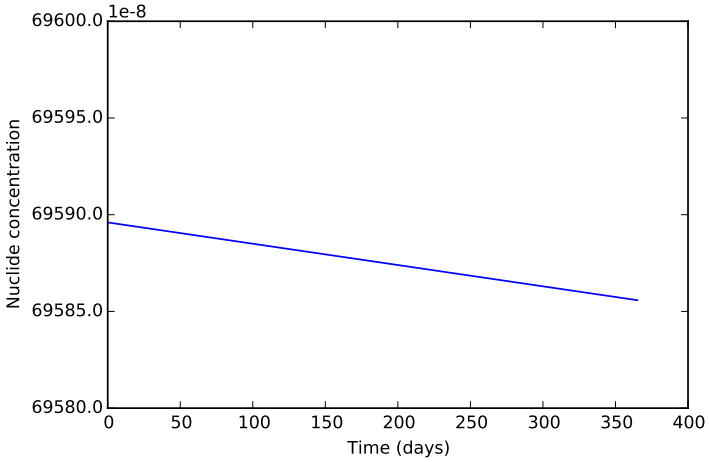


Figure 1: The evolution of the nuclide concentration of $^{209}$Bi

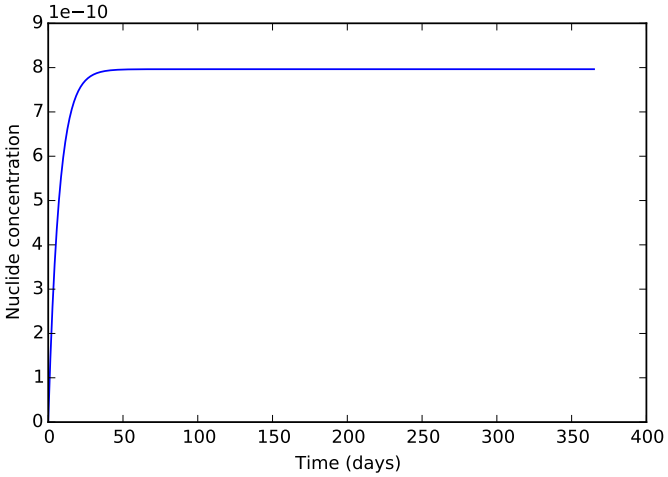


Figure 2: The evolution of the nuclide concentration of $^{210}$Bi

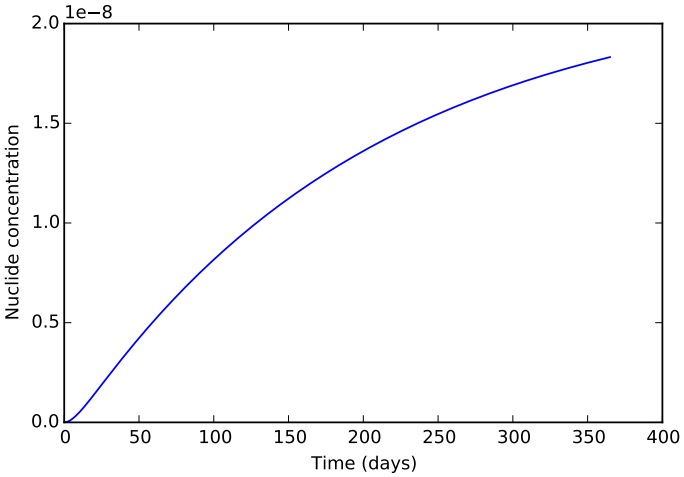


Figure 3: The evolution of the nuclide concentration of $^{210}$Po

### 1.4.2 The decay problem

It is a system of 3771 equations and contains nuclides which transform to other nuclides by radioactive decay. The Bateman matrix of this problem will be called the decay matrix. The decay matrix has 17533 nonzero elements and his structure can be seen in figure 4. The dots are the nonzero elements of the matrix and the colours represent the magnitude of the matrix element. It could be noticed that there is a cluster of points on the right side of the matrix. The decay transmutation that takes place here is called spontaneous fission.

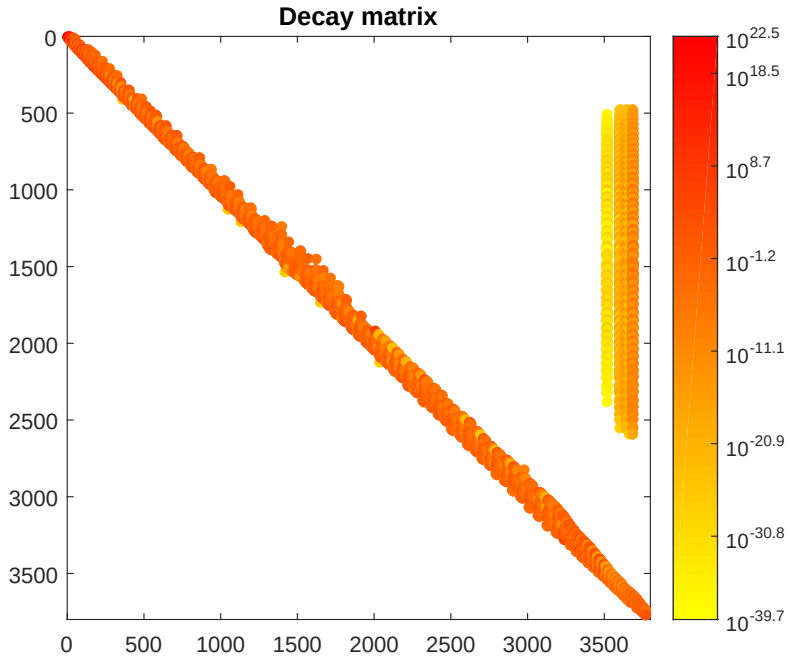


Figure 4: The structure of the decay matrix

The eigenvalues of the decay matrix are represented in figure 5. Herefor, a logarithmic scale for the x-axis is used. All the eigenvalues of the decay problem are negative. Since negative values can't be represented by using a logarithmic scale, the figure shows the opposite value of the eigenvalues. Notice that there are almost no eigenvalues between around $-10^{11}$ and $-10^{17}$. The largest eigenvalue in modulus of the decay problem is $3.01368 \cdot 10^{22}$.

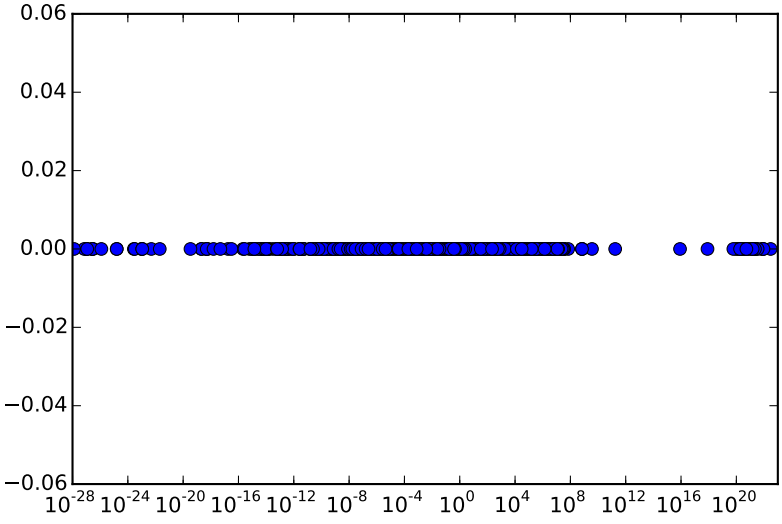


Figure 5: The values $-\lambda$, $\lambda$ being an eigenvalue of the decay matrix, pictured in the complex plane

Furthermore, the matrix 1-norm of the decay problem is $6.02737 \cdot 10^{22}$.

### 1.4.3 The irradiation problem

The irradiation problem has 3771 equations and consists besides radioactive decay, also of the production of nuclides by fission, neutron capture, ... The matrix of this problem will be referred to as the irradiation matrix. It has 69010 nonzero elements. Figure 6 shows the structure of the irradiation matrix, whereby the dots represent the nonzero elements of the matrix and the colours give an indication of the size of the matrix element.

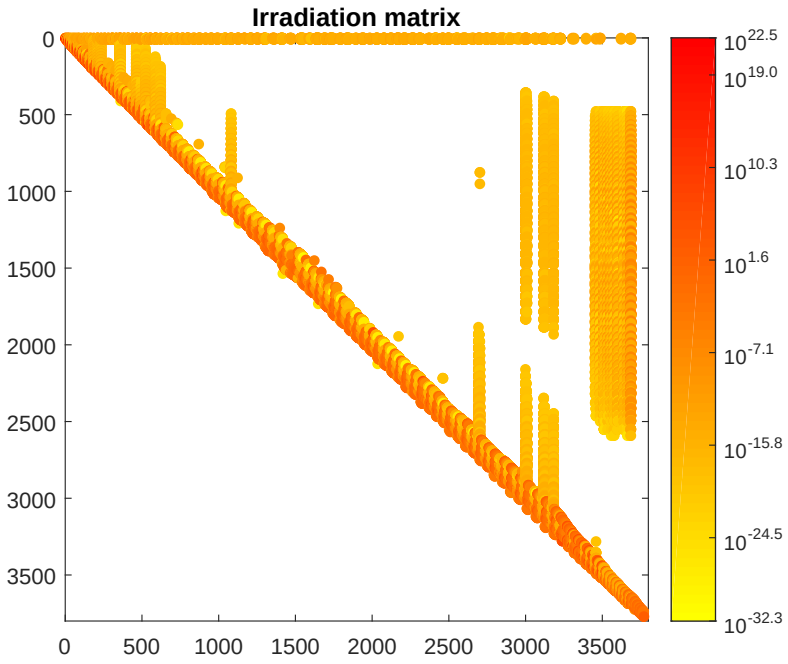


Figure 6: The structure of the irradiation matrix

In figure 7, the eigenvalues of the irradiation matrix with a negative real part are displayed. Since a logarithmic scale is used for the x-axis, the opposite value of the eigenvalues is shown. It can be noticed that there are almost no eigenvalues between around $-10^{11}$ and $-10^{18}$. The irradiation problem has also some eigenvalues with a positive real part, represented in figure 8. The biggest eigenvalue with a positive real part in modulus is 1722.28791. Since calculations are made with about 16 decimal digits, eigenvalues with a value lower than $10^6$ aren't calculated very accurate. Therefore, the value 1722.28791 is a numerical artefact.

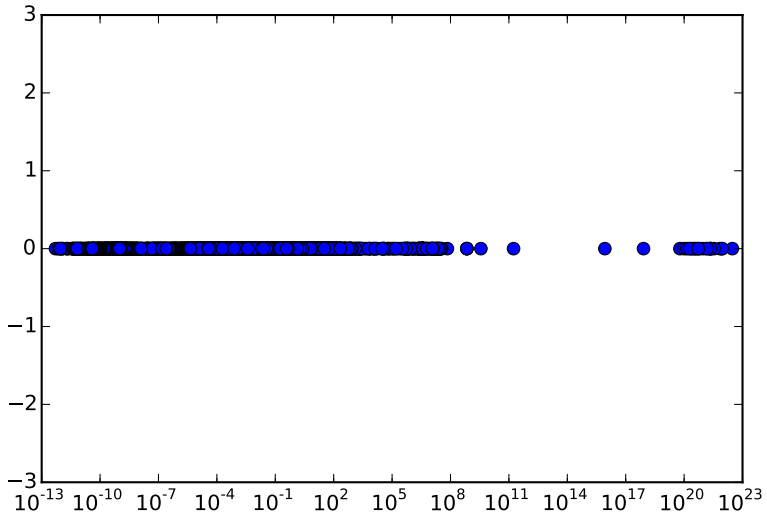


Figure 7: The values $-\lambda$, $\lambda$ being an eigenvalue of the irradiation matrix with a negative real part, pictured in the complex plane

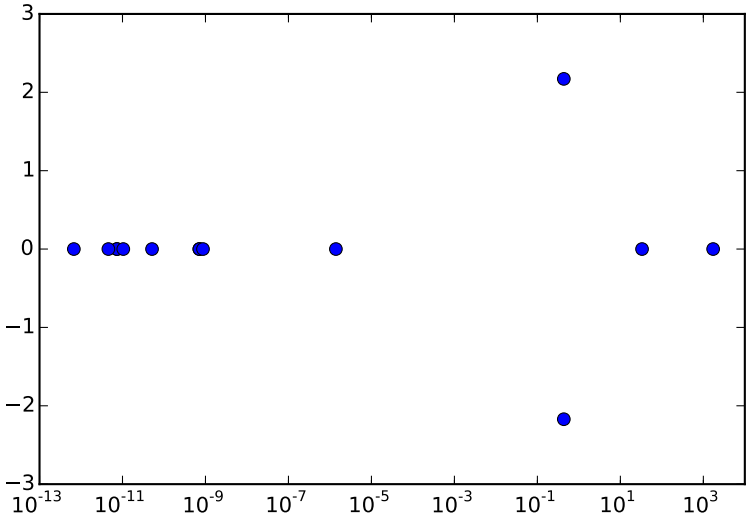


Figure 8: The values $\lambda$, $\lambda$ being an eigenvalue of the irradiation matrix with a positive real part, pictured in the complex plane

At last, $6.02737 \cdot 10^{22}$ is the matrix 1-norm.

### 1.4.4 The fresh fuel problem

The fresh fuel problem includes 3701 equations and contains fresh $UO_2$ fuel at the beginning of the irradiation. The fuel is irradiated for six days. In this problem, only nuclides of Uranium and Oxygen are given. The structure of the fresh fuel matrix, the Bateman matrix of this problem, is shown in figure 9. The number of nonzero elements in the fresh fuel matrix is 42464.

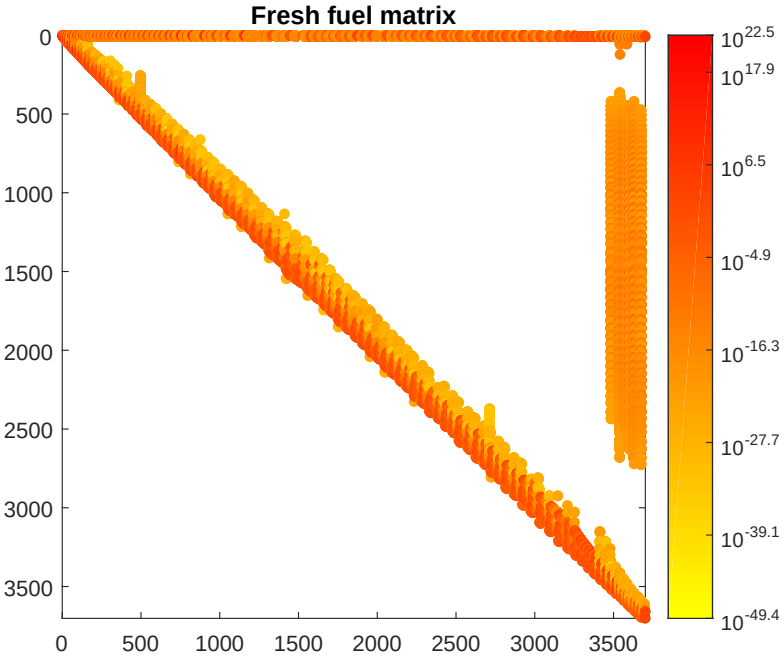


Figure 9: The structure of the fresh fuel matrix

The eigenvalues of this matrix with a negative real part are represented in figure 10. Again are the opposite value of the eigenvalues shown since a logarithmic scale is used for the x-axis. In figure 11, the eigenvalues with a positive real part are shown.

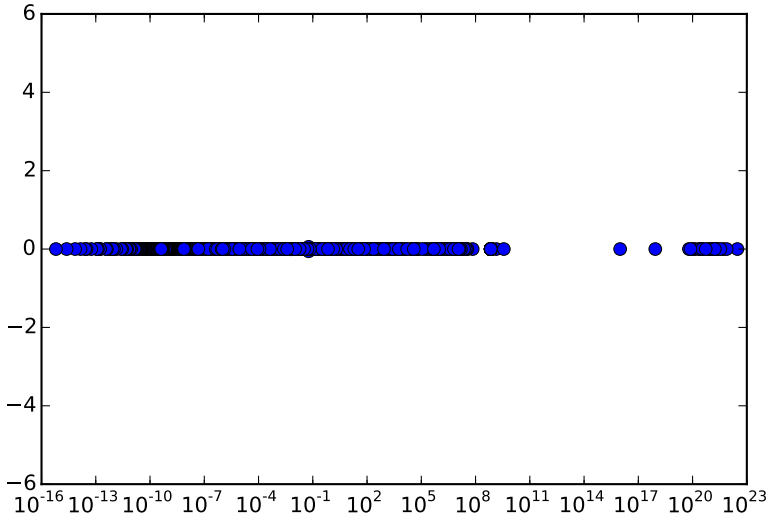


Figure 10: The values $-\lambda$, $\lambda$ being an eigenvalue of the fresh fuel matrix with a negative real part, pictured in the complex plane

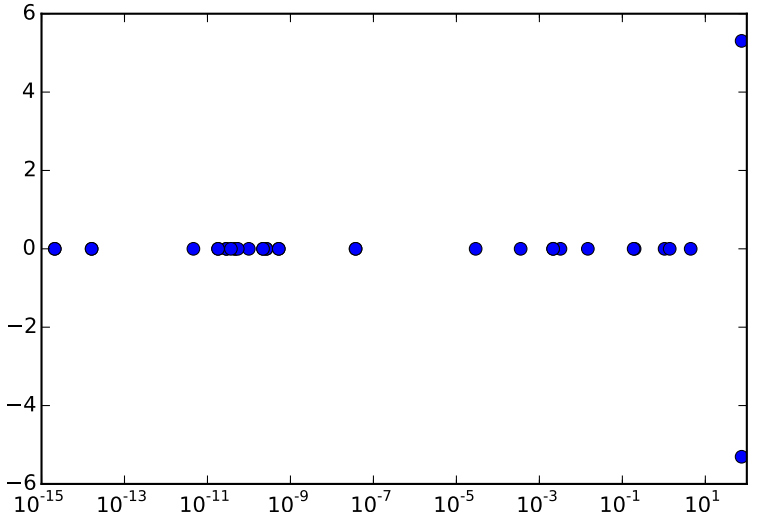


Figure 11: The values $\lambda$, $\lambda$ being an eigenvalue of the fresh fuel matrix with a positive real part, pictured in the complex plane

The matrix 1-norm of the fresh fuel matrix is $6.02737 \cdot 10^{22}$.

### 1.4.5 The burned fuel problem

The size of this system is 3701 equations. The initial concentration vector of this problem has more nonzero elements than the initial concentration vector in the fresh fuel problem. For 34.4 days, the fuel is irradiated. The Bateman matrix of this problem will be called the burned fuel matrix and his structure can be seen in figure 12. The burned fuel matrix has 44357 nonzero elements.
Figure 13, displays the eigenvalues with a negative real part. Because a logarithmic scale is used for the x-axis, the opposite value of the eigenvalues are shown. Figure 14 demonstrates the eigenvalues with a positive real part.
Furthermore, the matrix 1-norm is $6.02737 \cdot 10^{22}$.

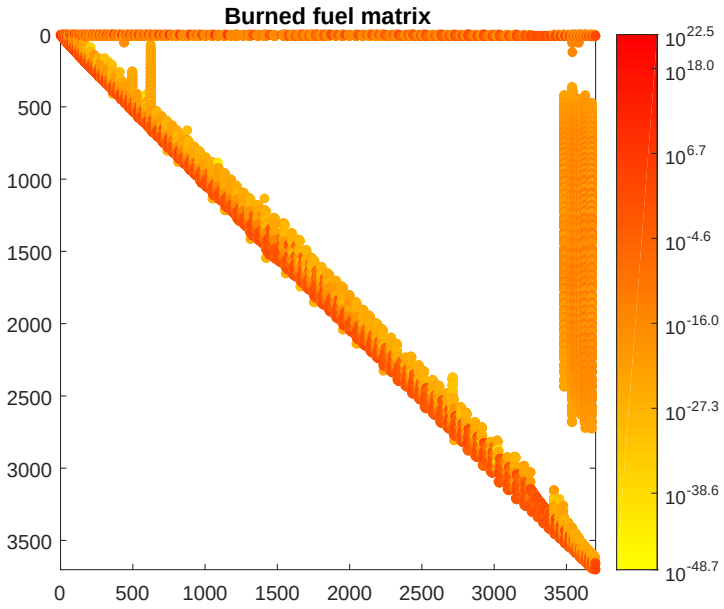


Figure 12: The structure of the burned fuel matrix

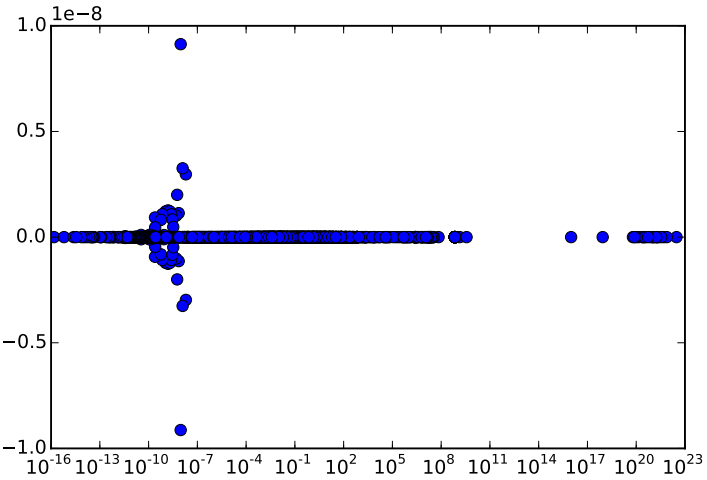


Figure 13: The values $-\lambda$, $\lambda$ being an eigenvalue of the burned fuel matrix with a negative real part, pictured in the complex plane

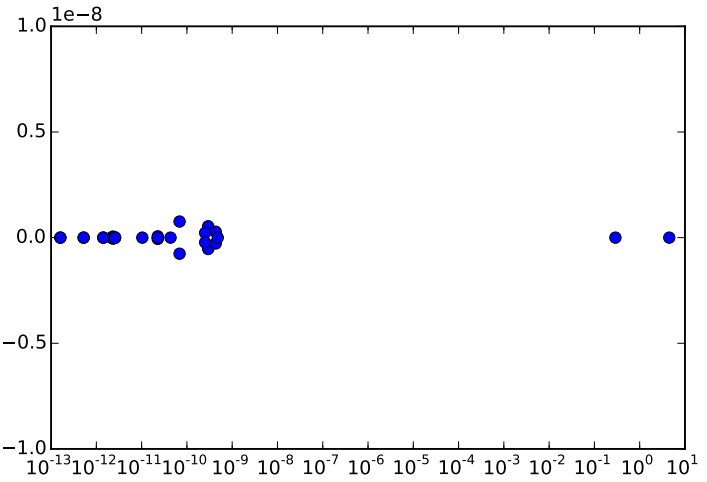


Figure 14: The values $\lambda$, $\lambda$ being an eigenvalue of the burned fuel matrix with a positive real part, pictured in the complex plane

## 1.5 Properties of the Bateman matrix

In this section, the properties of the Bateman matrix, discovered by the investigation of the different examples, will be summarized. The properties are

- depending on the number of nuclides one wishes to track, the system of Bateman equations can be small, like the Polonium problem, or very large, for example the 3771 equations of the decay problem
- the Bateman matrices are very sparse
- the half-life times can vary significantly which induces eigenvalues with extremely small and large magnitudes
- the eigenvalues of the Bateman matrix are typically limited to a region near the negative real axis[5]
- the matrix 1-norm can be very large.

Finally, it has to be remarked that the data in the Bateman matrices is based on experimental data. Hence, the numbers in the Bateman matrices are generally correct up to only about two or three significant digits.

# 2 The algorithms

There are several possibilities to analyse the Bateman equations. First, it can be tried to work out the exact solution of the Bateman equations, which contains the exponential of a matrix. For this purpose, the scaling and squaring algorithm will be examined and also the Chebyshev Rational Approximation method (CRAM). Another option is to solve the Bateman equations numerically using the RadauIIA method. This is the method which is currently used at the SCK-CEN to analyse the Bateman equations. Though, the RadauIIA method doesn't take into account the sparsity of the system which constitutes a disadvantage.

## 2.1 The exact solution

The first method which will be inspected to solve the Bateman equations, makes use of the exact solution of the Bateman equations namely $e^{Bt}n_0$. This can be calculated by first examining the matrix exponential and thereafter multiplying it with the initial concentration vector. Another way is to combine the computation of the matrix exponential with the vector multiplication.

### 2.1.1 Compute the matrix exponential

There are many ways to calculate the exponential of a matrix, usually based on one of the formulae summarized in table 2.

| Power series | Limit | Scaling and squaring |
|---|---|---|
| $I + M + \frac{M^2}{2!} + \frac{M^3}{3!} + \cdots$ | $\lim_{l\to\infty} \left(I + \frac{M}{l}\right)^l$ | $\left(e^{\frac{M}{2^l}}\right)^{2^l}$ |
| Cauchy integral | Jordan form | Interpolation |
| $\frac{1}{2\pi i}\int_\Gamma e^z(zI - M)^{-1}dz$ | $Zdiag(e^{J_k})Z^{-1}$ | $\sum_{i=1}^{n} f[\lambda_1, \ldots, \lambda_i] \prod_{j=1}^{i-1}(M - \lambda_j I)$ |
| Differential system | Schur form | Padé approximation |
| $Y'(t) = MY(t), Y(0) = I$ | $Qdiag(e^T)Q^*$ | $p_{kl}(M)q_{kl}(M)^{-1}$ |

Table 2: Different formulas for $e^M$ [1]

In this paragraph, the scaling and squaring algorithm will be explained based on papers of N.J. Higham[6],[7]. The scaling and squaring algorithm uses the property that $\left(e^{\frac{M}{\omega}}\right)^{\omega} = e^M$, for $M \in \mathbb{C}^{mxm}$ and $\omega \in \mathbb{C}$. In addition, it exploits the fact that the exponential of a matrix can be approximated by a Padé approximation for small norm of the matrix. A Padé approximation is defined in the following definition.

**Definition 2.1.1.1** (Padé approximation[6])

*For a given scalar function f(x), the rational function* $r_{\tilde{k}\tilde{m}}(x)=\frac{p_{\tilde{k}\tilde{m}}(x)}{q_{\tilde{k}\tilde{m}}(x)}$ *is a Padé approximation of f if*

- $p_{\tilde{k}\tilde{m}}$ *is a polynomial of degree at most* $\tilde{k}$
- $q_{\tilde{k}\tilde{m}}$ *is a polynomial of degree at most* $\tilde{m}$
- $q_{\tilde{k}\tilde{m}}(0) = 1$
- $f(x) - r_{\tilde{k}\tilde{m}}(x) = O(x^{\tilde{m}+\tilde{k}+1})$ .

One often chooses $\tilde{k}$ equals to $\tilde{m}$, in the scaling and squaring method. The reason for this is that it is more accurate than when $\tilde{k}$ isn't equal to $\tilde{m}$ and it can be computed at the same matrix cost. If $\tilde{k}$ is chosen equal to $\tilde{m}$, the Padé approximation will be noted by $r_{\tilde{m}}(x)$ instead of $r_{\tilde{m}\tilde{m}}(x)$. How does the scaling and squaring method work?

The scaling and squaring method first determines the value of $\tilde{s}$ such that $\frac{M}{2^{\tilde{s}}}$ has sufficient small norm. Hereafter, the Padé approximation of $r_{\tilde{m}}\left(\frac{M}{2^{\tilde{s}}}\right)$ is calculated. Finally, the approximation $e^M \approx r_{\tilde{m}}\left(\frac{M}{2^{\tilde{s}}}\right)^{2^{\tilde{s}}}$ will be achieved by squaring $r_{\tilde{m}}\left(\frac{M}{2^{\tilde{s}}}\right)$ $\tilde{s}$ times. The goal is to choose the parameters $\tilde{s}$ and $\tilde{m}$ so that the backward error is bounded by specified tolerances and at a minimal cost. The analysis of finding the parameters can be found in the paper of N.J. Higham[7].
Concerning the Bateman equations, the exponential of the Bateman matrix $Bt$ will be calculated first. After this, they multiply the result with the initial concentration vector.
Since the system of Bateman equations can be very large and is typically very sparse, another approach to find the solution of the Bateman equations is explained in the next section. This method combines the computation of the matrix exponential and the multiplying with the initial concentration vector.

### 2.1.2 Combine the action of the matrix exponential on a vector

The method discussed in this section, based on a paper of N.J. Higham[8], avoids the explicit formation of $e^{Bt}$. It uses the property that

$$e^M C = \left(e^{\tilde{s}^{-1}M}\right)^{\tilde{s}} C = e^{\tilde{s}^{-1}M}.e^{\tilde{s}^{-1}M}\dots e^{\tilde{s}^{-1}M}C, \quad \text{with}\, M \in \mathbb{C}^{mxm} \text{and}\, C \in \mathbb{C}^{mx\hat{m}}, \hat{m} \ll m. \tag{2.1.1}$$

In the case of the Bateman equation $\hat{m}$ is equal to one. The factor $e^{\tilde{s}^{-1}M}$ will be estimated by a truncated Taylor series $e^{\tilde{s}^{-1}M} \approx T_{\tilde{m}}(\tilde{s}^{-1}M) = \sum_{j=0}^{\tilde{m}} \frac{\left(\tilde{s}^{-1}M\right)^j}{j!}$. It isn't approximated by a rational function, because a truncated Taylor series avoids having to solve linear systems. The choice of parameters $\tilde{s}$ and $\tilde{m}$ will be done in the same way as the choice of the parameters to approach the matrix exponential in the previous paragraph. After the factor $e^{\tilde{s}^{-1}M}$ is estimated, the recurrence

$$C_{i+1} = T_{\tilde{m}}(\tilde{s}^{-1}M)C_i\,, \qquad i = 0, \dots, \tilde{s}-1 \,\text{and}\, C_0 = C \tag{2.1.2}$$

returns the approximation $C_{\tilde{s}} \approx e^M C$.

### 2.1.3 Chebyshev Rational Approximation method

Now, the Chebyshev Rational Approximation method (CRAM) will be discussed. The text in this section is based on a paper of M. Pusa and J. Lippänen[9]. CRAM uses a unique rational function $\hat{g}_{k,k}(x)$ to approximate the exponential function $e^x$. This rational function needs to satisfy the expression

$$\sup_{x\in\mathbb{R}^-} |\hat{g}_{k,k}(x) - e^x| = \inf_{g^*_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}^-} |g^*_{k,k}(x) - e^x| \right\}. \tag{2.1.3}$$

Hereby is $\pi_{k,k}$ the set of rational functions $g^*_{k,k}(x) = \frac{p_k(x)}{q_k(x)}$ and $p_k$, $q_k$ are polynomials of order k.
Due to numerical reasons, a partial fraction decomposition form will be used to determine this rational function. A partial fraction decomposition for the rational function $g^*_{k,k}$ looks like

$$g^*_{k,k}(x) = \alpha_0 + \sum_{j=1}^{k} \frac{\alpha_j}{x-\theta_j} \tag{2.1.4}$$

where $\alpha_0$ is the limit of the function $g^*_{k,k}$ at infinity and $\alpha_j$ are the residues at the poles $\theta_j$.
The coefficients of this partial fraction decomposition form are fixed for every CRAM. A property of a rational function with real-valued coefficients is that the poles of it are conjugated pairs. Based on this property, the expression (2.1.4) can be rewritten as

$$g^*_{k,k}(x) = \alpha_0 + 2\Re\left(\sum_{j=1}^{\frac{k}{2}} \frac{\alpha_j}{x-\theta_j}\right), \tag{2.1.5}$$

were $\Re$ represents the real part of a complex number.

This allows to lessen the computational cost. By applying this to the Bateman equations, the formal solution $e^{Bt}n_0$ of it, can be approximated by the representation

$$n = \alpha_0\, n_0 + 2\Re\left(\sum_{j=1}^{\frac{k}{2}} \alpha_j (Bt - \theta_j I)^{-1} n_0\right). \tag{2.1.6}$$

This results in solving $\frac{k}{2}$ linear equations of the form

$$(Bt - \theta_j I)x_j = \alpha_j\, n_0 \tag{2.1.7}$$

for finding the solution of the Bateman equations based on a Rational CRAM of order k.
A possible method to solve this linear system is to use a sparse Gaussian elimination. A sparse Gaussian elimination exploits the sparsity pattern of the matrices by first computing the symbolic $LU$ factorisation. This factorisation determines the nonzero structure of the resulting upper triangular matrix in advance and makes the numerical elimination more efficient. However, it should be mentioned that the code used for this master thesis, doesn't utilize a symbolic $LU$ factorisation. It makes use of a sparse method, contributed by SciPy (Scientific Computing Tools for Python)[10], to solve the system.

## 2.2 The RadauIIA Method

After regarding the methods which solve the Bateman equations by approximating the formal solution, a method which solves the Bateman equations numerically will be considered in this section. This method is the RadauIIA method.

### 2.2.1 Introduction

RadauIIA method is an implicit Runge-Kutta method. Runge-Kutta methods are used to solve problems of the form

$$\begin{aligned} \frac{dy}{dt} &= f(t, y(t)) \\ y(t_0) &= y_0\,. \end{aligned} \tag{2.2.1}$$

For the Bateman equations, the function $f(t, y(t))$ is the Bateman matrix multiplied with the vector $y(t)$ , the vector $y$ is noted by $n$ and $t_0$ is often zero. Runge-Kutta methods are defined by the formula

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, g_i) \tag{2.2.2}$$

with

$$g_i = y_n + h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, g_j), \qquad i = 1, \ldots, s\,. \tag{2.2.3}$$

Hereby $s$ denotes the number of stages which indicates the number of function evaluations needed. RadauIIA is a three stage method and it has order of accuracy five. The order of accuracy gives an indication of the rate of convergence to the exact solution. If the round-off errors aren't taken into account, the approximated solution will converge faster to the exact solution as the order of the method gets higher. It can be noticed that if $m$ is the dimension of the system of differential equations (2.2.1) and $s$ is the stage number of the Runge-Kutta method, then the formula (2.2.3) has $m \cdot s$ nonlinear equation with unknowns $g_1, \ldots, g_s$. For a Runge-Kutta method the coefficients of the formula (2.2.2) and the formula (2.2.3) are generally presented in a Butcher tableau. A Butcher tableau has the form

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & & & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array}\,.$$

In particular for the RadauIIA method, the Butcher tableau is

$$\begin{array}{c|ccc} \frac{4-\sqrt{6}}{10} & \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{4+\sqrt{6}}{10} & \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ 1 & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \\ \hline & \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{array} .$$

The matrix formed by the coefficients $a_{ij}$, $i = 1, \ldots, s$ and $j = 1, \ldots, s$, will be referred to as the matrix $A$. So the matrix $A$ of the RadauIIA method is

$$\begin{bmatrix} \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{bmatrix} .$$

An important feature of the RadauIIA method is that it is L-stable. This means that it is A-stable, which will be explained further, and satisfies to

$$\lim_{z \to \infty} R(z) = 0, \tag{2.2.4}$$

where $R(z)$ is the stablity function of the method.

The stability function $R(z)$ of the method can be explained as follows. If a Runge-Kutta method, defined by the formula (2.2.2), is applied to the problem

$$y' = \lambda y, \quad \lambda \in \mathbb{C}, \quad \Re(\lambda) < 0 \tag{2.2.5}$$

there will be obtained a 1-step difference equation of the form

$$y_{n+1} = R(\hat{h}) y_n, \quad \hat{h} = h\lambda. \tag{2.2.6}$$

The function $R(\hat{h})$ is called the stability function of the method[11]. The region in $\mathbb{C}$ where $|R(z)| \leqslant 1$ is called the stability region or domain. A method is called A-stable if it contains the stability region $\mathbb{C}^-$. This can be interpreted as the guarantee of the method that the numerical solution of the problem

$$\begin{aligned} y' &= \lambda y \\ y(0) &= 1 \end{aligned} \tag{2.2.7}$$

with $\Re(\lambda) < 0$, converges to the exact solution zero if t tends to infinity.
The RadauIIA method is implemented in RADAU5 solver of E. Hairer and G.Wanner[12]. Details of the implementation are given in the following sections.

### 2.2.2 Deriving the system to be solved

First of all, the formula (2.2.3) can be rewritten as

$$z_i = g_i - y_n \tag{2.2.8}$$

where

$$z_i = h \sum_{j=1}^{s} a_{ij} f(t_n + c_j h, y_n + z_j). \tag{2.2.9}$$

This is done to reduce the round-off errors. The equation (2.2.9) can be reformulated in vector form as

$$\begin{bmatrix} z_1 \\ \vdots \\ z_s \end{bmatrix} = A \begin{bmatrix} hf(t_n + c_1 h, y_n + z_1) \\ \vdots \\ hf(t_n + c_s h, y_n + z_s) \end{bmatrix} . \tag{2.2.10}$$

This is possible if the matrix $A$ is non-singular, like it is the case for the RadauIIA method. If the equation (2.2.10) is rewritten as

$$A^{-1}\begin{bmatrix} z_1 \\ \vdots \\ z_s \end{bmatrix} = \begin{bmatrix} hf(t_n + c_1 h, y_n + z_1) \\ \vdots \\ hf(t_n + c_s h, y_n + z_s) \end{bmatrix}, \tag{2.2.11}$$

it can be seen that the Runge-Kutta formula (2.2.2) can also be formulated as

$$y_{n+1} = y_n + \sum_{i=1}^{s} d_i z_i \tag{2.2.12}$$

where
$$(d_1, \ldots, d_s) = (b_1, \ldots, b_s)A^{-1}. \tag{2.2.13}$$

For the RadauIIA method, the vector $d$ is given by (0,0,1). This is because $b_i = a_{3i}$, $i = 1, 2, 3$.
To solve the system (2.2.10), the simplified Newton-iteration method is used. The Newton-iteration solves equations of the form

$$F(y) = 0, \qquad F : \mathbb{R}^m \to \mathbb{R}^m. \tag{2.2.14}$$

The iteration is defined as

$$y^{[\nu+1]} = y^{[\nu]} - J^{-1}(y^{[\nu]})F(y^{[\nu]}), \qquad \nu = 0, 1, 2, \ldots \tag{2.2.15}$$

where
$$J(y) = \frac{\partial F}{\partial y}(y) \quad \text{is the Jacobian of F with respect to y } [11]. \tag{2.2.16}$$

Equation (2.2.15) is often rewritten as

$$J(y^{[v]})(y^{[v+1]} - y^{[v]}) = -F(y^{[v]})\,. \tag{2.2.17}$$

In the case of an equation of the form $y - \varphi(y) = 0$, it will become

$$\left(I_m - \frac{\partial \varphi}{\partial y}(y^{[\nu]})\right)(y^{[\nu+1]} - y^{[\nu]}) = -(y^{[\nu]} - \varphi(y^{[\nu]}), \qquad \nu = 0, 1, 2, \ldots\ . \tag{2.2.18}$$

In order to apply this to the system (2.2.10), the following notation is used

- $Z^k = \left(z_1^k, \ldots, z_s^k\right)^T$ is the k$^{\text{th}}$ approximation to the solution
- $\Delta Z^k = \left(\Delta z_1^k, \ldots, \Delta z_s^k\right)^T = \left(z_1^{k+1} - z_1^k, \ldots, z_s^{k+1} - z_s^k\right)^T$ represents the increments
- $F(Z^k) = \left(f(t_n + c_1 h, y_n + z_1^k), \ldots, f(t_n + c_s h, y_n + z_s^k)\right)^T$.

Using this notation, the next system has to be solved

$$\begin{aligned} (I - hA \otimes J)\Delta Z^k &= -Z^k + h(A \otimes I)F(Z^k) \\ Z^{k+1} &= Z^k + \Delta Z^k\,. \end{aligned} \tag{2.2.19}$$

To solve this system, the first equation will be premultiplied by $(hA)^{-1} \otimes I$. This is achievable for the RadauIIA method, since the matrix $A$ is non-singular. Hence, the first equation of the system (2.2.19) becomes

$$((hA)^{-1} \otimes I)(I - hA \otimes J)\Delta Z^k = -((hA)^{-1} \otimes I)Z^k + h((hA)^{-1} \otimes I)(A \otimes I)F(Z^k) \tag{2.2.20}$$

or
$$((hA)^{-1} \otimes I) - (I \otimes J)\Delta Z^k = -((hA)^{-1} \otimes I)Z^k + F(Z^k).$$

Hereafter, the inverse matrix $A^{-1}$, is transformed to

$$T^{-1}A^{-1}T = \Lambda \ ,$$

with $\Lambda$ a simpler matrix such as a diagonal matrix or a block diagonal matrix.
Considering the RadauIIA method, the matrix $A^{-1}$ has one real eigenvalue, which will be denoted by $\hat{\gamma}$ and one complex conjugate eigenvalue pair $\hat{\alpha} \pm i\hat{\beta}$. Therefor, $\Lambda$ is

$$\begin{bmatrix} \hat{\gamma} & 0 & 0 \\ 0 & \hat{\alpha} & -\hat{\beta} \\ 0 & \hat{\beta} & \hat{\alpha} \end{bmatrix}. \tag{2.2.21}$$

Especially, $\hat{\gamma} = 3.63783$ , $\hat{\alpha} = 2.68108$ and $\hat{\beta} = 3.05043$. To find the matrix $T$, denote the eigenvectors of $A^{-1}$ by $v_1$, $v_2$ and $v_3$. More specific, $v_1 = (0.09123, 0.24172, 0.96605)^T$, $v_2 = (-0.14126 + 0.03003i, 0.20413 - 0.38294i, 1)^T$ and $v_3 = (-0.14126 - 0.03003i, 0.20413 + 0.38294i, 1)^T$. Then is $T$ defined as

$$T = \begin{pmatrix} v_1 & \frac{v_2+v_3}{2} & \frac{v_2-v_3}{2} \end{pmatrix} .$$

Thus the following decomposition of $A^{-1}$ is achieved for the RadauIIA method

$$\begin{bmatrix} 4.32558 & 0.33919 & 0.54177 \\ -4.17872 & -0.32769 & 0.47662 \\ -0.50288 & 2.57193 & -0.59604 \end{bmatrix} \cdot \begin{bmatrix} \frac{88-7\sqrt{6}}{360} & \frac{296-169\sqrt{6}}{1800} & \frac{-2+3\sqrt{6}}{225} \\ \frac{296+169\sqrt{6}}{1800} & \frac{88+7\sqrt{6}}{360} & \frac{-2-3\sqrt{6}}{225} \\ \frac{16-\sqrt{6}}{36} & \frac{16+\sqrt{6}}{36} & \frac{1}{9} \end{bmatrix} \cdot \begin{bmatrix} 0.09123 & -0.14126 & -0.03003 \\ 0.24172 & 0.20413 & 0.38294 \\ 0.96605 & 1 & 0 \end{bmatrix}$$
$$= \begin{bmatrix} 3.63783 & 0 & 0 \\ 0 & 2.68108 & 3.05043 \\ 0 & 3.05043 & 2.68108 \end{bmatrix}. \tag{2.2.22}$$

By using also the notation $W^k = (T^{-1} \otimes I)Z^k$, the system (2.2.19) is converted to

$$\begin{aligned} (h^{-1}\Lambda \otimes I - I \otimes J)\Delta W^k &= -h^{-1}(\Lambda \otimes I)W^k + (T^{-1} \otimes I)F((T \otimes I)W^k) \\ W^{k+1} &= W^k + \Delta W^k. \end{aligned} \tag{2.2.23}$$

With the notation $\gamma = h^{-1}\hat{\gamma}$, $\alpha = h^{-1}\hat{\alpha}$ and $\beta = h^{-1}\hat{\beta}$, the matrix $h^{-1}\Lambda \otimes I - I \otimes J$ can be written as

$$\begin{bmatrix} \gamma I - J & 0 & 0 \\ 0 & \alpha I - J & -\beta I \\ 0 & \beta I & \alpha I - J \end{bmatrix}. \tag{2.2.24}$$

This allows to split the system (2.2.23) into two linear systems, one with dimension $m$ and the other with dimension $2m$. To exploit the special structure of the $2m$-dimensional subsystem, it is transformed to a complex system of dimension $m$. This is done as follows. The $2m$-dimensional subsystem, which needs to be solved, has the form

$$\begin{bmatrix} \alpha I - J & -\beta I \\ \beta I & \alpha I - J \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}. \tag{2.2.25}$$

Multiplying the second equation with $i$ gives

$$\begin{bmatrix} \alpha I - J & -\beta I \\ i\beta I & i(\alpha I - J) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ ib \end{bmatrix}. \tag{2.2.26}$$

So, the following equation is accomplished

$$(\alpha I - J)u - \beta Iv + i\beta Iu + i(\alpha I - J)v = a + ib \tag{2.2.27}$$

which gives

$$(\alpha + i\beta)Iu - Ju + (\alpha + i\beta)Iv - iJv = a + ib \tag{2.2.28}$$

or

$$((\alpha + \beta i)I - J)\,(u + iv) = a + ib.$$

On this system, a Gaussian elimination is then applied.

### 2.2.3 Starting values

A common choice for the starting values in the simplified Newton iteration is $z_i^0 = 0$, $i = 1, \ldots, s$. Nevertheless, better starting values are mostly possible. The choice in the implementation of RadauIIA method in the RADAU5 solver of E. Hairer and G. Wanner[12], will be discussed in this paragraph. It exploits the fact that the RadauIIA method satisfies the condition $C(s)$, which is defined as follows

$$C(\eta): \qquad \sum_{j=1}^{s} a_{ij} c_j^{q-1} = \frac{c_i^q}{q} \qquad i = 1, \ldots, s, \quad q = 1, \ldots, \eta\,. \tag{2.2.29}$$

This condition, together with conditions $B(p)$ and $D(\zeta)$ given by

$$B(p): \qquad \sum_{i=1}^{s} b_i c_i^{q-1} = \frac{1}{q} \qquad q = 1, \ldots, p \tag{2.2.30}$$

$$D(\zeta): \qquad \sum_{i=1}^{s} b_i c_i^{q-1} a_{ij} = \frac{b_j}{q}(1 - c_j^q) \qquad j = 1, \ldots, s, \quad q = 1, \ldots, \zeta\,, \tag{2.2.31}$$

are used to construct implicit Runge-Kutta methods with good stability properties.
Furthermore, if condition $C(\eta)$ is fulfilled, it can be shown that $\mathcal{L}_i$=O($h^{\eta+1}$) holds, whereby the functional $\mathcal{L}_i$ is defined as

$$\mathcal{L}_i := y(t_n + c_i h) - y(t_n) - h \sum_{j=1}^{s} a_{ij} y'(t_n + c_j h)\,. \tag{2.2.32}$$

This results from the following derivation

$$\begin{aligned}
\mathcal{L}_i &= y(t_n + c_i h) - y(t_n) - h \sum_{j=1}^{s} a_{ij} y'(t_n + c_j h) \\
&= \sum_{k=1}^{\infty} \frac{(c_i h)^k}{k!} y^{(k)}(t_n) - h \sum_{j=1}^{s} a_{ij} \left( \sum_{l=0}^{\infty} \frac{(c_j h)^l}{l!} y^{(l+1)}(t_n) \right) \\
&= \sum_{k=1}^{\infty} \frac{(c_i h)^k}{k!} y^{(k)}(t_n) - \sum_{j=1}^{s} a_{ij} \left( \sum_{k=1}^{\infty} \frac{c_j^{k-1} h^k}{(k-1)!} y^{(k)}(t_n) \right) \\
&= \sum_{k=1}^{\infty} \left( \frac{c_i^k}{k!} - \sum_{j=1}^{s} a_{ij} \frac{c_j^{k-1}}{(k-1)!} \right) h^k y^k(t_n) \\
&= \sum_{k=1}^{\infty} \frac{1}{(k-1)!} \left( \frac{c_i^k}{k} - \sum_{j=1}^{s} a_{ij} c_j^{k-1} \right) h^k y^{(k)}(t_n)\,.
\end{aligned}$$

The RadauIIA method fulfils the condition $C(s)$ because it is the reflection or adjoint method of the RadauI method which is constructed such that it complies with $C(s)$.

The reflection of a Runge-Kutta method defined by formula (2.2.2), is the method that corresponds with the Butcher tableau[11]

$$\begin{array}{c|cccc} 1-c_s & b_s-a_{ss} & b_{s-1}-a_{s\,s-1} & \cdots & b_1-a_{s1} \\ 1-c_{s-1} & b_s-a_{s-1\,s} & b_{s-1}-a_{s-1\,s-1} & \cdots & b_1-a_{s-1\,1} \\ \vdots & \vdots & & & \vdots \\ 1-c_1 & b_s-a_{1s} & b_{s-1}-a_{1\,s-1} & \cdots & b_1-a_{11} \\ \hline & b_s & b_{s-1} & \cdots & b_1 \end{array} .$$

This reflection method exactly reverses the work of the given Runge-Kutta method.
Moreover, one can prove that if

$$z_i - y(t_n + c_i h) + y_n = O(h^r) \quad \text{and} \quad r \leqslant \eta \tag{2.2.33}$$

then

$$z_i - y(t_n + c_i h) + y_n = O(h^{r+1})\,. \tag{2.2.34}$$

This is because of the following derivation, using the definitions of $z_i$ (2.2.9) and $\frac{dy}{dt} = f(t, y(t))$ (2.2.1),

$$\begin{aligned} z_i &= h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, y_n + z_j) \\ &= h\sum_{j=1}^{s} a_{ij} f\left(t_n + c_j h, y(t_n + c_j h) + (z_j - y(t_n + c_j h) + y_n)\right) \\ &= h\sum_{j=1}^{s} a_{ij} \left( f(t_n + c_j h, y(t_n + c_j h)) + (z_j - y(t_n + c_j h) + y_n) \frac{\partial f}{\partial y}(t_n + c_j h, \xi_j) \right) \\ &= h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, y(t_n + c_j h)) + h\sum_{j=1}^{s} a_{ij} (z_j - y(t_n + c_j h) + y_n) \frac{\partial f}{\partial y}(t_n + c_j h, \xi_j) \\ &= h\sum_{j=1}^{s} a_{ij} y'(t_n + c_j h) + \mathrm{O}(h^{r+1}) \\ &= h\sum_{j=1}^{s} a_{ij} \left( \sum_{l=0}^{r-1} \frac{(c_j h)^l}{l!} y^{(l+1)}(t_n) \right) + \mathrm{O}(h^{r+1}) \\ &= \sum_{l=0}^{r-1} \left( \sum_{j=1}^{s} a_{ij} \frac{c_j^l}{l!} \right) h^{l+1} y^{(l+1)}(t_n) + \mathrm{O}(h^{r+1}) \\ &= \sum_{l=0}^{r-1} \frac{(c_i h)^{l+1}}{(l+1)!} + \mathrm{O}(h^{r+1}) \\ &= y(t_n + c_i h) - y_n + \mathrm{O}(h^{r+1}) \end{aligned} \tag{2.2.35}$$

for $min(y(t_n + c_j h), y_n + z_j) \leqslant \xi_j \leqslant max(y(t_n + c_j h), y_n + z_j)$, $j = 1, \ldots, s$.
Therefore, it has been achieved that if an implicit Runge-Kutta method satisfies condition $C(\eta)$ for $\eta \leqslant s$, then

$$z_i = y(t_n + c_i h) - y_n + O(h^{\eta+1})\,. \tag{2.2.36}$$

Assume now that $c_i \neq 0$, $i = 1, \ldots, s$ and consider the interpolation polynomial given by

$$q(0) = 0, \quad q(c_i) = z_i, \quad i = 1, \ldots, s. \tag{2.2.37}$$

This interpolation polynomial has degree $s$ and the interpolation error can be expressed by using theorem (2.2.3.1).

**Theorem 2.2.3.1** (Interpolation error[13])
*Let f($\hat{x}$) be defined on a closed interval $[a,b]$ and n+1 times continuously differentiable on $[a,b]$.*
*Consider $\hat{x}_i \in [a,b]$, $i = 1,\ldots,n$.*
*Let $f_i = f(\hat{x}_i)$, $i = 1,\ldots,n$ and let $p_n(\hat{x})$ be the interpolating polynomial of degree n that interpolates f at $\hat{x}_i$, i=1,...,n.*
*Then, for each $\hat{x} \in [a,b]$, there exists $\xi \in (a,b)$ such that the interpolation error*

$$E_n(\hat{x}) := f(\hat{x}) - p_n(\hat{x})$$

*is given by*

$$E_n(\hat{x}) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(\hat{x}-\hat{x}_0)(\hat{x}-\hat{x}_1)\ldots(\hat{x}-\hat{x}_n)\,. \tag{2.2.38}$$

If $q$ interpolates a function $f$ at $z = 0, c_1, \ldots, c_s$, the interpolation error can be written as follows

$$\begin{aligned} f(\hat{x}) - q(\hat{x}) &= \frac{\hat{x}\,(\hat{x}-c_1)\,(\hat{x}-c_2)\ldots(\hat{x}-c_s)}{(s+1)!} f^{(s+1)}(\xi) \\ &= u(\hat{x}) f^{(s+1)}(\xi)\,. \end{aligned}$$

Now consider

$$f(\hat{x}) = y(t_n + \hat{x}h) - y_n + \mathrm{O}(h^{\eta+1}) \tag{2.2.39}$$

and taking the derivative of this function gives

$$\begin{aligned} f'(\hat{x}) &= \frac{dy}{dx}(t_n+\hat{x}h)\frac{dx}{d\hat{x}} + \mathrm{O}(h^{\eta+1}) \\ &= hy'(t_n+\hat{x}h) + \mathrm{O}(h^{\eta+1})\,. \end{aligned} \tag{2.2.40}$$

From this, it can be seen that the $s^{th}$ derivative of the function $f$, can be expressed as follows

$$f^{(s+1)}(\hat{x}) = h^{s+1} y^{(s+1)}(t_n+\hat{x}h) + \mathrm{O}(h^{\eta+1})\,. \tag{2.2.41}$$

Hence is

$$f(\hat{x}) - q(\hat{x}) = y(t_n+\hat{x}h) - y_n + \mathrm{O}(h^{\eta+1}) - q(\hat{x}) = \mathrm{O}(h^{s+1}) + \mathrm{O}(h^{\eta+1}). \tag{2.2.42}$$

Since $\eta \leqslant$s, it follows that

$$y(t_n+\hat{x}h) - y_n - q(\hat{x}) = O(h^{\eta+1}). \tag{2.2.43}$$

This means that $q(\hat{x})$ is sufficiently good to approximate $y(t_n+\hat{x}h) - y_n$.
Consider now figure 15. Figure 15 shows the polynomial $q(\hat{x})$ which interpolates $z$ in the points $\hat{x} = c_i$, $i = 1,...,s$ and $\hat{x} = 0$ within the interval $\hat{x} \in [0,1]$, in other words

$$q(0) = 0, \quad q(c_1) = z_1, \quad q(c_2) = z_2 \quad \text{and} \quad q(c_3) = z_3\,. \tag{2.2.44}$$

These points are indicated with black dots in figure 15. Since $c_3 = 1$, it holds that

$$q(1) = q(c_3) = z_3 = g_3 - y_n = y_{n+1} - y_n\,. \tag{2.2.45}$$

If $y_n$ is added to $q(1)$, an approximation of $y_{n+1}$ is achieved. This is represented in figure 15 by the curve $q(\hat{x}) + y_n = q(x+1) + y_n$, whereby $x = \hat{x} - 1$. To get the new starting values of $z$, $y_{n+1}$ has to be subtracted from $q(x+1)+y_n$ and the interpolation polynomial $q$ has to be used outside the interval $\hat{x} \in [0,1]$. Therefore, the starting values for the simplified Newton iteration in the subsequent step are given by

$$z_i^n = q(1+wc_i) + y_n - y_{n+1}, \quad i = 1,..s, \quad w = \frac{h_{new}}{h_{old}}\,, \tag{2.2.46}$$

which are indicated as green dots in figure 15.

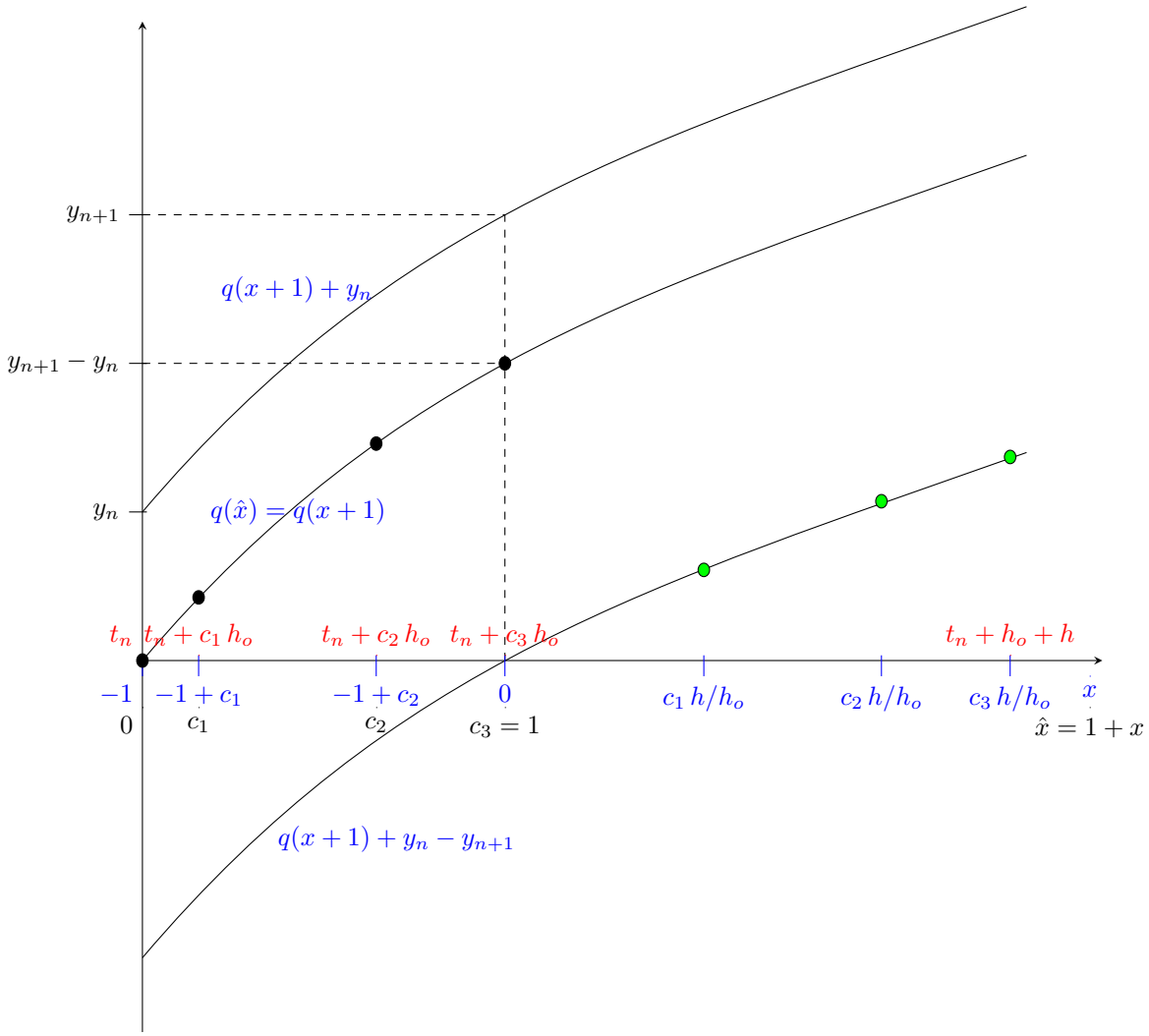


Figure 15: Starting values for the simplified Newton iteration in the subsequent step ($h = h_{new}$ and $h_o = h_{old}$)

In the implementation of RadauIIA method of E. Hairer and G. Wanner[12], the interpolation theory of Newton is used. To define the interpolation according to Newton, the divided differences will be introduced. The divided difference of order zero is defined as

$$f[\hat{x}_i] := f(\hat{x}_i) \tag{2.2.47}$$

and the divided difference of order one as

$$f[\hat{x}_i, \hat{x}_{i+1}] := \frac{f[\hat{x}_{i+1}] - f[\hat{x}_i]}{\hat{x}_{i+1} - \hat{x}_i}\,. \tag{2.2.48}$$

In general, the divided difference of order $j-1$ is given by the recursive forumula

$$f[\hat{x}_i, \hat{x}_{i+1}, \ldots, \hat{x}_j] := \frac{f[\hat{x}_{i+1}, \ldots, \hat{x}_j] - f[\hat{x}_i, \ldots, \hat{x}_{j-1}]}{\hat{x}_j - \hat{x}_i}\,. \tag{2.2.49}$$

Employing the divided differences, the interpolation polynomial $p_i$ of degree $i$ for $\{(\hat{x}_j, f(\hat{x}_j))\}_{j=0}^{i}$, according to Newton, is defined as

$$p_i(\hat{x}) = b_0 + b_1(\hat{x} - \hat{x}_0) + \ldots + b_i(\hat{x} - \hat{x}_0)\ldots(\hat{x} - \hat{x}_{i-1}) \tag{2.2.50}$$

with $b_i = f[\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_i]$[13].

### 2.2.4 Stopping criterium

Since the simplified Newton iteration computes consecutive approximated solutions of the system of equations, a criterium is needed to know when to end the iteration. Based on the book of E. Hairer and G. Wanner[12], the following control is done. After at least two iterations, the factor $\theta_k = \dfrac{\|\Delta Z^k\|}{\|\Delta Z^{k-1}\|}$ is calculated. This factor is an estimate of the convergence rate. The reason herefor is that the convergence of the simplified Newton iteration is linear and hence the following expression holds

$$\|\Delta Z^{k+1}\| \leqslant \theta \|\Delta Z^k\| \,.$$

The factor $\theta$ is hopefully smaller than one, so that the iteration converges.
Let be $Z^*$ the exact solution of (2.2.1). By using the triangle inequality to

$$Z^{k+1} - Z^* = (Z^{k+1} - Z^{k+2}) + (Z^{k+2} - Z^{k+3}) + \dots \quad , \tag{2.2.51}$$

it has been achieved that

$$\begin{aligned}
\|\Delta Z^{k+1} - Z^*\| &\leqslant \|Z^{k+1} - Z^{k+2}\| + \|Z^{k+2} - Z^{k+3}\| + \dots \\
&\leqslant \|\Delta Z^{k+1}\| + \|\Delta Z^{k+2}\| + \dots \\
&\leqslant \theta\|\Delta Z^k\| + \theta\|\Delta Z^{k+1}\| + \dots \\
&\leqslant \theta\|\Delta Z^k\| + \theta^2\|\Delta Z^k\| + \dots \\
&= (\theta + \theta^2 + \dots)\|\Delta Z^k\| \\
&= \frac{\theta}{1-\theta}\|\Delta Z^k\| \,.
\end{aligned}$$

Based on this, it can be concluded to end the iteration when

$$\frac{\theta_k}{1-\theta_k}\|\Delta Z^k\| \leqslant \kappa \cdot Tol \,. \tag{2.2.52}$$

A good choice for the value $\kappa$ is arround $10^{-1}$ or $10^{-2}$ [12] and Tol represents a predescribed tolerance.
The estimate of $\frac{\theta_k}{1-\theta_k}$ with $k$=0, namely the estimation for the first iteration, is

$$\frac{\theta_0}{1-\theta_0} = max\left(\frac{\theta_{old}}{1-\theta_{old}}, Uround\right)^{0.8} \tag{2.2.53}$$

where $\theta_{old}$ is the value of $\theta_k$ in the preceding step and Uround is the unit round-off.
To summarize the idea of the stopping criteria, the computation of the Newton iteration will be stopped and restarted with a smaller stepsize, when one of the following situations occur

- there exists a $k$ with $\theta_k \geqslant 1$, which means that the iteration "diverges"
- for some k, $\frac{\theta_k^{k_{max}-k}}{1-\theta_k}\|\Delta Z^k\| > \kappa \cdot Tol$ is fulfilled, with $k_{max}$ the maximum number of Newton iterations.

This holds as long as the maximum number of iterations hasn't been exceeded.
If the last $\theta_k$ was very small, say $\leqslant 10^{-3}$, or only one simplified Newton iteration was needed to satisfy (2.2.52), then the RADAU5 implementation doesn't recompute the Jacobian in the following step. Hence, the Jacobian is calculated only once for linear systems with constant coefficients.

### 2.2.5 Stepsize selection

There is no fixed stepsize in the RADAU5 solver. The following expression is used to predict the new stepsize

$$h_{new} = \frac{h_{old}}{max\left(facmin, min\left(facmax, \frac{1}{fac}\left(\frac{Tol}{\|Err\|}\right)^{\frac{1}{4}}\right)\right)}. \tag{2.2.54}$$

The explanation of the terms used in this expression is

- $h_{old}$ is the previous stepsize
- $h_{new}$ is the new stepsize
- $facmax$ and $facmin$ are maximal, respectively minimal, acceptable growth factors, $facmax$ is typically chosen to be 5 and $facmin$ as $\frac{1}{8}$
- Err is an error estimate, which will be explained in the next section
- Tol is a predescribed tolerance
- $fac = min\left(safe, safe \cdot \frac{1+2k_{max}}{k+k_{max}}\right)$
  Hereby denotes $k_{max}$ the maximum number of Newton iterations. Based on experience of E. Hairer and G. Wanner[12], is $k_{max}$ typically taken as 7 or 10. $k$ is the number of Newton iterations in the current step. The term safe is a safety factor in the stepsize prediction. Normally, it gets the value 0.9.

### 2.2.6 Error estimation

To estimate the error, an embedded pair of methods will be used. To explain an embedded pair of methods, consider a Runge-Kutta method defined by the coefficients $c$, $A$, $b^T$ and of order $p$. Hence, this method is of the form

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i f(t_n + c_i h, g_i) \tag{2.2.55}$$

with

$$g_i = y_n + h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, g_j), \qquad i = 1, \ldots, s\,. \tag{2.2.56}$$

The embedded method is a Runge-Kutta method of order $p+1$, but with coefficients $c$, $A$, $\hat{b}^T$. So, the only difference is the $b$ coefficients and the embedded method is given by

$$\hat{y}_{n+1} = y_n + h\sum_{i=1}^{s} \hat{b}_i f(t_n + c_i h, g_i) \tag{2.2.57}$$

with

$$g_i = y_n + h\sum_{j=1}^{s} a_{ij} f(t_n + c_j h, g_j), \qquad i = 1, \ldots, s\,. \tag{2.2.58}$$

The error will then be defined as

$$\hat{y}_{n+1} - y_{n+1} = h\sum_{i=1}^{s} (\hat{b}_i - b_i) f(t_n + c_i h, g_i)\,. \tag{2.2.59}$$

However, the RadauIIA method is already of optimal order. Therefore, it isn't possible to find a method of an order one higher. Hence, there has to be sought for a lower order method of the form

$$\hat{y}_{n+1} = y_n + h\left(\hat{b}_0 f(t_n, y_n) + \sum_{i=1}^{3} \hat{b}_i f(t_n + c_i h, g_i)\right)$$

and $\hat{b}_0 \neq 0$.
Based on the book of E. Hairer and G. Wanner[12], $\hat{b}_0$ is chosen to be equal to $\hat{\gamma}^{-1}$, with $\hat{\gamma}^{-1}$ the real eigenvalue of the matrix $A^{-1}$. This is done to save multiplications. Furthermore, there is chosen in the book of E. Hairer and G. Wanner[12] for an embedded method of order three. To find a Runge-Kutta method of order three, the coefficients have to satisfy to the so called order conditions[11].

These conditions are the following

$$\sum_i \hat{b}_i = 1 \tag{2.2.60}$$

$$\sum_i \hat{b}_i c_i = \frac{1}{2} \tag{2.2.61}$$

$$\sum_i \hat{b}_i c_i^2 = \frac{1}{3}\,. \tag{2.2.62}$$

Since the coefficients $c_i, i = 1, \ldots, 3$, are known from the RadauIIA method, the coefficients $\hat{b}_i, i = 1, \ldots, 3$, can be obtaind by solving a system of three equations with three unknowns. The coefficients $\hat{b}_i, i = 1, \ldots, 3$, of the third order embedded Runge-Kutta method are given by

$$\begin{aligned}
\hat{b}_1 &= -\frac{(3\sqrt{6}+2)(-\sqrt{6}+1+6\,\hat{\gamma}^{-1})}{36} \\
\hat{b}_2 &= \frac{(3\sqrt{6}-2)(\sqrt{6}+1+6\,\hat{\gamma}^{-1})}{36} \\
\hat{b}_3 &= \frac{1}{9} - \frac{\hat{\gamma}^{-1}}{3}\,.
\end{aligned}$$

The difference between the RadauIIA method and the embedded method will serve for the error estimation, which is given by

$$\hat{y}_{n+1} - y_{n+1} = \hat{\gamma}^{-1} h f(t_n, y_n) + h \sum_{i=1}^{3} (\hat{b}_i - b_i) f(t_n + c_i h, g_i)\,, \tag{2.2.63}$$

or rewritten in terms of $z_i$

$$\hat{y}_{n+1} - y_{n+1} = \hat{\gamma}^{-1} h f(t_n, y_n) + e_1 z_1 + e_2 z_2 + e_3 z_3\,. \tag{2.2.64}$$

Notice that for transforming to $z_i$, the difference $\hat{b} - b$ has to be multiplied with $A^{-1}$ to get the error estimation in terms of $z_i$. The terms $e_1, e_2$ and $e_3$ are

$$(e_1, e_2, e_3) = \frac{\hat{\gamma}^{-1}}{3}(-13 - 7\sqrt{6}, -13 + 7\sqrt{6}, -1)\,. \tag{2.2.65}$$

Since the embedded method is of order three, it holds

$$\hat{y}_{n+1} - y_{n+1} = \mathrm{O}(h^4)\,. \tag{2.2.66}$$

Nevertheless, when the problem $y' = \lambda y$ is considered, the difference will behave like $\hat{y}_{n+1} - y_{n+1} \approx \hat{\gamma}^{-1} h\lambda y_n$, which is unbounded as $h\lambda$ tends to infinity. Therefor, another error estimation is proposed in the book of E. Hairer and G. Wanner[12], which is

$$err = (I - h\hat{\gamma}^{-1} J)^{-1}(\hat{y}_{n+1} - y_{n+1}). \tag{2.2.67}$$

Since the $LU$ factorisation of $((h\hat{\gamma}^{-1})^{-1} I - J)$ has already been done for solving the system of equations (2.2.23), the computation of this error estimation is not expensive. It still holds that $err = \mathrm{O}(h^4)$ if $h \to 0$. For the problem $y' = \lambda y$ and $J = \lambda$, $err$ tends to $-1$ if $h\lambda$ approaches infinity.
As illustration of this, consider the problem

$$y' = -10^9 y,\ y(0) = 1 \text{ and end time } 6000000\,s. \tag{2.2.68}$$

In table 3, the norm of the difference (2.2.64) and the norm of the error estimation (2.2.67) of this problem for different moments in the time process can be found. The norm used in the implementation of the RadauIIA method was taken.

This norm is defined as

$$\|error\| = max\left(\sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{error_i}{sc_i}\right)^2}, 1\cdot 10^{-10}\right), \tag{2.2.69}$$

with $sc_i = AbsTol_i + max(|y_{n,i}|, |y_{n+1,i}|) \cdot RelTol_i$. Hereby stands $AbsTol$ for the predescribed absolute tolerance and $RelTol$ for the predescribed relative tolerance. They can be both scalars or both vectors of length N.
From table 3, it can be noticed that in the beginning of the time evolution the norm of the difference (2.2.64) is bigger than the norm of the error estimation (2.2.67). At the end, both norms become small.

| **The moment in the process of time (s)** | $\|\hat{\boldsymbol{y}}_{\boldsymbol{n+1}} - \boldsymbol{y}_{\boldsymbol{n+1}}\|$ | $\|\boldsymbol{err}\|$ |
|---|---|---|
| 100.0 s | $2.32079 \cdot 10^{8}$ | $2.32079 \cdot 10^{3}$ |
| 900.0 s | $1.39260 \cdot 10^{2}$ | $1.39260 \cdot 10^{-8}$ |
| 7300.0 s | $3.62641 \cdot 10^{1}$ | $3.62641 \cdot 10^{-9}$ |
| 58500.0 s | $1.11471 \cdot 10^{-10}$ | $1 \cdot 10^{-10}$ |
| 3744900.0 s | $1 \cdot 10^{-10}$ | $1 \cdot 10^{-10}$ |
| 6000000.0 s | $1 \cdot 10^{-10}$ | $1 \cdot 10^{-10}$ |

Table 3: The norm of the difference (2.2.64) and the norm of (2.2.67) for the problem (2.2.68)

Although the error estimation (2.2.67) is already much better for the problem $y' = \lambda y$ when $h\lambda$ approaches infinity, there is mentioned an additional error estimation in the book of E. Hairer and G. Wanner[12]. In the first step and after each rejected step for which $\|err\| > 1$, the following error estimation is used

$$\tilde{err} = (I - h\hat{\gamma}^{-1}J)^{-1}(\hat{\gamma}^{-1}hf(t_n, y_n + err) + e_1 z_1 + e_2 z_2 + e_3 z_3)\,. \tag{2.2.70}$$

This demands an extra function evaluation, but $\tilde{err}$ goes to zero when $h\lambda$ tends to infinity. This behaviour is the same as the error of the numerical solution.

# 3 Application/Implementation

In this part, the results of the different algorithms, discussed in section 2, are compared. These methods were

- the scaling and squaring algorithm to compute the matrix exponential
- the method that combines the action of the matrix exponential on a vector
- CRAM
- the RadauIIA method.

Hereafter, three methods which use a Padé approximation to the exponential are discussed. Finally, the modification of the implemented RADAU5 solver to sparse systems is considered.

## 3.1 Comparing the different algorithms

In this paragraph, the different algorithms, mentioned in section 2, will be analysed in function of the Bateman equations.

### 3.1.1 Implementation

To begin with, the implementation of the algorithms will be considered. It will be first explained for the Polonium problem and thereafter for the decay problem.

#### 3.1.1.1 The Polonium problem

The Polonium problem was first investigated. For this, the four different methods were implemented in Python 2.7.10[14].
Herefor, a laptop was used with the following specifications

- operating system: Windows 7 Ultimate (32 bits)
- processor: an Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz
- memory: 4 GB.

The scaling and squaring algorithm, which was explained based on papers of N.J. Higham[6],[7] (see section 2.1.1), is built in Python 2.7.10[14] with the command "linag.expm". To multiply the resulting matrix exponential with the initial concentration vector, the command "np.dot" was used. The code which implements the scaling and squaring algorithm can be found in Appendix B.1.1.
The function "linalg.expm.multiply" combines the action of computing the matrix exponential and multiplying with the initial concentration vector. In Appendix B.1.2, the implementation of this method is provided.
CRAM was implemented without making use of the symbolic $LU$ factorisation. To solve the system of equations, a sparse solver of linear systems build in Python was used. The command for this is "spsolve". The code which contains CRAM is given in Appendix B.1.3.
The solver RADAU5 of the RadauIIA method is created in Python through the Assimulo package[15]. Assimulo is a simulation package based on Python/Cython for solving ordinary differential equations. The code to evoke the RadauIIA method through the RADAU5 solver for solving the Polonium problem is presented in Appendix B.1.4.
To have an idea of time needed to solve the Bateman equations, the CPU (central processing unit) time, which is a measure of computing time, was computed. For the first three methods, not the RadauIIA method, the CPU time was measured with the command "time.clock". The function "time.clock" gives wall-clock seconds which passed after the first call to a function. The RadauIIA method has a built-in function to measure the time needed to solve the Bateman equations. Since the CPU time depends on possible other processes, the CPU time is calculated ten times whereafter the maximum and minimum time measurements are subtracted. At last, the average is taken of the remaining CPU times. This way of calculating the CPU time will be applied throughout this master thesis.

#### 3.1.1.2 The decay problem and the fresh fuel problem

After the analysis of the Polonium problem, the Python codes were changed to handle more general problems. The main modification was the way the matrix was read. The reformed codes can be found in Appendices B.2.1, B.2.2 and B.2.3. To run the code in Python 2.7.10, the same laptop was used as in the previous paragraph.
Thereafter, a code was built to evoke the RADAU5 solver[16], which contains the implementation of the RadauIIA method in GNU Fortan 4.6.3[18]. This was done because an objective of the master thesis is to change the original RADAU5 solver as implemented in Fortran, in order to take into account the sparsity of the Bateman equations. Several ways were tried to make it work properly. The difference between the implementations is the approach to multiply a matrix with a vector, which is needed for the right-hand side of the Bateman equations.
The first option was to write a code which implements a multiplication of matrix with a vector in the following way. The file, containing the Bateman matrix, was read line by line. Per line, an element $b_{ij}$ in a specific row $i$ and column $j$ of the Bateman matrix was multiplied with the corresponding element at position $j$ of the vector and after that it was added to the value of the resulting vector at position $i$. This way of multiplying a matrix with a vector was implemented in the code available in Appendix B.2.4.1 and will be mentioned as a standard multiplication of a matrix with a vector.
In Appendix B.2.4.2, another approach of implementing is given. This stores the Bateman matrix as Compressed Row Storage (CRS). CRS consists of three lists, namely

- the value list which contains the values of the elements in the matrix
- the column list which includes the corresponding column positions of the elements in the matrix
- the row list that contains elements so that the element at position $i$ indicates the first element of the $i^{\text{th}}$ row. If a matrix has $m$ rows, then the $(m+1)^{\text{th}}$ element, so the last one, of the value list holds the number of nonzero elements in the matrix increased by one.

For example, the matrix

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 0 & 5 \\ 25 & 0 & 0 & 0 \end{bmatrix} \tag{3.1.1}$$

would be saved using the following lists

- the value list is [4 1 2 1 1 5 25]
- the column list is [1 2 2 3 1 4 1]
- the row list is [1 3 5 7 8].

For the matrix multiplication with a vector, whereby the matrix is saved using the CSR[17], row by row will be examined. Assume for example that row $i$ is inspected. Then, the elements in the value list on the positions between the value of the $i^{\text{th}}$ position in the row list and $i+1^{\text{th}}$ position subtracted with one in the row list, are located in row $i$. The column in which these elements are situated can be found by looking at the corresponding elements in the column list. Based on this information, the $i^{\text{th}}$ element of the resulting vector can be found by adding the values attained by multiplying the elements in the value list with the elements in the vector. These elements in the vector have to be located at the position with the same column number as the elements in the value list.
The last option is not to implement it, but using a subroutine available in Fortran. Herefor, the package BLAS[19] (Basic Lineair Algebra Subprograms) was used. BLAS is a Fortran Library Routine Document for performing basic vector and matrix multiplications. The subroutine which performs matrix multiplication with a vector is DGEMV. The implementation based on this manner, is given in Appendix B.2.4.3.

According to these three implementations, the CPU time, which is a measure of computing time, was computed. This was done on a Linux computer with the following specifications

- distributor: Ubuntu
- processor: an Intel(R) Core(TM) i7-2600K CPU @ 3.40GHz processor
- memory: 16 GB.

It was calculated in the same way as was mentioned for the Polomium problem. For finding the nuclide concentration after 90 days for the nuclides in the decay problem, the CPU times are given in table 4.

| **Implementation methods** | **CPU time (s)** |
|---|---|
| Standard multiplication of a matrix with a vector | 160.306 |
| Exploiting CRS | 159.863 |
| Using the subroutine DGEMV | 504.53 |

Table 4: CPU time corresponding with the different implementations of multiplying a matrix with a vector

From this, it can be seen that the code which exploits CRS is the fastest. However, it is just marginally faster than the standard matrix multiplication with a vector without using a subroutine. This could be explained by the fact that the standard matrix multiplication of a matrix with a vector as well as the one which makes use of the CRS take into account the sparsity of the Bateman equations in contrast to the code which uses the subroutine DGEMV.
It should be noticed that the results obtained with RADAU5 in Python or the results of RADAU5 in Fortran don't differ a lot. To measure the differences in results achieved by the RADAU5 solver implemented in Python or Fortran, note $x_i$ the nuclide concentration of the $i^{th}$ nuclide resulting from Python and $y_i$ the nuclide concentration of the $i^{th}$ nuclide resulting from Fortran. Then the quantity scaled error per nuclide is determined as

$$\text{scaled error per nuclide} = \begin{cases} |x_i - y_i| & \text{if } |x_i| < 1 \\ \dfrac{|x_i - y_i|}{x_i} & \text{if } |x_i| \geqslant 1 \ . \end{cases} \tag{3.1.2}$$

The value of this quantity (comparing the nuclide concentration after 90 days) for each nuclide of the decay problem is presented in figure 16. From this picture, it can be concluded that the difference is negligible.

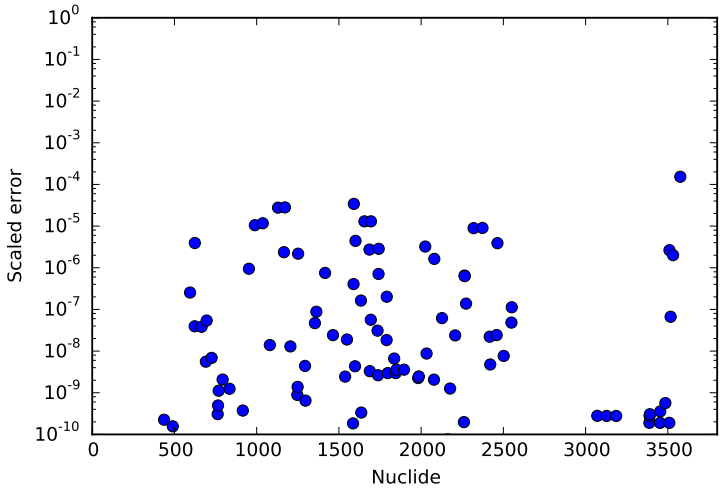


Figure 16: A representation of the difference in results between the Fortran and Python codes

### 3.1.2 Results

#### 3.1.2.1 The Polonium problem

The nuclide concentration of the different nuclides in the Polonium system after 90 days, calculated with the four methods, can be found in table 5. The RadauIIA method makes use of a relative tolerance equal to $10^{-4}$ and an absolute tolerance of $10^{-4}$. In table 5, the exact solution is also presented.

| | $^{209}$Bi | $^{210}$Bi | $^{210}$Po |
|---|---|---|---|
| Exact solution | $0.695886089 \cdot 10^{-3}$ | $7.96452197 \cdot 10^{-10}$ | $7.45182496 \cdot 10^{-9}$ |
| RadauIIA method | $0.695886089 \cdot 10^{-3}$ | $7.92878911 \cdot 10^{-10}$ | $7.45552826 \cdot 10^{-9}$ |
| The scaling and squaring algorithm | $0.695886089 \cdot 10^{-3}$ | $7.96452197 \cdot 10^{-10}$ | $7.45182495 \cdot 10^{-9}$ |
| Combine the action of the matrix exponential on a vector | $0.695886089 \cdot 10^{-3}$ | $7.96452197 \cdot 10^{-10}$ | $7.45182495 \cdot 10^{-9}$ |
| CRAM | $0.695886089 \cdot 10^{-3}$ | $7.96452197 \cdot 10^{-10}$ | $7.45182495 \cdot 10^{-9}$ |

Table 5: Nuclide concentration after 90 days

One only notices a difference between the exact solution and the solution obtained with RadauIIA.
The CPU time of the different methods is given in table 6. From this, it can be seen that RadauIIA is the fastest method for this problem.

| | CPU time (s) |
|---|---|
| RadauIIA method | 0.00118589178658 |
| The scaling and squaring algorithm | 0.00154966145394 |
| Combine the action of the matrix exponential on a vector | 0.00620829529319 |
| CRAM | 0.00829882947937 |

Table 6: CPU time for the different methods used to solve the Polonium problem

#### 3.1.2.2 The decay problem and the fresh fuel problem

Through research at the SCK-CEN, the results obtained from the RadauIIA method are acceptable. Therefore, it has to be investigated if the results achieved from the scaling and squaring algorithm, to compute the matrix exponential, or CRAM, agree with the results of the RadauIIA method. To this objective, the scaled error per nuclide (3.1.2) is used, but now with $x_i$ the nuclide concentration of the $i^{th}$ nuclide resulting from the RadauIIA method and $y_i$ the nuclide concentration of the $i^{th}$ nuclide resulting from the scaling and squaring algorithm or CRAM.
It has to be noticed that the method of combining the action of the matrix exponential on a vector gives an error message. Figure 17 and figure 18 show the quantity scaled error per nuclide in function of a specific nuclide, for the comparison of the RadauIIA method and the scaling and squaring algorithm to compute the matrix exponential. For the decay problem, it is given in figure 17 and it compares the nuclide concentration after 90 days. For the fresh fuel problem it can be seen in figure 18 and it compares the nuclide concentration after six days. Based on this, it can be concluded that the scaling and squaring algorithm isn't a good method to solve the Bateman equations.

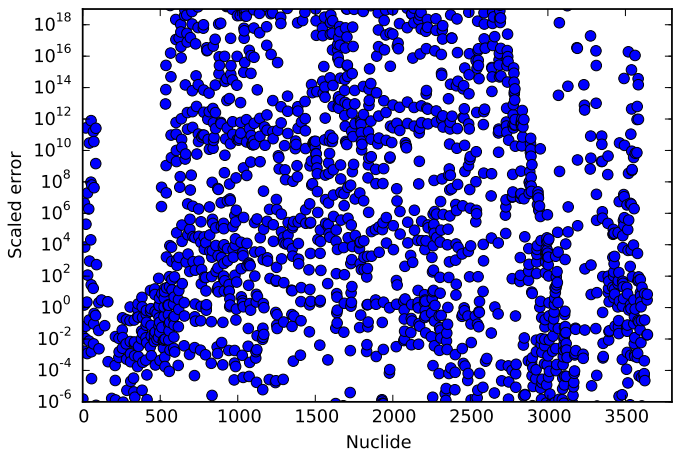


Figure 17: A representation of the difference in results between the RadauIIA method and the scaling and squaring algorithm for the decay problem

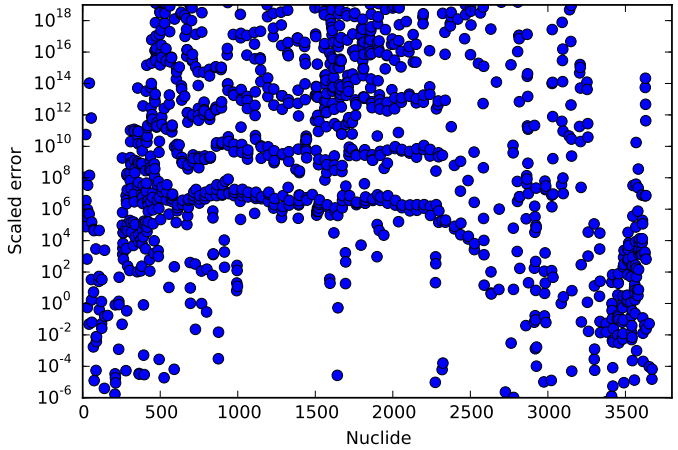


Figure 18: A representation of the difference in results between the RadauIIA method and the scaling and squaring algorithm for the fresh fuel problem

To see if CRAM is a useful way to solve the Bateman equations, figure 19 displays the quantity scaled error per nuclide (comparing the nuclide concentration after 90 days) in function of a specific nuclide for the decay problem. For the fresh fuel problem, this is shown in figure 20, whereby now the nuclide concentration is compared after six days. Again, it can be deduced that CRAM isn't an acceptable approach to solve the Bateman equations.

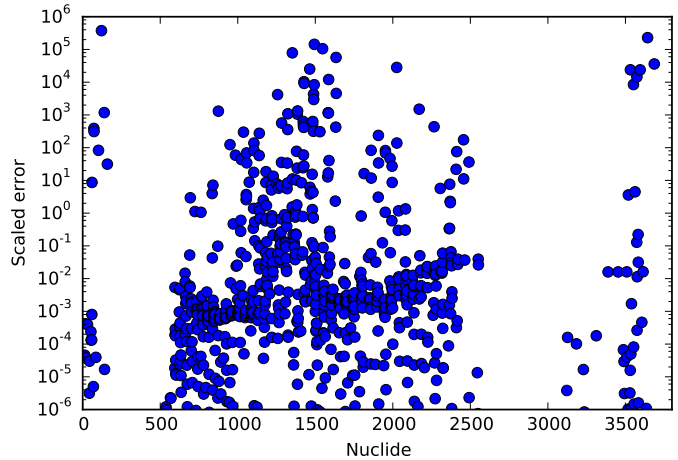


Figure 19: A representation of the difference in results between the RadauIIA method and CRAM for the decay problem

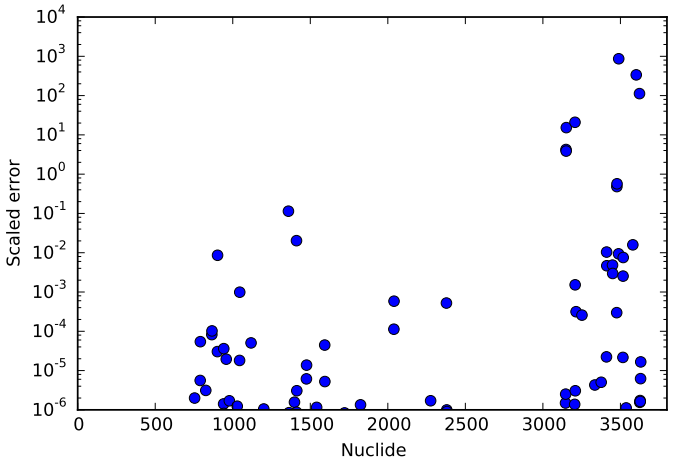


Figure 20: A representation of the difference in results between the RadauIIA method and CRAM for the fresh fuel problem

## 3.2 Implementation of methods using Padé approximations to the exponential

After the examination of the different algorithms to solve the Bateman equations, the new objective was to modify the existing implementation of the RadauIIA method. Rather than immediately starting to change the implementation, the first idea was to try to build a simple implementation of the RadauIIA method, so without solving the created linear system (2.2.23), without step modification, ... Instead the new implementation is based on the stability function $R(z)$ of the RadauIIA method and uses a fixed stepsize $h$. Like it was explained in section 2.2.1, the stability function $R(z)$ of a method can be interpreted as the numerical solution after one step for the problem

$$y' = \lambda y, \quad \lambda \in \mathbb{C}, \quad \Re(\lambda) < 0 \text{ [12]}. \tag{3.2.1}$$

So, if a problem $y' = My$ is considered, then the RadauIIA method can be written as

$$y_{n+1} = R(hM)y_n \tag{3.2.2}$$

where
$$R(hM) = \left(I - \frac{3}{5}hM + \frac{3}{10}\frac{h^2M^2}{2!} - \frac{1}{10}\frac{h^3M^3}{3!}\right)^{-1}\left(I + \frac{2}{5}hM + \frac{1}{10}\frac{h^2M^2}{2!}\right).$$

By rewriting this, the RadauIIA method becomes

$$\left(I - \frac{3}{5}hM + \frac{3}{10}\frac{h^2M^2}{2!} - \frac{1}{10}\frac{h^3M^3}{3!}\right)y_{n+1} = \left(I + \frac{2}{5}hM + \frac{1}{10}\frac{h^2M^2}{2!}\right)y_n. \tag{3.2.3}$$

This technique was also carried out for the implicit Euler method and the Trapezoidal rule. Note that both the implicit Euler method and the Trapezoidal rule are A-stable. This means that it contains the stability region $\mathbb{C}^-$, as mentioned in section 2.2.1 .
The implicit Euler method solves the expression

$$(I - hM)y_{n+1} = y_n \tag{3.2.4}$$

and the Trapezoidal rule solves

$$\left(I - \frac{hM}{2}\right)y_{n+1} = \left(I + \frac{hM}{2}\right)y_n. \tag{3.2.5}$$

Remark that for all these three methods, the Bateman equations are solved by using a Padé approximation (definition see section 2.1.1) for $e^x$, which explains the title of this section.
To work out the solution of this system of equations MUMPS (MUltifrontal Massively Parallel sparse direct Solver) 5.0.1 was used. MUMPS[20] is a solver for large linear systems and exploits the sparsity of a system. It can be used as a parallel solver, as well as a sequential one. A multifrontal approach forms the basis of MUMPS. Details of the implementation are provided on the website of MUMPS[20].

In Appendix C.1, the implementation of the implicit Euler method can be found and in Appendix C.2, C.3, the Trapezoidal rule, respectively the RadauIIA method. For the implementation, the program language Fortran was used and the program was IFORT 15.0.1[21] (mpiifort). The execution of the codes which implements these different methods, was carried out using the STEVIN Supercomputer Infrastructure at Ghent University (the high performance computing - HPC - infrastructure of Ghent University), funded by Ghent University, the Flemish Supercomputer Center (VSC), the Hercules Foundation and the Flemish Government – department Economic, Science and Innovation (EWI).

### 3.2.1 Results

The concentration of the different nuclides in the Polonium system, which were Bismuth-209, Bismuth-210 and Polonium-210, after 90 days was calculated. The stepsize was ten seconds and hence 777600 steps were taken. The results can be found in table 7.

| | $^{209}$Bi | $^{210}$Bi | $^{210}$Po | CPU time (s) |
|---|---|---|---|---|
| Exact solution | $6.958860886 \cdot 10^{-4}$ | $7.964521967 \cdot 10^{-10}$ | $7.451824964 \cdot 10^{-9}$ | |
| Implicit Euler method | $6.958959858 \cdot 10^{-4}$ | $7.964452407 \cdot 10^{-10}$ | $7.452551624 \cdot 10^{-9}$ | 0.51 |
| Trapezoidal rule | $6.958959857 \cdot 10^{-4}$ | $7.964585934 \cdot 10^{-10}$ | $7.452978684 \cdot 10^{-9}$ | 0.5 |
| RadauIIA method | $6.958959857 \cdot 10^{-4}$ | $7.964529839 \cdot 10^{-10}$ | $7.451302254 \cdot 10^{-9}$ | 0.43 |

Table 7: Nuclide concentration after 90 days and CPU time

It could be noticed that the CPU time for the different methods is higher than the CPU time for solving the Polonium system with the RadauIIA method, given in table 6.
The implicit Euler method, the Trapezoidal rule and the RadauIIA method were also applied on bigger Bateman matrices. However, the implementation of the RadauIIA method by using the stability function doesn't work for larger Bateman matrices. This is due to the requirement of calculating the square of the Bateman matrix and the third power of it. To see how well then the implicit Euler method and the Trapezoidal rule work for bigger Bateman matrices, the maximum scaled error over all nuclides was computed for the fresh fuel problem and the burned fuel problem. Different fixed stepsizes were used. The scaled error per nuclide (3.1.2) takes in this context $x_i$ as the nuclide concentration of the $i^{th}$ nuclide resulting from the RADAU5 solver of E. Hairer and G. Wanner. $y_i$ will be the nuclide concentration of the $i^{th}$ nuclide resulting from the implicit Euler method, the Trapezoidal rule or the RadauIIA method. Moreover, the CPU time to run the codes for different stepsizes was measured, in seconds. Like mentioned in section 1.4.4, the total irradiation time for the fresh fuel problem is six days. From section 1.4.5, the total irradiation time for the burned fuel problem is 34.4 days. To solve the fresh fuel problem, the used stepsizes are 10 s, 100 s, 960 s, 9600 s, 86400 s and 518400 s. For the burned fuel problem, the stepsizes 10 s, 96 s, 960 s, 9288 s, 99072 s, 990720 s and 2972160 s were chosen.
The implicit Euler method will be first considered. For the fresh fuel problem, the maximum scaled error per stepsize is represented in figure 21 and the CPU time (in seconds) per stepsize is given in figure 22.
From figure 21, it can be concluded that the maximum scaled error increases when the stepsize increases, so when less steps are taken. Furthermore, the maximum scaled error is small until stepsize 9600 s. From figure 22, it can be noticed that the bigger the stepsize, the shorter the CPU time. Since this is a natural conclusion and could also be noticed for the other situations (fresh/burned fuel problem and implicit Euler/Trapezoidal method), the figures of the CPU time against different fixed stepsizes will not be shown anymore.

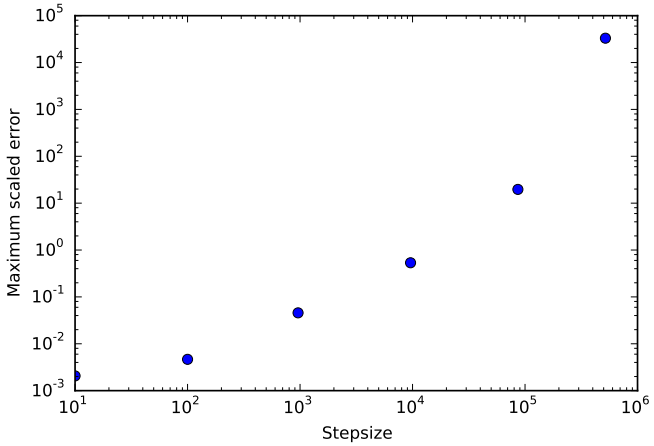


Figure 21: A representation of scaled error per nuclide for the fresh fuel problem using the implicit Euler method

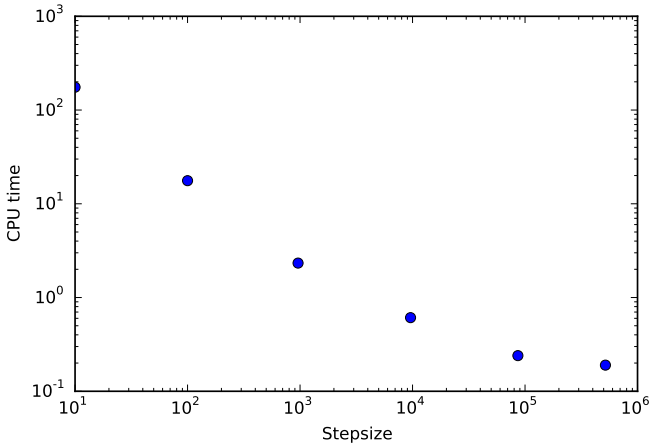


Figure 22: A representation of the CPU time for calculating the nuclide concentration of the fresh fuel problem using the implicit Euler method

Figure 23 displays the maximum scaled error per nuclide for the burned fuel problem. From this, it can be observed that the maximum scaled error first decreases when the stepsize increases and from stepsize 960 s on it increases. Anyway, the maximum scaled error is little for all stepsizes.

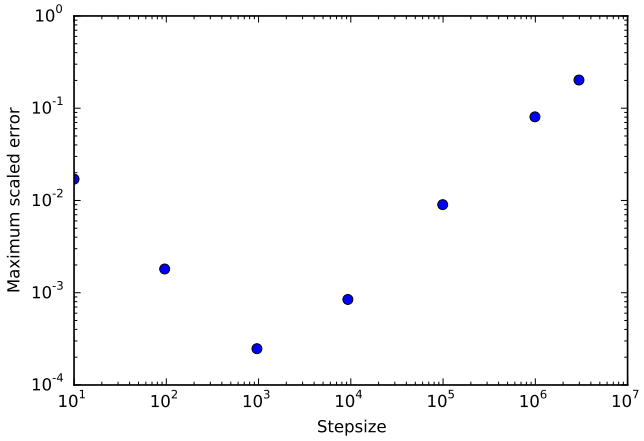


Figure 23: A representation of scaled error per nuclide for the burned fuel problem using the implicit Euler method

Now, the Trapezoidal rule will be examined. Figure 24 displays the maximum scaled error for the fresh fuel problem using different fixed stepsizes. It can be seen that the maximum scaled error is small for all stepsizes, except the stepsize 518400 s. The maximum scaled error for the burned fuel problem using different stepsizes is represented in figure 25. It indicates that the maximum scaled error is small for all chosen stepsizes.

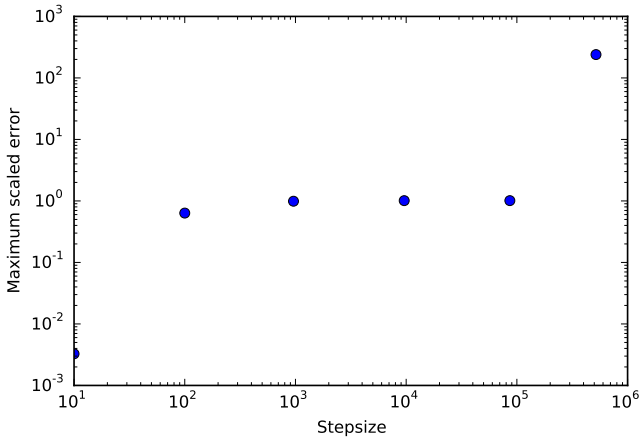


Figure 24: A representation of scaled error per nuclide for the fresh fuel problem using the Trapezoidal rule

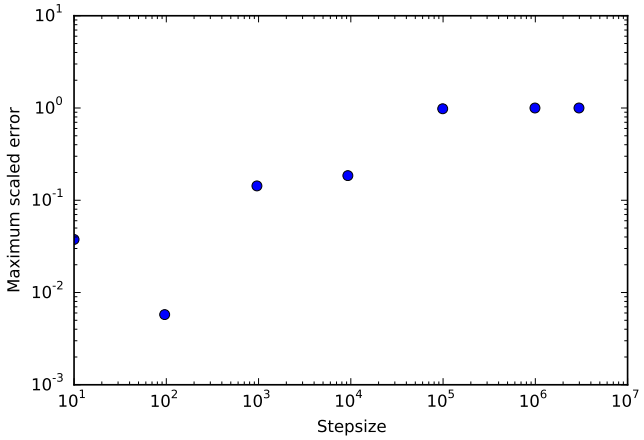


Figure 25: A representation of scaled error per nuclide for the burned fuel problem using the Trapezoidal rule

## 3.3 Adjusting the existing RADAU5 solver

In the previous section, a simple implementation of the RadauIIA method was explored. Now, the possibility will be examined to optimize the existing RADAU5 solver of the RadauIIA method for the Bateman equations.

### 3.3.1 Exploring the way of solving systems of equations

The implementation of the RadauIIA method consists of several parts, as discussed in section 2.2. First, the way of solving the equations (2.2.23) will be explored, which were

$$\begin{aligned}(h^{-1}\Lambda \otimes I - I \otimes J)\Delta W^k &= -h^{-1}(\Lambda \otimes I)W^k + (T^{-1} \otimes I)F((T \otimes I)W^k) \\ W^{k+1} &= W^k + \Delta W^k.\end{aligned} \tag{3.3.1}$$

This system was split into two linear systems, one real with dimension $m$ and one complex, also with dimension $m$. Currently, BLAS subroutines are used to solve the system of equations (3.3.1). As mentioned before, BLAS[19] is a fortran library for executing elementary vector and matrix multiplications. First of all,

a subroutine of BLAS is called to do a $LU$ factorisation. This factorisation allows to write the matrix of the system as a product of a lower triangular matrix, noted $L$, and an upper triangular matrix, noted $U$. The lower triangular matrix $L$ has ones on the diagonal. In addition, the decomposition can also make use of a permutation matrix $P$. Hence the $LU$ factorisation of a matrix $M$ can be written as $PM = LU$. To see how than a system $Mx = a$ can be worked out using this $LU$ factorisation, the system will be premultiplied with the permutation matrix $P$ on both sides of the equation. This gives the expression $PMx = Pa$. To represent the right-hand side of this new system, the vector $d$ will be used, so $d = Pa$. Hence the system $PMx = d$ is achieved. Replacing the term $PM$ through the factorisation $LU$, gives $LUx = d$. Consequently, to solve the system $Mx = a$ using the $LU$ factorisation, the system $Ly = d$ has to be examined first and thereafter the system $Ux = y$.
To perform a $LU$ factorisation on the real subsystem of the RadauIIA method, the subroutine DGETRF is called in the RADAU5 solver. For the complex subsystem, the subroutine ZGETRF is used. The real and complex subsystems are then solved using the subroutine DGETRS, respectively ZGETRS.
The problem of these subroutines is that it doesn't take into account the sparsity of the Bateman equations. After some research, several packages were found that solve systems of equations by exploiting the sparsity of it. One possibility was found in the Harwell Subroutine Library (HSL). This library contains the packages MA38[22] and ME38[23]. MA38 and ME38 solve sparse, unsymmetric systems of equations and are both based on a combined unifrontal/multifrontal algorithm. MA38 is specific for real systems of equations and ME38 for complex systems. Details of the implementation can be found in the user documentation contributed at the website of HSL[22],[23].
MA38 and ME38 were used to rebuild the left-hand side of the first equation of the system (3.3.1) in function of the data type used in MA38 and ME38. On this left-hand side, $LU$ factorisation was done. The code of this is given in Appendix D.1 for the real subsystem and in Appendix D.2 for the complex subsystem. After that, it would be necessary to incorporate it in the RADAU5 solver. However, based on the experience of researchers from KULeuven, MUMPS has been recommended. Therefore, subsequent work to change the implemented RADAU5 solver was done using MUMPS[20].
To use MUMPS for solving the system of equations (3.3.1), it has to be investigated how the matrix $h^{-1}\Lambda \otimes I - I \otimes J$ can be supplied to MUMPS. MUMPS has several possible matrix formats. For this master thesis, the used matrix format consists of three lists, namely

- a row indices list which contains the row positions of the elements in the matrix
- a column indices list which includes the corresponding column positions of the elements in the matrix
- an element values list which has the corresponding values of the elements in the matrix.

For example, the matrix

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 0 & 5 \\ 25 & 0 & 0 & 0 \end{bmatrix} \tag{3.3.2}$$

would be given to MUMPS using the following lists

- the row indices list is [1 1 2 2 3 3 4]
- the column indices list is [1 2 2 3 1 4 1]
- the element values list is [4 1 2 1 1 5 25].

To form the left-hand side of the system of equations (3.3.1), remember that it was split into two linear systems. One system contains the matrix $\gamma I - J$ and the other the matrix $(\alpha + \beta i)I - J$. In case of the Bateman equations, the Jacobian $J$ is simply the Bateman matrix. Therefore, the Bateman matrix was first read in as three lists, like needed for MUMPS. Thereafter, these three lists were manipulated to form the

matrices $\gamma I - J$ and $(\alpha + \beta i)I - J$. It should be noticed that if a certain position on the diagonal contains the value zero, it will not be included in the three lists. However, it is needed to fill in these position afterwards by the value $\gamma$ or $\alpha + \beta i$. To this end, an extra check function is built in the code to memorize these positions. Forming these matrices gives the left-hand of the system of equations (3.3.1).
Since the right-hand side of the system of equations (3.3.1) is a vector and the data type of the original RADAU5 solver corresponds with the datatype that MUMPS uses, the implementation of the right-hand side in the original RADAU5 solver doen't have to be changed.
To solve then the system of equations (3.3.1), MUMPS can be used. The package MUMPS works with several stages, namely

- initializing the package
- accomplishing an analysis
- performing the factorisation
- computing the solution
- terminating the package.

These stages are recognized by MUMPS through the parameter "job".
In the next section, two codes will be described whereby MUMPS is used for solving the system of equations (3.3.1) and is incorporated in a programme to get a solution of the Bateman equations.

### 3.3.2 Using the MUMPS package

In this paragraph, two codes will be described for solving the Bateman equations, both using the package MUMPS. One code will be a fixed stepsize implementation of the original RADAU5 solver of E. Hairer and G. Wanner[16]. The other code plugs in MUMPS in the original RADAU5 solver.

#### 3.3.2.1 A fixed stepsize implementation of the original RADAU5 solver with MUMPS

Initially, it was tried to build a stripped version of the original RADAU5 solver of E. Hairer and G. Wanner[16]. The main changes, compared to the original RADAU5 solver, were

- a fixed stepsize was selected, so that a single factorisation was needed for all the steps
- MUMPS was used to solve the system of equations (3.3.1) in order to take advantage of the sparsity of the system
- since the system of equations (3.3.1) is linear, there were no simplified Newton iterations done in order to solve the system of equations (3.3.1). Furthermore, the starting values of the simplified Newton iterations were chosen to be $z_i^0$=0, i=1,2,3, for all the steps.

The code of this can be found in Appendix D.3.1.
To perform tests of the code, it was excecuted on the HPC Infrastructure at Ghent University. The program language Fortran was used and the program was IFORT 15.0.1[21] (mpiifort).
To see for which stepsize the code works well to solve the Bateman equations, the maximum scaled error over all nuclides was computed for the fresh fuel problem and the burned fuel problem using different fixed stepsizes. The scaled error per nuclide is defined by the equation (3.1.2) whereby now $x_i$ is the nuclide concentration of the $i^{th}$ nuclide resulting from the original RADAU5 solver and $y_i$ is the nuclide concentration of the $i^{th}$ nuclide resulting from the stripped RADAU5 solver. In addition, the CPU time to run the code for the different stepsizes was calculated. Remember the total irradiation time for the fresh fuel problem is six days and for the burned fuel problem 34.4 days. The different stepsizes to solve the fresh fuel problem were 10 s, 100 s, 960 s, 9600 s, 86400 s and 518400 s. These are the same values as in section 3.2.1. In figure 26, the maximum scaled error for each stepsize can be seen for this problem. Until stepsize 86400 s, the maximum scaled error is small.

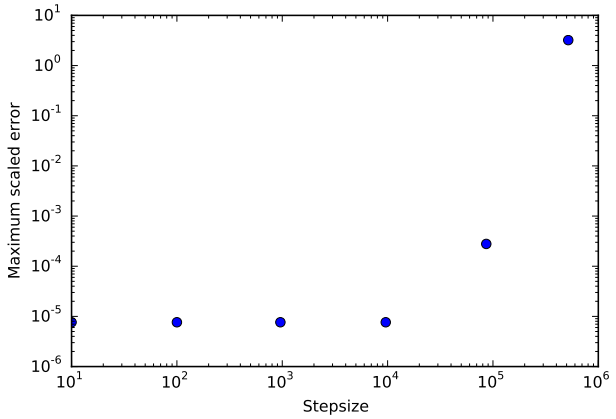


Figure 26: A representation of the maximum scaled error for the different stepsizes intended to solve the fresh fuel problem

Figure 27 displays the CPU time to execute the code for solving the fresh fuel problem using the same stepsizes. The CPU time was measured in seconds. It can be seen that the bigger the stepsize, so the less steps are taken, the shorter the CPU time.

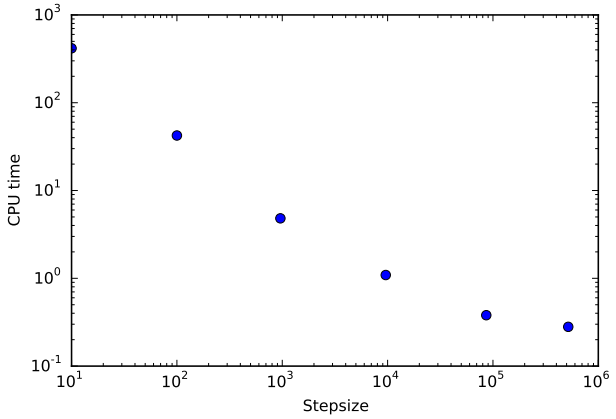


Figure 27: An illustration of the CPU time for the different stepsizes intended to solve the fresh fuel problem

To solve the burned fuel problem, the same stepsizes as in section 3.2.1 were chosen. These stepsizes were 10 s, 96 s, 960 s, 9288 s, 99072 s, 990720 s and 2972160 s. The maximum scaled error for each stepsize is shown in figure 28. From this, it can be noticed that for all stepsizes the maximum scaled error is small.

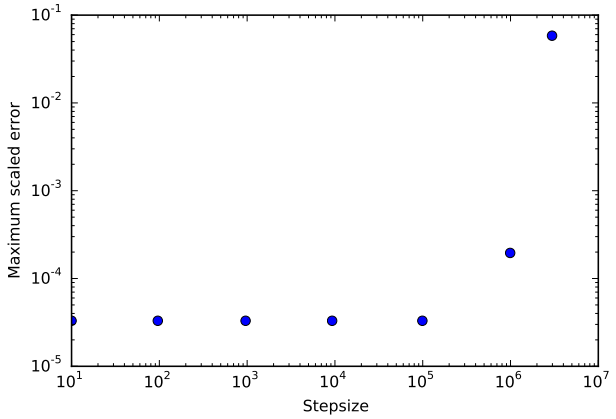


Figure 28: A representation of the maximum scaled error for the different stepsizes intended to solve the burned fuel problem

In figure 29, the CPU time (in seconds) to execute the code for solving the burned fresh fuel problem is demonstrated. Again, it can be observed that the bigger the stepsize, the shorter the CPU time.

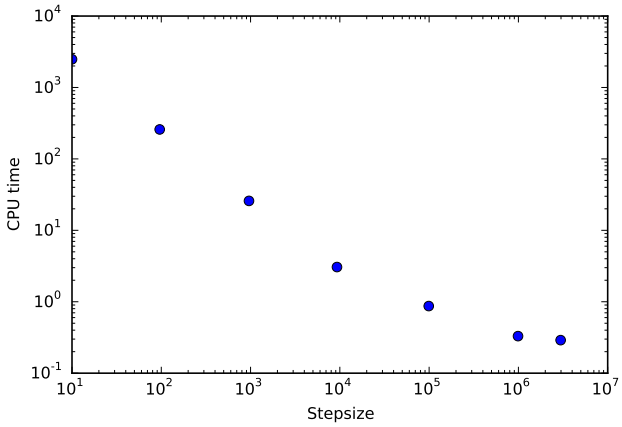


Figure 29: An illustration of the CPU time for the different stepsizes intended to solve the burned fuel problem

From calculating the maximum scaled error and the CPU time for the fresh fuel problem and the burned fuel problem using different stepsizes, the following can be concluded:
by using too small stepsizes, there isn't a gain in accuracy, only more computing time is needed.
In the future, it has to be investigated if it is possible to determine a fixed stepsize in the beginning of the code which guarantees a sufficiently accurate result for the Bateman equations whereby the computing time is not too large. In the next section, the modification of the original RADAU5 solver by plugging in MUMPS will be given. In this code, the stepsize is modified all the time, making the problem of finding a good fixed stepsize at the beginning to disappear.

#### 3.3.2.2 Modifying the existing RADAU5 solver with MUMPS

Now, it will be discussed how the package MUMPS, which can be used for solving sparse systems equations, can be plugged in the original RADAU5 solver. In section 3.3.1, it is discussed how the call to the BLAS subroutines for solving the system of equations can be changed to the call of the package MUMPS. Notice that for this, the subroutine which calls the Jacobian of the system of differential equations, has to be changed to the datatype of MUMPS. In addition, the way of calculating the error estimation in the original RADAU5 solver, required to do the stepsize prediction (see section 2.2.5), makes use of solving a system of equations and was therefore also modificated such that the package MUMPS could be used. To estimate the error in the original RADAU5 solver, the subroutine ESTRAD is called. In order to modify the subroutine ESTRAD, that what's necessary to do the error estimation in the case of the Bateman equations, was plugged in the original RADAU5 solver. In this way, there does not have to be a call to the subroutine ESTRAD. Thereafter, the call to the subroutine DEGTRS, for solving the system of equations, was changed to a call to MUMPS. The resulting code can be found in Appendix D.3.2.1 and the related driver is given in Appendix D.3.2.2.
This code was tested for the fresh fuel problem and the burned fuel problem whereby the fuel is irradiated for six days, respectively 34,4 days. The HPC infrastructure at Ghent University was used to perform the tests. Anew, the scaled error per nuclide (3.1.2) was used to see the difference in results with the original RADAU5 solver. In this case, $x_i$ is in the definition of scaled error per nuclide, the nuclide concentration of the $i^{th}$ nuclide resulting from the original RADAU5 solver and $y_i$ is the nuclide concentration of the $i^{th}$ nuclide resulting from the adapted RADAU5 solver in order to use MUMPS. For both the original RADAU5 solver as the new code, a relative tolerance of $10^{-4}$ and an absolute tolerance of $10^{-4}$ was taken.
Figure 30 shows the scaled error per nuclide for the fresh fuel problem and figure 31 displays it for the burned fuel problem. From both figures, it can be concluded that the results are very accurate.

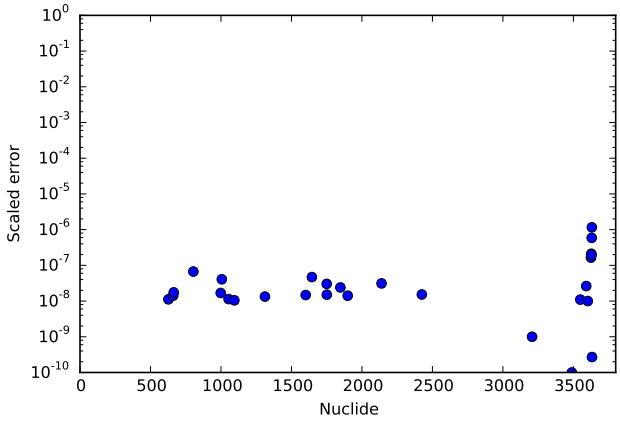


Figure 30: A representation of scaled error per nuclide for the fresh fuel problem

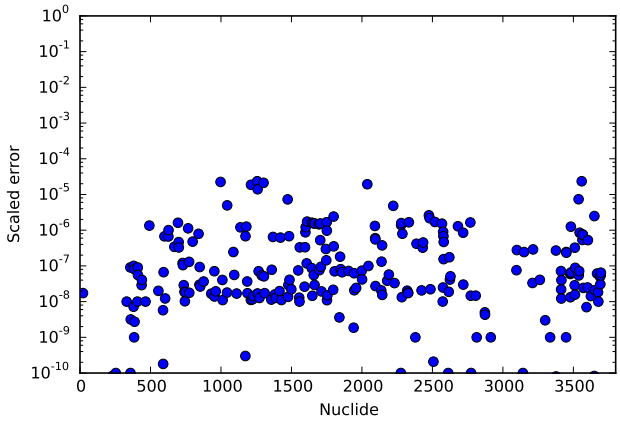


Figure 31: A representation of scaled error per nuclide for the burned fuel problem

The CPU time for calculating the nuclide concentrations after six days of irradiating the fresh fuel and 34.4 days for the burned fuel problem, using the original RADAU5 solver and the modificated RADAU5 solver using MUMPS, is given in table 8. It shows the CPU time is almost halved for the fresh fuel problem and roughly three times smaller for the burned fuel problem.

| | RADAU5 solver of E. Hairer and G. Wanner[16] | modificated RADAU5 solver using MUMPS |
|---|---|---|
| Fresh fuel problem | 36.26 s | 18.76 s |
| Burned fuel problem | 28.1 s | 10.46 s |

Table 8: The CPU times for the different implementations to solve the fresh fuel problem and the burned fuel problem

## 3.4 Comparing the different approaches to solve the Bateman equations

So far, three different ways are found to solve the Bateman equations, namely

- the implicit Euler method and the Trapezoidal rule whereby a fixed stepsize is used
- a fixed stepsize implementation of the original RADAU5 solver
- a modification of the original RADAU5 solver to take into account the sparsity of the Bateman systems.

All these three methods use MUMPS for solving sparse system of equations.
The first two approaches work with a fixed stepsize. A remaining research question for these approaches is to determine a fixed stepsize in the beginning of the code. This stepsize needs to guarantee a sufficiently accurate result for the Bateman equations and so that the computing time is not too large.
The third approach, which plugs in MUMPS in the original RADAU5 solver of E. Hairer and G. Wanner, gives accurate results and the computing time is greatly reduced. It has no further research questions and it is based on a code with a huge research background. Therefore, this approach is recommended.

# 4 Conclusion

In this master thesis, some examples of the Bateman equations were first explained. From the investigation of these examples, typical features of the Bateman equations were found. The objective of this master thesis was to improve the way the Bateman equations are currently solved at SCK-CEN. Hence, different algorithms which could possibly be used to solve the Bateman equations were considered. These algorithms were

- the scaling and squaring algorithm
- CRAM
- RadauIIA method.

These different algorithms were implemented and compared. At this moment, the RadauIIA method, which is found to be an acceptable method to solve the Bateman equations, is used. From the comparision with the results of the RadauIIA method, it could be seen that the scaling and squaring algorithm and CRAM aren't appropriate to solve the Bateman equations. Hence, methods using Padé approximations to the exponential were analysed. These methods were

- the implicit Euler method
- the Trapezoidal rule
- RadauIIA method

implemented using the stability function.
However, the RadauIIA method implemented using the stability function doesn't work for the Bateman equations. Moreover, it still has to be examined for the implicit Euler method and the Trapezoidal rule if it is feasible to determine a fixed stepsize at the beginning of the code, which will guarantee a sufficiently accurate result.
Finally, the RADAU5 solver which is the implementation of the RadauIIA method by E. Hairer and G. Wanner, was adjusted. This was done in two ways. One is a fixed stepsize implementation of the original RADAU5 solver and the other one plugs in MUMPS in the original RADAU5 solver. It could be concluded that the method that plugs in MUMPS in the original RADAU5 solver is the best way to solve the Bateman equations.

# Appendices

## A Summary

### A.1 English

Since 2004, SCK-CEN, the Belgian nuclear research centre, has been engaged in the development of the ALEPH code. A part of the ALEPH code consists of solving the Bateman equations. The Bateman equations are used to describe the time evolution of nuclides in a nuclear system. This evolution consists of radioactive decay from one nuclide to another and the production of nuclides by fission, neutron capture, ... The Bateman equations are a set of first order differential equations of the form

$$n'(t) = Bn(t), \qquad n(0) = n_0\,, \tag{A.1.1}$$

with $n(t) \in \mathbb{R}^{\mathrm{m}}$ the nuclide concentration vector, $n_0 \in \mathbb{R}^{\mathrm{m}}$ the initial nuclide concentration and $B \in \mathbb{R}^{\mathrm{mxm}}$ the Bateman or burnup matrix.
The diagonal elements of the Bateman matrix $b_{ii}$ describe the rate by which a nuclide $i$ is transformed to other nuclides. The off-diagonal elements $b_{ij}$ represent the rate by which nuclide $j$ is converted into nuclide $i$ by a physical process.
The Bateman equations can be solved exactly, giving the solution $n(t) = e^{Bt}n_0$. Here the exponential of the matrix $Bt$ is defined by

$$e^{Bt} = I + \sum_{k=1}^{\infty} \frac{(Bt)^k}{k!} \tag{A.1.2}$$

with $I$ the identity matrix.
Some features of the Bateman equations are

- the system of Bateman equations can be small or very large
- the Bateman matrices are very sparse
- the eigenvalues of the Bateman matrix can have extremely small and large magnitudes
- the eigenvalues of the Bateman matrices are typically limited to a region near the negative real axis
- the Bateman matrix 1-norm can be very large.

The objective of this master thesis was to improve the way the Bateman equations are currently solved at SCK-CEN. Hence, several algorithms, which could be possibly used to solve the Bateman equations, were considered. These algorithms are

- calculating the exponential of the system matrix using the scaling and squaring algorithm
- Chebyshev Rational Approximation method (CRAM)
- RadauIIA method.

The first two algorithms work out the exact solution of the Bateman equations, namely $n(t) = e^{Bt}n_0$. In particular, the scaling and squaring algorithm exploits the property that $\left(e^{\frac{M}{\omega}}\right)^{\omega} = e^{M}$, for $M \in \mathbb{C}^{mxm}$ and $\omega \in \mathbb{C}$. CRAM searches a rational function $g^*_{k,k}(x)$ that satisfies the equation

$$\sup_{x\in\mathbb{R}^-} |\hat{g}_{k,k}(x) - e^x| = \inf_{g^*_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}^-} |g^*_{k,k}(x) - e^x| \right\}, \tag{A.1.3}$$

where $\pi_{k,k}$ is the set of rational functions $g^*_{k,k}(x) = \dfrac{p_k(x)}{q_k(x)}$ and $p_k$, $q_k$ are polynomials of order k.
The third method, the RadauIIA method, solves the Bateman equations numerically. It is a three stage implicit Runge-Kutta method and has order of accuracy five. The RadauIIA method is found, through investigation at SCK-CEN, to be appropriate to solve the Bateman equations. Hence, it was controlled if the

results from the scaling and squaring algorithm and CRAM agree with the results of the RadauIIA method. Unfortunately, the results don't coincide. Therefore, there was looked for other ways to solve the Bateman equations. An idea was to use Padé approximations to the exponential. A Padé approximation $r_{\tilde{k}\tilde{m}}(x)$ of a scalar function $f$ is defined as the rational function $r_{\tilde{k}\tilde{m}}(x)=\frac{p_{\tilde{k}\tilde{m}}(x)}{q_{\tilde{k}\tilde{m}}(x)}$ were $p_{\tilde{k}\tilde{m}}$ and $q_{\tilde{k}\tilde{m}}$ are polynomials of degree at most $\tilde{k}$ and $\tilde{m}$, respectively, $q_{\tilde{k}\tilde{m}}(0) = 1$ and $f(x) - r_{\tilde{k}\tilde{m}}(x) = \mathrm{O}(x^{\tilde{m}+\tilde{k}+1})$. The methods, which use Padé approximations to the exponential, were

- the implicit Euler method
- the Trapezoidal rule
- RadauIIA method

implemented using the stability function.
The stability function of a method can be interpreted as the numerical solution after one step for the problem

$$y' = \lambda y, \quad \lambda \in \mathbb{C}, \quad \Re(\lambda) < 0 \; . \tag{A.1.4}$$

However, the RadauIIA method, implemented using the stability function, doesn't work for the Bateman equations. Furthermore, the implicit Euler method and the Trapezoidal rule make use of a fixed stepsize. It still has to be examined if it is feasible to determine a fixed stepsize at the beginning of the code, which will guarantee a sufficiently accurate result whereby the computing time is not too large.
Finally, the possibility to modify the RADAU5 solver, which is the implementation of the RadauIIA method by E. Hairer and G. Wanner, to take into account the properties of the Bateman equations, was investigated. This was done in two ways. One is a fixed stepsize implementation of the original RADAU5 solver and the other plugs in MUMPS in the original RADAU5 solver. MUMPS is a solver for large linear systems and it exploits the sparsity of a system. Anew, it has to be investigated for the fixed stepsize implementation of the original RADAU5 solver if it is possible to determine a fixed stepsize at the beginning of the code. The results of the approach which plugs in MUMPS in the original RADAU5 solver are very accurate and reduce the computing time significantly. Hence, it could be concluded that the method that plugs in MUMPS in the original RADAU5 solver is recommended to solve the Bateman equations.

## A.2 Dutch

Sinds 2004 is het SCK-CEN, het studiecentrum voor kernenergie, betrokken bij de ontwikkeling van de ALEPH code. Een deel van de ALEPH code bestaat uit het oplossen van de Bateman vergelijkingen. De Bateman vergelijkingen worden gebruikt om de tijdsevolutie van nucliden te beschrijven in een nucleair systeem. Deze evolutie omvat radioactief verval van een nuclide naar een andere nuclide en de productie van nucliden door kernsplijting, neutronenvangst, ... De Bateman vergelijkingen zijn een stelsel van eerste orde differentiaalvergelijkingen van de vorm

$$n'(t) = Bn(t), \qquad n(0) = n_0 \, , \tag{A.2.1}$$

met $n(t) \in \mathbb{R}^{\mathrm{m}}$ de nuclide concentratie vector, $n_0 \in \mathbb{R}^{\mathrm{m}}$ de begin nuclide concentratie en $B \in \mathbb{R}^{\mathrm{mxm}}$ de Bateman of burnup matrix.
De diagonaal elementen $b_{ii}$ beschrijven de snelheid waarmee een nuclide $i$ is omgevormd naar andere nucliden. De niet-diagonaalelementen $b_{ij}$ stellen de snelheid voor waarmee nuclide $j$ is omgezet naar nuclide $i$ door een fysisch proces.
De Bateman vergelijkingen kunnen exact worden opgelost met als oplossing $n(t) = e^{Bt}n_0$. Hierbij wordt de exponentiële van de matrix $Bt$ gedefinieerd door

$$e^{Bt} = I + \sum_{k=1}^{\infty} \frac{(Bt)^k}{k!} \tag{A.2.2}$$

met $I$ de identiteitsmatrix.

Enkele eigenschappen van de Bateman vergelijkingen zijn:

- het aantal Bateman vergelijkingen kan gering of groot zijn
- de Bateman matrices zijn ijl
- de eigenwaarden van de Bateman matrices kunnen heel klein of groot zijn
- de eigenwaarden van de Bateman matrices behoren typisch tot het negatieve deel van de reële as
- de Bateman matrix 1-norm kan erg groot zijn.

Het doel van deze masterproef was het verbeteren van de manier waarop de Bateman vergelijkingen op dit moment worden opgelost aan het SCK-CEN. Vandaar werden er verschillende algoritmen bestudeerd die mogelijk konden worden gebruikt voor het oplossen van de Bateman vergelijkingen. Deze algoritmen waren

- berekenen van de exponetiële van een matrix met "scaling and squaring" algoritme
- "Chebyshev Rational Approximation method" (CRAM)
- RadauIIA methode.

De eerste twee algoritmen werkten de exacte oplossing van de Bateman vergelijkingen uit, namelijk $n(t) = e^{Bt}n_0$. In het bijzonder maakt "scaling and squaring" algoritme gebruik van de eigenschap dat $\left(e^{\frac{M}{\omega}}\right)^{\omega} = e^{M}$, voor $M \in \mathbb{C}^{mxm}$ en $\omega \in \mathbb{C}$. CRAM zoekt een rationale functie $g^*_{k,k}(x)$ die voldoet aan de vergelijking

$$\sup_{x\in\mathbb{R}^-} |\hat{g}_{k,k}(x) - e^x| = \inf_{g^*_{k,k}\in\pi_{k,k}} \left\{ \sup_{x\in\mathbb{R}^-} |g^*_{k,k}(x) - e^x| \right\}, \tag{A.2.3}$$

waarbij $\pi_{k,k}$ de verzameling van rationale functies $g^*_{k,k}(x) = \dfrac{p_k(x)}{q_k(x)}$ is en $p_k$, $q_k$ veeltermen van graad k zijn.
De derde methode, RadauIIA methode, is een impliciete 3-traps Runge-Kutta methode van de vijfde orde. De RadauIIA methode werd door onderzoek aan het SCK-CEN geschikt bevonden voor het oplossen van de Bateman vergelijkingen. Vandaar werd er gecontroleerd of de resultaten van "scaling and squaring" algoritme en CRAM overeenkomen met de resultaten van de RadauIIA methode. De resultaten bleken echter niet overeen te stemmen. Daarom werd er gezocht naar andere manieren om de Bateman vergelijkingen op te lossen. Een idee was om Padé benaderingen voor de exponentiële functie te gebruiken. Een Padé benadering van een scalaire functie f is gedefinieerd als een rationale functie $r_{\tilde{k}\tilde{m}}(x)=\dfrac{p_{\tilde{k}\tilde{m}}(x)}{q_{\tilde{k}\tilde{m}}(x)}$ met $p_{\tilde{k}\tilde{m}}$ een veelterm van graad ten hoogste $\tilde{k}$, $q_{\tilde{k}\tilde{m}}$ een veelterm van graad ten hoogste $\tilde{m}$, $q_{\tilde{k}\tilde{m}}(0) = 1$ en $f(x) - r_{\tilde{k}\tilde{m}}(x) = \mathrm{O}(x^{\tilde{m}+\tilde{k}+1})$.
De methode die Padé benaderingen voor de exponentiële functie gebruiken, zijn

- de impliciete Euler methode
- de Trapeziumregel
- RadauIIA methode

geïmplementeerd gebruik makend van de stabiliteitsfunctie.
De stabiliteitsfunctie van een methode kan worden geïnterpreteerd als de numerieke oplossing na één stap van het probleem

$$y' = \lambda y, \quad \lambda \in \mathbb{C}, \quad \Re(\lambda) < 0\ . \tag{A.2.4}$$

Hoe dan ook, de RadauIIA methode geïmplementeerd gebruik makend van de stabiliteitsfunctie is niet geschikt voor het oplossen van de Bateman vergelijkingen. Bovendien maken de impliciete Euler methode en de Trapeziumregel gebruik van een vaste stapgrootte. Het moet nog worden onderzocht of het mogelijk is om een vaste stapgrootte aan het begin van de code te bepalen die een voldoende nauwkeurig resultaat garandeert zonder dat er teveel berekeningstijd nodig is.
Tot slot werd de mogelijkheid onderzocht om de RADAU5 solver te verbeteren, welke de implementatie is van de RadauIIA methode gemaakt door E. Hairer en G. Wanner, zodat er rekening gehouden wordt met de

eigenschappen van de Bateman vergelijkingen. Dit werd op twee manieren gedaan. De eerste methode is een vaste stap implementatie van de originele RADAU5 solver en de andere brengt MUMPS in in de originele RADAU5 solver. MUMPS is een solver voor grote lineaire stelsels en het benut de ijle structuur van matrices. Opnieuw dient er te worden onderzocht of het mogelijk is om een vaste stapgrootte te bepalen in het begin van de code voor de vaste stapgrootte implementatie. De resultaten van de methode die MUMPS inbrengt in de originele RADAU5 solver zijn erg nauwkeurig en de berekeningstijd verminderde. Vandaar wordt de methode die MUMPS inbrengt in de RADAU5 solver aangeraden om de Bateman vergelijkingen op te lossen.

# B The codes of the different algorithms

## B.1 The Polonium problem

### B.1.1 The scaling and squaring algorithm

```
import numpy as np
from scipy import linalg
import time

BatemanMatrix=np.array([[-1.83163*10**(-12),0,0],[1.83163*10**(-12),-1.60035*10**(-6),
0], [0,1.60035*10**(-6),-5.79764*10**(-8)]],float)
InitialVector=np.array([[6.95896*10**(-4)],[0],[0]],float)

t=raw_input("Give the end time in seconds ")
timesec=int(t)
timeday=timesec/86400.0

elapsed=np.zeros(10)

sum=0.0

for i in range(0,10):
        ti = time.clock()

        ExpMatrix=linalg.expm(BatemanMatrix*timesec)
        FinalCon=np.dot(ExpMatrix,InitialVector)

        elapsed[i] = time.clock() - ti
        sum=sum+elapsed[i]

sum=sum-max(elapsed)-min(elapsed)

ExecTime=sum/8.0

print ("The nuclide inventories for t=" + str(timeday) + " days ")
print(FinalCon)

print("The execution time is ")
print(ExecTime)
```

### B.1.2 Combine the action of the matrix exponential on a vector

```
import numpy as np
import scipy.linalg
import scipy.sparse.linalg
import time

BatemanMatrix=np.array([[-1.83163*10**(-12),0,0],[1.83163*10**(-12),-1.60035*10**(-6),
0],[0,1.60035*10**(-6),-5.79764*10**(-8)]],float)
InitialVector=np.array([[6.95896*10**(-4)],[0],[0]],float)

t=raw_input("Give the end time in seconds ")
timesec=int(t)
timeday=timesec/86400.0
```

```
elapsed=np.zeros(10)
sum=0.0

for i in range(0,10):
        ti = time.clock()

        FinalCon = scipy.sparse.linalg.expm_multiply(timesec*BatemanMatrix,
                InitialVector)

        elapsed[i] = time.clock() - ti
        sum=sum+elapsed[i]

sum=sum-max(elapsed)-min(elapsed)

ExecTime=sum/8.0

print ("The nuclide inventories for t=" + str(timeday) + " days ")
print(FinalCon)

print("The execution time is ")
print(ExecTime)
```

### B.1.3 Chebyshev Rational Approximation method

```
import time
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import spsolve


def CRAM(A, t, n0, alpha, theta):
        """ Implementation of the CRAM method for a certain
        approximation defined by the alpha and theta coefficients.
        It calculates $n = n_0 \cdot \exp (At)$.

        It implements equation (10) from

        Maria Pusa, "Rational Approximations to the Matrix Exponential
        in Burnup Calculations", Nuclear Science and Engineering, 169,
        p.155-167, 2011

        Args:
        A (np.array): A square matrix, usually sparse
        t (float): end time for the integration
        n0 (np.array): the initial condition vector
        alpha (np.array): the alpha coefficients to be used
        theta (np.array): the theta coefficients to be used

        Returns:
        n (np.array): the solution at time 't'
        """

        szi, szj = A.shape
        if (szi != szj):
                raise ValueError('Sorry, only square problems allowed')
```

```
33          At = sp.csr_matrix(A * t)
            n = alpha[0] * n0
35          Z = np.zeros_like(n)

37          for j, t in enumerate(theta):
                B = At - sp.eye(szi) * theta[j]
39              Z += alpha[j+1] * spsolve(B, n0)

41          n += 2.0 * Z

43          return n.real

45
    def CRAM_16(A, t, n0):
47          """ Implementation of the 16th order CRAM method.
            It calculates $n = n_0 \cdot \exp (At)$ at different time steps $t$.
49
            It implements equation (10) using the coefficients in
51          table II from

53          Maria Pusa, "Rational Approximations to the Matrix Exponential
            in Burnup Calculations", Nuclear Science and Engineering, 169,
55          p.155-167, 2011

57          Args:
            A (np.array): A square matrix, usually sparse
59          t (float): end time for the integration
            n0 (np.array): the initial condition vector
61

63          Returns:
            n (np.array): the solution at time 't'
65          """

67          alpha = np.array([+2.124853710495224e-16+0.000000000000000e+00j,
            -5.090152186522492e-07-2.422001765285229e-05j,
69          +2.115174218246603e-04+4.389296964738067e-03j,
            +1.133977517848393e+02+1.019472170421586e+02j,
71          +1.505958527002347e+01-5.751405277642182e+00j,
            -6.450087802553965e+01-2.245944076265210e+02j,
73          -1.479300711355800e+00+1.768658832378294e+00j,
            -6.251839246320792e+01-1.119039109428323e+01j,
75          +4.102313683541002e-02-1.574346617345547e-01j])

77          theta = np.array([-1.084391707869699e+01+1.927744616718165e+01j,
            -5.264971343442647e+00+1.622022147316793e+01j,
79          +5.948152268951177e+00+3.587457362018322e+00j,
            +3.509103608414918e+00+8.436198985884374e+00j,
81          +6.416177699099435e+00+1.194122393370139e+00j,
            +1.419375897185666e+00+1.092536348449672e+01j,
83          +4.993174737717997e+00+5.996881713603942e+00j,
            -1.413928462488886e+00+1.349772569889275e+01j])
85
            return _CRAM(A, t, n0, alpha, theta)
```

```
87 BatemanMatrix=np.array([[-1.83163*10**(-12),0,0],[1.83163*10**(-12),-1.60035*10**(-6),
   0],[0,1.60035*10**(-6),-5.79764*10**(-8)]],float)
89 InitialVector=np.array([6.95896*10**(-4),0,0],float)

91 t=raw_input("Give the end time in seconds ")
   timesec=int(t)
93 timeday=timesec/86400.0

95 elapsed=np.zeros(10)

97 sum=0.0

99 for i in range(0,10):
          ti = time.clock()
101
          FinalConc = CRAM_16(BatemanMatrix,timesec,InitialVector)
103
          elapsed[i] = time.clock() - ti
105       sum=sum+elapsed[i]

107 sum=sum-max(elapsed)-min(elapsed)

109 ExecTime=sum/8.0

111 print ("The nuclide inventories for t=" + str(timeday) + " days ")
   print(FinalConc)
113
   print("The execution time is ")
115 print(ExecTime)
```

### B.1.4 RadauIIA method

```
1 import numpy as np

3 from assimulo.solvers import Radau5ODE
  from assimulo.problem import Explicit_Problem
5
  #Define the right-hand side
7 def rhs(t,y):
          A=np.array([[-1.83163*10**(-12),0,0],[1.83163*10**(-12),-1.60035*10**(-6),0],
9         [0,1.60035*10**(-6),-5.79764*10**(-8)]],float)
          yd=np.dot(A,y)
11        return np.array([yd])

13 y0=np.array([[6.95896*10**(-4)],[0],[0]],float)
   t0=0.0
15
  #Define an Assimlo problem
17 mod = Explicit_Problem(rhs,y0,t0)

19 #Define an explicit solver
   sim = Radau5ODE(mod
21
  #Set the parameter
23 sim.atol = 1e-4
```

```
sim.rtol = 1e-4

#Simulate
t, y = sim.simulate(7776000.0)

print(y[-1])
```

## B.2 The decay problem and the fresh fuel problem

### B.2.1 The scaling and squaring algorithm

```
import numpy as np
from scipy import sparse
import time
import string

# ---------------------------------------------

name=raw_input("Give the filename where the Bateman matrix is saved ")
f = open(name, 'r')

size=0
for line in f:
        S=string.split(line)
        s=float(S[0])
        if s>size:
                size=s
        st=float(S[1])
        if st>size:
        size=st

f.close()

matrix=np.zeros((size,size))

g = open(name, 'r')

for line in g:
        T=string.split(line)
        rownr=int(T[0])
        columnnr=int(T[1])
        element=float(T[2])
        matrix[rownr-1,columnnr-1]=element

g.close()

# ---------------------------------------------

t=raw_input("Give the end time in seconds ")
timesec=int(t)
timeday=timesec/86400.0

# ---------------------------------------

matrixnew=matrix*timesec
matrixspars=sparse.csc_matrix(matrixnew)
```

```
47 # ----------------------------------------------------

49 namevec=raw_input("Give the filename where the initial concentration vector is saved ")
   number=np.loadtxt(namevec,usecols=(0,))
51 #Read the first column of the file

53 m=len(number)
   #The number of rows of the initial vector
55
   elemvec=np.loadtxt(namevec,usecols=(1,))
57 #Read the second column of the file

59 InitialVect=np.zeros(size)

61 for j in range(0,m):
           v=number[j]
63         b=elemvec[j]
           InitialVect[v-1]=b
65
   InitialVect = InitialVect.reshape(size,1)
67
   # ----------------------------------------------------
69
   ti = time.clock()
71
   ExpMatrix=sparse.linalg.expm(matrixspars)
73 ResVec=sparse.csc_matrix.dot(ExpMatrix,InitialVect)

75 elapsed = time.clock() - ti

77 print("The execution time is ")
   print(elapsed)
79
   nameresult=raw_input("Give the filename where the result will be saved ")
81 f = open(nameresult,"w")

83 sizeResVec=len(ResVec)

85 for i in range(0,sizeResVec):
           f.write(str(i+1)+'\t'+str(ResVec[i][0])+'\n')
87
   size=int(size)
89 for j in range(sizeResVec,size):
           f.write(str(j+1)+'\t'+str(0.000)+'\n')
91
   f.close()
```

### B.2.2 Chebyshev Rational Approximation method

```
import time
2 import numpy as np
  import scipy.sparse as sp
4 from scipy.sparse.linalg import spsolve
  from scipy import sparse
6 import string
```

```
8  def _CRAM(A, t, n0, alpha, theta):
           """ Implementation of the CRAM method for a certain
10         approximation defined by the alpha and theta coefficients.
           It calculates $n = n_0 \cdot \exp (At)$.
12
           It implements equation (10) from
14
           Maria Pusa, "Rational Approximations to the Matrix Exponential
16         in Burnup Calculations", Nuclear Science and Engineering, 169,
           p.155-167, 2011
18
           Args:
20         A (np.array): A square matrix, usually sparse
           t (float): end time for the integration
22         n0 (np.array): the initial condition vector
           alpha (np.array): the alpha coefficients to be used
24         theta (np.array): the theta coefficients to be used

26         Returns:
           n (np.array): the solution at time 't'
28         """

30         szi, szj = A.shape
           if (szi != szj):
32                 raise ValueError('Sorry, only square problems allowed')

34         At = sp.csr_matrix(A * t)
           n = alpha[0] * n0
36         Z = np.zeros_like(n)

38         tel=0

40         for j, t in enumerate(theta):
                   B = At - sp.eye(szi) * theta[j]
42                 Z += alpha[j+1] * spsolve(B, n0)

44         n += 2.0 * Z

46         return n.real

48 def CRAM_16(A, t, n0):
           """ Implementation of the 16th order CRAM method.
50         It calculates $n = n_0 \cdot \exp (At)$ at different time steps
           $t$.
52
           It implements equation (10) using the coefficients in
54         table II from

56         Maria Pusa, "Rational Approximations to the Matrix Exponential
           in Burnup Calculations", Nuclear Science and Engineering, 169,
58         p.155-167, 2011

60         Args:
           A (np.array): A square matrix, usually sparse
```

```
62          t (float): end time for the integration
            n0 (np.array): the initial condition vector
64
            Returns:
66          n (np.array): the solution at time 't'
            """
68
            alpha = np.array([+2.124853710495224e-16+0.000000000000000e+00j,
70          -5.090152186522492e-07-2.422001765285229e-05j,
            +2.115174218246603e-04+4.389296964738067e-03j,
72          +1.133977517848393e+02+1.019472170421586e+02j,
            +1.505958527002347e+01-5.751405277642182e+00j,
74          -6.450087802553965e+01-2.245944076265210e+02j,
            -1.479300711355800e+00+1.768658832378294e+00j,
76          -6.251839246320792e+01-1.119039109428323e+01j,
            +4.102313683541002e-02-1.574346617345547e-01j])
78
            theta = np.array([-1.084391707869699e+01+1.927744616718165e+01j,
80          -5.264971343442647e+00+1.622022147316793e+01j,
            +5.948152268951177e+00+3.587457362018322e+00j,
82          +3.509103608414918e+00+8.436198985884374e+00j,
            +6.416177699099435e+00+1.194122393370139e+00j,
84          +1.419375897185666e+00+1.092536348449672e+01j,
            +4.993174737717997e+00+5.996881713603942e+00j,
86          -1.413928462488886e+00+1.349772569889275e+01j])

88          return _CRAM(A, t, n0, alpha, theta)

90  #-----------------------------------------------------

92  name=raw_input("Give the filename where the Bateman matrix is saved ")
    f = open(name, 'r')
94
    size=0
96  for line in f:
            S=string.split(line)
98          s=float(S[0])
            if s>size:
100                 size=s
            st=float(S[1])
102         if st>size:
                    size=st
104
    f.close()
106
    batemanmatrix=np.zeros((size,size))
108
    g = open(name, 'r')
110
    for line in g:
112         T=string.split(line)
            rownr=int(T[0])
114         columnnr=int(T[1])
            element=float(T[2])
116         batemanmatrix[rownr-1,columnnr-1]=element
```

```
118 g.close()

120 #-----------------------------------------------

122 batemanmatrixspars=sparse.csr_matrix(batemanmatrix)
    #To make the Bateman matrix sparse
124
    # ----------------------------------------------
126
    namevec=raw_input("Give the filename where the initial concentration vector is saved ")
128 number=np.loadtxt(namevec,usecols=(0,))
    #Read the first column of the file
130
    m=len(number)
132 #The number of rows of the initial vector

134 elemvec=np.loadtxt(namevec,usecols=(1,))
    #Read the second column of the file
136
    InitialVect=np.zeros(size)
138
    for j in range(0,m):
140         v=number[j]
            b=elemvec[j]
142         InitialVect[v-1]=b

144 #-----------------------------------------

146 t=raw_input("Give the end time in seconds ")
    timesec=int(t)
148 timeday=timesec/86400.0

150 # ----------------------------------------------

152 ti = time.clock()

154 ResVect = CRAM_16(batemanmatrixspars,timesec,InitialVect)

156 elapsed = time.clock() - ti

158 print("-----------------")

160 print("The execution time is ")
    print(elapsed)
162
    nameresult=raw_input("Give the filename where the result will be saved ")
164 f = open(nameresult,"w")

166 sizeRes=len(ResVect)

168 for i in range(0,sizeRes):
        f.write(str(i+1)+'\t'+str(ResVect[i])+'\n')
170
    f.close()
```

### B.2.3 RadauIIA method in Python

```
import numpy as np
import string
from assimulo.solvers import Radau5ODE
from assimulo.problem import Explicit_Problem

#-----------------------------------------------
name=raw_input("Give the filename where the Bateman matrix is saved ")
f = open(name, 'r')

size=0
for line in f:
        S=string.split(line)
        s=float(S[0])
        if s>size:
                size=s
        st=float(S[1])
        if st>size:
                size=st

f.close()

batemanmatrix=np.zeros((size,size))

g = open(name, 'r')

for line in g:
        T=string.split(line)
        rownr=int(T[0])
        columnnr=int(T[1])
        element=float(T[2])
        batemanmatrix[rownr-1,columnnr-1]=element

g.close()

#-----------------------------------------------

namevec=raw_input("Give the filename where the initial concentration vector is saved ")
number=np.loadtxt(namevec,usecols=(0,))
#Read the first column of the file

m=len(number)
#The number of rows of the initial vector

elemvec=np.loadtxt(namevec,usecols=(1,))
#Read the second column of the file

InitialVect=np.zeros(size)

for j in range(0,m):
        v=number[j]
        b=elemvec[j]
        InitialVect[v-1]=b
```

```
InitialVect = InitialVect.reshape(size,1)

# -----------------------------------------------

#Define the right-hand side
        def rhs(t,y):
        yd=np.dot(batemanmatrix,y)

        return np.array([yd])

y0=InitialVect
t0=0.0

#Define the jacobian
def jac(t,y):
        return batemanmatrix

#Define an Assimlo problem
mod = Explicit_Problem(rhs,y0,t0)
mod.jac = jac

#Define an explicit solver
sim = Radau5ODE(mod)

#Set the parameter
sim.atol = 1e-4
sim.rtol = 1e-4
sim.inith = 1e-6
sim.usejac = True

#Simulate
t, y = sim.simulate(7776000.0)

finalRes=y[-1]

f = open("ResRadau90d.txt","w")

sizeFinalRes=len(finalRes)

for i in range(0,sizeFinalRes):
        f.write(str(i+1)+'\t'+str(finalRes[i])+'\n')

f.close()
```

### B.2.4 RadauIIA method in Fortran

#### B.2.4.1 Standard multiplication of a matrix with a vector

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C --- DRIVER FOR RADAU5 AT BATEMAN EQUATIONS
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c link dr_radau radau lapack lapackc dc_lapack
      IMPLICIT REAL*8 (A-H,O-Z)
C --- PARAMETERS FOR RADAU (FULL JACOBIAN)
      PARAMETER (ND=3771,NS=3,LWORK=(NS+1)*ND*ND+(3*NS+3)*ND+20,
     &           LIWORK=(2+(NS-1)/2)*ND+20)
```

```
9           DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
            EXTERNAL FUNCT,JAC
11  C --- DIMENSION OF THE SYSTEM
            N=3771
13  C --- COMPUTE THE JACOBIAN ANALYTICALLY
            IJAC=1
15  C --- JACOBIAN IS A FULL MATRIX
            MLJAC=N
17  C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
            IMAS=0
19  C --- OUTPUT ROUTINE IS NOT USED DURING INTEGRATION
            IOUT=0
21  C --- INITIAL VALUES
            X=0.0D0
23          DO I=1,N
                    Y(I)=0.0D0
25          END DO
            OPEN (3,file='decay_intvect.txt')
27          DO J=1,2451
                    READ(3,*) IND,VAL
29                  Y(IND)=VAL
            END DO
31          CLOSE(3)
    C --- ENDPOINT OF INTEGRATION
33          XEND=7776000.0D0
    C --- REQUIRED TOLERANCE
35          RTOL=1.0D-4
            ATOL=1.0D0*RTOL
37          ITOL=0
    C --- INITIAL STEP SIZE
39          H=1.0D-6
    C --- SET DEFAULT VALUES
41          DO I=1,20
                    IWORK(I)=0
43                  WORK(I)=0.D0
            END DO
45  C --- MEASURE THE TIME
             call cpu_time(start)
47  C --- CALL OF THE SUBROUTINE RADAU
            CALL RADAU5(N,FUNCT,X,Y,XEND,H,
49          &                   RTOL,ATOL,ITOL,
            &                   JAC,IJAC,MLJAC,MUJAC,
51          &                   FUNCT,IMAS,MLMAS,MUMAS,
            &                   SOLOUT,IOUT,
53          &                   WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
    C --- MEASURE THE TIME
55           call cpu_time(finish)
            write(*,*) "The elapsed time = ", finish-start
57  C --- PRINT FINAL SOLUTION
            OPEN (5,file='radau_decay_90d.txt')
59          DO K=1,N
                    WRITE(5,900) K, Y(K)
61          END DO
      900   FORMAT (I5,5x,E18.12)
63          CLOSE(5)
```

```
C --- PRINT STATISTICS
      WRITE (6,90) RTOL
 90   FORMAT('         rtol=',D8.2)
      WRITE (6,91) (IWORK(J),J=14,20)
 91   FORMAT(' fcn=',I5,' jac=',I4,' step=',I4,' accpt=',I4,
     &       ' rejct=',I3,' dec=',I4,' sol=',I5)
      STOP
      END
C
      SUBROUTINE FUNCT(N,X,Y,F,RPAR,IPAR)
C --- THE RIGHT-HAND SIDE OF THE BATEMAN EQUATIONS
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),F(N)
      DO I=1,N
            F(I)=0.0D0
      END DO
      OPEN (2,file='decay_matrix.txt')
      DO J=1,17533
            READ(2,*) IND_ROW,IND_COL,VALUE
            F(IND_ROW)=F(IND_ROW)+VALUE*Y(IND_COL)
      END DO
      CLOSE(2)
      RETURN
      END
C
      SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
C --- THE JACOBIAN OF BATEMAN EQUATIONS
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),DFY(LDFY,N)
      DO I=1,N
            DO J=1,N
                  DFY(I,J)=0.0D0
            END DO
      END DO
      OPEN (4,file='decay_matrix.txt')
      DO K=1,17533
            READ(4,*) IND_R,IND_C,VALUE_EL
            DFY(IND_R,IND_C)=VALUE_EL
      END DO
      CLOSE(4)
      RETURN
      END
```

#### B.2.4.2 Exploiting the Compressed Row Storage

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * *
C --- DRIVER FOR RADAU5 AT BATEMAN EQUATIONS
C * * * * * * * * * * * * * * * * * * * * * * * * * * *
c link dr_radau radau lapack lapackc dc_lapack
      IMPLICIT REAL*8 (A-H,O-Z)
C --- PARAMETERS FOR RADAU (FULL JACOBIAN)
      PARAMETER (ND=3771,NS=7,LWORK=(NS+1)*ND*ND+(3*NS+3)*ND+20,
     &           LIWORK=(2+(NS-1)/2)*ND+20)
      DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
      EXTERNAL FUNCT,JAC
```

```
C --- DIMENSION OF THE SYSTEM
      N=3771
C --- COMPUTE THE JACOBIAN ANALYTICALLY
      IJAC=1
C --- JACOBIAN IS A FULL MATRIX
      MLJAC=N
C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
      IMAS=0
C --- OUTPUT ROUTINE IS NOT USED DURING INTEGRATION
      IOUT=0
C --- INITIAL VALUES
      X=0.0D0
      DO I=1,N
            Y(I)=0.0D0
      END DO
      OPEN (3,file='decay_intvect.txt')
      DO J=1,2451
            READ(3,*) IND,VAL
            Y(IND)=VAL
      END DO
      CLOSE(3)
C --- ENDPOINT OF INTEGRATION
      XEND=7776000.0D0
C --- REQUIRED TOLERANCE
      RTOL=1.0D-4
      ATOL=1.0D0*RTOL
      ITOL=0
C --- INITIAL STEP SIZE
      H=1.0D-6
C --- SET DEFAULT VALUES
      DO I=1,20
            IWORK(I)=0
            WORK(I)=0.D0
      END DO
C --- MEASURE THE TIME
       call cpu_time(start)
C --- CALL OF THE SUBROUTINE RADAU
      CALL RADAU5(N,FUNCT,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JAC,IJAC,MLJAC,MUJAC,
     &                  FUNCT,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
C --- MEASURE THE TIME
       call cpu_time(finish)
       write(*,*) "The elapsed time = ", finish-start
C --- PRINT FINAL SOLUTION
      OPEN (5,file='radau_decay_90d2.txt')
      DO K=1,N
            WRITE(5,*) K, Y(K)
      END DO
      CLOSE(5)
C --- PRINT STATISTICS
      WRITE (6,90) RTOL
 90   FORMAT('      rtol=',D8.2)
```

```
      WRITE (6,91) (IWORK(J),J=14,20)
91    FORMAT(' fcn=',I5,' jac=',I4,' step=',I4,' accpt=',I4,
     &       ' rejct=',I3,' dec=',I4,' sol=',I5)
      STOP
      END
C
      SUBROUTINE FUNCT(N,X,Y,F,RPAR,IPAR)
C --- THE RIGHT-HAND SIDE OF THE BATEMAN EQUATIONS
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),F(N)
      INTEGER M(17533),IND_COL(17533),IND_ROW(17533)
      DIMENSION VALUE(17533)
      N_IND_ROW=2
C --- NUMBER TO COUNT HOW MANY NUMBERS THERE ARE IN IND_ROW
      N_LIST=0
C --- NUMBER TO COUNT HOW MANY NUMBERS THERE ARE IN THE LIST
      OPEN(2,file='decay_matrix.txt')
      IND_ROW(1)=1
      DO K=1,17534
            READ(2,*) M(K),L,r
            VALUE(K)=r
            IND_COL(K)=L
            N_LIST=N_LIST+1
            IF (K.NE.1.AND.M(K).NE.M(K-1)) THEN
                  IND_ROW(N_IND_ROW)=N_LIST
                  N_IND_ROW=N_IND_ROW+1
            END IF
      END DO
      IND_ROW(N_IND_ROW)=N_LIST+1
      CLOSE(2)
      DO I=1,N
            F(I)=0.0D0
            DO J=IND_ROW(I),IND_ROW(I+1)-1
                  F(I)=F(I)+VALUE(J)*Y(IND_COL(J))
            END DO
      END DO
      RETURN
      END
      SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
C --- THE JACOBIAN OF THE BATEMAN EQUATIONS
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),DFY(LDFY,N)
      DO I=1,N
            DO J=1,N
                  DFY(I,J)=0.0D0
            END DO
      END DO
      OPEN (4,file='decay_matrix.txt')
      DO K=1,17534
            READ(4,*) IND_R,IND_C,VALUE_EL
            DFY(IND_R,IND_C)=VALUE_EL
      END DO
      CLOSE(4)
      RETURN
      END
```

#### B.2.4.3 Using the subroutine DGEMV

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C --- DRIVER FOR RADAU5 AT BATEMAN EQUATIONS
C * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c link dr_radau radau lapack lapackc dc_lapack
      IMPLICIT REAL*8 (A-H,O-Z)
C --- PARAMETERS FOR RADAU (FULL JACOBIAN)
      PARAMETER (ND=3771,NS=7,LWORK=(NS+1)*ND*ND+(3*NS+3)*ND+20,
     &           LIWORK=(2+(NS-1)/2)*ND+20)
      DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK)
      EXTERNAL FUNCT,JAC
C --- PARAMETER IN THE DIFFERENTIAL EQUATION, NO MEANING IN THIS EXAMPLE
      RPAR=1.0D-6
C --- DIMENSION OF THE SYSTEM
      N=3771
C --- COMPUTE THE JACOBIAN ANALYTICALLY
      IJAC=1
C --- JACOBIAN IS A FULL MATRIX
      MLJAC=N
C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
      IMAS=0
C --- OUTPUT ROUTINE IS NOT USED DURING INTEGRATION
      IOUT=0
C --- INITIAL VALUES
      X=0.0D0
      DO I=1,N
            Y(I)=0.0D0
      END DO
      OPEN (3,file='decay_intvect.txt')
      DO J=1,2451
            READ(3,*) IND,VAL
            Y(IND)=VAL
      END DO
      CLOSE(3)
C --- ENDPOINT OF INTEGRATION
      XEND=7776000.0D0
C --- REQUIRED TOLERANCE
      RTOL=1.0D-4
      ATOL=1.0D0*RTOL
      ITOL=0
C --- INITIAL STEP SIZE
      H=1.0D-6
C --- SET DEFAULT VALUES
      DO I=1,20
            IWORK(I)=0
            WORK(I)=0.D0
      END DO
C --- MEASURE THE TIME
      call cpu_time(start)
C --- CALL OF THE SUBROUTINE RADAU
      CALL RADAU5(N,FUNCT,X,Y,XEND,H,
     &            RTOL,ATOL,ITOL,
     &            JAC,IJAC,MLJAC,MUJAC,
     &            FUNCT,IMAS,MLMAS,MUMAS,
```

```
54      &                       SOLOUT,IOUT,
        &                       WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
56  C --- MEASURE THE TIME
            call cpu_time(finish)
58          write(*,*) "The elapsed time = ", finish-start
    C --- PRINT FINAL SOLUTION
60          OPEN (5,file='radau_decay_routines_90d.txt')
            DO K=1,N
62                  WRITE(5,*) K, Y(K)
            END DO
64          CLOSE(5)
    C       WRITE (6,99) X,Y
66          C99     FORMAT(1X,'X =',E18.10,'     Y =',3E18.10)
    C --- PRINT STATISTICS
68          WRITE (6,90) RTOL
     90     FORMAT('          rtol=',D8.2)
70          WRITE (6,91) (IWORK(J),J=14,20)
     91     FORMAT(' fcn=',I5,' jac=',I4,' step=',I4,' accpt=',I4,
72         &       ' rejct=',I3,' dec=',I4,' sol=',I5)
            STOP
74          END
    C
76          SUBROUTINE FUNCT(N,X,Y,F,RPAR,IPAR)
    C --- THE RIGHT-HAND SIDE OF THE BATEMAN EQUATIONS
78          IMPLICIT REAL*8 (A-H,O-Z)
            DIMENSION Y(N),F(N), A(N,N)
80          DO I=1,3771
                    DO J=1,3771
82                          A(I,J)=0.0D0
                    END DO
84          END DO
            OPEN (2,file='decay_matrix.txt')
86          DO I=1,17533
                    READ(2,*) IND_ROW,IND_COL,VALUE
88                  A(IND_ROW,IND_COL)=VALUE
            END DO
90          CLOSE(2)

92          INCX=1
            INCY=1
94          LDA=3771
            M_DGEMV=3771
96          N_DGEMV=3771
            ALPHA=1.0D0
98          BETA=0.0D0

100         CALL DGEMV('N',M_DGEMV,N_DGEMV,ALPHA,A,LDA,Y,INCX,BETA,F,INCY)

102         RETURN
            END
104 C
            SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
106 C --- THE JACOBIAN OF THE BATEMAN EQUATIONS
            IMPLICIT REAL*8 (A-H,O-Z)
108         DIMENSION Y(N),DFY(LDFY,N),A(LDFY,N)
```

```
      DO I=1,N
110         DO J=1,N
                DFY(I,J)=0.0D0
112         END DO
      END DO
114   OPEN (4,file='decay_matrix.txt')
      DO K=1,17533
116         READ(4,*) IND_R,IND_C,VAL_EL
            DFY(IND_R,IND_C)=VAL_EL
118   END DO
      CLOSE(4)
120   RETURN
      END
```

# C Codes belonging to the section on the implementation of methods using Padé approximations to the exponential

## C.1 Implicit Euler method

```
      PROGRAM Euler

      IMPLICIT NONE
      INTEGER IERR, I, J, K, L
      INTEGER N, NZM, NZV
      INTEGER, ALLOCATABLE, DIMENSION(:) :: IND_ROW, IND_COL
      REAL, ALLOCATABLE, DIMENSION(:) :: VALUE_MATRIX
      INTEGER END_TIME, NUMB_STEPS, IND_VECTOR
      REAL H, VALUE_VECTOR
      REAL start, finish
      INTEGER, ALLOCATABLE, DIMENSION(:):: IND_DIAGN
      INTEGER, ALLOCATABLE, DIMENSION(:):: IND
      INTEGER ND, NNE, NC, NZME

      INCLUDE 'mpif.h'
      INCLUDE 'dmumps_struc.h'
      TYPE (DMUMPS_STRUC) mumps_par
      CALL MPI_INIT(IERR)
C Define a communicator for the package.
      mumps_par%COMM = MPI_COMM_WORLD
C  Initialize an instance of the package
C  for LU factorization (sym = 0, with working host)
      mumps_par%JOB = -1
      mumps_par%SYM = 0
      mumps_par%PAR = 1
      CALL DMUMPS(mumps_par)
      IF (mumps_par%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &              "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
     &              "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
        GOTO 500
      END IF

      mumps_par%ICNTL(4)=-1

      call cpu_time(start)

C  Define problem on the host (processor 0)

C INITIALIZE THE PROBLEM
C POLONIUM PROBLEM
C     N: DIMENSION OF THE SYSTEM
C     N=3
C     NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
C     NZM=5
C     NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
C     NZV=3
C     END_TIME: THE END TIME
C     END_TIME=7776000
```

```
C       THE STEPSIZE
C       H=1000

C FRESH FUEL PROBLEM
C       N: DIMENSION OF THE SYSTEM
C       N=3701
C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
C       NZM=42464
C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
C       NZV=5
C       END_TIME: THE END TIME
C       END_TIME=518400
C       THE STEPSIZE
C       H=10

C BURNED FUEL PROBLEM
C       N: DIMENSION OF THE SYSTEM
        N=3701
C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
        NZM=44357
C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
        NZV=1633
C       END_TIME: THE END TIME
        END_TIME=2972160
C       THE STEPSIZE
        H=10

C THE NUMBER OF STEPS
        NUMB_STEPS=int(END_TIME/H)

C  INCLUDE THE MATRIX
        ALLOCATE( IND_ROW ( NZM ) )
        ALLOCATE( IND_COL ( NZM ) )
        ALLOCATE( VALUE_MATRIX ( NZM ) )
C       OPEN(2,FILE='test_po.txt')
C       OPEN(2,FILE='matrixfreshfuel.txt')
        OPEN(2,FILE='matrixburnedfuel.txt')
        DO I=1,NZM
          READ(2,*) IND_ROW(I), IND_COL(I), VALUE_MATRIX(I)
        END DO
        CLOSE(2)

C --- CONTROL OF THE DIAGONAL. IF THE BATEMAN MATRIX HASN'T A VALUE ON
C --- THE DIAGIONAL, THE LEFT-HAND SIDE WILL HAVE THE VALUE 1
C IND_DIAGN will contain the diagonal positions which are filled
C IF THERE ARE ZEROS ON THE DIAGONAL, IND_DIAGN WILL CONTAIN
C A NUMBER GREATER THAN N
        ALLOCATE( IND_DIAGN (N) )
        DO I=1,N
          IND_DIAGN(I)=N+10
        END DO
C ND: NUMBER OF ELEMENTS ON THE DIAGONAL
        ND=0
C NNE: NUMBER OF NO ELEMENTS ON THE DIAGONAL
        NNE=1
```

```
105         DO I=1,NZM
              IF (IND_ROW(I).EQ.IND_COL(I)) THEN
107             ND=ND+1
                IND_DIAGN(ND)=IND_ROW(I)
109           END IF
            END DO
111   C IND: A VECTOR CONTAINING THE INDICES OF THE DIAGONAL POSITIONS
      C        WITH A ZERO NUMBER
113         ALLOCATE(IND(N-ND))
      C LESS ELEMENTS ON THE DIAGONAL (ND-NUMBER OF ELEMENTS ON DIAGONAL)
115   C THAN THE SIZE OF THE PROBLEM (N)
            IF (ND.NE.N) THEN
117   C CHECK EVERY DIAGONAL POSITION
              DO K=1,N
119             NC=0
                DO L=1,ND
121   C CHECK IF THERE CORRESPONDS AN INDICE OF THE BATEMAN MATRIX WITH
      C THE POSITION ON THE DIAGONAL
123   C AND IF THERE IS NO CORRESPONDENCE, IT WILL BE REMEMBERED
                  IF (IND_DIAGN(L).NE.K) THEN
125                 NC=NC+1
                    IF (NC.GE.ND) THEN
127                   IND(NNE)=K
                      NNE=NNE+1
129                 END IF
                  END IF
131             END DO
              END DO
133         END IF

135   C THE LEFT-HAND SIDE WILL CONTAIN NZME ELEMENTS
            NZME=NZM+NNE-1
137
            mumps_par%N=N
139         mumps_par%NZ=NZME

141   C MAKE THE LEFT-HAND SIDE OF THE PROBLEM
            ALLOCATE( mumps_par%IRN ( NZME ) )
143         ALLOCATE( mumps_par%JCN ( NZME ) )
            ALLOCATE( mumps_par%A ( NZME ) )
145         DO I=1,NZM
              mumps_par%IRN(I)=IND_ROW(I)
147           mumps_par%JCN(I)=IND_COL(I)
              IF (IND_ROW(I).EQ.IND_COL(I)) THEN
149             mumps_par%A(I)=1-H*VALUE_MATRIX(I)
              ELSE
151             mumps_par%A(I)=-H*VALUE_MATRIX(I)
              END IF
153         END DO

155         DO J=1,NNE-1
              mumps_par%IRN(NZM+J)=IND(J)
157           mumps_par%JCN(NZM+J)=IND(J)
              mumps_par%A(NZM+J)=1
159         END DO
```

```
C  INCLUDE THE INITIAL VECTOR
      ALLOCATE( mumps_par%RHS ( mumps_par%N ) )
      DO J=1,mumps_par%N
         mumps_par%RHS(J)=0.0D0
      END DO
C     OPEN(5,FILE='test_po_begin.txt')
C     OPEN(5,FILE='BeginVectorFreshFuel.txt')
      OPEN(5,FILE='BeginVectorBurnedFuel.txt')
      DO I=1,NZV
        READ(5,*) IND_VECTOR, VALUE_VECTOR
         mumps_par%RHS(IND_VECTOR) = VALUE_VECTOR
      END DO
      CLOSE(5)

C  Call package for solution
      mumps_par%JOB = 6
      CALL DMUMPS(mumps_par)
      IF (mumps_par%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
     &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
        GOTO 500
      END IF

C THE STEPS
      DO K=2,NUMB_STEPS
         mumps_par%JOB = 3
         CALL DMUMPS(mumps_par)
         IF (mumps_par%INFOG(1).LT.0) THEN
           WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
     &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
           GOTO 500
         END IF
      END DO

C PRINT THE RESULT IN A TEXT-FILE
C     OPEN(4,FILE='resultaat_po_euler.txt')
C     OPEN(4,FILE='resultaat_freshfuel_euler.txt')
      OPEN(4,FILE='resultaat_burnedfuel_euler.txt')

      DO I=1,mumps_par%N
         WRITE(4,900) I, mumps_par%RHS(I)
      END DO
  900 FORMAT(I5.5x,E19.8E3)
      CLOSE(4)

      call cpu_time(finish)
      write(*,*) "The elapsed time = ", finish-start

C  Deallocate user data
      IF ( mumps_par%MYID .eq. 0 )THEN
        DEALLOCATE( mumps_par%IRN )
        DEALLOCATE( mumps_par%JCN )
        DEALLOCATE( mumps_par%A )
```

```
215           DEALLOCATE( mumps_par%RHS )
        END IF
217     DEALLOCATE(IND_ROW)
        DEALLOCATE(IND_COL)
219     DEALLOCATE(VALUE_MATRIX)
        DEALLOCATE(IND_DIAGN)
221     DEALLOCATE(IND)

223   C Destroy the instance (deallocate internal data structures)
        mumps_par%JOB = -2
225     CALL DMUMPS(mumps_par)
        IF (mumps_par%INFOG(1).LT.0) THEN
227       WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
229     &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
          GOTO 500
231     END IF
   500  CALL MPI_FINALIZE(IERR)
233     STOP
        END
```

## C.2 The Trapezoidal rule

```
1       PROGRAM TRAPEZIUM

3       IMPLICIT NONE
        INTEGER IERR, I, J, K, L
5       INTEGER N, NZM, NZV
        INTEGER, ALLOCATABLE, DIMENSION(:) :: IND_ROW, IND_COL
7       INTEGER, ALLOCATABLE, DIMENSION(:):: imrhs_row, imrhs_col
        REAL, ALLOCATABLE, DIMENSION(:) :: VALUE_MATRIX, matrix_rhs, RHS
9       INTEGER END_TIME, NUMB_STEPS, IND_VECTOR
        REAL H, VALUE_VECTOR
11      REAL start, finish
        INTEGER, ALLOCATABLE, DIMENSION(:):: IND_DIAGN
13      INTEGER, ALLOCATABLE, DIMENSION(:):: IND
        INTEGER ND, NNE, NC, NZME
15
        INCLUDE 'mpif.h'
17      INCLUDE 'dmumps_struc.h'
        TYPE (DMUMPS_STRUC) mumps_par
19      CALL MPI_INIT(IERR)
  C Define a communicator for the package.
21      mumps_par%COMM = MPI_COMM_WORLD
  C  Initialize an instance of the package
23C  for LU factorization (sym = 0, with working host)
        mumps_par%JOB = -1
25      mumps_par%SYM = 0
        mumps_par%PAR = 1
27      CALL DMUMPS(mumps_par)
        IF (mumps_par%INFOG(1).LT.0) THEN
29        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
31      &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
          GOTO 500
33      END IF
```

```
35          mumps_par%ICNTL(4)=-1

37          call cpu_time(start)

39  C   Define problem on the host (processor 0)

41  C INITIALIZE THE PROBLEM
    C POLONIUM PROBLEM
43  C       N: DIMENSION OF THE SYSTEM
    C       N=3
45  C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
    C       NZM=5
47  C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
    C       NZV=3
49  C       END_TIME: THE END TIME
    C       END_TIME=7776000
51  C       THE STEPSIZE
    C       H=1000
53
    C FRESH FUEL PROBLEM
55  C       N: DIMENSION OF THE SYSTEM
    C       N=3701
57  C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
    C       NZM=42464
59  C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
    C       NZV=5
61  C       END_TIME: THE END TIME
    C       END_TIME=518400
63  C       THE STEPSIZE
    C       H=10
65
    C BURNED FUEL PROBLEM
67  C       N: DIMENSION OF THE SYSTEM
            N=3701
69  C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
            NZM=44357
71  C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
            NZV=1633
73  C       END_TIME: THE END TIME
            END_TIME=2972160
75  C       THE STEPSIZE
            H=10
77
    C THE NUMBER OF STEPS
79          NUMB_STEPS=int(END_TIME/H)

81  C   INCLUDE THE MATRIX
            ALLOCATE( IND_ROW ( NZM ) )
83          ALLOCATE( IND_COL ( NZM ) )
            ALLOCATE( VALUE_MATRIX ( NZM) )
85  C       OPEN(2,FILE='test_po.txt')
    C       OPEN(2,FILE='matrixfreshfuel.txt')
87          OPEN(2,FILE='matrixburnedfuel.txt')
```

```
89          DO I=1,NZM
              READ(2,*) IND_ROW(I), IND_COL(I), VALUE_MATRIX(I)
91          END DO
            CLOSE(2)
93
    C ---- CONTROL OF THE DIAGONAL. IF THE BATEMAN MATRIX HASN'T A VALUE ON
95  C ---- THE DIAGIONAL, THE LEFT-HAND SIDE WILL HAVE THE VALUE 1
    C IND_DIAGN will contain the diagonal positions which are filled
97  C IF THERE ARE ZEROS ON THE DIAGONAL, IND_DIAGN WILL CONTAIN
    C A NUMBER GREATER THAN N
99          ALLOCATE( IND_DIAGN (N) )
            DO I=1,N
101           IND_DIAGN(I)=N+10
            END DO
103 C ND: NUMBER OF ELEMENTS ON THE DIAGONAL
            ND=0
105 C NNE: NUMBER OF NO ELEMENTS ON THE DIAGONAL
            NNE=1
107         DO I=1,NZM
              IF (IND_ROW(I).EQ.IND_COL(I)) THEN
109             ND=ND+1
                IND_DIAGN(ND)=IND_ROW(I)
111           END IF
            END DO
113 C IND: A VECTOR CONTAINING THE INDICES OF THE DIAGONAL POSITIONS
    C         WITH A ZERO NUMBER
115         ALLOCATE(IND(N-ND))
    C LESS ELEMENTS ON THE DIAGONAL (ND-NUMBER OF ELEMENTS ON DIAGONAL)
117 C THAN THE SIZE OF THE PROBLEM (N)
            IF (ND.NE.N) THEN
119 C CHECK EVERY DIAGONAL POSITION
              DO K=1,N
121             NC=0
                DO L=1,ND
123 C CHECK IF THERE CORRESPONDS AN INDICE OF THE BATEMAN MATRIX WITH
    C THE POSITION ON THE DIAGONAL
125 C AND IF THERE IS NO CORRESPONDENCE, IT WILL BE REMEMBERED
                  IF (IND_DIAGN(L).NE.K) THEN
127                 NC=NC+1
                    IF (NC.GE.ND) THEN
129                   IND(NNE)=K
                      NNE=NNE+1
131                 END IF
                  END IF
133             END DO
              END DO
135         END IF


137
    C THE LEFT-HAND SIDE WILL CONTAIN NZME ELEMENTS
139
            NZME=NZM+NNE-1
141
            mumps_par%N=N
143         mumps_par%NZ=NZME
```

```
145 C MAKE THE LEFT-HAND SIDE OF THE PROBLEM
        ALLOCATE( mumps_par%IRN ( mumps_par%NZ ) )
147     ALLOCATE( mumps_par%JCN ( mumps_par%NZ ) )
        ALLOCATE( mumps_par%A ( mumps_par%NZ ) )
149     DO I=1,NZM
          mumps_par%IRN(I)=IND_ROW(I)
151       mumps_par%JCN(I)=IND_COL(I)
          IF (IND_ROW(I).EQ.IND_COL(I)) THEN
153         mumps_par%A(I)=1-((H*VALUE_MATRIX(I))/2.0)
          ELSE
155         mumps_par%A(I)=-H*VALUE_MATRIX(I)/2.0
          END IF
157     END DO

159     DO J=1,NNE-1
          mumps_par%IRN(NZM+J)=IND(J)
161       mumps_par%JCN(NZM+J)=IND(J)
          mumps_par%A(NZM+J)=1
163     END DO

165 C MAKE THE RIGHT-HAND SIDE MATRIX
        ALLOCATE( imrhs_row ( NZME ) )
167     ALLOCATE( imrhs_col ( NZME ) )
        ALLOCATE( matrix_rhs ( mumps_par%NZ ) )
169     DO I=1,NZM
          imrhs_row(I)=IND_ROW(I)
171       imrhs_col(I)=IND_COL(I)
          IF (IND_ROW(I).EQ.IND_COL(I)) THEN
173         matrix_rhs(I)=1+((H*VALUE_MATRIX(I))/2.0)
          ELSE
175         matrix_rhs(I)=H*VALUE_MATRIX(I)/2.0
          END IF
177     END DO
        DO J=1,NNE-1
179       imrhs_row(NZM+J)=IND(J)
          imrhs_col(NZM+J)=IND(J)
181       matrix_rhs(NZM+J)=1
        END DO
183
  C  INCLUDE THE INITIAL VECTOR
185     ALLOCATE( mumps_par%RHS ( mumps_par%N ) )
        ALLOCATE( RHS ( mumps_par%N ) )
187     DO J=1,mumps_par%N
          RHS(J)=0.0D0
189       mumps_par%RHS(J)=0.0D0
        END DO
191 C     OPEN(5,FILE='test_po_begin.txt')
  C     OPEN(5,FILE='BeginVectorFreshFuel.txt')
193     OPEN(5,FILE='BeginVectorBurnedFuel.txt')
        DO I=1,NZV
195       READ(5,*) IND_VECTOR, VALUE_VECTOR
          RHS(IND_VECTOR) = VALUE_VECTOR
197     END DO
        CLOSE(5)
```

```
199         DO J=1,mumps_par%NZ
              mumps_par%RHS(imrhs_row(J))=mumps_par%RHS(imrhs_row(J))
201         &                 +matrix_rhs(J)*RHS(imrhs_col(J))
            END DO
203
    C   Call package for solution
205         mumps_par%JOB = 6
            CALL DMUMPS(mumps_par)
207         IF (mumps_par%INFOG(1).LT.0) THEN
              WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
209         &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
            &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
211           GOTO 500
            END IF
213
    C THE STEPS
215         DO K=2,NUMB_STEPS
              DO J=1,mumps_par%N
217             RHS(J)=mumps_par%RHS(J)
                mumps_par%RHS(J)=0.0D0
219           END DO
              DO J=1,mumps_par%NZ
221             mumps_par%RHS(imrhs_row(J))=mumps_par%RHS(imrhs_row(J))
            &                 +matrix_rhs(J)*RHS(imrhs_col(J))
223           END DO
              mumps_par%JOB = 3
225           CALL DMUMPS(mumps_par)
              IF (mumps_par%INFOG(1).LT.0) THEN
227             WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
            &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
229         &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
                GOTO 500
231           END IF
            END DO
233
    C PRINT THE RESULT IN A TEXT-FILE
235 C       OPEN(4,FILE='resultaat_po_trapezium.txt')
    C       OPEN(4,FILE='resultaat_freshfuel_trapezium.txt')
237         OPEN(4,FILE='resultaat_burnedfuel_trapezium.txt')
            DO I=1,mumps_par%N
239           WRITE(4,900) I, mumps_par%RHS(I)
            END DO
241       900 FORMAT(I5.5x,E19.8E3)
            CLOSE(4)
243
            call cpu_time(finish)
245         write(*,*) "The elapsed time = ", finish-start

247 C   Deallocate user data
            IF ( mumps_par%MYID .eq. 0 )THEN
249           DEALLOCATE( mumps_par%IRN )
              DEALLOCATE( mumps_par%JCN )
251           DEALLOCATE( mumps_par%A   )
              DEALLOCATE( mumps_par%RHS )
253         END IF
```

```
      DEALLOCATE(IND_ROW)
255   DEALLOCATE(IND_COL)
      DEALLOCATE(VALUE_MATRIX)
257   DEALLOCATE(imrhs_row)
      DEALLOCATE(imrhs_col)
259   DEALLOCATE(matrix_rhs)
      DEALLOCATE(RHS)

261
C  Destroy the instance (deallocate internal data structures)
263   mumps_par%JOB = -2
      CALL DMUMPS(mumps_par)
265   IF (mumps_par%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
267   &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
      &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
269     GOTO 500
      END IF
271  500  CALL MPI_FINALIZE(IERR)
      STOP
273   END
```

## C.3 RadauIIA method implemented using the stability function

```
1     PROGRAM RADAU

3     IMPLICIT NONE
      INTEGER IERR, I, J, K, NONZERO
5     INTEGER IND_ROW, IND_COL
      INTEGER N, NZM, NZV
7     REAL VALUE_MATRIX
      REAL, ALLOCATABLE, DIMENSION(:) :: matrix_rhs, VECTOR
9     REAL, ALLOCATABLE, DIMENSION(:,:) :: MATRIX
      REAL, ALLOCATABLE, DIMENSION(:,:) :: MATRIX2, MATRIX3
11    REAL, ALLOCATABLE, DIMENSION(:,:) :: LHS, RHS
      INTEGER END_TIME, NUMB_STEPS, IND_VECTOR
13    REAL H, VALUE_VECTOR
      REAL start, finish

15
      INCLUDE 'mpif.h'
17    INCLUDE 'dmumps_struc.h'
      TYPE (DMUMPS_STRUC) mumps_par
19    CALL MPI_INIT(IERR)
C Define a communicator for the package.
21    mumps_par%COMM = MPI_COMM_WORLD
C  Initialize an instance of the package
23  C  for LU factorization (sym = 0, with working host)
      mumps_par%JOB = -1
25    mumps_par%SYM = 0
      mumps_par%PAR = 1
27    CALL DMUMPS(mumps_par)
      IF (mumps_par%INFOG(1).LT.0) THEN
29      WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
      &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
31    &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
        GOTO 500
33    END IF
```

```
        mumps_par%ICNTL(4)=-1
        call cpu_time(start)

C  Define problem on the host (processor 0)

C INITIALIZE THE PROBLEM
C       N: DIMENSION OF THE SYSTEM
C       N=3
C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
C       NZM=5
C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
C       NZV=3
C       END_TIME: THE END TIME
C       END_TIME=7776000
C       THE STEPSIZE
C       H=1000

C FRESH FUEL PROBLEM
C       N: DIMENSION OF THE SYSTEM
        N=3701
C       NZM: THE NUMBER OF NONZERO ELEMENTS IN THE MATRIX
        NZM=42464
C       NZV: THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
        NZV=5
C       END_TIME: THE END TIME
        END_TIME=518400
C       THE STEPSIZE
        H=960

C THE NUMBER OF STEPS
        NUMB_STEPS=int(END_TIME/H)

C  INCLUDE THE MATRIX
        ALLOCATE( MATRIX (N,N ) )
C       OPEN(2,FILE='test_po.txt')
        OPEN(2,FILE='matrixfreshfuel.txt')
        DO I=1,N
          DO J=1,N
            MATRIX(I,J)=0.0D0
          END DO
        END DO
        DO I=1,NZM
          READ(2,*) IND_ROW, IND_COL, VALUE_MATRIX
          MATRIX(IND_ROW,IND_COL)=VALUE_MATRIX
        END DO
        CLOSE(2)

C MAKE THE SQUARE MATRIX
        ALLOCATE( MATRIX2 ( N,N ) )
        DO I = 1,N
          DO K = 1,N
            MATRIX2(I,K) = 0.0
            DO J = 1,N
              MATRIX2(I,K) = MATRIX2(I,K) + MATRIX(I,J)*MATRIX(J,K)
            END DO
```

```
89            END DO
          END DO
91
C MAKE THE KUBIC MATRIX
93        ALLOCATE( MATRIX3 (N,N ) )

95        DO I = 1,N
            DO K = 1,N
97            MATRIX3(I,K) = 0.0
              DO J = 1,N
99              MATRIX3(I,K) = MATRIX3(I,K) + MATRIX2(I,J)*MATRIX(J,K)
              END DO
101         END DO
          END DO
103
C MAKE THE LEFT-HAND SIDE OF THE PROBLEM
105       ALLOCATE( LHS (N,N ) )
          DO I=1,N
107         DO J=1,N
              IF (I.EQ.J) THEN
109             LHS(I,J)=1-((3/5.0)*H*MATRIX(I,J))
          &                   +((3/10.0)*(H**2/2.0)*MATRIX2(I,J))
111       &                   -((1/10.0)*(H**3/6.0)*MATRIX3(I,J))
              ELSE
113             LHS(I,J)=(-(3/5.0)*H*MATRIX(I,J))
          &                   +((3/10.0)*(H**2/2.0)*MATRIX2(I,J))
115       &                   -((1/10.0)*(H**3/6.0)*MATRIX3(I,J))
              END IF
117         END DO
          END DO
119
          NONZERO=0
121       DO I=1,N
            DO J=1,N
123           IF (LHS(I,J).NE.(0.0D0)) THEN
                NONZERO=NONZERO+1
125           END IF
            END DO
127       END DO

129       ALLOCATE( mumps_par%IRN ( NONZERO ) )
          ALLOCATE( mumps_par%JCN ( NONZERO ) )
131       ALLOCATE( mumps_par%A ( NONZERO ) )

133       K=1
          DO I=1,N
135         DO J=1,N
              IF (LHS(I,J).NE.(0.0D0)) THEN
137             mumps_par%IRN(K)=I
                mumps_par%JCN(K)=J
139             mumps_par%A(K)=LHS(I,J)
                K=K+1
141           END IF
            END DO
143       END DO
```

```
145 C GIVE DIMENSION PARAMETERS TO MUMPS
        mumps_par%N=N
147     mumps_par%NZ=NONZERO

149 C MAKE THE RIGHT-HAND SIDE OF THE PROBLEM
        ALLOCATE( RHS (N,N ) )
151     DO I=1,N
          DO J=1,N
153         IF (I.EQ.J) THEN
              RHS(I,J)=1+((2/5.0)*H*MATRIX(I,J))
155     &                 +((1/10.0)*(H**2/2.0)*MATRIX2(I,J))
            ELSE
157           RHS(I,J)=((2/5.0)*H*MATRIX(I,J))
        &                 +((1/10.0)*(H**2/2.0)*MATRIX2(I,J))
159         END IF
          END DO
161     END DO

163 C  INCLUDE THE INITIAL VECTOR
        ALLOCATE( mumps_par%RHS ( mumps_par%N ) )
165     ALLOCATE( VECTOR ( mumps_par%N ) )
        DO J=1,mumps_par%N
167       VECTOR(J)=0.0D0
          mumps_par%RHS(J)=0.0D0
169     END DO
C       OPEN(5,FILE='test_po_begin.txt')
171     OPEN(5,FILE='BeginVectorFreshFuel.txt')

173     DO I=1,NZV
          READ(5,*) IND_VECTOR, VALUE_VECTOR
175       VECTOR(IND_VECTOR) = VALUE_VECTOR
        END DO
177     CLOSE(5)
        DO I=1,mumps_par%N
179       DO J=1,mumps_par%N
            mumps_par%RHS(I)=mumps_par%RHS(I)+RHS(I,J)*VECTOR(J)
181       END DO
        END DO
183 C  Call package for solution
        mumps_par%JOB = 6
185     CALL DMUMPS(mumps_par)
        IF (mumps_par%INFOG(1).LT.0) THEN
187       WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
189     &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
          GOTO 500
191     END IF

193 C THE STEPS
        DO K=2,NUMB_STEPS
195       DO J=1,mumps_par%N
            VECTOR(J)=mumps_par%RHS(J)
197         mumps_par%RHS(J)=0.0D0
          END DO
```

```
199           DO I=1,mumps_par%N
                DO J=1,mumps_par%N
201               mumps_par%RHS(I)=mumps_par%RHS(I)+RHS(I,J)*VECTOR(J)
                END DO
203           END DO
              mumps_par%JOB = 3
205           CALL DMUMPS(mumps_par)
              IF (mumps_par%INFOG(1).LT.0) THEN
207             WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
209     &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
                GOTO 500
211           END IF
        END DO
213
    C PRINT THE RESULT IN A TEXT-FILE
215
    C       OPEN(4,FILE='resultaat_po_radau.txt')
217     OPEN(4,FILE='resultaat_freshfuel_radau.txt')
        DO I=1,mumps_par%N
219       WRITE(4,*) I, mumps_par%RHS(I)
        END DO
221     CLOSE(4)

223     call cpu_time(finish)
        write(*,*) "The elapsed time = ", finish-start
225
    C  Deallocate user data
227     IF ( mumps_par%MYID .eq. 0 )THEN
          DEALLOCATE( mumps_par%IRN )
229       DEALLOCATE( mumps_par%JCN )
          DEALLOCATE( mumps_par%A   )
231       DEALLOCATE( mumps_par%RHS )
        END IF
233
        DEALLOCATE(MATRIX)
235     DEALLOCATE(MATRIX2)
        DEALLOCATE(MATRIX3)
237     DEALLOCATE(LHS)
        DEALLOCATE(VECTOR)
239
    C  Destroy the instance (deallocate internal data structures)
241     mumps_par%JOB = -2
        CALL DMUMPS(mumps_par)
243     IF (mumps_par%INFOG(1).LT.0) THEN
          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
245     &                 "  mumps_par%INFOG(1)= ", mumps_par%INFOG(1),
        &                 "  mumps_par%INFOG(2)= ", mumps_par%INFOG(2)
247       GOTO 500
        END IF
249  500  CALL MPI_FINALIZE(IERR)
        STOP
251
        END
```

# D Codes which adjust the existing RADAU5 solver

## D.1 MA38

```
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER N,NE
      PARAMETER (N=3,NE=5)
C --  N IS THE DIMENSION OF THE SYSTEM AND NE IS THE NUMBER OF NONZERO ELEMENTS
      DIMENSION VALUE_JAC(NE)
      INTEGER IND_ROW, IND_COL, NUMB_OFF_DIAG
      INTEGER INDICES_JAC(2*NE)
      DIMENSION VALUE_E1(5)
      INTEGER INDICES_E1(10)
      INTEGER KEEP(20), INFO(40), ICNTL(20)
      DOUBLE PRECISION CNTL(10), RINFO(20)
      DOUBLE PRECISION X(N), W(4*N)
      DIMENSION B(3)
C     THIS IS AN ARBITRARY VECTOR FOR CONTROLLING THE IMPLEMENTATION

C --  JACOBIAN
      OPEN(2,FILE='test_matrix.txt')
      NUMB_OFF_DIAG=0
C --- TO COUNT HOW MANY ELEMENTS THERE ARE OFF THE DIAGONAL
      DO I=1,NE
            READ(2,*) IND_ROW, IND_COL, VALUE_JAC(I)
            INDICES_JAC(I)=IND_ROW
            INDICES_JAC(I+NE)=IND_COL
            IF (INDICES_JAC(I).NE.INDICES_JAC(NE+I)) THEN
               NUMB_OFF_DIAG=NUMB_OFF_DIAG+1
            END IF
      END DO
      CLOSE(2)

C --- COMPUTE THE MATRIX E1
      U1=(6.D0+81.D0**(1.D0/3.D0)-9.D0**(1.D0/3.D0))/30.D0
      U1=1.0D0/U1
      H=1.0D0
      FAC1=U1/H
      DO I=1,N
            INDICES_E1(I)=I
            INDICES_E1(N+NUMB_OFF_DIAG+I)=I
            VALUE_E1(I)=FAC1
      END DO
      J=1
      DO I=1,NE
            IF (INDICES_JAC(I).EQ.INDICES_JAC(NE+I)) THEN
               L=INDICES_JAC(I)
               VALUE_E1(L)=-VALUE_JAC(I)+FAC1
            ELSE
               INDICES_E1(N+J)=INDICES_JAC(I)
               INDICES_E1(N+NE+J)=INDICES_JAC(NE+I)
               VALUE_E1(N+J)=-VALUE_JAC(I)
               J=J+1
            END IF
      END DO
```

```
C ---- LU DECOMPOSITION E1
      DO I=1,3
                B(I)=1.0D0
      END DO

      LVALUE=300
      LINDEX=300

      CALL MA38ID(KEEP,CNTL,ICNTL)

      ICNTL(3)=5

      CALL MA38AD(N,N+NUMB_OFF_DIAG,0,.FALSE.,LVALUE,LINDEX,
     &                  VALUE_E1,INDICES_E1,KEEP,CNTL,ICNTL,INFO,RINFO)

      IF (INFO(1) .LT. 0) THEN
                WRITE(*,*) 'ERROR MESSAGE LUDECOMPOSITION E1', INFO(1)
      ELSE
                WRITE(*,*) 'LUDECOMPOSITION E1 SUCCEEDED'
      END IF


      CALL MA38CD(N,0,.FALSE.,LVALUE,LINDEX,
     &                  VALUE_E1,INDICES_E1,KEEP,B,X,W,
     &                  CNTL,ICNTL,INFO,RINFO)

      WRITE(*,*) 'Solution SYSTEM',
     &                              (X(I), I=1,N)
      IF (INFO(1) .LT. 0) THEN
                WRITE(*,*) 'ERROR MESSAGE SOLVING SYSTEM CONTAINED E1', INFO(1)
      ELSE
                WRITE(*,*) 'SOLVING SYSTEM CONTAINED E1 SUCCEEDED'
      END IF

      END
```

## D.2 ME38

```
      IMPLICIT REAL*8(A-H,O-Z)
      INTEGER N,NE
      PARAMETER (N=3,NE=5)
C --  N IS THE DIMENSION OF THE SYSTEM AND NE IS THE NUMBER OF NONZERO ELEMENTS
      DIMENSION VALUE_JAC(NE)
      INTEGER INDICES_JAC(2*NE)
      INTEGER IND_ROW, IND_COL, NUMB_OFF_DIAG
      DIMENSION INDICES_E2(10)
      COMPLEX*16 VALUE_E2(5),B(5)
      INTEGER KEEP(20), INFO(40), ICNTL(20)
      DOUBLE PRECISION CNTL(10), RINFO(20)
      COMPLEX*16 X(N), W(4*N)

C --  JACOBIAN
      OPEN(2,FILE='test_matrix.txt')
      NUMB_OFF_DIAG=0
```

```
      DO I=1,NE
18              READ(2,*) IND_ROW, IND_COL, VALUE_JAC(I)
                INDICES_JAC(I)=IND_ROW
20              INDICES_JAC(I+NE)=IND_COL
                IF (INDICES_JAC(I).NE.INDICES_JAC(NE+I)) THEN
22                  NUMB_OFF_DIAG=NUMB_OFF_DIAG+1
                END IF
24      END DO
        CLOSE(2)
26
C ---- COMPUTE THE MATRIX E2
28      ALPH=(12.D0-81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))/60.D0
        BETA=(81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))*DSQRT(3.D0)/60.D0
30      CNO=ALPH**2+BETA**2
        ALPH=ALPH/CNO
32      BETA=BETA/CNO
        H=1.0D-6
34      ALPHN=ALPH/H
        BETAN=BETA/H
36      DO I=1,N
                INDICES_E2(I)=I
38              INDICES_E2(N+NUMB_OFF_DIAG+I)=I
                VALUE_E2(I)=DCMPLX(ALPHN,BETAN)
40      END DO
        J=1
42      DO I=1,NE
                IF (INDICES_JAC(I).EQ.INDICES_JAC(NE+I)) THEN
44                  L=INDICES_JAC(I)
                    VALUE_E2(L)=-VALUE_JAC(I)+DCMPLX(ALPHN,BETAN)
46              ELSE
                    INDICES_E2(N+J)=INDICES_JAC(I)
48                  INDICES_E2(N+NE+J)=INDICES_JAC(NE+I)
                    VALUE_E2(N+J)=-VALUE_JAC(I)
50                  J=J+1
                END IF
52      END DO

54  C -- LU DECOMPOSITION E2
        DO I=1,3
56              B(I)=DCMPLX(1.0D0,0.0D0)
        END DO
58
        LVALUE=300
60      LINDEX=300

62      CALL ME38ID(KEEP,CNTL,ICNTL)

64      CALL ME38AD(N,N+NUMB_OFF_DIAG,0,.FALSE.,LVALUE,LINDEX,
       &                VALUE_E2,INDICES_E2,KEEP,CNTL,ICNTL,INFO,RINFO)
66
        IF (INFO(1) .LT. 0) THEN
68              WRITE(*,*) 'ERROR MESSAGE LUDECOMPOSITION E2', INFO(1)
        ELSE
70              WRITE(*,*) 'LUDECOMPOSITION E2 SUCCEEDED'
        END IF
```

```
72        CALL ME38CD(N,0,.FALSE.,LVALUE,LINDEX,
     &                  VALUE_E2,INDICES_E2,KEEP,B,X,W,
74   &                  CNTL,ICNTL,INFO,RINFO)

76        WRITE(*,*) 'Solution SYSTEM',
     &                                (X(I), I=1,N)
78
          IF (INFO(1) .LT. 0) THEN
80              WRITE(*,*) 'ERROR MESSAGE SOLVING SYSTEM CONTAINED E2', INFO(1)
          ELSE
82              WRITE(*,*) 'SOLVING SYSTEM CONTAINED E2 SUCCEEDED', INFO(1)
          END IF
84
          END
```

## D.3 MUMPS

### D.3.1 A fixed stepsize implementation of the original RADAU5 solver

```
1         IMPLICIT REAL*8(A-H,O-Z)
C --- N IS THE DIMENSION OF THE SYSTEM
3  C --- NZM IS THE NUMBER OF NONZERO ELEMENTS IN THE BATEMAN MATRIX
C --- NZV IS THE NUMBER OF NONZERO ELEMENTS IN THE INITIAL CONCENTRATION VECTOR
5         PARAMETER (N=3701,NZM=42464,NZV=5)
          DIMENSION IND_ROW_JAC(NZM), IND_COL_JAC(NZM), VALUE_JAC(NZM)
7         INTEGER, ALLOCATABLE, DIMENSION(:):: IND_ROW_E1, IND_COL_E1
          REAL*8, ALLOCATABLE, DIMENSION(:):: VALUE_E1
9         INTEGER, ALLOCATABLE, DIMENSION(:):: IND_ROW_E2, IND_COL_E2
          DOUBLE COMPLEX, ALLOCATABLE, DIMENSION(:):: VALUE_E2
11        DIMENSION IND_DIAGN(N)
          INTEGER, ALLOCATABLE, DIMENSION(:):: IND
13        DIMENSION Y(N)
          DIMENSION Z1(N),Z2(N),Z3(N),F1(N),F2(N),F3(N)
15        DIMENSION CONT(N)
          DIMENSION F(N)
17        DIMENSION R21(N),R22(N)

19 C --- INITIALIZE MUMPS
          INCLUDE 'mpif.h'
21        INCLUDE 'dmumps_struc.h'
          INCLUDE 'zmumps_struc.h'
23        TYPE (DMUMPS_STRUC) mumps_parE1
          TYPE (ZMUMPS_STRUC) mumps_parE2
25        CALL MPI_INIT(IERR)
C --- Define a communicator for the package.
27        mumps_parE1%COMM = MPI_COMM_WORLD
          mumps_parE2%COMM = MPI_COMM_WORLD
29 C --- Initialize an instance of the package
C --- for LU factorization (sym = 0, with working host)
31        mumps_parE1%JOB = -1
          mumps_parE1%SYM = 0
33        mumps_parE1%PAR = 1
          mumps_parE2%JOB = -1
35        mumps_parE2%SYM = 0
          mumps_parE2%PAR = 1
37        CALL DMUMPS(mumps_parE1)
```

```
      IF (mumps_parE1%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
        &                 "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
        GOTO 500
      END IF
      CALL ZMUMPS(mumps_parE2)
      IF (mumps_parE2%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
        &                 "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
        GOTO 500
      END IF

      mumps_parE1%ICNTL(4)=-1
      mumps_parE2%ICNTL(4)=-1

      call cpu_time(start)

C --- Define problem on the host (processor 0)

C NSING: TO COUNT HOW OFTEN THE MATRICES E1 AND E2 ARE SINGULAR
      NSING=0

      CALL JAC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC)

C --- CONTROL OF THE DIAGONAL. IF THE JACOBIAN HASN'T A VALUE ON
C --- THE DIAGIONAL, E1 HAS TO BE FAC1 AND E2 HAS TO BE ALPHN+iBETAN
C IND_DIAGN WILL CONTAIN THE DIAGONAL POSITIONS WHICH ARE FILLED
C IF THERE ARE ZEROS ON THE DIAGONAL, IND_DIAGN WILL CONTAIN
C A NUMBER GREATER THAN N
      DO I=1,N
        IND_DIAGN(I)=N+10
      END DO
C ND: NUMBER OF ELEMENTS ON THE DIAGONAL
      ND=0
C NNE: NUMBER OF NO ELEMENTS ON THE DIAGONAL
      NNE=1
      DO I=1,NZM
        IF (IND_ROW_JAC(I).EQ.IND_COL_JAC(I)) THEN
          ND=ND+1
          IND_DIAGN(ND)=IND_ROW_JAC(I)
        END IF
      END DO
C IND: A VECTOR CONTAINING THE INDICES OF THE DIAGONAL POSITIONS
C       WITH A ZERO NUMBER
      ALLOCATE(IND(N-ND))
C LESS ELEMENTS ON THE DIAGONAL (ND-NUMBER OF ELEMENTS ON DIAGONAL)
C THAN THE SIZE OF THE PROBLEM (N)
      IF (ND.NE.N) THEN
C CHECK EVERY DIAGONAL POSITION
        DO K=1,N
          NC=0
          DO L=1,ND
C CHECK IF THERE CORRESPONDS AN INDICE OF THE JACOBIAN WITH THE
```

```
C POSITION ON THE DIAGONAL
C AND IF THERE IS NO CORRESPONDENCE, IT WILL BE REMEMBERED
                IF (IND_DIAGN(L).NE.K) THEN
                  NC=NC+1
                  IF (NC.GE.ND) THEN
                    IND(NNE)=K
                    NNE=NNE+1
                  END IF
                END IF
              END DO
            END DO
          END IF

C NZME: THE NUMBER OF NONZERO ELEMENTS IN THE MATRICES E1 AND E2

          NZME=NZM+NNE-1

          ALLOCATE(IND_ROW_E1(NZME))
          ALLOCATE(IND_COL_E1(NZME))
          ALLOCATE(VALUE_E1(NZME))

          ALLOCATE(IND_ROW_E2(NZME))
          ALLOCATE(IND_COL_E2(NZME))
          ALLOCATE(VALUE_E2(NZME))

C --- STEPSIZE: H, END TIME: TEND, NUMBER OF STEPS: NUMB_OF_STEPS
          H=1800.0D0
          TEND=518400.0D0

       10 CONTINUE

          DO WHILE (MOD(TEND,H).NE.0)
            H=H+1
          END DO

          NUMB_OF_STEPS=TEND/H

C --- COMPUTE THE MATRIX E1
          U1=(6.D0+81.D0**(1.D0/3.D0)-9.D0**(1.D0/3.D0))/30.D0
          U1=1.0D0/U1
C --- FAC1 STANDS FOR GAMMA (NOTATION ORIGINAL RADAU5 SOLVER)
          FAC1=U1/H

C         SUBROUTINE DECOMR(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC,IND_ROW_E1,
C        &                  IND_ROW_E1,VALUE_E1)
C --- MAKE MATRIX E1
          IND_ROW_E1=IND_ROW_JAC
          IND_COL_E1=IND_COL_JAC
          DO J=1,NZM
            IF (IND_ROW_JAC(J).EQ.IND_COL_JAC(J)) THEN
              VALUE_E1(J)=FAC1-VALUE_JAC(J)
            ELSE
              VALUE_E1(J)=-VALUE_JAC(J)
            END IF
          END DO
```

```
149         DO J=1,NNE-1
              IND_ROW_E1(NZM+J)=IND(J)
151           IND_COL_E1(NZM+J)=IND(J)
              VALUE_E1(NZM+J)=FAC1
153         END DO

155   C --- COMPUTE THE MATRIX E2
            ALPH=(12.D0-81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))/60.D0
157         BETA=(81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))*DSQRT(3.D0)/60.D0
            CNO=ALPH**2+BETA**2
159         ALPH=ALPH/CNO
            BETA=BETA/CNO
161         ALPHN=ALPH/H
            BETAN=BETA/H

163
      C     SUBROUTINE DECOMC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC,IND_ROW_E2,
165   C    &                  IND_ROW_E2,VALUE_E2)
      C --- MAKE MATRIX E2
167         IND_ROW_E2=IND_ROW_JAC
            IND_COL_E2=IND_COL_JAC
169         DO J=1,NZM
              IF (IND_ROW_JAC(J).EQ.IND_COL_JAC(J)) THEN
171             VALUE_E2(J)=DCMPLX(ALPHN,BETAN)-VALUE_JAC(J)
              ELSE
173             VALUE_E2(J)=-VALUE_JAC(J)
              END IF
175         END DO
            DO J=1,NNE-1
177           IND_ROW_E2(NZM+J)=IND(J)
              IND_COL_E2(NZM+J)=IND(J)
179           VALUE_E2(NZM+J)=DCMPLX(ALPHN,BETAN)
            END DO

181
      C --- TRANSFORM TO NAMES MUMPS
183         mumps_parE1%NZ=NZME
            mumps_parE1%N=N
185         mumps_parE2%NZ=NZME
            mumps_parE2%N=N

187
            ALLOCATE(mumps_parE1%IRN(mumps_parE1%NZ))
189         ALLOCATE(mumps_parE1%JCN(mumps_parE1%NZ))
            ALLOCATE(mumps_parE1%A(mumps_parE1%NZ))
191         ALLOCATE(mumps_parE1%RHS(mumps_parE1%N))
            ALLOCATE(mumps_parE2%IRN(mumps_parE2%NZ))
193         ALLOCATE(mumps_parE2%JCN(mumps_parE2%NZ))
            ALLOCATE(mumps_parE2%A(mumps_parE2%NZ))
195         ALLOCATE(mumps_parE2%RHS(mumps_parE2%N))

197         mumps_parE1%IRN=IND_ROW_E1
            mumps_parE1%JCN=IND_COL_E1
199         mumps_parE1%A=VALUE_E1
            mumps_parE2%IRN=IND_ROW_E2
201         mumps_parE2%JCN=IND_COL_E2
            mumps_parE2%A=VALUE_E2
```

```
203  C ---- Call package for the analysis (JOB=1) an the factorisation (JOB=2)

205        mumps_parE1%JOB = 1
           CALL DMUMPS(mumps_parE1)
207        IF (mumps_parE1%INFOG(1).EQ.-6) THEN
             NSING=NSING+1
209          IF (NSING.GE.5) THEN
               WRITE(*,*) 'MATRIX IS REPEATEDLY SINGULAR'
211            STOP
             END IF
213          H=H*0.5D0
             GOTO 10
215        END IF
           IF (mumps_parE1%INFOG(1).LT.0) THEN
217          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
           &              "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
219        &              "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
             GOTO 500
221        END IF

223        mumps_parE1%JOB = 2
           CALL DMUMPS(mumps_parE1)
225        IF (mumps_parE1%INFOG(1).EQ.-10) THEN
             NSING=NSING+1
227          IF (NSING.GE.5) THEN
               WRITE(*,*) 'MATRIX IS REPEATEDLY SINGULAR'
229            STOP
             END IF
231          H=H*0.5D0
             GOTO 10
233        END IF
           IF (mumps_parE1%INFOG(1).LT.0) THEN
235          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
           &              "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
237        &              "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
             GOTO 500
239        END IF

241        mumps_parE2%JOB = 1
           CALL ZMUMPS(mumps_parE2)
243        IF (mumps_parE1%INFOG(1).EQ.-6) THEN
             NSING=NSING+1
245          IF (NSING.GE.5) THEN
               WRITE(*,*) 'MATRIX IS REPEATEDLY SINGULAR'
247            STOP
             END IF
249          H=H*0.5D0
             GOTO 10
251        END IF
           IF (mumps_parE2%INFOG(1).LT.0) THEN
253          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
           &              "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
255        &              "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
             GOTO 500
257        END IF
```

```
259         mumps_parE2%JOB = 2
            CALL ZMUMPS(mumps_parE2)
261         IF (mumps_parE1%INFOG(1).EQ.-10) THEN
              NSING=NSING+1
263           IF (NSING.GE.5) THEN
                WRITE(*,*) 'MATRIX IS REPEATEDLY SINGULAR'
265             STOP
              END IF
267           H=H*0.5D0
              GOTO 10
269         END IF
            IF (mumps_parE2%INFOG(1).LT.0) THEN
271           WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
            &                 "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
273         &                 "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
              GOTO 500
275         END IF

277   C --- THE INTIAL CONCENTRATION VECTOR
            DO I=1,N
279           Y(I)=0.0D0
            END DO
281         OPEN(15,FILE='BeginVectorFreshfuel.txt')
            DO K=1,NZV
283           READ(15,*) IND_VEC, E
              Y(IND_VEC)=E
285         END DO

287   C --- DEFINE THE MATRIX T AND T_INVERSE
            T11=9.123239487089294 2792D-02
289         T12=-0.1412552950209542 0843D0
            T13=-3.0029194105147424492D-02
291         T21=0.2417179327071070 1896D0
            T22=0.2041293522937999 3199D0
293         T23=0.3829421127572619 3779D0
            T31=0.9660481826150929 3619D0
295         TI11=4.3255798900631553510D0
            TI12=0.3391992518158098 6954D0
297         TI13=0.5417705399358748 7119D0
            TI21=-4.1787185915519047273D0
299         TI22=-0.3276828207610623 8708D0
            TI23=0.4766235545005504 5196D0
301         TI31=-0.5028726349457868 7595D0
            TI32=2.5719269498556054292D0
303         TI33=-0.5960392048282249 2497D0

305         DO M=1,NUMB_OF_STEPS

307   C --- THE STARTING VALUES OF THE NEWTON ITERATION
              DO I=1,N
309             Z1(I)=0.D0
                Z2(I)=0.D0
311             Z3(I)=0.D0
                F1(I)=0.D0
```

```
313             F2(I)=0.D0
                F3(I)=0.D0
315           END DO

317   C ---- COMPUTE THE RIGHT-HAND SIDE
              DO I=1,N
319             CONT(I)=Y(I)+Z1(I)
              END DO
321           CALL FUNCTION(N,NZM,IND_ROW_JAC,IND_COL_JAC,VALUE_JAC,CONT,F)
              Z1=F
323           DO I=1,N
                CONT(I)=Y(I)+Z2(I)
325           END DO
              CALL FUNCTION(N,NZM,IND_ROW_JAC,IND_COL_JAC,VALUE_JAC,CONT,F)
327           Z2=F
              DO I=1,N
329             CONT(I)=Y(I)+Z3(I)
              END DO
331           CALL FUNCTION(N,NZM,IND_ROW_JAC,IND_COL_JAC,VALUE_JAC,CONT,F)
              Z3=F
333
              DO I=1,N
335             A1=Z1(I)
                A2=Z2(I)
337             A3=Z3(I)
                Z1(I)=TI11*A1+TI12*A2+TI13*A3
339             Z2(I)=TI21*A1+TI22*A2+TI23*A3
                Z3(I)=TI31*A1+TI32*A2+TI33*A3
341           END DO

343           DO I=1,N
                S2=-F2(I)
345             S3=-F3(I)
                R21(I)=Z1(I)
347             R22(I)=Z2(I)
                Z1(I)=Z1(I)-F1(I)*FAC1
349             Z2(I)=Z2(I)+S2*ALPHN-S3*BETAN
                CONT(I)=Z3(I)+S3*ALPHN+S2*BETAN
351           END DO

353   C ---- TRANSFORM NAME RHS TO NAME MUMPS
              DO I=1,N
355             mumps_parE1%RHS(I)=Z1(I)
              END DO
357

359   C ---- Call package for solution
              mumps_parE1%JOB = 3
361           CALL DMUMPS(mumps_parE1)
              IF (mumps_parE1%INFOG(1).LT.0) THEN
363             WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
        &                 "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
365     &                 "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
                GOTO 500
367           END IF
```

```
            DO I=1,N
369           Z1(I)=mumps_parE1%RHS(I)
            END DO
371
      C --- TRANSFORM NAME RHS TO NAME MUMPS
373         DO I=1,N
              mumps_parE2%RHS(I)=DCMPLX(Z2(I),CONT(I))
375         END DO

377   C --- Call package for solution
            mumps_parE2%JOB = 3
379         CALL ZMUMPS(mumps_parE2)
            IF (mumps_parE2%INFOG(1).LT.0) THEN
381           WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
          &                 "   mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
383       &                 "   mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
              GOTO 500
385         END IF

387         DO I=1,N
              Z2(I)=REAL(mumps_parE2%RHS(I))
389           Z3(I)=REAL(AIMAG(mumps_parE2%RHS(I)))
            END DO
391
      C --- TRANSFORM TO THE NEW Z
393         DO I=1,N
              F1I=F1(I)+Z1(I)
395           F2I=F2(I)+Z2(I)
              F3I=F3(I)+Z3(I)
397           F1(I)=F1I
              F2(I)=F2I
399           F3(I)=F3I
              Z1(I)=T11*F1I+T12*F2I+T13*F3I
401           Z2(I)=T21*F1I+T22*F2I+T23*F3I
              Z3(I)=T31*F1I+      F2I
403         END DO

405   C --- TRANSFORM TO NEW Y
            DO I=1,N
407           Y(I)=Y(I)+Z3(I)
            END DO
409
          END DO
411

413   C --- PRINT FINAL SOLUTION
          OPEN (5,file='radau_freshfuel.txt')
415       DO K=1,3701
            WRITE(5,*) K, Y(K)
417       END DO
          CLOSE(5)
419
          call cpu_time(finish)
421       WRITE(*,*) "The elapsed time", finish-start
```

```
C ---- Deallocate user data
      IF ( mumps_parE1%MYID .eq. 0 )THEN
        DEALLOCATE( mumps_parE1%IRN )
        DEALLOCATE( mumps_parE1%JCN )
        DEALLOCATE( mumps_parE1%A )
        DEALLOCATE( mumps_parE1%RHS )
      END IF

      IF ( mumps_parE2%MYID .eq. 0 )THEN
        DEALLOCATE( mumps_parE2%IRN )
        DEALLOCATE( mumps_parE2%JCN )
        DEALLOCATE( mumps_parE2%A )
        DEALLOCATE( mumps_parE2%RHS )
      END IF

      DEALLOCATE(IND_ROW_E1)
      DEALLOCATE(IND_COL_E1)
      DEALLOCATE(VALUE_E1)

      DEALLOCATE(IND_ROW_E2)
      DEALLOCATE(IND_COL_E2)
      DEALLOCATE(VALUE_E2)

      DEALLOCATE(IND)

C ---- Destroy the instance (deallocate internal data structures)
      mumps_parE1%JOB = -2
      CALL DMUMPS(mumps_parE1)
      IF (mumps_parE1%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "   mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
     &                 "   mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
        GOTO 500
      END IF

      mumps_parE2%JOB = -2
      CALL ZMUMPS(mumps_parE2)
      IF (mumps_parE2%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "   mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
     &                 "   mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
        GOTO 500
      END IF

  500 CALL MPI_FINALIZE(IERR)

      STOP

      END

      SUBROUTINE FUNCTION(N,NZE,IND_ROW,IND_COL,VALUE,VECTOR,F)
C ---- FUNCTION EVALUATION
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION F(N),VECTOR(N)
      DIMENSION IND_ROW(NZE),IND_COL(NZE),VALUE(NZE)
```

```
      DO I=1,N
479     F(I)=0.0D0
      END DO
481   DO J=1,NZE
        F(IND_ROW(J))=F(IND_ROW(J))
483   &                          +VALUE(J)*VECTOR(IND_COL(J))
      END DO
485   RETURN
      END SUBROUTINE FUNCTION
487
      SUBROUTINE JAC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC)
489 C --- JACOBIAN
      IMPLICIT REAL*8(A-H,O-Z)
491   DIMENSION IND_ROW_JAC(NZM),IND_COL_JAC(NZM),VALUE_JAC(NZM)
      OPEN(2,FILE='matrixfreshfuel.txt')
493   DO I=1,NZM
        READ(2,*) IND_ROW_JAC(I), IND_COL_JAC(I), VALUE_JAC(I)
495   END DO
      CLOSE(2)
497   RETURN
      END SUBROUTINE JAC
```

### D.3.2 A modification of the existing RADAU5 solver

#### D.3.2.1 The driver of the sparse RADAU5 solver

```
C * * * * * * * * * * * * * * * * * * * * * * * * * * *
2 C --- DRIVER FOR RADAU5 AT BATEMAN EQUATIONS
C * * * * * * * * * * * * * * * * * * * * * * * * * * *
4 c link dr_radau radau lapack lapackc dc_lapack
        IMPLICIT REAL*8 (A-H,O-Z)
6 C --- PARAMETERS FOR RADAU (FULL JACOBIAN)
        PARAMETER (ND=3701,NS=7,LWORK=(NS+1)*ND*ND+(3*NS+3)*ND+20,
8       &          LIWORK=(2+(NS-1)/2)*ND+20)
        DIMENSION Y(ND),WORK(LWORK),IWORK(LIWORK), ID_VECTOR(ND)
10      EXTERNAL FUNCT,JAC
C --- PARAMETER IN THE DIFFERENTIAL EQUATION, NO MEANING IN THIS EXAMPLE
12      RPAR=1.0D-6
C --- DIMENSION OF THE SYSTEM
14      N=3701
C --- THE NUMBER OF NONZERO ELEMENTS IN THE BATEMAN MATRIX
16      OPEN(4,FILE='matrix1_zw')
        NZM=0
18      DO WHILE (.true.)
          READ(4,*,end=999)
20        NZM=NZM+1
        END DO
22  999 CONTINUE
        CLOSE(4)
24 C --- COMPUTE THE JACOBIAN ANALYTICALLY
        IJAC=1
26 C --- JACOBIAN IS A FULL MATRIX
        MLJAC=N
28 C --- DIFFERENTIAL EQUATION IS IN EXPLICIT FORM
        IMAS=0
```

```
C --- OUTPUT ROUTINE IS NOT USED DURING INTEGRATION
      IOUT=0
C --- INITIAL VALUES
      X=0.0D0
      DO I=1,N
        Y(I)=0.0D0
      END DO
      OPEN (3,file='BeginVectorFreshfuel.txt')
      DO J=1,5
        READ(3,*) IND,E
        Y(IND)=E
      END DO
      CLOSE(3)
C --- ENDPOINT OF INTEGRATION
      XEND=518400.0D0
C --- REQUIRED TOLERANCE
      RTOL=1.0D-4
      ATOL=1.0D0*RTOL
      ITOL=0
C --- INITIAL STEP SIZE
      H=1.0D-6
C --- SET DEFAULT VALUES
      DO I=1,20
        IWORK(I)=0
        WORK(I)=0.D0
      END DO
      WORK(5)=0.99D0
      WORK(6)=2.0D0
C --- MEASURE THE TIME
       call cpu_time(start)
C --- CALL OF THE SUBROUTINE RADAU
      CALL RADAU5(N,NZM,FUNCT,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JAC,IJAC,MLJAC,MUJAC,
     &                  FUNCT,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
C --- MEASURE THE TIME
       call cpu_time(finish)
       write(*,*) "The elapsed time = ", finish-start
C --- PRINT FINAL SOLUTION
      OPEN (5,file='radau_sparse_freshfuel.txt')
      DO K=1,N
        WRITE(5,900) K, Y(K)
      END DO
 900  FORMAT(I5.5x,E19.8E3)
      CLOSE(5)
C --- PRINT STATISTICS
      WRITE (6,90) RTOL
      90    FORMAT('     rtol=',D8.2)
      WRITE (6,91) (IWORK(J),J=14,20)
      91    FORMAT(' fcn=',I5,' jac=',I4,' step=',I4,' accpt=',I4,
     &       ' rejct=',I3,' dec=',I4,' sol=',I5)
      STOP
      END
```

```
      SUBROUTINE FUNCT(N,X,Y,F,RPAR,IPAR)
C ---- THE RIGHT-HAND SIDE OF BATEMAN EQUATIONS
      IMPLICIT REAL*8 (A-H,O-Z)
      DIMENSION Y(N),F(N)
      INTEGER M(42464),IND_COL(42464),IND_ROW(42464)
      DIMENSION VALUE(42464)

      N_IND_ROW=2
C ---- NUMBER TO COUNT HOW MANY NUMBERS THERE ARE IN IND_ROW
      N_LIST=0
C ---- NUMBER TO COUNT HOW MANY NUMBERS THERE ARE IN THE LIST
      OPEN(2,file='matrixfreshfuel.txt')
      IND_ROW(1)=1
      DO K=1,42464
        READ(2,*) M(K),L,r
        VALUE(K)=r
        IND_COL(K)=L
        N_LIST=N_LIST+1
        IF (K.NE.1.AND.M(K).NE.M(K-1)) THEN
          IND_ROW(N_IND_ROW)=N_LIST
          N_IND_ROW=N_IND_ROW+1
        END IF
      END DO
      IND_ROW(N_IND_ROW)=N_LIST+1
      CLOSE(2)
      DO I=1,N
        F(I)=0.0D0
        DO J=IND_ROW(I),IND_ROW(I+1)-1
          F(I)=F(I)+VALUE(J)*Y(IND_COL(J))
        END DO
      END DO
      RETURN
      END

      SUBROUTINE JAC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC)
C ---- JACOBIAN
      IMPLICIT REAL*8(A-H,O-Z)
      DIMENSION IND_ROW_JAC(NZM),IND_COL_JAC(NZM),VALUE_JAC(NZM)
      OPEN(14,FILE='matrixfreshfuel.txt')
      DO I=1,NZM
        READ(14,*) IND_ROW_JAC(I), IND_COL_JAC(I), VALUE_JAC(I)
      END DO
      CLOSE(14)
      RETURN
      END SUBROUTINE JAC
```

#### D.3.2.2 The sparse RADAU5 solver

```
      SUBROUTINE RADAU5(N,NZM,FCN,X,Y,XEND,H,
     &                  RTOL,ATOL,ITOL,
     &                  JAC  ,IJAC,MLJAC,MUJAC,
     &                  MAS  ,IMAS,MLMAS,MUMAS,
     &                  SOLOUT,IOUT,
     &                  WORK,LWORK,IWORK,LIWORK,RPAR,IPAR,IDID)
C ---------------------------------------------------------
C A SPARSE VERSION OF THE RADAU5 SOLVER
```

```
C ORIGANALLY DEVELOPED BY E. HAIRER AND G. WANNER
C
C THESE SPARSE VERSION IS IMPLEMENTED IN ORDER TO MAKE IT MORE
C SUITABLE FOR SOLVING THE BATEMAN EQUATIONS AS A PART OF THE
C MASTER THESIS OF MAREN VRANCKX, GHENT UNIVERSITY AND IN
C COOPERATION WITH SCK–CEN
C -----------------------------------------------------------
C
C ORIGINAL RADAU5 DESCRIPTION
C
C      NUMERICAL SOLUTION OF A STIFF (OR DIFFERENTIAL ALGEBRAIC)
C      SYSTEM OF FIRST 0RDER ORDINARY DIFFERENTIAL EQUATIONS
C                        M*Y'=F(X,Y).
C      THE SYSTEM CAN BE (LINEARLY) IMPLICIT (MASS–MATRIX M .NE. I)
C      OR EXPLICIT (M=I).
C      THE METHOD USED IS AN IMPLICIT RUNGE–KUTTA METHOD (RADAU IIA)
C      OF ORDER 5 WITH STEP SIZE CONTROL AND CONTINUOUS OUTPUT.
C      CF. SECTION IV.8
C
C      AUTHORS: E. HAIRER AND G. WANNER
C               UNIVERSITE DE GENEVE, DEPT. DE MATHEMATIQUES
C               CH–1211 GENEVE 24, SWITZERLAND
C               E–MAIL:  Ernst.Hairer@math.unige.ch
C                        Gerhard.Wanner@math.unige.ch
C
C      THIS CODE IS PART OF THE BOOK:
C          E. HAIRER AND G. WANNER, SOLVING ORDINARY DIFFERENTIAL
C          EQUATIONS II. STIFF AND DIFFERENTIAL–ALGEBRAIC PROBLEMS.
C          SPRINGER SERIES IN COMPUTATIONAL MATHEMATICS 14,
C          SPRINGER–VERLAG 1991, SECOND EDITION 1996.
C
C      VERSION OF JULY 9, 1996
C      (latest small correction: January 18, 2002)
C
C      INPUT PARAMETERS
C      ----------------
C      N            DIMENSION OF THE SYSTEM
C
C      FCN          NAME (EXTERNAL) OF SUBROUTINE COMPUTING THE
C                   VALUE OF F(X,Y):
C                       SUBROUTINE FCN(N,X,Y,F,RPAR,IPAR)
C                       DOUBLE PRECISION X,Y(N),F(N)
C                       F(1)=...   ETC.
C                   RPAR, IPAR (SEE BELOW)
C
C      X            INITIAL X–VALUE
C
C      Y(N)         INITIAL VALUES FOR Y
C
C      XEND         FINAL X–VALUE (XEND–X MAY BE POSITIVE OR NEGATIVE)
C
C      H            INITIAL STEP SIZE GUESS;
C                   FOR STIFF EQUATIONS WITH INITIAL TRANSIENT,
C                   H=1.D0/(NORM OF F'), USUALLY 1.D–3 OR 1.D–5, IS GOOD.
C                   THIS CHOICE IS NOT VERY IMPORTANT, THE STEP SIZE IS
```

```
C                   QUICKLY ADAPTED.  (IF H=0.D0,  THE CODE PUTS H=1.D-6).
C
C     RTOL,ATOL     RELATIVE AND ABSOLUTE ERROR TOLERANCES.  THEY
C                   CAN BE BOTH SCALARS OR ELSE BOTH VECTORS OF LENGTH N.
C
C     ITOL          SWITCH FOR RTOL AND ATOL:
C                     ITOL=0: BOTH RTOL AND ATOL ARE SCALARS.
C                       THE CODE KEEPS,  ROUGHLY,  THE LOCAL ERROR OF
C                       Y(I) BELOW RTOL*ABS(Y(I))+ATOL
C                     ITOL=1: BOTH RTOL AND ATOL ARE VECTORS.
C                       THE CODE KEEPS THE LOCAL ERROR OF Y(I) BELOW
C                       RTOL(I)*ABS(Y(I))+ATOL(I).
C
C     JAC           NAME (EXTERNAL) OF THE SUBROUTINE WHICH COMPUTES
C                   THE PARTIAL DERIVATIVES OF F(X,Y) WITH RESPECT TO Y
C                   (THIS ROUTINE IS ONLY CALLED IF IJAC=1; SUPPLY
C                   A DUMMY SUBROUTINE IN THE CASE IJAC=0).
C                   FOR IJAC=1,  THIS SUBROUTINE MUST HAVE THE FORM
C                       SUBROUTINE JAC(N,X,Y,DFY,LDFY,RPAR,IPAR)
C                       DOUBLE PRECISION X,Y(N),DFY(LDFY,N)
C                       DFY(1,1)= ...
C                   LDFY,  THE COLUMN-LENGTH OF THE ARRAY,  IS
C                   FURNISHED BY THE CALLING PROGRAM.
C                   IF (MLJAC.EQ.N) THE JACOBIAN IS SUPPOSED TO
C                       BE FULL AND THE PARTIAL DERIVATIVES ARE
C                       STORED IN DFY AS
C                           DFY(I,J) = PARTIAL F(I) / PARTIAL Y(J)
C                   ELSE,  THE JACOBIAN IS TAKEN AS BANDED AND
C                       THE PARTIAL DERIVATIVES ARE STORED
C                       DIAGONAL-WISE AS
C                           DFY(I-J+MUJAC+1,J) = PARTIAL F(I) / PARTIAL Y(J).
C
C     IJAC          SWITCH FOR THE COMPUTATION OF THE JACOBIAN:
C                       IJAC=0: JACOBIAN IS COMPUTED INTERNALLY BY FINITE
C                           DIFFERENCES,  SUBROUTINE "JAC" IS NEVER CALLED.
C                       IJAC=1: JACOBIAN IS SUPPLIED BY SUBROUTINE JAC.
C
C     MLJAC         SWITCH FOR THE BANDED STRUCTURE OF THE JACOBIAN:
C                       MLJAC=N: JACOBIAN IS A FULL MATRIX.  THE LINEAR
C                           ALGEBRA IS DONE BY FULL-MATRIX GAUSS-ELIMINATION.
C                       0<=MLJAC<N: MLJAC IS THE LOWER BANDWITH OF JACOBIAN
C                           MATRIX (>= NUMBER OF NONZERO DIAGONALS BELOW
C                           THE MAIN DIAGONAL).
C
C     MUJAC         UPPER BANDWITH OF JACOBIAN  MATRIX (>= NUMBER OF NON-
C                   ZERO DIAGONALS ABOVE THE MAIN DIAGONAL).
C                   NEED NOT BE DEFINED IF MLJAC=N.
C
C     ------    MAS,IMAS,MLMAS,  AND MUMAS HAVE ANALOG MEANINGS       ------
C     ------    FOR THE "MASS MATRIX" (THE MATRIX "M" OF SECTION IV.8): -
C
C     MAS           NAME (EXTERNAL) OF SUBROUTINE COMPUTING THE MASS-
C                   MATRIX M.
C                   IF IMAS=0,  THIS MATRIX IS ASSUMED TO BE THE IDENTITY
C                   MATRIX AND NEEDS NOT TO BE DEFINED;
```

```
C                   SUPPLY A DUMMY SUBROUTINE IN THIS CASE.
C                   IF IMAS=1, THE SUBROUTINE MAS IS OF THE FORM
C                      SUBROUTINE MAS(N,AM,LMAS,RPAR,IPAR)
C                      DOUBLE PRECISION AM(LMAS,N)
C                      AM(1,1)= ....
C                      IF (MLMAS.EQ.N) THE MASS-MATRIX IS STORED
C                      AS FULL MATRIX LIKE
C                           AM(I,J) = M(I,J)
C                      ELSE, THE MATRIX IS TAKEN AS BANDED AND STORED
C                      DIAGONAL-WISE AS
C                           AM(I-J+MUMAS+1,J) = M(I,J).
C
C     IMAS          GIVES INFORMATION ON THE MASS-MATRIX:
C                      IMAS=0: M IS SUPPOSED TO BE THE IDENTITY
C                         MATRIX, MAS IS NEVER CALLED.
C                      IMAS=1: MASS-MATRIX  IS SUPPLIED.
C
C     MLMAS         SWITCH FOR THE BANDED STRUCTURE OF THE MASS-MATRIX:
C                      MLMAS=N: THE FULL MATRIX CASE. THE LINEAR
C                         ALGEBRA IS DONE BY FULL-MATRIX GAUSS-ELIMINATION.
C                      0<=MLMAS<N: MLMAS IS THE LOWER BANDWITH OF THE
C                         MATRIX (>= NUMBER OF NONZERO DIAGONALS BELOW
C                         THE MAIN DIAGONAL).
C                   MLMAS IS SUPPOSED TO BE .LE. MLJAC.
C
C     MUMAS         UPPER BANDWITH OF MASS-MATRIX (>= NUMBER OF NON-
C                   ZERO DIAGONALS ABOVE THE MAIN DIAGONAL).
C                   NEED NOT BE DEFINED IF MLMAS=N.
C                   MUMAS IS SUPPOSED TO BE .LE. MUJAC.
C
C     SOLOUT        NAME (EXTERNAL) OF SUBROUTINE PROVIDING THE
C                   NUMERICAL SOLUTION DURING INTEGRATION.
C                   IF IOUT=1, IT IS CALLED AFTER EVERY SUCCESSFUL STEP.
C                   SUPPLY A DUMMY SUBROUTINE IF IOUT=0.
C                   IT MUST HAVE THE FORM
C                      SUBROUTINE SOLOUT (NR,XOLD,X,Y,CONT,LRC,N,
C                                         RPAR,IPAR,IRTRN)
C                      DOUBLE PRECISION X,Y(N),CONT(LRC)
C                      ....
C                   SOLOUT FURNISHES THE SOLUTION "Y" AT THE NR-TH
C                      GRID-POINT "X" (THEREBY THE INITIAL VALUE IS
C                      THE FIRST GRID-POINT).
C                   "XOLD" IS THE PRECEEDING GRID-POINT.
C                   "IRTRN" SERVES TO INTERRUPT THE INTEGRATION. IF IRTRN
C                      IS SET <0, RADAU5 RETURNS TO THE CALLING PROGRAM.
C
C           ------  CONTINUOUS OUTPUT: ------
C                   DURING CALLS TO "SOLOUT", A CONTINUOUS SOLUTION
C                   FOR THE INTERVAL [XOLD,X] IS AVAILABLE THROUGH
C                   THE FUNCTION
C                            >>>    CONTR5(I,S,CONT,LRC)    <<<
C                   WHICH PROVIDES AN APPROXIMATION TO THE I-TH
C                   COMPONENT OF THE SOLUTION AT THE POINT S. THE VALUE
C                   S SHOULD LIE IN THE INTERVAL [XOLD,X].
C                   DO NOT CHANGE THE ENTRIES OF CONT(LRC), IF THE
```

```
C                   DENSE OUTPUT FUNCTION IS USED.
C
C     IOUT          SWITCH FOR CALLING THE SUBROUTINE SOLOUT:
C                       IOUT=0: SUBROUTINE IS NEVER CALLED
C                       IOUT=1: SUBROUTINE IS AVAILABLE FOR OUTPUT.
C
C     WORK          ARRAY OF WORKING SPACE OF LENGTH "LWORK".
C                   WORK(1), WORK(2),..., WORK(20) SERVE AS PARAMETERS
C                   FOR THE CODE. FOR STANDARD USE OF THE CODE
C                   WORK(1),...,WORK(20) MUST BE SET TO ZERO BEFORE
C                   CALLING. SEE BELOW FOR A MORE SOPHISTICATED USE.
C                   WORK(21),...,WORK(LWORK) SERVE AS WORKING SPACE
C                   FOR ALL VECTORS AND MATRICES.
C                   "LWORK" MUST BE AT LEAST
C                               N*(LJAC+LMAS+3*LE+12)+20
C                   WHERE
C                       LJAC=N                  IF MLJAC=N (FULL JACOBIAN)
C                       LJAC=MLJAC+MUJAC+1      IF MLJAC<N (BANDED JAC.)
C                   AND
C                       LMAS=0                  IF IMAS=0
C                       LMAS=N                  IF IMAS=1 AND MLMAS=N (FULL)
C                       LMAS=MLMAS+MUMAS+1      IF MLMAS<N (BANDED MASS-M.)
C                   AND
C                       LE=N                  IF MLJAC=N (FULL JACOBIAN)
C                       LE=2*MLJAC+MUJAC+1    IF MLJAC<N (BANDED JAC.)
C
C                   IN THE USUAL CASE WHERE THE JACOBIAN IS FULL AND THE
C                   MASS-MATRIX IS THE INDENTITY (IMAS=0), THE MINIMUM
C                   STORAGE REQUIREMENT IS
C                               LWORK = 4*N*N+12*N+20.
C                   IF IWORK(9)=M1>0 THEN "LWORK" MUST BE AT LEAST
C                            N*(LJAC+12)+(N-M1)*(LMAS+3*LE)+20
C                   WHERE IN THE DEFINITIONS OF LJAC, LMAS AND LE THE
C                   NUMBER N CAN BE REPLACED BY N-M1.
C
C     LWORK         DECLARED LENGTH OF ARRAY "WORK".
C
C     IWORK         INTEGER WORKING SPACE OF LENGTH "LIWORK".
C                   IWORK(1),IWORK(2),...,IWORK(20) SERVE AS PARAMETERS
C                   FOR THE CODE. FOR STANDARD USE, SET IWORK(1),...,
C                   IWORK(20) TO ZERO BEFORE CALLING.
C                   IWORK(21),...,IWORK(LIWORK) SERVE AS WORKING AREA.
C                   "LIWORK" MUST BE AT LEAST 3*N+20.
C
C     LIWORK        DECLARED LENGTH OF ARRAY "IWORK".
C
C     RPAR, IPAR    REAL AND INTEGER PARAMETERS (OR PARAMETER ARRAYS) WHICH
C                   CAN BE USED FOR COMMUNICATION BETWEEN YOUR CALLING
C                   PROGRAM AND THE FCN, JAC, MAS, SOLOUT SUBROUTINES.
C
C ---------------------------------------------------------------------
C
C     SOPHISTICATED SETTING OF PARAMETERS
C     ----------------------------------
C               SEVERAL PARAMETERS OF THE CODE ARE TUNED TO MAKE IT WORK
```

```
C               WELL. THEY MAY BE DEFINED BY SETTING WORK(1),...
C               AS WELL AS IWORK(1),... DIFFERENT FROM ZERO.
C               FOR ZERO INPUT, THE CODE CHOOSES DEFAULT VALUES:
C
C     IWORK(1)  IF IWORK(1).NE.0, THE CODE TRANSFORMS THE JACOBIAN
C               MATRIX TO HESSENBERG FORM. THIS IS PARTICULARLY
C               ADVANTAGEOUS FOR LARGE SYSTEMS WITH FULL JACOBIAN.
C               IT DOES NOT WORK FOR BANDED JACOBIAN (MLJAC<N)
C               AND NOT FOR IMPLICIT SYSTEMS (IMAS=1).
C
C     IWORK(2)  THIS IS THE MAXIMAL NUMBER OF ALLOWED STEPS.
C               THE DEFAULT VALUE (FOR IWORK(2)=0) IS 100000.
C
C     IWORK(3)  THE MAXIMUM NUMBER OF NEWTON ITERATIONS FOR THE
C               SOLUTION OF THE IMPLICIT SYSTEM IN EACH STEP.
C               THE DEFAULT VALUE (FOR IWORK(3)=0) IS 7.
C
C     IWORK(4)  IF IWORK(4).EQ.0 THE EXTRAPOLATED COLLOCATION SOLUTION
C               IS TAKEN AS STARTING VALUE FOR NEWTON'S METHOD.
C               IF IWORK(4).NE.0 ZERO STARTING VALUES ARE USED.
C               THE LATTER IS RECOMMENDED IF NEWTON'S METHOD HAS
C               DIFFICULTIES WITH CONVERGENCE (THIS IS THE CASE WHEN
C               NSTEP IS LARGER THAN NACCPT + NREJCT; SEE OUTPUT PARAM.).
C               DEFAULT IS IWORK(4)=0.
C
C        THE FOLLOWING 3 PARAMETERS ARE IMPORTANT FOR
C        DIFFERENTIAL-ALGEBRAIC SYSTEMS OF INDEX > 1.
C        THE FUNCTION-SUBROUTINE SHOULD BE WRITTEN SUCH THAT
C        THE INDEX 1,2,3 VARIABLES APPEAR IN THIS ORDER.
C        IN ESTIMATING THE ERROR THE INDEX 2 VARIABLES ARE
C        MULTIPLIED BY H, THE INDEX 3 VARIABLES BY H**2.
C
C     IWORK(5)  DIMENSION OF THE INDEX 1 VARIABLES (MUST BE > 0). FOR
C               ODE'S THIS EQUALS THE DIMENSION OF THE SYSTEM.
C               DEFAULT IWORK(5)=N.
C
C     IWORK(6)  DIMENSION OF THE INDEX 2 VARIABLES. DEFAULT IWORK(6)=0.
C
C     IWORK(7)  DIMENSION OF THE INDEX 3 VARIABLES. DEFAULT IWORK(7)=0.
C
C     IWORK(8)  SWITCH FOR STEP SIZE STRATEGY
C               IF IWORK(8).EQ.1  MOD. PREDICTIVE CONTROLLER (GUSTAFSSON)
C               IF IWORK(8).EQ.2  CLASSICAL STEP SIZE CONTROL
C               THE DEFAULT VALUE (FOR IWORK(8)=0) IS IWORK(8)=1.
C               THE CHOICE IWORK(8).EQ.1 SEEMS TO PRODUCE SAFER RESULTS;
C               FOR SIMPLE PROBLEMS, THE CHOICE IWORK(8).EQ.2 PRODUCES
C               OFTEN SLIGHTLY FASTER RUNS
C
C        IF THE DIFFERENTIAL SYSTEM HAS THE SPECIAL STRUCTURE THAT
C             Y(I)' = Y(I+M2)    FOR  I=1,...,M1,
C        WITH M1 A MULTIPLE OF M2, A SUBSTANTIAL GAIN IN COMPUTERTIME
C        CAN BE ACHIEVED BY SETTING THE PARAMETERS IWORK(9) AND IWORK(10).
C        E.G., FOR SECOND ORDER SYSTEMS P'=V, V'=G(P,V), WHERE P AND V ARE
C        VECTORS OF DIMENSION N/2, ONE HAS TO PUT M1=M2=N/2.
C        FOR M1>0 SOME OF THE INPUT PARAMETERS HAVE DIFFERENT MEANINGS:
```

```
C         - JAC: ONLY THE ELEMENTS OF THE NON-TRIVIAL PART OF THE
C                JACOBIAN HAVE TO BE STORED
C                IF (MLJAC.EQ.N-M1) THE JACOBIAN IS SUPPOSED TO BE FULL
C                   DFY(I,J) = PARTIAL F(I+M1) / PARTIAL Y(J)
C                FOR I=1,N-M1 AND J=1,N.
C                ELSE, THE JACOBIAN IS BANDED ( M1 = M2 * MM )
C                   DFY(I-J+MUJAC+1,J+K*M2) = PARTIAL F(I+M1) / PARTIAL Y(J+K*M2)
C                FOR I=1,MLJAC+MUJAC+1 AND J=1,M2 AND K=0,MM.
C         - MLJAC: MLJAC=N-M1: IF THE NON-TRIVIAL PART OF THE JACOBIAN IS FULL
C                0<=MLJAC<N-M1: IF THE (MM+1) SUBMATRICES (FOR K=0,MM)
C                     PARTIAL F(I+M1) / PARTIAL Y(J+K*M2), I,J=1,M2
C                   ARE BANDED, MLJAC IS THE MAXIMAL LOWER BANDWIDTH
C                   OF THESE MM+1 SUBMATRICES
C         - MUJAC: MAXIMAL UPPER BANDWIDTH OF THESE MM+1 SUBMATRICES
C                NEED NOT BE DEFINED IF MLJAC=N-M1
C         - MAS: IF IMAS=0 THIS MATRIX IS ASSUMED TO BE THE IDENTITY AND
C                NEED NOT BE DEFINED. SUPPLY A DUMMY SUBROUTINE IN THIS CASE.
C                IT IS ASSUMED THAT ONLY THE ELEMENTS OF RIGHT LOWER BLOCK OF
C                DIMENSION N-M1 DIFFER FROM THAT OF THE IDENTITY MATRIX.
C                IF (MLMAS.EQ.N-M1) THIS SUBMATRIX IS SUPPOSED TO BE FULL
C                   AM(I,J) = M(I+M1,J+M1)     FOR I=1,N-M1 AND J=1,N-M1.
C                ELSE, THE MASS MATRIX IS BANDED
C                   AM(I-J+MUMAS+1,J) = M(I+M1,J+M1)
C         - MLMAS: MLMAS=N-M1: IF THE NON-TRIVIAL PART OF M IS FULL
C                0<=MLMAS<N-M1: LOWER BANDWIDTH OF THE MASS MATRIX
C         - MUMAS: UPPER BANDWIDTH OF THE MASS MATRIX
C                NEED NOT BE DEFINED IF MLMAS=N-M1
C
C     IWORK(9)  THE VALUE OF M1.  DEFAULT M1=0.
C
C     IWORK(10) THE VALUE OF M2.  DEFAULT M2=M1.
C
C ----------
C
C     WORK(1)   UROUND, THE ROUNDING UNIT, DEFAULT 1.D-16.
C
C     WORK(2)   THE SAFETY FACTOR IN STEP SIZE PREDICTION,
C               DEFAULT 0.9D0.
C
C     WORK(3)   DECIDES WHETHER THE JACOBIAN SHOULD BE RECOMPUTED;
C               INCREASE WORK(3), TO 0.1 SAY, WHEN JACOBIAN EVALUATIONS
C               ARE COSTLY. FOR SMALL SYSTEMS WORK(3) SHOULD BE SMALLER
C               (0.001D0, SAY). NEGATIV WORK(3) FORCES THE CODE TO
C               COMPUTE THE JACOBIAN AFTER EVERY ACCEPTED STEP.
C               DEFAULT 0.001D0.
C
C     WORK(4)   STOPPING CRITERION FOR NEWTON'S METHOD, USUALLY CHOSEN <1.
C               SMALLER VALUES OF WORK(4) MAKE THE CODE SLOWER, BUT SAFER.
C               DEFAULT MIN(0.03D0,RTOL(1)**0.5D0)
C
C     WORK(5) AND WORK(6) : IF WORK(5) < HNEW/HOLD < WORK(6), THEN THE
C               STEP SIZE IS NOT CHANGED. THIS SAVES, TOGETHER WITH A
C               LARGE WORK(3), LU-DECOMPOSITIONS AND COMPUTING TIME FOR
C               LARGE SYSTEMS. FOR SMALL SYSTEMS ONE MAY HAVE
C               WORK(5)=1.D0, WORK(6)=1.2D0, FOR LARGE FULL SYSTEMS
```

```
339 C                WORK(5)=0.99D0, WORK(6)=2.D0 MIGHT BE GOOD.
    C                DEFAULTS WORK(5)=1.D0, WORK(6)=1.2D0 .
341 C
    C    WORK(7)     MAXIMAL STEP SIZE, DEFAULT XEND-X.
343 C
    C    WORK(8), WORK(9)   PARAMETERS FOR STEP SIZE SELECTION
345 C                THE NEW STEP SIZE IS CHOSEN SUBJECT TO THE RESTRICTION
    C                   WORK(8) <= HNEW/HOLD <= WORK(9)
347 C                DEFAULT VALUES: WORK(8)=0.2D0, WORK(9)=8.D0
    C
349 C-----------------------------------------------------------------------
    C
351 C      OUTPUT PARAMETERS
    C      -----------------
353 C      X              X-VALUE FOR WHICH THE SOLUTION HAS BEEN COMPUTED
    C                     (AFTER SUCCESSFUL RETURN X=XEND).
355 C
    C      Y(N)           NUMERICAL SOLUTION AT X
357 C
    C      H              PREDICTED STEP SIZE OF THE LAST ACCEPTED STEP
359 C
    C      IDID           REPORTS ON SUCCESSFULNESS UPON RETURN:
361 C                        IDID= 1  COMPUTATION SUCCESSFUL,
    C                        IDID= 2  COMPUT. SUCCESSFUL (INTERRUPTED BY SOLOUT)
363 C                        IDID=-1  INPUT IS NOT CONSISTENT,
    C                        IDID=-2  LARGER NMAX IS NEEDED,
365 C                        IDID=-3  STEP SIZE BECOMES TOO SMALL,
    C                        IDID=-4  MATRIX IS REPEATEDLY SINGULAR.
367 C
    C   IWORK(14)  NFCN     NUMBER OF FUNCTION EVALUATIONS (THOSE FOR NUMERICAL
369 C                       EVALUATION OF THE JACOBIAN ARE NOT COUNTED)
    C   IWORK(15)  NJAC     NUMBER OF JACOBIAN EVALUATIONS (EITHER ANALYTICALLY
371 C                       OR NUMERICALLY)
    C   IWORK(16)  NSTEP    NUMBER OF COMPUTED STEPS
373 C   IWORK(17)  NACCPT   NUMBER OF ACCEPTED STEPS
    C   IWORK(18)  NREJCT   NUMBER OF REJECTED STEPS (DUE TO ERROR TEST),
375 C                       (STEP REJECTIONS IN THE FIRST STEP ARE NOT COUNTED)
    C   IWORK(19)  NDEC     NUMBER OF LU-DECOMPOSITIONS OF BOTH MATRICES
377 C   IWORK(20)  NSOL     NUMBER OF FORWARD-BACKWARD SUBSTITUTIONS, OF BOTH
    C                       SYSTEMS; THE NSTEP FORWARD-BACKWARD SUBSTITUTIONS,
379 C                       NEEDED FOR STEP SIZE SELECTION, ARE NOT COUNTED
    C-----------------------------------------------------------------------
381 C *** *** *** *** *** *** *** *** *** *** *** *** ***
    C            DECLARATIONS
383 C *** *** *** *** *** *** *** *** *** *** *** *** ***
          IMPLICIT DOUBLE PRECISION (A-H,O-Z)
385       DIMENSION Y(N),ATOL(*),RTOL(*),WORK(LWORK),IWORK(LIWORK)
          DIMENSION RPAR(*),IPAR(*)
387       LOGICAL IMPLCT,JBAND,ARRET,STARTN,PRED
          EXTERNAL FCN,JAC,MAS,SOLOUT
389 C *** *** *** *** *** *** ***
    C        SETTING THE PARAMETERS
391 C *** *** *** *** *** *** ***
          NFCN=0
393       NJAC=0
```

```
      NSTEP=0
      NACCPT=0
      NREJCT=0
      NDEC=0
      NSOL=0
      ARRET=.FALSE.
C ---------- UROUND   SMALLEST NUMBER SATISFYING 1.0D0+UROUND>1.0D0
      IF (WORK(1).EQ.0.0D0) THEN
        UROUND=1.0D-16
      ELSE
        UROUND=WORK(1)
        IF (UROUND.LE.1.0D-19.OR.UROUND.GE.1.0D0) THEN
          WRITE(6,*) ' COEFFICIENTS HAVE 20 DIGITS, UROUND=',WORK(1)
          ARRET=.TRUE.
        END IF
      END IF
C ---------- CHECK AND CHANGE THE TOLERANCES
      EXPM=2.0D0/3.0D0
      IF (ITOL.EQ.0) THEN
        IF (ATOL(1).LE.0.D0.OR.RTOL(1).LE.10.D0*UROUND) THEN
          WRITE (6,*) ' TOLERANCES ARE TOO SMALL'
          ARRET=.TRUE.
        ELSE
          QUOT=ATOL(1)/RTOL(1)
          RTOL(1)=0.1D0*RTOL(1)**EXPM
          ATOL(1)=RTOL(1)*QUOT
        END IF
      ELSE
        DO I=1,N
          IF (ATOL(I).LE.0.D0.OR.RTOL(I).LE.10.D0*UROUND) THEN
            WRITE (6,*) ' TOLERANCES(',I,') ARE TOO SMALL'
            ARRET=.TRUE.
          ELSE
            QUOT=ATOL(I)/RTOL(I)
            RTOL(I)=0.1D0*RTOL(I)**EXPM
            ATOL(I)=RTOL(I)*QUOT
          END IF
        END DO
      END IF
C ---------- NMAX , THE MAXIMAL NUMBER OF STEPS -----
      IF (IWORK(2).EQ.0) THEN
        NMAX=100000
      ELSE
        NMAX=IWORK(2)
        IF (NMAX.LE.0) THEN
          WRITE(6,*) ' WRONG INPUT IWORK(2)=',IWORK(2)
          ARRET=.TRUE.
        END IF
      END IF
C ---------- NIT    MAXIMAL NUMBER OF NEWTON ITERATIONS
      IF (IWORK(3).EQ.0) THEN
        NIT=7
      ELSE
        NIT=IWORK(3)
        IF (NIT.LE.0) THEN
```

```
449               WRITE(6,*)' CURIOUS INPUT IWORK(3)=',IWORK(3)
                  ARRET=.TRUE.
451            END IF
          END IF
453   C ---------- STARTN  SWITCH FOR STARTING VALUES OF NEWTON ITERATIONS
          IF(IWORK(4).EQ.0)THEN
455          STARTN=.FALSE.
          ELSE
457          STARTN=.TRUE.
          END IF
459   C ---------- PARAMETER FOR DIFFERENTIAL-ALGEBRAIC COMPONENTS
          NIND1=IWORK(5)
461       NIND2=IWORK(6)
          NIND3=IWORK(7)
463       IF (NIND1.EQ.0) NIND1=N
          IF (NIND1+NIND2+NIND3.NE.N) THEN
465          WRITE(6,*)' CURIOUS INPUT FOR IWORK(5,6,7)=',NIND1,NIND2,NIND3
             ARRET=.TRUE.
467       END IF
      C ---------- PRED   STEP SIZE CONTROL
469       IF(IWORK(8).LE.1)THEN
             PRED=.TRUE.
471       ELSE
             PRED=.FALSE.
473       END IF
      C ---------- PARAMETER FOR SECOND ORDER EQUATIONS
475       M1=IWORK(9)
          M2=IWORK(10)
477       NM1=N-M1
          IF (M1.EQ.0) M2=N
479       IF (M2.EQ.0) M2=M1
          IF (M1.LT.0.OR.M2.LT.0.OR.M1+M2.GT.N) THEN
481          WRITE(6,*)' CURIOUS INPUT FOR IWORK(9,10)=',M1,M2
             ARRET=.TRUE.
483       END IF
      C ---------- SAFE     SAFETY FACTOR IN STEP SIZE PREDICTION
485       IF (WORK(2).EQ.0.0D0) THEN
             SAFE=0.9D0
487       ELSE
             SAFE=WORK(2)
489          IF (SAFE.LE.0.001D0.OR.SAFE.GE.1.0D0) THEN
                WRITE(6,*)' CURIOUS INPUT FOR WORK(2)=',WORK(2)
491             ARRET=.TRUE.
             END IF
493       END IF
      C ---------- THET     DECIDES WHETHER THE JACOBIAN SHOULD BE RECOMPUTED;
495       IF (WORK(3).EQ.0.D0) THEN
             THET=0.001D0
497       ELSE
             THET=WORK(3)
499          IF (THET.GE.1.0D0) THEN
                WRITE(6,*)' CURIOUS INPUT FOR WORK(3)=',WORK(3)
501             ARRET=.TRUE.
             END IF
503       END IF
```

```
      C ---- FNEWT   STOPPING CRITERION FOR NEWTON'S METHOD, USUALLY CHOSEN <1.
505         TOLST=RTOL(1)
            IF (WORK(4).EQ.0.D0) THEN
507            FNEWT=MAX(10*UROUND/TOLST,MIN(0.03D0,TOLST**0.5D0))
            ELSE
509            FNEWT=WORK(4)
               IF (FNEWT.LE.UROUND/TOLST) THEN
511               WRITE(6,*) ' CURIOUS INPUT FOR WORK(4)= ',WORK(4)
                  ARRET=.TRUE.
513            END IF
            END IF
515   C ---- QUOT1 AND QUOT2: IF QUOT1 < HNEW/HOLD < QUOT2, STEP SIZE = CONST.
            IF (WORK(5).EQ.0.D0) THEN
517            QUOT1=1.D0
            ELSE
519            QUOT1=WORK(5)
            END IF
521         IF (WORK(6).EQ.0.D0) THEN
               QUOT2=1.2D0
523         ELSE
               QUOT2=WORK(6)
525         END IF
            IF (QUOT1.GT.1.0D0.OR.QUOT2.LT.1.0D0) THEN
527            WRITE(6,*) ' CURIOUS INPUT FOR WORK(5,6)= ',QUOT1,QUOT2
               ARRET=.TRUE.
529         END IF
      C ---------- MAXIMAL STEP SIZE
531         IF (WORK(7).EQ.0.D0) THEN
               HMAX=XEND-X
533         ELSE
               HMAX=WORK(7)
535         END IF
      C ---------- FACL,FACR     PARAMETERS FOR STEP SIZE SELECTION
537         IF(WORK(8).EQ.0.D0)THEN
               FACL=5.D0
539         ELSE
               FACL=1.D0/WORK(8)
541         END IF
            IF(WORK(9).EQ.0.D0)THEN
543            FACR=1.D0/8.0D0
            ELSE
545            FACR=1.D0/WORK(9)
            END IF
547         IF (FACL.LT.1.0D0.OR.FACR.GT.1.0D0) THEN
               WRITE(6,*) ' CURIOUS INPUT WORK(8,9)= ',WORK(8),WORK(9)
549            ARRET=.TRUE.
            END IF
551   C *** *** *** *** *** *** *** *** *** *** *** *** ***
      C          COMPUTATION OF ARRAY ENTRIES
553   C *** *** *** *** *** *** *** *** *** *** *** *** ***
      C ------ IMPLICIT, BANDED OR NOT ?
555         IMPLCT=IMAS.NE.0
            JBAND=MLJAC.LT.NM1
557   C ---------- COMPUTATION OF THE ROW-DIMENSIONS OF THE 2-ARRAYS ---
      C -- JACOBIAN  AND  MATRICES E1, E2
```

```
559         IF (JBAND) THEN
              LDJAC=MLJAC+MUJAC+1
561           LDE1=MLJAC+LDJAC
            ELSE
563           MLJAC=NM1
              MUJAC=NM1
565           LDJAC=NM1
              LDE1=NM1
567         END IF
    C --- MASS MATRIX
569         IF (IMPLCT) THEN
              IF (MLMAS.NE.NM1) THEN
571             LDMAS=MLMAS+MUMAS+1
                IF (JBAND) THEN
573               IJOB=4
                ELSE
575               IJOB=3
                END IF
              ELSE
577             MUMAS=NM1
                LDMAS=NM1
579             IJOB=5
              END IF
    C ------- BANDWITH OF "MAS" NOT SMALLER THAN BANDWITH OF "JAC"
581           IF (MLMAS.GT.MLJAC.OR.MUMAS.GT.MUJAC) THEN
                WRITE (6,*) 'BANDWITH OF "MAS" NOT SMALLER THAN BANDWITH OF
583         & "JAC"'
                ARRET=.TRUE.
585           END IF
            ELSE
587           LDMAS=0
              IF (JBAND) THEN
589             IJOB=2
              ELSE
591             IJOB=1
                IF (N.GT.2.AND.IWORK(1).NE.0) IJOB=7
593           END IF
            END IF
595         LDMAS2=MAX(1,LDMAS)
    C ------- HESSENBERG OPTION ONLY FOR EXPLICIT EQU. WITH FULL JACOBIAN
597         IF ((IMPLCT.OR.JBAND).AND.IJOB.EQ.7) THEN
              WRITE(6,*)' HESSENBERG OPTION ONLY FOR EXPLICIT EQUATIONS WITH
599         &FULL JACOBIAN'
              ARRET=.TRUE.
601         END IF
    C ------- PREPARE THE ENTRY-POINTS FOR THE ARRAYS IN WORK -----
603         IEZ1=21
            IEZ2=IEZ1+N
605         IEZ3=IEZ2+N
            IEY0=IEZ3+N
607         IESCAL=IEY0+N
            IEF1=IESCAL+N
609         IEF2=IEF1+N
            IEF3=IEF2+N
611         IECON=IEF3+N
```

```
      IEJAC=IECON+4*N
      IEMAS=IEJAC+N*LDJAC
      IEE1=IEMAS+NM1*LDMAS
      IEE2R=IEE1+NM1*LDE1
      IEE2I=IEE2R+NM1*LDE1
C ------- TOTAL STORAGE REQUIREMENT ----------
      ISTORE=IEE2I+NM1*LDE1-1
      IF (ISTORE.GT.LWORK)THEN
        WRITE(6,*) ' INSUFFICIENT STORAGE FOR WORK, MIN. LWORK=',ISTORE
        ARRET=.TRUE.
      END IF
C -------- ENTRY POINTS FOR INTEGER WORKSPACE -----
      IEIP1=21
      IEIP2=IEIP1+NM1
      IEIPH=IEIP2+NM1
C ---------- TOTAL REQUIREMENT ------------------
      ISTORE=IEIPH+NM1-1
      IF (ISTORE.GT.LIWORK) THEN
        WRITE(6,*) ' INSUFF. STORAGE FOR IWORK, MIN. LIWORK=',ISTORE
        ARRET=.TRUE.
      END IF
C ------- WHEN A FAIL HAS OCCURED, WE RETURN WITH IDID=-1
      IF (ARRET) THEN
        IDID=-1
        RETURN
      END IF
C --------- CALL TO CORE INTEGRATOR ------------
      CALL RADCOR(N,NZM,FCN,X,Y,XEND,HMAX,H,RTOL,ATOL,ITOL,
     &     JAC,IJAC,MLJAC,MUJAC,MAS,MLMAS,MUMAS,SOLOUT,IOUT,IDID,
     &     NMAX,UROUND,SAFE,THET,FNEWT,QUOT1,QUOT2,NIT,IJOB,STARTN,
     &     NIND1,NIND2,NIND3,PRED,FACL,FACR,M1,M2,NM1,
     &     IMPLCT,JBAND,LDJAC,LDE1,LDMAS2,WORK(IEZ1),WORK(IEZ2),
     &     WORK(IEZ3),WORK(IEY0),WORK(IESCAL),WORK(IEF1),WORK(IEF2),
     &     WORK(IEF3),WORK(IEJAC),WORK(IEE1),WORK(IEE2R),WORK(IEE2I),
     &     WORK(IEMAS),IWORK(IEIP1),IWORK(IEIP2),IWORK(IEIPH),
     &     WORK(IECON),NFCN,NJAC,NSTEP,NACCPT,NREJCT,NDEC,NSOL,RPAR,IPAR)
      IWORK(14)=NFCN
      IWORK(15)=NJAC
      IWORK(16)=NSTEP
      IWORK(17)=NACCPT
      IWORK(18)=NREJCT
      IWORK(19)=NDEC
      IWORK(20)=NSOL
C ---------- RESTORE TOLERANCES
      EXPM=1.0D0/EXPM
      IF (ITOL.EQ.0) THEN
        QUOT=ATOL(1)/RTOL(1)
        RTOL(1)=(10.0D0*RTOL(1))**EXPM
        ATOL(1)=RTOL(1)*QUOT
      ELSE
        DO I=1,N
          QUOT=ATOL(I)/RTOL(I)
          RTOL(I)=(10.0D0*RTOL(I))**EXPM
          ATOL(I)=RTOL(I)*QUOT
        END DO
```

```
669         END IF
    C ------------ RETURN ------------
671         RETURN
            END
673 C
    C       END OF SUBROUTINE RADAU5
675 C
    C ***********************************************************
677 C
            SUBROUTINE RADCOR(N,NZM,FCN,X,Y,XEND,HMAX,H,RTOL,ATOL,ITOL,
679         &     JAC,IJAC,MLJAC,MUJAC,MAS,MLMAS,MUMAS,SOLOUT,IOUT,IDID,
            &     NMAX,UROUND,SAFE,THET,FNEWT,QUOT1,QUOT2,NIT,IJOB,STARTN,
681         &     NIND1,NIND2,NIND3,PRED,FACL,FACR,M1,M2,NM1,
            &     IMPLCT,BANDED,LDJAC,LDE1,LDMAS,Z1,Z2,Z3,
683         &     Y0,SCAL,F1,F2,F3,FJAC,E1,E2R,E2I,FMAS,IP1,IP2,IPHES,
            &     CONT,NFCN,NJAC,NSTEP,NACCPT,NREJCT,NDEC,NSOL,RPAR,IPAR)
685 C ----------------------------------------------------------
    C       CORE INTEGRATOR FOR RADAU5
687 C       PARAMETERS SAME AS IN RADAU5 WITH WORKSPACE ADDED
    C ----------------------------------------------------------
689 C            DECLARATIONS
    C ----------------------------------------------------------
691         IMPLICIT DOUBLE PRECISION (A-H,O-Z)
            DIMENSION Y(N),Z1(N),Z2(N),Z3(N),Y0(N),SCAL(N),F1(N),F2(N),F3(N)
693         DIMENSION FJAC(LDJAC,N),FMAS(LDMAS,NM1),CONT(4*N)
            DIMENSION E1(LDE1,NM1),E2R(LDE1,NM1),E2I(LDE1,NM1)
695         DIMENSION ATOL(*),RTOL(*),RPAR(*),IPAR(*)
            INTEGER IP1(NM1),IP2(NM1),IPHES(NM1)
697         COMMON /CONRA5/NN,NN2,NN3,NN4,XSOL,HSOL,C2M1,C1M1
            COMMON/LINAL/MLE,MUE,MBJAC,MBB,MDIAG,MDIFF,MBDIAG
699         LOGICAL REJECT,FIRST,IMPLCT,BANDED,CALJAC,STARTN,CALHES
            LOGICAL INDEX1,INDEX2,INDEX3,LAST,PRED
701         EXTERNAL FCN
            INTEGER, ALLOCATABLE, DIMENSION(:):: IND
703         INTEGER, ALLOCATABLE, DIMENSION(:):: IND_ROW_E1, IND_COL_E1
            REAL*8, ALLOCATABLE, DIMENSION(:):: VALUE_E1
705         INTEGER, ALLOCATABLE, DIMENSION(:):: IND_ROW_E2, IND_COL_E2
            DOUBLE COMPLEX, ALLOCATABLE, DIMENSION(:):: VALUE_E2
707         DIMENSION IND_ROW_JAC(NZM), IND_COL_JAC(NZM), VALUE_JAC(NZM)
            DIMENSION IND_DIAGN(N)
709         DIMENSION R21(N),R22(N)
    C *** *** *** *** *** *** ***
711 C  INITIALIZE MUMPS
    C *** *** *** *** *** *** ***
713         INCLUDE 'mpif.h'
            INCLUDE 'dmumps_struc.h'
715         INCLUDE 'zmumps_struc.h'
            TYPE (DMUMPS_STRUC) mumps_parE1
717         TYPE (ZMUMPS_STRUC) mumps_parE2
            CALL MPI_INIT(IERR)
719 C --- Define a communicator for the package.
            mumps_parE1%COMM = MPI_COMM_WORLD
721         mumps_parE2%COMM = MPI_COMM_WORLD
    C --- Initialize an instance of the package
723 C --- for LU factorization (sym = 0, with working host)
```

```
        mumps_parE1%JOB = -1
        mumps_parE1%SYM = 0
        mumps_parE1%PAR = 1
        mumps_parE2%JOB = -1
        mumps_parE2%SYM = 0
        mumps_parE2%PAR = 1
        CALL DMUMPS(mumps_parE1)
        IF (mumps_parE1%INFOG(1).LT.0) THEN
          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
     &                 "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
          GOTO 500
        END IF
        CALL ZMUMPS(mumps_parE2)
        IF (mumps_parE2%INFOG(1).LT.0) THEN
          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                 "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
     &                 "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
          GOTO 500
        END IF

        mumps_parE1%ICNTL(4)=-1
        mumps_parE2%ICNTL(4)=-1
C *** *** *** *** *** *** ***
C  INITIALISATIONS
C *** *** *** *** *** *** ***
C ---------- DUPLIFY N FOR COMMON BLOCK CONT -----
        NN=N
        NN2=2*N
        NN3=3*N
        LRC=4*N
C ---------- CHECK THE INDEX OF THE PROBLEM -----
        INDEX1=NIND1.NE.0
        INDEX2=NIND2.NE.0
        INDEX3=NIND3.NE.0
C ---------- COMPUTE MASS MATRIX FOR IMPLICIT CASE ----------
        IF (IMPLCT) CALL MAS(NM1,FMAS,LDMAS,RPAR,IPAR)
C ---------- CONSTANTS ----------
        SQ6=DSQRT(6.D0)
        C1=(4.D0-SQ6)/10.D0
        C2=(4.D0+SQ6)/10.D0
        C1M1=C1-1.D0
        C2M1=C2-1.D0
        C1MC2=C1-C2
        DD1=-(13.D0+7.D0*SQ6)/3.D0
        DD2=(-13.D0+7.D0*SQ6)/3.D0
        DD3=-1.D0/3.D0
        U1=(6.D0+81.D0**(1.D0/3.D0)-9.D0**(1.D0/3.D0))/30.D0
        ALPH=(12.D0-81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))/60.D0
        BETA=(81.D0**(1.D0/3.D0)+9.D0**(1.D0/3.D0))*DSQRT(3.D0)/60.D0
        CNO=ALPH**2+BETA**2
        U1=1.0D0/U1
        ALPH=ALPH/CNO
        BETA=BETA/CNO
        T11=9.123239487089294 2792D-02
```

```
779         T12=-0.14125529502095420843D0
            T13=-3.00291941051474244 92D-02
781         T21=0.24171793270710701896D0
            T22=0.20412935229379993199D0
783         T23=0.38294211275726193779D0
            T31=0.96604818261509293619D0
785         TI11=4.32557989006315535 10D0
            TI12=0.33919925181580986954D0
787         TI13=0.54177053993587487119D0
            TI21=-4.17871859155190472 73D0
789         TI22=-0.32768282076106238708D0
            TI23=0.47662355450055045196D0
791         TI31=-0.50287263494578687595D0
            TI32=2.57192694985560542 92D0
793         TI33=-0.59603920482822492497D0
            IF (M1.GT.0) IJOB=IJOB+10
795         POSNEG=SIGN(1.D0,XEND-X)

797         HMAXN=MIN(ABS(HMAX),ABS(XEND-X))
            IF (ABS(H).LE.10.D0*UROUND) H=1.0D-6
799         H=MIN(ABS(H),HMAXN)
            H=SIGN(H,POSNEG)
801         HOLD=H
            REJECT=.FALSE.
803         FIRST=.TRUE.
            LAST=.FALSE.
805         IF ((X+H*1.0001D0-XEND)*POSNEG.GE.0.D0) THEN
               H=XEND-X
807            LAST=.TRUE.
            END IF
809         HOPT=H
            FACCON=1.D0
811         CFAC=SAFE*(1+2*NIT)
            NSING=0
813         XOLD=X
            IF (IOUT.NE.0) THEN
815            IRTRN=1
               NRSOL=1
817            XOSOL=XOLD
               XSOL=X
819            DO I=1,N
                  CONT(I)=Y(I)
821            END DO
               NSOLU=N
823            HSOL=HOLD
               CALL SOLOUT(NRSOL,XOSOL,XSOL,Y,CONT,LRC,NSOLU,
825         &                  RPAR,IPAR,IRTRN)
               IF (IRTRN.LT.0) GOTO 179
827         END IF
            MLE=MLJAC
829         MUE=MUJAC
            MBJAC=MLJAC+MUJAC+1
831         MBB=MLMAS+MUMAS+1
            MDIAG=MLE+MUE+1
833         MDIFF=MLE+MUE-MUMAS
```

```
          MBDIAG=MUMAS+1
835       N2=2*N
          N3=3*N
837       IF (ITOL.EQ.0) THEN
            DO I=1,N
839           SCAL(I)=ATOL(1)+RTOL(1)*ABS(Y(I))
            END DO
841       ELSE
            DO I=1,N
843           SCAL(I)=ATOL(I)+RTOL(I)*ABS(Y(I))
            END DO
845       END IF
          HHFAC=H
847       CALL FCN(N,X,Y,Y0,RPAR,IPAR)
          NFCN=NFCN+1
849   C --- BASIC INTEGRATION STEP
       10 CONTINUE
851   C *** *** *** *** *** *** ***
      C  COMPUTATION OF THE JACOBIAN
853   C *** *** *** *** *** *** ***
          NJAC=NJAC+1
855       IF (IJAC.EQ.0) THEN
      C --- COMPUTE JACOBIAN MATRIX NUMERICALLY
857         IF (BANDED) THEN
      C --- JACOBIAN IS BANDED
859           MUJACP=MUJAC+1
              MD=MIN(MBJAC,M2)
861           DO MM=1,M1/M2+1
                DO K=1,MD
863               J=K+(MM-1)*M2
       12         F1(J)=Y(J)
865               F2(J)=DSQRT(UROUND*MAX(1.D-5,ABS(Y(J))))
                  Y(J)=Y(J)+F2(J)
867               J=J+MD
                  IF (J.LE.MM*M2) GOTO 12
869               CALL FCN(N,X,Y,CONT,RPAR,IPAR)
                  J=K+(MM-1)*M2
871               J1=K
                  LBEG=MAX(1,J1-MUJAC)+M1
873       14      LEND=MIN(M2,J1+MLJAC)+M1
                  Y(J)=F1(J)
875               MUJACJ=MUJACP-J1-M1
                  DO L=LBEG,LEND
877                 FJAC(L+MUJACJ,J)=(CONT(L)-Y0(L))/F2(J)
                  END DO
879               J=J+MD
                  J1=J1+MD
881               LBEG=LEND+1
                  IF (J.LE.MM*M2) GOTO 14
883             END DO
              END DO
885         ELSE
      C --- JACOBIAN IS FULL
887           DO I=1,N
                YSAFE=Y(I)
```

```
889                   DELT=DSQRT(UROUND*MAX(1.D-5,ABS(YSAFE)))
                      Y(I)=YSAFE+DELT
891                   CALL FCN(N,X,Y,CONT,RPAR,IPAR)
                      DO J=M1+1,N
893                     FJAC(J-M1,I)=(CONT(J)-Y0(J))/DELT
                      END DO
895                   Y(I)=YSAFE
                    END DO
897               END IF
                ELSE
899   C ---- COMPUTE JACOBIAN MATRIX ANALYTICALLY
                  CALL JAC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC)
901   C ---- CONTROL OF THE DIAGONAL. IF THE JACOBIAN HASN'T A VALUE ON
      C ---- THE DIAGIONAL, E1 HAS TO BE FAC1 AND E2 HAS TO BE ALPHN+iBETAN
903   C IND_DIAGN WILL CONTAIN THE DIAGONAL POSITIONS WHICH ARE FILLED
      C IF THERE ARE ZEROS ON THE DIAGONAL, IND_DIAGN WILL CONTAIN
905   C A NUMBER GREATER THAN N
                DO I=1,N
907               IND_DIAGN(I)=N+10
                END DO
909   C ND: NUMBER OF ELEMENTS ON THE DIAGONAL
                ND=0
911   C NNE: NUMBER OF NO ELEMENTS ON THE DIAGONAL
                NNE=1
913             DO I=1,NZM
                  IF (IND_ROW_JAC(I).EQ.IND_COL_JAC(I)) THEN
915                 ND=ND+1
                    IND_DIAGN(ND)=IND_ROW_JAC(I)
917               END IF
                END DO
919   C IND: A VECTOR CONTAINING THE INDICES OF THE DIAGONAL POSITIONS
      C         WITH A ZERO NUMBER
921             ALLOCATE(IND(N-ND))
      C LESS ELEMENTS ON THE DIAGONAL (ND-NUMBER OF ELEMENTS ON DIAGONAL)
923   C THAN THE SIZE OF THE PROBLEM (N)
                IF (ND.NE.N) THEN
925   C CHECK EVERY DIAGONAL POSITION
                  DO K=1,N
927                 NC=0
                    DO L=1,ND
929   C CHECK IF THERE CORRESPONDS AN INDICE OF THE JACOBIAN WITH THE
      C POSITION ON THE DIAGONAL
931   C AND IF THERE IS NO CORRESPONDENCE, IT WILL BE REMEMBERED
                      IF (IND_DIAGN(L).NE.K) THEN
933                     NC=NC+1
                        IF (NC.GE.ND) THEN
935                       IND(NNE)=K
                          NNE=NNE+1
937                     END IF
                      END IF
939                 END DO
                  END DO
941             END IF
      C NZE: THE NUMBER OF NONZERO ELEMENTS IN THE MATRICES E1 AND E2
943             NZME=NZM+NNE-1
```

```
            ALLOCATE(IND_ROW_E1(NZME))
            ALLOCATE(IND_COL_E1(NZME))
            ALLOCATE(VALUE_E1(NZME))
            ALLOCATE(IND_ROW_E2(NZME))
            ALLOCATE(IND_COL_E2(NZME))
            ALLOCATE(VALUE_E2(NZME))

          END IF

          CALJAC=.TRUE.
          CALHES=.TRUE.
       20 CONTINUE
C ---- COMPUTE THE MATRICES E1 AND E2 AND THEIR DECOMPOSITIONS
          FAC1=U1/H
          ALPHN=ALPH/H
          BETAN=BETA/H

C         SUBROUTINE DECOMR(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC,IND_ROW_E1,
C        &                  IND_ROW_E1,VALUE_E1)
C ---- MAKE MATRIX E1
          IND_ROW_E1=IND_ROW_JAC
          IND_COL_E1=IND_COL_JAC
          DO J=1,NZM
            IF (IND_ROW_JAC(J).EQ.IND_COL_JAC(J)) THEN
              VALUE_E1(J)=FAC1-VALUE_JAC(J)
            ELSE
              VALUE_E1(J)=-VALUE_JAC(J)
            END IF
          END DO

          DO J=1,NNE-1
            IND_ROW_E1(NZM+J)=IND(J)
            IND_COL_E1(NZM+J)=IND(J)
            VALUE_E1(NZM+J)=FAC1
          END DO

          mumps_parE1%NZ=NZME
          mumps_parE1%N=N

          ALLOCATE(mumps_parE1%IRN(mumps_parE1%NZ))
          ALLOCATE(mumps_parE1%JCN(mumps_parE1%NZ))
          ALLOCATE(mumps_parE1%A(mumps_parE1%NZ))
          ALLOCATE(mumps_parE1%RHS(mumps_parE1%N))

          mumps_parE1%IRN=IND_ROW_E1
          mumps_parE1%JCN=IND_COL_E1
          mumps_parE1%A=VALUE_E1

C ---- Call package for the analysis (JOB=1) an the factorisation (JOB=2)
          mumps_parE1%JOB = 1
          CALL DMUMPS(mumps_parE1)
          IF (mumps_parE1%INFOG(1).EQ.-6) GOTO 78
          IF (mumps_parE1%INFOG(1).LT.0) THEN
            WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
         &           "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
```

```
999         &                 "   mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
              GOTO 500
1001        END IF

1003        mumps_parE1%JOB = 2
            CALL DMUMPS(mumps_parE1)
1005        IF (mumps_parE1%INFOG(1).EQ.-10) GOTO 78
            IF (mumps_parE1%INFOG(1).LT.0) THEN
1007          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
            &                 "   mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
1009        &                 "   mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
              GOTO 500
1011        END IF

1013    C     SUBROUTINE DECOMC(N,NZM,VALUE_JAC,IND_ROW_JAC,IND_COL_JAC,IND_ROW_E2,
        C    &                  IND_ROW_E2,VALUE_E2)
1015    C --- MAKE MATRIX E2
            IND_ROW_E2=IND_ROW_JAC
1017        IND_COL_E2=IND_COL_JAC
            DO J=1,NZM
1019          IF (IND_ROW_JAC(J).EQ.IND_COL_JAC(J)) THEN
                VALUE_E2(J)=DCMPLX(ALPHN,BETAN)-VALUE_JAC(J)
1021          ELSE
                VALUE_E2(J)=-VALUE_JAC(J)
1023          END IF
            END DO

1025
            DO J=1,NNE-1
1027          IND_ROW_E2(NZM+J)=IND(J)
              IND_COL_E2(NZM+J)=IND(J)
1029          VALUE_E2(NZM+J)=DCMPLX(ALPHN,BETAN)
            END DO

1031
        C --- TRANSFORM TO NAMES MUMPS
1033        mumps_parE2%NZ=NZME
            mumps_parE2%N=N

1035
            ALLOCATE(mumps_parE2%IRN(mumps_parE2%NZ))
1037        ALLOCATE(mumps_parE2%JCN(mumps_parE2%NZ))
            ALLOCATE(mumps_parE2%A(mumps_parE2%NZ))
1039        ALLOCATE(mumps_parE2%RHS(mumps_parE2%N))

1041        mumps_parE2%IRN=IND_ROW_E2
            mumps_parE2%JCN=IND_COL_E2
1043        mumps_parE2%A=VALUE_E2

1045    C --- Call package for the analysis (JOB=1) an the factorisation (JOB=2)
            mumps_parE2%JOB = 1
1047        CALL ZMUMPS(mumps_parE2)
            IF (mumps_parE1%INFOG(1).EQ.-6) GOTO 78
1049        IF (mumps_parE2%INFOG(1).LT.0) THEN
              WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
            &                 "   mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
1051        &                 "   mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
              GOTO 500
1053
```

```
      END IF
1055
      mumps_parE2%JOB = 2
1057  CALL ZMUMPS(mumps_parE2)
      IF (mumps_parE1%INFOG(1).EQ.-10) GOTO 78
1059  IF (mumps_parE2%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
1061  &                 "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
      &                 "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
1063    GOTO 500
      END IF
1065
      NDEC=NDEC+1
1067  30 CONTINUE
      NSTEP=NSTEP+1
1069  IF (NSTEP.GT.NMAX) GOTO 178
      IF (0.1D0*ABS(H).LE.ABS(X)*UROUND) GOTO 177
1071  IF (INDEX2) THEN
        DO I=NIND1+1,NIND1+NIND2
1073      SCAL(I)=SCAL(I)/HHFAC
        END DO
1075  END IF
      IF (INDEX3) THEN
1077    DO I=NIND1+NIND2+1,NIND1+NIND2+NIND3
          SCAL(I)=SCAL(I)/(HHFAC*HHFAC)
1079    END DO
      END IF
1081  XPH=X+H
C *** *** *** *** *** *** ***
1083  C  STARTING VALUES FOR NEWTON ITERATION
C *** *** *** *** *** *** ***
1085  IF (FIRST.OR.STARTN) THEN
        DO I=1,N
1087      Z1(I)=0.D0
          Z2(I)=0.D0
1089      Z3(I)=0.D0
          F1(I)=0.D0
1091      F2(I)=0.D0
          F3(I)=0.D0
1093    END DO
      ELSE
1095    C3Q=H/HOLD
        C1Q=C1*C3Q
1097    C2Q=C2*C3Q
        DO I=1,N
1099      AK1=CONT(I+N)
          AK2=CONT(I+N2)
1101      AK3=CONT(I+N3)
          Z1I=C1Q*(AK1+(C1Q-C2M1)*(AK2+(C1Q-C1M1)*AK3))
1103      Z2I=C2Q*(AK1+(C2Q-C2M1)*(AK2+(C2Q-C1M1)*AK3))
          Z3I=C3Q*(AK1+(C3Q-C2M1)*(AK2+(C3Q-C1M1)*AK3))
1105      Z1(I)=Z1I
          Z2(I)=Z2I
1107      Z3(I)=Z3I
          F1(I)=TI11*Z1I+TI12*Z2I+TI13*Z3I
```

```
1109              F2(I)=TI21*Z1I+TI22*Z2I+TI23*Z3I
                  F3(I)=TI31*Z1I+TI32*Z2I+TI33*Z3I
1111            END DO
              END IF
1113  C *** *** *** *** *** *** ***
      C   LOOP FOR THE SIMPLIFIED NEWTON ITERATION
1115  C *** *** *** *** *** *** ***
              NEWT=0
1117          FACCON=MAX(FACCON,UROUND)**0.8D0
              THETA=ABS(THET)
1119   40     CONTINUE
              IF (NEWT.GE.NIT) GOTO 78
1121  C --- COMPUTE THE RIGHT-HAND SIDE
              DO I=1,N
1123            CONT(I)=Y(I)+Z1(I)
              END DO
1125          CALL FCN(N,X+C1*H,CONT,Z1,RPAR,IPAR)
              DO I=1,N
1127            CONT(I)=Y(I)+Z2(I)
              END DO
1129          CALL FCN(N,X+C2*H,CONT,Z2,RPAR,IPAR)
              DO I=1,N
1131            CONT(I)=Y(I)+Z3(I)
              END DO
1133          CALL FCN(N,XPH,CONT,Z3,RPAR,IPAR)
              NFCN=NFCN+3
1135  C --- SOLVE THE LINEAR SYSTEMS
              DO I=1,N
1137            A1=Z1(I)
                A2=Z2(I)
1139            A3=Z3(I)
                Z1(I)=TI11*A1+TI12*A2+TI13*A3
1141            Z2(I)=TI21*A1+TI22*A2+TI23*A3
                Z3(I)=TI31*A1+TI32*A2+TI33*A3
1143          END DO
      C         CALL SLVRAD(N,FJAC,LDJAC,MLJAC,MUJAC,FMAS,LDMAS,MLMAS,MUMAS,
1145  C     &                M1,M2,NM1,FAC1,ALPHN,BETAN,E1,E2R,E2I,LDE1,Z1,Z2,Z3,
      C     &                F1,F2,F3,CONT,IP1,IP2,IPHES,IER,IJOB)
1147          DO I=1,N
                S2=-F2(I)
1149            S3=-F3(I)
                R21(I)=Z1(I)
1151            R22(I)=Z2(I)
                Z1(I)=Z1(I)-F1(I)*FAC1
1153            Z2(I)=Z2(I)+S2*ALPHN-S3*BETAN
                CONT(I)=Z3(I)+S3*ALPHN+S2*BETAN
1155          END DO

1157  C --- TRANSFORM NAME RHS TO NAME MUMPS
              DO I=1,N
1159            mumps_parE1%RHS(I)=Z1(I)
              END DO

1161
      C --- Call package for solution
1163          mumps_parE1%JOB = 3
```

```
      CALL DMUMPS(mumps_parE1)
      IF (mumps_parE1%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                    "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
     &                    "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
        GOTO 500
      END IF
      DO I=1,N
        Z1(I)=mumps_parE1%RHS(I)
      END DO

C --- TRANSFORM NAME RHS TO NAME MUMPS
      DO I=1,N
        mumps_parE2%RHS(I)=DCMPLX(Z2(I),CONT(I))
      END DO

C --- Call package for solution
      mumps_parE2%JOB = 3
      CALL ZMUMPS(mumps_parE2)
      IF (mumps_parE2%INFOG(1).LT.0) THEN
        WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
     &                    "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
     &                    "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
        GOTO 500
      END IF

      DO I=1,N
        Z2(I)=REAL(mumps_parE2%RHS(I))
        Z3(I)=REAL(AIMAG(mumps_parE2%RHS(I)))
      END DO
      NSOL=NSOL+1
      NEWT=NEWT+1
      DYNO=0.D0
      DO I=1,N
        DENOM=SCAL(I)
        DYNO=DYNO+(Z1(I)/DENOM)**2+(Z2(I)/DENOM)**2
     &           +(Z3(I)/DENOM)**2
      END DO
      DYNO=DSQRT(DYNO/N3)
C --- BAD CONVERGENCE OR NUMBER OF ITERATIONS TO LARGE
      IF (NEWT.GT.1.AND.NEWT.LT.NIT) THEN
        THQ=DYNO/DYNOLD
        IF (NEWT.EQ.2) THEN
          THETA=THQ
        ELSE
          THETA=SQRT(THQ*THQOLD)
        END IF
        THQOLD=THQ
        IF (THETA.LT.0.99D0) THEN
          FACCON=THETA/(1.0D0-THETA)
          DYTH=FACCON*DYNO*THETA**(NIT-1-NEWT)/FNEWT
          IF (DYTH.GE.1.0D0) THEN
            QNEWT=DMAX1(1.0D-4,DMIN1(20.0D0,DYTH))
            HHFAC=.8D0*QNEWT**(-1.0D0/(4.0D0+NIT-1-NEWT))
            H=HHFAC*H
```

```
                REJECT=.TRUE.
                LAST=.FALSE.
                IF (CALJAC) GOTO 20
                GOTO 10
              END IF
            ELSE
              GOTO 78
             END IF
          END IF
          DYNOLD=MAX(DYNO,UROUND)
          DO I=1,N
            F1I=F1(I)+Z1(I)
            F2I=F2(I)+Z2(I)
            F3I=F3(I)+Z3(I)
            F1(I)=F1I
            F2(I)=F2I
            F3(I)=F3I
            Z1(I)=T11*F1I+T12*F2I+T13*F3I
            Z2(I)=T21*F1I+T22*F2I+T23*F3I
            Z3(I)=T31*F1I+     F2I
          END DO
          IF (FACCON*DYNO.GT.FNEWT) GOTO 40
C ---- ERROR ESTIMATION
C         CALL ESTRAD (N,FJAC,LDJAC,MLJAC,MUJAC,FMAS,LDMAS,MLMAS,MUMAS,
C        &             H,DD1,DD2,DD3,FCN,NFCN,Y0,Y,IJOB,X,M1,M2,NM1,
C        &             E1,LDE1,Z1,Z2,Z3,CONT,F1,F2,IP1,IPHES,SCAL,ERR,
C        &             FIRST,REJECT,FAC1,RPAR,IPAR)
          HEE1=DD1/H
          HEE2=DD2/H
          HEE3=DD3/H
          DO  I=1,N
            F2(I)=HEE1*Z1(I)+HEE2*Z2(I)+HEE3*Z3(I)
            CONT(I)=F2(I)+Y0(I)
          END DO
C ---- TRANSFORM NAME RHS TO NAME MUMPS
          DO I=1,N
            mumps_parE1%RHS(I)=CONT(I)
          END DO
C ---- Call package for solution
          mumps_parE1%JOB = 3
          CALL DMUMPS(mumps_parE1)
          IF (mumps_parE1%INFOG(1).LT.0) THEN
            WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
         &                    "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
         &                    "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
            GOTO 500
          END IF
          DO I=1,N
            CONT(I)=mumps_parE1%RHS(I)
          END DO
          ERR=0.D0
          DO  I=1,N
            ERR=ERR+(CONT(I)/SCAL(I))**2
          END DO
          ERR=MAX(SQRT(ERR/N),1.D-10)
```

```
      IF (ERR.GE.1.D0) THEN
1275    IF (FIRST.OR.REJECT) THEN
          DO I=1,N
1277        CONT(I)=Y(I)+CONT(I)
          END DO
1279      CALL FCN(N,X,CONT,F1,RPAR,IPAR)
          NFCN=NFCN+1
1281      DO I=1,N
            CONT(I)=F1(I)+F2(I)
1283      END DO
C ---  TRANSFORM NAME RHS TO NAME MUMPS
1285      DO I=1,N
            mumps_parE1%RHS(I)=CONT(I)
1287      END DO
C ---  Call package for solution
1289        mumps_parE1%JOB = 3
          CALL DMUMPS(mumps_parE1)
1291      IF (mumps_parE1%INFOG(1).LT.0) THEN
            WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
1293      &                      "  mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
          &                      "  mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
1295        GOTO 500
          END IF
1297      DO I=1,N
            CONT(I)=mumps_parE1%RHS(I)
1299      END DO
          ERR=0.D0
1301      DO I=1,N
            ERR=ERR+(CONT(I)/SCAL(I))**2
1303      END DO
          ERR=MAX(SQRT(ERR/N),1.D-10)
1305    END IF
      END IF
1307  C ---  COMPUTATION OF HNEW
      C ---  WE REQUIRE .2<=HNEW/H<=8.
1309      FAC=MIN(SAFE,CFAC/(NEWT+2*NIT))
          QUOT=MAX(FACR,MIN(FACL,ERR**.25D0/FAC))
1311      HNEW=H/QUOT
      C *** *** *** *** *** *** ***
1313  C   IS THE ERROR SMALL ENOUGH ?
      C *** *** *** *** *** *** ***
1315      IF (ERR.LT.1.D0) THEN
      C ---  STEP IS ACCEPTED
1317        FIRST=.FALSE.
          NACCPT=NACCPT+1
1319      IF (PRED) THEN
      C         ---  PREDICTIVE CONTROLLER OF GUSTAFSSON
1321        IF (NACCPT.GT.1) THEN
              FACGUS=(HACC/H)*(ERR**2/ERRACC)**0.25D0/SAFE
1323          FACGUS=MAX(FACR,MIN(FACL,FACGUS))
              QUOT=MAX(QUOT,FACGUS)
1325          HNEW=H/QUOT
          END IF
1327      HACC=H
          ERRACC=MAX(1.0D-2,ERR)
```

```
1329            END IF
                XOLD=X
1331            HOLD=H
                X=XPH
1333            DO I=1,N
                   Y(I)=Y(I)+Z3(I)
1335               Z2I=Z2(I)
                   Z1I=Z1(I)
1337               CONT(I+N)=(Z2I-Z3(I))/C2M1
                   AK=(Z1I-Z2I)/C1MC2
1339               ACONT3=Z1I/C1
                   ACONT3=(AK-ACONT3)/C2
1341               CONT(I+N2)=(AK-CONT(I+N))/C1M1
                   CONT(I+N3)=CONT(I+N2)-ACONT3
1343            END DO
                IF (ITOL.EQ.0) THEN
1345               DO I=1,N
                      SCAL(I)=ATOL(1)+RTOL(1)*ABS(Y(I))
1347               END DO
                ELSE
1349               DO I=1,N
                      SCAL(I)=ATOL(I)+RTOL(I)*ABS(Y(I))
1351               END DO
                END IF
1353            IF (IOUT.NE.0) THEN
                   NRSOL=NACCPT+1
1355               XSOL=X
                   XOSOL=XOLD
1357               DO I=1,N
                      CONT(I)=Y(I)
1359               END DO
                   NSOLU=N
1361               HSOL=HOLD
                   CALL SOLOUT(NRSOL,XOSOL,XSOL,Y,CONT,LRC,NSOLU,
1363           &                   RPAR,IPAR,IRTRN)
                   IF (IRTRN.LT.0) GOTO 179
1365            END IF
                CALJAC=.FALSE.
1367            IF (LAST) THEN
                   H=HOPT
1369               IDID=1
                   RETURN
1371            END IF
                CALL FCN(N,X,Y,Y0,RPAR,IPAR)
1373            NFCN=NFCN+1
                HNEW=POSNEG*MIN(ABS(HNEW),HMAXN)
1375            HOPT=HNEW
                HOPT=MIN(H,HNEW)
1377            IF (REJECT) HNEW=POSNEG*MIN(ABS(HNEW),ABS(H))
                REJECT=.FALSE.
1379            IF ((X+HNEW/QUOT1-XEND)*POSNEG.GE.0.D0) THEN
                   H=XEND-X
1381               LAST=.TRUE.
                ELSE
1383               QT=HNEW/H
```

```
            HHFAC=H
1385        IF (THETA.LE.THET.AND.QT.GE.QUOT1.AND.QT.LE.QUOT2) GOTO 30
            H=HNEW
1387      END IF
          HHFAC=H
1389      IF (THETA.LE.THET) GOTO 20
          GOTO 10
1391    ELSE
C ---   STEP IS REJECTED
1393      REJECT=.TRUE.
          LAST=.FALSE.
1395      IF (FIRST) THEN
            H=H*0.1D0
1397        HHFAC=0.1D0
          ELSE
1399        HHFAC=HNEW/H
            H=HNEW
1401      END IF
          IF (NACCPT.GE.1) NREJCT=NREJCT+1
1403      IF (CALJAC) GOTO 20
          GOTO 10
1405    END IF
C *** *** *** *** *** *** ***
1407 C  DESTROY MUMPS
C *** *** *** *** *** *** ***
1409 C --- Deallocate user data
        IF ( mumps_parE1%MYID .eq. 0 )THEN
1411      DEALLOCATE( mumps_parE1%IRN )
          DEALLOCATE( mumps_parE1%JCN )
1413      DEALLOCATE( mumps_parE1%A )
          DEALLOCATE( mumps_parE1%RHS )
1415    END IF
        IF ( mumps_parE2%MYID .eq. 0 )THEN
1417      DEALLOCATE( mumps_parE2%IRN )
          DEALLOCATE( mumps_parE2%JCN )
1419      DEALLOCATE( mumps_parE2%A )
          DEALLOCATE( mumps_parE2%RHS )
1421    END IF
        DEALLOCATE(IND_ROW_E1)
1423    DEALLOCATE(IND_COL_E1)
        DEALLOCATE(VALUE_E1)
1425    DEALLOCATE(IND_ROW_E2)
        DEALLOCATE(IND_COL_E2)
1427    DEALLOCATE(VALUE_E2)
        DEALLOCATE(IND)
1429
C --- Destroy the instance (deallocate internal data structures)
1431    mumps_parE1%JOB = -2
        CALL DMUMPS(mumps_parE1)
1433    IF (mumps_parE1%INFOG(1).LT.0) THEN
          WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
1435    &              " mumps_parE1%INFOG(1)= ", mumps_parE1%INFOG(1),
        &              " mumps_parE1%INFOG(2)= ", mumps_parE1%INFOG(2)
1437      GOTO 500
        END IF
```

```
1439        mumps_parE2%JOB = -2
            CALL ZMUMPS(mumps_parE2)
1441        IF (mumps_parE2%INFOG(1).LT.0) THEN
              WRITE(6,'(A,A,I6,A,I9)') " ERROR RETURN: ",
1443        &                 "  mumps_parE2%INFOG(1)= ", mumps_parE2%INFOG(1),
            &                 "  mumps_parE2%INFOG(2)= ", mumps_parE2%INFOG(2)
1445          GOTO 500
            END IF
1447
 500    CALL MPI_FINALIZE(IERR)
1449  C *** *** *** *** *** *** ***
      C --- UNEXPECTED STEP-REJECTION
1451   78   CONTINUE
            IF (IER.NE.0) THEN
1453          NSING=NSING+1
              IF (NSING.GE.5) GOTO 176
1455        END IF
            H=H*0.5D0
1457        HHFAC=0.5D0
            REJECT=.TRUE.
1459        LAST=.FALSE.
            IF (CALJAC) GOTO 20
1461        GOTO 10
      C --- FAIL EXIT
1463  176   CONTINUE
            WRITE(6,979)X
1465        WRITE(6,*) ' MATRIX IS REPEATEDLY SINGULAR, IER=',IER
            IDID=-4
1467        RETURN
  177   CONTINUE
1469        WRITE(6,979)X
            WRITE(6,*) ' STEP SIZE T0O SMALL, H=',H
1471        IDID=-3
            RETURN
1473  178   CONTINUE
            WRITE(6,979)X
1475        WRITE(6,*) ' MORE THAN NMAX =',NMAX,'STEPS ARE NEEDED'
            IDID=-2
1477        RETURN
      C --- EXIT CAUSED BY SOLOUT
1479  179   CONTINUE
            WRITE(6,979)X
1481   979  FORMAT(' EXIT OF RADAU5 AT X=',E18.4)
            IDID=2
1483        RETURN
            END
1485  C
      C       END OF SUBROUTINE RADCOR
1487  C
      C ***********************************************************
1489  C DOUBLE PRECISION FUNCTION CONTR5(I,X,CONT,LRC)
      C ----------------------------------------------------------
1491  C       THIS FUNCTION CAN BE USED FOR CONINUOUS OUTPUT. IT PROVIDES AN
      C       APPROXIMATION TO THE I-TH COMPONENT OF THE SOLUTION AT X.
1493  C       IT GIVES THE VALUE OF THE COLLOCATION POLYNOMIAL, DEFINED FOR
```

```
C       THE LAST SUCCESSFULLY COMPUTED STEP (BY RADAU5).
C -----------------------------------------------------------
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      DIMENSION CONT(LRC)
      COMMON /CONRA5/NN,NN2,NN3,NN4,XSOL,HSOL,C2M1,C1M1
      S=(X-XSOL)/HSOL
      CONTR5=CONT(I)+S*(CONT(I+NN)+(S-C2M1)*(CONT(I+NN2)
     &       +(S-C1M1)*CONT(I+NN3)))
      RETURN
      END
C
C       END OF FUNCTION CONTR5
C
C ***********************************************************
```

# References


[1] N.J. Higham, *Functions of Matrices, Theory and Computation*, SIAM (Society for Industrial and Applied Mathematics), 2008

[2] A. Stankoskiy and G. Van den Eynde, *Advanced Method for Calculations of Core Burn-Up, Activation of Structural Materials, and Spallation Products Accumulation in Accelerator-Driven Systems*, Science and Technology of Nuclear Installations, volume 2012, Article ID 545103, 2012, 12 pages

[3] Nuclear Power for Everybody, *Neutron Cross-section*, 2012. Available at http://www.nuclear-power.net/neutron-cross-section/#prettyPhoto

[4] Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario, 2015, *Maple (Version 2015.1)* [Computer program]. Available at http://www.maplesoft.com/

[5] M. Pusa and J. Leppänen, *Computing the Matrix Exponential in Burnup Calculations*, Nuclear science and engineering, Vol 164, feb 2010, 140-150

[6] N.J. Higham, *The Scaling and Squaring Method for the Matrix Exponential Revisited*, SIAM Review, Vol 51, No 4, 2009, pp. 747-764

[7] A.H. Al-Mohy and N.J. Higham, *A New Scaling and Squaring Algorithm for the Matrix Exponential*, SIAM J. Matrix Anal. Appl., Vol 31, No 3, 2009, pp. 970-989

[8] A. H. Al-Mohy and N. J. Higham, *Computing the Action of the Matrix Exponential, with an Application to Exponential Integrators*, Siam J. Sci Comput, Vol 33, No 2, 2011, pp. 488-511

[9] M. Pusa and J. Leppänen *Solving Linear Systems with Sparse Gaussian Elimination in the Chebyshev Rational Approximation Method*, Nucl. Sci Eng, Vol 175, 2013, pp. 250-258

[10] E. Jones, T. Oliphant, P. Peterson and others, 2001, *SciPy: Open source scientific tools for Python* [Computer program]. Available at http://www.scipy.org/

[11] M. Van Daele, *Cursus Numerieke methoden voor stelsels differentiaalvergelijkingen*, editie 2014-2015

[12] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, Springer, 2010, second edition

[13] A. Bultheel, *Inleiding tot de numerieke wiskunde*, Acco, 2007, tweede druk

[14] Python Software Foundation, Last updated on 2016, *Python (Version 2.7.10)* [Computer program]. Available at https://pythonhosted.org/spyder/ (Spyder: Scientic PYthon Development EnviRonment)

[15] Andersson, Christian and Führer, Claus and Åkesson, Johan, *Assimulo: A unified framework for {ODE} solvers*, Mathematics and Computers in Simulation, Vol 116, 0, 2015, pp 26-43. Available at http://dx.doi.org/10.1016/j.matcom.2015.04.007

[16] E. Hairer and G. Wanner, *RADAU5 solver based on the book Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems* [Computer program]. Available at http://www.unige.ch/∼hairer/software.html

[17] S. Blackford, *CRS Matrix-Vector Product*, 2000. Available at http://www.netlib.org/utk/people/Jack Dongarra/etemplates/node382.html

[18] Free Software Foundation, Inc, 2011, *GNU Fortran (4.6.3)* [Computer program]. Available at https://help.ubuntu.com/community/InstallingCompilers#Installing_the_GNU _Fortran_compilers

[19] Netlib, Updated November 2015, *BLAS: Basic Linear Algebra Subprograms (3.6.0)* [Computer program]. Available at http://www.netlib.org/blas/

[20] *MUMPS (5.0.1)* [Computer program]. Available at http://mumps.enseeiht.fr/index.php?page=home
P. R. Amestoy, I. S. Duff, J. Koster and J.-Y. L'Excellent, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal of Matrix Analysis and Applications, Vol 23, No 1, pp 15-41 (2001)
P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing, Vol 32 (2), pp 136-156 (2006)

[21] Intel Corporation, 1985-2014, *IFORT (15.0.1 20141023)* [Computer program]. Available at https://software.intel.com/en-us/

[22] HSL. A collection of Fortran codes for large scale scientific computation. Available at http://www.hsl.rl.ac.uk/
T. A. Davis, University of Florida, and I. S. Duff, Rutherford Appleton Laboratory, October 1995, *MA38 Sparse unsymmetric system: unsymmetric multifrontal method (1.1.0)* [Computer program]. Available at http://www.hsl.rl.ac.uk/specs/ma38.pdf

[23] HSL. A collection of Fortran codes for large scale scientific computation. Available at http://www.hsl.rl.ac.uk/
T. A. Davis, University of Florida, and I. S. Duff, Rutherford Appleton Laboratory, September 2000, *ME38 Sparse unsymmetric system: unsymmetric multifrontal method (1.1.1)* [Computer program]. Available http://www.hsl.rl.ac.uk/specs/me38.pdf