Bateman Equation: The Polonium Problem

University of Bologna

Theoretical and Numerical Aspects of Nuclear Physics

A.A. 2021/22

Introduction: the Bateman Equation

The Bateman equations are a set of first order ordinary differential equations of the form:

$$n'(t) = An(t) \qquad n(0) = n_0$$

n(t): nuclide concentration vector
A: Bateman matrix
n_0: initial nuclide concentration

Exact solution $\longrightarrow$ $n(t) = e^{At} n_0$

Matrix Exponential Method

Different algorithm could be used to solve the system of ODE. Here we focus on the ***matrix exponential method***, which consists in calculating the exponential of the system matrix. In mathematics, the matrix exponential is a matrix function on square matrices analogous to the ordinary exponential function.

On Python, in order to compute the exponential of a matrix the library *scipy* is needed; in particular, the function **scipy.linalg.expm***. This function implements the algorithm in ref. [2], which is a Pade approximation with a variable order that is decided based on the array data.

*[https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.expm.html]

The Polonium Problem [1]

A simple example of the Bateman equation is represented by the Polonium-210. It is the most widely available isotope of Polonium and is made via neutron capture by Bismuth-209. The production chain is the following:

$$^{209}Bi \xrightarrow{(n,\gamma)} {}^{210}Bi \xrightarrow{5,013d} {}^{210}Po \xrightarrow{138,376d} {}^{210}Pb$$

Lead-210 is a stable nuclide. The change in the nuclides concentration over time is:

$$\frac{dn_{Bi209}}{dt} = -\lambda_{(Bi209)} n_{Bi209}$$

$$\frac{dn_{Bi210}}{dt} = \lambda_{(Bi209)} n_{Bi209} - \lambda_{(Bi210)} n_{Bi210}$$

$$\frac{dn_{Po210}}{dt} = \lambda_{(Bi210)} n_{Bi210} - \lambda_{(Po210)} n_{Po210}$$

$\lambda_{nuclide}$: *decay constant of the nuclide*

The Polonium Problem

The Bateman equations can be rewritten in the matrix form:

$$\frac{d}{dt}\begin{pmatrix} n_{Bi209} \\ n_{Bi210} \\ n_{Po210} \end{pmatrix} = \begin{pmatrix} -\lambda_{Bi209} & 0 & 0 \\ \lambda_{Bi209} & -\lambda_{Bi210} & 0 \\ 0 & \lambda_{Bi210} & \lambda_{Po210} \end{pmatrix}\begin{pmatrix} n_{Bi209} \\ n_{Bi210} \\ n_{Po210} \end{pmatrix}$$

The matrix on the right-hand side is the Bateman matrix, which is used to find the nuclides concentration over a given period of time using the matrix exponential method on Python (v. 3.9.4).

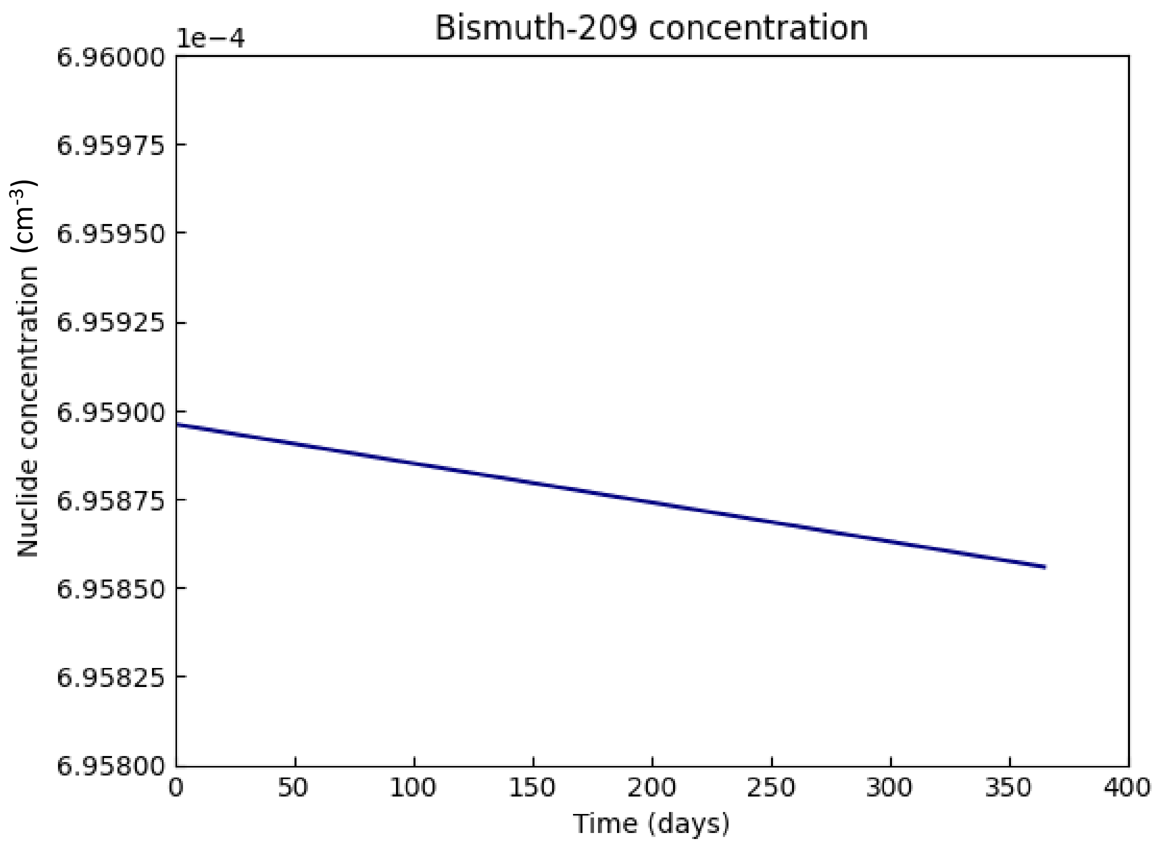
The Polonium Problem

In the following table are the constants used in the Bateman equations*:

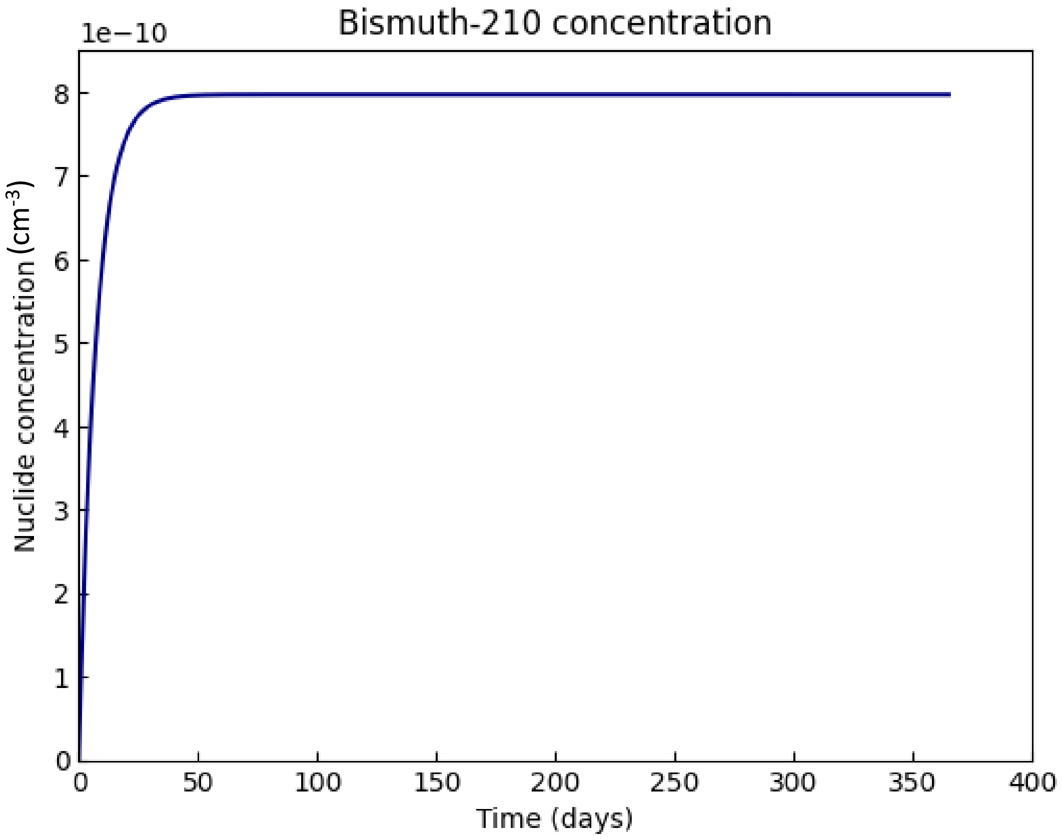
λ_{Bi209}	$1.83163 \cdot 10^{-12}\ s^{-1}$
λ_{Bi210}	$1.60035 \cdot 10^{-6}\ s^{-1}$
λ_{Po210}	$5.79764 \cdot 10^{-8}\ s^{-1}$
$n_{Bi209}(0)$	$6.95896 \cdot 10^{-4}\ cm^{-3}$
$n_{Bi210}(0)$	$0\ cm^{-3}$
$n_{Po210}(0)$	$0\ cm^{-3}$

** These values are given in the reference paper*

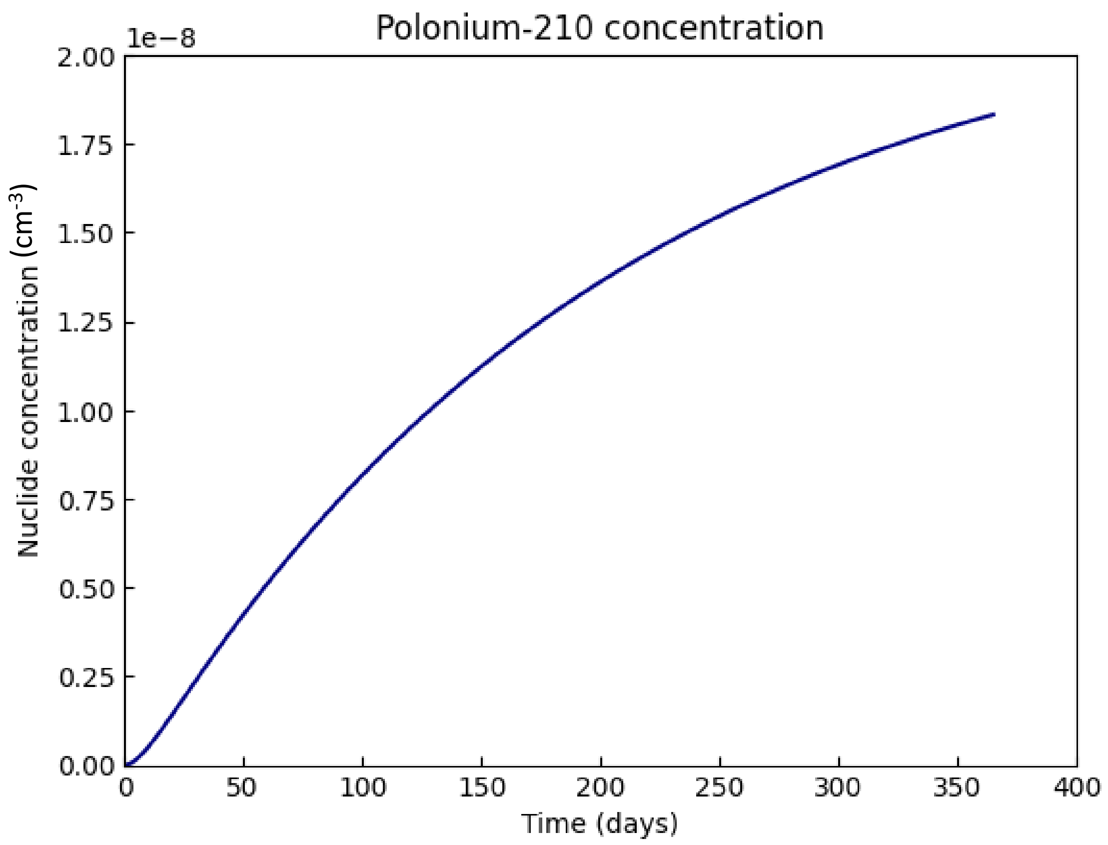
The Polonium Problem: Results



The Polonium Problem: Results



The Polonium Problem: Results



Conclusions

The resulting plots show the variation in the nuclide concentration over a period of 365 days of the different nuclides.

The results are compatible with what is expected to happen: Bismuth-209 decreases over time while Bismuth-210 and Polonium-210 both increases starting from an initial concentration of 0.

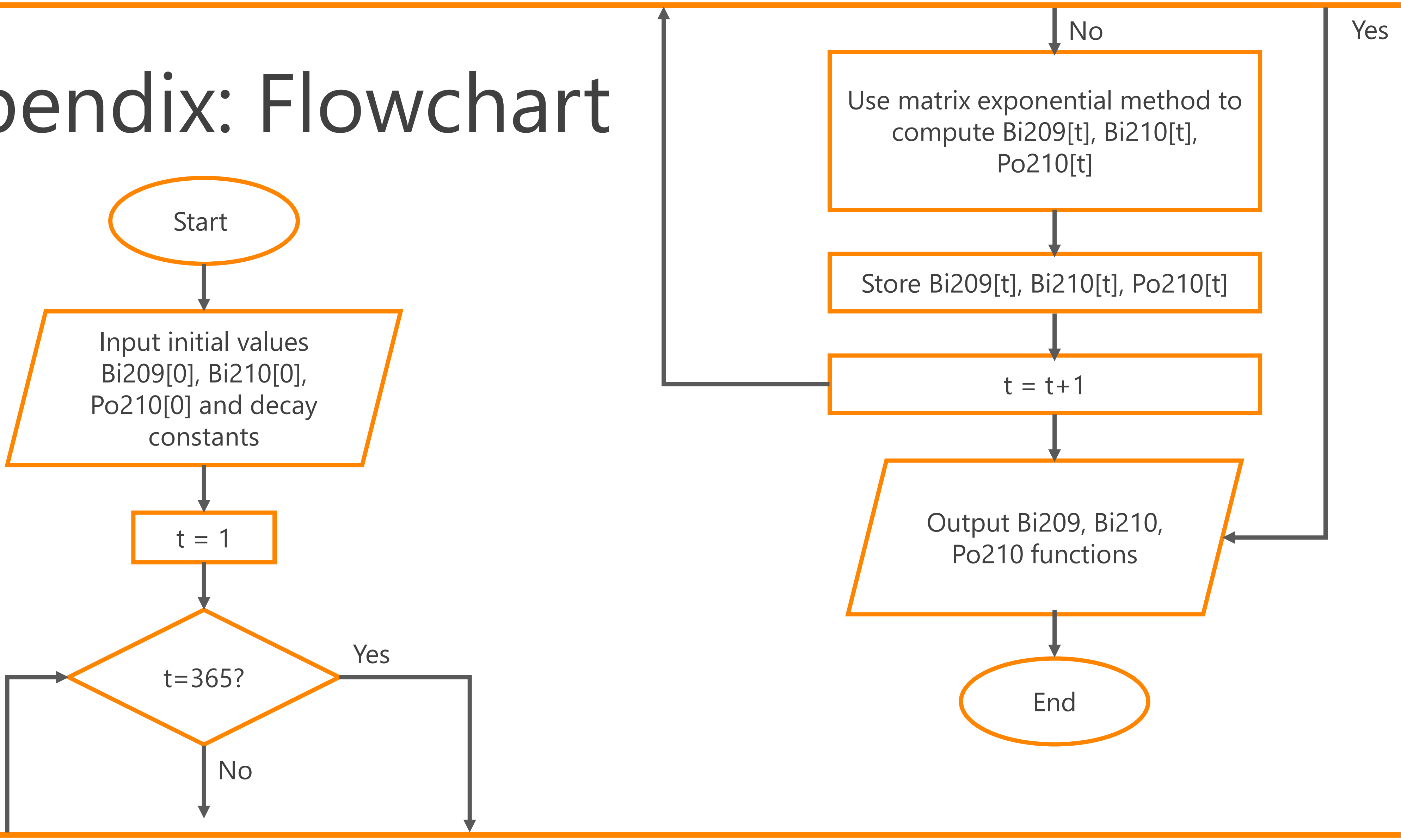
In particular, it seems that Bismuth-210 reaches equilibrium after ~93 days. When a closer look is given at the results, it emerges that the concentration of Bi-210 reaches a maximum of 7.96453e-10 cm^{-3}, which stays the same for some days, and then starts slowly decreasing again.

Bibliography

[1] Master Thesis: Implementation, validation and comparison of different algorithms to solve the Bateman equations for very large systems – Vranckx Maren 2015/16 (p. 4-6)

[2] Awad H. Al-Mohy and Nicholas J. Higham, (2009), "A New Scaling and Squaring Algorithm for the Matrix Exponential"

Appendix: Flowchart



Appendix: Python code

```python
def polonium_problem():

    # Decay constants of the nuclides
    lambdaBi_209 = 1.83163e-12
    lambdaBi_210 = 1.60035e-6
    lambdaPo_210 = 5.79764e-8

    # Time unit: 1 day
    T = 365 # Simulate for one year (365 days)
    dt = 60*60*24 # seconds in a day

    # Initial concentration of the different nuclides
    Bi209_0 = 6.95896e-4
    Bi210_0 = 0
    Po210_0 = 0
    init_value = [Bi209_0, Bi210_0, Po210_0]

    # Bateman Matrix
    A = np.array([[-lambdaBi_209, 0, 0],
                  [lambdaBi_209, -lambdaBi_210, 0],
                  [0, lambdaBi_210, -lambdaPo_210]])

    # Compute the solution to the Bateman's system
    bateman_sol = matrix_exponential_method(init_value,dt,T,A)
```

```python
    Bi209 = bateman_sol[:,0]
    Bi210 = bateman_sol[:,1]
    Po210 = bateman_sol[:,2]

    #Print results
    print_results(T, Bi209, Bi210, Po210, 't', 'Bismuth-209', 'Bismuth-210', 'Polonium-210')

    # Plots
    xmin = 0
    xmax = 400
    xlabel = 'Time (days)'
    ylabel = 'Nuclide concentration'

    plot(0,69580.0e-8,69600.0e-8, Bi209, xmin, xmax, xlabel, ylabel,'Bismuth-209 concentration')
    plot(1,0,8.5e-10,Bi210, xmin, xmax, xlabel, ylabel,'Bismuth-210 concentration')
    plot(2,0,2e-8,Po210, xmin, xmax, xlabel, ylabel,'Polonium-210 concentration')

    plt.show()

if __name__ == '__main__':
    polonium_problem()
```

Python implementation of the Polonium problem

Appendix: Python code

```
def matrix_exponential_method(U_0,dt,T,A):
    '''...

    # Definition of u matrix
    u = np.zeros((T+1, len(U_0)))

    # Fill u with initial values
    u[0] = U_0

    # Iteration
    for i in range(T):
        u[i+1] = linalg.expm((dt*A)).dot(u[i])

    return u
```

Python implementation of the matrix exponential method

initial values of ODE system's functions

Square matrix containing the coefficients of the ODE system

number of steps

number of points in each step

Returns a matrix in which each column contains the points solution to a different function of the ODE system

Appendix: Python code

```python
def print_results(N, array1, array2, array3, head1, head2, head3, head4):
    '''...
    table = []

    for i in range(N):
        table.append([i, array1[i], array2[i], array3[i]])

    print("Results")
    print(" ")
    print(tabulate(table, headers=[head1, head2, head3, head4]))
    print(" ")
```

This function creates a table with the results

```python
def plot(i,y_min,y_max,y,xmin,xmax,xlab,ylab,title):
    '''...

    plt.figure(i)
    plt.ylim(y_min,y_max)
    plt.xlim(xmin, xmax)
    plt.plot(y, '#000080')
    plt.title(title)
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    plt.ticklabel_format(axis = "y", style = "sci", scilimits=(0,0))
    plt.tick_params('both', direction = 'in')
```

This function plots the solutions