



# 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:

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

*n(t): nuclide concentration vector*

*A: Bateman matrix*

*n<sub>0</sub>: initial nuclide concentration*

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

# 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 <sup>[1]</sup>

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:



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

$$\begin{aligned}\frac{dn_{\text{Bi}209}}{dt} &= -\lambda_{(\text{Bi}209)}n_{\text{Bi}209} \\ \frac{dn_{\text{Bi}210}}{dt} &= \lambda_{(\text{Bi}209)}n_{\text{Bi}209} - \lambda_{(\text{Bi}210)}n_{\text{Bi}210} \\ \frac{dn_{\text{Po}210}}{dt} &= \lambda_{(\text{Bi}210)}n_{\text{Bi}210} - \lambda_{(\text{Po}210)}n_{\text{Po}210}\end{aligned}$$

$\lambda_{\text{nuclide}}$ : 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).

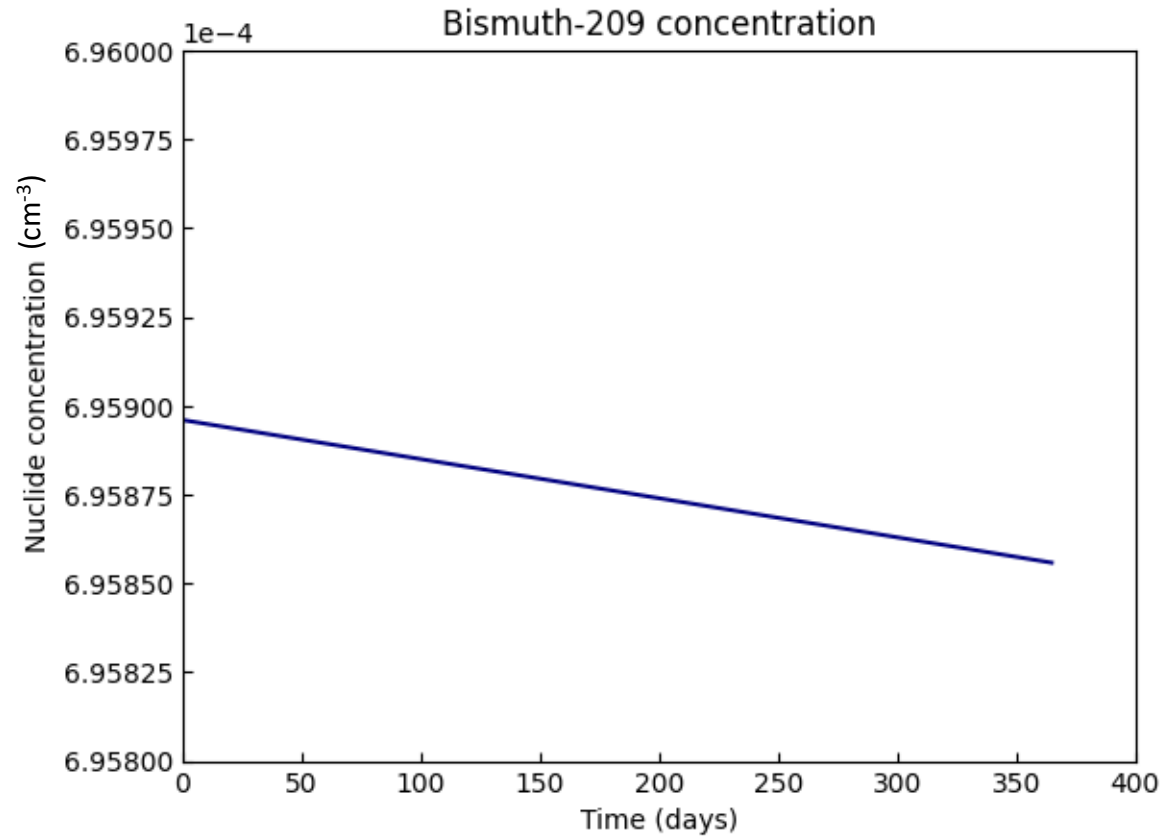
# The Polonium Problem

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

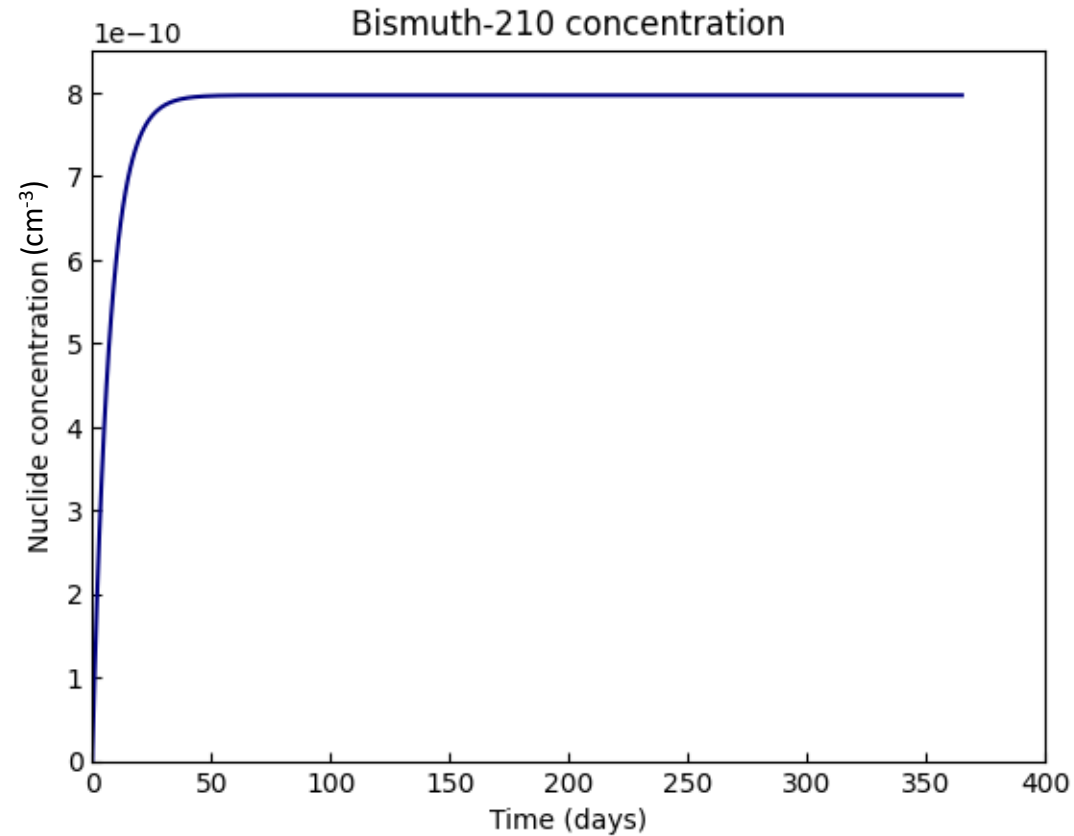
$\lambda_{\text{Bi209}}$	$1.83163 \cdot 10^{-12} \text{ s}^{-1}$
$\lambda_{\text{Bi210}}$	$1.60035 \cdot 10^{-6} \text{ s}^{-1}$
$\lambda_{\text{Po210}}$	$5.79764 \cdot 10^{-8} \text{ s}^{-1}$
$n_{\text{Bi209}}(0)$	$6.95896 \cdot 10^{-4} \text{ cm}^{-3}$
$n_{\text{Bi210}}(0)$	$0 \text{ cm}^{-3}$
$n_{\text{Po210}}(0)$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

\* *These values are given in the reference paper*

# The Polonium Problem: Results

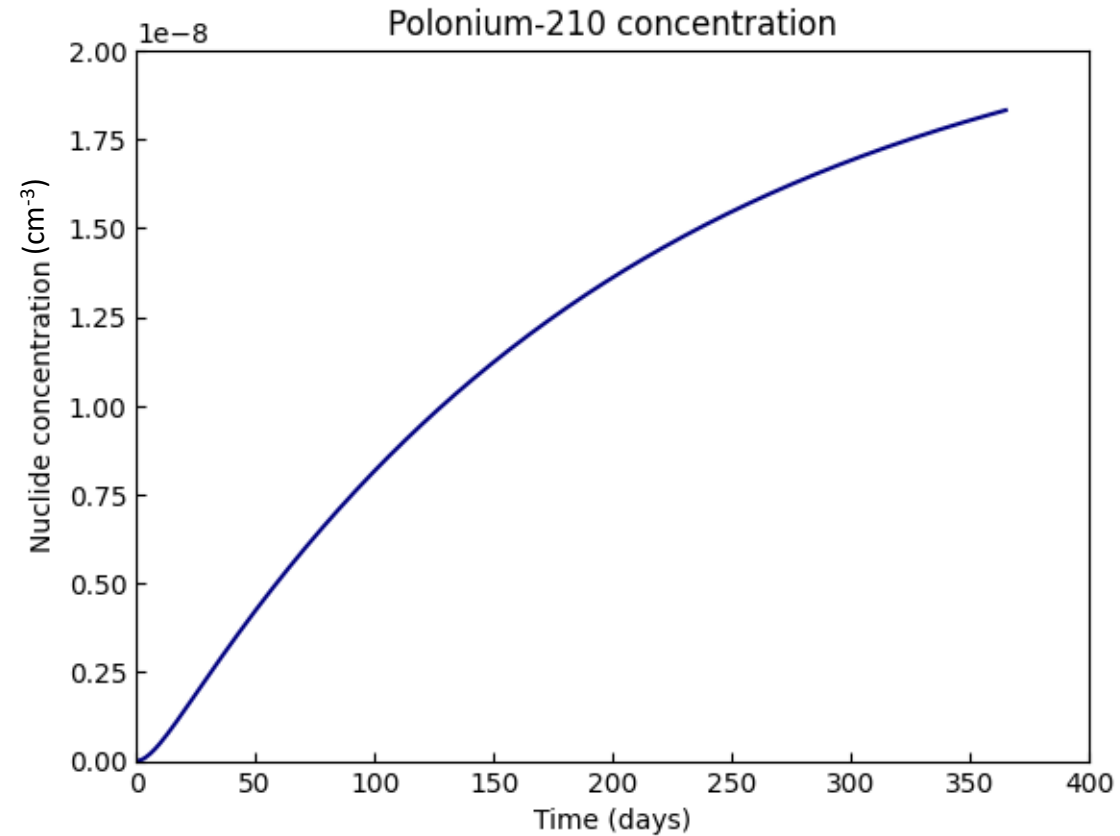


# The Polonium Problem: Results





# 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.

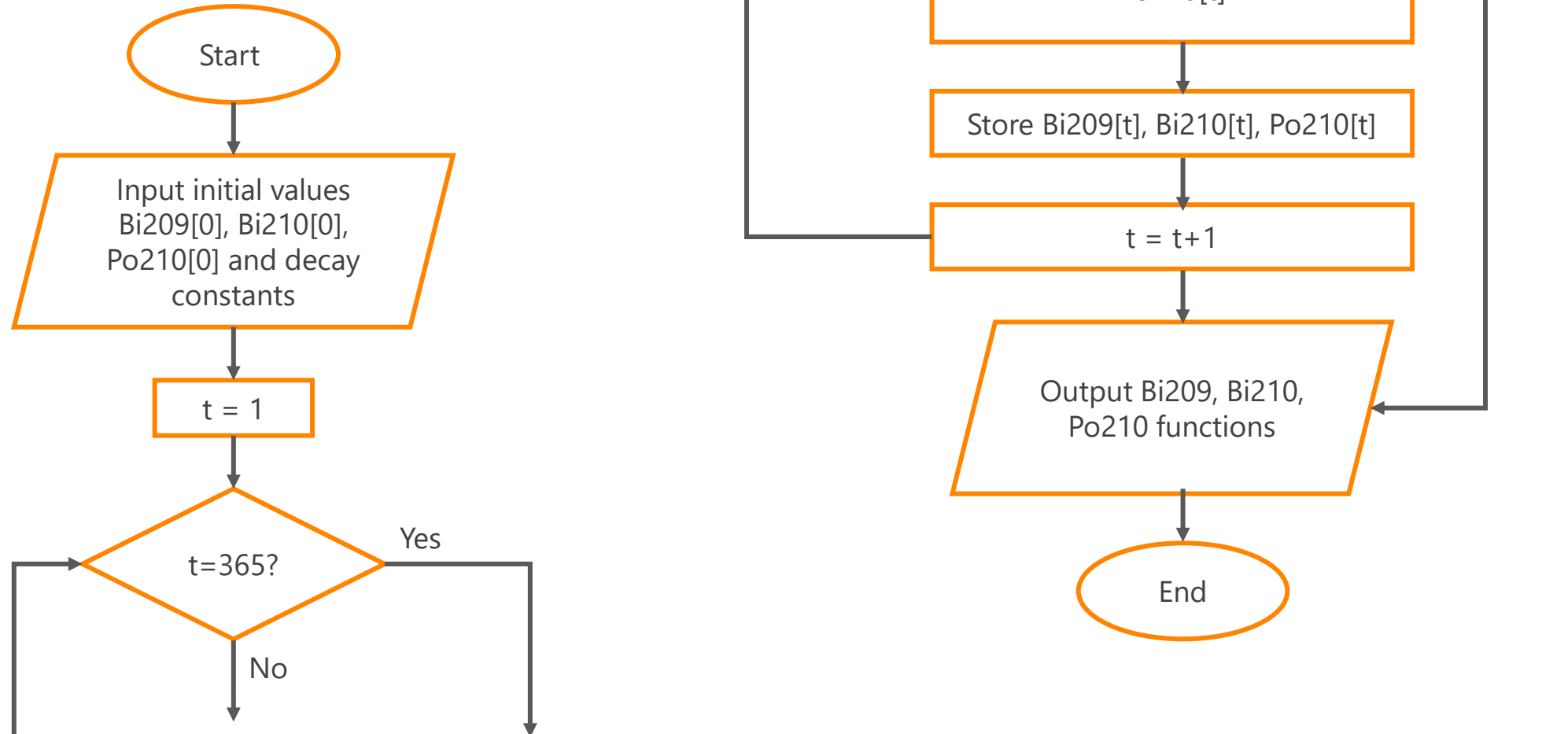
The results are compatible with what is expected to happen: Bismuth-209 decreases over time while Bismuth-210 and Polonium-210 both increases starting from an initial concentration of 0.

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  $7.96453\text{e-}10 \text{ cm}^{-3}$ , 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

```

98  def polonium_problem():
99
100     # Decay constants of the nuclides
101     lambdaBi_209 = 1.83163e-12
102     lambdaBi_210 = 1.60035e-6
103     lambdaPo_210 = 5.79764e-8
104
105     # Time unit: 1 day
106     T = 365 # Simulate for one year (365 days)
107     dt = 60*60*24 # seconds in a day
108
109     # Initial concentration of the different nuclides
110     Bi209_0 = 6.95896e-4
111     Bi210_0 = 0
112     Po210_0 = 0
113     init_value = [Bi209_0, Bi210_0, Po210_0]
114
115     # Bateman Matrix
116     A = np.array([[-lambdaBi_209, 0, 0],
117                  [lambdaBi_209, -lambdaBi_210, 0],
118                  [0, lambdaBi_210, -lambdaPo_210]])
119
120     # Compute the solution to the Bateman's system
121     bateman_sol = matrix_exponential_method(init_value,dt,T,A)

```

```

123     Bi209 = bateman_sol[:,0]
124     Bi210 = bateman_sol[:,1]
125     Po210 = bateman_sol[:,2]
126
127     #Print results
128     print_results(T, Bi209, Bi210, Po210, 't', 'Bismuth-209', 'Bismuth-210', 'Polonium-210')
129
130     # Plots
131     xmin = 0
132     xmax = 400
133     xlabel = 'Time (days)'
134     ylabel = 'Nuclide concentration'
135
136     plot(0,69580.0e-8,69600.0e-8, Bi209, xmin, xmax, xlabel, ylabel,'Bismuth-209 concentration')
137     plot(1,0,8.5e-10,Bi210, xmin, xmax, xlabel, ylabel,'Bismuth-210 concentration')
138     plot(2,0,2e-8,Po210, xmin, xmax, xlabel, ylabel,'Polonium-210 concentration')
139
140     plt.show()
141
142     if __name__ == '__main__':
143         polonium_problem()

```

Python implementation of the Polonium problem

# Appendix: Python code

```
9  def matrix_exponential_method(U_0,dt,T,A):
10 >  ''' ...
21
22  # Definition of u matrix
23  u = np.zeros((T+1, len(U_0)))
24
25  # Fill u with initial values
26  u[0] = U_0
27
28  # Iteration
29  for i in range(T):
30      u[i+1] = linalg.expm((dt*A)).dot(u[i])
31
32  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

```
37 def print_results(N, array1, array2, array3, head1, head2, head3, head4):
38 >     '''...
55     table = []
56
57     for i in range(N):
58         table.append([i, array1[i], array2[i], array3[i]])
59
60     print("Results")
61     print(" ")
62     print(tabulate(table, headers=[head1, head2, head3, head4]))
63     print(" ")
```

This function creates a table with the results

```
65 def plot(i,y_min,y_max,y,xmin,xmax,xlab,ylab,title):
66 >     '''...
85
86     plt.figure(i)
87     plt.ylim(y_min,y_max)
88     plt.xlim(xmin, xmax)
89     plt.plot(y, '#000080')
90     plt.title(title)
91     plt.xlabel(xlab)
92     plt.ylabel(ylab)
93     plt.ticklabel_format(axis = "y", style = "sci", scilimits=(0,0))
94     plt.tick_params('both', direction = 'in')
```

This function plots the solutions