Half-life of alpha decay

FK8029 - Computational Physics

Andreas Evensen

Department of Physics
Stockholm University
Sweden
April 3, 2024

Contents

1 Introduction

In the early 20th-centary, people started to investigate the radioactive decay of elements. The decay of an element is a random process, and there exists more than one type of decay, such as: alpha-, beta-, and gamma-decay. The alpha decay is a process when the nuclei emit an alpha particle, which is a helium nucleus. Scientists found a model, which describes the half-life of the nuclei, which is dependent on the kinetic energy of the outgoing alpha particle using quantum mechanics. In this report, one will investigate the model, and compare the result to the theory and tabulated data to see if the model is consistent.

2 Theory & Method

One can think of an alpha decay as a tunneling process; an alpha particle has to traverse a potential boundary which is higher than the kinetic energy of the particle. This region is called the classically forbidden region. Hence, we can view the alpha decay as a quantum mechanical process, where the alpha particle has to tunnel through the potential barrier. We can describe the process of alpha decay using the Schrödinger equation:

$$i\hbar\frac{\partial\psi}{\partial t} + V(r)\psi = H\psi. \tag{1}$$

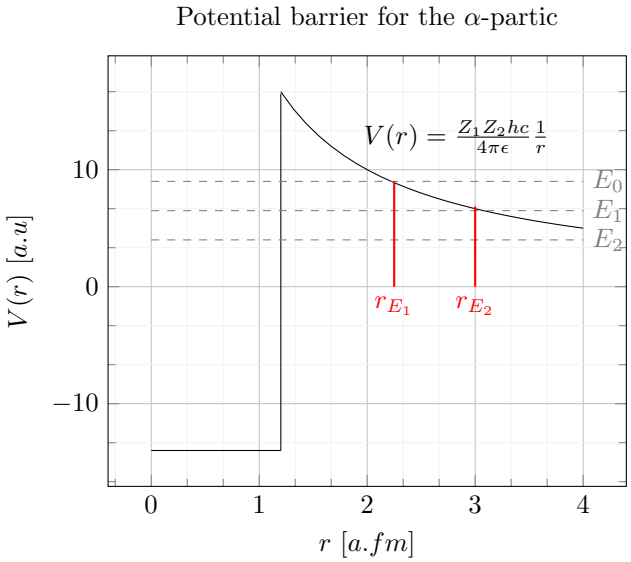


Figure 1: Potential barrier for alpha decay.

In the above figure, 1, we see the potential barrier for alpha decay. The potential barrier, $V(r)$ (the Columb repulsion) is discretized into n regions, where we assume that the potential is constant across those regions. This leads to the following equations:

$$\begin{aligned}\psi_i(r) &= A_i e^{k_i r} + B_i e^{-k_i r}, \quad r \in (r_i, r_{i+1}], \quad i = 1, 2, ..., n-1, \\ \psi_n(r) &= A_n e^{k_n r}, \quad r \in (r_{n-1}, \infty),\end{aligned} \tag{2}$$

The last wave function, ψ_n, is just the outgoing wave, since we assume no barrier for $r > r_{n-1}$. In the definition above, we defined the wave-number, k_i to be the following:

$$k_i = \frac{\sqrt{2mc^2(E - V_i)}}{\hbar c}, \quad i = 1, 2, ..., n, \tag{3}$$

where V_i is the potential in the i-th region. For the wave-function to be well-defined for this problem, the wave-functions at the interfaces must be equal, and also there derivatives, $\psi_i = \psi_{i+1}$ at r_i and $\psi_i' = \psi_{i+1}'$ at r_i. Solving the above equations, we get the following matrix equation:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ e^{ik_1 r_1} & e^{-ik_1 r_1} & -e^{ik_2 r_1} & -e^{-ik_2 r_1} & \dots & 0 & 0 & 0 \\ ik_1 e^{ik_1 r_1} & -ik_1 e^{-ik_1 r_1} & -ik_2 e^{ik_2 r_1} & ik_2 e^{-ik_2 r_1} & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \end{pmatrix} \cdot \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \tag{4}$$

where $\mathbf{x}$ is a vector containing the coefficients A_i and B_i. This matrix equation can be solved using various numerical techniques, such as: LU factorization, incomplete Chavosky factorization, or Gaussian elimination.

The unity equation is thus composed by the reflection and transmission probabilities, and in our system that is reflected as:

$$1 = \left|\frac{B_1}{A_1}\right|^2 + \left|\frac{A_n}{A_1}\right|^2 \frac{k_n}{k_1}, \tag{5}$$

where A, B are the elements in our unknown in equation (4). The k's are then the wave-functions wave-number. Using this, one derives the half life of the system as the oscillation frequency of the wave-function, which takes into account the masses of the system, and the kinetic energy of the outgoing α-particle:

$$t_{1/2} = \ln(2)\frac{T \cdot v}{2R}, \tag{6}$$

where T is the transmission probability, v is the effective velocity of the system, and R is the radii of the nuclei[1]. Below is a figure to visualize our discretization of the system:

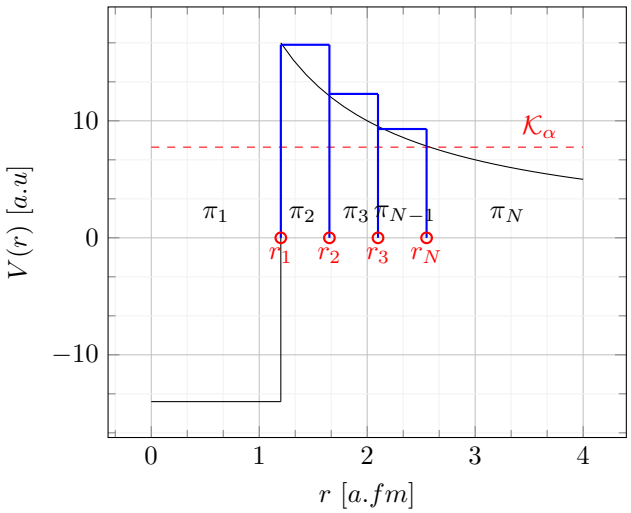


Figure 2: Visualization of the discretization process.

The above figure, 2, shows the discretization of the potential barrier, where the potential is constant in each region. Here, the radii r_1 is the size of the daughter nuclei, and r_N is the length of which the potential barrier is greater than the kinetic energy. The N regions all then have constant potential, and in region π_1, and π_N, the wave function is defined to oscillate.

3 Implementation & Results

The above theory was implemented in a C++ program4, where the matrix equation (4) was solved. Using the coefficient A_0 and A_n, we computed the transmission probabilities for various nuclei, which in turn derived the half-life as per equation (6) as shown in the figure below.

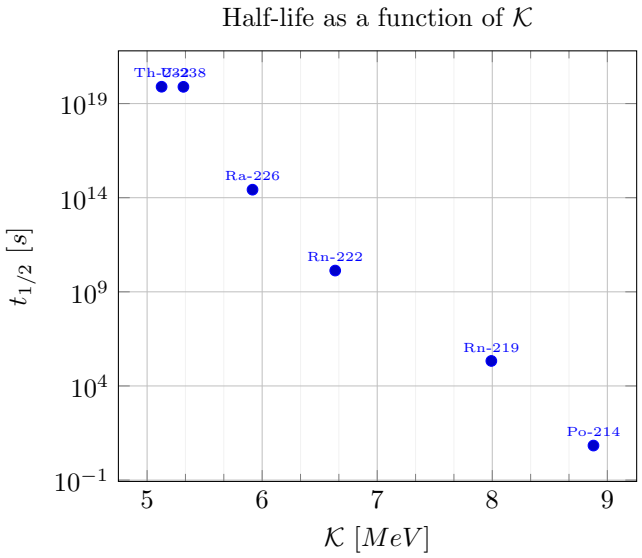


Figure 3: Half life as a function of $\mathcal{K}$ for various nuclei with $r_0 = 1.4$ fm and 100 number of bins.

The above figure exhibits the half-life difference between different nuclei depending on the kinetic energy of the outgoing α-particle. With small differences in the kinetic energy, the half-life can differ by several orders of magnitude, as visualized in the above figure. The results are not consistent with tabulated data, which is shown in the table below 5. However, the results are consistent with the theory, as the half-life is dependent on the kinetic energy of the outgoing α-particle.

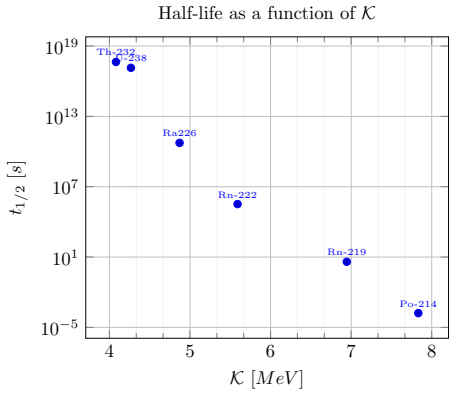
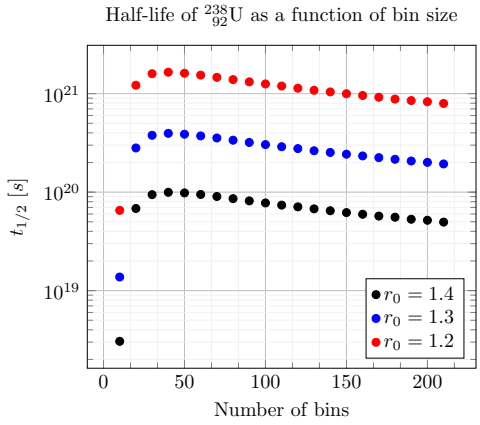


Figure 4: Half-life as a function of $\mathcal{K}$ for various nuclei for tabulated values[3].

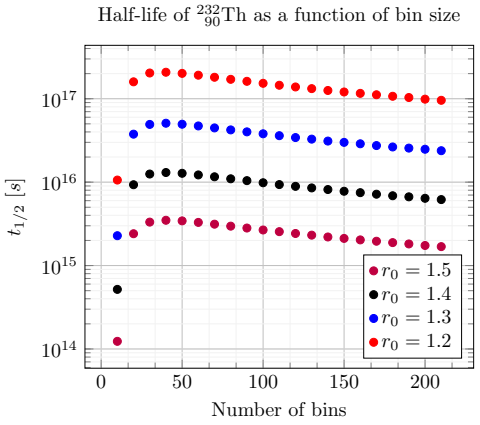
Nuclei	Half-life [s]
$^{238}_{92}$U	$1.4 \cdot 10^{17}$
$^{232}_{90}$Th	$4.4 \cdot 10^{17}$
$^{226}_{88}$Ra	$5.5 \cdot 10^{10}$
$^{222}_{86}$Rn	$3.3 \cdot 10^{5}$
$^{219}_{86}$Rn	3.96
$^{214}_{82}$Po	$164 \cdot 10^{-6}$

Figure 5: Half-life for various isotopes from tabulated data[3].

In the calculations, we had two free parameters, the number of bins, and the scaling of the radii, r_0. Varying the radii scaling, r_0, and the number of bins, we found that the half-life was very sensitive to the radii scaling, as shown in the figure below.



(a) Half life of $^{238}_{92}$U with varying bin size.

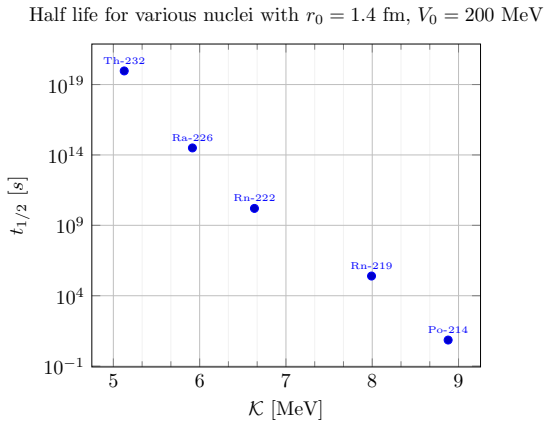


(b) Half life of $^{232}_{90}$Th with varying bin size.

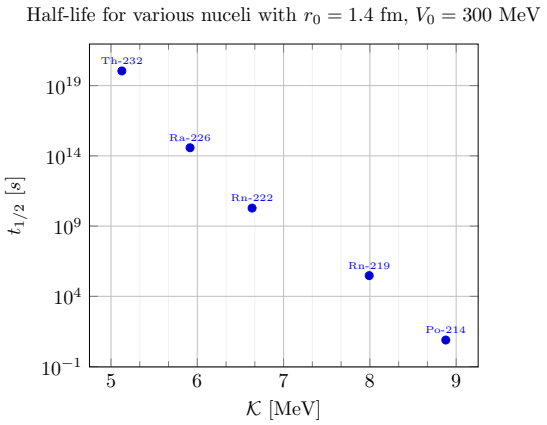
Figure 6: Half life of $^{238}_{92}$U and $^{232}_{90}$Th with varying bin size.

From the above figure, it can be seen that the half-life is very sensitive to the radii scaling r_0. This is expected, as the radii scaling determines the width of classically forbidden region, and thus affects the transmission probability. In both figures 6a and 6b, the half-life increases as the bin width decreases until it peaks and then converges. This indicates that the bin size has to be below a certain threshold to accurately describe the potential barrier, and thus the system.

The potential depth V_0 was also varied, and it was found that it had little impact on the half-life, as shown in the figure below.



(a) Half life of various nuclei with $V_0 = 200$ MeV.



(b) Half life of various nuclei with $V_0 = 300$ MeV.

Figure 7: Half life for varying potential depth.

Hence, as the potential depth V_0 has little impact on the half-life, we can state that the leading parameters are the radii, the kinetic energy and the potential itself.

4 Conclusion

In this report, a simple model for describing the half life of nuclei which decays via alpha decay was implemented. The results were consistent with the theory, as the half-life was dependent on the kinetic energy of the outgoing α-particle; however, the results were not consistent with tabulated data. There exists multiple reasons for this, such as the discretization of the potential, the initial radii scaling and an incomplete model. In this model, we do not take into account the spin of the nuclei, the angular momentum, or the parity of the system, which all affects the half-life.

In order to further improve the model, one could take into account the factors mentioned above, such as: angular momentum and parity. This would lead to a more accurate description of the system, and thus as a result, a more accurate model. When solving the matrix equation, partial LU factorization was used[2], which is a very efficient method for solving the matrix equation; however, different solver techniques were used, and the results differed significantly, which could be further investigated. Furthermore, the model could be improved by instead of using a constant potential in each region, one could use a trapezoid method to describe the potential within the regions more accurately.

References

[1]K. S. Krane, *Introductory nuclear physics* (John Wiley & Sons, 1991).

[2]G. Guennebaud, B. Jacob, et al., *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[3]NIST, *Atomic Weights and Isotopic Compositions with Relative Atomic Masses*, 2021.

Appendix

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <complex>
#include <cmath>
#include <fstream>
#include "Eigen/Dense"
#include "Eigen/Sparse"

const double hbar_c = 197.326; // MeV fm
const double alpha = 1.0 / 137.035399;
const int protonHelium = 2; // Two protons
const double speedOfLight = 299792458; // m/s
const long double c_squared = std::pow(speedOfLight * std::pow(10, 15), 2); // fm^2 / s^2
const double massOfHelium = 3727.379; // MeV
const double amuToMeV = 931.5; // MeV

/**
 * Prints a string to the console
 * @param[in] s the string to print
*/
void println(std::string s) {
    std::cout << s << std::endl;
}

/**
 * Preferences for the decay
 * @param[in] numberOfBins the number of bins used for the discretisation
 * @param[in] protonNumber the number of protons in the nuclei
 * @param[in] neutronNumber the number of neutrons in the nuclei
 * @param[in] massParent the mass of the parent nuclei in amu
 * @param[in] massDaughter the mass of the daughter nuclei in amu
 * @param[in] name the name of the decay
 * @param[in] r0 the scaling factor for the radii
*/
struct Preferences {
    int numberOfBins;
    int protonNumber;
    int neutronNumber;
    long double massParent; // MeV
    long double massDaughter; // MeV
    double r0;
    float potential = 134;
    std::string name;

    Preferences(int numberOfBins, int protonNumber, int neutronNumber, double massParent, double massDaughter, std::string name, double r0) {
        this->numberOfBins = numberOfBins;
        this->protonNumber = protonNumber;
        this->neutronNumber = neutronNumber;
        this->massParent = massParent * amuToMeV;
        this->massDaughter = massDaughter * amuToMeV;
        this->name = name;
        this->r0 = r0;
    };
```

```
};

struct Buffer {
    double long kineticEnergy; // MeV
    double long halfLife; // s
    std::string name;
};

/**
 * Computes the distance of at which the potential energy is equal to the kinetic energy
 * @param[in] Z the number of protons in the nuclei
 * @param[in] kineticEnergy the kinetic energy of the alpha particle
 * @returns the distance of at which the potential energy is equal to the kinetic energy
*/
long double findBoundary(int Z, long double kineticEnergy) {
    // V = K
    // V = alpha / r * hbar_c * helium * Z = K
    // alpha / energy * hbar_c * helium * Z = r

    return alpha / kineticEnergy * hbar_c * protonHelium* (Z);
}

/**
 * Computes the potential energy at a given distance r from the nuclei
 * @param[in] r the distance from the nuclei
 * @param[in] Z the number of protons in the nuclei
 * @param[in] radii the radii of the nuclei
 * @param[in] maximumDistance where the potential energy is equal to the kinetic energy
 * @returns the potential energy at a given distance r from the nuclei
*/
long double potentialEnergy(long double r, int Z, float radii, long double maximumDistance, float V0) {
    if (r < radii) {
        return -V0 + alpha / r * hbar_c * protonHelium * Z;
    } else if (r > maximumDistance) {
        return 0;
    } else {
        return alpha / r * hbar_c * protonHelium * Z;
    }
}

/**
 * Computes the wave number k for a given energy and potential energy
 * @param[in] energy the kinetic energy of the alpha particle
 * @param[in] potentialEnergy the potential energy at a given distance r from the nuclei
 * @returns the wave number k which might be complex
*/
std::complex<long double> computeK(long double energy, long double potentialEnergy) {
    if (energy < potentialEnergy) {
        return std::complex<long double>(std::sqrt(2 * massOfHelium * (potentialEnergy - energy)) / hbar_c, 0.0);
    }
    return std::complex<long double> (0.0, std::sqrt(2 * massOfHelium * (energy - potentialEnergy)) / hbar_c);
}

/**
 * Computes the energy of the alpha particle
 * @param[in] pref the preferences for the decay, e.g. number of bins, proton number, neutron number, mass of parent, mass of daughter, name
 * @returns the energy of the alpha particle
*/
```

```
113 long double computeAlphaEnergy(Preferences pref) {
114     return pref.massParent - (pref.massDaughter + massOfHelium);
115 }
116 
117 /**
118  * Computes the matrix for the given potential and energy
119  *
120  * @param[in] pref the preferences for the decay, e.g. number of bins, proton
     number, neutron number, mass of parent, mass of daughter, name
121  * @returns Buffer the buffer containing the half-life, kinetic energy and name of
     the decay
122 */
123 Buffer solve(Preferences pref) {
124 
125     int size = 2 * (pref.numberOfBins - 1) + 1;
126 
127     // We compute the energy of the alpha particle
128     long double energy = computeAlphaEnergy(pref); // MeV
129 
130     int massNumber = pref.protonNumber + pref.neutronNumber;
131 
132     long double minimumDistance = pref.r0 * std::pow(massNumber - 4, 1.0 / 3.0); //
     massNumber - 4 is the massNumber of the daughter nuclei
133 
134     long double maximumDistance = findBoundary(pref.protonNumber - 2, energy);
135 
136     long double stepSize = ( maximumDistance - minimumDistance ) / ( pref.
    numberOfBins - 2 );
137 
138     long double boundaries [2 * pref.numberOfBins - 2];
139     double offset = stepSize * 0.01;
140 
141     long double b;
142     for (int i = 0; i < pref.numberOfBins - 1; i++) {
143         b = minimumDistance + i * stepSize;
144         boundaries[2 * i] = b - offset;
145         boundaries[2 * i + 1] = b + offset;
146     }
147     boundaries[pref.numberOfBins - 1] = maximumDistance;
148 
149     Eigen::MatrixXcf m = Eigen::MatrixXcf::Zero(size, size);
150     m(0,0) = 1;
151 
152     // We have the size 2N - 1, where N is the number of bins
153     // We can fill from m(1,0) -> m(2*N - 1, 2*N - 2)
154 
155     for (int idx = 0; idx < 2 * pref.numberOfBins - 2; idx += 2) {
156         long double boundary = boundaries[idx] + offset;
157 
158         std::complex<long double> kLeft, kRight;
159 
160         kLeft = computeK(energy, potentialEnergy(boundaries[idx], pref.protonNumber
     - 2, minimumDistance, maximumDistance, pref.potential));
161         kRight = computeK(energy, potentialEnergy(boundaries[idx + 1], pref.
    protonNumber - 2, minimumDistance, maximumDistance, pref.potential));
162 
163         // Now we fill the matrix elements
164 
165         // \phi_i A_i
166         m(idx + 1, idx) = std::exp(kLeft * (boundary));
167         m(idx + 2, idx) = kLeft * std::exp(kLeft * (boundary));
168 
169         // \phi_i B_i
170         m(idx + 1, idx + 1) = std::exp(-kLeft * (boundary));
171         m(idx + 2, idx + 1) = - kLeft * std::exp(-kLeft * (boundary));
172 
173         // \phi_{i+1} A_{i+1}
```

```
        m(idx + 1, idx + 2) = - std::exp(kRight * (boundary));
        m(idx + 2, idx + 2) = - kRight * std::exp(kRight * (boundary));

        if (idx < 2 * pref.numberOfBins - 4){
            // Only the last row is different, because the matrix is constructed such that;
            // phi_n = a_n e^{ik_nx*)} + 0 * b_n e^{-ik_nx}
            // \phi_{i+1} B_{i+1}
            m(idx + 1, idx + 3) = std::exp(-kRight * (boundary));
            m(idx + 2, idx + 3) = -kRight * std::exp(-kRight * (boundary));
        }
    }

    // Solving the system of equations and producing the half life
    Eigen::VectorXcf RHS = Eigen::VectorXcf::Zero(size);

    RHS(0) = 1.0; // The "initial" condition

    Eigen::VectorXcf coefficients;
    coefficients = m.partialPivLu().solve(RHS);

    std::complex<long double> rate;

    rate = computeK(energy, potentialEnergy(boundaries[2 * pref.numberOfBins - 3], pref.protonNumber - 2, minimumDistance, maximumDistance, pref.potential));
    rate /= computeK(energy, potentialEnergy(boundaries[0], pref.protonNumber - 2, minimumDistance, maximumDistance, pref.potential));

    long double transmission;

    transmission = std::pow(std::abs(coefficients[size - 1] / coefficients[0]), 2) * (rate.real());// + rate.imag());

    /*
        The half-life is given by the following formula
        T = ln(2) * 2 * R / (v_eff * Transmission)
        where v is the velocity of the alpha particle
    */
    double effectiveMass = massOfHelium * pref.massDaughter / (pref.massDaughter + massOfHelium);

    long double v_eff = std::sqrt(2 * energy / effectiveMass); // c

    v_eff = v_eff * speedOfLight; // m / s

    long double tau = 2 * (minimumDistance * std::pow(10, -15)) / ( transmission * v_eff); // s

    long double halfLife = std::log(2) * tau;

    Buffer buffer;
    buffer.halfLife = halfLife;
    buffer.kineticEnergy = energy;
    buffer.name = pref.name;

    return buffer;
}

int main() {
    println("Input whether you want to perform the calculations for the isotope chain (y/n):");
    std::string input;

    std::cin >> input;
    if (input == "y") {
        println("Want to change the initial potential y/n: ");
        std::cin >> input;
```

```
234         if (input == "n") {
235 
236             const float V0 = 134; // MeV
237             int numberOfBins = 100;
238 
239             double deltaR = 1.4;
240 
241             Preferences pref1 = Preferences(numberOfBins, 82, 132, 213.9952014,
    209.9841885, "Po-214", deltaR); // Po-214 -> Pb-210
242             Preferences pref2 = Preferences(numberOfBins, 86, 133, 219.00948,
    214.9994200, "Rn-219", deltaR); // Radon-219 -> Polonium-215
243 
244             Preferences pref3 = Preferences(numberOfBins, 92, 146, 238.050788,
    234.043601, "U-238", deltaR); // Uranium-238 -> Thorium-234
245             Preferences pref4 = Preferences(numberOfBins, 90, 142, 232.0380536,
    228.0310703, "Th-232", deltaR); // Thorium-232 -> Radium-228
246             Preferences pref5 = Preferences(numberOfBins, 88, 136, 226.025408,
    222.0175763, "Ra-226", deltaR); // Radium-226 -> Radon-222
247             Preferences pref6 = Preferences(numberOfBins, 86, 136, 222.0175763,
    218.0089730, "Rn-222", deltaR); // Radon-218 -> Polonium-214
248 
249             Buffer buffers [6];
250 
251             buffers[0] = solve(pref1);
252             buffers[1] = solve(pref2);
253             buffers[2] = solve(pref3);
254             buffers[3] = solve(pref4);
255             buffers[4] = solve(pref5);
256             buffers[5] = solve(pref6);
257 
258             std::ofstream file("data.dat");
259 
260             file << "K\t t\t name" << std::endl;
261 
262             for (int i = 0; i < 6; i++) {
263                 file << buffers[i].kineticEnergy <<  "\t" << buffers[i].halfLife <<
     "\t" << buffers[i].name << std::endl;
264             }
265 
266             file.close();
267         }
268         else {
269             println("Input the potential energy: ");
270             float potential;
271             std::cin >> potential;
272 
273             int numberOfBins = 100;
274 
275             double deltaR = 1.4;
276 
277             Preferences pref1 = Preferences(numberOfBins, 82, 132, 213.9952014,
    209.9841885, "Po-214", deltaR); // Po-214 -> Pb-210
278             pref1.potential = potential;
279             Preferences pref2 = Preferences(numberOfBins, 86, 133, 219.00948,
    214.9994200, "Rn-219", deltaR); // Radon-219 -> Polonium-215
280             pref2.potential = potential;
281 
282             //Preferences pref3 = Preferences(numberOfBins, 92, 146, 238.050788,
    234.043601, "U-238", deltaR); // Uranium-238 -> Thorium-234
283             Preferences pref4 = Preferences(numberOfBins, 90, 142, 232.0380536,
    228.0310703, "Th-232", deltaR); // Thorium-232 -> Radium-228
284             pref4.potential = potential;
285             Preferences pref5 = Preferences(numberOfBins, 88, 136, 226.025408,
    222.0175763, "Ra-226", deltaR); // Radium-226 -> Radon-222
286             pref5.potential = potential;
287             Preferences pref6 = Preferences(numberOfBins, 86, 136, 222.0175763,
    218.0089730, "Rn-222", deltaR); // Radon-218 -> Polonium-214
```

```
288             pref6.potential = potential;
289
290             Buffer buffers [5];
291
292             buffers[0] = solve(pref1);
293             buffers[1] = solve(pref2);
294             //buffers[2] = solve(pref3);
295             buffers[2] = solve(pref4);
296             buffers[3] = solve(pref5);
297             buffers[4] = solve(pref6);
298
299             std::string filename = "data";
300
301             filename.append("_");
302             filename.append(std::to_string(float(potential)));
303             filename.append(".dat");
304
305             std::ofstream file(filename);
306
307             file << "K\t t\t name" << std::endl;
308
309             for (int i = 0; i < 5; i++) {
310                 file << buffers[i].kineticEnergy <<  "\t" << buffers[i].halfLife <<
     "\t" << buffers[i].name << std::endl;
311             }
312
313             file.close();
314         }
315
316     } else {
317         println("Input the decay: ");
318         std::vector<Preferences> pref;
319         std::cin >> input;
320
321         println("Input the radii scaling: ");
322         double r0;
323         std::cin >> r0;
324
325         int maximumBins = 21;
326
327
328         if (input == "cf-250") {
329             for (int i = 1; i < maximumBins + 1; i++) {
330                 pref.push_back(Preferences(i * 10, 98, 152, 250.0764045,
    246.0672237, "Cf-250", r0));
331             }
332         } else if (input == "pu-242") {
333             for (int i = 1; i < maximumBins + 1; i++) {
334                 pref.push_back(Preferences(i * 10, 94, 148, 242.059, 238.05078826,
    "Pu-242", r0));
335             }
336         } else if (input == "u-238") {
337             for (int i = 1; i < maximumBins + 1; i++) {
338                 pref.push_back(Preferences(i * 10, 92, 146, 238.050788, 234.043601,
     "U-238", r0));
339             }
340         } else if (input == "th-232") {
341             for (int i = 1; i < maximumBins + 1; i++) {
342                 pref.push_back(Preferences(i * 10, 90, 142, 232.0380536,
    228.0310703, "Th-232", r0));
343             }
344         } else if (input == "ra-226") {
345             for (int i = 1; i < maximumBins + 1; i++) {
346                 pref.push_back(Preferences(i * 10, 88, 136, 226.025408,
    222.0175763, "Ra-226", r0));
347             }
348         } else if (input == "rn-222") {
```

```
            for (int i = 1; i < maximumBins + 1; i++) {
                pref.push_back(Preferences(i * 10, 86, 136, 222.0175763,
    218.0089730, "Rn-222", r0));
            }
        } else {
            println("Invalid input");
            throw std::invalid_argument("Invalid input");
            return 0;
        }

        Buffer b [pref.size()];

        for (int i = 0; i < pref.size(); i++) {
            Buffer buffer = solve(pref[i]);
            //println("Half-life for " + pref[i].name + " with " + std::to_string(
    pref[i].numberOfBins) + " bins: " + std::to_string(buffer.halfLife));
            b[i] = buffer;
        }

        std::string filename;

        filename = "data";

        filename.append("_");
        filename.append(input);
        filename.append("_");
        filename.append(std::to_string(float(r0)));
        filename.append(".dat");

        std::ofstream file(filename);

        file << "binSize\t t\tr0" << std::endl;

        for (int i = 0; i < pref.size(); i++) {
            file << ((i + 1) * 10) <<  "\t" << b[i].halfLife << "\t" << r0 << std::
    endl;
        }

        file.close();
    }

    return 0;
}
```