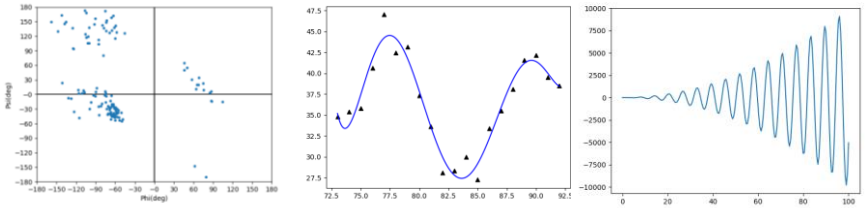
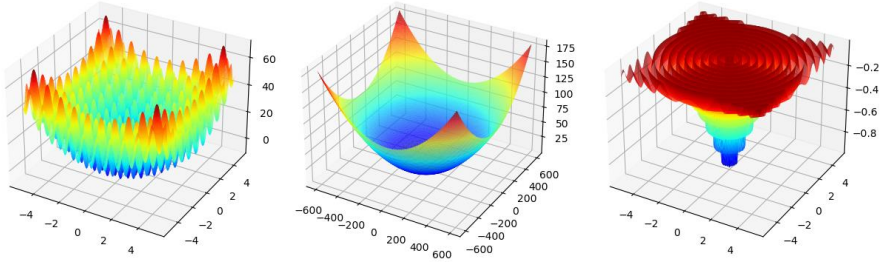
$$\frac{dv}{dt} = g - \frac{b}{m}v$$

$$\frac{dy}{y} = -kdt$$

$$\frac{dv}{dt} = a_0 + a_1 t + a_2 t^2$$





Introdução à Física Computacional

Aula 08

$$\frac{dx}{dt} = v_0 + at$$

$$\frac{dv}{dt} = a$$

$$\frac{du}{dt} = -k(u - T)$$

$$\frac{dy}{dt} = v_0 \sin\theta_0 - gt$$

A grande maioria dos sistemas físicos necessitam de equações diferenciais para a sua modelagem. Assim, a implementação em Python de programas que resolvem equações diferenciais é uma parte importante da Física Computacional. Iremos usar a biblioteca *SciPy* para a resolução de equações diferenciais. Especificamente o recurso *odeint()* da biblioteca *SciPy*. Para explicar a implementação desse recurso da biblioteca *SciPy*, iremos resolver uma equação diferencial linear de primeira ordem que modela o decaimento radioativo. Na explicação do código, destacaremos como implementar as resoluções de equações diferenciais similares à discutida aqui. Uma lista de projetos está listada no final da aula.

Considere o decaimento radioativo como um sistema a ser modelado. No decaimento radioativo, começamos da premissa de que a taxa de decaimento é proporcional à atividade de material radioativo presente na amostra. Isso leva à seguinte equação diferencial,

$$\frac{dN}{dt} = -kN$$

onde *N(>0)* é a atividade de substância radioativa presente no instante *t* e *k* é a constante de proporcionalidade. Para resolver essa equação diferencial, nós a reescrevemos da seguinte forma.

$$\frac{dN}{N} = -kdt$$

Integramos para obter a solução, como indicado abaixo:

$$\int \frac{dN}{N} = \int -kdt$$

$$ln(N) + C_1 = -kt + C_2 \rightarrow N = e^{C_2 - C_1} e^{-kt}$$

$$N = e^{C_2 - C_1} e^{-kt}$$

Assim temos:

$$N = Ce^{-kt}$$

onde *C* é a combinação de constantes de integração $e^{C_2 - C_1}$. A equação acima é chamada **solução geral** para a equação diferencial.

O valor de *C* é determinado se o valor inicial da atividade da substância radioativa for dado. Ou seja, resolvemos o **problema de valor inicial**. Então, temos uma expressão para a atividade de substância radioativa em qualquer instante *t*.

Para $t = 0$, temos $N = N_0$. Assim, a solução do problema de valor inicial tem a seguinte forma:

$$\mathrm{N} = N_0 e^{-kt}$$

Antes de vermos como resolver a equação diferencial em Python usando o *odeint()* da biblioteca *SciPy*, vamos usar a biblioteca *Matplotlib* para gerar o gráfico da solução da equação diferencial, a função *N(t)*.

$$N = N_0 e^{-kt}$$

Iremos gerar o gráfico da função acima no intervalo $0 \leq t \leq 20000$ *anos* com $k = 1{,}21285596.10^{-4}$ *ano*$^{-1}$ e $N_0 = 0{,}227$ *Bq*. Bq (becquerel) é a unidade de medida da atividade e representa uma desintegração (decaimento) por segundo.

Projeto	Equação Diferencial	Solução	Parâmetros Sugeridos
00	$\frac{dN}{dt} = -kN$	$N = N_0 e^{-kt}$	N_0 = *0,227 Bq;* *k = 1,21285596.10^{-4} ano^{-1}*

Projeto	Equação Diferencial	Sistema a Ser Modelado	Faixa de Valores para *t*
00	$\frac{dN}{dt} = -kN$	Decaimento radioativo (carbono-14).	*Entre 0 e 20000* *t = np.linspace(0,20000,200000)*

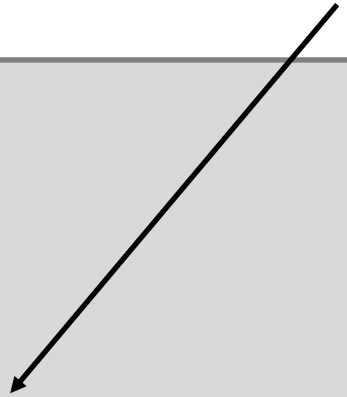
Como o código tem mais de 30 linhas, iremos mostrá-lo por partes. Começamos com a importação das bibliotecas *matplotlib.pyplot* e *NumPy.* Em seguida definimos a condição inicial *N0 = 0.227*. Depois definimos um *array* entre 0 e 20000 com 20000 números (anos) e atribuímos à variável *t*.

```
# Import section
import numpy as np
import matplotlib.pyplot as plt

# Initial condition
N0 = 0.227   # Initial value related to the dependent variable (N) (N0 = 0.227 Bq/g)

# Time points (years)
t = np.linspace(0,20000,20000)

# Define constant
k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
# Define plot parameters
title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
x_label_in = "t(year)"
y_label_in = "N(Bq)"
fontsize_title_in = 14
fontsize_x_label_in = 14
fontsize_y_label_in = 14
color_in = "blue"
```

Na linha seguinte, atribuímos *1.21285596e-04* à constante *k*. Nas próximas, linhas definimos parâmetros do gráfico a ser gerado com a biblioteca *Matplotlib*.

```
# Import section
import numpy as np
import matplotlib.pyplot as plt

# Initial condition
N0 = 0.227   # Initial value related to the dependent variable (N) (N0 = 0.227 Bq/g)

# Time points (years)
t = np.linspace(0,20000,20000)

# Define constant
k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
# Define plot parameters
title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
x_label_in = "t(year)"
y_label_in = "N(Bq)"
fontsize_title_in = 14
fontsize_x_label_in = 14
fontsize_y_label_in = 14
color_in = "blue"
```

Agora definimos a função $N = N_0 e^{-kt}$ que é a solução da equação diferencial da atividade da amostra. As últimas linhas geram o gráfico com os parâmetros definidos. Vejam como passamos a função $N = N_0 e^{-kt}$ para Python *N = N0*np.exp(-k*t)*

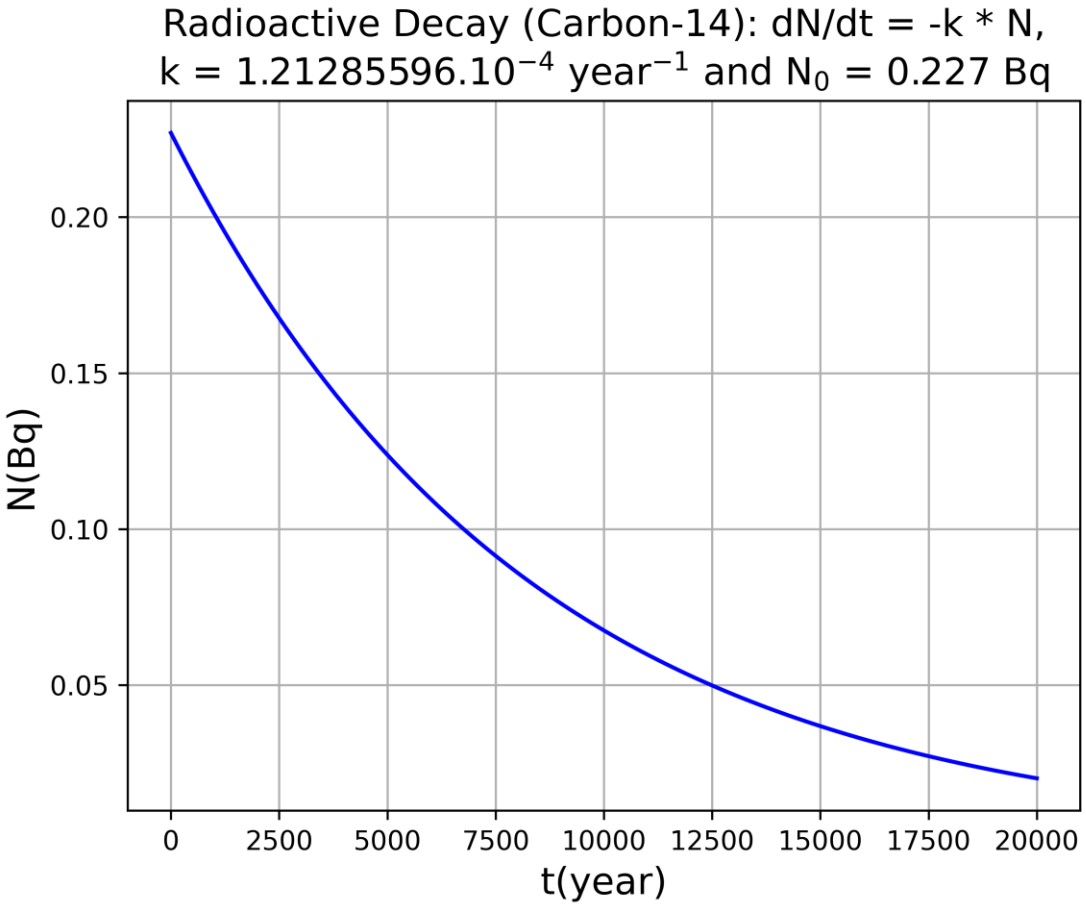


```
# Define function
N = N0*np.exp(-k*t)

# Plotting
plt.plot(t, N, c = color_in)
plt.title(title_in, fontsize = fontsize_title_in)
plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
plt.grid()
plt.show()
plt.savefig("radioactive_decay_01a.png",dpi=600)
```

Usamos o *np.exp()* para a função exponencial.

```
# Define function
N = N0*np.exp(-k*t)

# Plotting
plt.plot(t, N, c = color_in)
plt.title(title_in, fontsize = fontsize_title_in)
plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
plt.grid()
plt.show()
plt.savefig("radioactive_decay_01a.png",dpi=600)
```

Ao rodar o código, temos o gráfico mostrado abaixo. Destaco, não resolvemos a equação diferencial no programa *plot_radioactive_decay.py*. Simplesmente geramos o gráfico da solução da equação diferencial, que resolvemos de forma analítica nos slides anteriores.



Veremos a implementação da equação diferencial, com $k = 1{,}21285596.10^{-4}\ ano^{-1}$ e $N_0 = 0{,}227\ Bq$. Esses valores são para uma amostra de carbono-14 de 1 grama. Consideramos dados publicados para uma meia-vida do carbono-14 de 5715 anos (https://www.sciencedirect.com/topics/medicine-and-dentistry/carbon-14#).

$$\frac{dN}{dt} = -kN$$

Como o código anterior, este programa (*radioactive_decay.py*) tem mais de 30 linhas, iremos mostrá-lo por partes. Começamos o código com a importação das bibliotecas necessárias. Usaremos o *odeint()* da biblioteca *SciPy* para a solução da equação diferencial e o *plt* da biblioteca *matplotlib.pyplot* para gerarmos o gráfico.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define model() function
# function that returns dy/dt
def model(N,t,k):
    dNdt = -k * N
    return dNdt
# Define plotting() function
def plotting(t,y,title_in,x_label_in,y_label_in,fontsize_title_in,
             fontsize_x_label_in,fontsize_y_label_in,color_in):
    """Function to generate 2D plot"""
    # Plotting
    plt.plot(t, y, c = color_in)
    plt.title(title_in, fontsize = fontsize_title_in)
    plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
    plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
    plt.grid()
    plt.show()
    plt.savefig("radioactive_decay_01b.png",dpi=600)
```

Também importamos a biblioteca *NumPy* para gerarmos um *array* como eixo do tempo (*t = np.linspace(0,20000,20000)*). O uso da biblioteca *NumPy* ocorre mais à frente no código.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define model() function
# function that returns dy/dt
def model(N,t,k):
    dNdt = -k * N
    return dNdt
# Define plotting() function
def plotting(t,y,title_in,x_label_in,y_label_in,fontsize_title_in,
             fontsize_x_label_in,fontsize_y_label_in,color_in):
    """Function to generate 2D plot"""
    # Plotting
    plt.plot(t, y, c = color_in)
    plt.title(title_in, fontsize = fontsize_title_in)
    plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
    plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
    plt.grid()
    plt.show()
    plt.savefig("radioactive_decay_01b.png",dpi=600)
```

Na sequência, temos as definições de duas funções: *model()* e *plotting()*. Na função *model()*, calculamos *dNdt = -k*N* e retornamos seu valor. Os parâmetros da função *model(N,t,k)* são *N*, *t* e *k*. Essa função será chamada no programa principal (*main()*).

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define model() function
# function that returns dy/dt
def model(N,t,k):
    dNdt = -k * N
    return dNdt
# Define plotting() function
def plotting(t,y,title_in,x_label_in,y_label_in,fontsize_title_in,
             fontsize_x_label_in,fontsize_y_label_in,color_in):
    """Function to generate 2D plot"""
    # Plotting
    plt.plot(t, y, c = color_in)
    plt.title(title_in, fontsize = fontsize_title_in)
    plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
    plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
    plt.grid()
    plt.show()
    plt.savefig("radioactive_decay_01b.png",dpi=600)
```

Vejam como passamos a equação diferencial $\frac{dN}{dt} = -kN$ para Python *dNdt = -k * N*

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define model() function
# function that returns dy/dt
def model(N,t,k):
    dNdt = -k * N
    return dNdt
# Define plotting() function
def plotting(t,y,title_in,x_label_in,y_label_in,fontsize_title_in,
             fontsize_x_label_in,fontsize_y_label_in,color_in):
    """Function to generate 2D plot"""
    # Plotting
    plt.plot(t, y, c = color_in)
    plt.title(title_in, fontsize = fontsize_title_in)
    plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
    plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
    plt.grid()
    plt.show()
    plt.savefig("radioactive_decay_01b.png",dpi=600)
```

A função seguinte é a *plotting()*. Nela temos nove parâmetros, que são as entradas da função. Basicamente geramos o gráfico a partir desses parâmetros, como o título do gráfico definido pelo parâmetro *title_in*.

```
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Define model() function
# function that returns dy/dt
def model(N,t,k):
    dNdt = -k * N
    return dNdt
# Define plotting() function
def plotting(t,y,title_in,x_label_in,y_label_in,fontsize_title_in,
             fontsize_x_label_in,fontsize_y_label_in,color_in):
    """Function to generate 2D plot"""
    # Plotting
    plt.plot(t, y, c = color_in)
    plt.title(title_in, fontsize = fontsize_title_in)
    plt.xlabel(x_label_in,fontsize = fontsize_x_label_in)
    plt.ylabel(y_label_in,fontsize = fontsize_y_label_in)
    plt.grid()
    plt.show()
    plt.savefig("radioactive_decay_01b.png",dpi=600)
```

Agora temos a função *main()*, onde chamamos as duas funções já vistas. Mas antes atribuímos *0.227* à variável *N0*. Em seguida, geramos o array *t*, de forma similar à vista para o código anterior (*plot_radioactive_decay.py*).

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

Para resolver a equação diferencial, usamos *N = odeint(model,y0,t).* Nesta linha chamamos a função *model()* e resolvemos a equação diferencial implementada em *model()*. A partir dessa abordagem, podemos resolver outras equações diferenciais.

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

A solução numérica da equação diferencial implementada na função *model()* segue a seguinte abordagem. A função *model()* é a primeira a aparecer na chamada do *odeint()*. Seguem o valor inicial (*N0*) e o *array t*.

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

Na sequência vão os argumentos definidos em *args = (k,)*. A entrada de *args* é uma tupla, que só tem um elemento, a constante *k*. Vejam que quando temos somente um argumento, temos que deixar a vírgula, se fossem dois, ficaria *args=(k1,k2)*.

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

Para três argumentos da função, teríamos *args=(x1,x2,x3)* e assim para mais elementos. Veja que a vírgula no final é só para quando temos um elemento na tupla. As tuplas funcionam como listas imutáveis.

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

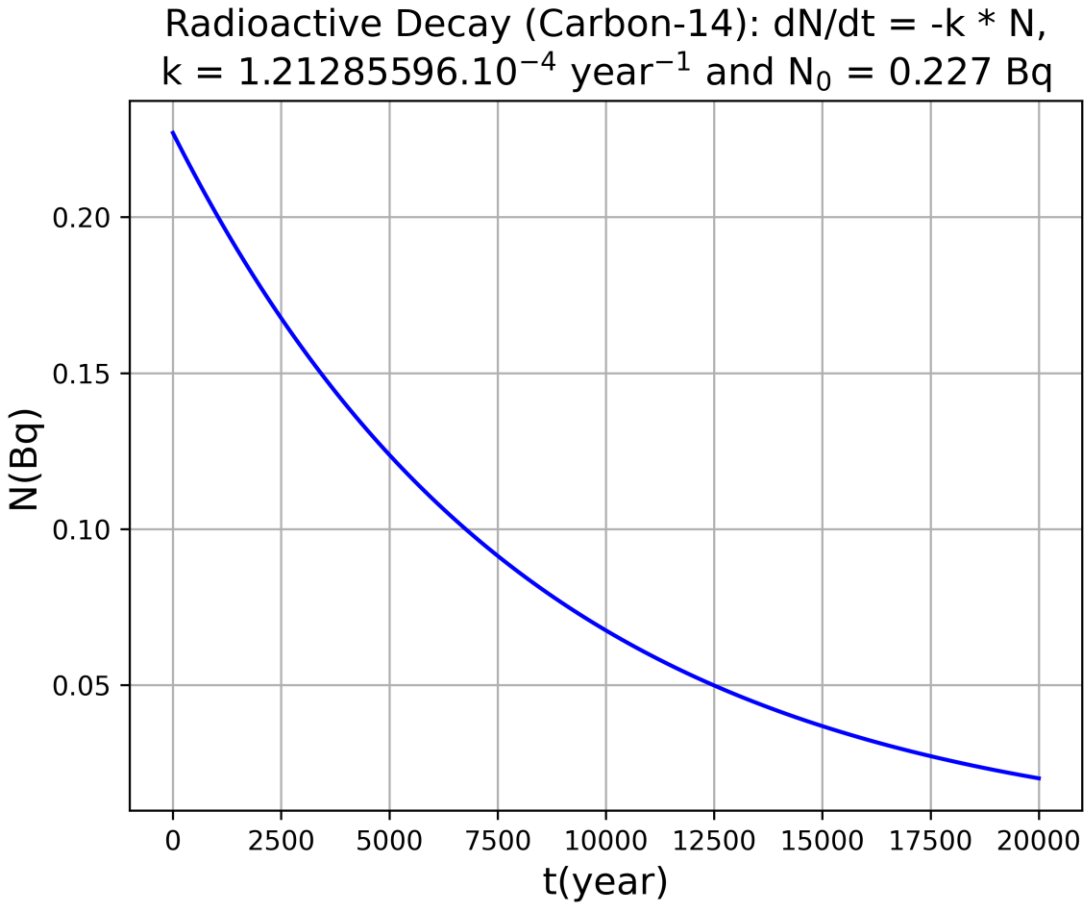
Nas próximas linhas, definimos os valores para gerarmos o gráfico e chamamos a função *plotting().* Usamos os valores definidos como argumentos da função.

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

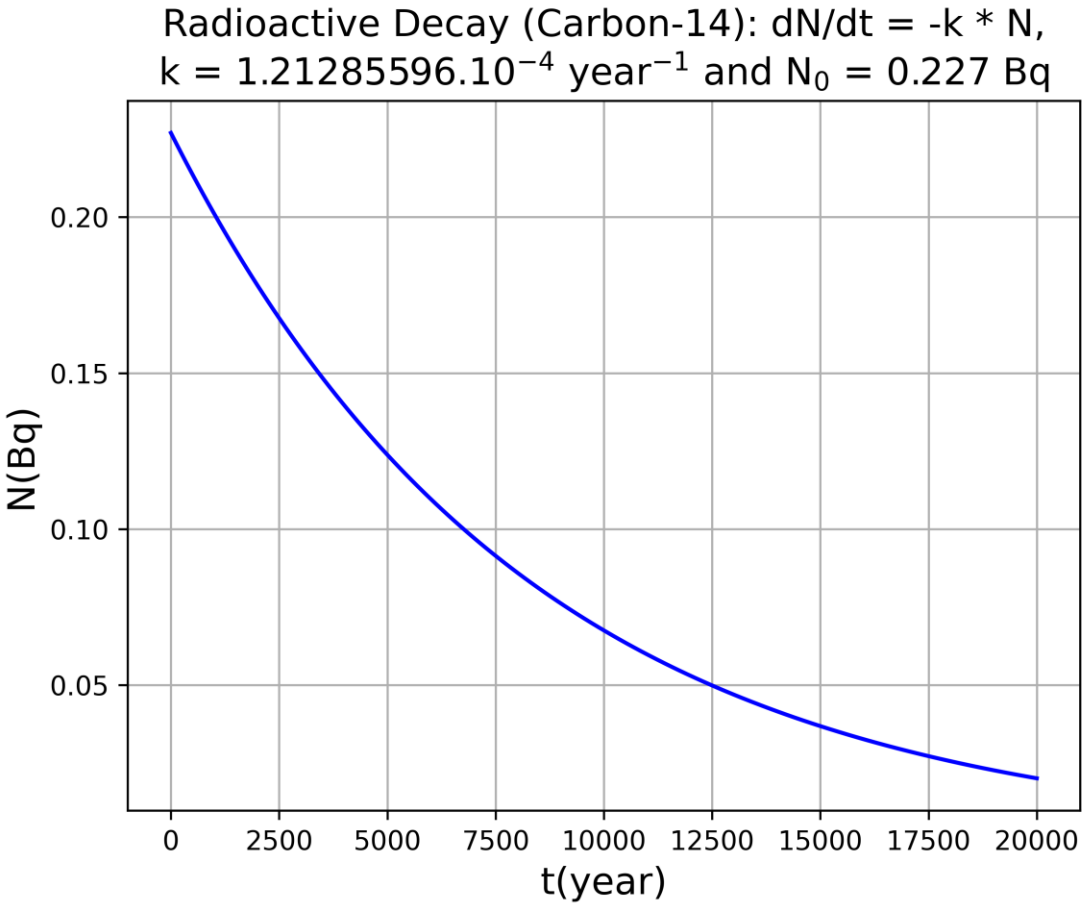
Por último, chamamos a função *main()* que executará todos os comandos da função *main().*

```
def main():
    # Initial condition
    N0 = 0.227   # Initial value related to the dependent variable (N)
    # Define constant
    k = 1.21285596e-04 # Carbon-14 half-life (t1/2) taken as 5,715 years
    # Time points (years)
    t = np.linspace(0,20000,20000)
    # Solve ODE
    N = odeint(model,N0,t,args=(k,))
    # Define plot parameters
    title_in = "Radioactive Decay (Carbon-14): dN/dt = -k * N,\n"
    title_in += "k = 1.21285596.10$^{-4}$ year$^{-1}$ and N$_0$ = 0.227 Bq"
    x_label_in = "t(year)"
    y_label_in = "N(Bq)"
    fontsize_title_in = 14
    fontsize_x_label_in = 14
    fontsize_y_label_in = 14
    color_in = "blue"
    # Call plotting() function
    plotting(t,N,title_in,x_label_in,y_label_in,fontsize_title_in,
            fontsize_x_label_in,fontsize_x_label_in,color_in)
main()
```

Ao rodar o código, temos o mesmo gráfico gerado com o programa *plot_radioactive_decay.py.* No programa *plot_radioactive_decay.py* fizemos diretamente o gráfico da função que é a solução da equação diferencial resolvida no programa *radioactive_decay.py*.



Podemos pensar no programa *plot_radioactive_decay.py* como um gabarito do programa *radioactive_decay.py*. O gráfico gerado no segundo programa (*radioactive_decay.py*) tem que ser o mesmo gerado no primeiro (*plot_radioactive_decay.py*). **Use essa abordagem no desenvolvimento dos projetos.**



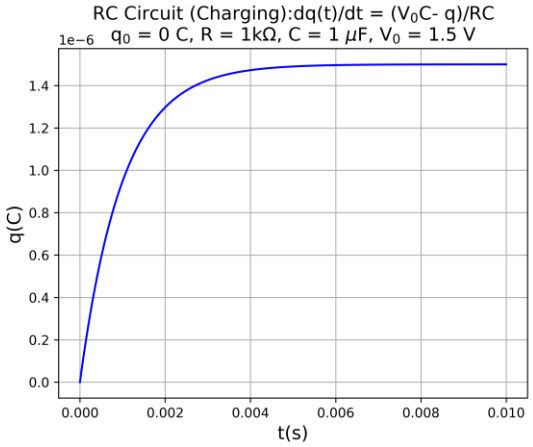
Agora veremos uma sequência de nove projetos onde serão implementadas a resolução de equações diferenciais que modelam sistemas físicos conhecidos. Vocês podem usar os códigos *plot_radioative_decay.py* e *radioative_decay.py* com protótipos dos seus programas. **Não esqueçam de mudar o nome dos seus programas.** Cada projeto prevê a entrega de dois programas. Por exemplo, para o projeto 01, vocês entregam os códigos: *projeto_01a.py* e *projeto_01b.py.* Um para gerar o gráfico da função (use como protótipo o código *plot_radioative_decay.py*) e outro para resolver a equação diferencial (use como protótipo o código *radioative_decay.py*).
Nos códigos protótipos, há várias linhas de comentários que iniciam com o símbolo #. Essas linhas estão em inglês e não são lidas pelo interpretador Python, são apenas para documentar o código. Vocês devem substituí-las por comentários sobre o seu código, ou simplesmente deletá-las. Você pode usar comentários em português. Não deixem as linhas originais de comentários nos seus programas, pode causar confusão se alguém for ler os seus códigos.

Data Final para Entrega do Projeto: 06/12/2023

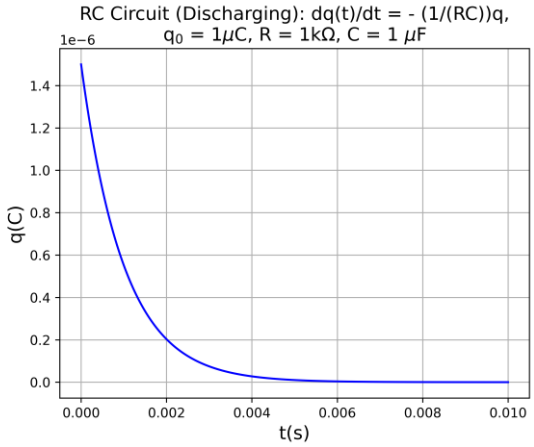
Projeto	Equação Diferencial	Solução	Parâmetros Sugeridos
01	$\frac{dq}{dt} = \left(\frac{V_0 C - q}{RC}\right)$	$q = V_0 C\left(1 - e^{-t/RC}\right)$	Q_0 = 0 C; V_0 = 1,5 V; R = 1 kΩ; C = 1 μF
02	$\frac{dq}{dt} = \left(-\frac{1}{RC}\right) q$	$q = Q_0 e^{-t/RC}$	Q_0 = 1.5.10^{-6}C; R = 1 kΩ; C = 1 μF
03	$\frac{dv}{dt} = a$	$v = v_0 + at$	v_0 = 5,0 m/s; a = 1,0 m/s^2
04	$\frac{dx}{dt} = v_0 + at$	$x = x_0 + v_0 t + \frac{a}{2} t^2$	x_0 = 2 m; v_0 = 5,0 m/s; a = 1,0 m/s^2
05	$\frac{dy}{dt} = v_0 \sin\theta_0 - gt$	$y = y_0 + v_0 \sin\theta_0\, t - \frac{g}{2} t^2$	y_0 = 0 m; v_0 = 30,0 m/s, θ_0 = 45º; g = 9,8 m/s^2
06	$\frac{dv}{dt} = a_0 + a_1 t + a_2 t^2$	$v = v_0 + a_0 t + \frac{a_1}{2} t^2 + \frac{a_2}{3} t^3$	v_0 = 0,0 m/s; a_0 = 8,302 m/s^2; a_1 = - 0,08874 m/s^3; a_2 = 1,6836.10^{-4} m/s^4
07	$\frac{dv}{dt} = g - \frac{b}{m} v$	$v = \left(\frac{mg}{b}\right)\left(1 - e^{-\frac{bt}{m}}\right)$	v_0 = 0,0 m/s; m = 65 kg; g = 9,8 m/s^2; b = 13,0 kg s^{-1}
08	$\frac{dT}{dt} = -k(T - T_a)$	$T = T_a + (T_0 - T_a) e^{-kt}$	T_0 = 98,6º C; T = 18,7º C; k = 0,203316 s^{-1}
09	$\frac{di}{dt} = \frac{V - Ri}{L}$	$i = \frac{V}{R}\left(1 - e^{-Rt/L}\right)$	i_0 = 0 A; V = 5 V; R = 50 Ω; L = 1 H

Projeto	Equação Diferencial	Sistema a Ser Modelado	Faixa de Valores para *t*
01	$\frac{dq}{dt} = \left(\frac{V_0 C - q}{RC}\right)$	Circuito RC em série com o capacitor sendo carregado.	*Entre 0 e 10*R*C* *t = np.linspace(0,10*R*C,1000)*
02	$\frac{dq}{dt} = \left(-\frac{1}{RC}\right) q$	Circuito RC em série com o capacitor sendo descarregado.	*Entre 0 e 10*R*C* *t = np.linspace(0,10*R*C,1000)*
03	$\frac{dv}{dt} = a$	Movimento unidimensional de uma partícula com aceleração constante. Estudo da velocidade.	*Entre 0 e 20* *t = np.linspace(0,20,1000)*
04	$\frac{dx}{dt} = v_0 + at$	Movimento unidimensional de uma partícula com aceleração constante. Estudo da posição.	*Entre 0 e 20* *t = np.linspace(0,20,1000)*
05	$\frac{dy}{dt} = v_0 \sin\theta_0 - gt$	Movimento balístico com o estudo da coordenada y.	*Entre 0 e 5* *t = np.linspace(0,5,1000)*
06	$\frac{dv}{dt} = a_0 + a_1 t + a_2 t^2$	Voo do ônibus especial. Estudo da velocidade.	*Entre 0 e 520* *t = np.linspace(0,520,1000)*
07	$\frac{dv}{dt} = g - \frac{b}{m} v$	Queda de um corpo levando em consideração a resistência do ar.	*Entre 0 e 100* *t = np.linspace(0,100,1000)*
08	$\frac{dT}{dt} = -k(T - T_a)$	Lei de resfriamento de Newton.	*Entre 0 e 50* *t = np.linspace(0,50,1000)*
09	$\frac{di}{dt} = \frac{V - Ri}{L}$	Circuito RL em série. Estudo da corrente elétrica.	*Entre 0 e 0.2* *t = np.linspace(0,0.2,1000)*

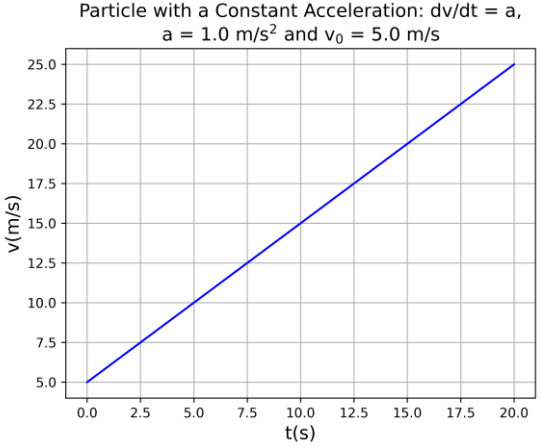
Seguem os gráficos esperados para cada projeto.



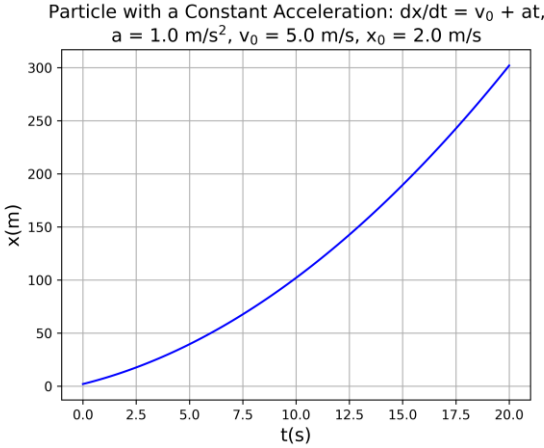
Projeto 01



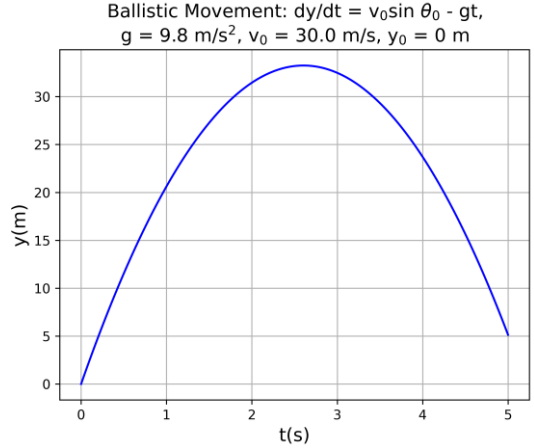
Projeto 02



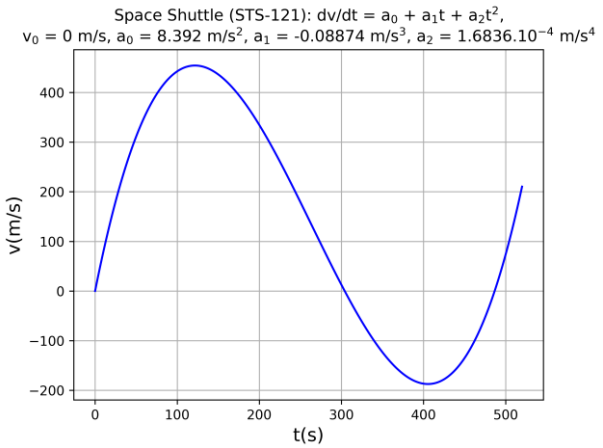
Projeto 03



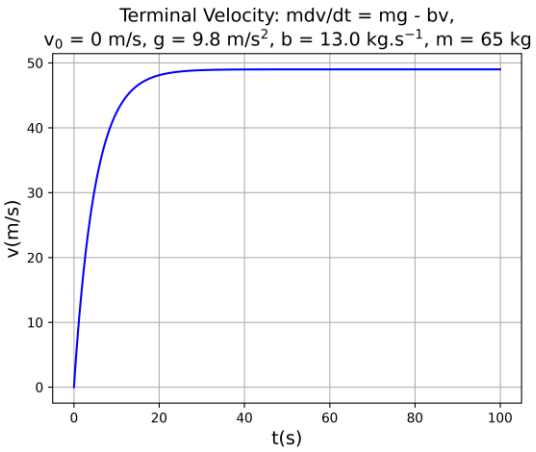
Projeto 04



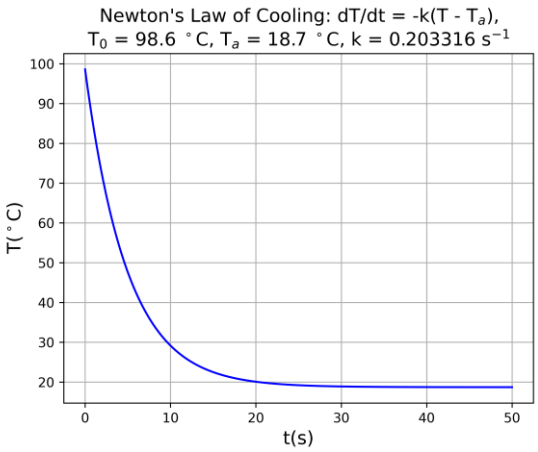
Projeto 05



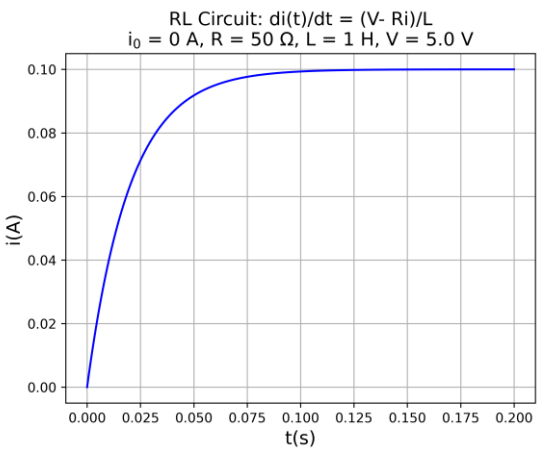
Projeto 06



Projeto 07



Projeto 08



Projeto 09

-BRESSERT, Eli. **SciPy and NumPy**. Sebastopol: O'Reilly Media, Inc., 2013. 56 p.
-DAWSON, Michael. **Python Programming, for the absolute beginner.** 3ed. Boston: Course Technology, 2010. 455 p.
-HAL, Tim, STACEY, J-P. **Python 3 for Absolute Beginners.** Springer-Verlag New York, 2009. 295 p.
-HETLAND, Magnus Lie. **Python Algorithms. Mastering Basic Algorithms in the Python Language.** Nova York: Springer Science+Business Media LLC, 2010. 316 p.
-IDRIS, Ivan. **NumPy 1.5. An action-packed guide dor the easy-to-use, high performance, Python based free open source NumPy mathematical library using real-world examples. Beginner's Guide.** Birmingham: Packt Publishing Ltd., 2011. 212 p.
-KIUSALAAS, Jaan. **Numerical Methods in Engineering with Python.** 2ed. Nova York: Cambridge University Press, 2010. 422 p.
-LANDAU, Rubin H. **A First Course in Scientific Computing: Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90**. Princeton: Princeton University Press, 2005. 481p.
-LANDAU, Rubin H., PÁEZ, Manuel José, BORDEIANU, Cristian C. **A Survey of Computational Physics. Introductory Computational Physics.** Princeton: Princeton University Press, 2008. 658 p.
-LUTZ, Mark. **Programming Python.** 4ed. Sebastopol: O'Reilly Media, Inc., 2010. 1584 p.
-NAGLE, R. Kent. **Equações diferenciais** (Portuguese Edition). Edição do Kindle.
-TOSI, Sandro. **Matplotlib for Python Developers.** Birmingham: Packt Publishing Ltd., 2009. 293 p.