**Trabalho Computacional 5**

**Entrega:** 22/11/2019

**PARTE 1**

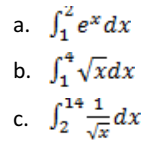
**Exercício 01 -** *Aproxime a integral de p(x)=x^4-3\*x^3+2\*x^2-3 entre -2 e 5 usando a regra do trapézio com n=30. Compare com o valor exato 14539/60.*

**Código implementado:**

| *def f(x):*  *return (x\*\*4)-(3\*(x\*\*3))+(2\*(x\*\*2))-3*  *xn = 5*  *x0 = -2*  *n = 30*  *h = (xn-x0)/n*  *x = x0 + h*  *soma = 0*  *for i in range(1, (n-1)):*  *soma += f(x)*  *x += h*  *I = (h/2)\*((f(x0)+f(xn))+(2\*soma))*  *print("Resultado: ", I)*  ***RESULTADO:***  *Resultado: 189.450730288066* |
| --- |

**Resposta:** O valor obtido foi de 189,451, com erro de aproximadamente 21% do valor exato (14539/60 = 242,317).

**Exercício 02 -** *Calcule as integrais a seguir pela regra dos trapézios e pela de Simpson, usando quatro e seis divisões de [a,b] respectivamente. Compare os resultados:*



**Código implementado:**

| *import math as mt*  *def fa(x):*  *return mt.exp(x)*  *def fb(x):*  *return x\*\*(1/2)*  *def fc(x):*  *return 1/(x\*\*(1/2))*  *def trapezio(item, x0, xn, n):*  *h = (xn-x0)/n*  *x = x0 + h*  *soma = 0*  *for i in range(1, (n-1)):*  *if item == 'a':*  *soma += fa(x)*  *elif item == 'b':*  *soma += fb(x)*  *elif item == 'c':*  *soma += fc(x)*  *x += h*  *if item == 'a':*  *I = (h/2)\*((fa(x0)+fa(xn))+(2\*soma))*  *elif item == 'b':*  *I = (h/2)\*((fb(x0)+fb(xn))+(2\*soma))*  *elif item == 'c':*  *I = (h/2)\*((fb(x0)+fb(xn))+(2\*soma))*  *return I*  *def simpsonMethod(item,min,max,n):*  *h=(max-min)/n*  *s=.0*  *aux=min*  *if item == 'a':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fa(aux)*  *elif(i%2==0):*  *s+=2\*fa(aux)*  *elif(i%2==1):*  *s+=4\*fa(aux)*  *aux+=h*  *s=(h/3)\*s*  *elif item == 'b':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fb(aux)*  *elif(i%2==0):*  *s+=2\*fb(aux)*  *elif(i%2==1):*  *s+=4\*fb(aux)*  *aux+=h*  *s=(h/3)\*s*  *elif item == 'c':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fc(aux)*  *elif(i%2==0):*  *s+=2\*fc(aux)*  *elif(i%2==1):*  *s+=4\*fc(aux)*  *aux+=h*  *s=(h/3)\*s*    *return s*    *print("Trapezio (a): ", trapezio('a', 1, 2, 4))*  *print("Trapezio (b): ", trapezio('b', 1, 4, 4))*  *print("Trapezio (c): ", trapezio('c', 2, 14, 4))*  *print("Simpson (a): ", simpsonMethod('a', 1, 2, 6))*  *print("Simpson (b): ", simpsonMethod('b', 1, 4, 6))*  *print("Simpson (c): ", simpsonMethod('c', 2, 14, 6))*  ***RESULTADO:***  *Trapezio (a): 3.2564252478736884*  *Trapezio (b): 3.3030108642123635*  *Trapezio (c): 10.13610738200025*  *Simpson (a): 4.670794226633775*  *Simpson (b): 4.6665630532224895*  *Simpson (c): 4.66148949120848* |
| --- |

**Resposta:** Como esperado o método de simpson obteve melhores resultados com um número de intervalos menor, com testes variando o valor de n, o método do trapézio atinge resultados melhores quando o n é grande, mas para o método de simpson o ganho de precisão não é tão grande ao aumentar n.

**Exercício 03 -** *Utilizando as integrais do exercício anterior com quantas divisões do intervalo, no mínimo, podemos esperar obter erros menores que 10-5 ?*

**Código implementado:**

| *import math as mt*  *def fa(x):*  *return mt.exp(x)*  *def fb(x):*  *return x\*\*(1/2)*  *def fc(x):*  *return 1/(x\*\*(1/2))*  *def trapezio(item, x0, xn, n):*  *h = (xn-x0)/n*  *x = x0 + h*  *soma = 0*  *for i in range(1, (n-1)):*  *if item == 'a':*  *soma += fa(x)*  *elif item == 'b':*  *soma += fb(x)*  *elif item == 'c':*  *soma += fc(x)*  *x += h*  *if item == 'a':*  *I = (h/2)\*((fa(x0)+fa(xn))+(2\*soma))*  *elif item == 'b':*  *I = (h/2)\*((fb(x0)+fb(xn))+(2\*soma))*  *elif item == 'c':*  *I = (h/2)\*((fb(x0)+fb(xn))+(2\*soma))*  *print(item, "Trapesio", "\n Quantidade de intervalos para erro menor que 10^-5", minQtdGap(x0,xn))*  *return I*  *def simpsonMethod(item,min,max,n):*  *h=(max-min)/n*  *s=.0*  *aux=min*  *if item == 'a':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fa(aux)*  *elif(i%2==0):*  *s+=2\*fa(aux)*  *elif(i%2==1):*  *s+=4\*fa(aux)*  *aux+=h*  *s=(h/3)\*s*  *elif item == 'b':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fb(aux)*  *elif(i%2==0):*  *s+=2\*fb(aux)*  *elif(i%2==1):*  *s+=4\*fb(aux)*  *aux+=h*  *s=(h/3)\*s*  *elif item == 'c':*  *for i in range(n+1):*  *if (i==0 or i==(n)):*  *s+=fc(aux)*  *elif(i%2==0):*  *s+=2\*fc(aux)*  *elif(i%2==1):*  *s+=4\*fc(aux)*  *aux+=h*  *s=(h/3)\*s*  *print(item, "Simpson", "\n Quantidade de intervalos para erro menor que 10^-5", minQtdGap(min,max))*  *return s*    *def minQtdGap(min,max):*  *n=0.*  *gap=(max-min)*  *ngap=0.*  *res=1.*  *x=0.*  *for i in range (int(min)\*1000,int(max+1)\*1000,1):*  *if(numpy.exp(i/1000)>x):*  *x=i/1000*  *while(res>=0.00001):*  *n+=1.0*  *ngap=gap/n*  *res=(max-min)\*(ngap\*\*2)\*(numpy.exp(x))/12*  *return n*  *trapezio('a', 1, 2, 4)*  *trapezio('b', 1, 4, 4)*  *trapezio('c', 2, 14, 4)*  *simpsonMethod('a', 1, 2, 6)*  *simpsonMethod('b', 1, 4, 6)*  *simpsonMethod('c', 2, 14, 6)*  ***RESULTADO:***  *a Trapesio*  *Quantidade de intervalos para erro menor que 10^-5 409.0*  *b Trapesio*  *Quantidade de intervalos para erro menor que 10^-5 5776.0*  *c Trapesio*  *Quantidade de intervalos para erro menor que 10^-5 6857609.0*  *a Simpson*  *Quantidade de intervalos para erro menor que 10^-5 409.0*  *b Simpson*  *Quantidade de intervalos para erro menor que 10^-5 5776.0*  *c Simpson*  *Quantidade de intervalos para erro menor que 10^-5 6857609.0* |
| --- |

**Exercício 04 -** *Calcule o valor aproximado da integral abaixo com três casas decimais de*

*precisão usando:*

a. Simpson

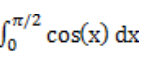
b. Trapézio

**Código implementado:**

| ***a)***  *import math*  *import numpy*  *def eqGiven(x):*  *return 1/(1+x)*  *def simpsonMethod(n,min,max):*  *h=(max-min)/n*  *s=.0*  *aux=min*  *for i in range(int(n+1)):*  *if (i==0 or i==(n)):*  *s+=eqGiven(aux)*  *elif(i%2==0):*  *s+=2\*eqGiven(aux)*  *elif(i%2==1):*  *s+=4\*eqGiven(aux)*  *aux+=h*  *s=(h/3)\*s*  *print("Método de Simpson", )*  *return s*  *def minQtdGap(min,max,d):*  *n=0.*  *gap=(max-min)*  *ngap=0.*  *res=1.*  *x=0.*  *for i in range (int(min)\*1000,int(max+1)\*1000,1):*  *if(i!=0):*  *if((24/(1+x)\*\*5)>x or x==0):*  *x=i/1000*  *while(res>=(1/10\*\*d)):*  *n+=1.0*  *ngap=gap/n*  *res=gap\*(ngap\*\*4)\*(24/(1+x)\*\*5)/(180)*  *if(n%2==1):*  *n+=1*  *return n*  *def pResult(s,min,max):*  *print("Integral de {} a {} de dx/(1+x):\n{} ".format(min,max,s))*  *min = 0.*  *max = 0.6*  *d = 3.*  *n = minQtdGap(min,max,d)*  *pResult(simpsonMethod(n,min,max),min,max)*  *print("Qtd de intervalos para precisão de {} casas decimais: {}".format(d,minQtdGap(min,max,d)))*  ***RESULTADO:***  *Integral de 0.0 a 0.6 de dx/(1+x):*  *0.47019230769230763*  *Qtd de intervalos para precisão de 3.0 casas decimais: 2.0*  *------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------*  ***b)***  *import math*  *import numpy*  *def eqGiven(x):*  *return 1/(1+x)*  *def integralTrapezio(min,max,n):*  *delta=(max-min)/n #delta dos tapezios*  *aux=.0*  *y=.0*  *y=eqGiven(min)*  *aux=min*  *for i in range (1,int(n)):*  *aux+=delta*  *y+=2\*eqGiven(aux)*  *res = (delta/2)\*(y+eqGiven(max))*  *return res*  *def minQtdGap(min,max,erro):*  *n=0.*  *gap=(max-min)*  *ngap=0.*  *res=1.*  *x=0.*  *for i in range (int(min)\*1000,int(max+1)\*1000,1):*  *if(i!=0):*  *if((2/(1+i)\*\*3)>x or x==0):*  *x=i/1000*  *while(res>=(1/10\*\*d)):*  *n+=1.0*  *ngap=gap/n*  *res=(max-min)\*(ngap\*\*2)\*(2/(1+x)\*\*3)/12 #segunda derivada de eqGiven*  *return n*  *min=0.*  *max=0.6*  *d = 3.0*  *n=minQtdGap(min,max,d)*  *print("Valor da integral de para dx/(1+x) de {} a {}:\n{}".format(min,max,integralTrapezio(min,max,n)))*  *print("Qtd de intervalos para precisão de {} casas decimais: {}".format(d,minQtdGap(min,max,d)))*  ***RESULTADO:***  *Valor da integral de para dx/(1+x) de 0.0 a 0.6:*  *0.4705107392607392*  *Qtd de intervalos para precisão de 3.0 casas decimais: 6.0* |
| --- |

**Exercício 05 -** *Determine h, a distância entre xi e xi+1, para que se possa avaliar a integral*

*abaixo com erro inferior a 10-3 pela regra de Simpson, onde*



**Código implementado:**

| *import math as mt*  *def eqGiven(x):*  *return mt.cos(x)*  *def simpsonMethod(n,min,max):*  *h=(max-min)/n*  *s=.0*  *aux=min*  *for i in range(int(n+1)):*  *if (i==0 or i==(n)):*  *s+=eqGiven(aux)*  *elif(i%2==0):*  *s+=2\*eqGiven(aux)*  *elif(i%2==1):*  *s+=4\*eqGiven(aux)*  *aux+=h*  *s=(h/3)\*s*  *print("Método de Simpson", )*  *return s*  *def minQtdGap(min,max,d):*  *n=0.*  *gap=(max-min)*  *ngap=0.*  *res=1.*  *x=0.*  *for i in range (int(min)\*1000,int(max+1)\*1000,1):*  *if(i!=0):*  *if((24/(1+x)\*\*5)>x or x==0):*  *x=i/1000*  *while(res>=(1/10\*\*d)):*  *n+=1.0*  *ngap=gap/n*  *res=gap\*(ngap\*\*4)\*(24/(1+x)\*\*5)/(180)*  *if(n%2==1):*  *n+=1*  *return n*  *min = 0.*  *max = mt.pi/2*  *d = 3.*  *n = minQtdGap(min,max,d)*  *print("Integral", simpsonMethod(n,min,max))*  *print("Qtd de intervalos para precisão de {} casas decimais: {}".format(d,minQtdGap(min,max,d)))*  ***RESULTADO:***  *Método de Simpson*  *Integral 1.0001345849741938*  *Qtd de intervalos para precisão de 3.0 casas decimais: 4.0* |
| --- |

**PARTE 2**

**Exercício 01 -** *#Paleontologia*

*A massa de uma amostra radioativa decai a uma taxa proporcional a sua massa.*

*(a) Expresse esse fato como uma equação diferencial para a massa M(t), usando k para a constante de proporcionalidade.*

***Resposta:*** dM/dt + k.M(t) = 0

*(b) Se a massa inicial é M0, ache a expressão para a massa M(t).*

***Resposta:*** Usando o método do fator integrante, teremos:

M’ + kM(t) = 0 ⇒ μM’ + μ’M = d(Mμ)/dt

Sabendo que μ = ek∫dt, teremos:

d(Mμ)/dt = μ\*0 ⇒ d(Mμ)/dt = 0

Portanto,

M(t) = M0 . e -k.t

*(c) A meia-vida de uma amostra é a quantidade de tempo necessária para metade da massa decair. Sabendo que a meia-vida do Carbono-14 é 5730 anos, encontre o valor de K para uma amostra de Carbono-14.*

*(d) Em quanto tempo uma amostra de Carbono-14 será reduzida a um quarto de sua massa original?*

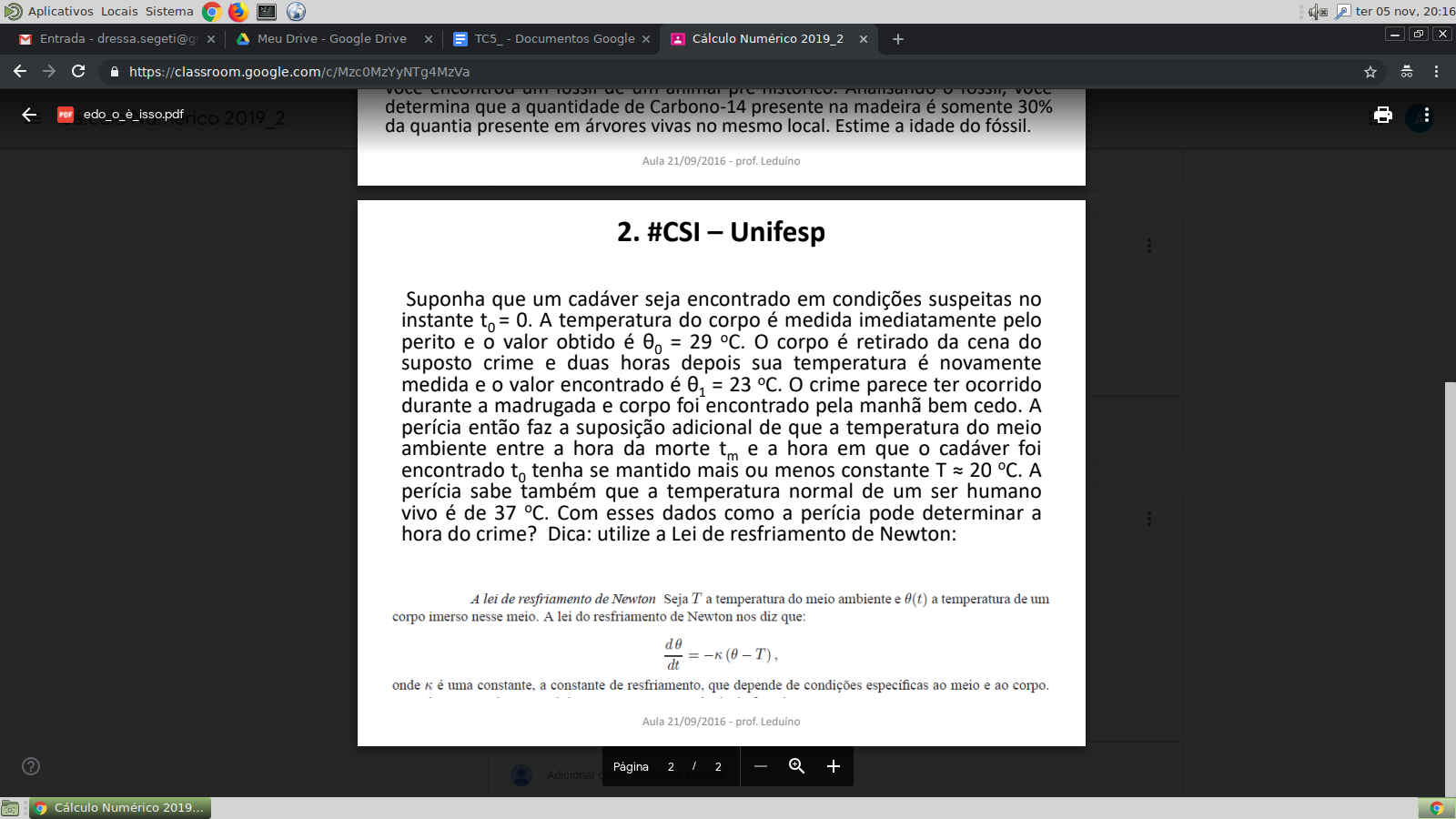
*(e) O Carbono-14 está naturalmente presente no meio ambiente. Todo organismo vivo absorve Carbono-14 pela alimentação e respiração. Quando morre, entretanto, o organismo não absorve mais Carbono-14. Suponha que você encontrou um fóssil de um animal pré-histórico. Analisando o fóssil, você determina que a quantidade de Carbono-14 presente na madeira é somente 30% da quantia presente em árvores vivas no mesmo local. Estime a idade do fóssil.*

**Código implementado:**

| *import numpy as np*  *import math as mt*  ***# Letra C***  *massa0 = 14*  *t = 5730*  *k = 0*  *M = 7*  *k = - np.log(M/massa0)/t*  *print("c) Valor de k = ", k)*  ***# Letra D***  *MM = 14/4*  *tt = 0*  *tt = - np.log(MM/massa0)/k*  *print("d) Valor de t = ", tt)*  ***# Letra E***  *MMM = 0.3*  *massa\_m = 1*  *ttt = 0*  *ttt= - np.log(MMM/massa\_m)/k*  *print("e) Valor de t(30%) = ", ttt)*  ***RESULTADOS:***  *c) Valor de k = 0.00012096809433855938*  *d) Valor de t = 11460.0*  *e) Valor de t(30%) = 9952.812854572363* |
| --- |

**Exercício 02 -** *#CSI – Unifesp*

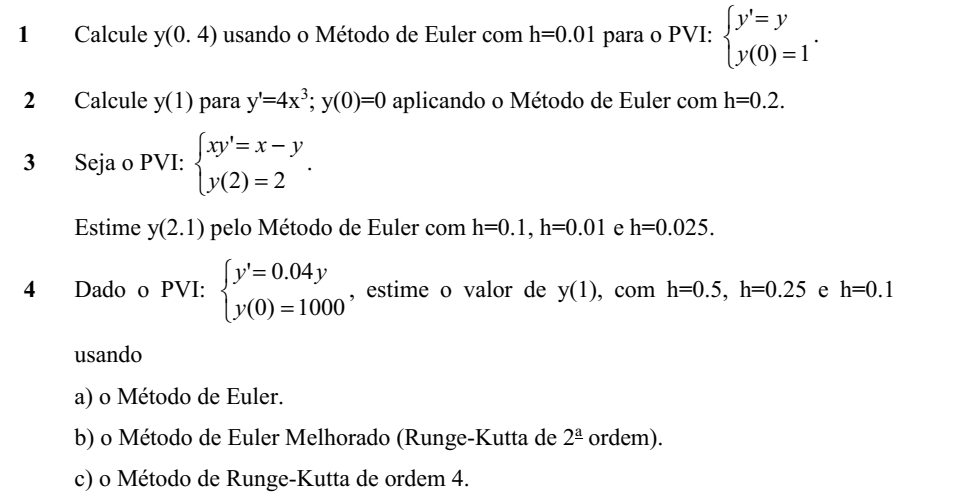
*Suponha que um cadáver seja encontrado em condições suspeitas no instante t0 = 0. A temperatura do corpo é medida imediatamente pelo perito e o valor obtido é θ0 = 29ºC. O corpo é retirado da cena do suposto crime e duas horas depois sua temperatura é novamente medida e o valor encontrado é θ1 = 23ºC. O crime parece ter ocorrido durante a madrugada e corpo foi encontrado pela manhã bem cedo. A perícia então faz a suposição adicional de que a temperatura do meio ambiente entre a hora da morte tm e a hora em que o cadáver foi encontrado t0 tenha se mantido mais ou menos constante T ≈ 20ºC. A perícia sabe também que a temperatura normal de um ser humano vivo é de 37ºC. Com esses dados como a perícia pode determinar a hora do crime? Dica: utilize a Lei de resfriamento de Newton:*

**

**Resolução:**

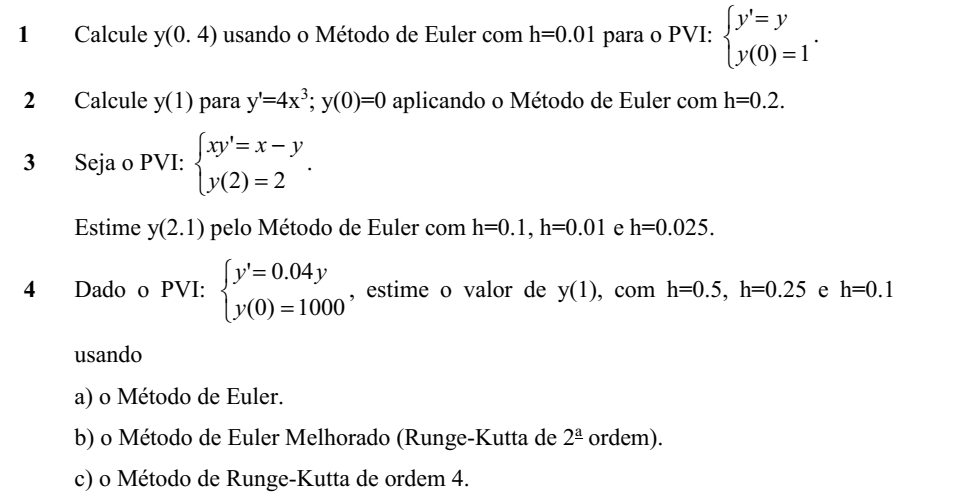
|  |
| --- |

**PARTE 3**



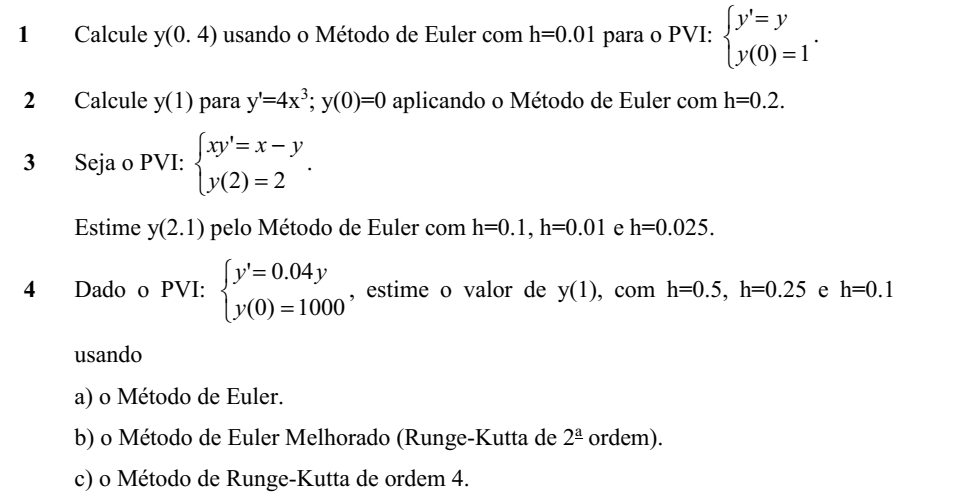
**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*(yn)*  *tn+=h*  *print(yn)*  *t0=0*  *y0=1*  *h=0.01*  *x=0.4*  *eulerEDO(h,y0,t0,x)*    ***RESULTADO:***  *1.4888637335882209* |
| --- |



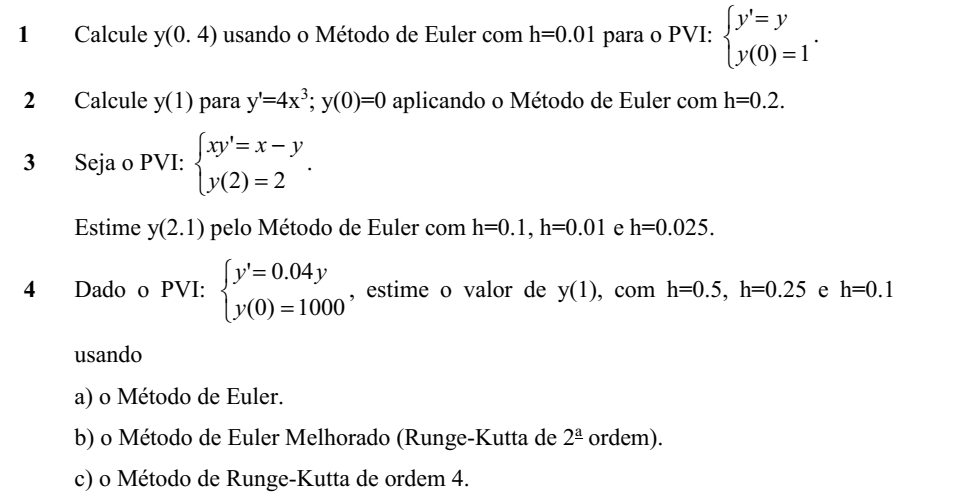
**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*(4\*tn\*\*3)*  *tn+=h*  *print(yn)*  *t0=0*  *y0=0*  *h=0.2*  *x=1*  *eulerEDO(h,y0,t0,x)*  ***RESULTADO:***  *1.4400000000000004* |
| --- |



**Código implementado:**

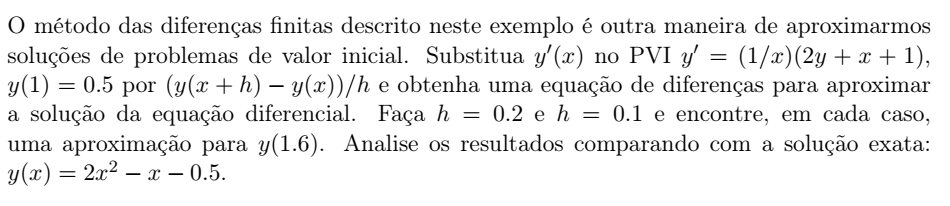
| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*((tn-yn)/tn)*  *tn+=h*  *return yn*  *'''letra a'''*  *t0=2*  *y0=2*  *h=0.1*  *x=2.1*  *print("O resultado(letra a) é",eulerEDO(h,y0,t0,x))*  *'''letra b'''*  *t0=2*  *y0=2*  *h=0.01*  *x=2.1*  *print("O resultado(letra b) é",eulerEDO(h,y0,t0,x))*  *'''letra c'''*  *t0=2*  *y0=2*  *h=0.025*  *x=2.1*  *print("O resultado(letra c) é",eulerEDO(h,y0,t0,x))*  ***RESULTADO:***  *O resultado (letra a) é 2.0047619047619047*  *O resultado (letra b) é 2.0026190476190475*  *O resultado (letra c) é 2.0029761904761902* |
| --- |



**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*(0.04\*yn)*  *tn+=h*  *return yn*  *def rungeKutta2(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *s1=0.04\*yn*  *s2=0.04\*(yn+h\*s1)*  *yn = yn+(h/2)\*(s1+s2)*  *tn+=h*  *return yn*  *def rungeKutta4(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *s1=0.*  *s2=0.*  *s3=0.*  *s4=0.*  *if(x!=0):*  *while(tn<=x):*  *s1=(0.04\*yn)*  *s2=(0.04\*(yn+(h/2\*s1)))*  *s3=(0.04\*(yn+(h/2\*s2)))*  *s4=(0.04\*(yn+(h\*s3)))*  *yn = yn + (h/6)\*(s1+2\*s2+2\*s3+s4)*  *tn+=h*  *return yn*  *'''Primeiro Valor de h'''*  *t0=0*  *y0=1000*  *h=0.5*  *x=1*  *print("O resultado(letra a, primeiro h) é",eulerEDO(h,y0,t0,x)) #Letra a*  *print("O resultado(letra b, primeiro h) é",rungeKutta2(h,y0,t0,x)) #Letra b*  *print("O resultado(letra c, primeiro h) é",rungeKutta4(h,y0,t0,x)) #Letra c*  *'''Segundo Valor de h'''*  *t0=0*  *y0=1000*  *h=0.25*  *x=1*  *print("O resultado(letra a, segundo h) é",eulerEDO(h,y0,t0,x)) #Letra a*  *print("O resultado(letra b, segundo h) é",rungeKutta2(h,y0,t0,x)) #Letra b*  *print("O resultado(letra c, segundo h) é",rungeKutta4(h,y0,t0,x)) #Letra c*  *'''Terceiro Valor de h'''*  *t0=0*  *y0=1000*  *h=0.1*  *x=1*  *print("O resultado(letra a, terceiro h) é",eulerEDO(h,y0,t0,x)) #Letra a*  *print("O resultado(letra b, terceiro h) é",rungeKutta2(h,y0,t0,x)) #Letra b*  *print("O resultado(letra c, terceiro h) é",rungeKutta4(h,y0,t0,x)) #Letra c*  ***RESULTADO:***  *O resultado(letra a, primeiro h) é 1061.208*  *O resultado(letra b, primeiro h) é 1061.832362408*  *O resultado(letra c, primeiro h) é 1061.8365464618164*  *O resultado(letra a, segundo h) é 1051.0100501*  *O resultado(letra b, segundo h) é 1051.2702268613002*  *O resultado(letra c, segundo h) é 1051.27109637168*  *O resultado(letra a, terceiro h) é 1044.8906449549856*  *O resultado(letra b, terceiro h) é 1044.9822326444335*  *O resultado(letra c, terceiro h) é 1044.9823548883462* |
| --- |

**Exercício 05 -**



**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*((1/tn)\*(2\*yn + tn + 1))*  *tn+=h*  *return yn*  *def solexataEDO(x):*  *yn=2\*(x\*\*2)-x-0.5*  *return yn*  *t0=1*  *y0=0.5*  *h=0.1*  *x=1.6*  *print("O resultado em 0.1 é",eulerEDO(h,y0,t0,x))*  *t0=1*  *y0=0.5*  *h=0.2*  *x=1.6*  *print("O resultado em 0.2 é",eulerEDO(h,y0,t0,x))*  *print("O resultado da solução exata é",solexataEDO(x))*  ***RESULTADO:***  *O resultado em 0.1 é 2.845454545454545*  *O resultado em 0.2 é 3.7*  *O resultado da solução exata é 3.020000000000001* |
| --- |

**Exercício 06 -**  *[Aplicação em farmacologia]*

*Um dos problemas clássicos em Farmacologia consiste em saber como decai a concentração de uma droga administrada em um paciente. A informação sobre esse fato admite que a dosagem seja aplicada na medida correta no paciente e o intervalo de tempo adequado entre cada aplicação. O modelo em questão assume que seja C(t) a concentração de um remédio na circulação sanguínea de um determinado paciente. À medida que o corpo elimina o remédio C(t) diminui a uma taxa proporcional à quantidade de medicamento que está presente naquele momento. Então a equação diferencial que modela o problema pode ser escrita como*



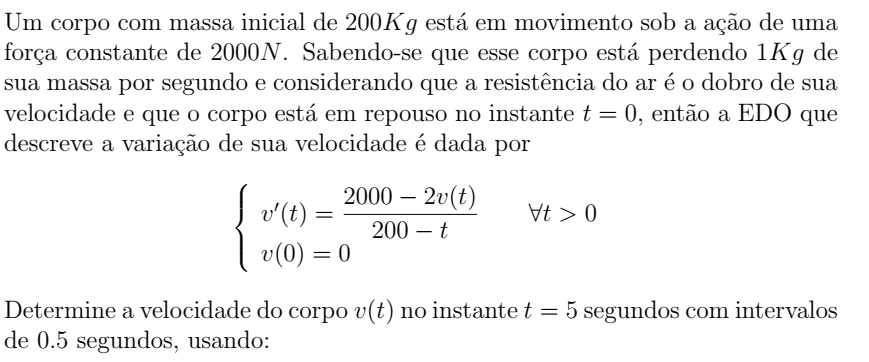
*onde k é um número positivo chamado constante de eliminação do remédio.****a)*** *Obtenha o valor de k analiticamente.*

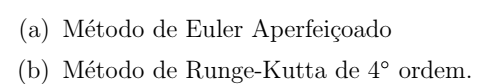
***b)*** *Use dois métodos numéricos estudados para determinar quanto tempo o paciente leva para eliminar 90% do medicamento aplicado, sabendo-se que o corpo elimina metade do remédio em 4 horas. Compare com o resultado real.*

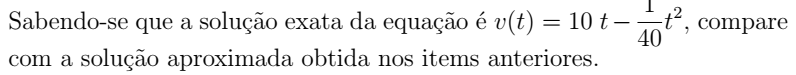
**Código implementado:**

|  |
| --- |

**Exercício 07 -**





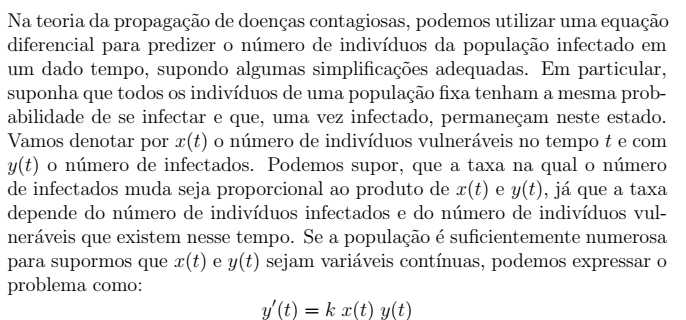


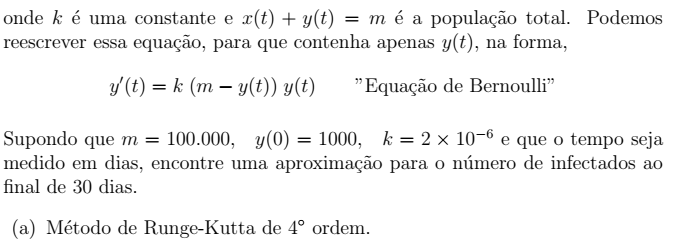
**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *if(x!=0):*  *while(tn<=x):*  *yn = yn + h\*(2000-2\*yn)/(200-tn)*  *tn+=h*  *return yn*  *def solexataEDO(x):*  *yn=(10\*x)+(1/40)\*(x\*\*2)*  *return yn*  *def rungeKutta4(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *s1=0.*  *s2=0.*  *s3=0.*  *s4=0.*  *if(x!=0):*  *while(tn<=x):*  *s1=(2000-yn)/(200-tn)*  *s2=(2000-yn+s1\*h/2)/(200-tn+h/2)*  *s3=(2000-yn+s2\*h/2)/(200-tn+h/2)*  *s4=(2000-yn+s3\*h)/(200-tn+h)*  *yn = yn + (h/6)\*(s1+2\*s2+2\*s3+s4)*  *tn+=h*  *return yn*  *t0=0*  *y0=0*  *h=0.5*  *x=5*  *print("Solução exata: ", solexataEDO(x))*  *print("(a) Metodo de Euler: ", eulerEDO(h,y0,t0,x))*  *print("(b) Runge-Kutta", rungeKutta4(h,y0,t0,x))*  ***RESULTADO:***  *Solução exata: 50.625*  *(a) Metodo de Euler: 54.310776942355886*  *(b) Runge-Kutta 55.0* |
| --- |

**Resposta:** Utilizando os Métodos de Euler e Runge-Kutta obteve-se resultados aproximados da solução exata, com erro em torno de 8% em ambos os casos.

**Exercício 08 -**





**Código implementado:**

| *import numpy*  *import math*  *def rungeKutta4(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *k=0.*  *s1=0.*  *s2=0.*  *s3=0.*  *s4=0.*  *if(x!=0):*  *while(tn<=x):*  *s1=2\*10\*\*(-6)\*(100000-yn)*  *s2=2\*10\*\*(-6)\*(100000-(yn+0.5\*h\*s1))\*(yn+0.5\*h\*s1)*  *s3=2\*10\*\*(-6)\*(100000-(yn+0.5\*h\*s2))\*(yn+0.5\*h\*s2)*  *s4=2\*10\*\*(-6)\*(100000-(yn+h\*s3))\*(yn+h\*s3)*  *yn = yn + (h/6)\*(s1+2\*s2+2\*s3+s4)*  *tn+=h*  *return yn*  *t0=0*  *y0=1000*  *h=0.00001*  *x=30*  *print("O número aproximado de pessoas infectadas será de", rungeKutta4(h,y0,t0,x))*  ***RESULTADO:***  *O número aproximado de pessoas infectadas será de 80295.74692040357* |
| --- |

**Exercício 09 -** *Resolva os exercícios anteriores usando uma função adequada do python. Compare com os resultados que você obteve.*

**Código implementado:**

|  |
| --- |

**PARTE 4**

**Exercício 1** - *A equação diferencial*

*descreve a carga Q sobre um capacitor com capacitância C durante um processo de carregamento envolvendo uma resistência R e uma tensão V. Se a carga é nula quando t=0, aproxime Q no intervalo [0,4], com R=2, C=3 e V=4 usando Runge-Kutta de 4a ordem.*

**Código implementado:**

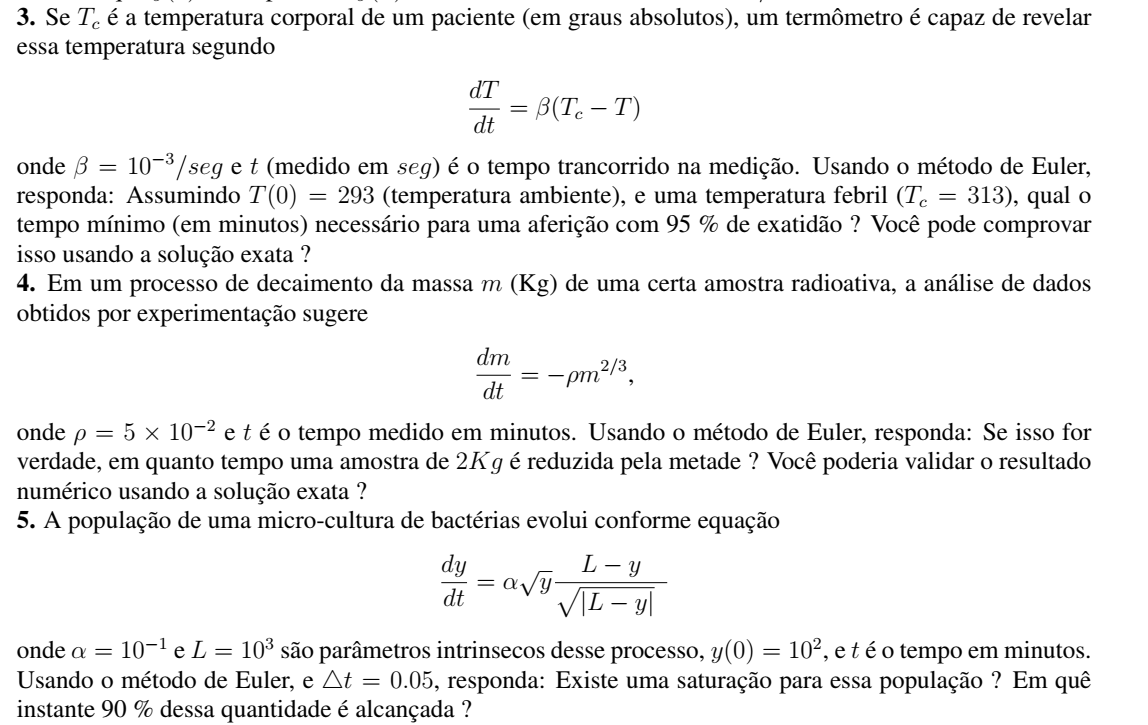
| *import numpy*  *import math*  *def rungeKutta4(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *s1=0.*  *s2=0.*  *s3=0.*  *s4=0.*  *R=2.*  *C=3.*  *V=4.*  *if(x!=0):*  *while(tn<=x):*  *s1=(R/C)\*((V\*C)-yn)*  *s2=(R/C)\*((V\*C)-yn+s1\*h/2)*  *s3=(R/C)\*((V\*C)-yn+s2\*h/2)*  *s4=(R/C)\*((V\*C)-yn+s3\*h)*  *yn = yn + (h/6)\*(s1+2\*s2+2\*s3+s4)*  *tn+=h*  *return yn*  *t0=0*  *y0=0*  *h=0.1*  *x=1*  *print("O valor de carga no capacitor em 1 será de", rungeKutta4(h,y0,t0,x))*  *t0=0*  *y0=0*  *h=0.1*  *x=2*  *print("O valor de carga no capacitor em 2 será de", rungeKutta4(h,y0,t0,x))*  *t0=0*  *y0=0*  *h=0.1*  *x=3*  *print("O valor de carga no capacitor em 3 será de", rungeKutta4(h,y0,t0,x))*  *t0=0*  *y0=0*  *h=0.1*  *x=4*  *print("O valor de carga no capacitor em 4 será de", rungeKutta4(h,y0,t0,x))*  ***RESULTADO:***  *O valor de carga no capacitor em 1 será de 6.53059289867596*  *O valor de carga no capacitor em 2 será de 9.124303348405798*  *O valor de carga no capacitor em 3 será de 10.592255028904004*  *O valor de carga no capacitor em 4 será de 11.310864064000826* |
| --- |

**Exercício 2 -** *A população em uma cidade muda na seguinte taxa:*

*Onde p(t) denota o número de pessoas na população no tempo t (em anos) e k0, k1 são constantes específicas. Dado que a população inicial é igual a 1000 e k0=0.02, k1=0.4, aproxime o tamanho da população após 10 anos usando o método de Runge-Kutta de 4a ordem com N=10.*

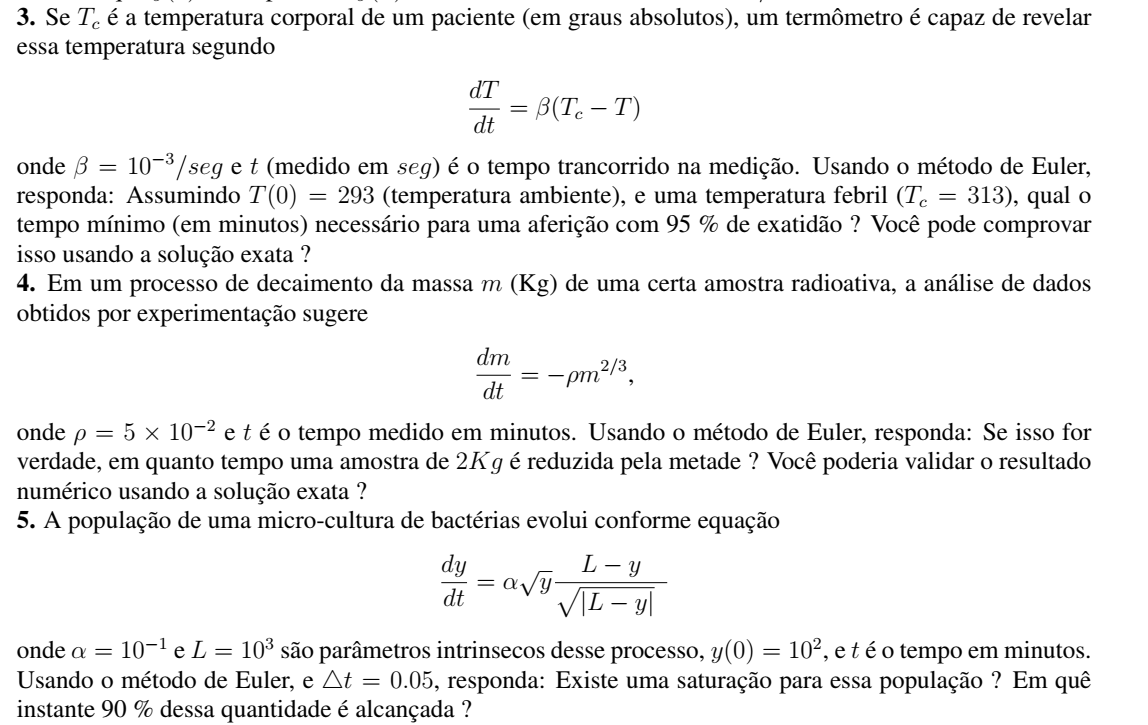
**Código implementado:**

| *import numpy*  *import math*  *def rungeKutta4(h,y0,t0,x):*  *yn=0.*  *yn=y0*  *tn=t0*  *s1=0.*  *s2=0.*  *s3=0.*  *s4=0.*  *k0=0.02*  *k1=0.4*  *if(x!=0):*  *while(tn<=x):*  *s1=(k0+(k1\*yn))\*yn*  *s2=(k0+(k1\*yn+s1\*h/2))\*(yn+s1\*h/2)*  *s3=(k0+(k1\*yn+s2\*h/2))\*(yn+s2\*h/2)*  *s4=(k0+(k1\*yn+s3\*h))\*(yn+s3\*h)*  *yn = yn + (h/6)\*(s1+2\*s2+2\*s3+s4)*  *tn+=h*  *return yn*  *t0=0*  *y0=1000*  *h=11*  *x=10*  *print("O valor populacional em 10 anos será", rungeKutta4(h,y0,t0,x))*  ***RESULTADO:***  *O valor populacional em 10 anos será 1.1172180726165105e+56* |
| --- |



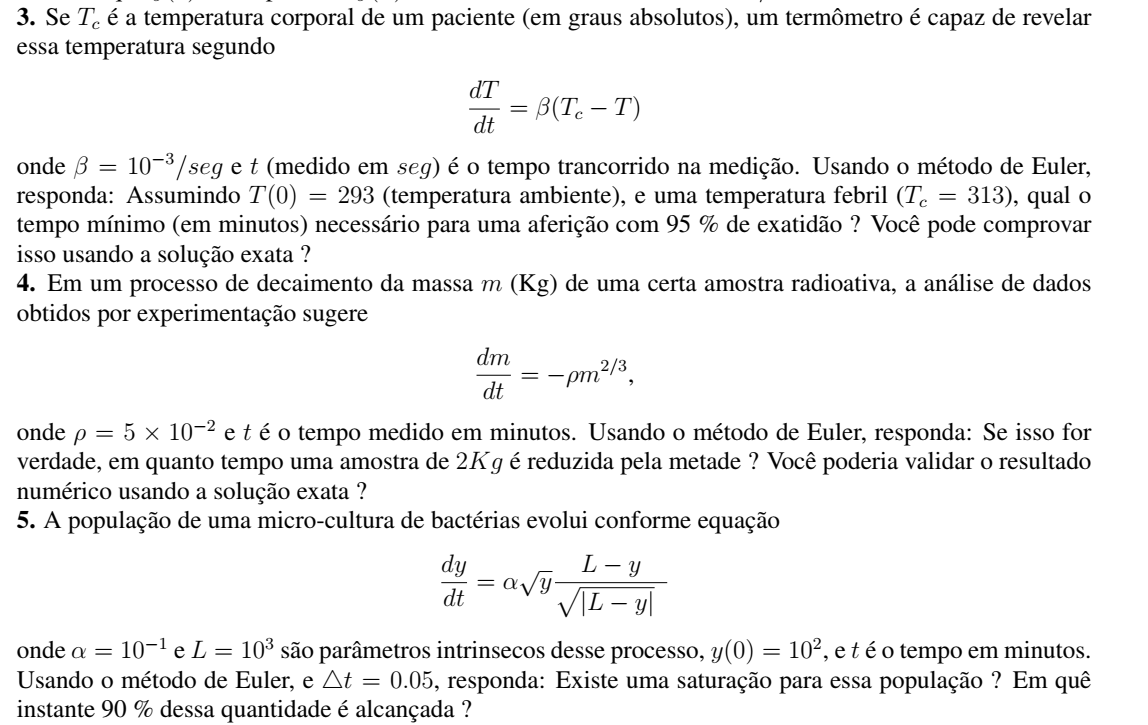
**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0):*  *yn=0.*  *yn=y0*  *tn=t0*  *if(True):*  *while(yn<=312):*  *yn = yn + h\*(0.001)\*(313 - yn)*  *tn+=h*  *return tn/60*  *t0=0*  *y0=293*  *h=1*  *print("O valor de tempo em minutos é", eulerEDO(h,y0,t0))*  ***RESULTADO:***  *O valor de tempo em minutos é 49.916666666666664* |
| --- |



**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0):*  *yn=0.*  *yn=y0*  *tn=t0*  *if(True):*  *while(yn>=1):*  *yn = yn - h\*(0.05)\*(yn)\*\*(2/3)*  *tn+=h*  *return tn*  *t0=0*  *y0=2*  *h=0.1*  *print("O valor de tempo da redução da amostra, em minutos, é", eulerEDO(h,y0,t0))*  ***RESULTADO:***  *O valor de tempo da redução da amostra, em minutos, é 15.59999999999996* |
| --- |



**Código implementado:**

| *import numpy*  *import math*  *def eulerEDO(h,y0,t0):*  *yn=0.*  *yn=y0*  *tn=t0*  *L=1000*  *if(True):*  *while(yn<=900):*  *yn = yn + h\*((0.1)\*((yn)\*\*(1/2)))\*(L-yn)/(L-yn)\*\*(1/2)*  *tn+=h*  *return tn*  *t0=0*  *y0=100*  *h=0.05*  *print("O valor de saturação para essa população é dada pelo valor L, visto que: L - População != 0")*  *print("O valor de tempo de 90% do total L da população, em minutos, é", eulerEDO(h,y0,t0))*  ***RESULTADO:***  *O valor de saturação para essa população é dada pelo valor L, visto que: L - População != 0*  *O valor de tempo de 90% do total L da população, em minutos, é 18.55000000000013* |
| --- |