Radioactive Decay (SKP)

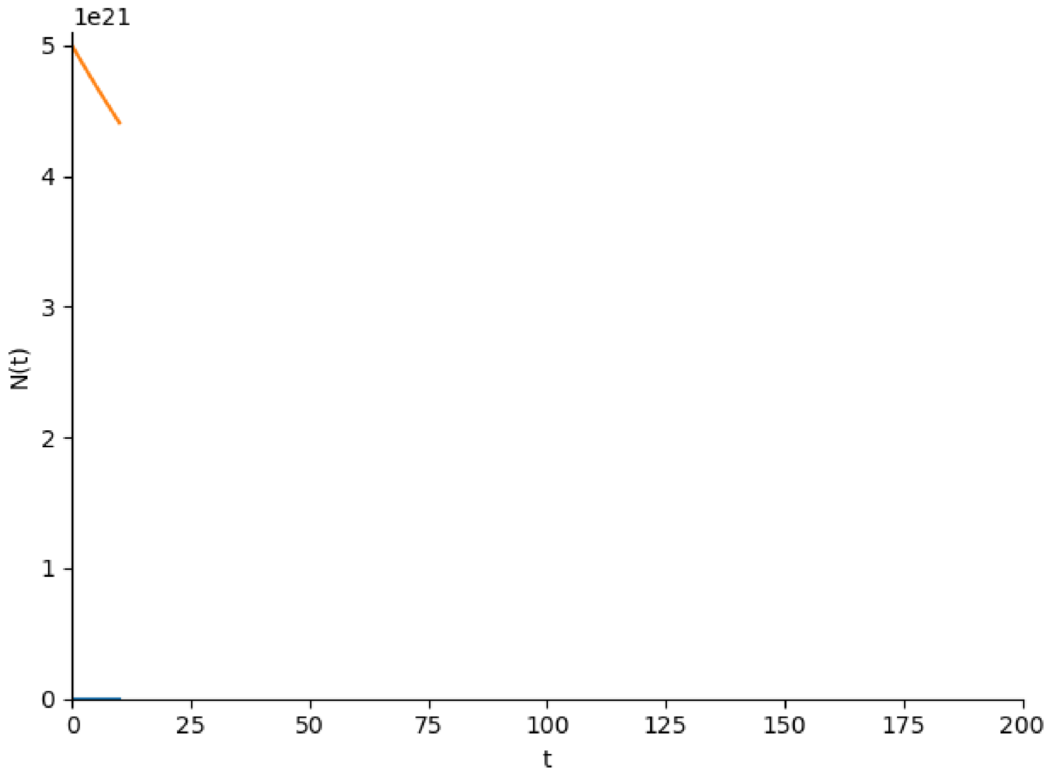
In [1]:
```
import sympy as smp
from sympy import *

t, lam, N0 = smp.symbols('t \lambda N_0', real=True, positive=True)
N = smp.symbols('N', cls=smp.Function)
N = N(t)
eq = Eq(N.diff(t), -lam*N)
display(eq)
Nsol = dsolve(eq, ics={N.subs(t,0):N0})
print('The solution of the differential equation is,')
display(Nsol)
```

$$\frac{d}{dt}N(t) = -\lambda N(t)$$

The solution of the differential equation is,

$$N(t) = N_0 e^{-\lambda t}$$

In [2]:
```
N00 = 5e21  # initial numbers of nuclei
t12 = 55  # half life
lam0 = 0.693/t12
Nplot = Nsol.subs([(lam,lam0), (N0,N00)])
smp.plot(t,Nplot.rhs,xlim=(0,200), ylim=(0,5.1e21), ylabel='N(t)')
```



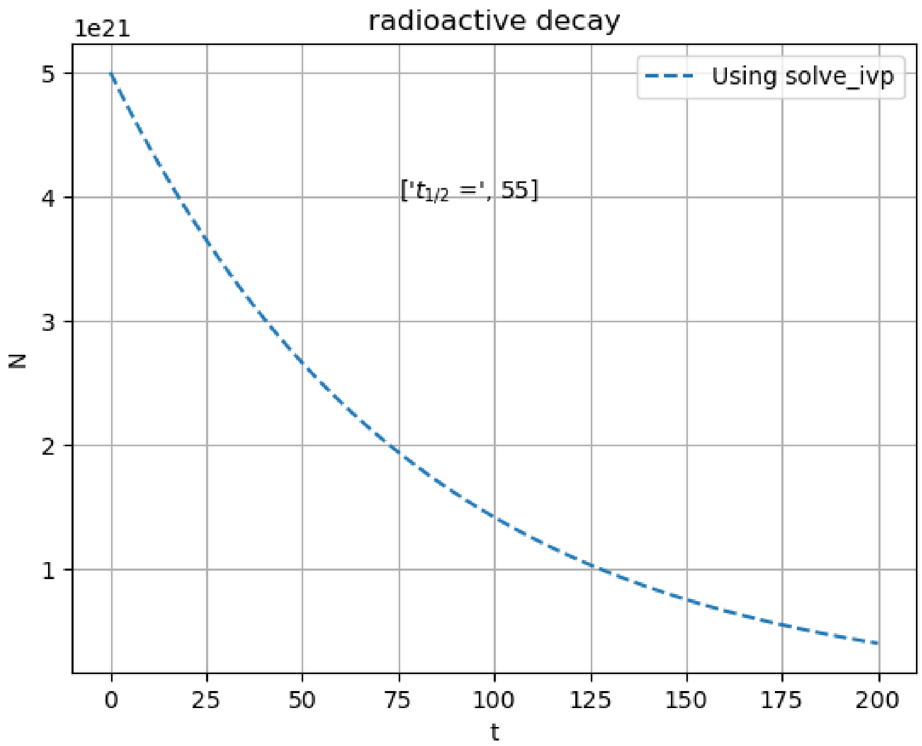
Out[2]: <sympy.plotting.plot.Plot at 0x1927a44dee0>

In [3]:
```
Nplot.rhs
```

Out[3]: $5.0 \cdot 10^{21} e^{-0.0126t}$

In [4]:
```python
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.integrate import odeint
from scipy.integrate import solve_ivp

N0 = 5e21  # initial numbers of nuclei
t12 = 55  # half life
lam = 0.693/t12

# Write the differential equation (t=x, N=y).
def dydx(x,y):
    return -lam*y

x_0, y_0 = 0, N0          # initial condition
x_min, x_max = x_0, 200    # lower and upper limit of x
dx = (x_max-x_0)/1000     # infinitesimal length
```

In [5]:
```python
# Using solve_ivp
y0 = y_0
x = np.linspace(x_min, x_max,500)
sol = solve_ivp(dydx, t_span=(min(x), max(x)), y0=[y0], t_eval=x)
y1 = sol.y[0]
plt.plot(x,y1, '--', label='Using solve_ivp')

plt.xlabel('t')
plt.ylabel('N')
plt.legend()
plt.title('radioactive decay')
plt.text(75,4e21, ["$t_{1/2}$ =", t12])
plt.grid()
plt.show()
```




```python
# ALL IN ONE PLOT
# Using odeint
y0 = y_0
x = np.linspace(x_min, x_max,500)
sol = odeint(dydx, y0=y0, t=x, tfirst=True)
y1 = sol.T[0]
plt.plot(x,y1, '--', label='Using odeint')

# Using solve_ivp
```

```
y0 = y_0
x = np.linspace(x_min, x_max,500)
sol = solve_ivp(dydx, t_span=(min(x), max(x)), y0=[y0], t_eval=x)
y1 = sol.y[0]
plt.plot(x,y1, '--', label='Using solve_ivp')

# Euler's Method
x, y = x_0, y_0
xmax = x_max
h = dx
xx, yy = [], []
while abs(x) < abs(xmax):
    xx.append(x)
    yy.append(y)
    x += h
    y += h*dydx(x,y)
plt.plot(xx,yy, '--', label='Euler\'s Method')

# Modified Euler's Method
x, y = x_0, y_0
xmax = x_max
h = dx
xx, yy = [], []
while abs(x) < abs(xmax):
    x += h
    dy = (h/2)*(dydx(x,y) + dydx(x + h, y + h*dydx(x,y)))
    y += dy
    xx.append(x), yy.append(y)
plt.plot(xx,yy, '--', label='Modified Euler\'s Method')

# Runge - Kutta Method
x, y = x_0, y_0
xmax = x_max
h = dx
xx, yy = [], []
while abs(x) < abs(xmax):
    xx.append(x), yy.append(y)
    x += h
    k1 = h * dydx(x,y)
    k2 = h * dydx(x + (h/2), y + (k1/2))
    k3 = h * dydx(x + (h/2), y + (k2/2))
    k4 = h * dydx(x + h, y + k3)
    y += (1/6)*(k1 + 2*(k2 + k3) + k4)
plt.plot(xx,yy, '--', label='Runge - Kutta Method')

plt.xlabel('t')
plt.ylabel('N')
plt.legend()
plt.title('radioactive decay')
plt.grid()
plt.show()
```

Successive Radioactive Decay

By radioactive decay, A is converted into B, B is converted into C and C is stable. Number of nuclei of A, B, C are N_{10}, N_{20}, N_{30} initially and N_1, N_2, N_3 at any time respectively. Decay constant of A and B are λ_1, λ_2 respectively.

In [6]:
```
import sympy as smp
from sympy import *
```

In [7]:
```
t, lam1, lam2, N10, N20, N30, N0 = smp.symbols('t \lambda_1 \lambda_2 N_{10} N_{20} N_{30}
                                              real=True, positive=True)
N1, N2, N3 = smp.symbols('N_1 N_2 N_3', cls=smp.Function)
N1 = N1(t)
N2 = N2(t)
N3 = N3(t)
```

In [8]:
```
eq1 = Eq(N1.diff(t), -lam1*N1)
eq2 = Eq(N2.diff(t), lam1*N1-lam2*N2)
eq3 = Eq(N3.diff(t), lam2*N2)
display(eq1, eq2, eq3)
```

$$\frac{d}{dt}\mathrm{N}_1(t) = -\lambda_1\mathrm{N}_1(t)$$

$$\frac{d}{dt}\mathrm{N}_2(t) = \lambda_1\mathrm{N}_1(t) - \lambda_2\mathrm{N}_2(t)$$

$$\frac{d}{dt}\mathrm{N}_3(t) = \lambda_2\mathrm{N}_2(t)$$

In [9]:
```
N1sol = dsolve(eq1, ics={N1.subs(t,0):N10})
N2sol = dsolve(eq2.subs(N1,N1sol.rhs), ics={N2.subs(t,0):N20})
N3sol = dsolve(eq3.subs(N2,N2sol.rhs), ics={N3.subs(t,0):N30})
display(N1sol, N2sol, N3sol)
```

$$\mathrm{N}_1(t) = N_{10}e^{-\lambda_1 t}$$

$$\mathrm{N}_2(t) = -\frac{N_{10}\lambda_1 e^{-\lambda_1 t}}{\lambda_1 - \lambda_2} + \frac{(N_{10}\lambda_1 + N_{20}\lambda_1 - N_{20}\lambda_2)\,e^{-\lambda_2 t}}{\lambda_1 - \lambda_2}$$

$$\mathrm{N}_3(t) = N_{10} + N_{20} + N_{30} + \frac{\lambda_2\left(-\frac{N_{10}\lambda_1 e^{-\lambda_2 t}}{\lambda_2} + N_{10}e^{-\lambda_1 t} - \frac{N_{20}\lambda_1 e^{-\lambda_2 t}}{\lambda_2} + N_{20}e^{-\lambda_2 t}\right)}{\lambda_1 - \lambda_2}$$

If of B and C are initially absent and $N_{10} = N_0$,

In [10]:
```
N1sol1 = N1sol.subs(N10,N0)
N2sol1 = N2sol.subs([(N10,N0), (N20,0)])
N3sol1 = N3sol.subs([(N10,N0), (N20,0), (N30,0)])
```

In [11]:
```
display(N1sol1, N2sol1, N3sol1)
```

$$\mathrm{N}_1(t) = N_0 e^{-\lambda_1 t}$$

$$\mathrm{N}_2(t) = \frac{N_0\lambda_1 e^{-\lambda_2 t}}{\lambda_1 - \lambda_2} - \frac{N_0\lambda_1 e^{-\lambda_1 t}}{\lambda_1 - \lambda_2}$$

$$\mathrm{N}_3(t) = N_0 + \frac{\lambda_2\left(-\frac{N_0\lambda_1 e^{-\lambda_2 t}}{\lambda_2} + N_0 e^{-\lambda_1 t}\right)}{\lambda_1 - \lambda_2}$$

Graphs for a given half-lives and initial numbers of nuclei:

In [12]:
```
t121, t122 = 1.6, 0.9
N00 = 2e20
```

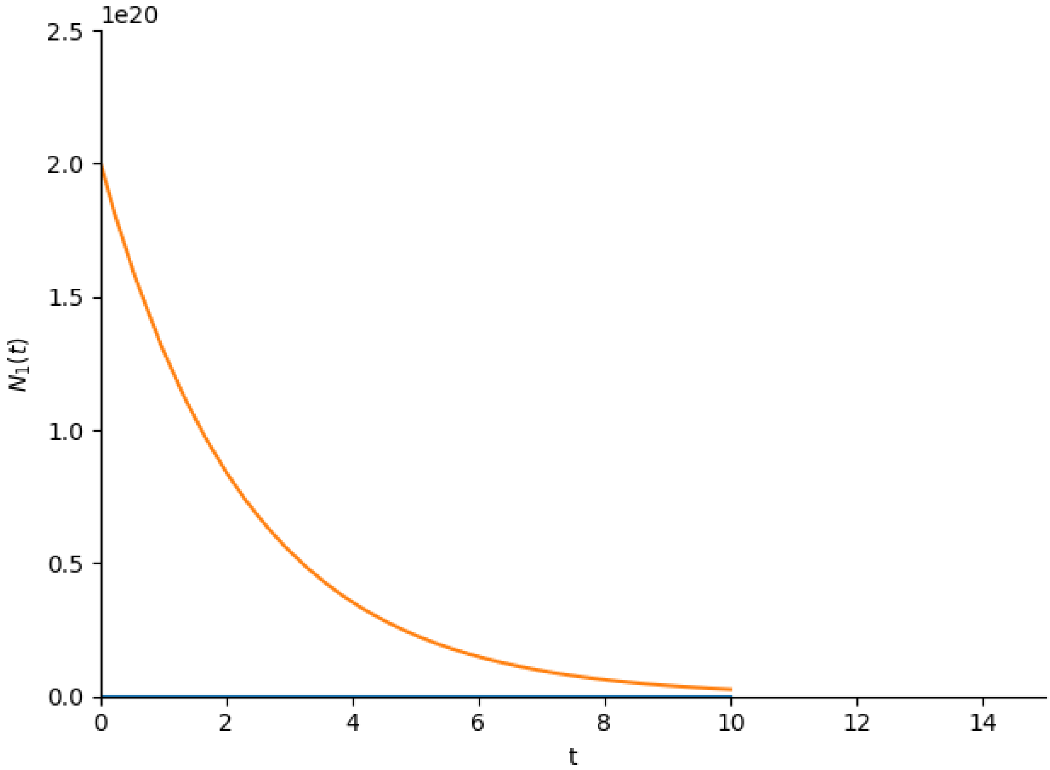
In [13]:
```
lam10 = (0.693/t121)
lam20 = (0.693/t122)
N1plot = N1sol1.subs([(lam1,lam10),(N0,N00)])
N2plot = N2sol1.subs([(lam1,lam10), (lam2,lam20), (N0,N00)])
N3plot = N3sol1.subs([(lam1,lam10), (lam2,lam20), (N0,N00)])
display(N1plot, N2plot, N3plot)
```

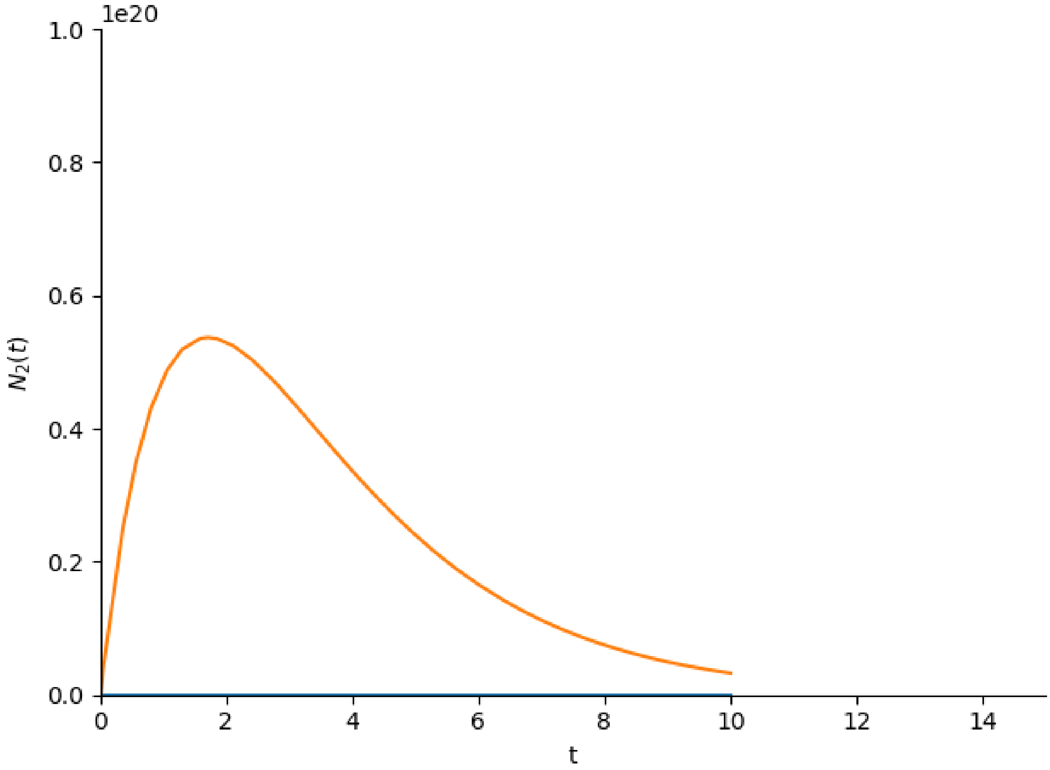
$$N_1(t) = 2.0 \cdot 10^{20} e^{-0.433125t}$$

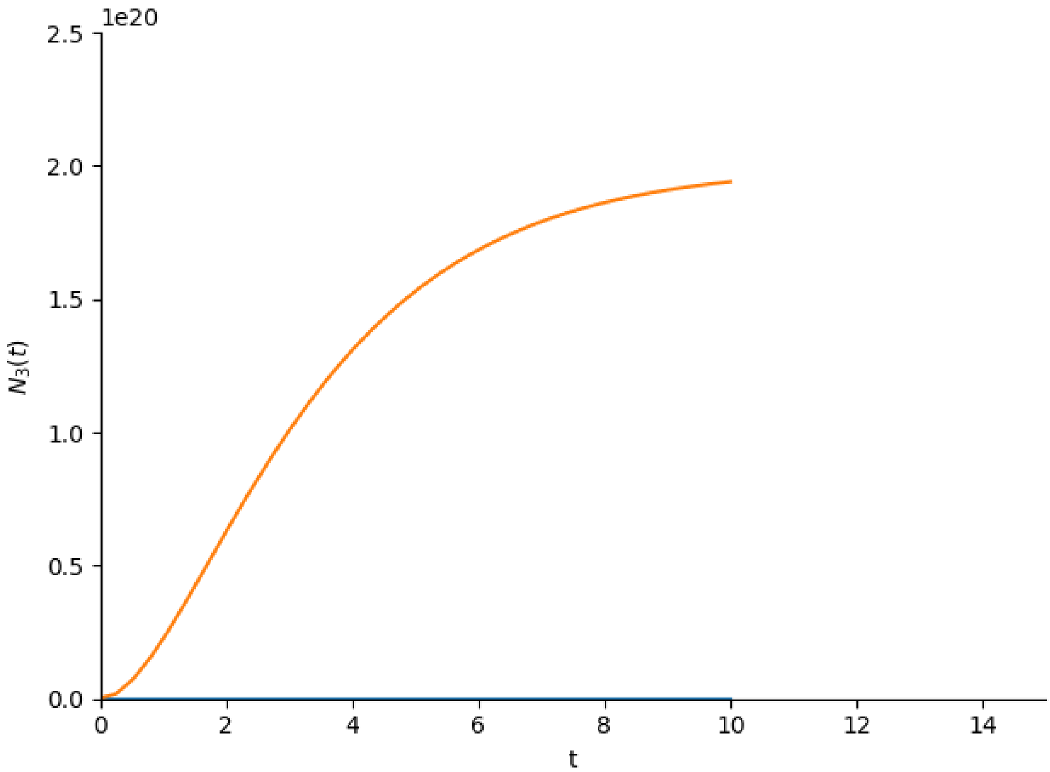
$$N_2(t) = -2.57142857142857 \cdot 10^{20} e^{-0.77t} + 2.57142857142857 \cdot 10^{20} e^{-0.433125t}$$

$$N_3(t) = 2.0 \cdot 10^{20} + 2.57142857142857 \cdot 10^{20} e^{-0.77t} - 4.57142857142857 \cdot 10^{20} e^{-0.433125t}$$

In [14]:
```
smp.plot(t, N1plot.rhs, xlim=(0,15), ylim=(0, 2.5e20), ylabel='$N_1(t)$')
smp.plot(t, N2plot.rhs, xlim=(0,15), ylim=(0, 1e20), ylabel='$N_2(t)$')
smp.plot(t, N3plot.rhs, xlim=(0,15), ylim=(0, 2.5e20), ylabel='$N_3(t)$')
```



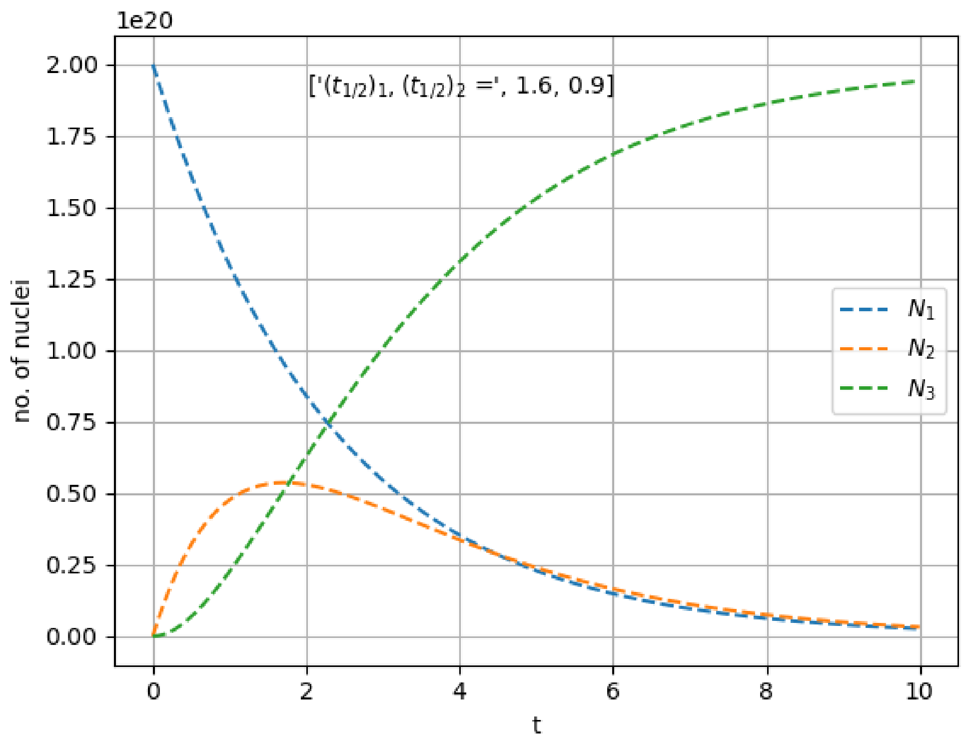




Out[14]: <sympy.plotting.plot.Plot at 0x192021c1850>

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.integrate import odeint
from scipy.integrate import solve_ivp

t121, t122 = 1.6, 0.9  # half lives
N00 = 2e20      # initial numbers of nuclei
lam1, lam2 = 0.693/t121, 0.693/t122

# Write the differential equation. (t=x, N1=y, N2=yp, N3=ypp)

def dSdx(x,S):
    y, yp, ypp = S
    return [-lam1*y, +lam1*y -lam2*yp, +lam2*yp]

def dydx(x,y,yp,ypp):
    return -lam1*y
def dypdx(x,y,yp,ypp):
    return +lam1*y -lam2*yp
def dyppdx(x,y,yp,ypp):
    return +lam2*yp

x_0, y_0, yp_0, ypp_0 = 0, N00, 0, 0  # initial conditions
x_min, x_max = x_0, 10     # lower and upper limit of x
dx = (x_max-x_0)/1000    # infinitesimal length
```

In [16]:
```
# Using solve_ivp
y0, yp0, ypp0 = y_0, yp_0, ypp_0
S0 = (y0,yp0,ypp0)
x = np.linspace(x_min, x_max,200)
sol = solve_ivp(dSdx, t_span=(min(x), max(x)), y0=S0, t_eval=x)
y1 = sol.y[0]
y2 = sol.y[1]
y3 = sol.y[2]
plt.plot(x,y1, '--', label='$N_1$')
plt.plot(x,y2, '--', label='$N_2$')
plt.plot(x,y3, '--', label='$N_3$')
plt.xlabel('t')
plt.ylabel('no. of nuclei')
plt.text(2,1.9e20, ["$(t_{1/2})_1, (t_{1/2})_2$ =", t121, t122])
plt.legend()
plt.grid()
plt.show()
```



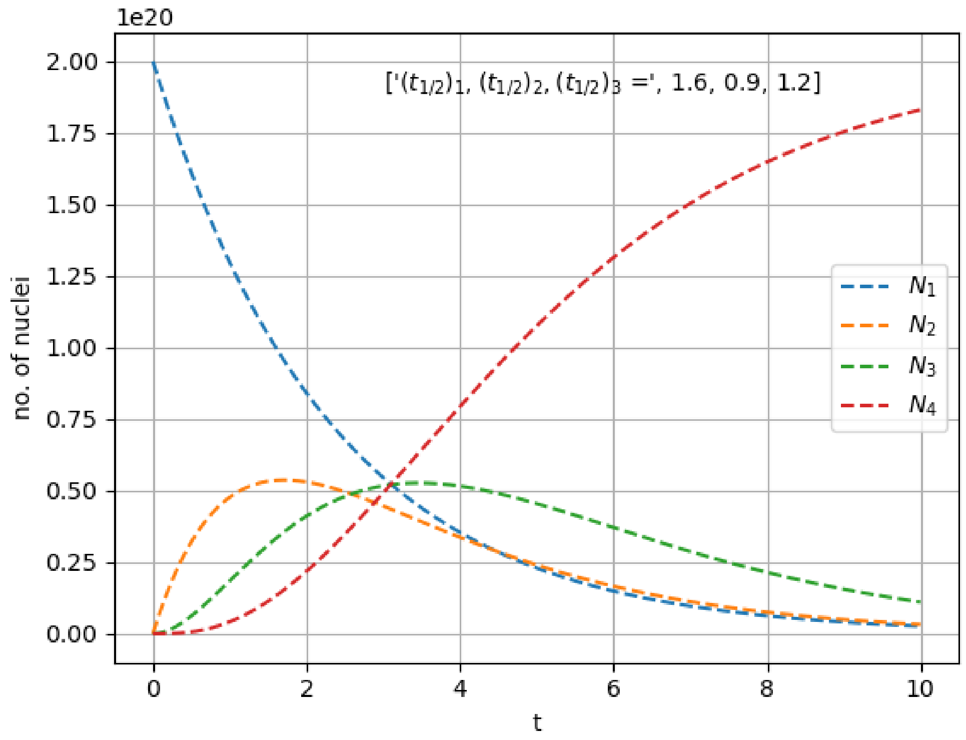

```
# ALL IN ONE PLOT
# Using odeint
y0, yp0, ypp0 = y_0, yp_0, ypp_0
S0 = (y0,yp0,ypp0)
x = np.linspace(x_min, x_max,200)
sol = odeint(dSdx, y0=S0, t=x, tfirst=True)
y1 = sol.T[0]
plt.plot(x,y1, '--', label='Using odeint')

# Using solve_ivp
y0, yp0, ypp0 = y_0, yp_0, ypp_0
S0 = (y0,yp0,ypp0)
x = np.linspace(x_min, x_max,200)
sol = solve_ivp(dSdx, t_span=(min(x), max(x)), y0=S0, t_eval=x)
y1 = sol.y[0]
plt.plot(x,y1, '--', label='Using solve_ivp')

# Euler's Method
x, y, yp, ypp = x_0, y_0, yp_0, ypp_0
xmax = x_max
h = dx
xx, yy, yyp, yypp = [], [], [], []
while abs(x) < abs(xmax):
    xx.append(x)
    yy.append(y)
```

```
    yyp.append(yp)
    yypp.append(ypp)
    x += h
    y += h*dydx(x,y,yp,ypp)
    yp += h*dypdx(x,y,yp,ypp)
    ypp += h*dyppdx(x,y,yp,ypp)
plt.plot(xx,yy, '--', label='Euler\'s Method')

# Modified Euler's Method
x, y, yp, ypp = x_0, y_0, yp_0, ypp_0
xmax = x_max
h = dx
xx, yy, yyp, yypp = [], [], [], []
while abs(x) < abs(xmax):
    xx.append(x)
    yy.append(y)
    yyp.append(yp)
    yypp.append(ypp)
    x += h
    dy = (h/2) * (dydx(x,y,yp,ypp) +
dydx(x+h, y+h*dydx(x,y,yp,ypp), yp+h*dypdx(x,y,yp,ypp), ypp+h*dyppdx(x,y,yp,ypp)))
    dyp = (h/2) * (dypdx(x,y,yp,ypp) +
dypdx(x+h, y+h*dydx(x,y,yp,ypp), yp+h*dypdx(x,y,yp,ypp), ypp+h*dyppdx(x,y,yp,ypp)))
    dypp = (h/2) * (dyppdx(x,y,yp,ypp) +
dyppdx(x+h, y+h*dydx(x,y,yp,ypp), yp+h*dypdx(x,y,yp,ypp), ypp+h*dyppdx(x,y,yp,ypp)))
    y += dy
    yp += dyp
    ypp += dypp
plt.plot(xx,yy, '--', label='Modified Euler\'s Method')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

Successive Radioactive Decay (A, B, C, D)

In [17]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.integrate import odeint
from scipy.integrate import solve_ivp

t121, t122, t123 = 1.6, 0.9, 1.2  # half lives
N00 = 2e20     # initial numbers of nuclei
lam1, lam2, lam3 = 0.693/t121, 0.693/t122, 0.693/t123

# Write the differential equation. (t=x, N1=y, N2=yp, N3=ypp)

def dSdx(x,S):
    y, yp, ypp, yppp = S
    return [-lam1*y, +lam1*y -lam2*yp, +lam2*yp -lam3*ypp, +lam3*ypp]

def dydx(x,y,yp,ypp,yppp):
    return -lam1*y
def dypdx(x,y,yp,ypp,yppp):
    return +lam1*y -lam2*yp
def dyppdx(x,y,yp,ypp,yppp):
    return +lam2*yp -lam3*ypp
def dypppdx(x,y,yp,ypp,yppp):
    return +lam3*ypp

x_0,y_0,yp_0,ypp_0,yppp_0 = 0,N00,0,0,0  # initial conditions
x_min, x_max = x_0, 10   # lower and upper limit of x
dx = (x_max-x_0)/1000    # infinitesimal length
```

In [18]:
```python
# Using solve_ivp
y0, yp0, ypp0, yppp0 = y_0, yp_0, ypp_0, yppp_0
S0 = (y0,yp0,ypp0,yppp0)
x = np.linspace(x_min, x_max,200)
sol = solve_ivp(dSdx, t_span=(min(x), max(x)), y0=S0, t_eval=x)
y1 = sol.y[0]
y2 = sol.y[1]
y3 = sol.y[2]
y4 = sol.y[3]
plt.plot(x,y1, '--', label='$N_1$')
plt.plot(x,y2, '--', label='$N_2$')
plt.plot(x,y3, '--', label='$N_3$')
plt.plot(x,y4, '--', label='$N_4$')
plt.text(3,1.9e20, ['$(t_{1/2})_1, (t_{1/2})_2, (t_{1/2})_3$ =', t121, t122, t123])
plt.xlabel('t')
plt.ylabel('no. of nuclei')
plt.legend()
plt.grid()
plt.show()
```




```python
# ALL IN ONE CODE
# Using odeint
y0, yp0, ypp0, yppp0 = y_0, yp_0, ypp_0, yppp_0
S0 = (y0,yp0,ypp0,yppp0)
x = np.linspace(x_min, x_max,200)
sol = odeint(dSdx, y0=S0, t=x, tfirst=True)
y1 = sol.T[0]
plt.plot(x,y1, '--', label='Using odeint')

# Using solve_ivp
y0, yp0, ypp0, yppp0 = y_0, yp_0, ypp_0, yppp_0
S0 = (y0,yp0,ypp0,yppp0)
x = np.linspace(x_min, x_max,200)
sol = solve_ivp(dSdx, t_span=(min(x), max(x)), y0=S0, t_eval=x)
y1 = sol.y[0]
plt.plot(x,y1, '--', label='Using solve_ivp')

# Euler's Method
x, y, yp, ypp, yppp = x_0, y_0, yp_0, ypp_0, yppp_0
xmax = x_max
h = dx
xx, yy, yyp, yypp, yyppp = [], [], [], [], []
while abs(x) < abs(xmax):
```

```python
    xx.append(x)
    yy.append(y)
    yyp.append(yp)
    yypp.append(ypp)
    yyppp.append(yppp)
    x += h
    y += h*dydx(x,y,yp,ypp,yppp)
    yp += h*dypdx(x,y,yp,ypp,yppp)
    ypp += h*dyppdx(x,y,yp,ypp,yppp)
    yppp += h*dypppdx(x,y,yp,ypp,yppp)
plt.plot(xx,yy, '--', label='Euler\'s Method')

# Modified Euler's Method
x, y, yp, ypp, yppp = x_0, y_0, yp_0, ypp_0, yppp_0
xmax = x_max
h = dx
xx, yy, yyp, yypp, yyppp = [], [], [], [], []
while abs(x) < abs(xmax):
    xx.append(x)
    yy.append(y)
    yyp.append(yp)
    yypp.append(ypp)
    yyppp.append(yppp)
    x += h
    dy = (h/2) * (dydx(x,y,yp,ypp,yppp) +
dydx(x+h, y+h*dydx(x,y,yp,ypp,yppp), yp+h*dypdx(x,y,yp,ypp,yppp),
ypp+h*dyppdx(x,y,yp,ypp,yppp), yppp+h*dypppdx(x,y,yp,ypp,yppp)))
    dyp = (h/2) * (dypdx(x,y,yp,ypp,yppp) +
dypdx(x+h, y+h*dydx(x,y,yp,ypp,yppp), yp+h*dypdx(x,y,yp,ypp,yppp),
ypp+h*dyppdx(x,y,yp,ypp,yppp), yppp+h*dypppdx(x,y,yp,ypp,yppp)))
    dypp = (h/2) * (dyppdx(x,y,yp,ypp,yppp) +
dyppdx(x+h, y+h*dydx(x,y,yp,ypp,yppp), yp+h*dypdx(x,y,yp,ypp,yppp),
ypp+h*dyppdx(x,y,yp,ypp,yppp), yppp+h*dypppdx(x,y,yp,ypp,yppp)))
    dyppp = (h/2) * (dypppdx(x,y,yp,ypp,yppp) +
dypppdx(x+h, y+h*dydx(x,y,yp,ypp,yppp), yp+h*dypdx(x,y,yp,ypp,yppp),
ypp+h*dyppdx(x,y,yp,ypp,yppp), yppp+h*dypppdx(x,y,yp,ypp,yppp)))
    y += dy
    yp += dyp
    ypp += dypp
    yppp += dyppp
plt.plot(xx,yy, '--', label='Modified Euler\'s Method')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

In [ ]: