Group 9

Yiling Chen, Woosung Lee, Jae Hyeok Kwak

In [117]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import sympy as sp
import scipy.interpolate
from resources306 import*
```

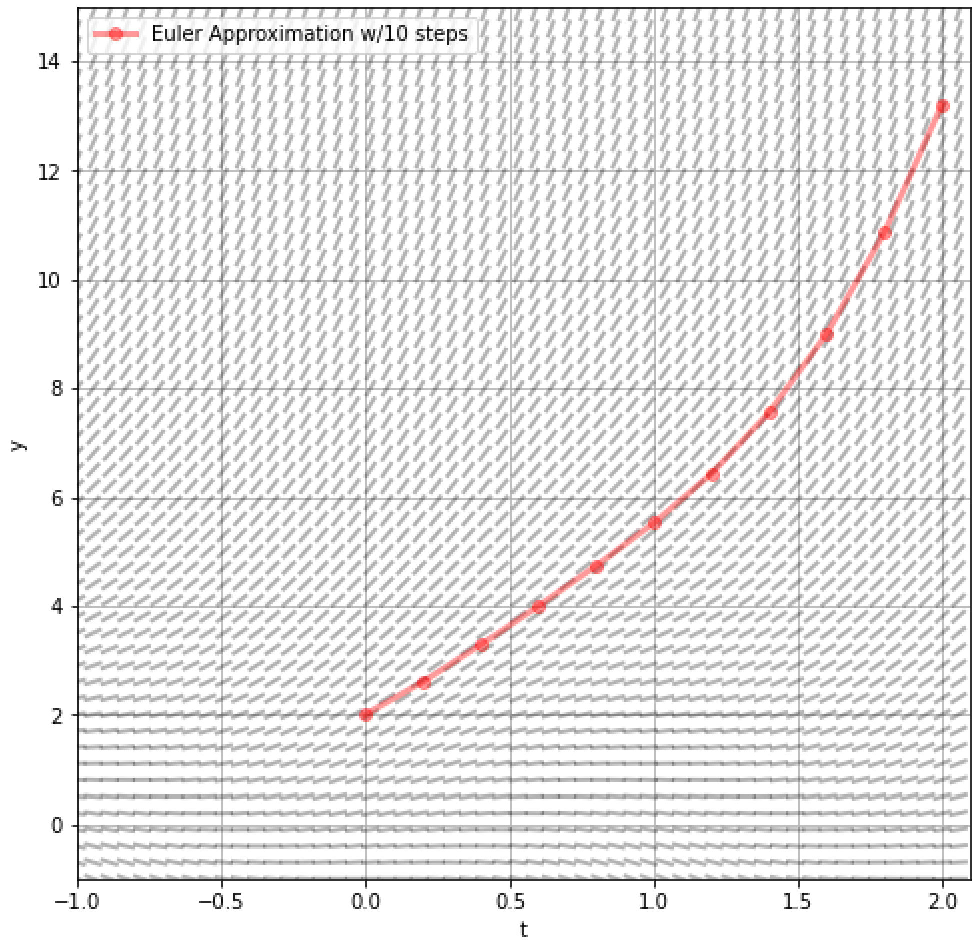
Problem 1a Euler's Method

$y'(f(t,y), a < t < b, y(a) = \alpha,\ w_0 = \alpha,\ w_{i+1} = w_i + h * f(t_i, w_i), for\ each\ i = 0, 1 ... N-1$

$h = \frac{(b-a)}{N}, t_i = a + i * h$

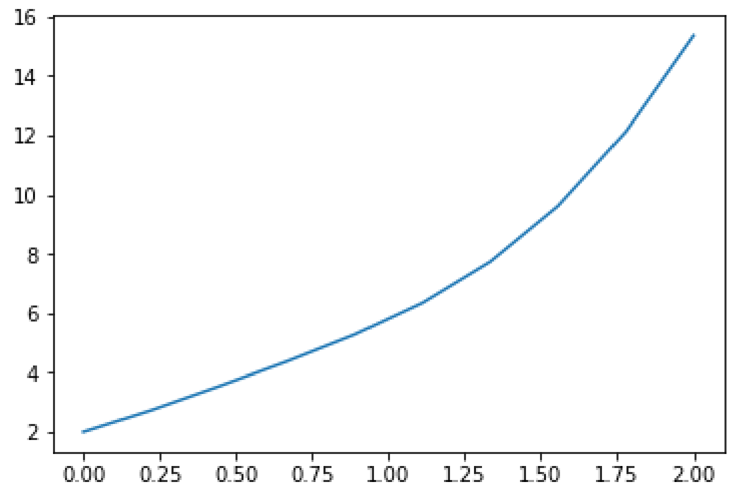
In [118]:
```
def euler_method(f,t,y,n,t_final):
    delta_t = (t_final-t)/n #This is our step size

    tlist = [t] #initialize lists in which to store the data
    ylist = [y] #for later plotting

    for i in np.arange(n): #Iterate of our steps
        slope = f(t,y) #Compute the slope at the current location
        y = y+slope*delta_t #compute the new y value
        t = t + delta_t #Advance in time by one step
        tlist.append(t) #Add the new t and y values at the ends of the list
        ylist.append(y)
    return tlist,ylist
```

In [120]:
```
y=2
t=0
t_final=2
n=10
def f_1(t,y):
    f=y+np.cos(np.pi*t)
    return f

[tlist,ylist]=euler_method(f_1,t,y,n,t_final)
plt.figure(figsize=(8,8))
slopefieldplot(f_1, -1,2.1,-1,15,0.3, lw=2)
plt.plot(tlist,ylist, 'ro-', lw=3, alpha=0.4, label ='Euler Approximation w
plt.xlabel('t');
plt.ylabel('y');
plt.grid();
plt.legend(loc='upper left');
```



In [121]:
```
t = np.linspace(0,2,10)
y_exact = ((2*np.pi**2+3)*np.exp(t)+np.pi*np.sin(np.pi*t)-np.cos(np.pi*t))/
plt.plot(t,y_exact)
```

Out[121]: [<matplotlib.lines.Line2D at 0x7fa0f0bfebb0>]



In [135]:
```
Err1 = [abs (y_exact[i] - ylist[i]) for i in np.arange(10)]
Err1
```

Out[135]: [0.0, 0.1278975839187968, 0.2495874018253934, 0.37099473451683274, 0.53577961581
0.8185470536021597, 1.3032655736051693, 2.057958085760581, 3.1199834008490335,

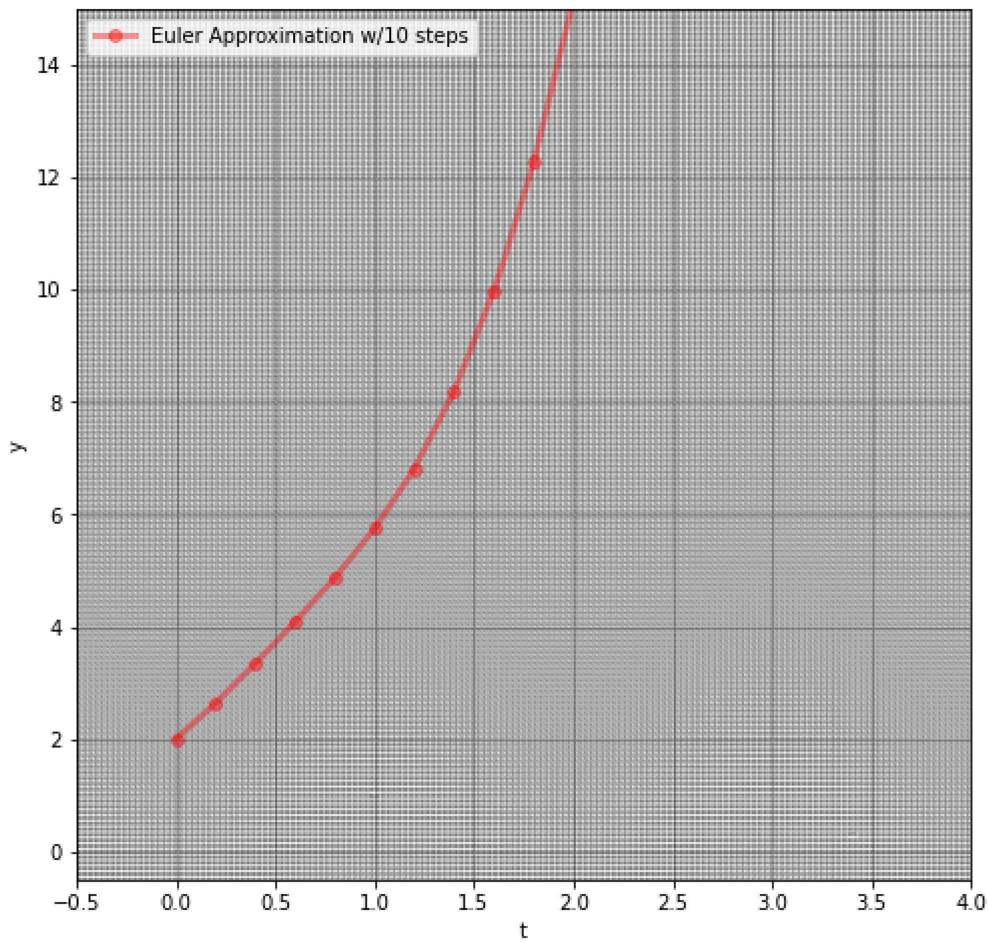
Problem 1b Modified Euler's Method

$y'(f(t,y), a <= t <= b, y(a) = \alpha, w_0 = \alpha,$
$w_{i+1} = w_i + \frac{h}{2} * f(t_i, w_i) + f(t_{i+1}, w_i + h * f(t_i, w_i), for\ each\ i = 0, 1 ... N-1$

$h = \frac{(b-a)}{N}, t_i = a + i * h$

In [123]:
```
def mod_euler(f,t,y,n,t_final):
    h=(t_final-t)/n
    tlist = [t] #Initialize lists in which to store the data
    ylist = [y] #for later plotting

    for i in np.arange(n):
        y=y+(h/2)*(f(t,y)+f(t+h,y+h*f(t,y)))
        t=t+h
        tlist.append(t) #add the new t and y values at the ends of the list
        ylist.append(y)
    return tlist,ylist
```

In [125]:
```
y=2
def f_1(t,y):
    f=y+np.cos(np.pi*t)
    return f
t=0
t_final=2
n=10
[tlist1,ylist1]=mod_euler(f_1,t,y,n,t_final)
plt.figure(figsize=(8,8))
slopefieldplot(f_1, -0.5,4,-0.5,15,0.1, lw=2)
plt.plot(tlist1,ylist1, 'ro-', lw=3, alpha=0.4, label ='Euler Approximation
plt.xlabel('t');
plt.ylabel('y');
plt.grid();
plt.legend(loc='upper left');
```



In [126]:
```
Err2 = [abs (y_exact[i] - ylist1[i]) for i in np.arange(10)]
Err2
```

Out[126]: [0.0, 0.08699588448130191, 0.1815069886241507, 0.27792362142130767, 0.3984138481
0.5914763060488237, 0.9166612472253739, 1.423905490902067, 2.1397614562364584,

Problem 1c Midpoint method

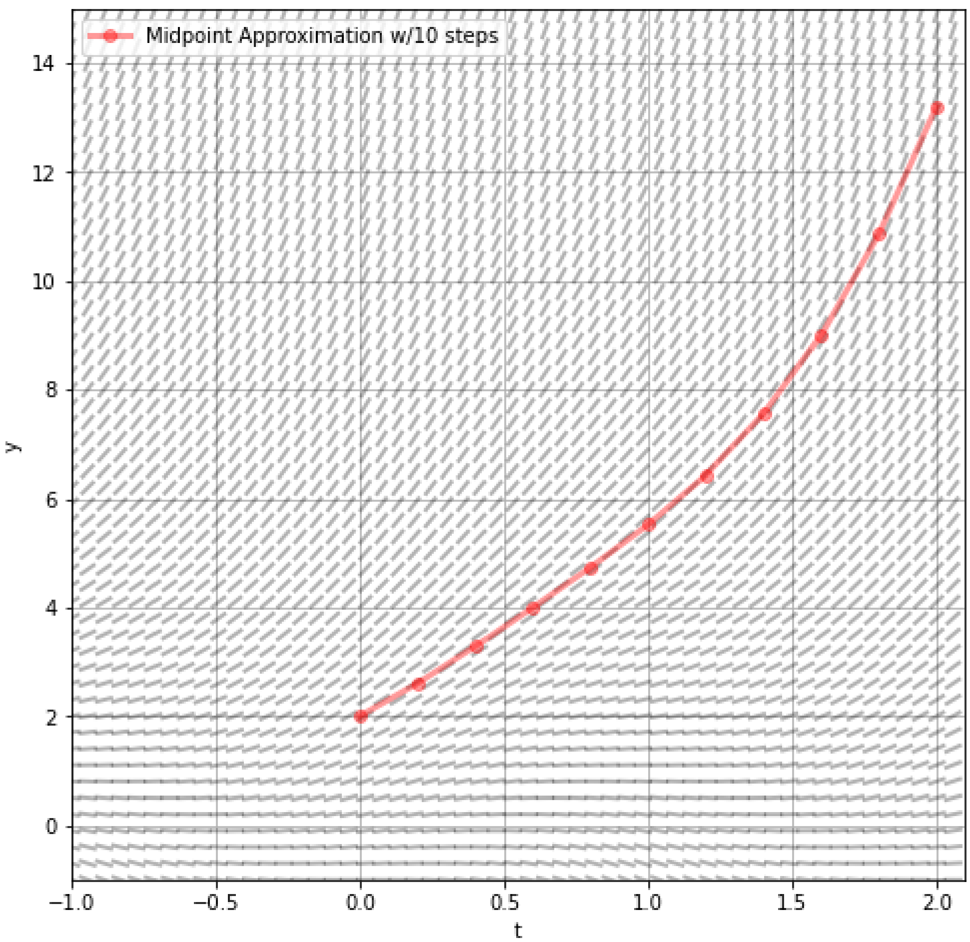
$y'(f(t,y), a <= t <= b, y(a) = \alpha, w_0 = \alpha,$

$$w_{i+1} = w_i + h * f(t_i + \frac{h}{2}, w_i + \frac{h}{2} * f(t_i, w_i), for\ each\ i = 0, 1 \ldots N-1$$

$$h = \frac{(b-a)}{N}, t_i = a + i * h$$

In [127]:
```
def Midpt_method(f,t,y,n,t_final):
    delta_t = (t_final-t)/n #This is our step size

    tlist = [t] #initialize lists in which to store the data
    ylist = [y] #for later plotting

    for i in range(n): #Iterate of our steps
        slope = f(t,y) #Compute the slope at the current location
        y = y+delta_t*(f(t+(h/2),y+(h/2)*slope)) #compute the new y value
        t = t + delta_t #Advance in time by one step
        tlist.append(t) #Add the new t and y values at the ends of the list
        ylist.append(y)
    return tlist,ylist
```

In [129]:
```
y=2
def f_1(t,y):
    f=y+np.cos(np.pi*t)
    return f
t=0
t_final=2
n=10
[tlist3,ylist3]=Midpt_method(f_1,t,y,n,t_final)
plt.figure(figsize=(8,8))
slopefieldplot(f_1, -1,2.1,-1,15,0.3, lw=2)
plt.plot(tlist,ylist, 'ro-', lw=3, alpha=0.4, label ='Midpoint Approximatio
plt.xlabel('t');
plt.ylabel('y');
plt.grid();
plt.legend(loc='upper left');
```



In [130]:
```
Err3 = [abs (y_exact[i] - ylist3[i]) for i in np.arange(10)]
Err3
```

Out[130]: [0.0, 0.07768628065976602, 0.16439562037837163, 0.2570477521614567, 0.3786989392
0.5767337209958745, 0.9079848972823115, 1.4190739955550349, 2.133867031913079,

Problem1d Runge-Kutta method of order four

$y'(f(t,y), a <= t <= b, y(a) = \alpha, w_0 = \alpha, k_1 = h * f(t,w); k_2 = h * f(t + \frac{h}{2}, w + \frac{k_1}{2});$

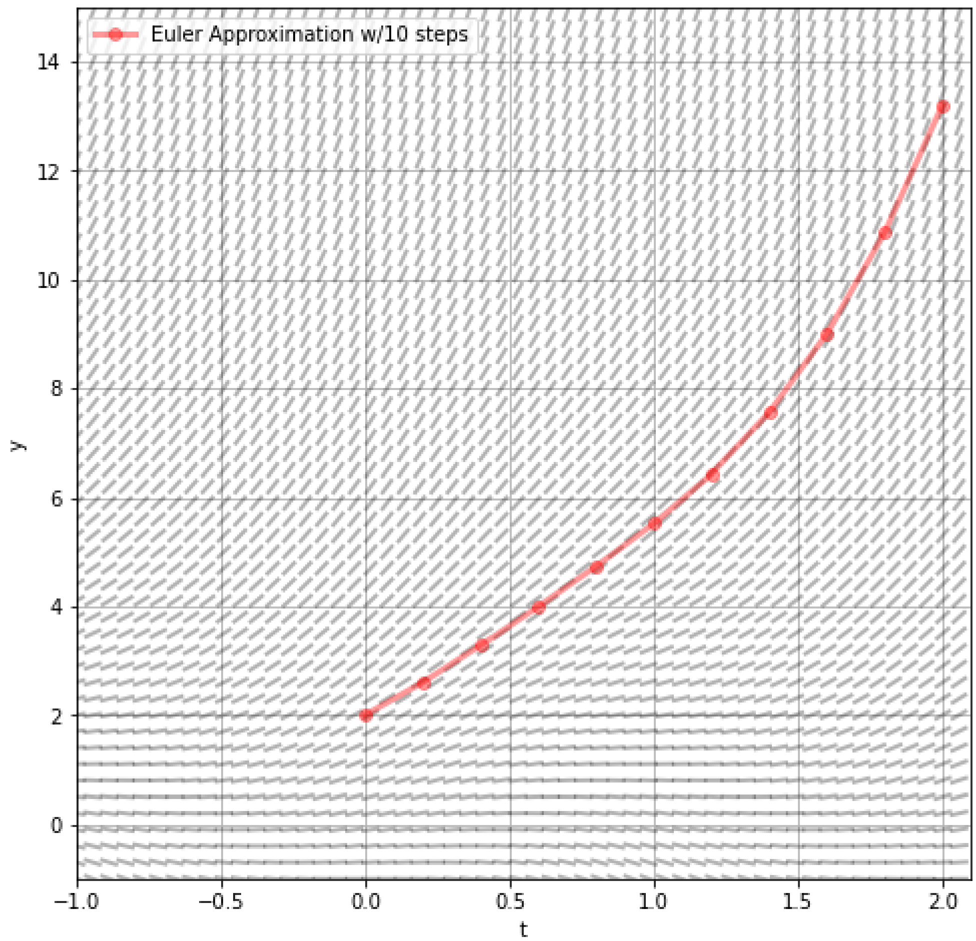
$k_3 = h * f(t + \frac{h}{2}, w + \frac{k_2}{2})$; $k_4 = h * f(t + h, w + k_3)$; $w = w + h * \frac{k_1 + 2*k_2 + 2*k_3 + k_4}{6}$, for $each\ i = 0, 1 ... N - 1$

$h = \frac{(b-a)}{N}, t_i = a + i * h$

In [131]:

```
def RK_method(f,t,y,n,t_final):
    delta_t = (t_final-t)/n #This is our step size

    tlist = [t] #initialize lists in which to store the data
    ylist = [y] #for later plotting

    for i in range(n): #Iterate of our steps
        slope = f(t,y) #Compute the slope at the current location
        k1 = slope*delta_t
        k2 = delta_t*f(t+(delta_t/2),y+(k1/2))
        k3 =delta_t*f(t+(delta_t/2),y+(k2/2))
        k4 =delta_t*f(t+delta_t,y+k3)
        y = y +(k1+(2*k2)+(2*k3)+k4)/6 #compute the new y value

        t = t + delta_t #Advance in time by one step
        tlist.append(t) #Add the new t and y values at the ends of the list
        ylist.append(y)
    return tlist,ylist
```

In [133]:
```
y=2
t=0
t_final=2
n=10
def f_1(t,y):
    f=y+np.cos(np.pi*t)
    return f

[tlist4,ylist4]=RK_method(f_1,t,y,n,t_final)
plt.figure(figsize=(8,8))
slopefieldplot(f_1, -1,2.1,-1,15,0.3, lw=2)
plt.plot(tlist,ylist, 'ro-', lw=3, alpha=0.4, label ='Euler Approximation w
plt.xlabel('t');
plt.ylabel('y');
plt.grid();
plt.legend(loc='upper left');
```



In [134]:
```
Err4 = [abs (y_exact[i] - ylist4[i]) for i in np.arange(10)]
Err4
```

Out[134]: [0.0, 0.0772746907054791, 0.16406511304184113, 0.25582389137729145, 0.3738621310
0.563990685656627, 0.8819600252414377, 1.373851023479876, 2.06326981157223, 2.9

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
import sympy as sp
import scipy.interpolate
from resources306 import*
```

Problem 2

m*(dv/dt) = -mg - k * sqrt(abs(v))

dv/dt = -g - (k/m) * sqrt(abs(v))

f(t,v) = -g - (k/m) * sqrt(abs(v))

Extrapolation method : Using extrapolation method, we can get high accuracy simply by predictable error, parameter, and step size h.

M - N1(h) = K1 * h + K2 * h^2 + K3 * h^3 + ...

M = N1(h/2) + K1 * (h/2) + K2 * (h^2/4) + K3 * (h^3/8) + ...

O(h^2) fomula for M:

M = N2(h) - (K2/2) * h^2 - (3 * K3)/4 * h^3 - ...

In [2]:
```
def extrapolate(f,a,b,y0,TOL,hmax,hmin):
    NK=[2,4,6,8,12,16,24,32]
    N=len(NK)
    Y=np.zeros(N)
    Q=np.zeros((N,N))
    T0=a
    W0=y0
    h=hmax
    FLAG=1
    for i in np.arange(N-1):
        for j in np.arange(i):
            Q[i,j]=( NK[i+1]/NK[j] )**2
    while FLAG==1:
        k=0
        NFLAG=0
        while (k<=N-1) and (NFLAG==0):
            HK=h/NK[k]
            T=T0
            W2=W0
            W3=W2+HK*f(T,W2)
            T=T0+HK
            for j in np.arange( NK[k]-1 ):
                W1=W2
                W2=W3
                W3=W1+2*HK*f(T,W2)
                T=T0+(j+1)*HK
            Y[k]=(W3+W2+HK*f(T,W3))/2

            if k>=1:
                j=k
                v=Y[0]

                while j>=1:
                    Y[j-1]=Y[j]+( Y[j]-Y[j-1] )/( Q[k-1,j-1]-1 )
                    j=j-1

                if abs(Y[0]-v)<=TOL:
                    NFLAG=1
            k=k+1
        k=k-1
        if NFLAG==0:
            h=h/2
            if h<hmin:
                print('min exceeded')
                FLAG=0
        else:
            W0=Y[0]
            T0=T0+h
            print(T0,W0,h)
            if T0>=b:
                FLAG=0
            elif T0+h>b:
                h=b-T0
            elif (k<=2) and (h<0.5*hmax):
                h=2*h
```

In [3]:
```
def f_2(t,v):
    g=9.8
    m=10
    k=0.5
    f=-g-(k/m)*np.sqrt(abs(v))
    return f
```

In [4]:
```python
a=0
b=5
v0=0
TOL=1e-4
hmax=0.25
hmin=0.05
extrapolate(f_2,a,b,v0,TOL,hmax,hmin)
```

```
0.25 -2.4618226723196095 0.25
0.5 -4.9356852417663735 0.25
0.75 -7.416668826704473 0.25
1.0 -9.903404643569619 0.25
1.25 -12.395108977711857 0.25
1.5 -14.89125471824358 0.25
1.75 -17.391455480834235 0.25
2.0 -19.895412144215662 0.25
2.25 -22.402884178878708 0.25
2.5 -24.91367267160583 0.25
2.75 -27.427609537068406 0.25
3.0 -29.94455028482495 0.25
3.25 -32.46436896233801 0.25
3.5 -34.98695449773224 0.25
3.75 -37.51220798004591 0.25
4.0 -40.040040588748454 0.25
4.25 -42.5703719857798 0.25
4.5 -45.10312904512663 0.25
4.75 -47.638244833922855 0.25
5.0 -50.175657784433064 0.25
```

In [5]:
```python
t = [0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, 2.25, 2.5, 2.75, 3, 3.25, 3.5, 3.75, 4, 4.25, 4.5, 4.75, 5]
v = [-2.4618226723196095, -4.9356852417663735, -7.416668826704473, -9.903404643569619, -12.395108977711857, -14.89125471824358,
    -17.391455480834235, -19.895412144215662, -22.402884178878708, -24.91367267160583, -27.427609537068406, -29.94455028482495,
    -32.46436896233801, -34.98695449773224, -37.51220798004591, -40.040040588748454, -42.5703719857798, -45.10312904512663,
    -47.638244833922855, -50.175657784433064]
plt.plot(t,v)
```

Out[5]: [<matplotlib.lines.Line2D at 0x1f1db832730>]