HIV_Bacteria_Computer_Lab

November 11, 2023

1 Python for Physical Modeling

1.1 Chapter 5: First Computer Lab

[2]:
```python
# import libraries
import numpy as np
import matplotlib.pyplot as plt
```

1.1.1 5.1 HIV Example

Here we explore a model of the viral load – the number of virons in the blood of a patient infected with HIV – after the administration of an antiretroviral drug. One model for the viral load predicts that the concentration $V(t)$ of HIV in the blood at time t after the start of treatment will be

$V(t) = Ae^{-\alpha t} + Be^{-\beta t}$ (equation 5.1)

Parameters:

- A and B specify the intial viral load
- α is the rate at which new cells are infected
- β is the rate at which virons are removed from the blood

1.1.2 5.1.1 Explore the model

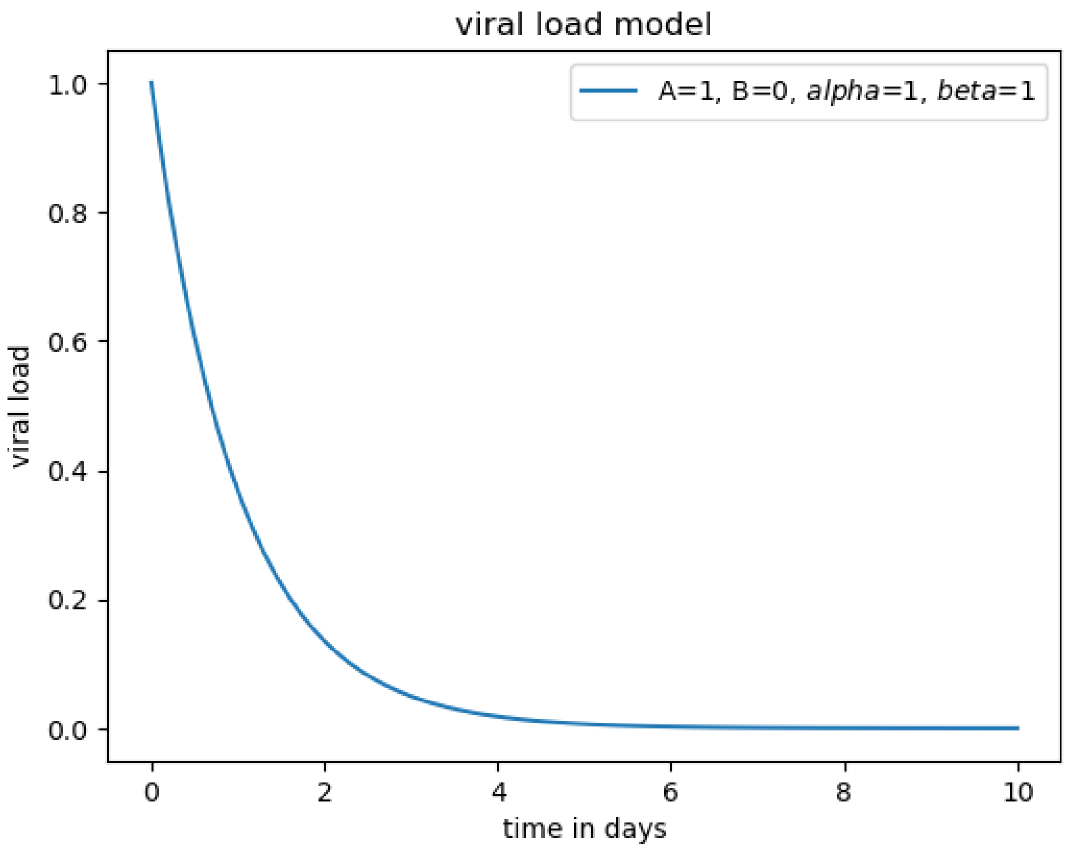
[3]:
```python
# discretize time
time = np.linspace(0, 10, 101)
time
```

[3]:
```
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,  0.8,  0.9,  1. ,
        1.1,  1.2,  1.3,  1.4,  1.5,  1.6,  1.7,  1.8,  1.9,  2. ,  2.1,
        2.2,  2.3,  2.4,  2.5,  2.6,  2.7,  2.8,  2.9,  3. ,  3.1,  3.2,
        3.3,  3.4,  3.5,  3.6,  3.7,  3.8,  3.9,  4. ,  4.1,  4.2,  4.3,
        4.4,  4.5,  4.6,  4.7,  4.8,  4.9,  5. ,  5.1,  5.2,  5.3,  5.4,
        5.5,  5.6,  5.7,  5.8,  5.9,  6. ,  6.1,  6.2,  6.3,  6.4,  6.5,
        6.6,  6.7,  6.8,  6.9,  7. ,  7.1,  7.2,  7.3,  7.4,  7.5,  7.6,
        7.7,  7.8,  7.9,  8. ,  8.1,  8.2,  8.3,  8.4,  8.5,  8.6,  8.7,
        8.8,  8.9,  9. ,  9.1,  9.2,  9.3,  9.4,  9.5,  9.6,  9.7,  9.8,
        9.9, 10. ])
```

Evaluate a compound expression invoking the time stay by using the solution of the viral load model in equation 5.1

set b = 0 and choose some values for alpha beta and A

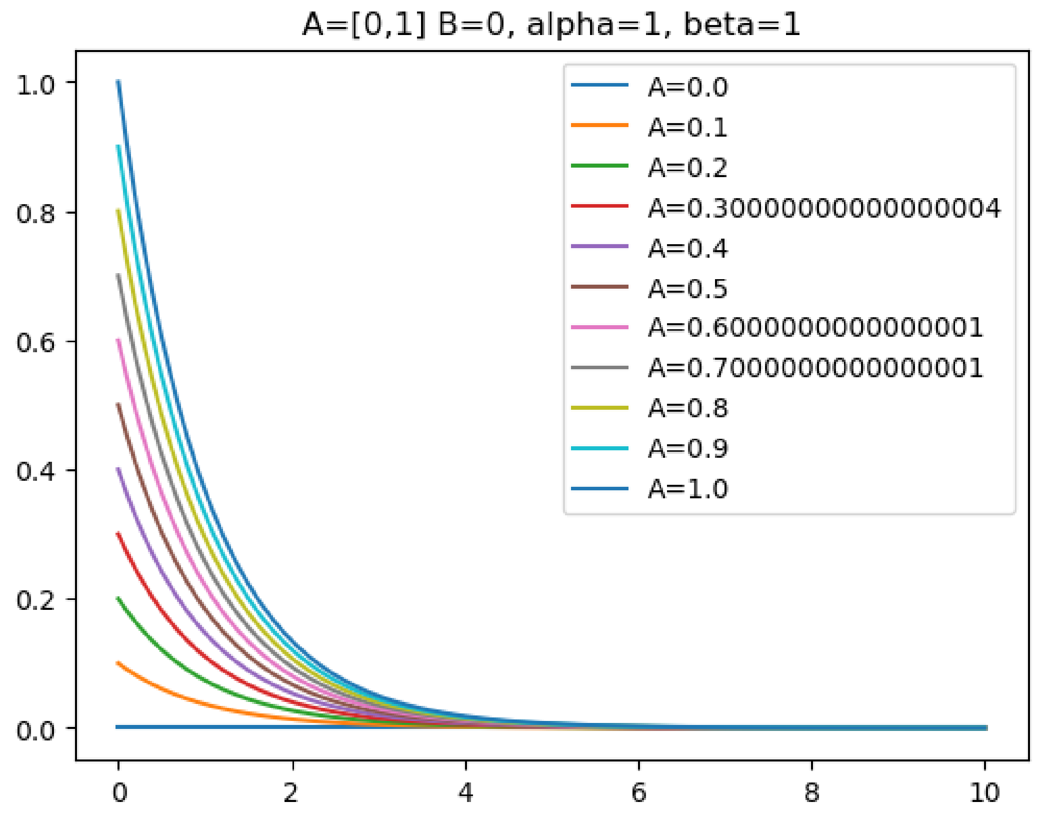
[4]:
```
# set parameters
A = 1
B = 0
alpha = 1
beta = 1

# evaluate model
viral_load = A * np.exp(-alpha*time) + B * np.exp(-beta*time)

# plot the evaluations over time
plt.plot(time, viral_load, label=f"A={A}, B={B}, $alpha$={alpha},␣
 ↪$beta$={beta}")
plt.legend()
plt.title('viral load model')
plt.xlabel('time in days')
plt.ylabel('viral load')
plt.show()
```

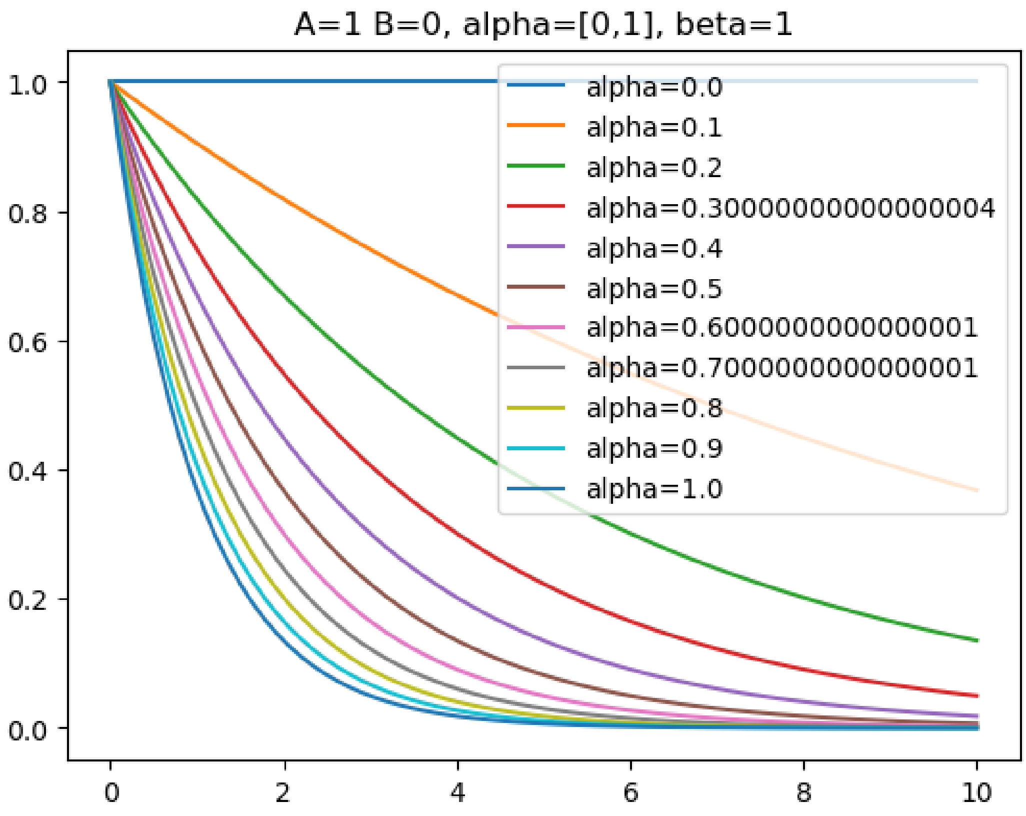


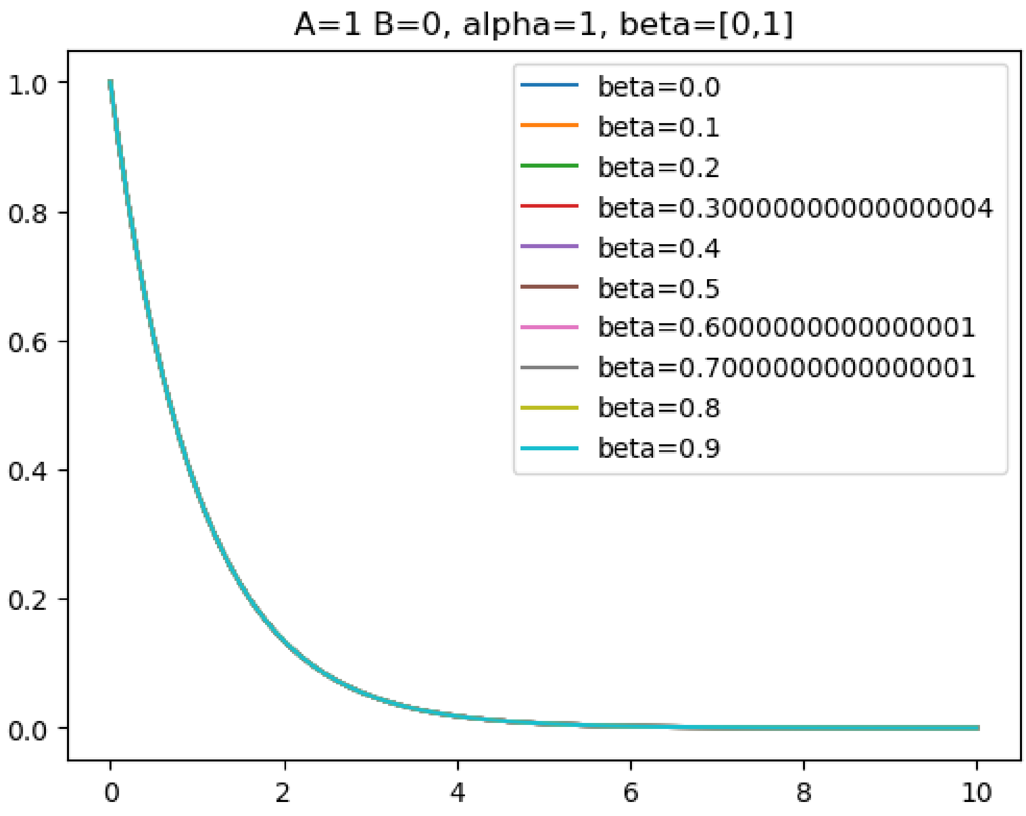
create a few more plots using different values of the four model parameters

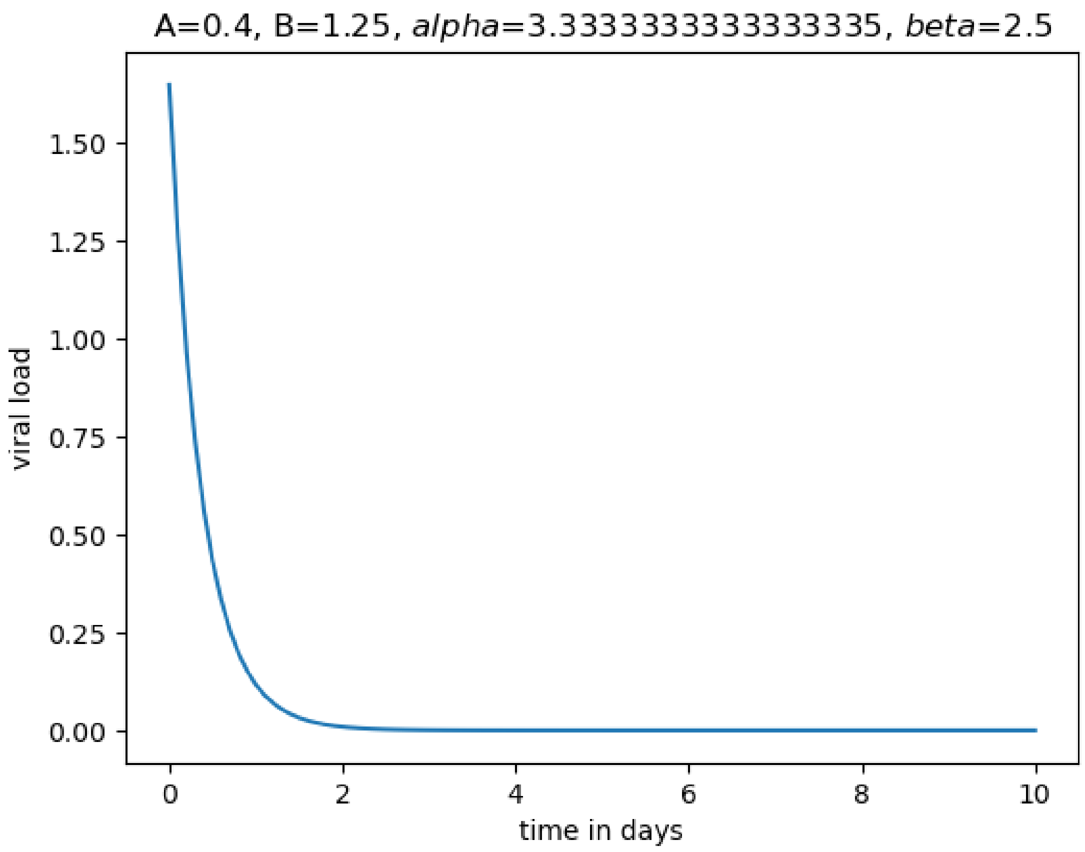
```
[5]: # range A from 0 to 1, B=0, alpha=1, beta=1
A_range = np.arange(0, 1.1, 0.1)
B=0
alpha=1
beta=1

range_vt_evals =  np.array([A_i * np.exp(-alpha*time) + B * np.exp(-beta*time)␣
 ↪for A_i in A_range])

range_index = 0
for vt_evals in range_vt_evals:
    plt.plot(time, vt_evals, label=f"A={A_range[range_index]}")
    range_index += 1
plt.title("A=[0,1] B=0, alpha=1, beta=1")
plt.legend()
plt.show()


# range alpha from 0 to 1, A=1, B=0, beta=1
A=1
B=0
alpha_range = np.arange(0, 1.1, 0.1)
beta=1

range_vt_evals =  np.array([A * np.exp(-alpha_i*time) + B * np.exp(-beta*time)␣
 ↪for alpha_i in alpha_range])

range_index = 0
for vt_evals in range_vt_evals:
    plt.plot(time, vt_evals, label=f"alpha={alpha_range[range_index]}")
    range_index += 1
plt.title("A=1 B=0, alpha=[0,1], beta=1")
plt.legend()
plt.show()


# range beta from 0 to 1, A=1, B=0, alpha=1
A=1
B=0
alpha=1
beta_range = np.arange(0, 1, 0.1)

range_vt_evals =  np.array([A * np.exp(-alpha*time) + B * np.exp(-beta_i*time)␣
 ↪for beta_i in beta_range])
```

```
range_index = 0
for vt_evals in range_vt_evals:
    plt.plot(time, vt_evals, '', label=f"beta={beta_range[range_index]}")
    range_index += 1
plt.title("A=1 B=0, alpha=1, beta=[0,1]")
plt.legend()
plt.show()


# set parameters for B as non zero value
A = 10**1/25
B = 10**1/8
alpha = 10**1/3
beta = 10**1/4

# evaluate model
viral_load = A * np.exp(-alpha*time) + B * np.exp(-beta*time)

# plot the evaluations over time
plt.plot(time, viral_load)
plt.title(f"A={A}, B={B}, $alpha$={alpha}, $beta$={beta}")
plt.xlabel('time in days')
plt.ylabel('viral load')
plt.show()

# TODO: remove redundancy here

#: TODO: repeat for A=0 and B=[0,1]?

# TODO: convert all multiple plots into subplots for each section
```


A=[0,1] B=0, alpha=1, beta=1
A=0.0
A=0.1
A=0.2
A=0.30000000000000004
A=0.4
A=0.5
A=0.6000000000000001
A=0.7000000000000001
A=0.8
A=0.9
A=1.0
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10


A=1 B=0, alpha=[0,1], beta=1
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
alpha=0.0
alpha=0.1
alpha=0.2
alpha=0.30000000000000004
alpha=0.4
alpha=0.5
alpha=0.6000000000000001
alpha=0.7000000000000001
alpha=0.8
alpha=0.9
alpha=1.0


A=1 B=0, alpha=1, beta=[0,1]
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
10
beta=0.0
beta=0.1
beta=0.2
beta=0.30000000000000004
beta=0.4
beta=0.5
beta=0.6000000000000001
beta=0.7000000000000001
beta=0.8
beta=0.9

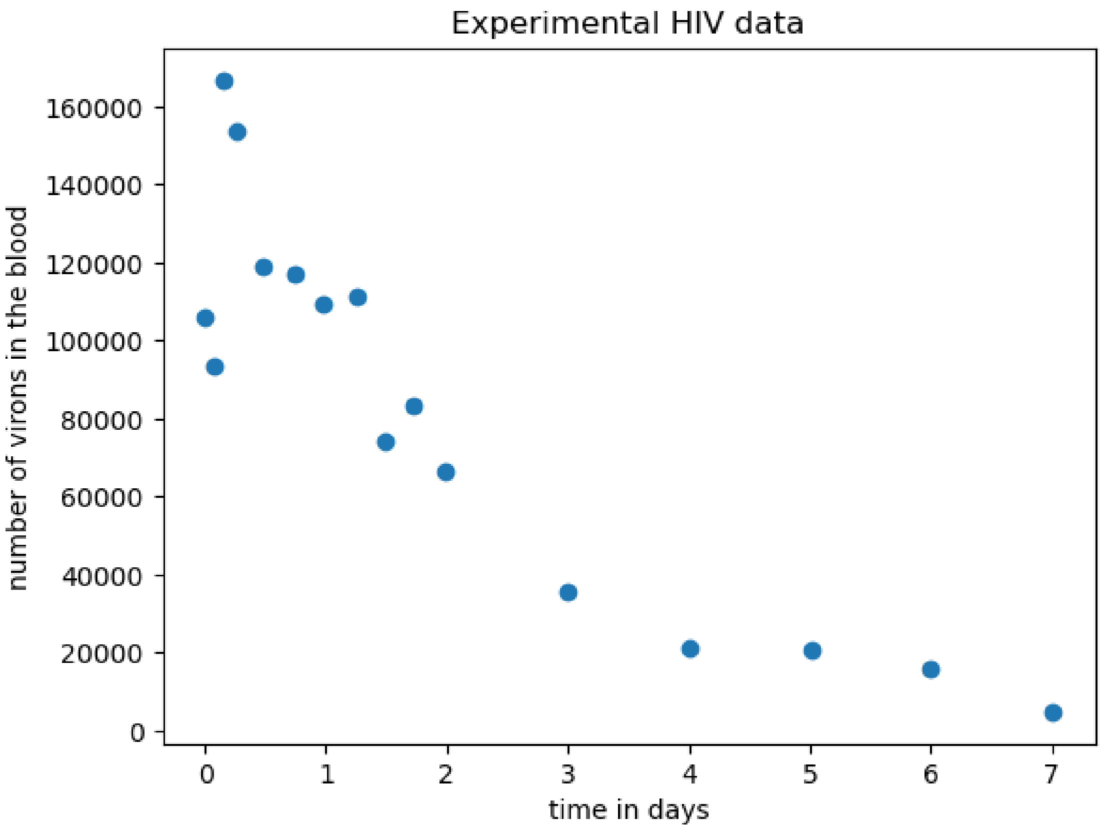


1.1.3 5.1.2 Experimental Data

obtain the dataset 01HIVseries and plot the data

[6]:
```python
hiv_data_file = "../pmls-data-master/01HIVseries/HIVseries.csv"
hiv_data = np.loadtxt(hiv_data_file, delimiter=',')
hiv_num_days = np.array(hiv_data[:,0])
hiv_num_virons = np.array(hiv_data[:,1])

print("HIV experimental data")
print("_______________________")
print("number of days since start of antiretroviral treatment:")
print(hiv_num_days)
print()
print("number of virons in the blood:")
print(hiv_num_virons)


plt.plot(hiv_num_days, hiv_num_virons, 'o')
plt.title('Experimental HIV data')
plt.xlabel('time in days')
plt.ylabel('number of virons in the blood')
plt.show()
```

```
HIV experimental data
----------------------
number of days since start of antiretroviral treatment:
[0.     0.0831 0.1465 0.2587 0.4828 0.7448 0.9817 1.2563 1.4926 1.7299
 1.9915 3.0011 4.0109 5.009  5.9943 7.0028]

number of virons in the blood:
[106100.   93240.  166720.  153780.  118800.  116900.  109570.  111350.
  74388.   83291.   66435.   35408.   21125.   20450.   15798.    4785.2]
```



Assignment: a. Plot the experimental data points and the function in equation 5.1 on the same axes. Adjust the four parameters of the model until you can see both the data and the model in your plot

The goal is to now tune the four parameters of Equation 5.1 until the model agrees with the data. It is hard to find the right needle in a four-dimensional haystack. We need a more systematic approach than just guessing. Consider the following: Assuming $\beta > \alpha$, how does the trial solution behave at long times?

If the data also behave that way can we use the long time-behavior to determine two of the four unkown constants, then hold them fixed while adjusting the other two?

Assignment b: Carry out this analysis so that you have only one remaining free parameter, which you can adjust farily easily. Adjust this parameter until you like what you see.

2 $V(t) = Ae^{-at} + Be^{-bt}$

looking at the data above we can see in the recorded data that

$V(0) = 106100 $

    &

$V(7.0028) = 4785.2$

assume $B = 0$

```
[7]: B = 0
     print(f"B= {B}")
```

```
B= 0
```

assume B = 0

assume beta > alpha

at $V(t = 0)$ with $B = 0$ we get:

$V(0) = Ae^{-a*0} + 0$

at $V(7.0028)$ with $B = 0$ we get:

$V(7, 0028) = Ae^{-a*7.0028} + 0$

we know the value of $V(0)$ and $V(7.0028)$:

$V(0) = A = 106100$

$V(7.0028) = Ae^{-a*7.0028} = 4785.2$

setting the value of A for $V(7.0028)$ we get:

$V(7.0028) = 106100e^{-a*7.0028} = 4785.2$

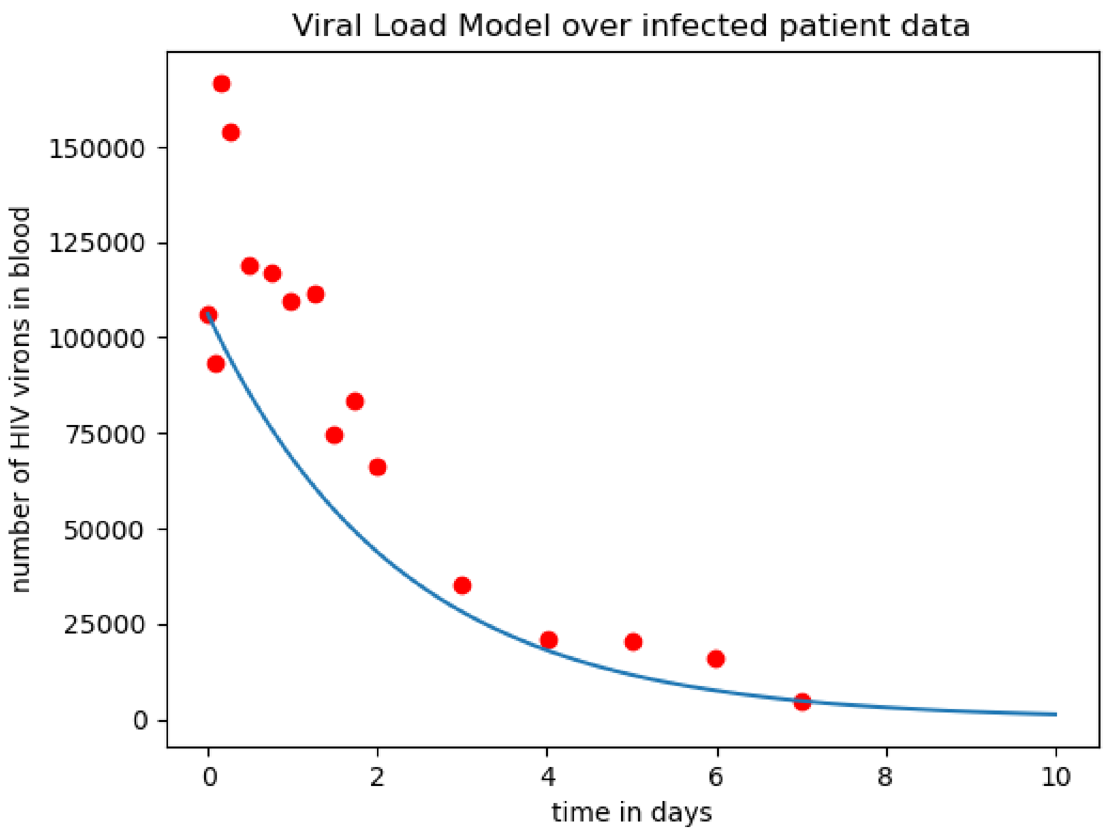
$e^{-a*7.0028} = \frac{4785.2}{106100}$

solving for alpha we get:

$\alpha = \frac{2500(log(2)+3log(5)-log(7)+log(1061)-log(1709))}{17507} = 0.442516$

```
[8]: B = 0
     A = 106100
     beta = 1
     alpha = 0.442516
```

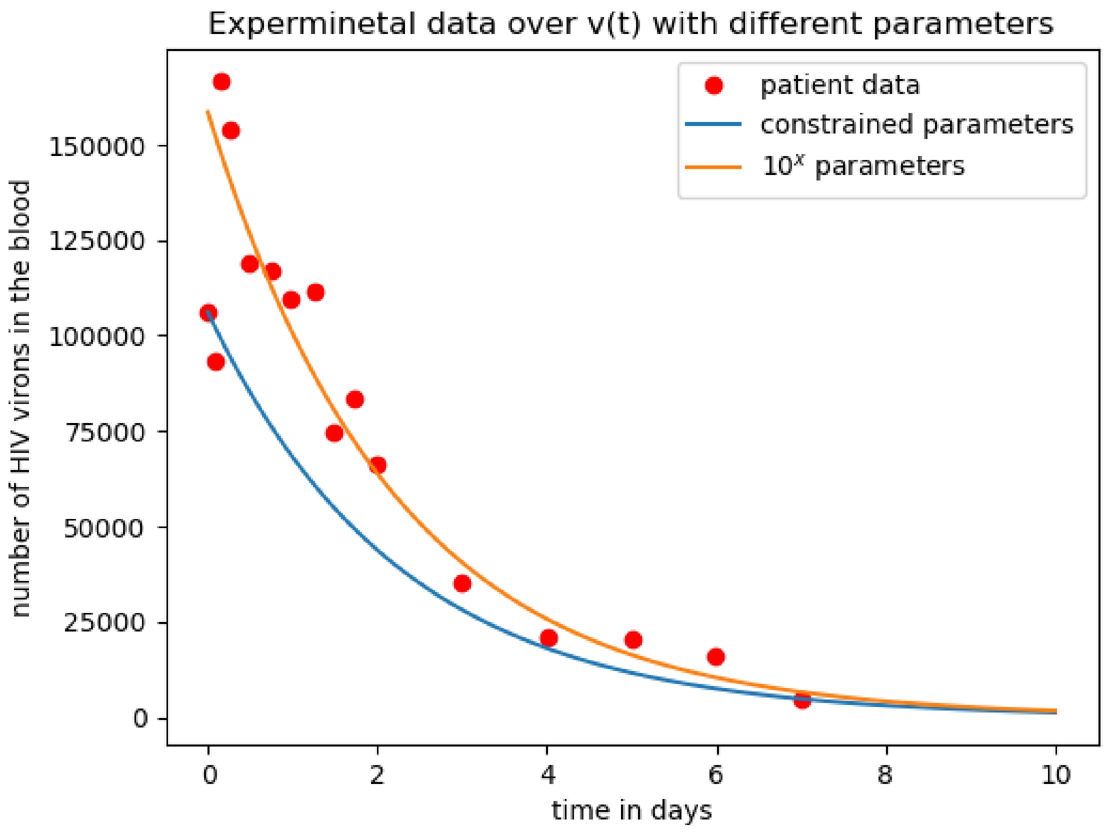
[9]:
```python
plt.plot(hiv_num_days, hiv_num_virons, 'ro')
t = np.linspace(0, 10, 101)
v_t = A*np.e**(-alpha*t) + B*np.e**(-beta*t)
plt.plot(t, v_t)
plt.title("Viral Load Model over infected patient data")
plt.xlabel("time in days")
plt.ylabel("number of HIV virons in blood")
plt.show()
```



alternative parameters where B=0 still and beta > alpha but paramets tuned by adjustments to 10^x

[10]:
```python
# original parameters
B = 0
A = 106100
beta = 1
alpha = 0.442516

plt.plot(hiv_num_days, hiv_num_virons, 'ro', label="patient data")
t = np.linspace(0, 10, 101)
v_t = A*np.e**(-alpha*t) + B*np.e**(-beta*t)
```

```
plt.plot(t, v_t, label="constrained parameters")

# alternative parameters
A = 10**5.2
B = 0
alpha = 10**1/22
beta = 10**0

v_t = A*np.e**(-alpha*t) + B*np.e**(-beta*t)
plt.plot(t, v_t, label="$10^x$ parameters")

plt.legend()
plt.xlabel("time in days")
plt.ylabel("number of HIV virons in the blood")
plt.title("Experminetal data over v(t) with different parameters")
plt.show()


print(f"1/alpha = {1/alpha}")
```



```
1/alpha = 2.2
```

Assignment c: The latency period of HIV is about ten years, or 3600 days. Based on your results how does the inverse of the T-cell infection rate $1/\alpha$, compare to the latency period? Is the latency period long because it takes HIV a long time to infect new cells? Or does the model suggest another scenario?

Alpha inverse is 2.2 compared to 10 years for the latency period of HIV, but im not sure what ibservations can be made based off of these two values

2.1 5.2 Bacterial Example

Here are two families of functions that come up in the analysis of the Novick-Weiner experiment:

$V(t) = 1 - \mathrm{e}^{-t/\tau}$

& (Equation 5.2)

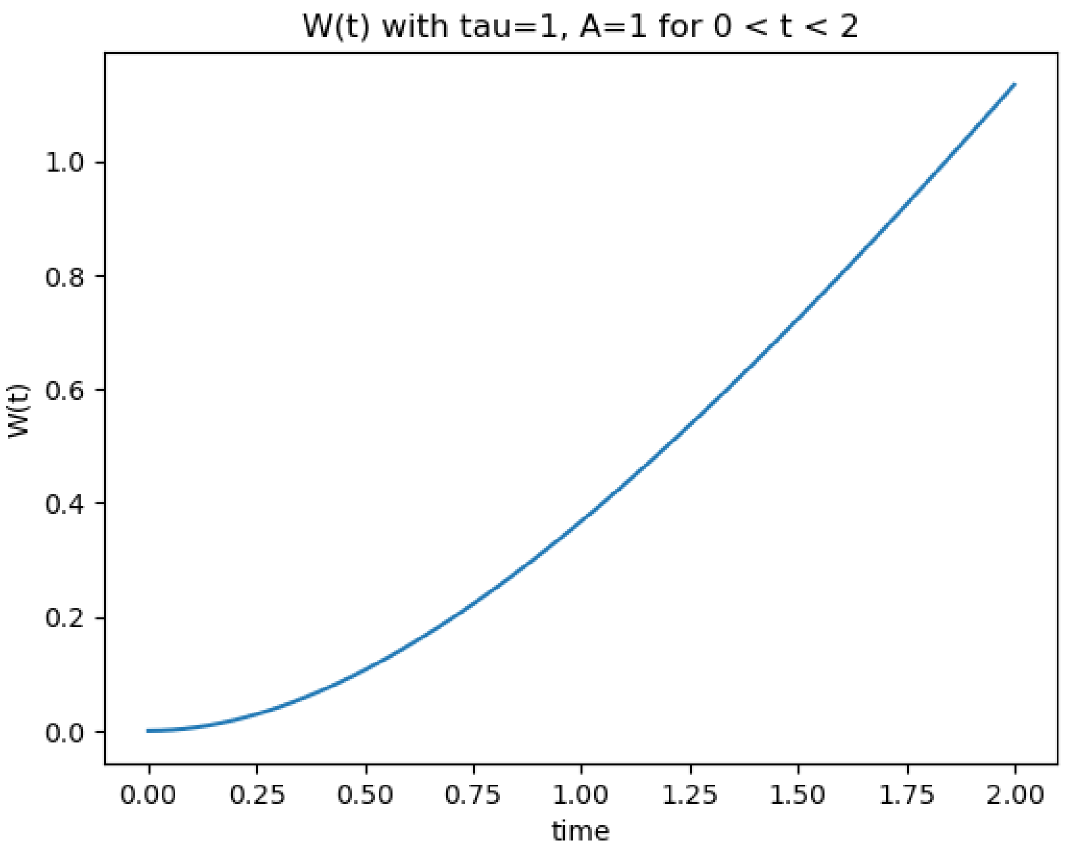
$W(t) = A\left(\mathrm{e}^{-t/\tau} - 1 + \frac{t}{\tau}\right)$

The parameters A and τ are constants.

Assignment a. choose $A = 1, \tau = 1$, and plot $W(t)$ for $0 < t < 2$.

[11]:
```python
def w(t, A, tau):
    return A * (np.exp(-t/tau) - 1 + (t/tau))
```

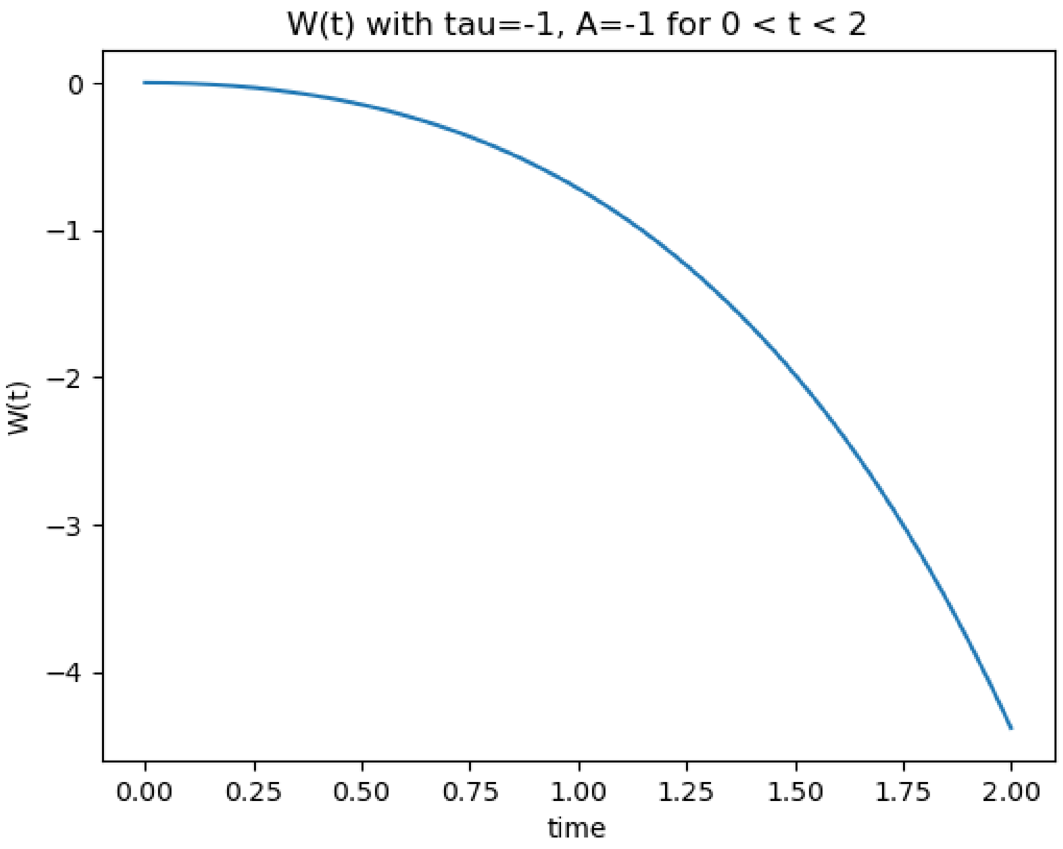
[12]:
```python
A=1
tau=1
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)
plt.plot(t, w_t)
plt.xlabel("time")
plt.ylabel("W(t)")
plt.title(f"W(t) with tau={tau}, A={A} for 0 < t < 2")
plt.show()
```

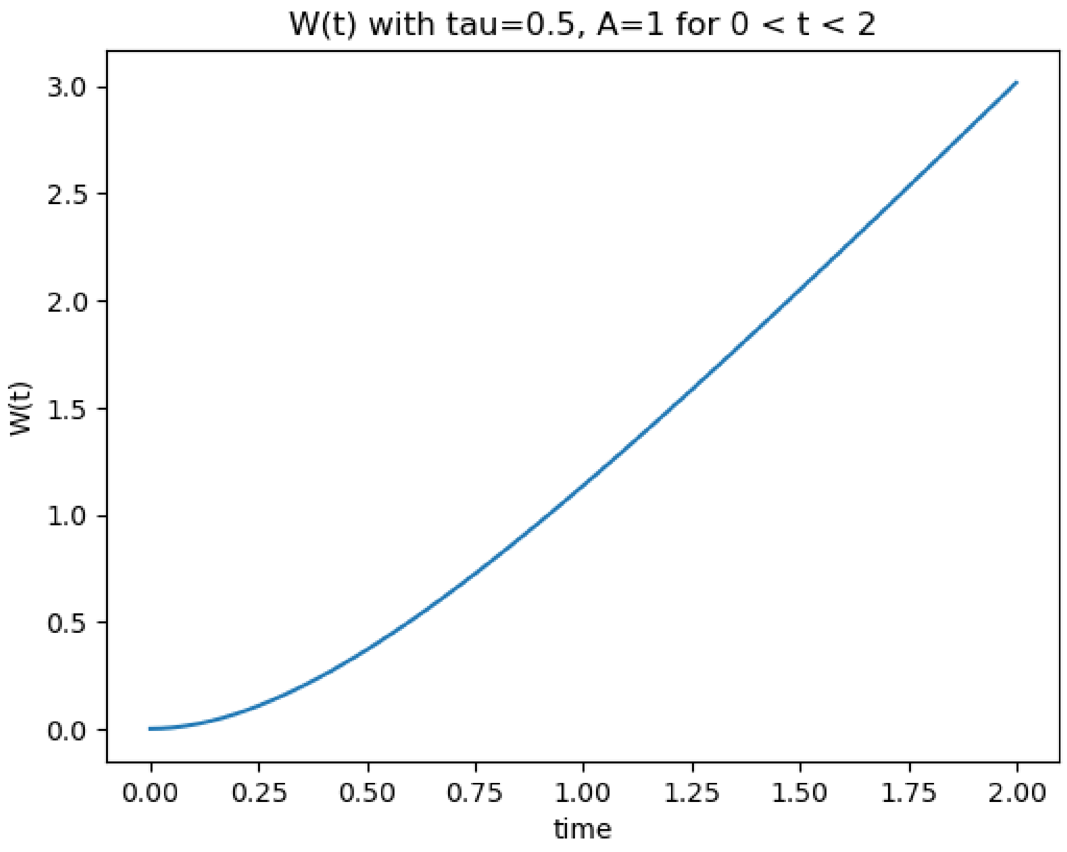


Assignment b. Make several arrays W1, W2, w3, and so on, using different values of τ and A, and plot them all on the same axes.

[13]:
```
A=-1
tau=-1
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)
plt.plot(t, w_t)
plt.xlabel("time")
plt.ylabel("W(t)")
plt.title(f"W(t) with tau={tau}, A={A} for 0 < t < 2")
plt.show()

A=1
tau=0.5
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)
plt.plot(t, w_t)
plt.xlabel("time")
plt.ylabel("W(t)")
```

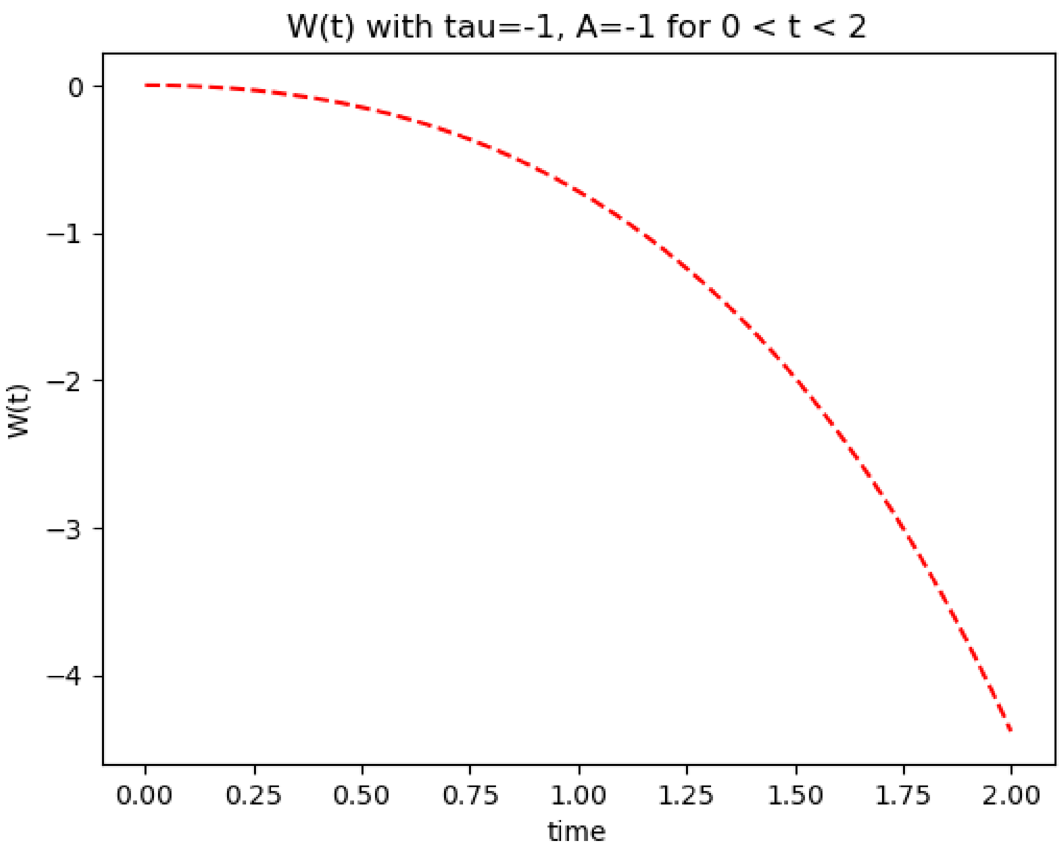
```python
plt.title(f"W(t) with tau={tau}, A={A} for 0 < t < 2")
plt.show()
```

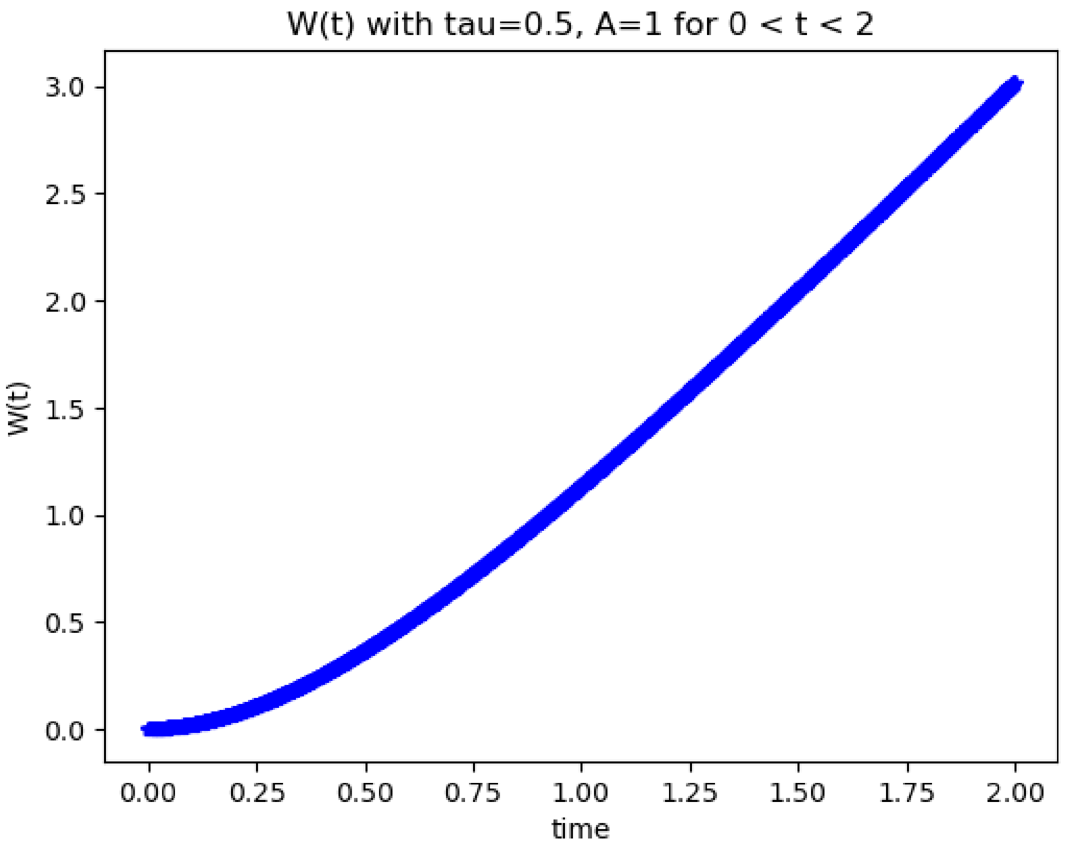




Assignment c. Change the colors and line styles (solid, dashed, and so on) of the lines.

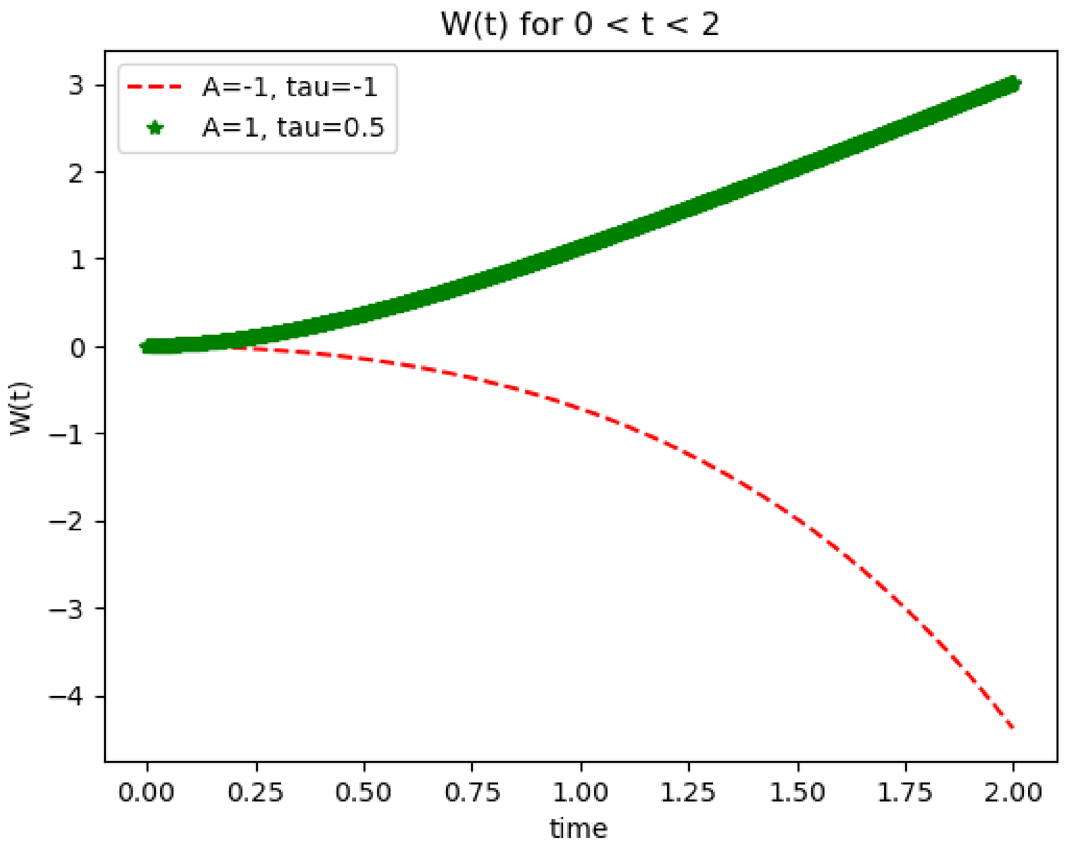
[14]:
```
A=-1
tau=-1
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)
plt.plot(t, w_t, 'r--')
plt.xlabel("time")
plt.ylabel("W(t)")
plt.title(f"W(t) with tau={tau}, A={A} for 0 < t < 2")
plt.show()

A=1
tau=0.5
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)

plt.plot(t, w_t, 'b*')
plt.xlabel("time")
plt.ylabel("W(t)")
plt.title(f"W(t) with tau={tau}, A={A} for 0 < t < 2")

plt.show()
```



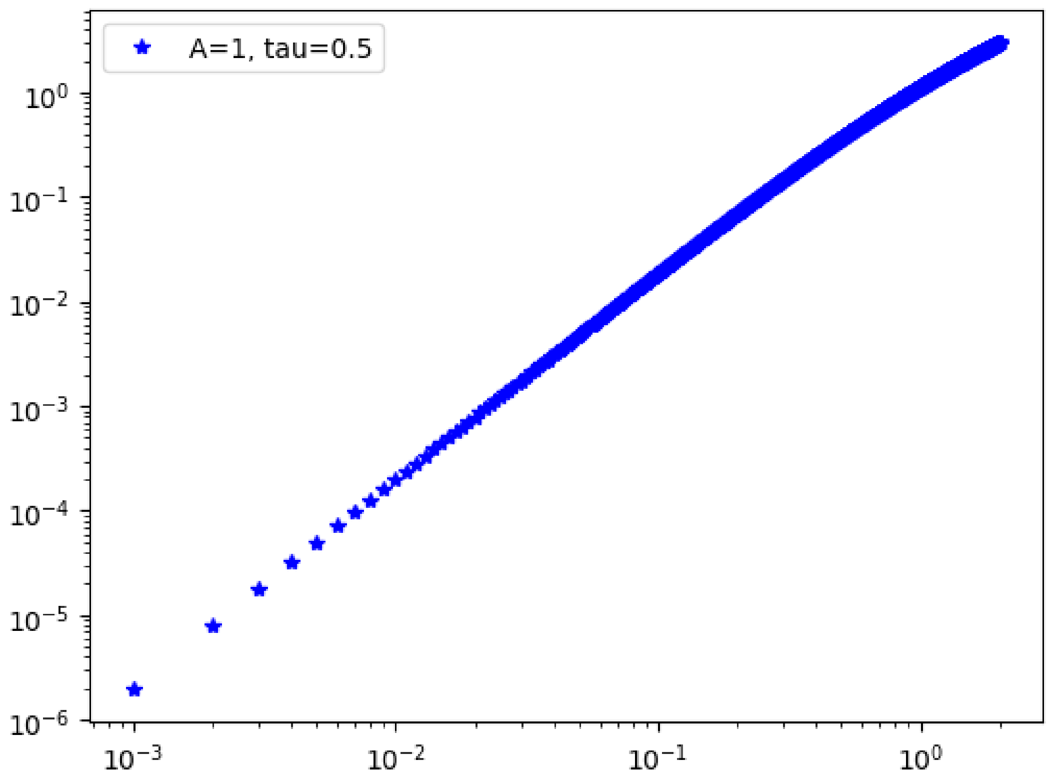


Assignment d. Add a legend to help a reader sort out the curves. Explore some of the other graph options.

[15]:
```
A=-1
tau=-1
t = np.arange(0.001, 2, 0.001)
w_t = w(t, A, tau)
label=f"A={A}, tau={tau}"
plt.plot(t, w_t, 'r--', label=label)
plt.xlabel("time")
plt.ylabel("W(t)")
plt.title("W(t) for 0 < t < 2")


A=1
tau=0.5
w_t = w(t, A, tau)
label=f"A={A}, tau={tau}"
plt.plot(t, w_t, 'g*', label=label)
plt.legend()
plt.show()
```



[16]:
```
A=1
tau=0.5
w_t = w(t, A, tau)
label=f"A={A}, tau={tau}"
plt.loglog(t, w_t, 'b*', label=label)
plt.legend()
plt.show()
```



2.2 5.2.2 Fit Experimental Data

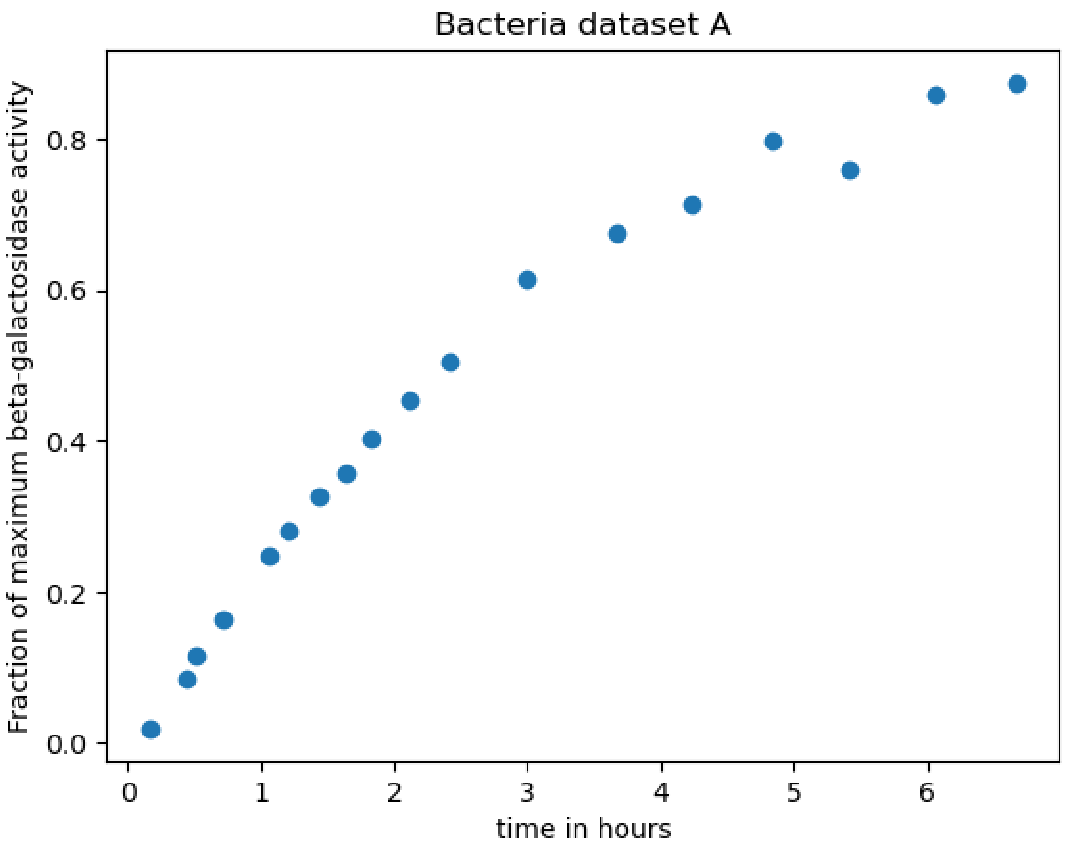
Follow the instructions of Section 4.1 to obtain the data set 15novick. Copy g149novicka.csv, g149novickA.npy, g149novickB.csv, g149novickB. npy, and README. txt into your working directory, or be sure you can locate these files using paths. The files contain time series data from a bacterial population in a culture.

Read in the bacteria datasets A and B

[17]:
```
bacteria_df_a = "../pmls-data-master/15novick/g149novickA.csv"
bacteria_data_a = np.loadtxt(bacteria_df_a, delimiter=',')
bacteria_df_b = "../pmls-data-master/15novick/g149novickB.csv"
bacteria_data_b = np.loadtxt(bacteria_df_b, delimiter=',')
```

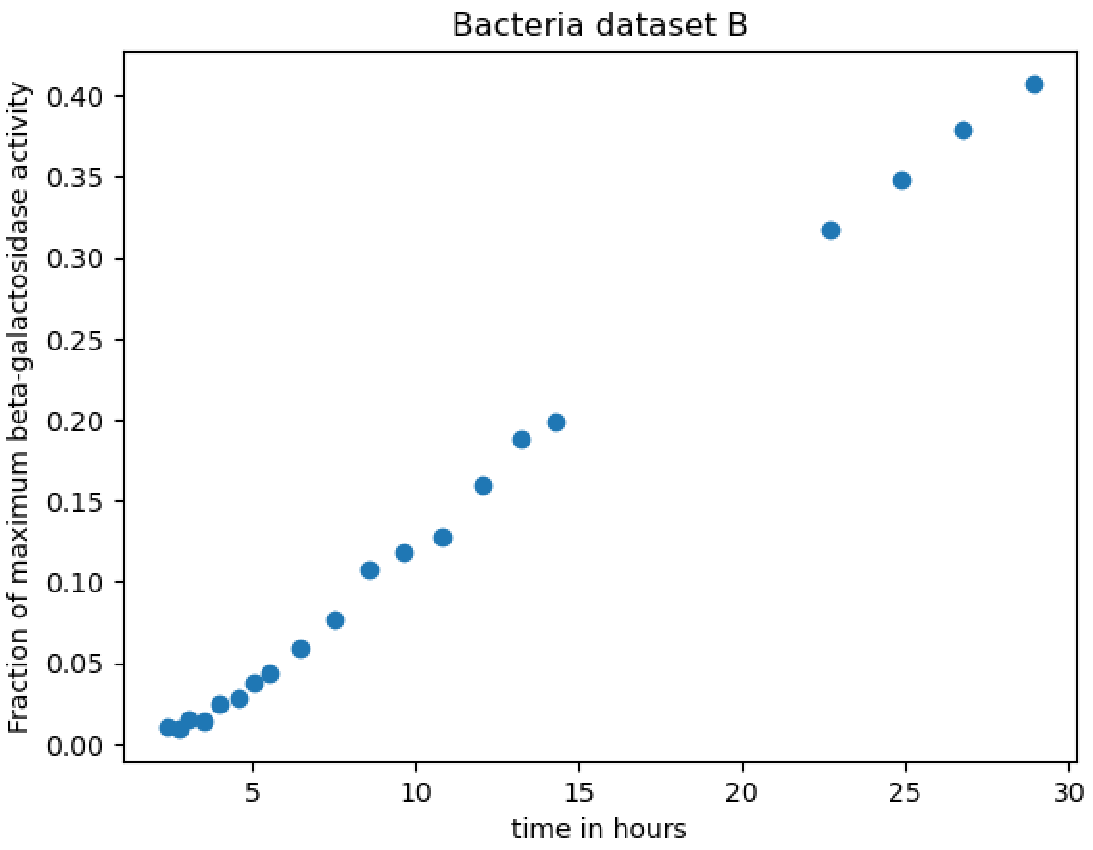
Display the data

[18]:
```
plt.plot(bacteria_data_a[:,0], bacteria_data_a[:,1], 'o')
plt.title("Bacteria dataset A")
plt.ylabel("Fraction of maximum beta-galactosidase activity")
plt.xlabel("time in hours")
plt.show()
print(bacteria_data_a)
```



```
[[0.1699 0.019 ]
 [0.4426 0.0855]
 [0.5111 0.1164]
 [0.7156 0.1639]
 [1.0564 0.247 ]
 [1.2041 0.2803]
 [1.4311 0.3278]
 [1.6465 0.3563]
 [1.8283 0.4038]
 [2.1119 0.4537]
 [2.4182 0.5059]
 [2.997  0.6152]
 [3.6763 0.6746]
 [4.2308 0.7126]
 [4.8316 0.7981]
 [5.407  0.7601]
 [6.0646 0.8599]
 [6.6638 0.8741]]
```

[19]:
```
plt.plot(bacteria_data_b[:,0], bacteria_data_b[:,1], 'o')
plt.title("Bacteria dataset B")
plt.ylabel("Fraction of maximum beta-galactosidase activity")
plt.xlabel("time in hours")
plt.show()
print(bacteria_data_b)
```



```
[[2.38320e+00 1.09000e-02]
 [2.72300e+00 9.34290e-03]
 [3.03580e+00 1.46000e-02]
 [3.51770e+00 1.45000e-02]
 [4.00140e+00 2.42000e-02]
 [4.56920e+00 2.86000e-02]
 [5.05280e+00 3.76000e-02]
 [5.53580e+00 4.35000e-02]
 [6.44580e+00 5.92000e-02]
 [7.52620e+00 7.64000e-02]
 [8.58090e+00 1.07100e-01]
 [9.63180e+00 1.18200e-01]
 [1.08242e+01 1.27800e-01]
 [1.20208e+01 1.60000e-01]
 [1.31884e+01 1.88500e-01]
 [1.42958e+01 1.98900e-01]
 [2.27088e+01 3.17300e-01]
 [2.48690e+01 3.48600e-01]
 [2.67739e+01 3.78400e-01]
 [2.89053e+01 4.07500e-01]]
```

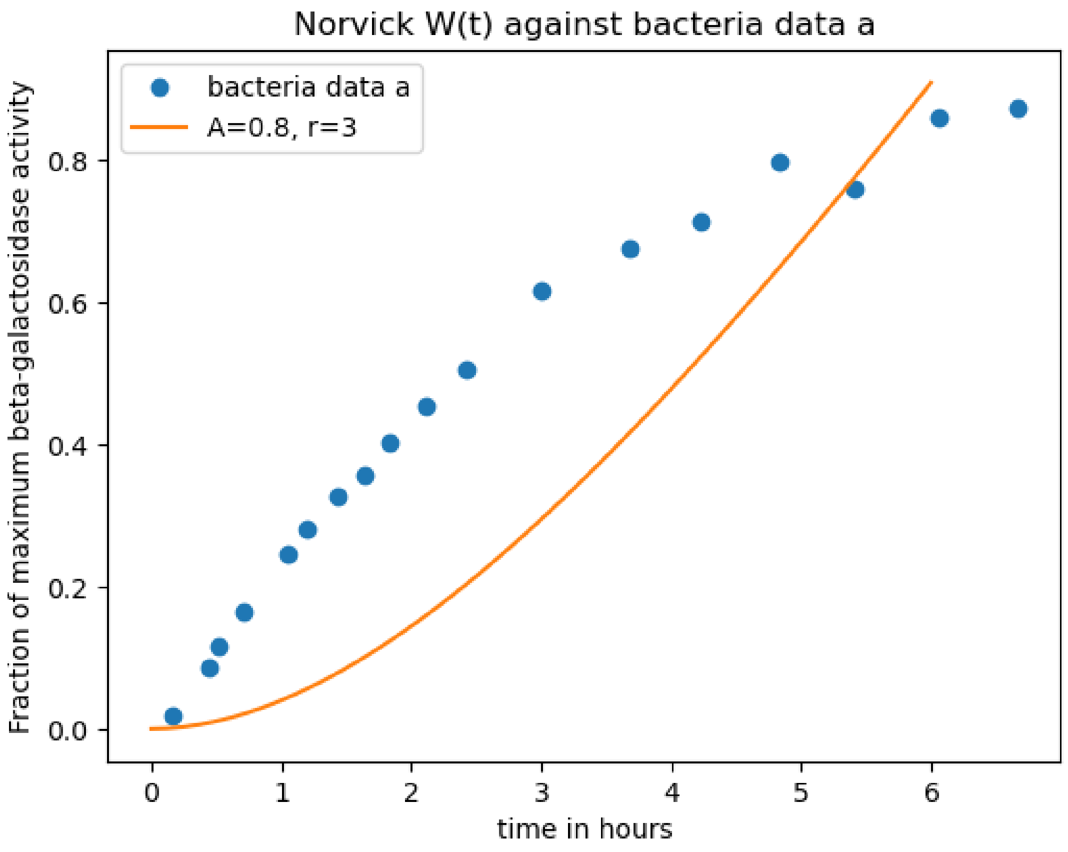
column 1 is time in hours column 2 is Fraction of maximum beta-galactosidase activity.

a. plot the experimental data points and your trial functions for V(t) on the same axes as you did for W(t) before

when you plot the experimental data, do not join the points by line segments; make each point a symbol such as a small circle or plus sign. label the axes of your plot select some reasonable values for the parameter r in the model and see if you can get a curve that fits thhe data well. label the curves, then add a legend to identify which curve is which. hint to find a good estimate of r make a semilog plot of the quantity 1.0 - data versus time, where data is the array of experimental data points. can you explain why this is helfpul? see section 4.3.2 for more details

[20]:
```python
# define v(t) and w(t)

def bact_v(t,r):
    return 1 - np.e**(-t/r)

def bact_w(t,r, A):
    return A*(np.e**(-t/r) - 1 + (t/r))
```

[21]:
```python
A = 0.8
r = 3
plt.close()
plt.plot(bacteria_data_a[:,0], bacteria_data_a[:,1], 'o', label="bacteria data␣
 ↪a")

t = np.linspace(0,6,50)
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")

plt.title("Norvick W(t) against bacteria data a")
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")
plt.show()
```

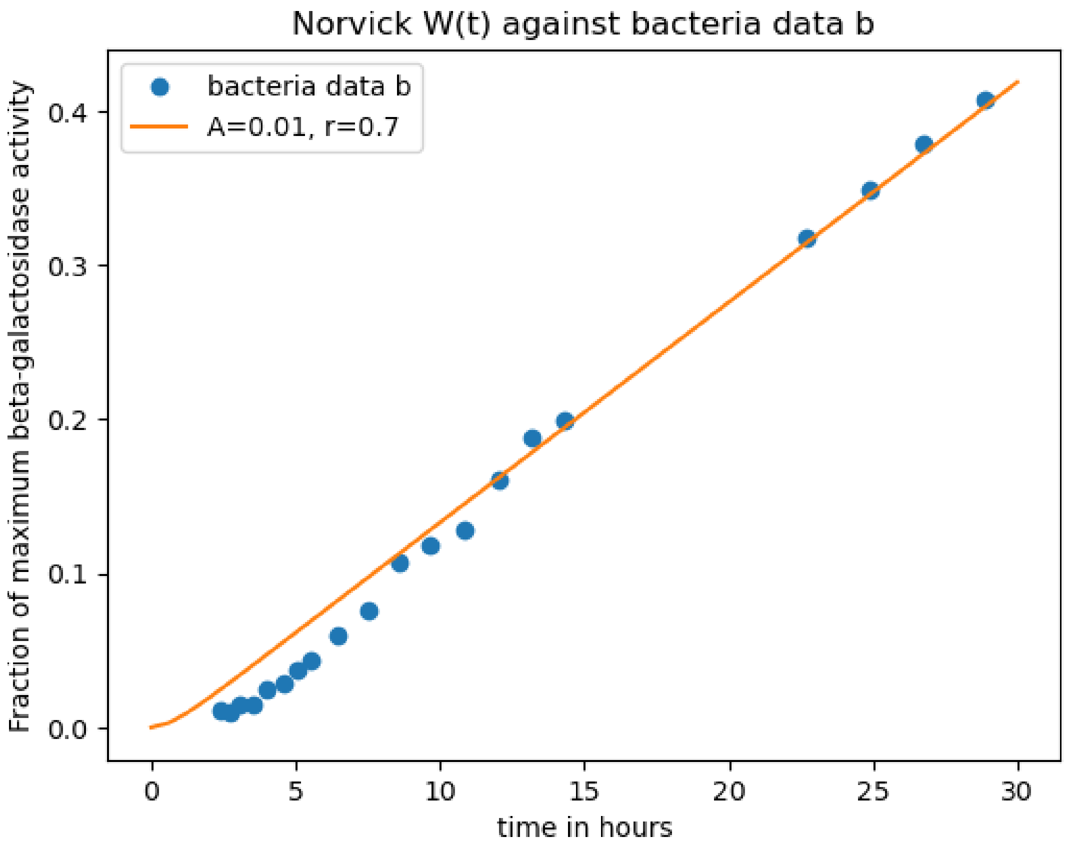


b. now try the same thing using the data in gl49novickB.csv. this time throw away all the data with time value greater than ten hours, and attempt to fit the remainign data to the family of functions W(t) in equation 5.2

hint you can throw away data by slicing an array

at large values of t but smaller than ten hours, both the data and the function W(t) become straight lines. find the slope and the y intercept of the straight line determined by equation 5.2 in terms of the two unkown quantities A and r. next estimate the slope and y intercept of the straight line determed by the data. from this figure out some good initial guesses for the values of A and r. then tweak the values to get a nicer looking fit

[22]:
```
A = 0.01
r = 0.7
plt.close()
plt.plot(bacteria_data_b[:,0], bacteria_data_b[:,1], 'o', label="bacteria data␣
  ↪b")

t = np.linspace(0,30,50)
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")
plt.title("Norvick W(t) against bacteria data b")
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")
plt.show()
```



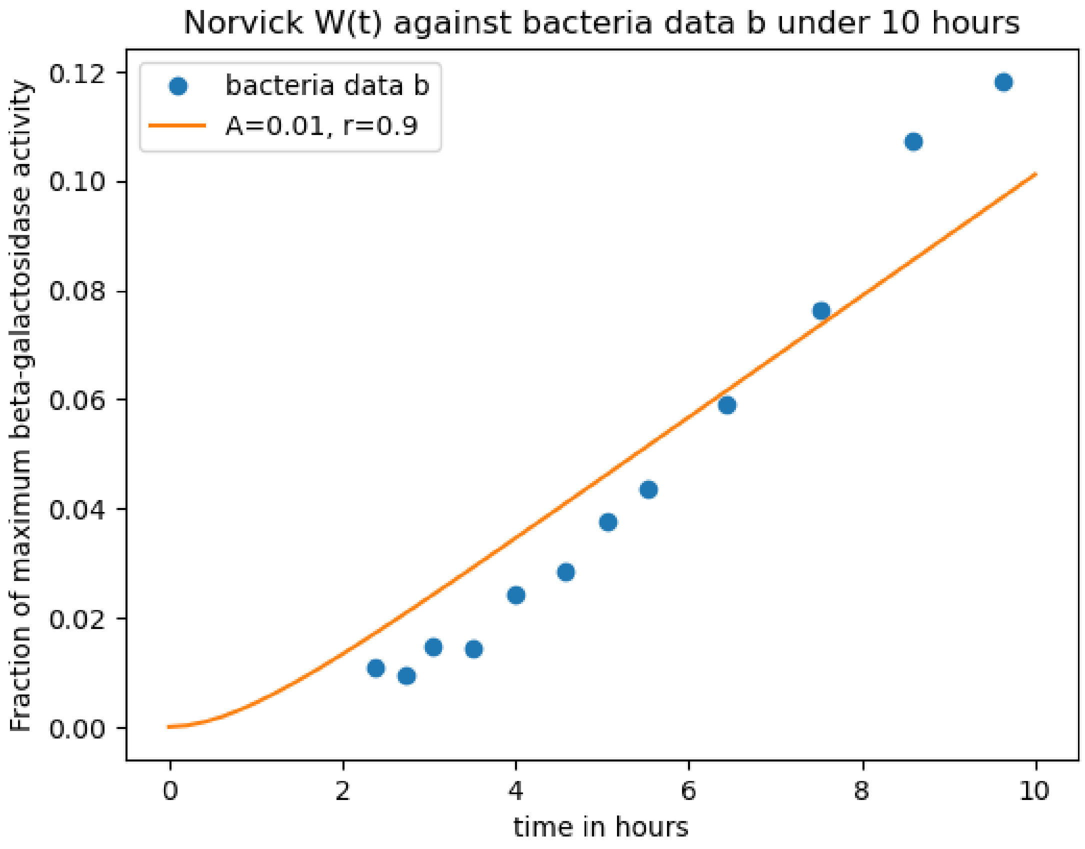
slice the data under 10 with list comprehension below

[23]:
```
sliced_b_time = bacteria_data_b[:-8,0]
sliced_b_time
```

[23]:
```
array([2.3832, 2.723 , 3.0358, 3.5177, 4.0014, 4.5692, 5.0528, 5.5358,
       6.4458, 7.5262, 8.5809, 9.6318])
```

[24]:
```
sliced_b_y=bacteria_data_b[:-8,1]
```

[25]:
```
A = 0.01
r = 0.9
plt.close()
plt.plot(sliced_b_time, sliced_b_y, 'o', label="bacteria data b")

t = np.linspace(0,10,50)
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")

plt.title("Norvick W(t) against bacteria data b under 10 hours")
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")

plt.show()
```



at large values of t but smaller than ten hours, both the data and the function W(t) become straight lines. find the slope and the y intercept of the straight line determined by equation 5.2 in terms of the two unkown quantities A and r. next estimate the slope and y intercept of the straight line determed by the data. from this figure out some good initial guesses for the values of A and r. then tweak the values to get a nicer looking fit

$V(t) = 1 - e^{-t/\tau}$

& (Equation 5.2)

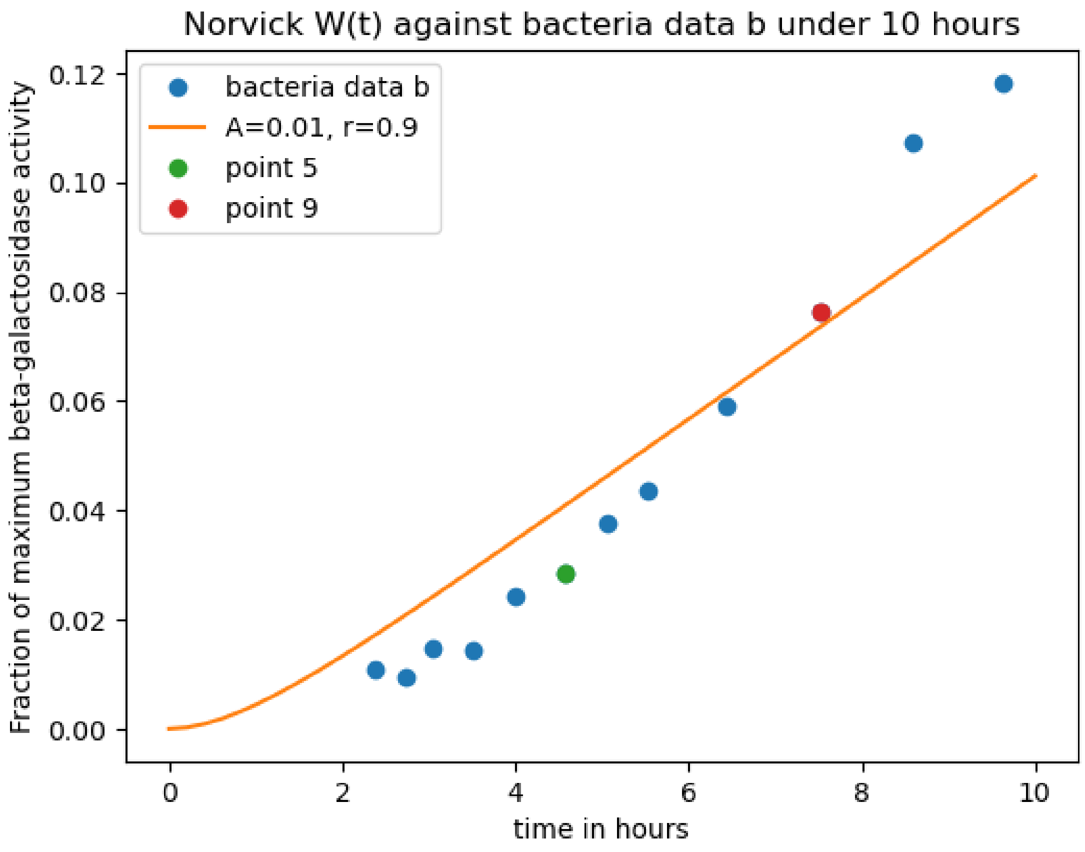
$W(t) = A\left(\mathrm{e}^{-t/\tau} - 1 + \frac{t}{\tau}\right)$

[26]:
```
# TODO: get two data points from recorded data that fit near function
# TODO: find the slope and y intercept using two points

#Todo: set recorded values as inout and output for v of t and solve for A and r

sliced_b_y[5]
sliced_b_time[5]
```

[26]:
```
4.5692
```

$W(t) = A\left(\mathrm{e}^{-t/\tau} - 1 + \frac{t}{\tau}\right)$

[27]:
```
# TODO: plot two points and display their values
#TODO add comments

A = 0.01
r = 0.9
plt.close()
plt.plot(sliced_b_time, sliced_b_y, 'o', label="bacteria data b")


t = np.linspace(0,10,50)
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")
plt.plot(sliced_b_time[5], sliced_b_y[5],'o', label="point 5")
plt.plot(sliced_b_time[9], sliced_b_y[9],'o', label="point 9")
plt.title("Norvick W(t) against bacteria data b under 10 hours")
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")


plt.show()
```



[28]:
```
print(f"point 9: {sliced_b_time[9]}, {sliced_b_y[9]}")
print(f"point 5: {sliced_b_time[5]}, {sliced_b_y[5]}")
```

```
point 9: 7.5262, 0.0764
point 5: 4.5692, 0.0286
```

slope of a line:

$y = mx + b$

$m = \frac{y_2 - y_1}{x_2 - x_1}$

$y = (\frac{y_2 - y_1}{x_2 - x_1})x + b$

$y_1 = 0.0764, y_2 = 0.0286, x_1 = 7.5262, x_2 = 4.5692$

$y = (\frac{0.0286 - 0.0764}{4.5692 - 7.5262})x + b$

[29]:
```
numer = 0.0286-0.0764
denom = 4.5692-7.5262
m = numer/denom
print(f"slope = m={m}")
```

```
slope = m=0.0161650321271559
```

y intercept

$b = y - mx$

$b = 0.0764 - (0.0161650321271559)(7.5262)$

```
[30]: b = sliced_b_y[5] - m*sliced_b_time[5]
      print("The y intecept " + str(b))
```

```
The y intecept -0.04526126479540074
```

```
[39]: print("The slope " + str(m))
```

```
The slope 0.0161650321271559
```

```
[40]: # define equation for line with above slope and y intercept
      def y(x, m, b):
          return m*x +b
```

```
[41]: t
```

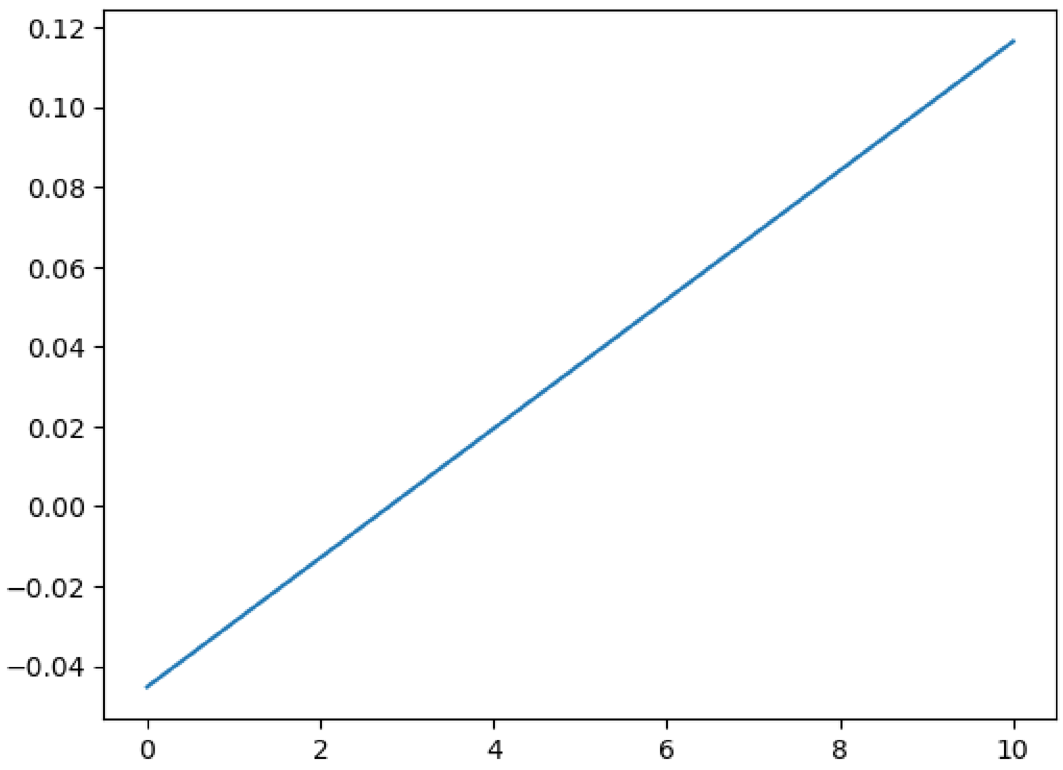
```
[41]: array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
              1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
              2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
              3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
              4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
              5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
              6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
              7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
              8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
              9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

```
[43]: # evaluate straight line over time domain
      y_t = y(t, m, b)
      y_t
```

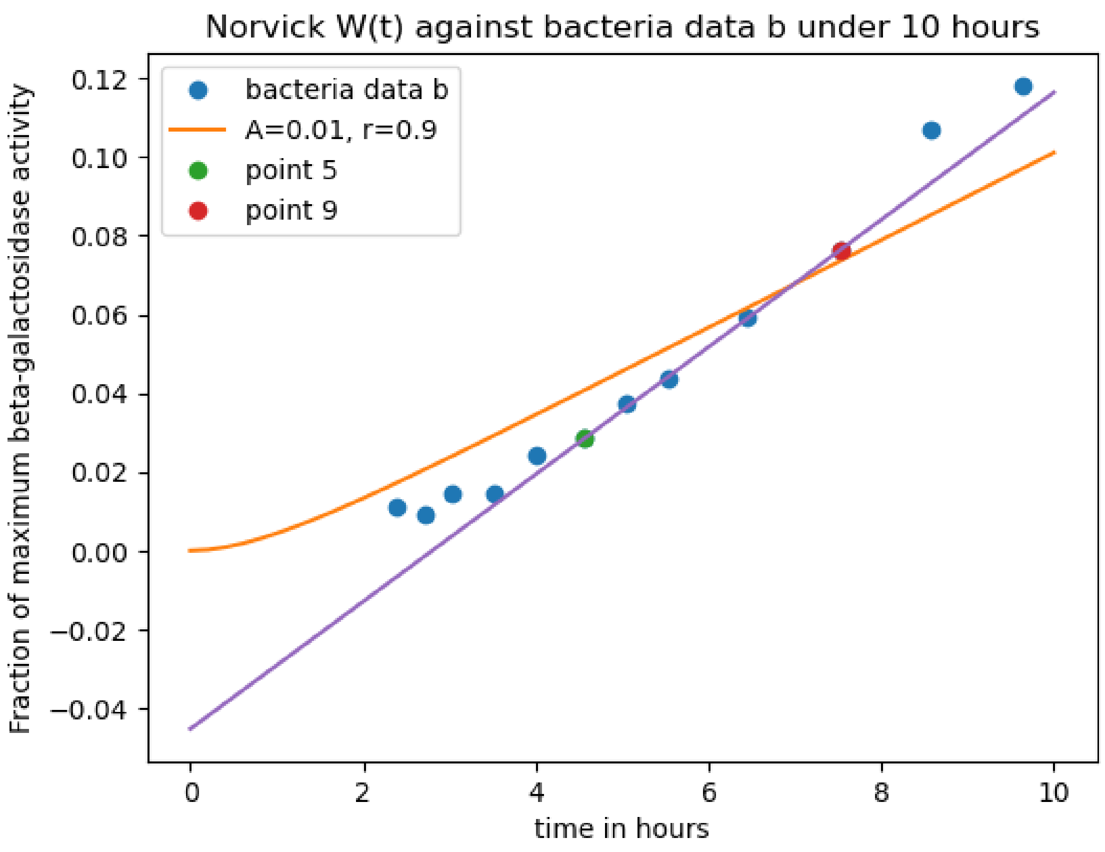
```
[43]: array([-0.04526126, -0.04196228, -0.03866329, -0.03536431, -0.03206532,
             -0.02876633, -0.02546735, -0.02216836, -0.01886938, -0.01557039,
             -0.0122714 , -0.00897242, -0.00567343, -0.00237444,  0.00092454,
              0.00422353,  0.00752251,  0.0108215 ,  0.01412049,  0.01741947,
              0.02071846,  0.02401744,  0.02731643,  0.03061542,  0.0339144 ,
              0.03721339,  0.04051238,  0.04381136,  0.04711035,  0.05040933,
              0.05370832,  0.05700731,  0.06030629,  0.06360528,  0.06690426,
              0.07020325,  0.07350224,  0.07680122,  0.08010021,  0.08339919,
              0.08669818,  0.08999717,  0.09329615,  0.09659514,  0.09989413,
              0.10319311,  0.1064921 ,  0.10979108,  0.11309007,  0.11638906])
```

```
[44]: plt.plot(t, y_t)
```

```
[44]: [<matplotlib.lines.Line2D at 0x10efe5950>]
```



[45]:
```
# plot y=mx + b over w(t) and bacteria data b excluding time values greater␣
↪than 10 hours

A = 0.01
r = 0.9
plt.close()
plt.plot(sliced_b_time, sliced_b_y, 'o', label="bacteria data b")


t = np.linspace(0,10,50)
y = m * t
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")
plt.plot(sliced_b_time[5], sliced_b_y[5],'o', label="point 5")
plt.plot(sliced_b_time[9], sliced_b_y[9],'o', label="point 9")
plt.title("Norvick W(t) against bacteria data b under 10 hours")
plt.plot(t, y_t)
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")
plt.show()
```



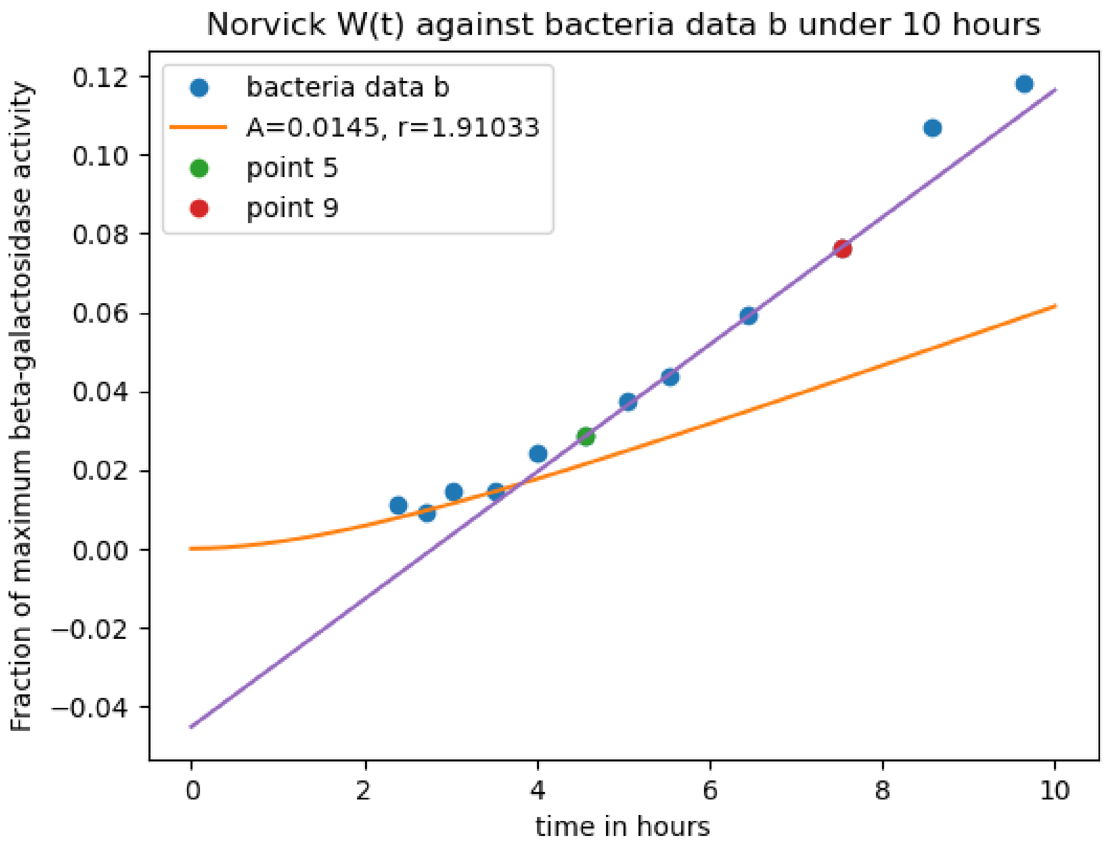
[54]:
```
# print out recorded data point again
print(f"point 3 in sliced data:\n({sliced_b_time[3]}, {sliced_b_y[3]})")
```

```
point 3 in sliced data:
(3.5177, 0.0145)
```

Using the data above we can solve w(t) for A and tau and we get r = 1.91033 and r = -3.06903 and A = 0.0145 which can be seen on the plot below

[55]:
```
# better values for A and tau?
A = 0.0145
# r = 1.91033
r - -3.06903
plt.close()
plt.plot(sliced_b_time, sliced_b_y, 'o', label="bacteria data b")


t = np.linspace(0,10,50)
y = m * t
```

```
w_eval = bact_w(t, r, A)
plt.plot(t, w_eval, label=f"A={A}, r={r}")
plt.plot(sliced_b_time[5], sliced_b_y[5],'o', label="point 5")
plt.plot(sliced_b_time[9], sliced_b_y[9],'o', label="point 9")
plt.title("Norvick W(t) against bacteria data b under 10 hours")
plt.plot(t, y_t)
plt.legend()
plt.xlabel("time in hours")
plt.ylabel("Fraction of maximum beta-galactosidase activity")


plt.show()
```



[38]:
```
# TODO: move hiv v(t) to a function and define it below the markdown latex cell␣
↪is this better than specifying the function at each evaluation like it␣
↪suggests in the book? probably
# TODO: can i link to wolframalpha solution for alpha
# TODO: remove redundancy in plotting code to vary parameter values lots of␣
↪retyped code for each plot abstract redundancy to individual methods
```