Tutorial 4 Data fitting for Reactor Design

March 1, 2018

0.1 Data fitting

As you may have noticed from your labs the world works a bit different from the textbook. In textbooks you are usually asked to predict what your reactor will give you if you know the rate of the reaction. In real world, sometimes we want to estimate the rate of the reaction by running the experiment and analyzing the concentrations. To do that we need to learn how to fit our data.

Hopefully, you are already persuaded and eager to learn how to use Python to fit your data, if not - please go back to the Dr. Yadav's Lecture 8.

So the here is the problem we want to solve:

You and your friend have an assignment from your boss to analyze how fast a secret chemical A can react into another secret chemical B.

$$A \rightarrow B$$

You and your buddy have run an experiment for the reaction in a batch reactor and got this data points below. We assume your friend likes Python as much as you do and as a good friend he has imported everything into Python arrays.

```
time_array = array([   0.,  125.,  250.,  375.,  500.,  625.,  750.,  875., 1000.,
       1125., 1250., 1375., 1500., 1625., 1750., 1875., 2000., 2125.,
       2250., 2375., 2500., 2625., 2750., 2875., 3000.])

C_A_array = array([10.28148623,  8.92676003,  7.47370739,  7.20498621,  6.01448975,
        4.85174028,  4.68868558,  4.01410753,  3.94055544,  3.04186757,
        2.40544184,  2.62290428,  2.18704979,  1.31502537,  1.03730156,
        2.28837979,  0.45062549,  1.43886825,  1.26808295,  1.24125421,
        0.28353151,  0.90117414,  0.74132173,  0.73710597, -0.07407658])
```

You know that `time_array` corresponds to time and `C_A_array` to $C_A(t)$.

You also know that it is the first order reaction, i.e:

$-r_A = kC_A$

Now $\frac{dC_A}{dt} = -kC_A$ and the solution is $C_A(t) = C_{A0} \cdot e^{-kt}$

Your boss says that you and your friend need to give him a reaction constant k otherwise your whole company won't be able to deliver a new product and will experience significant difficulties (especially after the recent tax reform). You suspect $C_{A0} = 10 moles$, but it would be nice to check that as well.

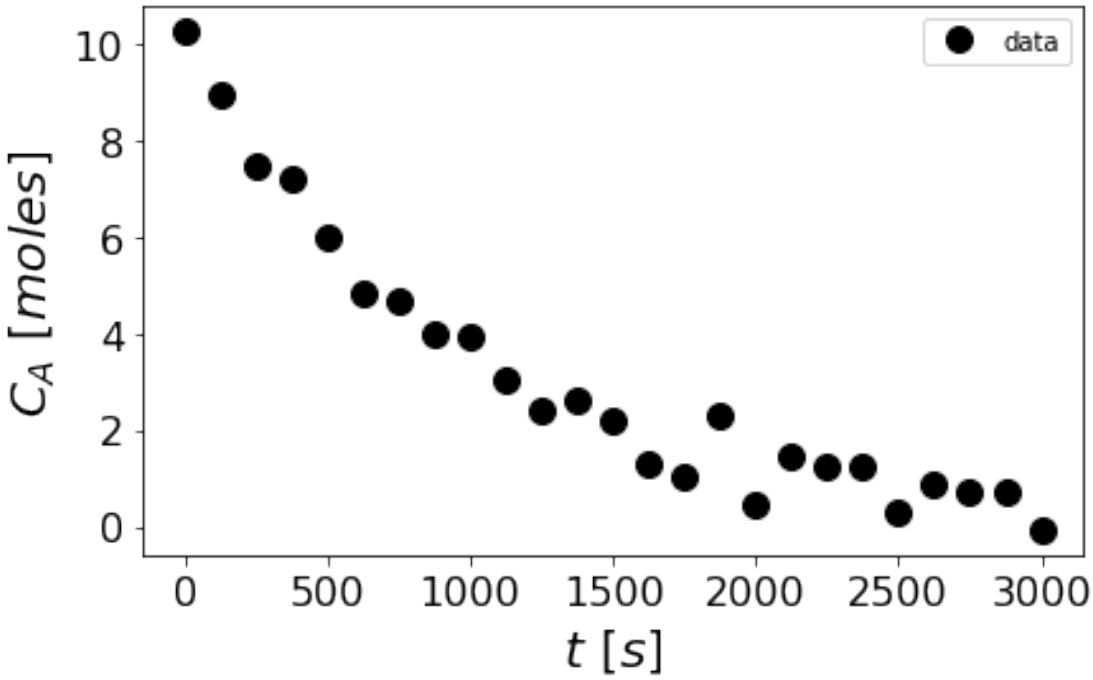
What should we do?

0.1.1 Plotting your data

The first thing you want to do is to plot your data. Lets do that using Python:

```
In [46]: import numpy as np # our matlab-like module
         import matplotlib.pyplot as plt # plotting modules
         plt.style.use('presentation') # just have in your script for prettier plotting
         # if 'presentation' doesn't work use 'seaborn' or 'ggplot'


         time_array = np.array([   0.,  125.,  250.,  375.,  500.,  625.,  750.,  875., 1000.,
                1125., 1250., 1375., 1500., 1625., 1750., 1875., 2000., 2125.,
                2250., 2375., 2500., 2625., 2750., 2875., 3000.])

         C_A_array = np.array([10.28148623,  8.92676003,  7.47370739,  7.20498621,  6.01448975,
                 4.85174028,  4.68868558,  4.01410753,  3.94055544,  3.04186757,
                 2.40544184,  2.62290428,  2.18704979,  1.31502537,  1.03730156,
                 2.28837979,  0.45062549,  1.43886825,  1.26808295,  1.24125421,
                 0.28353151,  0.90117414,  0.74132173,  0.73710597, -0.07407658])

         plt.plot(time_array, C_A_array, 'ko', label='data') #  ko = black o (black circles)

         plt.xlabel('$t\ [s]$') # we use $ $ to tell Python to use Latex you can just use plt.xl
         plt.ylabel('$C_A\ [moles]$') # here the slash \ creates a space between C_A and [moles]
         plt.legend() #brings up the legend
         plt.savefig('problem1.png') #saving the graph
         plt.show() # will show it below
```

```
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:2022: UserWarning: Th
  warnings.warn("This figure includes Axes that are not compatible "
```



0.2 How do we find k

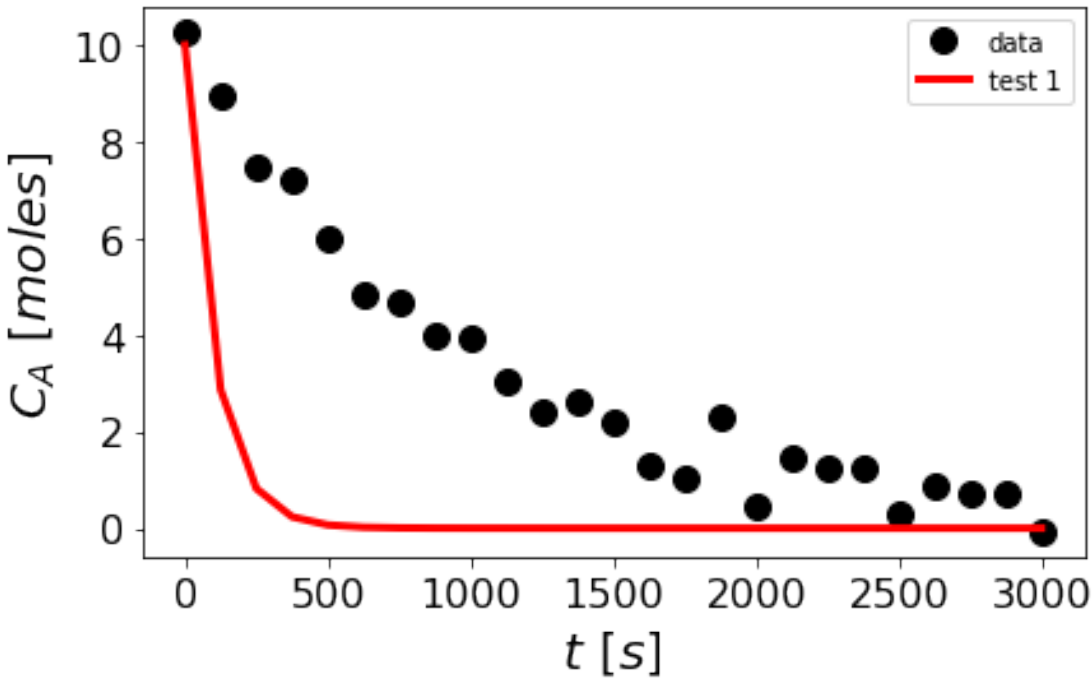
Ok! Looks something like an exponent. That is a good sign. But how do we find the k parameter???? Luckily, your friend have heard something about least squares method.

The idea is simple:

What if ... we take an arbitrary parameter $k = 0.01$ and see what it would give us? Lets see...

```
In [56]: k_test_1 = 0.01
         C_A_test_1_array = 10.*np.exp(-k_test_1*time_array) # this is our analytical solution

         # our data
         plt.plot(time_array, C_A_array, 'ko', label='data') #  ko = black o (black circles)
         # first guess
         plt.plot(time_array, C_A_test_1_array, 'r-', label='test 1') #  r- = solid red line

         plt.xlabel('$t\ [s]$') # we use $ $ to tell Python to use Latex you can just use plt.xl
         plt.ylabel('$C_A\ [moles]$') # here the slash \ creates a space between C_A and [moles]
         plt.legend()
         plt.show()

         #lets see what we get:
```

```
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:2022: UserWarning: Th
  warnings.warn("This figure includes Axes that are not compatible "
```



As we can see the fit isn't that great! To know the exact value we can calculate the squared difference between each data point and our predicted curve (red curve and black data) and call this parameter χ^2:

$$\chi^2 = \Sigma(data[i] - test[i])^2 = \Sigma(black[i] - red[i])^2$$

If the χ^2 parameter is really small then we have a good fitting. So the game of curve fitting becomes the game of selecting the parameter k so it produces the smallest χ^2 parameter.

Lets see what is it equal to:

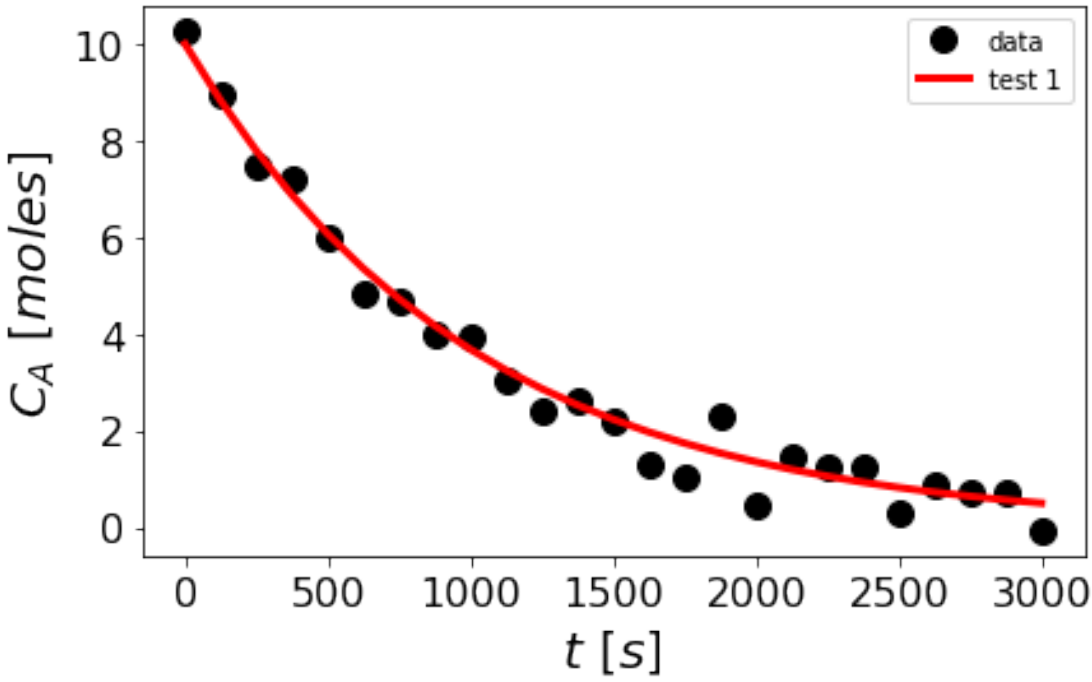
```
In [57]: # Lets go over each value of the both arrays and add the difference into the chi2
         chi2 = 0
         for i in range(len(C_A_array)):
             chi2 = chi2 + (C_A_array[i] - C_A_test_1_array[i])**2
         print('chi2 = {0}'.format(chi2))
         # you can just use print(chi2) if you want.
         # chi2 = 284.0954414312236 - that is a lot!

chi2 = 284.0954414312236
```

After many iterations of choosing k we came to a better guess value $k = 0.001$. Lets try it out:

```
In [53]: k_test_2 = 0.001
         C_A_test_2_array = 10.*np.exp(-k_test_2*time_array) # this is our analytical solution
```

```
        # our data
        plt.plot(time_array, C_A_array, 'ko', label='data') #  ko = black o (black circles)
        # first guess
        plt.plot(time_array, C_A_test_2_array, 'r-', label='test 1') #  r- = solid red line


        plt.xlabel('$t\ [s]$')
        plt.ylabel('$C_A\ [moles]$')

        plt.legend()
        plt.show()

        #lets see what we get:
```

```
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:2022: UserWarning: Th
  warnings.warn("This figure includes Axes that are not compatible "
```



Wow! That is perfect! lets see what χ^2 is equal to:

```
In [58]: chi2 = 0
         for i in range(len(C_A_array)):
             chi2 = chi2 + (C_A_array[i] - C_A_test_2_array[i])**2
         print('chi2 = {0}'.format(chi2))
```

```
chi2 = 4.102717065639098
```

Now it looks like something small - (and hence the graph looks to fit the data). In the ideal world of no random noises, $\chi^2 = 0$.

Now lets see how we can do the same thing using the stuff our friends from SciPy prepared for us:

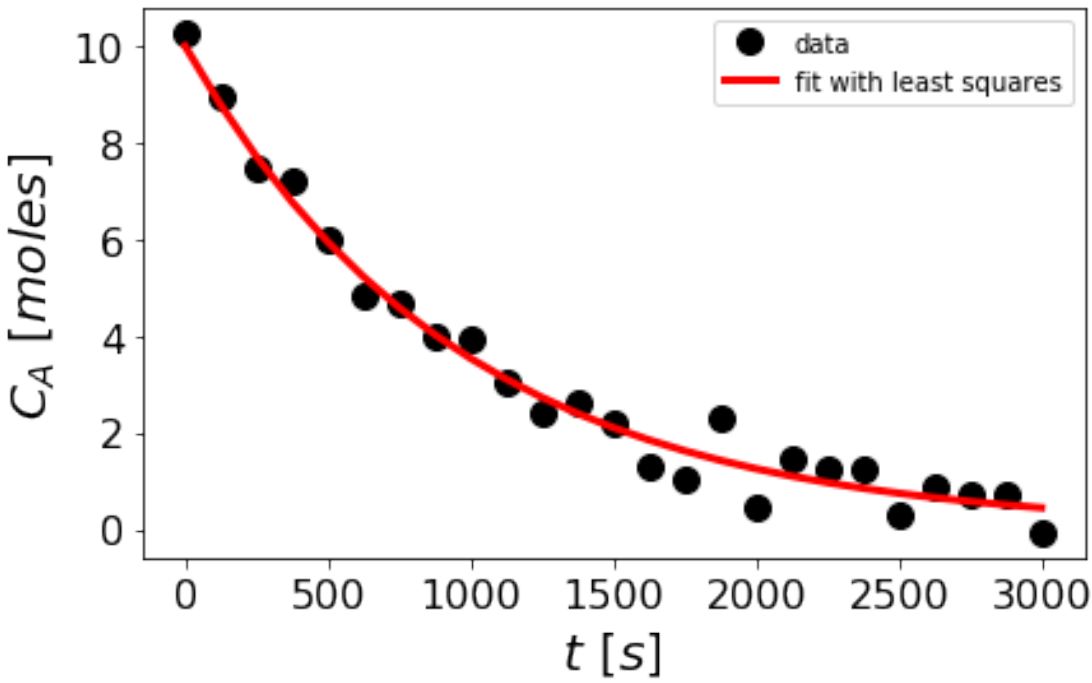
0.3 Solving everything using SciPY - easy to use fitting

```
In [82]: # step 0: importing all the jibber-jabber modules
         import numpy as np
         import matplotlib.pyplot as plt
         from scipy.optimize import curve_fit # this guy will help us fit everything

         #step 1: define the function you want to fit:

         def func(t, k):
             return 10. * np.exp(-k * t)


         # step 2: define the arrays you want to fit

         time_array = np.array([   0.,  125.,  250.,  375.,  500.,  625.,  750.,  875., 1000.,
                1125., 1250., 1375., 1500., 1625., 1750., 1875., 2000., 2125.,
                2250., 2375., 2500., 2625., 2750., 2875., 3000.])

         C_A_array = np.array([10.28148623,  8.92676003,  7.47370739,  7.20498621,  6.01448975,
                 4.85174028,  4.68868558,  4.01410753,  3.94055544,  3.04186757,
                 2.40544184,  2.62290428,  2.18704979,  1.31502537,  1.03730156,
                 2.28837979,  0.45062549,  1.43886825,  1.26808295,  1.24125421,
                 0.28353151,  0.90117414,  0.74132173,  0.73710597, -0.07407658])

         # step 3 run the curve fitting:

         popt, pcov = curve_fit(func, time_array, C_A_array, p0=(0.1))
         # parameters optimized, parameters covariance matrix (we don't need it for now) =
         # curve_fit(our function, our x data, our y data, p0=(our initial guess for the paramet

         # step 4 print your parameters

         print('Huraah! our parameter k = {0}'.format(popt))

         # step 5 plot your results:

         plt.plot(time_array, C_A_array, 'ko', label='data')
         plt.plot(time_array, func(time_array, popt), 'r-', label='fit with least squares')
         plt.xlabel('$t\ [s]$')
         plt.ylabel('$C_A\ [moles]$')

         plt.legend()
```

```
        plt.show()

        #step 6 report to your boss
```

```
Huraah! our parameter k = [0.00103939]
```

```
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:9: RuntimeWarning: o
  if __name__ == '__main__':
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:2022: UserWarning: Th
  warnings.warn("This figure includes Axes that are not compatible "
```



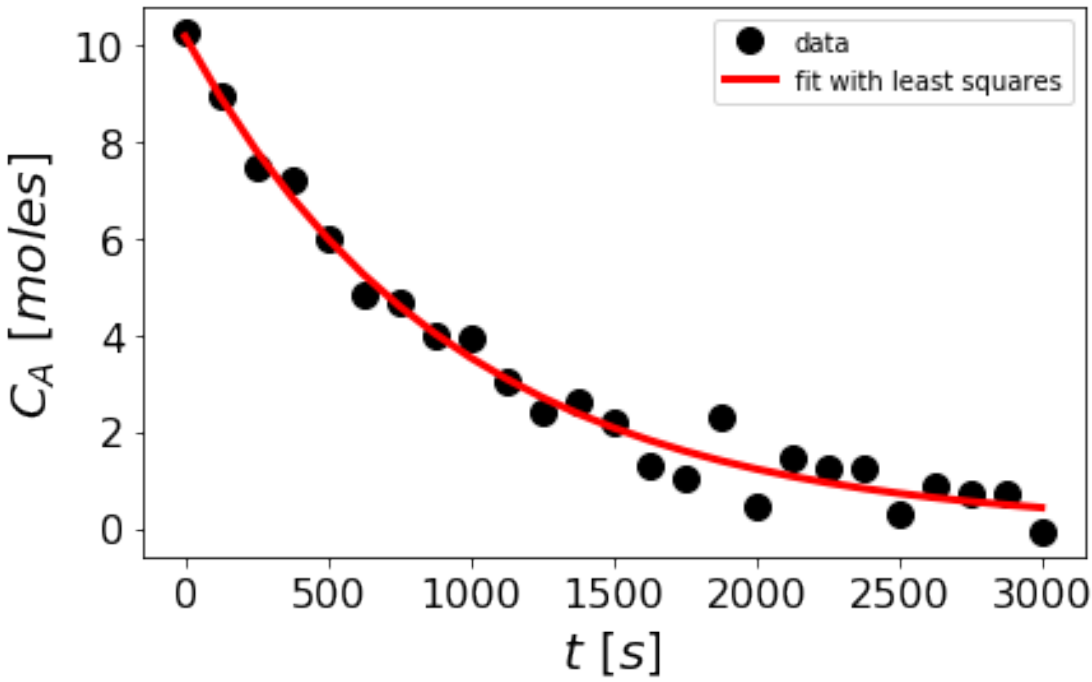
Ok great! We have saved our company and we have found the paramter. We were also asked to find C_A0 which supposed to be around $\sim$ 10. How do we do that? Ok, lets use our 6-step process:

```
In [85]: # step 0: importing all the jibber-jabber modules
         import numpy as np
         import matplotlib.pyplot as plt
         from scipy.optimize import curve_fit # this guy will help us fit everything

         #step 1: define the function you want to fit:

         # here you can see I have two parameters, so the fitted popt will have two as well
         def func(x, C_A0, k):
```

```
    return C_A0 * np.exp(-k * x)


# step 2: define the arrays you want to fit

time_array = np.array([   0.,  125.,  250.,  375.,  500.,  625.,  750.,  875., 1000.,
       1125., 1250., 1375., 1500., 1625., 1750., 1875., 2000., 2125.,
       2250., 2375., 2500., 2625., 2750., 2875., 3000.])

C_A_array = np.array([10.28148623,  8.92676003,  7.47370739,  7.20498621,  6.01448975,
        4.85174028,  4.68868558,  4.01410753,  3.94055544,  3.04186757,
        2.40544184,  2.62290428,  2.18704979,  1.31502537,  1.03730156,
        2.28837979,  0.45062549,  1.43886825,  1.26808295,  1.24125421,
        0.28353151,  0.90117414,  0.74132173,  0.73710597, -0.07407658])

# step 3 run the curve fitting:

popt, pcov = curve_fit(func, time_array, C_A_array, p0=(8, 0.1))
# curve_fit(our function, our x data, our y data, p0=(our initial guesses for C_A0 and
# since from the graph we kind of see C_A0 is around 8-10

# step 4 print your parameters

print('Huraah! our parameter C_A0 = {0}, k = {1}'.format(popt[0], popt[1]))

# step 5 plot your results:

plt.plot(time_array, C_A_array, 'ko', label='data')
plt.plot(time_array, func(time_array, popt[0], popt[1]), 'r-', label='fit with least sq
plt.xlabel('$t\ [s]$')
plt.ylabel('$C_A\ [moles]$')

plt.legend()
plt.show()

#step 6 report to your boss
```

```
Huraah! our parameter C_A0 = 10.176954832679453, k = 0.0010583507580345618
```

```
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:10: RuntimeWarning:
  # Remove the CWD from sys.path while we load stuff.
/Users/bazilevs/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py:2022: UserWarning: Th
  warnings.warn("This figure includes Axes that are not compatible "
```



Last remarks: there are 2 main things to consider when fitting the data:

1 What kind of model you are fitting your data with. Make sure you use your engineering 6-th sense for that

2 What kind of parameters we are expecting. So from plotting the data we can see that the half-life of our curve is around $1/500$ seconds - so our initial parameter should be around 0.002. Sometimes hitting the right initial parameter could be really important!

1 Home work:

Your boss asked you to try another reaction and get parameters from there.

$$2A -> B$$

Your boss told you that it is probably a second order reaction $-r_A = k[C_A]^2$.

Your friend has helped you out with the design equation and the exact solution for the current system:

$dC_A/dt = -kC_A = -kC_A^2$

The exact solution here would be

$1/C_A = kt + 1/C_{A0}$

Your friend suggested that it would be better to plot $1/C_A$ instead of C_A. You think it is a great idea. Conveniently, your data is presented in the neccesseary format:

Time:

```
time_array = array([ 0.        ,  0.41666667,  0.83333333,  1.25      ,  1.66666667,
        2.08333333,  2.5       ,  2.91666667,  3.33333333,  3.75      ,
```

```
       4.16666667,  4.58333333,  5.        ,  5.41666667,  5.83333333,
       6.25      ,  6.66666667,  7.08333333,  7.5       ,  7.91666667,
       8.33333333,  8.75      ,  9.16666667,  9.58333333, 10.        ])
```

and $1/C_A(time)$:

```
CA_inv_array = array([0.09955774, 0.10041539, 0.10164106, 0.10040534, 0.10151214,
       0.1027833 , 0.10252228, 0.10259392, 0.10369246, 0.10371607,
       0.10400771, 0.10517007, 0.10549644, 0.10558765, 0.10639114,
       0.10600782, 0.10506377, 0.10734534, 0.10655902, 0.10820498,
       0.1077726 , 0.10985819, 0.10912644, 0.11053292, 0.10939707])
```

Your task is to find parameters k and C_A0 from the experimental data.