jnk3_activation

March 12, 2018

1 Importing python libraries and the PySB model.

```
In [1]: # Importing libraries
        from jnk3_no_ask1 import model
        import numpy as np
        import matplotlib.pyplot as plt
        from pysb.simulator import ScipyOdeSimulator
        import pandas as pd
```

2 Here we load the fitted parameters that we obtained using the Particle Swarm Optimization (PSO). To obtain this parameter set we used the two sets that Nicole gave us. The first set has the following proteins:

- Arrestin
- MKK4
- MKK7
- JNK3

3 The second set of experiments has the following proteins:

- MKK4
- MKK7
- JNK3

4 Using this experimental data we defined a cost function whose value is roughly the squared difference between the experimental data and the simulations. This gave us values for the kinetic parameters of the model

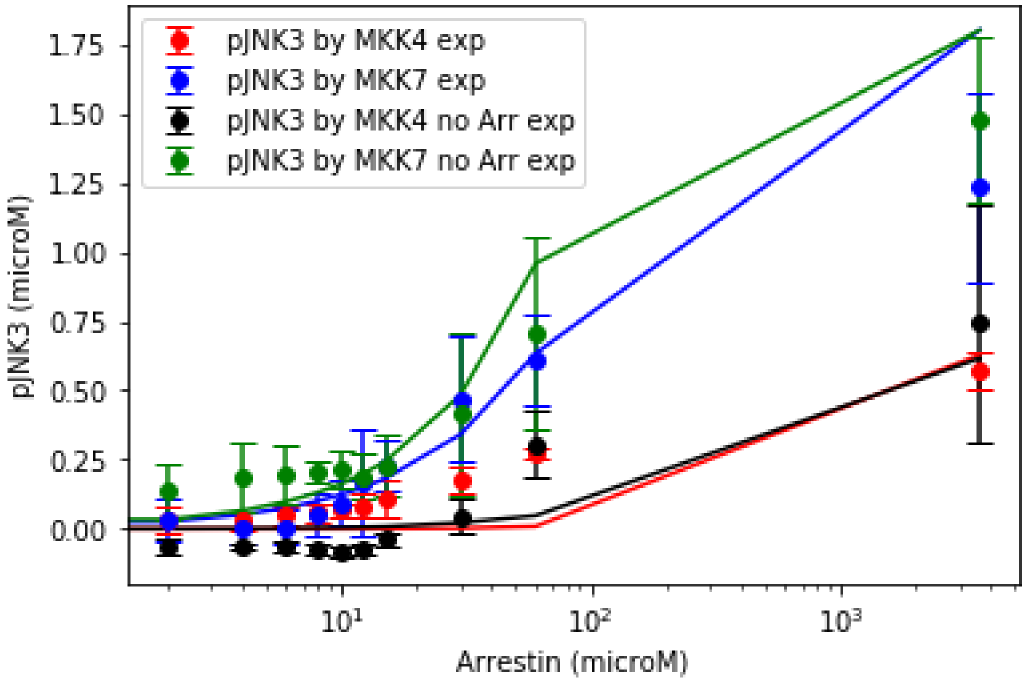
5 Note: The model calibration was done assuming that all the reactions are diffusion limited. Which means that we set all the k_forward (k_on) to 1.5E4 inversed microM * s

```
In [2]: # Loading fitted parameters
        param_values = np.array([p.value for p in model.parameters])
        idx_pars_calibrate = [15, 17, 19, 20, 21, 22, 23, 24, 25, 26, 27, 29, 31, 33, 35]
        rates_of_interest_mask = [i in idx_pars_calibrate for i, par in enumerate(model.paramete

        fitted_pars = np.load('jnk3_noASK1_calibrated_pars_arr_noarr.npy')
        param_values[rates_of_interest_mask] = 10 ** fitted_pars
```

6 Next, we present a plot of the results of the model calibration. The continuous lines represent the model simulations and the dots represent the experimental data.

```
In [3]: exp_data = pd.read_csv('../data/exp_data_arrestin_noarrestin.csv')

        tspan0 = [0, 2, 4, 6, 8, 10, 12, 15, 30, 60, 3600]
        solver0 = ScipyOdeSimulator(model, tspan=tspan0)
        sim = solver0.run(param_values=param_values).all
        arrestin_idx = [36]
        plt.semilogx(exp_data['Time (secs)'].values, sim['pTyr_jnk3'], color='red')
        plt.errorbar(exp_data['Time (secs)'].values, exp_data['pTyr_arrestin_avg'].values,
                     exp_data['pTyr_arrestin_std'].values,
                     linestyle='None', marker='o', capsize=5, color='red', label='pJNK3 by MKK4
        plt.semilogx(exp_data['Time (secs)'].values, sim['pThr_jnk3'], color='blue')
        plt.errorbar(exp_data['Time (secs)'].values, exp_data['pThr_arrestin_avg'].values,
                     exp_data['pThr_arrestin_std'].values,
                     linestyle='None', marker='o', capsize=5, color='blue', label='pJNK3 by MKK7

        param_values2 = np.copy(param_values)
        param_values2[arrestin_idx] = 0
        sim2 = solver0.run(param_values=param_values2).all

        plt.semilogx(exp_data['Time (secs)'].values, sim2['pTyr_jnk3'], color='black')
        plt.errorbar(exp_data['Time (secs)'].values, exp_data['pTyr_noarrestin_avg'].values,
                     exp_data['pTyr_noarrestin_std'].values,
                     linestyle='None', marker='o', capsize=5, color='black', label='pJNK3 by MKK
        plt.semilogx(exp_data['Time (secs)'].values, sim2['pThr_jnk3'], color='green')
        plt.errorbar(exp_data['Time (secs)'].values, exp_data['pThr_noarrestin_avg'].values,
                     exp_data['pThr_noarrestin_std'].values,
                     linestyle='None', marker='o', capsize=5, color='green', label='pJNK3 by MKK

        plt.xlabel('Arrestin (microM)')
        plt.ylabel('pJNK3 (microM)')
        plt.legend()
```

```
Out[3]: <matplotlib.legend.Legend at 0x10a5cf590>
```

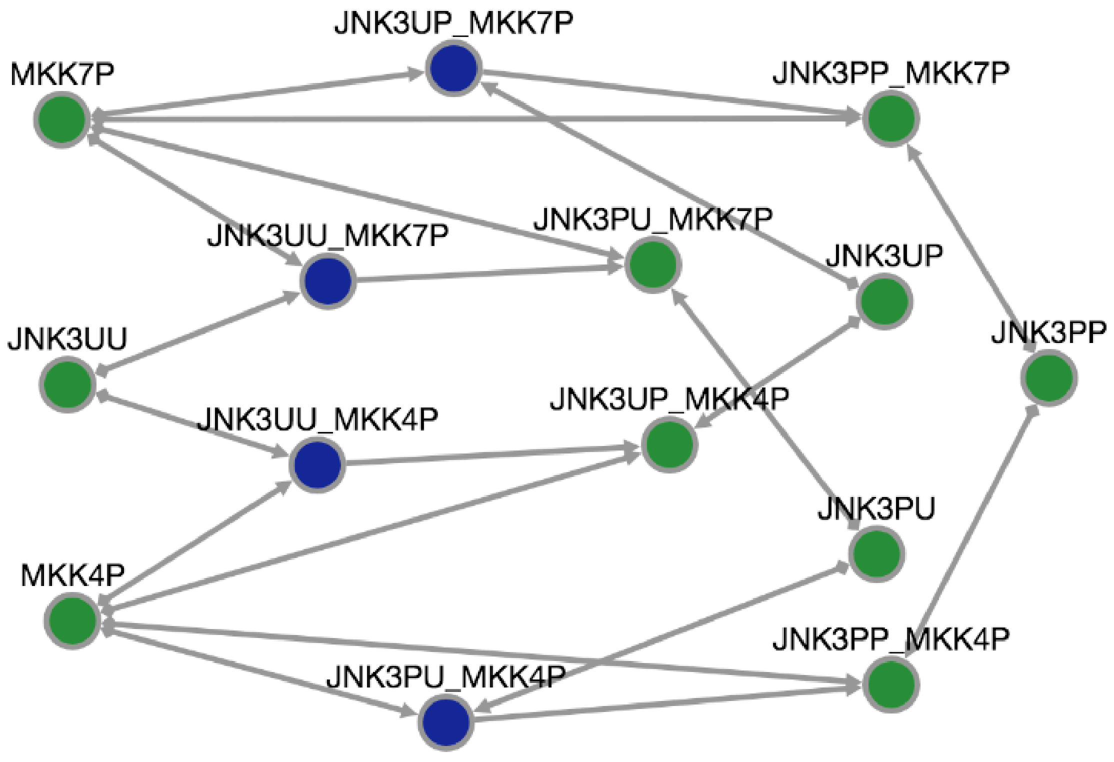


7 From here, we are going to be doing the analysis using the simplest model that consists in having JNK3, MKK4, and MKK7 interacting. This would correspond to the last experimental data set that Nicole sent us. We used this model because it is easier to visualize its network of interactions and it still allows us to study the order of the JNK3 phosphorilation.

```
In [4]: from jnk3_noask1_noarrestin import model as model_simple
        from tropical.cytoscapejs_visualization.model_visualization import ModelVisualization
```

```
In [5]: from IPython.display import Image
        Image(filename='noask1_noarrestin_network.png')
```

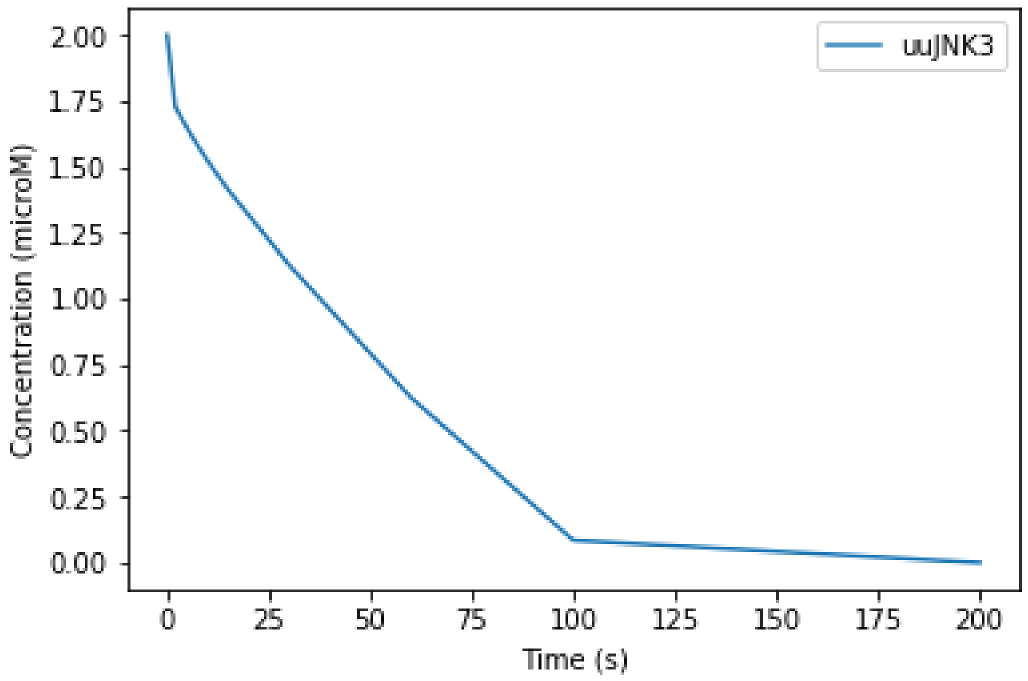
Out[5]:



```
In [6]: % matplotlib inline
        # Solving the PySB model to obtain the trajectories of the molecular species

        tspan = [0, 2, 4, 6, 8, 10, 12, 15, 30, 60, 100, 200, 500, 1000, 1500, 2000, 2500, 3000,
        solver = ScipyOdeSimulator(model_simple, tspan=tspan)
        param_values[36] = 0
        sim = solver.run(param_values=param_values).all
```

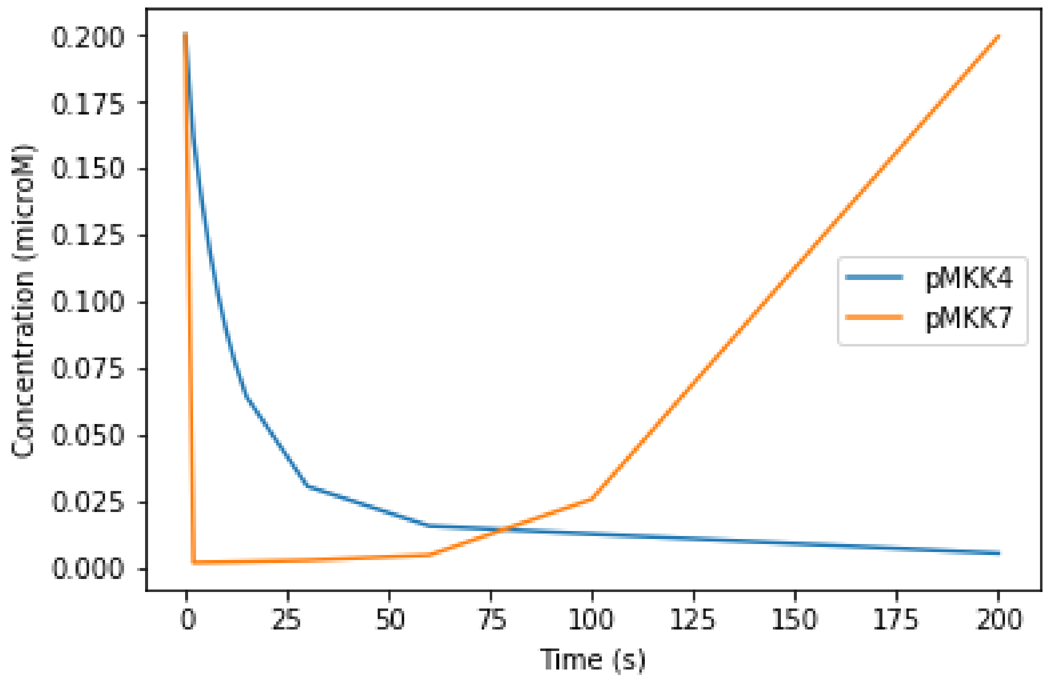
8 Figure 1. Here we show how inactive JNK3 is being used over time.

```
In [7]: plt.plot(tspan[:12], sim['__s2'][:12], label='uuJNK3')
        plt.legend()
        plt.xlabel('Time (s)')
        plt.ylabel('Concentration (microM)')

Out[7]: Text(0,0.5,u'Concentration (microM)')
```



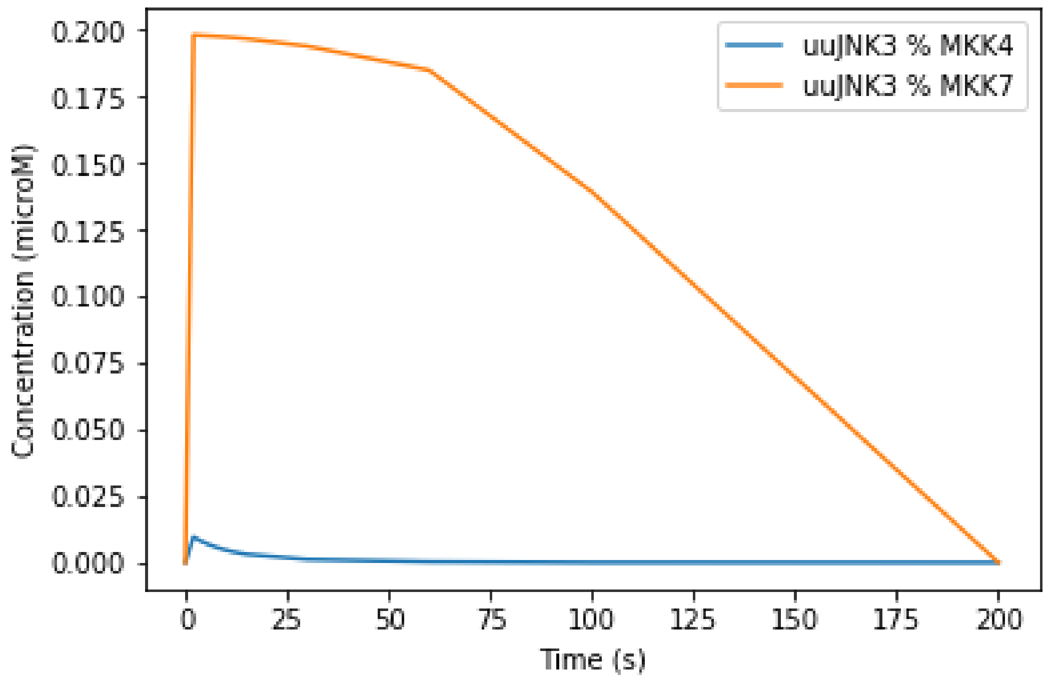
9 Figure 2. In this figure we can see how pMKK4 and pMKK7 are being used. pMKK7 gets rapidly consumed and then released, whereas pMKK4 it's slowly used and remains low, which means that it stays bound to the complexes.

```
In [8]: plt.plot(tspan[:12], sim['__s0'][:12], label='pMKK4')
        plt.plot(tspan[:12], sim['__s1'][:12], label='pMKK7')

        # plt.semilogx(tspan, sim['__s3'], label='uuJNK3')
        plt.legend()
        plt.xlabel('Time (s)')
        plt.ylabel('Concentration (microM)')

Out[8]: Text(0,0.5,u'Concentration (microM)')
```



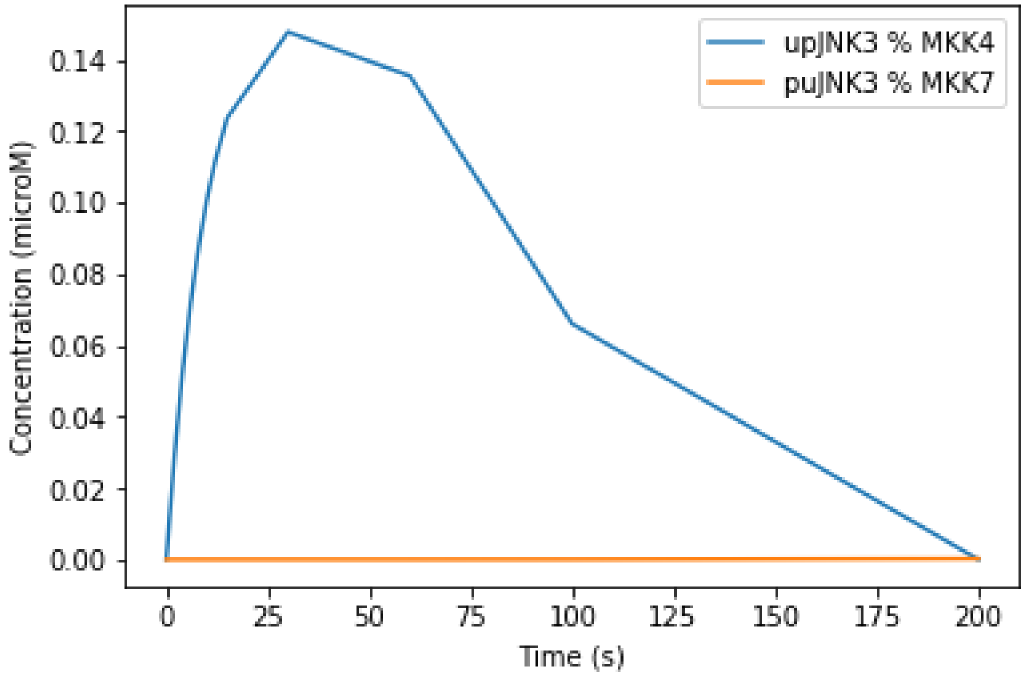
10 **Figure 3. In this figure we can see the dynamics of the complex formation of uuJNK3 with MKK4/7. From this, we can see that MKK7 rapidly binds to uuJNK3 and because the k_reverse = 234.50 inversed microM*s (dissociation constant) is small, the complex is more stable. On the other hand, for MKK4 we have k_reverse = 434163.36 which makes the MKK4%uuJNK3 complex more unstable and that's why it doesn't accumulate that much over time.**

```
In [9]: plt.plot(tspan[:12], sim['__s3'][:12], label='uuJNK3 % MKK4')
        plt.plot(tspan[:12], sim['__s4'][:12], label='uuJNK3 % MKK7')
        # plt.plot(tspan[:12], sim['__s6'][:12], label='upJNK3 % MKK4')
        # plt.plot(tspan, sim['__s7'], label='puJNK3 % MKK7')

        plt.legend()
        plt.xlabel('Time (s)')
        plt.ylabel('Concentration (microM)')

Out[9]: Text(0,0.5,u'Concentration (microM)')
```



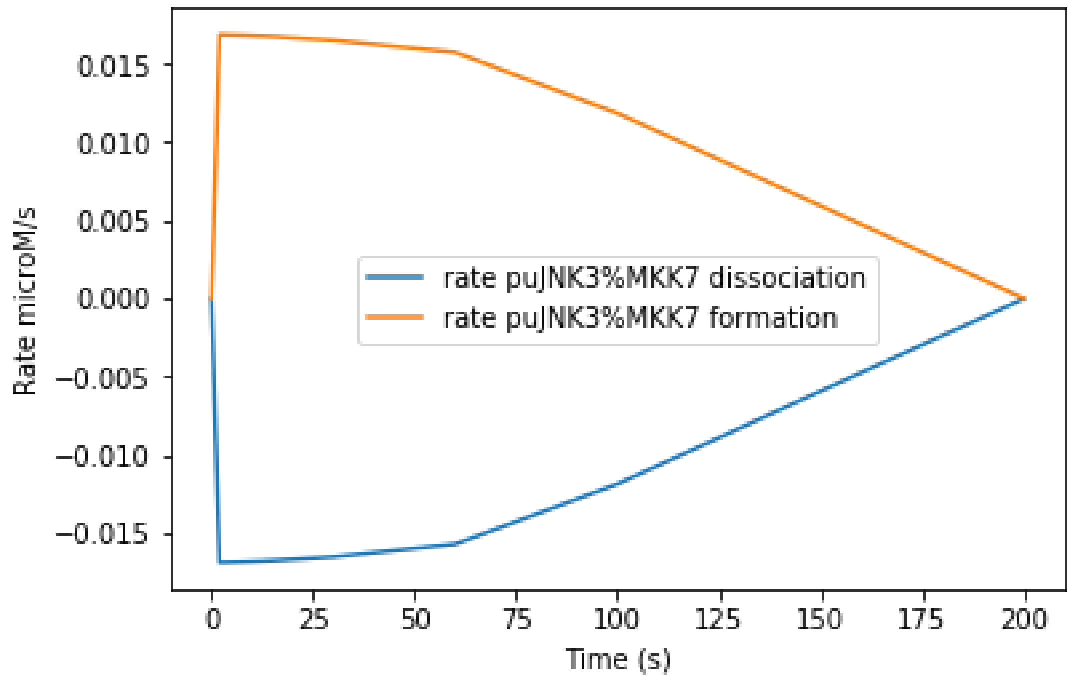
11 Figure 4. Here, we have that the puJNK3%MKK7 complex (Thr phosphorylation by MKK7) doesn't accumulate because the rate of creation is similiar to the rate of consumption as we can see in figure 5. On the other hand, the upJNK3%MKK4 complex (Tyr phosphorylation by MKK4) accumulates over time and then its concentration decreases when the dissociation rate is larger than the creation rate as we can see in figure 6

```
In [10]: plt.plot(tspan[:12], sim['__s5'][:12], label='upJNK3 % MKK4')
         plt.plot(tspan[:12], sim['__s6'][:12], label='puJNK3 % MKK7')

         plt.legend()
         plt.xlabel('Time (s)')
         plt.ylabel('Concentration (microM)')

Out[10]: Text(0,0.5,u'Concentration (microM)')
```



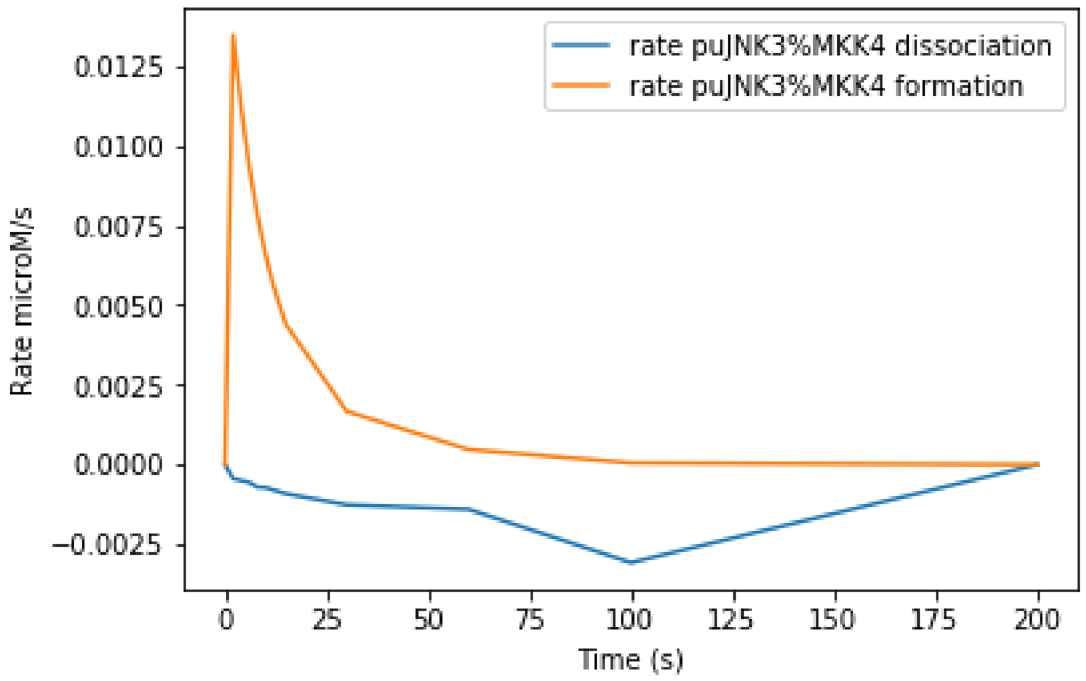
12 Figure 5. Comparation of the formation and dissociation rates of the puJNK3%MKK7 complex.

```
In [11]: plt.plot(tspan[:12], sim['__s1'][:12]*sim['__s8'][:12]*15000.0-sim['__s6'][:12]*1863107
         plt.plot(tspan[:12], sim['__s4'][:12]*0.08513195834607554, label='rate puJNK3%MKK7 form
         plt.legend()
         plt.xlabel('Time (s)')
         plt.ylabel('Rate microM/s')

Out[11]: Text(0,0.5,u'Rate microM/s')
```



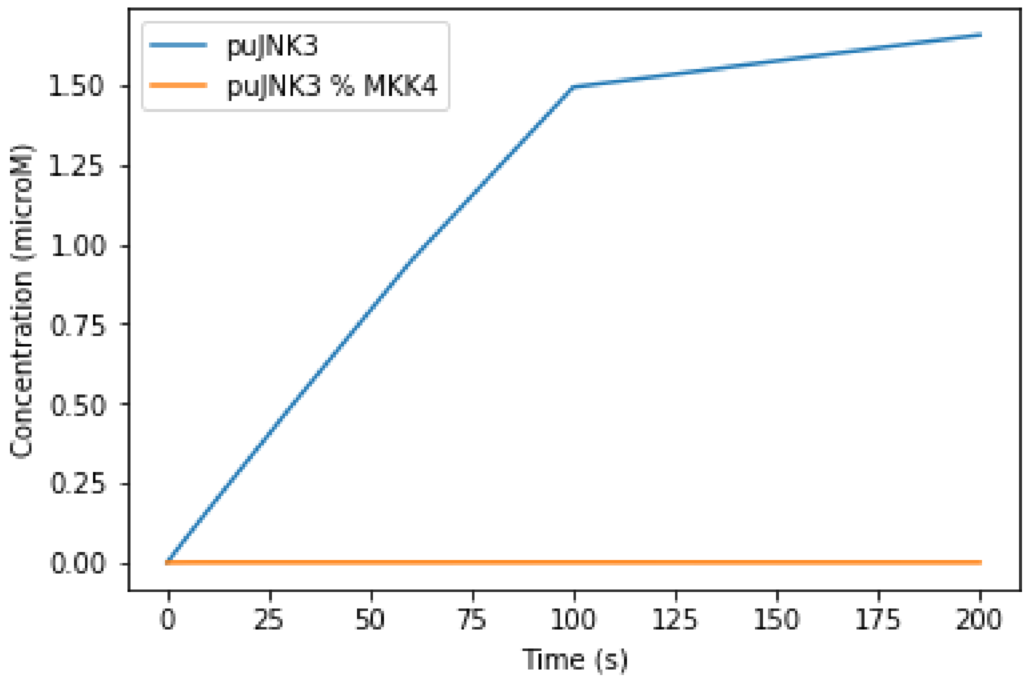
13 Figure 6. Comparation of the formation and dissociation rates of the upJNK3%MKK4 complex.

```
In [12]: plt.plot(tspan[:12], sim['__s0'][:12]*sim['__s7'][:12]*15000.0-sim['__s5'][:12]*61.0727
         plt.plot(tspan[:12], sim['__s3'][:12]*1.4005931958394613, label='rate puJNK3%MKK4 forma
         plt.legend()
         plt.xlabel('Time (s)')
         plt.ylabel('Rate microM/s')

Out[12]: Text(0,0.5,u'Rate microM/s')
```



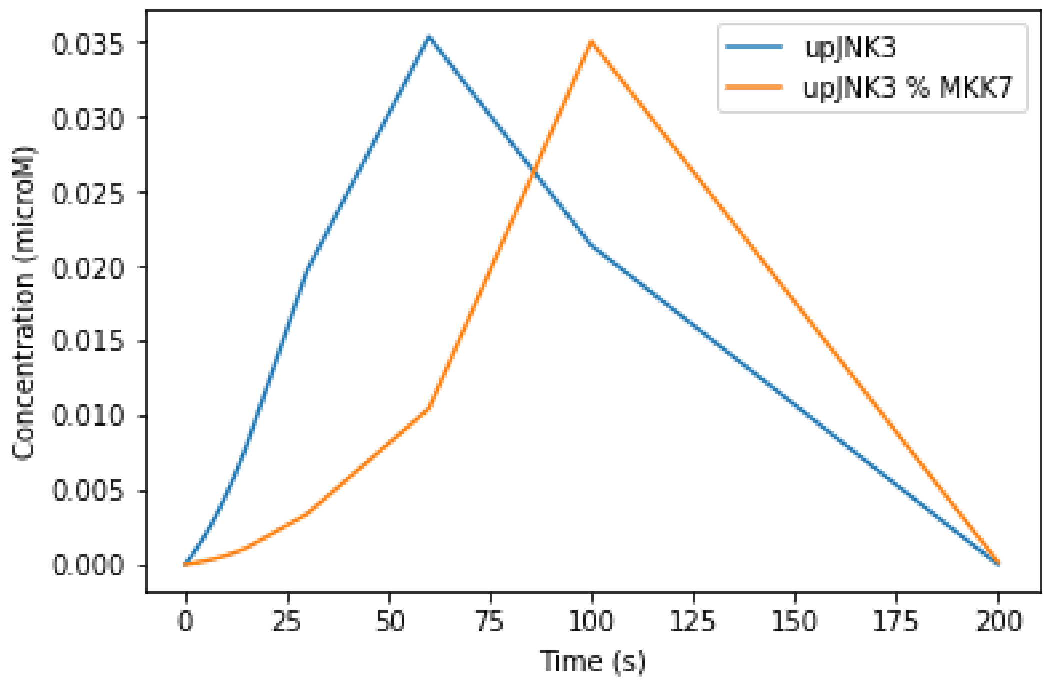
14 Figure 7. In this figure we see that puJNK3 gets dissociated and accumulated from the puJNK3%MKK7 complex faster than the upJNK3%MKK4 complex. This implies that MKK7 phosphorylates JNK3 before MKK4 does.

```
In [13]: plt.plot(tspan[:12], sim['__s8'][:12], label='puJNK3')
         plt.plot(tspan[:12], sim['__s9'][:12], label='puJNK3 % MKK4')

         plt.legend()
         plt.xlabel('Time (s)')
         plt.ylabel('Concentration (microM)')

Out[13]: Text(0,0.5,u'Concentration (microM)')
```



15 Figure 8. Here, we can see that there is a portion of the activated JNK3 where MKK4 phosphorylates JNK3 before MKK7 does.

```
In [14]: plt.plot(tspan[:12], sim['__s7'][:12], label='upJNK3')
         plt.plot(tspan[:12], sim['__s10'][:12], label='upJNK3 % MKK7')

         plt.legend()
         plt.xlabel('Time (s)')
         plt.ylabel('Concentration (microM)')

Out[14]: Text(0,0.5,u'Concentration (microM)')
```



16 Conclusions

- From the simulation analysis we can imply the order in JNK3 activation is that MKK7 first phosphorylates JNK3 at Thr and then MKK4 phosphorylates JNK3 at Tyr. Also, there is a small portion of the activated JNK3 where the order is reversed.
- Something that it is interesting and that I think would require more analysis or experiments is that when MKK4 binds inactive JNK3 it has a high k_reverse (~3 orders of magnitude difference), but when JNK3 is phosphorylated at one of the sites that k_reverse becomes very small. And, the opposite situation happens to MKK7.