

Scientific Computing Sheet 5

Brandon Tang

1.

```
1 // gillespie.h
2 #pragma once
3
4 #include <functional>
5 #include <random>
6 #include <vector>
7
8 struct Reaction {
9     std::function<double(std::vector<double>, double)> propensity;
10    // probability of reaction happening in [t, t+dt] = propensity(reactants,
    volume) * dt
11    std::vector<int> changes; // changes in reactants
12 };
13
14 std::pair<std::vector<double>, std::vector<std::vector<double>>> glllespie(double
t_final, std::vector<double> reactants, std::vector<Reaction> reactions, double
volume = 1.0);
```

```
1 #include <functional>
2 #include <random>
3 #include <vector>
4 #include <iostream>
5
6 #include "gillespie.h"
7 #include <cassert>
8
9 static constexpr double eps = 1e-12;
10 std::pair<std::vector<double>, std::vector<std::vector<double>>> glllespie(double
t_final, std::vector<double> reactants, std::vector<Reaction> reactions, double
volume) {
11     std::random_device rd; // obtain a random number from hardware
12     std::mt19937 gen(rd());
13     std::uniform_real_distribution<> dis(0.0, 1.0);
14
15     int nReactions = reactions.size();
16     int nSpecies = reactants.size();
17     std::vector<std::vector<double>> trajectory;
18     std::vector<double> times;
19     trajectory.push_back(reactants);
20     times.push_back(0);
21
22     double t = 0;
23     while (t < t_final) {
24         double r1 = dis(gen);
25         double r2 = dis(gen);
26
27         std::vector<double> cumulativePropensities(nReactions+1, 0);
28         for (int i = 0; i < nReactions; i++) {
29             cumulativePropensities[i+1] = reactions[i].propensity(reactants,
volume);
30             cumulativePropensities[i+1] += cumulativePropensities[i];
31         }
32         double a0 = cumulativePropensities[nReactions];
33         if (a0 < eps) {
34             // No more reactions possible
35             break;
```

```

36     }
37     double tau = (1.0 / a0) * log(1.0 / r1); // time to next reaction
38
39     // we binary search for the index (idx) of the last reaction such that
    such that tau * a0 >= cumulativePropensities[idx]
40     int idx = std::upper_bound(cumulativePropensities.begin(),
    cumulativePropensities.end(), r2 * a0) - 1 - cumulativePropensities.begin();
41     assert(idx >= 0 && idx < nReactions);
42
43     for (int i = 0; i < nSpecies; i++) {
44         reactants[i] += reactions[idx].changes[i];
45     }
46     t += tau;
47     trajectory.push_back(reactants);
48     times.push_back(t);
49 }
50
51 return {times, trajectory};
52 }

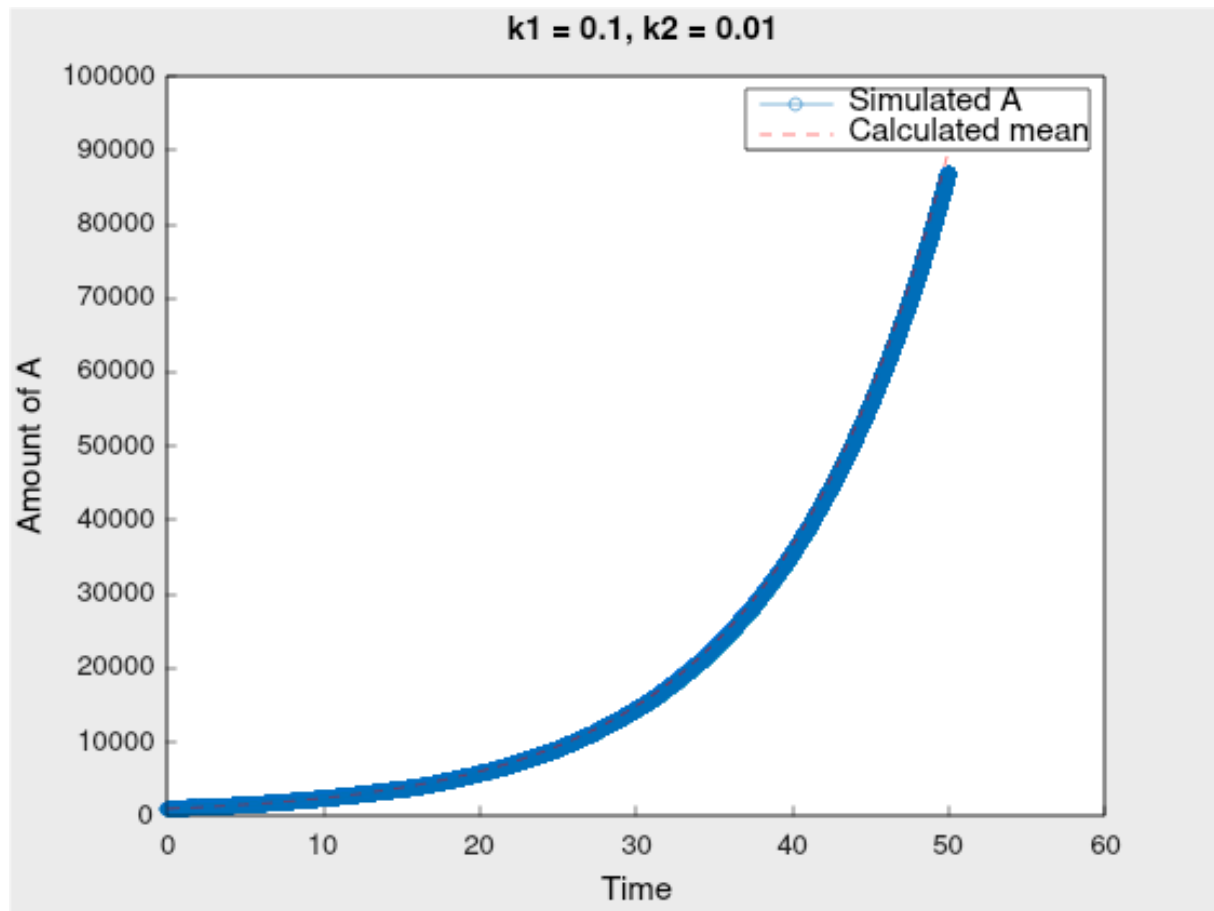
```

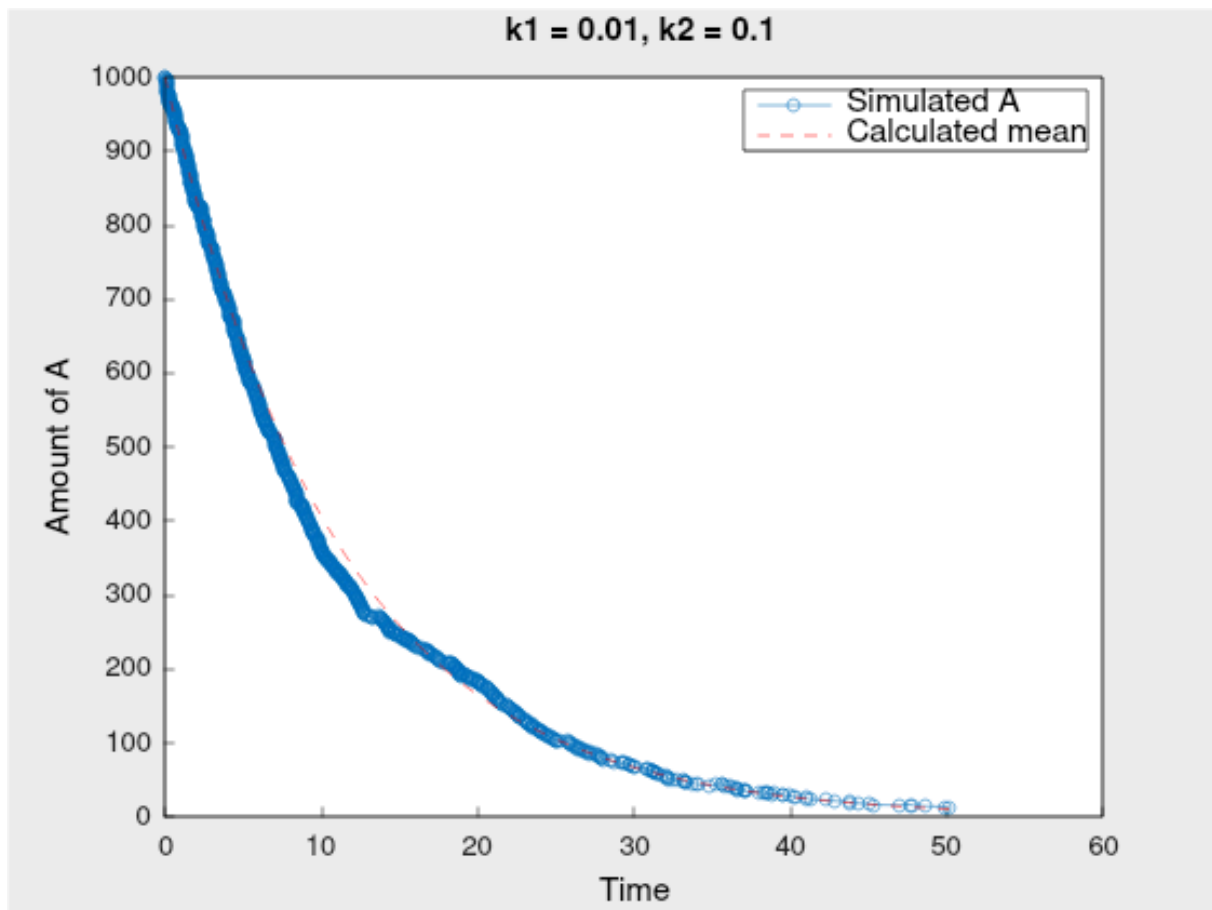
```

1  int main() {
2      using namespace matplotlib;
3
4      double A0 = 1000;
5      double k1 = 0.1;
6      Reaction dup = {
7          [](std::vector<double> reactants, double volume) { return k1 *
    reactants[0]; },
8          {1}};
9
10     double k2 = 0.01;
11     Reaction decay = {
12         [](std::vector<double> reactants, double volume) { return k2 *
    reactants[0]; },
13         {-1}};
14
15     std::vector<Reaction> reactions = {decay, dup};
16     std::vector<double> reactants = {A0};
17     auto [times, trajectory] = gillespie(50, reactants, reactions, 1.0);
18     double nTimePoints = trajectory.size();
19     std::vector<double> a_amt(nTimePoints);
20     for (int i = 0; i < nTimePoints; i++) {
21         a_amt[i] = trajectory[i][0];
22     }
23
24     auto compMean = [](double t) {
25         return A0 * exp((k1 - k2) * t);
26     };
27
28     std::vector<double> meanVal(nTimePoints);
29     for (int i = 0; i < nTimePoints; i++) {
30         meanVal[i] = compMean(times[i]);
31     }
32
33     title("k1 = 0.1, k2 = 0.01");
34     xlabel("Time");
35     ylabel("Amount of A");
36     plot(times, a_amt, "-o");
37     hold(on);
38     plot(times, meanVal, "--r");
39
40     ::matplotlib::legend({"Simulated A", "Calculated mean"});

```

```
41     show();  
42     return 0;  
43 }
```





From the graphs, we can see that indeed the mean seems to be correctly computed. The variance does seem to decrease to 0 when $k_1 < k_2$ as $t \rightarrow \infty$.