Scientific Computing Sheet 5

Brandon Tang

1.

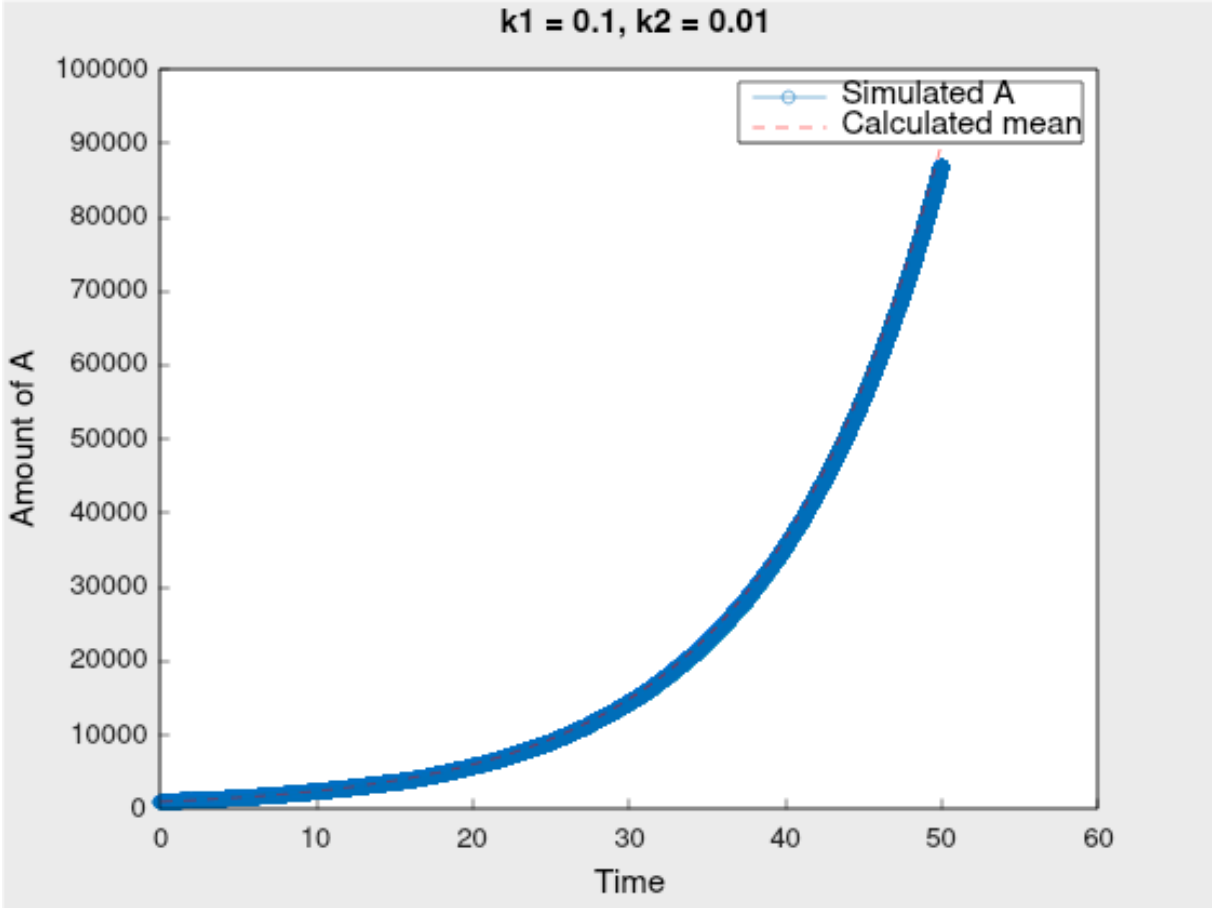
```cpp
// gillespie.h
#pragma once

#include <functional>
#include <random>
#include <vector>

struct Reaction {
    std::function<double(std::vector<double>, double)> propensity;
     // probability of reaction happening in [t, t+dt] = propensity(reactants, volume) * dt
    std::vector<int> changes;  // changes in reactants
};

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);
```

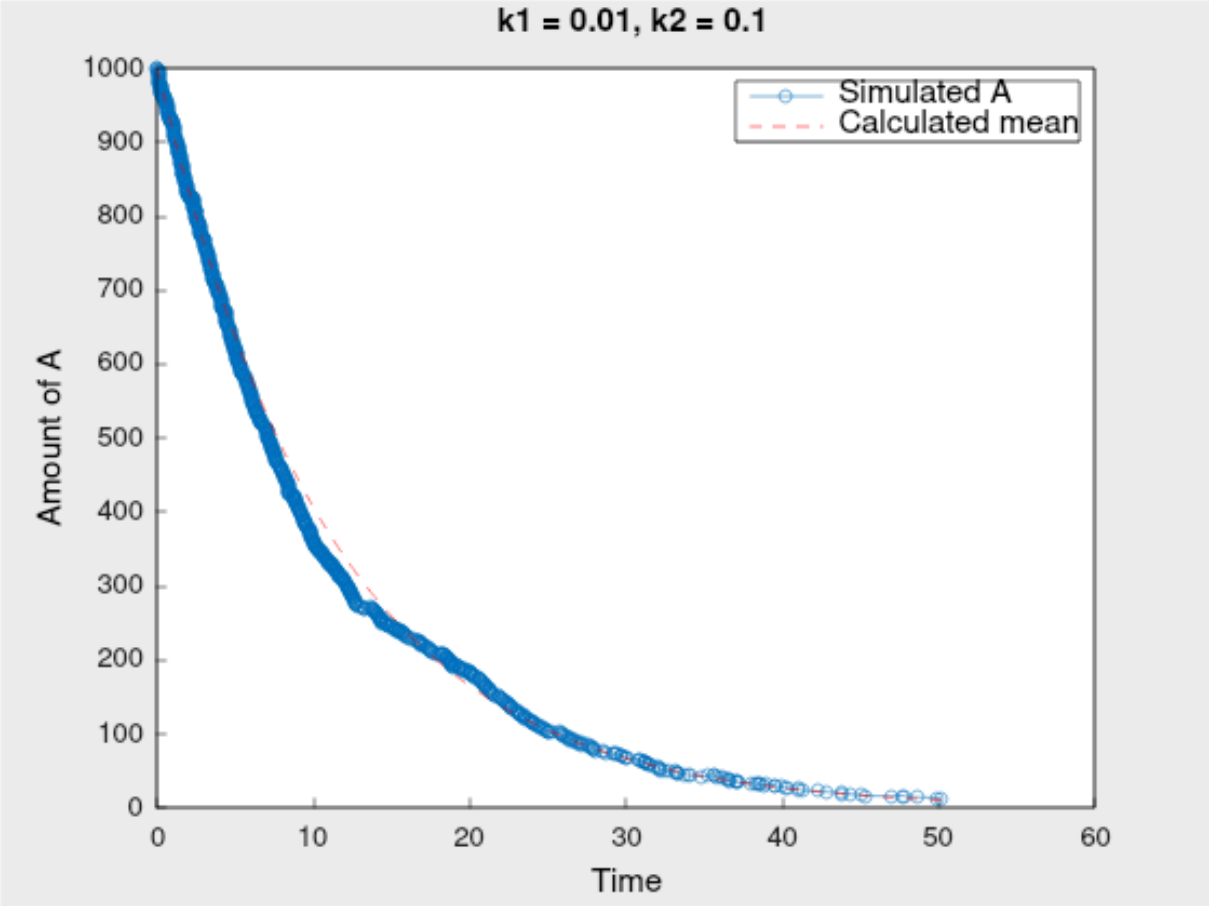
```cpp
#include <functional>
#include <random>
#include <vector>
#include <iostream>

#include "gillespie.h"
#include <cassert>

static constexpr double eps = 1e-12;
std::pair<std::vector<double>, std::vector<std::vector<double>>> glllespie(double t_final, std::vector<double> reactants, std::vector<Reaction> reactions, double volume) {
    std::random_device rd;  // obtain a random number from hardware
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(0.0, 1.0);

    int nReactions = reactions.size();
    int nSpecies = reactants.size();
    std::vector<std::vector<double>> trajectory;
    std::vector<double> times;
    trajectory.push_back(reactants);
    times.push_back(0);

    double t = 0;
    while (t < t_final) {
        double r1 = dis(gen);
        double r2 = dis(gen);

        std::vector<double> cumulativePropensities(nReactions+1, 0);
        for (int i = 0; i < nReactions; i++) {
                cumulativePropensities[i+1] = reactions[i].propensity(reactants, volume);
            cumulativePropensities[i+1] += cumulativePropensities[i];
        }
        double a0 = cumulativePropensities[nReactions];
        if (a0 < eps) {
            // No more reactions possible
            break;
```

```
        }
        double tau = (1.0 / a0) * log(1.0 / r1);  // time to next reaction

         // we binary search for the index (idx) of the last reaction such that such that tau * a0 >= cumulativePropensities[idx]
                    int   idx   =   std::upper_bound(cumulativePropensities.begin(), cumulativePropensities.end(), r2 * a0) - 1 - cumulativePropensities.begin();
        assert(idx >= 0 && idx < nReactions);

        for (int i = 0; i < nSpecies; i++) {
            reactants[i] += reactions[idx].changes[i];
        }
        t += tau;
        trajectory.push_back(reactants);
        times.push_back(t);
    }

    return {times, trajectory};
}
```

```
int main() {
    using namespace matplot;

    double A0 = 1000;
    double k1 = 0.1;
    Reaction dup = {
              [=](std::vector<double>  reactants,  double  volume)  {  return  k1  *  reactants[0]; },
        {1}};

    double k2 = 0.01;
    Reaction decay = {
              [=](std::vector<double>  reactants,  double  volume)  {  return  k2  *  reactants[0]; },
        {-1}};

    std::vector<Reaction> reactions = {decay, dup};
    std::vector<double> reactants = {A0};
    auto [times, trajectory] = glllespie(50, reactants, reactions, 1.0);
    double nTimePoints = trajectory.size();
    std::vector<double> a_amt(nTimePoints);
    for (int i = 0; i < nTimePoints; i++) {
        a_amt[i] = trajectory[i][0];
    }

    auto compMean = [=](double t) {
        return A0 * exp((k1 - k2) * t);
    };

    std::vector<double> meanVal(nTimePoints);
    for (int i = 0; i < nTimePoints; i++) {
        meanVal[i] = compMean(times[i]);
    }

    title("k1 = 0.1, k2 = 0.01");
    xlabel("Time");
    ylabel("Amount of A");
    plot(times, a_amt, "-o");
    hold(on);
    plot(times, meanVal, "--r");

    ::matplot::legend({"Simulated A", "Calculated mean"});
```

```
    show();
    return 0;
}
```




k1 = 0.01, k2 = 0.1
Simulated A
Calculated mean
Amount of A
Time
0
100
200
300
400
500
600
700
800
900
1000
0
10
20
30
40
50
60