David Curry
ID: 304755606
CEE/MAE M20
May 14, 2018

# Homework 6

## 1 The Shared Birthday Problem

### 1.1 Introduction

The purpose of this script is to determine the average minimum number of people in a group where 2 of their birthdays are within a week of each other. Assuming that there is an equal chance of being born on any given day, a random number generator can accomplish this task. In order to determine the average, I ran 10,000 iterations of the random number generator to get a more definitive answer. I also printed a histogram of the 10,000 data points to model the distribution.

### 1.2 Models and Methods

I first created a vector to store all the minimum numbers for all the 10,000 iterations and a vector to store the random numbers with the `zeros` function. Only one is shown below because they both use the same format.

```
    group = zeros(1,52);
```

Next I created a `for` loop to iterate through all 10,000 trials and defined a boolean variable to check that the while loop conditions are still true. I also defined the variable that the loop will check for birthdays near. Next I created a `while` loop to loop until a match is found. I then increased the *index* value and created a random number for every index value. Next I created a `for` loop to loop thru all the people in the group less than the index value to check for matches. In the for loop I used an `if` statement to check if any birthdays are within a week of each other. If so, the match variable was changed, and the while loop was broken. This is all shown below.

```
for m = 1:top
    index = 1;
    match = 0;
    group(index) = randi([1,365]);
        while match == 0
           index = index + 1;
           group(index) = randi([1,365]);
           for j = 1:index-1
            if abs(group(index) - group(j)) < 7 ||  abs(group(index) -
            group(j)) > 358
           N_vec(m) = index;
           match = 1;
```

Finally, I calculated the median and printed it out using the `median` and `fprintf` functions. I also created a histogram to model the distribution with the `histogram` function. I added labels to the histogram with the functions `xlabel` and `ylabel`. This is shown below.

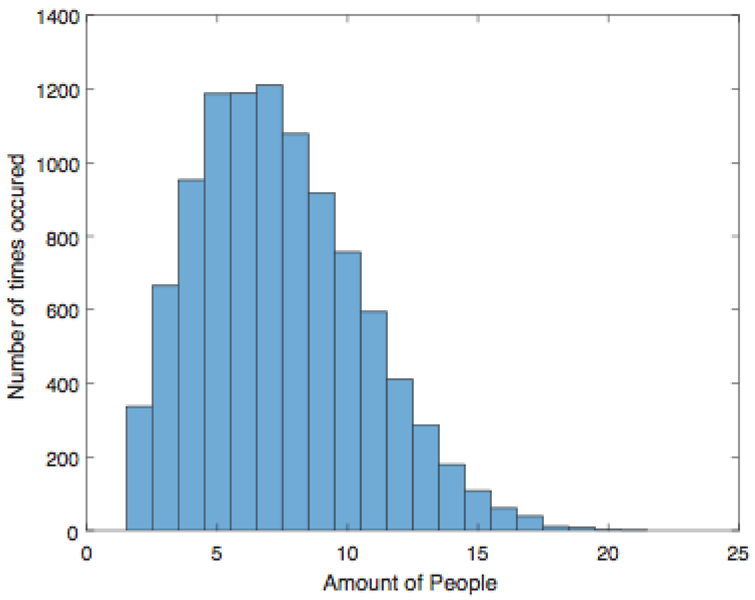
```
     x = median(N_vec);
     fprintf('Median Number of People = %1.0f',x);
     histogram(N_vec);
     xlabel('Amount of People');
     ylabel('Number of times occurred');
```

## 1.3 Calculations and Results

After running the script the following is printed.

```
     Median Number of People = 7
```

A histogram of the data is also created, shown below.



1.4 Discussion

As shown above, the median minimum number of people in a group to have 2 people with birthdays within a week of each other is 7 people. This means that there is greater than a 50% chance of having a birthday within a week of someone in a group of larger than 7 people. This value makes sense because in my experience it is very likely to know several people in a group of friends whose birthdays lie within a week of each other. I expect this value to go down in real life because it is a lot more common for some days of the year than others. For example, a good percentage of people have birthdays approximately 9 months after Valentine's Day for obvious reasons. It is the same for 9 months after New Years Eve.

## 2 Simulated Annealing

### 2.1 Introduction

The purpose of this script is to find the shortest, fully closed path between a set of N random points on a 2-D grid. I can do this by using a simulated annealing algorithm. This algorithm uses a function to find the path distance and iterates until it finds the smallest distance. After finding this distance, I plotted a graph of the random points with the final path and a plot of the distance vs the iteration number.

### 2.2 Models and methods

I first created a function to calculate the path distance for each specific iteration. This was done in a separate script. In this script I first used a function call to create the function `newpathdistance`. Next I defined the initial conditions and then created a `for` loop to iterate through the entire length of the input vectors. In the loop, I updated the distance calculation based on the given formulas. Most of this code is shown below. There is an else statement for the case when k = N but this is almost exactly the same code as the if statement so I left it out.

```
      function dist = getpathdistance(x, y, path)
      N = length(path);
      dist = 0;
      for k = 1:N
      if k ~= N
         distance = sqrt((x(path(k))-x(path(k+1)))^2 +
(y(path(k))-y(path(k+1)))^2);
        dist = dist + distance;
```

This finished the function, so I created a new script for the problem. In this script I set the random number generator to default before starting the code. I then required the user to input the number of points with the `input` function. Next I defined the x, y, and path vectors with the `zeros` function, `rand` function, `randperm` function, and a `for` loop. This is shown below.

```
      rng('default');
      N = input('Please input value for N: ');
      x = zeros(1,N);
      y = zeros(1,N);
      path = randperm(N);
      for k = 1:N
            x(k) = rand;
            y(k) = rand;
      end
```

I then called the function `getpathdistance` to find the initial path, and defined some constants and a vector to put all the distances in.

```
      dist = getpathdistance(x, y, path);
      T = 1000;
      index = 0;
      D_vec = zeros(1,1379);
```

Finally I created a `while` loop to iterate until the temperature dropped below 1. In the loop, I randomly swapped two values in the path and created a new vector `pathNew` for this swapped path. I then calculated the new distance from this path and the function `getpathdistance`. Then I calculated the difference between the old path and the new path and calculated the probability of the exploration metric *p*. I then used and `if elseif` statement to update the distance and path based on the exploration metric. Finally I updated the temp *T*, the index, and put the distance value in a vector. This is all shown below.

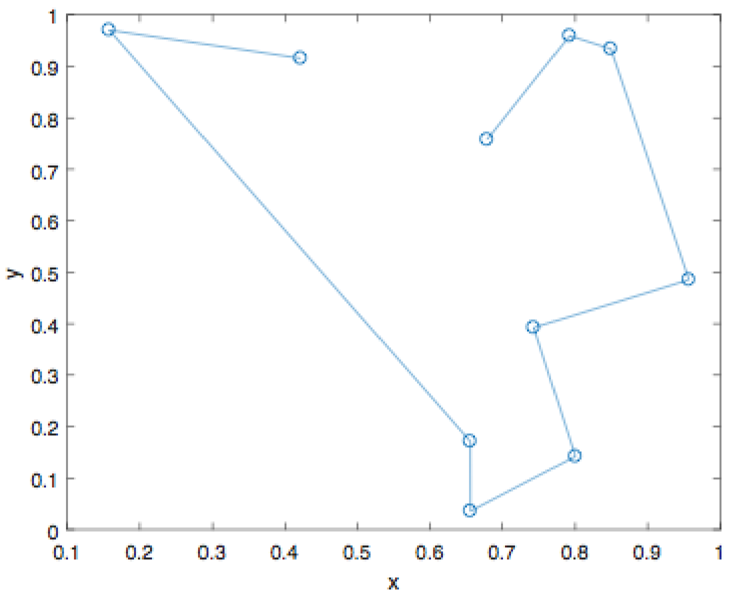
```
while T > 1
    swap = randperm(N,2);
    pathNew = path;
    pathNew(swap(1)) = path(swap(2));
    pathNew(swap(2)) = path(swap(1));
    distNew = getpathdistance(x,y,pathNew);
    diff = distNew - dist;
    p = exp(-1000*diff/T);
    z = rand;
    if diff < 0
        path = pathNew;
        dist = distNew;
    elseif p > z
        path = pathNew;
        dist = distNew;
    end
    T = T*(1 - 0.005);
    index = index + 1;
    D_vec(index) = dist;
end
```

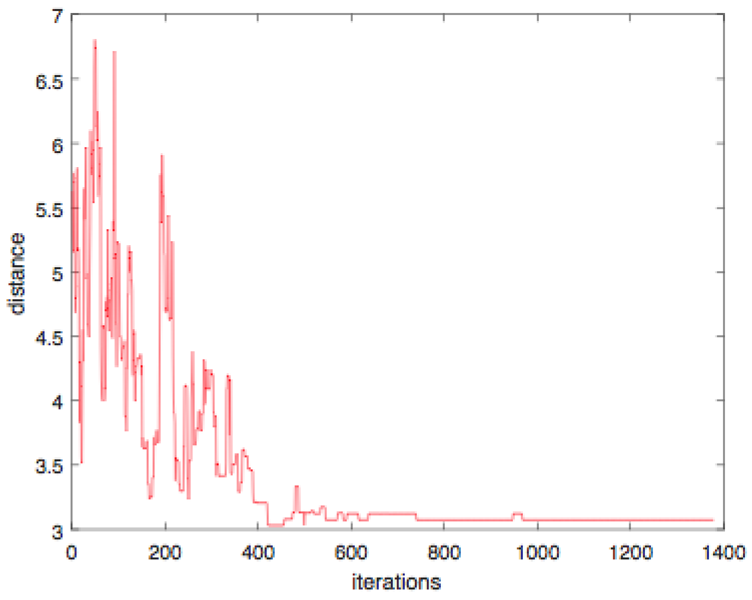
Finally I created vectors to graph this data with the `linspace` and `zeros` functions and a `for` loop. Then I graphed the two plots with the `plot`, `xlabel`, and `ylabel` functions. This is shown below.

```
iterations = linspace(1,index,index);
x2 = zeros(1,N);
y2 = zeros(1,N);
for k = 1:N
    x2(k) = x(path(k));
    y2(k) = y(path(k));
end
plot(x2,y2,'o-');
xlabel('x');
ylabel('y');
figure;
plot(iterations, D_vec,'r');
xlabel('iterations');
ylabel('distance');
```

## 2.3 Calculations and Results

After running the script and inputting 10 for the number of points, the following graphs are printed.





## 2.4 Discussion

As shown above, the code does accurately find the shortest path between all the points. Just looking by eye, I cannot think of another combination of path choices that would result in a shorter path. As N gets larger, the final path choices becomes more questionable whether it is actually the shortest path. If $\lambda$ was smaller and c was larger, these results would be improved because it would take more iterations to get the answer.