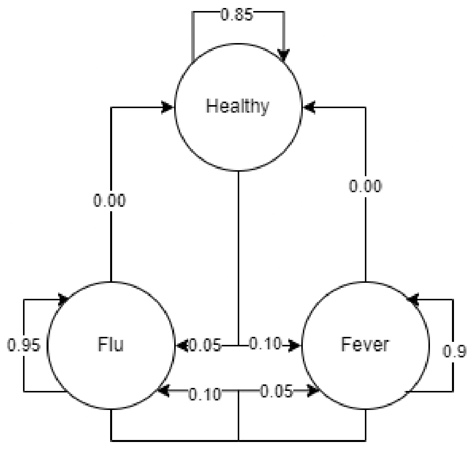
# Reliability Assignment 2

Simon dos Reis Spedsbjerg

April 30, 2024

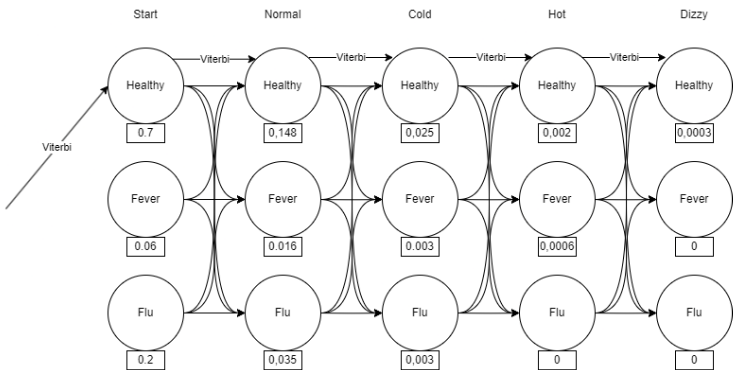
## 1 Individual

### 1.1 1 a



### 1.2 1 b

There is some rounding errors in the calculations and numbers, but this will not change the path in this case.

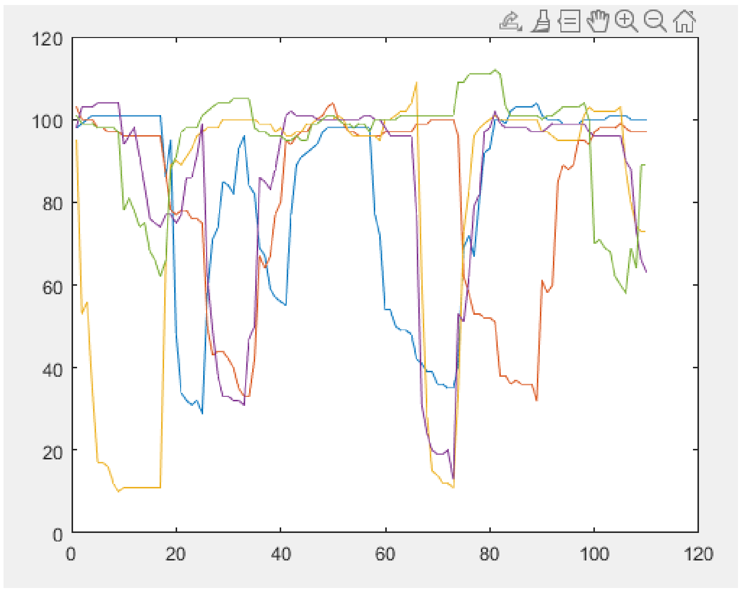


## 2 Generate a set of continuous data given the following table of information

Generating a set of random given some arguments can be done several ways, after developing three different ways we choose the one which gives the most realistic set.

- Random numbers, which can easily comply with the requirements, but is not realistic in any sort of way.
- A normal distribution, which do fulfill the requirements and most things, a normal distribution maps to real world, but it doesn't here.
- The last one which we used would add points with a certain distance between each other to ensure that we keep most data within the required 75% data range. The algorithm would then work its way towards the next point with a certain improvement, so it would get closer for each datapoint to the targeted point. There would be a chance of going away from the target just to add some noise to the data. [Listing 1]

Final data:



A data visualizer was also made to run in C# but was not used due to looking terrible.
All requirements are inherently meet except the mean, the mean can be implemented through post-processing by moving some of the datapoints which is outside the 75% data range either up of down until mean is meet.

## 3 Develop a machine learning model (HMM) applying the generated dataset

3. a. The data is converted into 4 discrete symbols like this:
Given value < 36 = 1
Given value >= 36 and < 65 = 2
Given value >= 65 and < 94 = 3
Given value > 94

It is done like this because it is somewhat close to a 4 way split.
b.

| | | | |
|---|---|---|---|
| HMM: | | | |
| pinit_lrn = | | | |
| 1.0000 | | | |
| 0.0000 | | | |
| 0.0000 | | | |
| A_lrn = | | | |
| 0.9573 | 0.0276 | 0.0151 | |
| 0.1176 | 0.7954 | 0.0870 | |
| 0.0000 | 0.0977 | 0.9023 | |
| B_lrn1 = | | | |
| 0.0000 | 0.0000 | 0.0000 | 1.0000 |
| 0.0000 | 0.0027 | 0.9672 | 0.0301 |
| 0.4399 | 0.5577 | 0.0024 | 0.0000 |

c. A bit challeging with the given code, but otherwise not many difficulties.
The more training data and the more discrete symbols used the more accurate the model might become. However, more training data means more time, and more discrete symbols might make it less general and therefore harder to use.
the calculation for the third part of the report are done in the assignment2_3.m file.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Viterbi: | | | | | | | | | | | | |
| current_distributions = | | | | | | | | | | | | |
| 0.0000 | | | | | | | | | | | | |
| 1.0000 | | | | | | | | | | | | |
| 0.0000 | | | | | | | | | | | | |
| all_distributions = | | | | | | | | | | | | |
| Columns 1 through 13 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9787 | 0.5353 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0213 | 0.4647 | 0.9997 | 0.9997 | 0.9997 | 0.9997 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0003 | 0.0003 | 0.0003 | 0.0003 |
| Columns 14 through 26 | | | | | | | | | | | | |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.9997 | 0.9997 | 0.9753 | 0.0378 | 0.9997 | 0.9997 | 1.0000 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0003 | 0.0003 | 0.0247 | 0.9622 | 0.0003 | 0.0003 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 27 through 39 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 40 through 52 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 53 through 65 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 66 through 78 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 79 through 91 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| Columns 92 through 104 | | | | | | | | | | | | |
| 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9963 | 0.9787 | 0.5353 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |
| 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0037 | 0.0213 | 0.4647 | 0.9997 | 0.9997 | 0.9997 | 0.9753 | 0.0005 |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0003 | 0.0003 | 0.0003 | 0.0247 | 0.9995 |
| Columns 105 through 110 | | | | | | | | | | | | |
| 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | | | | | | | |
| 0.0005 | 0.0378 | 0.9753 | 0.0378 | 0.9997 | 1.0000 | | | | | | | |
| 0.9995 | 0.9622 | 0.0247 | 0.9622 | 0.0003 | 0.0000 | | | | | | | |

## 4 Appendix

Full code: https://github.com/SSpedsbjerg/Reliability

Algorithm used to generate the data sets:

```csharp
class ContinousData : Distrubtion {

  double? range0 = null;
  double? range1 = null;
  double requiredDataPercentage = 0;

  public void SetTargetRange(double value1, double value2) {
    range0 = value1; range1 = value2;
  }

  public void voidTargetRange() {
    range0 = null; range1 = null;
  }

  public void setRequiredDataPercentage(double value) {
    requiredDataPercentage = value;
  }

  private List<double> GenerateTargetPoints() {
    if(range0 == null || range1 == null) { return null; }
    List<double> points = new List<double>();
    Random random = new Random();
    int totalPoints = (int)((this.GetCount() / 10) + 1);
    for (int i = 0; i < ((requiredDataPercentage / 100) * totalPoints) + 1; i++) {
      points.Add((double)(range0 + (random.NextDouble() * (range1 - range0))));
    }
    for (int i = 0; i <= totalPoints - ((requiredDataPercentage / 100) * totalPoints) + 1; i++) {
      points.Add((double)(this.GetMin() + (random.NextDouble() * (this.GetMax() - this.GetMin()))));
    }
    Shuffel(points);
    return points;
  }

  private void Shuffel(List<double> values) {
    Random random = new Random();
    int n = values.Count;
    while(n > 1) {
      n--;
      int k = random.Next(n + 1);
      double value = values[k];
      values[k] = values[n];
      values[n] = value;
    }
  }

  public List<double> GenerateData() {
    List<double> points = GenerateTargetPoints();
    List<double> values = new List<double>();
    int i = 0;
    Random random = new Random();
    values.Add(points[i]);
    int j = 0;
    while (this.GetCount() > values.Count()) {
      double maxAddition = ((double)range1 / ((double)range0 + (double)range1));
      try {
        values.Add(values.Last() + ((random.NextDouble() -0.2) * ((points[i + 1] - values.Last()))));
        j++;
        if (j >= (int)((this.GetCount() / points.Count()))) {
          i++;
          Console.WriteLine($"Total Points: {points.Count()}, Points used: {i}");
          j = 0;
          values.Add(points[i]);
        }
      }
      catch(ArgumentOutOfRangeException) {
        Console.WriteLine("Index");
        return values;
      }
    }
    return values;
  }
}
```

Listing 1: Datageneration