Computational Modelling: Week 3

Task 1: Exponentially distributed random numbers

In this task, an analytical Monte Carlo method was used to model the decay times of 1,000 particles in two cases where the mean life times, τ, were τ = 5 s and τ = 20 s.

The distribution of decay times is exponential,

$$p(t) = C \cdot \exp\left(-\frac{t}{\tau}\right), \qquad \text{Equation 1}$$

where t = 0…∞. The constant C was derived using the normalisation condition $\int_0^\infty p(t)dt = 1$,

$$C = \frac{1}{\tau}, \qquad \text{Equation 2}$$

which made the probability distribution

$$p(t) = \frac{1}{\tau} \cdot \exp\left(-\frac{t}{\tau}\right). \qquad \text{Equation 3}$$

Equation 3 was analytically inverted to find r, the probability of the particle decaying within a particular time interval, and rearranged for D, the distribution which describes the decay of particles with time,

$$D = -\tau \cdot \ln(1 - r). \qquad \text{Equation 4}$$

Using Equation 4, histograms were generated for the two cases which are shown in Figures 1 (τ = 5 s) and 2 (τ = 20 s).

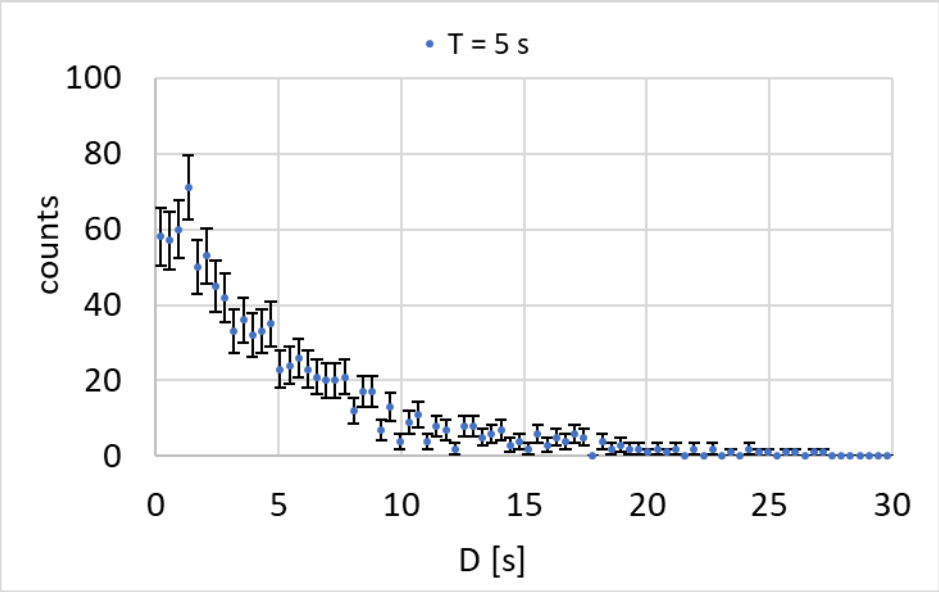

Figure 1: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 5 s, generated using an analytical method.

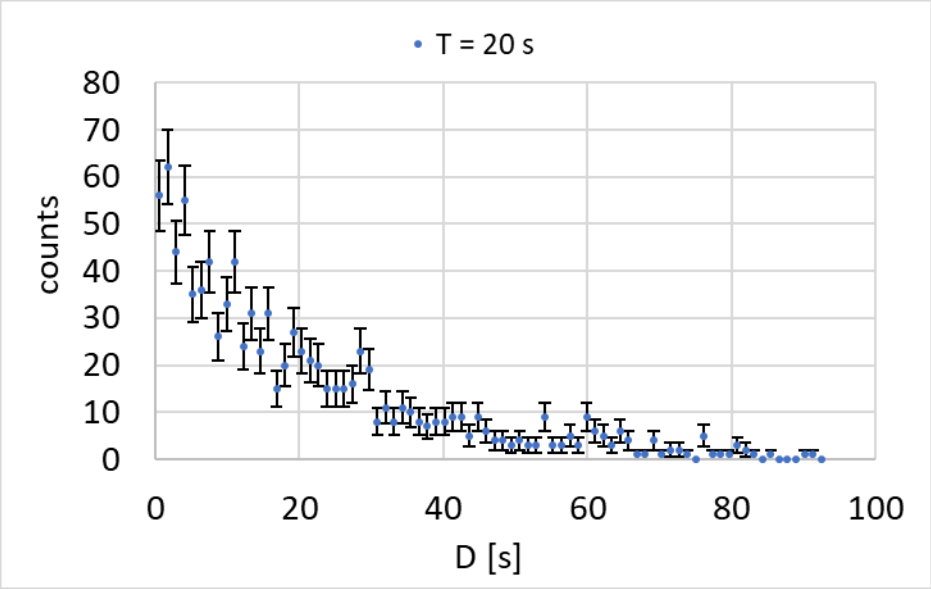

Figure 2: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 20 s, generated using an analytical method.

The histograms were generated using the `Histogram.java` class from Week 2. The number of trails was set to 1,000 in both instances as this was sufficiently high to give good resolution. The number of trials were looped over and the data were simulated by generating uniformly distributed random numbers between 0 and 1 for r. For each r, D was calculated (Line 19) using Equation 4 and added to the histogram. The random data were then printed and saved to a csv file. Finally, the results were plotted in Excel.

Both plots show an exponential decay in the counts over time. For τ = 5 s (Figure 1), the rate of decay is much faster than for τ = 20 s (Figure 2). The code used to generate the data is shown below:

```
1.  package task1;
2.  import java.util.Scanner;
3.  import java.util.Random;
4.  import static java.lang.Math.log;
5.
6.  public class Task1 {
7.
8.      static Random randGen = new Random();
9.      static Scanner keyboard = new Scanner(System.in);
10.
11.     public static void main(String[] args) {
12.         // create an instance of the Class Histogram: 80 bins from 0.0 to 30.0
13.         Histogram hist = new Histogram(80, 0, 30, "Uniform");
14.
15.         int trials = 1000; //1000 trials to get a smooth distribution and good appr
    oximation
16.         for (int i = 0; i < trials; i++) {
17.             double r = randGen.nextDouble(); //generate a random number between 0 a
    nd 1 for r
18.             double T = 5; //in this case, tau = 5 s
19.             double D = -T * Math.log(1 - r); //calculate D
20.             hist.fill(D);
21.         }
22.
23.         hist.print();
24.         hist.writeToDisk("T5Histogram.csv");
25.
```

```
26.         // creating an second instance of the Class Histogram: 80 bins from 0.0 to
    93.0
27.         Histogram hist2 = new Histogram(80, 0, 93, "Uniform");
28.         for (int i = 0; i < trials; i++) {
29.             //r and hence D generated using the same method as before:
30.             double r = randGen.nextDouble();
31.             double T = 20; //in this case, tau = 20 s
32.             double D = -T * Math.log(1 - r);
33.             hist2.fill(D);
34.         }
35.
36.         hist2.print();
37.         hist2.writeToDisk("T20Histogram.csv");
38.     }
39.
40. }
```

Task 2: Exponentially distributed random numbers (again)

In this task, random numbers were generated using a Hit-And-Miss method with the same exponential distribution as in Task 1, $p(t) = \exp\left(\frac{\tau}{t}\right)$.

In this task, time was restricted to t = 0…100s by generating random numbers between 1 and 0 using `randGen.nextDouble()` and multiplying by 100. A probability, `y`, was randomly generated and compared to the expected probability, p, calculated using t. If the random probability was smaller than the calculated probability, a 'hit' was recorded and t was stored in the data set. If y was larger than p, the data was not stored and the loop was repeated for that trial until a hit was recorded (`else { i-- },` Line 21).

Histograms were generated for τ = 5 s and τ = 20 s, the data saved to a csv and plotted in Excel (Figures 3 and 4 respectively).

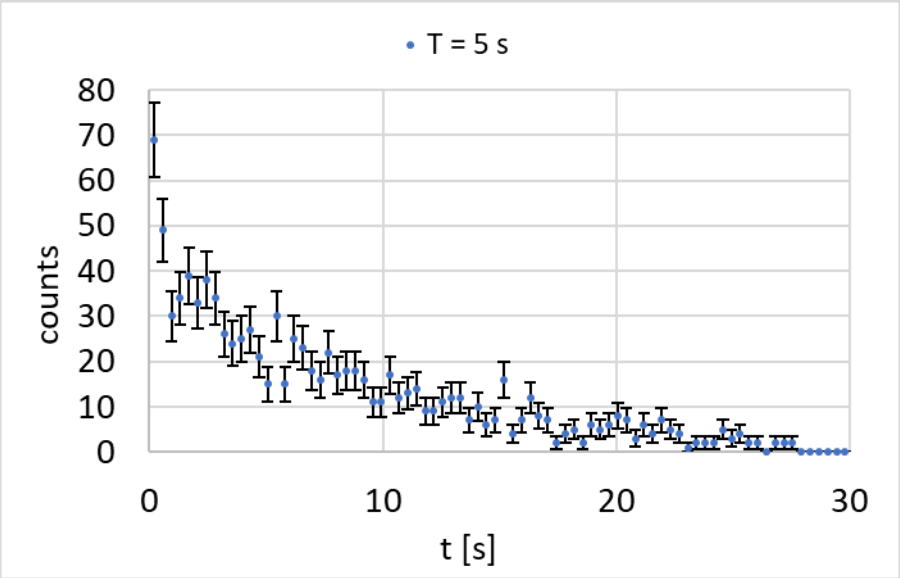

Figure 3: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 5 s. Random data were generated using the Hit-And-Miss method.

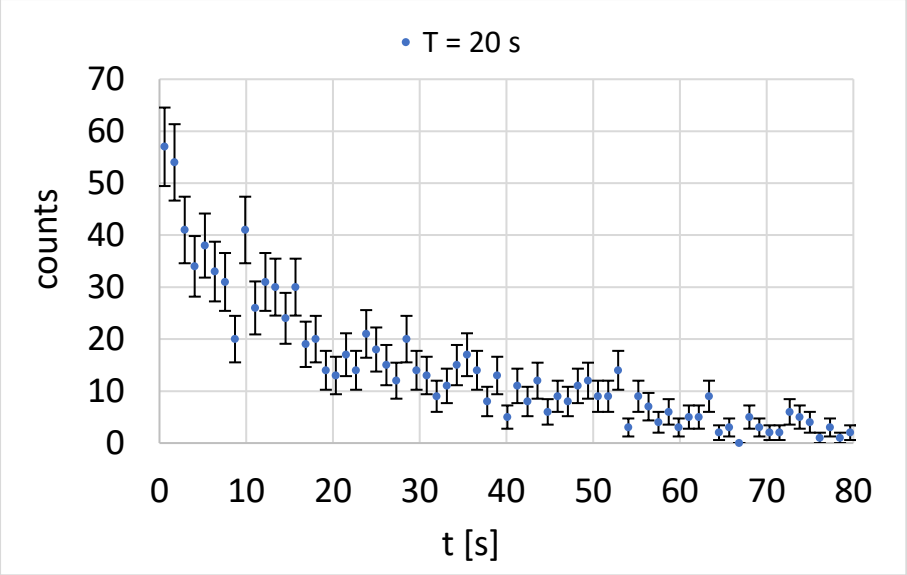

Figure 4: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 20 s. Random data were generated using the Hit-And-Miss method.

This method was less efficient that the first as it meant that the process had to be repeated for trials which did not record a 'hit' in order to obtain 1,000 counts. In the first method, a 'hit' was generated for each trial so less computation time was required. The Hit-And-Miss method was easier to implement in that no 'pen and paper' calculations were required. Hence, this could be a better method to use for complicated integrals when the number of events isn't too large. The code is shown below:

```
1.  package task2;
2.  import java.util.Random;
3.
4.  public class Task2 {
5.
6.      static Random randGen = new Random();
7.
8.      public static void main(String[] args) {
9.          //create an instance of the Class Histogram: 80 bins from 0.0 to 1.0
10.         Histogram hist = new Histogram(80, 0, 30, "Uniform");
11.
12.         int trials = 1000; //1000 trials to get a smooth distribution and good appr
    oximation
13.         for (int i = 0; i < trials; i++) {
14.             double t = randGen.nextDouble()*100; //generate a randon number between
     0 and 100 for t
15.             double T = 5; //in this case, tau = 5 s
16.             double y = randGen.nextDouble(); //randomly generated probability
17.             double p = Math.exp(-t/T); //calculated probability
18.
19.             if (y < p) { //if y is smaller than p, y is inside the distribution and
     can be included in the plot!
20.                 hist.fill(t);
21.             } else {
22.                 i--; //subtract 1 from i so that 1000 actual hits are recorded -
    > smooth distribution
23.             }
24.         }
25.
26.         hist.print();
27.         hist.writeToDisk("T5pVst2.csv");
```

```
28.
29.         //creating a second instance of the Class Histogram: 80 bins from 0.0 to 93
    .0
30.         Histogram hist2 = new Histogram(80, 0, 80, "Uniform");
31.         for (int i = 0; i < trials; i++) {
32.             //r and hence D generated using the same method as before:
33.             double t = randGen.nextDouble()*100;
34.             double y = randGen.nextDouble(); //to compare
35.             double T = 20; //in this case, tau = 20 s
36.             double p = Math.exp(-t/T);
37.
38.             if (y < p) {
39.                 hist2.fill(t);
40.             } else {
41.                 i--;
42.             }
43.         }
44.
45.         hist2.print();
46.         hist2.writeToDisk("T20pVst2.csv");
47.     }
48. }
```

Task 3: Exponential decay and detection

Finally, a program was created to simulate the distribution of the decay positions of particles as measured by a detector with limited resolution, σ. This was done in three cases; for a detector with perfect resolution, for σ = 0.02 m, and for σ = 0.10 m. In all cases, 1,000 particles were produced at d = 0 with a mean lifetime of τ = 10 ps travelling a speed of approximately c = 3 x 10^8 m/s and with relativistic time-dilation factor γ = 50. The code is shown at the end of this section.

Similarly to Task 1, an analytical method was employed. For each trial, a time, t, was randomly generated (Line 21). The lifetime of the particle was calculated using t and Equation 4 (Line 24). The proper distance travelled by the particle, D, was calculated in Line 26 via,

$$D = \gamma ct. \qquad \text{Equation 5}$$

D was entered into the data for the "perfect detector" plotted in Figure 5. To account for the limited detector resolution in the last two cases, a Gaussian smearing in the position of the particle, `d_detector`, was added to D:

```
25.             double d_detector = randGen.nextGaussian()*sigma; //randomly generating
a detector smudge
28.             hist2.fill(D + d_detector);   //d_measured = D + D_detector
```

The cases where σ = 0.02 m and σ = 0.10 m are plotted in Figures 6 and 7 respectively.

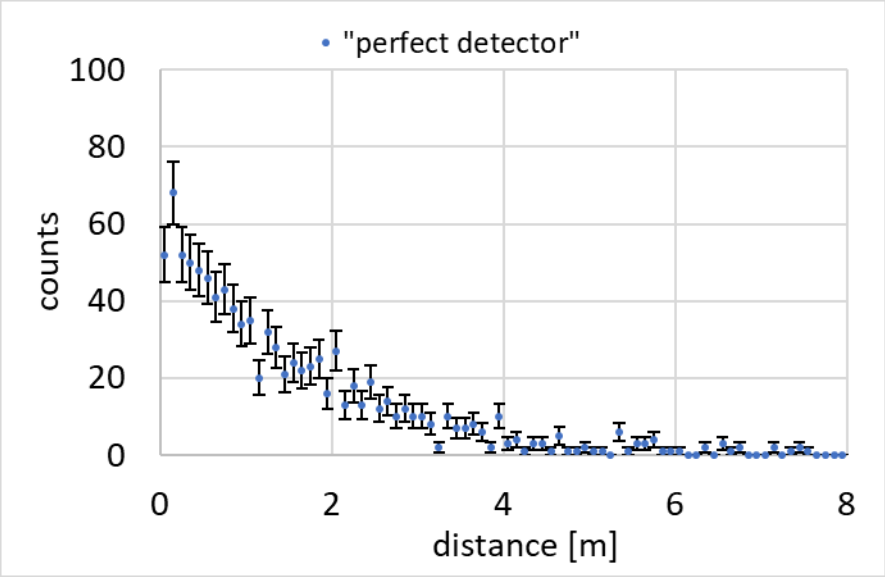

Figure 5: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 10 ps for a detector with perfect resolution.

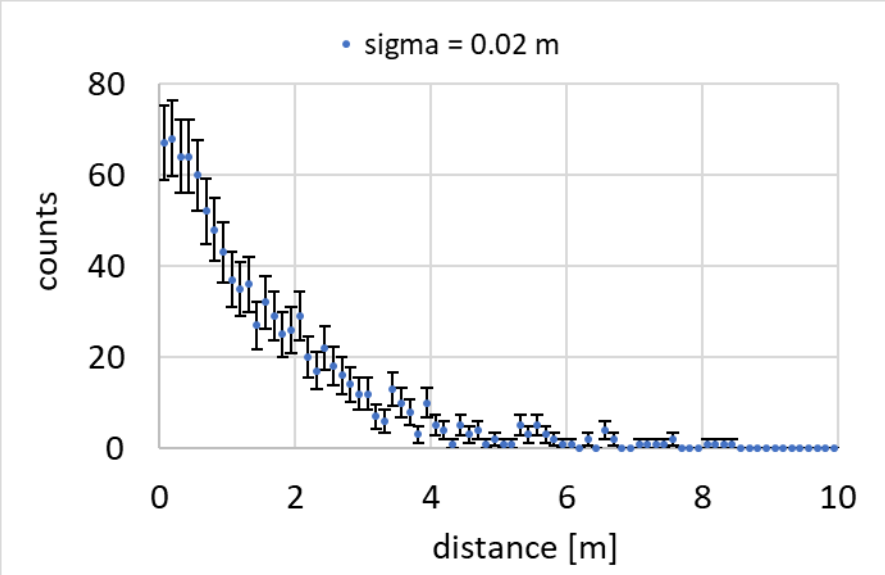

Figure 6: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 10 ps for a detector with resolution σ = 0.02 m.

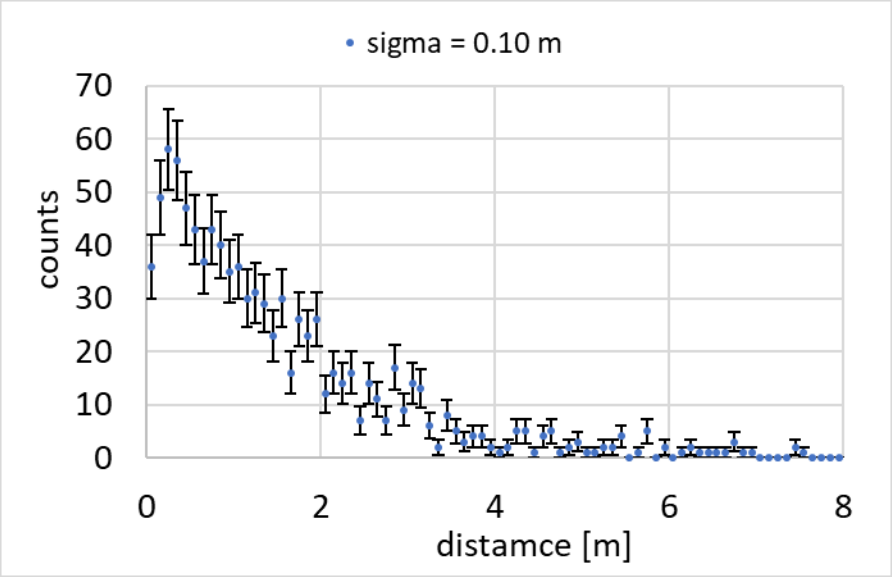

Figure 7: The distribution of particle decay times for 1,000 events with a mean lifetime of τ = 10 ps for a detector with resolution σ = 0.10 m.

By comparing Figures 5 and 7, it can be noted that as σ increased, the random scatter in the decay distance increased. The distribution also became more spread out and the maxima moved from 0 to σ; the distribution became more Gaussian. This flattened the distribution and the number of counts per bin decreased. In Figures 5 and 6 the maximum counts per bin was approximately 70 and in Figure 7 it was less than 60.

Code:

```
1.  package task3;
2.
3.  import java.util.Random;
4.
5.  public class Task3 {
6.
7.      public static void main(String[] args) {
8.          double T = 10E-11; //mean life time = 10 ps
9.          double c = 3E8; //approximate speed = 3*10^8 m/s
10.         double gamma = 50; //relativistic time dilation factor
11.         double d_0 = 0.; //particle is produced at d = 0
12.
13.
14.         Histogram hist = new Histogram(80,0,8, "Uniform");
15.         Histogram hist2 = new Histogram(80,0,10, "Uniform");
16.         Histogram hist3 = new Histogram(80,0,8, "Uniform");
17.
18.         Random randGen = new Random();
19.         int trials = 1000;
20.         for (int i = 0; i < trials; i++) {
21.             double t = randGen.nextDouble(); //randomly generate a time
22.             double sigma = 0.02; //in Case 2, detector resolution is 0.02m

23.             double d_detector = randGen.nextGaussian()*sigma; //randomly generating
    a detector smudge
24.             double lifetime = -T*Math.log(1-
    t); //the lifetime of the particle [s]
25.             double D = lifetime*gamma*c; //distance travelled by particle according
    to observer [m]
26.             hist.fill(D); //d_decay
```

```
27.             hist2.fill(D + d_detector);
28.         }
29.
30.         for (int i = 0; i < trials; i++) {
31.             double t = randGen.nextDouble(); //randomly generate a time
32.             double sigma = 0.10; //in Case 3, detector resolution is 0.10m

33.             double d_detector = randGen.nextGaussian()*sigma;
34.             double lifetime = -T*Math.log(1-
    t); //the lifetime of the particle [s]
35.             double D = lifetime*gamma*c; //distance travelled by particle according
     to observer [m]
36.             hist3.fill(D + d_detector);
37.         }
38.
39.
40.         hist.print();
41.         hist.writeToDisk("pVsd.csv"); //"perfect detector"
42.         hist2.print();
43.         hist2.writeToDisk("pVsdSigma1.csv"); //sigma = 0.02
44.         hist3.print();
45.         hist3.writeToDisk("pVsdSigma2.csv"); //sigma = 0.10
46.     }
47. }
```