Turn recording on!!!

Simulation Modeling

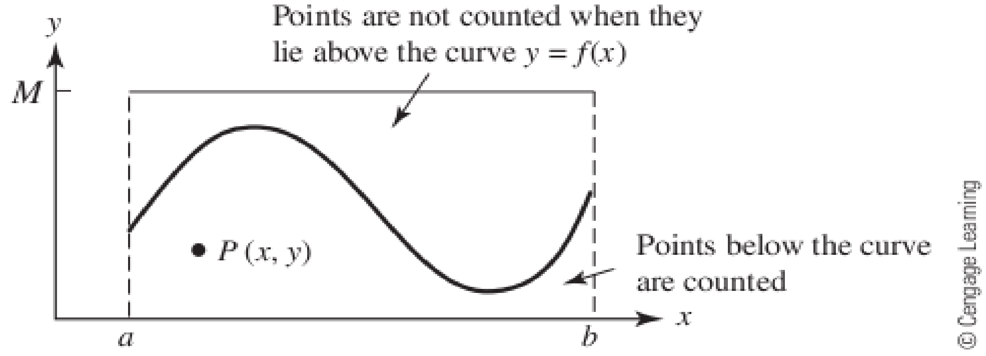
- Closely follows Chapter 05 of *A First Course in Mathematical Modeling*, Fifth Edition, by Giordano, Fox and Horton
- So far, we've tried to build models based on actual data, whether we understood the underlying physical processes or not
- Sometimes we're unable to collect the data we need to build a model, so we have to generate “fake data”
  - Consider the need to model the flow of elevators or traffic signals in order to seek the most efficient patterns
  - We would need a huge number of trials for statistically meaningful results, and we don't want to inconvenience people with lots of different experiments
  - Sometimes the system we want to model doesn't even exist yet, so we need to generate fake data
- Monte Carlo simulation – typically generating random numbers in meaningful proportions to simulate real-world data

Simulating deterministic behaviour with probabilistic processes

- Calculating the area under a curve
  - The actual area is deterministic, and typically available via analytical or numerical solutions
  - A probabilistic scheme uses Monte Carlo methods to generate random points that are either in the area or out
  - I don't really like the book's approach, but I'll show it, then show more traditional Monte Carlo integration methods

Monte Carlo – area under a curve

- Put the curve of interest in a rectangle, of which we can easily calculate the area

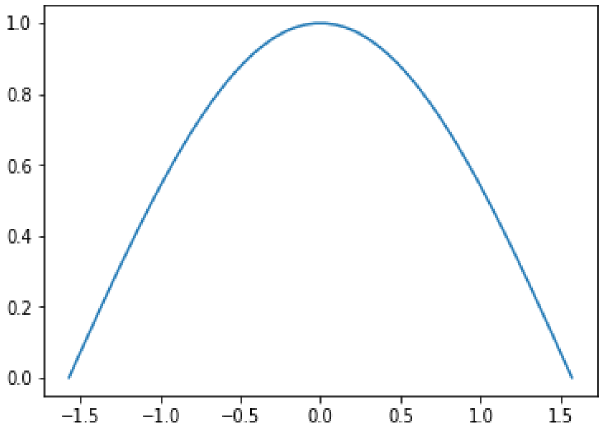


- Generate uniformly distributed random points within the rectangle. Count each point that lies below the curve
- The ratio of counted points to total points will be the proportion of the rectangular area that lies under the curve

$$\frac{\text{area under curve}}{\text{area of rectangle}} = \frac{\text{number of points counted below curve}}{\text{total number of random points}}$$

Monte Carlo example

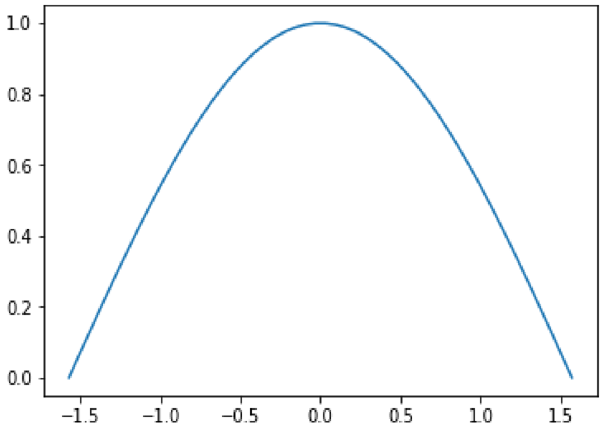
$$\int_{-\pi/2}^{\pi/2} \cos x dx$$




```
height = 1.0
a = -np.pi / 2.0
b = np.pi / 2.0

area_rect = height*(b-a)
print('area_rect %f' % area_rect)

N = 50
# Generate random coordinates
xcoords = np.random.uniform(low=a, high=b, size=N)
ycoords = np.random.uniform(low=0, high=height, size=N)

# Count number of points below the curve
counter = 0
for i in np.arange(N):
    if ycoords[i] <= np.cos(xcoords[i]):
        counter += 1

estimated_area = area_rect*counter/N

print('counter: %d, total: %d' % (counter, N))
print('estimated_area: %6.4f' % estimated_area)
```

Monte Carlo example

$$\int_{-\pi/2}^{\pi/2} \cos x dx$$

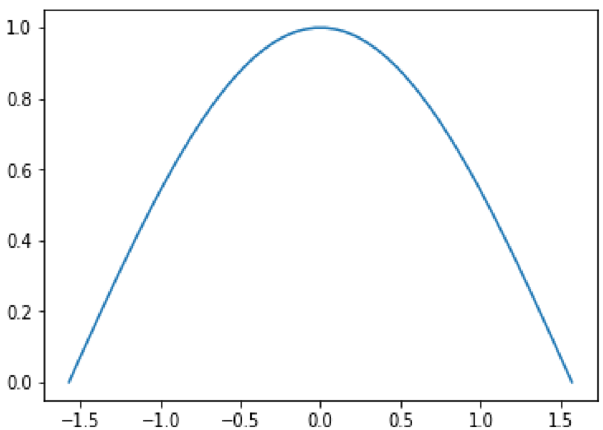



```
area_rect 3.141593
counter: 34, total: 50
estimated_area: 2.1363
```

```
height = 1.0
a = -np.pi / 2.0
b = np.pi / 2.0

area_rect = height*(b-a)
print('area_rect %f' % area_rect)

N = 50
# Generate random coordinates
xcoords = np.random.uniform(low=a, high=b, size=N)
ycoords = np.random.uniform(low=0, high=height, size=N)

# Count number of points below the curve
counter = 0
for i in np.arange(N):
    if ycoords[i] <= np.cos(xcoords[i]):
        counter += 1

estimated_area = area_rect*counter/N

print('counter: %d, total: %d' % (counter, N))
print('estimated_area: %6.4f' % estimated_area)
```

Monte Carlo example

$$\int_{-\pi/2}^{\pi/2} \cos x dx$$



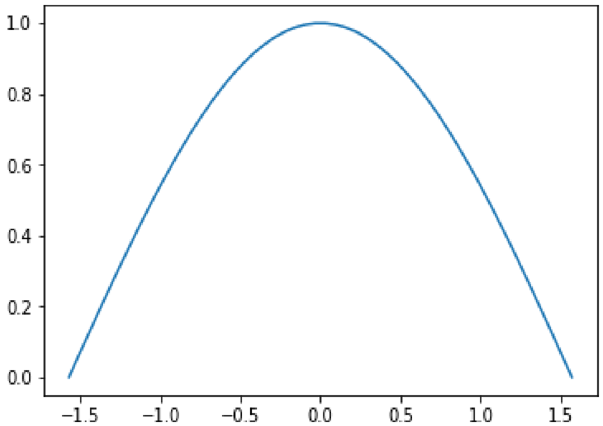

```
area_rect 3.141593
counter: 34, total: 50
estimated_area: 2.1363
```

```
height = 1.0
a = -np.pi / 2
b = np.pi / 2

area_rect = h
print('area_r

N = 50
# Generate random coordinates
xcoords = np.random.uniform(low=a, high=b, size=N)
ycoords = np.random.uniform(low=0, high=height, size=N)

# Count number of points below the curve
counter = 0
for i in np.arange(N):
    if ycoords[i] <= np.cos(xcoords[i]):
        counter += 1

estimated_area = area_rect*counter/N

print('counter: %d, total: %d' % (counter, N))
print('estimated_area: %6.4f' % estimated_area)
```

```
area_rect 3.141593
counter: 34, total: 50
estimated_area: 2.1363

counter: 27, total: 50
estimated_area: 1.6965

area_rect 3.141593
counter: 26, total: 50
estimated_area: 1.6336
```

Monte Carlo example

$$\int_{-\pi/2}^{\pi/2} \cos x dx$$




```
area_rect 3.141593
counter: 34, total: 50
estimated_area: 2.1363
```

```
height = 1.0
a = -np.pi / 2
b = np.pi / 2

area_rect = he
print('area_re

N = 50
# Generate random coordinates
xcoords = np.random.uniform(low=a, high=b, size=N)
ycoords = np.random.uniform(low=0, high=height, size=N)

# Count number
counter = 0
for i in np.ar
    if ycoords
        counte

estimated_area

print('counter                                     ))
print('estimat                                     ea)
```

```
area_rect 3.141593
counter: 34, total: 50
estimated_area: 2.1363

counter: 27, total: 50
estimated_area: 1.6965

area_rect 3.141593
counter: 26, total: 50
estimated_area: 1.6336
```

```
area_rect 3.141593
counter: 31805, total: 50000
estimated_area: 1.9984

area_rect 3.141593
counter: 31724, total: 50000
estimated_area: 1.9933

area_rect 3.141593
counter: 31970, total: 50000
estimated_area: 2.0087
```

More traditional Monte Carlo Algorithm

- Generate random points within interval, Ω, of integration
- Evaluate the integrand at each random point
- Sum all of the integrand evaluations and divide by the number of evaluations, to get the *mean* function value
- Multiply this mean value by “size” of interval

Monte Carlo Algorithm

$$\int_{\Omega} f(a,b,\ldots,z)\,d\Omega \approx \frac{\Omega}{n}\sum_{i=1}^{n} f(a_i,b_i,\ldots,z_i)$$

General *n*-dimensional form

- Error approximately $n^{-1/2}$
- Example - to gain extra decimal place of accuracy, *n* must be increased by factor of 100
- Not competitive for one or two dimensions
- Convergence rate independent of number of dimensions!

1D Monte Carlo Example

1D Monte Carlo Formula

$$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{i=1}^{n} f(x_i)$$

$$I(g) = \int_0^1 e^{-x^2}dx$$

$$\int_0^1 e^{-x^2}dx \approx \frac{1}{n}\sum_{i=1}^{n} e^{-x^2}$$

MATLAB script

```
# The function we're integrating
function val = f(x)
     val = exp(-x^2);
endfunction

MAXN = 100;

n = 0;
sum = 0;
while (n < MAXN)

   # Generates random number
   #  between 0 and 1
   x = rand;

   sum = sum + f(x);

   n++;
endwhile

result = sum/n
```

n	Sample Results			
10	0.68897	0.85972	0.72331	0.82431
100	0.75350	0.74103	0.74532	0.77929
1000	0.75231	0.74837	0.74338	0.74616
10000	0.74438	0.74454	0.74635	0.74840

2D Monte Carlo Example

2D Monte Carlo Formula

$$\int_a^b \int_c^d f(x,y)\,dxdy \approx \frac{(b-a)(c-d)}{n} \sum_{i=1}^{n} f((x,y)_i)$$

$$I(g) = \int_0^1 \int_0^1 e^{-x^2 y} dx$$

$$\int_0^1 \int_0^1 e^{-x^2 y} dxdy \approx \frac{1}{n} \sum_{i=1}^{n} e^{-x^2 y}$$

MATLAB script

```
# The function we're integrating
function val = f(x, y)
     val = exp(-y*x^2);
endfunction

MAXN = 10000;

n = 0;
sum = 0;
while (n < MAXN)

   # Generates random number
   #  between 0 and 1
   x = rand;
   y = rand;

   sum = sum + f(x,y);

   n++;
endwhile

result = sum/n
```

n	Sample Results			
10	0.88563	0.82103	0.88990	0.86539
100	0.85712	0.87623	0.88007	0.87531
1000	0.86726	0.85909	0.86455	0.86057
10000	0.86232	0.85982	0.86063	0.86130

4D Monte Carlo Example

4D Monte Carlo Formula

$$\int_a^b\int_c^d\int_e^f\int_g^h f(w,x,y,z)\,dwdxdydz \approx \frac{(b-a)(d-c)(f-e)(h-g)}{n}\sum_{i=1}^{n} f((w,x,y,z)_i)$$

$$I(g) = \int_0^1\int_0^1\int_0^1\int_0^1 e^{-wx^2yz}\,dwdxdydz$$

$$\int_0^1\int_0^1\int_0^1\int_0^1 e^{-wx^2yz}\,dwdxdydz \approx \frac{1}{n}\sum_{i=1}^{n} e^{-wx^2yz}$$

MATLAB script

```
# The function we're integrating
function val = f(w, x, y, z)
     val = exp(-y*w*z*x^2);
endfunction

MAXN = 100;

n = 0;
sum = 0;
while (n < MAXN)

   # Generates random number
   #  between 0 and 1
   w = rand;
   x = rand;
   y = rand;
   z = rand;

   sum = sum + f(w,x,y,z);

   n++;
endwhile

result = sum/n
```

n	Sample Results			
10	0.97300	0.99244	0.97744	0.98583
100	0.96397	0.95508	0.95703	0.95573
1000	0.96288	0.96175	0.96482	0.96202
10000	0.96113	0.96192	0.96253	0.96230

Generating Random Numbers

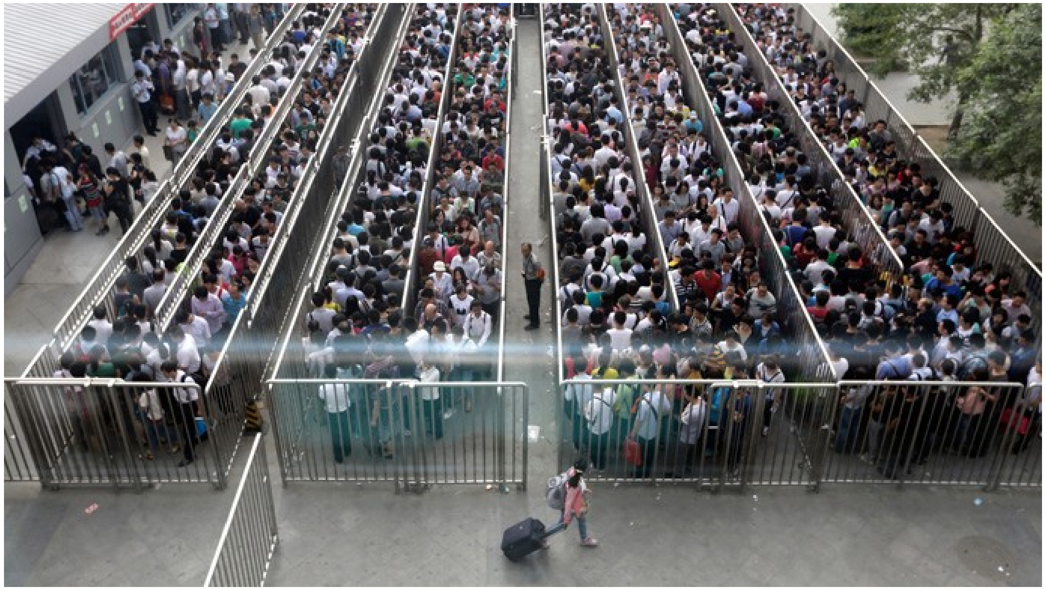
- Short section in text, I'm not going to cover
- In general, there are simple methods and complex methods
- There can be certain "gotchas" for some methods, such as repetition, degeneration to a certain value, etc.
- Be sure you know what you're getting when you use canned packages – Numpy has a lot!

Generating custom random distributions

- This is a central theme for the rest of this chapter – how to generate random numbers so that they simulate a specified distribution
- Generation of random uniform numbers – easy, we have functions for this
- Generation of random normal values – easy, we have functions for this
- Generation of custom distributions – requires approaches addressed in this chapter

Random uniform numbers

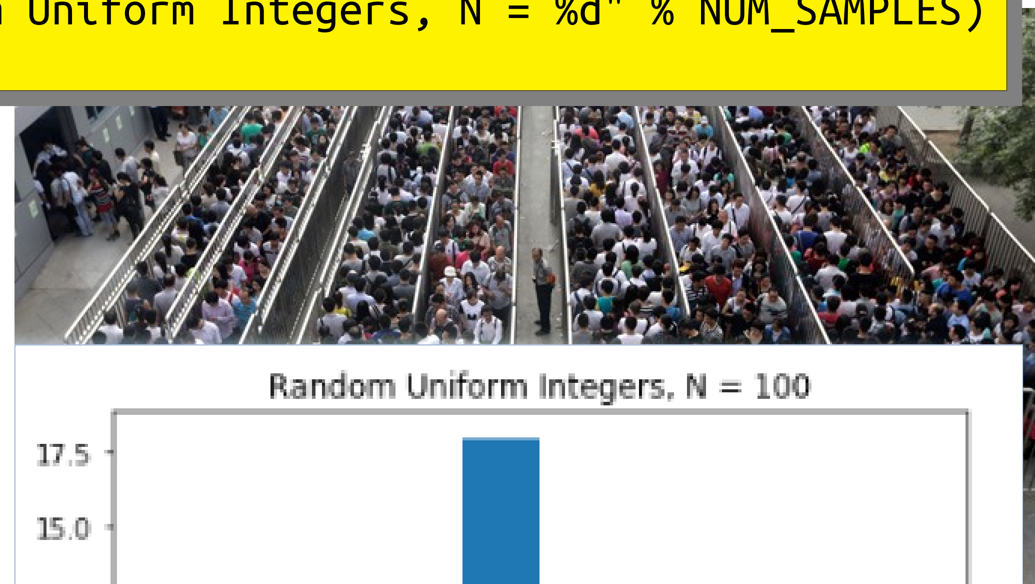
- Consider simulation of Beijing subway security procedures as a function of weight of individual passenger belongings
- We will assume that the weight of passenger belongings between 5 and 30kg is equally likely, or uniformly distributed.

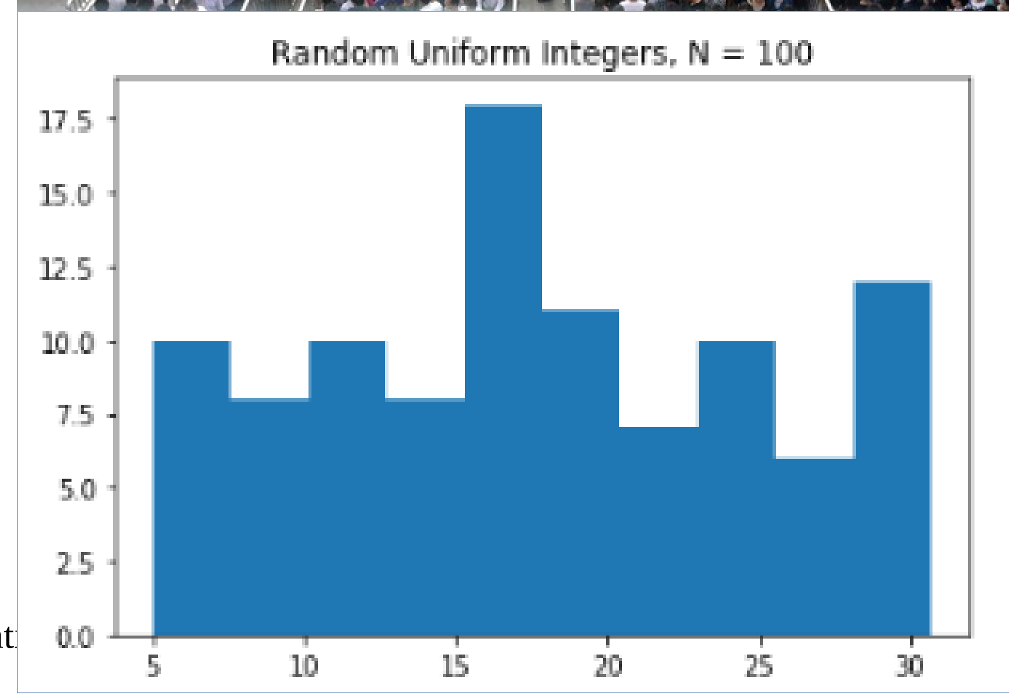


Random uniform numbers

- Consider sim
  Beijing subwa
  procedures as a function
  of weight of individual
  passenger belongings
- We will assume that the weight of passenger belongings between 5 and 30kg is equally likely, or uniformly distributed.

```
payload_kg = np.random.uniform(low=5, high=31,
                                size=NUM_SAMPLES)
plt.hist(payload_kg)
plt.title("Random Uniform Integers, N = %d" % NUM_SAMPLES)
plt.show()
```



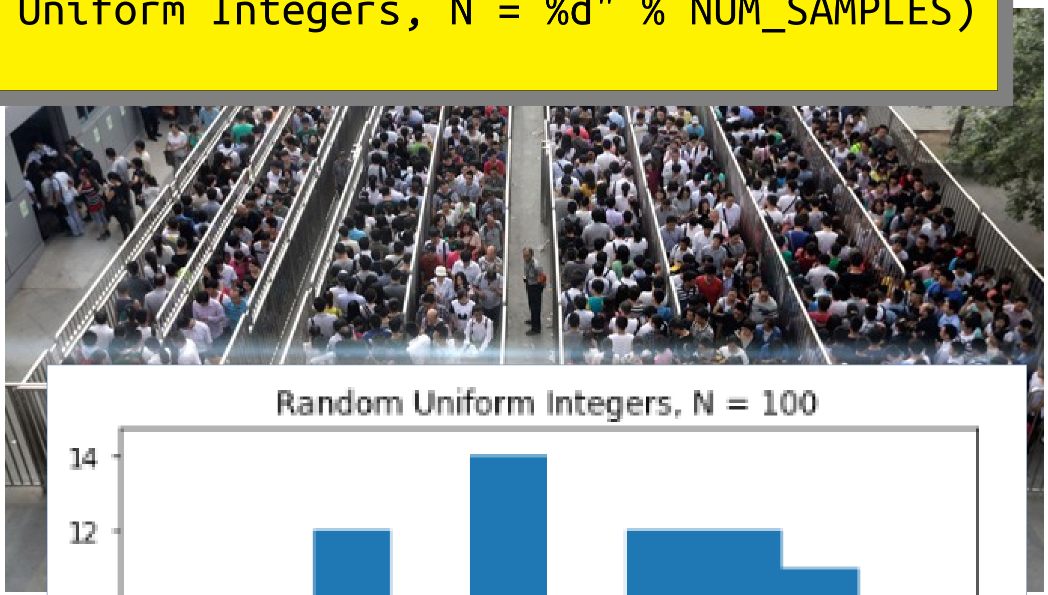


Random uniform numbers

- Consider sim
  Beijing subwa
  procedures as a function
  of weight of individual
  passenger belongings

```
payload_kg = np.random.uniform(low=5, high=31,
                                size=NUM_SAMPLES)
plt.hist(payload_kg)
plt.title("Random Uniform Integers, N = %d" % NUM_SAMPLES)
plt.show()
```

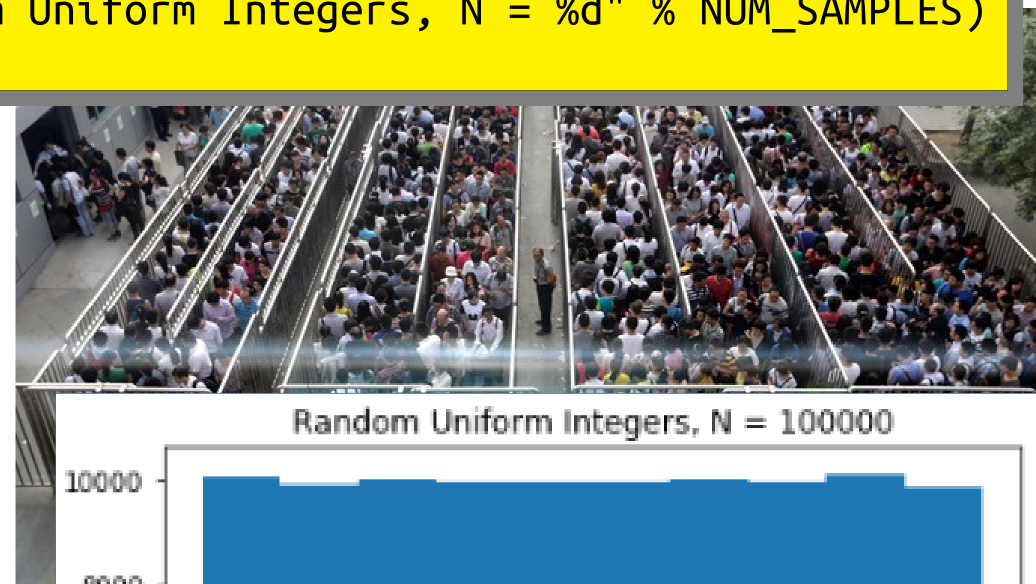
- We will assume that the weight of passenger belongings between 5 and 30kg is equally likely, or uniformly distributed.

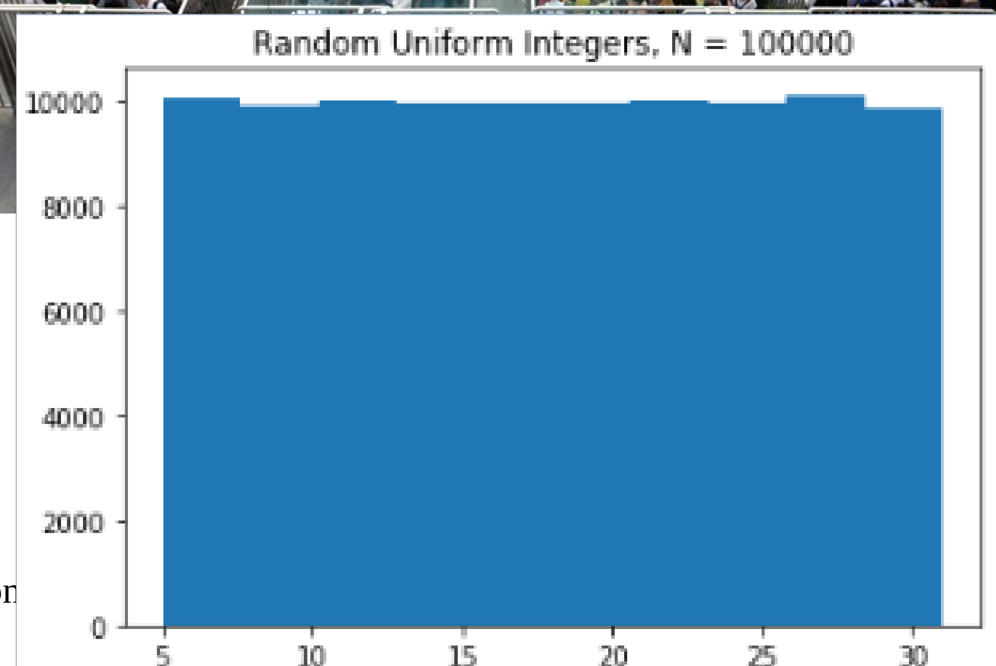


Random uniform numbers

- Consider sim Beijing subwa procedures as a function of weight of individual passenger belongings
- We will assume that the weight of passenger belongings between 5 and 30kg is equally likely, or uniformly distributed.

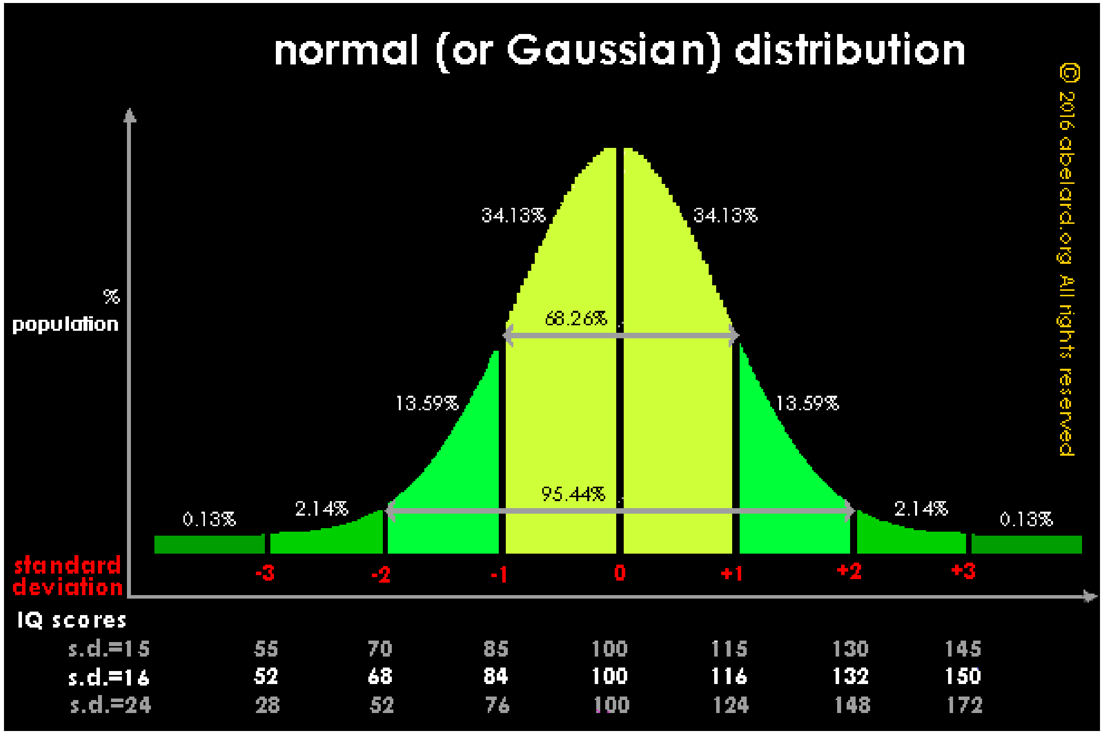
```
payload_kg = np.random.uniform(low=5, high=31,
                                size=NUM_SAMPLES)
plt.hist(payload_kg)
plt.title("Random Uniform Integers, N = %d" % NUM_SAMPLES)
plt.show()
```





Random normal numbers

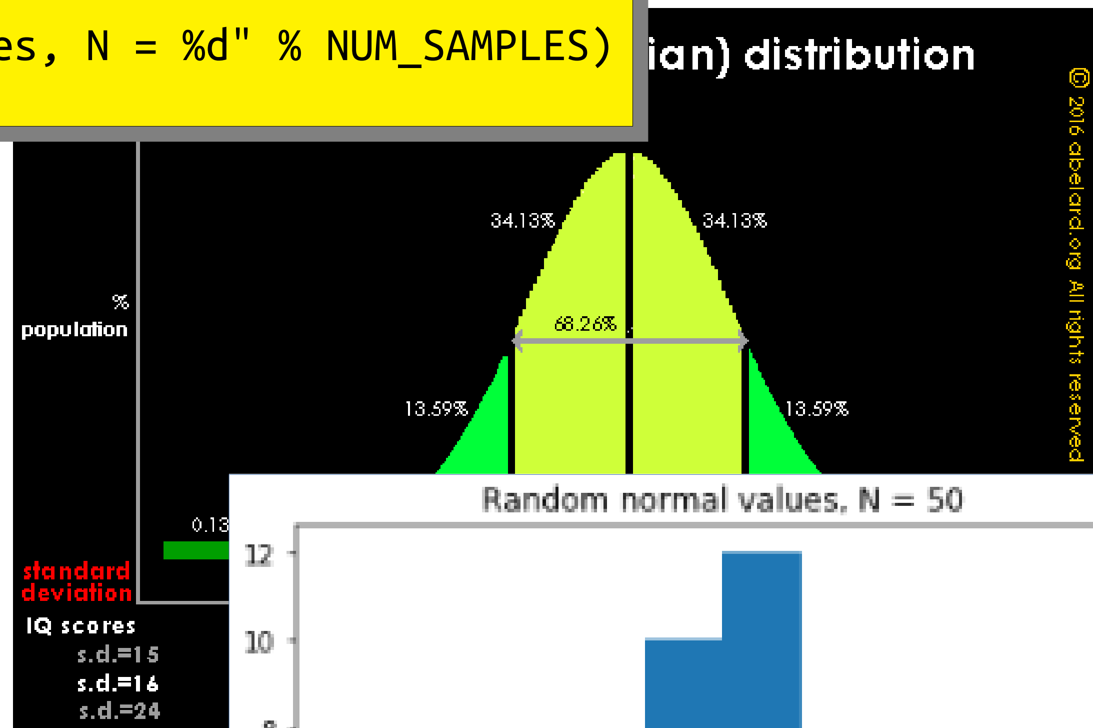
- Consider a simulation of fighter pilot training procedures where intelligence is a primary factor
- We will assume that intelligence is based on IQ, and is normally distributed

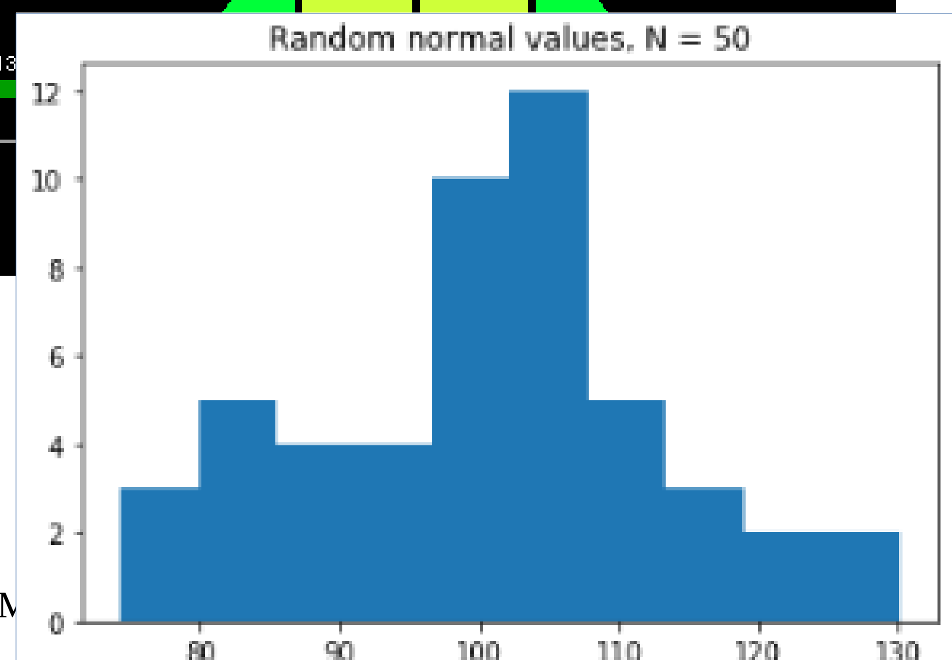


Random normal numbers

```
iq_samples = np.random.normal(loc=100.0, scale=15.0,
                              size=NUM_SAMPLES)
plt.hist(iq_samples)
plt.title("Random normal values, N = %d" % NUM_SAMPLES)
plt.show()
```

- pilot training procedures where intelligence is a primary factor
- We will assume that intelligence is based on IQ, and is normally distributed

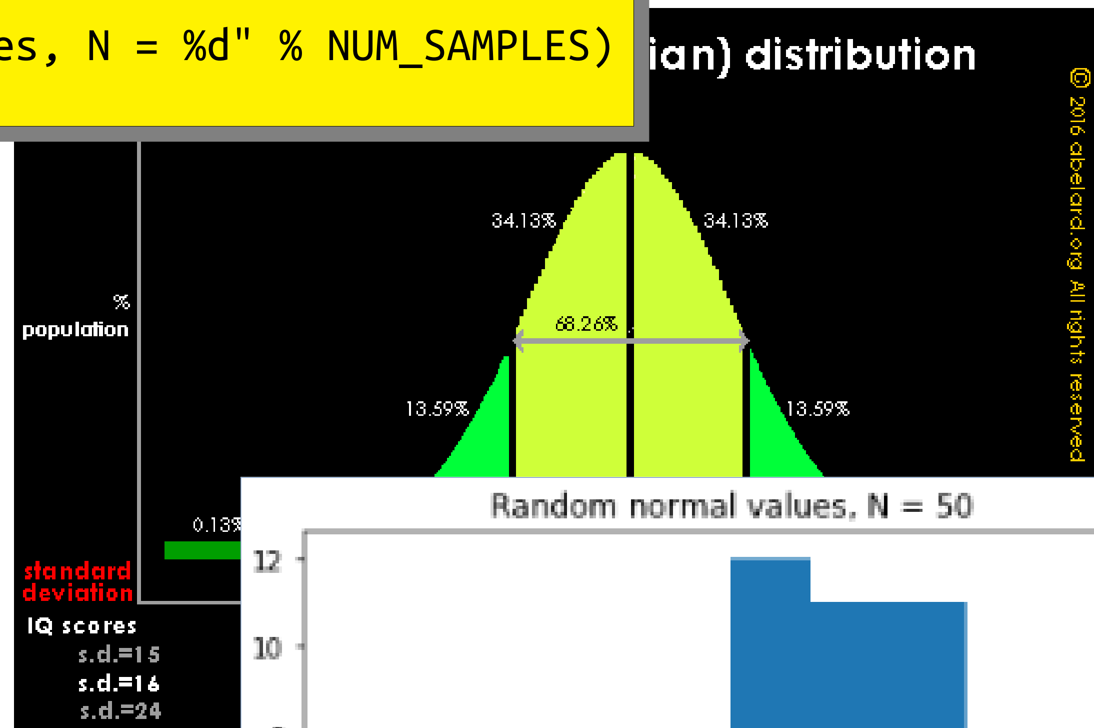


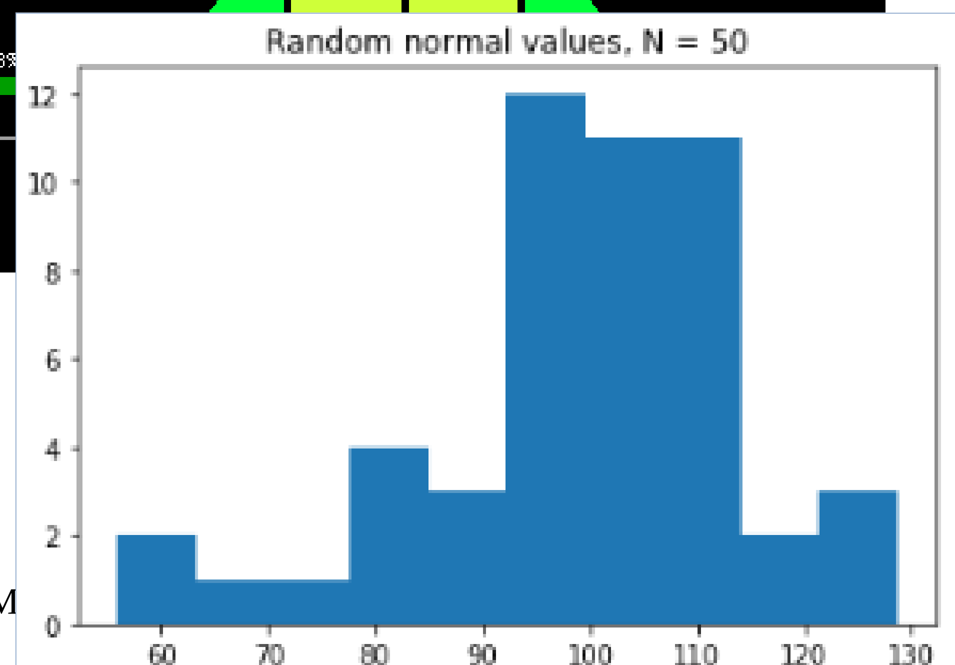


Random normal numbers

```
iq_samples = np.random.normal(loc=100.0, scale=15.0,
                              size=NUM_SAMPLES)
plt.hist(iq_samples)
plt.title("Random normal values, N = %d" % NUM_SAMPLES)
plt.show()
```

- pilot training procedures where intelligence is a primary factor
- We will assume that intelligence is based on IQ, and is normally distributed

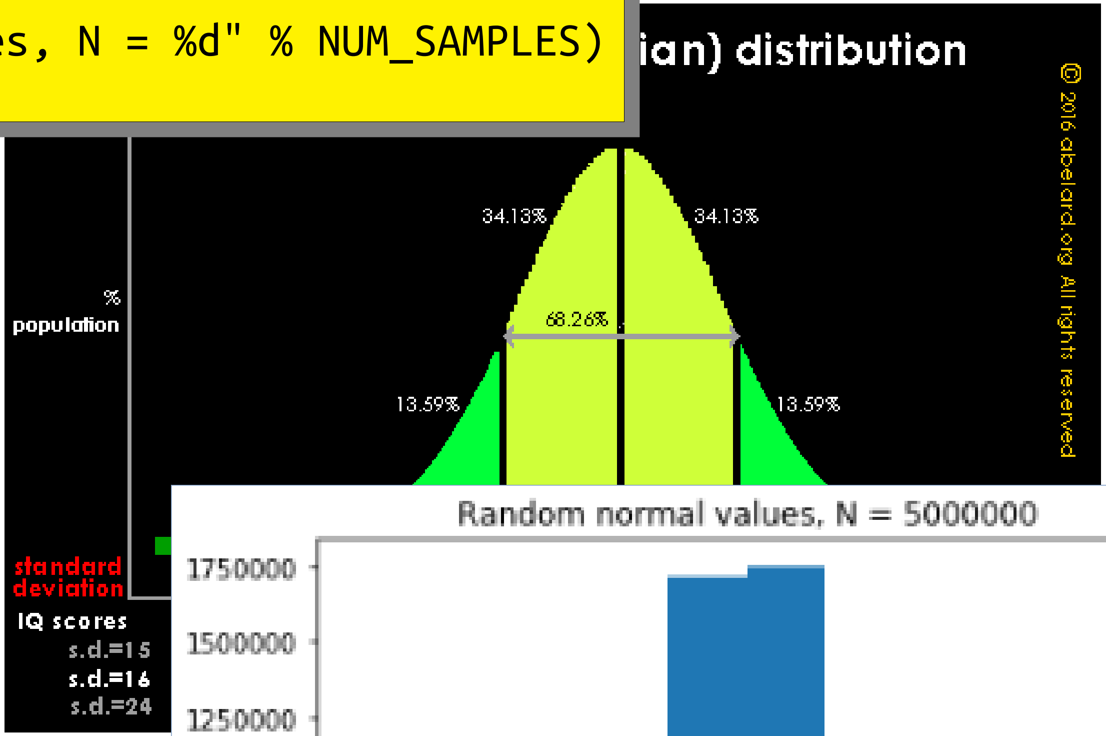


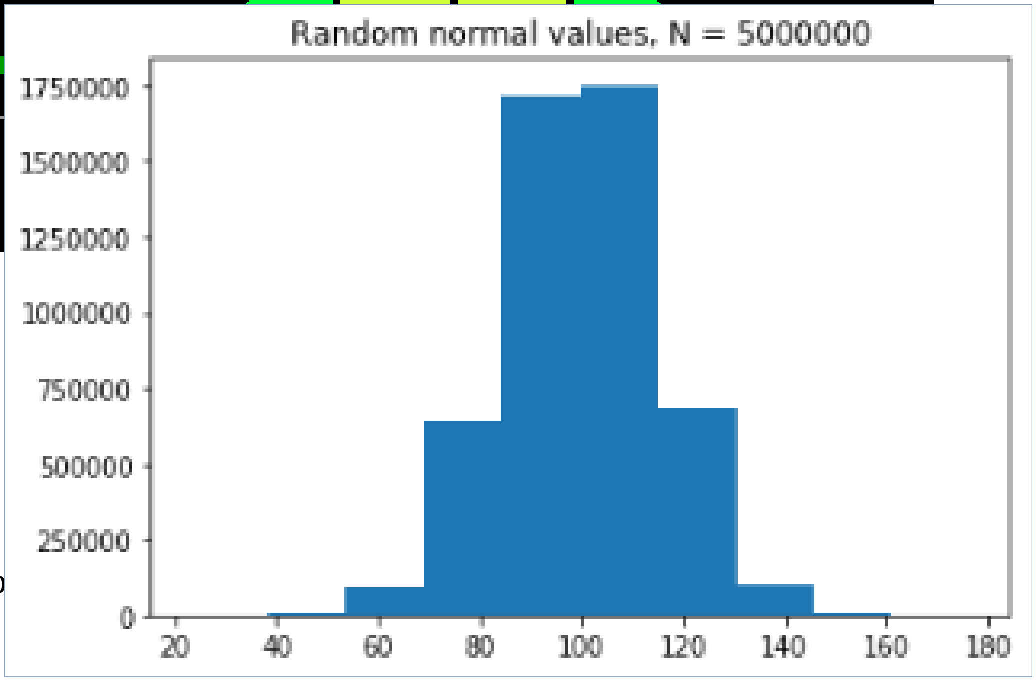


Random normal numbers

```
iq_samples = np.random.normal(loc=100.0, scale=15.0,
                              size=NUM_SAMPLES)
plt.hist(iq_samples)
plt.title("Random normal values, N = %d" % NUM_SAMPLES)
plt.show()
```

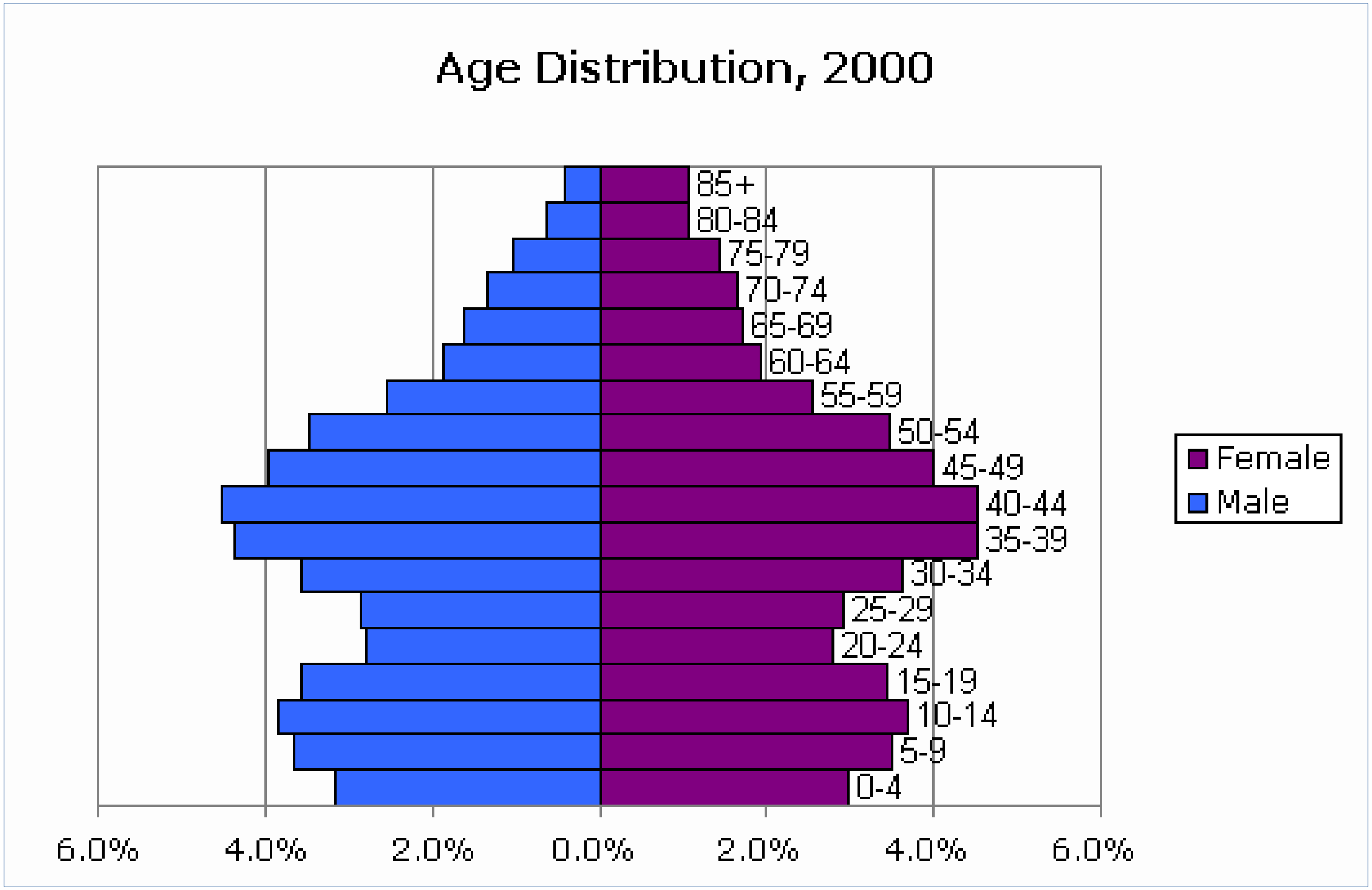
- pilot training procedures where intelligence is a primary factor
- We will assume that intelligence is based on IQ, and is normally distributed





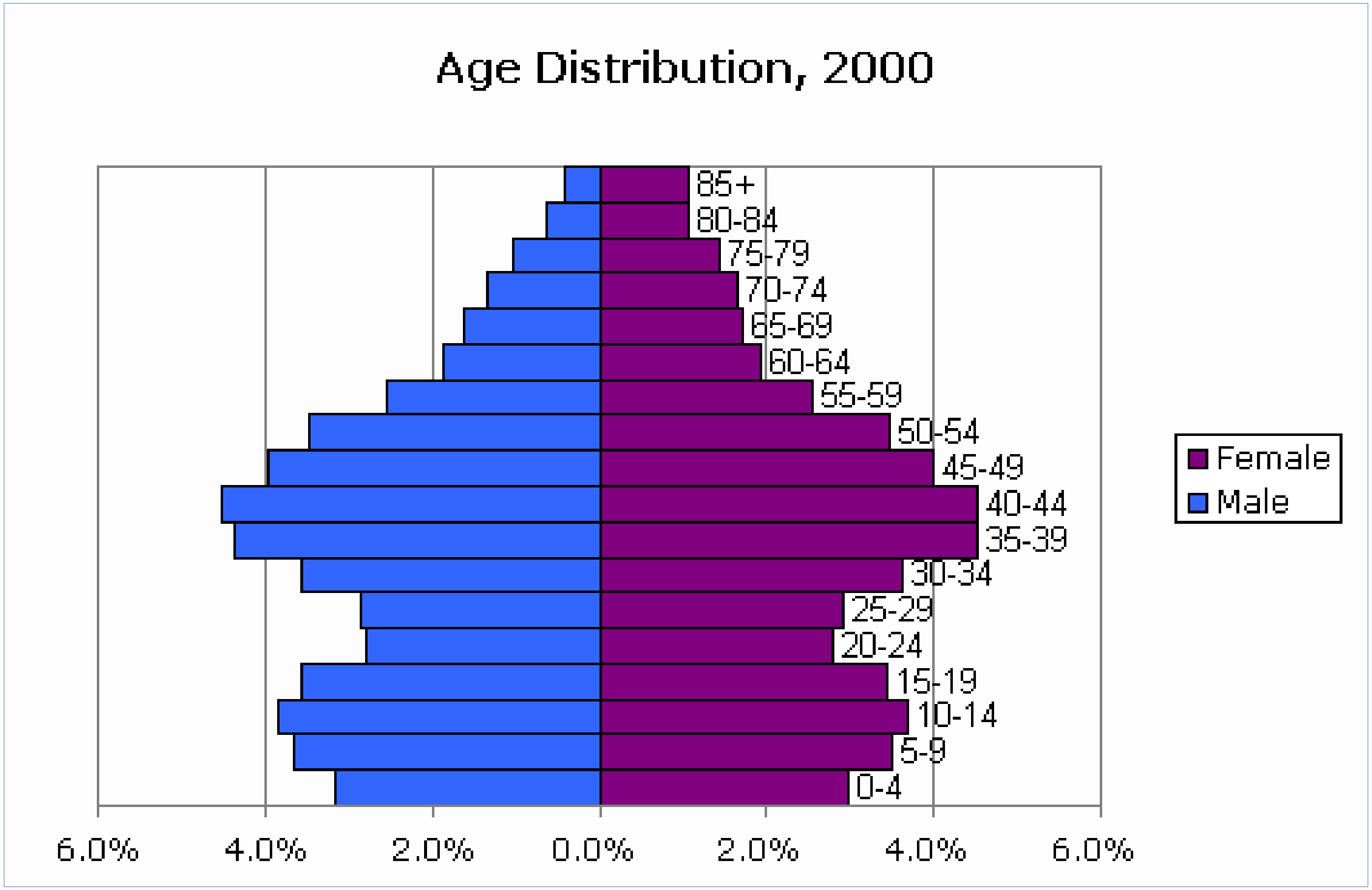
Random custom distribution numbers

- Consider a simulation of different long-term tax scenarios based in large part on the ages of the people in the society
- Assume that we want to use as input the population distribution in 2000



Random custom distribution numbers

- Consider a simulation of different long-term tax scenarios based in large part on the ages of the people in the society
- Assume that we want to use as input the population distribution in 2000



We would want to generate random individuals that fit this profile – 4.5% of the people should be males aged 40-44, 3% should be females aged 20-24, ...

Simulating probabilistic behaviour

- We will start with simple stuff
  - Flipping a fair coin
  - Rolling a fair die
  - Rolling an unfair die

Flipping fair coin

```
import numpy as np

M = 10    # Number of experiments
N = 10   # Number of flips per experiment

# Fair coin
for i in np.arange(M):

    # Array of uniformly distributed random numbers
    r = np.random.uniform(size=N)

    Heads = 0
    Tails = 0
    for s in r:
        if s < 0.5:
            Heads += 1
        else:
            Tails += 1

    print('Heads: %d, Tails %d, Percent Heads %6.4f'
          % (Heads, Tails, Heads/N))
```

Flipping fair coin

```
import numpy as np

M = 10    # Number of experiments
N = 10   # Number of flips per experiment

# Fair coin
for i in np.arange(M):

    # Array of uniformly distributed rando
    r = np.random.uniform(size=N)

    Heads = 0
    Tails = 0
    for s in r:
        if s < 0.5:
            Heads += 1
        else:
            Tails += 1

    print('Heads: %d, Tails %d, Percent Heads %6.4f'
          % (Heads, Tails, Heads/N))
```

```
Heads: 4, Tails 6, Percent Heads 0.4000
Heads: 3, Tails 7, Percent Heads 0.3000
Heads: 5, Tails 5, Percent Heads 0.5000
Heads: 5, Tails 5, Percent Heads 0.5000
Heads: 5, Tails 5, Percent Heads 0.5000
Heads: 6, Tails 4, Percent Heads 0.6000
Heads: 2, Tails 8, Percent Heads 0.2000
Heads: 4, Tails 6, Percent Heads 0.4000
Heads: 3, Tails 7, Percent Heads 0.3000
Heads: 4, Tails 6, Percent Heads 0.4000
```

Flipping fair coin

```
import numpy as np

M = 10    # Number of experiments
N = 10   # Number of flips per experiment

# Fair coin
for i in np.arange(M):

    # Array of uniformly distribu
    r = np.random.uniform(size=N)

    Heads = 0
    Tails = 0
    for s in r:
        if s < 0.5:
            Heads += 1
        else:
            Tails += 1

    print('Heads: %d, Tails %d,
          % (Heads, Tails, Heads,
```

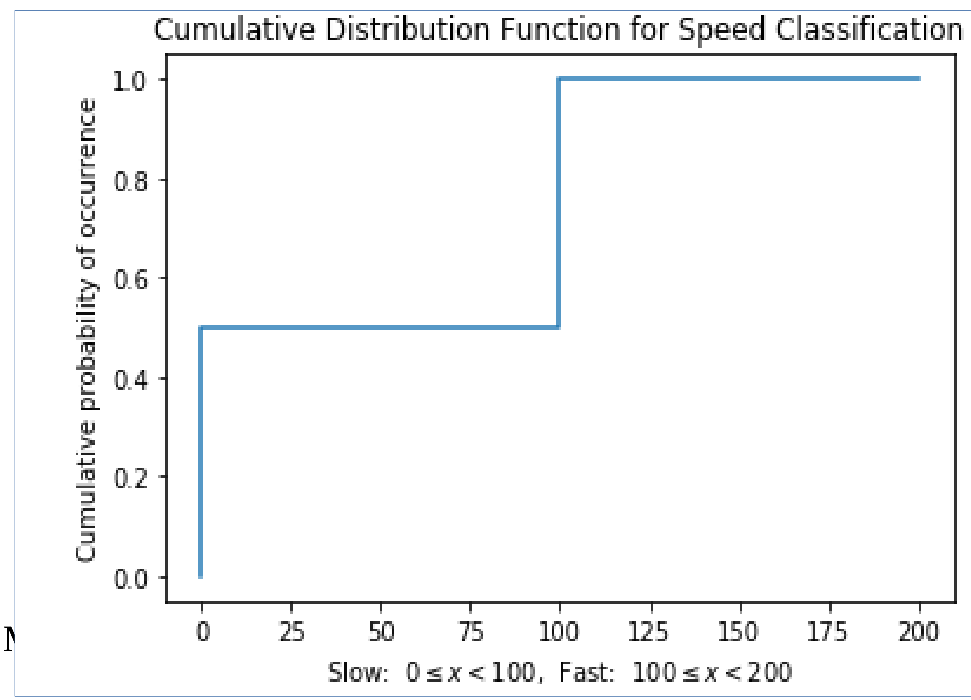
```
Heads: 499630, Tails 500370, Percent Heads 0.4996
Heads: 499350, Tails 500650, Percent Heads 0.4994
Heads: 500686, Tails 499314, Percent Heads 0.5007
Heads: 499491, Tails 500509, Percent Heads 0.4995
Heads: 500806, Tails 499194, Percent Heads 0.5008
Heads: 499388, Tails 500612, Percent Heads 0.4994
Heads: 499665, Tails 500335, Percent Heads 0.4997
Heads: 500087, Tails 499913, Percent Heads 0.5001
Heads: 500128, Tails 499872, Percent Heads 0.5001
Heads: 499694, Tails 500306, Percent Heads 0.4997
```

What we have done – we have generated random events where each outcome has a 50% chance of happening.

Cumulative Distribution Function

- We will make heavy use of these later, so let's get started with a simple example
- Assign numerical categories of [0,100) which always maps to <u>Slow</u>, and [100,200) which always maps to <u>Fast</u>
- If we generate random numbers, 0.3891 will map to <u>Slow</u> and 0.6281 will map to <u>Fast</u>
- The probability of a <u>Slow</u> outcome is 0.5–0.0=0.5, and the probability of a <u>Fast</u> outcome is 1.0-0.5=0.5

Random number interval	Cumulative occurrences	Percent occurrence
$x < 0$	0	0.00
$0 \leq x < 0.5$	0.5	0.50
$0.5 \leq x < 1.0$	1.0	0.50



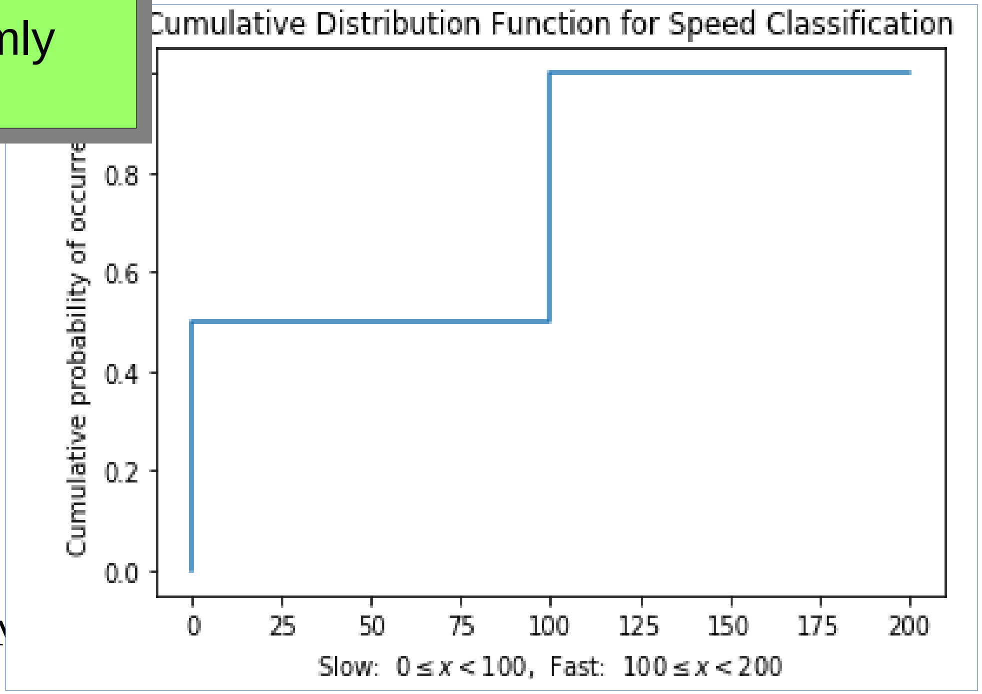
Cumulative Distribution Function

- We will make heavy use of these later, so let's get started with a s
- A [(S n
- If 0 0.6281 will map to Fast
- The probability of a Slow outcome is 0.5–0.0=0.5, and the probability of a Fast outcome is 1.0-0.5=0.5

Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to a speed. So far, all we have is that 0-100 constitutes Slow and 100-200 constitutes Fast, and each category is equally likely to occur.

We need a model to describe how a randomly chosen number will map to a speed

Random ...er ...al	Cumulative occurrences	Percent occurrence
	0	0.00
...< 0.5	0.5	0.50
...x < 1.0	1.0	0.50



Cumulative Distribution Function

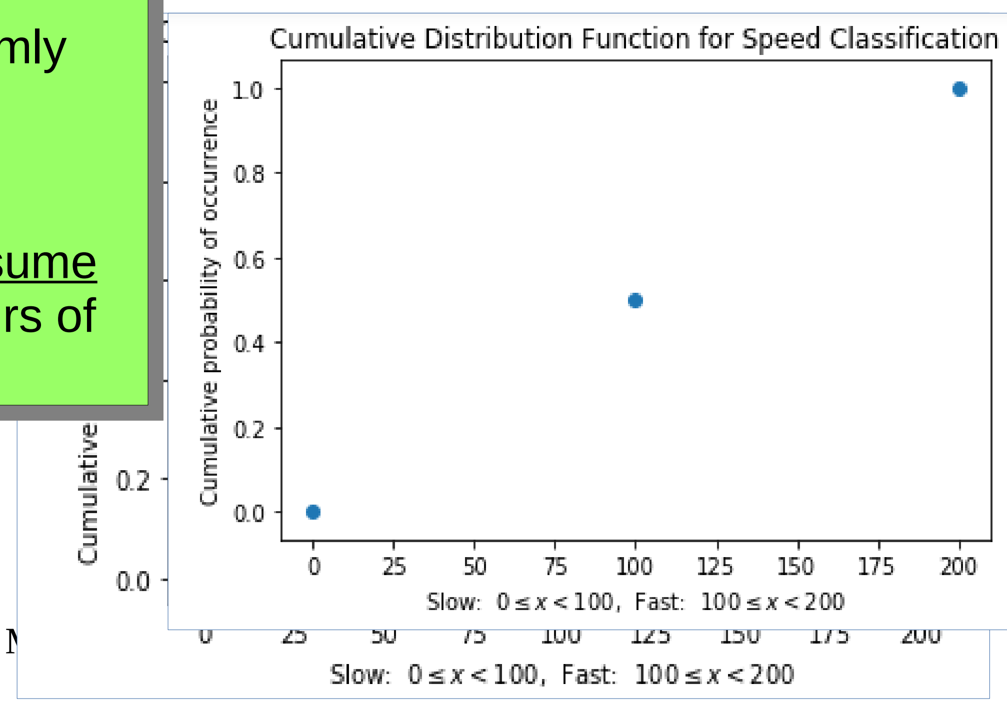
- We will make heavy use of these later, so let's get started with a s
- A [(S n
- If 0 0
- T is of a Fast outcome is 1.0-0.5=0.5

Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to a speed. So far, all we have is that 0-100 constitutes Slow and 100-200 constitutes Fast, and each category is equally likely to occur.

We need a model to describe how a randomly chosen number will map to a speed

To create this model, we'll use the right endpoints of each interval as data, and assume a piecewise linear relationship between pairs of data

Random ...er ...al	Cumulative occurrences	Percent occurrence
	0	0.00
...< 0.5	0.5	0.50
...x < 1.0	1.0	0.50



Cumulative Distribution Function

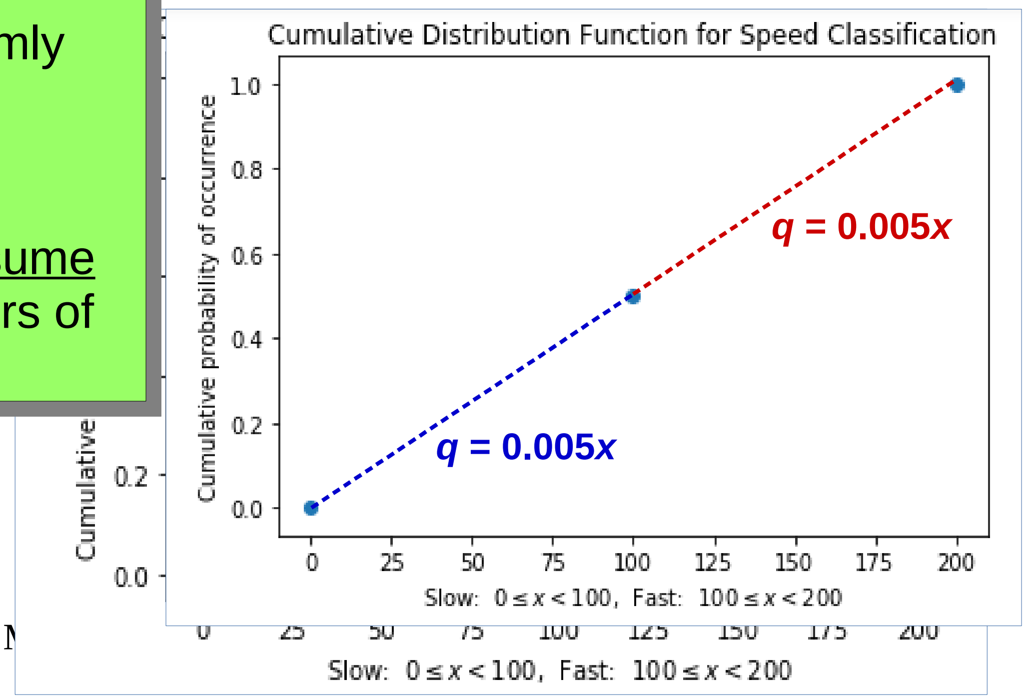
- We will make heavy use of these later, so let's get started with a s
- A [(S n
- If 0 0
- T is of a Fast outcome is 1.0-0.5=0.5

Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to a speed. So far, all we have is that 0-100 constitutes Slow and 100-200 constitutes Fast, and each category is equally likely to occur.

We need a model to describe how a randomly chosen number will map to a speed

To create this model, we'll use the right endpoints of each interval as data, and assume a piecewise linear relationship between pairs of data

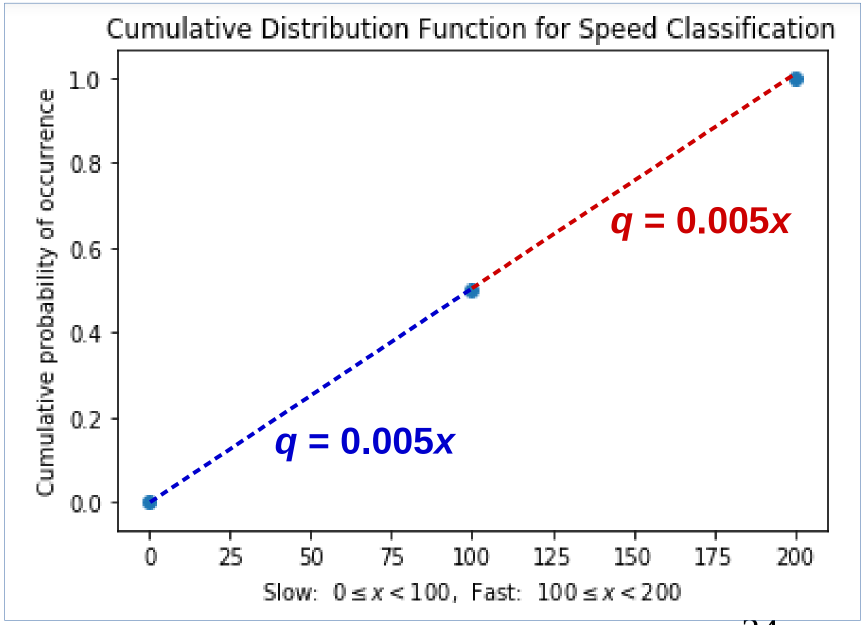
Random ...er ...al	Cumulative occurrences	Percent occurrence
	0	0.00
< 0.5	0.5	0.50
< 1.0	1.0	0.50



Cumulative Distribution Function

- We want to generate random numbers, *q*, and have each map to a speed, given our linear assumptions
- We need the <u>inverse</u> of the model functions in each step-wise interval

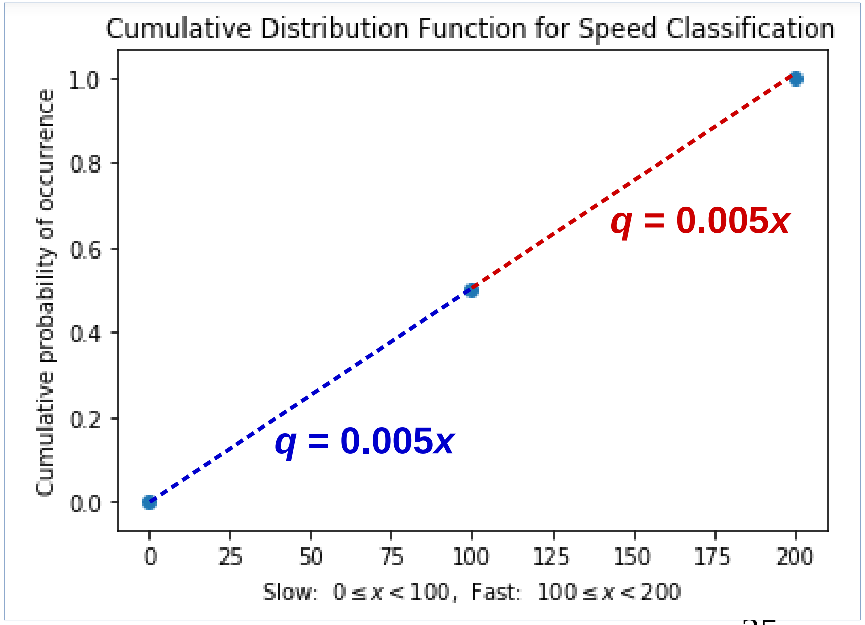
$$x = 200q, \text{ for } 0 \leq q < 0.5$$
$$x = 200q, \text{ for } 0.5 \leq q < 1$$



Cumulative Distribution Function

- We want to generate random numbers, *q*, and have each map to a speed, given our linear assumptions
- We need the <u>inverse</u> of the model functions in each step-wise interval

0.8293	166
0.0272	5
0.5523	110
0.7429	149
0.9872	197
0.0369	7
0.2478	50
0.7958	159
0.6430	129
0.1103	22

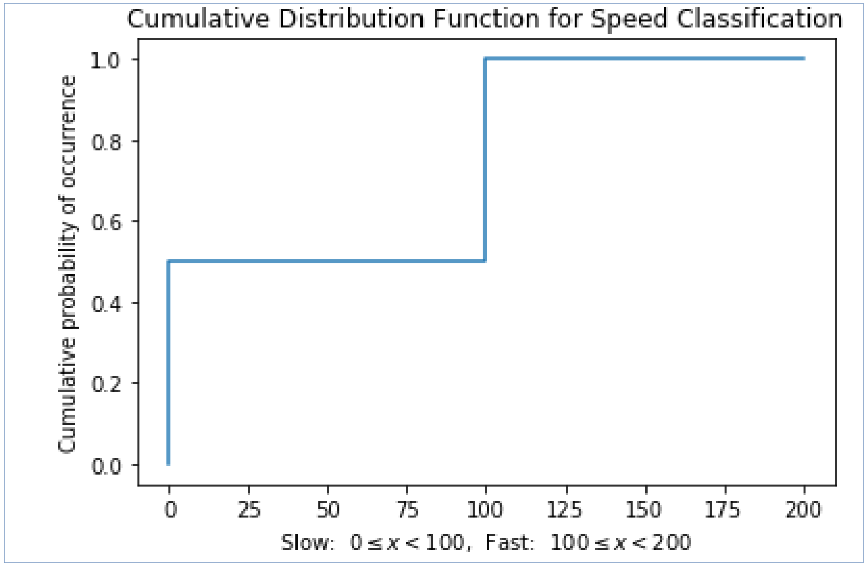


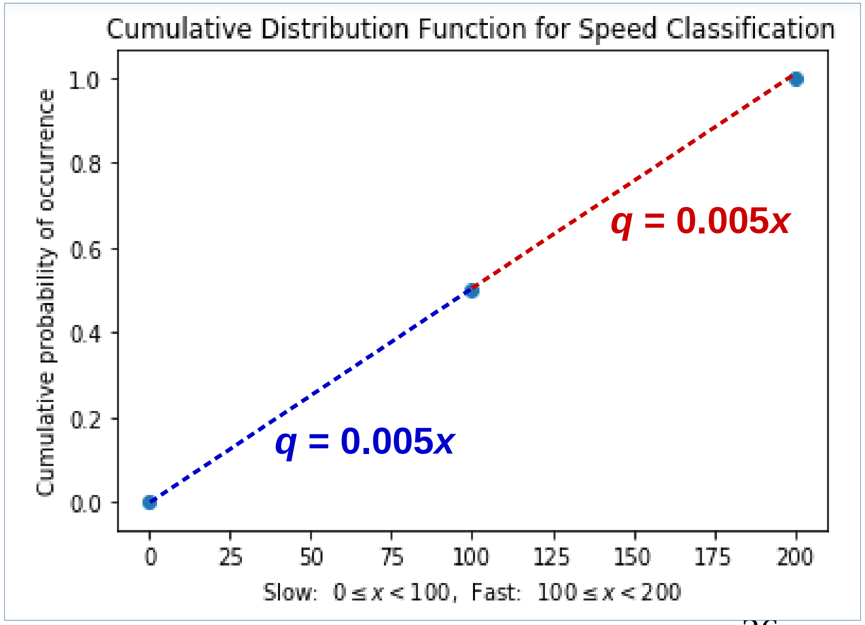
$$x = 200q, \ \text{for } 0 \leq q < 0.5$$
$$x = 200q, \ \text{for } 0.5 \leq q < 1$$

Cumulative Distribution Function

What have we done?

- Started with classification of Slow (0-100) and Fast (100-200) speeds, with Slow and Fast speeds equally likely to occur
- Created a cumulative distribution function (CDF) to lay out the probabilities of occurrence
- Using three right endpoints of histogram, created piecewise interpolating polynomial in each interval (we could have chosen quadratic, or cubic,...)
- Created inverse functions for each interval, so that every time we get a random number between 0 and 1, we can map it to a numeric speed. Fast speeds and Slow speeds will be equally likely to be chosen, and we'll get a range in each category
- Now, we are able to drive a simulation that needs a roughly equal (and varying) proportion of random Fast and Slow speeds





ing

Roll of a fair die

Roll value	P(roll)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

```
M = 10   # Number experiments
N = 10   # Number rolls

print('N = %d rolls' % N)
print('   1       2       3       4       5       6')
for i in np.arange(M):
    count1 = count2 = count3 = count4 = count5 = count6 = 0

    r = np.random.uniform(size=N)
    for s in r:
        if 0 <= s < 1.0/6.0:
            count1 += 1
        elif 1.0/6.0 <= s < 2.0/6.0:
            count2 += 1
        elif 2.0/6.0 <= s < 3.0/6.0:
            count3 += 1
        elif 3.0/6.0 <= s < 4.0/6.0:
            count4 += 1
        elif 4.0/6.0 <= s < 5.0/6.0:
            count5 += 1
        elif 5.0/6.0 <= s < 6.0/6.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

Roll value	P(roll)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

Roll of a fair die

```
N = 10 rolls
   1        2        3        4        5        6
0.1000   0.1000   0.1000   0.2000   0.0000   0.5000
0.1000   0.3000   0.1000   0.2000   0.1000   0.2000
0.2000   0.4000   0.2000   0.2000   0.0000   0.0000
0.1000   0.0000   0.2000   0.2000   0.2000   0.3000
0.0000   0.0000   0.0000   0.4000   0.3000   0.3000
0.0000   0.3000   0.1000   0.1000   0.2000   0.3000
0.4000   0.0000   0.0000   0.1000   0.5000   0.0000
0.3000   0.1000   0.1000   0.2000   0.1000   0.2000
0.1000   0.1000   0.1000   0.2000   0.0000   0.5000
0.0000   0.1000   0.2000   0.3000   0.2000   0.2000
```

Individual Experiments

```
M = 10   # Numb
N = 10   # Numb

print('N = %d r
print('   1
for i in np.ara
    count1 = co

    r = np.rand
    for s in r:
        if 0 <= s < 1.0/6.0:
            count1 += 1
        elif 1.0/6.0 <= s < 2.0/6.0:
            count2 += 1
        elif 2.0/6.0 <= s < 3.0/6.0:
            count3 += 1
        elif 3.0/6.0 <= s < 4.0/6.0:
            count4 += 1
        elif 4.0/6.0 <= s < 5.0/6.0:
            count5 += 1
        elif 5.0/6.0 <= s < 6.0/6.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

Roll of a fair die

Roll value	P(roll)
1	1/6
2	1/6
3	1/6
4	1/6
5	1/6
6	1/6

```
M = 10   # Numb
N = 10   # Numb

print('N = %d r
print('   1
for i in np.ara
    count1 = co

    r = np.rand
    for s in r:
        if 0 <= s <
            count1 +
        elif 1.0/6.0
            count2 += 1
        elif 2.0/6.0 <= s < 3.0/6.0:
            count3 += 1
        elif 3.0/6.0 <= s < 4.0/6.0:
            count4 += 1
        elif 4.0/6.0 <= s < 5.0/6.0:
            count5 += 1
        elif 5.0/6.0 <= s < 6.0/6.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

```
N = 1000000 rolls
   1        2        3        4        5        6
0.1665   0.1665   0.1666   0.1670   0.1670   0.1665
0.1673   0.1672   0.1667   0.1665   0.1659   0.1664
0.1665   0.1664   0.1670   0.1664   0.1667   0.1671
0.1667   0.1659   0.1668   0.1665   0.1671   0.1669
0.1662   0.1664   0.1665   0.1668   0.1670   0.1671
0.1666   0.1667   0.1663   0.1667   0.1670   0.1666
0.1660   0.1664   0.1666   0.1677   0.1666   0.1667
0.1667   0.1670   0.1667   0.1667   0.1663   0.1666
0.1665   0.1667   0.1674   0.1665   0.1664   0.1663
0.1666   0.1667   0.1667   0.1669   0.1661   0.1670
```

Individual Experiments

What we have done – we have generated random events where each outcome has a 1/6 probability of happening.

Roll of a weighted die

Roll value	P(roll)
1	0.1
2	0.1
3	0.2
4	0.3
5	0.2
6	0.1

```
M = 10   # Number experiments
N = 10   # Number rolls

print('N = %d rolls' % N)
print('   1       2       3       4       5       6')
for i in np.arange(M):
    count1 = count2 = count3 = count4 = count5 = count6 = 0

    r = np.random.uniform(size=N)
    for s in r:
        if 0 <= s < 0.1:
            count1 += 1
        elif 0.1 <= s < 0.2:
            count2 += 1
        elif 0.2 <= s < 0.4:
            count3 += 1
        elif 0.4 <= s < 0.7:
            count4 += 1
        elif 0.7 <= s < 0.9:
            count5 += 1
        elif 0.9 <= s < 1.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

With a good random number generator, this case should happen roughly 30% of the time

Roll value	P(roll)
1	0.1
2	0.1
3	0.2
4	0.3
5	0.2
6	0.1

Roll of a weighted die

```
N = 10 rolls
   1       2       3       4       5       6
0.1000  0.1000  0.4000  0.3000  0.0000  0.1000
0.0000  0.0000  0.2000  0.3000  0.4000  0.1000
0.0000  0.1000  0.2000  0.3000  0.3000  0.1000
0.0000  0.1000  0.1000  0.2000  0.2000  0.4000
0.0000  0.0000  0.1000  0.3000  0.4000  0.2000
0.2000  0.0000  0.1000  0.3000  0.2000  0.2000
0.2000  0.1000  0.2000  0.3000  0.2000  0.0000
0.1000  0.0000  0.3000  0.1000  0.5000  0.0000
0.3000  0.0000  0.1000  0.5000  0.1000  0.0000
0.1000  0.1000  0.2000  0.3000  0.3000  0.0000
```

Individual Experiments

```
M = 10   # Numb
N = 10   # Numb

print('N = %d r
print('   1
for i in np.ara
    count1 = co

    r = np.rand
    for s in r:
        if 0 <= s < 0.1:
            count1 += 1
        elif 0.1 <= s < 0.2:
            count2 += 1
        elif 0.2 <= s < 0.4:
            count3 += 1
        elif 0.4 <= s < 0.7:
            count4 += 1
        elif 0.7 <= s < 0.9:
            count5 += 1
        elif 0.9 <= s < 1.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

With a good random number generator, this case should happen roughly 30% of the time

Roll value	P(roll)
1	0.1
2	0.1
3	0.2
4	0.3
5	0.2
6	0.1

Roll of a weighted die

```
N = 1000000 rolls
   1       2       3       4       5       6
0.0997  0.1001  0.2001  0.3000  0.1999  0.1001
0.1000  0.0998  0.1999  0.3004  0.1997  0.1003
0.1002  0.1000  0.1995  0.2999  0.2006  0.0998
0.0995  0.1001  0.1993  0.3005  0.2001  0.1006
0.0995  0.1002  0.2004  0.2994  0.2006  0.1000
0.1003  0.0995  0.2001  0.3001  0.1993  0.1007
0.1002  0.0995  0.1996  0.3007  0.2002  0.0998
0.0993  0.1005  0.1995  0.3005  0.2002  0.1000
0.1003  0.0991  0.2000  0.3004  0.1999  0.1003
0.1000  0.1000  0.2004  0.2995  0.1996  0.1005
```

Individual Experiments

```
M = 10   # Numb
N = 10   # Numb

print('N = %d r
print('   1
for i in np.ara
    count1 = co

    r = np.rand
    for s in r:
        if 0 <= s <
            count1 +
        elif 0.1 <=
            count2 +
        elif 0.2 <=
            count3 +
        elif 0.4 <= s < 0.7:
            count4 += 1
        elif 0.7 <= s < 0.9:
            count5 += 1
        elif 0.9 <= s < 1.0:
            count6 += 1

    print('{:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}  {:6.4f}'.format(
        count1/N, count2/N, count3/N,
        count4/N, count5/N, count6/N))
```

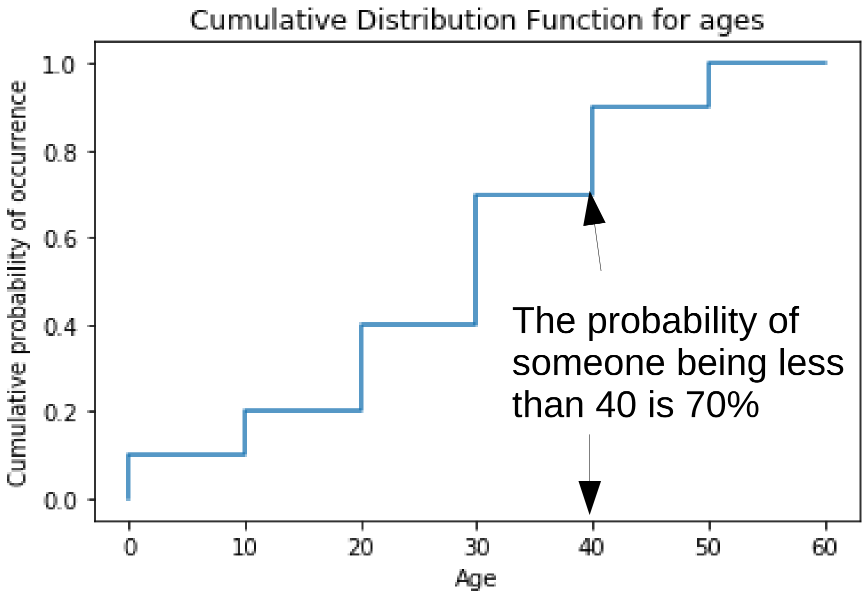
What we have done – we have generated random events where one outcome has a 30% chance of happening, two outcomes have a 20% chance of happening, and the other three have a 10% chance of happening.

happen roughly 30% of the time

A CDF for a similar scenario

- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and map them to ages, so that the ages are representative of the population distribution. If we randomly selected ages between 0 and 60 the young and old would be overrepresented

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10

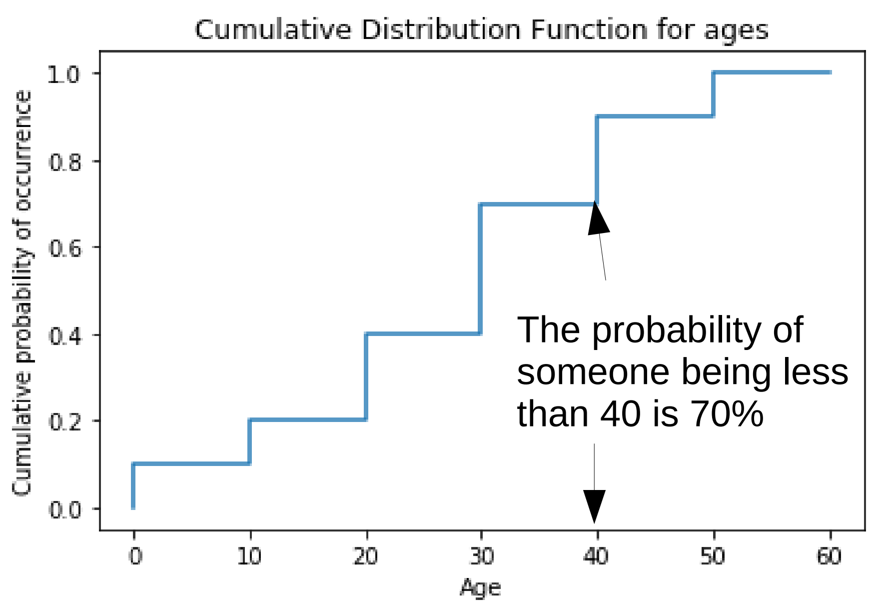


A CDF for a similar scenario

- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and map them to ages, so that the
  tribution. If we randomly selected
  ould be overrepresented

Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to an age.

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10



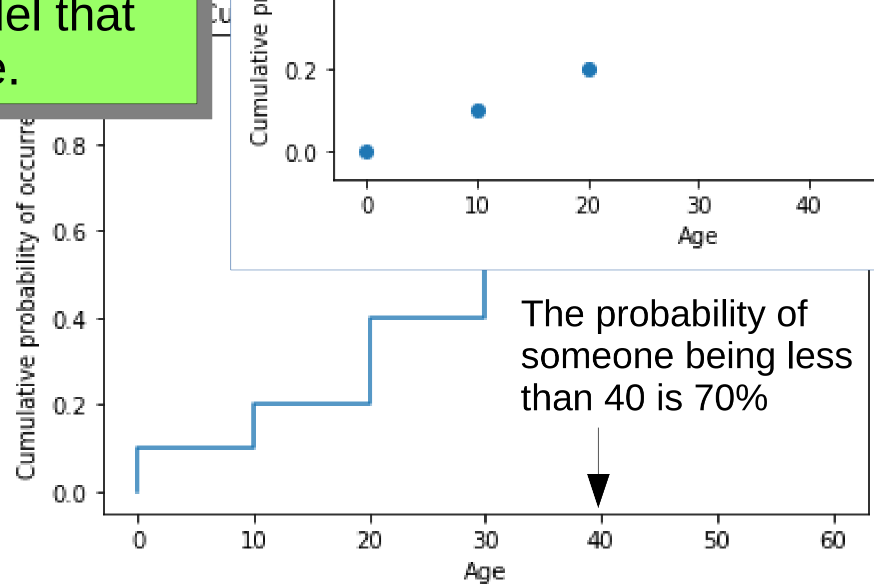
A CDF for a similar scenario

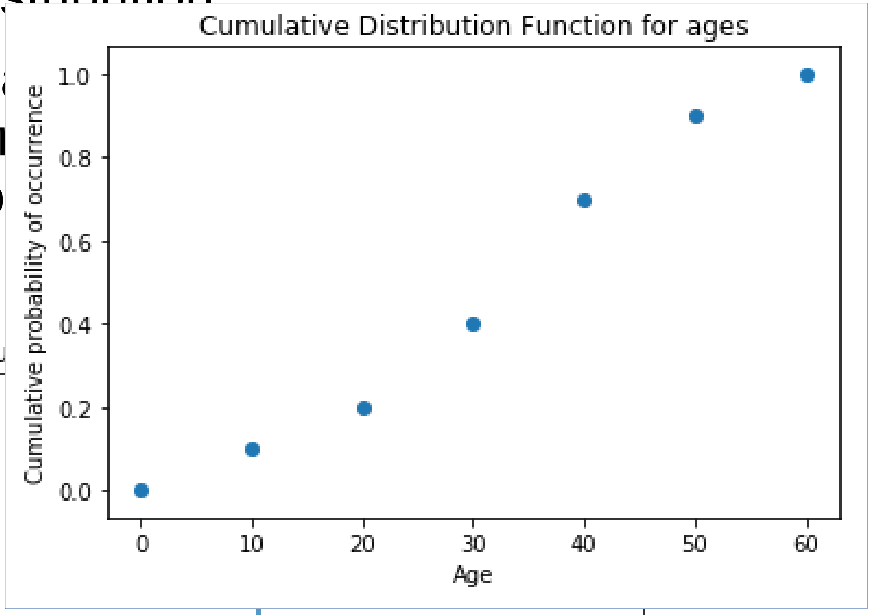
- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and ma

Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to an age.

Plot the endpoints, and determine a model that relates age and probability of occurrence.

0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10





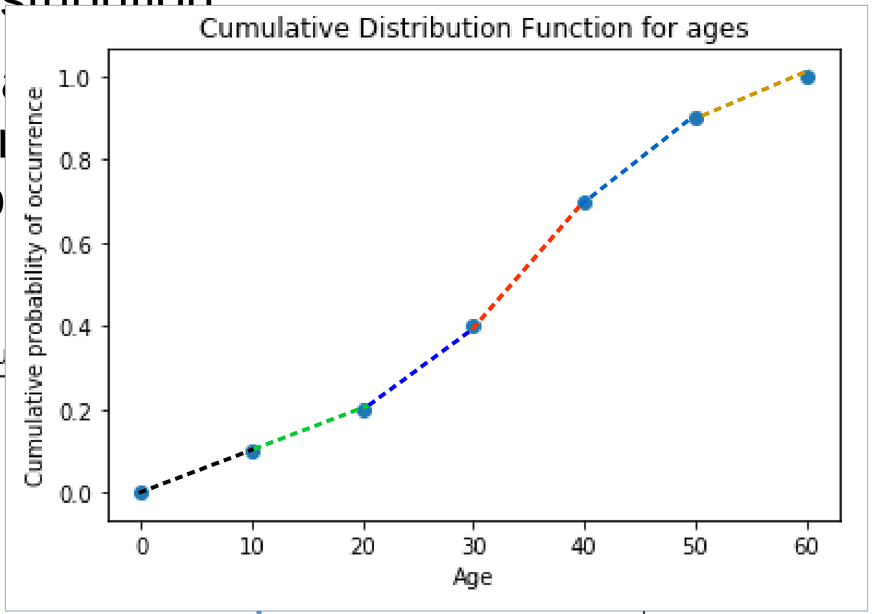
A CDF for a similar scenario

- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and ma

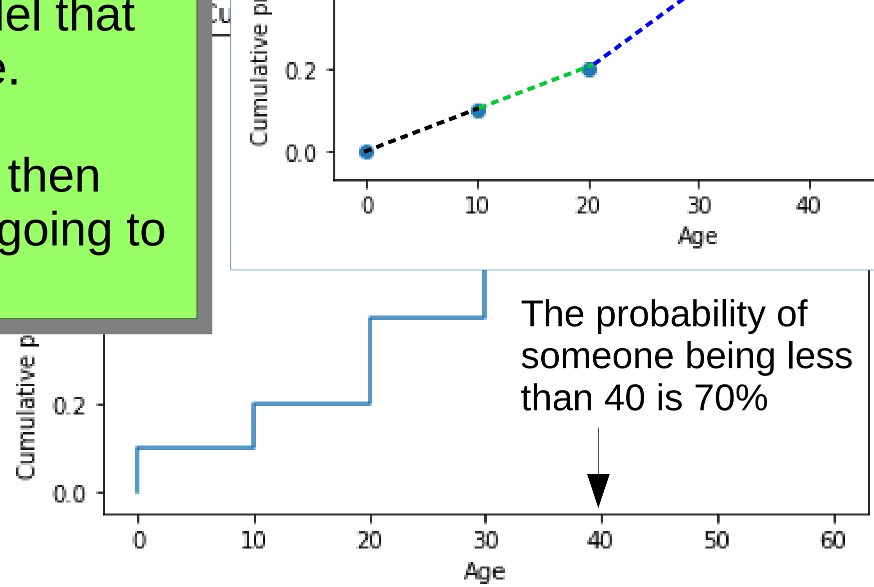
Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to an age.

Plot the endpoints, and determine a model that relates age and probability of occurrence.

I could try a cubic, or a cubic spline, and then derive the inverse functions, but I'm just going to go with <u>piecewise linear</u> for simplicity.



30-40	30
40-50	20
50-60	10



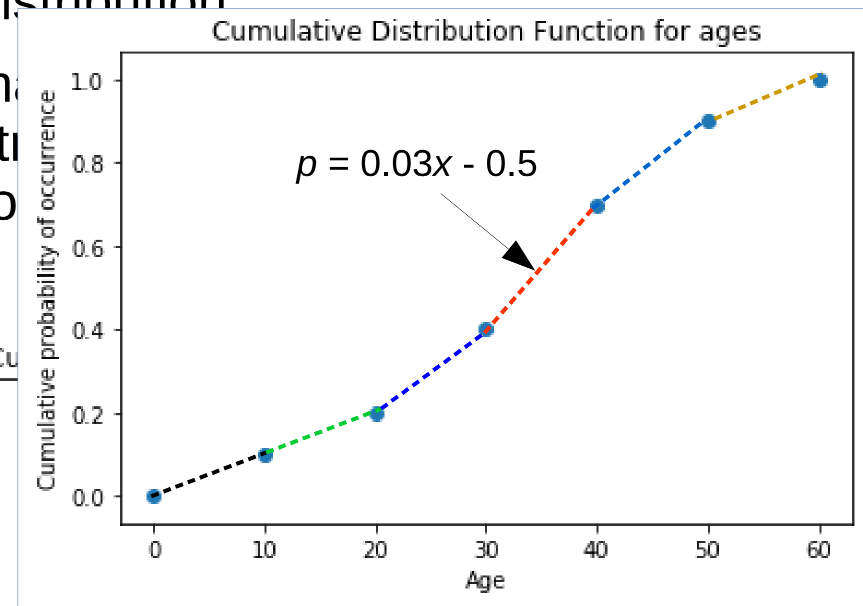
A CDF for a similar scenario

- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and ma

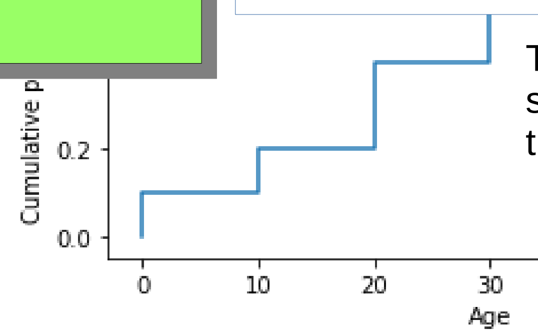
Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to an age.

Plot the endpoints, and determine a model that relates age and probability of occurrence.

I could try a cubic, or a cubic spline, and then derive the inverse functions, but I'm just going to go with piecewise linear for simplicity.



30-40	30
40-50	20
50-60	10




```
Piecewise params
Intvl    m      b
1      0.01   0.0
2      0.01   0.0
3      0.02  -0.2
4      0.03  -0.5
5      0.02  -0.1
6      0.01   0.4
```

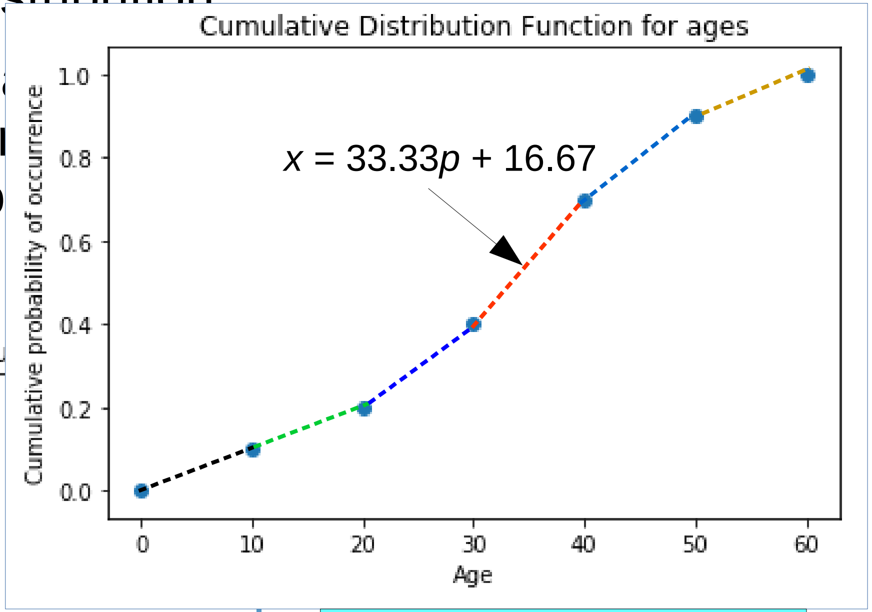
A CDF for a similar scenario

- Consider a model (a survey?) where we want to randomly select people based on their age
- The population of interest has the following distribution
- We want to generate random numbers and ma

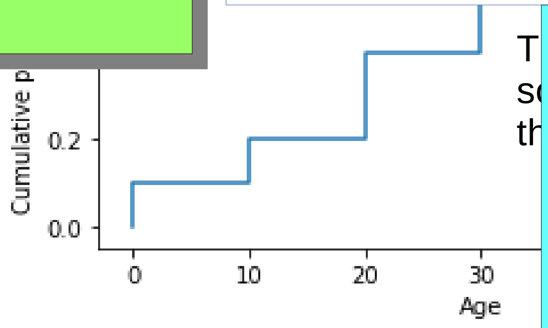
Where we want to go – be able to generate a series of random numbers to drive a simulation, mapping a random number to an age.

Plot the endpoints, and determine a model that relates age and probability of occurrence.

I could try a cubic, or a cubic spline, and then derive the inverse functions, but I'm just going to go with <u>piecewise linear</u> for simplicity.



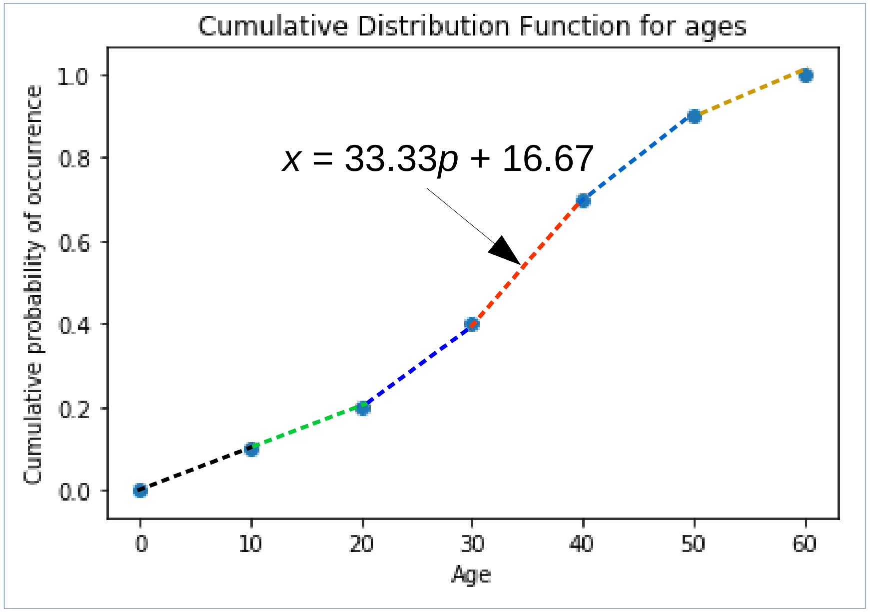
30-40	30
40-50	20
50-60	10




```
Piecewise params
for inverse funcs
Intvl    minv     binv
1      100.00    -0.00
2      100.00    -0.00
3       50.00    10.00
4       33.33    16.67
5       50.00     5.00
6      100.00   -40.00
```

A CDF for a similar scenario

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10

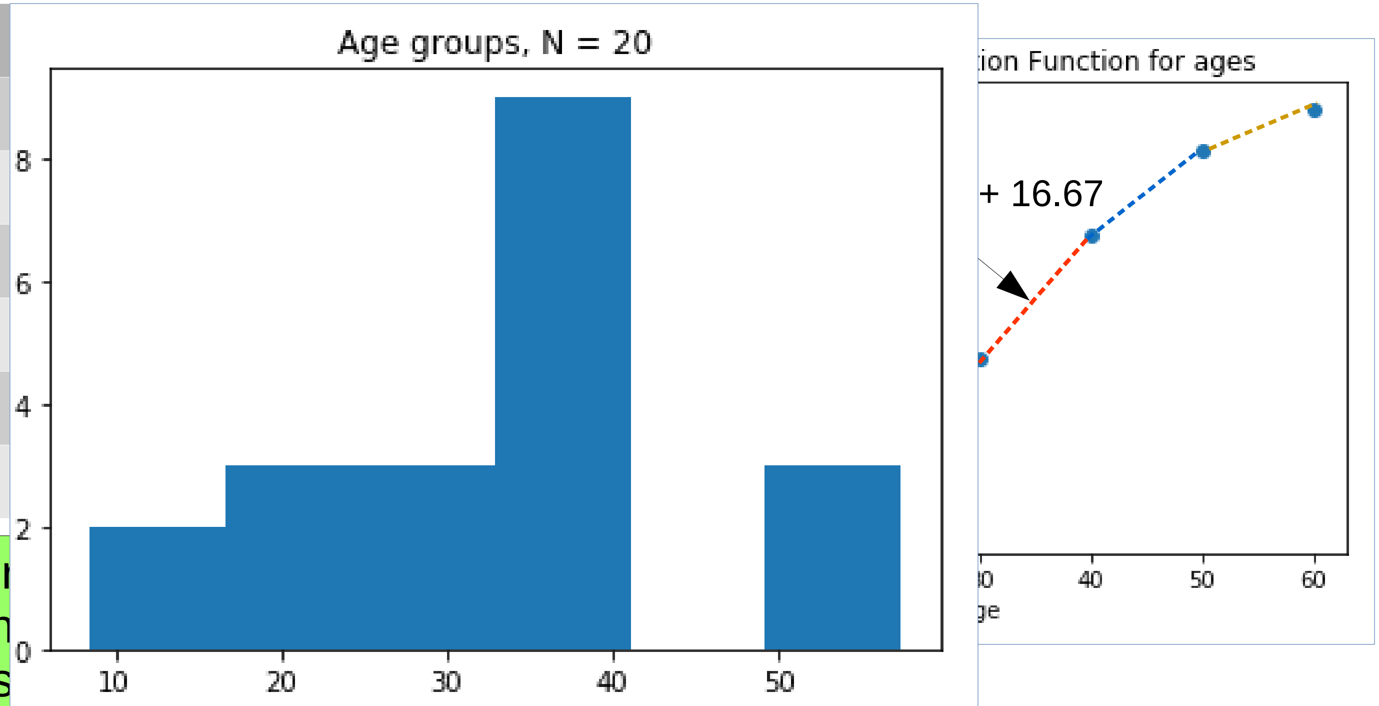


Now, I can generate random numbers, for each one determining the interval it belongs in, then using the appropriate inverse piecewise function to map to an age.

$$\begin{aligned}
0.0 \leq \quad p \quad &< 0.1, \quad x \quad = \quad 100p \\
0.1 \leq \quad p \quad &< 0.2, \quad x \quad = \quad 100p \\
0.2 \leq \quad p \quad &< 0.4, \quad x \quad = \quad 50p + 10 \\
0.4 \leq \quad p \quad &< 0.7, \quad x \quad = \quad 33.33p + 16.67 \\
0.7 \leq \quad p \quad &< 0.9, \quad x \quad = \quad 50p + 5 \\
0.9 \leq \quad p \quad &< 1.0, \quad x \quad = \quad 100p - 40
\end{aligned}$$

A CDF for a similar scenario

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10



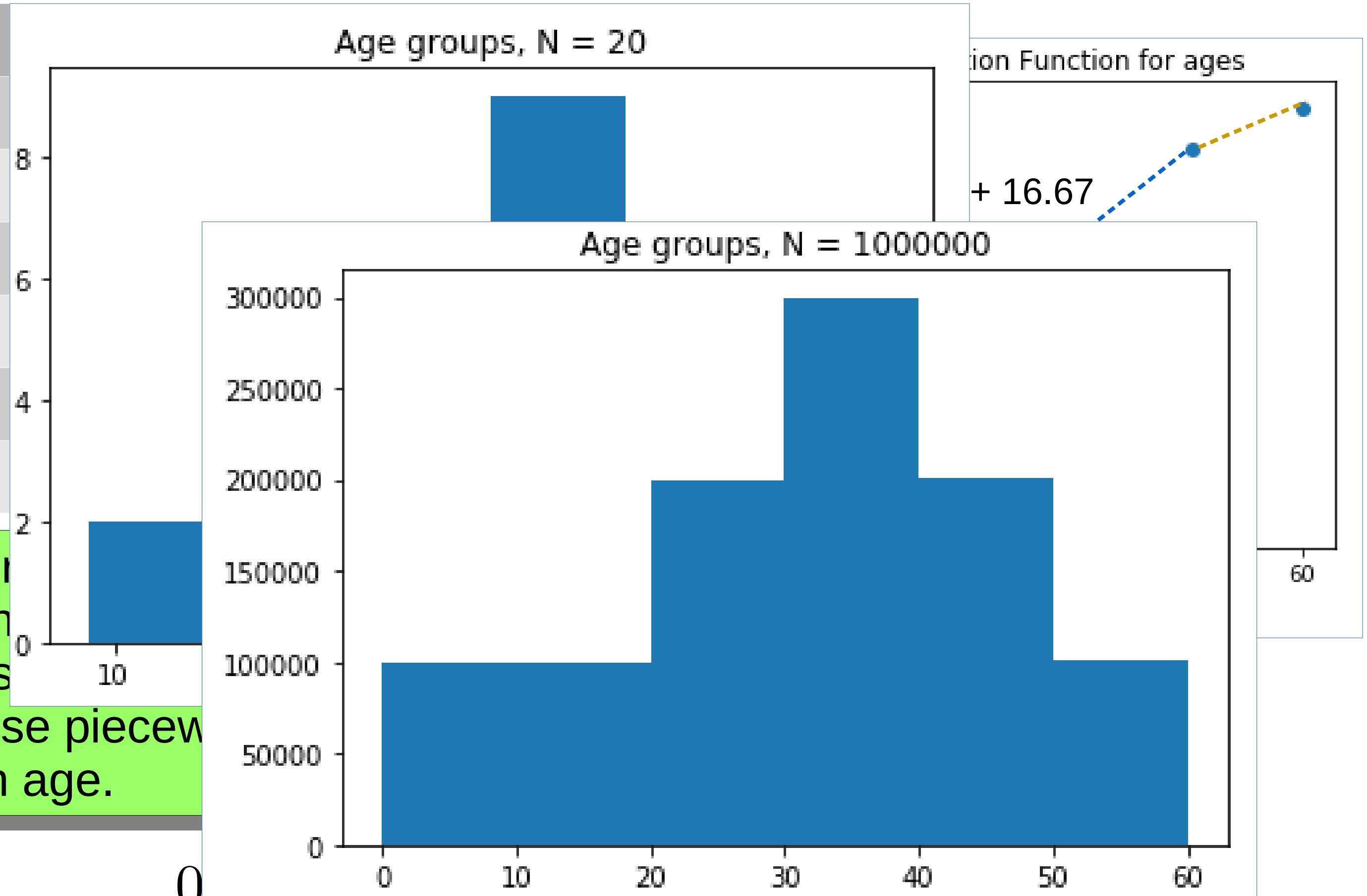
Now, I can generate r
numbers, for each on
the interval it belongs
the appropriate inverse piecewise
function to map to an age.

$$
\begin{array}{lclcl}
0.0 \leq & p & < 0.1, & x & = & 100p \\
0.1 \leq & p & < 0.2, & x & = & 100p \\
0.2 \leq & p & < 0.4, & x & = & 50p + 10 \\
0.4 \leq & p & < 0.7, & x & = & 33.33p + 16.67 \\
0.7 \leq & p & < 0.9, & x & = & 50p + 5 \\
0.9 \leq & p & < 1.0, & x & = & 100p - 40
\end{array}
$$

A CDF for a similar scenario

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10

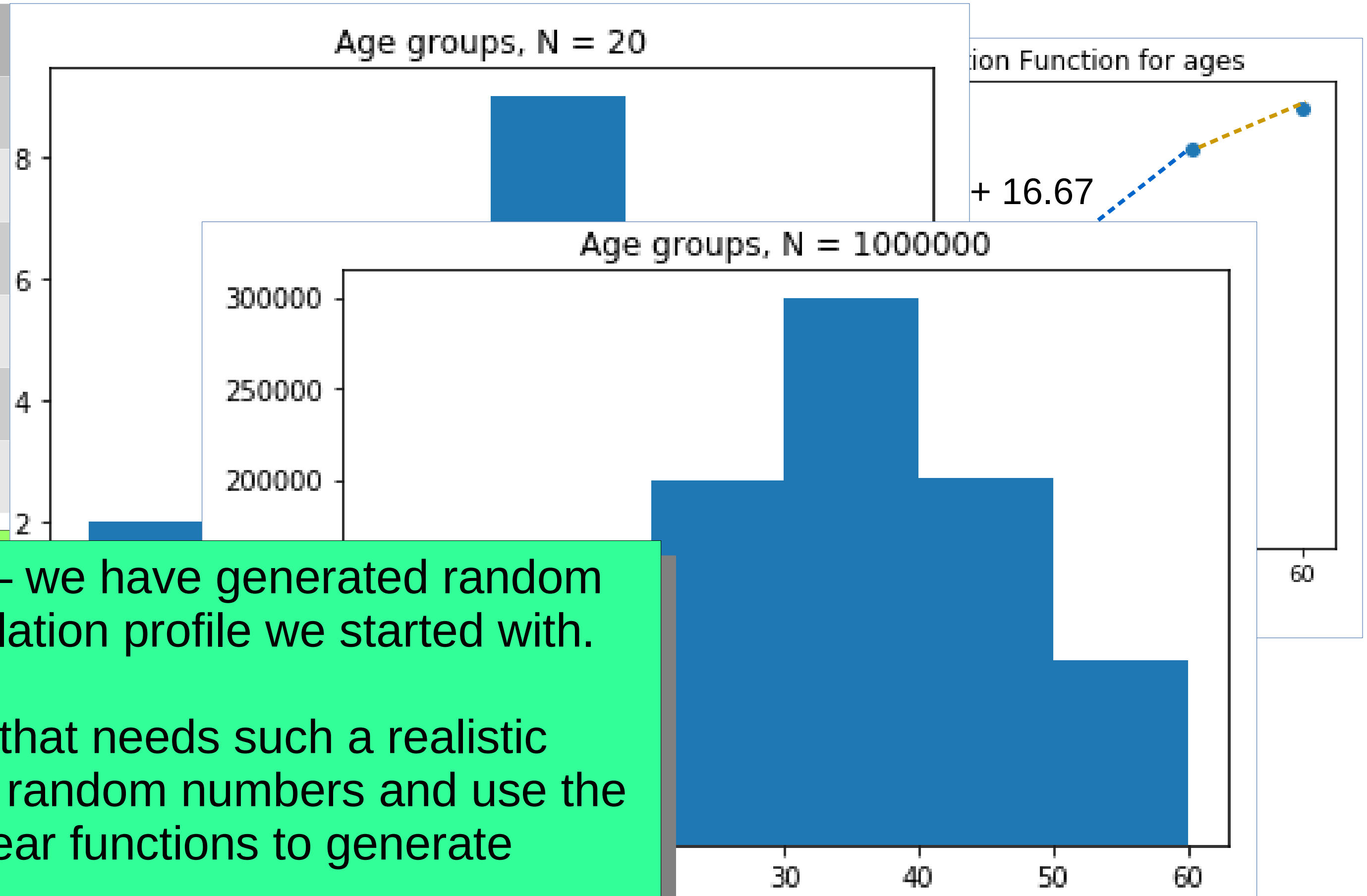
Now, I can generate r
numbers, for each on
the interval it belongs
the appropriate inverse piecew
function to map to an age.



$$
\begin{array}{llllll}
0.1 \leq & p & < 0.2, & x & = & 100p \\
0.2 \leq & p & < 0.4, & x & = & 50p + 10 \\
0.4 \leq & p & < 0.7, & x & = & 33.33p + 16.67 \\
0.7 \leq & p & < 0.9, & x & = & 50p + 5 \\
0.9 \leq & p & < 1.0, & x & = & 100p - 40
\end{array}
$$

A CDF for a similar scenario

Age range	Percent
0-10	10
10-20	10
20-30	20
30-40	30
40-50	20
50-60	10



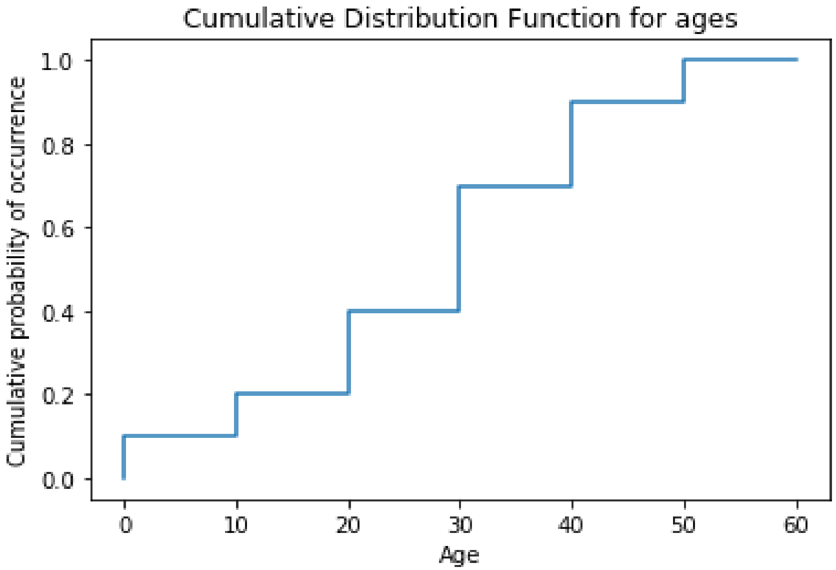
What we have done – we have generated random ages that fit the population profile we started with.

To drive a simulation that needs such a realistic sample, we generate random numbers and use the inverse piecewise linear functions to generate ages.

$$
\begin{array}{lllll}
0.1 \leq & p & < 0.2, & x & = & 100p \\
0.2 \leq & p & < 0.4, & x & = & 50p + 10 \\
0.4 \leq & p & < 0.7, & x & = & 33.33p + 16.67 \\
0.7 \leq & p & < 0.9, & x & = & 50p + 5 \\
0.9 \leq & p & < 1.0, & x & = & 100p - 40
\end{array}
$$

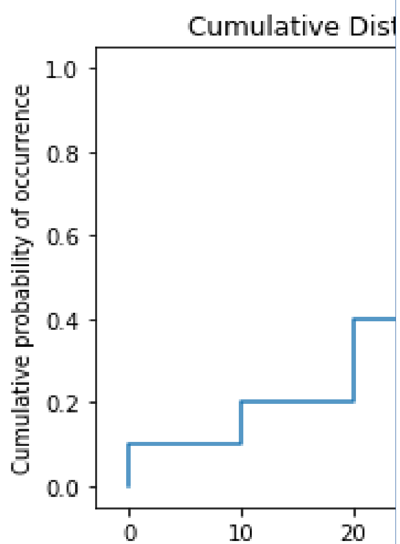
Some code for the previous example

```
# Lets do a CDF
age = np.array([0, 10,20,30,40,50,60])
q = np.array([0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0])

plt.step(age,q)
plt.title('Cumulative Distribution Function for ages')
plt.xlabel('Age')
plt.ylabel('Cumulative probability of occurrence')
plt.show()
```

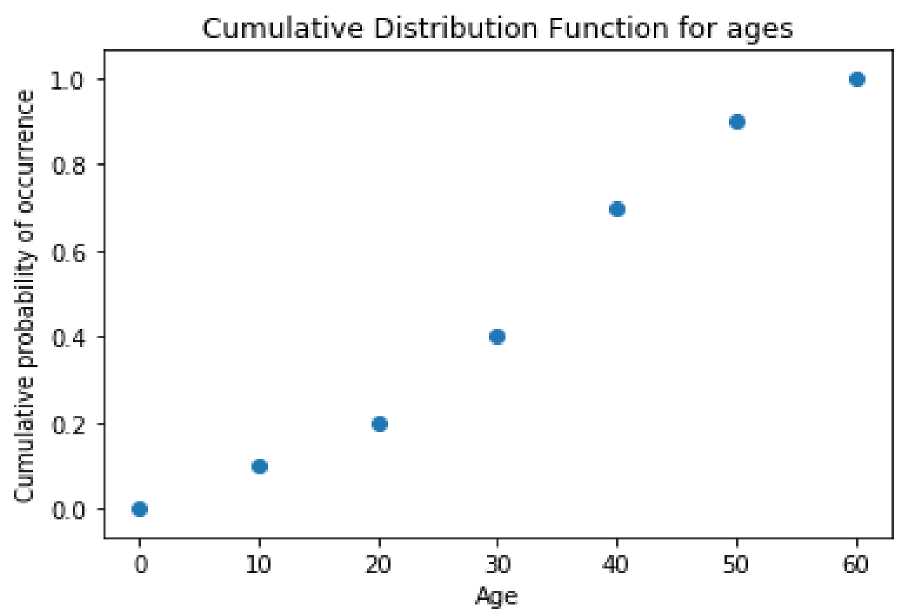


Some code for the previous example

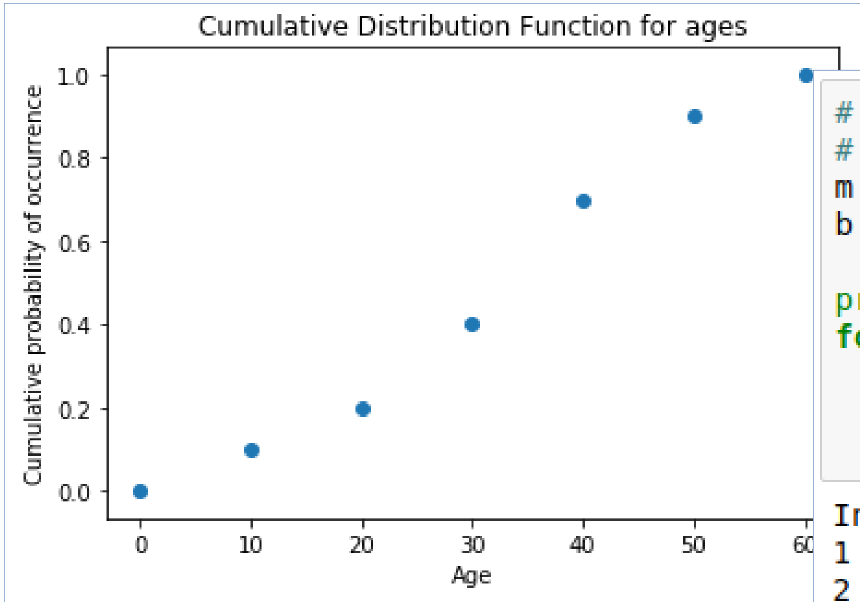
```
# Lets do a CDF
age = np.array([0, 10
q = np.array([0, 0.1,

plt.step(age,q)
plt.title('Cumulative
plt.xlabel('Age')
plt.ylabel('Cumulativ
plt.show()
```




```
age = np.array([0, 10,20,30,40,50,60])
q = np.array([0, 0.1, 0.2, 0.4, 0.7, 0.9, 1.0])

plt.scatter(age,q)
plt.title('Cumulative Distribution Function for ages')
plt.xlabel('Age')
plt.ylabel('Cumulative probability of occurrence')
plt.show()
```



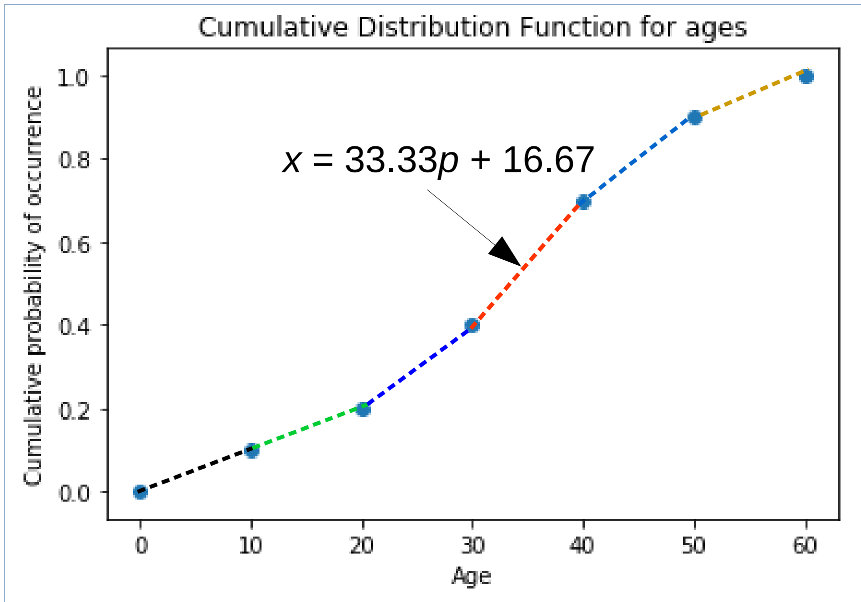
Some code for the previous example




```python
# Find m and b for the piecewise
# linear interpolations q = mx + b
m = np.zeros((age.size-1))
b = np.zeros((age.size-1))

print('Intvl   m     b')
for i in np.arange(age.size-1):
    m[i] = (q[i+1] - q[i]) / (age[i+1] - age[i])
    b[i] = q[i] - m[i]*age[i]
    print('{:1d}     {:4.2f}  {:4.1f}'.format(i+1, m[i], b[i]))
```

```
Intvl   m     b
1     0.01   0.0
2     0.01   0.0
3     0.02  -0.2
4     0.03  -0.5
5     0.02  -0.1
6     0.01   0.4
```

Some code for the previous example




```
Find m and b for the piecewise
linear interpolations q = mx + b
= np.zeros((age.size-1))
= np.zeros((age.size-1))

rint('Intvl   m     b')
or i in np.arange(age.size-1):
    m[i] = (q[i+1] - q[i]) / (age[i+1] - age[i])
    b[i] = q[i] - m[i]*age[i]
    print('{:1d}     {:4.2f}  {:4.1f}'.format(i+1, m[i], b[i]))
```

```
tvl    m      b
      0.01    0.0
      0.01    0.0
      0.02   -0.2
```

```
# My inverse functions will be x = (y-b)/m, or
# x = minv*q + binv
minv = 1.0/m
binv = -b/m
print('Intvl   minv    binv')
for i in np.arange(age.size-1):
    print('{:1d}     {:6.2f}  {:6.2f}'.format(i+1, minv[i], binv[i]))
```

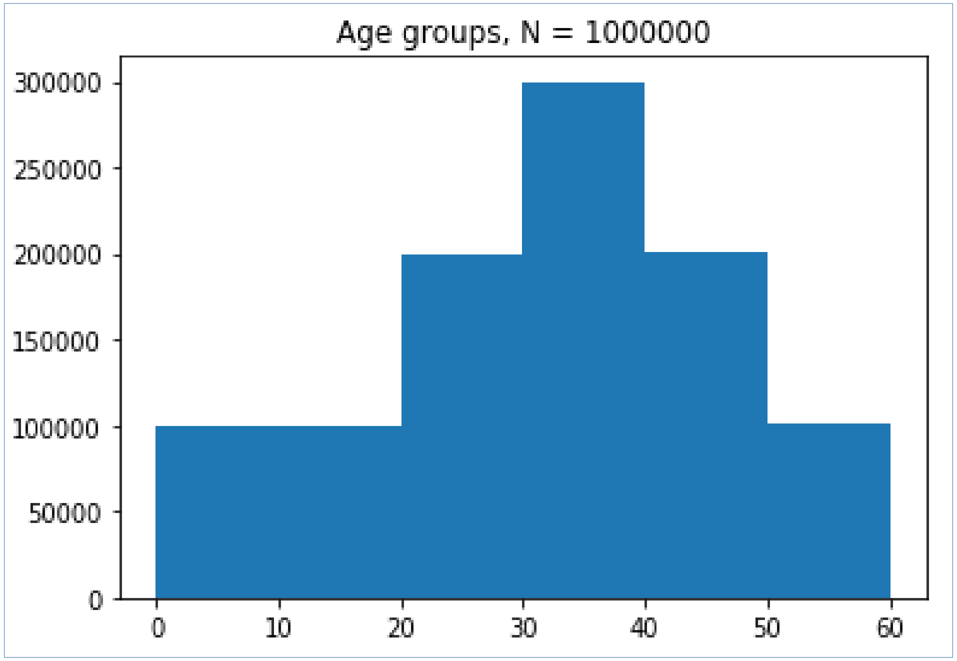
```
Intvl   minv    binv
1     100.00   -0.00
2     100.00   -0.00
3      50.00   10.00
4      33.33   16.67
5      50.00    5.00
6     100.00  -40.00
```

Some code for the previous example

```
# Next, generate a bunch of random numbers.
# For each one, determine its interval, then
# compute an age, and store

N = 1000000
randompvals = np.random.uniform(size=N)
randomages = np.zeros((N))
for i in np.arange(N):
    p = randompvals[i]

    # Select the appropriate piecewise interval
    if 0 <= p < 0.1:
        intvl = 1
    elif 0.1 <= p < 0.2:
        intvl = 2
    elif 0.2 <= p < 0.4:
        intvl = 3
    elif 0.4 <= p < 0.7:
        intvl = 4
    elif 0.7 <= p < 0.9:
        intvl = 5
    else:
        intvl = 6

    # Use the inverse function for the
    # selected interval
    randomages[i] = minv[intvl-1]*p + binv[intvl-1]

plt.hist(randomages, bins=6)
plt.title('Age groups, N = %d' % N)
plt.show()
```

Some code for the previous example

```
# Next, generate a bunch of random numbers.
# For each one, determine its interval, then
# compute an age, and store

N = 1000000
randompvals = np.random.uniform(size=N)
randomages = np.zeros((N))
for i in np.arange(N):
    p = randompvals[i]

    # Select the appropriate piecewise interval
    if 0 <= p < 0.1:
        intvl = 1
    elif 0.1 <= p < 0.2:
        intvl = 2
    elif 0.2 <= p < 0.4:
        intvl = 3
    elif 0.4 <= p < 0.7:
        intvl = 4
    elif 0.7 <= p < 0.9:
        intvl = 5
    else:
        intvl = 6

    # Use the inverse function for the
    # selected interval
    randomages[i] = minv[intvl-1]*p + binv[intvl-1]

plt.hist(randomages, bins=6)
plt.title('Age groups, N = %d' % N)
plt.show()
```



Inventory Model: Gasoline and Consumer Demand

- This is a hard section to follow and, in my opinion, not well presented. I will try to clarify here
- The section goes through a lot of development of a model that helps to determine how often, and how much gasoline should be delivered to a chain of gasoline stations
- The goal is to find parameters that minimize the average daily cost of delivering and storing sufficient gasoline at each station to meet consumer demand
  - Every delivery of gasoline incurs a substantial fixed cost, independent of the amount being delivered. So, we don't want to deliver every day but, on the other hand, we don't want a station to run out of gasoline.
  - A general model is developed

average daily cost = *f*(storage costs, delivery costs, **demand rate**)

Inventory Model: Gasoline and Consumer Demand (continued)

- The text gets really confusing here – it shows an analytic model developed EIGHT chapters ahead, and starts by making assumptions that the daily demand rate (gasoline purchased at a station, per day) is constant.
- The text goes on to say that in some cases, this may be a reasonable assumption, but if we look at the actual data from the past 1000 days, we see a lot of variation

Inventory Model: Gasoline and Consumer Demand (continued)

- The text gets really confusing here – it shows an analytic model de… HT chapters ahead, and starts by mak… ons that the <u>daily demand</u> … tation, per day) is c…

$$
\begin{aligned}
T^* &= \sqrt{\frac{2d}{sr}} \\
Q^* &= rT^*
\end{aligned}
$$

where

T^* = optimal time between deliveries in days
Q^* = optimal delivery quantity of gasoline in gallons
***r* = demand rate in gallons per day**
d = delivery cost in dollars per delivery
s = storage cost per gallon per day

- The text … ses, this may be a… we look at the actual data from the past 1000 days, we see a lot of variation

Inventory Model: Gasoline and Consumer Demand (continued)

Table 5.6 History of demand at a particular gasoline station

Number of gallons demanded	Number of occurrences (in days)
1000–1099	10
1100–1199	20
1200–1299	50
1300–1399	120
1400–1499	200
1500–1599	270
1600–1699	180
1700–1799	80
1800–1899	40
1900–1999	30
	1000

g here – it shows an

$\frac{2d}{sr}$

HT chapters ahead,

T^* ons that the daily

tation, per

eries in days

gasoline in gallons

r day

elivery

lay

ses, this

we look at

t 1000 days, we see a

lot of variation

Inventory Model: Gasoline and Consumer Demand (continued)

Table 5.6 History of demand at a particular gasoline station

Number of gallons demanded	Number of occurrences (in days)
1000–1099	10

g here – it shows an
HT chapters ahead,
ons that the daily
tation, per

ons

ses, this
we look at
ys, we see a
lo

Table 5.7 Probability of the occurrence of each demand level

Number of gallons demanded	Probability of occurrence
1000–1099	0.01
1100–1199	0.02
1200–1299	0.05
1300–1399	0.12
1400–1499	0.20
1500–1599	0.27
1600–1699	0.18
1700–1799	0.08
1800–1899	0.04
1900–1999	0.03
	1.00

Inventory Model: Gasoline and Consumer Demand (continued)

Table 5.6 History of demand at a particular gasoline station

Number of gallons demanded	Number of occurrences (in days)
1000–1099	10

g here – it shows an

$\frac{2d}{sr}$

HT chapters ahead,

$r*$ ons that the daily

tation, per

ons

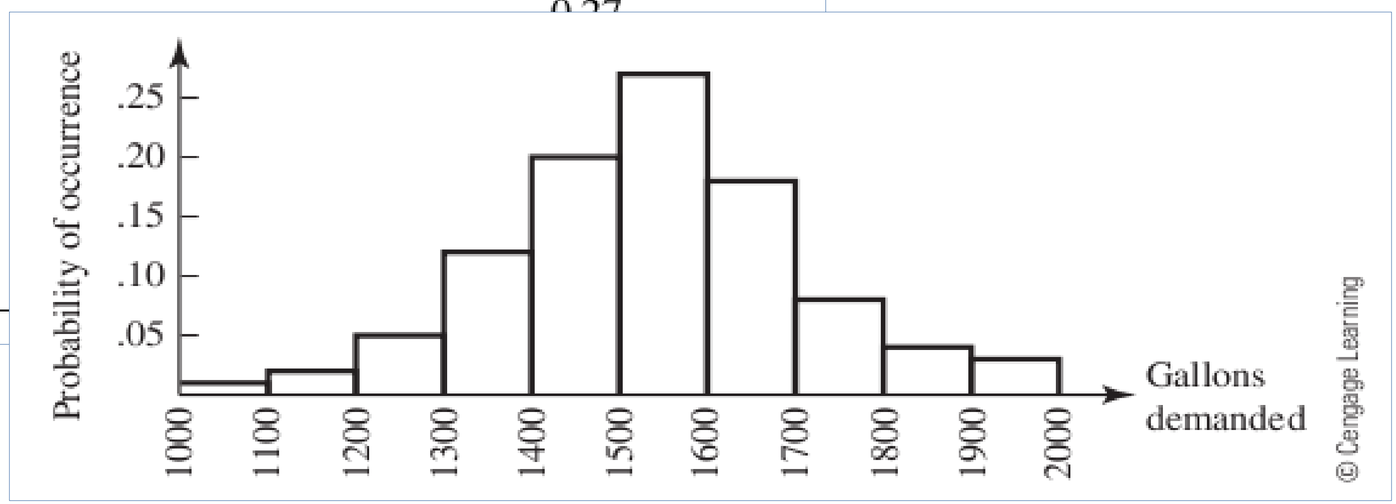
ses, this

we look at

lo

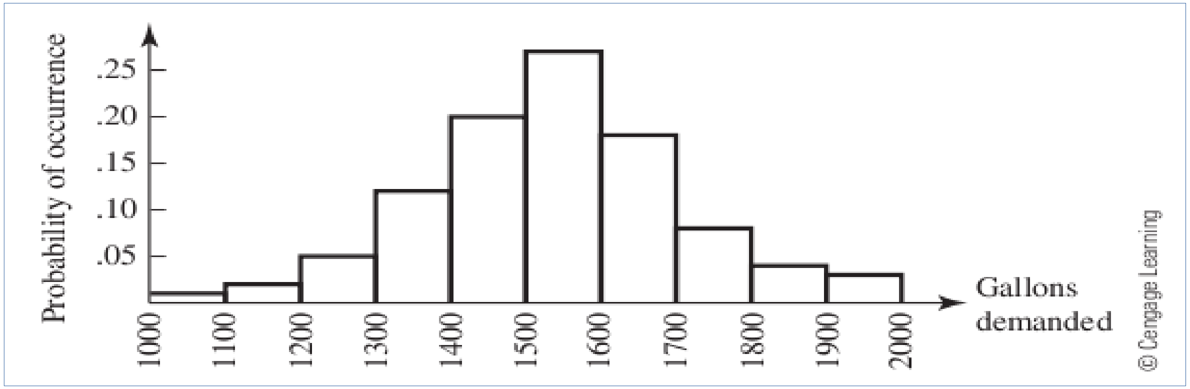
Table 5.7 Probability of the occurrence of each demand level

Number of gallons demanded	Probability of occurrence
1000–1099	0.01
1100–1199	0.02
1200–1299	0.05
1300–1399	0.12
1400–1499	0.20
1500–1599	0.27
1600–1699	
1700–1799	
1800–1899	
1900–1999	



Inventory Model: Gasoline and Consumer Demand (continued)

- The text then goes on to say that if we're happy with the assumption of constant demand rate, we might estimate 1550 gallons per day, based on the histogram, and feed that to the analytic model.



- But, of course, we're not happy with that assumption, and we want to simulate a submodel that expresses the varying demands suggested by the histogram

Outline of inventory algorithm

- Big picture

```
Iterate through inventory/delivery cycles
|
|    - Delivery of gas to station
|    - Add delivery cost to total running costs for station
|
|    Iterate through days in an inventory cycle
|    |
|    |    - Estimate a demand rate for the day
|    |    - Update station inventory and storage costs
|    |
|    End iteration through days in an inventory cycle
|
End iteration through inventory/delivery cycles

Report on daily average station cost as a result of above
```

Outline of inventory algorithm

- Big picture

```
Iterate through i
|
|    - Delivery of
|    - Add deliver
|
|    Iterate throu
|    |
|    |    - Estimate a demand rate for the day
|    |    - Update station inventory and storage costs
|    |
|    End iteration through days in an inventory cycle
|
End iteration through inventory/delivery cycles

Report on daily average station cost as a result of above
```

What would we do with this kind of model?

- Primary goal is to be able to experiment with parameters for reducing a station's daily average cost (to maximise profit)
- Experiment with things like delivery frequency and amount, different costs for local storage, different customer demand rates based on different factors (e.g. holidays, weekdays), ...

Outline of inventory algorithm

Inputs

```
    Q: delivery quantity        T: time between deliveries
    d: cost of delivery         s: cost of storage
    N: total simulation length
```

Initialise

```
    - simulation days remaining, K, set to total simulation length (in days)
    - current inventory, I, set to zero
    - total running cost, C, set to zero

for each inventory cycle
    - add delivery amount (gallons), Q, to inventory, I
    - add cost of delivery, d, to total running cost, C

    for each day in inventory cycle
        - compute gas demand for day, q_i
        - subtract daily demand from current inventory, I
        - compute per gallon storage cost for remaining inventory
        - update total running cost, C
    end for
end for

Compute average daily cost for the station
```

Outline of inventory algorithm

```
Inputs

    Q: delivery quantity          T: time between deliveries
    d: cost of delivery           s: cost of storage
    N: total simulation length

Initialise

    - simulation days remaining, K, set to total simulation length (in days)
    - current inventory, I, set to zero
    - total running cost, C, set to zero

for each inventory cycle
    - add delivery amount (gallons), Q, to inventory, I
    - add cost of delivery, d, to total running cost, C

    for each day in inventory cycle
        - compute gas demand for day, q_i
        - subtract daily demand from current inventory, I
        - compute per gallon storage cost for remaining inventory
        - update total running cost, C
    end for
end for

Compute average daily cost for the station
```

T days per inventory cycle

For our purposes, <u>this is the emphasis of the topic</u> - we will consider different ways to estimate daily demand rate

This is our output – we can run simulation over and over with different parameters in an attempt to find an agreeable combination that minimises the average daily cost for the station (helping to maximise profit)

Outline of inventory algorithm

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, q_i

- Constant value of 1250.0
- Random uniform value between 1000.0 and 2000.0
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Input

Initi

ength (in days)

for e

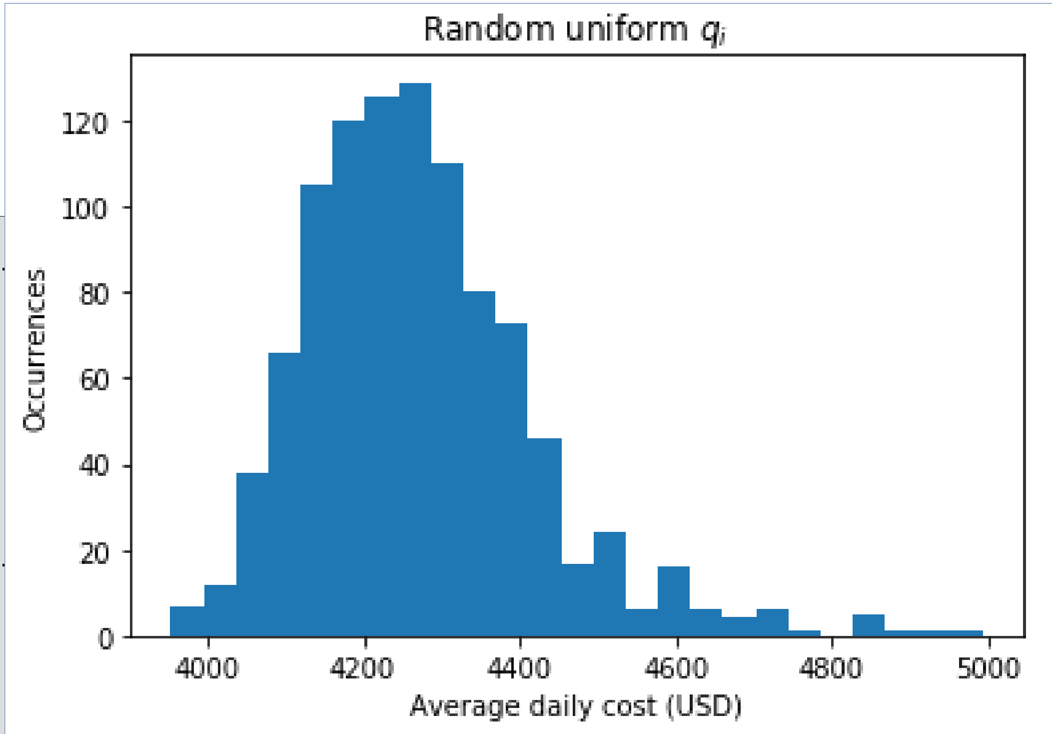
s the emphasis of
der different ways to
ate

tory

can run simulation
erent parameters in
greeable combination
that minimises the average daily cost for the station (helping to maximise profit)

end f

Compute average daily cost for the station



ory algorithm

four different

nt:

```
ons)
 (days)
ays)
per delivery)
er gallon per day)
.ength (in days)
```

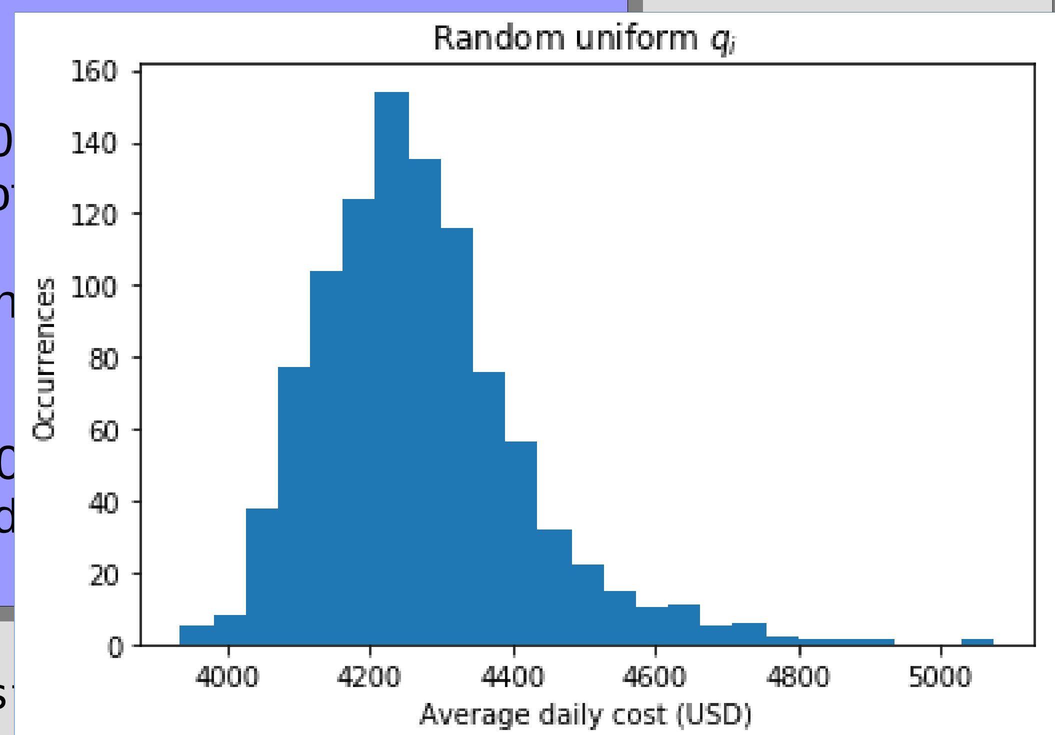
nate of daily demand,

***q**$_i$*

- Constant value of 1250.0
- Random uniform value between 10
- Random normal value with mean o
  deviation of 200.0
- Random value derived from deman
  textbook)

For each scenario, we will perform 10
histogram of the computed average d
simulation

```
for e
 -
 -
 f
 e
end f

Compute average daily cost for the s
```



Outline of inventory algorithm

Let's consider running this simulation under four different scenarios

All three will have the following

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, q_i

- Constant value of 1250.0
- Random uniform value between 1000.0 and 2000.0
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

In each case, I will first test that my q_i submodel looks reasonable, and then run the simulations with it

Input

Initi

ength (in days)

for e

end f

```
Compute average daily cost for the station
```

s the emphasis of der different ways to ate

tory

can run simulation erent parameters in greeable combination that minimises the average daily cost for the station (helping to maximise profit)

Constant q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, $\boldsymbol{q_i}$

- **Constant value of 1250.0**
- Random uniform value between 1000.0 and 2000.0
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Constant q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per del
s = 0.98    # Storage cost (dollars per gall
```

The one varying parameter will be the estimate of q_i

- **Constant value of 1250.0**
- Random uniform value between 1000.0 and 2000.0
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

```
def qi():
    """
    Daily demand in gallons
    """
    return 1250

# Test the qi distribution
N = 500
qi_vec = np.zeros((N))
for i in np.arange(N):
    qi_vec[i] = qi()
plt.hist(qi_vec)
plt.show()
```

Constant q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per del
s = 0.98    # Storage cost (dollars per gall
```

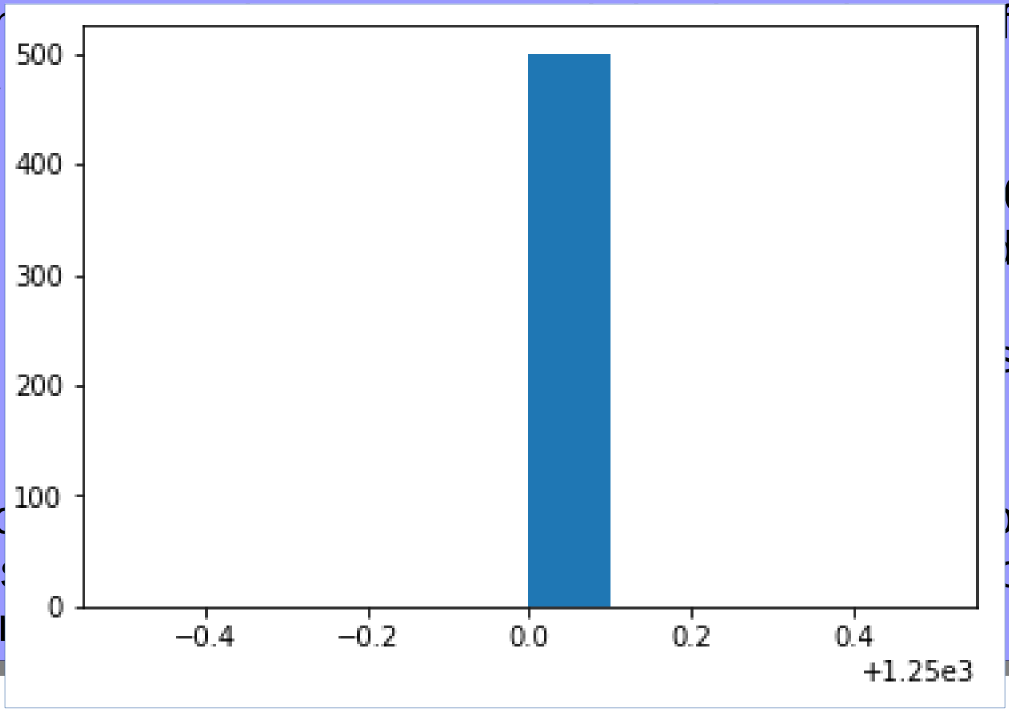
```
def qi():
    """
    Daily demand in gallons
    """
    return 1250

# Test the qi distribution
N = 500
qi_vec = np.zeros((N))
for i in np.arange(N):
    qi_vec[i] = qi()
plt.hist(qi_vec)
plt.show()
```

Th f

q_i

-
- 00.0
- standard
- s illustrated in

Fo ons and create a
hi om each
si



Constant q_i

Let's consi
scenarios

All three wi

```
Q = 10(
T = 7
N = 60
d = 50(
s = 0.(
```

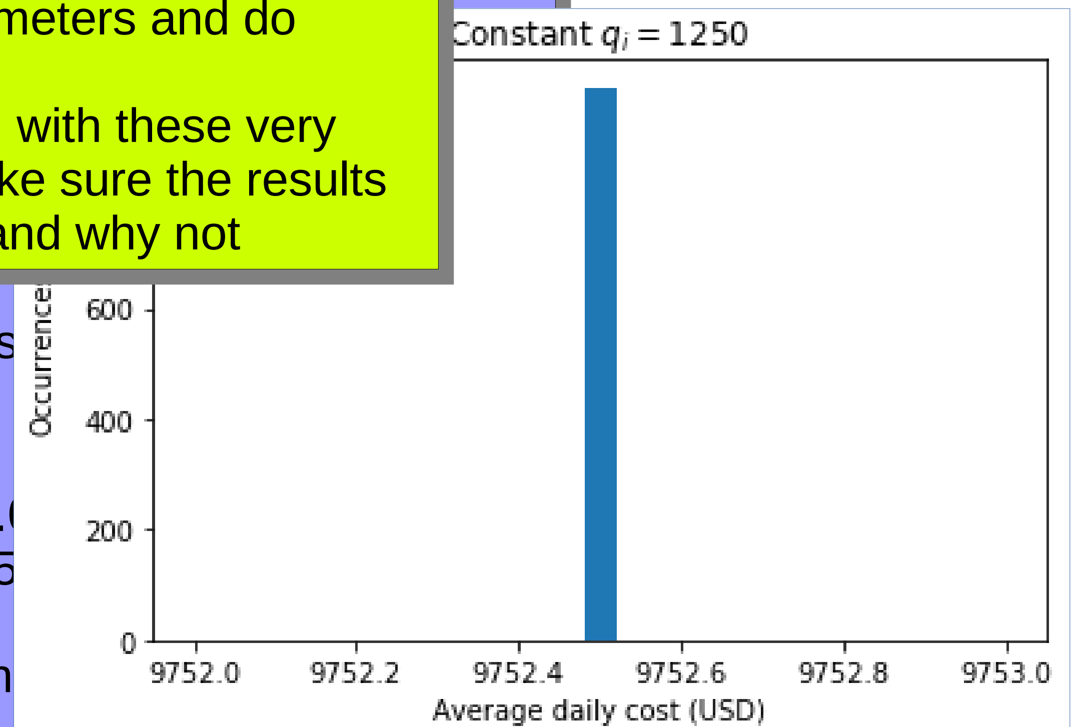
The one varying parameter will be the es
$\boldsymbol{q_i}$

- **Constant value of 1250.0**
- Random uniform value between 1000.(
- Random normal value with mean of 15
  deviation of 200.0
- Random value derived from demand h
  textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Note that this could provide an approach for testing that my full model works correctly.

- Everything is constant, so I expect the same answer every time
- I could set up “easy” parameters and do several iterations by hand
- I could then run the model with these very same parameters and make sure the results agree and, if not, understand why not



Random uniform q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, $\boldsymbol{q_i}$

- Constant value of 1250.0
- **Random uniform value between 1000.0 and 2000.0**
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Random uniform q_i

Let's consider running this simulation u
scenarios

All three will have the following held co

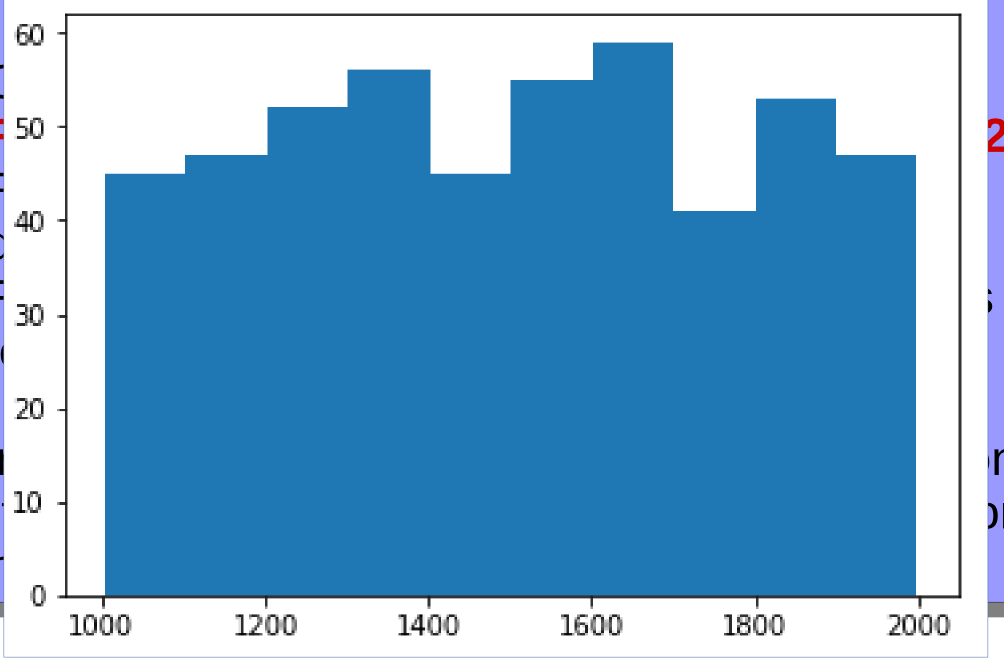
```
Q = 10000   # Delivery quantity (
T = 7       # Time between delive
N = 60      # Length of simulatio
d = 500.0   # Delivery cost (doll
s = 0.98    # Storage cost (dolla
```

```
def qi():
    """
    Daily demand in gallons
    """
    return np.random.uniform(low=1000, high=2000)

# Test the qi distribution
N = 500
qi_vec = np.zeros((N))
for i in np.arange(N):
    qi_vec[i] = qi()
plt.hist(qi_vec)
plt.show()
```

The one varying parameter will be the estimate of daily demand, ***q_i***

- C
- **F 2000.0**
- F standard c
- F illustrated in t

For ns and create a his om each sim



Random uniform q_i

Let's consider running this simulation under four different scenarios

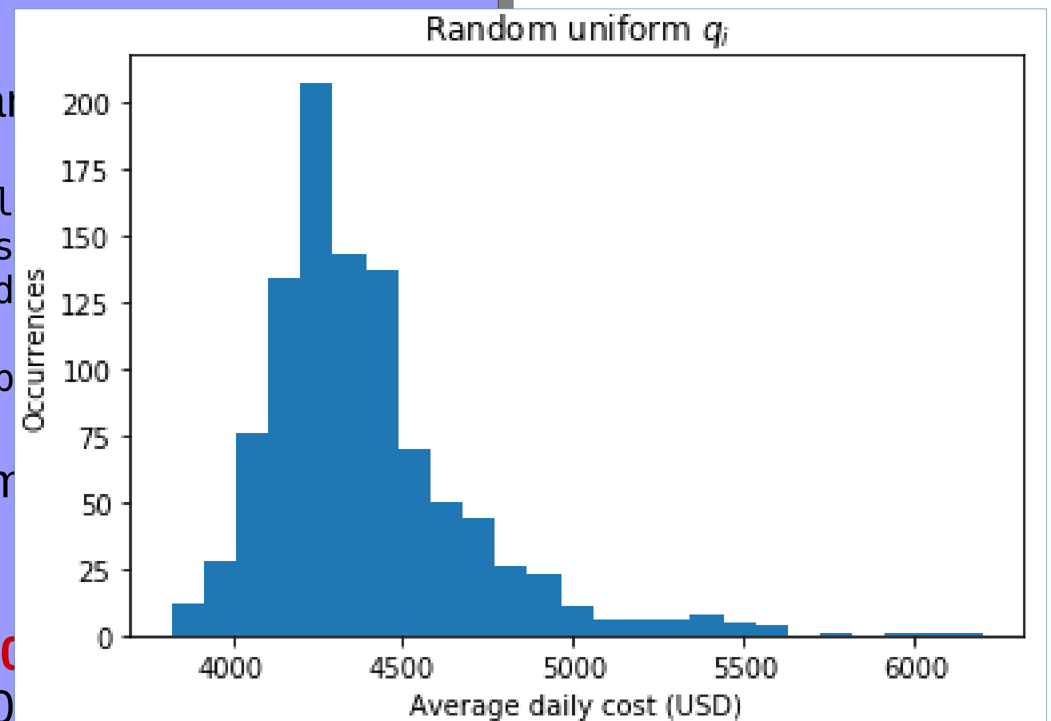
All three will have the following held constar

```
Q = 10000   # Delivery quantity (gall
T = 7       # Time between deliveries
N = 60      # Length of simulation (d
d = 500.0   # Delivery cost (dollars 
s = 0.98    # Storage cost (dollars p
```

The one varying parameter will be the estim
***q**$_i$*

- Constant value of 1250.0
- **Random uniform value between 1000.0**
- Random normal value with mean of 1500 deviation of 200.0
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation



Random normal q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, $\boldsymbol{q_i}$

- Constant value of 1250.0
- Random uniform value between 1000.0 and 2000.0
- **Random normal value with mean of 1500.0 and standard deviation of 200.0**
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Random normal q_i

Let's consider running this simulation
scenarios

All three will have the following held

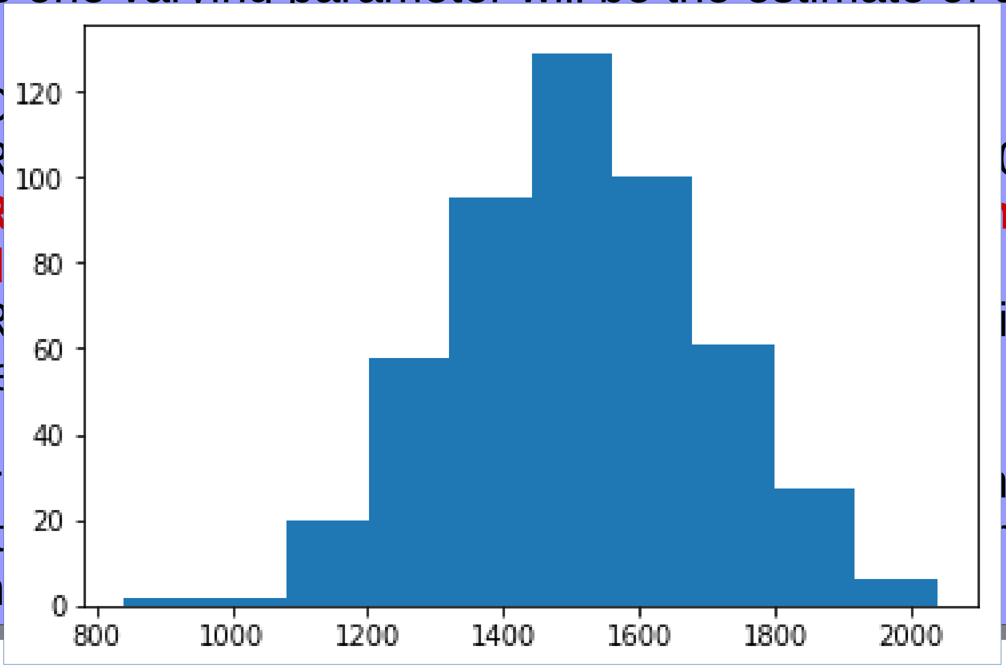
```
Q = 10000   # Delivery quantity
T = 7       # Time between deli
N = 60      # Length of simula
d = 500.0   # Delivery cost (d
s = 0.98    # Storage cost (do
```

```
def qi():
    """
    Daily demand in gallons
    """
    return np.random.normal(loc=1500.0, scale=200.0)

# Test the qi distribution
N = 500
qi_vec = np.zeros((N))
for i in np.arange(N):
    qi_vec[i] = qi()
plt.hist(qi_vec)
plt.show()
```

The one varying parameter will be the estimate of daily demand,
q_i

- C
- R 0.0
- **R nd standard d**
- R illustrated in te

For s and create a
hist m each
sim



Random normal q_i

Let's consider running this simulation under four different scenarios

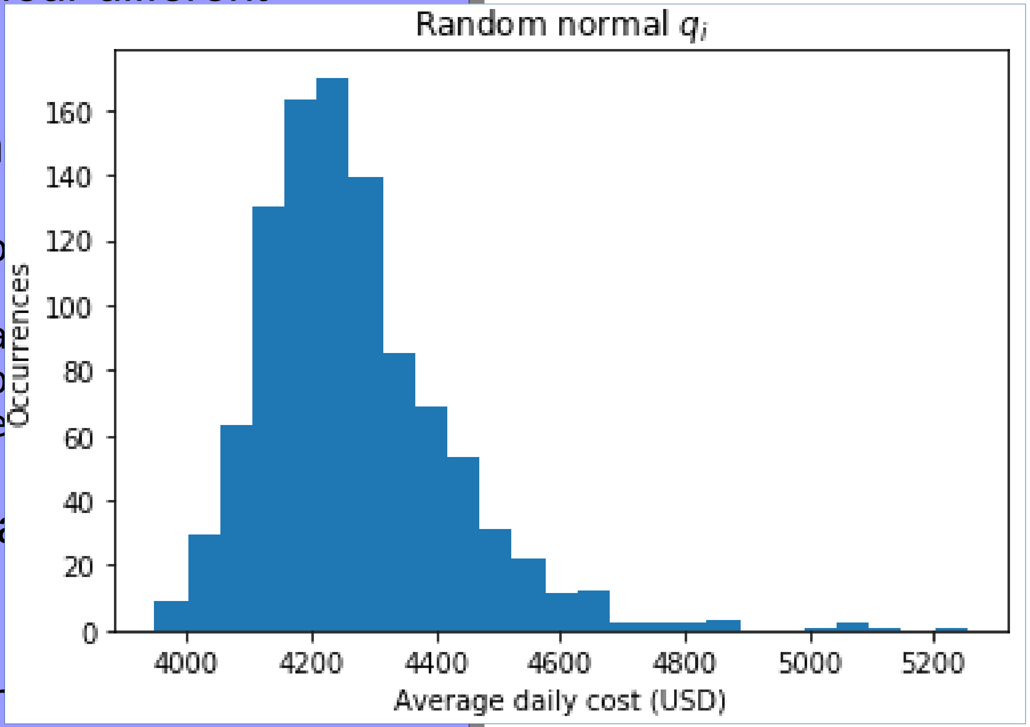
All three will have the following held constan

```
Q = 10000   # Delivery quantity (gallo
T = 7       # Time between deliveries
N = 60      # Length of simulation (da
d = 500.0   # Delivery cost (dollars p
s = 0.98    # Storage cost (dollars pe
```

The one varying parameter will be the estima
$\boldsymbol{q_i}$

- Constant value of 1250.0
- Random uniform value between 1000.0 ar
- **Random normal value with mean of 1500.0 and standard deviation of 200.0**
- Random value derived from demand history (as illustrated in textbook)

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation



Random from demand history q_i

Let's consider running this simulation under four different scenarios

All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estimate of daily demand, $\boldsymbol{q_i}$

- Constant value of 1250.0
- Random uniform value between 1000.0 and 2000.0
- Random normal value with mean of 1500.0 and standard deviation of 200.0
- **Random value derived from demand history (as illustrated in textbook)**

For each scenario, we will perform 1000 simulations and create a histogram of the computed average daily costs from each simulation

Random from demand history q_i

- In the following slides we show the textbook development
  - Actual data – collected history of customer demand (in intervals of 100) over 1000 days
  - Calculate probability of each demand level
  - Build a cumulative histogram of daily demand submodel
  - Represent the cumulative histogram with linear splines
  - Compute inverses of linear splines that compute a daily demand as a function of a uniform random number between 0 and 1.

Random from demand history q_i

Table 5.6 History of demand at a particular gasoline station

Number of gallons demanded	Number of occurrences (in days)
1000–1099	10
1100–1199	20
1200–1299	50
1300–1399	120
1400–1499	200
1500–1599	270
1600–1699	180
1700–1799	80
1800–1899	40
1900–1999	30
	1000

the textbook

customer demand (in

and level

aily demand submodel

am with linear splines

– Compute inverses of linear splines that compute a daily demand as a function of a uniform random number between 0 and 1.

Random from demand history q_i

Table 5.6 History of demand at a particular gasoline station

Nu

© C

Table 5.7 Probability of the occurrence of each demand level

Number of gallons demanded	Probability of occurrence
1000–1099	0.01
1100–1199	0.02
1200–1299	0.05
1300–1399	0.12
1400–1499	0.20
1500–1599	0.27
1600–1699	0.18
1700–1799	0.08
1800–1899	0.04
1900–1999	0.03
	1.00

textbook

omer demand (in

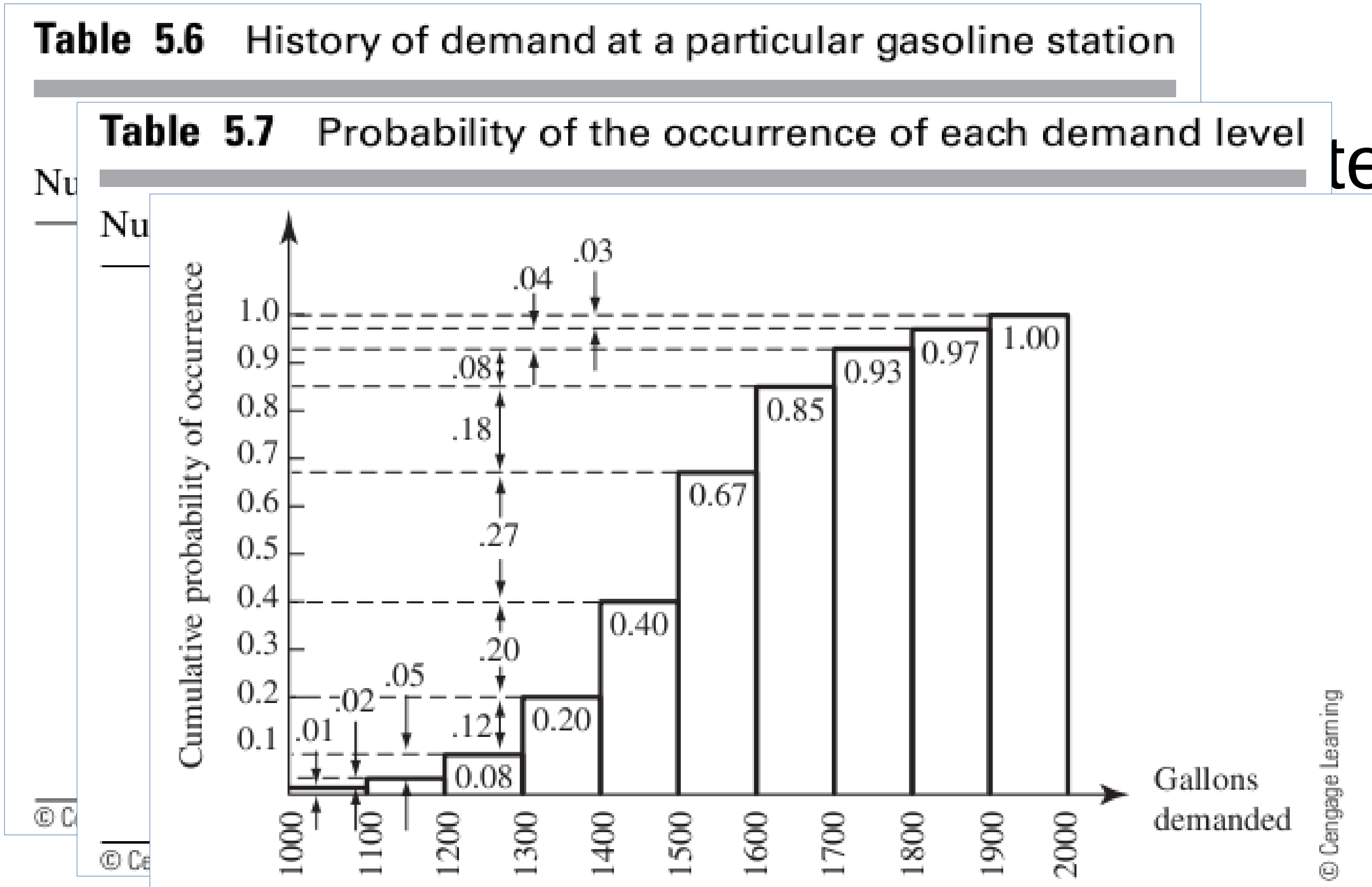
level

demand submodel

ith linear splines

t compute a daily demand as a function of a uniform random number between 0 and 1.

Random from demand history q_i



textbook

ner demand (in

vel

mand submodel

h linear splines

compute a daily demand as a function of a uniform random number between 0 and 1.

Random from demand history q_i

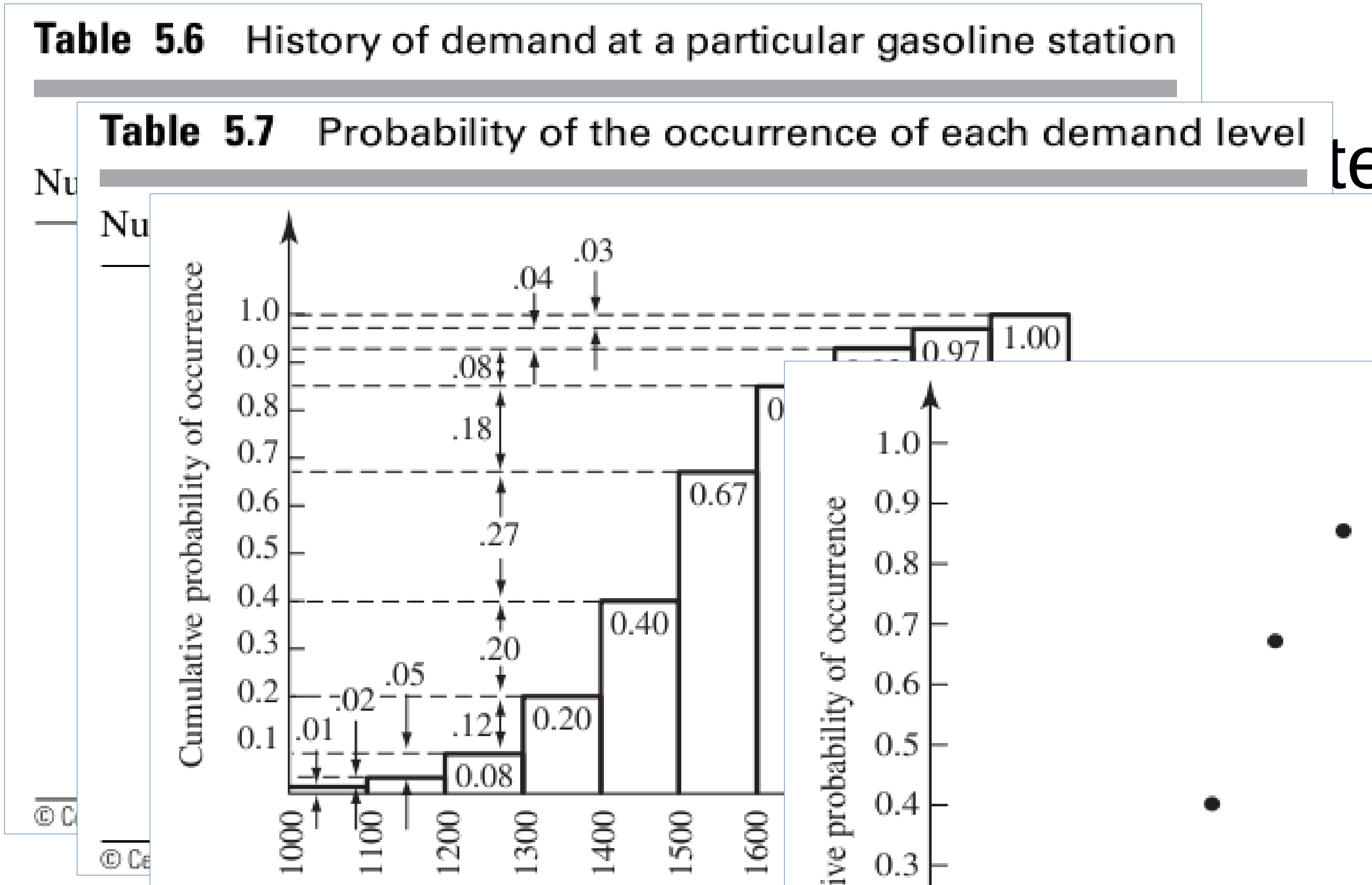


Table 5.6 History of demand at a particular gasoline station

Nu

Table 5.7 Probability of the occurrence of each demand level

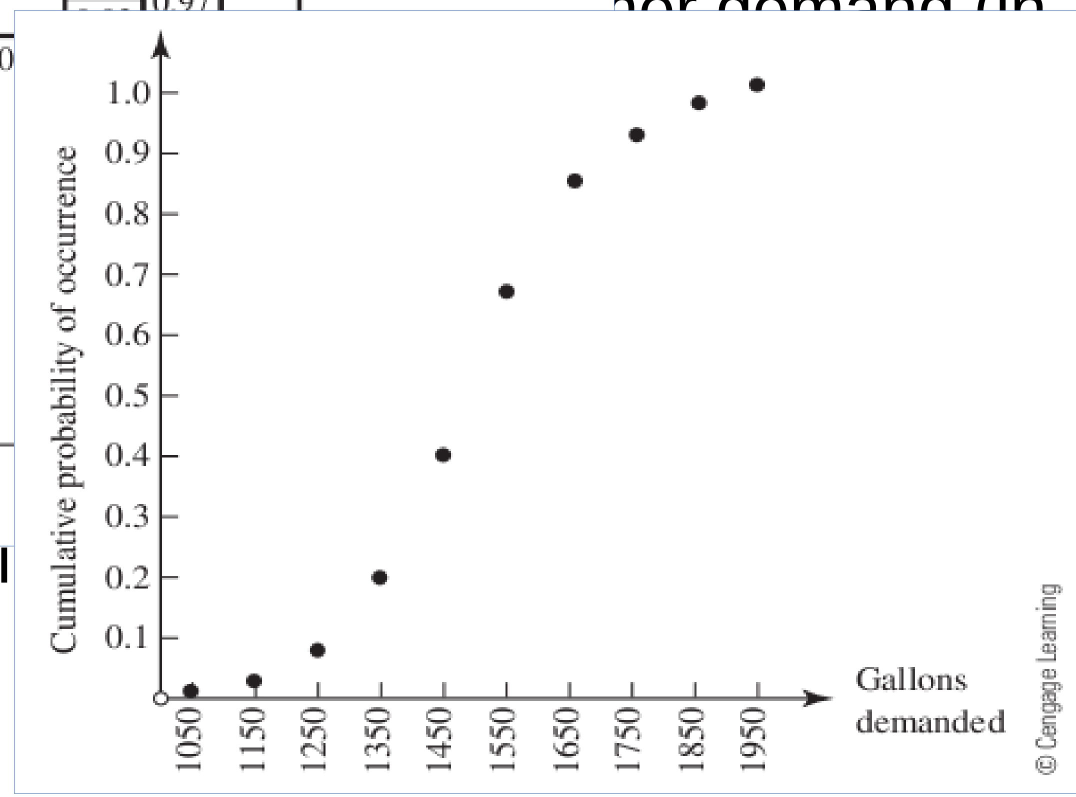
Nu

textbook

ner demand (in

demand as a functi

between 0 and 1.



Random from demand history q_i

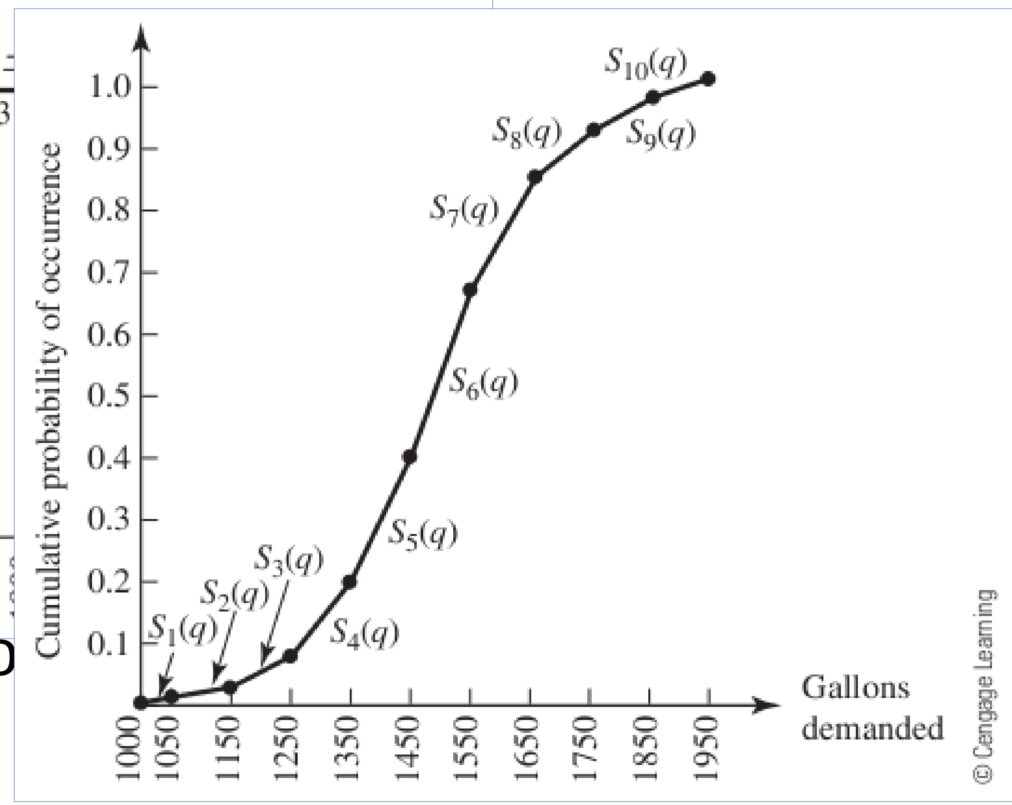
Table 5.6 History of demand at a particular gasoline station

Table 5.7 Probability of the occurrence of each demand level

textbook

Table 5.10 Linear splines for the empirical demand submodel

Demand interval	Linear spline
$1000 \le q < 1050$	$S_1(q) = 0.0002q - 0.2$
$1050 \le q < 1150$	$S_2(q) = 0.0002q - 0.2$
$1150 \le q < 1250$	$S_3(q) = 0.0005q - 0.545$
$1250 \le q < 1350$	$S_4(q) = 0.0012q - 1.42$
$1350 \le q < 1450$	$S_5(q) = 0.002q - 2.5$
$1450 \le q < 1550$	$S_6(q) = 0.0027q - 3.515$
$1550 \le q < 1650$	$S_7(q) = 0.0018q - 2.12$
$1650 \le q < 1750$	$S_8(q) = 0.0008q - 0.47$
$1750 \le q < 1850$	$S_9(q) = 0.0004q + 0.23$
$1850 \le q \le 2000$	$S_{10}(q) = 0.0002q + 0.6$



Random from demand history q_i

Table 5.6 History of demand at a particular gasoline station

Table 5.7 Probability of the occurrence of each demand level

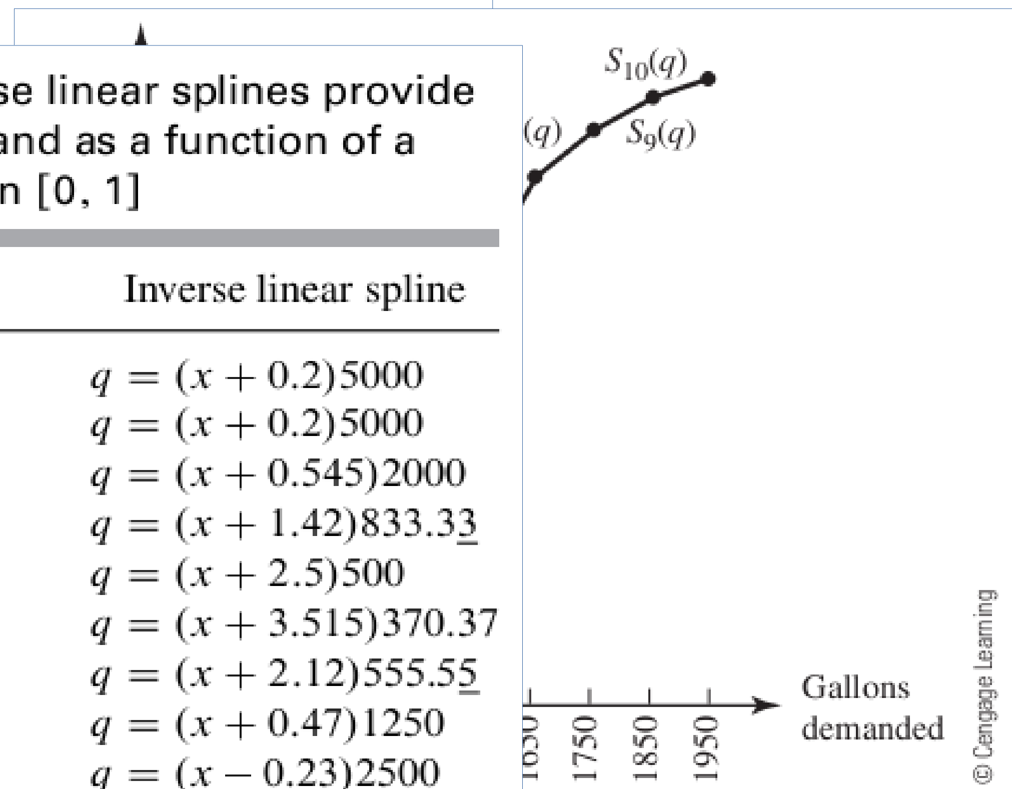
textbook

Table 5.10 Linear splines fo… demand submodel

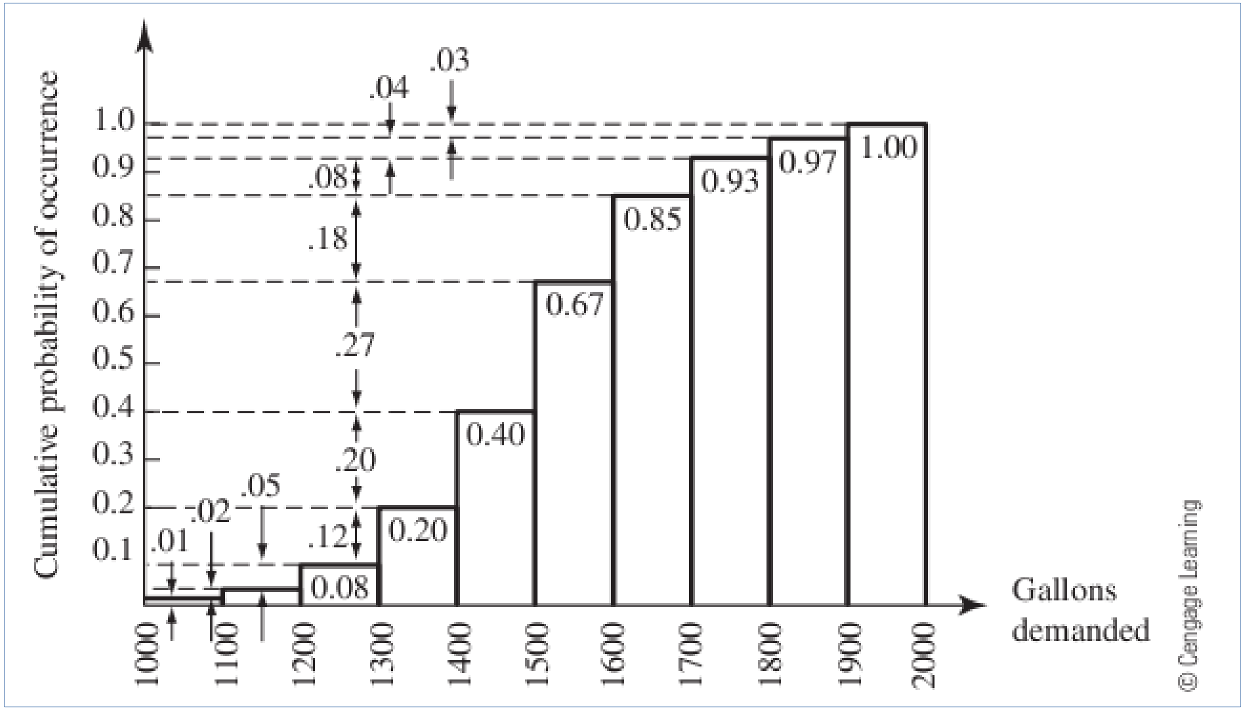
Demand interval	
$1000 \le q < 1050$	$S_1(q)$
$1050 \le q < 1150$	$S_2(q)$
$1150 \le q < 1250$	$S_3(q)$
$1250 \le q < 1350$	$S_4(q)$
$1350 \le q < 1450$	$S_5(q)$
$1450 \le q < 1550$	$S_6(q)$
$1550 \le q < 1650$	$S_7(q)$
$1650 \le q < 1750$	$S_8(q)$
$1750 \le q < 1850$	$S_9(q)$
$1850 \le q \le 2000$	$S_{10}(q)$

Table 5.11 Inverse linear splines provide for the daily demand as a function of a random number in [0, 1]

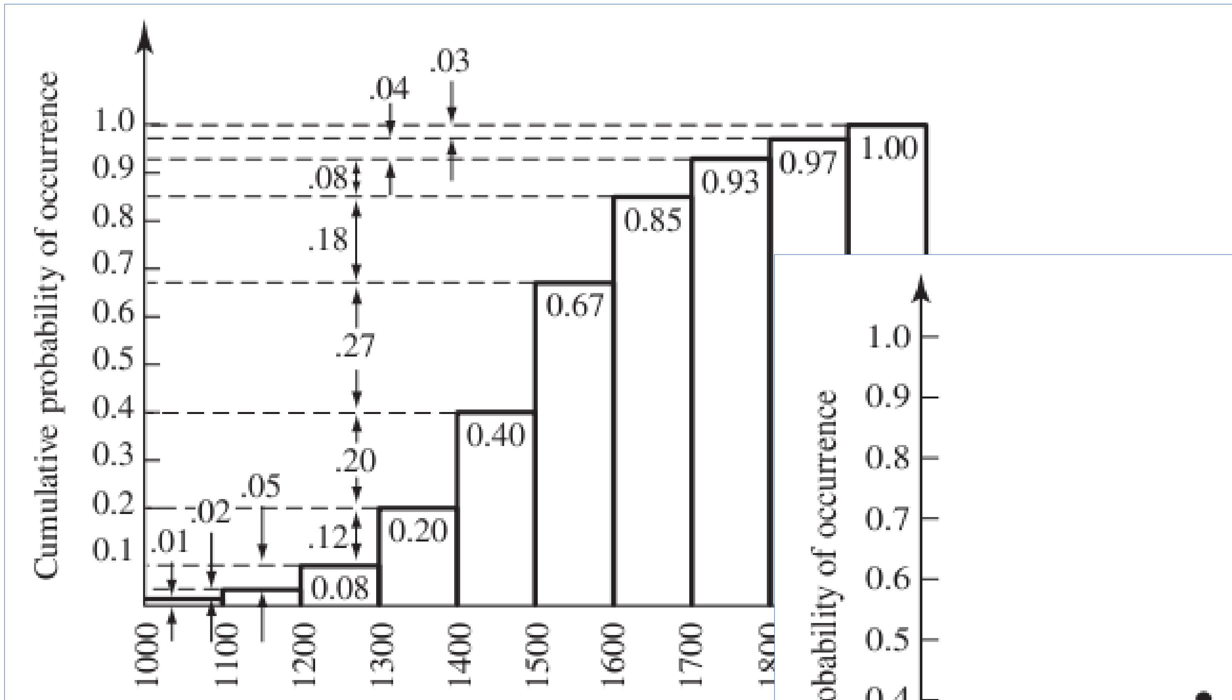
Random number	Inverse linear spline
$0 \le x < 0.01$	$q = (x + 0.2)5000$
$0.01 \le x < 0.03$	$q = (x + 0.2)5000$
$0.03 \le x < 0.08$	$q = (x + 0.545)2000$
$0.08 \le x < 0.20$	$q = (x + 1.42)833.3\underline{3}$
$0.20 \le x < 0.40$	$q = (x + 2.5)500$
$0.40 \le x < 0.67$	$q = (x + 3.515)370.37$
$0.67 \le x < 0.85$	$q = (x + 2.12)555.5\underline{5}$
$0.85 \le x < 0.93$	$q = (x + 0.47)1250$
$0.93 \le x < 0.97$	$q = (x - 0.23)2500$
$0.97 \le x \le 1.00$	$q = (x - 0.6)5000$

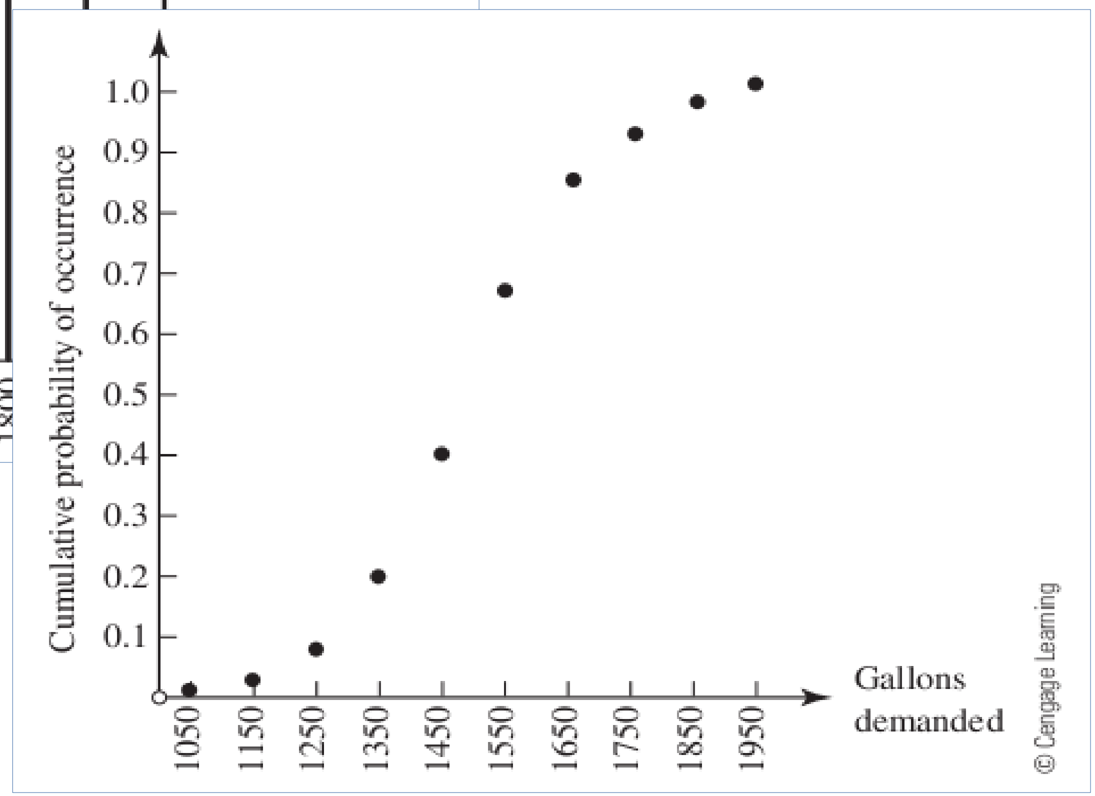


Could have used cubic splines instead of linear...



Could have used cubic splines instead of linear...

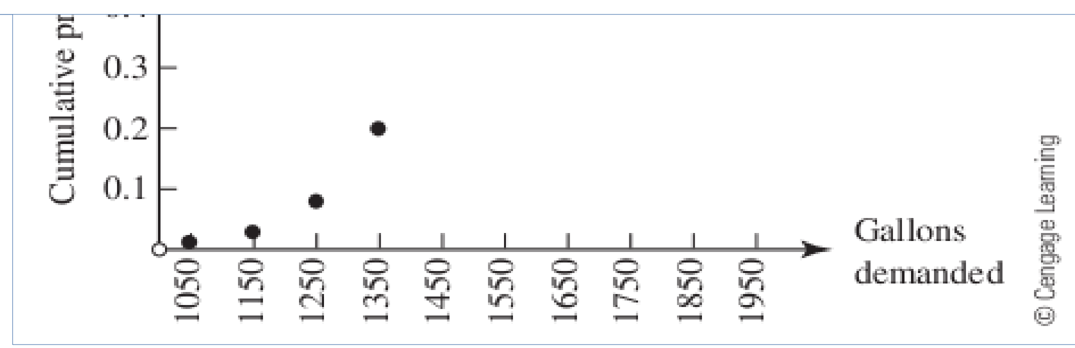




Could have used cubic splines instead of linear...

Table 5.12 An empirical cubic spline model for demand

Random number	Cubic spline
$0 \le x < 0.01$	$S_1(x) = 1000 + 4924.92x + 750788.75x^3$
$0.01 \le x < 0.03$	$S_2(x) = 1050 + 5150.18(x - 0.01) + 22523.66(x - 0.01)^2 - 1501630.8(x - 0.01)^3$
$0.03 \le x < 0.08$	$S_3(x) = 1150 + 4249.17(x - 0.03) - 67574.14(x - 0.03)^2 + 451815.88(x - 0.03)^3$
$0.08 \le x < 0.20$	$S_4(x) = 1250 + 880.37(x - 0.08) + 198.24(x - 0.08)^2 - 4918.74(x - 0.08)^3$
$0.20 \le x < 0.40$	$S_5(x) = 1350 + 715.46(x - 0.20) - 1572.51(x - 0.20)^2 + 2475.98(x - 0.20)^3$
$0.40 \le x < 0.67$	$S_6(x) = 1450 + 383.58(x - 0.40) - 86.92(x - 0.40)^2 + 140.80(x - 0.40)^3$
$0.67 \le x < 0.85$	$S_7(x) = 1550 + 367.43(x - 0.67) + 27.12(x - 0.67)^2 + 5655.69(x - 0.67)^3$
$0.85 \le x < 0.93$	$S_8(x) = 1650 + 926.92(x - 0.85) + 3081.19(x - 0.85)^2 + 11965.43(x - 0.85)^3$
$0.93 \le x < 0.97$	$S_9(x) = 1750 + 1649.66(x - 0.93) + 5952.90(x - 0.93)^2 + 382645.25(x - 0.93)^3$
$0.97 \le x \le 1.00$	$S_{10}(x) = 1850 + 3962.58(x - 0.97) + 51870.29(x - 0.97)^2 - 576334.88(x - 0.97)^3$



But, we will proceed with the linear splines...

Random from demand history q_i

Let's consider running this simulation u
scenarios

All three will have the following held co

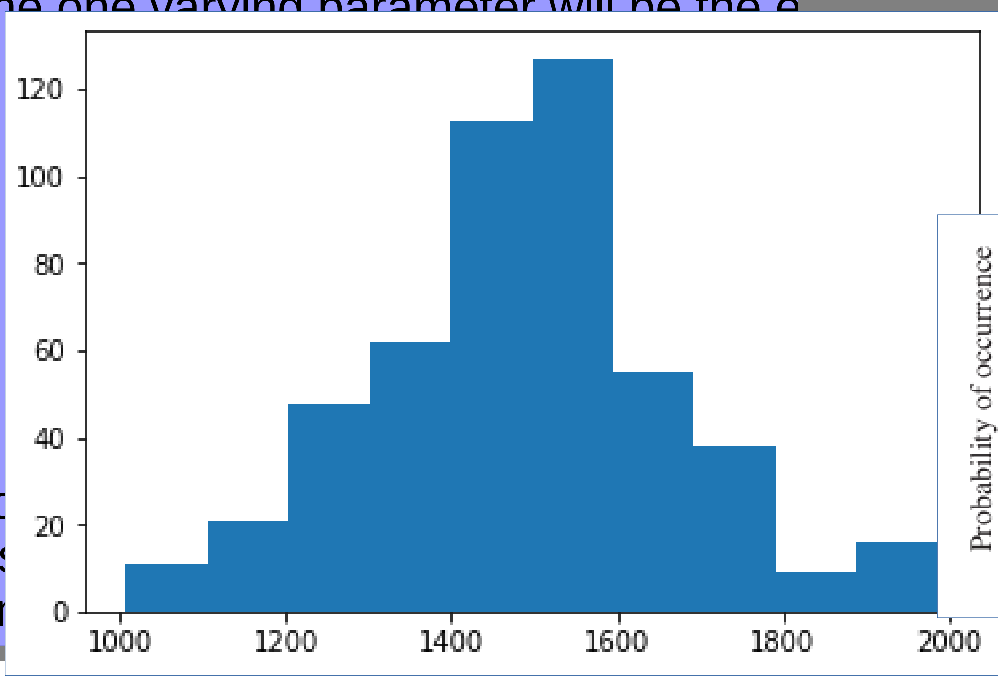
```
Q = 10000   # Delivery quantity (
T = 7       # Time between delive
N = 60      # Length of simulatio
d = 500.0   # Delivery cost (doll
s = 0.98    # Storage cost (dolla
```

The one varying parameter will be the e daily demand,
q_i
-
- 00.0
-
-

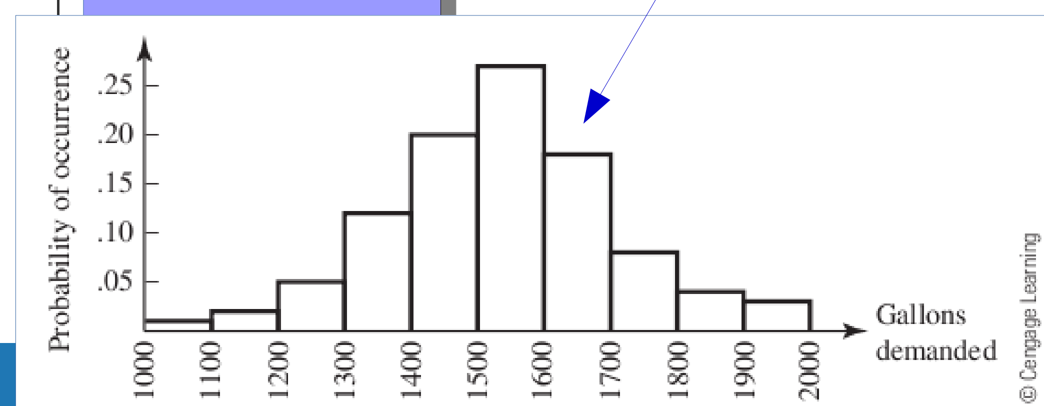
Fo
his
si

```
def qi():
    """
    Daily demand in gallons
    """
    return qi_invspline()

# Test the qi distribution
N = 500
qi_vec = np.zeros((N))
for i in np.arange(N):
    qi_vec[i] = qi()
plt.hist(qi_vec)
plt.show()
```

User-defined



Computed from history of demand table (Table 5.6) in textbook



Random from demand history q_i

Let's consider running this simulation under four different scenarios

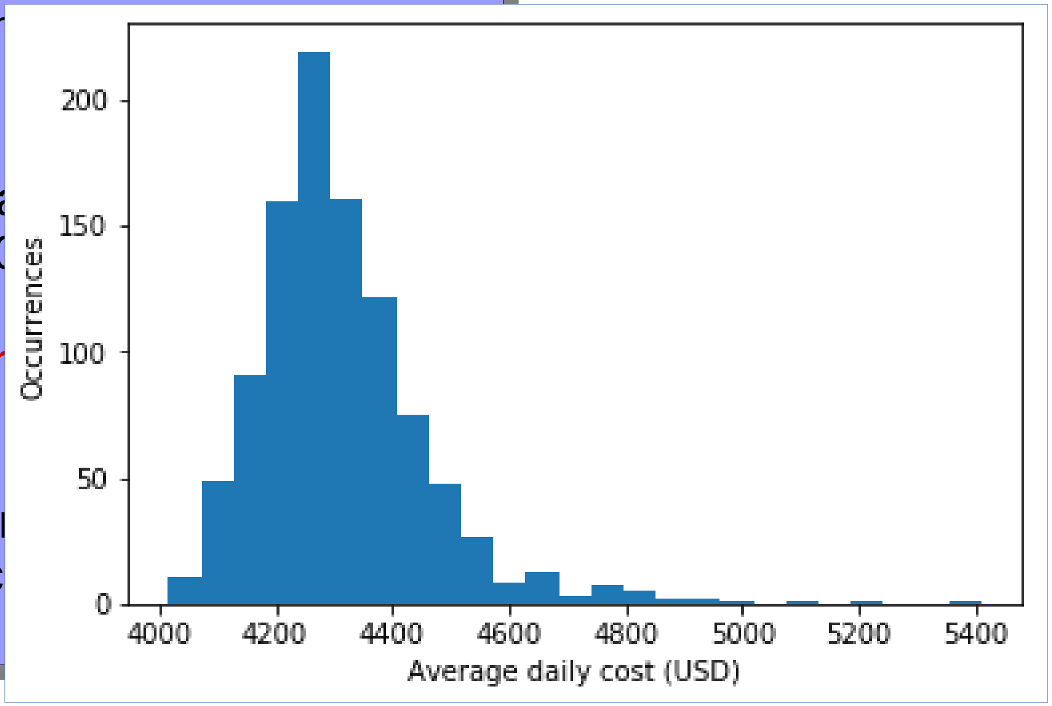
All three will have the following held constant:

```
Q = 10000   # Delivery quantity (gallons)
T = 7       # Time between deliveries (days)
N = 60      # Length of simulation (days)
d = 500.0   # Delivery cost (dollars per delivery)
s = 0.98    # Storage cost (dollars per gallon per day)
```

The one varying parameter will be the estir
$\boldsymbol{q_i}$

- Constant value of 1250.0
- Random uniform value between 1000.0 a
- Random normal value with mean of 1500 deviation of 200.0
- **Random value derived from demand h in textbook)**

For each scenario, we will perform 1000 si histogram of the computed average daily c simulation



Summary of inventory example

- Although fairly complex, it serves as a gentle introduction to more sophisticated simulations
- In our example, we hold almost all parameters constant and focus on the implementation of the submodel for daily customer demand at a gas station
- The goals of this presentation were to
  - Roughly present an overall algorithm to help clarify the textbook
  - Illustrate the insertion of different submodels for daily demand ranging from simple constant to random numbers based on measured demand
  - An important component – just in case I haven't stressed it enough – is the testing of our processes while we develop the model.
    - If you try to build the model all at once and then evaluate it, you will have a very hard time understanding if there are problems or not.
    - Finding a creative way to test each step is important for quality and sanity

Queuing model – Harbour system

- If you've taken the time to deeply understand the previous inventory model, this one should be easier to understand
- The scenario we want to simulate
  - We have one small harbour for unloading ships
  - Only one ship can be unloaded at a time
  - The time between arrivals of successive ships will vary, and we will generate random values for these
  - The time to unload a ship will depend on its cargo, and these will also be generated randomly

Queuing model – Harbour system

- If you've taken the time to deeply understand the previous inventory model, this one should be easier to understand
- The scenario we want to simulate
  - We have one small harbour for unloading ships
  - Only one
  - The time will vary, and we will g
  - The time rgo, and these will also

Courtesy Getty Images – Oil tankers and cargo ships wait to unload their cargo in the Chittagong harbour in Bangladesh

Queuing model – Harbour system

- If you've taken the time to deeply understand the prev … easier to unde…
- The …
  - We have one small harbour for unloading ships
  - Only one…
  - The time… will vary, and we will g…
  - The time… rgo, and these will also…

Questions

- What are the average and maximum times that ships spend in the harbour?
- What are the average and maximum times that ships wait to be unloaded?
- What percentage of the time is the unloading facility idle?

Courtesy Getty Images – Oil tankers and cargo ships wait to unload their cargo in the Chittagong harbour in Bangladesh

Simulation overview – big picture

- Input a specified number of ships, and for each assign a random (wrt to certain criteria)
  - Number of minutes arrived after previous ship
  - Number of minutes spent unloading
- Consider that when a ship arrives
  - There may be no other ships in the harbour, and the facility has been idle (wasting time!)
  - There may already be ships in the harbour unloading, one at a time, so it will have to wait (wasting time!)
- After all ships have been unloaded, simulation ends and we compute
  - Average and maximum time ship spends (waiting and unloading) in harbour
  - Average and maximum time ship spends waiting in harbour until it can unload
  - Percentage of time that the harbour facility has no ships to unload

Simple timeline example

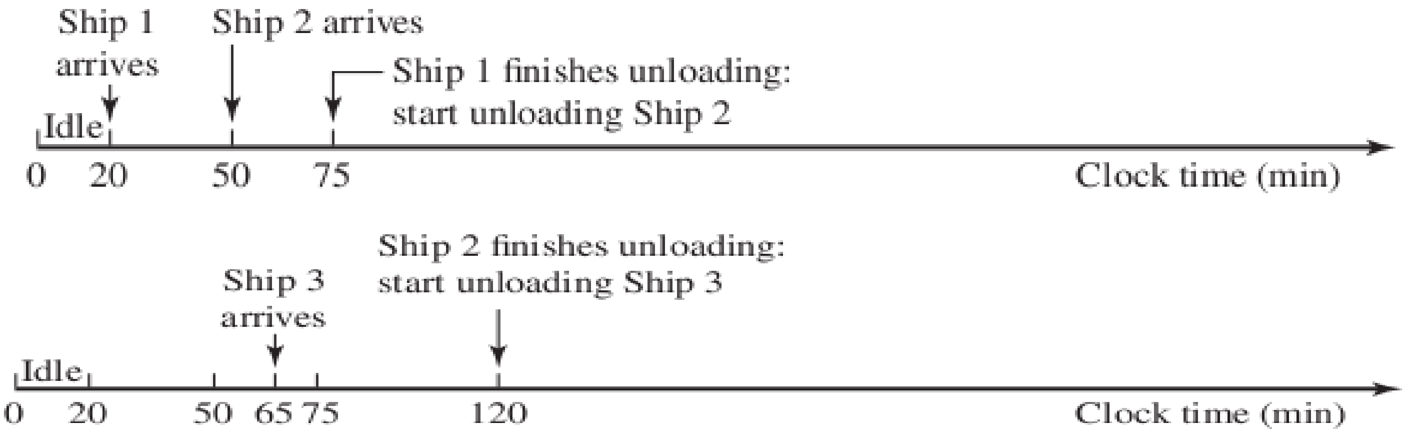
	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80

Simple timeline example

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80

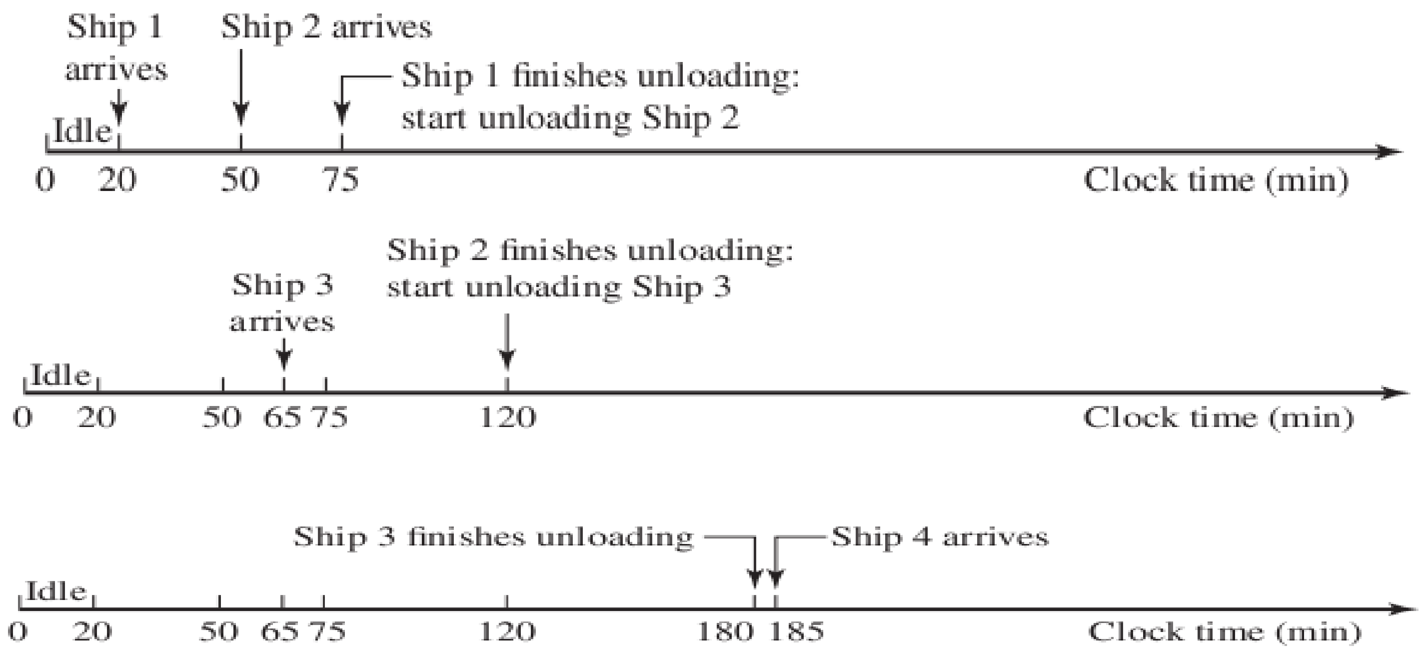
Simple timeline example

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80



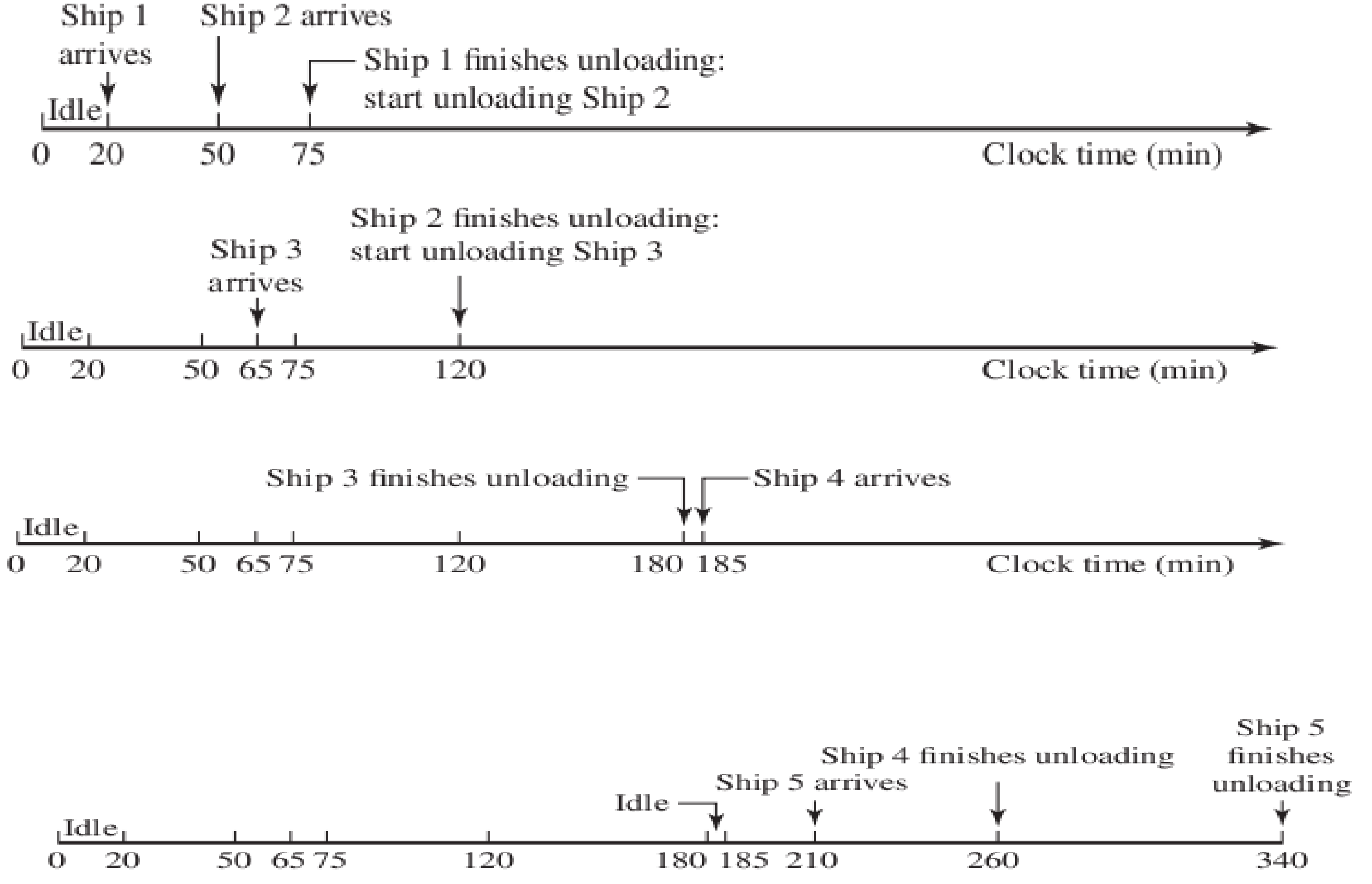
Simple timeline example

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80



Simple timeline example

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80



Simple timeline example

	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unloading time	55	45	60	75	80

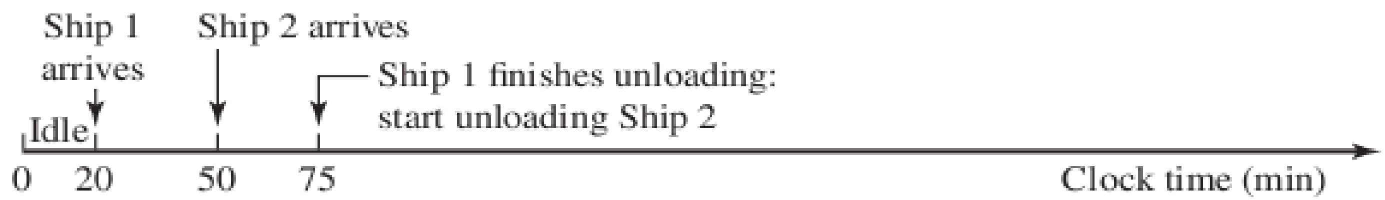
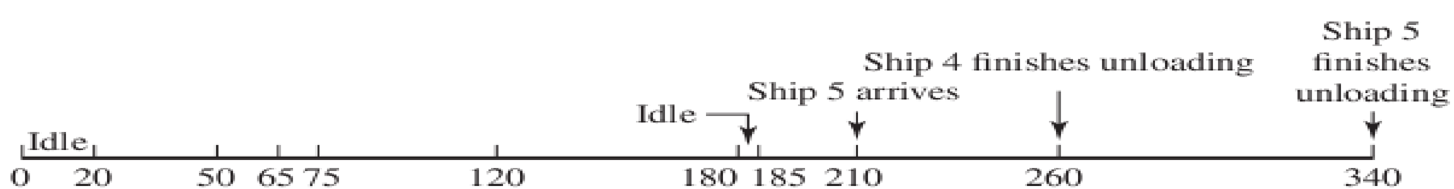


Table 5.14 Summary of the harbor system simulation

Ship no.	Random time between ship arrivals	Arrival time	Start service	Queue length at arrival	Wait time	Random unload time	Time in harbor	Dock idle time
1	20	20	20	0	0	55	55	20
2	30	50	75	1	25	45	70	0
3	15	65	120	2	55	60	115	0
4	120	185	185	0	0	75	75	5
5	25	210	260	1	50	80	130	0
Total (if appropriate):					130			25
Average (if appropriate):					26	63	89	



Simulation overview – big picture

- Input a specified number of ships, and for each assign a random (wrt to certain criteria)
  - Number of minutes arrived after previous ship
  - Number of minutes spent unloading
- Consider that when a ship arrives
  - There may be no other ships in the harbour, and the facility has been idle (wasting time!)
  - There may already be ships in the harbour unloading, one at a time, so it will have to wait (wasting time!)
- After all ships have been unloaded, simulation ends and we compute
  - Average and maximum time ship spends (waiting and unloading) in harbour
  - Average and maximum time ship spends waiting in harbour until it can unload
  - Percentage of time that the harbour facility has no ships to unload

Arrays, indexed by ship number

Arrays filled with randomly generated values
```
between[] - time between arrival of ship i and ship i-1
unload[]  - time required to unload ship i
```

Times, relative to simulation start time, t=0
```
arrive[] - time when ship i arrives
start[]  - time when ship i starts unloading
finish[] - time when ship i finishes unloading, leaves harbour
```

Other arrays
```
idle[]    - amount of time harbour is idle prior to unloading ship i
wait[]    - amount of time ship i waits in harbour before unloading
harbour[] - total time ship i spends in harbour
```

ture

(wrt to

le (wasting time!)

- There may already be ships in the harbour unloading, one at a time, so it will have to wait (wasting time!)

- After all ships have been unloaded, simulation ends and we compute
  - Average and maximum time ship spends (waiting and unloading) in harbour
  - Average and maximum time ship spends waiting in harbour until it can unload
  - Percentage of time that the harbour facility has no ships to unload

Arrays, in

Arrays filled with randomly ge

```
between[] - time between arr
unload[]  - time required to
```

Times, relative to simulation s

```
arrive[] - time when ship i
start[]  - time when ship i
finish[] - time when ship i
```

Other arrays

```
idle[]    - amount of time harbour is idle prior to unloading ship i
wait[]    - amount of time ship i waits in harbour before unloading
harbour[] - total time ship i spends in harbour
```

Start of simulation

```
# Random generation of between[1] and unload[1]
arrive[1] ← between[1]

# Initialise variables that keep track of total and
# max harbour times, total and max wait times, and
# total idle time

# Compute finish time for unloading of Ship 1
finish[1] ← arrive[1] + unload[1]
```

le (wasting time!)

- There may already be ships in the harbour unloading, one at a time, so it will have to wait (wasting time!)

- After all ships have been unloaded, simulation ends and we compute
  - Average and maximum time ship spends (waiting and unloading) in harbour
  - Average and maximum time ship spends waiting in harbour until it can unload
  - Percentage of time that the harbour facility has no ships to unload

Arrays, in

Arrays filled with randomly ge

```
between[] - time between arr
unload[]  - time required to
```

Times, relative to simulation s

```
arrive[] - time when ship i
start[]  - time when ship i
finish[] - time when ship i
```

Other arrays

```
idle[]    - amount of time har
wait[]    - amount of time shi
harbour[] - total time ship i
```

Start of simulation

```
#
ar

#
#
#

#
fi
```

Main loop

```
for each ship, i ← 2,...,n

    # generate random values between[i] and unload[i]

    # relative to t=0, compute arrive time of ship i
    arrive[i] ← arrive[i-1] + between[i]

    # compute time diff between ship i arrival and
    # finish of ship i-1
    timediff ← arrive[i] – finish[i-1]

    # Compute idle and wait times for ship i
    if timediff > 0
        idle[i] ← timediff;    wait[i] ← 0
    else
        wait[i] ← -timediff;   idle[i] = 0

    # Compute other stuff for ship i
    start[i] ← arrive[i] + wait[i]
    finish[i] ← start[i] + unload[i]
    harbor[i] ← wait[i] + unload[i]

    # Update variables for average and max harbour
    # times, average and max wait times, and idle time
end for

# Compute summaries
```

time!)

– There may already be s
wait (wasting time!)

- After all ships have bee
  - Average and maximum
  - Average and maximum
  - Percentage of time that

Arrays, in

Arrays filled with randomly ge
```
between[] - time between arr
unload[]  - time required to
```

Times, relative to simulation s
```
arrive[] - time when ship i
start[]  - time when ship i
finish[] - time when ship i
```

Other arrays
```
idle[]    - amount of time har
wait[]    - amount of time shi
harbour[] - total time ship i
```

time!)

- There may already be s ... wait (wasting time!)

- After all ships have bee
  - Average and maximum
  - Average and maximum
  - Percentage of time that

Start of simulation

```
#
ar

#
#
#

#
fi
```

Main loop

This is our main focus now

```
for each ship, i ← 2,...,n

    # generate random values between[i] and unload[i]

    # relative to t=0, compute arrive time of ship i
    arrive[i] ← arrive[i-1] + between[i]

    # compute time diff between ship i arrival and
    # finish of ship i-1
    timediff ← arrive[i] – finish[i-1]

    # Compute idle and wait times for ship i
    if timediff > 0
        idle[i] ← timediff;    wait[i] ← 0
    else
        wait[i] ← -timediff;   idle[i] = 0

    # Compute other stuff for ship i
    start[i] ← arrive[i] + wait[i]
    finish[i] ← start[i] + unload[i]
    harbor[i] ← wait[i] + unload[i]

    # Update variables for average and max harbour
    # times, average and max wait times, and idle time
end for

# Compute summaries
```

Random variable generation

- Textbook coverage specifies successive arrival times (*$between_i$*) between 15 and 145 minutes, and unloading times (*$unload_i$*) between 45 and 90 minutes
- I recommend picking easy-to-use constant values while developing model, and creating simple test cases for model verification
- From there, we can work with uniform, normal, or other randomly generated distributions
- And, as has been a majour theme in this chapter, we can use actual measurements to create an algorithm to generate these parameters in realistic proportions

Random variable generation

Table 5.18 Data collected for 1200 ships using the harbor facilities

Time between arrivals	Number of occurrences	Probability of occurrence	Unloading time	Number of occurrences	Probability of occurrence
15–24	11	0.009			
25–34	35	0.029			
35–44	42	0.035	45–49	20	0.017
45–54	61	0.051	50–54	54	0.045
55–64	108	0.090	55–59	114	0.095
65–74	193	0.161	60–64	103	0.086
75–84	240	0.200	65–69	156	0.130
85–94	207	0.172	70–74	223	0.185
95–104	150	0.125	75–79	250	0.208
105–114	85	0.071	80–84	171	0.143
115–124	44	0.037	85–90	109	0.091
125–134	21	0.017		1200	1.000
135–145	3	0.003			
	1200	1.000			

Note: All times are given in minutes.

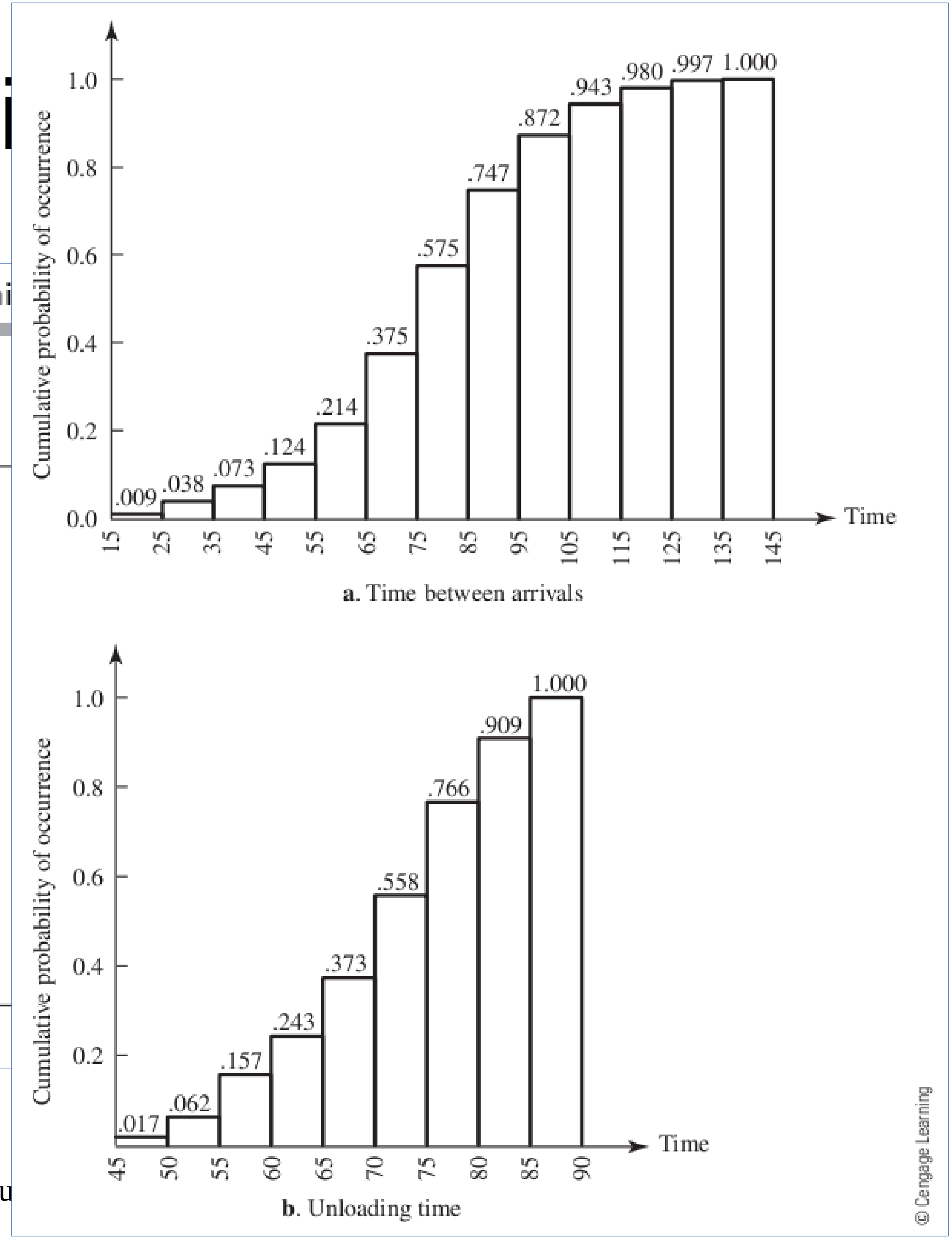
Random vari

- T
  (
  (
- I
  d
  v
- F
  r
- A
  a
  t

Table 5.18 Data collected for 1200 shi

Time between arrivals	Number of occurrences	Probability of occurrence
15–24	11	0.009
25–34	35	0.029
35–44	42	0.035
45–54	61	0.051
55–64	108	0.090
65–74	193	0.161
75–84	240	0.200
85–94	207	0.172
95–104	150	0.125
105–114	85	0.071
115–124	44	0.037
125–134	21	0.017
135–145	3	0.003
	1200	1.000

Note: All times are given in minutes.

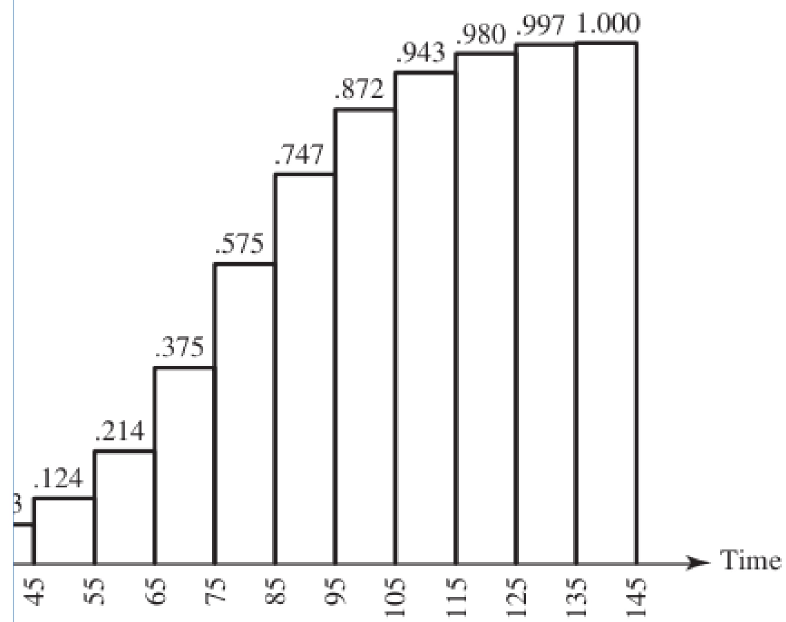


a. Time between arrivals

b. Unloading time

Table 5.19 Linear segment submodels provide for the time between arrivals of successive ships as a function of a random number in the interval [0, 1].

Random number interval	Corresponding arrival time	Inverse linear spline
$0 \le x < 0.009$	$15 \le b < 20$	$b = 555.6x + 15.0000$
$0.009 \le x < 0.038$	$20 \le b < 30$	$b = 344.8x + 16.8966$
$0.038 \le x < 0.073$	$30 \le b < 40$	$b = 285.7x + 19.1429$
$0.073 \le x < 0.124$	$40 \le b < 50$	$b = 196.1x + 25.6863$
$0.124 \le x < 0.214$	$50 \le b < 60$	$b = 111.1x + 36.2222$
$0.214 \le x < 0.375$	$60 \le b < 70$	$b = 62.1x + 46.7080$
$0.375 \le x < 0.575$	$70 \le b < 80$	$b = 50.0x + 51.2500$
$0.575 \le x < 0.747$	$80 \le b < 90$	$b = 58.1x + 46.5698$
$0.747 \le x < 0.872$	$90 \le b < 100$	$b = 80.0x + 30.2400$
$0.872 \le x < 0.943$	$100 \le b < 110$	$b = 140.8x - 22.8169$
$0.943 \le x < 0.980$	$110 \le b < 120$	$b = 270.3x - 144.8649$
$0.980 \le x < 0.997$	$120 \le b < 130$	$b = 588.2x - 456.4706$
$0.997 \le x \le 1.000$	$130 \le b \le 145$	$b = 5000.0x - 4855$

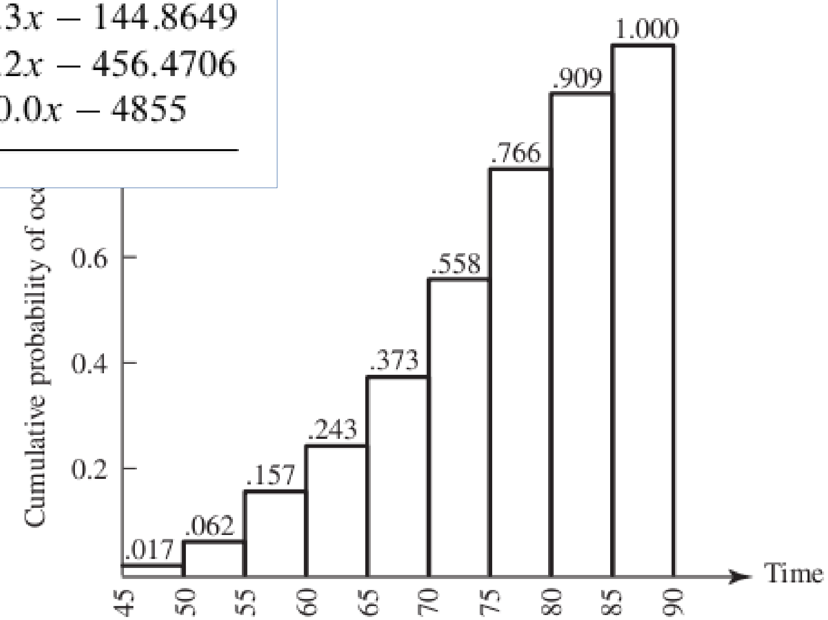


a. Time between arrivals

105–114	85	0.071
115–124	44	0.037
125–134	21	0.017
135–145	3	0.003
	1200	1.000

Note: All times are given in minutes.

- A
a
tl



b. Unloading time

Table 5.19 Linear segment submodels provide for the time between arrivals of successive ships as a function of a random number in the interval [0, 1].

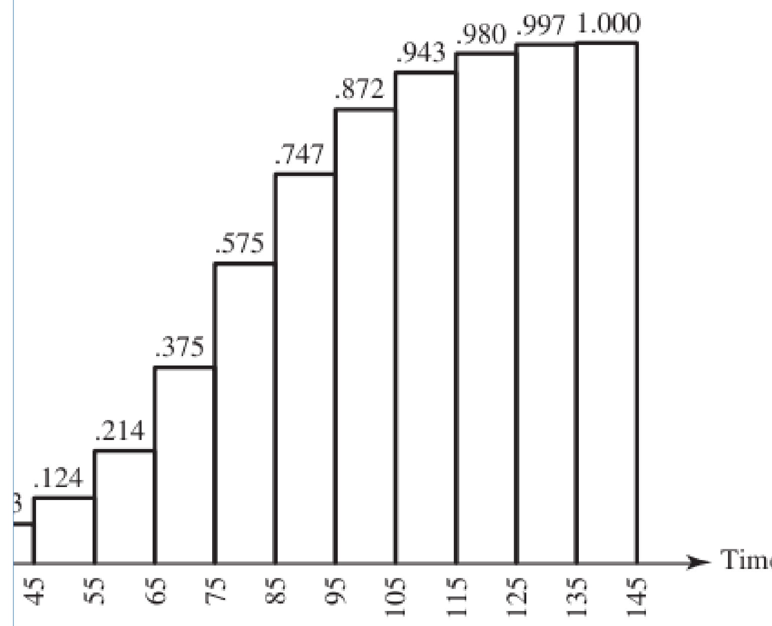
Random number interval	Corresponding arrival time	Inverse linear spline
$0 \le x < 0.009$	$15 \le b < 20$	$b = 555.6x + 15.0000$
$0.009 \le x < 0.038$	$20 \le b < 30$	$b = 344.8x + 16.8966$

Table 5.20 Linear segment submodels provide for the unloading time of a ship as a function of a random number in the interval [0, 1].

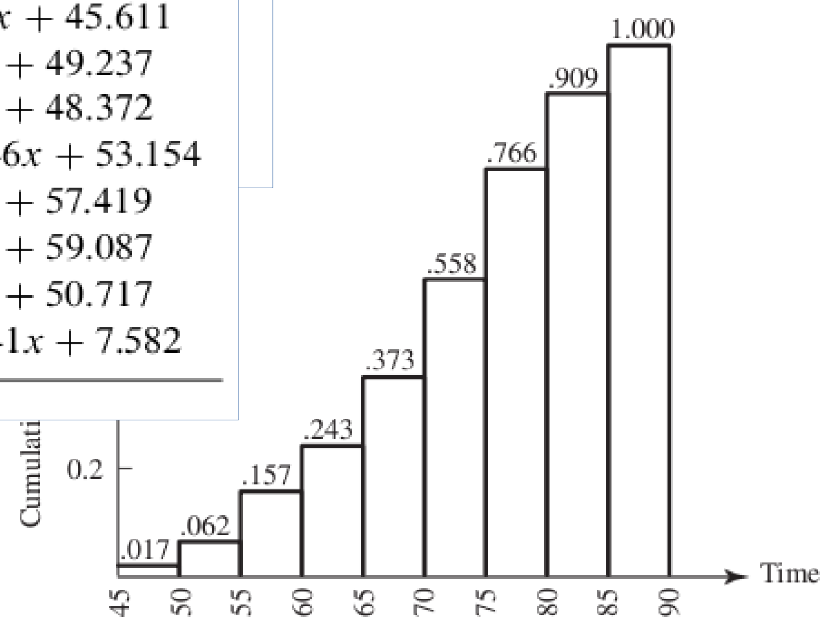
Random number interval	Corresponding unloading time	Inverse linear spline
$0 \le x < 0.017$	$45 \le u < 47.5$	$u = 147x + 45.000$
$0.017 \le x < 0.062$	$47.5 \le u < 52.5$	$u = 111x + 45.611$
$0.062 \le x < 0.157$	$52.5 \le u < 57.5$	$u = 53x + 49.237$
$0.157 \le x < 0.243$	$57.5 \le u < 62.5$	$u = 58x + 48.372$
$0.243 \le x < 0.373$	$62.5 \le u < 67.5$	$u = 38.46x + 53.154$
$0.373 \le x < 0.558$	$67.5 \le u < 72.5$	$u = 27x + 57.419$
$0.558 \le x < 0.766$	$72.5 \le u < 77.5$	$u = 24x + 59.087$
$0.766 \le x < 0.909$	$77.5 \le u < 82.5$	$u = 35x + 50.717$
$0.909 \le x \le 1.000$	$82.5 \le u \le 90$	$u = 82.41x + 7.582$

© Cengage Learning

Note: All times are given in minutes.



a. Time between arrivals



b. Unloading time

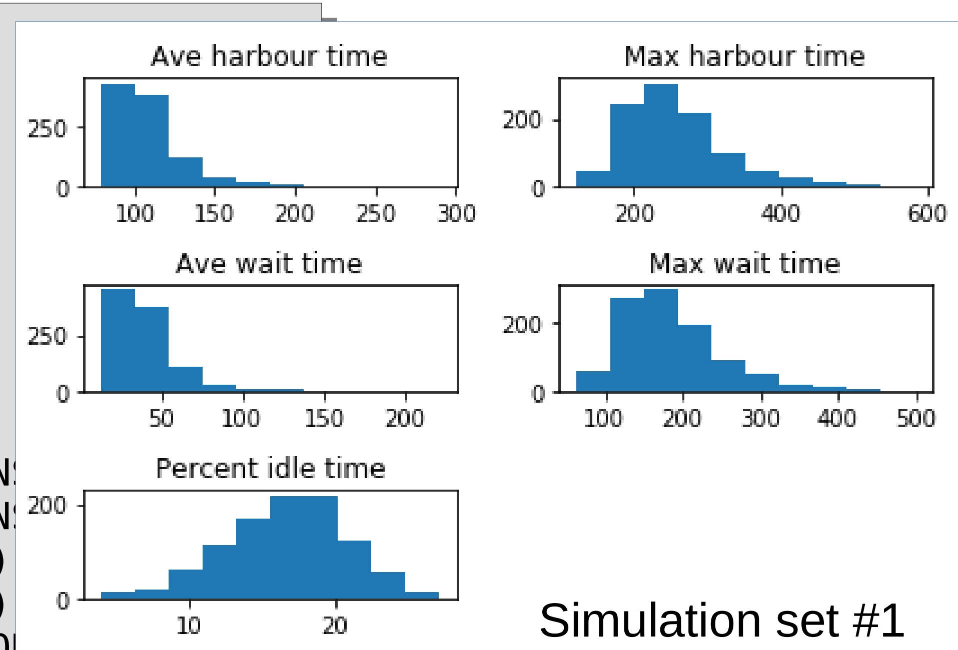
Simu

Part of my implementation

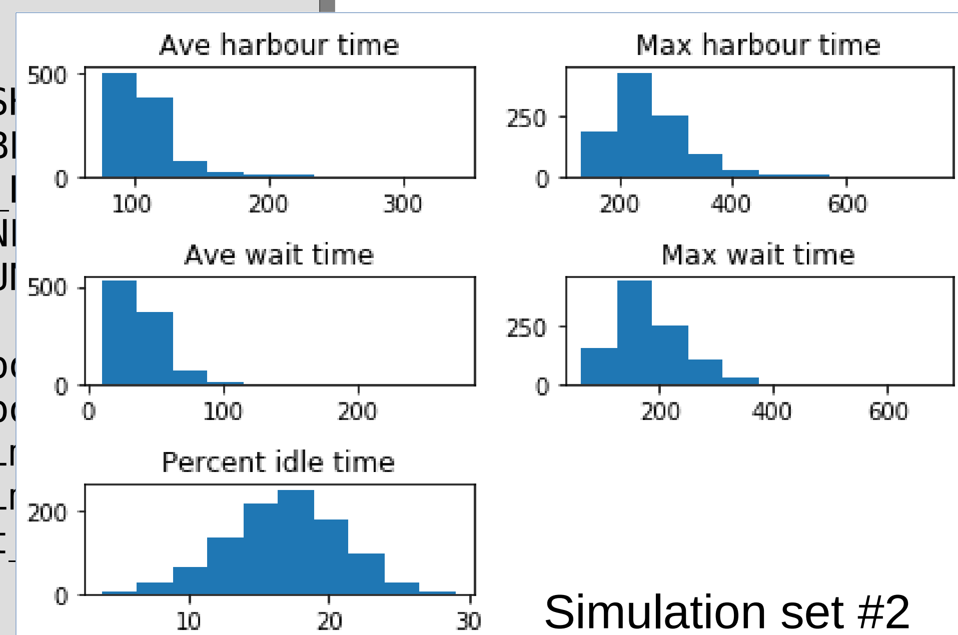
```
def main():

    NUM_SIMULATIONS = 1000
    NUM_SHIPS = 100
    MIN_BETWEEN_TIME = 15
    MAX_BETWEEN_TIME = 145
    MIN_UNLOAD_TIME = 45
    MAX_UNLOAD_TIME = 90

    # Keep track of stats for each simulation
    ave_harbour_times = np.zeros((NUM_SIMULATIONS))
    max_harbour_times = np.zeros((NUM_SIMULATIONS))
    ave_wait_times = np.zeros((NUM_SIMULATIONS))
    max_wait_times = np.zeros((NUM_SIMULATIONS))
    percent_idle_times = np.zeros((NUM_SIMULATIONS))

    for i in np.arange(NUM_SIMULATIONS):
        summary = single_simulation(nships=NUM_SHIPS,
                          between_time_low=MIN_BETWEEN_TIME,
                          between_time_high=MAX_BETWEEN_TIME,
                          unload_time_low=MIN_UNLOAD_TIME,
                          unload_time_high=MAX_UNLOAD_TIME)

        ave_harbour_times[i] = summary['ave_harbour_time']
        max_harbour_times[i] = summary['max_harbour_time']
        ave_wait_times[i] = summary['ave_wait_time']
        max_wait_times[i] = summary['max_wait_time']
        percent_idle_times[i] = summary['percent_idle_time']
.
.
.
```

Part of my implementation

```
def main():

    NUM_SIMULATIONS = 1000
    NUM_SHIPS = 100
    MIN_BETWEEN_TIME = 15
    MAX_BETWEEN_TIME = 145
    MIN_UNLOAD_TIME = 45
    MAX_UNLOAD_TIME = 90

    # Keep track of stats for each simulation
    ave_harbour_times = np.zeros((NUM_SIMULATIONS
    max_harbour_times = np.zeros((NUM_SIMULATIONS
    ave_wait_times = np.zeros((NUM_SIMULATIONS))
    max_wait_times = np.zeros((NUM_SIMULATIONS))
    percent_idle_times = np.zeros((NUM_SIMULATIONS))

    for i in np.arange(NUM_SIMULATIONS):
        summary = single_simulation(nships=NUM_SH
                                    between_time_low=MIN_BE
                                    between_time_high=MAX_B
                                    unload_time_low=MIN_UNL
                                    unload_time_high=MAX_UN

        ave_harbour_times[i] = summary['ave_harbo
        max_harbour_times[i] = summary['max_harbo
        ave_wait_times[i] = summary['ave_wait_tim
        max_wait_times[i] = summary['max_wait_tim
        percent_idle_times[i] = summary['percent_
.
.
.
```



Simulation set #1



Simulation set #2