

The Monte Carlo Method

Jan Verschelde

e-mail: jan@math.uic.edu

www.math.uic.edu/~jan

MCS 494 Special Topics in Computer Science:
Industrial Math & Computation, Fall 2002.

Adapted from Chapter 2 of “*Industrial Mathematics.
Modeling in Industry, Science, and Government*”
by Charles R. MacCluer.

Maple Solution to MTBF

```
> mu_1 := 11: mu_2 := 12: mu_3 := 13:
> sigma_1 := 1: sigma_2 := 2: sigma_3 := 3:
> d_1 := n -> stats[random,normald[mu_1,sigma_1]](n):
> d_2 := n -> stats[random,normald[mu_2,sigma_2]](n):
> d_3 := n -> stats[random,normald[mu_3,sigma_3]](n):

> N := 10000:
> s1 := d_1(N): s2 := d_2(N): s3 := d_3(N)

> s := zip((x,y,z) -> min(x,y,z), [s1], [s2], [s3]):
> describe[mean](s); describe[standarddeviation](s);
```

10.49809352

1.125215107

Servicing Requests – distribution of arrival time

```
> a := 10: # mean arrival time
> arrival_time := n -> stats[random,exponential[1/a,0]](n)
> samples := arrival_time(1000):
> describe[mean]([samples]);
```

10.17281213

```
> describe[standarddeviation]([samples]);
```

10.01505618

Servicing Requests – distribution of service time

```
> mu := 20: sigma := 1: # defines a normal distribution
```

```
> service_time := n -> stats[random,normald[mu,sigma]](n):
```

```
> service_time(3);
```

```
20.84015141, 19.28742867, 19.34413727
```

Busy processors reject jobs

The variables:

```
> m := 3:           # number of processors
> N := 10000:       # number of simulations
> B := [seq(0,i=1..m)]: # nobody is busy
> rejected_ones := 0: # count rejected requests
```

The main loop:

```
> for i from 1 to N do # conduct N experiments
```

In the loop:

1. sample and decide whether to accept or reject
2. update counter of rejections

Running the simulation N times

```
> A := arrival_time(1):      # new arrival time
> S := service_time(1):     # new service time
> for j from 1 to m do      # update status of processors
>   if B[j] < A             # A is later than busy time
>     then B[j] := 0:       # release processor
>     else B[j] := B[j]-A:  # decrease service time
>   end if:
> end do:
```

Keeping track of rejections

```
> reject := 1:           # assume we reject the job
> for j from 1 to m do # try to assign job to processor
>   if reject = 1 and B[j] = 0
>     then B[j] := S:   # assign job to j-th processor
>       reject := 0: # if assigned, then no reject
>   end if;
> end do:                # now we can update the counter
> rejected_ones := rejected_ones + reject;
> end do:                # end of main loop
```

Jobs wait in a queue

The variables:

```
> m := 3:           # number of processors
> N := 10000:       # number of simulations
> B := [seq(0,i=1..m)]: # nobody is busy
> waiting_time := 0: # sums total waiting time
```

The main loop

```
> for i from 1 to N do           # conduct N experiments
>   A := arrival_time(1):       # new arrival time
>   S := service_time(1):      # new service time
>   for j from 1 to m do       # update status of processors
>     if B[j] < A
>       then B[j] := 0:        # release processor
>       else B[j] := B[j]-A:   # decrease service time
>     end if:
>   end do:
>   min_time := min(op(B)):     # look for minimum
>   member(min_time,B,'p'):     # processor with minimal time
>   waiting_time := waiting_time + B[p]:
>   B[p] := B[p] + S:          # assign job to this processor
> end do:
```