With *Mathematica* and J: Gasoline Inventory Simulation

Cliff Reiter

Computational exercises and demonstrations appear frequently in the classes that I teach at Lafayette College. The software that I use for in-class demonstrations is usually *Mathematica* [4] or J [3]. I focus on one language or the other throughout the course when I expect the students to develop the ability to make substantial explorations and do independent experimentation. There is an ongoing expectation at Lafayette that *Mathematica* be used in Calculus Classes [2]; I also use *Mathematica* for my Numerical Analysis and Techniques of Math Modeling classes. I use J for teaching Number Theory and Linear Algebra and it is central to my mathematical visualization course that follows the text [6]. However, I always have interest in how the explorations that I do in one language would be done in the other.

I recently asked my math modeling students to explore an inventory simulation. While this is a toy problem, it involves a substantial data set, the need to develop a Monte Carlo Simulation of demands, and exploration of the behaviors that appear with various choices of parameters. After the exercise, one student mentioned that her family runs a fuel oil business and they have software to determine likely demand and they input data similar to what we used in the simulation. Her family was surprised and pleased to know she was learning something useful in college.

That encouraged me to share the exercise here. In particular, in this note we compare the *Mathematica* template solution for this simulation presented in class with an analogous effort in J.

Simulation Overview

The inventory demand simulation from our Techniques of Math Modeling text [1] is premised on having data for the number of gallons of gasoline sold by a station for a thousand weeks. The pattern involves demands of between 1000 and 2000 gallons per week broken into categories of size 100, with the most frequently occurring categories being toward the middle. We understand that there may be various costs associated with the delivery and storage of the gasoline and we desire not to run out of gasoline or exceed the station's storage capacity. The idea is to use the data to generate a realistic Monte Carlo simulation of demand, so that we might experiment with delivery strategies.

Random Reals, Linear Interpolation, Monte Carlo Simulation: *Mathematica*

We can easily generate random "real" numbers between 0 and 1. Timing the creation of a million random numbers of this type (ten times) shows that these sizable samples can typically be generated in about 0.2 seconds.

```
In[1]:= Random[]
Out[1]= 0.874221

In[2]:= Table[Random[], {5}]
Out[2]= {0.292129, 0.728089, 0.342794, 0.961358, 0.307707}

In[3]:= t = Table[Timing[x = Table[Random[], {1000000}];],
          {10}]
Out[3]= {{0.22 Second, Null}, {0.21 Second, Null}, {0.211 Second, Null},
         {0.2 Second, Null}, {0.22 Second, Null}, {0.211 Second, Null},
         {0.21 Second, Null}, {0.21 Second, Null}, {0.21 Second, Null},
         {0.211 Second, Null}}

In[4]:= Mean[First[Transpose[t]]]
Out[4]= 0.2113 Second
```

In order to simulate demands so that they appear in the categories with some specified frequencies, we first consider a small data set. Suppose that we want to be able to generate a random number from the intervals [0,100), [100,200), and [200,300) with frequencies 0.2, 0.5 and 0.3 respectively. That is, we assume that 20%

of the time the demand will be between 0 and 100. See Table I below. We then compute the cumulative sums as follows.

Demand	Frequency	Cumulative
$0 \le x < 100$	0.2	0.2
$100 \le x < 200$	0.5	0.7
$200 \le x < 300$	0.3	1.0

Table I. Simplified Demand and Cumulative Frequency

```
In[5]:= << Statistics`DataManipulation`

In[6]:= x = {0, 100, 200, 300}

Out[6]= {0, 100, 200, 300}

In[7]:= y = {0, 0.2, 0.5, 0.3}

Out[7]= {0, 0.2, 0.5, 0.3}

In[8]:= cy = CumulativeSums[y]

Out[8]= {0, 0.2, 0.7, 1.}
```

We select a random real from [0,1) and use that to select a random demand. For example, whenever the random real is between 0.7 and 1 we recognize that as from the third interval, which occurs with the appropriate frequency 0.3 and linearly interpolate in order to simulate a specific demand. Thus, to produce a random demand, we produce a random real and then apply the piecewise linear function through the points given by cumulative frequencies (with a leading 0 pre-pended) paired with the demand interval endpoints.

The *Mathematica* function `Interpolation` with an appropriate option can be used to create the piecewise linear interpolating function. It is the inverse to the cumulative frequency function and given a number from [0,1) results in the simulated demand.

```
In[9]:= f = Interpolation[Transpose[{cy, x}],
          InterpolationOrder → 1]

Out[9]= InterpolatingFunction[{ 0.    1. }, <>]

In[10]:= f[0]

Out[10]= 0.

In[11]:= f[0.2]

Out[11]= 100.

In[12]:= f[0.6]

Out[12]= 180.
```

The random demand function can then be defined and the fact that it produces the expected frequencies of numbers in the specified categories is illustrated as follows.

```
In[13]:= randd[n_] := Table[f[Random[]], {n}]

In[14]:= randd[2]

Out[14]= {191.488, 260.732}

In[15]:= Timing[s = randd[1000000];]

Out[15]= {4.667 Second, Null}

In[16]:= Frequencies[Floor[s / 100]]

Out[16]= ⎛199693   0⎞
         ⎜500428   1⎟
         ⎝299879   2⎠
```

Notice that creating a million random numbers with the desired distribution took less than 5 seconds.

Random Reals, Linear Interpolation, Monte Carlo Simulation: J

We can likewise easily generate random "real" numbers between 0 and 1 in J. Timing the creation of a million random numbers of this type (ten times) shows that a million such numbers can typically be generated in about 0.06 seconds, more than 3 times faster than *Mathematica*.

```
   ? 0
0.971379

   ? 5 $ 0
0.0466292  0.330109  0.01346  0.220004
0.600044

   timing=:6!:2

   10 timing 'x=: 1000000 ?@:$ 0'
0.0559868
```

Creating the small data set, including cumulative sums is straightforward; we use plus insert infixes for the cumulative sum.

```
   x=:0 100 200 300

   y=:0 0.2 0.5 0.3
```

```
   ]cy=:+/\y
0 0.2 0.7 1
```

Creating the piecewise linear function that fits the `cy` and `x` data can be done in various ways. One is to use the piecewise linear function from *raster5.ijs*, a script associated with the old (edition 2) of scripts associated with *Fractals, Visualization and J* [5]. Another, inspired by the cubic spline functions in the J script *spline.ijs*, is discussed below. We define the function `xy_to_p` that takes lists of `x` and `y` coordinates of the points for which we want the piecewise linear function as its arguments (`x` is assumed to be ordered). It produces the list of coefficients of the linear functions on the intervening subintervals. The adverb `lin_spline` creates an efficient function for evaluating that piecewise linear function.

```
   xy_to_p=:4 : 0
(2 ]\ y) %."1 2]2 (1&,.)\ x
)

   lin_spline=:1 : 0
'X Y'=.m
p=.X xy_to_p Y
{&p@:(0>.<:)@:(+/)@:}:@:(X&(<:/))p. ]
)

   f=:(cy;x) lin_spline

   f 0 0.1 0.2 0.6 0.7 1 1.1
0 50 100 180 200 300 333.333
```

Note that we extended the domain of the piecewise linear function so that it is extended beyond the endpoints of its domain using the linear functions on the corresponding ends. Now we define the random demand function and verify that it produces values in the expected intervals with the expected frequencies.

```
   randd=: f@:?@:$&0

   randd 2
206.029 136.503

   10 timing 's=:randd 1000000'
0.583247

   /:~ (({.,#)/.~)<.s%100
0 199767
1 500216
2 300017
```

Notice that the expected frequencies appear and that J produces the random demands almost ten times faster than in *Mathematica*.

The Text's Problem

The actual class illustration that we used was to create a simulated demand function `randd` for the data in Table II along with a function that runs day-by-day simulations of the daily demand. It also keeps track of some other information along the way.

Demand	Frequency	Cumulative
$1000 \le x < 1100$	0.01	0.01
$1100 \le x < 1200$	0.02	0.03
$1200 \le x < 1300$	0.05	0.08
$1300 \le x < 1400$	0.12	0.20
$1400 \le x < 1500$	0.20	0.40
$1500 \le x < 1600$	0.27	0.67
$1600 \le x < 1700$	0.18	0.85
$1700 \le x < 1800$	0.08	0.93
$1800 \le x < 1900$	0.04	0.97
$1900 \le x < 2000$	0.03	1.00

Table II. Demand and Cumulative Frequencies

We will define a function *MCIS* that does a Monte Carlo Inventory Simulation over a specified period. We assume that a certain quantity Q of gasoline is delivered with T days between deliveries. The simulation begins with day zero and ends with day N; At the start, the tanks are empty and a delivery is made. We also assume there is a delivery cost d per delivery and storage costs s per gallon per day. Internal to the function *MCIS*, n denotes the day, c the cumulative cost, In the current inventory level and z gives the accumulated values of n, In and c.

Monte Carlos Inventory Simulation: *Mathematica*

First we obtain the text data and demand function. Then we implement and apply the simulation using deliveries of 10,000 gallons every week with delivery

costs of $500 per delivery, storage costs of one-fifth of a penny per gallon per day. The result for the simulation to day 10 is shown.

```
In[17]:= x = Range[1000, 2000, 100]

Out[17]= {1000, 1100, 1200, 1300, 1400,
          1500, 1600, 1700, 1800, 1900, 2000}

In[18]:= y = {0, 0.01, 0.02, 0.05, 0.12, 0.2, 0.27,
           0.18, 0.08, 0.04, 0.03}

Out[18]= {0, 0.01, 0.02, 0.05, 0.12, 0.2, 0.27, 0.18, 0.08, 0.04, 0.03}

In[19]:= cy = CumulativeSums[y]

Out[19]= {0, 0.01, 0.03, 0.08, 0.2, 0.4, 0.67, 0.85, 0.93, 0.97, 1.}

In[20]:= f = Interpolation[
           Transpose[{CumulativeSums[y], x}],
           InterpolationOrder → 1]

Out[20]= InterpolatingFunction[{{ 0.    1. }}, <>]

In[21]:= randd[n_] := Table[f[Random[]], {n}]
In[22]:= MCIS[Q_, T_, d_, s_, N_] :=
          Block[{In = 0, c = 0, n = 0, z = {}},
           While[n <= N,
            c = c + In * s;
            If[0 == Mod[n, T], In = In + Q; c = c + d];
            In = In - f[Random[]];
            z = Join[z, {{n, In, c}}];
            n = n + 1];
           z
          ]

In[23]:= MCIS[10000, 7, 500, 0.002, 10]

Out[23]= ⎛ 0   8592.07    500    ⎞
         ⎜ 1   7158.11    517.184⎟
         ⎜ 2   5670.84    531.5  ⎟
         ⎜ 3   4401.39    542.842⎟
         ⎜ 4   2804.81    551.645⎟
         ⎜ 5   1258.9     557.254⎟
         ⎜ 6  −303.861    559.772⎟
         ⎜ 7   8122.42    1059.16⎟
         ⎜ 8   6437.06    1075.41⎟
         ⎜ 9   4827.97    1088.28⎟
         ⎝ 10  3263.11    1097.94⎠
```

Notice that at day 6 the inventory became negative and at that point the data becomes invalid since, for example, storage cost can not be negative. We are primarily interested in designing a delivery strategy with reduced cost while minimizing the likelihood of having inventory below 0 or above 20,000 (the capacity of our system). Thus, we define a function to give some summaries over longer runs. We decide that we will be pleased with a strategy if when we run the plan for a month, there is less than a 5% likelihood of falling outside the desired inventory range. Below we use the summary function on 31 day simulations with the minimum inventory, maximum inventory, last inventory and total cost given for each run.

```
In[25]:= MCISsum[Q_, T_, d_, s_, N_] :=
          Block[{In, c, n},
           {n, In, c} = Transpose[MCIS[Q, T, d, s, N]];
           {Min@@In, Max@@In, Last[In], Last[c]}
          ]

In[26]:= Table[MCISsum[10000, 6, 500, 0.002, 30], {3}]

Out[26]= ⎛ 615.279   12058.1   12058.1   3373.2  ⎞
         ⎜ 331.183   12941.4   12941.4   3361.01 ⎟
         ⎝ 668.671   11967.6   11967.6   3340.73 ⎠
```

Notice that there is a delivery on the last day, and that the maximum inventory is the same. Thus, it appears there is a mild accumulation of gasoline. Thus, the class was presented with a reasonable but not refined strategy for deliveries. They were asked to design a strategy that reduced costs further given the availability of larger tanker trucks. Most students had no trouble finding cost reductions, and staying within the other constraints. A couple students were able to give really nice descriptions leading to compelling choices.

The class then explored a similar problem involving the number of miles run by a runner during a week, based upon tabulated daily data. They were expected use the same tools, with far less specific guidance.

Monte Carlo Inventory Simulation: J

We update our simulated demand function and translate the *Mathematica* Monte Carlo Simulation almost directly into J.

```
   x=:1000+100*i.11

   y=:0 0.01 0.02 0.05 0.12 0.2 0.27
0.18 0.08 0.04 0.03

   ]cy=:+/\y
0 0.01 0.03 0.08 0.2 0.4 0.67 0.85
0.93 0.97 1
```

```
   f=:(cy;x) lin_spline

   randd=: f@:?@:$&0

   MCIS=: 3 : 0
'Q T d s N'=.y
c=.n=.In=.0
z=.i.0 3
while. n<: N do.
  c=.c+In*s
  if. 0=T|n do.
    In=.In+Q
    c=.c+d
  end.
  In=.In-randd 1
  z=.z,n,In,c
  n=.n+1
end.
z
)

   MCIS 10000 7 500 0.002 10
 0  8095.4     500
 1 6862.15 516.191
 2 5397.09 529.915
 3 4279.79 540.709
 4 2869.79 549.269
 5  1295.7 555.008
 6  _142.2   557.6
 7 7973.12 1057.32
 8 6246.84 1073.26
 9 4643.85 1085.76
10 3063.55 1095.04
```

Notice that we again see that we run out of gasoline on day 6. We use tacit functions to create the summary information. The verb `Ip` gives the inventory information while `Lp` gives the information from the last part.

```
   Ip=:(<./,>./)@:(1&{)@:|:

   Lp=:_2&{.@{:

   MCISsum=: (Ip,Lp)@:MCIS"1

   MCISsum 3#,:10000 6 500 0.002 30
1355.22 13483.7 13483.7 3411.46
451.477 10811.3 10811.3 3323.94
760.336 12548.4 12548.4 3351.77
```

Furthermore, we implement a loop-less version of `MCIS` and create the corresponding summary function easily enough, and get identical results so long as we reset the random seed appropriately.

```
   MCISa=:3 : 0
'Q T d s N'=.y
dd=.0=T|i.N+1
In=.+/\(Q*dd)-randd N+1
c=.+/\(d*dd)+s*0,}:In
(i.N+1),.In,.c
)

   9!:1]7^5

   MCISa 10000 7 500 0.002 10
 0  8095.4     500
 1 6862.15 516.191
 2 5397.09 529.915
 3 4279.79 540.709
 4 2869.79 549.269
 5  1295.7 555.008
 6  _142.2   557.6
 7 7973.12 1057.32
 8 6246.84 1073.26
 9 4643.85 1085.76
10 3063.55 1095.04

   MCISsuma=: (Ip,Lp)@:MCISa"1
```

Timing tests of 100 repetitions of 31 day simulations using *MCIS* take about 0.12 seconds total in both *Mathematica* and J, making the running time fairly insignificant. However, the loop-less J version (`MCISsuma`) is more than 10 times faster. We did not try a loop-less implementation of the *Mathematica* function, but we expect it would also be more efficient than the loopy version.

Postscript

We see that demand simulations can be readily implemented in *Mathematica* and J. These run quickly, but the J implementations were usually faster on core number crunching and its loop-less version ran very quickly. I note that both J and *Mathematica* would benefit by better help indices. Finding whether there are suitable linear splines does not seem to amount to simply typing "linear spline" or "piecewise linear" into the help search menu for either language. Readers may

want to glance back at the looped *Mathematica* and J implementations of `MCIS`. The flow of the J implementation is easier for my eye to follow. Indeed, I do not hesitate to use explicit, loopy functions when sharing J models with classes. Students are often more comfortable modifying such functions than tacit ones. However, regardless of the language, when efficiency becomes a driving issue, utilizing array arithmetic should be considered and that was easy to implement in the simulation that we did in J.

References

[1] Giordano, Weir & Fox, *A First Course in Mathematical Modeling, 3rd edition*, Thomson, Brooks/Cole, 2003.

[2] L. T. Hill and C. A. Reiter, Laboratories as an Enhancement to Calculus, *The Mathematica Journal*, 2 1 (1992) 41-44.

[3] Jsoftware, J6.01c, *http://www.jsoftware.com*, 2007.

[4] *Mathematica*™ 5.2, *http://www.wolfram.com* Wolfram Research, Inc., 2005.

[5] C. A. Reiter, *Fractals, Visualization and J, 3rd edition*, Published by Lulu, *http://www.lulu.com*, 2007.

[6] C. A. Reiter, Archive of *FVJ2* material, *http://www.lafayette.edu/~reiterc/j/index.html*, 2006.