Simulating Discrete Random Variables

Binomial

One way to simulate a binomial random variable is to simulate the events of which it is composed. If this binomial random variable has parameters n and p, then we can independently simulate n events, each with probability p of being a *success*. The number of *successes*, then, would be the outcome of the binomial random variable. The code listed below uses this method to create an integer valued function that returns the outcome of the random variable: (Note that in this function and in all other code in this discussion the type `extended` is used. This is a floating point variable type that is defined in Turbo Pascal. It has about 20 significant digits and can range from 3.4×10^{-4932} to 1.1×10^{4932}. This provides for greater accuracy in some of these applications; this is particularly necessary in the Poisson function on the next page. I recommend that you take advantage of any high precision real types that your compiler supports. In Fortran and C the type `double` carries about 16 significant digits.)

```
function binomial(n : integer; p : double): integer;

var i : integer;
    successes : integer;

begin
successes := 0;
for i := 1 to n do
   if random < p then inc(successes);
binomial := successes;
end;
```

Geometric

Here again we will use the method of simulating the events that make up the random variable. For a geometric random variable with parameter p we simulate independent events, each with probability p of a *success* occurring, until we observe a *success*. The number of tries it takes to get the first *success* is the outcome of the random variable.

```
function geometric(p : double): integer;

var i : integer;
    success : boolean;  (* becomes true if a success occurs *)

begin
success := false;
i := 0;
while not success do
  begin
  success := random < p;
  i := i + 1;
  end;
geometric := i;
end;
```

Poisson

The listing below is a function that will return the outcome of a poisson random variable with parameter λ. We will not be able to discuss the method used here until later in the course.

```
function poisson(lambda : double): longint;

var i : longint;
    product : double;
    compare : double;

begin
compare := exp(-lambda);
product := random;
i := 0;
while product > compare do
  begin
  product := product * random;
  i := i + 1;
  end;
poisson := i;
end;
```

Application

The code listed below generates 1000 observations from a binomial random variable with $n = 20$ and $p = 0.10$. It keeps track of the min and max of all the outcomes. Also it keeps track of $\sum_{i=1}^{1000} x_i$ and $\sum_{i=1}^{1000} x_i^2$ and uses these to calculate the sample mean (usually denoted by $\bar{x}$) and the sample variance (usually denoted by s^2). These have the following computational formulas:

$$\bar{x} = \frac{\sum_{i=1}^{m} x_i}{m} \qquad s^2 = \frac{\sum_{i=1}^{m} x_i^2 - \frac{(\sum_{i=1}^{m} x_i)^2}{m}}{m-1}$$

where here $m = 1000$. If m is large (1000 is fairly large) then these shouldn't be too far from the true mean and true variance ($\mu = np$ and $\sigma^2 = np(1-p)$).

```
program GenerateDiscreteDistributions;

var x : double;
    Min, Max : integer;
    SampleMean, SampleVariance : double;
    Sum, SumSq : double;
    i : integer;

(* insert random function here *)

(* insert binomial function here *)

begin
Randomize; (* Seeds Turbo Pascal's random number generator *)
Min := 50;
Max := 0;
Sum := 0;
SumSq := 0;
For i := 1 to 1000 do
begin
  x := Binomial(20,0.10);
  Sum := Sum + x;
  SumSq := SumSq + x * x;
  if x < Min then Min := x;
  if x > Max then Max := x;
end;
SampleMean := Sum/1000;
SampleVariance := (SumSq - (Sum * Sum)/1000)/1000;
WriteIn('Sample mean = ', SampleMean:1:6);
WriteIn('Sample variance = ', SampleVariance:1:6);
WriteIn('min = ', Min:1:0);
WriteIn('max = ', Max:1:0);
```