randoms.pl

Tim Menzies
Lane Deptartment of Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26506
tim@menzies.us

Contents

1 Random Predicates

Lets thrash around a little.

2 Header

2.1 Flags */

```
:- arithmetic_function(inf/0).
:- arithmetic_function(rand/0).
:- arithmetic_function(rand/2).
:- arithmetic_function(rand/3).
:- arithmetic_function(normal/2).
:- arithmetic_function(beta/1).
:- arithmetic_function(gamma/2).  /*
```

3 Any Random Solution */

```
any(X) :-
       setof(One,X^any1(X,One),All),
       member(_/X,All).

any1(X,Score/X) :-
       X,
       Score is rand. /*
```

4 Best

Same code as above, but with some assessment criteria thrown in. */

```
best(X) :-
       setof(One,X^best1(X,One),All0),
       beam(X,All0,All),
       member(_/X,All).

best1(X,Score/X) :-
       X,
       score(X,Score0),
       bound(X,Score0),
       Score is -1 * Score0. /*
```

If we have no knowledge of `X`, give it a random number. */

```
score(_,N) :-
       N is  rand(2147483647). /*
```

If we have knowledge of minimum values for a score, test it here. */

```
bound(_,Score) :-
       Score > 0. /*
```

Sometimes, we may just want to select the top *N* values: makes this a beam search. The default case is that there is no selection knowledge. */

```
beam(_,L,L).  /*
```

5 Rand */

```
rand(X)         :- X is random(inf)/inf.
rand(Min,Max,X) :-  X is Min + (Max-Min)*rand. /*
```

5.1 Beta */

```
rand(Min,Max,B,X) :-
   X is Min + (Max-Min)*beta(B).

beta(B,X) :- beta1(B,X),!.
beta(B,X) :- beta1(0.5,X),print(user,badBeta(B)).

beta1(0,0).
beta1(0.1,X)  :- X is 1- randˆ(1/9).
beta1(0.20,X) :- X is 1- randˆ0.25.
beta1(0.25,X) :- X is 1- randˆ0.33.
beta1(0.33,X) :- X is 1- randˆ0.5.
beta1(0.4,X)  :- X is 1- randˆ0.67.
beta1(0.50,X) :- X is rand.
beta1(0.60,X) :- X is randˆ0.67.
beta1(0.67,X) :- X is randˆ0.5.
beta1(0.75,X) :- X is randˆ0.33.
beta1(0.80,X) :- X is randˆ0.25.
beta1(0.9,X)  :- X is randˆ(1/9).
beta1(1,1). /*
```

5.2 Normal */

```
normal(M,S,N) :- box_muller(M,S,N).

box_muller(M,S,N) :-
   wloop(W0,X),
   W is sqrt((-2.0 * log(W0))/W0),
   Y1 is X * W,
   N is M + Y1*S.

wloop(W,X) :-
   X1 is 2.0 * rand - 1,
   X2 is 2.0 * rand - 1,
   W0 is X1*X1 + X2*X2,
   (W0  >= 1.0 -> wloop(W,X) ; W0=W, X = X1). /*
```

5.3 Gamma

(Only works for integer Alphas.)

A standard random *gamma* distribution has *mean=alpha/beta*. The *alpha* value is the "spread" of the distribution and controls the clustering of the distribution around the mean. As *alpha* increases, the *gamma* distribution evens out to become more evenly-distributed about the mean. That is, for large *alpha* (i.e. above 20), *gamma* can be modeled as a noraml function. The standard *alpha,beta* terminology can be confusing to some audiences. Hence, I define a (slightly) more-intuitive *gamma* distribution where:

myGamma(mean,alpha) = standardGamma(alpha,alpha/mean) */

```
gamma(Mean,Alpha,Out) :-
   Beta is Mean/Alpha,
   (Alpha > 20
   ->  Mean is Alpha * Beta,
       Sd is sqrt(Alpha*Beta*Beta),
       Out is normal(Mean,Sd)
   ;   gamma(Alpha,Beta,0,Out)).

gamma(0,_,X,X) :- !.
gamma(Alpha,Beta, In, Gamma) :-
   Temp is In + ( -1 * Beta * log(1-rand)),
   Alpha1 is Alpha - 1,
   gamma(Alpha1,Beta,Temp,Gamma). /*
```

5.4 Random Strings */

```
rstring(_,X)  :- nonvar(X),!.
rstring(A,X)  :- gensym(A,X). /*
```

5.5 Random Symbols */

```
rsym(X) :- rsym(g,X).

rsym(_,X)  :- nonvar(X),!, atom(X).
rsym(A,X)  :- gensym(A,X). /*
```

5.6 Random Members of a List */

```
rin(X,L) :- number(X),!, member(Y,L), X =:= Y.
rin(X,L) :- nonvar(X),!, member(X,L).
rin(X,L) :- any(member(X,L)).  /*
```

5.7 Random Numeric Taken From Some Range

The default case is that we step from some `Min` to `Max` number in incremets of one. */

```
rin(Min,Max,X) :- rin(Min,Max,1,X). /*
```

The usual case is that we step from some `Min` to `Max` number in incremets of `I`. */

```
rin(Min,Max,_,X) :- nonvar(X),!, number(X),Min =< X, X =< Max.
rin(Min,Max,I,X) :- any(rin1(Min,Max,I,X)).

rin1(Min,Max,I,X) :-
       Steps is integer((Max-Min)/I),
       between(0,Steps,Step),
       Num is Min + Step*I,
       X is min(Num,Max).  /*
```

6 Footer

6.1 Start-ups */

```
:- current_prolog_flag(max_integer,X),
   X1 is X - 1,
   retractall(inf(_)),
   assert(inf(X1)).
```