```
float[][] randomMarkov( int size , float probaDiag ){

  float[][] markov = new float[size][size] ;

  for( int i = 0 ; i < size ; ++i ){ // Loop over the lines of the Markov matrix //
    //Pick size-2 random points in [0,1-probaDiag]
    float[] probasCumul = new float[size] ;
    for( int k = 1 ; k < size-1 ; ++k ){
      probasCumul[k] = random(1-probaDiag) ;
    }
    //
    //Make sure the points are in the right order, to make it easy to compute interval lengths
    probasCumul = sort(probasCumul) ;
    probasCumul[0] = probaDiag ;
    //
    //Convert from point to interval length.
    probasCumul[1] = 1-probaDiag - probasCumul[size-1] ;
    for( int k = size-1 ; k>2 ; --k ){
      probasCumul[k] - = probasCumul[k-1] ;
    }
    //
    //Place the diagonal coefficient on the diagonal
    swap( probasCumul , i , 0 ) ;
    //
    //Store the probabilities in the matrix that will be returned in the end
    for( int j = 0 ; j < size ; ++j ){
      markov[i][j] = probasCumul[j] ;
    }
    //
  }
  return markov ;
}
```

I want to return a Markov matrix whose coefficients are picked at random, except for the diagonal ones that I specify by hand to make sure they are big enough (N.B. : they represent the probabilty that the colour will remain the same from one square to the next.).

So I have to uniformly pick size-1 probabilities that sum up to 1-probaDiag. To do so I place size-2 points on the [0,1-probaDiag] segment, since they delimitate size-1 subintervals, whose lengths sum up to exactly 1-probaDiag. Here's an example where size=5 :