Aleatoriedade e Processos de Monte Carlo

Pedro Barahona
DI/FCT/UNL

Introdução aos Computadores e à Programação
2º Semestre 2010/2011

Aleatoriedade

- Em muitas aplicações informáticas, nomeadamente em simulações que utilizam processos aleatórios ou estocásticos, muito utilizadas em engenharia.
- Um processo aleatório é aquele cujo resultado não é conhecido com rigor, mas que obedece a uma distribuição de probabilidades.
- Por exemplo, no lançamento de uma moeda ao ar, não se conhece o resultado a priori, mas, se a moeda não estiver viciada, assume-se que cerca de 50% das vezes sai cara, enquanto que nas outras cerca de 50% de vezes sai coroa. Já o número de vezes que no lançamento de um dado sai um "2" é de 1/6 (ou 16.66%).
- Tais processos podem ser simulados com recurso a números (pseudo-) aleatórios, conhecidos como processos ou simulações de **Monte Carlo**.
- Estes números são obtido por um **gerador de números aleatórios**, que no caso do Octave é invocado pela função pre-definida **`rand()`**, ou simplesmente **`rand`**.
- Esta função retorna um número aleatório entre no intervalo **[0, 1[** de cada vez que é invocada.

Exemplos: Moeda ao ar e Lançamento de um Dado

- Por exemplo o processo de **n** lançamentos de uma moeda ao ar, em que 1 significa caras e 2 significa coroas, pode ser simulado pela função

```
function L = moeda(n);
   for i = 1:n
      if rand > 0.5  L(i) = 1
      else           L(i) = 2
      endif
   end for
endfunction
```

- Já o lançamento de dados pode ser simulado pela função

```
function L = dado(n);
   for i = 1:n
      r = rand;
      if      r < 1/6  L(i) = 1;
      elseif  r < 2/6  L(i) = 2;
      ...
      elseif  r < 5/6  L(i) = 5
      else             L(i) = 6
      endif
    end for
endfunction
```

Exemplos: Moeda ao ar e Lançamento de um Dado

- Na realidade, para grandes números a frequência aproxima-se da probabilidade, isto é, se lançarmos 10 vezes uma moeda ao ar, o número de caras deve ser próximo de 5, pois a probabilidade de sair caras é ½ e ½ *10 = 5.
- Em geral a frequência de ocorrrência de um valor aleatório aproxima-se da sua probabilidade para grandes valores de n. Mais formalmente, denotando por
  - **p(X=a)** ou simplesmente $\mathbf{p_a}$, a probabilidade de um evento X tomar o valor A;
  - **n** o número de eventos e
  - $\mathbf{e_a}$ o valor esperado do número de ocorrências de X=a
  - $\mathbf{f_a}$ o número efectivo de ocorrências de X=a

  temos

  $$\mathbf{e_a = n * p_a} \text{ ; e}$$

  $$\mathbf{f_a \approx e_a},$$

  devendo o erro relativo $\varepsilon\mathbf{(n) = (\ f_a - e_a)/\ n}$ diminuir com o aumento do número de eventos.

Exemplo: Geração de Tômbolas

- Quando o número de possibilidades é muito elevado o esquema anterior é pouco prático. Basta pensar na simulação de extração de n bolas de um totoloto ou toto-milhoes com cerca de 50 bolas, que exigiria uma instrução condicional "impraticável"

```
r = rand();
if      r <  1/50  L(i) =  1;
elseif  r <  2/50  L(i) =  2;
 ...
elseif  r < 49/50  L(i) = 49;
else               L(i) = 50;
endif
```

- A utilização das funções de arredondamento normal (round), para cima (ceil) e para baixo (floor), permite uma grande simplificação de código. Em particular, a simulação de números de 1 a 50 pode ser gerado com a função ceil como indicado abaixo

```
function L = tombola(n);
   for i = 1:n
      L(i) = ceil(50*rand())
   end for
endfunction
```

Passeio Aleatório

Exemplo: Uma pessoa pretende percorrer um passeio entre as duas vias de uma estrada, em que começando no centro do passeio, pode guinar até k vezes para a direita ou a esquerda antes de sair do passeio e ser atropelada. Estando "bêbada", em cada passo que dá tem uma probabilidade de 50% de guinar para a direita .e outra igual de guinar para a esquerda. Sabendo-se que deverá dar n passos, qual a probabilidade de chegar ao fim da estrada sem sair do passeio?

- Para resolver este problema, basta utilizar a função passeio, que para um número **total** de percursos de **n** passos num passeio que admite **k** desvios, conta o número **na** de "atropelamentos".

```
function p = passeio(n,k,total);
   na = 0;
   for i = 1:total
      na = na + atropelado(n,k);
   endfor
   p = na/total
endfunction
```

- Naturalmente a função passeio utiliza uma função atropelado que deve retornar **1** se houver atropelamento e **0** no caso contrário.

Passeio Aleatório

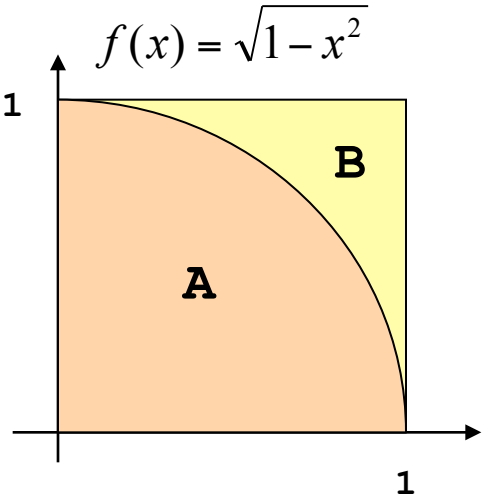
- Para a função atropelado, vamos considerar uma variável desvio, que começa em 0 e vai acumulando os desvios para a direita e para a esquerda.
- Se conseguir chegar ao fim do ciclo for sem atingir um desvio, em módulo, maior que k, a função retorna 0.
- De notar que logo que o desvio seja maior que k, a função pode retornar imediatamente o valor 1, saindo do ciclo for com a instrução de excepção **return**.

```
function p = atropelado(n,k);
   p = 0; desvio = 0;
   for i = 1:n
      if rand > 0.5  d = 1; else d = -1; endif;
      desvio = desvio + d;
      if abs(desvio) > k
         p = 1;
         return;
      endif
   endfor
endfunction
```

Área de uma Curva

- Os numeros aleatórios também podem ser utilizados na determinação aproximada de áreas. O raciocínio é o seguinte:
  1. Contornemos a área a determinar A, por uma área conhecida, B
  2. Consideremos um conjunto aleatório de pontos na área conhecida, e contabilizar a percentagem p de pontos que "cai" dentro da área a determinar.
  3. A área A pode ser aproximada por $A \approx pB$.

- Exemplo:
  1. Calculemos a área do ¼ de circulo de raio 1 ao lado.
  2. A área B é naturalmente de 1
  3. A área do ¼ de circulo deve ser próxima de $\pi/4$

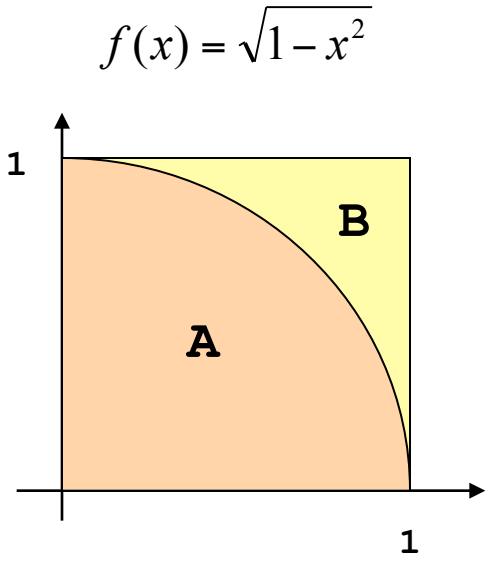


- Falta determinar a forma de gerar pontos aleatórios no quadrado, o que se pode fazer facilmente gerando x e y aleatórios no intervalo [0,1[e verificando se ficam abaixo da função y = sqrt(1-x^2)

Área de uma Curva

- Este algoritmo está implementado na função pi_prob abaixo indicada.
  - O ciclo for gera n pontos <x,y>,
  - Destes n pontos , c estão abaixo da curva do ¼ círculo.
  - A área do ¼ de círculo é a fracção c/n da área do quadrado (1*1)
  - O valor de pi é 4 vezes a área

```
function p = pi_prob();
   c = 0;
   for i = 1:n;
      x = rand(); y = rand();
      if y < sqrt(1-x^2)
         c = c+1;
      endif
   endfor
   area = (1*1)* c/n;
   p = 4*area;
endfunction
```



Outras Distribuições

- Muitos fenómenos, os eventos não seguem uma distribuição uniforme mas outras (normal, exponencial, etc).
- Em geral uma função de distribuição (conhecida como fdp - função de densidade de probabilidade) goza de algumas propriedades importantes:
  a. É definida apenas e para todos os valores que os eventos podem tomar;
  b. Nos pontos em que é definida toma valores positivos;
  c. A "superfície" delimitada pela fdp tem área 1.
- Alguns exemplos de fdp com esta propriedade são:
  - **Distribuição uniforme** :

    f(x) = 1 / (b-a) definida no intervalo X = [a , b];
  - **Distribuição normal:**

    f(x) = exp(-x2/s) definida no intervalo X =] -∞ , +∞ [;
  - **Distribuição exponencial**:

    f(x) = (1/m)* exp(-x/m) definida no intervalo X = [0 , +∞ [;

Outras Distribuições

- Neste contexto, e apesar de existirem métodos particulares para algumas distribuições, podemos simular, a partir de uma distribuição uniforme, uma qualquer distribuição de probabilidades, definida pela função **f** no intervalo **a .. b**, pelo método da aceitação/ rejeição, que segue o seguinte procedimento:
  1. Delimitar a fdp por um rectângulo, com base a..b e cuja altura h, não inferior ao máximo de f(x) no intervalo a..b.
  2. Gerar, com distribuição uniforme, um número x no intervalo a .. b;
  3. Gerar, com distribuição uniforme, um número y no intervalo 0 .. h;
  4. Aceitar o evento com valor x, no caso de ser y < f(x) .
- Naturalmente no caso do domínio da função ser infinito, este deve ser “truncado” de forma a que a área truncada da função seja muito inferior a 1.

Outras Distribuições

- Podemos assim simular a distribuição exponencial

$$f(x) = (1/m)^* \exp(-x/m) \text{ definida no intervalo } X = [\ 0\ ,\ +\infty\ [;$$

com a seguinte função:

```
function r = rand_exp(m, k);
  do
    x = k * rand();
    y = (1/m) * rand();
  until y < (1/m) * exp(-x/m)
endfunction
```

Notas:

1. Assume-se que o domínio da função é runcado para o intervalo 0..k. Nesse intervalo, o valor máximo da função é 1/m (no ponto x = 0);
2. Gera-se, com distribuição uniforme, um número x no intervalo 0.. k;
3. Gera-se, com distribuição uniforme, um número y no intervalo 0 .. 1/m;
4. Aceita-se o evento com valor x, no caso de ser y < f(x) .