Monitoria 9 nov

Números aleatórios

> Será que existem?

Na verdade a aleatoriedade é bem difícil de determinar matematicamente.

- **Falácia do apostador**

  O que acontece anteriormente não tem impacto no que vai acontecer em seguida.

- **O que é aleatoriedade?**

  Um processo aleatório é o processo repetitivo cujo resultado não descreve um padrão determinístico, mas segue uma distribuição de probabilidade. [1] (Fonte: Wikipedia)

- **Computadores podem gerar números aleatórios?**

  Não podem. Computadores são máquinas determinísticas e por isso são incapazes de gerar sequências sem nenhuma inter-relação ou regra clara para definir o próximo número.

Geradores de números pseudo-aleatórios

Números aleatórios no Fortran

Função `random_number(harvest)`:

Gera números aleatórios no intervalo $[0, 1]$ baseados numa *seed,* no caso *harvest.* Também pode gerar vetores inteiros de números aleatórios

Exemplo do Heitor:

https://gist.github.com/heitorPB/abc750898443d6302b0b733c8a87faa5#números-aleatórios

```
program teste
        implicit none

        ! numeros reais
        real(4)  :: r4
        real(8)  :: r8
        real(16) :: r16

        ! uma matriz 4x4 de reais
        real(8) :: muitos(4,4)

        ! inicia o gerador de numerinhos
        call random_seed()

        ! todo mundo recebe um número pseudoaleatório
        call random_number(r16)
        call random_number(r8)
        call random_number(r4)
        call random_number(muitos)

        ! olha só o resultado:
        print *, r16, r8, r4
        print *, muitos(1,:)
        print *, muitos(2,:)
        print *, muitos(3,:)
        print *, muitos(4,:)
end program teste
```

Resultado:

```
$ gfortran aleat.f90 -o aleat
$ ./aleat
  0.874676781539981845053480738391677618        0.57505145031418081       0.541935444
  0.73207031044638016       0.28057229020748109        1.9748846033733503E-002  0.35994364391829448
  0.32109422766730178       0.85784435637327083        1.1146918705856490E-002  0.12327309451436308
  0.68715436105949601       0.97579903334737772       0.69876248466784485       0.78320390097181725
  0.45095051262175967       0.74165725644654046       0.86718512447000606        9.9055518599761361E-002
```

Algoritmo de Metropolis

O algoritmo Metropolis-Hastings funciona gerando uma sequência de valores de tal maneira que, à medida que **mais e mais valores de amostra são produzidos,** a distribuição de valores se **aproxima mais da distribuição desejada $P(x)$.**

Esses valores de amostra são produzidos iterativamente, com a distribuição da próxima amostra **dependendo apenas do valor atual da amostra** (transformando a sequência de amostras em uma cadeia de Markov). Especificamente, a cada iteração, o algoritmo escolhe um candidato para o próximo valor de amostra com base no valor atual da amostra.

Então, **com alguma probabilidade, o candidato é aceito ou rejeitado**. Caso seja rejeitado, o valor do candidato é descartado e o valor atual é reutilizado na próxima iteração. A probabilidade de aceitação é determinado comparando os valores da função $f(x)$ dos valores de amostra atuais e candidatos em relação à distribuição desejada $P(x)$.

Para cada iteração:

- Gere um candidato x' para a próxima amostra, escolhendo na distribuição $g\left(x' \mid x_t\right)$.
- Calcule a taxa de aceitação $\alpha = f\left(x'\right)/f\left(x_t\right)$, que será usado para decidir se aceita ou rejeita o candidato. Como f é proporcional à densidade de , temos que $\alpha = f\left(x'\right)/f\left(x_t\right) = P\left(x'\right)/P\left(x_t\right)$
- Aceite ou rejeite :
  - Gere um número aleatório uniforme $u \in [0, 1]$.
  - E se $u \leq \alpha$, aceite o candidato definindo $x_{t+1} = x'$,
  - E se $u > \alpha$, rejeite o candidato e defina $x_{t+1} = x_t$ em vez de.

Pseudo-código:

```
P = 0.5 #Distribuição uniforme
i = random_number() #número entra 0,1

if i < P:
  ...
```