MAC2166 - Introdução à Computação

Escola Politécnica – Grande Área Elétrica – Primeiro Semestre de 2020

Segundo Exercício-Programa (EP2)
Data de entrega: **EP2 – 16/5/2020**

Veja em https://www.ime.usp.br/~mac2166/infoepsC as instruções de entrega dos exercícios-programa.

Robustez de eleições

Neste EP, vamos estudar a robustez do resultado de eleições quando as urnas falham com uma certa probabilidade, mudando o voto de alguns eleitores. Por exemplo, suponha que vai se realizar uma eleição com apenas dois candidatos, A e B, e com $N = 1.000.000$ votantes; não há votos brancos, nulos, ou abstenções. A porcentagem de votantes no candidato A é $a = 51\%$, porém cada voto pode ser registrado com uma **falha da urna**, ou seja, o voto pode ser alterado para o outro candidato com probabilidade de falha $f = 5\%$. As falhas da urna ocorrem de forma independente para cada votante. Uma primeira pergunta natural é:

> qual é a probabilidade de que o candidato B obtenha mais votos que A?

Se esse evento ocorrer, a eleição teve um **erro**. (Note que a palavra **falha** é utilizada para a falha de uma urna num voto, enquanto a palavra **erro** é utilizada para o resultado incorreto de uma eleição.) Naturalmente, a probabilidade de erro em uma eleição é uma função crescente da probabilidade de falha. Ela pode ser estimada utilizando experimentos aleatórios que simulam uma eleição T vezes: em cada simulação contamos os votos do candidato A com aquela possibilidade de falha nas urnas, e, caso ele perdesse, contamos o erro na eleição. A probabilidade de erro será estimada pela fração entre o número de eleições com erro pelo total T de simulações. Uma outra pergunta natural é:

> qual é a maior probabilidade de falha de urnas para que a probabilidade de erro numa eleição seja no máximo uma certa tolerância, tol?

Esta é a pergunta que seu programa deve responder, tendo em mãos:

- N: o número de votantes,
- a: a porcentagem de votantes no candidato A,
- T: o número de simulações usadas para estimar a probabilidade de erro numa eleição,
- tol: a probabilidade máxima aceita para erro numa eleição.

Exemplos de entrada e saída

```
Digite o numero de votantes (0 < N <= 2x10^9): 100
Digite a porcentagem de votos do candidato A (0.5 < a <= 1): .55
Digite o numero de simulacoes (0 < T <= 2x10^9): 10000
Digite a probabilidade toleravel de erros (0 <= tol <= 1): .01
Maior falha das urnas toleravel: 0.0470386
```

```
Digite o numero de votantes (0 < N <= 2x10^9): 1000
Digite a porcentagem de votos do candidato A (0.5 < a <= 1): .51
Digite o numero de simulacoes (0 < T <= 2x10^9): 10000
Digite a probabilidade toleravel de erros (0 <= tol <= 1): .01
Maior falha das urnas toleravel: 0.0197148
```

Estrutura do programa

Note pelo exemplo de entrada e saída acima que seu programa deve ler do usuário os parâmetros N, a, T e tol, nos limites especificados.

Seu programa deve obrigatoriamente começar com as linhas

```
/*****************************************************************/
/**                                                             **/
/**   Fulano de Tal (é o nome do aluno)          Número USP    **/
/**   Exercício-Programa xx                                     **/
/**   Professor: Ciclano de Tal                                 **/
/**   Turma: (01, 02, ou 03)                                    **/
/**                                                             **/
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define BISSEC_TOL (1e-6)
/* #define RANDOM_SEED 1234 */

/* Funções implementadas abaixo, devem ser copiadas no seu código */
void ativa_sorteador();
double sorteia_real();
```

```
/* Funções que vocês devem implementar, seguindo o protótipo abaixo */
int sorteia_voto_com_falha (double f);
double bissecta (int N, double a, int T, double tol);
double prob_erro (int N, double a, double f, int T);

/* "dá a partida" no gerador de números aleatórios */
void ativa_sorteador()
{
#ifdef RANDOM_SEED
  srand(RANDOM_SEED);
#else
  srand(time(NULL));
#endif
}

/* devolve um real sorteado uniformemente no intervalo [0,1] */
double sorteia_real()
{
  return (double) rand() / RAND_MAX;
}
```

Obviamente, o cabeçalho deve estar preenchido com seus dados. Além disso, você tem permissão para adicionar mais protótipos de funções que decidir criar junto dos protótipos já existentes. Além disso, logo após a declaração de variáveis na função `main`, você deve fazer a chamada de função:

```
  ativa_sorteador();
```

A função `sorteia_real`, que já está implementada acima, devolve um `double` sorteado uniformemente no intervalo $[0, 1]$. Ao executar o programa várias vezes, os valores devolvidos serão diferentes, o que pode causar dificuldades para depuração do código. Para este fim, a linha que define a constante `RANDOM_SEED` logo após os `#include` pode ser descomentada. Isso fará com que a execução do programa sempre sorteie os mesmos valores.

Você necessariamente deve implementar as seguintes funções com estes protótipos e especificações:

- `int sorteia_voto_com_falha (double f)`: devolve um inteiro não nulo com probabilidade `f`, e devolve 0 com probabilidade $1 - \texttt{f}$, onde $0 \leq \texttt{f} \leq 1$.
- `double prob_erro (int N, double a, double f, int T)`: estima a probabilidade de erro de uma eleição com `N` votantes, dos quais uma fração `a` vota no candidato A, e com probabilidade de falha `f`, utilizando `T` simulações. Os limites dos parâmetros são como no exemplo de entrada e saída.

- `double bissecta (int N, double a, int T, double tol)`: calcula a probabilidade máxima de falha das urnas para que a probabilidade de erro de uma eleição seja limitada superiormente por `tol`, utilizando o método da bissecção, como descrito a seguir.

Note que, devido ao experimento aleatório para estimar probabilidades, seu programa pode estar correto mesmo se não imprimir números exatamente idênticos aos dos exemplos acima. Porém, eles devem ser bem próximos aos dos exemplos.

Não se esqueça de rodar seu programa com os dados do exemplo do primeiro parágrafo, com $T = 100$.

O método da bissecção

Leia a página da Wikipedia sobre o método da bissecção. No nosso caso, queremos estimar a maior probabilidade de falha que ainda garante a vitória de A na eleição com probabilidade pelo menos $1 - \texttt{tol}$ (lembre que estamos supondo que uma fração maior que 50% dos votantes vota em A, e que queremos que a probabilidade de a eleição resultar em erro seja de no máximo `tol`). Observe que sabemos que com probabilidade $f = f_a = 0$ (não há falhas), certamente A ganhará a eleição. Da mesma forma, se a probabilidade for $f = f_b = 1$, ocorrerá erro, e B vencerá de forma incorreta. A ideia do método é, com posse destas duas probabilidades f_a e f_b, olhar o que acontece no ponto médio dos dois valores. Se neste ponto A ganha a eleição com probabilidade pelo menos $1 - \texttt{tol}$, podemos atualizar f_a. Por outro lado, se neste ponto a chance de vitória de B supera a tolerância `tol`, atualizamos f_b. Este processo é repetido até que a diferença de f_a e f_b seja menor que o valor `BISEC_TOL`.