7
Referências Bibliográficas

ACOSTA-MEJIA, Cesar A. **Improved *p* Charts to Monitor Process Quality**. IIE Transactions, v. 31, p. 509-516, 1999.

BORROR, Connie M.; CHAMP, Charles W.; RIGDON, Steven E. **Poisson EWMA Control Charts**. Journal of Quality Technology, v. 30, n. 4, p. 352-361, 1998.

BOURKE, Patrick D. **Detecting a Shift in Fraction Nonconforming Using Run-Length Control Charts with 100% Inspection**. Journal of Quality Technology, v. 23, n. 3, p. 225-238, 1991.

CESAR, Flávia de Lima. **Gráficos Adaptativos de Controle de Processo por Atributos**. Dissertação de Mestrado, Departamento de Engenharia Industrial, Pontifícia Universidade Católica do Rio de Janeiro, 2000.

CHANDRASEKARAN, S.; ENGLISH, J. R.; DISNEY, R. L. **Modeling and Analysis of EWMA Control Schemes With Variance-Adjusted Control Limits**. IIE Transactions, v. 27, p. 282-290, 1995.

COSTA, A. F. B.; EPPRECHT, E. K.; CARPINETTI, L. C. R. **Controle Estatístico de Qualidade**. São Paulo: Editora Atlas, 2004.

CROWDER, Stephen V. **Average Run Lengths of Exponentially Weighted Moving Average Control Charts**. Journal of Quality Technology, v. 19, n. 3, p. 161-164, 1987a.

CROWDER, Stephen V. **A Simple Method for Studying Run-Length Distributions of Exponentially Weighted Moving Average Charts**. Technometrics, v. 29, n. 4, p. 401-407, 1987b.

CROWDER, Stephen V. **Design of Exponentially Weighted Moving Average Schemes**. Journal of Quality Technology, v. 21, n. 3, p. 155-162, 1989.

EPPRECHT, E. K.; COSTA, A. F. B. **Adaptive Sample Size Control Charts for Attributes**. Quality Engineering, v. 13, n. 3, p. 465-473, 2001.

EPPRECHT, E. K.; COSTA, A. F. B.; MENDES, F. C. T. **Adaptive Control Charts for Attributes**. IIE Transactions, v. 35, p. 567-582, 2003.

GAN, F. F. **Monitoring Observations Generated from a Binomial Distribution Using Modified Exponentially Weighted Moving Average Control Chart**. Journal of Statistical Computation and Simulation, v. 37, p. 45-60, 1990a.

GAN, F. F. **Monitoring Poisson Observations Using Modified Exponentially Weighted Moving Average Control Charts**. Communications in Statistics – Simulation and Computation, v.19, n. 1, p. 103-124, 1990b.

GAN, F. F. **An Optimal Design of CUSUM Control Charts for Binomial Counts**. Journal of Applied Statistics, v. 20, n. 4, p. 445-460, 1993.

GAN, F. F. **Designs of Optimal Exponential CUSUM Control Charts**. Journal of Quality Technology, v. 26, n. 2, p. 109-124, 1994.

GAN, F. F. **Designs of One- and Two-Sided Exponential EWMA Charts**. Journal of Quality Technology, v. 30, n. 1, p. 55-69, 1998.

GAN, F. F.; CHANG, T. C. **Computing Average Run Lengths of Exponential EWMA Charts**. Journal of Quality Technology, v. 32, n. 2, p. 183-187, 2000.

GAN, F. F.; CHOI, K. P. **Computing Average Run Lengths for Exponential CUSUM Schemes**. Journal of Quality Technology, v. 26, n. 2, p. 134-143, 1994.

LUCAS, James M. **Counted Data CUSUM's**. Technometrics, v. 27, n. 2, p. 129-144, 1985.

LUCAS, James M. **Control Schemes for Low Count Levels**. Journal of Quality Technology, v. 21, n. 3, p. 199-201, 1989.

LUCAS, James M.; SACCUCCI, Michael S. **Exponentially Weighted Moving Average Control Schemes: Properties and Enhancements**. Technometrics, v. 32, n. 1, p. 1-12, 1990.

McCOOL, John I.; JOYNER-MOTLEY, Tracy. **Control Charts Applicable When the Fraction Nonconforming is Small**. Journal of Quality Technology, v. 30, n. 3, p. 240-247, 1998.

NELSON, Lloyd S. **A Control Chart for Parts-Per-Million Nonconforming Items**. Journal of Quality Technology, v. 26, n. 3, p. 239-240, 1994.

NELSON, Lloyd S. **Supplementary Runs Tests for *np* Control Charts**. Journal of Quality Technology, v. 29, n. 2, p. 225-227, 1997.

PRABHU, S. S.; MONTGOMERY, D.C.; RUNGER, G. C. **A Combined Adaptive Sample Size and Sampling Interval $\overline{X}$ Control Scheme**. Journal of Quality Technology, v. 26, p. 164-176, 1994.

RENDTEL, U. **CUSUM-Schemes With Variable Sampling Intervals and Sample Sizes**. Statistical Papers, v. 31, p. 103-118, 1990.

REYNOLDS JR., Marion R. **Optimal Variable Sampling Interval Control Charts**. Sequential Analysis, v. 8, p. 361-379, 1989.

REYNOLDS JR., Marion R. **Evaluating Properties of Variable Sampling Interval Control Charts**. Sequential Analysis, v. 14, p. 59-97, 1995.

REYNOLDS, M. R.; AMIN, R. W.; ARNOLD, J. C.; NACHLAS, J. A. **$\bar{X}$ Charts with Variable Sampling Intervals**. Technometrics, v. 30, p. 181-192, 1988.

REYNOLDS, Marion R.; ARNOLD, Jesse C. **EWMA Control Charts With Variable Sample Sizes and Variable Sampling Intervals**. IIE Transactions, v. 33, n. 6, p. 511-530, 2001.

REYNOLDS JR., Marion R.; STOUMBOS, Zachary G. **A CUSUM Chart for Monitoring a Proportion When Inspecting Continuously**. Journal of Quality Technology, v. 31, n. 1, p. 87-108, 1999.

RUNGER, G. C.; MONTGOMERY, D. C. **Adaptive Sampling Enhancements for Shewhart Control Charts**. IIE Transactions, v. 25, p. 41-51, 1993.

RUNGER, G. C.; PIGNATIELLO JR., J. J. **Adaptive Sampling for Process Control**. Journal of Quality Technology, v. 23, p. 135-155, 1991.

RYAN, T. P.; SCHWERTMAN, N. C. **Optimal Limits for Attributes Control Charts**. Journal of Quality Technology, v. 29, n. 1, p. 86-98, 1997.

SACCUCCI, Michael S.; AMIN, Raid W.; LUCAS, James M. **Exponentially Weighted Moving Average Control Schemes With Variable Sampling Intervals**. Communications in Statistics – Simulation and Computation, v. 21, n. 3, p. 627-657, 1992.

SACCUCCI, Michael S.; LUCAS, James M. **Average Run Lenghts for Exponentially Weighted Moving Average Control Schemes Using the Markov Chain Approach**. Journal of Quality Technology, v. 22, n. 2, p. 154-162, 1990.

TAGARAS, George. **A Survey of Recent Developments in the Design of Adaptive Control Charts**. Journal of Quality Technology, v. 30, n. 3, p. 212-231, 1998.

VARDEMAN, Stephen; RAY, Di-ou. **Average Run Lengths for CUSUM Schemes When Observations Are Exponentially Distributed**. Technometrics, v. 27, n. 2, p. 145-150, 1985.

VAUGHAN, Timothy S. **Variable Sampling Interval *np* Process Control Chart**. Communications in Statistics – Theory and Methods, v. 22, n. 1, p. 147-167, 1993.

WHITE, Carolyn H.; KEATS, J. Bert. **ARLs and Higher-Order Run-Length Moments for the Poisson CUSUM**. Journal of Quality Technology, v. 28, n. 3, p. 363-369, 1996.

WHITE, Carolyn H.; KEATS, J. Bert; STANLEY, James. **Poisson CUSUM Versus *c* Chart for Defect Data**. Quality Engineering, v. 9, n. 4, p. 673-679, 1997.

WOODALL, William H. **Control Charts Based on Attribute Data: Bibliography and Review**. Journal of Quality Technology, v. 29, n. 2, p. 172-183, 1997.

WU, Zhang; YEO, Song Huat. **Implementing Synthetic Control Charts for Attributes**. Journal of Quality Technology, v. 33, n. 1, p. 112-114, 2001.

WU, Zhang; YEO, Song Huat; SPEDDING, Trevor A. **A Synthetic Control Chart for Detecting Fraction Nonconforming Increases**. Journal of Quality Technology, v. 33, n. 1, p. 104-111, 2001.

ZIMMER, L. S.; MONTGOMERY, D. C.; RUNGER, G. C. **Three-state Sample Size $\overline{X}$ Chart**. International Journal of Production Research, v. 36, p. 733-743, 1998.

8
Apêndice A:
Programa para Obtenção de Projeto Ótimo de Esquema Vp

O programa em linguagem C, mostrado a seguir, fornece, para c_0 = 0,5, 1,0, 1,5, 2,0, 3,0, e 4,0, para h_S = 0,10, 0,25 e 0,50, e para γ^* = 1,5, 2,0 e 3,0, o projeto ótimo de esquemas Vp, com obtenção das medidas de desempenho, em cada caso.

```
/*************************************************************/
/*                                                           */
/*  Programa para projeto ótimo de esquemas Vp               */
/*                                                           */
/*************************************************************/

#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <math.h>
#include <stdarg.h>
#include <assert.h>



/*************************************************************/
/*  Funções auxiliares                                       */
/*************************************************************/

double FatTable[20];
static FILE* fout = NULL;

/*
Calcula o fatorial de n.  Utiliza uma tabela de constantes
pré-calculada para números menores que 20.
*/
double fat(int n)
{
  double val;
```

```
  if(n<20)
    return FatTable[n];

  val = FatTable[19];
  for(int i=20; i<=n; i++)
    val = val * i;
  return val;
}

/*
Inicializa a tabela de fatoriais pré-calculados
*/
void init_fat(void)
{
  int i;

  FatTable[0] = 1;
  for(i=1; i<20; i++)
    FatTable[i] = i * FatTable[i-1];
}

/*
Retorna a probabilidade de uma variável aleatória de Poisson
assumir um valor menor ou igual a "k".

k - Argumento para o qual a distribuição de Poisson será
    avaliada.

theta - Media da distribuição de Poisson
*/
double dpoidf(int k, double theta)
{
  int i;
  double val = 0.0;

  for(i=0; i<=k; i++)
    val += (exp(-theta)*pow(theta,i)) / fat(i);

  return val;
}
```

```
/*
Imprime os parâmetros recebidos no arquivo de saída e na tela
*/
void print(char* format, ...)
{
  va_list list;

  if(!fout)
  {
    fout = fopen("resultado_vp.txt", "w");
    if(!fout)
    {
      printf("Erro abrindo arquivo de saida\n");
      exit(-1);
    }
  }

  va_start(list, format);
  vfprintf(fout, format, list);
  fflush(fout);
  vprintf(format, list);
  va_end(list);
}



/*************************************************************/
/*  Funções de cálculo                                       */
/*************************************************************/

#define P(a,b) p[(a)-1][(b)-1]



/*
Calcula as probabilidades de transição
*/
void calprob(double lsc1, double lsa1, double lsc2, double
             lsa2,  double mS, double mL, double c0,
             double*r1,
             double p[3][3])
{
  double theta, a, b, c;
```

```
  int k;

  theta  = mS * c0;
  k      = (int)floor(lsa1);
  P(1,1) = dpoidf(k,theta);

  k      = (int)floor(lsc1);
  P(1,2) = dpoidf(k,theta) - P(1,1);

  theta  = mL * c0;
  k      = (int)floor(lsa2);
  P(2,1) = dpoidf(k,theta);
  k      = (int)floor(lsc2);
  P(2,2) = dpoidf(k,theta) - P(2,1);

  P(1,3) = 1 - P(1,1) - P(1,2);
  P(2,3) = 1 - P(2,1) - P(2,2);
  a      = P(2,3) - P(1,3);
  b      = P(1,2) + P(2,1) + P(1,3) - P(2,3);
  c      = -P(2,1);

  assert(a);

  *r1 = (-b + sqrt(b*b-4*a*c))/(2*a);
}


/*
Calcula o valor de TMAF
*/
void tmaf(double p[3][3], double r1, double hL, double hS,
          double* vtmaf)
{
  double den, num;

  den = (1-P(1,1))*(1-P(2,2))-P(1,2)*P(2,1);
  num = ((1-P(2,2))*hL+P(1,2)*hS)*r1;
  num = num + (P(2,1)*hL+(1-P(1,1))*hS)*(1-r1);

  assert(den);
```

```
  *vtmaf = num/den;
}

/*
Calcula o valor de TES
*/
void tes(double lsc1, double lsa1, double lsc2, double lsa2,
         double hL, double hS, double mS, double mL,
         double c1, double r1, double* vtes)
{
  double p[3][3];
  double r2, s, s1, s2, num, den, theta;
  int k;

  theta  = mS * c1;
  k      = (int)floor(lsa1);
  P(1,1) = dpoidf(k,theta);
  k      = (int)floor(lsc1);
  P(1,2) = dpoidf(k,theta)-P(1,1);

  theta  = mL*c1;
  k      = (int)floor(lsa2);
  P(2,1) = dpoidf(k,theta);
  k      = (int)floor(lsc2);
  P(2,2) = dpoidf(k,theta)-P(2,1);
  r2     = 1-r1;
  s      = r1*hL+r2*hS;
  s1     = r1*hL/s;
  s2     = r2*hS/s;
  den    = (1-P(1,1))*(1-P(2,2))-P(1,2)*P(2,1);
  num    = ((1-P(2,2))*hL+P(1,2)*hS)*s1 +
           (P(2,1)*hL+(1-P(1,1))*hS)*s2;

  assert(den);

  *vtes = num/den-(s1*hL+s2*hS)/2.0;
}


/*
Efetua os cálculos de TMAF e TES para os parâmetros correntes
```

```
*/
int calcula(double c0, double gama, double lsc1, double lsa1,
            double lsc2,   double lsa2, double mS, double mL,
            double hS, double* hL, double* vtes, double*
            vtmaf, double tmaffp)
{
  double p[3][3];
  double r1, mbarra;
  double c1 = c0 * gama;

  calprob(lsc1,lsa1,lsc2,lsa2,mS,mL,c0,&r1,p);

  assert(r1);

 mbarra = r1*mS+(1-r1)*mL;
  if (mbarra > 1.0)
    return 0;

  *hL=(1.0-(1.0-r1)*hS)/r1;

  tmaf(p,r1,*hL,hS,vtmaf);

  if (*vtmaf < tmaffp)
    return 0;

  tes(lsc1,lsa1,lsc2,lsa2,*hL,hS,mS,mL,c1,r1,vtes);
  return 1;
}


/*
Rotina principal. Contém os loops para otimização.
*/
void main(void)
{
  time_t t;

  double c0, gama, lsc1, lsa1, lsc2, lsa2, mS, mL,
         hL, hS, vtes, vtmaf;

  double lscfp, alfa, tmaffp, beta, tesfp;
```

```
  double tes_minimo;

  int k, lsc1i, lsc1f, lsc2i, lsc2f;

  /* Melhores resultados */
  double blsc1, blsa1, blsc2, blsa2, bmS, bmL,
         bhL, bhS, btes, btmaf;


/*********************************************************/
/*  Parâmetros de entrada -- ALTERAR SOMENTE AQUI        */
/*********************************************************/

  /* Valores para c0, hS e gama */
  double c0_vet[]   = {0.5, 1.0, 1.5, 2.0, 3.0, 4.0},
         hS_vet[]   = {0.1, 0.25, 0.5},
         gama_vet[] = {1.5, 2.0, 3.0};
  /* Discretização de mS */
  double mS_inicial = 0.1;
  double mS_final   = 0.9;
  double mS_inc     = 0.01;
  /* Discretização de mL */
  double mL_inicial = 1.05;
  double mL_final   = 5.00;
  double mL_inc     = 0.01;


/*********************************************************/
/*  FIM Parâmetros de entrada -- NAO ALTERAR MAIS        */
/*********************************************************/

  int num_c0   = sizeof(c0_vet)/sizeof(double);
  int num_hS   = sizeof(hS_vet)/sizeof(double);
  int num_gama = sizeof(gama_vet)/sizeof(double);
  int num_mS   = 1 + (int)((mS_final - mS_inicial) / mS_inc);
  int num_mL   = 1 + (int)((mL_final - mL_inicial) / mL_inc);


  t = time(NULL);
  print("Inicio: %s\n\n", ctime(&t));
```

```
init_fat();

for(int i_c0=0; i_c0 < num_c0; i_c0++) // c0
{
  print("  C0  GAMA*  LSC1  LSA1  LSC2  LSA2  MS"
        "  ML   HS     HL   TES   TMAF\n");
  print("----  ----  -----  ------  ------  ------  ------"
        "  ------  ------  ------  ------  ------\n");

  c0 = c0_vet[i_c0];

  for(int i_hS=0; i_hS<num_hS; i_hS++) // hS
  {
    hS = hS_vet[i_hS];

    for(int i_gama=0; i_gama<num_gama; i_gama++) // gama
    {
      gama = gama_vet[i_gama];

      lscfp  = 0.6195 + 1.00523*c0 + 2.983*sqrt(c0);
      k      = (int)floor(lscfp);
      alfa   = 1.0 - dpoidf(k, c0);
      tmaffp  = 1.0 / alfa;
      beta   = dpoidf(k, c0*gama);
      tesfp = 1.0 / (1-beta) - 0.5;

      tes_minimo = tesfp;

      for(int i_mS=0; i_mS<num_mS; i_mS++) // mS
      {
        mS = mS_inicial + i_mS*mS_inc;

        lsc1i =
             (int)(0.6195+1.00523*mS*c0+2.983*sqrt(mS*c0));
        lsc1f = lsc1i+2;

        for(int i_lsc1=lsc1i; i_lsc1<=lsc1f; i_lsc1++) //
          lsc1
        {
          lsc1 = i_lsc1 + 0.5;
```

```
for(int i_lsa1=0; i_lsa1<i_lsc1; i_lsa1++) //
  lsa1
{
  lsa1 = i_lsa1 + 0.5;

  for(int i_mL=0; i_mL<num_mL; i_mL++) // mL
  {
    mL = mL_inicial + i_mL*mL_inc;

    lsc2f = (int)(0.6195+1.00523*mL*c0+2.983 *
                sqrt(mL*c0));
    lsc2i = (int)((0.6195+1.00523*mL*c0+2.983 *
                sqrt(mL*c0))-c0);

    for(int i_lsc2=lsc2i; i_lsc2<=lsc2f;
      i_lsc2++)
    {
      lsc2 = i_lsc2 + 0.5;

      for(int i_lsa2=0; i_lsa2<i_lsc2;
      i_lsa2++)//lsa2
      {
      lsa2 = i_lsa2 + 0.5;

        if(!calcula(c0, gama, lsc1, lsa1, lsc2,
           lsa2, mS, mL, hS, &hL, &vtes, &vtmaf,
           tmaffp))
          continue;

       // Se resultado obtido melhor que o mínimo
       // até agora, armazena o resultado
        if(vtes < tes_minimo)
        {
          blsc1  = lsc1;
          blsc2  = lsc2;
          blsa1  = lsa1;
          blsa2  = lsa2;
          bhL    = hL;
          bhS    = hS;
          bmS    = mS;
          bmL    = mL;
```

```
                        btes   = vtes;
                        btmaf  = vtmaf;
                        tes_minimo = vtes;
                      }
                    } // lsa2
                  } // lsc2
                } // mL
              } // lsa1
            } // lsc1
          } // mS

          // Imprime o melhor resultado encontrado para estes
          // valores de c0, hS e gama
          print("%5.2f %6.2f %7.2f %7.2f %7.2f %7.2f %7.3f
                 %7.3f %7.3f %7.3f %7.2f %7.1f\n",
                c0, gama, blsc1, blsa1, blsc2, blsa2, bmS, bmL,
                bhS, bhL, btes, btmaf);

        }  // gama
        print("\n");
      } // hS
      print("\n");
    } // c0

    t = time(NULL);
    print("\n\nFim: %s\n\n", ctime(&t));
    fclose(fout);
}
```

9
Apêndice B:
Programa para Simulação dos Tempos até o Sinal obtidos pelo Esquema VSI EWMA

O programa em linguagem C, mostrado a seguir, implementa as simulações para obtenção da média e do desvio-padrão do tempo até o sinal, usando como dados de entrada os valores de c_0, c_1, λ, LSA, LSC, h_S e h_L de esquemas VSI EWMA.


```
#include <assert.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "random.h"

long  SimulationSeed = 0;
long  N1;

     /*********************************************************/
     /*  Parâmetros de entrada -- ALTERAR SOMENTE AQUI        */
     /*********************************************************/


long   RUNS   = 100000;

double LAMBDA = 0.80;
long   N2     = 10000;

double C0  = 2.00;
double C1  = 3.00;    /* gama = C1/C0 */

double LSA = 2.08;
double LSC = 5.79;

double HL  = 1.345;
double HS  = 0.50;

     /*********************************************************/
     /*  FIM Parâmetros de entrada -- NAO ALTERAR MAIS        */
     /*********************************************************/
```

```
double CalculaTs(double* Z, double* h)
{
  long   i, j;
  long   c;
  double Tsf = 0.0;
  double Q;
  int    serie_invalida = 1;

  /* Gerar N1 valores de Ci, onde C ~ Poisson(C0) e calcular Zi;
     i = 1, .., N1; se algum Zi > LSC, descarta a série
  */
  Z[0] = C0;
  while(serie_invalida)
  {
    serie_invalida = 0;
    for(i=1; i<=N1; i++)
    {
      c = RandomPoisson(C0);
      Z[i] = (1.0-LAMBDA)*Z[i-1] + LAMBDA*c;
      if(Z[i] > LSC)
        serie_invalida = 1;
    }
  }

  /* Cálculo de h[1..N1] */
  for(j=1; j<=N1; j++)
  {
    if (Z[j-1] <= LSA)
      h[j] = HL;
    else
      h[j] = HS;
  }

  /* Gerar valores de Ci agora com Ci ~ Poisson (C1) e calcular
     Zi; i = N1+1 até sinal; até ocorrência de um sinal (ou até
     gerar N2 valores); Calcula também os valores para hi
     correspondentes
  */
  for(j=N1+1; j<=N1+N2; j++)
  {
    c = RandomPoisson(C1);
    Z[j] = (1.0-LAMBDA)*Z[j-1] + LAMBDA*c;
    if(Z[j-1] <= LSA)
      h[j] = HL;
    else if(Z[j-1] < LSC)
      h[j] = HS;
    else
    {
      h[j] = -1; /* SINAL */
      break;
    }
  }
  printf("%d - ", j-N1);
  assert(j<=N1+N2); /* Passou pelos N2 elementos sem sinal */

  /* Cálculo de TSf*/
  for(j=N1+1; h[j]!=-1; j++)
    Tsf += h[j];

  /* Calcula Ts */
  Q = RandomUniform() * h[N1+1];
```

```
  return Tsf - Q;
}

void RunSimulation(void)
{
  double* Z;
  double* h;
  double* Ts;
  long    i;
  double  mean;
  double  stddev;

  N1 = (long)ceil(log(0.01)/(2*log(1.0-LAMBDA)));

  /* Aloca vetores (vetores com um elemento a mais para serem
     indexados a partir de 1 e não de 0)
  */
  Z = (double*)malloc((N1+N2+1)*sizeof(double));
  assert(Z);

  h = (double*)malloc((N1+N2+1)*sizeof(double));
  assert(h);

  Ts = (double*)malloc((RUNS+1)*sizeof(double));
  assert(Ts);

  /* Loop de simulação para cálculo de TS */
  for(i=1; i<=RUNS; i++)
  {
    Ts[i] = CalculaTs(Z, h);
    printf("Ts[%d] = %g\n", i, Ts[i]);
  }

  /* Calcula média e desvio-padrão dos TS */
  mean = 0.0;
  for(i=1; i<=RUNS; i++)
    mean += Ts[i];
  mean = mean / RUNS;

  stddev = 0.0;
  for(i=1; i<=RUNS; i++)
    stddev += (Ts[i]-mean)*(Ts[i]-mean);
  stddev = sqrt(stddev / RUNS);

  printf("N1 = %d\n",N1);
  printf("Média = %g\n", mean);
  printf("Desvio-padrão = %g\n", stddev);

  /* Libera memória */
  free(Z);
  free(h);
  free(Ts);
}


int main(int argc, char* argv[])
{
  time_t  t;

  RandomInit(SimulationSeed);
```

```
  t = time(NULL);
  printf("Início da Simulação: %s\n", ctime(&t));

  RunSimulation();

  t = time(NULL);
  printf("Final da Simulação: %s\n\n", ctime(&t));

  system("pause");
  return 0;
}
```