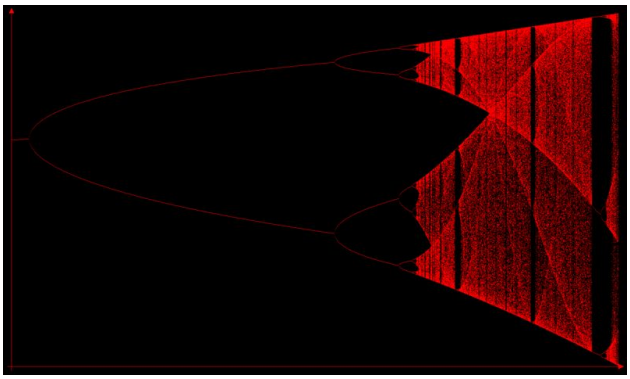
# ISD
# Relatório Mapa Logístico
# Grupo 6

Sérgio Viana (49402) Nuno Machado (49408) André Santos (49344)

10 de Julho de 2012

**Resumo**

Relatório relativo ao trabalho de grupo elaborado no âmbito da cadeira de Introdução aos Sistemas Dinâmicos do 2º ano da Licenciatura em Engenharia Informática da Universidade do Minho. Tem como objectivo explicar o conceito de Mapa Logístico, fazer uma síntese do que foi o nosso trabalho, descrever o programa que concebemos e apresentar alguns resultados de vários testes feitos.



## Conteúdo

# 1 Introdução

## 1.1 Equação Logística

A equação logística, ou modelo de Verhulst é um modelo do crescimento populacional com recursos limitados, formulado pela primeira vez por Pierre Verhulst (1845, 1847), e é dado por:

$$\frac{\partial x}{\partial t} = rx(1-x)$$

onde $x$ é a população para o instante $t$ e $r$ é o ritmo máximo de crescimento da população.
Este tipo de função cresce rapidamente no início , mas a taxa de crescimento tende a diminuir e acaba por ser zero.
Pode descrever vários tipos de situações, como por exemplo o número de indivíduos de uma dada espécie que dispõe de recursos limitados, ou a difusão de informação numa sociedade, mas também se aplica em áreas como a economia, podendo representar a venda de novos produtos ao longo do tempo ou a curva da procura em função do preço crescente.

## 1.2 Mapa Logístico

A versão discreta da equação logística é conhecida como mapa logístico. Foi apresentada pelo biólogo Robert May em 1976, e tem a seguinte forma:

$$x_{n+1} = rx_n(1-x_n)$$

em que

- $x_n$ representa a população na unidade de tempo $n$, e portanto $x_0$ representa a população inicial;
- $r$ é um número positivo e representa o crecimento da população.

Esta equação traduz dois efeitos da dinâmica populacional:

- natalidade, onde enquanto a população for pequena vai crescer a um ritmo porporcional ao seu tamanho;
- mortalidade causada pela escassez de alimento, que se manifesta à medida que a população aumenta e esgota os recursos do ecossistema.

Variando o valor de $r$, observa-se o seguinte comportamento:

- Com $r$ entre 0 e 1, a população acabará por morrer, qualquer que seja a população inicial;
- Com $r$ entre 1 e 2, a população rapidamente estabilizará no valor $\frac{r-1}{r}$;
- Com $r$ entre 2 e 3, a população também tende para $\frac{r-1}{r}$, mas não tão rapidamente;
- Com $r$ entre 3 e $1+\sqrt{6}$ a população oscilará entre dois valores para sempre;
- Com $r$ entre $3,45$ e $3,54$ (aprox.) oscilará entre quatro valores;
- Com $r$ ligeiramente acima de 3,54, a equação tenderá para 8 valores, depois 16, 32, etc. A proporção entre o comprimento de dois intervalos sucessivos entre bifurcações aproxima-se da constante de Feigenbaum $\delta = 4,669...$
- Com $r$ aproximadamente igual a $3,57$ começa o caos. Já não são observáveis quaisquer oscilações, e pequenas variações na população inicial provocam resultados completamente divergentes;
- Apesar de a maior parte dos valores acima de $3,57$ exibir um comportamento caótico, certos valores isolados não o fazem. Por exemplo, a partir de $1+\sqrt{8}$ existe uma gama de parâmetros $r$ que oscila entre 3 valores, depois entre 6, 12, etc. Existem outros intervalos que provocam oscilações entre 5 valores, 7, 11, etc; todos os períodos de oscilação ocorrem;
- Para valores de $r$ acima de 4, os valores para os quais a função tende deixam o intervalo $[0,1]$ e apresentam praticamente todos um comportamento caótico.

### 1.3 Diagrama de Bifurcação

Um diagrama de bifurcação mostra os sucessivos valores para os quais o mapa logístico tende em função de $r$. No eixo vertical estão representados os possíveis valores para os quais a função tende, e no eixo horizontal está representado $r$.

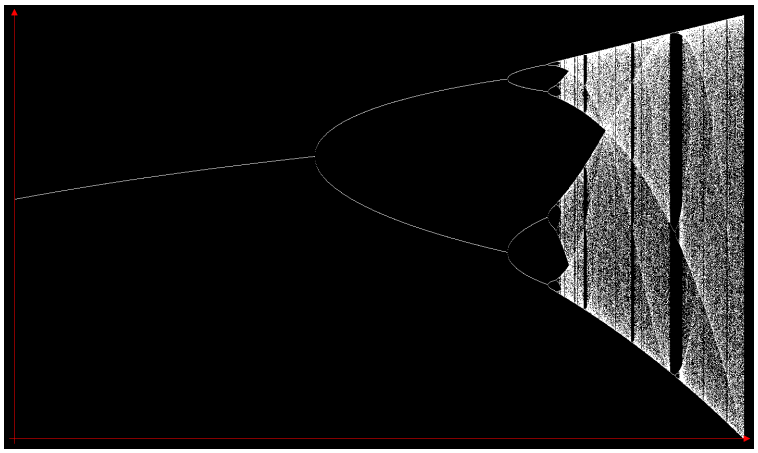


Figura 1: Diagrama de Bifurcação do Mapa Logístico

## 2 Implementação do Problema

O objectivo do nosso trabalho era desenvolver um programa que, a partir do mapa logístico ou uma equação similar, produzisse o respectivo diagrama de bifurcação. A linguagem usada foi `C`, com recurso à biblioteca gráfica OpenGL[1].
O nosso programa percorre uma gama de valores de $r$, através de pequenos incrementos, desenhando os pontos respeitantes aos valores para os quais a função tendia. Assim, para cada valor de $r$:

1. Partindo de um valor $x_0$, a função é aplicada um número elevado de vezes (por exemplo, 50000), sem guardar qualquer resultado ou desenhar qualquer ponto;
2. Depois, são realizadas mais algumas iterações (um valor típico seria 200) em que, em cada iteração é guardado o valor resultante. Este passo e o anterior garantem que os pontos que serão desenhados são efectivamente pontos para os quais a função tende, ou que, pelo menos, se aproximam deles;
3. Após realizadas todas as iterações necesárias para um só valor de $r$, os pontos resultantes são desenhados;
4. De seguida, incrementa-se $r$ e repete-se tudo.

Por razões relacionadas com o *debugging* do programa, cada vez que a função é desenhada é impresso na consola o valor de algumas das variáveis usadas. Esta funcionalidade foi mantida por a termos considerado útil. Fica aqui portanto a legenda das variáveis da consola:

```
LOGISTIC MAP v2.01      2007.12.16

r_min:          2.300000        /* Valores minimo e maximo de r     */
r_max:          20.000000       /* usados nos calculos e mostrados  */
```

[1] [Em Ubuntu] é necessário estarem instalados os pacotes freeglut3, freeglut3-dev, glutg3, libglut3, libglut3-dev, libxmu-dev, libxmu-headers, libxmu6, libxmuu-dev, libxmuu1.

```
x_min:          0.000000        /* no grafico, e valores minimo e   */
x_max:          3.000000        /* maximo de x mostrados no grafico */

offset:         50000           // Iteracoes realizadas antes de desenhar os pontos
num_pontos:     200             // Pontos calculados para cada valor de r

wgl:            1440            // Largura (em pixeis) da janela do grafico
hgl:            852             // Altura (em pixeis) da janela do grafico
```

Ao programa foram ainda acrescentadas as possibilidades de fazer sucessivos zoom-ins, ou de voltar ao zoom inicial. O programa foi testado para as funções $x_{n+1} = rx_n(1 - x_n)$, $x_{n+1} = r \sin x_n(1 - \sin x_n)$, $x_{n+1} = r{x_n}^2(1 - {x_n}^2)$, $x_{n+1} = 4 \sin x_n(1 - x_n)$.

# 3 Testes

## 3.1 $x_{n+1} = rx_n(1 - x_n)$

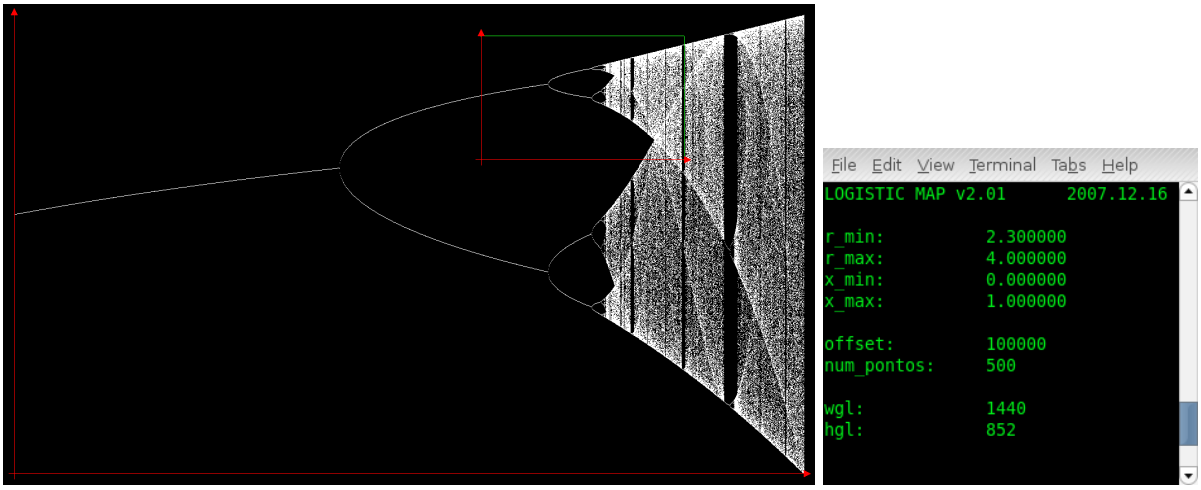



Figura 2: Diagrama de Bifurcação de $x_{n+1} = rx_n(1 - x_n)$ e output da consola

## 3.2 $x_{n+1} = r \sin x_n(1 - \sin x_n)$

## 3.3 $x_{n+1} = r{x_n}^2(1 - {x_n}^2)$

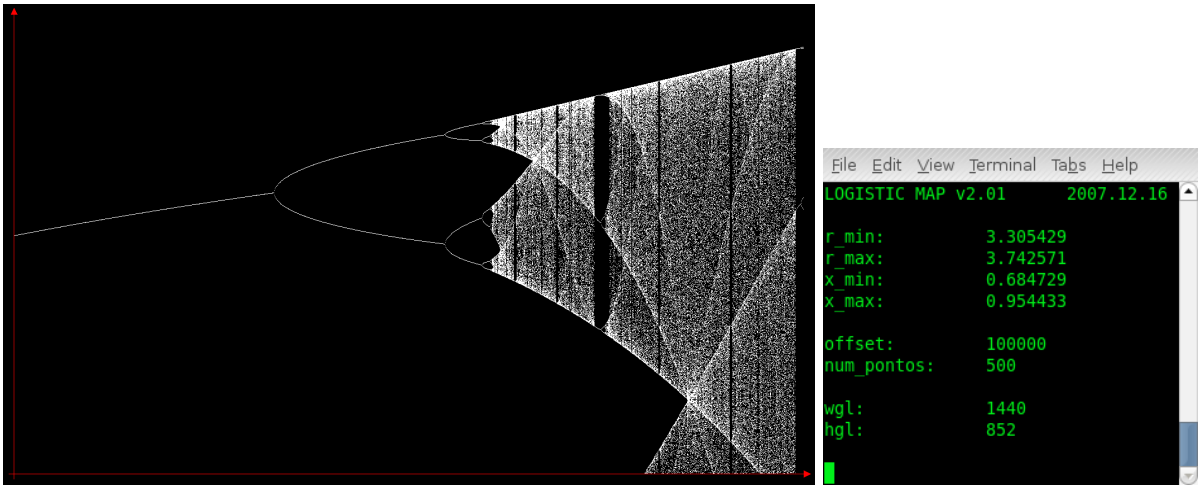



Figura 3: Zoom da secção assinalada na figura anterior

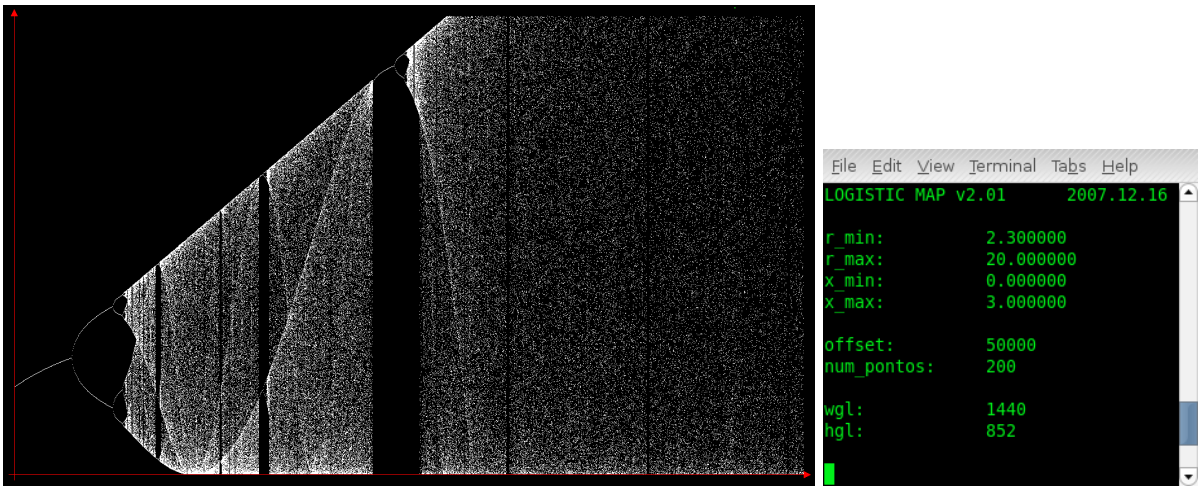



Figura 4: Diagrama de Bifurcação de $x_{n+1} = r \sin x_n (1 - \sin x_n)$ e output da consola

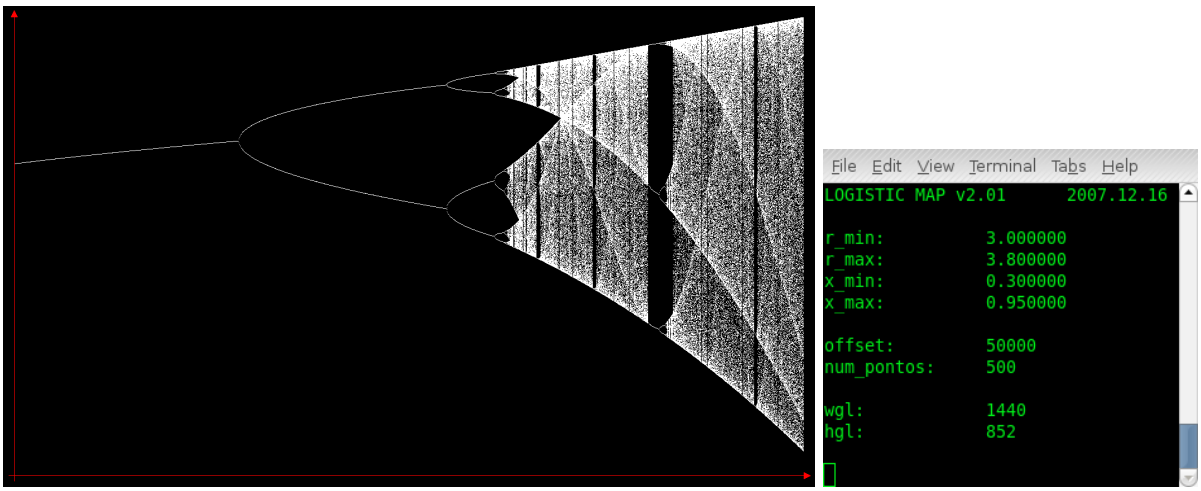



Figura 5: Diagrama de Bifurcação de $x_{n+1} = r{x_n}^2(1 - {x_n}^2)$ e output da consola

### 3.4 $x_{n+1} = 4 \sin r x_n (1 - x_n)$

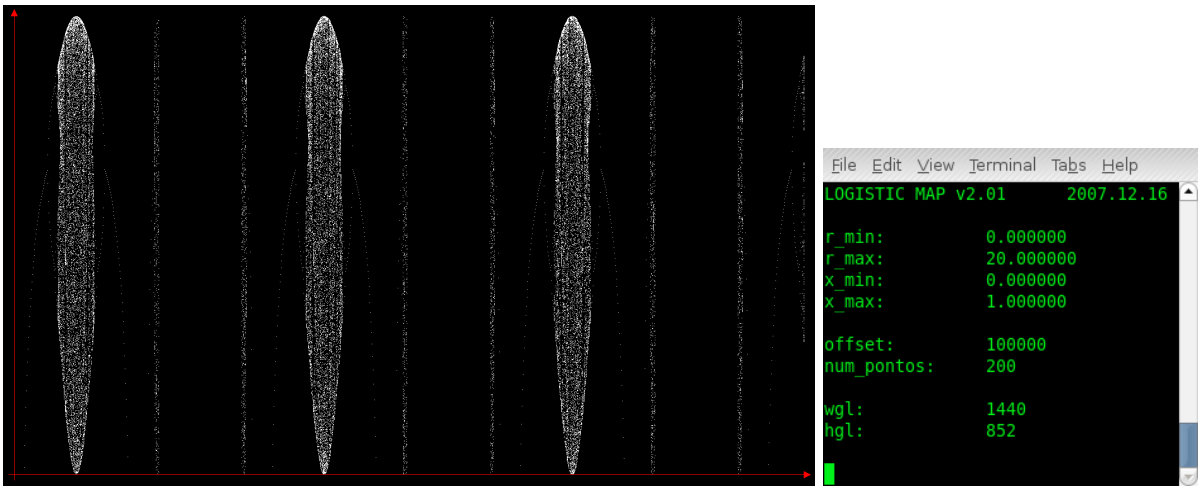



Figura 6: Diagrama de Bifurcação de $x_{n+1} = 4 \sin r x_n (1 - x_n)$ e output da consola

## 4 Conclusão

No comportamento do mapa logístico é irrelevante a população inicial. Aplicando recursivamente uma função, observando os pontos para onde ela tende e fazendo variar um determinado parâmetro, é possível traçar um diagrama de bifurcação dessa função. O diagrama de bifurcação é um exemplo de um fractal, porque tem sucessivas bifurcações. As funções testadas apresentam um comportâmento que, não sendo igual, apresenta bastantes semelhanças.

## 5 Fontes Consultadas


http://en.wikipedia.org/wiki/Logistic_map
http://en.wikipedia.org/wiki/Logistic_function
http://en.wikipedia.org/wiki/Pierre_Fran%C3%A7ois_Verhulst
http://en.wikipedia.org/wiki/Feigenbaum_constant
http://w3.ualg.pt/~lnunes/Pessoal/Disciplinas/Modelacao-texto.htm#5.%20Sistemas%20din%E2micos%20e%20caos
http://www.cs.xu.edu/math/math120/01f/logistic.pdf
http://mathworld.wolfram.com/LogisticMap.html
http://mathworld.wolfram.com/LogisticEquation.html


## A Código

```
////////////////////////////////////////////////////////////////////////////////
// Logistic Map v2.01.c //
// 2007.12.16 //
// Desenha cada conjunto de pontos para cada r de uma vez,//
// permite redimensionamento da janela, diferentes funçoes,//
// Inclui eixos a 100%, faz zoom direitinho, falta a numeracao //
// nos eixos. Faz ouput das variaveis na consola //
// Nuno Machado, 49408 //
```

```
// Sérgio Viana, 49402 //
// Andre Santos, 49344 //
////////////////////////////////////////////////////////////////////////////
#include <GL/glut.h>
#include <GL/gl.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define BORDER 20

/////////////////////////////////////////////////////////
//variaveis globais
int n;
int  i=0;
long int  offset = 50000;
double r_min = 0.0;
double r_max = 4.0;
double x_min = 0.0;
double x_max = 1.0;
int  num_pontos = 200;
double  x0 = 0.5;
double  r = 0.0;
double  x = 0.5;
int  wgl = 1434;
int  hgl = 852;
int  mouse_presses = 0;
double zoom_r;
double zoom_x;
int  jumps = 800;
double  razao_r;
double razao_x;
double margem_r;
double margem_x;


void output(GLfloat x, GLfloat y, char *text)
{
    char *p;

    glPushMatrix();
    glTranslatef(x, y, 0);
    for (p = text; *p; p++)
        glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
    glPopMatrix();
}

/////////////////////////////////////////////////////////
//Funcoes Calc
double calc_0(){
return (r*x*(1-x));
}

double calc_1(){
```

```
return (r * sin(x) * (1 - sin(x)));
}

double calc_2(){
return (r * pow(x,2) * (1 - pow(x,2)));
}

double calc_3(){
return (4*sin(r) * x *(1-x));
}

double (*calc[10])() = {calc_0,calc_1,calc_2,calc_3};

////////////////////////////////////////////////////////
//Funcoes set_glbvars (inicializam as variáveis
//com os melhores valores para cada função calc)
int set_glbvars_0(){
i = 0;
offset = 100000;
r_min = 2.3;
r_max = 4.0;
x_min = 0.0;
x_max = 1.0;
num_pontos = 500;
x0 = 0.5;
r = 0.0;
x = 0.5;
mouse_presses = 0;
jumps = (int) (wgl+100)/(r_max-r_min+2*margem_r);

razao_r = (r_max-r_min)/(wgl-2*BORDER);
razao_x = (x_max-x_min)/(hgl-2*BORDER);
margem_r = BORDER*razao_r;
margem_x = BORDER*razao_x;
return 1;
}

int set_glbvars_1(){
i = 0;
offset = 50000;
r_min = 2.3;
r_max = 20.0;
x_min = 0.0;
x_max = 3.0;
num_pontos = 200;
x0 = 0.5;
r = r_min;
x = x0;
mouse_presses = 0;
jumps = (int) (wgl+100)/(r_max-r_min+2*margem_r);

razao_r = (r_max-r_min)/(wgl-2*BORDER);
razao_x = (x_max-x_min)/(hgl-2*BORDER);
margem_r = BORDER*razao_r;
```

```
margem_x = BORDER*razao_x;
return 1;
}

int set_glbvars_2(){
i = 0;
offset = 50000;
r_min = 3.0;
r_max = 3.8;
x_min = 0.3;
x_max = .95;
num_pontos = 500;
x0 = 0.5;
r = r_min;
x = x0;
mouse_presses = 0;
jumps = (int) (wgl+100)/(r_max-r_min+2*margem_r);

razao_r = (r_max-r_min)/(wgl-2*BORDER);
razao_x = (x_max-x_min)/(hgl-2*BORDER);
margem_r = BORDER*razao_r;
margem_x = BORDER*razao_x;
return 1;
}

int set_glbvars_3(){
i = 0;
offset = 100000;
r_min = 0.0;
r_max = 20.0;
x_min = 0.0;
x_max = 1.0;
num_pontos = 200;
x0 = 0.5;
r = 0.0;
x = 0.5;
mouse_presses = 0;
jumps = (int) (wgl+100)/(r_max-r_min+2*margem_r);

razao_r = (r_max-r_min)/(wgl-2*BORDER);
razao_x = (x_max-x_min)/(hgl-2*BORDER);
margem_r = BORDER*razao_r;
margem_x = BORDER*razao_x;
return 1;
}

int (*set_glbvars[10])() = {set_glbvars_0,set_glbvars_1,set_glbvars_2,set_glbvars_3};

////////////////////////////////////////////////
//Funcao que imprime na consola as
//variaveis usadas para o desenho
int print_vars(){
system("clear");
printf("LOGISTIC MAP v2.01\t2007.12.16\n\n");
```

```
printf("r_min:\t\t%f\n",r_min);
printf("r_max:\t\t%f\n",r_max);
printf("x_min:\t\t%f\n",x_min);
printf("x_max:\t\t%f\n\n",x_max);
printf("offset:\t\t%ld\n",offset);
printf("num_pontos:\t%d\n\n",num_pontos);
printf("wgl:\t\t%d\n",wgl);
printf("hgl:\t\t%d\n\n",hgl);
return 0;
}

//////////////////////////////////////////
//Funcão que desenha os eixos
int draw_eixos(){
//Cor vermelho
glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_LINES);

//Eixo RR
glVertex2f(r_min-margem_r/2,x_min);
glVertex2f(r_max+margem_r/3, x_min);
//Eixo XX
glVertex2f(r_min,x_min-margem_x/2);
glVertex2f(r_min,x_max+margem_x/3);
glEnd();
//Seta RR
glBegin(GL_TRIANGLES);
glVertex2f(r_max, x_min + margem_x/3);
glVertex2f(r_max, x_min -margem_x/3);
glVertex2f(r_max+ 2*margem_r/3, x_min);
//Seta XX
glColor3f(1.0f, .0f, 0.0f);
        glVertex2f(r_min - margem_r/3,x_max);
glVertex2f(r_min + margem_r/3, x_max);
glVertex2f(r_min, x_max +2*margem_x/3);

        glEnd();
glFlush();
return 1;
}

 //////////////////////////////////////////
// Funcão que desenha os pontos no monitor
void RenderScene(void){
while (i < (offset + num_pontos)){
x = calc[n]();
glColor3f(1.0f, 1.0f, 1.0f);
glBegin(GL_POINTS);
if ((r>r_min)&&(r<r_max)&&(x>x_min)&&(x<x_max))
glVertex2f(r,x);
glEnd();
i++;
}
// Flush drawing commands and swap
```

```
glutSwapBuffers();
}

/////////////////////////////////////////
// Called by GLUT library when idle
//(window not being resized or moved)
void TimerFunction(int value){
i = 0;
x = x0;
r+= (1.0/jumps);
while (i < offset){
x = calc[n]();
i++;
}
// Redraw the scene with new coordinates
glutPostRedisplay();
draw_eixos();
glutTimerFunc(1,TimerFunction, 1);
}

 ///////////////////////////////////////////
// Set up the Rendering state
void SetupRC(void)
{
// Set clear color to white
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT);
}

/////////////////////////////////////////
// Called by GLUT library when the
// window has chanaged size
void ChangeSize(GLsizei w, GLsizei h){
wgl = w; hgl = h;
set_glbvars[n]();
print_vars();
glClear(GL_COLOR_BUFFER_BIT);
if (h == 0) h = 1;
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho (r_min-margem_r , r_max + margem_r  , x_min-margem_x, x_max + margem_x, 1.0, -1.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
///////////////////////////////////////////
//Funcao que realiza o zoom
int zoom(){
x = x0;
r = r_min;
razao_r = (r_max-r_min)/(wgl-20);
razao_x = (x_max-x_min)/(hgl-20);
margem_r = BORDER*razao_r;
margem_x = BORDER*razao_x;
```

```
jumps = (int) (wgl+100)/(r_max-r_min+2*margem_r);
print_vars();
glClear(GL_COLOR_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho (r_min-margem_r , r_max + margem_r  , x_min-margem_x, x_max + margem_x, 1.0, -1.0);
glMatrixMode(GL_MODELVIEW);
return 1;
}

int draw_board(double a1,double b1,double a2,double b2){
glBegin(GL_LINES);
glVertex2d(a1,b1);
glVertex2d(a1,b2);
glVertex2d(a1,b1);
glVertex2d(a2,b1);
glVertex2d(a2,b2);
glVertex2d(a1,b2);
glVertex2d(a2,b2);
glVertex2d(a2,b1);
glEnd();
return 1;
}

////////////////////////////////////////////
void mouse(int button, int state, int r1, int x1){
double r2,x2,r3,x3;
double zoom_rtemp,zoom_xtemp;

//se o botao direito for pressionado:
if (button==GLUT_RIGHT_BUTTON){
set_glbvars[n]();  //reset às variaveis
glClear(GL_COLOR_BUFFER_BIT); //limpar o ecra
ChangeSize(wgl,hgl); //redesenhar
}

if (button==GLUT_LEFT_BUTTON){
if (state == GLUT_DOWN){ //se o botao esquerdo for pressionado:
if (mouse_presses==0){ //pela primeira vez
zoom_r = ((r_max-r_min+2*margem_r)*r1/wgl)+r_min-margem_r;
zoom_x = ((x_max-x_min+2*margem_x)*(hgl-x1)/hgl)+x_min-margem_x;
mouse_presses = 1;
r2 = ((r_max-r_min+2*margem_r)*(r1-1)/wgl)+r_min-margem_r;
r3 = ((r_max-r_min+2*margem_r)*(r1+1)/wgl)+r_min-margem_r;
x2 = ((x_max-x_min+2*margem_x)*(hgl-(x1-1))/hgl)+x_min-margem_x;
x3 = ((x_max-x_min+2*margem_x)*(hgl-(x1+1))/hgl)+x_min-margem_x;
glColor3f(0.1f,1.0f,0.1f);
glRectf(r2,x2,r3,x3);
}
else { //ou pela segunda
zoom_rtemp = ((r_max-r_min+2*margem_r)*r1/wgl)+r_min-margem_r;
zoom_xtemp = ((x_max-x_min+2*margem_x)*(hgl-x1)/hgl)+x_min-margem_x;
r2 = ((r_max-r_min+2*margem_r)*(r1-1)/wgl)+r_min-margem_r;
r3 = ((r_max-r_min+2*margem_r)*(r1+1)/wgl)+r_min-margem_r;
```

```
x2 = ((x_max-x_min+2*margem_x)*(hgl-(x1-1))/hgl)+x_min-margem_x;
x3 = ((x_max-x_min+2*margem_x)*(hgl-(x1+1))/hgl)+x_min-margem_x;
glColor3f(0.1f,1.0f,0.1f);
glRectf(r2,x2,r3,x3);

if (zoom_rtemp<zoom_r){ r_min = zoom_rtemp;
r_max = zoom_r;
}
else { r_min = zoom_r;
r_max = zoom_rtemp;
}
if (zoom_xtemp<zoom_x){ x_min = zoom_xtemp;
x_max = zoom_x;
}
else { x_min = zoom_x;
x_max = zoom_xtemp;
}
glColor3f(0.1f,1.0f,0.1f);
//glRectf(r_min,x_min,r_max,x_max);
draw_board(r_min,x_min,r_max,x_max);
mouse_presses = 2;
}
}
else if (mouse_presses==2) { //se o botao esquerdo for libertado
mouse_presses = 0;
zoom();
}
}
}
////////////////////////////////////////////////////////////////////////////
//Funcao que verifica os argumentos passados à main()
int verify_args(int argc, char *argv[]){
if (argc!=2){
printf("\n./logistic num_funcao\n");
return 0;
}
else {
sscanf(argv[1],"%d",&n);
if ((n<0)||(n>=10)){
return 0;
printf("\n./logistic num_funcao\n");
}
else return 1;
}

}
////////////////////////////////////////////////////////
// Main program entry point
int main(int argc, char* argv[]){
if (!(verify_args(argc,argv))) return 0;
else {
set_glbvars[n]();
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
```

```
glutInitWindowSize(wgl,hgl);
glutCreateWindow("Logistic Map");
glutDisplayFunc(RenderScene);
glutReshapeFunc(ChangeSize);
glutTimerFunc(1, TimerFunction, 1);
glutMouseFunc(mouse);
SetupRC();
glutMainLoop();
}
return 0;
}
```