**3.1.4 Instrumentación computacional del método de la bisección**

**function c = biseccion(f, a, b, e)**

**while b-a >= e**

**c=(a+b)/2;**

**if f(c)==0**

**return**

**else**

**if sign(f(a))==sign(f(c))**

**a=c;**

**else**

**b=c;**

**end**

**end**

**end**

**3.3.6 Instrumentación computacional del método de Newton**

**function [u,k]=newton(f,v,u,e,m)**

**t=u;**

**for k=1:m**

**u=u-subs(f,v,u)/subs(diff(f,v),v,u);**

**if abs(t-u)<e**

**return**

**end**

**t=u;**

**end**

**u=[ ];**

**k=m;**

**3.5.4 Instrumentación computacional del método de Newton para un sistema de n**

**ecuaciones no-lineales.**

**function u = snewton(f, v, u)** %Sistemas no lineales – LRO/2011

**n=length(f);**

**for i=1:n** %Obtención de la matriz jacobiana J

**for j=1:n**

**J(i,j)=diff(f(i),v(j));**

**end**

**end**

**for i=1:n** %Sustitución del vector u en J

**for j=1:n**

**for k=1:n**

**if findstr(char(J(i,j)),char(v(k)))>0**

**J(i,j)=subs(J(i,j),v(k),u(k));**

**end**

**end**

**end**

**end**

**for i=1:n**

**for j=1:n**

**f(i)=subs(f(i),v(j),u(j));** %Sustitución del vector u en el vector f

**end**

**end**

**u=u-inv(eval(J))\*eval(f);** %Obtención de la nueva aproximación u

# 4.2.3 Instrumentación computacional del método básico de Gauss-Jordan

# function x=gaussjordan(a,b)

**n=length(b);**

**a=[a,b];** %matriz aumentada

**for e=1:n**

**a(e,e:n+1)=a(e,e:n+1)/a(e,e);** %normalizar fila e

**for i=1:n**

**if i~=e**

**a(i,e:n+1)=a(i,e:n+1)-a(i,e)\*a(e,e:n+1);** %reducir otras filas

**end**

**end**

**end**

**x=a(1:n,n+1);** %vector solución

# 4.3.3 Instrumentación computacional del método básico de Gauss

# 

**function x=gauss1(a,b)**

**n=length(b);**

**a=[a,b];** %Matriz aumentada

**for e=1:n**

# a(e,e:n+1)=a(e,e:n+1)/a(e,e); %Normalizar la fila e

# for i=e+1:n %Reducir otras filas

# a(i,e:n+1)=a(i,e:n+1)-a(i,e)\*a(e,e:n+1);

# end

**end**

# x(n,1)=a(n,n+1); %Solución del sistema triangular

# for i=n-1:-1:1

# x(i,1)=a(i,n+1)-a(i,i+1:n)\*x(i+1:n,1);

# end

# 4.3.5 Instrumentación computacional del método de Gauss con pivoteo

# function x=gauss(a,b)

# [n,m]=size(a);

**if n~=m** %Verificar si la matriz es cuadrada

**x=[ ];**

**return;**

**end**

# a=[a,b]; %Matriz aumentada

# for e=1:n

# [z, p]=max(abs(a(e:n,e))); %Pivoteo por filas

# p=p+e-1;

# t=a(e,e:n+1); %Intercambio de filas

# a(e,e:n+1)=a(p,e:n+1);

# a(p,e:n+1)=t;

**if abs(a(e,e))<1.0e-10** %Si el divisor es ~cero, no hay solución

# x=[ ];

# return;

# end

# a(e,e:n+1)=a(e,e:n+1)/a(e,e); %Normalizar la fila e

# for i=e+1:n %Reducir otras filas

# a(i,e:n+1)=a(i,e:n+1)-a(i,e)\*a(e,e:n+1);

# end

# end

# x(n,1)=a(n,n+1); %Solución del sistema triangular

# for i=n-1:-1:1

# x(i,1)=a(i,n+1)-a(i,i+1:n)\*x(i+1:n,1);

# end

**4.5.1 Instrumentación computacional: sistemas singulares**

**function [x,a]=slin(a,b)**

**[n,m]=size(a);**

**z=max(max(a));**

**v=[n+1:m];** %Vector inicial de variables libres

**a(1:n,m+1)=b;** %Matriz aumentada

**if n>m** %Mas ecuaciones que variables

**x=[ ];**

**a=[ ];**

**return;**

**end**

**a=a/z;** %Estandariza la matriz p' reducir error

**for e=1:n** %Ciclo para n etapas

**[z,p]=max(abs(a(e:n,e)));** %Pivoteo por filas

**p=p+e-1;**

**t=a(e,e:m+1);** %Cambiar filas

**a(e,e:m+1)=a(p,e:m+1);**

**a(p,e:m+1)=t;**

**if abs(a(e,e))<1.0e-10** %Sistema singular

**v=[v, e];** %Agregar variable libre y continuar

**else**

**a(e,e:m+1)=a(e,e:m+1)/a(e,e);** %Normalizar la fila e

**for i=1:n** %Reducir otras filas

**if i~=e**

**a(i,e:m+1)=a(i,e:m+1)-a(i,e)\*a(e,e:m+1);**

**end**

**end**

**end**

**end**

**x=[ ];**

**if length(v)==0;** %Sistema consistente. Solución única

**x=a(1:n,m+1);** %El vector X es la última columna de A

**a(:,m+1)=[ ];** %Eliminar la última columna de A

**return;**

**end**

**4.6.1 Instrumentación computacional: sistemas tridiagonales**

**function x = tridiagonal(a, b, c, d)**

**n = length(d);**

**w(1) = b(1);**

**g(1) = d(1)/w(1);**

**for i = 2:n** %Transformación matricial

**w(i) = b(i) - a(i)\*c(i-1)/w(i-1);**

**g(i) = (d(i) - a(i)\*g(i-1))/w(i);**

**end**

**x(n) = g(n);**

**for i = n-1:-1:1** %Obtención de la solución

**x(i) = g(i) - c(i)\*x(i+1)/w(i);**

**end**

**5.1.1 Instrumentación computacional del método de Jacobi**

**function x = jacobi(a,b,x)**

**n=length(x);**

**t=x;** % t es asignado con el vector X ingresado

**for i=1:n**

**s=a(i,1:i-1)\*t(1:i-1)+a(i,i+1:n)\*t(i+1:n);**

**x(i) = (b(i) - s)/a(i,i);**

**end**

**5.1.2 Instrumentación computacional del método de Jacobi con control de iteraciones**

**function [x,k] = jacobim(a,b,x,e,m)**

**n=length(x);**

**for k=1:m**

**t=x;**

**for i=1:n**

**s=a(i,1:i-1)\*x(1:i-1)+a(i,i+1:n)\*x(i+1:n);**

**x(i) = (b(i) - s)/a(i,i);**

**end**

**if norm((x-t),inf)<e**

**return**

**end**

**end**

**x=[ ];**

**k=m;**

**5.2.1 Instrumentación computacional del método de Gauss-Seidel**

**function x = gaussseidel(a,b,x)**

n=length(x);

**for** i=1:n

s=a(i,1:i-1)\*x(1:i-1)+a(i,i+1:n)\*x(i+1:n); %Usa el vector x actualizado

x(i) = (b(i) - s)/a(i,i);

**end**

**5.2.2 Instrumentación computacional del método de Gauss-Seidel con control de**

**iteraciones**

**function [x,k] = gaussseidelm(a,b,x,e,m)**

n=length(x);

**for** k=1:m

t=x;

**for** i=1:n

s=a(i,1:i-1)\*x(1:i-1)+a(i,i+1:n)\*x(i+1:n);

x(i) = (b(i) - s)/a(i,i);

**end**

**if** norm((x-t),inf)<e

**return**

**end**

**end**

x=[ ];

k=m;

**5.3.1 Instrumentación computacional del método de relajación**

**function x = relajacion(a,b,x,w)**

n=length(x**);**

**for** i=1:n

s=a(i,1:n)\*x(1:n); %Usa el vector x actualizado

x(i)=x(i)+w\*(b(i)-s)/a(i,i);

**end**

**6.2.2 Instrumentación computacional: polinomio de Lagrange**

**function p = lagrange(x, f, v)**

n=length(x);

syms t; %Variable para el polinomio

p=0;

**for** i=1:n

L=1;

**for** j=1:n

**if** i ~= j

L=L\*(t-x(j))/(x(i)-x(j));

**end**

**end**

p=p+L\*f(i); %entrega **p(t)** en forma simbólica

**end**

p=simplify(p); %simplificación algebraica

**if** nargin==3 %verifica si existe un parámetro adicional

t=v;

p=eval(p);% entrega el resultado de **p** evaluado en **v**

**end**

**6.3.1 Instrumentación computacional: polinomio de Lagrange para dos variables**

**function** p=lagrange2(x, y, f, u, v)

[n,m]=size(f);

**for** i=1:m

r(i)=lagrange(x, f(:, i), u);% cada columnas es enviada (resultados parciales)

**end**

p=lagrange(y, r, v); %interpolación final en la otra dirección

**6.9.3 Instrumentación computacional del trazador cúbico natural**

**function [a,b,c,d]=trazador(x,y,z)**

% Trazador Cúbico Natural: a(i)(x-x(i))^3+b(i)(x-x(i))^2+c(i)(x-x(i))+d(i), n>3

% z es opcional: es el vector de puntos para evaluar al trazador

% Entrega puntos del trazador o los coeficientes de los polinomios segmentarios

n=length(x);

clear A B C D;

**if** n<4

**return**

**end**

**for** i=1:n-1

h(i)=x(i+1)-x(i);

**end**

s(1)=0;

s(n)=0;

B(1)=2\*(h(1)+h(2));

C(1)=h(2);

D(1)=6\*((y(3)-y(2))/h(2)-(y(2)-y(1))/h(1))-h(1)\*s(1);

**for** i=2:n-3 % Sistema tridiagonal para obtener S

A(i)=h(i);

B(i)=2\*(h(i)+h(i+1));

C(i)=h(i+1);

D(i)=6\*((y(i+2)-y(i+1))/h(i+1)-(y(i+1)-y(i))/h(i));

**end**

A(n-2)=h(n-2);

B(n-2)=2\*(h(n-2)+h(n-1));

D(n-2)=6\*((y(n)-y(n-1))/h(n-1)-(y(n-1)-y(n-2))/h(n-2))-h(n-1)\*s(n);

u=tridiagonal(A,B,C,D);

**for** i=2:n-1

s(i)=u(i-1);

**end**

**for** i=1:n-1 % Coeficientes del trazador cúbico natural

a(i)=(s(i+1)-s(i))/(6\*h(i));

b(i)=s(i)/2;

c(i)=(y(i+1)-y(i))/h(i)-(2\*h(i)\*s(i)+h(i)\*s(i+1))/6;

d(i)=y(i);

**end**

**if** nargin==3 % Puntos del trazador cúbico natural

p=[ ];

m=length(z);

**for** k=1:m

t=z(k);

**for** i=1:n-1

**if** t>=x(i) & t<=x(i+1)

p(k)=a(i)\*(t-x(i))^3+b(i)\*(t-x(i))^2+c(i)\*(t-x(i))+d(i);

**end**

**end**

**end**

**if** m>1

k=m;i=n-1;

p(k)=a(i)\*(t-x(i))^3+b(i)\*(t-x(i))^2+c(i)\*(t-x(i))+d(i);

**end**

clear a b c d;

a=p;

**end**

**6.9.6 Instrumentación computacional del trazador cúbico sujeto**

**function [a,b,c,d]=trazadorsujeto(x,y,u,v,z)**

% Trazador Cúbico Sujeto

% u,v son las pendientes en los extremos

% a(i)(z-x(i))^3+b(i)(z-x(i))^2+c(i)(z-x(i))+d(i), n>3

% z es opcional: es el vector de puntos para evaluar al trazador

% Entrega puntos del trazador o los coeficientes de los polinomios segmentarios

n=length(x);

clear A B C D;

**if** n<4

**return**

**end**

**for** i=1:n-1

h(i)=x(i+1)-x(i);

**end**

B(1)=-2\*h(1)/6;

C(1)=-h(1)/6;

D(1)=u-(y(2)-y(1))/h(1);

**for** i=2:n-1

A(i)=h(i-1);

B(i)=2\*(h(i-1)+h(i));

C(i)=h(i);

D(i)=6\*((y(i+1)-y(i))/h(i)-(y(i)-y(i-1))/h(i-1));

**end**

A(n)=h(n-1)/6;

B(n)=h(n-1)/3;

D(n)=v-(y(n)-y(n-1))/h(n-1);

s=tridiagonal(A,B,C,D);

**for** i=1:n-1 % Coeficientes del trazador cúbico sujeto

a(i)=(s(i+1)-s(i))/(6\*h(i));

b(i)=s(i)/2;

c(i)=(y(i+1)-y(i))/h(i)-(2\*h(i)\*s(i)+h(i)\*s(i+1))/6;

d(i)=y(i);

**end**

**if** nargin==5 % Puntos del trazador cúbico sujeto

p=[ ];

m=length(z);

**for** k=1:m

t=z(k);

**for** i=1:n-1

**if** t>=x(i) & t<=x(i+1)

p(k)=a(i)\*(t-x(i))^3+b(i)\*(t-x(i))^2+c(i)\*(t-x(i))+d(i);

**end**

**end**

**end**

**if** m>1

k=m;i=n-1;

p(k)=a(i)\*(t-x(i))^3+b(i)\*(t-x(i))^2+c(i)\*(t-x(i))+d(i);

**end**

clear a b c d;

a=p;

**end**

**7.1.3 Instrumentación computacional de la fórmula de los trapecios**

**function r = trapecios(f, a, b, m)**

h=(b-a)/m;

s=0;

**for** i=1: m - 1

s=s + f(a + i\*h);

**end**

r = h/2\*(f(a) + 2\*s + f(b));

**7.1.6 Instrumentación computacional de la fórmula de Simpson**

**function area = simpson(f, a, b, m)**

h=(b-a)/m;

s=0;

for i=1:m-1

s=s+2\*(mod(i,2)+1)\*f(a+i\*h); %Sumar los términos con coeficientes 4, 2, 4, 2,...,4

end

area=h/3\*(f(a) + s + f(b));

**7.3.2 Instrumentación computacional de la cuadratura de Gauss**

**function s = cgauss(f, a, b)**

t1=-(b-a)/2\*1/sqrt(3)+(b+a)/2;

t2= (b-a)/2\*1/sqrt(3)+(b+a)/2;

s = (b-a)/2\*(f(t1) + f(t2));

**7.3.3 Instrumentación extendida de la cuadratura de Gauss**

**function t=cgaussm(f, a, b, m)**

h=(b-a)/m;

t=0;

x=a;

**for** i=1:m

a=x+(i-1)\*h;

b=x+i\*h;

s=cgauss(f,a,b);

t=t+s;

**end**

**7.5.1 Instrumentación computacional de fórmula de Simpson en dos direcciones**

**function s=simpson2(f, ax, bx, ay, by, mx, my)**

dy=(by-ay)/my;

y=ay;

**for** i=1:my+1

g = subs(f,'y',y); %Se redefine f con los valores de y

r(i) = simpson(inline(g), ax, bx, mx);

y = y+dy;

**end**

s=0;

**for i**=2:my

s=s+2\*(2-mod(i,2))\*r(i);

**end**

s=dy/3\*(r(1)+s+r(my+1));

## 9.1.3 Instrumentación computacional de la fórmula de Euler

**function [x,y] = euler(f, x, y, h)**

**y=y + h\*f(x,y);**

**x=x+h;**

## 9.1.5 Instrumentación computacional de la fórmula de Heun

**function [x,y] = heun(f, x, y, h)**

k1=h\*f(x,y);

k2=h\*f(x+h, y+k1);

y=y+0.5\*(k1+k2);

x=x+h;

## 9.1.7 Instrumentación computacional de la fórmula de Runge-Kutta

**function [x,y]=rk4(f, x, y, h)**

k1=h\*f(x,y);

k2=h\*f(x+h/2, y+k1/2);

k3=h\*f(x+h/2, y+k2/2);

k4=h\*f(x+h, y+k3);

y=y+1/6\*(k1+2\*k2+2\*k3+k4);

x=x+h;

## 9.2.2 Instrumentación computacional de la fórmula de Heun para dos

**function [x,y,z]=heun2(f, g, x, y, z, h)**

k1y=h\*f(x,y,z)

k1z=h\*g(x,y,z)

k2y=h\*f(x+h,y+k1y,z+k1z)

k2z=h\*g(x+h,y+k1y,z+k1z)

y=y+0.5\*(k1y+k2y);

z=z+0.5\*(k1z+k2z);

x=x+h;

## 9.2.4 Instrumentación computacional de la fórmula de Runge-Kutta

**function [x,y,z]=rk42(f,g,x,y,z,h)**

k1y=h\*f(x,y,z);

k1z=h\*g(x,y,z);

k2y=h\*f(x+h/2,y+k1y/2,z+k1z/2);

k2z=h\*g(x+h/2,y+k1y/2,z+k1z/2);

k3y=h\*f(x+h/2,y+k2y/2,z+k2z/2);

k3z=h\*g(x+h/2,y+k2y/2,z+k2z/2);

k4y=h\*f(x+h,y+k3y,z+k3z);

k4z=h\*g(x+h,y+k3y,z+k3z);

y=y+1/6\*(k1y+2\*k2y+2\*k3y+k4y);

z=z+1/6\*(k1z+2\*k2z+2\*k3z+k4z);

x=x+h;

**9.6.3 Instrumentación computacional: diferencias finitas**

**function [u,v] = edodif(P, Q, R, S, x0, y0, xn, yn, n)**

% Método de Diferencias Finitas

% Solución de una EDO con condiciones constantes en los bordes

h=(xn-x0)/n;

clear a b c d; % generación de las diagonales del sistema tridiagonal

**for i=1:n-1**

x=x0+h\*i;

a(i)=eval(P);

b(i)=eval(Q);

c(i)=eval(R);

d(i)=eval(S);

u(i)=x;

**end**

d(1)=d(1)-a(1)\*y0;

d(n-1)=d(n-1)-c(n-1)\*yn;

v=tridiagonal(a,b,c,d); % Solucion del sistema tridiagonal

**9.6.5 Instrumentación computacional: diferencias finitas**

**function [u,v]=edodifdi(P,Q,R,S,x0,dy0,xn,yn,n)**

% Método de Diferencias Finitas

% Solución de una EDO con una derivada a la izquierda

% y una condición constante a la derecha

h=(xn-x0)/n;

**clear** a b c d; % generación de las diagonales del sistema tridiagonal

**for** i=1:n % corresponde a las ecuaciones i = 0, 1, 2, ..., n-1

x=x0+h\*(i-1);

a(i)=eval(P);

b(i)=eval(Q);

c(i)=eval(R);

d(i)=eval(S);

u(i)=x;

**end**

x=h;

c(1)=c(1)+eval(P);

d(1)=d(1)+eval(P)\*2\*h\*dy0;

d(n)=d(n)-c(n)\*yn;

v=tridiagonal(a,b,c,d); % solucion del sistema tridiagonal

**10.2.2 Instrumentación computacional: EDP tipo parabólico**

% Resolución de una EDP con el Método de Diferencias Finitas

% Ecuación de diferencias estandarizada

% U(i,j+1)=(P)U(i-1,j) + (Q)U(i,j) + (R)U(i-1,j)

% P,Q,R son constantes evaluadas con los datos de la EDP

clf;

m=11; % Número de puntos en x

n=100; % Número de niveles en t

Ta=60; Tb=40; % Condiciones en los bordes

To=25; % Condición en el inicio

dx=0.1; dt=0.01; % incrementos

L=1; % longitud

c=4; % dato especificado

**clear** x U;

U(1)=Ta; % Asignación inicial

U(m)=Tb;

**for** i=2:m-1

U(i)=To;

**end**

**k = dt/(c\*dx^2);**

**P=k;**

**Q=1-2\*k;**

**R=k;**

hold on;

title('Curvas de distribución térmica');

xlabel('X (distancia)');

ylabel('U (temperatura)');

x=0:dx:L; % Coordenadas para el gráfico

plot(x,U,'r'); grid on; % Distribución inicial

**for** j=1:n

U=**EDPDIF**(P,Q,R,U,m);

**if** mod(j,5)==0

plot(x,U,'r'); % Para graficar curvas cada 5 niveles de t

**pause**

**end**

**end**

**function** u=**EDPDIF**(P,Q,R,U,m)

% Solución U(x,t) de una EDP con condiciones constantes en los bordes

% Método explícito de diferencias finitas

u(1)=U(1);

**for** i=2:m-1

u(i)=P\*U(i-1)+Q\*U(i)+R\*U(i+1);

**end**

u(m)=U(m);

**10.2.4 Instrumentación computacional: EDP tipo parabólico**

% Método de Diferencias Finitas implícito para una EDP

% Ecuación de diferencias estandarizada

% (P)ui-1,j + (Q)ui,j + (R)ui+1,j = -ui,j-1

% P,Q,R son constantes evaluadas con los datos de la EDP

clf;

m=11; % Número de puntos en x

n=100; % Número de niveles en t

Ta=60; Tb=40; % Condiciones en los bordes

To=25; % Condición en el inicio

dx=0.1; dt=0.01; % incrementos

L=1; % longitud

c=4; % dato especificado

**clear** x u U;

U(1)=Ta; % Asignación inicial

U(m)=Tb;

**for** i=2:m-1

U(i)=To;

**end**

**k=dt/(c\*dx^2);**

**P=k;**

**Q=-1-2\*k;**

**R=k;**

hold on;

title('Curvas de distribución térmica ');

xlabel('X (distancia)');

ylabel('U (temperatura)');

x=0:dx:L; % Coordenadas para el grafico

plot(x,U,'r'); % Distribución inicial

**for** j=1:n

**U =** **EDPDIFPI(P, Q, R, U, m);**

**if** mod(j,5)==0

plot(x,U,'r'); % Para graficar curvas cada 5 niveles de t

**pause**

**end**

**end**

**10.2.6 Instrumentación computacional: EDP tipo parabólica**

% Método de Diferencias Finitas implícito para una EDP

% condiciones en los bordes:

% A la izquierda una derivada

% Condición inicial y en el borde derecho variables

% Ecuación de diferencias estandarizada

% (P)ui-1,j + (Q)ui,j + (R)ui+1,j = -ui,j-1

% P,Q,R son constantes evaluadas con los datos de la EDP

clf;

m=11; % Número de puntos en x

n=50; % Número de niveles en t

der0=-5; % Derivada en el borde izquierdo

dx=0.1;

dt=0.1; % incrementos

L=1; % longitud

c=4; % dato especificado

clear x U;

x=0;

**for** i=1:m % Condición variable en el inicio

U(i)=40\*x;

x = x + dx;

**end**

**k=dt/(c\*dx^2);**

**P=k;**

**Q=-1-2\*k;**

**R=k;**

hold on;

title('Curvas de distribución térmica');

xlabel('X (distancia)');

ylabel('U (temperatura)');

x=0:dx:L; % Coordenadas para el grafico

t=0;

**for** j=1:n

t = t + dt;

U(m) = 20 + 10\*sin(t); % Condición variable en el borde derecho

U = EDPDIFPID(P, Q, R, U, der0, dx, m);

plot(x,U,'r');

**pause;**

**end**

**function** U = EDPDIFPID(P, Q, R, U, der0, dx, m)

% Método de Diferencias Finitas Implicito

% EDP con una derivada a la izquierda y condiciones variables

% Generación del sistema tridiagonal

**clear** a b c d;

**for** i=1:m-1

a(i)=P;

b(i)=Q;

c(i)=R;

d(i)=-1\*U(i);

**end**

c(1)=2\*P;

d(1)=d(1)-P\*der0\*2\*dx;

d(m-1)=d(m-1)-c(m-1)\*U(m);

u=tridiagonal(a,b,c,d);

U=[ u U(m)]; % Incluir dato en el extremos

**10.3.2 Instrumentación computacional: EDP tipo elíptica**

% Programa para resolver una EDP Elíptica

% con condiciones constantes en los bordes

Ta=60;Tb=60;Tc=50;Td=70; % Bordes izquierdo, derecho, abajo, arriba

n=10; % Puntos interiores en dirección horizontal (X)

m=10; % Puntos interiores en dirección vertical (Y)

miter=100; % Máximo de iteraciones

e=0.001; % Error de truncamiento relativo requerido 0.1%

clear u;

**for** i=1:n+2

u(i,1)=Tc;

u(i,m+2)=Td;

**end**

**for** j=1:m+2

u(1,j)=Ta;

u(n+2,j)=Tb;

**end**

p=0.25\*(Ta+Tb+Tc+Td); %valor inicial es el promedio de los bordes

**for** i=2:n-1

**for** j=2:m-1

u(i,j)=p;

**end**

**end**

k=0; %conteo de iteraciones

conv=0; %señal de convergencia

**while** k<miter & conv==0

k=k+1;

t=u;

**for** i=2:n+1

**for** j=2:m+1

u(i,j)=0.25\*(u(i-1,j)+u(i+1,j)+u(i,j+1)+u(i,j-1));

**end**

**end**

**if** norm((u-t),inf)/norm(u,inf)<e

conv=1;

**end**

**end**

**if** conv==1

disp(u); % Muestra la solución final en la malla

disp(k); % Cantidad de iteraciones realizadas

[x,y]=meshgrid(1:m+2, 1:n+2); % Malla para el grafico en tres dimensiones

surf(x,y,u) % Grafico en tres dimensiones

colormap **copper**  % Color

shading **flat**  % Suavizado

**else**

disp('No converge');

**end**

**10.4.2 Instrumentación computacional: EDP tipo hiperbólica**

% Método de Diferencias Finitas explícito para una EDP Hiperbólica

% con parametro c^2 dt^2/dx^2 = 1

% Extremos fijos, estiramiento central y velocidad inicial cero

clf;

m=11; % Número de puntos en x

n=50; % Número de niveles en t

c=2; % dato especificado

dx=0.1;

dt=sqrt(dx^2/2^2); % incrementos

L=1; % longitud

clear x U0 U1 Uj;

U0(1)=0; % Extremos fijos

U0(m)=0;

x=0;

**for** i=2:m-1 % Posición inicial de la cuerda

x=x+dx;

**if** x<L/2;

U0(i)=-0.5\*x;

**else**

U0(i)=0.5\*(x-1);

**end**

**end**

x=0:dx:L; % Coordenadas para el grafico

plot(x,U0,'r'); % Distribución inicial

axis([0,1,-0.5,0.5]);

**pause;**

**clf;**

U1=EDPDIFH1(U0,m); % Calculo del primer nivel

plot(x,U1,'r');

axis([0,1,-0.5,0.5]);

**pause;**

**clf;**

**for** j=1:n

Uj=EDPDIFHJ(U0,U1,m); % Siguientes niveles

plot(x,Uj,'r');

axis([0,1,-0.5,0.5]);

**pause;**

**clf;**

U0=U1;

U1=Uj;

**end**

**function** U1=EDPDIFH1(U0,m)

% Solución U(x,t) de una EDP hiperbolica con parametro igual a 1

% Cálculo del primer nivel de la solucion

U1(1)=U0(1);

**for** i=2:m-1

U1(i)=0.5\*(U0(i-1)+U0(i+1));

**end**

U1(m)=U0(m);

**function** Uj=EDPDIFHJ(U0,U1,m)

% Solución U(x,t) de una EDP hiperbolica con parametro igual a 1

% Cálculo de los siguientes niveles de la solucion

Uj(1)=U1(1);

**for** i=2:m-1

Uj(i)=U1(i+1)+U1(i-1)-U0(i);

**end**

Uj(m)=U1(m);

**function** **U = EDPDIFPI(P, Q, R, U, m)**

% Solución de una EDP con condiciones constantes en los bordes

% Método de Diferencias Finitas Implícito

% Generación del sistema tridiagonal

**clear** a b c d;

**for** i=1:m-2

a(i)=P;

b(i)=Q;

c(i)=R;

d(i)=-1\*U(i+1);

**end**

d(1)=d(1)-a(1)\*U(1);

d(m-2)=d(m-2)-c(m-2)\*U(m);

u=tridiagonal(a,b,c,d);

U=[U(1) u U(m)]; % Incluir datos en los extremos