Resolução de sistemas não lineares

Leonardo da Silva Garcia Leite

Maio 2013

1 Introdução

É comum no ramo da física e matemática encontrarmos sistemas não lineares que não podem ou que apresentam resolução extremamente complicada analiticamente. Esses sistemas podem ser solucionados numericamente utilizando uma generalização do método de Newton para dimensões superiores em conjunto com um algoritmo de resolução de sistemas lineares.

2 Fundamentos teóricos

2.1 Método Newton em 1 dimensão

O método de Newton consiste em escolhermos um valor inicial e iterarmos o algoritmo de forma a encontrar a raiz da equação desejada através do cálculo da intersecção da derivada com o eixo x. Uma das formas de se deduzir é usando a expansão da função em série de Taylor em torno de x temos

$$f(x+\Delta x) \approx f(x) + \frac{df(x)}{dx}\Delta x + \mathcal{O}(2) \tag{1}$$

agora usando $\Delta x = x_0 + x$ onde x_0 é uma raiz da função obtemos

$$\Delta x = -\frac{f(x)}{f'(x)} \tag{2}$$

Ou ainda de forma iterada

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3}$$

Para sucesso do método precisamos de uma boa escolha de x_i para que seja próximo da raiz e que $f'(x_i)$ não seja identicamente nula.

2.2 Método Newton em 2 dimensões aplicado para sistemas não lineares

Usando processo análogo ao de uma dimensão fazemos

$$f_1(x+\Delta x, y+\Delta y) \approx f_1(x,y) + \frac{\partial f1(x,y)}{\partial x}\Delta x + \frac{\partial f1(x,y)}{\partial y}\Delta y + \mathcal{O}(2) \tag{4}$$

$$f_2(x+\Delta x, y+\Delta y) \approx f_2(x,y) + \frac{\partial f2(x,y)}{\partial x}\Delta x + \frac{\partial f2(x,y)}{\partial y}\Delta y + \mathcal{O}(2) \tag{5}$$

utilizando-se $\Delta x = x_0 + x$ e $\Delta y = y_0 + y$ onde x_0 e y_0 são raízes da equação. Obtemos o sistema

$$\begin{cases} \frac{\partial f1(x,y)}{\partial x}\Delta x + \frac{\partial f1(x,y)}{\partial y}\Delta y + f_1(x,y) = 0 \\ \frac{\partial f2(x,y)}{\partial x}\Delta x + \frac{\partial f2(x,y)}{\partial y}\Delta y + f_2(x,y) = 0 \end{cases} \tag{6}$$

Que podemos escrever em forma matricial

$$\begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} -f1(x,y) \\ -f2(x,y) \end{bmatrix} \tag{7}$$

A primeira matriz é famosas entre o meio matemático e é a matriz Jacobiana. O sistema pode ser resolvido facilmente para Δx e Δy utilizando qualquer método de resolução de sistema lineares.

Um dos métodos é chamado de decomposição LU, que consiste em transformar a matriz dos coeficientes em um produto de duas matrizes, uma triangular inferior e uma triangular superior, aumentando o numero de sistemas a serem resolvidos mas diminuindo consideravelmente sua complexidade.

Em notação matemática

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \tag{8}$$

$$\begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} 1 & U_{12} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \tag{9}$$

Onde [A] é matriz dos coeficientes, [x] a matriz dos deltas e [B] a matriz das f(x,y), sendo que [A] e [B] serão calculadas no ponto x, y escolhidos como valores iniciais. Como o sistema [L][U]=[A] não é único fazemos a escolha mas simples escolhendo a diagonal principal de [U] como 1.

Obtemos então o seguinte sistema

$$\begin{cases} Ux = y \\ Ly = B \end{cases} \tag{10}$$

Realizando as multiplicações das matrizes acima podemos encontrar o termo geral para [U], [y], [L] e [x] que serão dados por:

$$L_{ij} = A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj} \tag{11}$$

$$U_{ij} = \frac{1}{L_{ii}} \left(A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj} \right) \tag{12}$$

$$y_i = \frac{1}{L_{ii}} \left(B_i - \sum_{j=1}^{i-1} L_{ij} y_j \right) \tag{13}$$

$$x_i = \frac{1}{U_{ii}} \left(y_i - \sum_{k=i+1}^{N} U_{ik} x_k \right) \tag{14}$$

E por fim realizamos o último calculo do método de Newton

$$\begin{cases} x_{i+1} = x_i + \Delta x \\ y_{i+1} = y_i + \Delta y \end{cases} \tag{15}$$

Que pode ser iterado até $f1(x_{i+1}, y_{i+1}) = f1(x_{i+1}, y_{i+1}) \approx 0$. Apesar desse resultado ser para o caso de dimensão 2 o procedimento pode ser feito de maneira análoga para dimensões superiores.

3 Resultados

Considere o seguinte sistema

$$\begin{cases} Sen(t) + x^2 + 2y^4 + 5z = 6 \\ t^5 + 3x + 3e^{-y} - 2z^2 = 4 \\ Senh(t^2) + 2x^3 - Cos(y) + z^2 = 2 \\ t^3 + x + 2y + 3z = 4 \end{cases} \tag{16}$$

Analisando por alto o sistema podemos pensar em escolher $t = 0$ e $y = 0$ para zerarmos os termos com oscilatórios e hiperbólicos assim restando um sistema mais simples, que com um pouco de mais de intuição leva á solução

$$\begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \tag{17}$$

Numericamente escolhendo valores próximos a esses como valores iniciais, como por exemplo $x = 0.9$, $y = 0$, $z = 0.9$ e $t = 0$, teremos como solução exatamente o vetor acima com apenas 8 passos.

Devido as funções oscilatórias que normalmente são um problema para o método de Newton, alguns valores iniciais não convergem e outros apresentam outras soluções sem ser a citada acima.

Um outro bom conjunto de valores inicias é pode ser obtido zerando as funções oscilatórias usando o valor de π como por exemplo

$$\begin{bmatrix} x_0 = 2 \\ y_0 = \frac{\pi}{2} \\ z_0 = 2 \\ t_0 = \pi \end{bmatrix} \rightarrow \begin{bmatrix} x = -0.6948132 \\ y = -0.185800 \\ z = 0.908856 \\ t = 1.327585 \end{bmatrix} \tag{18}$$

que convergiu em 18 passos.

Como o sistema não apresenta solução única é possível ainda encontrar outras soluções para diferentes valores iniciais. Todos os resultados foram obtidos usando-se $\|f_i(x, y, z, t)\| < 1e - 7$.

4 Apêndice

```
#include <stdio.h>
#include <math.h>
#define N 4

// funcoes
double funcao1(double x, double y, double z, double t){
        return sin(t)+pow(x,2)+2*pow(y,4)+5*z -6;
}
double funcao2(double x, double y, double z, double t){
        return pow(t,5) + 3*x + 3*exp(-y) -2*pow(z,2) - 4;
}
double funcao3(double x, double y, double z, double t){
        return sinh(pow(t,2))+2*pow(x, 3) - cos(y) + pow(z, 2) - 2;
}
double funcao4(double x, double y, double z, double t){
        return pow(t, 3)+ x + 2*y + 3*z - 4;
}

// derivadas da funcao 1
double df1x(double x, double y, double z, double t){    return 2*x;             }
double df1y(double x, double y, double z, double t){    return 8*pow(y, 3);     }
double df1z(double x, double y, double z, double t){    return 5;               }
double df1t(double x, double y, double z, double t){    return cos(t);          }

// derivadas da funcao 2
double df2x(double x, double y, double z, double t){    return 3;               }
double df2y(double x, double y, double z, double t){    return -3*exp(-y);      }
double df2z(double x, double y, double z, double t){    return -4*z; }
double df2t(double x, double y, double z, double t){    return 5*pow(t,4);      }

// derivadas da funcao 3
double df3x(double x, double y, double z, double t){    return 6*pow(x,2);}
double df3y(double x, double y, double z, double t){    return sin(y);          }
double df3z(double x, double y, double z, double t){    return 2*z;     }
double df3t(double x, double y, double z, double t){    return 2*t*cosh(t*t);}

// derivadas da funcao 4
double df4x(double x, double y, double z, double t){    return 1;               }
double df4y(double x, double y, double z, double t){    return 2;               }
double df4z(double x, double y, double z, double t){    return 3;               }
double df4t(double x, double y, double z, double t){    return 3*pow(t, 2);     }
```

```
double soma(double l[N][N], double u[N][N], int i, int j, int limite){
    int k;
    double s=0;
    for(k=0;k<=limite;k++)
        s+=l[i][k]*u[k][j];
    return s;
}

// imprime uma matriz
void imprime(double vetor[N][N]){
    int i,j;
    for(i=0;i<N;i++){
      for(j=0;j<N;j++){
        printf("\t%lf",vetor[i][j]);
        if(j==N-1) printf("\n");
       }
    }
}

// imprime um vetor
void imprimeV(double vetor[N]){
    int i;
    for(i=0;i<N;i++){
        printf("\t%lf",vetor[i]);
    }
    printf("\n");
}

int main()
{
        double x, y, z, t;
        double dx, dy, dz, dt;
        double ep = 1e-7;
        double sum;

        double a[N][N] = {0};
        double l[N][N] = {0}, u[N][N] = {0};
        double mx[N] = {0}, my[N] = {0};
        double gama[N] = {0};

        int i, j, k, passos = 1;

        // ansatz
        x = 2;
        y = 3.14/2;
        z = 2;
        t = 3.14;

        // inicio do metodo de Newton
        while(fabs(funcao1(x, y, z, t)) > ep || fabs(funcao2(x, y, z, t)) > ep ||
        fabs(funcao3(x, y, z, t)) > ep || fabs(funcao4(x, y, z, t)) > ep ){

                // definindo o jacobiano
                a[0][0] = df1x(x, y, z, t);
                a[0][1] = df1y(x, y, z, t);
                a[0][2] = df1z(x, y, z, t);
                a[0][3] = df1t(x, y, z, t);

                a[1][0] = df2x(x, y, z, t);
```

```
a[1][1] = df2y(x, y, z, t);
a[1][2] = df2z(x, y, z, t);
a[1][3] = df2t(x, y, z, t);

a[2][0] = df3x(x, y, z, t);
a[2][1] = df3y(x, y, z, t);
a[2][2] = df3z(x, y, z, t);
a[2][3] = df3t(x, y, z, t);

a[3][0] = df4x(x, y, z, t);
a[3][1] = df4y(x, y, z, t);
a[3][2] = df4z(x, y, z, t);
a[3][3] = df4t(x, y, z, t);

// definindo a matriz my
my[0] = -funcao1(x, y, z, t);
my[1] = -funcao2(x, y, z, t);
my[2] = -funcao3(x, y, z, t);
my[3] = -funcao4(x, y, z, t);

// resolvendo o sistema, L*gama = my e U*mx = gama

// calculando L e U
//valores iniciais
for(i=0;i<N;i++){
        l[i][0] = a[i][0];
        u[i][i] = 1;
        u[0][i] = a[0][i]/l[0][0];
}

//calculo dos lij e uij
for(i=1;i<N;i++){//comecam de 1 a frente

        //calculo dos lij
        for(j=1;j<=i;j++)
                l[i][j] = a[i][j] - soma(l,u,i,j,j-1);

        //calculo dos uij
        for(j=(i+1);j<N;j++)
                u[i][j] = (a[i][j] - soma(l,u,i,j,i-1))/l[i][i];
}

//L e U, calculamos a matriz gama                    L*gama = my
for(i=0;i<N;i++){
        sum = 0;
        for(j=0;j<i;j++)
                sum += l[i][j]*gama[j];
        gama[i] = (1/l[i][i])*(my[i] - sum);
}

// A matriz mx (solucao do sistema LU*mx = y)  U*mx = gama
for(i=N-1;i>=0;i--){
        sum = 0;
        for(j=i+1;j<N;j++)
                sum += u[i][j]*mx[j];
        mx[i] = (1/u[i][i])*(gama[i] - sum);
}

// solucao do metodo de Newton
x += mx[0];
y += mx[1];
```

```
                z += mx[2];
                t += mx[3];

                passos++;

        printf("RESULTADO: \n\n x = %lf , y = %lf , z = %lf ,
          t = %lf . Com %d passos . \n\n", x, y, z, t, passos);
                if(passos > 500) break;
        }

        // resultado

        system("pause");
}
```