MAP2210 Aplicações em Álgebra Linear
—Questão 1 P2—

Vitor Gonçalves Ribeiro Nº Usp:9379548

June 5, 2022

Abstract

Neste trabalho iremos aproximar numericamente uma equação diferencial de segunda ordem usando uma matriz esparsa, e analisaremos a ordem dessa aproximação, discutiremos o limite de erro da resolução computacional e abordaremos as vantagens em tempo na utilização de um modelo de matriz esparsa contra uma matriz normal.

1 Modelo da Solução

Neste modelo iremos aproximar numericamente a equação diferencial de segunda ordem

$$u''(x) = f(x),\ x \in (a, b) \tag{1}$$

Para isso discretizaremos a solução usando o Método das Diferenças Finitas, com esquemas diferentes entre as linhas da matriz, esses esquemas serão:

$$-\frac{1}{h^2}(u_{k-1} - 2u_k + u_{k+1}) = f_k \tag{2}$$

$$-\frac{1}{h^2}(2u_k - 5u_{k+1} + 4u_{k+2} + u_{k+3}) = f_k \tag{3}$$

$$-\frac{1}{12h^2}(-u_{k-2} + 16u_{k-1} - 30u_k + 16u_{k+1} - u_{k+2}) = f_k \tag{4}$$

Nos esquemas acima temos $u_k \approx u(x_k)$ e $f(x_k) = f_k$, onde o conjunto de pontos $x_k = a + hk,\ k = 0, 1, \ldots, n$ discretiza o domínio de definição da solução. Note que o passo $h = (b - a)/n$ é determinado pela escolha de n, que será nossa variável de análise.
Usaremos a equação (3) para a primeira linha da matriz (não temos um u_{k-2} para essa linha), e a condição (2) para a ultima (não temos um u_{k+2} nessa linha), para as demais utilizaremos a condição (4).
Utilizando condições de contorno de Dirichlet, ou seja os valores de *u(a)* e *u(b)* são dados, podemos transformar nosso problema na solução da expressão:

$$\begin{bmatrix} 2 & -5 & 4 & -1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\ 16 & -30 & 16 & -1 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\ -1 & 16 & -30 & 16 & -1 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 16 & -30 & 16 & -1 & 0 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 16 & -30 & 16 & -1 & 0 & \ldots & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & -1 & 16 & -30 & 16 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \ldots & 0 & 0 & 0 & 1 & -2 \end{bmatrix} * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ \vdots \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} = \begin{bmatrix} -a \\ -12a - u_0 \\ -12a \\ -12a \\ \vdots \\ \vdots \\ -12a + u_n \\ -a - u_n \end{bmatrix}$$

Como $u_0 = u(a)$ e $u_n = u(b)$ e são dados, elas vão para o lado da conhecido da equação.

2 Modelo Computacional

Para criar o modelo computacional descrito acima, foi criada a função P2Q1, onde recebe como parâmetros:

- f0: valor de u_0
- f1: valor de u_n
- x0: valor de a
- x1: valor de b
- n: numero de divisões n
- b: vetor resultado f_k

Essa função cria a Matriz A, conforme descrito na sessão de Modelo da Solução usando a biblioteca do python scipy.sparse, para aproveitar da esparsidade da matriz, reduzindo o tempo de execução. A função é mostrada abaixo.

```
def P2Q1(f0, f1, x0, x1, n, b):
    t = time()
    a = b*((x1-x0)/n)**2
    if(n>=5):
        A = lil_matrix((n-1,n-1))
        for i in range(n-1):
        if(i==0):
                        A[i,i] = 2
                A[i,i+1] = -5
                A[i,i+2] = 4
                A[i,i+3] = -1
                a[i] = -a[i]
                elif(i==n-2):
                        A[i,i] = -2
                        A[i,i-1] = 1
                        a[i] = -a[i] - f1
                else:
                        a[i] = -12*a[i]
                        if(i>1):
                                A[i,i-2] = -1
                        else:
                                a[i] = a[i] + f0
                        A[i,i-1]=16
                        A[i,i]=-30
                        A[i,i+1]=16
                        if(i<n-3):
```

```
                A[i,i+2]=-1
            else:
                a[i] = a[i] + f1
    else:
        print("numero de colunas insuficientes")
    A = A.tocsr()
    return linalg_sparse.spsolve(A,a),time()-t
```

Ao final essa função retorna a resolução da equação, ou seja os valores de u_k, e o tempo de execução, esses resultados serão mostrados em tabelas na próxima sessão.

3 Resultados

Para obter os resultados desse problema foi usada a equação

$$-u''(x) = (cos(x) - sin^2(x))exp(cos(x)) = f(x) \tag{5}$$

que a solução analítica é conhecida

$$u(x) = exp(cos(x)) \tag{6}$$

dessa forma podemos calcular o erro, e assim aproximar a ordem p do erro ($O(h^p)$)

n	$h_n = \frac{(T-t_0)}{n}$	$\|e(T,h_n)\|_2$	ordem p_2
128	4.90874e-02	2.65068e-04	—
256	2.45437e-02	1.71826e-05	3.94735e+00
512	1.22718e-02	1.08473e-06	3.98554e+00
1024	6.13592e-03	6.79975e-08	3.99571e+00
2048	3.06796e-03	4.24739e-09	4.00083e+00
4096	1.53398e-03	2.31112e-10	4.19992e+00
8192	7.66990e-04	2.00766e-10	2.03076e-01
16384	3.83495e-04	9.31555e-10	-2.21413e+00
32768	1.91748e-04	3.60026e-09	-1.95039e+00

Table 1: Método de Euler aplicado ao Problema de Cauchy em $t = T$.

4 análise

n	$h_n = \frac{(T-t_0)}{n}$	$\|e(T,h_n)\|_\infty$	ordem p_∞
128	4.90874e-02	4.54881e-04	nan
256	2.45437e-02	2.96326e-05	3.94023e+00
512	1.22718e-02	1.87824e-06	3.97973e+00
1024	6.13592e-03	1.17977e-07	3.99280e+00
2048	3.06796e-03	7.38815e-09	3.99715e+00
4096	1.53398e-03	4.61880e-10	3.99962e+00
8192	7.66990e-04	2.58854e-10	8.35379e-01
16384	3.83495e-04	1.18161e-09	-2.19054e+00
32768	1.91748e-04	4.61141e-09	-1.96445e+00

Table 2: Método de Euler aplicado ao Problema de Cauchy em $t = T$.

n	$h_n = \frac{(T-t_0)}{n}$	*tempo A sparsa*(s)	*tempo A normal*(s)
128	4.90874e-02	7.48713e-02	2.58684e-01
256	2.45437e-02	9.59039e-03	1.16580e-02
512	1.22718e-02	2.22425e-02	2.84448e-02
1024	6.13592e-03	3.70302e-02	1.16141e-01
2048	3.06796e-03	3.46773e-02	4.44659e-01
4096	1.53398e-03	9.52613e-02	2.06000e+00
8192	7.66990e-04	1.19611e-01	1.21814e+01
16384	3.83495e-04	2.02301e-01	9.48454e+01
32768	1.91748e-04	4.00943e-01	Morto

Table 3: Método de Euler aplicado ao Problema de Cauchy em $t = T$.