Exercício Programa II

MAP3121 - Métodos Numéricos e Aplicações - 1º Semestre 2020

Nome	NUSP
Isabella Bologna Salomão	9267161
Renato de Oliveira Freitas	9837708

São Paulo, 10 de Julho de 2020

Tarefas

A partir dos pontos **p** fornecidos, são gerados os vetores $u_k(T, x_i)$ a partir do Método de Crank-Nicolson desenvolvido no EP1. Esses vetores são salvos numa matriz de dimensão n_f x N

```
#**** Cálculo dos Vetores uk *****
def matrix_uk(Item:Problem):
    '''
    Cria a matriz uk(T,xi)
    ----
      Aproximações da solução usando o método de Crank-Nicolson para cada fonte de calor em pk
    '''
    for s in range(0, Item.nf):
        crank_nicolson_method(Item, s)
        Item.uk[s] = Item.u[-1]
```

O Método de Crank-Nicolson utiliza o índice do ponto $\mathbf{p}_k$ para calcular a fonte de calor referente a esse ponto.

```
def heat_source(self, x, t, k):
        '''Heat source function f(x,t) = r(t) * gh(x)
        Args:
        -----
            x:   Posição
            t:   Tempo
            k:   Índice do vetor de pontos pk para o cálculo de gh(x)
        '''
        if ((self.p[k] - self.dx/2) <= x <= (self.p[k] + self.dx/2)):
            return self.r(t) * (1/self.dx)  # gh(x) = 1/h
        else:
            return 0
```

Dado o vetor $u_T(x_i)$ referente a cada item do enunciado, podemos montar e resolver o sistema do problema para recuperar os coeficientes a_k na função

```
def solve_normal_system(Item:Problem):
```

Primeiramente, o sistema normal é montado:

```
    #* Matrizes para o sistema da forma Ax = B
    A = np.zeros((Item.nf, Item.nf))
    B = np.zeros(Item.nf)
    #* Calculo produto interno <u,v> = ∑(i=1,N-1) u(xi)v(xi)
    for i in range(0, Item.nf):
        B[i] = np.vdot(Item.gabarito, Item.uk[i])
        for j in range(0, Item.nf):
            A[i][j] = np.vdot(Item.uk[i], Item.uk[j])
```

Com as matrizes **A** e **B** do sistema normal em mãos, fazemos a decomposição LDL^t do sistema de acordo com o seguinte algoritmo, sendo d_{jj} os valores da matriz diagonal D e l_{ij} os valores da matriz L:

$$d_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk}$$

$$l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_{kk} l_{jk} \right) \times \left(\frac{1}{d_{jj}} \right)$$

```
D = np.zeros((Item.nf, Item.nf))
L = np.zeros((Item.nf, Item.nf))
for i in range(0, Item.nf):
    for j in range(0, i+1):
        if i == j:
            L[i][i] = 1.0  # Diagonal principal de L = 1
            ld = 0.0
            for k in range(0, j):
                ld += L[j][k]**2 * D[k][k]
            D[j][j] = A[j][j] - ld
        else:
            ldl = 0.0
            for k in range(0, j):
                ldl += L[i][k] * D[k][k] * L[j][k]
            L[i][j] = (A[i][j] - ldl) / D[j][j]
```

Por último, basta resolver o sistema decomposto, salvando o resultado no vetor **a** de dimensão n_f:

```
#* Resolver o sistema
    '''
    Loop principal de resolução do sistema L.D.Lt * ak = b
    ---
    Eq:
        Lt.a = y -> a = y.L
        L.D.y = b
    Solve:
        yi = (bi -  Σ(k=1,i-1) Lik * (yk * dkk) ) / dii (I)     , i = 1...n
        ai = yi -  Σ(k=i+1,n) lki . ak       (II)    , i = n...1
    '''
    # (I):
    y = np.zeros(Item.nf)
    for i in range(0, Item.nf):
        sum_y = 0.0
        for k in range(0, i):
            sum_y += (L[i][k] * (y[k] * D[k][k]))
        y[i] = ( B[i] - sum_y)/ D[i][i]

    # (II):
    for i in range(Item.nf-1, -1, -1):
        sum_x = 0.0
        for k in range(i+1, Item.nf): # só entra em i<nf
            sum_x += L[k][i] * Item.a[k]
        Item.a[i] = (y[i] - sum_x)
```

Resultados dos Testes

a) N = 128, n_f = 1 e p_1 = 0.35

Para esse teste, precisamos apenas construir o vetor $u_1(T, x_i)$, com i = 1, ... N-1 através do método de Crank-Nicolson. O vetor encontrado foi o seguinte:

```
[0.         0.04335124 0.08671629 0.13010894 0.17354301 0.2170323
  0.26059063 0.30423181 0.34796966 0.391818   0.43579066 0.47990147
  0.52416428 0.56859292 0.61320127 0.65800318 0.70301253 0.74824321
  0.7937091  0.83942412 0.88540217 0.93165717 0.97820304 1.02505371
  1.07222311 1.11972526 1.1675742  1.2157841  1.26436915 1.31334347
  1.36272087 1.41251466 1.46273756 1.51340235 1.56452356 1.6161201
  1.66821691 1.72084028 1.77399953 1.8276539  1.88169735 1.93607125
  1.99118273 2.0484541  2.10836906 2.15319477 2.12294144 2.07760455
  2.03492259 1.99441757 1.95467284 1.91528697 1.87632428 1.83789649
  1.80005007 1.76278142 1.72606993 1.68989641 1.65424767 1.61911492
  1.58449113 1.55036934 1.51674203 1.48360115 1.45093839 1.41874536
  1.38701374 1.35573531 1.32490193 1.29450552 1.26453806 1.23499154
  1.20585796 1.17712937 1.14879782 1.12085543 1.0932943  1.06610659
  1.0392845  1.01282024 0.98670606 0.96093425 0.93549712 0.91038701
  0.8855963  0.8611174  0.83694273 0.81306477 0.789476   0.76616895
  0.74313617 0.72037024 0.69786376 0.67560938 0.65359976 0.63182758
  0.61028556 0.58896644 0.56786298 0.54696798 0.52627425 0.50577463
  0.48546197 0.46532916 0.44536909 0.4255747  0.40593893 0.38645473
  0.36711509 0.34791301 0.3288415  0.3098936  0.29106234 0.2723408
  0.25372205 0.23519917 0.21676526 0.19841345 0.18013684 0.16192858
  0.1437818  0.12568964 0.10764528 0.08964187 0.07167257 0.05373055
  0.035809   0.01790109 0.        ]
```

Tomamos $u_T(x_i) = 7 * u_1(T, x_i)$, montamos o sistema normal como mostrado no enunciado, que será trivial da forma $164.17074432\, a_1 = 1149.19521025$, obtendo como resultado a_1 = 7.

b) N = 128, n_f = 4 e p = [0.15, 0.3, 0.7, 0.8]

Nesse item, o sistema normal encontrado após os cálculos dos vetores $u_k(T, x_i)$, $k = 1, ..., 4$ e utilizando $u_T(x_i) = 2.3 * u_1(T, x_i) + 3.7 * u_2(T, x_i) + 0.3 * u_3(T, x_i) + 4.2 * u_4(T, x_i)$ será da forma:

$$\begin{bmatrix} 58.47728592 & 83.93738497 & 51.91311038 & 36.29935268 \\ 83.93738497 & 141.67298984 & 99.69442754 & 70.26089614 \\ 51.91311038 & 99.69442754 & 141.67298984 & 109.78573787 \\ 36.29935268 & 70.26089614 & 109.78573787 & 91.21508692 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 613.09729641 \\ 1042.2501399 \\ 991.87153181 \\ 759.49291332 \end{bmatrix}$$

Fazendo a decomposição LDL^t da matriz A simétrica anterior, encontramos:

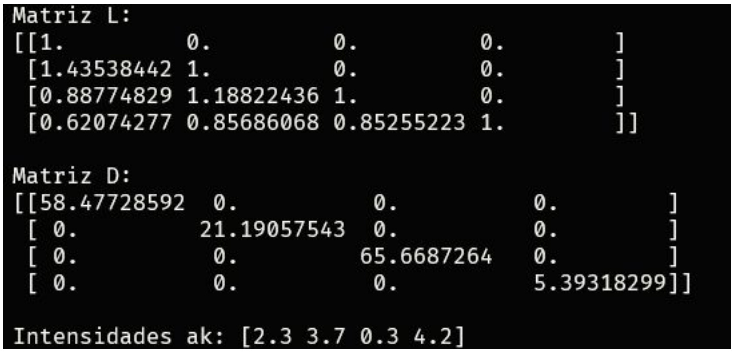
$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1.43538442 & 1 & 0 & 0 \\ 0.88774829 & 1.18822436 & 1 & 0 \\ 0.62074277 & 0.85686068 & 0.85255223 & 1 \end{bmatrix}$$

$$D = \begin{bmatrix} 58.47728592 & 0 & 0 & 0 \\ 0 & 21.19057543 & 0 & 0 \\ 0 & 0 & 65.6687264 & 0 \\ 0 & 0 & 0 & 5.39318299 \end{bmatrix}$$

Resolvendo esse sistema, chegamos a

$$a_k = \begin{bmatrix} 2.3 & 3.7 & 0.3 & 4.2 \end{bmatrix}$$

A imagem abaixo mostra esse mesmo resultado como saída do programa.



```
Matriz L:
[[1.         0.         0.         0.        ]
 [1.43538442 1.         0.         0.        ]
 [0.88774829 1.18822436 1.         0.        ]
 [0.62074277 0.85686068 0.85255223 1.        ]]

Matriz D:
[[58.47728592  0.          0.          0.        ]
 [ 0.         21.19057543  0.          0.        ]
 [ 0.          0.         65.6687264   0.        ]
 [ 0.          0.          0.          5.39318299]]

Intensidades ak: [2.3 3.7 0.3 4.2]
```

c) Arquivo com p = [0.15, 0.2, 0.3, 0.35, 0.5, 0.6, 0.7, 0.73, 0.85, 0.9]

Neste item, é necessário ler o arquivo que contém a localização dos pontos **p** e os valores de $u_T(x)$.

```
uT = []
with open('test.txt') as f:
    first_line = f.readline()
    l1 = first_line.split('       ')
    p = np.zeros(len(l1))
    for i in range(0,len(l1)):
        p[i] = float(l1[i]) # vetor com os pontos para a fonte de calor
    for line in f:
        uT.append(float(line)) # uT(xi)
```

De acordo com o valor **N** fornecido pelo usuário, os valores são selecionados para serem compatíveis com o tamanho da malha.

```
    def exact_solution(self, uT:np.ndarray):
        coef = int(2048/self.N)
        for i in range(0, self.N):
            self.gabarito[i] = uT[i*coef]
```

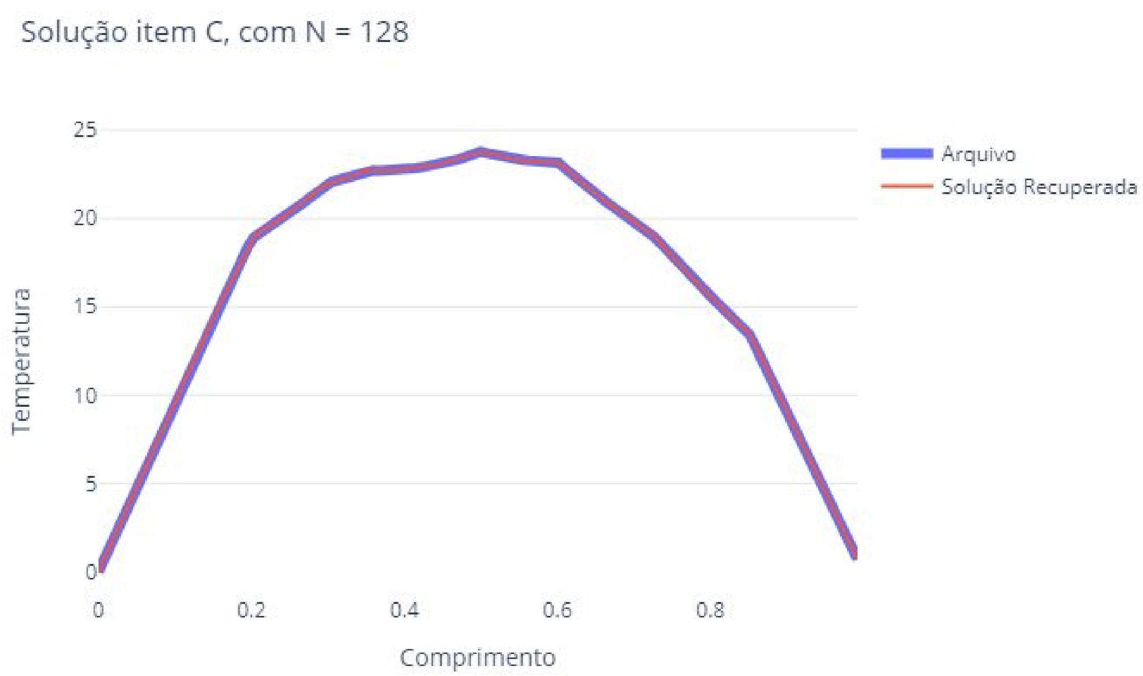
Com os dados selecionados corretamente, basta montar e resolver o sistema de acordo com os métodos citados anteriormente.

a_k	N = 128	N = 256	N = 512	N = 1024	N = 2048
a_1	1.20912318	0.90450103	0.92868838	1.00728132	1.0
a_2	4.83925872	5.07757264	5.05370784	4.99244301	5.0
a_3	1.88724086	2.1008536	2.04370105	1.98587673	2.0
a_4	1.58339993	1.41415569	1.46767067	1.51325847	1.5
a_5	2.21450405	2.22924501	2.19676333	2.19269284	2.2
a_6	3.12129478	3.10461386	3.09113117	3.09515288	3.1
a_7	0.37734029	0.5094526	0.63758752	0.65232665	0.6
a_8	1.49234829	1.38650879	1.27168722	1.25378989	1.3
a_9	3.9751388	3.94987865	3.87809487	3.87966706	3.9
a_{10}	0.40414515	0.4189313	0.53055678	0.52973663	0.5

Para este item, calculamos também o erro quadrático entre a solução calculada e os valores medidos.

```python
#***** Erro Quadrático *****
def quatratic_error(Item: Problem):
    '''
    Calculo do Erro Quadratico
    '''
    sum_mmq = 0
    for i in range(1, Item.N - 1):
        calc_solution = 0.0
        for k in range(0, Item.nf):
            calc_solution += Item.a[k] * Item.uk[k][i]
        sum_mmq += (Item.gabarito[i]- calc_solution) ** 2
    return math.sqrt(Item.dx * sum_mmq)
```

	N = 128	N = 256	N = 512	N = 1024	N = 2048
Erro	0.0244532761 96356885	0.0123634386 12941308	0.0084766276 66325707	0.0037793102 80899021	2.5284042743 954526e-12



d) Dados com ruído aleatório (1 + 0.01r), r ∈ [-1, 1]

Neste item, cada valor lido do arquivo é multiplicado por um ruído aleatório da forma

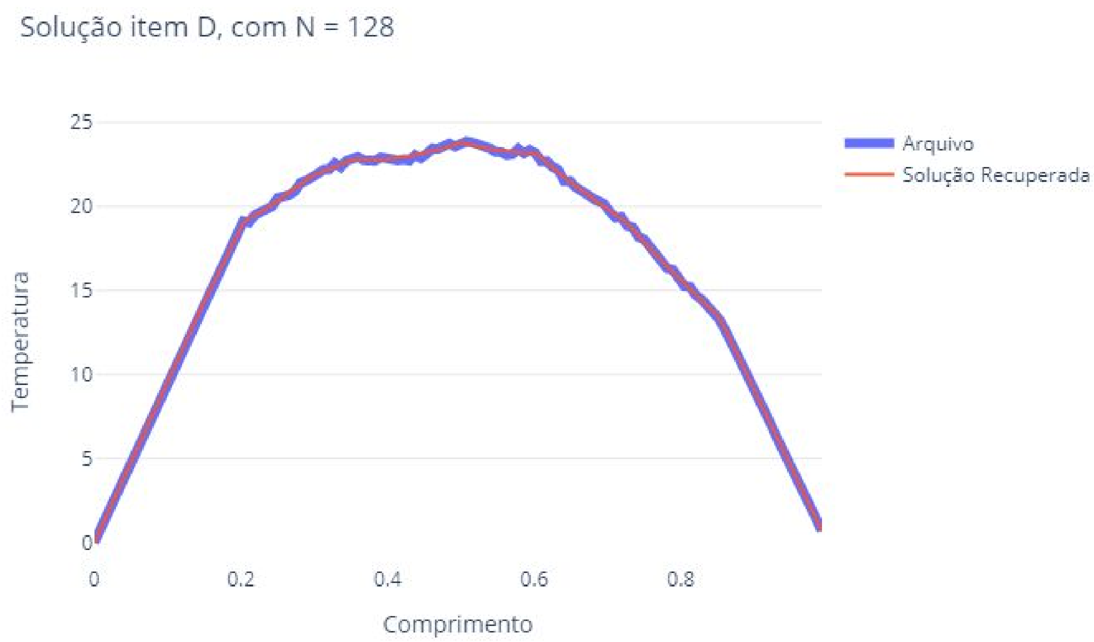
(1 + 0.01r), r ∈ [-1, 1].

```
#* Leitura dos dados do arquivo .txt
uT = []
with open('test.txt') as f:
    first_line = f.readline()
    l1=first_line.split('        ')
    p = np.zeros(len(l1))
    for i in range(0, len(l1)):
        p[i] = float(l1[i]) # vetor com os pontos para a fonte de calor

    for line in f:
        ruido = 1 + 0.01 * ((np.random.random() - 0.5) * 2)  # Ruido = 1 + 0.01r, r ∈ [-1, 1]
        uT.append(ruido * float(line))  # uT(xi) * ruido
```

A resolução do problema é análoga ao item C.

a_k	N = 128	N = 256	N = 512	N = 1024	N = 2048
a_1	1.2577034	0.92269885	0.99132706	0.96725384	0.98492983
a_2	4.7196944	5.06475758	4.9438593	5.00533958	4.99997983
a_3	2.13147597	2.14554153	2.19423591	2.05461108	2.01821472
a_4	1.41338888	1.34249031	1.35835001	1.44626695	1.50131687
a_5	2.24903429	2.28957939	2.2503154	2.25313286	2.19226088
a_6	3.02328035	3.0327376	3.01178409	3.01083866	3.08391821
a_7	0.54178883	0.48972254	0.73517794	0.75833818	0.59507066
a_8	1.39994263	1.4734019	1.23481792	1.18789538	1.34042467
a_9	3.98568176	3.91066463	0.83088584	3.88620726	3.87433226
a_{10}	0.40491855	0.42850515	0.55912187	0.52169813	0.51360764



	N = 128	N = 256	N = 512	N = 1024	N = 2048
Erro	0.10294580272838558	0.10532150210963113	0.10371272447711369	0.10407586545178191	0.1037148742913366