

**MAP 2320 – MÉTODOS NUMÉRICOS EM EQUAÇÕES  
DIFERENCIAIS II**

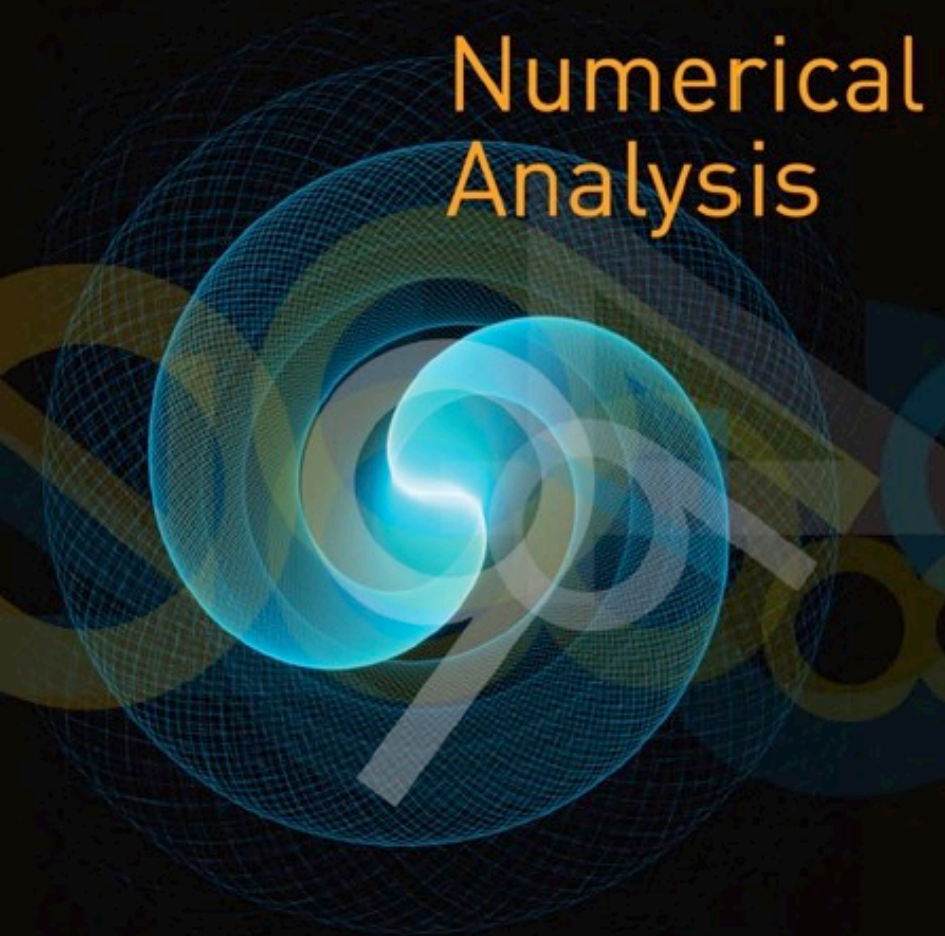
**2º Semestre - 2019**

**Prof. Dr. Luis Carlos de Castro Santos**

lsantos@ime.usp.br

Richard L. Burden  
J. Douglas Faires

# Numerical Analysis

An abstract graphic featuring a glowing blue and green spiral that winds inward towards the center. The spiral is set against a dark background with a faint, grid-like pattern. The colors of the spiral transition from a bright blue at the center to a greenish-yellow towards the outer edges.

Ninth Edition

# Numerical Analysis

NINTH EDITION

**Richard L. Burden**

*Youngstown State University*

**J. Douglas Faires**

*Youngstown State University*

---

## 12 Numerical Solutions to Partial Differential Equations 713

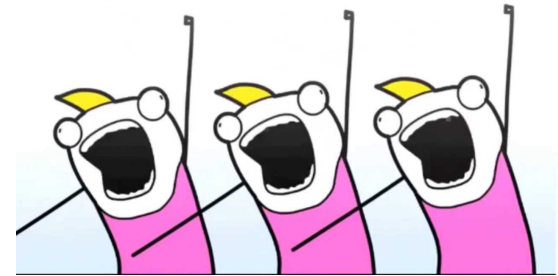
12.1 Elliptic Partial Differential Equations 716

12.2 Parabolic Partial Differential Equations 725

12.3 Hyperbolic Partial Differential Equations 739

12.4 An Introduction to the Finite-Element Method 746

12.5 Survey of Methods and Software 760



O que queremos ?

- O menor custo computacional !

Porque não podemos ?

- Erro de discretização e Limite de Estabilidade

O que faremos?

Buscar métodos incondicionalmente estáveis !

$$D_- u(\bar{x}) \equiv \frac{u(\bar{x}) - u(\bar{x} - h)}{h}.$$

## Backward-Difference Method

To obtain a method that is **unconditionally stable**, we consider an implicit-difference method that results from using the backward-difference quotient for  $(\partial u / \partial t)(x_i, t_j)$  in the form

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_j) - u(x_i, t_{j-1}))}{k} + \frac{k}{2} \frac{\partial^2 u}{\partial t^2}(x_i, \mu_j),$$

where  $\mu_j$  is in  $(t_{j-1}, t_j)$ . Substituting this equation, together with Eq. (12.8) for  $\partial^2 u / \partial x^2$ , into the partial differential equation gives

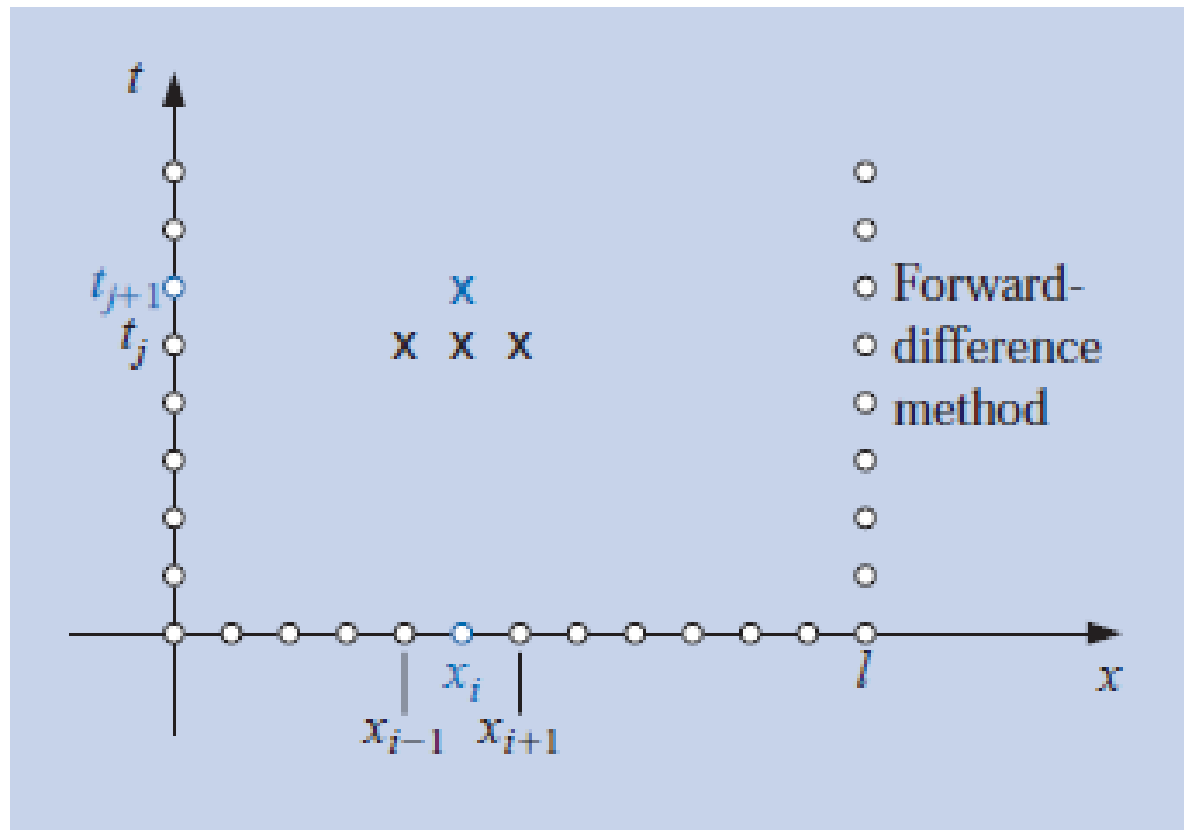
$$\begin{aligned} \frac{u(x_i, t_j) - u(x_i, t_{j-1}))}{k} - \alpha^2 \frac{u(x_{i+1}, t_j) - 2u(x_i, t_j) + u(x_{i-1}, t_j))}{h^2} \\ = -\frac{k}{2} \frac{\partial^2 u}{\partial t^2}(x_i, \mu_j) - \alpha^2 \frac{h^2}{12} \frac{\partial^4 u}{\partial x^4}(\xi_i, t_j), \end{aligned}$$

for some  $\xi_i \in (x_{i-1}, x_{i+1})$ .

$$D^2 u(\bar{x}) = \frac{1}{h^2} [u(\bar{x} - h) - 2u(\bar{x}) + u(\bar{x} + h)]$$



Since the boundary and initial conditions associated with the problem give information at the circled mesh points, the figure shows that no explicit procedures can be used to solve Eq. (12.12). Recall that in the Forward-Difference method (see Figure 12.10), approximations at  $(x_{i-1}, t_{j-1})$ ,  $(x_i, t_{j-1})$ , and  $(x_{i+1}, t_{j-1})$  were used to find the approximation at  $(x_i, t_j)$ . So an explicit method could be used to find the approximations, based on the information from the initial and boundary conditions.



If we again let  $\lambda$  denote the quantity  $\alpha^2(k/h^2)$ , the Backward-Difference method becomes

$$(1 + 2\lambda)w_{ij} - \lambda w_{i+1,j} - \lambda w_{i-1,j} = w_{i,j-1},$$

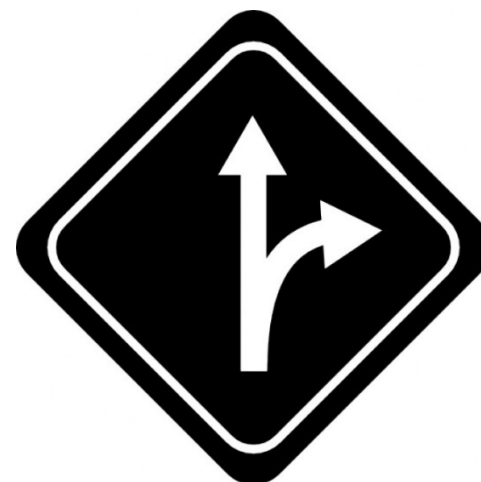
for each  $i = 1, 2, \dots, m-1$  and  $j = 1, 2, \dots$ . Using the knowledge that  $w_{i,0} = f(x_i)$ , for each  $i = 1, 2, \dots, m-1$  and  $w_{m,j} = w_{0,j} = 0$ , for each  $j = 1, 2, \dots$ , this difference method has the matrix representation:

$$\begin{bmatrix} (1+2\lambda) & -\lambda & 0 & \cdots & 0 \\ & -\lambda & \ddots & \ddots & \\ & 0 & \ddots & \ddots & -\lambda \\ & 0 & \ddots & 0 & -\lambda & (1+2\lambda) \end{bmatrix} \begin{bmatrix} w_{1,j} \\ w_{2,j} \\ \vdots \\ w_{m-1,j} \end{bmatrix} = \begin{bmatrix} w_{1,j-1} \\ w_{2,j-1} \\ \vdots \\ w_{m-1,j-1} \end{bmatrix}, \quad (12.13)$$

or  $A\mathbf{w}^{(j)} = \mathbf{w}^{(j-1)}$ , for each  $i = 1, 2, \dots$



Hence, we must now solve a linear system to obtain  $\mathbf{w}^{(j)}$  from  $\mathbf{w}^{(j-1)}$ . However  $\lambda > 0$ , so the matrix  $A$  is positive definite and strictly diagonally dominant, as well as being tridiagonal. We can consequently use either the Crout Factorization Algorithm 6.7 or the SOR Algorithm 7.3 to solve this system. Algorithm 12.2 solves (12.13) using Crout factorization, which is acceptable unless  $m$  is large. In this algorithm we assume, for stopping purposes, that a bound is given for  $t$ .



## Tridiagonal Matrices

Matrices of bandwidth 3 occurring when  $p = q = 2$  are called **tridiagonal** because they have the form

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & & \\ 0 & a_{32} & a_{33} & a_{34} & \\ \vdots & & & & \\ 0 & \cdots & 0 & a_{n,n-1} & a_{nn} \end{bmatrix}.$$

The factorization algorithms can be simplified considerably in the case of band matrices because a large number of zeros appear in these matrices in regular patterns. It is particularly interesting to observe the form the Crout or Doolittle method assumes in this case.

To illustrate the situation, suppose a tridiagonal matrix  $A$  can be factored into the triangular matrices  $L$  and  $U$ . Then  $A$  has at most  $(3n - 2)$  nonzero entries. Then there are only  $(3n - 2)$  conditions to be applied to determine the entries of  $L$  and  $U$ , provided, of course, that the zero entries of  $A$  are also obtained.

Suppose that the matrices  $L$  and  $U$  also have tridiagonal form, that is,

$$L = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & & \\ 0 & & \ddots & \\ \vdots & & & \\ 0 & \cdots & 0 & l_{n,n-1} & l_{nn} \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 1 & u_{12} & 0 & \cdots & 0 \\ 0 & 1 & & & \\ \vdots & & \ddots & & \\ 0 & \cdots & 0 & u_{n-1,n} & 1 \end{bmatrix}.$$

There are  $(2n - 1)$  undetermined entries of  $L$  and  $(n - 1)$  undetermined entries of  $U$ , which totals  $(3n - 2)$ , the number of possible nonzero entries of  $A$ . The 0 entries of  $A$  are obtained automatically.

The multiplication involved with  $A = LU$  gives, in addition to the 0 entries,

$$a_{11} = l_{11};$$

$$a_{i,i-1} = l_{i,i-1}, \quad \text{for each } i = 2, 3, \dots, n; \quad (6.13)$$

$$a_{ii} = l_{i,i-1}u_{i-1,i} + l_{ii}, \quad \text{for each } i = 2, 3, \dots, n; \quad (6.14)$$

and

$$a_{i,i+1} = l_{ii}u_{i,i+1}, \quad \text{for each } i = 1, 2, \dots, n-1. \quad (6.15)$$

A solution to this system is found by first using Eq. (6.13) to obtain all the nonzero off-diagonal terms in  $L$  and then using Eqs. (6.14) and (6.15) to alternately obtain the remainder of the entries in  $U$  and  $L$ . Once an entry  $L$  or  $U$  is computed, the corresponding entry in  $A$  is not needed. So the entries in  $A$  can be overwritten by the entries in  $L$  and  $U$  with the result that no new storage is required.

Algorithm 6.7 solves an  $n \times n$  system of linear equations whose coefficient matrix is tridiagonal. This algorithm requires only  $(5n - 4)$  multiplications/divisions and  $(3n - 3)$  additions/subtractions. Consequently, it has considerable computational advantage over the methods that do not consider the tridiagonality of the matrix.

**Example 5** Determine the Crout factorization of the symmetric tridiagonal matrix

$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix},$$

and use this factorization to solve the linear system

$$\begin{aligned} 2x_1 - x_2 &= 1, \\ -x_1 + 2x_2 - x_3 &= 0, \\ -x_2 + 2x_3 - x_4 &= 0, \\ -x_3 + 2x_4 &= 1. \end{aligned}$$

**Solution** The  $LU$  factorization of  $A$  has the form

$$\begin{aligned}
 A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ 0 & a_{32} & a_{33} & a_{34} \\ 0 & 0 & a_{43} & a_{44} \end{bmatrix} &= \begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ 0 & l_{32} & l_{33} & 0 \\ 0 & 0 & l_{43} & l_{44} \end{bmatrix} \begin{bmatrix} 1 & u_{12} & 0 & 0 \\ 0 & 1 & u_{23} & 0 \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
 &= \begin{bmatrix} l_{11} & l_{11}u_{12} & 0 & 0 \\ l_{21} & l_{22} + l_{21}u_{12} & l_{22}u_{23} & 0 \\ 0 & l_{32} & l_{33} + l_{32}u_{23} & l_{33}u_{34} \\ 0 & 0 & l_{43} & l_{44} + l_{43}u_{34} \end{bmatrix}
 \end{aligned}$$

Thus

$$\begin{array}{ll}
 a_{11} : & 2 = l_{11} \implies l_{11} = 2, & a_{12} : & -1 = l_{11}u_{12} \implies u_{12} = -\frac{1}{2}, \\
 a_{21} : & -1 = l_{21} \implies l_{21} = -1, & a_{22} : & 2 = l_{22} + l_{21}u_{12} \implies l_{22} = -\frac{3}{2}, \\
 a_{23} : & -1 = l_{22}u_{23} \implies u_{23} = -\frac{2}{3}, & a_{32} : & -1 = l_{32} \implies l_{32} = -1, \\
 a_{33} : & 2 = l_{33} + l_{32}u_{23} \implies l_{33} = \frac{4}{3}, & a_{34} : & -1 = l_{33}u_{34} \implies u_{34} = -\frac{3}{4}, \\
 a_{43} : & -1 = l_{43} \implies l_{43} = -1, & a_{44} : & 2 = l_{44} + l_{43}u_{34} \implies l_{44} = \frac{5}{4}.
 \end{array}$$

This gives the Crout factorization

$$A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & \frac{3}{2} & 0 & 0 \\ 0 & -1 & \frac{4}{3} & 0 \\ 0 & 0 & -1 & \frac{5}{4} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -\frac{2}{3} & 0 \\ 0 & 0 & 1 & -\frac{3}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix} = LU.$$



Solving the system

$$Lz = \begin{bmatrix} 2 & 0 & 0 & 0 \\ -1 & \frac{3}{2} & 0 & 0 \\ 0 & -1 & \frac{4}{3} & 0 \\ 0 & 0 & -1 & \frac{5}{4} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{gives} \quad \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ 1 \end{bmatrix},$$

and then solving

$$Ux = \begin{bmatrix} 1 & -\frac{1}{2} & 0 & 0 \\ 0 & 1 & -\frac{2}{3} & 0 \\ 0 & 0 & 1 & -\frac{3}{4} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ 1 \end{bmatrix} \quad \text{gives} \quad \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \quad \blacksquare$$

The Crout Factorization Algorithm can be applied whenever  $l_{ii} \neq 0$  for each  $i = 1, 2, \dots, n$ . Two conditions, either of which ensure that this is true, are that the coefficient matrix of the system is positive definite or that it is strictly diagonally dominant.

To solve the  $n \times n$  linear system

$$\begin{array}{lll} E_1 : & a_{11}x_1 + a_{12}x_2 & = a_{1,n+1}, \\ E_2 : & a_{21}x_1 + a_{22}x_2 + a_{23}x_3 & = a_{2,n+1}, \\ & \vdots & \vdots \\ E_{n-1} : & a_{n-1,n-2}x_{n-2} + a_{n-1,n-1}x_{n-1} + a_{n-1,n}x_n & = a_{n-1,n+1}, \\ E_n : & a_{n,n-1}x_{n-1} + a_{nn}x_n & = a_{n,n+1}, \end{array}$$

which is assumed to have a unique solution:

**INPUT** the dimension  $n$ ; the entries of  $A$ .

**OUTPUT** the solution  $x_1, \dots, x_n$ .

*(Steps 1–3 set up and solve  $Lz = b$ .)*

**Step 1** Set  $l_{11} = a_{11}$ ;  
 $u_{12} = a_{12}/l_{11}$ ;  
 $z_1 = a_{1,n+1}/l_{11}$ .

**Step 2** For  $i = 2, \dots, n-1$  set  $l_{i,i-1} = a_{i,i-1}$ ; ( *$i$ th row of  $L$ .*)  
 $l_{ii} = a_{ii} - l_{i,i-1}u_{i-1,i}$ ;  
 $u_{i,j+1} = a_{i,j+1}/l_{ii}$ ; ( *$(i+1)$ th column of  $U$ .*)  
 $z_i = (a_{i,n+1} - l_{i,i-1}z_{i-1})/l_{ii}$ .

**Step 3** Set  $l_{n,n-1} = a_{n,n-1}$ ; ( *$n$ th row of  $L$ .*)  
 $l_{nn} = a_{nn} - l_{n,n-1}u_{n-1,n}$ .  
 $z_n = (a_{n,n+1} - l_{n,n-1}z_{n-1})/l_{nn}$ .

*(Steps 4 and 5 solve  $Ux = z$ .)*

**Step 4** Set  $x_n = z_n$ .

**Step 5** For  $i = n-1, \dots, 1$  set  $x_i = z_i - u_{i,i+1}x_{i+1}$ .

**Step 6** **OUTPUT**  $(x_1, \dots, x_n)$ ;  
**STOP.**



## Heat Equation Backward-Difference

INPUT endpoint  $l$ ; maximum time  $T$ ; constant  $\alpha$ ; integers  $m \geq 3, N \geq 1$ .

OUTPUT approximations  $w_{i,j}$  to  $u(x_i, t_j)$  for each  $i = 1, \dots, m-1$  and  $j = 1, \dots, N$ .

Step 1 Set  $h = l/m$ ;  
 $k = T/N$ ;  
 $\lambda = \alpha^2 k / h^2$ .

Step 2 For  $i = 1, \dots, m-1$  set  $w_i = f(ih)$ . (Initial values.)  
 (Steps 3–11 solve a tridiagonal linear system using Algorithm 6.7.)

Step 3 Set  $l_1 = 1 + 2\lambda$ ;  
 $u_1 = -\lambda/l_1$ .

Step 4 For  $i = 2, \dots, m-2$  set  $l_i = 1 + 2\lambda + \lambda u_{i-1}$ ;  
 $u_i = -\lambda/l_i$ .

Step 5 Set  $l_{m-1} = 1 + 2\lambda + \lambda u_{m-2}$ .

Step 6 For  $j = 1, \dots, N$  do Steps 7–11.

Step 7 Set  $t = jk$ ; (Current  $t_j$ .)  
 $z_1 = w_1/l_1$ .

Step 8 For  $i = 2, \dots, m-1$  set  $z_i = (w_i + \lambda z_{i-1})/l_i$ .

Step 9 Set  $w_{m-1} = z_{m-1}$ .

Step 10 For  $i = m-2, \dots, 1$  set  $w_i = z_i - u_i w_{i+1}$ .

Step 11 OUTPUT ( $t$ ); (Note:  $t = t_j$ .)  
 For  $i = 1, \dots, m-1$  set  $x = ih$ ;  
 OUTPUT ( $x, w_i$ ). (Note:  $w_i = w_{i,j}$ .)

Step 12 STOP. (The procedure is complete.)



Use the Backward-Difference method (Algorithm 12.2) with  $h = 0.1$  and  $k = 0.01$  to approximate the solution to the heat equation

$$\frac{\partial u}{\partial t}(x, t) - \frac{\partial^2 u}{\partial x^2}(x, t) = 0, \quad 0 < x < 1, \quad 0 < t,$$

subject to the constraints

$$u(0, t) = u(1, t) = 0, \quad 0 < t, \quad u(x, 0) = \sin \pi x, \quad 0 \leq x \leq 1.$$

**Solution** This problem was considered in Example 1 where we found that choosing  $h = 0.1$  and  $k = 0.0005$  gave quite accurate results. However, with the values in this example,  $h = 0.1$  and  $k = 0.01$ , the results were exceptionally poor. To demonstrate the unconditional stability of the Backward-Difference method, we will use  $h = 0.1$  and  $k = 0.01$  and again compare  $w_{i,50}$  to  $u(x_i, 0.5)$ , where  $i = 0, 1, \dots, 10$ .

(a) Forward-Difference method with  $h = 0.1, k = 0.0005$  and  $\lambda = (1)^2(0.0005/(0.1)^2) = 0.05$  gives the results in the third column of Table 12.3. As can be seen these results are quite accurate.

Table 12.3



$x_i$	$u(x_i, 0.5)$	$w_{i,1000}$ $k = 0.0005$	$ u(x_i, 0.5) - w_{i,1000} $	$w_{i,50}$ $k = 0.01$	$ u(x_i, 0.5) - w_{i,50} $
0.0	0	0		0	
0.1	0.00222241	0.00228652	$6.411 \times 10^{-5}$	$8.19876 \times 10^7$	$8.199 \times 10^7$
0.2	0.00422728	0.00434922	$1.219 \times 10^{-4}$	$-1.55719 \times 10^8$	$1.557 \times 10^8$
0.3	0.00581836	0.00598619	$1.678 \times 10^{-4}$	$2.13833 \times 10^8$	$2.138 \times 10^8$
0.4	0.00683989	0.00703719	$1.973 \times 10^{-4}$	$-2.50642 \times 10^8$	$2.506 \times 10^8$
0.5	0.00719188	0.00739934	$2.075 \times 10^{-4}$	$2.62685 \times 10^8$	$2.627 \times 10^8$
0.6	0.00683989	0.00703719	$1.973 \times 10^{-4}$	$-2.49015 \times 10^8$	$2.490 \times 10^8$
0.7	0.00581836	0.00598619	$1.678 \times 10^{-4}$	$2.11200 \times 10^8$	$2.112 \times 10^8$
0.8	0.00422728	0.00434922	$1.219 \times 10^{-4}$	$-1.53086 \times 10^8$	$1.531 \times 10^8$
0.9	0.00222241	0.00228652	$6.511 \times 10^{-5}$	$8.03604 \times 10^7$	$8.036 \times 10^7$
1.0	0	0		0	

1000 steps

50 steps

(b) Forward-Difference method with  $h = 0.1, k = 0.01$  and  $\lambda = (1)^2(0.01/(0.1)^2) = 1$  gives the results in the fifth column of Table 12.3. As can be seen from the sixth column, these results are worthless.

RECORDANDO A AULA ANTERIOR

The results listed in Table 12.4 have the same values of  $h$  and  $k$  as those in the fifth and sixth columns of Table 12.3, which illustrates the stability of this method. ■

**Table 12.4**

$x_i$	$w_{i,50}$	$u(x_i, 0.5)$	$ w_{i,50} - u(x_i, 0.5) $
0.0	0	0	
0.1	0.00289802	0.00222241	$6.756 \times 10^{-4}$
0.2	0.00551236	0.00422728	$1.285 \times 10^{-3}$
0.3	0.00758711	0.00581836	$1.769 \times 10^{-3}$
0.4	0.00891918	0.00683989	$2.079 \times 10^{-3}$
0.5	0.00937818	0.00719188	$2.186 \times 10^{-3}$
0.6	0.00891918	0.00683989	$2.079 \times 10^{-3}$
0.7	0.00758711	0.00581836	$1.769 \times 10^{-3}$
0.8	0.00551236	0.00422728	$1.285 \times 10^{-3}$
0.9	0.00289802	0.00222241	$6.756 \times 10^{-4}$
1.0	0	0	



50 tridiagonal  
solutions

## Forward Difference

Para cada ponto interno  $m-2$  realizam-se cerca de 3 multiplicações

Para cada nível de tempo  $3*(m-2)$  multiplicações, portanto para atingir o instante  $T$ , são necessárias  $3*(m-2)*T/k$  multiplicações.

O método é limitado em estabilidade logo  $k$  deve respeitar

$$k \leq \frac{h^2}{2\alpha^2}$$

## Backward Difference

Para cada nível de tempo  $5*(m-4)$  multiplicações/divisões, portanto para atingir o instante  $T$ , são necessárias  $5*(m-4)*T/k$  multiplicações/divisões.

Apesar do custo maior (no caso ligeiramente maior) o método é incondicionalmente estável.

Porque





The reason that the Backward-Difference method does not have the stability problems of the Forward-Difference method can be seen by analyzing the eigenvalues of the matrix  $A$ . For the Backward-Difference method (see Exercise 14), the eigenvalues are

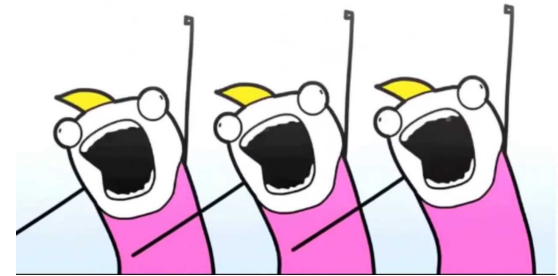
$$\mu_i = 1 + 4\lambda \left[ \sin \left( \frac{i\pi}{2m} \right) \right]^2, \quad \text{for each } i = 1, 2, \dots, m-1.$$

Since  $\lambda > 0$ , so we have  $\mu_i > 1$  for all  $i = 1, 2, \dots, m-1$ . Since the eigenvalues of  $A^{-1}$  are the reciprocals of those of  $A$ , the spectral radius of  $A^{-1}$ ,  $\rho(A^{-1}) < 1$ . This implies that  $A^{-1}$  is a convergent matrix.

An error  $\mathbf{e}^{(0)}$  in the initial data produces an error  $(A^{-1})^n \mathbf{e}^{(0)}$  at the  $n$ th step of the Backward-Difference method. Since  $A^{-1}$  is convergent,

$$\lim_{n \rightarrow \infty} (A^{-1})^n \mathbf{e}^{(0)} = \mathbf{0}.$$

So the method is stable, independent of the choice of  $\lambda = \alpha^2(k/h^2)$ . In the terminology of Chapter 5, we call the Backward-Difference method an **unconditionally stable** method.



O que queremos ?

- O menor custo computacional !

Porque não podemos ?

- Erro de discretização e Limite de Estabilidade



O que faremos?

Buscar métodos incondicionalmente estáveis !




The local truncation error for the method is of order  $O(k + h^2)$ , provided the solution of the differential equation satisfies the usual differentiability conditions. In this case, the method converges to the solution of the partial differential equation with this same rate of convergence (see [IK], p. 508).

The weakness of the Backward-Difference method results from the fact that the local truncation error has one of order  $O(h^2)$ , and another of order  $O(k)$ . This requires that time intervals be made much smaller than the  $x$ -axis intervals. It would clearly be desirable to have a procedure with local truncation error of order  $O(k^2 + h^2)$ . The first step in this direction is to use a difference equation that has  $O(k^2)$  error for  $u_t(x, t)$  instead of those we have used previously, whose error was  $O(k)$ . This can be done by using the Taylor series in  $t$  for the function  $u(x, t)$  at the point  $(x_i, t_j)$  and evaluating at  $(x_i, t_{j+1})$  and  $(x_i, t_{j-1})$  to obtain the Centered-Difference formula

$$\frac{\partial u}{\partial t}(x_i, t_j) = \frac{u(x_i, t_{j+1}) - u(x_i, t_{j-1}))}{2k} + \frac{k^2}{6} \frac{\partial^3 u}{\partial t^3}(x_i, \mu_j),$$

where  $\mu_j \in (t_{j-1}, t_{j+1})$ .



$$D_0 u(\bar{x}) \equiv \frac{u(\bar{x} + h) - u(\bar{x} - h)}{2h}$$

The difference method that results from substituting this and the usual difference quotient for  $(\partial^2 u / \partial x^2)$ , Eq. (12.8), into the differential equation is called **Richardson's method** and is given by

$$\frac{w_{i,j+1} - w_{i,j-1}}{2k} - \alpha^2 \frac{w_{i+1,j} - 2w_{ij} + w_{i-1,j}}{h^2} = 0. \quad (12.14)$$

This method has local truncation error of order  $O(k^2 + h^2)$ , but unfortunately, like the Forward-Difference method, it has serious stability problems (see Exercises 11 and 12).

**Incondicionalmente Instável !**

## Crank-Nicolson Method

A more rewarding method is derived by averaging the Forward-Difference method at the  $j$ th step in  $t$ ,

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} = 0,$$

which has local truncation error

$$\tau_F = \frac{k}{2} \frac{\partial^2 u}{\partial t^2}(x_i, \mu_j) + O(h^2),$$

and the Backward-Difference method at the  $(j + 1)$ st step in  $t$ ,

$$\frac{w_{i,j+1} - w_{i,j}}{k} - \alpha^2 \frac{w_{i+1,j+1} - 2w_{i,j+1} + w_{i-1,j+1}}{h^2} = 0,$$

which has local truncation error

$$\tau_B = -\frac{k}{2} \frac{\partial^2 u}{\partial t^2}(x_i, \hat{u}_j) + O(h^2).$$

If we assume that

$$\frac{\partial^2 u}{\partial t^2}(x_i, \hat{\mu}_j) \approx \frac{\partial^2 u}{\partial t^2}(x_i, \mu_j),$$

then the averaged-difference method,

$$\frac{w_{i,j+1} - w_{ij}}{k} - \frac{\alpha^2}{2} \left[ \frac{w_{i+1,j} - 2w_{ij} + w_{i-1,j}}{h^2} + \frac{w_{i+1,j+1} - 2w_{ij+1} + w_{i-1,j+1}}{h^2} \right] = 0,$$

has local truncation error of order  $O(k^2 + h^2)$ , provided, of course, that the usual differentiability conditions are satisfied.

This is known as the **Crank-Nicolson method** and is represented in the matrix form

$$A\mathbf{w}^{(j+1)} = B\mathbf{w}^{(j)}, \quad \text{for each } j = 0, 1, 2, \dots, \quad (12.15)$$

where

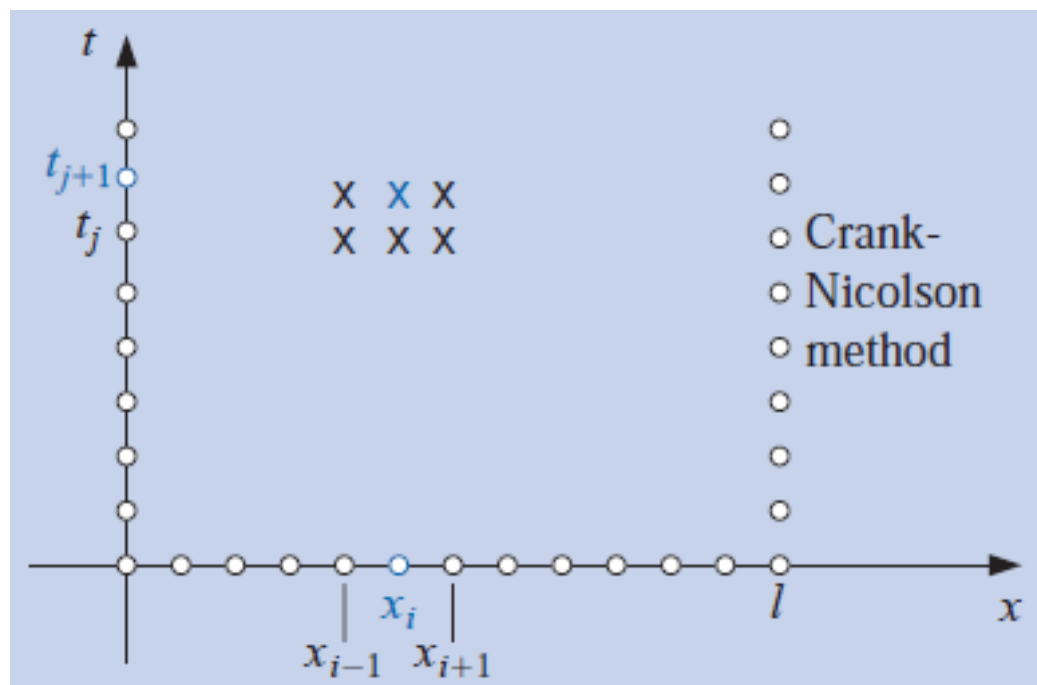
$$\lambda = \alpha^2 \frac{k}{h^2}, \quad \mathbf{w}^{(j)} = (w_{1j}, w_{2j}, \dots, w_{m-1j})^t,$$

and the matrices  $A$  and  $B$  are given by:

$$A = \begin{bmatrix} (1+\lambda) & -\frac{\lambda}{2} & 0 & \dots & 0 \\ -\frac{\lambda}{2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -\frac{\lambda}{2} & (1+\lambda) \end{bmatrix}$$

$$B = \begin{bmatrix} (1-\lambda) & \frac{\lambda}{2} & 0 & \dots & 0 \\ \frac{\lambda}{2} & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & \frac{\lambda}{2} & (1-\lambda) \end{bmatrix}$$

The nonsingular matrix  $A$  is positive definite, strictly diagonally dominant, and tridiagonal matrix. Either the Crout Factorization 6.7 or the SOR Algorithm 7.3 can be used to obtain  $w^{(j)}$  from  $w^{(j-1)}$ , for each  $j = 0, 1, 2, \dots$ . Algorithm 12.3 incorporates Crout factorization into the Crank-Nicolson technique. As in Algorithm 12.2, a finite length for the time interval must be specified to determine a stopping procedure. The verification that the Crank-Nicolson method is unconditionally stable and has order of convergence  $O(k^2 + h^2)$  can be found in [IK], pp. 508–512. A diagram showing the interaction of the nodes for determining an approximation at  $(x_i, t_j)$  is shown in Figure 12.11.





## Crank-Nicolson

INPUT endpoint  $l$ ; maximum time  $T$ ; constant  $\alpha$ ; integers  $m \geq 3, N \geq 1$ .

OUTPUT approximations  $w_{i,j}$  to  $u(x_i, t_j)$  for each  $i = 1, \dots, m-1$  and  $j = 1, \dots, N$ .

Step 1 Set  $h = l/m$ ;

$$k = T/N;$$

$$\lambda = \alpha^2 k / h^2;$$

$$w_m = 0.$$

Step 2 For  $i = 1, \dots, m-1$  set  $w_i = f(ih)$ . (Initial values.)

(Steps 3–11 solve a tridiagonal linear system using Algorithm 6.7.)

Step 3 Set  $l_1 = 1 + \lambda$ ;

$$u_1 = -\lambda / (2l_1).$$

Step 4 For  $i = 2, \dots, m-2$  set  $l_i = 1 + \lambda + \lambda u_{i-1} / 2$ ;

$$u_i = -\lambda / (2l_i).$$

Step 5 Set  $l_{m-1} = 1 + \lambda + \lambda u_{m-2} / 2$ .

Step 6 For  $j = 1, \dots, N$  do Steps 7–11.

Step 7 Set  $t = jk$ ; (Current  $t_j$ .)

$$z_1 = \left[ (1 - \lambda)w_1 + \frac{\lambda}{2}w_2 \right] / l_1.$$

Step 8 For  $i = 2, \dots, m-1$  set

$$z_i = \left[ (1 - \lambda)w_i + \frac{\lambda}{2}(w_{i+1} + w_{i-1} + z_{i-1}) \right] / l_i.$$

Step 9 Set  $w_{m-1} = z_{m-1}$ .

Step 10 For  $i = m-2, \dots, 1$  set  $w_i = z_i - u_i w_{i+1}$ .

Step 11 OUTPUT ( $t$ ); (Note:  $t = t_j$ .)

For  $i = 1, \dots, m-1$  set  $x = ih$ ;

OUTPUT ( $x, w_i$ ). (Note:  $w_i = w_{i,j}$ .)

Step 12 STOP. (The procedure is complete.)

Use the Crank-Nicolson method with  $h = 0.1$  and  $k = 0.01$  to approximate the solution to the problem

$$\frac{\partial u}{\partial t}(x, t) - \frac{\partial^2 u}{\partial x^2}(x, t) = 0, \quad 0 < x < 1 \quad 0 < t,$$

subject to the conditions

$$u(0, t) = u(1, t) = 0, \quad 0 < t,$$

and

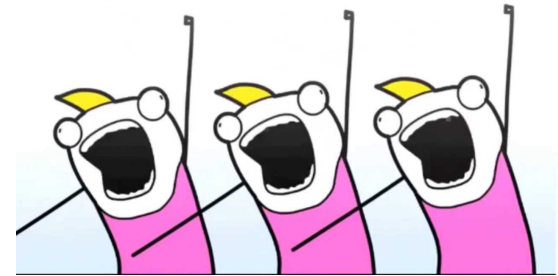
$$u(x, 0) = \sin(\pi x), \quad 0 \leq x \leq 1.$$

**Solution** Choosing  $h = 0.1$  and  $k = 0.01$  gives  $m = 10$ ,  $N = 50$ , and  $\lambda = 1$  in Algorithm 12.3. Recall that the Forward-Difference method gave dramatically poor results for this choice of  $h$  and  $k$ , but the Backward-Difference method gave results that were accurate to about  $2 \times 10^{-3}$  for entries in the middle of the table. The results in Table 12.5 indicate the increase in accuracy of the Crank-Nicolson method over the Backward-Difference method, the best of the two previously discussed techniques. ■

Table 12.5

$x_i$	$w_{i,50}$	$u(x_i, 0.5)$	$ w_{i,50} - u(x_i, 0.5) $
0.0	0	0	
0.1	0.00230512	0.00222241	$8.271 \times 10^{-5}$
0.2	0.00438461	0.00422728	$1.573 \times 10^{-4}$
0.3	0.00603489	0.00581836	$2.165 \times 10^{-4}$
0.4	0.00709444	0.00683989	$2.546 \times 10^{-4}$
0.5	0.00745954	0.00719188	$2.677 \times 10^{-4}$
0.6	0.00709444	0.00683989	$2.546 \times 10^{-4}$
0.7	0.00603489	0.00581836	$2.165 \times 10^{-4}$
0.8	0.00438461	0.00422728	$1.573 \times 10^{-4}$
0.9	0.00230512	0.00222241	$8.271 \times 10^{-5}$
1.0	0	0	





O que queremos ?

- O menor custo computacional !

Porque não podemos ?

- Erro de discretização e Limite de Estabilidade

O que faremos?

Buscar métodos incondicionalmente estáveis !

# MAP 2320 – MÉTODOS NUMÉRICOS EM EQUAÇÕES DIFERENCIAIS II

**2º Semestre - 2019**

## Roteiro do curso

- Introdução
- Séries de Fourier
- **Método de Diferenças Finitas**
- **Equação do calor transiente (parabólica)**
- Equação de Poisson (elíptica)
- Equação da onda (hiperbólica)