CS522 - Partial Differential Equations

Tibor Jánosi

April 5, 2005

1 Numerical Differentiation

In principle, differentiation is a simple operation. Indeed, given a function specified as a closed-form formula, its differentiation involves the mechanical application of the chain rule, and of the rules that refer to the differentiation of elementary functions. Numerical differentiation is more challenging. We will now understand some of the difficulties.

Consider an example when the function to differentiate is inferred based on a set of sampled values (measurements), and these measurements are affected by noise. If one interpolates the function so that the interpolated function "goes" through all measured points, the implied function derivative will often have no relationship to the underlying "truth." Figure 1 below illustrates this point.

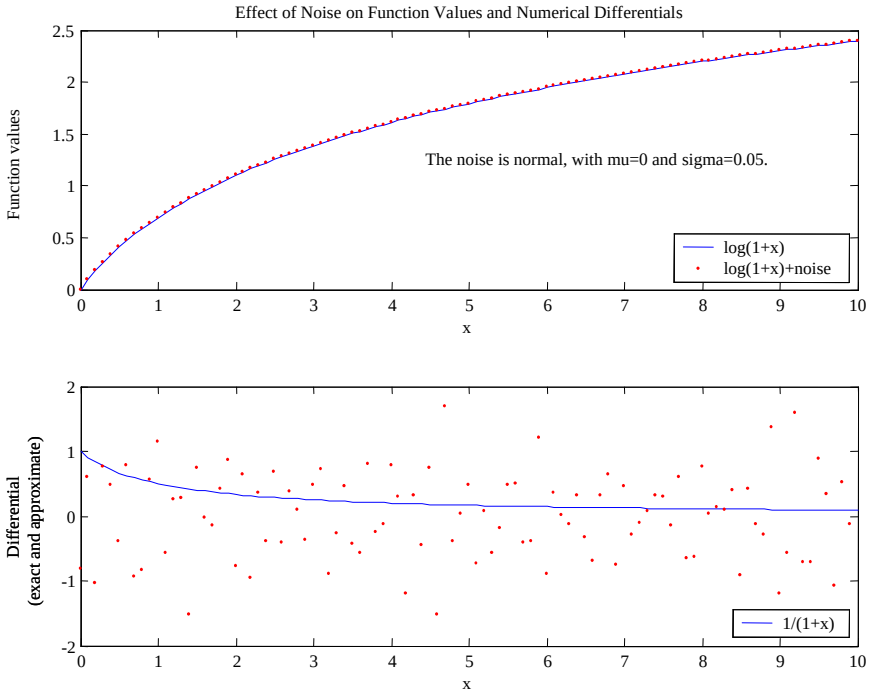

Figure 1: Effect of noise on the inferred value of numerical derivatives. Functions interpolated from noisy measurements should not be used directly to generate approximate values for their derivatives.

In our example, a simple, well-behaved function is perturbed by normal noise with

0 mean and standard deviation 0.05. On the graph, the values of the noisy function are practically indistinguishable from the unperturbed function. Indeed, the noisy values could be used directly in certain applications; for example, to integrate the function. If we use these perturbed values to compute numerical derivatives, however, the results bear no relation to the underlying reality.

So what can one do? One solution is to infer a reasonably smooth function from the noisy measurements, then use this function to compute approximate derivatives. If one knows the general form of the underlying function, then a least-squares method can be used to infer values for the unknown parameters. Once these parameters are determined, one can differentiate the function either numerically, or analytically.

It is, of course, possible for the underlying "true" function not to be known. In such cases one can try to choose functions whose general shape is in agreement with the measured data, or use spline techniques to approximate the unknown underlying function. Such methods are often reasonably successful, but they involve some amount of experimentation before obtaining a satisfactory outcome.

1.1 Techniques for Differentiation

The simplest - and perhaps most obvious - technique is that of functions provided as closed-form formulas. Such functions can be differentiated "by hand," or by using symbolic tools like Mathematica. The result can then be implemented directly.

1.1.1 Automated Differentiation

A generalization of this idea resulted in the idea of automated differentiation. Even if a function is not available as a closed-form formula, the program that computes it consists of a (perhaps very long) sequence of elementary operations, which can be seen as function applications. The derivatives of such elementary functions are known, so all one has to do is to suitably combine these derivatives (by using the chain rule) in order to obtain the derivative of the initial function. It is thus possible to rewrite the source code of a function in order to generate the derivative of the respective function with respect to one, or even all of its input arguments.

In practice, there are many problems that automated differentiation must overcome. Also, there are some inherent limitations. What should happen, for example, if certain libraries are only available in compiled form - should one try to reverse engineer them and develop automated differentiation for binary code as well? But what if legal impediments prevent libraries from being reverse engineered?

Despite some of these problems, automated differentiation enjoys increasing popularity, due to the high precision of these methods, and due to the low computational cost that they entail.

1.1.2 Finite Differences

Consider a differentiable function[1] $f : \mathbb{R} \to \mathbb{R}$, and the usual definition of the derivative $f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}$. Traditionally, differentiation has been approached as a purely numerical problem, and the limit above has been approximated by using finite differences.

Consider the Taylor-series expansions of function f around x:

$$\begin{aligned} f(x+h) &= f(x) + \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 + \frac{1}{3!}f'''(x)h^3 + ... \\ f(x-h) &= f(x) - \frac{1}{1!}f'(x)h + \frac{1}{2!}f''(x)h^2 - \frac{1}{3!}f'''(x)h^3 + ... \end{aligned}$$

Let us now examine at the following finite-difference approximations of the derivative:

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x)}{h} - \frac{1}{2!}f''(x)h - \frac{1}{3!}f'''(x)h^2 - ... \\ f'(x) &= \frac{f(x)-f(x-h)}{h} + \frac{1}{2!}f''(x)h - \frac{1}{3!}f'''(x)h^2 + ... \\ f'(x) &= \frac{f(x+h)-f(x-h)}{2h} - \frac{2}{3!}f'''(x)h + ... \end{aligned}$$

Assuming that h is small, and ignoring all terms containing h we get the following approximations:

$$\begin{aligned} f'(x) &\approx \frac{f(x+h)-f(x)}{h} \\ f'(x) &\approx \frac{f(x)-f(x-h)}{h} \\ f'(x) &\approx \frac{f(x+h)-f(x-h)}{2h} \end{aligned}$$

The three methods to compute numerical derivatives are called "forward approximation," "backward approximation," and "symmetric approximation," respectively, and all of them are exact in the limit. For finite values of h, the residual term of the forward and backward approximation is $O(h)$, while the residual term of the symmetric approximation is $O(h^2)$.

1.1.3 Richardson's Extrapolation

The finite precision of numerical computations can severely limit our practical ability to use any of the three finite-differences approximation given above. In other words, we might not reasonably approach the limit $h \to 0$. Richardson's method can help us overcome this difficulty by extrapolating the value obtained for the derivative for finite h's to the point $h = 0$.

[1] In the following we will implicitly assume that functions have all the properties that are needed in the context that we are using them.

Let us denote by $\mathbf{f}^{'}(h)$ the approximate value of the derivative $f^{'}(x)$, computed for a step size of h. Further, let us assume that a relationship of the form given below holds:

$$\mathbf{f}^{'}(h) = a_0 + a_1 h^p + O(h^r),\ r > p > 0$$

The value that we want to compute is $\mathbf{f}^{'}(0) = a_0$. Let us now compute function $\mathbf{f}^{'}$ for two values of h, h_1, and h_2. We get the following:

$$\begin{aligned} \mathbf{f}^{'}(h_1) &= a_0 + a_1 h_1^p + O(h_1^r) \\ \mathbf{f}^{'}(h_2) &= a_0 + a_1 h_2^p + O(h_2^r) \end{aligned}$$

Ignoring the residuals, we can solve for a_0:

$$\mathbf{f}^{'}(0) = a_0 = \mathbf{f}^{'}(h_1) + \frac{\mathbf{f}^{'}(h_1) - \mathbf{f}^{'}(h_2)}{\left(\frac{h_2}{h_1}\right)^p - 1}$$

Note that $\mathbf{f}^{'}(0)$ is still an approximation. This idea can be extrapolated so that more terms are considered in the expansion of $\mathbf{f}^{'}(h)$ given above; if n terms are considered, then n approximate values of the derivative must be computed for n different values of h. The resulting system of equations can then be solved to recover the value of a_0. For sufficiently well-behaved functions f, this method can produce very accurate results.

1.1.4 Higher-Order Derivatives

Analogously to the mathematical definition of higher-order derivatives, we can define finite-difference approximations to such derivatives using derivatives of lower order.

Consider the mathematical definition of the second derivative:

$$f^{''}(x) = \lim_{h \to 0} \frac{f'(x+h) - f^{'}(x)}{h}$$

Since we have several ways to approximate first-order derivatives, we will also have several approximations of the second derivative. We will, however, only consider the symmetric central-difference approximation given below:

$$\begin{aligned} f^{''}(x) &\approx \frac{f^{'}(x) - f^{'}(x-h)}{h} \\ &\approx \frac{\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x-h)}{h}}{h} \\ &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \end{aligned}$$

As it can be seen, the formula above is the backward difference of the forward difference approximations of the first derivative. This approximation is accurate to the order of $O(h^2)$.

1.2 Partial Differential Equations

For our immediate purposes, the most important differential equation will be the heat equation:

$$\frac{\partial f}{\partial t}(t,x) = \frac{\partial^2 f}{\partial x^2}(t,x)$$

Because the highest-order partial derivative that appears in equation above is 2, this is a second-order partial derivative equation.

Let us consider the general form of a second-order partial differential equation of a function f with arguments t and x:

$$c_1\frac{\partial^2 f}{\partial x^2} + c_2\frac{\partial^2 f}{\partial x \partial t} + c_3\frac{\partial^2 f}{\partial t^2} + c_4\frac{\partial f}{\partial x} + c_5\frac{\partial f}{\partial t} + c_5 f + c_6 = 0$$

Such partial differential equations are classified with respect to the sign of the expression $E = c_2^2 - 4c_1c_3$ as hyperbolic (if $E > 0$), parabolic (if $E = 0$), and elliptic (if $E < 0$).

Informally speaking, hyperbolic pde's describe time-dependent processes that are not evolving toward a steady state, parabolic pde's describe processes that are evolving toward a steady state, while elliptic processes have already reached a steady state (their solution is not time-dependent). When the coefficients c_i are not constant, the equation can change type, so this classification is of limited value.

The heat equation is a parabolic equation.

1.2.1 Initial and Boundary Conditions

As in the case of regular differential equations, solving a pde yields a family of solutions. To select a unique function from such a family one needs to impose further conditions; typically these consist of the initial condition and boundary conditions. For the specific case of the heat equation as applied to the problem of option values, our set of conditions will consist of the following:

$$\begin{aligned} f(0,x) &= \mathbf{f}_0(x) \\ \lim_{x\to\infty} f(t,x) &= \mathbf{f}_\infty \\ \lim_{x\to-\infty} f(t,x) &= \mathbf{f}_{-\infty} \end{aligned}$$

In the formula above $\mathbf{f}_0$ is a function independent of t, while $\mathbf{f}_\infty$ and $\mathbf{f}_{-\infty}$ are constants. As we will see below, practical considerations prevent us from directly using limits at infinity. Rather, we will choose a pair of values $x_{\min}$ and $x_{\max}$ so that they represent suitable approximations of ∞ and $-\infty$, respectively. We then obtain the following set of conditions:

$$\begin{aligned} f(0,x) &= \mathbf{f}_0(x) \\ f(t,x_{\max}) &= \mathbf{f}_\infty \\ f(t,x_{\min}) &= \mathbf{f}_{-\infty} \end{aligned}$$

1.2.2 Semidiscrete Solutions

The heat equation can be solved by discretizing the space variable x while leaving the time variable t continuous. To achieve this, we first divide the interval $[x_{\min}, x_{\max}]$ into an integer number of subintervals of length δx; let $N = \frac{x_{\max} - x_{\min}}{\delta x}$. In the second stage we associate a function $f_i(t)$, with each point $x_i = x_{\min} + i\delta x$, where $0 < i < N$. Using the symmetric central-difference approximation for the second partial derivative with respect to x, we get the following:

$$f_i^{'}(t) = \frac{f_{i+1}(t) - 2f_i(t) + f_{i-1}(t)}{(\delta x)^2},\ 1 < i < N-1$$

The initial and boundary conditions yield the following relations:

$$\begin{aligned} f_i(0) &= \mathbf{f}_0(x_i),\ 1 < i < N-1 \\ f_N(t) &= \mathbf{f}_\infty \\ f_0(t) &= \mathbf{f}_{-\infty} \end{aligned}$$

In effect, this method corresponds to computing the curves that result when we slice the surface $f(t, x)$ with planes of the form $x = x_i$; hence, this approach is called the method of lines.

By introducing the notation $F(t) = [f_1(t)\ f_2(t)\ f_3(t)\ ...\ f_{n-1}(t)]^T$, where $\bullet^T$ represents the transpose operator, we obtain the following system of ordinary differential equations:

$$F^{'}(t) = \frac{1}{(\delta x)^2} \underbrace{\begin{bmatrix} -2 & 1 & 0 & \dots & 0 \\ 1 & -2 & 1 & \dots & 0 \\ 0 & 1 & -2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -2 \end{bmatrix}}_{C} F(t) + \frac{1}{(\delta x)^2} \begin{bmatrix} \mathbf{f}_{-\infty} \\ 0 \\ 0 \\ \dots \\ \mathbf{f}_\infty \end{bmatrix}$$

The matrix of coefficients C is of size $(N-1)^2$, but only $3(N-1) - 2 = 3N - 5$ coefficients are non-zero. The higher N grows (i.e. the smaller δx becomes), the smaller the proportion of non-zero coefficients becomes. In the limit, the proportion of non-zero coefficients tends to 0, but even for $N = 100$ we only get a ratio of approximately 3% for such coefficients.

For the benefit of readers with more background in numerical analysis, we note here that the system of ordinary differential equations above tends to be very stiff, thus one must be careful to choose an appropriate method to solve it.

Sparse Matrices Matrices whose proportion of non-zero coefficients is very small with respect to their total number of coefficients are called sparse matrices. Sparse matrices are commonly encountered in several important areas of numerical computations. Special techniques have been developed to represent and manipulate such matrices so as to

minimize the amount of memory that would be wasted if the full matrix were explicitly represented. In this area, a lot of effort is expended to define matrix algorithms that minimize the number of new non-zero elements (the "fill") that are produced as a result of various matrix operations. The specific sparse matrix representation and algorithms that are chosen depend in general on the degree of sparsity, as well as on the sparsity pattern (i.e. the distribution of non-zero coefficients) exhibited by the matrices at hand.

Matrices encoded using sparse matrix representations incur an overhead with respect to regular matrices when various operations are performed. Depending on this overhead, any advantage related to memory savings can be swamped by the increased processing time that the sparse matrix representation entails. As a very approximate rule of thumb, one should probably not contemplate using sparse matrix representations unless the total number of coefficients coefficients is at least of a few thousand (or even higher).

For the example at hand, we have a particularly simple sparsity pattern which we could address by linearizing the matrix. To achieve this, we can transfer all coefficients into an array C^{linear} of size $3N-5$, going line by line in the original matrix C. We obtain the following representation of C^{linear}:

$$C^{linear} = \begin{bmatrix} -2 & 1 & 1 & -2 & 1 & \dots & -2 \end{bmatrix}$$

Given the pair of indices[2] (i, j) in matrix C, the corresponding index k in array C^{linear} will be given by $[3(i-1) - \min(i-1, 1)] + [j - \max(i-2, 0)]$.[3] Of course, the previous formula assumes that (i, j) denotes a non-zero coefficient. Can you establish what relation must hold between i and j so that this assumption is true?

Matlab provides support for sparse-matrix computations; type **lookfor sparse** at your Matlab prompt to learn more about this topic.

1.2.3 Discrete Solutions

In the preceding section we have discretized the space dimension x while leaving the time dimension t continuous. We can, of course, also discretize the time dimension.

As before, we first divide the interval $[x_{\min}, x_{\max}]$ into an integer number of subintervals of length δx; let $N_x = \frac{x_{\max} - x_{\min}}{\delta x}$. Next, we choose a sufficiently large maximum value of t, $t_{\max}$, and we divide the interval $[0, t_{\max}]$ into an integer number of subintervals of length δt; let $N_t = \frac{t_{\max} - 0}{\delta t}$. These two steps fully discretize the domain of the pde at hand. The corresponding situation is illustrated in figure 2.

We introduce the notation $f_n^m = f(m\delta t, x_{\min} + n\delta x)$, $0 \le m \le N_t$, $0 \le n \le N_x$, to denote the values of the function at the points of the resulting mesh.

Using the various finite-difference approximations for the first and second-order derivative, we can now write down the equations needed for solving the heat equation.

[2] Here, as elsewhere in the course, we stick with Matlab's convention and use 1-based indices.

[3] Try to understand how this formula was derived. Our use of parentheses should help you in this process.

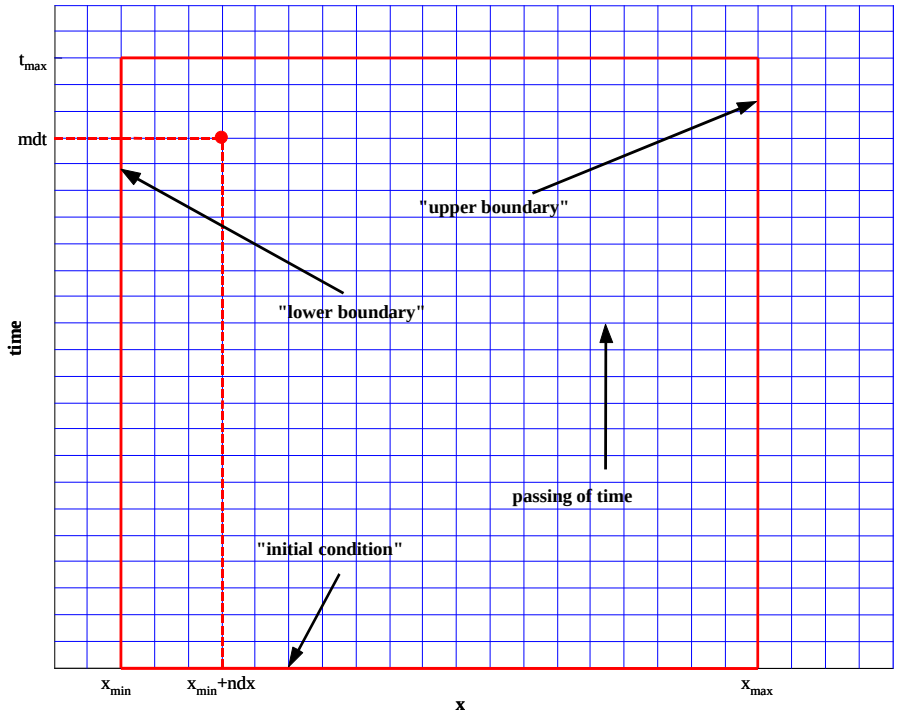

Figure 2: Illustration of the heat equation's discretized domain.

The Explicit Finite-Difference Method We choose the forward difference equation for the time derivative, and the symmetric central difference for the space derivative. Ignoring residual terms we get the approximate equality below:

$$\frac{f_n^{m+1} - f_n^m}{\delta t} = \frac{f_{n+1}^m - 2f_n^m + f_{n-1}^m}{(\delta x)^2},\ 0 < m < N_t,\ 0 < n < N_x$$

Introducing the notation $\alpha = \frac{\delta t}{(\delta x)^2}$, we get:

$$f_n^{m+1} = \alpha f_{n+1}^m + (1 - 2\alpha) f_n^m + \alpha f_{n-1}^m,\ 0 < m < N_t,\ 0 < n < N_x$$

The relationship between these quantities is illustrated in figure 3.

Since the values $f_n^0 = f(x_{\min} + n\delta x, 0) = \mathbf{f}_0(x_{\min} + n\delta x)$, $0 \le n \le N_x$, and $f_0^m = \mathbf{f}_{-\infty}$ and $f_{N_x}^m = \mathbf{f}_\infty$, $0 \le m \le N_t$ are known (given), we can compute the values of the function at all points in the domain of interest.

In principle, we know how to compute an approximate solution for the heat equation. But how good will this solution be? Can we understand how the quality of the result depends on the choice of δx and δt?

The explicit finite difference method has an error term of the form $O(\delta t) + O((\delta x)^2)$, i.e. it is first-order accurate in time and second-order accurate in space.

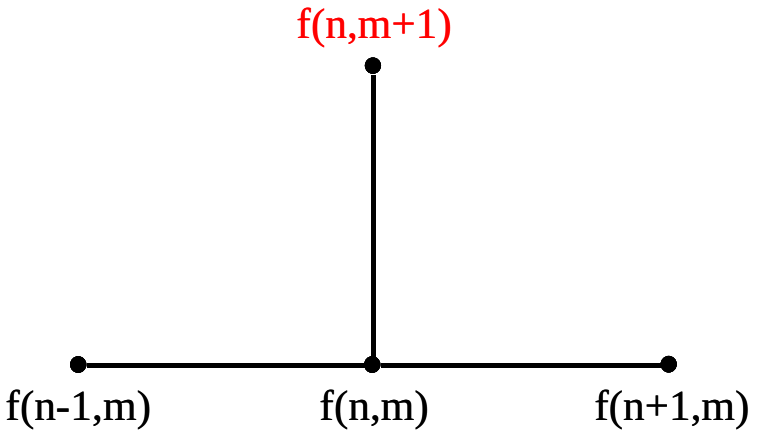

Figure 3: Relationship between known values (in black), and the newly computed value (in red) in the explicit finite-difference discretization.

Ideally, the solution to a pde problem should be both stable and consistent. Loosely speaking, **stability** means that small perturbations do not cause the corresponding approximate solutions to diverge without bound. **Consistency** means that as the step sizes δx and δt decrease toward 0 the truncation error (the error due to the discrete approximation of the derivative) decreases toward 0. For the convergence of an approximate solution to the true solution of the underlying problem it is necessary for the respective solution to be both consistent and stable.

As we can see from the discussion of the error terms, the explicit finite-difference method is consistent. Without providing more details, we will state that the finite-difference method is stable if $\alpha \leq \frac{1}{2}$, and not stable if $\alpha > \frac{1}{2}$.

The Fully-Implicit Method We again use the forward difference for the time derivative and the symmetric central-difference approximation for the second-order space derivative. For the latter second derivative, however, we will not choose the time coordinate to be $m\delta t$, as we did before; we will fix it at $(m+1)\delta t$. We then get:

$$\frac{f_n^{m+1} - f_n^m}{\delta t} = \frac{f_{n+1}^{m+1} - 2f_n^{m+1} + f_{n-1}^{m+1}}{(\delta x)^2},\ 0 < m < N_t,\ 0 < n < N_x$$

Keeping the same interpretation for $\alpha = \frac{\delta t}{(\delta x)^2}$, we get:

$$-\alpha f_{n-1}^{m+1} + (1+2\alpha) f_n^{m+1} - \alpha f_{n+1}^{m+1} = f_n^m,\ 0 < m < N_t,\ 0 < n < N_x$$

The relationship between these quantities is illustrated in figure 4.

As the reader can immediately see, the fully implicit method does not allow for the immediate computation of the new function values. Rather, a system of equations must

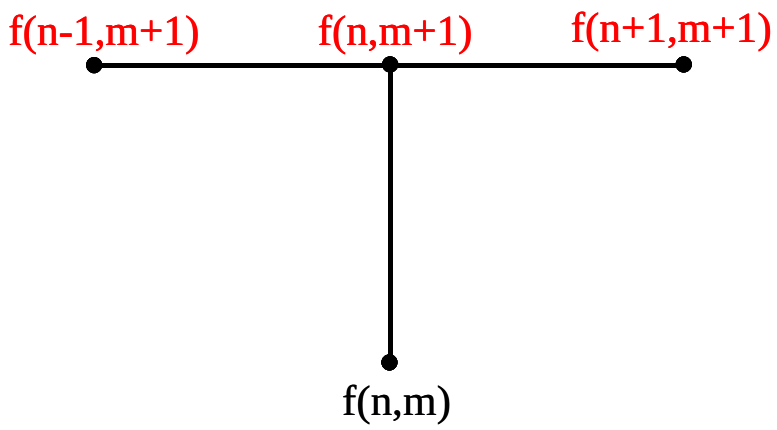

Figure 4: Relationship between known value (in black), and the newly computed values (in red) in the fully implicit finite-difference discretization.

be set up and solved. Let $F^m = \begin{bmatrix} f_1^m & f_2^m & f_3^m & \cdots & f_{N_x-1}^m \end{bmatrix}^T$, be column of unknown values at time $m\delta t$ (and note that f_0^m and $f_{N_x}^m$ are known). We obtain the following system of equations:

$$\underbrace{\begin{bmatrix} 1+2\alpha & -\alpha & 0 & \dots & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & \dots & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1+2\alpha & -\alpha \\ 0 & 0 & 0 & \dots & -\alpha & 1+2\alpha \end{bmatrix}}_{M} F^{m+1} = F^m + \alpha \underbrace{\begin{bmatrix} f_0^{m+1} \\ 0 \\ 0 \\ \dots \\ 0 \\ f_{N_x}^{m+1} \end{bmatrix}}_{b^m}$$

We can write the relation above in the much more compact form:

$$MF^{m+1} = b^m$$

If M were invertible, we could immediate compute $F^{m+1} = M^{-1}b^m$.

But is M invertible? Not necessarily, if α is arbitrary. However, if $\alpha > 0$, as it must be in this case, M is invertible. We prove this statement below.

Theorem 1 *(Gershgorin's Circle Theorem) Let A be a complex square matrix $A = (a_{ij})_{i,j=1,n}$ and let λ be equal to one of its eigenvalues. Define $R_i = \sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}|$, for each i, $1 \leq i \leq n$. Then λ will be in at least one of the disks $D_i : \{z \mid |z - a_{ii}| \leq R_i\}$.*

Proof. Let λ be an eigenvalue of A, and $v = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$ its corresponding eigenvector. We immediately get that $Av = \lambda v$. Further, choose i^* so that $|v_{i^*}| = \max\{|v_j| \mid 1 \leq j \leq n\}$. We immediately get that $|v_{i^*}| > 0$ (why?).

Now, consider the i^*-th component of the equality $Av = \lambda v$; we get:

$$\sum_{j=1}^{n} a_{i^*j} v_j = \lambda v_{i^*}$$

We can rewrite these equations as follows:

$$\sum_{\substack{j=1 \\ j \neq i^*}}^{n} a_{i^*j} v_j = (\lambda - a_{i^*i^*}) v_{i^*}$$

Finally, we have:

$$|\lambda - a_{i^*i^*}| = \left| \sum_{\substack{j=1 \\ j \neq i^*}}^{n} a_{i^*j} \frac{v_j}{v_{i^*}} \right| = \sum_{\substack{j=1 \\ j \neq i^*}}^{n} |a_{i^*j}| \underbrace{\left| \frac{v_j}{v_{i^*}} \right|}_{\leq 1} \leq \sum_{\substack{j=1 \\ j \neq i^*}}^{n} |a_{i^*j}|$$

■

Lemma 2 *Matrix* $\mathbf{M}$ *defined below has no negative real eigenvalues.*

$$\mathbf{M} = \begin{bmatrix} 2 & -1 & 0 & \dots & 0 & 0 \\ -1 & 2 & -1 & \dots & 0 & 0 \\ 0 & -1 & 2 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2 & -1 \\ 0 & 0 & 0 & \dots & -1 & 2 \end{bmatrix}$$

Proof. By applying Gershgorin's theorem we immediately get that all eigenvalues of matrix $\mathbf{M}$ must be in at least one of the disks given below:

$$\begin{aligned} D_1 &: \{z \mid |z-2| \leq 1\} \\ D_2 &: \{z \mid |z-2| \leq 2\} \end{aligned}$$

But D_1 is included in D_2, so we conclude that all eigenvalues of $\mathbf{M}$ must be in disk D_2. Hence $\mathbf{M}$ has no negative eigenvalues. ■

Theorem 3 *For any real, positive* α*, matrix* M *(as defined above) is invertible.*

Proof. Let us assume that M is not invertible. Then $\det(M) = 0$. We can immediately establish the obvious equality $M = I + \alpha\mathbf{M}$, where I is the identity matrix of size n, and $\mathbf{M}$ has been defined in the previous lemma. We now have:

$$\begin{aligned} \det(M) &= 0 \\ \det(I + \alpha\mathbf{M}) &= 0 \\ \det(\mathbf{M} + \frac{1}{\alpha} I) &= 0 \end{aligned}$$

The last relation implies that equation $\mathbf{M}y = -\frac{1}{\alpha}y$ admits a non-trivial solution (i.e. $y \neq \mathbf{0}$), which is the same thing as saying that one of $\mathbf{M}$'s eigenvalues must be $-\frac{1}{\alpha}$. Since we assumed that $\alpha > 0$, this means that $\mathbf{M}$ admits a negative eigenvalue. We have reached a contradiction, hence M is not invertible. ■

Returning now to the problem of solving the system of equations $MF^{m+1} = F^m$, we know that matrix M is in fact non-singular for any choice of $\delta x > 0$ and $\delta t > 0$. Hence $F^{m+1} = M^{-1}F^m$. However, this approach is not usually pursued. Without providing more details, we state that in general matrix inversions are avoided in numerical computations; they are expensive and can be sensitive to small errors in the initial matrix (if the matrix is ill-conditioned). In this case there is a further disadvantage: while the original matrix M is tridiagonal, hence sparse, its inverse M^{-1} would be dense. If we could avoid computing M^{-1} we could perhaps avoid the need to store $(N_x-1)^2$ coefficients, thus greatly reducing the memory consumption of our algorithm.

One approach to solving the system of equations is to use the so-called **LU** ("lower-upper") decomposition of matrix M. In a general LU decomposition of a matrix A, the respective matrix will be written as the product $L_A U_A$, where A is a lower-triangular matrix with unitary main diagonal (i.e. the main diagonal contains only 1's), and U_A is an upper-triangular matrix. Given the special form of our matrix M, its LU decomposition is especially simple to obtain by solving the following matrix equation:

$$\begin{bmatrix} 1+2\alpha & -\alpha & 0 & \dots & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & \dots & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1+2\alpha & -\alpha \\ 0 & 0 & 0 & \dots & -\alpha & 1+2\alpha \end{bmatrix} =$$

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ l_1 & 1 & 0 & \dots & 0 & 0 \\ 0 & l_2 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 & 0 \\ 0 & 0 & 0 & \dots & l_{N_x-2} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} d_1 & u_1 & 0 & \dots & 0 & 0 \\ 0 & d_2 & u_2 & \dots & 0 & 0 \\ 0 & 0 & d_3 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & d_{N_x-2} & u_{N_x-2} \\ 0 & 0 & 0 & \dots & 0 & d_{N_x-1} \end{bmatrix}}_{U}$$

We can easily solve this system of equations "by hand." First, by equating the terms that give the coefficient with indices $(1,1)$ of the result, we immediately obtain $d_1 = 1+2\alpha$. The equations for the coefficients with indices $(i, i+1)$ immediately yield the values $u_i = -\alpha$, $0 < i < N_x - 1$. Similarly, from the coefficients with indices $(i, i-1)$ we get that $l_{i-1}d_{i-1} = -\alpha$, $1 < i < N_x$. This implies that $l_i = -\frac{\alpha}{d_i}$, $0 < i < N_x - 1$. Finally, we determine the values d_i by examining the equations that correspond to the coefficients (i,i) in the resulting matrix. We get that $l_{i-1}u_{i-1} + d_i = 1 + 2\alpha$, $1 < i < N_x$. We immediately get that $d_i = 1 + 2\alpha - \frac{\alpha^2}{d_{i-1}}$, $1 < i < N_x$. These relations allow us to determine matrices L and U in a simple loop.

Note that we if we do not explicitly represent the 1's on the main diagonal of L_M, then we can represent both L_M and U_M in the same space that we used originally to store M. This is important if we used a linearization method like the one described above.

We can now solve the system of equations in two steps:

$$\begin{aligned} L_M v^m &= b^m \\ U_M F^{m+1} &= v^m \end{aligned}$$

We can solve the first system of equations by "forward substitution" in one pass; the second system can then be solved by "backward substitution." Can you estimate the total number of operations (i.e. the number of additions, subtractions, multiplications and divisions) needed to solve these systems?

At each step m we compute all the values f_n^m by solving the system of equations $MF^{m+1} = b^m$. The advantage of this method, however, is that its stability is much better than that of the explicit finite-difference method; i.e. the time step δt can be significantly bigger relative to δx in the case of the fully implicit method.

The Crank-Nicholson Method Recall that the error term for both the explicit finite-difference and fully implicit method was of the form $O(\delta t) + O((\delta x)^2)$. The Crank-Nicholson method improves this error term to $O((\delta t)^2) + O((\delta x)^2)$.

Let us recall the formulas we wrote for the explicit finite-difference method and the fully implicit method:

$$\begin{aligned} \frac{f_n^{m+1} - f_n^m}{\delta t} &= \frac{f_{n+1}^m - 2f_n^m + f_{n-1}^m}{(\delta x)^2},\ 0 < m < N_t,\ 0 < n < N_x \\ \frac{f_n^{m+1} - f_n^m}{\delta t} &= \frac{f_{n+1}^{m+1} - 2f_n^{m+1} + f_{n-1}^{m+1}}{(\delta x)^2},\ 0 < m < N_t,\ 0 < n < N_x \end{aligned}$$

We immediately obtain the following relation:

$$\frac{f_n^{m+1} - f_n^m}{\delta t} = \frac{1}{2}\left[\frac{f_{n+1}^m - 2f_n^m + f_{n-1}^m}{(\delta x)^2} + \frac{f_{n+1}^{m+1} - 2f_n^{m+1} + f_{n-1}^{m+1}}{(\delta x)^2}\right]$$

Again, using the notation $\alpha = \frac{\delta t}{(\delta x)^2}$, we get:

$$f_n^{m+1} - f_n^m = \frac{1}{2}\alpha(f_{n+1}^m - 2f_n^m + f_{n-1}^m + f_{n+1}^{m+1} - 2f_n^{m+1} + f_{n-1}^{m+1})$$

Separating the terms that refer to time $m\delta t$ and time $(m+1)\delta t$ we get:

$$-\frac{1}{2}\alpha f_{n-1}^{m+1} + (1+\alpha)f_n^{m+1} - \frac{1}{2}\alpha f_{n+1}^{m+1} = \underbrace{\frac{1}{2}\alpha f_{n-1}^m + (1-\alpha)f_n^m + \frac{1}{2}\alpha f_{n+1}^m}_{Z_n^m}$$

The relationship between these quantities is illustrated in figure 5.

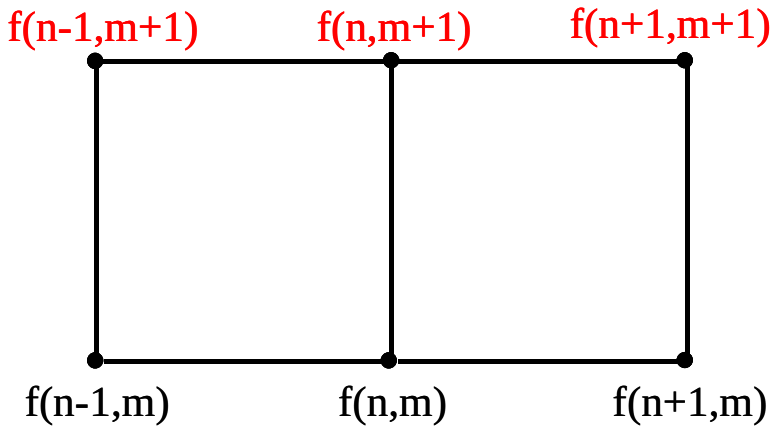

Figure 5: Relationship between known values (in black), and the newly computed values (in red) in the Crank-Nicholson method.

By analogy to the fully implicit method above we can write the following system of equations:

$$\underbrace{\begin{bmatrix} 1+\alpha & -\frac{1}{2}\alpha & 0 & \dots & 0 & 0 \\ -\frac{1}{2}\alpha & 1+\alpha & -\frac{1}{2}\alpha & \dots & 0 & 0 \\ 0 & -\frac{1}{2}\alpha & 1+\alpha & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1+\alpha & -\frac{1}{2}\alpha \\ 0 & 0 & 0 & \dots & -\frac{1}{2}\alpha & 1+\alpha \end{bmatrix}}_{M_{CN}} F^{m+1} = \underbrace{\begin{bmatrix} Z_1^m \\ Z_2^m \\ Z_3^m \\ \dots \\ Z_{N_x-2}^m \\ Z_{N_x-1}^m \end{bmatrix} + \frac{1}{2}\alpha \begin{bmatrix} f_0^{m+1} \\ 0 \\ 0 \\ \dots \\ 0 \\ f_{N_x}^{m+1} \end{bmatrix}}_{b_{CN}^m}$$

We use the subscript CN to denote matrices related to the Crank-Nicholson method, so that they are distinguishable from the analogous matrices we defined for the fully implicit method. Also, we reused the definition $F^m = \begin{bmatrix} f_1^m & f_2^m & f_3^m & \cdots & f_{N_x-1}^m \end{bmatrix}^T$.

The system of equations $M_{CN} F^{m+1} = b_{CN}^m$ can be solved by the method of LU decomposition as described above.