PY421/621 – Advanced Computing in Physics

Project #1:

-Blurring an image, the discretized Laplacian, the diffusion equation and the wave equation.

-Concepts of locality and parallelism, accuracy and stability of the algorithms, analysis in terms of eigenvectors and eigenvalues of the Laplacian operators.

Blurring an image, the discretized Laplace operator.

We assume that we know how to capture an image and convert the graphics information into a two dimensional, $N_x \times N_y$, array of intensity values for the individual pixels of the image. For simplicity we consider the overall intensity only and not the separate color values. Thus in practice we will be dealing with monochrome images.

With a suitable rescaling, the intensity values can be taken to be floating point numbers ranging from 0 to 1. We will denote the array of intensity values by $f(i,j)$, where the indices range over

$$i = 0 \ldots N_x - 1, \quad j = 0 \ldots N_y - 1 \tag{1}$$

The "blurring" operation consists in replacing the current value of the image field f with a new value f' according to the algorithm

$$f(i,j) \to f'(i,j) = (1-r)f(i,j) + r\frac{f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1)}{4} \tag{2}$$

where r is a parameter chosen in the range from 0 to 1. Equation 2 states that some fraction r of the current value of the pixel intensity is to be replaced with the average intensity of the nearest neighbor pixels. It is intuitevely clear that the procedure will dilute the information content in the image and that, if we repeat the substitution of Eq. 2 for several blurring steps, the image will diffuse and lose its detail. This is indeed what we observe if we write and execute a program that implements the blurring algorithm and diplays the resulting images every so many blurring steps.

Note: Eq. 2, as written above, is ill defined for the values of i and j which cause the shifted indices ($i+1$, $i-1$ etc.) to go beyond the bounds of the array. In order to make the equation well defined throughout the range of the array one must introduce a prescription for dealing with the indices at the boundary of the array. This is in particular crucial for the actual implementation of the algorithm in a computer code. Several different

types of boundary conditions can be imposed. We will use the very convenient periodic boundary conditions by which $f(N_x, j)$ is identified with $f(0, j)$ and, similarly, $f(-1, j) = f(N_x - 1, j)$, $f(i, N_y) = f(i, 0)$, $f(i, -1) = f(i, N_y - 1)$.

Running the code, i.e. experimenting with the algorithm, shows several interesting things.

First of all, the occurrence of some kind of diffusion process is manifest.

Then, if we calculate the average intensity of the image

$$f_{av} = \frac{1}{N_x\, N_y} \sum_{i,j} f(i, j) \;, \tag{3}$$

the average intensity squared

$$f^2_{av} = \frac{1}{N_x\, N_y} \sum_{i,j} f(i, j)^2 \tag{4}$$

and the standard deviation

$$df = f^2_{av} - (f_{av})^2 \;, \tag{5}$$

and print out the results, we observe that f_{av} stays, of course, constant (this is a simple check on our code) and that df decreases along the blurring process, with a rate of decay which eventually becomes exponential.

Finally it is interesting to check what happens if we take $r > 1$. If we use a debugger to inspect the elements of the array f, or if we print out the values of some selected elements at each blurring step, we see that for $r > 1$ the algorithm becomes unstable: the elements of f grow very rapidly and eventually go into overflow. It is also interesting to observe that, as the elements grow in magnitude, they also change sign at each step. Later on we will understand why this happens.

We wish now to make connection with a continuum field $f(x, y)$. For the purpose we will assume that the variation of the pixel intensity from site to site is small, so that it makes sense to consider the elements of f as the values taken by a continuum field on a discretized set of space points, i.e. on the vertices of some regular lattice. In reality, this assumption of "continuity" may not be satisfied by our images, which can exhibit sharp edges. However, it will be satisfied better and better if the image is subjected to increasing numbers of blurring steps. Moreover, it will be satisfied if f does represent the discretization of an actual continuum field.

It is useful to introduce a lattice spacing a and to associate to each pixel real x and y coordinates defined by

$$x = i\, a \quad y = j\, a \tag{6}$$

The actual value of a is arbitrary. a is a dimensionful quantity which can be given whatever value is appropriate to our calculation. Of course, it is important that the rate of variation of f be commensurate to the value of a, so that the hypothesis of continuity can hold.

Notice also that it would possible to take different spacings a_x and a_y along the two axes, but for simplicity we will take $a_x = a_y = a$.

We will also denote by

$$L_x = N_x\, a \quad L_y = N_y\, a \tag{7}$$

the total extent of the image.

We can now use a Taylor series expansion to relate the values taken by f at the neighboring lattice points $x + a, y$ etc. to $f(x, y)$. This gives

$$f(x+a,y) = f(x,y) + \partial_x f\Big|_{x,y} a + \partial_x^2 f\Big|_{x,y} \frac{a^2}{2} + \partial_x^3 f\Big|_{x,y} \frac{a^3}{6} + \partial_x^4 f\Big|_{x,y} \frac{a^4}{24} + \ldots \tag{8a}$$

$$f(x-a,y) = f(x,y) - \partial_x f\Big|_{x,y} a + \partial_x^2 f\Big|_{x,y} \frac{a^2}{2} - \partial_x^3 f\Big|_{x,y} \frac{a^3}{6} + \partial_x^4 f\Big|_{x,y} \frac{a^4}{24} + \ldots \tag{8a}$$

and similarly for $f(x, y + a)$, $f(x, y - a)$.

From these two equations we find that the combination $f(x + a, y) + f(x - a, y)$ is given by

$$f(x+a,y) + f(x-a,y) = 2f(x,y) + \partial_x^2 f\Big|_{x,y} a^2 + O(a^4) \tag{9}$$

Equivalently, we have

$$\frac{f(x+a,y) + f(x-a,y) - 2f(x,y)}{a^2} = \partial_x^2 f\Big|_{x,y} + O(a^2) \tag{10}$$

Equations (9) and (10) tell us that the sum over neighbors in the x direction in the blurring algorithm is related to an approximation of the second derivative with respect to x of the field. Indeed, it is better to invert the order of reasoning, proceeding as follows. From the Taylor series expansion of the field we find that the second derivative of a continuum field can be approximated by the "central difference" formula

$$\partial_x^2 f(x,y) \approx \frac{f(x+a,y) + f(x-a,y) - 2f(x,y)}{a^2} \tag{11}$$

with an error proportional to a^2 ($\times \partial_x^4 f \Big|_{x,y}$, to set the scale).

This is not the only possible numerical approximation to the second derivative. Other formulae producing higher accuracy can easily be devised, but for the majority of computational applications Eq. (11) provides a simple and adequate approximation. It will be convenient to use a special notation for the r.h.s. of Eq. (11). We will denote it by $\partial_{x,L}^2 f$ and, similarly, we will denote by $\partial_{y,L}^2 f$ the corresponding central difference approximation to $\partial_y^2 f$.

Adding approximations for $\partial_x^2 f$ and $\partial_y^2 f$ we find

$$\Delta f(x,y) = \frac{f(x+a,y) + f(x-a,y) + f(x,y+a) + f(x,y-a) - 4f(x,y)}{a^2} + O(a^2) \quad (12)$$

where we represented by Δ the Laplace operator $\partial_x^2 + \partial_y^2$.

Eq. (12) constitutes again a very useful numerical approximation formula. We will denote the lattice discretized Laplacian in the r.h.s. of Eq. (12) by Δ_L.

With this in mind we can now formally relate the blurring algorithm to a diffusion process.

The diffusion equation and the wave equation.

Substituting in Eq. (2) we find

$$f(x,y) \to f'(x,y) = f(x,y) + (\partial_x^2 f \Big|_{x,y} + \partial_y^2 f \Big|_{x,y}) \frac{ra^2}{4} \quad (13)$$

where we have omitted terms of higher order in a.

The Laplacian operator

$$\Delta = \partial_x^2 + \partial_y^2 \quad (14),$$

is of fundamental importance in mathematical physics. We see that, in the limit of smoothly varying fields, our blurring algorithm can be reinterpreted through the application of the Laplacian operator.

$$f(x,y) \to f'(x,y) = f(x,y) + \Delta f \Big|_{x,y} \frac{ra^2}{4} \quad (15)$$

It is useful to define

$$dt = \frac{ra^2}{4} \quad (16)$$

and to think of dt as a very small time step in some kind of evolution process implemented by the blurring algorithm: f and f' can be thought of as the values taken by a single function $f(x, y, t)$, varying in time, at the subsequent time values t and $t + dt$.

Equations (15) and (16) then give

$$\frac{f(x, y, t + dt) - f(x, y, t)}{dt} = \Delta f \Big|_{x,y} \tag{17}$$

But, neglecting higher orders in dt, the l.h.s. of this equation reduces to the time derivative of f. We thus find that the blurring algorithm implements the numerical solution of the equation

$$\partial_t f = \Delta f \tag{18}$$

Equation (18) is the diffusion equation, or heat equation, a fundamental equation of mathematical physics which enters in the description of many phenomena involving diffusion.

Of course, the above conceptual steps can be followed in two ways. We learn from them that, when we start from a smooth image, the blurring algorithm can be interpreted as some type of diffusion process. However, and more importantly, we can also go the other way around and observe that through these considerations we have learned how to discretize the Laplacian operator and solve numerically the diffusion equation.

The discretization of the Laplacian operator, or before that, of the second order derivatives ∂_x^2, ∂_y^2, is of crucial importance for computational applications.

The value of a, as we have stated above, is rather arbitrary. It sets the scale of length for the phenomena we wish to describe. The only important thing is that the field $f(x, y)$ varies little over the length scale specified by a so that terms of higher order in the Taylor series can be neglected. However, if we wish to implement the solution of the heat equation by the "blurring" algorithm, then the value of the time step dt is related to a and must be taken equal to $ra^2/4$. If we keep the dimensionless quantity r fixed, but rescale a, and therefore of all lenghts, by a constant factor c, $a \to c\,a$, Eq. (16) implies then that the time variable is rescaled according to $t \to c^2\,t$. This rescaling corresponds to the invariance of the heat equation: if $f(x, y, t)$ is a solution, so is $f(x/c, y/c, t/c^2)$. Since we can vary r, the value of dt is not fixed by the value of a. However, as we have observed already, r cannot be taken larger than 1. Larger values of r give origin to a numerical instability. It is in this sense that a^2 set the scale for dt. We can take values of dt smaller than $a^2/4$ ($r < 1$), making the simulation of the diffusion equation slower in computer time and more accurate in physical time (because the error in the discretization of the time derivative, Eq. (17), is of order dt), but we cannot take dt larger than $a^2/4$.

We can use our results for the discretized Laplacian for solving numerically another very important equation, namely the equation of D' Alembert, or wave equation

$$\frac{1}{v^2}\partial_t^2 f = \Delta f \tag{19}$$

In order to solve this equation one must assign initial values to f and $\partial_t f$. It is convenient to define the time derivative of f as a separate variable

$$\partial_t f(x, y, t) = p_f(x, y, t) \tag{20}$$

(the notation p_f is borrowed from field theory). The wave equation can the be recast in the form of two independent equations of the first order in time

$$\partial_t f(x, y, t) = p_f(x, y, t) \tag{21a}$$

$$\partial_t p_f(x, y, t) = v^2 \Delta f(x, y, z) \tag{21b}$$

As above, the derivatives with respect to time can be replaced with finite differences taken over the evolution step.

$$\partial_t f(x, y, t) = \frac{f(x, y, t + dt) - f(x, y, t)}{dt} = \frac{f' - f}{dt} \tag{21a}$$

$$\partial_t p_f(x, y, t) = \frac{p_f(x, y, t + dt) - p_f(x, y, t)}{dt} = \frac{p'_f - p_f}{dt} \tag{22b}$$

Combining Eqs. (21) and (22) we arrive at the evolution algorithm

$$f(x, y) \to f'(x, y) = f(x, y) + p_f(x, y)\, dt \tag{23a}$$

$$p_f(x, y) \to p'_f(x, y) = p_f(x, y) + v^2 \Delta f(x, y)\, dt \tag{23b}$$

where, of course, we will replace the Laplacian operator Δ with its discretized form given in Eq. (12).

Equations (23) can be easily implemented in a computer code and, in particular, we can experiment feeding into the simulation the discretized image we used for the diffusion process. Since the propagation of the wave is in two dimensions, however, the image spreads out, like waves in a pond, and soon loses its features. It is more interesting to simulate the wave equation in one dimension, replacing Eq. (23b) with

$$p_f(x, y) \to p'_f(x, y) = p_f(x, y) + v^2 \partial_{x,L}^2 f(x, y)\, dt \tag{23c}$$

Here we propagate the waves in the x direction, considering y like a parameter (all of the image lines corresponding to $y = \text{const}$ propagate independently along the x with common wave velocity v).

If we use as initial condition for f an image field $f_0(x, y)$ and for p_f the derivative $-\partial_x f_0(x, y)$ we will see the image move to the right, almost unchanged, corresponding to the solution of the one-dimensional wave equation $f(x, y, t) = f_0(x - vt, y)$.

This numerical experiment also leads to some important observations.

i) The choice of the initial condition $p_f(x, y, t = 0) = -\partial_x f_0(x, y)$ is, of course, motivated by the desire of producing just one wave moving to the right. In general, the solution of the wave equation is given by a superposition of waves traveling in opposite directions. If we had started from the given image field $f(x, y, t = 0) = f_0(x, y)$, but set $p_f = 0$, these initial data would have corresponded to an equal superposition of two image waves traveling to the right and to the left, leading to a more messy evolution pattern. It is actually an interesting experiment to start with these initial data and see the image split.

ii) The initial value of $-\partial_x f_0(x, y)$ must be determined numerically. It is better to use the so called central difference approximation to the derivative

$$\partial_{x,c} f(x) = \frac{f(x + a) - f(x - a)}{2a} \tag{24}$$

rather than either the forward or backward approximations

$$\partial_{x,f} f(x) = \frac{f(x + a) - f(x)}{a} \tag{25a}$$

$$\partial_{x,b} f(x) = \frac{f(x) - f(x - a)}{a} \tag{25b}$$

Indeed, it is easy to see from a Taylor series expansion of $f(x + a)$ and $f(x - a)$ (done under the assumption that $f(x)$ represents a continuous function with a sufficient number of continuous derivatives) that the error in the approximation of $\partial_x f(x)$ by $\partial_{x,f} f(x)$ or $\partial_{x,b} f(x)$ is of order a, while the error in the approximation of $\partial_x f(x)$ by $\partial_{x,c} f(x)$ is of order a^2 (the terms of order a cancel because of symmetry).

iii) The value of dt is again to some extent arbitrary. Smaller values of dt lead to improved accuracy in the simulation of the continuum time system, but use more CP time for the same physical evolution time. Larger values of dt produce the opposite. However, dt cannot be taken larger than a witout running into a numerical instability. We see, once again, that the spatial discretization also sets the scale for the discretization of the time evolution. With the wave equation, though, the limit is less stringent, $dt \sim a$ rather than $dt \sim a^2$.

iv) The image propagates reasonably well, but some aliasing of the edges becomes soon apparent (aliasing is the phenomenon observed for instance on a television screen when

the lines which separate regions of different color or intensity, which ought to be sharp, instead exibit fringes). We can also observe that performing some blurring steps on the initial image reduces the amount of aliasing. The reason for this behavior lies in the change of dispersion formula produced by the discretization of the spatial axis. We recall from the study of the wave equation that a pure wave with wave number k propagates with frequency $\omega = v|k|$. The equation relating ω and k is called the dispersion formula

$$\omega = F(k) \tag{26}$$

With the wave equation the dispersion formula takes the very simple form

$$\omega = v|k| \tag{27}$$

The fact that waves propagate along the x-axis without deformation, or, using a technically more accurate term, without dispersion, is a consequence of this very simple dispersion formula, which states that all wave length components propagate with the same phase velocity.

However we replace ∂_x^2 with $\partial_{x,L}^2$ in the wave equation (we keep time continous for now) the dispersion formula for the equation

$$\partial_t^2 f(x,t) = v^2 \partial_{x,L}^2 f(x,t) \tag{28}$$

is no longer given by Eq. (27). Rather, it takes the form

$$\omega = v\sqrt{\frac{2-2\cos(ka)}{a^2}} \tag{29}$$

as can be seen by substituting into the equation a pure wave *Ansatz* of the form

$$f(x,t) = e^{\imath kx - \imath\omega t} \tag{30}$$

Because of the different dispersion formula, in the spatially discretized system different wavelengths propagate with different velocity. For long wavelengths the discretized equation approximates rather well the continuum system, and the dispersion is hardly noticeable. For shorter wavelengths (i.e. for the sharp edges) the approximation becomes poorer and the distortion due to the dispersion becomes quite apparent. Blurring the image reduces the magnitude of its short wavelength components, and this is why the blurred image appears to propagate with less distortion.

Data parallel computations.

Implementing the diffusion algorithm (or the simulation of the wave equation) can lead to a substantial use of computer time. A very accurate calculation might require a discretization

of spatial coordinates by a 1000×1000 array. As we have observed, a refinement of the spatial discretization by some factor of α requires that the time step be reduced by a factor α^2 for numerical stability. If we assume that the simulation of the diffusion process ends up demanding $100,000$ time steps, we see that the actual number of floating point operations to be performed is of the order of 10^{12} (an order of magnitude of 10 operations per site, times 10^6 sites in our discretized system, times 10^5 steps). Even with today's ultrafast computers, this is not a totally negligible number of operations.

However, a closer examination of the algorithm reveals that this very large number of operations is the result of the identical repetition of a rather elementary set of arithmetic and data motion operations, replicated over all sites of the lattice and iterated for a large number of times. The replication across the lattice is of particular interest. The operations that need to be done at the various sites are of similar nature and, *especially*, independent on one another. This is expressed by saying that the calculation is data parallel. This data parallelism is syntactically captured by languages implementing array instructions, like Fortran 90, where, by using array notation and the C-shift data motion operations, one can code the whole computation in a very compact manner.

More importantly, the data parallel nature of the calculation implies that in a computer with multiple processing units, different processors could perform the basic operations at different sites, achieving a substantial speed-up of the computation. In principle, since the operations to be performed at the various sites at each iteration are completely independent of one another, the speed-up could be equal to the number of processors. I.e., if it takes a time T to perform one iteration of the diffusion algorithm on a lattice with $L \times L$ sites on a single processor machine, a multiprocessor with N_p units (of the same type) should be able to perform the iteration in a time T/N_p. (So long as the number of processors does not exceed the number of sites. If $N_p > L^2$ some of the processors would have to remain idle, but with current multiprocessors, which tend to have a contained number of rather powerful units, in practice this will never be the case.) In reality, the speed-up will be limited by architectural factors such as the way in which memory is organized (it could be shared, or distributed among processors, or a mix of the two) and the efficiency by which data can be moved to and from the processors (i.e. of data communication). Indeed, although the operations to be done at the individual sites are independent, they require accessing the values of the data at neighboring sites. No matter how the sites are subdivided among several processors, some of the neighbors of every processor sites must be assigned to some other processor. Since the corresponding data will be required at the next iteration, this implies interprocessor communications (possibly via a shared memory). The manner in which data can be distributed in a multiprocessing environment and the many possible ways in which interprocessor communications can be organized form a vast and important subject, into which we cannot enter here. The gains in performance which can obtained from a computer with multiple processors depend crucially on the interplay between these architectural factors and the type of parallelism of the calculation. The example we have been dealing with belongs to the category of "embarrassingly parallel" calculations, in the sense that distribution of the data is so regular and the algorithm so uniform throughout the data set that the computation represents an ideal case for the

application of multiprocessing, and also that not much effort is required on the programmer's side to reap the advantages of parallel computing. It is anyway of great interest to experiment with multiprocessing even in such a straightforward case and measure the speed-up which can be obtained.

Accuracy and stability considerations.

It is interesting and instructive to study in more detail a numerical algorithm such as the one we introduced above for solving the diffusion equation. The basic step consists in replacing the current value of the array $f_{i,j}$ with a new value $f'_{i,j}$ according to the equation

$$f_{i,j} \to f'_{i,j} = f_{i,j} + (\Delta_L f)_{i,j}\, dt \tag{31}$$

where $(\Delta_L f)_{i,j}$ represents the array obtained by acting on f with the discretized Laplacian operator.

It is crucial to observe that this is a linear operation acting on the elements of f, considered as a vector in a vector space of dimensionality $N = N_x \times N_y$. From this point of view, the fact that f carries the two indices i and j should not lead us into confusion, thinking for instance of f as a matrix. Rather, we should think of all of the components of f as the components of a one-dimensional array of size N. This is, after all, how the compiler will arrange all of the entries of f in the memory of the computer, namely as a single array of numbers which can be indexed by an integer $k = i + N_x\, j$ if we follow the ordering of Fortran (first index varies first), or $k = j + N_y\, i$ if we follow the ordering of C (last index varies first). In any case the index k ranges from 0 to $N - 1$ as i and j vary from 0 to $N_x - 1$ and $N_y - 1$ respectively.

The action of the discretized Laplacian can then be represented by a matrix $-A_{k,k'}$ (we represent Δ_L with $-A$ because the Laplacian is a negative semidefinite operator, see later)

$$(\Delta_L f)_k = -\sum_{k'} A_{k,k'} f_{k'} \tag{32}$$

where we have used again a single index k to denote all of the components of the two-dimensional array $(\Delta_L f)_{i,j}$.

In practice, one would never implement the action of the discretized Laplacian through explicit matrix multiplication, as in Eq. (32). This would be extremely wasteful of memory and computer time, if not altogether impossible. Indeed, the matrix A is what one calls a sparse matrix (very sparse indeed), i.e. a matrix where only a small subset of elements are different from 0 while all of the other vanish. To verify this statement in regards to A let us just consider that, for given k, i.e. for a given pair of indices i, j, the only terms in the r.h.s. of Eq. (32) which contribute to $(\Delta_L f)_k$ are those with a k' that corresponds to either $i' = i, j' = j$, or $i' = i \pm 1, j' = j$, or $i' = i, j' = j \pm 1$. These are 5 for k' index out of a total of $N = N_x \times N_y$ possible values. We thus see that A has only $5N$ non-vanishing matrix elements out of a total of N^2 elements.

A is an $N \times N$ symmetric matrix which will therefore have N distinct mutually orthogonal eigenvectors which can be normalized to unit length. Let us denote these eigenvectors by $f_k^{(m)}$, $m = 0, \ldots N-1$ and the corresponding eigenvalues (which do not have to be all different) by λ_m:

$$\sum_{k'} A_{k,k'} f_{k'}^{(m)} = \lambda_m f_k^{(m)} \tag{33}$$

As we will soon see, with the very simple geometry and boundary conditions which we have been considering, the eigenvectors and eigenvalues of A can be found explicitly. In a more general context this would typically not be possible, but the equivalent of the matrix A *will always have a complete set of eigenvectors* and associated eigenvalues. Let us therefore expand the discretized field f as

$$f_k = \sum_m c_m f_k^{(m)} \tag{34}$$

Substituting into Eq. (31) we find

$$\begin{aligned} f_k = \sum_m c_m f_k^{(m)} \rightarrow f'_k = \sum_m c'_m f_k^{(m)} &= \sum_m c_m f_k^{(m)} - \Big[\sum_m c_m \sum_{k'} A_{k,k'} f_{k'}^{(m)} \Big] \, dt \\ &= \sum_m c_m f_k^{(m)} - \Big[\sum_m \lambda_m c_m f_k^{(m)} \Big] \, dt \end{aligned} \tag{35}$$

i.e.

$$c_m \rightarrow c'_m = c_m - \lambda_m c_m \, dt = (1 - \lambda_m \, dt) c_m \tag{36}$$

This analysis shows that in the basis of the eigenvectors of A the algorithm becomes diagonal. The coefficients c_m which give the expansion of the current value of the field do not mix during the iterations, but change, separately, according to Eq. (36).

If we performed a similar analysis for the continuous time evolution (but still discretized in space), we should start from the equation

$$\frac{df_k(t)}{dt} = -\sum_{k'} A_{k,k'} f_{k'}(t) \tag{37}$$

Expanding the field $f_k(t)$ again into normal modes (see Eq. (34) above) we would get

$$\frac{df_k(t)}{dt} = \sum_m \frac{dc_m(t)}{dt} f_k^{(m)} = -\Big[\sum_m c_m(t) \sum_{k'} A_{k,k'} f_{k'}^{(m)} \Big] = -\sum_m c_m(t) \lambda_m f_k^{(m)} \tag{38}$$

from which it follows that

$$\frac{dc_m(t)}{dt} = -\lambda_m c_m(t) \tag{39}$$

This equation is solved by

$$c_m(t) = e^{-\lambda_m t} c_m(0) \tag{40}$$

Equation (40) tells us that, when t is incremented by dt, $c_m(t)$ changes according to

$$c_m(t) \to c'_m \equiv c_m(t + dt) = e^{-\lambda_m \, dt} c_m(t) \tag{41}$$

Comparing Eqs. (36) and (41) we see that the effect of the time discretization is to replace the correct evolution factors $\exp(-\lambda_m \, dt)$ with their first order approximations $(1-\lambda_m \, dt)$. How good is this approximation? This clearly depends from the value of dt and the value of λ_m. For a given dt, the smaller is λ_m, the better is the approximation. As we will see, typically the eigenvalues of the discretized Laplacian range between numbers of order 1 all the way to numbers of order $1/a^2$. Thus the accuracy by which the correct, continuous time evolution is approximated by the numerical algorithm is not uniform, but varies for the different normal mode components of the field. As it turns out, the normal mode components with smaller eigenvalues are those which correspond to the long range features of the system, those which typically have more physical significance and which we would like to simulate better. The normal modes with the largest eigenvalues correspond instead to lattice artifacts, i.e. modes which depend on the discretization and have little bearing on the physical properties of the system. (This under the assumption that we have taken a lattice spacing a sufficiently small that the length scales of physical significance are substantially larger than a). This is rather fortunate, in the sense that the requirement that the error in the approximation, which is of order $(\lambda_m \, dt)^2$, be small for the physically significant modes does not constrain too severley the magnitude of the time step dt.

A more stringent constraint on dt comes, however, form the requirement that the algorithm be stable. Indeed, as we see form Eq. (36), if the value of dt is such that

$$\lambda_{max} \, dt > 2, \tag{42}$$

where λ_{max} denotes the largest eigenvalue, then the the corresponding expansion coefficient c_{max}, together with all those other c_m for which $\lambda_m \, dt > 2$, will increase in magnitude from iteration to iteration (with alternating sign), rather than decrease as the continuous time evolution demands. The algorithm thus becomes unstable. We may not care so much about the fact that the temporal behavior of the modes with largest eigenvalues is poorly approximated, so long as their magnitude does not increase, since typically this will be very small in the expansion of the initial field configuration. But it is crucial that these modes do not increase in magnitude, especially considering that values $\lambda_m \, dt > 2$ give origin to

an exponential growth. In general, the largest eigenvalues of the discretized Laplacian are of order $1/a^2$. Thus stability demands

$$dt < const. \times a^2, \tag{43}$$

This is a very severe constraint. It implies that if we want to increase the accuracy of the spatial discretization by reducing the lattice spacing by some factor s, the computational costs not only increase by the obvious factor of s^D, where D is the dimensionality of the system (because we will have s^D as many data elements), but also of an additional factor of s^2, because we will have to reduce the time step by that factor.

Let us conclude by finding the explicit eigenvalues and eigenvectors of the discretized Laplacian for the system we have considered in this project, where, because of the very simple geometry, they can be calculated analytically. The system, we recall, consists of a rectangular lattice of N_x by N_y sites with periodic boundary conditions. We consider the operators $-\partial^2_{x,L}$ and $-\partial^2_{y,L}$, defined by

$$-(\partial^2_{x,L} f)_{i,j} = \frac{2f_{i,j} - f_{i+1,j} - f_{i-1,j}}{a^2} \tag{44a}$$

$$-(\partial^2_{y,L} f)_{i,j} = \frac{2f_{i,j} - f_{i,j+1} - f_{i,j-1}}{a^2}, \tag{44b}$$

where $0 \le i < N_x$, $0 \le j < N_y$, and the operator

$$A \equiv -\Delta_L = -\partial^2_{x,L} - \partial^2_{y,L} \tag{45}$$

The uniformity of the lattice and the invariance of the operators under translation suggest that we look for eigenvectors with exponential dependence on i and j. Indeed, if we consider the vector (normalized to 1)

$$u_i^{(k_x)} = \frac{1}{\sqrt{N_x}} \exp\Big(2\pi\imath \frac{ik_x}{N_x}\Big) \tag{46a}$$

it is easy to verify that the following equation is satisfied

$$\begin{aligned} 2u_i^{(k_x)} - u_{i+1}^{(k_x)} - u_{i-1}^{(k_x)} &= \Big[2 - \exp\Big(\frac{2\pi\imath k_x}{N_x}\Big) - \exp\Big(-\frac{2\pi\imath k_x}{N_x}\Big)\Big] u_i^{(k_x)} = \\ &= 2\Big[1 - \cos\Big(\frac{2\pi k_x}{N_x}\Big)\Big] u_i^{(k_x)} \end{aligned} \tag{47}$$

Thus we see that $u_i^{(k_x)}$ is an eigenvector of $-\partial_{x,L}$ with eigenvalue $2[1-\cos(2\pi k_x/N_x)]/a^2$. Periodicity demands that k_x be an integer. On the other hand, it is clear from Eq. (46a)

that k_x and $k_x + N_x$ give origin to identical vectors u_i. Thus the possible values of k_x are $0, 1, 2 \ldots N_x - 1$ and we have N_x distinct eigenvectors.

Similarly we find that

$$v_j^{(k_y)} = \frac{1}{\sqrt{N_j}} \exp\Big(2\pi\imath \frac{jk_y}{N_y}\Big), \tag{46b}$$

with $k_y = 0, 1 \ldots N_y - 1$, are eigenvectors of $-\partial_{y,L}$ with eigenvalues $2[1-\cos(2\pi k_y/N_y)]/a^2$.

Combining the two, we find for the eigenvectors of $A = -\Delta_L$

$$f_{i,j}^{(k_x,k_y)} = u_i^{(k_x)}\, v_j^{(k_y)} = \frac{1}{\sqrt{N_x N_y}} \exp\Big[2\pi\imath\Big(\frac{ik_x}{N_x} + \frac{jk_y}{N_y}\Big)\Big] \tag{48}$$

with

$$\lambda_{k_x,k_y} = \frac{4 - 2\cos(2\pi k_x/N_x) - 2\cos(2\pi k_y/N_y)}{a^2} \tag{49}$$

as the corresponding eigenvalues.

The pair of integers k_x, k_y label the N distinct eigenvectors and the corresponding eigenvalues. Since, as we noticed above, k_x and $k_x + N_x$ give origin to the same eigenvector, it is sometime more convenient to take the range of k_x as $-N_x/2 + 1 \le k_x \le N_x/2$, assuming that N_x is even, or $-N_x/2 + 1/2 \le k_x \le N_x/2 - 1/2$ if N_x is odd. Similarly one takes $-N_y/2 + 1 \le k_y \le N_y/2$ or $-N_y/2 + 1/2 \le k_y \le N_y/2 - 1/2$, for N_y even or odd, respectively. This gives origin to a more symmetric labeling of the eigenmodes.

The eigenvectors are exponential waves of wavelength $l_x = N_x a/k_x$ and $l_y = N_y a/k_y$ in the x and y directions. So long as $k_x \ll N_x$ and $k_y \ll N_y$, these wavelengths will span several lattice sites. This the the scale of length which characterizes the physical features of the system under consideration (almost by definition, since we should always take a lattice spacing much smaller that this length in order to provide an accurate description of the system). On the other hand, for $k_x \ll N_x$ and $k_y \ll N_y$, the cosines in Eq. (49) can be expanded into Taylor series and we get

$$\lambda_{k_x,k_y} \approx \Big(\frac{2\pi k_x}{N_x a}\Big)^2 + \Big(\frac{2\pi k_y}{N_y a}\Big)^2 = \Big(\frac{2\pi}{l_x}\Big)^2 + \Big(\frac{2\pi}{l_y}\Big)^2 \tag{50}$$

Thus we see that eigenvalues λ_{k_x,k_y} corresponding to the "physical" eigenmodes have a magnitude independent of a and determined by the physical length scales of the system.

On the other hand, if we take k_x and k_y close or equal to end point values of their respective ranges of definition, then the ratioes k_x/N_x, k_y/N_y get very close to $1/2$ (in magnitude) and the arguments of the cosines in Eq. (49) get very close to π. Correspondingly

$$\lambda_{k_x,k_y} \approx \frac{8}{a^2} \tag{51}$$

In conclusion, this analysis shows that, as anticipated above, the eigenvalues of the discretized Laplacian range between values of order "one" (by which we mean values which depend on the physical properties of the system, but are independent of a) and values of order $1/a^2$. This is typical, not just specific to the very simple geometry which we are now considering. Thus the limitations on the magnitude of the time step in the computational solution of the diffusion equation or of the wave equation are quite general.

Finally, let us observe that, if we do know the eigenvectors and eigenvalues of the discretized Laplacian, we can solve the diffusion equation directly, without having to iterate discretized time steps. Indeed, we can also "unblur" the image, reversing the effects of the blurring operation. What we must do is expand the original field $f_{i,j}(t=0)$ into the complete set of eigenvectors of $-\Delta_L$, as in Eq. (34). This can be done using the ortonormality of the eigenvectors, which gives

$$c_{k_x,k_y} = \sum_{i,j} f_{i,j}^{(k_x,k_y)*} f_{i,j} = \sum_{i,j} \frac{1}{\sqrt{N_x N_y}} \exp\Big[-2\pi\imath\Big(\frac{ik_x}{N_x} + \frac{jk_y}{N_y}\Big)\Big] f_{i,j} \tag{51}$$

This equation can be implemented very efficiently by the algorithm of the fast Fourier transform (FFT), which we will consider later in the course.

Once we have determined the initial expansion coefficients, we can implement any number n of computational diffusion steps all at once by

$$c_{k_x,k_y} \to c'_{k_x,k_y} = (1 - \lambda_{k_x,k_y}\, dt)^n c_{k_x,k_y} \tag{52}$$

and by reconstructiong then the field from Eq. (34), which in this case reduces to the direct Fourier transform

$$f'_{i,j} = \sum_{k_x,k_y} \frac{1}{\sqrt{N_x N_y}} \exp\Big[2\pi\imath\Big(\frac{ik_x}{N_x} + \frac{jk_y}{N_y}\Big)\Big] c'_{k_x,k_y} \tag{53}$$

Of course, if we can avail ourselves of the expansion into normal modes, there is no reason for discretizing the time evolution. We can implement the continuous time evolution directly by replacing Eq. (52) with

$$c_{k_x,k_y} \to c'_{k_x,k_y} = \exp[-\lambda_{k_x,k_y}\, t] c_{k_x,k_y} \tag{54}$$

The problem is that, apart from very special cases, the eigenvectors and eigenvalues are not known analytically. In principle it would be possible to determine them computationally, but the procedure would be computationally very costly, and carrying out the steps equivalent to Eqs.(52) to (54) above would be also quite demanding, if not simply unfeasible.

Equations (52-54) open another interesting possibility in the case under study. By reversing them we can "unblur" the image. The required steps can be conveniently implemented with the FFT and, provided that the image has not been blurred to the point that the coefficients of the expansion have fallen below machine accuracy, the result is quite spectacular.