Due Monday, December28, 24:00

This problem set is worth 24 points (6 for style). As usual, the total is more than 18 points, extras go to bonus.

- Among many sources, Numerical Recipes is the one i followed most closely in class. You might want to have a look at it for finite difference methods for PDEs, but some of its sections are well above our level.
- We will have some problems with actual new physics as well as some problems that are meant to be instructive, e.g. solving the wave equation in ten different ways even though we know the exact analytical solutions. Do not think of these as empty chores. There are many aspects of a numerical PDE scheme that one needs to see to understand what happens in practice in terms of stability, convergence, dissipation, dispersion... Try to pay attention to these so that you can have an informed opinion about what method to use for your own research problems.
- I do not prescribe specific grid spacing in many of the following. Use values you think to be appropriate, and do not be afraid to test a few to find the optimal. Remember that the most accurate is not always the optimal.
- I sometimes tell it explicitly, but in all cases, show that you have convergence. Remember Lax's theorem, so not seeing convergence would be a sign of something wrong in your implementation or numerical methods. Besides, you would want to make sure that the order of convergence is as expected, and you are not wasting computational resources due to false beliefs about the efficiency of your code.
- Note that complex numbers are an integral part of SciPy, so there is no need to separate the real and imaginary parts of functions as is done in older programming languages.

Problem 24 Poisson equation with other boundary conditions

Consider the Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

on $[-1\ 1] \times [-1\ 1]$ with the source

$$f(x) = \begin{cases} \left(1 - \frac{x^2+y^2}{a^2}\right) & x^2 + y^2 \leq a^2 \\ 0 & \text{otherwise} \end{cases},$$

and homogenous Neumann BCs. You can assume that these are the dimensionless quantities that are obtained after scaling the original ones.

(a) Use a uniform cell-centered grid, the two dimensional version of the grid described in problem 21, and write the discretized equation as a matrix problem. Remember, the matrix is symmetric in this case.

(b) Solve this problem for $a = 0.5, 0.05$. What are sensible choices of $\Delta x = \Delta y = h$ in each case? Refine your grids for both cases and provide your convergence analysis, what is the order of convergence?

Problem 25 Quasi two-dimensional motion of the atmosphere (6 points)

Solve the Problem in Chapter 8 of your textbook (Kutz). Namely, you have a system of coupled nonlinear PDEs for $\omega(x, y, t)$ and $\psi(x, y, t)$

$$\frac{\partial \omega}{\partial t} + \left(\frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} \right) = \nu \nabla^2 \omega \tag{1}$$

$$\nabla^2 \psi = \omega \tag{2}$$

with periodic boundary conditions on the domain $x \in [-L,\ L],\ y \in [-L,\ L]$

$$\omega(-L, y, t) = \omega(L, y, t) \tag{3}$$
$$\omega(x, -L, t) = \omega(x, L, t) \tag{4}$$
$$\psi(-L, y, t) = \psi(L, y, t) \tag{5}$$
$$\psi(x, -L, t) = \psi(x, L, t) \tag{6}$$

and the initial condition

$$\omega(x, y, 0) = e^{-2x^2 - y^2/20} \ . \tag{7}$$

(a) Show that if ψ is a solution, so is $\psi + C$ for any constant C. This means you have to take some care in solving the elliptic equation for ψ, which is discussed in Kutz.

(b) Solve the system of PDEs until $t = 10$ for $L = 10$ and $\nu = 10^{-3}$. The solution is discussed in Chapter 8 using a few different methods. It is enough for you to implement one method in Python. Make matplotlib movies of the time evolution of ω and ψ that last until $t = 10$ (you can chose any reasonable frame rate). Some of the snapshots of ω are in Fig. 8.3 of Kutz, you can qualitatively check your results by comparison.

Problem 26 ODEs all the way: Method of Lines

(a) We mentioned the *method of lines* in class and showed that the instability of the FTCS scheme for the advection equation is a result of the instability of the forward Euler method for the resulting ODE system. Repeat the same procedure for RK2 and RK4. Specifically, calculate the condition on $z = \lambda h$ for stability, and plot the region on the complex plane. Based on these and the eigenvalues of the first derivative, determine the stability and the CFL conditions of RK2 and RK4 when used in the method of lines to solve the advection equation. The results are similar for all boundary conditions, but feel free to assume any if it makes things easier.
Hint: There was a link to the eigenvalues and eigenvectors of the discrete second derivative in problem 21, and you can calculate the eigenvalues of the first derivative matrices similarly. There is also a general formula for the eigenvalues of an $n \times n$ tridiagonal Toeplitz matrix:

$$\lambda_k = a + 2\sqrt{bc}\cos\left(\frac{k\pi}{n+1}\right) \quad , \quad k = 1, 2, \ldots n \ ,$$

where a is the diagonal element, and b and c are the super- and sub-diagonal ones, respectively.

(b) We saw how to recast the wave equation as an advection equation using the variables $(\partial_t u, \partial_x u)$ in class. Repeat the previous part for this case. We have a matrix here unlike the single variable advection equation, so even though the solutions are similar, this is not trivial.

(c) Repeat the previous part, but this time imitate our approach from the ODEs even more closely by reducing the wave equation to first order using the variables $(\partial_t u, u)$.

Problem 27 Evolving the wave all the possible ways (9 points)

Consider the wave equation

$$\frac{\partial^2 u}{\partial t^2} = v^2 \frac{\partial^2 u}{\partial x^2}$$

with periodic boundary conditions $u(t, 0) = u(t, L)$ and the initial conditions

$$u(0, x) = \sin\frac{\pi x}{L} + \frac{1}{4}\sin\frac{4\pi x}{L} + \frac{1}{16}\sin\frac{16\pi x}{L}$$
$$\partial_t u(0, x) = -\frac{\pi v}{L}\left(\cos\frac{\pi x}{L} + \cos\frac{4\pi x}{L} + \cos\frac{16\pi x}{L}\right) .$$

These initial conditions represent three perfect modes moving to the right. Use scaled variables so that $v = L = 1$. Choose Δx so that the shortest wavelength component in the initial data ($L/8$) can be considered short, and you can see the effects of finite differencing on different modes.

(a) Perform the time evolution using:

- FCTS without numerical dissipation ($\lambda = 0.25, 2$)
- Lax method ($\lambda = 0.25, 2$)
- Leapfrog method ($\lambda = 0.25, 2$)
- Cranck-Nicholson method ($\lambda = 0.25, 2$)
- Method of lines with RK4 with ($\partial_t u, u$) as first order variables ($\lambda = 0.25, 2, 4$)

for the CFL factors λ. Plot your results as 2D surfaces and comment on them, e.g. is there unphysical dissipation, dispersion? Any instabilities? How do these effects depend on the wavelength of the modes? You should continue the evolutions long enough that each mode goes through many periods. Compare the propagation speeds and accuracies of each method (the exact solution can be obtained analytically).

(b) We discussed the many benefits of numerical dissipation in class. The wave equation already has a second space derivative, so we add a fourth space derivative for enhanced stability

$$\frac{ds_j}{dt} = v^2 \frac{u_{j+1} - 2u_j + u_{j-1}}{(\Delta x)^2} \overbrace{-\frac{\epsilon}{16} u_{j+2} - 4u_{j+1} + 6u_j - 4u_{j-1} + u_{j-2}}^{\mathcal{L}_D} ,$$

where the numerical dissipation term is nothing but

$$\mathcal{L}_D = -(\Delta x)^4 \frac{\epsilon}{16} \frac{\partial^4 u}{\partial x^2} .$$

This term converges to zero as $\Delta x \to 0$ *no slower than the convergence of the methods we normally use*, and it kills off high frequencies as expected. von Neumann analysis shows that you need ϵ to be small for stability. $\epsilon < 1$ is needed for leapfrog, not sure about others schemes, but you typically choose $\epsilon \ll 1$ anyway.

Repeat part (a) for leapfrog and the method of lines with the dissipative term. Report any changes, comment on them. Run the code twice longer than the no dissipation schemes. Implement these such that you have an option for turning the dissipation on or off (rather than having two separate implementations). Try a a few different values of ϵ, and report the one you think to be the optimal.

(c) Changing the velocity parameter $v^2 \to v^2(1 + 0.1u^2)$ gives us a nonlinear wave equation. Repeat part (a) for for this case. For leapfrog and method of lines try the numerical dissipation both on and off. Comment as usual.

Problem 28 Time evolution with boundary conditions (Bonus, 6 points)

(a) Repeat part (b) from the previous problem for homogenous Neumann and Dirichlet BCs. The initial conditions in that case are not compatible with these boundary conditions, make sure you understand why. Change the initial values such that you have one short, one intermediate and one long wavelength mode as before, for each BC. This is important to see the effect of numerics on different wavelengths.

(b) Boundary conditions for PDEs can be vastly more complicated than the three standard options for ODEs. One commonly needed case is a *reflectionless*[1] boundary, where we want any incoming wave

[1] This BC goes under many names, *absorbing, Sommerfeld, radiation* and others. It is an active research field in higher dimensions where it is not usually possible to impose a BC that is reflectionless for all incidence angles.

to just go through as if there is no boundary. This is extremely useful if you are discretizing an infinite space, but only interested in what happens in a finite portion of it. You just discretize a finite piece of it with reflectionless boundaries, so that everything that hits the boundaries go on their way. Otherwise, you would get reflections, look at your results from part (a) again. This means you have to put boundaries far away so that unphysical reflections would not come back from the boundaries and spoil your calculation. You end up evolving the PDE in a much bigger region than you normally need to, leading to a big waste of resources.

How do we get reflectionless boundaries? No reflection means we only have waves moving the $+x$ direction at $x = L$, i.e.

$$u(t, x) = f(x - vt) \quad , \quad x = L \ .$$

But we know an equation whose solutions are exactly these, the advection!

$$\frac{\partial u}{\partial t} = -v \frac{\partial u}{\partial x} \ .$$

Then we impose this as our boundary condition

$$\left(\frac{\partial u}{\partial t} + v \frac{\partial u}{\partial x} \right) \Bigg|_{x=L} = 0 \ .$$

The condition is similar at $x = -L$, but we only allow left-moving waves, so you just change $v \to -v$.

Repeat part(a) with reflectionless boundaries and the initial conditions

$$u(0, x) = \begin{cases} \frac{1}{4} \sin \frac{4\pi x}{L} + \frac{1}{16} \sin \frac{16\pi x}{L} & L/4 < x < 3L/4 \\ 0 & \text{otherwise} \ , \end{cases}$$

$$\partial_t u(0, x) = 0 \ .$$

Make sure you impose the BCs using stencils that are at least as accurate as the numerical time evolution scheme. Comment on your results. Do you really see no reflection? Do your results depend on wavelength or the amount of numerical dissipation? Feel free to add other wavelengths and observe the results.

(c) Repeat part (b) with the nonlinear waves from the previous question. Compare the results to the linear case.

One example where reflectionless boundaries are important is gravitational wave simulations, so that we can make sure the gravitational waves formed by black hole mergers do not come back from the artificial outer boundary, and spoil the results (personally, i use other methods). Another, perhaps even more unusual, boundary condition used in numerical relativity is the "no boundary" boundary condition. Black holes have physical singularities in general relativity, meaning functions go to infinity at some place inside them. A computer cannot handle infinity, and one way to address this issue is *excision*: identifying where the infinity is, and removing a volume around it from the grid. If you do this, you have an inner boundary at the surface of the region you remove. What boundary condition do you impose here? No boundary condition at all! When you come to the boundary, you simply start using one sided stencils to calculate your space derivatives, and evolve in time as usual! This works because of the nature of the black hole. Remember that nothing can escape the black hole if you get within a certain region of it, i.e. if you go within the *event horizon*. So causality allows you to go in only one direction, inwards. This is not just words, any physical effect, and if you simulate it right any numerical effect, also travels inwards. If you choose your excision surface inside the event horizon, you know that nothing is coming out of your surface.[2] Then, imposing no condition at all works just fine.

The main point of this exercise is not teaching you new boundary conditions, rather I wanted to show that there are many possibilities out there. Depending on what problem you work on, you can use some of the above, or never hear of them again.

[2] In technical language, all characteristics are directed inwards.

Problem 29 Quantum mechanics on the grid (9 points)

Consider the time dependent quantum harmonic oscillator in two dimensions

$$i\hbar\frac{\partial\psi}{\partial t} = \left[-\frac{\hbar^2}{2m}\nabla^2 + \frac{1}{2}m\omega^2(x^2+y^2)\right]\psi \ = (H_x + H_y)\psi$$

with the initial data

$$\psi(t=0,x) = \frac{\psi_0(x)\psi_0(y) - \psi_2(x)\psi_1(y) + i\ \psi_3(x)\psi_4(y)}{\sqrt{3}}\ ,$$

where ψ_n are the wavefunctions for the n^{th} eigenstate of the one dimensional equation satisfying

$$\left[-\frac{\hbar^2}{2m}\frac{\partial^2}{\partial x^2} + \frac{1}{2}m\omega^2x^2\right]\psi_n(x) = \left(n+\frac{1}{2}\right)\hbar\omega\psi_n(x)\ .$$

These are the eigenfunctions you would get from separation of variables together with the boundary conditions $\psi(\pm\infty) = 0$. You can look up their exact forms at any source.[3]

You will evolve this system to $t = 32/\omega$. Make sure you understand that this is sufficient to see the qualitative behavior of the numerical evolution for the given initial data.

(a) Show that the exact analytical solution is

$$\psi(t=0,x) = \frac{e^{-i\omega t}\psi_0(x)\psi_0(y) - e^{-4i\omega t}\psi_2(x)\psi_1(y) + i\ e^{-8i\omega t}\psi_3(x)\psi_4(y)}{\sqrt{3}}\ ,$$

(b) The natural time scale is ω^{-1}, so we will measure t in this unit.[4]. Show that by using appropriate units for length ℓ, the Schrödinger equation reduces to

$$i\frac{\partial\psi}{\partial t} = \left[-\frac{1}{2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right) + \frac{x^2+y^2}{2}\right]\psi \ = (H_x + H_y)\psi\ .$$

Find ℓ. Use this form of the equation in your simulations. This is equivalent to scaling t and x to unitless quantities.

We encounter an infinite spatial domain once again. We used reflectionless boundary conditions in the previous problem which is an option for SE as well, but it requires a discussion of more advanced topics we do not want to get involved with. Instead, we will use an idea from the previous problem set and express the SE in compactified coordinates.

One should be more careful with coordinate changes in PDEs compared to ODEs since change of coordinates affect the CFL conditions. For our case, the implicit methods we will use are less sensitive to the CFL condition, which is good news. More importantly, in compactification the spatial distance between two grid points is getting larger in real space. This actually helps satisfy the CFL conditions where we want small time steps per space step in a rough sense. Doing the opposite, e.g. magnifying a small region in space because we want to investigate it better, could cause potential trouble. In general, be careful with coordinate changes and check your CFL condition.

Compactification might perform well for stability, but it necessarily makes you lose accuracy near infinity since you represent an infinite distance with only a few points, and in general you loose accuracy away from the center as the derivative of the compactification function increases (you have more and more real space per grid space). For example, you should pick a d_c in

$$x = \tan\frac{x_c}{d_c}\ ,\ x_c \in [0, \frac{\pi d_c}{2}]\ ,$$

[3]Be careful that there are two main different conventions for the Hermite polynomials which are used in defining these functions.

[4]This is equivalent to redefining $\omega t \to t$ in our convention of keeping the same symbols after scaling them

and similarly in y, such that the main features of your solution are near the center. This way you do not lose valuable information about the functions, and only use the compactification to handle the relatively boring exponential decay.

(c) Perform the numerical evolution using the Crank-Nicholson method

$$\psi_{ij}^{n+1} = \psi_{ij}^{n} + \frac{\Delta t}{(\Delta x)^2}\left[\frac{(\bar{H}_x\psi)_{ij}^{n+1} + (\bar{H}_x\psi)_{ij}^{n}}{2} + \frac{(\bar{H}_y\psi)_{ij}^{n+1} + (\bar{H}_y\psi)_{ij}^{n}}{2}\right]$$

where $\bar{H} = \Delta x^2 H$, e.g.

$$(\bar{H}_x\psi)_{ij}^{n} = -\frac{\psi_{i+1,j}^{n} - 2\psi_{ij}^{n} + \psi_{i-1,j}^{n}}{2} + \frac{x^2\Delta x^2\psi_{ij}^{n}}{2}$$

Solving for the future time step in the implicit Crank-Nicholson scheme is cheap in one space dimensions since we need to invert a tridiagonal matrix. In D dimensions, we get *fringes*, bands in the matrix typically a distance $\mathcal{O}(h^{-(D-1)})$ from the main diagonal which makes solution considerably harder. Have different options for your linear solution in the CN step: standard `solve`, sparse `spsolve`, banded `solve_banded`, and possibly others.

Plot, report, compare and comment as usual.

(d) Sparse solvers and special numerical techniques can speed up CN quite a bit, but a more powerful and far reaching technique in higher dimensions is *alternating direction implicit (ADI) method*, which is a basic form of *operator splitting*. Instead of taking one implicit time step in both variables, we take two half steps where the evolution is implicit in only one of the dimensions:

$$\begin{aligned}\psi_{ij}^{n+1/2} &= \psi_{ij}^{n} + \frac{\Delta t}{2(\Delta x)^2}\left[(\bar{H}_x\psi)_{ij}^{n+1/2} + (\bar{H}_y\psi)_{ij}^{n}\right] \\ \psi_{ij}^{n+1} &= \psi_{ij}^{n+1/2} + \frac{\Delta t}{2(\Delta x)^2}\left[(\bar{H}_x\psi)_{ij}^{n+1/2} + (\bar{H}_y\psi)_{ij}^{n+1}\right] \ .\end{aligned}$$

Show that, the total effect of these two steps has the same order of convergence with Crank-Nicholson.

Why bother with the two steps? Show that each half step is implicit, but the linear system to be solved has a tridiagonal symmetric matrix, which is a lot easier to solve than the banded matrices in usual CN where bands are far away from the diagonal. Explicitly write the linear system to be solved for each half step. Do not implement this method yet.

(e) Implement ADI in Python with the following modification: perform the two half steps in both orders, that is x first y second and y first x second, and average the two as your result for the time step $n+1$. This is a simple, and somewhat wasteful, symmetric operator splitting scheme which usually improvs convergence. A well known example of this class is the *Strang splitting*.

(f) Let's go back to one spatial dimension and examine the time dependent nonlinear Schrödinger equation

$$i\frac{\partial\psi}{\partial t} = \overbrace{-\frac{1}{2}\frac{\partial^2\psi}{\partial x^2}}^{L[\psi]}\overbrace{-|\psi|^2\psi}^{N[\psi]}$$

with the initial condition

$$\psi(t=0,x) = \mathrm{sech}x + \epsilon\ e^{-(x^2+y^2)}(2x^2-1)(2y^2-1)\ .$$

You might remember the $\epsilon = 0$ case to be the exact solution of the time independent equation from problem 23. Evolve this system for $\epsilon = 0, 0.1$ using CN, and repeat the usual analysis. Can you see that

the exact solution preserves its shape despite the strong nonlinearity? The problem is conceptually similar to the previous cases, but you need to solve a nonlinear root finding problem at every step which is much costlier. [5]

Now, let us re-solve this problem with another example of operator splitting. Recall that the general logic of operator splitting is evolving the parts of the RHS ($L[\psi]$ and $N[\psi]$) separately with different methods that is cheaper than solving them together. Section 9.5 of the textbook outlines the main idea. At every time step, first evolve the linear piece with CN, and realize that root finding is much easier now. The nonlinear part is typically the hard one, but here it is is just like an ODE, and can be solved exactly as

$$i\frac{\partial\psi}{\partial t} = -|\psi|^2\psi \implies u_j^{n+1} = u_j^n e^{i|u_j^n|^2\Delta t} \ .$$

Note that you do this repeatedly for each single time step Δt, not globally for all t. Still, this costs almost nothing!

Unlike ADI we are not taking any half time steps here. First, we are evolving for a full time step Δt using CN as if $L[\psi]$ is the only RHS. Then, *starting from the result of the CN evolution*, we evolve a full time step as if $N[\psi]$ is the only RHS, which can be done analytically as explained above.

Compare the solution from this method to the previous one and to the exact solution when $\epsilon = 0$. Consider both accuracy and speed. Try symmetrizing the operator splitting, and see if anything changes.

This was a very short glimpse at operator splitting which is exceedingly important in numerically modeling complex nonlinear systems. You would encounter these methods everywhere from astrophysics to biophysics.

Optional for those who enjoy coding (no points, bonus or otherwise)

Below is a challenging problem I asked last year, but I decided to remove it this year. It has some valuable lessons, so have a look at it when you have time. At least read part (a), which is theoretical, and has some relevance to the final project.

When a point charge is inside an ionic fluid, the oppositely charged ions are preferentially attracted to the the point charge, and tend to neutralize the electric field. This is called electric field screening, and the best known simplified analysis of this problem is the Debye-Hückel theory. We will have a toy-model analysis of this theory for a large molecule in a conducting nano-reservoir. The electrical potential in such a set-up approximately obeys the following equation:

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial y^2} = -\frac{\rho_0}{\epsilon_0}\exp\left(-\frac{q\phi}{k_B T}\right) \ . \tag{8}$$

The exponential term is the usual Boltzmann factor for the ions. The boundary conditions are such that $\phi = 0$ on the $x = \pm L$, $y = \pm L$, $z = \pm L$ planes, i.e. on the surface of a cube of side length $2L$ (the conducting container), and $\phi = V_0$ on the surface $x^2 + y^2 + z^2 = a^2 < L^2$ (the "charged particle" in the container).

(a) Show that you can scale the relevant quantities so that the Poisson equation reduces to

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial y^2} = -\exp\left(-\frac{\phi}{\phi_0}\right) \ , \tag{9}$$

with vanishing ϕ at $x = \pm 1$, $y = \pm 1$, $z = \pm 1$, and $\phi(x, y, z|\ x^2 + y^2 + z^2 = \alpha^2) = v_0$. Find α and ϕ_0.

We made a trick here. You can remember that when we scaled our quantities in the previous problems, we renamed them such as $\rho \equiv \frac{r}{r_0}$ etc. This becomes tedious after a while and you run out of symbols,

[5] This is an example of a soliton. The observation and understanding of why orderly behavior emerges in many nonlinear systems has had a huge impact in both physics and mathematics.

so you simply keep the same symbol after the scaling $\frac{r}{r_0} \to r$, and remember that you are using the scaled quantities. This is actually more natural than we describe it to be. Scaling lengths with r_0 is equivalent to using the units of r_0 to measure lengths rather than whatever we were using before (SI units in our case). There is no need to change the symbols of quantities when we use different units. So, x, y, z are all quantities of length in both equations, but we use meters to measure them in Eq. 8, and Ls to measure them in Eq. 9. After all, measuring lengths in meters is nothing but counting how many meters there are in them, i.e. scaling them by a meter! **We will continue to use this logic in the final project.**

(b) Solve Eq. 9 for $\alpha = 0.5$, $v_0 = 1$, $\phi_0 = 10$, i.e we have the molecule in a rather tight box. There are two big differences from Problem 24.

- The PDE is nonlinear, hence you need to use relaxation rather than direct matrix solution.
- There is an inner boundary which is not a planar surface. Treat this surface in the most straightforward way: fix all the points inside the inner boundary as

$$\phi(x_i, y_j, z_k|\ x_i^2 + y_j^2 + z_k^2 \leq \alpha^2) = v_0$$

Schematically, you need to set up a grid, and solve a system of nonlinear equations for each point in the grid between the boundaries. After that point, you just solve the same way you solve any root finding problem. First try solving it with the builtin solver if you can. It will run into trouble if you have a fine grid, but should work for some very course grids.

Our second method will be manually implemented: the *nonlinear Gauss-Seidel scheme*.[6] Here, we will consider the equation for a particular grid point I[7] on its own, ϕ_I being the unknown to be nonlinearly solved for while keeping the other grid point functions constant:

$$L_I(\ldots, \phi_I, \ldots,) = 0$$
$$\implies \phi_I^{\text{new}} = \phi_I^{\text{old}} - \frac{L_I(\vec{\phi}^{\text{old}})}{(\partial L_I/\partial \phi_I)|_{\vec{\phi}^{\text{old}}}} \ .$$

Namely, you do *one* Newton iteration at each grid point. You keep the main aspect of Gauss-Seidel in that once you update a grid point, you immediately start using the updated value at the next point.

Recall that in the standard method for solving systems of nonlinear equations, at every iteration step we linearize the whole system for all ϕ_I using the Jacobian, and solve the linearized equations. We start the next iteration by linearizing around the solution of the previous iteration. The nonlinear Gauss-Seidel is *almost* equivalent to this. When the only nonlinearity at a grid point is due to the grid function at that point, and the neighboring grid points behave linearly, then the linearization of the system can be done individually at each grid point. The Newton iteration above is simply a single iteration of Gauss-Seidel over the linearized equations. Mind that a single Gauss-Seidel iteration does **not** solve the linearized system of equations, it just brings our initial guess, hopefully, closer to the true solution. But in many situations we do not need to solve this equation fully anyway, remember that the linearized equation itself is only one iteration step in solving the fully nonlinear solution. Instead of fully solving this linearized equation, we *partially* solve it by applying another iterative method, this time a method for solving *linear* equations. Let us first take the extreme approach, just apply a single Gauss-Seidel iteration to bring our guess a little bit closer to the true solution of the linearized equation, and then before attempting to get a better approximate solution, immediately renew the linearization. Then try up to 10 Gauss-Seidel iterations, which are commonly called *s*weeps in this context, at one linearized equation before linearizing again. Compare the results. How many GS sweeps give you the shortest overall computation time? Note that there is not a magic number here, optimal number of sweeps you find will not apply to other PDEs.

[6] I call this method as such imitating *Numerical Recipes*, but I am not sure how common this naming is.
[7] Here we use the single index I to cumulatively represent the three indices (ijk).

Note that I have not given any grid spacing or initial guess for ϕ. These are yours to decide and *optimize*. There is no need to build a matrix in the nonlinear Gauss-Seidel, which also means there is no need to have a single 1D vector to store ϕ. Actually, it is more natural to store it on a 3D array with three indices ϕ_{ijk}. At each Gauss-Seidel sweep you just have a triple `for` loop over `i, j ,k` and update ϕ_{ijk} as prescribed above. You should of course skip updating the boundaries, and in our case the inside of the inner boundary.

When do you stop linearizing and sweeping? A good measure of the success of a guess function $\phi^{(\mathrm{g})}$ is given by the residual

$$R[\phi^{(\mathrm{g})}] = \vec{\nabla}^2\phi^{(\mathrm{g})} + \exp\left(-\frac{\phi^{(\mathrm{g})}}{\phi_0}\right) ,$$

which should ideally be 0. Calculate it at every step, and stop when it is below a certain tolerance.

If you could make the builtin solver work, compare your results to the relaxation method.

(c) In the previous part, plot your residual as a function of the number of relaxation sweeps performed. How does the number of sweeps needed for a *satisfactory* solution change with the grid size h for the optimal number of sweeps for a given linearization?

(d) Depending on your initial guess, you might or might not see in your solution that relaxation is very efficient in the beginning, but loses its effectiveness near the end. To see this, start with an initial guess that has features at many scales, i.e. a function with smooth (long wavelength, low k) as well as very fine (short wavelength, high k) components. Observe that short wavelength modes are erased by relaxation quite fast, but the long ones persist for a long while. This is usually the bottleneck in relaxation methods: a satisfactory solution takes prohibitively long for typical numbers of grid points.

The easiest way to see the reason for this is going to the second derivative matrix in 2D. We found its eigenvalues and eigenvectors in problem 21, and you can see the exact results in Wikipedia. By inspecting these, explain why relaxation is inefficient for long wavelength modes. *Hint: In the linear problem, relaxation is simply applying the second derivative matrix.*

Relaxation is rarely used by itself to solve large scale nonlinear elliptic equations. *Multigrid methods* are the common approach which use the fact that one can transfer the problem to a coarser grid after some relaxation sweeps. The computational cost is lower, and also coarser grids happen to be more efficient to relax long wavelengths. This is repeated for successively coarser grids, and when the long wavelength piece is solved properly, the result is interpolated back to finer grids. The computational cost is linear in the number of grid points, which is hard to beat. We just mention this as general information, you do not need to know the details of multigrid methods.