0.1 Pseudospectral Method for Nonlinear PDEs

0.2 Problem Statement and Solution

Up to now we have considered linear problems, which may be treated exclusively in Fourier space. We have seen that in this case spectral methods yield a highly accurate and simple way to calculate derivatives. We now want to generalize this method to nonlinear partial differential equations. To this end consider the PDE for the field $f(x,t)$

$$\frac{\partial}{\partial t} f(x,t) = L(f(x,t)) + N(f(x,t)) \tag{1}$$

The right hand side involves a linear operator L, which may contain linear functions of f like multiplication with a constant factor or linear differential operators like the Laplacian. N denotes a nonlinear operator, which for example contains powers of f or nonlinear differential operators. The Fourier transform of equation (1) yields

$$\frac{\partial}{\partial t} \tilde{f}(k,t) = \tilde{L}(\tilde{f}(k,t)) + \mathcal{F}[N(f(x,t))] \tag{2}$$

While the Fourier transforms of the linear terms of equation (1) can be written down in a straight-forward manner, the nonlinear term needs further treatment. It has to be noted, that it is possible in principle to treat the nonlinear term in Fourier space, which is made clear best with an example. Consider the nonlinearity

$$N(f(x,t)) = f(x,t)f(x,t) \tag{3}$$

The Fourier transform is readily written down as

$$\mathcal{F}[N(f(x,t))] = \mathcal{F}(f(x,t)) * \mathcal{F}(f(x,t)) = \tilde{f}(k,t) * \tilde{f}(k,t) \tag{4}$$

That is, this simple quadratic nonlinearity leads to a convolution in Fourier space. Let n denote the number of grid points used for the discretization of f. Then the above expression involves n^2 operations. Of course, the situation gets even worse for a higher nonlinearity. To circumvent this problem we proceed as follows. Instead of evaluating the nonlinearity in Fourier space, we may transform $\tilde{f}(k,t)$ back to real space. The computational cost for this operation scales with $n \log n$ thanks to the *fast fourier transform* algorithm. Back in real space the multiplication may easily be done with an operation scaling with n. After evaluating the nonlinearity in real space, we transform

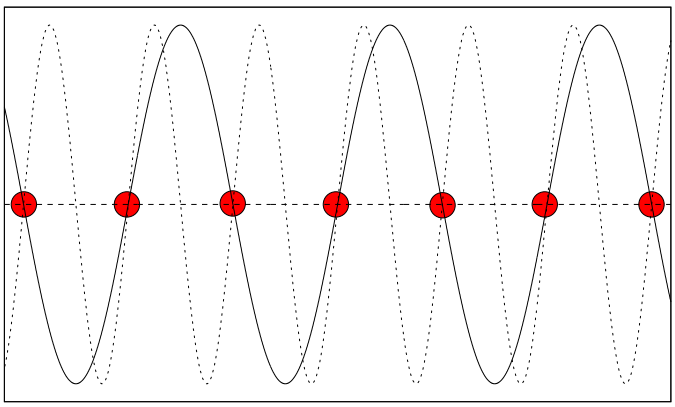

Figure 1: A rapidly oscillating sinusoidal function cannot be resolved by the red grid points and hence is misinterpreted as a sinusoidal function with a lower frequency.

back to Fourierspace, again at a cost scaling with $n \log n$. If the nonlinearity contains not only powers of f but also linear derivatives, these may also be calculated in Fourier space and transformed back together with $\tilde{f}$. Within this procedure no operation more expensive than $n \log n$ had to be executed, yielding an excellent overall performance of the algorithm compared to the brute force way.

0.3 Aliasing and Dealiasing

Whenever treating a nonlinear partial differential equation, the danger of introducing aliasing errors to the numerical solution lurks around the corner. Consider for example the simple nonlinearity $f(x,t)f(x,t)$ introduced in the last paragraph. Now let f be initialized with a highly oscillating sine, which is hardly resolved by the numerical grid. Evaluating the the nonlinearity yields

$$f(x,t)f(x,t) = \sin(\omega x)\sin(\omega x) = -\frac{1}{2}\cos(2\omega x) \tag{5}$$

i.e. the nonlinearity introduces an even more rapid oscillating cosine, which cannot be resolved by the numerical grid. As a consequence this oscillating function is interpreted as a oscillating function with a lower frequency. This is also visualized in figure 1. Depending on your problem, this effect may introduce severe errors to your numerical solution or even result in a numerical instability.

There are a at least two ways which help to avoid this effect. Probably the most simple one is to choose a grid which is fine enough to accurately resolve all the scales which may appear during the course of the simulation. This is an appropriate solution for many physical systems. Considering for example model equations for pattern formation, here often a small wave vector band is excited and the spectral density of the field rapidly decays far from these excited modes.

In case of, for example, Navier-Stokes turbulence the situation turns out to be a bit different. Here, the spectral density of, say, the kinetic energy decays algebraically (with a $k^{-\frac{5}{3}}$ spectrum) indicating that a wide range of spatial modes are excited. Together with the fact, that we have a quadratic nonlinearity for this problems, a pseudospectral treatment of this problems yields an alternative solution called dealiasing. Different approaches to de-alias the fields have been proposed in the literature, the easiest being the two-thirds rule introduced by Orszag. Let $k_{\max}$ denote the highest wavenumber resolved by the numerical grid. Provided, that all fields involved in calculating the nonlinearity are filtered according to

$$\tilde{f}(k,t) = \begin{cases} \tilde{f}(k,t) & \text{if} \quad k < \frac{2}{3}k_{\max} \\ 0 & \text{else} \end{cases} \tag{6}$$

ensures that the numerical solution is free from aliasing errors. It may be reasoned from figure 2, that by this truncation of the input fields in Fourier space a quadratic nonlinearity may only produce higher modes which are readily filtered after calculating the nonlinearity. It is very easy to apply this method. You only have to

- null the upper third of Fourier coefficients of all fields which are involved in calculating the nonlinearity
- null the upper third of Fourier coefficients of the array holding the Fourier coefficients of the nonlinearity after transforming back into Fourier space.

It has to be stressed that this method only helps to suppress aliasing errors and does not increase the resolution of the numerical solution. A further disadvantage of this simple method is that, being a sharp filter in Fourier space, it may introduce Gibbs oscillations to your numerical solution. So the best way to avoid aliasing errors is to accurately resolve the numerical solution of your problem not pushing it to the limit.

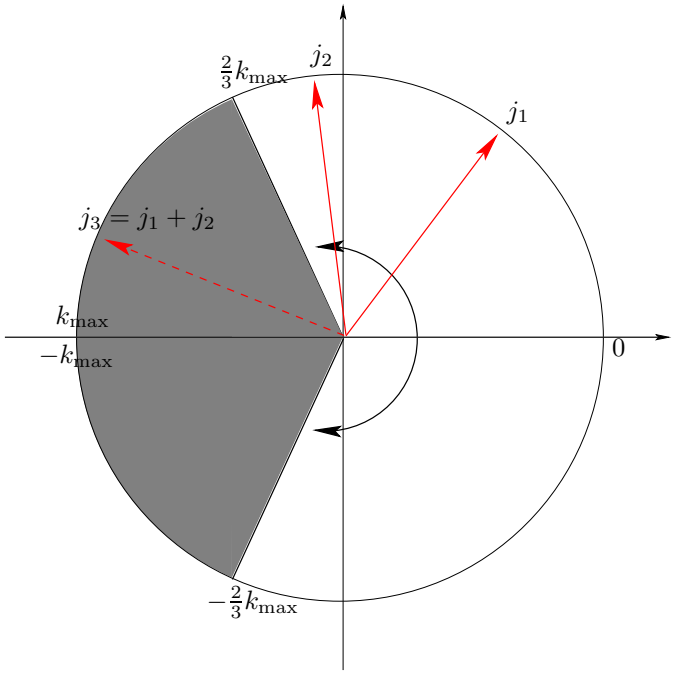

Figure 2: Two-thirds rule by Orszag. A coupling of two modes corresponds to a addition of angles in this figure. After the Fourier coefficients of the input fields are nulled according to the gray-shaded areas, the sum of two arbitrary modes j_1 and j_2 results in a mode j_3 lying in the gray-shaded area. This is nulled again after the nonlinearity is transformed back to Fourier space.

0.4 A Simple Example: Burgers Equation

Consider a real velocity field $u(x,t)$ which obeys the simple nonlinear PDE

$$\frac{\partial}{\partial t}u(x,t) + u(x,t)\frac{\partial}{\partial x}u(x,t) = \nu\frac{\partial^2}{\partial x^2}u(x,t) \tag{7}$$

which is known as Burgers equation. It shares the advective nonlinearity with the Navier-Stokes equation, albeit in a single spatial dimension. At the same time, the huge mathematical difficulty arising due to the nonlocal pressure term is absent in this equation. According to the above notation the linear and nonlinear operators for this PDE take the form

$$\begin{aligned} L(u(x,t)) &= \nu\frac{\partial^2}{\partial x^2}u(x,t) \\ N(u(x,t)) &= -u(x,t)\frac{\partial}{\partial x}u(x,t) \end{aligned} \tag{8}$$

ν denotes the kinematic viscosity. The Burgers equation is known to steepen negative gradients leading to the formation of so-called shocks. These shocks are smoothed out by viscous effects. For a sufficiently high viscosity, this equation may efficiently be solved by a pseudospectral method, as we want to exemplify in the following. Time-stepping can be achieved with a numerical scheme of your choice, fourth-order Runge-Kutta methods turn out to do a rather good job. After initializing an initial condition $u(x, t = 0)$ the field is transformed to Fourier space yielding $\tilde{u}(k, t = 0)$. The right hand side then may be treated in several easy steps according to

- dealias $\tilde{u}(k,t)$
- calculate the derivative according to $ik\tilde{u}(k,t)$
- transform $\tilde{u}(k,t)$ and $ik\tilde{u}(k,t)$ back to real space
- calculate the nonlinearity $N(x,t) = u(x,t) \cdot \frac{\partial}{\partial x}u(x,t)$
- transform $N(x,t)$ back to Fourier space
- dealias $\tilde{N}(k,t)$
- calculate the Laplacian $-k^2\tilde{u}(k,t)$
- build the output array according to $\tilde{N}(k,t) - \nu k^2\tilde{u}(k,t)$

Taking for example a sinusoidal function as an initial conditions, negative gradients steepen yielding the characteristic shock structure.

0.5 Appendix: Ordering of Fourier Coefficients

Due to algorithmic details of the fast Fourier transform the Fourier coefficients are ordered in a somewhat "strange" manner, at least for a newby. This causes problems especially when calculating derivatives of functions in Fourier space, as calculating the derivative corresponds to a multiplication with a real wave number (or vector in more than one spatial dimension). Hence the wave vectors have to be stored in a real array with exactly the same ordering as the Fourier coefficients. A complex field sampled by N grid points is transformed to a complex array of Fourier coefficients being ordered according to

$$\tilde{f}_0, \tilde{f}_1, \ldots, \tilde{f}_{\frac{N}{2}}, \tilde{f}_{-\frac{N}{2}+1}, \tilde{f}_{-\frac{N}{2}+2}, \ldots, \tilde{f}_{-1} \tag{9}$$

If we consider a real field f the Fourier coefficients exhibit a symmetry $\tilde{f}_{-j} = \tilde{f}_j^*$. This is exploited by most Fourier transform by only storing roughly half of the array, reducing the memory needed for holding the coefficients and giving a speed increase of roughly a factor of two. The Fourier coefficients in Fourier space are then ordered according to

$$\tilde{f}_0, \tilde{f}_1, \ldots, \tilde{f}_{\frac{N}{2}} \tag{10}$$

Let L denote the physical length of the simulation domain. The real array of wave numbers then takes the form

$$k(i) = \begin{cases} \frac{2\pi}{L} i & \text{if} \quad i = 0, \ldots, \frac{N}{2} \\ \frac{2\pi}{L}(-N+i) & \text{if} \quad i = \frac{N}{2}+1, \ldots, N-1 \end{cases} \tag{11}$$

In case of a real field, the above-mentioned symmetry may be exploited. Then the array holding the wave vectors has also roughly half the size resulting in the index i running only $i = 0, \ldots, \frac{N}{2}$. These considerations are generalized to a two-dimensional field $f(\boldsymbol{x}, t)$ in a straight-forward manner. In this case we have a two-dimensional array for the Fourier coefficients. In case f represents a complex field, the wave vectors are arranged according to

$$k_x(i,j) = \begin{cases} \frac{2\pi}{L} i & \text{if} \quad i = 0, \ldots, \frac{N}{2} \\ \frac{2\pi}{L}(-N+i) & \text{if} \quad i = \frac{N}{2}+1, \ldots, N-1 \end{cases} \tag{12}$$

$$k_y(i,j) = \begin{cases} \frac{2\pi}{L} j & \text{if} \quad j = 0, \ldots, \frac{N}{2} \\ \frac{2\pi}{L}(-N+j) & \text{if} \quad j = \frac{N}{2}+1, \ldots, N-1 \end{cases}$$

In case of a real field, half of the Fourier coefficients suffice and the index i for the wave vectors reduces to $i = 0, \ldots, \frac{N}{2}$. One should note that different implementations of the fast Fourier algorithms exhibit a different ordering of Fourier coefficients. The ordering presented here is for example used by the `FFTW` library.

0.6 Appendix: Flow Diagram for a Typical Simulation Code