Pseudospectral methods for the practitioner (Draft)

R.Toral

March 9, 2002

ALGORITHM/algorithm.tex

1 Fourier series versus Fourier discrete transform

As an introduction, let us describe exactly the relation between the Fourier series and the Fourier discrete transform. In these notes we will be interested in (real or complex) functions $f(x)$ defined in a d-dimensional space $x \in \mathcal{R}^d$. The function $f(x)$ is periodic in all the space dimensions, i.e.:

$$f(x + L\hat{\mathrm{e}}_i) = f(x) \qquad\qquad i = 1, \ldots, d \tag{1}$$

and $\hat{\mathrm{e}}_i$ is the unitary vector in space-direction i: $\hat{\mathrm{e}}_1 = (1, 0, 0, \ldots)$, $\hat{\mathrm{e}}_2 = (0, 1, 0, \ldots)$, etc. We introduce the Fourier transform of periodic functions as:

$$\tilde{f}(q) = \mathcal{F}_P[f(x)] \equiv \frac{1}{L^d} \int_{[0,L]^d} dx\ f(x) \mathrm{e}^{iqx} \tag{2}$$

We introduce the following notation:

$$\tilde{f}_k = \tilde{f}(q_k), \qquad\qquad q_k = \frac{2\pi}{L} k \tag{3}$$

and $k = (k_1, \ldots, k_d)$. It is well known that (given some general conditions) the following Fourier series is an exact representation of $f(x)$:

$$f(x) = \mathcal{F}_P^{-1}[\tilde{f}(q)] \equiv \sum_{k=-\infty}^{\infty} \tilde{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{4}$$

From the numerical point of view, it is very expensive to compute the $\tilde{f}_k$ because their calculation involves a d-dimensional integration. Therefore, the **first approximation** is to compute them using only the values of the function $f(x)$ at selected lattice points $x_n = n\Delta x$, for $n = (n_1, \ldots, n_d)$ and $n_i = 0, \ldots, N-1$. The physical length is $L = N\Delta x$. We compute the coefficients of the Fourier series using only $f_n \equiv f(x_n)$,

$$\tilde{f}_k \approx \frac{1}{N^d} \hat{f}_k \tag{5}$$

where we have defined define the discrete Fourier transform as [1]:

$$\hat{f}_k = \mathcal{F}_D[f_n] \equiv \sum_{n=0}^{N-1} f_n \mathrm{e}^{i\frac{2\pi}{N}kn} \tag{6}$$

Although the coefficiens of the discrete Fourier transform, $\hat{f}_k$ are an approximation to the coefficients of the Fourier series, $\tilde{f}_k$, there is an exact relation between $\tilde{f}_k$ and $\hat{f}_k$:

$$\hat{f}_k = N^d \sum_{m=-\infty}^{\infty} \tilde{f}_{k+mN} \tag{7}$$

If $f(x)$ is a real function, we have the additional property:

$$\hat{f}_k^* = \hat{f}_{-k} \tag{8}$$

The **second approximation** is that the infinite sum in (4) is replaced by a finite one using the discrete coefficients:

$$f(x) \approx \bar{f}(x) \equiv \frac{1}{N^d} \sum_{k=k_1}^{k_2} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{9}$$

The nice thing of these two approximations (5) and (9) is that we can demand that they cancel each other and the previous formula becomes then exact for the lattice points x_n, i.e. $f(x_n) = \bar{f}(x_n)$. This is achieved by using any set of values k_1, k_2 such that $k_2 - k_1 = N - 1$. Therefore, we define the discrete inverse Fourier transform as:

$$\mathcal{F}_D^{-1}[\hat{f}_k] \equiv \frac{1}{N^d} \sum_{k=k_1}^{k_1+N-1} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{N}kn} \tag{10}$$

[1] In this and other cases, we use a "pseudo" 1-d notation. It should be clear what is meant for the indexes in the case $d > 1$

(the value of k_1 is irrelevant in this definition and the usual definition takes simply $k_1 = 0$). Using the property

$$\frac{1}{N^d} \sum_n \mathrm{e}^{i\frac{2\pi}{N}nk} = \delta_{k,0} \tag{11}$$

it is easy to prove that

$$\mathcal{F}_D^{-1}[\hat{f}_k] = f(x_n) = f_n \tag{12}$$

Of course, the nice thing about the discrete Fourier transforms $\mathcal{F}_D$ and $\mathcal{F}_D^{-1}$ is that they can be computed numerically very efficiently. There are several routines to perform the discrete Fourier transforms. One has to make sure that the factors of N and π used in their particular definition agree with the ones used here.

The question now is which values of k_1 and k_2 should we use. Since we just said that any values such that $k_2 - k_1 = N - 1$ would do as far as the values $f(x_n)$ are concerned, what do we mean by choosing the right values for k_1 and k_2? The answer is that we mean that the function $\bar{f}(x)$ is a good approximation to $f(x)$ *also at other values of* x.

To choose k_1 and k_2 in order to obtain the best approximation of $\bar{f}(x)$ to $f(x)$ we have to understand a little bit more deeply the relation between the coefficients of the Fourier series and the coefficients of the discrete Fourier transform. The way we have defined them implies that the $\hat{f}_k$ are an approximation to the $\tilde{f}_k$. How good an approximation? It turns out that the approximation is very bad for large values of $|k|$. And the reason is due to a property of the $\hat{f}_k$ called "aliasing" which is not present in the coefficients $\tilde{f}_k$. The property, which is easily derived from Eq.(6), states that the coefficients of the discrete Fourier transform are periodic with a period of N in the indexes:

$$\hat{f}_{k+N} = \hat{f}_k \tag{13}$$

(again, we use here a 1-d notation; the formula is valid if we increase any of the coefficients of the vector of indexes k by N). This formula states, for example, that $\hat{f}_N = \hat{f}_0$, whereas the coefficients of the Fourier series $\tilde{f}_k$ for a well behaved function decrease with k for large k. In practise, it means that the approximation (5) of $\hat{f}_k$ to $\tilde{f}_k$ begins to worsen for $|k| > N/2$ (see figure (1) later). For $|k| > N/2$ the $\hat{f}_k$ are given the values of their "alias" modes. For instance, when $N = 8$, $\hat{f}_{18} = \hat{f}_{10} = \hat{f}_2 = \hat{f}_{-6} \ldots$. But we can only affirm that $\tilde{f}_2 \approx \hat{f}_2/N$, we can not say that $\tilde{f}_{10} \approx \hat{f}_{10}/N$.

This result and the fact that the Fourier series involves an infinite sum of positive and negative terms, leads us to keep in the truncated Fourier series (9) a symmetric interval for (k_1, k_2). If $N = 2M + 1$ is an odd number, the obvious choice is $(k_1, k_2) = (-M, M)$ and the series is:

$$f(x) \approx \frac{1}{N^d} \sum_{k=-M}^{M} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{14}$$

If $f(x)$ is a real function, this choice (and any other satisfying $k_1 = -k_2$) has the additional advantage that the imaginary part of (9) is exactly zero for all values of x. Let us prove that (9) is real for $k_1 = -k_2$:

$$\left(\sum_{k=-k_2}^{k_2} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \right)^* = \sum_{k=-k_2}^{k_2} \hat{f}_k^* \mathrm{e}^{i\frac{2\pi}{L}kx} = \sum_{k=-k_2}^{k_2} \hat{f}_{-k} \mathrm{e}^{i\frac{2\pi}{L}kx} = \sum_{k=-k_2}^{k_2} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{15}$$

If $N = 2M$ is an even number (the commonest case in practice due to the fact that fast Fourier routines are more effective and the only one considered henceforth), one can not fulfill the requirements $k_1 = -k_2$ and $k_2 - k_1 = N - 1$. In this case there are two equally reasonable choices $(k_1, k_2) = (-M + 1, M)$ or $(k_1, k_2) = (-M, M - 1)$. In the first option we write the Fourier series as:

$$f(x) \approx \bar{f}_1(x) \equiv \frac{1}{N^d} \sum_{k=-M+1}^{M} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{16}$$

However, for a real function $f(x)$, this expression has in general a non-zero imaginary part for arbitrary values of x. The best alternative in the case of N even is to use a truly symmetric choice, namely:

$$f(x) \approx \bar{f}_2(x) \equiv \frac{1}{N^d} \left[\frac{1}{2} \hat{f}_{-M} \mathrm{e}^{i\frac{2\pi}{L}Mx} + \sum_{k=-M+1}^{M-1} \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} + \frac{1}{2} \hat{f}_M \mathrm{e}^{-i\frac{2\pi}{L}Mx} \right] \tag{17}$$

which will be written as

$$f(x) \approx \bar{f}_2(x) \equiv \frac{1}{N^d} {\sum_{k=-M}^{M}}' \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{18}$$

where the prime in the sum indicates that the first and last terms are halved. This symmetric option is a little bit more messy to implement in practice

although it has the advantage that the imaginary part is exactly zero. The only difference between the expressions (16) and (18) is in the imaginary part. In fact if we substract (16) from (18) we obtain:

$$\frac{i\hat{f}_M}{N^d}\sin\left(\frac{2\pi}{L}Mx\right) \tag{19}$$

in the case $N = 2M$ we can prove using (8,13) that $\hat{f}_{-M} = \hat{f}_M$ is real and, therefore, this last expression is purely imaginary (although at the lattice points $x_n = n\Delta x$ this imaginary part vanishes). Therefore, an alternative expression to (18) somewhat more useful for the numerical programming is to take the real part of (16):

$$\bar{f}_2(x) = \mathcal{R}\left[\frac{1}{N^d}\sum_{k=-M+1}^{M}\hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx}\right] \tag{20}$$

Following Sanz-Serna, we use as an example the function $f(x) = 3/(5 - 4\cos x)$. This function is periodic of period $L = 2\pi$. The coefficients of the Fourier series are given by $\tilde{f}_k = 2^{-|k|}$. The coefficients of the discrete Fourier transform, $\hat{f}_k$, depend on the number of points N used in the discretization of $(0, 2\pi)$. Figure (1) shows the difference between the coefficients $\hat{f}_k$ and $\tilde{f}_k$ for different values of N.

Figure (2) shows the function and its different approximants, while figure (3) shows the imaginary part of the approximants. We have also included the approximant that uses the exact Fourier coefficients, namely:

$$\frac{1}{N}\sum_{k=-N/2}^{N/2}\tilde{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{21}$$

and the one that makes a big alias error (although is the one that comes naturally from the inverse discrete Fourier transform):

$$\frac{1}{N}\sum_{k=0}^{N-1}\hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{22}$$

The message is that, in the case of even $N = 2M$ one has to use either approximant (16), (18). The second one is better because the imaginary part of this approximant, for real functions, is always equal to zero. Alternatively one culd use the approximant (21) which uses the exact coefficients, but it is very rare in practice to know the exact coefficients.

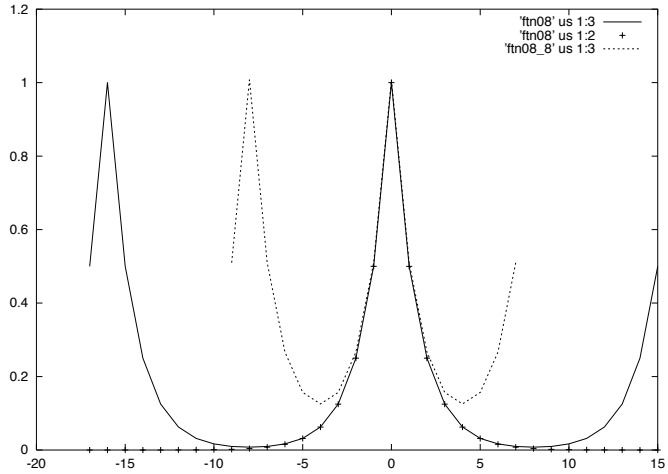

Figure 1: Comparison between the coefficients of the Fourier series, $\tilde{f}_k$ (symbols), and those of the discrete Fourier series, $N\hat{f}_k$ (lines), for the function $f(x) = 3/(5 - 4\cos x)$. The dotted line corresponds to $N = 8$ and the solid line to $N = 16$. Notice that, due to aliasing, the discrete coefficients are a good approximation to the coefficients of the (infinite) Fourier series only for $|k| < N/2$. We have used the program raul/ALGORITHM/dif.f to produce the data for these plots.

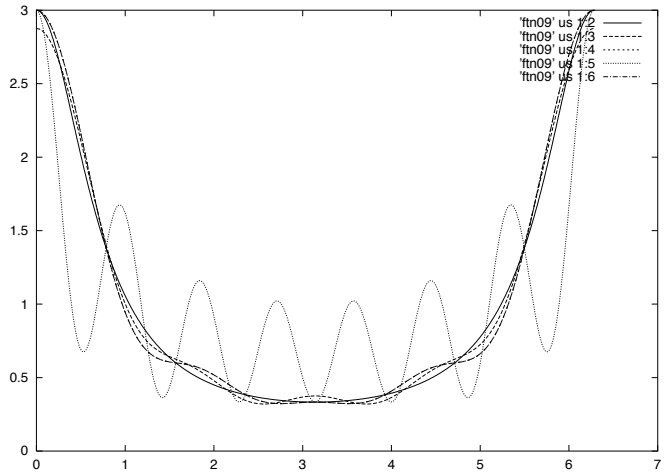

Figure 2: Comparison between the function $f(x) = 3/(5 - 4\cos x)$, solid line 2, and its differents Fourier approximants in the case $N = 8$. Line 3 is for (21); line 4 is for (16); line 5 is for (22) and line 6 is for (18).

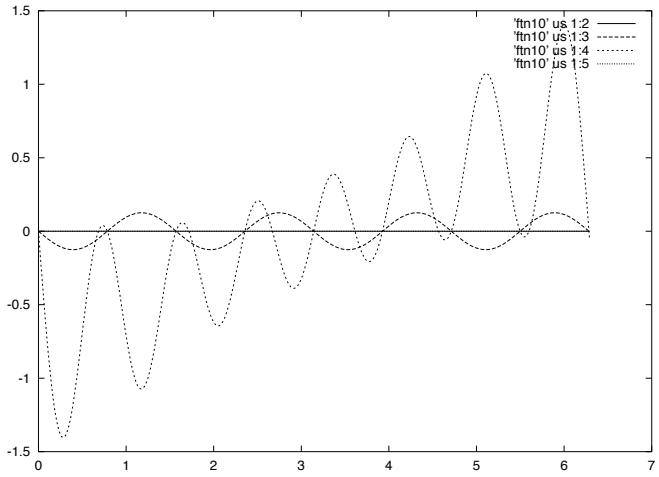

Figure 3: Imaginary part of the Fourier approximants to the $f(x) = 3/(5 - 4\cos x)$ in the case $N = 8$. Same line meanings than in figure (2).

2 Algorithms for partial differential equations

The algorithms we propose are valid for the class of equations which contain linear and non linear terms:

$$\frac{\partial u(x,t)}{\partial t} = \mathcal{L}[u] + \mathcal{N}[u] \tag{23}$$

Although the results are quite general, we will consider the specific case of the 1-d Nikolaevskii equation:

$$\frac{\partial u(x,t)}{\partial t} = -\frac{\partial^2}{\partial x^2}\left[\epsilon - \left(1 + \frac{\partial^2}{\partial x^2}\right)^2\right] u - \left(\frac{\partial u}{\partial x}\right)^2 \tag{24}$$

or, written in terms of the variable

$$v(x,t) = \frac{\partial u(x,t)}{\partial x}, \tag{25}$$

the equation becomes:

$$\frac{\partial v(x,t)}{\partial t} = -\frac{\partial^2}{\partial x^2}\left[\epsilon - \left(1 + \frac{\partial^2}{\partial x^2}\right)^2\right] v - \frac{\partial v^2}{\partial x} \tag{26}$$

We impose that the field $u(x,t)$ satisfies periodic boundary conditions in the interval $x \in [0,L]^d$ and we assume that the Fourier transform of $u(x,t)$, as defined in the previous section, does exist.

$$\tilde{u}(q,t) \equiv \mathcal{F}_P[u(x,t)] = \frac{1}{L^d} \int_{[0,L]^d} dx\, u(x,t) \mathrm{e}^{iqx} \tag{27}$$

The inverse transform is given by:

$$u(x,t) \equiv \mathcal{F}_P^{-1}[\tilde{u}(q,t)] = \sum_{k=-\infty}^{\infty} \tilde{u}_k(t) \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{28}$$

with the notation $\tilde{u}_k(t) = \tilde{u}(q_k,t)$, $q_k = 2\pi k/L$. We apply now this Fourier operator $\mathcal{F}_P$ to the differential equation. Under this transformation the linear term of the equation usually assumes a non-differential form. For example, integration per parts yields:

$$\mathcal{F}_P \left[\frac{\partial f(x)}{\partial x}\right]\Bigg|_{q=q_k} = -iq_k \tilde{f}_k \tag{29}$$

and therefore:

$$\frac{\partial f(x)}{\partial x} = -\mathcal{F}_P^{-1}[iq\tilde{f}(q)] \tag{30}$$

Proceeding this way, we obtain that the linear operator of the Nikolaevskii equation in Fourier space as:

$$\mathcal{F}_P \left[-\frac{\partial^2}{\partial x^2}\left[\epsilon - \left(1+\frac{\partial^2}{\partial x^2}\right)^2\right] u(x,t)\right]\Bigg|_{q=q_k} = \omega_k \tilde{u}_k(t) \tag{31}$$

where $\omega_k = \omega(q_k)$ and

$$\omega(q) \equiv q^2(\epsilon - (1-q^2)^2) \tag{32}$$

The non-linear terms might be transformed in a slightly more complicated form. For instance, for the non-linear term of the first version of the Nikolaevskii equation, we use:

$$\left(\frac{\partial f(x)}{\partial x}\right)^2 = \left(\mathcal{F}_P^{-1}[iq\tilde{f}(q)]\right)^2 \tag{33}$$

Applying the Fourier transform to the equation (24) we obtain:

$$\frac{\partial \tilde{u}_k(t)}{\partial t} = \omega_k \tilde{u}_k(t) + \tilde{a}_k(t) \tag{34}$$

where the non-linear term is:

$$\tilde{a}_k(t) = -\mathcal{F}_P \left[\left(\mathcal{F}_P^{-1}[iq\tilde{u}(q,t)] \right)^2 \right] \Big|_{q=q_k} \tag{35}$$

For reasons that will be clear later, this expression is better that the mathematically equivalent one:

$$\tilde{a}_k(t) = \mathcal{F}_P \left[\left(\mathcal{F}_P^{-1}[q\tilde{u}(q,t)] \right)^2 \right] \Big|_{q=q_k} \tag{36}$$

We approximate the coefficients $\tilde{f}_k$ of the Fourier series by the corresponding ones $\hat{f}_k$ of the discrete Fourier series using Eq.(5). The result is a set of N coupled (complex) equations for the discrete Fourier variables:

$$\frac{d\hat{u}_k(t)}{dt} = \omega_k \hat{u}_k(t) + \hat{a}_k(t), \qquad k = -\frac{N}{2} + 1, \frac{N}{2} \tag{37}$$

where

$$\hat{u}_k(t) = \mathcal{F}_D[u(x,t)] \tag{38}$$

$$\hat{a}_k(t) = \mathcal{F}_D[\mathcal{N}[u(x,t)]] \tag{39}$$

The equations for other values of k are not needed since, due to aliasing, they simply reproduce this basic set of equations. The pseudospectral methods we will develop, use (37) as the basic set of equations to integrate. The integration is done in the discrete Fourier space. Whenever the value of the field in real space $u(x,t)$ is needed we have to use Eq.(18). For a given equation, ω_k and $\hat{a}_k(t)$ will have different expressions, but the algorithms will apply independently of their specific form. A feature of the pseudospectral methods is that, usually, the nonlinear term is computed much more easily in the real space. If this is the case, we use Eq.(18). Let us give some examples of the expression of $\hat{a}_k(t)$ in terms of the Fourier coefficient $\hat{u}_k(t)$.

- A nonlinear term of the form: $\mathcal{N}[u] = u(x,t)^2$. In this case it is irrelevant whether we use Eq.(18) or Eq.(22) since both give the same values for f_n. The $a_k(t)$ will be computed as:

$$a_k = \mathcal{F}_D[(\mathcal{F}_D^{-1}[\hat{u}_k(t)])^2] \tag{40}$$

- First version of the Nikolaevskii equation (24): In this case, we have to be very careful because Eq.(18) and Eq.(22) give different results after differentiation. We use, of course, Eq.(18) which yields:

$$\frac{\partial f}{\partial x} \approx \sum_{k=-M}^{M\,\prime} \left(-i\frac{2\pi}{L}k\right) \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{41}$$

Since $\hat{f}_M = \hat{f}_{-M}$ the first and last terms (the halved ones) cancel and we can write:

$$\frac{\partial f}{\partial x} \approx \sum_{k=-M+1}^{M-1} \left(-i\frac{2\pi}{L}k\right) \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{42}$$

or

$$\frac{\partial f}{\partial x} \approx \sum_{k=0}^{N-1} (-i\bar{q}_k) \hat{f}_k \mathrm{e}^{-i\frac{2\pi}{L}kx} \tag{43}$$

with the convention that:

$$\bar{q}_k = \frac{2\pi}{L}k \qquad k = 0, \frac{N}{2} - 1 \tag{44}$$

$$\bar{q}_{\frac{N}{2}} = 0 \tag{45}$$

$$\bar{q}_k = \frac{2\pi}{L}(k-N) \qquad k = \frac{N}{2}+1, \ldots, N-1 \tag{46}$$

$$\tag{47}$$

The final recipe to compute the nonlinear term for Eq.(24) is:

$$\hat{a}_k(t) = -\mathcal{F}_D \left[\left(\mathcal{F}_D^{-1}[i\bar{q}_k \hat{u}_k(t)] \right)^2 \right] \tag{48}$$

- Second version of the Nikolaevskii equation (26): The final expression is:

$$\hat{a}_k(t) = i\bar{q}_k \mathcal{F}_D \left[\left(\mathcal{F}_D^{-1}[\hat{u}_k(t)] \right)^2 \right] \tag{49}$$

We are ready now to implement a numerical algorith. Before, some important points are in order:

1. For the case of real fields $u(x,t)$ it is possible to store its discrete Fourier transform using only N real variables. This is because of the symmetry relation $\hat{f}_k^* = \hat{f}_{-k}$. Different routines use different storage

rules and we are using here the ones used in the Numerical Recipes book. Namely, we need to store only the following complex numbers $\hat{f}_0, \hat{f}_1, \ldots, \hat{f}_{\frac{N}{2}-1}, \hat{f}_{\frac{N}{2}}$. Moreover, $\hat{f}_0$ and $\hat{f}_{\frac{N}{2}}$ are real numbers. Therefore, we store the Fourier transform in a real vector with N components. For instance, for $N = 8$, the real field is stored in an 8-component real vector as $(f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7)$. The Fourier transform $\hat{f}_k = g_k + ih_k$ is stored in an 8-component real vector as $(g_0, g_4, g_1, h_1, g_2, h_2, g_3, h_3)$. Using this notation, the (real) amplifying function which satisfies $\omega(q) = \omega(-q)$ is stored as $(\omega_0, \omega_4, \omega_1, \omega_1, \omega_2, \omega_2, \omega_3, \omega_3)$. The wavevectors $\bar{q}_k$ are stored as $(0, 0, q_1, q_1, q_2, q_2, q_3, q_3)$. The second zero is due to equation (45) above.

In summary, one only integrates the equations for $\hat{u}_k$ for $k = 0, 1, \ldots, N/2$ and uses that $\hat{u}_0$ and $\hat{u}_{\frac{N}{2}}$ are real numbers. If $\hat{u}_k$ is needed for $k = -\frac{N}{2} + 1, \ldots, -1$ one must use $\hat{u}_{-k} = \hat{u}_k^*$.

One has to make sure, though, that all the direct Fourier transforms are applied to real fields and that all the inverse Fourier transforms are applied to complex fields which are the Fourier transform of a real field. This is why it is better to use expression (35) than (36) because $iq\tilde{u}$ is the Fourier transform of $-\partial_x u$ (a real field) whereas there is no real function whose Fourier transform is $q\tilde{u}$.

An extra bonus of using only half of the Fourier components explicitely in the numerical calculation and, therefore, using the Fourier transform of a real field is that the imaginary part of the field is never computed. This turns out to be important when there is a numerical instability in the imaginary part. In those cases, the imaginary part of the field, that must be zero always, starts growing after some time hence leading to the collapse of the algorithm. This numerical instability is avoided by our algorithm that never computes the imaginary part.

2. It is very convenient to make a change of variables based upon the exact solution of the linear part. The new variable, $z_k(t)$ is defined by:

$$\hat{z}_k(t) = \mathrm{e}^{-\omega_k t}\hat{u}_k(t) \tag{50}$$

The equation for the new variable is:

$$\frac{d\hat{z}}{dt} = \hat{z}_k' = \mathrm{e}^{-\omega_k t}\hat{a}_k(t) \tag{51}$$

It is to this equation that we apply any of the existing algorithms. In particular we have considered Runge-Kutta and predictor corrector methods.

2.1 Predictor-Corrector

A basic integration algorithm (not yet a predictor-corrector) is:

$$\hat{z}_k(t+h) = \hat{z}_k(t) + \frac{h}{2}[3\hat{z}'_k(t) - \hat{z}'_k(t-h)] \tag{52}$$

The evolution equation for the original variable becomes then:

$$\hat{u}_k(t+h) = \mathrm{e}^{\omega_k h}\left[\hat{u}_k(t) + \frac{h}{2}[3\hat{a}_k(t) - \mathrm{e}^{\omega_k h}\hat{a}_k(t-h)]\right] \tag{53}$$

I have programmed this using fortran 90. For the Fourier transform I use the realft routine from Numerical Recipes (with extra factors of 2 and N in order to account for the definitions above). The algorithm is very stable. I have tested for the Nikolaevskii equation using different time steps. For $h = 0.05$ the algorithm does not crash and gives results very close to those using half the time step $h = 0.025$. In the semi-implicit method, the results for $h = 0.05$ and $h = 0.025$ are very different already at initial times. Since this is a chaotic equation, results never agree with different time steps for arbitrarily large integration time.

Using the fast fourier routines of the Silicon Graphics machine, the algorithm performs as follows:

For $N = 8192$, $\delta x = 0.31$, $h = 0.05$, $\epsilon = 0.01$ it takes around 140 min to reach $t = 50,000$. This yields $\epsilon t = 500$. If we were to use $\epsilon = 0.0001$ and $\epsilon t = 1000$ (assuming that we could still use $h = 0.05$) that would take around 20 days in the Silicon Graphics (as compared to around one year we estimated with other algorithms).

We have used in our simulations a 4-th order predictor-corrector method (see any book on numerical methods). The predictor is the Adams–Bashforth four-step method, which for the equation $\dot{y} = f(x, y)$ reads:

$$y_{i+1} = y_i + \frac{h}{24}[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}] \tag{54}$$

The corrector is Adams-Moulton three-step:

$$y_{i+1} = y_i + \frac{h}{24}[9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}] \tag{55}$$

For our equation in the variable z these algorithms read, respectively:

$$\hat{z}_k(t+h) = \hat{z}_k(t) + \frac{h}{24}\left[55\mathrm{e}^{-\omega_k t}\hat{a}_k(t) - 59\mathrm{e}^{-\omega_k(t-h)}\hat{a}_k(t-h) + 37\mathrm{e}^{-\omega_k(t-2h)}\hat{a}_k(t-2h) - 9\mathrm{e}^{-\omega_k(t-3h)}\hat{a}_k(t-3h)\right] \tag{56}$$

and

$$\hat{z}_k(t+h) = \hat{z}_k(t) + \frac{h}{24}\left[9\mathrm{e}^{-\omega_k(t+h)}\hat{a}_k(t+h) + 19\mathrm{e}^{-\omega_k t}\hat{a}_k(t) - 5\mathrm{e}^{-\omega_k(t-h)}\hat{a}_k(t-h) + \mathrm{e}^{-\omega_k(t-2h)}\hat{a}_k(t-2h)\right] \tag{57}$$

Going back to the original variable $\hat{u}$, the integration algorithms read:

$$\hat{u}_k(t+h) = \mathrm{e}^{h\omega_k}\left[\hat{u}_k(t) + \frac{h}{24}\left(55\hat{a}_k(t) - 59\mathrm{e}^{h\omega_k}\hat{a}_k(t-h) + 37\mathrm{e}^{2h\omega_k}\hat{a}_k(t-2h) - 9\mathrm{e}^{3h\omega_k}\hat{a}_k(t-3h)\right)\right] \tag{58}$$

and

$$\hat{u}_k(t+h) = \mathrm{e}^{h\omega_k}\hat{u}_k(t) + \frac{h}{24}\left[9\hat{a}_k(t+h) + 19\mathrm{e}^{h\omega_k}\hat{a}_k(t) - 5\mathrm{e}^{2h\omega_k}\hat{a}_k(t-h) + \mathrm{e}^{3h\omega_k}\hat{a}_k(t-2h)\right] \tag{59}$$

In the program I have used Fortran90 that simplifies enormly the code. The basic integration steps are programmed as follows:

```
call nonlinear(u,u4,uc,u4c,qc,L,coeff)
v=w*(u+dt55*u4-dt59*u3+dt37*u2-dt9*u1)
call nonlinear(v,u1,vc,u1c,qc,L,coeff)
u=dt9*u1+w*(u+dt19*u4-dt5*u3+dt1*u2)
u1=u2*w
u2=u3*w
u3=u4*w
```

In this code we have used the following notation:

$$\begin{aligned}
\texttt{u1} &\rightarrow \mathrm{e}^{3\omega_k h}\hat{a}_k(t-3h)\\
\texttt{u2} &\rightarrow \mathrm{e}^{2\omega_k h}\hat{a}_k(t-2h)\\
\texttt{u3} &\rightarrow \mathrm{e}^{\omega_k h}\hat{a}_k(t-h)\\
\texttt{u4} &\rightarrow \hat{a}_k(t)\\
\texttt{u1} &\rightarrow \hat{a}_k(t+h)\\
\texttt{w} &\rightarrow \mathrm{e}^{\omega_k h}
\end{aligned}$$

which are updated after every time step using trivial relations. `dt55` stands for $55h/24$ and so on. The subroutine `nonlinear(u,a1,...)` returns in `a1` the nonlinear term. In an earlier version, I did not compute again $\hat{a}_k(t+h)$ after the corrector step but rather used the value after the predictor. I believe this is correct since the order of the error involved is beyond the order of the algorithm. However, I have decided to implement the method exactly, although the price one has to pay is to compute the nonlineal function twice per time step. If one wants to compute this function only once, modify the code as follows:

```
v=w*(u+dt55*u4-dt59*u3+dt37*u2-dt9*u1)
call nonlinear(v,a1,uc,a1c,qc,L,coeff)
u=dt9*a1+w*(u+dt19*u4-dt5*u3+dt1*u2)
u1=u2*w
u2=u3*w
u3=u4*w
u4=a1
```

The nonlinear subroutine for the first version of Nikolaevskii equation, see Eq.(48) is:

```
subroutine nonlinear(u,v,uc,vc,qc,L,coeff)
implicit double precision (a-h,o-z)
dimension u(L),v(L)
double complex uc(L/2),vc(L/2),qc(L/2),coeff(L/2+15)
vc=qc*uc
call realft(v,L,-1,coeff)
v=-v*v
call realft(v,L,1,coeff)
return
end
```

here `qc` is a complex vector whose components are $\frac{2\pi i}{N\Delta x}k$ for $k=0,\ldots,\frac{N}{2}-1$ or, in brief, `qc` is nothing but iq. For the second version of Nikolaevskii equation, Eq.(49) the algorithm is:

```
subroutine nonlinear(u,v,uc,vc,qc,L,coeff)
implicit double precision (a-h,o-z)
dimension u(L),v(L)
double complex uc(L/2),vc(L/2),qc(L/2),coeff(L/2+15)
```

```
v=u
call realft(v,L,-1,coeff)
v=v*v
call realft(v,L,1,coeff)
vc=qc*vc
return
end
```

2.2 Runge-Kutta

The previous 4th-order predictor-corrector method requires to be warmed up by generating the 4 first time steps by a method of equivalent accuracy. We use a 4th-order Runge-Kutta method as folows. We write the equation (subindex k and hats are not written out for the sake of clarity) as:

$$\dot{z} = \mathrm{e}^{-\omega t} a = \mathrm{e}^{-\omega t} a(\mathrm{e}^{\omega t} z) \equiv f(z,t) \tag{60}$$

The classic 4-th order Runge-Kutta reads:

$$\begin{aligned} k_1 &= hf(z,t) && (61)\\ k_2 &= hf(z+\frac{k_1}{2}, t+\frac{h}{2}) && (62)\\ k_3 &= hf(z+\frac{k_2}{2}, t+\frac{h}{2}) && (63)\\ k_4 &= hf(z+k_3, t+h) && (64)\\ z(t+h) &= z(t) + \frac{h}{6}k_1 + \frac{h}{3}k_2 + \frac{h}{3}k_3 + \frac{h}{6}k_4 && (65) \end{aligned}$$

In the case that $f(z,t)$ is given by (60), the k_i's are given by:

$$\begin{aligned} k_1 &= h\mathrm{e}^{-\omega t} a(\mathrm{e}^{\omega t} z) && (66)\\ k_2 &= h\mathrm{e}^{-\omega\left(t+\frac{h}{2}\right)} a\left(\mathrm{e}^{\omega\left(t+\frac{h}{2}\right)}\left(z+\frac{k_1}{2}\right)\right) && (67)\\ k_3 &= h\mathrm{e}^{-\omega\left(t+\frac{h}{2}\right)} a\left(\mathrm{e}^{\omega\left(t+\frac{h}{2}\right)}\left(z+\frac{k_2}{2}\right)\right) && (68)\\ k_4 &= h\mathrm{e}^{-\omega(t+h)} a\left(\mathrm{e}^{\omega(t+h)}\left(z+k_3\right)\right) && (69) \end{aligned}$$

The notation can be simplified introducing:

$$u_1 = \mathrm{e}^{\frac{\omega h}{2}}\left(u(t) + \frac{h}{2} a(u(t))\right) \tag{70}$$

$$u_2 = \mathrm{e}^{\frac{\omega h}{2}} \left(u(t) + \frac{h}{2} \mathrm{e}^{-\frac{\omega h}{2}} a(u_1) \right) \tag{71}$$

$$u_3 = \mathrm{e}^{\omega h} \left(u(t) + h \mathrm{e}^{-\frac{\omega h}{2}} a(u_2) \right) \tag{72}$$

And the final algorithm for the u variable is:

$$\hat{u}_k(t+h) = \mathrm{e}^{\omega h} \left[\hat{u}_k(t) + \frac{h}{6} \hat{a}_k(\hat{u}(t)) \right] + \frac{h}{3} \mathrm{e}^{\frac{\omega h}{2}} \hat{a}_k(\hat{u}_1) + \frac{h}{3} \mathrm{e}^{\frac{\omega h}{2}} \hat{a}_k(\hat{u}_2) + \frac{h}{6} \hat{a}_k(\hat{u}_3) \tag{73}$$

3 Program

```
c SPT/upc.f
c Predictor corrector method for Nikolaevskii's equation in the u variable
c
implicit double precision (a-h,o-z)
parameter (L=256)
dimension u(L),u1(L),u2(L),u3(L),u4(L),v(L)
double complex qc(L/2),coeff(L/2+15)
double complex uc(L/2),u1c(L/2),u2c(L/2),u3c(L/2),u4c(L/2),vc(L/2)
dimension w(L),w2(L)
equivalence (uc(1),u(1))
equivalence (vc(1),v(1))
equivalence (u1c(1),u1(1))
equivalence (u2c(1),u2(1))
equivalence (u3c(1),u3(1))
equivalence (u4c(1),u4(1))
open(55,file='upc.in',status='old')
read(55,*) epsilon,dt,nt,nm,nw,u00,dx,iseed

call dran_ini(iseed)
call zfft1di(L/2,coeff)

pi=4.0d0*datan(1.0d0)
dt2=0.5d0*dt
dt3=dt/3.0d0
dt6=dt/6.0d0
dt24=dt/24.0d0
dt55=55.0d0*dt24
dt59=59.0d0*dt24
dt37=37.0d0*dt24
dt9 = 9.0d0*dt24
dt19=19.0d0*dt24
dt5 = 5.0d0*dt24
dt1 = 1.0d0*dt24

do j=1,L/2
        qc(j)=(j-1)*2.0d0*pi/(L*dx)*cmplx(0.0d0,1.0d0)
```

```
enddo

do j=1,L/2
q=(j-1)*2.0d0*pi/(L*dx)
w(2*j-1)=q**2*(epsilon-(1.0d0-q**2)**2)
w(2*j)=q**2*(epsilon-(1.0d0-q**2)**2)
enddo
j=L/2+1
q=(j-1)*2.0d0*pi/(L*dx)
w(2)=q**2*(epsilon-(1.0d0-q**2)**2)

w2=dexp(dt2*w)
w=dexp(dt*w)
do j=1,L
u(j)=u00*dran_u()
enddo
call meanvv(u,L,um)
write(6,*) 'initial',um
write(68,*) 0.0d0,um
call realft(u,L,1,coeff)
write(66,*) nt/nw,L
call nonlinear(u,u1,uc,u1c,qc,L,coeff)
u1=u1*w**3
do ijk=1,3
v=u
call nonlinear(u,u4,uc,u4c,qc,L,coeff)
u=w*(u+dt6*u4)
u4=w2*(v+dt2*u4)
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff)
u=u+w2*dt3*u4
u4=w2*v+dt2*u4
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff)
u=u+w2*dt3*u4
u4=w2*(w2*v+dt*u4)
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff)
u=u+dt6*u4
if (ijk.eq.1) then
call nonlinear(u,u2,uc,u2c,qc,L,coeff)
```

```
u2=u2*w**2
else if (ijk.eq.2) then
call nonlinear(u,u3,uc,u3c,qc,L,coeff)
u3=u3*w
endif
enddo
v=u
call realft(v,L,-1,coeff)
call meanvv(v,L,um)
write(6,*) 3*dt,um
do ijk=4,nt
call nonlinear(u,u4,uc,u4c,qc,L,coeff)
v=w*(u+dt55*u4-dt59*u3+dt37*u2-dt9*u1)
call nonlinear(v,u1,vc,u1c,qc,L,coeff)
u=dt9*u1+w*(u+dt19*u4-dt5*u3+dt1*u2)
u1=u2*w
u2=u3*w
u3=u4*w
if (mod(ijk,nm).eq.0) then
        write(6,*) ijk
v=u
call realft(v,L,-1,coeff)
call meanvv(v,L,um)
write(68,*) ijk*dt,um
write(6,*) ijk*dt,um
endif
if (mod(ijk,nw).eq.0) then
v=u
call realft(v,L,-1,coeff)
do j=1,L
write(66,*) v(j)
enddo
endif
enddo
end

subroutine nonlinear(u,v,uc,vc,qc,L,coeff)
implicit double precision (a-h,o-z)
```

```
dimension u(L),v(L)
double complex uc(L/2),vc(L/2),qc(L/2),coeff(L/2+15)
vc=qc*uc
call realft(v,L,-1,coeff)
v=-v*v
call realft(v,L,1,coeff)
return
end


subroutine meanvv(u,L,um)
implicit double precision (a-h,o-z)
dimension u(L)
um=0.0d0
do j=1,L
um=um+u(j)
enddo
um=um/L
return
end

*************************************************************************
***** Modified from Numerical Recipes to use routines ZFFT1D ********
*************************************************************************

      subroutine realft(data,n,isign,coeff)
      implicit double precision (a-h,o-z)
      dimension  data(n)
      double complex coeff(n/2+15)
      theta=3.141592653589793d0/dble(n/2)
        c1=0.5d0
      if (isign.eq.1) then
        c2=-0.5d0
        call zfft1d(+1,n/2,data,1,coeff)
      else
        c2=0.5d0
        theta=-theta
      endif
```

```
      wpr=-2.0d0*dsin(0.5d0*theta)**2
      wpi=dsin(theta)
      wr=1.0d0+wpr
      wi=wpi
      n2p3=n+3
      do 11 i=2,n/4
        i1=2*i-1
        i2=i1+1
        i3=n2p3-i2
        i4=i3+1
        h1r=c1*(data(i1)+data(i3))
        h1i=c1*(data(i2)-data(i4))
        h2r=-c2*(data(i2)+data(i4))
        h2i=c2*(data(i1)-data(i3))
        data(i1)=h1r+wr*h2r-wi*h2i
        data(i2)=h1i+wr*h2i+wi*h2r
        data(i3)=h1r-wr*h2r+wi*h2i
        data(i4)=-h1i+wr*h2i+wi*h2r
        wtemp=wr
        wr=wr*wpr-wi*wpi+wr
        wi=wi*wpr+wtemp*wpi+wi
11    continue
      if (isign.eq.1) then
h1r=data(1)
data(1)=h1r+data(2)
        data(2)=h1r-data(2)
      else
h1r=data(1)
data(1)=c1*(h1r+data(2))
data(2)=c1*(h1r-data(2))
        call zfft1d(-1,n/2,data,1,coeff)
c I multiply by 2.0d0/N to account for the fact that
c in the inverse tranform realfft you must multiply by 2 (see the
c Numerical Recipes cookbook).
data=2.0d0*data/n
      endif
      return
      end
```

Other modifications implemented so far are the following:

1. - Program GOS/pc4.f integrates the complex Ginzburg-Landau equation in dimension 2 with the GOS spatial derivative terms:

$$\frac{\partial A(\vec{r},t)}{\partial t} = \Box_k^2 A + \epsilon A(1 - 3|A|^2) \tag{74}$$

where the GOS operator is:

$$\Box_k \equiv \vec{k} \cdot \vec{\nabla} - \frac{i}{2k}\nabla^2 \tag{75}$$

Where $\vec{k} = k(\cos\theta, \sin\theta)$ and one takes usually $k = 1$. Since the variables are complex, the program is slightly simpler. One has to be careful with the correct use of the values for the vectors in Fourier space. The amplifying factor for the GOS operator in Fourier space reads:

$$w(\vec{q}) = w(q_x, q_y) = -\left(q_x \cos\theta + q_y \sin\theta - \frac{1}{2}(q_x^2 + q_y^2)\right)^2 \tag{76}$$

The program uses a 4th order predictor corrector initialized by a 4th order Runge-Kutta.

2. - Program BH/d=2/gospc4.f integrates the Busse-Heikes equations with GOS or Laplacian or Directional or NWS derivatives in dimension 2, for the three complex variables A_1, A_2, A_3. The equations are:

$$\frac{\partial A_1(\vec{r},t)}{\partial t} = \mathcal{L}_1 A_1 + A_1(1 - |A_1|^2 - g_1|A_2|^2 - g_2|A_3|^2) \tag{77}$$

$$\frac{\partial A_2(\vec{r},t)}{\partial t} = \mathcal{L}_2 A_2 + A_2(1 - |A_2|^2 - g_1|A_3|^2 - g_2|A_1|^2) \tag{78}$$

$$\frac{\partial A_3(\vec{r},t)}{\partial t} = \mathcal{L}_3 A_3 + A_3(1 - |A_3|^2 - g_1|A_1|^2 - g_2|A_2|^2) \tag{79}$$

where $\mathcal{L}_i$ can be any of the spatial derivative terms. The GOS terms (in Fourier space) are:

$$w_1(\vec{q}) = -\left(q_x \cos\theta_1 + q_y \sin\theta_1 - \frac{1}{2}(q_x^2 + q_y^2)\right)^2 \tag{80}$$

$$w_2(\vec{q}) = -\left(q_x \cos\theta_2 + q_y \sin\theta_2 - \frac{1}{2}(q_x^2 + q_y^2)\right)^2 \tag{81}$$

$$w_3(\vec{q}) = -\left(q_x \cos\theta_3 + q_y \sin\theta_3 - \frac{1}{2}(q_x^2 + q_y^2)\right)^2 \tag{82}$$

where $\vec{k}_i = k(\cos\theta_i, \sin\theta_i)$ and $\theta_3 = \theta_2 + \frac{\pi}{3} = \theta_1 + \frac{2\pi}{3}$.

The NWS terms replace the Laplacian in the above expression by a directional second derivative.

In this program, the first 3 integration steps are done (incorrectly) using the Euler method instead of the 4-th order predictor-corrector. Program BH/d=2/gospc4_equ.f uses a simpler code by using som equivalence statements. Otherwise is completeley equivalent (and gives the same results) to BH/d=2/gospc4.f.

3. - Program BH/d=2/gospc5.f does the same integration by using a 5th order predictor corrector and an Euler initialization. Finally, program BH/d=2/gospc5_equ.f uses a 5th order predictor corrector and a 4th order Runge Kutta for initialization.

4 Stochastic algorithms

4.1 A single variable

Let us start by a simple example: a single variable, $u(t)$, equation with an additive noise term:

$$\dot{u} = \omega u + a(u) + \xi(t) \tag{83}$$

where $\xi(t)$ is white noise of zero mean and correlations:

$$\langle \xi(t)\xi(t') \rangle = 2\epsilon\delta(t-t') \tag{84}$$

The solution of the linear equation ($a(u) = 0$):

$$u(t) = \mathrm{e}^{\omega t}u(0) + \mathrm{e}^{\omega t}\int_0^t ds \mathrm{e}^{-\omega s}\xi(s) \tag{85}$$

suggests the change of variables:

$$u(t) = \mathrm{e}^{\omega t}z(t) + \mathrm{e}^{\omega t}\int_0^t ds \mathrm{e}^{-\omega s}\xi(s) \equiv \mathrm{e}^{\omega t}z(t) + G(t) \tag{86}$$

The variable $z(t)$ satisfies:

$$\dot{z} = \mathrm{e}^{-\omega t}a(t) \tag{87}$$

the u dependance of $a(u)$ translates into a time dependance $a(t) \equiv a(u(t))$. This differential equation can be solved using any existing method. As a way of example, consider the explicit method:

$$z(t+h) = z(t) + \frac{h}{2}\left[3\dot{z}(t) - \dot{z}(t-h)\right] = z(t) + \frac{h}{2}\left[3\mathrm{e}^{-\omega t}a(t) - \mathrm{e}^{-\omega(t-h)}a(t-h)\right] \tag{88}$$

by replacing $z(t) = \mathrm{e}^{-\omega t}\left(u(t) - G(t)\right)$ we arrive at:

$$\begin{aligned} u(t+h) &= G(t+h) + \mathrm{e}^{\omega h}\left[u(t) - G(t) + \frac{h}{2}\left(3a(t) - \mathrm{e}^{\omega t}a(t-h)\right)\right] \\ &\equiv g(t) + \mathrm{e}^{\omega h}\left[u(t) + \frac{h}{2}\left(3a(t) - \mathrm{e}^{\omega t}a(t-h)\right)\right] \end{aligned} \tag{89}$$

where we have introduced the Gaussian process:

$$g(t) = G(t+h) - \mathrm{e}^{\omega h}G(t) = \mathrm{e}^{\omega(t+h)}\int_t^{t+h} ds\ \mathrm{e}^{-\omega s}\xi(s) \tag{90}$$

This can be computed by noticing that for $t_i = ih$ it is:

$$\langle g(t_i)g(t_j)\rangle = \frac{\epsilon}{\omega}\left(\mathrm{e}^{2\omega h}-1\right)\delta_{i,j} \tag{91}$$

The integration code is the following:

```
a1=a(u)
u=eqs*dran_g()+ewh*(u+h2*(3.0d0*a1-a2))
a2=ewh*a1
```

Where we use the notation:

$$\begin{aligned} \texttt{a1} &\rightarrow a(t) \\ \texttt{a2} &\rightarrow \mathrm{e}^{\omega h}a(t-h) \end{aligned} \tag{92}$$

and the definitions:

$$\begin{aligned} \texttt{ewh} &\rightarrow \mathrm{e}^{\omega h} \\ \texttt{h2} &\rightarrow h/2 \\ \texttt{eqs} &\rightarrow \sqrt{\frac{\epsilon}{\omega}\left(\mathrm{e}^{2\omega h}-1\right)} \end{aligned} \tag{93}$$

A fourth-order predictor corrector is very simple now. We just give the algorithm:

$$\begin{aligned} &z(t+h) = z(t) + \\ &\frac{h}{24}\left[55\mathrm{e}^{-\omega t}a(t) - 59\mathrm{e}^{-\omega(t-h)}a(t-h) + 37\mathrm{e}^{-\omega(t-2h)}a(t-2h) - 9\mathrm{e}^{-\omega(t-3h)}a(t-3h)\right] \end{aligned} \tag{94}$$

and

$$\begin{aligned} &z(t+h) = z(t) + \\ &\frac{h}{24}\left[9\mathrm{e}^{-\omega(t+h)}a(t+h) + 19\mathrm{e}^{-\omega t}a(t) - 5\mathrm{e}^{-\omega(t-h)}a(t-h) + \mathrm{e}^{-\omega(t-2h)}a(t-2h)\right] \end{aligned} \tag{95}$$

For the original variable , the algorithms read:

$$\begin{aligned} &u(t+h) = g(t) + \mathrm{e}^{h\omega}\left[u(t) + \right. \\ &\left.\frac{h}{24}\left(55a(t) - 59\mathrm{e}^{h\omega}a(t-h) + 37\mathrm{e}^{2h\omega}a(t-2h) - 9\mathrm{e}^{3h\omega}a(t-3h)\right)\right] \end{aligned} \tag{96}$$

and

$$u(t+h) = g(t) + \mathrm{e}^{h\omega} u(t) + \qquad (97)$$
$$\frac{h}{24}\left[9a(t+h) + 19\mathrm{e}^{h\omega} a(t) - 5\mathrm{e}^{2h\omega} a(t-h) + \mathrm{e}^{3h\omega} a(t-2h)\right]$$

The code is the following:

```
a4=a(u)
g=eqs*dran_g()
u0=g+ewh*(u+h24*(55.d0*a4-59.d0*a3+37.d0*a2-9.d0*a1))
u1=a(u0)
u=g+h9*u1+ewh*(u+h24*(19.d0*a4-5.d0*a3+a2))
a1=ewh*a2
a2=ewh*a3
a3=ewh*a4
```

where, in this case, we have computed the nonlinear term twice (see comment in the first section). Program SDE/pc4.f implements this method for the one-variable stochastic equation:

$$\dot{x} = wx - x^3 + \xi \qquad (98)$$

So far, the method is initialized by using a deterministic 4th order Runge-Kutta method for the first 3 steps. This is not correct, and it can be considered as if the initial condition was a different one, and a 4-th order *stochastic* Runge-Kutta should be used. This is probably not too important since the equation is stochastic and one averages over initial conditions anyway (or at least, check that the results do not depend on initial conditions). The code is the following:

```
C SDE/pc4.f
C Method to integrate SDE integrating out the linear part.
C In this program we apply to the simple case
C  dx/dt = w x +a(x) +xi(t)
C using a fourth order predictor-corrector algorithm

implicit double precision (a-h,o-z)
a(x)=-x**3
open(55,file='pc4.in',status='old')
```

```
open(67,file='pc4.out',status='old')
read(55,*) x0,w,epsilon,h,n,n0,nw,nc,iseed
do while (.true.)
read(67,*,end=999)
enddo
999 continue
call dran_ini(iseed)
ewh=dexp(h*w)
h24=h/24.0d0
h9=9.0d0*h24
nt=0
xm=0.0d0
xm2=0.0d0
eqs=dsqrt(epsilon*(dexp(2.0d0*w*h)-1.0d0)/w)

x=x0
a1=ewh**3*a(x)
x=x+h*(w*x+a(x))+dsqrt(2.0d0*epsilon*h)*dran_g()
a2=ewh**2*a(x)
x=x+h*(w*x+a(x))+dsqrt(2.0d0*epsilon*h)*dran_g()
a3=ewh*a(x)
x=x+h*(w*x+a(x))+dsqrt(2.0d0*epsilon*h)*dran_g()

do i=1,n0
a4=a(x)
g=eqs*dran_g()
x0=g+ewh*(x+h24*(55.d0*a4-59.d0*a3+37.d0*a2-9.d0*a1))
u1=a(x0)
x=g+h9*u1+ewh*(x+h24*(19.d0*a4-5.d0*a3+a2))
a1=ewh*a2
a2=ewh*a3
a3=ewh*a4
if (mod(i,nw).eq.0) write(66,*) i*h,x,u2

enddo

do i=1,n
a4=a(x)
```

```
g=eqs*dran_g()
x0=g+ewh*(x+h24*(55.d0*a4-59.d0*a3+37.d0*a2-9.d0*a1))
u1=a(x0)
x=g+h9*u1+ewh*(x+h24*(19.d0*a4-5.d0*a3+a2))
a1=ewh*a2
a2=ewh*a3
a3=ewh*a4
if (mod(i,nw).eq.0) write(66,*) i*h,x,u2
if (mod(i,nc).eq.0) then
nt=nt+1
xm=xm+dabs(x)
xm2=xm2+x*x
endif
enddo
xm=xm/nt
xm2=xm2/nt-xm*xm
write(67,*) h,xm,dsqrt(xm2/nt)
end
```

We consider now a second-order Runge-Kutta method. This is based upon the classical midpoint Runge-Kutta method.

$$k_1 = \frac{h}{2} f(z,t) \tag{99}$$

$$z(t+h) = z(t) + hf(t+\frac{h}{2}, z(t)+k_1) \tag{100}$$

For our case it can be shown to reduce to:

$$u_1 = \mathrm{e}^{\frac{\omega h}{2}} \left(u(t) + \frac{h}{2} a(t) \right) + g_1(t) \tag{101}$$

$$u(t+h) = g(t) + \mathrm{e}^{\omega h} u(t) + h \mathrm{e}^{\frac{\omega h}{2}} a(u_1)$$

where:

$$g_1(t) = G(t+\frac{h}{2}) - \mathrm{e}^{\frac{\omega h}{2}} G(t) = \mathrm{e}^{\omega(t+\frac{h}{2})} \int_t^{t+\frac{h}{2}} ds \mathrm{e}^{-\omega s} \xi(s) \tag{102}$$

$$g(t) = G(t+h) - \mathrm{e}^{\omega h} G(t) = \mathrm{e}^{\frac{\omega h}{2}} g_1(t) + g_1(t+\frac{h}{2}) \tag{103}$$

Finally, notice that $g_1(t)$ and $g_1(t+\frac{h}{2})$ are independent, zero mean, Gaussian variables and that $\langle g_1(t_i)g_1(t_j)\rangle = \langle g_1(t_i)^2\rangle\delta_{i,j}$ for $t_i = ih$ and

$$\langle g_1(t)^2\rangle = \sqrt{\frac{\epsilon}{\omega}\left(\mathrm{e}^{\omega h}-1\right)} \tag{104}$$

This is a second-order Runge-Kutta. It is less general than the Heun method that applies to the multiplicative equation $\dot{u} = f(u) + g(u)\xi(t)$. It would be interesting to compare both algorithms.

4.2 Stochastic partial differential equations

We now consider a field $u(x,t)$ that satisfies a stochastic partial differential equation:

$$\frac{\partial u(x,t)}{\partial t} = \mathcal{L}[u] + \mathcal{N}[u] + \xi(x,t) \tag{105}$$

with white noise in space and time:

$$\langle \xi(x,t)\xi(x',t')\rangle = 2\epsilon\delta(x-x')\delta(t-t') \tag{106}$$

We consider again periodic boundary conditions in the space $x \in [0,L]^d$. We take the Fourier transform to obtain:

$$\frac{\partial \tilde{u}(q,t)}{\partial t} = \omega(q)\tilde{u}(q,t) + \tilde{a}(q,t) + \tilde{\xi}(q,t) \tag{107}$$

Where the Fourier transform of the noise is defined using the general relation:

$$\tilde{\xi}(q,t) = \frac{1}{L^d}\int_{[0,L]^d} dx\ \mathrm{e}^{iqx}\xi(x,t) \tag{108}$$

notice that, being the Fourier transform of a real field, it satisfies $\tilde{\xi}(q,t) = \tilde{\xi}^*(-q,t)$. It is easy to verify that $\tilde{\xi}(q,t)$ are complex Gaussian variables of zero mean, and correlations given by

$$\langle \tilde{\xi}(q_k,t)\tilde{\xi}(q_{k'},t')\rangle = \frac{2\epsilon}{L^d}\delta_{k+k'}\delta(t-t') \tag{109}$$

(remember that $q_k = 2\pi k/L$). As in the case of non-stochastic partial differential equations we want to introduce now the discrete Fourier transform.

However, one has to be very careful when defining the discrete Fourier transform of the white noise. One can not use directly Eq.(6) putting $f_n = \xi(x_n)$ because $\xi(x_n)$ is a Gaussian variable of infinite variance. There are several ways of getting the proper result. They all use in one way or another the relation between the Dirac-delta and the Kronecker-delta functions. In real space of dimension d, this relation is:

$$\delta(x_n) \to (\Delta x)^{-d}\delta_n \tag{110}$$

(remember that $x_n = n\Delta x$). This relation can be obtained by demanding that the integral of a Dirac-delta function coincides with the sum of the Kronecker-delta:

$$1 = \int_{-\infty}^{\infty} dx\ \delta(x) = (\Delta x)^d \sum_n (\Delta x)^{-d}\delta_n \tag{111}$$

With this relation in mind, we can identify the discrete equivalent of the white noise in space and time: it is a set of white noises in time with zero mean and correlations given by $2\epsilon(\Delta x)^{-1}\delta(t-t')$:

$$\xi(x_n, t) \to \xi_n(t) \tag{112}$$

with

$$\begin{aligned} \langle \xi_n(t) \rangle &= 0 \\ \langle \xi_n(t)\xi_{n'}(t') \rangle &= \frac{2\epsilon}{(\Delta x)^d}\delta_{n,n'}\delta(t-t') \end{aligned} \tag{113}$$

In this way, we can go from continuum to discrete space as usual. Notice that, according to the criterion above, the correlations of the $\xi(x,t)$ and $\xi_n(t)$ are equal. In effect,

$$\langle \xi(x,t)\xi(x',t') \rangle = \langle \xi_n(t)\xi_{n'}(t') \rangle \tag{114}$$

implies:

$$2\epsilon\delta(x-x')\delta(t-t') = \frac{2\epsilon}{(\Delta x)^d}\delta_{n,n'}\delta(t-t') \tag{115}$$

which is nothing but Eq.(110).

After taking the Fourier transform of the original equation (105) and replacing the continuum Fourier transform by the discrete one, we obtain:

$$\frac{\partial \hat{u}_k(t)}{\partial t} = \omega_k \hat{u}_k(t) + \hat{a}_k(t) + \hat{\xi}_k(t) \tag{116}$$

where $\hat{\xi}_k(t)$ is the discrete Fourier transform of $\xi_n(t)$, $\hat{\xi}_k(t) = \mathcal{F}_D[\xi_n(t)]$. The correlations of $\hat{\xi}_k(t)$ can be computed using its definition:

$$\langle \hat{\xi}_k(t)\hat{\xi}_{k'}(t')\rangle = \sum_n \sum_{n'} \mathrm{e}^{\frac{2\pi i}{N}(nk+n'k')} \langle \xi_n(t)\xi_{n'}(t')\rangle =$$

$$\sum_n \sum_{n'} \mathrm{e}^{\frac{2\pi i}{N}(nk+n'k')} \frac{2\epsilon}{\Delta x}\delta_{n,n'}\delta(t-t') = \frac{2\epsilon N^d}{(\Delta x)^d}\delta(t-t')\delta_{k+k'} \quad (117)$$

where we have used Eq.(11). We check now that this result corresponds to the continuum one, Eq.(109), if we make the usual correspondence

$$\tilde{\xi}(q_k, t) \to \frac{1}{N^d}\hat{\xi}_k(t) \quad (118)$$

Indeed:

$$\langle \tilde{\xi}(q_k,t)\tilde{\xi}(q_{k'},t')\rangle = \langle [N^{-d}\hat{\xi}_k(t)][N^{-d}\hat{\xi}_{k'}(t')]\rangle = \frac{2\epsilon}{(N\Delta x)^d}\delta_{k+k'}\delta(t-t') \quad (119)$$

which is the expected result, see Eq.(109).

Before using any numerical algorithm to solve the set of equations (116), we make a change of variables based upon the solution of the linear equation.

$$\hat{u}_k(t) = \mathrm{e}^{\omega_k t}\hat{z}_k(t) + \mathrm{e}^{\omega_k t}\int_0^t ds\ \mathrm{e}^{-\omega_k s}\hat{\xi}_k(s) \equiv \mathrm{e}^{\omega_k t}\hat{z}_k(t) + \hat{G}_k(t) \quad (120)$$

with the definition:

$$\hat{G}_k(t) = \mathrm{e}^{\omega_k t}\int_0^t ds\ \mathrm{e}^{-\omega_k s}\hat{\xi}_k(s) \quad (121)$$

$\hat{z}_k(t)$ satisfies the equation:

$$\frac{\partial \hat{z}_k(t)}{\partial t} = \hat{a}_k(t) \quad (122)$$

to which we apply any of the predictor-corrector or Runge-Kutta algorithms explained before. For instance, the one given by Eq.(52), which now is:

$$\hat{z}_k(t+h) = \hat{z}_k(t) + \frac{h}{2}\left[3\mathrm{e}^{-\omega_k t}\hat{a}_k(t) - \mathrm{e}^{-\omega_k(t-h)}\hat{a}_k(t-h)\right] \quad (123)$$

Going back to the variable $\hat{u}_k(t)$ using Eq.(120), we obtain:

$$\hat{u}_k(t+h) = \hat{g}_k(t) + \mathrm{e}^{\omega_k h}\left[\hat{u}_k(t) + \frac{h}{2}[3\hat{a}_k(t) - \mathrm{e}^{\omega_k h}\hat{a}_k(t-h)]\right] \tag{124}$$

where

$$\hat{g}_k(t) \equiv \hat{G}_k(t+h) - \mathrm{e}^{\omega_k h}\hat{G}_k(t) = \mathrm{e}^{\omega_k(t+h)}\int_t^{t+h} ds\ \mathrm{e}^{-\omega_k s}\hat{\xi}_k(s) \tag{125}$$

As in the one-variable case, we can show that $\langle \hat{g}_k(t_i)\hat{g}_{k'}(t_j)\rangle = 0$ for $i \neq j$. For equal times the correlation is:

$$\langle \hat{g}_k(t)\hat{g}_{k'}(t)\rangle = \frac{\mathrm{e}^{2\omega_k h}-1}{\omega_k}\frac{N\epsilon}{\Delta x}\delta_{k+k'} \tag{126}$$

(we have assumed that $\omega_k = \omega_{-k}$). In practise, the variables $\hat{g}_k(t)$ can be obtained by writing them as:

$$\hat{g}_k(t) = \sqrt{\frac{\mathrm{e}^{2\omega_k h}-1}{\omega_k}\frac{\epsilon}{\Delta x}}\hat{v}_k(t) \tag{127}$$

where

$$\hat{v}_k(t) = \mathcal{F}_D[v_n(t)] \tag{128}$$

and $v_n(t)$ is a set of independent Gaussian random numbers of zero mean and variance one.

The fourth-order predictor corrector is now trivial to obtain. We write it here just for reference:

$$\begin{aligned}&\hat{u}_k(t+h) = \hat{g}_k(t) + \mathrm{e}^{h\omega_k}\left[\hat{u}_k(t) + \right. \\ &\left.\frac{h}{24}\left(55\hat{a}_k(t) - 59\mathrm{e}^{h\omega_k}\hat{a}_k(t-h) + 37\mathrm{e}^{2h\omega_k}\hat{a}_k(t-2h) - 9\mathrm{e}^{3h\omega_k}\hat{a}_k(t-3h)\right)\right]\end{aligned} \tag{129}$$

and

$$\begin{aligned}&\hat{u}_k(t+h) = \hat{g}_k(t) + \mathrm{e}^{h\omega_k}\hat{u}_k(t) + \\ &\frac{h}{24}\left[9\hat{a}_k(t+h) + 19\mathrm{e}^{h\omega_k}\hat{a}_k(t) - 5\mathrm{e}^{2h\omega_k}\hat{a}_k(t-h) + \mathrm{e}^{3h\omega_k}\hat{a}_k(t-2h)\right]\end{aligned} \tag{130}$$

This is easy to implement, simply add in the deterministic program the stochastic terms. The main evolution routine is

```
call dran_gv(xi,L)
call realft(xi,L,1,coeff)
xi=w4*xi
call nonlinear(u,u4,uc,u4c,qc,L,coeff,al)
v=xi+w*(u+dt55*u4-dt59*u3+dt37*u2-dt9*u1)
call nonlinear(v,a1,vc,a1c,qc,L,coeff,al)
u=xi+dt9*a1+w*(u+dt19*u4-dt5*u3+dt1*u2)
u1=u2*w
u2=u3*w
u3=u4*w
```

Here we have defined

$$\texttt{w4} \to \sqrt{\frac{e^{2\omega_k h}-1}{\omega_k}\frac{\epsilon}{\Delta x}} \tag{131}$$

Although the vector $\hat{\xi}_k$ could be generated directly in Fourier space (see Ojalvo and Sancho, appendix B), we have generated here as the discrete Fourier transform of a real random vector. Program SDE/spc4.f has implemented this algorithm for the stochastic one-dimensional KPZ equation:

$$\frac{\partial h(\vec{r},t)}{\partial t} = \nabla^2 h + \lambda|\vec{\nabla} h|^2 + \xi(\vec{r},t) \tag{132}$$

Program SDE/spc5.f has implemented this algorithm for the stochastic one-dimensional Lai-Das Sarma-Villain equation:

$$\frac{\partial h(\vec{r},t)}{\partial t} = -\nabla^4 h + \lambda\nabla^2|\vec{\nabla} h|^2 + \xi(\vec{r},t) \tag{133}$$

Again, the method is initialized by using an Euler method for the first 3 steps. This is not correct, and it can be considered as if the initial condition was a different one, and a 4-th order stochastic Runge-Kutta should be used. This is probably not too important since the equation is stochastic and one averages over initial conditions anyway (or at least, check that the results do not depend on initial conditions). The code is the following:

```
c SDE/spc4.f
c Predictor corrector method for Stochastic's KPZ equation
c
implicit double precision (a-h,o-z)
```

```
parameter (L=256)
dimension u(L),u1(L),u2(L),u3(L),u4(L),v(L)
dimension xi(L)
double complex qc(L/2),coeff(L/2+15)
double complex uc(L/2)
double complex u1c(L/2),u2c(L/2),u3c(L/2),u4c(L/2),vc(L/2)
dimension w(L),w2(L),w4(L)
equivalence (uc(1),u(1))
equivalence (vc(1),v(1))
equivalence (u1c(1),u1(1))
equivalence (u2c(1),u2(1))
equivalence (u3c(1),u3(1))
equivalence (u4c(1),u4(1))
open(55,file='spc4.in',status='old')
read(55,*) epsilon,al,dt,nt,nm,nw,u00,dx,iseed

call dran_ini(iseed)
call zfft1di(L/2,coeff)

pi=4.0d0*datan(1.0d0)
dt2=0.5d0*dt
dt3=dt/3.0d0
dt6=dt/6.0d0
dt24=dt/24.0d0
dt55=55.0d0*dt24
dt59=59.0d0*dt24
dt37=37.0d0*dt24
dt9 = 9.0d0*dt24
dt19=19.0d0*dt24
dt5 = 5.0d0*dt24
dt1 = 1.0d0*dt24

do j=1,L/2
        qc(j)=(j-1)*2.0d0*pi/(L*dx)*cmplx(0.0d0,1.0d0)
enddo

do j=1,L/2
q=(j-1)*2.0d0*pi/(L*dx)
```

```
w(2*j-1)=-q**2
w(2*j)=-q**2
enddo
j=L/2+1
q=(j-1)*2.0d0*pi/(L*dx)
w(2)=-q**2

w4(1)=dsqrt(2.0d0*dt*epsilon/dx)
do i=2,L
w4(i)=dsqrt((dexp(2.0d0*dt*w(i))-1.0d0)*epsilon/(w(i)* dx))
enddo
w2=dexp(dt2*w)
w=dexp(dt*w)
do j=1,L
u(j)=u00*dran_u()
enddo
call meanvv(u,L,um,um2)
write(6,*) 'initial',um,um2
write(68,*) 0.0d0,um,um2
call realft(u,L,1,coeff)
write(66,*) nt/nw,L
call nonlinear(u,u1,uc,u1c,qc,L,coeff,al)
u1=u1*w**3
do ijk=1,3
v=u
call nonlinear(u,u4,uc,u4c,qc,L,coeff,al)
u=w*(u+dt6*u4)
u4=w2*(v+dt2*u4)
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff,al)
u=u+w2*dt3*u4
u4=w2*v+dt2*u4
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff,al)
u=u+w2*dt3*u4
u4=w2*(w2*v+dt*u4)
call nonlinear(u4,u4,u4c,u4c,qc,L,coeff,al)
u=u+dt6*u4
if (ijk.eq.1) then
call nonlinear(u,u2,uc,u2c,qc,L,coeff,al)
```

```
u2=u2*w**2
else if (ijk.eq.2) then
call nonlinear(u,u3,uc,u3c,qc,L,coeff,al)
u3=u3*w
endif
enddo
v=u
call realft(v,L,-1,coeff)
call meanvv(v,L,um,um2)
write(6,*) 3*dt,um
do ijk=4,nt
call dran_gv(xi,L)
call realft(xi,L,1,coeff)
xi=w4*xi
call nonlinear(u,u4,uc,u4c,qc,L,coeff,al)
v=xi+w*(u+dt55*u4-dt59*u3+dt37*u2-dt9*u1)
call nonlinear(v,u1,vc,u1c,qc,L,coeff,al)
u=xi+dt9*u1+w*(u+dt19*u4-dt5*u3+dt1*u2)
u1=u2*w
u2=u3*w
u3=u4*w
if (mod(ijk,nm).eq.0) then
        write(6,*) ijk
v=u
call realft(v,L,-1,coeff)
call meanvv(v,L,um,um2)
write(68,*) ijk*dt,um,um2
write(6,*) ijk*dt,um,um2
endif
if (mod(ijk,nw).eq.0) then
v=u
call realft(v,L,-1,coeff)
do j=1,L
write(66,*) v(j)
enddo
endif
enddo
end
```

```
subroutine nonlinear(u,v,uc,vc,qc,L,coeff,al)
implicit double precision (a-h,o-z)
dimension u(L),v(L)
double complex uc(L/2),vc(L/2),qc(L/2),coeff(L/2+15)
vc=qc*uc
call realft(v,L,-1,coeff)
v=al*v*v
call realft(v,L,1,coeff)
return
end


subroutine meanvv(u,L,um,um2)
implicit double precision (a-h,o-z)
dimension u(L)
um=0.0d0
um2=0.0d0
do j=1,L
um=um+u(j)
um2=um2+u(j)*u(j)
enddo
um=um/L
um2=um2/L-um*um
return
end
```

5 Errors in the pseudospectral methods

It is important to analyse which are the sources of errors of the pseudospectral methods. Even when the time-integrator (Runge-Kutta, predictor-corrector or whatever) is stable, there is always the question of whether the solution of the pseudospectral method should converge towards the (exact) solution of the problem.

The most important problem is aliasing which has been discussed thoroughly in previous sections.

The second largest source of error (also related to aliasing) is that even if the approximant at $t = 0$ is a very good one, it might get bad (or very bad!) as time goes on. This problem can be very well understood since it already appears in linear equations id $d = 1$:

$$\frac{\partial u(x,t)}{\partial t} = \mathcal{L}[u] \tag{134}$$

with initial condition $u(x,0) = u^0(x)$ and periodic boundary conditions in the interval $(0, L)$. This equation, in Fourier space, adopts the form:

$$\frac{\partial \tilde{u}(q,t)}{\partial t} = \omega(q)\tilde{u} \tag{135}$$

whose solution is:

$$\tilde{u}(q,t) = \tilde{u}^0(q)\mathrm{e}^{\omega(q)t} \tag{136}$$

The solution in real space is obtained by using the inverse transform in the series form:

$$u(x,t) = \sum_{k=-\infty}^{\infty} \tilde{u}_k^0 \mathrm{e}^{-iq_k x}\mathrm{e}^{\omega_k t} \tag{137}$$

(recall the notation $F_k = F(q_k)$ for any function $F(q)$). $\tilde{u}_k^0$ is obtained as the Fourier transform of the (periodic) function $u^0(x)$:

$$\tilde{u}_k^0 = \frac{1}{L}\int_{[0,L]} u^0(x)\mathrm{e}^{iq_k x}dx \tag{138}$$

This solution can be put in a more compact form using a propagator:

$$u(x,t) = \int_{[0,L]} dy\ u^0(y)G(x-y,t) \tag{139}$$

where:

$$G(z,t) = \frac{1}{L} \sum_{k=-\infty}^{\infty} \mathrm{e}^{-iq_k z + \omega_k t} \tag{140}$$

Using

$$\sum_{k=-\infty}^{\infty} \mathrm{e}^{ika} = 2\pi\delta(a) \tag{141}$$

it is easy to prove that

$$G(z, t=0) = \delta(z) \tag{142}$$

So far, everything is exact. The approximation of replacing the sum in (137) by a finite sum is the method of Galerkin:

$$u(x,t) = \sum_{k=-N/2}^{N/2} \tilde{u}_k^0 \mathrm{e}^{-iq_k x} \mathrm{e}^{\omega_k t} \tag{143}$$

However, this method requires the knowledge of the exact Fourier coefficients $\tilde{u}_k^0$ (or a good numerical evaluation of them).

In the pseudospectral methods one replaces the above expressions by the corresponding ones using the discrete Fourier transform:

$$u(x,t) = \frac{1}{N} \sum_{k=-N/2}^{N/2\ \prime} \hat{u}_k^0 \mathrm{e}^{-iq_k x} \mathrm{e}^{\omega_k t} \tag{144}$$

(remember that the prime indicates halving of the first and last terms in the sum). The $\hat{u}_k^0$ are obtained as the discrete Fourier transform of the initial condition $\hat{u}_k^0 = \mathcal{F}_D[u_n^0]$.

We ask now which is the difference between the exact solution (137) and the one given by (144). The answer can be obtained by recalling relation (7), such that the numerical solution can be written as:

$$u(x,t) = \sum_{k=-N/2}^{N/2\ \prime} \sum_{m=-\infty}^{\infty} \tilde{u}_{k+mN}^0 \mathrm{e}^{-iq_k x} \mathrm{e}^{\omega_k t} \tag{145}$$

Expanding the sum we obtain:

$$\frac{1}{2}\mathrm{e}^{\omega_{-N/2} t} \quad [\ \ \ldots + \tilde{u}_{-N/2-N}^0 \mathrm{e}^{-iq_{-N/2} x} + \tilde{u}_{-N/2}^0 \mathrm{e}^{-iq_{-N/2} x} + \tilde{u}_{-N/2+N}^0 \mathrm{e}^{-iq_{-N/2} x} + \ldots] +$$

$$
\begin{aligned}
\mathrm{e}^{\omega_{-N/2+1}t}\ [\ & \ldots+\tilde{u}^0_{-N/2+1-N}\mathrm{e}^{-iq_{-N/2+1}x}+\tilde{u}^0_{-N/2+1}\mathrm{e}^{-iq_{-N/2+1}x}+\tilde{u}^0_{-N/2+1+N}\mathrm{e}^{-iq_{-N/2+1}x}+\ldots]+\\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
\mathrm{e}^{\omega_0 t}\ [\ & \ldots+\tilde{u}^0_{-N}\mathrm{e}^{-iq_0x}+\tilde{u}^0_0\mathrm{e}^{-iq_0x}+\tilde{u}^0_N\mathrm{e}^{-iq_0x}+\ldots]+\\
\mathrm{e}^{\omega_1 t}\ [\ & \ldots+\tilde{u}^0_{1-N}\mathrm{e}^{-iq_1x}+\tilde{+}u^0_1\mathrm{e}^{-iq_1x}+\tilde{+}u^0_{1+N}\mathrm{e}^{-iq_1x}+\ldots]+\\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
\mathrm{e}^{\omega_{N/2}t}\ [\ & \ldots+\tilde{u}^0_{N/2+1-N}\mathrm{e}^{-iq_{N/2+1}x}+\tilde{u}^0_{N/2+1}\mathrm{e}^{-iq_{N/2+1}x}+\tilde{u}^0_{N/2+1+N}\mathrm{e}^{-iq_{N/2+1}x}+\ldots]+\\
\frac{1}{2}\mathrm{e}^{\omega_{N/2}t}\ [\ & \ldots+\tilde{u}^0_{N/2-N}\mathrm{e}^{-iq_{N/2}x}+\tilde{u}^0_{N/2}\mathrm{e}^{-iq_{N/2}x}+\tilde{u}^0_{N/2+N}\mathrm{e}^{-iq_{N/2}x}+\ldots]
\end{aligned}
\tag{146}
$$

Taking into account that for the lattice points it is:

$$
\mathrm{e}^{iq_kx_n}=\mathrm{e}^{iq_{k+mN}x_n}\qquad\forall m \tag{147}
$$

we can rewrite the previous formula as:

$$
\begin{aligned}
\frac{1}{2}\mathrm{e}^{\omega_{-N/2}t}\ [\ & \ldots+\tilde{u}^0_{-N/2-N}\mathrm{e}^{-iq_{-N/2-N}x}+\tilde{u}^0_{-N/2}\mathrm{e}^{-iq_{-N/2}x}+\tilde{u}^0_{-N/2+N}\mathrm{e}^{-iq_{-N/2+N}x}+\ldots]+\\
\mathrm{e}^{\omega_{-N/2+1}t}\ [\ & \ldots+\tilde{u}^0_{-N/2+1-N}\mathrm{e}^{-iq_{-N/2+1-N}x}+\tilde{u}^0_{-N/2+1}\mathrm{e}^{-iq_{-N/2+1}x}+\tilde{u}^0_{-N/2+1+N}\mathrm{e}^{-iq_{-N/2+1+N}x}+\ldots]\\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
\mathrm{e}^{\omega_0 t}\ [\ & \ldots+\tilde{u}^0_{-N}\mathrm{e}^{-iq_{-N}x}+\tilde{u}^0_0\mathrm{e}^{-iq_0x}+\tilde{u}^0_{0+N}\mathrm{e}^{-iq_Nx}+\ldots]+\\
\mathrm{e}^{\omega_1 t}\ [\ & \ldots+\tilde{u}^0_{1-N}\mathrm{e}^{-iq_{1-N}x}+\tilde{u}^0_1\mathrm{e}^{-iq_1x}+\tilde{u}^0_{1+N}\mathrm{e}^{-iq_{1+N}x}+\ldots]+\\
& \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
\mathrm{e}^{\omega_{N/2-1}t}\ [\ & \ldots+\tilde{u}^0_{N/2-1-N}\mathrm{e}^{-iq_{N/2-1-N}x}+\tilde{u}^0_{N/2-1}\mathrm{e}^{-iq_{N/2-1}x}+\tilde{u}^0_{N/2-1+N}\mathrm{e}^{-iq_{N/2-1+N}x}+\ldots]+\\
\frac{1}{2}\tilde{\mathrm{e}}^{\omega_{N/2}t}\ [\ & \ldots+\tilde{u}^0_{N/2-N}\mathrm{e}^{-iq_{N/2-N}x}+\tilde{u}^0_{N/2}\mathrm{e}^{-iq_{N/2}x}+\tilde{u}^0_{N/2+N}\mathrm{e}^{-iq_{N/2+N}x}+\ldots]
\end{aligned}
\tag{14}
$$

(valid only for $x=x_n$). This can be written in a more compact form:

$$
u(x_n,t)=\sum_{k=-\infty}^{\infty}\tilde{u}^0_k\mathrm{e}^{-iq_kx_n}E_k(t) \tag{149}
$$

This is similar to the exact solution (137) with the exception that the E_k's are equal to the alias values of $\mathrm{e}^{\omega_k t}$:

$$
E_k(t)=\begin{cases}\mathrm{e}^{\omega_K t} & \text{if } k-K=0 \bmod(N),\ K\in(-N/2,N/2)\\ \frac{1}{2}\mathrm{e}^{\omega_{N/2}t}+\frac{1}{2}\mathrm{e}^{\omega_{-N/2}t} & \text{for } k-N/2=0 \bmod(N)\end{cases} \tag{150}
$$

Which means that the modes with $|k| > N/2$ evolve not with the factor ω_k but with the factor of their aliases. For instance, if $N = 8$ the modes $k = -17, -9, -1, 7, 15, 23, 31, \ldots$ all evolve using the factor ω_{-1}. This is a source of errors. If the higher order coefficients $\tilde{u}_k^0$ are very small, then their contribution to the solution is small. If, on the other hand, the function $u^0(x)$ is not smooth (as, for instance, the case of white noise) then the convergence towards the exact solution might need a prohibitely large value of N.

In other words, the exact solution, Eq. (137), involves the evolution of infinite Fourier modes. In the pseudospectral method, indeed infinite modes do appear in the evolution equation, see (149), but for those modes with $|k| > N/2$ their amplifying factors ω_k are not the correct ones. In the Galerkin method, Eq.(143), the modes with $|k| > N/2$ simply do not appear.

As an example we consider the equation:

$$\partial_t u(x,t) = u + \partial_x u \tag{151}$$

For which the amplifying factor is complex:

$$\omega(q) = 1 - iq \tag{152}$$

It is easy to check that the solution, for arbitrary initial condition, is:

$$u(x,t) = \mathrm{e}^t u^0(x+t) \tag{153}$$

In the following figures, we compare the exact solution for a given initial condition with the one obtained with the pseudospectral method. The captions are self contained.

The message is that you need indeed a very good resolution for the Fourier modes. Even if you think that your resolution is fine enough for the initial condition, that does not mean that it will continue to be so as time evolves, since the large modes (which are necessarily incorrect) might increase exponentially with time.

References

[1] J.M. Sanz-Serna, *Fourier Techniques in Numerical Methods for Evolutionary Problems* in Third Granada Lectures in Computational Physics, Springer Lecture Notes in Physics 448, pp. 145-200 (1995).

[2] J. García-Ojalvo and J.M. Sancho, *Noise in Spatially Extended Systems*, Springer, New York (1999).

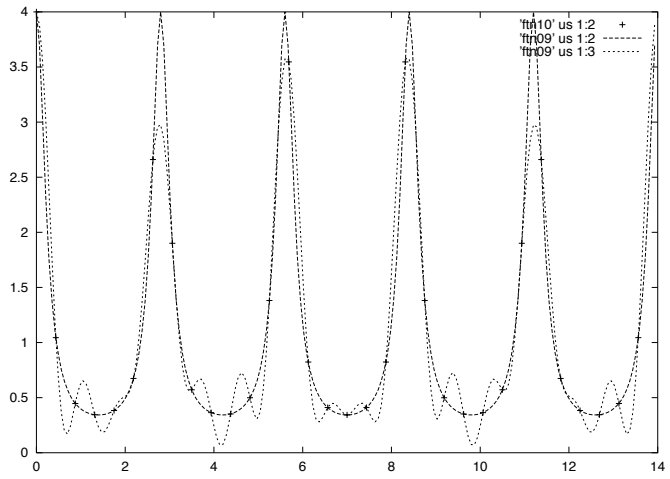

Figure 4: Numerical solution using the pseudospectral method for the equation $\partial_t u = u + \partial_x u$ with the initial condition $u(x, t = 0) = 3/(5 - 4\cos(10\pi x/L) - 0.25\cos(20\pi x/L))$ in the case $L = 14$, $N = 32$ for times $t = 0$ (this and next figures have been produced with the program dif2.f). Line 2 is the exact solution and the symbols are the value of the exact solution at the lattice points. Line 3 is the solution obtained with the numerical method.

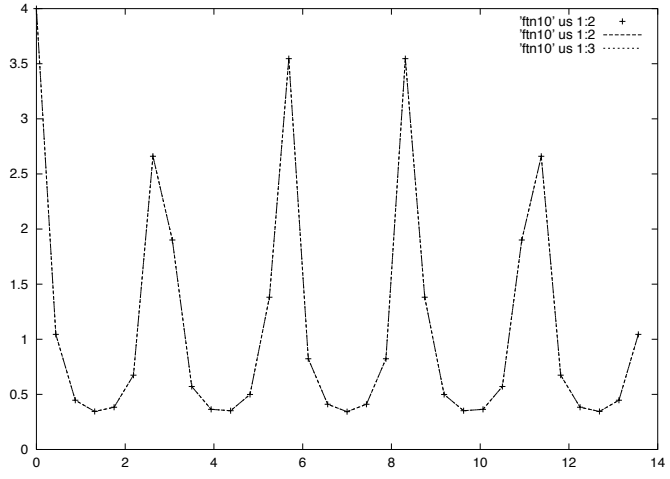

Figure 5: Same as figure (4) but plotting only the lattice points.

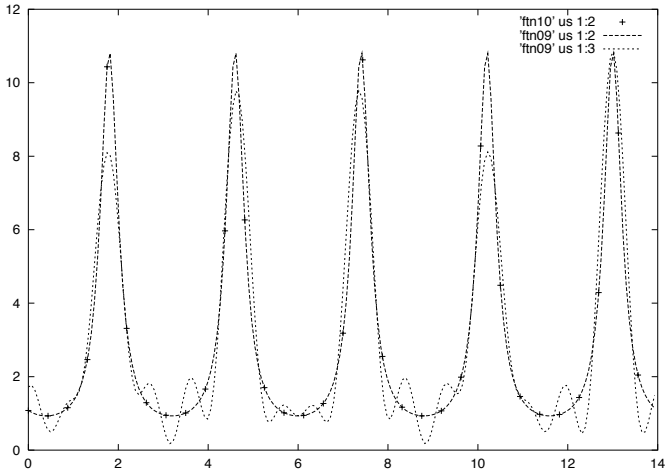

Figure 6: Same as figure (4) at time $t = 1$.

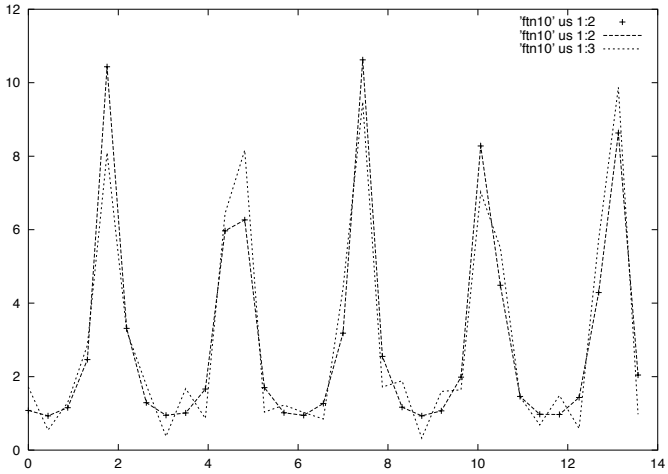

Figure 7: Same as figure (6) but plotting only the lattice points.

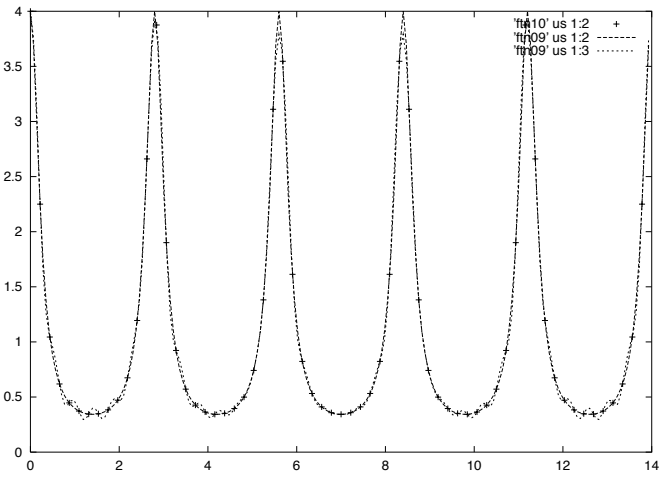

Figure 8: Same as figure (4) with $N = 64$.

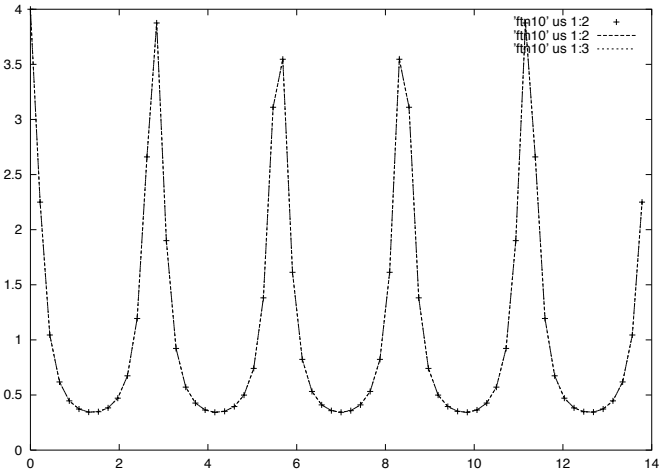

Figure 9: Same as figure (5) with $N = 64$.

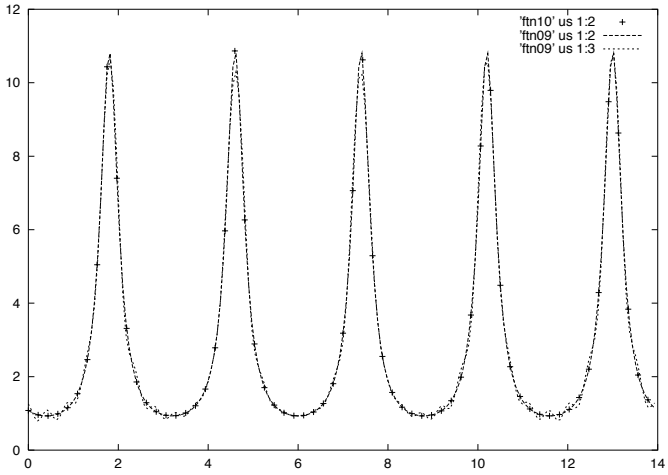

Figure 10: Same as figure (6) with $N = 64$.

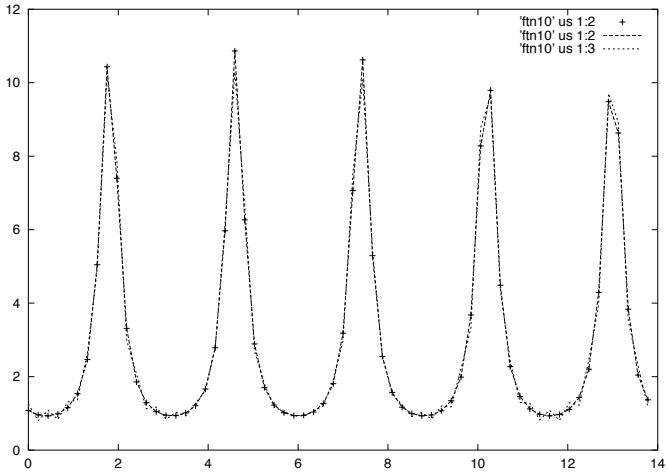

Figure 11: Same as figure (7) with $N = 64$.

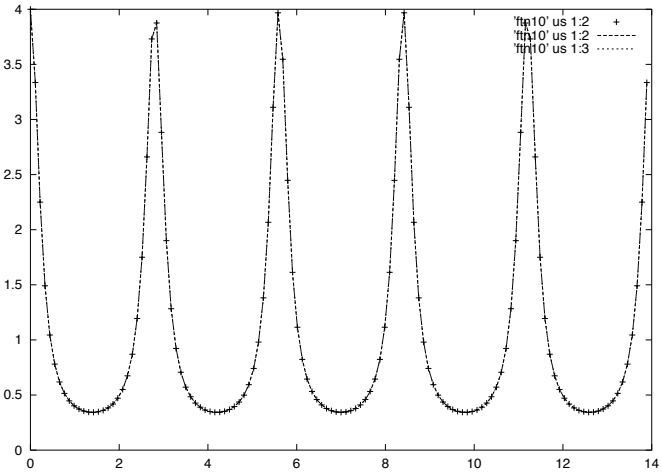

Figure 12: Same as figure (5) with $N = 128$.

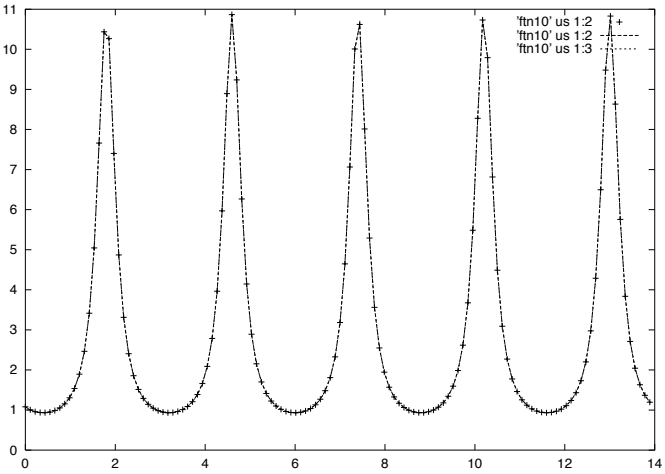

Figure 13: Same as figure (7) with $N = 128$.