Joint ICTP-IAEA College on Advanced Plasma Physics
ICTP
Trieste, Italy
18 – 29 August 2014

Aliasing effects and de-aliasing methods in turbulence simulations

Tutor: Bengt Eliasson
E-mail: bengt.eliasson@strath.ac.uk
Homepage: http://www.strath.ac.uk/physics/staff/academic/bengteliasson/

- The Nyqvist theorem: one needs more than two grid-points per wavelength λ in order to represent a solution on a numerical grid.
- Therefore the wavenumber $k = 2\pi/\lambda$ must be in the range $-k_{x,Nyqvist} < k < k_{x,Nyqvist}$, where $k_{x,Nyqvist} = \pi/\Delta x$ is the Nyqvist wavenumber, $\Delta x = L_x/N_x$ is the grid size, L_x is the domain size, and N_x the number of grid points.
- For an equidistant grid in multiple dimensions, this rule applies separately to each dimensions.
- In a simulation, the grid size can initially be chosen small enough so that this condition is fulfilled, but when the simulation proceeds, nonlinearities and space-dependent coefficients can lead to the violation of the Nyqvist theorem and to aliasing effects.

- Example: multiply $f_1(x,y) = \exp(ik_{x1}x + k_{y1}y)$ and $f_2(x,y) = \exp(ik_{x2}x + k_{y2}y)$ numerically. The result should be $f_1 f_2 = \exp[i(k_{x1}+k_{x2})x + i(k_{y1}+k_{y2})y]$.
- If f_1 and f_2 are represented on a numerical grid, then $(k_{x1}+k_{x2})$ and/or $(k_{y1}+k_{y2})$ may fall outside the the Nyqvist limits and can no more be represented on the grid. If $(k_{x1}+k_{x2}) > k_{x,Nyqvist}$, then the numerical result of the multiplication $f_1 f_2$ is $\exp[i(k_{x1}+k_{x2}-2k_{x,Nyqvist})x + i(k_{y1}+k_{y2})y]$, while if $(k_{x1}+k_{x2}) < -k_{x,Nyqvist}$, the result of the multiplication is $\exp[i(k_{x1}+k_{x2}+2k_{x,Nyqvist})x + i(k_{y1}+k_{y2})y]$.

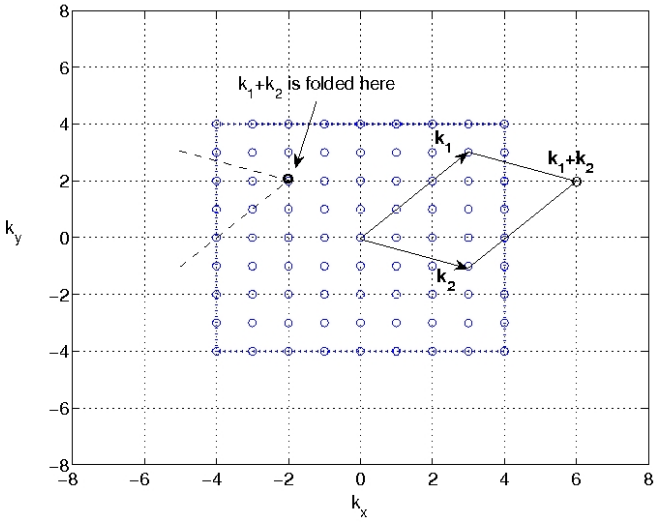


Figure 1: Aliasing effects leads to the folding of the result of multiplication by units of $2k_{x,Nyqvist}$ where $k_{x,Nyqvist} = \pi/\Delta x$. In this example we have used a box size of $L_x \times L_y = 2\pi \times 2\pi$ with $N_x \times N_y = 8$, so that $k_{x,Nyqvist} = k_{y,Nyqvist} = 4$

- This is the aliasing effect. It is illustrated in Figure 1, where $k_{x1} + k_{x2} > k_{x,Nyqvist}$ and the resulting product is folded in into a solution with negative wavenumbers in the x-direction.

- The aliasing effect leads sometimes to numerical instabilities.

Aliasing effects can be avoided by removing the highest part of the wave-spectrum of the solution, so that the frequency components are not folded back into the computational domain. For example, if the physical problem contains quadratic nonlinearities (i.e. the solution multiplied by itself or by derivatives of itself), then the aliasing effect is eliminated by keeping only frequency components corresponding to wavenumbers less than 2/3 of the Nyqvist wavenumber, in each dimension, i.e. components corresponding to $|k_x| \geq (2/3)N_{x,Nyqvist}$ and $|k_y| \geq (2/3)N_{y,Nyqvist}$ are deleted from the solution. This is illustrated in figure 2(a), where wave components corresponding to $|k_x| > 8/3$ and $|k_y| > 8/3$ should be removed from the solution.

Example of implementation in Matlab, 1 dimensions:

```
FR_A=fft(R_A);
FR_A(ceil(Nx/3)+1:Nx-ceil(Nx/3)+1)=0;
R_A=ifft(FR_A);
```

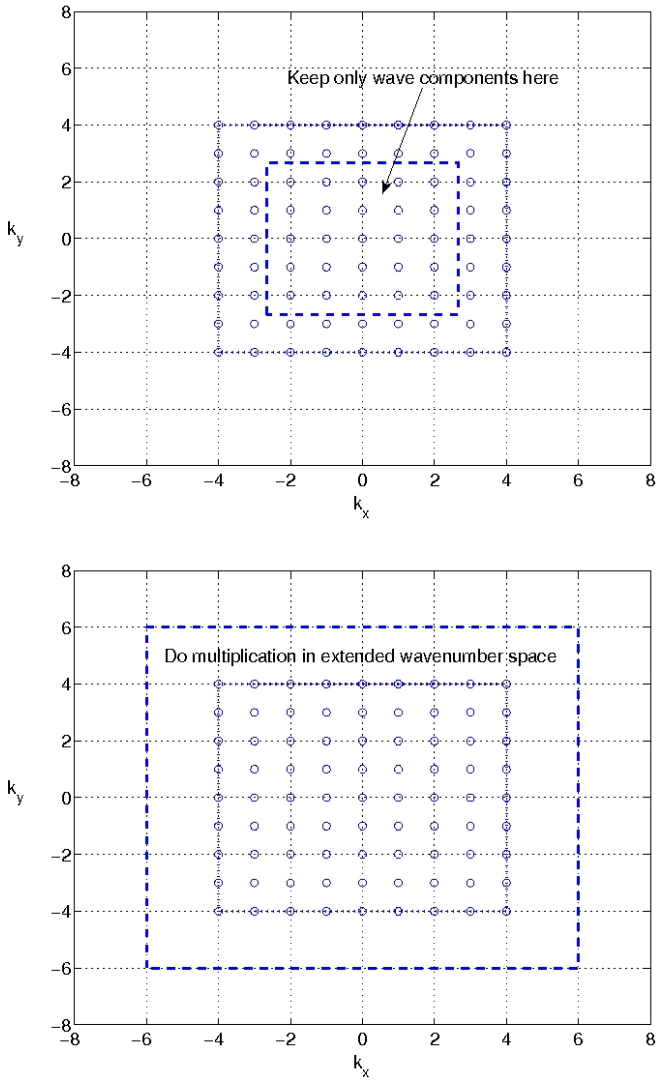


Figure 2: a) The 2/3-rule dealiasing scheme and b) dealising by using spectral interpolation and multiplication in extended wavenumber space.

Alternative method (pointed out to me by Nail Gumerov, U. Maryland[1]):

1. Temporarily interpolate the solution on a finer grid, where the grid size is small enough to avoid the aliasing effect in the multiplication.

2. Do the multiplication on the finer grid.

3. Restrict the solution to the coarser grid again, disregarding the higher frequency components.

In practice, using the pseudospectral method, again for a quadratic nonlinearity:

1. Fourier transform the solution in space, then expand the solution in wavenumber-space with zeros (zero-padding) to an additional 50% or more of its original size, and then inverse Fourier transforming the solution.

2. Do the multiplication on the finer grid.

3. Fourier transform the solution, restrict the solution to its original size, and inverse Fourier transform the solution.

This is illustrated in Fig. 2(b) where the Nyqvist wavenumber is extended from 4 to 6.

Example of implementation in Matlab, 1 dimension (`f1` and `f2`, which are represented as column vectors, should be multiplied):

```
Nx=length(f1);

F1=fft(f1);
F2=fft(f2);
% Remove Nyqvist wavenumbers
F1(Nx/2+1)=0;
F2(Nx/2+1)=0;
% Do the zero-padding
F1=[F1(1:Nx/2) ; zeros(2*(ceil(Nx/4)-1),1) ; F1(Nx/2+1:Nx)];
F2=[F2(1:Nx/2) ; zeros(2*(ceil(Nx/4)-1),1) ; F2(Nx/2+1:Nx)];
% Inverse Fourier transform to obtain the functions on a finer grid
F1=ifft(F1);
F2=ifft(F2);
% Do the multiplication and Fourier transform the result
R=fft(F1.*F2);
```

[1]Nail A. Gumerov et al.,"Efficient spectral and pseudospectral algorithms for 3D simulations of whistler-mode waves in a plasma", J. Comput. Phys. **230**, 2506–2519 (2011)

```
% Restrict the solution to its original size
R=[R(1:Nx/2) ; R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx)];
% Set Nyqvist wavenumber to zero
R(Nx/2+1)=0;
% Do the inverse Fourier transform to obtain the result on the original grid
R=ifft(R);
```

A check that the result is correct is that `R` should be real-valued, i.e. `norm(imag(R))` should give the result 0.

In 2D: Use `fft2` and `ifft2` instead of `fft` and `ifft`:

```
function R=multiply(f1,f2)
  s=size(f1);
  Nx=s(1);
  Ny=s(2);

  F1=fft2(f1);
  F2=fft2(f2);
  F1(Nx/2+1,:)=0;
  F1(:,Ny/2+1)=0;
  F2(Nx/2+1,:)=0;
  F2(:,Ny/2+1)=0;

  F1=[F1(1:Nx/2,:) ; zeros(2*(ceil(Nx/4)-1),Ny) ; F1(Nx/2+1:Nx,:)];
  F1=[F1(:,1:Ny/2) , zeros(Nx+2*(ceil(Nx/4)-1),2*(ceil(Ny/4)-1)) , F1(:,Ny/2+1:N

  F2=[F2(1:Nx/2,:) ; zeros(2*(ceil(Nx/4)-1),Ny) ; F2(Nx/2+1:Nx,:)];
  F2=[F2(:,1:Ny/2) , zeros(Nx+2*(ceil(Nx/4)-1),2*(ceil(Ny/4)-1)) , F2(:,Ny/2+1:N

  F1=ifft2(F1);
  F2=ifft2(F2);

  R=fft2(F1.*F2);

   R=[R(1:Nx/2,1:Ny/2) , R(1:Nx/2,Ny/2+1+2*(ceil(Ny/4)-1):2*(ceil(Ny/4)-1)+Ny);
      R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx, 1:Ny/2) , R(Nx/2+1+2*(ceil(

  R(Nx/2+1,:)=0;
  R(:,Ny/2+1)=0;
  R=ifft2(R);
```

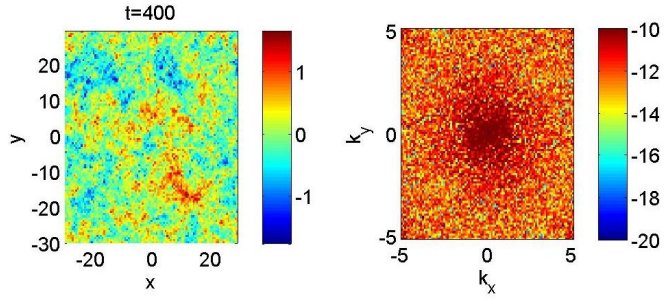


Figure 3: Simulation of 2D incompressible Navier-Stokes equation with dealiasing but no dissipation. Stable but noisy.

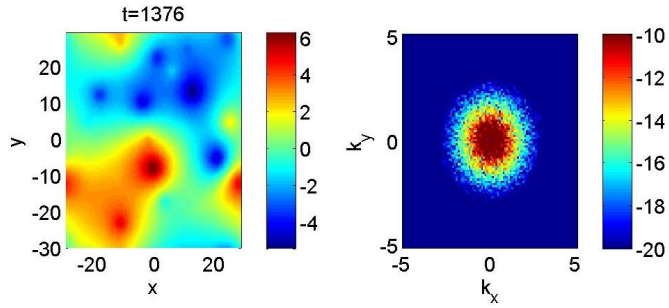


Figure 4: Simulation of 2D incompressible Navier-Stokes equation with dealiasing combined with dissipation at highest 2/3 wavenumbers. Here the numerical simulation shows nicer and "physical" results. One can see vortex merging to larger and larger vortices.

It should be noted that the dealiasing schemes described here are not dissipative, and in turbulence simulation it is necessary to also introduce some dissipation in the system on the highest wavenumbers. As an example we take the two-dimensional (2D) incompressible Navier-Stokes' equation

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla P}{\rho}, \qquad \nabla \cdot \mathbf{u} = 0, \tag{1}$$

where $\mathbf{u}$ is the fluid velocity, P is the hydrostatic pressure, and ρ is the density. Since $\nabla \cdot \mathbf{u} = 0$, we can write the velocity as $\mathbf{u} = \nabla \times \mathbf{A}$ for a vector $\mathbf{A}$. Choosing $\nabla \cdot \mathbf{A} = 0$ and ρ =constant, we can take the curl of both sides of (1) to obtain

$$-\frac{\partial \nabla^2 \mathbf{A}}{\partial t} + \nabla \times \{[(\nabla \times \mathbf{A}) \cdot \nabla](\nabla \times \mathbf{A})\} = 0, \tag{2}$$

In 2D, where $\mathbf{A}$ varies in the x and y directions, $\mathbf{A}$ is directed along the z-

direction, $\mathbf{A} = A_z\widehat{\mathbf{z}}$, and the equation (2) simplifies to

$$-\nabla^2\frac{\partial A_z}{\partial t} + \{A_z, \nabla^2 A_z\} = 0, \tag{3}$$

where the Poisson bracket is defined as

$$\{f, g\} = \frac{\partial f}{\partial x}\frac{\partial g}{\partial y} - \frac{\partial f}{\partial y}\frac{\partial g}{\partial x}, \tag{4}$$

and $\nabla^2 = \partial^2/\partial x^2 + \partial^2/\partial y^2$. As initial condition we take random numbers on the grid points with values between -1 and $+1$. With the second de-aliasing described above, the simulation is stable (Fig. 3) but noisy, which shows that the de-aliasing scheme has not introduced any damping in the system. In Fig. 4, the solution is weakly damped on the highest $2/3$ of the wavenumbers, so that we have introduced a numerical dissipation range. In this case one can see vortices developing and merging into larger and larger vortices. The solution is remarkably smooth.

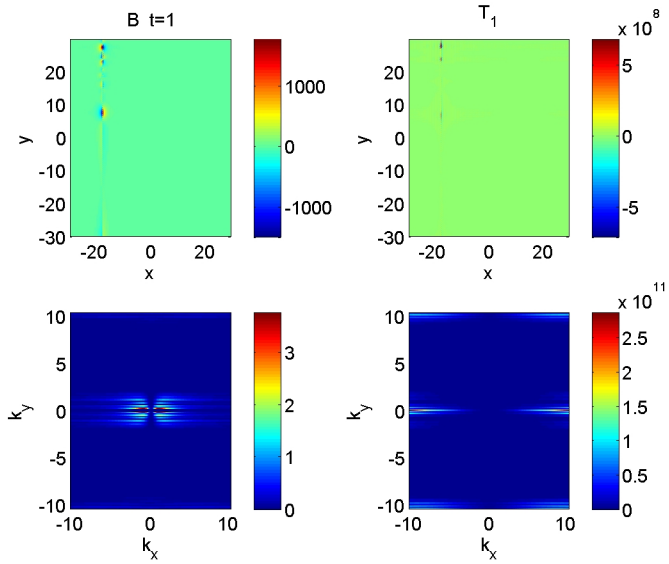


Figure 5: Simulation of MEDV model with no de-aliasing and no dissipation. The simulation shows strong numerical instability.

As a second example, we take a model for drift waves in plasmas, where the electron fluid is coupled to the electron temperature fluctuations via temperature and density gradients. The magnetic electron drift vortex (MEDV) modes are

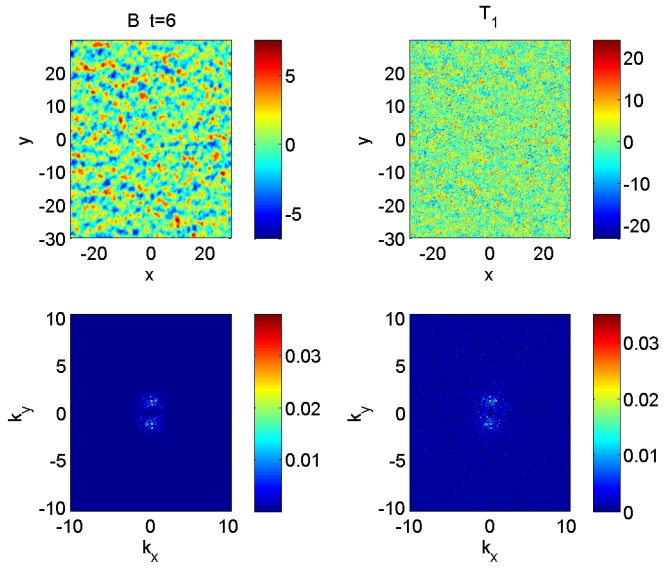


Figure 6: Simulation of MEDV with de-aliasing but no dissipation. The simulation is numerically stable but noisy. The introduction of a weak numerical dissipation at highest wavenumbers would give nicer results.

described by the model[2]

$$\frac{\partial}{\partial t}\left(B_z - \nabla^2 B_z\right) = \{B_z, \nabla^2 B_z\} - \frac{\partial T_1}{\partial y}, \tag{5}$$

$$\frac{\partial T_1}{\partial t} = -\{B, T_1\} - \sigma \frac{\partial B}{\partial y}, \tag{6}$$

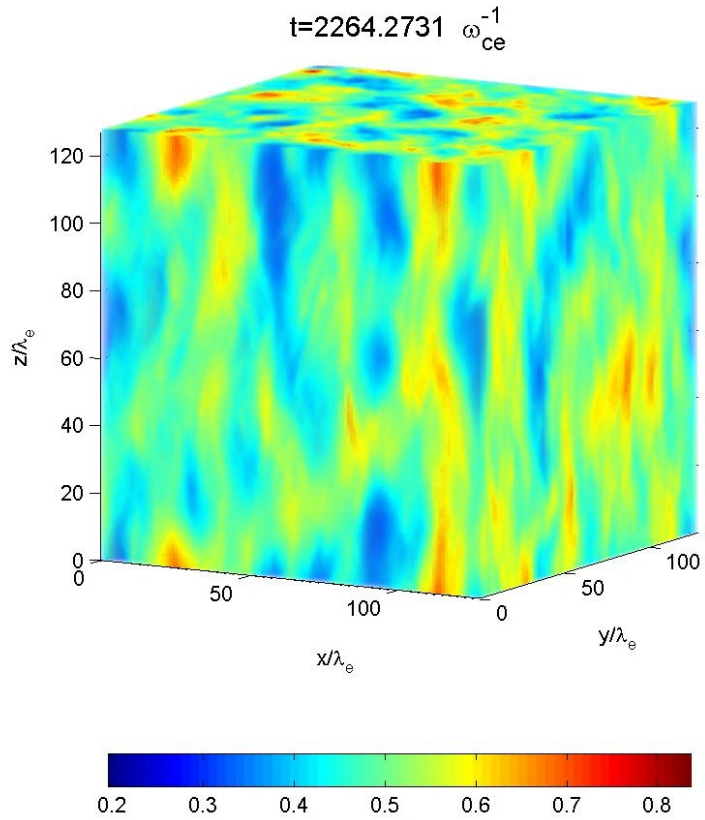
where B is the magnetic field and T_1 is the temperature fluctuations (in normalized units). σ can be positive or negative. We will use $\sigma = -1$. This system is very sensitive to aliasing effects, and we see in Fig. 5 that the solution "explodes" after short time. On the other hand, after introducing de-aliasing using the second method above, but no numerical dissipation, the solution stabilizes (Fig. 6).

Finally the 3D case: Electron magnetohydrodynamic equations

$$\frac{\partial}{\partial t}(\mathbf{B} - \lambda_e^2 \nabla^2 \mathbf{B}) = \frac{c^2}{4\pi e n_0} \nabla \times [(\mathbf{B} - \lambda_e^2 \nabla^2 \mathbf{B}) \times (\nabla \times \mathbf{B})], \tag{7}$$

[2]B. Eliasson, P. K. Shukla, and V. P. Pavlenko, Physics of Plasmas **16**, 042306, doi:10.1063/1.3103785 (2009).

```
% Do the de-aliased cross product between two functions f1 and f2.
function R=crossmult(f1,f2)
s=size(f1);
Nx=s(1);
Ny=s(2);
Nz=s(3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f1(1:Nx,1:Ny,1:Nz,1)=fftn(f1(1:Nx,1:Ny,1:Nz,1));
f1(1:Nx,1:Ny,1:Nz,2)=fftn(f1(1:Nx,1:Ny,1:Nz,2));
f1(1:Nx,1:Ny,1:Nz,3)=fftn(f1(1:Nx,1:Ny,1:Nz,3));

f1(Nx/2+1,1:Ny,1:Nz,1:3)=0;
f1(1:Nx,Ny/2+1,1:Nz,1:3)=0;
f1(1:Nx,1:Ny,Nz/2+1,1:3)=0;

f1=cat(1,f1(1:Nx/2,1:Ny,1:Nz,1:3) , zeros(2*(ceil(Nx/4)-1),Ny,Nz,3) , f1(Nx/2+1:Nx,1:Ny,1:Nz,1:3));
f1=cat(2,f1(:,1:Ny/2,:,1:3) , zeros(Nx+2*(ceil(Nx/4)-1),2*(ceil(Ny/4)-1),Nz,3) , f1(:,Ny/2+1:Ny,:,1:3));
f1=cat(3,f1(:,:,1:Nz/2,1:3) , zeros(Nx+2*(ceil(Nx/4)-1),Ny+2*(ceil(Ny/4)-1),2*(ceil(Nz/4)-1),3) , f1(:,:,Nz/2+1:Nz,1:3));

f1(:,:,:,1)=ifftn(f1(:,:,:,1));
f1(:,:,:,2)=ifftn(f1(:,:,:,2));
f1(:,:,:,3)=ifftn(f1(:,:,:,3));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

f2(1:Nx,1:Ny,1:Nz,1)=fftn(f2(1:Nx,1:Ny,1:Nz,1));
f2(1:Nx,1:Ny,1:Nz,2)=fftn(f2(1:Nx,1:Ny,1:Nz,2));
f2(1:Nx,1:Ny,1:Nz,3)=fftn(f2(1:Nx,1:Ny,1:Nz,3));

f2(Nx/2+1,1:Ny,1:Nz,1:3)=0;
f2(1:Nx,Ny/2+1,1:Nz,1:3)=0;
f2(1:Nx,1:Ny,Nz/2+1,1:3)=0;

f2=cat(1,f2(1:Nx/2,:,:,1:3) , zeros(2*(ceil(Nx/4)-1),Ny,Nz,3) , f2(Nx/2+1:Nx,:,:,1:3));
f2=cat(2,f2(:,1:Ny/2,:,1:3) , zeros(Nx+2*(ceil(Nx/4)-1),2*(ceil(Ny/4)-1),Nz,3) , f2(:,Ny/2+1:Ny,:,1:3));
f2=cat(3,f2(:,:,1:Nz/2,1:3) , zeros(Nx+2*(ceil(Nx/4)-1),Ny+2*(ceil(Ny/4)-1),2*(ceil(Nz/4)-1),3) , f2(:,:,Nz/2+1:Nz,1:3));

f2(:,:,:,1)=ifftn(f2(:,:,:,1));
f2(:,:,:,2)=ifftn(f2(:,:,:,2));
f2(:,:,:,3)=ifftn(f2(:,:,:,3));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

R=cross(f1,f2,4);
clear f1 f2

R(:,:,:,1)=fftn(R(:,:,:,1));
R(:,:,:,2)=fftn(R(:,:,:,2));
R(:,:,:,3)=fftn(R(:,:,:,3));

R=cat(1, ...
    cat(2, ...
      cat(3, ...
        R(1:Nx/2,1:Ny/2,1:Nz/2,1:3), ...
        R(1:Nx/2,1:Ny/2,Nz/2+1+2*(ceil(Nz/4)-1):2*(ceil(Nz/4)-1)+Nz,1:3) ...
      ), ...
      cat(3, ...
        R(1:Nx/2,Ny/2+1+2*(ceil(Ny/4)-1):2*(ceil(Ny/4)-1)+Ny,1:Nz/2,1:3), ...
        R(1:Nx/2,Ny/2+1+2*(ceil(Ny/4)-1):2*(ceil(Ny/4)-1)+Ny,Nz/2+1+2*(ceil(Nz/4)-1):2*(ceil(Nz/4)-1)+Nz,1:3) ...
      ) ...
    ), ...
    cat(2, ...
      cat(3, ...
        R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx, 1:Ny/2,1:Nz/2,1:3), ...
        R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx, 1:Ny/2, Nz/2+1+2*(ceil(Nz/4)-1):2*(ceil(Nz/4)-1)+Nz,1:3) ...
      ), ...
      cat(3, ...
        R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx, Ny/2+1+2*(ceil(Ny/4)-1):2*(ceil(Ny/4)-1)+Ny,1:Nz/2,1:3), ...
        R(Nx/2+1+2*(ceil(Nx/4)-1):2*(ceil(Nx/4)-1)+Nx, Ny/2+1+2*(ceil(Ny/4)-1):2*(ceil(Ny/4)-1)+Ny,Nz/2+1+2*(ceil(Nz/4)-1):2*(ceil(Nz/4)-1)+Nz,1:3) ...
      ) ...
    ) ...
  );

R(Nx/2+1,1:Ny,1:Nz,1:3)=0;
R(1:Nx,Ny/2+1,1:Nz,1:3)=0;
R(1:Nx,1:Ny,Nz/2+1,1:3)=0;

R(1:Nx,1:Ny,1:Nz,1)=ifftn(R(1:Nx,1:Ny,1:Nz,1));
R(1:Nx,1:Ny,1:Nz,2)=ifftn(R(1:Nx,1:Ny,1:Nz,2));
R(1:Nx,1:Ny,1:Nz,3)=ifftn(R(1:Nx,1:Ny,1:Nz,3));
```



In conclusion: in turbulence simulations, de-aliasing schemes can be used to avoid nonlinear numerical instabilities due to aliasing effects, but numerical dissipation should also be used to reduce noise in the simulations.