Small post processor

Felix Dietzsch[a]

[a] *Institut fr Chemieingenieurwesen und Energieverahrenstechnik,...*

Contents

1. Clear complete workspace

For new Matlab projects best practise is to clear the complete workspace and the command window. It is also a good habit to close all remaining figures since Matlab does not automatically open a new window each time a plot call is invoked.

```
display('Clear workspace ...')
path('./functions',path) % add functions directory the
    Matlab path
close all % close all figures
clear all % clear workspace
clc % clear command window
%set(0,'DefaultFigureWindowStyle','docked')
[datadir,flag]=ClearWs();
```

The above mentioned clears are performed in the function `ClearWs`. In addition some basic parameters like the name of the data directory or the dimensionality of the problem are also defined.

```
function [datadir,flag]=ClearWs()
    datadir='data'; % specify the directory containg the
        data
    flag='3D'; % specify the subdirectory of data
end
```

2. Read data files

During the evaluation of the function `ReadData` all data files neseccary for the calculation of the spectrum and the correlation coefficients are read, namely the velocity components. In addition the import operations are enclosed in a **`tic;`**... **`;toc`** block measuring the time needed for reading the ASCII data. What you should get from the tic/toc block is that most of the time is spend during data I/O (Input/Output operations), nearly 220 s. The actual computation needs only about 8 s. What you can easily calculate from this is that the computation of the spectrum is nearly 27 times faster then the data import. Why the computation of Fourier transforms is that fast we will come to that later. Although the ASCII data format ist not the prefered choice in terms of speed and size, we will use it since other methodologies require additional knowledge of data processing. Just for your information a very famous and highly protable data format is hdf5. It is a software library that runs on a range of computational platforms, from laptops to massively parallel systems and implements a high-level API (Application programming interface) with C, C++, Fortran 90, and Java interfaces. Besides its hierarchical structure it is highly optimized for parallel I/O operations and can be read by nearly all data processing tools.

```
display('Read data ...')
[uvel,vvel,wvel,time_read] = ReadData(datadir,flag,'uvel',
    'vvel','wvel');
% test=importdata('data/3D/CFX_velocity_field.dat');
% uvel=reshape(test(:,1),33,33,33);
% vvel=reshape(test(:,2),33,33,33);
% wvel=reshape(test(:,3),33,33,33);
```

```
function [uvel,vvel,wvel,time] = ReadData(datadir,flag,...
                                          u_name,...
```

```
                     v_name,...
                     w_name)
    tic; % enable timer
    uvel=importdata([datadir,'/',flag,'/',u_name]);
    vvel=importdata([datadir,'/',flag,'/',v_name]);
    wvel=importdata([datadir,'/',flag,'/',w_name]);
    time = toc; % end timer
end
```

3. Set neccessary parameters

For further computations it is important to define some parmeters of the DNS simulation such as

- Number of grid points in on direction n_p,
- Physical length of one direction L_x,
- Physical grid spacing $\triangle x$,
- Kinematic viscosity ν.

```
display('Set parameters ...')
[u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel);
% u=u-mean2(u);
% v=v-mean2(v);
% w=w-mean2(w);
```

```
function [u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel)
    dim=385; % number of points in one dimension
    Lx=0.4;
%     dim=33;
%     Lx=3.2e-2; % domain size
    Ly=Lx;
    Lz=Lx;
    dx=Lx/(dim-1); % grid spacing
    dy=dx;
    dz=dx;
    nu=1.7e-5; % viscosity
    u=reshape(uvel,dim,dim,dim); % reshape arrays to have them in 3D
    v=reshape(vvel,dim,dim,dim);
    w=reshape(wvel,dim,dim,dim);
    clear uvel vvel wvel
end
```

4. Compute 3D spectrum

The core of the code is contained in the function `PowerSpec`. It computes the three dimensional energy spectrum from the given velocity fields, obtained from a direct numerical simulation. Although the theoretical analysis is relatively demanding compared to one dimensional spectra its worth investing the effort. The theory of one dimensional spectra relies on the assumption that the propagation of spectral waves (κ_1) is in the direction of the observed velocity fields or to say it differently one dimenional spectra and correlation functions are Fourier transform pairs. The theory of correlation functions will be discussed in section 7. A key drawback of this theory is that the calculated spectrum has contributions from all wavenumbers $\boldsymbol{\kappa}$, so that the magnitude of $\boldsymbol{\kappa}$ can be appreciably larger than κ_1. This phenomenon is called aliasing. In order to avoid these aliasing effects is also possible to produce correlations that involve all possible directions. The three dimensional Fourier transformation of such a correlation produces a spectrum that not only depends on a single wavenumber but on the wavenumber vector κ_i. Though the directional information contained in κ_i eliminates the aliasing problem the complexity makes a physical reasoning impossible. For homogeneous isotropic turbulence the situation can be considerably simplified. From the knowledge that the velocity field is isotropic it can be shown that the velocity spectrum tensor is fully determined by

$$\Phi_{ij}(\boldsymbol{\kappa}) = A(\kappa)\delta_{ij} + B(\kappa)\kappa_i\kappa_j, \tag{1}$$

where $A(\kappa)$ and $B(\kappa)$ are arbitrary scalar functions. Since we assume incompressible fluids (mathematically expressed by $\nabla \cdot u = 0$ or $\kappa_i u_i = 0$ the following condition holds

$$\kappa_i \, \Phi_{ij}(\boldsymbol{\kappa}) = 0. \tag{2}$$

It can be shown that this yields a relation between A and B by means of

$$B(\kappa) = -\frac{A(\kappa)}{\kappa^2} \tag{3}$$

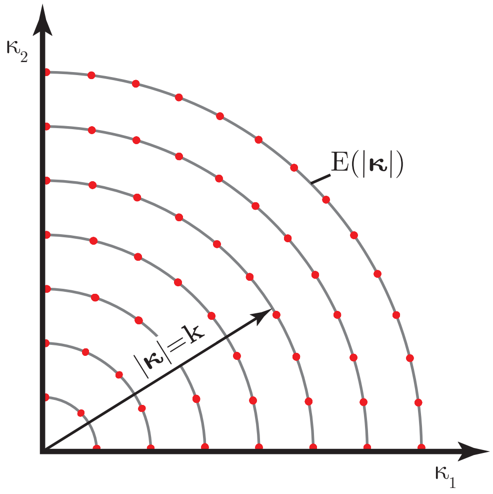


Fig. 1: Illustration of the two dimensional shell integration

In the end this gives a relation between the three dimensional energy spectrum function $E(|\boldsymbol{\kappa}|)$ and the velocity spectrum tensor Φ_{ij}.

$$\Phi_{ij} = \frac{E(|\boldsymbol{\kappa}|)}{4\pi\,(|\boldsymbol{\kappa}|)^2}\left(\delta_{ij} - \frac{\kappa_i\kappa_j}{(|\boldsymbol{\kappa}|)^2}\right) \tag{4}$$

The question is now how the remaining variable (A or B) can be determined. Regarding the turbulent kinetic energy we know that

$$k = \int\limits_{-\infty}^{\infty} E(|\boldsymbol{\kappa}|)\,\mathrm{d}k = \sum_{\boldsymbol{\kappa}} E(\boldsymbol{\kappa}) = \sum_{\boldsymbol{\kappa}} \frac{1}{2}\left\langle u^*(\boldsymbol{\kappa})\,u(\boldsymbol{\kappa})\right\rangle = \iiint\limits_{-\infty}^{\infty} \frac{1}{2}\Phi_{ii}(\boldsymbol{\kappa})\,\mathrm{d}\boldsymbol{\kappa}. \tag{5}$$

Comparing the second and last expression we get

$$E(|\boldsymbol{\kappa}|) = \oiint \frac{1}{2}\,\Phi_{ii}(\boldsymbol{\kappa})\,\mathrm{d}S(\kappa). \tag{6}$$

This integral can be solved analytically by utilizing again the assumption of isotropy. For these kind of flows the energy spectrum function can be regarded as the sum of kinetic energy (in wave number space) on different energy levels. Each of these energy levels is denoted by a spherical shell in wave number space. Since the surface of a sphere is completly determined by its radius the surface integral can be solved analytically. The idea of this integration is illustrated in Fig. 1. As a result of this one gets

$$E(|\boldsymbol{\kappa}|) = \oiint \frac{1}{2}\,\Phi_{ii}(\boldsymbol{\kappa})\,\mathrm{d}S(\kappa) = 4\pi(|\boldsymbol{\kappa}|)^2\,\Phi_{ii}(|\boldsymbol{\kappa}|). \tag{7}$$

Introducing this relation to equations (1) and (3) one arrives at an expression for the variable B.

$$B = -\frac{E(|\boldsymbol{\kappa}|)}{4\pi(|\boldsymbol{\kappa}|)^2} \tag{8}$$

Together with the approximation of the integral of Φ (equation (5))

$$\iiint_{-\infty}^{\infty} \frac{1}{2}\Phi_{ii}(\boldsymbol{\kappa})\,\mathrm{d}\boldsymbol{\kappa} \approx \frac{1}{2}\sum_{\boldsymbol{\kappa}} \Phi_{ii}(\boldsymbol{\kappa})\,(\Delta\kappa)^3, \tag{9}$$

where $\Delta\kappa$ refers to the step size in wave number space, the final expression of the three dimensional discrete energy spectrum can be derived.

$$E(|\boldsymbol{\kappa}|) = 2\pi(|\boldsymbol{\kappa}|)^2\,\frac{\langle u^*(\boldsymbol{\kappa})\,u(\boldsymbol{\kappa})\rangle}{(\Delta\kappa)^3} \tag{10}$$

The calling sequence for the computation of the energy spectrum reads

```
display('Compute spectrum...')
[spectrum,k,bin_counter,time_spec] = PowerSpec(u,v,w,Lx,dim);
```

```
function [spectrum,k,bin_counter,time] = PowerSpec(u,v,w,...
                                                      L,dim)
    tic;
    uu_fft=fftn(u);
    vv_fft=fftn(v);
    ww_fft=fftn(w);

    muu = abs(uu_fft)/length(u)^3;
    mvv = abs(vv_fft)/length(v)^3;
    mww = abs(ww_fft)/length(w)^3;

    muu = muu.^2;
    mvv = mvv.^2;
    mww = mww.^2;

    rx=[0:1:dim-1] - (dim-1)/2;
    ry=[0:1:dim-1] - (dim-1)/2;
    rz=[0:1:dim-1] - (dim-1)/2;

    test_x=circshift(rx',[(dim+1)/2 1]);
    test_y=circshift(ry',[(dim+1)/2 1]);
    test_z=circshift(rz',[(dim+1)/2 1]);

    [X,Y,Z]= meshgrid(test_x,test_y,test_z);
    r=(sqrt(X.^2+Y.^2+Z.^2));

    dx=2*pi/L;
    k=[1:(dim-1)/2].*dx;
    spectrum=zeros(size(k,2),1);
    bin_counter=zeros(size(k,2),1);
    for N=2:(dim-1)/2-1
        picker = (r(:,:,:)*dx <= (k(N+1) + k(N))/2) & ...
                 (r(:,:,:)*dx > (k(N) + k(N-1))/2);
        spectrum(N) = sum(muu(picker))+...
                      sum(mvv(picker))+...
                      sum(mww(picker));
        bin_counter(N) = size(find(picker==1),1);
    end
    % compute first value of spectrum
    picker = (r(:,:,:)*dx <= (k(2) + k(1))/2);
    spectrum(1) = sum(muu(picker))+...
                  sum(mvv(picker))+...
                  sum(mww(picker));
    bin_counter(1) = size(find(picker==1),1);
    % compute last value of spectrum
    picker = (r(:,:,:)*dx > (k(end) + k(end-1))/2 & ...
              r(:,:,:)*dx <= k(end));
    spectrum(end) = sum(muu(picker))+...
                    sum(mvv(picker))+...
                    sum(mww(picker));
    bin_counter(end) = size(find(picker==1),1);
    %compute final spectrum
    spectrum = spectrum*2*pi.*k'.^2./(bin_counter.*dx.^3);
    time=toc;
end
```

5. Compute dissipation and turbulent kinetic energy

The function **SpecProp** calculates the kinetic energy both from the velocities and the previously computed spectrum. The latter one is calcualted by

$$k = \int_{-\infty}^{\infty} E(|\boldsymbol{\kappa}|)\,\mathrm{d}|\boldsymbol{\kappa}| \tag{11}$$

A second integral, also evaluated in this routine, gives the value of the Dissipation

$$\epsilon = 2\int_{-\infty}^{\infty} \nu(|\boldsymbol{\kappa}|)^2 E(|\boldsymbol{\kappa}|)\,\mathrm{d}|\boldsymbol{\kappa}|, \tag{12}$$

where ν refers to the kinematic viscosity. The calling sequence reads

```
display('Compute kinetic energy...')
[Dissipation,kin_E_Sp,kin_E_Ph,up] = SpecProp(spectrum,k,
    ...
                                               nu,u,v,w,dim);
```

```
function [Dissipation,kin_E_Sp,kin_E_Ph,up] = ...
                                        SpecProp(E,k,nu,u,v,w,dim)
    kin_E_Sp = trapz(k,E);
    Dissipation = trapz(k,2*nu.*k.^2.*E');
    up = sqrt(1/3/dim^3*sum(sum(sum(u.^2+v.^2+w.^2))));
    kin_E_Ph = 3/2*up^2;
end
```

6. Kolmogorov properties

According to the Kolomogorov hypotheses the length scale η, characterisitc velocity u_η and the characteristic time scale τ of the smallest swirls in the flow are computed within the function **KolmoScale**. From a dimensionality analysis Kolmogorov derived

$$\eta = \left(\frac{\nu^3}{\epsilon}\right)^{1/4}, \tag{13}$$

$$u_\eta = (\epsilon\,\nu)^{1/4}, \tag{14}$$

$$\tau_\eta = \left(\frac{\nu}{\epsilon}\right)^{1/2}. \tag{15}$$

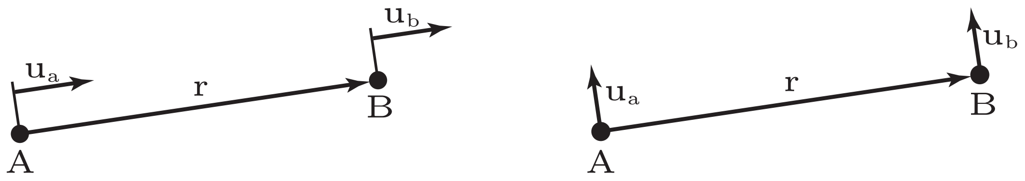
For further reading concerning his theory it is refered to Pope [1], Hinze [2] and Tennekes and Lumley [3].

```
display('Compute Kolmogorov scales...')
[eta,u_eta,tau]=KolmoScale(nu,Dissipation);
```

```
function [eta,u_eta,tau]=KolmoScale(nu,Dissipation)
    eta = (nu^3/Dissipation)^(1/4);
    u_eta = (nu*Dissipation)^(1/4);
    tau = (nu/Dissipation)^(1/2);
end
```

7. Compute correlations

From a general perspective correlation functions are a measure of how much two physical quantities are connected. So how is this helpful for the analysis of turbulent flows? For seemingly chaotic and random procescees it would be beneficial if we had a measure of how the velocity at point A is influenced by the velocity at point B. A maybe more intuitive quantity that can be calculated from the correlation functions is the integral lengthscale which gives a measure of the largest eddies in the flow. In fluid dynamics one generally differentiates between two forms of correlation functions, the longitudinal and the transversal or lateral correlation function. The difference between both forms is illustrated in figure 2. Computing a correlation can be a tedious work (requireing tremendeous effort) especially if you have large data sets. From theory it is well known that the multiplication of the transform of a data set and its complex conjugate are an accurate representation of the correlation function. Using the FFT approach this gives an enormeous speed advantage.



(a) Longitudinal correlation **(b)** Transversal correlation

Fig. 2: Illustration of different correlation functions

Since we already computed the veloity correlation tensor we may use this result in order to compute the correlation tensor.

$$R_{ij} = \frac{cov(U_i, U_j)}{\sqrt{\sigma_i^2\, \sigma_j^2}} = \frac{\left\langle u_i'\, u_j' \right\rangle}{\sqrt{\sigma_i^2\, \sigma_j^2}} \tag{16}$$

```
display('Compute Correlations...')
[R11,R22,r,R1,R2,R3]=Correlation(u,v,w,Lx,dim);
```

The content of `Correlation` reads

```
function [R11,R22,r,R1,R2,R3]=Correlation(u,v,w,Lx,dim)
    scaling = 1;
    NFFT = 2.^nextpow2(size(u)); %power of 2 fitting length of u
    u_fft=fftn(u,NFFT)./scaling;
    NFFT = 2.^nextpow2(size(v));
    v_fft=fftn(v,NFFT)./scaling;
    NFFT = 2.^nextpow2(size(w));
    w_fft=fftn(w,NFFT)./scaling;

    Rij_x=(u_fft.*conj(u_fft)); % compute velo. correlation tensor
    Rij_y=(v_fft.*conj(v_fft));
    Rij_z=(w_fft.*conj(w_fft));

    % x-component
    NFFT = 2.^nextpow2(size(u_fft));
    R1=ifftn(Rij_x,NFFT)/std2(u)^2/dim^3;

    % y-component
    NFFT = 2.^nextpow2(size(v_fft));
    R2=ifftn(Rij_y,NFFT)/std2(v)^2./dim^3;
    % z-component
    NFFT = 2.^nextpow2(size(w_fft));
    R3=ifftn(Rij_z,NFFT)/std2(w)^2./dim^3;

    R11 = (reshape(R3(1,1,:),NFFT(1),1)+R2(1,:,1)'+R1(:,1,1))/3;
    R11 = R11(1:size(u_fft)/2+1);
    %
    R1_22 = (R1(1,:,1)+R3(1,:,1))/2;
    R2_22 = (R2(:,1,1)+R3(:,1,1))/2;
    R3_22 = (reshape(R1(1,1,:),size(u_fft,1),1)+...
             reshape(R2(1,1,:),size(u_fft,1),1))/2;

    R22 = (R1_22'+R2_22+R3_22)/3;
    R22 = R22(1:size(u_fft)/2+1);

    r = linspace(0,Lx/2,size(u_fft,1)/2+1)/(Lx/2);
end
```

References

[1] S. B. Pope, Turbulent Flows, Cambridge University Press, 1 edition, 2000.

[2] J. O. Hinze, Turbulence, Mcgraw-Hill College, 2 edition, 1975.

[3] H. Tennekes, J. L. Lumley, A first course in turbulence, The MIT Press, 1972.

8. Plotting

```
display('Save results...')
close all
comte=importdata('Comte-Bellot.txt');
kC=comte.data(:,1).*100;
EC=comte.data(:,2)./100^3;
vkp=importdata('data/3D/CTRL_TURB_ENERGY');
```

```
h=loglog(vkp(:,1),vkp(:,3),'*-b');hold on
set(h,'LineWidth',1);
h=loglog(k,spectrum,'r-s');
set(h,'LineWidth',1);
h=loglog(kC,EC,'*-g');
set(h,'LineWidth',1);
legend('VKP','Dietzsch','Comte-Bellot')
saveas(gcf,'spectrum.eps','psc2')
```