Small post processor

1. Contents

2. 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.

```
path('./functions',path) % add functions directory the Matlab path
[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()
    close all
    clear all
    clc

    datadir='data'; % specify the directory containg the data
    flag='3D'; % specify the subdirectory of data
end
```

3. 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.

```
[uvel,vvel,wvel,time_read] = ReadData(datadir,flag);
```

```
function [uvel,vvel,wvel,time] = ReadData(datadir,flag)
    tic; % enable timer
    uvel=importdata([datadir,'/',flag,'/uvel']);
    vvel=importdata([datadir,'/',flag,'/vvel']);
    wvel=importdata([datadir,'/',flag,'/wvel']);
    time = toc; % end timer
end
```

4. 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.

```
[u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel);
```

```
function [u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel)
    dim=256; % number of points in one dimension
    Lx=5e-3; % domain size
    Ly=Lx;
    Lz=Lx;
    dx=Lx/dim; % 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
```

5. Compute 3D spectrum

The core of the provided 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 a later section. 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 aliasing effects usually connected with a one dimensional spectrum it 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 simplified by integrating the three dimensional spectrum over spherical shells. The idea of this integration is illustrated in Fig. 1

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

Since the surface of a sphereis completly determined by its radius the surface integral can be solved analytically.

$$\oiint ( ) \mathrm{d}S(\kappa) = 4\pi\kappa^2 \cdot ( ) \tag{2}$$

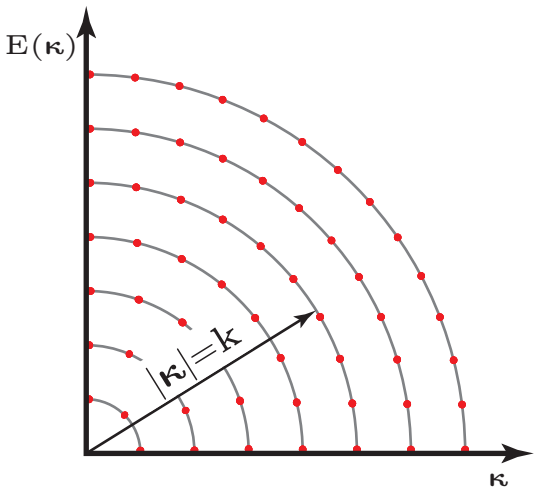

Fig. 1: Illustration of two dimensional shell integration

This leads to

$$E(|\kappa|) = \frac{1}{2}\,\Phi_{ii}(|\boldsymbol{\kappa}|) \tag{3}$$

```
[spectrum,k,mu,mv,mw,time_spec] = PowerSpec(u,v,w,Lx);
```

The content of `PowerSpec` reads

```
function [spectrum,k,mu,mv,mw,time] = power_spec(u,v,w,L)
% nx = size(u,1);
% ny = size(u,2);
% nz = size(u,3);
tic;
NFFT = 2.^nextpow2(size(u)); % next power of 2 fitting the length of u
u_fft=fftn(u,NFFT);
v_fft=fftn(v,NFFT);
w_fft=fftn(w,NFFT);

% Calculate the numberof unique points
NumUniquePts = ceil((NFFT(1)+1)/2);

% FFT is symmetric, throw away second half
u_fft = u_fft(1:NumUniquePts,1:NumUniquePts,1:NumUniquePts);
v_fft = v_fft(1:NumUniquePts,1:NumUniquePts,1:NumUniquePts);
w_fft = w_fft(1:NumUniquePts,1:NumUniquePts,1:NumUniquePts);

mu = abs(u_fft)/length(u)^3;
mv = abs(v_fft)/length(v)^3;
mw = abs(w_fft)/length(w)^3;
```

```
22
23 % Take the square of the magnitude of fft of x.
24 mu = mu.^2;
25 mv = mv.^2;
26 mw = mw.^2;
27
28 % Since we dropped half the FFT, we multiply mx by 2 to keep the same energy.
29 % The Nyquist component, if it exists, is unique and should not be multiplied by 2.
30
31 if rem(NFFT, 2) % odd nfft excludes Nyquist point
32   mu(2:end,2:end,2:end) = mu(2:end,2:end,2:end)*2;
33   mv(2:end,2:end,2:end) = mv(2:end,2:end,2:end)*2;
34   mw(2:end,2:end,2:end) = mw(2:end,2:end,2:end)*2;
35 else
36   mu(2:end -1,2:end -1,2:end -1) = mu(2:end -1,2:end -1,2:end -1)*2;
37   mv(2:end -1,2:end -1,2:end -1) = mv(2:end -1,2:end -1,2:end -1)*2;
38   mw(2:end -1,2:end -1,2:end -1) = mw(2:end -1,2:end -1,2:end -1)*2;
39 end
40 % Compute the radius vector along which the energies are sumed
41 mx=NumUniquePts;
42 my=NumUniquePts;
43 mz=NumUniquePts;
44
45 dx=pi/L;
46 dy=pi/L;
47 dz=pi/L;
48 for I=1:mx
49        X0(I)=(I)*dx;
50 end
51
52 for J=1:my
53        Y0(J)=(J)*dy;
54 end
55
56 for K=1:mz
57        Z0(K)=(K)*dz;
58 end
59
60 for I=1:mx
61     for J=1:my
62         for K=1:mz
63             R(I,J,K)=sqrt(X0(I)*X0(I)+Y0(J)*Y0(J)+Z0(K)*Z0(K));
64         end
65     end
66 end
67
68 % P=mod(nx,2);
69 % if (P < 1)
70 %     Nmax=mx-0.5;
71 % else
72     Nmax=mx;
73 % end
74 spectrum=zeros(Nmax,1);
75 for N=1:Nmax
76     Radius1=sqrt(3)*(N-1)*dx; %lower radius bound
77     Radius2=sqrt(3)*N*dx; %upper radius bound
78     % bild logical index for selecting values lying on the shell
```

```
    logical = (Radius1 <= R(:,:,:)) & (R(:,:,:) < Radius2);
    % build summation over shell components
    T_EVP1=sum(mu(logical))+sum(mv(logical))+sum(mw(logical));
    % put them at position N in the spectrum
    spectrum(N)=T_EVP1./2;
end
k=[1:Nmax].*dx;
spectrum = 1./(2*pi)^3.*spectrum;
time=toc;
end
```

6. Compute dissipation and turbulent kinetic energy

```
[Dissipation,kin_E_Sp,kin_E_Ph,up] = SpecProp(spectrum,k,nu,u,v,w);
```

The content of `SpecProp` reads

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

7. Kolmogrov properties

```
[eta,u_eta,tau]=KolmoScale(nu,Dissipation);
```

The content of `KolmoScale` reads

```
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
```

8. Compute model spectra

```
PlotModelSpec(k,spectrum,Dissipation,up,Lx,eta,nu);
```

The content of `PlotModelSpec` reads

```
function PlotModelSpec(k,spectrum,Dissipation,up,Lx,eta,nu)
    % Von Karman–Pao Spektren
    close all
```

```
    kd = k(end);
    ke = pi/Lx/2;
    A = 1.5;
    up = mean2(up);

    VKP1 = A*up^5/Dissipation.*(k./ke).^4./(1+(k./ke).^2).^(17/6).*exp(-3/2*A.*(k./kd).^(4/3));

    kd = 1./eta;
    VKP2 = 1.5*(k./kd).^(-5/3)./(Dissipation*nu^5)^(-1/4).*exp(-1.5*1.5.*(k./kd).^(4/3));

    % Kolmogorov Spektrum
    Kolmo=1.5*Dissipation^(2/3)*(k.^(-5/3));

    % Plot spectra
    h=loglog(k,Kolmo,k,VKP1,k,VKP2,k,spectrum);
    set(h,'LineWidth',2);

    h=legend('Kolmogorov','VKP1','VKP2','Computed');
    set(h,'Location','SouthWest')
end
```

9. Compute correlations

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

$$R_{ij} = \frac{cov(U_i, U_j)}{\sqrt{\sigma_i^2\,\sigma_j^2}} = \frac{(u_i' - \mu_i)\,(u_j - \mu_j)}{\sqrt{\sigma_i^2\,\sigma_j^2}} \tag{4}$$

```
[R11,R22,r,R1,R2,R3]=Correlation(u,v,w,Lx);
```

The content of `Correlation` reads

```
function [R11,R22,r,R1,R2,R3]=correlation(u,v,w,Lx)
    scaling = 1;
    NFFT = 2.^nextpow2(size(u)); % next power of 2 fitting the length of u
    u_fft=fftn(u,NFFT)./scaling; %2 pi --> definition of FFT
    %
    NFFT = 2.^nextpow2(size(v));
    v_fft=fftn(v,NFFT)./scaling;
    %
    NFFT = 2.^nextpow2(size(w));
    w_fft=fftn(w,NFFT)./scaling;

```

```
12      Rij_x=(u_fft.*conj(u_fft)); % compute velo. correlation tensor
13      Rij_y=(v_fft.*conj(v_fft));
14      Rij_z=(w_fft.*conj(w_fft));
15
16      % x-component
17      NFFT = 2.^nextpow2(size(u_fft));
18      R1=ifftn(Rij_x,NFFT)/std2(u)^2./prod(NFFT);
19      % y-component
20      NFFT = 2.^nextpow2(size(v_fft));
21      R2=ifftn(Rij_y,NFFT)/std2(v)^2./prod(NFFT);
22      % z-component
23      NFFT = 2.^nextpow2(size(w_fft));
24      R3=ifftn(Rij_z,NFFT)/std2(w)^2./prod(NFFT);
25
26      R11 = (reshape(R3(1,1,:),NFFT(1),1)+R2(1,:,1)'+R1(:,1,1))/3;
27      R11 = R11(1:size(u_fft)/2+1);
28      %
29      R1_22 = (R1(1,:,1)+R3(1,:,1))/2;
30      R2_22 = (R2(:,1,1)+R3(:,1,1))/2;
31      R3_22 = (reshape(R1(1,1,:),size(u_fft,1),1)+reshape(R2(1,1,:),size(u_fft,1),1))/2;
32
33      R22 = (R1_22'+R2_22+R3_22)/3;
34      R22 = R22(1:size(u_fft)/2+1);
35
36      r = linspace(0,Lx/2,size(u_fft,1)/2+1)/(Lx/2);
37  end
```