

Small post processor

Felix Dietzsch^a

^a*Institut für Chemieingenieurwesen und Energieverahrenstechnik,...*

Contents

1	Contents	1
2	Clear complete workspace	1
3	Read data files	2
4	Set neccessary parameters	3
5	Compute 3D spectrum	4
6	Compute dissipation and turbulent kinetic energy	7
7	Kolmogrov properties	8
8	Compute model spectra	8
9	Compute correlations	9

1. Contents

- Clear complete workspace
- Read data files
- Set neccessary parameters
- Compute 3D spectrum
- Compute dissipation and turbulent kinetic energy
- Kolmogrov properties
- Compute model spectra
- Compute correlations

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.

```

1 path('./functions',path) % add functions directory the
   Matlab path
2 close all % close all figures
3 clear all % clear workspace
4 clc % clear command window
5 set(0, 'DefaultFigureWindowStyle', 'docked')
6 [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.

```

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

```

3. Read data files

During the evaluation of the function `ReadData` all data files necessary 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 spent during data I/O (Input/Output operations), nearly 220s. The actual computation needs only about 8s. What you can easily calculate from this is that the computation of the spectrum is nearly 27 times faster than the data import. Why the computation of Fourier transforms is that fast we will come to that later. Although the ASCII data format is not the preferred 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 portable 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,'uvel','vvel','wvel');
```

```
1 test=importdata('data/3D/CFX_velocity_field.dat');
2 uvel=reshape(test(:,1),33,33,33);
3 vvel=reshape(test(:,2),33,33,33);
4 wvel=reshape(test(:,3),33,33,33);

1 function [uvel,vvel,wvel,time] = ReadData(datadir,flag,...
2                                     u_name,...
3                                     v_name,...
4                                     w_name)
5     tic; % enable timer
6     uvel=importdata([datadir,'/',flag,'/',u_name]);
7     vvel=importdata([datadir,'/',flag,'/',v_name]);
8     wvel=importdata([datadir,'/',flag,'/',w_name]);
9     time = toc; % end timer
10 end
```

4. Set necessary parameters

For further computations it is important to define some parameters 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 Δx ,
- Kinematic viscosity ν .

```
1 [u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel);
2 % u=u-mean2(u);
3 % v=v-mean2(v);
4 % w=w-mean2(w);
```

```

1 function [u,v,w,dim,Lx,dx,nu]=Params(uvel,vvel,wvel)
2 %      dim=257; % number of points in one dimension
3 %      Lx=0.4;
4      dim=33;
5      Lx=3.2e-2; % domain size
6      Ly=Lx;
7      Lz=Lx;
8      dx=Lx/(dim-1); % grid spacing
9      dy=dx;
10     dz=dx;
11     nu=1.7e-5; % viscosity
12     u=reshape(uvel,dim,dim,dim); % reshape arrays to have
        them in 3D
13     v=reshape(vvel,dim,dim,dim);
14     w=reshape(wvel,dim,dim,dim);
15     clear uvel vvel wvel
16 end

```

5. 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 dimensional 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 $\boldsymbol{\kappa}_i$. Though the directional information contained in $\boldsymbol{\kappa}_i$ eliminates the aliasing problem the complexity makes a physical reasoning impossible. For homogeneous isotropic turbulence the situation can be sim-

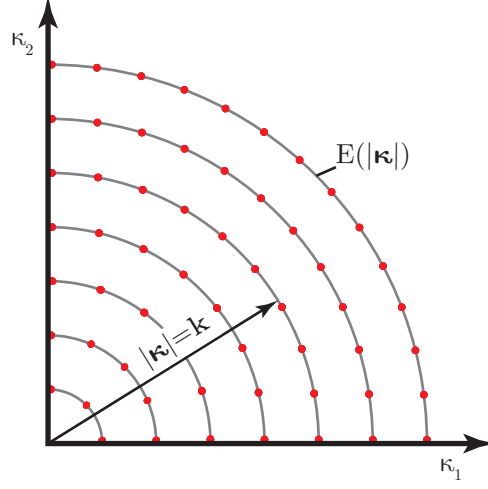


Fig. 1: Illustration of the two dimensional shell integration

plified by integrating the three dimensional spectrum over spherical shells. The idea of this integration is illustrated in Fig. 1

$$E(\kappa) = \oint\!\!\!\oint E(\boldsymbol{\kappa}) dS(\kappa) = \oint\!\!\!\oint \frac{1}{2} \Phi_{ii}(\boldsymbol{\kappa}) dS(\kappa) \quad (1)$$

Since the surface of a sphere is completely determined by its radius the surface integral can be solved analytically.

$$\oint\!\!\!\oint () dS(\kappa) = 4\pi\kappa^2 \cdot () \quad (2)$$

This leads to

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

```
1 [spectrum,k,time_spec] = PowerSpec(u,v,w,Lx,dim);
```

```
1 function [spectrum,k,time] = PowerSpec(u,v,w,L,dim)
2     tic;
```

```

3      NFFT = 2.^nextpow2(size(u)); % next power of 2 fitting
      the length of u
4
5      uu_fft=fftn(u);
6      vv_fft=fftn(v);
7      ww_fft=fftn(w);
8
9      muu = abs(uu_fft)/length(u)^3;
10     mvv = abs(vv_fft)/length(v)^3;
11     mww = abs(ww_fft)/length(w)^3;
12
13     % Take the square of the magnitude of fft of x.
14     %mu = mu.^2;
15     %mv = mv.^2;
16     %mw = mw.^2;
17     muu = muu.^2;
18     mvv = mvv.^2;
19     mww = mww.^2;
20
21     % % % % %   for i=1:dim-1
22     % % % % %       xx(i) = i-(dim+1)/2;
23     % % % % %       yy(i) = i-(dim+1)/2;
24     % % % % %       zz(i) = i-(dim+1)/2;
25     % % % % %   end
26     % equivalent see above
27     rx=[0:1:dim-1] - (dim-1)/2;
28     ry=[0:1:dim-1] - (dim-1)/2;
29     rz=[0:1:dim-1] - (dim-1)/2;
30
31
32     test_x=circshift(rx', [(dim+1)/2 1]);
33     test_y=circshift(ry', [(dim+1)/2 1]);
34     test_z=circshift(rz', [(dim+1)/2 1]);
35
36     [X,Y,Z]= meshgrid(test_x,test_y,test_z);
37     r=(sqrt(X.^2+Y.^2+Z.^2));
38
39     dx=2*pi/L;
40     k=[1:(dim-1)/2].*dx;
41     spectrum=zeros(size(k,2),1);
42     bin_counter=zeros(size(k,2),1);
43     for N=2:(dim-1)/2-1
44
45         picker = (r(:, :, :)*dx <= (k(N+1) + k(N))/2) & ...
46                 (r(:, :, :)*dx > (k(N) + k(N-1))/2);

```

```

47         spectrum(N) = sum(muu(picker))+sum(mvv(picker))+
           sum(mww(picker));
48         bin_counter(N) = size(find(picker==1),1);
49     end
50     % special handling for first and last energy value
       necessary
51     picker = (r(:, :, :)*dx <= (k(2) + k(1))/2);
52     spectrum(1) = sum(muu(picker))+sum(mvv(picker))+sum(
       mww(picker));
53     bin_counter(1) = size(find(picker==1),1);
54     picker = (r(:, :, :)*dx > (k(end) + k(end-1))/2 & r
       (:, :, :)*dx <= k(end));
55     spectrum(end) = sum(muu(picker))+sum(mvv(picker))+sum(
       mww(picker));
56     bin_counter(end) = size(find(picker==1),1);
57     spectrum=0.5*spectrum./(dx);%(2*pi)^3;%
58     % %      bin_counter.*4*pi.*k'.^2;
59     time=toc;
60 end

```

6. 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 calculated by

$$k = \int E(\kappa) d\kappa \quad \kappa = |\boldsymbol{\kappa}| \quad (4)$$

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

$$\epsilon = 2 \int \nu \kappa^2 E(\kappa) d\kappa \quad (5)$$

```

1  [Dissipation, kin_E_Sp, kin_E_Ph, up] = SpecProp(spectrum, k,
           nu, u, v, w, dim);
2  kin_E_Ph
3  kin_E_Sp

```

```

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

```

7. Kolmogorov properties

```

1 [eta,u_eta,tau]=KolmoScale(nu,Dissipation);
2 eta
3 u_eta
4 tau

```

The content of KolmoScale reads

```

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

```

8. Compute model spectra

```

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

```

The content of PlotModelSpec reads

```

1 function PlotModelSpec(k,spectrum,Dissipation,up,Lx,eta,nu
    )
2     % Von Karman-Pao Spektren
3     close all
4     kd = k(end);
5     ke = pi/Lx/2;
6     A = 1.5;
7     up = mean2(up);
8

```



```

9      VKP1 = A*up^5/Dissipation.*(k./ke).^4./(1+(k./ke).^2).
      ^ (17/6).* ...
10      exp(-3/2*A.*(k./kd).^ (4/3));
11
12      kd = 1./eta;
13      VKP2 = 1.5*(k./kd).^(-5/3)./(Dissipation*nu^5)^(-1/4).
      * ...
14      exp(-1.5*1.5.*(k./kd).^ (4/3));
15
16      % Kolmogorov Spektrum
17      Kolmo=1.5*Dissipation^(2/3)*(k.^(-5/3));
18
19      % Plot spectra
20      h=loglog(k,Kolmo,k,VKP1,k,VKP2,k,spectrum);
21      set(h,'LineWidth',2);
22
23      h=legend('Kolmogorov','VKP1','VKP2','Computed');
24      set(h,'Location','SouthWest')
25  end

```

9. Compute correlations

Computing a correlation can be a tedious work (requiring tremendous 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 enormous speed advantage. Since we already computed the velocity correlation tensor we may use this result in order to compute the correlation tensor.

$$R_{ij} = \frac{cov(U_i, U_j)}{\sqrt{\sigma_i^2 \sigma_j^2}} = \frac{\langle u'_i u'_j \rangle}{\sqrt{\sigma_i^2 \sigma_j^2}} \quad (6)$$

```

1  [R11,R22,r,R1,R2,R3]=Correlation(u,v,w,Lx,dim);
2  close all
3  figure
4  plot(r,R11,r,R22)
5  legend('R11','R22')

```

The content of Correlation reads

```

1 function [R11,R22,r,R1,R2,R3]=Correlation(u,v,w,Lx,dim)
2     scaling = 1;
3     NFFT = 2.nextpow2(size(u)); % next power of 2 fitting
        the length of u
4     u_fft=fft(u,NFFT)./scaling; %2 pi —> definition of
        FFT
5
6     %
7     NFFT = 2.nextpow2(size(v));
8     v_fft=fft(v,NFFT)./scaling;
9     %
10    NFFT = 2.nextpow2(size(w));
11    w_fft=fft(w,NFFT)./scaling;
12
13    Rij_x=(u_fft.*conj(u_fft)); % compute velo.
        correlation tensor
14    Rij_y=(v_fft.*conj(v_fft));
15    Rij_z=(w_fft.*conj(w_fft));
16
17    % x-component
18    NFFT = 2.nextpow2(size(u_fft));
19    R1=ifft(Rij_x,NFFT)/std2(u)^2/dim^3;
20
21    % y-component
22    NFFT = 2.nextpow2(size(v_fft));
23    R2=ifft(Rij_y,NFFT)/std2(v)^2./dim^3;
24    % z-component
25    NFFT = 2.nextpow2(size(w_fft));
26    R3=ifft(Rij_z,NFFT)/std2(w)^2./dim^3;
27
28    R11 = (reshape(R3(1,1,:),NFFT(1),1)+R2(1,:,1)'+R1
        (:,1,1))/3;
29    R11 = R11(1:size(u_fft)/2+1);
30    %
31    R1_22 = (R1(1,:,1)+R3(1,:,1))/2;
32    R2_22 = (R2(:,1,1)+R3(:,1,1))/2;
33    R3_22 = (reshape(R1(1,1,:),size(u_fft,1),1)+...
        reshape(R2(1,1,:),size(u_fft,1),1))/2;
34
35
36    R22 = (R1_22'+R2_22+R3_22)/3;
37    R22 = R22(1:size(u_fft)/2+1);
38

```

```

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

```

```

1 close all
2 sohm=importdata('data/3D/SPECTRUM_00.SET');
3 h=loglog(sohm(:,1),sohm(:,2),'*-b');hold on
4 set(h,'LineWidth',1);
5 h=loglog(k,spectrum,'r-s');
6 set(h,'LineWidth',1);
7 legend('Sohm','Dietzsch')
8 saveas(gcf,'spectrum.eps','psc2')

```