Small post processor

Felix Dietzsch[a]

[a]*River Valley Technologies, SJP Building, Cotton Hills, Trivandrum, Kerala, India 695014*

Contents

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.

```matlab
path('./functions',path) % add functions directory the
    Matlab path
close all % close all figures
clear all % clear workspace
clc % clear command window

[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.

```matlab
function [datadir,flag]=ClearWs()
    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,'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
```

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);
% 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=257; % number of points in one dimension
    Lx=3.2e-1; % 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
```

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

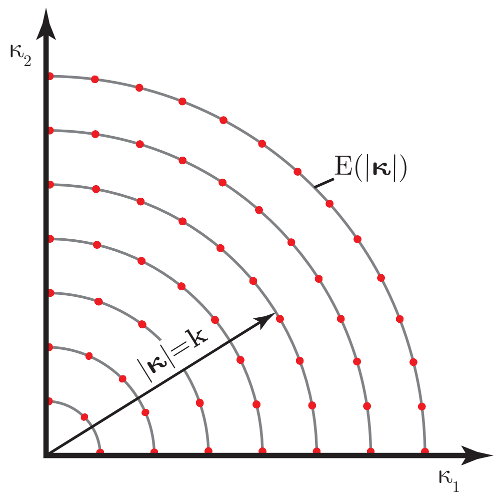

Fig. 1: Illustration of the two dimensional shell integration

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.

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

This leads to

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

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

The content of `PowerSpec` reads

```
function [spectrum,k,time] = PowerSpec(u,v,w,L,dim)
    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);
    % NFFT=33;
    uu_fft=fftn(u);
    vv_fft=fftn(v);
    ww_fft=fftn(w);

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

    % Take the square of the magnitude of fft of x.
    %mu = mu.^2;
    %mv = mv.^2;
    %mw = mw.^2;
    muu = muu.^2;
    mvv = mvv.^2;
    mww = mww.^2;

    % Since we dropped half the FFT, we multiply mx by 2
       to keep the same energy.
    % The Nyquist component, if it exists, is unique and
       should not be multiplied by 2.

    %if rem(NFFT, 2) % odd nfft excludes Nyquist point
        %mu(2:end,2:end,2:end) = mu(2:end,2:end,2:end)*2;
        %mv(2:end,2:end,2:end) = mv(2:end,2:end,2:end)*2;
        %mw(2:end,2:end,2:end) = mw(2:end,2:end,2:end)*2;
    %else
        %mu(2:end -1,2:end -1,2:end -1) = mu(2:end -1,2
            :end -1,2:end -1)*2;
        %mv(2:end -1,2:end -1,2:end -1) = mv(2:end -1,2
            :end -1,2:end -1)*2;
        %mw(2:end -1,2:end -1,2:end -1) = mw(2:end -1,2
            :end -1,2:end -1)*2;
    %end
    % Compute the radius vector along which the energies
       are sumed
    %mx=NumUniquePts;
    %my=NumUniquePts;
    %mz=NumUniquePts;

% % % % %   for i=1:dim-1
% % % % %       xx(i) = i-(dim+1)/2;
% % % % %       yy(i) = i-(dim+1)/2;
% % % % %       zz(i) = i-(dim+1)/2;
% % % % %     end
    % equivalent see above
    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));

% % %   rx=[0:17 -15:-1]*2*pi/L;
% % %   ry=[0:17 -15:-1]*2*pi/L;
% % %   rz=[0:17 -15:-1]*2*pi/L;
% % %
% % %   [X,Y,Z]= meshgrid(rx,ry,rz);
% % %   r=(sqrt(X.^2+Y.^2+Z.^2));
% % %
% % %     test_spec=zeros(29,1);
% % %     for i=1:(dim+1)/2
% % %         for j=1:(dim+1)/2
% % %             for k=1:(dim+1)/2
% % %                 pos=1+round(r(i,j,k)/(2*pi/L)+0.5);
% % %               if (r(i,j,k)>((dim-1)*pi/L/1E6) && r(i,j,k)<(dim-1)*pi/L)
% % %                       test_spec(pos)=test_spec(pos)+muu(i,j,k)+mvv(i,j,k)+mww(i,j,k);
% % %                 end
% % %             end
% % %         end
% % %     end
% % %
% % %     test_spec=0.5*test_spec;
    dx=2*pi/L;
    k=[1:(dim-1)/2].*dx;
    for N=2:(dim-1)/2-1
%       Radius1=sqrt(3)*(N-1); %lower radius bound
%       Radius2=sqrt(3)*N; %upper radiusbound
%         Radius1= k(N);
%         Radius2= k(N+1);
%       picker = ((Radius1 <= r(:,:,:)*dx) & (r(:,:,:)*dx) < Radius2);
        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));
%         picker = (r(:,:,:)*dx > (k(N+1)-k(N))/2 + k(N));
%         spectrum(N+1) = sum(muu(picker))+sum(mvv(picker))+sum(mww(picker));
    end
    % special handling for first and last energy value necessary
    picker = (r(:,:,:)*dx <= (k(2) + k(1))/2);
    spectrum(1) = sum(muu(picker))+sum(mvv(picker))+sum(mww(picker));
    picker = (r(:,:,:)*dx > (k(end) + k(end-1))/2);
    spectrum(end) = sum(muu(picker))+sum(mvv(picker))+sum(mww(picker));
    spectrum=0.5*spectrum./(2*pi/L);%(2*pi)^3;%

%   dx=2*pi/L;
    %dy=pi/L;
    %dz=pi/L;
    %for I=1:mx
        %X0(I)=(I-1)*dx;
    %end
%
```

```
    %for J=1:my
        %Y0(J)=(J-1)*dy;
    %end
%
    %for K=1:mz
        %Z0(K)=(K-1)*dz;
    %end
%
    %for I=1:mx
        %for J=1:my
            %for K=1:mz
                %R(I,J,K)=sqrt(X0(I)*X0(I)+Y0(J)*Y0(J)+Z0(
                   K)*Z0(K));
            %end
        %end
    %end

    %% P=mod(nx,2);
    %% if (P < 1)
    %%     Nmax=mx-0.5;
    %% else
%   Nmax=(dim+1)/2;
    %% end
    %spectrum=zeros(Nmax,1);
    %for N=1:Nmax
        %Radius1=sqrt(3)*(N-1)*dx; %lower radius bound
        %Radius2=sqrt(3)*N*dx; %upper radius bound
        %% bild picker index for selecting values lying on
            the shell
        %picker = (Radius1 <= R(:,:,:)) & (R(:,:,:) <
           Radius2);
        %% build summation over shell components
        %T_EVP1=sum(mu(picker))+sum(mv(picker))+sum(mw(
           picker));
        %% put them at position N in the spectrum
        %spectrum(N)=T_EVP1.*0.5.*6;
    %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,dim);
kin_E_Ph
kin_E_Sp
```

The content of `SpecProp` reads

```
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
```

7. Kolmogrov properties

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

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.

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

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

```
close all
figure
plot(r,R11,r,R22)
legend('R11','R22')
```

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)); % 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;

    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
```

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