Done

Definition (Done)

- Matlab code is parallelized
  - works for arbitrary timestep and crossssection (no out of memory issue)
- Get smooth, non-aliased FFT. Uses xcorr.
- Code repo is available at: `https://github.com/michaelraba/correlateFix`
  - key files:
    - 1) xcg.m
    - 2) fftStep.m
    - 3) findAzimuthalModes3.m runs $fft(\theta)$ / xcorr(r) with $r^{1/2} corrMat r'^{1/2}$ / fft(x-dir)

  +Alternative to (3) findAzimuthalModes2.m runs / first xcorr(r) with $r^{1/2}$corrMat $r'^{1/2}$ then followed by $fft(\theta)$, followed by fft(x-dir).
    - This produces the same graph/result.

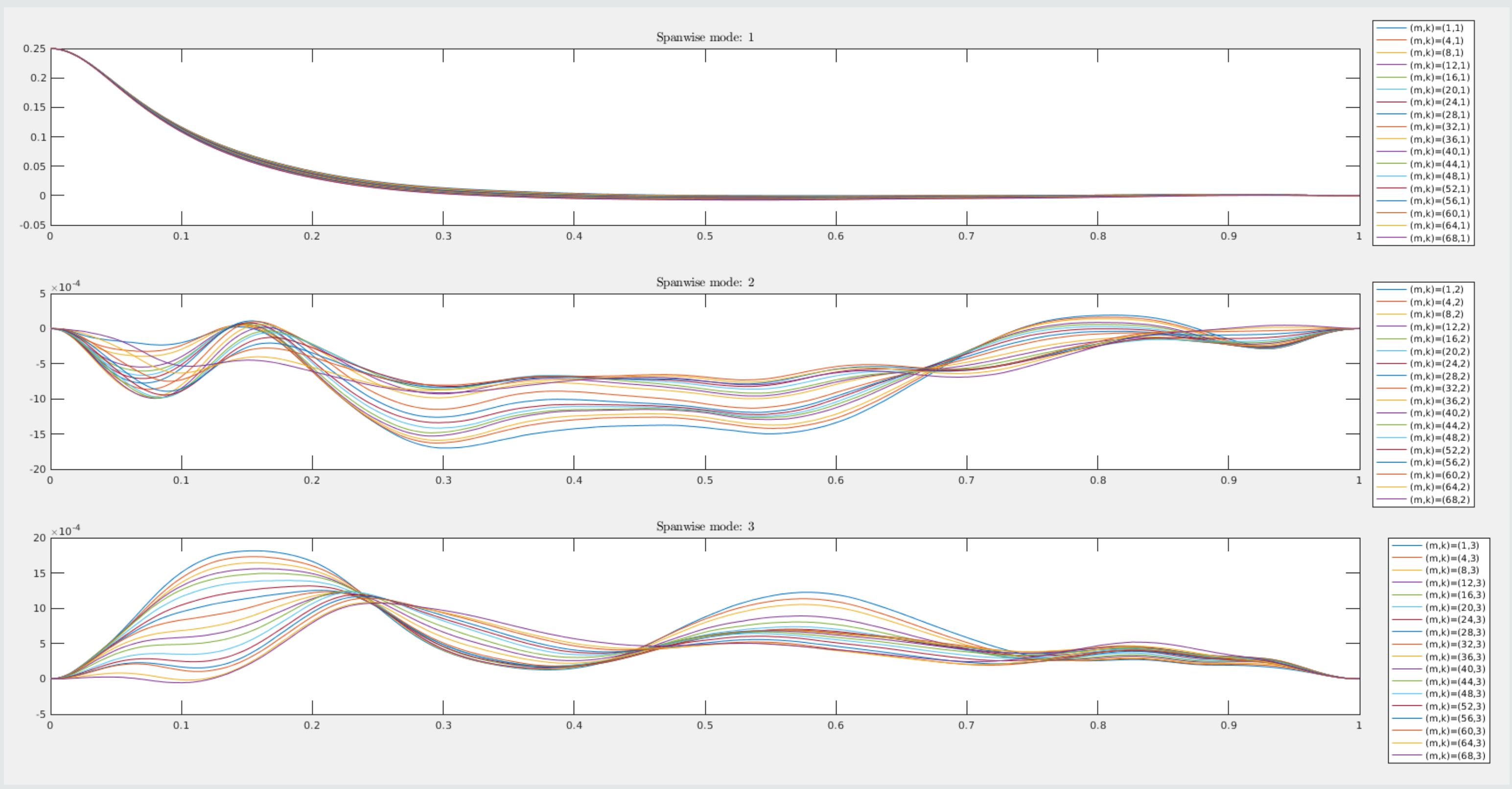
Definition (Procedure)

- the master branch code follows procedure:
  1. take fft azimuthally
  2. take 'rcorr' (xcorr but with sqrt-r) modification
  3. take fourier x-dir
  4. average in t with trapz()

Definition (Alternative Procedure)

- An alternative code follows swapped steps 1/2 procedure; this was tried to minimize code run time. That was shown to give equivalent results as the master branch code. However, we don't use this.
  1. take 'rcorr' (xcorr but with sqrt-r) modification
  2. take fft azimuthally
  3. take fourier x-dir
  4. average in t with trapz()

Definition (Example of Small Run)



More Graphs

- The following graph is the same run as the previous page, more k -modes are shown.

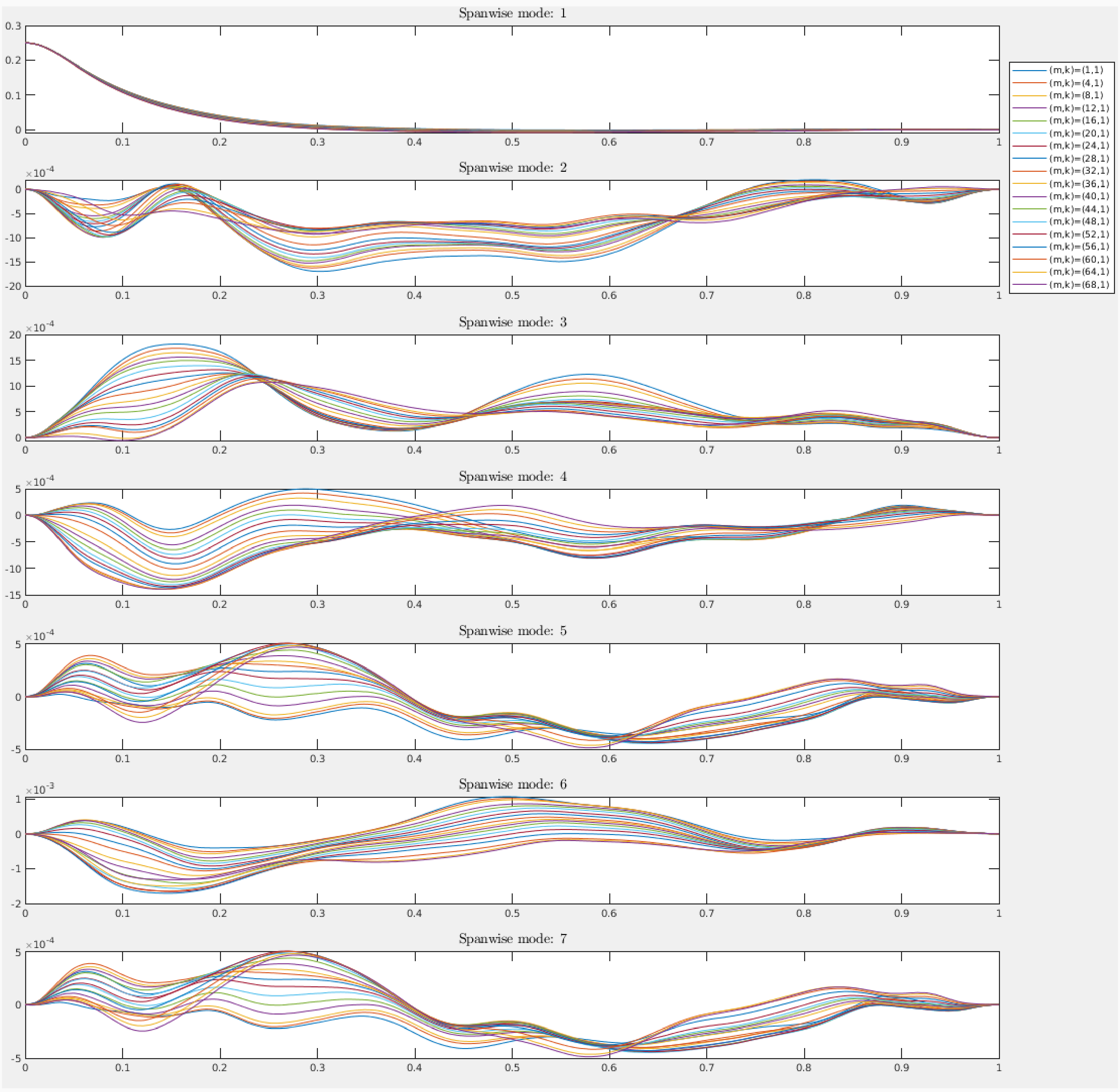


Figure 1: **fft** first 7 spanwise (x-dir or k) modes; each graph shows first $m \in \{1, 4, 8, 12, 16, \ldots, 64, 68\}$ (azimuthal) modes. (Run is 10 crosssections, 600 timesteps)

Procedure(s) xcorr/fft(theta)/fft(x)

```
+ There are two procedures (code branches) which are parallelized:
Branch A. xcorr->fft(x)->fft(theta)
Branch B. fft(theta)->xcorr->fft(x)

+ Branch A and B produce equivalent graphs.

+ We are using xcorr rather than

    $XX^*$    (*)

because we were able to get smooth data with that, and unable so far
to get smooth data with the equivalent (*)
```

Procedure and Corresponding Intermediate Data

Definition (1. Step by Step)

- **Big Idea:** For the FFT part of the code, all intermediate data is written to disk, in timeblocks, in order to allow for large timestep and cross-section parameters. Note that disk writing is absolutely necessary for large timesteps — this cannot be handled exclusively in ram.
- **The code follows procedure** using the timebloc workflow:
  - **(0)** compute velocity fluctuation v'
  - **(1)** compute correlation
  - **(2)** compute fft in x dir
  - **(3)** fourier in θ dir
  - **(4)** avgTimeEnd, which is the end result of the fft
  - **Intermediate Saving Data in .mat files.** is saved for each step as a `*.mat` file. This directory is in `constants.m`.
    - eg, `/mnt/archLv/mike/podData/apr18/`
    - this directory contains intermediate calculations for steps (0)-(4)
    - files are split into TimeBloc as they are processed to avoid out of memory issues.

- **Example Directory Containing Intermediate data files:**

```
/mnt/archLv/mike/podData/apr18:
total used in directory 296M available 475.1 GiB
-rw-rw-r-- 1 mi mi 323K May 12 19:37 avgTimeEnd[Case]C2T4[crossSec]2.mat
-rw-rw-r-- 1 mi mi  20M May 13 11:12 qMinusQbar[Case]C2T4[crossSec]1[TimeBloc]1.mat
-rw-rw-r-- 1 mi mi  20M May 12 19:36 qMinusQbar[Case]C2T4[crossSec]1[TimeBloc]2.mat
-rw-rw-r-- 1 mi mi  20M May 12 19:36 qMinusQbar[Case]C2T4[crossSec]2[TimeBloc]1.mat
-rw-rw-r-- 1 mi mi  20M May 12 19:36 qMinusQbar[Case]C2T4[crossSec]2[TimeBloc]2.mat
-rw-rw-r-- 1 mi mi  38M May 12 19:36 xcorrDone[Case]C2T4[crossSec]1[TimeBloc]1.mat
-rw-rw-r-- 1 mi mi  38M May 12 19:36 xcorrDone[Case]C2T4[crossSec]1[TimeBloc]2.mat
-rw-rw-r-- 1 mi mi  38M May 12 19:36 xcorrDone[Case]C2T4[crossSec]2[TimeBloc]1.mat
-rw-rw-r-- 1 mi mi  38M May 12 19:36 xcorrDone[Case]C2T4[crossSec]2[TimeBloc]2.mat
-rw-rw-r-- 1 mi mi  35M May 12 19:36 xdirPostFft[Case]C2T4[crossSec]2[TimeBloc]1.mat
-rw-rw-r-- 1 mi mi  35M May 12 19:37 xdirPostFft[Case]C2T4[crossSec]2[TimeBloc]2.mat
```

- Procedure Code: findAzimuthalModes3.m
  - 1) take fft in azimuthal
  - 2) correlate
  - 3) fourier in xDir
- Step 1 — fft azimuthally

```
% File: findAzimuthalModes3.m
function [qq]=findAzimuthalModes3(currentTime, currentCrossSec,
↪  qMinusQbar_noCsYet,xcorrDone,aliasStr)
  [ntimesteps, rMin, rMax, ss, ncs, plotOn, azimuthalSet ,azimuthalSetSize ,printStatus ,lags,
↪  blocLength, saveDir]=constants();
  [postAzimuthFft_noCsYet]=initData2("postAzimuthFft_noCsYet");
if aliasStr=="noAlias"
elseif aliasStr=="alias"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
% begin azimuthal -> NEW and Needs a little changing (came after xcorr)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
for timeBloc = 1:blocLength% time
    parfor t = 1:ntimesteps % time % parfor
        for  r = 1:540
            vec = zeros(1080,1);
            vec2 = zeros(1080,1);
            for zz=1:1080 % there are currently 1080 azimuthal modes.
            aa=qMinusQbar_noCsYet(t).circle(zz).dat(r,1); % this can perhaps be truncated to
↪  540, then duplicated for the second half, too prevent aliasing.!
            vec(zz)= aa;
            end % for zz
            aa=fft(vec);
            bb = flip(aa);
            cc = zeros(1080,1);
            for i=1:540
              cc(i) =aa(i);
              cc(1080 - i + 1 ) = aa(i); % get all 1080
            end % i
            postAzimuthFft_noCsYet(t).circle(1,r).dat=cc;
        end % r...
    end % parfor t
clear qMinusQbar_noCsYet; % yes, clear this..
```

- Step 2 — correlate

```
        saveStr=['/mnt/archLv/mike/podData/apr18/qMinusQbar[Case]C' num2str(ncs) 'T'
↪  num2str(ntimesteps) '[crossSec]' num2str(currentCrossSec) '[TimeBloc]' num2str(timeBloc)
↪  '.mat'       ];
        load(saveStr,'qMinusQbar_noCsYet');
    ordStr="xcorrNow";
        parfor t=1:ntimesteps%
            vec = zeros(1,540); % collect radial points..
                for m=1:1080
                for r=1:540% size xcorr
                  aa = qMinusQbar_noCsYet(t).circle(m).dat(r,1);
                  vec(r) = aa;
                end % r
                  [bb, lags] = rcorr(vec,"normalized"); % bb is 1079 because of xcorr ! <- new
↪  annotat.
                xcorrDone(t).circle(m).dat=bb';
                end % m
        end % (little)t
  saveStr=['/mnt/archLv/mike/podData/apr18/xcorrDone[Case]C' num2str(ncs) 'T'
↪  num2str(ntimesteps) '[crossSec]' num2str(currentCrossSec) '[TimeBloc]' num2str(timeBloc)
↪  '.mat'       ];
   save(saveStr,'xcorrDone','-v7.3');
end % timeBloc % end timebloc here .. (updated order..)
qq = xcorrDone; % asign qq and exi
end % fc
```

Procedure Code: fft in x

- Procedure Code:
- Step 3 — fft in x

```
% begin fft x-dir % file fftStep.m
parfor t=1:ntimesteps
for r=1:1079
for m=1:azimuthalSetSize
  aa = xdirNew(t).RadialCircle(r).azimuth(m).dat;
  %ab = fft(aa(end/2:end));
  ab = fft(aa);
  xdirPostFft(t).RadialCircle(r).azimuth(m).dat = ab;
  %hold on;
  %plot(real(ab));
  %pause(0.1)
end % m
end % r
end % t (little)
        sprintf('%s','saving xdirPostFft...')
        saveStr=['/mnt/archLv/mike/podData/apr18/xdirPostFft[Case]C' num2str(ncs) 'T'
 ↪  num2str(ntimesteps) '[crossSec]' num2str(c) '[TimeBloc]' num2str(timeBloc) '.mat'       ];
        save(saveStr,'xdirPostFft','-v7.3');
        sprintf('%s%s','Saved xdirpostfft into file ',saveStr);
```

- Step 4 — time averaging

```
% Time Averaging % file fftStep.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
aMat = zeros(1079,1);
for t=1:ntimesteps
for c=1:ncs
for m=1:azimuthalSetSize
for r=1:1079
   aa = xdirPostFft(t).RadialCircle(r).azimuth(m).dat(c,1);
   aMat(r) = aa;
end % r
if t<ntimesteps || timeBloc ~= blocLength
    avgTimeEnd(c).circle(m).dat = avgTimeEnd(c).circle(m).dat + aMat;
elseif t==ntimesteps && timeBloc == blocLength
   avgTimeEnd(c).circle(m).dat = (avgTimeEnd(c).circle(m).dat + aMat)/(ntimesteps*blocLength);
end % if
end % m
end % c
end % t (little)
end % timeBloc
```

Procedure Code: Modified xcorr.m (radial version)

- Procedure Code: Modified xcorr.m (radial version)
- We have modified the built in matlab function xcorr.m so its pre and most multiplied by $r^{1/2}$ and $r'^{1/2}$.

```matlab
function c = crosscorr(x,y,maxlag)
% Compute cross-correlation for vector inputs. Output is clipped based on
% maxlag but not padded if maxlag >= max(size(x,1),size(y,1)).
nx = numel(x);
ny = numel(y);
m = max(nx,ny);
maxlagDefault = m-1;
mxl = min(maxlag,maxlagDefault);

if any(~isfinite(x),'all') || any(~isfinite(y),'all')
    c1 = conv(x,conj(flip(y)));
    if mxl <= maxlagDefault
        % Clip if maxlag is small. This would be a no-op for larger maxlag.
        % Account for 0 lag corresponding to c1(ny).
        c1 = c1(max(1,ny-mxl):(ny+min(mxl,nx-1)));
    end
    % Pad the head or tail if nx >= ny or nx < ny, respectively.
    % Note that we may need to both clip and pad: rcorr(1:5,1:3,3).
    c = zeros(2*mxl+1,1,'like',c1);
    offset = max(0,mxl-ny+1);
    c(offset+(1:numel(c1))) = c1;
else
    m2 = findTransformLength(m);
    %X = fft(x,m2,1);
    %Y = fft(y,m2,1);

    % stretch this by root-radial amount

    for ii = 1:length(x)
      x(ii) = x(ii) * sqrt(ii); % modified bit (raba)
      y(ii) = y(ii) * sqrt(ii);
    end % for
    X = fft(x,m2,1);
    Y = fft(y,m2,1);

    if isreal(x) && isreal(y)
        c1 = ifft(X.*conj(Y),[],1,'symmetric');
    else
        c1 = ifft(X.*conj(Y),[],1);
    end
    % Keep only the lags we want and move negative lags before positive
    % lags.
    c = [c1(m2 - mxl + (1:mxl)); c1(1:mxl+1)];
end
```

Depricated Procedure B Graphs

- 'old' procedure graph; I put this here for completeness.
  - code in `file:///mnt/archLv/mike/podTimeCoefficientCopy/streamFft.m`
  - this is not parallel code and is not considered main branch code.

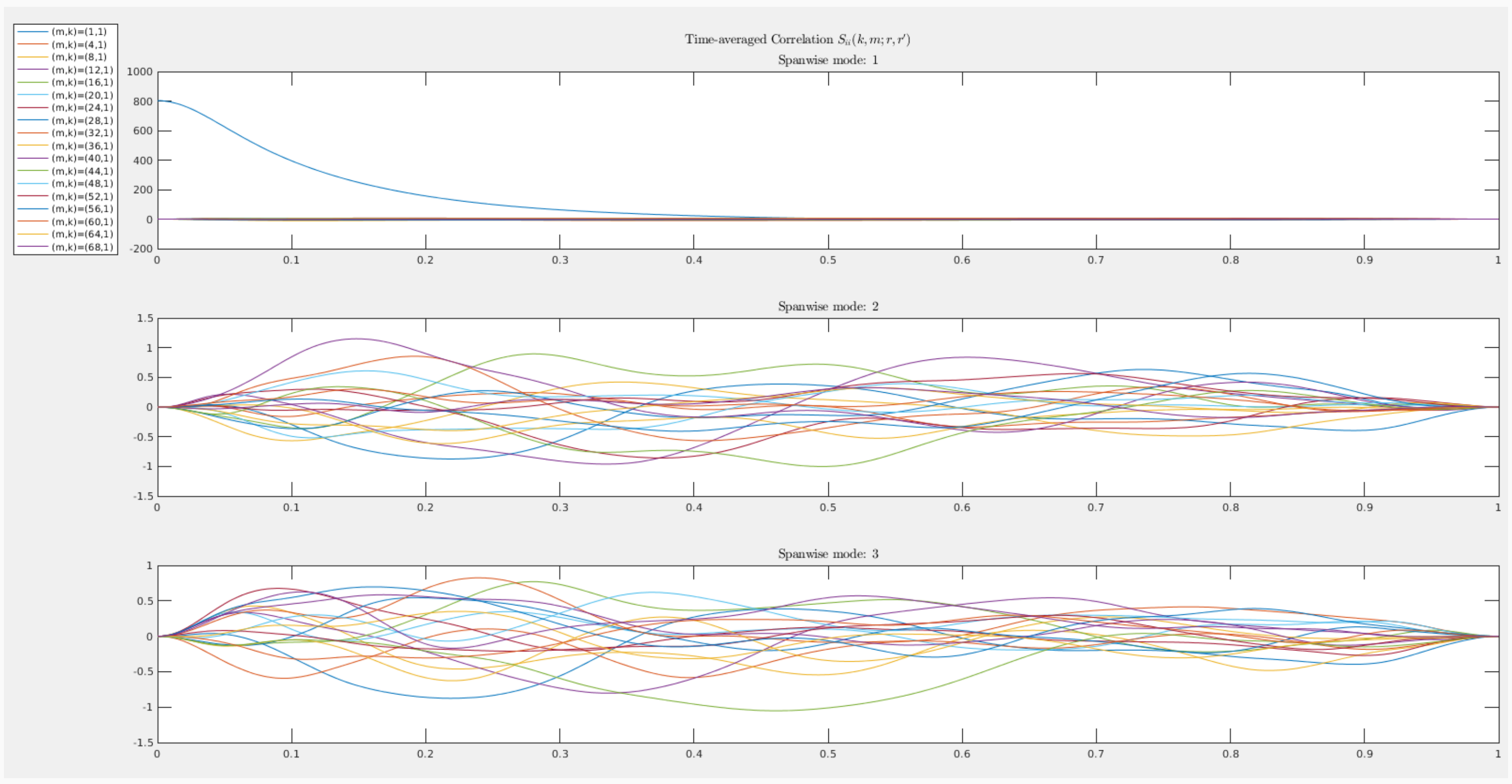


Figure 2: result of code **streamFft.m** branch; 3 cs, 80 ts

Code Flow Step by Step

Definition (1. File constants.m)

- Parameter `ntimesteps` and `blocLength`
  - eg if ntimesteps=200 and blocLength=3, then the total number effect timesteps is 600.
- `ncs` number of crosssections
- main file `xcorrGpu`. loads in

Definition (2. xcorrGpu.m - main() driver function)

- calls functions, most importantly fftStep, which is is the main driver for most functions.
- using (L)oad or (g)enerate velocity fluctuations

```
>> xcorrGpu
(L)oad or (G)enerate? If load eg c2t4
>  c200t400
```

- the .mat file consisting of 200 crosssections and 400 total timesteps is loaded into memory.
- If (G) selected, then parameters in `constants.m` are used.

Definition (3. fftStep - main driver function)

- This function is responsible for the following tasks, which are executed in the order of crossection and then in timeBlocs:
- A. Initializes Data $\rightsquigarrow$ calls `initData2.m`
- B. Reading in crosssection data (extracted from vIsit) $\rightsquigarrow$ calls `readCircles2.m`
- C. Finding velocity fluctuations v'
  - a. Find average $\bar{q}$
  - b. Find the fluctuation $q - \bar{q}$
  - c. Fluctuation is save to disk under `./saveDir/qMinutsQbar`
- D. `findAzimuthalModes2.m` $\rightsquigarrow$
  - a. Find correlation
  - b. Find azimuthal fft
  - c. Result (B.b) saved the result to disk.
- E. Find fft in spanwise direction
- F. Find time average using trapz
  - a. Time average is saved to disk under `./saveDir/avgTimeEnd`

Order of Operations is Indepdent

Definition (Order of Operations is Indepdent)

- the order of the following gives the same end-graph:
    - (a) correlate in $r \rightsquigarrow$ fft in θ
    - (b) fft in $\theta \rightsquigarrow$ correlate in r

    The following are equivalent graphs for runs of (a) and (b)

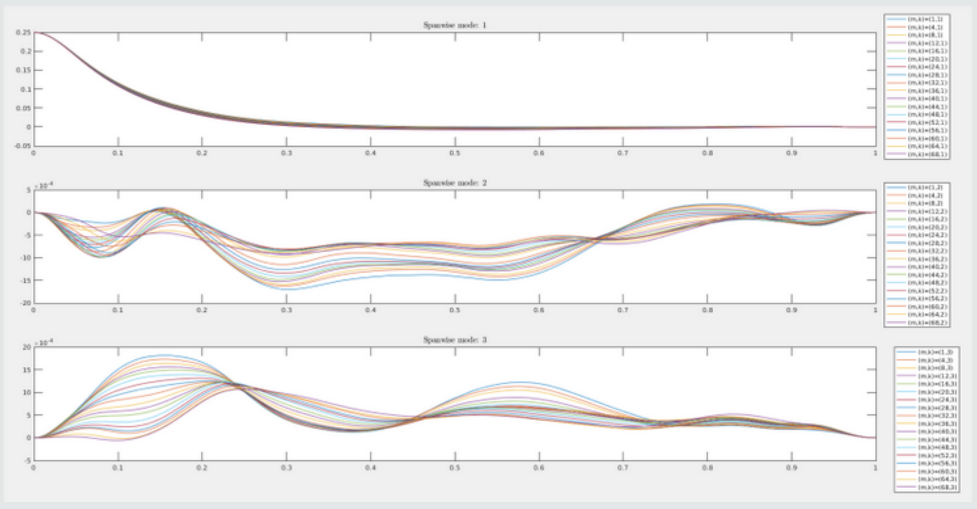


Figure 3: Order of Operations is Independent: Order of correlation and fft in different directions gives same end graphs.

Useful Reference

Definition (Useful Reference:)

- Reference: Gudunev Thesis `https://www3.nd.edu/~sgordeye/thesis.pdf` , page 28 (41 pdf page)
- Explicit POD mode integration weighting is given eg by,

$$\mathsf{W} = \{\mathsf{w}, \dots, \mathsf{w}\}, \quad \mathsf{w} = (0.5, \overbrace{1, 1, \dots, 1}^{M-2}, 0.5)^T\text{-trapezoidal weighting } M\text{-vector,} \tag{1}$$

- for

$$\Phi_{\alpha\beta}\mathsf{W}\varphi_\beta = \frac{\lambda}{\Delta y}\varphi_\alpha \tag{2}$$

- or
- Note, that is a different (duct) geometry

* Questions to ask Bijay

Definition (Questions to ask Bijay)

- Q1 do we need full range of timesteps in order to capture the statistically steady turublent (mean) flow?
  - how can we check we have enough timesteps $[t_0, t_1, \dots, t_n]$ where n is sufficiently large?
- Q2 How to capture fourier series coefficients c_k from DFT? Seems to be a lot of confusion about this
  - Definitely 'not' the same thing !
- Q3 how deal with out of memory issue
- Q4 is correlating at different steps $\rightsquigarrow$ lead to different result? according to hellstom2016.eq.(2.2),

$$\mathsf{S}\left(k; m; r, r'\right) = \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau \mathbf{u}(k; m; r, t)\mathbf{u}^*\left(k; m; r', t\right)\mathrm{d}t \tag{3}$$

- ie, need to correlate only after fourier-ing (both directions)! $\rightsquigarrow$ and then average in t.
- Q5 the POD in hellstrom2016 is a 3-component vectorial one;
  - I know how to compute each correlation S_{ij} separate, but not sure how all 3 directions would be included all at once...
- Q7 can we have negative wavenumbers? That would happen for the cross-spectral (ie not autospectral).
  - in that case how display? Is it same by symmetry.

$$\Phi_x^{(n)}(k; m; r) = \Phi_x^{(n)}(k; -m; r) = \Phi_x^{(n)^*}(-k; m; r) \tag{Smits2016 Equation after (2.5)}$$

- would we actually have to invoke this or is that passive observation?
- Q8 h2016 sez,
- does this mean that streamwise mode number's not significant? -> can we just do for 1 cross-section and expect same behavior? (surely not, but what does the above mean?)
- (streamwise mode number fourier of crosssecions)

Progress

Definition (Done:)

- **I get Smooth signal (!)** using:
  - Step (1). xcorr to compute correlation matrix
  - Step (2) fft(x-dir)
  - Step (3) fourier$_{\text{Series}}$ in azimuthal-dir
  - hellstrom2016 uses this routine (see eq 2.2)
  - then averages in time.

Definition (Fail:)

- Could not do non-xcorr way (tutkin, cb approach) yet
  - problem: signal looks very non-smooth
    - issue with improper windowing (probably)
    - Upshot: xcorr method and cb (multiplication instead of xcorr), tutkin should be mathematically equivalent
    - same computational order in correlation step
    - tutkin, cb: $\mathcal{O}(n)$ for multiplication
    - 'intuitive' approach: $\mathcal{O}(nlog(n))$ for fft-ifft

Definition (In Progress)

- Parallize code (semi-done, this can be improved)
  - easy way is to use parfor loops
  - other finer control using MPI calls
    - For now, use **matlab coder**
    - this converts matlab to c++
    - uses MPI to do parallel
- Simple Test Case for fourier transform
- Test for getting correct Fourier Series from the DFT

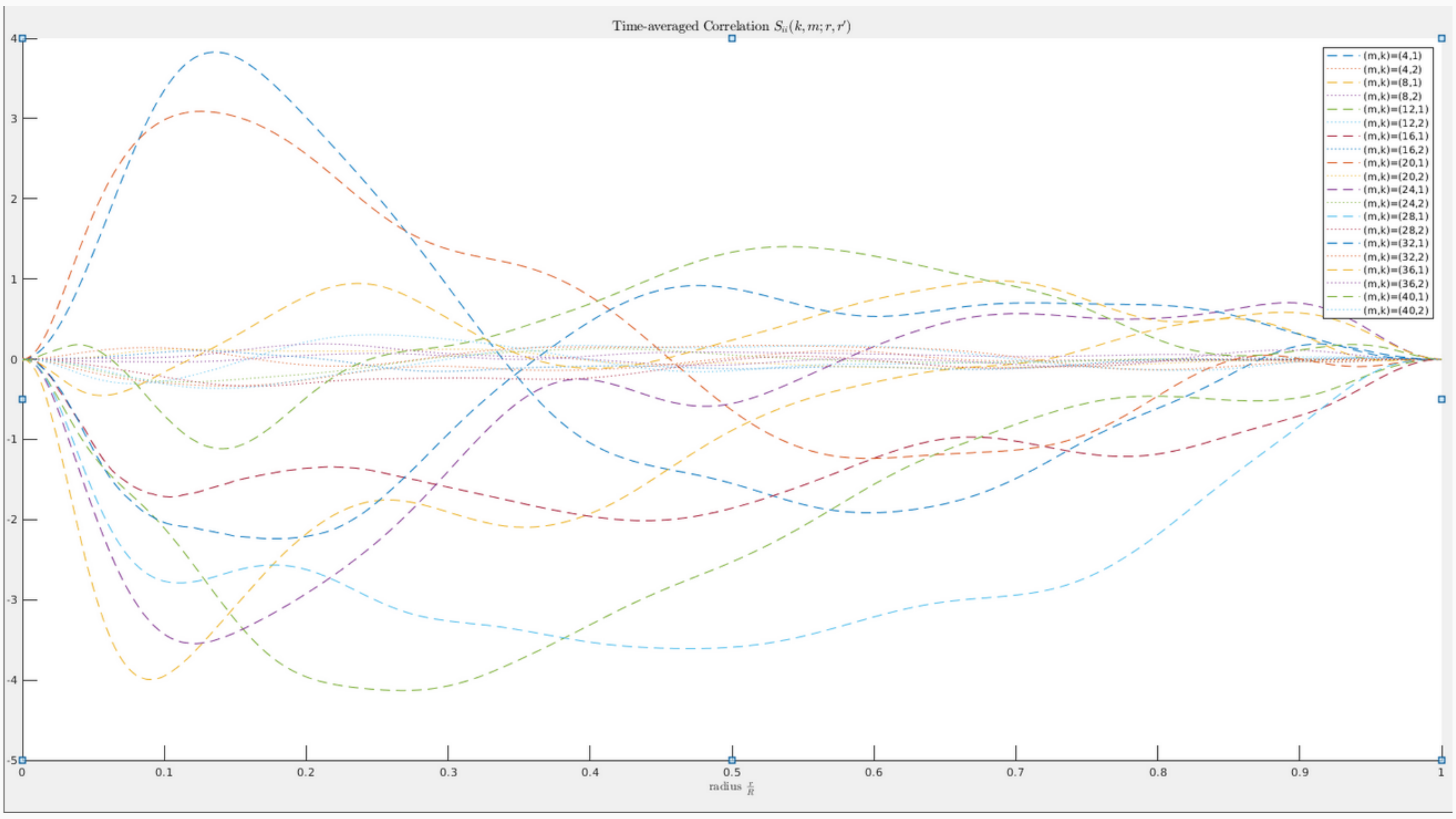


Figure 4: xcorr $\rightsquigarrow$ (x-fft) $\rightsquigarrow$ (azimuth-fft) Approach.

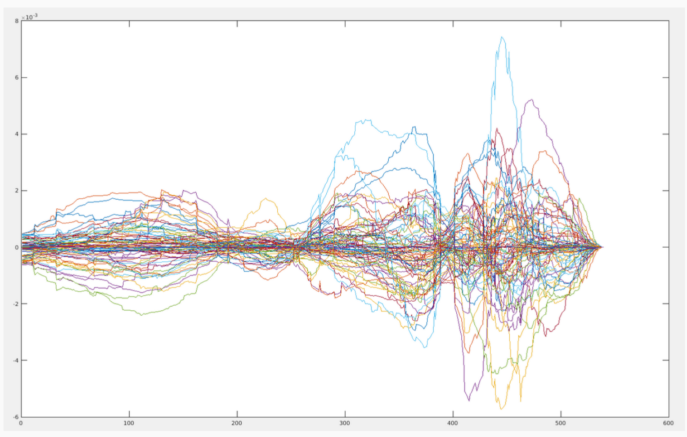


Figure 5: Unsightly signal using **multiplcation approach**, probably just needs hamming filter etc

3 Approaches

Definition (3 Approaches)

- Approach (i) Intuitive
- (1) use matlab's xcorr,
- (2) then fft in x
- (3) then fft in azimuthally
- **Outcome 1:** I get a symmetric graph

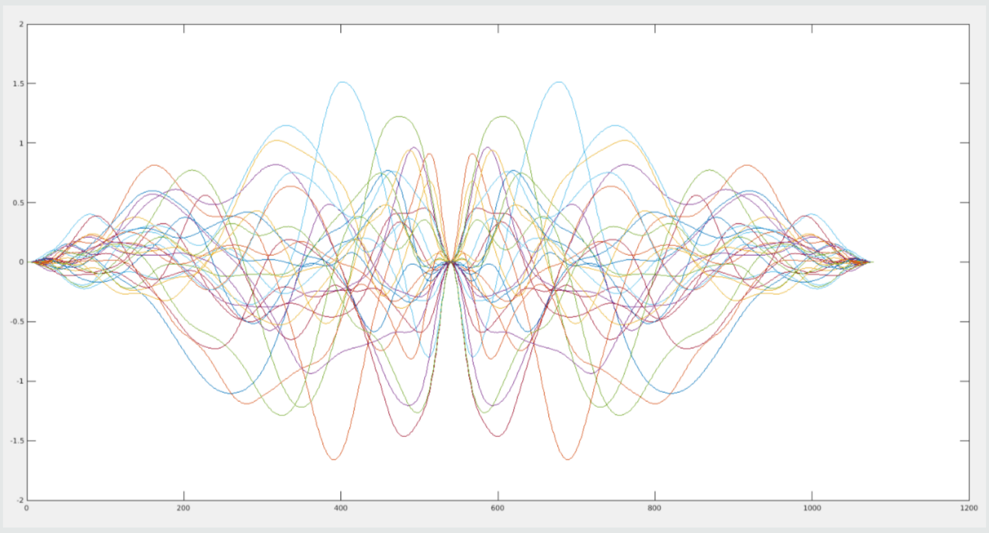


Figure 6: graph of hellstrom2016.eq.(2.2) — time averaged autospectral signal $S(k, m; r, r')$ — before POD

- **Outcome 2:** this is very quick (fft is $\mathcal{O}(nlog(n)$ . . .
- Approach (ii) - follow Tutkin Paper to the letter
- (1) Fourier in time — nb our case is statistically steady; or just fourier in x
- (2) just *calculate product of fourier transformed signals in tutkin.eq(9)
  - do not do lags — see explanation on next page on why we do not compute lags
- (3) calculate fourier **Series** (not fft).
  - **Outcome 2:** this is very slow, lot of full matrix matrix mult (maybe half matrix mult): $\mathcal{O}(N^2)$ . . .
- **Approach (iii) - CB Approach** — just like (ii) but optimize that by:
- fourier in θ
- form covariance
- fourier streamwise

- Code for Approach (i)

```
if fftCase=="A" % xcorr method: (1) xcorr (2) fft-x (3) fft-azimuthal

[radialXcorrStruct]=fftStreamwise(qMinusQbar, radialXcorrStruct,"populateXcorr")
[XcorrData,lags]=DoXcorrRadial(radialXcorrStruct,XcorrData,"none"); %none, graph, graphPause
[XcorrDonePreCsFft]=DoFftXcorr(XcorrData,XcorrDonePreCsFft,"setupVec","xDir","off",lags)
% Call fft() in x dir;
[XcorrDonePostCsFft]=DoFftXcorr(XcorrDonePreCsFft,XcorrDonePostCsFft,"runFft","xDir","off",lags)
[XcorrDoneCsFftDonePreAzimFft]=DoFftXcorr(XcorrDonePostCsFft,XcorrDoneCsFftDonePreAzimFft
 ↪  ,"setupVec","azimDir","off",lags)
% Call fft in azimuthal dir θ; do not calculate
% $S_{i,j}(r,r';m;f) = \frac{1}{2\pi}\sum_{m=0}^{N} \tilde{S}_{i,j}(r,r';\Delta\theta;f)\, e^{-im\Delta\theta} \mathrm{d}(\Delta\theta)$
[XcorrDoneCsFftDonePostAzimFft]=DoFftXcorr(XcorrDoneCsFftDonePreAzimFft,XcorrDoneCsFftDonePostA
 ↪  zimFft,"runFft","azimDirCaseA","off",lags)
% * Then Follow hellstrom2016.equation.2.2,
% time-average (in integral sense); we call trapz() for this.
% eg find $\frac{1}{T}\int_0^T S(k,m,t;r,r')\,dt$
[XcorrAzimDonePreTimeAvg]=DoFftXcorr(XcorrDoneCsFftDonePostAzimFft,XcorrAzimDonePreTimeAvg,"set
 ↪  upVec","timeDir","off",lags)
[XcorrAzimDonePostTimeAvg]=DoFftXcorr(XcorrAzimDonePreTimeAvg  ,XcorrAzimDonePostTimeAvg
 ↪  ,"runFft","timeDir","off",lags)
[Skmr]=DoFftXcorr(XcorrAzimDonePostTimeAvg,Skmr,"setupVec","Skmr","off",lags)
plotSkmr(Skmr,"none")
sprintf('%s','pause');
```

Justification for Just Multiplying — No xcorr() use

Definition (Problem/confusion with the Multiply approach)

- Approach: Just Multiply. No Xcorr.
- I would like to do this method; seems easier; but can't get symmetric autocorrelation yet. Does this require lagging?
- From the explanation below, it seems the cross-spectral is just a straight multiply . . . * Justification for Just Multiplying — No xcorr() use

Definition (Justification for Just Multiplying — No xcorr() use)

- Just as the Power Spectral Density (PSD) is the Fourier transform of the autocovariance function we may define the Cross Spectral Density (CSD) as the Fourier transform of the cross-covariance function

$$P_{12}(w) = \sum_{n=-\infty}^{\infty} \sigma_{x_1 x_2}(n) \exp(-iwn) \tag{4}$$

- Note that if $x_1 = x_2$, the CSD reduces to the PSD. Now, the cross-covariance of a signal is given by

$$\sigma_{x_1 x_2}(n) = \sum^{\infty} x_1(l) x_2(l-n) \tag{5}$$

- Substituting this into the earlier expression gives

$$P_{12}(w) = \sum_{n=\infty}^{\infty} \sum_{l=-\infty}^{\infty} x_1(l) x_2(l-n) \exp(-iwn) \tag{6}$$

- By noting that

$$\exp(-iwn) = \exp(-iwl) \exp(iwk) \tag{7}$$

where $k = l - n$ we can see that the CSD splits into the product of two integrals,

$$P_{12}(w) = X_1(w) X_2(-w) \tag{8}$$

where

$$X_1(w) = \sum_{l=-\infty}^{\infty} x_1(l) \exp(-iwl) \tag{9}$$

$$X_2(-w) = \sum_{k=\infty}^{\infty} x_2(k) \exp(+iwk) \tag{10}$$

- For real signals $X_2^*(w) = X_2(-w)$ where $*$ denotes the complex conjugate the cross spectral density is given by

$$P_{12}(w) = X_1(w) X_2^*(w) \tag{11}$$

- When take the cross-correlation, equivalent to:
- $\rightsquigarrow$ take a bunch of shifts in the input space/time and then add them up.
- When take a Fourier transform of a signal:
- $\rightsquigarrow$ take a bunch of frequency filters and then add them up.
- In both, **sum over the entire space of possible shifts**.
- Simply taking the Fourier transform of a signal is already summing over a bunch of shifts $\rightsquigarrow$ that's why it looks like there are no shifts, it's hiding in the Fourier transform.
- The convolution theorem and its proof gives details.
- **difference between cross-correlation and convolution is just a flip in the signal (a minus sign on the shift).**

Computing a DFT with the FFT

- **Task: read fourier (series) coefficients off the DFT !**
- We need to compute fourier series in the periodic (azimuth) direction.
- Note, fourier series coefficients are not the same thing as DFT !
  - However we can use of DFT to find c_k fourier-coeffients

Definition (Approach 1: use Poisson Formula ; involves differencing the DFT values.)

- Define

$$\begin{aligned}\Delta x &= A/N \\ x_n &= n\Delta x \text{ for } -N/2 < n < N/2. \\ f_n &= f(x_n)\end{aligned}$$

- DFT of the sequence $\{f_n\}$ is defined in [1] as the sequence $\{F_k\}$ where

$$F_k = \frac{1}{N}\sum_{n=-N/2+1}^{N/2} f_n \exp(-2\pi i n k/N) \tag{12}$$

- Supposing the function f sampled has the fourier series,

$$f(x) = \sum_{k=-\infty}^{\infty} c_k \exp(2\pi i k x/A) \tag{13}$$

- Then, the **Fourier coefficients c_k, as related to the DFT**, are

$$F_k = c_k + \sum_{j=-\infty}^{\infty} (c_{k+jN} - c_{k-jN}) \tag{14}$$

- This means that the c_k simply equal the F_k if f has no frequency components higher than $N/2$ Hz because all the terms in the infinite sum above are zero. That is, if f is bandlimited and we have sampled at a frequency higher than the Nyquist frequency, then we can simply read the Fourier coefficients off the DFT.
- However, when f is not band-limited, or when it is band-limited but we have not sampled it finely enough, we will have aliasing. Our Fourier coefficients will differ from our DFT output by an error term involving high-frequency Fourier series coefficients.
- In application it's usually too much to hope that your function is exactly band-limited, but it may be approximately band-limited. The Fourier coefficients of smooth functions eventually decay rapidly (see details here) and so the error in approximating Fourier series coefficients by DFT terms is small.

Definition (*Approach 2: Reconstruct fourier-coefficients from DFT with formula)

$$\begin{aligned}X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \\ &= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N}kn\right) - i\cdot\sin\left(\frac{2\pi}{N}kn\right)\right]\end{aligned}$$

- Code for this,

```python
from numpy.fft import fft
x = array([ 0.,  1.,  0.,  1.])
y = fft(x)

#first rescale it
nfft = len(x)
y /= nfft

n = arange(0,4)
# notice that y[1] and y[3] are identically zero:
x_reconstructed = y[0] +y[2] * cos(2*2*pi/nfft*n)
```

Two approaches and Code doTutkinEq9innerProduct()

Definition (Two Approaches!)

- The above derivation means that the CSD can be evaluated in one of two ways!
- Approach (i). by first estimating the cross-covariance and Fourier transforming or
- Approach (ii). by taking the Fourier transforms of each signal and multiplying (after taking the conjugate of one of them).
- A number of algorithms exist which enhance the spectral estimation ability of each method. These algorithms are basically extensions of the algorithms for PSD estimation, for example, for type (i) methods we can perform Blackman-Tukey windowing of the cross-covariance function and for type (ii) methods we can employ Welch's algorithm for averaging modified periodograms before multiplying the transforms. See Carter [8] for more details.

- Code doTutkinEq9innerProduct() follows Approach (ii)

```
%% calculate spatial correlation matrix
% c1^^I=^^IU*U';
function [Stilda]=doTutkinEq9innerProduct(t,c,vecTutkunEq8,Stilda,deltaTheta)
  % find difference AND take fft.
[ntimesteps rMin rMax ss ncs plotOn azimuthalSet azimuthalSetSize]=constants();
  azimuthalSetMinusOne = azimuthalSet;
  azimuthalSetMinusOne(end) =[]; % chop off end element 1079. Ends at 1050
%for tt=1:ntimesteps
  azimuthCounter=1;
%for jj=1:azimuthalSetSize - 1 % this is Delta s.
jj=1;


if deltaTheta == 1
  deltaThetaFix = 0;
else
  deltaThetaFix = deltaTheta;
end
% this should be a vector of time series, if temporal fft-ing to get frequency,
% or, if fft is taken on cross-sections, then it needs to be crosssection.
for ii=1:540
  vvv1(ii) = vecTutkunEq8(t).RadialCircle(ii).azimuth(jj).dat(1,c);

  vvv2(ii) = vecTutkunEq8(t).RadialCircle(ii).azimuth(jj+deltaThetaFix).dat(1,c);
end % ii
%aa=xcorr(vvv1,vvv2,"normalized");
aa=correlateVecs(t,c,vvv1,vvv2);
hold on;
plot(aa)
%aa=xcorr(vvv1,vvv2);
for ii=1:2*540-1
Stilda(t).azimuth(deltaTheta).dat(ii,1)=aa(ii);
%end % tt
azimuthCounter=azimuthCounter+1;
%end %jj
end %ii
%
end %f
```

Calculation of cross-spectra

Definition (Step 2 Redux: Cross Spectra Calculation)

- Direct Correlation — don't use

The discrete spatial correlation is,

$$R_{\mathrm{II}}(x,y) = \sum_{i=-K}^{K} \sum_{j=-L}^{L} I(i,j)I'(i+x,j+y). \tag{15}$$

- The variables I and I' are the samples as extracted from the pipe cross-sections.
- Efficient way using FFT

The alternative to calculating the cross-correlation directly using Eq. (5.1) is to take advantage of the correlation theorem which states that the cross-correlation of two functions is equivalent to a complex conjugate multiplication of their Fourier transforms:

theor(

where $\hat{I}$ and $\hat{I}'$ are the Fourier transforms of the functions I and I', respectively.

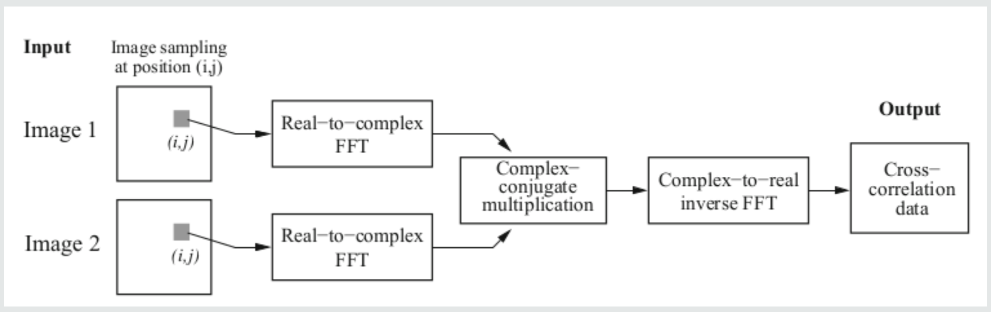


Figure 7: Implementation of cross-correlation using FFT's. We omit the last step of calculating $\mathcal{F}^{-1}$ since we want cross-spectra.

- Approach A.
- take fft in streamwise x direction $\rightsquigarrow$ wavenumber k
- form complex-conjugate multiplication
- form fft with azimuthal separation $\rightsquigarrow$ wavenumber m
- Approach B. (cb)
- form fft with azimuthal separation m
- form complex-conjugate multiplication
- take fft in streamwise x direction $\rightsquigarrow$ wavenumber k

References

Spectral Analysis

- Ta-Hsin Li. *Time series with mixed spectra*. CRC Press, 2013.
- W Penny. *Signal Processing Course*. discussion of autocovariance page 173. April 2000. URL: `https://www.fil.ion.ucl.ac.uk/~wpenny/course/array.pdf`.
- Robert H Shumway, David S Stoffer, and David S Stoffer. *Time series analysis and its applications*. Vol. 3. discussion of autocovariance page 173. Springer, 2000.

piv analysis

- Markus et al Raffel. *Particle Image Velocimetry, a practical guide*. 2018.
- Salvador B Rodriguez. *Applied Computational Fluid Dynamics and Turbulence Modeling*. Tech. rep. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2018.

Other Papers, which I found helpful for Understanding the scope and language of spectral/POD Anaylsis

- SEAN C. C. BAILEY et al. “Azimuthal structure of turbulence in high Reynolds number pipe flow”. In: *Journal of Fluid Mechanics* 615 (Nov. 2008), pp. 121–138. ISSN: 1469-7645. DOI: `10.1017/s0022112008003492`. URL: `http://dx.doi.org/10.1017/S0022112008003492`.
- Rahul Deshpande et al. “Two-dimensional cross-spectrum of the streamwise velocity in turbulent boundary layers”. In: *Journal of Fluid Mechanics* 890 (Mar. 2020). ISSN: 1469-7645. DOI: `10.1017/jfm.2020.139`. URL: `http://dx.doi.org/10.1017/jfm.2020.139`.
- J. G. M. Eggels et al. “Fully developed turbulent pipe flow: a comparison between direct numerical simulation and experiment”. In: *Journal of Fluid Mechanics* 268 (June 1994), pp. 175–210. ISSN: 1469-7645. DOI: `10.1017/s002211209400131x`. URL: `http://dx.doi.org/10.1017/S002211209400131X`.
- M. Hultmark et al. “Turbulent Pipe Flow at Extreme Reynolds Numbers”. In: *Physical Review Letters* 108.9 (Feb. 2012). ISSN: 1079-7114. DOI: `10.1103/physrevlett.108.094501`. URL: `http://dx.doi.org/10.1103/PhysRevLett.108.094501`.
- Roeland de Kat and Bharathram Ganapathisubramani. “Frequency–wavenumber mapping in turbulent shear flows”. In: *Journal of Fluid Mechanics* (2015). DOI: `10.1017/jfm.2015.558`. URL: `http://dx.doi.org/10.1017/jfm.2015.558`.
- Z. LIU, R. J. ADRIAN, and T. J. HANRATTY. “Large-scale modes of turbulent channel flow: transport and structure”. In: *Journal of Fluid Mechanics* 448 (Nov. 2001), pp. 53–80. ISSN: 1469-7645. DOI: `10.1017/s0022112001005808`. URL: `http://dx.doi.org/10.1017/S0022112001005808`.
- I. Marusic et al. “Wall-bounded turbulent flows at high Reynolds numbers: Recent advances and key issues”. In: *Physics of Fluids* 22.6 (June 2010). very good Review of key issues; discusses our POD pipe problem in the context of recent (2015) research efforts, p. 065103. ISSN: 1089-7666. DOI: `10.1063/1.3453711`. URL: `http://dx.doi.org/10.1063/1.3453711`.
- P. ORLANDI and M. FATICA. “Direct simulations of turbulent flow in a pipe rotating about its axis”. In: *Journal of Fluid Mechanics* 343 (July 1997), pp. 43–72. ISSN: 1469-7645. DOI: `10.1017/s0022112097005715`. URL: `http://dx.doi.org/10.1017/S0022112097005715`.
- P. Orlandi and M. Fatica. “Direct simulations of turbulent flow in a pipe rotating about its axis”. English. In: *Oceanographic Literature Review* 45.7 (1998), p. 1257.
- Hans A. Panofsky and Robert A. McCormick. “Properties of spectra of atmospheric turbulence at 100 metres”. In: *Quarterly Journal of the Royal Meteorological Society* (1954), pp. 546–564. DOI: `10.1002/qj.49708034604`. URL: `http://dx.doi.org/10.1002/qj.49708034604`.
- Xiaohua Wu, J. R. Baltzer, and R. J. Adrian. “Direct numerical simulation of a 30R long turbulent pipe flow at R+ = 685: large- and very large-scale motions”. In: *Journal of Fluid Mechanics* (). DOI: `10.1017/jfm.2012.81`. URL: `http://dx.doi.org/10.1017/jfm.2012.81`.
- MARK V. ZAGAROLA and ALEXANDER J. SMITS. “Mean-flow scaling of turbulent pipe flow”. In: *Journal of Fluid Mechanics* 373 (Oct. 1998), pp. 33–79. ISSN: 1469-7645. DOI: `10.1017/s0022112098002419`. URL: `http://dx.doi.org/10.1017/S0022112098002419`.

Blank Page

Blank Page. Older Presentations follow

2 Plans

Plan A: keep working on Code or Hybrid Approach (best)

- (A) **Need sampling frequency!** Since we are taking temporal FFT, *need $\Delta = t$ in order to use correct block size.
- Seems that it is about $t \leq 3$, $\because$ cannot get Wu-Moin with larger t.
- (B) **Redo Correlation Tensor** just do direct calculation
- (C) **Using (B), recreate Wu-Moin** before doing any POD analysis.
- gives check before doing pod calculations

Plan B: Just Adopt Other Code

- **DeepBlue umich POD**
- `https://deepblue.lib.umich.edu/bitstream/handle/2027.42/92348/POD_Code_Chen_et_al.pdf?sequence=1&isAllowed=y`
- **Spectral POD code by Oliver Schmidt**
- `https://flowphysics.ucsd.edu/index.php/spectral-proper-orthogonal-decomposition-matlab/`
- See if dr bailey has data for his 2009 postdoc code ?
- good code, just tricky to interpret without being able to step through with data

Wu-Moin Correlation Coefficient Graph

- Update

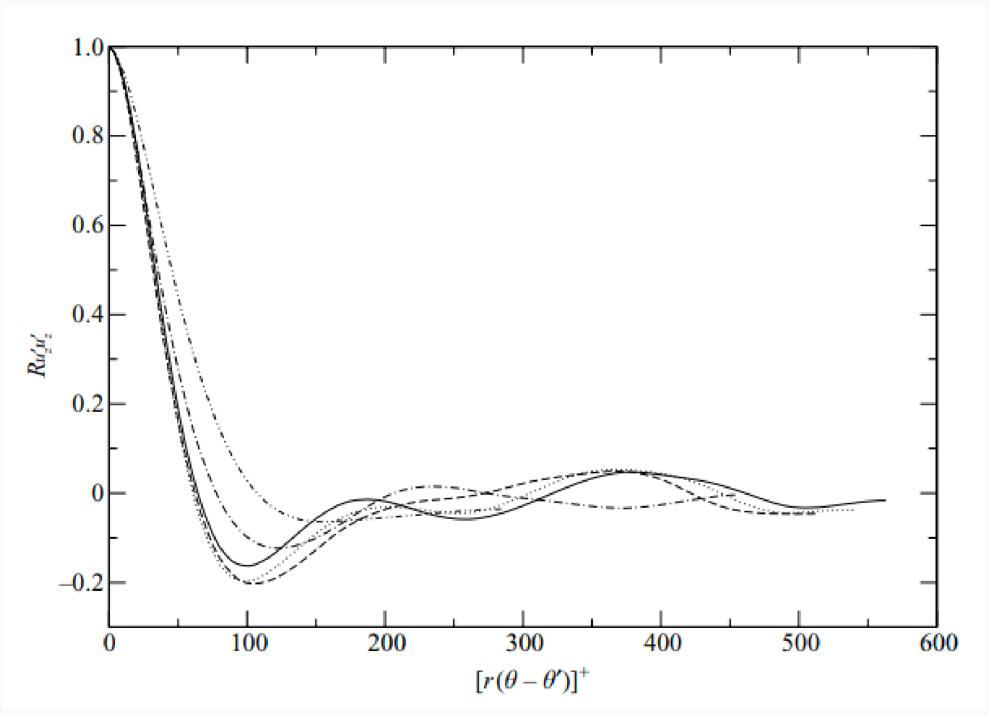
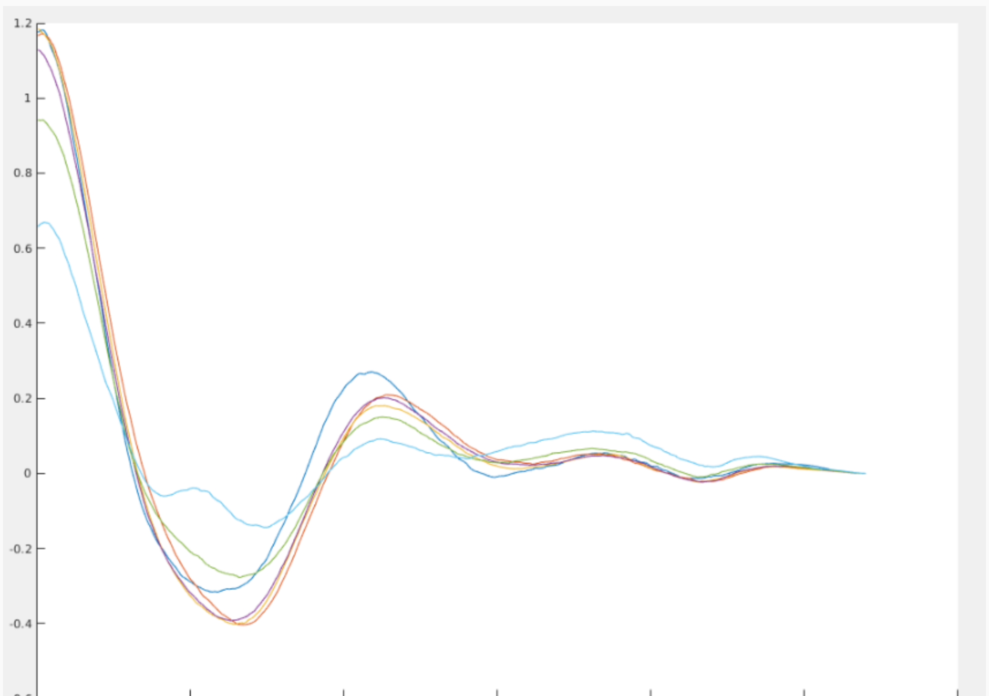


Figure 8: (Wu-Moin FIGURE 31.) Two-point correlation coefficient $R_{u_z' u_z'}$ at $Re_D = 5300$ as a function of local azimuthal separation $[r(\theta - \theta')]^+$. Solid line: $1 - r = 0.01$ $\quad ((1-r)^+ = 1.81)$; dotted line: $1 - r = 0.05$; dashed line: $1 - r = 0.1$; chain-dotted line: $1 - r = 0.2$; chain-dotted-dotted line: $1 - r = 0.5$.

- Shows azimuthal separation (ie $i, j = 3$ streamwise, nb, i, j do not vary)



- Correlation Matrix using Direct Calcuation

Re-doing Step 2: Azimuthal Cross-Spectra not using xcorr

Definition (Step 2: Azimuthal Cross-Spectra)

- Cross-spectra are computed and block averaging is performed as follows

$$\tilde{S}_{i,j}\left(r,r';\Delta\theta;f\right) = \frac{\left\langle \hat{u}_i(r,\theta,f)\hat{u}_j^*\left(r',\theta+\Delta\theta,f\right)\right\rangle}{T} \qquad \text{(Tutkin Equation 9)}$$

- nb: T is the block size from the temporal-fft in Step 1 — $\hat{u}_i(r,\theta,f) = \int_{-T/2}^{T/2} e^{-i2\pi ft} u_i(r,\theta,t)dt$

```
%% calculate spatial correlation matrix
% c1=U*U';
function [Stilda]=doTutkinEq9innerProduct(t,c,vecTutkunEq8)
  % find difference AND take fft.
[ntimesteps rMin rMax ss ncs plotOn azimuthalSet azimuthalSetSize]=constants();

%struct('A', repmat({zeros(3)}, 100, 1))
for tt=1:ntimesteps
for jj=1:azimuthalSetSize
  Stilda(tt).azimuth(jj).dat=zeros(540,540);
end
end

  azimuthalSetMinusOne = azimuthalSet;
  azimuthalSetMinusOne(end) =[]; % chop off end element 1079. Ends at 1050
for tt=1:ntimesteps
  azimuthCounter=1;
%for jj=1:azimuthalSetMinusOne % local azimuthal separation.
for jj=1:azimuthalSetSize % local azimuthal separation.

for ii=1:540
   vvv1(ii) = vecTutkunEq8(tt).dat(ii,jj);
   vvv2(ii) = vecTutkunEq8(tt).dat(ii,jj+1);
end % ii
aa=transpose(vvv1)*ctranspose(transpose(vvv2));
%fill with inner product correlation matrix
 for ii=1:540
Stilda(tt).azimuth(jj).dat(:,ii)=aa(:,ii);
end % ii
azimuthCounter=azimuthCounter+1;
end %jj
end %tt
%
end %f
```

Wu and Moin Two-Point Correlation Coefficient $R_{u'_z u'_z}$ at $Re_D = 5300$

Goal

Status: done

Goal: Replicate Wu's CC Tensor

Replicate Wu-Moin two-point correlation coefficient as a function of local azimuthal separation $\Delta theta$ for different wall distances

- Link to Wu-Moin 2008 (page 108): `https://sci-hubtw.hkvisa.net/10.1017/s0022112008002085`
- Wu and Moin Two-Point Correlation Coefficient $R_{u'_z u'_z}$ at $Re_D = 5300$

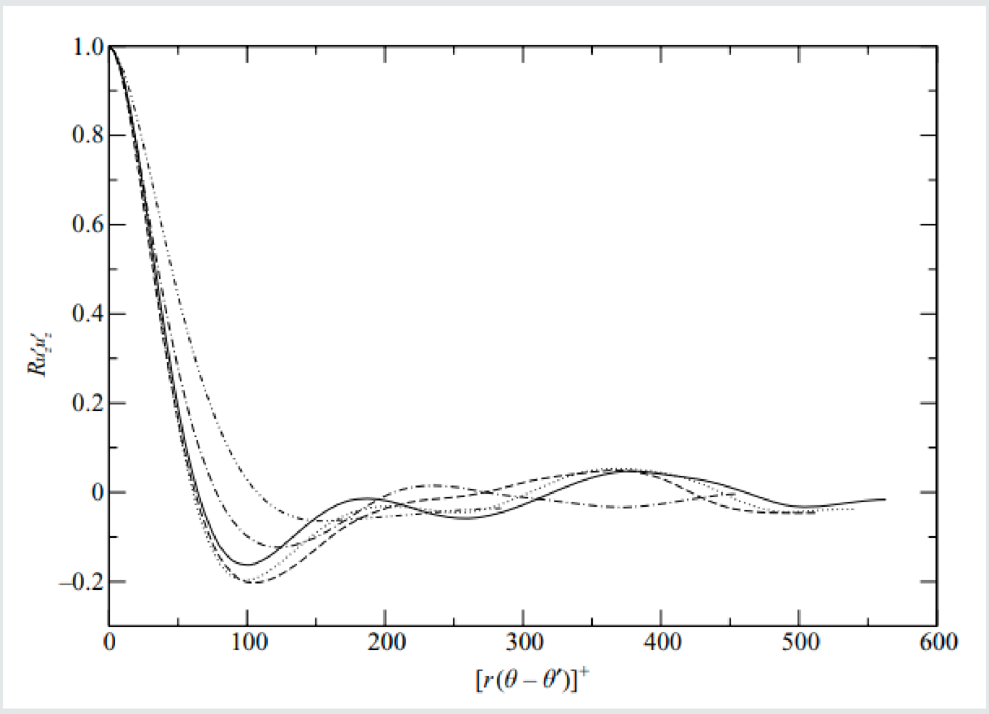


Figure 10: (Wu-Moin FIGURE 31.) Two-point correlation coefficient $R_{u'_z u'_z}$ at $Re_D = 5300$ as a function of local azimuthal separation $[r(\theta-\theta')]^+$. Solid line: $1-r = 0.01$ $((1-r)^+ = 1.81)$; dotted line: $1-r = 0.05$; dashed line: $1-r = 0.1$; chain-dotted line: $1-r = 0.2$; chain-dotted-dotted line: $1-r = 0.5$.

- Shows azimuthal separation (ie $i, j = 3$ streamwise, nb, i, j do not vary)

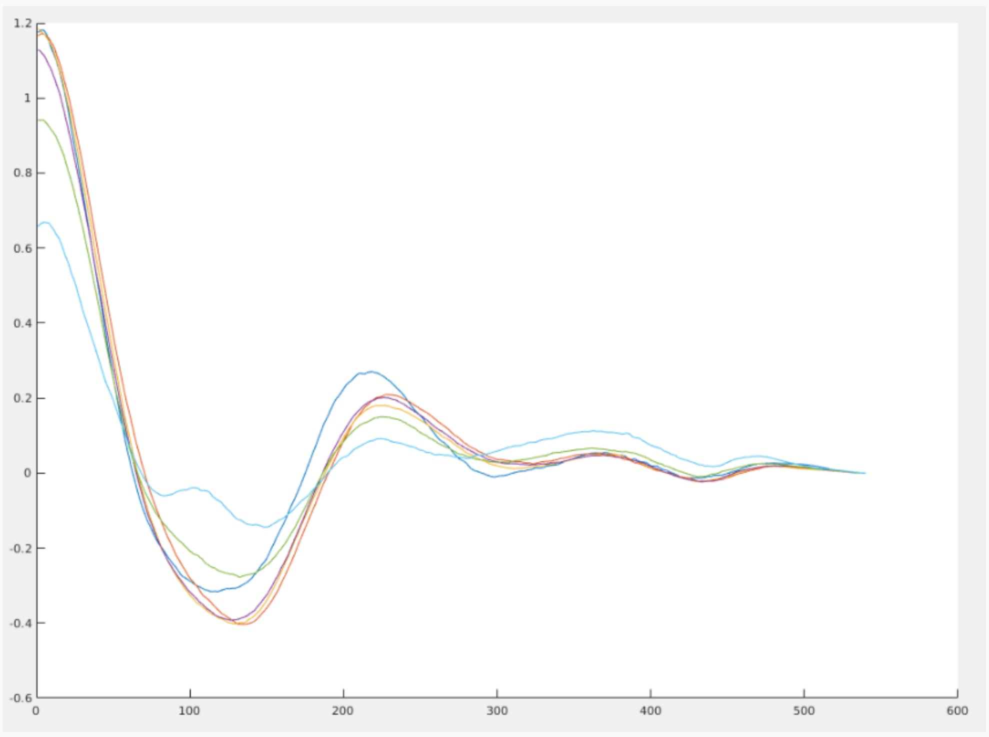


Figure 11: plot of **Sij(1).azimuth(1).dat**

- Sij is the result of Step 3 Calculation; **See below slides for Tutkin's 3 Step Procedure**

$$S_{i,j}(r, r'; m; f) = \frac{1}{2\pi}\sum_{m=0}^{n} \tilde{S}_{j,j}(r, r'; \Delta\theta; f)\, e^{-im\Delta\theta} d(\Delta\theta) \tag{17}$$

- The data matlab struct has form **Sij(ntimesteps).azimuth(numberof $\Delta\theta$ separations).dat(540 points)**

Plan: 3 Step Procedure

3 Steps to Correlation Tensor

- Follows Tutkin, Johanson and George 2006 Eq (8),(9),(9), *viz*:
- Step 1 Time fourier transform (eq8) for all pairs of points.

$$\hat{u}_i(r,\theta,f) = \int_{-T/2}^{T/2} e^{-i2\pi ft} u_i(r,\theta,t)\mathrm{d}t \qquad \text{(Tutkin Equation 8 — perhaps optional)}$$

- Step 2 (Eq 9): 2-point Cross Correlation For each two-point pair $(r_1),(\theta_1),(r_2),(\theta_2)$, where $\Delta\theta = \theta_2 - \theta_1$, find two-point crosscorrelation tensor.
- In *Tutkin*, the three-point correlation tensor is found
- In *Wu*, the correlation is found in streamise direction u_z. Thus we adopt Tutkin's algorithm to achieve Wu's result.

$$\tilde{S}_{i,j}\left(r,r';\Delta\theta;f\right) = \frac{\left\langle \hat{u}_i(r,\theta,f)\hat{u}_j^*\left(r',\theta+\Delta\theta,f\right)\right\rangle}{T} \qquad \text{(Tutkin Equation 9)}$$

- Step 3 (Eq 10) Azimuthal Fourier series expansion azimuthal separation $\Delta\theta$

$$S_{i,j}\left(r,r';m;f\right) = \frac{1}{2\pi}\sum_{m=0}^{N} \tilde{S}_{i,j}\left(r,r';\Delta\theta;f\right) e^{-im\Delta\theta}\mathrm{d}(\Delta\theta) \qquad \text{(Tutkin Equation 10)}$$

Step 1 (Perhaps Optional): Fourier Transform in Time

- Compare with Bailey File: calcSijFreq.m
- Fourier transformation of the instantaneous velocities is performed in time
- for the finite size record length
  - Issue: does T correspond to blocksize?
  - need to find out blocksize. Should be function of max f.

$$\hat{u}_i(r,\theta,f) = \int_{-T/2}^{T/2} e^{-i2\pi ftr} u_i(r,\theta,t)\mathrm{d}t \qquad \text{(Tutkin Equation 8 — perhaps optional)}$$

```matlab
function [vecTutkunEq8]=doTutkinEq8(t,c,fftTimeQminusQbar)
  % find difference AND take fft.
[ntimesteps rMin rMax ss ncs plotOn azimuthalSet azimuthalSetSize]=constants();

% initialize vecTut
for tt=1:ntimesteps
  vecTutkunEq8(tt).dat = zeros(540,1079);
  end % tt
for ii=1:540
for jj=1:1079
for tt=1:ntimesteps
  timeVecPreIntegrate(tt) = fftTimeQminusQbar(tt).dat(ii,jj);
  end % tt
  ad =  fft(timeVecPreIntegrate);

for ff=1:ntimesteps %% define uHat = uHat(r,theta,f)
  vecTutkunEq8(ff).dat(ii,jj) =  ad(ff);
end % tt
  end % ii
  end % jj
toc
end %fc
```

Step 2: Azimuthal Cross-Spectra

Definition (Step 2: Azimuthal Cross-Spectra)

- Cross-spectra are computed and block averaging is performed as follows

$$\tilde{S}_{i,j}\left(r, r'; \Delta\theta; f\right) = \frac{\left\langle \hat{u}_i(r,\theta,f)\hat{u}_j^*\left(r', \theta + \Delta\theta, f\right)\right\rangle}{T} \qquad \text{(Tutkin Equation 9)}$$

- Code

```
function [Stilda]=doTutkinEq9ReduxZwo(t,c,vecTutkunEq8)
  % find difference AND take fft.
[ntimesteps rMin rMax ss ncs plotOn azimuthalSet azimuthalSetSize]=constants();

  azimuthalSetMinusOne = azimuthalSet;
  azimuthalSetMinusOne(end) =[]; % chop off end element 1079. Ends at 1050
for tt=1:ntimesteps
  azimuthCounter=1
for jj= azimuthalSetMinusOne % local azimuthal separation.

for ii=1:540
%for tt=1:ntimesteps
   vvv1(ii) = vecTutkunEq8(tt).dat(ii,azimuthCounter);
   %vvv2(tt) = ctranspose(vecTutkunEq8(tt).dat(ii,jj+1));
   % do not take transpose, just do correlation.
   vvv2(ii) = vecTutkunEq8(tt).dat(ii,azimuthCounter+1);
end % tt
%aa=xcorr(vvv1,vvv2);
aa=xcov(vvv1,vvv2);
%for tt=1:2*540-1
for ii=1:2*540-1
%Stilda(tt).azimuth(jj).radial(ii).dat=aa(tt);
Stilda(tt).azimuth(azimuthCounter).dat(ii,1)=aa(ii);
end % tt
azimuthCounter=azimuthCounter+1
end %jj
end %ii
%end %iiPrime
%
end %fc
```

Step 3: Numerical Spatial Fourier

Definition (Step 3: Numerical Spatial Fourier)

- The doubly transformed cross-spectra are computed by the

  numerical approximation to the Fourier series expansion of Eq. (9) in the azimuthal direction
- Uses Taylor's Frozen Field Hypothesis for spatial fourier.

$$S_{i,j}\left(r,r';m;f\right)=\frac{1}{2\pi}\sum_{\mathrm{m=0}}^{N}\tilde{S}_{i,j}\left(r,r';\Delta\theta;f\right)e^{-im\Delta\theta}\mathrm{d}(\Delta\theta) \tag{18}$$

- This computes the needed two-point velocity correlation tensor $R_{ij}(x,x',t,t')$ used in the classical POD integral equation (see Gamard & George 2002 `https://sci-hubtw.hkvisa.net/10.1063/1.1471875` , page 2517).

- Code

```
function [Sij]=doTutkinEq10Redux(t,c,Stilda)
  % find difference AND take fft.
[ntimesteps rMin rMax ss ncs plotOn azimuthalSet azimuthalSetSize]=constants();
dtheta = 50;
blocksize=size(Stilda,1);
for tt=1:ntimesteps % frequency f
for ii=1:540
      sprintf('%s%d%s%d','*calculating cross-spectra for tt=',tt,'and ii=',ii)
%for iiPrime=1:540 % this is implicitly understood. Leave out for now.
counterS  = 1;
for mm=azimuthalSet
    deg=0;
    sumSij=zeros(blocksize,1);
    for n=azimuthalSet
        rad=deg/180*pi;
        sumSij=Stilda(tt).azimuth(n).dat(ii,1)*exp(-1i*mm*rad)*dtheta/180*pi+sumSij;
        deg=deg+dtheta;
    end % for n
    Sij(tt).azimuth(:,counterS).dat(ii,iiPrime)=sumSij;
    counterS = counterS +1;
end %for mm
%end % iiPrime
end % ii
end % freq f
end %fc
```

Definition (Classical POD following Citirini \ George 2001)

Classical POD Procedure

- POD Implementation — Classical
- Follows Reference eg [6]
- Link: `https://sci-hub.hkvisa.net/10.1017/s0022112000001087` page 141
- Vgl Hellstrom 2016: `https://sci-hub.hkvisa.net/10.1017/jfm.2016.100`
- Citriniti takes fft in time and works in spectral domain f
- hellstrom (2016 and others) does not take time-fft, but instead integrates in time in order to throw out time.
- Upshot: For now: Procedure is to follow Citriniti! ! !

- Do for the streamwise velocity component,
- Work with the double Fourier transform given b

$$\hat{u}_i(r,m,f) = \int_0^\infty \int_0^\infty \mathrm{e}^{-i(2\pi f t + m\theta)} u(r,\theta,t) \mathrm{d}\theta \mathrm{d}t \tag{Tutkin Eq 10,Citriniti Eq 2.4}$$

- Formed the 2-point cross-spectrum $S_{11}(r,r',m,f)$,

$$S_{i,j}\left(r,r',m,f\right) = \left\langle \hat{u}_i(r,m,f)\hat{u}_j^*\left(r',m,f\right)\right\rangle . \tag{19}$$

- Integral Equation for the POD is,

$$\int S_{i,j}\left(r,r',m,f\right)\varphi_j\left(r',m,f\right) r'\mathrm{d}r' = \lambda(m,f)\varphi_i(r,m,f) \tag{20}$$

- Make kernal Hermitian symmetric wrt the integrand as required by Hilbert-Schmidt theory,

$$B_{i,j}\left(r,r',m,f\right) = r^{1/2} S_{i,j}\left(r,r',m,f\right) r'^{1/2} \tag{21}$$
$$\phi_i(r,m,f) = r''^{1/2\varphi_i(r,m,f)} \tag{22}$$

- Integral equation to solve is now,

$$\int B_{i,j}\left(r,r',m,f\right)\phi_j\left(r',m,f\right)\mathrm{d}r' = \lambda(m,f)\phi_i(r,m,f) \tag{23}$$

where λ is now an eigenspectrum, decomposed into azimuthal Fourier modes.

Spectral Analysis

- Markus Raffel, Christian E Willert, Jürgen Kompenhans, et al. *Particle image velocimetry: a practical guide*. Vol. 2. discussion of phase shift of the imaginary part being equivalent to the real part Phi POD mode. Springer, 1998.
- Murat Tutkun, Peter B. V. Johansson, and William K. George. "Three-Component Vectorial Proper Orthogonal Decomposition of Axisymmetric Wake Behind a Disk". In: *AIAA Journal* 46.5 (May 2008), pp. 1118–1134. ISSN: 1533-385X. DOI: `10.2514/1.31074`. URL: `https://sci-hub.hkvisa.net/10.2514/1.31074`.
- XIAOHUA WU and PARVIZ MOIN. "A direct numerical simulation study on the mean velocity characteristics in turbulent pipe flow". In: *Journal of Fluid Mechanics* 608 (July 2008), pp. 81–112. ISSN: 1469-7645. DOI: `10.1017/s0022112008002085`. URL: `http://dx.doi.org/10.1017/s0022112008002085`.

Classical POD

- Sean CC Bailey and Alexander J Smits. "Experimental investigation of the structure of large-and very-large-scale motions in turbulent pipe flow". In: *Journal of Fluid Mechanics* 651 (2010), pp. 339–356.
- Stephan Gamard et al. "Application of a "slice" proper orthogonal decomposition to the far field of an axisymmetric turbulent jet". In: *Physics of Fluids* 14.7 (2002), p. 2515. ISSN: 1070-6631. DOI: `10.1063/1.1471875`. URL: `http://dx.doi.org/10.1063/1.1471875`.
- Leo Hellstroem. "Coherent Structures in Turbulent Pipe Flow". PhD thesis. Princeton University, 2015. URL: `https://drive.google.com/file/d/1gPC02JtRyHa9cdRUOZwoQfkCrJLwDfYV/view?usp=sharing`.
- Leo H. O. Hellström, Ivan Marusic, and Alexander J. Smits. "Self-similarity of the large-scale motions in turbulent pipe flow". In: *Journal of Fluid Mechanics* 792 (Mar. 2016). ISSN: 1469-7645. DOI: `10.1017/jfm.2016.100`. URL: `http://dx.doi.org/10.1017/jfm.2016.100`.
- Bernd R. Noack. "Turbulence, Coherent Structures, Dynamical Systems and Symmetry". In: *AIAA Journal* 51.12 (Dec. 2013), pp. 2991–2991. ISSN: 1533-385X. DOI: `10.2514/1.j052557`. URL: `http://dx.doi.org/10.2514/1.J052557`.

Snapshot POD

- J Baltzer, R Adrian, and X Wu. "Turbulent boundary layer structure identification via POD". In: *Proceedings of the summer program*. snapshot POD and discussion of meaning of imaginary part of POD modes (just a phase shift); see Liu et al 2001. 2010, p. 55.
- Leo H.O. Hellström and Alexander J. Smits. "Structure identification in pipe flow using proper orthogonal decomposition". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 375.2089 (Mar. 2017), p. 20160086. ISSN: 1471-2962. DOI: `10.1098/rsta.2016.0086`. URL: `http://dx.doi.org/10.1098/rsta.2016.0086`.
- Z. LIU, R. J. ADRIAN, and T. J. HANRATTY. "Large-scale modes of turbulent channel flow: transport and structure". In: *Journal of Fluid Mechanics* 448 (Nov. 2001). discussion of phase shift of the imaginary part being equivalent to the real part Phi POD mode, pp. 53–80. ISSN: 1469-7645. DOI: `10.1017/s0022112001005808`. URL: `http://dx.doi.org/10.1017/S0022112001005808`.

Turbulence

- C.D. Argyropoulos and N.C. Markatos. "Recent advances on the numerical modelling of turbulent flows". In: *Applied Mathematical Modelling* 39.2 (Jan. 2015). dns of turbulent pipe flows p 716; good review paper!, p. 716. ISSN: 0307-904X. DOI: `10.1016/j.apm.2014.07.001`. URL: `http://dx.doi.org/10.1016/j.apm.2014.07.001`.
- SEAN C. C. BAILEY et al. "Azimuthal structure of turbulence in high Reynolds number pipe flow". In: *Journal of Fluid Mechanics* 615 (Nov. 2008). has correlation coefficient on page 127, pp. 121–138. ISSN: 1469-7645. DOI: `10.1017/s0022112008003492`.
- Peter Alan Davidson. *Turbulence: an introduction for scientists and engineers*. Oxford university press, 2015. Chap. 9.
- M. Hultmark et al. "Turbulent Pipe Flow at Extreme Reynolds Numbers". In: *Physical Review Letters* 108.9 (Feb. 2012). ISSN: 1079-7114. DOI: `10.1103/physrevlett.108.094501`. URL: `http://dx.doi.org/10.1103/PhysRevLett.108.094501`.
- I. Marusic et al. "Wall-bounded turbulent flows at high Reynolds numbers: Recent advances and key issues". In: *Physics of Fluids* 22.6 (June 2010). Motivation why we care about wall turbulence and coherent structures, p. 065103. ISSN: 1089-7666. DOI: `10.1063/1.3453711`. URL: `http://dx.doi.org/10.1063/1.3453711`.
- Tomás Chacón Rebollo and Roger Lewandowski. *Mathematical and numerical foundations of turbulence models and applications*. Springer, 2014.
- Pierre Sagaut and Claude Cambon. *Homogeneous Turbulence Dynamics*. Springer, 2008.
- B.M. Sumer and D.R. Fuhrman. *Turbulence In Coastal And Civil Engineering*. Advanced Series On Ocean Engineering. World Scientific Publishing Company, 2020, pp. 184–. ISBN: 9789813234321. URL: `https://books.google.com/books?id=ztLkDwAAQBAJ`.

References 2

Background and Pipe Energy Mode Behavior

- Joseph H Citriniti and William K George. “Reconstruction of the global velocity field in the axisymmetric mixing layer utilizing the proper orthogonal decomposition”. In: *Journal of Fluid Mechanics* 418 (2000), pp. 137–166 (cit. on p. 27).
- Andrew Duggleby and Mark R Paul. “Computing the Karhunen–Loéve dimension of an extensively chaotic flow field given a finite amount of data”. In: *Computers & Fluids* 39.9 (2010), pp. 1704–1710.
- Stephan Gamard et al. “Application of a “slice” proper orthogonal decomposition to the far field of an axisymmetric turbulent jet”. In: *Physics of Fluids* 14.7 (2002), pp. 2515–2522.
- William K George. “Insight into the dynamics of coherent structures from a proper orthogonal decomposition dy”. In: *International seminar on wall turbulence*. 1988.
- Stanislav V Gordeyev and Flint O Thomas. “Coherent structure in the turbulent planar jet. Part 1. Extraction of proper orthogonal decomposition eigenmodes and their self-similarity”. In: *Journal of Fluid Mechanics* 414 (2000), pp. 145–194.