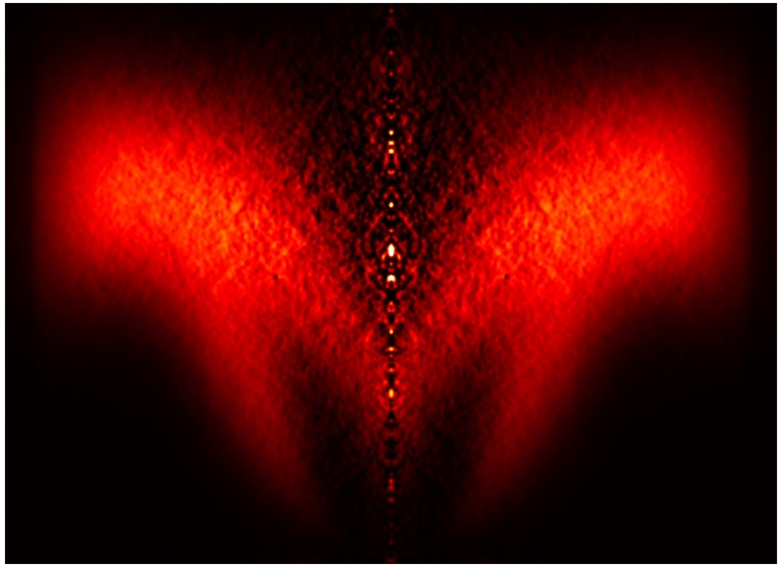
Calculation of flame describing functions from experimental data



POLKA – 6th Scientific Workshop

17 March 2023

Preethi Rajendram Soundararajan
Department of Engineering, University of Cambridge, UK

EM2C · cnrs · CentraleSupélec · université PARIS-SACLAY · ANNULIGhT · European Commission

In this lecture

- Lecture:
  - Essentials of signal processing
- Exercise
  - Measuring FDFs experimentally
  - Calculating FDFs from experimental data

Sampling theorem

How fast should you sample data?

How fast should you sample data?

Nyquist-Shannon criterion

An analog signal should be sampled at a rate **greater than twice** its highest frequency

$$f s > 2 \times f_m$$

Sampling frequency

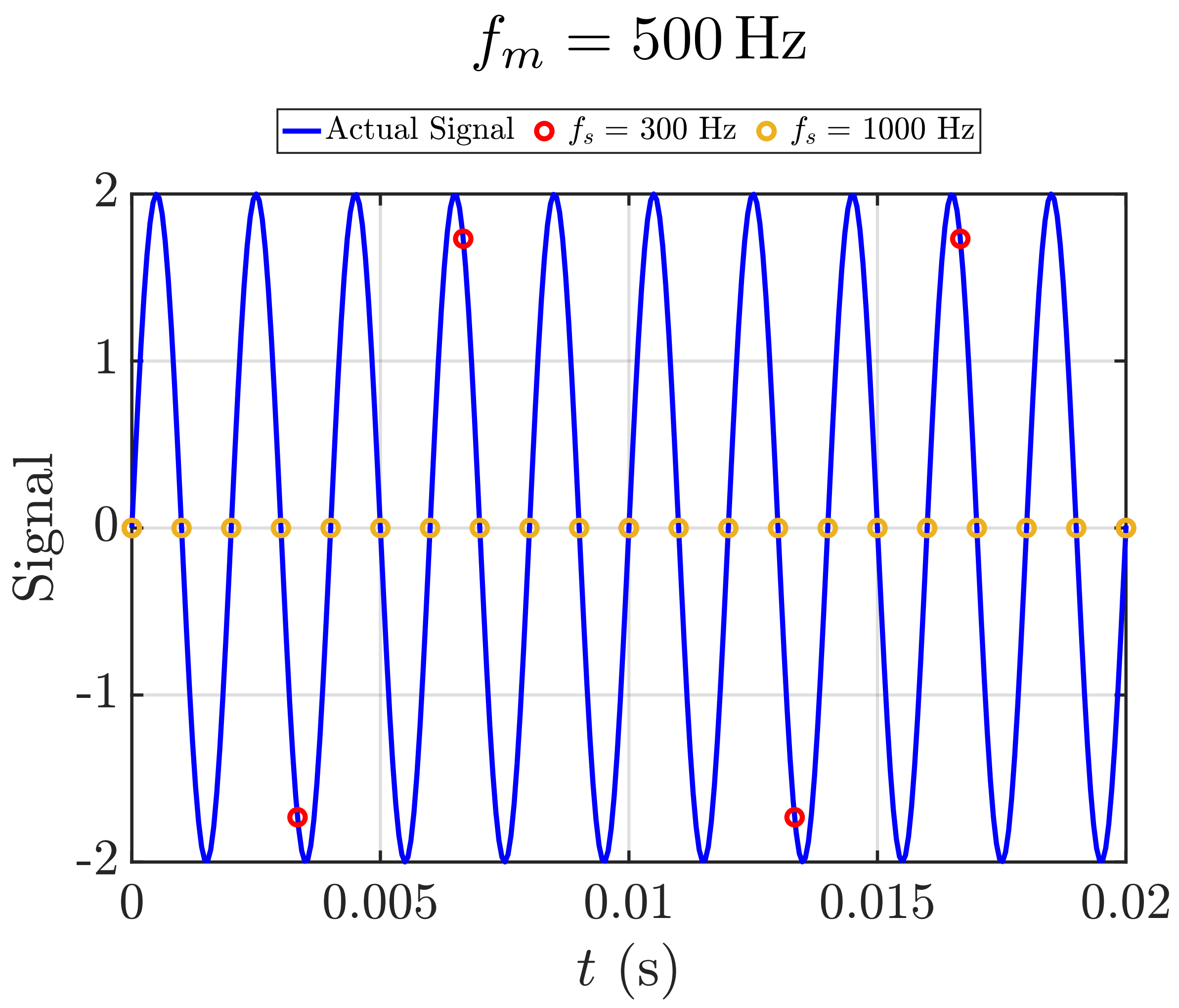
Maximum frequency

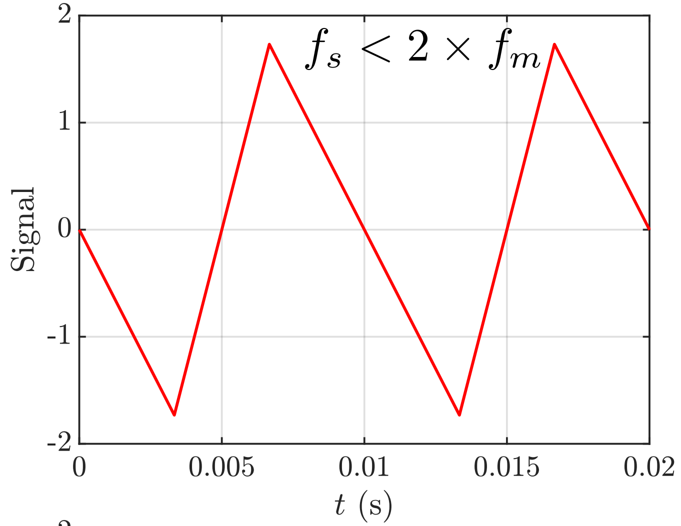
Nyquist frequency

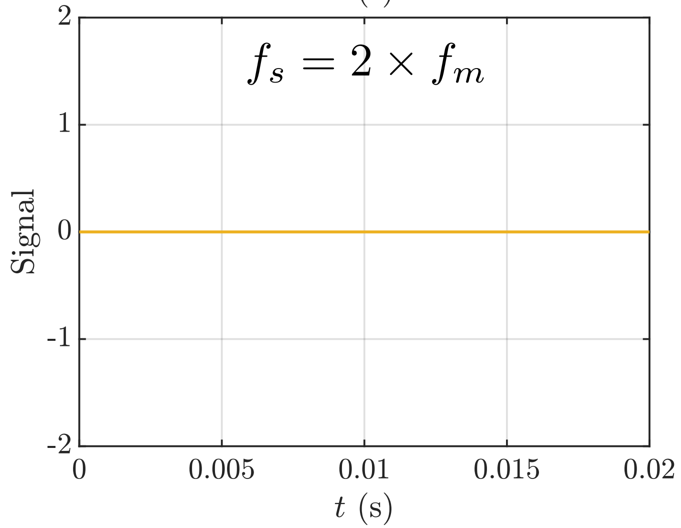
$$f_{\mathrm{Nq}} = 2 \times f_m$$

Sampling theorem

Nyquist-Shannon criterion: $fs > 2 \times f_m$



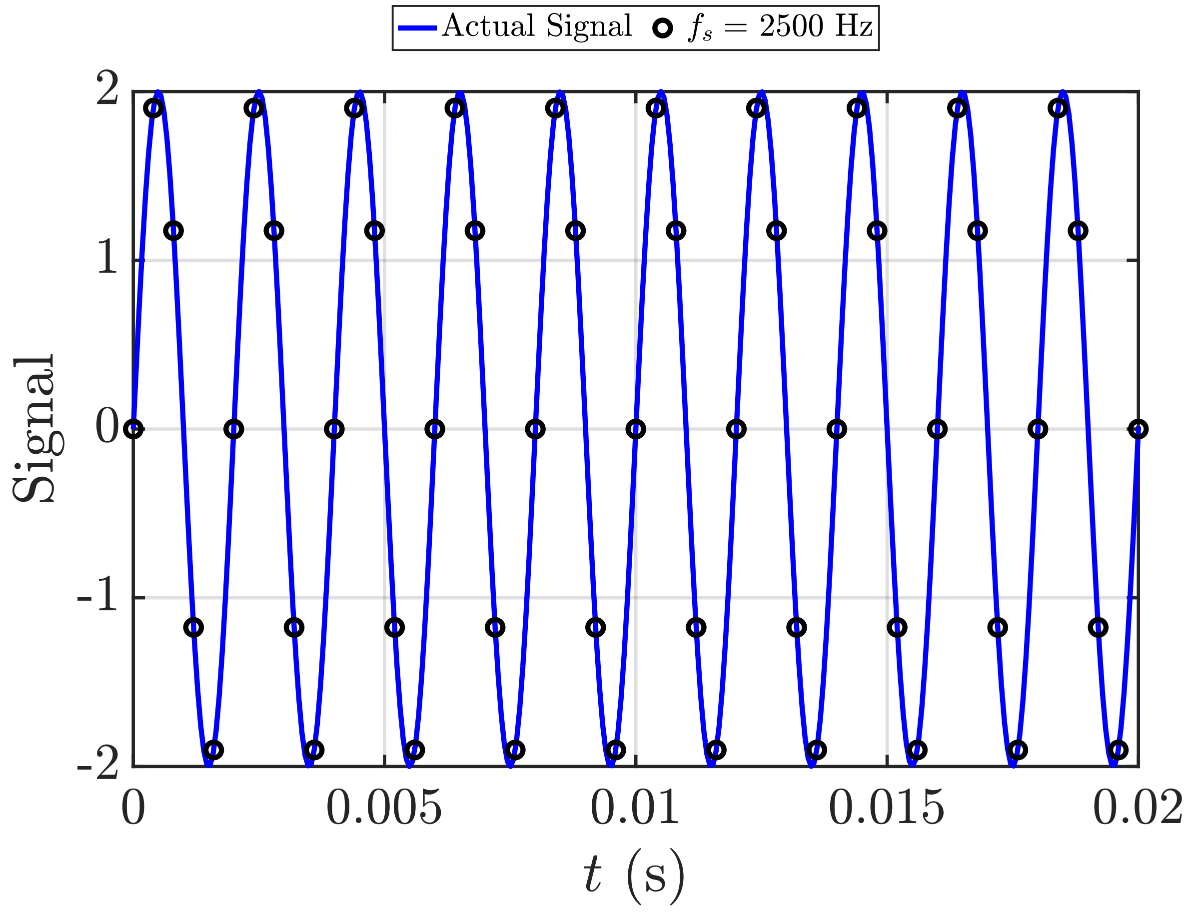


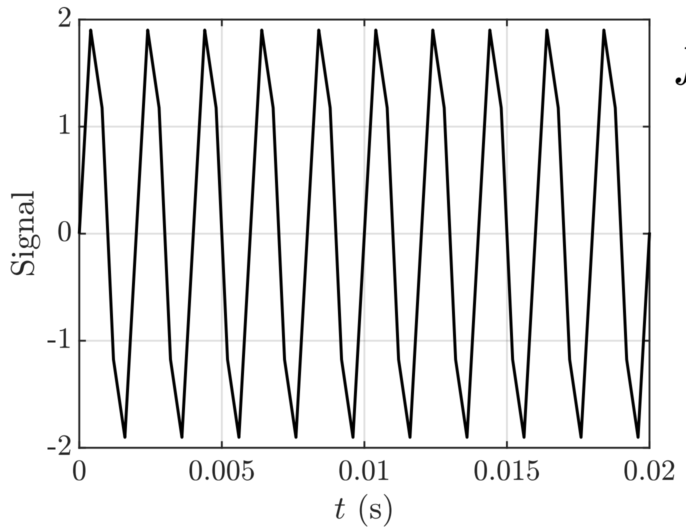


Sampling theorem

Nyquist-Shannon criterion: $fs > 2 \times f_m$

$f_m = 500\,\text{Hz}$



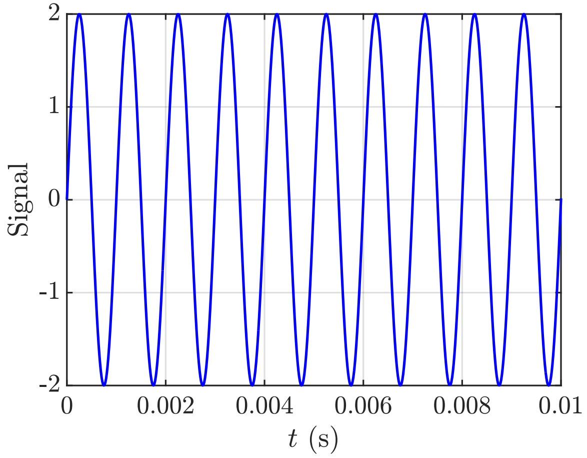


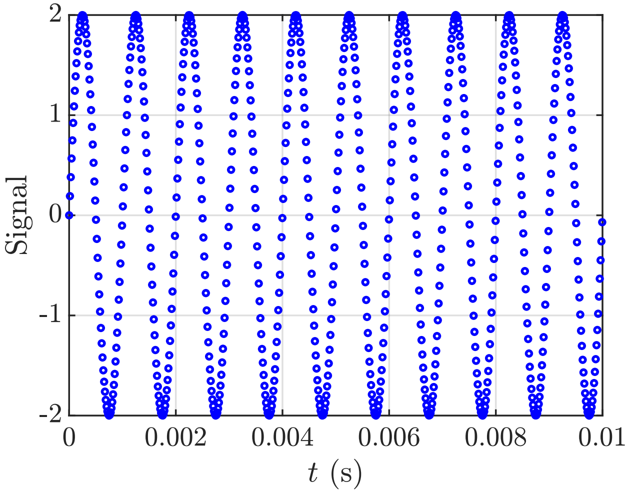
$f_s > 2 \times f_m$

We got the frequency correct, but the amplitude?

In practice, because of noise etc., a sampling frequency **larger than ten times** is better

Spectral analysis



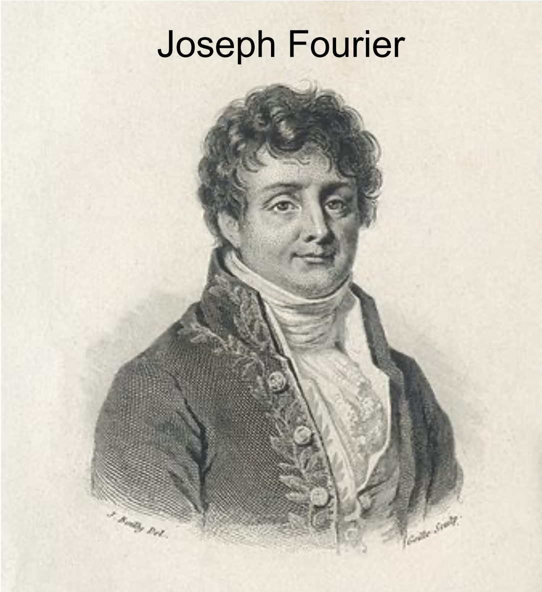


Fourier transform

Time domain → Frequency domain

$$X_c(f) = \int_{-\infty}^{+\infty} x_c(t) e^{-i2\pi f t} dt$$

Frequency domain   Time domain



For discrete signals

Discrete Fourier transform

$$X(\omega) = \sum_{n=0}^{N-1} x(t) e^{-i\omega t}$$

$$t = n \times \frac{1}{fs}$$

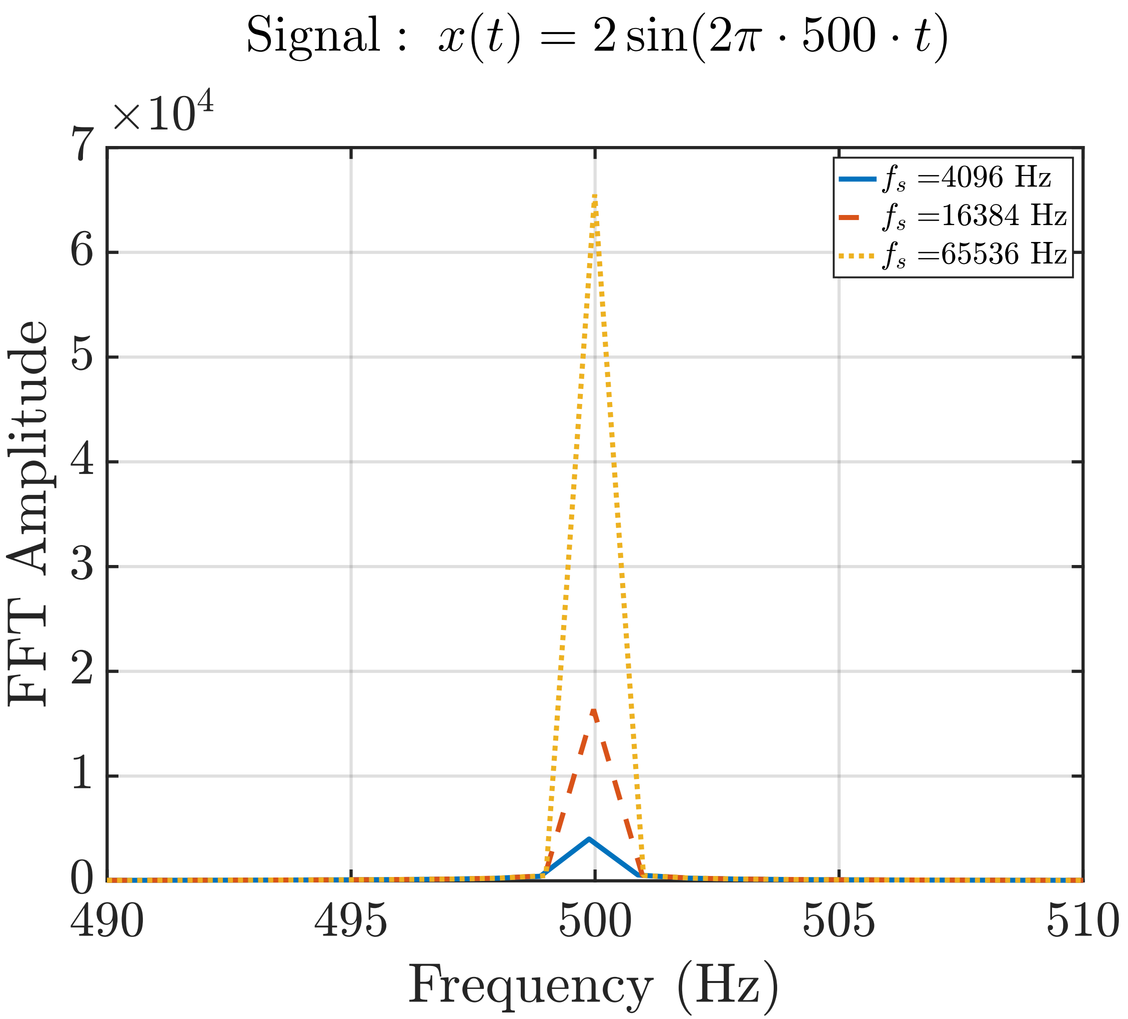
N – No. of points

Fast Fourier transform algorithm

Cooley – Tukey algorithm

For numerical efficiency, no. of samples is a power of 2

Spectral analysis – FFT dependance on sampling rate

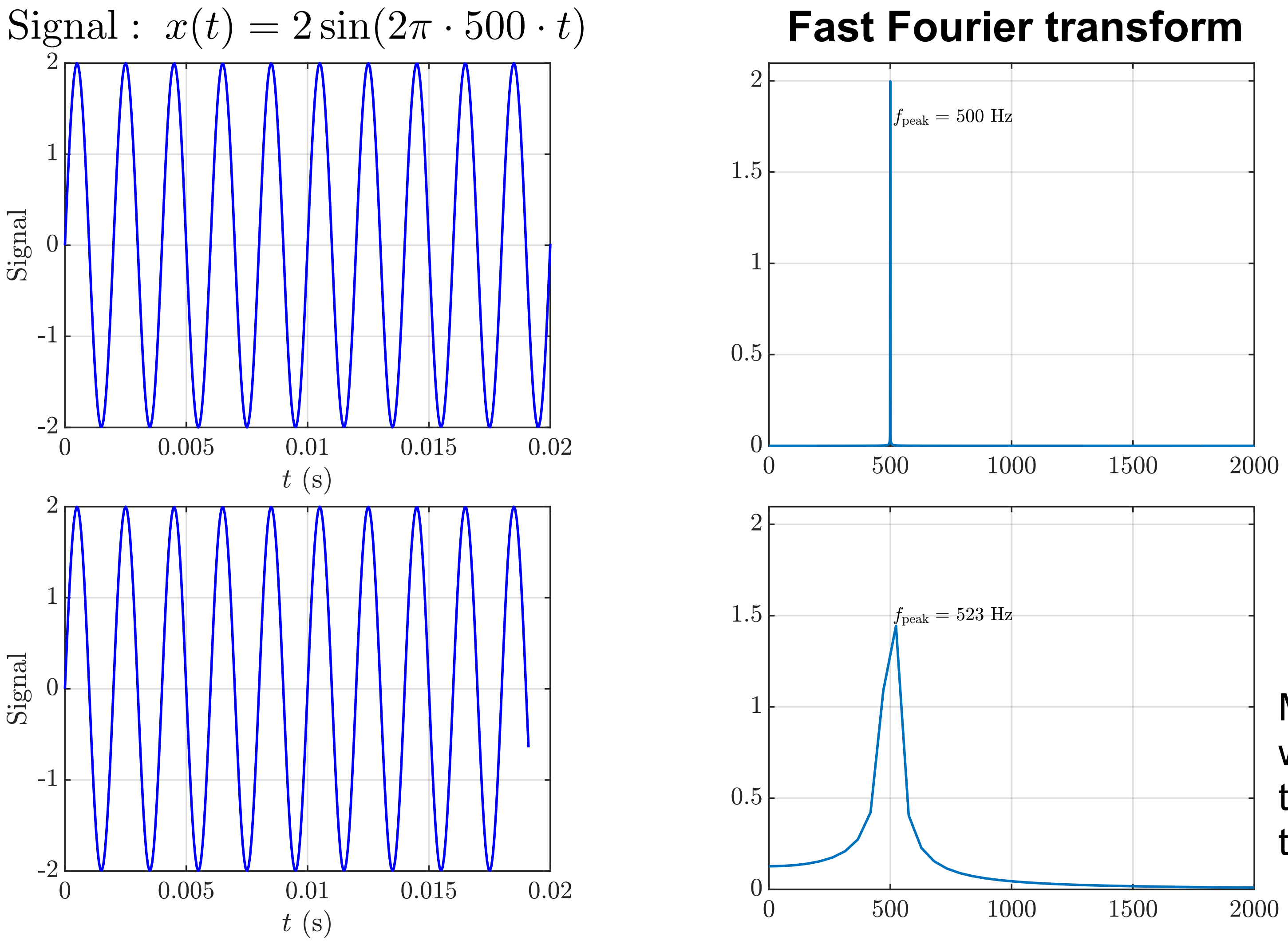


$$X(\omega) = \sum_{n=0}^{N-1} x(t) e^{-i\omega t}$$

Amplitude depends on N number of points in DFT

Normalize by frequency resolution

Spectral analysis – spectral leakage



DFT assumes the signal is periodic

Multiplying the signal by windows of finite length in which the amplitude gradually reduces to zero at the edges

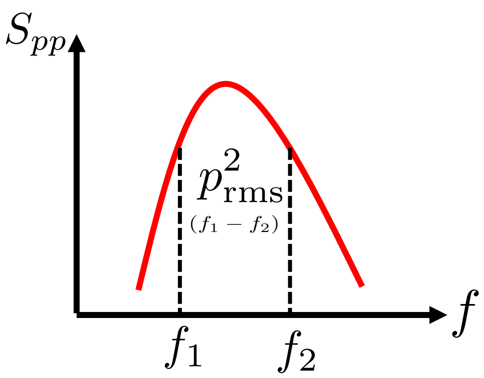
This can be avoided by windowing

Power spectral density

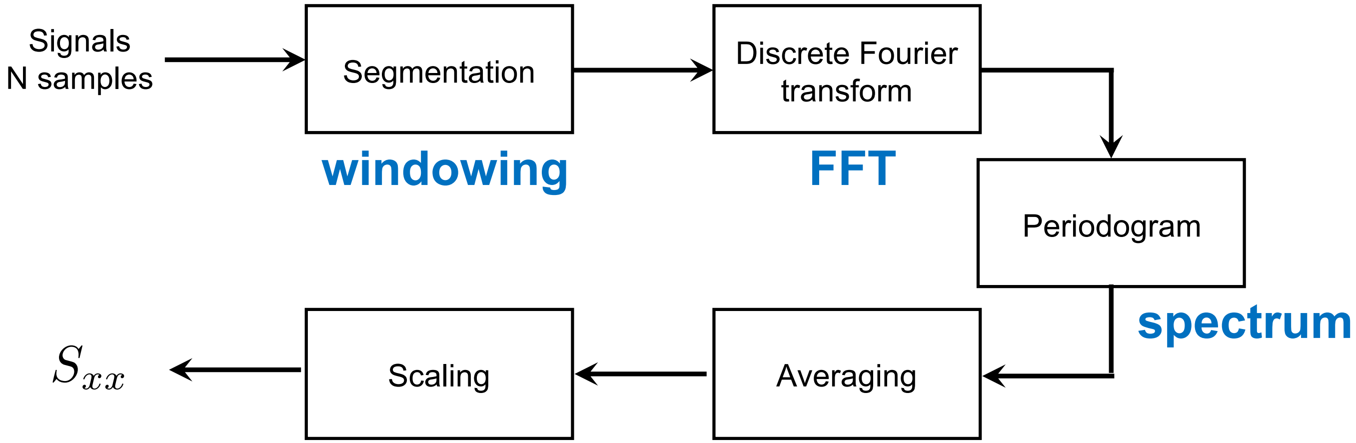
Power spectral density

mean square value per unit frequency

$$p_{\mathrm{rms}}^2 = \int S_{pp}(f)\mathrm{d}f$$

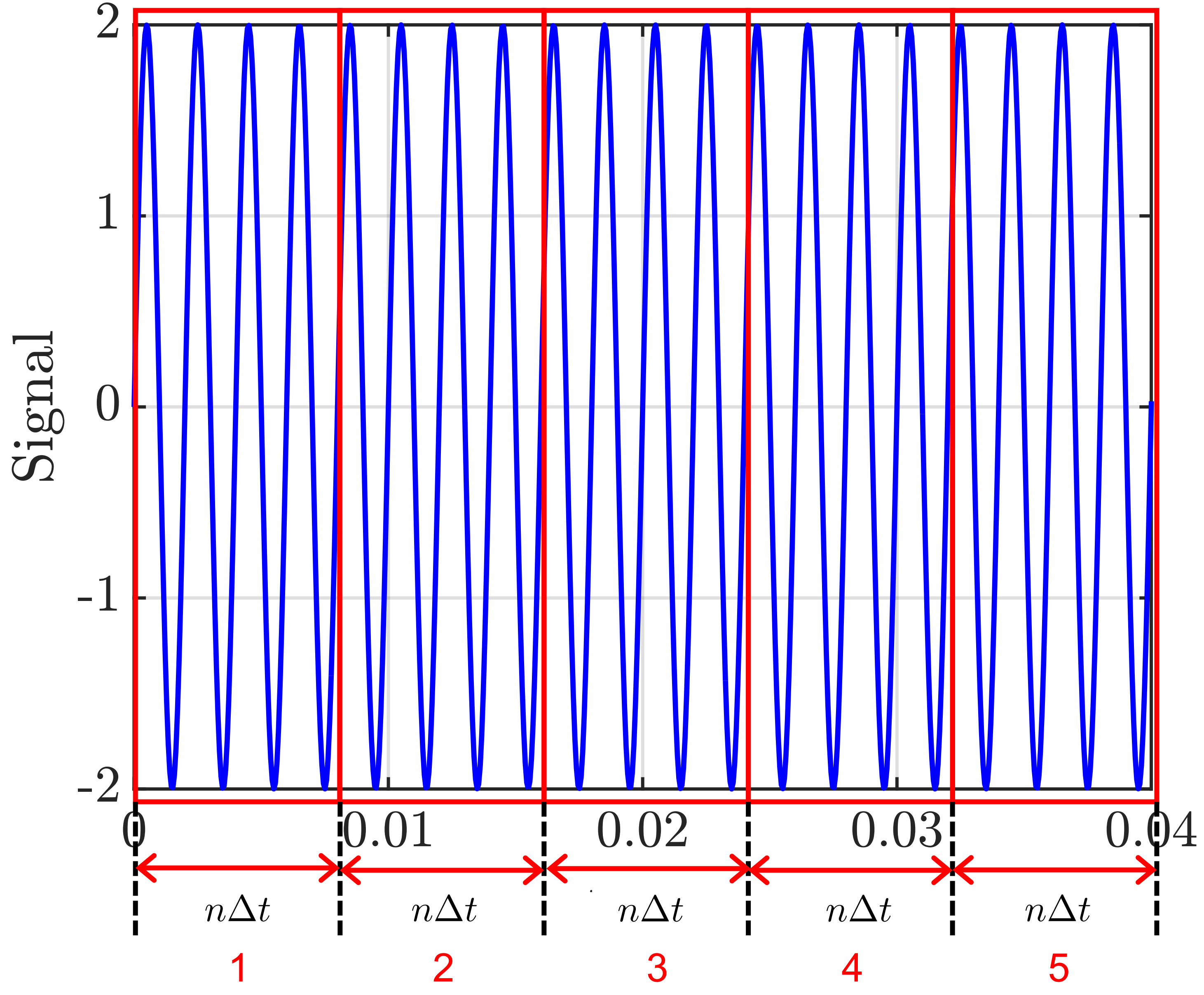


Estimating power spectral density using Welch's periodogram technique

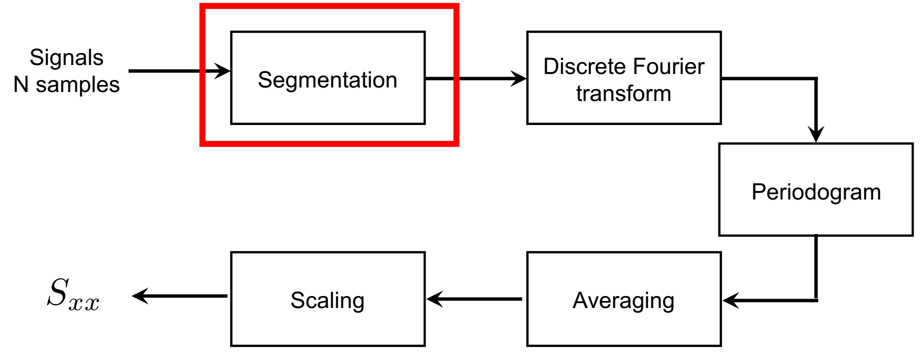


Reproduced from Denis Veynante, Survey of signal processing techniques

Spectral analysis – Windowing



$$\text{Signal}: \; x(t) = 2\sin(2\pi \cdot 500 \cdot t)$$



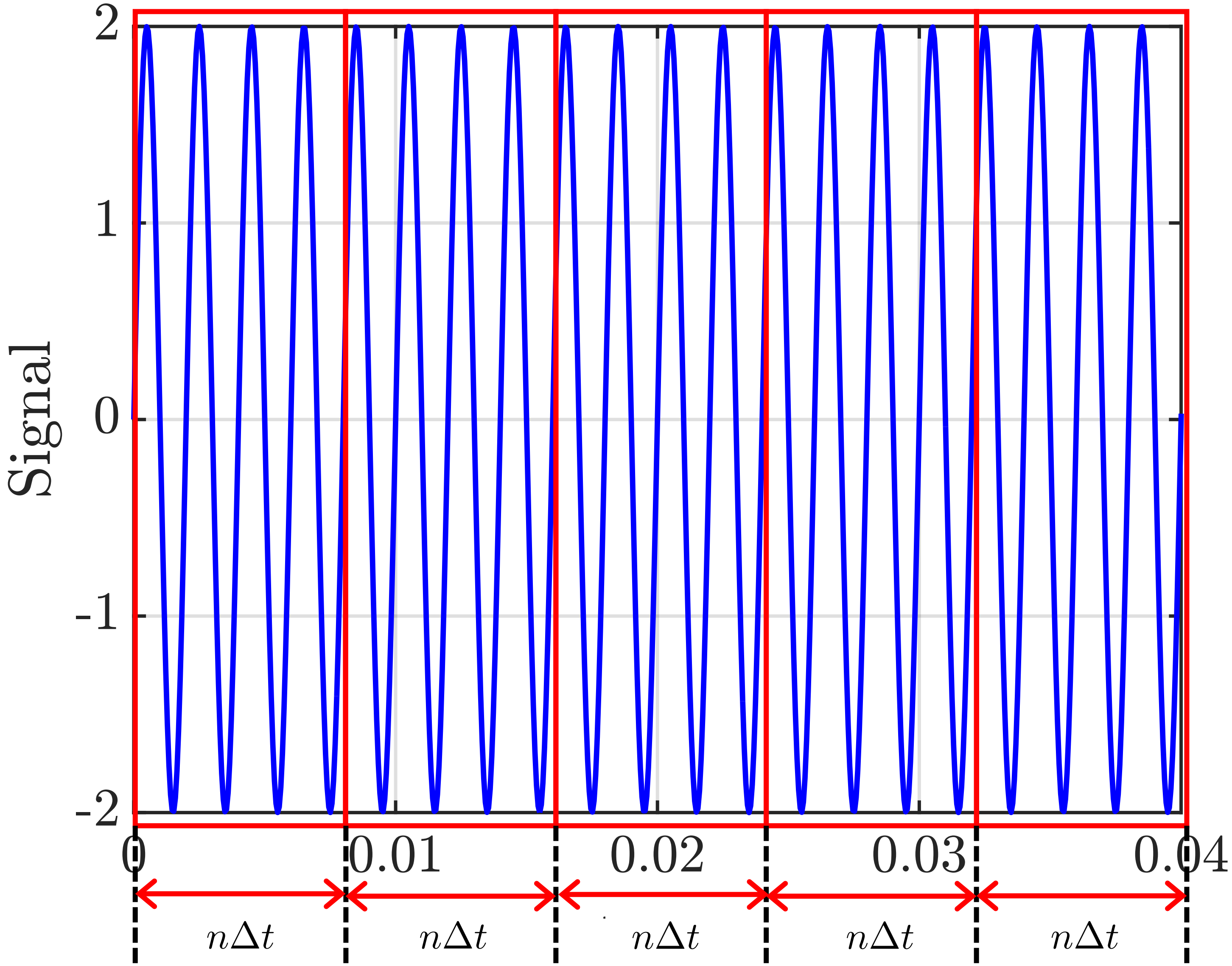
No. of windows: M

No. of points/window: n

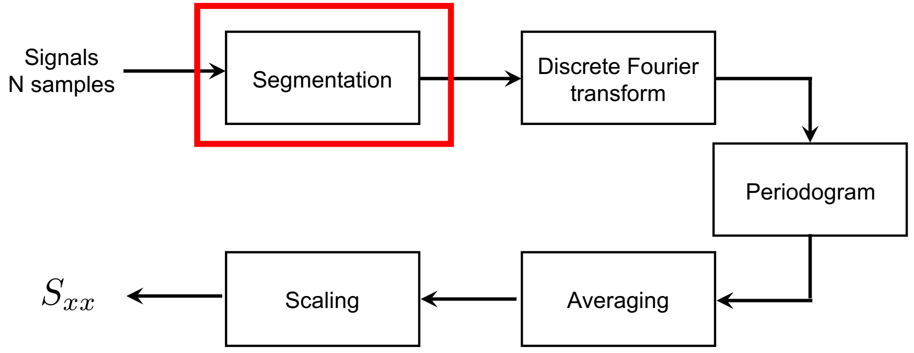
$$N = M \times n$$

Different types of windowing functions exist
Hanning, Hamming etc.
Choice depends on the application

Spectral density using Welch's periodogram approach



$$\text{Signal}: \; x(t) = 2\sin(2\pi \cdot 500 \cdot t)$$



No. of windows: M

No. samples/window: n

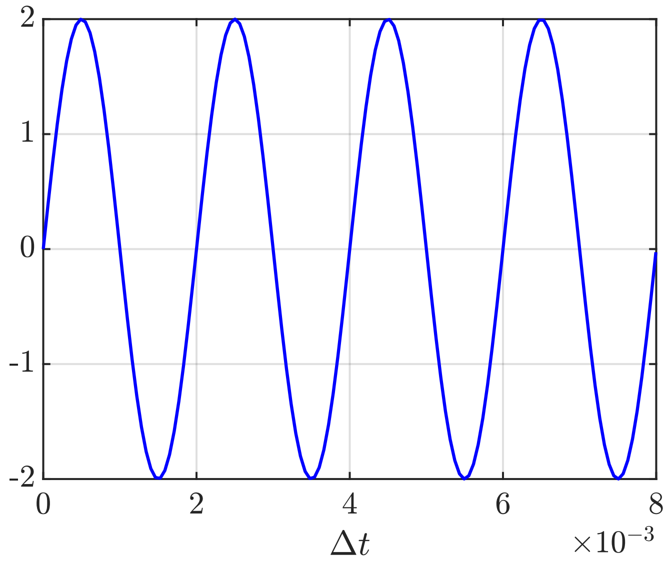
$$N = M \times n$$

Frequency resolution $$\Delta f = \frac{1}{n\Delta t}$$

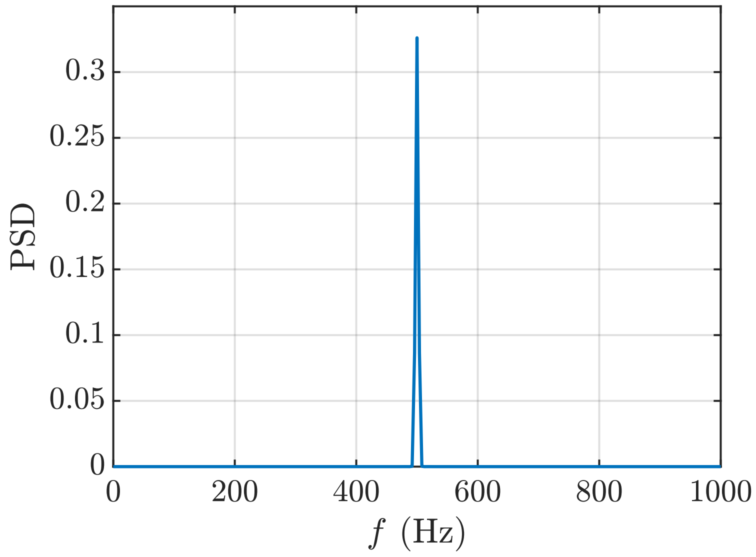
Δt is the time step

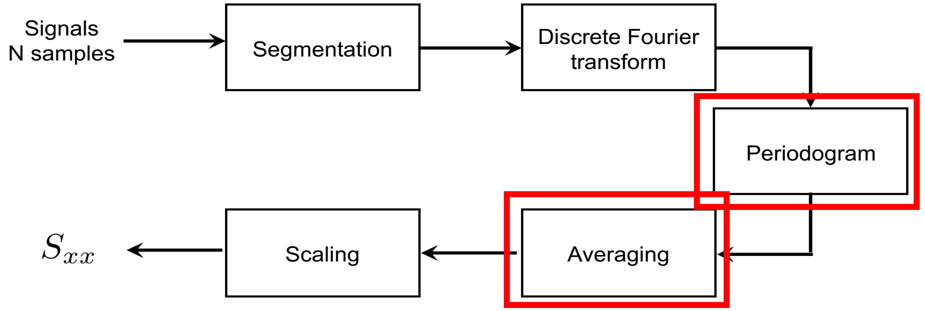
Spectral density using Welch's periodogram approach

Windowed signal



Periodogram





In the next step, averaging of periodograms from different windows is averaged

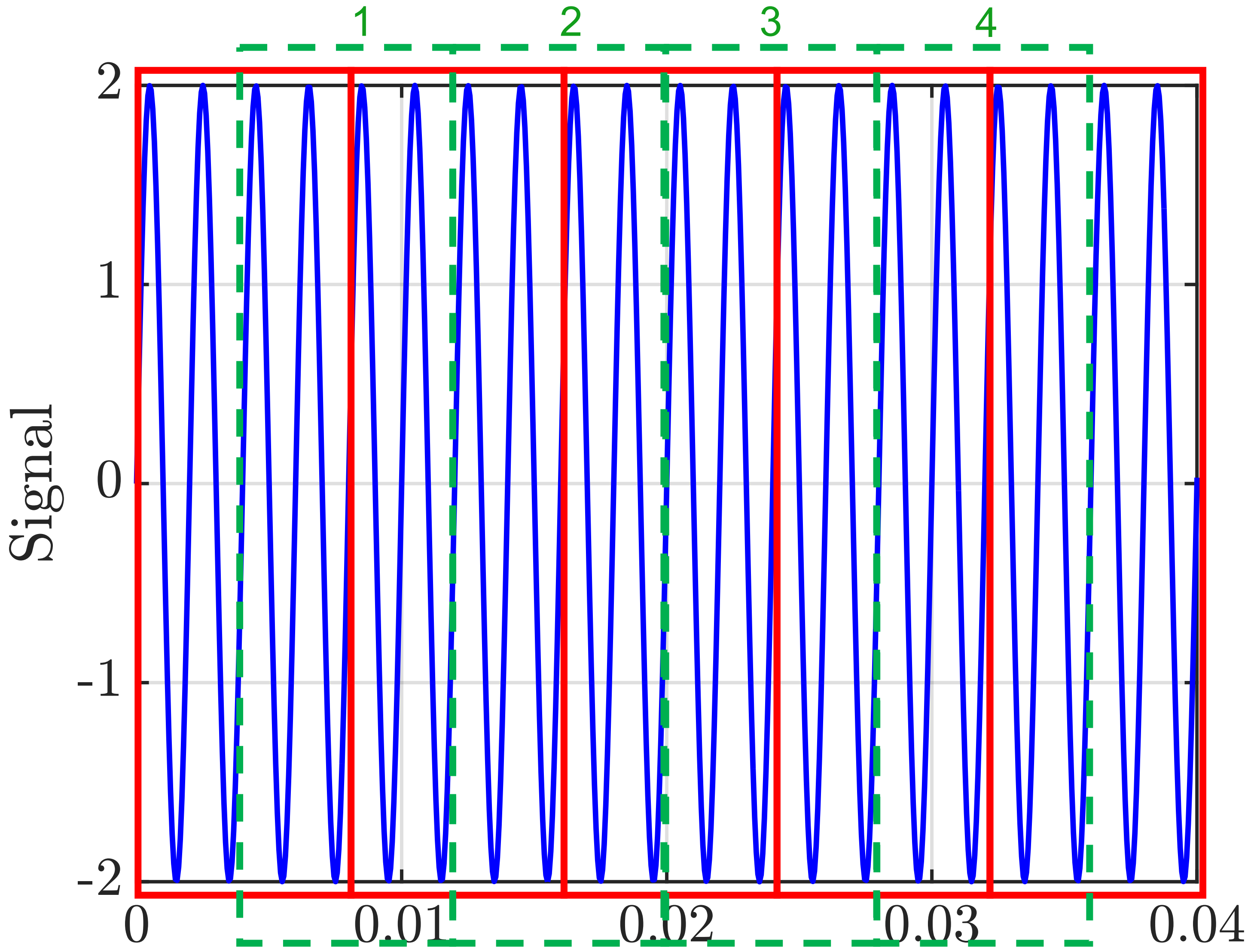
Practical signals have noise – averaging operation smoothens the effect of noise

Increasing M improves statistical stability and provides better spectral estimates

Frequency resolution decreases

$$\Delta f = \frac{1}{n\Delta t}$$

Spectral estimate – overlapping



Improving statistical stability without compromising frequency resolution

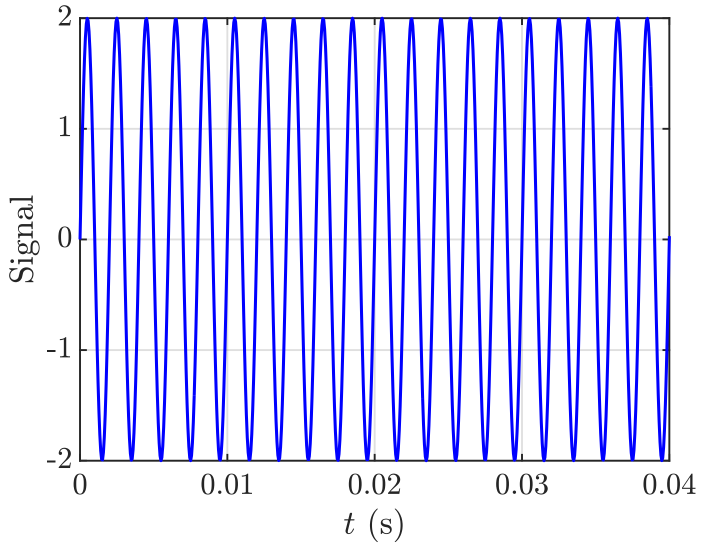
Overlapping

No. of orginal **windows** = 5

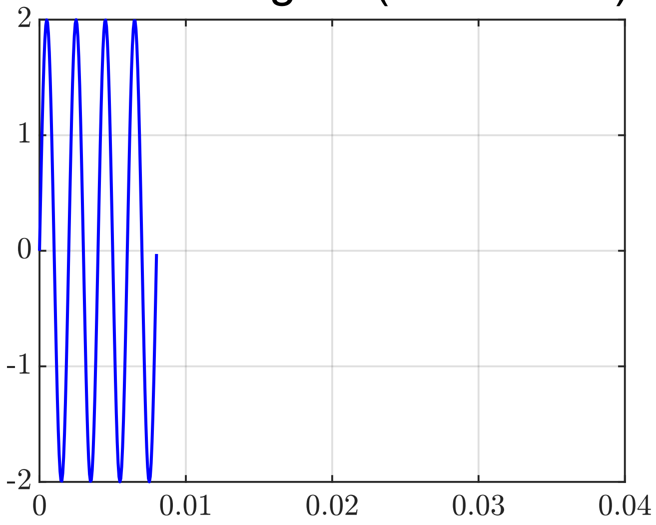
with 50% **overlap**
Total no. of windows = 9

Spectral estimate – Zero-padding

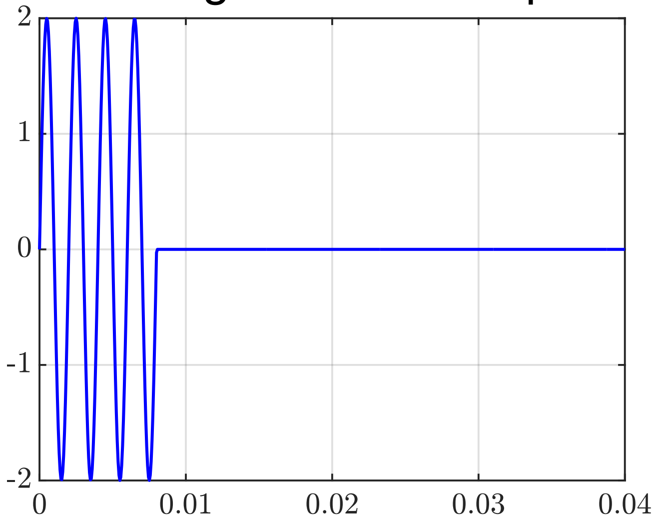
Original signal



windowed signal (5 windows)



windowed signal after zero-padding



Artifically increasing frequency resolution

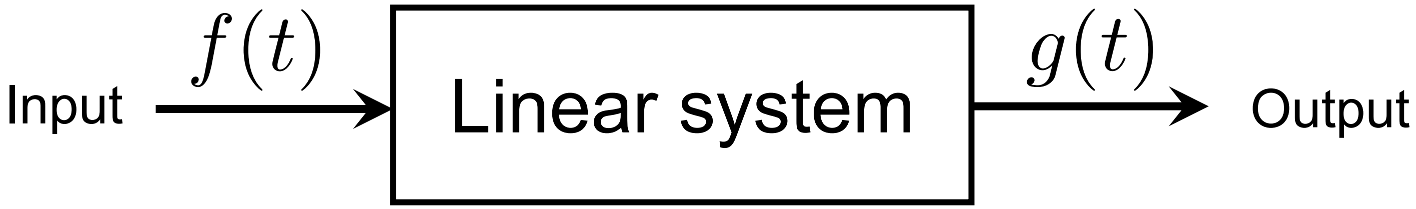
$$\Delta f = \frac{1}{n\Delta t}$$

n is increased after zero-pading

No actual information in zero-padded data

It is not exactly resolution that increases but it is rather the frequency precision

Transfer function



$$S_{fg}(\omega) = H(j\omega)S_{ff}(\omega)$$

Cross-power spectral density

Tranfer function

Power spectral density

Gives a gain and a phase

P. A. Nelson & S. J. Elliot (1992), Active control of sound

Coherence function

Useful when the output is noisy, which is the case for all practical signals

Degree of dissimilarity of two signals

Coherence function

$$\gamma_{fg}^2(\omega) = \frac{|S_{fg}(\omega)|^2}{S_{ff}(\omega)S_{gg}(\omega)}$$

f(t) : input
g(t) : output

$$0 \leq \gamma_{fg}^2 \leq 1$$

If there is no source of noise corrupting output, coherence is 1

Signal-to-noise ratio

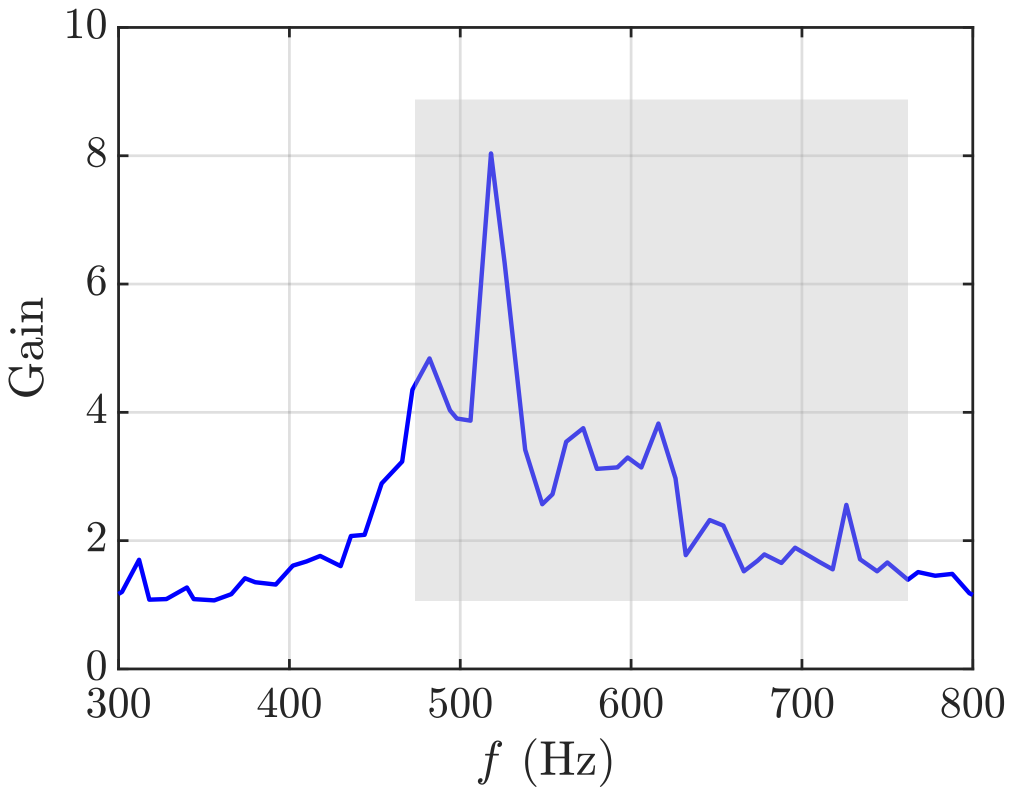
$$\frac{S_{gg}(\omega)}{S_{nn}(\omega)} = \frac{\gamma_{fg}^2(\omega)}{1 - \gamma_{fg}^2(\omega)}$$

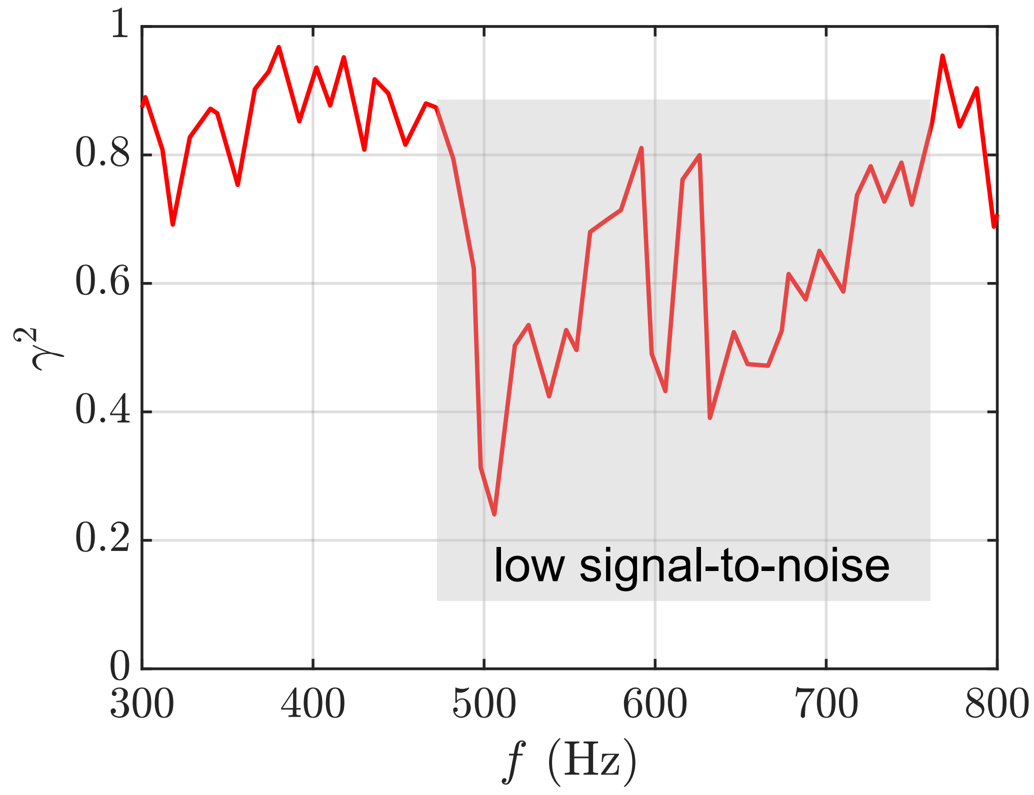
P. A. Nelson & S. J. Elliot (1992), Active control of sound

Coherence function

Coherence function

$$\gamma^2_{fg}(\omega) = \frac{|S_{fg}(\omega)|^2}{S_{ff}(\omega)S_{gg}(\omega)}$$





Flame transfer function

$$\mathcal{F}(\omega) = \frac{\dot{Q}'(\omega)/\overline{\dot{Q}}}{u'(\omega)/\overline{u}}$$

In premixed or quasi-premixed mode of combustion

Linear description cannot explain nonlinear features limit cycle oscillations, mode switching etc. that are commonly found in a system exhibiting a thermoacoustic instability

Flame describing function

Flame transfer function

$$\mathcal{F}(\omega) = \frac{\dot{Q}'(\omega)/\overline{\dot{Q}}}{u'(\omega)/\overline{u}}$$

Flame describing function[1,2] is an extension of flame transfer function, and it represents the flame response depending on the frequency and amplitude of the incoming velocity perturbations

Flame describing function

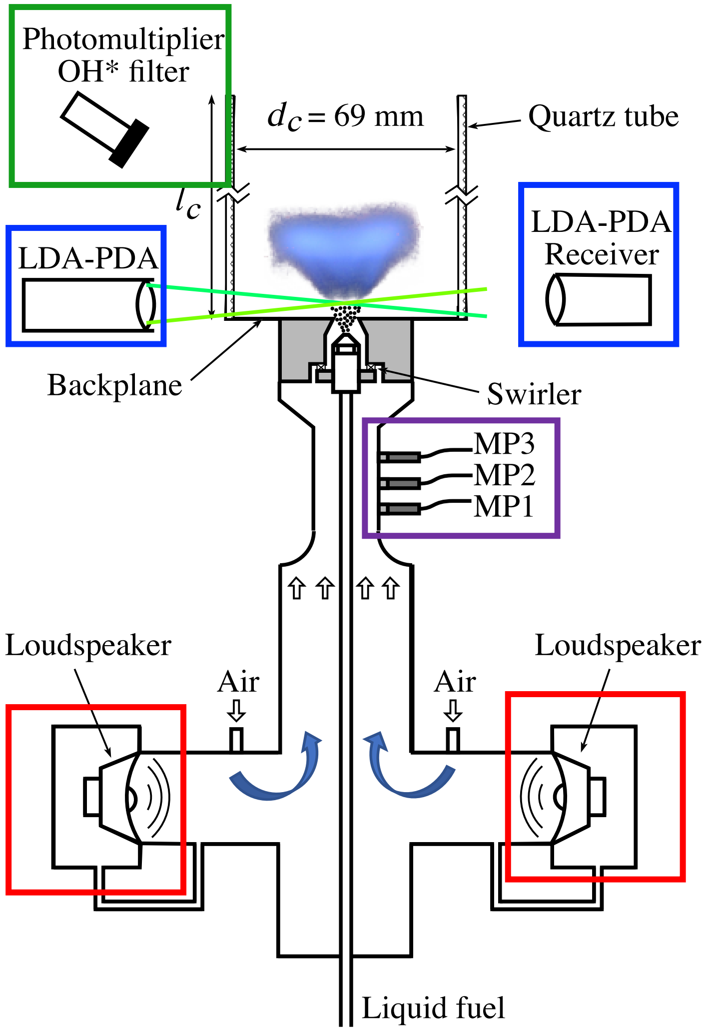
$$\mathcal{F}(\omega, |u'|) = \frac{\dot{Q}'(\omega, |u'|)/\overline{\dot{Q}}}{u'(\omega)/\overline{u}}$$

Gain & phase information can be obtained

[1] A. P. Dowling (1997) *J. Fluid Mechanics, 346.*
Non-linear self-excited oscillations of a ducted flame

[2] N. Noiray, D. Durox, T. Schuller, and S. Candel (2008) *J. Fluid Mechanics, 615.* A unified framework for nonlinear combustion instability analysis based on the flame describing function

SICCA-Spray combustor – experiments



Measuring equipments

Light intensity fluctuations – Photomultiplier with OH* filter
OH* chemiluminescence at 308 nm
Approximated to heat release rate fluctuations

Velocity fluctuations (injector exit) – Laser Doppler anemometry
Measures mean & fluctuating velocity

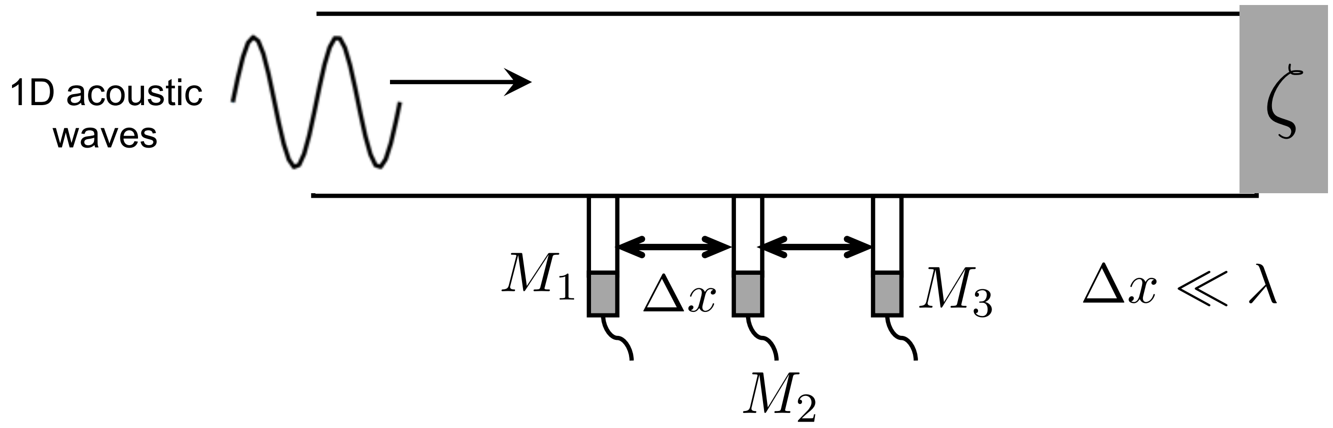
Velocity fluctuations (plenum) – Multi-microphone method
Calculate acoustic velocity from acoustic pressure signals

Input signal

Flow modulation – loudspeakers
Pulsations produced at different amplitude and frequencies

Multi-microphone method

Recall from POLKA's 2nd scientific workshop – Prof. Hans Boden



Two microphones are mostly sufficient
But with third microphone, fidelity can be improved

Acoustic momentum equation

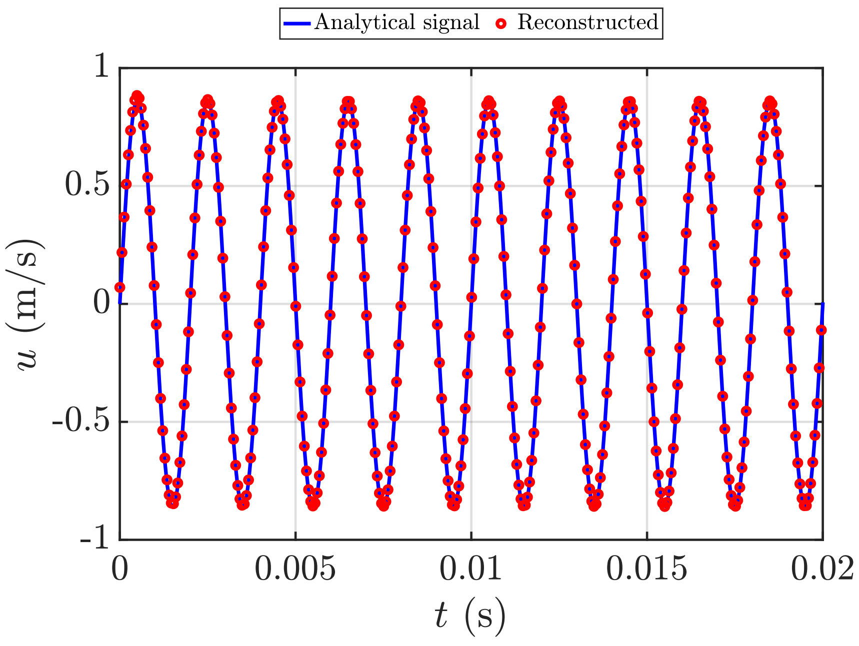
$$\frac{\partial u}{\partial t} = \frac{-1}{\rho_0} \frac{\partial p}{\partial x}$$

$$u \simeq \frac{-i}{\rho_0 \omega \Delta x}(p(x_2) - p(x_1))$$

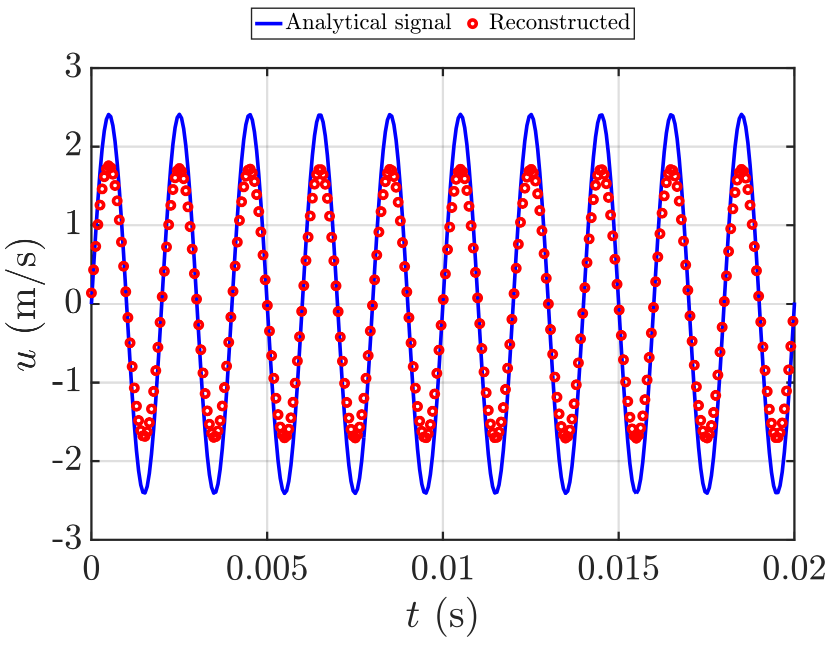
Analytical pressure signals using Hilbert transform is necessary

Multi-microphone method

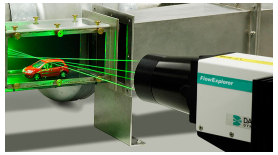
$\Delta x \ll \lambda$

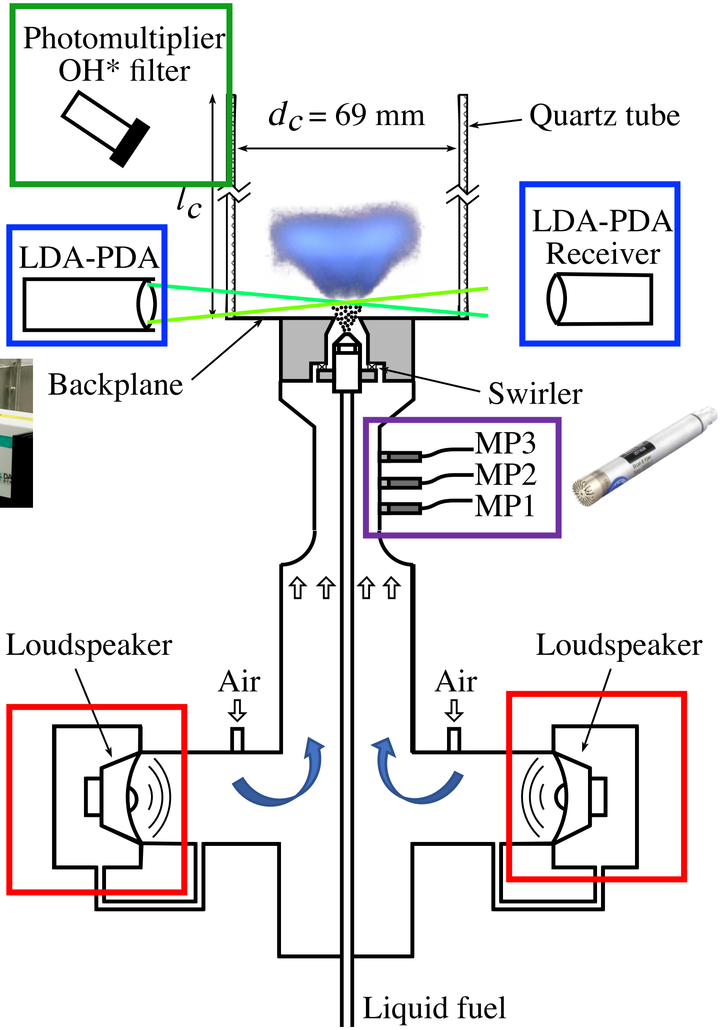


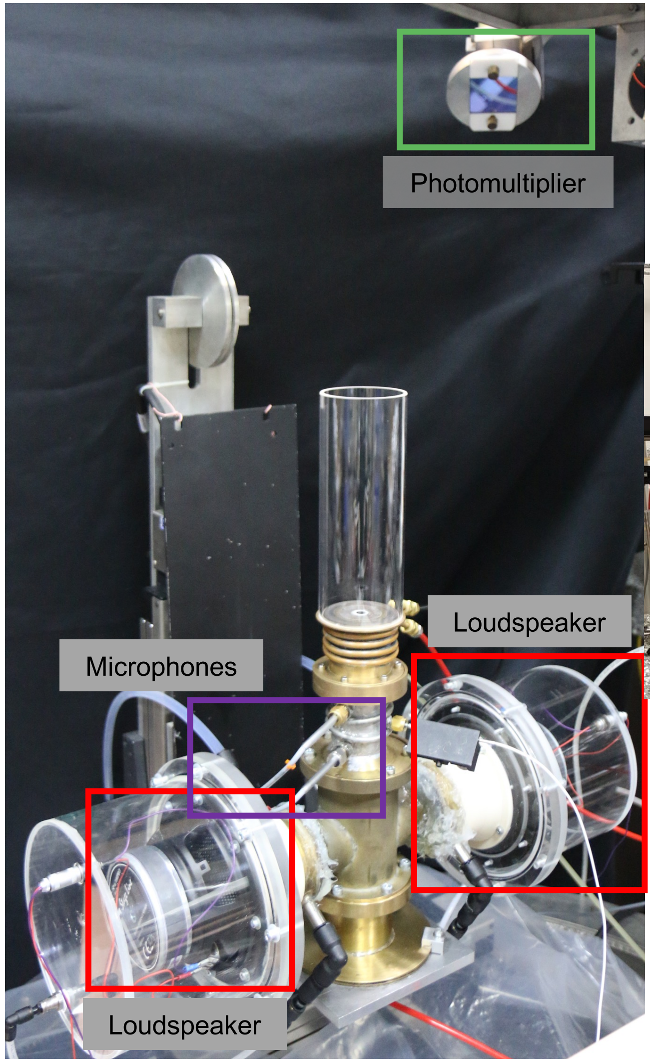
$\Delta x \simeq \lambda$



SICCA-Spray combustor – experiments







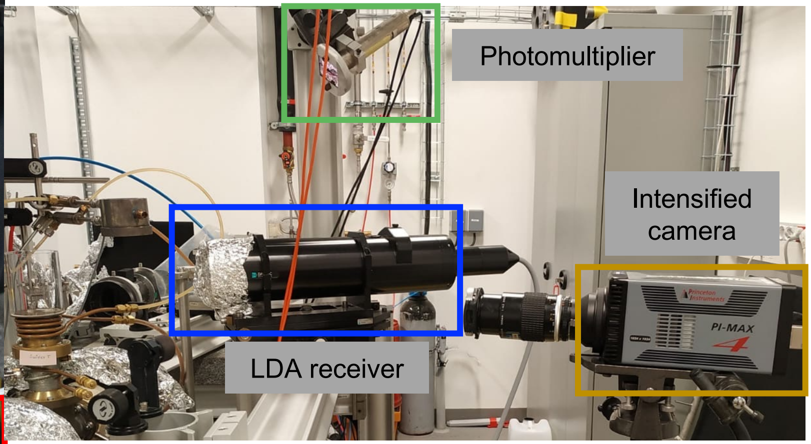
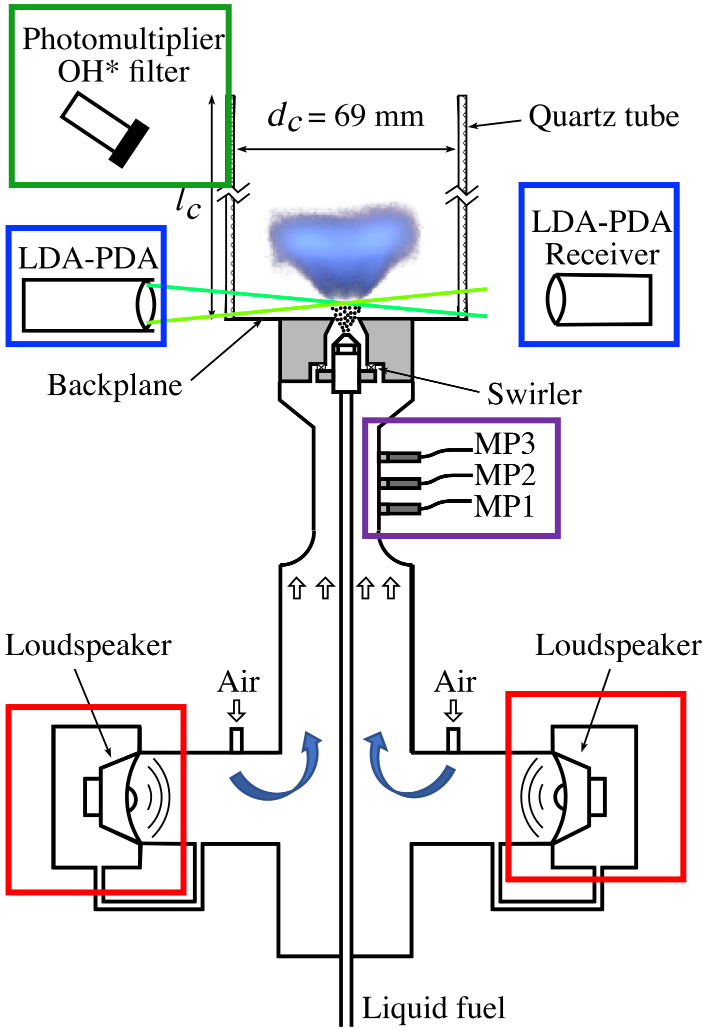


Image sources
Microphone : Bruel & Kjaer
Loudspeaker: Monacor
LDA systen: Dantec

SICCA-Spray combustor – experiments



In the input .mat file, look for

Photomultiplier signal: **PM** (V)

LDA signal (chamber velocity): **u** (m/s)

Microphones (plenum) – only two
MP1: **Pr2** - bottom mic (Pa)
MP3: **Pr4** - top mic (Pa)

Input signal to loudspeakers
Signal generator: **Gene** (V)
Generator frequency: **refFreq** (Hz)
Reference signal – sinusoidal

How is the measurement carried out?

Step 1: Set an amplifier voltage

Step 2: Sine frequency sweep from 250 to 850 Hz

Step 3: Change amplifier voltage

6 voltage levels

Amplifier voltages:
500, 1000, 1500, 2000, 2500, 3000 mV

Frequency ramp range: 250 Hz – 850 Hz
Frequency ramp rate: 4.5 Hz/s
Measurement acquired in blocks of 2s

Measurement obtained at different frequencies and amplitudes

Matlab exercise

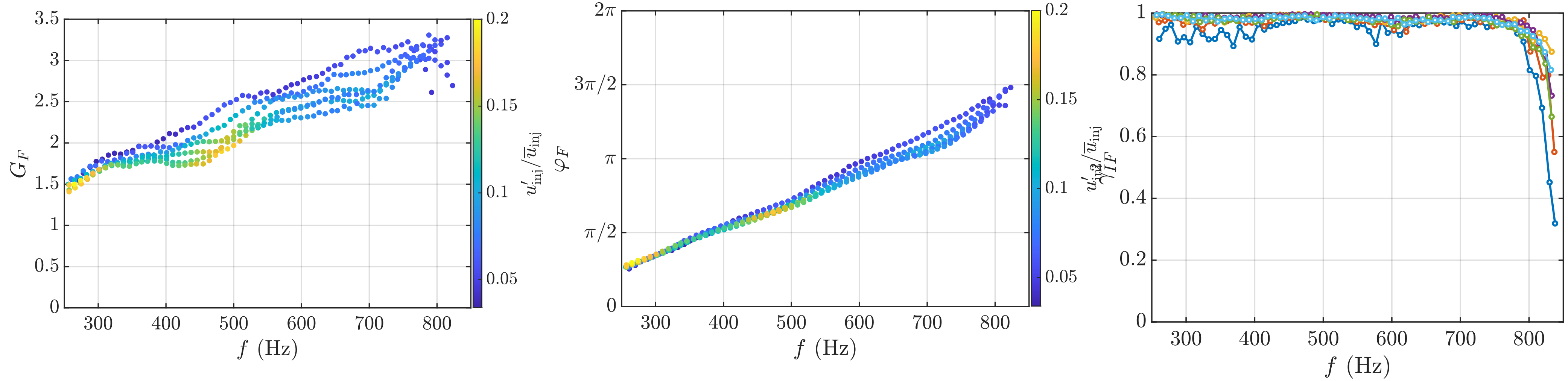
- Open the script: Exercise**POLKA_exercise.m**
- Data file: Exercise\Data
- Section that need to be completed are commented as “**TO BE COMPLTED**”
- For determining acoustic velocity **VelAcoustic.m** function should be completed

Calculating flame describing function

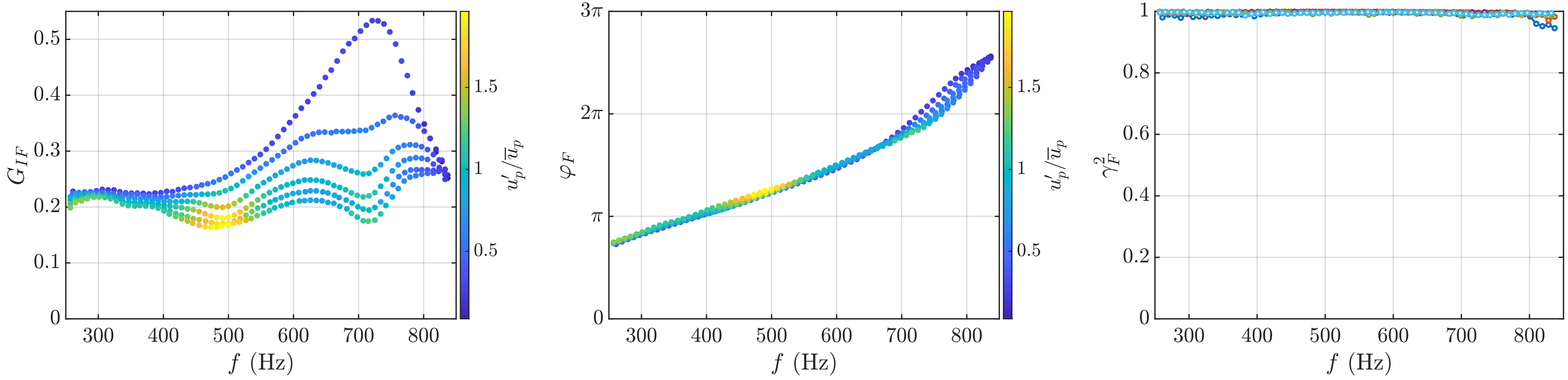
You are given:

- Microphone signals: **P2**, **P4** (Pa) in plenum
- Velocity signal: **u** (m/s) at injector exit
- Photomultiplier signal: **PM** (V)
- Reference signal from generator: **Gene** (V)
- Frequency of reference signal: **refFreq** (Hz)

Flame describing function



Injector + flame describing function



Step 1: Bandpass filtering

- Remove noise from measured signals
  - **Command**: bandpass
  - Specify **lower cut-off and upper cut-off** frequencies (already given)
  - Specify the **sampling rate** (Fs)
  - **Variables**: uBlock, PMBlock, Pr2Block, Pr4Block

```
% ------ Bandpass filtering (TO BE COMPLETED) ----- %
% use the reference frequency for bandpass filtering microphone, velocity
% and PM signals: filter 5% around the reference frequency
% Matlab command: bandpass

freqFilterLower = refFreq(i) - 5/100*refFreq(i);
freqFilterHigher = refFreq(i) + 5/100*refFreq(i);

% calculating mean values for normalizing transfer functions
uMeanInjector = mean(uBlock);
uBulkPlenum = 2.7; % m/s predetermined
PMMean = mean(PMBlock);

% use variables: uBlock for velocity
%                PMBlock for light intensity
%                Pr2Block for lower plenum microphone
%                Pr4Block for upper plenum microphone

uFilt = bandpass(uBlock,[freqFilterLower,freqFilterHigher],Fs); % filterin
PMFilt = bandpass(PMBlock,[freqFilterLower,freqFilterHigher],Fs); % filter
Pr2Filt = bandpass(Pr2Block,[freqFilterLower,freqFilterHigher],Fs); % filt
Pr4Filt = bandpass(Pr4Block,[freqFilterLower,freqFilterHigher],Fs); % filt
```

Step 2: Finding acoustic velocity in the plenum

Fill in **VelAcoustic.m** function

$$u \simeq \frac{-i}{\rho_0 \omega \Delta x}(p(x_2) - p(x_1))$$

top mic bottom mic

dX

dP Use **hilbertRev** function

$$u = \frac{\mathrm{imag}(\Delta P)}{2\pi f \rho \Delta x}$$

```
function [u] = VelAcoustic(pressure_bottom,pressure_top,dX,rho,freq)

% 1. Find pressure difference
% and determine the Hilbert transform using hilbertRev function

dP = hilbertRev(pressure_top-pressure_bottom); % hilbert transform w

% 2. Determine u using the formula
% dX - distance between microphones
% rho - density
% freq - frequency of the microphone signals
u = imag(dP) / (rho*dX*2*pi*freq);

end
```

Step 3: Calculating transfer functions

1. Calculate cross-power spectral density between velocity & heat release rate signals
    - **Command**: cpsd
    - **Variables**: input, output
    - **Welch's parameters**: dataWindow, noverlap, n_fft,Fs
2. Calculate power spectral density of velocity signals
    - **Command**: pwelch
    - **Variables**: input
    - **Welch's parameters**: dataWindow, noverlap, n_fft,Fs

```
% Transfer function - injector + flame

input = uPlenum./uBulkPlenum; % this is relative velocity fluctuations in
output = PMFilt./PMMean; % this is relative light intensity fluctuations f

% CPSD between input and output
[cpsdAmpIF,cpsdFreqIF] = cpsd(input,output,dataWindow,noverlap,n_fft,Fs);
[~,indexForMax] = max(abs(cpsdAmpIF));
cpsdPeakIF = cpsdAmpIF(indexForMax);
freqIF(i,vloop) = cpsdFreqIF(indexForMax);

% PSD of input
[psdAmpIF,psdFreqIF] = pwelch(input,dataWindow,noverlap,n_fft,Fs);
[~,indexForMax] = max(psdAmpIF);
psdPeakIF = psdAmpIF(indexForMax);
```

First do it for plenum signals → to obtain injector + flame transfer function

Then do it for chamber signals → to obtain flame transfer function

Step 3: Calculating transfer functions

- To calculation gain
  - **Command**: abs(cpsd/psd)
  - **Variables**: cpsdPeakIF, psdPeakIF for injector + flame transfer function
  - **Variables**: cpsdPeakF, psdPeakF for flame transfer function
- To calculation phase
  - **Command**: angle(cpsd)
  - **Variables**: cpsdPeakIF for injector + flame transfer function
  - **Variables**: cpsdPeakF for flame transfer function

```
gainIF(i,vloop) = abs(cpsdPeakIF/psdPeakIF);
phaseIF(i,vloop) = angle(cpsdPeakIF);
```

- To calculation coherence function
  - **Command**: abs(cpsd)^2/(psd-input * psd-output)
  - **Variables**: cpsdPeakIF, psdPeakIF, psdPeakOut
  - **Variables**: cpsdPeakF, psdPeakF, psdPeakOut

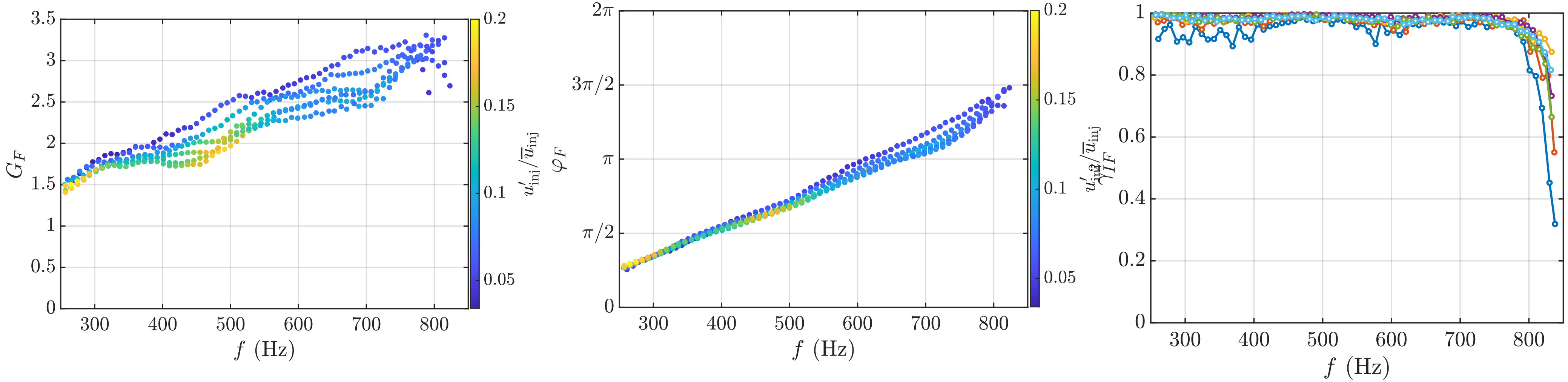
$$\gamma_{fg}^2(\omega) = \frac{|S_{fg}(\omega)|^2}{S_{ff}(\omega)S_{gg}(\omega)}$$

```
coherenceIF(i,vloop) = abs(cpsdPeakIF)^2/(psdPeakIF*psdPeakOut);
```

f(t) : input; g(t) : output

Results

Flame describing function



Injector + flame describing function