# LTPDA Training session

Topic 3: spectral analysis

Mauro Hueller

AEI Hannover, 09-10/03/2009

# Power Spectral Density Estimation (1)

Definition:

$$P_{xx}(f) = \frac{1}{f_s} \sum_{m=-\infty}^{+\infty} R_{xx}(m) \exp(-2\pi i \cdot f \cdot m / f_s)$$

Estimates the *one-sided* PSD:

$$\tilde{P}_{xx}(f) = \begin{cases} 0, & -\frac{f_s}{2} \le f \le 0 \\ 2P_{xx}(f), & 0 \le f \le \frac{f_s}{2} \end{cases}$$

# Power Spectral Density Estimation (2)

The PSD at each frequency is estimated via the Welch method:

Given a discretized signal $x[n]$ of length *N*

- Data are divided into segments of length *L* and multiplied by a window:

$$w(n)$$

this also reduces the edge-effects (simulating a periodic sequence)

The PSD at each frequency *f* is estimated as:

$$\hat{P}_{xx}(f) = \frac{\left|X_L\right|^2}{f_s \cdot L \cdot U}$$

where

$$x_L[n] = x[n]w[n]$$

$$x_L(f) = \sum_{n=0}^{L-1} x_L[n]\exp\left(-2\pi i \cdot \frac{f \cdot n}{f_s}\right)$$

$$U = \frac{1}{L}\sum_{n=0}^{L-1}\left|w(n)^2\right|$$

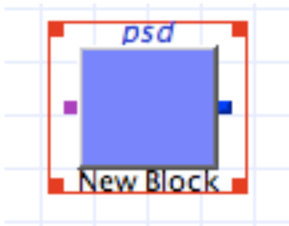
# Power Spectral Density Estimation (3)

- Methods:
  - `ao/psd`: linear frequency scale
  - `ao/lpsd`: log-frequency scale
  - `specwin`: implements spectral windows

# Power Spectral Density Estimation (4)

```
a = [ao with time-series data]
S = a.psd(plist('win',win,…
  'nfft',nfft,'olap',olap,…
  'order',order,'scale',scale))
```

Or add a block on a workbench:



# Power Spectral Density Parameters

| Name | Description | Values | Default |
|---|---|---|---|
| scale | the output quantity | ‘PSD’ gives Power Spectral Density [m^2 Hz^-1]<br>‘ASD’ gives Amplitude Spectral Density [m Hz^-1/2]<br>‘PS’ gives Power Spectrum [m^2]<br>‘ASD’ gives Amplitude Spectrum [m] | ‘PSD’ |
| win | A spectral window to multiply the data by | ‘BH92’ or ‘Rectangular’ or …<br>(name or object) | Taken from user preferences |
| nfft | Window length | -1: one window, length = ao data set<br>Or: length (number of points) | -1 |
| order | Order of segment detrending (prior to windowing) | -1: no detrending<br>0: mean subtraction<br>N: order N polynomial trend subtraction | 0 |
| olap | Percentage overlap between adjacent segments | -1: taken from window parameters<br>0: no overlap<br>100: total overlap | -1 |

# Power Spectral Density Estimation (5)

Features:

- Multiple inputs:

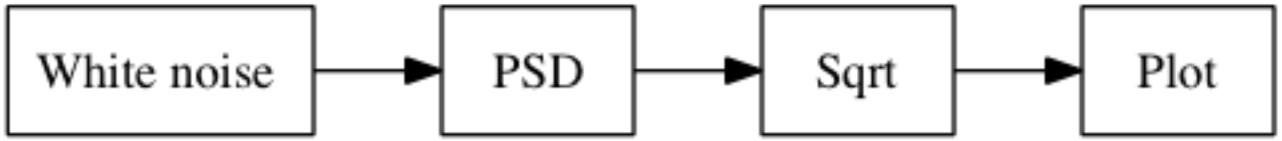
```
S = psd(a1,a2,a3,plist())
```

- Matrix inputs:

```
S = psd([a1 a2;a3 a4],plist())
```

# PSD Exercise 1

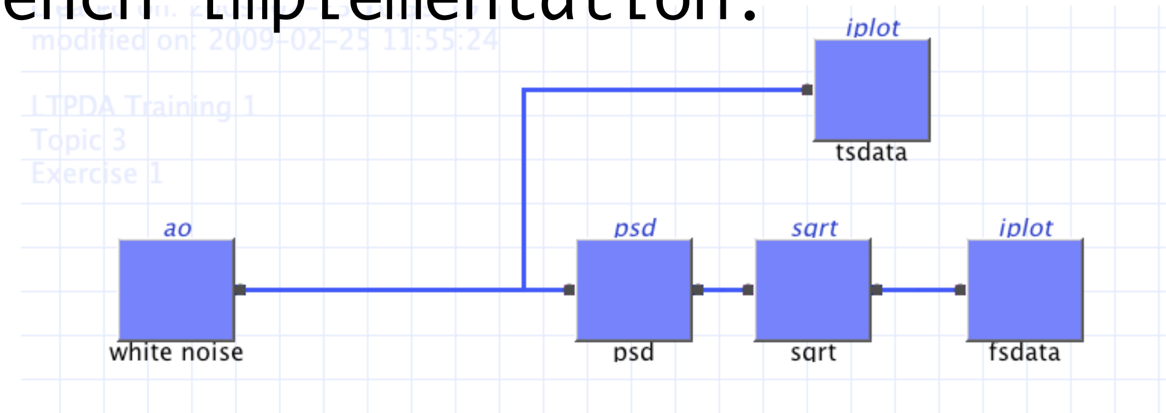
Very simple idea:




Parameters for psd:
all default

# PSD Exercise 1

- Workbench implementation:



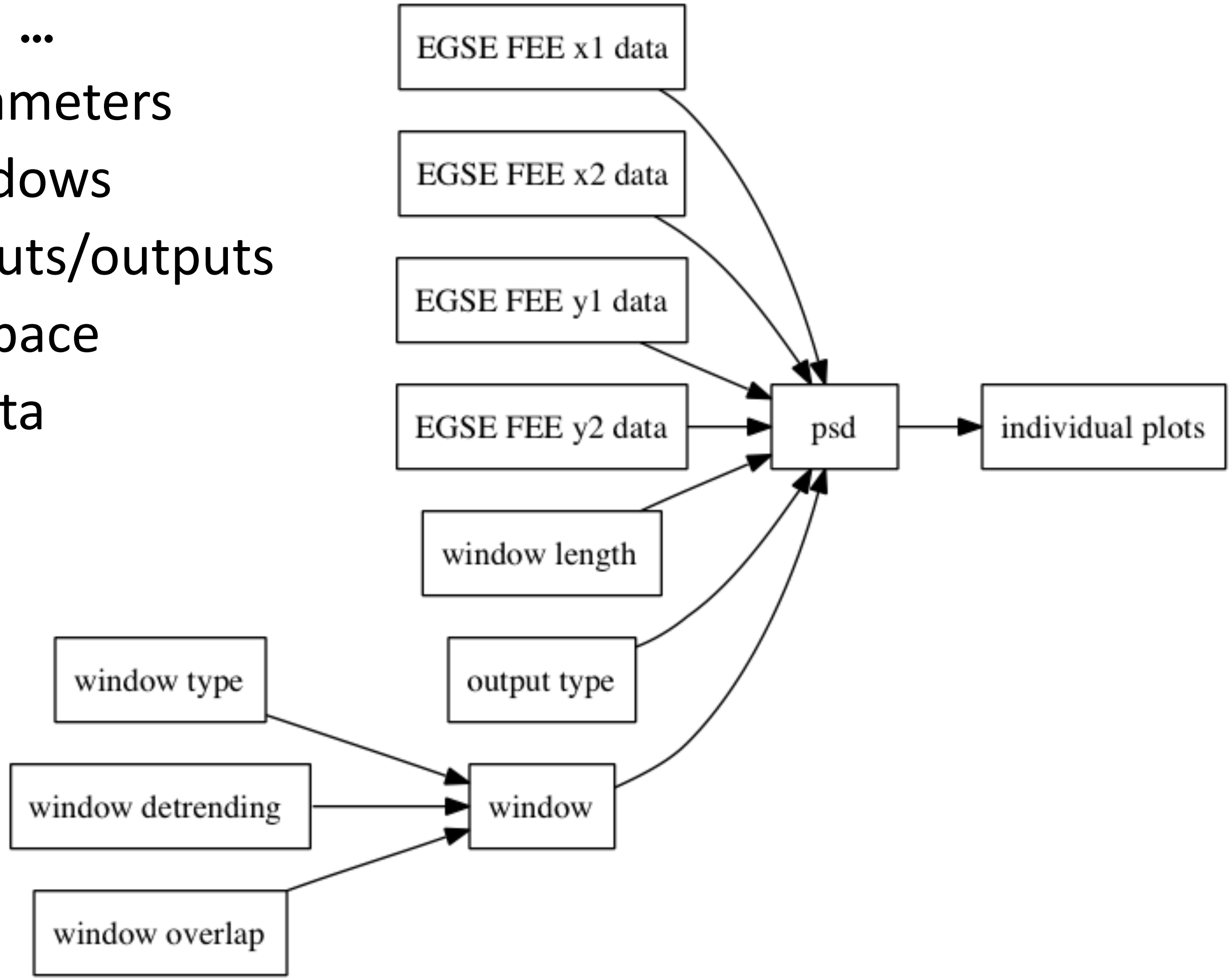

- Matlab terminal implementation:

```
>> a1 =
   ao(plist('waveform','noise','type','normal','nsecs',
   1000,'sigma',1.0,'yunits','m'))
>> iplot(a1)
>> S1 = a1.psd(plist('win','BH92'))
>> P1 = sqrt(S1)
>> iplot(P1)
```

# PSD Exercise 2

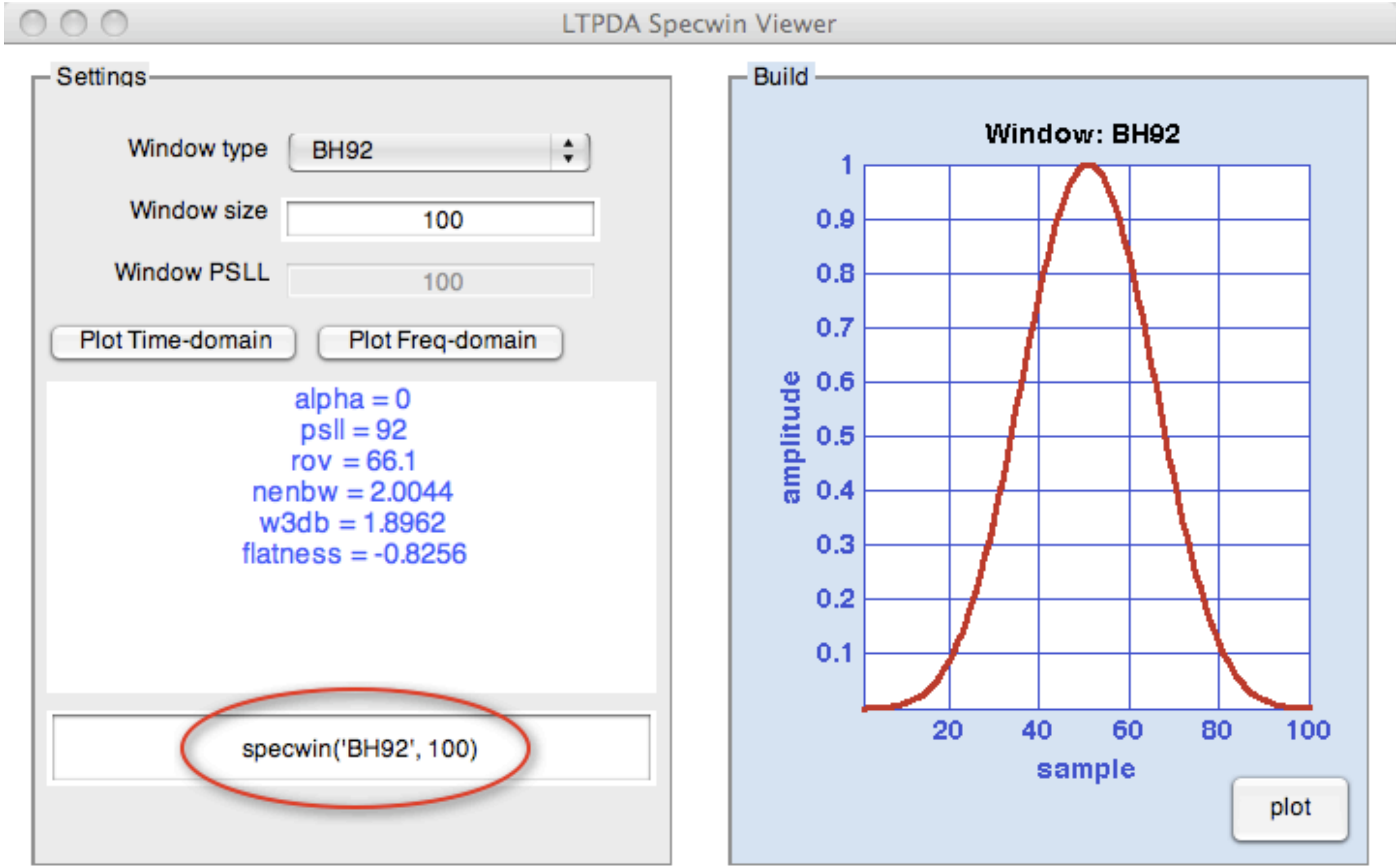
More involved …

- Playing with parameters
- Playing with windows
- Adding block inputs/outputs
- Output to workspace
- Saving output data



# PSD Exercise 2

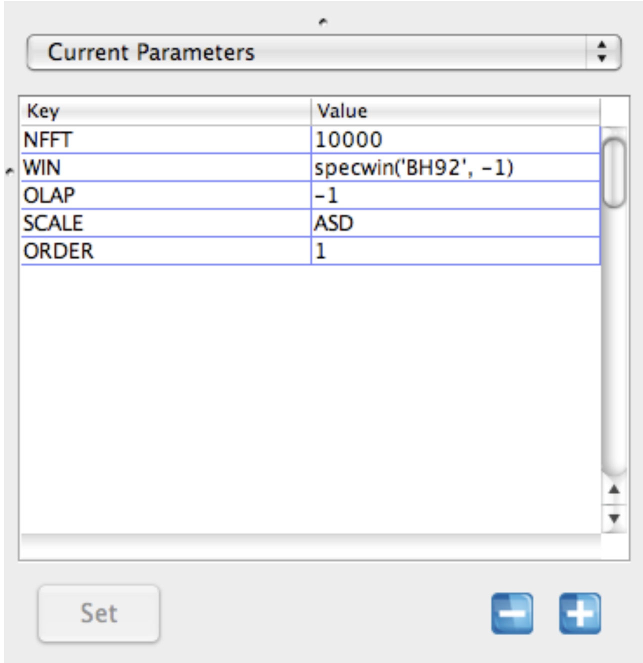
## Spectral windows

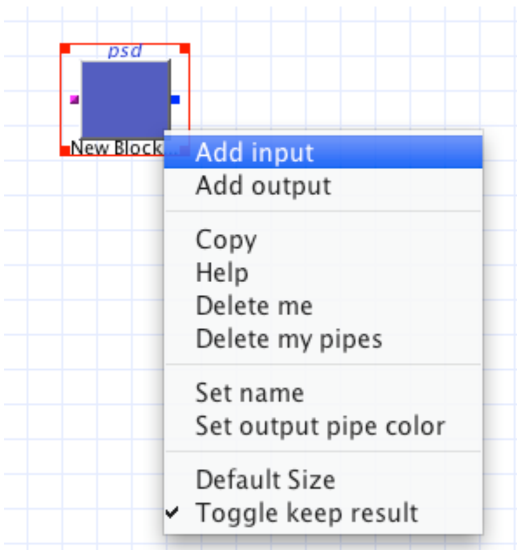


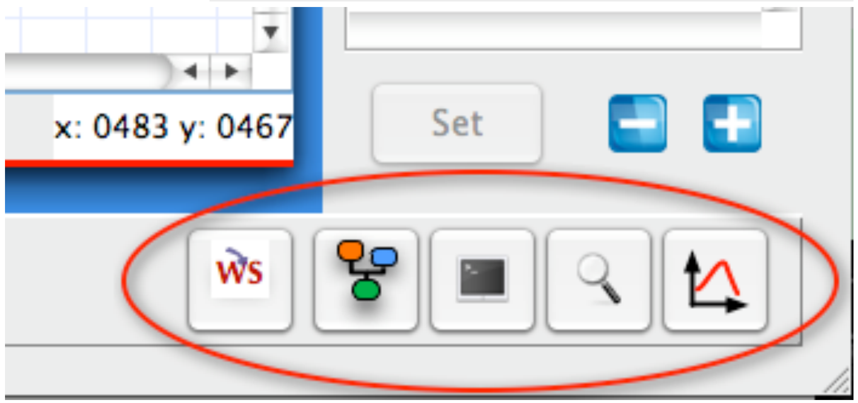
# PSD Exercise 2

More involved …

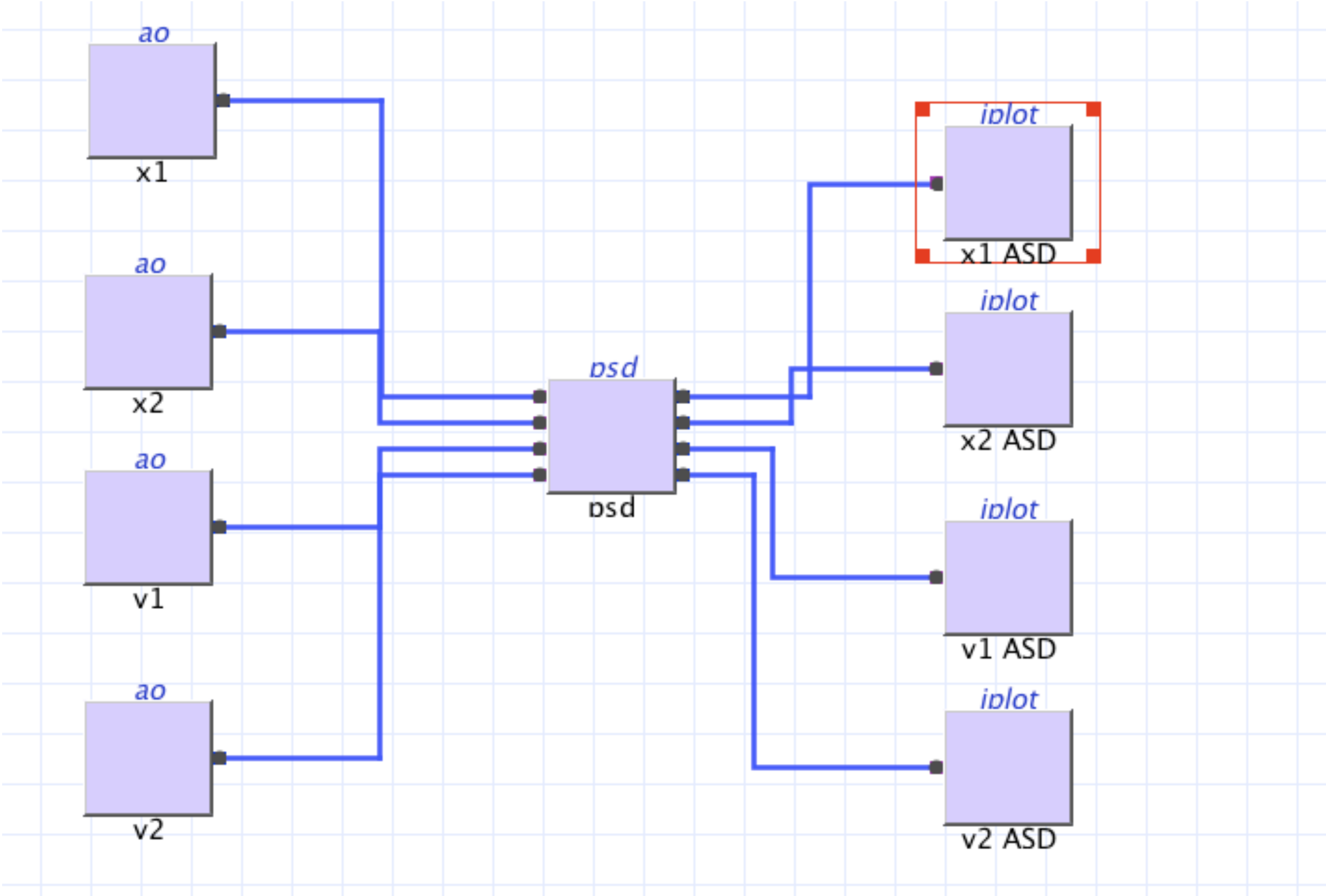
- Playing with parameters
- Adding inputs/outputs to blocks
- Output to workspace
- Saving output data



# PSD Exercise 2



# Log-Scale Power Spectral Density Estimation

Implementation of the algorithm described in
"Measurement 39 (2006) 120-129"

The same as `psd` but:

- Reduces individual point variance by adjusting the window length at each frequency
- Frequency bins and number of averages are calculated automatically
- Slower because of the much higher number of DFT evaluation
- Energy content of the spectrum is preserved
- Reduced resolution due to shorter window length
- Lower uncertainty

# Log-Scale Power Spectral Density Parameters

The same as `psd` but with:

| Name | Description | Values | Default |
|---|---|---|---|
| Kdes | Desired number of averages | An integer number | 100 |
| Jdes | Number of spectral frequencies to calculate | An integer number | 1000 |
| Lmin | Minimum segment length | An integer number | 0 |

And `nfft` has no meaning, that is calculated for each frequency

# Log-scale Power Spectral Density Estimation

Features:

- Multiple inputs:

```
S = lpsd(a1,a2,a3,plist())
```

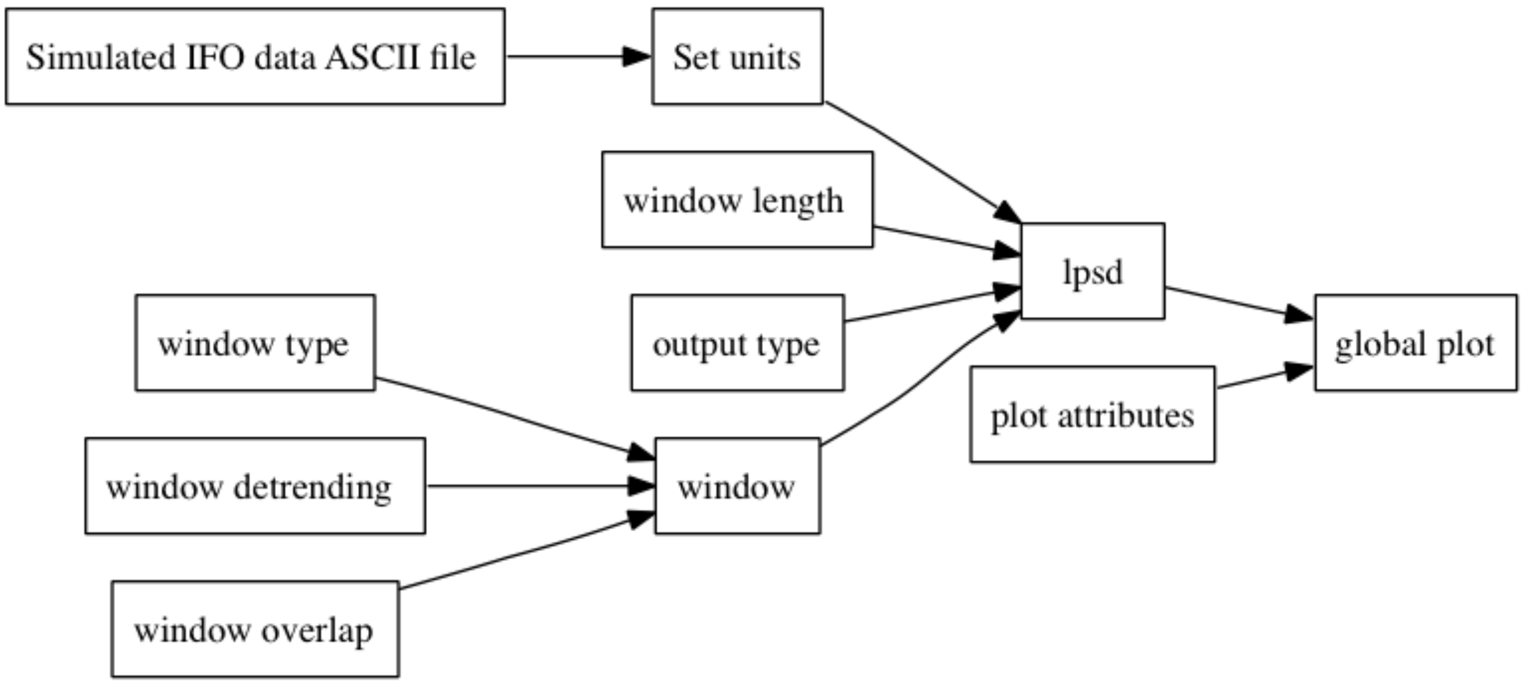
- Matrix inputs:

```
S = lpsd([a1 a2;a3 a4],plist())
```

# PSD Exercise 3

`Using lpsd`

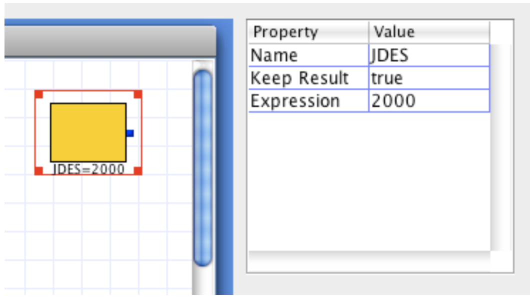
- Log-scale psd calculation to reduce variance
- Using MDC1 IFO data
- Setting units
- Setting plot features

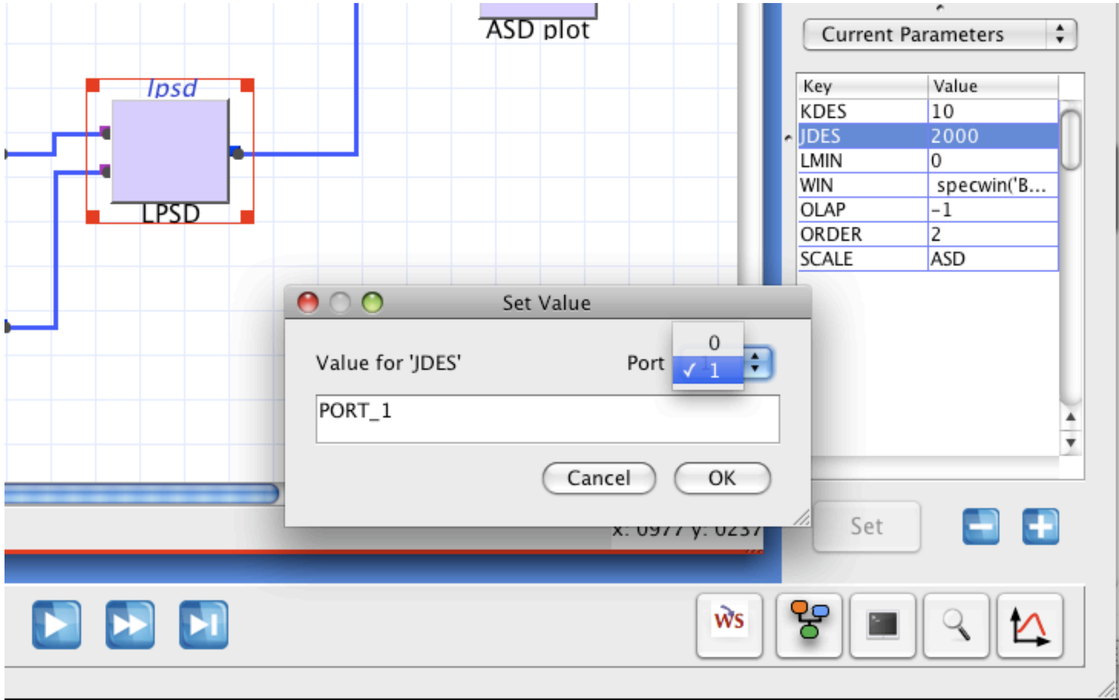


# PSD Exercise 3

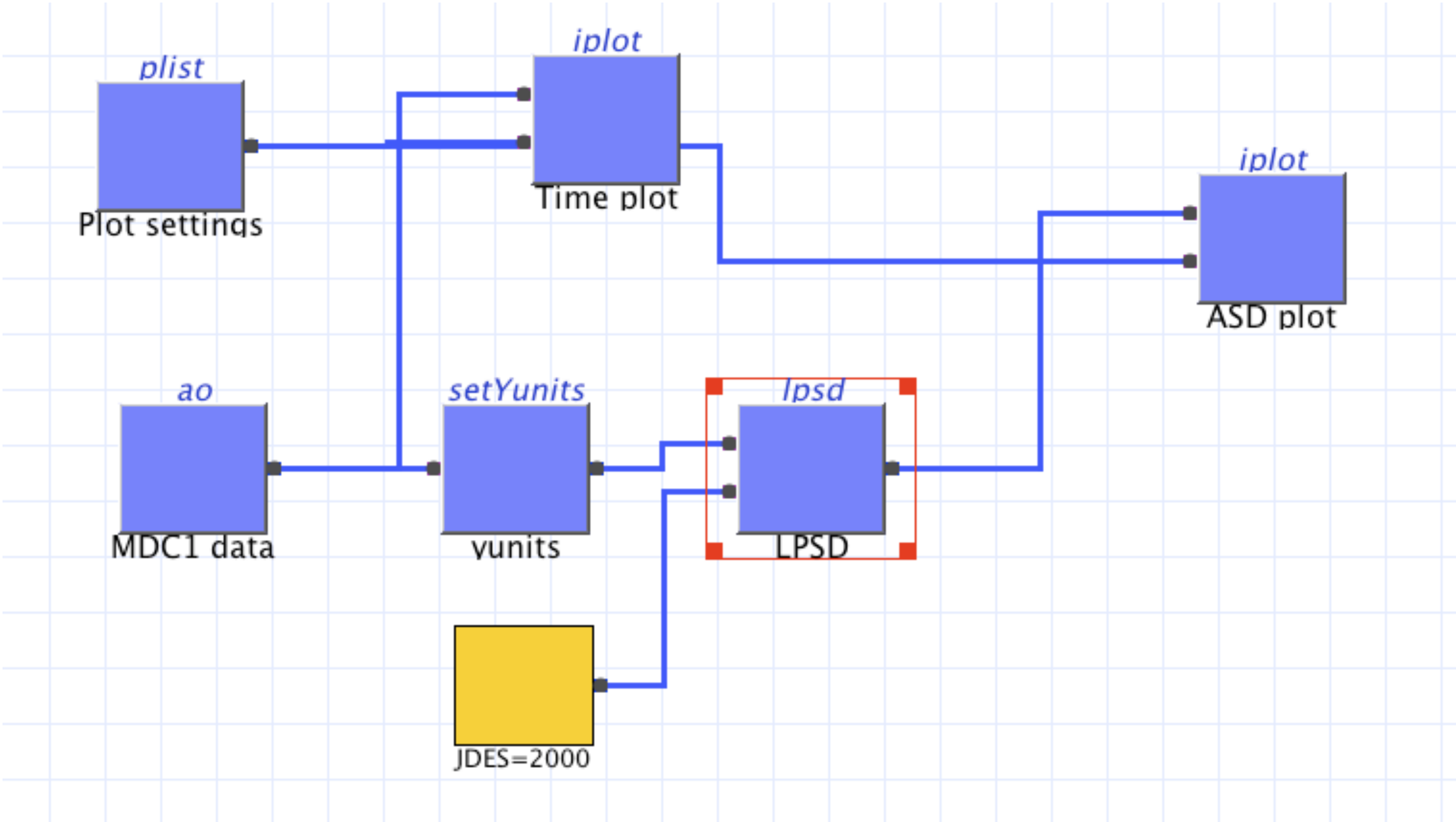
Using `lpsd`

- Log-scale psd calculation to reduce variance
- Using MDC1 IFO data
- Setting units
- Setting plot features
- Adding constant blocks



# PSD Exercise 3



# Cross- Power Spectral Density Estimation

Definition:

$$C_{xy}\left(f\right)=\frac{1}{f_s}\sum_{m=-\infty}^{+\infty}R_{xy}\left(m\right)\exp(-2\pi i\cdot f\cdot m/f_s)$$

Estimates the *one-sided* CPSD:

$$\tilde{C}_{xy}\left(f\right)=\begin{cases}0, & -\frac{f_s}{2}\le f\le 0\\ 2P_{xy}\left(f\right), & 0\le f\le\frac{f_s}{2}\end{cases}$$

# Cross- Power Spectral Density Estimation (2)

The CPSD at each frequency is estimated via the Welch method:

Given two discretized signals $x[n]\ y[n]$ of length *N*

Data are divided into segments of length *L* and multiplied by a window:

$$w(n)$$

this also reduces the edge-effects (simulating a periodic sequence)

The CPSD at each frequency *f* is estimated as:

$$\hat{C}_{xy}(f) = \frac{X_L Y_L^*}{f_s \cdot L \cdot U}$$

where

$$x_L(f) = \sum_{n=0}^{L-1} x_L[n] \exp\left(-2\pi i \cdot \frac{f \cdot n}{f_s}\right) \qquad U = \frac{1}{L}\sum_{n=0}^{L-1} \left| w(n)^2 \right|$$

# Cross- Power Spectral Density Estimation (3)

- Methods:
  - `ao/cpsd`: linear frequency scale
  - `ao/lcpsd`: log-frequency scale

Similarly, we can evaluate *coherence*:

$$Coh_{xy}(f) = \frac{\left|C_{xy}(f)\right|^2}{P_{xx}(f)P_{yy}(f)}$$

- Methods:
  - `ao/cohere`: linear frequency scale
  - `ao/lcohere`: log-frequency scale

# Cross- Power Spectral Density Estimation (4)

```
C = cpsd(x,y,plist('win',win,…
  'nfft',nfft,'olap',olap,…
  'order',order,'scale',scale))
Cxy = index(C,1,2)
Cyx = index(C,2,1)
Cxx = index(C,1,1)
```

# Cross- Power Spectral Density Parameters

| Name | Description | Values | Default |
|---|---|---|---|
| win | A spectral window to multiply the data by | 'BH92' or 'Rectangular' or … name or object | Taken from user preferences |
| nfft | Length of the window | -1: one window, length = ao data set length<br>Or: length (number of points) | -1 |
| order | Order of segment detrending (prior to windowing) | -1: no detrending<br>0: mean subtraction<br>N: order N polynomial trend subtraction | 0 |
| olap | Percentage overlap between adjacent segments | -1: taken from window parameters<br>0: no overlap<br>100: total overlap | -1 |

And similarly for `lcpsd` …

# Transfer Function Estimation

Definition:

$$T_{xy}(f) = \frac{C_{xy}(f)}{P_{yy}(f)}$$

The TFE at each frequency is estimated via the Welch method

- Methods:
  - `ao/tfe`: linear frequency scale
  - `ao/ltfe`: log-frequency scale

# Transfer Function Estimation (2)

```
T = tfe(x,y,plist(‘win’,win,…
  ‘nfft’,nfft,‘olap’,olap,…
  ‘order’,order,‘scale’,scale))
Txy = index(C,1,2)
Tyx = index(C,2,1)
Txx = index(C,1,1)
```

# Transfer Function Estimation Parameters

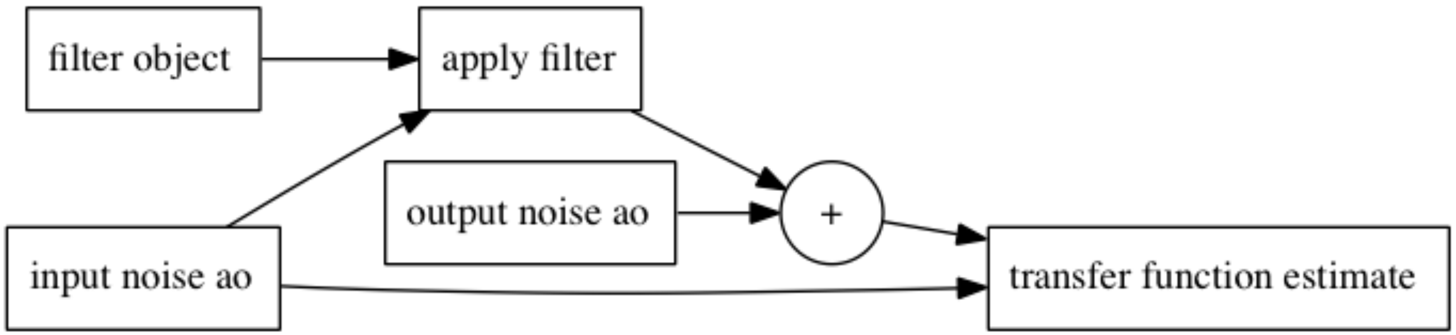
| Name | Description | Values | Default |
|---|---|---|---|
| win | A spectral window to multiply the data by | 'BH92' or 'Rectangular' or … name or object | Taken from user preferences |
| nfft | Length of the window | -1: one window, length = ao data set length<br>Or: length (number of points) | -1 |
| order | Order of segment detrending (prior to windowing) | -1: no detrending<br>0: mean subtraction<br>N: order N polynomial trend subtraction | 0 |
| olap | Percentage overlap between adjacent segments | -1: taken from window parameters<br>0: no overlap<br>100: total overlap | -1 |
| Variance | Computes transfer function variance | 'yes' or 'no' | 'no' |

And similarly for `ltfe` …

# TFE Exercise 1

Using `tfe`

- Simulated data input: white noise
- Band-pass filter object
- Filtering the input noise
- Adding output white noise
- Estimate the transfer function

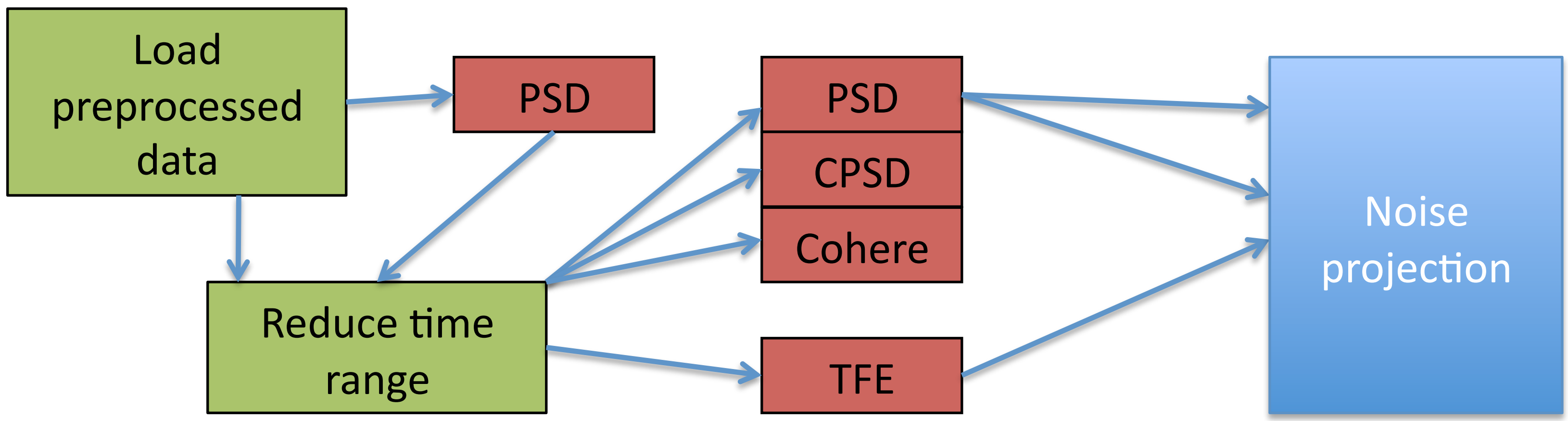


# IFO/Temperature Example

Estimating the empirical transfer function

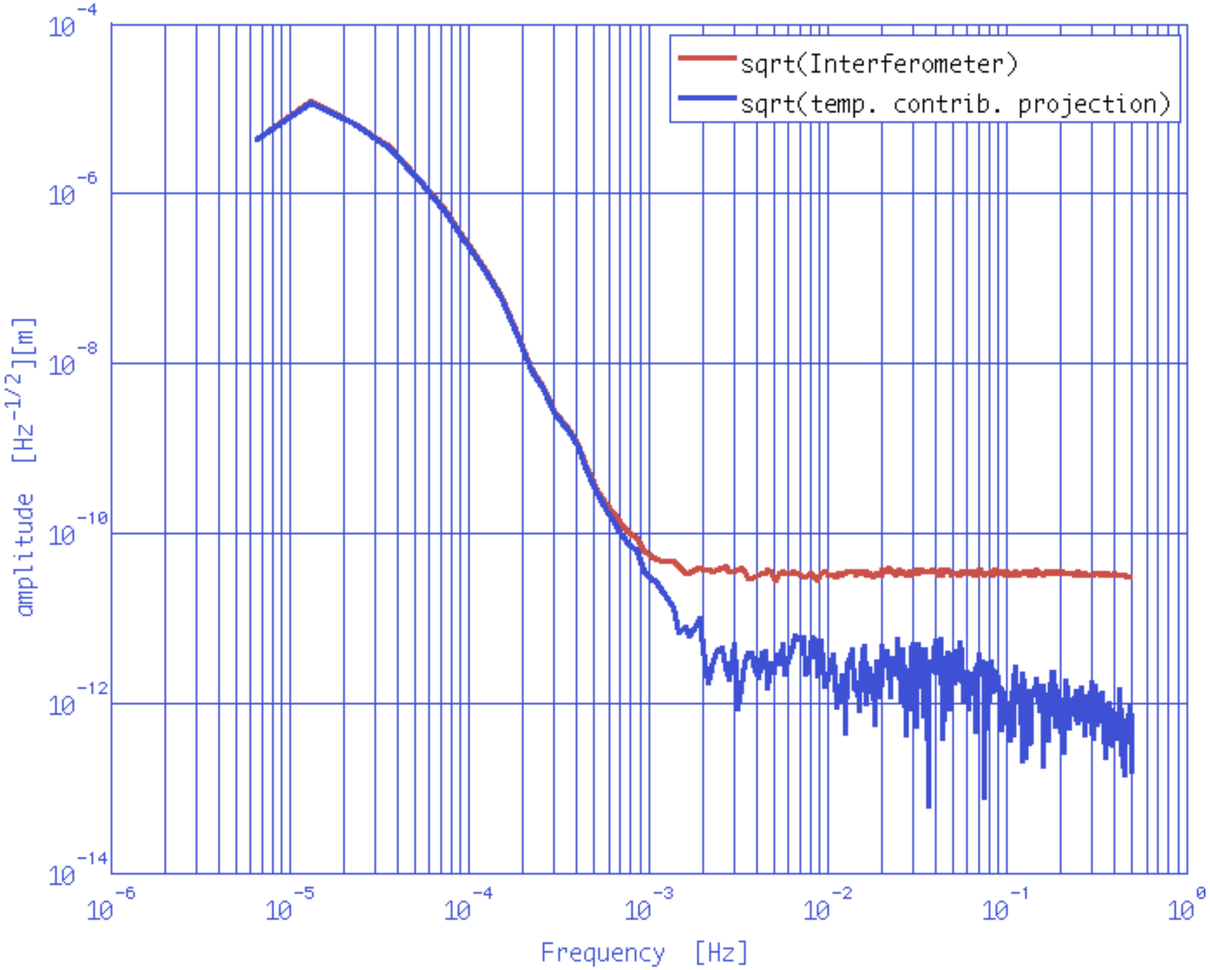
temperature -> position

- Load preprocessed data
- Evaluate PSD of *T* and *x*
- Reduce the time range
- Evaluate CPSD and cross-coherence of *T* and *x*
- Estimate the transfer function of *T* into *x*
- Perform the noise projection



# IFO/Temperature Example

- We are aiming to obtain:



# Changes after R2.0

- cpsd, lcpsd, cohere, lcohere, tfe,ltfe will output a *single* object

```
Cxy = cpsd(x,y,plist('nfft', 1000));
```

- All methods will include variance (if number of windows > 1) in the *procinfo* field