# Example_OSSE_obs_NOT_mapping_NATL60

April 18, 2019

## 0.1 CASE STUDY 1: COMPARISON WITH INDEPENDENT ALONG-TRACK

**INTRODUCTION**

We here propose to show the behaviour of the spectral analysis in an OSSE case study based on NATL60 simulation. The input comes from hourly SSH data from a 1 year long period of the run NATL60-CJM165 made at IGE (Grenoble). We used the SWOTsimulator to sample alongtrack data (nadir-like dataset) on this simulation. The constellation is similar to the 20030101-20031231 period: Jason 1, Envisat, Geosat2, Topex/Poseidon interleaved. SWOTsimulator gives us the true SSH ($SSH_{model}$) and a pseudo-obs SSH ($SSH_{obs}$ = $SSH_{model}$ + white noise). These pseudo-obs $SSH_{obs}$ are then used in the DUACS mapping. This case study 1 is similar to case study 2 except that we are doing here the evaluation of the maps against indepedent along-track data.

**CASE STUDY #1 SUMMARY:** - the along-track of reference (i.e., for comparison) is **Geosat2**. In this case study #2, this along-track **is NOT included** in the mapping. - the $SSH_{map}$ are generated from **3 nadir altimeters** TPN, J1, EN with the DUACS system - we focus on the region in the north-atlantic bassin **(10řx10ř box centered near 330řE-44řN)** - this case study is similar to the approach used in the manuscript

Note that the ratio PSD(mapping_error)/PSD($SSH_{model}$) can gives access to the estimation of the map resolution. However, the DUACS maps and along-track products gives us access to only: PSD($SSH_{map}$), PSD($SSH_{obs}$) and PSD(intrumental_error). We will have to play with this three latter quantities to estimate the resolution. In the study cases 1 and 2, we shows that the PSD($SSH_{map}$), PSD($SSH_{obs}$) and PSD(intrumental_error) can estimate the resolution and that it is in good agreement with the ratio PSD(mapping_error)/PSD($SSH_{model}$).

PYTHON MODULE

```
In [1]: from netCDF4 import Dataset
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy import signal, interpolate
        from matplotlib.ticker import FormatStrFormatter
```

**READ THE SSH SEGMENTS**

*Note*: The SSH segments were generated using same algorithm as used for the study described in manuscript. There are 800km long, referenced by their mean latitude and mean longitude and they overlap over 25%

**NOTATION**

- **$SSH_{model}$**: alongtrack SSH from model
- **$SSH_{obs}$**: alongtrack SSH from model including white noise

- **SSH**$_{map}$: SSH mapped with the DUACS algorithm using 3 altimeters data, and interpolated onto the alongtrack SSH$_{model}$ path
- **intrumental_error**: SSH$_{obs}$ minus SSH$_{model}$
- **mapping_error**: SSH$_{model}$ minus SSH$_{map}$

In [2]:
```
nc = Dataset('./data/dataset_independent_alongtrack_G2.nc','r')
ssh_model = nc.variables['ssh_model'][:]
ssh_obs = nc.variables['ssh_obs'][:]
ssh_map = nc.variables['ssh_map'][:]
dx = nc.variables['resolution'][:]
nperseg = nc.dimensions['segment_size'].size
nc.close()
```

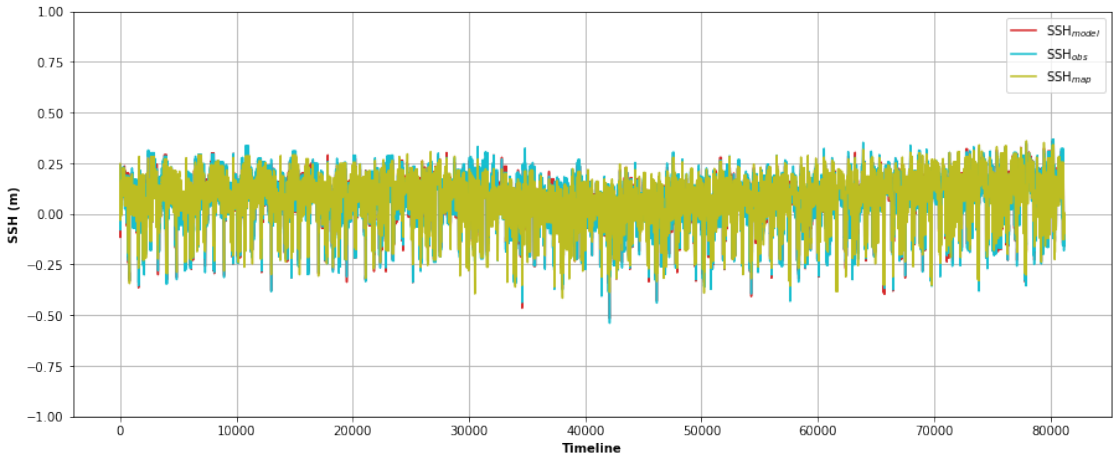
**COMPUTE INSTRUMENTAL ERROR & MAPPING ERROR**

In [3]:
```
intrumental_error = ssh_obs - ssh_model
mapping_error = ssh_model - ssh_map
```

**QUANTITY THAT CAN BE EVALUATED FROM ALTIMETRY DATA** : SSH$_{map}$ - SSH$_{obs}$

In [4]:
```
# What can evaluated from altimetry product (we don't have access to ssh_model)
ssh_obs_minus_map = ssh_obs - ssh_map
```

In [5]:
```
plt.figure(figsize=(15, 6))
plt.xlabel("Timeline", fontweight='bold')
plt.ylabel("SSH (m)", fontweight='bold')
plt.plot(ssh_model, c='C3', label='SSH$_{model}$')
plt.plot(ssh_obs, c='C9', label='SSH$_{obs}$')
plt.plot(ssh_map, c='C8', label='SSH$_{map}$')
plt.ylim(-1, 1)
plt.legend(loc='best')
plt.grid()
plt.show()
```

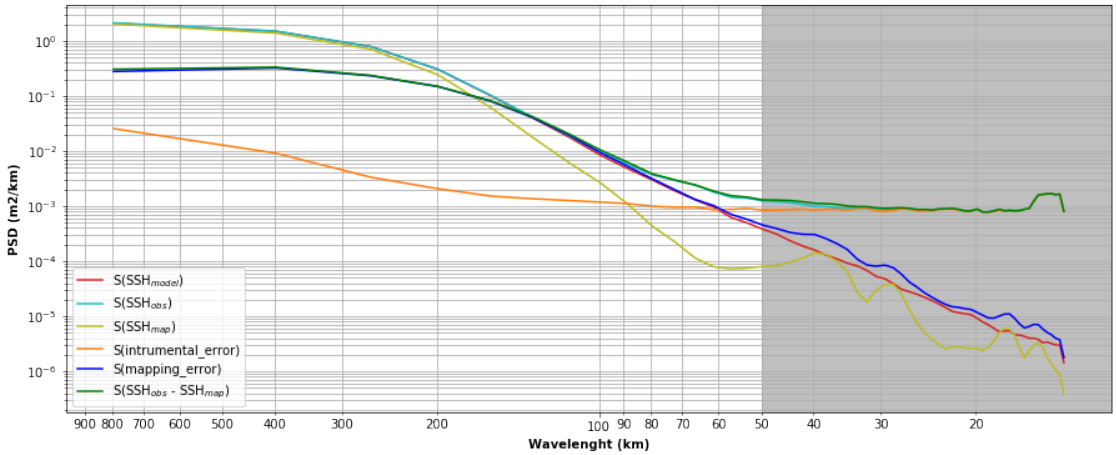



**SPECTRAL CONTENT OF EACH SIGNAL**

In [6]:
```
freq, psd_ssh_model = signal.welch(ssh_model,
                                   fs=1/dx,
                                   nperseg=nperseg,
                                   scaling='density',
                                   noverlap=0)

_, psd_ssh_obs = signal.welch(ssh_obs,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_ssh_map = signal.welch(ssh_map,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_intrumental_error = signal.welch(intrumental_error,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)

_, psd_mapping_error = signal.welch(mapping_error,
                                    fs=1/dx,
                                    nperseg=nperseg,
                                    scaling='density',
                                    noverlap=0)

_, psd_ssh_obs_minus_map = signal.welch(ssh_obs_minus_map,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)
```

In [7]:
```
fig, ax = plt.subplots(figsize=(15, 6))
plt.plot(1/freq, psd_ssh_model, c='C3', label='S(SSH$_{model}$)')
plt.plot(1/freq, psd_ssh_obs, c='C9', label='S(SSH$_{obs}$)')
plt.plot(1/freq, psd_ssh_map, c='C8', label='S(SSH$_{map}$)')
plt.plot(1./freq, psd_intrumental_error, c='C1', label='S(intrumental_error)')
plt.plot(1./freq, psd_mapping_error, c='b', label='S(mapping_error)')
plt.plot(1./freq, psd_ssh_obs_minus_map, c='g', label='S(SSH$_{obs}$ - SSH$_{map}$)')
plt.gca().invert_xaxis()
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('PSD (m2/km)', fontweight='bold')
```

```
plt.legend(loc='best')
plt.grid(which='both')
ax.axvspan(0, 50, color='grey', alpha=0.5)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.show()
```




The figure represents the power spectral densities from each type of signal. It illustrates a good agreement of the power spectral density S(SSH$_{obs}$), S(SSH$_{map}$) and S(SSH$_{model}$) for wavelength > 200-300 km. For wavelength < 200 km the power spectral density of the SSH$_{map}$ is less energetic than for the SSH$_{obs}$, linked to the filtering properties of te DUACS mapping system.

The signal-noise-ratio for the input along-track is between 50-60km (crossing of the red and orange lines), setting the along-track resolution limit as defined in Dufau et al (2016).

For wavelenght greater than this along-track resolution limit, the power spectral density of the mapping error S(mapping_error) is larger than the power spectral density of the instrumental error S(instrumental_error).

The power spectral density S(SSH$_{map}$) for wavelenght smaller than this along-track resolution limit (approx. grey area) is below the grid spacing of the DUACS maps. The power spectral density S(SSH$_{map}$) below this limit is hence resulting from the interpolation of the SSH$_{map}$ onto the alongtrack position

**CROSS SPECTRUM**

```
In [8]: freq, cross_psd_ssh_map_ssh_obs = signal.csd(ssh_map,
                                                    ssh_obs,
                                                    fs=1./dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

**SPECTRAL COHERENCE**

```
In [9]: # Compute coherence ssh_map ssh_obs
        freq, coh_ssh_map_ssh_obs = signal.coherence(ssh_map,
```

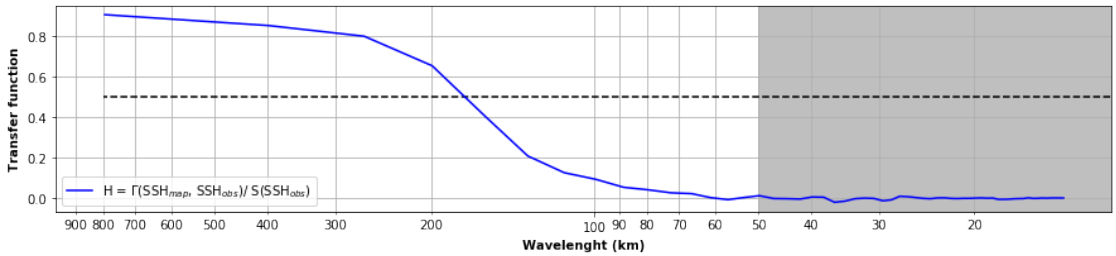
```
                                                    ssh_obs,
                                                    fs=1/dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)

        # Compute coherence ssh_map ssh_model
        _, coh_ssh_map_ssh_model = signal.coherence(ssh_map,
                                                    ssh_model,
                                                    fs=1/dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

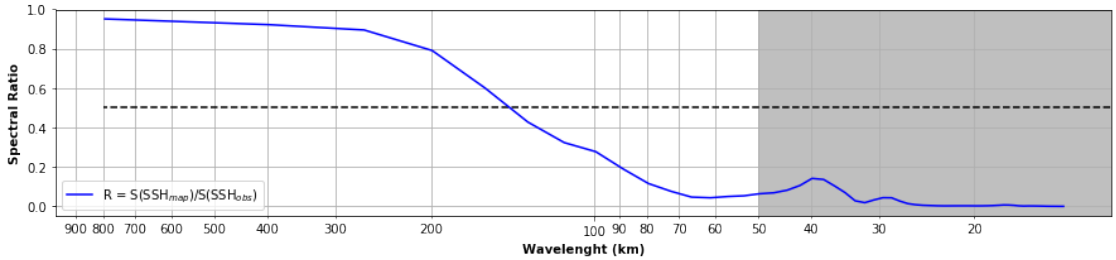
In [10]:
```
        # TRANSFER FUNCTION
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, cross_psd_ssh_map_ssh_obs/psd_ssh_obs,
                 c='b', label='H = $\Gamma$(SSH$_{map}$, SSH$_{obs}$)/ S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Transfer function', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # SPECTRAL RATIO
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, psd_ssh_map/psd_ssh_obs, c='b',
                 label='R = S(SSH$_{map}$)/S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Spectral Ratio', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # MAGNITUDE SQUARED COHERENCE
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, coh_ssh_map_ssh_obs, c='C8',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{obs}$)', lw=6)
        plt.plot(1./freq, coh_ssh_map_ssh_model, c='b',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{model}$)')
```

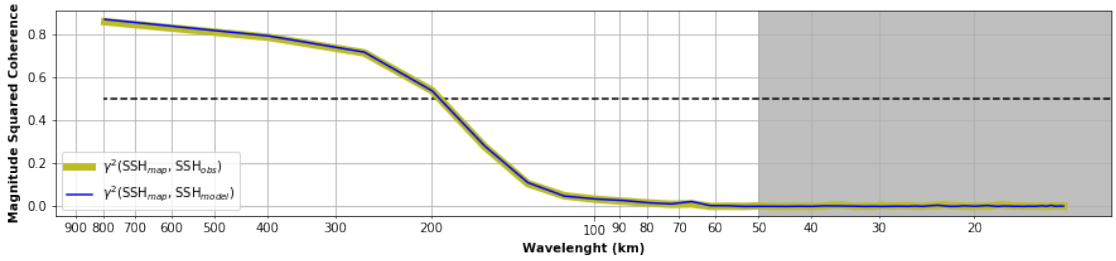
```
plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
plt.gca().invert_xaxis()
plt.grid(which='both')
plt.legend()
plt.xscale('log')
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
ax.axvspan(0, 50, color='grey', alpha=0.5)
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('Magnitude Squared Coherence', fontweight='bold')

# RATIO ERROR SPECTRUM / (SSH_OBS SPECTRUM)
fig, ax = plt.subplots(figsize=(15, 3))
plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
plt.plot(1./freq, (psd_mapping_error)/psd_ssh_model, c='r',
        label='S(mapping_error)/S(SSH$_{model}$)')
plt.plot(1./freq, (psd_ssh_obs_minus_map)/(psd_ssh_obs), c='b',
         label='S(SSH$_{obs}$ - SSH$_{map}$)/S((SSH$_{obs}$)')
plt.plot(1./freq, (psd_ssh_obs_minus_map - psd_intrumental_error)/
         (psd_ssh_obs - psd_intrumental_error), c='C8',
         label='(S(SSH$_{obs}$ - SSH$_{map}$) - S(intrumental_error))/(S(SSH$_{obs}$)

percent = 0.5

upper_bound = np.maximum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                         np.abs(psd_ssh_obs_minus_map -
                          (1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))

lower_bound = np.minimum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                        (psd_ssh_obs_minus_map -
                         np.abs(1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))

ax.fill_between(1./freq, lower_bound, upper_bound, color='C8', alpha=0.5)
plt.ylim(0,2)
plt.gca().invert_xaxis()
plt.grid(which='both')
plt.legend()
plt.xscale('log')
ax.axvspan(0, 50, color='grey', alpha=0.5)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.xlabel('Wavelenght (km)', fontweight='bold')
```

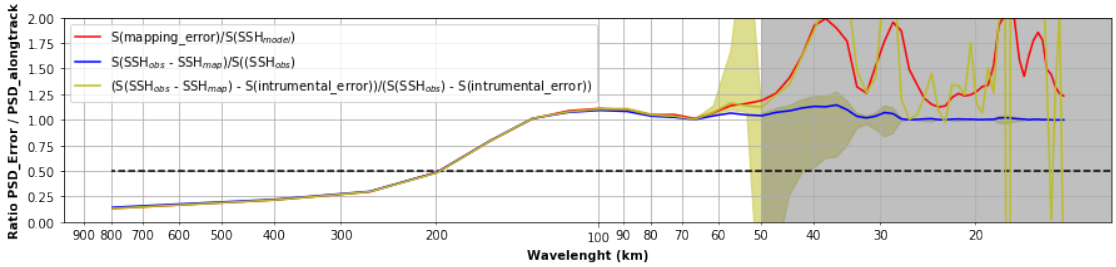
```
        plt.ylabel('Ratio PSD_Error / PSD_alongtrack', fontweight='bold')

        plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/numpy/core/numeric.py:501: ComplexWarning: Casting compl
  return array(a, dtype, copy=False, order=order)
```



**CONCLUSIONS**

We here used "realistic" dataset to investigate the response of various spectral metrics as a solution for estimating the resolution of the DUACS gridded products. The results for this case study #1 sligthly differ from case study #2 where the reference along-track dataset is used in the mapping process. The transfer function, the magnitude squared coherence and the ratio PSD error, PSD signal are affected by the independency/NON-indepedendy of the reference dataset. The spectal ratio PSD map / PSD alongtrack is weakly affected. Obviously, when the reference data is non-independent it "forces" the SSH map to be closer to the SSH obs, hence the curves of the tranfer function, the magnitude squared coherence and the ratio PSD Error / PS_alongtrack are shifted toward higher wavelenght.

Our main conclusions are:

1) The estimation of the spectral metrics are affected by considering or not the reference data as independent. Indepedent data lead to coarser resolution (spectral metrics are shifted towards higher wavenumber)

2) Magnitude squared coherence and Ratio PSD error/PSD along-track are in good agreements, meaning the phase consistency is the dominant factor controlling the "quality" of the DUACS maps

3) Spectral ratio is weakly affected by considering or not the reference dataset as independent

4) Considering noise or not in the alongtrack dataset as weak impact on the coherence (probably linked to the fact that mapping error >> intrumental error at wavelength that we are looking at.

5) The theoretical basis for the coherence-based measure of resolution at MSC=0.5 was probably mis-interpreted. It might be clearer to defined it as the coherent/incoherent threshold.

6) However, based on Tom's comments and also due to the fact that the coherence based measure is insensitive to the difference in the signals amplitudes, an other approach based on the spectral ratio of the error over signal is preferable for estimating the resolution of the DUACS maps. By defining the resolution as the 0.5 threshold of the ratio PSD(Error)/PSD(ref) (limit where there is two times more signal than error) we shows that for both case study, the estimation of this ratio as S(mapping_error)/S($SSH_{model}$) or S(($SSH_{map}$ - ($SSH_{obs}$)/S(($SSH_{obs}$) are similar for wavelenght > 50 km.

Tom's review elegantly shows the behaviour of the spectral metrics in various scenarios of noise level (no noise, high noise , low noise level). It is hence an assessment of the resolution in a general context. Our study with realistic data is more specific (applied in altimetry data processing context).

# Example_OSSE_obs_IN_mapping_NATL60

April 18, 2019

## 0.1 CASE STUDY 2: COMPARISON WITH NON-INDEPENDENT ALONG-TRACK

**INTRODUCTION**

We here propose to show the behaviour of the spectral analysis in an OSSE case study based on NATL60 simulation. The input comes from hourly SSH data from a 1 year long period of the run NATL60-CJM165 made at IGE (Grenoble). We used the SWOTsimulator to sample alongtrack data (nadir-like dataset) on this simulation. The constellation is similar to the 20030101-20031231 period: Jason 1, Envisat, Geosat2, Topex/Poseidon interleaved. SWOTsimulator gives us the true SSH ($SSH_{model}$) and a pseudo-obs SSH ($SSH_{obs}$ = $SSH_{model}$ + white noise). These pseudo-obs $SSH_{obs}$ are then used in the DUACS mapping.

**CASE STUDY #2 SUMMARY:** - the along-track of reference (i.e., for comparison) is **Jason1**. In this case study #2, this along-track **is included** in the mapping. - the $SSH_{map}$ produced are generated from **3 nadir altimeters** TPN, J1, EN with the DUACS system - we focus on the region in the north-atlantic bassin **(10řx10ř box centered near 330řE-44řN)**

Note that the ratio PSD(mapping_error)/PSD($SSH_{model}$) can gives access to the estimation of the map resolution. However, the DUACS maps and along-track products gives us access to only: PSD($SSH_{map}$), PSD($SSH_{obs}$) and PSD(intrumental_error). We will have to play with this three latter quantities to estimate the resolution. In the study cases 1 and 2, we shows that the PSD($SSH_{map}$), PSD($SSH_{obs}$) and PSD(intrumental_error) can estimate the resolution and that it is in good agreement with the ratio PSD(mapping_error)/PSD($SSH_{model}$).

PYTHON MODULE

```
In [1]: from netCDF4 import Dataset
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy import signal, interpolate
        from matplotlib.ticker import FormatStrFormatter
```

**READ THE SSH SEGMENTS**

*Note*: The SSH segments were generated using same algorithm as used for the study described in the manuscript. There are 800km long, referenced by their mean latitude and mean longitude and they overlap over 25%

**NOTATION**

- **$SSH_{model}$**: alongtrack SSH from model
- **$SSH_{obs}$**: alongtrack SSH from model including white noise
- **$SSH_{map}$**: SSH mapped with the DUACS algorithm using 3 altimeters data, and interpolated onto the alongtrack $SSH_{model}$ path

- **intrumental_error**: $SSH_{obs}$ minus $SSH_{model}$
- **map_error**: $SSH_{model}$ minus $SSH_{map}$

```
In [2]: nc = Dataset('./data/dataset_NON_independent_alongtrack_J1.nc','r')
        ssh_model = nc.variables['ssh_model'][:]
        ssh_obs = nc.variables['ssh_obs'][:]
        ssh_map = nc.variables['ssh_map'][:]
        dx = nc.variables['resolution'][:]
        nperseg = nc.dimensions['segment_size'].size
        nc.close()
```

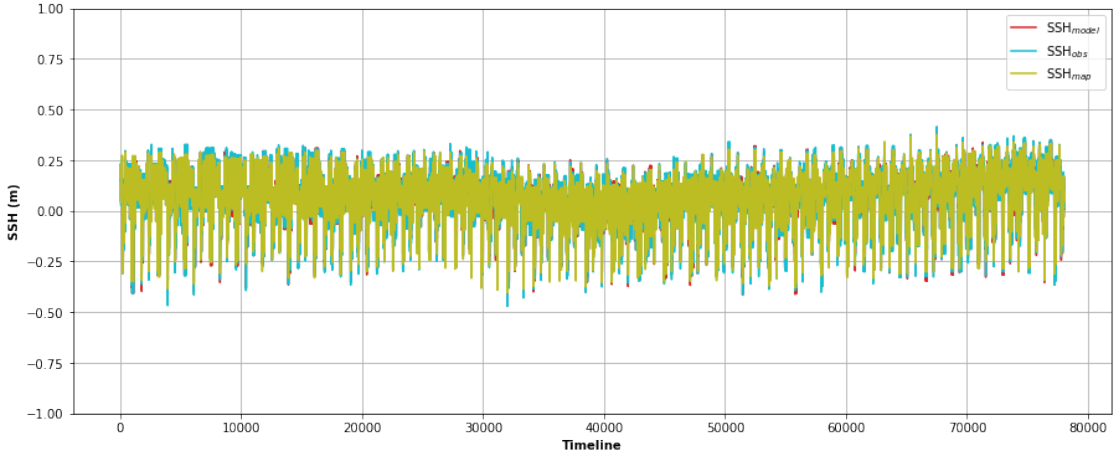
**COMPUTE INSTRUMENTAL ERROR & MAPPING ERROR**

```
In [3]: intrumental_error = ssh_obs - ssh_model
        mapping_error = ssh_model - ssh_map
```

**QUANTITY THAT CAN BE EVALUATED FROM ALTIMETRY DATA** : $SSH_{map}$ - $SSH_{obs}$

```
In [4]: # What can evaluated from altimetry product (we don't have access to ssh_model)
        psd_ssh_obs_minus_map = ssh_obs - ssh_map
```

```
In [5]: plt.figure(figsize=(15, 6))
        plt.xlabel("Timeline", fontweight='bold')
        plt.ylabel("SSH (m)", fontweight='bold')
        plt.plot(ssh_model, c='C3', label='SSH$_{model}$')
        plt.plot(ssh_obs, c='C9', label='SSH$_{obs}$')
        plt.plot(ssh_map, c='C8', label='SSH$_{map}$')
        plt.ylim(-1, 1)
        plt.legend(loc='best')
        plt.grid()
        plt.show()
```

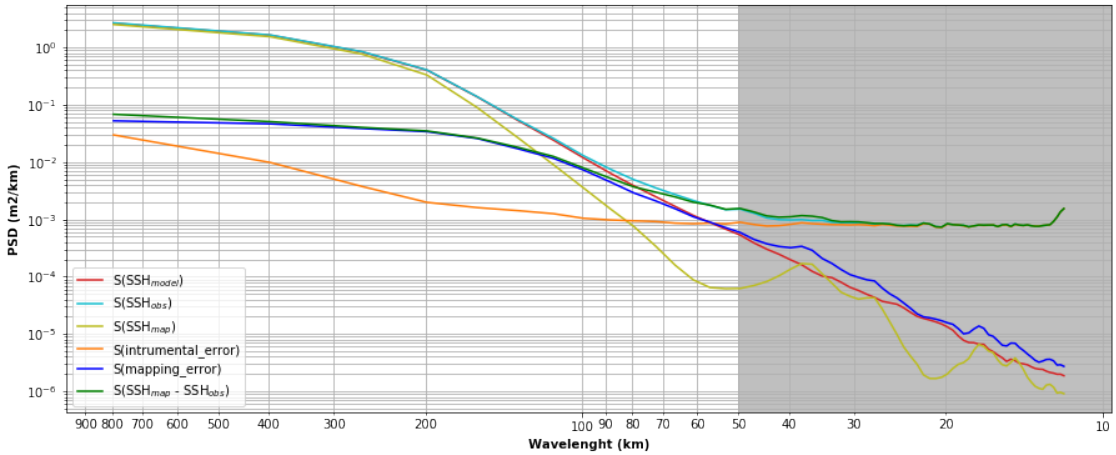



**SPECTRAL CONTENT OF EACH SIGNAL**

In [6]:
```
freq, psd_ssh_model = signal.welch(ssh_model,
                                   fs=1/dx,
                                   nperseg=nperseg,
                                   scaling='density',
                                   noverlap=0)

_, psd_ssh_obs = signal.welch(ssh_obs,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_ssh_map = signal.welch(ssh_map,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_intrumental_error = signal.welch(intrumental_error,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)

_, psd_mapping_error = signal.welch(mapping_error,
                                    fs=1/dx,
                                    nperseg=nperseg,
                                    scaling='density',
                                    noverlap=0)

_, psd_ssh_obs_minus_map = signal.welch(psd_ssh_obs_minus_map,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)
```

In [7]:
```
fig, ax = plt.subplots(figsize=(15, 6))
plt.plot(1/freq, psd_ssh_model, c='C3', label='S(SSH$_{model}$)')
plt.plot(1/freq, psd_ssh_obs, c='C9', label='S(SSH$_{obs}$)')
plt.plot(1/freq, psd_ssh_map, c='C8', label='S(SSH$_{map}$)')
plt.plot(1./freq, psd_intrumental_error, c='C1', label='S(intrumental_error)')
plt.plot(1./freq, psd_mapping_error, c='b', label='S(mapping_error)')
plt.plot(1./freq, psd_ssh_obs_minus_map, c='g', label='S(SSH$_{map}$ - SSH$_{obs}$)')
ax.axvspan(0, 50, color='grey', alpha=0.5)
plt.gca().invert_xaxis()
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Wavelenght (km)', fontweight='bold')
```

```
plt.ylabel('PSD (m2/km)', fontweight='bold')
plt.legend(loc='best')
plt.grid(which='both')
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.show()
```




The figure represents the power spectral densities from each type of signal. It illustrates a good agreement of the power spectral density S(SSH$_{obs}$), S(SSH$_{map}$) and S(SSH$_{model}$) for wavelength > 200-300 km. For wavelength < 200 km the power spectral density of the SSH$_{map}$ is less energetic than for the SSH$_{obs}$, linked to the filtering properties of te DUACS mapping system.

The signal-noise-ratio for the input along-track is between 50-60km (crossing of the red and orange lines), setting the along-track resolution limit as defined in Dufau et al (2016).

For wavelenght greater than this along-track resolution limit, the power spectral density of the mapping error S(mapping_error) is larger than the power spectral density of the instrumental error S(instrumental_error).

The power spectral density S(SSH$_{map}$) for wavelenght smaller than this along-track resolution limit (approx. grey area) is below the grid spacing of the DUACS maps. The power spectral density S(SSH$_{map}$) below this limit is hence resulting from the interpolation of the SSH$_{map}$ onto the reference alongtrack position

**CROSS SPECTRUM**

```
In [8]: freq, cross_psd_ssh_map_ssh_obs = signal.csd(ssh_map,
                                                    ssh_obs,
                                                    fs=1./dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

**SPECTRAL COHERENCE**

```
In [9]: # Compute coherence ssh_map ssh_obs
        freq, coh_ssh_map_ssh_obs = signal.coherence(ssh_map,
```

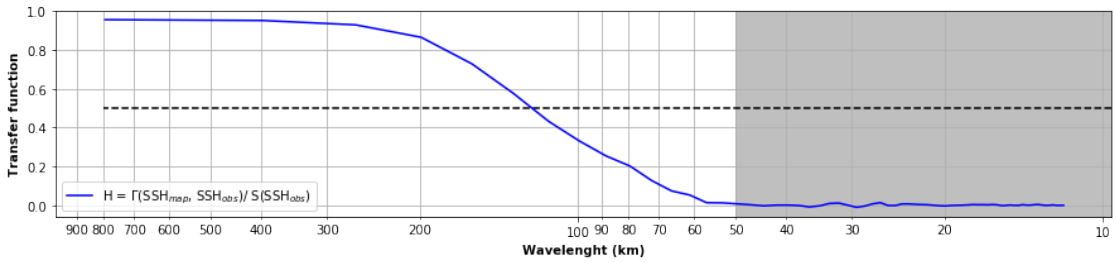
```
                                                    ssh_obs,
                                                    fs=1/dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)

        # Compute coherence ssh_map ssh_model
        _, coh_ssh_map_ssh_model = signal.coherence(ssh_map,
                                                    ssh_model,
                                                    fs=1/dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

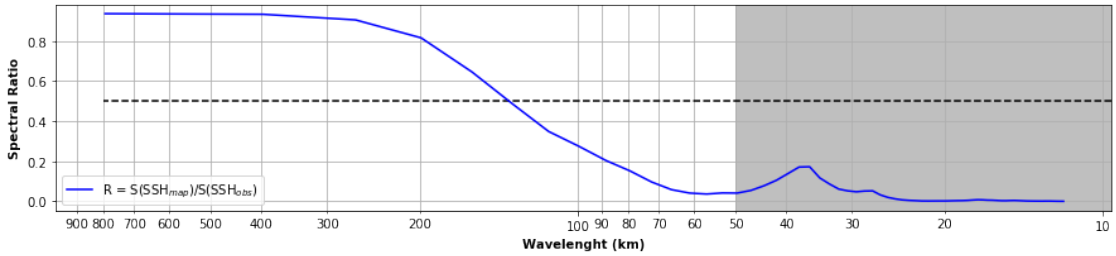
In [10]:
```
        # TRANSFER FUNCTION
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, cross_psd_ssh_map_ssh_obs/psd_ssh_obs,
                 c='b', label='H = $\Gamma$(SSH$_{map}$, SSH$_{obs}$)/ S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Transfer function', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # SPECTRAL RATIO
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, psd_ssh_map/psd_ssh_obs, c='b',
                 label='R = S(SSH$_{map}$)/S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Spectral Ratio', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # MAGNITUDE SQUARED COHERENCE
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, coh_ssh_map_ssh_obs, c='C8',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{obs}$)', lw=6)
        plt.plot(1./freq, coh_ssh_map_ssh_model, c='b',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{model}$)')
```

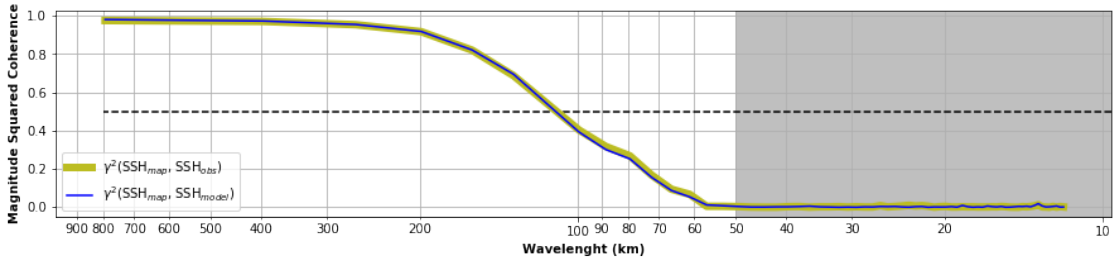
```
plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
plt.gca().invert_xaxis()
plt.grid(which='both')
plt.legend()
plt.xscale('log')
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
ax.axvspan(0, 50, color='grey', alpha=0.5)
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('Magnitude Squared Coherence', fontweight='bold')

# RATIO ERROR SPECTRUM / (SSH_OBS SPECTRUM)
fig, ax = plt.subplots(figsize=(15, 3))
plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
plt.plot(1./freq, (psd_mapping_error)/psd_ssh_model, c='r',
        label=' S(mapping_error)/S(SSH$_{model}$)')
plt.plot(1./freq, (psd_ssh_obs_minus_map)/(psd_ssh_obs), c='b',
         label='S((SSH$_{map}$ - (SSH$_{obs}$)/S((SSH$_{obs}$)')
#plt.plot(1./freq, (psd_ssh_obs_minus_map - psd_intrumental_error)/
#         (psd_ssh_obs - psd_intrumental_error), c='C8',
#         label='(S(SSH$_{map}$ - SSH$_{obs}$) - S(intrumental_error))/(S(SSH$_{obs}$

percent = 0.5

#percent = 0.5
upper_bound = np.maximum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                         (psd_ssh_obs_minus_map -
                          (1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))
lower_bound = np.minimum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                         (psd_ssh_obs_minus_map -
                          (1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))

ax.fill_between(1./freq, lower_bound, upper_bound, color='C8', alpha=0.5)
plt.ylim(0,2)
plt.gca().invert_xaxis()
plt.grid(which='both')
plt.legend()
plt.xscale('log')
ax.axvspan(0, 50, color='grey', alpha=0.5)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.xlabel('Wavelenght (km)', fontweight='bold')
```

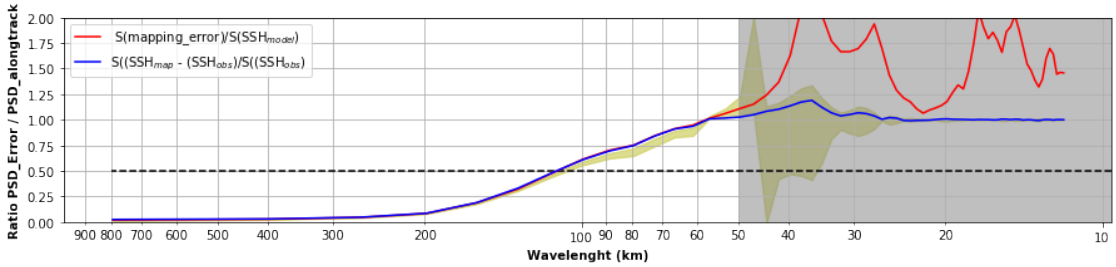
```
        plt.ylabel('Ratio PSD_Error / PSD_alongtrack', fontweight='bold')
        plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/numpy/core/numeric.py:501: ComplexWarning: Casting compl
  return array(a, dtype, copy=False, order=order)
```



This figure represents various metrics to estimate the resolution the map products: the transfer function, the spectral ratio between $SSH_{map}$ and $SSH_{obs}$, the Magnitude Squared Coherence and the ratio PSD error / PSD $SSH_{obs}$. The threshold $\gamma^2$ = 0.5 and Q = 0.5 gives relatively similar resolution limit for the maps.

The yellow envelop in lower panel gives the spread of the ratio (S($SSH_{map}$ - $SSH_{obs}$) - $\alpha$.S(intrumental_error))/(S($SSH_{obs}$) - $\alpha$.S(intrumental_error)) for $\alpha = \pm$ 50% increased/decreased instrumental error level. It shows the sensitivity of the ratio to the noise level in the reference along-track dataset. For wavelenght > 50 km, the ratio is weakly impacted by the amount of nosie level, and the three curve are rather similar.

# Example_OSSE_obs_NOT_mapping_NATL60_high_instrumental_nois

April 18, 2019

## 0.1 CASE STUDY 3: INDEPENDENT ALONG-TRACK WITH HIGHER INSTRUMENTAL NOISE

**INTRODUCTION**

We here propose to show the behaviour of the spectral analysis in an OSSE case study based on NATL60 simulation. The input comes from hourly SSH data from a 1 year long period of the run NATL60-CJM165 made at IGE (Grenoble). We used the SWOTsimulator to sample alongtrack data (nadir-like dataset) on this simulation. The constellation is similar to the 20030101-20031231 period: Jason 1, Envisat, Geosat2, Topex/Poseidon interleaved. SWOTsimulator gives us the true SSH ($SSH_{model}$) and a pseudo-obs SSH ($SSH_{obs}$ = $SSH_{model}$ + white noise). These pseudo-obs $SSH_{obs}$ are then used in the DUACS mapping.

**CASE STUDY #3 SUMMARY:** - the along-track of reference (i.e., for comparison) is **Geosat2**. In this case study #3, this along-track **is NOT included** in the mapping. **We virtually add extra noise to this reference along-track to simulate a case where the along-track resolution is above the filtering properties of the DUACS system** - the $SSH_{map}$ are generated from **3 nadir altimeters** TPN, J1, EN with the DUACS system - we focus on the region in the north-atlantic bassin (**10řx10ř box centered near 330řE-44řN**)

PYTHON MODULE

```
In [1]: from netCDF4 import Dataset
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy import signal, interpolate
        from matplotlib.ticker import FormatStrFormatter
```

**READ THE SSH SEGMENTS**

*Note*: The SSH segments were generated using same algorithm as for the study described in the manuscript. There are 800km long, referenced by their mean latitude and mean longitude and they overlap over 25%

**NOTATION**

- **$SSH_{model}$**: alongtrack SSH from model
- **$SSH_{obs}$**: alongtrack SSH from model including white noise
- **$SSH_{map}$**: SSH mapped with the DUACS algorithm using 3 altimeters data, and interpolated onto the alongtrack $SSH_{model}$ path
- **intrumental_error**: $SSH_{obs}$ minus $SSH_{model}$

- **mapping_error**: $SSH_{model}$ minus $SSH_{map}$

```
In [2]: nc = Dataset('./data/dataset_independent_alongtrack_G2.nc','r')
        ssh_model = nc.variables['ssh_model'][:]
        ssh_obs = nc.variables['ssh_obs'][:]
        ssh_map = nc.variables['ssh_map'][:]
        dx = nc.variables['resolution'][:]
        nperseg = nc.dimensions['segment_size'].size
        nc.close()
```

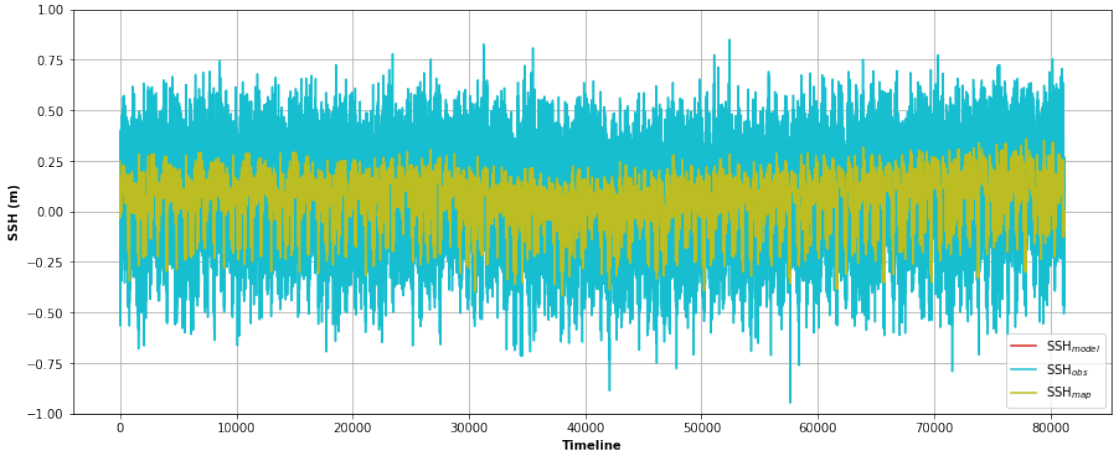
**COMPUTE INSTRUMENTAL ERROR & MAPPING ERROR**

```
In [3]: # Extra noise
        extra_noise = np.random.normal(0,0.16, ssh_model.size)
        intrumental_error = ssh_obs - ssh_model + extra_noise
        mapping_error = ssh_model - ssh_map
        ssh_obs = intrumental_error + ssh_model
```

**QUANTITY THAT CAN BE EVALUATED FROM ALTIMETRY DATA** : $SSH_{map}$ - $SSH_{obs}$

```
In [4]: # What can evaluated from altimetry product (we don't have access to ssh_model)
        ssh_obs_minus_map = ssh_obs - ssh_map
```

```
In [5]: plt.figure(figsize=(15, 6))
        plt.xlabel("Timeline", fontweight='bold')
        plt.ylabel("SSH (m)", fontweight='bold')
        plt.plot(ssh_model, c='C3', label='SSH$_{model}$')
        plt.plot(ssh_obs, c='C9', label='SSH$_{obs}$')
        plt.plot(ssh_map, c='C8', label='SSH$_{map}$')
        plt.ylim(-1, 1)
        plt.legend(loc='best')
        plt.grid()
```

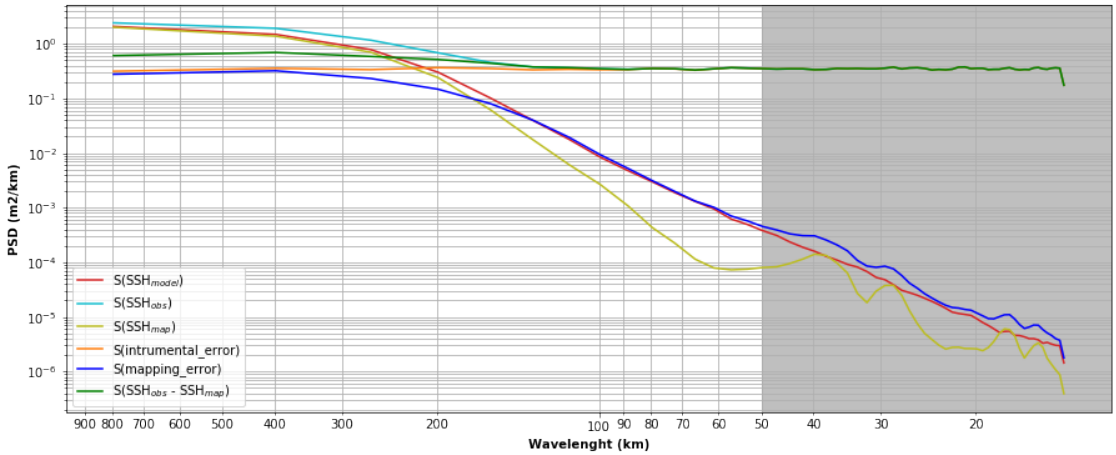



**SPECTRAL CONTENT OF EACH SIGNAL**

In [6]:
```
freq, psd_ssh_model = signal.welch(ssh_model,
                                   fs=1/dx,
                                   nperseg=nperseg,
                                   scaling='density',
                                   noverlap=0)

_, psd_ssh_obs = signal.welch(ssh_obs,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_ssh_map = signal.welch(ssh_map,
                              fs=1/dx,
                              nperseg=nperseg,
                              scaling='density',
                              noverlap=0)

_, psd_intrumental_error = signal.welch(intrumental_error,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)

_, psd_mapping_error = signal.welch(mapping_error,
                                    fs=1/dx,
                                    nperseg=nperseg,
                                    scaling='density',
                                    noverlap=0)

_, psd_ssh_obs_minus_map = signal.welch(ssh_obs_minus_map,
                                        fs=1/dx,
                                        nperseg=nperseg,
                                        scaling='density',
                                        noverlap=0)
```

In [7]:
```
fig, ax = plt.subplots(figsize=(15, 6))
plt.plot(1/freq, psd_ssh_model, c='C3', label='S(SSH$_{model}$)')
plt.plot(1/freq, psd_ssh_obs, c='C9', label='S(SSH$_{obs}$)')
plt.plot(1/freq, psd_ssh_map, c='C8', label='S(SSH$_{map}$)')
plt.plot(1./freq, psd_intrumental_error, c='C1', label='S(intrumental_error)')
plt.plot(1./freq, psd_mapping_error, c='b', label='S(mapping_error)')
plt.plot(1./freq, psd_ssh_obs_minus_map, c='g', label='S(SSH$_{obs}$ - SSH$_{map}$)')
plt.gca().invert_xaxis()
plt.yscale('log')
plt.xscale('log')
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('PSD (m2/km)', fontweight='bold')
```

```
plt.legend(loc='best')
plt.grid(which='both')
ax.axvspan(0, 50, color='grey', alpha=0.5)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.show()
```




The figure represents the power spectral densities from each type of signal. The agreement of the power spectral density S(SSH$_{obs}$), S(SSH$_{map}$) and S(SSH$_{model}$) is less obvious than in the previous case study.

The signal-noise-ratio for the input along-track is between 200-300km (crossing of the red and orange lines), setting the along-track resolution limit as defined in Dufau et al (2016).

For all wavelenght the power spectral density of the instrumental error S(instrumental_error) is larger than the power spectral density of the mapping error S(mapping_error) .

The wavelenght in the grey area are below the grid spacing of the DUACS maps. The power spectral density S(SSH$_{map}$) below this limit is hence resulting from the interpolation of the SSH$_{map}$ onto the alongtrack position

**CROSS SPECTRUM**

```
In [8]: freq, cross_psd_ssh_map_ssh_obs = signal.csd(ssh_map,
                                                    ssh_obs,
                                                    fs=1./dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

**SPECTRAL COHERENCE**

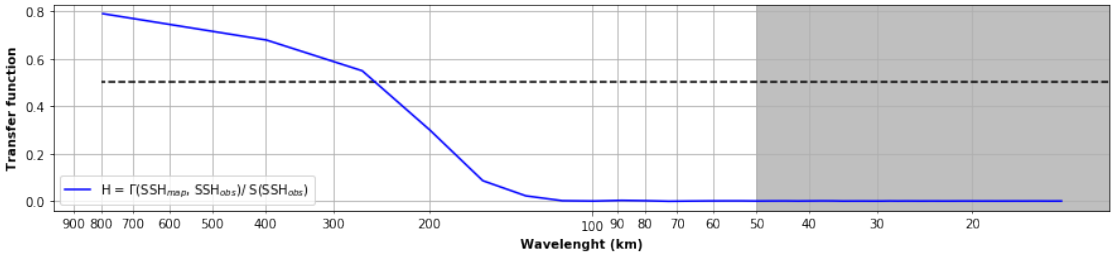
```
In [9]: # Compute coherence ssh_map ssh_obs
        freq, coh_ssh_map_ssh_obs = signal.coherence(ssh_map,
                                                     ssh_obs,
                                                     fs=1/dx,
                                                     nperseg=nperseg,
```

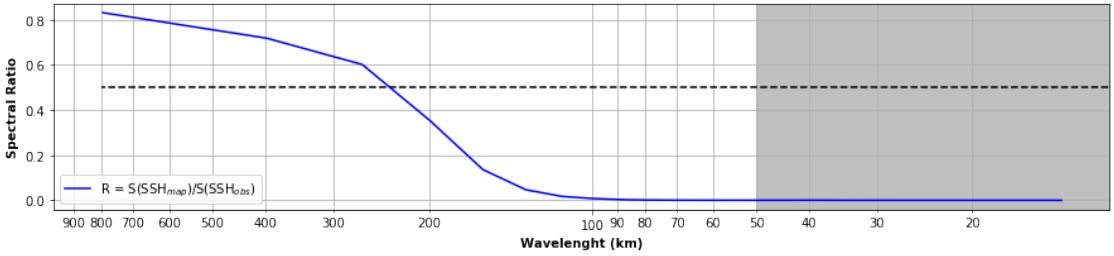
```
                                                    noverlap=0)

        # Compute coherence ssh_map ssh_model
        _, coh_ssh_map_ssh_model = signal.coherence(ssh_map,
                                                    ssh_model,
                                                    fs=1/dx,
                                                    nperseg=nperseg,
                                                    noverlap=0)
```

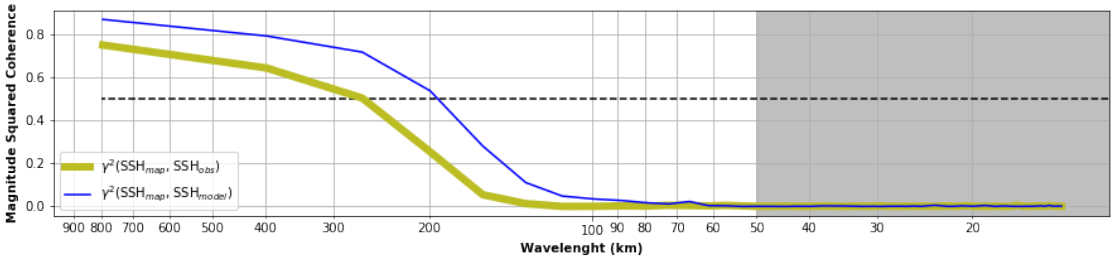
In [10]:
```
        # TRANSFER FUNCTION
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, cross_psd_ssh_map_ssh_obs/psd_ssh_obs,
                 c='b', label='H = $\Gamma$(SSH$_{map}$, SSH$_{obs}$)/ S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Transfer function', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # SPECTRAL RATIO
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, psd_ssh_map/psd_ssh_obs, c='b',
                 label='R = S(SSH$_{map}$)/S(SSH$_{obs}$)')
        plt.xscale('log')
        plt.gca().invert_xaxis()
        ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
        ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
        plt.xlabel('Wavelenght (km)', fontweight='bold')
        plt.ylabel('Spectral Ratio', fontweight='bold')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.legend()
        ax.axvspan(0, 50, color='grey', alpha=0.5)
        plt.grid(which='both')

        # MAGNITUDE SQUARED COHERENCE
        fig, ax = plt.subplots(figsize=(15, 3))
        plt.plot(1./freq, coh_ssh_map_ssh_obs, c='C8',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{obs}$)', lw=6)
        plt.plot(1./freq, coh_ssh_map_ssh_model, c='b',
                 label='$\gamma^2$(SSH$_{map}$, SSH$_{model}$)')
        plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
        plt.gca().invert_xaxis()
        plt.grid(which='both')
```

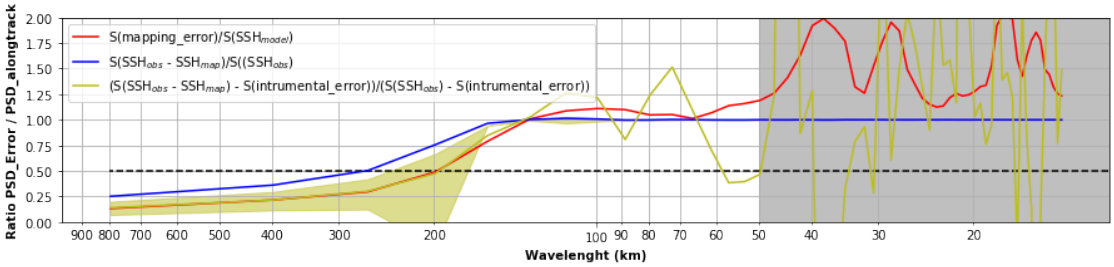
```
plt.legend()
plt.xscale('log')
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
ax.axvspan(0, 50, color='grey', alpha=0.5)
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('Magnitude Squared Coherence', fontweight='bold')

# RATIO ERROR SPECTRUM / (SSH_OBS SPECTRUM)
fig, ax = plt.subplots(figsize=(15, 3))
plt.hlines(0.5, xmin=0, xmax=800, linestyle='--')
plt.plot(1./freq, (psd_mapping_error)/psd_ssh_model, c='r',
        label='S(mapping_error)/S(SSH$_{model}$)')
plt.plot(1./freq, (psd_ssh_obs_minus_map)/(psd_ssh_obs), c='b',
         label='S(SSH$_{obs}$ - SSH$_{map}$)/S((SSH$_{obs}$)')
plt.plot(1./freq, (psd_ssh_obs_minus_map - psd_intrumental_error)/
         (psd_ssh_obs - psd_intrumental_error), c='C8',
         label='(S(SSH$_{obs}$ - SSH$_{map}$) - S(intrumental_error))/(S(SSH$_{obs}$)

percent = 0.5

upper_bound = np.maximum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                         np.abs(psd_ssh_obs_minus_map -
                          (1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))

lower_bound = np.minimum((psd_ssh_obs_minus_map -
                          (1 - percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 - percent)*psd_intrumental_error),
                        (psd_ssh_obs_minus_map -
                         np.abs(1 + percent)*psd_intrumental_error)/
                (psd_ssh_obs - (1 + percent)*psd_intrumental_error))

ax.fill_between(1./freq, lower_bound, upper_bound, color='C8', alpha=0.5)
plt.ylim(0,2)
plt.gca().invert_xaxis()
plt.grid(which='both')
plt.legend()
plt.xscale('log')
ax.axvspan(0, 50, color='grey', alpha=0.5)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.f'))
ax.xaxis.set_minor_formatter(FormatStrFormatter('%.f'))
plt.xlabel('Wavelenght (km)', fontweight='bold')
plt.ylabel('Ratio PSD_Error / PSD_alongtrack', fontweight='bold')

plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/numpy/core/numeric.py:501: ComplexWarning: Casting compl
  return array(a, dtype, copy=False, order=order)
```



**CONCLUSIONS**

Here the spectral ratio corrected from S(intrumental error isrelatively similar to PSD(mapping_error)/PSD($SSH_{model}$)) but it becomes very sensitive to the amount of instrumental noise (yellow spread delimiting a $\pm$ 50% increase/decrese noise level)