Assignment 1: Estimating sea-level trend using least-squares

G.H. Garrett, 4450973
Delft University of Technology, Delft, Zuid-Holland, 2628 HS

This assignment gave some interesting opportunities to touch up on my passion in data analysis, providing me with the volition to write a Medium post titled "A story of Satellites, Signals and Statistics" which can be found on my Medium page (`https://medium.com/@g.h.garrett`). Furthermore the software developed within this assignment can be found on my GitHub (`https://github.com/ggarrett13`). The software will be converted into a package to provide easy installation via *python-pip* within a week of submission. I fully encourage the use of the package under the MIT License. Finally, the assignment as a whole, considering the extra documentation and handling took approximately 15 hours to complete.

I. Estimating sea-level trend using least-squares

Part a): Estimate the sea-level trend and bias using least-squares and plot the data and your estimated trend. Write your own code for the least-squares estimation.

The task is broken up into steps that are required to adapt modular code for general application.

1. Create a function that solves for the unweighted vector of parameters ($\hat{\beta}$) estimation depending on the given arguments, taking into consideration the three forms of the LSQ algorithm, given the information matrix (H) and observations ($\bar{y}$).

$$\hat{\beta} = \begin{cases} (H^T H)^{-1} H^t \bar{y} & \forall m > n \\ H^{-1} \bar{y} & \forall m = n \\ H^t (H H^t)^{-1} \bar{y} & \forall m < n \end{cases}$$

   Where m is the quantity of observations, n is the quantity of predictors (or basis functions), $\hat{\beta}$ is the least squares solution of the vector of parameters ($\bar{\beta}$), H is the information matrix and $\bar{y}$ is the vector of observations.

$$\bar{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix} \quad (1) \qquad H = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{pmatrix} \quad (2) \qquad \bar{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix} \quad (3)$$

2. Create a modular object (relational data structure) that represents a linear model which takes a list of basis functions as its argument and can output H given the independent variable (x) which provides input to the basis functions.

3. Test the model analysis tool on the linear model containing trend and bias (equation 4) and plot the result.

$$y = \beta_0 + \beta_1 \cdot x \tag{4}$$

The code for the above can be found on GitHub (link found on the front page) for the most recent version. Additionally the version at the time of submission of the report can be found in the appendix. Plotting model 1, described by Equation 4 results in Figure 1, $\hat{\beta}$ is given by Equation 5.

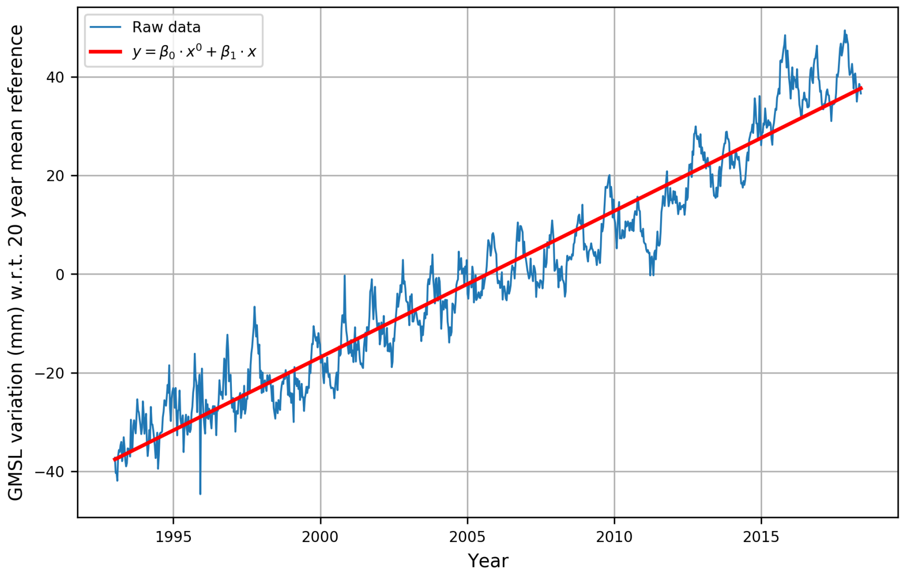

Fig. 1: Plot of model 1 ($y = \beta_0 + \beta_1 \cdot x$) superimposed on the raw observations.

$$\begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} -5.94 \cdot 10^3 \\ 2.96 \cdot 10^0 \end{pmatrix} \tag{5}$$

Part b): Do the same as for a) but simultaneously estimate a signal with a period of 1 year. Hint: use a combination of a sine and a cosine function. Explain in the report why it makes sense to estimate a yearly signal in sea level data, and explain if and why the estimated trend differs from a).

The code used for **Part a)** can be applied directly to the new equation describing the design linear model in Equation 6.

$$y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot \sin(2\pi f x) + \beta_3 \cdot \cos(2\pi f x), \tag{6}$$

where $f = 1/T$ and the period being used to model the periodic signal is 1 year, therefore $f = 1$.

Plotting the predicted solution ($\hat{y}$) provides Figure 2 and gives the least square solution for $\hat{\beta}$ given by Equation 7.

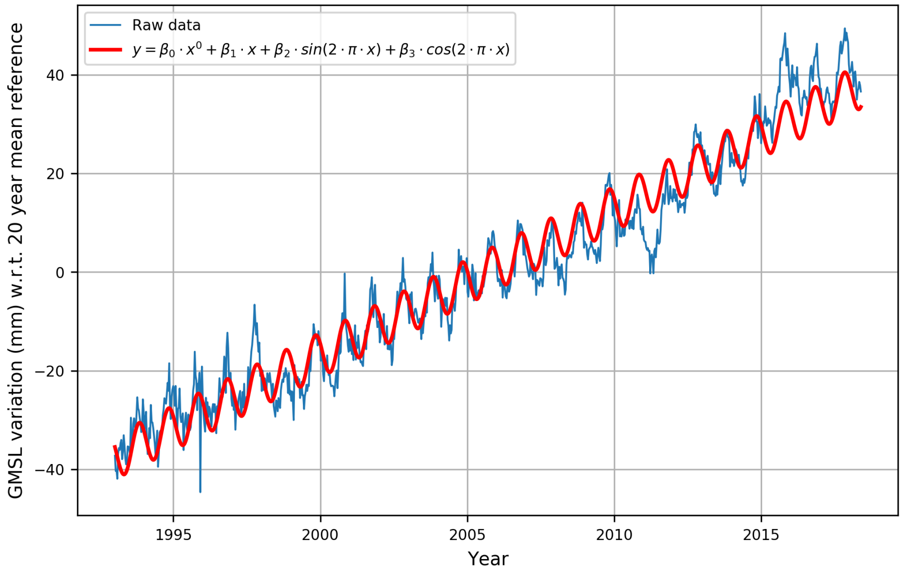

Fig. 2: Plot of model 2 ($y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot \sin(2\pi x) + \beta_3 \cdot \cos(2\pi x)$) superimposed on the raw observations.

$$\begin{pmatrix} \beta_0 & \beta_1 & \beta_2 & \beta_3 \end{pmatrix}^T = \begin{pmatrix} -5.94 \cdot 10^3 \\ 2.96 \cdot 10^0 \\ -3.85 \cdot 10^0 \\ 2.32 \cdot 10^0 \end{pmatrix} \tag{7}$$

Observations

1. The periodic signal (of T = 1 year) included in the linear model provides correct time interval predictions for the periodic signal visible within the observation data.

2. The periodic signal (of T = 1 year) does not provide any predictive power for the outlying peaks (local maximums) and troughs (local minimums) seen in the observation data.

Analysis & Evaluation

1. The yearly period can both be observed by the data and reasoned by environmental sciences as the yearly freezing and thawing of the polar ice caps due to the inclination of the Earth's axis and the effect this has due to solar radiation incident.

2. The linear model can benefit by the inclusion of additional basis functions following further analysis.

3. Within 3 significant figures, the values for β_0 and β_1 have remained constant during the recalculation of $\hat{\beta}$ on Equation 6 using the algorithm for unweighted least squares.

Part c): Find out if the residuals are normally distributed. Explain your test and the conclusions in the report.

Test Procedure

1. Firstly, calculate the vector of residuals for the linear model ($\bar{\epsilon}_l$) given by the difference between the actual observations ($\bar{y}$) and predicted values ($\hat{y}$). Use an algorithim to plot a histogram of residuals with 30 bins.

$$\bar{\epsilon}_l = \bar{y} - \hat{y} = \bar{y} - H\hat{\beta} \tag{8}$$

2. Secondly compare with the result of Step 1 to a normal distribution with the mean (μ) and standard deviation (σ) of $\bar{\epsilon}_l$.

$$p(\epsilon_l) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{\epsilon_l - \mu}{\sigma}\right)\right] \tag{9}$$

3. Finally calculate the skewness of the discrete random variable distribution (γ_1) and the kurtosis by old definition (γ_2) and finally kurtosis by the new definition (γ_2'). γ_1 is calculated by:

$$\gamma_1 = \frac{E[(X - \lambda_1)^3]}{\sigma^3}, \tag{10}$$

where:

$$E[X^k] = \lambda_k = \int_{-\infty}^{\infty} x^k p(x) dx = \sum_{i=1}^{m} p(x_i) x_i^k, \tag{11}$$

and:

$$\mu_k = E[(X-\lambda_1)^k] = \int_{-\infty}^{\infty} (x-\lambda_1)^k p(x)dx = \sum_{i=1}^{m} p(x_i)(x_i-\lambda_1)^k. \tag{12}$$

Finally the old definition of kurtosis is given by Equation 13 and the new by Equation 14.

$$\gamma_2 = \frac{\mu_4}{\sigma^4} \tag{13}$$

$$\gamma_2' = \gamma_2 - 3 \tag{14}$$

Results

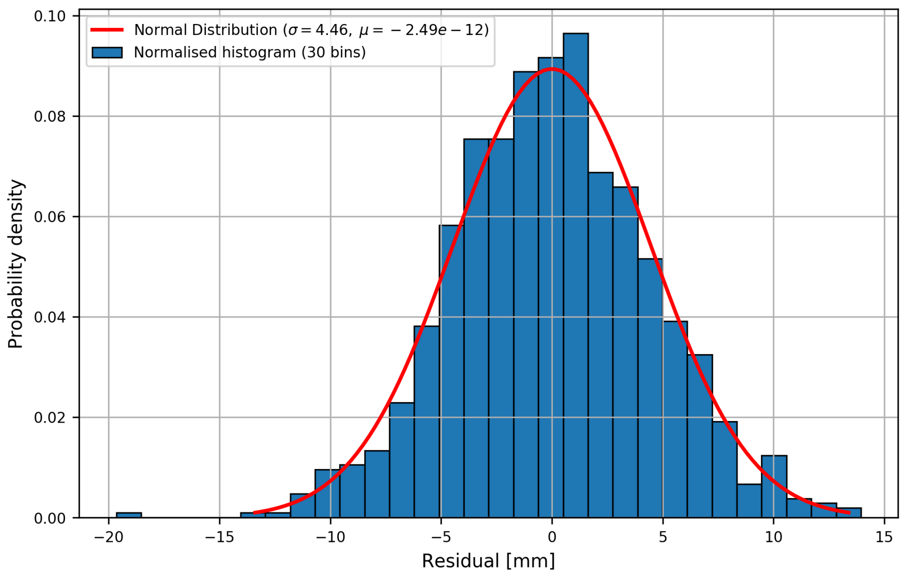

Fig. 3: Plot of model 2 ($y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot \sin(2\pi x) + \beta_3 \cdot \cos(2\pi x)$) normalised residuals with ideal normal distribution superimposed.

Table 1: Tabulated values for skewness (γ_1) and new kurtosis (γ_2') for model 1 and 2.

Parameter	**Model 1**	**Model 2**
γ_1	0.208	0.00183
γ_2'	-0.384	-0.0803

Conclusions

1. The normalised histogram plot of $\bar{\epsilon}_l$ follows the shape of the ideal normal distribution calculated closely.

2. The calculated μ of the residuals tends towards zero which indicates that error is largely from random error and systematic error is close to negligible in the model.

3. There exists some outliers in the negative region of the residuals. This introduces some skewness in the distribution.

4. The inclusion of the yearly periodic signal reduced the skewness (γ_1) by two orders of magnitude and the new kurtosis (γ_2') by one. Therefore the residuals better represent a gaussian distribution as a result of the inclusion of the periodic signal.

Part d): Do the same as b) and c) but now include an acceleration term. Explain if and why the estimated trend and the residuals differ from your answers to b) and c)

The equation used to model the data is as follows below with the inclusion of the acceleration term x^2. The results follow directly from the previous procedures.

$$y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot \sin(2\pi x) + \beta_4 \cdot \cos(2\pi x) \tag{15}$$

Results

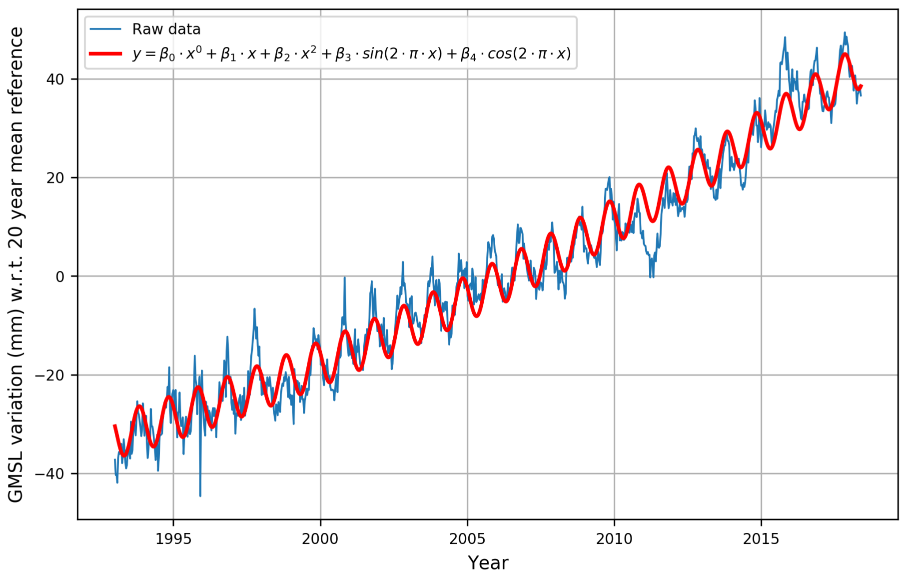

Fig. 4: Plot of model 3 ($y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot \sin(2\pi x) + \beta_4 \cdot \cos(2\pi x)$) superimposed on the raw observations.

$$\begin{pmatrix} \beta_0 & \beta_1 & \beta_2 & \beta_3 & \beta_4 \end{pmatrix}^T = \begin{pmatrix} 1.84 \cdot 10^5 \\ -1.87 \cdot 10^2 \\ 4.73 \cdot 10^{-2} \\ -3.96 \cdot 10^0 \\ 2.28 \cdot 10^0 \end{pmatrix} \tag{16}$$

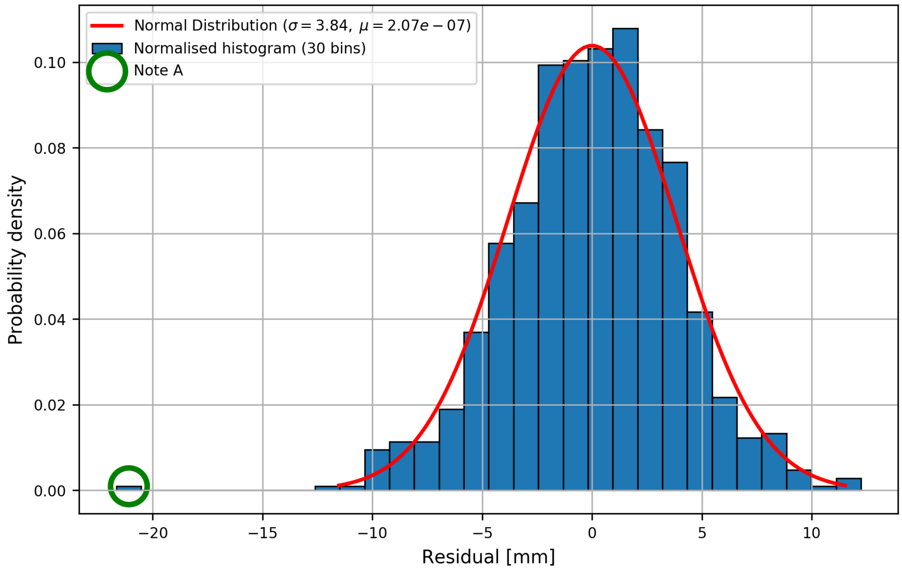

Fig. 5: Plot of model 3 ($y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot \sin(2\pi x) + \beta_4 \cdot \cos(2\pi x)$) normalised residuals with ideal normal distribution superimposed.

Parameter	**Model 1**	**Model 2**	**Model 3**
γ_1	0.208	0.00183	-0.221
γ_2'	-0.384	-0.0803	0.506

Conclusions

1. According to the measurement of γ_1 and γ_2', the probability density function of model 3's residuals have worsened in representation of a Gaussian distribution. This could be as a result of the increased 3rd and 4th order moment about the mean caused by *Note A* in Figure 5 which has increased in negativity from model 2.

2. The standard deviation (σ) has decreased significantly from model 2 to model 3. This means

that there is less variation in the residuals about the mean which is approximately equal to zero.

3. Overall it can be said that the model is a better predictor of the observable true value from the observed values, or in other words the random error has been reduced in the model.

Part e): Explain conceptually what the differences are between residuals and true errors.

Residuals Residuals are the difference between the observed values of a statistical sample and the predicted (or estimated) values according to a predictive model derived from the statistical sample set (e.g. sample mean).

True errors True errors are the difference between the observed values of a statistical sample and the (unobservable) true values (e.g. population mean).

Part f): Estimate the standard deviation of the trend estimate. You can now assume that standard deviation of the measurements (measurement error) can be computed from the residuals as follows:

$$\sigma_l = \frac{\epsilon\epsilon^T}{m - n} \tag{17}$$

For the symbols see section 8.5 of the lecture notes. In the report, give the equation and describe what each symbol represents and comment on the value of the standard deviation.

Using Equation 8 to calculate $\bar{\epsilon}_l$, and Equation 17, the standard deviation of the measurements can be obtained (σ_l), where m is the quantity of observed values and n is the quantity of predictors (or basis functions). Equation 18 defines the observation covariance matrix. The leading diagonal contains the variance of each observation, and the off-diagonals contain the covariance between observations.

$$P_{yy} = \begin{pmatrix} \sigma_{l,1}^2 & \sigma_{l,1}\sigma_{l,2} & \dots & \sigma_{l,1}\sigma_{l,2} \\ \sigma_{l,2}\sigma_{l,1} & \sigma_{l,2}^2 & \dots & \sigma_{l,2}\sigma_{l,m} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{l,m}\sigma_{l,1} & \sigma_{l,m}\sigma_{l,2} & \dots & \sigma_{l,m}^2 \end{pmatrix} \tag{18}$$

Typically for **independent** observations which do not influence other observations, the following equation defines the observation covariance matrix (P_{yy}):

$$P_{yy} = \sigma_l^2 I \tag{19}$$

Following the calculation of P_{yy}, the parameter covariance matrix (P_{xx}) can be obtained through Equation I.

$$P_{xx} = (H^t P_{yy}^{-1} H)^{-1} \tag{20}$$

The previous calculation procedure is carried out for model 3 and is seen following with all calculations made to 3 significant figures.

$$P_{yy} = \sigma_l^2 I = (14.8)^2 \cdot I_m$$

$$P_{xx} = (H^t P_{yy}^{-1} H)^{-1} = \begin{pmatrix} 1.11 \cdot 10^8 & -1.11 \cdot 10^5 & 2.76 \cdot 10^1 & -6.66 \cdot 10^1 & -2.26 \cdot 10^1 \\ -1.11 \cdot 10^5 & 1.10 \cdot 10^2 & -2.75 \cdot 10^{-2} & 6.64 \cdot 10^{-2} & 2.25 \cdot 10^{-2} \\ 2.76 \cdot 10^1 & -2.75 \cdot 10^{-2} & 6.85 \cdot 10^{-6} & -1.65 \cdot 10^{-5} & -5.63 \cdot 10^{-6} \\ -6.66 \cdot 10^1 & 6.64 \cdot 10^{-2} & -1.65 \cdot 10^{-5} & 3.17 \cdot 10^{-2} & -7.63 \cdot 10^{-5} \\ -2.26 \cdot 10^1 & 2.25 \cdot 10^{-2} & -5.63 \cdot 10^{-6} & -7.63 \cdot 10^{-5} & 3.18 \cdot 10^{-2} \end{pmatrix}$$

Part g): Explain what the off-diagonal terms of the parameter co-variance matrix represent.

The off-diagonal terms represent the covariance between differing parameters, which is a measure of the joint variability of two parameters. From this value we can obtain the correlation between different predictors using the following equation:

$$\rho = \frac{\sigma_{\beta_i \beta_j}}{\sigma_{\beta_i} \sigma_{\beta_j}} \tag{21}$$

Part h): Explain if you expect that the sea level data you downloaded contain other physical signals except a bias, trend, acceleration and a yearly signal. Describe how you can test this with the data you have.

In order to obtain better representations of the signals that are consisted within the raw data, a Fourier analysis can be carried out. The signals found can be then included in the model analysis.

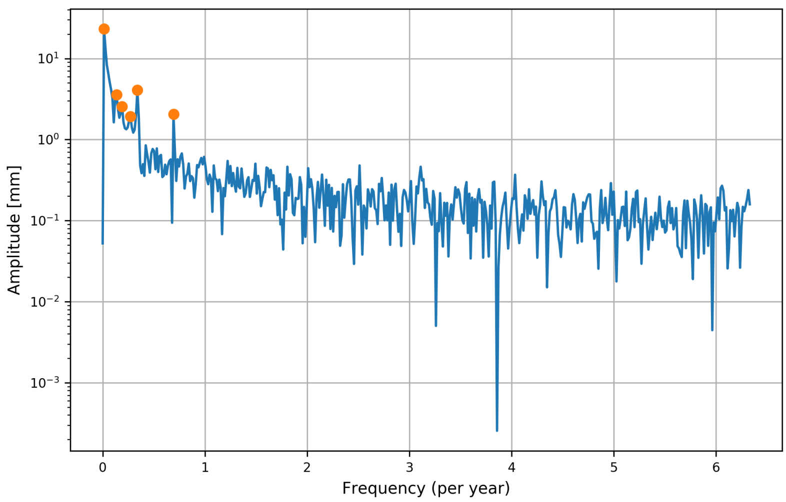

Fig. 6: Fourier analysis carried out on the data set with the 6 signals greater than 1 mm in magnitude marked.

The collection of 6 points above 1 mm in amplitude are shown in the vector seen in Equation 23. These x values are used as new frequencies to be placed into the signal:

$$y = \beta_0 \sin(2\pi f_1 x) + \beta_1 \cos(2\pi f_1 x)...\beta_{m-1} \sin(2\pi f_{m/2} x) + \beta_m \cos(2\pi f_{m/2} x) \quad (22)$$

This is then added onto model 3 in order to create a new model 4. Consisting of various frequency signals.

$$(\bar{x}, y) = \begin{pmatrix} (0.01359, 23.10) \\ (0.1359, 3.570) \\ (0.1902, 2.565) \\ (0.2718, 1.920) \\ (0.3397, 4.075) \\ (0.6930, 2.056) \end{pmatrix} \quad (23)$$

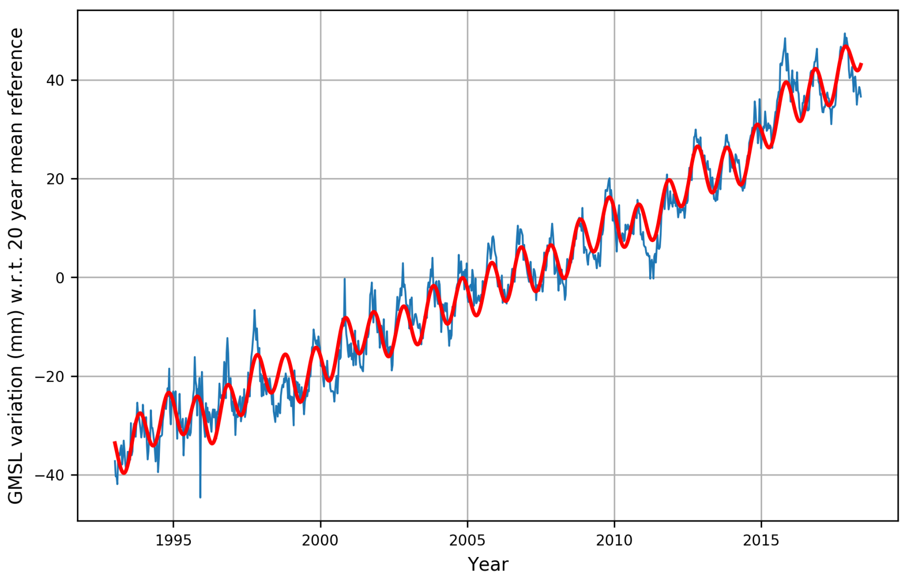

Fig. 7: Plot of model 4 superimposed on the raw observations.

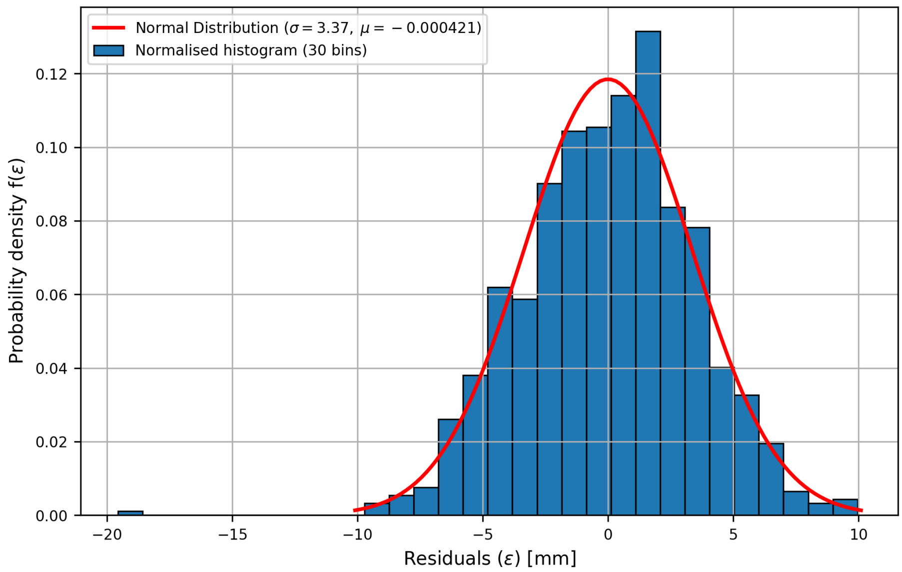

Fig. 8: Plot of model 4 normalised residuals with ideal normal distribution superimposed.

Conclusion

1. The trend model seems to have significantly improved seen by Figure 7. Some anomalous troughs and peaks have been estimated correctly.

2. The standard deviation of the distributed residuals has decreased from 3.84 to 3.37 showing better prediction of the unobservable true values with reduction of random error.

3. Considering the quantity of extra predictive terms (or basis functions) the design model may not have significantly improved predictive capabilities when considering the amount of extra computation required has been more than doubled (due to more than double the basis functions).

Part i): Mention four different error sources that affect the global mean sea level estimate derived from a satellite altimeter, assuming that we are interested in the climate-related sea level rise.

Error sources

1. Potential systematic error in the satellite altimeter measurement due to imperfect calibration.
2. Further altimeter error due to ageing or error in the ground processing of data.
3. Aleatory error due to orbit solutions and gravitational models that are constantly improving.
4. The required correction for the wet troposphere which can be affected by long-term instrument drifts. These drifts may be due to internal temperature changes induced by yaw manoeuvres.

II. Github code

A. functions.py

```python
import numpy as np


def parameter_covariance(H, Pyy=None):
    if Pyy is not None:
        return np.linalg.inv(np.matmul(H.T, np.matmul(np.linalg.inv(Pyy), H)))
    else:
        return np.linalg.inv(np.matmul(H.T, H))


def unweighted_least_squares(H, y):
    """
    :param H: Information matrix (np.ndarray)
    :param y: Vector of observations (np.ndarray)
    :return: Vector of parameters (np.ndarray)
    """
    # m > n
    if H.shape[0] > H.shape[1]:
        return np.matmul(np.linalg.inv(np.matmul(H.T, H)), np.matmul(H.T, y))
    # m = n
    elif H.shape[0] == H.shape[1]:
        return np.matmul(np.linalg.inv(H), y)
    # m < n
    elif H.shape[0] < H.shape[1]:
        return np.matmul(H.T, np.matmul(np.linalg.inv(np.matmul(H, H.T)), y))


def weighted_least_squares(H, y, Py):
    """
    TODO: (**) Complete other forms with covariance !=(m>n).
    :param H:
    :param y:
    :param Py:
    :return:
    """
    # m > n
    if H.shape[0] > H.shape[1]:
        return np.matmul(np.linalg.inv(np.matmul(np.matmul(H.T, np.linalg.inv(Py)), H)),
                         np.matmul(np.matmul(H.T, np.linalg.inv(Py)), y))
    else:
        raise NotImplementedError("TODO: Complete other forms with covariance !=(m>n)")


def kth_order_moment_about_zero(x, k, bins=None):
    """
    TODO: Complete kth_order_moment_about_zero doc-strings.
    :param x:
    :param k:
    :param bins:
    :return:
    """
    if bins:
        p_i, edges = np.histogram(x, density=True, bins=30)
        x_i = [np.mean([edges[i], edges[i + 1]]) for i in range(len(edges) - 1)]
        return np.sum(np.multiply(np.power(x_i, k), p_i))
```

```python
    else:
        return np.sum(np.multiply(np.power(x, k), x / np.sum(x)))


def kth_order_moment_about_mean(x, k, bins=None):
    """
    TODO: Complete kth_order_moment_about_mean doc-strings.
    :param x:
    :param k:
    :param bins:
    :return:
    """
    if bins:
        p_i, edges = np.histogram(x, density=True, bins=30)
        x_i = [np.mean([edges[i], edges[i + 1]]) for i in range(len(edges) - 1)]
        return np.sum(np.multiply(np.power(x_i - np.mean(x), k), p_i))
    else:
        return np.sum(np.multiply(np.power(x - np.mean(x), k), x / np.sum(x)))
```

B. model_design.py

```python
import numpy as np
import sympy as sp
from sympy.utilities.lambdify import lambdify
import matplotlib.pyplot as plt


MOD = 0
SMALL_SIZE = 8 + MOD
MEDIUM_SIZE = 10 + MOD
BIGGER_SIZE = 12 + MOD

plt.rc('font', size=SMALL_SIZE)  # controls default text sizes
plt.rc('axes', titlesize=SMALL_SIZE)  # fontsize of the axes title
plt.rc('axes', labelsize=MEDIUM_SIZE)  # fontsize of the x and y labels
plt.rc('xtick', labelsize=SMALL_SIZE)  # fontsize of the tick labels
plt.rc('ytick', labelsize=SMALL_SIZE)  # fontsize of the tick labels
plt.rc('legend', fontsize=SMALL_SIZE)  # legend fontsize
plt.rc('figure', titlesize=BIGGER_SIZE)  # fontsize of the figure title


class NonLinearDesignModel(object):
    def __init__(self, basis_functions):
        """
        TODO: (*) Complete non-linear design model.
        :param basis_functions: (list)
        """
        self._basis_functions = basis_functions
        raise NotImplementedError("TODO: Complete non-linear design model.")


class LinearDesignModel(object):
    def __init__(self, basis_functions):
        """
        :param basis_functions: (list)
        """
        self._basis_functions = basis_functions

    @property
    def basis_functions(self):
        return self._basis_functions

```

```
    def __add__(self, other):
        return LinearDesignModel(self.basis_functions + other.basis_functions)

    def __radd__(self, other):
        return LinearDesignModel(other.basis_functions + self.basis_functions)

    def __repr__(self):
        vec_param = '  '.join(['B' + str(i) for i in range(len(self._basis_functions))])
        vec_basis = '  '.join(self._basis_functions)
        return 'DesignModel(y = [{}]^T [{}])'.format(vec_param, vec_basis)

    def __str__(self):
        return 'y = ' + ' + '.join(['B_{}'.format(i) + '*' + j for
                                     i, j in enumerate(self._basis_functions)]) + ''

    def __latex__(self):
        return '$' + self.__str__().replace('*', '\cdot{}').replace('pi', '\pi').replace('B', '\\beta') + '$'

    def information_matrix(self, x):
        information_matrix = np.array([])
        for basis in self.basis_functions:
            if len(information_matrix) == 0:
                if (basis == '1') or (basis == 'x^0'):
                    information_matrix = np.full((len(x), 1), 1)
                else:
                    information_matrix = lambdify(sp.Symbol('x'), basis, 'numpy')(x)
            else:
                information_matrix = np.c_[information_matrix, lambdify(sp.Symbol('x'), basis, 'numpy')(x)]
        return information_matrix
```

C. data_analysis.py

```
from functions import kth_order_moment_about_mean
from functions import kth_order_moment_about_zero
from functions import unweighted_least_squares
from functions import weighted_least_squares
from model_design import LinearDesignModel
from model_design import NonLinearDesignModel
import matplotlib.pyplot as plt
import numpy as np
import statistics
from matplotlib.pyplot import figure
import matplotlib.mlab as mlab


class DataAnalysis2D(object):

    @staticmethod
    def skewness(var, bins=None):
        """
        TODO: Complete skewness doc-strings.
        :param var:
        :param bins:
        :return:
        """
        return kth_order_moment_about_mean(var, 3, bins=bins) / np.power(np.std(var), 3)

```

```
    @staticmethod
    def kurtosis_old(var, bins=None):
        """
        TODO: Complete kurtosis_old doc-strings.
        :param var:
        :param bins:
        :return:
        """
        return kth_order_moment_about_mean(var, 4, bins=bins) / np.power(np.std(var), 4)

    @staticmethod
    def kurtosis_new(var, bins=None):
        """
        TODO: Complete kurtosis_new doc-strings.
        :param var:
        :param bins:
        :return:
        """
        return kth_order_moment_about_mean(var, 4, bins=bins) / np.power(np.std(var), 4) - 3

    def __init__(self, design_model: LinearDesignModel, x: np.ndarray, y: np.ndarray, Py: np.ndarray = None):
        """
        :param design_model: Design model to be analysed (LinearDesignModel/NonLinearDesignModel)
        :param x: Independent variable associated with observations (np.ndarray)
        :param y: Vector of observations (np.ndarray)
        """
        self._x = x
        self._y = y
        self._Py = Py
        self._design_model = design_model

    @property
    def Py(self):
        try:
            return self._Py
        except AttributeError:
            raise AttributeError("Matrix Py has not been provided, please set using Py setter.")

    @Py.setter
    def Py(self, arg):
        self._Py = arg

    def unweighted_least_squares(self):
        """
        :return: Unweighted least squares solution of the vector of parameters (np.ndarray)
        """
        return unweighted_least_squares(self._design_model.information_matrix(self._x), self._y)

    def unweighted_prediction(self, x):
        """
        :param x: Independent variable associated with observations (np.ndarray)
        :return: Unweighted predictions using the design model (np.ndarray)
        """
        return np.matmul(self._design_model.information_matrix(x), self.
```

```
            unweighted_least_squares())

    def unweighted_residuals(self):
        """
        :return: Unweighted residuals between observations and unweighted
            prediction (np.ndarray)
        """
        return self._y - self.unweighted_prediction(self._x)

    def weighted_least_squares(self):
        """
        :return: Unweighted least squares solution of the vector of
            parameters (np.ndarray)
        """
        return weighted_least_squares(self._design_model.
            information_matrix(self._x), self._y, Py=self.Py)

    def weighted_prediction(self, x):
        """
        :param x: Independent variable associated with observations (np.
            ndarray)
        :return: Weighted predictions using the design model (np.ndarray)
        """
        return np.matmul(self.weighted_least_squares(), self._design_model
            .information_matrix(x))

    def weighted_residuals(self):
        """
        :return: Weighted residuals between observations and weighted
            prediction (np.ndarray)
        """
        return self._y - self.weighted_prediction(self._x)

    def plot_model(self, plot_type='matplotlib', show_save=('show'), title
        =None, ylabel=None, xlabel=None, name=None,
                   type='unweighted', legend=True):
        """
        TODO: (*) Finish doc-strings for plot_model.
        :param plot_type:
        :param show_save:
        :param title:
        :param ylabel:
        :param xlabel:
        :param name:
        :param type:
        :return:
        """
        if plot_type is 'matplotlib':
            figure(num=None, figsize=(8, 5), dpi=300, facecolor='w',
                edgecolor='k')
            plt.plot(self._x, self._y, label='Raw data', linewidth=1)
            if title:
                plt.title(title)
            plt.grid()
            smoothed_x = np.linspace(self._x[0], self._x[-1], 1000)
            if type is 'weighted':
                predicted_y = np.matmul(self._design_model.
                    information_matrix(smoothed_x),
                                        self.weighted_least_squares())
            elif type is 'unweighted':
                predicted_y = np.matmul(self._design_model.
                    information_matrix(smoothed_x),
                                        self.unweighted_least_squares())
            else:
```

```
                raise SystemError("{} type not recognised, please use <weighted> or <unweighted>.".format(type))
            plt.plot(smoothed_x, predicted_y,
                     label=self._design_model.__latex__() + ' ' + str(type),
                     linestyle='-',
                     linewidth=2,
                     color='red')
            if legend:
                plt.legend()
            plt.ylabel(ylabel)
            plt.xlabel(xlabel)
            if 'show' in show_save:
                plt.show()
            if 'save' in show_save:
                plt.savefig('plots/' + str(name) + '.png', bbox_inches='tight')

    def plot_residual_hist(self, plot_type='matplotlib', show_save=('show'), title=None, ylabel=None, xlabel=None,
                           name=None, bins=30, type='unweighted'):
        '''
        TODO: (*) Finish doc-strings for plot_residual_hist.
        :param plot_type:
        :param show_save:
        :param title:
        :param ylabel:
        :param xlabel:
        :param name:
        :param bins:
        :return:
        '''
        if type is 'unweighted':
            residuals = self.unweighted_residuals()
        elif type is 'weighted':
            residuals = self.weighted_residuals()
        else:
            raise SystemError("{} type argument not recognised. Please use weighted or unweighted".format(type))
        mu = statistics.mean(residuals)
        sigma = statistics.stdev(residuals)
        x = np.linspace(mu - 3 * sigma, mu + 3 * sigma, 100)
        if plot_type is 'matplotlib':
            figure(num=None, figsize=(8, 5), dpi=300, facecolor='w', edgecolor='k')
            if title:
                plt.title(title)
            plt.grid()
            plt.hist(residuals,
                     bins=bins,
                     density=True,
                     label='Normalised histogram ({} bins)'.format(bins),
                     edgecolor='black',
                     linewidth=0.8)
            plt.plot(x, mlab.normpdf(x, mu, sigma),
                     label='Normal Distribution ($\sigma={0:.3g}'.format(sigma) + ',\;\mu={0:.3g}$)'.format(mu),
                     color='red', linewidth='2')

            # DEVELOPMENT FOR ANOMALIES
            # p, ed = np.histogram(residuals, bins=30, density=True)
            # mid = [np.mean([ed[i], ed[i + 1]]) for i in range(len(ed) - 1)]
            # plt.scatter(mid[0], p[0], s=400, facecolors='none',
```

```
                    edgecolors='g', linewidths=3, label='Note A')

            plt.legend()
            plt.ylabel(ylabel)
            plt.xlabel(xlabel)
            if 'show' in show_save:
                plt.show()
            if 'save' in show_save:
                plt.savefig('plots/' + str(name) + '.png', bbox_inches='tight')
```

D. gmst_handler.py

```
import numpy as np
import pandas as pd

GMSL_HDR = 50
GSML_COLUMNS = ['altt', 'mfc', 'year', 'n_obs', 'n_wobs', 'gmsl', 'gmsl_std', 'smoothed', 'gmsl_gia', 'gmsl_gia_std',
                'gmsl_gia_smthed_20', 'gmsl_gia_smthed_sig']
GSML_FILENAME = 'example_dataset/GMSL_TPJAOS_4.2_199209_201805.txt'


def txt_to_array(relative_path, skip_header):
    return np.genfromtxt(relative_path, skip_header=skip_header)


def array_to_dataframe(array, columns):
    return pd.DataFrame(array, columns=columns)
```

E. example_analysis.py

```
from example.gmst_handler import *
from model_design import LinearDesignModel
from data_analysis import DataAnalysis2D
import scipy.fftpack
from scipy.signal import argrelextrema
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

if __name__ == '__main__':
    data_array = txt_to_array(GSML_FILENAME, skip_header=GMSL_HDR)
    data_frame = array_to_dataframe(data_array, columns=GSML_COLUMNS)
    refined_data_frame = data_frame[['year', 'gmsl']]

    #######################################################################################

    # MODEL 1
    #######################################################################################

    # model 2 involving a trend, bias and signal
    model_1 = LinearDesignModel(basis_functions=['x^0', 'x'])

    # instantiate a data analysis model with the given data and linear model
    model_1_analysis = DataAnalysis2D(model_1, refined_data_frame['year'].values, refined_data_frame['gmsl'].values)

    # plot the trend + bias model for the data set
    model_1_analysis.plot_model(
        ylabel='GMSL variation (mm) w.r.t. 20 year mean reference',
        xlabel='Year',
        show_save=('save'),
        name='model1_plot')

    # plot the residual probability density function
```

```
    model_1_analysis.plot_residual_hist(
        ylabel='Probability density f($\epsilon$)',
        xlabel='Residuals ($\epsilon$) [mm]',
        show_save=('save'),
        name='model1_residuals')

    # print the least square solution
    print('Vector of parameters: ', model_1_analysis.unweighted_least_squares())

    # print the skewness of the residual distribution
    print('Skewness: ', model_1_analysis.skewness(model_1_analysis.unweighted_residuals(), 30))

    # print the kurtosis of the residual distribution
    print('Kurtosis: ', model_1_analysis.kurtosis_new(model_1_analysis.unweighted_residuals(), 30))

    ####################################################################################
    # MODEL 2
    ####################################################################################
    # model 2 involving a trend, bias and signal
    model_2 = LinearDesignModel(basis_functions=['x^0', 'x', 'cos(2*pi*x)', 'sin(2*pi*x)'])

    # instantiate a data analysis model with the given data and linear model
    model_2_analysis = DataAnalysis2D(model_2, refined_data_frame['year'].values, refined_data_frame['gmsl'].values)

    # plot the trend + bias model for the data set
    model_2_analysis.plot_model(
        ylabel='GMSL variation (mm) w.r.t. 20 year mean reference',
        xlabel='Year',
        show_save=('save'),
        name='model2_plot'
    )

    # plot the residual probability density function
    model_2_analysis.plot_residual_hist(

        ylabel='Probability density f($\epsilon$)',
        xlabel='Residuals ($\epsilon$) [mm]',
        show_save=('save'),
        name='model2_residuals'
    )

    # print the least square solution
    print('Vector of parameters: ', model_2_analysis.unweighted_least_squares())

    # print the skewness of the residual distribution
    print('Skewness: ', model_2_analysis.skewness(model_2_analysis.unweighted_residuals(), 30))

    # print the kurtosis of the residual distribution
    print('Kurtosis: ', model_2_analysis.kurtosis_new(model_2_analysis.unweighted_residuals(), 30))

    ####################################################################################
    # MODEL 3
```

```
    ####################################################################################

    # model 3 involving a trend, bias and signal
    model_3 = LinearDesignModel(basis_functions=['x^0', 'x', 'x^2', 'cos(2*pi*x)', 'sin(2*pi*x)'])

    # instantiate a data analysis model with the given data and linear model
    model_3_analysis = DataAnalysis2D(model_3, refined_data_frame['year'].values, refined_data_frame['gmsl'].values)

    # plot the trend + bias model for the data set
    model_3_analysis.plot_model(
        ylabel='GMSL variation (mm) w.r.t. 20 year mean reference',
        xlabel='Year',
        show_save=('save'),
        name='model3_plot'
    )

    # plot the residual probability density function
    model_3_analysis.plot_residual_hist(
        ylabel='Probability density f($\epsilon$)',
        xlabel='Residuals ($\epsilon$) [mm]',
        show_save=('save'),
        name='model3_residuals'
    )

    # print the least square solution
    print('Vector of parameters: ', model_3_analysis.unweighted_least_squares())

    # print the skewness of the residual distribution
    print('Skewness: ', model_3_analysis.skewness(model_3_analysis.unweighted_residuals(), 30))

    # print the kurtosis of the residual distribution
    print('Kurtosis: ', model_3_analysis.kurtosis_new(model_3_analysis.unweighted_residuals(), 30))

    ####################################################################################
    # FOURIER ANALYSIS
    ####################################################################################

    x = refined_data_frame['year'].values
    y = refined_data_frame['gmsl'].values

    # Number of sample points
    N = len(x)

    # sample spacing
    T = (y[-1]-y[0]) / N

    # fast fourier transform
    yf = scipy.fftpack.fft(y)
    xf = np.linspace(0.0, 1.0 / (2.0 * T), N / 2)

    refined = 2.0 / N * np.abs(yf[:N // 2])
    idx = argrelextrema(refined, np.greater)

    figure(num=None, figsize=(8, 5), dpi=300, facecolor='w', edgecolor='k')
    plt.grid()
    # fig, ax = plt.subplots()
```

```python
    # plt.plot(xf, np.log(2.0 / N * np.abs(yf[:N // 2])))
    print(refined[idx][refined[idx] >= 1.0])
    print(xf[idx][refined[idx] >= 1.0])

    plt.semilogy(xf, 2.0 / N * np.abs(yf[:N // 2]))
    plt.semilogy(xf[idx][refined[idx] >= 1.0], refined[idx][refined[idx] >= 1.0], linewidth=0, marker='o')
    plt.ylabel('Amplitude [mm]')
    plt.xlabel('Frequency (per year)')
    plt.savefig('plots/fourier_analysis.png')
    # plt.show()

    ##########################################################################################
    # MODEL 4
    ##########################################################################################
    f = [0.01358823, 0.13588228, 0.19023519, 0.27176456, 0.3397057, 0.69299964]
    _basis_functions = ['x^0', 'x', 'x^2', 'cos(2*pi*x)', 'sin(2*pi*x)']
    for freq in f:
        _basis_functions += ['cos(2*pi*x*{})'.format(str(freq)), 'sin(2*pi*x*{})'.format(str(freq))]

    print(_basis_functions)

    # model 4 involving a trend, bias and signal
    model_4 = LinearDesignModel(basis_functions=_basis_functions)

    # instantiate a data analysis model with the given data and linear model
    model_4_analysis = DataAnalysis2D(model_4, refined_data_frame['year'].values, refined_data_frame['gmsl'].values)

    # plot the trend + bias model for the data set
    model_4_analysis.plot_model(
        ylabel='GMSL variation (mm) w.r.t. 20 year mean reference',
        xlabel='Year',
        show_save=('save'),
        name='model4_plot',
        legend=False
    )

    # plot the residual probability density function
    model_4_analysis.plot_residual_hist(
        ylabel='Probability density f($\epsilon$)',
        xlabel='Residuals ($\epsilon$) [mm]',
        show_save=('save'),
        name='model4_residuals',
    )

    # print the least square solution
    print('Vector of parameters: ', model_4_analysis.unweighted_least_squares())

    # print the skewness of the residual distribution
    print('Skewness: ', model_4_analysis.skewness(model_4_analysis.unweighted_residuals(), 30))

    # print the kurtosis of the residual distribution
    print('Kurtosis: ', model_4_analysis.kurtosis_new(model_4_analysis.unweighted_residuals(), 30))
```