Assignment 1

Lauge Thode Hermansen
s204111

Torben T. Nguyen
s204121

Vincent Olesen
s184017

Tymoteusz Barcinski
s221937

February 23, 2023

1 Data set

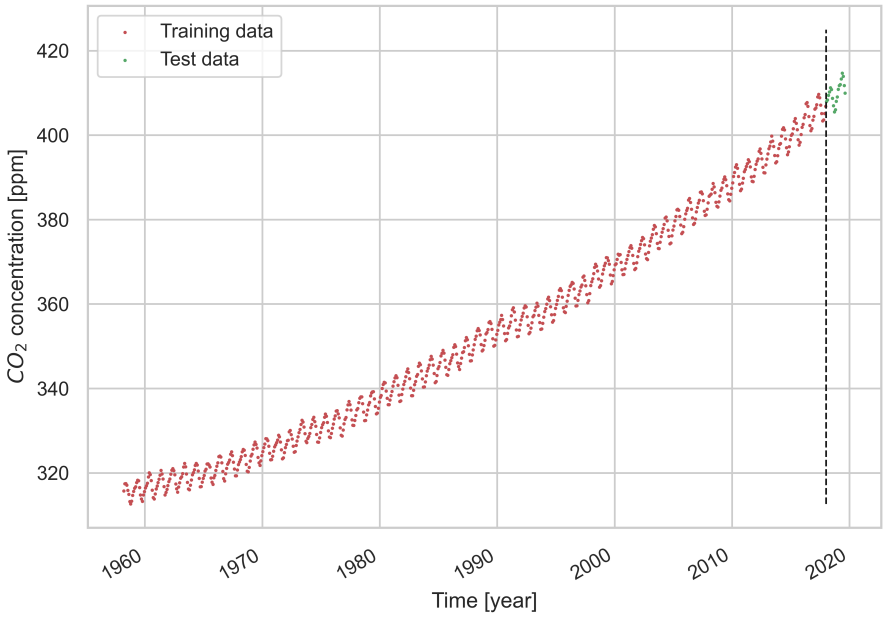


Figure 1: An example graph

This assignment analyzes monthly atmospheric CO_2 concentrations from Mauna Loa, Hawaii, using data available from the National Oceanic and Atmospheric Administration's Earth System Research Laboratory (NOAA/ESRL).

The data set consists of monthly observations of CO_2 concentration in parts per million (PPM) from January 1970 to January 2020, that is 738 observations.

For the purpose of this assignment, we divided the data into a training set consisting of the first 718 observations and a test set consisting of the last 20 observations.

There appears to be a global trend of it increasing slightly super-linearly. There also appears to be minor local harmonic oscillations with an annual period, approximately. Overall, the relation between CO_2 and time seems to contain a lot of structure that may be modeled.

The provided data is given in regular time increments of 0.083 or 0.084, where the unit is a year. As observations are approximately equally spaced in time, we can instead consider $[1 \ldots, N]$ as an

appropriate index of time, where $N = 738$. For model interpretability and ease of use reasons, the time is therefore transformed to have integer increments of 1, where the unit is a month.

2 General linear model

We use the observation index $[1 \dots N]$ as the time index throughout the models in the report, where $N = 718$ is the number of observations in the training set. From fig. 1 a reasonable model would include an intercept, a linear trend and an annual harmonic part. We consider the following model suggested in the assignment

$$Y_t = \alpha + \beta_t t + \beta_s \sin\left(\frac{2\pi}{T}t\right) + \beta_c \cos\left(\frac{2\pi}{T}t\right) + \varepsilon_t, \quad t \in \{0, \dots, N-1\} \tag{1}$$

or in vector notation

$$Y_t = \boldsymbol{x}_t^\top \boldsymbol{\theta} + \varepsilon_t, \quad \boldsymbol{x}_t = \begin{pmatrix} 1 \\ t \\ \sin\left(\frac{2\pi}{T}t\right) \\ \cos\left(\frac{2\pi}{T}t\right) \end{pmatrix}, \quad \boldsymbol{\theta} = \begin{pmatrix} \alpha \\ \beta_t \\ \beta_s \\ \beta_c \end{pmatrix} \quad t \in \{0, \dots, N-1\} \tag{2}$$

with the period $T = 12$ months. We assume that $\mathrm{E}[\varepsilon_t] = 0$ and $\mathrm{Cov}[\varepsilon_i, \varepsilon_j] = \sigma^2 \boldsymbol{\Sigma}_{ij}$, where $\boldsymbol{\Sigma}$ is the covariance matrix.

2.1 Ordinary least-squares parameter estimation

To the parameters of eq. (1), we will firstly consider parameter estimation using ordinary least-squares (OLS) regression. For that, assume that $\boldsymbol{\Sigma} = \boldsymbol{I}$ which indicates that each ε_i for $i = 1 \dots N$

From [3, Theorem 3.1], the OLS parameter estimate, is given by

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{y}, \quad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^\top \\ \vdots \\ \boldsymbol{x}_N^\top \end{pmatrix} \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

The uncertainty of the parameter estimate, in this case represented by the variance of the parameters, is derived in [3, Theorem 3.2], and is given by

$$\mathrm{Var}\left[\hat{\boldsymbol{\theta}}\right] = \sigma^2 \left(\boldsymbol{X}^\top \boldsymbol{X}\right)^{-1} \tag{3}$$

An unbiased estimate of σ^2 is given by [3, Theorem 3.4], with $\boldsymbol{\Sigma} = \boldsymbol{I}$, which is an assumption made by the OLS. That is

$$\hat{\sigma}^2 = \frac{(\boldsymbol{y} - \boldsymbol{X}^\top \hat{\boldsymbol{\theta}})^\top (\boldsymbol{y} - \boldsymbol{X}^\top \hat{\boldsymbol{\theta}})}{N - p} \tag{4}$$

where N is the size of the training set, and $p = 4$ is the number of model parameters. Specifically, $\hat{\sigma}^2$ was estimated to be 12.185.

The resulting parameter estimates and the standard deviation of the parameter estimates are presented in table 2. Further, the fitted data and its residuals can be seen in respectively fig. 2 and fig. 4.

2.2 Weighted least-squares parameter estimation

From fig. 4, it is clear that the residuals of the OLS regression are correlated. That is, the assumption of iid. residuals in OLS is violated. Consequently, we will assume that the correlation structure of the residuals is an exponential decaying function of the time distance between two observations, that is

$$\mathrm{Cor}[\varepsilon_i, \varepsilon_j] = \rho^{|i-j|} \tag{5}$$

Consequently, the covariance structure of the residual is given by

$$\sigma^2\boldsymbol{\Sigma} = \sigma^2 \begin{pmatrix} 1 & \rho & \rho^2 & \cdots \\ \rho & 1 & \rho & \cdots \\ \rho^2 & \rho & 1 & \cdots \\ \vdots & \vdots & \vdots & \end{pmatrix} \tag{6}$$

Under the above assumption, and assuming $\boldsymbol{\Sigma}$ is known, the parameters of the model and the uncertainty of these, can be estimated by using weighted least-squares (WLS), in a similar way to the OLS

$$\hat{\boldsymbol{\theta}} = \left(\boldsymbol{X}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{X}\right)^{-1} \boldsymbol{X}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{y} \tag{7}$$

$$\mathrm{Var}\left[\hat{\boldsymbol{\theta}}\right] = \sigma^2 \left(\boldsymbol{X}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{X}\right)^{-1} \tag{8}$$

which is given in respectively [3, Theorem 3.3] and [3, Theorem 3.4].

An unbiased estimate of σ^2 is given by [3, Theorem 3.4]

$$\hat{\sigma}^2 = \frac{(\boldsymbol{y} - \boldsymbol{X}^\top \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{y} - \boldsymbol{X}^\top \hat{\boldsymbol{\theta}})}{N - p}$$

where N is the size of the training set, and $p = 4$ is the number of model parameters. Specifically, $\hat{\sigma}^2$ was estimated to be 12.26813.

As ρ and consequently $\boldsymbol{\Sigma}$ is unknown, an approach to WLS parameter estimation is the Relaxation Algorithm given in [2, p. 49]. An initial correlation structure of $\boldsymbol{\Sigma}_0 = \boldsymbol{I}$ is used. To improve the estimate of $\boldsymbol{\Sigma}$, ρ was estimated in each iteration using

$$\hat{\rho} = \mathrm{Cor}\left[\boldsymbol{\varepsilon}, \bar{\boldsymbol{\varepsilon}}\right], \quad \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_{N-1} \end{pmatrix}, \quad \bar{\boldsymbol{\varepsilon}} = \begin{pmatrix} \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

that is the lag 1 auto-correlation of the residuals.

Using five iterations of the Relaxation Algorithm, the following correlations were estimated. From table 1, it is clear, that the estimate of ρ converges quickly.

Iteration	ρ
1	0.98209
2	0.98224
3	0.98224
4	0.98224
5	0.98224

Table 1: Estimates of the correlation of residuals, using five iterations of the Relaxation Algorithm

The resulting parameter estimates and the standard deviation of the parameter estimates are presented in table 2. Further, the fitted data and its residuals can be seen in respectively fig. 2 and fig. 4.

2.3 Comparison of OLS and WLS parameter estimation

The parameter estimates using OLS and WLS for eq. (1), using the training data set is given in table 2.

	OLS		WLS	
	Estimate	Std.	Estimate	Std.
α	306.928362	0.26028527	307.470864	2.3182772
β_t	0.128374044	0.00062856	0.129839150	0.00538119
β_s	1.69090265	0.18423962	1.66890379	0.06760636
β_c	2.25306869	0.18423962	2.29457132	0.06760636

Table 2: General linear model parameter estimates and standard deviation of parameter estimates, using respectively OLS and WLS.

The fitted models along with the training and test data are given in fig. 2 and fig. 3.

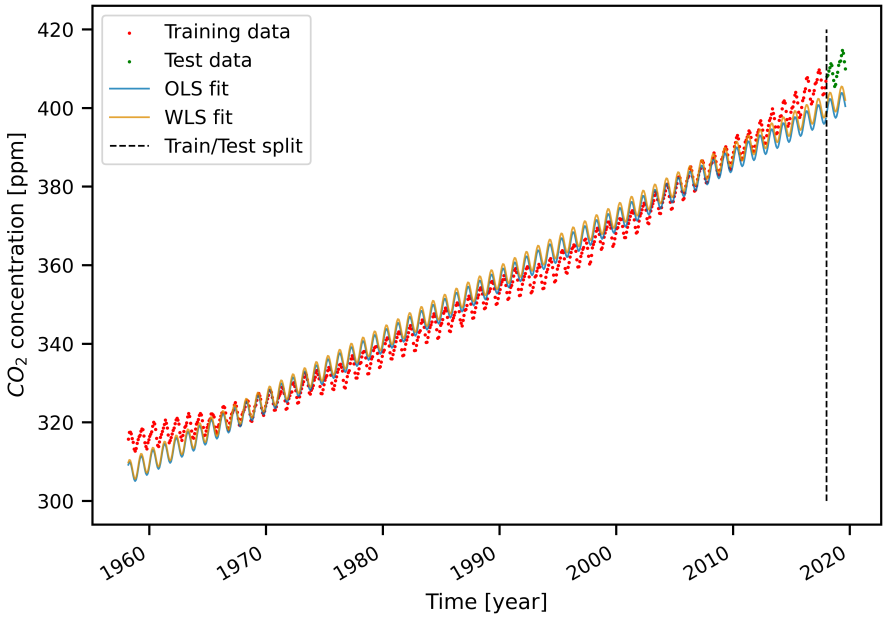


Figure 2: Plot of the fitted general linear model, fitted using OLS and WLS.

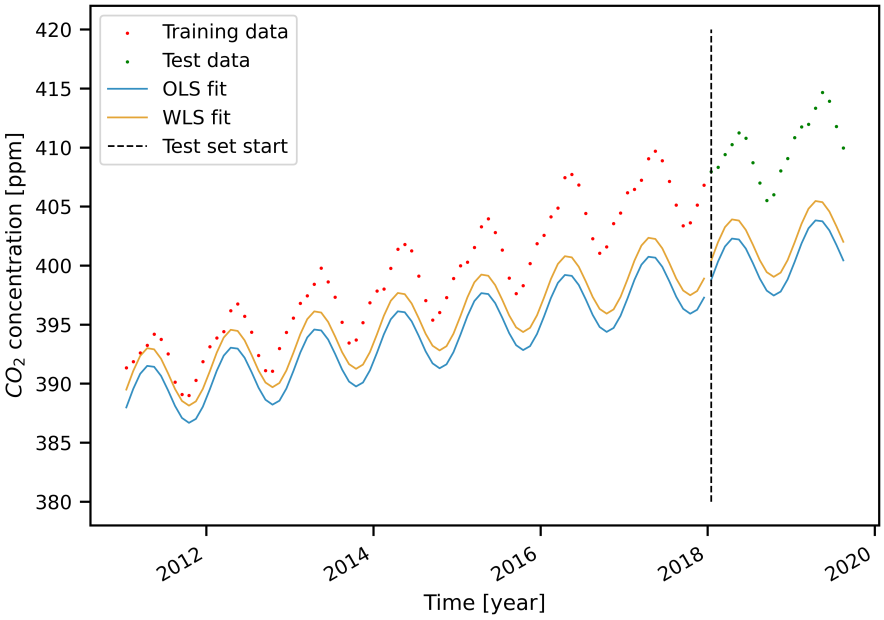


Figure 3: Plot of the fitted general linear model, fitted using OLS and WLS.

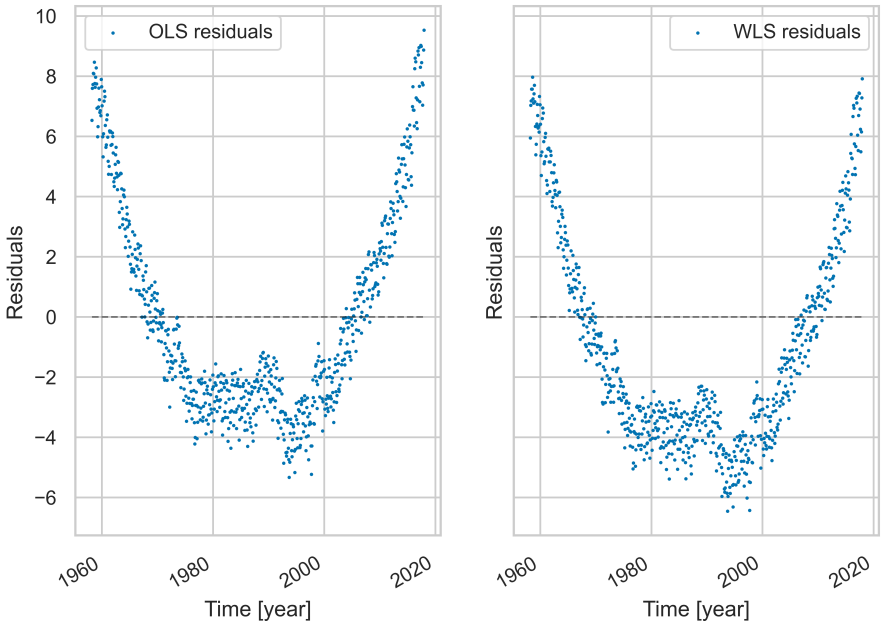


Figure 4: Residuals of linear model, estimated with respectively OLS and WLS.

From table 2, it can be seen that the estimated model parameters do not differ significantly.

From fig. 2 and fig. 4 it is clear, that the linear model, regardless of the parameter estimation method, fails to model the increasing trend in the CO_2 concentration. Further, for the residuals in fig. 4, it can be seen for OLS, that the assumption of identically distributed and independent residuals is violated. It motivates the consideration of the WLS estimator which assumes additional structure on the covariance matrix. After the introduction of the covariance structure, the correlation of the residuals is taken into account when the estimation of parameters is performed. It implies that the assumptions on [3, Theorem 3.3] and [3, Theorem 3.4] are violated.

3 Local linear trend model

3.1 Model formulation

The previous OLS model structure, eq. (1), is adapted to a local linear trend model (LLT) according to [3, Section 3.4.3]. Set $\lambda = 0.9$.

$$Y_{t+j} = \boldsymbol{f}(j)^\top \boldsymbol{\theta}_t + \epsilon_{t+j},$$

where the ϵ_{t+j} are assumed to be mutually independent, identically distributed, and with zero mean, and constant variance σ^2. First, a feature/design vector function, $f(t)$, that maps from a time point t to a feature vector must be defined to follow the model given in the phrasing of the assignment

$$\boldsymbol{f}(t) = \begin{pmatrix} 1 \\ t \\ sin\left(\frac{2\pi}{p}t\right) \\ cos\left(\frac{2\pi}{p}t\right) \end{pmatrix}, \quad f(0) = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

A transition matrix, $\boldsymbol{L}$, such that $\boldsymbol{f}(t+1) = \boldsymbol{L}\boldsymbol{f}(t)$ is found using the rules for $\cos(a+b)$ and $\sin(a+b)$:

$$\boldsymbol{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & \cos\left(\frac{2\pi}{p}\right) & \sin\left(\frac{2\pi}{p}\right) \\ 0 & 0 & -\sin\left(\frac{2\pi}{p}\right) & \cos\left(\frac{2\pi}{p}\right) \end{pmatrix}$$

The model parameters are estimated/updated [3, Theorem 3.12] and predictions [3, Section 3.4.3.2] are made via the following procedure:

Algorithm 1 Iterative parameter estimation for LLT models

$(b, \lambda) \leftarrow (10, 0.9)$ ▷ Burn-in, "forgetting factor"
$\boldsymbol{F_0}, \boldsymbol{h}_0, \hat{\sigma}_b \leftarrow \boldsymbol{0}$
for $t \leftarrow 0...N-1$ **do**
  if $t \geq b$ **then** ▷ Are we past burn-in?
    $\hat{Y}_{t+1} \leftarrow \boldsymbol{f}(1)^\top \boldsymbol{\theta_t}$ ▷ Predict
    $\hat{\sigma}^2_{t+1} = \frac{(t-b)\hat{\sigma}^2_t + \frac{(Y_{t+1}-\hat{Y}_{t+1})^2}{1+\boldsymbol{f}(1)^\top \boldsymbol{F}_t^{-1}\boldsymbol{f}(1)}}{t-b+1}$ ▷ (Recursive rewrite of [1, Lecture 4, part D])
  end if
  $\boldsymbol{F}_{t+1} \leftarrow \boldsymbol{F}_t + \lambda^t \boldsymbol{f}(-t)\boldsymbol{f}(-t)^\top$
  $\boldsymbol{h}_{t+1} \leftarrow \lambda \boldsymbol{L}^{-1}\boldsymbol{h}_t + \boldsymbol{f}(0) Y_{t+1}$ ▷ Indexing for targets start at 1, so Y_i, where $i = 1, 2, \cdots$
  $\boldsymbol{\theta}_{t+1} = \boldsymbol{F}^{-1}_{\boldsymbol{t+1}}\boldsymbol{h}_{\boldsymbol{t+1}}$ ▷ Only when inverse exists
end for
$\hat{Y}_{\text{test}} \leftarrow \boldsymbol{f}(1...N_{\text{test}})^\top \boldsymbol{\theta_N}$ ▷ Predict test set

3.2 Verification

The above model is fitted incrementally to the training data to yield one-step predictions. From these, residuals are calculated and plotted for model verification.

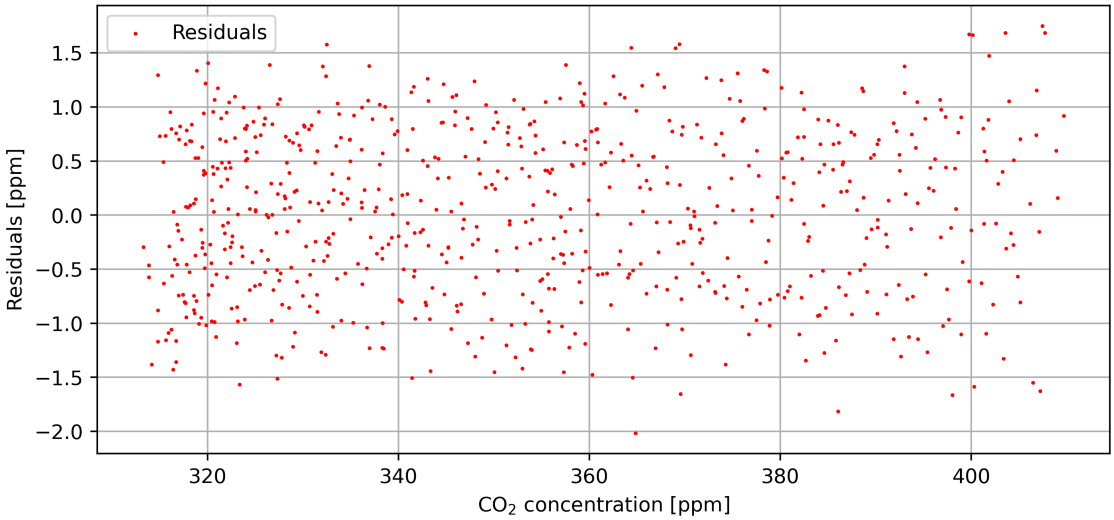


Figure 5: Training data residuals plotted as a function of CO_2 concentration.

From fig. 5 the residuals appear homoscedastic and with mean 0. This appears to support the model assumptions previously mentioned in section 3.1.

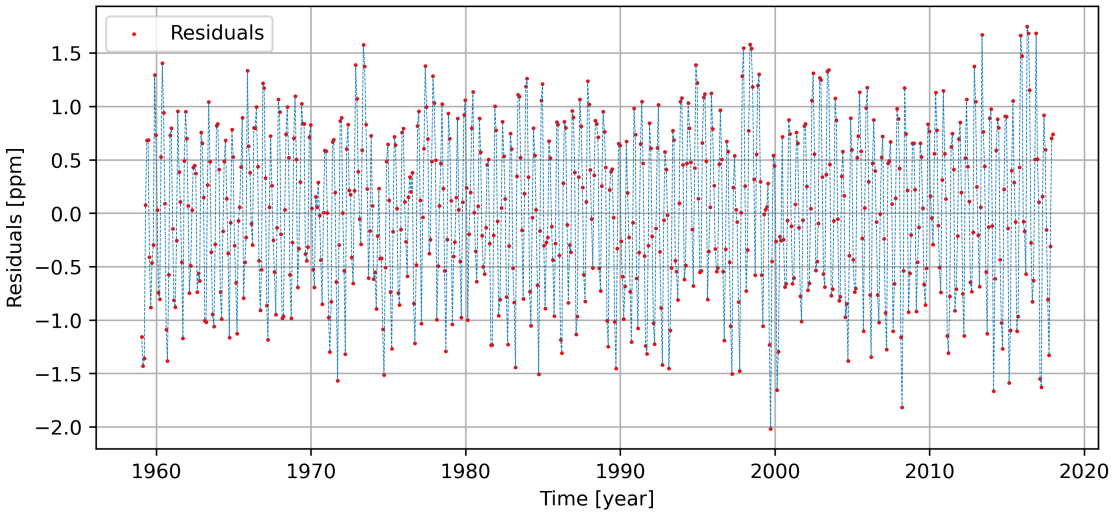


Figure 6: Training data residuals plotted as a function of time.

However, the plot shown in Figure 6 presents a different perspective. Although the residuals seem to exhibit homoscedasticity over time, they are not independent and identically distributed due to the occurrence of neighboring elements with opposite signs.

The model is therefore invalid since an assumption about ϵ_t is violated. The exercise, however, asks us to make prediction intervals later, and so far the course has only introduced t-distributions. The residuals are therefore assumed to still be kind of i.i.d. via the normal distribution (despite not even having normal density). This is to enable the calculation of prediction intervals. Non-parametric methods are deemed out of scope.

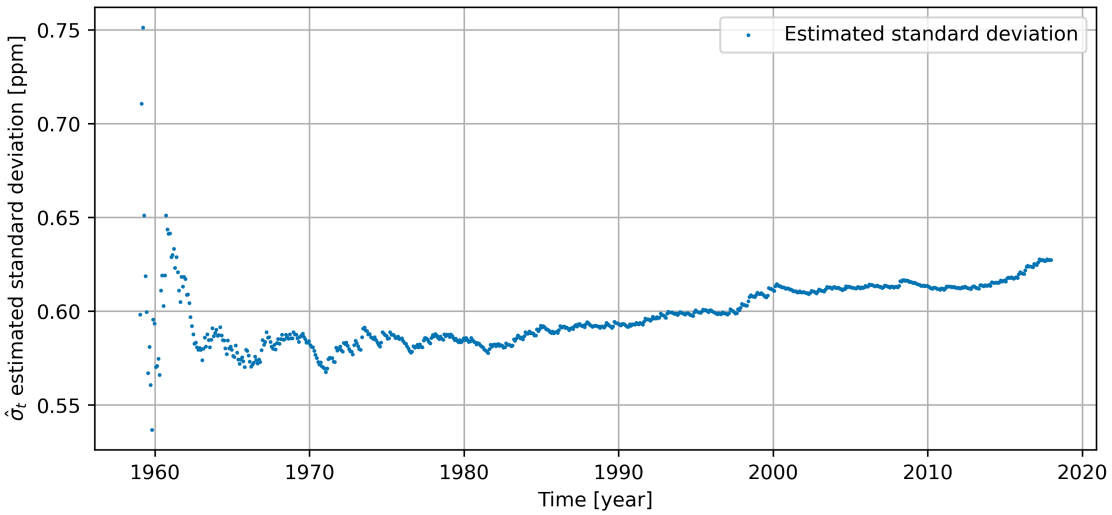


Figure 7: Global estimates of one-step prediction residual standard deviation.

Since the residual variance appears relatively constant, the global estimator, mentioned in section 3.1, for the variance is used. The estimates are shown on fig. 7, with a final estimate from the training data of approximately 0.627 for the standard deviation.

3.3 Prediction

The LLT one-step predictions (including 95% prediction intervals) are plotted on top of the training and test data. The estimated mean for each time step in the training data is also plotted. Recall, that α in eq. (1) is the estimated mean. This mean is not plotted for the test data as it would just be a horizontal line since the model is not updated anymore.

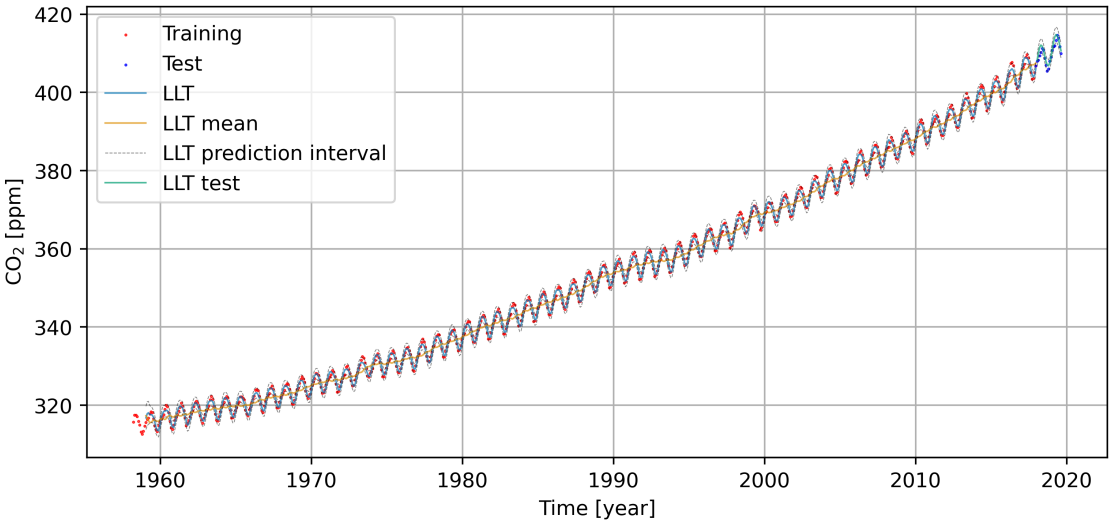


Figure 8: Full local linear trend one-step predictions on training and test data.

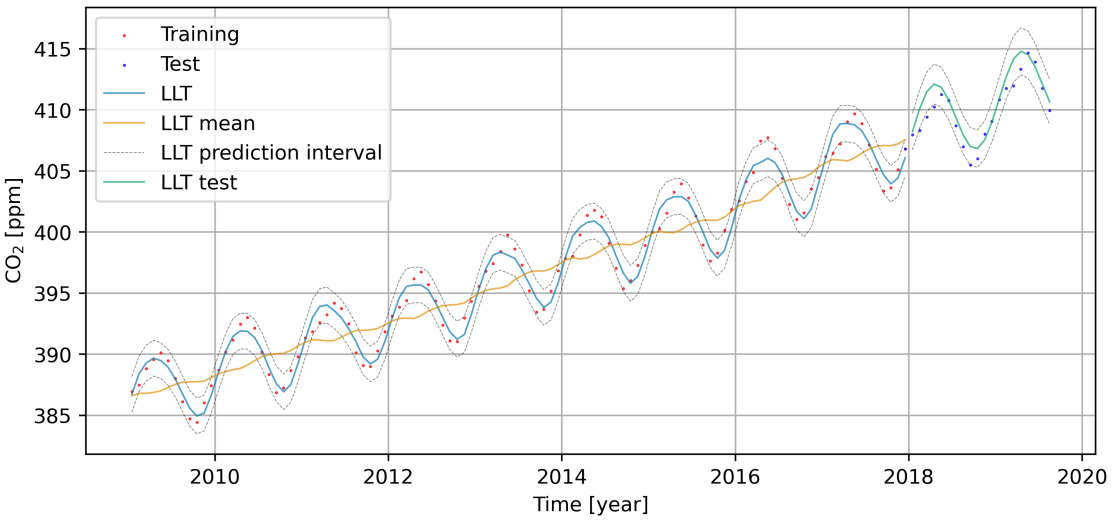


Figure 9: Zoomed local linear trend one-step predictions on training and test data.

The mean prediction seems to filter away the annual oscillations well while staying true to the data. The model performance on the training and test data appears to be really good. The annual oscillations are captured together with the general upwards trend. It even adapts surprisingly well to "kinks" in the general upwards trend as seen around the year 1990. These patterns transfer well to the test data. Predictions also appear to be well within the prediction intervals. This is especially clear from 2010 onwards, where the prediction intervals clearly suffer from overcoverage.

	CO_2		**PI**	
Month	Y	$\hat{Y}$	2.5%	97.5%
1	407.96	408.21	406.72	409.71
2	408.32	410.08	408.51	411.64
6	410.79	410.83	409.22	412.44
12	409.07	409.00	407.37	410.62
20	409.95	410.64	408.81	412.48

Table 3: Table of future month LLT predictions on test data.

Looking at table 3 the overcoverage is demonstrated further. This likely stems from the violation of the model assumptions mentioned earlier. However, again - it is also clear that the model predictions are very accurate and close to the true targets.

4 Optimal λ

4.1 Finding optimal λ

By adjusting the forgetting factor λ in the local trend model, one can achieve a balance between the effective number of observations used for parameter estimation and the model's ability to capture local trends. This hyperparameter can be estimated by defining an appropriate loss function and conducting an optimization procedure. In [3, Section 3.4.2.2], the sum of squared one-step prediction errors (SSE) is proposed as the objective function for minimizing to find the optimal λ^*. To determine λ^*, the first 100 observations were excluded, and the sum of squared residuals was plotted as a function of λ, as shown in Figure 10. The results indicate that the optimal value of $\lambda^* = 0.933$.

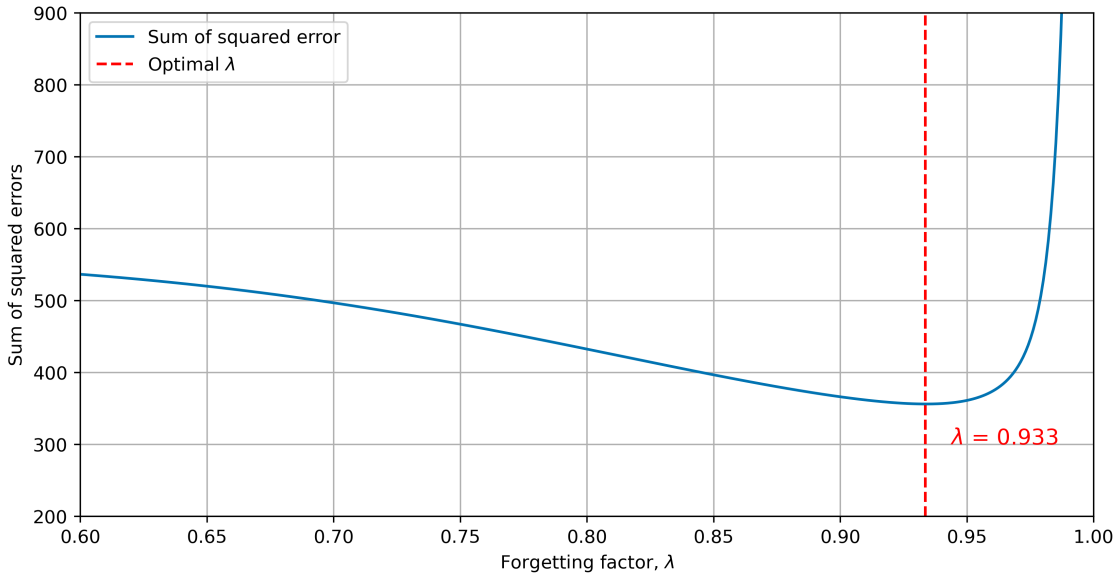


Figure 10: Sum of squared residuals as a function of λ

Figures fig. 11 and fig. 12 present the training and test data filtered with the local trend model with the optimal λ. The results are not visibly different from the predictions obtained in the previous section. All target values, except for one, are within the prediction intervals which indicates a good fit. However, the prediction interval is invalid, because of assumption violations.

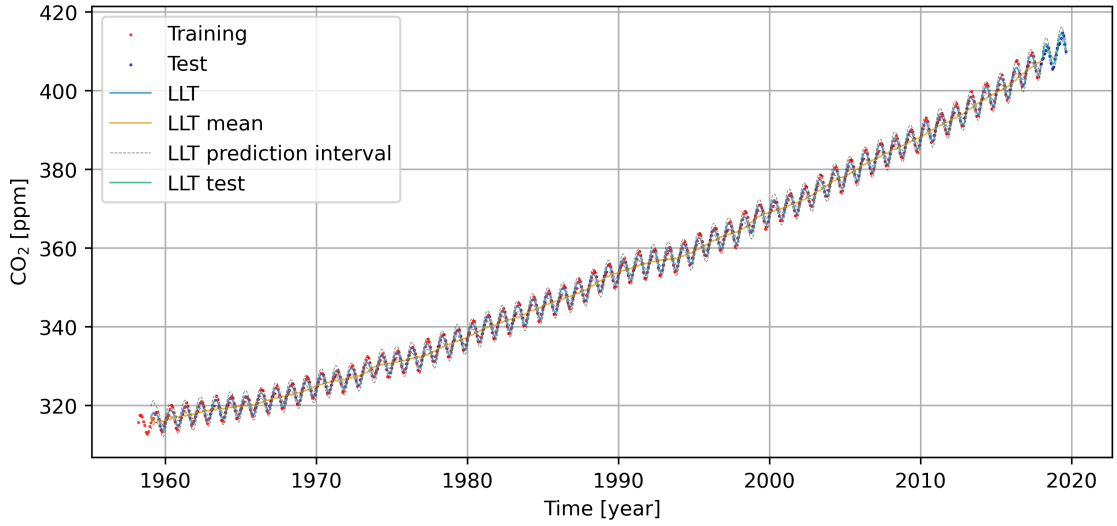


Figure 11: Full local linear trend one-step predictions on training and test data for optimal λ

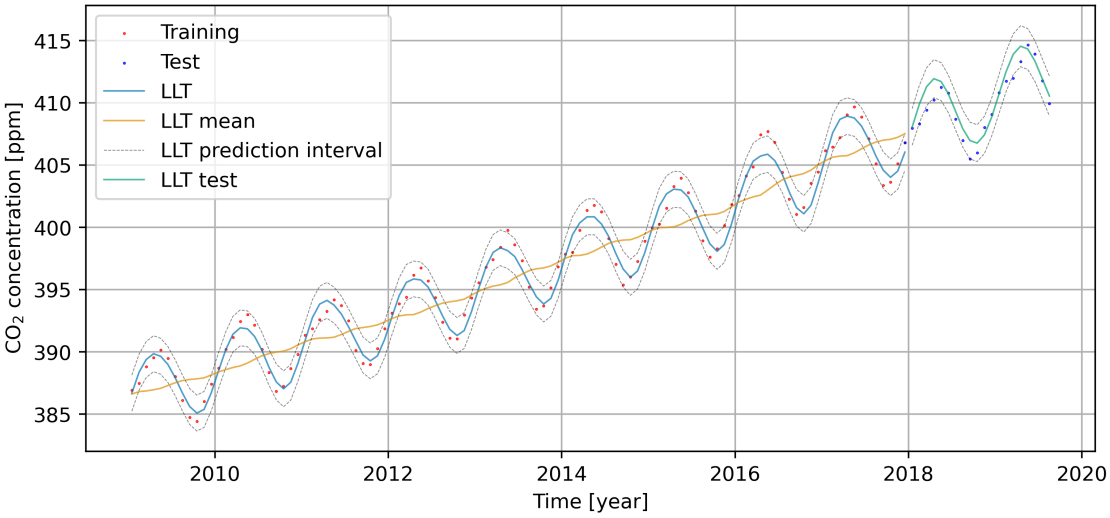


Figure 12: Zoomed local linear trend one-step predictions on training and test data for optimal λ

4.2 Comparison of local trend models with different values of λ

Comparing the results from the table section 4.2 with table 3 it can be concluded that the prediction intervals shrink slightly for the optimal λ. The sum of squared one-step prediction errors for the two models on the test set are as follows: $SSE_{\lambda=0.9} = 24.63958$, $SSE_{\lambda=0.933} = 20.27077$. There is a 17% reduction in the SSE when optimal λ is used.

	CO_2		**PI**	
Month	Y	$\hat{Y}$	2.5%	97.5%
1	407.96	408.07	406.60	409.54
2	408.32	409.90	408.40	411.41
6	410.79	410.74	409.23	412.25
12	409.07	408.84	407.32	410.36
20	409.95	410.54	408.95	412.14

Table 4: Table of future month LLT predictions on test data for optimal λ

5 Conclusion

Based on the performed analysis, the local trend model is chosen over the OLS and WLS models as it performs better on the test set, has smaller and well-behaved residuals, and exhibits an ability to capture the local trend in the data by the reestimation of parameters. A possible extension to the OLS and WLS models would include the quadratic term with respect to time index. Possible extensions to the local trend model would include the addition of the quadratic term with respect to time index, the addition of a higher frequency harmonic term to deal with the oscillations seen in the residuals, specification of the desired prediction horizons and including it in the loss function in the optimization procedure for optimal lambda. To investigate whether the above-mentioned extensions would improve the model, one should compare the loss on the test set, analyze the residuals, and check that the model assumptions are satisfied.

References

[1] Lasse Engbo Christiansen. *02417 lectures*. 2019.

[2] Graham C. Goodwin and Robert L. Payne, eds. *Edited by*. Vol. 136. Mathematics in Science and Engineering. Elsevier, 1977, p. iii. DOI: https://doi.org/10.1016/S0076-5392(08)63227-0. URL: https://www.sciencedirect.com/science/article/pii/S0076539208632270.

[3] Henrik Madsen. *Time Series Analysis*. Oct. 2008. ISBN: 978-1-4200-5967-0. DOI: 10.1201/9781420059687.

Appendix: Code

Question 1.1

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import scipy
import scipy.stats as stats
import seaborn as sns
from tqdm import tqdm
```

In [2]:
```
df = pd.read_csv('./A1_co2.txt', sep=' ')
df.head()
```

Out[2]:

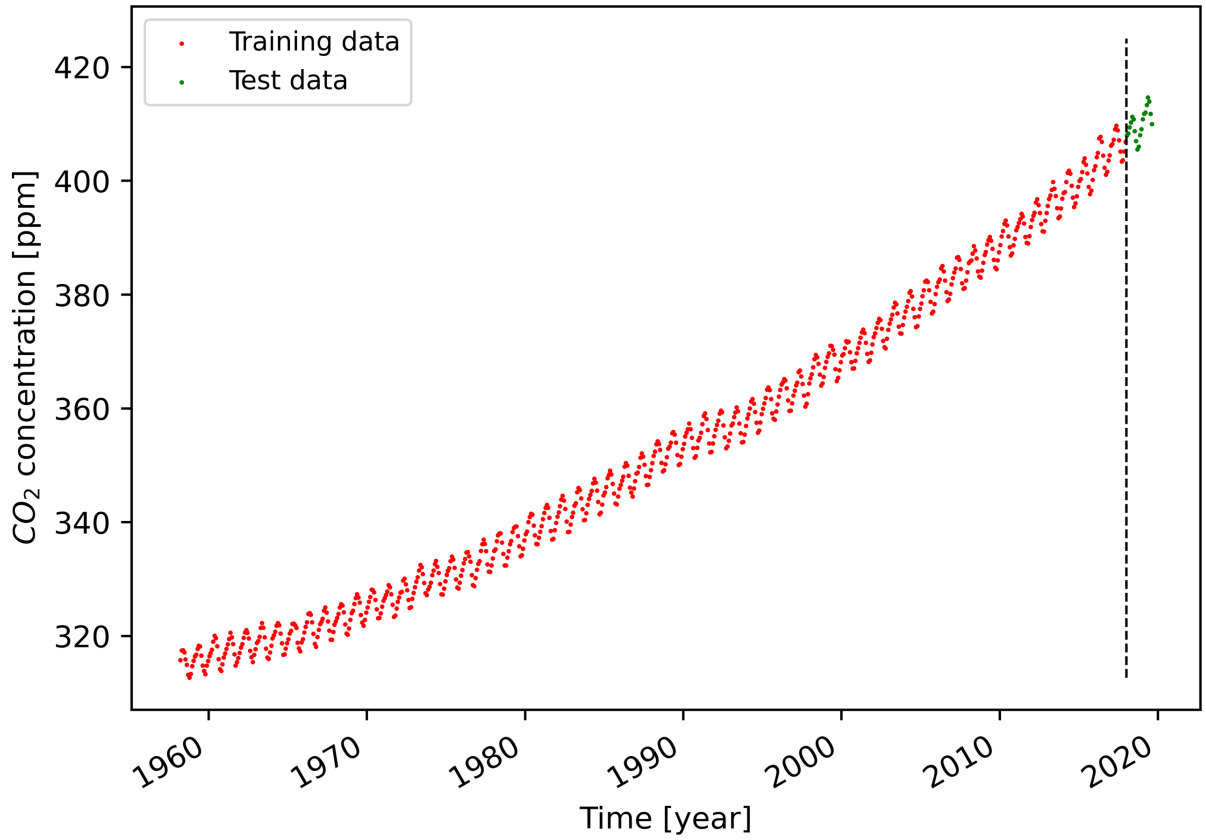
	year	month	time	co2
0	1958	3	1958.208	315.71
1	1958	4	1958.292	317.45
2	1958	5	1958.375	317.50
3	1958	6	1958.458	317.10
4	1958	7	1958.542	315.86

> You should not use the observations for years 2018 and 2019 (Last 20 observations) for estimations/training - only for comparisons/testing.

In [3]:
```
# Let the train set be the measurements before 2018, and the test set be
train = df[df['year'] < 2018]
test = df[df['year'] >= 2018]
```

In [4]:
```
# Plot the data
fig, ax = plt.subplots(dpi=800)

# Set the style
sns.set_context("paper", rc={"lines.linewidth": 0.8})
sns.set_palette("colorblind")

ax.scatter(train["time"], train["co2"], label="Training data", s=0.5, c="
ax.scatter(test["time"], test["co2"], label="Test data", s=0.5, c="g")

ax.vlines(2018, train["co2"].min(), 425, linestyles="dashed", colors="k")

ax.legend()

ax.set_xlabel("Time [year]")
ax.set_ylabel("$CO_2$ concentration [ppm]")

fig.autofmt_xdate()
fig.show()
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/915935225.py:19: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```



Question 1.2

Question 1.2.1

In [5]:
```
MONTHS_IN_YEAR = 12

X_train = np.vstack(
    [
        np.ones_like(train.index),
        train.index,
        np.sin(2 * np.pi / MONTHS_IN_YEAR * train.index),
        np.cos(2 * np.pi / MONTHS_IN_YEAR * train.index),
    ]
).T

y_train = train["co2"].values


X_test = np.vstack(
    [
        np.ones_like(test.index),
        test.index,
        np.sin(2 * np.pi / MONTHS_IN_YEAR * test.index),
        np.cos(2 * np.pi / MONTHS_IN_YEAR * test.index),
    ]
).T

y_test = test["co2"].values
```

In [6]:
```
beta_ols, _, _, _ = np.linalg.lstsq(X_train, y_train)
beta_ols
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/2496290795.py:1: FutureWarning: `rcond` parameter will change to the default of machine precision times ``max(M, N)`` where M and N are the input matrix dimensions.
To use the future default and silence this warning we advise to pass `rcond=None`, to keep using the old, explicitly pass `rcond=-1`.
  beta_ols, _, _, _ = np.linalg.lstsq(X_train, y_train)
```

Out[6]:
```
array([3.06928362e+02, 1.28374044e-01, 1.69090265e+00, 2.25306869e+00])
```

Question 1.2.2

In [7]:
```
N, p = X_train.shape

residuals = y_train - X_train @ beta_ols

sigma_ols2 = np.sum(residuals ** 2) / (N - p)
sigma_ols2
```

Out[7]:
```
12.185026694430931
```

In [8]:
```
sigma_ols = np.sqrt(sigma_ols2)
sigma_ols
```

Out[8]:
```
3.490705758787316
```

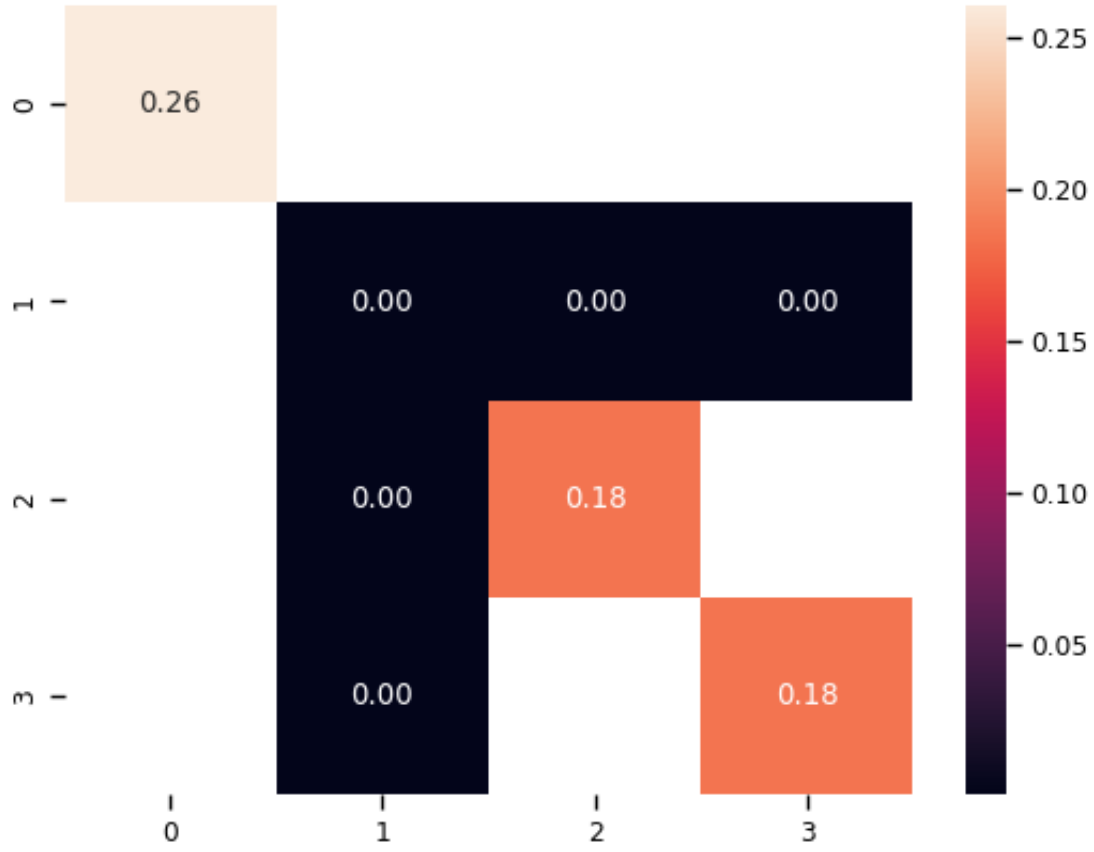
In [9]:
```
var_beta_ols = sigma_ols2 * np.linalg.inv(X_train.T @ X_train)
var_beta_ols
```

Out[9]:
```
array([[ 6.77484239e-02, -1.41638461e-04, -4.00049374e-04,
        -2.70584607e-04],
       [-1.41638461e-04,  3.95086363e-07,  9.35333307e-07,
         9.35333307e-07],
       [-4.00049374e-04,  9.35333307e-07,  3.39442373e-02,
        -7.99112497e-05],
       [-2.70584607e-04,  9.35333307e-07, -7.99112497e-05,
         3.39442373e-02]])
```

In [10]:
```
# Heatmap of the variance of the coefficients
sns.heatmap(np.sqrt(var_beta_ols), annot=True, fmt=".2f")
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/3628586960.py:2: RuntimeWarning: invalid value encountered in sqrt
  sns.heatmap(np.sqrt(var_beta_ols), annot=True, fmt=".2f")
```

Out[10]:
```
<AxesSubplot: >
```



In [11]:
```
np.sqrt(np.diag(var_beta_ols))
```

Out[11]:
```
array([0.26028527, 0.00062856, 0.18423962, 0.18423962])
```

Question 1.2.3

Question 1.2.4

Plan-of-attack: Relaxtion algorithm

In [12]:
```
def rho_matrix(rho: float, n: int):
    """
    Returns the covariance matrix for the observations of a stationary AR
    """
    rhos = np.vander([rho], n, increasing=True)
    return scipy.linalg.toeplitz(rhos)


N, p = X_train.shape

# Initial guess of correlation structure
Sigma_wls = np.eye(N)

# Initial guess of coefficients
beta_wls = beta_ols

betas_wls = []

for _ in range(6):
    # E-step
    Sigma_inv = np.linalg.inv(Sigma_wls)

    H = np.linalg.inv(X_train.T @ Sigma_inv @ X_train) @ X_train.T @ Sigm

    beta_new = H @ y_train

    # M-step
    residuals = y_train - X_train @ beta_new

    rho = np.corrcoef(residuals[:-1], residuals[1:])[0, 1] # 1-lag autoco
    print(rho)

    Sigma_new = rho_matrix(rho, N)

    #if np.allclose(beta, beta_new, atol=1e-6) and np.allclose(Sigma, Sig
    #   break

    beta_wls = beta_new
    Sigma_wls = Sigma_new

    betas_wls.append(beta_wls)

beta_wls
```

```
0.9820921007263159
0.9822410526176385
0.9822427429486106
0.982242762307222
0.9822427625289551
0.9822427625314976
```

Out[12]:
```
array([3.07470864e+02, 1.29839150e-01, 1.66890379e+00, 2.29457132e+00])
```

Question 1.2.6

In [13]:
```
# Calculate the variance of the coefficients
sigma_wls2 = (residuals.T @ np.linalg.inv(Sigma_wls) @ residuals ) / (N -

var_beta_wls = np.linalg.inv(X_train.T @ np.linalg.inv(Sigma_wls) @ X_tra
var_beta_wls
```

Out[13]:
```
array([[ 5.37440920e+00, -1.03811751e-02, -5.30015696e-04,
        -1.12349504e-03],
       [-1.03811751e-02,  2.89572528e-05,  2.30615165e-06,
         2.30615165e-06],
       [-5.30015696e-04,  2.30615165e-06,  4.57062027e-03,
         1.28481766e-05],
       [-1.12349504e-03,  2.30615165e-06,  1.28481766e-05,
         4.57062027e-03]])
```

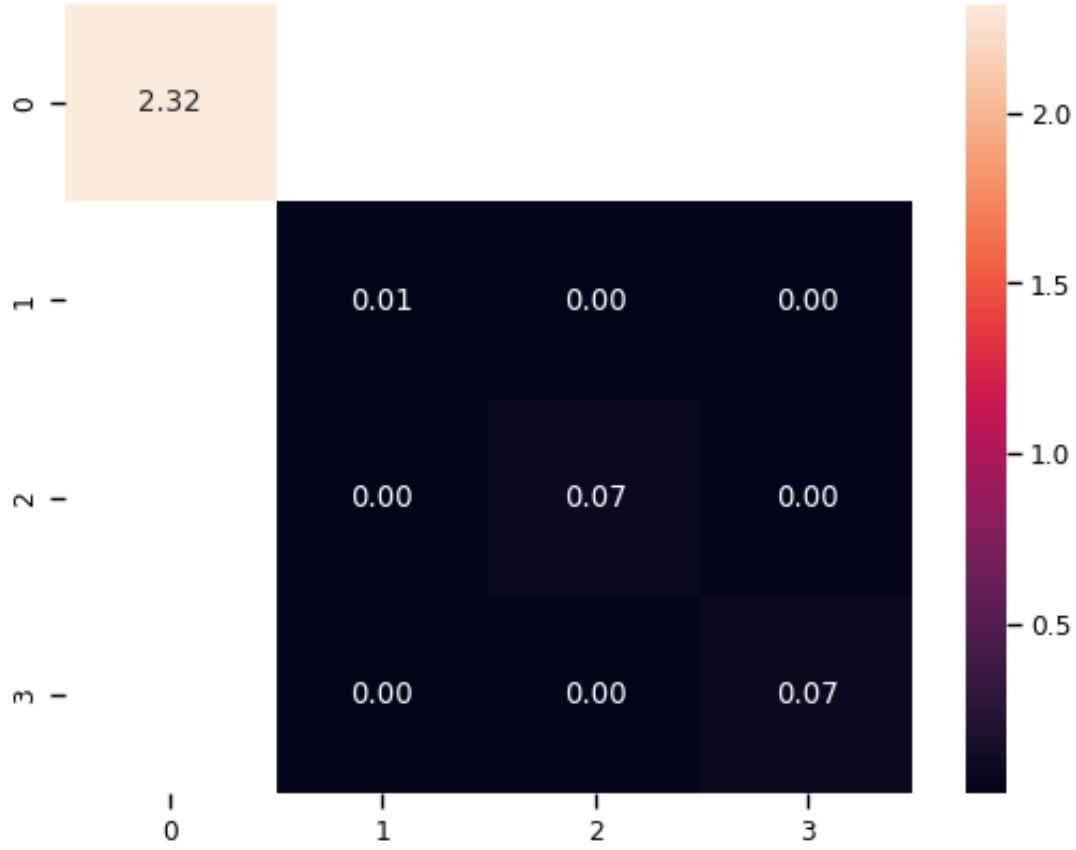
In [14]:
```
sigma_wls2
```

Out[14]:
```
12.268130156533738
```

In [15]:
```
sns.heatmap(np.sqrt(var_beta_wls), annot=True, fmt=".2f")
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/13725621
71.py:1: RuntimeWarning: invalid value encountered in sqrt
  sns.heatmap(np.sqrt(var_beta_wls), annot=True, fmt=".2f")
```

Out[15]:
```
<AxesSubplot: >
```



In [16]:
```
np.sqrt(np.diag(var_beta_wls))
```

Out[16]:
```
array([2.3182772 , 0.00538119, 0.06760636, 0.06760636])
```

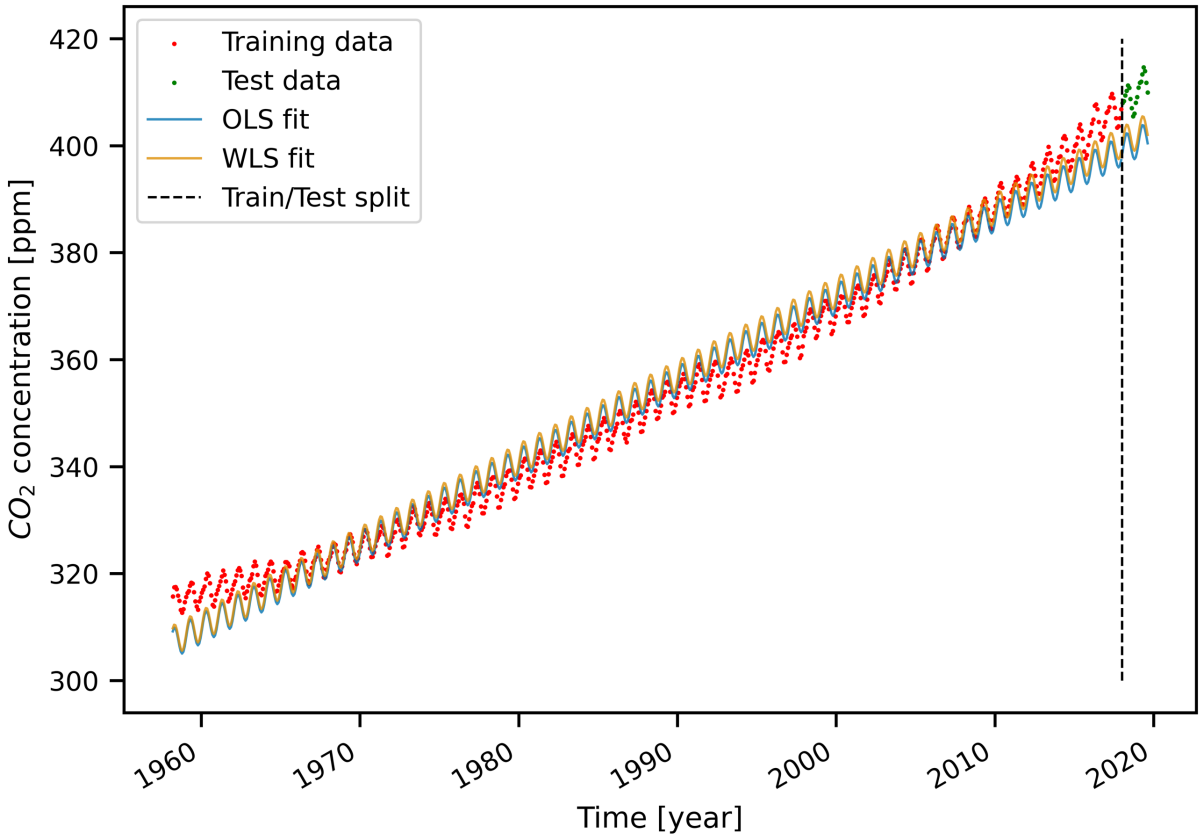
In [17]:
```
beta_ols - beta_wls
```

Out[17]:
```
array([-0.54250225, -0.00146511,  0.02199886, -0.04150264])
```

Model comparison plot

In [18]:
```
y_hat_ols = X_train @ beta_ols
y_hat_wls = X_train @ beta_wls

y_test_hat_ols = X_test @ beta_ols
y_test_hat_wls = X_test @ beta_wls

# Plot the train set and the fitted curves
fig, ax = plt.subplots(dpi=800)

# Set the style
sns.set_context("paper", rc={"lines.linewidth": 0.8})
sns.set_palette("colorblind")

ax.scatter(train["time"], train["co2"], label="Training data", s=0.5, c="
ax.scatter(test["time"], test["co2"], label="Test data", s=0.5, c="g")

ax.plot(train["time"], y_hat_ols, label="OLS fit", alpha=0.8, c="C0")
ax.plot(test["time"], y_test_hat_ols, alpha=0.8, c="C0")

ax.plot(train["time"], y_hat_wls, label="WLS fit", alpha=0.8, c="C1")
ax.plot(test["time"], y_test_hat_wls, alpha=0.8, c="C1")

ax.vlines(2018, 300, 420, linestyles="dashed", label="Train/Test split",
ax.legend()


ax.set_xlabel("Time [year]")
ax.set_ylabel("$CO_2$ concentration [ppm]")

fig.autofmt_xdate()
fig.show()
```

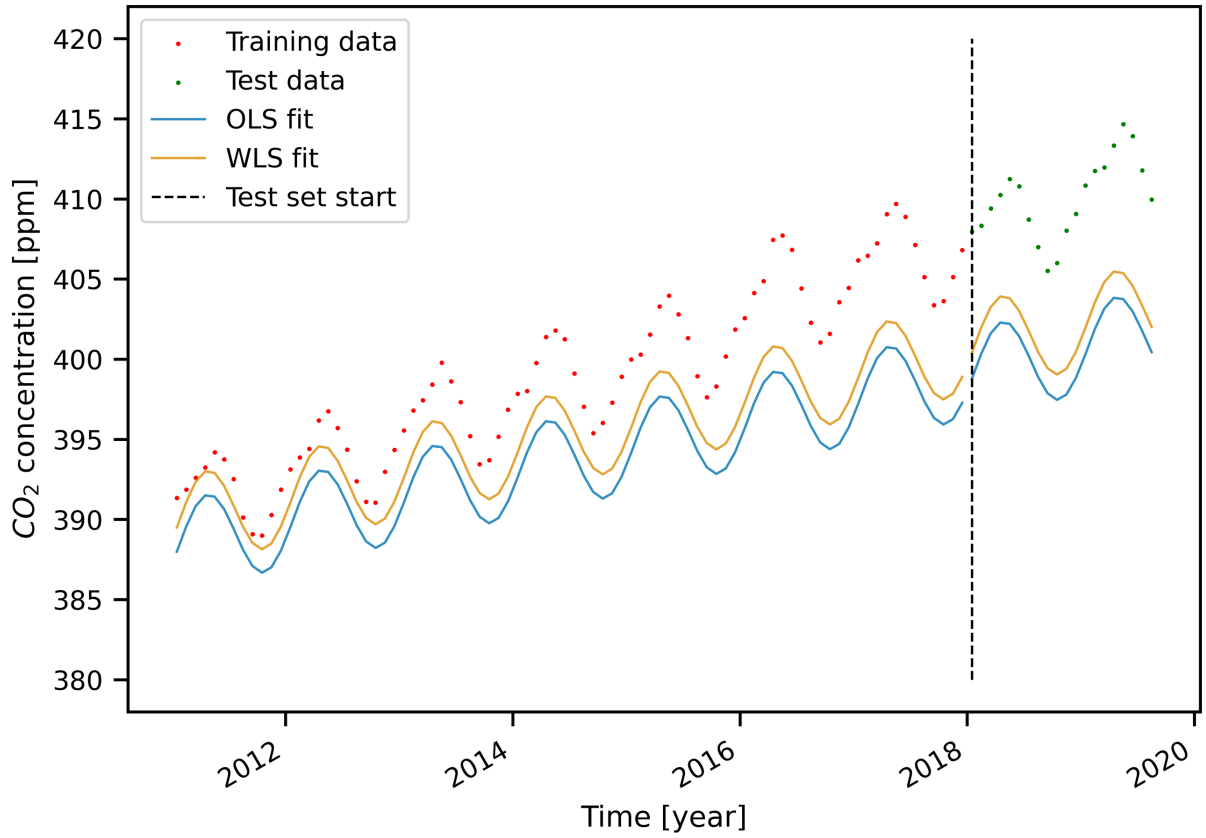
```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/37388481
00.py:31: UserWarning: Matplotlib is currently using module://matplotlib_
inline.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  fig.show()
```



In [19]:

```
y_hat_ols = X_train @ beta_ols
y_hat_wls = X_train @ beta_wls

# Plot the train set and the fitted curves
fig, ax = plt.subplots(dpi=800)

# Set the style
sns.set_context("paper", rc={"lines.linewidth": 0.8})
sns.set_palette("colorblind")

idx_2010 = train["year"] > 2010

ax.scatter(train["time"][idx_2010], train["co2"][idx_2010], label="Traini
ax.scatter(test["time"], test["co2"], label="Test data", s=0.5, c="g")

ax.plot(train["time"][idx_2010], y_hat_ols[idx_2010], label="OLS fit", al
ax.plot(test["time"], y_test_hat_ols, alpha=0.8, c="C0")

ax.plot(train["time"][idx_2010], y_hat_wls[idx_2010], label="WLS fit", al
ax.plot(test["time"], y_test_hat_wls, alpha=0.8, c="C1")


ax.vlines(test["time"].min(), 380, 420, linestyles="dashed", label="Test

ax.legend()

ax.set_xlabel("Time [year]")
ax.set_ylabel("$CO_2$ concentration [ppm]")

fig.autofmt_xdate()
fig.show()
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/3975576035.py:31: UserWarning: Matplotlib is currently using module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```

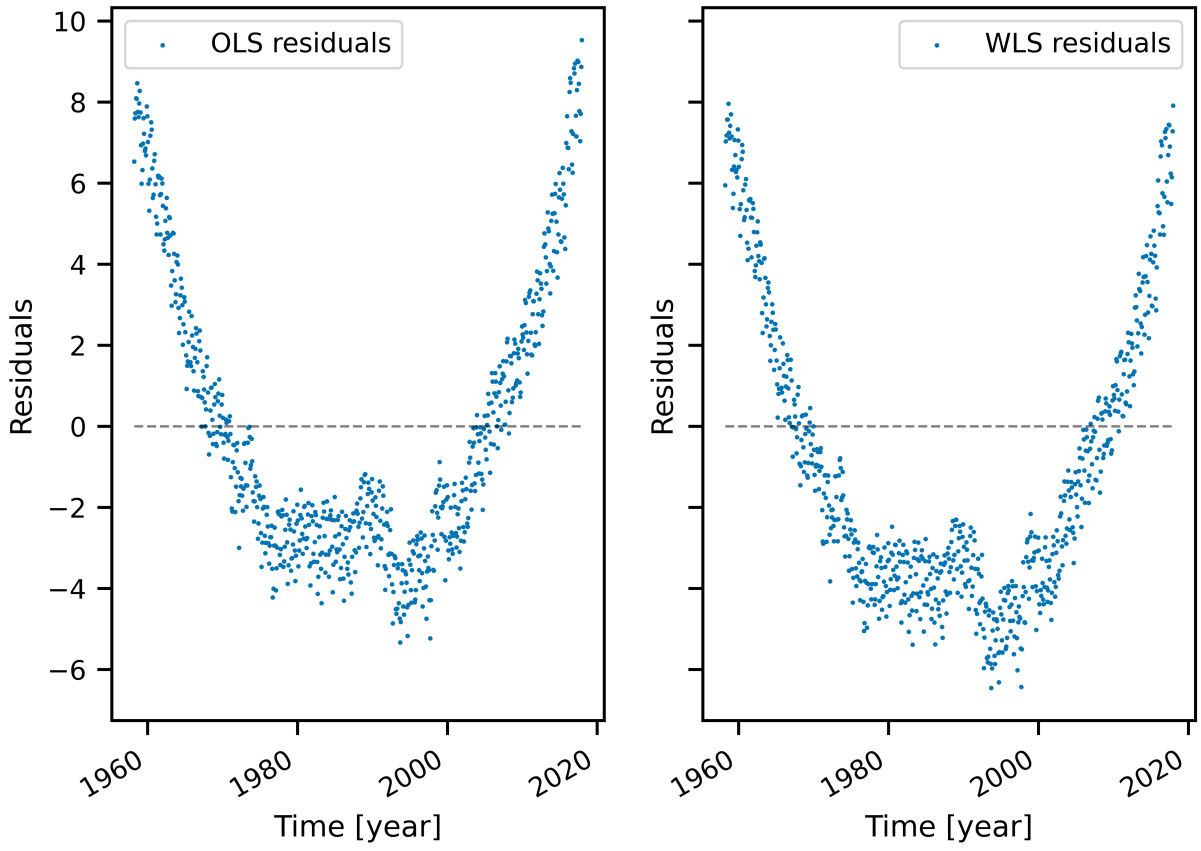


Model comparison

In [20]:
```
residuals_ols = train["co2"] - y_hat_ols
residuals_wls = train["co2"] - y_hat_wls
```

In [21]:
```
# Scatter plot of the residuals for the OLS and WLS fits
fig, ax = plt.subplots(ncols=2, sharey=True, dpi=800)

# Set the style
sns.set_context("paper", rc={"lines.linewidth": 0.8})
sns.set_palette("colorblind")

ax[0].scatter(train["time"], residuals_ols, label="OLS residuals", s=0.5,
ax[1].scatter(train["time"], residuals_wls, label="WLS residuals", s=0.5)

ax[0].hlines(0, train["time"].min(), train["time"].max(), linestyle="--",
ax[1].hlines(0, train["time"].min(), train["time"].max(), linestyle="--",

ax[0].legend()
ax[1].legend()

ax[0].set_xlabel("Time [year]")
ax[1].set_xlabel("Time [year]")

ax[0].set_ylabel("Residuals")
ax[1].set_ylabel("Residuals")

fig.autofmt_xdate()
fig.show()
```

```
/var/folders/p_/6wwjz4v11fs26vb5j8zwskgm0000gn/T/ipykernel_16591/10314583
36.py:24: UserWarning: Matplotlib is currently using module://matplotlib_
inline.backend_inline, which is a non-GUI backend, so cannot show the fig
ure.
  fig.show()
```



In [ ]:

Question 1.3

In [22]:
```
plt.rc("axes", axisbelow=True)
plt.rcParams["figure.dpi"] = 400
plt.rcParams["figure.figsize"] = (9, 4)
sns.set_palette("colorblind")
```

Question 1.3.2

Filter the data with the chosen model.

In [23]:
```
# Load data

year, month, time, co2 = df.values.T

year = year.astype(int)
month = month.astype(int)

# Index data by indices. 12 months per year
# NOTE: There is a constant time delta between each data point,
#  so indexing directly by indices is fine.
p = 12

# Split into test and train
test_idx = year >= 2018
time_test, co2_test = time[test_idx], co2[test_idx]
time_train, co2_train = time[~test_idx], co2[~test_idx]
y_train = co2_train[:, None]

train_n = len(time_train)
test_n = len(time_test)

param_n = 4
```

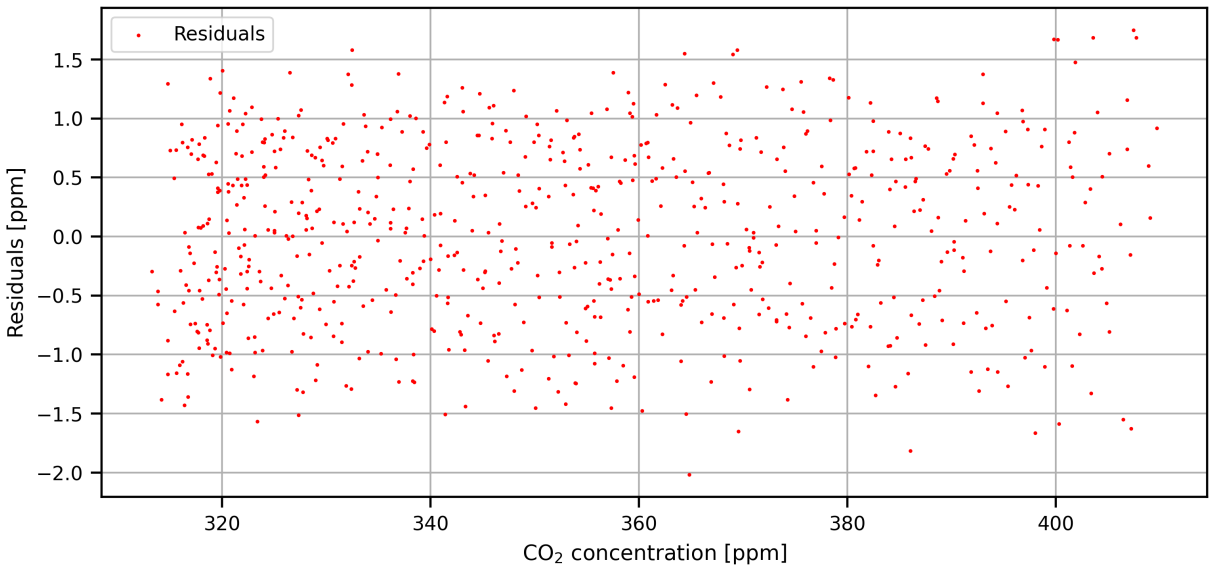
In [24]:
```
def llt_pred_interval(y_pred, t, F, f, var, alpha=0.05):
    return y_pred + np.array([1, -1]) * stats.t.ppf(
        1 - alpha / 2, t - param_n
    ) * np.sqrt(var) * np.sqrt(1 + f.T @ np.linalg.inv(F) @ f)


def f(t):
    return np.array(
        [
            [
                1,
                t,
                np.sin(2 * np.pi / p * t),
                np.cos(2 * np.pi / p * t),
            ]
        ]
    ).T


def llt_predict(start=10, lamb=0.9):
    assert start > 0, "Start must be greater than 0"
```

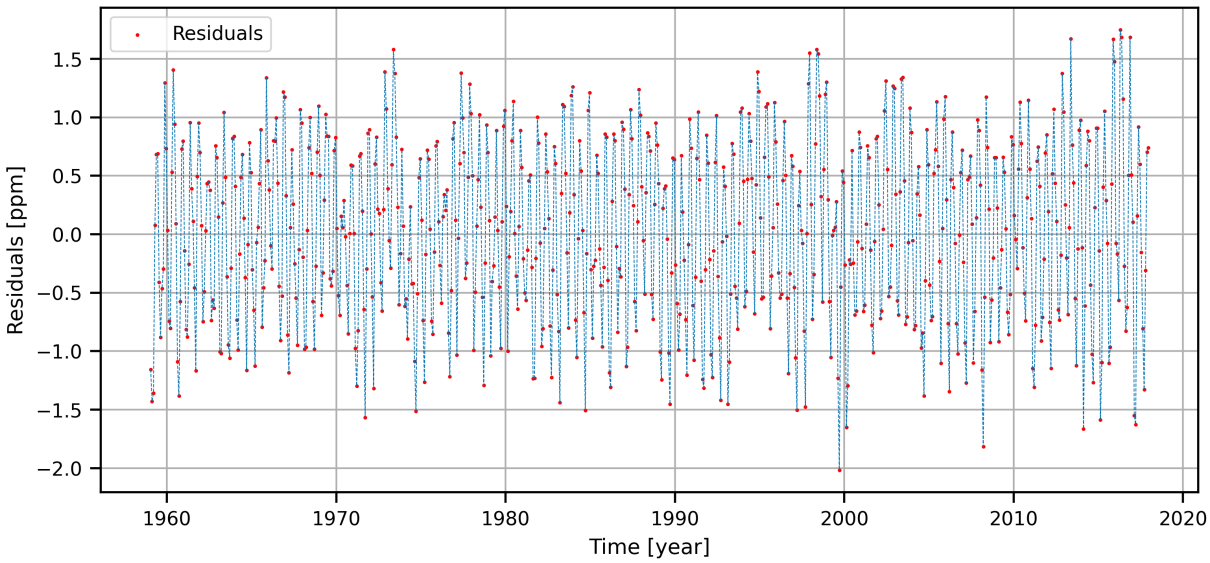
```
    # Createnp.linalg.inverse transition matrix
    L_inv = np.linalg.inv(
        np.array(
            [
                [1, 0, 0, 0],
                [1, 1, 0, 0],
                [0, 0, np.cos(2 * np.pi / p), np.sin(2 * np.pi / p)],
                [0, 0, -np.sin(2 * np.pi / p), np.cos(2 * np.pi / p)],
            ]
        )
    )

    # Create design matrix for t'th time step
    f_0 = f(0)

    F = np.zeros((param_n, param_n))
    h = np.zeros((param_n, 1))

    # Store predictions and variances
    y_pred_train = []
    y_pred_train_interval = []
    y_pred_train_mean = []
    y_pred_train_var = []
    y_pred_train_var_sum = 0
    thetas = []

    # Go through each data point and make one-step predictions
    for i in range(0, train_n):
        # If burn-in period is over, make predictions
        if i >= start:
            F_inv = np.linalg.inv(F)

            # Calculate parameters for previous time step
            theta = F_inv @ h

            # Make one step prediction using parameters
            #  from previous data to predict current data.
            y_pred = f(1).T @ theta

            # Calculate noise variance from next prediction
            y_pred_train_var_sum += (
                (y_train[i, 0] - y_pred[0]) ** 2 / (1 + f(1).T @ F_inv @
            )[0, 0]
            y_pred_train_var.append(y_pred_train_var_sum / (i - start + 1

            # Store predictions
            y_pred_train.append(y_pred[0, 0])
            y_pred_train_mean.append(theta[0, 0])
            thetas.append(theta)

            # Store prediction interval
            y_pred_train_interval.append(
                llt_pred_interval(y_pred, i - 1, F, f(1), y_pred_train_va
            )

        # Update F and h
        F = F + lamb**i * f(-i) @ f(-i).T
        h = lamb * L_inv @ h + f_0 @ y_train[i : i + 1]

    # Calculate final parameters
```

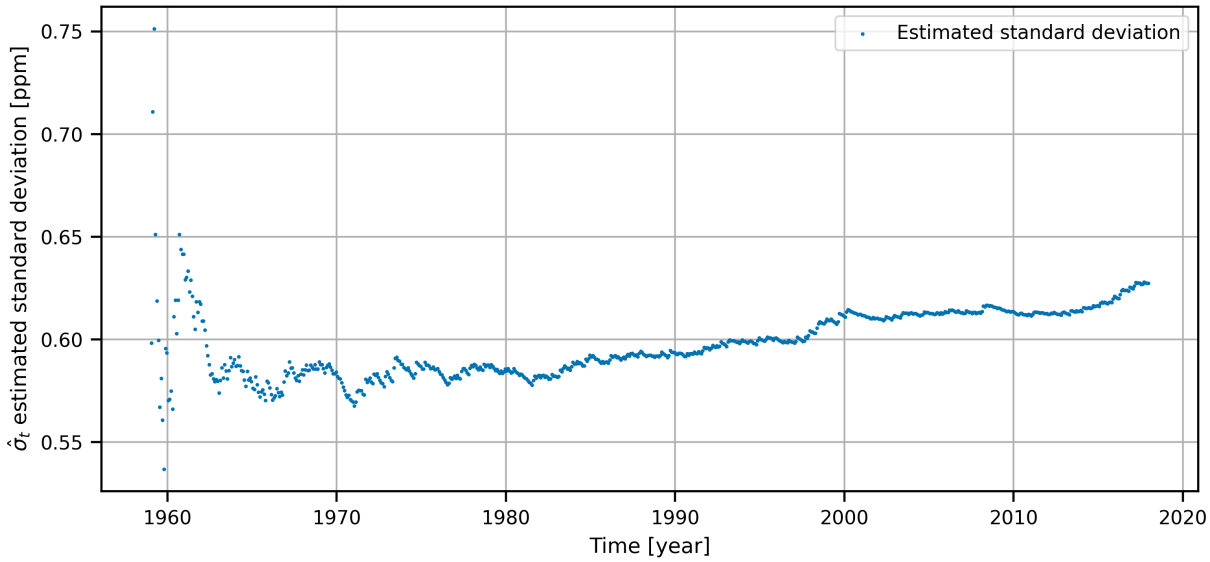
```
    theta = np.linalg.inv(F) @ h
    thetas.append(theta)

    # Make predictions for test data
    y_pred_test = np.array([f(t + 1).squeeze() for t in range(test_n)]) @

    # Make intervals for test data
    y_pred_test_interval = np.array(
        [
            llt_pred_interval(
                y_pred_test[t], train_n, F, f(t + 1), y_pred_train_var[-1
            )
            for t in range(test_n)
        ]
    ).squeeze()

    # Return a big mess
    return (
        np.array(y_pred_train),
        np.array(y_pred_train_interval).squeeze(),
        np.array(y_pred_train_mean),
        np.array(y_pred_train_var),
        y_pred_test,
        y_pred_test_interval,
    )


# Amount of data to skip predictions for
burn_in_1 = 10

# Retrieve predictions
(
    llt_y_pred_train,
    llt_y_pred_train_interval,
    llt_y_pred_train_mean,
    llt_y_pred_train_var,
    llt_y_pred_test,
    llt_y_pred_test_interval,
) = llt_predict(start=burn_in_1, lamb=0.9)
```

In [25]:

```
llt_residuals = y_train[burn_in_1:, 0] - llt_y_pred_train

# Plot residuals as function of y
plt.scatter(y_train[burn_in_1:, 0], llt_residuals, s=0.8, c='r', label='R

plt.xlabel("CO$_2$ concentration [ppm]")
plt.ylabel("Residuals [ppm]")

plt.legend()
plt.grid()

plt.show()


# Plot residuals as function of time
plt.plot(time_train[burn_in_1:], llt_residuals, '--', linewidth=0.4)
plt.scatter(time_train[burn_in_1:], llt_residuals, c='r', s=0.8, label='R

plt.xlabel("Time [year]")
plt.ylabel("Residuals [ppm]")

plt.legend()
plt.grid()

plt.show()


# Plot variance
plt.scatter(time_train[burn_in_1:], np.sqrt(llt_y_pred_train_var), s=0.8,

plt.xlabel("Time [year]")
plt.ylabel("$\hat \sigma_t$ estimated standard deviation [ppm]")

plt.legend()
plt.grid()

plt.show()

print(np.sqrt(llt_y_pred_train_var[-1]))
```







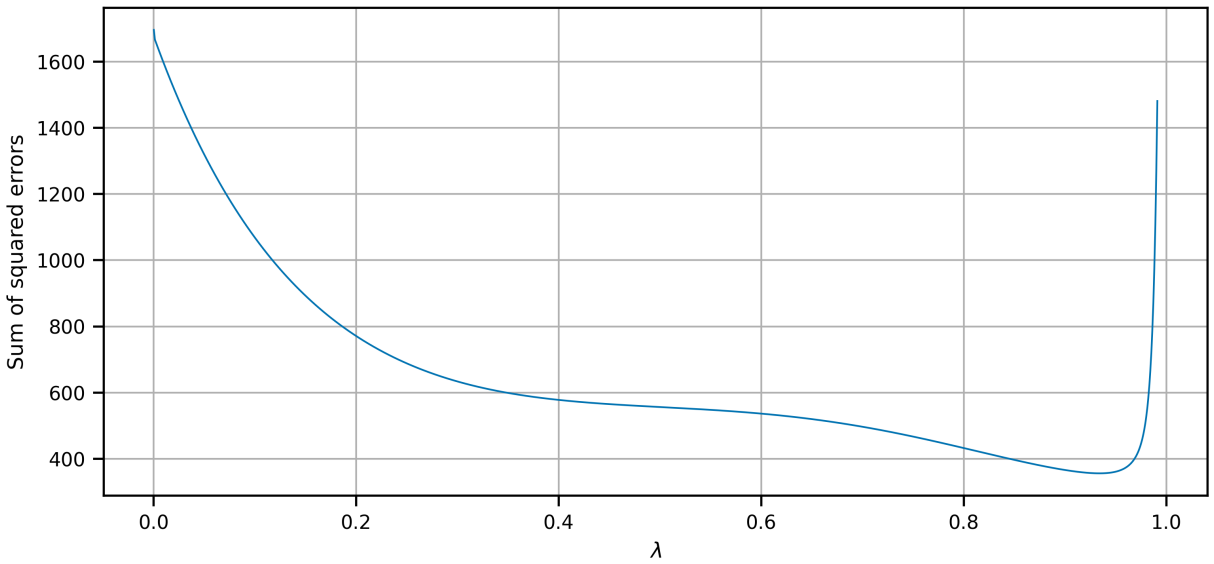

```
0.6273878545097442
```

1.4

In [26]:

```
# Set ranges of points to skip and lambdas to test
burn_in_2 = 100
lambs = np.linspace(0+2e-4, 1-9e-3, 1000)

# Predict and calculate squared error for each lambda
sqerrors = [((y_train[burn_in_2:, 0] - llt_predict(start=burn_in_2, lamb=

# Plot squared error as a function of lambda
plt.plot(lambs, sqerrors)

plt.grid()
plt.xlabel('$\lambda$')
plt.ylabel('Sum of squared errors')

plt.show()

print(lambs[800+np.argmin(sqerrors[800:])])
```

```
100%|██████████| 1000/1000 [01:00<00:00, 16.53it/s]
```

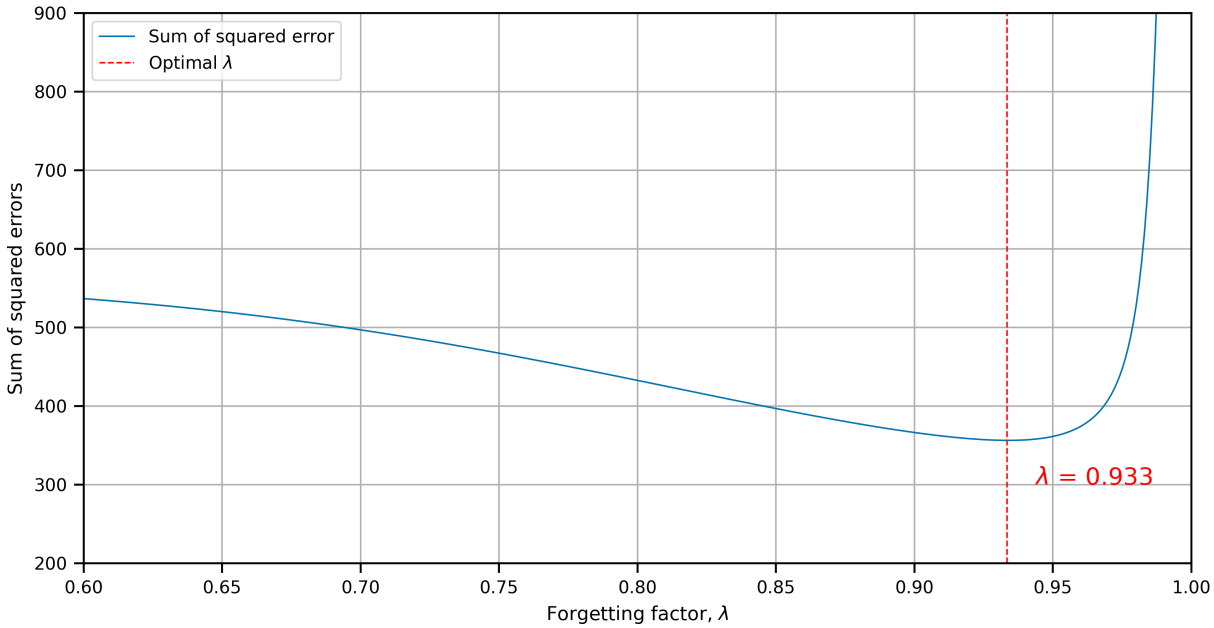



```
0.933476076076076
```

In [27]:

```
optimal_lambda = lambs[800 + np.argmin(sqerrors[800:])]
print(optimal_lambda)
print(sqerrors[800 + np.argmin(sqerrors[800:])])


fig, ax = plt.subplots(figsize=(10, 5))
ax.grid()

# Plot squared error as a function of lambda
ax.plot(lambs, sqerrors, label="Sum of squared error")
ax.vlines(
    optimal_lambda, 200, 900, color="r", linestyles="dashed", label="Opti
)

plt.text(
    optimal_lambda + 0.01,
    300,
    f"$\lambda$ = {optimal_lambda:.3f}",
    color="r",
    fontsize=12,
)

ax.set_xlim(0.6, 1)
ax.set_ylim(200, 900)

ax.set_xlabel("Forgetting factor, $\lambda$")
ax.set_ylabel("Sum of squared errors")
ax.legend()

plt.show()
```

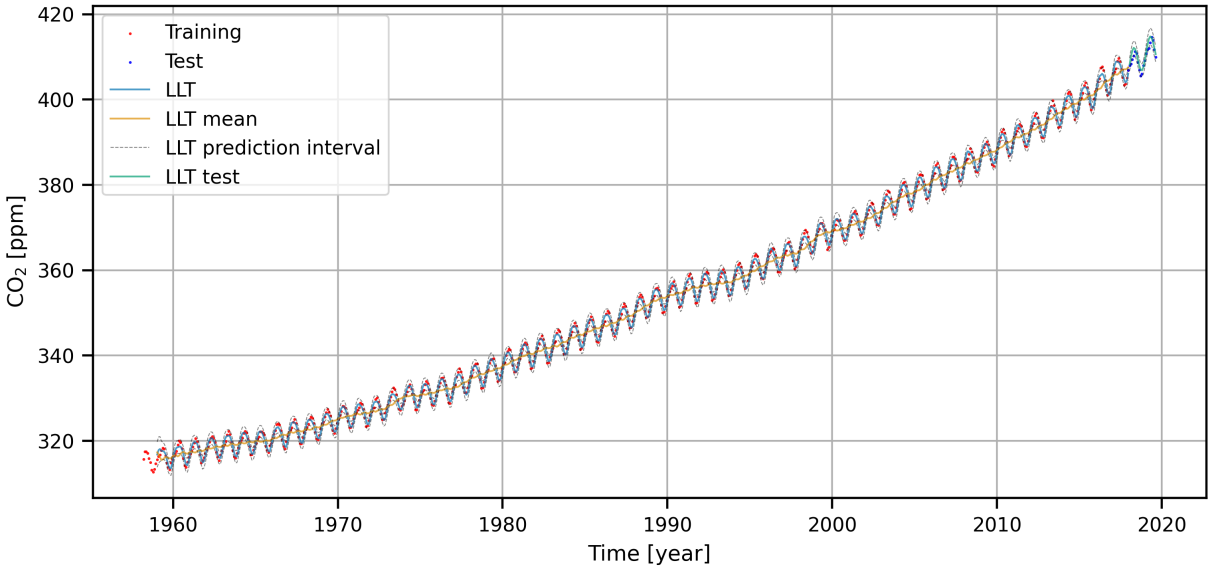
```
0.933476076076076
356.2038625197131
```



1.1

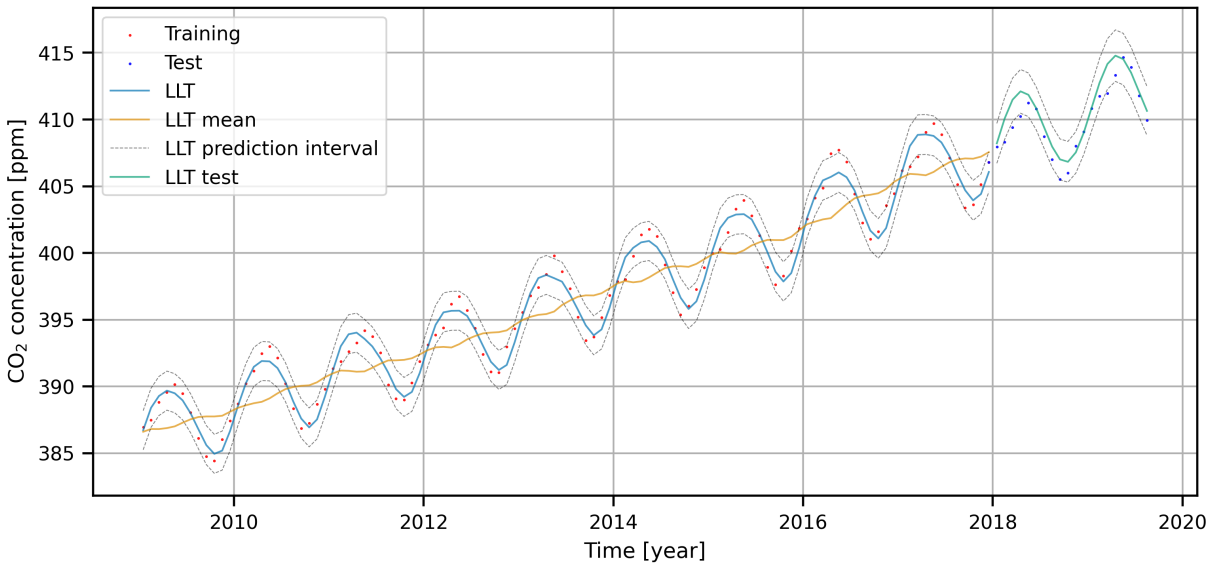
In [28]:

```
# Set plot options
plot_model_params = {"alpha": 0.7, "linestyle": '-', "linewidth": 0.8}
plot_interval_params = {"alpha": 0.5, "linestyle": '--', "linewidth": 0.4
plot_data_params = {"alpha": 0.9, "s": 0.2}


# Plot data
plt.scatter(time_train, co2_train, label="Training", c="r", **plot_data_p
plt.scatter(np.concatenate([time_train[-1, None], time_test]), np.concate

# Plot model predictions
plt.plot(time_train[burn_in_1:], llt_y_pred_train, label="LLT", **plot_mo
plt.plot(time_train[burn_in_1:], llt_y_pred_train_mean, label="LLT mean",
plt.plot(time_train[burn_in_1:], llt_y_pred_train_interval[:, 0], label="
plt.plot(time_train[burn_in_1:], llt_y_pred_train_interval[:, 1], c='k',

plt.plot(time_test, llt_y_pred_test, label="LLT test", **plot_model_param
plt.plot(time_test, llt_y_pred_test_interval[:, 0], c='k', **plot_interva
plt.plot(time_test, llt_y_pred_test_interval[:, 1], c='k', **plot_interva



# Plot
plt.xlabel("Time [year]")
plt.ylabel("CO$_2$ [ppm]")

plt.legend()
plt.grid()

plt.show()
```



In [29]:

```
zoom_idx = np.argwhere(year[~test_idx] >= 2009).squeeze()

# Plot data
plt.scatter(time_train[zoom_idx], co2_train[zoom_idx], label="Training",
plt.scatter(np.concatenate([time_train[-1, None], time_test]), np.concate

# Plot model predictions
plt.plot(time_train[zoom_idx], llt_y_pred_train[zoom_idx-burn_in_1], labe
plt.plot(time_train[zoom_idx], llt_y_pred_train_mean[zoom_idx-burn_in_1],
plt.plot(time_train[zoom_idx], llt_y_pred_train_interval[zoom_idx-burn_in
plt.plot(time_train[zoom_idx], llt_y_pred_train_interval[zoom_idx-burn_in

plt.plot(time_test, llt_y_pred_test, label="LLT test", **plot_model_param
plt.plot(time_test, llt_y_pred_test_interval[:, 0], c='k', **plot_interva
plt.plot(time_test, llt_y_pred_test_interval[:, 1], c='k', **plot_interva


# Plot
plt.xlabel("Time [year]")
plt.ylabel("CO$_2$ concentration [ppm]")

plt.legend()
plt.grid()

plt.show()
```



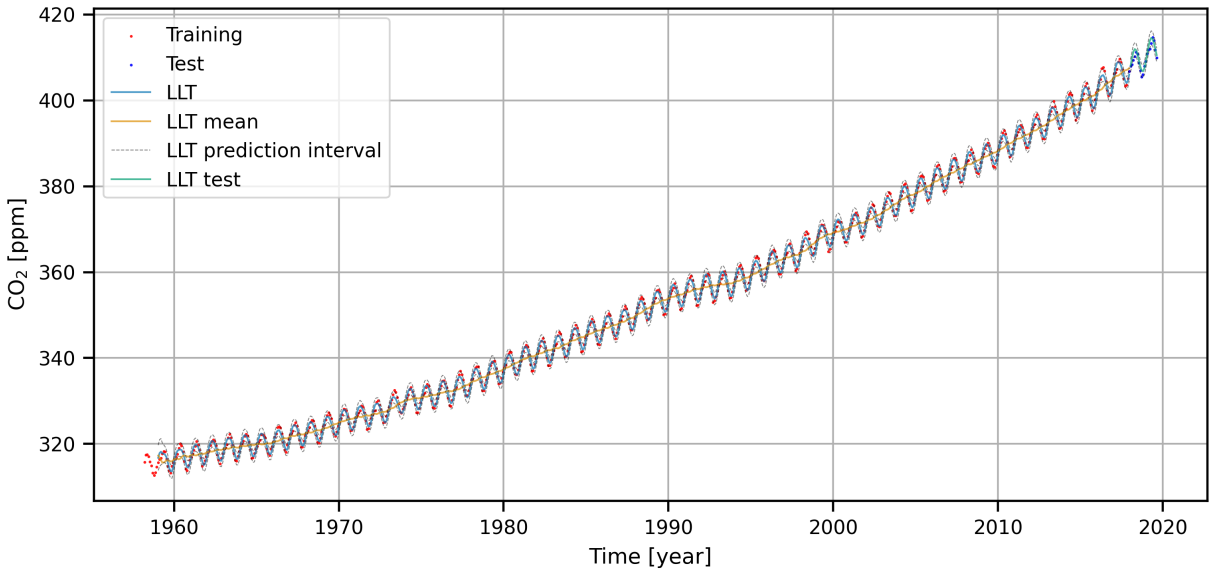
In [30]:
```
# Print table of future predictions
y_pred_test_idx = np.array([1, 2, 6, 12, 20])-1

for t, true, pred, lower, upper in zip(y_pred_test_idx+1, co2_test[y_pred
    print(f"{t} & {true:.2f} & {pred:.2f} & {lower:.2f} & {upper:.2f}\\\\
```

```
1 & 407.96 & 408.21 & 406.72 & 409.71\\
2 & 408.32 & 410.08 & 408.51 & 411.64\\
6 & 410.79 & 410.83 & 409.22 & 412.44\\
12 & 409.07 & 409.00 & 407.37 & 410.62\\
20 & 409.95 & 410.64 & 408.81 & 412.48\\
```

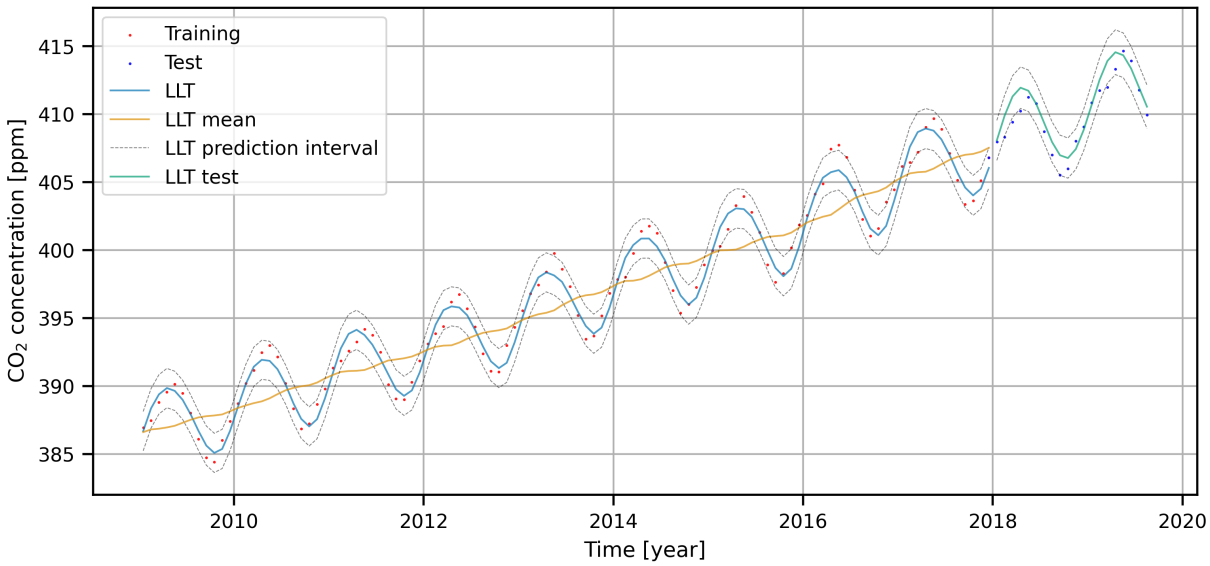
Question 1.4.2

In [31]:

```
BURN_IN = 10
optimal_llt_y_pred_train, optimal_llt_y_pred_train_interval, optimal_llt_

# Set plot options
plot_model_params = {"alpha": 0.7, "linestyle": '-', "linewidth": 0.8}
plot_interval_params = {"alpha": 0.5, "linestyle": '--', "linewidth": 0.4
plot_data_params = {"alpha": 0.9, "s": 0.2}


# Plot data
plt.scatter(time_train, co2_train, label="Training", c="r", **plot_data_p
plt.scatter(np.concatenate([time_train[-1, None], time_test]), np.concate

# Plot model predictions
plt.plot(time_train[BURN_IN:], optimal_llt_y_pred_train, label="LLT", **p
plt.plot(time_train[BURN_IN:], optimal_llt_y_pred_train_mean, label="LLT
plt.plot(time_train[BURN_IN:], optimal_llt_y_pred_train_interval[:, 0], l
plt.plot(time_train[BURN_IN:], optimal_llt_y_pred_train_interval[:, 1], c

plt.plot(time_test, optimal_llt_y_pred_test, label="LLT test", **plot_mod
plt.plot(time_test, optimal_llt_y_pred_test_interval[:, 0], c='k', **plot
plt.plot(time_test, optimal_llt_y_pred_test_interval[:, 1], c='k', **plot


# Plot
plt.xlabel("Time [year]")
plt.ylabel("CO$_2$ [ppm]")

plt.legend()
plt.grid()

plt.show()
```



In [32]:
```
zoom_idx = np.argwhere(year[~test_idx] >= 2009).squeeze()

# Plot data
plt.scatter(time_train[zoom_idx], co2_train[zoom_idx], label="Training",
plt.scatter(np.concatenate([time_train[-1, None], time_test]), np.concate

# Plot model predictions
plt.plot(time_train[zoom_idx], optimal_llt_y_pred_train[zoom_idx-BURN_IN]
plt.plot(time_train[zoom_idx], optimal_llt_y_pred_train_mean[zoom_idx-BUR
plt.plot(time_train[zoom_idx], optimal_llt_y_pred_train_interval[zoom_idx
plt.plot(time_train[zoom_idx], optimal_llt_y_pred_train_interval[zoom_idx

plt.plot(time_test, optimal_llt_y_pred_test, label="LLT test", **plot_mod
plt.plot(time_test, optimal_llt_y_pred_test_interval[:, 0], c='k', **plot
plt.plot(time_test, optimal_llt_y_pred_test_interval[:, 1], c='k', **plot


# Plot
plt.xlabel("Time [year]")
plt.ylabel("CO$_2$ concentration [ppm]")

plt.legend()
plt.grid()

plt.show()
```



In [33]:
```
# Print table of future predictions
y_pred_test_idx = np.array([1, 2, 6, 12, 20])-1

for t, true, pred, lower, upper in zip(y_pred_test_idx+1, co2_test[y_pred
    print(f"{t} & {true:.2f} &{pred:.2f} & {lower:.2f} & {upper:.2f}\\\\"
```

```
1 & 407.96 &408.07 & 406.60 & 409.54\\
2 & 408.32 &409.90 & 408.40 & 411.41\\
6 & 410.79 &410.74 & 409.23 & 412.25\\
12 & 409.07 &408.84 & 407.32 & 410.36\\
20 & 409.95 &410.54 & 408.95 & 412.14\\
```