Co2 concentration in the atmosphere since 1958

Introduction

In this notebook, I'll use the data of Mauna Loa Observatory, Hawaii provided by the Scripps CO2 Program (https://scrippsco2.ucsd.edu/data/atmospheric_co2/primary_mlo_co2_record.html). The goal will be to explain the data using a model, and then try to predict the future concentration in atmosphere of co2 at this place until 2025.

Show hidden code

Load the data

Let's download the last version of the scripps program so that it's reusable for future work. But the data we'll use in this notebook is a static dataset corresponding to the latest data available when the notebook was created for reproducibility

In [2]:

```
!curl https://scrippsco2.ucsd.edu/assets/data/atmospheric/stations/in_situ_co2/monthly/monthly_in_situ_co2_mlo.csv --output co2_data.csv
!cp co2_data.csv co2_data_new.csv
!cp ../input/scripps-co2-program-06122021/monthly_in_situ_co2_mlo.csv co2_data.csv
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 76791  100 76791    0     0  65915      0  0:00:01  0:00:01 --:--:-- 65915
```

Let's check that the file has not been modified somehow by checking its hash

In [3]:

```
expected_hash = "645983b005053fb3ba442ad034966c7f23173975"
assert sp.getoutput("sha1sum ./co2_data.csv")[:len(expected_hash)] == expected_h
ash, "File has been somehow corrupted"
```

In [4]:

```
# Let's load the data !
data = pd.read_csv("./co2_data.csv", comment='"', skiprows=[55, 56])
# According to the website, an unavailable data point is represented by the value
 -99.99
data[data == -99.99] = np.NaN
data
```

Out[4]:

	Yr	Mn	Date	Date	CO2	seasonally	fit	seasonally	CO2
0	1958	1	21200	1958.0411	NaN	NaN	NaN	NaN	NaN
1	1958	2	21231	1958.1260	NaN	NaN	NaN	NaN	NaN
2	1958	3	21259	1958.2027	315.71	314.43	316.20	314.91	315.71
3	1958	4	21290	1958.2877	317.45	315.16	317.30	314.99	317.45
4	1958	5	21320	1958.3699	317.51	314.71	317.87	315.07	317.51
...	...	...	...	...	...	...	...	...	...
763	2021	8	44423	2021.6219	414.34	415.90	414.53	416.12	414.34
764	2021	9	44454	2021.7068	412.90	416.42	NaN	NaN	412.90
765	2021	10	44484	2021.7890	NaN	NaN	NaN	NaN	NaN
766	2021	11	44515	2021.8740	NaN	NaN	NaN	NaN	NaN
767	2021	12	44545	2021.9562	NaN	NaN	NaN	NaN	NaN

768 rows × 10 columns

Let's see if we can make some assumptions about the data and check them

In [5]:

```
# I expect that the date columns are not contradictory
if not (data.iloc[:, 0].values == np.floor(data.iloc[:, 3].values)).all(): warni
ngs.warn("Date columns are contradictory !")
# I expect that if one value is missing, then every other one the same date are mi
ssing
if not ((data.values == -99.99).sum(axis=-1) == 6).all(): warnings.warn("There a
re entries where some values are missing but not all of them")
```

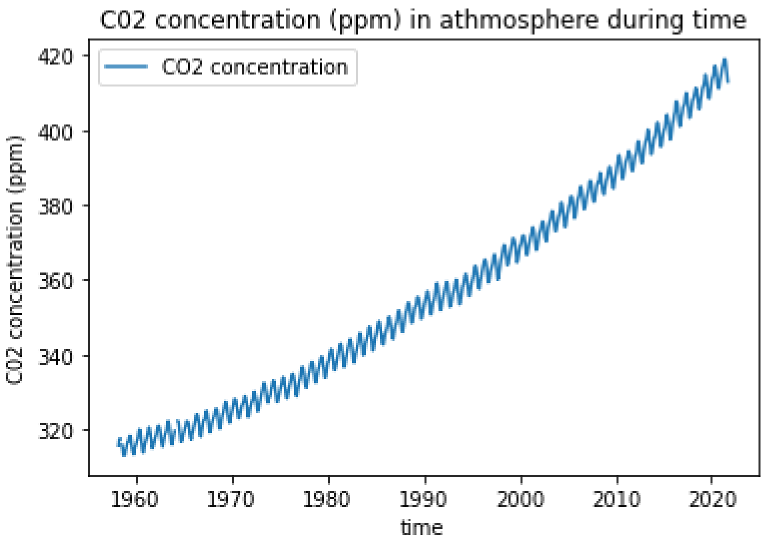
We'll consider only a subset of the variables of the dataset, namely the measured co2 concentration, and the date expressed in a continuous way. The other columns are either redundancies or calculated columns.

In [6]:

```
# Separate the data that interest us
mois = data.iloc[:, 1]
annees = data.iloc[:, 3]
co2 = data.iloc[:, 4]
```

In [7]:

```
plt.title("C02 concentration (ppm) in athmosphere during time")
plt.plot(annees, co2, label="CO2 concentration")
plt.legend()
plt.xlabel("time")
plt.ylabel("C02 concentration (ppm)")
plt.show()
```



Models

Now that the data is loaded and that we had a look at it, we are gonna fit models to it and try to express in the siplest way possible the data.

Notation

During the notebook I'll describe the models using the following notation:

- t denotes the date expressed in a continous way
- y_t is the concentration of CO2 in the atmosphere at time t
- β_i denotes the parameters of the model

Linear model

As it's better to start simple, i'll first only try to explain the data using a linear model:

$$y_t = \beta_0 + \beta_1 t$$

Show hidden code

In [9]:

```python
# Now let's use our function to fit our first model
res = make_reg(annees, co2)
res.summary()
```

Out[9]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.976
Model:	OLS	Adj. R-squared:	0.976
Method:	Least Squares	F-statistic:	3.064e+04
Date:	Thu, 30 Dec 2021	Prob (F-statistic):	0.00
Time:	10:57:42	Log-Likelihood:	-2229.5
No. Observations:	758	AIC:	4463.
Df Residuals:	756	BIC:	4472.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2821.1070	18.154	-155.397	0.000	-2856.746	-2785.468
Date	1.5967	0.009	175.048	0.000	1.579	1.615

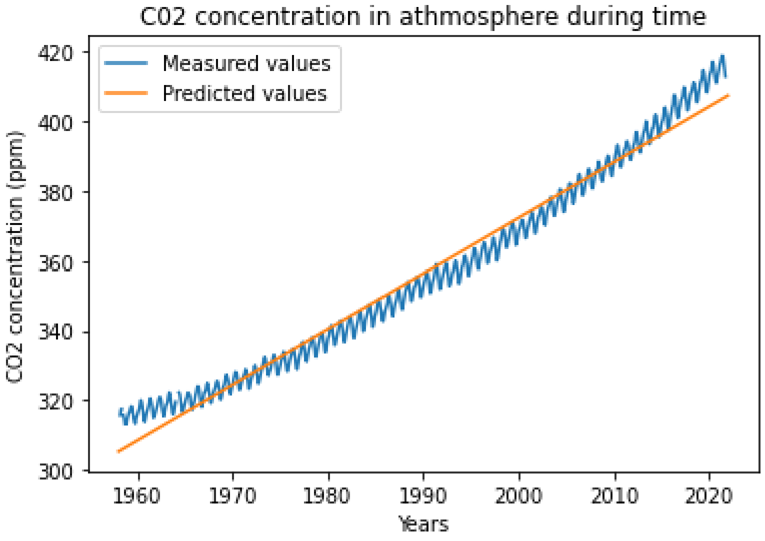
Omnibus:	40.528	Durbin-Watson:	0.075
Prob(Omnibus):	0.000	Jarque-Bera (JB):	44.311
Skew:	0.572	Prob(JB):	2.39e-10
Kurtosis:	2.691	Cond. No.	2.17e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.17e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

We see that the R^2 is very good (0.976). Furthermore, the p-values of the coefficient are very low which is a good sign. Now let's visually see the fitte curve

In [10]:

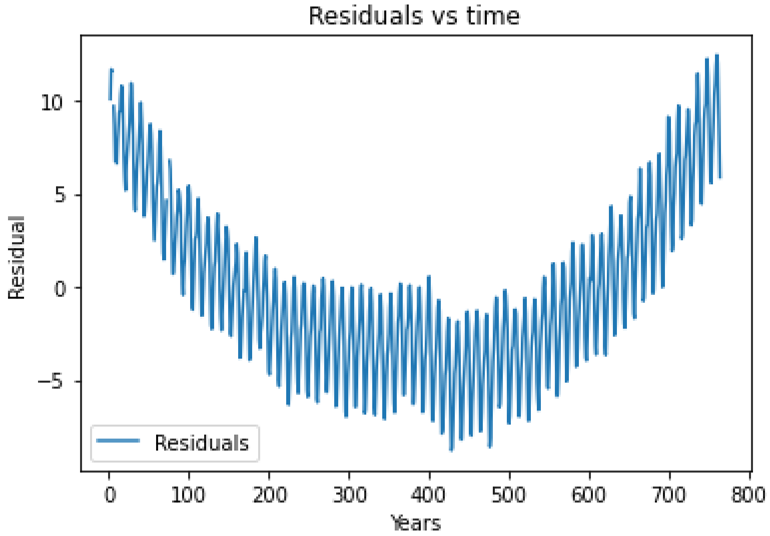
```
co2_pred = res.predict(sm.add_constant(annees))
plot_predictions_vs_real(annees, co2_pred, co2)
```

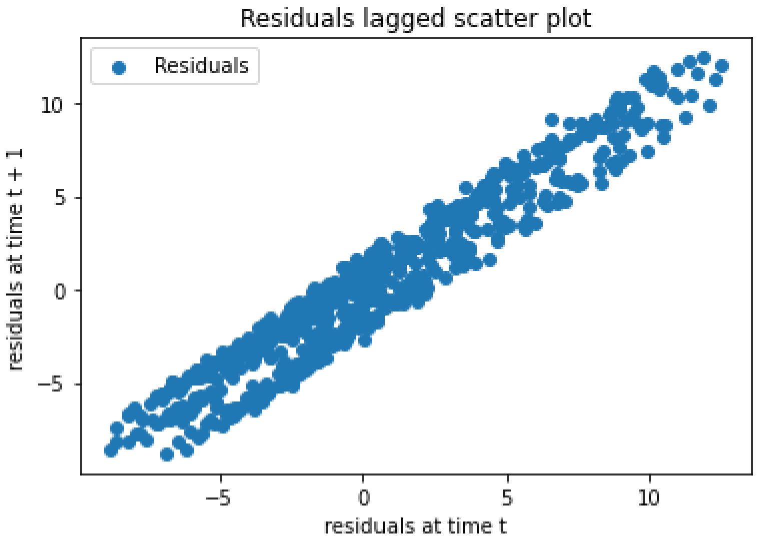


We clearly see that the linear explains well the global trend. However it's clear that the true data is not linear. Maybe plotting the residuals could be helpfull

In [11]:

```
plot_residuals(co2_pred, co2)
```





So we clearly see that there is a structure with the shape of a banana in the residuals over time. This is not a good sign as some assumptions of the model are completly false. Furthermore, the impression is confirmed by the lagged plot

Red flag: structure in the residuals

Exponential model

Visually, it looks like the curve is following a slite exponential-ish tendancy. Let's try to fit to $\log(y)$ instead of y itself:

$$\log(y_t) = \beta_0 + \beta_1 t$$

In [12]:

```
res = make_reg(annees, np.log(co2))
res.summary()
```

Out[12]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.984
Model:	OLS	Adj. R-squared:	0.984
Method:	Least Squares	F-statistic:	4.683e+04
Date:	Thu, 30 Dec 2021	Prob (F-statistic):	0.00
Time:	10:57:44	Log-Likelihood:	2390.2
No. Observations:	758	AIC:	-4776.
Df Residuals:	756	BIC:	-4767.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-2.9862	0.041	-72.942	0.000	-3.067	-2.906
Date	0.0045	2.06e-05	216.412	0.000	0.004	0.004

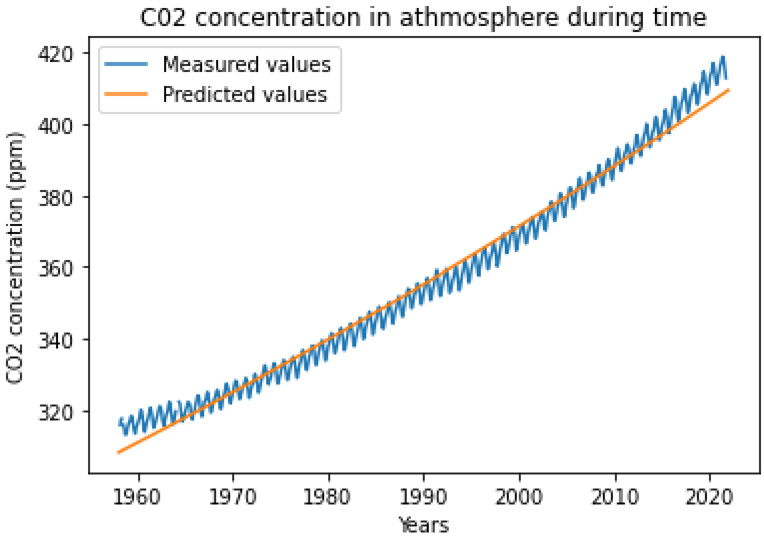
Omnibus:	25.319	Durbin-Watson:	0.116
Prob(Omnibus):	0.000	Jarque-Bera (JB):	26.524
Skew:	0.439	Prob(JB):	1.74e-06
Kurtosis:	2.737	Cond. No.	2.17e+05

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.17e+05. This might indicate that there are
strong multicollinearity or other numerical problems.

The R^2 of 0.984 is better than the previous one. The p-values indicate also a strong effect of the explainatory variables. I don't understand why the AIC is so low I thnk it's better to not take it into account for this model.

In [13]:

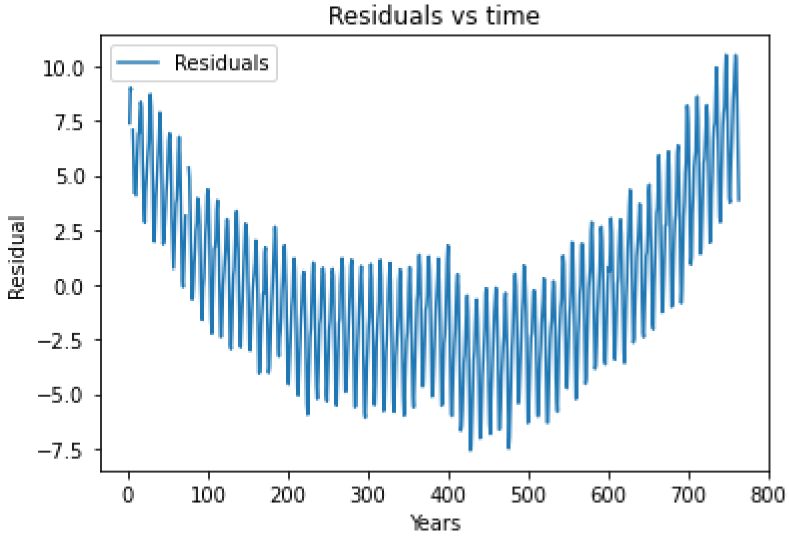
```
co2_pred = np.exp(res.predict(sm.add_constant(annees)))
plot_predictions_vs_real(annees, co2_pred, co2)
```

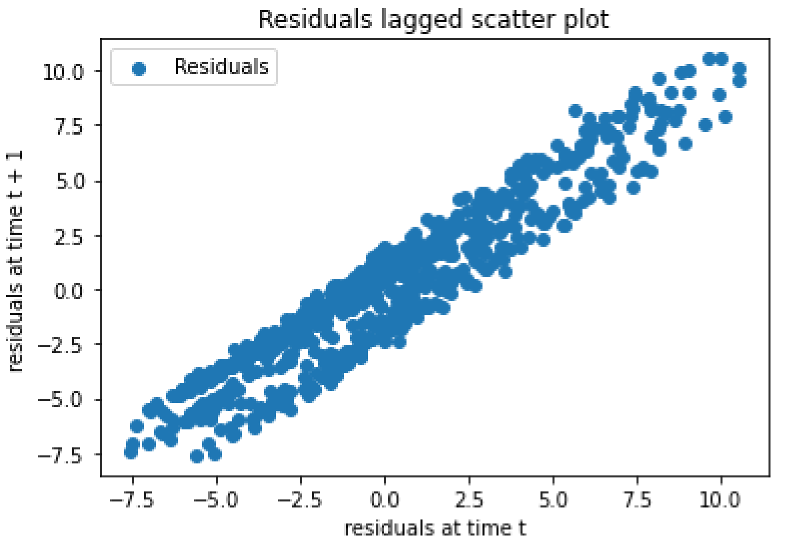


Visually, the log model looks better than the previous one, but we still see that it is not good enough. On the boundaries of the data, the error is pretty big, which is a problem if we want to extrapolate

In [14]:

```
plot_residuals(co2_pred, co2)
```





As in the linear model, we still have a banana structure in the residuals over time. Same for the lag plot .

Red flag: still structure in the data

Polynomial model

Ok so the growth isn't linear or exponential, so I'll now assume that it is quadratic. To see if it's the case, let's fit a polynomial model:

$$y_t = \beta_0 + \beta_1 t + \beta_2 t^2$$

In [15]:

```
res = make_reg(np.stack((annees, annees**2), axis=-1), co2)
res.summary()
```

Out[15]:

OLS Regression Results

Dep. Variable:	y	R-squared:	0.994
Model:	OLS	Adj. R-squared:	0.994
Method:	Least Squares	F-statistic:	6.582e+04
Date:	Thu, 30 Dec 2021	Prob (F-statistic):	0.00
Time:	10:57:45	Log-Likelihood:	-1683.6
No. Observations:	758	AIC:	3373.
Df Residuals:	755	BIC:	3387.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.007e+04	1072.372	46.691	0.000	4.8e+04	5.22e+04
x1	-51.5620	1.078	-47.841	0.000	-53.678	-49.446
x2	0.0134	0.000	49.323	0.000	0.013	0.014

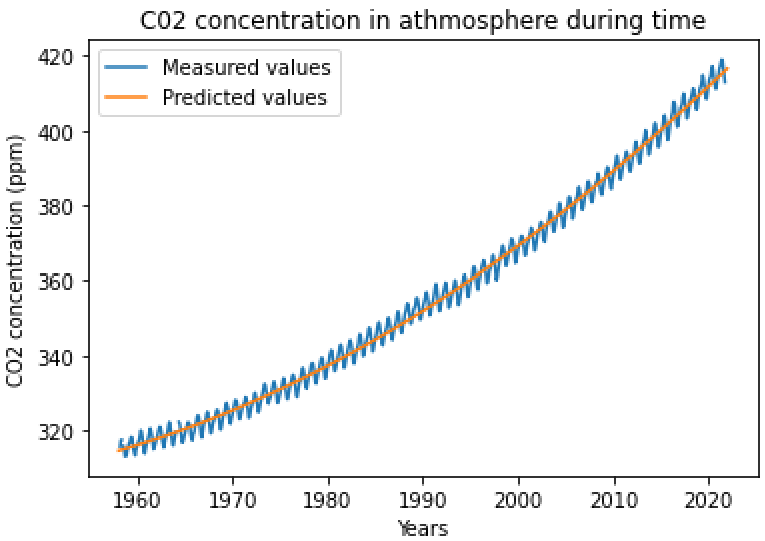
Omnibus:	110.510	Durbin-Watson:	0.317
Prob(Omnibus):	0.000	Jarque-Bera (JB):	28.328
Skew:	-0.102	Prob(JB):	7.06e-07
Kurtosis:	2.075	Cond. No.	5.23e+10

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.23e+10. This might indicate that there are
strong multicollinearity or other numerical problems.

Once again we have an even better R^2. All 3 p-values are very small meaning that all explainatory variable have a linear effect. The AIC of 3373 is the best one so far, which is a good indication that we are not yet overfitting

In [16]:

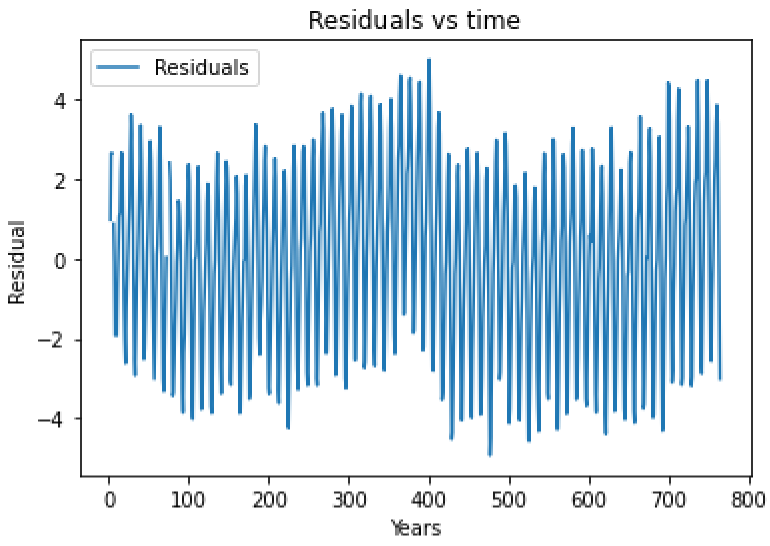
```
co2_pred = res.predict(sm.add_constant(np.stack((annees, annees**2), axis=-1)))
plot_predictions_vs_real(annees, co2_pred, co2)
```

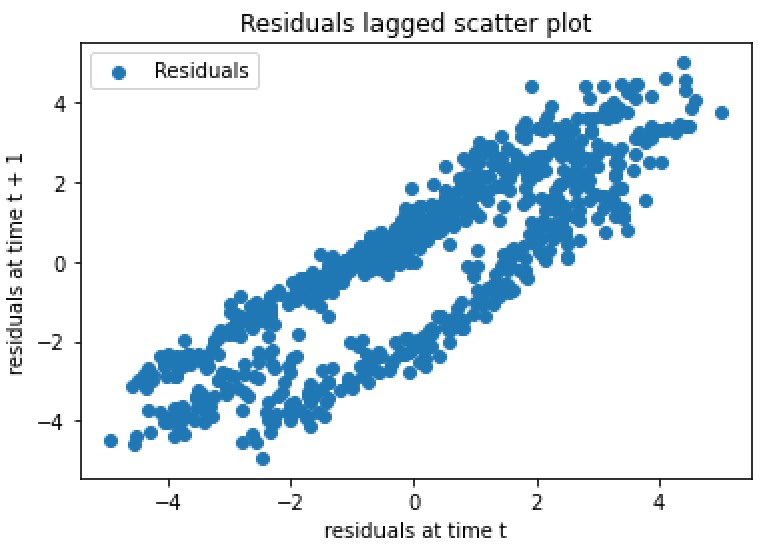


The visual plot looks much better, we have visually the best possible explaintation for the global tendancy

In [17]:

```
plot_residuals(co2_pred, co2)
```





No clear structure other than the seasonal component in the time vs residuals plot. The mean and std appear to be the same during time. This seasonnal component can explain the correlation observed in the lagged plot. We should probably try to describe it

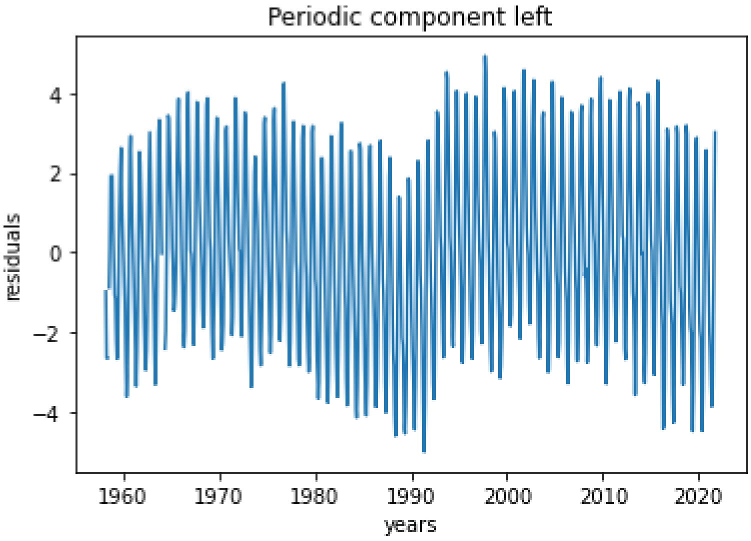
Seasonal component

Note of the me from the future: This section was quite a mess and end up on nothing usefull, i left it so that we can follow the path of toughts that led to the end but nothing usefull for the study until the next session

Fourier decomposition approach

In [18]:

```
periodic = co2_pred - co2
plt.title("Periodic component left")
plt.plot(annees, periodic)
plt.xlabel("years")
a = plt.ylabel("residuals")
```



I want to try to conduct a fourier transform decomposition, but there are some missing datapoints. fft expect a uniformly sampled signal. There is multiple things I can do from here:

1. Apply another harmonic decomposition algorithm that do not expect a uniformly sampled signal
2. Find an interpolation scheme to "fill in" the missing datapoints (that may introduce bias)
3. Perform my analysis on the biggest slice without missing datapoints

I've made some research to find a non uniform fourier transform algorithm but all I found was approximations, and algorithm where I have no clue how to use them. As I do not have a signal processing expert under the hand to ask questions to.
More details here (https://cims.nyu.edu/cmcl/nufft/nufft.html)

Now to choose between the second and last option I want to have a look at the location of the missing data points

In [19]:

```
na_mask = periodic.isna()
print("Date of missing values:", list(zip(np.floor(annees[na_mask]).astype(np.in
t).tolist(), mois[na_mask].tolist())))
```

```
Date of missing values: [(1958, 1), (1958, 2), (1958, 6), (1958, 1
0), (1964, 2), (1964, 3), (1964, 4), (2021, 10), (2021, 11), (2021,
12)]
```

I see that I have a slice between April 1964 and October 2021 with no missing values that could be used while dropping the rest. Another argument in favor of the slice option versus the interpolation one is that there are consectuive missing values, meaning that interpolation will be less precise.

So I've decided to go with the slice option.

In [20]:

```
print(np.arange(len(periodic))[na_mask])
```

```
[  0   1   5   9  73  74  75 765 766 767]
```

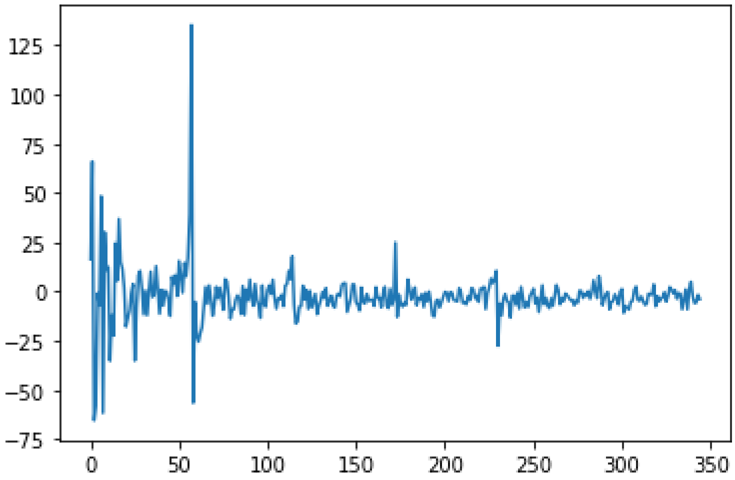
In [21]:

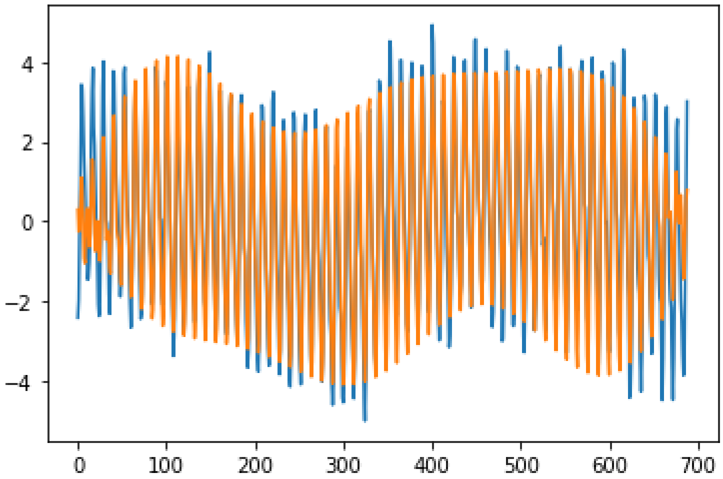
```
periodic_slice = periodic[76:765]
assert not periodic_slice.isna().any(), "Bad indices, there is still nans in the data"
periodic_slice = periodic_slice.values
print("Proportion of the data used to perform fft: {:.4f}".format(len(periodic_slice) / len(periodic)))
```

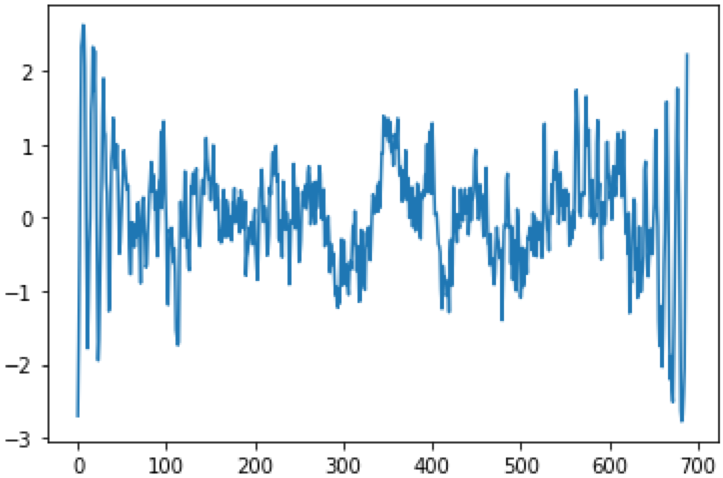
```
Proportion of the data used to perform fft: 0.8971
```

In [22]:

```
def fourierExtrapolation(x, t, n_harm=4):
    n = x.size
    x_freqdom = np.fft.rfft(x)
    plt.plot(x_freqdom)
    plt.show()
    indexes = np.argsort(np.absolute(x_freqdom))
    indexes_keep = indexes[-(1+n_harm*2):]
    x_approx = np.zeros_like(x_freqdom)
    x_approx[indexes_keep] = x_freqdom[indexes_keep]
    return np.fft.irfft(x_approx, t.max()+1)[t]

approx = fourierExtrapolation(periodic_slice, np.arange(len(periodic_slice)))
plt.plot(periodic_slice)
plt.plot(approx)
plt.show()
plt.plot(periodic_slice - approx)
plt.show()
```







Well, the result isn't very good, we see that there is a clear boundary effect that will hurt if we try to extrapolate.

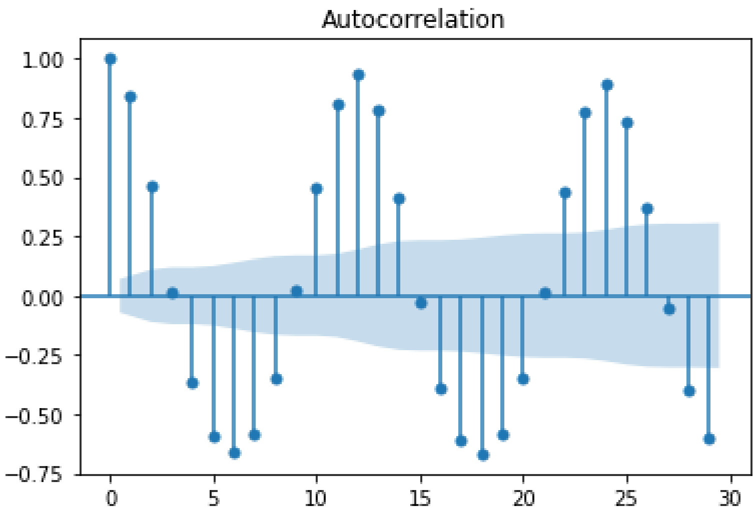
Given this result, I want to try to go in another direction to predict the seasonal component. I will rather try to adjust the existing model than decomposing the problem

Autoregressive approach

As the fourier gave nothing of interest, i will try to caracterize the seasonal component using an autoregressive model, meaning that i'll use the past values to predict the futur ones. The advantage is that it can express much more than a linear model. The disadvantage is that it's harder to use and when extrapolating, the error blow up in time as it uses past predictions to predict futur ones.

In [23]:

```
plot = plot_acf(periodic[~na_mask])
```



The ACF plot show the correlation of y_t and y_{t-x}. On this plot we clearly see a sinusoidal component. The wavelength would be around 12 units of time. As the samples are given on the 15 of each month, a period of 12 samples correspond to exactly a year, which is intuitively explainable.

Polynomial + autoregressive model

Given what we just said, we will try to improve our best model (the polynomial one) by adding the value of y that we are trying to predict with a lag of 12 time units:

$$y_t = \beta_0 + \beta_1 y_{t-12} + \beta_2 t + \beta_3 t^2$$

In [24]:

```
# Transform time and co2 series into polynomial and lagged variables
def make_ploynomial_and_seasonal(x, ts):
    return np.stack((ts[:-12], x[12:], x[12:]**2), axis=-1)

x = make_ploynomial_and_seasonal(annees, co2)
m = ~np.isnan(x).any(axis=-1)
res = make_reg(x[m], co2[12:][m])
res.summary()
```

Out[24]:

OLS Regression Results

Dep. Variable:	y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	6.291e+05
Date:	Thu, 30 Dec 2021	Prob (F-statistic):	0.00
Time:	10:57:49	Log-Likelihood:	-648.39
No. Observations:	743	AIC:	1305.
Df Residuals:	739	BIC:	1323.
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1660.8149	556.521	2.984	0.003	568.264	2753.365
x1	0.9709	0.010	101.610	0.000	0.952	0.990
x2	-1.7310	0.569	-3.041	0.002	-2.848	-0.614
x3	0.0005	0.000	3.101	0.002	0.000	0.001

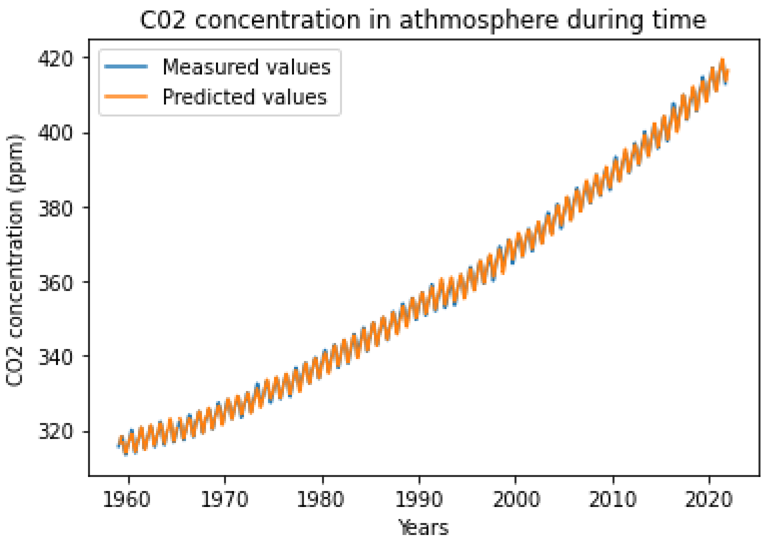
Omnibus:	8.559	Durbin-Watson:	0.540
Prob(Omnibus):	0.014	Jarque-Bera (JB):	8.611
Skew:	0.263	Prob(JB):	0.0135
Kurtosis:	3.027	Cond. No.	1.04e+11

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.04e+11. This might indicate that there are
strong multicollinearity or other numerical problems.

The model is now much better ! We have a R^2 of 1 meaning that we explained all the explainable variance. The AIC is way better than the preivous one, meaning that we can believe that we don't overfit. P-value wise, we lost the treshold of 0.001 but we are still better than 0.01.

In [25]:

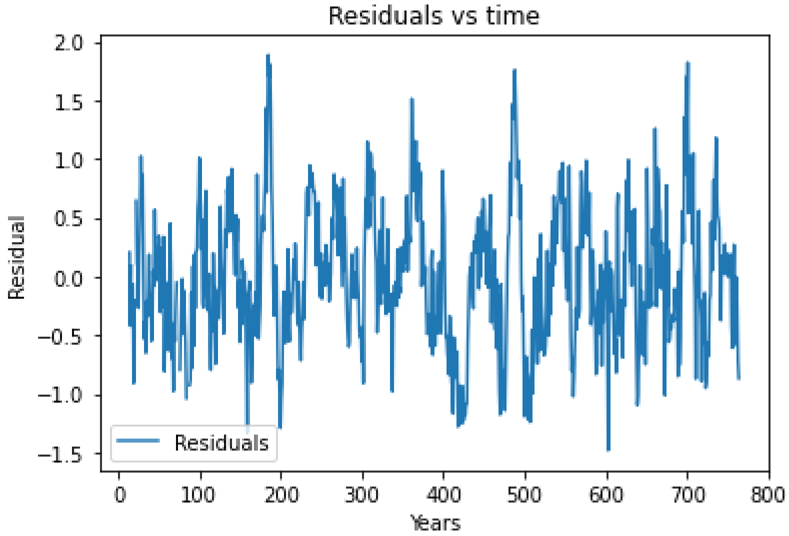
```
co2_pred = res.predict(sm.add_constant(make_ploynomial_and_seasonal(annees, co2
)))
plot_predictions_vs_real(annees[12:], co2_pred, co2[12:])
```

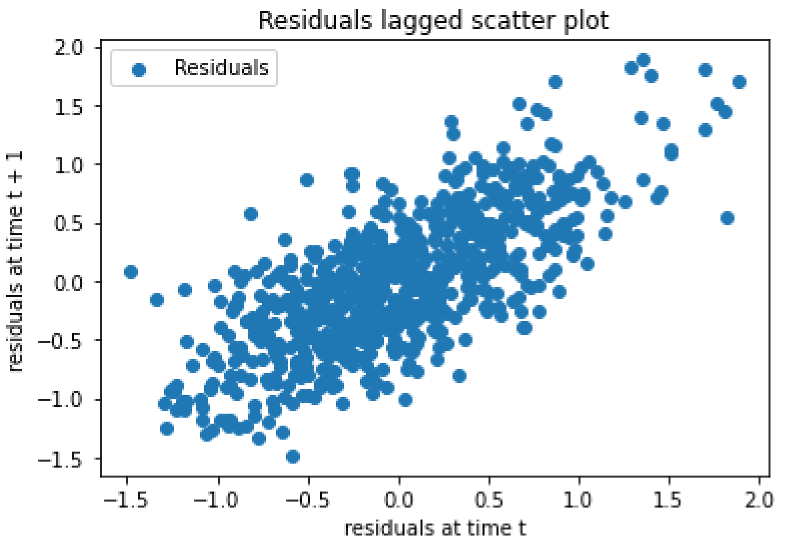


Visually the predictions looks very good. Some spikes can be seen sometimes that are slightly off the predictions but other than that nothing to really say.

In [26]:

```
plot_residuals(co2_pred, co2[12:])
```





The residuals still have some seasonal component that is not captured by the model, but there is no clear frequency associated to it.

Linear + seasonal model

As we saw earlier the coefficient associated to the t^2 term is very small. Let's try to simplify the model by removing it:

$$y_t = \beta_0 + \beta_1 y_{t-12} + \beta_2 t$$

In [27]:

```python
def make_linear_and_seasonal(x, ts):
    return np.stack((ts[:-12], x[12:]), axis=-1)

x = make_linear_and_seasonal(annees, co2)
m = ~np.isnan(x).any(axis=-1)
res = make_reg(x[m], co2[12:][m])
res.summary()
```

Out[27]:

OLS Regression Results

Dep. Variable:	y	R-squared:	1.000
Model:	OLS	Adj. R-squared:	1.000
Method:	Least Squares	F-statistic:	9.328e+05
Date:	Thu, 30 Dec 2021	Prob (F-statistic):	0.00
Time:	10:57:50	Log-Likelihood:	-653.19
No. Observations:	743	AIC:	1312.
Df Residuals:	740	BIC:	1326.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-64.2772	13.714	-4.687	0.000	-91.200	-37.355
x1	0.9965	0.005	206.437	0.000	0.987	1.006
x2	0.0337	0.008	4.358	0.000	0.019	0.049

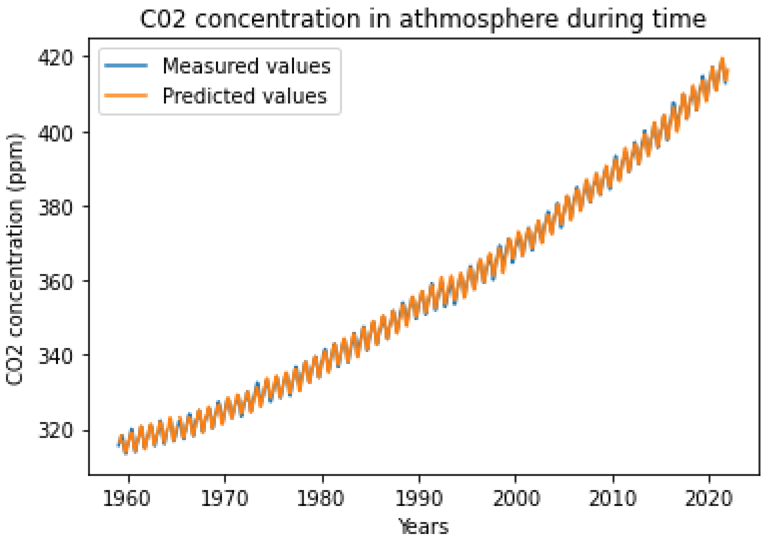
Omnibus:	6.486	Durbin-Watson:	0.543
Prob(Omnibus):	0.039	Jarque-Bera (JB):	6.374
Skew:	0.222	Prob(JB):	0.0413
Kurtosis:	3.094	Cond. No.	1.29e+06

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.29e+06. This might indicate that there are
strong multicollinearity or other numerical problems.

the R^2 is the same, and the p-values are very small. However, the AIC is slithly higher, so according to this metric it's worth it to keep the quadratic component in t he equation.

In [28]:

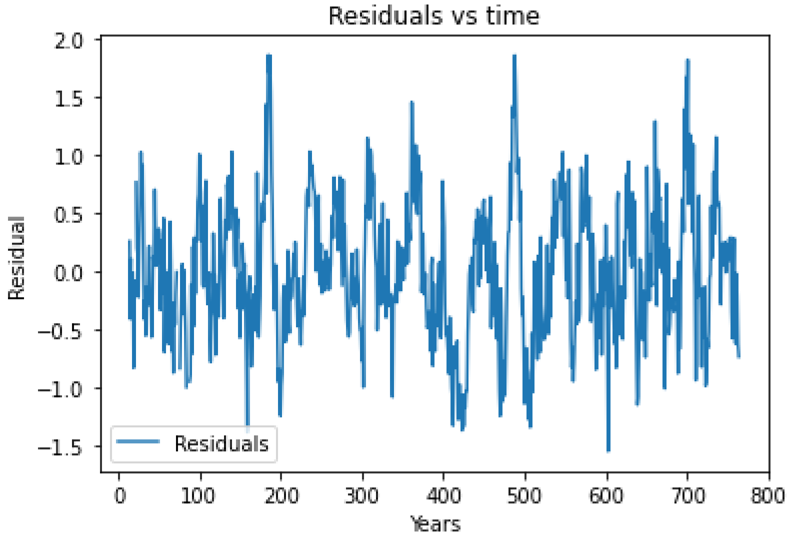
```
co2_pred = res.predict(sm.add_constant(make_linear_and_seasonal(annees, co2)))
plot_predictions_vs_real(annees[12:], co2_pred, co2[12:])
```

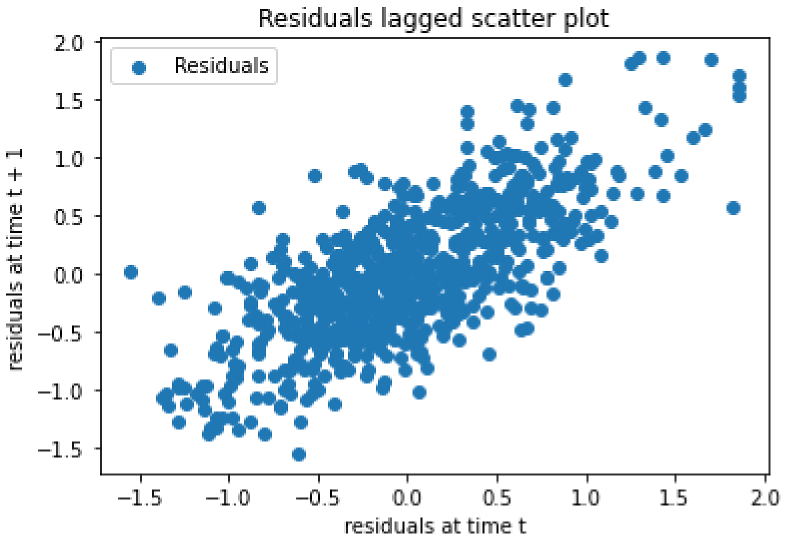


Visually there is no big difference with the previous model

In [29]:

```
plot_residuals(co2_pred, co2[12:])
```





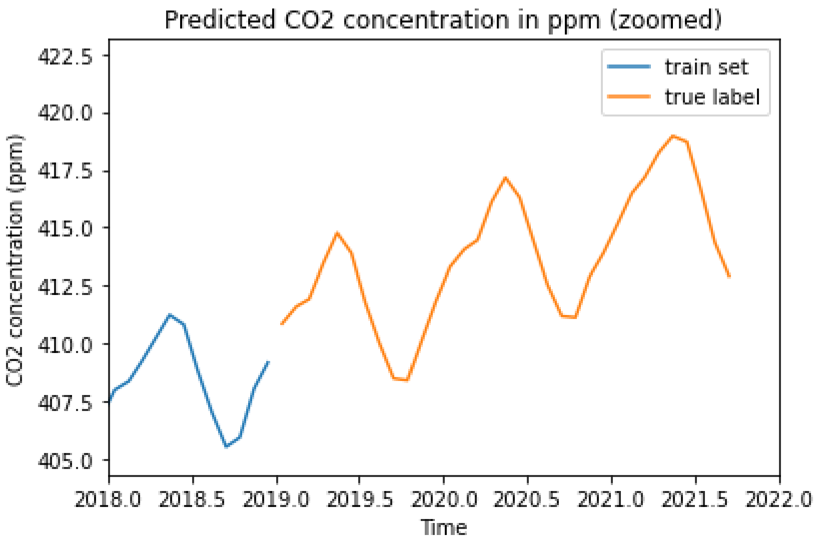
Really similar to the previous one

Test on the existing data

I will now test the considered models trained on a subset of the data, and see their performances on the remaining data. I don't argue that it proves anything, but at best it could prove us wrong. AS the final goal is to predict until 2025 and we are currently at the end of 2021, we will try to predict 2019, 2020, 2021 using past data

In [30]:

```
first_test_year = 2019
train_mask = annees < first_test_year
# Split into train and test set
annees_train, co2_train = annees[train_mask], co2[train_mask]
annees_test, co2_test = annees[~train_mask], co2[~train_mask]

def predictions_plot(preds=None, ci=None, zoom=False):
    plt.plot(annees_train, co2_train, label="train set")
    plt.plot(annees_test, co2_test, label="true label")
    if preds is not None:
        plt.plot(annees_test, preds, label="Predicted")
        plt.fill_between(annees_test, ci[:, 0], ci[:, 1], alpha=.3)
        # Plot MSE to have a numerical metric
        print("MSE={:.4f}".format(np.mean((preds - co2_test)**2)))
    if zoom:
        plt.title("Predicted CO2 concentration in ppm (zoomed)")
        x_window = [2018, 2022]
        m = np.logical_and(annees_test >= x_window[0], annees_test <= x_window[1
])
        if preds is not None:
            data = np.concatenate((co2_test[m].ravel(), preds[m].ravel(), ci[m].
ravel()))
        else:
            data = co2_test[m]
        plt.xlim(x_window)
        plt.ylim([np.nanmin(data)*0.99, np.nanmax(data)*1.01])
    else:
        plt.title("Predicted CO2 concentration in ppm")
    plt.xlabel("Time")
    plt.ylabel("CO2 concentration (ppm)")
    plt.legend()
    plt.show()

predictions_plot(zoom=True)
```

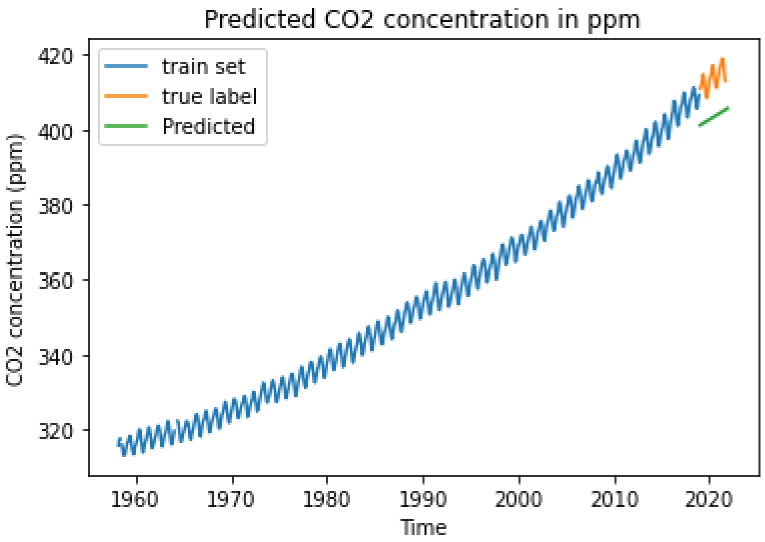


Linear model

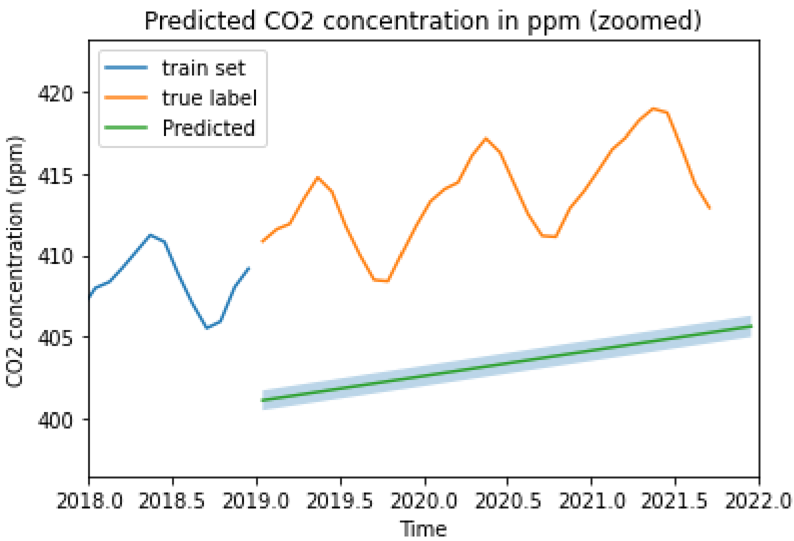
In [31]:

```
res = make_reg(annees_train, co2_train)
x = sm.add_constant(annees_test)
co2_pred = res.predict(x)
ci = res.get_prediction(x).conf_int()
predictions_plot(co2_pred, ci)
predictions_plot(co2_pred, ci, zoom=True)
```

MSE=116.6252



MSE=116.6252



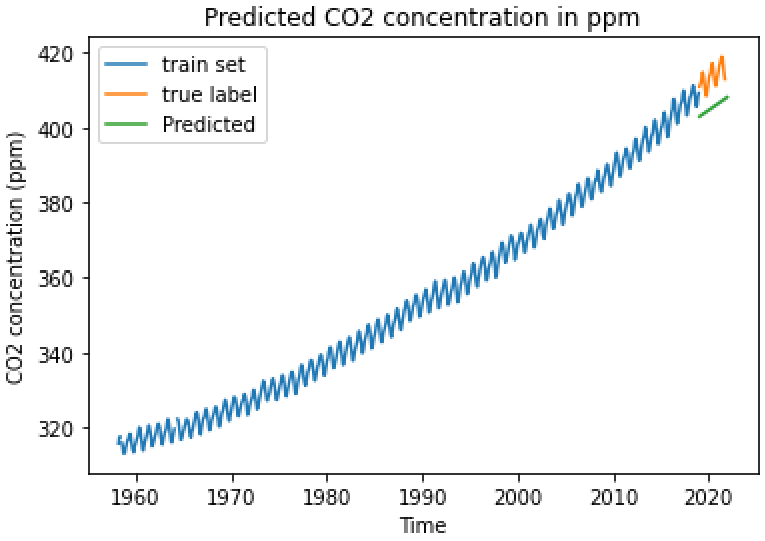
As expected, the linear model is not very good, the true value are really under-estimated. Furthermore, the confidence interval is wrong, probably because the assumptions of the model are violated

Log model

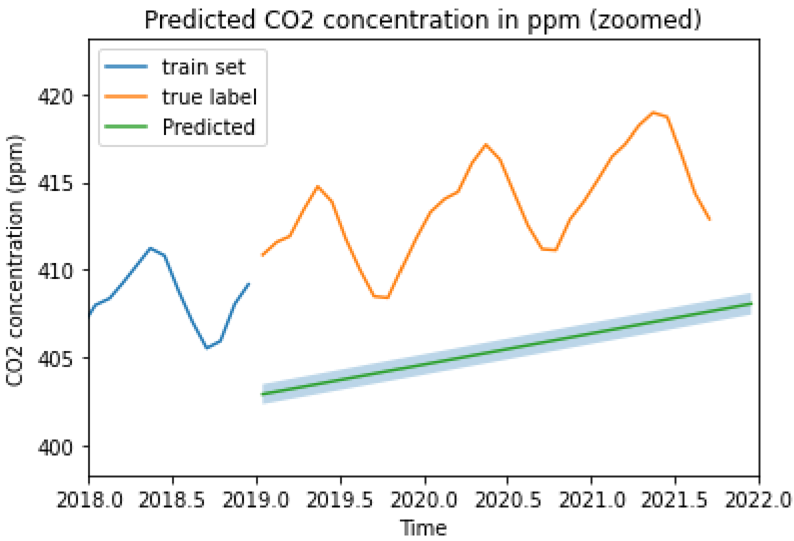
In [32]:

```
res = make_reg(annees_train, np.log(co2_train))
x = sm.add_constant(annees_test)
co2_pred = np.exp(res.predict(x))
ci = np.exp(res.get_prediction(x).conf_int())
predictions_plot(co2_pred, ci)
predictions_plot(co2_pred, ci, zoom=True)
```

MSE=76.9584



MSE=76.9584



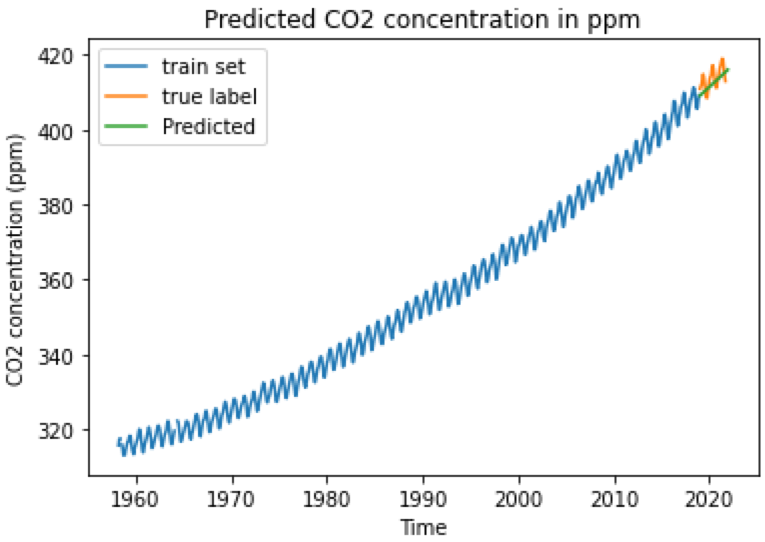
The exponential model is less wrong but still underestimates a lot the ground truth. Here again, the confidence interval are wrong probably for violated assumptions reasons (residuals not independant)

Polynomial model

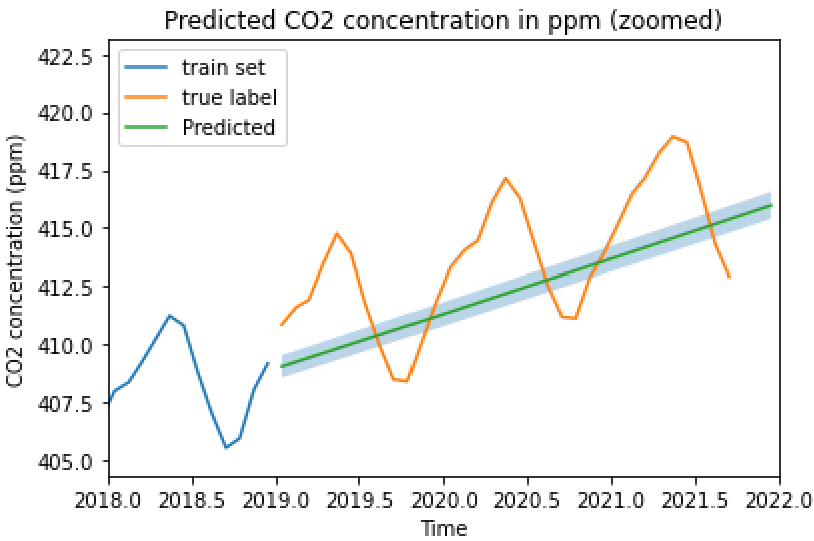
In [33]:

```
res = make_reg(np.stack((annees_train, annees_train**2), axis=-1), co2_train)
x = sm.add_constant(np.stack((annees_test, annees_test**2), axis=-1))
co2_pred = res.predict(x)
ci = res.get_prediction(x).conf_int()
predictions_plot(co2_pred, ci)
predictions_plot(co2_pred, ci, zoom=True)
```

MSE=7.4225



MSE=7.4225



The polynomial model is much better (MSE 10 times smaller than previous one) but as expected, the seasonal component is completly ignored. Once again confidence interval fail probably for the same reasons

Polynomial + lagged variable

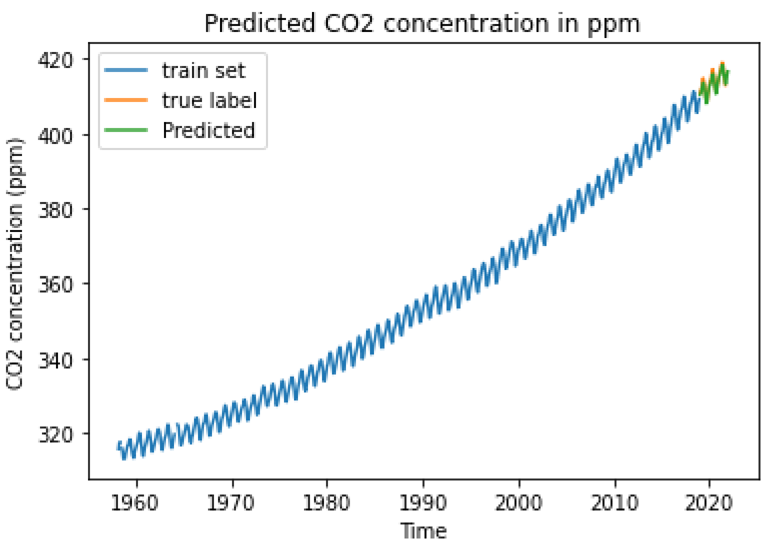
In [34]:

```
# The autoregressive model needs to predict gradualy as it can be necessary to rel
y on past predictions
def autoregressive_extrapolation(reg, f, annees_test, co2_train, min_lag=12):
    n = len(annees_test)
    preds = co2_train.values
    cis = np.zeros((len(co2_train), 2))
    cis[:, 0] = co2_train.values
    cis[:, 1] = co2_train.values
    while len(annees_test):
        co2_batch, annees_batch = preds[-min_lag:], annees_test[:min_lag]
        n_pred = len(annees_batch)
        annees_test = annees_test[min_lag:]
        preds = np.concatenate((preds, np.zeros(n_pred)))
        cis = np.concatenate((
            cis,
            np.stack((np.zeros(n_pred), np.zeros(n_pred)), axis=-1)
        ), axis=0)
        annees_batch = np.concatenate((np.zeros(min_lag), annees_batch))
        x = f(
            annees_batch,
            preds[-min_lag-n_pred:]
        )
        co2_pred = res.predict(sm.add_constant(x))
        # Lower bound calculated by considering the best case scenario
        ci_low = res.get_prediction(
            sm.add_constant(f(annees_batch, cis[-min_lag-n_pred:, 0]))
        ).conf_int()[:, 0]
        # Higher bound calculated by considering the worst case scenario
        ci_high = res.get_prediction(
            sm.add_constant(f(annees_batch, cis[-min_lag-n_pred:, 1]))
        ).conf_int()[:, 1]

        preds[-n_pred:] = co2_pred
        cis[-n_pred:, 0] = ci_low
        cis[-n_pred:, 1] = ci_high

    return preds[-n:], cis[-n:]
```

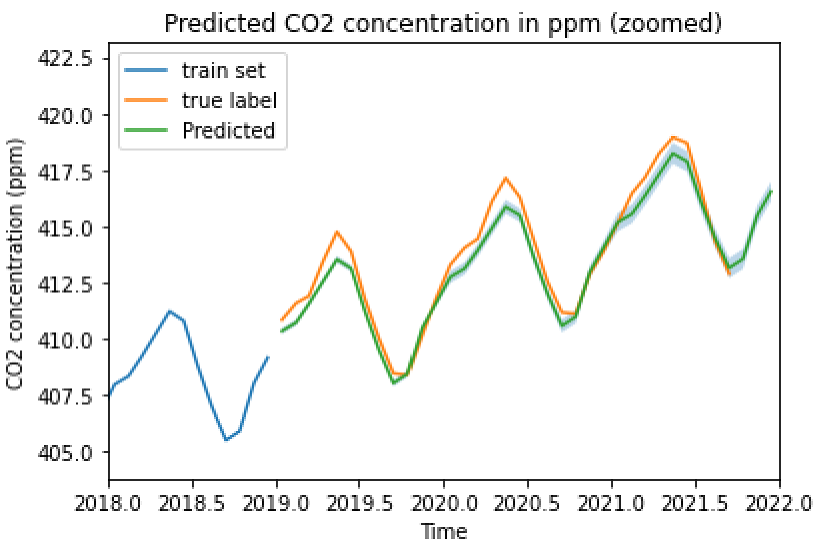
In [35]:

```
x = make_ploynomial_and_seasonal(annees_train, co2_train)
m = ~np.isnan(x).any(axis=-1)
res = make_reg(x[m], co2_train[12:][m])

co2_pred, ci = autoregressive_extrapolation(res, make_ploynomial_and_seasonal, a
nnees_test, co2_train)
predictions_plot(co2_pred, ci)
predictions_plot(co2_pred, ci, zoom=True)
```

MSE=0.4590



MSE=0.4590



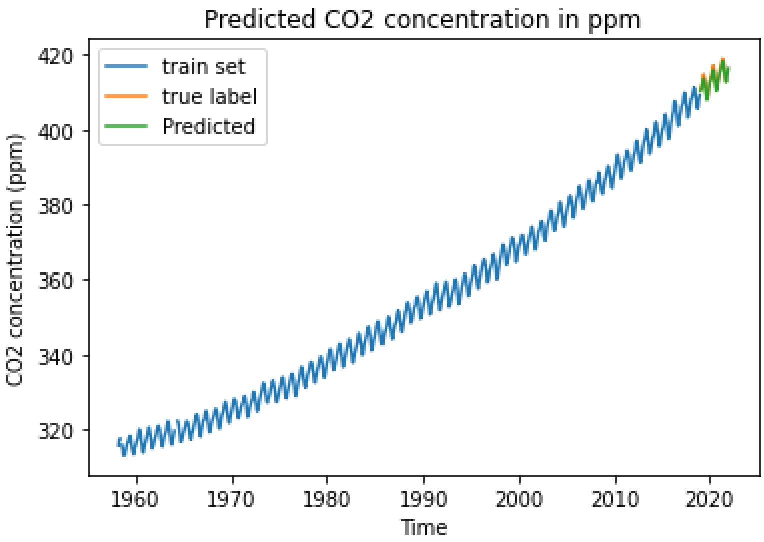
Visually, we can see that the model is very close to ground truth. I'm surprised that the confidence intervals are wrong given that I consider the worst possible case at each step. I belive that it's again due to the violated assumptions of linear models but if anyone have an idea please share it in comments.

Linear + lagged variable

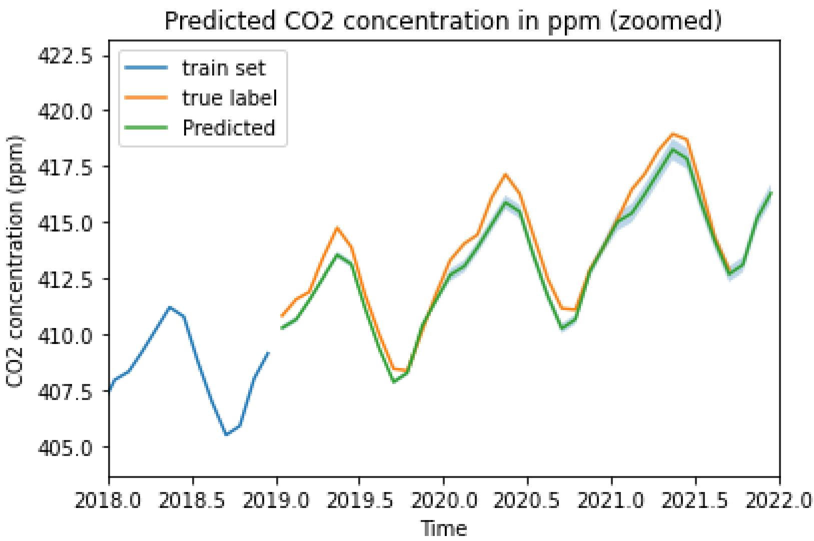
In [36]:

```
x = make_linear_and_seasonal(annees_train, co2_train)
m = ~np.isnan(x).any(axis=-1)
res = make_reg(x[m], co2_train[12:][m])

co2_pred, ci = autoregressive_extrapolation(res, make_linear_and_seasonal, annee
s_test, co2_train)
predictions_plot(co2_pred, ci)
predictions_plot(co2_pred, ci, zoom=True)
```

```
MSE=0.5421
```




```
MSE=0.5421
```



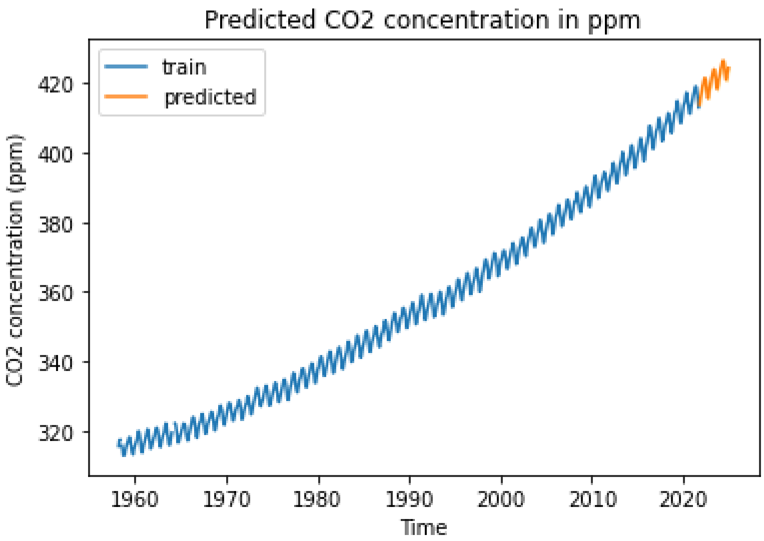
Same idea with the linear + seasonal. Do note however that the MSE is better for the polynomial than the linear one. At least it doesn't invalidate what we said about the AIC

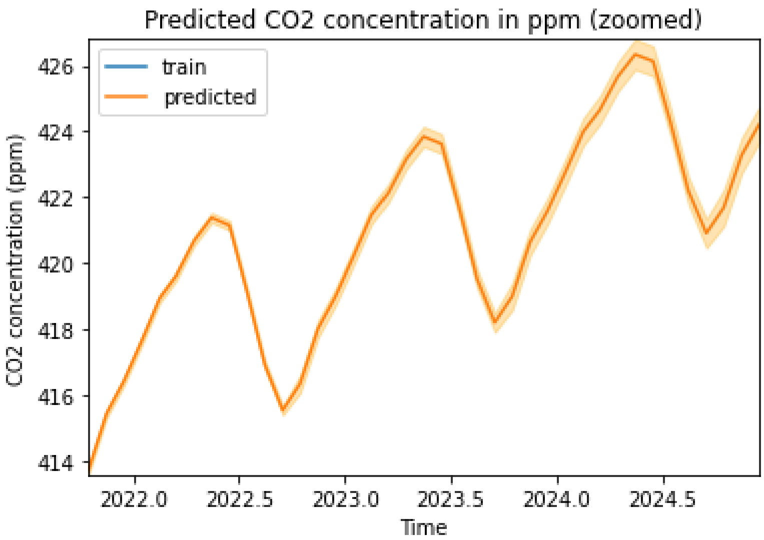
Final prediction

The best AIC so far is the polynomial one, so we will use it to make our predictions

In [37]:

```
x = make_ploynomial_and_seasonal(annees, co2)
m = ~np.isnan(x).any(axis=-1)
res = make_reg(x[m], co2[12:][m])

# Predict until 2025
annees_to_predict = [2022, 2023, 2024]
# Average over the measure date in the month (no big difference)
dates_prop = np.unique(annees - np.floor(annees)).reshape((-1, 2)).mean(axis=-1)
annees_pred = np.repeat(annees_to_predict, len(dates_prop)) + np.repeat(np.expan
d_dims(dates_prop, axis=0), 3, axis=0).ravel()
# Last months of current year are None so we have to also predict them
annees_pred = np.concatenate((annees[-3:], annees_pred))

co2_pred, ci = autoregressive_extrapolation(res, make_ploynomial_and_seasonal, a
nnees_pred, co2[:-3])
plt.title("Predicted CO2 concentration in ppm")
plt.plot(annees, co2, label="train")
plt.plot(annees_pred, co2_pred, label="predicted")
plt.fill_between(annees_pred, ci[:, 0], ci[:, 1], color='orange', alpha=.3)
plt.xlabel("Time")
plt.ylabel("CO2 concentration (ppm)")
plt.legend()
plt.show()

plt.title("Predicted CO2 concentration in ppm (zoomed)")
plt.plot(annees, co2, label="train")
plt.plot(annees_pred, co2_pred, label="predicted")
plt.fill_between(annees_pred, ci[:, 0], ci[:, 1], color='orange', alpha=.3)
plt.xlim([annees_pred.min(), annees_pred.max()])
plt.ylim([ci.min(), ci.max()])
plt.xlabel("Time")
plt.ylabel("CO2 concentration (ppm)")
plt.legend()
plt.show()
```



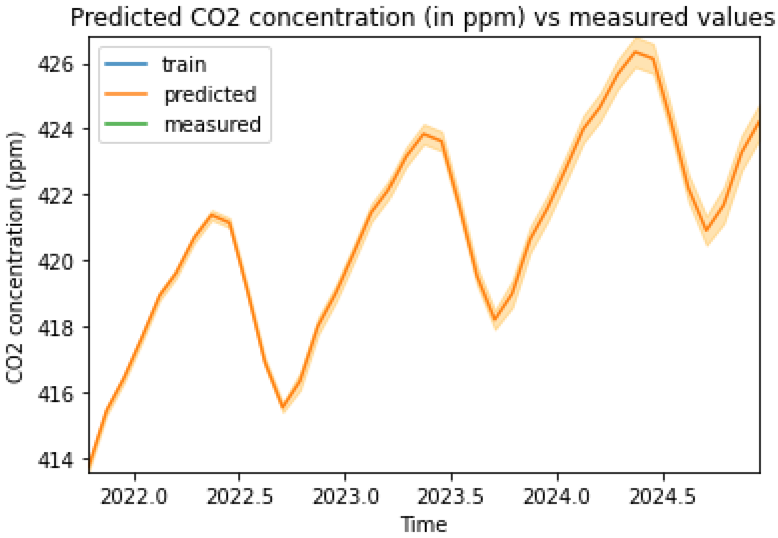


In [38]:

```
new_data = pd.read_csv("./co2_data_new.csv", comment='"', skiprows=[55, 56])
new_annee = new_data.iloc[:, 3].values
new_co2 = new_data.iloc[:, 4].values
m = new_annee >= annees_pred.min()
new_annee = new_annee[m]
new_co2 = new_co2[m]
n_missing = len(annees_pred) - len(new_annee)
if n_missing > 0:
    new_annee = np.concatenate((new_annee, annees_pred[-n_missing:]))
    new_co2 = np.concatenate((new_co2, np.ones(n_missing) * -99.99))
new_co2[new_co2 == -99.99] = np.nan
```

In [39]:

```
extrapolation = pd.DataFrame({
    "Year": np.floor(annees_pred).astype(np.int),
    "Month": map(lambda x: month_abbr[int(x)+1], np.floor((annees_pred - np.floo
r(annees_pred))*12)),
    "Co2 Prediction": co2_pred,
    "Co2 True value": new_co2,
    "0.05 ci": ci[:, 0],
    "0.95 ci": ci[:, 1],
    "In Interval": np.logical_and(ci[:, 0] <= new_co2, new_co2 <= ci[:, 1]),
    "Squared error": (co2_pred - new_co2)**2
})
plt.title("Predicted CO2 concentration (in ppm) vs measured values")
plt.plot(annees, co2, label="train")
plt.plot(annees_pred, co2_pred, label="predicted")
plt.fill_between(annees_pred, ci[:, 0], ci[:, 1], color='orange', alpha=.3)
plt.plot(annees_pred, new_co2, label="measured")
plt.xlim([annees_pred.min(), annees_pred.max()])
plt.ylim([ci.min(), ci.max()])
plt.xlabel("Time")
plt.ylabel("CO2 concentration (ppm)")
plt.legend()
plt.show()
print("So far, MSE={:.4f}".format(np.nanmean(extrapolation["Squared error"])))
extrapolation
```




```
So far, MSE=nan
```

Out[39]:

	Year	Month	Co2 Prediction	Co2 True value	0.05 ci	0.95 ci	In Interval	Squared error
0	2021	Oct	413.719579	NaN	413.586094	413.853064	False	NaN
1	2021	Nov	415.436996	NaN	415.309855	415.564138	False	NaN
2	2021	Dec	416.425995	NaN	416.299282	416.552708	False	NaN
3	2022	Jan	417.657984	NaN	417.529281	417.786688	False	NaN
4	2022	Feb	418.948227	NaN	418.814268	419.082185	False	NaN
5	2022	Mar	419.626134	NaN	419.488528	419.763740	False	NaN
6	2022	Apr	420.683390	NaN	420.538544	420.828236	False	NaN
7	2022	May	421.381154	NaN	421.231170	421.531137	False	NaN
8	2022	Jun	421.147186	NaN	421.000641	421.293731	False	NaN
9	2022	Jul	419.165415	NaN	419.031781	419.299050	False	NaN
10	2022	Aug	416.931509	NaN	416.797282	417.065735	False	NaN
11	2022	Sep	415.542248	NaN	415.398730	415.685767	False	NaN
12	2022	Oct	416.346423	NaN	416.075847	416.615479	False	NaN
13	2022	Nov	418.022560	NaN	417.763799	418.280820	False	NaN
14	2022	Dec	418.991234	NaN	418.733203	419.249285	False	NaN
15	2023	Jan	420.196101	NaN	419.934320	420.458581	False	NaN
16	2023	Feb	421.457530	NaN	421.185944	421.730534	False	NaN
17	2023	Mar	422.123769	NaN	421.845334	422.403949	False	NaN
18	2023	Apr	423.159020	NaN	422.867056	423.453307	False	NaN
19	2023	May	423.844979	NaN	423.543365	424.149258	False	NaN
20	2023	Jun	423.626653	NaN	423.331314	423.924352	False	NaN
21	2023	Jul	421.711197	NaN	421.439624	421.983661	False	NaN
22	2023	Aug	419.551242	NaN	419.278320	419.823437	False	NaN
23	2023	Sep	418.211319	NaN	417.920840	418.499988	False	NaN
24	2023	Oct	419.000613	NaN	418.588995	419.408066	False	NaN
25	2023	Nov	420.636751	NaN	420.241638	421.030588	False	NaN
26	2023	Dec	421.585767	NaN	421.191574	421.980160	False	NaN
27	2024	Jan	422.764380	NaN	422.364875	423.166000	False	NaN
28	2024	Feb	423.997911	NaN	423.584670	424.415311	False	NaN
29	2024	Mar	424.652892	NaN	424.229992	425.080882	False	NaN
30	2024	Apr	425.666857	NaN	425.224978	426.115473	False	NaN
31	2024	May	426.341430	NaN	425.885946	426.804628	False	NaN
32	2024	Jun	426.138367	NaN	425.691460	426.592119	False	NaN
33	2024	Jul	424.287367	NaN	423.873272	424.704129	False	NaN
34	2024	Aug	422.199285	NaN	421.782941	422.613720	False	NaN
35	2024	Sep	420.907338	NaN	420.466090	421.343602	False	NaN
36	2024	Oct	421.682260	NaN	421.125325	422.231607	False	NaN
37	2024	Nov	423.279641	NaN	422.743207	423.813950	False	NaN
38	2024	Dec	424.209646	NaN	423.674226	424.745731	False	NaN

Conclusion

We manage to fit a model using a quadratic growth + a seasonal (yearly) component with a test MSE less than 1ppm. We failed to provide a valid confidence interval and we believe it may due to violated assumptions of linear models, namely the independance between error samples. There is still periodic comopnents in the residuals that could be reolved by further work.