MAUNA LOA CO_2 FLASK DATA ANALYSIS

A Beginner's Guide (Python Version)

This tutorial will walk you through how to work with real atmospheric CO_2 data collected at the **Mauna Loa Observatory** in Hawaii. This data is part of NOAA's Global Monitoring Laboratory (GML) long-term record of greenhouse gases. The Mauna Loa Observatory provides one of the longest continuous records of atmospheric CO_2, making it a vital source for atmospheric research worldwide.

Whether you're new to Python, atmospheric science, or data analysis, this guide is designed to help you get started step-by-step.

In this tutorial, you'll learn how to:

- **Load data** in two common scientific formats:
  - Plain text (.txt)
  - NetCDF (.nc)
- **Explore the dataset structure**, including:
  - Viewing metadata
  - Listing available variables
  - Checking data size and data types
- **Filter the data**, such as:
  - Removing rejected values (flagged entries)
  - Narrowing down by date range or custom conditions
- **Visualize CO_2 levels** over time using Python plotting tools
- **Apply NOAA GML's curve-fitting method** to:
  - Identify long-term trends
  - Separate out seasonal variations

Step 1: Import Required Python Packages

To begin our analysis, we need to load some Python libraries. These packages will help us:

- Load and work with data tables
- Perform calculations and filter data

- Create time series plots

We'll also import two custom Python modules provided by NOAA GML:

- `ccg_filter.py`: contains the filtering and curve-fitting tools
- `ccg_dates.py`: provides date conversion and handling functions used in GML's analysis

Make sure these two files (`ccg_filter.py` and `ccg_dates.py`) are in the same folder as this notebook.

In [39]:
```
# Import common Python packages
import pandas as pd  # For loading and working with tables (dataframes)
import matplotlib.pyplot as plt  # For plotting
import matplotlib.dates as mdates  # For formatting time axes in plots

# Import NOAA GML custom modules
import ccg_dates  # Date utilities
import ccg_filter  # Data filtering and curve fitting
```

Tip: Set Default Plotting Styles Early

You can customize the default appearance of all your plots by setting `matplotlib` parameters at the beginning of your notebook. This isn't required, but it helps keep your plots consistent and easier to read.

This is especially useful when you're making multiple plots throughout your analysis.

In [40]:
```
# Set default plot styles for consistent formatting
plt.rcParams['font.size'] = 16           # Base font size for text
plt.rcParams['axes.titlesize'] = 18      # Title font size for each plot
plt.rcParams['axes.labelsize'] = 16      # Font size for axis labels
plt.rcParams['xtick.labelsize'] = 14     # Font size for tick labels on x-ax
plt.rcParams['ytick.labelsize'] = 14     # Font size for tick labels on y-ax
plt.rcParams['legend.fontsize'] = 14     # Font size for plot legends
```

Step 2: Load the CO_2 Data (Text Format)

NOAA's Global Monitoring Laboratory provides Mauna Loa CO_2 measurements in an ASCII text format. This file contains both metadata (information about the data) and the actual measurement values. If you prefer to use the NetCDF file instead of the text file, see the Appendix for loading and processing instructions.

In this step, we'll:

- Locate and read the file
- Skip the metadata lines
- Load the data into a table (called a DataFrame) using the `pandas` library

To learn more about how these CO_2 flask samples are collected, visit:
https://gml.noaa.gov/ccgg/behind_the_scenes/surface.html

In [41]:
```
# Insert the path to your downloaded ASCII text file
file_path = "co2_mlo_surface-flask_1_ccgg_event.txt"

# First, find out how many header lines there are (they start with "#")
with open(file_path, "r") as file:
    header_lines = [line.strip() for line in file if line.startswith('#')]

# Count the number of header lines so we can skip them when reading the data
n = len(header_lines)

# Print the number of lines to skip
print(f"Number of header lines: {n}")
```

```
Number of header lines: 148
```

In [42]:
```
# Now read the data, skipping the header lines
flask_data = pd.read_csv(
    file_path,
    sep=r'\s+',       # Use whitespace as the separator
    skiprows=n,       # Skip the metadata lines
    header=0          # First remaining line is the column header
)

# Display the first 5 rows of the data
flask_data.head()
```

Out[42]:

	site_code	year	month	day	hour	minute	second	datetime	time_decimal
0	MLO	1969	8	20	17	55	0	1969-08-20T17:55:00Z	1969.634922
1	MLO	1969	8	20	17	55	0	1969-08-20T17:55:00Z	1969.634922
2	MLO	1969	8	20	18	30	0	1969-08-20T18:30:00Z	1969.634989
3	MLO	1969	8	20	18	30	0	1969-08-20T18:30:00Z	1969.634989
4	MLO	1969	8	27	19	15	0	1969-08-27T19:15:00Z	1969.654252

5 rows × 22 columns

Step 3: Explore the Data

Now that we have loaded the CO_2 data, let's explore the dataset.

We will:

- Review metadata or file header info
- Examine column names and data types
- Check dataset size (rows × columns)
- Verify date/time formats

In [43]:
```
# Print the metadata/header rows at the top of the file
with open(file_path, "r") as file:
    header_lines = [line.strip() for line in file if line.startswith('#')]

print("File Header / Metadata:\n")
for line in header_lines:
    print(line)
```

File Header / Metadata:

```
# header_lines : 149
#
# ------------------------------------------------------------->>>>
# DATA SET NAME
#
# dataset_name: co2_mlo_surface-flask_1_ccgg_event
#
# ------------------------------------------------------------->>>>
# DESCRIPTION
#
# dataset_description: Atmospheric Carbon Dioxide Dry Air Mole Fractions fro
m the NOAA GML Carbon Cycle Cooperative Global Air Sampling Network, startin
g in 1968
#
# ------------------------------------------------------------->>>>
# CITATION
#
# dataset_citation: Lan, X., J.W. Mund, A.M. Crotwell, K.W. Thoning, E. Mogl
ia, M. Madronich, K. Baugh, G. Petron, M.J. Crotwell, D. Neff, S. Wolter, T.
Mefford and S. DeVogel (2025), Atmospheric Carbon Dioxide Dry Air Mole Fract
ions from the NOAA GML Carbon Cycle Cooperative Global Air Sampling Network,
1968-2024,  Version: 2025-04-26, https://doi.org/10.15138/wkgj-f215
#
# ------------------------------------------------------------->>>>
# FAIR USE POLICY
#
# dataset_fair_use: These data are made freely available to the public and t
he scientific community in the belief that their wide dissemination will lea
d to greater understanding and new scientific insights. To ensure that GML r
eceives fair credit for their work please include relevant citation text in
publications. We encourage users to contact the data providers, who can prov
ide detailed information about the measurements and scientific insight.  In
cases where the data are central to a publication, coauthorship for data pro
viders may be appropriate.
#
# ------------------------------------------------------------->>>>
# WARNING
#
# dataset_warning: Every effort is made to produce the most accurate and pre
cise measurements possible.  However, we reserve the right to make correctio
ns to the data based on recalibration of standard gases or for other reasons
deemed scientifically justified.  We are not responsible for results and con
clusions based on use of these data without regard to this warning.
#
# ------------------------------------------------------------->>>>
# GLOBAL ATTRIBUTES
#
# site_code : MLO
# site_name : Mauna Loa, Hawaii
# site_country : United States
# site_country_flag : https://gml.noaa.gov/webdata/ccgg/ObsPack/images/flag
s/UNST0001.GIF
# site_latitude : 19.5362
# site_longitude : -155.5763
```

```
# site_elevation : 3397.0
# site_elevation_unit : masl
# site_position_comment : This is the nominal location of the site. The samp
ling location at many sites has changed over time, and we report here the mo
st recent nominal location. The actual sampling location for each observatio
n is not necessarily the site location. The sampling locations for each obse
rvation are reported in the latitude, longitude, and altitude variables.
# site_utc2lst : -10.0
# site_utc2lst_comment : Add 'site_utc2lst' hours to convert a time stamp in
UTC (Coordinated Universal Time) to LST (Local Standard Time).
# site_url : https://gml.noaa.gov/obop/mlo/
# dataset_creation_date : 2025-04-26T12:49:21.678589
# dataset_num : 50
# dataset_name : co2_mlo_surface-flask_1_ccgg_event
# dataset_parameter : co2
# dataset_parameter_name : Carbon Dioxide
# dataset_parameter_industrial_name : Carbon Dioxide
# dataset_project : surface-flask
# dataset_platform : fixed
# dataset_selection : event
# dataset_selection_tag : event
# dataset_comment : For more information about these data, please see http
s://gml.noaa.gov/aftp/data/trace_gases/co2/flask/README_co2_surface-flask_cc
gg.html
# dataset_calibration_scale : CO2_X2019
# dataset_start_date : 1969-08-20T17:55:00Z
# dataset_stop_date : 2025-04-03T19:00:00Z
# dataset_usage_description : Please cite the product's citation when using
data from this dataset. Relevant literature references for this dataset are
listed below for convenience.
# dataset_provider_license : These data were produced by NOAA and are not su
bject to copyright protection in the United States. NOAA waives any potentia
l copyright and related rights in these data worldwide through the Creative
Commons Zero 1.0 Universal Public Domain Dedication (CC0-1.0).
# dataset_reference_total_listed : 1
# dataset_reference_1_name : Conway, T.J., P.P. Tans, L.S. Waterman, K.W. Th
oning, D.R. Kitzis, K.A. Masarie, and N. Zhang, Evidence for interannual var
iability of the carbon cycle from the NOAA/GMD global air sampling network,
J. Geophys. Res.,99,22831 22855, 1994.
# dataset_contribution : These data are provided by NOAA. Principal investig
ators include Xin Lan (NOAA).
# lab_total_listed : 1
# lab_1_number : 1
# lab_1_abbr : NOAA
# lab_1_name : NOAA Global Monitoring Laboratory
# lab_1_address1 : 325 Broadway
# lab_1_address2 : NOAA R/GML-1
# lab_1_address3 : Boulder, CO 80305-3328
# lab_1_country : United States
# lab_1_url : https://gml.noaa.gov/ccgg/
# lab_1_parameter : Lab has contributed measurements for: co2
# lab_1_country_flag : https://gml.noaa.gov/webdata/ccgg/ObsPack/images/flag
s/UNST0001.GIF
# lab_1_logo : https://gml.noaa.gov/webdata/ccgg/ObsPack/images/logos/noaa_m
edium.png
# lab_1_ongoing_atmospheric_air_comparison : T
```

```
# provider_total_listed : 1
# provider_1_name : Xin Lan
# provider_1_address1 : NOAA GML
# provider_1_address2 : 325 Broadway R/GML-1
# provider_1_address3 : Boulder, CO 80305-3328
# provider_1_country : United States
# provider_1_affiliation : National Oceanic and Atmospheric Administration
# provider_1_email : Xin.Lan@noaa.gov
# provider_1_parameter : Provider has contributed measurements for: co2
# ------------------------------------------------------------>>>>
# VARIABLE ATTRIBUTES
#
# site_code:long_name : site_name_abbreviation.
# site_code:comment : Site code is an abbreviation for the sampling site nam
e.
# time_components:_FillValue : -9
# time_components:long_name : integer_components_of_UTC_date/time
# time_components:order : year, month, day, hour, minute, second
# time_components:comment : Calendar time components as integers. Times and
dates are UTC. Time-averaged values are reported at the start of the averagi
ng interval.
# datetime:long_name : air_sample_date_and_time_in_UTC
# datetime:comment : Air sample date and time in UTC ISO 8601 format. Time-a
veraged values are reported at the start of the averaging interval.
# time_decimal:_FillValue : -999.999
# time_decimal:long_name : sample_decimal_year_in_UTC
# time_decimal:comment : decimal year in UTC. Time-averaged values are repor
ted at the start of the averaging interval.
# air_sample_container_id:long_name : Air_Sample_Container_ID
# air_sample_container_id:comment : ID of air sample container. See provider
_comment if available.
# value:_FillValue : -999.999
# value:long_name : measured_mole_fraction_of_trace_gas_in_dry_air
# value:units : micromol mol-1
# value:comment : Mole fraction reported in units of micromol mol-1 (10-6 mo
l per mol of dry air); abbreviated as ppm (parts per million).
# value:scale_comment : CO2_X2019
# value_unc:_FillValue : -999.999
# value_unc:long_name : estimated_uncertainty_in_reported_value
# value_unc:units : micromol mol-1
# value_unc:comment : This is the estimated uncertainty of the reported valu
e. See provider_comment if available.
# latitude:_FillValue : -999.999
# latitude:standard_name : latitude
# latitude:long_name : sample_latitude_in_decimal_degrees
# latitude:units : degrees_north
# latitude:comment : Latitude at which air sample was collected.
# longitude:_FillValue : -999.999
# longitude:standard_name : longitude
# longitude:long_name : sample_longitude_in_decimal_degrees
# longitude:units : degrees_east
# longitude:comment : Longitude at which air sample was collected using a ra
nge of -180 degrees to +180 degrees.
# altitude:_FillValue : -999.999
# altitude:standard_name : altitude
# altitude:long_name : sample_altitude_in_meters_above_sea_level
```

```
# altitude:units : m
# altitude:comment : Altitude (in meters above sea level). See provider_comm
ent if available.
# altitude:provider_comment : Altitude for this dataset is the sum of surfac
e elevation (masl) and sample intake height (magl).
# elevation:_FillValue : -999.999
# elevation:standard_name : elevation
# elevation:long_name : surface_elevation_in_meters_above_sea_level
# elevation:units : m
# elevation:comment : Surface elevation in meters above sea level. See provi
der_comment if available.
# intake_height:_FillValue : -999.999
# intake_height:long_name : sample_intake_height_in_meters_above_ground_leve
l
# intake_height:units : m
# intake_height:comment : Sample intake height in meters above ground level
(magl). See provider_comment if available.
# method:long_name : air_sample_collection_method
# method:comment : Air sample collection method. See provider_comment if ava
ilable.
# method:provider_comment : A single-character code is used to identify the
sample collection method. The codes are: P - Sample collected using a portab
le, battery powered pumping unit. Two flasks are connected in series, flushe
d with air, and then pressurized to 1.2 - 1.5 times ambient pressure. D - Si
milar to P but the air passes through a condenser cooled to about 5 deg C to
partially dry the sample. G - Similar to D but with a gold-plated condenser.
T - Evacuated flask filled by opening an O-ring sealed stopcock. S - Flasks
filled at NOAA GML observatories by sampling air from the in situ CO2 measur
ement air intake system. N - Before 1981, flasks filled using a hand-held as
pirator bulb. After 1981, flasks filled using a pump different from those us
ed in method P, D, or G. F - Five liter evacuated flasks filled by opening a
ground glass, greased stopcock
# event_number:long_name : Unique_Air_Sample_Event_Number
# event_number:comment : Identifies each discrete air sample collected at so
me time and location with a unique sample event number.  The event number (r
eported as a string) can be used to relate measurements of different trace g
ases and isotopes from the same sample.
# instrument:long_name : instrument_ID_to_detect_atmospheric_parameter
# instrument:comment : Instrument ID used to detect atmospheric parameter. S
ee provider_comment if available.
# analysis_datetime:long_name : air_sample_measurement_date_and_time_in_LT
# analysis_datetime:comment : Air sample measurement date and time in LT. Se
e provider_comment if available.
# qcflag:long_name : quality_control_flag
# qcflag:comment : This quality control flag is provided by the contributing
PIs. See provider_comment if available.
# qcflag:provider_comment : This is the NOAA 3-character quality control fla
g.  Column 1 is the REJECTION flag. An alphanumeric other than a period (.)
in the FIRST column indicates a sample with obvious problems during collecti
on or analysis. This measurement should not be interpreted. Column 2 is the
SELECTION flag. An alphanumeric other than a period (.) in the SECOND column
indicates a sample that is likely valid but does not meet selection criteria
determined by the goals of a particular investigation. For example, it might
not have been obtained during 'background' conditions.  Column 3 is the INFO
RMATION flag. An alphanumeric other than a period (.) in the THIRD column pr
ovides additional information about the collection or analysis of the sampl
```

```
e. A P in the 3rd column of the QC flag indicates the measurement result is
preliminary and has not yet been carefully examined by the PI.  The P flag i
s removed once the quality of the measurement has been determined.
#
# VARIABLE ORDER
#
```

> The header metadata gives context about how the data was collected and what each column means - don't skip it!

In [44]:
```
# Display the column names in the DataFrame
flask_data.columns
```

Out[44]:
```
Index(['site_code', 'year', 'month', 'day', 'hour', 'minute', 'second',
       'datetime', 'time_decimal', 'air_sample_container_id', 'value',
       'value_unc', 'latitude', 'longitude', 'altitude', 'elevation',
       'intake_height', 'method', 'event_number', 'instrument',
       'analysis_datetime', 'qcflag'],
      dtype='object')
```

In [45]:
```
# Check the number of rows and columns (shape of the DataFrame)
flask_data.shape
```

Out[45]:
```
(10972, 22)
```

In [46]:
```
# Check the data type of each column
flask_data.dtypes
```

Out[46]:
```
site_code                 object
year                       int64
month                      int64
day                        int64
hour                       int64
minute                     int64
second                     int64
datetime                  object
time_decimal             float64
air_sample_container_id   object
value                    float64
value_unc                float64
latitude                 float64
longitude                float64
altitude                 float64
elevation                float64
intake_height            float64
method                    object
event_number               int64
instrument                object
analysis_datetime         object
qcflag                    object
dtype: object
```

For easier plotting and analysis, we will convert the 'datetime' column values to a datetime object.

In [47]:
```
# Convert the 'datetime' column to actual datetime objects, if it's not alre
flask_data['datetime'] = pd.to_datetime(flask_data['datetime'])

# Check data types again to confirm the change
flask_data.dtypes
```

Out[47]:
```
site_code                             object
year                                   int64
month                                  int64
day                                    int64
hour                                   int64
minute                                 int64
second                                 int64
datetime                  datetime64[ns, UTC]
time_decimal                          float64
air_sample_container_id                object
value                                 float64
value_unc                             float64
latitude                              float64
longitude                             float64
altitude                              float64
elevation                             float64
intake_height                         float64
method                                 object
event_number                           int64
instrument                             object
analysis_datetime                      object
qcflag                                 object
dtype: object
```

Step 4: Filter the Data and Plot CO_2 Levels Over Time

Quality control (QC) flags are included in this dataset to help identify measurements that should be rejected for scientific use.

Quality Control (QC) Flags

The `qcflag` column uses a three-character code to describe measurement quality. Example: `"C.."`, `".S."`, or `"..."`.

- If the first character **is not** a dot (`.`), it means the measurement was **rejected** by the principal investigator.
- If the first character **is** a dot (`.`), the measurement is considered **valid**.

To ensure we're working with only high-quality data — and to keep our dataset manageable — we'll:

- Remove any rows where the **first character in `qcflag` is not `.`** (indicating the measurement was rejected)
- Limit the dataset to a span of **15 years** (2008-2023)

In [48]:
```
# Filter to include only measurements from 2008-2023
flask_data = flask_data[(flask_data['year'] >= 2008) &
                        (flask_data['year'] <= 2023)]

# Create a mask for QC flags that start with '.' (valid data)
valid_qc_mask = flask_data['qcflag'].str.match(r'^\..*')

# Filter to include only valid data
good_flask_data = flask_data[valid_qc_mask].reset_index(drop=True)

# Display the first few rows of the cleaned data
good_flask_data.head()
```

Out[48]:

	site_code	year	month	day	hour	minute	second	datetime	time_decima
0	MLO	2008	1	2	20	8	0	2008-01-02 20:08:00+00:00	2008.005024
1	MLO	2008	1	2	20	8	0	2008-01-02 20:08:00+00:00	2008.005024
2	MLO	2008	1	2	20	35	0	2008-01-02 20:35:00+00:00	2008.005076
3	MLO	2008	1	2	20	35	0	2008-01-02 20:35:00+00:00	2008.005076
4	MLO	2008	1	10	20	14	0	2008-01-10 20:14:00+00:00	2008.026894

5 rows × 22 columns

In [49]:
```
# Check the new shape of the filtered dataset
# The number of columns will remain the same, but the amount of rows will de
good_flask_data.shape
```

Out[49]: (3255, 22)

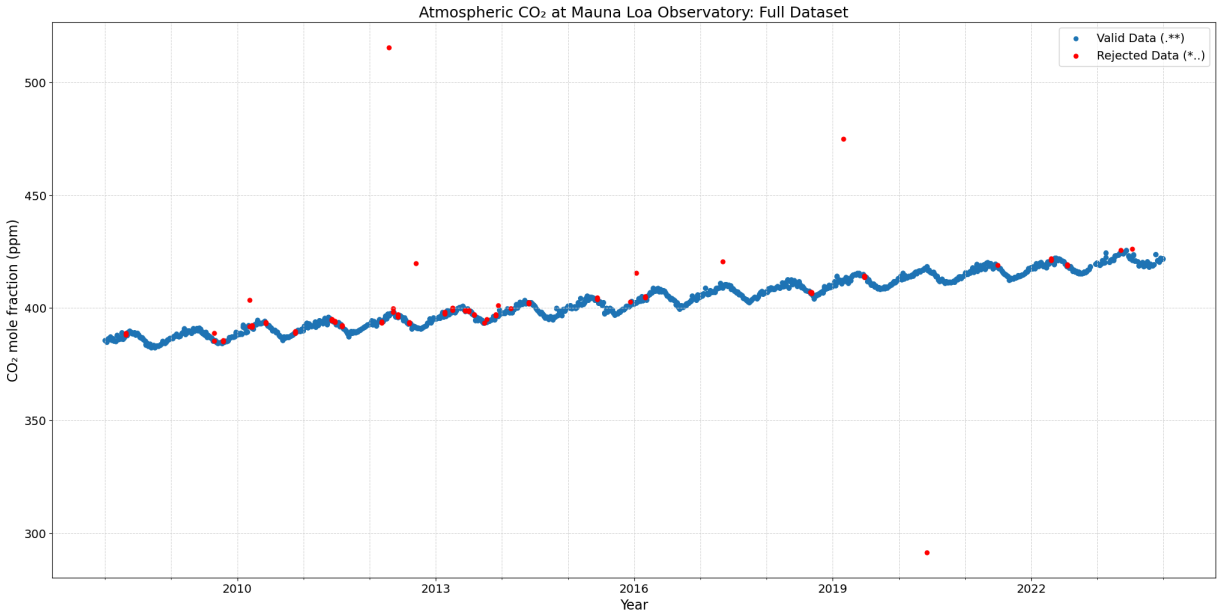
The plot below shows **both valid and rejected** flask CO_2 measurements.

Rejected data points often appear as outliers or noisy scatter, making it harder to see the real pattern in atmospheric CO_2 levels.

> *Each point represents one flask sample, plotted by sample date vs. CO_2 mole fraction.*

In [50]:
```
# Plot showing both valid and rejected measurements
plt.figure(figsize=(25, 12))

# Create mask for valid data (qcflag starts with '.')
mask = flask_data['qcflag'].str.match(r'^\..*')
```

```
# Plot valid points
plt.scatter(
    flask_data.loc[mask, 'datetime'],
    flask_data.loc[mask, 'value'],
    s=25,
    label='Valid Data (.**)'  # QC flag starts with a dot
)

# Plot rejected points (first character is not a dot)
plt.scatter(
    flask_data.loc[~mask, 'datetime'],
    flask_data.loc[~mask, 'value'],
    s=25,
    color='red',
    label='Rejected Data (*..)'
)

# Format axes
plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO₂ at Mauna Loa Observatory: Full Dataset")
plt.xlabel('Year')
plt.ylabel('CO₂ mole fraction (ppm)')
plt.legend()
plt.show()
```

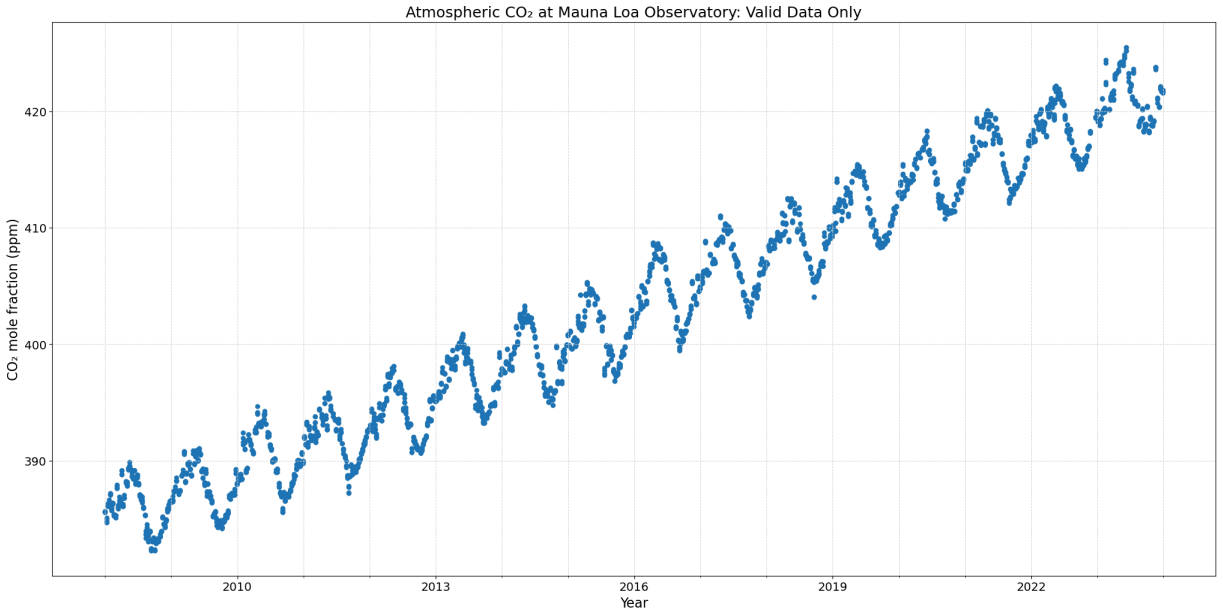


Now Plot Only the Valid Data

Filtering out rejected and missing values reveals clearer patterns in the time series.

> *Note: The range of CO_2 mole fraction values decreases significantly - from a spread of ~300 ppm to a much cleaner ~50 ppm.*

In [51]:
```
# Plot valid data only
plt.figure(figsize=(25, 12))

plt.scatter(good_flask_data['datetime'], good_flask_data['value'], s=25)

# Format axes
plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO₂ at Mauna Loa Observatory: Valid Data Only")
plt.xlabel('Year')
plt.ylabel('CO₂ mole fraction (ppm)')
plt.show()
```



Additional Data Exploration

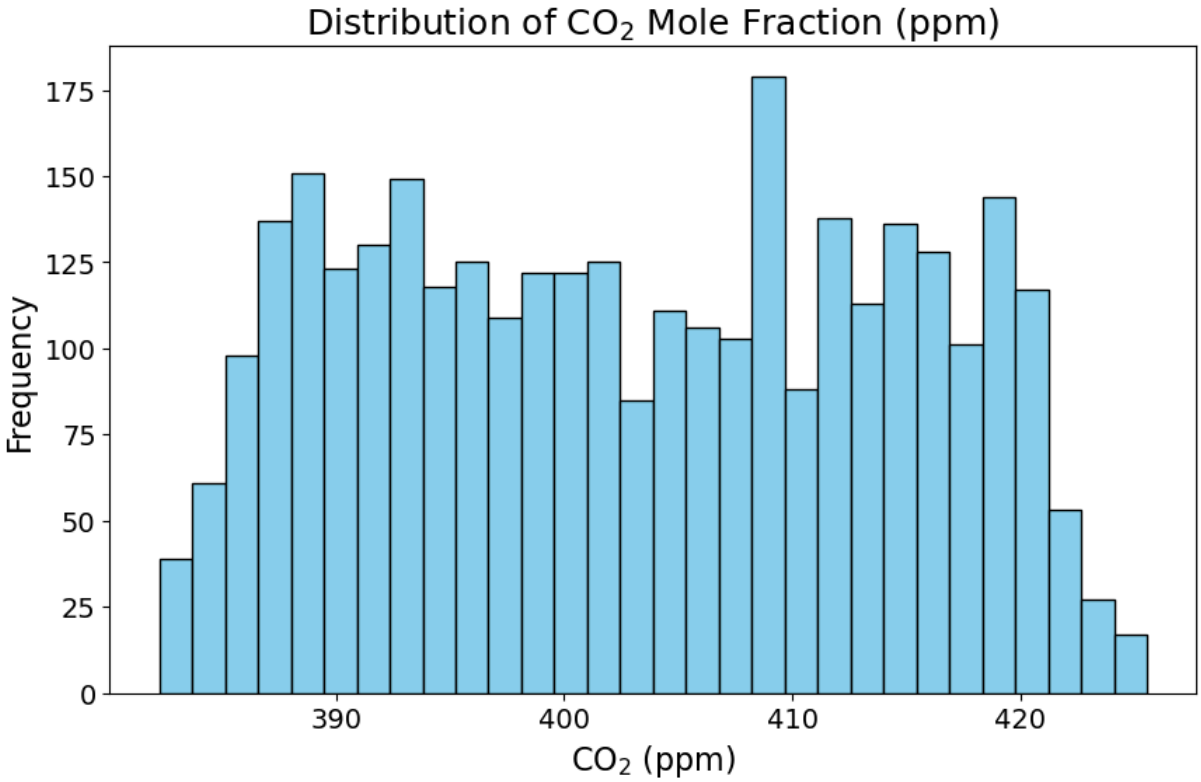
To deepen our understanding, let's calculate some summary statistics on the filtered dataset:

In [52]:
```
print("Summary statistics for CO₂ levels 2008 – 2023 (valid data only):")
print(good_flask_data['value'].describe())
```

```
Summary statistics for CO₂ levels 2008 – 2023 (valid data only):
count    3255.000000
mean      403.052575
std        11.256475
min       382.260000
25%       393.160000
50%       402.850000
75%       412.905000
max       425.540000
Name: value, dtype: float64
```

We can also visualize the distribution of CO_2 measurements using a histogram:

In [53]:
```
plt.figure(figsize=(10, 6))
plt.hist(good_flask_data['value'], bins=30, color='skyblue', edgecolor='blac
plt.title("Distribution of CO$_2$ Mole Fraction (ppm)")
plt.xlabel("CO$_2$ (ppm)")
plt.ylabel("Frequency")
plt.show()
```



Step 5: Apply NOAA GML's Curve Fitting Method

This method helps us to extract important patterns in the CO_2 data:

- A long-term trend (overall rise in CO_2)
- A yearly seasonal cycle (non-sinusoidal oscillations)
- Short-term variations

This technique is based on the work by Thoning et al. (1989) and NOAA GML's implementation. For more details and to follow along with the full tutorial, visit: https://gml.noaa.gov/ccgg/mbl/crvfit/crvfit.html

In [54]:
```
# Set x and y data for the filter
xp = good_flask_data['time_decimal']
yp = good_flask_data['value']

# Create filter object using NOAA GML's curve fitting class
filt = ccg_filter.ccgFilter(
    xp,  # Time values in decimal years
    yp,  # CO2 mole fraction values
    shortterm=80,        # Short term smoothing cutoff in days
    longterm=667,        # Long term trend cutoff in days
    # Interval for evenly spaced data (0 means use original spacing)
    sampleinterval=0,
    numpolyterms=3,      # Polynomial terms (quadratic)
    numharmonics=4,      # Number of harmonics for seasonal cycle
    timezero=-1,         # Reference time zero for polynomial fitting
    gap=0,               # No gap-filling in interpolation
    use_gain_factor=False,  # Disable gain factor in harmonics
    debug=False          # Disable debug messages
)
```

We can now extract components of the fitted model for plotting:

- `x0` : equally spaced time points interpolated from xp
- `y1` : function fit value at time x
- `y2` : polynomial fit value at time x
- `y3` : smoothed data value at time x
- `y4` : trend at time x

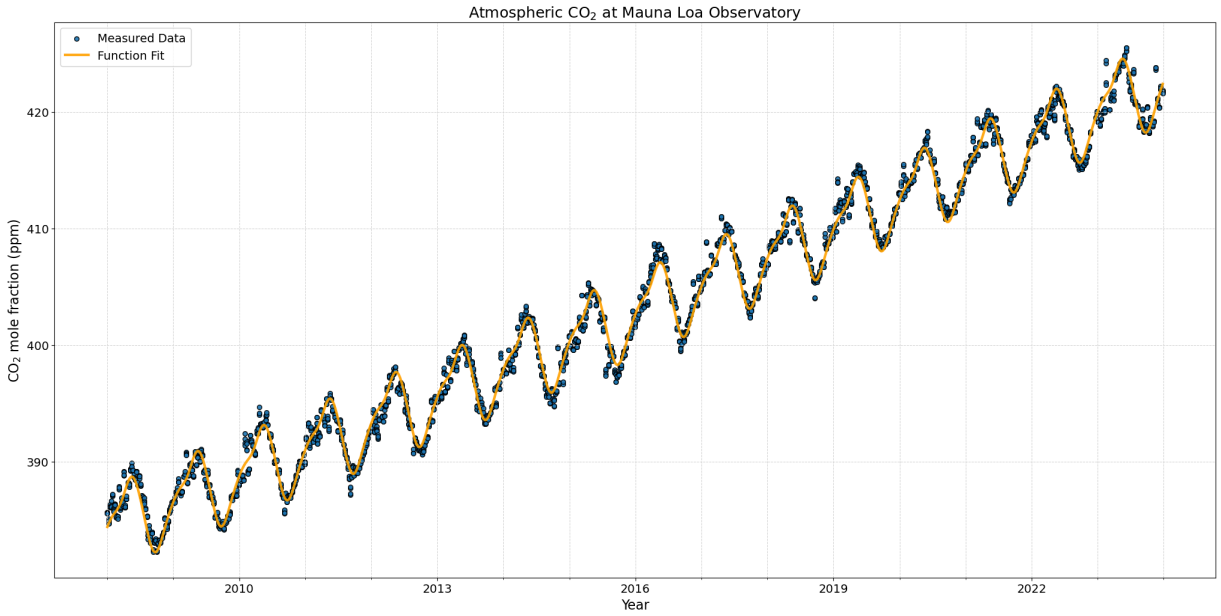
In [55]:
```
# Extract components from the filter
x0 = filt.xinterp
y1 = filt.getFunctionValue(x0)
y2 = filt.getPolyValue(x0)
y3 = filt.getSmoothValue(x0)
y4 = filt.getTrendValue(x0)

# Convert decimal dates to datetime for plotting
x0_dates = [ccg_dates.datetimeFromDecimalDate(i) for i in x0]
xp_dates = [ccg_dates.datetimeFromDecimalDate(i) for i in xp]
```

We can plot various aspects of the `filt` object to visualize the different components: a long-term trend, a non-sinusoidal yearly cycle, and short term variations.

Plot 1: Function Fit vs. Measured Data

In [56]:
```
plt.figure(figsize=(25, 12))
plt.scatter(xp_dates, yp, s=30, alpha=0.9,
            edgecolors='black', label="Measured Data")
```

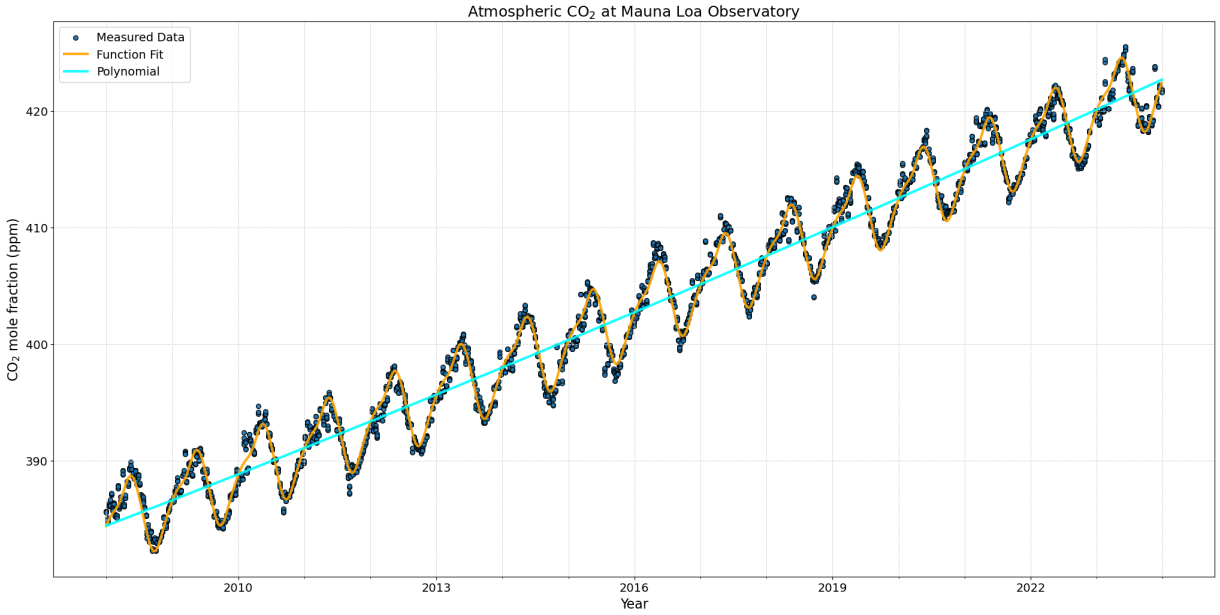
```
plt.plot(x0_dates, y1, color='orange', linewidth=3,
         alpha=0.9, label="Function Fit")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))  # Minor ticks ever
plt.gca().xaxis.set_major_locator(
    mdates.YearLocator(3))  # Major ticks every 3 years
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('CO$_2$ mole fraction (ppm)')
plt.legend()
plt.show()
```



Plot 2: Polynomial and Function Fit vs. Measured Data

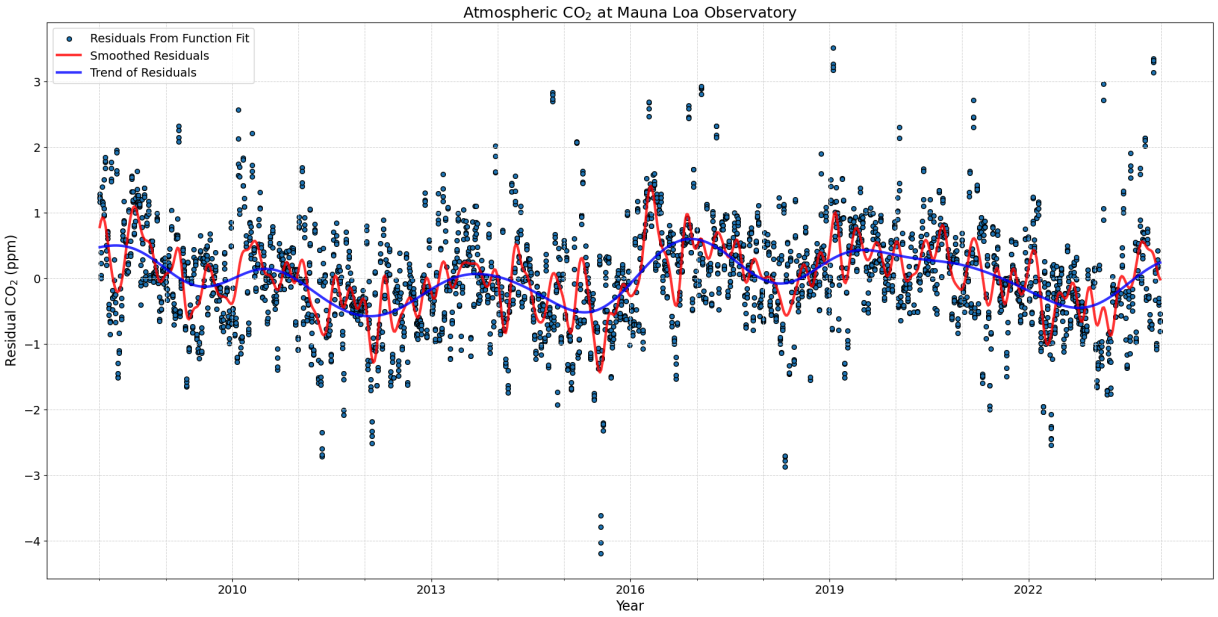
In [57]:
```
plt.figure(figsize=(25, 12))
plt.scatter(xp_dates, yp, s=30, alpha=0.9,
            edgecolors='black', label="Measured Data")
plt.plot(x0_dates, y1, color='orange', linewidth=3,
         alpha=0.9, label="Function Fit")
plt.plot(x0_dates, y2, color='cyan', linewidth=3,
         alpha=0.9, label="Polynomial")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')
```

```
plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('CO$_2$ mole fraction (ppm)')
plt.legend()
plt.show()
```



Plot 3: Residuals from Function Fit, Smoothed Residuals, and Trend Components

In [58]:
```
# Residuals from function, smoothed residuals, and trend of residuals
resid_from_func = filt.resid
resid_smooth = filt.smooth
resid_trend = filt.trend

plt.figure(figsize=(25, 12))

plt.scatter(xp_dates, resid_from_func, s=30, edgecolors='black',
            label="Residuals From Function Fit")
plt.plot(x0_dates, resid_smooth, c='red', linewidth=3,
         alpha=0.8, label="Smoothed Residuals")
plt.plot(x0_dates, resid_trend, c='blue', linewidth=3,
         alpha=0.8, label="Trend of Residuals")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('Residual CO$_2$ (ppm)')
plt.legend()
plt.show()
```



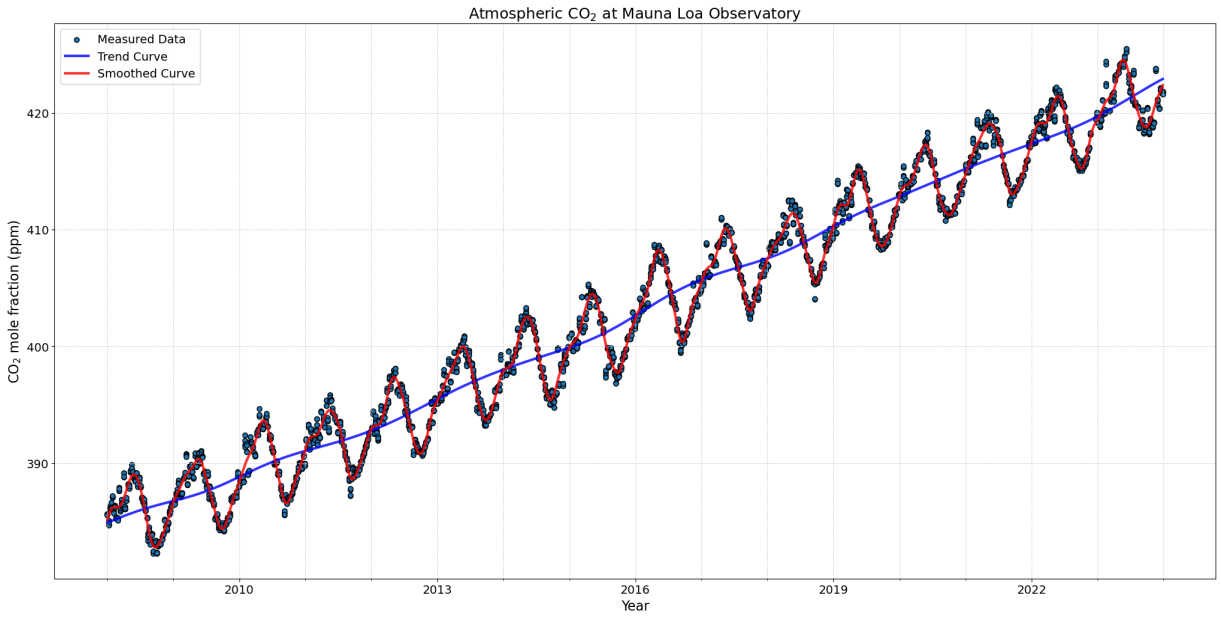
At this stage, the curve fitting process is complete. Now, we combine different components of the fitted function and filtered data to isolate the specific signals we're interested in analyzing. The key components are:

- **Smoothed Data**: Represents the data with only short-term fluctuations removed. It combines the fitted function with residuals filtered using the short-term cutoff.
- **Trend**: Captures the long-term upward movement in CO_2 levels after removing seasonal cycles. This includes the polynomial portion of the fit plus residuals filtered with the long-term cutoff.
- **Detrended Seasonal Cycle**: Shows the yearly oscillation after removing the overall trend. This consists of the harmonic (seasonal) part of the fit combined with short-term filtered residuals.
- **Seasonal Amplitude**: Measures the size of the seasonal cycle, calculated as the difference between peak and trough in the detrended seasonal cycle.
- **Growth Rate**: The rate at which the long-term trend increases, found by taking the first derivative of the trend component.

Plot 4: Measured Data with Smoothed and Trend Curves

In [59]:
```
trend = filt.getTrendValue(xp)

plt.figure(figsize=(25, 12))

plt.scatter(xp_dates, yp, s=35, edgecolors='black', label="Measured Data")
plt.plot(xp_dates, trend, c='blue', linewidth=3,
         alpha=0.8, label="Trend Curve")
plt.plot(x0_dates, y3, c='red', linewidth=3, alpha=0.8, label="Smoothed Curv

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
```

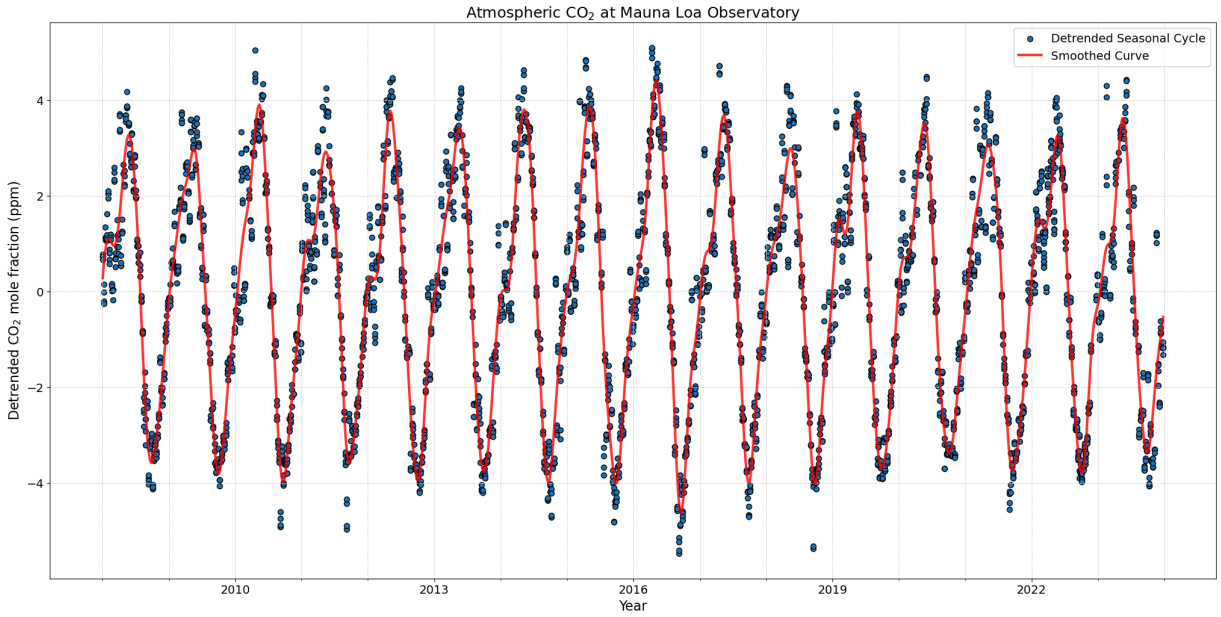
```
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('CO$_2$ mole fraction (ppm)')
plt.legend()
plt.show()
```



Plot 5: Smoothed Seasonal Cycle

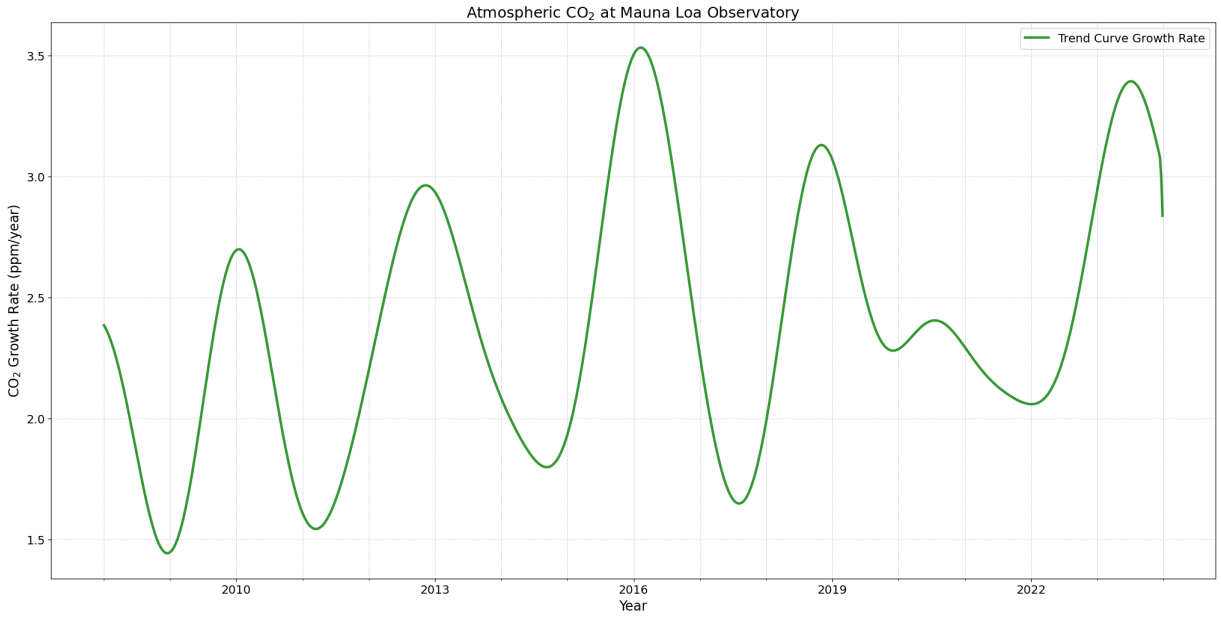
In [60]:
```
detrend = yp - trend
harmonics = filt.getHarmonicValue(x0)
smooth_cycle = harmonics + filt.smooth - filt.trend

plt.figure(figsize=(25, 12))

plt.scatter(xp_dates, detrend, s=40, edgecolors='black',
            label="Detrended Seasonal Cycle")
plt.plot(x0_dates, smooth_cycle, c='red', linewidth=3,
         alpha=0.8, label="Smoothed Curve")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
```

```
plt.ylabel('Detrended CO$_2$ mole fraction (ppm)')
plt.legend()
plt.show()
```



Plot 6: Growth Rate of the Trend Curve

In [61]:
```
growth_rate = filt.getGrowthRateValue(xp)

plt.figure(figsize=(25, 12))

plt.plot(xp_dates, growth_rate, c='green', linewidth=3,
         alpha=0.8, label="Trend Curve Growth Rate")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('CO$_2$ Growth Rate (ppm/year)')
plt.legend()
plt.show()
```



Step 6: Monthly Means

Using monthly means computed by the filter, we can create a DataFrame and plot to visualize monthly variations.

In [62]:
```
# Get monthly means (year, month, avg, sd, n)
mm = filt.getMonthlyMeans()

mm_df = pd.DataFrame(mm, columns=['year', 'month', 'avg', 'sd', 'n'])

# Convert year + month to datetime for plotting
mm_df['date'] = pd.to_datetime(mm_df[['year', 'month']].assign(day=1))

mm_df.head()
```

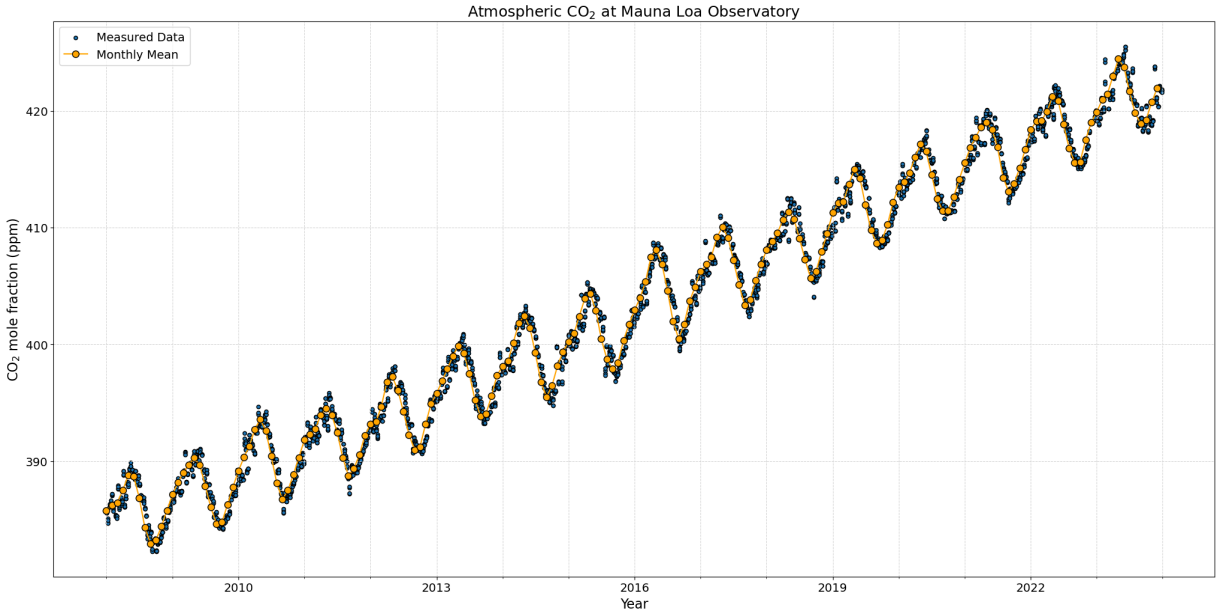
Out[62]:

	year	month	avg	sd	n	date
0	2008	1	385.747075	0.404402	5	2008-01-01
1	2008	2	386.222799	0.009245	4	2008-02-01
2	2008	3	386.437696	0.200889	4	2008-03-01
3	2008	4	387.497973	0.451456	4	2008-04-01
4	2008	5	388.789741	0.281445	5	2008-05-01

Plot 7: Monthly Means with Measured Data

In [63]:
```
plt.figure(figsize=(25, 12))

plt.scatter(xp_dates, yp, s=20, edgecolors='black', label="Measured Data")
plt.plot(mm_df['date'], mm_df['avg'], linestyle='-', marker='o', markersize=
         color='orange', markeredgecolor="black", label="Monthly Mean")
```

```
plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory")
plt.xlabel('Year')
plt.ylabel('CO$_2$ mole fraction (ppm)')
plt.legend()
plt.show()
```



Summary and Takeaways

In this notebook, you learned how to work with real atmospheric CO_2 data from Mauna Loa Observatory. You practiced loading and exploring data, filtering out invalid measurements, and visualizing CO_2 levels over time.

Then, you applied NOAA's curve fitting method to separate long-term trends from seasonal and short-term variations in the data. This technique helps scientists better understand how CO_2 is changing in the atmosphere.

By completing these steps, you gained hands-on experience with data cleaning, time series analysis, and advanced curve fitting - all important skills for studying environmental data.

Keep exploring and experimenting with these tools to deepen your understanding of atmospheric science and data analysis!

Appendix: Loading CO_2 Data (NetCDF Format)

In addition to the text file format, NOAA provides atmospheric CO_2 flask measurements in **NetCDF format** — a widely-used format for storing structured scientific data.

This section shows how to:

- Open the NetCDF file using the `xarray` package
- View metadata and available variables
- Convert selected variables into a `pandas.DataFrame`
- Use the resulting DataFrame with the rest of this tutorial

> *This is optional. The NetCDF and text files contain the same data. You only need to load one.*

In [64]:
```
# *Note: You may need to install the package netcdf4, which is a dependency

# if necessary, uncomment the following line and run this cell to install
# pip install netcdf4
```

In [65]:
```
# import required libraries
import xarray as xr
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
```

In [66]:
```
# Path to your NetCDF file
file_path = "co2_mlo_surface-flask_1_ccgg_event.nc"

# Open dataset
ds = xr.open_dataset(file_path)

# View dataset structure
ds
```

Out[66]: xarray.Dataset

▶ Dimensions: (obs: 10972, calendar_components: 6, dim_concerns: 6)

▶ Coordinates: (0)

▶ Data variables: (25)

▶ Indexes: (0)

▶ Attributes: (57)

> The unique format of the NetCDF file allows the user to view both the metadata and dataset variables with only the command above - no need for extra print statements.

We'll extract a subset of useful fields to create a structured DataFrame for analysis comparable to that performed above:

In [67]:
```
# Variables to include in the DataFrame
variables_to_use = [
    'datetime', 'time_decimal', 'air_sample_container_id', 'value',
    'value_unc', 'latitude', 'longitude', 'altitude', 'elevation',
    'intake_height', 'method', 'event_number', 'instrument',
    'analysis_datetime', 'qcflag'
]

# Create the DataFrame
flask_data_nc = ds[variables_to_use].to_dataframe().reset_index()

# Decode all byte string columns in flask_data_nc
for col in flask_data_nc.columns:
    if flask_data_nc[col].dtype == object and isinstance(flask_data_nc[col].
        flask_data_nc[col] = flask_data_nc[col].str.decode('utf-8')


# Preview the data
flask_data_nc.head()
```

Out[67]:

	obs	datetime	time_decimal	air_sample_container_id	value	value_unc
0	0	1969-08-20T17:55:00Z	1969.634922	33-69	323.170013	0.531
1	1	1969-08-20T17:55:00Z	1969.634922	34-69	324.720001	0.531
2	2	1969-08-20T18:30:00Z	1969.634989	31-69	331.019989	0.531
3	3	1969-08-20T18:30:00Z	1969.634989	32-69	-999.989990	NaN
4	4	1969-08-27T19:15:00Z	1969.654252	35-69	-999.989990	NaN

You can now use the `flask_data_nc` DataFrame (created from the NetCDF file) just like the text-based version. The structure is the same, so the rest of the analysis will work in the same way.

So, continue with filtering, plotting, and curve fitting just as described in steps 3-5 above:

Step 3(b): Explore the NetCDF DataFrame

In [68]:
```
# Column names
flask_data_nc.columns
```

Out[68]:
```
Index(['obs', 'datetime', 'time_decimal', 'air_sample_container_id', 'valu
e',
       'value_unc', 'latitude', 'longitude', 'altitude', 'elevation',
       'intake_height', 'method', 'event_number', 'instrument',
       'analysis_datetime', 'qcflag'],
      dtype='object')
```

In [69]:
```
# Dimensions of the DataFrame: (rows, columns)
flask_data_nc.shape
```

Out[69]:
```
(10972, 16)
```

In [70]:
```
# Data types of each column
flask_data_nc.dtypes
```

Out[70]:
```
obs                       int64
datetime                 object
time_decimal            float64
air_sample_container_id  object
value                   float32
value_unc               float32
latitude                float32
longitude               float32
altitude                float32
elevation               float32
intake_height           float32
method                   object
event_number             object
instrument               object
analysis_datetime        object
qcflag                   object
dtype: object
```

In [71]:
```
# Convert the 'datetime' column to datetime format (if it isn't already)
flask_data_nc['datetime'] = pd.to_datetime(flask_data_nc['datetime'])
flask_data_nc.dtypes
```

Out[71]:
```
obs                                    int64
datetime                  datetime64[ns, UTC]
time_decimal                          float64
air_sample_container_id                object
value                                 float32
value_unc                             float32
latitude                              float32
longitude                             float32
altitude                              float32
elevation                             float32
intake_height                         float32
method                                 object
event_number                           object
instrument                             object
analysis_datetime                      object
qcflag                                 object
dtype: object
```

Step 4(b): Filter and Visualize the CO_2 Data

We'll remove any rows that:

- Have a `qcflag` starting with any character other than `"."`
- Are not in the time range of 2008 - 2023

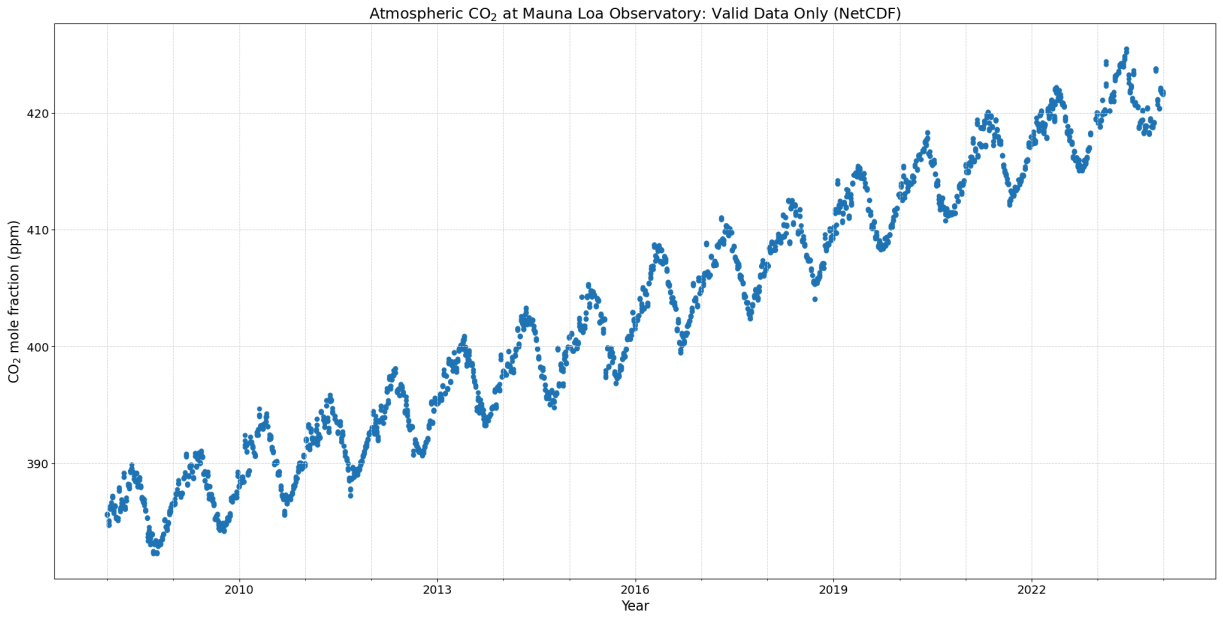
For simplicity, we will only include the plot of the filtered, valid CO_2 data points. The process of filtering to remove invalid or questionable data works the same way regardless of the data source, whether from the text file or the NetCDF file.

In [72]:
```
# Filter to years 2008-2023
flask_data_nc = flask_data_nc[(flask_data_nc['datetime'].dt.year >= 2008) &
                              (flask_data_nc['datetime'].dt.year <= 2023)]

# Keep only entries where the first character of the qcflag is "."
good_nc_data = flask_data_nc[flask_data_nc['qcflag'].str.match(
    r'^\..*')].reset_index(drop=True)

good_nc_data.shape
```

Out[72]:
```
(3255, 16)
```

In [73]:
```
plt.figure(figsize=(25, 12))

plt.scatter(good_nc_data['datetime'], good_nc_data['value'], s=25)

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')
```

```
plt.title("Atmospheric CO$_2$ at Mauna Loa Observatory: Valid Data Only (Net
plt.xlabel('Year')
plt.ylabel('CO$_2$ mole fraction (ppm)')
plt.show()
```



Step 5(b): Apply Curve Fitting to NetCDF Data

In this step, we begin the process of curve fitting and extracting the signal components from the CO_2 data.

Since the NetCDF data structure can vary slightly, we'll demonstrate the first few cells here. The rest of the analysis follows the same process as with the text-based dataset, so you can apply those steps directly once the initial data preparation is done.

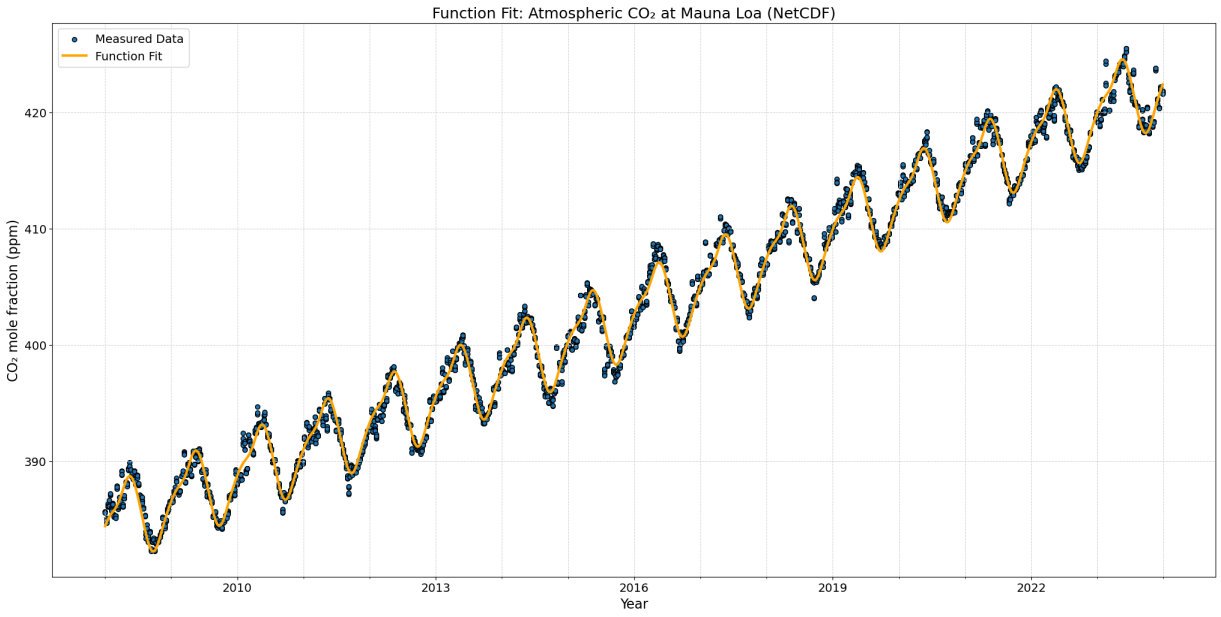
In [74]:
```
xp = good_nc_data['time_decimal']
yp = good_nc_data['value']

filt = ccg_filter.ccgFilter(xp, yp,
                            shortterm=80,
                            longterm=667,
                            sampleinterval=0,
                            numpolyterms=3,
                            numharmonics=4,
                            timezero=-1,
                            gap=0,
                            use_gain_factor=False,
                            debug=False)
```

In [75]:
```
# Get time and curve components
x0 = filt.xinterp
y1 = filt.getFunctionValue(x0)
y2 = filt.getPolyValue(x0)
y3 = filt.getSmoothValue(x0)
y4 = filt.getTrendValue(x0)
```

```
# Convert to datetime for plotting
x0_dates = [ccg_dates.datetimeFromDecimalDate(i) for i in x0]
xp_dates = [ccg_dates.datetimeFromDecimalDate(i) for i in xp]
```

Plot Function Fit with Flask Data

In [76]:
```
plt.figure(figsize=(25, 12))
plt.scatter(xp_dates, yp, s=30, edgecolors='black', label="Measured Data")
plt.plot(x0_dates, y1, color='orange', linewidth=3, label="Function Fit")

plt.gca().xaxis.set_minor_locator(mdates.YearLocator(1))
plt.gca().xaxis.set_major_locator(mdates.YearLocator(3))
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.grid(True, which='both', axis='x', linestyle='--',
         linewidth=0.7, color='lightgray')
plt.grid(True, which='major', axis='y', linestyle='--',
         linewidth=0.7, color='lightgray')

plt.title("Function Fit: Atmospheric CO₂ at Mauna Loa (NetCDF)")
plt.xlabel("Year")
plt.ylabel("CO₂ mole fraction (ppm)")
plt.legend()
plt.show()
```



This appendix demonstrated how to load and prepare atmospheric CO_2 data from a NetCDF file using the `xarray` and `pandas` libraries. While the core dataset is the same as the text version, some formatting steps, like converting byte strings, are unique to NetCDF.

Once processed, the NetCDF data can be used in exactly the same way as the text-based data for filtering, curve fitting, and visualization. This flexibility allows you to work with whichever format best suits your needs or workflow.