Can you predict the future..?

Gaussian Process Modelling for Forward Prediction

Anna Scaife[1]

[1] Jodrell Bank Centre for Astrophysics
University of Manchester

@radastrat

September 6, 2017

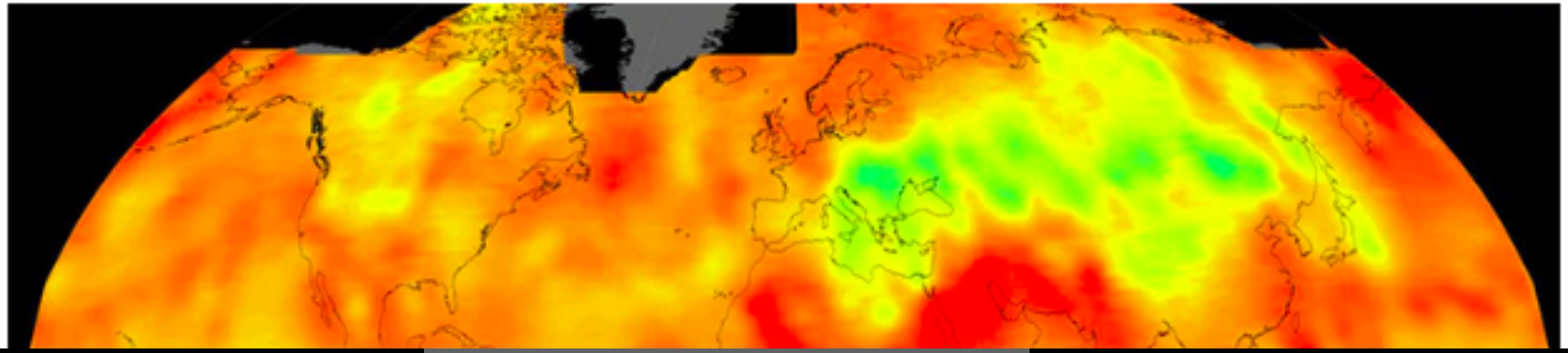
THE SPACE REPORTER

HOME | NASA | SOLAR SYSTEM | ASTRONOMY | INTERNATIONAL SPACE STATION | PRIV

News Ticker March 28, 2017 in Astronomy: **Astronomers find most massive and 'pure' brown dwarf**

Earth's atmospheric CO2 reaches record high levels

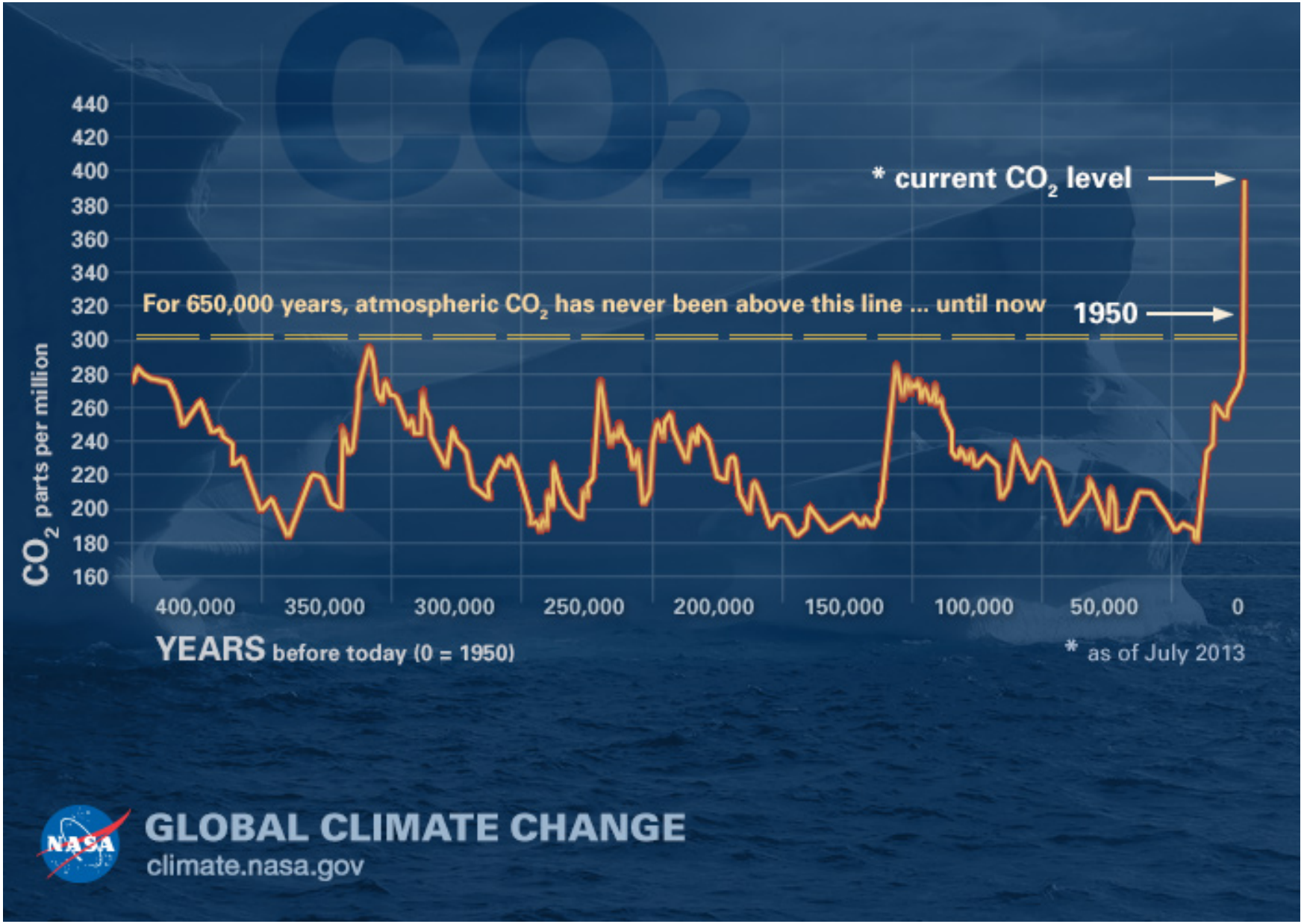
Posted on June 15, 2016 by Laurel Kornfeld



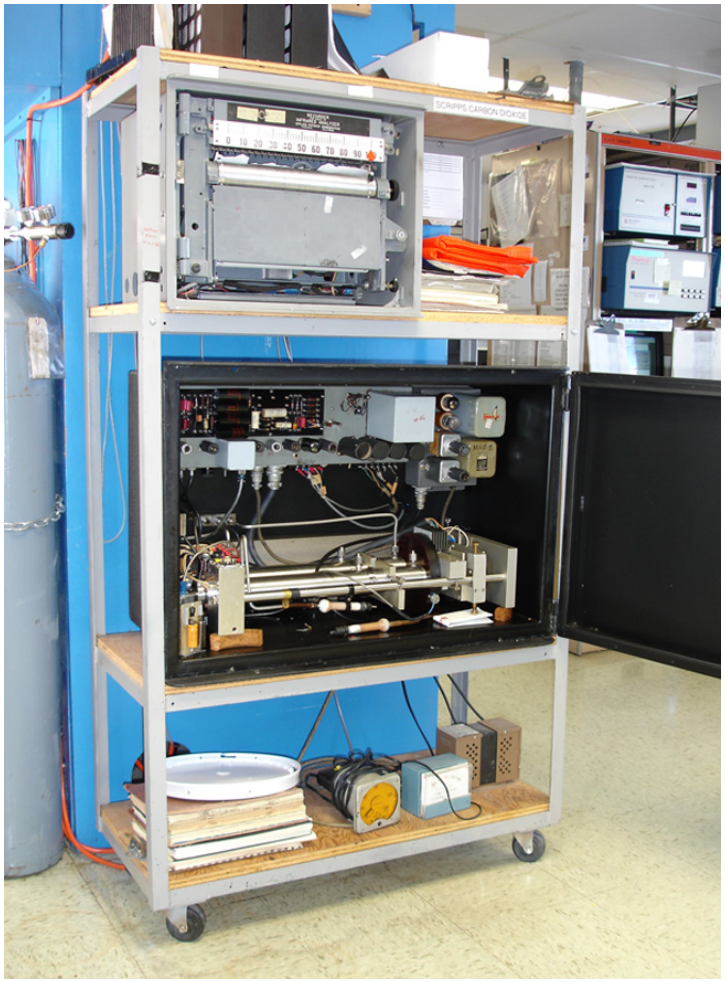

sign in
become a supporter
subscribe
search
jobs
dating
more
UK edition
theguardian
UK
world
politics
sport
football
opinion
culture
business
lifestyle
fashion
environment
tech
travel
all sections
home › environment › climate change
wildlife
energy
pollution
Climate change
Guardian Environment Network
The world passes 400ppm carbon dioxide threshold. Permanently
We are now living in a 400ppm world with levels unlikely to drop below the symbolic milestone in our lifetimes, say scientists. Climate Central reports

In May 2016, the carbon dioxide content of the Earth's atmosphere hit a value of 400 parts per million.

Last time it was at that level, human life did not exist.


440
420
400
380
360
340
320
300
280
260
240
220
200
180
160
CO_2 parts per million
* current CO_2 level
For 650,000 years, atmospheric CO_2 has never been above this line ... until now
1950
400,000
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0
YEARS before today (0 = 1950)
* as of July 2013
GLOBAL CLIMATE CHANGE
climate.nasa.gov

Measuring CO_2



www.esrl.noaa.gov/gmd/ccgg/about/co2_measurements.html

TIME

Ecocentric
All things green, from conservation to Capitol Hill

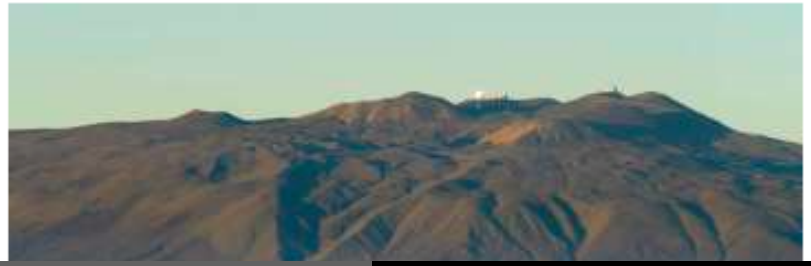
CLIMATE SCIENCE

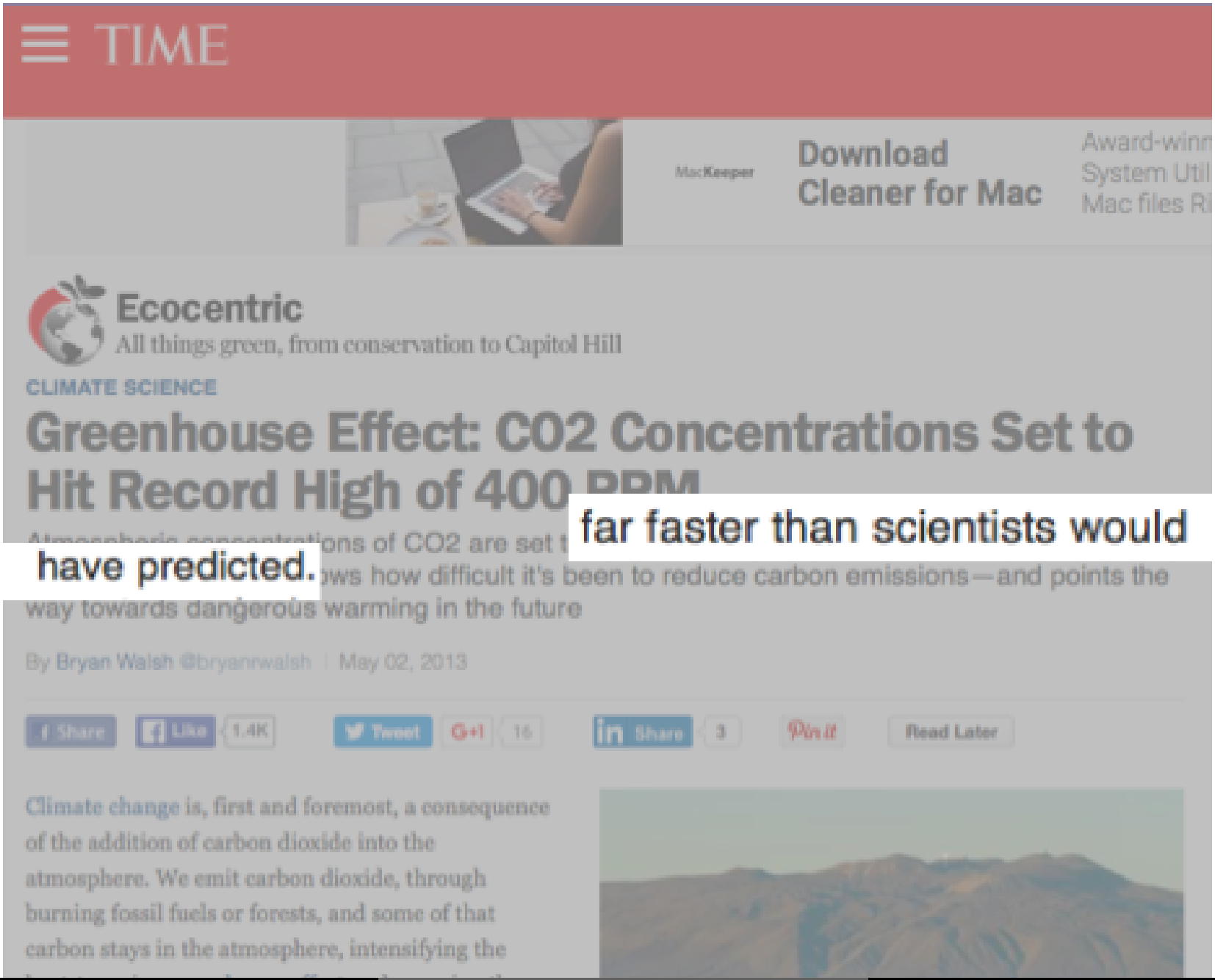
Greenhouse Effect: CO2 Concentrations Set to Hit Record High of 400 PPM

Atmospheric concentrations of CO2 are set to pass 400 ppm, far faster than scientists would have predicted. That shows how difficult it's been to reduce carbon emissions—and points the way towards dangerous warming in the future

By Bryan Walsh @bryanrwalsh | May 02, 2013

Climate change is, first and foremost, a consequence of the addition of carbon dioxide into the atmosphere. We emit carbon dioxide, through burning fossil fuels or forests, and some of that carbon stays in the atmosphere, intensifying the




TIME
Download
Cleaner for Mac
Ecocentric
All things green, from conservation to Capitol Hill
CLIMATE SCIENCE
Greenhouse Effect: CO2 Concentrations Set to
Hit Record High of 400 PPM
far faster than scientists would
have predicted.
ons of CO2 are set t
ows how difficult it's been to reduce carbon emissions—and points the
way towards dangerous warming in the future
By Bryan Walsh @bryanrwalsh | May 02, 2013
Share
Like
1.4K
Tweet
G+1
16
Share
3
Pin it
Read Later
Climate change is, first and foremost, a consequence
of the addition of carbon dioxide into the
atmosphere. We emit carbon dioxide, through
burning fossil fuels or forests, and some of that
carbon stays in the atmosphere, intensifying the

The increase has been caused by people, particularly the burning of fossil fuels and deforestation.

So how do you **quantitatively** predict a value at any given point? You'd need to construct a detailed mathematical model that accounted for:

- population
- fossil fuel burning
- deforestation

The increase has been caused by people, particularly the burning of fossil fuels and deforestation.

So how do you **quantitatively** predict a value at any given point? You'd need to construct a detailed mathematical model that accounted for:

- population
- fossil fuel burning
- deforestation
- global photosynthesis cycle
- oceanic carbon cycle
-
-
- unknown factors...

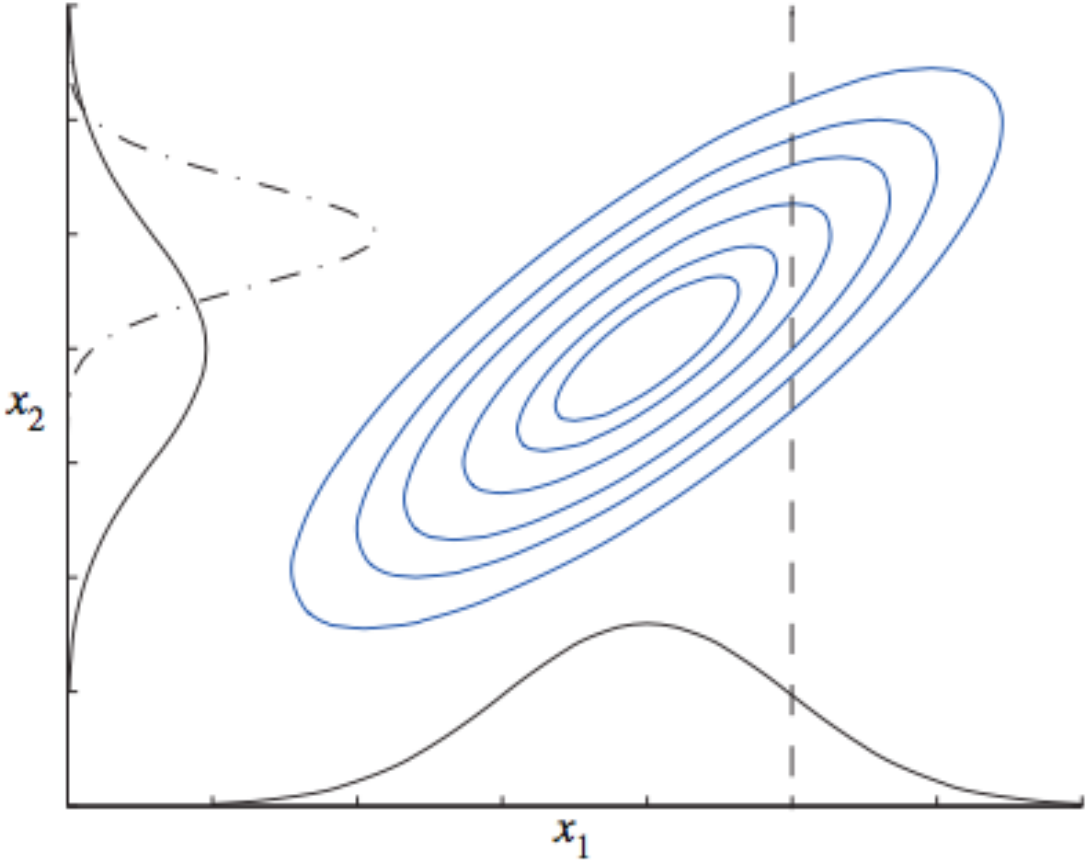
The increase has been caused by people, particularly the burning of fossil fuels and deforestation.

So how do you **quantitatively** predict a value at any given point? You'd need to construct a detailed mathematical model that accounted for:

- population
- fossil fuel burning
- deforestation
- global photosynthesis cycle
- oceanic carbon cycle
-
-
- unknown factors...

But... what if you didn't need to model all of those factors? What if you could predict the future pattern of behaviour just based on previously measured data?

Covariance



Numpy Multivariate Normal Distribution

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.12 Manual | NumPy Reference | Routines | Random sampling (`numpy.random`) | index | next | previous

numpy.random.multivariate_normal

numpy.random.multivariate_normal(*mean, cov*[, *size*])

Draw random samples from a multivariate normal distribution.

The multivariate normal, multinormal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its mean and covariance matrix. These parameters are analogous to the mean (average or "center") and variance (standard deviation, or "width," squared) of the one-dimensional normal distribution.

Parameters:

mean : *1-D array_like, of length N*
Mean of the N-dimensional distribution.

cov : *2-D array_like, of shape (N, N)*
Covariance matrix of the distribution. It must be symmetric and positive-semidefinite for proper sampling.

size : *int or tuple of ints, optional*
Given a shape of, for example, `(m,n,k)`, `m*n*k` samples are generated, and packed in an *m*-by-*n*-by-*k* arrangement. Because each sample is *N*-dimensional, the output shape is `(m,n,k,N)`. If no shape is specified, a single (*N*-D) sample is returned.

Returns:

out : *ndarray*
The drawn samples, of shape *size*, if that was provided. If not, the shape is `(N,)`.
In other words, each entry `out[i,j,...,:]` is an N-dimensional value drawn from the distribution.

Previous topic
numpy.random.multinomial

Next topic
numpy.random.negative_binomial

Multi-variate Normal Example

```
# make an array of positions
# these are evenly spaced, but they don't have to be
x = np.arange(0, 20.,0.01)

# use numpy to draw a sample from the multi-variate
# normal distribution, N(0,K), at all the positions in x
y = np.random.multivariate_normal(np.zeros(len(x)),K)
```

Covariance Matrix

For a dataset with n data points, the covariance matrix will be $n \times n$ in size, because it covers every **pair** of points twice, i.e. x_1, x_2 and x_2, x_1.

$$\mathbf{K}(\mathbf{x}, \mathbf{x}) = \begin{pmatrix} k(x_1, x_1) & k(x_1, x_2) & \dots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \dots & k(x_2, x_n) \\ \vdots & \vdots & \vdots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \dots & k(x_n, x_n) \end{pmatrix} \tag{1}$$

The value of each matrix element is set by a function called the covariance kernel, $k(x_m, x_n)$, which needs to be determined.

If we think that there will also be measurement noise on each data point that is **uncorrelated** between the data points then we can add that on:

$$\mathbf{V}(\mathbf{x}, \mathbf{x}) = \mathbf{K}(\mathbf{x}, \mathbf{x}) + \sigma^2 \mathbf{I}. \tag{2}$$

Random Events

Let's use a **Squared Exponential Kernel** (otherwise known as a Gaussian kernel):

$$k(x_1, x_2) = h^2 \exp\left(\frac{-(x_1 - x_2)^2}{\lambda^2}\right) \tag{3}$$

```
def cov_kernel(x1,x2,h,lam):

    """
    Squared-Exponential covariance kernel
    """

    k12 = h**2*np.exp(-1.*(x1 - x2)**2/lam**2)

    return k12
```

Hyper-parameters

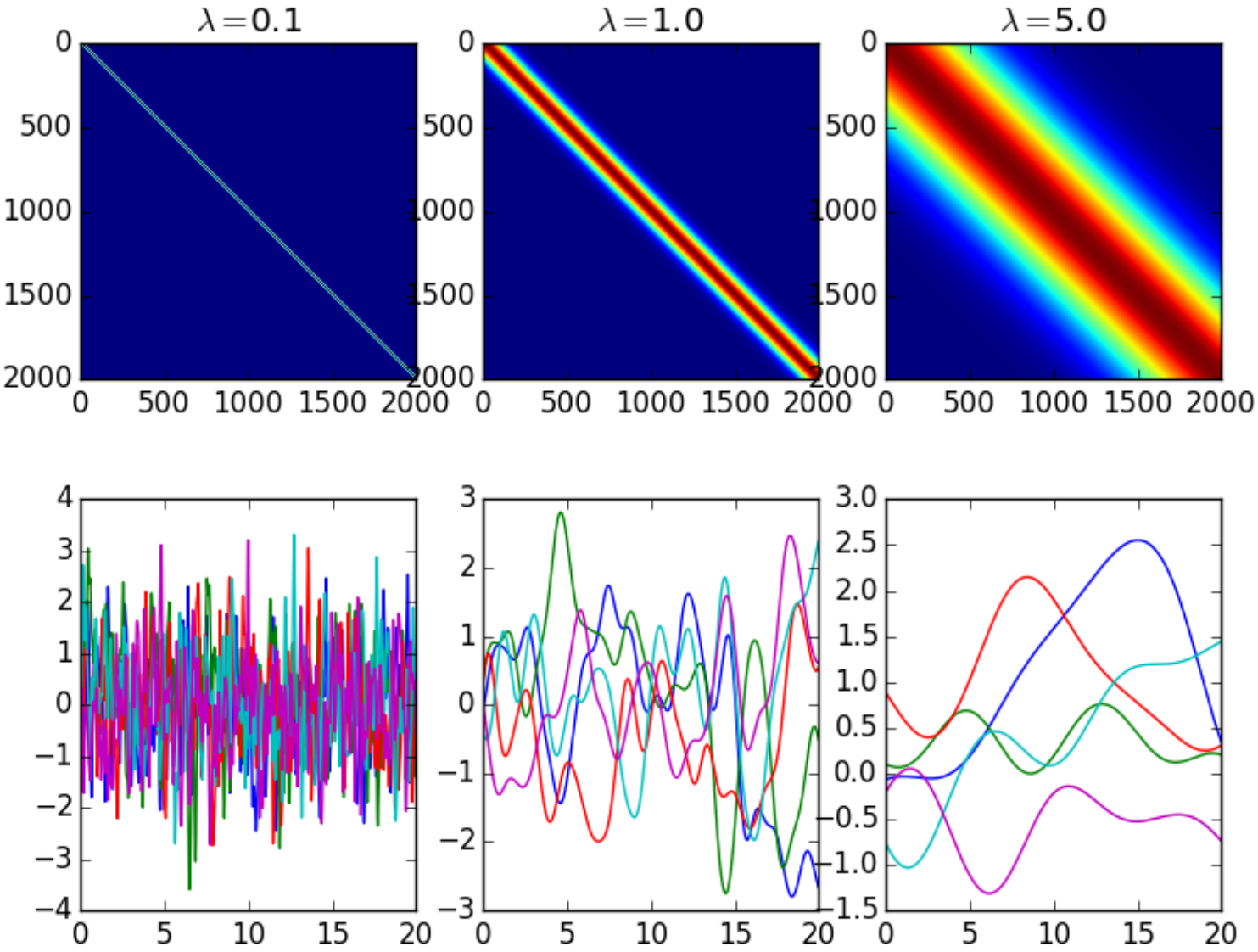
h and λ are called hyper-parameters

They are not "parameters" because they do not represent anything physical about the data itself. They only parameterise the covariance.

```python
def make_K(x, h, lam):

    """
    Make covariance matrix from covariance kernel
    """

    # for a data array of length x, make a covariance matrix x*x:
    K = np.zeros((len(x),len(x)))

    for i in range(0,len(x)):
        for j in range(0,len(x)):

            # calculate value of K for each separation:
            K[i,j] = cov_kernel(x[i],x[j],h,lam)

    return K
```

Once we have a covariance matrix we can start drawing samples from it using the **numpy** library:

```
# make an array of 200 evenly spaced positions between 0 and 20:
x1 = np.arange(0, 20.,0.01)

# make a covariance matrix:
K = make_K(x1,h,lam)

# draw samples from a co-variate Gaussian
# distribution, N(0,K), at positions x1:
y1 = np.random.multivariate_normal(np.zeros(len(x1)),K)
```


λ=0.1
λ=1.0
λ=5.0
0
500
1000
1500
2000
0 500 1000 1500 2000
4
3
2
1
0
−1
−2
−3
−4
0 5 10 15 20
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

References

- Gaussian Processes for Machine Learning, Carl Edward Rasmussen and Chris Williams, the MIT Press (`www.gaussianprocess.org`)
- Gaussian processes for time-series modelling, S. Roberts, M. Osborne, M. Ebden, S. Reece, N. Gibson and S. Aigrain, Phil. Trans. R. Soc. A 2013 371, 20110550 (`www.robots.ox.ac.uk/~sjrob/Pubs/philTransA_2012.pdf`)

Predictions

To calculate the value of a hypothetical measurement at another position, x_*, we can use the following equations. These give us the mean ($\mathbf{m}_*$) and the variance ($\mathbf{C}_*$) at that point, i.e. the value and the uncertainty on that value.

$$\mathbf{m}_* = \mathbf{K}(\mathbf{x}_*, \mathbf{x})^{\mathbf{T}}\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{y} \tag{4}$$

$$\mathbf{C}_* = \mathbf{K}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{K}(\mathbf{x}_*, \mathbf{x})^{\mathbf{T}}\mathbf{K}(\mathbf{x}, \mathbf{x})^{-1}\mathbf{K}(\mathbf{x}_*, \mathbf{x}) \tag{5}$$

See Chapter 2 of Rasmussen & Williams to see where this comes from in more detail.

`www.gaussianprocess.org/gpml/chapters/RW2.pdf`

Predictions - Zero Mean

To calculate the value of a hypothetical measurement at another position, x_*, we can use the following equations. These give us the mean ($\mathbf{m}_*$) and the variance ($\mathbf{C}_*$) at that point, i.e. the value and the uncertainty on that value.

$$\mathbf{m}_* = \mathbf{k}_*^\mathbf{T}\mathbf{K}^{-1}\mathbf{y} \tag{6}$$

$$\mathbf{C}_* = \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\mathbf{T}\mathbf{K}^{-1}\mathbf{k}_* \tag{7}$$

See Chapter 2 of Rasmussen & Williams to see where this comes from in more detail.

`www.gaussianprocess.org/gpml/chapters/RW2.pdf`

Predictions - Non-zero Mean

To calculate the value of a hypothetical measurement at another position, x_*, we can use the following equations. These give us the mean ($\mathbf{m}_*$) and the variance ($\mathbf{C}_*$) at that point, i.e. the value and the uncertainty on that value.

$$\mathbf{m}_* = \mu_* + \mathbf{k}_*^{\mathbf{T}}\mathbf{K}^{-1}(\mathbf{y} - \mu) \tag{8}$$

$$\mathbf{C}_* = \mathbf{k}(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^{\mathbf{T}}\mathbf{K}^{-1}\mathbf{k}_* \tag{9}$$

See Chapter 2 of Rasmussen & Williams to see where this comes from in more detail.

`www.gaussianprocess.org/gpml/chapters/RW2.pdf`

If we then take the final realization, which has $\lambda = 5$, and select 5 points from it as training data then we can calculate the posterior mean and variance at every other point based on those five training points.

First let's select our **training data points** and our **test data points**:

```
# set number of training points
nx_training = 5

# randomly select the training points:
tmp = np.random.uniform(low=0.0, high=2000.0, size=nx_training)
tmp = tmp.astype(int)

condition = np.zeros_like(x1)
for i in tmp: condition[i] = 1.0

y_train = y1[np.where(condition==1.0)]
x_train = x1[np.where(condition==1.0)]
y_test = y1[np.where(condition==0.0)]
x_test = x1[np.where(condition==0.0)]
```

Numpy Linear Algebra

```
# define the covariance matrix:
K = make_K(x_train,h,lam)

# take the inverse:
iK = np.linalg.inv(K)
```

SciPy.org Sponsored By ENTHOUGHT

Scipy.org | Docs | NumPy v1.12 Manual | NumPy Reference | Routines | Linear algebra (`numpy.linalg`) | index | next | previous

numpy.linalg.inv

numpy.linalg.inv(*a*) [source]

Compute the (multiplicative) inverse of a matrix.

Given a square matrix *a*, return the matrix *ainv* satisfying `dot(a, ainv) = dot(ainv, a) = eye(a.shape[0])`.

Parameters: **a** : *(..., M, M) array_like*
Matrix to be inverted.

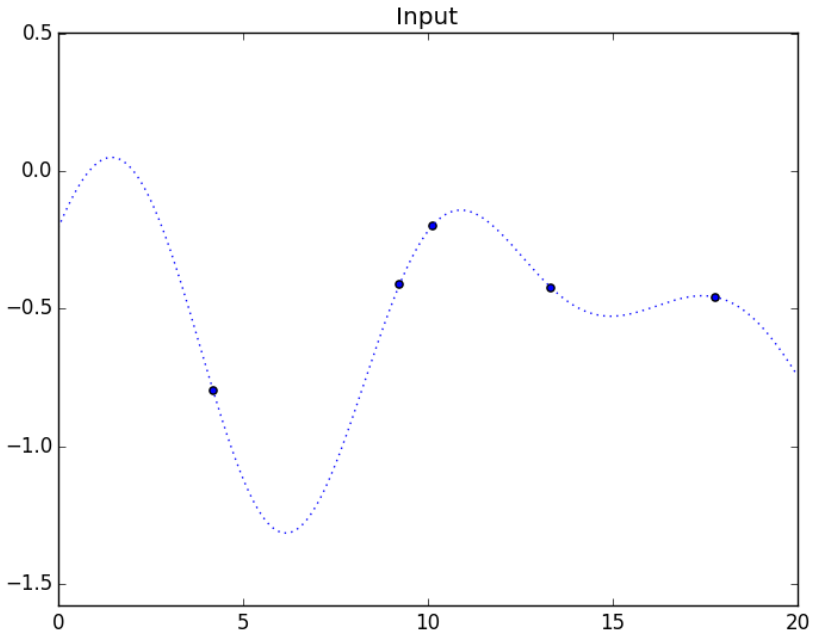
Returns: **ainv** : *(..., M, M) ndarray or matrix*
(Multiplicative) inverse of the matrix *a*.

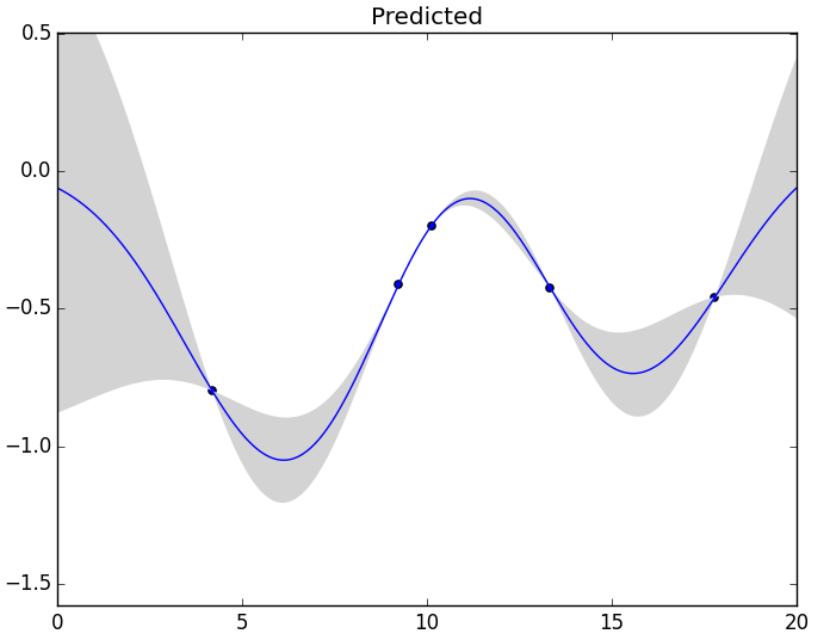
Raises: **LinAlgError**
If *a* is not square or inversion fails.

Previous topic
numpy.linalg.lstsq

Next topic
numpy.linalg.pinv

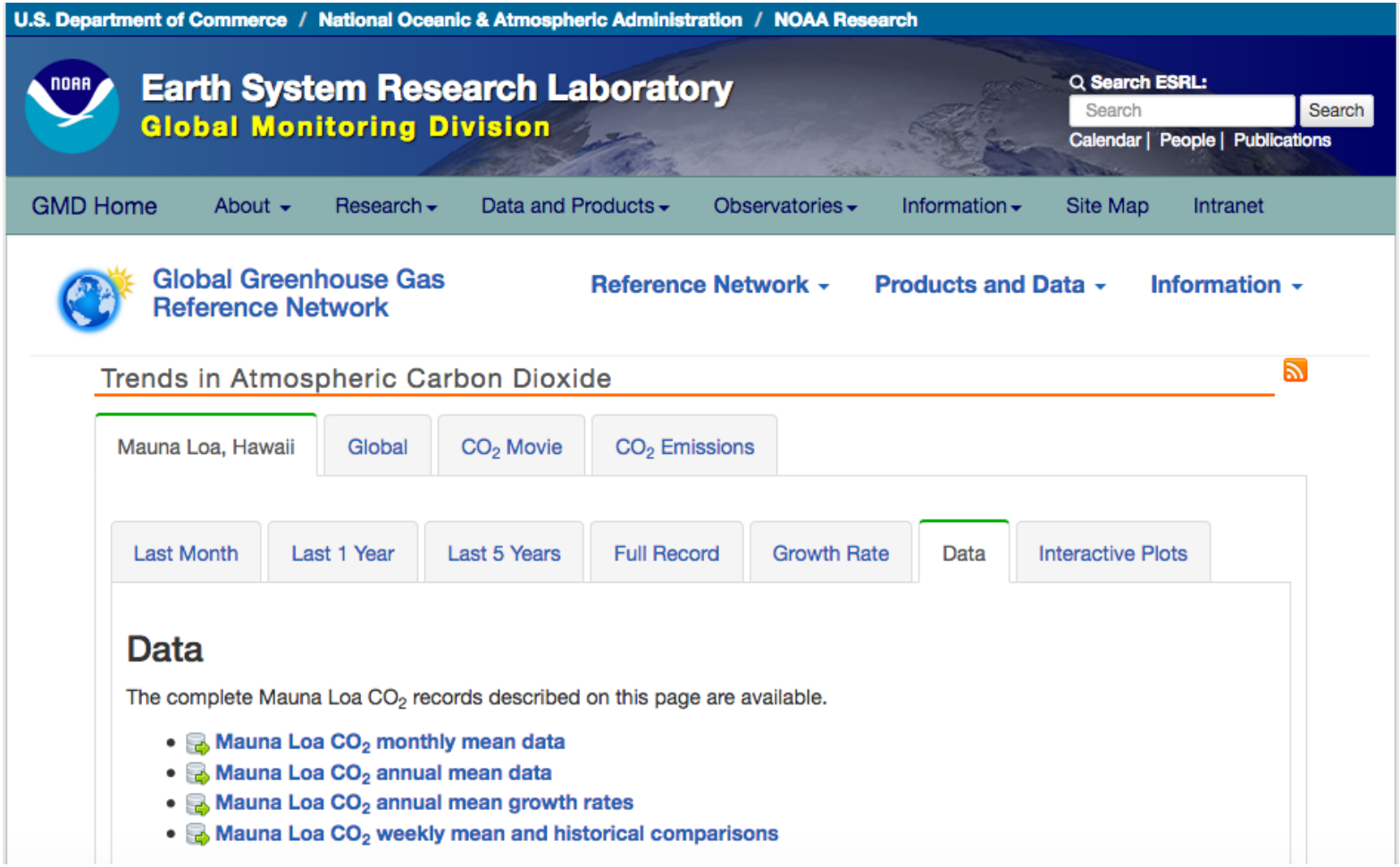
```
mu=[]; sig=[]
for xx in x_test:

    # find the 1d covariance matrix:
    K_x = cov_kernel(xx, x_train, h, lam, 0.0)

    # find the kernel for (x,x):
    k_xx = cov_kernel(xx, xx, h, lam, 0.0)

    # calculate the posterior mean and variance:
    mu_xx = np.dot(K_x.T,np.dot(iK,y_train))
    sig_xx = k_xx - np.dot(K_x.T,np.dot(iK,K_x))

    # append values into lists
    # note sqrt to get stdev from variance
    mu.append(mu_xx)
    sig.append(np.sqrt(np.abs(sig_xx)))
```


Input
0.5
0.0
−0.5
−1.0
−1.5
0
5
10
15
20


Predicted
0.5
0.0
−0.5
−1.0
−1.5
0
5
10
15
20

CO_2 Data

Now we have our methodology established, we need some data.


U.S. Department of Commerce / National Oceanic & Atmospheric Administration / NOAA Research
Earth System Research Laboratory
Global Monitoring Division
Search ESRL:
Search
Search
Calendar | People | Publications
GMD Home
About
Research
Data and Products
Observatories
Information
Site Map
Intranet
Global Greenhouse Gas Reference Network
Reference Network
Products and Data
Information
Trends in Atmospheric Carbon Dioxide
Mauna Loa, Hawaii
Global
CO2 Movie
CO2 Emissions
Last Month
Last 1 Year
Last 5 Years
Full Record
Growth Rate
Data
Interactive Plots
Data
The complete Mauna Loa CO2 records described on this page are available.
Mauna Loa CO2 monthly mean data
Mauna Loa CO2 annual mean data
Mauna Loa CO2 annual mean growth rates
Mauna Loa CO2 weekly mean and historical comparisons

StatsModels
Statistics in Python

Install | Support | Bugs | Develop | Examples | FAQ | previous | next | index

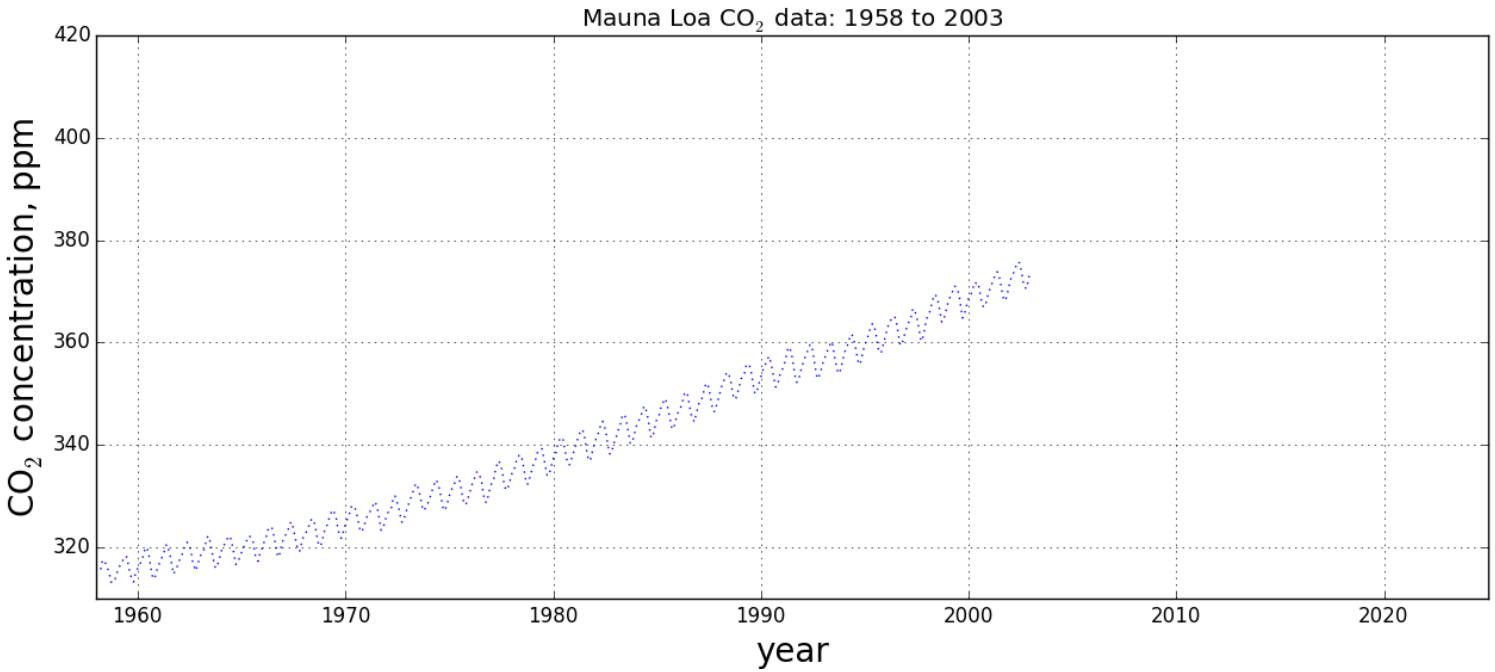
Table Of Contents

The Datasets Package

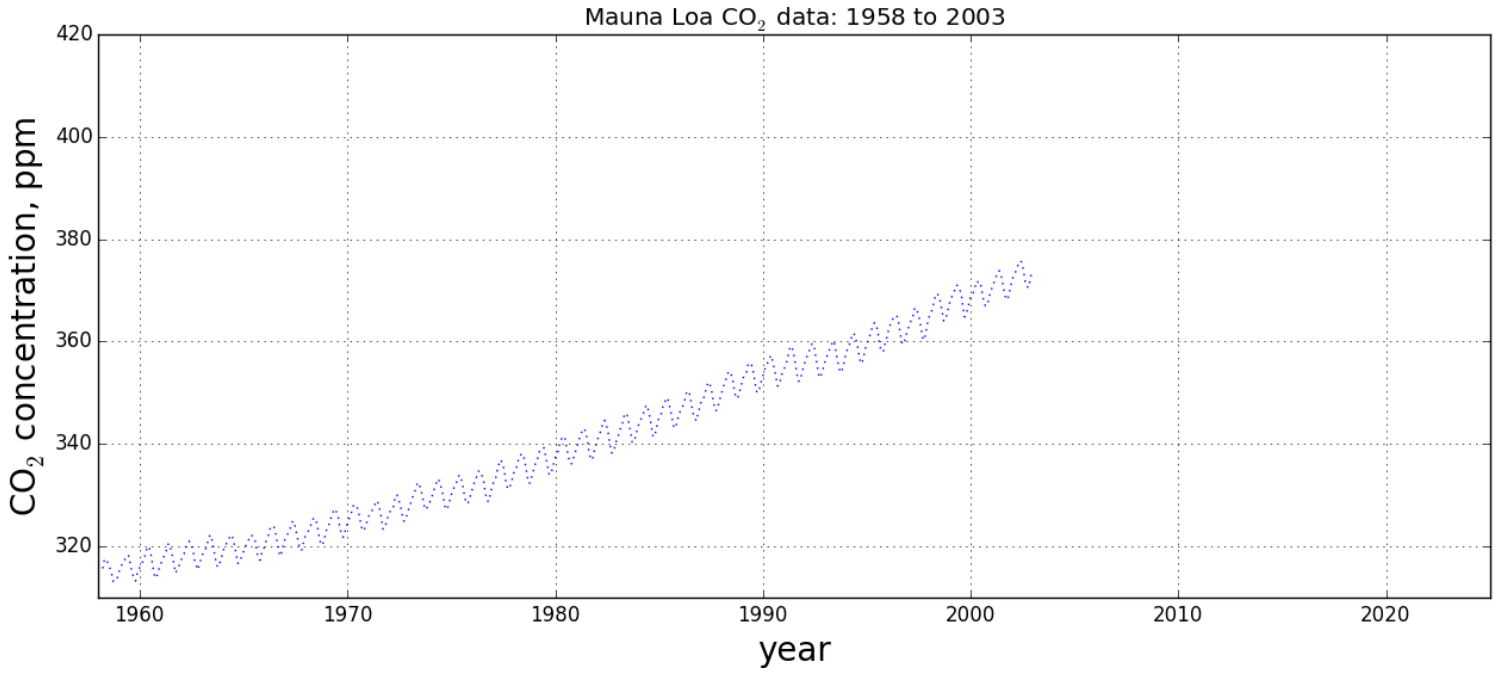
`statsmodels` provides data sets (i.e. data *and* meta-data) for use in examples, tutorials, model testing, etc.

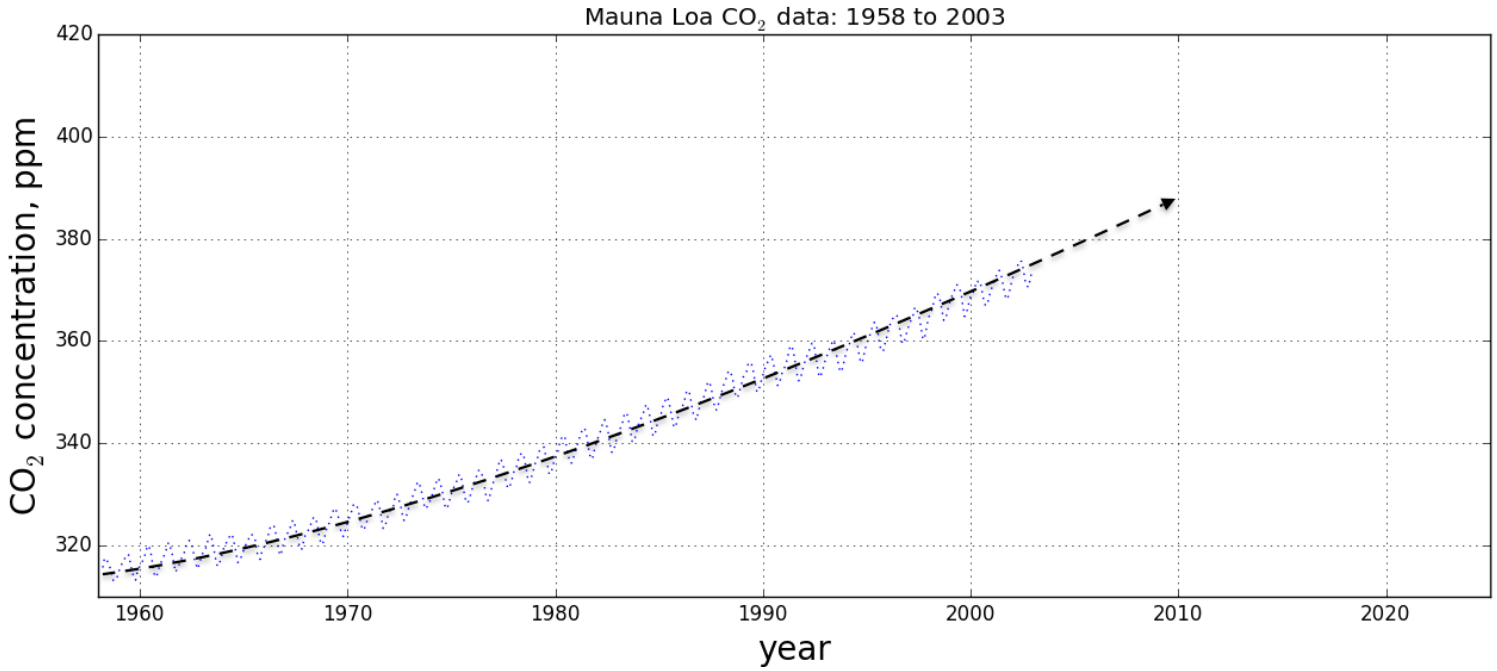
```
import statsmodels.api as sm

# grab the online data:
data = sm.datasets.get_rdataset("co2").data

# extract time data values:
t = np.array(data.time)

# extract CO_2 data values:
y = np.array(data.co2)
```

CO_2 Data

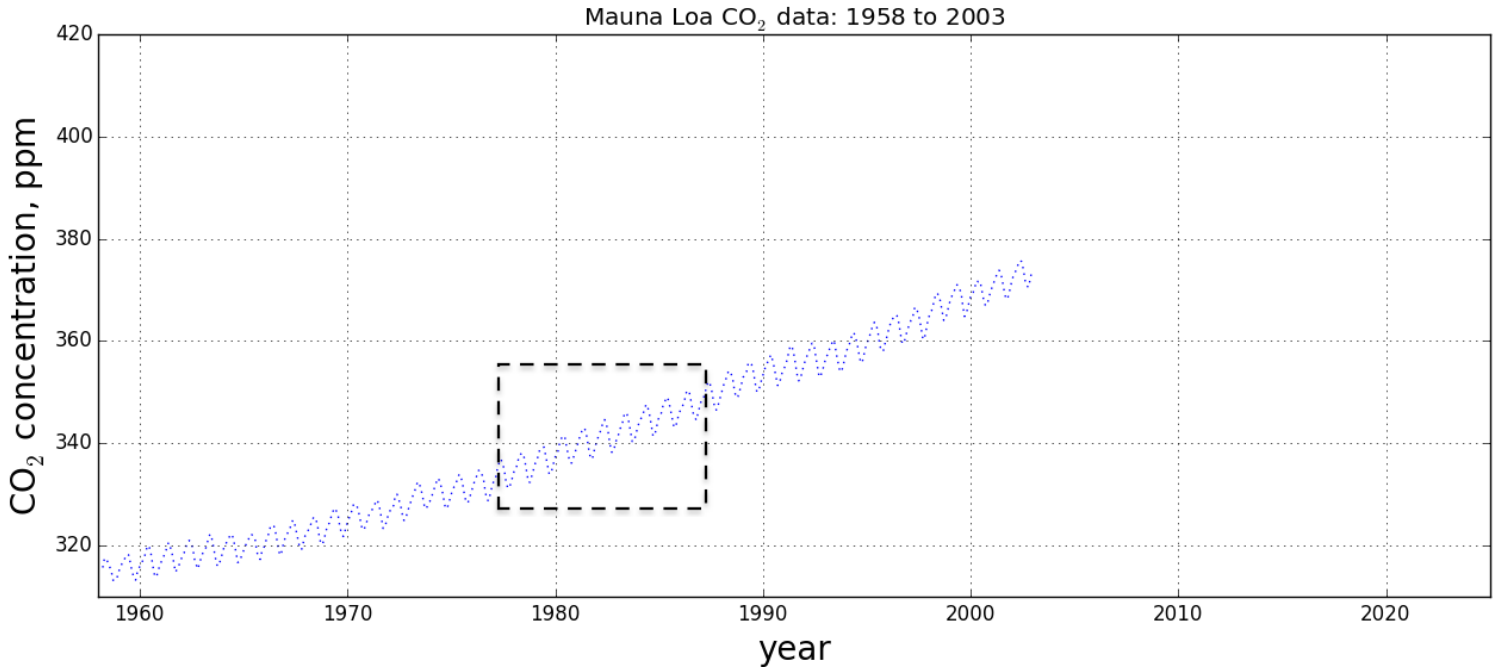


There are of course more data from the year 2003 to now. I have selected the data up to 2003 to be our **training data points**. We are going to use these training data to predict the future at **test data points** from 2003 to 2025...

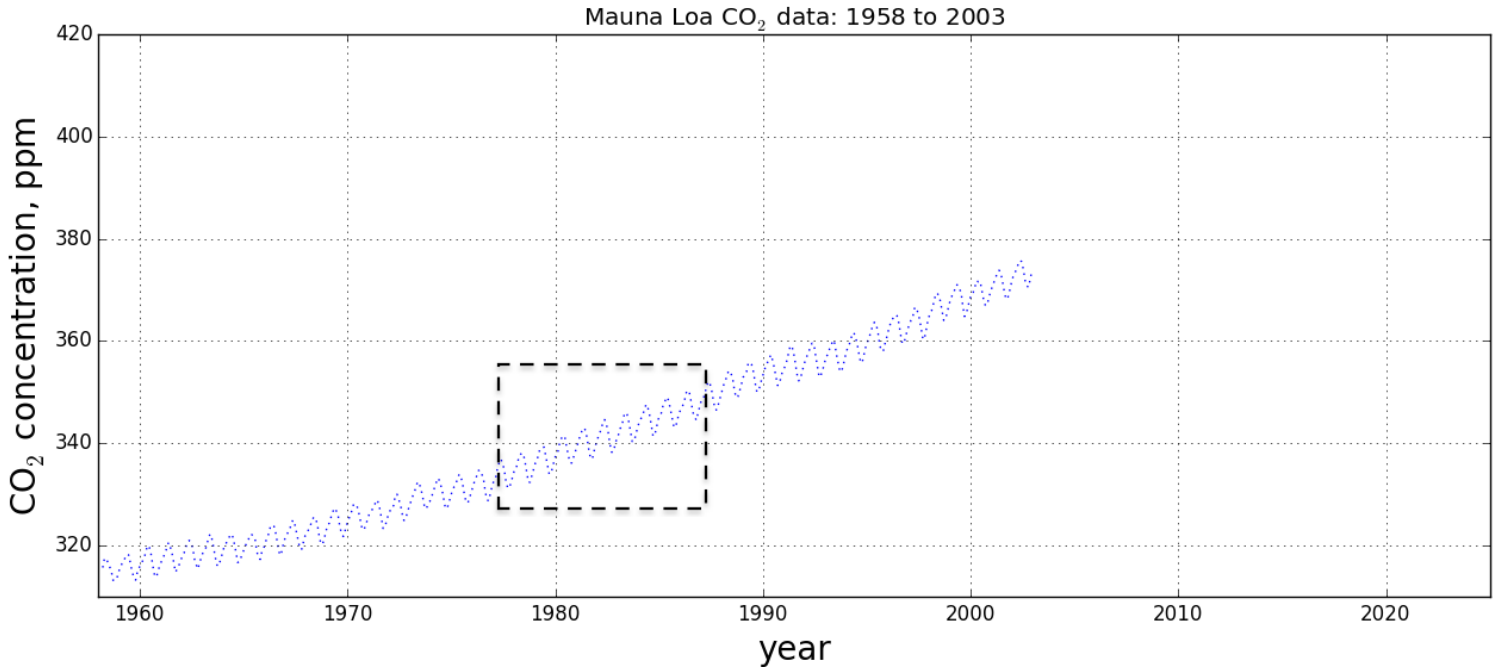

Mauna Loa CO_2 data: 1958 to 2003
420
400
380
360
340
320
CO_2 concentration, ppm
1960
1970
1980
1990
2000
2010
2020
year



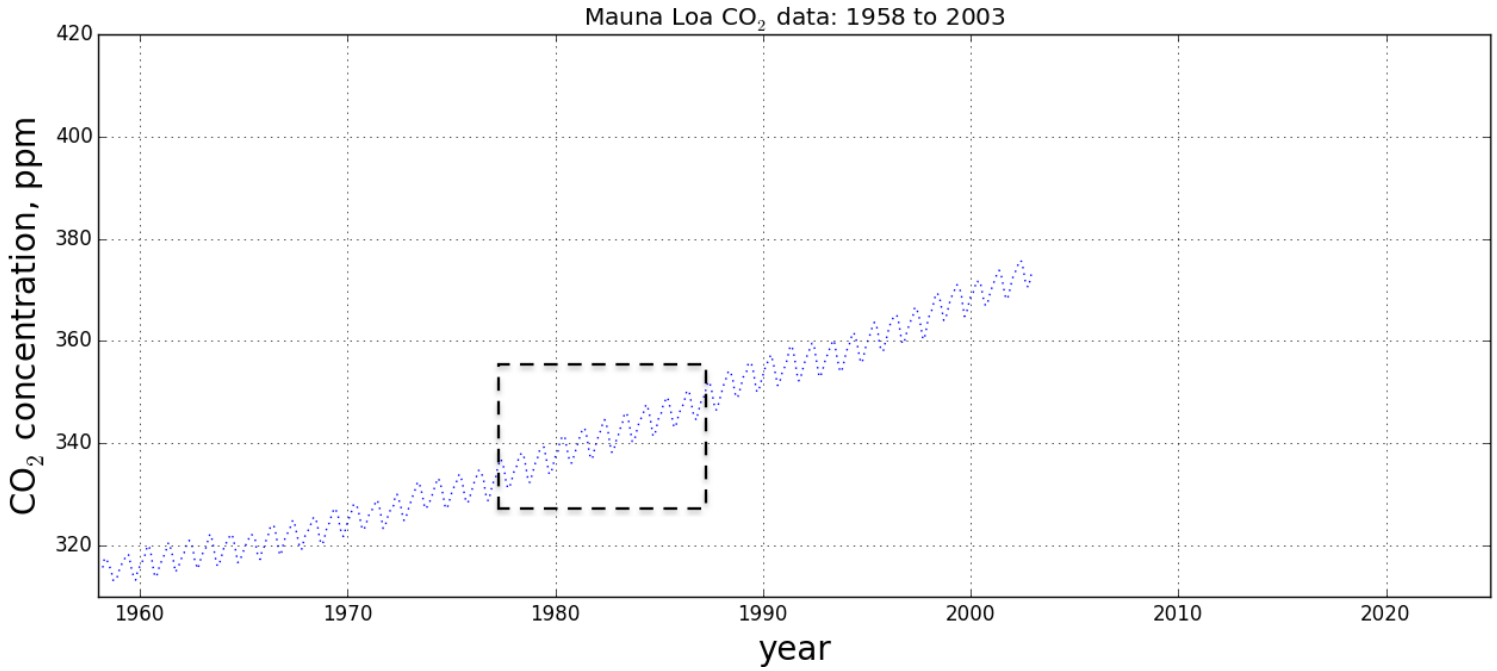
- Feature One: Long term smooth rising trend



- Feature One: Long term smooth rising trend
- Feature Two: Seasonal variation



- Feature One: Long term smooth rising trend
- Feature Two: Seasonal variation
- Feature Three: Medium term irregularities



- Feature One: Long term smooth rising trend
- Feature Two: Seasonal variation
- Feature Three: Medium term irregularities
- Feature Four: Noise

George GPM Library

We could write a library that defines a whole load of different covariance kernels to describe each of these trends... but infact somebody has already done it for us:

`http://dan.iel.fm/george/current/`

Feature One: Long term smooth rising trend

```
import george
from george import kernels

# Squared exponential kernel
# h = 66; lambda = 67
k1 = 66.0**2 * kernels.ExpSquaredKernel(67.0**2)
```

$$k(x_i, x_j) = h^2 \exp\left(\frac{-(x_i - x_j)^2}{\lambda^2}\right) \tag{10}$$

Feature Two: Seasonal variation

```
# periodic covariance kernel with exponential component to
# allow decay away from periodicity:
# h = 2.4; lambda = 90; gamma = 1.3; P = 1
k2 = 2.4**2 * kernels.ExpSquaredKernel(90**2) \
                        * kernels.ExpSine2Kernel(2.0 / 1.3**2, 1.0)
```

$$k(x_i, x_j) = h^2 \exp\left(-\frac{(x_i - x_j)^2}{\lambda^2} - \frac{2}{\gamma^2}\sin^2\left(\frac{\pi(x_i - x_j)}{P}\right)\right) \tag{11}$$

Feature Three: Medium term irregularities

```
# rational quadratic kernel
# h = 0.66; alpha = 0.78; beta = 1.2
k3 = 0.66**2 * kernels.RationalQuadraticKernel(0.78, 1.2**2)
```

$$k(x_i, x_j) = h^2 \left(1 + \frac{(x_i - x_j)^2}{2\alpha\beta^2}\right)^{-\alpha} \tag{12}$$

Feature Four: Noise

```
# noise kernel: includes correlated noise & uncorrelated noise
# h = 0.18; lambda = 1.6; sigma = 0.19
k4 = 0.18**2 * kernels.ExpSquaredKernel(1.6**2) \
                        + kernels.WhiteKernel(0.19)
```

$$k(x_i, x_j) = h^2 \exp\left(\frac{-(x_i - x_j)^2}{\lambda^2}\right) + \sigma^2\delta \tag{13}$$

Combined Kernels

Let's now put all these components together:

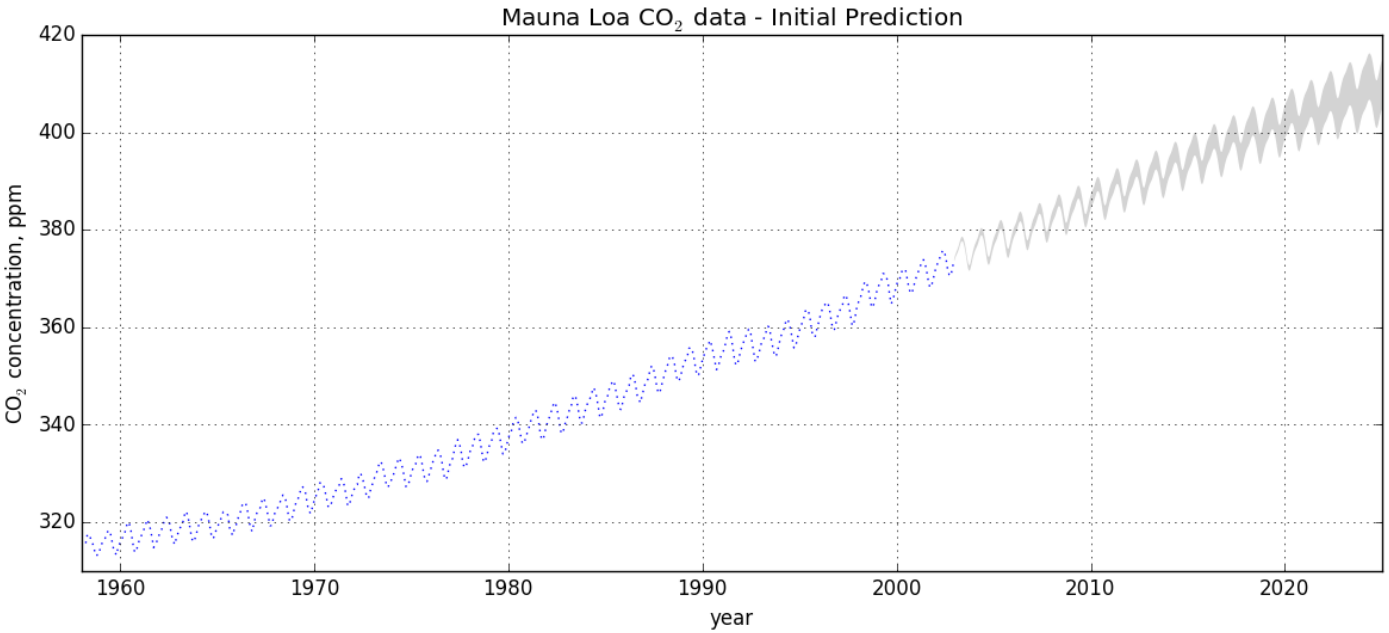
```
k1 = 66.0**2 * kernels.ExpSquaredKernel(67.0**2)
k2 = 2.4**2 * kernels.ExpSquaredKernel(90**2) \
                        * kernels.ExpSine2Kernel(2.0 / 1.3**2, 1.0)
k3 = 0.66**2 * kernels.RationalQuadraticKernel(0.78, 1.2**2)
k4 = 0.18**2 * kernels.ExpSquaredKernel(1.6**2) \
                        + kernels.WhiteKernel(0.19)

kernel = k1 + k2 + k3 + k4
```

Using George

```
# first we feed our combined kernel to the George library:
gp = george.GP(kernel, mean=np.mean(y))

# then we compute the covariance matrix:
gp.compute(t)
```

```
# range of times for prediction:
x = np.linspace(max(t), 2025, 2000)

# calculate expectation and variance at each point:
mu, cov = gp.predict(y, x)
std = np.sqrt(np.diag(cov))
```

Initial Prediction



To use the scipy library optimization function we need to provide (1) some function to optimize and (2) the **gradient** of that function.

Here we define the objective function for the optimization as a negative log-likelihood. We could write this function ourselves, but in fact george has a built in log-likelihood that we can simply call directly.

The log-likelihood is computed as:

$$\log \mathcal{L} \propto (\mathbf{y} - X^T\mathbf{x})^T C^{-1} (\mathbf{y} - X^T\mathbf{x}) \tag{14}$$

where y is the variable and x are the points at which it is measured; C is the covariance matrix and X is the operator that maps x onto y.

```
def nll(p):

    # Update the kernel parameters and compute the likelihood.
    gp.kernel[:] = p
    ll = gp.lnlikelihood(y, quiet=True)

    # The scipy optimizer doesn't play well with infinities.
    return -ll if np.isfinite(ll) else 1e25
```

```
def grad_nll(p):

    # Update the kernel parameters and compute the likelihood gradient.
    gp.kernel[:] = p
    gll = gp.grad_lnlikelihood(y, quiet=True)

    return -gll
```

Optimization

Then run the optimization routine:

```
import scipy.optimize as op

# initial guess at parameters:
p0 = gp.kernel.vector

# run optimization:
results = op.minimize(nll, p0, jac=grad_nll)
```

Update

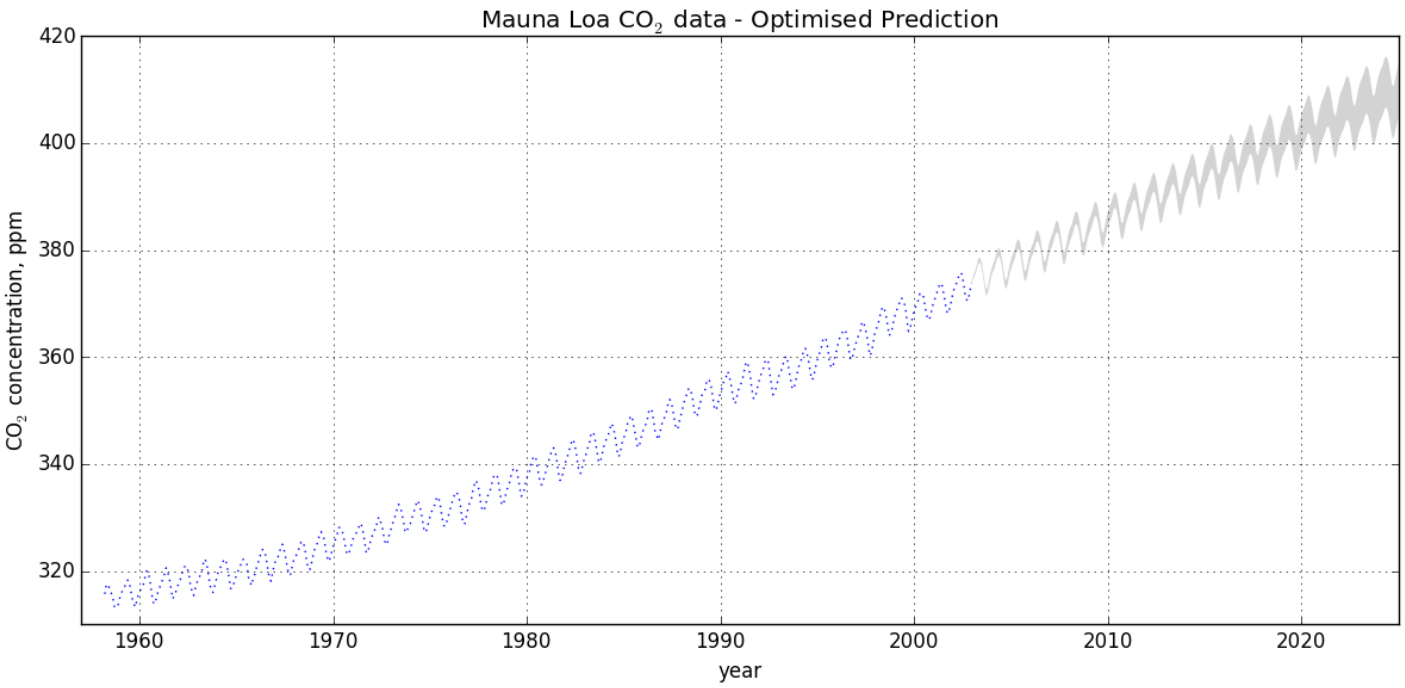
Update the kernel with the results of the optimization:

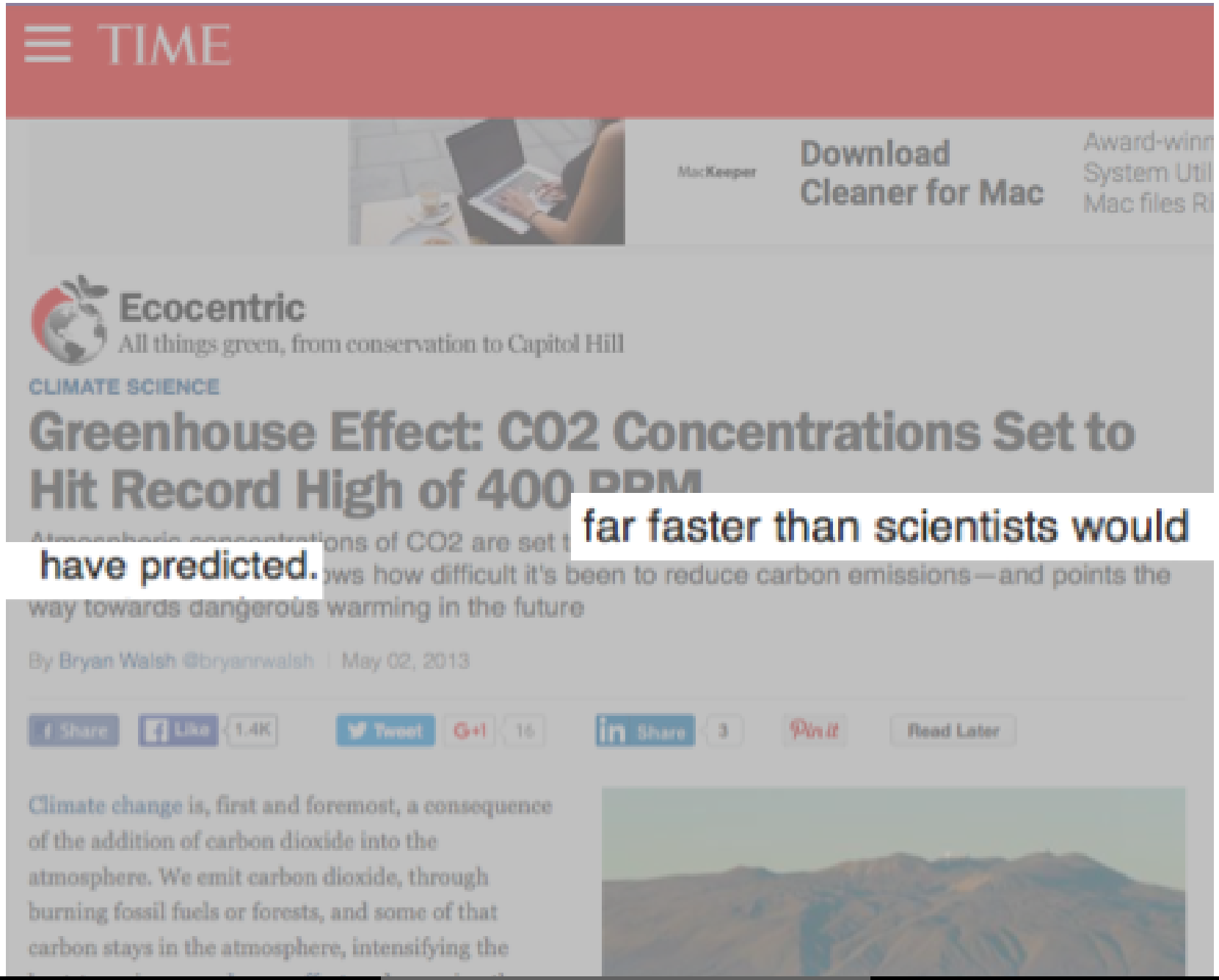
```
gp.kernel[:] = results.x
```

Rerun the prediction with the updated parameters:

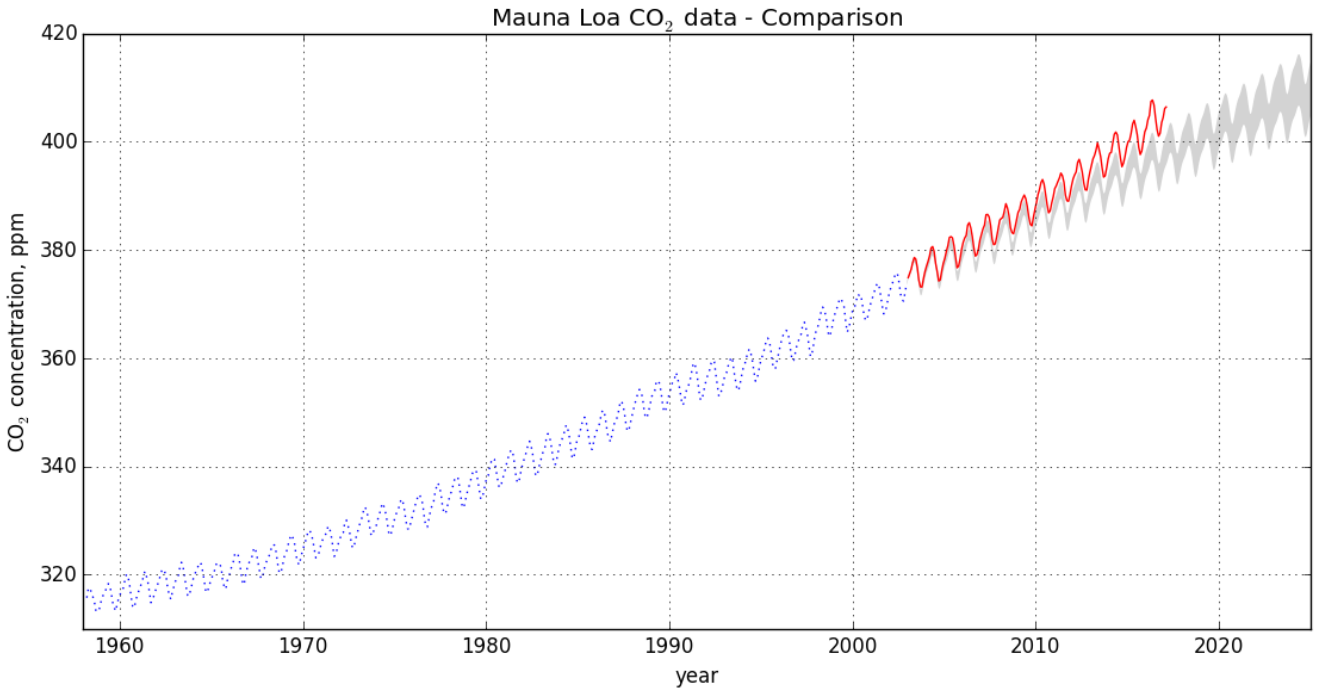
```
# range of times for prediction:
x = np.linspace(max(t), 2025, 2000)

# calculate expectation and variance at each point:
mu, cov = gp.predict(y, x)
std = np.sqrt(np.diag(cov))
```

Optimised Prediction




TIME
Download
Cleaner for Mac
Ecocentric
All things green, from conservation to Capitol Hill
CLIMATE SCIENCE
Greenhouse Effect: CO2 Concentrations Set to
far faster than scientists would
have predicted.
way towards dangerous warming in the future
By Bryan Walsh @bryanrwalsh | May 02, 2013
Climate change is, first and foremost, a consequence of the addition of carbon dioxide into the atmosphere. We emit carbon dioxide, through burning fossil fuels or forests, and some of that carbon stays in the atmosphere, intensifying the

Comparison


Mauna Loa CO_2 data - Comparison
420
400
380
360
340
320
CO_2 concentration, ppm
1960
1970
1980
1990
2000
2010
2020
year

Last Slide

All of the Python in this lecture can be found in:

`https://github.com/as595/AllOfYourBases/tree/master/TIARA/GaussianProcessModelling`

GPMIntro.ipynb
GPMCarbonDioxide.ipynb