

Can you predict the future..?

Gaussian Process Modelling for Forward Prediction

Anna Scaife¹

¹Jodrell Bank Centre for Astrophysics
University of Manchester

@radastrat



September 7, 2017

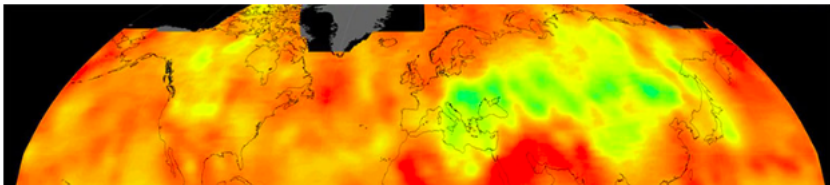
THE SPACE REPORTER

[HOME](#)[NASA](#)[SOLAR SYSTEM](#)[ASTRONOMY](#)[INTERNATIONAL SPACE STATION](#)[PRIVATE](#)**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

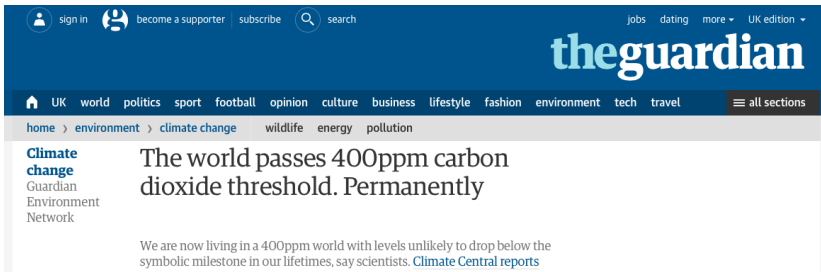
 [UK](#) [world](#) [politics](#) [sport](#) [football](#) [opinion](#) [culture](#) [business](#) [lifestyle](#) [fashion](#) [environment](#) [tech](#) [travel](#) [all sections](#)

[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](#)



The screenshot shows the Guardian website's header and a news article. The header is dark blue with the Guardian logo in white. Navigation links include 'sign in', 'become a supporter', 'subscribe', and 'search'. A secondary navigation bar lists various sections like 'UK', 'world', 'politics', etc. The article headline is 'The world passes 400ppm carbon dioxide threshold. Permanently'. Below the headline is a sub-headline: '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](#)'.

sign in become a supporter subscribe search

jobs dating more UK edition

theguardian

home UK world politics sport football opinion culture business lifestyle fashion environment tech travel all sections

home > environment > climate change wildlife energy pollution

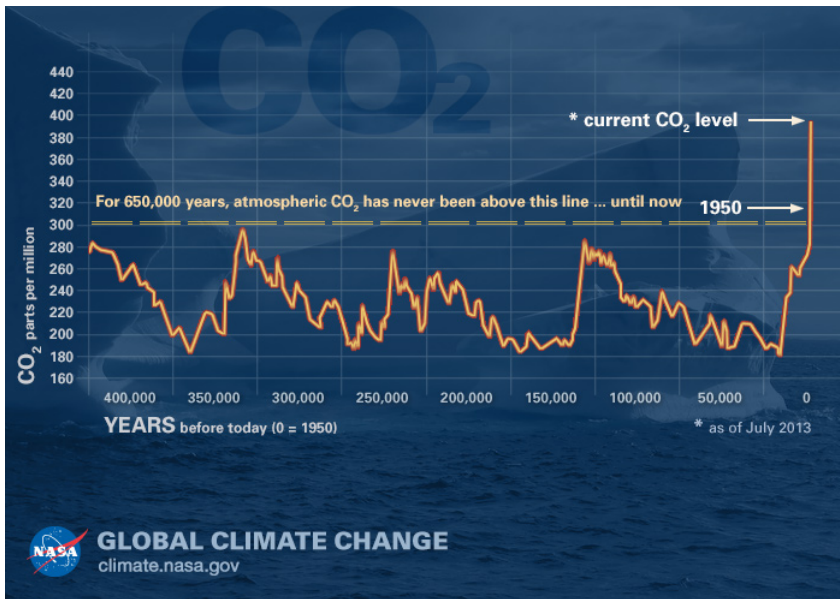
Climate change
Guardian Environment Network

The world passes 400ppm carbon dioxide threshold. Permanently

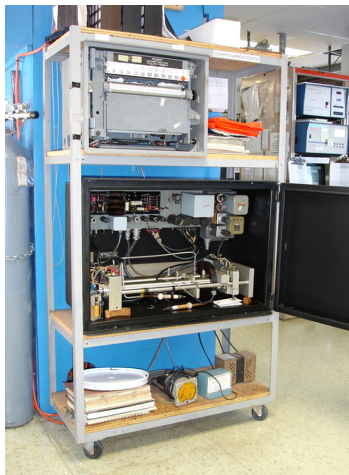
We are now living in a 400ppm world with levels unlikely to drop below the symbolic milestone in our lifetimes, say scientists. [Climate Central reports](#)

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

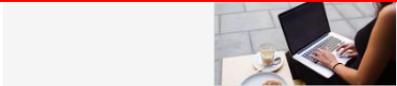
Last time it was at that level, human life did not exist.



Measuring CO₂



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



MacKeeper

Download
Cleaner for Mac

Award-winn
System Utili
Mac files Ri



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

[f Share](#) [f Like](#) 1.4K [t Tweet](#) [G+1](#) 16 [in 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






MacKeeper

Download
Cleaner for Mac

Award-winning
System Util
Mac files R



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 rise far faster than scientists would have predicted.

shows how difficult it's been to reduce carbon emissions—and points the way towards dangerous warming in the future

By Bryan Walsh @bryanwalsh | May 02, 2013




1.4K





3




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...

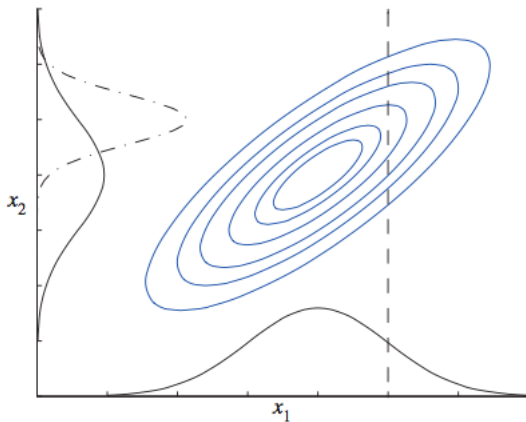
The increase has been caused by people, particularly the burning of fossil fuels and deforestation.

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...

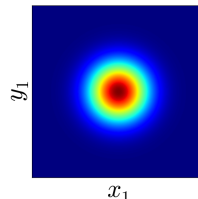
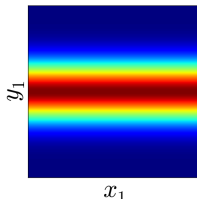
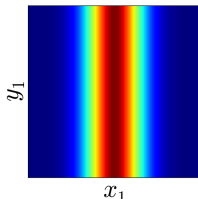
But... what if you didn't need to model all of those factors? What if you could **predict the future** pattern of behaviour just based on previously measured data?

Covariance



Aside: Independent Variables

$$\mathbf{K}(\mathbf{x}, \mathbf{x}) = \begin{bmatrix} k(x_1, x_1) & 0 \\ 0 & k(x_2, x_2) \end{bmatrix} = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$$



Numpy Multivariate Normal Distribution


[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, ..., z]` 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 .

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

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:

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

Random Events

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

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

```
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.

```
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
```

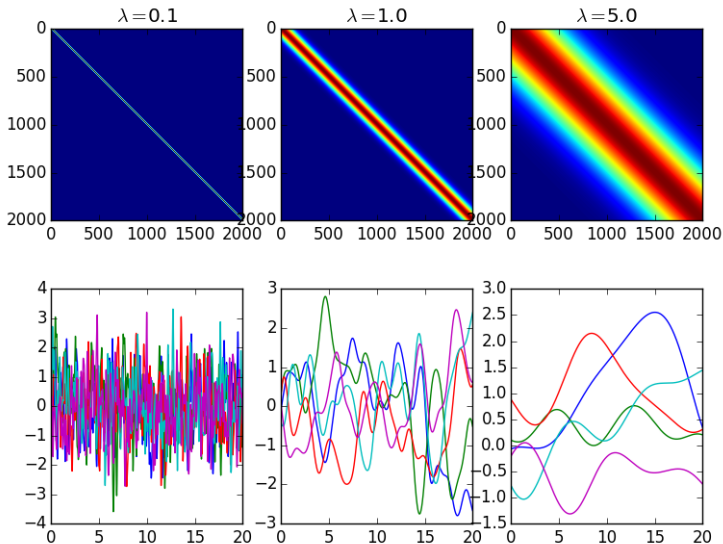
Once we have a covariance matrix we can start drawing samples from it using the **numpy** library:

```
# 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)
```

$$p(y(\mathbf{x})) = N(\mu(\mathbf{x}), K(\mathbf{x}, \mathbf{x}))$$



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

Given a set of data $\mathbf{y} = y(\mathbf{x})$ with a known covariance function, k , we can calculate the posterior probability, $p(y_*)$, at a test point \mathbf{x}_* , using the joint probability distribution:

$$p\left(\begin{bmatrix} \mathbf{y} \\ y_* \end{bmatrix}\right) = N\left(\begin{bmatrix} \boldsymbol{\mu} \\ \mu_* \end{bmatrix}, \begin{bmatrix} K(\mathbf{x}, \mathbf{x}) & K(\mathbf{x}, \mathbf{x}_*) \\ K(\mathbf{x}_*, \mathbf{x}) & K(\mathbf{x}_*, \mathbf{x}_*) \end{bmatrix}\right)$$

Predictions

To calculate the value of a hypothetical measurement at another position, \mathbf{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.

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

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

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.

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

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

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

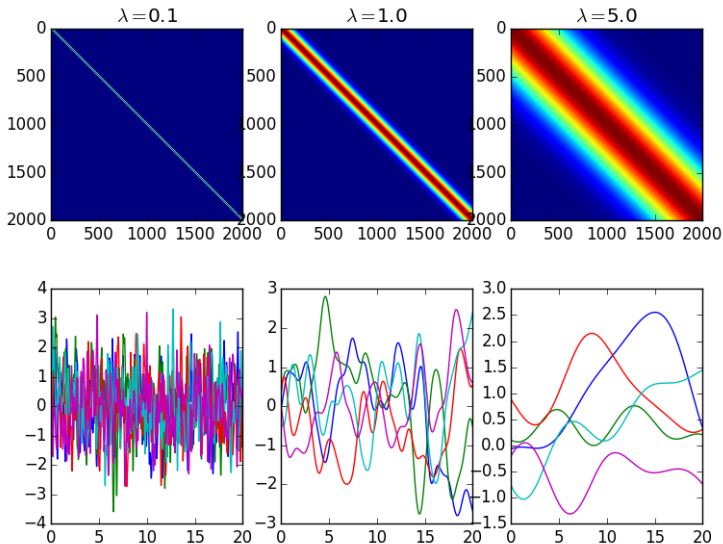
To calculate the value of a hypothetical measurement at another position, x_* , we can use the following equations. These give us the mean (\mathbf{m}_*) and the variance (\mathbf{C}_*) at that point, i.e. the value and the uncertainty on that value.

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

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

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

Scalable Data
SCIENTIFIC
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)`

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.

[\[source\]](#)

Previous topic

[numpy.linalg.lstsq](#)

Next topic

[numpy.linalg.pinv](#)

```

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

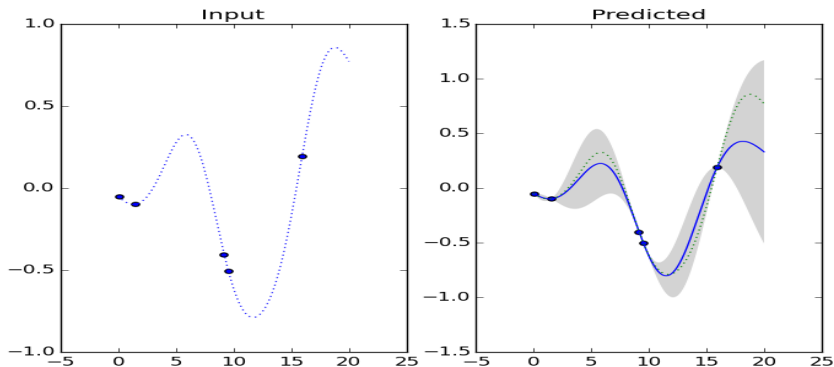
    # find the 1d covariance matrix:
    K_x = cov_kernel(xx, x_train, h, lam, 0.0)

    # 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)))


```



CO₂ 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

Q Search ESRL: 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

- [Using Datasets from Stata](#)
- [Using Datasets from R](#)
- [R Datasets Function](#)

The Datasets Package

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

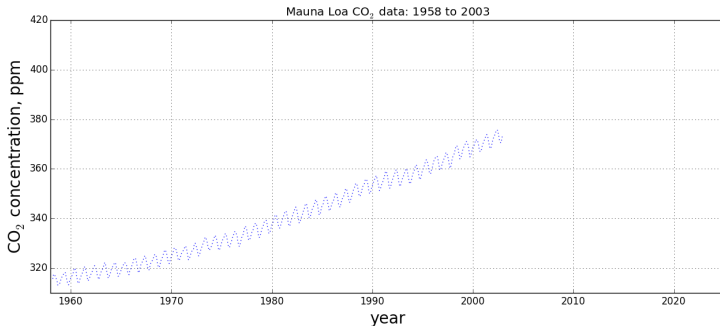
```
import statsmodels.api as sm

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

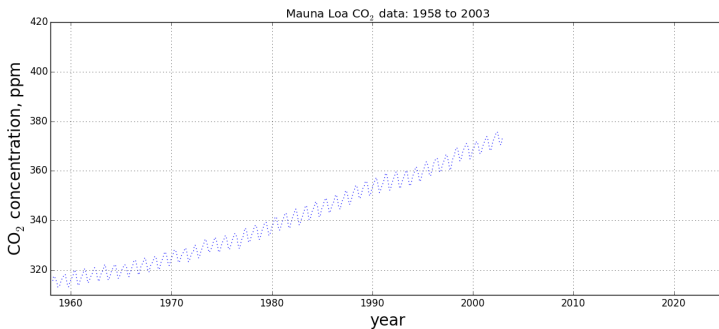
# extract time data values:
t = np.array(data.time)

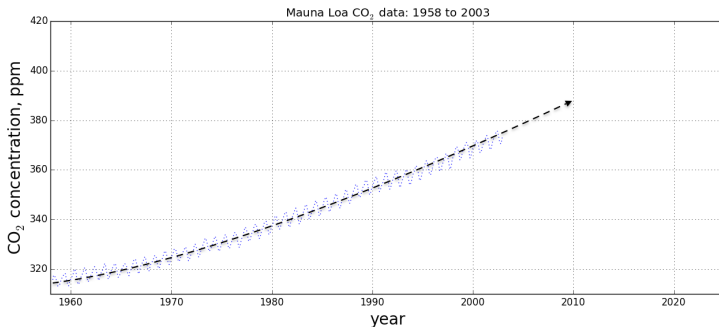
# extract CO_2 data values:
y = np.array(data.co2)
```

CO₂ Data

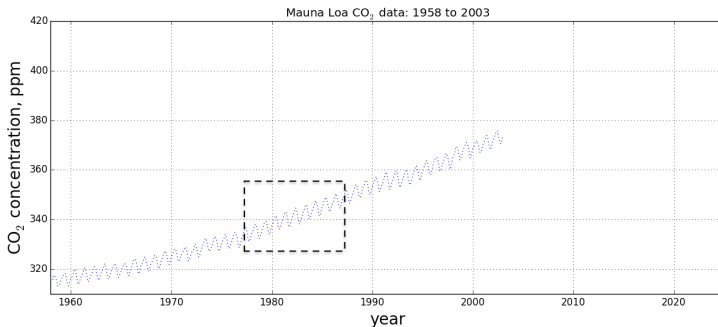


There are of course more data from the year 2003 to now. I have selected the data up to 2003 to be our **training data points**. We are going to use these training data to **predict the future** at **test data points** from 2003 to 2025...

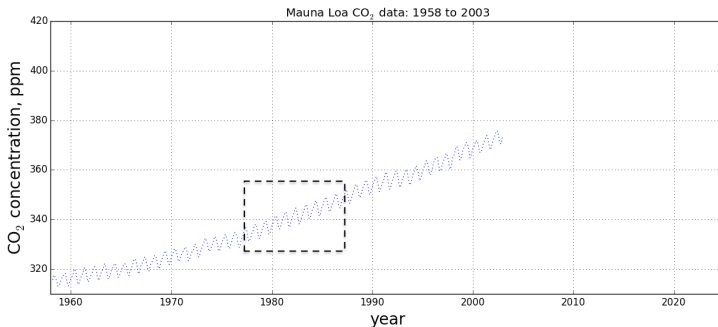




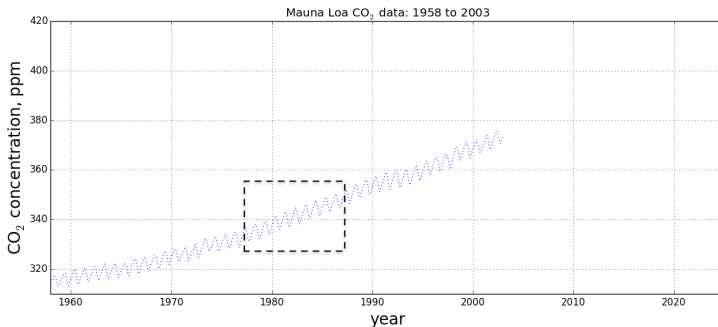
- **Feature One:** Long term smooth rising trend



- **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

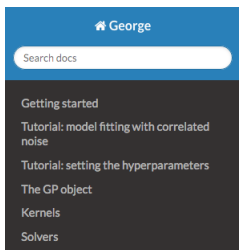


- **Feature One:** Long term smooth rising trend
- **Feature Two:** Seasonal variation
- **Feature Three:** Medium term irregularities
- **Feature Four:** Noise

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/>



[Docs](#) » [George](#)

[Edit on GitHub](#)

George

George is a fast and flexible Python library for Gaussian Process Regression. A full introduction to the theory of Gaussian Processes is beyond the scope of this documentation but the best resource is available for free online: [Rasmussen & Williams \(2006\)](#).

George is being actively developed in a [public repository on GitHub](#) so if you have any trouble, [open an issue](#) there.

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)
```

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

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)
```

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

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)
```

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

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)
```

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

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:
```

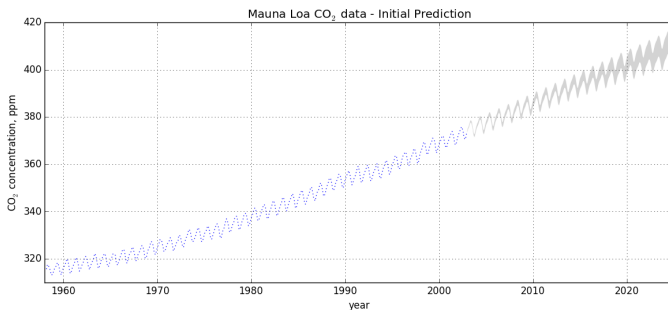
```
x = np.linspace(max(t), 2025, 2000)
```

```
# 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:

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

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

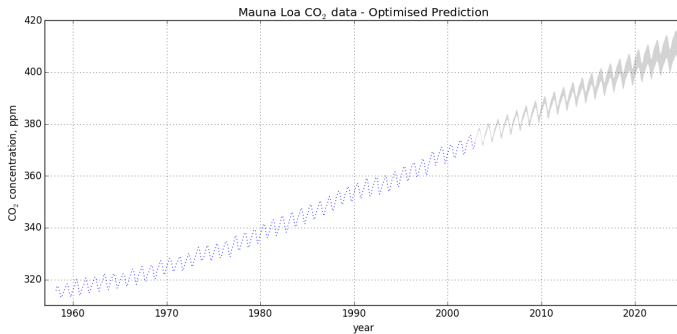
Update the kernel with the results of the optimization:

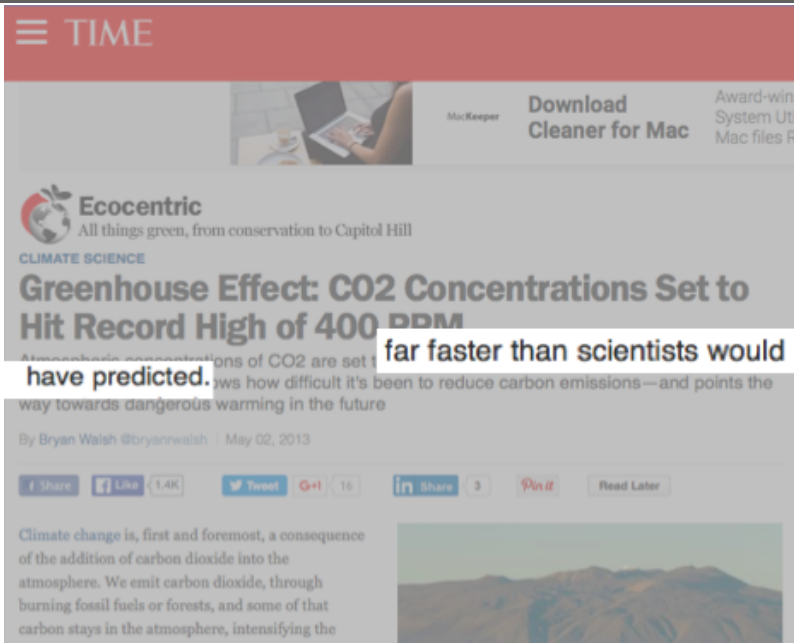
```
gp.kernel[:] = results.x
```

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

Award-winning System Utility Mac files R

MacKeeper

 **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 rise far faster than scientists would have predicted.

shows how difficult it's been to reduce carbon emissions—and points the way towards dangerous warming in the future

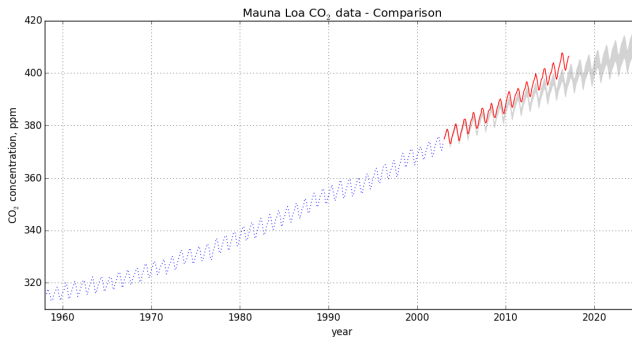
By Bryan Walsh @bryanwalsh | May 02, 2013

Share Like 1.4K Tweet G+ 15 in 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

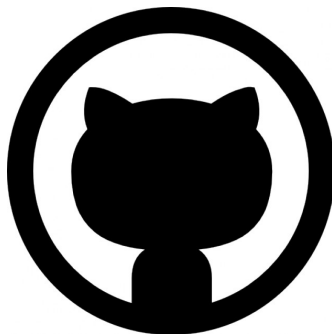


Comparison



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

Kepler Orrery

<https://youtu.be/AnY1Vl6BaYo?t=26s>

arXiv:1703.009710

Inferring probabilistic stellar rotation periods using Gaussian processes

Ruth Angus,^{1★} Timothy Morton², Suzanne Aigrain³, Daniel Foreman-Mackey⁴
& Vinesh Rajpaul³

¹*Simons Fellow, Department of Astronomy, Columbia University, NY, NY*

²*Department of Astrophysical Sciences, Princeton University, Princeton, NJ*

³*Subdepartment of Astrophysics, University of Oxford, UK*

⁴*Sagan Fellow, Department of Astronomy, University of Washington, Seattle, WA*

arXiv:1706.05459

FAST AND SCALABLE GAUSSIAN PROCESS MODELING WITH APPLICATIONS TO ASTRONOMICAL TIME SERIES

DANIEL FOREMAN-MACKEY^{1,2}, ERIC AGOL^{2,3}, SIVARAM AMBIKASARAN⁴, AND
RUTH ANGUS^{5,6}