Gaussian Processes

[edit]

Jan 9, 2019

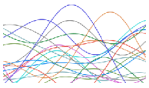
at **MLSS, Stellenbosch, South Africa** on Jan 9, 2019 [jupyter][reveal] ***Neil D. Lawrence, Amazon Cambridge and University of Sheffield***

$$\mathbf{a}$$

$$\mathbf{,X})\boldsymbol{\alpha}$$

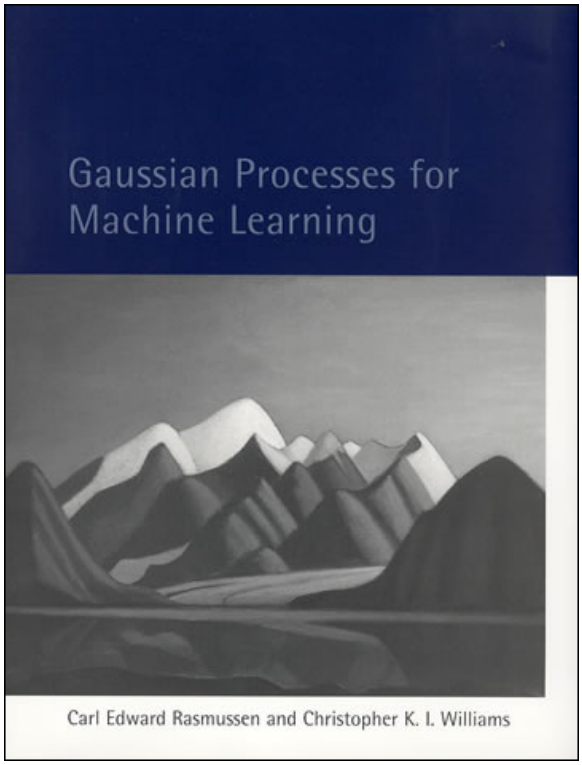
iance

$$\mathbf{K}_{*,f}\mathbf{K}^{-1}\mathbf{K}_{f,*}$$



Abstract

Classical machine learning and statistical approaches to learning, such as neural networks and linear regression, assume a parametric form for functions. Gaussian process models are an alternative approach that assumes a probabilistic prior over functions. This brings benefits, in that uncertainty of function estimation is sustained throughout inference, and some challenges: algorithms for fitting Gaussian processes tend to be more complex than parametric models. In this sessions I will introduce Gaussian processes and explain why sustaining uncertainty is important.



Rasmussen and Williams (2006) is still one of the most important references on Gaussian process models. It is available freely online.

What is Machine Learning?

What is machine learning? At its most basic level machine learning is a combination of

$$\text{data} + \text{model} \xrightarrow{\text{compute}} \text{prediction}$$

where *data* is our observations. They can be actively or passively acquired (meta-data). The *model* contains our assumptions, based on previous experience. That experience can be other data, it can come from transfer learning, or it can merely be our beliefs about the regularities of the universe. In humans our models include our inductive biases. The *prediction* is an action to be taken or a categorization or a quality score. The reason that machine learning has become a mainstay of artificial intelligence is the importance of predictions in artificial intelligence. The data and the model are combined through computation.

In practice we normally perform machine learning using two functions. To combine data with a model we typically make use of:

a prediction function a function which is used to make the predictions. It includes our beliefs about the regularities of the universe, our assumptions about how the world works, e.g. smoothness, spatial similarities, temporal similarities.

an objective function a function which defines the cost of misprediction. Typically it includes knowledge about the world's generating processes (probabilistic objectives) or the costs we pay for mispredictions (empiricial risk minimization).

The combination of data and model through the prediction function and the objectie function leads to a *learning algorithm*. The class of prediction functions and objective functions we can make use of is restricted by the algorithms they lead to. If the prediction function or the objective function are too complex, then it can be difficult to find an appropriate learning algorithm. Much of the acdemic field of machine learning is the quest for new learning algorithms that allow us to bring different types of models and data together.

A useful reference for state of the art in machine learning is the UK Royal Society Report, Machine Learning: Power and Promise of Computers that Learn by Example.

You can also check my blog post on "What is Machine Learning?"

In practice, we normally also have uncertainty associated with these functions. Uncertainty in the prediction function arises from

1. scarcity of training data and
2. mismatch between the set of prediction functions we choose and all possible prediction functions.

There are also challenges around specification of the objective function, but for we will save those for another day. For the moment, let us focus on the prediction function.

Neural Networks and Prediction Functions

Neural networks are adaptive non-linear function models. Originally, they were studied (by McCulloch and Pitts (McCulloch and Pitts 1943)) as simple models for neurons, but over the last decade they have become popular because they are a flexible approach to modelling complex data. A particular characteristic of neural network models is that they can be composed to form highly complex functions which encode many of our expectations of the real world. They allow us to encode our assumptions about how the world works.

We will return to composition later, but for the moment, let's focus on a one hidden layer neural network. We are interested in the prediction function, so we'll ignore the objective function (which is often called an error function) for the moment, and just describe the mathematical object of interest

$$f(\mathbf{x}) = \mathbf{W}^\top \boldsymbol{\phi}(\mathbf{V}, \mathbf{x})$$

Where in this case $f(\cdot)$ is a scalar function with vector inputs, and $\boldsymbol{\phi}(\cdot)$ is a vector function with vector inputs. The dimensionality of the vector function is known as the number of hidden units, or the number of neurons. The elements of this vector function are known as the *activation* function of the neural network and $\mathbf{V}$ are the parameters of the activation functions.

Relations with Classical Statistics

In statistics activation functions are traditionally known as *basis functions*. And we would think of this as a *linear model*. It's doesn't make linear predictions, but it's linear because in statistics estimation focuses on the parameters, $\mathbf{W}$, not the parameters, $\mathbf{V}$. The linear model terminology refers to the fact that the model is *linear in the parameters*, but it is *not* linear in the data unless the activation functions are chosen to be linear.

Adaptive Basis Functions

The first difference in the (early) neural network literature to the classical statistical literature is the decision to optimize these parameters, $\mathbf{V}$, as well as the parameters, $\mathbf{W}$ (which would normally be denoted in statistics by $\boldsymbol{\beta}$)[1].

In this tutorial, we're going to go revisit that decision, and follow the path of Radford Neal (Neal 1994) who, inspired by work of David MacKay (MacKay 1992) and others did his PhD thesis on Bayesian Neural Networks. If we take a Bayesian approach to parameter inference (note I am using inference here in the classical sense, not in the sense of prediction of test data, which seems to be a newer usage), then we don't wish to fit parameters at all, rather we wish to integrate them away and understand the family of functions that the model describes.

Probabilistic Modelling

This Bayesian approach is designed to deal with uncertainty arising from fitting our prediction function to the data we have, a reduced data set.

The Bayesian approach can be derived from a broader understanding of what our objective is. If we accept that we can jointly represent all things that happen in the world with a probability distribution, then we can interogate that probability to make predictions. So, if we are interested in predictions, y_* at future points input locations of interest, $\mathbf{x}_*$ given previously training data, $\mathbf{y}$ and corresponding inputs, $\mathbf{X}$, then we are really interogating the following probability density,

$$p(y_*|\mathbf{y}, \mathbf{X}, \mathbf{x}_*),$$

there is nothing controversial here, as long as you accept that you have a good joint model of the world around you that relates test data to training data, $p(y_*, \mathbf{y}, \mathbf{X}, \mathbf{x}_*)$ then this conditional distribution can be recovered through standard rules of probability (data + model $\rightarrow$ prediction).

We can construct this joint density through the use of the following decomposition:

$$p(y_*|\mathbf{y}, \mathbf{X}, \mathbf{x}_*) = \int p(y_*|\mathbf{x}_*, \mathbf{W}) p(\mathbf{W}|\mathbf{y}, \mathbf{X}) \mathrm{d}\mathbf{W}$$

where, for convenience, we are assuming *all* the parameters of the model are now represented by $\boldsymbol{\theta}$ (which contains $\mathbf{W}$ and $\mathbf{V}$) and $p(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X})$ is recognised as the posterior density of the parameters given data and $p(y_*|\mathbf{x}_*, \boldsymbol{\theta})$ is the *likelihood* of an individual test data point given the parameters.

The likelihood of the data is normally assumed to be independent across the parameters,

$$p(\mathbf{y}|\mathbf{X}, \mathbf{W}) = \prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{W}),$$

and if that is so, it is easy to extend our predictions across all future, potential, locations,

$$p(\mathbf{y}_*|\mathbf{y}, \mathbf{X}, \mathbf{X}_*) = \int p(\mathbf{y}_*|\mathbf{X}_*, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X}) \mathrm{d}\boldsymbol{\theta}.$$

The likelihood is also where the *prediction function* is incorporated. For example in the regression case, we consider an objective based around the Gaussian density,

$$p(y_i|f(\mathbf{x}_i)) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - f(\mathbf{x}_i))^2}{2\sigma^2}\right)$$

In short, that is the classical approach to probabilistic inference, and all approaches to Bayesian neural networks fall within this path. For a deep probabilistic model, we can simply take this one stage further and place a probability distribution over the input locations,

$$p(\mathbf{y}_*|\mathbf{y}) = \int p(\mathbf{y}_*|\mathbf{X}_*, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}, \mathbf{X}) p(\mathbf{X}) p(\mathbf{X}_*) \mathrm{d}\boldsymbol{\theta} \mathrm{d}\mathbf{X} \mathrm{d}\mathbf{X}_*$$

and we have *unsupervised learning* (from where we can get deep generative models).

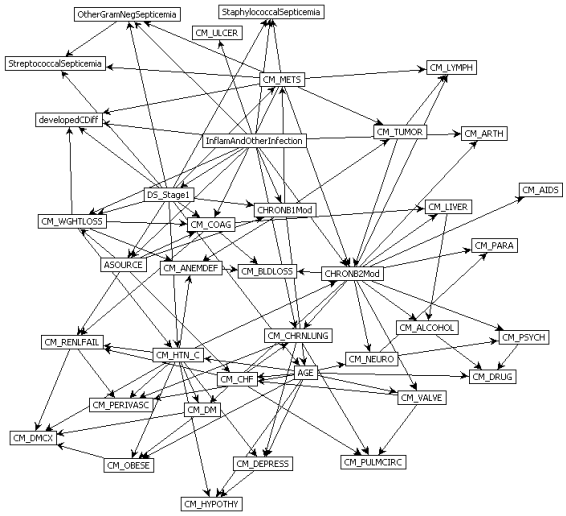
Graphical Models

One way of representing a joint distribution is to consider conditional dependencies between data. Conditional dependencies allow us to factorize the distribution. For example, a Markov chain is a factorization of a distribution into components that represent the conditional relationships between points that are neighboring, often in time or space. It can be decomposed in the following form.

$$p(\mathbf{y}) = p(y_n|y_{n-1}) p(y_{n-1}|y_{n-2}) \dots p(y_2|y_1)$$

By specifying conditional independencies we can reduce the parameterization required for our data, instead of directly specifying the parameters of the joint distribution, we can specify each set of parameters of the conditonal independently. This can also give an advantage in terms of interpretability. Understanding a conditional independence structure gives a structured understanding of data. If developed correctly, according to causal methodology, it can even inform how we should intervene in the system to drive a desired result (Pearl 1995).

However, a challenge arise when the data becomes more complex. Consider the graphical model shown below, used to predict the perioperative risk of *C Difficile* infection following colon surgery (Steele et al. 2012).



To capture the complexity in the interelationship between the data the graph becomes more complex, and less interpretable.

Performing Inference

As far as combining our data and our model to form our prediction, the devil is in the detail. While everything is easy to write in terms of probability densities, as we move from data and model to prediction there is that simple $\xrightarrow{\text{compute}}$ sign, which is now burying a wealth of difficulties. Each integral sign above is a high dimensional integral which will typically need approximation. Approximations also come with computational demands. As we consider more complex classes of functions, the challenges around the integrals become harder and prediction of future test data given our model and the data becomes so involved as to be impractical or impossible.

Statisticians realized these challenges early on, indeed, so early that they were actually physicists, both Laplace and Gauss worked on models such as this, in Gauss's case he made his career on prediction of the location of the lost planet (later reclassified as a asteroid, then dwarf planet), Ceres. Gauss and Laplace made use of maximum a posteriori estimates for simplifying their computations and Laplace developed Laplace's method (and invented the Gaussian density) to expand around that mode. But classical statistics needs better guarantees around model performance and interpretation, and as a result has focussed more on the *linear* model implied by

$$f(\mathbf{x}) = \mathbf{w}^{(2)\top}\boldsymbol{\phi}(\mathbf{W}_1, \mathbf{x})$$

$$\mathbf{w}^{(2)} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{C}\right).$$

The Gaussian likelihood given above implies that the data observation is related to the function by noise corruption so we have,

$$y_i = f(\mathbf{x}_i) + \epsilon_i,$$

where

$$\epsilon_i \sim \mathcal{N}\left(0, \sigma^2\right)$$

and while normally integrating over high dimensional parameter vectors is highly complex, here it is *trivial*. That is because of a property of the multivariate Gaussian.

Gaussian processes are initially of interest because

1. linear Gaussian models are easier to deal with
2. Even the parameters *within* the process can be handled, by considering a particular limit.

Let's first of all review the properties of the multivariate Gaussian distribution that make linear Gaussian models easier to deal with. We'll return to the, perhaps surprising, result on the parameters within the nonlinearity, $\boldsymbol{\theta}$, shortly.

To work with linear Gaussian models, to find the marginal likelihood all you need to know is the following rules. If

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \boldsymbol{\epsilon},$$

where $\mathbf{y}$, $\mathbf{x}$ and $\boldsymbol{\epsilon}$ are vectors and we assume that $\mathbf{x}$ and $\boldsymbol{\epsilon}$ are drawn from multivariate Gaussians,

$$\mathbf{x} \sim \mathcal{N}\left(\boldsymbol{\mu}, \mathbf{C}\right) \tag{1}$$
$$\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Sigma}\right) \tag{2}$$

then we know that $\mathbf{y}$ is also drawn from a multivariate Gaussian with,

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{W}\boldsymbol{\mu}, \mathbf{W}\mathbf{C}\mathbf{W}^\top + \boldsymbol{\Sigma}\right).$$

With apprioriately defined covariance, $\boldsymbol{\Sigma}$, this is actually the marginal likelihood for Factor Analysis, or Probabilistic Principal Component Analysis (Tipping and Bishop 1999), because we integrated out the inputs (or *latent* variables they would be called in that case).

However, we are focussing on what happens in models which are non-linear in the inputs, whereas the above would be *linear* in the inputs. To consider these, we introduce a matrix, called the design matrix. We set each activation function computed at each data point to be

$$\phi_{i,j} = \phi(\mathbf{w}_j^{(1)}, \mathbf{x}_i)$$

and define the matrix of activations (known as the *design matrix* in statistics) to be,

$$\boldsymbol{\Phi} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \dots & \phi_{1,h} \\ \phi_{1,2} & \phi_{1,2} & \dots & \phi_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n,1} & \phi_{n,2} & \dots & \phi_{n,h} \end{bmatrix}.$$

By convention this matrix always has n rows and h columns, now if we define the vector of all noise corruptions, $\boldsymbol{\epsilon} = \left[\epsilon_1, \dots \epsilon_n\right]^\top$.

If we define the prior distribution over the vector $\mathbf{w}$ to be Gaussian,

$$\mathbf{w} \sim \mathcal{N}\left(\mathbf{0}, \alpha\mathbf{I}\right),$$

then we can use rules of multivariate Gaussians to see that,

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{0}, \alpha\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2\mathbf{I}\right).$$

In other words, our training data is distributed as a multivariate Gaussian, with zero mean and a covariance given by

$$\mathbf{K} = \alpha\boldsymbol{\Phi}\boldsymbol{\Phi}^\top + \sigma^2\mathbf{I}.$$

This is an $n \times n$ size matrix. Its elements are in the form of a function. The maths shows that any element, index by *i* and *j*, is a function *only* of inputs associated with data points *i* and *j*, $\mathbf{y}_i$, $\mathbf{y}_j$.
$k_{i,j} = k\left(\mathbf{x}_i, \mathbf{x}_j\right)$

If we look at the portion of this function associated only with $f(\cdot)$, i.e. we remove the noise, then we can write down the covariance associated with our neural network,

$$k_f\left(\mathbf{x}_i, \mathbf{x}_j\right) = \alpha \boldsymbol{\phi}(\mathbf{W}_1, \mathbf{x}_i)^\top \boldsymbol{\phi}\left(\mathbf{W}_1, \mathbf{x}_j\right)$$

so the elements of the covariance or *kernel* matrix are formed by inner products of the rows of the *design matrix*.

Gaussian Process

This is the essence of a Gaussian process. Instead of making assumptions about our density over each data point, y_i as i.i.d. we make a joint Gaussian assumption over our data. The covariance matrix is now a function of both the parameters of the activation function, $\mathbf{V}$, and the input variables, $\mathbf{X}$. This comes about through integrating out the parameters of the model, $\mathbf{w}$.

Basis Functions

We can basically put anything inside the basis functions, and many people do. These can be deep kernels (Cho and Saul 2009) or we can learn the parameters of a convolutional neural network inside there.

Viewing a neural network in this way is also what allows us to beform sensible *batch* normalizations (Ioffe and Szegedy 2015).

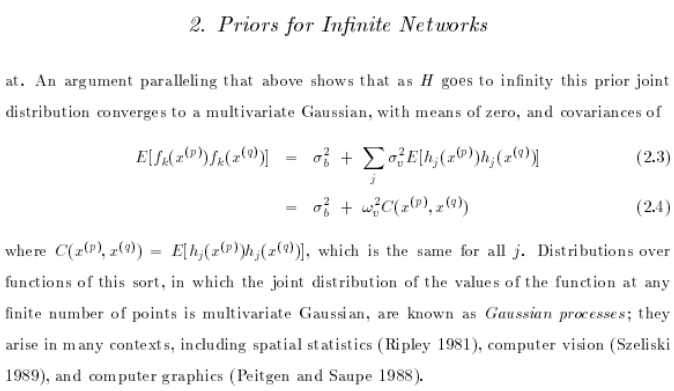
Non-degenerate Gaussian Processes

The process described above is degenerate. The covariance function is of rank at most h and since the theoretical amount of data could always increase $n \rightarrow \infty$, the covariance function is not full rank. This means as we increase the amount of data to infinity, there will come a point where we can't normalize the process because the multivariate Gaussian has the form,

$$\mathcal{N}\left(\mathbf{f}|\mathbf{0}, \mathbf{K}\right) = \frac{1}{\left(2\pi\right)^{\frac{n}{2}} \det \mathbf{K}^{\frac{1}{2}}} \exp\left(-\frac{\mathbf{f}^\top \mathbf{K} \mathbf{f}}{2}\right)$$

and a non-degenerate kernel matrix leads to $\det \mathbf{K} = 0$ defeating the normalization (it's equivalent to finding a projection in the high dimensional Gaussian where the variance of the the resulting univariate Gaussian is zero, i.e. there is a null space on the covariance, or alternatively you can imagine there are one or more directions where the Gaussian has become the delta function).

In the machine learning field, it was Radford Neal (Neal 1994) that realized the potential of the next step. In his 1994 thesis, he was considering Bayesian neural networks, of the type we described above, and in considered what would happen if you took the number of hidden nodes, or neurons, to infinity, i.e. $h \rightarrow \infty$.



2. *Priors for Infinite Networks*

at. An argument paralleling that above shows that as H goes to infinity this prior joint distribution converges to a multivariate Gaussian, with means of zero, and covariances of

$$E[f_k(x^{(p)}) f_k(x^{(q)})] = \sigma_b^2 + \sum_j \sigma_v^2 E[h_j(x^{(p)}) h_j(x^{(q)})] \quad (2.3)$$

$$= \sigma_b^2 + \omega_v^2 C(x^{(p)}, x^{(q)}) \quad (2.4)$$

where $C(x^{(p)}, x^{(q)}) = E[h_j(x^{(p)}) h_j(x^{(q)})]$, which is the same for all j. Distributions over functions of this sort, in which the joint distribution of the values of the function at any finite number of points is multivariate Gaussian, are known as *Gaussian processes*; they arise in many contexts, including spatial statistics (Ripley 1981), computer vision (Szeliski 1989), and computer graphics (Peitgen and Saupe 1988).

Page 37 of Radford Neal's 1994 thesis

In loose terms, what Radford considers is what happens to the elements of the covariance function,

$$
\begin{aligned}
k_f\left(\mathbf{x}_i, \mathbf{x}_j\right) &= \alpha \boldsymbol{\phi}(\mathbf{W}_1, \mathbf{x}_i)^\top \boldsymbol{\phi}\left(\mathbf{W}_1, \mathbf{x}_j\right) \\
&= \alpha \sum_k \phi\left(\mathbf{w}_k^{(1)}, \mathbf{x}_i\right) \phi\left(\mathbf{w}_k^{(1)}, \mathbf{x}_j\right)
\end{aligned}
$$

if instead of considering a finite number you sample infinitely many of these activation functions, sampling parameters from a prior density, $p(\mathbf{v})$, for each one,

$$
k_f\left(\mathbf{x}_i, \mathbf{x}_j\right) = \alpha \int \phi\left(\mathbf{w}^{(1)}, \mathbf{x}_i\right) \phi\left(\mathbf{w}^{(1)}, \mathbf{x}_j\right) p(\mathbf{w}^{(1)}) \mathrm{d}\mathbf{w}^{(1)}
$$

And that's not *only* for Gaussian $p(\mathbf{v})$. In fact this result holds for a range of activations, and a range of prior densities because of the *central limit theorem*.

To write it in the form of a probabilistic program, as long as the distribution for $\boldsymbol{\phi}_i$ implied by this short probabilistic program,

$$
\begin{aligned}
\mathbf{v} &\sim p(\cdot) \\
\boldsymbol{\phi}_i &= \boldsymbol{\phi}\left(\mathbf{v}, \mathbf{x}_i\right),
\end{aligned}
$$

has finite variance, then the result of taking the number of hidden units to infinity, with appropriate scaling, is also a Gaussian process.

Further Reading

To understand this argument in more detail, I highly recommend reading chapter 2 of Neal's thesis (Neal 1994), which remains easy to read and clear today. Indeed, for readers interested in Bayesian neural networks, both Raford Neal's and David MacKay's PhD thesis (MacKay 1992) remain essential reading. Both theses embody a clarity of thought, and an ability to weave together threads from different fields that was the business of machine learning in the 1990s. Radford and David were also pioneers in making their software widely available and publishing material on the web.

Bayesian Inference by Rejection Sampling

One view of Bayesian inference is to assume we are given a mechanism for generating samples, where we assume that mechanism is representing on accurate view on the way we believe the world works.

This mechanism is known as our *prior* belief.

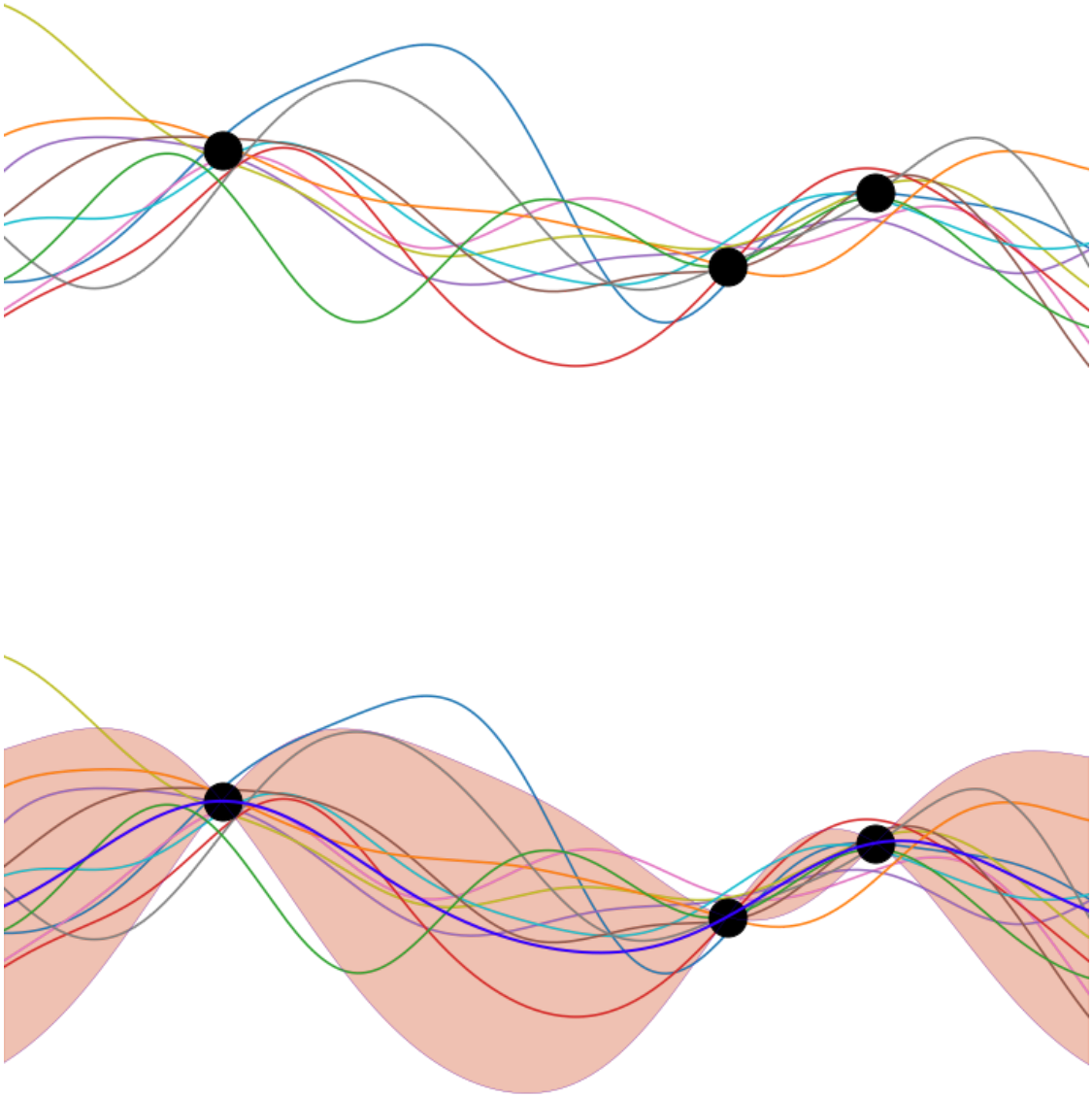
We combine our prior belief with our observations of the real world by discarding all those samples that are inconsistent with our prior. The *likelihood* defines mathematically what we mean by inconsistent with the prior. The higher the noise level in the likelihood, the looser the notion of consistent.

The samples that remain are considered to be samples from the *posterior*.

This approach to Bayesian inference is closely related to two sampling techniques known as *rejection sampling* and *importance sampling*. It is realized in practice in an approach known as *approximate Bayesian computation* (ABC) or likelihood-free inference.

In practice, the algorithm is often too slow to be practical, because most samples will be inconsistent with the data and as a result the mechanism has to be operated many times to obtain a few posterior samples.

However, in the Gaussian process case, when the likelihood also assumes Gaussian noise, we can operate this mechanims mathematically, and obtain the posterior density *analytically*. This is the benefit of Gaussian processes.



One view of Bayesian inference is we have a machine for generating samples (the prior*), and we discard all samples inconsistent with our data, leaving the samples of interest (the* posterior*). The Gaussian process allows us to do this analytically.*

Sampling a Function

We will consider a Gaussian distribution with a particular structure of covariance matrix. We will generate *one* sample from a 25-dimensional Gaussian density.

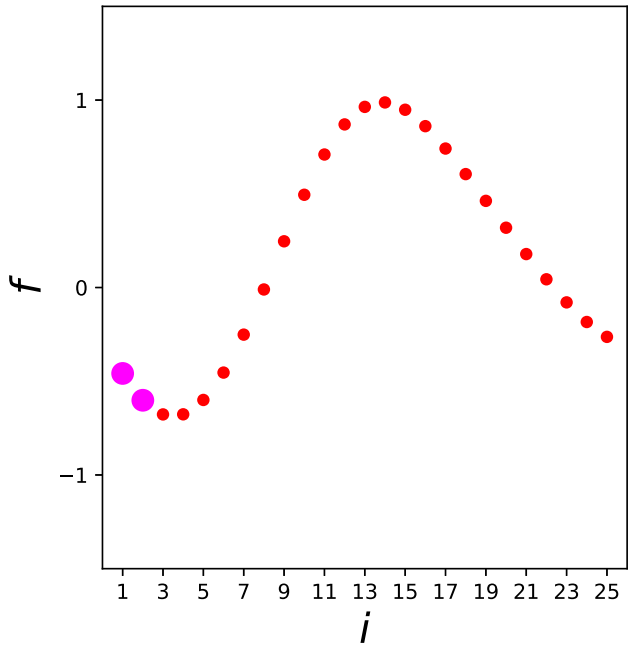
$$\mathbf{f} = \left[f_1, f_2 \dots f_{25}\right].$$

in the figure below we plot these data on the *y*-axis against their *indices* on the *x*-axis.

```
from mlai import Kernel
```

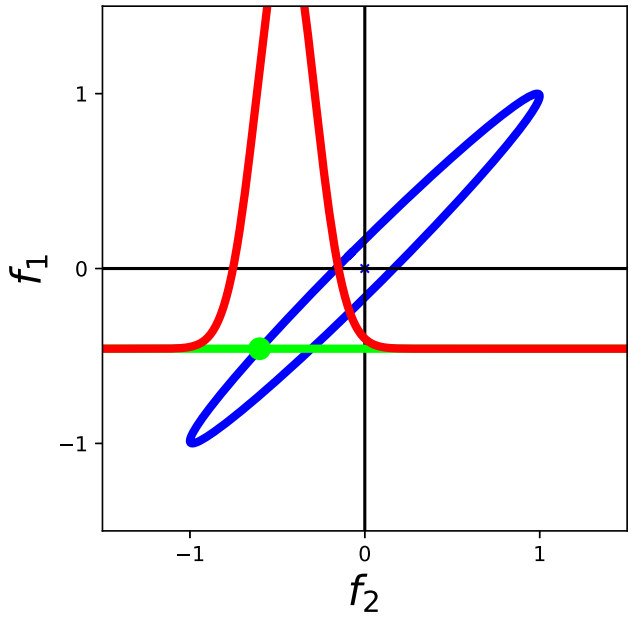
```
from mlai import polynomial_cov
```

```
from mlai import exponentiated_quadratic
```



$$
\begin{matrix} & & i \\ i' & \begin{bmatrix} 1.0 & 0.986 \\ 0.986 & 1.0 \end{bmatrix} \end{matrix}
$$

A 25 dimensional correlated random variable (values ploted against index)



$$
\begin{matrix} & & i \\ i' & \begin{bmatrix} 1.0 & 0.986 \\ 0.986 & 1.0 \end{bmatrix} \end{matrix}
$$

The joint Gaussian over f_1 and f_2 along with the conditional distribution of f_2 given f_1

Uluru

When viewing these contour plots, I sometimes find it helpful to think of Uluru, the prominent rock formation in Australia. The rock rises above the surface of the plane, just like a probability density rising above the zero line. The rock is three dimensional, but when we view Uluru from the classical position, we are looking at one side of it. This is equivalent to viewing the marginal density.

The joint density can be viewed from above, using contours. The conditional density is equivalent to *slicing* the rock. Uluru is a holy rock, so this has to be an imaginary slice. Imagine we cut down a vertical plane orthogonal to our view point (e.g. coming across our view point). This would give a profile of the rock, which when renormalized, would give us the conditional distribution, the value of conditioning would be the location of the slice in the direction we are facing.

Prediction with Correlated Gaussians

Of course in practice, rather than manipulating mountains physically, the advantage of the Gaussian density is that we can perform these manipulations mathematically.

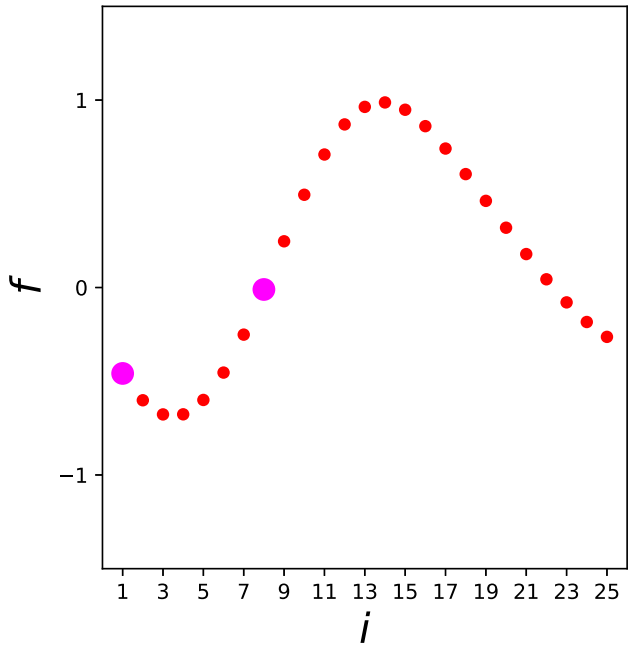
Prediction of f_2 given f_1 requires the *conditional density*, $p(f_2|f_1)$.Another remarkable property of the Gaussian density is that this conditional distribution is *also* guaranteed to be a Gaussian density. It has the form,

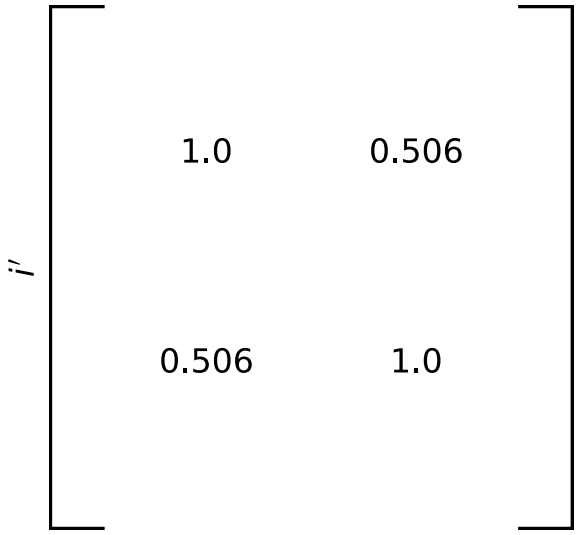
$$p(f_2|f_1) = \mathcal{N}\left(f_2|\frac{k_{1,2}}{k_{1,1}}f_1, k_{2,2} - \frac{k_{1,2}^2}{k_{1,1}}\right)$$

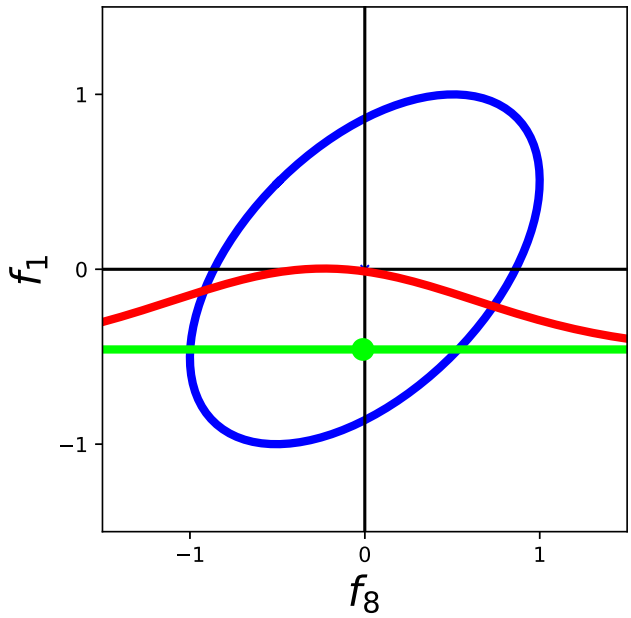
where we have assumed that the covariance of the original joint density was given by

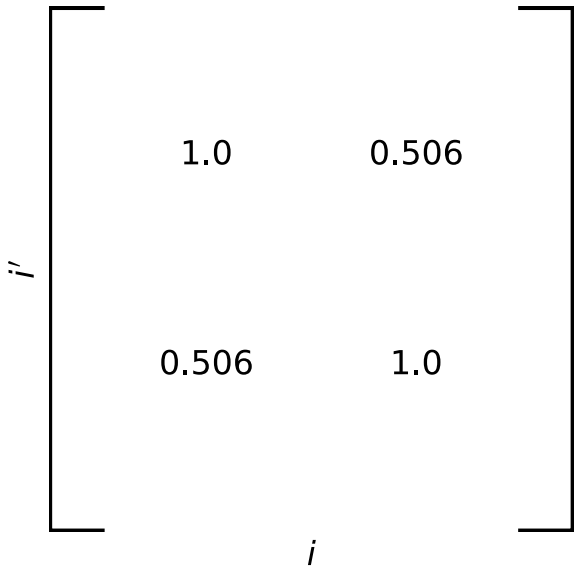
$$\mathbf{K} = \begin{bmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2}. \end{bmatrix}$$

Using these formulae we can determine the conditional density for any of the elements of our vector $\mathbf{f}$. For example, the variable f_8 is less correlated with f_1 than f_2. If we consider this variable we see the conditional density is more diffuse.









The joint Gaussian over f_1 and f_8 along with the conditional distribution of f_8 given f_1

- Covariance function, $\mathbf{K}$
- Determines properties of samples.
- Function of $\mathbf{X}$,

$$k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- Posterior mean

$$f_D(\mathbf{x}_*) = \mathbf{k}(\mathbf{x}_*, \mathbf{X})\mathbf{K}^{-1}\mathbf{y}$$

- Posterior covariance

$$\mathbf{C}_* = \mathbf{K}_{*,*} - \mathbf{K}_{*,\mathbf{f}}\mathbf{K}^{-1}\mathbf{K}_{\mathbf{f},*}$$

- Posterior mean

$$f_D(\mathbf{x}_*) = \mathbf{k}(\mathbf{x}_*, \mathbf{X})\boldsymbol{\alpha}$$

- Posterior covariance

$$\mathbf{C}_* = \mathbf{K}_{*,*} - \mathbf{K}_{*,\mathbf{f}}\mathbf{K}^{-1}\mathbf{K}_{\mathbf{f},*}$$

Exponentiated Quadratic Covariance

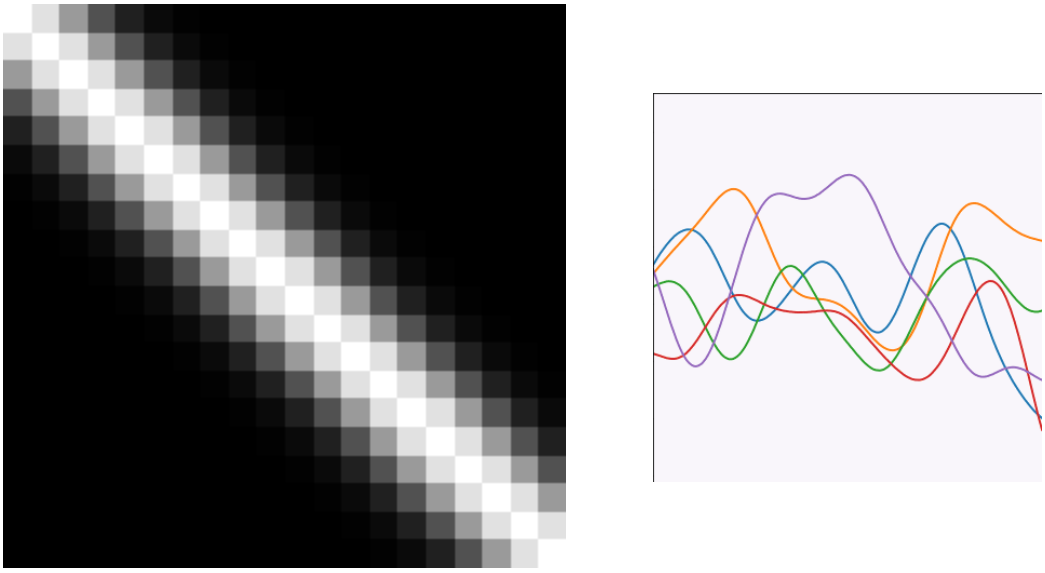
The exponentiated quadratic covariance, also known as the Gaussian covariance or the RBF covariance and the squared exponential. Covariance between two points is related to the negative exponential of the squared distnace between those points. This covariance function can be derived in a few different ways: as the infinite limit of a radial basis function neural network, as diffusion in the heat equation, as a Gaussian filter in *Fourier space* or as the composition as a series of linear filters applied to a base function.

The covariance takes the following form,

$$k(\mathbf{x}, \mathbf{x}') = \alpha \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\ell^2}\right)$$

where ℓ is the *length scale* or *time scale* of the process and *α* represents the overall process variance.

$$k(\mathbf{x}, \mathbf{x}') = \alpha \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\ell^2}\right)$$



The exponentiated quadratic covariance function.

Olympic Marathon Data

- Gold medal times for Olympic Marathon since 1896.
- Marathons before 1924 didn’t have a standardised distance.
- Present results using pace per km.
- In 1904 Marathon was badly organised leading to very slow times.

Image from Wikimedia Commons

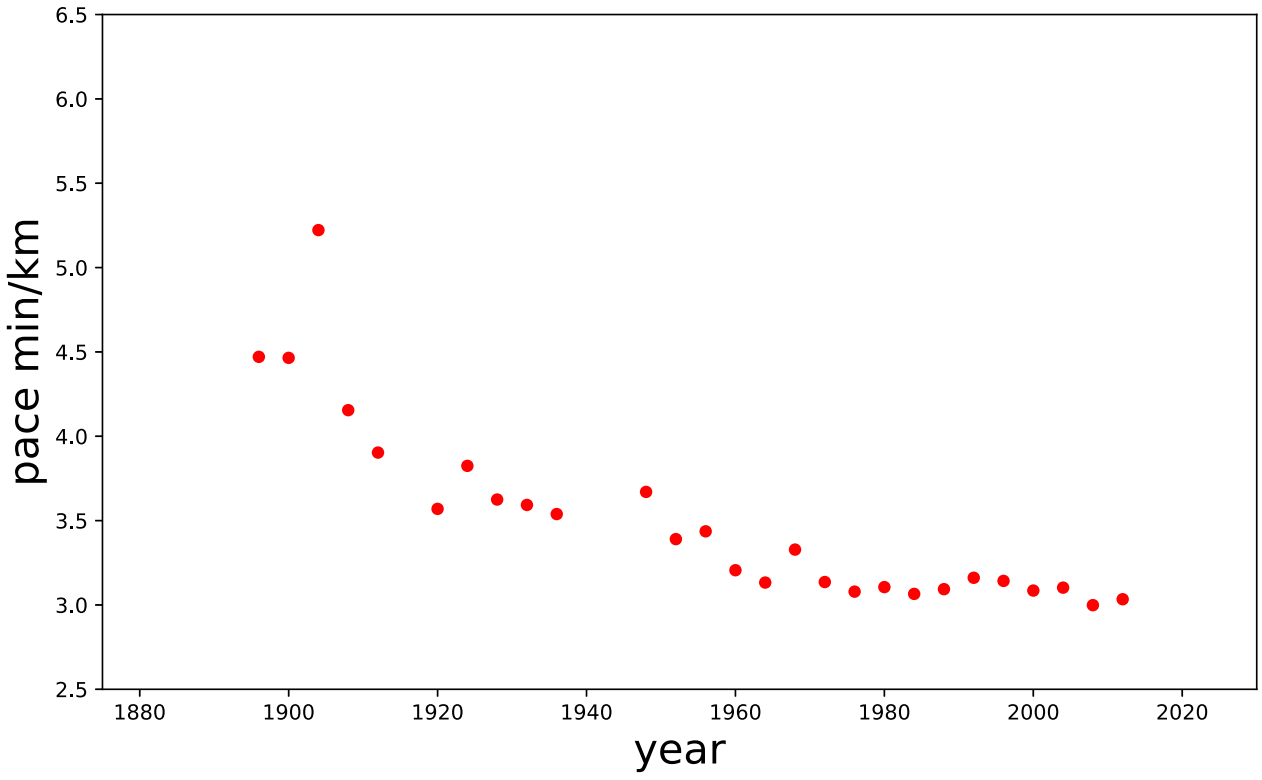
http://bit.ly/16kMKHQ

The first thing we will do is load a standard data set for regression modelling. The data consists of the pace of Olympic Gold Medal Marathon winners for the Olympics from 1896 to present. First we load in the data and plot.

```
import numpy as np
import pods
```

```
data = pods.datasets.olympic_marathon_men()
x = data['X']
y = data['Y']

offset = y.mean()
scale = np.sqrt(y.var())
```



Things to notice about the data include the outlier in 1904, in this year, the olympics was in St Louis, USA. Organizational problems and challenges with dust kicked up by the cars following the race meant that participants got lost, and only very few participants completed.

More recent years see more consistently quick marathons.

Data is fine for answering very specific questions, like "Who won the Olympic Marathon in 2012?", because we have that answer stored, however, we are not given the answer to many other questions. For example, Alan Turing was a formidable marathon runner, in 1946 he ran a time 2 hours 46 minutes (just under four minutes per kilometer, faster than I and most of the other Endcliffe Park Run runners can do 5 km). What is the probability he would have won an Olympics if one had been held in 1946?

ATHLETICS

MARATHON AND DECATHLON CHAMPIONSHIPS

The Amateur Athletic Association championships for this year were concluded at Loughborough College Stadium, Leicestershire, on Saturday, with the second, and last, day of the Decathlon and the decision of the Marathon championship.

MARATHON CHAMPIONSHIP (26 miles 385 yds.) (record: 2hrs. 30min. 57.6sec., by H. W. Payne, Windsor to Stamford Bridge, on July 5, 1929; standard time: 3hrs. 5min.).—J. T. Holden (Tipton Harriers), 2hrs. 33min. 20 1-5sec., 1; T. Richards (South London Harriers), 2hrs. 36mni. 7sec., 2; D. McNab Robertson (Maryhill Harriers, Glasgow), 2hrs. 37min. 54 3-5sec., 3; J. E. Farrell (Maryhill Harriers), 2hrs. 39min. 46 2-5sec., 4; Dr. A. M. Turing (Walton A.C.), 2hrs. 46min. 3sec., 5; L. H. Griffiths (Reading A.C.), 2hrs. 47min. 50 2-5sec., 6.

DECATHLON CHAMPIONSHIP.—H. J. Moesgaard-Kjeldsen (Polytechnic Harriers, London), 5,965 points, 1; Captain H. Whittle (Army and Reading A.C.), 5,650, 2;

Alan Turing, in 1946 he was only 11 minutes slower than the winner of the 1948 games. Would he have won a hypothetical games held in 1946? Source: Alan Turing Internet Scrapbook

Our first objective will be to perform a Gaussian process fit to the data, we'll do this using the GPy software.

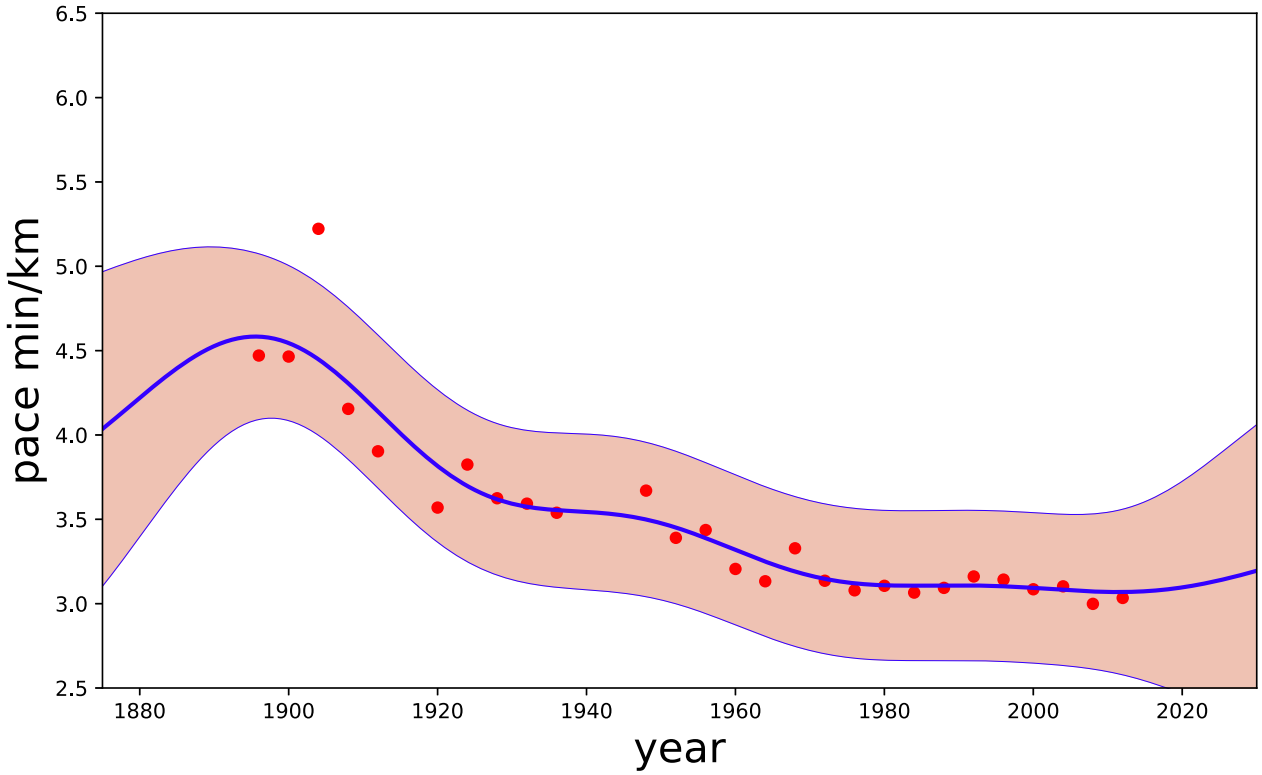
```
import GPy
```

```
m_full = GPy.models.GPRegression(x,yhat)
_ = m_full.optimize() # Optimize parameters of covariance function
```

The first command sets up the model, then `m_full.optimize()` optimizes the parameters of the covariance function and the noise level of the model. Once the fit is complete, we'll try creating some test points, and computing the output of the GP model in terms of the mean and standard deviation of the posterior functions between 1870 and 2030. We plot the mean function and the standard deviation at 200 locations. We can obtain the predictions using `y_mean, y_var = m_full.predict(xt)`

```
xt = np.linspace(1870,2030,200)[:,np.newaxis]
yt_mean, yt_var = m_full.predict(xt)
yt_sd=np.sqrt(yt_var)
```

Now we plot the results using the helper function in `teaching_plots`.



Fit Quality

In the fit we see that the error bars (coming mainly from the noise variance) are quite large. This is likely due to the outlier point in 1904, ignoring that point we can see that a tighter fit is obtained. To see this making a version of the model, `m_clean`, where that point is removed.

```
x_clean=np.vstack((x[0:2, :], x[3:, :]))
y_clean=np.vstack((y[0:2, :], y[3:, :]))

m_clean = GPy.models.GPRegression(x_clean,y_clean)
_ = m_clean.optimize()
```

Can we determine covariance parameters from the data?

$$\mathcal{N}\left(\mathbf{y}|\mathbf{0},\mathbf{K}\right) = \frac{1}{(2\pi)^{\frac{n}{2}}\det\mathbf{K}^{\frac{1}{2}}}\exp\left(-\frac{\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}}{2}\right)$$

$$\mathcal{N}\left(\mathbf{y}|\mathbf{0},\mathbf{K}\right) = \frac{1}{(2\pi)^{\frac{n}{2}}\det\mathbf{K}^{\frac{1}{2}}}\exp\left(-\frac{\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}}{2}\right)$$

$$\log\mathcal{N}\left(\mathbf{y}|\mathbf{0},\mathbf{K}\right) = -\frac{1}{2}\log\det\mathbf{K} - \frac{\mathbf{y}^\top\mathbf{K}^{-1}\mathbf{y}}{2} - \frac{n}{2}\log 2\pi$$

\errorFunction(\parameterVector) = {\color{yellow} \frac{1}{2}\log\det{\kernelMatrix}} + {\color{cyan} \frac{\dataVector^{\top}\kernelMatrix^{-1

The parameters are *inside* the covariance function (matrix).

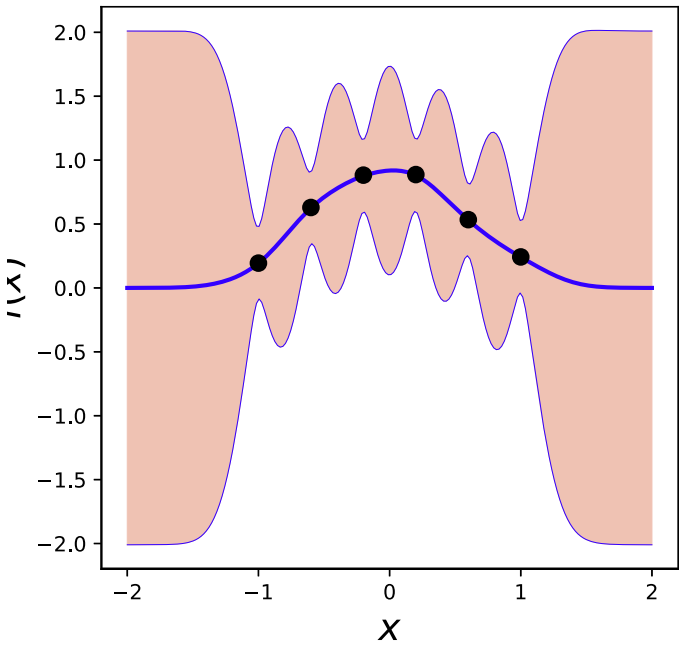
$$k_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j; \boldsymbol{\theta})$$

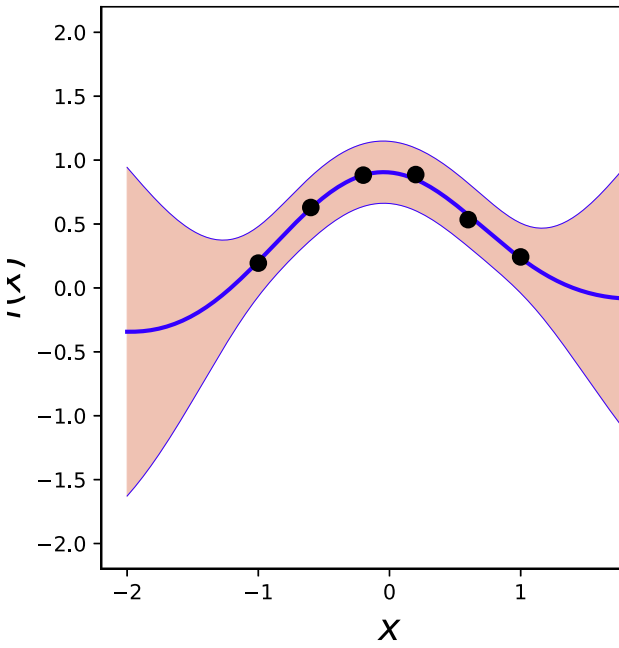
$$\mathbf{K} = \mathbf{R}\boldsymbol{\Lambda}^2\mathbf{R}^\top$$

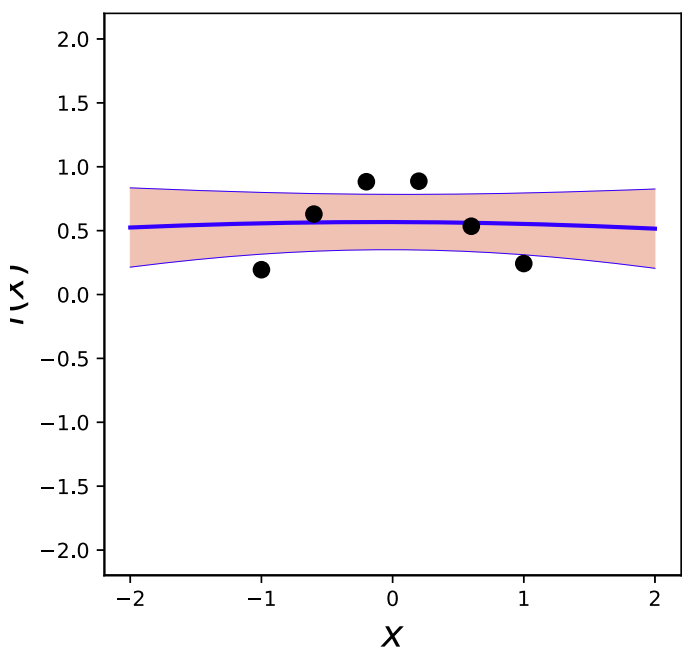
```
gpoptimizePlot1
```

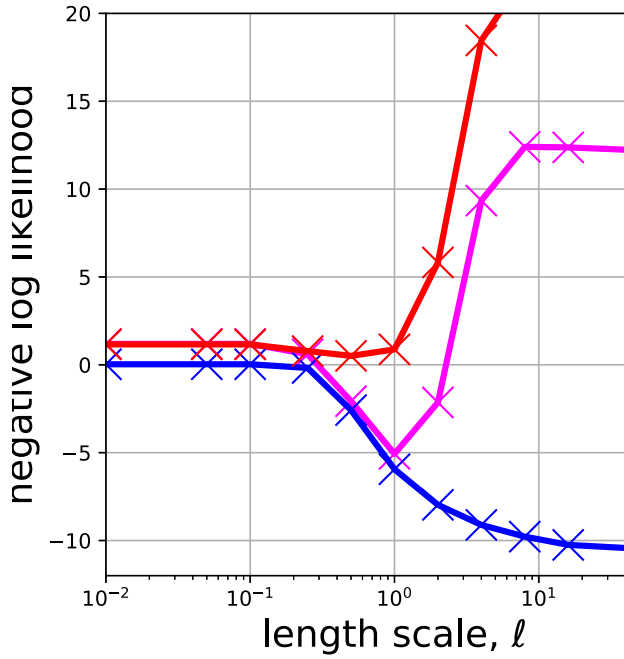
$\boldsymbol{\Lambda}$ represents distance on axes. $\mathbf{R}$ gives rotation.

- $\boldsymbol{\Lambda}$ is *diagonal*, $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$.
- Useful representation since $\det \mathbf{K} = \det \boldsymbol{\Lambda}^2 = \det \boldsymbol{\Lambda}^2$.









Variation in the data fit term, the capacity term and the negative log likelihood for different lengthscales.

Della Gatta Gene Data

- Given given expression levels in the form of a time series from Della Gatta et al. (2008).

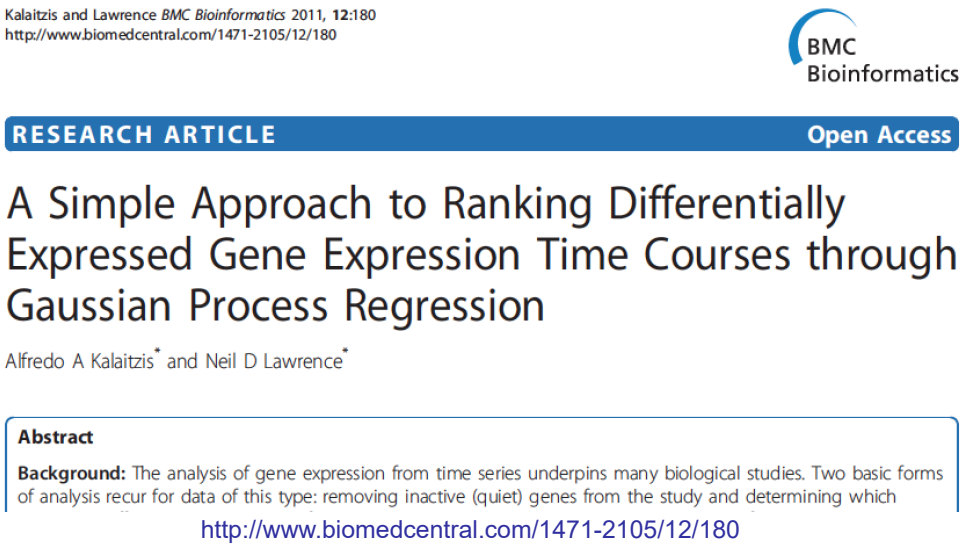
```
import numpy as np
import pods
```

```
data = pods.datasets.della_gatta_TRP63_gene_expression(data_set='della_gatta',gene_number=937)

x = data['X']
y = data['Y']

offset = y.mean()
scale = np.sqrt(y.var())
```

- Want to detect if a gene is expressed or not, fit a GP to each gene Kalaitzis and Lawrence (2011).



Kalaitzis and Lawrence *BMC Bioinformatics* 2011, **12**:180
http://www.biomedcentral.com/1471-2105/12/180

BMC Bioinformatics

RESEARCH ARTICLE Open Access

A Simple Approach to Ranking Differentially Expressed Gene Expression Time Courses through Gaussian Process Regression

Alfredo A Kalaitzis* and Neil D Lawrence*

Abstract

Background: The analysis of gene expression from time series underpins many biological studies. Two basic forms of analysis recur for data of this type: removing inactive (quiet) genes from the study and determining which

http://www.biomedcentral.com/1471-2105/12/180

Our first objective will be to perform a Gaussian process fit to the data, we'll do this using the GPy software.

```
import GPy
```

```
m_full = GPy.models.GPRegression(x,yhat)
m_full.kern.lengthscale=50
_ = m_full.optimize() # Optimize parameters of covariance function
```

Initialize the length scale parameter (which here actually represents a *time scale* of the covariance function to a reasonable value. Default would be 1, but here we set it to 50 minutes, given points are arriving across zero to 250 minutes.

```
xt = np.linspace(-20,260,200)[:,np.newaxis]
yt_mean, yt_var = m_full.predict(xt)
yt_sd=np.sqrt(yt_var)
```

Now we plot the results using the helper function in `teaching_plots`.

Now we try a model initialized with a longer length scale.

```
m_full2 = GPy.models.GPRegression(x,yhat)
m_full2.kern.lengthscale=2000
_ = m_full2.optimize() # Optimize parameters of covariance function
```

Now we try a model initialized with a lower noise.

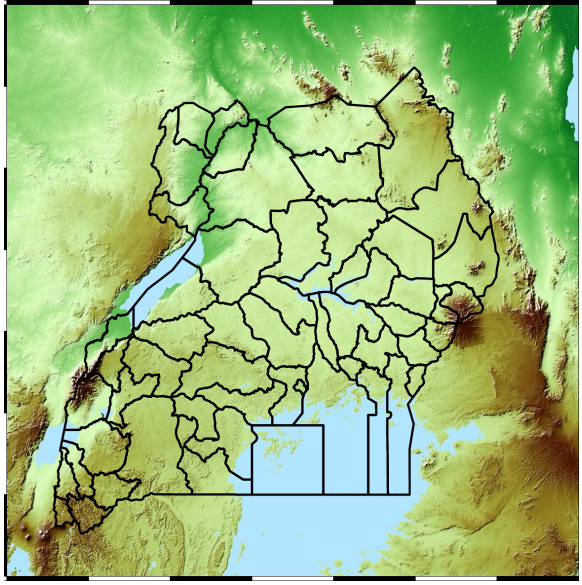
```
m_full3 = GPy.models.GPRegression(x,yhat)
m_full3.kern.lengthscale=20
m_full3.likelihood.variance=0.001
_ = m_full3.optimize() # Optimize parameters of covariance function
```

Example: Prediction of Malaria Incidence in Uganda

As an example we'll consider the prediction of Malaria incidence in Uganda. For the purposes of this study malaria reports come in two forms, HMIS reports from health centres and Sentinel data, which is curated by the WHO. There are limited sentinel sites and many HMIS sites.

The work is from Ricardo Andrade Pacheco's PhD thesis, completed in collaboration with John Quinn and Martin Mubangizi (Andrade-Pacheco et al. 2014,Mubangizi et al. (2014),). John and Martin were initally from the AI-DEV group from the University of Makerere in Kampala and more latterly they were based at UN Global Pulse in Kampala.

Malaria data is spatial data. Uganda is split into districts, and health reports can be found for each district. This suggests that models such as conditional random fields could be used for spatial modelling, but there are two complexities with this. First of all, occasionally districts split into two. Secondly, sentinel sites are a specific location within a district, such as Nagongera which is a sentinel site based in the Tororo district.



Data SRTM/NASA from https://dds.cr.usgs.gov/srtm/version2_1

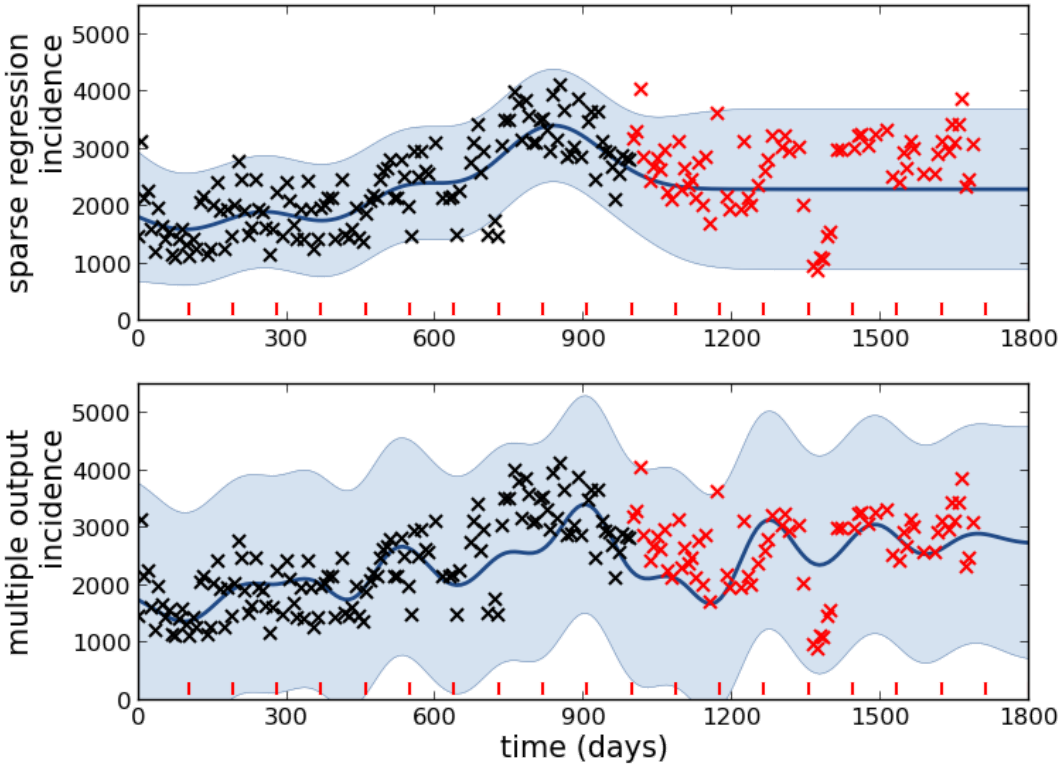
(Andrade-Pacheco et al. 2014,Mubangizi et al. (2014))

THe Kapchorwa District, home district of Stephen Kiprotich.

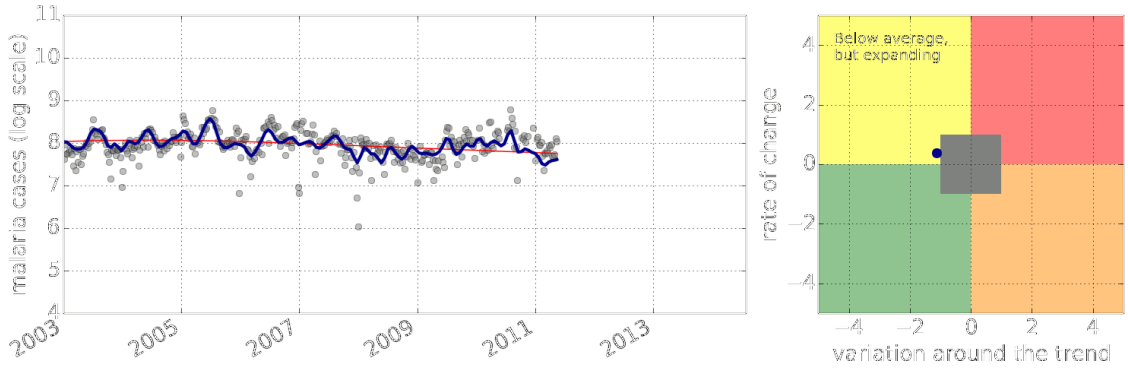
Stephen Kiprotich, the 2012 gold medal winner from the London Olympics, comes from Kapchorwa district, in eastern Uganda, near the border with Kenya.

The Tororo District, where the sentinel site, Nagongera is located

Mubende


sparse regression incidence
multiple output incidence
0
1000
2000
3000
4000
5000
0
300
600
900
1200
1500
1800
time (days)

Early Warning Systems



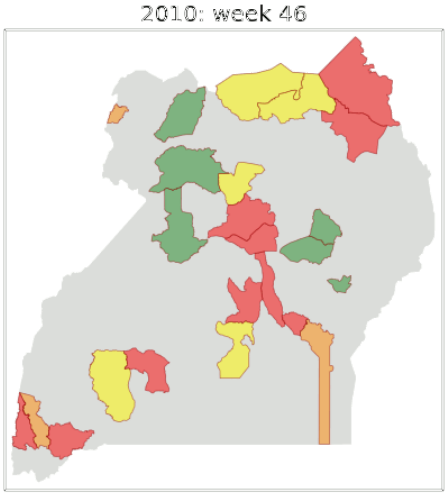
Estimate of the current disease situation in the Kabarole district over time. Estimate is constructed with a Gaussian process with an additive covariance funciton.

Health monitoring system for the Kabarole district. Here we have fitted the reports with a Gaussian process with an additive covariance function. It has two components, one is a long time scale component (in red above) the other is a short time scale component (in blue).

Monitoring proceeds by considering two aspects of the curve. Is the blue line (the short term report signal) above the red (which represents the long term trend? If so we have higher than expected reports. If this is the case *and* the gradient is still positive (i.e. reports are going up) we encode this with a *red* color. If it is the case and the gradient of the blue line is negative (i.e. reports are going down) we encode this with an *amber* color. Conversely, if the blue line is below the red *and* decreasing, we color *green*. On the other hand if it is below red but increasing, we color *yellow*.

This gives us an early warning system for disease. Red is a bad situation getting worse, amber is bad, but improving. Green is good and getting better and yellow good but degrading.

Finally, there is a gray region which represents when the scale of the effect is small.



The map of Ugandan districts with an overview of the Malaria situation in each district.

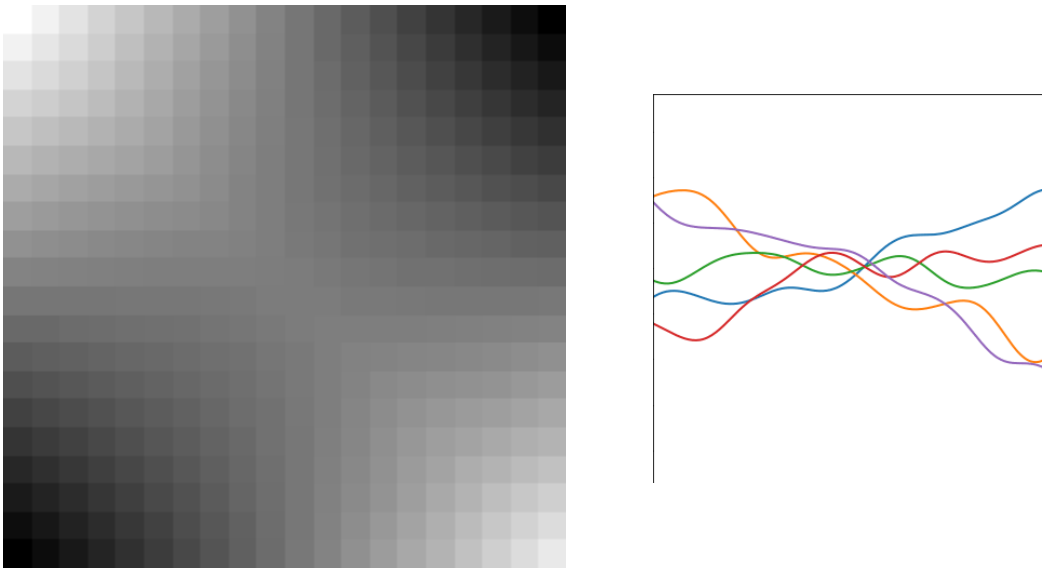
These colors can now be observed directly on a spatial map of the districts to give an immediate impression of the current status of the disease across the country.

Additive Covariance

An additive covariance function is derived from considering the result of summing two Gaussian processes together. If the first Gaussian process is *g*(·), governed by covariance $k_g(\cdot,\cdot)$ and the second process is *h*(·), governed by covariance $k_h(\cdot,\cdot)$ then the combined process *f*(·)=*g*(·)+*h*(·) is govererned by a covariance function,

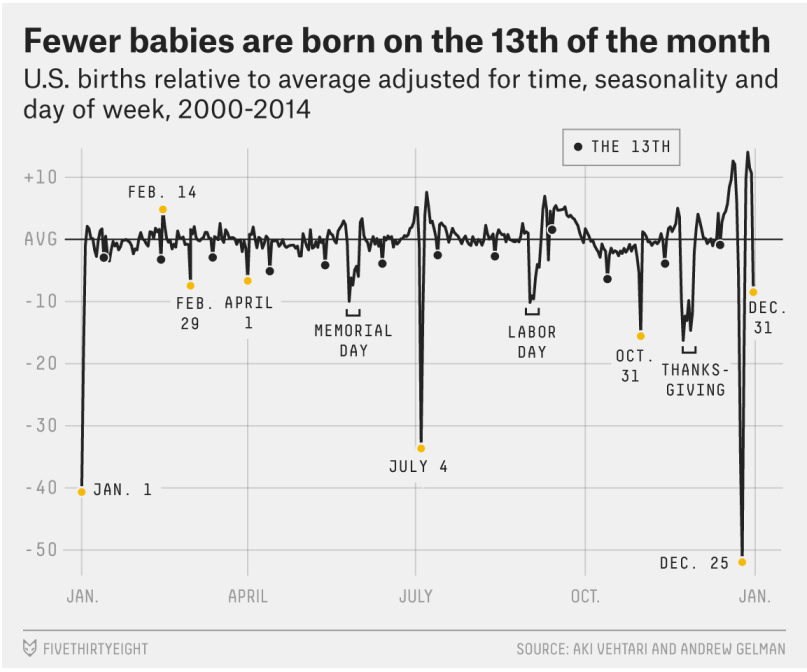
$$k_f(\mathbf{x},\mathbf{x}') = k_g(\mathbf{x},\mathbf{x}') + k_h(\mathbf{x},\mathbf{x}')$$

$$k_f(\mathbf{x},\mathbf{x}') = k_g(\mathbf{x},\mathbf{x}') + k_h(\mathbf{x},\mathbf{x}')$$



An additive covariance function formed by combining two exponentiated quadratic covariance functions.

{### Analysis of US Birth Rates



This is a retrospective analysis of US births by Aki Vehtari. The challenges of forecasting. Even with seasonal and weekly effects removed there are significant effects on holidays, weekends, etc.

There's a nice analysis of US birth rates by Gaussian processes with additive covariances in Gelman et al. (2013). A combination of covariance functions are used to take account of weekly and yearly trends. The analysis is summarized on the cover of the book.

Basis Function Covariance

The fixed basis function covariance just comes from the properties of a multivariate Gaussian, if we decide

$$\mathbf{f} = \boldsymbol{\Phi}\mathbf{w}$$

and then we assume

$$\mathbf{w} \sim \mathcal{N}\left(\mathbf{0}, \alpha\mathbf{I}\right)$$

then it follows from the properties of a multivariate Gaussian that

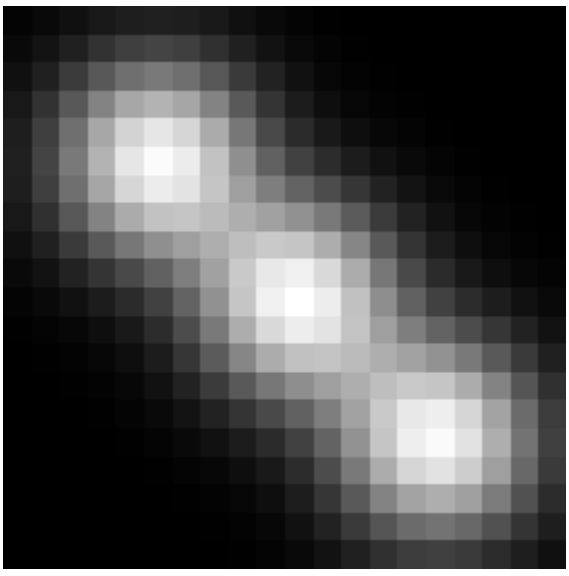
$$\mathbf{f} \sim \mathcal{N}\left(\mathbf{0}, \alpha\boldsymbol{\Phi}\boldsymbol{\Phi}^\top\right)$$

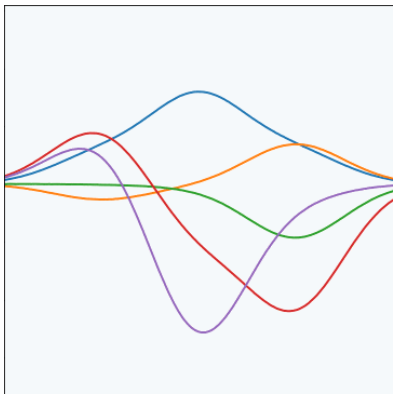
meaning that the vector of observations from the function is jointly distributed as a Gaussian process and the covariance matrix is $\mathbf{K} = \alpha\boldsymbol{\Phi}\boldsymbol{\Phi}^\top$, each element of the covariance matrix can then be found as the inner product between two rows of the basis funciton matrix.

```
from mlai import basis_cov
```

```
from mlai import radial
```

$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^\top\boldsymbol{\phi}(\mathbf{x}')$$





A covariance function based on a non-linear basis given by $\boldsymbol{\phi}(\mathbf{x})$.

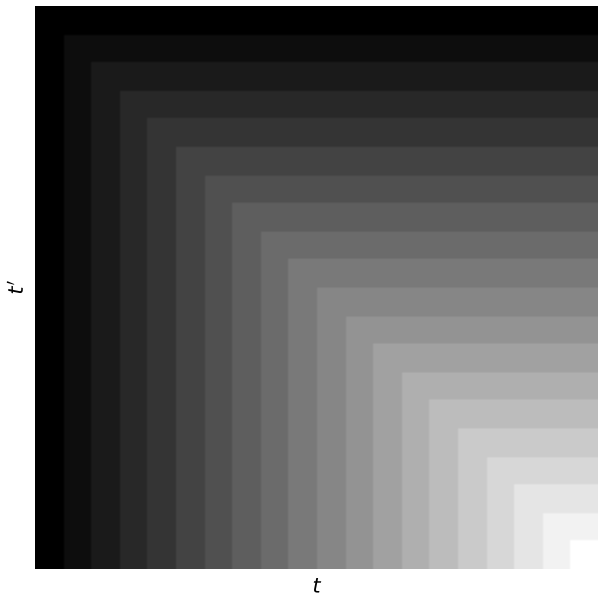
Brownian Covariance

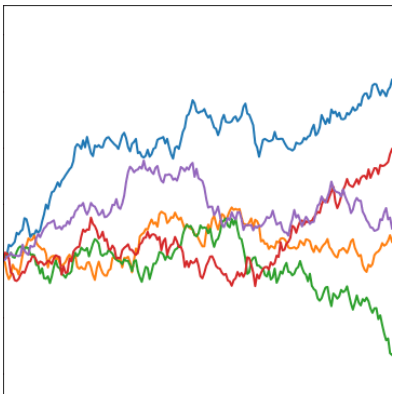
```
from mlai import brownian_cov
```

Brownian motion is also a Gaussian process. It follows a Gaussian random walk, with diffusion occuring at each time point driven by a Gaussian input. This implies it is both Markov and Gaussian. The covariance function for Brownian motion has the form

$$k(t, t') = \alpha \min(t, t')$$

$$k(t, t') = \alpha \min(t, t')$$



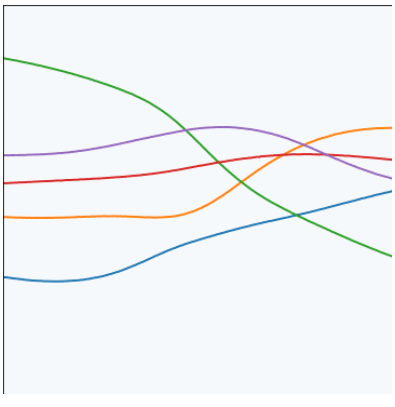


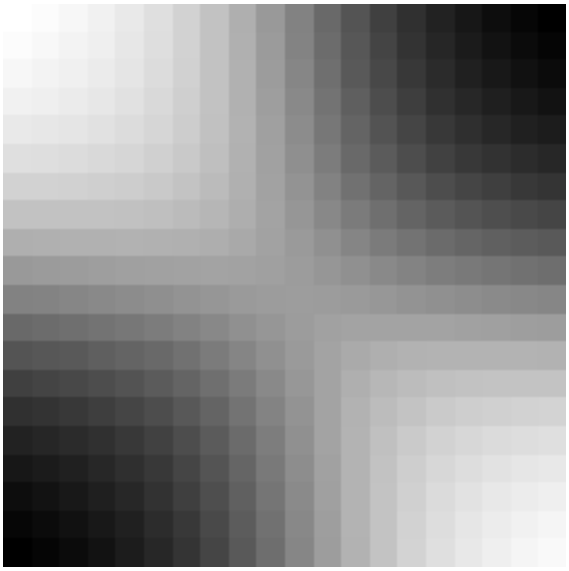
MLP Covariance

```
from mlai import mlp_cov
```

The multi-layer perceptron (MLP) covariance, also known as the neural network covariance or the arcsin covariance, is derived by considering the infinite limit of a neural network.

$$k(\mathbf{x}, \mathbf{x}') = \alpha \arcsin\left(\frac{w\mathbf{x}^\top\mathbf{x}' + b}{\sqrt{\left(w\mathbf{x}^\top\mathbf{x} + b + 1\right)\left(w\mathbf{x}'^\top\mathbf{x}' + b + 1\right)}}\right)$$



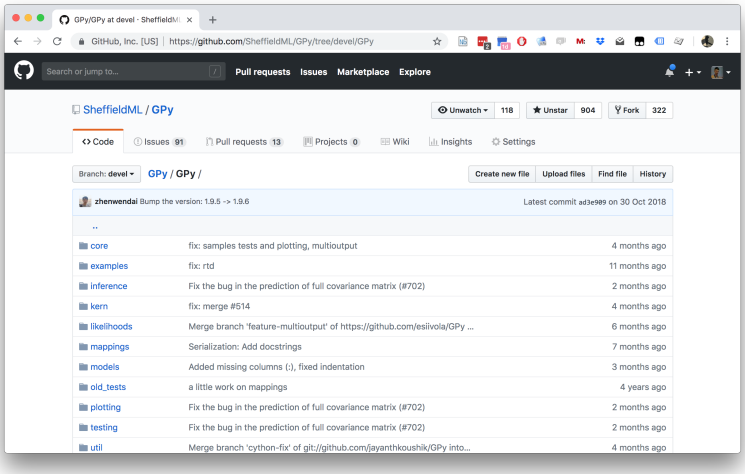


The multi-layer perceptron covariance function. This is derived by considering the infinite limit of a neural network with probit activation functions.

GPSS: Gaussian Process Summer School

If you're interested in finding out more about Gaussian processes, you can attend the Gaussian process summer school, or view the lectures and material on line. Details of the school, future events and past events can be found at the website http://gpss.cc.

GPy: A Gaussian Process Framework in Python



GPy is a BSD licensed software code base for implementing Gaussian process models in python. This allows GPs to be combined with a wide variety of software libraries.

The software itself is avaialble on GitHub and the team welcomes contributions.

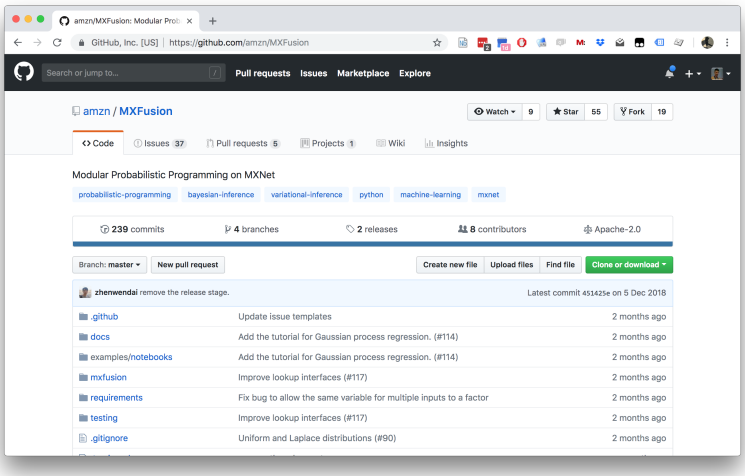
The aim for GPy is to be a probabilistic-style programming language, i.e. you specify the model rather than the algorithm. As well as a large range of covariance functions the software allows for non-Gaussian likelihoods, multivariate outputs, dimensionality reduction and approximations for larger data sets.

Other Software

GPy has inspired other software solutions, first of all GPflow, which uses Tensor Flow's automatic differentiation engine to allow rapid prototyping of new covariance functions and algorithms. More recently, GPyTorch uses PyTorch for the same purpose.

GPy itself is being restructured with MXFusion as its computational engine to give similiar capabilities.

MXFusion: Modular Probabilistic Programming on MXNet



https://github.com/amzn/MXFusion

</tr> </table>

Acknowledgments

Stefanos Eleftheriadis, John Bronskill, Hugh Salimbeni, Rich Turner, Zhenwen Dai, Javier Gonzalez, Andreas Damianou, Mark Pullin, Michael Smith, James Hensman, John Quinn, Martin Mubangizi.

References

Andrade-Pacheco, Ricardo, Martin Mubangizi, John Quinn, and Neil D. Lawrence. 2014. “Consistent Mapping of Government Malaria Records Across a Changing Territory Delimitation.” *Malaria Journal* 13 (Suppl 1). doi:10.1186/1475-2875-13-S1-P5.

Cho, Youngmin, and Lawrence K. Saul. 2009. “Kernel Methods for Deep Learning.” In *Advances in Neural Information Processing Systems 22*, edited by Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, 342–50. Curran Associates, Inc. http://papers.nips.cc/paper/3628-kernel-methods-for-deep-learning.pdf.

Della Gatta, Giusy, Mukesh Bansal, Alberto Ambesi-Impiombato, Dario Antonini, Caterina Missero, and Diego di Bernardo. 2008. “Direct Targets of the Trp63 Transcription Factor Revealed by a Combination of Gene Expression Profiling and Reverse Engineering.” *Genome Research* 18 (6). Telethon Institute of Genetics; Medicine, 80131 Naples, Italy.: 939–48. doi:10.1101/gr.073601.107.

Gelman, Andrew, John B. Carlin, Hal S. Stern, and Donald B. Rubin. 2013. *Bayesian Data Analysis*. 3rd ed. Chapman; Hall.

Ioffe, Sergey, and Christian Szegedy. 2015. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” In *Proceedings of the 32nd International Conference on Machine Learning*, edited by Francis Bach and David Blei, 37:448–56. Proceedings of Machine Learning Research. Lille, France: PMLR. http://proceedings.mlr.press/v37/ioffe15.html.

Kalaitzis, Alfredo A., and Neil D. Lawrence. 2011. “A Simple Approach to Ranking Differentially Expressed Gene Expression Time Courses Through Gaussian Process Regression.” *BMC Bioinformatics* 12 (180). doi:10.1186/1471-2105-12-180.

MacKay, David J. C. 1992. “Bayesian Methods for Adaptive Models.” PhD thesis, California Institute of Technology.

McCulloch, Warren S., and Walter Pitts. 1943. “A Logical Calculus of the Ideas Immanent in Nervous Activity.” *Bulletin of Mathematical Biophysics* 5: 115–33.

Mubangizi, Martin, Ricardo Andrade-Pacheco, Michael Thomas Smith, John Quinn, and Neil D. Lawrence. 2014. “Malaria Surveillance with Multiple Data Sources Using Gaussian Process Models.” In *1st International Conference on the Use of Mobile Ict in Africa*.

Neal, Radford M. 1994. “Bayesian Learning for Neural Networks.” PhD thesis, Dept. of Computer Science, University of Toronto.

Pearl, Judea. 1995. “From Bayesian Networks to Causal Networks.” In *Probabilistic Reasoning and Bayesian Belief Networks*, edited by A. Gammerman, 1–31. Alfred Waller.

Rasmussen, Carl Edward, and Christopher K. I. Williams. 2006. *Gaussian Processes for Machine Learning*. Cambridge, MA: mit.

Steele, S, A Bilchik, J Eberhardt, P Kalina, A Nissan, E Johnson, I Avital, and A Stojadinovic. 2012. “Using Machine-Learned Bayesian Belief Networks to Predict Perioperative Risk of Clostridium Difficile Infection Following Colon Surgery.” *Interact J Med Res* 1 (2): e6. doi:10.2196/ijmr.2131.

Tipping, Michael E., and Christopher M. Bishop. 1999. “Probabilistic Principal Component Analysis.” *Journal of the Royal Statistical Society, B* 6 (3): 611–22. doi:doi:10.1111/1467-9868.00196.

---

1. In classical statistics we often interpret these parameters, β, whereas in machine learning we are normally more interested in the result of the prediction, and less in the prediction. Although this is changing with more need for accountability. In honour of this I normally use $\boldsymbol{\beta}$ when I care about the value of these parameters, and $\mathbf{w}$ when I care more about the quality of the prediction.↩

- Work by Eric Meissner and Zhenwen Dai.
- Probabilistic programming.
- Available on Github </td>

Neil Lawrence's Talks

Neil Lawrence's Talks
N.Lawrence@sheffield.ac.uk
lawrennd
lawrennd

talks given by Neil Lawrence