Data Analysis in Astronomy and Physics

Lecture 14: Gaussian Processes

M. Röllig

Introduction

Gaussian Processes for Machine Learning by Carl Edward Rasmussen and Christopher K. I. Williams, The MIT Press, 2006. ISBN 0-262-18253-X.

A Gaussian process is a stochastic process (**a collection of random variables indexed by time or space**), such that every finite collection of those random variables has a multivariate normal distribution, i.e. every finite linear combination of them is normally distributed.

If a Gaussian process is assumed to have mean zero, defining the **covariance function** completely defines the process' behaviour.

The covariance function describes how closely related the random variables are to each other.

Pattern Recognition and Machine Learning, by Bishop, Christopher , ISBN 978-0-387-31073-2

https://github.com/krasserm/bayesian-machine-learning/blob/master/gaussian_processes.ipynb

https://www.jgoertler.com/visual-exploration-gaussian-processes/

Introduction

A common application for Gaussian processes is to **predict the value for an unseen point** from training data, i.e. **interpolation and extrapolation**.

To achieve this, we need to employ Gaussian processes to describe the functional behavior across a given domain, based on some input data.

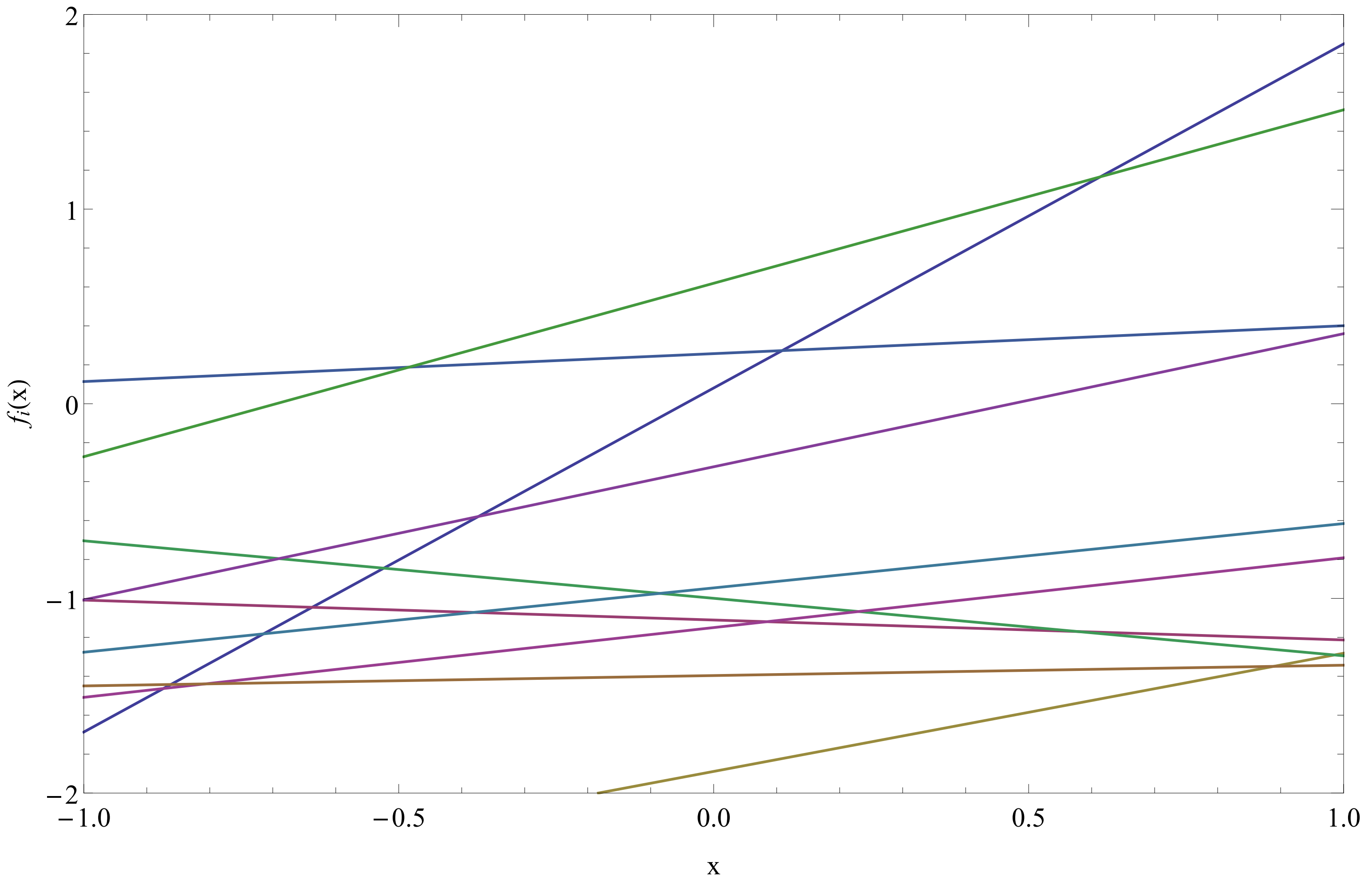
How can we use a stochastic/random process to produce a **function** AND how can we ensure that this function describes a given data?

Introduction

One possibility to randomly create a function is to choose a parametrized functional form and randomly generate the function parameters.

```
In[ ]:= randomLinear[] := Module[{μ, y0},
          {μ, y0} = RandomVariate[NormalDistribution[], 2];
          Function[{x}, y0 + μ x]]
```

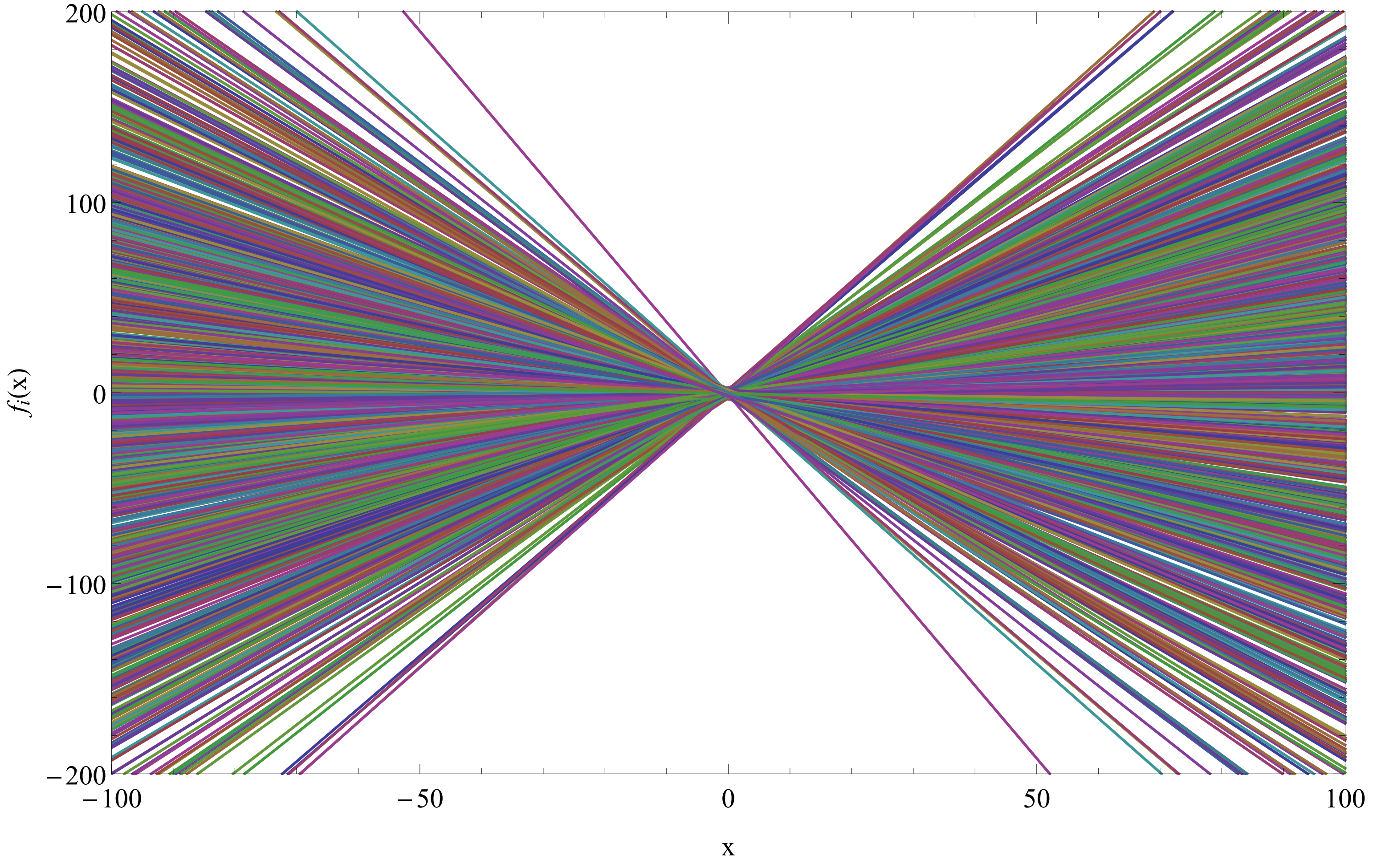
Out[•]=



Introduction

This covers an infinite number of possible linear functions. However, because of the (subjective) choice of probability distribution that we used to draw the parameters from, we do not cover all possible linear functions!

Out[ ]=

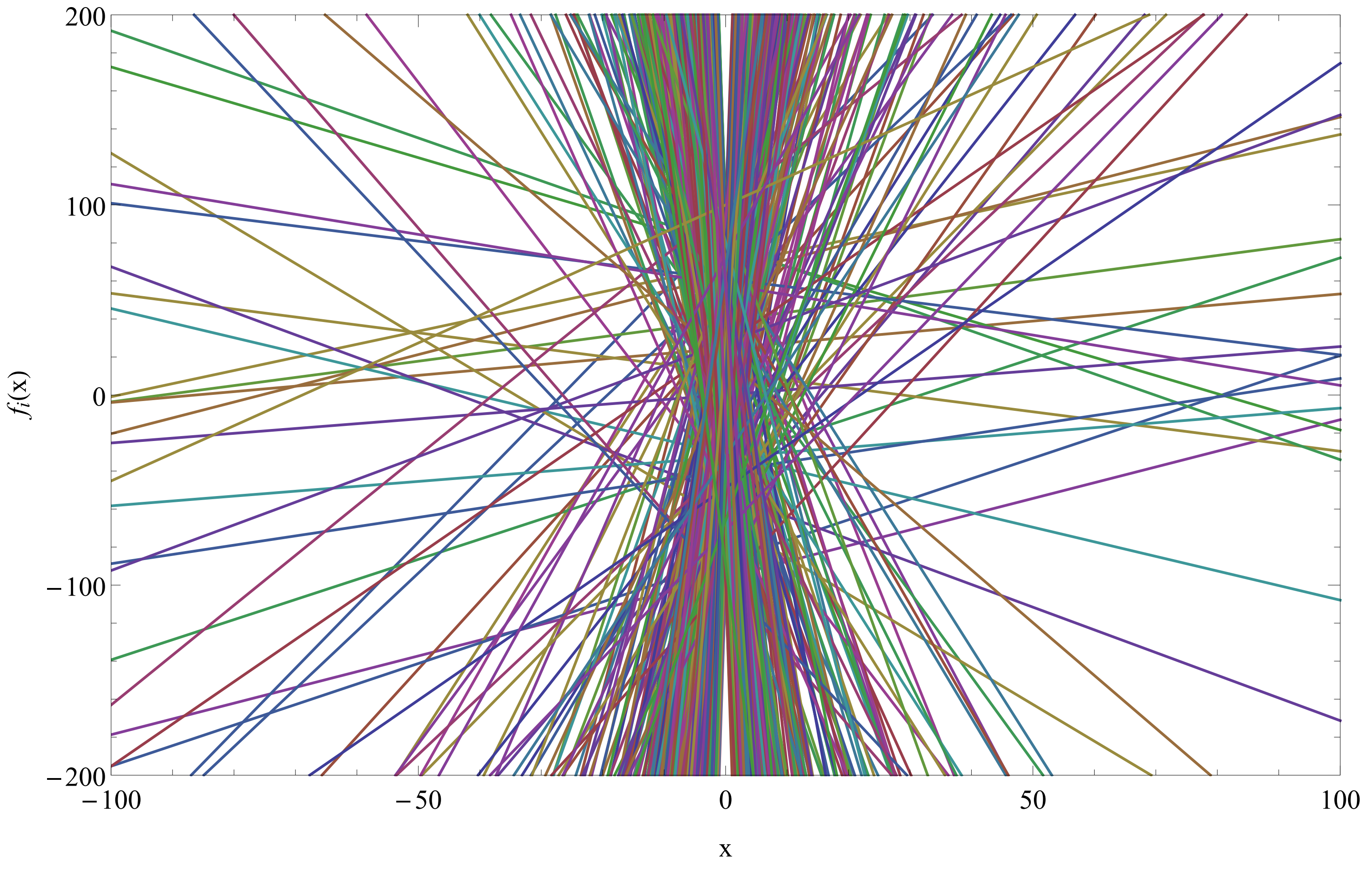


Not a uniform distribution! Maybe we take a different distribution.

Introduction

Uniformly distribution slopes and intersects.

Out[ ]=



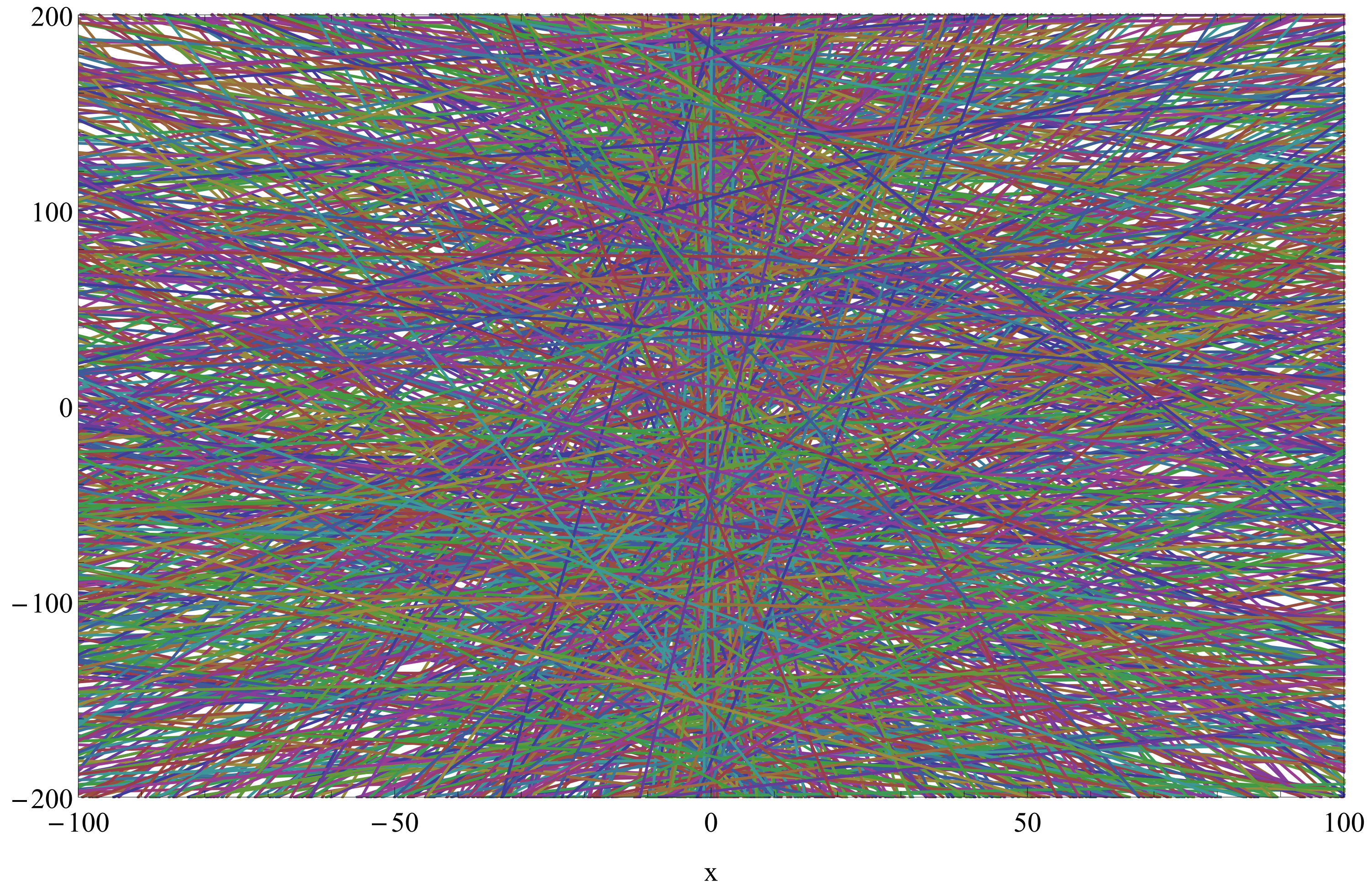
Introduction

Still not uniform in slope. We could convert a uniformly created random angle into a linear slope:

Out[ ]//TraditionalForm=

$$\boxed{\tan(\phi) = \frac{\Delta y}{\Delta x} = \text{slope}}$$

Out[ ]=



Introduction

This looks better, but is still not uniform. Our distribution choice forces the y-intersect to take place within our plot y-range.

- The choice of probability distributions from which to draw the parameters of the function is subjective and limiting.
- The choice of function parametrization excludes infinitely many other possible functions to be drawn.

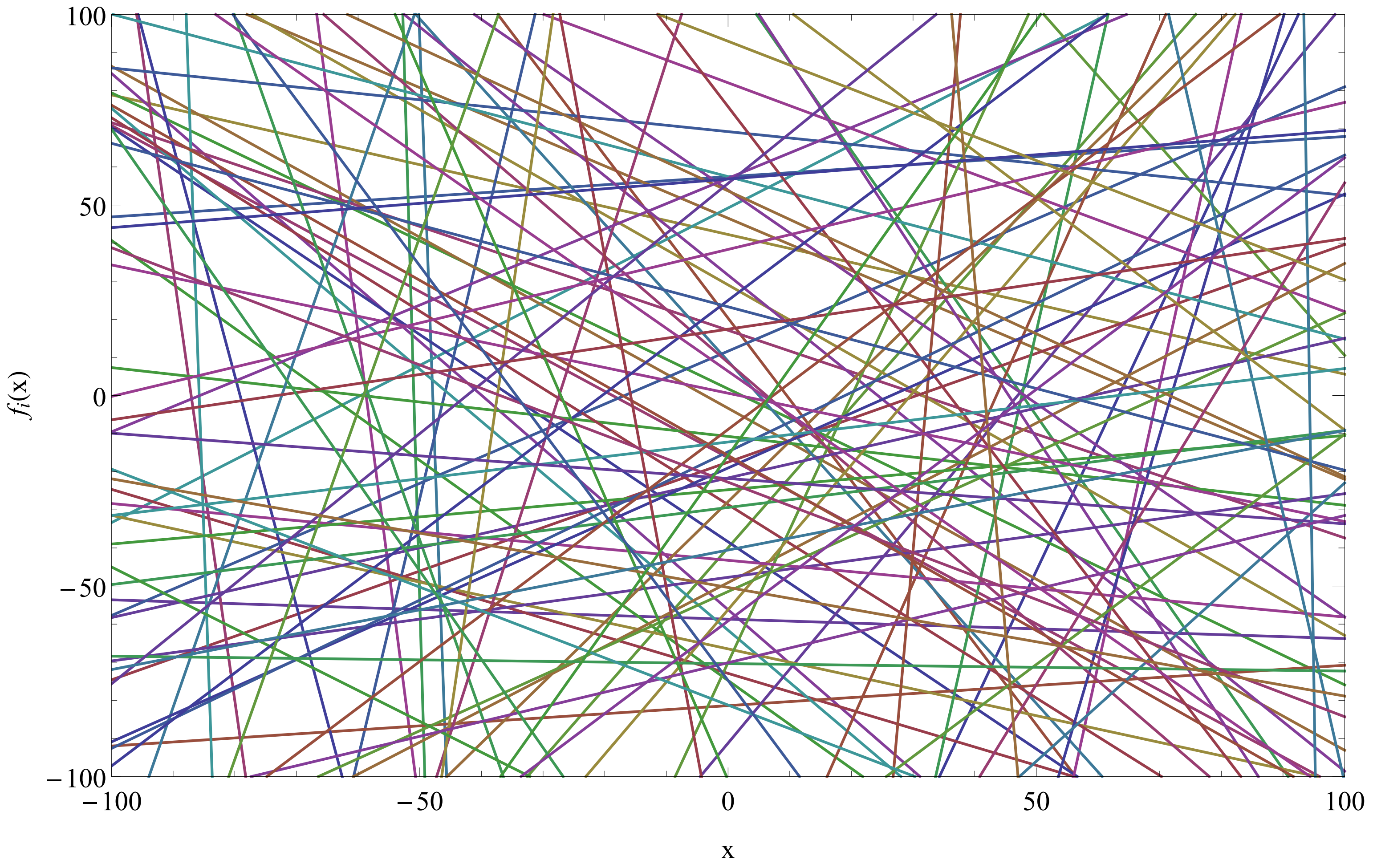
Introduction

Let’s now look at a parameter free way to create a function. Taking our previous example of a random linear function we could just randomly create two {x,y} pairs and connect the two. The 2-point form of a linear function is:

Out[ ]//TraditionalForm=

$$\boxed{y = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x - x_1) - y_1}$$

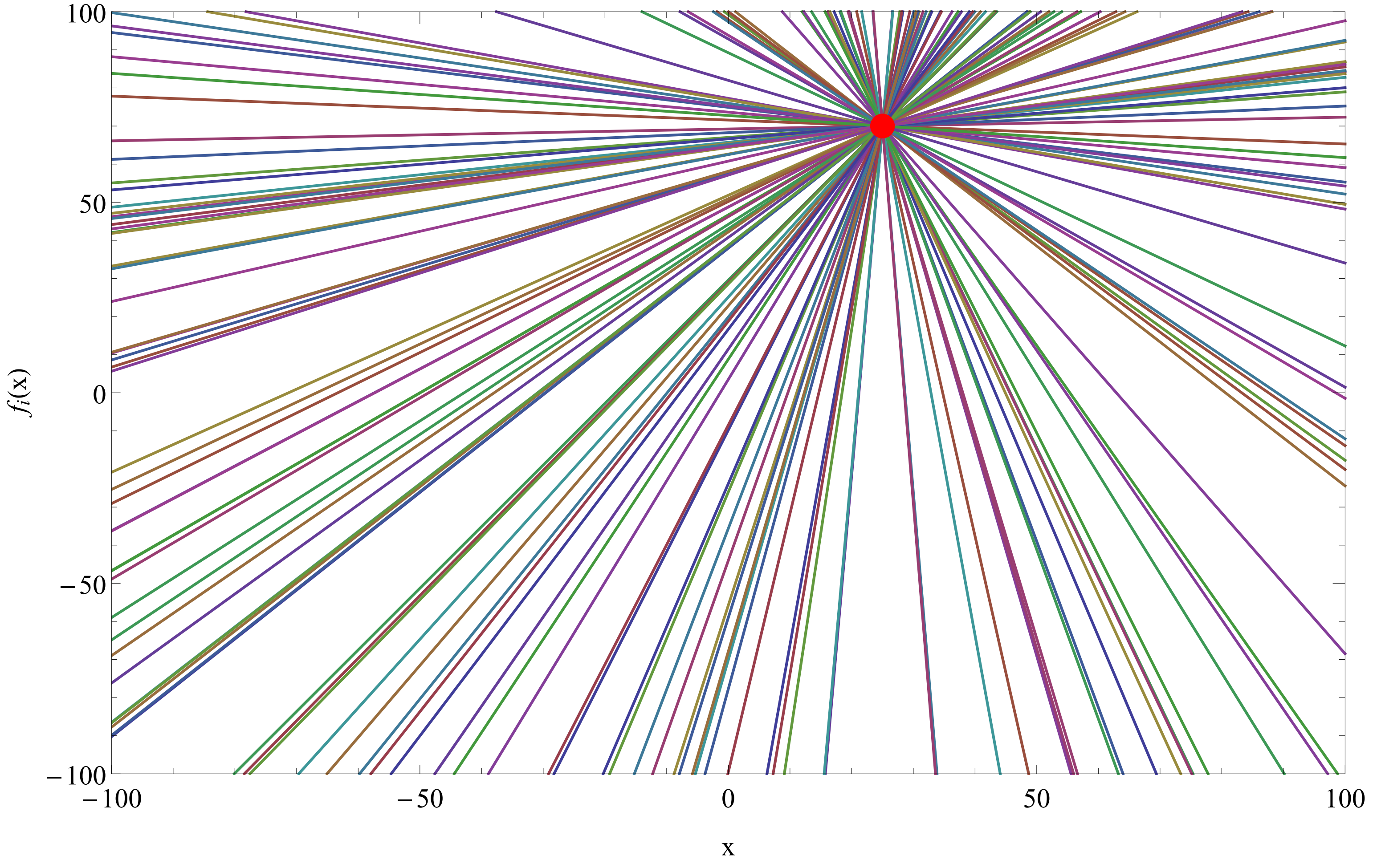
Out[•]=



Introduction

Now imagine, we have some data and our random functions should be related to that data. In regular linear interpolation, the function is forced to pass through the data. Given one data point, we practically fix one of the two random points:

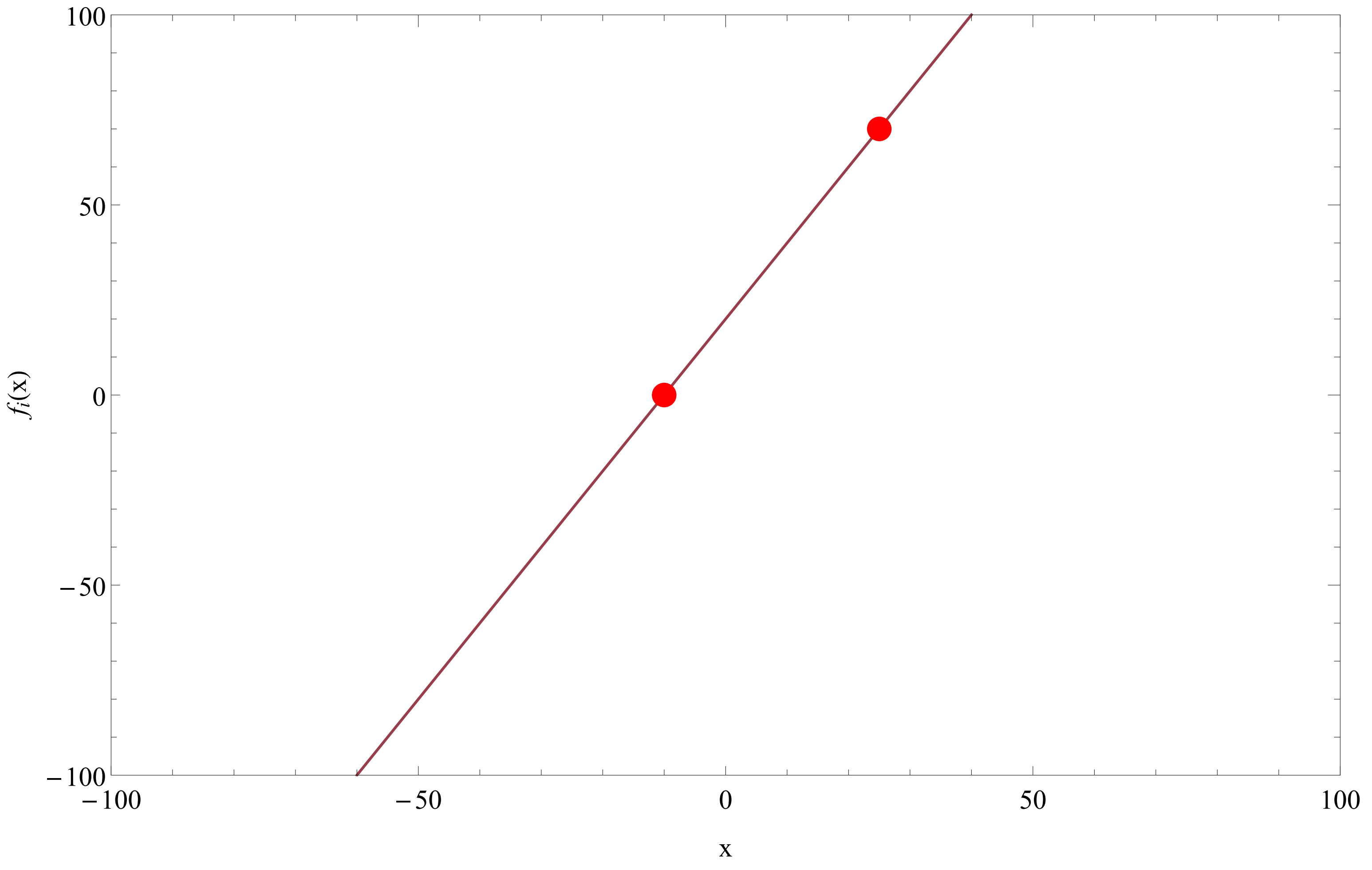
Out[ ]=



Introduction

If we have two data points, just one of the many possibilities remains:

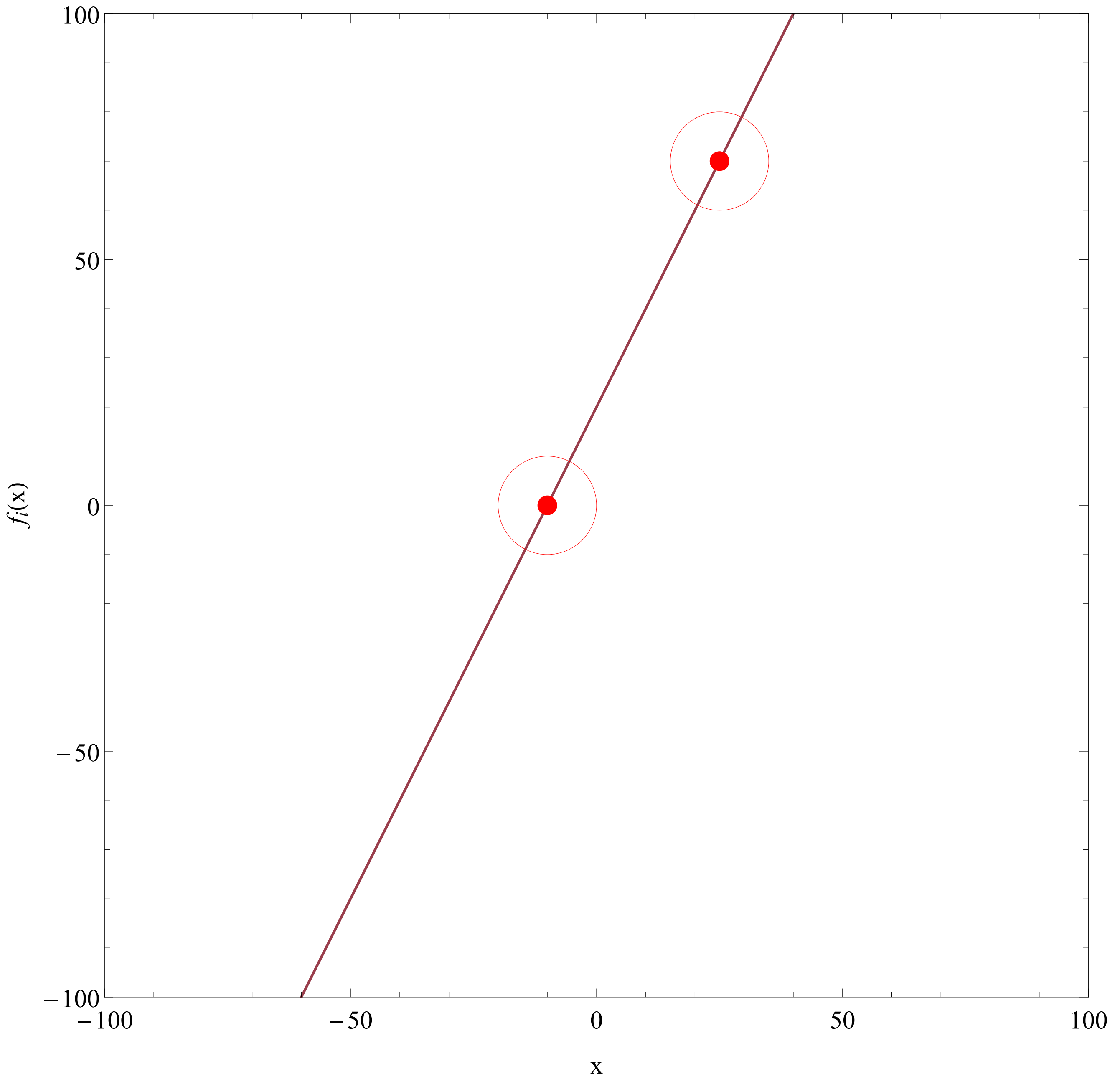
Out[ ]=



Introduction

But what if our data has some intrinsic uncertainty? Let’s assume, that our 1σ margin of error is 10 and that our uncertainty is normally distributed

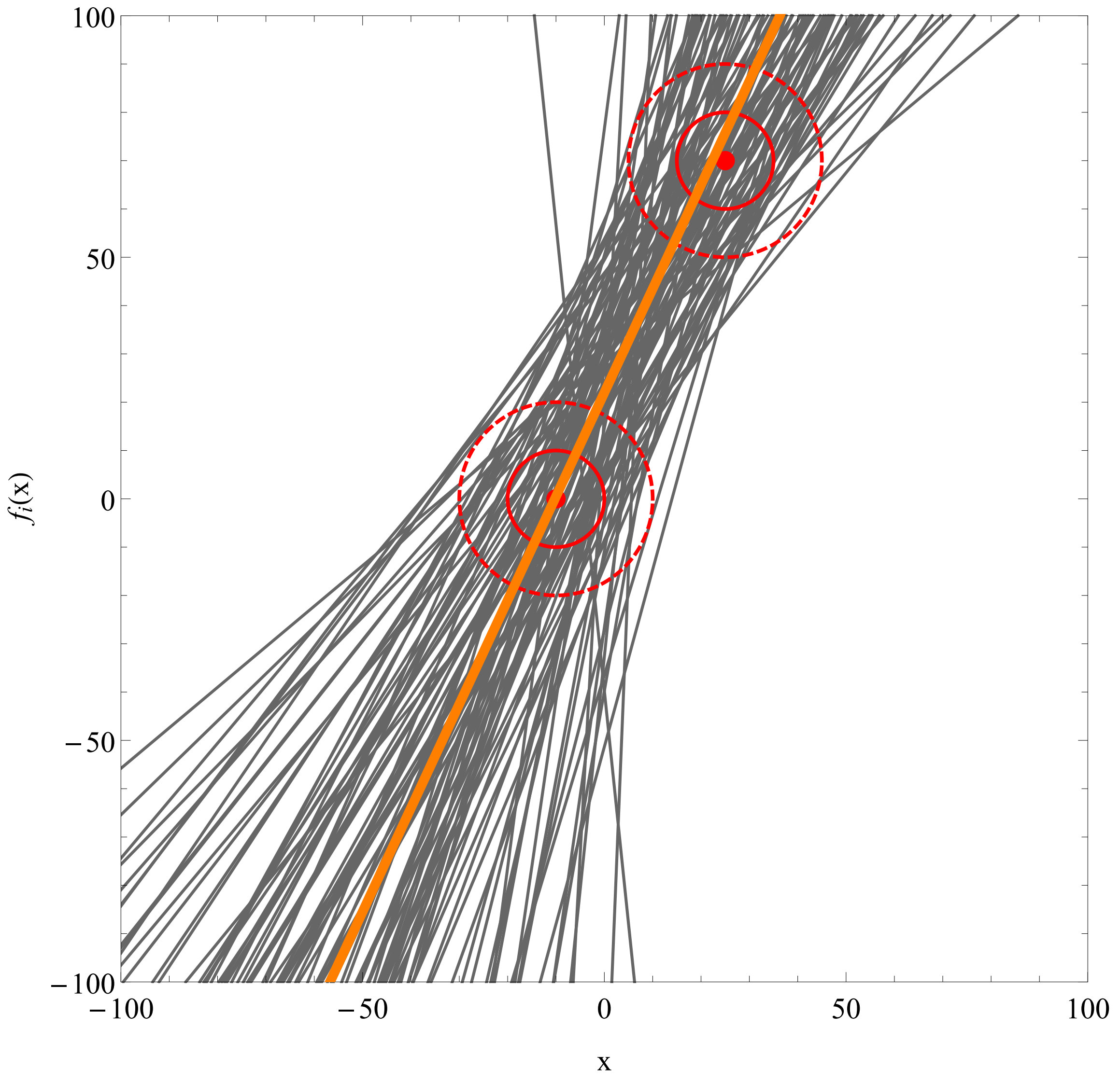
Out[•]=



Introduction

A 1σ CI means, that we are 68% certain that the real data value is within that circle. A random linear function describing the data consequently has a 68% chance to pass through the circle. To pass through both circles, the probability is 68% × 68% = 0.46, about half of the random lines in the plot below pass through the inner circle. The dashed circles correspond to 2σ CI’s, i.e. 95% × 95% = 0.9025. About 10 lines should not

Out[•]=



Bayesian Linear Regression

The previous considerations are closely related to the concept of Bayesian Linear Regression. Recall, that linear regression boils down to solving the normal equation

Out[∘]//TraditionalForm=

$$\boxed{\left(X^{\top} X\right) \widehat{\beta} = X^{\top} t}$$

where X is called design matrix, t is the output vector (target variable), i.e. the list of function values of our data, and $\hat{\beta}$ is the vector of model parameters that minimizes the squared sum of residuals $\|\, t - X\,\beta \,\|^2$.

Out[∘]//TraditionalForm=

$$\boxed{S(\beta) = \sum_{i=1}^{n} \left| t_i - \sum_{j=0}^{p} X_{\mathrm{ij}}\, \beta_j \right|^2 = \|\, t - X\,\beta \,\|^2}$$

Bayesian Linear Regression

Now let’s consider the least squares approach, and its relation to maximum likelihood, in more detail:
Again, we try to find a target variable t that is given by a deterministic function $y(x, \beta)$ with additive Gaussian noise (with zero mean, and variance γ^{-1}), so that:

Out[∘]//TraditionalForm=

$$\boxed{t = y(x, \beta) + \epsilon}$$

As t randomly scatters around the “true” value $y(x, \beta)$ we can write:

In[∘]:=
```
equation[𝓅["t|x", β, γ] == 𝒩["t|y[x,β]", γ^-1]]
```

Out[∘]//TraditionalForm=

$$\boxed{p(\mathrm{t|x}, \beta, \gamma) = \mathcal{N}\left(\mathrm{t|y[x,}\beta\mathrm{]}, \frac{1}{\gamma}\right)}$$

Bayesian Linear Regression

Alternative resource: https://github.com/krasserm/bayesian-machine-learning/blob/master/bayesian_linear_regression.ipynb

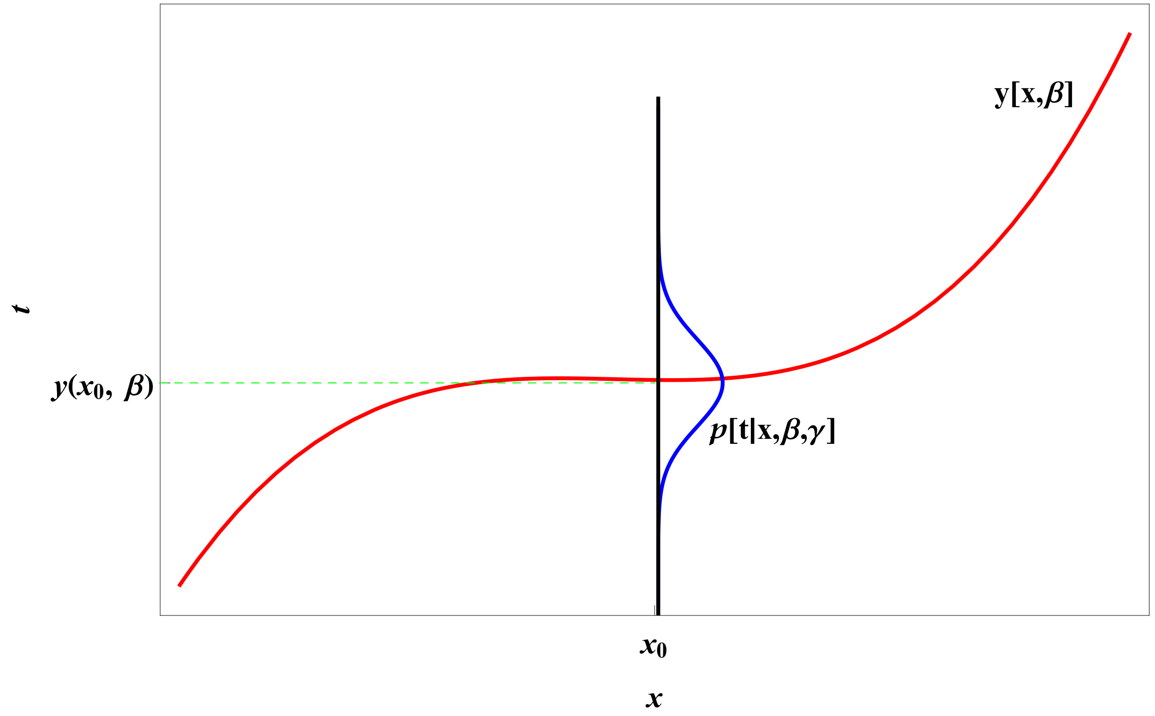
In the case of a Gaussian conditional distribution of the above form the conditional mean will be simply

Out[∘]//TraditionalForm=

$$\boxed{\mathbb{E}(\mathrm{t}|\mathrm{x}) = \int t\, p(\mathrm{t}|\mathrm{x})\, dt = y(x, \beta)}$$

The red line shows the mean given by y(x,β), the blue line shows the Gaussian conditional distribution for t given x and the precision is given by the parameter γ, which is related to the variance by $\gamma^{-1} = \sigma^2$.

Out[∘]=



Bayesian Linear Regression

The term linear in Linear Regression means that the model f is a linear function of the parameters β as well as a linear function of the input variables x_i. Generally, setting $x = (x_1, \ldots x_D)^T$ and $w = (\beta_1, \ldots, \beta_D)$, we can write

Out[∘]//TraditionalForm=

$$y(x, \beta) = \beta_0 + \sum_{j=1}^{M-1} \beta_j \, \phi_j(x)$$

where $\phi_j(x)$ are the basis functions used in the regression. The total number of model parameters is M. β_0 is sometimes called bias parameter. It is convenient to define $\phi_0(x) = 1$ so that

Out[∘]//TraditionalForm=

$$y(x, \beta) = \sum_{j=0}^{M-1} \beta_j \, \phi_j(x) = x^{\mathsf{T}} \, \phi(x)$$

Bayesian Linear Regression

So we can write the **likelihood function** which is a function of the adjustable parameters β and γ

Out[∘]//TraditionalForm=

$$\boxed{p(\mathrm{t}|\mathrm{X}, \beta, \gamma) = \prod_{n=1}^{N} \mathcal{N}\left(t_n | \beta^T \phi(x_n), \frac{1}{\gamma}\right)}$$

In the following we will drop the dependence from the input x. Taking the logarithm of the likelihood we have

Out[∘]//TraditionalForm=

$$\log(p(\mathrm{t}|\beta, \gamma)) = \sum_{n=1}^{N} \log\left(\mathcal{N}\left(t_n | \beta^T \phi(x_n), \frac{1}{\gamma}\right)\right) = \frac{1}{2} \mathrm{N} \log(\gamma) - \frac{1}{2} \mathrm{N} \log(2\pi) - \gamma\, \mathrm{E}_D(\beta)$$

with the sum-of-squares error function

Out[∘]//TraditionalForm=

$$\mathrm{E}_D(\beta) = \frac{1}{2} \sum_{n=1}^{N} \left(t_n - \beta^{\mathsf{T}} \phi(x_n)\right)^2$$

Bayesian Linear Regression

Finding the model that maximizes the likelihood of the data we got is then equivalent to minimizing the sum-of-squares function $E_D(\beta)$. Setting the gradient of the log-likelihood to zero gives

Out[∘]//TraditionalForm=

$$0 = \sum_{n=1}^{N} t_n\, \phi(x_n)^{\mathsf{T}} - \beta^{\mathsf{T}} \left(\sum_{n=1}^{N} \phi\,(x_n)\, \phi\,(x_n)^T \right)$$

with the solution

Out[∘]//TraditionalForm=

$$\boxed{\beta_{\mathrm{ML}} = (\Phi^T \Phi)^{-1}\, \Phi^{\mathsf{T}}\, t}$$

Bayesian Linear Regression

This is again the normal equation with the design matrix

Out[∘]//TraditionalForm=

$$\Phi = \begin{pmatrix} \phi_0(x_1) & \phi_1(x_1) & \cdots & \phi_{M-1}(x_1) \\ \phi_0(x_2) & \phi_1(x_2) & \cdots & \phi_{M-1}(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_N) & \phi_1(x_N) & \cdots & \phi_{M-1}(x_N) \end{pmatrix}$$

with the Moore-Penrose pseudo-inverse of the matrix Φ:

Out[∘]//TraditionalForm=

$$\boxed{\Phi^{\dagger} = (\Phi^T \Phi)^{-1} \, \Phi^{\mathsf{T}}}$$

One technique that is often used to control the over-fitting phenomenon in such cases is that of **regularization**, which involves **adding a penalty term to the error function** in order to discourage the coefficients from reaching large values.

Bayesian Linear Regression

Let’s introduce a prior probability distribution over the model parameters β with mean m_0 and covariance S_0.

Out[ ]//TraditionalForm=

$$p(\beta) = \mathcal{N}(\beta \mid m_0, S_0)$$

An important property of the multivariate Gaussian distribution is that if two sets of variables are jointly Gaussian, then the **conditional distribution of one set conditioned on the other is again Gaussian**. Similarly, the marginal distribution of either set is also Gaussian.

conjugate Gaussian prior distribution ⇒the posterior will also be Gaussian

Bayesian Linear Regression

It can be shown that the corresponding **posterior** probability distribution is of the form

Out[ ]//TraditionalForm=

$$p(\beta \mid t) = \mathcal{N}(\beta \mid m_N, S_N)$$

where

Out[ ]=

$$\boxed{\begin{aligned} m_N &= S_N \left(S_0^{-1}\, m_0 + \gamma\, \Phi^T\, t\right) \\ S_N^{-1} &= S_0^{-1} + \gamma\, \Phi^T\, \Phi \end{aligned}}$$

Here, S_0 and m_0 are the mean and the covariance of the conjugate prior (previous slide).

Bayesian Linear Regression

To simplify the consideration, consider a zero-mean isotropic Gaussian governed by a single precision parameter α so that

Out[∘]//TraditionalForm=

$$p(\beta \mid \alpha) = \mathcal{N}(\beta \mid 0, \alpha^{-1}\,\mathrm{I})$$

and the corresponding posterior distribution over β is then given by

Out[∘]=

$$m_N = \gamma\, S_N\, \Phi^{\mathsf{T}}\, t$$
$$S_N^{-1} = \alpha^{-1}\,\mathrm{I} + \gamma\, \Phi^{\mathsf{T}}\, \Phi$$

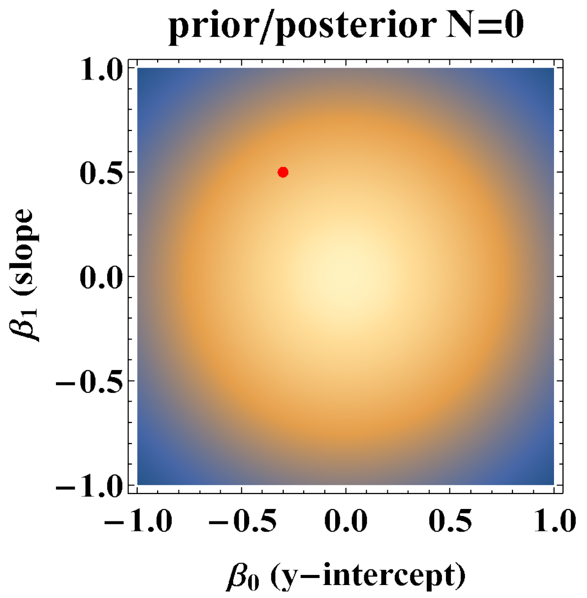
The log of the posterior distribution is given by the sum of the log likelihood and the log of the prior and, as a function of β, takes the form

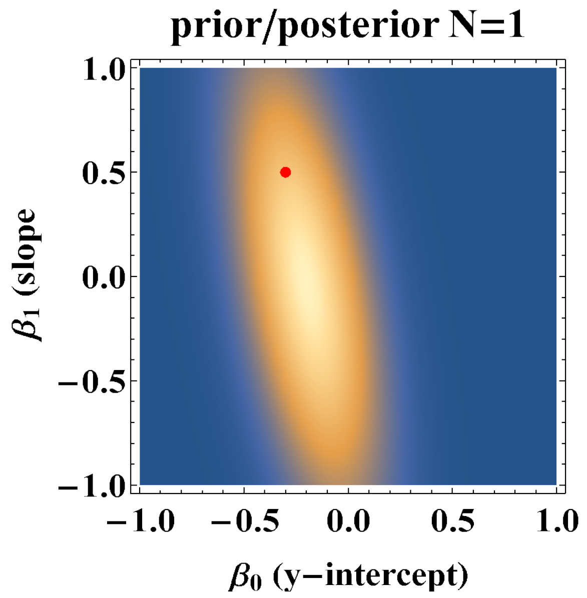
Out[∘]//TraditionalForm=

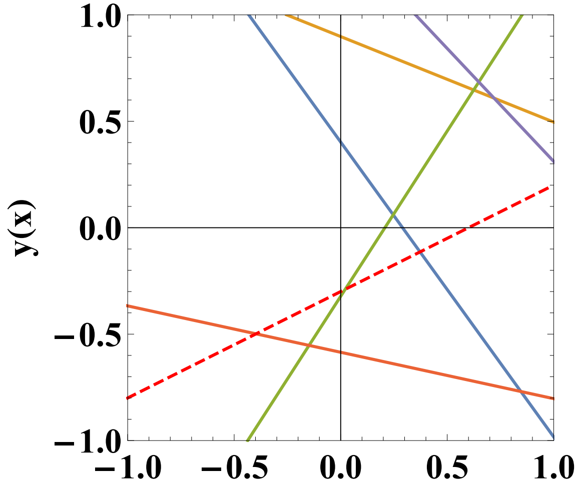
$$\log(p(\beta|\mathrm{t})) = \frac{1}{2}\gamma \sum_{n=1}^{N}\left(t_n - \beta^{\mathsf{T}}\,\phi(x_n)\right)^2 - \frac{\alpha}{2}\beta^{\mathsf{T}}\,\beta + \mathrm{const}$$

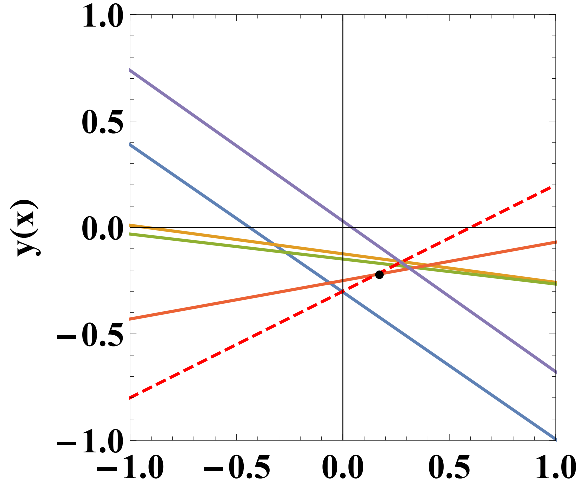
Therefore, to maximize the posterior probability, we minimize the sum-of-squares error function with the addition of a quadratic **regularization** term with λ=α/γ.

Bayesian Linear Regression

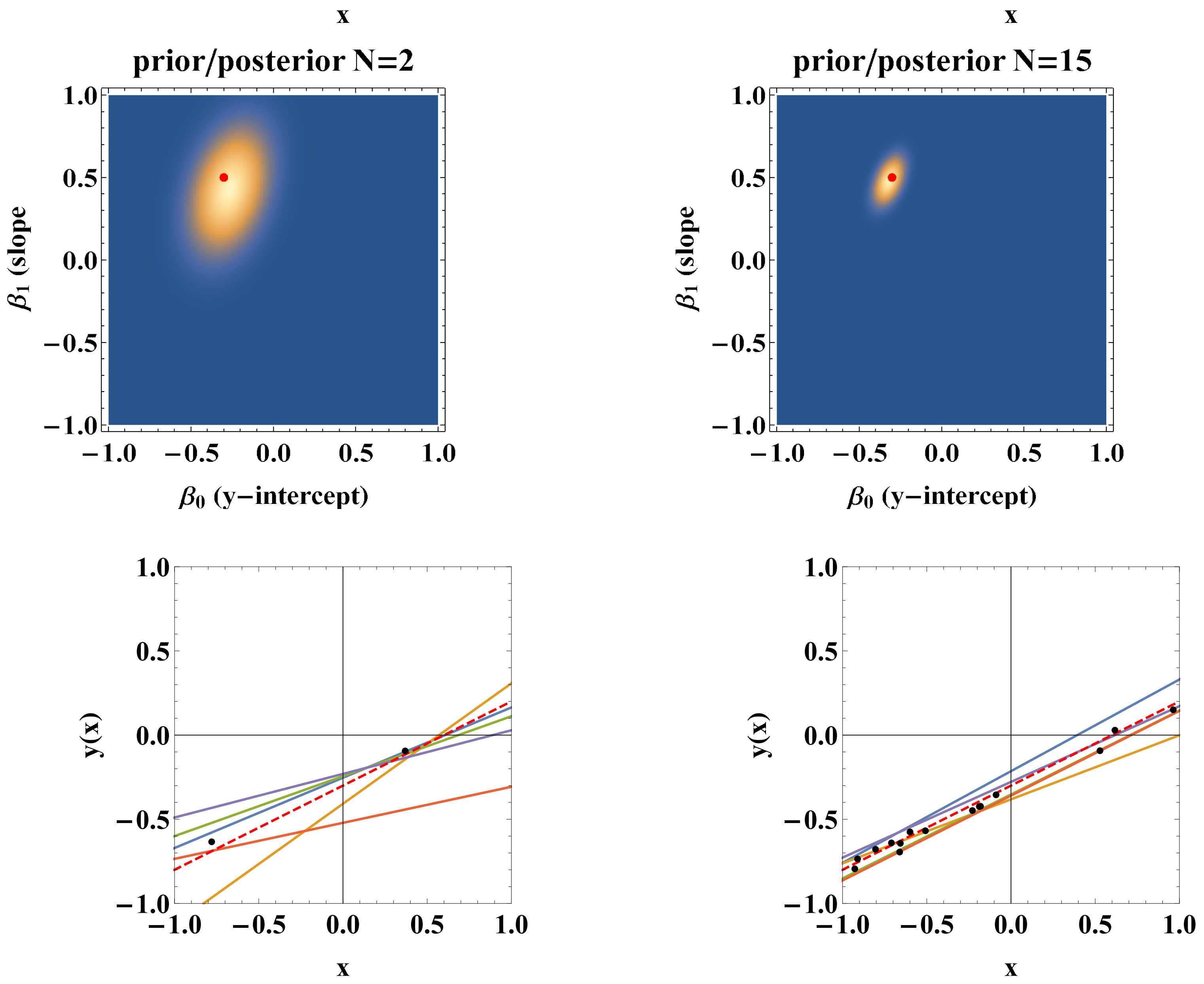








Out[∘]=


x
prior/posterior N=2
β_1 (slope
β_0 (y−intercept)
y(x)
x
x
prior/posterior N=15
β_1 (slope
β_0 (y−intercept)
y(x)
x

The left column shows the posterior distribution $p(\beta \mid t) = \mathcal{N}(\beta \mid m_N, S_N)$ in the $\{\beta_0, \beta_1\}$ projection for different number of (random) data points N. The red dot shows the real value. The right column shows 5 random draws (realizations) from the posterior distribution together with the real function (red, dashed) and the data points.

Bayesian Linear Regression

Often we are interested in predictions on *t* for new data values of *x*. This requires that we evaluate the **predictive distribution** defined by

Out[∘]//TraditionalForm=

$$p(t\,|\mathbf{t},\,\alpha,\,\gamma) = \int p(t\,|\,\beta,\,\gamma)\,p(\beta\,|\mathrm{t},\,\alpha,\,\gamma)\,d\beta$$

Here **t** is the vector of target values from the **training data set**. The conditional distribution p(t|x,β, γ) of the target variable is given by

Out[∘]//TraditionalForm=

$$p(t\,|\,\beta,\,\gamma) = \mathcal{N}\left(t\,|\,y(x,\,\beta),\,\frac{1}{\gamma}\right)$$

and the posterior weight distribution is given by

Out[∘]//TraditionalForm=

$$p(\beta\,|\mathbf{t},\,\alpha,\,\gamma) = \mathcal{N}(\beta\,|\,m_N,\,S_N)$$

Bayesian Linear Regression

From the general result of convolving two Gaussian distributions we find

Out[ ]//TraditionalForm=

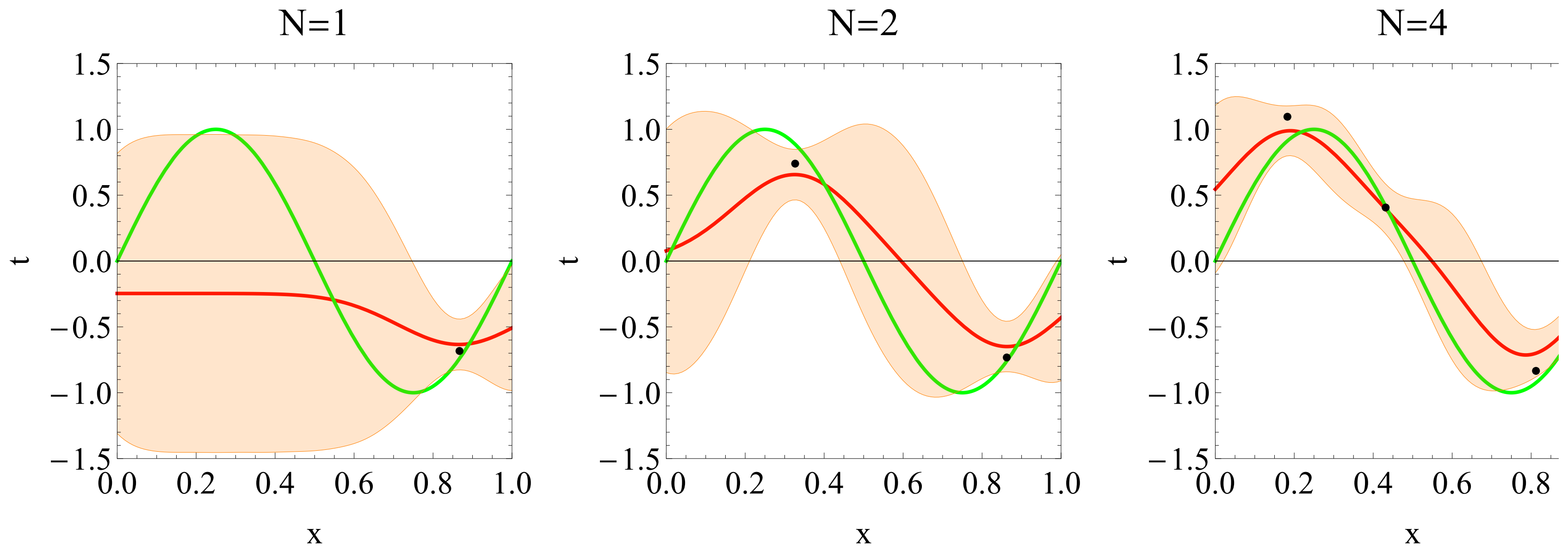
$$p(t\,|\mathrm{x}, \mathbf{t}, \alpha, \gamma) = \mathcal{N}\left(t\,|\,{m_N}^{\mathsf{T}}\,\phi(x),\, \sigma_N^2(x)\right)$$

where the variance $\sigma_N{}^2[x]$ of the predictive distribution is given by

Out[ ]//TraditionalForm=

$$\sigma_N^2(x) = \frac{1}{\gamma} + \phi(x)^{\mathsf{T}}.S_N.\phi(x)$$

Out[ ]=

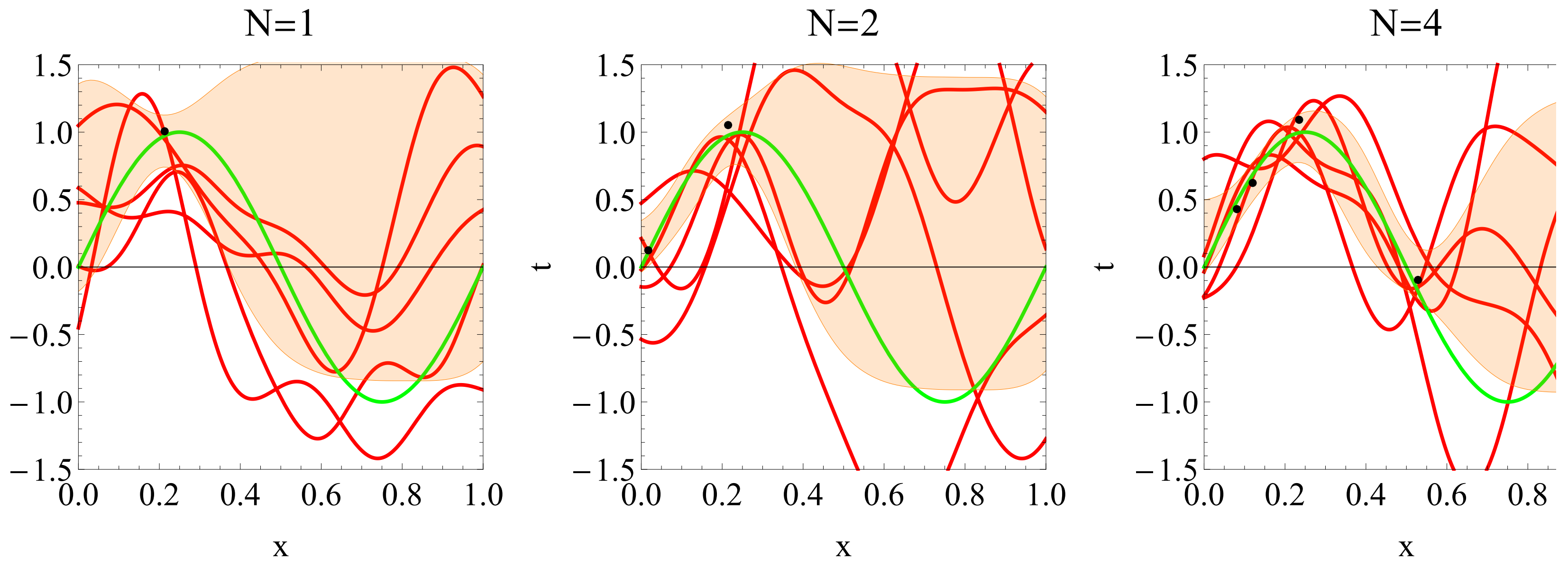


For each plot, the red curve shows the mean of the corresponding Gaussian predictive distribution, and the red shaded region spans one standard deviation either side of the mean. Note that the predictive uncertainty depends on x and is smallest in the neighbourhood of the data points.

Bayesian Linear Regression

In order to gain insight into the covariance between the predictions at different values of x, we can draw samples from the posterior distribution over β, and then plot the corresponding functions y(x,β), as shown below

Out[∘]=

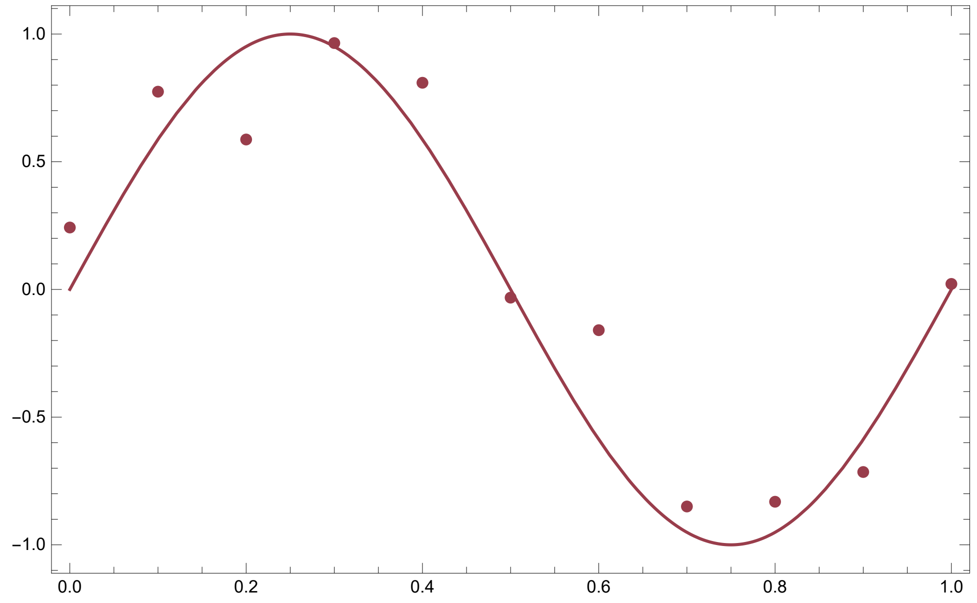


Every single realization is a drawn from the posterior probability distribution

Overfitting

To understand overfitting let’s look at some example data. We take a Sine function with some random noise.

Out[]=



A polynomial of degree 9 is the most complex function that we can uniquely fit to the data. Our expectation is that a higher complexity is equivalent to a better representation of the data.

Overfitting

We set up and solve the normal equation :

```
In[◦]:= X = Table[{1, x[[i]], x[[i]]^2, x[[i]]^3, x[[i]]^4, x[[i]]^5, x[[i]]^6, x[[i]]^7, x[[i]]^8, x[[i]]^9, x[[i]]^10}, {i, 1, Length[x]}];
        β̂ = {β0, β1, β2, β3, β4, β5, β6, β7, β8, β9, β10};
        Solve[Thread[Flatten[(Transpose[X].X).β̂, 1] == Transpose[X].y], {β0, β1, β2, β3, β4, β5, β6, β7, β8, β9, β10}] // Quiet

Out[◦]= {{β0 → 0.242468, β1 → -173.317, β2 → 5181.25, β3 → -59 750.5, β4 → 363 036., β5 → -1.30349 × 10^6,
          β6 → 2.9061 × 10^6, β7 → -4.06587 × 10^6, β8 → 3.4715 × 10^6, β9 → -1.65247 × 10^6, β10 → 335 945.}}
```

Or with built in Mathematica tools:

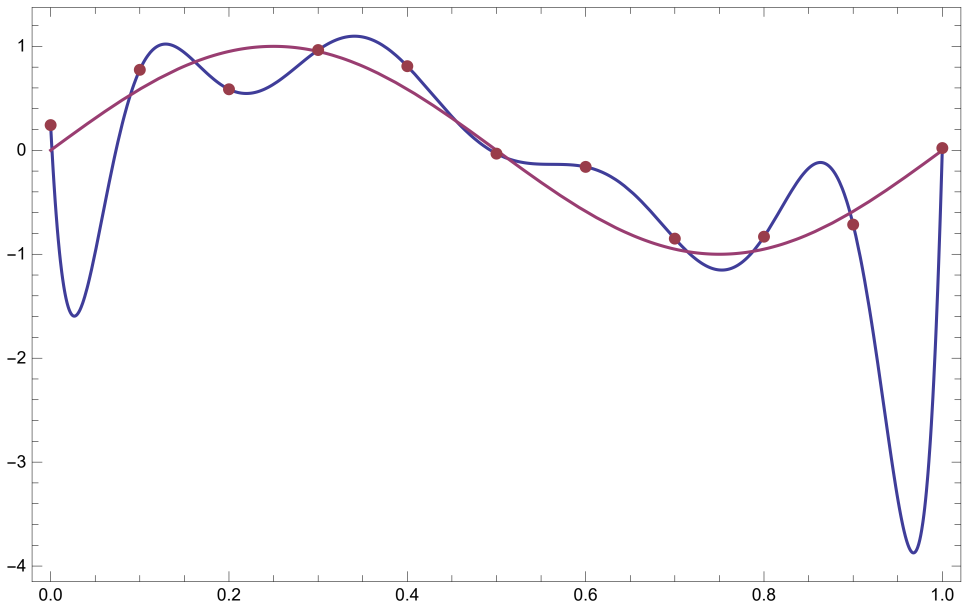
```
In[◦]:= fit = LinearModelFit[Transpose[{x, y}], {z, z^2, z^3, z^4, z^5, z^6, z^7, z^8, z^9, z^10}, z];
        fit["BestFitParameters"]

Out[◦]= {0.242452, -167.966, 5037.06, -58 203.7, 354 175., -1.27316 × 10^6, 2.84101 × 10^6, -3.9776 × 10^6, 3.39802 × 10^6, -1.61822 × 10^6, 329 102.}
```

Overfitting

The result is actually pretty bad. What is going on here?

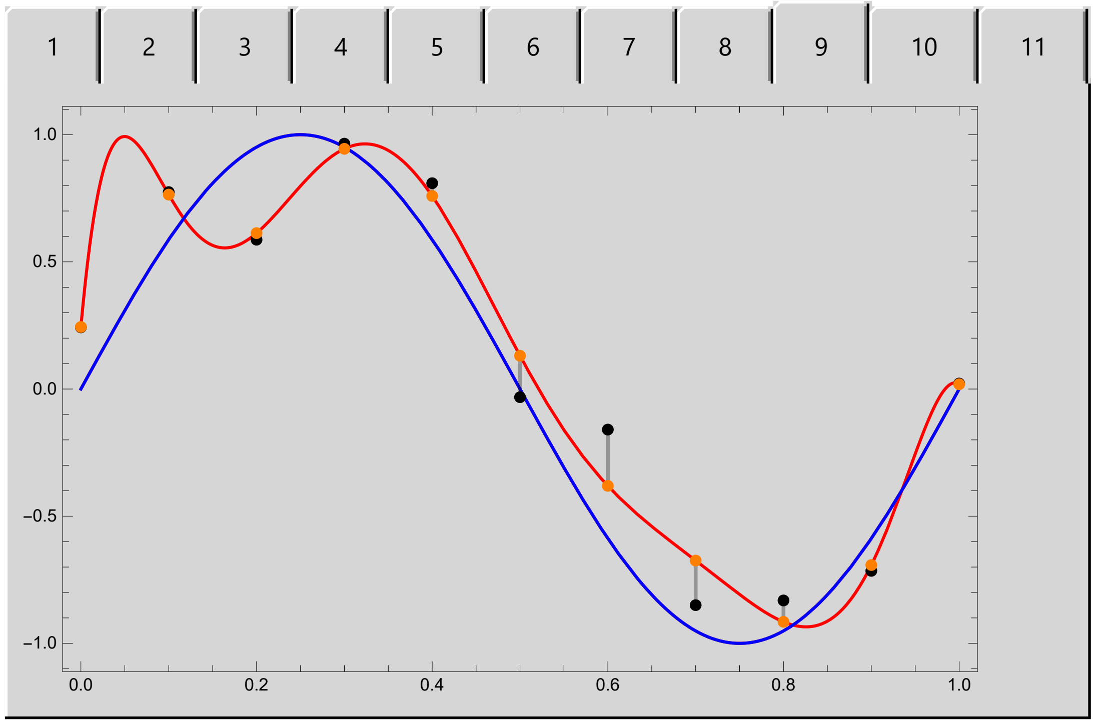
Out[ ]=



Overfitting

Our goal was to find a polynomial of degree 9 that minimizes the squared residuals!
This is not equivalent to best represent the (underlying) functional form! The residuals to our respective fit function consists of **random and of systematic errors** and we are trying to minimize both, i.e. the higher the degree of the polynomial, the closer the fit passes through all data points. W.R.T. the optimal solution the fit shows a large variance!

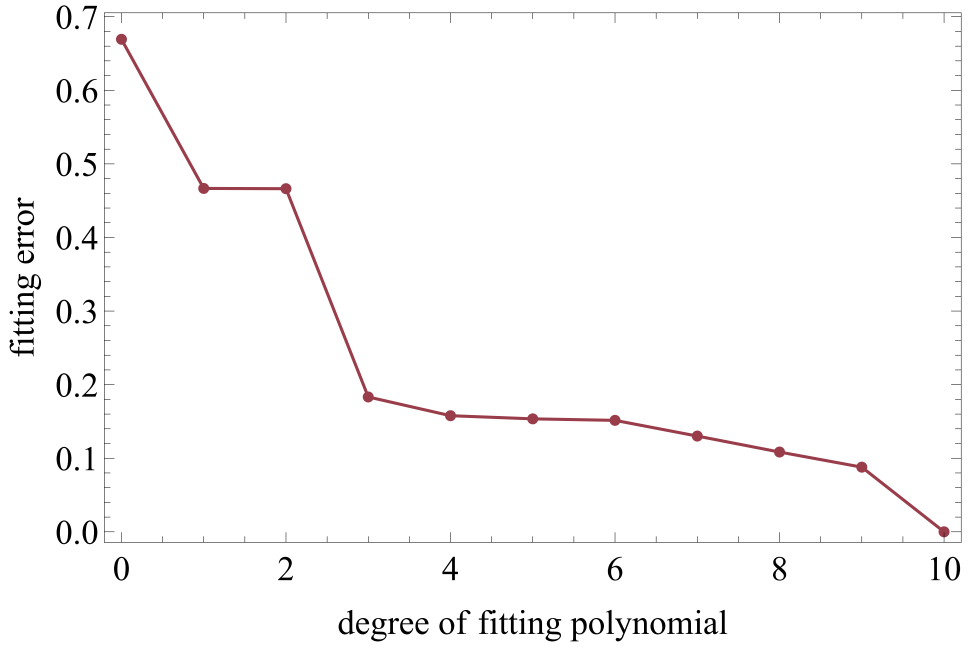
Out[*]=



Overfitting

We can also plot the total error of our fit, which is just $\sqrt{\frac{\text{res}^2}{N}}$.

Out[ ]=

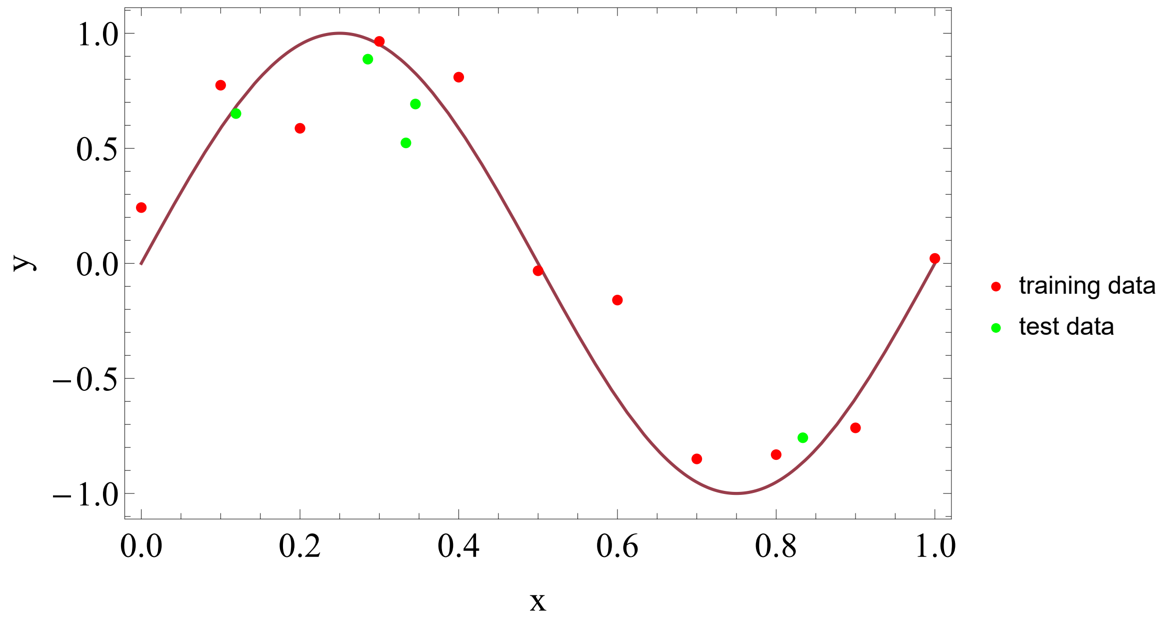


The error reaches 0 for a degree of 10, as expected for 10 data points.

Overfitting

Now imagine we test our model (i.e. fit) on a set of test data. (We randomly create a set)

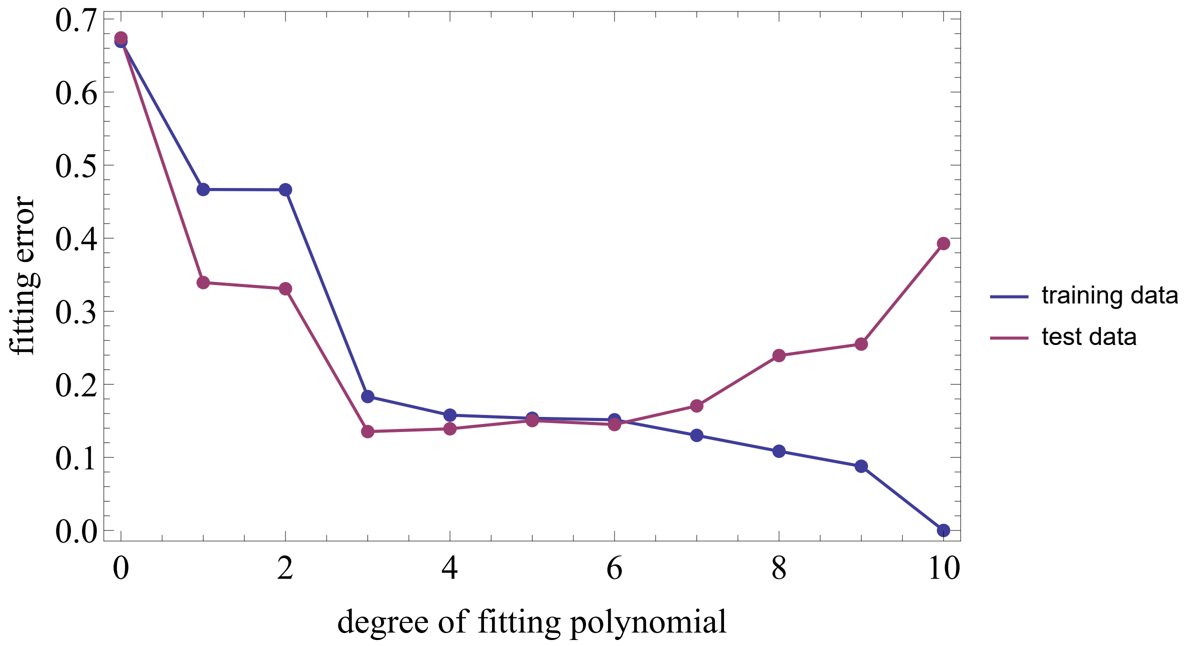
Out[ ]=



Overfitting

For every polynomial degree of our fitting function we can also compute the error with respective to the test data and compare it with the error of the training data

Out[•]=



As we can see, the error of the fit w.r.t. to the test data also decreases, but after reaching a minimum it starts to increase again as the fitted polynomial becomes strongly oscillating between the training data points. For degrees higher than 5...7 the general quality of the fit becomes worse. This effect is called overfitting.

Overfitting

When we take a look at the magnitude of the polynomial coefficients, i.e. the fitting parameters, we note, that the higher the polynomial degree the larger the absolute magnitude of the coefficients becomes.

Out[ ]=

	1	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}
0	0.0737648										
1	0.797124	−1.44672									
2	0.768222	−1.25404	−0.192683								
3	0.147181	8.61361	−26.0694	17.2511							
4	0.251755	4.98257	−7.91416	−11.7972	14.5242						
5	0.279489	2.80854	9.9973	−62.2574	72.3031	−23.1116					
6	0.268324	5.06576	−17.6917	56.8376	−157.823	181.583	−68.2316				
7	0.250744	16.7097	−211.746	1215.75	−3447.89	4989.11	−3559.73	997.572			
8	0.243843	38.0417	−655.183	4625.2	−16 461.2	32 266.8	−35 497.9	20 579.6	−4895.5		
9	0.241806	86.1473	−1820.26	15 466.7	−68 521.9	176 142.	−273 172.	252 233.	−127 706.	27 291.2	
10	0.242452	−167.966	5037.06	−58 203.7	354 175.	-1.27316×10^6	2.84101×10^6	-3.9776×10^6	3.39802×10^6	-1.61822×10^6	329 102.

Overfitting

Writing down just the min and max value of all cases:

Out[▫]//TableForm=

	min	max
1	0.0737648	0.0737648
2	-1.44672	0.797124
3	-1.25404	0.768222
4	-26.0694	17.2511
5	-11.7972	14.5242
6	-62.2574	72.3031
7	-157.823	181.583
8	-3559.73	4989.11
9	-35 497.9	32 266.8
10	-273 172.	252 233.
11	-3.9776×10^6	3.39802×10^6

In order to fit the noise+signal the polynomial has to find very large/small coefficients. We will make use of this fact to reduce the overfitting. The approach we will choose is called **regularization**.

Regularization

So far we were minimizing the error function, which was just the sum of all squared residuals, with respect to the fitting parameters $\beta = \{\beta_0, \ldots, \beta_M\}$:

Out[ ]//TraditionalForm=

$$\mathrm{E}(\beta) = \sum_{i=1}^{n} (y(x_i, \beta) - t_i)^2$$

As we saw, higher polynomial degrees lead to worse fitting quality. Hence, we introduce an additional error term that penalizes fitting functions of higher degree. We make use of the fact that higher polynomials tend to have absolutely larger coefficients:

```
equationNoBox["Ẽ"[β] == Sum[(y[x_i, β] - t_i)^2, {i, 1, n}] + λ Norm[β]^2]
```

Out[ ]//TraditionalForm=

$$\tilde{E}(\beta) = \sum_{i=1}^{n} (y(x_i, \beta) - t_i)^2 + \lambda \, \|w\|^2$$

were $\| \beta \|^2 = \beta^T . \beta = \beta_0{}^2 + \ldots + \beta_M{}^2$ and the term λ controls the relative importance of the regularization term compared with the sum-of-squares error term.

Regularization

We start with the fit without regularization, i.e. $\lambda = 0$, equivalent to Log[λ] = $-\infty$:

```
In[ ]:=
model = β0 + xx β1 + xx^2 β2 + xx^3 β3 + xx^4 β4 + xx^5 β5 + xx^6 β6 + xx^7 β7 + xx^8 β8 + xx^9 β9;
f1 = FindFit[Transpose[{x, y}], model, {β0, β1, β2, β3, β4, β5, β6, β7, β8, β9}, xx]
```

```
Out[ ]= {β0 → 0.241806, β1 → 86.1473, β2 → -1820.26, β3 → 15 466.7,
  β4 → -68 521.9, β5 → 176 142., β6 → -273 172., β7 → 252 233., β8 → -127 706., β9 → 27 291.2}
```

Regularization

And now two attempts with different values of λ:

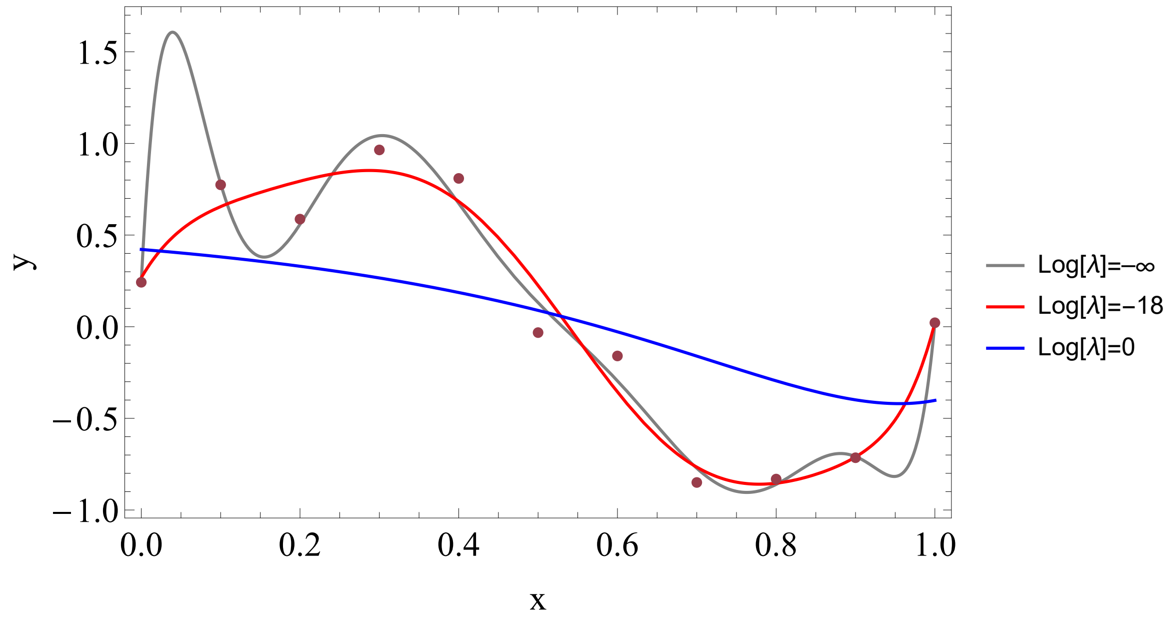
```
In[•]:= f2 = FindFit[Transpose[{x, y}], model, {β0, β1, β2, β3, β4, β5, β6, β7, β8, β9}, xx, FitRegularization → {"Tikhonov", Exp[-18.]}]
        f3 = FindFit[Transpose[{x, y}], model, {β0, β1, β2, β3, β4, β5, β6, β7, β8, β9}, xx, FitRegularization → {"Tikhonov", Exp[0.]}]

Out[•]= {β0 → 0.26751, β1 → 7.3913, β2 → -54.4953, β3 → 240.739,
         β4 → -500.324, β5 → 271.521, β6 → 291.865, β7 → -156.538, β8 → -287.912, β9 → 187.508}

Out[•]= {β0 → 0.421664, β1 → -0.362399, β2 → -0.414164, β3 → -0.300052,
         β4 → -0.171353, β5 → -0.0603349, β6 → 0.0291113, β7 → 0.0998359, β8 → 0.155684, β9 → 0.200032}
```

Regularization

Out[ ]=



As we see, for Log[λ] = 18. the overfitting is reduced and the fit looks ok. For even larger values of λ the quality of the fit becomes worse again, because even medium value coefficients are prohibited by the penalty function.

Regularization

The following Table compares the coefficients for all three cases:

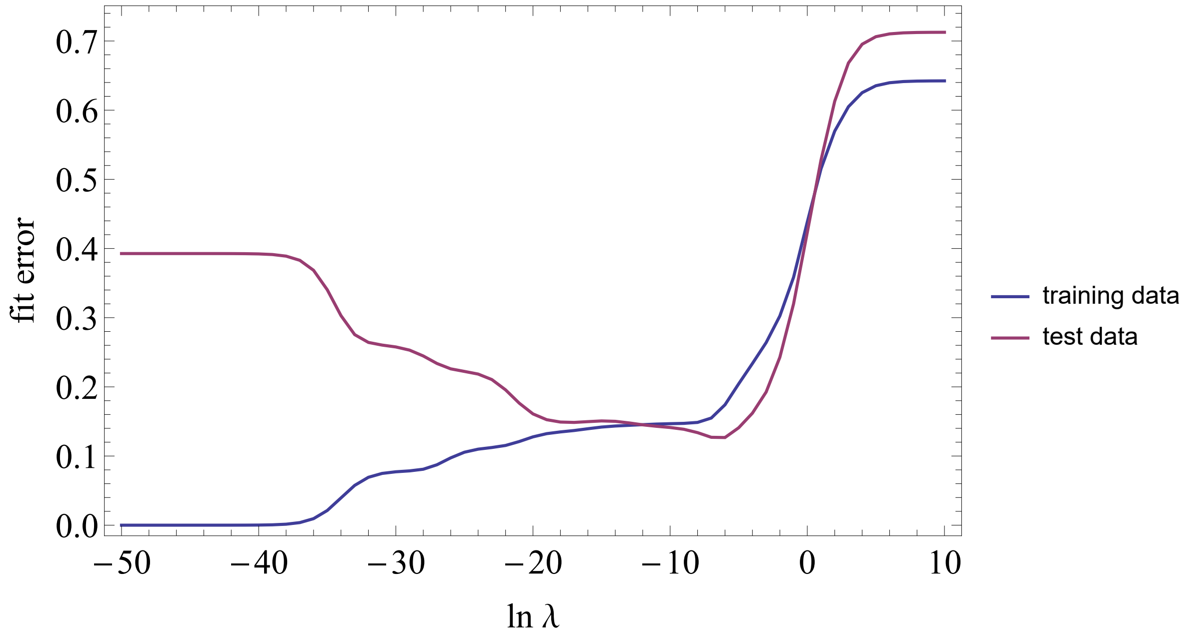
Out[ ]//TableForm=

	Log[λ]=-∞	Log[λ]=-18	Log[λ]=0
β0	0.241806	0.26751	0.421664
β1	86.1473	7.3913	-0.362399
β2	-1820.26	-54.4953	-0.414164
β3	15 466.7	240.739	-0.300052
β4	-68 521.9	-500.324	-0.171353
β5	176 142.	271.521	-0.0603349
β6	-273 172.	291.865	0.0291113
β7	252 233.	-156.538	0.0998359
β8	-127 706.	-287.912	0.155684
β9	27 291.2	187.508	0.200032

Regularization

The impact of the regularization term on the generalization error can be seen by plotting the value of the RMS error for both training and test sets against ln λ. We see that in effect λ now controls the effective complexity of the model and hence determines the degree of over-fitting.

Out[∘]=



Regularized least squares

Adding a regularization term to an error function now controls over-fitting, so that the total error function to be minimized takes the form:

Out[∘]//TraditionalForm=

$$\mathrm{E}_D(\beta) + \lambda\, \mathrm{E}_w(\beta)$$

One of the simples regularization is given by the sum-of-squares of the weight vector elements:

Out[∘]//TraditionalForm=

$$\mathrm{E}_w(\beta) = \frac{1}{2}\beta^{\mathsf{T}}\beta$$

Regularized least squares

We saw that the sum-of-squares error function is given by

Out[ ]//TraditionalForm=

$$\mathrm{E}_D(\beta) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \beta^{\mathsf{T}}\,\phi(x_n)\right)^2$$

then the total error function becomes

Out[ ]//TraditionalForm=

$$\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \beta^{\mathsf{T}}\,\phi(x_n)\right)^2 + \frac{\lambda}{2}\beta^{\mathsf{T}}\,\beta$$

This particular choice of regularizer is known in the machine learning literature as weight decay because in sequential learning algorithms, it encourages weight values to decay towards zero, unless supported by the data.

Regularized least squares

It has the advantage that the error function remains a quadratic function of β, and so its exact minimizer can be found in closed form. Specifically, setting the gradient of the total error function with respect to β to zero, and solving for β as before, we obtain

Out[ ]//TraditionalForm=

$$\boxed{\beta = (\lambda \mathrm{I} + \Phi^{T}\Phi)^{-1}\,\Phi^{\mathsf{T}}\,t}$$

A more general regularizer is sometimes used, for which the regularized error takes the form

Out[ ]//TraditionalForm=

$$\boxed{\frac{1}{2}\sum_{n=1}^{N}\left(t_n - \beta^{\mathsf{T}}\,\phi(x_n)\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{M}|\beta_j|^q}$$

where $q = 2$ corresponds to the quadratic regularizer from above. The case of q = 1 is know as the lasso in the statistics literature. It has the property that if λ is sufficiently large, some of the coefficients β_j are driven to zero, leading to a sparse model in which the corresponding basis functions play no role.

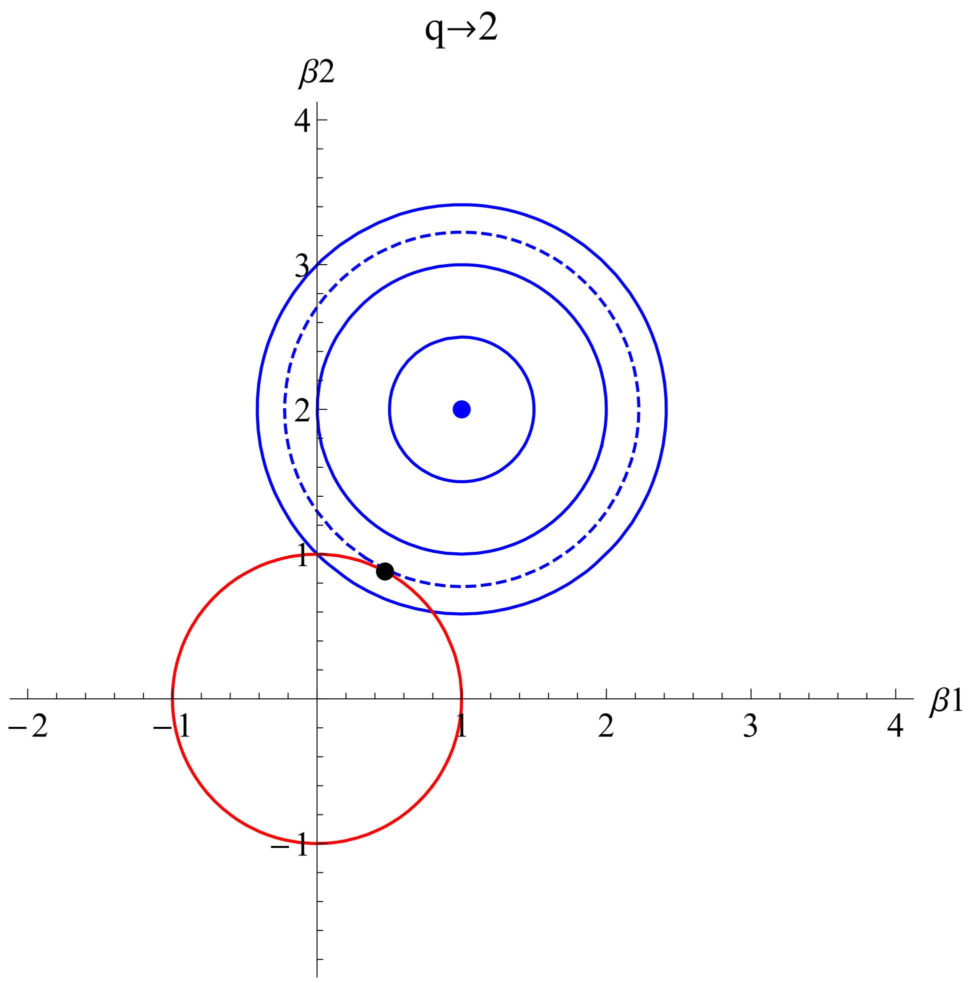
Regularized least squares

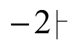
To get an intuitive idea on how the regularization term affects the results of the fit, we note that minimizing the regularized error is equivalent to minimizing the unregularized error function subject to the constraint

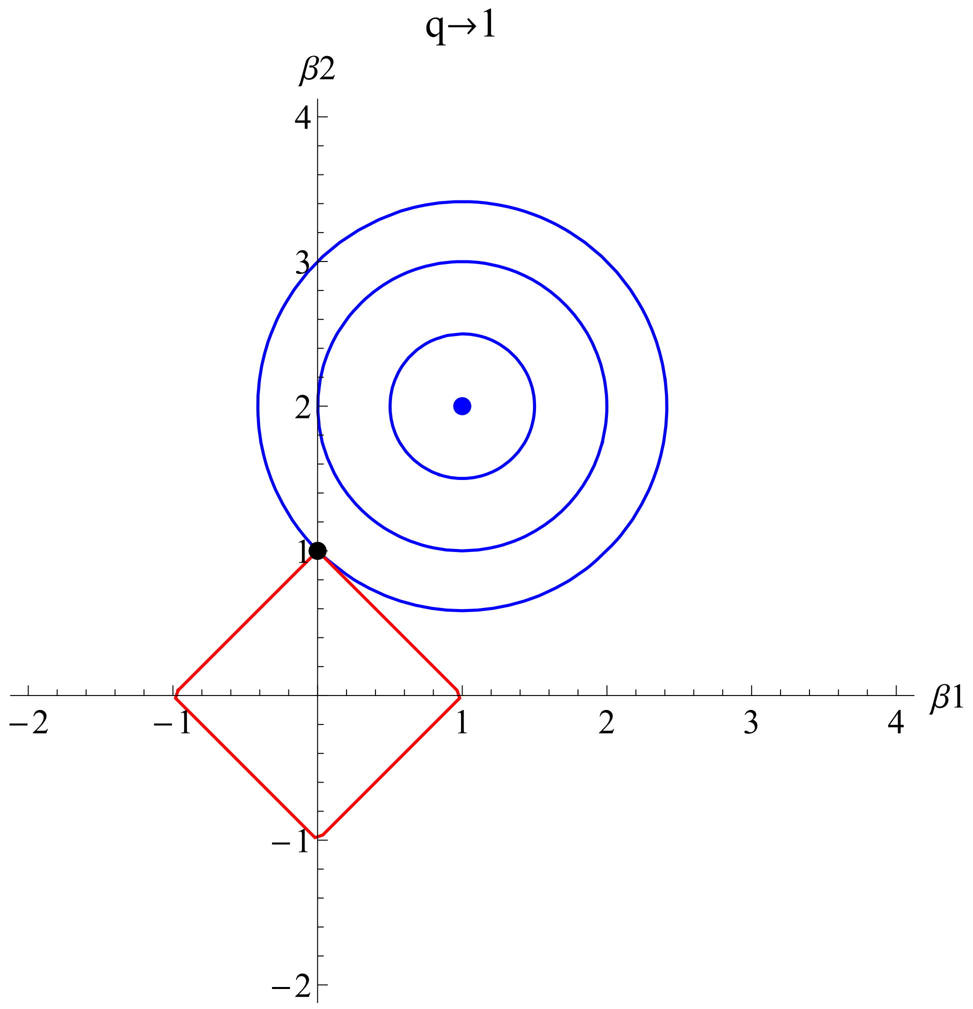
Out[]//TraditionalForm=

$$\sum_{j=1}^{M} |\beta_j|^q \leq \eta$$

Out[]=



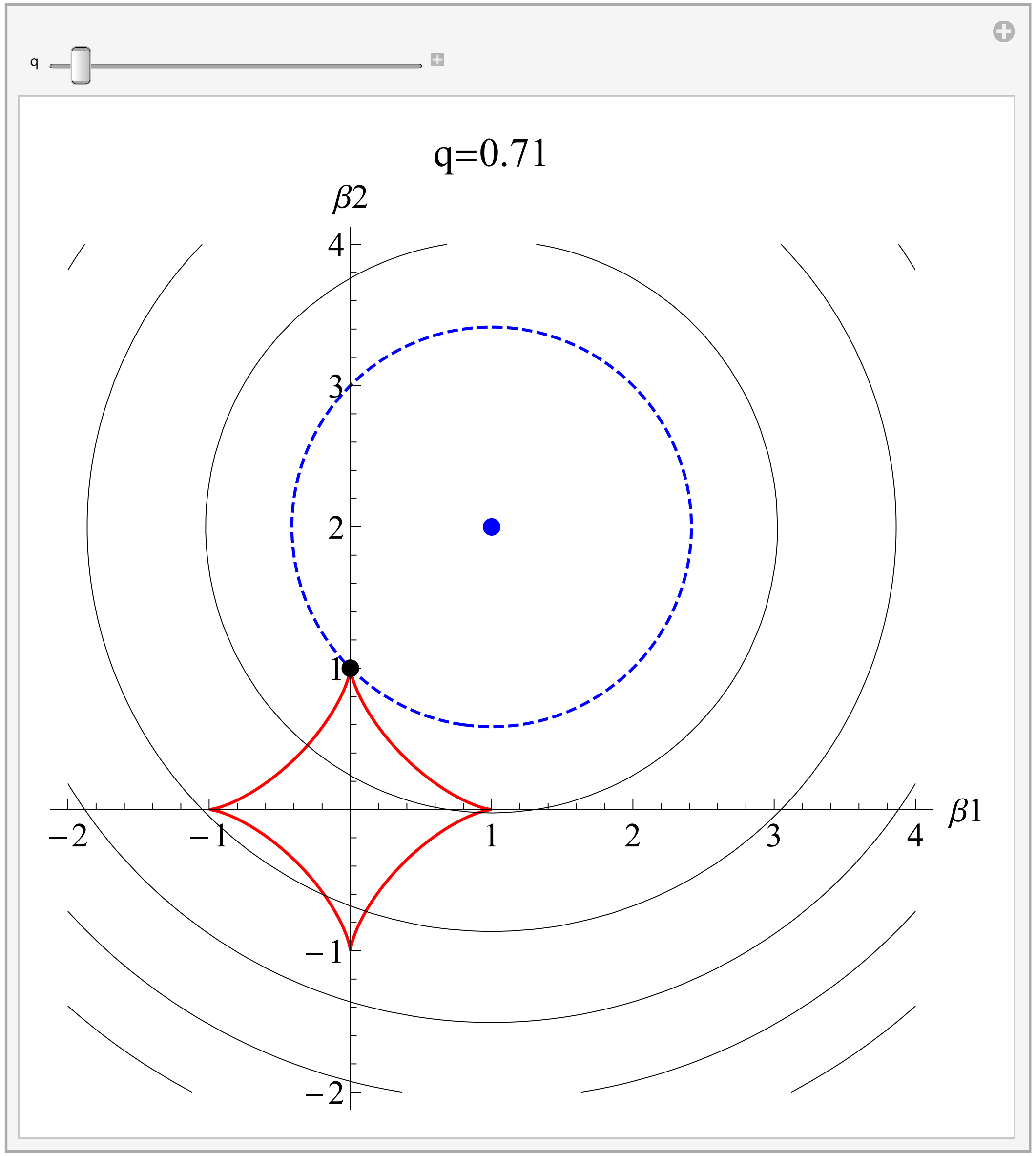

−2


q→1
β2
β1
4
3
2
1
−1
−2
−2
−1
1
2
3
4

Regularized least squares

The parameter q can be used to affect what minimum is chosen. Depending on the

Out[∘]=



Bias-Variance Decomposition

The question remains how to choose a suitable value for the regularization coefficient λ and for the number of basis functions, i.e. model parameters.

The goal is to formulate a loss function which leads to a corresponding optimal prediction once we are given the conditional distribution $p\,(t \mid x)$.

Bias-Variance Decomposition

A common choice of loss function in regression problems is the squared loss given by $L(t, y(x)) = \{y(x) - t\}^2$. In this case, the optimal specific estimate $y(x)$ of the value of t for each input x is

Out[ ]//TraditionalForm=

$$h(x) = y_{\text{opt}}(x) = \frac{\int t\, p(x, t)\, dt}{p(x)} = \int t\, p(t \mid x)\, dt = \mathbb{E}_t(t \mid x)$$

Bias-Variance Decomposition

The expected squared loss can be written in the form

Out[ ]//TraditionalForm=

$$\mathbb{E}(L) = \int (y(x) - h(x))^2\, p(x)\, dx + \int\int (h(x) - t)^2\, p(x, t)\, dx\, dt$$

The second term (independent of $y(x)$) arises from the intrinsic noise on the data and represents the minimum achievable value of the expected loss.

The first term depends on our choice for the function $y(x)$, and we will seek a solution for $y(x)$ which makes this term a minimum. Because it is nonnegative, the smallest that we can hope to make this term is zero.

In practice we have a data set D containing only a finite number N of data points, and consequently we do not know the regression function $h(x)$ exactly.

Bias-Variance Decomposition

Suppose we had a large number of data sets each of size N and each drawn independently from the distribution $p(t, x)$.

For any given data set D, we can run our fitting (learning) algorithm and obtain a prediction function $y(x; D)$.

Different data sets from the ensemble will give different functions and consequently different values of the squared loss.

The performance of a particular fitting/learning algorithm is then assessed by taking the average over this ensemble of data sets.

Bias-Variance Decomposition

Consider the integrand of the first term in the previous expression, which for a particular data set $\mathcal{D}$ takes the form

Out[ ]//TraditionalForm=

$$(y(x\,;\mathcal{D})-h(x))^2$$

The expectation value of this term with respect to $\mathcal{D}$ can be written as:

Out[ ]//TraditionalForm=

$$\mathbb{E}_{\mathcal{D}}\left((y(x\,;\mathcal{D})-h(x))^2\right)=\underbrace{(\mathbb{E}_{\mathcal{D}}(y(x\,;\mathcal{D}))-h(x))^2}_{(\text{bias})^2}+\underbrace{\mathbb{E}_{\mathcal{D}}\left((y(x\,;\mathcal{D})-\mathbb{E}_{\mathcal{D}}(y(x\,;\mathcal{D})))^2\right)}_{\text{variance}}$$

The first term is called the squared bias It represents the extent to which the average prediction over all data sets differs from the desired regression function. The second term, called the variance, measures the extent to which the solutions for individual data sets vary around their average, and hence this measures the extent to which the function $y(x;\mathcal{D})$ is sensitive to the particular choice of data set.

Bias-Variance Decomposition

Inserting this into the expression for $\mathbb{E}(L)$ gives the relation:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

where

$$(\text{bias})^2 = \int (\mathbb{E}_{\mathcal{D}}(y(x;\mathcal{D})) - h(x))^2 \, p(x) \, dx$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}}\left((y(x;\mathcal{D}) - \mathbb{E}_{\mathcal{D}}(y(x;\mathcal{D})))^2\right) p(x) \, dx$$

$$\text{noise} = \int\int (h(x) - t)^2 \, p(x, t) \, dx \, dt$$

Bias-Variance Decomposition

Our goal is to minimize the expected loss. As we shall see, there is a trade-off between bias and variance, with very flexible models having low bias and high variance, and relatively rigid models having high bias and low variance.

Out[•]=

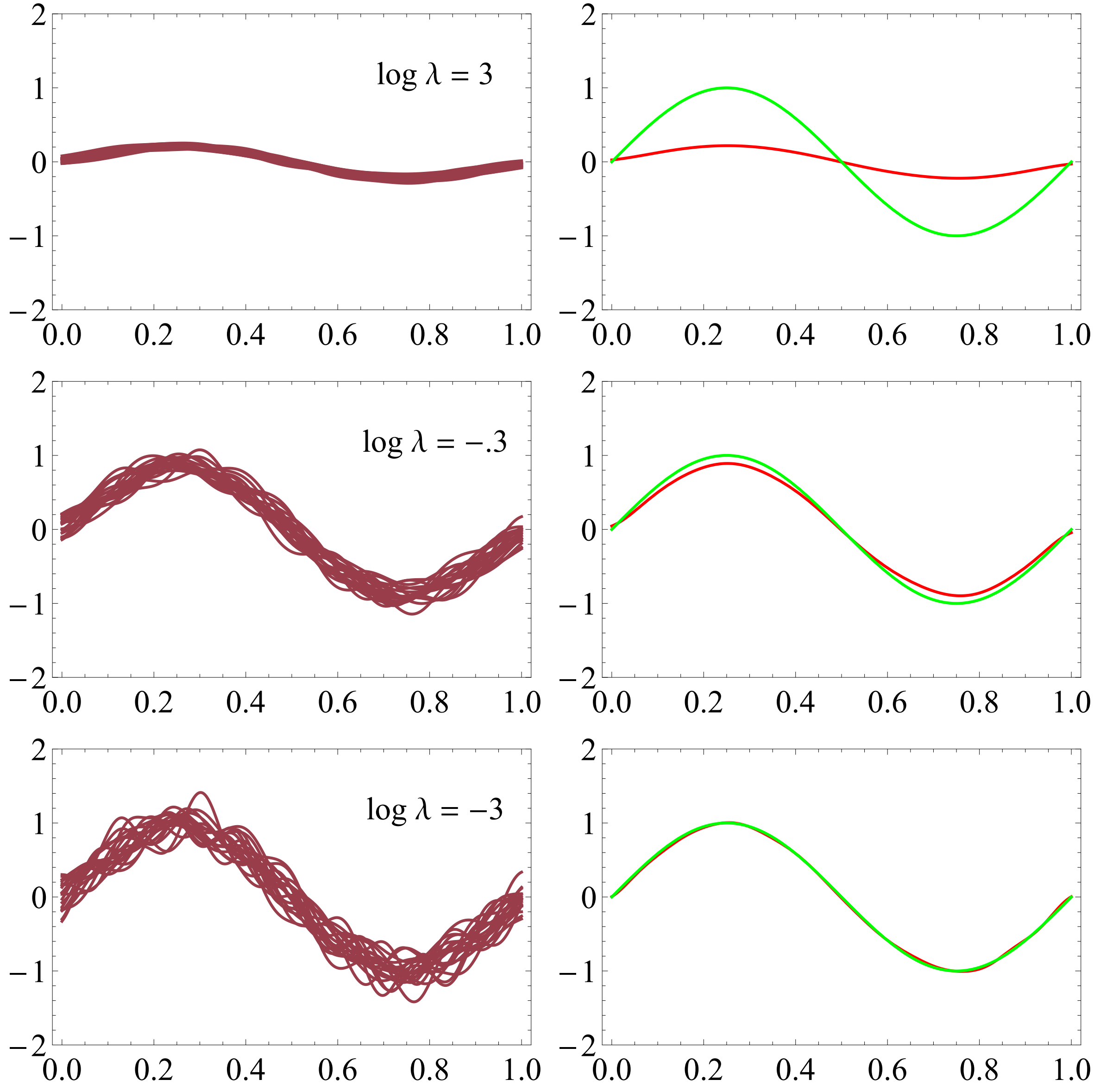


Illustration of the dependence of bias and variance on model complexity, governed by a regularization parameter λ. There are L = 100 data sets, each having N = 25 data points, and there are 24 Gaussian basis functions in the model so that the total number of parameters is M = 25 including the bias parameter (w_0).

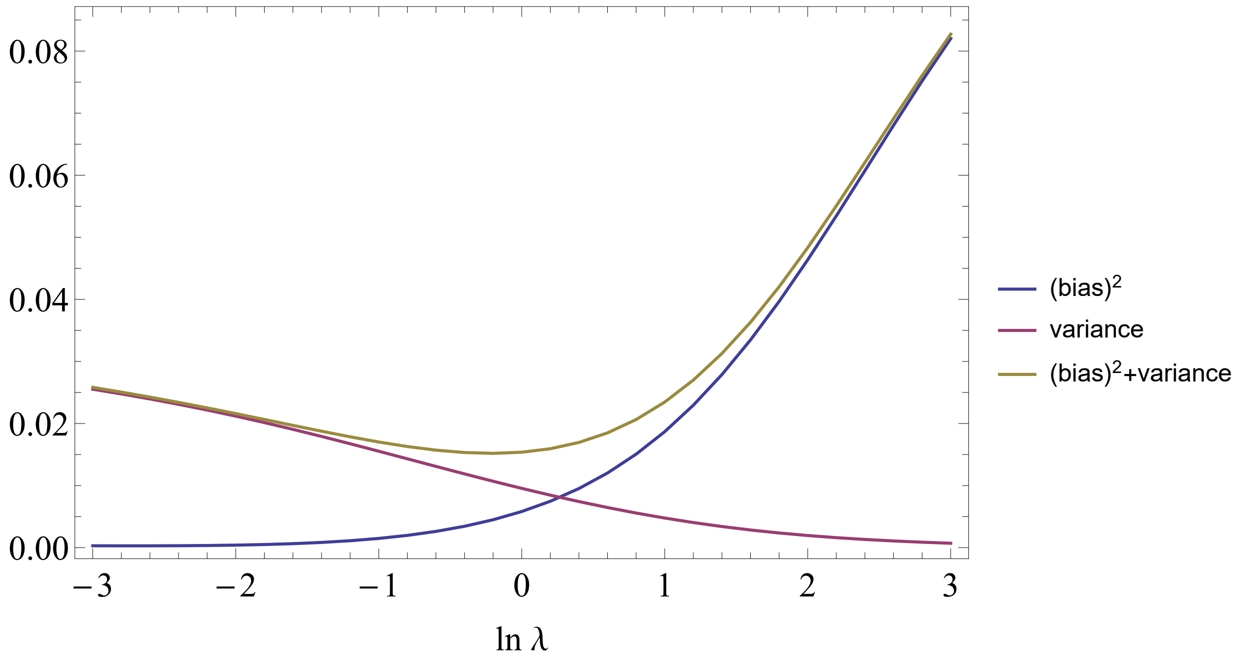
The left column shows the result of fitting the model to the data sets for various values of $\ln \lambda$ (for clarity, only 20 of the 100 fits are shown). The right column shows the corresponding average of the 100 fits (red) along with the sinusoidal function from which the data sets were generated (green).

- top row: large regularization $\longrightarrow$ low variance (all red curves close together), high bias (red and green curve very different)
- bottom row: small regularization $\longrightarrow$ high variance (large scatter of red curves), low bias (red and green curve agree well)

Bias-Variance Decomposition

We see that small values of λ allow the model to become finely tuned to the noise on each individual data set leading to large variance. Conversely, a large value of λ pulls the weight parameters towards zero leading to large bias.

Out[ ]=



- limited practical value
- based on averages to many sets of training data
- in practice we usually have only one data set

Gaussian Process

Gaussian processes (GPs) are non-parametric methods of modelling data.

- Instead of inferring a distribution over the parameters of a parametric function Gaussian processes can be used to infer a distribution over functions directly.
- A Gaussian process defines a prior over functions. After having observed some function values it can be converted into a posterior over functions.

Gaussian Process

A Gaussian process is a random process where any point $x \in \mathbb{R}^{\mathcal{D}}$ is assigned a random variable f(x) and where the joint distribution of a finite number of these variables $p(f\,(\text{x1}), \ldots, f\,(\text{xN}))$ is itself Gaussian

Out[ ]//TraditionalForm=

$$p(f \mid X) = \mathcal{N}(f \mid \mu, K)$$

- $f = \{f(x_1), \ldots, f(x_N)\}$
- $\mu = \{m(x_1), \ldots, m(x_N)\}$ m: mean function
- $K_{\text{ij}} = k(x_i, x_j)$ k: a positive definite kernel function or covariance function.

Gaussian process is a distribution over functions whose shape (smoothness, …) is defined by K. If points x_i and x_j are considered to be similar by the kernel the function values at these points, $f(x_i)$ and $f(x_j)$, can be expected to be similar too.

Gaussian Process

Returning to linear regression, consider a model defined in terms of a linear combination of M fixed basis functions given by the elements of the vector ϕ(x) so that

Out[∘]//TraditionalForm=

$$y(x) = \beta^{\mathsf{T}}.\phi(x)$$

where x is the input vector and β is the M-dimensional weight vector. Now consider a prior distribution over β given by an isotropic Gaussian of the form

Out[∘]//TraditionalForm=

$$p(\beta) = \mathcal{N}(\beta \mid 0,\ \alpha^{-1}\ \mathrm{I})$$

the parameter α is the inverse variance of the distribution. For any given value of β, the definition of $y(x)$ defines a particular function of x.

The probability distribution over β therefore induces a probability distribution over functions $y\,(x)$.

Gaussian Process

In practice, we evaluate this function at specific values of x, our data points $x_1, \ldots, x_N$. Thus we need the joint distribution of the function values $y(x_1), \ldots, y(x_N)$, which we abbreviate $y_n = y(x_n)$.

Out[∘]//TraditionalForm=

$$y = \Phi.\beta$$

with the design matrix Φ with elements $\Phi_{\mathrm{nk}} = \phi_k(x_n)$.

y is a linear combination of Gaussian distributed variables and is therefore itself Gaussian. Its mean and covariance are

Out[∘]//TraditionalForm=

$$\mathbb{E}(y) = \Phi\,\mathbb{E}(\beta) = 0$$

Out[∘]//TraditionalForm=

$$\mathrm{cov}(y) = \mathbb{E}\left(y.y^{\mathsf{T}}\right) = \Phi\,\mathbb{E}\left(\beta.\beta^{\mathsf{T}}\right).\Phi^{\mathsf{T}} = \frac{1}{\alpha}\Phi.\Phi^{\mathsf{T}} = K$$

Gaussian Process

K is the Gram matrix with elements

Out[∘]//TraditionalForm=

$$K_{\mathrm{nm}} = k(x_n,\, x_m) = \frac{1}{\alpha}\, \phi(x_n)^{\mathsf{T}}.\phi(x_m)$$

k is the kernel function. Assuming a mean zero, the specification of a Gaussian process is complete by giving the covariance of *y* (*x*) evaluated at any two values of *x* which is given by the kernel function. We can also define the kernel function directly, rather than indirectly through a choice of basis function.

Common choices are a Gaussian kernel or an exponential kernel:

Out[∘]//TraditionalForm=

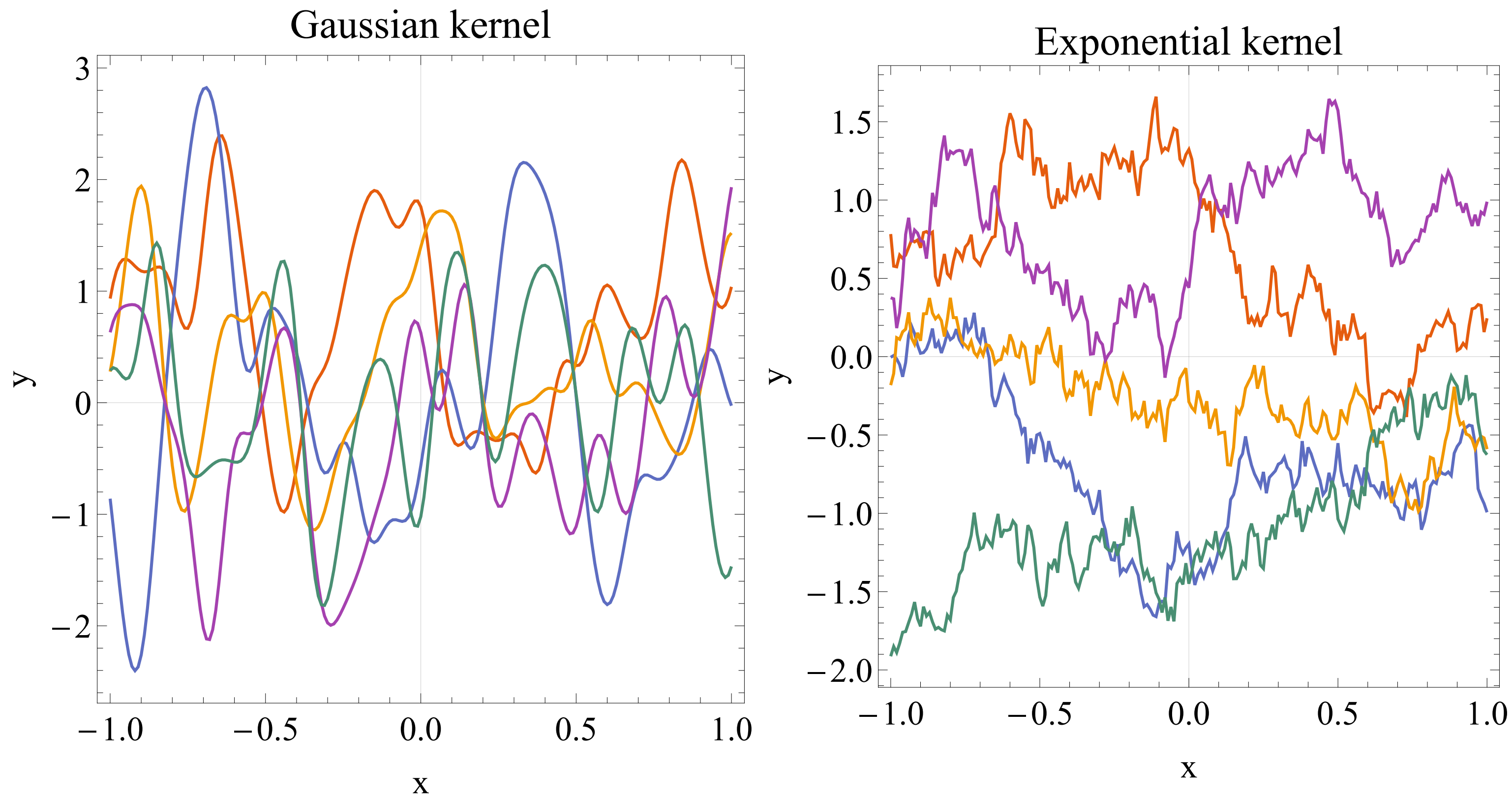
$$\boxed{k(x_m,\, x_n) = \exp\left(-\frac{\|x_n - x_m\|^2}{2\,\sigma^2}\right)}$$

Out[∘]//TraditionalForm=

$$\boxed{k(x_m,\, x_n) = \exp(-\theta\,|x_n - x_m|)}$$

Gaussian Process

Out[∘]=



Samples from Gaussian processes for a ‘Gaussian’ kernel (left) and an exponential kernel (right).

Gaussian processes for regression

Usually, data is noisy, i.e a combination of signal y_n + random noise ϵ_n

Out[∘]//TraditionalForm=

$$t_n = y_n + \epsilon_n$$

Assuming normally distributed noise, this is equivalent to

Out[∘]//TraditionalForm=

$$p(t_n \mid y_n) = \mathcal{N}\left(t_n \mid y_n, \frac{1}{\gamma}\right)$$

where the hyperparameter γ represents the precision of the noise. The noise is now described by a normal distribution of all possible sample points t_n scattered normally around the real signal y_n . The noise of all data points is uncorrelated, so we can write the joint distribution:

Out[∘]//TraditionalForm=

$$p(t \mid y) = \mathcal{N}\left(t \mid y, \frac{\mathrm{I}_N}{\gamma}\right)$$

Gaussian processes for regression

We can describe the marginal distribution $p(\boldsymbol{y})$ over all possible functions y in terms of a Gaussian process:

Out[ ]//TraditionalForm=

$$p(y) = \mathcal{N}(y \mid 0, K)$$

From Bayes theorem, it follows, that in order to find the marginal distribution $p(t)$, conditioned on the input values $x_1, \ldots, x_N$, we need to integrate over y.

Out[ ]//TraditionalForm=

$$\boxed{p(t) = \int p(t \mid y)\, p(y)\, dy = \mathcal{N}(t \mid 0, C)}$$

where the covariance matrix C has elements

Out[ ]//TraditionalForm=

$$C(x_n, x_m) = k(x_n, x_m) + \gamma^{-1}\, \delta(n, m)$$

Gaussian processes for regression

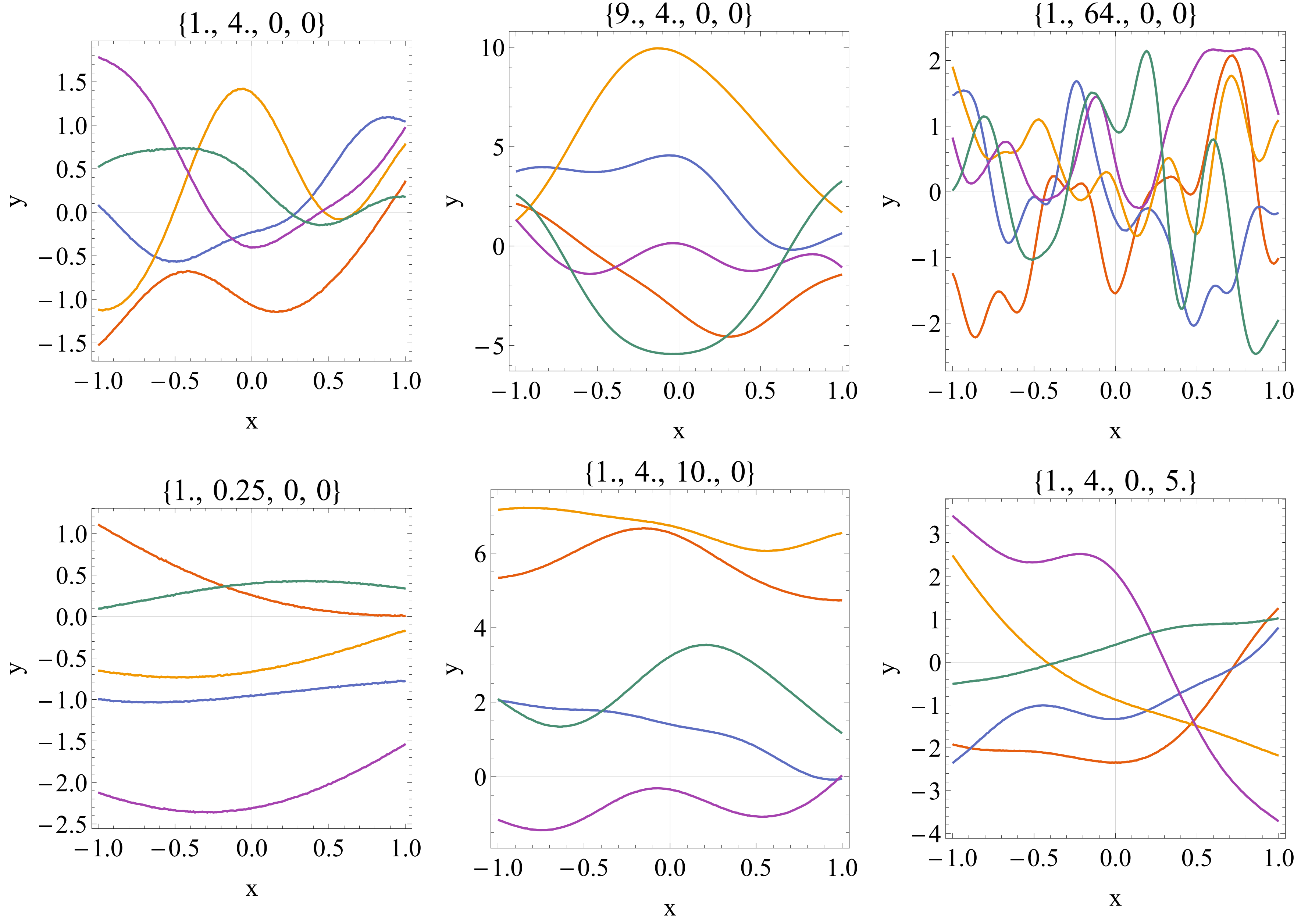
Another widely used kernel function combines a Gaussian kernel with a linear and a constant term

Out[ ]//TraditionalForm=

$$\boxed{k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 (x_n)^{\mathsf{T}}.x_m}$$

Samples from a Gaussian process prior defined by the covariance function above are shown in the set of Figures below. The title above each plot denotes $(\theta_0, \theta_1, \theta_2, \theta_3)$.

Out[∘]=



Gaussian processes for regression

So far, we didn’t use any data yet. We just specified the x values, where we compute the kernel function. Our goal in regression, however, is to make predictions of the target variables for new inputs, given a set of training data.

Let us suppose that $\boldsymbol{t}_N = (t_1, \ldots, t_N)^T$, corresponding to input values $x_1, \ldots, x_N$, comprise the observed training set, and our goal is to predict the target variable $\boldsymbol{t}_{N+1}$ for a new input vector $\boldsymbol{x}_{N+1}$. This requires that we evaluate the predictive distribution $p(t_{N+1} \mid \boldsymbol{t}_N)$.

Gaussian processes for regression

We start by writing down the joint distribution

Out[ ]//TraditionalForm=

$$\boxed{p(t_{N+1}) = \mathcal{N}(t_{N+1} \mid 0, C_{N+1})}$$

where $\boldsymbol{t}_{N+1} = (t_1, \ldots, t_N, t_{N+1})^T$ and C_{N+1} is an $(N+1) \times (N+1)$ covariance matrix. We can partition the covariance matrix as follows

Out[ ]//TraditionalForm=

$$C_{N+1} = \begin{pmatrix} C_N & \mathrm{k} \\ \mathrm{k}^{\mathsf{T}} & c \end{pmatrix}$$

where C_N is the $N \times N$ covariance matrix with elements for $n, m = 1, \ldots, N$, the vector k has elements $k(x_n, x_{N+1})$ for $n = 1, \ldots, N$, and the scalar $c = k(x_{N+1}, x_{N+1}) + \gamma^{-1}$. From this we know that the conditional distribution $p(\boldsymbol{t}_{N+1} \mid \boldsymbol{t})$ is a Gaussian distribution with mean and covariance given by

Out[ ]//TraditionalForm=

$$\boxed{m(x_{N+1}) = k^{\mathsf{T}}.(C_N)^{-1}.t}$$

Out[ ]//TraditionalForm=

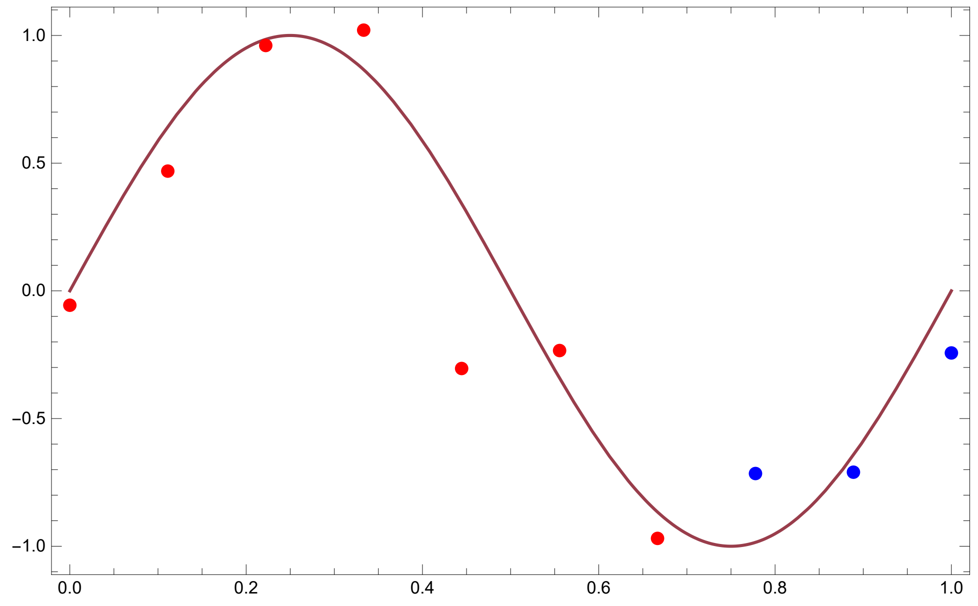
$$\boxed{\sigma^2(x_{N+1}) = c - k^{\mathsf{T}}.(C_N)^{-1}.k}$$

This formalism is straight forward to extend for an arbitrary number of new input points.

Example

Let’s look at a practical example. We take noisy sinusoidal data as shown in the Figure below. We remove the last three data points and take the remaining 7 as training data. In the following, we will compute predictions for the last three data points, which we call test data.

Out[ ]=



Example

Setting up the example:

In[ ]:=
```
xs = testData[[All, 1]];
xobs = trainingData[[All, 1]];
yobs = trainingData[[All, 2]];
σ1 = 1.; σ0 = 10^-1; lengthScale = 0.1;
kernel1 = Function[{x1, x2}, σ1^2 * Exp[-(EuclideanDistance[x1, x2]/lengthScale)^2]];
nugget = Function[x, σ0^2];
nugget0 = Function[x, 0.];
```

Example

Now we compute the elements of the covariance matrix. C_N itself is a covariance matrix across all test data points. c is the covariance matrix across the new data points. k mixes the test and the training data.

Out[∘]//TraditionalForm=

$$C_{N+1} = \begin{pmatrix} C_N & \mathrm{k} \\ \mathrm{k}^{\mathsf{T}} & c \end{pmatrix}$$

In[∘]:=
```
k = Transpose@Table[kernel1[xs[[i]], #] & /@ xobs, {i, 1, Length[xs]}];
c = covarianceMatrix[xs, kernel1, nugget0];
Cn = covarianceMatrix[xobs, kernel1, nugget];
```

Example

Now we can compute the mean value and the standard deviation for each *x* value of the test data:

Out[ ]//TraditionalForm=

$$\boxed{m(x_{N+1}) = k^{\mathsf{T}}.(C_N)^{-1}.t}$$

Out[ ]//TraditionalForm=

$$\boxed{\sigma^2(x_{N+1}) = c - k^{\mathsf{T}}.(C_N)^{-1}.k}$$

In[ ]:=
```
mus = (Transpose[k].Inverse[Cn]).yobs;
sigma = c - Transpose[k].Inverse[Cn].k;
(* numerical Problems ahead!*)
sigma = c - 1/2 (Transpose[k].Inverse[Cn].k + Transpose[Transpose[k].Inverse[Cn].k]);
stds = (Diagonal@sigma)^(1/2);
errxs = Join[xs, Reverse[xs]];
errys = Join[mus + 2 stds, Reverse[mus - 2 stds]];
```

Example

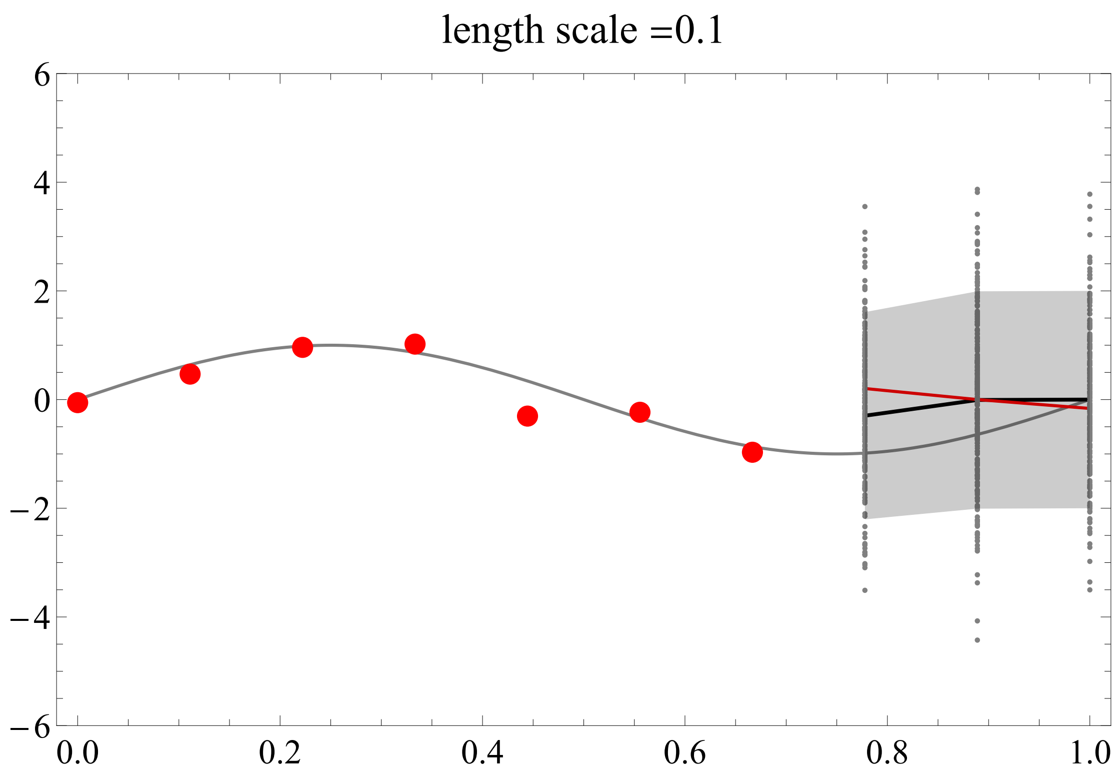
The following is just to ensure, that the covariance matrix σ^2 is numerically positive definite.

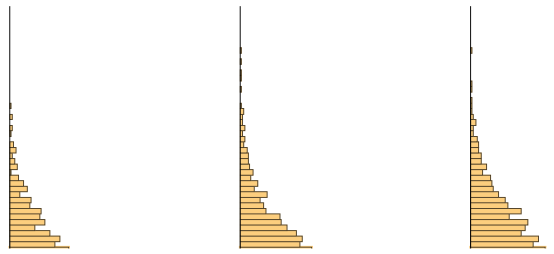
```
In[ ]:= Σs = sigma;
Σs = Σs + 2 * Abs@Min[Eigenvalues[Σs]] * IdentityMatrix[Length[Σs]]; Σs = (Σs + Transpose[Σs]) / 2;
```

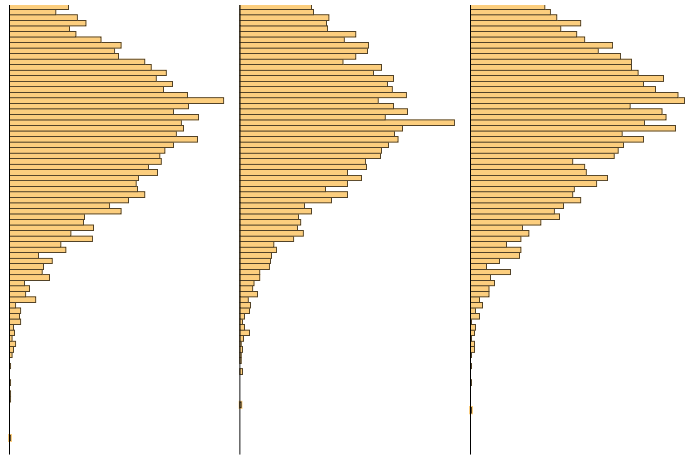
Example

Now we can draw random samples from the multi-normal distribution, that is fully described by the mean values $m(x_{N+1})$ and the covariance matrix $\sigma^2(x_{N+1})$. Every single sample is a prediction of function values for all x-values of the test data set (3 in our case).

Out[ ]=





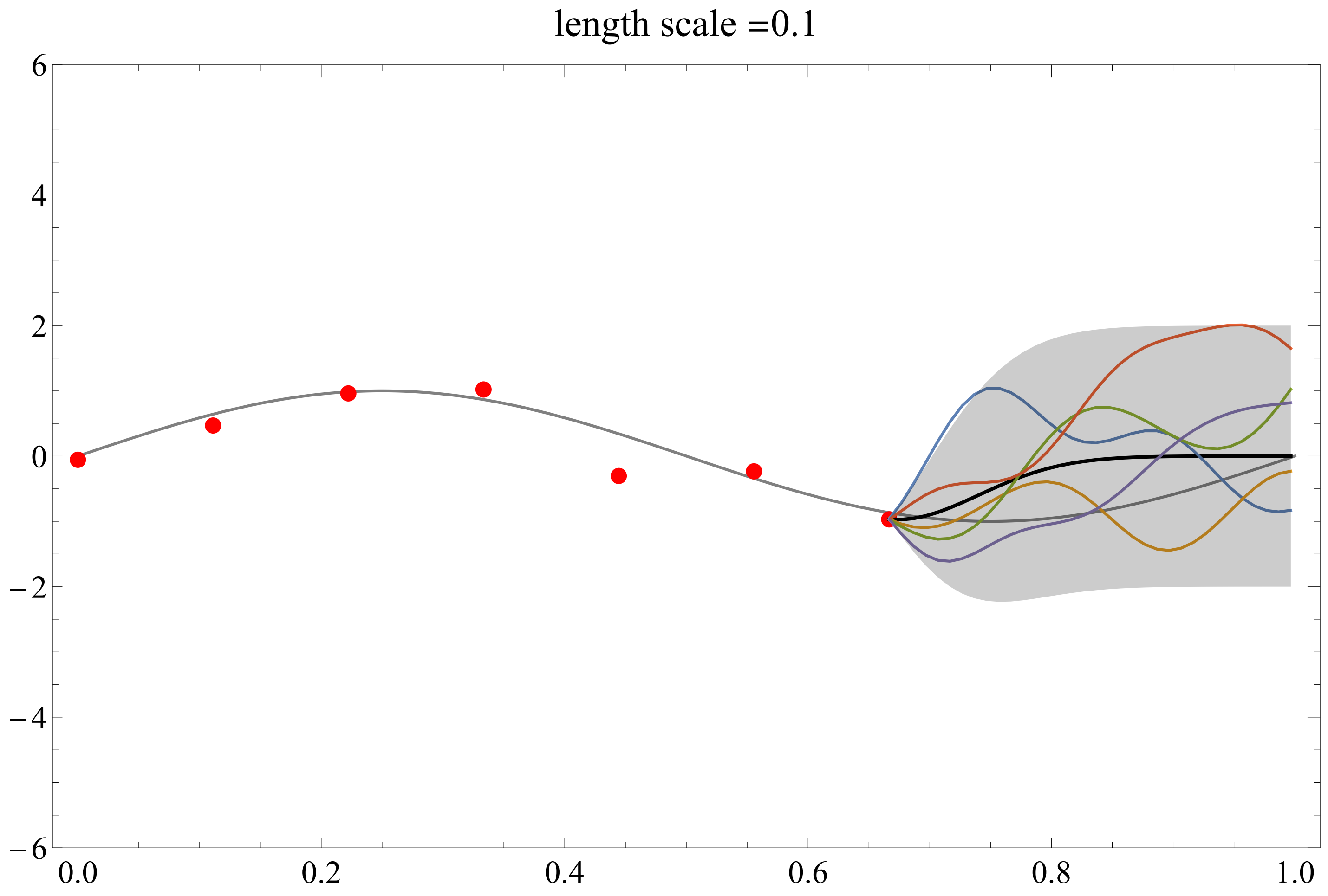


The gray points are 250 sampled predictions for each test x- value. The histograms on the right show the respective distribution. The gray area indicates the 2-standard deviations range around the respective mean values of our predictions. The red line connects the three sampled points from one random sample.

Example

If we put in a large number of x-values right from our training data set, we get a much higher resolved prediction:

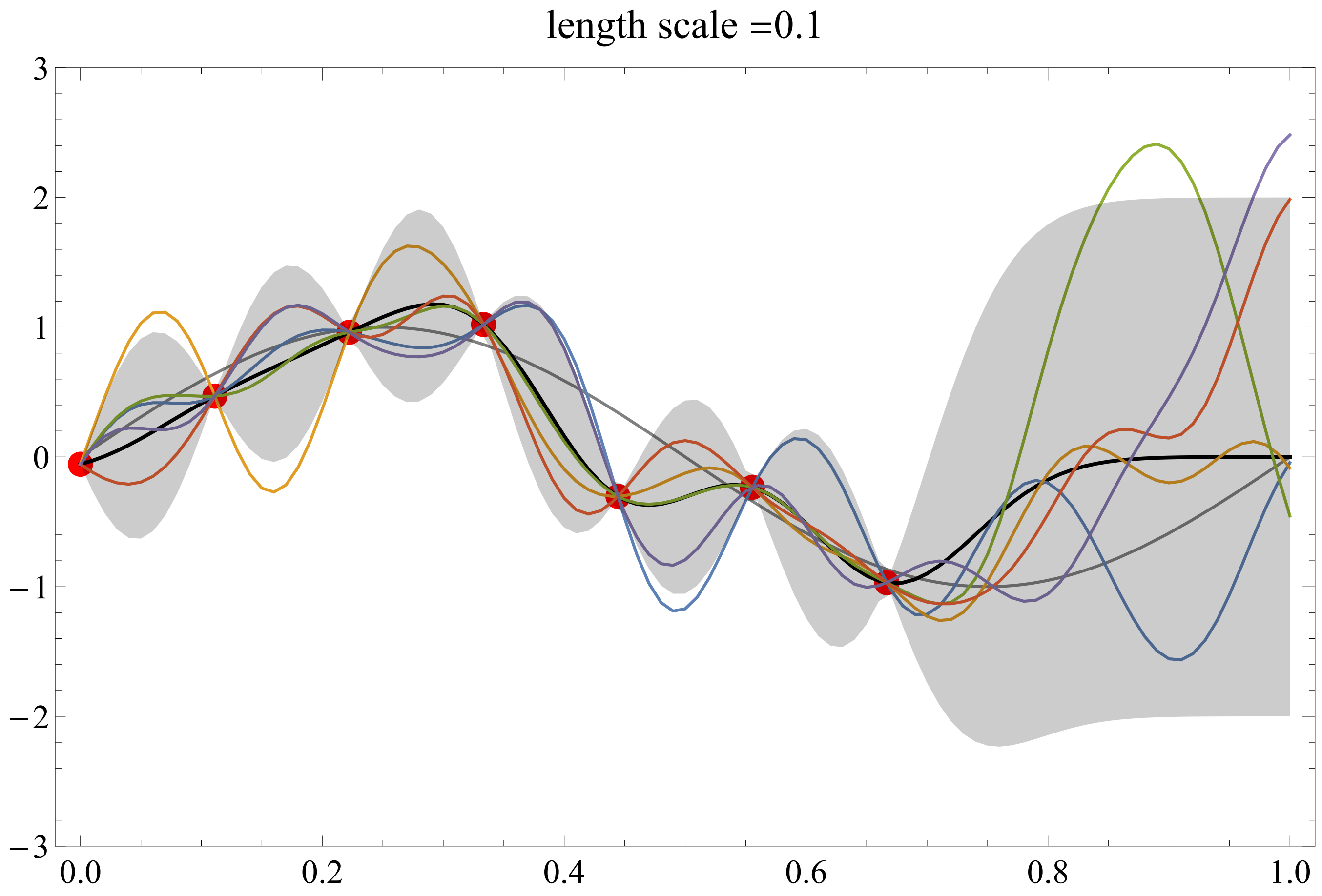
Out[ ]=



Example

Now display, how the fit looks across the full range. To do so, we put in a test data set, i.e. x-values, that cover the full range of our plot.

Out[ ]=

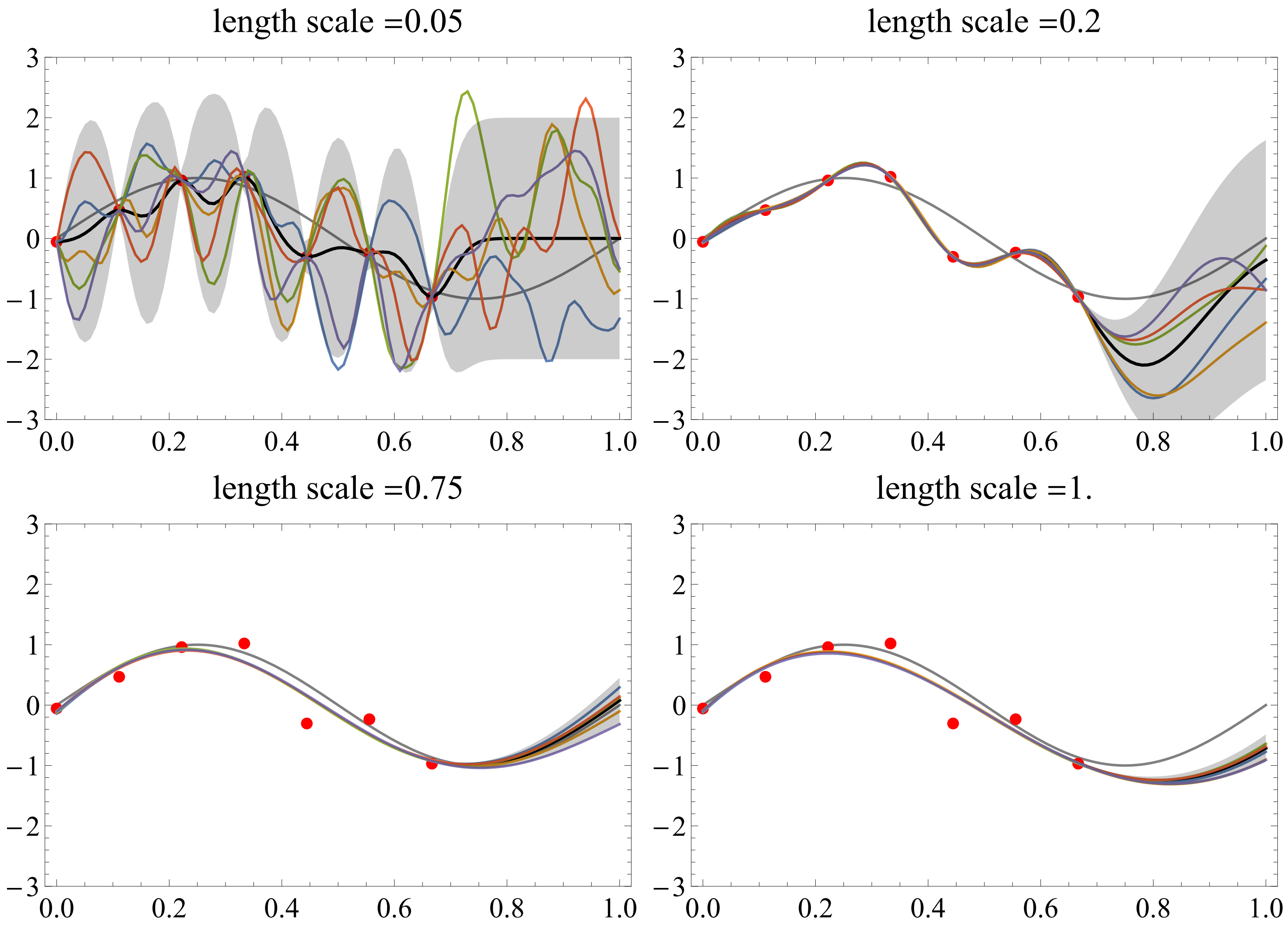


We note, that the gray area indicating our uncertainty of the predictions is small close to our training data points (of course!) and larger between our data points. It becomes largest, when we leave the training range.

Example

Note, that the mean predictions approaches a zero mean for far extrapolations, which means we lack information of how the function behaves far to the right. **This is a consequence of the length scale in our kernel!**

Out[ ]=



length scale =0.05
length scale =0.2
length scale =0.75
length scale =1.
3
2
1
0
−1
−2
−3
0.0
0.2
0.4
0.6
0.8
1.0

Noisy Data

In case of noisy training data, the matrices C_N and c are modified such that their diagonal elements carry the additional noise contribution

Out[ ]//TraditionalForm=

$$C_{N+1} = \begin{pmatrix} C_N + \sigma_N.\mathrm{I} & \mathrm{k} \\ \mathrm{k}^\mathsf{T} & c + \sigma.\mathrm{I} \end{pmatrix}$$

where σ_N and σ are the error vectors of the training data and test data points respectively.

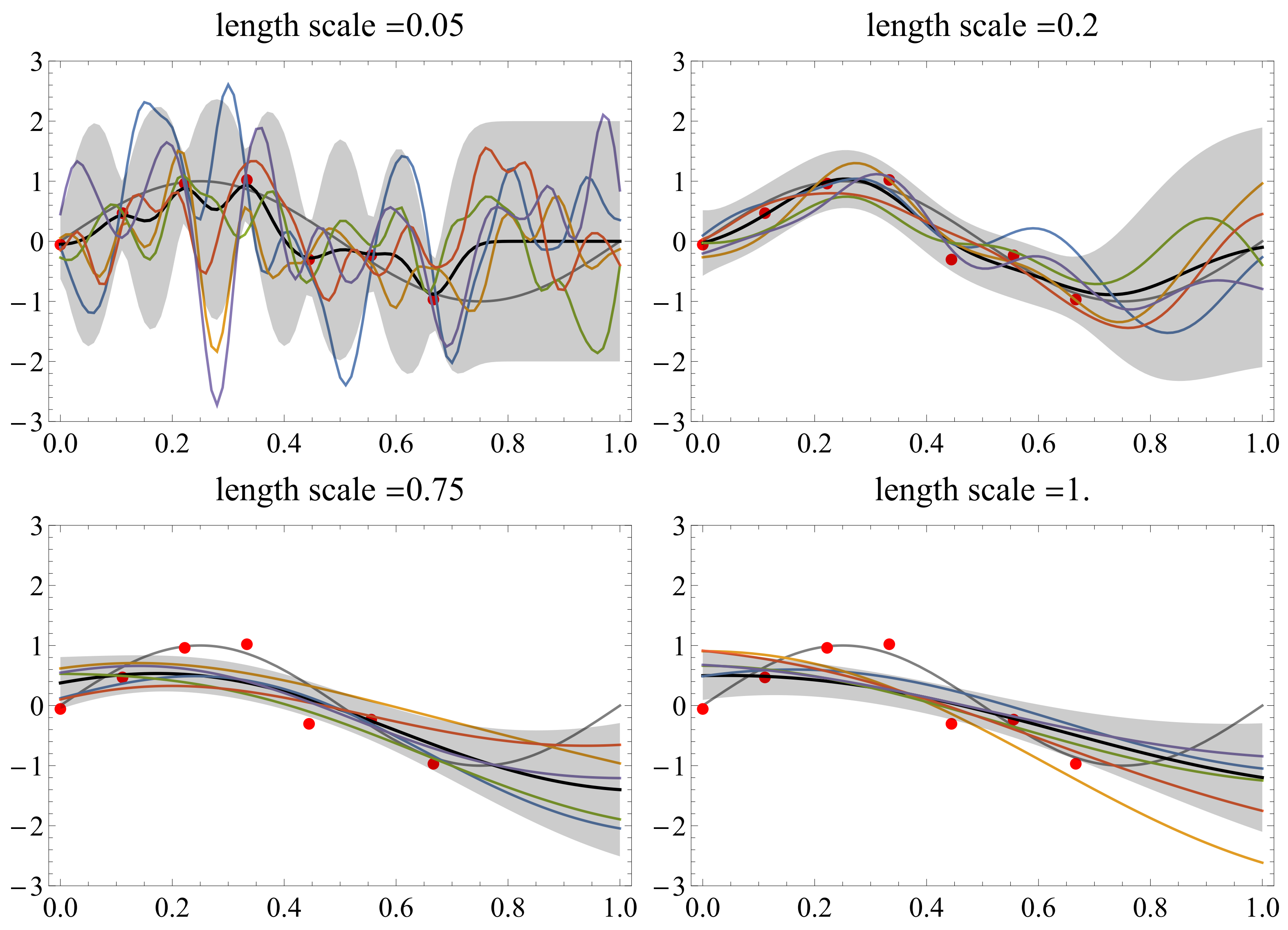
Note, that by construction the matrices C_{N+1}, and C_N and c are symmetric and positive definite. However, due to finite numerical floating point precision, they frequently are neither symmetric nor positive definite. This prohibits sampling from the Multinormal distribution. To circumvent this problems, several steps might necessary:

- (small) numerical noise terms σ_N and σ, called nuggets, are added to C_N and c even in case of no stochastic noise
- symmetric, positive definite approximations of C_N and c are computed. Many algorithms are proposed to find a good approximation. The easiest solution to make a square matrix $\mathcal{M}$ symmetric is to compute $\frac{1}{2}(\mathcal{M} + \mathcal{M}^T)$

Noisy Data

The Figure below show the same results as in the previous section but assuming a noise level of 0.3.

Out[ ]=



We note, that large length scales lead to bad approximations of individual data points, which are not included into the 95% confidence interval any more.

Choosing the hyperparameters

We saw, that the prediction is a sensitive function of the assumed hyperparameters of the kernel. Techniques for learning the hyperparameters are based on the evaluation of the likelihood function $p(\boldsymbol{t} \mid \theta)$ where θ denotes the hyperparameters of the Gaussian process model. The simplest approach is to make a point estimate of θ by maximizing the log likelihood function.

Out[ ]//TraditionalForm=

$$\log(p(t \mid \theta)) = -\frac{1}{2}\log(|C_N|) - \frac{1}{2}\, t^{\mathsf{T}}.(C_N)^{-1}.t - \frac{N}{2}\log(2\,\pi)$$

To maximize the likelihood, we have to find roots of the gradient:

Out[ ]//TraditionalForm=

$$\frac{\partial}{\partial\theta_i}\log(p(t \mid \theta)) = -\frac{1}{2}\,\mathrm{Tr}\left[(C_N)^{-1}\,\frac{\partial C_N}{\partial\theta_i}\right] + \frac{1}{2}\, t^{\mathsf{T}}.(C_N)^{-1}.\frac{\partial C_N}{\partial\theta_i}.(C_N)^{-1}.t$$

Because $\log p(t \mid \theta)$ will in general be a nonconvex function, it can have multiple maxima.

Choosing the hyperparameters

```
In[ ]:= xs = testData[[All, 1]];
        xobs = trainingData[[All, 1]];
        yobs = trainingData[[All, 2]];
        σ1 =.; σ0 = 0.3; lengthScale =.;
```

$$\text{kernel1 = Function}\left[\{x1, x2\}, \sigma1^2 * \text{Exp}\left[-\left(\frac{\text{EuclideanDistance}[x1, x2]}{\text{lengthScale}}\right)^2\right]\right];$$

$$\text{nugget = Function}\left[x, \sigma0^2\right];$$

```
        k = Transpose@Table[kernel1[xs[[i]], #] & /@ xobs, {i, 1, Length[xs]}];
        c = covarianceMatrix[xs, kernel1, nugget];
        Cn = covarianceMatrix[xobs, kernel1, nugget];

In[ ]:= Clear[f]
```

$$\text{f[lengthScale\_?NumericQ] := -0.5 Log[Det[Cn]] - 0.5 yobs.Inverse[Cn].yobs} - \frac{\text{Length[xobs]}}{2}\text{Log}[2\pi]$$

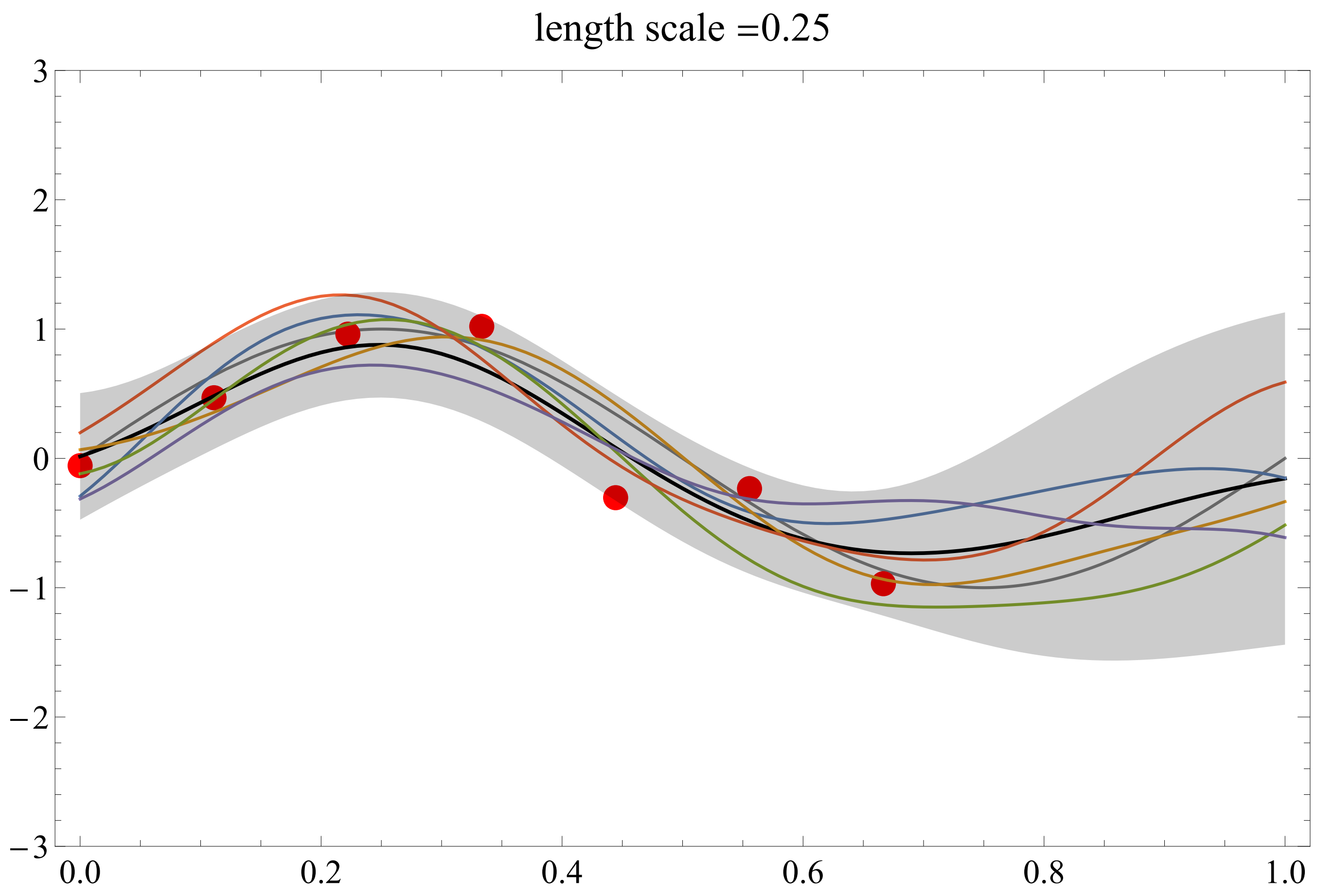
```
In[ ]:= NMaximize[{f[lengthScale], 5 > lengthScale > 0.01, σ1 > 0}, {lengthScale, σ1}, Method → "NelderMead"]

Out[ ]= {-6.20339, {lengthScale → 0.249647, σ1 → 0.652829}}
```

Choosing the hyperparameters

```
In[]:= gaussianProcess[0.25, 0.3, .653]
```

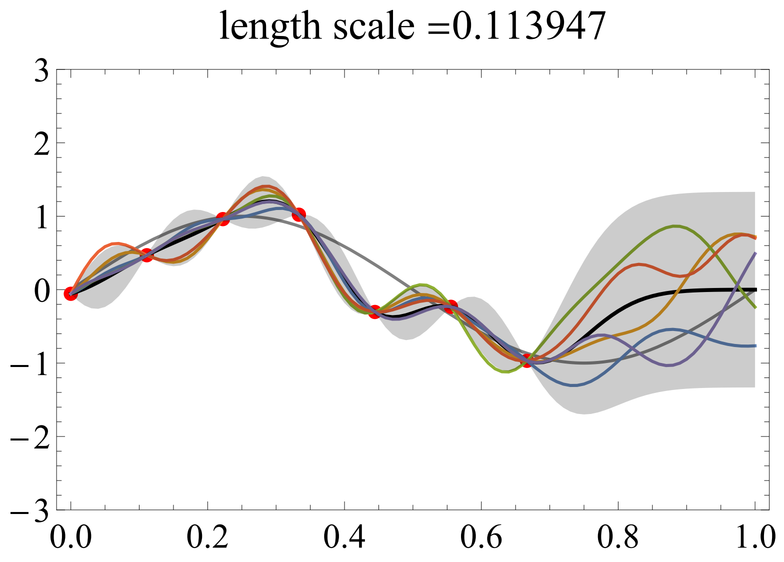
Out[]=



Choosing the hyperparameters

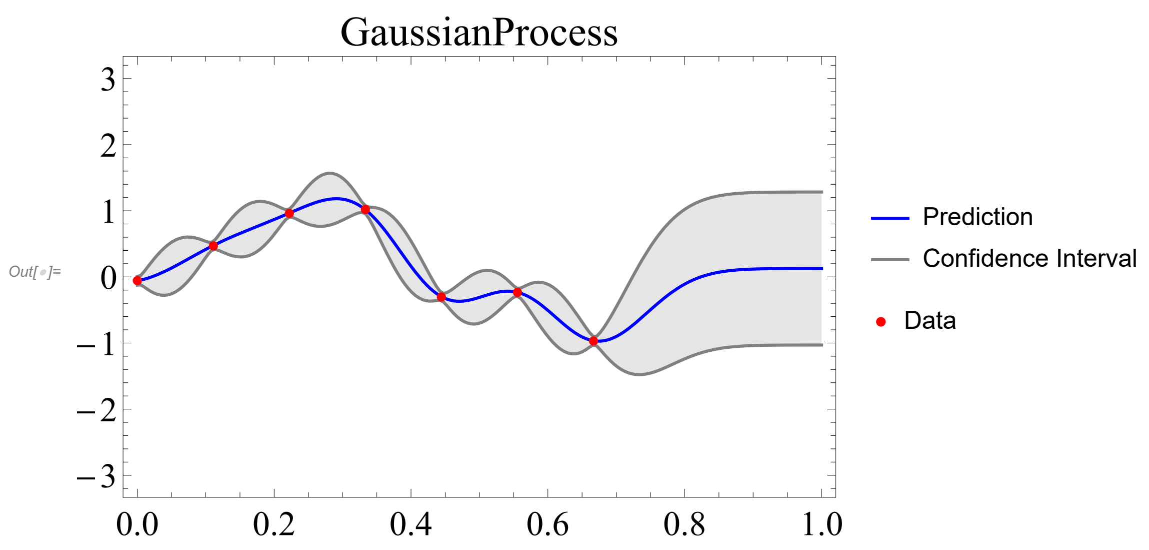
We can test our results with the built-in routines. First we find the optimal solution for a noise free data set:

Out[ ]=



and compare with the built-in Mathematica Gaussian process prediction:

Out[ ]=

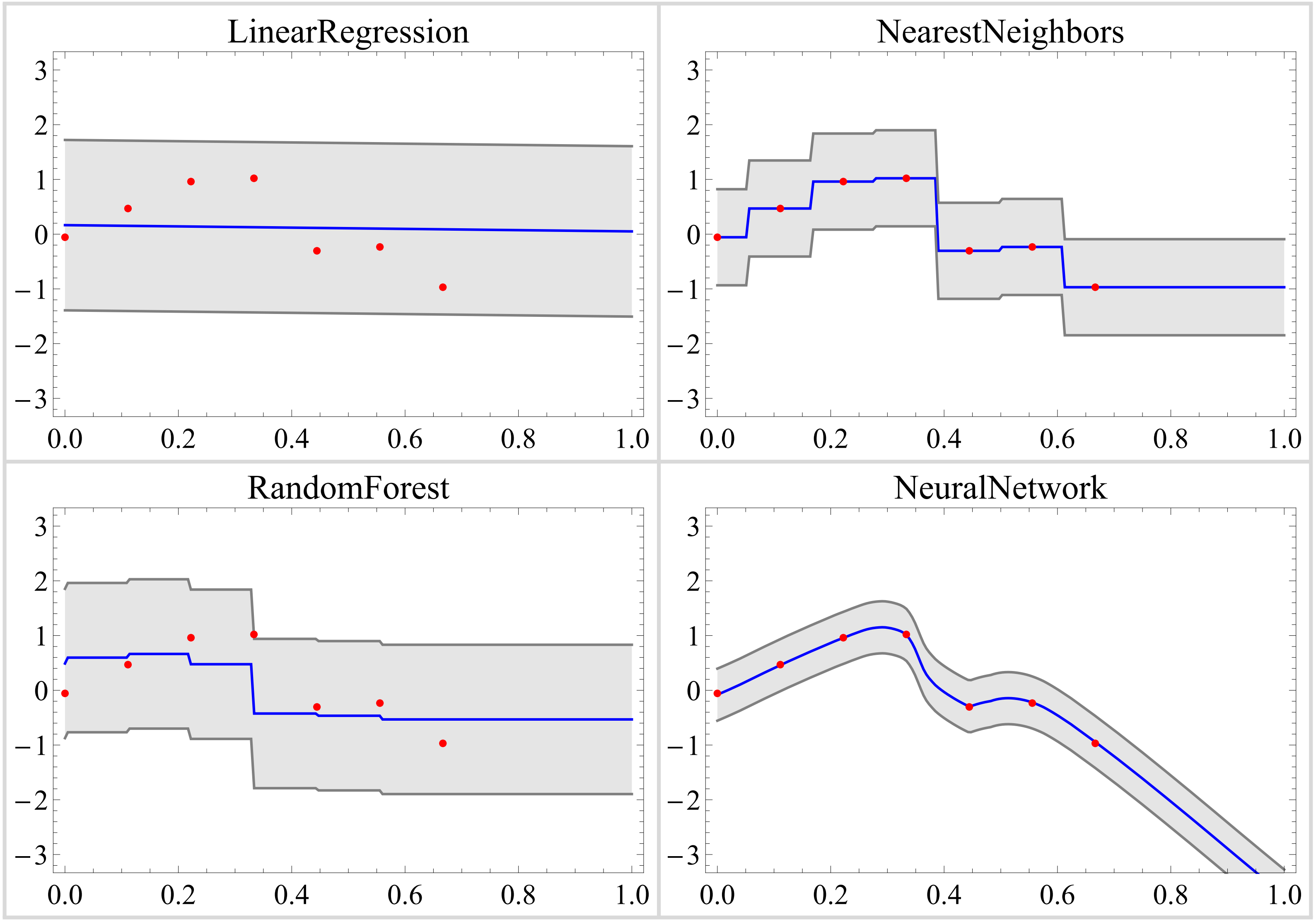


Visual comparison shows almost no difference!

Other predictive methods

Compare with other prediction methods

Out[∘]=



● Init

A GP prior $p(f \mid X)$ can be converted into a GP posterior $p(f \mid X, y)$ after having observed some data y. The posterior can then be used to make predictions f_* given new input X_*:

Out[•]//TraditionalForm=

$$\boxed{p(f_* \mid X_*, X, y) = \int p(f_* \mid X_*, f)\, p(f \mid X, y)\, df = \mathcal{N}(f_* \mid \mu_*, \Sigma_*)}$$

By definition of the GP, the joint distribution of observed data y and predictions f_* is

$$\begin{pmatrix} y \\ f_* \end{pmatrix} \approx \mathcal{N}\left(0, \begin{pmatrix} K_y & K_* \\ K_* & K_{**} \end{pmatrix}\right)$$

With N training data and N_* new input data, $K = \kappa(X, X) + \sigma_y{}^2 I = K + \sigma_y{}^2 I$ is $N \times N$, $K = \kappa(X, X_*)$ is $N \times N_*$ and $K_{**} = \kappa(X_*, X_*)$ is $N_* \times N_*$. $\sigma_y{}^2$ is the noise term in the diagonal of K_y.It is set to zero if training targets are noise-free and to a value greater than zero if observations are noisy.

Out[•]=