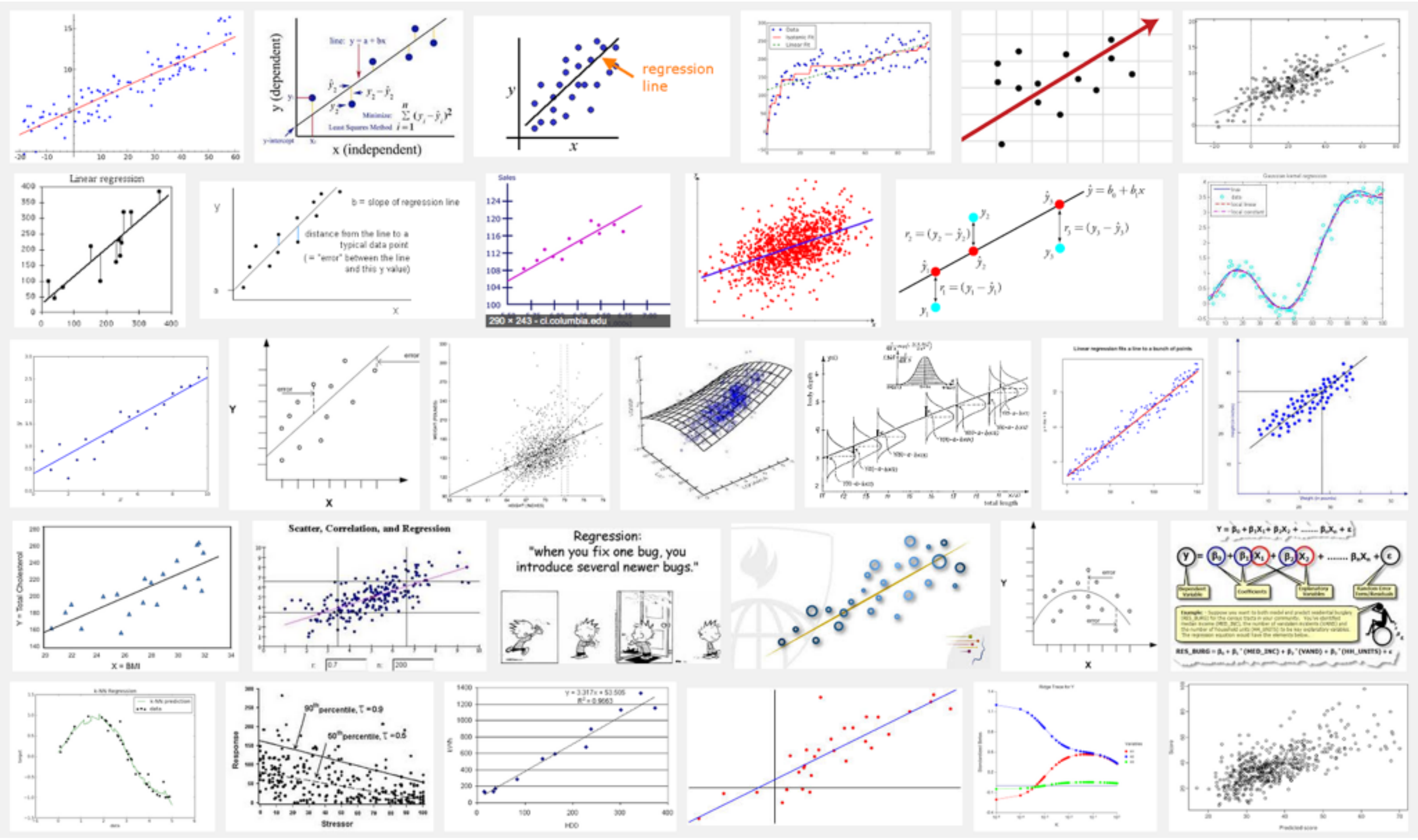
CAUP Programmer's club   24/2/2016

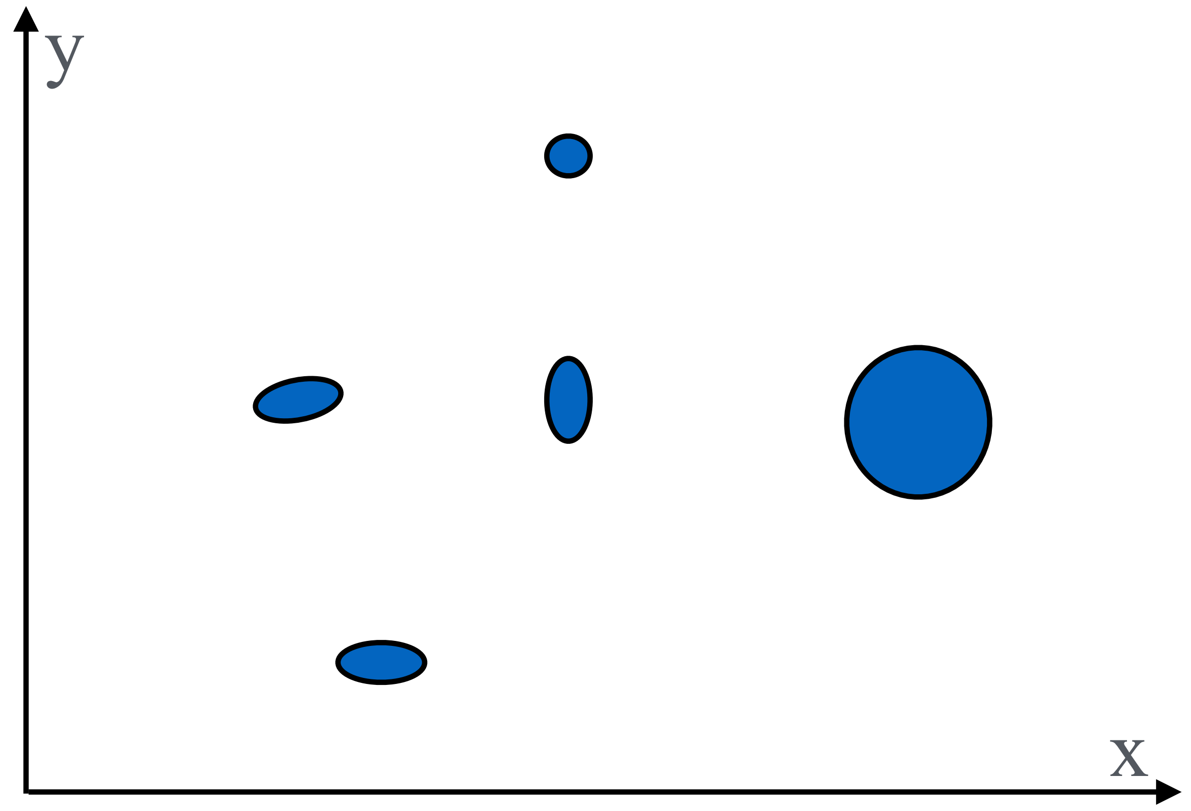
Regression

Jarle Brinchmann (CAUP/Leiden)

Regression - according to Google.



The overall idea



Ideally we would like to know the full likelihood

$$p(x, y)$$

Often too time-consuming, or does not help our physical understanding. In this case we focus on the expectation value of y

$$y = p(x; \vec{\theta})$$

where $\vec{\theta}$ is a set of parameters for a model. This is what regression techniques help us with.

More reading, not just on regression

Below you can find some books of use. The titles take you to the Amazon page and some are also freely available on the web, in which case I add links to those sites too.

- "Statistics, Data Mining, and Machine Learning in Astronomy" - Ivezic, Connolly, VanderPlas & Gray, is a good introduction with examples relevant to astronomy.
- "Pattern Classification" - Duda, Hart & Stork, is a classic in the field.
- "Pattern Recognition and Machine Learning" - Bishop, is a very good and comprehensive book. Personally I really like this one.
- "Bayesian Data Analysis" - Gelman, is often the first book you are pointed to if you ask questions about Bayesian analysis.
- "Information Theory, Inference and Learning Algorithms" - MacKay, is a very readable book on a lot of related topics. The book is also freely available on the web.
- "Introduction to Statistical Learning" - James et al is a readable introduction (fairly basic) to statistical technique of relevance. It is also freely available on the web.
- "Elements of Statistical Learning" - Hastie et al, is a slightly more advanced version of the Introduction to Statistical Learning with much the same authors. This is also freely available on the web.

Standard linear regression

In this case we typically have p observables at each of N points (predictors) and want to predict a response variable, y_i

A standard way to fit this is to minimise the residual sum of squares (RSS):

$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the estimate of y_i. In the simplest case this is

$$\hat{y}_i = a + b x_i$$

although it is common to generalise this:

Standard linear regression

In this case we typically have p observables at each of N points (predictors) and want to predict a response variable, y_i

A standard way to fit this is to minimise the residual sum of squares (RSS):

$$\mathrm{RSS} = \sum_i (y_i - \hat{y}_i)^2$$

where $\hat{y}_i$ is the estimate of y_i. Which for linear regression is:

$$\hat{y}_i = \theta_0 + \sum_{j=1}^{p} \theta_j x_{ij}$$

Common formulation - the design matrix

The problem to solve is then often written:

$$Y = \mathrm{M}\boldsymbol{\theta}$$

Where Y is $(y_1, y_2, \ldots, y_N)$ and $\boldsymbol{\theta}=(\theta_1, \theta_2, \ldots, \theta_p)$

M is known as the design matrix and is

$$\mathrm{M} = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{N,1} & x_{N,2} & \cdots & x_{N,p} \end{pmatrix}$$

Common formulation - the design matrix

$$Y = \mathsf{M}\boldsymbol{\theta}$$

If we also introduce the covariance matrix of uncertainties on Y:

$$\mathsf{C} = \begin{pmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sigma_N^2 \end{pmatrix}$$

the general solution of the linear regression is given by

$$\boldsymbol{\theta} = \left(\mathsf{M}^T \mathsf{C}^{-1} \mathsf{M}\right)^{-1} \left(\mathsf{M}^T \mathsf{C}^{-1} Y\right)$$

with uncertainties on the parameters given by

$$\Sigma_{\boldsymbol{\theta}} = \left(\mathsf{M}^T \mathsf{C}^{-1} \mathsf{M}\right)^{-1}$$

Regression in a programming context

Python:

A lot of the relevant code is in `sklearn.linear_model`

`statsmodels` offer an alternative, more similar to R.

And some of the data we will use is in `astroML`

So:

```
pip install sklearn / conda install sklearn
pip install astroML / conda install astroML
pip install statsmodels / conda install statsmodels [optional]
pip install pandas / conda install pandas
pip install seaborn / conda install seaborn [for plotting]
```

R:

Built-in fitting in `lm` and a wide range of related packages.

Linear regression - trying it out

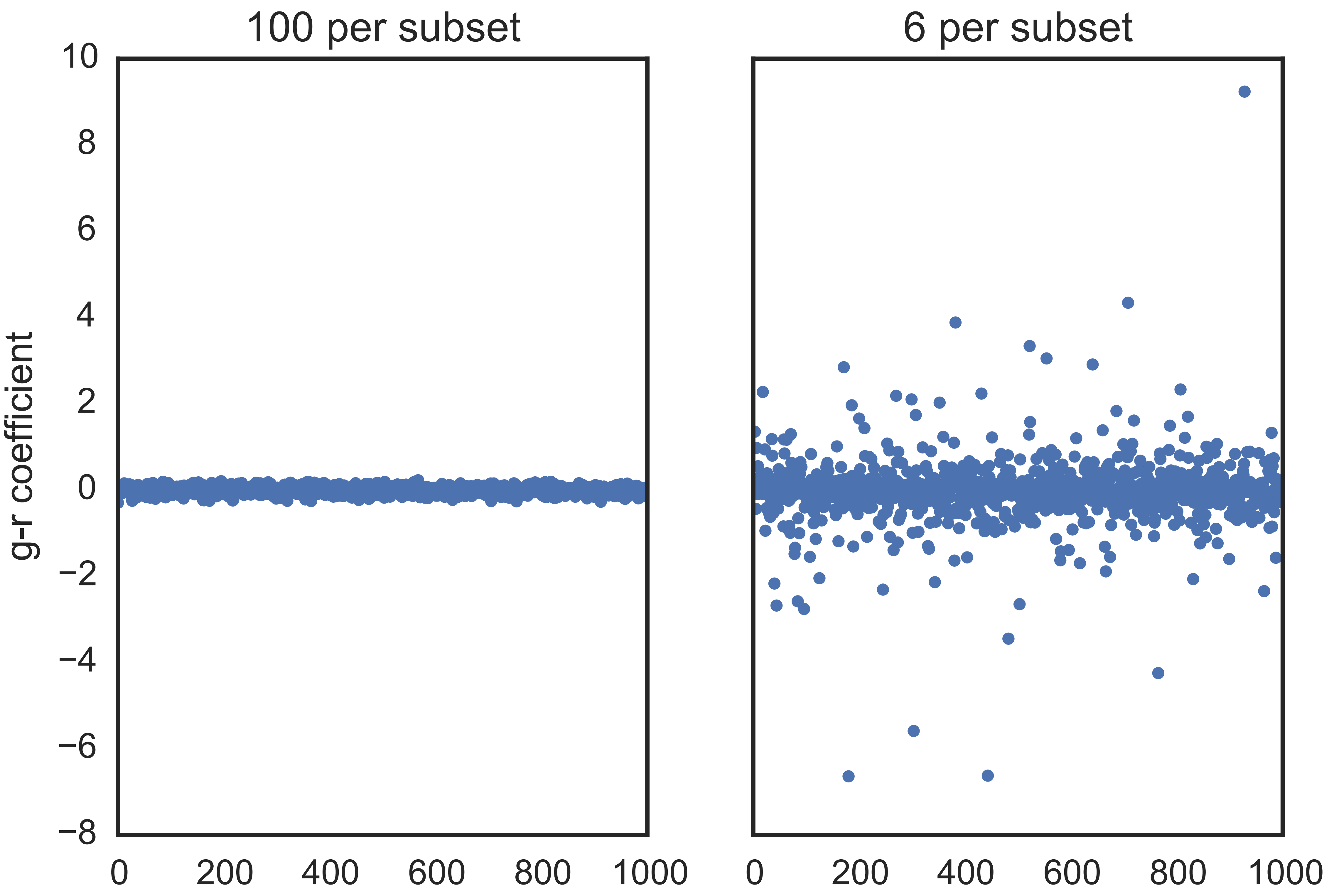
```
M, T = pickle_from_file('T-vs-colour-regression.pkl')
```

```
from astroML.linear_model import LinearRegression

model = LinearRegression(fit_intercept=True)
result = model.fit(M, T/1e4)
Tpred = model.predict(M)

result.coef_   # Coefficients of the fit.
```

Note that the *intercept* is the first element of the coefficient array. So if M is N_{obj}x4, `coef_` will be **5** elements long.

Regularising linear regression - ridge regression

When N is comparable to p, linear regression typically has large variance. To combat this we might want to trade a bit of bias for a lower variance.



Fits of 1000 random subsets with a linear regressor and showing just one coefficient.

Regularising linear regression - ridge regression

When N is comparable to p, linear regression typically has large variance. To combat this we might want to trade a bit of bias for a lower variance.

We do this by **regularising** the solution and minimise:

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

Here we limit the *size* of the parameter vector ***β***.

This does introduce a regularisation parameter: λ

We determine this using CV, BIC/AIC, or other similar techniques.

Those sneaky words: CV, BIC, AIC

CV: Cross-validation

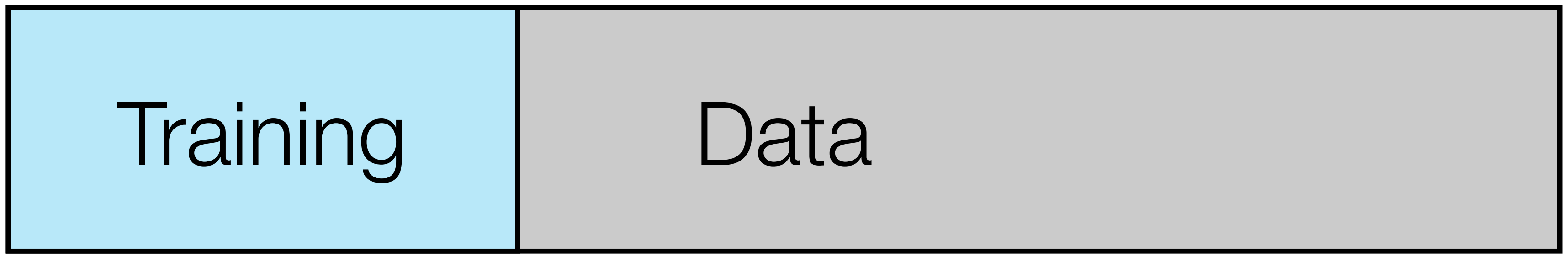
BIC: Bayesian Information Criterion

AIC: Aikaike Information Criterion

Cross-validation

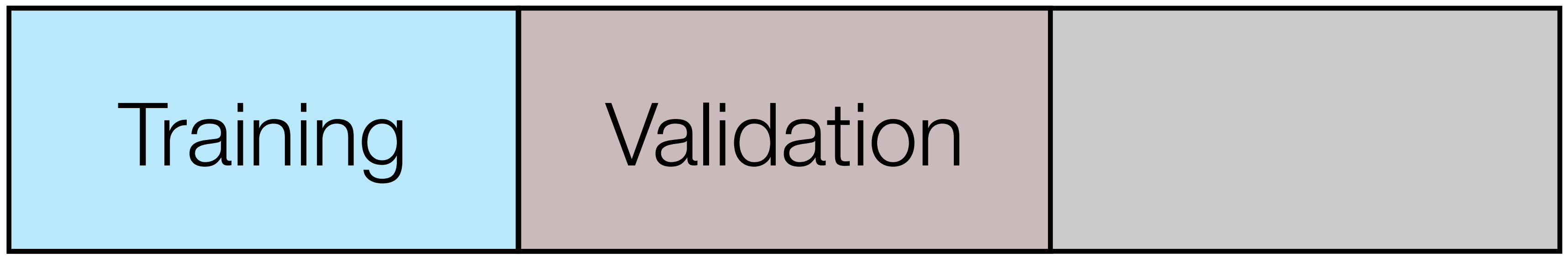
One possibility

Cross-validation



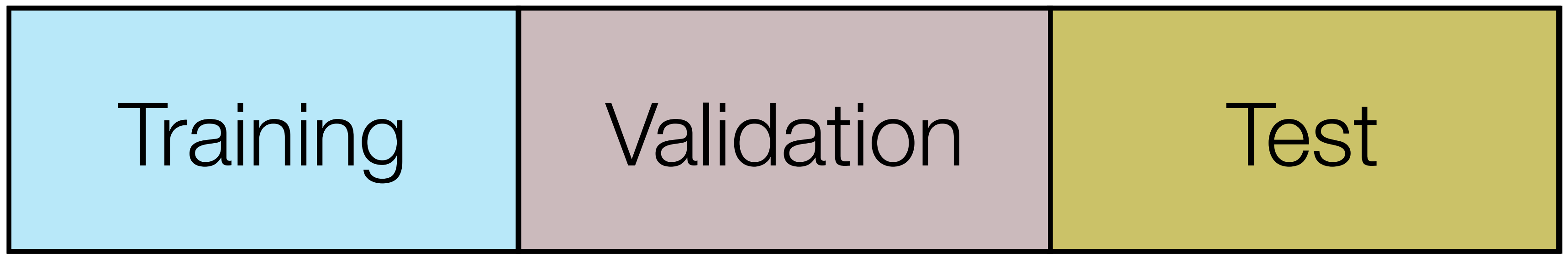
One possibility

Cross-validation



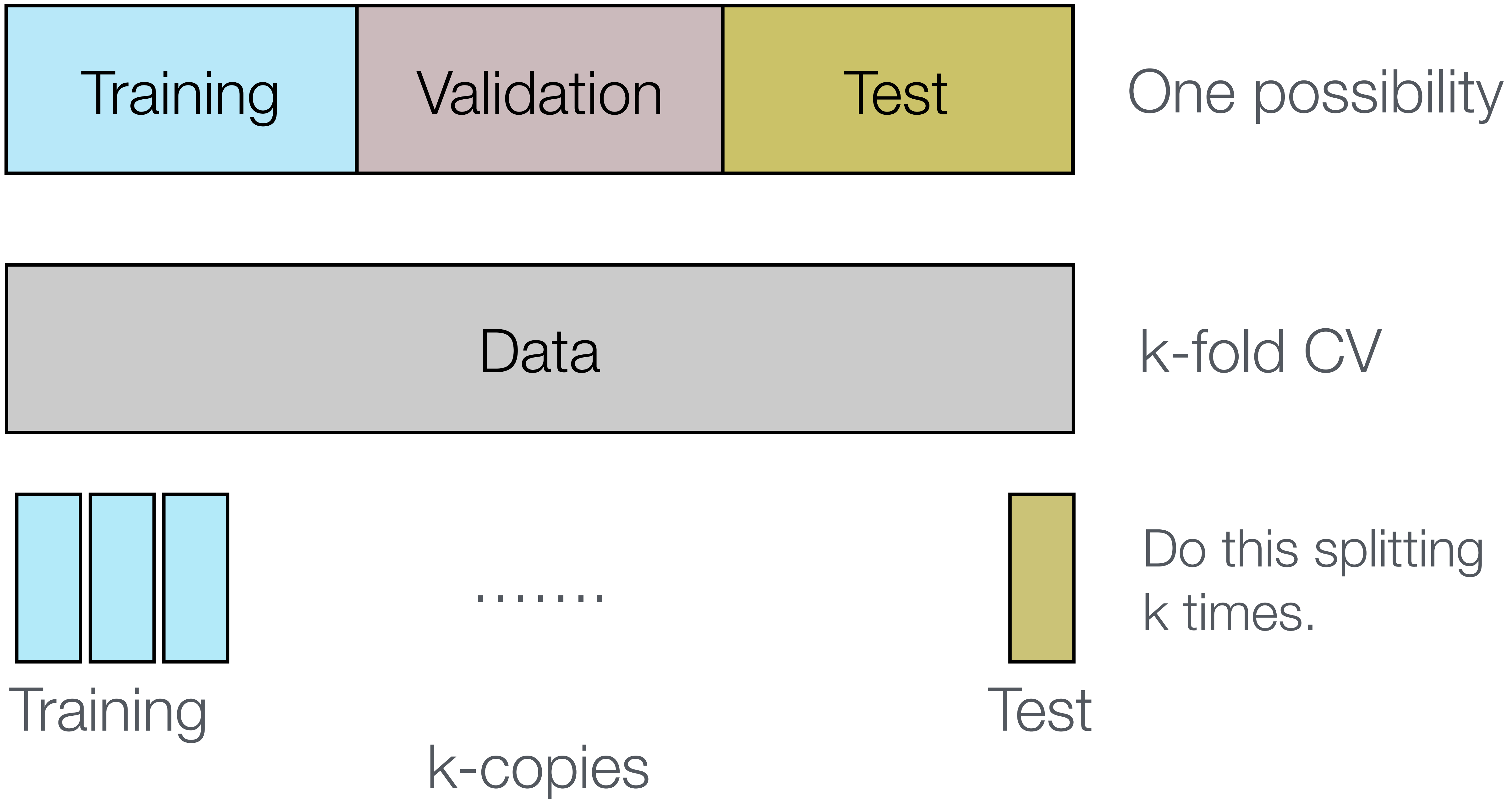
One possibility

Cross-validation



One possibility

How do we choose the bandwidth?



It is generally recommended to use 5- or 10- fold cross-validation.

How do we then test?

Running the linear fit on your training sample with one λ gives a function

$$\hat{y}_\lambda(x)$$

Evaluate this on your test sample & calculate the residuals - sum these in quadrature (for instance), to calculate the 'loss' or 'fitness'

$$\mathrm{RSS}_\lambda$$

Do this for many λ and choose the one giving the smallest RSS.

Ridge regression - how to

```
from sklearn.linear_model import Ridge
model = Ridge(alpha=0.05, normalize=True)

result = model.fit(M, T/1e4)
Tpred = model.predict(M)

res.coef_   # Coefficients of the fit.
```

Note: alpha = λ in my (and others') notation.

Very similar to LinearRegression - with one exception:

The normalize keyword.

Ridge regression - normalisation

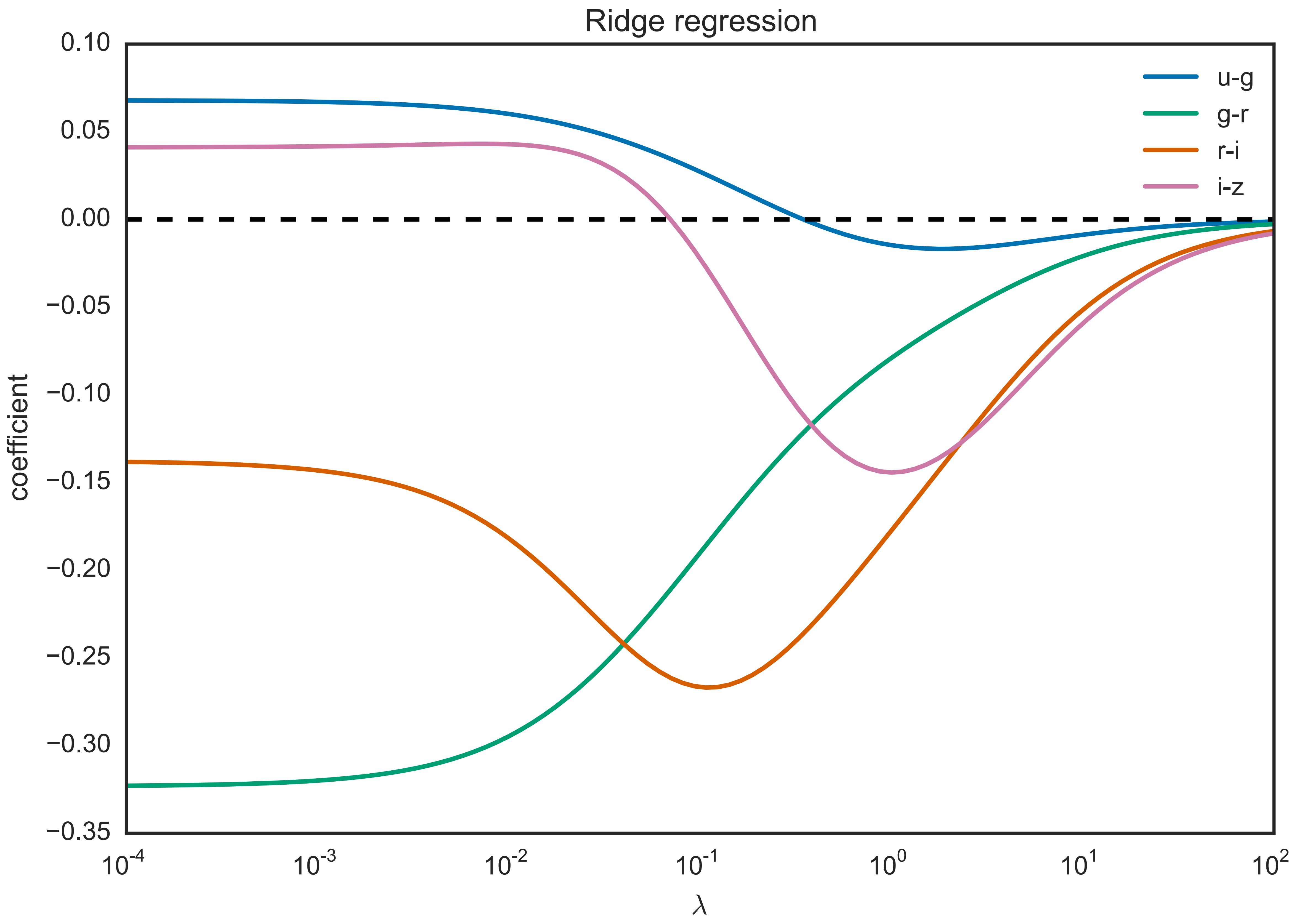
Ridge regression can also be seen to want to minimise

$$\sum_{i=1}^{N} \left(y_i - \theta_0 - \sum_{j=1}^{p} \theta_j x_{ij} \right)^2$$

subject to

$$\sum_{j=1}^{p} \theta_j^2 \leq s$$

Obviously that length will depend on the units of x_{ij}. It is therefore common to “whiten” or standardize x by dividing by its standard deviation (ideally robustly).


Ridge regression
coefficient
λ
u-g
g-r
r-i
i-z
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}

Ridge regression - degrees of freedom

There are of course p parameters, but because of the constraints in ridge regression, the effective number of degrees of freedom is not p - rather it is a smaller number depending on λ.

As far as I know this is not available through sklearn, but you can calculate it from the SVD of X:

$$\mathsf{X} = \mathsf{UDV}^T$$

If the diagonal entries in D are d_j, the d.o.f. is:

$$\mathrm{df} = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda}$$

Regularising linear regression - LASSO

Ridge regression minimises the l_2 norm of the coefficients. The Lasso (Least Absolute Shrinkage and Selection Operator) minimises the l_1 norm.

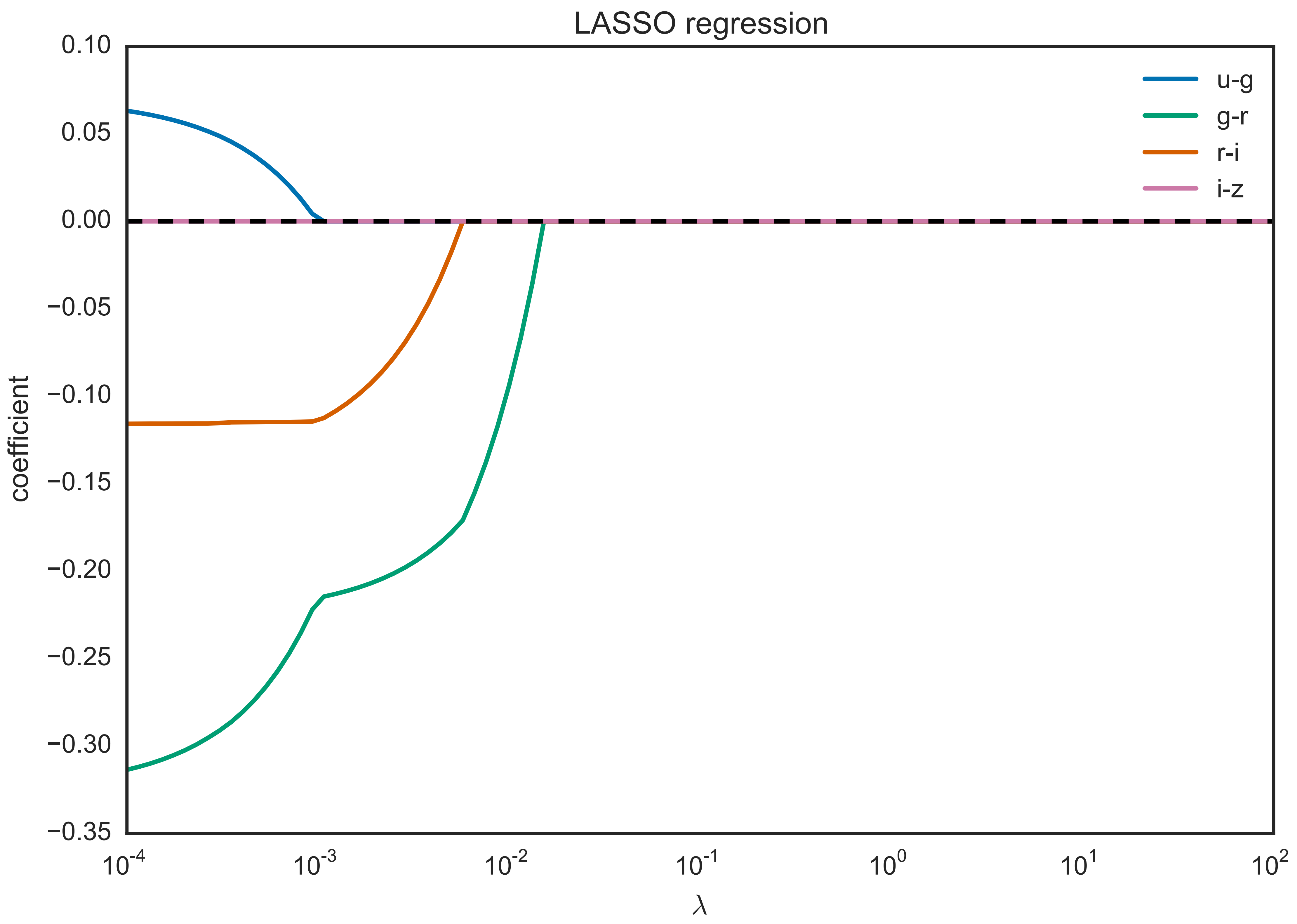
The minimisation is now of

$$\mathrm{RSS} + \lambda \sum_{j=1}^{p} |\theta_j|$$

Lasso regression - how to

```
from sklearn.linear_model import Lasso
model = Lasso(alpha=alpha, normalize=True)

result = model.fit(M, T/1e4)
Tpred = model.predict(M)

res.coef_   # Coefficients of the fit.
```

So just like ridge regression - but the result is somewhat different


LASSO regression
coefficient
λ
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−0.35
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
u-g
g-r
r-i
i-z

Variable selection with the Lasso

So some coefficients end up being set to zero!

This fact means that the lasso performs ***variable selection***.

This feature of the lasso can be phrased to say
that it returns **sparse models**.

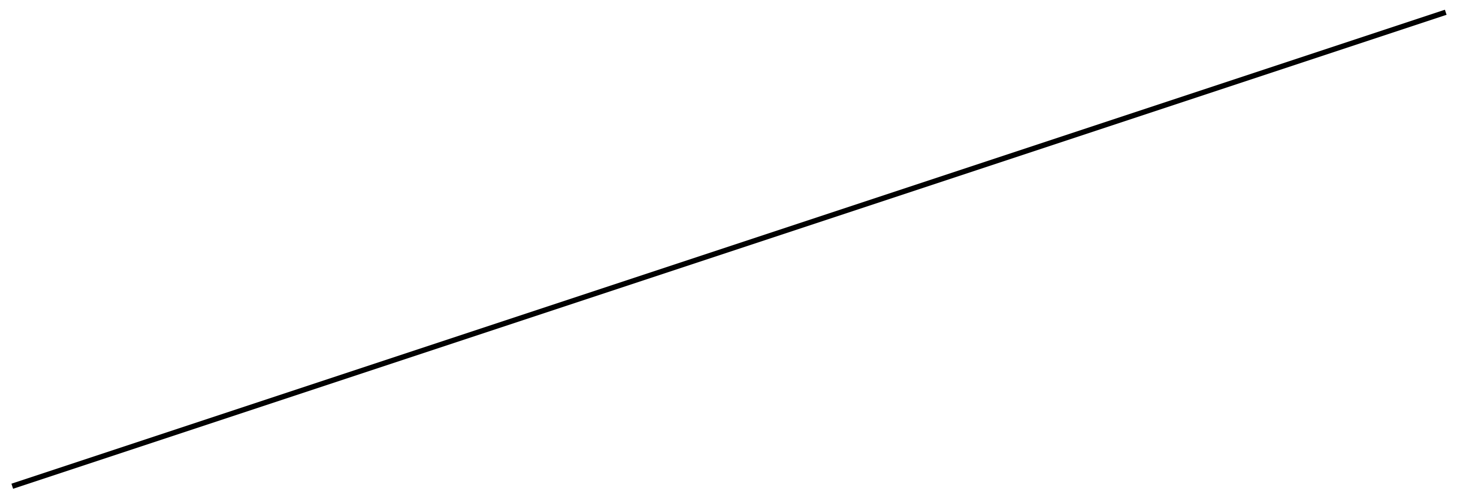
Basically it can be interpreted to say which variables
(predictors) are most important for predicting the output.

Oh - the CV part

```
from sklearn.linear_model import RidgeCV
model = RidgeCV(alpha=[0.01, 0.05, 0.1]
                normalize=True)

result = model.fit(M, T/1e4)
Tpred = model.predict(M)

res.coef_   # Coefficients of the fit.
res.alpha_  # The alpha used
```

I would recommend doing it manually a few times to understand what is happening though.

The dullness of straight lines



The dullness of straight lines

Can’t we be more exciting?

Yes we can!

1. Change the basis functions - still linear but powerful. This is the class of polynomial regression but go much further and allows us to use **many** basis functions (see example)

$$y_i = \theta_0 + \sum_{j=1}^{N} \theta_j \phi_j(x_i)$$

2. Go non-linear

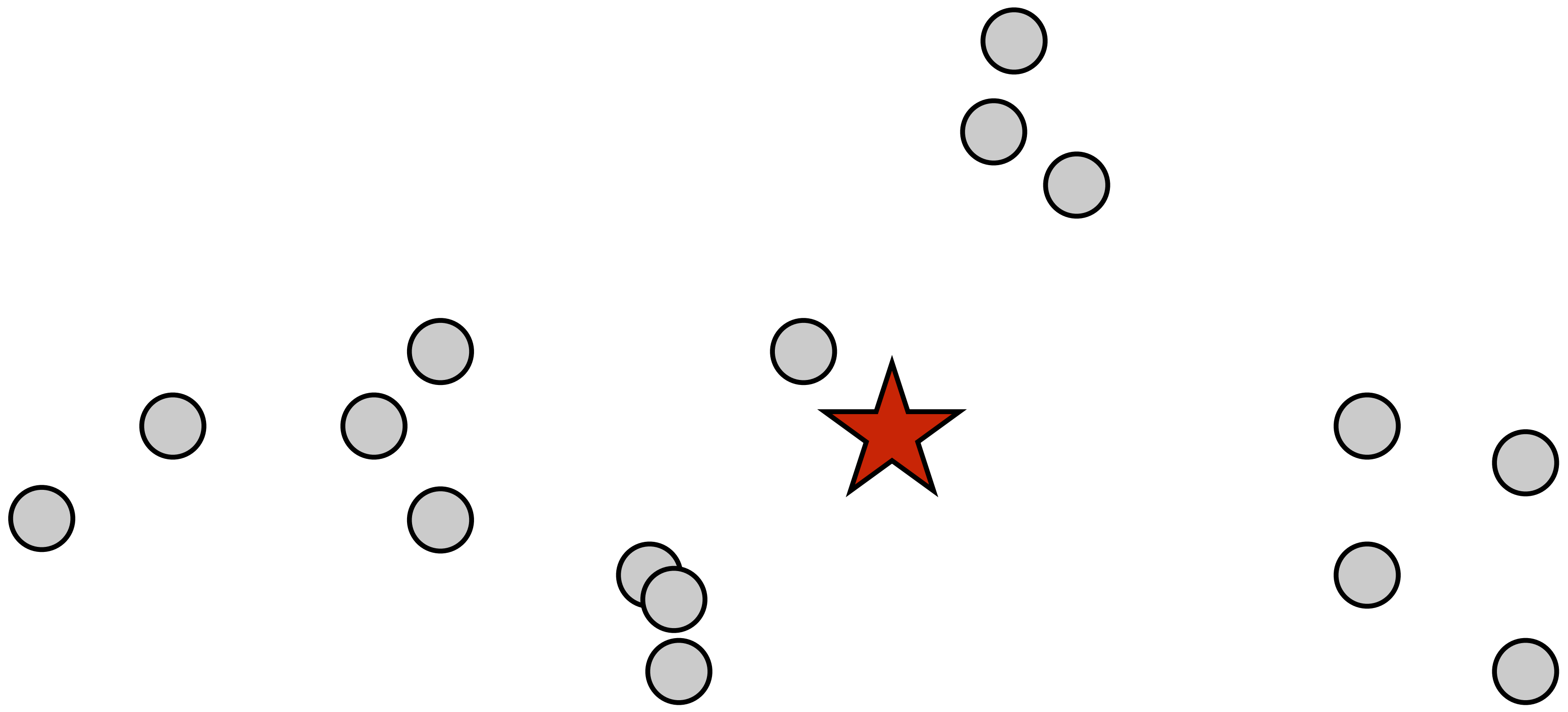
3. Do local regression fits

The main techniques

- Nearest neighbour regression
  - Take the mean of the k nearest points
- Kernel regression
  - Calculate the weighted mean of training points
- Locally linear regression
  - Calculate a weighted linear regression at each point
- Gaussian process regression
  - Drop fixed functions and try to fit in the space of “all” functions

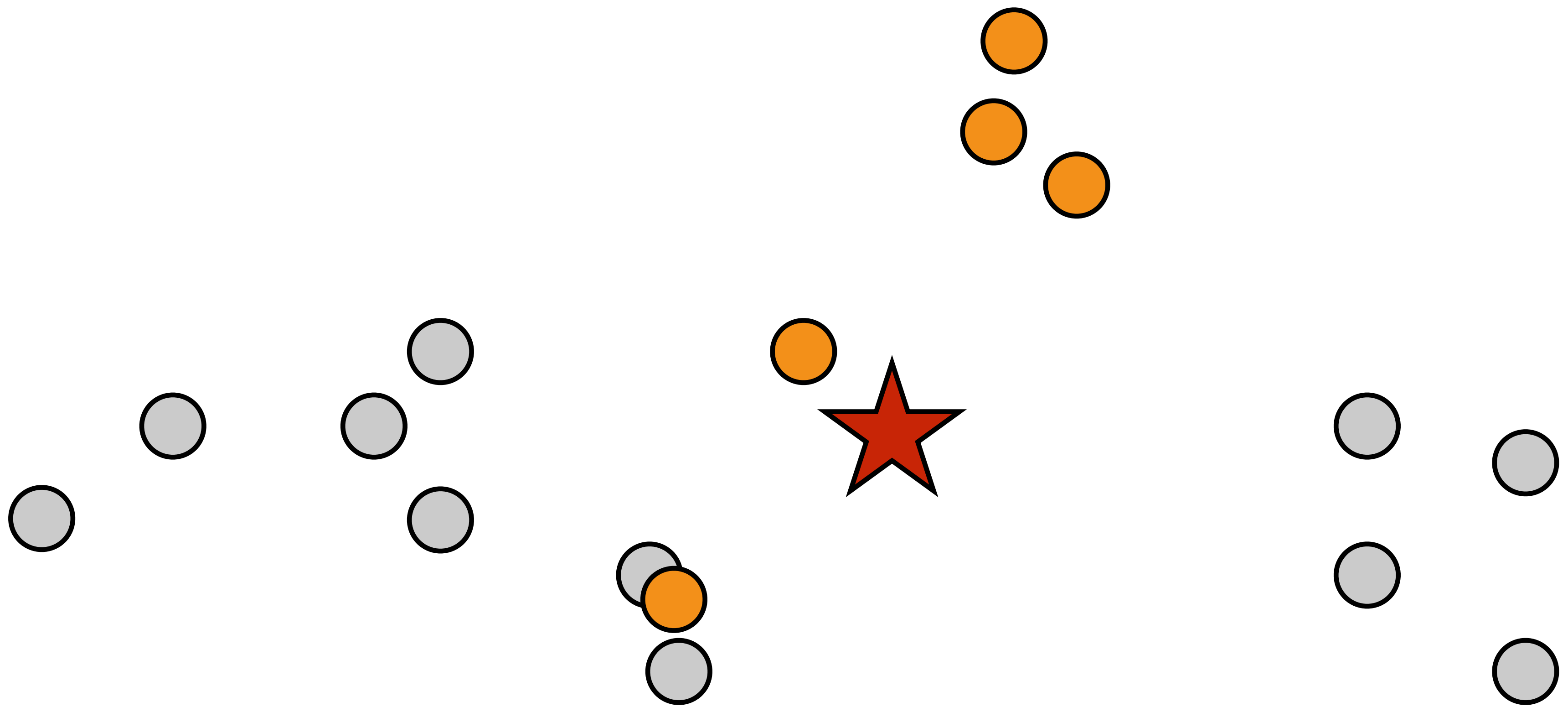
Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$



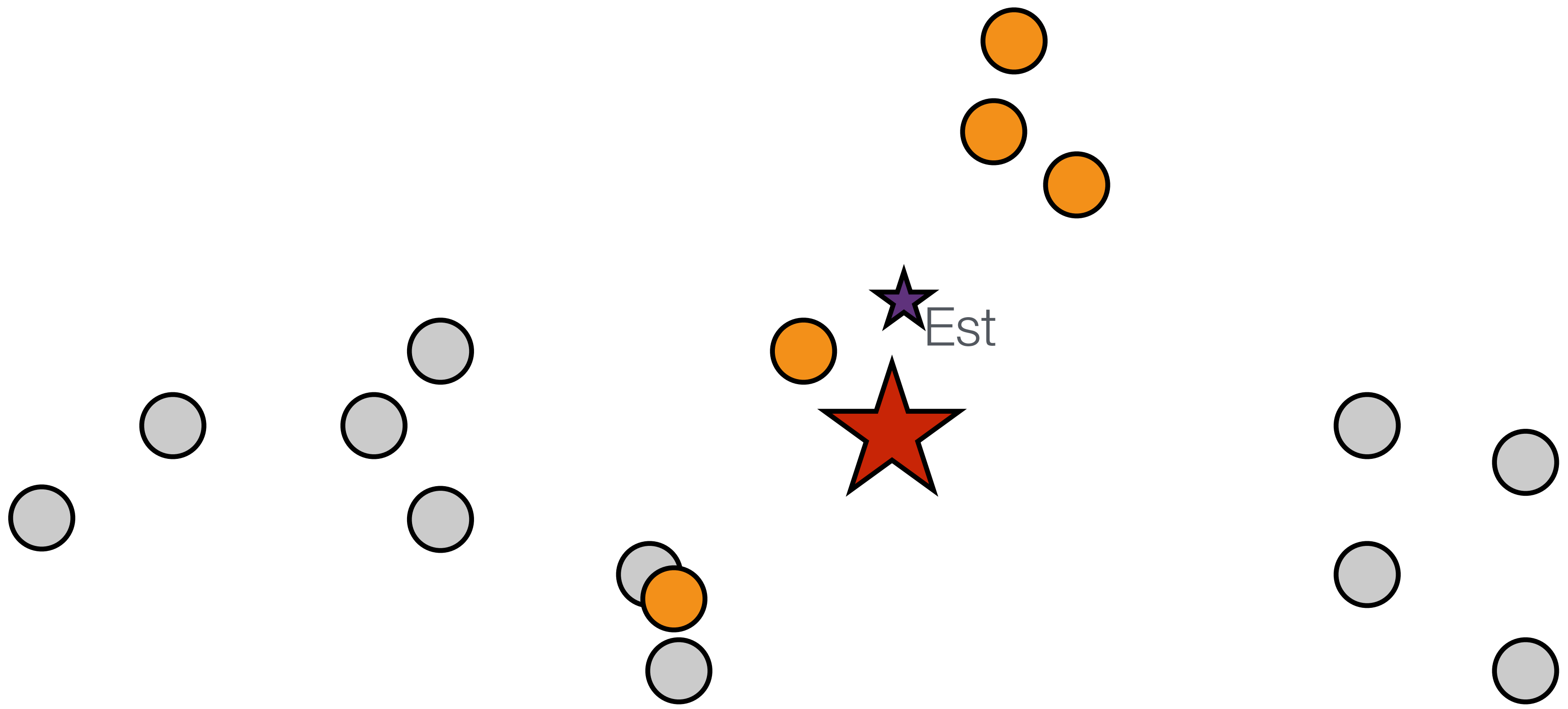
Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$



Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \mathrm{Neighbours(x;\ k)}} y_j$$

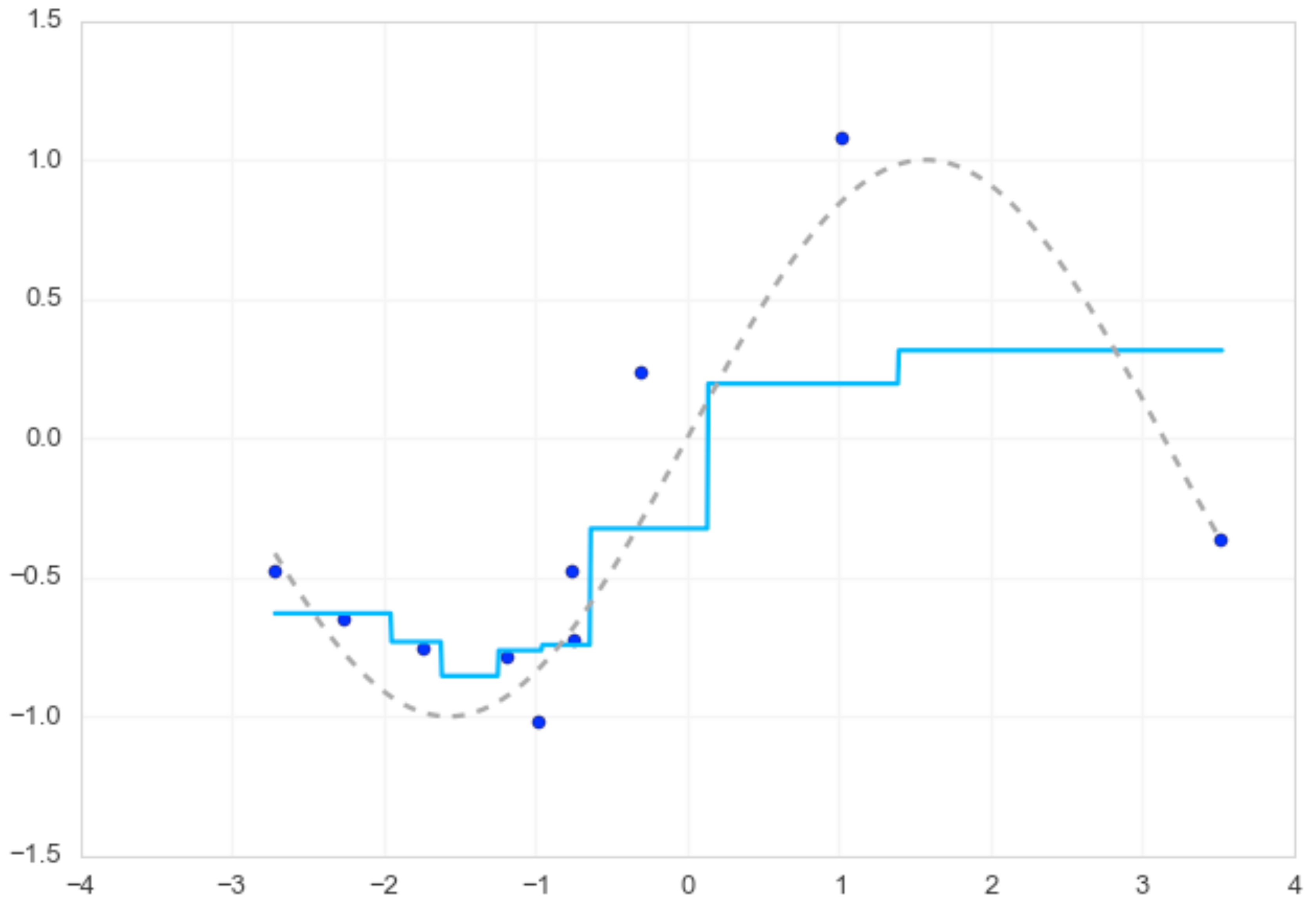


Nearest neighbour regression

$$\hat{y}(x) = \frac{1}{k} \sum_{x_j \in \text{Neighbours(x; k)}} y_j$$

```
from sklearn import neighbors

k = 3
knn = neighbors.KNeighborsRegressor(k)
y_est = knn.fit(X, y).predict(Xplot)
```

Nearest neighbour regression



```
from sklearn import neighbors

k = 3
knn = neighbors.KNeighborsRegressor(k)
y_est = knn.fit(X, y).predict(Xplot)
```

Kernel regression

In knn regression we give equal weight to each point. If instead we give a variable weight we get kernel regression

$$\hat{y}(x) = \sum_{i=1}^{N} K_h\left(x, x_i\right) y_i$$

It is actually not necessary that the x_i are at the same place as y_i, but I will assume that they are. (if they are not you have to be careful with the normalisation of the basis functions)

h is a complexity parameter so needs to be determined by AIC/BIC or cross-validation for instance.

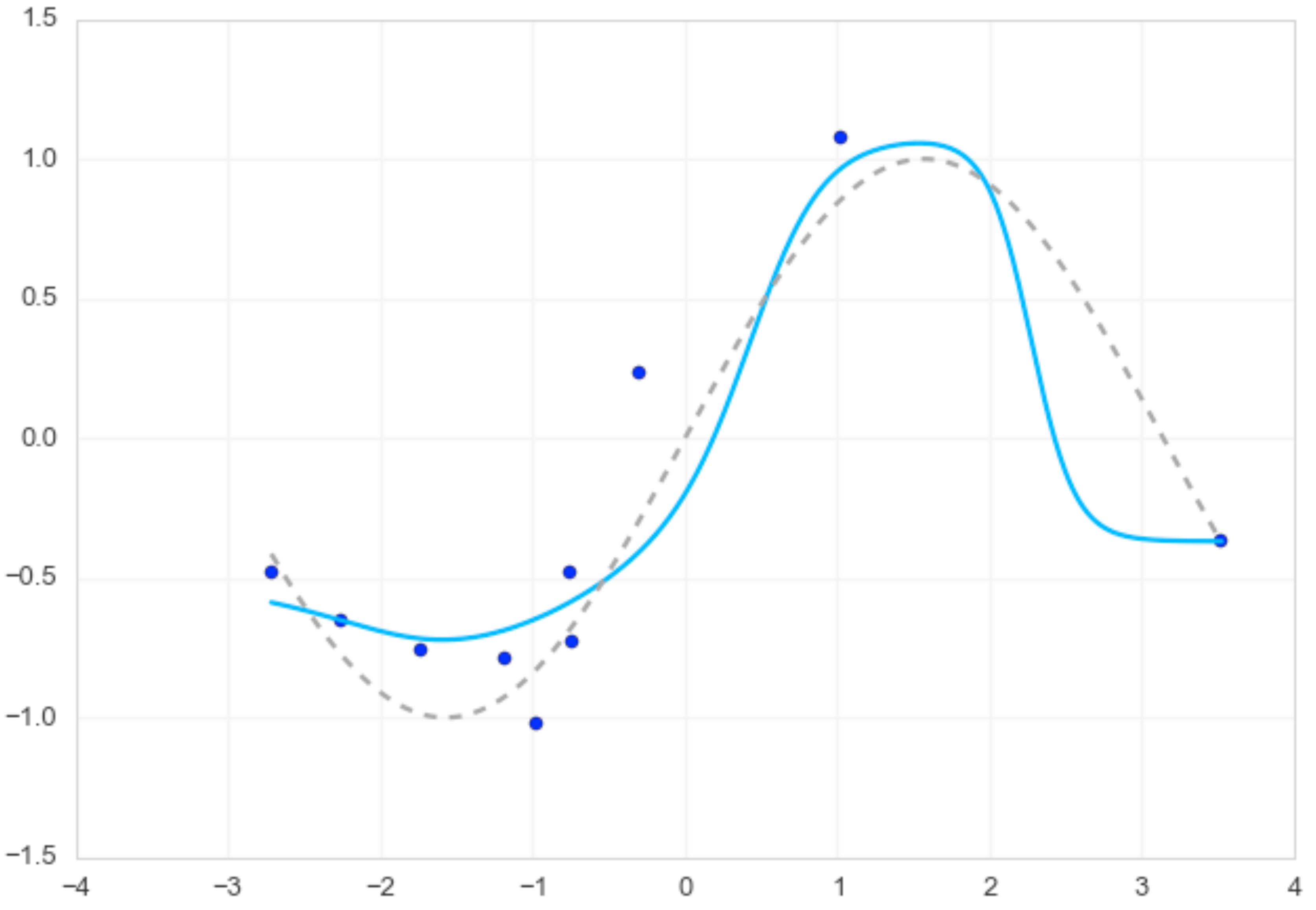
Kernel regression

The most common formulation of kernel regression renormalises the kernel functions to give the Nadaraya-Watson method:

$$\hat{y}(x) = \frac{\sum_{i=1}^{N} K_h\left(x, x_i\right) y_i}{\sum_{i=1}^{N} K_h\left(x, x_i\right)}$$

```python
from astroML.linear_model import NadarayaWatson

model = NadarayaWatson('gaussian', np.asarray(0.6))
model.fit(X, y)
y_est = model.predict(Xplot)
```

Kernel regression



```
from astroML.linear_model import NadarayaWatson

model = NadarayaWatson('gaussian', np.asarray(0.6))
model.fit(X, y)
y_est = model.predict(Xplot)
```

Locally linear regression

In knn and kernel regression we effectively work with the zeroth level Taylor expansion - the constant term. The next step is to fit a weighted linear regression:

$$\theta_0(x), \theta_1(x) = \underset{\theta_0, \theta_1}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1(x - x_i))^2 \, K_h(x, x_i)$$

This turns out to be very useful in many situations and is often used as a powerful smoother under the name loess/lowess and a powerful package `locfit` is available in R (see rpy2)

h is a complexity parameter so needs to be determined by AIC/BIC or cross-validation for instance.

Locally linear regression

The weight/kernel is usually taken to be the tri-cubic function:

$$w_i = (1 - |t|^3)^3 \; I(|t| \leq 1)$$

with $t = (x - x_i)/h$

Python packages:

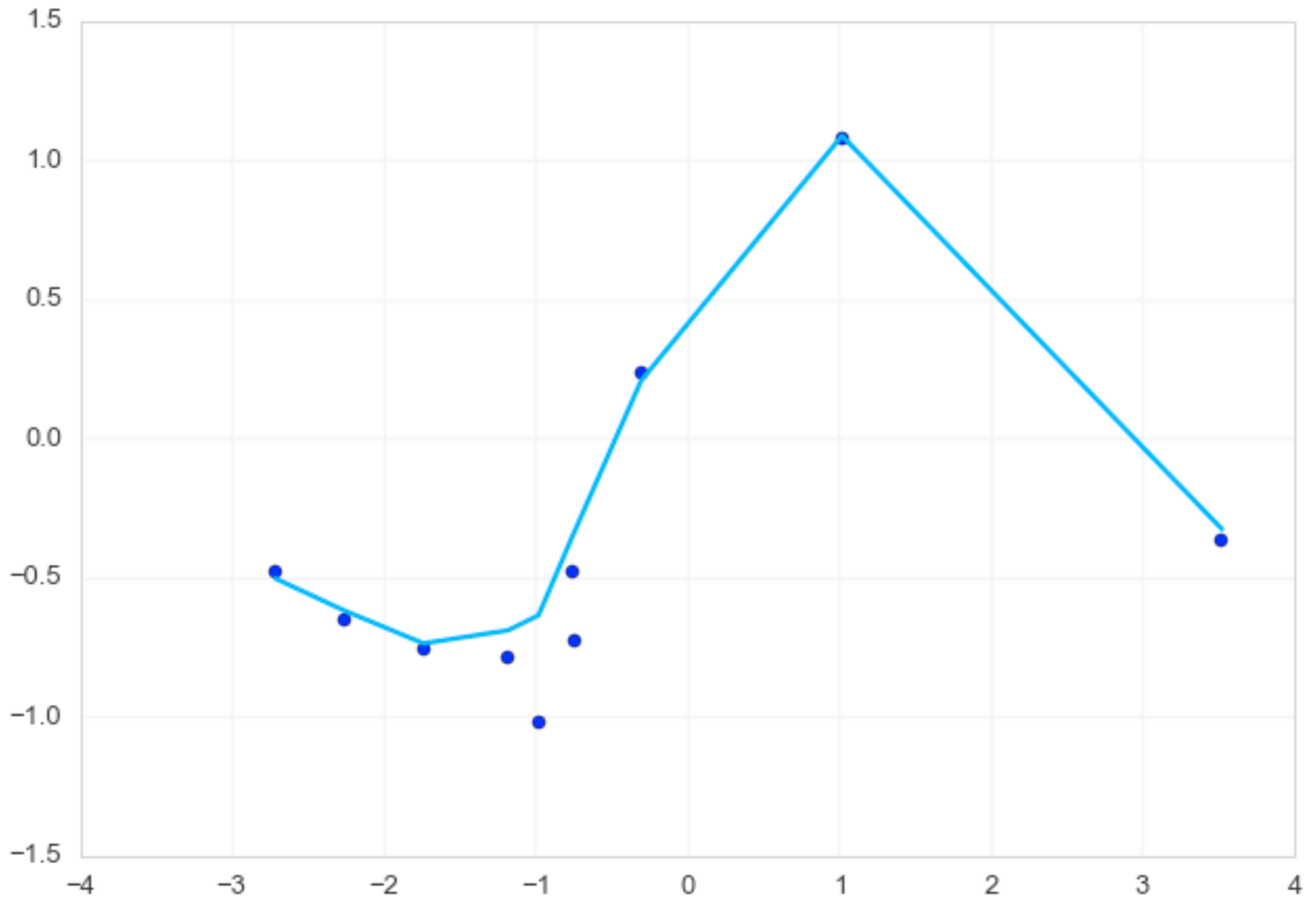
```
statsmodels.nonparametric.smoothers_lowess.lowess
cylowess
```

This is an area where R is better, but cylowess is decent.

Locally linear regression

```
import cylowess
c_lowess = cylowess.lowess

res_c = c_lowess(y,x)
plt.plot(res_c[:, 0], res_c[:, 1])
```

Note the order!!



Gaussian process regression

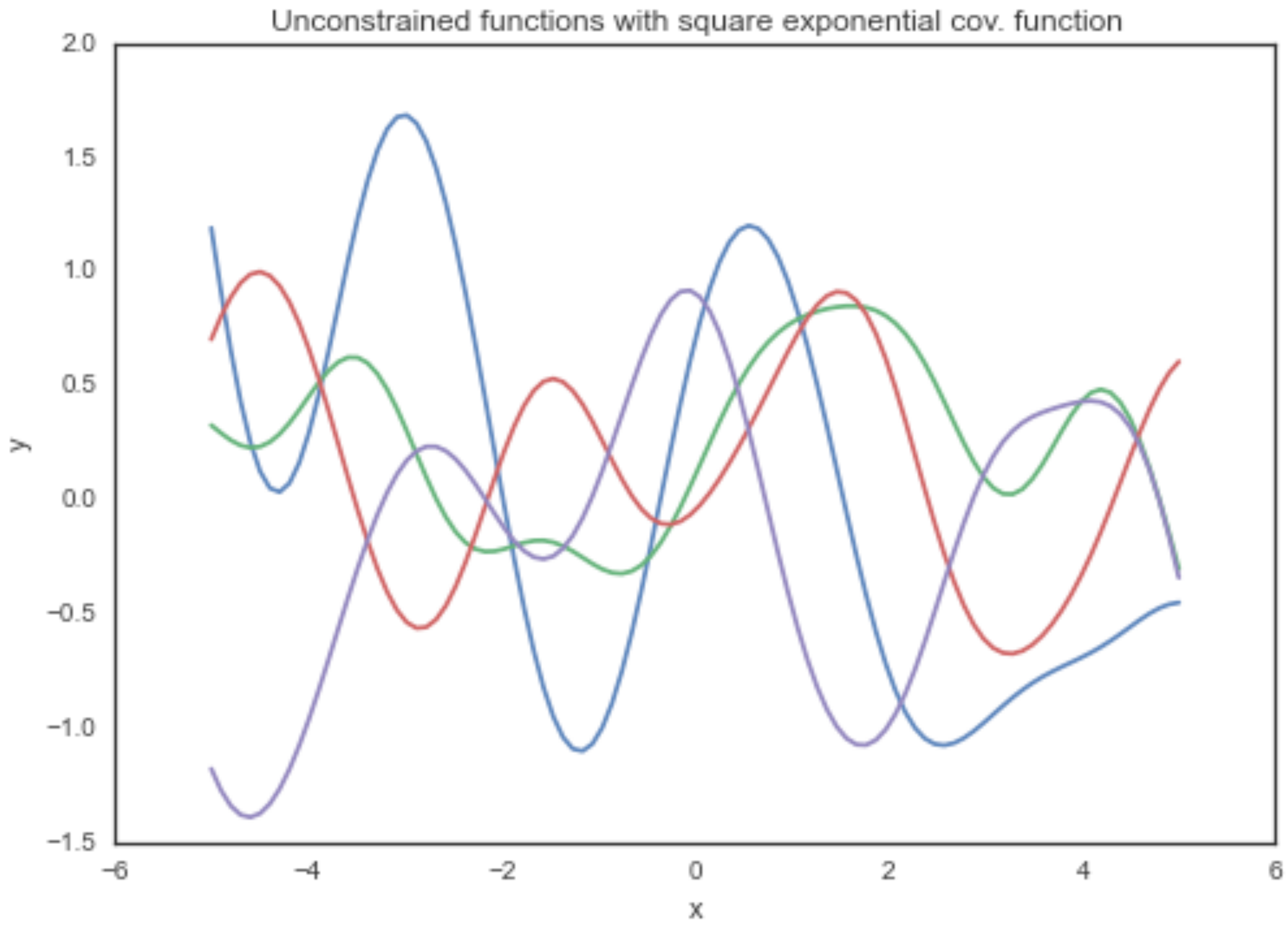
In this case we apply a prior in function space - this prior is specified using a mean & covariance function (since that is all we need for a Gaussian). The most common is:

$$\mathrm{Cov}\left(x, x'\right) = K(x, x') = \exp\left(-\frac{|x - x'|}{2h}\right)$$

If we set the mean to zero, we can then draw random functions

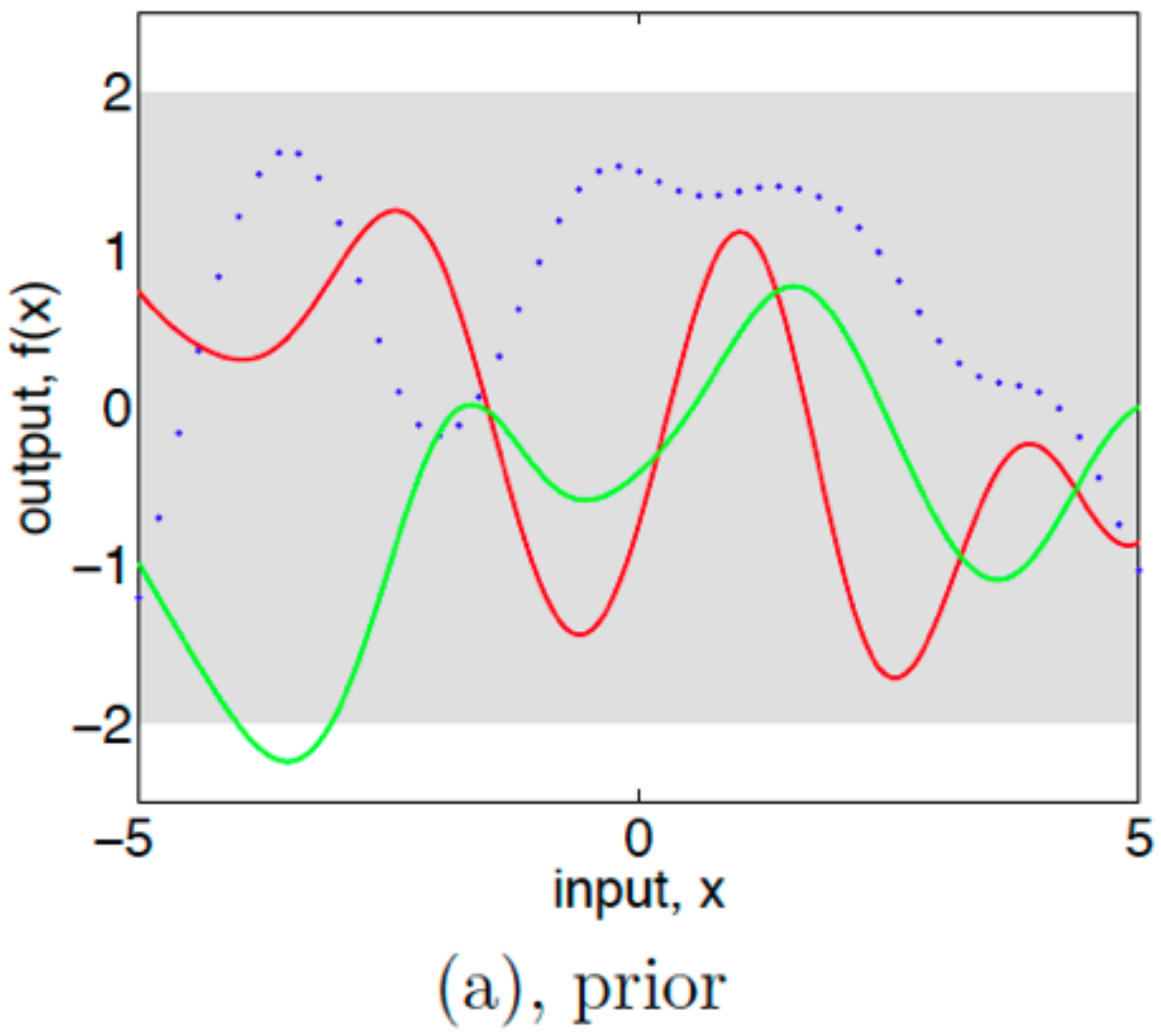
Gaussian process regression

Random functions - h=1

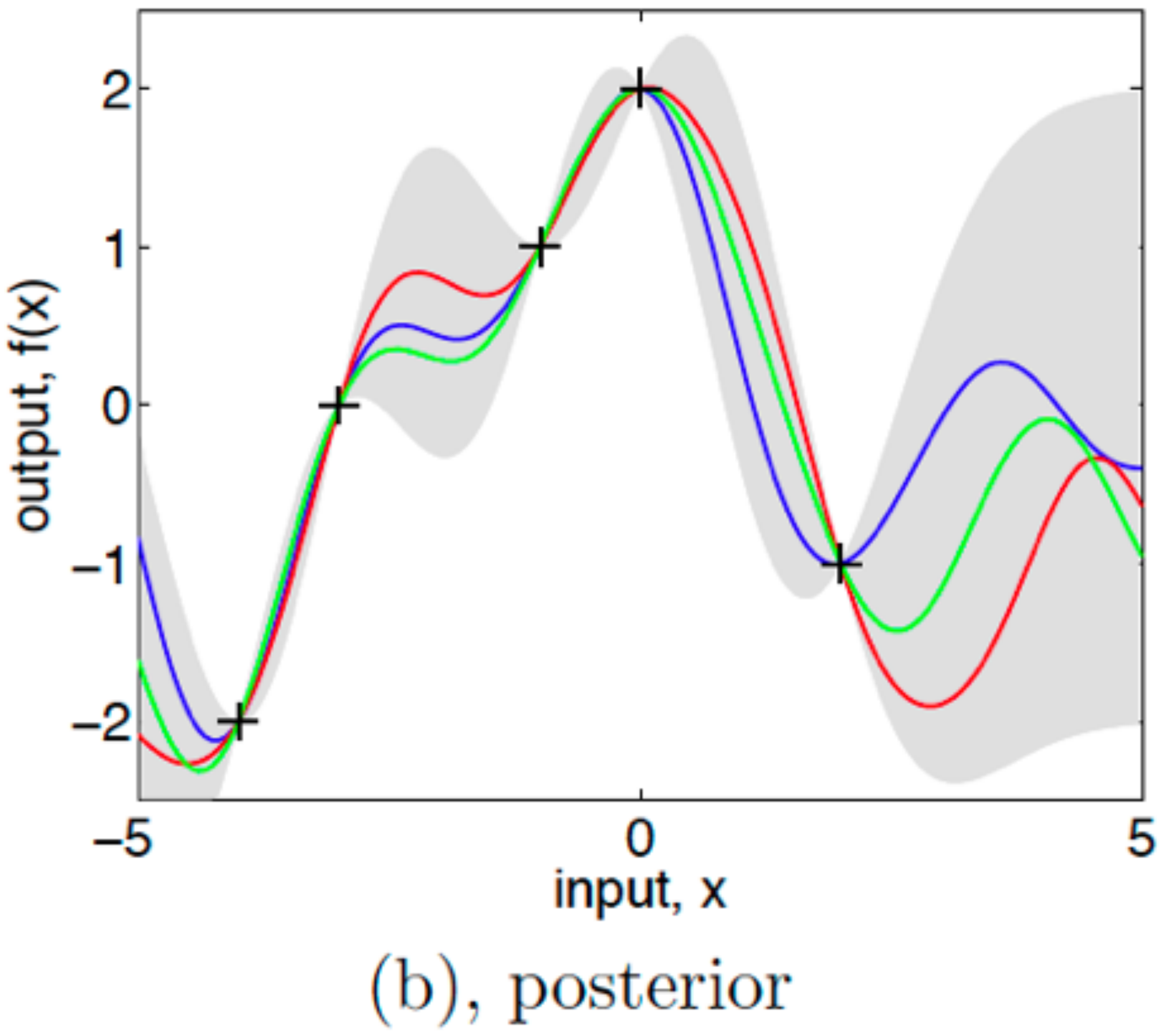


Gaussian process regression

We apply constraints by multiplying the prior with the likelihood:



(a), prior



(b), posterior

Taken from Rasmussen & Williams, “Gaussian processes for Machine Learning”, 2006, Figure 2.2
http://www.gaussianprocess.org/

Gaussian Process Regression - features

+ Very flexible
+ Provides covariance estimates on predictions

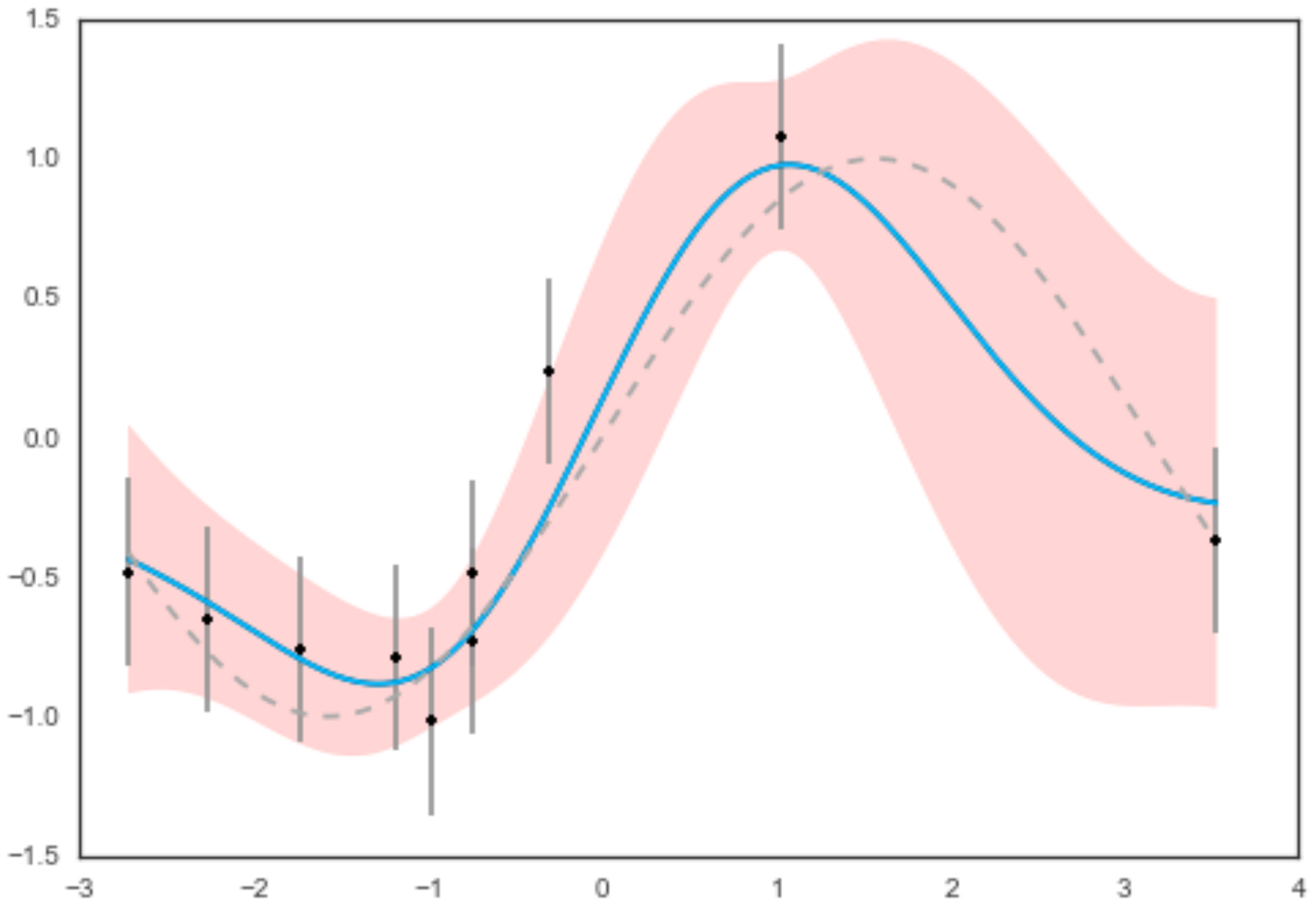
- Fairly slow

Python usage:

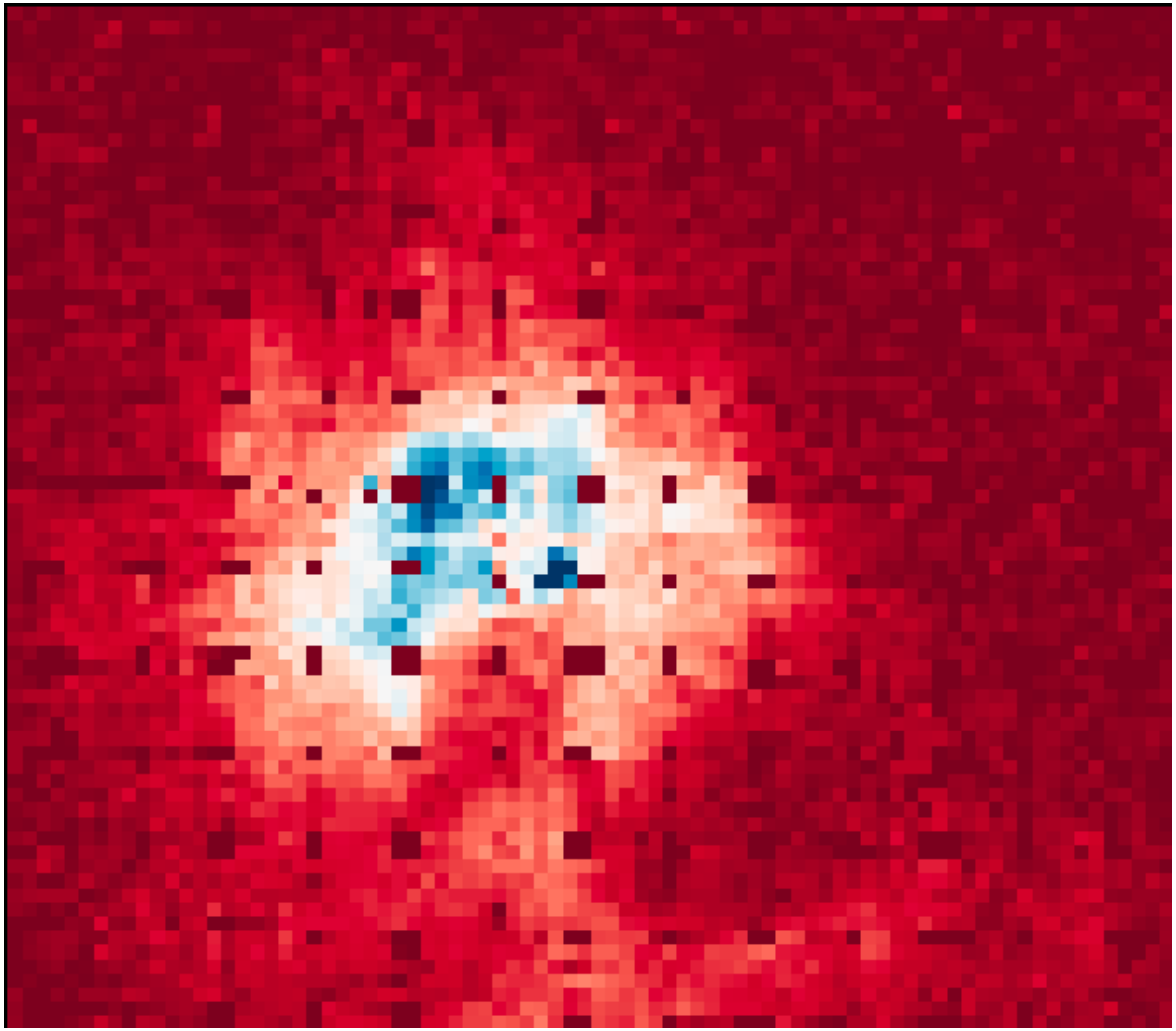
```
from sklearn.gaussian_process import GaussianProcess

gp = GaussianProcess(corr='squared_exponential',
                     theta0=0.1, thetaL=1e-2,
                     thetaU=1, normalize=False,
                     nugget=(dy/y)**2, random_start=1)
```

Example use:

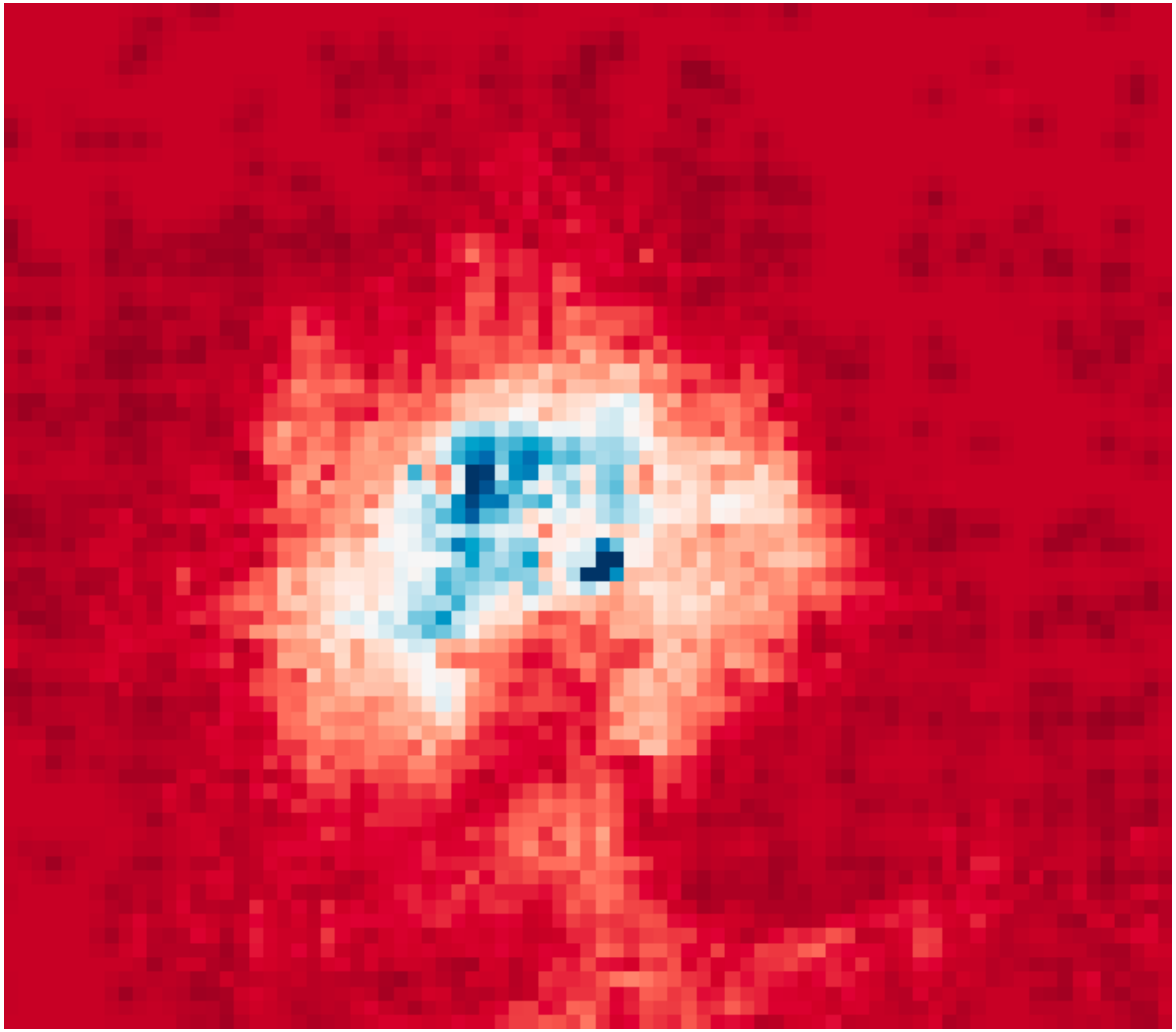
```
g = gp.fit(x[:, np.newaxis], y)
y_pred, MSE = gp.predict(xplot[:, np.newaxis], eval_MSE=True)
sigma = np.sqrt(MSE)
plot_a_fit(x, y, dy, xplot, y_pred, sigma, include_true=True)
```



Example for an image



Example for an image



Taking a step - back, the summary

Taking a step - back, the summary

I have a good idea where my data come from

Taking a step - back, the summary

I have a good idea where my data come from

non-linear fits (e.g. lmfit from last Programmers Club)
Bayesian models

Taking a step - back, the summary

I have a good idea where my data come from

non-linear fits (e.g. lmfit from last Programmers Club)
Bayesian models

My data seems to have a lot of collinearity & I have a lot of data.

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?

Taking a step - back, the summary

I have a good idea where my data come from

non-linear fits (e.g. lmfit from last Programmers Club)
Bayesian models

My data seems to have a lot of collinearity & I have a lot of data.

regular linear regression, basis function regression?

I haven't got a lot of data and I would like to interpret the fitted model

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven’t got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven't got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven’t got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	local regression, kernel regression, **Gaussian process regression**

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven’t got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	local regression, kernel regression, **Gaussian process regression**
I have a lot of outliers in my data…	

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven’t got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	local regression, kernel regression, **Gaussian process regression**
I have a lot of outliers in my data…	robust regression, Bayesian outlier detection

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven't got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	local regression, kernel regression, **Gaussian process regression**
I have a lot of outliers in my data…	robust regression, Bayesian outlier detection
I have errors in all variables…	

Taking a step - back, the summary

I have a good idea where my data come from	non-linear fits (e.g. lmfit from last Programmers Club) Bayesian models
My data seems to have a lot of collinearity & I have a lot of data.	regular linear regression, basis function regression?
I haven't got a lot of data and I would like to interpret the fitted model	ridge & LASSO regression, maybe kernel regression
My data has a lot of local structure, complex shape, and uncertainties…	local regression, kernel regression, **Gaussian process regression**
I have a lot of outliers in my data…	robust regression, Bayesian outlier detection
I have errors in all variables…	Bayesian models, hyperfit (http://hyperfit.icrar.org/), linmix in IDL (Kelly 2007, ApJ).