# Dublin R Workshop on Gaussian Processes

Mick Cooney
michael.cooney@applied.ai

January 2016

`https://bitbucket.org/kaybenleroll/dublin_r_workshops`.

Code is available in the `wsgp201601/` directory.

Part of this workshop is based on an interesting blog post I came across on R-Bloggers by James Keirstead:

`http://www.jameskeirstead.ca/blog/gaussian-process-regression-with-r/`

Another good blog post to get started is "Basic Regression in Gaussian Processes" by Carl Boettiger:

`http://www.carlboettiger.info/2012/10/17/basic-regression-in-gaussian-processes.html`

The key text for this topic is Gaussian Processes by Rasmussen and Williams. The book is available in PDF online and is excellent though a little dense. It has also been described as using a style that is "an affront to mathematical notation" but I am not enough of an expert to comment beyond noting that it can be confusing.

`http://www.gaussianprocess.org/`

I highly recommend you look at that site and book if exploring further.

# 1. Introduction

In probability theory and statistics, Gaussian processes are a family of statistical distributions in which time plays a role.

In a Gaussian process, every point in some input space is associated with a normally distributed random variable. Moreover, every finite collection of those random variables has a multivariate normal distribution.

The distribution of a Gaussian process is the joint distribution of all those (infinitely many) random variables, and as such, it is a distribution over functions.

Before we can do any work, we first need to determine how to computationally generate these Gaussian Processes. If we take a single sample from a Gaussian Process, what do we have?

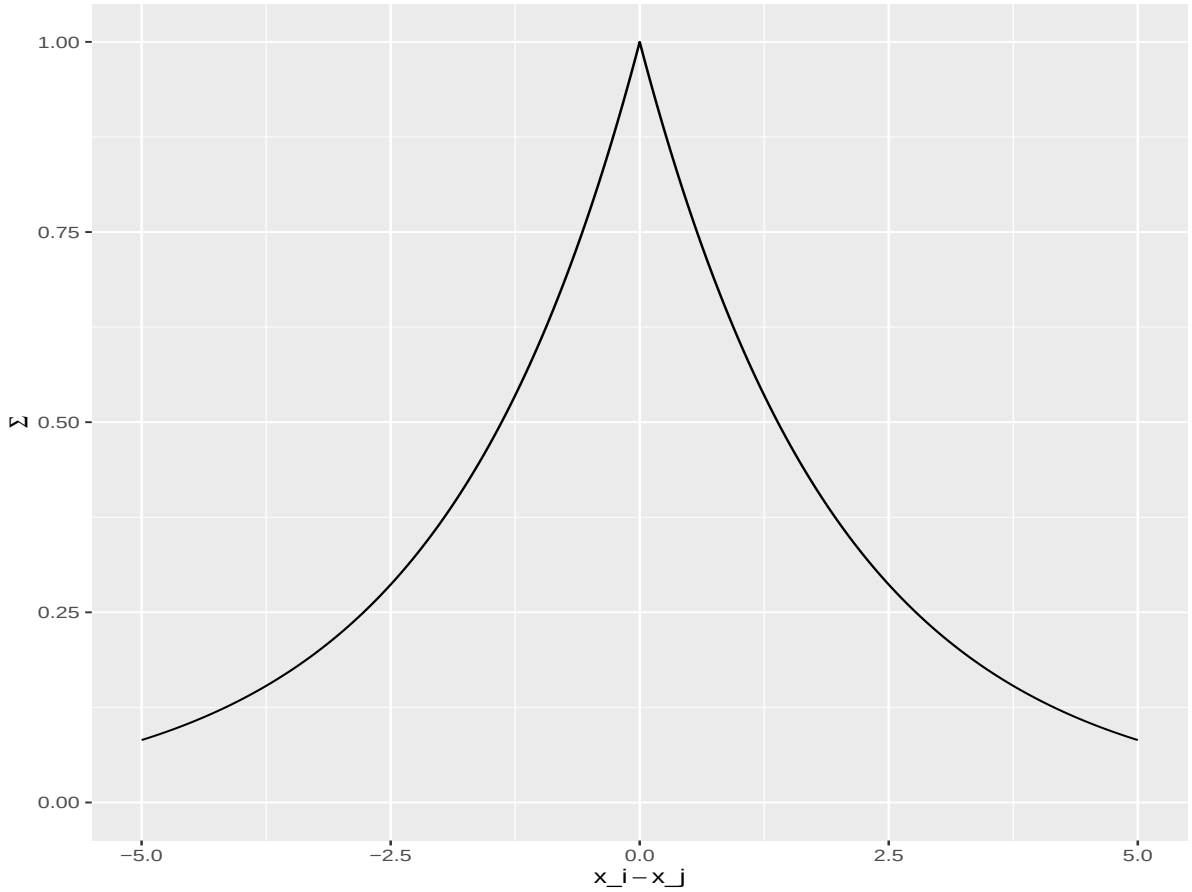
A statistical process is a generalisation of a distribution. A distribution describes random variables that are scalars or vectors. For a process, it describes functions.

At first glance this may appear problematic, but by taking finite samples of the function we can use approximations that are good enough. As a result, we treat a function $f(x)$ as a vector, with each component of the vector being the value of the function $f(x)$ at each discrete step.

To generate these values, each realization of the process is a draw from a multivariate normal distribution. For simplicity we assume the distribution is zero-mean, $\mu_i = 0,\ \forall i$.

We now need to focus on what to use for $\Sigma$ the covariance matrix for the distribution. There are a number of ways to generate this, but we will use the following method as a default:

$$\Sigma_{ij} = \exp\left(-\frac{1}{2} * \left(\frac{|x_i - x_j|}{l}\right)^2\right),\ l > 0$$




The intuition here is that we are generating smooth and continuous functions by ensuring that closer points of the function are more highly correlated.

**Exercise 1.1** Generate 1000 points from a normal distribution with $\mu = 0.0003$ and $\sigma = 0.01$. *HINT:* The R function `rnorm()` can be used to do this.

**Exercise 1.2** Perform some simple checks to ensure you have sampled this data properly. Summary statistics, simple plots of the data and other exploratory methods may be of use for this.

**Exercise 1.3** Generate 1000 points from a multivariate normal distribution using the following parameters:

$$\mu = \begin{bmatrix} 1.0 \\ 1.0 \end{bmatrix} \quad \Sigma = \begin{bmatrix} 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix}$$

*HINT:* The function `mvrnorm` from the `MASS` package is a useful function to do this, though there are others.

**Exercise 1.4** Using summary checks and visualisations, check that the generated data is consistent with what was requested.

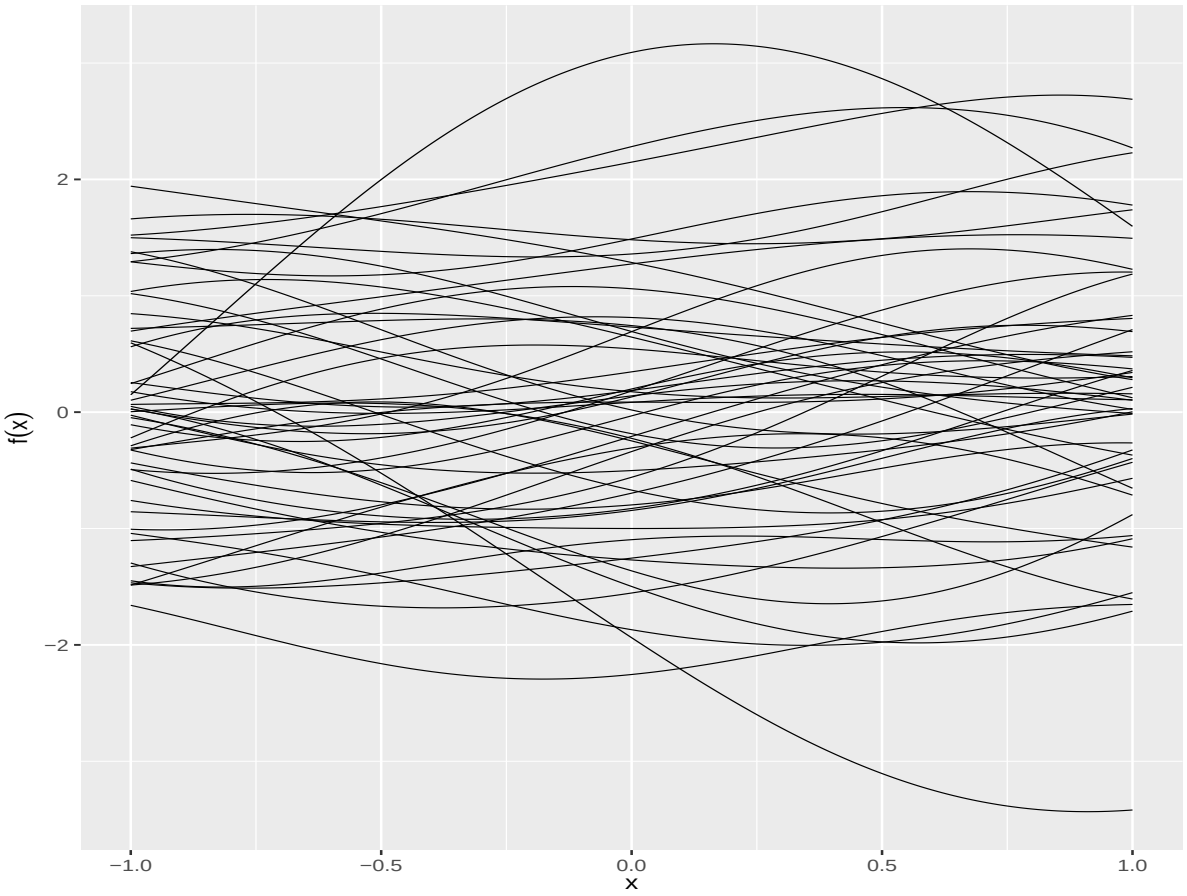
**Exercise 1.5** Using the above distribution, suppose we know that $x_1 = 0.5$. What can we infer about the value of $x_2$? It may make sense to generate more data points to do this.

**Exercise 1.6** Repeat the above task for a 3D multivariate normal.

**Exercise 1.7** Generate 50 samples of a Gaussian Process using zero means and the above covariance function with $x \in (-1, 1)$ and step size of 0.01. Plot the output.

**Exercise 1.8** Repeat the above process but with a step size of 0.1 and 0.001.

**Exercise 1.9** How do the realizations compare with one another? Does the step size have much of an effect on the output?



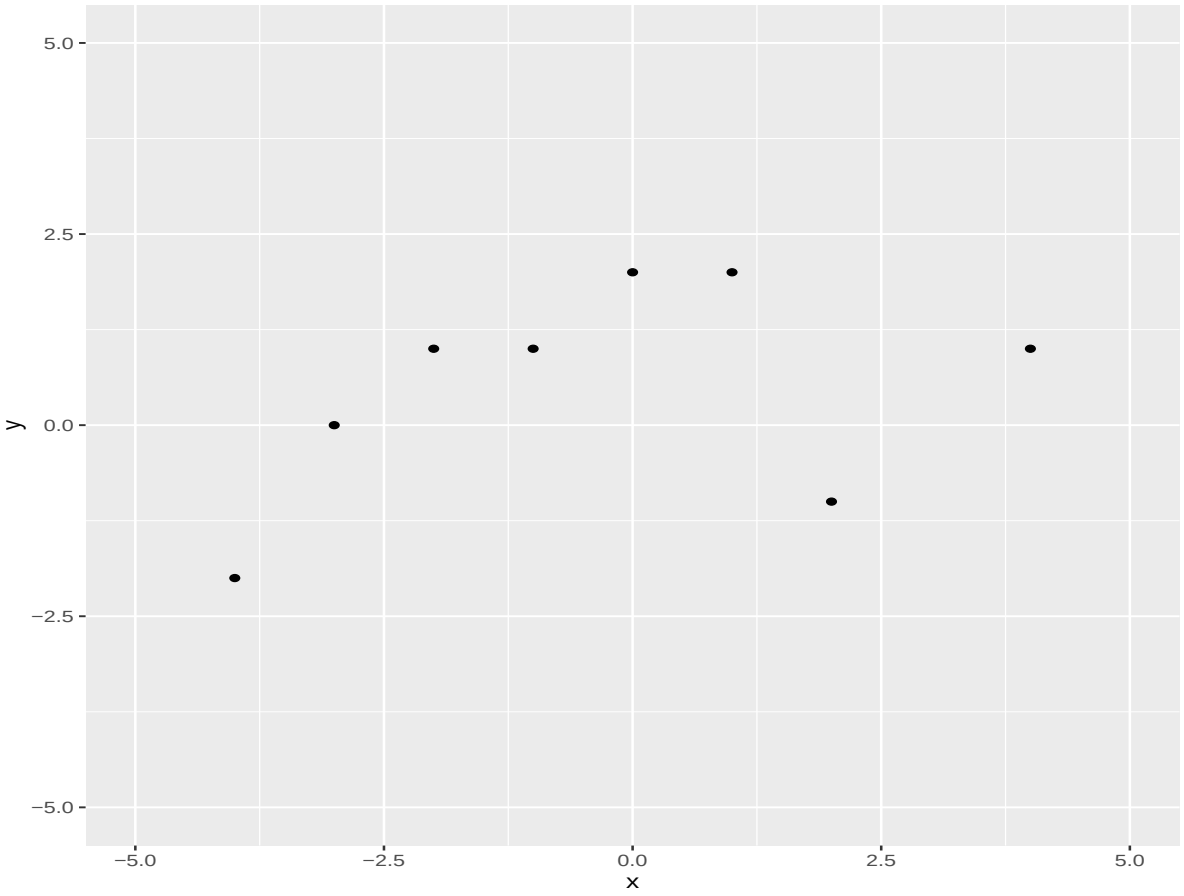
## 2. Gaussian Processes and Simple Regression

Now that we know how to generate Gaussian Processes, we can focus on how we use them in various situations.

In the simplest case, we have a bunch of data that we wish to 'fit' the functions to. This process can naturally adapt to uncertainty in the data, but for the moment we will assume we want to fit to exact values.

```
data_dt <- data.table(x=c(-4,-3,-2,-1, 0, 1, 2, 4),
                      y=c(-2, 0, 1, 1, 2, 2,-1, 1))

qplot(x, y, data = data_dt, geom = 'point', xlim = c(-5,5), ylim = c(-5,5))
```




Conceptually, we sample from the GP, only keeping those functions that satisfy the conditions imposed by the data.

From a computational point of view this will be horribly inefficient, so instead we condition on the data and generate from a modified process that satisfies the data.

We first describe some notation. In this workshop, we will use $x$ to denote the discrete input values over which we evaluate the process realisations. So each GP will have a realisation $f(x)$.

We will denote the data we which to use to fit the GPs as $D$, and the components of $D$ as being $D_x$ and $D_y$. Hopefully this will make the equations simpler the understand than in the book or blogpost.

So, after some mathematics and algebraic manipulation, we see that we can conditionally sample the GP by

modifying $\mu$ and $\Sigma$ in the multivariate normal from which we sample:

$$\begin{aligned} \mu &= k(x, D_x)\, k(D_x, D_x)^{-1} D_y \\ \Sigma &= k(x, x) - k(x, D_x)\, k(D_x, D_x)^{-1}\, k(D_x, x) \end{aligned}$$

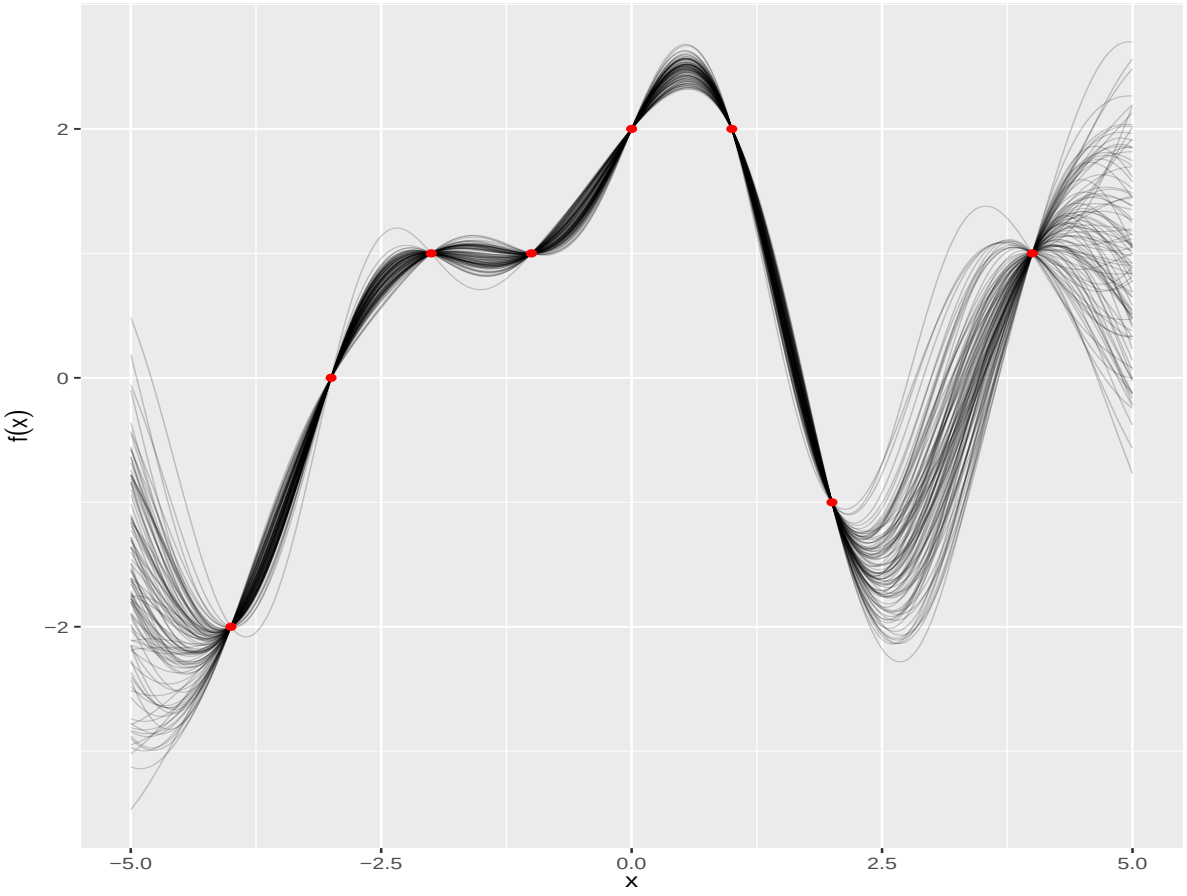
**Exercise 2.1** Using the data in `data_dt`, calculate the modified parameters $\mu$ and $\Sigma$, using an interval of $(-5, 5)$ for x with a discrete step size of 0.01. *HINT:* You may find '`%*%`' and `solve()` useful for this.

**Exercise 2.2** Use these parameters to create 50 samples that fits the data. Plot the output.

**Exercise 2.3** Calculate the resulting inference for the $f(2.5)$ with a band of 80%.

**Exercise 2.4** Repeat the above process but using 1,000 samples. How does this affect our 80% interval?

**Exercise 2.5** Repeat the above with a 0.005 interval for $x$. How does this affect the inference of $f(2.5)$?



# 3. Capturing Uncertainty in Regression

In many cases, we do not have exact measurements for our data and wish to capture this uncertainty in our approach. Gaussian processes handle this in a natural way: rather than enforcing a hard constraint for each data point, we instead require that the sample goes 'close' to it. We add a probability distribution around each point, and the samples obey this 'soft' constraint.

For the moment we assume the noise around each observation is normal and i.i.d. but GP's can handle more complicated covariance structures. We denote the standard deviation of this noise as $\sigma$.

Once again, we use some theory and algebra to get:

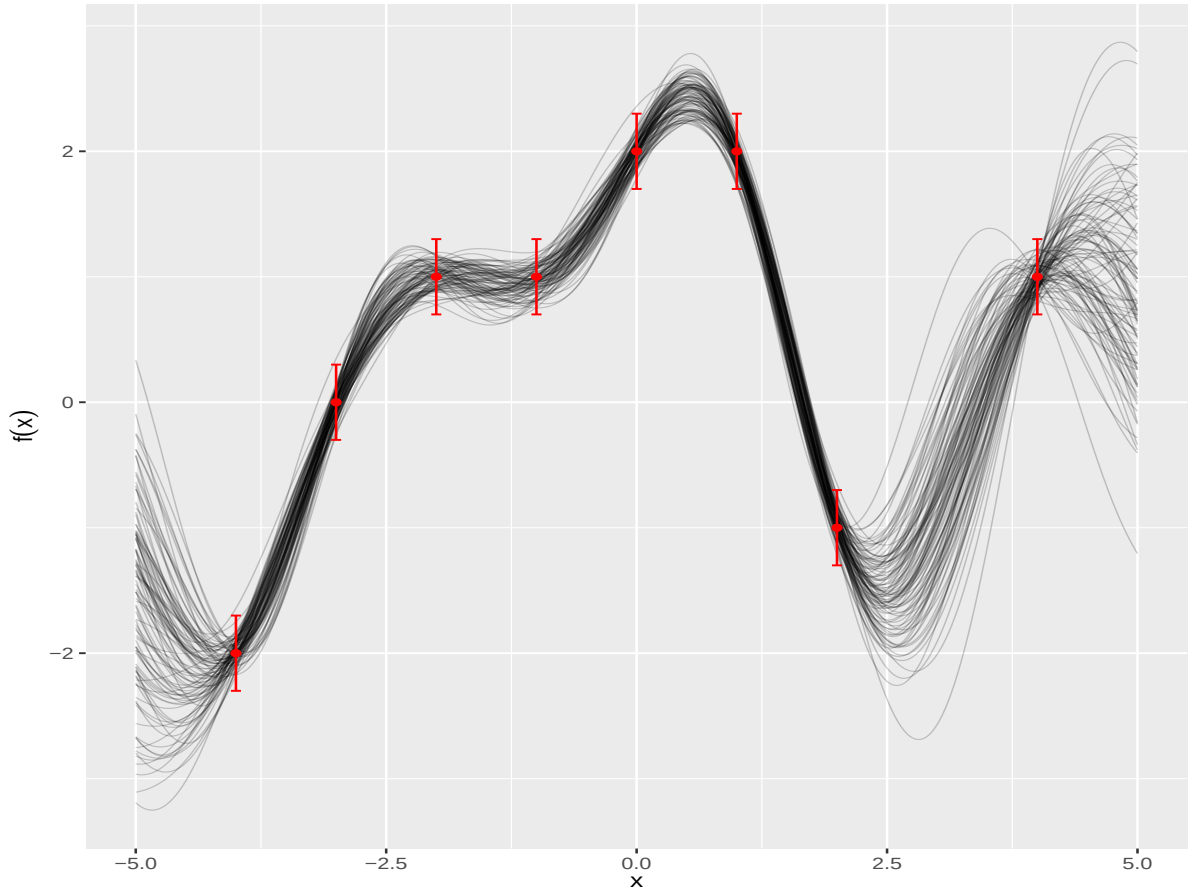
$$\begin{aligned} \mu &= k(x, D_x)\left(k(D_x, D_x) + \sigma^2 I\right)^{-1} D_y \\ \Sigma &= k(x, x) - k(x, D_x)\left(k(D_x, D_x) + \sigma^2 I\right) y^{-1}\, k(D_x, x) \end{aligned}$$

**Exercise 3.1** Using the data in `data_dt` and a noise std dev of 0.1, calculate the modified parameters $\mu$ and $\Sigma$, using an interval of $(-5, 5)$ for x with a discrete step size of 0.01.

**Exercise 3.2** Use these parameters to create 50 samples that fits the data. Plot the output.

**Exercise 3.3** Calculate the resulting inference for the $f(2.5)$ with a band of 80%. How does this compare to the inference for the noise-free data?

**Exercise 3.4** Investigate the effect of reducing the step size to 0.005.


2
0
−2
f(x)
−5.0
−2.5
0.0
2.5
5.0
x

# 4. Gaussian Processes in the Wild

When using GPs in anger, we do not want to be doing all that linear algebra are sampling, so we use R packages to do as much of the work as possible.

There are a number of packages that implement Gaussian processes, and the simplest seems to be the package `kernlab`.

**Exercise 4.1** Use the `gausspr()` function with the data from the previous section to predict a value for $f(2.5)$. Assume there is no noise in the data.

**Exercise 4.2** Repeat the process but with an additive noise term for $y$ of 0.1.

**Exercise 4.3** Use the provided functions `regression_func()` and `add_additive_noise()` to generate 20 data points for the function in the rectangle $((-5, -5), (5, 5))$. with 0.1 as additive noise.

**Exercise 4.4** Infer values for data that is not in your sample and compare the actual value of the function. What effect does the variance input to the GP regression have on the quality of the inference?