An introduction to Gaussian Processes

Andrew C Parnell

github.com/andrewcparnell/intro_to_gaussian_processes

Why learn about Gaussian Processes?

- They are a great introduction to thinking about the multivariate normal distribution
- They are widely used in spatial statistics and machine learning
- They are very flexible and extendable
- Some of you will be using them for your PhD

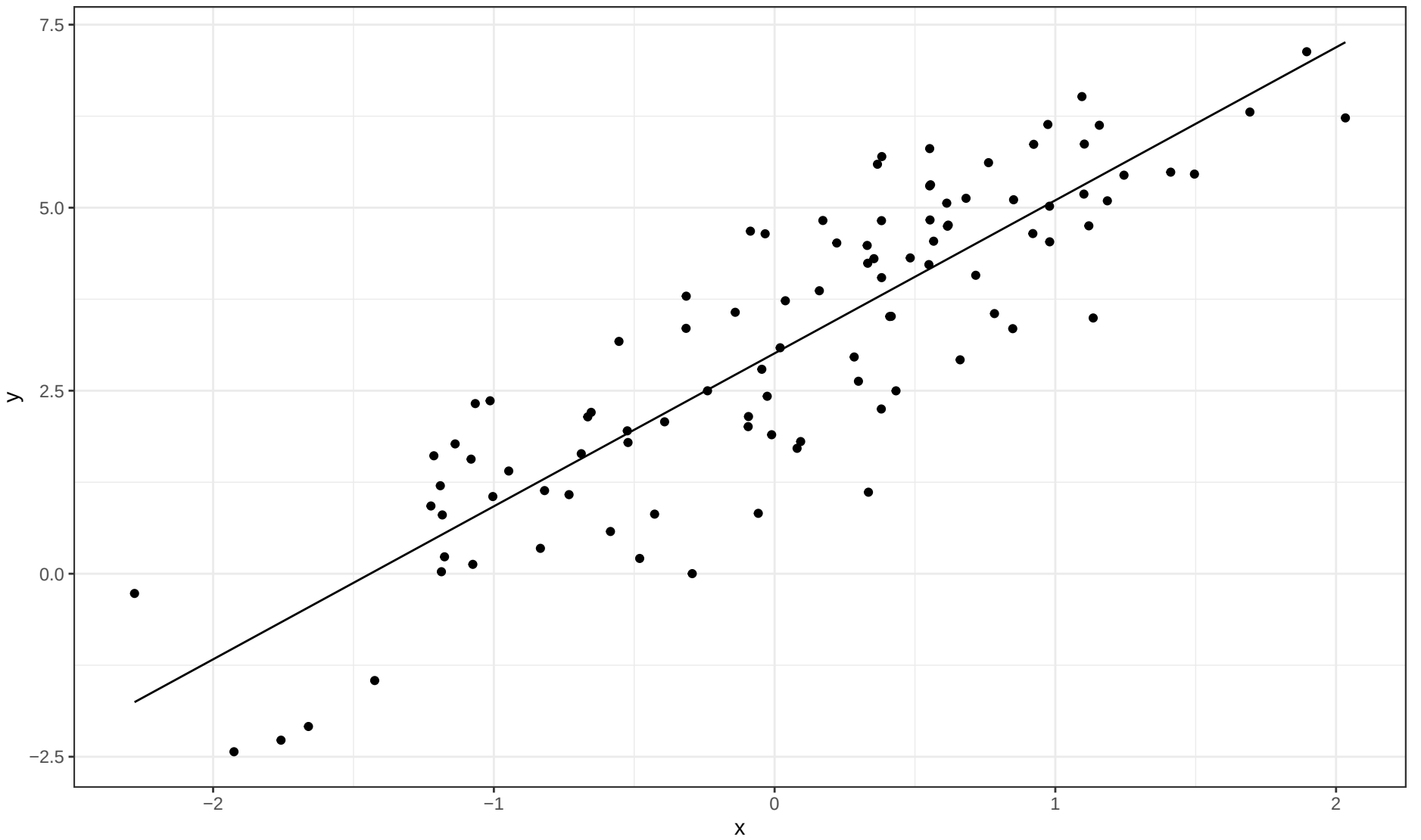
To follow this presentation you will need to understand linear regression and a bit of matrix algebra

Back to linear regression

```
lr_dat = tibble(
  x = sort(rnorm(100)),
  y = rnorm(100, 3 + 2 * x, 1),
  fits = fitted(lm(y ~ x))
)
p = ggplot(lr_dat, aes(x, y)) +
  geom_point() +
  geom_line(aes(y = fits))
```

Linear regression plot

```
print(p)
```



Linear regression maths

We normally write this as:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i;\ i = 1, \ldots, n,\ \epsilon_i \sim N(0, \sigma^2)$$

. . . so there are three parameters to estimate.

Linear regression maths

We normally write this as:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i;\ i = 1, \ldots, n,\ \epsilon_i \sim N(0, \sigma^2)$$

. . . so there are three parameters to estimate.

Another way of writing this is:

$$y_i | x_i \sim N(\beta_0 + \beta_1 x_i, \sigma^2)$$

Linear regression in matrices

Suppose we write:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

Linear regression in matrices

Suppose we write:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}, \beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}$$

We can then write the model as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_n \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

or

$$y = X\beta + \epsilon$$

Fitting linear regression via maximum likelihood

One way to fit is to compute the (log) likelihood and maximise it:

$$\log L = \sum_{i=1}^{n} \log \left\{ \sqrt{\frac{1}{2\pi\sigma^2}} \exp \left[-\frac{1}{2\sigma^2} (y_i - \beta_0 - \beta_1 x_i)^2 \right] \right\}$$

Fitting linear regression via maximum likelihood

One way to fit is to compute the (log) likelihood and maximise it:

$$\log L = \sum_{i=1}^{n} \log \left\{ \sqrt{\frac{1}{2\pi\sigma^2}} \exp \left[-\frac{1}{2\sigma^2} (y_i - \beta_0 - \beta_1 x_i)^2 \right] \right\}$$

```
beta_0 = 0; beta_1 = 0; sigma = 1
with(lr_dat, sum(dnorm(y, beta_0 + beta_1 * x, sd = sigma,
                       log = TRUE)))
```

```
## [1] -807.1095
```

Fitting linear regression via maximum likelihood

One way to fit is to compute the (log) likelihood and maximise it:

$$\log L = \sum_{i=1}^{n} \log \left\{ \sqrt{\frac{1}{2\pi\sigma^2}} \exp \left[-\frac{1}{2\sigma^2} (y_i - \beta_0 - \beta_1 x_i)^2 \right] \right\}$$

```
beta_0 = 0; beta_1 = 0; sigma = 1
with(lr_dat, sum(dnorm(y, beta_0 + beta_1 * x, sd = sigma,
                       log = TRUE)))
```

```
## [1] -807.1095
```

```
beta_0 = 3; beta_1 = 2; sigma = 1
with(lr_dat, sum(dnorm(y, beta_0 + beta_1 * x, sd = sigma,
                       log = TRUE)))
```

```
## [1] -146.4469
```

LR via the multivariate normal distribution

If you simplify the calculation on the previous slide you get:

$$\log L = -\frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

(I have dropped the the 2π as it's just a constant)

LR via the multivariate normal distribution

If you simplify the calculation on the previous slide you get:

$$\log L = -\frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

(I have dropped the the 2π as it's just a constant)

This can be re-written in matrix format

$$\log L = -\frac{1}{2}\log\left[det(\Sigma)\right] - \frac{1}{2}(y - X\beta)^T\Sigma^{-1}(y - X\beta)$$

where $\Sigma = \sigma^2 I$

LR via the multivariate normal distribution

If you simplify the calculation on the previous slide you get:

$$\log L = -\frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

(I have dropped the the 2π as it's just a constant)

This can be re-written in matrix format

$$\log L = -\frac{1}{2}\log\left[det(\Sigma)\right] - \frac{1}{2}(y - X\beta)^T\Sigma^{-1}(y - X\beta)$$

where $\Sigma = \sigma^2 I$

This is the likelihood for a **multivariate normal distribution** written $y \sim MVN(X\beta, \Sigma)$

Fitting linear regressions using the multivariate normal

Two different ways to write the log likelihood:

```
beta_0 = 3; beta_1 = 2; sigma = 1
with(lr_dat, sum(dnorm(y, beta_0 + beta_1 * x, sd = sigma,
                       log = TRUE)))
```

```
## [1] -146.4469
```

Fitting linear regressions using the multivariate normal

Two different ways to write the log likelihood:

```
beta_0 = 3; beta_1 = 2; sigma = 1
with(lr_dat, sum(dnorm(y, beta_0 + beta_1 * x, sd = sigma,
                       log = TRUE)))
```

```
## [1] -146.4469
```

```
library(mvtnorm)
Sigma = diag(sigma, nrow(lr_dat))
X = with(lr_dat, cbind(1,x))
beta = c(beta_0, beta_1)
with(lr_dat, dmvnorm(y, X%*%beta, Sigma,
                     log = TRUE))
```

```
## [1] -146.4469
```

Changing the off-diagonals

The matrix Sigma currently looks like this:

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix}$$

What would happen if we changed the off-diagonal terms? These represent the covariances (or scaled correlations) between the y-values

Changing the off-diagonals

The matrix Sigma currently looks like this:

$$\Sigma = \begin{bmatrix} \sigma^2 & 0 & \dots & 0 \\ 0 & \sigma^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma^2 \end{bmatrix}$$

What would happen if we changed the off-diagonal terms? These represent the covariances (or scaled correlations) between the y-values

Gaussian processes

The *key idea* in Gaussian Processes is to change the off-diagonals of Σ so that the you get higher correlations between y_i and y_j when x_i and x_j are close.

Autocovariance functions

The most common autocovariance function (ACF):

$$\Sigma_{ij} = \tau^2 \exp\left[-\frac{(x_i - x_j)^2}{\theta}\right]$$

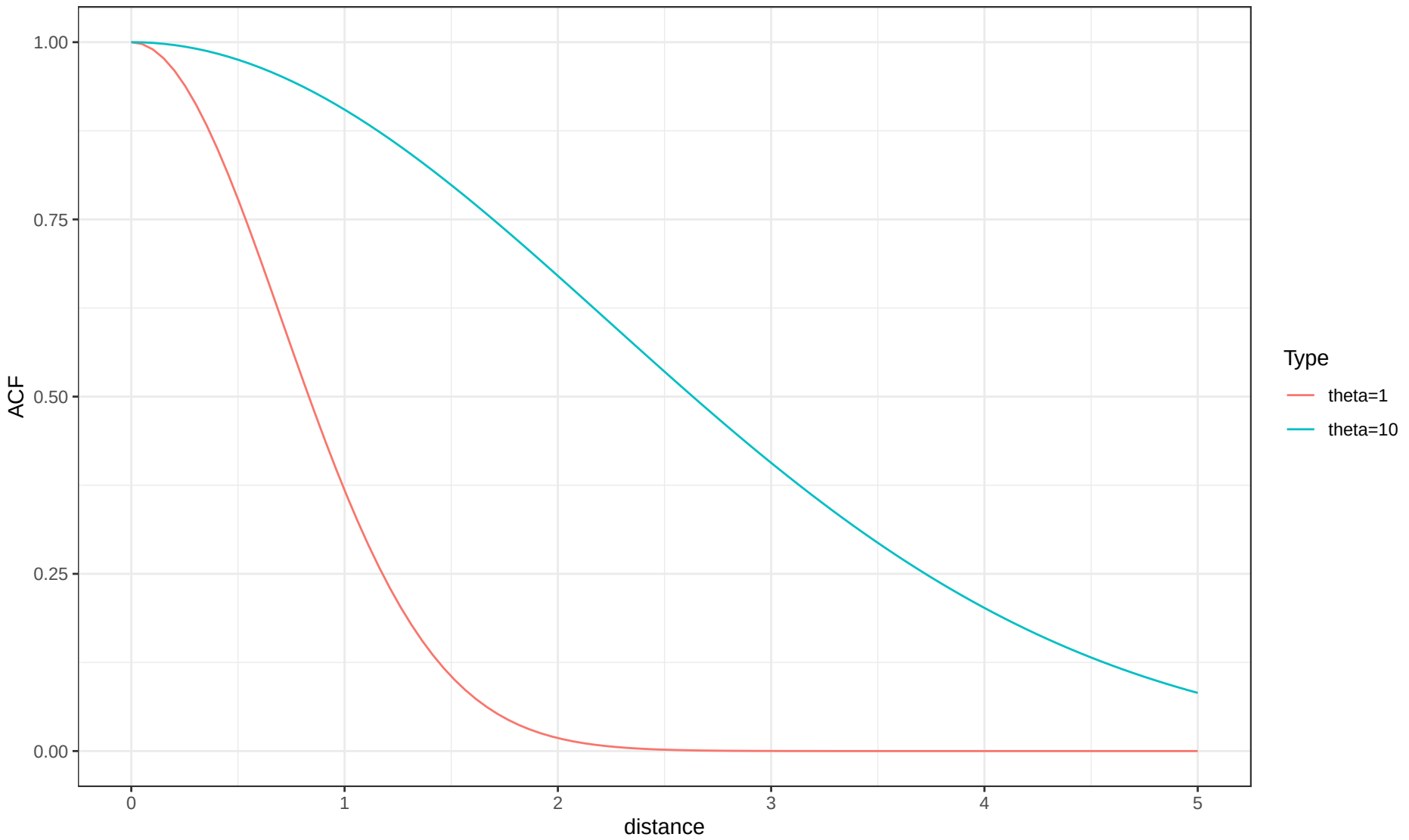
for $i \neq j$ with $\Sigma_{ii} = \tau^2 + \sigma^2$

This can be plotted:

```
sigma = 1; theta1 = 10; theta2 = 1
acv = tibble(
  distance = seq(0, 5, length = 100),
  `theta=10` = sigma^2 * exp(-(distance ^ 2) / theta1),
  `theta=1` = sigma^2 * exp(-(distance ^ 2) / theta2)
)
p = acv %>%
  gather(key = Type, value = ACF, -distance) %>%
  ggplot(aes(x = distance, y = ACF, colour = Type)) +
  geom_line()
```

Pictures of squared exponential covariance funct

```
print(p)
```



Fitting a model via the MVN

We can fit this model in exactly the same way via the log-likelihood:

```
suppressPackageStartupMessages(library(fields))
sigma = 1; tau = 1
Sigma = with(lr_dat,
             diag(sigma^2, nrow(lr_dat)) +
               tau^2 * Exp.cov(x, x,
                               theta = theta2, p = 2))
round(Sigma[1:3,1:3], 3)
```

```
##       [,1]  [,2]  [,3]
## [1,] 2.000 0.882 0.762
## [2,] 0.882 2.000 0.972
## [3,] 0.762 0.972 2.000
```

```
with(lr_dat, dmvnorm(y, X%*%beta, Sigma,
                     log = TRUE))
```

```
## [1] -150.6072
```

The magic GP formula

Suppose you want to predict some new y values for some given new x values. Write these as y_{new} and x_{new}, and set up the problem as:

$$\begin{bmatrix} y \\ y_{new} \end{bmatrix} \sim MVN\left(\begin{bmatrix} X\beta \\ X_{new}\beta \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_{new} \\ \Sigma_{new}^T & \Sigma_{new,new} \end{bmatrix}\right)$$

where $\Sigma_{new,i,j} = \tau^2 \exp\left[-\frac{(x_i - x_{new,j})^2}{\theta}\right]$,
$\Sigma_{new,new,i,j} = \tau^2 \exp\left[-\frac{(x_{new,i} - x_{new,j})^2}{\theta}\right]$, and $\Sigma_{new,new,i,i} = \sigma^2 + \tau^2$

The magic GP formula

Suppose you want to predict some new y values for some given new x values. Write these as y_{new} and x_{new}, and set up the problem as:

$$\begin{bmatrix} y \\ y_{new} \end{bmatrix} \sim MVN\left(\begin{bmatrix} X\beta \\ X_{new}\beta \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma_{new} \\ \Sigma_{new}^T & \Sigma_{new,new} \end{bmatrix}\right)$$

where $\Sigma_{new,i,j} = \tau^2 \exp\left[-\frac{(x_i - x_{new,j})^2}{\theta}\right]$,
$\Sigma_{new,new,i,j} = \tau^2 \exp\left[-\frac{(x_{new,i} - x_{new,j})^2}{\theta}\right]$, and $\Sigma_{new,new,i,i} = \sigma^2 + \tau^2$

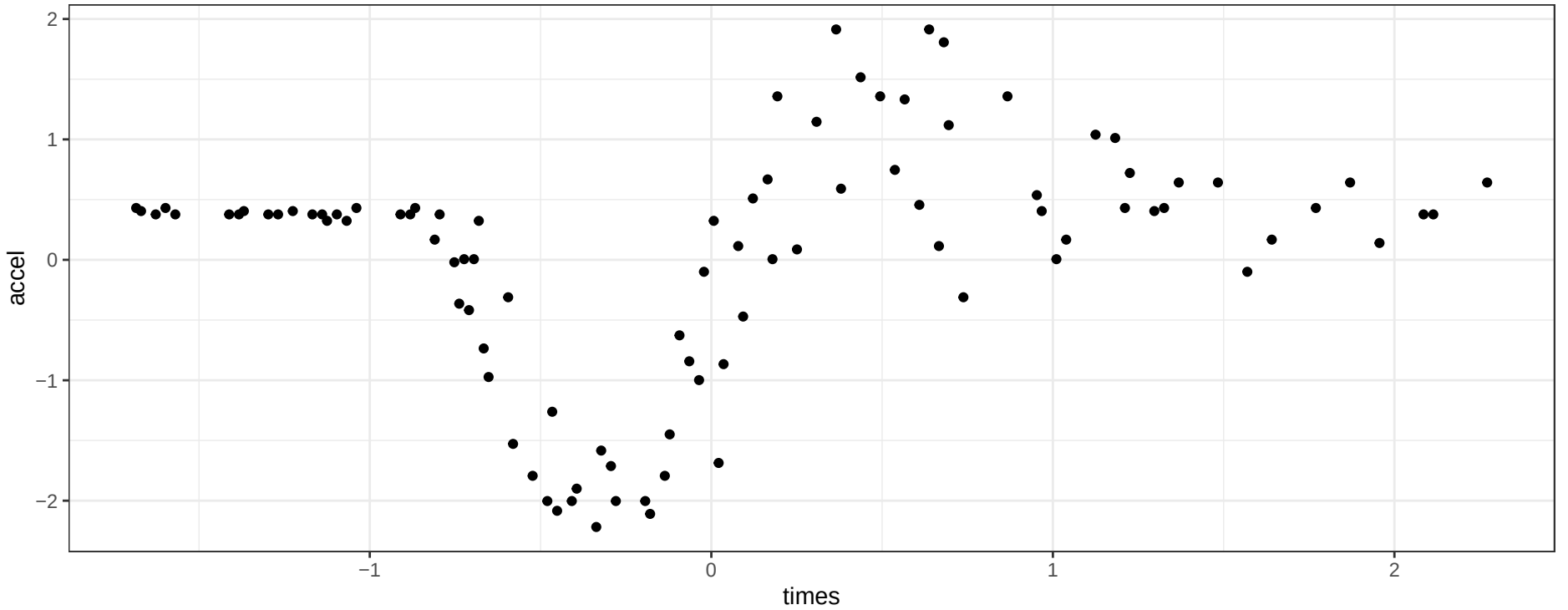
The magic formula gives you

$$y_{new}|y \sim MVN(X_{new}\beta + \Sigma_{new}^T\Sigma^{-1}(y - X\beta)), \Sigma_{new,new} - \Sigma_{new}^T\Sigma^{-1}\Sigma_{new})$$

This means that, once the parameters are estimated, we can produce instant predictions!

A new regression data set

```
suppressPackageStartupMessages(library(boot))
scale2 = function(x) (x - mean(x))/sd(x)
motor2 = motor %>% mutate_all(scale2)
motor2 %>%
  ggplot(aes(x = times, y = accel)) +
  geom_point()
```



Fitting a GP

```
nll = function(par) {
  tau = par[1]
  sigma = par[2]
  theta = par[3]
  Sigma = with(motor2,
               diag(sigma^2, nrow(motor2)) +
                 tau^2 * Exp.cov(times, times,
                                 theta = theta, p = 2))
  with(motor2, -dmvnorm(accel,
                        cbind(1, times)%*%beta,
                        sigma = Sigma, log = TRUE))
}
nll(c(1,1,1))
```

```
## [1] 138.4153
```

Optimising the parameters

```
library(optimx)
answer = optimx(par = c(1,1,1),
                fn = nll,
                method = 'BFGS')
print(answer)
```

```
##              p1          p2        p3    value fevals gevals n
## BFGS 3.441868 -0.4365699 0.7728277 81.24886     57     12
##      kkt1 kkt2 xtime
## BFGS TRUE TRUE   0.1
```

Making predictions using GPs

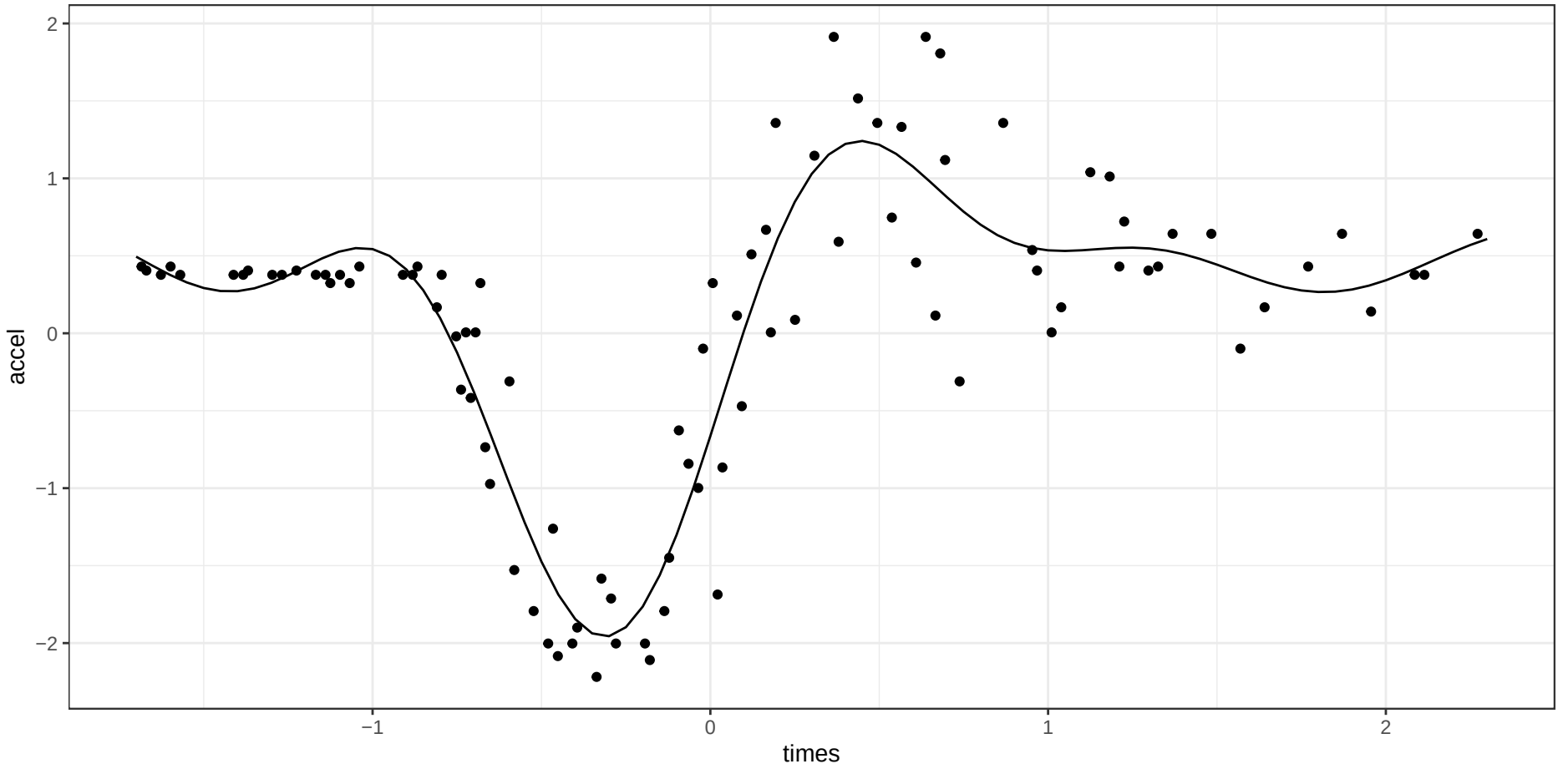
Maynooth University National University of Ireland Maynooth | Hamilton Institute

Now use the magic formula to predict for new x values:

```
tau = answer$p1
sigma = answer$p2
theta = answer$p3
x_new = pretty(motor2$times, n = 100)
Sigma_new = with(motor2,
                 tau^2 * Exp.cov(times, x_new,
                                 theta = theta, p = 2))
Sigma = with(motor2,
                 diag(sigma^2,nrow(motor2)) + tau^2 * Exp.cov(time
                                 theta = theta, p = 2))
GP_pred = tibble(
  x_new = x_new,
  mu_pred = t(Sigma_new)%*%solve(Sigma)%*%motor2$accel
)
```

Plot of predictions

```
ggplot(motor2, aes(x = times, y = accel)) +
  geom_point() +
  geom_line(data = GP_pred, aes(x = x_new, y = mu_pred))
```



Shiny app

Maynooth University
National University of Ireland Maynooth
Hamilton Institute

andrewcparnell.shinyapps.io/GP_explore

or

is.gd/gp_play

What else can you do with GPs?

- You can put the linear regression terms back in to the mean if you want
- If you drop the σ^2 off the diagonal you can perform interpolation rather than smoothing
- The x values don't have to be one dimensional. Provided you can create a univariate distance between them you can run a GP. This is the foundation for spatial statistics and Kriging
- Lots of work on clever covariance functions
- Easy to make Bayesian by putting priors on parameters
- Big road block is calculation of Σ^{-1}

Summary

- You have now seen that a Gaussian Process is just a multivariate normal distribution
- The first clever trick is to make the observations correlated to each other by an autocovariance function
- The second clever trick is to use the magic multivariate normal formula to get instant access to predictions (and confidence intervals)

Thanks, and enjoy using Gaussian Processes!