1. Simulating Gaussian vectors

Simon Vaughan [*]

July 8, 2016

Generating independent Gaussian variables.

We can generate a single random number from a Guassian (=Normal) distribution as follows

```
  x <- rnorm(1, mean = 0, sd = 1)
  print(x)

## [1] -0.9708054
```

The `rnorm()` function generates the random number. The probability density function (pdf) of the 1-dimensional Gaussian distribution is

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2} \frac{(x-\mu)^2}{2\sigma^2} \right) \tag{1}$$

The first part $(1/\sqrt{2\pi\sigma^2})$ ensures that the pdf integrates to 1. I.e. that

$$\int_{-\infty}^{+\infty} p(x) dx = 1 \tag{2}$$

which should be true of any well-defined pdf. The scalars μ and σ^2 define the mean and variance of the distribution. These two numbers completely specify the 1-dimensional Gaussian distribution.

The mean (or expectation) is

$$E[x] = \int_{-\infty}^{+\infty} x p(x) dx = \mu. \tag{3}$$

(For a Gaussian this is also the peak position, or *mode*.) The variance is a measure of the spread around this

$$V[x] = E[(x-\mu)^2] = \int_{-\infty}^{+\infty} (x-\mu)^2 p(x) dx = \sigma^2. \tag{4}$$

We could equally well ask it to produce $n = 20$ random numbers and then plot them. One way to do this is to define a vector (a 1-dimensional array) and then use a `for` loop to populate this by repeatedly drawing one random value.

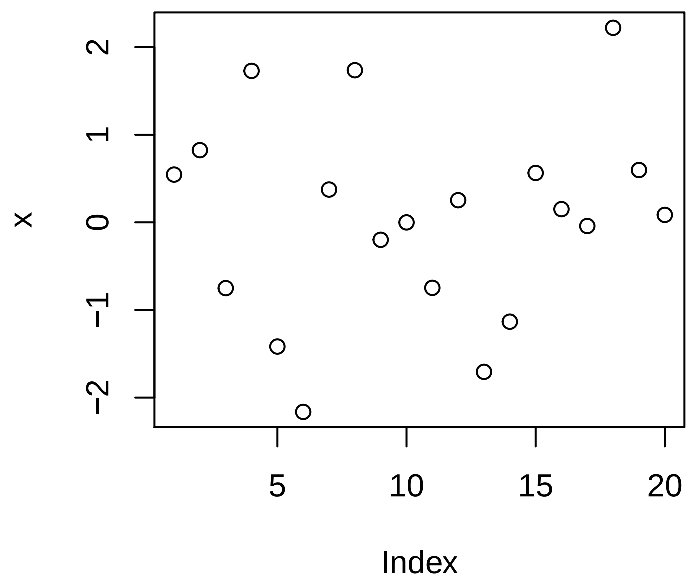
```
    n <- 20
    x <- array(NA, dim = n)
    for (i in 1:n) {
      x[i] <- rnorm(1, mean = 0, sd = 1)
    }
```

[*]Email: sav2@le.ac.uk

More efficient is to ask the function `rnorm()` to generate n values in one go.

The `rnorm()` function is itself written in C and so is fast. In general R code is faster if more work can be done 'inside' compiled functions. Manipulating vectors and arrays using functions will be much faster than using R to manipulate the elements using an explicit loop.

```r
n <- 20
x <- rnorm(n, mean = 0, sd = 1)
plot(x)
```



In this case all the elements of x are independent and identically distributed (iid). They all have the same Gaussian distribution (with mean $\mu = 0$ and variance $\sigma^2 = 1$), and they are independent, meaning that each point is drawn at random with no connection to any other point in the sequence.

We can think of these as $n = 20$ draws of a scalar (1-dimensional) random variable. Or we could think of this as a single draw of a n-dimensional random vector, $\mathbf{x}$.

The n-dimensional Gaussian distribution

The n-dimensional Gaussian distribution has the following pdf

$$p(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x}-\boldsymbol{\mu})\right) \tag{5}$$

where $\mathbf{x}$ is an n-dimensional random vector, $\boldsymbol{\mu}$ is an n-dimensional vector of means and Σ is an $n \times n$ matrix called the *covariance matrix*. The (vector) mean $\boldsymbol{\mu}$ and covariance (matrix) Σ completely specify a Gaussian distribution for any number of dimensions.

Like the 1-dimensional case (equation 1), the first term in equation 5 $(1/([2\pi]^{n/2}|\Sigma|^{1/2}))$ is a normalisation term and ensures that

$$\int_{-\infty}^{+\infty} p(\mathbf{x})d\mathbf{x} = 1 \tag{6}$$

Inside the exponential is a *quadratic form*

$$Q = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}) \tag{7}$$

which is the n-dimensional equivalent to the exponent in equation 1 which we can write as $(x - \mu)(\sigma^2)^{-1}(x - \mu)$.

A valid covariance matrix must have certain properties. It must be a square, symmetric matrix, $\Sigma_{ij} = \Sigma_{ji}$ for $i, j = 1, 2, \ldots, n$. It must be *positive semi-definite*. This several things (most of which are equivalent to each other):

- Has a determinant $|\Sigma| \geq 0$
- Has a unique Cholesky decomposition $\Sigma = LL^T$ where L is a lower triangular matrix
- Has an inverse Σ^{-1} that is also positive semi-definite
- Has quadratic form $\mathbf{y}^T \Sigma \mathbf{y} \geq 0$ and also $\mathbf{y}^T \Sigma^{-1} \mathbf{y} \geq 0$ for any real n-vector $\mathbf{y}$ (with equality only when $\mathbf{y} = \mathbf{0}$).
- All the eigenvalues are $\lambda_i \geq 0$

Therefore $|\Sigma|^{-1/2}$ and Σ^{-1} in equation 5 are well-defined. An example is:

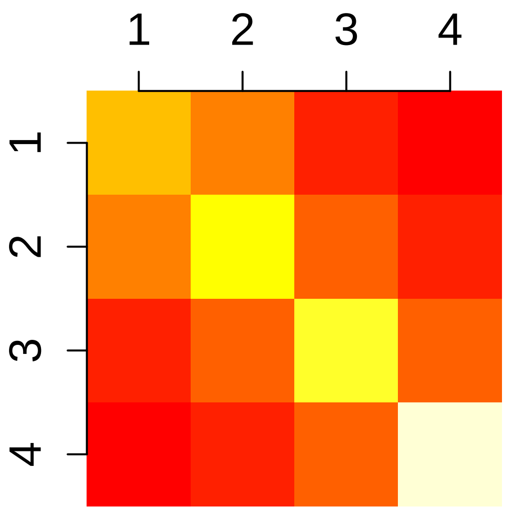
$$\Sigma = \begin{pmatrix} 1.00 & 0.50 & 0.10 & 0.01 \\ 0.50 & 1.50 & 0.40 & 0.10 \\ 0.10 & 0.40 & 2.00 & 0.30 \\ 0.01 & 0.10 & 0.30 & 3.00 \end{pmatrix} \tag{8}$$

We can understand the information contained in Σ as follows. The leading diagonal entires represent the variances of the elements of the random vector $\mathbf{x}$. I.e. $\Sigma_{ii} = \sigma_i^2$. The off-diagonal elements are the *covariances* between the ith and jth elements, i.e. $cov(x_i, x_j) = \Sigma_{ij}$. These tell us if the ith and jth elements are well-correlated or not. If the covariances $\Sigma_{ij} = 0$ for $i \neq j$ then the value of x_i is uncorrelated[1] with x_j. The covariances must obey $|\Sigma_{ij}|^2 \leq \Sigma_{ii}\Sigma_{jj}$.

We can visualise this as an image using `image(S)`. (Although this does require a few extra lines to 'flip' the image so that the row $i = 1$ appear at the top.)

```
  S <- c(1.00, 0.50, 0.10, 0.01,
         0.50, 1.50, 0.40, 0.10,
         0.10 ,0.40 ,2.00 ,0.30,
         0.01, 0.10, 0.30, 3.00)
S <- matrix(S, nrow=4)
image(x=1:4, y=1:4, z=sqrt(S[1:4,4:1]),
      xlab="", ylab="", bty = "n", asp=1,
      cex.axis=1.4, xaxt="n", yaxt="n")
axis(3, at = 1:4, cex.axis=1.4)
axis(2, at = 1:4, pos = 0.5, labels=c("4","3","2","1"), cex.axis=1.4)
```

[1]In general, there is a difference between *independent* and uncorrelated. However, in the special case of a Gaussian distribution they are equivalent.s



What is covariance? Let's start with the mean, which describes the 'centre' of the distribution of the random variable

$$\mu = E[x] = \int_{-\infty}^{+\infty} xp(x)dx. \tag{9}$$

This is the definition of the *mean* or *expectation* of a single variable. The same applies to vectors.

$$\boldsymbol{\mu} = E[\mathbf{x}] = \int_{-\infty}^{+\infty} \mathbf{x}p(\mathbf{x})d\mathbf{x}. \tag{10}$$

The variance is the expectation of the *square deviation* around the mean

$$\sigma_x^2 = E[(x-\mu)(x-\mu)], \tag{11}$$

it describes the typical spread around the mean. If we have many variables (x_i, x_j, etc.), we need to know not just how each one is spread but whether each one varies in a way that is correlated with any other variables. The covariance tells us how x_i and x_j vary together

$$\Sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] \tag{12}$$

So you can see

$$\Sigma_{ii} = E[(x_i - \mu_i)(x_i - \mu_i)] = \sigma_i^2 \tag{13}$$

Informally, you can think about it this way. If x_j is often above its mean when x_i is also above its mean (and x_j is below usually when x_i is below), then the product $(x_i - \mu_i)(x_j - \mu_j)$ is likely to be positive, and positive in expectation. So there will be positive covariance. If x_j is independent of x_i, so that it is equally likely to be above or below its mean when x_i is above its mean, then $(x_i - \mu_i)(x_j - \mu_j)$ is equally likely to be positive or negative, so will tend to zero in expectation. In this case the covariance is zero. So it tells us how much two variables, or two elements of a random vector, vary together.

UNIVERSITY OF LEICESTER

Generating n-dimensional random vectors

To generate an n-dimensional Gaussian random vector we first define a vector of n means, and an $n \times n$ covariance matrix. We will first begin with the identity matrix I_n.

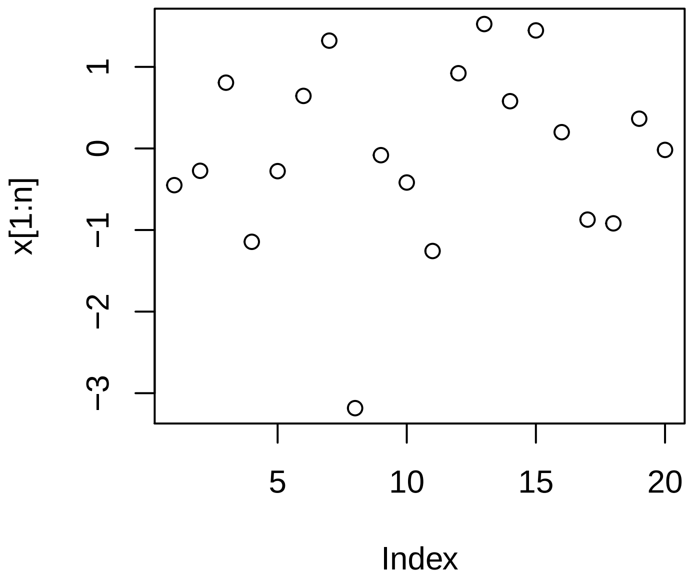
```
  mu <- array(0, dim = n)
  S <- diag(n)
  print(S[1:6, 1:6])

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    0    0    0    0    0
## [2,]    0    1    0    0    0    0
## [3,]    0    0    1    0    0    0
## [4,]    0    0    0    1    0    0
## [5,]    0    0    0    0    1    0
## [6,]    0    0    0    0    0    1
```

Now we need to load an extra *package* which contains the function we need

```
  install.packages("mvtnorm")
```

Once this is installed we never need to do this again. Now we have access to a new function rmvnorm.

```
  x <- mvtnorm::rmvnorm(1, mu, S)
  plot(x[1:n])
```



The output of `rmvnorm` is a $1 \times n$ array. In order to plot the 20 value, the simplest method is to use `[1:n]` to extract the elements 1 to n. Notice there is no *structure* to the output data, again the values are all indepenent of one another.

Changing the covariance matrix

Now, let's introduce more structure to the data, by using a less trivial covariance matrix. We first define positions along the horizontal axis by a vector $\mathbf{t}$. This might be as simple as $\mathbf{t} = \{1, 2, \ldots, n\}$ but the spacing could be arbitrary, $\mathbf{t} = \{t_i | i = 1, 2, \ldots, n\}$.

We define the *lag* as the separation (in time) between two points, $\tau_{ij} = t_i - t_j$. And the absolute value of this is $r = |\tau|$. The *autocovariance function* (ACV) defines the variances and covariances for each element of a random vector $\mathbf{y}$ as a function of r.

$$ACV(r) = A \exp\left(-\frac{r^2}{2l^2}\right) \tag{14}$$

or equivalently

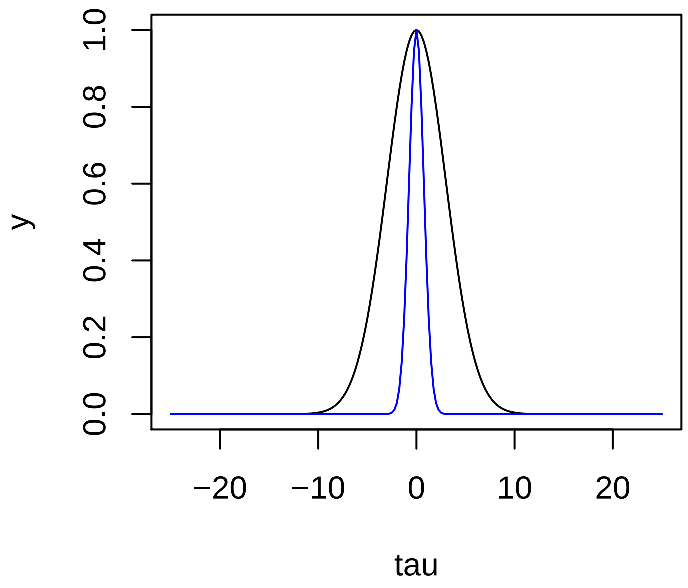
$$ACV(|t_i - t_j|) = A \exp\left(-\frac{1}{2}\frac{|t_i - t_j|^2}{l^2}\right) \tag{15}$$

First, let's plot this for two different l values. We define a function

```r
#define a function to compute the simple ACV
acv <- function(tau, A, l) {
  acov <- A*exp(-0.5*(tau/l)^2)
  return(acov)
}
```

Now we can use this to compute an ACV when we need it.

```r
A <- 1.0   # set the amplitude parameter
l <- 3.0   # set the scale parameter
tau <- seq(-25, 25, by = 0.25) # define lags
y <- acv(tau, A, l)            # compute ACV
plot(tau, y, type = "l")       # plot the result
l <- 0.75                      # change the scale
y.2 <- acv(tau, A, l)          # re-compute the ACV
lines(tau, y.2, col = "blue")  # overlay the new curve
```



Now let's define a vector of times $\mathbf{t} = \{t_1, t_2, \ldots, t_M\}$.

```
t <- c(1, 2, 4, 7.5, 8.0, 8.5, 9, 10)
```

We want to compute the covariance matrix using the autocovariance function (equation 15) and the list of times. We first make a matrix of all the time differences $\tau_{ij} = |t_i - t_j|$

```
  tau <- outer(t, t, "-")
  tau <- abs(tau)
  print(tau)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]  0.0  1.0  3.0  6.5  7.0  7.5  8.0  9.0
## [2,]  1.0  0.0  2.0  5.5  6.0  6.5  7.0  8.0
## [3,]  3.0  2.0  0.0  3.5  4.0  4.5  5.0  6.0
## [4,]  6.5  5.5  3.5  0.0  0.5  1.0  1.5  2.5
## [5,]  7.0  6.0  4.0  0.5  0.0  0.5  1.0  2.0
## [6,]  7.5  6.5  4.5  1.0  0.5  0.0  0.5  1.5
## [7,]  8.0  7.0  5.0  1.5  1.0  0.5  0.0  1.0
## [8,]  9.0  8.0  6.0  2.5  2.0  1.5  1.0  0.0
```

The R function `outer()` is very useful. It takes two vectors $\{x_i\}$ and $\{y_i\}$ and an operation and computes the operation for every pair of values. In this case the operation is simply "-" so it subtracts all the t_j from all the t_i and arranges the output in a matrix, i.e. $\tau_{ij} = |t_i - t_j|$.

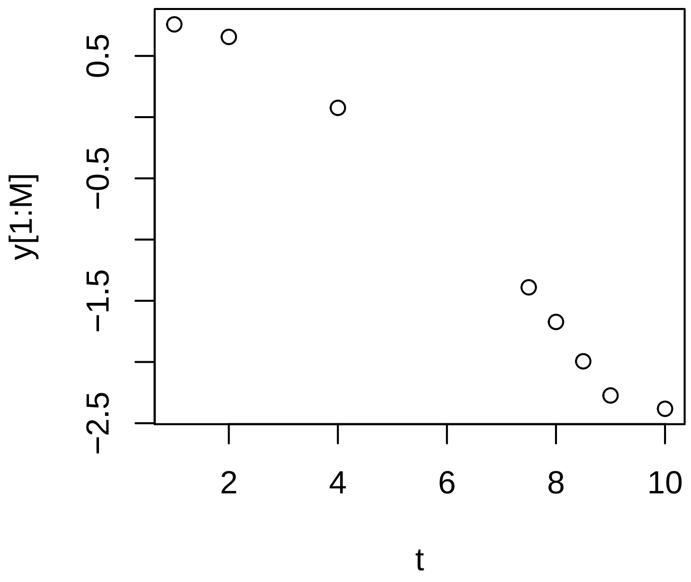
Now we can use our function `acv()` defined above to compute the ACV for every value of τ_{ij} in the matrix. This works because the function `acv()` will accept a matrix as input, i.e. $S_{ij} = ACV(\tau_{ij})$.

```
  S <- acv(tau, 1.0, 2)
  print(signif(S[1:5,1:5], 3))    # print only to 3 signif. fig.
```

```
##          [,1]   [,2]  [,3]    [,4]    [,5]
## [1,] 1.00000 0.8820 0.325 0.00509 0.00219
## [2,] 0.88200 1.0000 0.607 0.02280 0.01110
## [3,] 0.32500 0.6070 1.000 0.21600 0.13500
## [4,] 0.00509 0.0228 0.216 1.00000 0.96900
## [5,] 0.00219 0.0111 0.135 0.96900 1.00000
```

Now we can use this to compute a new Gaussian vector

```
  M <- length(t)                    # how many elements in the vector?
  mu <- array(0, dim=M)             # set all means to zero
  y <- mvtnorm::rmvnorm(1, mu, S) # generate random vector
  plot(t, y[1:M])
```



There is a more structure in these data. We can see this more easily if we evalute many more points. However, we may start to see numerical problems. If you repeat the above steps but first set

```
    t <- seq(0, 10, by = 0.2)
```

so that $\mathbf{t} = \{0, 0.2, 0.4, \ldots, 9.8, 10.0\}$ you may find that `rmvnorm()` fails to run. For example

```
  t <- seq(0,10,by=0.2)
  M <- length(t)              # how many elements in the vector?
  mu <- array(0, dim=M)     # set all means to zero
  tau <- outer(t, t, "-")
  tau <- abs(tau)
  S <- acv(tau, 1.0, 1)
  y <- mvtnorm::rmvnorm(1, mu, S)    # generate a new, random vector

## Warning in sqrt(ev$values):  NaNs produced
```

The matrix S that we made is not quite right. This because we are computing with a 51×51 matrix. Tiny numerical errors when computing or storing the matrix could mean it is not quite positive definite (so $\det S \leq 0$).

But in R we can add new matrix functions which allow us to fix these issues, and force S to be positive definite. We need a new package called `Matrix`. You may need to install it on your computer before you can use it.

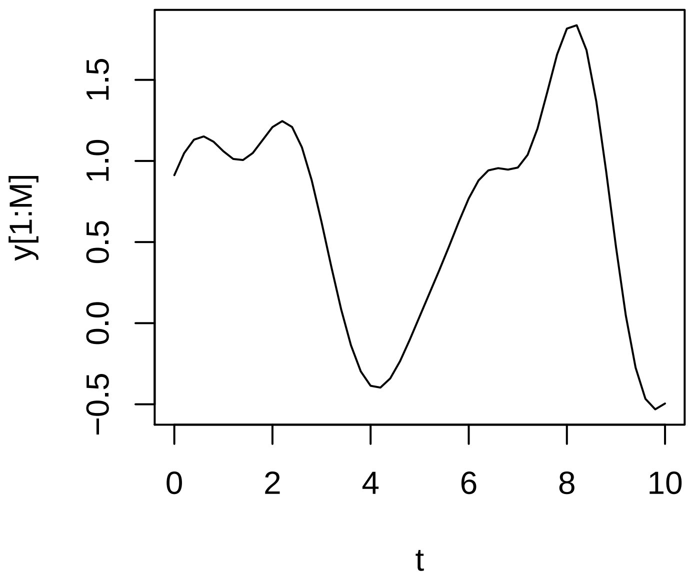
```
  install.packages("Matrix")
```

Now we can use a function called `nearPD()` to find the nearest positive definite matrix. If you read the instructions (`? Matrix::nearPD`) you will find the output is not in a simple matrix formal. So we need some extra lines to handle this, and convert it to a normal `matrix` object.

```
pd <- Matrix::nearPD(S)
if (pd$converged == FALSE) {
  stop('near PD failed to converge')
}
S <- matrix(pd$mat, nrow = M)
```

Now we have a positive definite S we can run `rmvnorm()` as before.

```
  y <- mvtnorm::rmvnorm(1, mu, S)    # generate a new, random vector
  plot(t, y[1:M], type = "l")
```



And we can re-run with more points if needed and it should still work. Although as $M > 1000$ and the S matrix gets very large ($10^3 \times 10^3$) we may find the numerical stability becomes a problem again.

Now, let's generate several vector outputs using a simple loop

```r
plot(0, 0, type = "n", xlim = c(0, 10), ylim = c(-3, 3),
     xlab = "time", ylab = "y(t)")  # open blank plot
for (i in 1:10) {
  y <- mvtnorm::rmvnorm(1, mu, S)    # generate a new, random vector
  lines(t, y[1:M], lwd = 2, col = rainbow(10)[i])
}
```

The figures show what happens when we repeat the above for $l = 0.2$ and $l = 5$.

There may be better ways to fix the numerical issues that arise when M becomes large. This would be good to study.

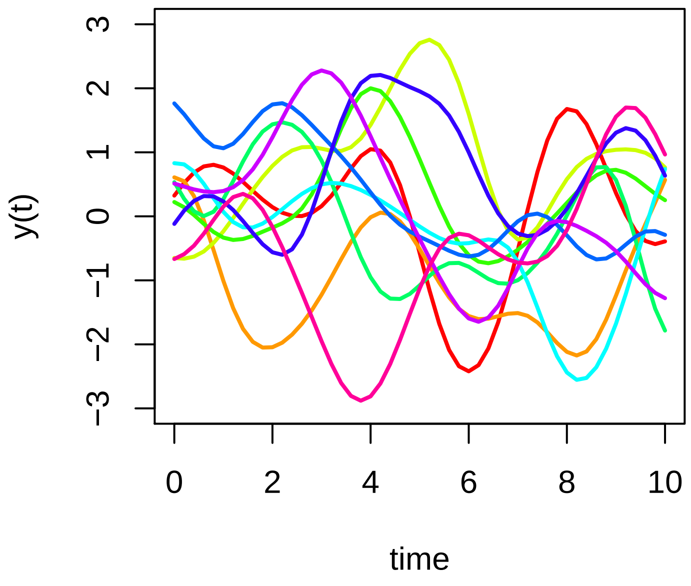


Figure 1: Example GP output with $l = 1$

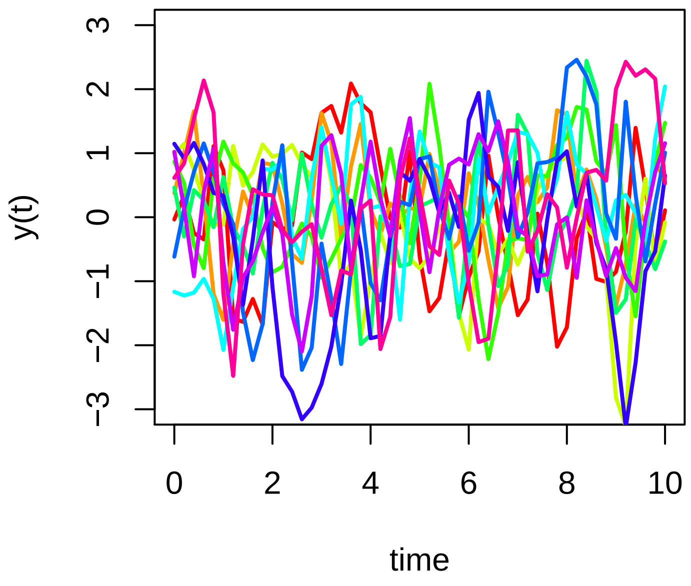


Figure 2: Example GP outputwith $l = 0.2$

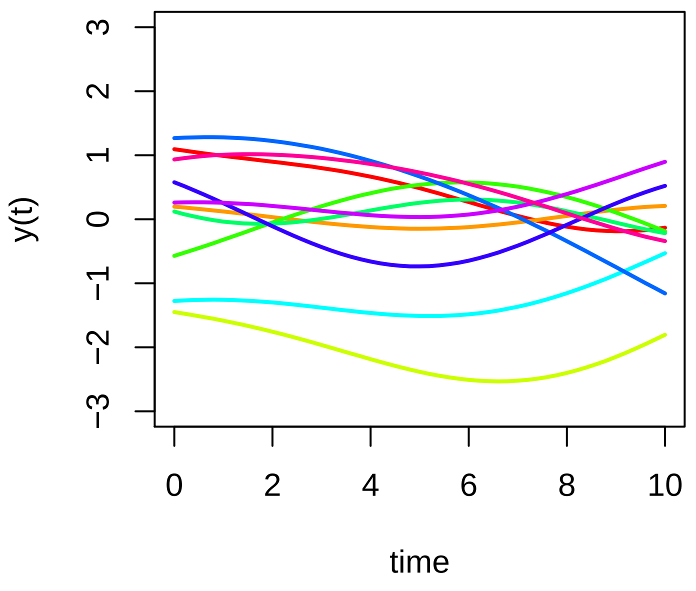


Figure 3: Example GP output with $l = 5$