Teaching Multivariable Math in DASC 2594

John Tipton

The University of Arkansas

2022/06/01 (updated: 2022-06-02)

The dasc2594 package

- Goals
  - Make it easier for students to apply concepts in the class
  - Make it easier to learn fundamental concepts
  - Make it easier for me to develop homework questions (see HW-06-fall-2021.Rmd for example)

The dasc2594 package -- Key functions

- Row operations
  - `row_add()`
  - `row_multiply()`
  - `row_swap()`
  - `rref()`
  - `
- Visualizations
  - `plot_change_basis()`
  - `plot_tangent_plane()`
  - `plot_transformation()`

The dasc2594 package -- Key functions

- Write matrices and systems of equations out to latex
  - `array_to_latex()`
  - `array_to_matrix_equation()`
  - `array_to_system()`
  - `array_to_vector_equation()`
  - `latex_equation_to_png()`
- Check matrix properties
  - `is_basis()`
  - `is_consistent()`
  - `is_invertible()`
  - `is_unique()`
  - `is_valid_row()`
- Generate random examples
  - `elementary_matrix()`
  - `make_basis()`
  - `make_eigen()`
  - `make_system_of_equations()`

Labs

- Making the concepts real and concrete
- Watch first 11 minutes on your own time https://www.youtube.com/watch?v=aVwxzDHniEw

Recall that we first introduced a basis using polynomials (e.g., the functions 1, x, and x^2 span the set of quadratic polynomials). We can extend this idea to fit smooth functions by expanding data using transformations of the input variables. Consider the data below which show a plot of sales over two years time (24 months) where the true trend process `trend` is assumed to be known.

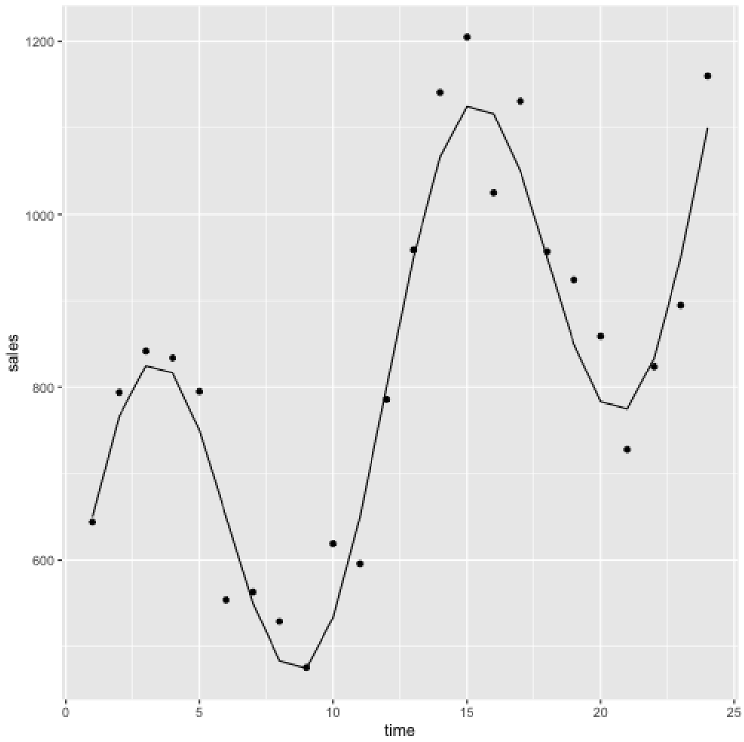
The basis

First, we load the data and generate a spline basis expansion of the variable `time`. This can be done with the `splines` library where the functional response will be a linear combination of the basis functions

```
library(splines)
library(ggfortify) # for the basis plotting
sales <- read_csv(here::here("data", "sales.csv"))
autoplot(bs(sales$time, df = 12))
```

The data

```r
sales <- read_csv(here::here("data", "sales.csv"))
p <- ggplot(sales, aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = trend))
p
```

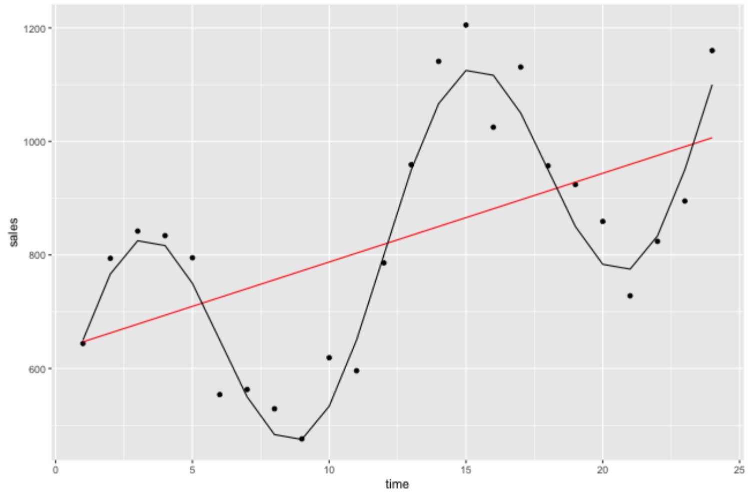


Fitting a model

For example, a linear model can be fit to the `sales_data` using

and the predictions from the linear model `mod` can be generated and plotted using `predict()` as

```
sales %>%
  mutate(pred_lm = predict(mod)) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = pred_lm), color = "red") +
  geom_line(aes(x = time, y = trend))
```



Fitting a model

Notice that the linear function does not capture the variation in seasonal trend in the data (although it does capture the long-term trend). Rather than forming a basis over a vector space, we will construct a basis over a more abstract function space where the dimension of the function space. This will give us a more intuitive understanding of a "basis". Because the basis of functions are just wiggles, as the dimension of the basis function space increases, the set of functions in the span of the basis increases (the possible functions that can be created get more and more flexible/"wiggly").

Fitting a model

The functional basis expansion we will use are called **b-splines** and can be created using the `bs()` function in `R` and the dimension of the function space is determined by the "degrees of freedom" parameter `df`. For example, the model with `df=4` can be fit and plotted and the predictions from the b-spline model `mod_bs` can be generated and plotted using `predict()` as

```
mod_bs <- lm(sales ~ bs(time, df = 4), data = sales)

sales %>%
  mutate(pred_bs = predict(mod_bs)) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = pred_bs), color = "red") +
  geom_line(aes(x = time, y = trend))
```

Examining the Math

Let's take a close look under the hood of these functions. First, lets looks the the **design matrix $\mathbf{X}$** which is created using 4 degrees of freedom using `df = 4` option in the b-spline function `bs()` that generates the b-spline basis.

```
X <- model.matrix(~ bs(time, df = 4), data = sales)
```

Just like with linear regression, the ordinary least squares solution is given by

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}\mathbf{y}$$

```
y <- sales$sales
beta_hat <- solve(t(X) %*% X) %*% t(X) %*% y
```

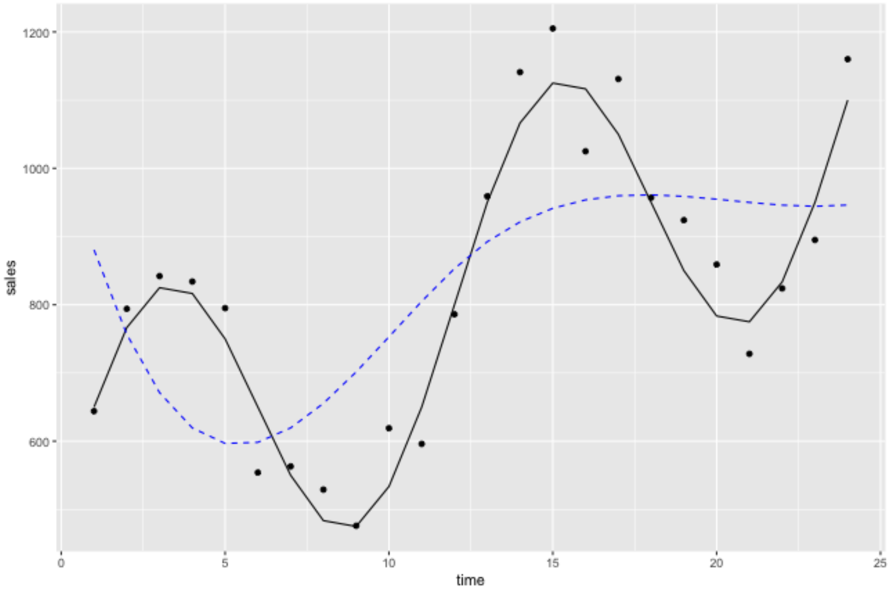
and the predicted values at the observed locations are given by $\mathbf{X}\hat{\boldsymbol{\beta}}$ with the fitted functional response shown below.

```
preds <- X %*% beta_hat
```

Examining the Math

Adding the fitted b-spline mean response to the data with `df=4` to the plot of the data in the ggplot object `p`.

```
p + geom_line(aes(time, y = preds), color = "blue", lty = 2)
```



Notice that the fitted response in blue is underfitted relative to the true mean response shown in black.

Fitting the Model

Rather than having to fit this "by hand," we can use the `lm()` function like so and save the fitted model to an object:

```
model_4 <- lm(y ~ X - 1)
```

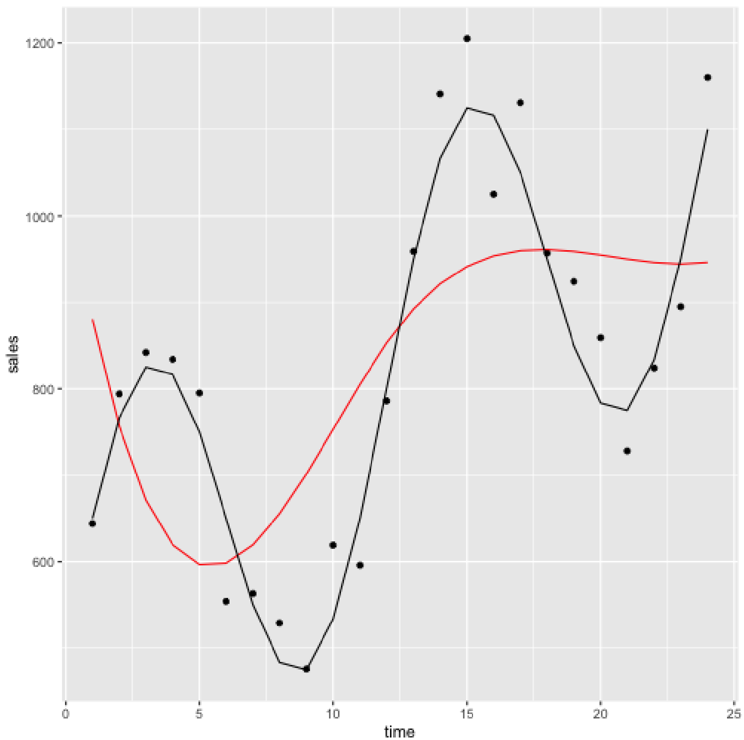
where the `-1` in the formula `y ~ X - 1` tells the `lm()` function that the y-intercept has already been included in the design matrix `X`.

To generate predictions from the `lm()` model, you can use the `predict()` function like so and save the variable in the object `preds` using

```
preds <- predict(model_4)
```

Fitting the model

```
sales %>%
  mutate(preds = preds) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = preds), color = "red") +
  geom_line(aes(x = time, y = trend))
```



Question 1: Part a

For this lab, change the degrees of freedom `df` to `df = 8` and `df = 18` and explore what happens as compared to `df = 4`. Describe what happens to the rank of the design matrix $\mathbf{X}$ and the how the rank influences the fitted response function.

1) To do this, first create 3 model matrices. First, create `X4` using `df=4` in the `model.matrix()` function, `X8` using `df=8` in the `model.matrix()` function, and `X18` using `df = 18` in the `model.matrix()` function.

2) Using the matrices `X4`, `X8`, and `X18`, determine the rank of each of these matrices using the `qr()` function and extracting the variable `rank`

3) Fit the model using `lm()` to each of the matrices `X4`, `X8`, and `X18` and save these as `mod_4`, `mod_8`, and `mod_18`.

4) Using the fitted models `mod_4`, `mod_8`, and `mod_18`, generate predictions `preds_4`, `preds_8`, and `preds_18` using the `predict()` function.

5) Plot the observed data in `sales` with the predicted lines from `preds_4`, `preds_8`, and `preds_18`. Make the line for `preds_4` red, `preds_8` blue, and `preds_18` orange.

Question 1: Part a (Solution)

```r
# df = 4, rank = 5
X4 <- model.matrix(~ bs(time, df = 4), data = sales)
qr(X4)$rank
```

```
## [1] 5
```

```r
mod_4 <- lm(y ~ X4 - 1)
preds_4 <- predict(mod_4)

# df = 8, rank = 9
X8 <- model.matrix(~ bs(time, df = 8), data = sales)
qr(X8)$rank
```

```
## [1] 9
```

```r
mod_8 <- lm(y ~ X8 - 1)
preds_8 <- predict(mod_8)

# df = 18, rank = 19
X18 <- model.matrix(~ bs(time, df = 18), data = sales)
qr(X18)$rank
```

```
## [1] 19
```

```r
mod_18 <- lm(y ~ X18 - 1)
preds_18 <- predict(mod_18)
```

Question 1: Part a (Solution)

```
sales %>%
  mutate(preds_4 = preds_4,
         preds_8 = preds_8,
         preds_18 = preds_18) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = preds_4), color = "red") +
  geom_line(aes(x = time, y = preds_8), color = "blue") +
  geom_line(aes(x = time, y = preds_18), color = "orange") +
  geom_line(aes(x = time, y = trend))
```

Question 1: Part b

Try increasing `df` to exactly equal the number of observations minus 1. Describe the fit of the model to the data. Generate the design matrix `X` and plot the predictions `preds` using this design matrix. Why would this not be a good idea to fit this model in practice? (think about how this model would fit new data not yet seen by the model)

Question 1: Part b (Solution)

```r
length(y)
```

```
## [1] 24
```

```r
# df = 24-1, rank = 24
X <- model.matrix(~ bs(time, df = 23), data = sales)
qr(X)$rank
```

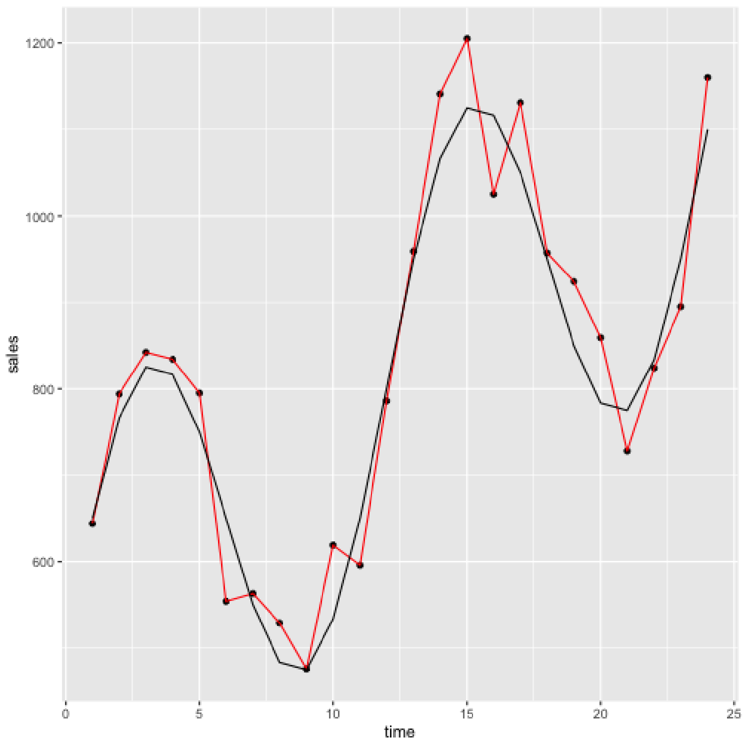
```
## [1] 24
```

```r
# generate new predictions
preds <- predict(lm(y ~ X - 1))
```

The model now fits the data perfectly! Because the number of free variables in the system of equation is equal to the number of equations (data points), there is a perfect fit. This would not be a good model because it will not fit new, unseen data well.

Question 1: Part b (Solution)

```
# plot new predictions
sales %>%
  mutate(preds = preds) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = preds), color = "red") +
  geom_line(aes(x = time, y = trend))
```



Question 1: Part c

What about increasing `df` to a number larger than the number of months (24) -- what happens when you fit the model with `lm` and get the parameter estimates for the coefficients $\hat{\beta}$? Knowing what you know about how the estimates $\hat{\beta}$ are being calculated, what explains this error?

Question 1: Part c (Solution)

```r
length(y)
```

```
## [1] 24
```

```r
# df = 24-1, rank = 24
X <- model.matrix(~ bs(time, df = 25), data = sales)
qr(X)$rank
```

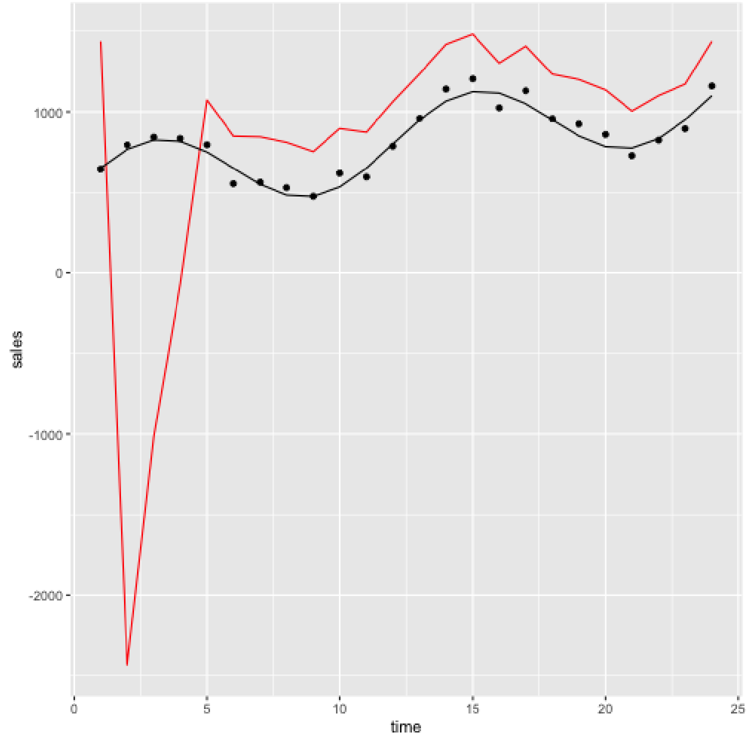
```
## [1] 24
```

```r
# generate new predictions
preds <- predict(lm(y ~ X - 1))
```

The model starts to fail dramatically. There are a number of free variables in the system of equations and these cause poor model fits

Question 1: Part c (Solution)

```
# plot new predictions
sales %>%
  mutate(preds = preds) %>%
  ggplot(aes(x = time, y = sales)) +
  geom_point() +
  geom_line(aes(x = time, y = preds), color = "red") +
  geom_line(aes(x = time, y = trend))
```



Question 1: Part d (Solution)

Think about $df = n - 1$ as a system of equations with n equations (rows) and df unknowns (columns) with each vector in the system of equations being linearly independent if $df < n - 1$. How many solutions does this system of equation have when $df = 10$? What about when `df = n`? Use the plots generated above (parts a-c) to answer this question.

Question 1: Part d (Solution)

```
X <- model.matrix(~ bs(time, df = 10), data = sales)
dim(X)
```

```
## [1] 24 11
```

```
qr(X)$rank
```

```
## [1] 11
```

```
# rref(X)
```

When `df=10`, there are no free variables and the rank of the design matrix is `df+1` which is equal to the number of columns of `X`. Thus, the columns of `X` are linearly independent (you could see this if you used `rref()` on the matrix `X` as there will be a pivot in every column. Thus the solution will be unique.

Question 1: Part d (Solution)

```r
X <- model.matrix(~ bs(time, df = length(y)), data = sales)
dim(X)
```

```
## [1] 24 25
```

```r
qr(X)$rank
```

```
## [1] 24
```

```r
# rref(X)
```

When `df = n`, there is a free variable and the rank of the design matrix is `n` which is not equal to the number of columns of `X` which is `n+1`. Thus, the columns of `X` are linearly dependent (you could see this if you used `rref()` on the matrix `X` as there will not be a pivot in every column. Thus the solution will be not be unique.