Linear Regression as a Linear Map

Companion notes for the "Regression Manifold" slideshow. The video unfolds in three movements — fitting a line, finding the model's range, and deriving the least-squares solution geometrically — with light click markers so you can follow along either while presenting or while reading.

Movement 1: Regression as matrix-vector multiplication

We start with three data points at $t = 0.3, 1.0, 1.7$, with observed values $y \approx 0.26, 2.72, 1.52$. Our goal is to characterize how these points were generated, so we can predict y for other values of t. A line $y = mt + b$ through the data — here the best fit turns out to be exactly $m = 0.9,\ b = 0.6$ — has specific values for m and b, but those aren't the only options: sliders underneath the axes let us choose any m and b we like, with the line updating live as we drag them.

Each choice of (m, b) produces three specific predictions — one per data point. Setting $m = 1.2,\ b = 0.3$ predicts $(0.66,\ 1.5,\ 2.34)$; a different choice, $m = 1.0,\ b = 1.0$, predicts $(1.3,\ 2.0,\ 2.7)$ instead. Each prediction is really a 3-D vector, so this is a map from the 2-D space of parameters (m, b) to a 3-D space of predictions — the sliders wander a bit (visiting $(0.4, 1.5)$ and $(1.5, 0.3)$ along the way) while a small parameter-space plot and a 3-D prediction-space plot track the current point in each space.

The three prediction equations,

$$0.3m + b = 0.26 \qquad 1.0m + b = 2.72 \qquad 1.7m + b = 1.52,$$

are each the same linear combination of t_i and 1 — writing that 1 explicitly (as $b(1)$) makes the pattern visible. That's exactly matrix-vector multiplication:

$$\begin{bmatrix} 0.3 & 1 \\ 1.0 & 1 \\ 1.7 & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} 0.26 \\ 2.72 \\ 1.52 \end{bmatrix}$$

The two columns are the two *basis functions* we guessed the data followed — t and the constant function 1 — evaluated at each t_i. This matrix, built directly out of the model we chose, is what turns a parameter guess into a prediction.

Movement 2: The range of the model

What does the full space of possible predictions look like? Filling parameter space with a grid of rainbow-colored (m, b) guesses (each with its own faint prediction line) and mapping every point over to 3-D prediction space, then rotating that space around, reveals something striking: even though prediction space is 3-D, every possible prediction lands on the same 2-D *plane*. No matter which (m, b) we pick, we can never predict anything outside that plane — this plane, stretched out to the full extent of the box, **is** the range of our model: the "model manifold." (The null space, by contrast, is trivial here — the only parameter choice that predicts all zeros is $m = b = 0$, since our two columns aren't parallel.)

Movement 3: Orthogonal projection and the pseudoinverse

Here's the catch: the *true* observed data point doesn't sit on that plane at all — no choice of m, b can reach it exactly, because real data has noise. If our notion of "best" is ordinary Euclidean distance, the closest we can get is the **orthogonal projection** of the true point onto the model manifold — visualized as a small translucent sphere centered on the true point, whose surface just touches the plane at the projected point y_m.

That's the same idea as the component decomposition from the other examples in this course: the true data vector y splits into a component *on* the model manifold (y_m, drawn in teal) and a component *perpendicular* to it (drawn in orange) — the part of the data the model structurally cannot capture, no matter the parameters.

Because that perpendicular component vanishes under A^T, applying A^T sends both y and its projection y_m to the same place:

$$A^T y_m = A^T y$$

Substituting $y_m = A\hat{x}$ (our best prediction, for the best parameters $\hat{x}$) and using that $A^T A$ is symmetric and positive definite — hence invertible — we can solve directly for $\hat{x}$:

$$A^T A\hat{x} = A^T y \qquad \Longrightarrow \qquad \hat{x} = (A^T A)^{-1} A^T y$$

This is the Moore–Penrose pseudoinverse, arrived at geometrically rather than by minimizing an error formula: it's just "map the observed point down to the parameters that produce its projection." Mapping the observed data vector through this formula sends it straight to the corresponding point in parameter space — which recovers our very first best-fit line. The Euclidean distance between the observed points and that line in prediction space is exactly the sum of squared error — left as an exercise for the reader.