Moving Beyond Linearity

A Complete Guide to Non-Linear Regression Methods

The Core Problem

Standard linear regression assumes:

$$y = \beta_0 + \beta_1 x + \varepsilon$$

This says: "For every 1-unit increase in x, y changes by exactly B1, *everywhere*."

But reality is rarely this simple. Consider:

* **Wage vs. Age**: Earnings rise quickly early in your career, plateau, then maybe decline. A straight line misses this entirely.
* **Dose-response curves**: Low doses do nothing, medium doses help, high doses might harm.

The goal: Capture non-linear patterns while keeping things interpretable and avoiding overfitting.

1. Polynomial Regression

Intuition

Instead of fitting a line, fit a curve. A quadratic (x squared) can capture 'U-shapes' or inverted U-shapes. A cubic (x cubed) can capture 'S-curves'.

The Mathematics

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \varepsilon_i$$

Key insight: This is *still* linear regression! You're just regressing on transformed features. Create new columns:

x	x^2	x^3
2	4	8
3	9	27
5	25	125

Then run ordinary least squares on this expanded matrix.

The Catch

High-degree polynomials (say d > 4) become **wildly unstable** at the boundaries. Imagine fitting a degree-10 polynomial - it might oscillate crazily near the edges of your data. This is called *Runge's phenomenon*.

2. Step Functions

Intuition

Why force a smooth curve at all? Just chop X into bins and estimate a separate average for each bin. Think of it like: "People aged 20-30 earn X on average. People aged 30-40 earn Y on average."

The Mathematics

Create cutpoints c1 < c2 < ... < cK and define indicator variables:

$$C_j(X) = \mathbf{1}(c_j \le X < c_{j+1})$$

Then fit:

$$y_i = \beta_0 + \beta_1 C_1(x_i) + \cdots + \beta_K C_K(x_i) + \varepsilon_i$$

Interpretation

* B0 = mean response in the baseline bin
* Bj = how much higher/lower bin j is compared to baseline

The Catch

You lose information within bins. If wages steadily rise from 30 to 40, the step function just says "everyone 30-40 earns the same." The "staircase" can look ugly and miss real trends.

3. Basis Functions (The Unifying Framework)

Both polynomials and step functions are special cases of:

$$y_i = \beta_0 + \sum_{k=1}^{K} \beta_k b_k(x_i) + \varepsilon_i$$

where b_k(.) are **basis functions** - fixed transformations of X.

Method	Basis Functions
Polynomial (degree 3)	b1(x) = x, b2(x) = x^2, b3(x) = x^3
Step function	bk(x) = 1 if ck <= x < c(k+1), else 0
Splines	More complex - see below

> **Why this matters**: Since it's linear in the Bk, all standard linear regression theory (standard errors, confidence intervals, F-tests) still applies!

4. Regression Splines

The Problem with Polynomials and Steps

* **Polynomials**: Global - changing data at one end affects the fit everywhere. Unstable at boundaries.
* **Steps**: Too rigid - discontinuous jumps look unnatural.

The Solution: Piecewise Polynomials + Smoothness

Fit separate low-degree polynomials in different regions, but **force them to connect smoothly**.

Building Up: From Piecewise to Splines

Step 1: Piecewise polynomials (no constraints)

Say we have one knot at c = 50. We fit a different cubic on each side.

Problem: 8 free parameters. The two cubics might not even meet at x=50! You get an ugly discontinuity.

Step 2: Add constraints

1. **Continuity**: Curves must meet at knots - removes 1 parameter per knot
2. **Smooth 1st derivative**: No 'kinks' - removes 1 more per knot
3. **Smooth 2nd derivative**: Curvature matches - removes 1 more per knot

> For a **cubic spline** with K knots: Degrees of freedom = K + 4

The Truncated Power Basis

A clever way to write a cubic spline with knots at specific locations:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \sum_{k=1}^{K} \theta_k (x - \xi_k)_+^3$$

where (x - xi_k)_+^3 equals (x - xi_k)^3 if x > xi_k, and 0 otherwise.

Intuition: Start with a global cubic. At each knot, "turn on" an adjustment that modifies the cubic behaviour from that point onwards.

Natural Splines: Taming the Boundaries

Problem: Splines can behave erratically beyond the outermost knots (extrapolation).

Solution: Force the function to be **linear** (not cubic) in the boundary regions.

> A natural cubic spline with K knots has **K degrees of freedom** (vs. K+4 for a regular cubic spline).

Choosing Knots

Where?

* Evenly spaced across the range of X
* At quantiles (e.g., 25th, 50th, 75th percentiles) - puts more knots where data is dense

How many? Use cross-validation to minimize CV error.

5. Smoothing Splines

A Different Philosophy

Regression splines: "Choose knot locations and number, then fit"

Smoothing splines: "Put a knot at every data point, but penalise roughness"

The Optimisation Problem

Find g(x) to minimise:

$$\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g''(t)]^2 \, dt$$

Breaking this down:

* g''(t) = second derivative = "curvature" at point t
* Integral of [g''(t)]^2 dt = total squared curvature = how "wiggly" the function is
* lambda = tuning parameter controlling the trade-off

The Role of lambda

lambda	Behaviour

lambda = 0	No penalty -> g interpolates every point -> massive overfitting
lambda -> infinity	Infinite penalty -> g''(t) = 0 everywhere -> g must be linear
Intermediate lambda	Balance between fit and smoothness

The Remarkable Solution

The function minimising this criterion is a **natural cubic spline with knots at every unique xi**.

Wait - n knots sounds like overfitting! But the penalty term **shrinks** the coefficients, effectively reducing complexity.

Effective Degrees of Freedom

$$df_\lambda = \sum_{i=1}^{n} S_{ii}(\lambda)$$

where S(lambda) is the 'smoother matrix'. When lambda = 0: df = n. When lambda -> infinity: df = 2.

Efficient Cross-Validation

For smoothing splines, LOOCV has a shortcut:

$$\text{LOOCV} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i - \hat{g}(x_i)}{1 - S_{ii}}\right)^2$$

> You only need to fit the model **once** to compute LOOCV - no need to refit n times!

6. Local Regression (LOESS/LOWESS)

Intuition

To predict at a point x0:

1. Look at nearby points
2. Fit a simple model (usually linear) using only those neighbours
3. Use that local fit for your prediction
4. Repeat for every x0 you care about

It's like having a "sliding window" that fits a fresh model wherever you need a prediction.

The Algorithm

1. **Choose span s** (fraction of data to use, e.g., s = 0.3 means 30% of points)
2. **For each target x0**: Find the k = floor(s*n) nearest neighbours. Assign weights (closest points get higher weights).
3. **Fit weighted least squares** to these points.
4. **Predict**: f(x0) = B0_hat + B1_hat * x0

The Span Parameter

Span s	Effect
Small (e.g., 0.1)	Very local -> wiggly, adapts to local patterns, high variance
Large (e.g., 0.9)	Almost global -> smooth, may miss local patterns, low variance

The Curse of Dimensionality

Critical limitation: Local regression fails for p > 3 or 4 predictors.

In high dimensions, "local" becomes meaningless. To capture even 10% of the data, you might need to span nearly the entire range of each variable.

7. Generalised Additive Models (GAMs)

The Problem

All methods so far handle **one predictor**. What about multiple predictors?

The Solution: Additivity

Assume the effects of different predictors **add up**:

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \varepsilon$$

Each fj can be a spline, polynomial, local regression - whatever captures the relationship between Y and Xj.

Example: Wage Model

$$\text{Wage} = \beta_0 + f_1(\text{Age}) + f_2(\text{Year}) + f_3(\text{Education}) + \varepsilon$$

* f1(Age): natural spline with 4 df
* f2(Year): natural spline with 4 df
* f3(Education): step function (education is categorical)

Fitting GAMs: Backfitting

You can't just run one regression - the fj's interact during estimation.

Backfitting algorithm:

1. Initialise: fj_hat = 0 for all j
2. Repeat until convergence: For j = 1, ..., p: Compute partial residuals, fit fj_hat to these residuals, centre the fit.

Intuition: Each step asks "Given the current fits for all other variables, what's the best fj?" Then update and repeat.

GAMs for Classification

For binary Y, model the log-odds:

$$\log\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right) = \beta_0 + f_1(x_1) + \cdots + f_p(x_p)$$

Pros and Cons

Advantages:

* Automatically captures non-linear effects - no manual transformation hunting
* Interpretable: plot each fj(Xj) separately to see how that variable affects Y
* Better predictions when relationships are genuinely non-linear

Limitations:

* Additivity is a strong assumption

* Misses **interactions**: if Age and Year interact, the additive model can't capture this
* Solution: Manually add interaction terms - but this reintroduces dimensionality problems

Summary: When to Use What

Method	Best For	Watch Out
Polynomial	Simple curves, few predictors	Boundary instability, global effects
Step Functions	Naturally categorical-like relationships	Loses within-bin information
Regression Splines	Smooth, interpretable curves	Need to choose knots
Smoothing Splines	Automatic smoothness, single predictor	Computationally heavier
Local Regression	Highly flexible curves, single predictor	Fails in high dimensions
GAMs	Multiple predictors, non-linear effects	Misses interactions

Quick Reference: Key Formulas

Polynomial Regression

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \varepsilon_i$$

Smoothing Spline Objective

$$\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g''(t)]^2 \, dt$$

GAM Model

$$y = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_p(x_p) + \varepsilon$$