# Natural-Spline-Based Generalized Additive Models
A Comparative Study for Energy Forecasting and Spinal BMD Prediction

Peilin Rao, Kelvin Luu, Jazlin Ong, Belle Ho, MJ Bailey

**Abstract**

Our project investigates a comprehensive application of spline fitting techniques within the framework of generalized additive models (GAMs) in two different areas; energy consumption forecasting, and spinal bone mineral density (BMD) prediction. Several spline fitting techniques are evaluated, including natural cubic splines with two knot selection strategies based on percentile and the K-means clustering algorithm. Furthermore, a penalized cubic smooth spline is considered that directly minimizes the residual error while enforcing smoothness. The performance of these methods is evaluated on the Southern California Energy Consumption dataset (January 2018 – January 2024) and the Spinal BMD dataset from the *Elements of Statistical Learning* and a synthetic dataset for algorithm benchmarking. From the experiments conducted, it was noticed that the percentile based knot selection method is relatively easy to implement but is susceptible to produce unstable and non-convergent results when data is skewed. On the other hand, the K-means based method and cubic smooth splines provide better representation of the data tendencies and produce better performance metrics and more understandable results. Even though the baseline predictions were robust in energy consumption modelling, the GAM was not able to explain all the variance in the data. These findings highlight the need to choose knots and regularize GAMs appropriately for large, real-world datasets.

## 1 Introduction

Generalized additive models (GAMs) are a very popular tool for describing nonlinear relationships in a wide range of applications. Their additive form is statistically efficient and easy to interpret, and they can be useful when the response variable has complicated structure or when there are complex interactions present. This report explores the use of spline methods in the context of GAMs for two essentially different predictive tasks: energy consumption forecasting and the assessment of spinal bone mineral density (BMD)

The energy consumption analysis is conducted on a large-scale dataset from Southern California which consists of data of more than 370,000 observations for six years and includes factors such as temperature, lighting, and building characteristics. At the same time, the Spinal BMD dataset contains a more specific view of the clinical data, which includes the information about the BMD of adolescents as well as the information about their demographics. These two datasets are quite different and both present situations in which it is important to identify slight changes in the nonlinear behaviour.

In our approach, we propose several spline fitting strategies. First, we propose natural cubic splines with knot placement at percentile markers, a heuristic that seeks to highlight the more populated regions of the data. Then we propose a different approach that uses K-means clustering to determine the knot locations dynamically to improve the responsiveness of the model to local phenomena. Finally, a penalized cubic smooth spline technique is employed to ensure that the model does not overfit the data, a practice that is usually recommended when fitting spline functions [1, 2]. The remaining of this report presents the data sets, the description of the backfitting algorithm for GAMs, the comparison of the results and the significance of the findings.

## 2 Dataset

The first dataset used in this project is the Southern California Energy Consumption dataset, obtained from Kaggle, containing electricity usage data from across Southern California from January 2018 to January 2024 [3]. It includes over 370,000 records detailing energy consumption in kWh by date, sector, temperature, lighting consumption, and other related factors. The dataset was provided in CSV format, and is publicly available under Kaggle's terms of use. The second dataset used in this project is the Bone Mineral Density (BMD) dataset, sourced from Elements of Statistical Learning. It contains relative spinal bone mineral density measurements of 261 North American adolescents, as well as age, gender, ethnicity, and id numbers to account for repeat measurements, as the data is from a longitudinal study. The dataset was provided in CSV format, and is from 1999. A synthetic, fixed, randomly generated dataset was also used to show performance of GAM backfitting.

## 3 Model and Methodology

### 3.1 GAM

A concise way to write a GAM is via its link function $g$ applied to the conditional mean of the response $Y$. If $Y$ belongs to an exponential-family distribution with mean $\mu$, we write

$$g(\mu) \;=\; \alpha \;+\; \sum_{j=1}^{p} f_j(x_j),$$

Here, $g$ is a known link function, $\alpha$ is the intercept, and each $f_j$ is a smooth function of the predictor $x_j$.

The fitting process of a GAM is roughly as follows. After initializing a guess of $\hat{\alpha}$ and the $f_i$, we iterate through each predictor variable $x_i$ in turn, updating the corresponding $f_i$ as we go. We do this by computing *residuals* for the current stage. The residuals essentially measure, for each data point, the portion of the target variable that is left unexplained after applying our model to the other predictor variables, excluding the one currently selected. Then, we update the $f_i$ by computing a "smoothing function" over the residuals. A detailed implementation of GAM backfitting can be found in Section 3.5.

We tested two methods of generating the smoothing function. The first uses the standard natural cubic spline. The second method of smoothing was to use optimization techniques to construct a cubic smooth spline minimizing some objective function. For the natural cubic spline only, we want to choose a subset of points to interpolate along to avoid issues. Therefore we tried two knot selection algorithms to choose the interpolating points. The first takes data at specific percentiles, and the other uses K-means to choose cluster centers.

We organize our discussion as follows. In Section 3.2, 3.3, and 3.4, we discuss the various spline and knot selection choices. In Section 3.5, we provide further detail for the backfitting algorithm for GAMs.

## 3.2 Natural cubic spline with percentile knot selection

The natural cubic spline is commonly used for interpolation. It is helpful for ensuring smoothness while preventing oscillatory behavior that arises from high-degree polynomials. Natural cubic spline interpolates all points that it is given exactly. However, using all the data points would not yield very good results, as computing residuals tend to be slightly noisy and clustered in the domain. To address this issue, we chose to use percentile-based knot selection, where a subset of data points is chosen at specific percentile markers as interpolation nodes. Our heuristic suggests that using percentiles would give precedence to points in clusters where modeling accurately would be more difficult.

More precisely, the *percentiles* are picked over the distribution of individual predictor variables. In order to work with data that takes on duplicate values in the predictor variable, we defer to the corresponding value in the target to break ties.

The methodology begins with sorting the dataset in lexicographic order to ensure a consistent selection of knots. The sorting is implemented using `numpy.lexsort`. Once sorted, we selected knot locations at specified percentiles to ensure that the chosen points reflect the overall data distribution. This allows our model to prioritize denser regions where more precise modeling is required.

In the implementation, the function `get_knots_percentile_method()` is used to determine the percentile-based knots. Given a data set $(x_i, y_i)$, it computes the corresponding indices at the chosen percentile values and returns the selected data points. The code for `get_knots_percentile_method()` is given in the appendix 2.

Once the knots are selected, the natural cubic spline coefficients were computed using a tridiagonal system while enforcing natural boundary conditions, using algorithm 3.4 given in the textbook [4, p. 147]. Each piece of the spline equation follows the standard form:

$$a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

where the coefficients are computed by solving the system. Our `natural_cubic_spline_coeff()` implements the coefficient calculation and ensures that the spline fits the selected knots smoothly while avoiding excessive complexity. The full code is in the appendix 1.

Below is a revised version of your section that adopts a more natural, academic tone and presents the pseudocode in a style more akin to the backfitting algorithm example you provided.

## 3.3 k-Means-Based Knot Selection for GAM Spline Fitting

In our approach, we use k-means clustering to choose the knot positions dynamically during spline fitting within the context of a generalized additive model (GAM). The idea is to have more spline basis functions where there are more observations, hence enabling the spline $f(x)$ to learn the nonlinear patterns in the residuals locally.

Formally, for a univariate feature $x_1, x_2, \ldots x_N$, the k-means algorithm partitions the data into

$k$ clusters by minimizing the within-cluster sum of squares (WCSS):

$$\min_{C_1,\ldots,C_k} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2,$$

where $\mu_i$ is the centroid of cluster $C_i$. These centroids are then used as candidate knot positions. To capture the local behaviour of the residuals we centre the spline we fit, we first aggregate the backfitting residuals within each cluster and then fit a natural spline to the resulting pairs $(\mu_i, \bar{r}_i)$. Finally, the fitted spline is centered by subtracting its average over the knot locations. The process is detailed in the pseudocode below.

**Algorithm 1** K-Means-Based Knot Selection for GAM Spline Fitting

```
1:  for each spline feature $j$ do
2:      Let $\mathbf{x}^{(j)} = \{x_{1j}, x_{2j}, \ldots, x_{Nj}\}$
3:      Determine the set of unique values $U = \text{unique}(\mathbf{x}^{(j)})$
4:      Set $n_{\text{clusters}} = \min(\text{max\_clusters}, |U|)$
5:      $[\boldsymbol{\mu}^{(j)}, \mathbf{z}] \leftarrow \text{KMeans}(\mathbf{x}^{(j)}, n_{\text{clusters}})$
6:      for each cluster $i = 1, 2, \ldots, n_{\text{clusters}}$ do
7:          Compute the average residual:
                $\bar{r}_i = \frac{1}{|C_i|} \sum_{x \in C_i} r(x)$
8:      end for
9:      Sort the centroids $\boldsymbol{\mu}^{(j)}$ and corresponding $\{\bar{r}_i\}$ in ascending order.
10:     Fit a natural spline $f_j$ to the data points $\{(\mu_i, \bar{r}_i)\}$.
11:     Center the spline:
                $f_j^{\text{centered}}(x) = f_j(x) - \frac{1}{n_{\text{clusters}}} \sum_{i=1}^{n_{\text{clusters}}} f_j(\mu_i)$
12: end for
```

This procedure leverages the inherent structure in the data to "vote" on the most informative knot locations. By adapting the density of knots to the local distribution of data points, the resulting GAM becomes more flexible and better able to model complex, nonlinear relationships. A more detailed python code is in appendix 3 .

### 3.4 Cubic smooth spline

In energy consumption modeling, our goal is to capture key usage patterns while avoiding overfitting to small fluctuations. We achieve this by penalizing overly "wiggly" solutions, ensuring the model strikes a balance between accurate fit and smoothness.

### Penalized Residual Sum of Squares (PRSS)

A well-known approach is to minimize the Penalized Residual Sum of Squares: [2]

$$\mathrm{PRSS}(\alpha, f_1, \ldots, f_p) = \sum_{i=1}^{N} \left( y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij}) \right)^2 + \sum_{j=1}^{p} \lambda_j \int \left( f_j''(t) \right)^2 \, dt, \tag{1}$$

where:

- $\alpha$ is an overall intercept;
- $f_j$ are component functions for each predictor $x_{ij}$;
- $\lambda_j \geq 0$ control how strongly we penalize curvature in each $f_j$.

Solving this penalized problem yields an additive cubic smooth spline model—each $f_j$ is a cubic spline that remains sufficiently flexible while avoiding excessive oscillations.

### Natural vs. Penalized Cubic Splines

- **Natural Cubic Splines:** Provide exact interpolation—the spline passes through every data point. This can overfit noisy or large datasets, causing excessive "wiggles" between points.
- **Penalized Cubic Splines (GAM):** Incorporate a smoothing penalty on the spline's curvature. This reduces overfitting and yields a more interpretable trend [1, 2].

### Simple Example using Tips.csv

To illustrate these concepts, we use a simple real-world dataset called `tips.csv` from the Seaborn-Data GitHub repository [5]. It contains restaurant tipping details such as `total_bill`, `tip`, and `day`. For our illustration, `total_bill` is treated as a numeric predictor, `day` as a categorical factor, and `tip` as the outcome.

A Generalized Additive Model (GAM) is then fit with penalized cubic splines for `total_bill` and a factor term for `day`:

```
gam = LinearGAM(s(0) + f(1)).fit(X, y)
```

This setup penalizes overly wiggly behavior in the spline, producing a smoother function to show how `tip` changes with `total_bill` across different days—effectively balancing flexibility and stability without overfitting.

### Resulting Plot

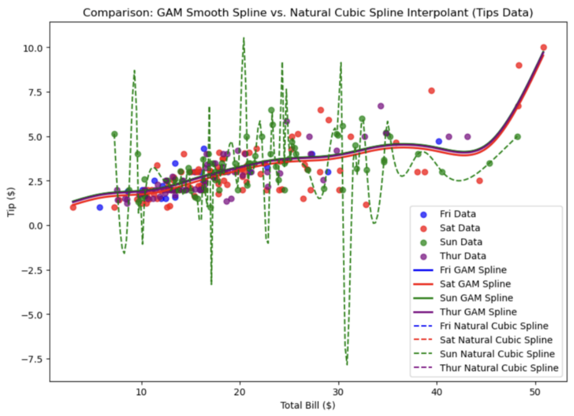


Figure 1

From the resulting plot as shown in figure 1:

- **Natural cubic spline (dashed)**: It passes exactly through each data point. With noisy or plentiful data, it can become extremely “wiggly” or exhibit large fluctuations between nearby points.
- **Penalized GAM spline (solid)**: It smooths out the data, capturing a more general trend without large oscillations. This is because pygam applies a smoothing penalty to avoid overfitting random noise.

Hence, we see that in scenarios with many (and possibly noisy) observations, the penalized spline approach typically offers a more stable, interpretable shape than the un-penalized natural cubic interpolation.

This simple tips.csv example highlights how penalized splines can outperform exact natural cubic splines in the presence of noisy or abundant data.

## 3.5 Backfitting

Generalized additive model (GAM) assumes an additive structure where the dependent variable is modeled as:

$$Y = \alpha + \sum_{j=1}^{p} f_j(x_j)$$

To estimate these functions, we used the backfitting algorithm, which is an iterative method that updates each feature’s function sequentially while keeping the others fixed.

The backfitting algorithm first initializes parameters. $\alpha$ is a global intercept that is set as the mean of $Y$, and all feature functions $f_j$ are initially set to zero. In each iteration, the residuals are computed by subtracting the effects of all other features.

For features that were modeled using splines, the algorithm selects the knots using K Means clustering. This ensures that the smoothing function is based on representative points rather than arbitrary fixed points. Then, the selected knots are used to fit a natural cubic spline. The spline is adjusted to ensure that its mean value over the knots is zero, which prevents bias assumption across multiple iterations.

Convergence condition is defined as when the maximum change in function values falls below a defined threshold. The iterative update continues until convergence.

The whole process is described by the algorithm 2

**Algorithm 2** Backfitting Algorithm for Additive Models

```
1: Initialize: $\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i$; $\hat{f}_j \leftarrow 0$ for all $j$
2: repeat
3:     for $j = 1, 2, \ldots, p$ do
4:         $\hat{f}_j \leftarrow S_j \left[ y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik}) \right]_{i=1}^{N}$
5:         $\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^{N} \hat{f}_j(x_{ij})$
6:     end for
7: until convergence: $\max_j \|\hat{f}_j^{\text{new}} - \hat{f}_j^{\text{old}}\| < \epsilon$
```

## 3.6 Workflow on Energy Consumption dataset with Cubic Smooth Spline

### Data Loading & Preprocessing

The script reads our dataset and extracts a timestamp column to generate time-based features (Year, Month, Week, Day). Certain features that can grow exponentially (e.g., `Energy Price`, `Building Age`) are log-transformed (e.g., `log_Energy Price ($/kWh)`).

### Continuous Feature Transformations

Many numeric predictors are marked for splines or linear effects. Others, such as `Temperature (°C)` or `Month`, are designated as "fourier". Their `sin/cos` expansions are created (with possible second harmonics) to capture cyclical patterns.

### Categorical Features

Features like `Building Type` or `Power Outage Indicator` are treated as factors in the GAM, resulting in separate intercepts for each category. The `Year` variable is handled as a linear effect, capturing a potential simple upward/downward trend over time.

### Design Matrix Construction

The code systematically combines these transformed columns (Fourier expansions, log transforms, splines, etc.) into a single DataFrame `X`. Categorical features are converted into integer codes so that `pyGAM` can treat them as factor terms.

### Building the GAM

Each column in `X` is assigned a term: `s(i)` for splines, `l(i)` for linear effects, and `f(i)` for factor terms. These terms are combined into a `TermList`, forming a `LinearGAM` that models `Energy`

`Consumption (kWh)` as the sum of smooth and/or linear contributions from each feature.

**Training and Evaluation**

The dataset is split into training and testing sets. A custom list of smoothing penalties (`lam`) is set: splines receive a lower penalty (e.g., 10.0) for more flexibility, while linear or factor terms receive a higher penalty (e.g., 50.0) to limit overfitting. After fitting the GAM, the code prints the $R^2$ value and other metrics (MAE, RMSE) on the test set, along with a model summary. It also produces residual plots and predicted versus actual plots.

**Feature Selection & Simplified GAM**

The script demonstrates the use of `Lasso` to identify the features with the strongest signals. It then refits a smaller GAM on the subset of selected features, thereby improving interpretability (fewer terms) and illustrating how the model can be refined.

## 4 Results and Discussion

### 4.1 Performance comparison on Spinal BMD dataset

We evaluated the performance of different spline methods on the Spinal BMD dataset. The two variables we used in the data set were the age, gender and spinal bone mineral density (BMD) measurements. We used age and gender to predict the BMD values. The goal was to assess how well each method captured the trends in the data while reducing overfitting in clustered regions and underfitting in sparse areas. The three methods compared are natural cubic spline with percentile knot selection (§3.2), natural cubic spline with K-means knot selection (§3.3), and the cubic smooth spline (§3.4).

First, we remark that in practice the percentile knot selection can lead to residual blow-up and non-convergence if the knots are spaced too far apart. This occurs primarily when the distribution is skewed. [maybe image of one of the steps]. This problem is not alleviated by decreasing the percentile distance, since this will increase the number of points interpolated and cause the very issues we were trying to prevent.

For the Spinal BMD dataset, the percentile knot method converges, but the end splines fluctuate and do not fit the data well at all visually. It is shown in figure 2

This is corroborated by the bad $R^2$ scores of $-3.121$ for the male data and $-1.422$ for the female data. The mean absolute error (MAE) for both is around 0.08 which is $\sim 25\%$ of the range of the BMD values. Overall, this method is not good.

Our guess for the problem with the percentile knot selection is as follows. The data is fairly noisy but evenly distributed. In particular, the percentile knots are roughly evenly spaced along the Age domain, which mitigates the spline blow-up issue. The problem is that because of the noise, the point selection process at each of these evenly distributed points tends to fluctuate between large and small BMD values and residuals. A good fit would require focusing on points roughly following the shape of the data. The percentiles fail to take this into account.

In comparison, the natural cubic splines with K-means knots and the cubic smooth splines both fit the shape of the data fairly well, as shown in figure 3. In terms of summary statistics, both do comparably: K-means has an $R^2$ of 0.156 (male) and 0.442 (female) compared to the smooth

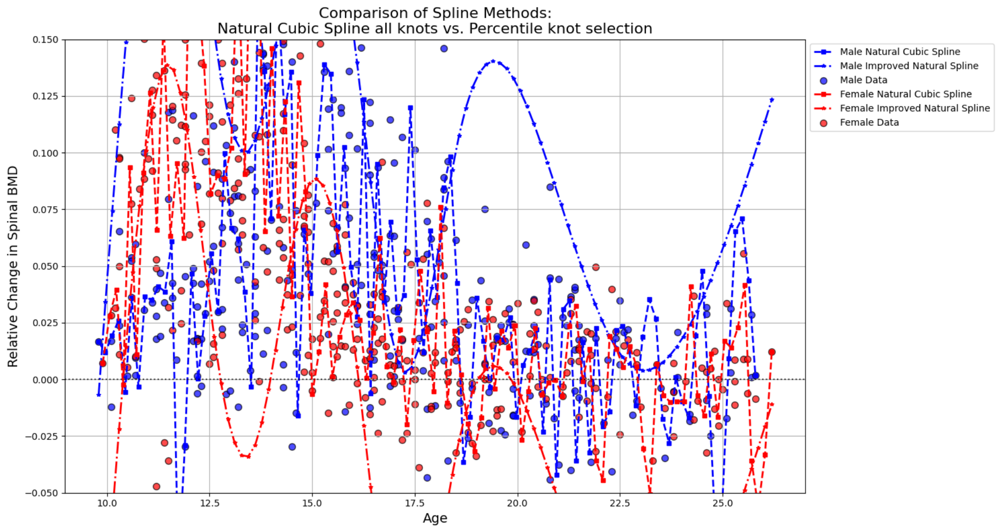


Figure 2: Comparison of natural cubic spline with percentile knots.

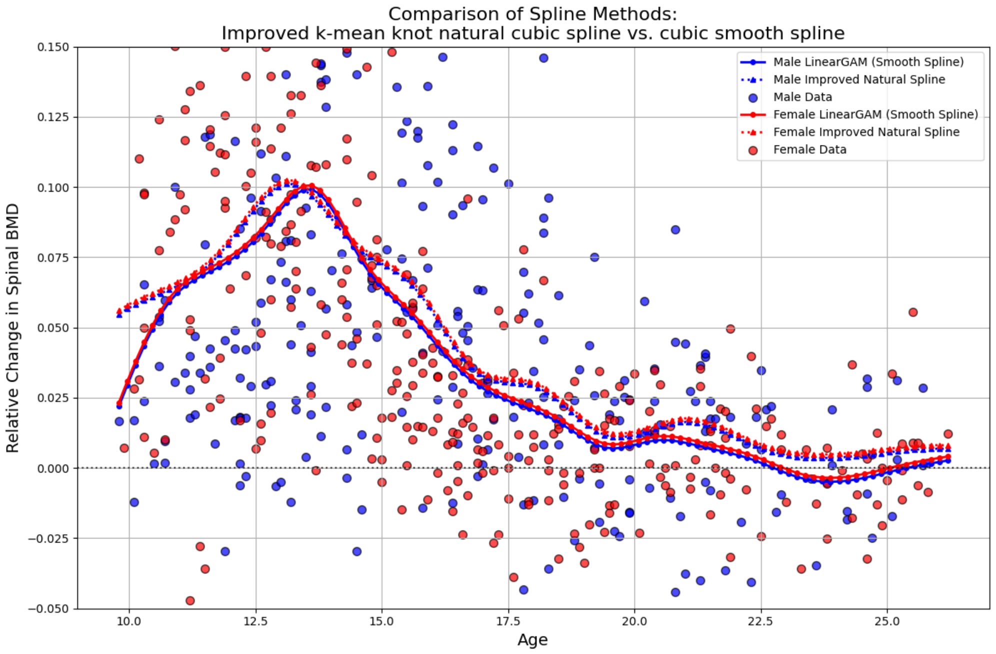


Figure 3: Comparison of cubic smooth spline with k-mean knots spline.

splines 0.181 (male), 0.468 (female). In terms of MAE, K-means yields 0.039 (male) and 0.035 (female). Cubic smooth splines yield 0.038 (male) and 0.033 (female).

If our guess for why the percentile knots did so poorly is correct, then the reason why the K-means selection does much better is because the choice of cluster centers correlates better with overall shape. Moreover, the cubic smooth spline forgoes knot choices altogether and performs the best overall because it directly minimizes the residuals of all data points, not just a subset (up to some smoothness constraints). The function updates in this case are somehow "optimal".

## 4.2 Performance on energy consumption prediction

**Evaluation of the Full GAM:**

- $\mathbf{R^2}$ **Score:** $\approx -0.0042$.
- A negative $R^2$ indicates the model's predictions perform slightly worse than a simple mean-based baseline on the test set.
- **Pseudo R-Squared from gam.summary():** $\approx 0.0043$.
- This similarly suggests the model explains only around 0.43% of the outcome variance, which is extremely low.
- Effective DoF is high for many terms, indicating the model is fairly complex.
- Warnings about p-values in penalized models highlight that these significance tests are not strictly valid when smoothing parameters are estimated.

**Interpretation**

- **Predictive Power:** Despite many features (spline or otherwise) and carefully chosen transformations (Fourier/log), the model does not capture substantial variance in Energy Consumption on the test data.
- **Possible Over-Complexity:** The large number of terms with modest or negligible effect on the outcome can lead to an overall poor fit.
- **Regularization:** Even though each term is penalized, the final model's complexity may still be too high or the penalty levels insufficiently tuned.

Therefore, a negative $R^2$ and near-zero pseudo $R^2$ underscore the difficulty of modeling the target variable with the existing features and settings.

### 4.2.1 Resulting Plots

#### Partial Dependence Curve Near Zero

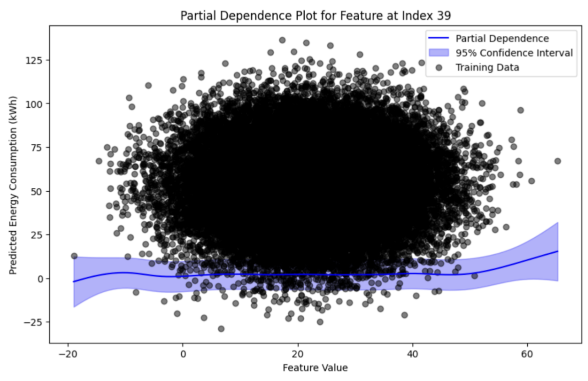


Figure 4

The solid blue line (the model's partial dependence) hovers close to zero over almost the entire range of this feature. That generally means the GAM is estimating little net impact of this variable on predicted energy consumption—even when the feature's value changes.

- **Relatively Wide Confidence Bands:** The light blue shaded region is the 95% confidence interval; it remains fairly wide, indicating uncertainty in the model's estimate of how this feature affects consumption. In particular, near the right edge (feature values $> 50$), the confidence interval expands sharply—often a sign of fewer data points in that region, causing more uncertainty.
- **Data Cloud:** The black points represent the actual training data plotted against the same feature dimension. Notice how they cluster into a broad, dense cloud with large variation in $y$ (energy consumption). Because the partial dependence line is nearly flat while actual consumption spans from below zero to above 100, the model is essentially attributing very little systematic effect from this feature alone.
- Possible Interpretation: If you expected a strong relationship here, the near-flat partial dependence suggests the model is not finding one (or at least, not a clear one). The wide confidence interval and massive data spread imply the model is not very confident in any precise relationship.

In short, the partial dependence plot demonstrates a near-zero effect (and a good deal of uncertainty) for this particular feature, which is consistent with the poor overall predictive power the GAM showed.

## Predicted vs. Actual Values

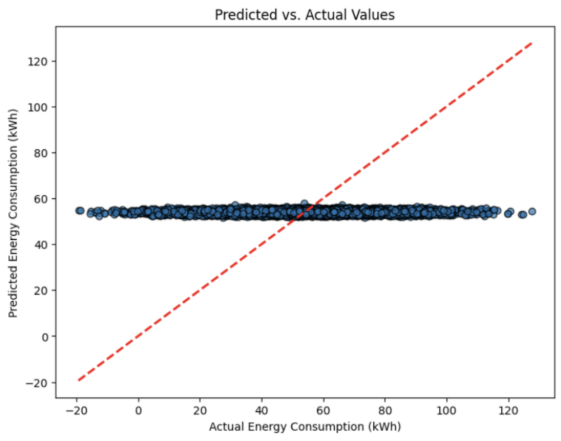


Figure 5

- **Visual Description:** Almost all predictions (y-axis) cluster around $\sim$ 50–55 kWh, while the actual energy consumption (x-axis) spans a wider range (roughly $-20$ to 120 kWh). The diagonal red line ($y = x$) is where perfect predictions would lie.
- **Interpretation:** The model outputs are relatively constant and do not vary in proportion to the actual values. Predictions fail to track the true range of energy consumption, reflected in a near-horizontal band. This explains the low $R^2$: the model does not differentiate high from low consumption effectively.

### Conclusion on Energy Consumption Prediction Performance

From the presented graphs and metrics, it is clear that the model struggles to capture the substantial variability in actual energy consumption. The very low (even negative) $R^2$ scores and the near-horizontal predictions indicate that, for individual point forecasts, the model does not distinguish high-usage from low-usage scenarios effectively.

However, despite these limitations, the model still provides a consistent, stable baseline estimate around the mean usage (roughly 50–55 kWh). In practical settings, this can be valuable as an initial or fallback reference:

- **Operational Planning:** Even if the model cannot finely differentiate at the extremes, its stable baseline may be sufficient for certain daily operational decisions, budgeting, or resource allocation where rough average estimates suffice.
- **Baseline for Further Improvement:** Because the residuals are roughly symmetric around zero, there is no systematic bias; the current model can serve as a foundation for iterative improvements (e.g., adding more relevant variables, tuning hyperparameters, or experimenting with alternate modeling approaches).
- **Risk Assessment:** The wide distribution of residuals highlights the inherent noise or missing information in the data; acknowledging this can help guide risk mitigation strategies (e.g., over-allocation of capacity, scheduling maintenance checks).

In short, while the model does not excel at precise, individualized consumption forecasts, it still offers a useful average-level prediction that may support certain high-level planning tasks.

## 5 Conclusion

In conclusion, this study shows that in the application of generalized additive models; spline methodology and knot selection is important when dealing with complex datasets. Our findings show that natural cubic splines with percentile-based knot selection could be unstable and not converge, especially in the presence of noisy or skewed data. On the other hand, the knot selection by the K-means clustering and the cubic smooth spline methods gave better and more stable models. Specifically, the cubic smooth spline method achieved a near optimal balance between model flexibility and regularization by directly minimizing the penalized residual sum of squares. The GAM applied to the energy consumption dataset was a stable baseline predictor around the mean but failed to capture much of the variability in the target variable due to the high noise and high dimensionality of the data. In general, our findings suggest that appropriate feature transformation, knot selection, and regularization are important in GAMs. Future work should aim to improve these methods by employing more sophisticated regularization strategies and including more predictive variables.

## 6 Presentation and Github

Presentation Link
Github link

## References

[1] Simon N. Wood. *Generalized Additive Models: An Introduction with R.* Chapman and Hall/CRC, 2006.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2 edition, 2009.

[3] Caleb. Southern california energy consumption. `https://www.kaggle.com/dsv/9712788`, 2024.

[4] Richard L. Burden, J Douglas Faires, and Annette M. Burden. *Numerical analysis.* Cengage Learning, tenth edition. edition, 2016.

[5] Michael Waskom. Seaborn-data/tips.csv at master · mwaskom/seaborn-data. `https://github.com/mwaskom/seaborn-data/blob/master/tips.csv`. Accessed 21 Mar. 2025.

## A Python Code for Natural Cubic Spline

The following code was used to compute and evaluate a natural cubic spline:

```python
def natural_cubic_spline_coeff(x, y):
    """
    Compute natural cubic spline coefficients.
    Returns arrays: a, b, c, d for each interval and the knots.
    """

    # Remove duplicates by taking unique x and corresponding y (or average
        duplicates if needed)
    unique_x, idx = np.unique(x, return_index=True)
    x = unique_x
    y = y[idx]


    n = len(x) - 1
    h = np.diff(x)
    A = np.zeros((n+1, n+1))
    b_vec = np.zeros(n+1)
    # Natural boundary conditions
    A[0, 0] = 1
    A[n, n] = 1
    for j in range(1, n):
        A[j, j-1] = h[j-1]
        A[j, j]   = 2 * (h[j-1] + h[j])
        A[j, j+1] = h[j]
        b_vec[j]  = 3 * ((y[j+1] - y[j]) / h[j] - (y[j] - y[j-1]) / h[j
            -1])
    c = np.linalg.solve(A, b_vec)
    b_coeff = np.zeros(n)
    d_coeff = np.zeros(n)
    a_coeff = y[:-1]
    for j in range(n):
        b_coeff[j] = (y[j+1] - y[j]) / h[j] - h[j]*(2*c[j] + c[j+1])/3
        d_coeff[j] = (c[j+1] - c[j]) / (3*h[j])
    return a_coeff, b_coeff, c[:-1], d_coeff, x

def evaluate_spline(x_eval, knots, a, b, c, d):
    """
    Evaluate the natural cubic spline at points x_eval.
    """
    x_eval = np.atleast_1d(x_eval)
    y_eval = np.zeros_like(x_eval)
    for i, xi in enumerate(x_eval):
        j = np.searchsorted(knots, xi) - 1
        j = np.clip(j, 0, len(a)-1)
        dx = xi - knots[j]
        y_eval[i] = a[j] + b[j]*dx + c[j]*(dx**2) + d[j]*(dx**3)
    return y_eval

def natural_spline(data):
    """
    Fits a natural cubic spline to 2D data (first column: x, second: y)
    and returns a function that evaluates the spline.
    """
    x = data[:, 0]
    y = data[:, 1]
    order = np.argsort(x)
    x_sorted = x[order]
    y_sorted = y[order]
    a, b, c, d, knots = natural_cubic_spline_coeff(x_sorted, y_sorted)
    def spline_func(x_val):
        return evaluate_spline(x_val, knots, a, b, c, d)
    return spline_func
```

Listing 1: Natural cubic spline implementation in Python

## B Python Code for percentile knot selection spline

The following code was used to compute and evaluate a natural cubic spline with percentile knot selection:

```
def get_knots_percentile_method(x, y, percentile):
    """
    Select knot indices for each feature based on the given percentile.
    """
    assert percentile > 0 and percentile <= 100
    N = len(x)
    num_features = x.shape[1]
    sorted_features_idx = [
        np.lexsort((y.reshape(-1,), x[:, i].reshape(-1,))) for i in range(
            num_features)
    ]
    split_idx = [0]
    current = 0
    step_size = int(np.ceil(N * (percentile / 100)))
    while current + step_size < N:
        current += step_size
        split_idx.append(current)
    if split_idx[-1] != N-1:
        split_idx.append(N-1)
    split_idx = np.array(split_idx).astype(int)
    knots_features_idx = [ idx[split_idx] for idx in sorted_features_idx ]
    # Remove duplicate x values for each feature
    for feature, idx in enumerate(knots_features_idx):
        x_idx = x[idx, feature]
        non_dupe_idx = list(dict(zip(x_idx, idx)).values())
        knots_features_idx[feature] = non_dupe_idx
    return knots_features_idx



def fit_gam_natural_spline_percentile_method(x, y, verbose=False):
    """
    Fits an additive model (GAM) using natural cubic splines with
    selected knots and a backfitting algorithm.
    Returns a function that evaluates the fitted GAM.
    """
    # Ensure y is 2D
    if not isinstance(y, np.ndarray):
        y = np.array(y)
    if len(y.shape) == 1:
        y = y.reshape(-1, 1)

    # Constants for backfitting
    THRESHOLD = 1e-5  # Convergence threshold
    KNOT_PERCENTILE = 10  # Use 10th percentile for knot selection
    N = len(y)
    num_features = x.shape[1]

    alpha_hat = np.average(y)  # global intercept
    # Initialize spline functions to zero for each feature
    f_i = [np.vectorize(lambda a: 0) for _ in range(num_features)]

    # Select knot indices for each feature
    knots_features_idx = get_knots_percentile_method(x, y, KNOT_PERCENTILE
        )

    count = 0
    error = np.inf
    while error > THRESHOLD or count == 0:
        feature = count % num_features
        knots_idx = knots_features_idx[feature]
        # Compute residuals excluding the current feature's effect
        other_effect = np.sum(
            np.array([f_i[k](x[knots_idx, k]) for k in range(num_features)
                if k != feature]),
            axis=0
        ).reshape(-1, 1)
        residuals = y[knots_idx] - alpha_hat - other_effect
        # Prepare data for spline fitting: first column is x, second is
            residuals
        datapoints_feature = np.hstack([x[knots_idx, feature].reshape(-1,
            1), residuals])
        new_f_feature = natural_spline(datapoints_feature)

        # Center the spline function so that its average over the knots is
             zero
        def get_new_f_feature_normalized(f, data, knots, feature):
            def new_f_feature_normalized(datapoints):
                return f(datapoints) - np.average(f(data[knots, feature]))
            return np.vectorize(new_f_feature_normalized)
        new_f_feature_normalized = get_new_f_feature_normalized(
            new_f_feature, x, knots_idx, feature)

        # (Optional) Plot intermediate spline fits if verbose is True.
        if verbose:
            x_vals = np.linspace(np.min(x[:, feature]), np.max(x[:,
                feature]), num=50)
            plt.figure()
            plt.plot(x_vals, new_f_feature(x_vals), 'g', label='new_spline
                ')
            plt.plot(x_vals, new_f_feature_normalized(x_vals), 'r', label=
                'new_spline_normalized')
            plt.plot(x_vals, f_i[feature](x_vals), 'b', label='prev')
            plt.scatter(x[knots_idx, feature], residuals, c='r', label='
                nodes')
            plt.title(f"Feature {feature} Update")
            plt.legend()
            plt.show()

        error = np.max(np.abs(new_f_feature_normalized(x[knots_idx,
            feature]) - f_i[feature](x[knots_idx, feature])))
        f_i[feature] = new_f_feature_normalized
        count += 1

    def gam_function(x_eval):
        # x_eval should be a 2D array with shape (num_points, num_features
            )
        return alpha_hat + np.sum(np.array([f_i[i](x_eval[:, i]) for i in
            range(num_features)]), axis=0)
    return gam_function


percentile_model = fit_gam_natural_spline_percentile_method(X, y, verbose=
    True)
```

Listing 2: Percentile knot selection implementation in Python

## C Python Code for k-mean knot selection spline

The following code was used to compute and evaluate a natural cubic spline with k-mean knot selection:

```
def k_mean_spline(x, y, feature_types, verbose=False):
    """
    Fits an additive model (GAM) of the form:
        y = alpha + sum_{j} f_j(x_j)
    where each f_j is either a natural spline or a linear function.

    Parameters:
        x : 2D numpy array, shape (N, num_features)
        y : 1D or 2D numpy array, shape (N,) or (N, 1)
        feature_types : list of strings of length num_features
                        each element should be 's' (spline) or 'l' (linear
                            )
        verbose : bool, if True, intermediate plots are shown
```

```
Returns:
    gam_function: callable that takes x_eval (2D array of shape (
        num_points, num_features))
                  and returns the fitted response.
"""
# Ensure y is 2D
if not isinstance(y, np.ndarray):
    y = np.array(y)
if len(y.shape) == 1:
    y = y.reshape(-1, 1)

THRESHOLD = 1e-5  # Convergence threshold for backfitting updates
KNOT_PERCENTILE = 10  # Used to determine number of clusters for
   spline features
N, num_features = x.shape

# Global intercept: the overall average of y.
alpha_hat = np.average(y)

# Initialize function estimates for each feature to be zero.
# f_i[j](value) returns the current contribution of feature j.
f_i = [np.vectorize(lambda a: 0) for _ in range(num_features)]

# Precompute k-means clustering for features that are to be modeled as
    splines.
# For linear features, no knot data is needed.
knot_data = [None] * num_features
n_clusters = int(100 / KNOT_PERCENTILE) + 1  # e.g., for
   KNOT_PERCENTILE=10, n_clusters=11
for i in range(num_features):
    if feature_types[i] == 's':
        data = x[:, i].reshape(-1, 1)
        unique_vals = np.unique(data)
        n_clusters_feature = min(n_clusters, len(unique_vals))
        kmeans = KMeans(n_clusters=n_clusters_feature, random_state=0)
            .fit(data)
        centers = kmeans.cluster_centers_.flatten()
        labels = kmeans.labels_
        knot_data[i] = (centers, labels)

# Backfitting loop
count = 0
error = np.inf
while error > THRESHOLD or count == 0:
    feature = count % num_features

    # Compute the current residuals by removing the effects of all
        other features.
    # This is: residual = y - alpha_hat - sum_{j != feature} f_j(x[:,
        j])
        other_effect = np.sum(
            np.array([f_i[j](x[:, j]) for j in range(num_features) if j !=
                feature]),
            axis=0
        ).reshape(-1, 1)
        residuals_all = y - alpha_hat - other_effect  # shape (N, 1)
        residuals_all = residuals_all.flatten()

        # Update the current feature depending on whether it's spline or
          linear.
        if feature_types[feature] == 's':
            # Spline Update for Feature 'feature'
            centers_unsorted, labels = knot_data[feature]
            unique_labels = np.unique(labels)

            # For each cluster, compute the average residual.
            avg_residuals = []
            for lab in unique_labels:
                avg_res = np.mean(residuals_all[labels == lab])
                avg_residuals.append(avg_res)
            avg_residuals = np.array(avg_residuals)
            # Get corresponding centers (each label corresponds to a
              center given by k-means)
            centers = np.array([centers_unsorted[lab] for lab in
              unique_labels])
            # Sort the clusters by their center values so that the spline
              is fit on an ordered set.
            sort_order = np.argsort(centers)
            sorted_centers = centers[sort_order]
            sorted_avg_residuals = avg_residuals[sort_order]

            # Prepare datapoints for spline fitting: each row is [knot_x,
              average_residual].
            datapoints_feature = np.column_stack([sorted_centers,
              sorted_avg_residuals])
            new_f_feature = natural_spline(datapoints_feature)

            # Center the spline function so that its average over the knot
                centers is zero.
            def get_new_f_feature_normalized(f, knot_x):
                def new_f_feature_normalized(datapoints):
                    return f(datapoints) - np.average(f(knot_x))
                return np.vectorize(new_f_feature_normalized)
            new_f_feature_normalized = get_new_f_feature_normalized(
              new_f_feature, sorted_centers)

            if verbose:
                # Plot intermediate spline fits.
                x_vals = np.linspace(np.min(x[:, feature]), np.max(x[:,
                    feature]), num=50)
                plt.figure()
                plt.plot(x_vals, new_f_feature(x_vals), 'g', label='New
                    Spline')
                plt.plot(x_vals, new_f_feature_normalized(x_vals), 'r',
                    label='Centered Spline')
                plt.plot(x_vals, f_i[feature](x_vals), 'b', label='
                    Previous Estimate')
                plt.scatter(sorted_centers, sorted_avg_residuals, c='r',
                    label='Knot Centroids')
                plt.title(f"Feature {feature} Spline Update")
                plt.legend()
                plt.show()

            # Evaluate the maximum change (error) over the knot centroids.
            error = np.max(np.abs(new_f_feature_normalized(sorted_centers)
                 - f_i[feature](sorted_centers)))
            f_i[feature] = new_f_feature_normalized

        elif feature_types[feature] == 'l':
            # Linear Update for Feature 'feature'
            X_feature = x[:, feature]
            # Center the feature to avoid identifiability issues.
            mean_feature = np.mean(X_feature)
            X_centered = X_feature - mean_feature

            # Compute the least-squares coefficient (beta) for the linear
                term.
            var = np.dot(X_centered, X_centered)
            if var == 0:
                beta = 0.0
            else:
                beta = np.dot(X_centered, residuals_all) / var
            # Define the new linear function.
            new_f_feature = lambda x_val: beta * (x_val - mean_feature)

            # Compute the update error over the observed values.
            error = np.max(np.abs(new_f_feature(X_feature) - f_i[feature](
                X_feature)))

            # Update the function estimate for this feature.
            f_i[feature] = np.vectorize(new_f_feature)

            if verbose:
                x_vals = np.linspace(np.min(X_feature), np.max(X_feature),
                     num=50)
                plt.figure()
                plt.plot(x_vals, new_f_feature(x_vals), 'g', label='New
                    Linear')
                plt.plot(x_vals, f_i[feature](x_vals), 'b', label='
                    Previous Estimate')
                plt.scatter(X_feature, residuals_all, c='r', label='Data
                    Points')
                plt.title(f"Feature {feature} Linear Update")
                plt.legend()
                plt.show()
        else:
            raise ValueError(f"Invalid feature type '{feature_types[
                feature]}' for feature {feature}. Use 's' for spline or 'l'
                 for linear.")

        count += 1

    # Define the final GAM function.
    def gam_function(x_eval):
        # x_eval should be a 2D array with shape (num_points, num_features
            )
        # Compute the predicted value as alpha_hat plus the sum of
            contributions from each feature.
        return alpha_hat + np.sum(np.array([f_i[i](x_eval[:, i]) for i in
            range(num_features)]), axis=0)

    return gam_function
```
Listing 3: K-mean knot selection implementation in Python

# Natural Cubic Splines and Generalized Additive Models: A Comparative Study

Kelvin Luu, Belle Ho, MJ Bailey, Peilin Rao, Jazlin Ong

Math 151A

March 21, 2025

## Outline

## Introduction & Motivation

- **Motivation:** Enhance data interpolation while avoiding oscillations and error blow-up (Runge's phenomenon).
- **Objective:** Develop robust spline methods and integrate them into a Generalized Additive Model (GAM) framework.
- **Applications:** Energy consumption prediction and Bone Mineral Density (BMD) analysis.

## Datasets Overview

- **Southern California Energy Consumption:** Over 370,000 records (2018–2024) from Kaggle.
- **Bone Mineral Density (BMD):** 261 records from a longitudinal study (1999).
- **Synthetic Dataset:** A fixed set of 600 randomly generated records for performance testing.

## Generalized Additive Models (GAMs)

- **Model Definition:**

$$Y = \alpha + \sum_{j=1}^{p} f_j(x_j)$$

- **Key Features:**
  - Each $f_j$ is estimated nonparametrically (typically via spline smoothing).
  - Allows flexible modeling of nonlinear relationships.
  - Can incorporate both continuous and categorical predictors.
- **Estimation:** A backfitting algorithm iteratively refines each $f_j$ while enforcing the constraint

$$\sum_{i=1}^{N} f_j(x_{ij}) = 0, \quad \forall j,$$

for identifiability.

## Natural Cubic Splines with Percentile Knot Selection

- Sort data lexicographically (using `numpy.lexsort`).
- Select knots at specified percentiles to emphasize dense regions.
- Compute spline coefficients via a tridiagonal system with natural boundary conditions.

## K-means Knot Selection for GAM Spline Fitting

- **Objective:** Determine adaptive knot locations by clustering the feature values.
- Given a feature vector $\{x_1, x_2, \ldots, x_N\}$, k-means clustering minimizes:

$$\min_{C_1,\ldots,C_k} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2,$$

  where $\mu_i$ is the centroid of cluster $C_i$.
- **Process:**
  1. **Clustering:** Apply k-means to $x[:, j]$ for each spline feature $j$.
  2. **Residual Aggregation:** For each cluster $i$, compute the average residual $\bar{r}_i$ from the backfitting process.
  3. **Spline Fitting:** Fit a natural spline to $\{(\mu_i, \bar{r}_i)\}$ and center the spline by subtracting its mean over the centroids.

## Cubic Smooth Splines

- **Objective:** Achieve a balance between fitting the data and maintaining smoothness.
- **Penalized Residual Sum of Squares:**

$$PRSS(\alpha, f_1, \ldots, f_p) = \sum_{i=1}^{N} \left( y_i - \alpha - \sum_{j=1}^{p} f_j(x_{ij}) \right)^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t)^2 \, dt,$$

  where $\lambda_j \geq 0$ are tuning parameters.
- **Result:** The minimizer is an additive cubic smooth spline model where each $f_j$ is a cubic smooth spline.

## GAM Backfitting: Overview

- **Initialization:** Set

$$\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i, \quad \hat{f}_j \equiv 0 \quad \forall j.$$

- **Iterative Updates:** For each predictor $j$, update

$$\hat{f}_j \leftarrow S_j\left[y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\right]_{i=1}^{N}$$

  and then center $\hat{f}_j$ by subtracting its mean.

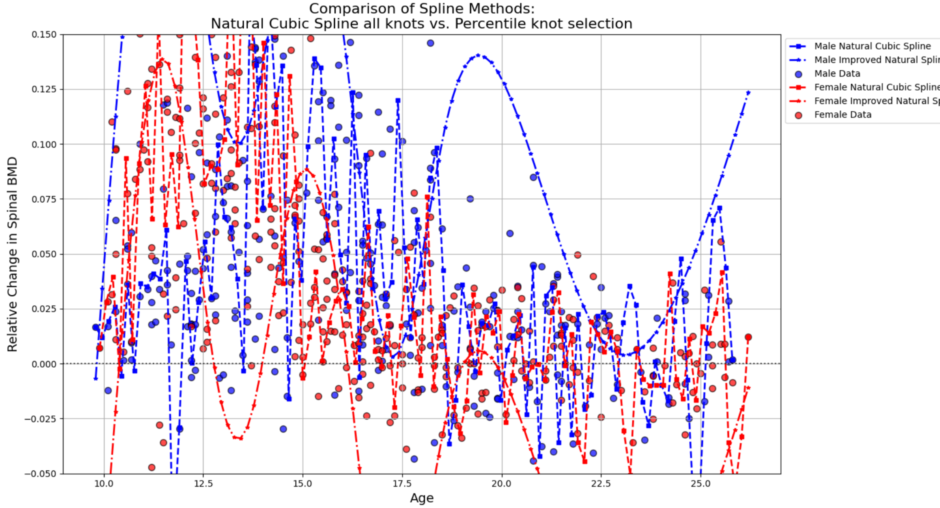
- **Convergence:** Continue iterations until the maximum change in the functions is below a set threshold.

## Backfitting Algorithm (Detailed)

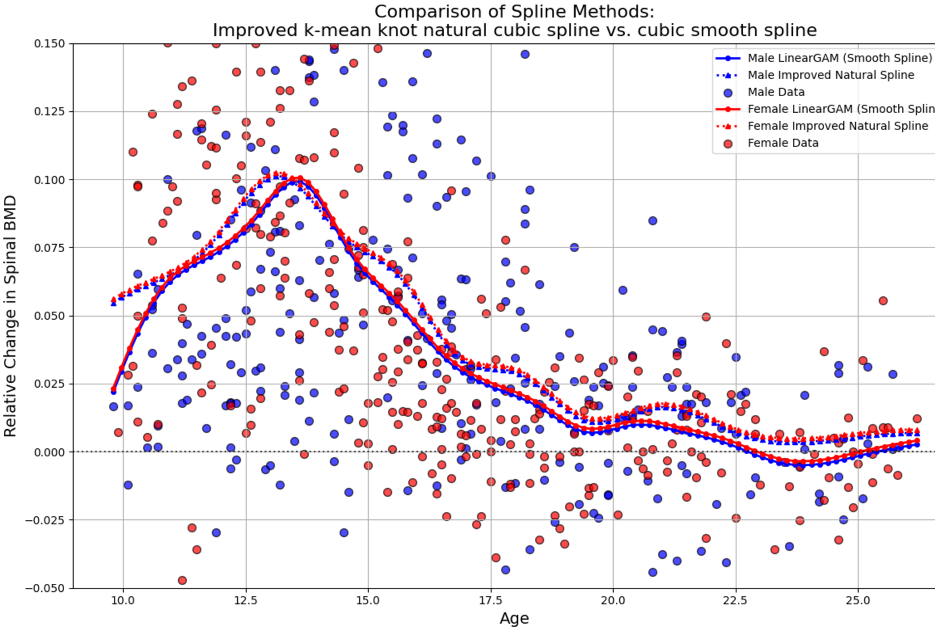
**Algorithm 1** Backfitting Algorithm for Additive Models

```
1: Initialize: $\hat{\alpha} = \frac{1}{N}\sum_{i=1}^{N} y_i$; $\hat{f}_j \leftarrow 0$ for all $j$
2: repeat
3:     for $j = 1, 2, \ldots, p$ do
4:         $\hat{f}_j \leftarrow S_j\left[y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\right]_{i=1}^{N}$
5:         $\hat{f}_j \leftarrow \hat{f}_j - \frac{1}{N}\sum_{i=1}^{N} \hat{f}_j(x_{ij})$
6:     end for
7: until convergence: $\max_j \|\hat{f}_j^{\text{new}} - \hat{f}_j^{\text{old}}\| < \epsilon$
```

- **Note:** The constraint $\sum_{i=1}^{N} \hat{f}_j(x_{ij}) = 0$ ensures identifiability.



## Comparison: Natural Spline (Full Knot Interpolation) vs. Natural Spline (Percentile Knot Selection)

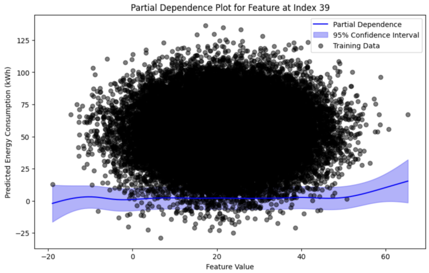


## Comparison: Natural Spline (K-means Centroid Knots) vs. Cubic

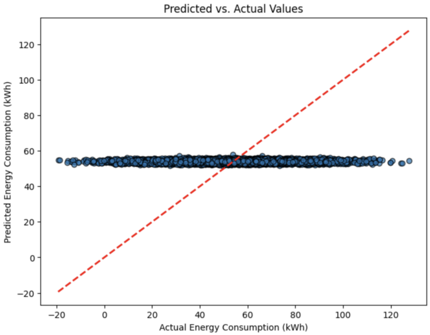
## Results on Bone Mineral Density (BMD) Dataset

- **Methods Compared:**
  1. Natural cubic spline with percentile knot selection.
  2. Natural cubic spline with k-means knot selection.
  3. Cubic smooth spline (penalized).
- **Observations:**
  - Percentile selection may cause issues with skewed data.
  - K-means and penalized splines produce smoother, more robust fits.

# Energy Consumption Visualizations



Partial Dependence Plot



Predicted vs Actual Values

## Energy Consumption Prediction Performance

- **Performance Metrics:**
  - $R^2$ Score: $\approx -0.0042$ (indicating performance slightly worse than a mean-based baseline).
  - Partial dependence plots suggest near-zero effect for several predictors.
- **Interpretation:**
  - The model struggles to capture the full variability in energy consumption.
  - Despite limitations, the stable baseline estimate can aid operational planning.

## Conclusion & Future Work

- **Key Findings:**
  - Adaptive knot selection (both percentile and k-means) enhances spline flexibility.
  - Penalized cubic smooth splines effectively reduce overfitting compared to exact natural cubic splines.
  - The GAM backfitting framework integrates these splines, though challenges in predictive performance remain.
- **Future Directions:**
  - Explore alternative feature transformations and regularization strategies.
  - Enhance model tuning for improved accuracy.
  - Extend the methodology to additional complex datasets.

## References I

- Southern California Energy Consumption dataset, *Kaggle*. https://www.kaggle.com/datasets/datasetengineer/southern-california-energy-consumption

- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

- Wood, S. N. (2006). *Generalized Additive Models: An Introduction with R*. Chapman and Hall/CRC.

- Bachrach LK, et al. (1999). Bone Mineral Acquisition in Youth. *J Clin Endocrinol Metab*.