# Penalized-Constrained Regression

Combining Regularization and Domain Constraints for Cost Estimation

Kevin Joy Max Watstein

2026-02-16

# Table of contents

## Abstract

Small datasets with intercorrelation pose serious challenges to the stability of coefficients generated by Ordinary Least Squares (OLS) regression. A motivating example in cost estimating is Learning Curve with Rate Effect analysis, where datasets are typically small, the lot midpoint (Learning) is correlated to lot quantity (Rate) as production ramps up, and slopes are expected to be $\leq 100\%$. Lasso, Ridge, and Elastic Net regularization methods address multicollinearity by penalizing coefficients. Separately, constrained optimization methods can impose explicit restrictions on coefficient values when prior knowledge about their behavior is known—such as bounding slopes within a known range. This paper investigates the combined effects of penalized regularization methods and constrained optimization. We explore how to assess model stability and goodness-of-fit using likelihood-free diagnostic techniques suited to optimization-based regressions, such as cross-validation for generalization error.

# 1 Introduction

While developing Cost Estimating Relationships (CERs) for small datasets (5-30 data points), a recurring pattern emerged. The regression models often showed strong fit statistics with high $R^2$ and low standard error, but nonsensical coefficients. Coefficients often had wrong signs, implausible magnitudes, and poor p-values. As Department of Defense (DoD) analysts, this story may feel all too familiar.

Intercorrelated datasets are a frequent presence in cost analysis, causing regression models to misbehave. This is especially true for Learning Curve with Rate Effect analysis, where datasets are typically small and the lot midpoint is usually correlated to lot quantity. Traditional remedies for multicollinearity can be insufficient for learning curve datasets. Increasing sample size is infeasible as it requires waiting years for additional production lots and cost data to become available. Dropping explanatory variables, such as rate effect, leads to model misspecification that ignores a fundamental cost driver. This issue is compounded when predicting outside the relevant range which is very common. Even when these remedies are employed, OLS can still produce implausible coefficients with the wrong signs, such as learning curve slopes >100%. Regularization techniques like Ridge, Lasso, and Elastic Net can help stabilize estimates, but they don't guarantee plausible coefficients. Constrained optimization can enforce domain knowledge, but without regularization, it may not solve the underlying multicollinearity problem. This paper explores the combination of penalized regularization and constrained optimization to address both challenges.

For our research, we reviewed existing literature on multicollinearity, regularization, constrained optimization, and learning theory. Drawing on these foundations, we developed the Penalized-Constrained Regression (PCReg) framework, which integrates Elastic Net penalties with domain-knowledge coefficient constraints. We then used Monte Carlo simulation to evaluate PCReg's accuracy estimating the true parameters and compared its performance against OLS across a range of sample sizes and collinearity conditions, measuring both coefficient recovery and out-of-sample prediction accuracy.

This paper develops and validates PCReg for cost estimation, providing PCReg:

1. **Model**: integration of Elastic Net penalties and constraint optimization
2. **Diagnostic**: Boostrap confidence intervals and GDF-adjusted fit statistics for optimization-based regression
3. **Practical Guidance**: when to use PCReg and how to assess model fit

Additionally, we have developed the following tools for the cost estimating community to apply PCReg:

1. **Python Package**: PCReg application in Python
2. **Excel Template**: PCReg application in Excel with limited implementation

# 2 Motivating Example

## 2.1 The Learning Curve Problem

The motivating example for this research is the learning curve with rate effect:

$$\text{Average Unit Cost} = T_1 \cdot (\text{Lot Midpoint})^b \cdot (\text{Lot Quantity})^c \cdot \varepsilon$$

$$\begin{aligned} \text{where: } T_1 &= \text{theoretical first unit cost} \\ b &= \text{learning slope in log space} \\ c &= \text{rate slope in log space} \\ \varepsilon &= \text{multiplicative error} \end{aligned} \tag{1}$$

$b$ and $c$ are both slopes in log space. When transformed to unit space, they become the learning curve slope (LCS = $2^b$) and the rate effect ($2^c$), both typically ranging from 70-100%. As DoD programs ramp up production from Engineering Manufacturing Development (EMD) to Full Rate Production (FRP), lot midpoint (the learning predictor) becomes highly correlated with lot quantity (the rate predictor). By definition LCS and Rate Effect are $\leq 100\%$, meaning costs strictly decrease with cumulative production or lot quantity.

If estimated LCS or Rate Effect are >100% this is often an indicator of multicollinearity or that other explanatory variables such as supply chain disruptions or diseconomies of scale are missing and need to be modeled, not a violation of the definition. Once these factors are properly modeled as separate explanatory variables, estimated LCS and Rate Effect should be $\leq 100\%$, consistent with their definition. It is the analyst's responsibility to ensure the model is properly specified and includes all relevant cost drivers.

## 2.2 Example Scenario

We demonstrate the "Learning Curve Problem" using the following dataset which was generated by our Simulation Study (Section 5).

Table 1: Example Scenario Dataset

| lot_midpoint | lot_quantity | observed_cost |
|---|---|---|
| 11.694 | 30.000 | 44.822 |
| 44.592 | 30.000 | 43.961 |
| 79.598 | 40.000 | 42.172 |
| 129.251 | 60.000 | 39.040 |
| 226.613 | 140.000 | 38.229 |
| 414.709 | 241.000 | 26.349 |
| 713.347 | 360.000 | 22.446 |
| 1076.104 | 360.000 | 21.463 |
| 1437.463 | 360.000 | 24.394 |
| 1812.743 | 390.000 | 24.922 |

Using OLS, we get the following results:

Table 2: Example Scenario OLS Results

| Metric | OLS |
|---|---|
| $T_1$ | 114 |
| Learning Rate | 100.8% |
| Rate Effect | 82.5% |
| $R^2$ | 0.91 |
| F-statistic | 36.7 (p=0.00019) |

We find that by using OLS this example dataset yields invalid coefficents, specifcally a LCS of 100.8% which violates our definiton of LCS being $\leq 100\%$. But otherwise this is a good fit with strong $R^2$ (0.91) and significant F-statistic (p=0.00019). This is a common pattern in cost estimating where OLS produces good fit statistics but implausible coefficients.

# 3 Theoretical Foundations

## 3.1 Ridge Regularization

C. M. Theobald, in *Generalizations of Mean Square Error Applied to Ridge Regression* (Theobald 1974), showed that for any OLS problem there exists a ridge parameter $\lambda^* > 0$ such that the ridge estimator has strictly lower mean squared error (MSE) than OLS. R. W. Farebrother, in *Further Results on the Mean Square Error of Ridge Regression* (Farebrother 1976), extended Theobald's result to broader classes of weighted MSE criteria. Together, these results establish that ridge shrinkage can always improve on OLS in terms of population MSE, not merely training error. The optimal $\lambda^*$ depends on unknown population parameters, which cross-validation estimates empirically.

## 3.2 Lasso Regularization and Constrained Optimization

Our paper is not the first to explore combining regularization (L1 penalties) with constraint optimization. James, Paulson, and Rusmevichientong (2020) developed the *Penalized and Constrained* optimization method (PAC) to compute the solution path for high-dimensional, linearly-constrained criteria. The paper demonstrated that PAC methods remain superior to unconstrained alternatives even when constraints are imperfectly specified:

> *"PAC and relaxed PAC are surprisingly robust to random violations in the constraints… they were still both superior to their unconstrained counterparts for all values [of constraint error] and all settings."*

# 4 PCReg Framework

The PCReg framework addresses the Learning Curve Problem, highlighted by our Section 2.2, by applying the Section 3 research findings to the cost estimation domain. Because PCReg uses optimization rather than closed-form estimation, traditional OLS diagnostics are not directly available, so PCReg also provides specialized model diagnostics based on generalized degrees of freedom, bootstrap inference, and cross-validation.

## 4.1 PCReg Model

The PCReg Model integrates regularization penalties (L1 and L2) with domain-knowledge constraints ($\theta_{\text{lower}} \leq \theta \leq \theta_{\text{upper}}$) to stabilize coefficient estimates, while ensuring they remain within plausible ranges.

> **Notation:** We use $\theta$ (rather than $\beta$) to emphasize that PCReg handles both linear regression and nonlinear optimization problems. For standard linear models, $\theta$ represents regression coefficients; for nonlinear models (e.g., learning curves), $\theta$ represents the optimized parameters. See Section A in the Appendix for complete notation details.

### 4.1.1 Objective Function

The objective function minimized by PCReg combines a loss function with regularization penalties subject to coefficient bounds:

$$\text{Objective:} \quad \arg\min_{\theta} \underbrace{\sum_{i=1}^{n} L(y_i, \hat{y}_i)}_{\text{Loss}} + \underbrace{\lambda\left[\alpha \|\theta\|_1 + \frac{1-\alpha}{2}\|\theta\|_2^2\right]}_{\text{Elastic Net Penalty}}$$

$$\text{Subject to:} \quad \theta_{\text{lower}} \leq \theta \leq \theta_{\text{upper}} \quad \text{(parameter bounds)}$$

$$\begin{aligned}
\text{where:} \quad \hat{y}_i = f(X_i, \theta) &= \text{predicted value (linear or nonlinear)} \\
L(y_i, \hat{y}_i) &= \text{loss function (e.g., SSE, SSPE)} \\
\lambda \geq 0 &= \text{regularization penalty} \\
\alpha \in [0, 1] &= \text{L1/L2 ratio} \\
\|\theta\|_1 = \textstyle\sum_j |\theta_j| &= \text{Lasso penalty} \\
\|\theta\|_2^2 = \textstyle\sum_j \theta_j^2 &= \text{Ridge penalty}
\end{aligned}$$

The lower and upper bounds ($\theta_{lower}$ and $\theta_{upper}$) can be set to achieve loose bounds or tight bounds for the coefficient:

- **Loose bounds** (e.g., 70-100%): Wide range which allows flexibility while preventing egregious coefficient violations
- **Tight bounds** (e.g., 85-95%): Narrower range which incorporates strong prior knowledge but risks over-constraining coefficients

#### 4.1.2 Model Specification

The users can specify the following components of the PCReg model:

- **Functional Form**: Defaults to linear (`X @ coef + intercept`), but any function can be entered
- **Loss function**: Defaults to Sum of Squared Percentage Errors (SSPE), but you can use Sum of Squared Errors (SSE), Mean Squared Error (MSE), or any custom loss function
- **Coefficient bounds**: Componentwise constraints that encode domain knowledge (e.g., LCS between 70% and 100%)
- **Penalty exclusion**: Option to exclude specific coefficients from regularization while still applying bounds
- **Optimization Method**: Defaults to Sequential Least Squares Programming (SLSQP) for constrained optimization, but you can use any optimization method

The framework maximizes flexibility and collapses to known regression techniques in special cases:

- OLS when functional form is linear, `alpha=0`, and bounds are not specified
- Ridge when functional form is linear, `alpha>0` and l1_ratio=0
- Lasso when functional form is linear, `alpha>1` and l1_ratio=1
- Elastic Net when functional form is linear, `alpha>0` and `0 < l1_ratio < 1`
- Constrained regression when bounds are specified but `alpha=0`

#### 4.1.3 Cross Validation (CV) for Hyperparameter Selection

Finding the optimal penalty parameters for PCReg requires testing different values rather than using a formula. CV is a method that evaluates how well a model predicts new data by testing it on observations not used during training. The CV process evaluates all combinations of user-specified penalties ($\lambda$) and L1 ratio values ($\alpha$) to find the hyperparameters that produce the best out-of-sample predictions.

Traditional K-Fold CV (Stone 1974) splits the training data into K subsets, repeatedly training on K-1 subsets and validating on the remaining one. For small datasets, this approach further reduces the already limited training sample size.

Generalized Cross Validation (GCV) approximates leave-one-out cross validation using a mathematical formula rather than actually splitting data (Golub, Heath, and Wahba 1979). It estimates prediction error based on training residuals adjusted by model complexity (degrees of freedom), making it efficient for small samples.

PCReg implements both K-Fold CV and GCV, with GCV as the default for small datasets. For each combination of penalty parameters, the GCV score is calculated as:

$$\text{GCV} = \frac{n \cdot \text{RSS}}{(n - \text{df})^2}$$

$$\begin{aligned} \text{where:} \quad \text{RSS} &= \text{residual sum of squared errors} \\ n &= \text{number of observations} \\ \text{df} &= \text{effective degrees of freedom} \end{aligned}$$

The optimal penalty parameters are those that minimize the GCV score. See **?@sec-generalized-**

**degrees-freedom** for how PCReg calculates effective degrees of freedom for constrained and penalized models.

## 4.2 Model Diagnostics

The flexibility of PCReg means goodness of fit statitics cannot gurantee classical inference (p-values and F-statistic) apply due to supporting coefficient bounds, regularization penalties, custom loss functions and custom functional forms. No single inference framework covers all scenarios. When coefficients are at bounds (from constraints or L1 penalties), classical theory breaks down. Additionally, introducing custom loss and functional forms require specifiying their corresponding distributional assumptions as in Generalized Linear Models (GLMs). We therefore focus on fit statistics that are robust and informative across model specifications:

### 4.2.1 Optimization Diagnostics

We report optimization diagnostics such as convergence status, number of iterations, and whether a parameter is at a bound. When coefficients are at bounds, this indicates that constraints are actively shaping the solution. Note: we have found that applying constraints can change parameters without forcing to a bound. Just because a coefficient is not at a bound does not mean it is unaffected by the constraints; constraints can still significantly impact the optimization process. When coefficients are not at bounds, this suggests that the data naturally satisfies the constraints.

### 4.2.2 In-Sample Fit Statistics

Standard in-sample fit measures remain valid across all model configurations, including Coefficient of Determination ($R^2$), SSE, Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and residual diagnostics.

### 4.2.3 Generalized Degrees of Freedom (GDF)-adjusted fit statistics

Generalized Cross Validation (GCV), Standard Error of Estimate (SEE), and Adjusted $R^2$ all rely on degrees of freedom. However, classical degrees of freedom ($df = n - p$) assumes unconstrained estimation. GDF suggests that optimization based models ought to lose degrees of freedom to take into account the constraints. We implement both Gaines, Kim, and Zhou (2018) and Hu (2010) GDF formulas to properly adjust fit statistics (SEE, SPE, Adjusted $R^2$) and account for the effective model complexity.

**Gaines et al. (2018) Formula (Default):** Gaines, Kim, and Zhou (2018) derive degrees of freedom for constrained Lasso by counting only constraints that are actually binding at the solution:

$$\text{GDF}_{\text{Gaines}} = n - |\text{Active predictors}| - |\text{Equality constraints}| - |\text{Binding inequality constraints}|$$

where $n$ is the sample size and active predictors are non-zero coefficients. This reflects the intuition that non-binding constraints don't reduce effective degrees of freedom.

**Hu (2010) Alternative Formula:** Hu (2010) proposes a more conservative approach where *all specified constraints* count against degrees of freedom, regardless of whether they bind:

$$\text{GDF}_{\text{Hu}} = n - p - (\#\ \text{Constraints}) + (\#\ \text{Redundancies})$$

where $p$ is the total number of estimated parameters (including intercept if fitted). Hu's method always reduces GDF when constraints are specified, making it appropriate for ZMPE-type CERs where constraints fundamentally shape the solution.

#### 4.2.4 Bootstrap Confidence Intervals

Standard bootstrap confidence intervals can be misleading for penalized regression (Goeman 2025), as they assume interior solutions and no regularization bias. Our framework addresses this by computing **both** constrained and unconstrained bootstrap resampling:

- **Constrained bootstrap**: Resamples data and refits the model *with* the specified bounds and regularization (penalized-constrained estimation), quantifying coefficient stability under the imposed modeling constraints
- **Unconstrained bootstrap**: Resamples data and fits OLS *without* bounds or penalization (pure unconstrained estimation), showing coefficient variability in the absence of constraints and regularization. This serves as a baseline for comparison.

### 4.3 Software Implementation

The DoD cost estimating community can apply PCReg using either a Python Package or Excel Template.

#### 4.3.1 Python Package

We developed a full feature complete Python package, `penalized_constrained`, following scikit-learn conventions and API design principles. The package provides a comprehensive implementation of the PCReg framework with support for model fitting, hyperparameter selection via cross-validation, diagnostic reporting with GDF-adjusted fit statistics, and bootstrap confidence intervals. See the Appendix for detailed documentation and usage examples.

#### 4.3.2 Excel Template

To make the methodology accessible to practitioners without programming expertise, we developed an Excel Template that implements a simplified version of the PCReg algorithm. This template provides an intuitive interface for applying PCReg to cost estimation problems.

### 4.4 Example Scenario

Now we will apply PCReg to the example scenario introduced earlier.

#### 4.4.1 PCReg Model Specification

We apply the following PCReg Model Specification to the example scenario:

```
# Define the nonlinear functional form
def prediction_fn(X, params):
    """Learning curve with rate effect: T1 * (midpoint^b) * (quantity^c)"""
    T1, b, c = params
    return T1 * (X[:, 0] ** b) * (X[:, 1] ** c)
```

```
# Configure PCReg model
pc_gcv = pcreg.PenalizedConstrainedCV(
    coef_names=['T1', 'b', 'c'],
    bounds={'T1': (0, None), 'b': (-0.5, 0), 'c': (-0.5, 0)},
    prediction_fn=prediction_fn,
    fit_intercept=False,
    x0=[100, -0.1, -0.1],
    alphas=np.linspace(0, 5, 10),  # Range of lambda values to test
    l1_ratios= [0.0, 0.5, 1.0],  # Test Ridge, Elastic Net, and Lasso penalties
    selection='gcv',           # Generalized Cross-Validation
    loss='sspe',               # Sum of Squared Percentage Errors
    penalty_exclude=['T1'],    # Don't penalize the intercept
    n_jobs=1
)

# Fit the model
pc_gcv.fit(X_train, y_train)
```

*Note:* Due to scikit-learn API conventions, $\lambda$ is labeled `alpha` and $\alpha$ is labeled `l1_ratio` in the code. These correspond to regularization strength and L1/L2 mixing ratio in the theoretical formulation.

**Key Specification Details:**

- **Coefficients**: $T_1$ (first unit cost), $b$ (learning slope), $c$ (rate slope)
- **Bounds**: $T_1 > 0$, $-0.5 \leq b \leq 0$, $-0.5 \leq c \leq 0$
- **alphas (penalty)**: Range between 0 and 5 using linear spacing to explore Lasso, Ridge, and Elastic Net penalties
- **l1_ratios**: Range between 0 and 1 to explore different mixes of L1 and L2 regularization
- **Penalty Selection**: GCV automatically selects optimal $\lambda$ without data splitting
- **Loss Function**: SSPE minimizes percentage errors (appropriate for cost data)
- **Penalty Exclusion**: $T_1$ is not penalized (only slopes are regularized)

Below is a comparison of OLS and PCReg specifications used for this example scenario:

Table 3: Comparison of OLS and PCReg coefficient Specification

| Specification | OLS | PCReg |
|---|---|---|
| loss function | SSE | SSPE |
| functional form | Log-Linear | Nonlinear (Learning Curve) |
| lambda (penalty) | N/A | 1.6667 |
| alpha (L1 ratio) | N/A | 0.00 |

### 4.4.2 PCReg Model Results

Using OLS and PCReg, we get the following results:

Table 4: Comparison of OLS and PCReg coefficient estimates

| Metric | OLS | PCReg |
|---|---|---|
| $T_1$ | 114 | 100 |
| Learning Rate | 100.8% | 96.4% |
| Rate Effect | 82.5% | 88.3% |
| $R^2$ (Train) | 0.91 | 0.90 |

### 4.5 PCReg Model Diagnostics

We were able to find a reasonable solution with PCReg that satisfies our domain knowledge constraints (LCS and Rate Effect $\leq 100\%$) and we still have a reasonble (albeit worse) $R^2$. But how stable are the coefficients? The PCReg model includes bootstrap diagnostics to assess coefficient stability and the impact of constraints. Comparing constrained and unconstrained bootstrap distributions reveals the impact of constraints on coefficient stability. When the constrained and unconstrained distributions differ substantially, constraints are actively shaping the solution. When they are similar, the data naturally satisfies the constraints. The constrained and unconstrained bootstrap results for each coefficient are summarized below.

Table 5: Constrained Bootstrap (with bounds and regularization)

| Coefficient | Mean | Std | 95% CI |
|---|---|---|---|
| $T_1$ | 99.48 | 7.85 | [82.69, 120.57] |
| $b$ | -0.0650 | 0.0285 | [-0.1378, -0.0284] |
| $c$ | -0.1633 | 0.0329 | [-0.2076, -0.0932] |

Table 6: Unconstrained Bootstrap (no bounds, $\alpha$=0)

| Coefficient | Mean | Std | 95% CI |
|---|---|---|---|
| $T_1$ | 109.74 | 15.40 | [100.00, 143.55] |
| $b$ | -0.0050 | 0.0709 | [-0.1781, 0.1119] |
| $c$ | -0.2510 | 0.0916 | [-0.3988, -0.0546] |

Figure 1 shows the Bootstrap coefficient distributions comparing constrained (blue) vs unconstrained (red) estimation. Vertical lines show fitted values. Constraints reduce variance and keep estimates within economically plausible ranges.

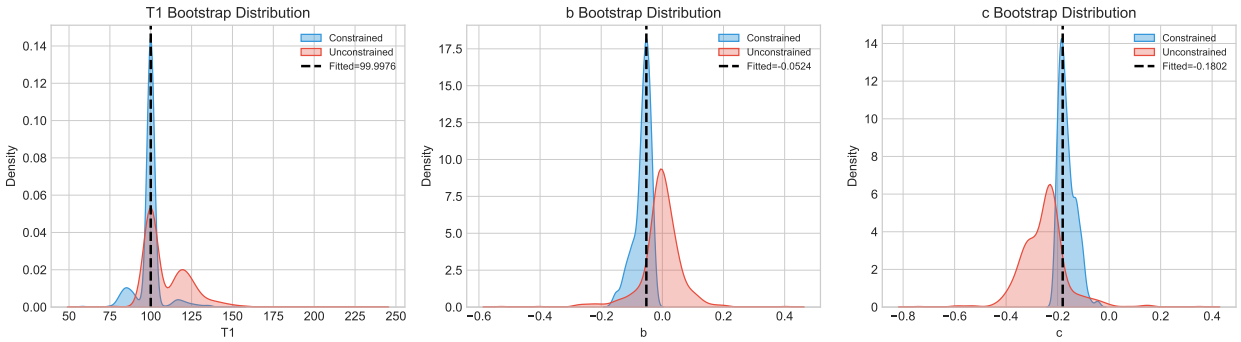


Figure 1: Bootstrap Coefficient Distributions

The bootstrap results above reveal three key diagnostic insights:

1. **Constraints at bounds:** When bootstrap samples frequently hit constraint boundaries, the data is pulling towards implausible values, which is exactly when constraints help most.
2. **Variance reduction:** The constrained bootstrap shows tighter distributions than the unconstrained, reducing coefficient uncertainty at the cost of some bias.
3. **Divergence indicates constraint impact:** The differences between constrained and unconstrained means indicate how much the constraints are actively shaping the solution.

A comprehensive diagnostic report with full model specifications and bootstrap distributions is provided in Appendix A. An interactive HTML version has also been saved to `pcreg_diagnostic_report.html` for detailed exploration.

# 5 Simulation Study

We performed a Monte Carlo simulation study to evaluate PCReg and compare with traditional OLS.

## 5.1 Siumulation Design

Our simulation study specifies four factors and generates 8,100 scenarios (81 factor combinations × 100 replications).

Table 7: Simulation study factorial design

| Factor | Levels |
|---|---|
| Number of Training Lots | 5, 10, 30 |
| CV error | 0.01, 0.1, 0.2 |
| Learning rate | 85%, 90%, 95% |
| Rate effect | 80%, 85%, 90% |

For each scenario, we:

1. **Randomly selected a quantity profile** from the SAR database (actual defense program procurement histories)
2. **Generated simulated average unit costs** using the learning curve model (Equation 1) with the scenario's true parameters
3. **Added multiplicative lognormal noise** with the specified coefficient of variation (CV)
4. **Split data**: First $n$ training lots for training, remaining lots for test

This approach ensures realistic lot structures (varying quantities, realistic ramp-up patterns) while controlling the true underlying parameters. The number of test lots varies by scenario depending on the program's total lot history. Some programs have many available lots beyond training, others have few or none. This variability is acceptable as it reflects real-world conditions.

## 5.2 Example Scenario

The following tables describe the example scenario. Table 8 shows the simulation factor levels selected for this scenario from the factorial design (Table 7), and Table 9 shows the derived characteristics specific to this scenario.

Table 8: Example Scenario Simulation Factors

| Factor | Value |
|---|---|
| Number of Training Lots | 10 |
| CV Error | 0.1 |
| True Learning Rate | 95.0% |
| True Rate Effect | 90.0% |

Table 9: Example Scenario Derived Characteristics

| Characteristic | Value |
|---|---|
| Total lots (from SAR) | 30 |
| Test lots | 20 |
| True $T_1$ | 100 |
| Predictor Correlation | 0.95 |

Below is the example scenario simulation dataset with training and test lots. The training data matches the dataset shown earlier, but now we also see the test lots with their true underlying costs.

Table 10: Motivating example dataset with training and test lots

| lot_type | lot_midpoint | lot_quantity | observed_cost | true_cost |
|---|---|---|---|---|
| train | 11.694 | 30 | 44.822 | 49.71 |
| train | 44.592 | 30 | 43.961 | 45.022 |
| train | 79.598 | 40 | 42.172 | 41.287 |
| train | 129.251 | 60 | 39.040 | 37.451 |
| train | 226.613 | 140 | 38.229 | 31.586 |
| train | 414.709 | 241 | 26.349 | 27.811 |
| train | 713.347 | 360 | 22.446 | 25.136 |
| train | 1076.1 | 360 | 21.463 | 24.382 |
| train | 1437.46 | 360 | 24.394 | 23.865 |
| train | 1812.74 | 390 | 24.922 | 23.176 |
| test | 2208.26 | 400 | — | 22.752 |
| test | 2669.38 | 525 | — | 21.526 |
| test | 3231.51 | 600 | — | 20.798 |
| test | 3789.4 | 512 | — | 21.056 |
| test | 4320.37 | 550 | — | 20.627 |
| test | 4866.8 | 542 | — | 20.491 |
| test | 5408.09 | 540 | — | 20.343 |
| test | 5955.19 | 554 | — | 20.12 |
| test | 6509.39 | 554 | — | 19.988 |
| test | 7063.56 | 554 | — | 19.868 |
| test | 7615.72 | 550 | — | 19.779 |
| test | 8165.84 | 550 | — | 19.677 |
| test | 8715.95 | 550 | — | 19.583 |
| test | 9266.04 | 550 | — | 19.494 |
| test | 9816.12 | 550 | — | 19.411 |
| test | 10366.2 | 550 | — | 19.333 |
| test | 10916.3 | 550 | — | 19.259 |
| test | 11466.3 | 550 | — | 19.189 |
| test | 12016.4 | 550 | — | 19.123 |
| test | 12556.5 | 530 | — | 19.168 |

Figure 2 shows the Learning curve data with `python n_train` training lots (blue) and the true

underlying cost distribution (dashed line). Point sizes reflect lot quantities. The goal is to predict the true relationship, not just fit the noisy observations.

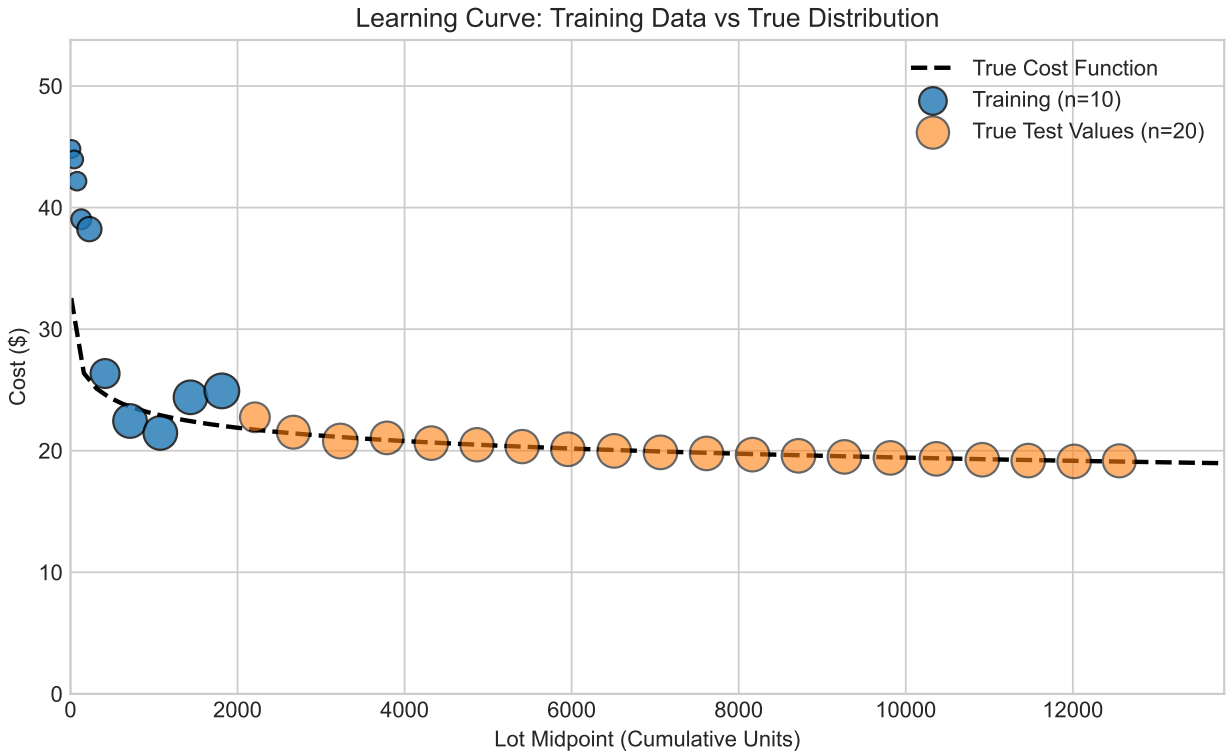


Figure 2: Example Scenario Learning Curve

OLS achieves a *lower* Train $R^2$ OLS (0.912) than PCReg (0.907). This is expected, penalties and constraints add bias to the training fit. However, PCReg achieves a lower (better) Test MAPE, meaning its predictions are closer to the true underlying costs on out-of-sample data, as seen below.

Table 11: Comparison of estimation methods.

| Metric | True | OLS | OLS-LearnOnly | PCReg |
|---|---|---|---|---|
| $T_1$ | 100 | 114 | 79 | 100 |
| Learning Rate | 95.0% | 100.8% | 89.2% | 97.4% |
| Rate Effect | 90.0% | 82.5% | – | 87.3% |
| Valid Coefficients | Yes | NO | Yes | Yes |
| Train $R^2$ | – | 0.912 | 0.796 | 0.907 |
| Test MAPE (vs True) | – | 9.8% | 8.4% | 3.9% |

Figure 3 shows the Residuals for each model on training data (left) and against true test values (right). PCReg shows larger training residuals but smaller errors when predicting the true underlying relationship.

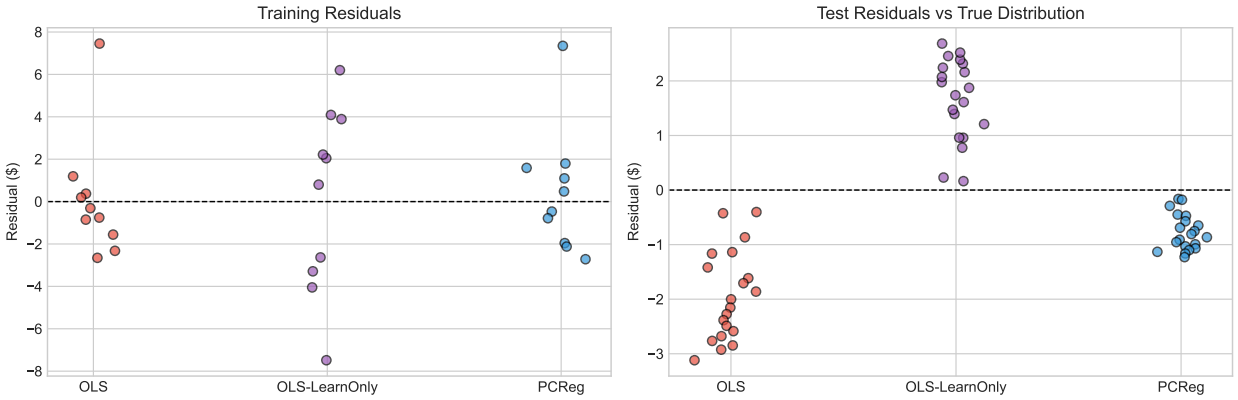


Figure 3: Example Scenario Residuals

# 6 Key Findings

Across 8,100 simulation scenarios, OLS produced economically unreasonable coefficients (learning curve or rate effect outside 70-100%) in **16.7%** of cases (1,355 scenarios).

## 6.1 PCReg Significantly Outperforms OLS When Coefficients Are Unreasonable

When OLS produces unreasonable coefficients (n=1,355), PCReg wins **78.7%** of scenarios on Test MAPE (lower is better).

| Metric | OLS Mean | PCReg Mean | OLS Median | PCReg Median |
|---|---|---|---|---|
| Test MAPE | 0.2862 | 0.1456 | 0.2094 | 0.1179 |
| T1 APE | 66.5658 | 0.5108 | 0.5694 | 0.3418 |
| LC Abs Error | 0.1182 | 0.0483 | 0.0943 | 0.0489 |
| RC Abs Error | 0.3548 | 0.1219 | 0.2347 | 0.1 |

## 6.2 PCReg Performs Comparably When OLS Coefficients Are Reasonable

When OLS produces reasonable coefficients, OLS "wins" on Test MAPE in **66.3%** of scenarios. However, the performance differences are negligible in practical terms.

| Metric | OLS Mean | PCReg Mean | OLS Median | PCReg Median |
|---|---|---|---|---|
| Test MAPE | 0.0549 | 0.061 | 0.0253 | 0.0298 |
| T1 APE | 0.1628 | 0.1562 | 0.0623 | 0.0709 |
| LC Abs Error | 0.0156 | 0.0154 | 0.007 | 0.007 |
| RC Abs Error | 0.0304 | 0.0353 | 0.0153 | 0.0172 |

: Performance comparison when OLS produces reasonable coefficients (n=6,745). {#tbl-reasonable}

**Key Insight**: When OLS coefficients are reasonable, the mean Test MAPE difference is only **0.61 percentage points** (11.1% relative difference). While statistically detectable (Wilcoxon p=nan), this difference is negligible in practice. OLS's occasional large errors (visible in the heavy right tail) inflate the mean, but the median performance is nearly identical.

## 6.3 Summary: Win Rates by Coefficient Reasonableness

Table 14: PCReg win rates against OLS by coefficient reasonableness

| OLS Coefficients | N Scenarios | PCReg Win Rate (Test MAPE) |
|---|---|---|
| Reasonable (70-100%) | 6,745 | 33.7% |
| Unreasonable (<70% or >100%) | 1,355 | 78.7% |
| **Overall** | 8,100 | 41.2% |

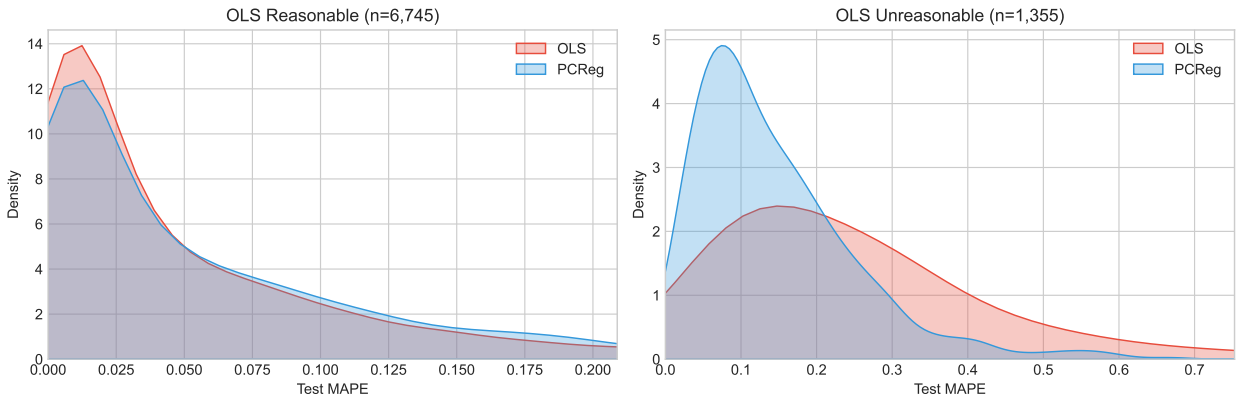


Figure 4: Distribution of Test MAPE for OLS vs PCReg, stratified by whether OLS produced reasonable coefficients. When unreasonable (right), OLS shows a heavy right tail of large errors that PCReg avoids.

### 6.4 OLS-LearnOnly Performs Poorly Outside Training Range

OLS with only the learning variable (OLS-LearnOnly) ignores the rate effect, which leads to poor extrapolation:

Table 15: Average Test MAPE by model

| Model | Mean Test MAPE |
|---|---|
| OLS-LearnOnly | 15.9% |
| OLS | 9.9% |
| PCReg | 7.7% |

OLS-LearnOnly has worse Test MAPE than full OLS in **63.9%** of scenarios. While simplifying the model may seem appealing, omitting the rate effect leads to systematic prediction errors outside the training range.

PCReg outperforms OLS-LearnOnly on Test MAPE in **67.9%** of all scenarios:

- When OLS coefficients are **reasonable**: PCReg wins **70.9%**
- When OLS coefficients are **unreasonable**: PCReg wins **52.8%**

Across all scenarios, PCReg outperforms OLS on Test MAPE in **41.2%** of cases, indicating a consistent predictive advantage even when OLS is competitive.

#### 6.4.1 Coefficient Bias Analysis

Constraints introduce bias in coefficient estimates. We analyze whether this bias is systematic and how it affects prediction:

Table 16: Coefficient bias statistics (Estimated - True). Mean/Median near 0 indicates unbiased estimation; lower Std indicates more stable estimates.

| | Coefficient | Statistic | OLS | PCReg |
|---|---|---|---|---|
| 0 | $T_1$ | Mean | 1108.37 | -3.58 |
| 1 | $T_1$ | Median | -0.22 | -2.73 |
| 2 | $T_1$ | Std | 69135.60 | 46.12 |
| 3 | $b$ | Mean | -0.0002 | -0.0078 |
| 4 | $b$ | Median | -0.0002 | -0.0017 |
| 5 | $b$ | Std | 0.11 | 0.05 |
| 6 | $c$ | Mean | -0.0023 | 0.03 |
| 7 | $c$ | Median | 0.0003 | 0.01 |
| 8 | $c$ | Std | 0.29 | 0.12 |

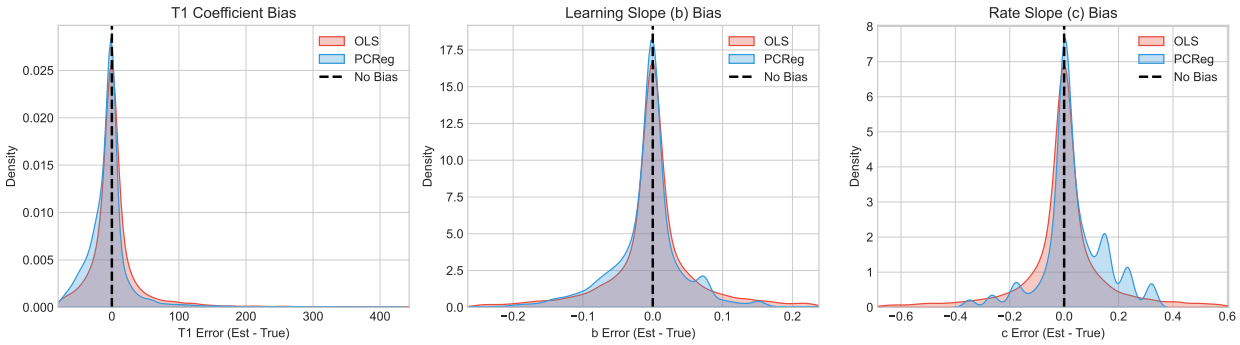


Figure 5: Distribution of coefficient errors by model (1st-99th percentile). Vertical line at 0 indicates no bias. PCReg shows tighter distributions despite some bias, leading to lower variance in predictions.

**Key Insight**: While OLS is theoretically unbiased (mean errors near 0), it has high variance—the distribution is wide. PCReg may have slight bias but much lower variance, leading to better overall prediction accuracy (the classic bias-variance tradeoff).

## 6.5 Parameter Effects on Model Performance

Beyond coefficient reasonableness, several design factors systematically influence when PCReg outperforms OLS. We examine sample size, predictor correlation, and noise level.

**By OLS Coefficient Reasonableness:**

| OLS Coefficients | N | PCReg Win Rate | OLS Mean MAPE | PCReg Mean MAPE |
|---|---|---|---|---|
| Unreasonable | 1355 | 78.7% | 28.6% | 14.6% |
| Reasonable (70-100%) | 6745 | 33.7% | 5.5% | 6.1% |

**By Training Lots:**

| Sample Size | N | PCReg Win Rate | % OLS Unreasonable | OLS Mean MAPE | PCReg Mean MAPE |
|---|---|---|---|---|---|
| 5 | 2700 | 56.9% | 32.0% | 16.2% | 10.7% |
| 10 | 2700 | 46.5% | 16.8% | 7.4% | 6.4% |
| 30 | 2700 | 20.2% | 1.4% | 3.7% | 4.8% |

**By Predictor Correlation:**

| Correlation | N | PCReg Win Rate | % OLS Unreasonable | OLS Mean MAPE | PCReg Mean MAPE |
|---|---|---|---|---|---|
| <0.80 | 3187 | 31.8% | 8.5% | 6.7% | 6.5% |
| 0.80-0.90 | 873 | 52.9% | 22.1% | 10.3% | 7.7% |
| 0.90-0.95 | 2204 | 38.9% | 15.2% | 9.8% | 7.6% |
| >0.95 | 1416 | 56.5% | 35.0% | 14.5% | 8.7% |

**By Noise Level (CV Error):**

| CV Error | N | PCReg Win Rate | % OLS Unreasonable | OLS Mean MAPE | PCReg Mean MAPE |
|---|---|---|---|---|---|
| 0.01 | 2700 | 36.7% | 0.1% | 0.9% | 1.0% |
| 0.1 | 2700 | 43.3% | 16.2% | 9.4% | 8.1% |
| 0.2 | 2700 | 43.7% | 33.9% | 19.6% | 14.1% |

**Key Observations:**

1. **Sample Size Effect**: As sample size decreases, OLS becomes less stable. With n=5 lots, PCReg wins **56.9%** of scenarios compared to **20.2%** at n=30. The rate of unreasonable OLS coefficients also increases from **1.4%** to **32.0%** as sample size decreases.

2. **Correlation Effect**: Higher predictor correlation destabilizes OLS. At correlation >0.95, PCReg wins **56.5%** of scenarios versus **31.8%** at lower correlations. This is the multicollinearity effect. When predictors are highly correlated, OLS coefficient estimates become unstable and constraints provide crucial stability.

3. **Noise Level Effect**: Higher CV error increases the advantage of PCReg. With more noise, OLS has greater difficulty separating learning and rate effects, leading to more unreasonable coefficient estimates.

4. **Compounding Effects**: These factors interact—small samples with high correlation and high noise represent the most challenging scenarios for OLS, where PCReg provides the greatest benefit.

# 7 Recommendations: When to Use PCReg

**Practical Decision Rules:**

1. **If OLS produces unreasonable coefficients** (LC or RC outside 70-100%): **Use PCReg**, it significantly outperforms OLS in these scenarios
2. **For small samples (n  5 or 10 lots) with noisy data**: **Prefer PCReg**, it wins 60-67% of scenarios
3. **For large samples (n  30 lots)**: **OLS is generally preferred**, it wins ~80% of scenarios
4. **For intermediate cases**: Either method is acceptable; PCReg provides insurance against unreasonable coefficients with minimal downside

**Bottom line**: The difference between OLS and PCReg is typically small and can be controlled by explicitly defining loose constraints and arbitrarily small penalties.

## 7.1 Software

The `penalized-constrained` Python package was developed specifically for the cost estimating community. As of this paper's publication, the software is in active development but has achieved stable functionality.

**Installation:** While we anticipate release to PyPI for convenient installation via `pip install penalized-constrained`, the package is currently available via GitHub. To install from the development repository:

```
pip install git+https://github.com/frankij11/Penalized-Constrained-Regression.git
```

For quick-start guides and basic usage examples, see the package documentation on GitHub or Section D in the appendices.

A key advantage of this framework is that **PCReg collapses to OLS** when no constraints are defined, no penalties are applied ( =0), and a linear functional form is used. This allows analysts to use a single, cohesive library for all regression analysis. From standard OLS to fully constrained penalized models with custom functional forms all without switching tools or workflows. This library is designed to integrate seamlessly Python's scikit-learn data science workflows and allow machine learning techniques to solve for optimal parameters.

## 7.2 Summary

Our simulation study demonstrates that PCReg provides meaningful advantages in small-sample, high-noise scenarios where OLS is most likely to produce unreasonable coefficients, while large samples favor standard OLS. The constraints introduce beneficial bias that reduces prediction variance—a classic bias-variance tradeoff that works in the analyst's favor when data are limited. With GCV enabling reliable hyperparameter selection using as few as 5 observations and OLS-LearnOnly's poor extrapolation performance reinforcing that omitting the rate effect oversimplifies the problem, analysts have clear guidance: match the method to the sample size and data quality, with PCReg serving as effective insurance when uncertainty is high.

## References

Craven, Peter, and Grace Wahba. 1978. "Smoothing Noisy Data with Spline Functions: Estimating the Correct Degree of Smoothing by the Method of Generalized Cross-Validation." *Numerische*

*Mathematik* 31: 377–403. https://doi.org/10.1007/BF01404567.

Farebrother, R. W. 1976. "Further Results on the Mean Square Error of Ridge Regression." *Journal of the Royal Statistical Society: Series B* 38 (2): 248–50.

Gaines, Brian R., Jeongyoun Kim, and Hua Zhou. 2018. "Algorithms for Fitting the Constrained Lasso." *Journal of Computational and Graphical Statistics* 27 (4): 861–71. https://doi.org/10.1080/10618600.2018.1473777.

Goeman, Jelle J. 2025. *Penalized: L1 (Lasso and Fused Lasso) and L2 (Ridge) Penalized Estimation in GLMs and in the Cox Model*. https://CRAN.R-project.org/package=penalized.

Golub, Gene H., Michael Heath, and Grace Wahba. 1979. "Generalized Cross-Validation as a Method for Choosing a Good Ridge Parameter." *Technometrics* 21 (2): 215–23. https://doi.org/10.1080/00401706.1979.10489751.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Springer. https://doi.org/10.1007/978-0-387-84858-7.

Hu, Shu-Ping. 2010. "Generalized Degrees of Freedom for Constrained CERs." PRT-191. Tecolote Research.

James, Gareth M., Courtney Paulson, and Paat Rusmevichientong. 2020. "Penalized and Constrained Optimization: An Application to High-Dimensional Website Advertising." *Journal of the American Statistical Association* 115 (529): 107–22. https://doi.org/10.1080/01621459.2019.1609970.

Stone, Mervyn. 1974. "Cross-Validatory Choice and Assessment of Statistical Predictions." *Journal of the Royal Statistical Society: Series B (Methodological)* 36 (2): 111–47.

Theobald, C. M. 1974. "Generalizations of Mean Square Error Applied to Ridge Regression." *Journal of the Royal Statistical Society: Series B* 36 (1): 103–6.

Zou, Hui, and Trevor Hastie. 2005. "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society: Series B* 67 (2): 301–20.

# A PCReg Mathematical Notation

## A.1 Complete Notation Table

Table 21: Complete mathematical notation for PCReg framework

| Symbol | Definition | Example |
|---|---|---|
| $X$ | Feature matrix (n × p) | Lot midpoint, quantity, complexity |
| $X_i$ | i-th row of X | Predictor values for observation i |
| $y_i$ | Observed response | Actual cost for observation i |
| $\theta$ | Parameter vector (p × 1) | Coefficients being estimated |
| $\hat{y}_i = f(X_i, \theta)$ | Predicted value | Model prediction |
| $\lambda \geq 0$ | Regularization strength | Overall penalty weight |
| $\alpha \in [0, 1]$ | L1/L2 mixing parameter | 0=Ridge, 1=Lasso |
| $L(y_i, \hat{y}_i)$ | Loss function | SSE, SSPE, MSE, etc. |

## A.2 Prediction Function Forms

**Linear Model (Default):**

$$\hat{y}_i = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} + \cdots + \theta_p x_{ip} = X_i \theta + \theta_0$$

For linear models, $\theta = [\beta_1, \ldots, \beta_p]$ represents regression coefficients (slopes), and $\theta_0$ is the intercept.

**Nonlinear Model (Learning Curve Example):**

$$\hat{y}_i = T_1 \cdot (\text{LotMidpoint}_i)^b \cdot (\text{LotQuantity}_i)^c$$

Here $\theta = [T_1, b, c]$ with domain-specific meanings:

- $T_1$ = Base cost parameter
- $b$ = Learning curve slope exponent
- $c$ = Rate effect slope exponent

## A.3 Relationship to Standard Elastic Net

PCReg generalizes the standard Elastic Net formulation from Zou and Hastie (2005) and scikit-learn:

**Standard Elastic Net:**

$$\min_{\beta} \frac{1}{2n} \|y - X\beta\|_2^2 + \lambda \left[ \alpha \|\beta\|_1 + \frac{1 - \alpha}{2} \|\beta\|_2^2 \right]$$

**PCReg Extensions:**

1. **Coefficient bounds:** $\theta_{\text{lower}} \leq \theta \leq \theta_{\text{upper}}$ (domain constraints)
2. **Flexible loss functions:** $L(y, \hat{y})$ beyond squared error (e.g., SSPE for cost data)
3. **Nonlinear models:** $f(X, \theta)$ can be any differentiable function
4. **Selective penalties:** Exclude specific parameters (e.g., intercepts, base costs) via `penalty_exclude`

### A.4 Cost Estimating Terminology Mapping

For the cost estimating community, PCReg notation maps to familiar CER concepts:

Table 22: Mapping between mathematical and cost estimating terminology

| Math Notation | CER Terminology | Example |
|---|---|---|
| $X$ | Cost drivers | Lot size, quantity, weight |
| $\theta$ | CER parameters | Slopes, intercepts, exponents |
| $y$ | Historical costs | Actual program costs |
| $\hat{y}$ | Cost estimates | CER predictions |
| Bounds on $\theta$ | Domain constraints | LCS: 70-100%, b: -0.5 to 0 |
| $\lambda$ | Regularization | Shrinkage for small datasets |

PCReg addresses the typical CER challenge: small samples (5-30 points), correlated predictors, and physically-motivated bounds on parameters.

# B Simulation Details

## B.1 Data Generation Process

For each of the 8,100 scenarios:

1. **Select quantity profile**: Randomly sample a defense program from the SAR database with sufficient lot history
2. **Extract lot structure**: Use actual procurement quantities and calculate lot midpoints
3. **Generate true costs**: Apply learning curve model with scenario parameters
4. **Add noise**: Multiply true costs by lognormal error: $Y_{obs} = Y_{true} \cdot e^{\epsilon}$ where $\epsilon \sim N(-\sigma^2/2, \sigma^2)$
5. **Split data**: First $n$ lots for training, **all remaining lots** for test (variable test set size)

## B.2 Model Specifications

Table 23: Models compared in main paper

| Model | Description | Constraints |
|---|---|---|
| OLS | Standard log-log OLS | No |
| OLS_LearnOnly | OLS with learning variable only | No |
| PCReg | GCV-selected penalty + constraints | Yes |

# C Full Results (All Models)

Table 24: Overall model performance across all 8,100 scenarios. Lower Test MAPE is better.

| Model | Test MAPE | Test SSPE | b Error | c Error | R2 |
|---|---|---|---|---|---|
| PCReg_Tight | 0.0561 | 0.0861 | 0.0200 | 0.0337 | 0.852 |
| PCReg_ConstrainOnly | 0.0764 | 0.2052 | 0.0354 | 0.0791 | 0.877 |
| PCReg | 0.0773 | 0.2074 | 0.0338 | 0.0827 | 0.871 |
| PCRegGCV_LogMSE | 0.0778 | 0.2832 | 0.0341 | 0.0776 | 0.877 |
| PCReg_CV | 0.0794 | 0.2538 | 0.0353 | 0.0814 | 0.862 |
| BayesianRidge | 0.0801 | 0.4595 | 0.0360 | 0.0886 | 0.882 |
| PCReg_AICc | 0.0808 | 0.2171 | 0.0349 | 0.0913 | 0.863 |
| RidgeCV | 0.0952 | 0.9362 | 0.0481 | 0.1216 | 0.896 |
| LassoCV | 0.0957 | 0.9767 | 0.0428 | 0.1082 | 0.858 |
| OLS | 0.0994 | 0.9727 | 0.0520 | 0.1319 | 0.897 |
| OLS_LearnOnly | 0.1592 | 0.9543 | 0.0827 | 0.2361 | 0.789 |

# D Software Documentation

## D.1 Installation

```
pip install penalized-constrained
```

## D.2 Basic Usage

```
import penalized_constrained as pcreg
import numpy as np

model = pcreg.PenalizedConstrainedCV(
    coef_names=['T1', 'b', 'c'],
    bounds={
        'T1': (0, None),      # T1 must be positive
        'b': (-0.5, 0),       # Learning rate 70-100%
        'c': (-0.5, 0)        # Rate effect 70-100%
    },
    prediction_fn=lambda X, p: p[0] * X[:,0]**p[1] * X[:,1]**p[2],
    loss='sspe',
    selection='gcv'
)
model.fit(X_train, y_train)
```

## D.3 Citation

```
@inproceedings{joy2026pcreg,
  title={Penalized-Constrained Regression: Combining Regularization
         and Domain Constraints for Cost Estimation},
  author={Joy, Kevin and Watstein, Max},
  booktitle={ICEAA Professional Development \& Training Workshop},
  year={2026}
}
```

# E Cross Validation Methods

## E.1 Overview

Cross Validation (CV) is a resampling technique used to evaluate how well a statistical model generalizes to independent data. In the context of PCReg, CV helps us select optimal penalty parameters ($\lambda$ and $\alpha$) that balance model fit against overfitting. This appendix provides technical details on the CV methods implemented in our framework.

## E.2 K-Fold Cross Validation

### E.2.1 Methodology

K-Fold CV divides the training data into K roughly equal-sized subsets (folds). The model is trained K times, each time using K-1 folds for training and the remaining fold for validation. The CV score is the average prediction error across all K validation folds.

### E.2.2 Algorithm

For each candidate penalty parameter value:

1. Partition training data into K folds: $D = \{F_1, F_2, ..., F_K\}$
2. For each fold $k = 1, ..., K$:
   - Train model on $D \setminus F_k$ (all data except fold k)
   - Predict on validation fold $F_k$
   - Calculate validation error: $E_k = \sum_{i \in F_k} L(y_i, \hat{y}_i)$
3. Compute CV score: $\text{CV}_K = \frac{1}{K} \sum_{k=1}^{K} E_k$
4. Select penalty parameters that minimize $\text{CV}_K$

### E.2.3 Choice of K

- **K=5 or K=10**: Common choices balancing bias and variance
- **Leave-One-Out (LOO, K=n)**: Minimum bias but high computational cost and high variance
- **Smaller K**: Less computation, higher bias (less training data per fold), lower variance
- **Larger K**: More computation, lower bias (more training data per fold), higher variance

### E.2.4 Advantages

- Directly estimates out-of-sample prediction error
- Straightforward interpretation
- Well-established in machine learning literature
- Provides robust estimates when data is sufficient

### E.2.5 Disadvantages

- Reduces effective training sample size (uses only $(K-1)/K$ of data per fold)
- Problematic for small datasets (n $< 20$)
- Computationally expensive (requires K model fits per penalty value)
- Can have high variance when K is large or n is small
- May produce unstable penalty selection with small samples

## E.3 Generalized Cross Validation (GCV)

### E.3.1 Motivation

GCV (Golub, Heath, and Wahba 1979; Craven and Wahba 1978) was developed to approximate Leave-One-Out (LOO) cross validation without the computational burden of fitting n separate models. For small datasets common in cost estimation (n=5-20), GCV provides penalty selection without further reducing the training sample.

### E.3.2 Mathematical Formulation

GCV estimates the prediction error using the full training data and a degrees-of-freedom adjustment:

$$\text{GCV} = \frac{n \cdot \text{RSS}}{(n - \text{df})^2}$$

where:

- $n$ = sample size
- $\text{RSS} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ = residual sum of squares on training data
- df = effective degrees of freedom (Generalized Degrees of Freedom, GDF)

The penalty parameters that minimize GCV are selected as optimal.

### E.3.3 Degrees of Freedom Adjustment

The denominator $(n - \text{df})^2$ penalizes model complexity. For PCReg, we use the Gaines et al. (2018) GDF formula:

$$\text{df} = |\text{Active predictors}| + |\text{Equality constraints}| + |\text{Binding inequality constraints}|$$

This accounts for the effective number of parameters after accounting for regularization and active constraints.

### E.3.4 GCV as LOO Approximation

GCV approximates the LOO cross validation score:

$$\text{LOO} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i^{(-i)}\right)^2$$

where $\hat{y}_i^{(-i)}$ is the prediction for observation $i$ from a model trained without observation $i$.

The key approximation is:

$$y_i - \hat{y}_i^{(-i)} \approx \frac{y_i - \hat{y}_i}{1 - h_i}$$

where $h_i$ is the leverage (diagonal element of the hat matrix). GCV replaces individual leverages with the average: $h_i \approx \text{df}/n$.

#### E.3.5 Advantages

- **No data splitting**: Uses full training sample, crucial for small datasets
- **Efficient computation**: Single model fit per penalty value (vs. K fits for K-Fold)
- **Deterministic**: No randomness from fold assignments
- **Asymptotically equivalent to LOO**: Converges to LOO as n increases
- **Optimal for small samples**: Maintains training power when $n < 20$

#### E.3.6 Disadvantages

- Requires differentiable loss and functional forms to compute GDF
- Approximation quality depends on leverage uniformity
- Less intuitive than K-Fold CV
- May be unstable if GDF calculation is unreliable
- Not applicable to all model types (works well for penalized regression)

### E.4 Assumptions and Limitations

#### E.4.1 Common Assumptions (Both K-Fold and GCV)

1. **Independence**: Observations are independent and identically distributed (i.i.d.)
   - Violation: Time series data, clustered data, or spatial correlation
   - Impact: CV may underestimate prediction error
2. **Stationarity**: The data generation process remains stable
   - Violation: Non-stationary processes or regime changes
   - Impact: Past data may not predict future well
3. **Representative samples**: Training data represents the population of interest
   - Violation: Selection bias, small unrepresentative samples
   - Impact: Selected penalties may not generalize
4. **No data leakage**: Validation data must be truly held out
   - Violation: Feature engineering on full dataset before CV
   - Impact: Overly optimistic error estimates

#### E.4.2 Specific to K-Fold CV

- **Sufficient data**: Requires enough observations to create meaningful folds (typically $n > 50$ for K=10)
- **Balanced folds**: Folds should be representative of full dataset

#### E.4.3 Specific to GCV

- **Smooth loss function**: GCV derivation assumes differentiable loss
- **Accurate GDF**: Requires reliable degrees of freedom calculation
- **Linear-like behavior**: Works best for models with relatively smooth penalty effects

### E.5 Comparison and Recommendations

Table 25: Comparison of K-Fold CV and GCV

| Criterion | K-Fold CV | GCV | Recommendation |
|---|---|---|---|
| **Sample size n < 10** | Poor | Good | **Use GCV** |
| **Sample size 10 n < 20** | Fair | Good | **Prefer GCV** |
| **Sample size 20 n < 50** | Good | Good | Either acceptable |
| **Sample size n 50** | Good | Good | **Prefer K-Fold** (more robust) |
| **Computational cost** | High (K fits) | Low (1 fit) | GCV faster |
| **Interpretability** | High | Moderate | K-Fold clearer |
| **Deterministic** | No (random folds) | Yes | GCV if reproducibility critical |

### E.5.1 Practical Guidance for Cost Estimation

1. **Small samples (n=5-15 lots)**: Use GCV (default in our implementation)
   - Preserves full training data
   - More stable penalty selection
   - Computationally efficient
2. **Medium samples (n=15-30 lots)**: Either method acceptable
   - GCV for consistency with small-sample cases
   - K-Fold (K=5) for more conservative penalty selection
3. **Large samples (n>30 lots)**: Use K-Fold CV (K=10)
   - More robust to model assumptions
   - Direct estimate of out-of-sample error
   - Standard practice in machine learning
4. **When GDF is unreliable**: Fall back to K-Fold CV
   - Non-differentiable loss functions
   - Highly constrained solutions with uncertain active set

## E.6 Implementation in PCReg

The `PenalizedConstrainedCV` class supports both methods:

```
# GCV (default for small datasets)
model_gcv = pcreg.PenalizedConstrainedCV(
    selection='gcv',  # Generalized Cross Validation
    ...
)

# K-Fold CV
model_kfold = pcreg.PenalizedConstrainedCV(
    selection='cv',   # K-Fold Cross Validation
    cv_folds=5,      # Number of folds (default=5)
    ...
)
```

The implementation automatically handles: - GDF calculation for GCV - Stratified fold assignment for K-Fold CV - Grid search over penalty parameter space - Selection of optimal penalty values

### E.7 References for Cross Validation

Key papers on cross validation methods:

- Golub, Heath, and Wahba (1979): Original GCV formulation
- Craven and Wahba (1978): Theoretical foundations of GCV
- Stone (1974): Cross-validatory choice of prediction rules
- Hastie, Tibshirani, and Friedman (2009): Comprehensive treatment in Section 7.10

# F Detailed Diagnostic Report

This appendix provides comprehensive diagnostic information for the PCReg model fit from the motivating example.

## F.1 Model Specification

| Parameter | Value |
|---|---|
| Model Type | PenalizedConstrainedCV |
| Loss Function | sspe |
| Alpha ($\lambda$) | 1.666667 |
| L1 Ratio ($\alpha$) | 0.0000 |
| Fit Intercept | False |
| Optimization Method | SLSQP |
| Convergence Status | Yes |
| GDF Method | gaines |

## F.2 Coefficient Estimates

| Parameter | Estimate | Bootstrap SE | 95% CI Lower | 95% CI Upper | Lower Bound | Upper Bound | Status |
|---|---|---|---|---|---|---|---|
| T1 | 99.9976 | 7.84584 | 82.6878 | 120.569 | 0 | None | free |
| b | -0.052357 | 0.028546 | -0.13784 | -0.028375 | -0.5 | 0.0 | free |
| c | -0.180243 | 0.032862 | -0.207598 | -0.093203 | -0.5 | 0.0 | free |

## F.3 Fit Statistics

| Statistic | Value |
|---|---|
| $R^2$ | 0.900604 |
| Adjusted $R^2$ | 0.872206 |
| SEE | 3.44416 |
| SPE | 0.103307 |
| MAPE | 0.0693 |
| RMSE | 2.88159 |
| CV | 0.087908 |
| GDF | 7 |

## F.4 Constraint Summary

**Number of specified bounds:** 5

**Number of active constraints:** 0

No constraints are active (all parameters are in the interior).

## F.5 Bootstrap Results

**Number of bootstrap samples:** 500

**Confidence level:** 95%

**Successful constrained fits:** 500

**Successful unconstrained fits:** 500

### F.5.1 Bootstrap Summary Statistics

| Parameter | Constrained Mean | Constrained Std | Constrained CI Lower | Constrained CI Upper | Unconstrained Mean | Unconstrained Std | Unconstrained CI Lower | Unconstrained CI Upper |
|---|---|---|---|---|---|---|---|---|
| T1 | 99.477643 | 7.845841 | 82.687838 | 120.569444 | 109.736319 | 15.398629 | 99.996917 | 143.553038 |
| b | -0.064994 | 0.028546 | -0.137840 | -0.028375 | -0.005025 | 0.070858 | -0.178079 | 0.111875 |
| c | -0.163312 | 0.032862 | -0.207598 | -0.093203 | -0.250956 | 0.091588 | -0.398782 | -0.054577 |
| Intercept | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |

## F.6 Data Summary

| Property | Value |
|---|---|
| Number of samples | 10 |
| Number of features | 2 |
| Target mean | 32.7798 |
| Target std | 9.1401 |
| Target min | 21.4632 |
| Target max | 44.8221 |

## F.7 Model Equation

**Custom functional form:**

```
def prediction_fn(X, params):
    T1, b, c = params
    return T1 * (X[:, 0] ** b) * (X[:, 1] ** c)
```

**Model Equation:** `Custom prediction function Parsed: T1 * X[:, 0] ** b * X[:, 1] ** c`