M441 Project 2 - Least Squares

Snow Depth Modeling: Bridger, Montana

Lucas Clouser

2025-11-02

-

Note:

There is an html version of this report hosted at https://m441project2.clouslucas.cc/ that has better code formatting.

Abstract

This project investigates the relationship between snow depth and time throughout the year in the Bridger Range near Bozeman, Montana. Using daily snow depth data, we model seasonal snow accumulation and melt using least squares methods. Linear models are derived from the normal equations and solved using Cholesky, QR, and least-squares decompositions to compare numerical stability and conditioning. Polynomial models are also explored to capture nonlinear seasonal behavior, particularly during the melt period.

The median snow depth during the accumulation period is well modeled by a simple linear trend, indicating a steady and approximately constant rate of snow buildup. During the melt period, a third-order polynomial is used. However, year-to-year variability in snowfall and temperature introduces substantial variance, so no single model achieves a consistently low mean error for specific dates.

Overall, the model serves as a robust *retrospective* description of median snow depth trends but **not** a reliable *predictive* tool for individual snow depth values.

Introduction

Motivation

The SNOTEL station at Brackett Creek, located on the eastern slopes of the Bridger Range near Bozeman, Montana, provides a continuous long-term record of daily snow depth. This site lies just north of the Bridger Bowl Ski Area and serves as a reliable indicator of local snowpack conditions that influence both mountain recreation and watershed behavior.

Seasonal snowpack in the Bridger Range plays a central role in local hydrology, ecology, and recreation. Snow accumulation and melt govern the timing and magnitude of spring runoff that feeds river systems, supporting agriculture, wildlife, and downstream water use. For Bozeman and the surrounding communities, total snow depth also has a strong recreational and economic impact—driving ski and snowboard conditions at Bridger Bowl and shaping the length and quality of the winter season.

Understanding how snow depth evolves across the accumulation and melt seasons in the Bridgers provides valuable insight into both local water availability and the variability of winter conditions. This project focuses on modeling the *shape* of the annual snow depth cycle using least squares regression methods.

From a numerical methods standpoint, the dataset offers a real-world case for analyzing how centering, conditioning, and matrix decomposition methods (Cholesky, QR, and SVD-based least squares) affect the stability and interpretability of regression models.

Research Question

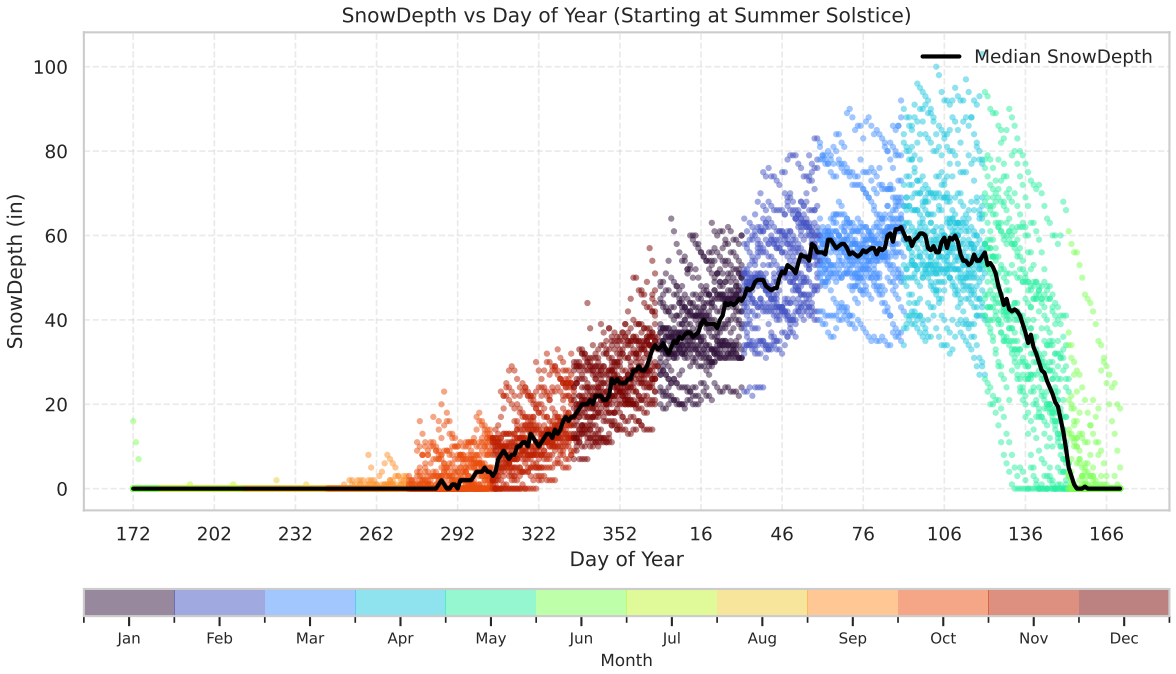
How accurately can least squares line fit regression methods model the seasonal pattern of snow accumulation and melt at the Brackett Creek SNOTEL site in the Bridger Range, and to what extent can these models explain seasonal and daily structure?

Dataset

The dataset used in this analysis was compiled from the Brackett Creek SNOTEL site, which provides daily records of snow depth in inches. Individual CSV files for each calendar year from 2002 through 2022 were downloaded directly from the NRCS website, concatenated, and cleaned to form a single continuous dataset. The resulting file, `combined_snowdepth.csv`, contains two primary columns:

- `Date`: the observation date (YYYY-MM-DD)
- `SnowDepth`: the measured snow depth in inches on that date

Plotting the snowdepth vs day of year with a line representing the daily median snow depth:



The dataset is divided into **seasonal periods** to isolate distinct phases of the snowpack cycle for separate modeling.

The **Cold Period** (days 274–85) corresponds roughly to **late September through late March**, encompassing autumn, winter, and the early part of spring when temperatures remain below freezing and snow steadily accumulates in the Bridger Range. This period represents the buildup of the seasonal snowpack.

The **Melt Period** (days 86–165) spans approximately **late March through mid-June**, covering the transition from spring into early summer. During this time, rising temperatures lead to sustained melting of the snowpack, resulting in a rapid decline in total snow depth.

Hypotheses

These hypotheses were developed following exploratory modeling of the dataset:

H1 (Accumulation—Linear):
During the Cold Period (late Sep–late Mar), the **median** snow depth vs. day-of-period is well approximated by a **linear** model, yielding strong fit on medians.

H2 (Melt—Cubic):
During the Melt Period (late Mar–mid Jun), the **median** snow depth vs. day-of-period is better captured by a **third-order polynomial** than a linear model.

H3 (Seasonal structure vs. day-specific prediction):
Models will **explain seasonal structure** (shape and timing of accumulation/melt in medians) but **not** provide **accurate day-specific predictions** across years.

Creating Models

Linear

The goal was to see whether a linear relationship between time (day within the cold season) and snow depth could capture the general accumulation pattern.
Intuitively, if storms and weather patterns are roughly consistent from year to year, we'd expect snow depth to increase at a fairly constant rate over this period.

Using least squares regression, the model finds the line that minimizes the total squared difference between observed snow depths and the predicted ones.
This produces an equation of the form:

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

where β_0 represents the estimated snow depth at the start of the cold season and β_1 represents the average rate of accumulation per day.
In other words, β_1 is the slope of the line showing how quickly snow builds up through the winter months.

Design Matrix and Normal Equations

To model snow depth as a linear function of time within a seasonal period, we start with the basic linear model:

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

where:

- y is the snow depth measurement
- x is the day number within the seasonal period (e.g., days since start of cold season)
- β_0 is the intercept (snow depth at day 0 of the period)
- β_1 is the slope (rate of snow accumulation/melt)
- ε is the error term

For n data points (x_i, y_i), we can write this in matrix form:

$$\mathbf{y} = \mathbf{A}\beta + \varepsilon$$

where:

- $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}^T$ is the response vector
- $\beta = \begin{bmatrix} \beta_0 & \beta_1 \end{bmatrix}^T$ is the parameter vector
- $\mathbf{A} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$ is the **design matrix**

The least squares solution minimizes the sum of squared errors:

$$S(\beta) = \|\mathbf{y} - \mathbf{A}\beta\|^2 = (\mathbf{y} - \mathbf{A}\beta)^T(\mathbf{y} - \mathbf{A}\beta)$$

$$S(\beta) = \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{A}\beta - \beta^T\mathbf{A}^T\mathbf{y} + \beta^T\mathbf{A}^T\mathbf{A}\beta$$

Since $\mathbf{y}^T\mathbf{A}\beta$ is a scalar, it equals its transpose:

$$\mathbf{y}^T\mathbf{A}\beta = (\mathbf{y}^T\mathbf{A}\beta)^T = \beta^T\mathbf{A}^T\mathbf{y}$$

Therefore:

$$S(\beta) = \mathbf{y}^T\mathbf{y} - 2\beta^T\mathbf{A}^T\mathbf{y} + \beta^T\mathbf{A}^T\mathbf{A}\beta$$

Now take the derivative with respect to β:

$$\frac{\partial S}{\partial \beta} = -2\mathbf{A}^T\mathbf{y} + 2\mathbf{A}^T\mathbf{A}\beta$$

Set the derivative to zero to find the minimum:

$$-2\mathbf{A}^T\mathbf{y} + 2\mathbf{A}^T\mathbf{A}\beta = 0$$

Divide by 2:

$$-\mathbf{A}^T\mathbf{y} + \mathbf{A}^T\mathbf{A}\beta = 0$$

Rearrange to obtain the **normal equations**:

$$\boxed{\mathbf{A}^T\mathbf{A}\beta = \mathbf{A}^T\mathbf{y}}$$

Numerical Stablity

When x spans 0–365, $\mathbf{A}^T\mathbf{A}$ becomes ill-conditioned because the intercept column (all 1s) and day number column are nearly orthogonal, creating large off-diagonal elements. Centering the x values by subtracting the mean reduces the correlation between columns, making $\mathbf{A}^T\mathbf{A}$ better conditioned while preserving the slope estimate β_1.

$$x_{\text{centered}} = x - \bar{x}$$

When applying the model we simply need to subtract the same centering constant $\bar{x}$ from any new prediction day x_{new} before computing:

$$\hat{y} = \beta_0 + \beta_1(x_{\text{new}} - \bar{x})$$

This ensures the centered day scale used during model fitting is maintained for predictions.

Solvers

Cholesky

The Cholesky decomposition exploits the fact that $\mathbf{A}^T\mathbf{A}$ is symmetric positive definite to efficiently solve the normal equations.

Process:

1. **Decompose**: Factor $\mathbf{A}^T\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2. **Forward substitution**: Solve $\mathbf{L}\mathbf{z} = \mathbf{A}^T\mathbf{y}$ for $\mathbf{z}$
3. **Backward substitution**: Solve $\mathbf{L}^T\beta = \mathbf{z}$ for β

QR

The QR decomposition avoids forming the normal equations entirely by working directly with the design matrix $\mathbf{A}$.

Process:

1. **Decompose**: Factor $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is orthogonal and $\mathbf{R}$ is upper triangular
2. **Transform**: Compute $\mathbf{Q}^T\mathbf{y}$
3. **Backward substitution**: Solve $\mathbf{R}\beta = \mathbf{Q}^T\mathbf{y}$ for β

Numpy LSTSQ

The `lstsq` function uses NumPy's optimized least-squares routine, which typically employs singular value decomposition (SVD) internally for robust solution of the normal equations.

Analysis

We apply solve for the linear line of best fit using these methods for the 'Cold' period.

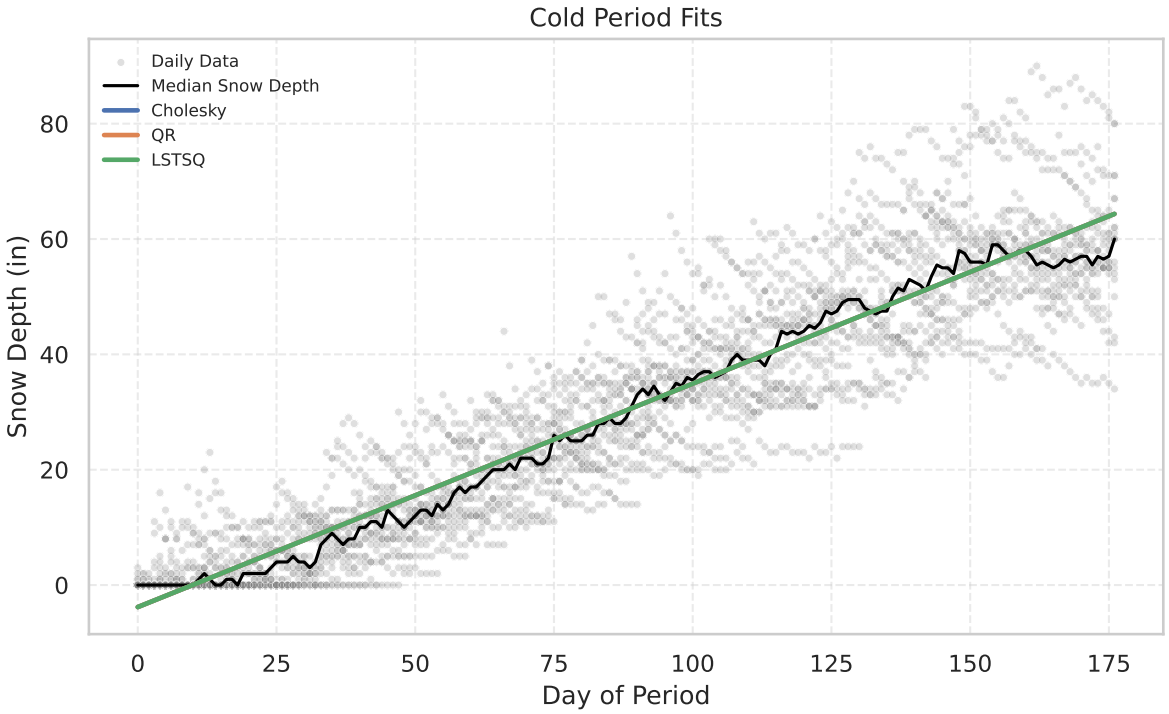
Condtioning

We compute the condition numbers as follows:

- Cholesky uncentered: 3.97e+04
- Cholesky centered: 2.60e+03
- QR uncentered: 1.99e+02
- QR centered: 5.10e+01
- LSTSQ uncentered: 1.99e+02
- LSTSQ centered: 5.10e+01

We can see that the conditioning improves if we center the data on x. Proceeding all data will be centered for modeling unless otherwise noted.

Ploting



As we can each of our numerical methods produce the same, reasonable, line of best fit. Discussion of the model viability is in the conclustion section.

Polynomial

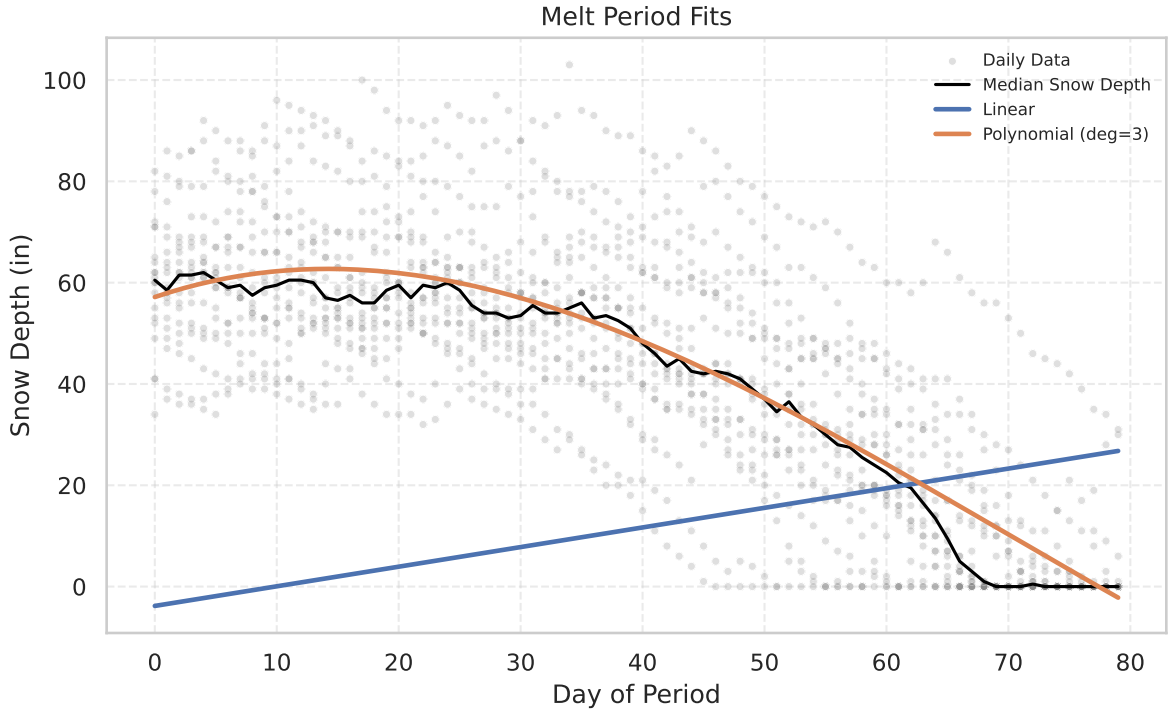
Up to this point, the linear model worked well for describing the **Cold Period**, when snow generally accumulates at a fairly steady rate.
However, once temperatures start to rise in late March, the relationship between time and snow depth becomes much less linear.
Melting tends to start slow, accelerate through spring, and then taper off near the end of the melt season.
To capture this kind of curved behavior, we introduced a **polynomial model** for the Melt Period.

Method

The polynomial model assumes that snow depth can be expressed as a higher-order function of time (day within the melt period):

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_n x^n$$

In practice, we used `numpy.polyfit` to fit the data for the Melt Period, which returns the coefficients that minimize the least-squares error between the predicted and observed snow depths.



Analysis

The cubic polynomial fit showed a clear improvement over the linear model during the Melt Period.

This nonlinearity matches the physical process pretty well. Early in the melt season, daily highs barely reach freezing, so the rate of snow loss is small. As sunlight increases and temperatures climb, the melt rate accelerates rapidly, which corresponds to the steep part of the polynomial curve. Finally, once most of the snowpack is gone, remaining shaded or compacted snow patches melt more slowly, leading to a flattening of the curve near the end.

Even though the polynomial fit did a better job describing the overall shape of the melt period, there was still noticeable year-to-year variation around the median. That variation is driven by random storm events and temperature swings that the model can't capture. So while the polynomial version improves the general seasonal trend, it still doesn't provide much predictive power for any specific day.

Summary of Results

Fit Metrics

Each regression model is plotted as a continuous line representing its predicted snow-depth trend.

For each model, we compute quantitative goodness-of-fit metrics against both the full daily dataset and the daily medians:

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}, \quad \text{RMSE} = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}, \quad \text{MAE} = \frac{1}{n}\sum|y_i - \hat{y}_i|$$

These metrics quantify how well each model explains observed variation and how large its average prediction error is.

We also visualizes **model uncertainty** by computing a daily residual spread:

$$c(\text{day}) = \text{median}(|y_{\text{obs}} - \hat{y}|)$$

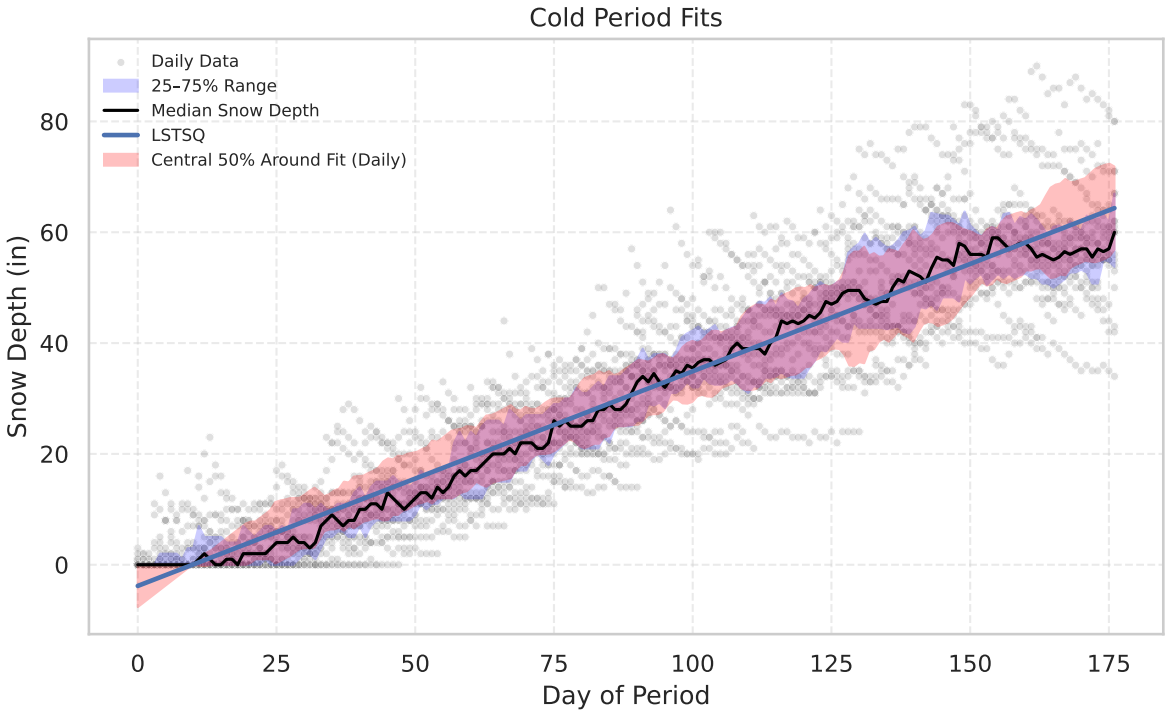
It then plots a red symmetric band around each fitted line, bounded by

$$\hat{y} \pm c(\text{day}),$$

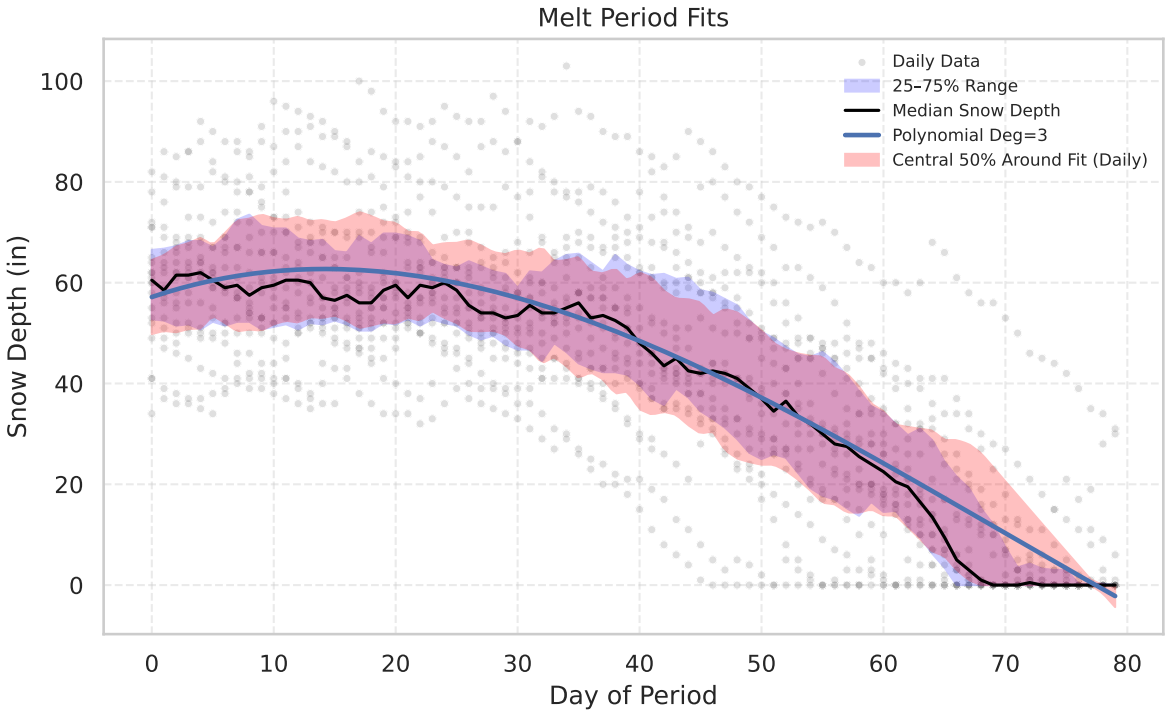
representing the range within which roughly half of all data points fall for each day.
This *central-50 % residual band* functions as a **robust, non-parametric confidence region** — showing day-by-day model variability without assuming normal residuals.

Visually:

- The **blue IQR band** shows the *empirical variability* of observed data.
- The **red residual band** shows the *model's typical deviation* from observations.
- When the red band lies *within* the IQR, model errors are smaller than natural day-to-day variability — indicating a good fit.
- When it extends *beyond* the IQR, the model under- or over-predicts systematically for that time of year.



	LSTSQ	
	Full	Median
R2	0.843	0.983
RMSE	8.540	2.623
MAE	6.565	2.146



	Polynomial Deg=3	
	Full	Median
R2	0.629	0.963
RMSE	15.814	4.215
MAE	12.142	3.099

Our analysis validates the seasonal hypotheses while revealing critical limitations in predictive capability. **H1** and **H2** are strongly supported when evaluating median trends, with both models achieving $R^2 > 0.96$ against median snow depth. However, the same models perform substantially worse on the full dataset, with R^2 values dropping to 0.84 and 0.63 respectively for the Cold and Melt periods.

This divergence in metrics exposes the fundamental nature of our model: it excels at capturing seasonal structure but fails completely at day-specific prediction. The wide central 50% bands around our fits, often spanning 20-40 inches, demonstrate that for any given day, the model's prediction is statistically indistinguishable from simply using the daily median. The regression line essentially smooths what we could already see by connecting median values.

Ultimately, we've built a mathematically sound descriptive model, not a predictive one. While it accurately characterizes the average seasonal pattern from the past two decades, it fails as a forecasting tool for any given season. This is because our approach aggregates data across years,

erasing the unique sequence and dependencies that define a single season's evolution. The high variance between years, driven by chaotic weather systems, ensures that a simple line fit to aggregated data cannot overcome the fundamental unpredictability of future conditions.

Taking the median of the data is often more robust, as it directly describes the empirical trend of the observations without assuming a specific model form.
However, fitting a regression model can become computationally advantageous when the dataset is large enough that the $O(N \log N)$ cost of sorting for the median exceeds the $O(N)$ cost of fitting a linear model.
In such cases, the regression must still be verified to approximate the median trend well, rather than being dominated by outliers or asymmetrical data.
Alternatively, using a **weighted mean** or robust regression approach can provide a similar effect by down-weighting the influence of outliers while maintaining the efficiency and smoothness of a fitted model.

Conclusions

Key Takeaways about Curve Fitting

Through this project, we gained several important insights about least squares modeling and its practical applications:

1. **Data preprocessing**: We found that gathering, alinging and cleaning the data is half the battle.
2. **The solving method for least squares is not so important**: We implemented three different solution methods (Cholesky, QR, and SVD-based), and when our problem was well-conditioned, they all gave us identical results. This was reassuring—it showed that the mathematics is consistent regardless of which computational route we take. In practice I will normally use a regression from a library, though it is useful to know what the linear algebra under the hood is.
3. **Some data is inherently unsuited for precise regression:** Although we can see a general pattern in the data, the high variance between years means that combining the data across years results in a model that cannot accurately represent any single year's unique conditions. The "best fit" is, by definition, a compromise that doesn't perfectly fit any individual season. This taught us a vital lesson about the limitations of our tools: regression can describe an average trend, but it cannot erase the beautiful, messy variability of the natural world.

Interpreting Curve Fits in the Wild

Now that we've built models ourselves, we look at curve fits in news articles and online content with more critical eyes. Here's what we've learned to watch for:

First, we ask about uncertainty: Does the graph show confidence intervals or residual bands? Our snow depth models had those wide central-50% bands that reminded us how much natural variation exists. When media presentations show only the smooth fit line, they're often hiding this important context.

We think about model choice: Why was a particular curve form chosen? Our experience with the melt period taught us that the choice between linear and polynomial isn't arbitrary—it should reflect the underlying process. When we see curves in media, we now wonder if the shape makes physical sense.

We're skeptical of over-interpretation: Our high R^2 values on median data didn't translate to precise daily predictions. Similarly, when we see impressive statistics in media graphics, we consider whether they're describing general trends or claiming precise predictive power.

Apendix

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import seaborn as sns
from numpy.linalg import cholesky, qr, cond, lstsq, solve
from scipy.optimize import curve_fit
from sage.all import *

def plot_snowdepth_shifted(df, solstice_day=172, ycol='SnowDepth'):
    """
    Plot snow depth vs. rebased day-of-year (starting at summer solstice),
    including the median snow depth for each day.

    Parameters
    ----------
    df : DataFrame
        Input dataset containing 'Date', 'Month', 'DayOfYear', 'SnowDepth',
        and 'MedianSnowDepth' columns.
    solstice_day : int, optional
        Day-of-year for the summer solstice (default 172).
    ycol : str, optional
        Column name to plot on the y-axis (default 'SnowDepth').

    Returns
    -------
    None
        Displays a scatter plot with median line and month colorbar.
    """

    work = df.copy()
    work['DayShifted'] = (work['DayOfYear'] - solstice_day) % 366
    work = work.sort_values('DayShifted')

    # Discrete colormap by month
    cmap = plt.cm.turbo
    month_colors = cmap(np.linspace(0, 1, 12))
    norm = mcolors.BoundaryNorm(np.arange(1, 14), 12)
```

```
    # Plot
    plt.figure(figsize=(10, 6))
    sc = plt.scatter(work['DayShifted'], work[ycol],
                     c=work['Month'],
                     cmap=mcolors.ListedColormap(month_colors),
                     norm=norm,
                     s=14, alpha=0.5, edgecolor='none')

    # Median Line (using precomputed MedianSnowDepth)
    median_by_day = work.groupby('DayShifted')['MedianSnowDepth'].first()
    plt.plot(median_by_day.index, median_by_day.values,
             color='black', linewidth=2.5,
             label='Median {}'.format(ycol))

    # Labels and grid
    plt.xlabel('Day of Year')
    plt.ylabel(ycol + ' (in)')
    plt.title('{} vs Day of Year (Starting at Summer Solstice)'.format(ycol))
    plt.grid(True, linestyle='--', alpha=0.4)
    plt.legend(frameon=False, loc='upper right')

    # Horizontal colorbar for months
    cbar = plt.colorbar(sc, orientation='horizontal', pad=0.12, aspect=40)
    cbar.set_ticks(np.arange(1.5, 13.5))
    cbar.set_ticklabels(['Jan','Feb','Mar','Apr','May','Jun',
                         'Jul','Aug','Sep','Oct','Nov','Dec'])
    cbar.ax.tick_params(labelsize=9)
    cbar.set_label('Month', fontsize=10)

    # X-axis ticks labeled by true day-of-year
    xticks = np.arange(0, 366, 30)
    plt.xticks(xticks, [str(int((solstice_day + x) % 366)) for x in xticks])

    plt.tight_layout()
    plt.show()


def add_period(df, name, start, end, col_prefix=None):

    """
    Add boolean and day-count columns for a named seasonal period.
```

```
    Parameters
    ----------
    df : DataFrame
        Input dataset with 'Date' and 'DayOfYear' columns.
    name : str
        Name of the period (e.g., 'Cold', 'Melt').
    start, end : int or str
        Start and end of the period as day-of-year integers or ISO date strings.
    col_prefix : str, optional
        Optional column name prefix override.

    Returns
    -------
    DataFrame
        Original DataFrame with '<prefix>_InPeriod' and '<prefix>_Day' columns.
    """

    prefix = col_prefix or name

    # Absolute date mode
    if isinstance(start, str) and isinstance(end, str):
        start_date = pd.to_datetime(start)
        end_date = pd.to_datetime(end)

        if end_date >= start_date:
            in_period = (df['Date'] >= start_date) & (df['Date'] <= end_date)
            day_val = (df['Date'] - start_date).dt.days
        else:
            in_period = (df['Date'] >= start_date) | (df['Date'] <= end_date)
            day_val = ((df['Date'] - start_date).dt.days) % 365

    # Day-of-year mode
    elif isinstance(start, (int, np.integer)) and isinstance(end, (int, np.integer)):
        # Normalize within [1, 365]
        start = int(start) % 365 or 365
        end = int(end) % 365 or 365

        # Inclusive cyclic logic
        if start <= end:
            in_period = (df['DayOfYear'] >= start) & (df['DayOfYear'] <= end)
            day_val = df['DayOfYear'] - start
        else:
```

```
            # Wrap-around case (e.g. 274 -> 90)
            in_period = (df['DayOfYear'] >= start) | (df['DayOfYear'] <= end)
            # Ensure continuous day count across boundary
            day_val = (df['DayOfYear'] - start) % 365
    else:
        raise ValueError("start and end must both be ints (DayOfYear) or both be strings (abs

    # Add columns to df
    df[f'{prefix}_InPeriod'] = in_period
    df[f'{prefix}_Day'] = np.where(in_period, day_val, np.nan)

    return df

def get_period_data(df, period_name):

    """
    Extract X and y arrays for a given period.

    Parameters
    ----------
    df : DataFrame
        Dataset with '<Period>_InPeriod' and '<Period>_Day' columns.
    period_name : str
        Period name (e.g., 'Cold', 'Melt').

    Returns
    -------
    tuple
        (X, y, fallback_model) where fallback_model returns NaNs if empty.
    """

    in_col = f"{period_name}_InPeriod"
    day_col = f"{period_name}_Day"

    # Ensure required columns exist
    for col in [in_col, day_col, "SnowDepth"]:
        if col not in df.columns:
            raise KeyError(f"Missing required column '{col}' in DataFrame")

    # Subset valid data
    df_period = df[df[in_col]].dropna(subset=[day_col, "SnowDepth"])
```

```
    if df_period.empty:
        print(f"No data found for period '{period_name}'")
        return None, None, lambda x: np.full_like(np.asarray(x), np.nan, dtype=float)

    X = df_period[day_col].values
    y = df_period["SnowDepth"].values

    return X, y, None

def make_line_model(coeffs, X_mean=0.0):
    """
    Create a simple linear model function from coefficients.

    Parameters
    ----------
    coeffs : array_like
        [intercept, slope]
    X_mean : float, optional
        Mean of X used for centering (default 0.0)
    centered : bool, optional
        Whether to subtract X_mean before predicting (default True)

    Returns
    -------
    function
        model(x) -> predicted y
    """
    a, b = coeffs

    return lambda x: a + b * (np.asarray(x, dtype=float) - X_mean)


def plot_period_fits(df, period_name, fit_functions, labels=None,
                     title=None, show_bands=False):
    """
    Seaborn-based plot with a *daily symmetric central-50% band* around each fit:
    for each day-of-period, compute c(day) = median(|residual|) and plot y_fit ± c(day).
    This yields ~half of the data around the fit line for each day.
    """

    in_col, day_col = f"{period_name}_InPeriod", f"{period_name}_Day"
    df_period = df[df[in_col] & df[day_col].notna()].copy()
```

```
    if df_period.empty:
        print(f"No data found for period '{period_name}'")
        return {}

    X = df_period[day_col].values
    y = df_period["SnowDepth"].values

    # --- Median & Quartiles (for context background band) ---
    q25 = df_period.groupby(day_col)["SnowDepth"].quantile(0.25)
    q75 = df_period.groupby(day_col)["SnowDepth"].quantile(0.75)
    med  = df_period.groupby(day_col)["SnowDepth"].median()
    median_df = pd.DataFrame({day_col: med.index, "q25": q25.values, "q75": q75.values, "medi
    X_med, y_med = median_df[day_col].values, median_df["median"].values

    # --- Plot setup ---
    plt.figure(figsize=(8, 5))
    sns.set_theme(style="whitegrid")

    sns.scatterplot(data=df_period, x=day_col, y="SnowDepth",
                    color="gray", alpha=0.25, s=15, label="Daily Data")

    if show_bands:
        plt.fill_between(median_df[day_col], median_df["q25"], median_df["q75"],
                         color="blue", alpha=0.2, label="25-75% Range")

    plt.plot(median_df[day_col], median_df["median"],
             color="black", linewidth=1.5, label="Median Snow Depth")

    # --- Fit lines & metrics ---
    metrics = {}
    x_fit = np.linspace(X.min(), X.max(), 200)

    for i, f in enumerate(fit_functions):
        label = labels[i] if labels and i < len(labels) else f"Model {i+1}"

        y_pred_full = f(X)
        y_pred_med  = f(X_med)
        y_fit       = f(x_fit)

        # --- Metrics ---
        def _metrics(y_true, y_pred):
            ss_res = np.sum((y_true - y_pred) ** 2)
```

```
            ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)
            r2  = 1 - ss_res / ss_tot if ss_tot != 0 else np.nan
            rmse = np.sqrt(np.mean((y_true - y_pred) ** 2))
            mae  = np.mean(np.abs(y_true - y_pred))
            return r2, rmse, mae

        r2_full, rmse_full, mae_full = _metrics(y, y_pred_full)
        r2_med,  rmse_med,  mae_med  = _metrics(y_med, y_pred_med)

        metrics[label] = {
            "Full":   {"R2": r2_full, "RMSE": rmse_full, "MAE": mae_full},
            "Median": {"R2": r2_med,  "RMSE": rmse_med,  "MAE": mae_med},
        }

        plt.plot(x_fit, y_fit, lw=2.2, label=f"{label}")

        # --- Daily symmetric central-50% band around the fit ---
        if show_bands:
            df_fit = df_period.copy()
            df_fit["Residual"] = df_fit["SnowDepth"] - f(df_fit[day_col])

            # For each day: c(day) = median(|residual|)
            mad50 = (
                df_fit.assign(abs_res=lambda d: np.abs(d["Residual"]))
                      .groupby(day_col)["abs_res"]
                      .quantile(0.5)  # 50th percentile of |residual|
                      .reset_index()
                      .rename(columns={"abs_res": "c"})
            )

            # Interpolate c(day) over x_fit for smooth band
            c_interp = np.interp(x_fit, mad50[day_col], mad50["c"])

            # Symmetric band: y_fit ± c(day)
            plt.fill_between(
                x_fit, y_fit - c_interp, y_fit + c_interp,
                color="red", alpha=0.25,
                label=None if i > 0 else "Central 50% Around Fit (Daily)"
            )

    plt.xlabel("Day of Period")
    plt.ylabel("Snow Depth (in)")
```

```
    plt.title(title or f"{period_name} Period Fits")
    plt.legend(frameon=False, fontsize=8)
    plt.grid(True, linestyle="--", alpha=0.4)
    plt.tight_layout()
    plt.show()

    return metrics

def forward_substitution(L, b):

    """
    Solve L y = b for y, where L is lower-triangular.

    Parameters
    ----------
    L : (n, n) array_like
        Lower-triangular matrix.
    b : (n,) array_like
        Right-hand side vector.

    Returns
    -------
    y : ndarray
        Solution vector satisfying L y = b.
    """
    L = np.asarray(L, dtype=float)
    b = np.asarray(b, dtype=float)
    n = L.shape[0]
    y = np.zeros_like(b)

    for i in range(n):
        y[i] = (b[i] - np.dot(L[i, :i], y[:i])) / L[i, i]
    return y

def backward_substitution(U, y):
    """
    Solve U x = y for x, where U is upper-triangular.

    Parameters
    ----------
    U : (n, n) array_like
        Upper-triangular matrix.
```

```
    y : (n,) array_like
        Right-hand side vector.

    Returns
    -------
    x : ndarray
        Solution vector satisfying U x = y.
    """
    U = np.asarray(U, dtype=float)
    y = np.asarray(y, dtype=float)
    n = U.shape[0]
    x = np.zeros_like(y)

    for i in range(n - 1, -1, -1):
        x[i] = (y[i] - np.dot(U[i, i + 1:], x[i + 1:])) / U[i, i]
    return x

# Load Data Set
df = pd.read_csv('data/combined_snowdepth.csv', parse_dates=['Date'])
df['Month'] = df['Date'].dt.month
df['Year']  = df['Date'].dt.year
df['DayOfYear'] = df['Date'].dt.dayofyear

# Compute median snow depth by day of year
median_by_day = df.groupby('DayOfYear')['SnowDepth'].median().rename('MedianSnowDepth')
df = df.merge(median_by_day, on='DayOfYear', how='left')

plot_snowdepth_shifted(df)

# By day-of-year
df = add_period(df, 'Cold', 274, 85)
df = add_period(df, 'Melt', 86, 165)

def build_normal_equations(df, period_name, center=True):

    # Get data
    X, y, empty_model = get_period_data(df, period_name)

    # Convert to numeric arrays
    X = np.asarray(X, dtype=float).ravel()
    y = np.asarray(y, dtype=float).ravel()
```

```
    # Center
    X_mean = np.mean(X) if center else 0.0
    x = X - X_mean

    # Linear design matrix: [1, x]
    A = np.column_stack((np.ones_like(x), x))

    # Normal equations
    ATA = A.T @ A
    ATy = A.T @ y

    return ATA, ATy, X_mean, A, y

def solve_cholesky(df, period_name, centered=True):
    # Build normal equations
    ATA, ATy, X_mean, A, y = build_normal_equations(df, period_name, center=centered)
    if ATA is None:
        return np.array([np.nan, np.nan]), np.nan, np.nan

    cond_number = cond(ATA)

    # Cholesky decomposition
    L = cholesky(ATA)

    # Solve via substitution
    y_temp = forward_substitution(L, ATy)
    coeffs = backward_substitution(L.T, y_temp)

    return coeffs, cond_number, X_mean

def solve_qr(df, period_name, centered=True):

    # Build normal equations (gets design matrix + mean)
    ATA, ATy, X_mean, A, y = build_normal_equations(df, period_name, center=centered)
    if ATA is None:
        return np.array([np.nan, np.nan]), np.nan, np.nan

    # QR decomposition (A = Q R)
    Q, R = np.linalg.qr(A)

    # Solve R   = Q  y
    Qt_y = Q.T @ y
```

```
    coeffs = backward_substitution(R, Qt_y)

    # Condition number from R
    cond_number = np.linalg.cond(R)

    return coeffs, cond_number, X_mean

def solve_lstsq(df, period_name, centered=True):

    # Build design matrix and vectors
    ATA, ATy, X_mean, A, y = build_normal_equations(df, period_name, center=centered)

    # Solve using NumPy's least-squares
    coeffs, _, _, _ = lstsq(A, y, rcond=None)

    # Compute condition number
    cond_number = np.linalg.cond(A)

    return coeffs, cond_number, X_mean

# Build predictor functions
f_chol = make_line_model(chol_coeffs, chol_mean)
f_qr   = make_line_model(qr_coeffs, qr_mean)
f_ls   = make_line_model(lstsq_coeffs, lstsq_mean)

# Plot all together
m = plot_period_fits(df, "Cold", [f_chol, f_qr, f_ls], ["Cholesky", "QR", "LSTSQ"])

def make_polyfit_model(df, period_name, degree=2):

    # Get X and y
    X, y, empty_model = get_period_data(df, period_name)

    # Fit polynomial
    coeffs = np.polyfit(X, y, degree)

    # Predictor function
    model = lambda x_new: np.polyval(coeffs, np.asarray(x_new, dtype=float))

    return model

# Fit the linear model for melt
```

```
melt_coeffs, _, melt_mean = solve_lstsq(df, 'Cold')
f_lin_melt = make_line_model(melt_coeffs, melt_mean)
# Fit polynomial model for melt
f_poly_melt = make_polyfit_model(df, "Melt", degree=3)

# Plot
melt_poly_metrics = plot_period_fits(df, "Melt", [f_lin_melt,f_poly_melt], ["Linear", "Polyno

cold_metrics=plot_period_fits(df, "Cold", [f_ls], ["LSTSQ"], show_bands=True)

pd.concat({k: pd.DataFrame(v) for k, v in cold_metrics.items()}, axis=1).round(3)

#| echo: False
melt_metrics=plot_period_fits(df, "Melt", [f_poly_melt], ["Polynomial Deg=3"], show_bands=Tru

#import pandas as pd
pd.concat({k: pd.DataFrame(v) for k, v in melt_metrics.items()}, axis=1).round(3)
```