Methods for Smoothing Experimental Data in the smooth Program

Technical Documentation

Version 5.4 | October 13, 2025

Contents

Introduction

The `smooth` program implements three sophisticated methods for smoothing experimental data with the capability of simultaneous derivative computation. Each method has specific properties, advantages, and areas of application.

General Smoothing Problem

Experimental data often contains random noise:

```
y_obs(x_i) = y_true(x_i) + ε_i
```

The goal of smoothing is to estimate `y_true` while suppressing `ε_i` and preserving physically relevant signal properties.

Program Structure

```
bash

./smooth [options] data_file
```

Basic Parameters:

- `-m {0|1|2}` - method selection (polyfit|savgol|tikhonov)
- `-n N` - smoothing window size (polyfit, savgol)
- `-p P` - polynomial degree (polyfit, savgol, max 12)
- `-l λ` - regularization parameter (tikhonov)
- `-l auto` - automatic λ selection using GCV (tikhonov)
- `-d` - display first derivative in output (optional)
- `-g` - show detailed grid uniformity analysis (optional)

Note on polynomial degree: Degrees > 6 may generate numerical stability warnings.

Note on derivatives: From version 5.1, first derivative output is optional. Without the `-d` switch, the program outputs only smoothed values. With the `-d` switch, it outputs both smoothed values and first derivatives.

Note on grid analysis: The `-g` flag (added in version 5.2) provides detailed grid uniformity statistics helpful for understanding your data and choosing appropriate smoothing parameters.

What's New in Version 5.4

Major Improvements

Tikhonov Regularization (Hybrid Implementation):

- **Automatic discretization selection:** Method automatically chooses between average coefficient (ratio < 2.5) and local spacing (ratio ≥ 2.5) based on grid uniformity
- **Harmonic mean for better accuracy:** Uses harmonic mean for interval averaging in nearly-uniform grids (more accurate than arithmetic mean)
- **Fixed boundary conditions:** Corrected missing superdiagonal element in local spacing method
- **Improved GCV optimization:** Enhanced with over-fitting penalty and L-curve sanity check for large datasets
- **Better functional computation:** Mathematically correct D^2 discretization matching the matrix formulation

Grid Analysis:

- **Detailed reporting with `-g` flag:** Comprehensive grid uniformity statistics including CV, ratio, spacing details
- **Better recommendations:** Program suggests optimal methods based on detected grid characteristics

API Structure

All smoothing methods have consistent APIs:

```c
// Polyfit
PolyfitResult* polyfit_smooth(double *x, double *y, int n,
                              int window_size, int poly_degree);
void free_polyfit_result(PolyfitResult *result);

// Savitzky-Golay
SavgolResult* savgol_smooth(double *x, double *y, int n,
                            int window_size, int poly_degree);
void free_savgol_result(SavgolResult *result);

// Tikhonov - Hybrid implementation in v5.4
TikhonovResult* tikhonov_smooth(double *x, double *y, int n, double lambda);
void free_tikhonov_result(TikhonovResult *result);
```

Polynomial Fitting (POLYFIT)

Mathematical Foundations

The POLYFIT method uses local polynomial fitting with least squares method in a sliding window.

Problem: For each point x_i, we fit a polynomial of degree p to the surrounding n points:

$$P(x) = a_0 + a_1(x-x_i) + a_2(x-x_i)^2 + ... + a_p(x-x_i)^p$$

Optimization criterion:

$$\min \Sigma[y_j - P(x_j)]^2 \quad \text{for } j \in [i-n/2, i+n/2]$$

Construction of Normal Equations

For polynomial coefficients, we solve a system of linear equations:

$$\begin{bmatrix} \Sigma(x-x_i)^0 & \Sigma(x-x_i)^1 & \dots & \Sigma(x-x_i)^p \\ \Sigma(x-x_i)^1 & \Sigma(x-x_i)^2 & \dots & \Sigma(x-x_i)^{p+1} \\ \vdots & \vdots & \ddots & \vdots \\ \Sigma(x-x_i)^p & \Sigma(x-x_i)^{p+1} & \dots & \Sigma(x-x_i)^{2p} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_p \end{bmatrix} = \begin{bmatrix} \Sigma y(x-x_i)^0 \\ \Sigma y(x-x_i)^1 \\ \vdots \\ \Sigma y(x-x_i)^p \end{bmatrix}$$

where summation is over points in the window around x_i.

Derivative Computation

Derivatives are computed analytically from polynomial coefficients:

```
f(x_i) = a_0
f'(x_i) = a_1
f''(x_i) = 2a_2
```

Edge Handling

At edges, asymmetric windows are used with extrapolation of the fitted polynomial:

```
c

// For point x_k < x_{n/2}
f(x_k) = Σ_{m=0}^p a_m * (x_k - x_{n/2})^m
```

Efficient Implementation

The program uses LAPACK routine `dposv` for solving symmetric positive definite systems at each point:

```
c

// System solution for polynomial coefficients
dposv_(&uplo, &matrix_size, &nrhs, C, &matrix_size, B, &matrix_size, &info);
```

The normal equations matrix is symmetric and positive definite, making `dposv` optimal for this application.

Modularized Implementation

```
c

// polyfit.h
typedef struct {
    double *y_smooth;    // Smoothed values
    double *y_deriv;     // First derivatives
    int n;               // Number of points
    int poly_degree;     // Polynomial degree
    int window_size;     // Window size
} PolyfitResult;
```

Characteristics

Advantages:

- Excellent local approximation
- Analytical computation of derivatives of any order
- Adaptable to changes in curvature
- Good preservation of local extrema
- Works with moderately non-uniform grids

Disadvantages:

- Sensitive to outliers
- Boundary effects at edges
- Possible Runge oscillations for high polynomial degrees (p > 6)
- Numerical instability warnings for degrees > 6

Savitzky-Golay Filter (SAVGOL)

Theoretical Foundations

The Savitzky-Golay filter is an optimal linear filter for smoothing and derivatives based on local polynomial regression. The key innovation is pre-computation of convolution coefficients.

Fundamental principle: For given parameters (window size, polynomial degree, derivative order), there exist universal coefficients c_k such that:

$$f^{(d)}(x_i) = \Sigma_{k=-n_L}^{n_R} c_k \cdot y_{i+k}$$

Key Difference from POLYFIT Method

While both SAVGOL and POLYFIT use polynomial approximation, they differ fundamentally in their computational approach:

POLYFIT approach:

- For each data point, fits a new polynomial to the surrounding window
- Solves the least squares problem individually for each point
- Coefficients of the polynomial change with each window position
- Computationally intensive: $O(n \cdot p^3)$

SAVGOL approach (Method of Undetermined Coefficients):

- Recognizes that for equidistant grids, the filter coefficients are translation-invariant
- Uses the **method of undetermined coefficients** to pre-compute universal weights
- These weights depend only on the window geometry, not on the actual data values
- Applies the same weights as a linear convolution across all data points
- Computationally efficient: $O(p^3)$ once, then $O(n \cdot w)$ for application

CRITICAL: Grid Uniformity Requirement

Version 5.3+ Important Feature: The Savitzky-Golay method now enforces grid uniformity checking.

The mathematical foundation of SG filter assumes **uniformly spaced data points**. The method is based on fitting polynomials in normalized coordinate space where points are at integer positions: {..., -2, -1, 0, 1, 2, ...}.

Uniformity Check:

```
CV = std_dev(spacing) / avg(spacing)

If CV > 0.05:  REJECT - Grid too non-uniform for SG
If CV > 0.01:  WARNING - Nearly uniform, proceed with caution
If CV ≤ 0.01:  OK - Grid sufficiently uniform
```

What happens when grid is rejected:

```
========================================
ERROR: Savitzky-Golay method not suitable for non-uniform grid!
========================================
Grid analysis:
  Coefficient of variation (CV) = 0.2341
  Threshold for uniformity = 0.0500

RECOMMENDED ALTERNATIVES:
  1. Use Tikhonov method: -m 2 -l auto
     (Works correctly with non-uniform grids)
  2. Use Polyfit method: -m 0 -n 5 -p 2
     (Local fitting, less sensitive to spacing)
  3. Resample your data to uniform grid before smoothing
```

The Method of Undetermined Coefficients

The Savitzky-Golay method seeks a linear combination of data points:

$$\hat{y}_0 = c_{-nl} \cdot y_{-nl} + \ldots + c_0 \cdot y_0 + \ldots + c_{nr} \cdot y_{nr}$$

where the coefficients c_k are "undetermined" and must satisfy the condition that the filter exactly reproduces polynomials up to degree p.

The key insight: For a given window configuration and polynomial degree, these coefficients can be determined once and applied universally - but only on uniform grids!

Coefficient Derivation

Coefficients are derived from the condition that the filter must exactly reproduce polynomials up to degree p.

Moment conditions:

$$\Sigma_{j=-n_L}^{n_R} c_j \cdot j^m = \delta_{m,d} \cdot d! \quad \text{for } m = 0,1,\ldots,p$$

where:

- `δ_{m,d}` is the Kronecker delta
- `d` is the derivative order
- `d!` is factorial

This leads to a system of linear equations where the unknowns are the filter coefficients `c_j`.

Matrix Formulation

We solve a system of linear equations:

```
[1   -n_L    (-n_L)²   ...  (-n_L)^p  ] [c_{-n_L}]   [δ_{0,d}·0!]
[1  -n_L+1  (-n_L+1)² ... (-n_L+1)^p ] [c_{-n_L+1}] = [δ_{1,d}·1!]
[ ⋮     ⋮        ⋮     ⋱.      ⋮     ] [    ⋮    ]   [    ⋮     ]
[1    n_R     n_R²     ...    n_R^p   ] [  c_{n_R} ]   [δ_{p,d}·p!]
```

Computational Efficiency

The brilliance of the Savitzky-Golay approach becomes apparent when processing large datasets:

Example for 10,000 data points, window size 21, polynomial degree 4:

- **POLYFIT:** Must solve 10,000 separate 5×5 linear systems
- **SAVGOL:** Solves only ONE 5×5 system, then performs 10,000 simple weighted sums

This difference explains why SAVGOL is preferred for real-time signal processing and large datasets, while maintaining the same mathematical accuracy as POLYFIT **for uniform grids**.

Efficient Implementation

The program uses LAPACK routine `dposv` for solving the symmetric positive definite system when computing filter coefficients:

```c
// Solve linear system for Savitzky-Golay coefficients
dposv_(&uplo, &matrix_size, &nrhs, A, &matrix_size, B, &matrix_size, &info);
```

Modularized Implementation

```c
```

```c
// savgol.h
typedef struct {
    double *y_smooth;    // Smoothed values
    double *y_deriv;     // First derivatives
    int n;               // Number of points
    int poly_degree;     // Polynomial degree
    int window_size;     // Window size
} SavgolResult;

// Coefficient computation
void savgol_coefficients(int nl, int nr, int poly_degree,
                         int deriv_order, double *c);
```

Optimal Properties

The Savitzky-Golay filter minimizes approximation error in the least squares sense and maximizes signal-to-noise ratio for polynomial signals **on uniform grids**.

Characteristics

Advantages:

- Optimal for polynomial signals on uniform grids
- Excellent preservation of moments and peak areas
- Efficient implementation (convolution)
- Minimal phase distortion
- Simultaneous computation of functions and derivatives

Disadvantages:

- **Requires uniform grid** - automatically rejected if CV > 0.05
- Fixed coefficients for entire window
- May introduce oscillations at sharp edges
- Limited adaptability
- Numerical warnings for degrees > 6

Tikhonov Regularization (TIKHONOV)

Theoretical Foundation

Tikhonov regularization solves the ill-posed inverse smoothing problem using a variational approach. We seek a function minimizing the functional:

Continuous formulation:

$$J[u] = \int (y(x) - u(x))^2 \, dx + \lambda \int (u''(x))^2 \, dx$$

Data fidelity term Smoothness penalty

Discrete formulation:

$$J[u] = ||y - u||^2 + \lambda ||D^2 u||^2$$

where:

- $||y - u||^2$ = $\Sigma(y_i - u_i)^2$ is the **data fidelity term**
- $||D^2 u||^2$ = $\Sigma(D^2 u_i)^2$ is the **regularization term** (smoothness penalty)
- λ is the **regularization parameter** controlling the balance
- D^2 is the discrete second derivative operator

The Regularization Parameter λ

The parameter λ is the **heart of Tikhonov regularization -** it controls the balance between fitting the data and smoothing the result.

Physical Interpretation

λ = 0: No smoothing, u = y (exact data fit)
  J[u] = ||y - u||² only

λ → ∞: Maximum smoothing, u → straight line
  J[u] ≈ λ||D²u||² dominates

λ optimal: Balanced between data fit and smoothness
  Both terms contribute meaningfully

Mathematical Role

The minimization of J[u] leads to:

$$(I + \lambda D^{\wedge}TD)u = y$$

Effect of λ on the solution:

- **Small λ (< 0.01):** Matrix ≈ I → solution u ≈ y (minimal smoothing)
- **Large λ (> 1.0):** Matrix ≈ λD^TD → strong curvature penalty (heavy smoothing)

- **Optimal λ:** Matrix components balanced → noise removed, signal preserved

Frequency Domain Interpretation

In Fourier space, Tikhonov acts as a low-pass filter:

$$\hat{H}(\omega) = 1 / (1 + \lambda\omega^4)$$

where ω is spatial frequency.

Effect:

- **Low frequencies (slow variations):** $\hat{H} \approx 1$ → preserved
- **High frequencies (noise, rapid variations):** $\hat{H} \approx 1/(\lambda\omega^4)$ → attenuated
- **Cutoff frequency:** ω_c ∝ λ^(-1/4)

This means:

Larger λ → Lower cutoff → More aggressive low-pass filtering → Smoother result
Smaller λ → Higher cutoff → Less filtering → Result closer to data

Practical Guidelines for λ Selection

1. Automatic Selection (RECOMMENDED):

```
bash

./smooth -m 2 -l auto data.txt
```

Uses Generalized Cross Validation (GCV) to find optimal λ.

2. Manual Selection:

Data Characteristics	Recommended λ	Reasoning
Low noise, important details	0.001 - 0.01	Preserve features
Moderate noise	0.01 - 0.1	Balanced (default: 0.1)
High noise	0.1 - 1.0	Strong smoothing
Very noisy, global trends	1.0 - 10.0	Maximum smoothing

3. Iterative Refinement:

```
bash
```

```
# Start with automatic
./smooth -m 2 -l auto data.txt

# If result is over-smoothed (details lost):
./smooth -m 2 -l 0.01 data.txt

# If result is under-smoothed (still noisy):
./smooth -m 2 -l 1.0 data.txt
```

4. Diagnostic Criteria:

The program outputs functional components:

```
# Functional J = 1.234e+02 (Data: 5.67e+01 + Regularization: 6.67e+01)
# Data/Total ratio = 0.460, Regularization/Total ratio = 0.540
```

Good balance indicators:

- Data term: 30-70% of total functional
- Regularization term: 30-70% of total functional

Warning signs:

- Data term > 95%: Under-smoothed (λ too small)
- Regularization term > 95%: Over-smoothed (λ too large)

5. Grid-Dependent Considerations:

For non-uniform grids with ratio h_max/h_min > 5:

```
bash

# Start with more conservative (larger) λ
./smooth -m 2 -l 0.5 nonuniform_data.txt

# GCV may be less accurate - check results visually
./smooth -m 2 -l auto nonuniform_data.txt
```

λ and Grid Spacing

The effective regularization strength depends on grid spacing:

```
Effective strength ∝ λ / h²_avg

Same λ on finer grid   → weaker smoothing
Same λ on coarser grid → stronger smoothing
```

For dimensional consistency, λ has units [Length²].

Second Derivative Discretization (Hybrid Method v5.4)

Version 5.4 Implementation: Automatic selection between two discretization schemes based on grid uniformity.

Grid Uniformity Detection

```
ratio = h_max / h_min

ratio < 2.5:  Nearly uniform     → Average Coefficient Method
ratio ≥ 2.5:  Highly non-uniform → Local Spacing Method
```

Method 1: Average Coefficient (for ratio < 2.5)

Used for uniform and mildly non-uniform grids. More robust numerically.

Discretization: For interior point (i) with neighbors at spacing h_left and h_right, use **harmonic mean**:

```
h_harm = 2·h_left·h_right / (h_left + h_right)

D²u_i ≈ (u_{i-1} - 2u_i + u_{i+1}) / h_harm²
```

Why harmonic mean?

- More accurate than arithmetic mean for averaging intervals
- Gives greater weight to smaller spacing (physically correct)
- For h_left = h_right, reduces to standard formula

Matrix construction:

```
c = λ · Σ(1/h_i²) / (n-1)    (average coefficient)

A[i,i]   = 1 + 2c  (interior points)
A[i,i±1] = -c      (off-diagonals)
```

Method 2: Local Spacing (for ratio ≥ 2.5)

Used for highly non-uniform grids. More accurate for variable spacing.

Discretization: For point (i) with left spacing (h₁ = x[i] - x[i-1]) and right spacing (h₂ = x[i+1] - x[i]):

```
D²u_i ≈ (2/(h₁+h₂)) · [u_{i-1}/h₁ - u_i·(1/h₁+1/h₂) + u_{i+1}/h₂]
```

This is the **correct second derivative formula** for non-uniform grids derived from Taylor expansion.

Matrix construction:

$$w = 2\lambda / (h_1 + h_2)$$

$$A[i,i] = 1 + w\cdot(1/h_1 + 1/h_2)$$
$$A[i,i-1] = -w/h_1$$
$$A[i,i+1] = -w/h_2$$

The resulting matrix is:

- Symmetric
- Positive definite
- Tridiagonal (bandwidth = 1)

Boundary Conditions

Natural boundary conditions (second derivative = 0 at ends):

Left boundary (i=0):

$$D^2u_0 \approx (u_1 - u_0) / h_0^2$$

$$A[0,0] \mathrel{+}= \lambda/h_0^2$$
$$A[0,1] \mathrel{+}= -\lambda/h_0^2$$

Right boundary (i=n-1):

$$D^2u_{n-1} \approx (u_{n-1} - u_{n-2}) / h_{n-1}^2$$

$$A[n-1,n-1] \mathrel{+}= \lambda/h_{n-1}^2$$

Critical fix in v5.4: The boundary superdiagonal element A[0,1] was missing in previous versions, causing isolation of the first point. This is now corrected.

Functional Computation

The actual value of the minimized functional is computed for diagnostic purposes:

Data term:

$$\|y - u\|^2 = \Sigma(y_i - u_i)^2$$

Regularization term (must match matrix formulation!):

For **average coefficient method:**

$$\|D^2u\|^2 = \Sigma_{interior} [(u_{i-1} - 2u_i + u_{i+1})/h_{harm}^2]^2 + 0.5\cdot[(u_1 - u_0)/h_0^2]^2 + 0.5\cdot[(u_{n-1} - u_{n-2})/h_{n-1}^2]^2$$

For **local spacing method:**

$$\|D^2u\|^2 = \Sigma_{interior} [D^2u_i]^2 \cdot (h_1+h_2)/2 + 0.5\cdot[(u_1 - u_0)/h_0^2]^2 \cdot h_0 + 0.5\cdot[(u_{n-1} - u_{n-2})/h_{n-1}^2]^2 \cdot h_{n-1}$$

Note the **weighting factors** in local spacing method for proper integration over non-uniform grid.

Total functional:

$$J[u] = \|y - u\|^2 + \lambda\|D^2u\|^2$$

Variational Approach

The minimum of functional J[u] satisfies the Euler-Lagrange equation:

$$\partial J/\partial u_i = 0 \implies -2(y_i - u_i) + 2\lambda(D^T D\, u)_i = 0$$

which leads to the linear system:

$$(I + \lambda D^T D)u = y$$

Matrix Representation

Matrix $A = I + \lambda D^T D$ is:

- Symmetric
- Positive definite
- Tridiagonal (banded with bandwidth 1)

This structure allows efficient solution using LAPACK's banded solver `dpbsv`.

Generalized Cross Validation (GCV)

For automatic λ selection (`-l auto`), we minimize the GCV criterion:

$$GCV(\lambda) = n\cdot RSS(\lambda) / (n - tr(H_\lambda))^2$$

where:

- $RSS(\lambda) = \|y - u_\lambda\|^2$ is the residual sum of squares

- $H_\lambda = (I + \lambda D^T D)^{-1}$ is the influence matrix (smoother matrix)
- $tr(H_\lambda)$ is the trace (effective number of parameters)

Interpretation:

- $tr(H_\lambda)$ measures model complexity (degrees of freedom)
- Small λ: tr(H) ≈ n (interpolation, overfitting)
- Large λ: tr(H) ≈ 2 (straight line, underfitting)
- Optimal λ: minimizes prediction error

Trace estimation using eigenvalues:

For uniform grids with natural boundary conditions:

$$tr(H_\lambda) \approx \Sigma_{k=1}^{n} 1/(1 + \lambda\mu_k)$$

where eigenvalues:
$\theta_k = \pi k/n$
$\mu_k = 4 \cdot \sin^2(\theta_k/2) / h^2$

Note: This approximation is exact for uniform grids but approximate for non-uniform grids. For highly non-uniform grids (ratio > 5), the program issues a warning.

Enhanced GCV in v5.4

Over-fitting penalty:

```
If tr(H)/n > 0.7:
    GCV_modified = GCV · exp(10·(tr(H)/n - 0.7))
```

This exponential penalty prevents selection of too-small λ that would lead to overfitting.

L-curve backup (for n > 20000):

For very large datasets, GCV trace approximation may be inaccurate. The program also computes the L-curve (plot of $||D^2u||^2$ vs $||y-u||^2$) and finds the corner point with maximum curvature:

$$\kappa = |x'y'' - y'x''| / (x'^2 + y'^2)^{(3/2)}$$

where:
$x = \log(||y - u||^2)$
$y = \log(||D^2u||^2)$

If GCV and L-curve disagree significantly, the program uses the more conservative (larger) λ.

Efficient Implementation

The program uses LAPACK routine `dpbsv` for solving symmetric positive definite banded systems:

```
c

// Banded matrix storage (LAPACK column-major format)
AB[0,j] = superdiagonal elements
AB[1,j] = diagonal elements

// System solution
dpbsv_(&uplo, &n, &kd, &nrhs, AB, &ldab, b, &n, &info);
```

Complexity:

- Memory: O(n) for banded storage
- Time: O(n) for factorization and back-substitution

This is **optimal** for tridiagonal systems.

Hybrid Implementation (v5.4)

```
c

typedef struct {
    double *y_smooth;          // Smoothed values
    double *y_deriv;           // First derivatives
    double lambda;             // Used parameter
    int n;                     // Number of points
    double data_term;          // ||y - u||²
    double regularization_term; // λ||D²u||²
    double total_functional;    // J[u]
} TikhonovResult;

// Main function
TikhonovResult* tikhonov_smooth(double *x, double *y, int n, double lambda);

// Automatic λ selection
double find_optimal_lambda_gcv(double *x, double *y, int n);

// Memory cleanup
void free_tikhonov_result(TikhonovResult *result);
```

Characteristics

Advantages:

- Global optimization with theoretical foundation

- Flexible balance between data fidelity and smoothness (controlled by λ)
- Robust to outliers (quadratic penalty less sensitive than least squares)
- Efficient for large datasets (O(n) memory and time)
- **Automatic λ selection via GCV** - no guessing needed
- **Excellent for non-uniform grids** - correct discretization automatic
- **Unified approach** - same algorithm for uniform and non-uniform grids
- Works well for noisy data with global trends

Disadvantages:

- Single global parameter λ (cannot vary locally)
- May suppress local details if λ too large
- GCV may fail for some data types (especially highly non-uniform grids)
- Requires LAPACK library
- Boundary effects if data has discontinuities at edges

Method Comparison

Computational Complexity

Method	Time	Memory	Scalability
POLYFIT	$O(n \cdot p^3)$	$O(p^2)$	Good for small p
SAVGOL	$O(p^3) + O(n \cdot w)$	O(w)	Excellent for large n
TIKHONOV	O(n)	O(n)	Excellent

Note: w = window size, p = polynomial degree (≤12), n = number of data points.

Smoothing Quality

Property	POLYFIT	SAVGOL	TIKHONOV
Local adaptability	*****	****	**
Extreme preservation	****	*****	***
Noise robustness	***	****	*****
Derivative quality	*****	*****	***
Boundary behavior	**	***	****
Non-uniform grids	***	✗	*****
Ease of use (v5.4)	****	****	*****

Property	POLYFIT	SAVGOL	TIKHONOV
Parameter selection	Manual	Manual	Auto (GCV)

Key: ✗ = Not suitable (automatically rejected)

Grid Type Compatibility

Grid Type	POLYFIT	SAVGOL	TIKHONOV
Perfectly uniform (CV < 0.01)	✓	✓	✓
Nearly uniform (CV < 0.05)	✓	⚠	✓
Moderately non-uniform (0.05 < CV < 0.2)	✓	✗	✓
Highly non-uniform (CV > 0.2)	⚠	✗	✓
Very large ratio (h_max/h_min > 10)	⚠	✗	✓*

Legend:

- ✓ = Recommended
- ⚠ = Usable with caution
- ✗ = Rejected or not recommended
- * = Uses local spacing method automatically

Practical Recommendations

Method Selection by Data Type

POLYFIT - when:

- Data has variable curvature
- You need to preserve local details
- You have moderately noisy data
- You want highest quality derivatives
- Grid has moderate spacing variations
- You need local adaptability

SAVGOL - when:

- **Grid is uniform (CV < 0.05) -** automatically checked!
- You want mathematically optimal linear smoothing for polynomial signals
- Data contains periodic or oscillatory components that need preservation

- You need excellent peak shape preservation (areas, moments)
- You want minimal phase distortion in the smoothed signal
- You're processing time series or spectroscopic data on uniform grids
- You need simultaneous high-quality function and derivative estimation
- Computational efficiency is critical (large datasets)

TIKHONOV - when:

- **Grid is non-uniform** - works perfectly automatically!
- Data is very noisy
- You need global consistency
- **You want automatic parameter selection (λ auto)** - highly recommended!
- You prefer global optimization approaches over local fitting
- You want robust handling of outliers
- You want the simplest workflow (one parameter, automatic selection)
- You need to process very large datasets efficiently
- **You're not sure which method to use** - Tikhonov with `-l auto` is safest!

Parameter Selection

Window size (n) for POLYFIT/SAVGOL:

```
n = 2*k + 1    (odd number)

Recommendations:
- Low noise: n = 5-9
- Medium noise: n = 9-15
- High noise: n = 15-25

Rule of thumb: n ≈ 2p + 3
```

Polynomial degree (p):

- Linear trends: p = 1-2
- Smooth curves: p = 2-3
- Complex signals: p = 3-4
- Advanced applications: p = 5-8
- Maximum: p ≤ 12
- Recommended maximum: p < n/2

Note: Degrees > 6 may cause numerical instability warnings.

Lambda (λ) for TIKHONOV:

RECOMMENDED:
- Auto selection: -l auto (uses GCV optimization)

MANUAL SELECTION:
Starting points by noise level:
- Low noise: λ = 0.001 - 0.01
- Medium noise: λ = 0.01 - 0.1 (default: 0.1)
- High noise: λ = 0.1 - 1.0
- Very noisy data: λ = 1.0 - 10.0

Full range: 10^{-6} to 10^{3}

ITERATIVE REFINEMENT:
1. Start with -l auto
2. Check functional balance (should be 30-70% each)
3. If over-smoothed: decrease λ by factor of 10
4. If under-smoothed: increase λ by factor of 10
5. Repeat until satisfied

GRID-DEPENDENT:
For non-uniform grids (ratio > 5):
- Start more conservative (larger λ)
- GCV may be less accurate - check visually

Usage Examples

Basic Syntax

bash

```
# Polynomial fitting (smoothed values only)
./smooth -m 0 -n 7 -p 2 data.txt

# Polynomial fitting with derivatives
./smooth -m 0 -n 7 -p 2 -d data.txt

# Savitzky-Golay (smoothed values only)
# NOTE: Will be rejected if grid is non-uniform!
./smooth -m 1 -n 9 -p 3 data.txt

# Savitzky-Golay with derivatives
./smooth -m 1 -n 9 -p 3 -d data.txt

# Tikhonov with automatic λ (RECOMMENDED)
./smooth -m 2 -l auto data.txt

# Tikhonov with automatic λ and derivatives
./smooth -m 2 -l auto -d data.txt

# Tikhonov with manual λ
./smooth -m 2 -l 0.01 data.txt

# Tikhonov with manual λ and derivatives
./smooth -m 2 -l 0.01 -d data.txt

# Grid analysis (works with any method)
./smooth -m 2 -l auto -g data.txt
```

Output Format

Without `-d` flag:

```
# Data smooth - Tikhonov regularization with lambda = 1e-01
# Functional J = 1.234e+02 (Data: 5.67e+01 + Regularization: 6.67e+01)
# Data/Total ratio = 0.460, Regularization/Total ratio = 0.540
#    x          y
  0.00000E+00  1.00000E+00
  1.00000E+00  2.71828E+00
  ...
```

With `-d` flag:

```
# Data smooth - Tikhonov regularization with lambda = 1e-01
# Functional J = 1.234e+02 (Data: 5.67e+01 + Regularization: 6.67e+01)
# Data/Total ratio = 0.460, Regularization/Total ratio = 0.540
#    x          y          y'
  0.00000E+00  1.00000E+00  1.00000E+00
  1.00000E+00  2.71828E+00  2.71828E+00
  ...
```

With `-g` flag (grid analysis):

```
# ========================================
# GRID UNIFORMITY ANALYSIS
# ========================================
# Grid uniformity analysis:
#   n = 1000 points
#   h_min = 9.500000e-03, h_max = 1.200000e-02, h_avg = 1.000000e-02
#   h_max/h_min = 1.26, CV = 0.052
#   Grid type: NON-UNIFORM
#   Uniformity score: 0.94
#   Standard deviation: 5.200000e-04
#   Detected clusters: 0
#   Recommendation: Grid is nearly uniform - standard methods work well
# ========================================
#
# Using Average Coefficient Method (h_max/h_min = 1.26)
# GCV optimization for n=1000 points (h_max/h_min = 1.26)
...
```

Working with Non-uniform Grids (v5.4)

```
bash
```

```
# First, analyze your grid
./smooth -m 2 -l auto -g nonuniform_data.txt

# If ratio < 2.5, program uses average coefficient method
# Output: "Using Average Coefficient Method (h_max/h_min = 1.85)"

# If ratio ≥ 2.5, program uses local spacing method
# Output: "Using Local Spacing Method (h_max/h_min = 8.42)"

# Automatic selection - no user intervention needed!
# Just use the same command:
./smooth -m 2 -l auto nonuniform_data.txt

# For very non-uniform grids (ratio > 5), you may see:
# "WARNING: Highly non-uniform grid detected!"
# "GCV trace approximation may be less accurate."
# In this case, try manual λ or check results visually.
```

Typical Workflow

1. **Quick data exploration with grid analysis:**

```
bash

  # Check grid uniformity
  ./smooth -m 2 -l auto -g data.txt > smooth_data.txt

  # Review output comments for:
  # - Grid uniformity (CV, ratio)
  # - Which discretization method was used
  # - Functional balance (data vs regularization)
```

2. **Choose method based on grid:**

```
bash

  # For uniform grids (CV < 0.05):
  ./smooth -m 1 -n 9 -p 3 -d data.txt

  # For non-uniform grids:
  ./smooth -m 2 -l auto -d data.txt
```

3. **Refine λ if needed:**

```
bash
```

```
# If automatic λ gives over-smoothing:
./smooth -m 2 -l 0.01 -d data.txt

# If under-smoothing:
./smooth -m 2 -l 1.0 -d data.txt
```

4. **For publication graphics:**

```
bash

# Final smoothing with derivatives
./smooth -m 2 -l auto -d data.txt > publication_data.txt

# Check functional balance in output comments
# Ideal: both terms contribute 30-70%
```

Grid Analysis Module

The `grid_analysis` module provides comprehensive analysis of input data and helps optimize smoothing parameters.

Main Functions

```
c

// Complete grid analysis
GridAnalysis* analyze_grid(double *x, int n, int store_spacings);

// Quick uniformity check
int is_uniform_grid(double *x, int n, double *h_avg, double tolerance);

// Method recommendation
const char* get_grid_recommendation(GridAnalysis *analysis);

// Optimal window size
int optimal_window_size(GridAnalysis *analysis, int min_window, int max_window);
```

GridAnalysis Structure

```
c
```

```c
typedef struct {
    double h_min;          // Minimum spacing
    double h_max;          // Maximum spacing
    double h_avg;          // Average spacing
    double h_std;          // Standard deviation
    double ratio_max_min;  // h_max/h_min ratio
    double cv;             // Coefficient of variation
    double uniformity_score;// Uniformity score (0-1)
    int is_uniform;        // 1 = uniform, 0 = non-uniform
    int n_clusters;        // Number of detected clusters
    int reliability_warning;// Reliability warning
    char warning_msg[512];  // Warning text
    double *spacings;      // Array of spacings (optional)
    int n_points;          // Number of points
    int n_intervals;       // Number of intervals (n-1)
} GridAnalysis;
```

Example Analysis Output

```
# ========================================
# GRID UNIFORMITY ANALYSIS
# ========================================
# Grid uniformity analysis:
#   n = 1000 points
#   h_min = 1.000000e-02, h_max = 1.000000e-01, h_avg = 5.500000e-02
#   h_max/h_min = 10.00, CV = 0.450
#   Grid type: NON-UNIFORM
#   Uniformity score: 0.35
#   Standard deviation: 2.475000e-02
#   Detected clusters: 2
#   Recommendation: High non-uniformity - adaptive methods recommended
# WARNING: HIGH grid non-uniformity: h_max/h_min = 10.0
# Adaptive methods are strongly recommended.
# Consider using smaller regularization parameters.
#
# WARNING: 2 abrupt spacing changes detected (possible data clustering).
# Standard methods may over-smooth clustered regions.
# ========================================
```

Grid Uniformity Thresholds

```
CV < 0.01:   Perfectly uniform - all methods work optimally
             → POLYFIT, SAVGOL, TIKHONOV all excellent

CV < 0.05:   Nearly uniform - SAVGOL works with warning
```

→ SAVGOL may show warning but works
→ POLYFIT and TIKHONOV work fine

0.05 ≤ CV < 0.20: Moderately non-uniform
→ SAVGOL rejected automatically
→ POLYFIT usable
→ TIKHONOV recommended (uses average coef if ratio < 2.5)

CV ≥ 0.20: Highly non-uniform
→ SAVGOL rejected
→ POLYFIT with caution
→ TIKHONOV strongly recommended (may use local spacing)

RATIO:
ratio < 2.5: Tikhonov uses average coefficient method
ratio ≥ 2.5: Tikhonov uses local spacing method
ratio > 10: Warning issued, GCV may be less accurate

Compilation and Installation

Requirements

- C compiler (gcc, clang)
- LAPACK and BLAS libraries
- Make (optional)

Compilation using Make

```bash
# Standard compilation
make

# Debug build
make debug

# Clean
make clean

# Install to user's home directory
make install-user

# Install to system (requires root)
make install
```

Manual Compilation

```bash
# Standard compilation
gcc -o smooth smooth.c polyfit.c savgol.c tikhonov.c \
    grid_analysis.c decomment.c -llapack -lblas -lm -O2

# With warnings
gcc -Wall -Wextra -pedantic -o smooth smooth.c polyfit.c savgol.c \
    tikhonov.c grid_analysis.c decomment.c -llapack -lblas -lm -O2
```

File Structure

```
smooth/
├── smooth.c          # Main program (v5.2: added -g flag)
├── polyfit.c/h       # Polynomial fitting module
├── savgol.c/h        # Savitzky-Golay module (v5.3: with uniformity check)
├── tikhonov.c/h      # Tikhonov module (v5.4: hybrid implementation)
├── grid_analysis.c/h # Grid analysis
├── decomment.c/h     # Comment removal
├── revision.h        # Program version
├── Makefile          # Build system
└── README.md         # This documentation
```

Conclusion

The `smooth` program v5.4 provides three complementary smoothing methods in a modular architecture with advanced input data analysis:

- **POLYFIT** - local polynomial approximation using least squares method
- **SAVGOL** - optimal linear filter with pre-computed coefficients (uniform grids only)
- **TIKHONOV** - global variational method with hybrid automatic discretization
- **GRID_ANALYSIS** - automatic analysis and method recommendation

Version 5.4 Highlights

Major Improvements:

1. **Tikhonov hybrid implementation** - automatic selection between average coefficient (ratio < 2.5) and local spacing (ratio ≥ 2.5) methods
2. **Harmonic mean** - more accurate interval averaging for nearly-uniform grids
3. **Fixed boundary conditions** - corrected missing superdiagonal element

4. **Enhanced GCV** - over-fitting penalty and L-curve backup for large datasets
5. **Better diagnostics** - functional balance reporting, λ selection guidance
6. **Grid analysis with** `-g` - detailed uniformity statistics for parameter optimization

When to Use Each Method

Quick Decision Tree:

```
Is your grid uniform (CV < 0.05)?
├─ YES: Use SAVGOL for best computational efficiency
│       smooth -m 1 -n 9 -p 3 -d data.txt
│
└─ NO:  Use TIKHONOV for correct handling
       smooth -m 2 -l auto -d data.txt

Need local adaptability?
└─ Use POLYFIT regardless of grid
   smooth -m 0 -n 7 -p 2 -d data.txt

Not sure?
└─ Use TIKHONOV with automatic λ - safest choice!
   smooth -m 2 -l auto -g -d data.txt
```

Each method has a strong mathematical foundation and is optimized for specific data types. The program provides automatic guidance on method selection and parameters, with extensive diagnostics to ensure correct usage.

Best Practices

1. **Always check grid first:** Use `-g` flag to understand your data
2. **Start with automatic:** Use `-l auto` for Tikhonov, let GCV find optimal λ
3. **Check functional balance:** Look for 30-70% split between data and regularization terms
4. **Iterate if needed:** Adjust λ manually if automatic selection doesn't satisfy requirements
5. **Use derivatives wisely:** Add `-d` only when needed - cleaner output without it
6. **Understand the trade-off:** More smoothing (larger λ) = more noise reduction but less detail

---

Document revision: 2025-10-13
Program version: smooth v5.4
Dependencies: LAPACK, BLAS