Local Polynomial Approximation: Theory and Applications

Signal Processing and Image Analysis

July 16, 2025

Abstract

This document presents a comprehensive treatment of Local Polynomial Approximation (LPA) methods, transitioning from traditional sparsity-based approaches to more flexible polynomial modeling paradigms. We examine the theoretical foundations, mathematical formulations, and practical implementations of LPA techniques, with particular emphasis on weighted variants and their applications in signal processing and image analysis. The development includes rigorous mathematical derivations, computational algorithms, and connections to convolution-based implementations.

Contents

1 Introduction to Local Polynomial Approximation

Local Polynomial Approximation (LPA) represents a fundamental shift in signal processing methodology, moving away from global sparsity constraints toward localized polynomial modeling. This approach recognizes that many real-world signals exhibit piecewise smooth behavior that can be effectively captured through local polynomial representations.

1.1 Motivation and Historical Context

Traditional signal processing approaches often rely on global assumptions about signal structure, such as sparsity in transformed domains (e.g., DCT, wavelet transforms). While these methods have proven successful in many applications, they impose uniform constraints across the entire signal domain. In contrast, LPA methods adapt to local signal characteristics, providing more flexible and often more accurate approximations.

Definition 1.1 (Local Polynomial Approximation)**.** Given a signal $f : \mathbb{R} \to \mathbb{R}$ and a point $x_0 \in \mathbb{R}$, a local polynomial approximation of degree L is a polynomial $P_L(x)$ of the form:

$$P_L(x) = \sum_{j=0}^{L} \beta_j (x - x_0)^j \tag{1}$$

that minimizes a local fitting criterion within a neighborhood of x_0.

1.2 Fundamental Paradigm Shift

The transition from sparsity-based to polynomial-based modeling represents a significant conceptual advancement:

- **Sparsity-based approach**: Assumes signal can be represented as a linear combination of few basis functions from a fixed dictionary
- **Polynomial-based approach**: Assumes signal can be locally approximated by polynomials of appropriate degree

This shift enables adaptive processing where each spatial location can have its own optimal approximation parameters, leading to superior performance in regions with varying signal characteristics.

2 Mathematical Formulation of Local Polynomial Approximation

2.1 Signal Model and Notation

Consider a one-dimensional signal model:

$$y(t) = f(t) + \eta(t) \tag{2}$$

where $f(t)$ represents the underlying smooth signal and $\eta(t)$ denotes additive white Gaussian noise with variance σ^2.

For discrete processing, we work with sampled versions. Let $\mathbf{y} = [y_1, y_2, \ldots, y_M]^T$ represent an M-dimensional signal vector extracted from a local neighborhood around a central pixel.

2.2 Local Polynomial Model

Within a local neighborhood, we assume the signal can be well approximated by a polynomial of degree L:

$$f(t_i) \approx \sum_{j=0}^{L} \beta_j t_i^j, \quad i = 1, 2, \ldots, M \tag{3}$$

where $\{\beta_j\}_{j=0}^{L}$ are the polynomial coefficients to be estimated, and $\{t_i\}_{i=1}^{M}$ are the spatial locations within the neighborhood.

Remark 2.1. The polynomial degree L must satisfy $L+1 \leq M$ to ensure an over-determined system. This constraint prevents overfitting and ensures stable coefficient estimation.

2.3 Matrix Formulation

The polynomial approximation can be expressed in matrix form. Define the design matrix $\mathbf{T} \in \mathbb{R}^{M \times (L+1)}$:

$$\mathbf{T} = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^L \\ 1 & t_2 & t_2^2 & \cdots & t_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_M & t_M^2 & \cdots & t_M^L \end{bmatrix} \tag{4}$$

The polynomial coefficient vector is $\beta = [\beta_0, \beta_1, \ldots, \beta_L]^T \in \mathbb{R}^{L+1}$.

The polynomial approximation becomes:

$$\mathbf{f} \approx \mathbf{T}\beta \tag{5}$$

2.4 Least Squares Formulation

The optimal polynomial coefficients are obtained by minimizing the squared approximation error:

$$\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^{L+1}} \|\mathbf{y} - \mathbf{T}\beta\|^2 \tag{6}$$

Theorem 2.2 (Unweighted LPA Solution)**.** The least squares solution to the local polynomial approximation problem is:

$$\hat{\beta} = (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{y} \tag{7}$$

provided that $\mathbf{T}^T\mathbf{T}$ is invertible.

Proof. Taking the derivative of the objective function (6) with respect to β and setting it to zero:

$$\frac{\partial}{\partial\beta}\|\mathbf{y} - \mathbf{T}\beta\|^2 = \frac{\partial}{\partial\beta}(\mathbf{y} - \mathbf{T}\beta)^T(\mathbf{y} - \mathbf{T}\beta) \tag{8}$$

$$= -2\mathbf{T}^T(\mathbf{y} - \mathbf{T}\beta) = 0 \tag{9}$$

Solving for β yields the normal equations:

$$\mathbf{T}^T\mathbf{T}\beta = \mathbf{T}^T\mathbf{y} \tag{10}$$

The solution follows directly when $\mathbf{T}^T\mathbf{T}$ is invertible. □

3 Weighted Local Polynomial Approximation

3.1 Motivation for Weighting

In many practical scenarios, not all samples within a neighborhood should contribute equally to the polynomial fit. Samples closer to the center of the neighborhood are typically more relevant for estimating the signal at that location. Weighted LPA addresses this by introducing spatially varying weights.

3.2 Weighted Formulation

The weighted LPA problem incorporates a diagonal weight matrix $\mathbf{W} \in \mathbb{R}^{M \times M}$:

$$\hat{\beta}_w = \arg\min_{\beta \in \mathbb{R}^{L+1}} \|\mathbf{W}(\mathbf{y} - \mathbf{T}\beta)\|^2 \tag{11}$$

where $\mathbf{W} = \text{diag}(w_1, w_2, \ldots, w_M)$ with $w_i \geq 0$ for all i.

Theorem 3.1 (Weighted LPA Solution)**.** The weighted least squares solution is:

$$\hat{\beta}_w = (\mathbf{T}^T\mathbf{W}^2\mathbf{T})^{-1}\mathbf{T}^T\mathbf{W}^2\mathbf{y} \tag{12}$$

Proof. The weighted objective function can be written as:

$$\|\mathbf{W}(\mathbf{y} - \mathbf{T}\beta)\|^2 = \|\mathbf{W}\mathbf{y} - \mathbf{W}\mathbf{T}\beta\|^2 \tag{13}$$

This is equivalent to solving the unweighted problem:

$$\arg\min_\beta \left\|\tilde{\mathbf{y}} - \tilde{\mathbf{T}}\beta\right\|^2 \tag{14}$$

where $\tilde{\mathbf{y}} = \mathbf{W}\mathbf{y}$ and $\tilde{\mathbf{T}} = \mathbf{W}\mathbf{T}$.

Applying Theorem 2.2:

$$\begin{aligned}
\hat{\beta}_w &= (\tilde{\mathbf{T}}^T\tilde{\mathbf{T}})^{-1}\tilde{\mathbf{T}}^T\tilde{\mathbf{y}} && (15)\\
&= ((\mathbf{W}\mathbf{T})^T(\mathbf{W}\mathbf{T}))^{-1}(\mathbf{W}\mathbf{T})^T\mathbf{W}\mathbf{y} && (16)\\
&= (\mathbf{T}^T\mathbf{W}^2\mathbf{T})^{-1}\mathbf{T}^T\mathbf{W}^2\mathbf{y} && (17)
\end{aligned}$$

□

3.3 Weight Selection Strategies

Common weight functions include:

1. **Uniform weights**: $w_i = 1$ for all i (reduces to unweighted case)
2. **Gaussian weights**: $w_i = \exp(-\frac{(t_i - t_0)^2}{2\sigma_w^2})$
3. **Binary weights**: $w_i \in \{0, 1\}$ for adaptive support selection

Example 3.2 (Binary Weight Example)**.** Consider a signal with a discontinuity. Using binary weights allows selective processing:

- Left-side filter: $\mathbf{w} = [1, 1, 1, 0, 0]^T$
- Right-side filter: $\mathbf{w} = [0, 0, 1, 1, 1]^T$

This prevents blurring across discontinuities while maintaining smoothing within homogeneous regions.

4 QR Decomposition Approach

4.1 Motivation for QR Decomposition

Direct computation of $(\mathbf{T}^T\mathbf{T})^{-1}$ can be numerically unstable, especially when $\mathbf{T}$ is ill-conditioned. The QR decomposition provides a numerically stable alternative while revealing the underlying geometric structure of the problem.

4.2 QR Decomposition of Design Matrix

Theorem 4.1 (QR Decomposition)**.** Any matrix $\mathbf{T} \in \mathbb{R}^{M\times(L+1)}$ with $M \geq L+1$ and full column rank can be decomposed as:

$$\mathbf{T} = \mathbf{QR} \tag{18}$$

where $\mathbf{Q} \in \mathbb{R}^{M\times(L+1)}$ has orthonormal columns ($\mathbf{Q}^T\mathbf{Q} = \mathbf{I}_{L+1}$) and $\mathbf{R} \in \mathbb{R}^{(L+1)\times(L+1)}$ is upper triangular.

4.3 LPA Solution via QR Decomposition

Using the QR decomposition, the LPA solution becomes:

$$\begin{aligned}
\hat{\beta} &= (\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{y} && (19)\\
&= ((\mathbf{QR})^T(\mathbf{QR}))^{-1}(\mathbf{QR})^T\mathbf{y} && (20)\\
&= (\mathbf{R}^T\mathbf{Q}^T\mathbf{QR})^{-1}\mathbf{R}^T\mathbf{Q}^T\mathbf{y} && (21)\\
&= (\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{Q}^T\mathbf{y} && (22)\\
&= \mathbf{R}^{-1}\mathbf{Q}^T\mathbf{y} && (23)
\end{aligned}$$

The signal estimate is:

$$\hat{\mathbf{f}} = \mathbf{T}\hat{\beta} = \mathbf{QRR}^{-1}\mathbf{Q}^T\mathbf{y} = \mathbf{QQ}^T\mathbf{y} \tag{24}$$

Proposition 4.2 (Projection Interpretation)**.** The matrix $\mathbf{P} = \mathbf{QQ}^T$ represents the orthogonal projection onto the column space of $\mathbf{T}$. The LPA estimate is the orthogonal projection of the noisy signal onto the space of polynomials of degree L.

4.4 Weighted QR Decomposition

For weighted LPA, we apply QR decomposition to the weighted design matrix $\mathbf{WT}$:

$$\mathbf{WT} = \tilde{\mathbf{Q}}\tilde{\mathbf{R}} \tag{25}$$

The weighted solution becomes:

$$\begin{aligned}
\hat{\beta}_w &= \tilde{\mathbf{R}}^{-1}\tilde{\mathbf{Q}}^T\mathbf{Wy} && (26)\\
\hat{\mathbf{f}}_w &= \tilde{\mathbf{Q}}\tilde{\mathbf{Q}}^T\mathbf{Wy} && (27)
\end{aligned}$$

5 Convolution Implementation

5.1 From Matrix Operations to Convolution

A key insight in LPA is that the estimation can be implemented as a convolution operation, enabling efficient computation across entire signals or images.

5.2 Derivation of Convolution Kernel

Consider the estimation of the signal value at the center of the neighborhood. Let i_c denote the central index. The estimate at this location is:

$$\hat{f}(t_{i_c}) = \mathbf{e}_{i_c}^T \mathbf{Q}\mathbf{Q}^T \mathbf{y} \tag{28}$$

where $\mathbf{e}_{i_c}$ is the unit vector with 1 in the i_c-th position.

Theorem 5.1 (Convolution Kernel Formula)**.** The LPA estimation at the central location can be expressed as:

$$\hat{f}(t_{i_c}) = \sum_{i=1}^{M} h_i y_i = \mathbf{h}^T \mathbf{y} \tag{29}$$

where the convolution kernel is:

$$\mathbf{h} = \mathbf{Q}\mathbf{Q}^T \mathbf{e}_{i_c} \tag{30}$$

5.3 Explicit Kernel Computation

The convolution kernel can be computed explicitly as:

$$\mathbf{h} = \sum_{j=0}^{L} \beta_j \mathbf{q}_j \tag{31}$$

where $\mathbf{q}_j$ are the columns of $\mathbf{Q}$ and:

$$\beta_j = \mathbf{q}_j^T \mathbf{e}_{i_c} = q_{j,i_c} \tag{32}$$

5.4 Special Cases

Example 5.2 (Zero-Order Polynomial (Moving Average))**.** For $L = 0$, the design matrix is $\mathbf{T} = \mathbf{1} = [1, 1, \ldots, 1]^T$.

The QR decomposition gives:

$$\mathbf{Q} = \frac{1}{\sqrt{M}}\mathbf{1} \tag{33}$$

$$\mathbf{R} = \sqrt{M} \tag{34}$$

The convolution kernel becomes:

$$\mathbf{h} = \frac{1}{M}\mathbf{1} \tag{35}$$

This is the standard moving average filter.

Example 5.3 (Weighted Zero-Order Polynomial)**.** For weighted zero-order polynomial with normalized weights $\sum_{i=1}^{M} w_i^2 = 1$:

$$\tilde{\mathbf{Q}} = \mathbf{w} \tag{36}$$

$$\tilde{\mathbf{R}} = 1 \tag{37}$$

The convolution kernel is:

$$\mathbf{h} = \mathbf{w} \tag{38}$$

This shows that Gaussian smoothing corresponds to weighted zero-order polynomial fitting.

6 Statistical Properties and Performance Analysis

6.1 Bias-Variance Decomposition

The mean squared error (MSE) of the LPA estimator can be decomposed into bias and variance components:

$$\text{MSE}(\hat{f}(t_0)) = \text{Bias}^2(\hat{f}(t_0)) + \text{Var}(\hat{f}(t_0)) \tag{39}$$

6.2 Bias Analysis

Theorem 6.1 (Bias of LPA Estimator)**.** For a signal $f(t)$ that is $(L+1)$-times differentiable, the bias of the LPA estimator is:

$$\text{Bias}(\hat{f}(t_0)) = \frac{f^{(L+1)}(t_0)}{(L+1)!} \sum_{i=1}^{M} h_i (t_i - t_0)^{L+1} + O(h^{L+2}) \tag{40}$$

where h is the neighborhood size.

Corollary 6.2 (Bias for Polynomial Signals)**.** If the true signal is a polynomial of degree L or less, the LPA estimator is unbiased.

6.3 Variance Analysis

Theorem 6.3 (Variance of LPA Estimator)**.** For additive white Gaussian noise with variance σ^2, the variance of the LPA estimator is:

$$\text{Var}(\hat{f}(t_0)) = \sigma^2 \left\|\mathbf{h}\right\|^2 = \sigma^2 \mathbf{e}_{i_c}^T \mathbf{Q}\mathbf{Q}^T \mathbf{e}_{i_c} \tag{41}$$

Remark 6.4. The variance decreases as the effective number of samples increases, but the bias may increase due to the larger neighborhood size. This creates a fundamental bias-variance tradeoff.

6.4 Optimal Neighborhood Selection

The optimal neighborhood size balances bias and variance:

$$h_{\text{opt}} = \arg\min_h \left[\text{Bias}^2(h) + \text{Var}(h)\right] \tag{42}$$

In practice, this leads to adaptive algorithms that select different neighborhood sizes and shapes based on local signal characteristics.

7 Adaptive Neighborhood Selection

7.1 Motivation for Adaptivity

Fixed neighborhood sizes and shapes are suboptimal for signals with varying local characteristics. Adaptive methods adjust the approximation parameters based on local signal properties.

7.2 Directional Filtering

Binary weights enable directional filtering, which is particularly useful near discontinuities:

Definition 7.1 (Directional Kernels)**.** A set of directional kernels $\{\mathbf{h}_d\}_{d=1}^{D}$ provides estimates along different directions or orientations. Each kernel uses binary weights to select samples from a specific spatial direction.

Example 7.2 (One-Dimensional Directional Kernels)**.** For a 1D signal with neighborhood size $M = 5$:

$$\mathbf{h}_{\text{left}} : \text{weights } [1, 1, 1, 0, 0] \tag{43}$$

$$\mathbf{h}_{\text{right}} : \text{weights } [0, 0, 1, 1, 1] \tag{44}$$

$$\mathbf{h}_{\text{center}} : \text{weights } [0, 1, 1, 1, 0] \tag{45}$$

7.3 Intersection of Confidence Intervals

The Intersection of Confidence Intervals (ICI) rule provides a principled approach for adaptive neighborhood selection:

---

1. Compute estimates $\{\hat{f}_d\}$ and confidence intervals $\{CI_d\}$ for each directional kernel
2. Find the intersection of all confidence intervals: $CI_{\text{intersect}} = \bigcap_{d=1}^{D} CI_d$
3. Select the estimator with the largest neighborhood whose confidence interval contains $CI_{\text{intersect}}$

7.4 Computational Complexity

The convolution implementation provides significant computational advantages:

- **Direct matrix approach**: $O(M^2 L)$ operations per pixel
- **Convolution approach**: $O(ML)$ operations per pixel
- **FFT-based convolution**: $O(M \log M)$ operations per pixel for large M

8 Extensions and Applications

8.1 Two-Dimensional Extension

The LPA framework extends naturally to images by considering 2D polynomials:

$$P(x, y) = \sum_{i=0}^{L_x} \sum_{j=0}^{L_y} \beta_{i,j} x^i y^j \tag{46}$$

The design matrix becomes:

$$\mathbf{T}_{2D} = \begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \cdots \\ 1 & x_2 & y_2 & x_2^2 & x_2 y_2 & y_2^2 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{47}$$

8.2 Robust LPA

For signals contaminated with outliers, robust loss functions can be employed:

$$\hat{\beta}_{\text{robust}} = \arg\min_{\beta} \sum_{i=1}^{M} \rho(y_i - \mathbf{t}_i^T \beta) \tag{48}$$

where $\rho(\cdot)$ is a robust loss function (e.g., Huber loss, ℓ_1 loss).

8.3 Multi-Scale Processing

LPA can be integrated into multi-scale frameworks for enhanced performance:

1. Apply LPA at multiple scales
2. Combine estimates using scale-dependent weights
3. Propagate information across scales

9 Conclusion

Local Polynomial Approximation provides a powerful and flexible framework for signal processing that adapts to local signal characteristics. Key advantages include:

- **Adaptivity**: Neighborhood size and shape adapt to local signal properties
- **Computational efficiency**: Convolution-based implementation
- **Theoretical foundation**: Well-understood bias-variance properties
- **Extensibility**: Natural extensions to higher dimensions and robust variants

The method represents a significant advancement over traditional fixed-basis approaches, enabling more accurate and artifact-free signal processing in applications ranging from denoising to edge-preserving smoothing.

10 Appendix: Implementation Details

10.1 MATLAB Implementation

```
function h = lpa_kernel(support, degree, weights)
% Compute LPA convolution kernel
% support: spatial support locations
% degree: polynomial degree
% weights: optional weight vector

M = length(support);
L = degree;

% Design matrix
T = zeros(M, L+1);
for i = 1:M
    for j = 0:L
        T(i, j+1) = support(i)^j;
    end
end

% Apply weights if provided
if nargin > 2 && ~isempty(weights)
    W = diag(weights);
    T = W * T;
end

% QR decomposition
[Q, R] = qr(T, 0);

% Central index
ic = ceil(M/2);

% Compute kernel
h = Q * Q' * eye(M, 1) * ic;
end
```

10.2 Python Implementation

```
import numpy as np
from scipy.linalg import qr

def lpa_kernel(support, degree, weights=None):
    """
    Compute LPA convolution kernel
```

```
Parameters:
support: array of spatial support locations
degree: polynomial degree
weights: optional weight vector

Returns:
h: convolution kernel
"""

M = len(support)
L = degree

# Design matrix
T = np.zeros((M, L+1))
for i in range(M):
    for j in range(L+1):
        T[i, j] = support[i]**j

# Apply weights if provided
if weights is not None:
    W = np.diag(weights)
    T = W @ T

# QR decomposition
Q, R = qr(T, mode='economic')

# Central index
ic = M // 2

# Compute kernel
e_ic = np.zeros(M)
e_ic[ic] = 1
h = Q @ Q.T @ e_ic

return h
```