Mathematics of Relative-Kernel Smoothing on a Geometric Grid

Setup

Assume the supplied SciPy distribution represents a random variable

$$Z \sim \texttt{kernel}.$$

The distribution describes the relative displacement from a point q. With scale factor `alpha`,

$$q' = q(1 + \alpha Z).$$

Equivalently,

$$Z = \frac{q' - q}{\alpha q}.$$

Therefore the continuous smoothing operator is

$$\bar{y}(q) = \int_0^\infty y(q') K(q, q')\, dq',$$

with

$$\boxed{K(q, q') = \frac{1}{\alpha q} f\left(\frac{q' - q}{\alpha q}\right)}$$

where $f = \texttt{kernel.pdf}$. The implementation does not need to call `pdf`; this formula explains the continuous kernel.

Positive-domain normalization

The physical domain is $q' > 0$. Since

$$q' = q(1 + \alpha Z),$$

valid values satisfy

$$1 + \alpha Z > 0 \quad \Longleftrightarrow \quad Z > -\frac{1}{\alpha}.$$

Let $F = \texttt{kernel.cdf}$. The positive-domain mass is

$$\boxed{N = 1 - F\left(-\frac{1}{\alpha}\right)}$$

and the normalized positive-domain kernel is

$$K_+(q, q') = \frac{1}{N} \frac{1}{\alpha q} f\left(\frac{q' - q}{\alpha q}\right), \qquad q' > 0.$$

This corresponds to the implementation pattern

$$\texttt{z\_domain\_min} = -1/\alpha, \qquad \texttt{norm\_mass} = 1 - F(\texttt{z\_domain\_min}).$$

Log-coordinate transformation

Introduce the log-relative coordinate

$$u = \log(q'/q), \qquad q' = qe^u.$$

Then

$$dq' = qe^u\, du$$

and

$$z = \frac{q' - q}{\alpha q} = \frac{e^u - 1}{\alpha}.$$

Substituting into the normalized kernel gives the log-space kernel

$$\boxed{g(u) = \frac{1}{N}\frac{e^u}{\alpha} f\left(\frac{e^u - 1}{\alpha}\right)}$$

and the smoothed value becomes

$$\bar{y}(q) = \int g(u)\, y(qe^u)\, du.$$

This is shift-invariant on a geometric grid. If

$$q_i = q_0\rho^i, \qquad h = \log\rho,$$

then

$$q_i e^{mh} = q_{i+m}.$$

So the log-space integral can be approximated by a fixed stencil:

$$\boxed{\bar{y}_i \approx \sum_m w_m y_{i+m}.}$$

Discrete weights from the CDF

The log-grid cell for offset m is

$$L_m = \left(m - \frac{1}{2}\right) h, \qquad U_m = \left(m + \frac{1}{2}\right) h.$$

Mapping the cell bounds back to the distribution coordinate gives

$$z_L = \frac{e^{L_m} - 1}{\alpha}, \qquad z_U = \frac{e^{U_m} - 1}{\alpha}.$$

The exact cell-integrated weight is

$$\boxed{w_m = \frac{F(z_U) - F(z_L)}{N}}$$

where $F =$ `kernel.cdf`. This is why the implementation computes weights with CDF differences instead of sampling the PDF.

Equivalently, before finite-support clipping and tail truncation, the central formula is

$$\boxed{w_m = \frac{F\left(\frac{\exp((m+\frac{1}{2})h) - 1}{\alpha}\right) - F\left(\frac{\exp((m-\frac{1}{2})h) - 1}{\alpha}\right)}{1 - F(-1/\alpha)}}$$

with $h = \log\rho$.

Role of `support` and `ppf`

The method `kernel.support()` gives the range where the distribution can have nonzero density:

$$z \in [z_{\min}, z_{\max}].$$

After enforcing $q' > 0$, the valid range becomes

$$z \in \left[\max\left(z_{\min}, -\frac{1}{\alpha}\right), z_{\max}\right].$$

This is mapped to log-space with

$$u = \log(1 + \alpha z).$$

For compact kernels, such as rectangle or triangle kernels, `support()` gives exact finite stencil bounds.

For infinite-support kernels, such as a Gaussian, the exact stencil would be infinite. In that case the implementation uses `cdf` and `ppf` to choose a finite interval containing $1 - \texttt{tail}$ of the positive-domain kernel mass. This is a computational truncation, not a change in the mathematical kernel.

Method summary

Method	Mathematical role	Implementation role
`pdf(z)`	Defines the continuous relative kernel shape $f(z)$.	Not needed for the discrete weights.
`cdf(z)`	Gives integrated probability $F(z)$.	Used to compute exact cell weights $F(z_U) - F(z_L)$.
`support()`	Gives where the kernel can be nonzero.	Used to find exact finite stencil bounds when possible.
`ppf(p)`	Gives the inverse CDF.	Used only to truncate infinite-support kernels by tail probability.

Boundary normalization

On a finite array, some offsets $i + m$ fall outside the available data near the boundaries. The implementation computes the available weight mass

$$D_i = \sum_{m:0\le i+m<n} w_m$$

and returns

$$\boxed{\bar{y}_i \approx \frac{\sum_{m:0\le i+m<n} w_m y_{i+m}}{D_i}}$$

when $D_i > 0$. This preserves constants near the finite-domain boundaries.