Generator Documentation

Janice Wu

October 13, 2025

1 Overview

This module provides a collection of generators for generating financial time series. The core approaches include:

- **Non-parametric pathwise bootstrap** in signature space.
- **Hybrid KRR + residual bootstrap**: parametric drift via kernel ridge on signatures, plus non-parametric residual sampling.
- **KRR on signatures** to learn parametric conditional mean and volatility.
- **ARIMA** (for log-returns; d=0) with state-space simulation.
- **GARCH(p,q)** with optional automatic scaling of returns for numerical stability.

All methods assume evenly spaced sampling with step size $\Delta t =$ `dt` and use a two-channel path embedding $\big(t, x_t\big)$ when computing path signatures.

Dependencies. `numpy`, `iisignature`, `statsmodels` (ARIMA), `arch` (GARCH), `tqdm` (optional).

2 Notation

Let $\{S_t\}$ denote prices, $\{Y_t\} = \{\log S_t\}$ log-prices, and $\{r_t\}$ log-returns:

$$r_t \equiv \log \frac{S_t}{S_{t-1}} = Y_t - Y_{t-1}.$$

Stock process can often be describe as

$$dS_t = \mu(t, S_t)dt + \sigma(t, S_t)dW_t \iff r_t = \tilde{\mu}(t, \{r_\tau\}_{\tau=1}^{t-1}, S_0)dt + \sigma(t, \{r_\tau\}_{\tau=1}^{t-1}, S_0)dW_t$$

For a fixed window length L (`lookback`), we form the 2D path segment

$$\mathbf{P} = \big[(t_0, x_{t_0}), \ldots, (t_{L-1}, x_{t_{L-1}})\big] \in \mathbb{R}^{L\times 2}, \quad t_j = j\,\Delta t,$$

where x is a 1D series (typically returns). The signature (or log-signature) of $\mathbf{P}$ up to level $\ell =$ `sig_level` is computed by `iisignature`.

3 Utility

3.1 `logrets_to_prices(logrets, s0)`

Given log-returns $\{r_t\}_{t=1}^T$ and starting price $S_0 > 0$ (broadcastable to batch shape), the price path is

$$S_t = S_0 \exp\Bigl(\sum_{i=1}^{t} r_i\Bigr), \qquad t = 1, \ldots, T.$$

Shapes: `logrets` of shape $(\ldots, T)$ returns prices with shape $(\ldots, T{+}1)$. Broadcasting on leading axes is supported.

4 Generator Classes

4.1 `BootstrapPathwise`

Idea. Build a library of (signature, future segment) pairs from historical returns windows, then generate by KNN lookup in signature space and appending a sampled neighbor's forward segment.

Fit Log Returns. For each path x (1D log-returns), windows $x_{i-L:i}$ produce

$$\mathbf{P}_i = \begin{bmatrix} t & x \end{bmatrix} \in \mathbb{R}^{L\times 2}, \quad \boldsymbol{s}_i = \mathrm{Sig}_\ell(\mathbf{P}_i), \quad \mathbf{f}_i = x_{i:i+F},$$

where $F =$ `forward`. The library stores $\{\boldsymbol{s}_i\}$ and $\{\mathbf{f}_i\}$.

Generate Log Returns. Given a seed history $x_{1:m}$ with $m \geq L$, iterate until length n_{tot}:

$$\hat{\boldsymbol{s}} = \mathrm{Sig}_\ell\bigl(\begin{bmatrix} t & x_{m-L+1:m} \end{bmatrix}\bigr), \quad d_i = \|\boldsymbol{s}_i - \hat{\boldsymbol{s}}\|_2.$$

Take k nearest indices; sample one neighbor (uniform or softmax in $-d$) and append its forward segment (capped not to exceed the requested horizon).

Key arguments. `lookback`, `sig_level`, `forward`, `dt`, `k`, `neighbor_weighting` ("uniform"/"softmax").

Complexity. Library building is $O(NL)$ signature calls, N windows total. Each step's kNN by brute force is $O(N)$; use partial sort for top-k.

4.2 `HybridKRRBootstrap`

Idea. Predict the drift of the next return via linear-kernel ridge on signatures; obtain a residual by KNN bootstrap.

Training. For each window $x_{i-L:i}$,

$$\boldsymbol{s}_i = \mathrm{Sig}_\ell(\mathbf{P}_i), \qquad y_i = \frac{x_i}{\Delta t}, \qquad \alpha = (K + \lambda I)^{-1} y, \quad K = SS^\top,\ S = [\boldsymbol{s}_i]_i.$$

Then residuals:

$$\varepsilon_i = x_i - (\alpha^\top S \boldsymbol{s}_i)\,\Delta t.$$

Generate Log Returns. At time t, compute $\hat{\boldsymbol{\sigma}}$, predict drift

$$\hat{\mu}_t = \alpha^\top S\hat{\boldsymbol{\sigma}}, \qquad \hat{r}_{t+1} = \hat{\mu}_t\,\Delta t + \varepsilon^\star,$$

where $\varepsilon^\star$ is drawn from $\{\varepsilon_i\}$ of kNN neighbors in signature space (uniform or softmax-weighted).

Key args. `lookback`, `sig_level`, `dt`, `lam`, `k`, `neighbor_weighting`.

4.3 `KRRSignature`

Goal. Learn *returns-native* conditional mean and volatility from signature features, then simulate

$$r_{t+1} = \mu_t\,\Delta t + \sigma_t\sqrt{\Delta t}\,Z_t, \qquad Z_t \sim \mathcal{N}(0,1).$$

Targets. For each window $r_{i-L:i-1}$ (length L), form $\boldsymbol{\sigma}_i$ and set

$$\mu_i = \frac{r_i}{\Delta t}, \qquad \log\sigma_i = \log\left(\frac{\mathrm{std}(r_{i-L:i-1})}{\sqrt{\Delta t}} + 10^{-8}\right).$$

Fit two KRR heads (linear kernel) with separate ridge penalties $\lambda_\mu, \lambda_\sigma$.

Generate Log Returns. At each step: compute signature of the latest L returns, predict μ_t and $\log\sigma_t$, set $\sigma_t = \exp(\log\sigma_t)$ and draw r_{t+1} by the Gaussian rule above. A deterministic (mean) path is available with $Z_t \equiv 0$.

Key args. `lookback`, `sig_level`, `dt`, `lam_mu`, `lam_sig`, optional `random_state`.

4.4 `ARIMAGen`

Model. ARMA(p,q) on log-returns (enforced by d=0). Fitted via `statsmodels`' `ARIMA`. Simulation uses the state-space representation:

$$r_t = \phi(B)\,r_{t-1} + \theta(B)\,\varepsilon_t, \quad \varepsilon_t \sim \mathcal{N}(0,\sigma^2),$$

with standard ARMA polynomials in the backshift operator B.

4.5 `GARCHGen`

Model. Constant-mean GARCH(p,q) on log-returns:

$$r_t = \mu + \varepsilon_t, \qquad \varepsilon_t = \sigma_t z_t, \qquad \sigma_t^2 = \omega + \sum_{i=1}^{p}\alpha_i\varepsilon_{t-i}^2 + \sum_{j=1}^{q}\beta_j\sigma_{t-j}^2,$$

with z_t standard Normal or standardized Student-t.

Scaling. For numerical stability, an internal factor c rescales data $r_t^{(s)} = c\,r_t$ before fitting (`scale="auto"` uses $c = 100$ if $\mathrm{std}(r) < 0.05$). Parameters map back as

$$\mu = \mu^{(s)}/c, \qquad \omega = \omega^{(s)}/c^2, \qquad \alpha_i, \beta_j \text{ unchanged}.$$

Warm-start. Optionally accepts `seed_returns` to roll the volatility recursion across recent history before forecasting.

5 Common Arguments & Validation

- `lookback` L: windows must satisfy $L > 0$ and $L \leq$ length of the seed/history.
- `dt`: positive step size used to build the time channel and to scale drifts/volatilities.
- `sig_level`: positive integer; signature dimension is checked via `iisignature.siglength(2, sig_level)`.
- `k` (kNN): automatically clamped to library size.
- `return_full_path`: when `False`, generators return only the continuation (alignment-friendly for plotting); when `True`, they include the seed.

6 Numerical Notes & Tips

- **Signature stability.** Standardize or winsorize returns before signature extraction if outliers are severe.
- **kNN distances.** "softmax" neighbor weighting with temperature = std of top-k distances smooths transitions.
- **KRR conditioning.** The linear kernel $K = SS^\top$ is solved with a ridge term; λ should increase with feature dimension.
- **ARIMA order.** For returns, use $(p, 0, q)$. Differencing $(d > 0)$ is generally unnecessary for stationary returns.
- **GARCH scaling.** Heed `arch`'s `DataScaleWarning`. The built-in `scale` option addresses convergence and keeps outputs in original units.