Submitted to *Annals of Actuarial Science*

ARTICLE

Supplement to: Package **CovRegpy**: Regularised Covariance Regression and Forecasting in **Python**

Cole van Jaarsveldt[1], Gareth W. Peters[2], Matthew Ames[3], and Mike Chantler[4]

[1]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, United Kingdom of Great Britain and Northern Ireland (UK), cv25@hw.ac.uk [2]Department of Statistics & Applied Probability, University of California, Santa Barbara, California, United States of America (USA), garethpeters@ucsb.edu [3]ResilientML, Melbourne, Australia, matt.ames@resilientml.com [4]School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, UK, m.j.chantler@hw.ac.uk

Abstract

This paper serves as a formal supplement to "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python". In this work, algorithmic variations are presented for Singular Spectrum Analysis (SSA) and Singular Spectrum Decomposition (SSD). Additionally, the pseudo-code for the Empirical Mode Decomposition (EMD) sifting procedure as well as the original X11 sifting procedure, as opposed to the numerous variations constructed since its inception, are presented in the appendix. The EMD sifting procedure is an unnecessary tautology to those familiar with the algorithm as EMD refers exclusively to the sifting procedure as opposed to the subsequent Huang transform (HT) as well, but it is included here for clarification. The entire procedure is referred to as the Hilbert-Huang transform (HHT) as a tribute to the creator (Huang) of EMD. Implicit factors and Regularised Covariance Regression (RCR) are presented in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python" with little detail on combining the two models to forecast covariance with the financial setting. Explicit forecasting methods are presented herein as well as a lagging alternative with both presented in a case study. Furthermore, the formulation of the non-linear multivariate extension of GARCH Dynamic Conditional Correlation (DCC) is presented herein to supplement the script included in the associated software package. Algorithmic extensions developed for SSA and SSD (both presented in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python") are presented herein to assist with the implementation included in the software package.

Keywords: Portfolio Optimisation, Regularised Covariance Regression (RCR), Empirical Mode Decomposition (EMD), Singular Spectrum Analysis (SSA), Singular Spectrum Decomposition (SSD), X11, Implicit Factors, Risk Premia Parity, Risk Parity, Long\Short Equity;

Data Availability Statement

The data, Python code, figures, and other replication materials that support this study are openly available in CovRegpy at:

https://zenodo.org/doi/10.5281/zenodo.10827714.

A regularly maintained version catalogue of the software package CovRegpy, as well as detailed installation instructions and examples for both experienced and new users, can be found here:

https://github.com/Cole-vJ/CovRegpy.

8. Supplement Introduction

The Python functions that do not form the main contribution of this work (i.e. not `CovRegpy.py`, `CovRegpy_DCC.py`, `CovRegpy_RPP.py`, `CovRegpy_SSA.py`, `CovRegpy_SSD.py`, or `CovRegpy_X11.py` which are outlined in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python") can be found in the following directory for additional scripts which are used in the the supplemental case studies in Section 12, Section 13, and Section 14:

`https://github.com/Cole-vJ/CovRegpy/tree/main/CovRegpy_add_scripts`.

The case studies which are presented in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python" are contained in:

`https://github.com/Cole-vJ/CovRegpy/tree/main/aas_examples`.

The additional case studies and examples (some of which are presented in this supplement) can be found in:

`https://github.com/Cole-vJ/CovRegpy/tree/main/examples`.

Section 9 describes the explicit forecasting methods available in this package. Single neuron neural network forecasting is discussed in Section 9.1, before Gaussian process forecasting and instantaneous frequency forecasting are discussed in Sections 9.2 and 9.3, respectively. The scripts for single neuron neural network forecasting (Section 9.1) and Gaussian process forecasting (Section 9.2) can be found in:

`https://github.com/Cole-vJ/CovRegpy/blob/main/CovRegpy_add_scripts/CovRegpy_forecasting.py`,

with the instantaneous frequency forecasting being in:

`https://github.com/Cole-vJ/CovRegpy/blob/main/CovRegpy_add_scripts/CovRegpy_IFF.py`.

Section 10 presents the notation used in `CovRegpy_DCC.py` for the DCC multivariate generalized autoregressive conditional heteroskedasticity (MGARCH) covariance forecasting framework. Section 11 details the extensions proposed herein for SSA (Section 11.1) and SSD (Section 11.2).

The SSA extensions developed herein, namely decomposing SSA (Section 11.1.1) and Kolmogorov-Smirnov SSA (Section 11.1.2), are included in `CovRegpy_SSA.py` as optional inputs. The extensions available for SSD, namely modified embedding (Section 11.2.1) and the scaling factor (Section 11.2.2) were proposed in Bonizzi et al. (2014). These extensions were not included in the original presentation of the technique owing to the focus of the paper and the view that these extensions remove the strength of the downsampling without decimation upon which SSD is based. The initial trend adjusting SSD is presented herein for the first time as in the original work an embedding factor of $L = T/3$ is strictly used. Within this extended context, the embedding ratio would be '3' - we make this flexible and adjustable with our software package and we motivate this in Section 11.2.3.

Section 12 is a case study demonstrating covariance regression on synthetic data constructed using the Cholesky decomposition to correlate returns based on some underlying sinusoidal structures and as such will have a well-defined solution. In Section 13, the explicit forecasting techniques (Section 9) are displayed on a synthetically generated amplitude and frequency-modulated sinusoidal structure. Frequency and amplitude-modulated structures are common in time series analysis and occur regularly in several fields. The case studies are concluded with Section 14 that used 5 very large market cap S&P500 stocks to demonstrate a real-world case study in which the techniques in this software package are used to make portfolio weighting decisions based on risk appetite and covariance forecasting.

This supplement is concluded in Section 15 which includes easily translatable pseudo-code for both the EMD algorithm (Algorithm 2) and the X11 algorithm (Algorithm 3).

Note to Users

Extensive studies have been conducted on the robustness of and potential for algorithmic extensions to EMD, see van Jaarsveldt et al. (2023) and van Jaarsveldt et al. (2021). Whilst not formally compared against SSA and SSD in those works or in this work, the extensions to SSA and SSD developed or presented in this work were an attempt to robustify these methods. These variations are by no means exhaustive and are presented herein for completeness sake and one should use this package in conjunction with the AdvEMDpy package for regularised covariance regression and forecasting. Most importantly, as already detailed in van Jaarsveldt et al. (2023) and van Jaarsveldt et al. (2021), these algorithms are not necessarily competitors and can be used in conjunction to exceed the resolution capabilities of any one individual algorithm. This work, as well as the others already mentioned, promote interdisciplinary exchange and collaboration.

9. Dynamic Covariance Forecasting Models for Time Series

In addition to time series decomposition using the implicit factor models as outlined in Section 4 of "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python" and RCR used to calculate the unattributable (or systemic or contemporaneous) ($\mathbf{\Psi}$) and attributable (or structural) covariance ($\mathbf{B}$) in Section 5 of "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python", one would benefit from work being able to explicitly forecast time series rather than merely being able to apply RCR to lagged data. The work in this section builds upon work done in van Jaarsveldt et al. (2023) and van Jaarsveldt et al. (2021) on EMD and the forecasting of the implicit factors.

These forecasting techniques are introduced here and should be used with care as more assumptions are required to formally forecast the independent variable rather than lagging the independent variable against the dependent variables. Several robust forecasting techniques are listed here for those intending to formally forecast the independent variables. A short cast study of these forecasting techniques is conducted in Section 13 on a specifically constructed synthetic modulated time series.

9.1 Neural Network

This forecasting methodology is developed here and is based upon Deng et al. (2001) and van Jaarsveldt et al. (2023) on using multivariate regression to accurately extrapolate a smooth curve with the intent of estimating the locations of the nearest extrema for fitting extrema envelopes in EMD. The following work regarding forecasting using a single-neuron neural network reduces to a multivariate regression model in the absence of an activation function or rather with the identity function as the activation function. The objective function is:

$$\arg\min_{\mathbf{w}} ||\mathbf{wP} - \mathbf{y}||_2^2 + \alpha||\mathbf{w}||_2^2, \tag{1}$$

with the weighting vector, $\mathbf{w}$ being of dimension k which is the fitting window dimension and is constructed as:

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_k \end{bmatrix}, \tag{2}$$

with $\mathbf{P}$ being constructed similarly as Equation (20) where the time series is embedded (with an embedding dimension or the number of samples being m) in a matrix as:

$$\mathbf{P} = \begin{bmatrix} y_{T-(m+k)} & y_{T-(m+k-1)} & \cdots & y_{T-(k+1)} \\ y_{T-(m+k-1)} & y_{T-(m+k-2)} & \cdots & y_{T-k} \\ \vdots & \vdots & \ddots & \vdots \\ y_{T-(m+1)} & y_{T-m} & \cdots & y_{T-2} \end{bmatrix}, \tag{3}$$

with T being the length of the time series (first element y_0; final element y_{T-1}) and with the target vector, $\mathbf{y}$, being defined as:

$$\mathbf{y} = \begin{bmatrix} y_{T-m} & y_{T-(m-1)} & \cdots & y_{T-1} \end{bmatrix}. \tag{4}$$

The notation above, and throughout, is analogous to `Python` for easy reference to accompanying software and Equations (4) and (8), respectively, such that:

$$y_{T-m} := \texttt{y[-m]} \text{ and } \mathbf{y}_2 := \texttt{np.append(y[-int(k - 1):], y\_forecast[:1])} \tag{5}$$

Once the weighting vector, $\mathbf{w}$, has been estimated, it is iteratively applied to the end of the time series to extrapolate the time series. That is, with y_T being the first point extrapolated, it is calculated as:

$$y_T = \mathbf{w}\mathbf{y}_1, \tag{6}$$

with $\mathbf{y}_1$ being:

$$\mathbf{y}_1 = \begin{bmatrix} y_{T-k} & y_{T-(k-1)} & \cdots & y_{T-1} \end{bmatrix}^T, \tag{7}$$

and with,

$$\mathbf{y}_2 = \begin{bmatrix} y_{T-(k-1)} & y_{T-(k-2)} & \cdots & y_{T-1} & y_T \end{bmatrix}^T, \tag{8}$$

etc. The uncertainty in the prediction of each successive time series realisation grows exponentially owing to the uncertainty inherent in each extrapolation.

9.2 Gaussian Process

Those familiar with Gaussian processes would possibly conclude that it would be well-suited in its original state with a relatively simple kernel. It would be well suited to be applied directly to structures extracted using X11, SSA, and SSD as these structures would be nearly homogeneous in their frequency content, whereas structures isolated using EMD can have more complex frequency structures and content. More complex and compound kernels are needed to capture and accurately forecast these structures.

In this setting a periodic kernel, the most well-known of which being the Exponentiated Sine Squared kernel, is highly recommended. When the structures are more complex, as in IMFs extracted using EMD, more complex kernels are required. It is recommended that several compound kernels be used that accommodate decaying periodic structures. The better-suited kernel would combine Exponentiated Sine Squared kernels with radially decaying kernels such as the Radial Basis kernel and the Rational Quadratic kernel. These are discussed and demonstrated more thoroughly in Section 13.1.3

9.3 Instantaneous Frequency Forecasting

This technique takes advantage of the relative sparsity of many seemingly complex structures in the frequency domain. IMFs (the structures extracted using EMD) are frequency and amplitude-modulated time series. One could then independently forecast both the instantaneous frequency and the instantaneous amplitude followed by a Fourier transform into the temporal domain. As this is a new method, only linear IFF is provided in this package, but this is not essential and more complex frequency and amplitude forecasting is possible. It remains to fit the phases of the structures.

By referring to Equation (11) and Equation (13) in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python" one can simply extrapolate both the instantaneous amplitude, $a(t)$, and the instantaneous frequency, $\omega(t)$, before transforming the result back in the temporal domain. To demonstrate, a sine wave such as $y = \sin(t)$ has a relatively complex representation in the temporal domain, whereas it has a simple instantaneous amplitude, $a(t) = 1$, and instantaneous frequency, $\omega(t) = 1$. The only difficulty lies in the phase which needs to be fitted as the phase is lost in the integral. A simple, yet effective example, is demonstrated in Section 13.1.4.

10. Dynamic Conditional Correlation

This serves as a very brief (owing to the scope and focus of this work) description of the steps involved in setting up one's equations for optimisation in the forecasting of the covariance using Dynamic Conditional Correlation (DDC) multivariate generalized autoregressive conditional-heteroskedasticity (MGARCH) - this is implementable using the `CovRegpy_DCC.py` script provided in this script for fair comparison of our model with the MGARCH models with different frameworks. For a more in-depth and formal review of MGARCH models and their history as multivariate extensions of traditional GARCH models, one can see Bauwens et al. (2006).

With regards to the model at hand, let y_t be an N-dimensional stochastic process with I_{t-1} being the sigma field generated by past discrete realisations of the process y_t up to and including time $t-1$, and with θ being a finite parameter vector the DCC MGARCH model can be defined as:

$$y_t = \mu_t(\theta) + \epsilon_t(\theta), \tag{9}$$

with $\mu_t(\theta)$ being the conditional mean of the process and with

$$\epsilon_t(\theta) = \mathbf{H}_t^{\frac{1}{2}}(\theta) z_t, \tag{10}$$

with $\mathbf{H}_t$ being a positive $N \times N$ definite matrix. Let z_t be an N-dimensional vector with the first two centred moments of z_t defined as:

$$\begin{aligned} \mathbb{E}[z_t] &= \mathbf{0_N}, \text{ and} \\ \text{Var}(z_t) &= \boldsymbol{I_N}, \end{aligned} \tag{11}$$

where $\mathbf{0}_N$ is the N-dimensional zero vector and with $\boldsymbol{I_N}$ being the $N \times N$ identity matrix. With z_t defined as above in Equation (11), the conditional distribution of y_t as defined in Equation (9) is then calculated as:

$$\begin{aligned} \text{Var}(y_t | I_{t-1}) &= \text{Var}_{t-1}(y_t) \\ &= \mathbf{H}_t^{\frac{1}{2}}(\theta) \text{Var}_{t-1}(z_t) \left(\mathbf{H}_t^{\frac{1}{2}}(\theta) \right)' \\ &= \mathbf{H}_t(\theta). \end{aligned} \tag{12}$$

From Equation (12) it follows that given any positive definite matrix $\mathbf{H}_t(\theta)^{\frac{1}{2}}$ defined as above, the conditional covariance of y_t is given by $\mathbf{H}_t(\theta)$. The different ways of estimating $\mathbf{H}_t(\theta)$ divide the different MGARCH models broadly into 3 groups namely: direct extensions of GARCH

(Bollerslev et al. (1988)), linear combinations of direct extensions of GARCH models, and non-linear combinations of direct extensions. DDC MGARCH falls under the final categorisation and was introduced in Engle (2002) as:

$$\mathbf{H}_t = \mathbf{D}_t \mathbf{R}_t \mathbf{D}_t, \tag{13}$$

with,

$$\mathbf{D}_t = \text{diag}\{h_{11t}^{\frac{1}{2}}, \ldots, h_{NNt}^{\frac{1}{2}}\}, \tag{14}$$

with $h_{iit} = \sigma_i^2 t$ being modelled as a GARCH(p, q) process such that:

$$\sigma_i^2 t = \omega_{iit} + \sum_{j=1}^{p} \beta_j \sigma_{i(t-j)}^2 + \sum_{j=1}^{q} \alpha_j \epsilon_{i(t-j)}^2, \tag{15}$$

and with,

$$\mathbf{R}_t = (1 - a - b)\bar{\mathbf{R}} + a\mathbf{U}_{(t-1)} + b\mathbf{R}_{(t-1)}, \tag{16}$$

where,

$$\mathbf{U}_t = u_t u_t^T, \tag{17}$$

where,

$$u_t = \mathbf{D}_t^{-1} r_t, \tag{18}$$

with r_t being the returns at time t. With this framework mostly repeated verbatim from Engle (2002), the likelihood function to be minimised with $r_t | I_{t-1} \sim \mathcal{MVN}(\mathbf{0_N}, \mathbf{H}_t)$ and with $\mathcal{MVN}(\cdot, \cdot)$ being the multivariate normal distribution, becomes:

$$\begin{aligned}
L &= -\frac{1}{2} \sum_{t=1}^{T} \left(N\log(2\pi) + \log|\mathbf{H}_t| + r_t^T \mathbf{H}_t^{-1} r_t \right) \\
&= -\frac{1}{2} \sum_{t=1}^{T} \left(N\log(2\pi) + \log|\mathbf{D}_t \mathbf{R}_t \mathbf{D}_t| + r_t^T \mathbf{D}_t^{-1} \mathbf{R}_t^{-1} \mathbf{D}_t^{-1} r_t \right) \\
&= -\frac{1}{2} \sum_{t=1}^{T} \left(N\log(2\pi) + 2\log|\mathbf{D}_t| + \log|\mathbf{R}_t| + u_t^T \mathbf{R}_t^{-1} u_t \right) \\
&= -\frac{1}{2} \sum_{t=1}^{T} \left(N\log(2\pi) + 2\log|\mathbf{D}_t| + \log|\mathbf{R}_t| + r_t^T \mathbf{D}_t^{-1} \mathbf{D}_t^{-1} r_t - u_t^T u_t + u_t^T \mathbf{R}_t^{-1} u_t \right).
\end{aligned} \tag{19}$$

11. Implicit Factor Model Extensions

As detailed in 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python', the title refers to the relative difficulty in relating these factors to easily observable factors such as, for example, the factor models proposed in Fama and French (1993) and Fama and French (2015) (examples of explicit factor models) that relate the returns of assets above the 'risk-free' rate to other observable financial factors such as the book to market value. An early example of what one could refer to as an implicit factor model as a principal portfolio model can be used in the accompanying script titled `CovRegpy_PCA.py` with an easily repeatable script example using the five largest assets in the S&P500 (by market cap) at the time of the script's creation.

11.1 Singular Spectrum Analysis

As already alluded to in 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python', the major drawback of SSA in its originally proposed form, see Hassani (2007), is that it was a trend filtering technique rather than a formal decomposition algorithm. It can be used as a decomposition algorithm if one makes a relatively minor adjustment to the Grouping step of SSA in Section 3.4.3 in 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python'.

11.1.1 Decomposing Singular Spectrum Analysis Extension

By slightly modifying this algorithm originally proposed in Hassani (2007) one can optimally (in some sense) decompose the time series. This has been named the Decomposing Singular Spectrum Analysis (D-SSA) and is done by modifying the Grouping step. Rather than group all the structures, one can keep all the components whose indices are listed in $\boldsymbol{I}$ separate and apply the Diagonal Averaging step to each structure independently. One can observe in Figure 1 the decomposition of the example time series.

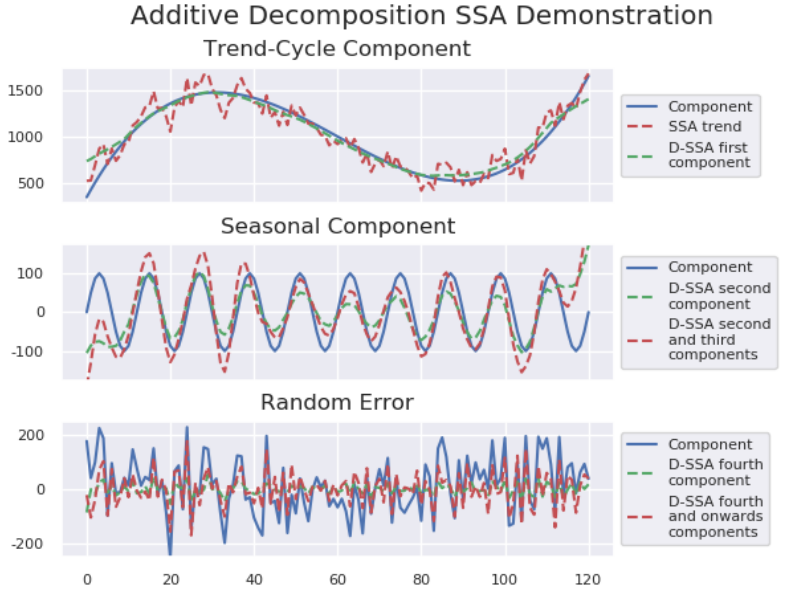

Figure 1. SSA and D-SSA trend estimate and component isolation compared against the corresponding underlying structures.

The above decomposition can be achieved by running the below code. All plots within this section can be seen by running the script `CovRegpy_SSA.py` which contains the below code.

```
ssa_trend, ssa_decomp = CovRegpy_ssa(x11_time_series, L=10, est=8)
```

One can note the seasonal component extracted using D-SSA is more smooth than the seasonal component extracted using X11. This is not true for the trend-cycle component which contains more noise than the corresponding component extracted using X11 - it is because of this observation that KS-SSA was developed.

11.1.2 Kolmogorov-Smirnov Singular Spectrum Analysis Extension

Another interesting algorithmic variation to traditional SSA is named Kolmogorov-Smirnov Singular Spectrum Analysis (KS-SSA) uses Gaussian distribution or Normality assumptions about the distribution of the errors about the estimated trend to optimise (MSE) the initial trend extraction. Using the traditional Kolmogorov-Smirnov framework, by testing all values for L and each resulting subset, $\boldsymbol{I}$, of the resulting decomposition one can arrive at a trend that minimizes the Kolmogorov-Smirnov test statistic.

If one wants to optimise both `L` and `est` using the described Kolmogorov-Smirnoff framework, then the relevant code inputs are `KS_test`, `plot_KS_test`, and `KS_scale_limit`. To perform the Kolmogorov-Smirnoff optimisation one must have `KS_test=True`. To observe each incremental plot of the Kolmogorov-Smirnoff, one must have `plot_KS_test=True`. This results in numerous plots like the images on the right of Figure 2.

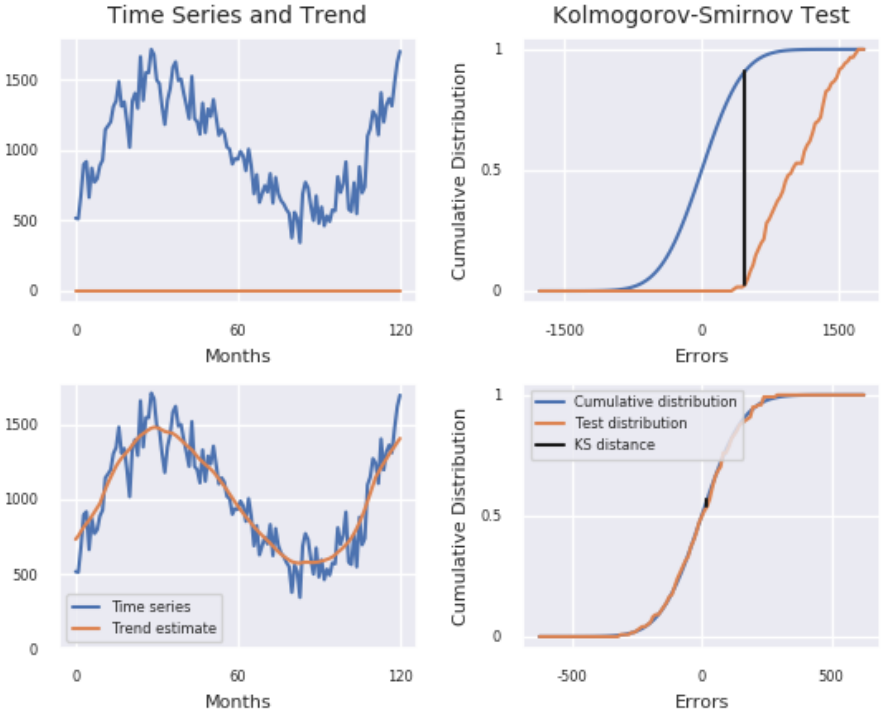

Figure 2. Kolmogorov-Smirnov Singular Spectrum Analysis (KS-SSA) Kolmogorov-Smirnov test statistic plot demonstrating cumulative distribution test function and KS distance.

A nuance that was discovered while exploring this technique is the need for a lower bound on the standard deviation of the error distribution about the underlying trend. If this lower bound did not exist (`KS_scale_limit=1` in the examples in Figure 2 and Figure 3), the trend would try to approximate the time series as closely as possible and incorporate the noise as well. Further studies should be conducted to analyse the effect of this limit on trend estimation.

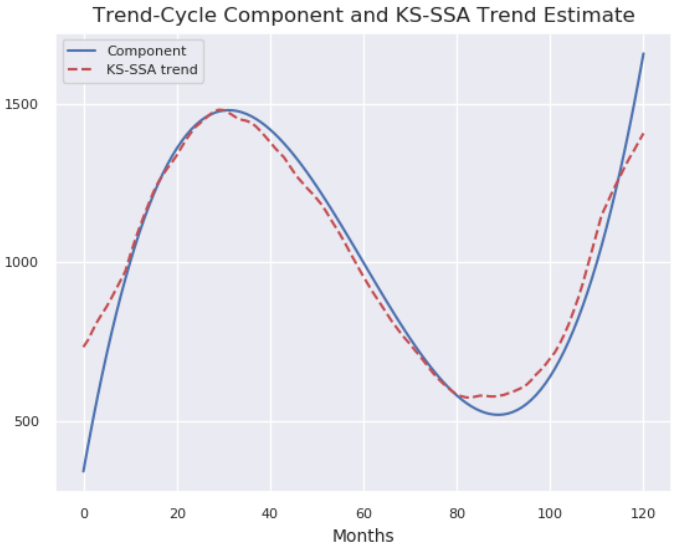

Figure 3. Kolmogorov-Smirnov Singular Spectrum Analysis (KS-SSA) optimised trend estimation - compare against the top image of Figure 1.

11.2 Singular Spectrum Decomposition

As stated in 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python', the major difference between SSA and SSD is the formalisation of the relatively narrow bandwidth downsampling (without decimation) to isolate structures based on energy densities of the time series power spectral density (PSD). The additional features and steps outlined in Sections 11.2.1 and 11.2.2 are not necessary, but were included in the original formulation of the technique in Bonizzi et al. (2014). These two additions are criticised here as they (particularly the modified embedding step detailed in 11.2.1) encourage or rather coerce the extracted structure to be more cyclical and constant in frequency - this obstructs any meaningful analysis of modulated structures which can be extracted using downsampling with an appropriately selected bandwidth. Downsampling without decimation, as discussed in van Jaarsveldt et al. (2023), is an effective filtering or smoothing technique.

11.2.1 Modified Embedding in SSD

This is a modified version of the embedding presented in Equation (17) in Section 3.4.1 of 'Regularised Covariance Regression and Forecasting in Python' and repeated here in Equation (20) for easy reference:

$$\boldsymbol{X} = [\boldsymbol{X}_1(t), \ldots, \boldsymbol{X}_K(t)], \tag{20}$$

with $\boldsymbol{X}_j(t) = [x(t_j), \ldots, x(t_{j+L-1})]^T$, $K = T - L + 1$, and L being the embedding dimension. The modified embedding of the time series, $\boldsymbol{X}^{mod}$, is calculated as:

$$\boldsymbol{X}^{mod} = [\boldsymbol{X}_1^{mod}(t), \ldots, \boldsymbol{X}_K^{mod}(t)], \tag{21}$$

such that $\boldsymbol{X}_j^{mod}(t) = [x(t_j), \ldots, x(t_T), x(t_1), \ldots, x(t_{j-1})]^T$ which is referred to in Bonizzi et al. (2014) as 'wrapping' the time series around to promote a more cyclical structure. In Bonizzi et al. (2014) this step and the subsequent step detailed in Section 11.2.2 are performed upon the downsampled component to coerce out a more cyclical structure.

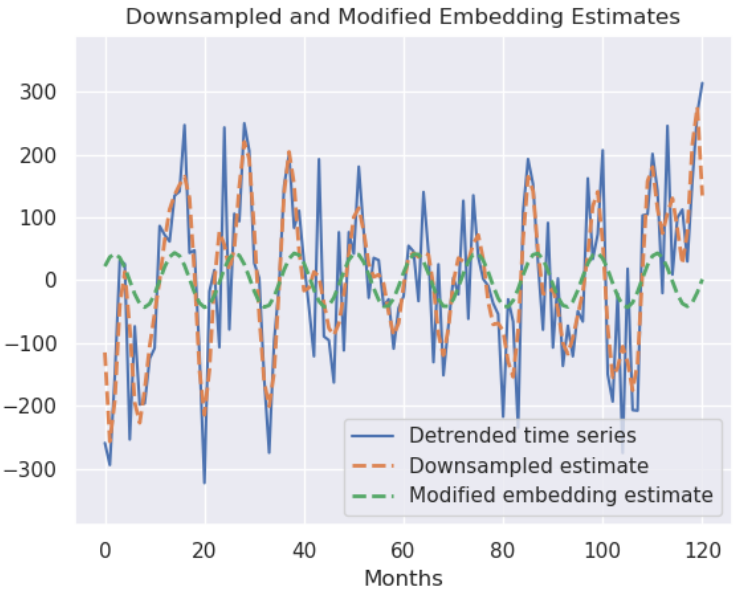

Figure 4. Plot demonstrating downsampled estimate and the modified embedding estimate of downsampled estimate with both being estimates of the next structure to be isolated from the detrended time series.

11.2.2 Scaling Factor in SSD

Once the component is extracted or isolated using the modified embedding step, the structure is scaled using a scaling factor to optimise the MSE error of the optimised component compared against the remaining time series:

$$min_a||ag_j - v_j||_2^2, \tag{22}$$

with the component isolated being $\tilde{g}_j = a^{opt} g_j$. This serves to scale the component that has been smoothed and reduced as a result of the modified embedding discussed in Section 11.2.1 that results in a more cyclical structure but scales down these structures that may also be amplitude modulated.

11.2.3 Initial Trend Adjusting Singular Spectrum Decomposition Extension

As stated in Section 3.5.1 of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python', an embedding factor of $L = T/3$ is used if a significant trend is detected. An embedding ratio in this setting would be 3, but with the Initial Trend Adjusting Singular Spectrum Decomposition (ITA-SSD) technique such that this embedding ratio can be adjusted to better extract the initial trend without hindering the remainder of the algorithm the initial trend extraction can be both optimised, robustified, and automated. If an initial significant trend is not sufficiently and wholly removed, it will be distributed amongst the remaining isolated components - this is referred to as leakage or more specifically frequency leakage.

In Figure 6, Equation (22) of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python' with the initialised values as in Equation (24) of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python' is plotted as well as dashed lines denoting the fixed means as in Equation (23) of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python'. Each Gaussian function is plotted as well as the cumulative function to demonstrate the initialised unfitted function.

Figure 5. Plot demonstrating scaling factor used to scale decomposition component before extraction.

In Figure 7, Equation (22) of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python' has been fitted to the Energy spectral density of the detrended time series. Once this has been fitted, Equation (27) of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python' is used to calculate the frequency boundary or bandwidth that will be used to estimate this component. This bandwidth is then transformed into the temporal space where it is embedded as in Section 11.2.1.

The seasonal component is very easily extracted using SSD owing to the downsampling component of the algorithm followed by the modified embedding step which results in smoothing. The ITA-SSD algorithmic variation is far more robust in estimating the original trend. These algorithmic variations are therefore well-suited to both remove trends and fixed-frequency seasonal components.

12. Supplemental Case Study 1: Regularised Covariance Regression

In this case study 15 synthetic implicit factors are constructed with sinusoids with a random phase with frequencies of annual, semi-annually, tri-annually, et cetera. This case study can be found in `CovRegpy_CASE_STUDY_LASSO_synthetic.py`. The base covariance of the 5 synthetic assets is constructed using 3 years' worth of data from the five largest assets in the S&P 500 at the time. Cholesky decomposition is used to correlate the structures daily after which direct RCR and LASSO RCR are used to estimate the true parameters as in Equation (26) in Section 12.2 and Section 12.3, respectively.

12.1 Cholesky Decomposition and Instantaneous Correlation

In Benoit (1924), Cholesky decomposition was proposed where a Hermitian positive-definite matrix, $\mathbf{\Sigma}$, can be uniquely decomposed such that:

$$\mathbf{\Sigma} = \mathbf{L}\mathbf{L}^{*}, \tag{23}$$

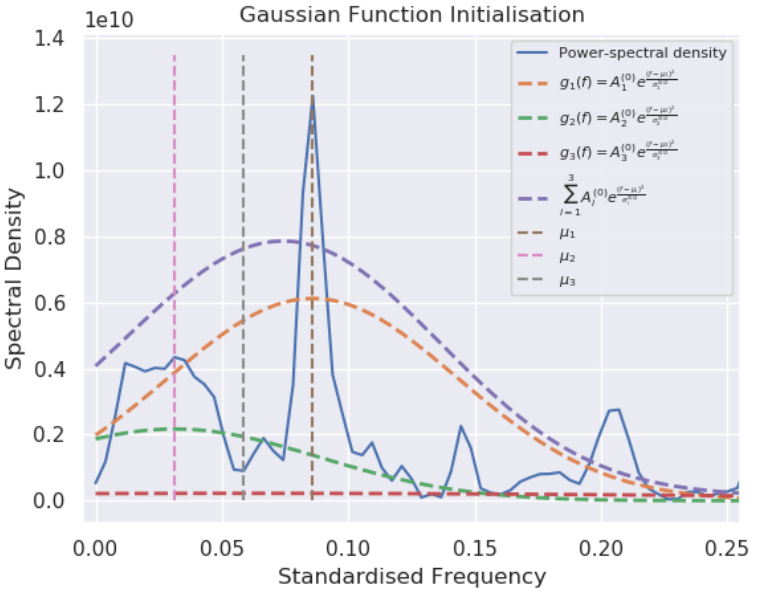

Figure 6. Plot of initialisation of Gaussian functions to be fitted to power spectral densities for the downsampling step of SSD.

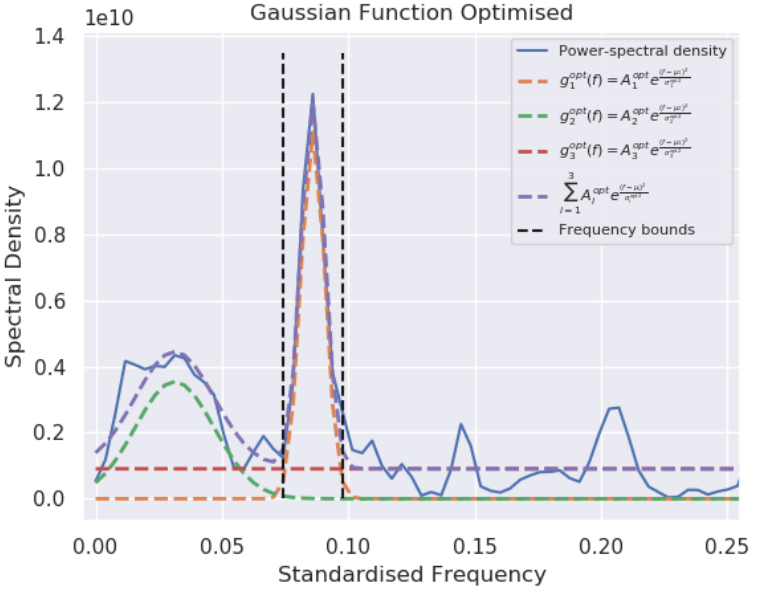

Figure 7. Plot of fitting of Gaussian functions to power spectral densities for the downsampling step of SSD.

where $\mathbf{L}$ is a lower triangular matrix consisting of real positive values along the diagonals and with $\mathbf{L}^*$ the conjugate transpose of $\mathbf{L}$. If $\mathbf{\Sigma}$ is real, then it is symmetric and positive-definite which can then be a covariance matrix. It can be shown that, given a matrix of realisations of the appropriate size, $\boldsymbol{x} \in \mathbb{R}^n$, with covariance matrix $\mathbf{I}_n$, i.e. that are uncorrelated multivariate standard normal,

the covariance of the product of $\mathbf{L}$ and $\boldsymbol{x}$ can be calculated as:

$$\text{cov}(\mathbf{L}\boldsymbol{x}) = \mathbb{E}[\mathbf{L}\boldsymbol{x}\boldsymbol{x}^T\mathbf{L}^T] = \mathbf{L}\mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T]\mathbf{L}^T = \mathbf{L}\mathbf{I}_n\mathbf{L}^* = \boldsymbol{\Sigma}, \tag{24}$$

which can be used to correlate a set of uncorrelated observations to the desired covariance of $\boldsymbol{\Sigma}$. To ensure a sensible and practical, base covariance structure, $\boldsymbol{\Psi}$, the covariance of the log-returns for MSFT, AAPL, GOOGL, AMZN, and TSLA over the period from 31 December 2018 until 1 January 2022. The covariance for this period is:

$$\boldsymbol{\Psi} = \begin{bmatrix} 2.5571 \times 10^{-4} & 1.4417 \times 10^{-4} & 1.4559 \times 10^{-4} & 1.7278 \times 10^{-4} & 2.0255 \times 10^{-4} \\ 1.4417 \times 10^{-4} & 1.9108 \times 10^{-4} & 1.2254 \times 10^{-4} & 1.3834 \times 10^{-4} & 1.5510 \times 10^{-4} \\ 1.4559 \times 10^{-4} & 1.2254 \times 10^{-4} & 1.9731 \times 10^{-4} & 1.5425 \times 10^{-4} & 1.4630 \times 10^{-4} \\ 1.7278 \times 10^{-4} & 1.3834 \times 10^{-4} & 1.5425 \times 10^{-4} & 2.0196 \times 10^{-4} & 1.8464 \times 10^{-4} \\ 2.0255 \times 10^{-4} & 1.5510 \times 10^{-4} & 1.4630 \times 10^{-4} & 1.8464 \times 10^{-4} & 9.3950 \times 10^{-4} \end{bmatrix}. \tag{25}$$

The covariance in Equation (25) is displayed graphically in Figure 8. This makes the comparison with the base, unattributable covariance ($\boldsymbol{\Psi}$) derived using direct RCR and LASSO RCR more visually interpretable. It is this base covariance that is used in Equation (28) to construct the dynamic covariance structure over the 3 years.

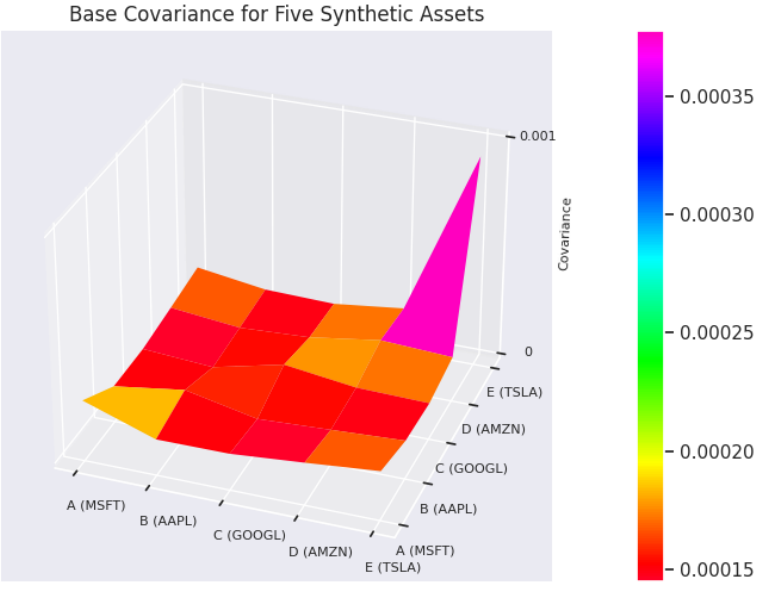

Figure 8. Base covariance of five synthetic assets derived from 3 years' returns of MSFT, AAPL, GOOGL, AMZN, and TSLA over the period from 31 December 2018 until 1 January 2022.

Without lack of generalisation, the coefficients for the $\mathbf{B}$ matrix are randomly generated such that they are all elements of the set $\mathbf{B}_{ij} \in \{-1, 0, 1\}$. The coefficients used can be seen below:

$$\mathbf{B} = \begin{bmatrix} 1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & -1 & 0 & -1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 1 & -1 \\ 0 & 0 & 1 & -1 & 1 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 1 & 0 & 1 \end{bmatrix}. \tag{26}$$

Three years' worth of covariance data needs to be generated to construct instantaneous covariance over the same period. Fifteen structures are generated to replicate annual structures, semi-annual structures, quarterly structures, monthly structures as well as many others to construct the synthetic example. In the first row, the annual structure is constructed, the semi-annual structure is in the second row and so on as demonstrated in Equation (27).

$$\boldsymbol{x} = \begin{bmatrix} \cos\left(2\pi \times \left(\frac{3\times 0}{1097} + u_1\right)\right) & \cos\left(2\pi \times \left(\frac{3\times 1}{1097} + u_1\right)\right) & \cdots & \cos\left(2\pi \times \left(\frac{3\times 1097}{1097} + u_1\right)\right) \\ \cos\left(2\pi \times \left(\frac{6\times 0}{1097} + u_2\right)\right) & \cos\left(2\pi \times \left(\frac{6\times 1}{1097} + u_2\right)\right) & \cdots & \cos\left(2\pi \times \left(\frac{6\times 1097}{1097} + u_2\right)\right) \\ \vdots & \vdots & \ddots & \vdots \\ \cos\left(2\pi \times \left(\frac{45\times 0}{1097} + u_{15}\right)\right) & \cos\left(2\pi \times \left(\frac{45\times 1}{1097} + u_{15}\right)\right) & \cdots & \cos\left(2\pi \times \left(\frac{45\times 1097}{1097} + u_{15}\right)\right) \end{bmatrix}, \tag{27}$$

with $u_i \sim \mathcal{U}(0, 1)\ \forall\ i \in \{1, \ldots, 15\}$ to ensure a random phase. Therefore, the covariance of the synthetic assets at time point it is then calculated as:

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Psi} + \mathbf{B}\boldsymbol{x}_t\boldsymbol{x}_t^T\mathbf{B}^T, \tag{28}$$

with $\boldsymbol{x}_t$ being the t^{th} column of matrix $\boldsymbol{x}$. With this framework, the covariance structure over the three years under investigation is synthesized. With $\left[\boldsymbol{\Sigma}_t\right]_{jj}$ being the variance of asset j at time point t. With this, in Figure 9 , $\left[\boldsymbol{\Sigma}_t\right]_{jj}$ is plotted for $j \in \{1, 2, 3, 4, 5\}$.

With this constructed dynamic covariance over the three years, the daily logarithmic returns are then calculated using Equation (24). At each time point (t) the returns are initialised as:

$$\boldsymbol{r} = [r_1, r_2, r_3, r_4, r_5]^T, \tag{29}$$

with $r_k \sim \mathcal{N}(r, r)\ \forall\ k \in \{1, \ldots, 5\}$ and r being the approximate daily risk-free rate of $r = \frac{0.01}{365}$. The returns are initialised before being correlated using Equation (24) as:

$$\boldsymbol{r}_t = \mathbf{L}_t\boldsymbol{r}, \tag{30}$$

with $\mathbf{L}_t$ being the lower triangular matrix derived using Cholesky decomposition as shown in Equation (23) with $\boldsymbol{\Sigma}_t$ being the covariance of the synthetic assets at time point t. This results in the cumulative synthetic returns observable in Figure 10.

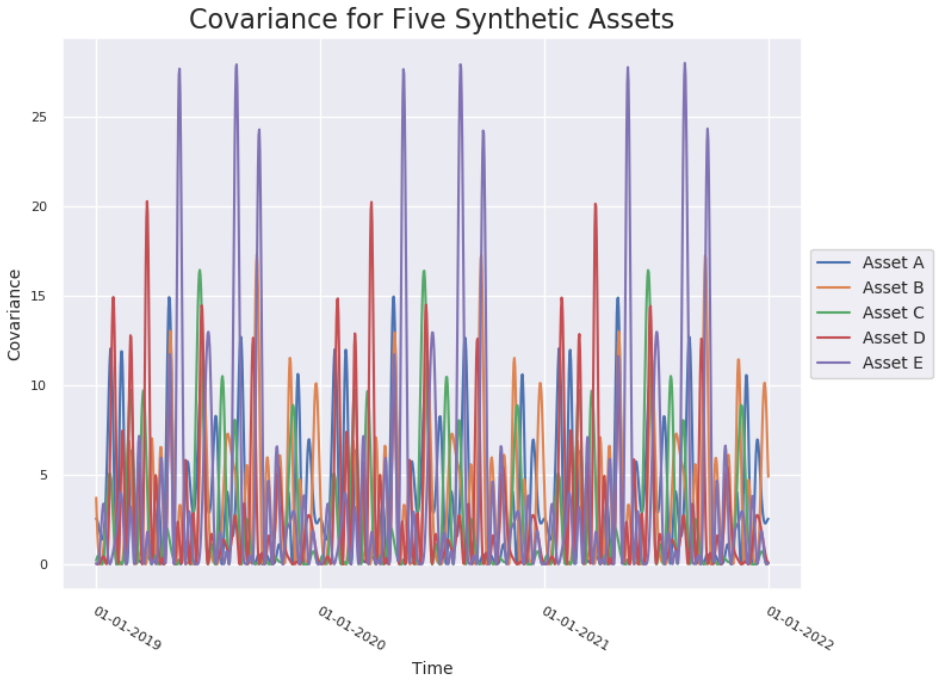

Figure 9. Dynamic covariance structure of five synthetic assets over the three years from 1 January 2019 until 1 January 2022.

12.2 Direct Estimation

By applying the direct RCR, as in Section 5.3 in in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python", to this synthetic data one can arrive at Figure 11. The code in `CovRegpy_CASE_STUDY_LASSO_synthetic.py` to calculate the underlying coefficients can be seen below.

```
B_est, Psi_est = \
    cov_reg_given_mean(A_est=np.zeros_like(coef),
                       basis=spline_basis_transform, x=x[:, :-1],
                       y=returns_subset.T, iterations=10,
                       technique='direct')
```

It can be observed in Figure 11 that the estimated coefficients very closely estimate the true underlying coefficients. This is confirmation of the effectiveness of the estimation method via the random error formulation of covariance regression. One should note the impressive resolution of the coefficients given the instantaneous Cholesky decomposition at each time point. It should be noted that another possible solution would be a complete inversion of all the solutions owing to the sign-indifference of the solution, provided that the signs of all coefficients are identically opposite.

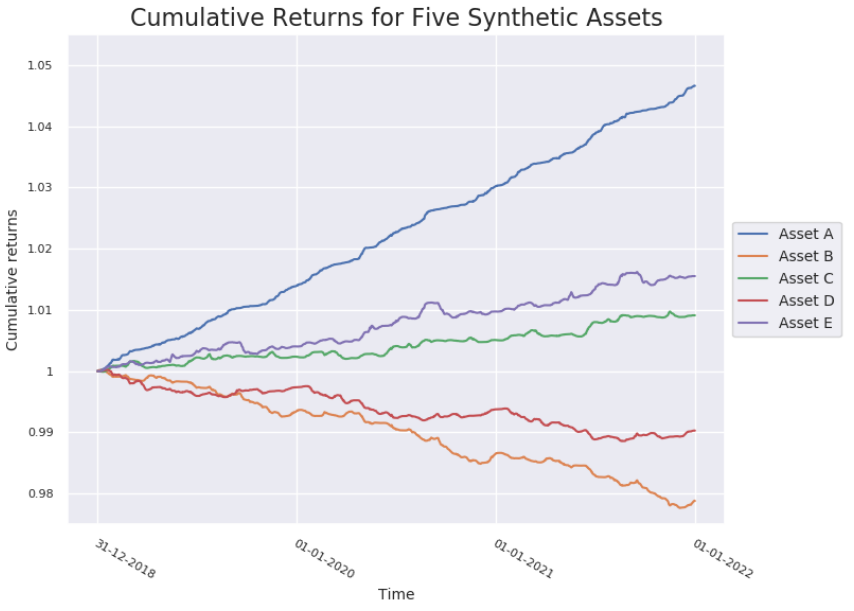

Figure 10. Cumulative returns of five synthetic assets over the three years from 1 January 2019 until 1 January 2022.

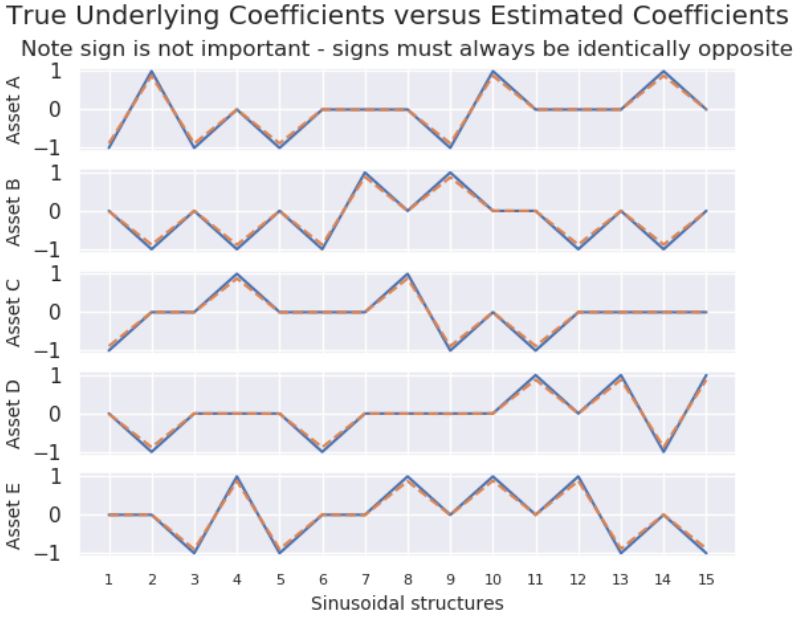

Figure 11. Direct estimation of coefficients (dashed orange line) using covariance regression overlaid upon the true underlying coefficients (solid blue line).

12.3 Lasso Estimation

As opposed to Section 12.2, the estimates are now derived using LASSO RCR as outlined in Section 5.4 in "Package CovRegpy: Regularised Covariance Regression and Forecasting in Python". This code is also available in `CovRegpy_CASE_STUDY_LASSO_synthetic.py`. All other available RCR methods can be applied to arrive at a solution, but it is advised to not use and critique group-LASSO RCR as there is no grouping of the coefficients in this synthetic data case study.

```
B_est, Psi_est = \
    cov_reg_given_mean(A_est=np.zeros_like(coef),
                       basis=spline_basis_transform, x=x[:, :-1],
                       y=returns_subset.T, iterations=10,
                       technique='lasso', alpha=1e-8)
```

In Figure 12 it is observable that all the coefficients are estimated to be identically zero for the first month. The penalty is relatively minute, but as one can note by running `CovRegpy_CASE_STUDY_LASSO_synthetic.py`, it results in erroneous coefficients half of the time over the first year. From this solution, it can be inferred that no attributable covariance is present. It can be observed in Figure 13 that as a result of the attributable covariance being estimated to be not present, the unattributable covariance ($\mathbf{\Psi}$) is estimated to be far larger (several orders of magnitude) than the base covariance as can be seen in Figure 8.

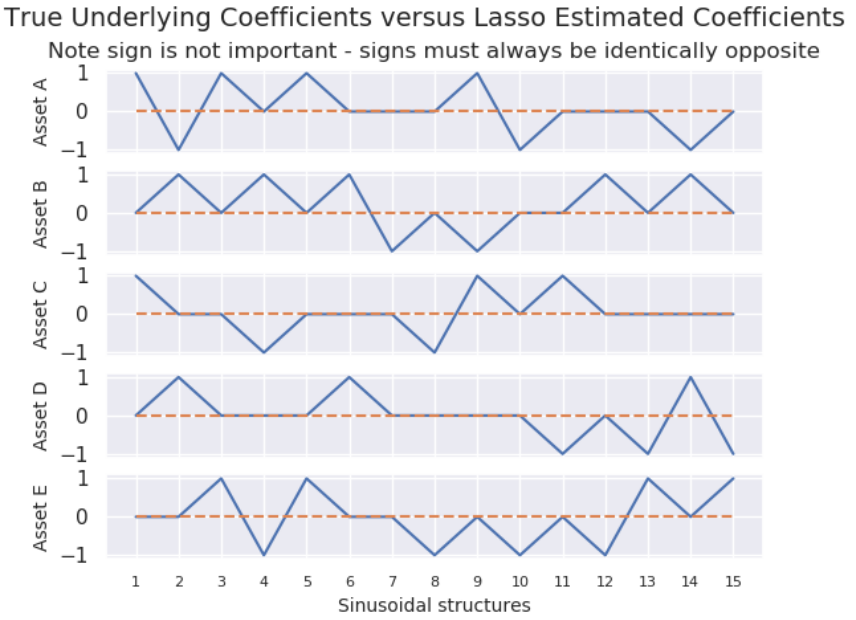

Figure 12. LASSO estimation of coefficients (dashed orange line) using covariance regression overlaid upon the true underlying coefficients (solid blue line) showing identically zero estimates.

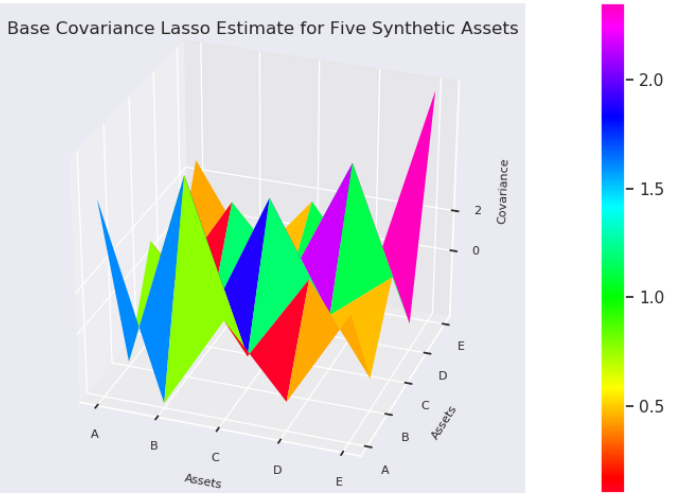

Figure 13. Lasso estimation of base unattributable covariance ($\boldsymbol{\Psi}$) using covariance regression - note large (several orders of magnitude) difference between this figure and Figure 8 owing to apparent lack of attributable covariance.

In the following month the results are far more pleasing as can be seen in Figure 14. This inconsistency in the solutions derived using LASSO RCR can be attributed to instantaneous Cholesky decomposition and correlating of returns based on dynamic synthetic covariates.

In Figure 15 the base unattributable covariance ($\hat{\boldsymbol{\Psi}}$) is much closer to the true underlying base unattributable covariance ($\boldsymbol{\Psi}$) as in Figure 8. Small deviations from both the true coefficients in Figure 14 and the true underlying unattributable covariance ($\boldsymbol{\Psi}$), as can be seen by comparing Figure 8 and Figure 15, can be attributed, again, to the random error formulation of covariance regression and the instantaneous Cholesky decomposition and correlating. This is not a significant error when the relative magnitude of the attributable covariance is taken into account.

13. Supplemental Case Study 2: Explicit Forecasting

An amplitude and frequency-modulated sinusoid is synthesized to replicate common IMF structures that are isolated in EMD. With:

$$t \in \left\{0, \frac{1}{200}\pi, \frac{2}{200}\pi, \ldots, 5\pi\right\}, \tag{31}$$

the following is plotted in Figure 16:

$$g_t = a_t \sin\left(f_t\right), \tag{32}$$

with

$$a_t = \frac{t + \frac{5}{4}\pi}{\frac{5}{4}\pi}, \tag{33}$$

and

$$f_t = t + \frac{t^2}{10\pi}. \tag{34}$$

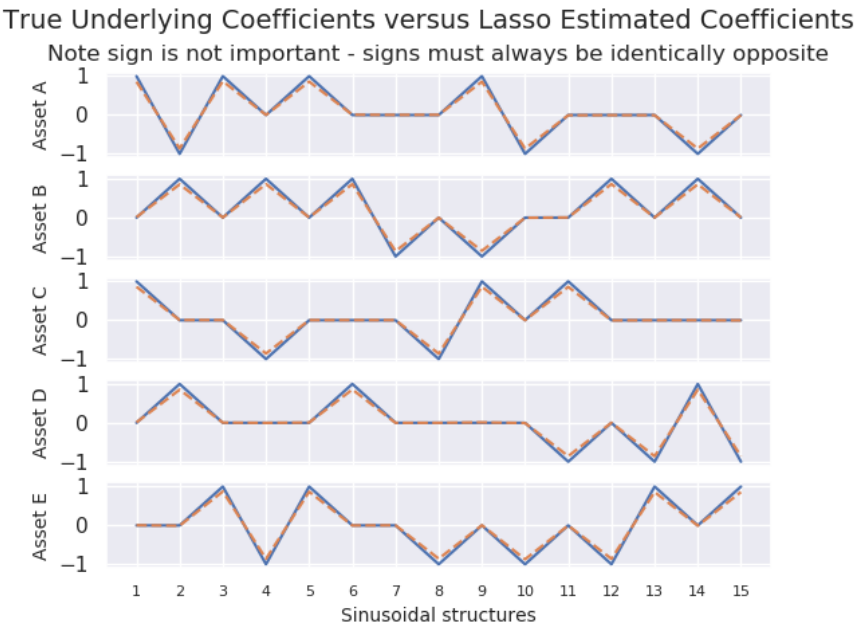

Figure 14. LASSO estimation of coefficients (dashed orange line) using covariance regression overlaid upon the true underlying coefficients (solid blue line).

13.1 Forecasting Approaches

Broadly speaking, there are two ways to address the problem of forecasting an independent variable given a dependent variable. One can implicitly forecast the dependent variable by simply regressing a lagged version of the independent variable. This results in a remaining interval of independent variables that can be used to forecast the dependent variables. This approach is briefly discussed in Section 13.1.1. An alternative, that requires additional assumptions, would be to regress the entirety of the independent variables against the dependent variables after which one would have to explicitly forecast or extrapolate the independent variables to then estimate the dependent variables over the same interval. Several techniques for doing this are discussed in Sections 13.1.2, 13.1.3, and 13.1.4.

13.1.1 Implicit Forecasting using a Lagged Effect Assumption

In Figure 17 implicit forecasting is demonstrated. Rather than making restrictive assumptions to explicitly forecast the independent variable which may then be used to predict the dependent variable, this model assumes that there is a delayed causative relationship between the variables. This also, with the ever-increasingly restrictive regulation of banking and related sectors, requires fewer assumptions and therefore is more defensible.

13.1.2 Multivariate Regression or Unactivated Single Neuron Neural Network Forecasting

While this is referred to as the Neural Network Edge Effect in Deng et al. (2001) and van Jaarsveldt et al. (2023), in the absence of an activation function (or with an identity activation function) it is merely a multivariate regression. There are several ways to estimate the weights, but owing to the smooth nature of the IMFs or implicit factors, ridge regression is recommended. The objective function is:

$$\min_{\mathbf{w}} ||\mathbf{wP} - \mathbf{y}||_2^2 + \alpha||\mathbf{w}||_2^2, \tag{35}$$

with

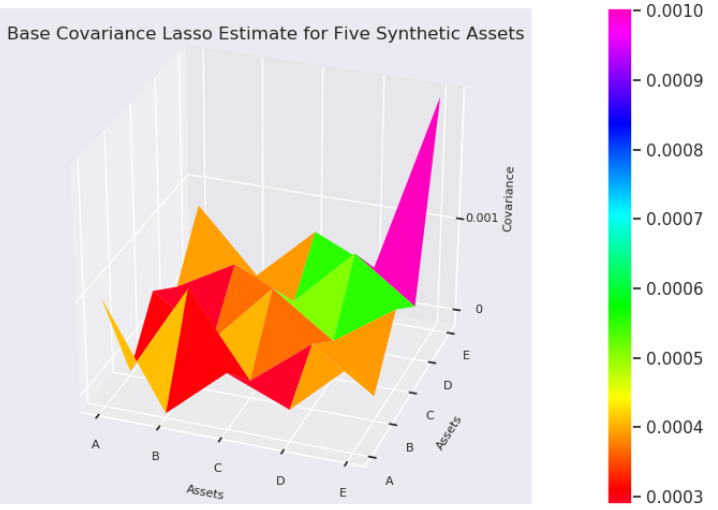

Figure 15. Lasso estimation of base unattributable covariance ($\hat{\mathbf{\Psi}}$) using covariance regression demonstrating the accuracy of unattributable estimate when attributable correct ascertained.

$$\mathbf{w} = \begin{bmatrix} w_1 \; w_2 \; \dots \; w_k \end{bmatrix}, \tag{36}$$

and

$$\mathbf{P} = \begin{bmatrix} y_{T-(m+k)} & y_{T-(m+k-1)} & \cdots & y_{T-(k+1)} \\ y_{T-(m+k-1)} & y_{T-(m+k-2)} & \cdots & y_{T-k} \\ \vdots & \vdots & \ddots & \vdots \\ y_{T-(m+1)} & y_{T-m} & \cdots & y_{T-2} \end{bmatrix}, \tag{37}$$

and

$$\mathbf{y} = \begin{bmatrix} y_{T-m} \; y_{T-(m-1)} \; \cdots \; y_{T-1} \end{bmatrix}. \tag{38}$$

In this setting and for consistency with the code, the $k =$ `no_sample` and $m =$ `fit_window`. The `no_sample` is recommended to encompass at least one wavelength of the structure to accurately forecast the structure. The implementable code can be seen below and can be found in `CovRegpy_CASE_STUDY_forecasting.py`:

```
neural_network_forecast = CovRegpy_neural_network(time_series,
                                                  no_sample=300,
                                                  fit_window=200,
                                                  alpha=1.0)
```

13.1.3 Gaussian Processes used in Forecasting

Gaussian processes can be used to forecast the IMF structure. The difficulty lies in the flexibility of the structures which need to be forecast. The recommended kernel should utilize the periodic kernel named the Exponentiated Sine Squared kernel in Equation (39).

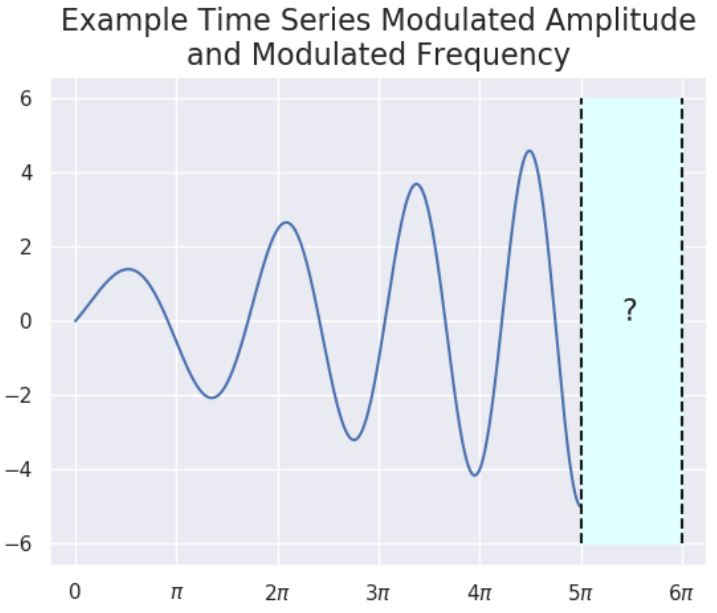

Figure 16. Discrete synthetic modulated frequency and amplitude example time series to which the explicit forecasting techniques in this section are applied.

$$k_{\text{ESS}}(t_1,t_2)=\sigma^2\text{exp}\left(-\frac{2\cdot\sin^2\left(\frac{\pi|t_1-t_2|}{p}\right)}{l^2}\right). \tag{39}$$

In addition to the periodic kernel, owing to the flexibility of the potential structures, one should utilise compound kernels such as those described in Equation (40) and Equation (41) below:

$$k_{\text{comp}_1}(t_1,t_2)=\sigma^2\text{exp}\left(-\frac{(t_1-t_2)^2}{2l_1^2}\right)\text{exp}\left(-\frac{2\cdot\sin^2\left(\frac{\pi|t_1-t_2|}{p}\right)}{l_2^2}\right), \tag{40}$$

and

$$k_{\text{comp}_2}(t_1,t_2)=\sigma^2\left(1+\frac{(t_1-t_2)^2}{2\alpha l_1^2}\right)^{-\alpha}\text{exp}\left(-\frac{2\cdot\sin^2\left(\frac{\pi|t_1-t_2|}{p}\right)}{l_2^2}\right). \tag{41}$$

The compound kernels in Equation (40) and Equation (41) utilise the Squared Exponential Kernel or the Radial Basis Kernel (RBK):

$$k_{RBF}(t_1,t_2)=\sigma^2\text{exp}\left(-\frac{(t_1-t_2)^2}{2l^2}\right), \tag{42}$$

and the Rational Quadratic Kernel (RQK):

$$k_{RQ}(t_1,t_2)=\sigma^2\left(1+\frac{(t_1-t_2)^2}{2\alpha l^2}\right)^{-\alpha}, \tag{43}$$

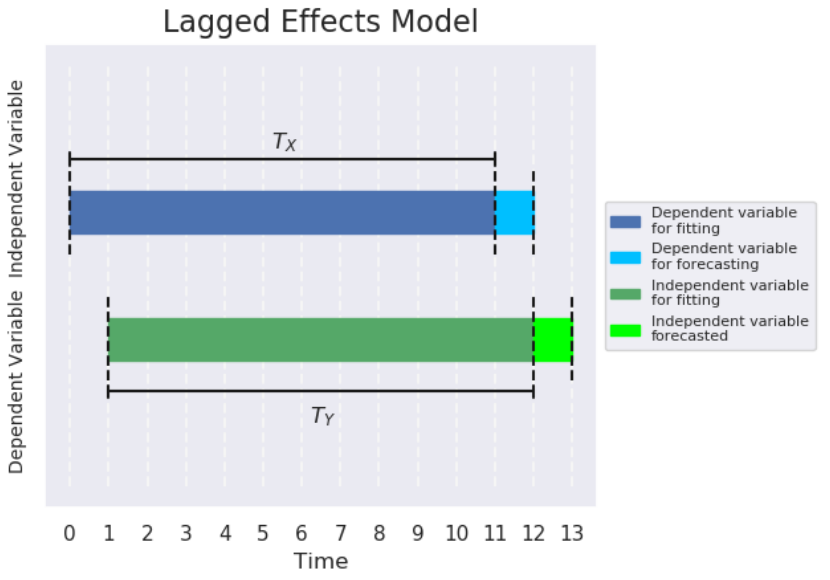

Figure 17. Plot demonstrating a simple example of a lagged effects model constructed at time point 12 in which the one time period delay is used for implicit forecasting.

respectively. These decaying functions allow more flexibility for locally periodic structures that do not exactly repeat themselves as is mostly observed in the real world. With $\sigma = 1$, $t_2(\text{or } t_1) = 5$, $l_1 = 1$, $p = 1$, $l_2 = 1$, and $\alpha = 1$, Equation (40) and Equation (41) are plotted over $t_1(\text{or } t_2) \in [0, 10]$ in Figure 19. To replicate Figure 20 one can apply the code below by reconstructing the example as described in this section or one can run the script titled `CovRegpy_CASE_STUDY_forecasting.py`.

```
y_forecast, sigma, y_forecast_upper, y_forecast_lower = \
    gp_forecast(time, time_series, time_extended,
                kernel, 0.95, plot=False)
```

13.1.4 Instantaneous Frequency Forecasting

In Fourier and Darboux (1822), it was shown that periodic functions can be written as an infinite sum of harmonics. This implies that functions can be transformed into the frequency space. In this space, the structures may be less complex than in the temporal space leading to easier forecasting. The Hilbert transform of the synthetic time series in Figure 16 can be seen in Figure 21. This structure has been slightly perturbed by the sifting algorithm, but the linearly modulated frequency is visible with the true linear structure overlaid.

The instantaneous amplitude is not as discernible in the Hilbert transform in Figure 21 as the instantaneous frequency. Independently plotting the instantaneous amplitude results in Figure 22. The linear modulation, or any modulation of the amplitude for that matter, is less obscured by inaccuracies in the sifting process owing to the lack of an inverse transform when compared against the instantaneous frequency calculation. The resulting instability in the instantaneous frequency when compared against the relative instability of the instantaneous amplitude can be observed by comparing Figure 22 and Figure 23.

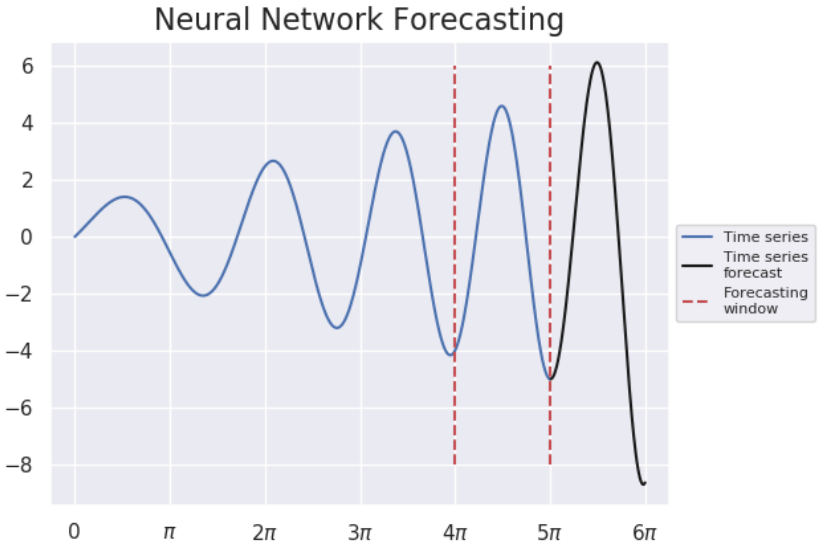

Figure 18. Plot of the multivariate regression forecasting example explained in Equation (35) and applied to Equation (32) to extrapolate 200 time points.

14. Supplemental Case Study 3: Real-World Equity Case Study

In this real-world example entitled `CovRegpy_CASE_STUDY_appendix.py`, available at this link:

https://github.com/Cole-vJ/CovRegpy,

the script utilizes a sister package called "AdvEMDpy", which can be found here:

https://github.com/Cole-vJ/AdvEMDpy.

The purpose of the script is to isolate different frequency components from the daily closing stock prices of AAPL, AMZN, GOOGL, MSFT, and TSLA, and forecast the next day's covariance structure using RCR (Regularized Covariance Regression). The raw equity data is decomposed using EMD, although the reader may also attempt the decomposition using the implicit factor models described within this work for comparison. The following code is present in the script:

```
emd = AdvEMDpy.EMD(time_series=np.asarray(all_data.iloc[:, j]),
                   time=time[lag:int(model_days + lag + 1)])
imfs, _, _, _, _, _, _ = \
    emd.empirical_mode_decomposition(knot_envelope=
        np.linspace(time[lag:int(model_days + lag + 1)][0],
                    time[lag:int(model_days + lag + 1)][-1],
                    knots), matrix=True)
```

This code instantiates an EMD object with the daily equity data of the jth asset before decomposing it. The decomposed data is stored in the matrix x. Once the equity data for each asset has been decomposed and stored in matrix x, the RCR model can be fitted using the following script. It should be noted that the script uses ridge regression RCR or L2-RCR, which is developed in the "CovRegpy" package. For a better understanding of the coefficients required for mean isolation

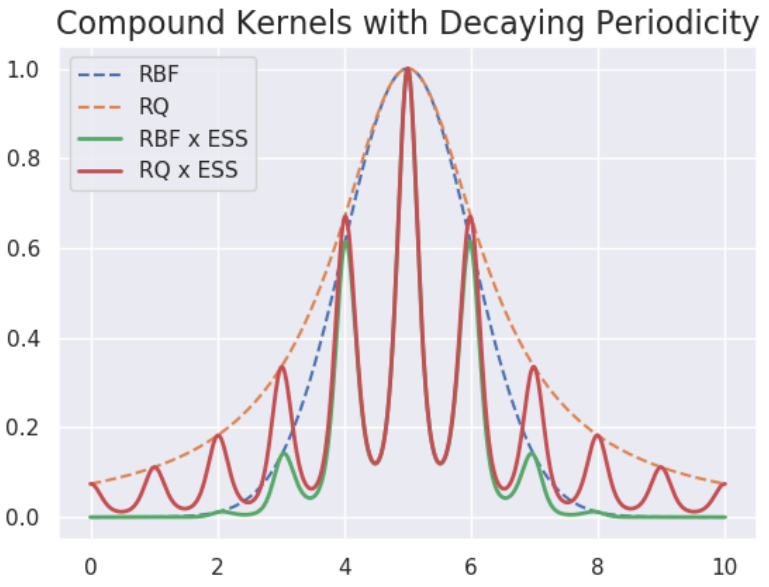

Figure 19. Compound kernels allowing decaying periodicity using Radial Basis Function kernel and Rational Quadratic kernel.

(`coef`) and the basis required with the associated coefficients (`spline_basis_transform`), the reader is encouraged to experiment with the scripts contained within the case study folder.

```
B_est, Psi_est = \
    cov_reg_given_mean(A_est=coef, basis=spline_basis_transform,
                       x=x[:, :-1], y=realised_returns.T,
                       iterations=10, technique='ridge',
                       max_iter=500)
```

Furthermore, with the forecasted covariance structure, one can calculate the risk premia parity weighting as described above and plot the resulting returns and risk. This can be compared against various common benchmarks such as the stocks themselves, the equally weighted portfolio, the principal component portfolio, the global minimum variance portfolio, the maximum Sharpe ratio, and the resulting efficient frontier as in Figure 26.

```
weights_Model = \
    equal_risk_parity_weights_summation_restriction(variance_Model).x
model_variance = global_obj_fun(weights_Model,
                                variance_Model)
model_returns = sum(weights_Model * np.mean(realised_returns,
                                            axis=0) * 365)
```

This example provides a brief overview and utilizes data from Yahoo! Finance (2021) for users to explore and consider potential model extensions, as well as the final analysis.

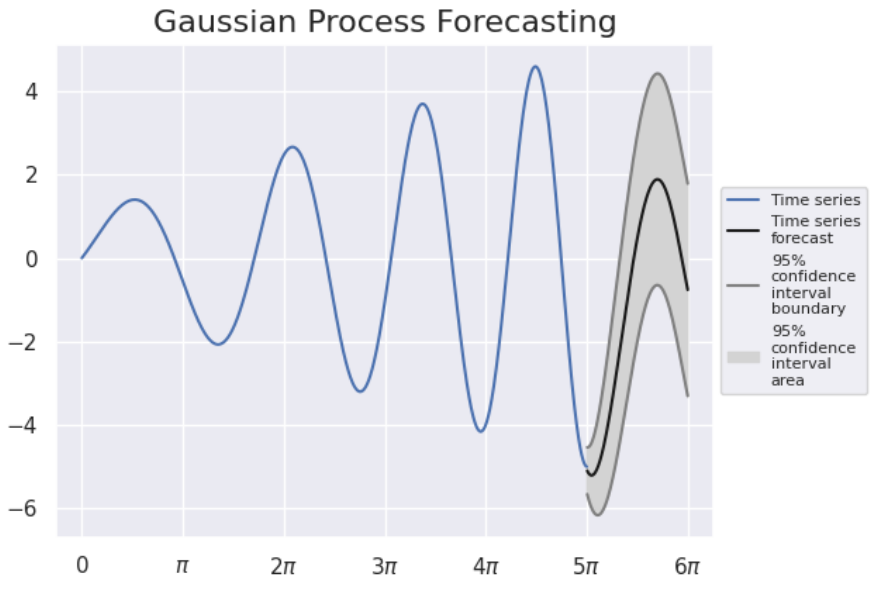

Figure 20. Gaussian process forecasting with confidence intervals - the outputs from the above code `y_forecast`, `y_forecast_upper`, `y_forecast_lower` being plotted with `sigma` being the uncertainty level.

15. Appendices

In these appendices, helpful pseudo-code is presented for the sifting EMD algorithm and the sifting X11 algorithm. For further reading on the EMD method, its extensions, and algorithmic variations see van Jaarsveldt et al. (2023) and van Jaarsveldt et al. (2021). For the Python code and package dedicated to EMD and its algorithmic variations see:

`https://github.com/Cole-vJ/AdvEMDpy`.

The two core features (for discrete time series) that an intrinsic mode function (IMF) must satisfy are repeated here in C1 and C2 for easy reference in Algorithm 2. An additional condition that prevents over-sifting is presented in C3. A detailed analysis as well as recommendations for those interested can be found in van Jaarsveldt et al. (2023).

The pseudo-code for the original X11 technique proposed in Shiskin et al. (1967) can be found in Algorithm 3. The code can be found in this package in the script titled `CovRegpy_X11.py` with accompanying notes detailing the asymmetric Henderson weighting conundrum.

Appendix A. EMD Sifting Algorithm

As noted above, these conditions are repeated here for ease of reference when reading and translating the pseudo-code in Algorithm 2. The conditions are:

$$\text{C1 } \operatorname{abs}\left(\left|\left\{\frac{d\gamma_k(t)}{dt}=0 : t\in(0,T)\right\}\right| - \left|\left\{\gamma_k(t)=0 : t\in(0,T)\right\}\right|\right) \leq 1,$$

$$\text{C2 } \sum_t \operatorname{abs}\left(\tilde{\gamma}_k^{\mu}(t)\right) \leq \epsilon_2, \text{ and}$$

$$\text{C3 } SD_{(p,q)} = \sum_{t=t_0}^{t_N} \left[\frac{\left|\left(h_{(p,q-1)}(t) - h_{(p,q)}(t)\right)\right|^2}{h_{(p,q-1)}^2(t)}\right] < \epsilon_3.$$

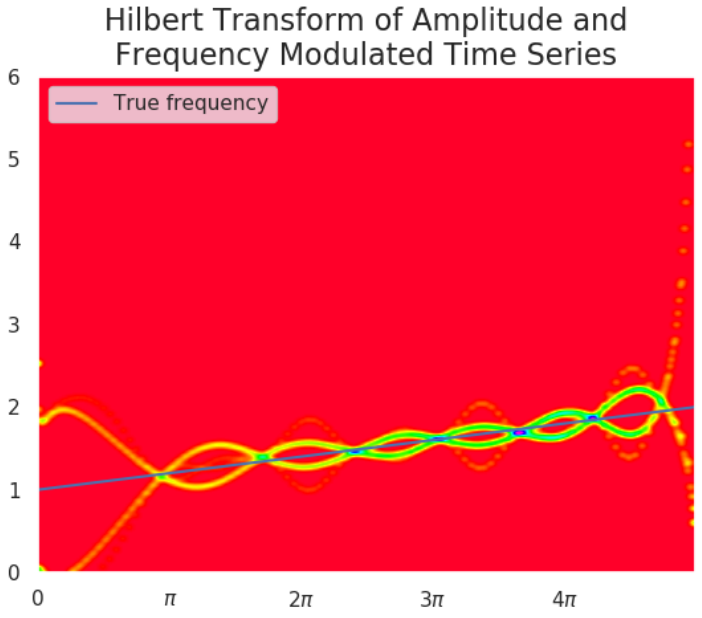

Figure 21. Hilbert transform of amplitude and frequency modulated time series with true underlying instantaneous frequency overlaid and increasing instantaneous amplitude indicated with increasingly intense blue spectrum.

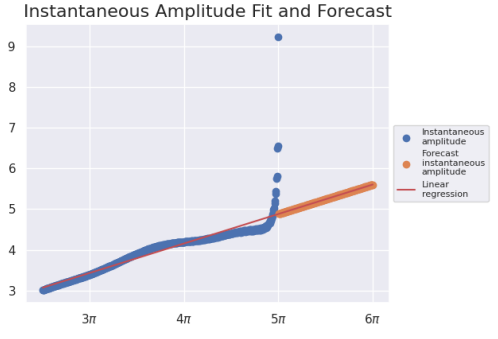

Figure 22. Instantaneous amplitude linear regression extrapolation.

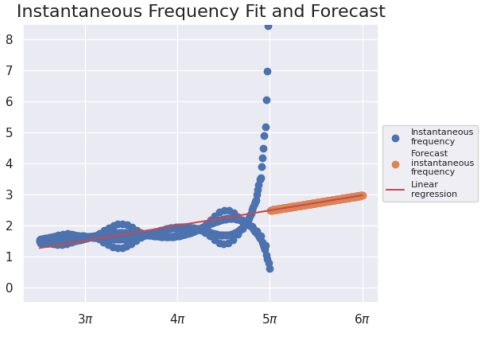

Figure 23. Instantaneous frequency linear regression extrapolation.

with ϵ_2 and ϵ_2 being adjustable parameters in the algorithm. The most robust version of C3 combined four conditions, but this is not the focus of this work, see van Jaarsveldt et al. (2023).

Appendix B. X11 Sifting Algorithm

In the code in Section 3.3 of 'Package CovRegpy: Regularised Covariance Regression and Forecasting in Python', `trend_window_width_1=13` corresponds to a in $\text{ma}_a(\cdot)$ in Algorithm 3, `trend_window_width_2=13` corresponds to b in $\text{Hma}_b(\cdot)$, and `trend_window_width_3=13` corresponds to c in $\text{Hma}_c(\cdot)$.

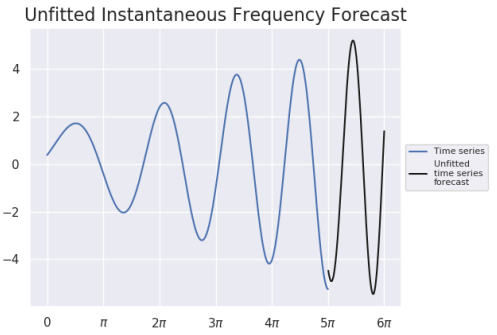

Figure 24. Instantaneous frequency forecasting without fitting.

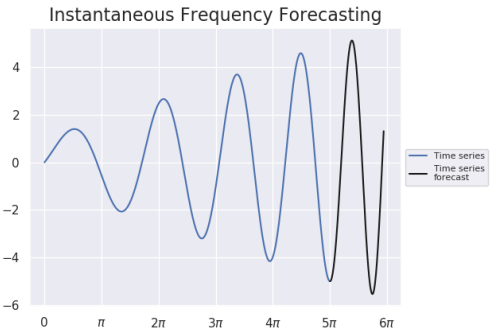

Figure 25. Instantaneous frequency forecasting with phase fitted.

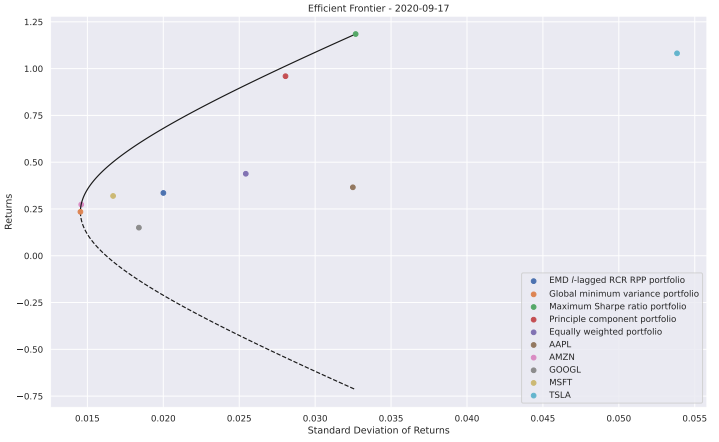

Figure 26. Efficient frontier formed by AAPL, AMZN, GOOGL, MSFT, and TSLA stock indices from 15 October 2018 until 16 September 2020.

Algorithm 2 Sifting Process

Require: $x(t) = h_{(1,0)}(t) = r_1(t)$
  Initialize:
  (1) $M(t_i)$ of $h_{(1,0)}(t)$
  (2) $m(t_j)$ of $h_{(1,0)}(t)$
  while $|\{M(t_i)\}| + |\{m(t_j)\}| > 2$ **do**
    fit $\tilde{h}^M(t)$ through $M(t_i)$
    fit $\tilde{h}^m(t)$ through $m(t_j)$
    calculate $\tilde{h}^\mu(t) = \dfrac{\tilde{h}^M(t) + \tilde{h}^m(t)}{2}$
    if C1 and C2 or C3 **then**
      store $\gamma_p(t) = h_{(p,q)}(t)$
      calculate $r_{p+1}(t) = r_p(t) - \gamma_p(t)$
      set $h_{(p+1,0)}(t) = r_{p+1}(t)$
      find $M(t_i)$ of $h_{(p+1,0)}(t)$
      find $m(t_j)$ of $h_{(p+1,0)}(t)$
    else
      calculate $h_{(p,q+1)}(t) = h_{(p,q)}(t) - \tilde{h}^\mu(t)$
      find $M(t_i)$ of $h_{(p,q+1)}(t)$
      find $m(t_j)$ of $h_{(p,q+1)}(t)$
    end if
  end while
  store $r_K(t)$

Algorithm 3 X11

Require: $x(t), k, l, m, n$
  $T_1^*(t) = \text{ma}_a(x(t))$
  $S_1^*(t) = x(t) - T_1^*(t)$
  $S_2^*(t) = S_{m\times n}(S_1^*(t))$
  $T_2^*(t) = x(t) - S_2^*(t)$
  $T_3^*(t) = \text{Hma}_b(T_2^*(t))$
  $S_3^*(t) = x(t) - T_3^*(t)$
  $S(t) = S_{m\times n}(S_3^*(t))$
  $T_4^*(t) = x(t) - S(t)$
  $T(t) = \text{Hma}_c(T_4^*(t))$
  $\epsilon(t) = T(t) - T_4^*(t)$

References

L. Bauwens, S. Laurent, and J. Rombouts. 2006. Multivariate GARCH Models: A Survey. *Journal of Applied Econometrics* 21, 1 (2006), 79–109. https://doi.org/10.1002/jae.842

C. Benoit. 1924. Note Sur Une Méthode de Résolution des équations Normales Provenant de L'Application de la Méthode des Moindres Carrés à un Système D'équations Linéaires en Nombre Inférieur à Celui des Inconnues. - Application de la Méthode a la Résolution D'un Systéme Defini D'équations. *Bulletin Géodésique* 2, 1 (1924), 67–77. https://doi.org/10.1007/BF03031308

T. Bollerslev, R. Engle, and J. Wooldridge. 1988. A Capital Asset Pricing Model with Time-Varying Covariances. *Journal of Political Economy* 96, 1 (1988), 116–131. https://doi.org/10.1086/261527

P. Bonizzi, J. Karel, O. Meste, and R. Peeters. 2014. Singular Spectrum Decomposition: A New Method for Time Series Decomposition. *Advances in Adaptive Data Analysis* 6, 4 (2014), (1450011) 1–34. https://doi.org/10.1142/S1793536914500113

Y. Deng, W. Wang, C. Qian, Z. Wang, and D. Dai. 2001. Boundary-processing-technique in EMD method and Hilbert transform. *Chinese Science Bulletin* 46, 1 (2001), 954–960. https://doi.org/10.1007/BF02900475

R. Engle. 2002. Dynamic Conditional Correlation: A Simple Class of Multivariate Generalized Autoregressive Conditional Heteroskedasticity Models. *Journal of Business & Economic Statistics* 20, 3 (2002), 339–350. https://doi.org/10.1198/073500102288618487

E. Fama and K. French. 1993. Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics* 33, 1 (1993), 3–56.

E. Fama and K. French. 2015. A five-factor asset pricing model. *Journal of Financial Economics* 116, 1 (2015), 1–22. https://doi.org/10.1016/j.jfineco.2014.10.010

J. Fourier and G. Darboux. 1822. *Théorie analytique de la chaleur*. A Paris, Chez Firmin Didot, Pére Et Fils, Paris.

H. Hassani. 2007. Singular Spectrum Analysis: Methodology and Comparison. *Journal of Data Science* 5, 2 (2007), 239–257. https://doi.org/10.6339/JDS.2007.05(2).396

J. Shiskin, A. Young, and J. Musgrave. 1967. *The X-11 Variant of the Census Method II Seasonal Adjustment Program*. Technical Report 15. U.S. Department of Commerce, Washington D.C.

C. van Jaarsveldt, M. Ames, G. Peters, and M. Chantler. 2023. Package AdvEMDpy: Algorithmic variations of empirical mode decomposition in Python. *Annals of Actuarial Science* (2023), 1–37. https://doi.org/10.1017/S1748499523000088

C. van Jaarsveldt, G. Peters, M. Ames, and M. Chantler. 2021. Tutorial on Empirical Mode Decomposition: Basis Decomposition and Frequency Adaptive Graduation in Non-Stationary Time Series. *Available at SSRN 3913330* (2021). https://doi.org/10.2139/ssrn.3913330

Yahoo! Finance. 2021. https://uk.finance.yahoo.com/. Online; accessed 22 February 2021.

Cite this article: C. van Jaarsveldt, G. Peters, M. Ames, and M. Chantler. Supplement to: Package **CovRegpy**: Regularised Covariance Regression and Forecasting in Python. *Annals of Actuarial Science*, 1–28. 10.1017/xxxxx