Санкт-Петербургский государственный университет

Прикладная математика и информатика

Отчет по учебной практике 2 (проектно-технологической) (семестр 3)

Использование метода SSA для анализа временных рядов на языке Python

Выполнил:

Козак Михаил Валерьевич

группа 21.М03-мм

Научный руководитель:

к. ф.-м. н., доцент

Голяндина Нина Эдуардовна

Кафедра Статистического Моделирования

Санкт-Петербург

2022

Оглавление

Глава 1

Введение

Singular spectrum analysis (SSA) [1] – метод для анализа временных рядов, основанный на преобразовании одномерного временного ряда в многомерный, с последующим применением к нему метода главных компонент. Данный метод находит применение в различных сферах: климатология, океанология, геофизика, техника, обработка изображений, медицина, эконометрика и многие другие.

Для использования SSA не требуется предварительное задание модели для анализа различных произвольных рядов. Основная цель данного метода заключается в разложении ряда в сумму интерпретируемых компонент: тренд, преиодические компоненты, шум. Полученное разложение может служить основой для прогнозирования дальнейших изменений как самого ряда, так и его отдельных составляющих.

В рамках данной работы были рассмотрены как написанный для языка R пакет Rssa [2, 3], так и уже существующие реализации данного метода для языка Python [4, 5, 6], недостаток которых заключается в том, что они не обладают большой скоростью в связи с тем, что в их реализации не используются быстрые алгоритмы, которые, помимо этого, в пакете Rssa реализованы с использованием кода на языке C. По этой причине было принято решение реализовать данный метод посредством адаптации существующего модуля к языку Python.

Помимо адаптации модуля Rssa к языку Python также возникла необходимость в визуализации результатов, в связи с чем потребовалось прибегнуть к сторонним модулям для отображения графиков в среде Python.

Также одной из задач является применение полученных результатов к реальным данным: разложение временного ряда на его отдельные составляющие (шум, тренд, период) и визуализация полученного разложения.

В рамках текущей работы были выполнены задачи, связанные с портированием для пакета pyrssa тестов, написанных для проверки работы функциональности модуля Rssa. Помимо этого, были расширены существующие возможности конвертации типов данных между R и Python, добавлены новые функции, связанные с алгоритмами для достижения лучшей разделимости компонент разложения и их автоматической группировкой, добавлен дополнительный метод прогнозирования и расширены возможности

уже добавленных функций, проведено обновление и дополнение структур создаваемых объектов и методов визуализации с целью упрощения работы для пользователей.

Также, помимо работы непосредственно над улучшением модуля, на основе полученных результатов с помощью пакета pyrssa проведен анализ и прогноз временного ряда параметров вращения Земли.

Материал, относящийся к новым результатам, описан в разделах 3.1, 3.2, 3.3, 5.

Глава 2

Метод SSA

2.1. Алгоритм

Рассмотрим временной ряд длины $N > 2$: $F = (f_1, \ldots, f_N), f_i \in \mathbb{R}$. Будем также предполагать, что ряд ненулевой: $\exists i : f_i \neq 0$. Помимо этого, числа $1, \ldots, N1$ можно интерпретировать не только как дискретные моменты времени, но и как некоторые метки, которые имеют линейно-упорядоченную структуру.

Базовый алгоритм состоит из двух дополняющих друг друга этапов: разложения и восстановления.

Разложение. Пусть L – некоторое целое число *(длина окна)*, $1 < L < N$. Сперва необходимо провести процедуру вложения, которая переводит исходный временной ряд в последовательность многомерных векторов. Данная процедура образует $K = N - L + 1$ *векторов вложения*:

$$X_i = (f_i, \ldots, f_{i+L-1})^{\mathrm{T}}, 1 \leq i \leq K,$$

имеющих размерность L.

L–траекторная матрица ряда F:

$$\mathbf{X} = [X_1 : \cdots : X_K]$$

состоит из векторов вложения в качестве столбцов.

Следующим шагом является сингулярное разложение траекторной матрицы временного ряда.

Пусть $\mathbf{S} = \mathbf{X}\mathbf{X}^{\mathrm{T}}$. Обозначим $\lambda_1, \ldots, \lambda_L$ *собственные числа* матрицы $\mathbf{S}$, взятые в неубывающем порядке: $\lambda_1 \geq \cdots \geq \lambda_L \geq 0$ и $U_1, \ldots, U_L$ – ортонормированную систему собственных векторов матрицы $\mathbf{S}$, соответствующих собственным числам.

Пусть $d = \max\{i : \lambda_i > 0\}$. Если обозначить $V_i = \frac{X^{\mathrm{T}} U_i}{\sqrt{\lambda_i}}, i = 1, \ldots, d$, то сингулярное разложение матрицы $\mathbf{X}$ может быть записано в следующем виде:

$$\mathbf{X} = \mathbf{X}_1 + \cdots + \mathbf{X}_d,$$

где $\mathbf{X}_i = \sqrt{\lambda_i} U_i V_i^{\mathrm{T}}$.

Восстановление. На основе сингулярного разложения производится процедура группировки, которая делит все множество индексов $\{1, \ldots, d\}$ на m непересекающихся подмножеств $I_1, \ldots, I_m$.

Пусть $I = \{i_1, \ldots, i_p\}$. Тогда *результирующая матрица* X_I, определяется как

$$\mathbf{X}_I = \mathbf{X}_{i_1} + \cdots + \mathbf{X}_{i_p}.$$

Такие матрицы вычисляются для $I = I_1, \ldots, I_m$, и разложение может быть записано в сгруппированном виде

$$\mathbf{X} = \mathbf{X}_{I_1} + \cdots + \mathbf{X}_{I_m}.$$

На последнем шаге алгоритма каждая матрица сгруппированного разложения переводится в новый ряд длины N.

Пусть $\mathbf{Y} \in \mathbb{R}^{L \times K}$ – некоторая матрица с элементами y_{ij}. Положим $L^* = min(L, K)$, $K^* = max(L, K)$ и $N = L + K - 1$. Пусть $y^*_{ij} = y_{ij}$, если $L < K$, и $y^*_{ij} = y_{ji}$ иначе. *Диагональное усреднение* переводит матрицу $\mathbf{Y}$ в ряд $g_1, \ldots, g_N$ по формуле:

$$g_k = \begin{cases} \frac{1}{k+1} \sum\limits_{m=1}^{k+1} y^*_{m,k-m+2} & \text{для } 1 \le k < L^*, \\ \frac{1}{L^*} \sum\limits_{m=1}^{L^*} y^*_{m,k-m+2} & \text{для } L^* \le k < K^*, \\ \frac{1}{N-k} \sum\limits_{m=k-K^*+2}^{N-K^*+1} y^*_{m,k-m+2} & \text{для } K^* \le k < N. \end{cases}.$$

Применяя диагональное усреднение к результирующим матрицам $\mathbf{X}_{I_k}$, получаем ряды $\widetilde{F}^{(k)} = (\widetilde{f}^{(k)}_1, \ldots, \widetilde{f}^{(k)}_N)$, и, следовательно, исходный ряд $(f_0, \ldots, f_{N-1})$ раскладывается в сумму m рядов:

$$f_n = \sum_{k=1}^{m} \widetilde{f}^{(k)}_n.$$

2.2. Прогнозирование

.

Рекуррентное прогнозирование. Пусть I – это некоторая группа собственных троек $P_i \in \mathsf{R}^L$, $i \in I$ – соответствующие собственные векторы, $\underline{P_i}$ – их первые $L - 1$

координат, и π_i – последняя координата P_i, $\nu^2 = \sum_i \pi_i^2$. Определим $R = (a_{L-1}, \ldots, a_1)^{\mathrm{T}}$ как

$$R = \frac{1}{1-\nu^2} \sum_{i \in I} \pi_i \underline{P_i}. \tag{2.1}$$

Алгоритм рекуррентного прогнозирования состоит из следующих действий

1. ряд $\mathbb{Y}_{N+M} = (y_1, \ldots, y_{N+M})$ определяется как

$$y_i = \begin{cases} \widetilde{x}_i & i = 1, \ldots, N, \\ \sum\limits_{j=1}^{L-1} a_j y_{i-j} & i = N+1, \ldots, N+M. \end{cases} \tag{2.2}$$

2. числа $y_{N+1}, \ldots, y_{N+M}$ являются результатом прогнозирования M точек ряда.

Таким образом, рекуррентный прогноз выполняется непосредственным применением ЛРФ с коэффициентами $\{a_j, j = 1, \ldots, L-1\}$.

Возможен иной подход. Введём оператор $\mathcal{P}_{\mathrm{Rec}} : \mathsf{R}^L \mapsto \mathsf{R}^L$ по следующей формуле:

$$\mathcal{P}_{\mathrm{Rec}} Y = \begin{pmatrix} \overline{Y} \\ R^{\mathrm{T}} \overline{Y} \end{pmatrix}. \tag{2.3}$$

Обозначим

$$Z_i = \begin{cases} \widetilde{X}_i & i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Rec}} Z_{i-1} & i = K+1, \ldots, K+M. \end{cases} \tag{2.4}$$

Матрица $\mathbf{Z} = [Z_1 : \ldots : Z_{K+M}]$ является траекторной для ряда $\mathbb{Y}_{N+M}$, и тогда выражение (2.4) можно рассматривать как векторную форму (2.2).

Для получения прогноза, близкого к точным значениям ряда, важно правильным образом образовать группу I, отвечающую за сигнал, присутствующий в ряде. Следует принять во внимание тот факт, что результаты SVD-разложения траекторной матрицы ряда будут, вообще говоря, отличаться при добавлении новых значений к ряду.

Векторное прогнозирование. Обозначим $\mathcal{L}_r = \mathrm{span}(P_i, i \in I)$, $\widehat{X}_i$ – проекция вектора X_i на $\mathcal{L}_r$. Рассмотрим матрицу

$$\Pi = \underline{\mathbf{V}} \underline{\mathbf{V}}^{\mathrm{T}} + (1-\nu^2) R R^{\mathrm{T}}, \tag{2.5}$$

где $\underline{\mathbf{V}} = [\underline{P_1} : \ldots : \underline{P_r}]$ и R определено в (2.1).Матрица Π является матрицей линейного оператора, задающего ортогональную проекцию $\mathsf{R}^{L-1} \mapsto \underline{\mathcal{L}_r}$, where $\underline{\mathcal{L}_r} = \mathrm{span}(\underline{P_i}, i \in I)$. Определим линейный оператор $\mathcal{P}_{\mathrm{Vec}} : \mathsf{R}^L \mapsto \mathcal{L}_r$ по формуле

$$\mathcal{P}_{\mathrm{Vec}} Y = \begin{pmatrix} \Pi \overline{Y} \\ R^{\mathrm{T}} \overline{Y} \end{pmatrix}. \tag{2.6}$$

Алгоритм векторного прогноза.

1. В обозначениях, введенных выше, определим векторы Z_i следующим образом:

$$Z_i = \begin{cases} \widehat{X}_i & i = 1, \ldots, K, \\ \mathcal{P}_{\mathrm{Vec}} Z_{i-1} & i = K+1, \ldots, K+M+L-1. \end{cases} \tag{2.7}$$

2. Образовав матрицу $\mathbf{Z} = [Z_1 : \ldots : Z_{K+M+L-1}]$ и проведя ее диагональное усреднение, мы получим ряд $y_1, \ldots, y_{N+M+L-1}$.

3. Числа $y_{N+1}, \ldots, y_{N+M}$ образуют M членов векторного прогноза.

Глава 3

Пакет Rssa и его адаптация к Python

Реализация метода SSA входит в пакет Rssa, который содержит большое количество различных инструментов для анализа и прогнозирования временных рядов и обработки изображений, а также имеет множество визуальных инструментов, полезных при выборе параметров SSA и проверки результатов. Это единственная реализация метода SSA для языка R [7], доступная в CRAN, а также почти наверняка самая эффективная, во многом благодаря тому, что данный пакет реализован с использованием быстрых алгоритмов на языке C.

В связи с тем, что данный пакет уже обладает необходимой функциональностью и эффективностью, а также с учетом сложности переписывания данных алгоритмов под специфику языка Python [8], при использовании метода SSA для данного языка программирования было принято решение взять за основу существующий пакет, обращаясь к нему с помощью интерпретатора для языка R. Для решения поставленной задачи был выбран модуль rpy2 [9], позволяющий запускать код на языке R средствами Python и обладающий весьма гибкой функциональностью. Таким образом, задача состояла в том, чтобы адаптировать код на языке R незаметно от пользователя и с минимальным ущербом в скорости работы.

3.1. Конвертация типов данных

Данные языки во многом похожи, имеют схожие типы данных, но в то же время существуют достаточно серьезные отличия, среди которых, например, встроенная поддержка векторных вычислений в языке R, которой нет в языке Python. Для решения данной проблемы возникла необходимость в использовании пакета NumPy [10] для Python, который позволяет работать с многомерными массивами, а также имеет эффективно реализованные математические функции для работы с ними. В связи с тем, что зачастую анализ временных рядов ведется на основе таблиц данных, дополнительно потребовался популярный для этих целей пакет Pandas [11]. Благо, конвертация из NumPy и Pandas в R уже была реализована в модуле rpy2, однако, помимо этого возникла необходимость в том, чтобы связать встроенные типы данных Python с языком R.

Приведенный ниже программный код, отвечающий за правила конвертации, выделен в отдельный файл, импортируемый в основной части модуля.

Листинг 3.1. Конвертация из Python в R.

```
import rpy2.robjects.conversion as conversion
from rpy2.robjects.numpy2ri import converter as numpy_converter

pyrssa_converter = conversion.Converter('pyrssa converter')
pyrssa_converter.py2rpy.register(type(None), none_to_null)
pyrssa_converter.py2rpy.register(range, range_to_vec)
pyrssa_converter.py2rpy.register(list, list_to_vec)
pyrssa_converter.py2rpy.register(dict, dict_to_vec)
pyrssa_converter.py2rpy.register(SSA, pyrssa_to_rssa)

pyrssa_conversion_rules = default_converter + pyrssa_converter +\
                          numpy_converter + pandas_converter
```

Помимо данной проблемы возникли также сложности и с операциями между константами и векторами. В языке R для того, чтобы произвести эту операцию, существуют векторизованные функции lapply и sapply, которые применяют к вектору необходимую операцию с большей скоростью, чем проход по циклу. В Python же таких операций непосредственно в самом языке нет, в связи с чем было принято решение использовать наследование классов из языка R с дополненными необходимыми операциями. Данная функциональность реализована в некоторых пакетах [12] для языка Python, в связи с чем есть возможность альтернативного определения операций с векторами, но на текущий момент было принято решение оставить встроенные в язык R векторные операции в связи с тем, что они требуют меньшего количества операций конвертации между двумя языками. В листинге 3.2 приведен пример для вещественных векторов.

Листинг 3.2. Наследование для FloatVector.

```
class FloatVector(robjects.FloatVector):
        def __init__(self, obj):
        super().__init__(obj)
```

```
def __mul__(self, other):
        if isinstance(other, float):
                return r.sapply(self, "*", other)
        elif isinstance(other, int):
                return r.sapply(self, "*", other)

def __truediv__(self, other):
        if isinstance(other, float):
                return r.sapply(self, "/", other)
        elif isinstance(other, int):
                return r.sapply(self, "/", other)

def __add__(self, other):
        if isinstance(other, float):
                return r.sapply(self, "+", other)
        elif isinstance(other, int):
                return r.sapply(self, "+", other)

def __pow__(self, power, modulo=None):
        if isinstance(power, float)
           or isinstance(power, int):
                return robjects.r.sapply(self, "^", power)

__rmul__ = __mul__

__radd__ = __add__
```

В языках R и Python также существует разница между списками: в языке R список может быть как именованным, так и безымянным, в отличие от Python, где отдельно существуют списки без имен и словари с именами. Например, для того, чтобы передать в R список, состоящий из списков, необходимо либо преобразовать его в словарь, назначив имена, либо описать более сложную логику преобразований. В то же время,

данная возможность важна для пользователей пакета в связи с частым использованием вложенных списков при указании группировок компонент разложения. Функция, учитывающая данную логику, приведена в листинге ниже.

Листинг 3.3. Функция для преобразования к списку в языке R.

```
def list_to_vec(obj):
    if is_int_arr(obj):
        return IntVector(list(obj))
    elif is_float_arr(obj):
        return FloatVector(list(obj))
    elif is_of_int_lists_arr(obj):
        result = robjects.ListVector.from_length(len(obj))
        for i, x in enumerate(obj):
            if isinstance(x, int):
                result[i] = robjects.IntVector([x])
            else:
                result[i] = robjects.IntVector(list(x))
        return result
    else:
        return robjects.ListVector(list(obj))
```

3.2. Функции модуля Rssa

Аналогичным образом возникла необходимость в адаптации непосредственно функций модуля Rssa, где достаточно было лишь указать некоторые необходимые параметры, которые автоматически транслируются из Python в R посредством модуля rpy2, для которого ранее были указаны правила конвертации типов данных.

Важной составляющей при адаптации пакета Rssa является сохранение типов данных, получаемых в результате выполнения функций из него – разложения, восстановления, прогноза и других. Часть информации, которая необходима для дальнейшей визуализации и анализа данных, хранится в скрытом от пользователя виде, в связи с чем при трансляции на язык Python потребовались дополнительные действия для её

получения. Рассмотрим реализацию для разложения SSA.

Листинг 3.4. Реализация типа данных SSA в Python.

```
import pandas as pd
import numpy as np
from typing import Callable
from rpy2 import robjects
import rpy2.robjects.packages as rpackages


r_ssa = rpackages.importr('Rssa')
ssa_get = robjects.r('utils::getFromNamespace("$.ssa", "Rssa")')


class SSABase:

    def __init__(self, x, ssa_object, call):
        self.obj = ssa_object
        self.sigma = ssa_get(self.obj, "sigma")
        self.U = ssa_get(self.obj, "U").T
        self.V = ssa_get(self.obj, "V")
        self.series = pd.DataFrame(x)
        self.call = call

    @property
    def F(self):
        return self.series

    def contributions(self, idx=None)
        if idx is None:
            idx = range(1, self.nsigma() + 1)
        return r_ssa.contributions(self.obj, idx)
```

```
    def nspecial(self):
        return r_ssa.nspecial(self.obj)[0]

    def nsigma(self):
        return r_ssa.nsigma(self.obj)[0]

    def nu(self):
        return r_ssa.nu(self.obj)[0]

    def __str__(self):
        result = str(self.obj).split("\n")
        result[result.index("Call:") + 1] = self.call
        return "\n".join(result)

    def __repr__(self):
        return self.__str__()


class SSA(SSABase):

    def __init__(self, x,
                 L=None,
                 neig=None,
                 mask=None,
                 wmask=None,
                 kind="1d-ssa",
                 circular=False,
                 column_projector="none",
                 row_projector="none",
                 svd_method="auto",
                 call=None):
```

```
if L is None:
    L = (len(x) + 1) // 2

if isinstance(x, pd.DataFrame):
    x = x.iloc[:, 0]

self.L = L
self.kind = kind
super().__init__(x,
        r_ssa.ssa(x, L=L, neig=neig,
                  mask=mask, wmask=wmask,
                  kind=kind,
                  circular=circular,
                  column_projector=column_projector,
                  row_projector=row_projector,
                  svd_method=svd_method), call=call)
```

Описанный выше класс выделен в отдельный файл, а получение необходимых скрытых полей и методов производится с помощью функции `ssa_get`, получающей доступ к скрытому методу для доступа к данным, реализованному в Rssa. Сам класс SSA наследует методы и поля базового класса SSABase, которые являются общими для всех видов сингулярного разложения в пакете.

Помимо самих классов, в основном файле пакета pyrssa находятся функции, обращающиеся к соответствующим классам. Данная обертка позволит отделить в будущем различные виды схожих объектов на основе передаваемых параметров, а также сократить объем программного кода и упростить работу для пользователя. Список реализованных на данный момент функций пакета Rssa приведен в Приложении А. Далее представлен текущий код функции обращения к классу SSA, позволяющей построить необходимый вид разложения и передать дополнительные параметры, используемые при выводе информации о полученном объекте.

Листинг 3.5. Реализация функции для обращения к типу данных SSA в Python.

```
from pyrssa import SSA
```

```
import inspect

def ssa(x, L=None, neig=None, mask=None,
        wmask=None, kind="1d-ssa", circular=None,
        column_projector="none", row_projector="none",
        svd_method="auto"):

        return SSA(x, L=L, neig=neig, mask=mask, wmask=wmask,
                   kind=kind, column_projector=column_projector,
                   row_projector=row_projector,
                   svd_method=svd_method,
                   call=_get_call(inspect.currentframe().f_back))
```

Как для SSA, так и для других классов, реализованных для пакета pyrssa, были определены функции вывода полученного объекта. В листинге выше видно, что при передаче параметров для класса SSA также передается строка вызова функции, которая позволяет пользователю впоследствии отследить, с какими параметрами было произведено разложение. Аналогичным образом это производится в языке R, однако в Python из-за отличий синтаксиса языков возникла необходимость замены строки вызова. Пример вывода в языке Python приведен ниже.

Листинг 3.6. Вывод объекта SSA в языке Python.

```
Call:
prs.ssa(fort, L=84, kind="1d-ssa")

Series length: 174,      Window length: 84, SVD method: eigen
Special triples:  0

Computed:
Eigenvalues: 50,         Eigenvectors: 50, Factor vectors: 0

Precached: 0 elementary series (0 MiB)
```

```
Overall memory consumption (estimate): 0.0506 MiB
```

3.3. Визуализация

Помимо получения разложения и его вывода в формате текста, для дальнейшего анализа существует необходимость в визуализации полученных результатов. Первоначальные попытки свелись к тому, чтобы точно так же обращаться к исходным средствам языка R. Однако, в силу того, что метод для визуализации графиков в пакете Rssa был переопределен и отличается от стандартного, возникли непредвиденные сложности, в связи с чем пришлось прибегнуть к известному и хорошо задокументированному пакету matplotlib [13] для языка Python, который предназначен для визуализации графиков. Далее для удобства сравнения интерфейсов будет приведен программный код на языках программирования R и Python, результаты работы для каждого из которых изображены на графиках ниже.

Листинг 3.7. Программный код для визуализации разложения и восстановления ряда AustralianWine на языке R.

```
library("Rssa")

data("AustralianWine", package = "Rssa")
wine <- window(AustralianWine,
               end = time(AustralianWine)[174])

fort <- wine[, "Fortified"]
s.fort <- ssa(fort, L = 84, kind = "1d-ssa")

plot(s.fort)
plot(s.fort, type = "vectors", idx = 1:8)
plot(s.fort, type = "paired", idx = 2:11,
  plot.contrib = FALSE)
print(parestimate(s.fort, groups = list(2:3, 4:5),
method = "pairs"))
```

```
plot(wcor(s.fort, groups = 1:30),
scales = list(at = c(1, 5, 10, 15, 20, 25, 30)))

r.fort <- reconstruct(s.fort,
groups = list(Trend = 1,
Seasonality = 2:11))
plot(r.fort, add.residuals = TRUE, add.original = TRUE,
plot.method = "xyplot",
superpose = TRUE, auto.key = list(columns = 2))
plot(r.fort, add.residuals = TRUE, add.original = TRUE,
plot.method = "xyplot", layout=c(2, 2))
```

Листинг 3.8. Программный код для визуализации разложения и восстановления ряда AustralianWine на языке Python.

```
import pyrssa as prs
import pandas as pd
import numpy as np

AustralianWine = prs.data("AustralianWine")
fort = AustralianWine['Fortified'][:174]
fort.index = pd.date_range(start='1980/01/01', freq='M',
                           periods=len(fort))
s_fort = prs.ssa(fort, L=84, kind="1d-ssa")

prs.plot(s_fort)
prs.plot(s_fort, kind="vectors", idx=range(1, 9))
prs.plot(s_fort, kind="paired", idx=np.arange(2, 13),
         contrib=False)
prs.plot(prs.wcor(s_fort, groups=range(1, 31)),
       scales=[1] + list(range(5, 31, 5)))

r_fort = prs.reconstruct(s_fort,
```

```
                        groups={"Trend": 1,
                                "Seasonality": range(2, 12)})
prs.plot(r_fort, x=fort.index, add_original=True,
         add_residuals=True,
         method="xyplot", layout=(2, 2), superpose=False)
prs.plot(r_fort, x=fort.index, add_original=True,
         add_residuals=True,
         method="xyplot", superpose=True)
```

Также стоит отметить, что для построения графиков в пакете Rssa необходимо передавать параметр вида или метода для построения графика в общую для всех графиков функцию `plot`, что может быть не всегда удобно для пользователя в связи с тем, что для разных графиков наборы допустимых параметров могут отличаться. В связи с этим было принято решение написать обертку, позволяющую строить графики как старым способом, так и новым, обращаясь к интересующему виду или методу построения напрямую, без его указания в функции. Примерный программный код обертки и оба варианта обращения к функциям визуализации приведены ниже.

Листинг 3.9. Структура для функций визуализации в пакете pyrssa.

```
class Plot:

    @staticmethod
    def vectors(x: SSABase, idx=None, contrib=True, layout=None):
        ...

    @staticmethod
    def paired(x: SSABase, idx=None, contrib=True, layout=None):
        ...

    ...

    def __call__(self, obj, x_labels=None,
```

```
                kind: Literal["vectors", "paired"] = None,
                add_residuals=True, add_original=True,
                idx=None, scales=None, contrib=True,
                layout=None, superpose=False,
                method: Literal["matplot", "xyplot"] = None,
                support_lines=True):

        if kind == "vectors":
            return self.vectors(obj, idx=idx,
                                contrib=contrib, layout=layout)
        elif kind == "paired":
            return self.paired(obj, idx=idx,
                               contrib=contrib, layout=layout)
        ...

plot = Plot()
```

Листинг 3.10. Различные возможности обращения к функциям визуализации в Python.

```
import pyrssa as prs

AustralianWine = prs.data("AustralianWine")
fort = AustralianWine['Fortified'][:174]
s_fort = prs.ssa(fort, L=84, kind="1d-ssa")

# Old style plot call
prs.plot(s_fort, kind="vectors", idx=range(1, 9))

# New style plot call
prs.plot.vectors(s_fort, idx=range(1, 9))
```

3.3.1. Разложение

Для разложения SSA существует несколько различных основных визуализаций, к которым относятся график норм собственных векторов полученного разложения, парный и одиночный графики собственных векторов, график взвешенных корреляций. В силу большого объема каждой функции программный код реализации будет приведен только для взвешенных корреляций.

Листинг 3.11. Визуализация взвешенных корреляций с помощью matplotlib.

```
from pyrssa import WCorMatrix
import matplotlib.pyplot as plt

@staticmethod
def _wcor(wcor_matrix: WCorMatrix, scales=None, support_lines=True):
    plt.imshow(wcor_matrix, cmap='gray_r', vmin=0, vmax=1)
    plt.gca().invert_yaxis()

    if scales is None:
        ticks = range(len(wcor_matrix.groups))
        labels = wcor_matrix.groups
    else:
        ticks = np.array(scales) - 1
        labels = scales
    if support_lines:
        for i in scales:
            plt.plot([i - 1, i - 1],
                     [0, len(wcor_matrix.groups) - 1],
                     "k-.", lw=0.5, alpha=0.5)
            plt.plot([0, len(wcor_matrix.groups) - 1],
                     [i - 1, i - 1],
                      "k-.", lw=0.5, alpha=0.5)

    plt.title("W-correlation matrix")
```

```
plt.xticks(ticks, labels=labels)
plt.yticks(ticks, labels=labels)

minor_cnt = wcor_matrix.shape[0]
plt.xticks(np.arange(-.5, minor_cnt, 1), minor=True)
plt.yticks(np.arange(-.5, minor_cnt, 1), minor=True)

plt.grid(which='minor', color='w', linestyle='-', linewidth=0.2)
plt.tick_params(which='minor', bottom=False, left=False)
plt.show()
```

Ниже будут приведены примеры для каждого из перечисленных выше графиков, а также сравнение визуализации в Python и R. За основу взят временной ряд AustralianWine, приводимый в примерах [2].

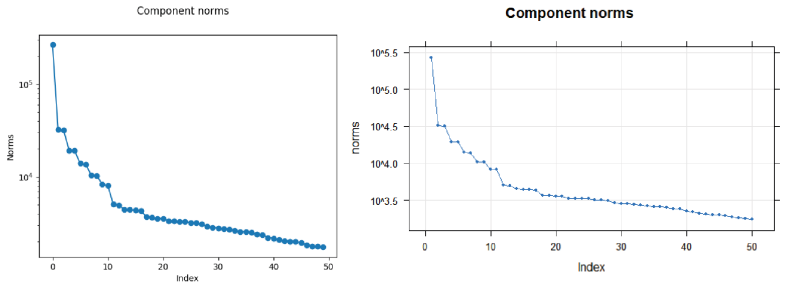

Рис. 3.1. График норм компонент разложения в Python и R.

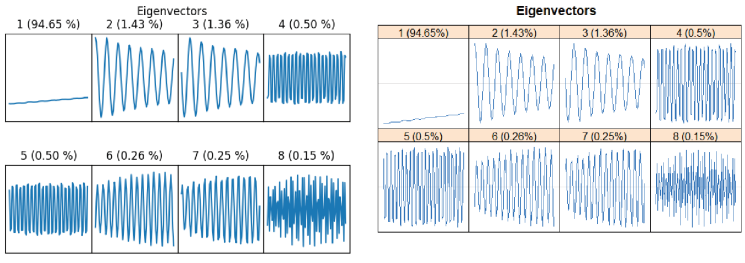

Рис. 3.2. График собственных векторов компонент разложения в Python и R.

3.3.2. Восстановление

После первичного анализа разложения временного ряда и выбора компонент для восстановления исходного ряда, возникает необходимость в визуализации результатов

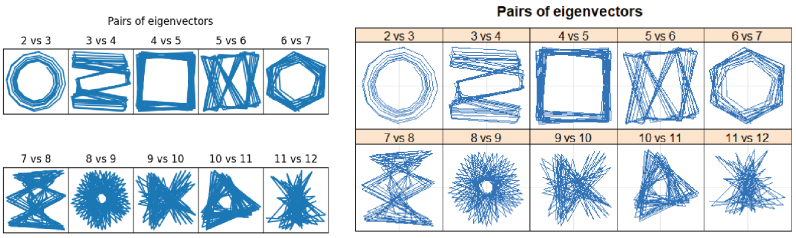

Рис. 3.3. Парный график собственных векторов компонент разложения в Python и R.

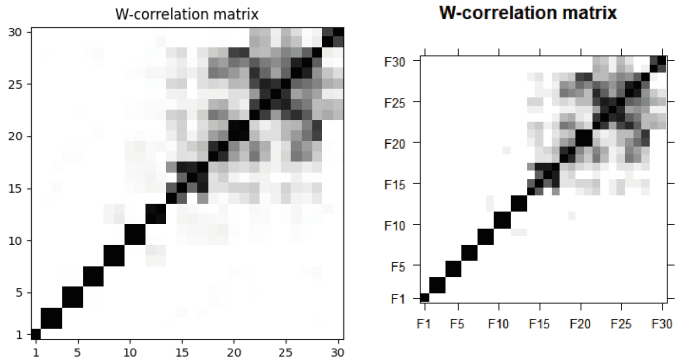

Рис. 3.4. График взвешенных корреляций компонент разложения в Python и R.

восстановления. Функция отрисовки графиков имеет регулируемые параметры, позволяющие изображать полученное восстановление как на общем графике, так и на отдельных.

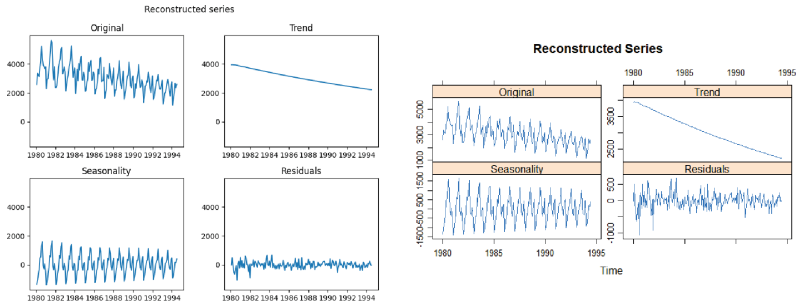

Рис. 3.5. Визуализация восстановления на отдельных графиках в Python и R.

Реализованные на данный момент графики в будущем нуждаются в косметических доработках, изменениях и улучшениях, однако на данный момент они обладают достаточно хорошей информативностью для использования в исследованиях, а график взвешенных корреляций имеет даже более высокую детализацию по сравнению с аналогичным в R.

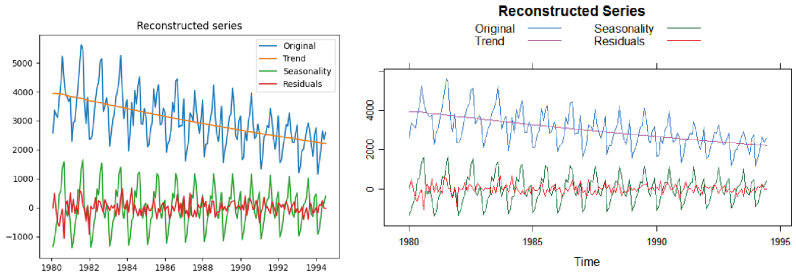

Рис. 3.6. Визуализация восстановления на общем графике в Python и R.

3.4. Модельный пример

Полученный в рамках задачи по адаптации модуля Rssa к языку Python результат может быть применен к какому-либо временному ряду. Для большей наглядности возьмем за основу модельный ряд, в котором будут присутствовать тренд, шум и сезонность.

$$F_N = F_N^{(1)} + F_N^{(2)} + F_N^{(3)},$$

где $F_N^{(1)} = (f_1^{(1)}, \ldots, f_N^{(1)}), f_n^{(1)} = 0.65n$ – компонента ряда, содержащая тренд. $F_N^{(2)} = (f_0^{(2)}, \ldots, f_{N-1}^{(2)}), f_n^{(2)} = \cos(\frac{2\pi n}{6})$ – компонента ряда, отвечающая за периодичность, а $F_N^{(3)} = (f_1^{(3)}, \ldots, f_N^{(3)})$ – шум, где $f_n^{(3)}$ – независимые случайные величины, имеющие одно и то же нормальное распределение с нулевым средним и стандартным отклонением, равным 0.5.

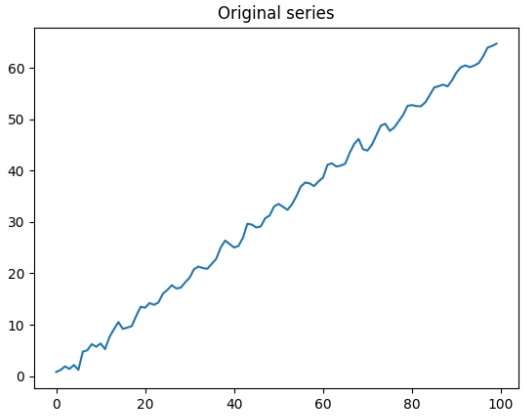

Рис. 3.7. График исходного временного ряда.

Далее, для разложения данного ряда на тренд, шум и периодичность, необходимо задать определенную длину окна. В связи с тем, что мы будем анализировать ряд

базовым методом, нет смысла брать длину окна, большую, чем половина длины ряда. Таким образом, если $N = 100$, то $L \leq 50$.

Для разделения временного ряда на компоненты сперва обратимся к графикам полученных с помощью метода SSA собственных векторов.

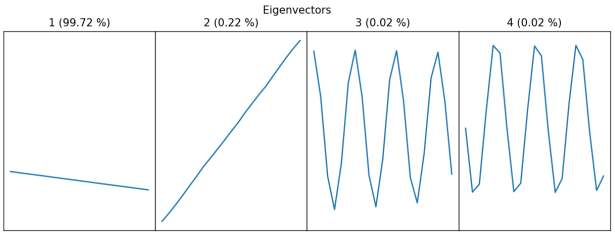

Рис. 3.8. Собственные векторы.

В связи с тем, что в данном ряде тренд ярко выраженный, на первых позициях будет находиться большинство собственных векторов, соответствующих тренду. Таким образом, чтобы извлечь из имеющегося ряда тренд, необходимо собрать тройки, соответствующие медленно меняющимся сингулярным векторам. В данном случае, это первые два вектора.

Помимо этого необходимо выделить и сезонность. Третий и четвертый векторы имеют периодические колебания, в связи с чем для большей наглядности можно их сгруппировать и построить график.

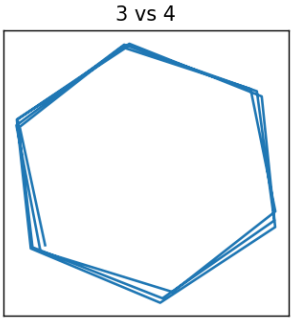

Рис. 3.9. Двумерная диаграмма пары векторов 3 и 4.

Из данного графика достаточно хорошо видно, что период $T = 6$, так как на графике отчетливо виден шестиугольник. Это соответствует действительности, потому что в данном модельном ряде был выбран именно такой период. Далее построим реконструк-

цию ряда, где первая и вторая компоненты будут соответствовать тренду, а третья и четвертая – сезонности.

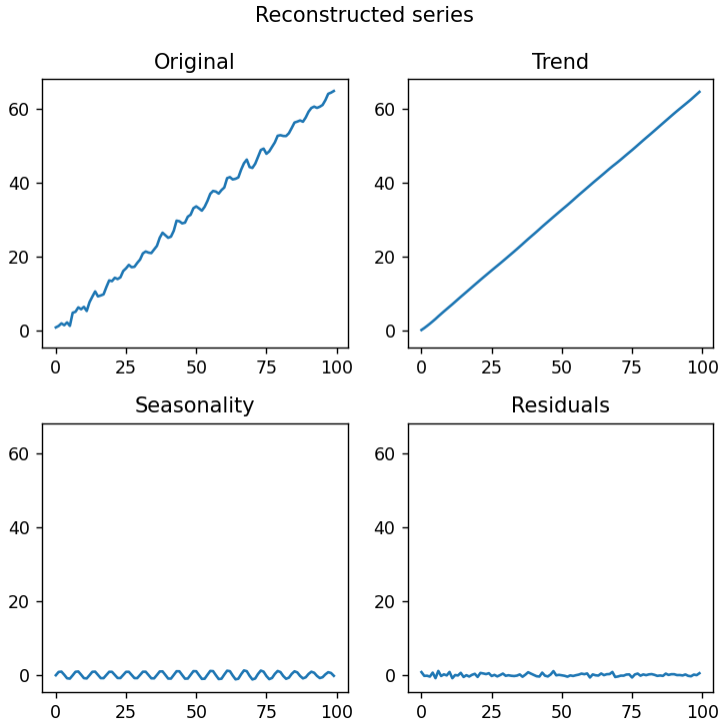

Рис. 3.10. Реконструкция временного ряда.

С использованием средств визуализации в Python можем проверить качество получившегося разложения.

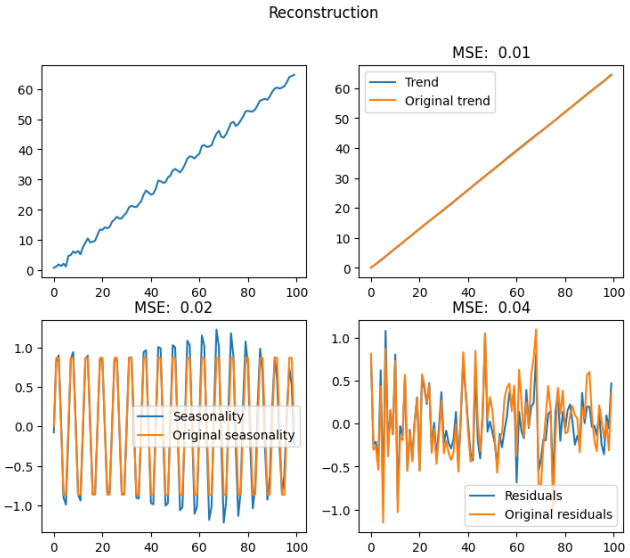

Рис. 3.11. Реконструкция модельного временного ряда.

Как видно из графиков, точность разложения достаточно высока, а тренд и сезонность определились практически идеально. Ниже приведен код, в результате которого удалось построить приведенные выше графики.

Листинг 3.12. Программный код для разложения и восстановления модельного ряда на Python.

```
import pyrssa as prs
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse


np.random.seed(1)
N = 100
sigma = 0.5
noise = np.random.normal(size=N) * sigma
trend = 0.65 * np.arange(N)
seasonality = np.sin(2 * np.pi * np.arange(N) / 6)
series = seasonality + noise + trend

plt.plot(series)
plt.title("Original series")
plt.show()

s_series = prs.ssa(series, L=21, kind="1d-ssa")

prs.plot(s_series, kind="vectors", idx=range(1, 5))
prs.plot(s_series, kind="paired", idx=range(3, 5),
contrib=False)

r_series = prs.reconstruct(s_series,
groups={"Trend": range(1, 3),
        "Seasonality": range(3, 5)})
fig, ax = plt.subplots(2, 2)
```

```
fig.suptitle("Reconstruction")
ax[0, 0].plot(np.arange(N), series, label="Original")
ax[0, 1].plot(np.arange(N), r_series.Trend, label='Trend')
ax[0, 1].plot(np.arange(N), trend, label='Original trend')
ax[0, 1].set_title(
f"MSE: {mse(trend, r_series.Trend): .2f}")
ax[0, 1].legend()
ax[1, 0].plot(np.arange(N), r_series.Seasonality,
label='Seasonality')
ax[1, 0].plot(np.arange(N), seasonality,
label='Original seasonality')
ax[1, 0].set_title(
f"MSE: {mse(seasonality, r_series.Seasonality): .2f}")
ax[1, 0].legend()
ax[1, 1].plot(np.arange(N), r_series.residuals,
label='Residuals')
ax[1, 1].plot(np.arange(N), noise,
label='Original residuals')
ax[1, 1].set_title(
f"MSE: {mse(noise, r_series.residuals): .2f}")
ax[1, 1].legend()
plt.show()
```

Далее построим прогноз полученного модельного ряда.

Листинг 3.13. Программный код для прогноза модельного ряда на Python.

```
for_length = 100
for_range = np.arange(N, N + for_length)
sigma = 0.5
for_trend = 0.65 * for_range
for_seasonality = np.sin(2 * np.pi * for_range / 6)
for_signal = for_seasonality + for_trend
```

```
rfor = prs.rforecast(s_series, groups={"Signal": range(1, 5)},
length=for_length)
plt.plot(for_signal)
plt.plot(rfor.Signal)
plt.legend(["Original series", "Forecasted series"])
plt.title(f"MSE: {mse(for_signal, rfor.Signal): .2f}")
plt.show()
```

Полученный результат приведен на графике ниже.

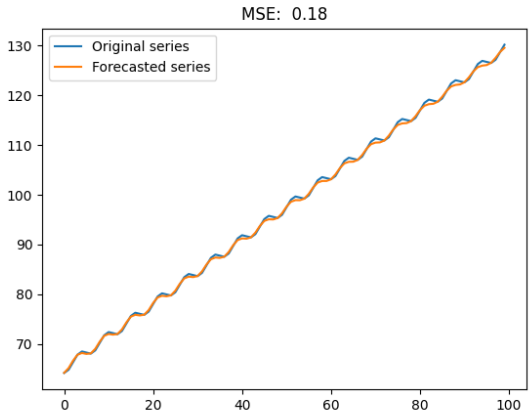

Рис. 3.12. Прогноз временного ряда.

3.5. Установка модуля

Как было сказано ранее, данный пакет опубликован на сайте проекта PyPi [14] и может быть установлен с помощью простой команды pip install pyrssa. При установке данного пакета автоматически подтягиваются зависимости, связанные с модулями NumPy, pandas, rpy2, matplotlib. Дополнительно проверяется наличие установленной библиотеки Rssa и в противном случае производится ее установка. Программный код, реализующий установку зависимостей, приведен ниже.

Листинг 3.14. Настройки проекта для публикации пакета на сайте PyPi.

```
from setuptools import setup

setup(
```

```
name='pyrssa',
packages=['pyrssa', 'pyrssa.classes'],
include_package_data=True,
package_data={
        'data': ['*']},
version='1.0.6',
description='Rssa for Python',
license="Apache 2.0",
install_requires=['rpy2<=3.4.5', 'pandas',
                  'numpy', 'matplotlib'])
```

Листинг 3.15. Содержимое файла для инсталляции модуля Rssa.

```
import rpy2.robjects.packages as rpackages
from rpy2.robjects.vectors import StrVector
utils = rpackages.importr('utils')
utils.chooseCRANmirror(ind=1)

def install_required():
        pack_names = ('Rssa',)
        names_to_install = [x for x in pack_names
                    if not rpackages.isinstalled(x)]
        if len(names_to_install) > 0:
                utils.install_packages(
                StrVector(names_to_install))
```

На данный момент для пакета rpy2 обнаружены проблемы, начиная с версии 3.5.0, которые относятся к автоматическому приведению типов из R в Python, в связи с чем часть функциональности модуля теряется, поэтому на данный момент поставлено ограничение на версию модуля rpy2 вплоть до 3.4.5. Также, для того, чтобы модуль Rssa мог быть установлен, пользователь предварительно должен установить и настроить текущую версию языка R на своем устройстве, что является некоторым барьером в применении данного модуля.

Глава 4

Анализ временного ряда

4.1. Исходные данные

Исходными данными для анализа являются временные ряды параметров вращения Земли (ПВЗ), опубликованные Международной службой вращения Земли (МСВЗ) в бюллетене C04 [15]. Описание рядов и процедур, используемых при обработке, приводится в [16]. Каждый из параметров измеряется ежедневно, за основу взята модифицированная юлианская дата (MJD), используемая Международным астрономическим союзом в качестве стандарта нумерации дней.

В рамках данной работы будет проанализирован ряд LOD (Length Of Day), который отражает разницу между фактическим временем полного оборота Земли вокруг своей оси и 86400 секундами. Ниже приведен график данного ряда.

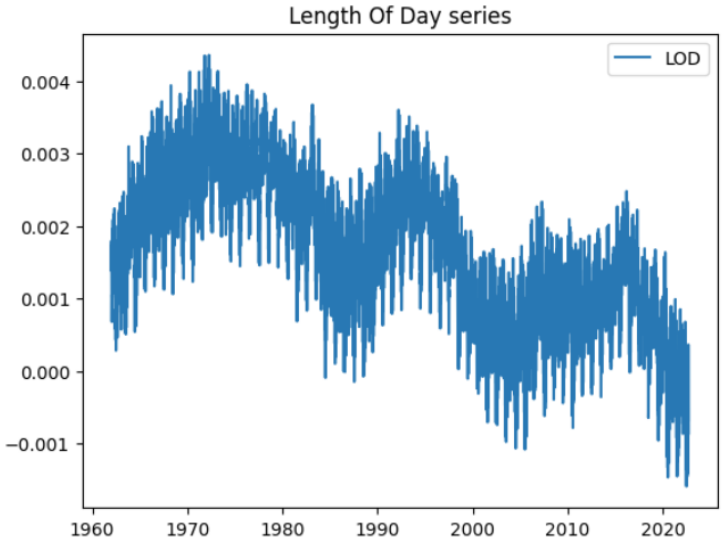

Рис. 4.1. График ряда долготы дня.

4.2. Предварительный анализ ряда

Сперва проанализируем ряд, оценив его параметры. Для анализа периодических составляющих ряда прибегнем к построению периодограммы, отражающей вклад различных частот в разложение Фурье исходного ряда. Построим графики с различными

решетками частот для более наглядного выделения существующих пиков периодограммы.

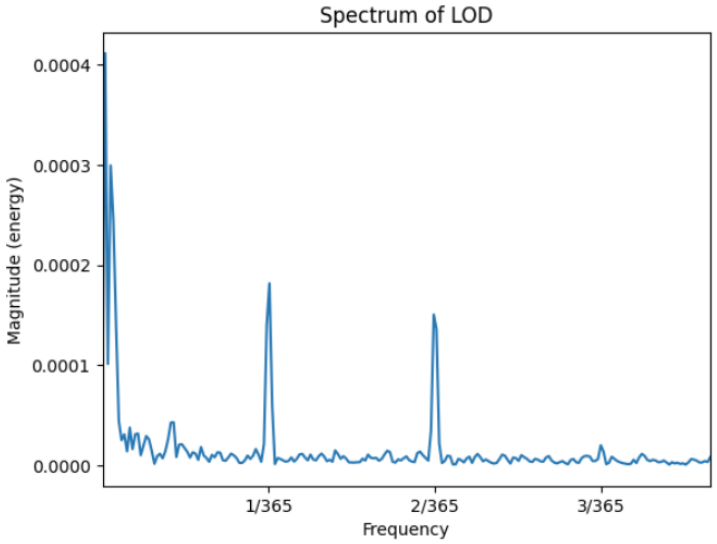

Рис. 4.2. Периодограмма ряда долготы дня для годовой решетки частот.

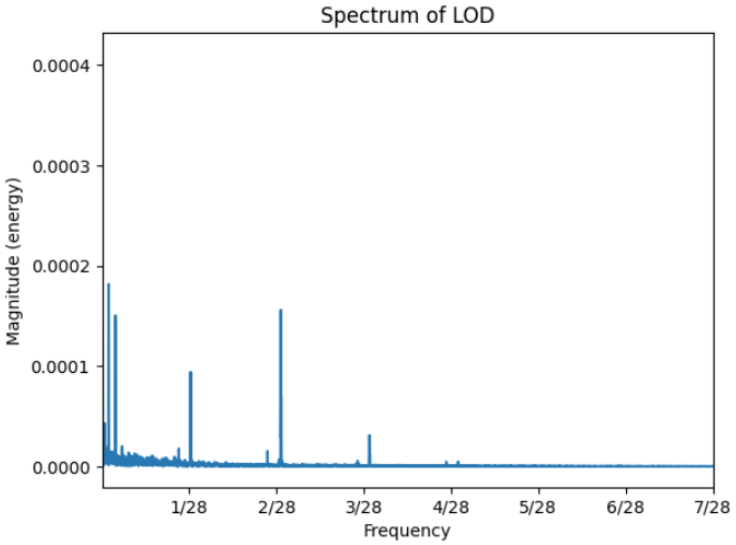

Рис. 4.3. Периодограмма ряда долготы дня для месячной решетки частот.

Как видно из графиков, периоды присутствующих в рядах гармоник приблизительно равны 365 и 28 дням. Рассмотрим базовое разложение SSA для данных рядов, отделив сезонные компоненты.

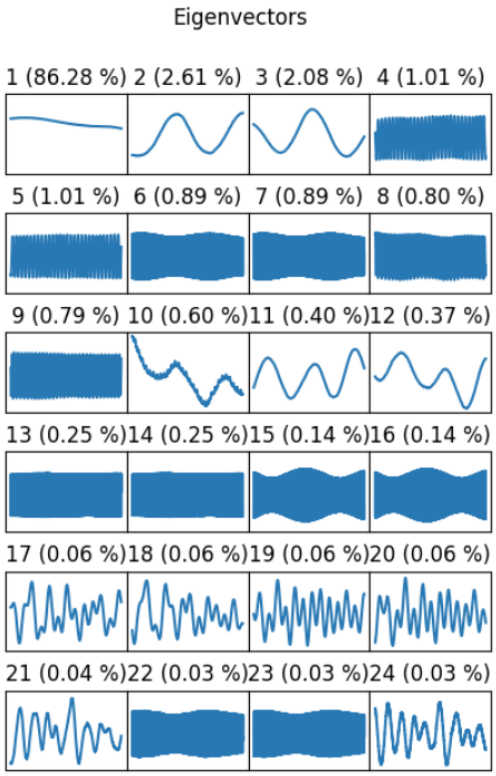

Рис. 4.4. График собственных векторов разложения ряда долготы дня.

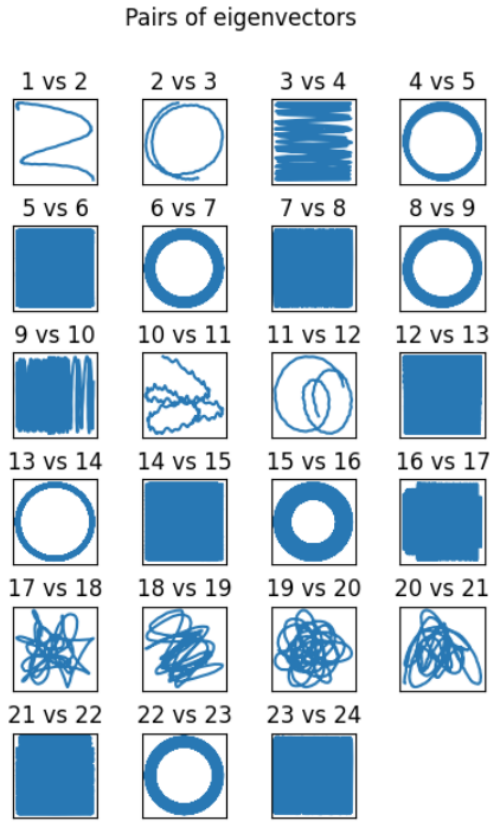

Рис. 4.5. Парный график собственных векторов разложения ряда долготы дня.

На графике векторов ряда долготы дня видны пары 4 и 5, 8 и 9, которые являются составляющими периода длиной в 365 дней, а также 6 и 7, 13 и 14, которые составляют период 28 дней. Оставшиеся пары также образуют один из двух указанных выше периодов.

4.3. Подбор параметров для прогноза

Для выбора параметров L, r и метода прогнозирования при анализе ряда долготы дня прибегнем к перебору по сетке, рассмотрев все комбинации параметров $L = 300, 600, \ldots, 3600$, $r = 1, \ldots, 50$, а также рекуррентный и векторный метод прогноза.

В качестве меры ошибки будем использовать среднеквадратическую ошибку MSE (Mean Squared Error). Разделим ряд на несколько составляющих: базовый отрезок длиной в 15 лет, на основе которого строится разложение, валидационный отрезок длиной в 10 лет, который используется для оценки прогноза, а также тестовый отрезок длиной в один год, на котором полученный результат будет проверен впоследствии. Для наиболее качественного оценивания прибегнем к кросс-валидации, в рамках которой рассматриваемые базовый и валидационный отрезок будут смещаться по очереди 10 раз на один год вперед вплоть до начала тестового отрезка. Выбранные ранее параметры были предварительно оценены как оптимальные в другой работе, связанной с анализом данных рядов [17].

Результат данного алгоритма выведем с помощью объемного графика, по оси x и y которого находятся параметры L, r, а по оси z изображено значение MSE. В связи с тем, что разница между некоторыми значениями составляет несколько порядков, данный график приведен в логарифмической шкале по оси z. Дополнительно с помощью цветовой карты от синего до красного цвета изображены лучшие и худшие значения среднеквадратической ошибки.

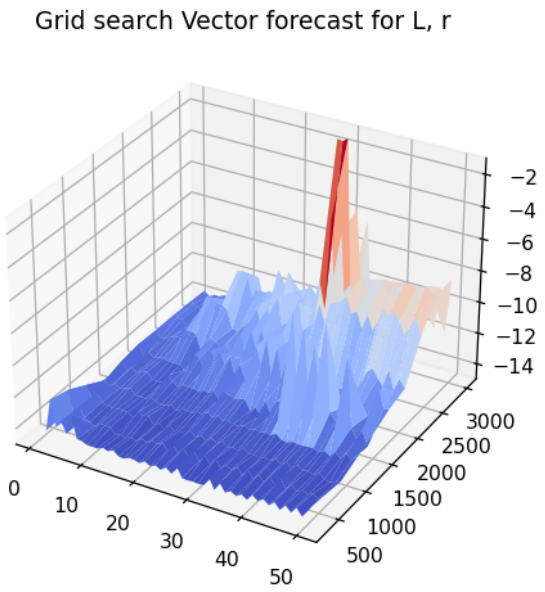

Рис. 4.6. График зависимости MSE от различных значений параметров L, r.

По данному графику видно, что наилучшее значение достигнуто для наименьшего по параметру $L = 300$, в связи с чем построим аналогичную сетку, выбрав параметр $L = 100, 200, \ldots, 600$.

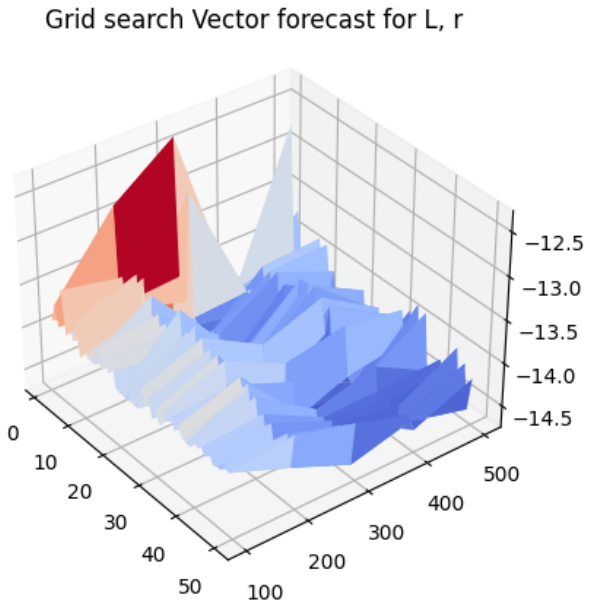

Рис. 4.7. График зависимости MSE от различных значений параметров L, r на малой сетке по параметру L.

В связи с тем, что векторный прогноз показал более точные результаты при подборе параметров, остановимся на этом методе. Наименьшее значение MSE действительно

оказалось достигнуто на параметрах $L = 300, r = 32$. Построим прогноз на основе данных параметров.

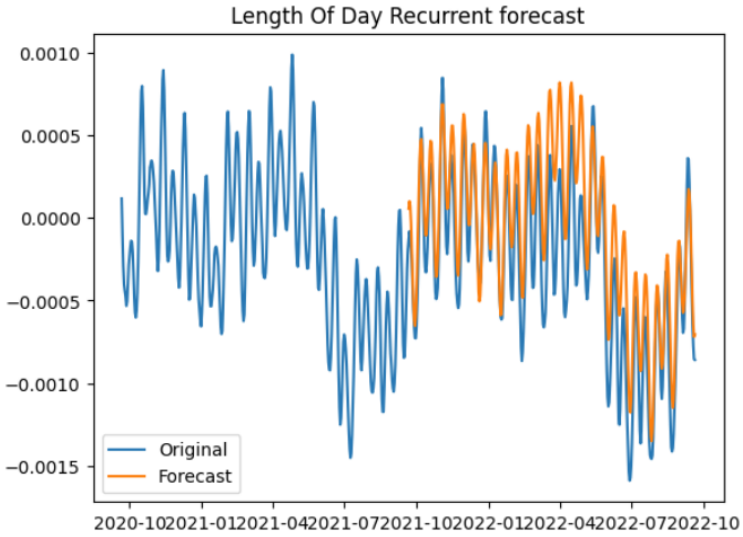

Рис. 4.8. График прогноза на основе подобранных параметров.

Полученный результат прогнозирования можно сравнить с прогнозом, полученным с помощью сайта [18], написанного в рамках указанной ранее работы [17].

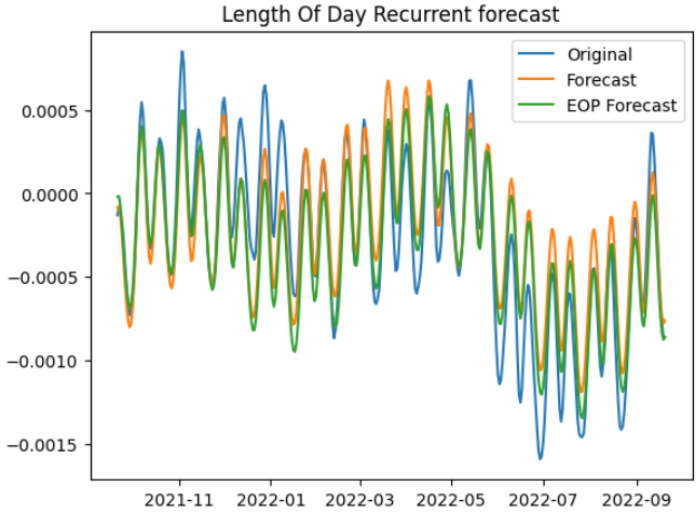

Рис. 4.9. График сравнения прогнозов временного ряда долготы дня.

Визуально данные прогнозы достаточно близки, в различных частях ряда каждый из прогнозов описывает значения лучше другого, а при проверке значений MSE можно заключить, что отличия у прогнозов минимальны: 6.6e-8 для прогноза, полученного в рамках анализа, и 6.0e-8 для прогноза с помощью сайта.

Глава 5

Написание тестов для pyrssa

Несмотря на то, что пакет pyrssa является адаптацией пакета Rssa, для которого уже существуют исчерпывающие тесты функциональности, для проверки правильности работы в рамках языка Python требуется портирование тестов и возможность проверки получаемых результатов, которые потенциально могут быть искажены при конвертации типов данных, а также в случае, если какие-либо функции пакета rpy2 не работают так, как это предполагается.

5.1. Загрузка данных

Для тестов, используемых в пакете Rssa, были заранее сгенерированы эталонные данные, представляющие собой корректные результаты работы проверяемых функций. Для краткости описание структуры будет приведено к тестам, написанным для базового метода SSA. Хранение файлов организовано в формате .rda (R Data), который является форматом хранения данных в языке R, и в данном случае этот файл содержит различные временные ряды, для которых сосчитаны результаты работы функций при различных значениях параметра длины окна. Помимо этого, для них также указаны разные варианты группировки компонент, получаемых при разложении исходных рядов. Также для каждого из временных рядов дополнительно указаны параметры, содержащие название ряда, названия содержащихся в них результатов, наборы используемых значений параметра длины окна, группировки, указывается точность, с которой должны совпадать эталонные и получаемые значения, а также различные методы построения сингулярного разложения для рядов.

Для наглядности работы и возможности внесения корректировок в рамках языка Python реализована структура хранения данной информации в различных папках. Таким образом, в корневой папке для тестов находится дочерняя директория, содержащая данные, внутри которой идет подразделение на виды проверок, для каждого из видов созданы директории под различные длины окна, а внутри них в формате .csv собраны таблицы, столбцы которых состоят из результатов различных вариантов группировки компонент. Хранение дополнительных параметров организованы в формате

JSON. Примерное содержание такого файла указано в листинге ниже.

Листинг 5.1. Содержимое файла JSON с параметрами для тестов.

```
{
        "name": "co2",
        "names": ["series", "reconstruction",
          "rforecast.orig",
          "rforecast.rec", "vforecast"],
        "what": ["reconstruct", "rforecast", "vforecast"],
        "kind": "1d-ssa",
        "Ls":
                {"reconstruct":  [17, 234, 235, 300, 400],
         "forecast":  [17, 100, 222, 234]},
        "groups":
  {"reconstruct": [[1], [2], [3], [4], ...],
                "forecast":  [[1], [1, 2], [3, 4, 5], ...]},
        "len": 100,
        "neig": 20,
        "tolerance": 2e-07,
        "svd_methods":
                {"reconstruct": [
                        ["svd", "eigen", "propack"]],
         "forecast": [
                        ["svd", "eigen", "propack"]]}
}
```

После обновления структуры хранения данных для тестов также возникла необходимость в их корректном считывании при загрузке в Python. Для решения этой задачи были также реализованы специальные функции. Код для этих функций приведен ниже за исключением значений некоторых параметров и кода иных функций, которые в целом не влияют на основную логику.

Листинг 5.2. Функции для загрузки данных для тестирования.

```
def load_df(df_name, kinds: list, file_format="csv"):
 filename = os.path.join(TEST_DATAFRAME_DIRECTORY,
                         df_name)
    print(filename)
    if kinds is None:
        kinds = os.listdir(filename)
    data_folders = [
    [os.path.join(filename, name), name]
    for name in kinds
    if os.path.isdir(os.path.join(filename,
                                  name))]
    df = {"series":
        pd.read_csv(
            os.path.join(
                filename,
                f"series.{file_format}")),
          "pars": None}

        params_path = os.path.join(filename,
                                   "pars.json")
        if os.path.exists(params_path):
        df["pars"] = read_json(params_path)

        for data_folder in data_folders:
        folder_files = [
  [os.path.join(data_folder[0], file), file]
  for file in os.listdir(data_folder[0])]
        df[data_folder[1]] = {
     file[1].replace(f".{file_format}", ""):
     pd.read_csv(file[0])
     for file in folder_files}
```

```
    return df


def load_dataframes(df_names=DATAFRAME_NAMES,
kinds=DATAFRAME_KINDS):
        return {df_name: load_df(df_name, kinds=kinds)
                        for df_name in df_names}
```

5.2. Тестирование

После загрузки данных следующей задачей становится их непосредственное тестирование. Для упрощения и автоматизации части процессов за основу был взят стандартный для Python модуль unittest [19], поставляемый вместе с программной оболочкой языка. Удобство данного модуля заключается также и в том, что с его помощью можно быстро запускать тесты в продвинутых средах разработки, имея возможность посмотреть на результаты каждого из тестов без необходимости в изучении лишь отладочной информации. Однако, при решении задачи написания тестов возникли сложности с тем, что встроенной поддержки сравнения табличных данных данный пакет не имеет, а визуальное восприятие больших объемов кода было затруднено. В связи с этим тестирование посредством модуля unittest и действительное сравнение значений были вынесены в различные места. Далее будут приведены выдержки, относящиеся к каждой из тестируемых функций.

5.2.1. Восстановление

Для проведения тестов на восстановление временного ряда предварительно строится его разложение на компоненты, а затем с учетом параметров тестирования происходит подсчет необходимых значений. Ниже приведены листинги, относящиеся к тестированию правильности восстановления временных рядов.

Листинг 5.3. Функции для подсчета значений восстановления временного ряда при тестировании.

```
def compute_reconstructions(
                x, Ls, groups,
                kind: Literal["1d-ssa", "toeplitz-ssa"] = "1d-ssa",
                svd_method: Literal["auto", "eigen",
                                    "propack", "nutrlan",
                                    "svd"] = "auto",
                neig=None,
                column_projector="none",
                row_projector="none",
                **kwargs):
    if neig is None:
        if isinstance(groups, dict):
            neig = np.max(list(chain(*groups.values())))
        elif isinstance(groups, list):
            neig = np.max([np.max(i) for i in groups])
        else:
            raise TypeError(f"Wrong type for groups:
                            {type(groups)}")

    def compute_reconstruction(L):
        ss = prs.ssa(x, L, kind=kind, svd_method=svd_method,
                     neig=min(neig, L),
                     column_projector=column_projector,
                     row_projector=row_projector,
                     **kwargs)
        rec = prs.reconstruct(ss, groups=groups)
        return pd.DataFrame({key: rec[key] for key in rec.names})

    return {f"L{L_value}": compute_reconstruction(L_value)
            for L_value in Ls}
```

Тестирование функций подразумевает перебор различных параметров для полу-

чения результатов. В силу большого объема функции, в которой производится тестирование, фрагмент, относящийся к инициализации параметров будет опущен, и далее приведены лишь выдержки кода, производящие перебор по параметрам.

Листинг 5.4. Перебор параметров восстановления и проверка результатов на правильность.

```
for i in range(len(Ls)):
    L = Ls[i]
    L_name = f"L{L}"
    for svd_method in svd_methods[i]:
        print(f"Data: {name},
                  Kind: {kind},
                  SVD-Method: {svd_method},
                  L: {L}...", end="")
    reconstruction =
        compute_reconstructions(series, [L],
                                groups=groups,
                                kind=kind,
                                svd_method=svd_method,
                                neig=neig,
                                column_projector=column_projector,
                                row_projector=row_projector)
    pd.testing.assert_frame_equal(
        reconstruction[L_name],
        test_data["reconstruction"][L_name],
        rtol=tolerance)
    print("OK")
```

Описанный в листинге выше код выполняет перебор значений параметра длины окна, а также для каждого из них перебор различных методов построения сингулярного разложения. Для каждой комбинации параметров строится свое разложение и затем с помощью функции пакета pandas, позволяющей сравнивать наборы данных, производится сравнение значений с заданной точностью. В случае, если возникают от-

личия, функция проверки выбрасывает исключение. Ниже приведен код, относящийся к запуску тестов и обработке возникющих исключений.

Листинг 5.5. Запуск тестов правильности восстановления временного ряда и обработка исключений.

```
import unittest
from tests.tests_methods import test_test_data, all_test_data


class Test1DSSA(unittest.TestCase):

    def test_reconstruct(self):
        what = "reconstruct"
        names = ["co2", "fr1k.nz", "fr1k", "fr50", "fr50.nz"]
        for name in names:
            try:
                test_test_data(what, all_test_data[name])
            except:
                self.fail("Wrong result")
```

В приведенном фрагменте кода производится наследование класса TestCase модуля unittest, благодаря чему при запуске тесты автоматически подгружаются и поочередно запускаются. При этом, с использованием конструкции try-except обрабатывается исключение, в случае возникновения которого дальнейшее проведение тестов для текущей функции останавливается и производится переход к следующим функциям.

5.2.2. Прогнозирование

Тестирование производится как для рекуррентного, так и для векторного методов прогнозирования, однако, в связи с практически идентичной логикой программного кода, в некоторых случаях приведены будут лишь фрагменты, относящиеся к рекуррентному варианту. Стоит отметить лишь одно существенное отличие: рекуррентный прогноз может быть построен по значениям как исходного ряда, так и восстановленно-

го, в то время, как векторный прогноз основывается лишь на полученных в результате разложения результатах. В листинге ниже приведена функция подсчета результатов прогнозирования для обоих методов в зависимости от значения параметра метода прогноза.

Листинг 5.6. Функции для подсчета значений прогноза временного ряда при тестировании.

```
def compute_forecasts(
                    x, Ls, groups, length,
                    kind: Literal["1d-ssa", "toeplitz-ssa"] = "1d-ssa",
                    forecast_method: Literal["recurrent",
                                             "vector"] = "recurrent",
                    base: Literal["reconstructed",
                                  "original"] = "reconstructed",
                    svd_method: Literal["auto", "eigen",
                                        "propack", "nutrlan",
                                        "svd"] = "auto",
                    neig=None,
                    column_projector="none",
                    row_projector="none",
                    **kwargs):

    if neig is None:
        if isinstance(groups, dict):
            neig = np.max(list(chain(*groups.values())))
        elif isinstance(groups, list):
            neig = np.max([np.max(i) for i in groups])
        else:
            raise TypeError(f"Wrong type for groups: {type(groups)}")

    def compute_forecast(L):
        ss = prs.ssa(x, L, kind=kind, svd_method=svd_method,
                     neig=min(neig, L),
```

```
                    column_projector=column_projector,
                    row_projector=row_projector,
                    **kwargs)
        if forecast_method == "recurrent":
            rec = prs.rforecast(ss, groups=groups,
                                length=length, base=base,
                                only_new=False)
        else:
            rec = prs.vforecast(ss, groups=groups,
                                length=length, only_new=False)

        return pd.DataFrame({key: rec[key] for key in rec.names})

    return {f"L{L_value}": compute_forecast(L_value)
            for L_value in Ls}
```

Листинг 5.7. Перебор параметров рекуррентного прогнозирования и проверка результатов на правильность.

```
for i in range(len(Ls_forecast)):
    L = Ls_forecast[i]
    L_name = f"L{L}"
    for svd_method in svd_methods[i]:
        print(f"Data: {name},
              Kind: {kind},
              SVD-Method: {svd_method},
              L: {L}...", end="")
        rforecast_orig =
            compute_forecasts(series,
                              [L],
                              groups=forecast_groups,
                              length=length,
```

```
                        kind=kind,
                        forecast_method="recurrent",
                        base="original",
                        svd_method=svd_method,
                        neig=neig,
                        column_projector=column_projector,
                        row_projector=row_projector,
                        **kwargs)
rforecast_rec =
    compute_forecasts(series,
                        [L],
                        groups=forecast_groups,
                        length=length,
                        kind=kind,
                        forecast_method="recurrent",
                        base="reconstructed",
                        svd_method=svd_method,
                        neig=neig,
                        column_projector=column_projector,
                        row_projector=row_projector,
                        **kwargs)

pd.testing.assert_frame_equal(
    rforecast_orig[L_name],
    test_data["rforecast.orig"][L_name],
    rtol=tolerance)
pd.testing.assert_frame_equal(
    rforecast_rec[L_name],
    test_data["rforecast.rec"][L_name],
    rtol=tolerance)
print("OK")
```

Листинг 5.8. Запуск тестов правильности рекуррентного прогнозирования временного ряда и обработка исключений.

```
import unittest
from tests.tests_methods import test_test_data, all_test_data


class Test1DSSA(unittest.TestCase):
    def test_rforecast(self):
        what = "rforecast"
        names = ["co2", "fr1k.nz", "fr1k", "fr50", "fr50.nz"]
        for name in names:
            try:
                test_test_data(what, all_test_data[name])
            except:
                self.fail("Wrong result")
```

В рамках тестирования функций восстановления и прогнозирования были выявлены и устранены проблемы, связанные с обработкой параметров при их передаче в rpy2, однако дальнейшее тестирование может выявить иные проблемы, которые также потребуют корректировки работы некоторых функций.

Глава 6

Сравнение скорости Rssa и pyrssa

При использовании пакета Rssa в качестве основы для реализации метода SSA на языке Python одной из важнейших характеристик является скорость, а если быть точнее, то ее сохранение близкой к исходной. Для полноценного анализа скорости работы возьмем за основу ранее рассмотренный ряд длины дня из набора данных ПВЗ. Замер времени работы будет проводиться при различных комбинациях параметров L длины окна и r количества компонент, на основе которых сначала строилось разложение SSA, а затем восстановление ряда. Программный код для обоих языков программирования приведен ниже.

Листинг 6.1. Программный код для анализа времени работы на языке R.

```
L_range <- seq(from=300, to=3000, by=300)
r_range <- 1:50

ssa_check <- function(time_series, L, r) {

        time_series_ssa <- ssa(time_series, L=L)
        ser_rec <- reconstruct(time_series_ssa,
                               groups=list("Signal"=1:r))
        return(ser_rec)

}

for (L_value in L_range) {
        for(r_value in r_range) {
                time_df[r_value, as.character(L_value)] <-
                system.time(ssa_check(series, L_value, r_value))[3]
        }
}
```

Листинг 6.2. Программный код для анализа времени работы на языке Python.

```
import time
time_df = []

L_range = range(300, 3001, 300)
r_range = range(1, 51)

for r_value in r_range:
        cur_time = []
        for L_value in L_range:
                start_time = time.time()
                check_ssa(lod_series, L_value, r_value)
                check_time = time.time() - start_time
                cur_time.append(check_time)
time_df.append(cur_time)
```

В результате работы получена большая таблица данных, для более удобного визуального восприятия ниже приведены графики для параметров $L = 300, 1500, 3000$, где по оси x варьируется число r компонент для восстановления, а по оси y изображено время, затраченное на выполнение разложения и восстановления.

Рис. 6.1. График затраченного времени для L=300.

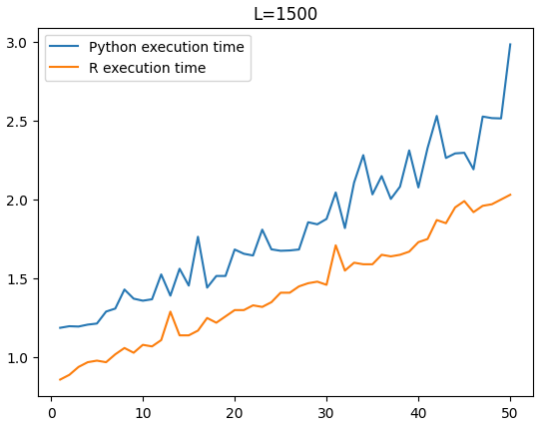

Рис. 6.2. График затраченного времени для L=1500.

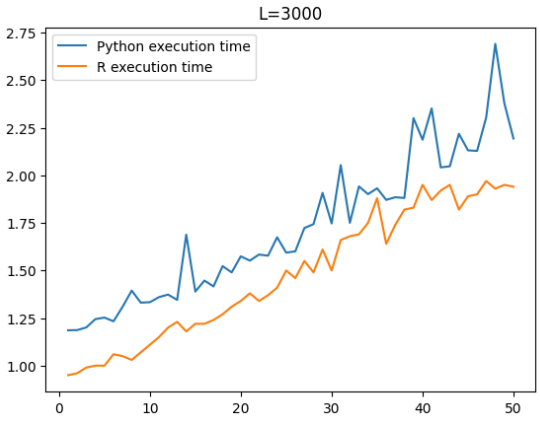

Рис. 6.3. График затраченного времени для L=3000.

Из данных графиков видно, что используемое в языке Python дополнительное время отличается на некоторый константный промежуток времени, который логично возникает из переводов типов данных между языками и дополнительных операций. В среднем данный промежуток составил 0.35 секунды, что в рамках данного ряда в процентном соотношении составляет порядка 15-20%. Таким образом, в дальнейшем стоит задача максимально сократить это время.

Глава 7

Заключение

В рамках данной работы были рассмотрены уже существующие реализации метода SSA для языка Python, изучены различные особенности пакета Rssa, а также реализована часть методов и примеров применения данного пакета.

Помимо адаптации модуля Rssa к языку Python, удалось применить модуль matplotlib для визуализации результатов, заменив тем самым встроенные в R средства отрисовки графиков, обращение к которым в связи с особенностями для пакета Rssa было осложнено.

Также с помощью модулей unittest и pandas были адаптированы тесты для проверки результатов функций восстановления и прогнозирования временных рядов, позволившие выявить и устранить часть существующих проблем.

Реализованный пакет опубликован на сайте проекта PyPi [14], текущая функциональность пакета позволяет проводить исследования временных рядов без привязки к средствам языка R со стороны пользователя.

С использованием полученных результатов, а также теории по базовому методу SSA, проанализирован и разложен на отдельные компоненты модельный ряд, изучены некоторые особенности, связанные с правильным выбором параметров для метода SSA, таких как длина окна и компоненты, составляющие тренд и сезонность временного ряда. Также были поанализированы временные ряды параметров вращения Земли, и на подобранных с помощью кросс-валидации параметрах построен прогноз на год вперед.

С учетом текущей скорости работы модуля pyrssa по сравнению с Rssa, все еще остается необходимым продолжать работу над оптимизацией модуля, а также расширять возможности использования встроенных в Python структур данных и наращивать функциональность визуализации результатов работы.

Список литературы

[1] Голяндина Н.Э. *Метод «Гусеница»-SSA: анализ временных рядов: Учеб. пособие.* СПб: Изд-во СПбГУ, 2004.

[2] Nina Golyandina, Anton Korobeynikov и Anatoly Zhigljavsky. *Singular Spectrum Analysis with R.* Springer, 2018.

[3] *Rssa: A collection of methods for singular spectrum analysis.* [Электронный ресурс]. 2021. URL: `http://CRAN.R-project.org/package=Rssa` (дата обр. 07.01.2022).

[4] *Singular Spectrum Analysis, A Python Package for Time Series Classification Documentation.* [Электронный ресурс]. URL: `https://pyts.readthedocs.io/en/stable/auto_examples/decomposition/plot_ssa.html` (дата обр. 08.01.2022).

[5] *Introducing SSA for Time Series Decomposition.* [Электронный ресурс]. URL: `https://www.kaggle.com/jdarcy/introducing-ssa-for-time-series-decomposition` (дата обр. 27.11.2021).

[6] *Multivariate Singular Spectrum Analysis in Python.* [Электронный ресурс]. URL: `https://github.com/kieferk/pymssa` (дата обр. 27.11.2021).

[7] *R documentation.* [Электронный ресурс]. URL: `https://www.rdocumentation.org` (дата обр. 08.01.2022).

[8] *Python documentation.* [Электронный ресурс]. URL: `https://docs.python.org/3/` (дата обр. 08.01.2022).

[9] *Documentation for rpy2.* [Электронный ресурс]. URL: `https://rpy2.github.io/doc/latest/html/index.html` (дата обр. 08.01.2022).

[10] *Documentation for NumPy.* [Электронный ресурс]. 2022. URL: `https://numpy.org/doc/` (дата обр. 25.09.2022).

[11] *Documentation for pandas.* [Электронный ресурс]. 2022. URL: `https://pandas.pydata.org/docs/` (дата обр. 29.09.2022).

[12] *6 способов значительно ускорить pandas с помощью пары строк кода. Часть 1.* [Электронный ресурс]. URL: `https://habr.com/ru/post/503726/` (дата обр. 08.01.2022).

[13] *Documentation for matplotlib*. [Электронный ресурс]. URL: https://devdocs.io/matplotlib~3.1 (дата обр. 08.01.2022).

[14] *pyrssa package on PyPi*. [Электронный ресурс]. 2022. URL: https://pypi.org/project/pyrssa/ (дата обр. 30.09.2022).

[15] *International Earth rotation and reference systems service Earth orientation parameters EOP (IERS) 14 C04*. [Электронный ресурс]. 2022. URL: https://hpiers.obspm.fr/iers/eop/eopc04/eopc04_IAU2000.62-now (дата обр. 29.09.2022).

[16] *SSA EOP Forecast*. [Электронный ресурс]. 2022. URL: http://eoppredict.ru/ (дата обр. 30.09.2022).

[17] Голяндина Н.Э. Охотников Г.И. *Применение метода анализа сингулярного спектра*. СПб: Изд-во СПбГУ, 2017.

[18] *SSA EOP Forecast*. [Электронный ресурс]. 2022. URL: http://eoppredict.ru/ (дата обр. 30.09.2022).

[19] *unittest – Unit testing framework*. [Электронный ресурс]. 2023. URL: https://docs.python.org/3/library/unittest.html (дата обр. 08.03.2023).

Приложение А

Реализованные для языка Python функции

bforecast	Функция, реализующая алгоритм прогнозирования на основе бутстрэп-выборок.
grouping_auto_pgram	Функция для автоматической группировки компонент разложения на основе частотного анализа.
grouping_auto_wcor	Функция для автоматической группировки компонент разложения на основе взвешенных корреляций.
hmatr	Функция для построения матрицы неоднородности временного ряда.
iossa	Функция, реализующая итеративный алгоритм для улучшения разделимости компонент разложения.
parestimate	Функция для оценки параметров (частот и коэффициентов) набора собственных векторов SSA.
plot	Функция для отрисовки различных графиков, связанных с методом SSA.
reconstruct	Функция для восстановления ряда на основе сгруппированных элементарных компонент.
rforecast	Функция, реализующая алгоритм рекуррентного прогнозирования.
ssa	Функция для выполнения различных видов разложения SSA.
vforecast	Функция, реализующая алгоритм векторного прогнозирования.
wcor	Функция для нахождения взвешенных корреляций собственных векторов SSA.

Таблица А.1. Список функций пакета Rssa, реализованных в pyrssa.

Приложение Б

Таблицы времени работы на R и Python

L \ r	10	20	30	40	50
300	2.33	2.51	2.77	3.03	3.28
900	0.95	1.24	1.44	1.65	2.03
1500	1.08	1.30	1.46	1.73	2.03
2200	1.12	1.40	1.55	1.75	2.01
3000	1.11	1.34	1.50	1.95	1.94

Таблица Б.1. Время работы разложения SSA и восстановления временного ряда ПВЗ на языке R в зависимости от параметров L, r.

L \ r	10	20	30	40	50
300	2.80	3.28	3.19	3.56	3.93
900	1.45	1.65	1.92	2.08	2.36
1500	1.35	1.68	1.87	2.08	2.98
2200	1.43	1.70	1.92	2.32	2.52
3000	1.33	1.57	1.74	2.18	2.19

Таблица Б.2. Время работы разложения SSA и восстановления временного ряда ПВЗ на языке Python в зависимости от параметров L, r.