МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ОТЧЕТ

по лабораторной работе

по дисциплине

Методы и средства обработки сигналов

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Авербух М. Л.

(подпись) (фамилия, и.,о.)

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Халеев А.А.

(подпись) (фамилия, и.,о.)

\_\_\_21-ВМз\_\_\_\_\_\_\_\_

(шифр группы)

Работа защищена «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_

С оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2023

**Задание**

Имеем функцию

1. Пользователем задаются:

a1, b1, a2, b2, a3, b3, x0 (начальное значение), xк (конечное значение), Δx (шаг).

1. Расчет y(x) по заданным значениям a1, b1, a2, b2, a3, b3, x0 (начальное значение), xк (конечное значение), Δx (шаг).
2. Отображение векторов x и y (в виде таблицы) с возможностью редактирования.
3. Построение графика y(x) по указанным векторам.
4. Наложение случайного шума на сигнал (отображение на графике)
5. Восстановление искаженного сигнала (отображение на графике)

**Проектное решение**

Задание выполнено на языке Python, с реализацией графического пользовательского интерфейса для наглядности результатов, а также для упрощения анализа результатов работы.

***Функциональные возможности разрабатываемого приложения:***

1. Построение графиков функции, искаженного и восстановленного отображения функции
2. Пользовательская регулировка уровня шума
3. Выбор фильтра для восстановления функции
4. Изменение параметров функции, предельных значений построения, шага дискретизации
5. Управление масштабом графика по оси «Y» в случае выхода графика за границы видимой области
6. Сохранение графика в виде изображения
7. Увеличение части графика, перемещение по графику
8. Настройка отображения осей

***Зависимости / используемые библиотеки / инструменты приложения:***

1. **NumPy (numpy):** Библиотека для работы с массивами и математических операций. Используется для генерации данных и операций с массивами.
2. **Pandas (pandas):** Библиотека для анализа и манипуляции данными. Применяется в классах **Exponential** и **MovingAverage** для работы со скользящим средним и экспоненциальным сглаживанием.
3. **Matplotlib (matplotlib):** Библиотека для создания статических, интерактивных и анимированных визуализаций в Python. В коде используется для построения графиков.
4. **SciPy (scipy):** Библиотека, используемая для научных и технических вычислений. В коде импортируются функции **medfilt** для медианного сглаживания и **gaussian\_filter** для гауссовского сглаживания.
5. **PySide6 (PySide6):** Библиотека Python, предоставляющая доступ к Qt библиотекам для создания графического интерфейса пользователя.
6. **Matplotlib Qt backend (matplotlib.backends.backend\_qt5agg):** Используется для интеграции визуализаций Matplotlib с графическим интерфейсом пользователя Qt.

Эти библиотеки и инструменты вместе образуют основу для разработки приложения, которое может обрабатывать и визуализировать данные, предоставлять пользовательский интерфейс и применять различные методы сглаживания данных.

**Ход выполнения**

***Построение графиков***

Класс **PlotLayout** наследуется от **QVBoxLayout** и **QWidget**, и организует построение графиков в несколько этапов:

1. **Инициализация:**
   * Конструктор принимает параметры сигнала **a1**, **a2**, **a3**, **b1**, **b2**, **b3**, начальное и конечное значения **x0**, **xk** и шаг **dx**.
   * Создается словарь **self.params**, содержащий эти параметры для дальнейшего использования при построении графиков.
   * Создается холст **MplCanvas** для рисования графиков с использованием **matplotlib**.
   * Устанавливается начальный уровень шума **self.noise\_lvl** и стратегия сглаживания **self.sm\_strategy**.
2. **Сигналы:**
   * Определены сигналы **noise\_lvl\_changed**, **strategy\_changed**, и **params\_changed**, которые связаны с методом **draw\_plot**. Эти сигналы эмитируются, когда соответствующие параметры изменяются.
3. **Построение графика (draw\_plot):**
   * Метод **draw\_plot** вызывается для инициализации графика и в ответ на изменения параметров.
   * Очищает текущий график перед новой отрисовкой.
   * Рассчитывает значения **x** и **y** для оригинального сигнала на основе текущих параметров и добавляет шум к **y**, создавая зашумленный сигнал **y\_noisy**.
   * Применяет выбранную стратегию сглаживания к зашумленному сигналу, получая **y\_smooth**.
   * Рисует на графике оригинальный сигнал, зашумленный сигнал и восстановленный сигнал.
   * Добавляет легенду, заголовок и фиксирует ось Y с учетом масштаба **self.y\_scale**.
   * Перерисовывает график.
4. **Изменение уровня шума (change\_noise\_lvl):**
   * Метод **change\_noise\_lvl** изменяет **self.noise\_lvl** и эмитирует сигнал **noise\_lvl\_changed**.
5. **Изменение стратегии сглаживания (change\_strategy):**
   * Метод **change\_strategy** изменяет объект **self.sm\_strategy** в зависимости от переданного названия стратегии и эмитирует сигнал **strategy\_changed**.
6. **Изменение параметров (change\_params):**
   * Метод **change\_params** обновляет словарь **self.params** новыми значениями и эмитирует сигнал **params\_changed**.
7. **Изменение масштаба по оси Y (y\_scale\_inc и y\_scale\_dec):**
   * Методы **y\_scale\_inc** и **y\_scale\_dec** используются для увеличения и уменьшения значения **self.y\_scale**, которое влияет на масштаб оси Y графика.
   * При увеличении масштаба (**y\_scale\_inc**), **self.y\_scale** увеличивается на 1, и эмитируется сигнал **params\_changed**, что вызывает перерисовку графика.
   * При уменьшении масштаба (**y\_scale\_dec**), **self.y\_scale** уменьшается на 1, но не меньше нуля, чтобы избежать отрицательного масштаба. Если **self.y\_scale** уже равен 0, уменьшение не происходит. Также, после уменьшения масштаба, эмитируется сигнал **params\_changed** для обновления графика.

Класс **PlotLayout** представляет собой комплексную систему для построения и обновления графиков в зависимости от пользовательских вводов и взаимодействий. Он интегрирует **matplotlib** для рисования графиков и использует сигнально-слотовый механизм Qt для обновления графиков в ответ на изменения параметров.

***Наложение шума***

Наложение шума в классе **PlotLayout** реализовано в методе **draw\_plot**. Вот как это происходит:

1. **Генерация Х и У:**
   * Сначала создается массив **x** с использованием функции **np.arange**, которая генерирует последовательность чисел от **self.params['x0']** до **self.params['xk']** с шагом **self.params['dx']**.
   * Затем вычисляется соответствующий массив **y** путем подставления значений **x** в формулу оригинального сигнала.
2. **Генерация шума:**
   * Шум генерируется с помощью функции **np.random.normal**, которая создает массив случайных значений, распределенных по нормальному (гауссову) распределению с математическим ожиданием **0** и стандартным отклонением **self.noise\_lvl**, причем размер массива равен длине массива **y**.
3. **Наложение шума на сигнал:**
   * Сгенерированный шумовой массив **noise** складывается с массивом **y**, создавая зашумленный сигнал **y\_noisy**.

Этот зашумленный сигнал **y\_noisy** затем отображается на графике в методе **draw\_plot** вместе с оригинальным сигналом и сигналом после сглаживания, демонстрируя эффект шума на данные.

***Используемые фильтры***

В коде реализованы четыре стратегии сглаживания шумных данных:

1. **Экспоненциальное сглаживание (Exponential):** Экспоненциальное сглаживание придает более новым данным больший вес, что позволяет отслеживать тренды более чутко к недавним изменениям. В этом классе используется метод **ewm** (Exponential Weighted functions) из библиотеки pandas с параметром **span**, который определяет скорость убывания весов. Чем меньше значение **span**, тем больший вес имеют последние значения. Возвращаемое значение - сглаженный набор данных.
2. **Гауссово сглаживание (Gaussian):** Гауссово сглаживание использует гауссовский фильтр, применяя свертку шумных данных с гауссовой функцией. Параметр **sigma** контролирует степень сглаживания; большие значения **sigma** приводят к более сильному сглаживанию. Гауссово сглаживание эффективно сглаживает шум и сохраняет края.
3. **Медианное сглаживание (Median):** Медианное сглаживание - это нелинейный фильтр, который заменяет каждый элемент массива медианой элементов в окне заданного размера (**kernel\_size**). Этот метод хорошо справляется с сигналами, которые содержат выбросы или импульсный шум, поскольку медиана устойчива к таким выбросам.
4. **Скользящее среднее (Moving Average):** Скользящее среднее сглаживает временные ряды путем замены каждого значения средним значением в окне фиксированного размера (**window\_size**). Это простой и часто используемый метод сглаживания, который уменьшает временную вариабельность и выделяет долгосрочные тенденции.

Каждый из этих фильтров реализуется в виде класса, который наследует от абстрактного базового класса **SmoothStrategy**, и каждый класс предоставляет реализацию метода **get\_y**, который принимает зашумленный сигнал **y\_noisy** и возвращает сглаженный сигнал.

**Приложение 1**

**Программный код**

***main.py:***

from sys import argv  
  
from PySide6.QtGui import QIcon  
from PySide6.QtWidgets import QApplication  
  
from source.app import MainWindow  
  
  
def main():  
 app = QApplication(argv)  
 window = MainWindow()  
 window.setWindowIcon(QIcon("designed\_ui/icons/logo.png"))  
 window.show()  
 app.exec()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

***app.py:***

import os  
from enum import IntEnum  
from fnmatch import fnmatch  
  
from PySide6.QtCore import Signal  
from PySide6.QtWebEngineWidgets import QWebEngineView  
from PySide6.QtWidgets import QInputDialog, QMainWindow, QStackedLayout, QTextEdit  
  
from designed\_ui.designed\_interface import Ui\_MainWindow  
from source.html\_utils import HtmlAdapter, HtmlWidget  
  
  
class MainWindow(QMainWindow, Ui\_MainWindow, HtmlWidget):  
 temp\_saved = Signal()  
 temp\_loaded = Signal()  
 temp\_generated = Signal()  
  
 class Mode(IntEnum):  
 TEXT = 0  
 HTML = 1  
  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 self.current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))  
  
 self.update\_templates()  
 self.text\_edit = QTextEdit()  
 self.html\_render = QWebEngineView()  
 self.html\_render.reload()  
 self.text\_lay = QStackedLayout()  
 self.text\_lay.setStackingMode(QStackedLayout.StackOne)  
 self.text\_lay.addWidget(self.text\_edit)  
 self.text\_lay.addWidget(self.html\_render)  
 self.text\_view.setLayout(self.text\_lay)  
 *# ---------------------------- connections------------------------------------* self.text\_btn.clicked.connect(self.show\_text)  
 self.render\_btn.clicked.connect(self.show\_html)  
 self.generate\_btn.clicked.connect(self.generate)  
 self.clear\_btn.clicked.connect(self.text\_edit.clear)  
 self.clear\_btn.clicked.connect(self.show\_text)  
 self.save\_btn.clicked.connect(self.save\_modal)  
 self.load\_btn.clicked.connect(self.load\_template)  
 self.temp\_saved.connect(self.update\_templates)  
 self.temp\_loaded.connect(self.show\_text)  
 self.temp\_generated.connect(self.show\_text)  
 *# ----------------------------------------------------------------------------* @property  
 def sections(self) -> int:  
 return self.sections\_spin.value()  
  
 @property  
 def divs(self) -> int:  
 return self.divs\_spin.value()  
  
 @property  
 def bordered(self) -> bool:  
 return self.bordered\_check.isChecked()  
  
 @property  
 def color(self) -> str:  
 return self.text\_color\_spin.currentText()  
  
 @property  
 def alignment(self) -> str:  
 return self.alignment\_spin.currentText()  
  
 @property  
 def headers(self) -> bool:  
 return self.headers\_check.isChecked()

def generate(self) -> None:  
 html\_agent = HtmlAdapter()  
 html\_agent.build\_page(self)  
 html\_text = html\_agent.get\_html()  
 self.text\_edit.setPlainText(html\_text)  
 self.temp\_generated.emit()  
  
 def show\_text(self) -> None:  
 self.text\_lay.setCurrentIndex(self.Mode.TEXT)

def show\_html(self) -> None:  
 self.html\_render.setHtml(self.text\_edit.toPlainText())  
 self.text\_lay.setCurrentIndex(self.Mode.HTML)  
  
 def save\_modal(self) -> None:  
 modal = QInputDialog()  
 modal.setWindowTitle("HTML template saving")  
 modal.setLabelText("Enter filename:")  
 modal.exec()  
 if modal.accepted:  
 self.save\_template(modal.textValue())  
  
 def save\_template(self, filename) -> None:  
 with open(f"templates/{filename.rstrip('.html')}.html", "w") as output:  
 output.write(self.text\_edit.toPlainText())  
 self.temp\_saved.emit()  
  
 def update\_templates(self) -> None:  
 self.templates.clear()  
 self.templates.addItems(self.get\_templates())  
  
 @staticmethod  
 def get\_templates() -> tuple[str, ...]:  
 return tuple(entry for entry in os.listdir("templates")

if fnmatch(entry, "\*.html"))  
  
 def load\_template(self) -> None:  
 template\_name = str(self.templates.currentText())  
 with open(f"templates/{template\_name}", "r") as temp:  
 template\_text = temp.read()  
 self.text\_edit.setPlainText(template\_text)  
 self.temp\_loaded.emit()

***html\_utils.py:***

from abc import ABC, ABCMeta, abstractmethod  
from collections import deque  
from dataclasses import dataclass  
from typing import Any, Union  
  
from PySide6.QtCore import QObject  
  
from source.html\_tags import HtmlTag, DoubleTag, SingleTag, UniqueTag, TagContent, HTML\_SINGLES, \ HTML\_DOUBLES, HTML\_UNIQUES  
  
  
class \_ABCQObjectMeta(type(QObject), ABCMeta): ...  
  
  
class HtmlWidget(QObject, ABC, metaclass=\_ABCQObjectMeta):  
 @property  
 @abstractmethod  
 def sections(self) -> int: ...  
  
 @property  
 @abstractmethod  
 def divs(self) -> int: ...  
  
 @property  
 @abstractmethod  
 def bordered(self) -> bool: ...  
  
 @property  
 @abstractmethod  
 def headers(self) -> bool: ...  
  
 @property  
 @abstractmethod  
 def color(self) -> str: ...  
  
 @property  
 @abstractmethod  
 def alignment(self) -> str: ...  
  
  
@dataclass  
class Style:  
 name: str  
 body: str  
  
  
class HtmlAdapter:  
 *""" This class is used to implement the "adapter" pattern"""* def \_\_init\_\_(self) -> None:  
 self.director = HtmlDirector()  
  
 def build\_page(self, obj: HtmlWidget) -> None:  
 style = self.create\_style(color=obj.color, alignment=obj.alignment, bordered=obj.bordered)  
 self.director.build\_tree(sections\_num=obj.sections, divs\_num=obj.divs, div\_style=style,  
 headers=obj.headers)  
  
 def get\_html(self) -> str:  
 return self.director.get\_html()  
  
 @staticmethod  
 def create\_style(\*, color: str, alignment: str, bordered: bool, name: str = "container") -> "Style":  
 style = f".{name} {{"  
 if color:  
 style += f"color: {color}; "  
 if alignment:  
 style += f"text-align: {alignment}; "  
 if bordered:  
 style += "border: 1px solid black; "  
 style += "}"  
 return Style(name, style)

class Strategy(ABC):  
 *""" This class is a head of hierarchy, that provide to implement a part of the strategy pattern"""* @staticmethod  
 @abstractmethod  
 def add(node: list[Any], value: HtmlTag) -> list[Any]:  
 pass  
  
  
class Node(Strategy):  
 @staticmethod  
 def add(node: list[Any], value: HtmlTag) -> list[Any]:  
 if isinstance(value, DoubleTag):  
 new\_node = [value]  
 node.append(new\_node)  
 return new\_node  
 else:  
 return Leaf.add(node, value)  
  
  
class Leaf(Strategy):  
 @staticmethod  
 def add(node: list[Any], value: HtmlTag) -> list[Any]:  
 node.append(value)  
 return node  
  
  
class HtmlBuilder:  
 *""" This class is used to implement those patterns:  
 - factory method  
 - builder  
 - strategy (a part of)  
 - state  
 """* def \_\_init\_\_(self) -> None:  
 self.tree: list[Any] = list()  
 self.branch\_ptr: list[Any] = self.tree  
 self.node\_stack: deque = deque()  
  
 def add(self, value: str, \*, strategy: type[Strategy], specs: str = "") -> None:  
 *""" Implements a part of "strategy" pattern"""* self.node\_stack.append(self.branch\_ptr)  
 content = self.create\_content(value, specs) *# a part of "factory method" pattern* self.branch\_ptr = strategy.add(self.branch\_ptr, content)  
  
 @staticmethod  
 def create\_content(value: str, specs: str) -> HtmlTag:  
 *""" Implements "factory method" pattern"""* if value in HTML\_SINGLES:  
 return SingleTag(value, tag\_specs=specs)  
 elif value in HTML\_DOUBLES:  
 return DoubleTag(value, tag\_specs=specs)  
 elif value in HTML\_UNIQUES:  
 return UniqueTag(value, tag\_specs=specs)  
 else:  
 return TagContent(value)  
  
 def to\_previous(self) -> "HtmlBuilder":  
 *""" Implements "state" pattern - we can return to previous state using node callstack"""* if self.node\_stack:  
 last\_branch = self.node\_stack.pop()  
 if last\_branch is self.branch\_ptr:  
 self.to\_previous() *# recursive call* else:  
 self.branch\_ptr = last\_branch  
 if not self.node\_stack:  
 self.node\_stack.append(self.branch\_ptr)  
 return self  
  
 def get\_result(self):  
 return self.tree

class HtmlDirector:  
 def \_\_init\_\_(self) -> None:  
 self.html\_builder = HtmlBuilder()  
  
 def build\_tree(self, \*, sections\_num: int, divs\_num: int, div\_style: Style, headers: bool) -> None:  
 self.html\_builder.add("!DOCTYPE", strategy=Leaf, specs="html")  
 self.html\_builder.add("html", strategy=Node)  
 self.html\_builder.add("head", strategy=Node)  
 self.html\_builder.add("style", strategy=Node, specs='type="text/css"')  
 self.html\_builder.add(div\_style.body, strategy=Leaf)  
 self.html\_builder.to\_previous().to\_previous()  
 self.html\_builder.add("body", strategy=Node)  
 self.html\_builder.add("header", strategy=Leaf)  
 self.html\_builder.add("main", strategy=Node)  
 for s\_num in range(1, sections\_num + 1):  
 self.html\_builder.add("section", strategy=Node)  
 for d\_num in range(1, divs\_num + 1):  
 self.html\_builder.add("div", strategy=Node, specs=f"class={div\_style.name}")  
 if headers:  
 h\_level = d\_num if d\_num <= 6 else 6  
 self.html\_builder.add(f"h{h\_level}", strategy=Node)  
 self.html\_builder.add(f"section-{s\_num} div-{d\_num} message", strategy=Leaf)  
 self.html\_builder.to\_previous()  
 else:  
 self.html\_builder.add(f"section-{s\_num} div-{d\_num} message", strategy=Leaf)  
 self.html\_builder.to\_previous()  
 self.html\_builder.to\_previous()  
 self.html\_builder.to\_previous()  
 self.html\_builder.add("footer", strategy=Leaf)  
  
 def get\_html(self) -> str:  
 res = ""  
 space\_tab = " "  
  
 def tree\_traversal(node: list[Union[HtmlTag, Any]], level: int = -1) -> None:  
 *""" It's a function that traverses the html tree recursively to write result to nonlocal "res" variable """* nonlocal res  
 first\_tag = node[0]  
 res += space\_tab \* level + first\_tag.tag + '\n'  
 for child in node[1:]:  
 match child:  
 case SingleTag():  
 res += space\_tab \* (level + 1) + child.tag + '\n'  
 case DoubleTag():  
 res += space\_tab \* (level + 1) + child.tag + child.tag + '\n'  
 case list():  
 tree\_traversal(child, level + 1) *# recursive case* if isinstance(first\_tag, DoubleTag):  
 res += space\_tab \* level + first\_tag.tag + '\n'  
  
 tree\_traversal(self.html\_builder.tree)  
 return res

***html\_tags.py:***

from abc import ABC, abstractmethod  
from itertools import cycle, repeat  
from typing import Iterator  
  
HTML\_SINGLES = {'!DOCTYPE', 'area', 'base', 'br', 'col', 'command', 'embed', 'hr', 'img', 'input',   
 'keygen', 'link', 'meta', 'param', 'source', 'track', 'wbr'}  
  
HTML\_DOUBLES = {"a", "abbr", "address", "article", "aside", "audio", "b", "bdi", "bdo",   
 "blockquote", "button", "canvas", "caption", "cite", "code", "data", "datalist",   
 "dd", "del", "details", "dfn", "div", "dl", "dt", "em", "fieldset", "figcaption",   
 "figure", "footer", "form", "h1", "h2", "h3", "h4", "h5", "h6", "hgroup", "i",   
 "iframe", "ins", "kbd", "label", "legend", "li", "map", "mark", "menu",   
 "menuitem", "meter", "nav", "noscript", "object", "ol", "optgroup", "option",   
 "output", "p", "pre", "progress", "q", "rp", "rt", "ruby", "s", "samp", "script",   
 "section", "select", "small", "span", "strong", "style", "sub", "summary", "sup",

"table", "tbody", "td", "textarea", "tfoot", "th", "thead", "time", "title", "tr",

"u", "ul", "var", "video"}  
  
HTML\_UNIQUES = {"html", "head", "body", "header", "main", "footer"}  
  
  
class HtmlTag(ABC):  
 def \_\_init\_\_(self, tag\_name: str, \*, tag\_specs: str = ""):  
 self.\_tag\_name = tag\_name  
 self.\_tag\_specs = tag\_specs  
 self.\_tag\_iter = self.\_get\_tag\_gen()  
  
 def \_\_repr\_\_(self):  
 return f"{type(self).\_\_name\_\_}({self.\_tag\_name})"  
  
 @abstractmethod  
 def \_get\_tag\_gen(self) -> Iterator: ...  
  
 @property  
 def tag(self) -> str:  
 return next(self.\_tag\_iter)  
  
  
class SingleTag(HtmlTag):  
 def \_get\_tag\_gen(self) -> Iterator:  
 return repeat(f"<{self.\_tag\_name}{' ' \* bool(self.\_tag\_specs)}{self.\_tag\_specs}>")  
  
  
class DoubleTag(HtmlTag):  
 def \_get\_tag\_gen(self) -> Iterator:  
 return cycle((f"<{self.\_tag\_name}{' ' \* bool(self.\_tag\_specs)}{self.\_tag\_specs}>",

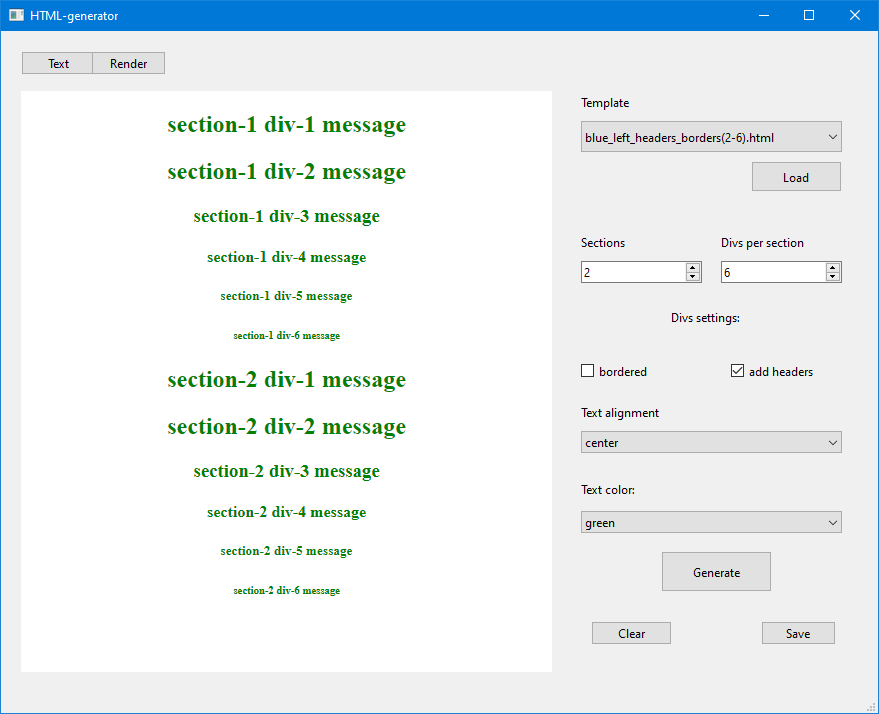
f"</{self.\_tag\_name}>"))  
  
  
*# flyweight*class UniqueTag(DoubleTag):  
 *""" This class is used to implement "flyweight" pattern """* \_\_instances: dict[str, HtmlTag] = dict()  
  
 def \_\_new\_\_(cls, tag\_name: str, \*args, \*\*kwargs):  
 return cls.\_\_instances.setdefault(tag\_name, super().\_\_new\_\_(cls))  
  
  
class TagContent(SingleTag):  
  
 def \_get\_tag\_gen(self) -> Iterator:  
 return repeat(self.\_tag\_name)

**Приложение 2**

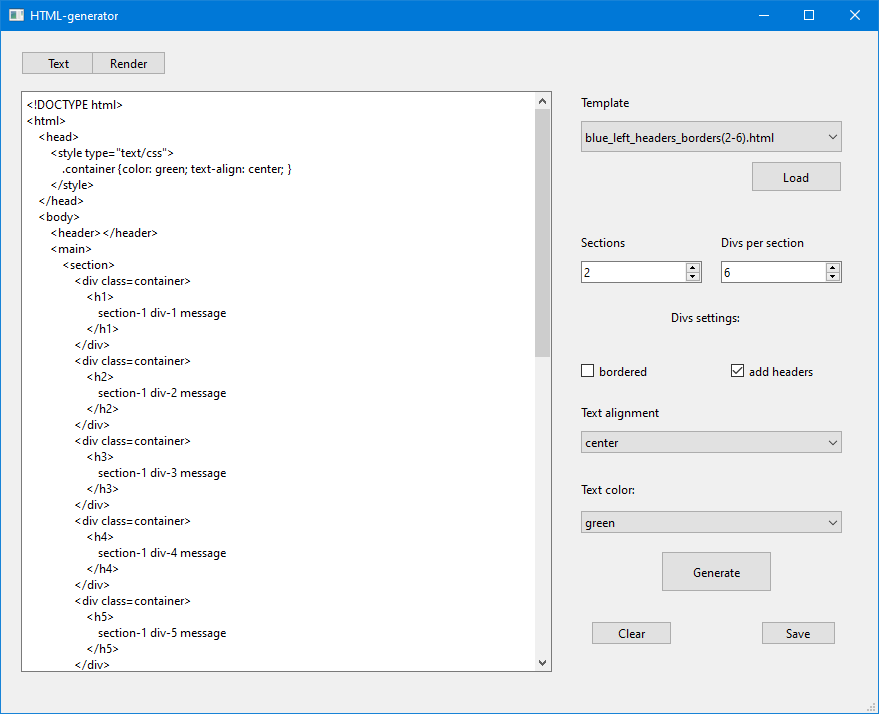
**Результаты тестирования**

*Генерация шаблона без рамки, с разноуровневыми заголовками зеленого цвета, по центру*

*Окно “Render”:*

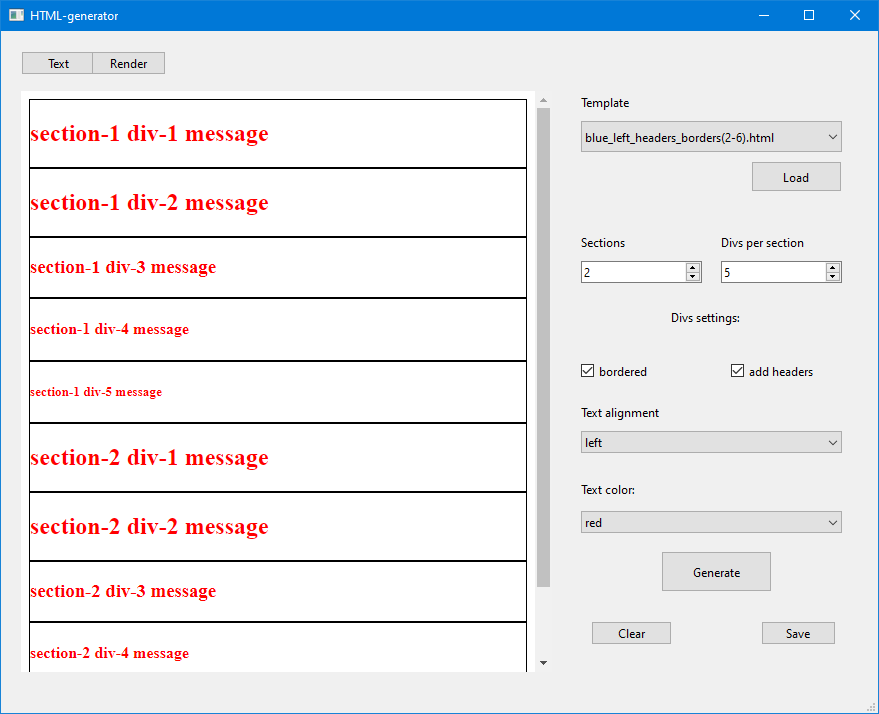
****

*Окно “Text”:*

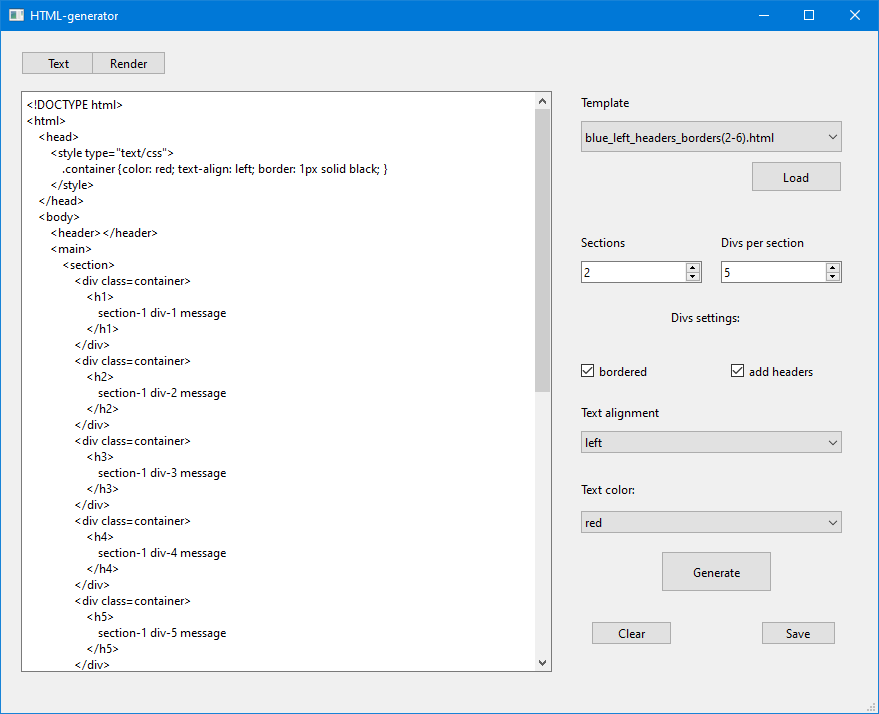
****

*Генерация шаблона c рамками, с разноуровневыми заголовками красного цвета, слева*

*Окно “Render”:*

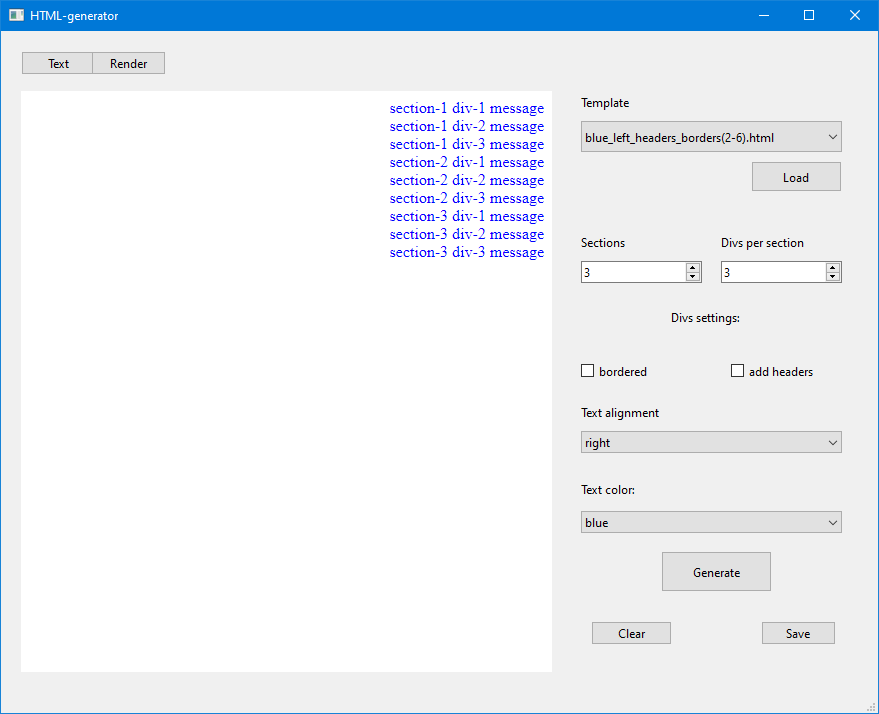
****

*Окно “Text”:*

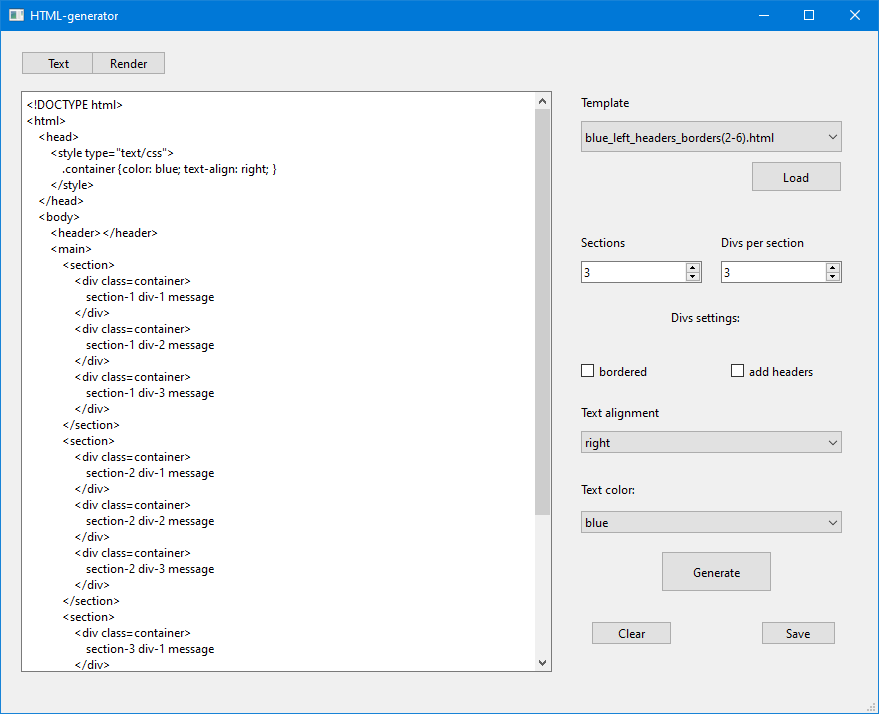
****

*Генерация шаблона без рамок, без заголовков, с текстом синего цвета, справа*

*Окно “Render”:*

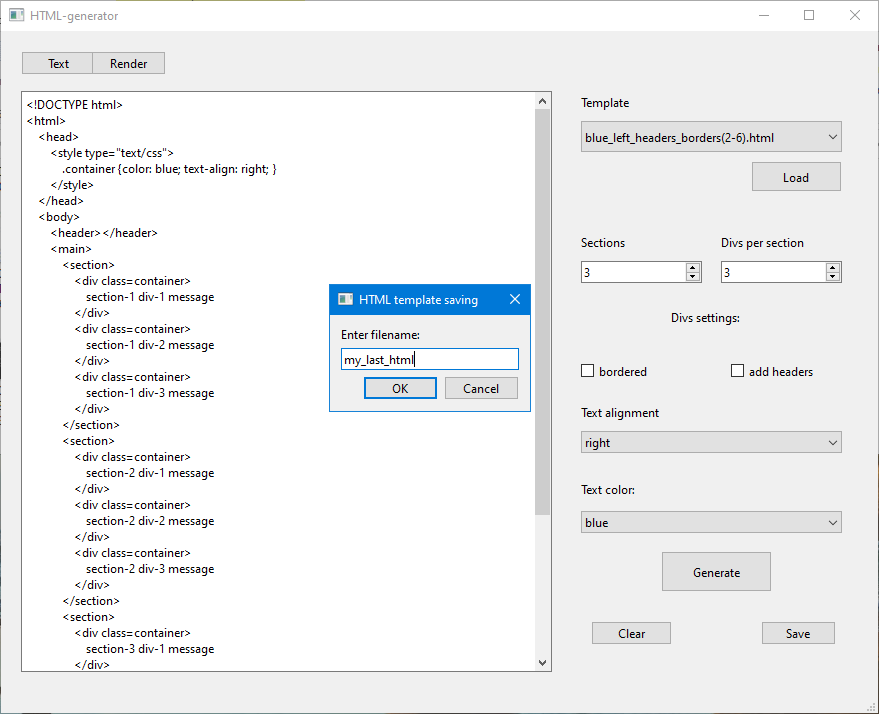
****

*Окно “Text”:*

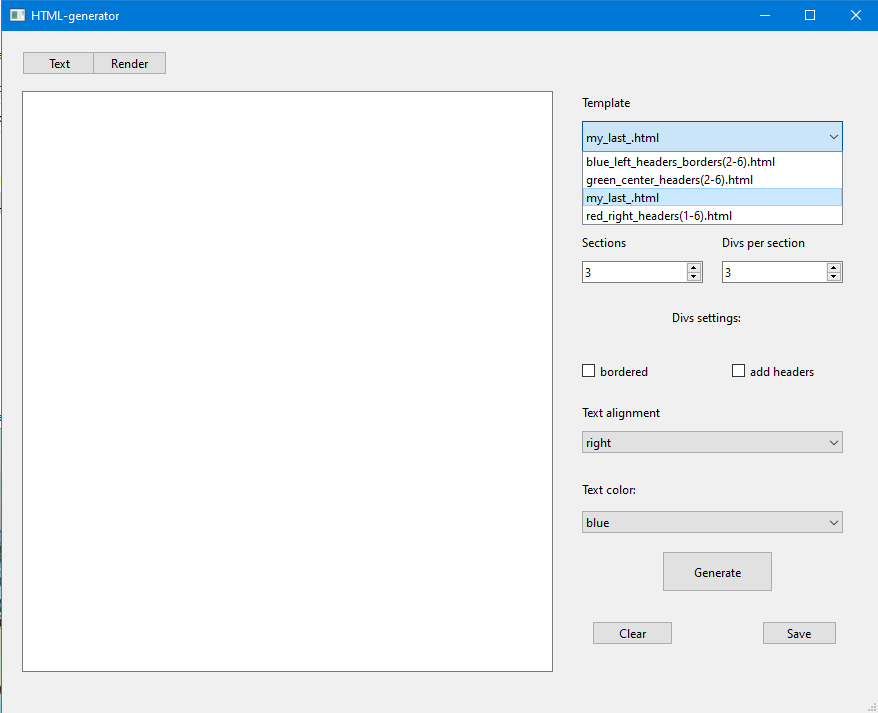
****

*Проверка работоспособности элементов сохранения и загрузки шаблонов*

*Результат нажатия кнопки “Save”:*

****

*Результат нажатия кнопки “Load”:*

****

**Вывод:**

Разработанное приложение работает без ошибок, полностью выполняет задуманную функциональность. С помощью программы можно генерировать шаблоны разного уровня вложенности, с возможностью добавления заголовков убывающего уровня, выбором цвета текста а также позиции на странице. Также есть возможность очистки окна ввода html-кода, загрузки и сохранения шаблонов.

Использование шаблонов проектирования в проекте позволило улучшить читаемость кода, повысить производительность, гибкость системы, а так же сделало ее расширяемой, позволило снизить межмодульную связность, что положительно сказалось на возможности тестирования системы, ее использования и доработки.