РАЗДЕЛ I
ГИС В НАУЧНЫХ И ПРИКЛАДНЫХ ИССЛЕДОВАНИЯХ

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ АВТОМАТИЗАЦИИ РАСЧЕТА ПЛОЩАДЕЙ ВОДНЫХ ОБЪЕКТОВ ПО ДАННЫМ СПЕКТРАЛЬНЫХ ИНДЕКСОВ НА ОСНОВЕ ИНСТРУМЕНТОВ И БИБЛИОТЕК PYTHON

Ф. М. Андреев, Е. Н. Сутырина

Иркутский государственный университет, ул. К. Маркса, 1, 664003, г. Иркутск, Россия fmandreev@yandex.ru

В данной статье представлена разработка программного обеспечения для автоматизации процесса расчета площадей водных объектов на основе данных спектральных индексов. Используя язык программирования Python и его обширные библиотеки, разработано решение, позволяющее эффективно обрабатывать спутниковые изображения и извлекать необходимую информацию о водных поверхностях. В работе рассматриваются методы применения спектрального индекса *WRI (Water Ratio Index)* для обнаружения и классификации водных объектов и объектов суши. Описана архитектура созданного программного обеспечения, его основные модули и алгоритмы. Проведены экспериментальные исследования с использованием реальных данных дистанционного зондирования Земли на примере устьевой области дельты Верхней Ангары (В. Ангары) и о. Ярки, результаты которых подтвердили эффективность и практическую значимость разработанного инструмента.

Ключевые слова: спектральные индексы; Python; автоматизация расчета индексов; статистический анализ; разработка программного обеспечения; Верхняя Ангара; остров Ярки.

DEVELOPMENT OF SOFTWARE FOR AUTOMATING THE CALCULATION OF THE AREAS OF WATER BODIES BASED ON SPECTRAL INDEX DATA BASED ON PYTHON TOOLS AND LIBRARIES

F. M. Andreev, Y. N. Sutyrina

Irkutsk State University, K. Marx str., 1, 664003, Irkutsk, Russia fmandreev@yandex.ru

This article presents the development of software for automating the process of calculating the areas of water bodies based on spectral index data. Using the Python programming language and its extensive libraries, a solution has been developed that makes it possible to efficiently process satellite images and extract the necessary information about water surfaces. The paper discusses the methods of using the spectral index *WRI (Water Ratio Index)* for the detection and classification of water bodies and land objects. The architecture of the created software, its main modules and algorithms are described. Experimental studies

have been carried out using real data from remote sensing of the Earth on the example of the estuary area of the Upper Angara Delta and island of Yarky, the results of which confirmed the effectiveness and practical significance of the developed tool.

Keywords: spectral indexes; Python; automation of index calculation; statistical analysis; software development; Upper Angara River; island of Yarki.

Развитие технологий дистанционного зондирования Земли (ДЗЗ) и растущая доступность высокоточных спутниковых снимков открывают новые возможности для автоматизации процессов обнаружения и анализа водных объектов. Применение спектральных индексов (на примере данной работы – *WRI*) позволяет эффективно выделять водные поверхности на спутниковых изображениях, основываясь на уникальных характеристиках отражательной способности воды в различных спектральных диапазонах.

Индекс *WRI* основан на соотношении спектральных каналов, чувствительных к отражению воды и растительности, что делает его особенно эффективным при разделении водных и наземных объектов на снимках. При этом *WRI* демонстрирует высокую устойчивость к влиянию атмосферных условий и может применяться для обработки данных из различных источников спутниковой информации. Язык программирования Python, с его простым синтаксисом и обширной экосистемой библиотек для научных вычислений и обработки геопространственных данных, стал одним из ведущих инструментов в области геоинформационных технологий и анализа данных ДЗЗ. Библиотеки, такие как *Matplotlib*, *Pandas*, *Geopandas*, *Rasterio*, *NumPy* и *OpenCV*, предоставляют мощные средства для обработки, анализа и визуализации спутниковых снимков, что делает Python оптимальным выбором для разработки специализированного программного обеспечения в данной сфере [1-4].

Объектом исследования в данной работе является устьевая область реки Верхняя Ангара, а именно остров Ярки. Остров Ярки представляет особый интерес в контексте изучения динамики водных поверхностей из-за своего расположения и геоморфологических особенностей. На примере острова Ярки был разработан и протестирован программный скрипт, позволяющий автоматизировать процесс расчёта спектрального индекса *WRI* и определения площадей водных объектов.

Целью данной работы является создание программного обеспечения на основе инструментов и библиотек Python для автоматизации расчёта площадей водных объектов по данным спектрального индекса *WRI*. Разработанный скрипт способен автоматически перебирать заданные директории с данными спутниковых снимков, выполнять вычисление индекса *WRI* и проводить расчёт площадей водных поверхностей, что значительно упрощает и ускоряет процесс анализа.

В рамках исследования были решены следующие задачи:

1. Изучение особенностей применения спектрального индекса *WRI* для выделения водных поверхностей на спутниковых снимках.

2. Разработка алгоритма автоматизированного расчёта индекса *WRI* и определение площадей водных объектов.

3. Реализация программного скрипта на языке Python с использованием соответствующих библиотек.

4. Тестирование разработанного скрипта на данных острова Ярки и оценка его эффективности и точности.

Ожидается, что результаты данной работы будут способствовать улучшению методов мониторинга водных ресурсов и предоставят эффективный инструмент для специалистов в области гидрологии, экологии, геоинформатики и смежных дисциплин. Автоматизация процесса расчёта площадей водных объектов с использованием индекса *WRI* позволит оперативно получать актуальные данные, необходимые для принятия обоснованных решений в сфере управления водными ресурсами и экологического планирования.

Алгоритм работы. Начальный алгоритм (рис. 1) расчета выглядит следующим образом: Чтение входного растра *WRI* – Обрезка по маске – Векторизация растрового слоя по типу «вода» – Векторизация растрового слоя по типу «суша» – Выделение объектов, классифицированных как «вода» – Выделение объектов, классифицированных как «суша» – Сохранение выделенных объектов в отдельные слои – Исправление геометрий – Расчет площадных параметров с помощью калькулятора полей.

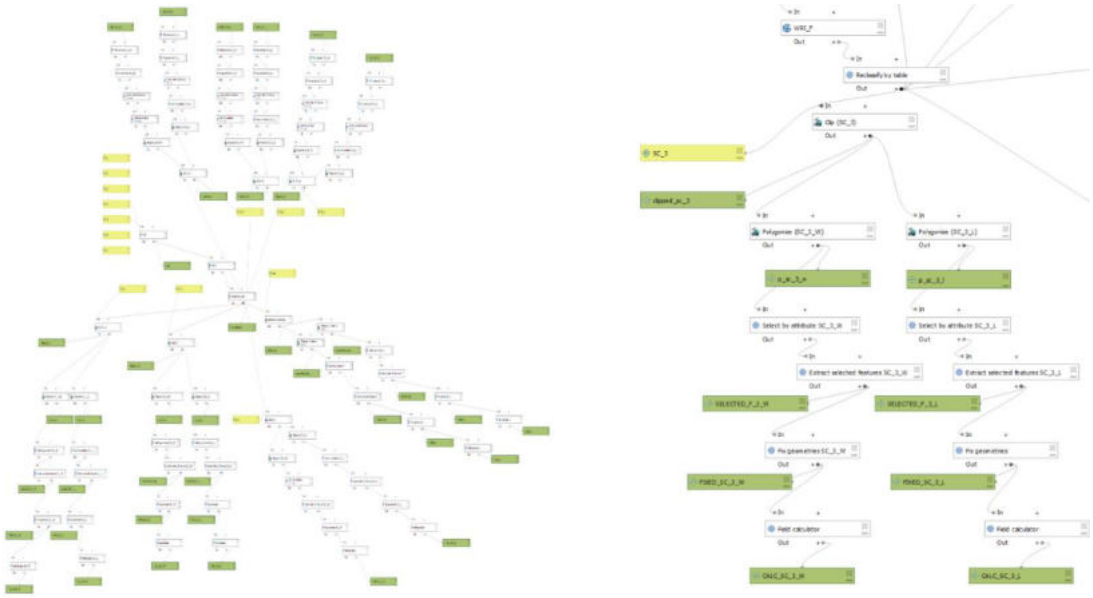

Рис. 1. Алгоритм геомодели (слева – полный алгоритм для расчета по каждой из масок, справа – алгоритм для отдельной области)

Описание работы скрипта. Словарь настройки каналов отдельных источников данных. Для решения задачи идентификации источника исходных данных (типа спутника) был составлен словарь с критериями идентификации источника снимка (рис. 2). Исходными данными выступают спутниковые снимки *Landsat 5, 7, 8, 9* (USGS Earthexplorer).

```
satellite_band_mappings = {
    'Landsat_5': {
        'Green': 2,
        'Red': 3,
        'NIR': 4,
        'SWIR': 5  # SWIR1
    },
    'Landsat_7': {
        'Green': 2,
        'Red': 3,
        'NIR': 4,
        'SWIR': 5  # SWIR1
    },
    'Landsat_8': {
        'Green': 3,
        'Red': 4,
        'NIR': 5,
        'SWIR': 6  # SWIR1
    },
    'Landsat_9': {
        'Green': 3,
        'Red': 4,
        'NIR': 5,
        'SWIR': 6  # SWIR1
    }
}
```

```
def detect_satellite(folder_path):
    """
    Функция для определения типа
спутника на основе имен файлов в
папке.
    Возвращает ключ из
satellite_band_mappings или None.
    """
    tif_files =
glob.glob(os.path.join(folder_path,
'*.tif')) +
glob.glob(os.path.join(folder_path,
'*.TIF'))
    for file in tif_files:
        filename =
os.path.basename(file)
        if 'LT05' in filename or
'LE05' in filename:
            return 'Landsat_5'
        elif 'LE07' in filename or
'LT07' in filename:
            return 'Landsat_7'
        elif 'LC08' in filename or
'LO08' in filename:
            return 'Landsat_8'
        elif 'LC09' in filename or
'LO09' in filename:
            return 'Landsat_9'
    return None  # Спутник не
определен
```

Рис. 2. Фрагмент кода – словарь с настройками каналов для каждого спутника (слева) и идентификации источника данных (справа)

Функция *detect_satellite* принимает путь к папке и осуществляет поиск всех файлов с расширениями «*.tif*» и «*.TIF*» внутри нее. Затем она извлекает имена этих файлов и анализирует их на наличие специфических подстрок, которые указывают на тип спутника серии Landsat. Если в имени файла присутствуют подстроки «*LT05*» или «*LE05*», функция идентифицирует спутник как «*Landsat_5*» и возвращает соответствующую строку. При обнаружении подстрок «*LE07*» или «*LT07*» она возвращает «*Landsat_7*». Если в имени файла содержатся подстроки «*LC08*» или «*LO08*», функция определяет спутник как «*Landsat_8*». Аналогично, при

наличии подстрок «*LC09*» или «*LO09*», возвращается «*Landsat_9*». Если ни одна из этих подстрок не обнаружена в именах файлов, функция возвращает значение «*None*», что означает, что тип спутника не был определен. Таким образом, эта функция помогает автоматически определить тип спутника на основе структуры имен файлов в заданной директории.

В представленном коде создаётся словарь под названием *satellite_band_mappings*, который служит для хранения соответствия между названиями спектральных каналов и их порядковыми номерами для разных спутников серии *Landsat*. Этот словарь структурирован таким образом, что для каждого ключа, обозначающего конкретный спутник (например, «*Landsat5*», «*Landsat7*», «*Landsat8*», «*Landsat9*»), хранится вложенный словарь с парами «название канала» и «номер канала». Для спутников *Landsat 5* и *Landsat 7* каналы «*Green*», «*Red*», «*NIR*» и «*SWIR*» соответствуют номерам 2, 3, 4 и 5 соответственно, где «*SWIR*» обозначает канал в коротковолновом инфракрасном диапазоне (*SWIR1*). Для спутников более нового поколения, *Landsat 8* и *Landsat 9*, те же каналы имеют немного иные порядковые номера: «*Green*» – 3, «*Red*» – 4, «*NIR*» – 5, «*SWIR*» – 6. Такая разница обусловлена изменениями в конфигурации датчиков на разных спутниках серии *Landsat*, где структура и нумерация спектральных каналов могут отличаться. Этот словарь обеспечивает программное соответствие и позволяет автоматически подбирать правильные номера каналов для обработки данных с разных спутников, что особенно полезно при разработке универсальных алгоритмов для анализа спутниковых изображений.

Далее необходимо выполнить загрузку маски, в контуре которой, в дальнейшем, будут выполняться определение границ объекта и расчет площади (рис. 3).

```
# Загрузка маски
print(mask_file)
mask_gdf = gpd.read_file(mask_file)
        mask_geometry = [mapping(mask_gdf.unary_union)]
```

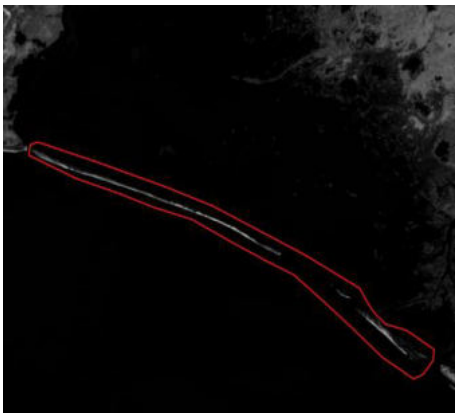

Рис. 3. Фрагмент кода – загрузка расчетной маски (слева) и положение маски на спутниковом снимке (справа)

В данном фрагменте кода выполняется загрузка и обработка геопространственной маски из файла: сначала с помощью команды *print(mask_file)* выводится путь или название файла маски, что позволяет проверить корректность указанного пути; затем функция *gpd.read_file(mask_file)* из библиотеки *GeoPandas* считывает файл маски и сохраняет его содержимое в переменную *mask_gdf* как *geo-DataFrame*, содержащий геометрические данные маски; после этого с помощью *mask_gdf.unary_union* происходит объединение всех отдельных геометрий из маски в одну единую геометрию, что актуально в случаях, когда маска состоит из нескольких полигонов или объектов; затем объединённая геометрия преобразуется в формат словаря с геометрическими координатами посредством функции *mapping()*, и результат помещается в список *mask_geometry*, позволяя использовать эту геометрию в дальнейших геопространственных операциях или анализе.

Обход всех подпапок в корневой папке. Далее будет запущен программный код, который выполнит полный обход всех подпапок, расположенных внутри корневой директории. Этот процесс позволит последовательно просмотреть и обработать содержимое каждой отдельной папки, обеспечивая тщательный анализ всей файловой структуры в корневом каталоге. Далее скрипт осуществляет перебор всех подпапок в указанной корневой директории и для каждой обнаруженной папки выполняет следующие шаги: выводит сообщение о начале обработки текущей папки, затем с помощью функции *detect_satellite* пытается определить, данные какого спутника содержатся в данной папке (рис. 4).

```
# Обход всех подпапок в корневой папке
for folder_name in os.listdir(root_folder):
    folder_path = os.path.join(root_folder, folder_name)
    if os.path.isdir(folder_path):
        print(f"nОбработка папки: {folder_name}")

        # Определяем спутник
        satellite = detect_satellite(folder_path)
        if satellite is None:
            print(f"  Не удалось определить спутник для папки {folder_name}. Пропуск.")
            continue
        else:
            print(f"  Определен спутник: {satellite}")

        # Получаем дату из имени папки или файла (требуется адаптация под ваши данные)
        date = folder_name  # Предполагается, что имя папки содержит дату

        # Получаем соответствие номеров каналов для текущего спутника
        band_mapping = satellite_band_mappings[satellite]
        bands_data = {}
        meta = None  # Инициализируем meta
```

Рис. 4. Фрагмент кода – обход всех поддиректорий в основной

Данная функция является одной из основных в программе. если определить спутник не удаётся, скрипт информирует об этом и пропускает текущую папку, переходя к следующей в цикле; если спутник успешно определён, выводится соответствующее сообщение; далее из имени папки извлекается дата, предполагая, что название папки содержит нужную информацию о дате снимков; после этого скрипт получает соответствие номеров спектральных каналов для определённого спутника из словаря *satellite_band_mappings*; затем инициализируется пустой словарь *bands_data* для хранения данных по каналам и переменная meta, изначально равная *None*, предназначенная для хранения метаданных снимков.

Поиск файлов с данными каналов. Следующим этапом запускается программный код, предназначенный для поиска файлов, содержащих данные о каналах. Этот процесс включает в себя рекурсивный обход всех каталогов и подкаталогов в заданной директории, с целью обнаружения и идентификации всех файлов, относящихся к данным каналов. Код последовательно проверяет каждый файл на соответствие определённым критериям (рис. 5).

```
for band_name, band_num in band_mapping.items():
        # Создаем шаблоны поиска файлов с
учетом различных форматов именования
        band_patterns = [
          f'*_B{band_num}.TIF',  # Например,
*_B3.TIF
          f'*_B0{band_num}.TIF', # Например,
*_B03.TIF
          f'*_B{band_num}.tif',
          f'*_B0{band_num}.tif',
          f'*_SR_B{band_num}.TIF',  # Например,
*_SR_B3.TIF
          f'*_SR_B0{band_num}.TIF',
          f'*_SR_B{band_num}.tif',
          f'*_SR_B0{band_num}.tif'
        ]

        band_files = []
        for pattern in band_patterns:
          search_pattern = os.path.join(folder_path,
pattern)

band_files.extend(glob.glob(search_pattern))
```

```
if band_files:
  band_file = band_files[0]  # Выбираем первый
найденный файл
  print(f"  Найден файл для канала {band_name}
(Band {band_num}):
{os.path.basename(band_file)}")
  # Открываем файл и считываем данные канала
  with rasterio.open(band_file) as src:
    bands_data[band_name] =
src.read(1).astype('float32')
    if meta is None:
      meta = src.meta.copy()  # Сохраняем
метаданные для будущего использования
else:
  print(f"  Файл для канала {band_name} (Band
{band_num}) не найден.")
          f'*_B{band_num}.tif',
          f'*_B0{band_num}.tif',
          f'*_SR_B{band_num}.TIF',  # Например,
*_SR_B3.TIF
          f'*_SR_B0{band_num}.TIF',
          f'*_SR_B{band_num}.tif',
          f'*_SR_B0{band_num}.tif'
        ]

        band_files = []
        for pattern in band_patterns:
          search_pattern = os.path.join(folder_path,
pattern)

band_files.extend(glob.glob(search_pattern))
```

Рис. 5. Фрагмент кода – поиск файлов в поддиректориях по заданным критериям

В данном фрагменте скрипта осуществляется поиск файлов с данными спектральных каналов. Для каждого канала из словаря *band_mapping* извлекаются его имя и номер. Затем создаётся список шаблонов *band_patterns*, учитывающий различные варианты именования файлов каналов – с разными регистрами расширений *(.TIF и .tif*), с наличием или отсутствием ведущего нуля в номере канала (например, *_B3.TIF* и *_B03.TIF*), а также с префиксом *_SR_*, который может обозначать обработанные данные отражательной способности поверхности. Инициализируется пустой список *band_files* для хранения найденных файлов. Далее скрипт перебирает каждый шаблон из band_patterns, соединяет его с путём к текущей папке *folder_path* и с помощью функции *glob.glob* выполняет поиск файлов, соответствующих данному шаблону. Найденные файлы добавляются в список *band_files*, собирая таким образом все возможные файлы данных для текущего спектрального канала для последующей обработки. В данном скрипте происходит последовательная обработка спектральных каналов спутниковых изображений: для каждого канала формируется список возможных шаблонов имён файлов, учитывающих различные варианты написания (например, с префиксом *«SR»*, с ведущим нулём в номере канала, разные регистры расширения *«.TIF»* или *«.tif»)*, после чего в заданной папке осуществляется поиск файлов, соответствующих этим шаблонам, с помощью функции *glob.glob*; если найден хотя бы один файл, то выбирается первый из них, выводится сообщение о его успешном обнаружении и начинается процесс его чтения с использованием библиотеки *rasterio* – файл открывается, данные канала считываются и приводятся к типу *float32* для дальнейшей обработки, а метаданные сохраняются при необходимости; если же файлы не были найдены, скрипт информирует об отсутствии файла для данного канала; таким образом, скрипт обеспечивает гибкий и надёжный способ сбора данных по каждому спектральному каналу, учитывая возможные вариации в именах файлов и гарантируя, что все необходимые данные будут корректно считаны и готовы для последующего анализа.

Проверка наличия всех необходимых каналов и расчет WRI. Далее скрипт выполняет проверку наличия всех необходимых спектральных каналов и расчёт индекса удержания воды (*WRI*) следующим образом: сначала он загружает спутниковое изображение с помощью библиотеки rasterio и извлекает метаданные растра, чтобы определить доступные каналы; затем формирует список необходимых каналов для расчёта *WRI*, который включает *B3* (зелёный), *B8* (ближний инфракрасный, *NIR*), *B11* (коротковолновый инфракрасный 1, *SWIR1*) и *B12* (коротковолновый инфракрасный 2, *SWIR2*); после этого код последовательно проверяет наличие каждого из этих каналов в данных растра, проходя по списку и сверяя с

доступными каналами, и если какой-либо канал отсутствует, выводит сообщение об ошибке и прекращает выполнение, так как без полного набора данных расчёт невозможен; при наличии всех необходимых каналов, код читает данные каждого канала и преобразует их в тип *float32* для точности вычислений; затем он рассчитывает числитель и знаменатель для формулы *WRI*, суммируя соответствующие массивы пикселей зелёного и *NIR* каналов для числителя, и *SWIR1* и *SWIR2* каналов для знаменателя, при этом обрабатывает возможные случаи деления на ноль, заменяя нулевые значения знаменателя на *NaN*, чтобы избежать ошибок; далее он выполняет поэлементное деление числителя на знаменатель, получая массив значений *WRI* для каждого пикселя изображения; после расчёта код может обработать аномальные значения, такие как бесконечности или *NaN*, и сохраняет полученный индекс *WRI* в новый растровый файл, используя исходный профиль данных для сохранения геопривязки и пространственных характеристик изображения (рис. 6).

```
# Проверяем, что все необходимые каналы были загружены
if all(band in bands_data for band in band_mapping):
    print("  Все необходимые каналы найдены. Выполняется расчет WRI.")

    # Извлекаем необходимые каналы
    green = bands_data['Green']
    red = bands_data['Red']
    nir = bands_data['NIR']
    swir = bands_data['SWIR']

    # Рассчитываем WRI
    numerator = green + red
    denominator = nir + swir
    # Для избежания деления на ноль
    denominator = np.where(denominator == 0, np.nan, denominator)
    wri = numerator / denominator
```

Рис. 6. Фрагмент кода – проверка наличия данных для расчета и расчет WRI

Данный код выполняет конвертацию реклассифицированного растра *WRI* в векторный слой для получения контура острова. Он извлекает геометрии из растра, где значение пикселей равно единице (области с высоким индексом удержания воды), и создает на их основе полигоны. Затем формируется *GeoDataFrame* с этими геометриями, затем все полигоны объединяются в один общий контур острова с помощью операции *unary_union*. Площадь острова вычисляется как разница между общей площадью области (31.929 м^2) и суммарной площадью выявленных водных объектов. Полученный контур острова сохраняется в виде векторного слоя в формате *Shapefile*. Далее информация о дате, спутнике и вычислен-

ной площади острова добавляется в *Excel*-таблицу и словарь для последующего построения графиков. В конце программа сохраняет обновленную *Excel*-таблицу по указанному пути (табл.).

Пример расчетного Excel-файла

Date	Area
11.06.1995	3,013
…	…
21.09.2023	1,780
29.09.2023	1,611
23.10.2023	1,572

На следующем этапе скрипт выполняет обрезку растра до расчетной области (рис. 7).

```
print("  Применение маски к WRI слою.")
            with rasterio.open(wri_output_filepath) as src:
                wri_masked, out_transform = mask.mask(src, mask_geometry,
crop=True)
                out_meta = src.meta.copy()
                # Обновляем метаданные
                out_meta.update({"driver": "GTiff",
                                 "height": wri_masked.shape[1],
                                 "width": wri_masked.shape[2],
                                 "transform": out_transform})

            wri_masked = wri_masked[0]  # Извлекаем единственный массив

            # Реклассификация WRI по порогу
            threshold = 1.0  # Задайте подходящее значение порога

            wri_reclassified = np.where(wri_masked > threshold, 1,
0).astype('uint8')

            # Сохранение WRI реклассифицированного изображения
            output_folder = os.path.join(folder_path, 'output')
            if not os.path.exists(output_folder):
                os.makedirs(output_folder)
            wri_reclassified_filepath = os.path.join(output_folder,
f"{folder_name}_WRI_reclassified.tif")

            out_meta.update(dtype=rasterio.uint8, count=1)
            with rasterio.open(wri_reclassified_filepath, 'w', **out_meta) as
dst:
                dst.write(wri_reclassified, 1)
            print(f"  Реклассифицированное WRI изображение сохранено по пути:
{wri_reclassified_filepath}")
```

Рис. 7. Фрагмент кода – Применение маски к WRI слою

Данный фрагмент кода выполняет применение географической маски к слою *WRI*, реклассификацию пикселей по заданному порогу и сохранение результирующего изображения. Сначала код открывает исходный растр *WRI*, применяет к нему маску на основе заданной геометрии *mask_geometry*,

обрезая растр по границам маски и обновляя метаданные (размеры, трансформации, драйвер). Затем извлекается единственный слой из массива данных *wri_masked*. После этого происходит реклассификация значений растра: пикселям с значениями выше порога (в данном случае 1.0) присваивается значение 1, остальным – 0, что создает бинарное изображение (рис. 8).

```
# Конвертация растра в вектор и получение контура острова
            print("  Конвертация растра в вектор для получения контура
острова.")
            shapes_generator = shapes(wri_reclassified,
transform=out_meta['transform'])
            geoms = [shape(geom) for geom, value in shapes_generator if
value == 1]

            if geoms:
                # Создаем GeoDataFrame
                gdf = gpd.GeoDataFrame({'geometry': geoms},
crs=mask_gdf.crs)
                # Объединяем полигоны в один (если необходимо)
                gdf = gpd.GeoDataFrame(gpd.GeoSeries(gdf.unary_union),
columns=['geometry'])
                # Расчет площади острова
                gdf['area'] = gdf['geometry'].area / 1e6  # Площадь в
квадратных километрах
                island_area = 31.929 - gdf['area'].sum()
                print(f"  Площадь острова: {island_area:.2f} кв.км")

                # Сохранение векторного слоя острова
                island_shp_filepath = os.path.join(output_folder,
f"{folder_name}_island.shp")
                gdf.to_file(island_shp_filepath)
                print(f"  Векторный слой острова сохранен по пути:
{island_shp_filepath}")

                # Добавление данных в Excel таблицу
                df_areas = df_areas.append({'Date': date, 'Satellite':
satellite, 'Area': island_area}, ignore_index=True)

                # Добавление данных в словарь для графика
                areas_dict[date] = island_area
            else:
                print("  Остров не обнаружен на изображении.")
        else:
            print("  Не все необходимые каналы были найдены в данной
папке для расчета WRI.")
```

Рис. 8. Фрагмент кода – конвертация растрового слоя в вектор и сохранение полигонального слоя объекта

Код проверяет наличие выходной папки *output*, при отсутствии создаёт её, и сохраняет реклассифицированное изображение в формате *GeoTIFF* по заданному пути, обновляя метаданные и указывая тип данных как uint8. В конце выводится сообщение с путем сохраненного файла.

Данный код выполняет конвертацию реклассифицированного растра *WRI* в векторный слой для получения контура острова. Он извлекает геометрии из растра, где значение пикселей равно единице (области с высоким индексом удержания воды), и создает на их основе полигоны. Затем формируется *GeoDataFrame* с этими геометриями, затем все полигоны объединяются в один общий контур острова с помощью операции *unary_union*.

Результаты и выводы. На данном этапе разработки реализованы решения по автоматизации функции расчета для неограниченного количества данных, их идентификации и обработки.

В результате работы над проектом разработан рабочий код скрипта, который станет основой для создания будущей полноценной программы для выполнения расчетов геометрических параметров исследуемых водных объектов и суши. В рамках дальнейшей работы планируется разработка полноценного GUI-интерфейса, реализация возможности создания векторной маски непосредственно в самом ПО, подгрузка дополнительных статистических параметров (расходов, уровней воды) в табличной форме для выполнения аналитики и визуализации.

Библиографические ссылки

1. Автоматизированная информационная система государственного мониторинга водных объектов (АИС ГМВО) [Электронный ресурс]. URL: https://gmvo.skniivh.ru/ (дата обращения: 25.10.2024).

2. Python [Electronic resource]. URL: https://www.python.org/ (date of access: 01.11.2024).

3. Project Jupyter [Electronic resource]. URL: https://jupyter.org/ (date of access: 01.11.2024).

4. USGS Earthexplorer [Electronic resource]. URL: https://earthexplorer.usgs.gov/ (date of access: 01.11.2024).