ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе №5
hard7: Триангуляция

Выполнил:
Бабич Александр Петрвоич
Группа №J312

Санкт-Петербург
2025

Содержание

1. Введение

Цель работы: разработка и тестирование алгоритма триангуляции для определения местоположения абонента на основе сигналов сотовых вышек.

Основные задачи:

1. Реализация поиска ближайших вышек в заданном радиусе
2. Расчёт расстояний с использованием формулы Гаверсинуса:

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{1}$$

3. Создание алгоритма трилатерации
4. Оценка точности с помощью метрик:
   - Абсолютная ошибка
   - Нормированная метрика:

$$\text{метрика} = \begin{cases} \frac{\text{ошибка}}{50}, & \text{если ошибка} \leq 50\text{м} \\ 0, & \text{иначе} \end{cases}$$

2. Теоретические аспекты

2.1. Преобразование координат

Географические координаты преобразуются в локальную декартову систему:

$$X = R \cdot (\lambda - \lambda_0) \cdot \cos\phi_0$$

$$Y = R \cdot (\phi - \phi_0)$$

где (ϕ_0, λ_0) - опорная точка.

2.2. Математическая модель

Система уравнений для трёх вышек:

$$\begin{cases} (x - x_1)^2 + (y - y_1)^2 = d_1^2 \\ (x - x_2)^2 + (y - y_2)^2 = d_2^2 \\ (x - x_3)^2 + (y - y_3)^2 = d_3^2 \end{cases}$$

Решение методом исключения переменных.

3. Реализация и красивые картинки

3.1. Функция haversine

Реализация расчёта расстояний с учётом кривизны Земли:

- Конвертация географических координат в радианы
- Применение формулы Харверсина:
- Расчёт расстояния через арксинус:
- Возврат расстояния в метрах (R = 6371 км)
- Листинг:

```python
def haversine(lat1, lon1, lat2, lon2):  3 usages  Alexander Babich
    """считаем расстояние в метрах между двумя точками (lat, lon)"""
    if any(coord is None for coord in [lat1, lon1, lat2, lon2]):
        return float('inf')  # координатов нет ну бан)

    # переводим всё в радианы (люблю профмат)
    r_lat1, r_lon1, r_lat2, r_lon2 = map(radians, [lat1, lon1, lat2, lon2])
    dlon, dlat = r_lon2 - r_lon1, r_lat2 - r_lat1
    # формула гаверсинуса(описана ниже)
    a = sin(dlat / 2) ** 2 + cos(r_lat1) * cos(r_lat2) * sin(dlon / 2) ** 2
    return R_EARTH_METERS * 2 * atan2(sqrt(a), sqrt(1 - a))
```

3.2. Класс CoordinateConverter

Преобразование между географическими и декартовыми координатами:

- Инициализация с опорной точкой:
- Преобразование в локальные координаты (X,Y):

$$X = R_{\text{earth}} \cdot (\lambda - \lambda_0) \cdot \cos \phi_0$$

- Обратное преобразование в широту/долготу:

$$\phi = \phi_0 + \frac{Y}{R_{\text{earth}}}$$

- Листинг:

```python
class CoordinateConverter:  1 usage  Alexander Babich
    def __init__(self, ref_lat, ref_lon):  Alexander Babich
        self.ref_lat_rad = radians(ref_lat)
        self.ref_lon_rad = radians(ref_lon)
        self.cos_ref_lat = cos(self.ref_lat_rad)

    def to_cartesian(self, lat, lon):  1 usage  Alexander Babich
        x = R_EARTH_METERS * (radians(lon) - self.ref_lon_rad) * self.cos_ref_lat
        y = R_EARTH_METERS * (radians(lat) - self.ref_lat_rad)
        return x, y

    def to_latlon(self, x, y):  1 usage  Alexander Babich
        if x is None or y is None: return None, None
        lat_rad = self.ref_lat_rad + y / R_EARTH_METERS
        if abs(self.cos_ref_lat) < 1e-9:
            lon_rad = self.ref_lon_rad
        else:
            lon_rad = self.ref_lon_rad + x / (R_EARTH_METERS * self.cos_ref_lat)
        return degrees(lat_rad), degrees(lon_rad)
```

3.3. Поиск ближайших вышек

Алгоритм работы функции find_nearby_stations:

- Расчёт расстояний до всех станций через haversine()
- Фильтрация станций в радиусе 400 метров
- Сортировка по расстоянию
- Листинг:

```python
def find_nearby_stations(  2 usages  github-actions[bot] +1
        lat: float,
        lon: float,
        stations: pd.DataFrame,
        R: float
) -> pd.DataFrame:
    """
    Найти ближайшие станции в радиусе R от точки
    """
    stations['distance_to_user'] = stations.apply(
        lambda row: haversine(lat, lon, row['lat'], row['lon']), axis=1
    )
    nearby = stations[stations['distance_to_user'] <= R].copy()
    return nearby
```

3.4. Алгоритм трилатерации

Реализация метода линейных наименьших квадратов:

- Построение системы уравнений для тройки вышек:

$$\begin{cases} 2(x_2 - x_1)x + 2(y_2 - y_1)y = d_1^2 - d_2^2 + x_2^2 - x_1^2 + y_2^2 - y_1^2 \\ 2(x_3 - x_1)x + 2(y_3 - y_1)y = d_1^2 - d_3^2 + x_3^2 - x_1^2 + y_3^2 - y_1^2 \end{cases}$$

- Медианное усреднение результатов

- Листинг:

```python
def trilaterate_3_stations_cartesian(s1_x, s1_y, d1, s2_x, s2_y, d2, s3_x, s3_y, d3):  1 usage  Alexander Babich
    A = 2 * (s2_x - s1_x);
    B = 2 * (s2_y - s1_y)
    C = d1 ** 2 - d2 ** 2 - s1_x ** 2 + s2_x ** 2 - s1_y ** 2 + s2_y ** 2
    D = 2 * (s3_x - s1_x);
    E = 2 * (s3_y - s1_y)
    F = d1 ** 2 - d3 ** 2 - s1_x ** 2 + s3_x ** 2 - s1_y ** 2 + s3_y ** 2
    denom = (A * E - B * D)
    if abs(denom) < 1e-9: return None, None
    return (C * E - F * B) / denom, (A * F - C * D) / denom
```

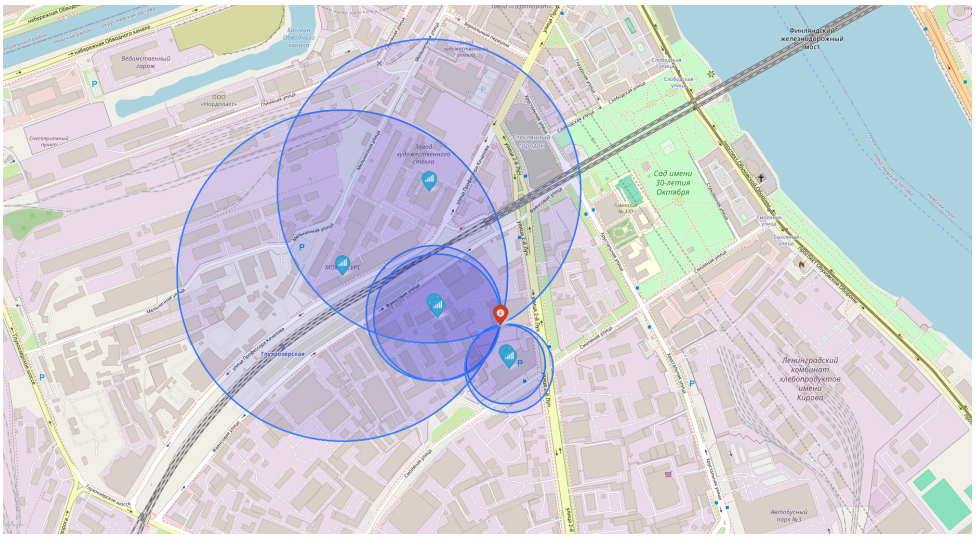
- Листинг Тригуляции:

```python
def triangulate(  1 usage  Alexander Babich +1
        stations: pd.DataFrame,
        distances: Union[np.ndarray, Dict[int, float], pd.DataFrame]
) -> Union[np.ndarray, List[float]]:
    """
    Решить задачу триангуляции по заданным координатам станции и измерениями до точки
    Возвращает предсказанные координаты
    """
    if len(stations) < 3 or len(stations) != len(distances):
        return None  # ну 3, потому что ТРИ(Е)угольник, ю ноу

    s_list_for_triang = []
    for i in range(len(stations)):
        s_list_for_triang.append({
            'x': stations.iloc[i]['x'], 'y': stations.iloc[i]['y'],
            'd': distances[i]
        })

    results_xy = []

    for s1, s2, s3 in combinations(s_list_for_triang, r: 3):
        pred_x, pred_y = trilaterate_3_stations_cartesian(
            s1['x'], s1['y'], s1['d'], s2['x'], s2['y'], s2['d'], s3['x'], s3['y'], s3['d']
        )
        if pred_x is not None:
            results_xy.append((pred_x, pred_y))

    if not results_xy: return None

    all_x = [res[0] for res in results_xy]
    all_y = [res[1] for res in results_xy]
    return [np.median(all_x), np.median(all_y)]  # возвращаем усредненные координаты
```

3.5. Визуализация результатов

Интерактивные карты с использованием Folium:

- Цветовая схема маркеров:
  - Зелёный - реальное положение
  - Красный - предсказанное положение
  - Синий(фиолетовый) - базовые станции
- Пример визуализации:



4. Сложности

Формула Харверсина опять эти ваши вычислительные математики. И сделать красивую визуализацию.

5. Выводы

В ходе лабораторной работы мной был разработан алгоритм определения местоположения абонентов на основе триангуляции по координатам ближайших базовых станций. Для вычисления расстояний между точками использовалась формула Харверсин, учитывающая кривизну Земли и обеспечивающая высокую точность геодезических расчётов. Реализация триангуляции выполнена через метод линейных наименьших квадратов, который позволяет эффективно решать систему уравнений и получать предсказанные координаты с приемлемой точностью. По итогам экспериментов вычислены средняя и медианная нормированные ошибки, что позволило количественно оценить качество предсказаний. Для наглядной оценки создана визуализация с помощью интерактивных карт, отображающих реальные и предсказанные положения абонентов, а также базовые станции.