# Мануальчик по вайб-кодингу

Команда: **Kirill + Vibe**

9 августа 2025 г.

# Функция аппроксимации поверхности квадратичной функцией

Дана функция, аппроксимирующая поверхность набором точек $(x_i, y_i, z_i)$ квадратичной функцией:

$$z = Ax^2 + By^2 + Cxy + Dx + Ey + F,$$

где $A, B, C, D, E, F$ — коэффициенты.

## Вычисление коэффициентов

1. Входные $x, y, z$ преобразуются в одномерные массивы.
2. Строится матрица:
$$A = \begin{bmatrix} x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n^2 & y_n^2 & x_n y_n & x_n & y_n & 1 \end{bmatrix}$$
3. Решается задача наименьших квадратов:
$$\min_{\mathbf{c}} \|A\mathbf{c} - \mathbf{z}\|_2, \quad \mathbf{c} = [A, B, C, D, E, F]^T.$$
4. Результат — вектор $[A, B, C, D, E, F]$.

# Вычисление кривизн

Для $z = Ax^2 + By^2 + Cxy + Dx + Ey + F$ в $(0, 0)$:

$$z_x = D, \quad z_y = E, \quad z_{xx} = 2A, \quad z_{yy} = 2B, \quad z_{xy} = C.$$

$$H = \frac{(1 + z_y^2) z_{xx} - 2 z_x z_y z_{xy} + (1 + z_x^2) z_{yy}}{2(1 + z_x^2 + z_y^2)^{3/2}}, \quad K = \frac{z_{xx} z_{yy} - z_{xy}^2}{(1 + z_x^2 + z_y^2)^2}.$$

Если знаменатель мал, $H$ или $K$ принимаются равными нулю.

# Поиск локальных минимумов

Дано $(X_i, Y_i, Z_i)$. Для каждой точки:

- Соседи: по радиусу $r$ или $k$ ближайших.
- Локальный минимум, если $Z_i < Z_j$ для всех $j$ из соседей.

Функция возвращает булев массив, где `True` — минимум.

## Вычисление геометрических признаков

Функция `compute_geometric_features` считает для каждой точки: глубину, размер бассейна, площадь бассейна, градиенты и кривизны.

```
import numpy as np
import pandas as pd
from math import atan2, pi
from scipy.spatial import cKDTree
from numpy.linalg import lstsq
```

Импортируем необходимые библиотеки: - numpy (np) — для работы с массивами и математикой. - pandas (pd) — для работы с таблицами данных. - atan2, pi — функции из модуля math (угол наклона и число ). - cKDTree — структура данных для быстрого поиска соседей. - lstsq — метод наименьших квадратов для подбора коэффициентов.

```
coords = df[['X','Y']].values
zs = df['Z'].values
n = len(df)

tree = cKDTree(coords)

depth = np.zeros(n)
basin_count = np.zeros(n, int)
basin_area_est = np.zeros(n)
mean_grad = np.zeros(n)
grad_x = np.zeros(n)
grad_y = np.zeros(n)
mean_curv = np.zeros(n)
gauss_curv = np.zeros(n)
orientation = np.zeros(n)

neighbors_R = tree.query_ball_point(coords, r=R)
neighbors_Rb = tree.query_ball_point(coords, r=R_basin)
area_circle = pi * (R_basin**2)
```

Здесь мы: - Извлекаем координаты X, Y и высоты Z из датафрейма. - Определяем общее количество точек n. - Создаём KD-дерево для поиска ближайших соседей. - Инициализируем пустые массивы для хранения глубины, размеров бассейнов, кривизн и т.д. - Находим соседей в радиусе R и $R_{\text{basin}}$. - Вычисляем площадь круга радиуса $R_{\text{basin}}$ для оценки площади бассейна.

```
for i in range(n):
    neigh = [j for j in neighbors_R[i] if j != i]
    if len(neigh) < max(3, min_neighbors):
        req_k = min(max(min_neighbors+1, len(neigh)+1), n)
        _, idx_k = tree.query(coords[i], k=req_k)
        neigh = [j for j in np.atleast_1d(idx_k) if j != i][:max(3, min_neighbors)]
    neigh = np.array(neigh, int)
```

Цикл по всем точкам: - Получаем список соседей в радиусе R. - Если их слишком мало, добираем k ближайших соседей. - Исключаем саму точку из списка соседей. - Преобразуем список соседей в массив целых чисел.

```
    neigh_z = zs[neigh] if neigh.size else np.array([zs[i]])
    depth_i = np.mean(neigh_z) - zs[i]
    depth[i] = depth_i if depth_i > 0 else 0.0
```

Вычисляем глубину: - Берём высоты соседей (`neigh_z`). - Находим разницу между средней высотой соседей и текущей точкой. - Если глубина положительная — сохраняем, иначе ставим 0.

```
    thr = zs[i] + depth_i * basin_frac_of_depth if basin_threshold is None else
        basin_threshold
    idx_rb = [j for j in neighbors_Rb[i] if j != i]
    if basin_condition == 'below':
        basin_idx = [j for j in idx_rb if zs[j] <= thr]
    else:
        basin_idx = [j for j in idx_rb if zs[j] >= thr]
    basin_count[i] = len(basin_idx)
    total_in_circle = len(idx_rb)
    basin_area_est[i] = (basin_count[i] * (area_circle / total_in_circle)) if
        total_in_circle > 0 else 0.0
```

Определяем бассейн: - Вычисляем пороговую высоту для попадания в бассейн (`thr`). - Получаем соседей в радиусе $R_{\text{basin}}$. - В зависимости от условия (`below` или нет) отбираем точки ниже или выше порога. - Считаем количество таких точек и оцениваем площадь бассейна.

```
    xs = coords[neigh,0] - coords[i,0] if neigh.size else np.array([0.0])
    ys = coords[neigh,1] - coords[i,1] if neigh.size else np.array([0.0])
    zs_fit = neigh_z.copy() if neigh.size else np.array([zs[i]])

    xs_fit = np.append(xs, 0.0)
    ys_fit = np.append(ys, 0.0)
    zs_fit = np.append(zs_fit, zs[i])
```

Подготовка данных для аппроксимации: - Переводим координаты соседей в локальную систему (центр в текущей точке). - Берём высоты соседей. - Добавляем в массивы текущую точку (ноль по координатам).

```
    try:
        if xs_fit.size >= 6:
            A_coef,B_coef,C_coef,D_coef,E_coef,F_coef = fit_quadratic_local(xs_fit,
                ys_fit, zs_fit)
        else:
            Xlin = np.column_stack([xs_fit, ys_fit, np.ones_like(xs_fit)])
            a_lin,b_lin,c_lin = lstsq(Xlin, zs_fit, rcond=None)[0]
            A_coef = B_coef = C_coef = 0.0
            D_coef, E_coef, F_coef = a_lin, b_lin, c_lin
    except:
        A_coef = B_coef = C_coef = D_coef = E_coef = F_coef = 0.0
```

Аппроксимация поверхности: - Если точек достаточно, строим квадратичную модель поверхности. - Если мало — линейную модель. - В случае ошибки все коэффициенты обнуляем.

```
    zx, zy, H, K = surface_curvatures_from_coeffs(A_coef, B_coef, C_coef, D_coef, E_coef
        , F_coef)
    grad_x[i], grad_y[i] = zx, zy
    mean_grad[i] = np.sqrt(zx*zx + zy*zy)
    mean_curv[i], gauss_curv[i] = H, K
    orientation[i] = atan2(zy, zx) if not (np.isnan(zx) or np.isnan(zy)) else 0.0

out = pd.DataFrame({
    'depth':depth,'basin_count':basin_count,'basin_area_est':basin_area_est,
    'mean_grad':mean_grad,'grad_x':grad_x,'grad_y':grad_y,
    'mean_curvature':mean_curv,'gauss_curvature':gauss_curv,'orientation':orientation
}, index=df.index)

out = out.fillna(0.0).replace([np.inf,-np.inf], 0.0)
return out
```

Завершающий этап: - Вычисляем производные и кривизны поверхности (`H` — средняя, `K` — гауссова). - Сохраняем градиенты, кривизны и ориентацию. - Формируем итоговую таблицу признаков (`out`). - Заменяем пропуски и бесконечности на нули. - Возвращаем результат.