Георгий Димитриади



Библиотеки Python для научных, прикладных и промышленных вычислений

18+

Георгий Димитриади

Библиотеки Python для научных, прикладных и промышленных вычислений

«Автор»
2026

Димитриади Г.

Библиотеки Python для научных, прикладных и промышленных вычислений / Г. Димитриади — «Автор», 2026

Эта книга посвящена архитектуре и методам применения библиотек Python для научных, прикладных и промышленных вычислений.Последовательно рассмотрены восемь уровней вычислительной экосистемы: базовые численные и символьные методы (NumPy, SciPy, SymPy, mpmath), семейство пакетов SciKits (scikit-learn, scikit-image, scikit-bio, scikit-survival, scikit-rf, scikit-opt, scikit-geometry, scikit-hep), анализ табличных и многомерных данных (Pandas, Polars, Statsmodels, xarray, NetworkX), расширения машинного обучения, ускорение расчетов на CPU и GPU (Numba, Cython, CuPy, RAPIDS, Dask, JAX), научная визуализация, промышленное моделирование и АСУ ТП (Pyomo, OR-Tools, python-control, FMPy, CoolProp, Cantera, PyAnsys, pandapower, SimPy), а также узкопрофильные естественнонаучные пакеты.Каждый раздел содержит описание методов, примеры кода, контрольные вопросы и лабораторные работы. Предназначено для студентов, аспирантов, научных сотрудников и инженеров-исследователей.

© Димитриади Г., 2026
© Автор, 2026

Содержание

Раздел 1. Базовая математика и численные вычисления 5
Раздел 2. Семейство пакетов SciKits 11
Конец ознакомительного фрагмента. 18

Библиотеки Python для научных, прикладных и промышленных вычислений

Раздел 1. Базовая математика и численные вычисления

Раздел 1 охватывает фундаментальный слой вычислительной экосистемы Python - библиотеки NumPy, SciPy, SymPy и mpmath, обеспечивающие векторизованные матричные расчеты, численные методы, символьную алгебру и арифметику произвольной точности.

1.1. Библиотека NumPy: многомерные массивы и векторизованные вычисления

Назначение и архитектура

Библиотека NumPy (*Numerical Python*) является фундаментальным компонентом научной экосистемы Python. Стандартные контейнеры языка Python (например, list) представляют собой динамические массивы указателей на разнородные объекты в памяти, что приводит к значительным накладным расходам при проверке типов и разыменовании ссылок в циклах.

Центральной структурой данных NumPy является класс numpy.ndarray - многомерный однородный массив фиксированного размера. Элементы такого массива имеют единый статический тип данных (dtype, например float64, int32, complex128) и располагаются в непрерывном блоке оперативной памяти (по строкам в стиле C или по столбцам в стиле Fortran).

Такая организация памяти обеспечивает два ключевых преимущества:

Векторизация вычислений. Арифметические и математические операции применяются сразу ко всему массиву без написания явных циклов на уровне интерпретатора Python. Вычисления выполняются в скомпилированных функциях на языках C и Fortran (универсальные функции - ufunc), использующих векторные инструкции процессора (SIMD).

Транслирование (broadcasting). Механизм автоматического согласования размерностей позволяет выполнять поэлементные операции над массивами разной формы без физического дублирования данных в памяти. Два массива считаются совместимыми для транслирования, если при сравнении их размерностей справа налево для каждой оси выполняется одно из условий: размерности равны, либо одна из них равна единице.

Основные методы и функции NumPy

numpy.array(object, dtype=None) - создание объекта ndarray из списка, кортежа или другого итерируемого объекта с явным или автоматическим определением типа данных.

numpy.zeros(shape) / numpy.ones(shape) / numpy.empty(shape) - выделение памяти и инициализация массива заданной размерности нулями, единицами или неинициализированными значениями.

numpy.arange(start, stop, step) / numpy.linspace(start, stop, num) - генерация одномерных массивов арифметической прогрессии с заданным шагом или фиксированным количеством точек на интервале.

numpy.ndarray.reshape(newshape) - изменение размерности (формы) массива без копирования данных в памяти (возвращает представление - view, если это возможно).

numpy.matmul(a, b) (оператор @) - матричное умножение двумерных массивов или стеков матриц.

numpy.linalg.solve(a, b) - решение системы линейных алгебраических уравнений (СЛАУ) вида Ax = b методом LU-разложения с частичным выбором ведущего элемента.

numpy.linalg.eig(a) / numpy.linalg.svd(a) - вычисление собственных значений и собственных векторов квадратной матрицы; сингулярное разложение (SVD) прямоугольной матрицы.

numpy.fft.fft(a) / numpy.fft.ifft(a) - прямое и обратное одномерное дискретное преобразование Фурье на основе алгоритма быстрого преобразования Фурье (БПФ).

numpy.einsum(subscripts, *operands) - вычисление сверток, следов, транспонирования и тензорных произведений по правилу суммирования Эйнштейна.

Пример программной реализации

python

```
import numpy as np
# Формирование матрицы коэффициентов A (3x3) и вектора правых частей b
A = np.array([
[4.0, -1.0, 0.0],
[-1.0, 4.0, -1.0],
[0.0, -1.0, 4.0]
], dtype=np.float64)
b = np.array([15.0, 10.0, 10.0], dtype=np.float64)
# Решение СЛАУ Ax = b и проверка невязки
x = np.linalg.solve(A, b)
residual = np.linalg.norm(A @ x - b, ord=2)
# Вычисление собственных значений и числа обусловленности
eigenvalues, eigenvectors = np.linalg.eig(A)
cond_number = np.linalg.cond(A)
print(f"Вектор решения x: {x}")
print(f"Евклидова норма невязки: {residual:.2e}")
print(f"Число обусловленности: {cond_number:.4f}")
```

Используйте код с осторожностью.

***

1.2. Библиотека SciPy: численные методы и научные алгоритмы

Назначение и структура

Библиотека SciPy (*Scientific Python*) является прямой надстройкой над NumPy и предоставляет готовые реализации фундаментальных численных алгоритмов, обернутых вокруг проверенных библиотек на языках Fortran и C (QUADPACK, ODEPACK, MINPACK, FITPACK, FFTPACK).

Библиотека структурирована в виде независимых тематических подмодулей, требующих явного импорта:

scipy.integrate - численное интегрирование функций и решение систем обыкновенных дифференциальных уравнений (ОДУ);

scipy.optimize - задачи безусловной и условной минимизации, нелинейная регрессия (метод наименьших квадратов), поиск корней нелинейных уравнений;

scipy.interpolate - одномерная и многомерная интерполяция, построение сплайнов;

scipy.linalg - расширенные функции линейной алгебры (матричные экспоненты, разложения Холецкого, Шура, QR);

scipy.sparse - хранение разреженных матриц (форматы CSR, CSC, COO) и итерационные решатели для больших систем;

scipy.signal - проектирование цифровых и аналоговых фильтров, свертка, спектральный анализ сигналов;

scipy.stats - непрерывные и дискретные вероятностные распределения, описательная статистика, параметрические и непараметрические критерии проверки гипотез.

Основные методы и функции SciPy

scipy.integrate.quad(func, a, b) - вычисление определенного однократного интеграла методом адаптивной квадратуры Гаусса-Кронрода.

scipy.integrate.solve_ivp(fun, t_span, y0, method='RK45') - численное решение задачи Коши для систем ОДУ первого порядка. Поддерживает явные методы Рунге-Кутты (RK45, DOP853) для нежестких задач и неявные методы (Radau, BDF) для жестких систем.

scipy.optimize.minimize(fun, x0, method=..., bounds=..., constraints=...) - численный поиск минимума скалярной функции многих переменных (методы Нелдера-Мида, Бройдена - Флетчера - Гольдфарба - Шанно BFGS, L-BFGS-B, последовательного квадратичного программирования SLSQP).

scipy.optimize.root(fun, x0) - нахождение корней векторной нелинейной функции $F(x) = 0$.

scipy.interpolate.CubicSpline(x, y) - построение кусочно-полиномиального кубического сплайна с заданными граничными условиями.

scipy.signal.butter(N, Wn, btype='low') / scipy.signal.filtfilt(b, a, x) - расчет коэффициентов цифрового фильтра Баттерворта и двунаправленная фильтрация сигнала без фазовых искажений.

scipy.stats.kstest(rvs, cdf) - критерий согласия Колмогорова-Смирнова для проверки принадлежности выборки заданному теоретическому распределению.

Пример программной реализации

python

```
import numpy as np
from scipy.integrate import solve_ivp
from scipy.optimize import minimize
# 1. Численное интегрирование системы ОДУ (затухающий осциллятор Дуффинга)
def duffing_oscillator(t, state, delta=0.2, alpha=-1.0, beta=1.0):
x, v = state
dxdt = v
dvdt = -delta * v - alpha * x - beta * (x ** 3)
return [dxdt, dvdt]
sol = solve_ivp(
fun=duffing_oscillator,
t_span=(0.0, 20.0),
y0=[1.0, 0.0],
method="RK45",
t_eval=np.linspace(0, 20, 500)
)
# 2. Условная нелинейная оптимизация (функция Розенброка с ограничением)
rosenbrock = lambda x: (1.0 - x)**2 + 100.0 * (x - x**2)**2
constraint = {"type": "ineq", "fun": lambda x: 1.5 - (x**2 + x**2)}
opt_res = minimize(rosenbrock, x0=np.array([0.0, 0.0]), method="SLSQP", constraints=constraint)
print(f"Конечное состояние осциллятора: x(20) = {sol.y[0, -1]:.4f}")
print(f"Точка минимума с учетом ограничения: {opt_res.x}")
```

Используйте код с осторожностью.

***

1.3. Библиотека SymPy: система компьютерной алгебры

Назначение и принципы работы

Библиотека SymPy (*Symbolic Python*) представляет собой систему компьютерной алгебры (CAS), написанную целиком на языке Python без внешних бинарных зависимостей. В отличие от NumPy и SciPy, оперирующих численными приближениями с конечной разрядностью, SymPy работает с точными математическими объектами и графами выражений.

Использование символьных вычислений позволяет выполнять аналитические преобразования до этапа численного моделирования: выводить точные формулы градиентов и матриц Якоби, находить аналитические решения дифференциальных уравнений, выполнять разложение в ряды Тейлора и преобразовывать полученные символьные выражения в высокоскоростные функции NumPy для последующей подстановки массивов данных.

Основные методы и функции SymPy

sympy.symbols(names) - объявление символьных переменных и задание их математических свойств (вещественные, положительные, целые).

sympy.simplify(expr) / sympy.expand(expr) / sympy.factor(expr) - алгебраическое упрощение выражений, раскрытие скобок и разложение многочленов на множители.

sympy.diff(expr, var, n) - вычисление аналитической производной n-го порядка по переменной var.

sympy.integrate(expr, (var, a, b)) - вычисление неопределенного или определенного интеграла в замкнутой аналитической форме.

sympy.limit(expr, var, point) / sympy.series(expr, var, point, n) - вычисление пределов функций и разложение в степенной ряд Тейлора.

sympy.solve(equations, variables) / sympy.dsolve(eq, func) - аналитическое решение алгебраических уравнений и обыкновенных дифференциальных уравнений.

sympy.Matrix.jacobian(variables) - символьное построение матрицы Якоби для вектор-функции.

sympy.lambdify(args, expr, modules='numpy') - компиляция символьного выражения в стандартную функцию Python, совместимую с массивами NumPy.

Пример программной реализации

python

```
import sympy as sp
import numpy as np
# Объявление символьных переменных
x, y, t = sp.symbols('x y t', real=True)
# Определение вектор-функции и вычисление матрицы Якоби
F = sp.Matrix([x**2 * sp.sin(y), sp.exp(-x) + y**3])
J = F.jacobian([x, y])
# Аналитическое решение линейного ОДУ второго порядка: y''(t) + 4*y(t) = 0
func_y = sp.Function('y')
ode = sp.Eq(func_y(t).diff(t, 2) + 4 * func_y(t), 0)
general_sol = sp.dsolve(ode, func_y(t))
# Трансляция символьного выражения в векторизованную функцию NumPy
det_J = J.det()
det_J_num = sp.lambdify((x, y), det_J, modules='numpy')
grid_val = det_J_num(np.array([1.0, 2.0]), np.array([0.5, 1.0]))
print(f"Определитель матрицы Якоби: {det_J}")
print(f"Общее решение ОДУ: {general_sol}")
print(f"Численные значения на сетке: {grid_val}")
```

Используйте код с осторожностью.

***

1.4. Библиотека mpmath: арифметика произвольной точности

Назначение и применение

Стандартные вещественные числа с двойной точностью (float64, стандарт IEEE 754) обеспечивают около 15–17 значащих десятичных цифр и ограничены диапазоном экспоненты. При решении плохо обусловленных систем линейных уравнений, вычислении осциллирующих

интегралов, исследовании специальных функций в теории чисел и квантовой механике ошибки округления накапливаются, что приводит к потере устойчивости алгоритмов.

Библиотека mpmath реализует вещественную и комплексную арифметику с плавающей точкой произвольной точности, управляемой пользователем. Она включает реализации практически всех специальных математических функций (гипергеометрических, эллиптических, функций Бесселя, дзета-функции Римана), а также алгоритмы численного дифференцирования, интегрирования и матричных разложений высокой точности.

Основные методы и настройки mpmath

mpmath.mp.dps / mpmath.mp.prec - глобальная настройка точности вычислений в количестве значащих десятичных знаков (dps) или двоичных битов мантиссы (prec).

mpmath.workdps(n) - контекстный менеджер для временного изменения точности вычислений на заданном участке кода.

mpmath.mpf(val) / mpmath.mpc(real, imag) - создание вещественного или комплексного числа с произвольной точностью.

mpmath.quad(f, [a, b]) - численное интегрирование с высокой точностью методами Танха-Синха и Гаусса-Лежандра.

mpmath.findroot(f, x0) - нахождение корней нелинейных уравнений с произвольной точностью.

mpmath.matrix() / mpmath.lu_solve(A, b) - линейная алгебра повышенной точности для плохо обусловленных матриц (например, матриц Гильберта).

Пример программной реализации

python

```
import mpmath as mp
# Установка точности вычислений: 60 значащих десятичных цифр
mp.mp.dps = 60
# Решение плохо обусловленной СЛАУ с матрицей Гильберта размерности 15x15
n = 15
H = mp.hilbert(n)
b = mp.matrix([mp.mpf(1) for _ in range(n)])
x = mp.lu_solve(H, b)
# Вычисление невязки с заданной точностью
residual = mp.norm(H * x - b)
# Вычисление константы Апери zeta(3) с высокой точностью
apery_const = mp.zeta(3)
print(f"Норма невязки для матрицы Гильберта (60 знаков): {mp.nstr(residual, 10)}")
print(f"Значение zeta(3): {mp.nstr(apery_const, 50)}")
```

Используйте код с осторожностью.

***

Контрольные вопросы к разделу 1

В чем заключается принципиальное отличие структуры numpy.ndarray от стандартного списка list языка Python с точки зрения организации памяти и скорости выполнения операций?

Каков механизм работы транслирования массивов (broadcasting) в NumPy и при каких условиях размерности двух массивов считаются совместимыми?

В чем состоит различие между представлением массива (view) и его копией (copy) в NumPy? Какие операции над массивами возвращают представление?

В чем заключается разница между численным и символьным дифференцированием? В каких инженерных задачах целесообразно применять SymPy, а в каких - SciPy?

Как функция sympy.lambdify() связывает символьные вычисления с библиотекой NumPy и почему ее использование критично при последующем численном моделировании?

Какие численные методы интегрирования реализованы в функции scipy.integrate.solve_ivp() и по какому признаку выбирается метод для жестких систем обыкновенных дифференциальных уравнений?

В каких вычислительных задачах стандартной 64-битной арифметики с плавающей точкой (float64) недостаточно и требуется применение библиотеки mpmath?

Практическая лабораторная работа 1

Тема: «Численное и аналитическое моделирование динамических и алгебраических систем»

Цель работы: Освоить методы векторизованных матричных вычислений, символьных преобразований, численного решения дифференциальных уравнений и вычислений повышенной точности.

Порядок выполнения:

С помощью SymPy задать аналитическое выражение функции двух переменных (f(x, y) = e^{-x^2 - y^2} cos(2x)), вычислить ее градиент и матрицу Гессе в символьном виде. Транслировать определитель матрицы Гессе в функцию NumPy с помощью sp.lambdify() и вычислить его значения на двумерной координатной сетке np.meshgrid().

С помощью NumPy сформировать систему линейных алгебраических уравнений Ax = b размерности 100 × 100 с диагональным преобладанием, решить ее функцией np.linalg.solve(), вычислить число обусловленности матрицы (np.linalg.cond()) и проверить евклидову норму невязки (Vert{}Ax - bVert{}_2).

С помощью scipy.integrate.solve_ivp() выполнить численное интегрирование системы уравнений Лоренца (σ = 10, ρ = 28, β = 8/3) на интервале t при двух наборах начальных условий, отличающихся на 10. Вычислить расхождение траекторий во времени.

С помощью mpmath решить систему линейных уравнений с матрицей Гильберта размерности 20 × 20 при точности 15 и 80 десятичных знаков, сравнив полученные решения с точным аналитическим вектором.

Перейти к написанию полного текста Раздела 2 (Семейство пакетов SciKits) или сразу сформировать готовый файл .docx по мере готовности разделов?

Раздела 2 (Семейство пакетов SciKits)

Раздел 2 «Семейство пакетов SciKits» охватывает независимые научные расширения над базовым вычислительным стеком NumPy и SciPy - от стандарта машинного обучения scikit-learn до узкопрофильных пакетов для обработки изображений (scikit-image), биоинформатики (scikit-bio), анализа выживаемости (scikit-survival), радиочастотной инженерии (scikit-rf), эвристической оптимизации (scikit-opt), вычислительной геометрии (scikit-geometry) и физики высоких энергий (scikit-hep).

Раздел 2. Семейство пакетов SciKits

Семейство SciKits (*SciPy Toolkits*) представляет собой совокупность независимых пакетов с открытым исходным кодом, разрабатываемых научным сообществом для решения специализированных прикладных задач поверх массивов NumPy и алгоритмов SciPy. Пакеты этого семейства не включаются в основную поставку SciPy из-за узкой предметной специализации или различных циклов разработки, однако поддерживают строгую совместимость структур данных и единые соглашения о проектировании программных интерфейсов.

2.1. Библиотека scikit-learn: классическое машинное обучение

Назначение и архитектура API

Библиотека scikit-learn является отраслевым стандартом для решения задач классического машинного обучения: классификации, регрессии, кластеризации, снижения размерности, отбора признаков и валидации моделей.

Ключевым архитектурным достижением scikit-learn является унифицированный объектно-ориентированный программный интерфейс (API), определяющий три базовые абстракции:

Оценщик (Estimator) - любой объект, способный оценивать параметры на основе обучающих данных с помощью метода .fit(X, y).

Преобразователь (Transformer) - оценщик, выполняющий трансформацию данных с помощью методов .transform(X) и .fit_transform(X, y) (например, масштабирование признаков или метод главных компонент).

Предиктор (Predictor) - оценщик, формирующий прогнозы на новых данных с помощью метода .predict(X), а также возвращающий вероятности классов (.predict_proba(X)) или значения решающей функции (.decision_function(X)).

Последовательное объединение преобразователей и предикторов в единый конвейер (Pipeline) предотвращает утечку данных (data leakage) при кросс-валидации.

Основные классы и методы scikit-learn

sklearn.model_selection.train_test_split(*arrays, test_size, stratify) - разбиение выборки на обучающую и тестовую подвыборки с возможностью стратификации по целевой переменной.

sklearn.model_selection.StratifiedKFold(n_splits) / cross_val_score(estimator, X, y, cv) - перекрестная проверка с сохранением пропорций классов в каждом блоке.

sklearn.preprocessing.StandardScaler() / OneHotEncoder() - стандартизация числовых признаков и унитарное кодирование категориальных переменных.

sklearn.decomposition.PCA(n_components) - линейное снижение размерности методом главных компонент на основе сингулярного разложения.

sklearn.ensemble.RandomForestClassifier() / GradientBoostingRegressor() - ансамблевые алгоритмы на базе решающих деревьев.

sklearn.pipeline.Pipeline(steps) / sklearn.compose.ColumnTransformer(transformers) - построение конвейеров обработки данных и раздельных преобразований для разнородных столбцов.

sklearn.metrics.classification_report() / roc_auc_score() / mean_squared_error() - расчет метрик качества моделей классификации и регрессии.

Пример программной реализации

python

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
# Генерация синтетической выборки для бинарной классификации
X, y = make_classification(n_samples=1000, n_features=20, n_informative=10, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)
# Построение конвейера обработки и классификации
model_pipeline = Pipeline([
("scaler", StandardScaler()),
("pca", PCA(n_components=10)),
("classifier", RandomForestClassifier(n_estimators=150, random_state=42))
])
# Оценка методом 5-блочной кросс-валидации и обучение на всей обучающей выборке
cv_scores = cross_val_score(model_pipeline, X_train, y_train, cv=5, scoring="roc_auc")
model_pipeline.fit(X_train, y_train)
test_accuracy = model_pipeline.score(X_test, y_test)
print(f"Средний ROC-AUC на кросс-валидации: {np.mean(cv_scores):.4f}")
print(f"Точность на тестовой выборке: {test_accuracy:.4f}")
Используйте код с осторожностью.
***
```

2.2. Библиотека scikit-image: научная обработка изображений

Назначение и представление данных

Библиотека scikit-image предназначена для количественного анализа и обработки двумерных и трехмерных (вольюметрических) растровых изображений в научных исследованиях, микроскопии, материаловедении и дистанционном зондировании. В отличие от библиотеки OpenCV, ориентированной на задачи компьютерного зрения в реальном времени, scikit-image оперирует непосредственно стандартными массивами NumPy без использования специализированных структур-оберток.

Полутоновые изображения представляются двумерными массивами формы (M, N), цветные RGB-изображения - трехмерными массивами формы (M, N, 3), а томографические объемы - массивами формы (Z, Y, X).

Основные модули и функции scikit-image

skimage.io.imread(fname) / imsave(fname, arr) - чтение и запись изображений в различных форматах (TIFF, PNG, JPEG).

skimage.color.rgb2gray(rgb) - преобразование цветного изображения в полутоновое по весовой формуле яркости.

skimage.filters.gaussian(image, sigma) / sobel(image) - линейная гауссова фильтрация шумов и выделение градиентов яркости оператором Собеля.

skimage.filters.threshold_otsu(image) - автоматическое вычисление глобального порога бинаризации путем минимизации внутриклассовой дисперсии.

skimage.morphology.erosion() / dilation() / opening() / closing() - операции математической морфологии с заданным структурирующим элементом (skimage.morphology.disk).

skimage.segmentation.watershed(image, markers) - сегментация соприкасающихся объектов методом водораздела.

skimage.measure.label(label_image) / regionprops_table(label_image, properties) - маркировка связных компонент и извлечение геометрических характеристик объектов (площадь, периметр, эксцентриситет, эквивалентный диаметр) в виде словаря или таблицы.

Пример программной реализации

```
python
import numpy as np
from skimage import data, filters, morphology, measure
# Загрузка эталонного микроскопического изображения монет и подавление шума
image = data.coins()
smoothed = filters.gaussian(image, sigma=1.5)
# Пороговая бинаризация методом Оцу и морфологическое закрытие разрывов
thresh = filters.threshold_otsu(smoothed)
binary_mask = morphology.closing(smoothed > thresh, morphology.disk(3))
# Маркировка связных областей и расчет морфометрических параметров
labeled_objects = measure.label(binary_mask)
props = measure.regionprops_table(
labeled_objects,
intensity_image=image,
properties=["label", "area", "eccentricity", "mean_intensity"]
)
# Фильтрация мелких артефактов (площадью менее 100 пикселей)
valid_areas = props["area"][props["area"] > 100]
print(f"Обнаружено объектов: {len(valid_areas)}, средняя площадь: {np.mean(valid_areas):.1f} пикс.")
```

Используйте код с осторожностью.

***

2.3. Библиотека scikit-bio: биоинформатика и анализ последовательностей

Назначение и область применения

Библиотека scikit-bio предоставляет структуры данных и вычислительные алгоритмы для биоинформатики, метагеномики и анализа экологических сообществ. Она включает инструменты валидации и манипуляции с нуклеотидными и аминокислотными последовательностями, алгоритмы парного и множественного выравнивания, работу с филогенетическими деревьями и статистический анализ матриц расстояний.

Основные классы и функции scikit-bio

skbio.DNA(sequence) / skbio.RNA(sequence) / skbio.Protein(sequence) - типизированные объекты биологических последовательностей с поддержкой метаданных, позиционных оценок качества и методов транскрипции/трансляции.

skbio.alignment.local_pairwise_align_ssw(seq1, seq2) - быстрое локальное выравнивание последовательностей алгоритмом Смита-Ватермана с использованием SIMD-инструкций.

skbio.TreeNode.read(file, format='newick') - чтение, обход и расчет эволюционных расстояний (например, метрики UniFrac) на филогенетических деревьях.

skbio.diversity.alpha_diversity(metric, counts) / beta_diversity(metric, counts) - расчет индексов внутригруппового (Шеннона, Симпсона, Chao1) и межгруппового (Брея-Кертиса) биологического разнообразия.

skbio.stats.ordination.pcoa(distance_matrix) - анализ главных координат (метрическое многомерное шкалирование) для визуализации матриц различий.

skbio.stats.distance.permanova(distance_matrix, grouping) - непараметрический многомерный дисперсионный анализ на основе перестановок.

Пример программной реализации

```
python
from skbio import DNA
from skbio.diversity import alpha_diversity, beta_diversity
```

```
# Работа с нуклеотидной последовательностью ДНК
dna_seq = DNA("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", metadata={"id": "seq_01"})
mrna_seq = dna_seq.transcribe()
protein_seq = mrna_seq.translate()
# Расчет индексов альфа- и бета-разнообразия для матрицы численности видов (3 образца, 4 вида)
abundance_matrix = [,
,
[0, 2, 15, 8]
]
sample_ids = ["Sample_A", "Sample_B", "Sample_C"]
shannon_idx = alpha_diversity("shannon", abundance_matrix, ids=sample_ids)
bray_curtis_dm = beta_diversity("braycurtis", abundance_matrix, ids=sample_ids)
print(f"Транслированная белковая последовательность: {protein_seq}")
print(f"Индекс Шеннона для образцов:n{shannon_idx}")
Используйте код с осторожностью.
***
```

2.4. Библиотека scikit-survival: анализ выживаемости и надежности

Назначение и специфика данных

Библиотека scikit-survival расширяет архитектуру scikit-learn для решения задач анализа времени до наступления события (*time-to-event analysis*): прогнозирования времени отказа промышленного оборудования, оценки кредитного дефолта или выживаемости пациентов в клинических исследованиях.

Особенностью таких задач является наличие цензурированных данных (*right-censored data*), когда за время наблюдения целевое событие не наступило, и известна только нижняя граница времени безотказной работы. Стандартные методы регрессии в этом случае дают смещенные оценки.

Основные классы и функции scikit-survival

sksurv.util.Surv.from_arrays(event, time) - формирование структурированного массива пар (статус_события, время), используемого в качестве целевой переменной y.

sksurv.nonparametric.kaplan_meier_estimator(event, time) - непараметрическая оценка функции выживания Каплана-Мейера.

sksurv.linear_model.CoxPHSurvivalAnalysis(alpha) - регрессионная модель пропорциональных рисков Кокса с возможностью L2-регуляризации.

sksurv.ensemble.RandomSurvivalForest(n_estimators) / GradientBoostingSurvivalAnalysis() - ансамблевые модели выживаемости на основе решающих деревьев.

sksurv.svm.FastSurvivalSVM() - машина опорных векторов для задач ранжирования и регрессии времени до события.

sksurv.metrics.concordance_index_censored(event_indicator, event_time, estimate) - расчет индекса согласованности Харрелла (C-index) для оценки качества ранжирования рисков.

Пример программной реализации

```
python
import numpy as np
from sksurv.util import Surv
from sksurv.linear_model import CoxPHSurvivalAnalysis
from sksurv.metrics import concordance_index_censored
# Формирование данных об отказах 6 узлов оборудования (True - отказ зафиксирован, False - цензурировано)
```

```
events = np.array([True, False, True, True, False, True])
times = np.array([120.0, 200.0, 85.0, 150.0, 240.0, 60.0])
y_surv = Surv.from_arrays(event=events, time=times)
# Матрица признаков эксплуатации (температура, вибрация)
X_features = np.array([
[75.0, 1.2], [60.0, 0.8], [82.0, 1.8],
[70.0, 1.1], [58.0, 0.7], [90.0, 2.3]
])
# Обучение модели пропорциональных рисков Кокса и расчет C-индекса
cox_model = CoxPHSurvivalAnalysis()
cox_model.fit(X_features, y_surv)
risk_scores = cox_model.predict(X_features)
c_index = concordance_index_censored(events, times, risk_scores)[0]
print(f"Коэффициенты риска признаков: {cox_model.coef_}")
print(f"Индекс согласованности Харрелла (C-index): {c_index:.4f}")
```

Используйте код с осторожностью.

***

2.5. Библиотека scikit-rf: радиочастотная и микроволновая инженерия

Назначение и область применения

Библиотека scikit-rf (*skrf*) предназначена для анализа и проектирования радиочастотных (RF) и микроволновых цепей, волноводов и антенно-фидерных трактов. Она обеспечивает чтение и запись данных векторных анализаторов цепей в стандартном формате Touchstone (.s1p, .s2p, ..., .snp), преобразование матриц параметров (S-, Z-, Y-, ABCD-параметры), каскадное соединение многополюсников, деэмбеддинг и калибровку измерений.

Основные классы и методы scikit-rf

skrf.Network(file) - создание объекта СВЧ-цепи из файла Touchstone или на основе массивов частот и комплексной матрицы рассеяния S.

skrf.Network.s_db / s_deg / vswr - свойства для извлечения амплитуды S-параметров в децибелах, фазы в градусах и коэффициента стоячей волны по напряжению (КСВН).

skrf.Network.cascade(other) (оператор **) - каскадное соединение двух двухполюсников.

skrf.Network.deembed(other) - математическое исключение влияния измерительной оснастки (de-embedding).

skrf.media.Coaxial(frequency, Dint, Dout, epsilon_r) / MLine() - расчет параметров линий передачи (коаксиальных кабелей, микрополосковых линий).

skrf.calibration.SOLT(ideals, measured) - выполнение однопортовой или двухпортовой калибровки методом Short-Open-Load-Thru.

Пример программной реализации

```
python
import numpy as np
import skrf as rf
# Создание частотной сетки от 1 до 5 ГГц (401 точка) и моделирование коаксиальной линии
freq = rf.Frequency(1, 5, 401, "GHz")
coax_media = rf.media.Coaxial(frequency=freq, Dint=1.0e-3, Dout=3.5e-3, epsilon_r=2.1)
# Формирование отрезка линии длиной 50 мм и каскадное соединение двух отрезков
line_segment = coax_media.line(d=50, unit="mm", name="line_50mm")
cascaded_line = line_segment ** line_segment
# Извлечение коэффициента передачи S21 (в дБ) и входного КСВН на частоте 3 ГГц
idx_3ghz = np.argmin(np.abs(freq.f - 3e9))
```

s21_db = cascaded_line.s_db[idx_3ghz, 1, 0]
vswr_in = cascaded_line.s_vswr[idx_3ghz, 0, 0]
print(f"Потери передачи S21 на 3 ГГц: {s21_db:.4f} дБ, КСВН: {vswr_in:.4f}")
Используйте код с осторожностью.
***

2.6. Библиотека scikit-opt: метаэвристические алгоритмы оптимизации

Назначение и алгоритмы

Библиотека scikit-opt реализует популяционные и траекторные метаэвристические алгоритмы для поиска глобального экстремума нелинейных, недифференцируемых, разрывных и многоэкстремальных целевых функций, для которых классические градиентные методы (scipy.optimize) сходятся к ближайшему локальному минимуму.

Основные классы и методы scikit-opt

sko.GA.GA(func, n_dim, size_pop, max_iter, lb, ub) - классический генетический алгоритм с бинарным или вещественным кодированием, операторами селекции, кроссовера и мутации.

sko.PSO.PSO(func, n_dim, pop, max_iter, lb, ub, w, c1, c2) - оптимизация методом роя частиц (Particle Swarm Optimization).

sko.DE.DE(func, n_dim, size_pop, max_iter, lb, ub) - алгоритм дифференциальной эволюции.

sko.SA.SA(func, x0, T_max, T_min, L) - метод имитации отжига (Simulated Annealing).

sko.ACA.ACA_TSP(func, n_dim, size_pop, max_iter, distance_matrix) - муравьиный алгоритм для решения комбинаторной задачи коммивояжера (TSP).

Algorithm.run() - запуск итерационного процесса оптимизации; возвращает вектор оптимальных параметров best_x и значение целевой функции best_y.

Пример программной реализации

python
import numpy as np
from sko.DE import DE
# Многоэкстремальная тестовая функция Растригина (глобальный минимум 0 в точке x = 0)
def rastrigin(x):
return 10 * len(x) + np.sum(x**2 - 10 * np.cos(2 * np.pi * x))
# Поиск глобального минимума в 4-мерном пространстве методом дифференциальной эволюции
de_optimizer = DE(
func=rastrigin,
n_dim=4,
size_pop=60,
max_iter=200,
lb=[-5.12] * 4,
ub=[5.12] * 4
)
best_x, best_y = de_optimizer.run()
print(f"Найденный вектор параметров: {np.round(best_x, 5)}")
print(f"Значение целевой функции: {best_y[0]:.6e}")
Используйте код с осторожностью.

2.7. Библиотека scikit-geometry: вычислительная геометрия

Назначение и базовые примитивы

Библиотека scikit-geometry предоставляет высокоуровневый интерфейс Python к вычислительной геометрической библиотеке C++ CGAL (*Computational Geometry Algorithms Library*).

Она предназначена для точных геометрических расчетов в системах автоматизированного проектирования (САПР), робототехнике, геоинформатике и раскрое листовых материалов.

Основные классы и функции scikit-geometry

skgeom.Point2(x, y) / Segment2(p, q) / Ray2() / Line2() - базовые двумерные геометрические примитивы.

skgeom.Polygon(vertices) / PolygonWithHoles(outer, holes) - представление простых многоугольников и многоугольников с внутренними вырезами.

skgeom.boolean_set.intersect(p1, p2) / join(p1, p2) / difference(p1, p2) - точные теоретико-множественные булевы операции над полигонами.

skgeom.minkowski.minkowski_sum(p1, p2) - вычисление суммы Минковского двух многоугольников (используется при планировании траекторий роботов с учетом их габаритов).

Конец ознакомительного фрагмента.

Текст предоставлен ООО «Литрес».

Прочитайте эту книгу целиком, купив полную легальную версию на Литрес.

Безопасно оплатить книгу можно банковской картой Visa, MasterCard, Maestro, со счета мобильного телефона, с платежного терминала, в салоне МТС или Связной, через PayPal, WebMoney, Яндекс.Деньги, QIWI Кошелек, бонусными картами или другим удобным Вам способом.