DOI 10.37882/2223-2966.2023.9-2.25

ОПЫТ ИСПОЛЬЗОВАНИЯ ПАКЕТА PY-PDE ДЛЯ РЕШЕНИЯ ДИФФЕРЕНЦИАЛЬНЫХ УРАВНЕНИЙ

EXPERIENCE OF USING PY-PDE PACKAGE FOR SOLVING DIFFERENTIAL EQUATIONS

A. Suvorov

Summary. In this article capabilities of py-pde Python package are considered. The package is used for numerical solution of a big class of differential equations in which the unknown function can depend on time and space variables. Examples of using this package for solving linear differential equations with one unknown function are given.

Keywords: Differential equation, numerical solution, time dependency, python.

Суворов Александр Павлович
Кандидат технических наук, доцент, Московский государственный строительный университет
suvorovap@mgsu.ru

Аннотация. В данной статье рассматриваются возможности пакета языка Python py-pde. Этот пакет предназначен для численного решения большого класса дифференциальных уравнений, в которых неизвестная функция может зависеть от времени и от пространственных переменных. Приводятся примеры использования пакета для решения линейных дифференциальных уравнений с одной неизвестной функцией.

Ключевые слова: дифференциальное уравнение, численное решение, временная зависимость, python.

Введение

В данной статье мы рассматриваем возможности пакета языка Python py-pde [1]. Этот пакет предназначен для численного решения большого класса дифференциальных уравнений, в которых неизвестная функция может зависеть от одной пространственной переменной и времени или от двух пространственных переменных и времени. В случае двух пространственных переменных задача может быть решена только на прямоугольной области. Если обозначить неизвестную функцию как u, то дифференциальное уравнение должно иметь такой общий вид

$$\frac{\partial u}{\partial t} = \text{правая часть}$$

где правая часть может включать в себя любые производные функции u по пространственным координатам, но не производную по времени.

Хотя py-pde и содержит документацию [1] с большим количеством примеров дифференциальных уравнений (на английском языке), в этой статье приводятся дополнительные примеры, которые позволят отечественному пользователю лучше понять достоинства и недостатки этого пакета. К тому же надо отметить, что каждый пример (уравнение), приведенный в документации, рассматривает лишь отдельную, малую часть возможностей программы. Здесь же мы попытаемся для меньшего количества примеров упомянуть сразу большое количество опций.

Пример 1. Рассмотрим уравнение

$$\Delta u + f = 0,$$

где неизвестная функция $u = u(x, y, t)$ определена на квадратной области $x \in [0,1], y \in [0,1], f = 8(s + g)$ — заданная функция интенсивности источников, s, g — заданные числа. Функция u удовлетворяет на границе области следующим граничным условиям

$$u(x = 0) = u(x = 1) = 4gy(1 - y)$$

$$u(y = 0) = u(y = 1) = 4sx(1 - x)$$

Точное решение данного уравнения можно быть легко получено: $u = 4gy(1 - y) + 4sx(1 - x)$. Поэтому данное уравнение следует решать с помощью py-pde только для тестирования программы.

В соответствии с общей концепцией данной программы py-pde решает уравнение для функции, зависящей от времени, т.е. ищется решение для такого уравнения

$$\frac{\partial u}{\partial t} = \Delta u + f.$$

Поэтому решение исходного уравнения, в котором временная зависимость не присутствует, может быть получено методом установления [2], предположив, что t достаточно большое число. При использовании пакета py-pde требуется определить лишь часть уравнения, стоящую справа от $\frac{\partial u}{\partial t}$ после знака равенства.

Запишем короткую программу для решения данного уравнения

```
from pde import PDE, CartesianGrid, ScalarField
import pde
import numpy as np
import matplotlib.pyplot as plt
```

```
s=5
g=12
grid = CartesianGrid([[0, 1], [0, 1]], [64,64]) # генерация сетки
bc_x = [{“value_expression”: f”4*{g}*y*(1-y)” }, {“value_expression”: f”4*{g}*y*(1-y)”}]
bc_y = [{“value_expression”: f”4*{s}*x*(1-x)” } , {“value_expression”: f”4*{s}*x*(1-x)”}]
eq = PDE ({“u”: f” laplace(u) + 8*({s} + {g})”}, bc = [bc_x, bc_y ])
state = ScalarField(grid, 0.0) ## задание нулевых начальных условий
res = eq.solve(state, t_range=500000e-6, method=”scipy”) ## решение до времени t_range
res.plot(cmap=»magma»)
```

Решение данного уравнения начинается с генерации сетки, размер которой 64 на 64 точки. Далее задаются граничные условия для левого и правого краев области (переменная bc_x), потом задаются граничные условия для нижнего и верхнего краев (переменная bc_y). Это заданные значения неизвестной функции. Выражения для определения граничных условий записываются как строки, которые могут быть интерпретированы пакетом sympy. Далее используется класс PDE, который требует определения правой части уравнения $\Delta u + f$. Это уравнение тоже записывается как строка sympy. Также полностью записываются все граничные условия. Обратим внимание на оператор laplace. Помимо оператора laplace в py-pde существуют операторы d2_dx2, d2_dy2, d_dx, d_dy. Поэтому вместо laplace(u) можно было бы использовать d2_dx2(u)+d2_dy2(u). Не рекомендуется использовать d_dx(d_dx(u)) вместо d2_dx2(u) . Это приведет к неправильному результату.

С помощью класса ScalarField задается скалярное поле на сетке grid. По существу это массив значений, определенный для данной сетки. К этому массиву применимы многие операции библиотеки numpy. Значение данного поля в данном примере равно нулю — это начальное условие, которое по существу и не нужно задавать для решения исходного уравнения, но нужно задавать в данном пакете, так как неизвестная функция u всегда рассматривается как функция, зависящая от времени.

Далее используется функция solve для решения. При этом задаются начальное условие в виде скалярного поля, диапазон времени, для которого будет искаться решение (переменная t_range). Решение res будет выведено только для конечного момента времени t_range. Решение res тоже представляет собой скалярное поле, определенное на сетке grid. График с данным решением представлен на рис. 1. Можно убедиться, что данное решение является установившимся и не зависит от значения t_range, если последнее выбрано достаточно большим числом.

Заметим, что ввиду симметрии задачи решение можно было бы получить на четверти области $x \in [0, 0.5], y \in [0, 0.5]$, но тогда для краев $x = 0$, $y = 0$, нужно было бы определить нулевые значения для производных функции u (условия симметрии).

В этом случае в программе нужно сделать следующие изменения:

```
grid = CartesianGrid([[0, 0.5], [0, 0.5]], [64,64]) # генерация сетки
bc_x = [{“value_expression”: f”4*{g}*y*(1-y)” }, {“derivative”: f”0”}]
bc_y = [{“value_expression”: f”4*{s}*x*(1-x)” } , {“derivative”: f”0” }]
```

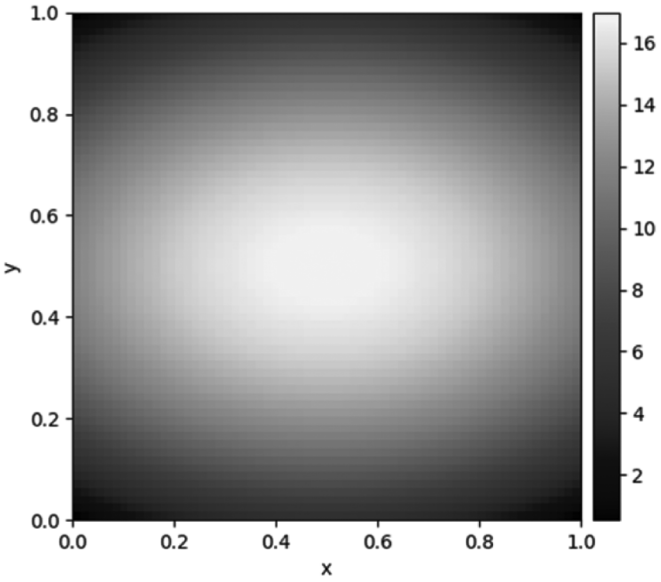

Рис. 1. Решение уравнения $\Delta u + f = 0$ на квадратной сетке 64 на 64

Заметим, что при задании граничных условий сначала задаются условия для левого края области, затем для правого. Аналогично, для другого направления: сначала записываются условия для нижнего края области, затем для верхнего.

Для получившегося решения в виде поля res доступны многие операции, в том числе нахождение производных. Это возможно при помощи операции gradient

```
resd_dx = res.gradient(bc = [bc_x, bc_y ])[0]
resd_dx.plot(cmap=”magma”)
```

При этом указываются вновь граничные условия. Градиент содержит два поля: производная по переменной x и y, поэтому указав индекс 0, мы сообщили программе, что нас интересует производная по x.

Теперь рассмотрим случай, когда функция источников задана не на всей области, а только для ее части $x \in [0.25, 0.75], y \in [0.25, 0.75]$. К сожалению, команда

eq = PDE ({«u»: f» laplace(u) + 8*({s} + {g})*Heaviside(x-0.25,0.5)*Heaviside(y-0.25,0.5)»}, bc = [bc_x, bc_y])

вызывает проблемы, хотя функция Heaviside определена в библиотеке sympy. Здесь мы рассматриваем решение задачи только для четверти области. Поэтому можно заменить пороговую функцию более гладкой функцией, например, tanh

eq = PDE ({«u»: f» laplace(u) + 8*({s} + {g})*(tanh((x-0.25)/0.002)/2+0.5)*(tanh((y-0.25)/0.002)/2+0.5)»}, bc = [bc_x, bc_y])

Далее рассмотрим решение данной задачи, но при помощи пользовательского класса. Задачу будем решать на четверти области ввиду симметрии, а функцию источников $8(s + g)$ будем считать определенной только для части области $x \in [0.25, 0.75], y \in [0.25, 0.75]$. Приводим код программы.

```
from pde import PDE, CartesianGrid, ScalarField, PDEBase
import pde
import numpy as np
import matplotlib.pyplot as plt
s=5
g=12
class MyPDE(PDEBase):
 def __init__(self, bc="auto_periodic_neumann"):
 super().__init__()
    bc_x = [{"value_expression": f"4*{g}*y*(1-y)" }, {"derivative":"0"}]
 bc_y = [{"value": f"4*{s}*x*(1-x)"}, {"derivative":"0"}]
 self.bc = [bc_x, bc_y]

def evolution_rate(self, state, t=0):
state_lap = state.laplace(bc=self.bc)
return state_lap + 8*(s+g)*np.heaviside(xfield-0.25,0.5)*np.heaviside(yfield-0.25,0.5)

grid = CartesianGrid([[0, 0.5], [0, 0.5]], [32,32])
state = ScalarField(grid, 0.0) # задать начальные условия
xfield = ScalarField.from_expression(grid, "x")
yfield = ScalarField.from_expression(grid, "y")

eq = MyPDE() # задать уравнение
result = eq.solve(state, t_range=500000e-6, method='scipy')
result.plot()
```

В этом классе должна быть определена функция evolution_rate, в которой задается и возвращается правая часть уравнения. Обратим внимание на то, что здесь оператор laplace действует над полем решения state. Оператор laplace, как и все остальные операторы, вычисляющие производные, требует задание граничных условий bc. Далее в правой части уравнения могут присутствовать и другие скалярные поля, заданные на той же сетке. Здесь мы используем поля xfield и yfield, которые просто заменяют переменные *x* и *y*. Также используется функция heaviside из библиотеки numpy, которую можно применить к полю. Обратим внимание на то, что здесь нельзя в качестве координат просто указать название переменной *x* или *y*, так как это команда языка Python, а не символьная строчка, и переменные *x*, *y* не определены.

К сожалению, такая имплементация задачи, хотя и является более общей, приводит к значительному росту времени исполнения задачи по сравнению с предыдущим кодом. Но использование другого метода при решении приводит все-таки к убыстрению работы программы:

result = eq.solve(state, t_range=500000e-6, method="AdaptiveSolverBase")

Пример 2. Рассмотрим уравнение для функции одной пространственной переменной

$$\frac{d^2u}{dr^2} + \frac{1}{r}\frac{du}{dr} - \frac{u}{r^2} = 0.$$

Такое уравнение может возникать при решении задачи о радиально-симметричной деформации цилиндра, и в этом случае $u(r)$ — радиальное перемещение точек цилиндра. Если на внутренней и внешней стенках цилиндра, $r = r_i$ и $r = r_e$, действуют радиальные напряжения σ_i, σ_e, соответственно, то граничные условия могут быть записаны следующим образом

$$\left(K - \frac{2}{3}G\right)\left(\frac{du}{dr} + \frac{u}{r}\right) + 2G\frac{du}{dr} = \sigma_i r = r_i$$

$$\left(K - \frac{2}{3}G\right)\left(\frac{du}{dr} + \frac{u}{r}\right) + 2G\frac{du}{dr} = \sigma_e r = r_e$$

Здесь K, G — модули упругости материала цилиндра, которые считаются заданными числами.

Как всегда, py-pde будет решать уравнение

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial r^2} + \frac{1}{r}\frac{\partial u}{\partial r} - \frac{u}{r^2}.$$

В программе py-pde граничные условия Робина, например, для внешнего края, должны быть записаны в таком виде

$$\frac{du}{dr} + \frac{K - \frac{2}{3}G}{K + \frac{4}{3}G}\frac{1}{r_e}u = \frac{\sigma_e}{K + \frac{4}{3}G}$$

т.е. должны быть найдены множитель перед функцией и свободный член, причем коэффициент перед производной принимается равным единице.

Рассмотрим решение задачи для цилиндра, для которого радиальная координата меняется в диапазо-

не $0.5 \le r \le 1.0$. Пусть внутренняя сторона цилиндра не загружена, т.е. $\sigma_i = 0$, а на внешней стороне цилиндра приложены напряжения $\sigma_e = -100$.

Приводим код программы

```
from pde import PDE, CartesianGrid, ScalarField
import numpy as np
import matplotlib.pyplot as plt
K = 43383.94
G = 20000
L = 1
grid = CartesianGrid([[0.5*L, L]], 64) # сетка
c1 = (K-2/3*G)/(K+4/3*G)
c2 = -100/(K+4/3*G)
eq = PDE ({“u”: f” laplace(u) + d_dx(u)/x — u/x**2”}, bc = [{“type”:”mixed”, “value”: c1/L, “const”: 0}, {“type”:”mixed”, “value”: c1/L, “const”: c2}])
state = ScalarField(grid, 0.0) # начальные условия
res = eq.solve(state, t_range=500000e-6, method=”scipy”)
plt.plot(np.linspace(0.5,1,64), res.data)
plt.grid(‘on’)
plt.xlabel(‘r’)
plt.ylabel(‘u(r)’)
plt.show()
```

Сетка является одномерной и содержит 64 точки. В определении уравнения мы используем оператор первой производной d_dx. В определении граничных условий записываем сначала условия на левом конце, потом на правом конце области. Используем тип граничных условий mixed (смешанный). График, показывающий перемещения цилиндра, приведен на рис. 2.

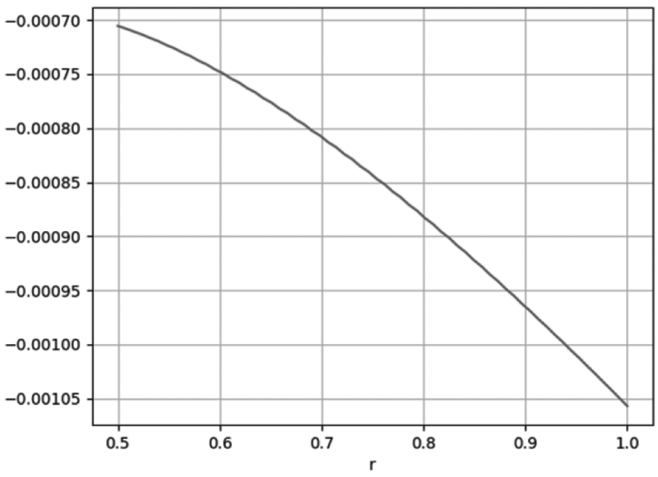

Рис. 2. Радиальные перемещения точек цилиндра при его загружении внешним давлением, внутреннее давление равно нулю

Пример 3. Рассмотрим уравнение массопереноса, в котором зависимость неизвестной функции *u* от времени имеет конкретный смысл. Решим уравнение

$$\frac{\partial u}{\partial t} = 0.01\frac{\partial^2 u}{\partial x^2} - \frac{\partial u}{\partial x}$$

в области $x \in [0,1]$.. Граничные условия

$$u(0,t) = 1, \frac{\partial u}{\partial x}(1,t) = 0.$$

Начальное условие

$$u(x, t = 0) = 0.$$

Как мы видим, коэффициент диффузии в данной задаче мал (0.01). Поэтому перенос вещества происходит в основном за счет адвекции.

Приводим код программы

```
from pde import PDE, CartesianGrid, ScalarField
import pde
import numpy as np
import matplotlib.pyplot as plt
grid = CartesianGrid([[0,1]], 64) # сетка

eq = PDE ({“u”: f” 0.01*laplace(u) — d_dx(u)”},
bc = [{“value_expression”:1}, {“value”: 0}])

state = ScalarField(grid, 0.0)
storage = MemoryStorage()
res = eq.solve(state, t_range=750000e-6, method=”scipy”, tracker=storage.tracker(5000e-6))
plot_kymograph(storage)
```

Здесь создается экземпляр класса MemoryStorage, который запоминает все промежуточные состояния вплоть до конечного времени t_range. Затем на основе этой информации строится график, по вертикальной шкале которого откладывается время, а по горизонтальной шкале откладывается координата *x* (рис. 3).

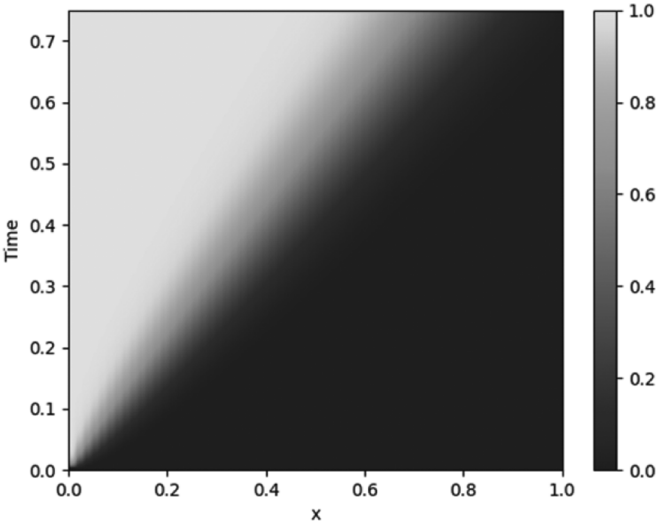

Рис. 3. Зависимость концентрации вещества *u* от координаты *x* и времени в случае переноса вещества слева направо

Заключение

В этой статье рассматривается использование пакета py-pde для решения линейных дифференциальных уравнений с одной неизвестной функцией. Пакет может быть использован как для задач, в которых неизвестная функция зависит от времени, так и для стационарных задач. В последнем случае задача решается методом установления, т.е. используется решение для большого значения времени, когда решение можно считать неизменяющемся во времени. Приведены примеры решения уравнения на отрезке и в прямоугольной области. В случае двух пространственных переменных задача может быть решена только для прямоугольной области.

К достоинствам пакета относятся возможность найти не только само решение, но и производные от него, возможность задавать разного рода граничные условия, в том числе зависящие от времени, рассматривать функции источников, которые тоже зависят произвольным образом от времени и пространственных переменных. К недостаткам данной программы относится тот факт, что граничные условия удовлетворяются лишь приближенно, но точность улучшается при сгущении сетки. Также надо отметить невысокую скорость работы программы, если сетка становится густой.

ЛИТЕРАТУРА

1. Документация к пакету py-pde https://py-pde.readthedocs.io/en/latest/
2. Зализняк, В.Е. Численные методы. Основы научных вычислений: учебник и практикум для вузов / В. Е. Зализняк. — 2-е изд., перераб. и доп. — Москва: Издательство Юрайт, 2023. — 356 с. — (Высшее образование). — ISBN 978-5-534-02714-3. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/510699

Журнал «Современная наука: актуальные проблемы теории и практики»