Однослойные нейронные сети в задачах аппроксимации многомерных функций и решении эллиптических уравнений

Г.В. Орлов[1], Е.И. Несмиянов[1,2]
[1]ФГУП «РФЯЦ- ВНИИТФ имени академика Е.И. Забабахина», Снежинск, Россия
[2] Южно-Уральский государственный университет, Челябинск, Россия

Задача нашего доклада — познакомить слушателей с технологией применения современных методов аппроксимации многомерных функций и решения задач математической физики средствами простейших однослойных нейронных сетей на исключительно удобном и мощном языке Python с применением модуля тензорной алгебры torch, предназначенного для решения многогранных задач нейронных сетей и не только.

В конце 60-х годов прошлого века были доказаны две теоремы:

А.Н. Колмогоровым «О представлении непрерывных функций нескольких переменных в виде суперпозиции непрерывных функций одного переменного // Доклады АН СССР, 1957, Т. 114». и независимо

В. И. Арнольдом «О представлении функций нескольких переменных в виде суперпозиции функций меньшего числа переменных, Матем. просв., 1958, вып. 3».

Эти теоремы породили многочисленные уточнения конкретного вида суперпозиции непрерывных функций. Самый простой вид суперпозиции был предложен для однослойных нейронных сетей в виде следующей теоремы -

Theorem 2 [1]. (Cybenko G.V. Approximation by Superpositions of Sigmoidal function //
Mathematics of Control Signals and Systems. 1989. vol.2. pp. 303-314.)

Let σ be any continuous sigmoidal function. Then finite sums of the form

$$u(x) = \sum_{j=1}^{N} \alpha_j \, \sigma\left(y_j^T x + \theta_j\right) \quad (1)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which $|G(x) - f(x)| < \varepsilon$ for all $x \in I_n$.

Definition. We say that σ is sigmoidal if $\sigma(t) \to \begin{cases} 1 & t \to +\infty \\ 0 & t \to -\infty \end{cases}$.

Сейчас мы покажем, как элементарно решается задача аппроксимации $u(x) \approx f(x)$ многомерной функции f(x) средствами модуля torch, определяя искомые параметры p_{min} формулы (1) как $p_{min} = \underset{p}{argmin} [\![u(x) - f(x)]\!]_{L^2}$. Но сначала представим (1) в ясном и удобном виде. Пусть

N - размерность задачи,

K – число нейронов одного слоя,

x=[**x**$_1$, **x**$_2$, …, **x**$_L$] — набор L точек пространства I_N, **x**$_l$=[x$_{1l}$,x$_{1l}$,...,x$_{1l}$]^T — вектор-колонка точки.

Тогда (1) можно записать следующим образом:

$$u(x) = \overbrace{[w_1, w_2, \dots, w_K]}^{w} \cdot \sigma \left(\overbrace{\begin{bmatrix} \mu_{11} & \mu_{12} & \cdots & \mu_{1N} \\ \mu_{21} & \mu_{22} & \cdots & \mu_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{K1} & \mu_{K2} & \cdots & \mu_{KN} \end{bmatrix}}^{\mu} @ \overbrace{\begin{bmatrix} \overbrace{x_{11}}^{x_1} & \overbrace{x_{12}}^{x_2} & & \overbrace{x_{1L}}^{x_L} \\ x_{21} & x_{22} & \cdots & x_{2L} \\ \vdots & \vdots & & \vdots \\ x_{N1} & x_{N2} & & x_{NL} \end{bmatrix}}^{x} + \overbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}}^{b} \right) +$$

$$B = w \cdot \sigma(\mu @ x + b) + B, \quad (2)$$

где параметрами функции *u(**x**)* в количестве (N+2)· K+1 являются следующие величины :

w размерность [1, K], **μ** размерность [K, N], **b** размерность [K, 1], B, (3)

а размерность произведения $\mu @ x$ есть [K, L].

Удивительный факт — имея из популярной статьи только формулу (1), молодой дипломник Е.И. Несмиянов, соавтор этого доклада, сумел понять как работают в ней все компоненты и вывести формулу (2) добавив параметр общего сдвига B. Далее он написал на языке Python программу аппроксимации $u(x) \approx f(x)$, для чего, не зная о системе torch, вручную написал градиенты десятков параметров для некоторой программы минимизации. Титанический труд и способности!

Теперь теорему (1) можно сформулировать более понятно. Аппроксимации вида (2) плотны в I_N в метрике C, то есть, для любой $U(x) \in C(I_n)$

$$||U(x) - u(x)||_C \underset{K \to \infty}{\to} 0, \quad (4)$$

где K — число, **независимых** реакций вида $w_k \sigma(\mu_k \cdot x + b_k)$.

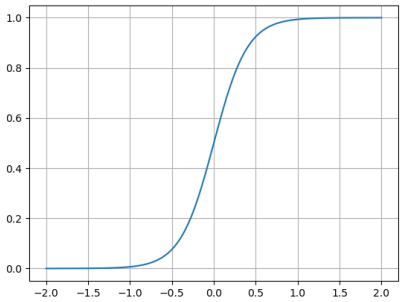
Функцию вида (2) естественно назвать KAN-аппроксимацией (KAN-моделью, KAN-представлением и т.п.) так как с одной стороны многомерная непрерывная функция U(x) согласно теореме Колмогорова-Арнольда раскладывается в сумму суперпозиций одномерных функций, а с другой - по теореме Цыбенко эта суперпозиция есть произведение одномерных функций активации нейронов, то есть формула (2) задает однослойную нейронную сеть с K нейронами.

Инженерные публикации о нейронных сетях пестрят двумя десятками различных видов сигмоидов. Легко показать, что большинство из них эквивалентны, а популярный сигмоид Relu в виде наклонной линии задает для KAN-моделей плоскость в N-мерном пространстве I_N **при любом** $K \geq N$, но **независимых** реакций нейронов вида $w_k \sigma(\mu_k \cdot x + b_k)$ здесь всего N! Следует сказать, что траектории программы минимизации при движении к минимуму невязки конечно будут отличаться для разных, хотя и эквивалентных видов сигмоидов. Мы будем пользоваться правильным

и простым сигмоидом вида

$$0 < \sigma(x) = \frac{1}{1+e^{-\lambda x}} < 1. \qquad (5)$$

С таким сигмоидом KAN-функции (2) являются бесконечно гладкими на всем пространстве I_N , а за счет параметра **μ** способны отразить любую конечную крутизну аппроксимируемой функции. Ситуации очень напоминает аппроксимацию обобщенными функциями с основным пространством бесконечно гладких функций медленного роста.



Для любого n = 1,2,…,N имеем

$$\frac{\partial}{\partial x_n} u(x) = w \cdot \sigma'(\mu @ x + b) @ \mu_{:n}, \mu_{:n} = \begin{bmatrix} \mu_{1n} \\ \mu_{2n} \\ \vdots \\ \mu_{Kn} \end{bmatrix} .$$
$$\sigma'(s) = \lambda \cdot \sigma(s)\big(1 - \sigma(s)\big)$$

(6)

Из (6) легко получить смешанную производную функции $u(x)$по$\partial x_n \partial x_m$:

$$\frac{\partial^2}{\partial x_n \partial x_m} u\,(x) = w \cdot \sigma''(\mu_k @ x + b_k) @ \mu_{:n} @ \mu_{:m}, n, m = 1,2,\ldots,N .$$
$$\sigma''(s) = \lambda^2 \cdot \sigma(s)\big(1 - \sigma(s)\big)\big(1 - 2 \cdot \sigma(s)\big)$$

(7)

Из (7) следует, что оператор Лапласа функции $u(x)$имеет следующий вид:

$$\Delta u(x) = w \cdot \sigma''(\mu_k @ x + b_k) @ \sum_{n=1}^{N} \mu_{:n} @ \mu_{:n} = w \cdot \sigma''(\mu_k @ x + b_k) \begin{bmatrix} \mu_{1:} \cdot \mu_{1:} \\ \mu_{2:} \cdot \mu_{2:} \\ \vdots \\ \mu_{K:} \cdot \mu_{K:} \end{bmatrix} .$$
$$\mu_{k:} = [\mu_{k1}, \mu_{k2}, \ldots, \mu_{kN}], k = 1,2,\ldots,K ,$$

(8)

Задачу KAN-аппроксимации$U(x) \approx u(x)$многомерной функции *U(x)* будем решать нахождением таких параметров w, μ, b, B однослойной нейронной сети (2), которые минимизируют невязку

$$||U(x) - u(x)||_{L^2} .$$

(9)

Эту задачу мы будем решать с привлечением мощной системы разработки нейронных сетей torch

https://github.com/torch/torch7/wiki/Cheatsheet, http://torch.ch/docs/package-docs.html.

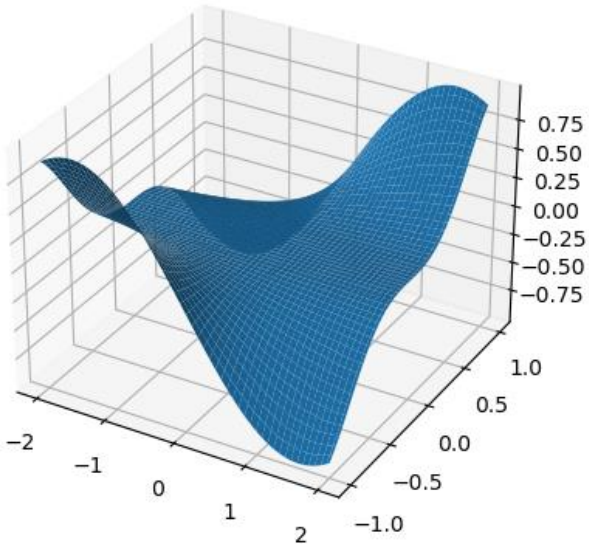
На языке Python в системе torch вся работа осуществляется с тензорами, при этом все ее операции могут осуществляться на графической плате в технологии CUDA. Для этого, необходимо лишь перевести тензор методом .cuda() в память графической платы. После этого все операции с этим тензором, включая промежуточные, будут производиться очень быстро (примерно в 10 раз быстрее) на ядрах графической платы и в ее памяти. Мощь системы torch проявляется в том, что кроме указания тензору .cuda() для работы с ним в технологии CUDA больше ничего не требуется! Все многостраничные и многотрудные методы, описываемые в руководствах по работе с cudatoolkit здесь совершенно не требуются.

Задача минимизации решается в системе torch одним из 11 методов градиентного спуска, для которых нахождение градиента функции (2) по параметрам w, μ, b, B осуществляется методом автоматического символьного дифференцирования и не требует от пользователя никаких усилий. Это еще один весомый аргумент в пользу системы torch, особенно в случае когда количество нейронов K не только изменяется в процессе подбора, но и может доходить до многих десятков и даже сотен. Например, для двумерной функции с типичным значением K=20 количество параметров KAN-аппроксимации равно 81.

Обычное заблуждение относительно языка Python заключается в уверенности, что программы на этом языке работают медленно, так как это интерпретатор. Такое мнение отстало минимум на два десятка лет. Действительно, на первом проходе происходит интерпретация конструкций языка но **с сохранением полученных кодов**! При повторном проходе этих конструкций для исполнения используются их готовые коды. Более важным является то, что эти коды — это коды виртуальной машины LLVM, которые исполняются интерпретаторами, написанными разработчиками аппаратуры, знающими и умеющими выжать из своего устройства максимум возможностей. Технологии LLVM не менее двух десятков лет, и она является повсеместно используемым стандартом для большинства процессоров, включая CUDA. Именно поэтому современные LLVM часто превосходят по производительности прежние технологи, основанные на кодах, полученных прямой трансляцией с популярных языков программирования.

Покажем как получаются KAN-аппроксимации с участием системы torch на примере аппроксимации двумерной функции

$U(x,y) = sin(x \cdot y) \cdot y^2, -2 \le x \le 2, -1 \le y \le 1$ (10)

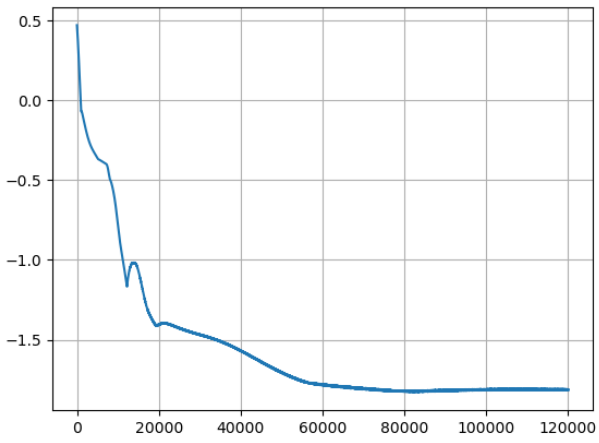



```
N = 2; K = 20
#задание пространственной сетки grid
x = np.linspace(-2,+2,100)
y = np.linspace(-1,+1,200)
grid = [x,y] = np.meshgrid(x,y)
# задание исходной функции
U = np.sin(x*y])*y*y
# grid_c это тензор grid в CUDA
grid_c = torch.tensor(grid).cuda()
def sigma(s): # определение сигмоида
    return 1/(1+torch.exp(-5.*s))
# задание исх.параметров случайными величинами
params = (torch.rand(N*K+K+K+1)-0.5).requires_grad_(True)
optimiser = torch.optim.Adam([params],lr=lr) # выбор метода минимизации
```

В программах для всех величин сноской _c отмечается тот факт, что параметр находится в памяти CUDA графической платы. Используемая видеокарта — это Nvidia RTX3070, 5 888 ядер в CUDA и 120 тензорных ядер. В системе torch можно уверенно рекомендовать к использованию метод стохастического градиентного спуска Adam. Методы минимизации работают одинаковым образом: от начального значения параметров `params` (10) в пользовательском цикле повторяются команды

```
# вычисление KAN-аппроксимации u≈U (2) для тензоров w,mu,b,B
u_c = w_c@sigma(mu_c@grid_c+b_c)+B_c
loss = torch.norm(U-u_c)   # вычисление невязки (9)
torch.optimiser.zero_grad()
loss.backward()  # автоматическое дифференцирование loss
                 # по параметрам params = [w, mu,b,B]
optimiser.step() # шаг метода минимизации по параметрам params
```

После предложенного пользователем выхода из цикла минимизации, осуществляется требуемая обработка полученных результатов. Так функция $lg||U(x) - u(x)||_C(i)$ где i - номер итерации, имеет следующий график. Здесь за время 1м 38 с была получена относительная точность менее 1.5%.



Результаты KAN-аппроксимации многомерных функций открывают новые возможности в аппроксимации, например, уравнений состояния причем сразу в потенциалах. Области фаз могут быть произвольными и даже не связными. Поверхности раздела фаз также можно задавать KAN-аппроксимациями. На этом пути просматривается очень перспективное направление симплектических KAN-аппроксимаций, при которых сохраняется структура симплектического многообразия в фазовом пространстве, что гарантирует правильное поведение траекторий и интегралов движения, а энергия системы остается постоянной. Следует сказать, что в системе torch для таких работ есть компонента toch.nn.

Перейдем к KAN-аппроксимациям решений уравнений математической физики. Идея такой аппроксимации совершенно понятна — дифференциальный оператор D(U(**x**))=0, примененный к бесконечно дифференцируемой KAN-модели u(**x**)≈U(**x**) с неизвестными параметрами дает KAN-аппроксимацию D(u(**x**)), для которой, устремляя $[\![D(u(x))]\!]_{L^2} \to 0$ можно найти искомые параметры KAN-представления решения u(**x**). Уже здесь возникают трудности для эллиптического уравнения $\Delta U(x,y) = 0$, поскольку решение этого уравнения определено с точностью до произвольной гармонической функции, то есть функции, оператор Лапласа от которой равен нулю. Учет граничных и краевых условий требует одновременной минимизации и невязок этих условий и невязки уравнения.

Рассмотрим KAN-решение двумерной задачи Дирихле:

$$\Delta U|_{\Omega} = 0, \Omega = \{(x,y)\},$$
$$S = \partial\Omega, U_S = U(S)(x,y) \in S'$$

(11)

где Ω - заданная конечная область с границей S, U(x,y) — искомая функция в области Ω, U_S(x,y)=U(S) — заданное значение для функции U на границе S. Будем находить параметры KAN-аппроксимации минимизацией функционала невязки $loss = ||\Delta u(\Omega)||_{L_2} + \alpha \cdot ||u(S) - U(S)||_{L_2} \to 0$:

```
def sigma(s): return 1.0/(1+torch.exp(-lmbd*s))
def d2Sigma(sgm): return lmbd*lmbd*sgm*(1-sgm)*(1-lmbd*sgm)
# оператор Лапласа от KAN-аппроксимации u(x,y) на сетке grid
sgm =sigma(mu_c@grid_c+b_c)
Deltau_c = w_c@(((mu_c*mu_c).sum(dim=1)[:,None])*d2Sigma(sgm))
```

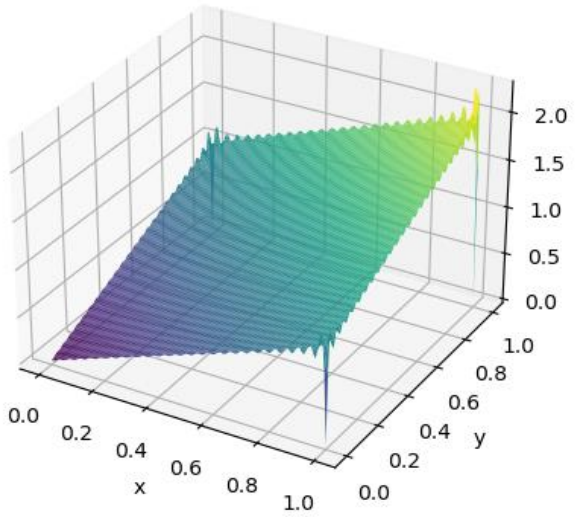
```
# значение u(S) на сетке границы S, US-заданное U(S) на границе S
uS_c = w_c@sigma(mu_c@S_c+b_c)+B_c
loss = torch.norm(Deltau_c)+alpha*torch.norm(uS_c-US_c)
```
(12)

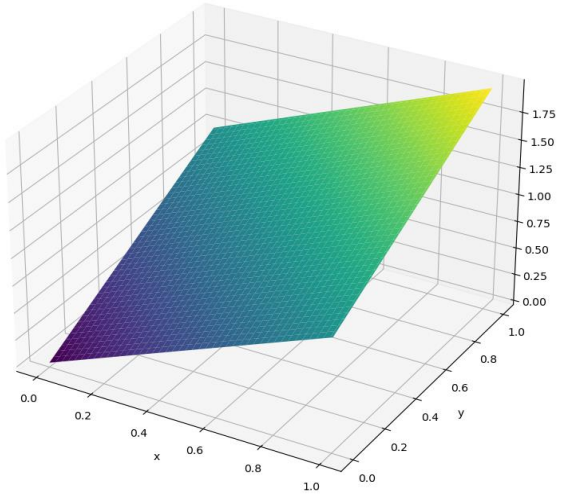
Найдем KAN-решение следующей простой тестовой задачи Дирихле (11) на двумерном единичном квадрате с исходной функцией $U(x,y) = x + y, 0 \le x, y \le 1$, задающей граничное значение на сторонах квадрата. Точное решение этой задачи (см. Линейные уравнения математической физики. Справочная математическая библиотека. - М.: Наука, 1964) таково:

$$U(x,y) = u_1(x,y) + u_2(x,y) + u_3(x,y) + u_4(x,y)$$
$$u_1(x,y) = \sum_{n=1}^{n=\infty} \frac{2}{\pi n \cdot sh\pi n}(a - b \cdot (-1)^n) \cdot sh\big(\pi n(1-y)\big) \cdot sin\pi nx$$
$$u_2(x,y) = \sum_{n=1}^{n=\infty} \frac{2}{\pi n \cdot sh\pi n}(d - c \cdot (-1)^n) \cdot sh(\pi ny) \cdot sin\pi nx$$
$$u_3(x,y) = \sum_{n=1}^{n=\infty} \frac{2}{\pi n \cdot sh\pi n}(a - d \cdot (-1)^n) \cdot sh\big(\pi n(1-x)\big) \cdot sin\pi ny$$
$$u_2(x,y) = \sum_{n=1}^{n=\infty} \frac{2}{\pi n \cdot sh\pi n}(b - c \cdot (-1)^n) \cdot sh(\pi nx) \cdot sin\pi ny$$

(13)

Сравним это решение с KAN-решением:





Точное решение $U(x,y)$ (13) для $1 \le n = 50$

KAN-решение $[x] = [y] = 128, K = 20, \alpha = 1$

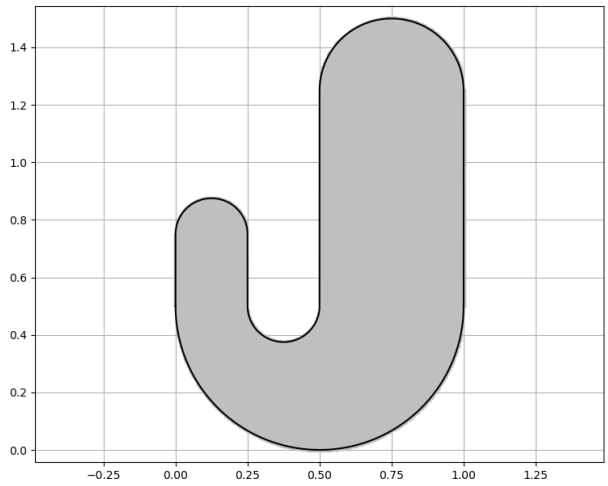
Точное решение представляют ряды Фурье (13). При конечном количестве членов этих рядов из-за разрыва краевых значений на концах своих интервалов такое точное решение будет иметь явное проявление эффекта Гиббса. В то же время KAN-решение этой краевой задачи является плоскостью без видимого такого проявления с максимальным отклонением от точного плоского решения менее 0.02% в угловых точках заданной области. Внутри области отклонение на порядок меньше. Нам представляется, что это хороший пример “простой“ задачи.

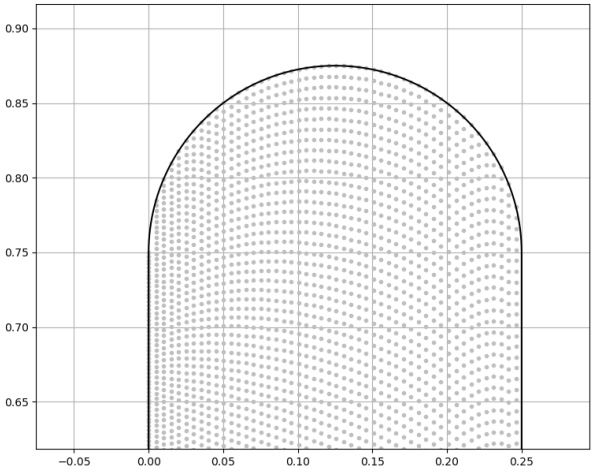
Найдем численное решение следующей тестовой краевой задачи Пуассона:

$$\Delta U(x,y) = f(x,y)$$
$$U(x,y) = x^3 + y3, f(x,y) = 6 \cdot (x+y), U(S) = x^3 + y^3, S = \partial\Omega\text{'}$$

(14)

где областьΩимеет сложную геометрию представленного ниже вида.





Область Ωи ее границы S Фрагмент сетки сетки областиΩи ее границы S

Краевые задачи Пуассона с такой геометрией области для сеточных методов обычно представляют значительные трудности. Следует заметить, что начинать задание исходных данных следует с задания сеточной границы S, после чего сетка `grid` внутренней области может строится любым удобным способом, включая случайный наброс точек, так как алгоритм KAN-решения задачи не требует какой либо регулярности от указанных множеств, а название „сетка“ используется здесь из-за уважения к традициям.

Легко понять, что решение собственно уравнения Пуассона (14) определено с точностью до какой-то гармонической функции. Ситуация аналогична неопределенности константы C при взятии неопределенного интеграла, где неопределенность константы раскрывается заданием граничных условий интеграла. Совершенно аналогично, неопределенность решения уравнения (14) раскрывается заданием граничного значения гармонической функции g(x,y), для которой по определению$\Delta g(x,y) = 0$.

Таким образом, численное решение задачи Пуассона разбивается на два отдельных этапа - Арсенин В.Я. Математическая физика. Основные уравнения и специальные функции. - М. : Наука, 1966. На первом этапе находится численное $uP(x,y)$решение уравнения Пуассона, которое есть сумма

$$uP(x,y) = u(x,y) + g(x,y), (x,y) \in \Omega,$$
$$uP_S(S) = U(S) + g(S) \quad \text{'}$$

(15)

откуда получаем граничное значение неизвестной гармонической функции g(x,y) -

$$g(S) = uP_S(S) - U(S). \quad (16)$$

Решая на втором этапе задачу Дирихле

$$\Delta g(x,y) = 0, (x,y) \in \Omega$$
$$g(x,y) = uP(x,y) - U(x,y), (x,y) \in S'$$

(17)

находим KAN-аппроксимацию ug(x,y) функции g(x,y), откуда искомая численная KAN-аппроксимация решения $u(x,y) \approx U(x,y)$задачи Дирихле (14) есть разность

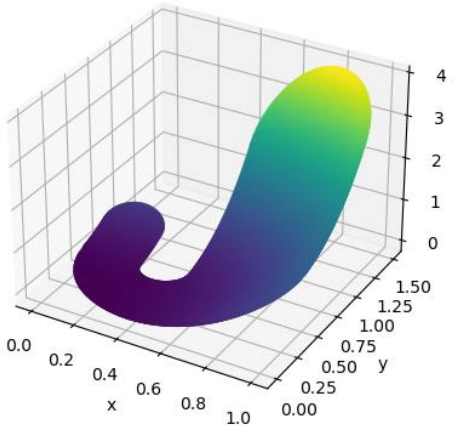
$$u(x,y) = uP(x,y) - ug(x,y) \approx U(x,y).$$

(18)

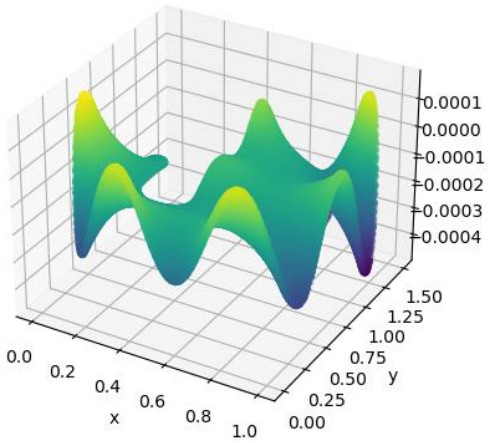
Обратим внимание на то, что такой способ решения задач Дирихле дает **разные** KAN-функции uP(x,y) и ug(x,y) для решения совершенно **разных задач**, что отличается от немалого числа инженерных публикаций, в которых поиск параметров численного решения$u(x,y)$задачи Пуассона осуществляется наивной минимизацией невязки **одной** функции

$$loss = ||\Delta u(x,y) - f(x,y) \in \Omega||_{L_2} + \alpha \cdot ||u(x,y) - U(x,y) : (x,y) \in S||_{L_2} \to 0$$

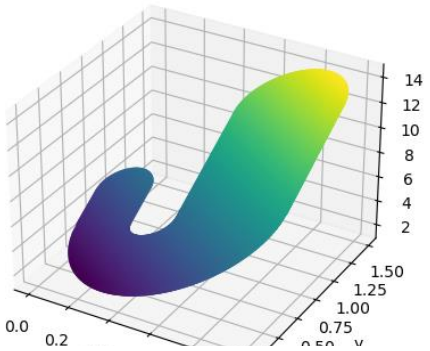
с каким-то назначенным параметромα. Понятно, что описать единой функцией, которая действием оператора Лапласа определяет в области Ωзаданную правую часть$f(x,y) \neq 0$и одновременно этим же оператором определяет нулевую функцию в этой же области вряд ли возможно. Так выглядят графики решения задачи (14):



Точное U и KAN-решение u краевой задачи Пуассона



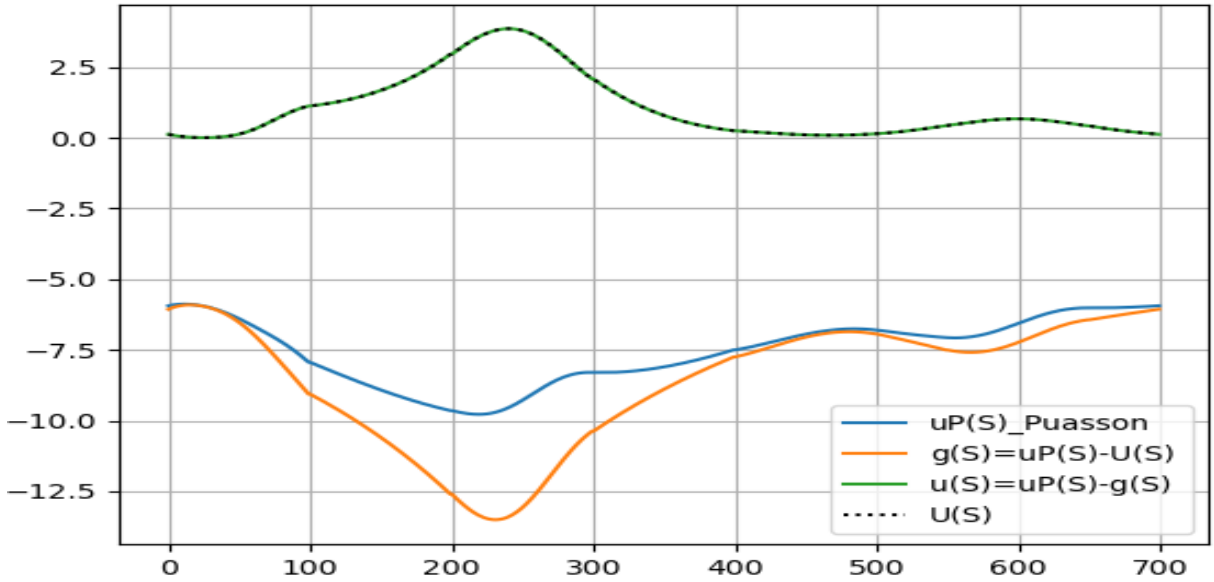
Разность $U - u$точного и KAN-решения Явно не случайный шум - это опять



гармоническая функция, но маленькая и ее можно изложенным способом снова уменьшить!

Точная правая часть f(x,y) и ее KAN-аппроксимация

Здесь точные функции и их KAN-аппроксимации на графиках неотличимы



Заданное граничное значение U(S), граничное значение KAN-решения уравнения Пуассона uP(S), граничное значение g(S) =uP(S)-U(S) задачи Дирихле , граничное значение u(S)=uP(S)-g(S), заданное граничное значение U(S) · · · ·. Исходное граничное значение U(S) с отклонением 0.0005 и с относительной точностью 0.012% совпало с граничным значением u(S) KAN-решения краевой задачи Пуассона.

Найдем KAN-решение следующей тестовой краевой задачи Неймана

$$\Delta U(x,y) = 0\ (x,y) \in \Omega, \tag{19}$$

$$S = \partial\Omega, \frac{\partial U}{\partial n}(S) = \psi(x,y), x,y \in S, \tag{20}$$

с областьюΩпредыдущей задачи ее границей S. Здесь

$\frac{\partial U}{\partial n} = \nabla U \cdot n_S$- нормальная производная в точке границы S,

$\mathbf{n}_S(x,y)$ - единичный вектор внешней нормали к границе S в точке$(x,y) \in S$, то есть внешняя нормаль к касательной границы в этой точке; совокупность таких нормалей обозначим как $\mathbf{n}_S(S)$,

$\psi \in C_S$ - заданная непрерывная функция на S.

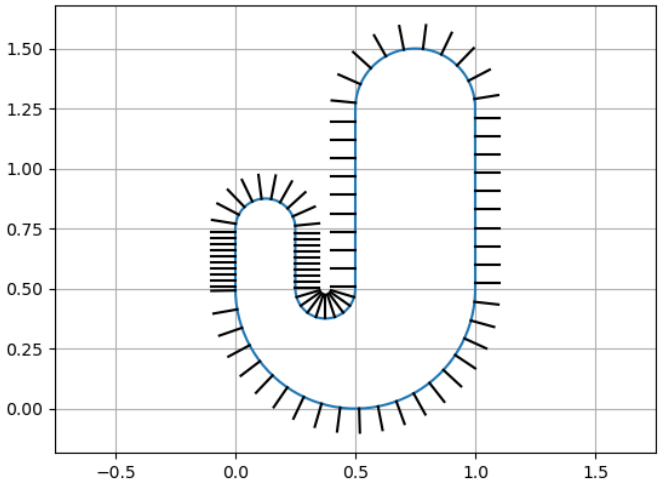
Граница S области Ωзадается объединенным набором восьми сеточных кривых, который из точки x=0, y=0.5 задает последовательный обход границы S в положительном направлении. Для каждого отрезка сеточной границы S задается его срединная точка, совокупность которых определяет сетку границы S_N задачи Неймана. В каждой из срединных точек сетки S_N возводится внешний

перпендикуляр к отрезку сетки S, что задает единичный вектор нормали $\mathbf{n}_S$ в срединной точке. Полагая U(x,y) гармоническому тестовому полиному

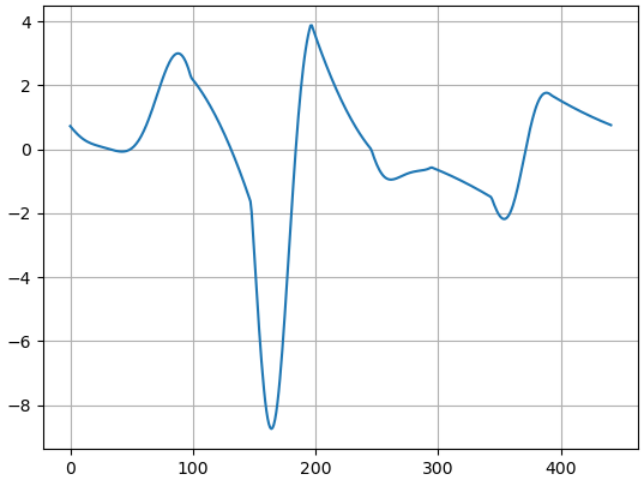
$$U(x,y) = x^3 - 3 \cdot xy^2, \tag{21}$$

имеем

$$gradU(x,y) = (3x^2 - 3y^2, -6xy)$$
$$\psi(S_N) = gradU(S_N) \cdot n_S(S_N) \tag{22}$$

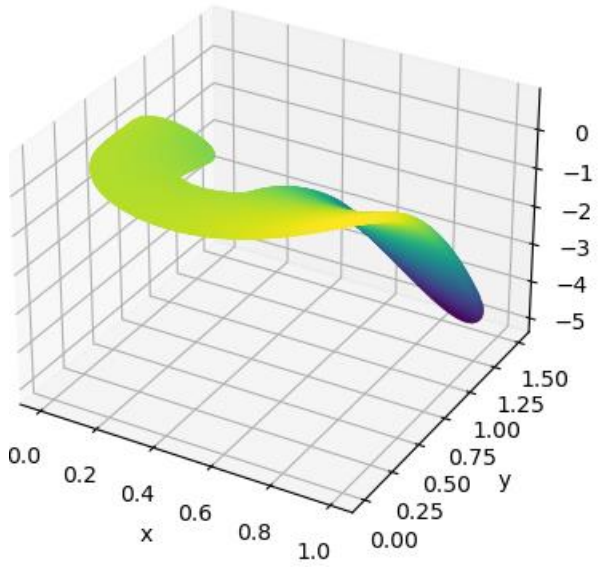


Граница S_N Неймана с каждой пятой внешней каждой пятой внешней нормалью длиной 0.1

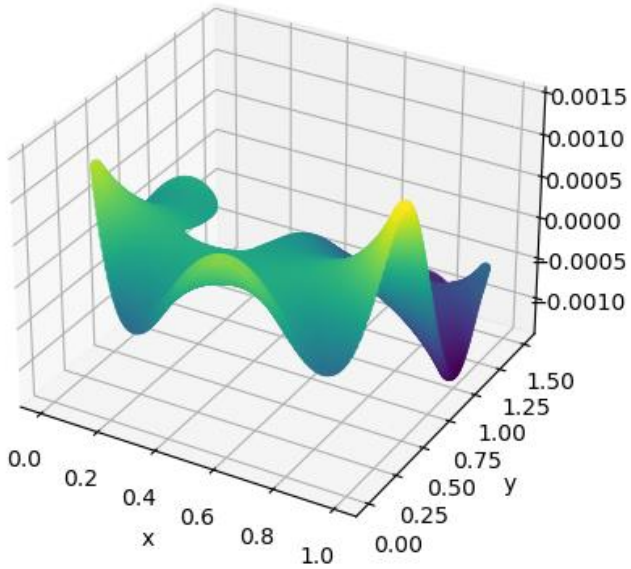


Граничная функция $\psi(S_N)$ (22)

Графики решения задачи Неймана для U(x,y) (21) следующий вид:



Точное U и KAN-решение задачи Неймана



Разность $U - u$точного и KAN-решения краевой задачи Неймана

Здесь точные функции и их KAN-аппроксимации, включая граничную фунцию $\psi(S_{N,})$, на графиках неотличимы.

Следует иметь в виду, что полученное решение является неопределенным с точностью до константы C, найти которую можно, находя решение внешней задачи Неймана и предполагая, что

оно на бесконечности равно нулю. В нашем случае такая константа была получена усреднением отклонения KAN-решения u(x,y) от известного точного решения U (x,y):

$$C = \frac{1}{L}\sum_{l=1}^{L}|U_l - u_l| \, .$$