Сравнение решений экранированного уравнения Пуассона Методом Конечных Элементом и Нейронной Сетью Глубокого Обучения

Михневич Роман ИНБИКСТ 803 группа

Постановка Задачи

Имеется дифференциальное уравнение в частных производных вида:



С условием задачи Коши:



Где f(x,y) - произвольная гладкая функция, оператор Дельта - лапласиан.

Исследование:

1. Решить задачу Методом Конечных Элементов и сравнить результат с верным решением.
2. Спроектировать и обучить физическую нейросеть (PiNN), считающую функцию u, и сравнить с верным решением и решением по МКЭ.
3. Сравнить затраты времени и ресурсов вычислительной машины на оба способа решения.

Метод Конечных Элементов

Для реализации МКЭ был выбран язык PascalABC.Net по причине наличия в нем вспомогательных библиотек динамической 3D графики, многочленов и решения СЛАУ. Для ускорения работы метода были написаны библиотеки разреженных матриц и решения разреженной матрицы методом Холецкого.

В качестве триангуляции была выбрана Триангуляция Делоне, реализованная “по построению”. Это значительно упростило создание триангуляции, но отняло главное преимущество МКЭ - возможность распределить точки неравномерно. За базисную функцию была взята пирамидальная высотой 1.

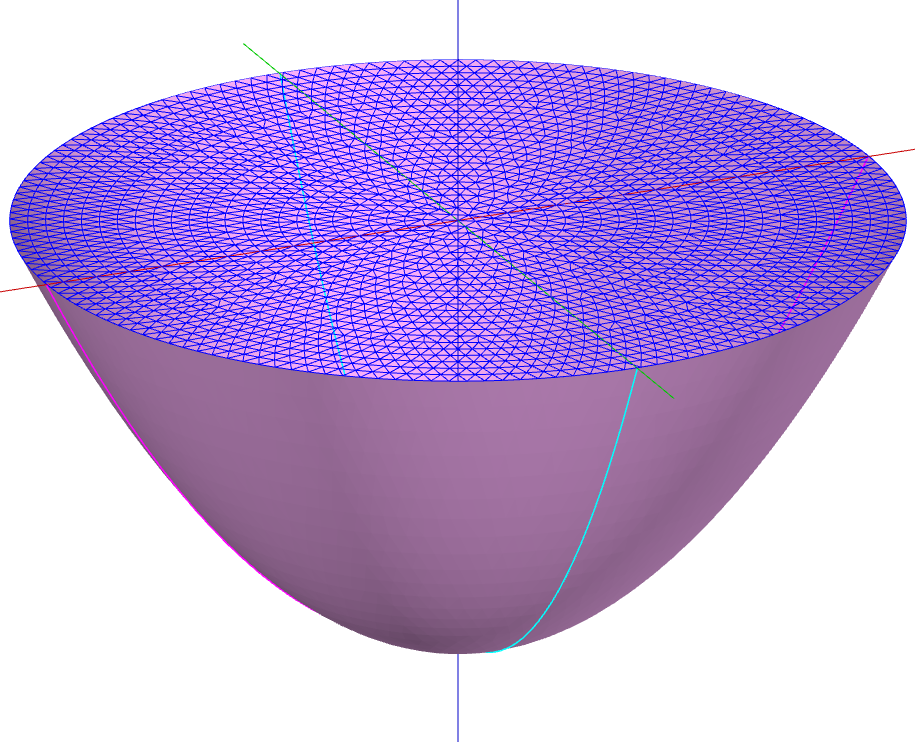
Для заполнения матрицы СЛАУ использовался уникальный подход: вместо точки за базовый объект шага был взят треугольник. Это позволило сократить вычисления за счет того, что квадрат базиса всегда одинаков внутри треугольника независимо от того, какой это базис.

Время решения матрицы 1300х1300 составляет порядка 2 секунд, однако время решения растет около кубически с ростом ее размерности. Так, на матрице 2300х2300 время решения составляет 7.8 секунд.

Точность решения (сумма ошибки по всем узлам) составляет порядка 0.05, однако данная точность заведомо неверна для расчета по причине того, что МКЭ теряет в точности из-за аппроксимации треугольниками. Эти потери сильно зависят от количества точек разбиений и кривизны функции f.

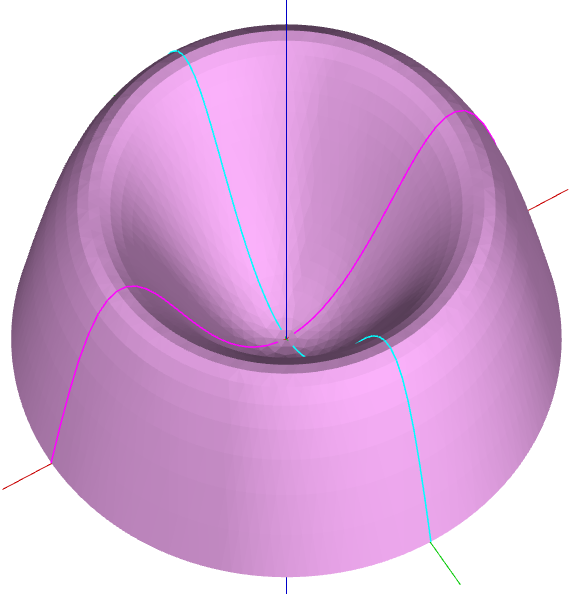
Результат Работы

F = X^2+Y^2-5

U = X^2+Y^2-1

F = Sin(Pi\*(X^2 + Y^2)) - 4\*Pi\*Cos(Pi\*(X^2 + Y^2)) + 4\*Pi^2\*X^2\*Sin(Pi\*(X^2 + Y^2)) + 4\*Pi^2\*Y^2\*Sin(Pi\*(X^2 + Y^2))

U = sin(PI\*(X^2+Y^2))



Проблемы реализации МКЭ

Во время написания программы был обнаружен ряд симптомов, которые могут встречатся в будущих проектах и являющихся сложными для дебаггинга:

1. “Просадка вдоль прямой”: Функция неверно рассчитывается вдоль прямой y=x либо y=-x, “проседая” на ней. Причина: Неверный расчет разбиения треугольника при вычислении добавки к вектору правой части.
2. “Вдавленные треугольники”: Треугольники около узлов +6 степени выглядят выпуклыми, а -6 вогнутыми. Причина: неверная связь между базисными функциями и точками. Из-за этого интеграл добавляется к неверной точке и происходит своеобразный “поворот” произведений базисов. Обнаруживается тестами на функциях вида отличного от f(x^2+y^2)
3. “Нерассчитываемые точки”: Некоторые точки на границе считаются как NaN несмотря на то, что деления на ноль не происходит. Причина: Выход счетчиков за границы массива. Проблема обнаруживается на этапе компиляции, но при дополнительной ошибке в виде учета точек на краю обращается в логическую ошибку и практически не обнаружима деббагингом.

Физическая Нейронная Сеть

Для создания нейронной сети был выбран язык python с библиотекой TensorFlow 2.0 как наиболее развитой для создания NN на данный момент. Модель нейронной сети представляет собой полносвязную сеть со слоями 2-(tanh)-10-(tanh)-10-1, которая представляет собой искомую функцию u(x,y). Для решения второго уравнения в модель был добавлен предпоследний слой -(sigmoid)-10-

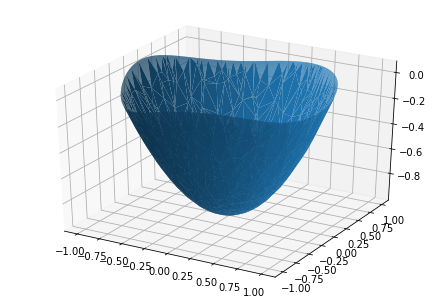
Основным преимуществом PiNN является то, что для решения задач Коши не требуется подбирать выборку. Функция потерь всегда может быть подобрана так, что подав на вход любую допустимую точку функция потерь будет сводится к нулю. В нашем случае функция потерь состоит из двух компонент: Потери на Условии Коши и Потери на Заданной Функции. Условие Коши в нашем случае проверяется тем, что на границе области PiNN возвращает ноль. Потери на Заданной Функции рассчитываются как абсолютная разность правой и левой частей уравнения. Таким образом мы можем использовать сеть для создания гладкой функции, приближающей искомую с высокой точностью без необходимости решать исходное уравнение.

Основным недостатком такого подхода стоит отметить скорость обучения. За 5 минут сеть способна очень точно рассчитать центральный массив точек, но край (даже при высоком коэффициенте при Потери на Краю) расчитывается как околосинусоидальная функция.

Выигрыш в обучении возникает при периодическом изменении набора точек. Так, заменяя набор точек каждую третью эпоху можно добится как очень быстрого начального снижения функции потерь, так и высокой точности на протяжении всего обучения. При этом точки внутри круга стоит определять полностью случайным образом, а на окружности лишь задавать случайное смещение равномерного распределения точек, поскольку так Нейронная Сеть допускает меньше ошибок на границе.

Результаты Работы

F = X^2+Y^2-5

U = X^2+Y^2-1

F = Sin(Pi\*(X^2 + Y^2)) - 4\*Pi\*Cos(Pi\*(X^2 + Y^2)) + 4\*Pi^2\*X^2\*Sin(Pi\*(X^2 + Y^2)) + 4\*Pi^2\*Y^2\*Sin(Pi\*(X^2 + Y^2))

U = sin(PI\*(X^2+Y^2))

Функция ошибки при реконфигурации сети падала постоянно, однако упала за полчаса всего на 60, достигнув 120, так что для решения этой задачи необходимо более тщательно выбирать конфигурацию сети либо обучать ее на более мощных машинах.

Сравнение методов

Метод Конечных Элементов показал себя гораздо эффективнее по сравнению с Нейронной Сетью, однако исследование, использованное для создания Нейросети говорит, что при расчетах на видеокарте NVidia Titan X за 5000+ эпох по 10000 точек можно добиться точности в четвертом порядке для уравнений в разы сложнее приведенных. Также, при увеличении числа узлов время расчета МКЭ растет кубически, когда время работы PiNN не увеличивается вовсе из-за того, что она рассчитывает гладкую функцию по всему объему, а не только на заданных узлах. Также, если убрать границу при первой функции, 15000 точек триангуляции считаются по МКЭ около 6 часов, в то время как точность решения PiNN за эти же 6 часов может достичь 0.0001.

Ссылки

1. <https://www.sciencedirect.com/science/article/pii/S0021999118307125> - Статья про PiNN
2. <https://colab.research.google.com/drive/1WCaXqf6psRuC1RwMgIotSko6MfE0nB1I?usp=sharing> - Ссылка на PiNN (Google Colab)
3. Ссылка на PascalABC.Net проект (Github)