Physics Informed Neural Network

Buckley Laverette

Выполнил: Комаров Николай
Группа: 5040102/40201

Проблематика

$$Buckley - Laverette\ eq.: \frac{\partial S_w(x,t)}{\partial t} + \frac{\partial f(S_w(x,t))}{\partial x} = 0$$

Начальные Условия: $S_w(x,0) = S_0$

Граничные Условия: $S_w(a,t) = S_a,$

$S_w(b,t) = S_b$

- Отсутствует аналитическое решение
- Численные методы – имеют высокую сложность реализации, требуют тонкой настройки
- Как получить решение ???

Проблематика

Что мы знаем – Начальное распределение и Граничные Условия. Всё, что между ними – ***неизвестно***

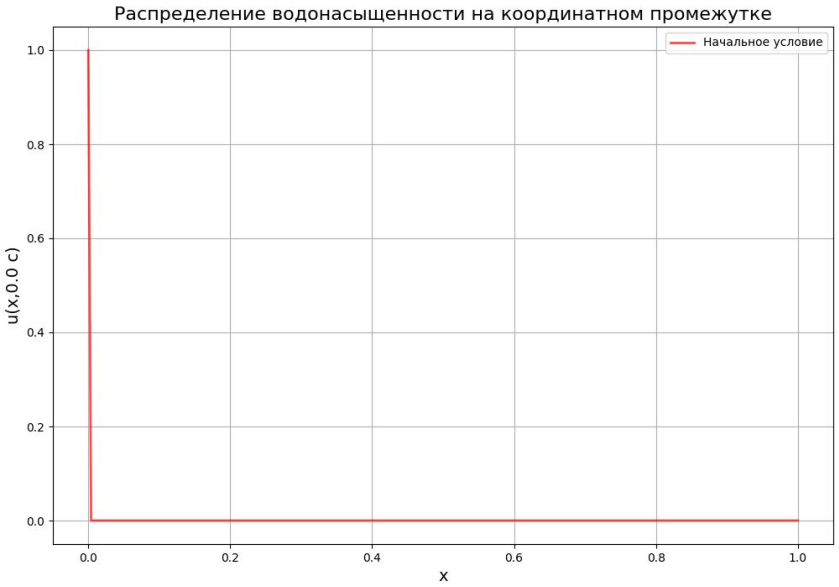


Рис. 1 Начальное условие

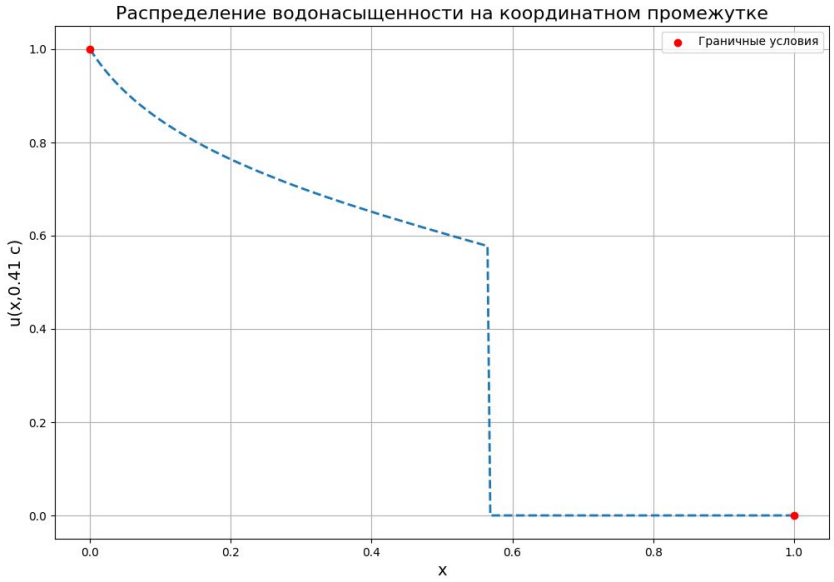


Рис. 2 Граничные условия

Цели

- Построить ***предиктивную модель*** на основе методов машинного обучения для вычисления значений водонасыщенности на координатном промежутке

- Использовать для нахождения значений функции заданные ***начальные*** и ***граничные*** условия

- Реализовать ***web***–приложение

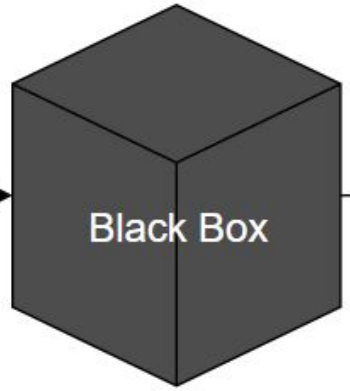
Какие подходы использовать

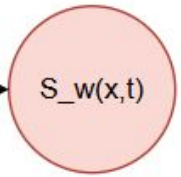
- Стандартные методы машинного обучения не подходят

значения функции в промежутке [x1, x2] ***неизвестны*** *– обучаться не на чем*

- Нейронные сети:

	x	t	S_w
initial	[0..1]	0	[1..0]
left_bound	0	[0..1]	[1..1]
right_bound	1	[0..1]	[0..0,8]





Архитектура Нейронной Сети

Нормализация

x = [a;b] → [0;1]:

```
normalized_data[col] = (data[col] - min_val) / (max_val - min_val)
```

Генерация коллокационных точек

Data = { (x=0, t=0) ; (x=1, t=1) } → { (x=0, t=0) ; (x=0.1, t=0) … (x=0.1, t=0.1) … (x=1, t=1) }

```
lb = X_setka.min(axis=0) # [x=0, t=0]
ub = X_setka.max(axis=0) # [x=1, t=1]
X_collocate = lb + (ub - lb) * lhs(2, num_points)
```

Архитектура Нейронной Сети

Нейронная сеть с последовательными слоями

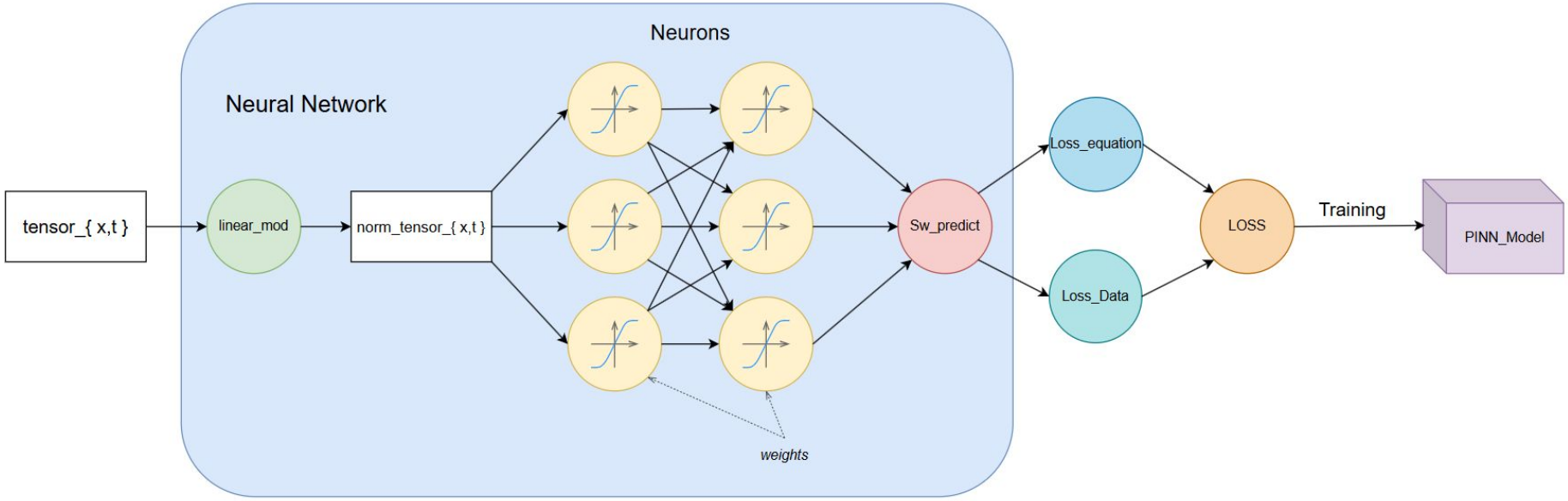
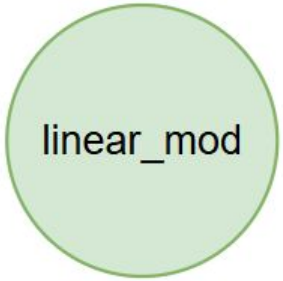


Рис. 1 Архитектура Нейронной Сети

Архитектура Нейронной Сети

Преобразование данных

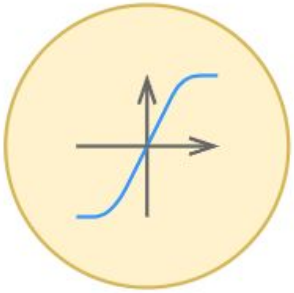


Линейное преобразование данных для оптимальной работы нейронов

{ (0,0) … (0,1) } → { (-1,-1) … (1,1) }

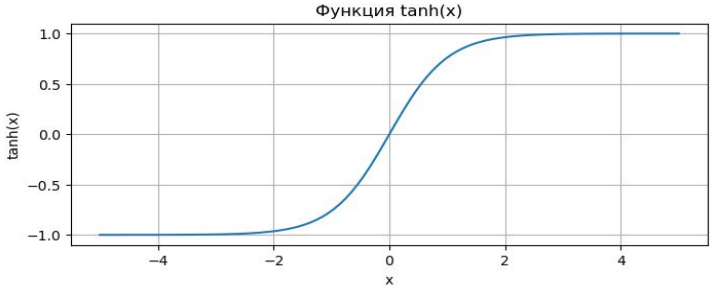
```
self.u_model.add(tf.keras.layers.Rescaling(scale=2.0, offset=-1.0))
```

Активация нейронов



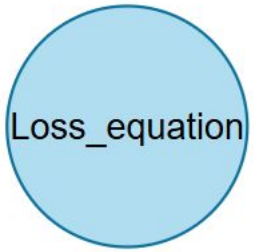
```
for width in layers[1:-1]:
 self.u_model.add(tf.keras.layers.Dense(
 width,
 activation=tf.nn.tanh,
kernel_initializer=tf.keras.initializers.GlorotNormal(),
 bias_initializer='zeros'))
```

“хорошие” производные →



Архитектура Нейронной Сети

Ошибка дифф. уравнения



$$\frac{\partial U_{w\_predict}(x,t)}{\partial t} + \frac{\partial f(U_{w\_predict}(x,t))}{\partial x} = Loss_{equation} \; ? \; 0$$

```
f_model:
u = self.u_model(X_f)
u_t = tape.gradient(u, self.t_f)
f_u_x = tape.gradient(f_u, self.x_f)
return u_t + f_u_x
Loss_eq = tf.reduce_mean(tf.square(self.f_model()))
```

Ошибка по Н.У. и Г.У.

Используем MSE (среднеквадр-я ошибка):

```
u_pred = self.u_model(X_u) # полученные зн-я из модели
data_loss = tf.reduce_mean(tf.square(u - u_pred)) # u - зн-я
водонасыщенности в Н.У. и Г.У.
```

Архитектура Нейронной Сети

Ошибка для обучения Нейронной сети


```
Loss = weight_eq * Loss_equation + weight_data * Loss_Data

total_loss = loss_weights[0] * Loss_Data + loss_weights[1] *
   loss_equation
```

Обучение Нейронной сети

Оптимизатор Adam

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla LOSS$$

$$a_t = \beta_2 a_{t-1} + (1-\beta_2)(\nabla LOSS)^2$$

коррекция моментов:

$$\tilde{m}_t = \frac{m_t}{1-\beta_1^t}$$

$$\tilde{a}_t = \frac{a_t}{1-\beta_2^t}$$

обновление весовых коэффициентов:

$$w_t = w_{t-1} - \eta \frac{\tilde{m}_t}{\sqrt{\tilde{a}_t} + \epsilon}$$

η – шаг обучения

m_t - скользящее среднее относительно среднего градиента

a_t – скользящее среднее относительно среднеквадратичного градиента

β_1, β_2 – коэффициенты импульса

Обучение Нейронной сети

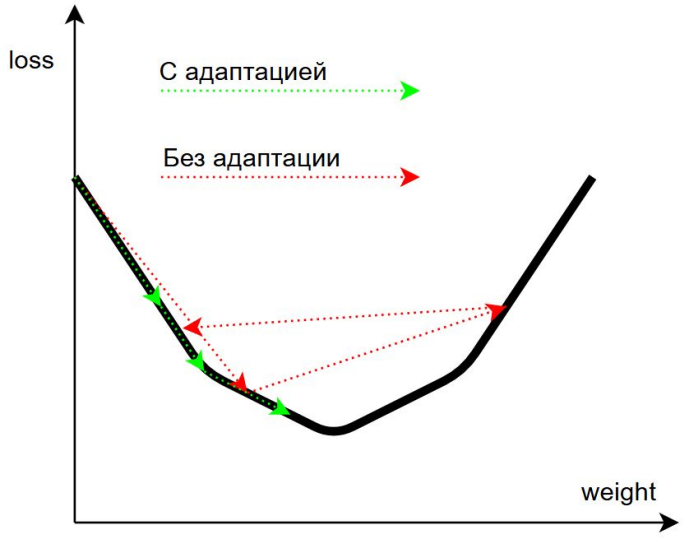
Уменьшение шага обучения. Экспоненциальное затухание

$$\eta = \eta_0 * (decay\_rate)^{\frac{step}{decay\_step}}$$

```
lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=1e-2,
    decay_steps=2000,
    decay_rate=0.95)
```

Параметры оптимизатора Адам

```
tf_optimizer = tf.keras.optimizers.Adam(
    learning_rate=lr_schedule,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-7)
```



Архитектура Нейронной Сети

Обучение Нейронной сети

На каждом шаге обучения получаем ошибку, берем градиент и обновляем веса в нейронах согласно конфиг-ии оптимизатора:

```
def train_step(self, X_u, u, loss_weights=None):

    with tf.GradientTape() as tape:
         total_loss, data_loss, pde_loss = self.loss_fn(X_u, u, loss_weights)

    grads = tape.gradient(total_loss, self.__wrap_training_variables())

    self.optimizer.apply_gradients(zip(grads,
    self.__wrap_training_variables()))

    return total_loss, data_loss, pde_loss
```

Движемся по эпохам обучения:

```
for epoch in range(tf_epochs):

    # Определяем веса ошибок для текущей эпохи
    current_weights = get_loss_weights_for_epoch(epoch,
loss_weights_schedule)

    total_loss, data_loss, pde_loss = self.train_step(X_u, u,
current_weights)
```

Компоненты приложения

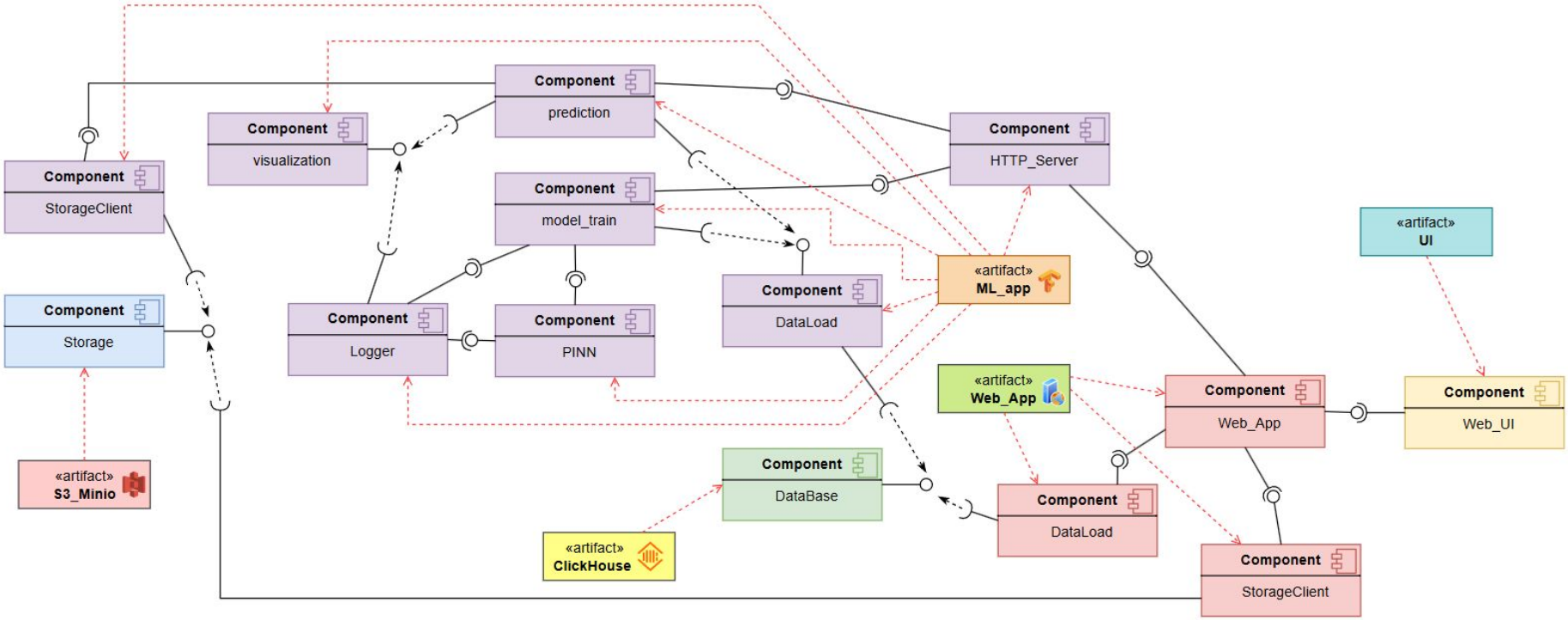


Рис. 2 Компоненты

Архитектура Приложения

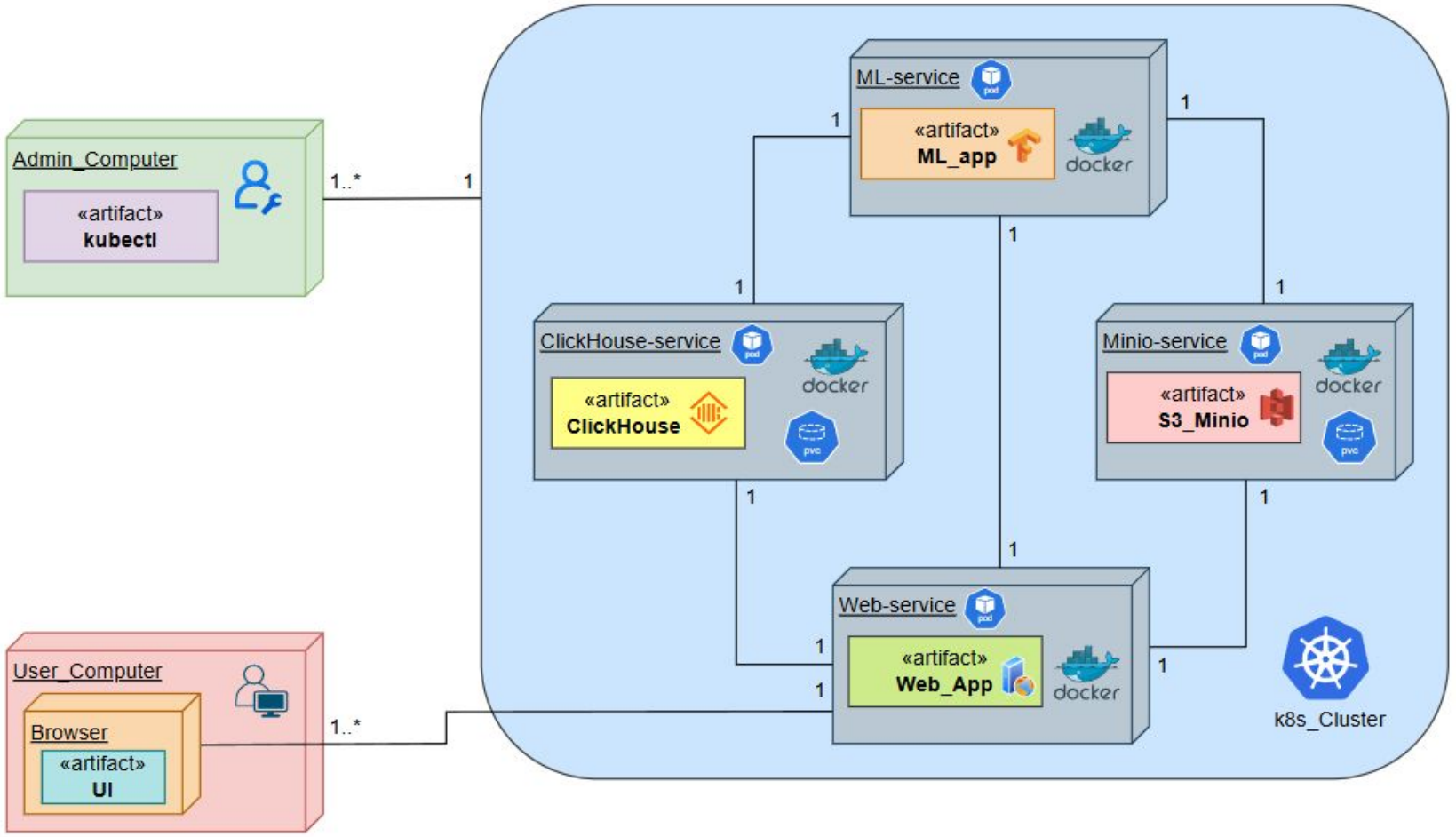


Рис. 3 Размещение

Результаты

Графики ошибки по эпохам

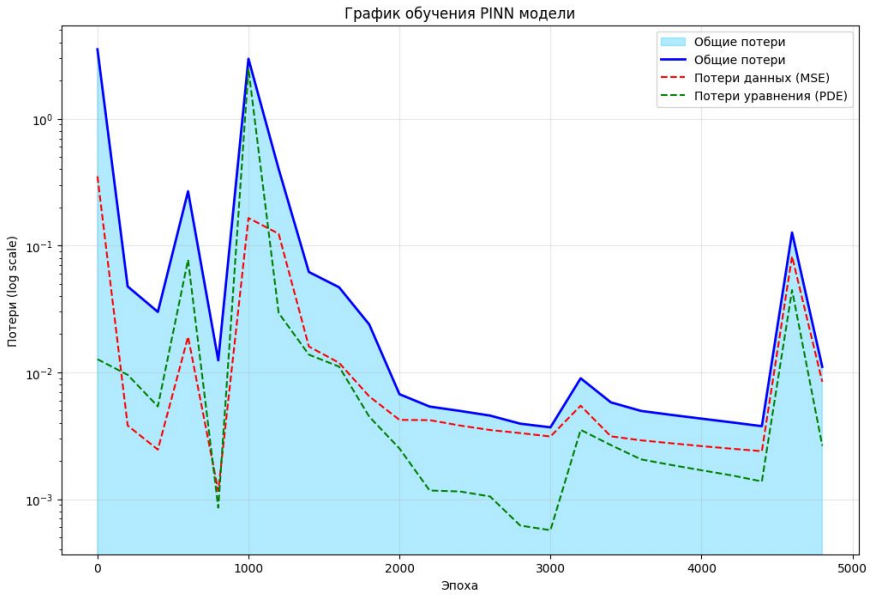


Рис. 4 Размещение
5000 эпох
5 слоев, 10 нейронов

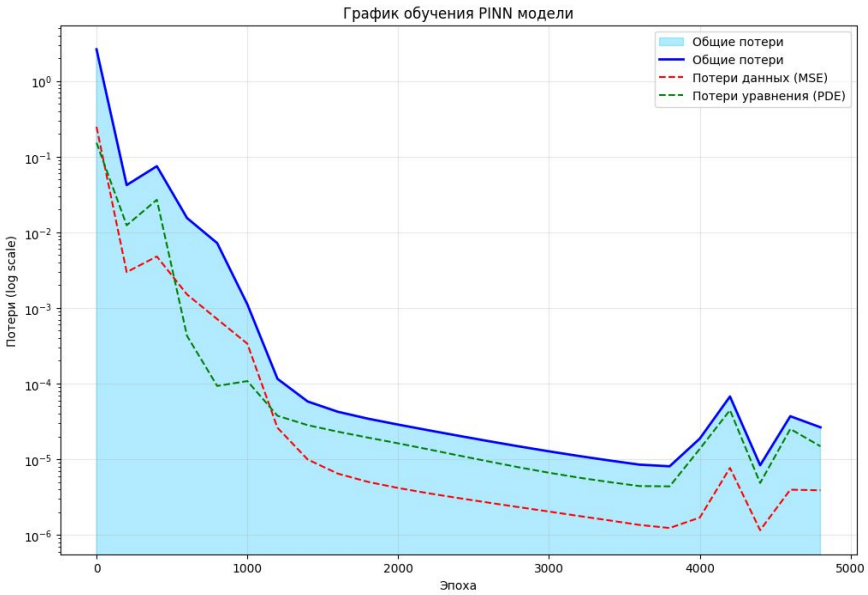


Рис. 5 Размещение
5000 эпох
10 слоев, 20 нейронов

Результаты

Полученная функция распределения

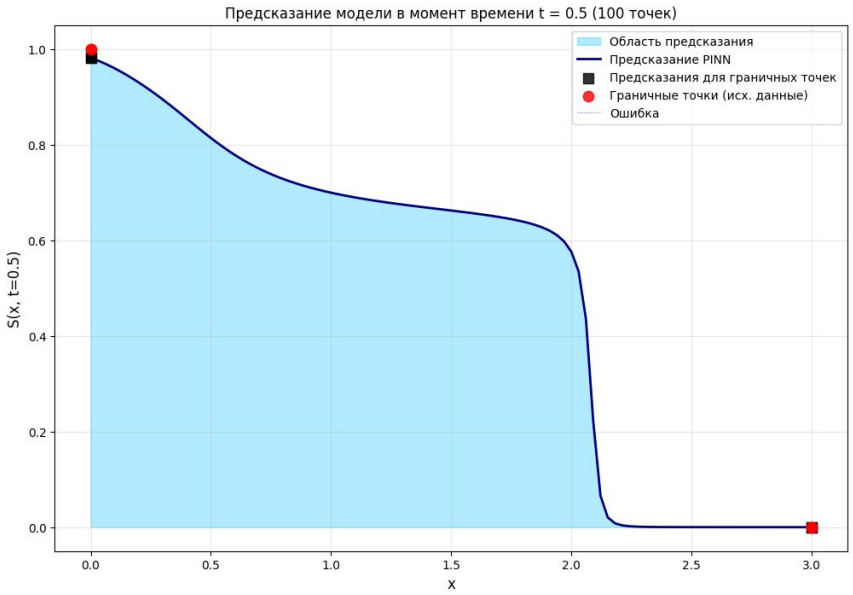


Рис. 6 График распределения предсказанной функции в момент времени t=0.5

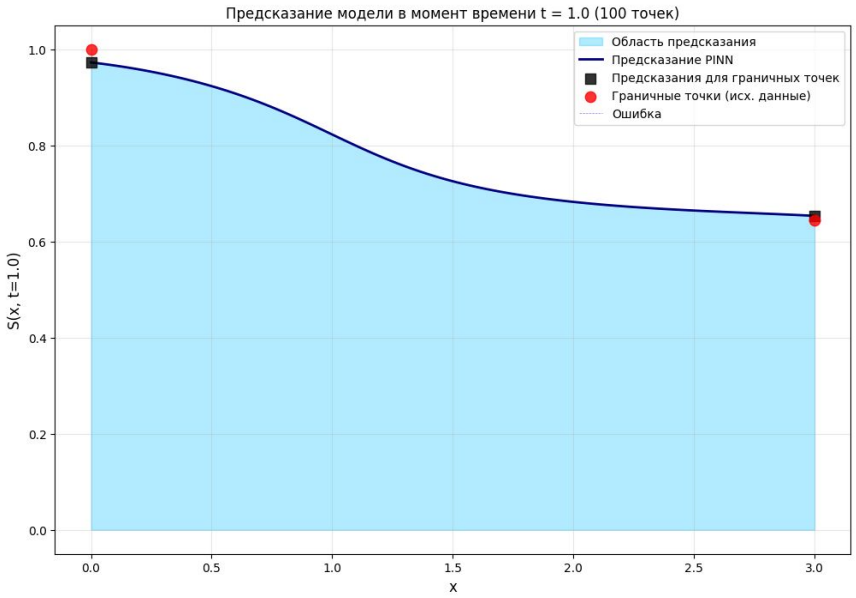


Рис. 7 График распределения предсказанной функции в момент времени t=1.0

Результаты для известного распределения

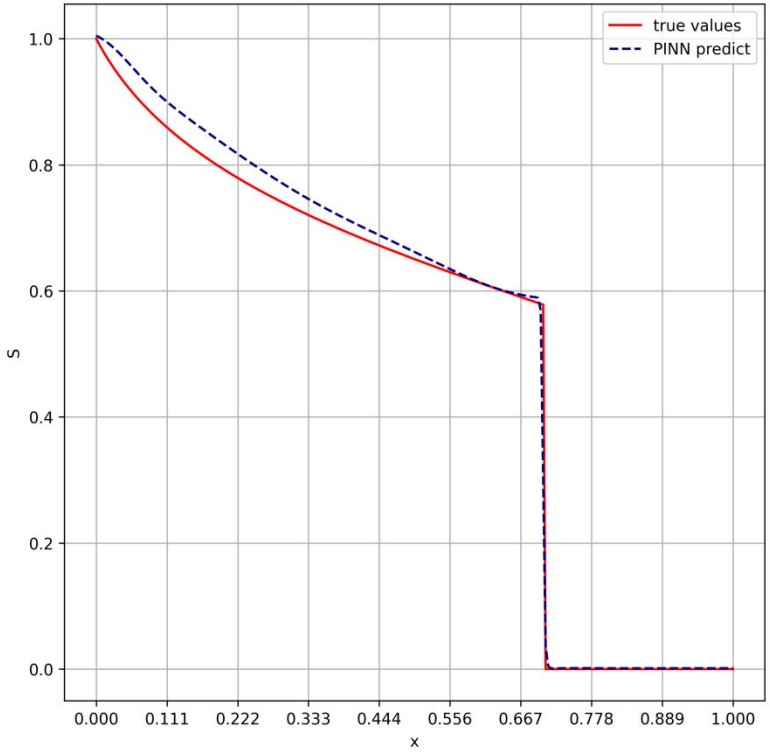


Рис. 8 График распределения предсказанной функции в момент времени t=0.5

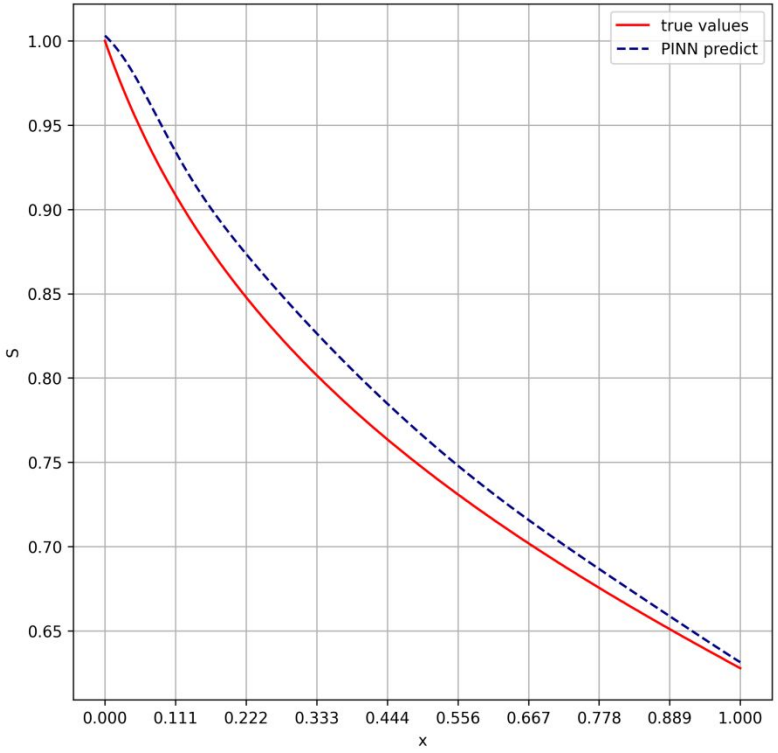


Рис. 9 График распределения предсказанной функции в момент времени t=0.5

Источники

Tensorflow:

- https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

Preparation Data:

- https://www.geeksforgeeks.org/data-analysis/different-types-of-data-sampling-methods-and-techniques/
- https://datasciencegenie.com/latin-hypercube-sampling-vs-monte-carlo-sampling/

PINN Buckley-Laverette:

- (PDF) Data-Free and Data-Efficient Physics-Informed Neural Network Approaches to Solve the Buckley-Leverett Problem
- https://www.researchgate.net/publication/346559662_Buckley-Leverett_Theory_for_Two-Phase_Immiscible_Fluids_Flow_Model_with_Explicit_Phase-Coupling_Terms

K8s:

- https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

ClickHouse:

- https://clickhouse.com/docs/integrations/python

S3 Minio:

- https://pypi.org/project/minio/

FastApi:

- https://fastapi.tiangolo.com/