Projekt 2.2 – Hail cannon simulation using PINN

Mateusz Cyganek, Adrian Chrobot

Zmodyfikowany kod

Stan początkowy nie był zmieniany.

Podobnie jak w wersji cpp, od obliczanej wartości odejmowane są pochodne funkcji działka – `bx()` przemnożone przez `dfdx()`, oraz `by()` przemnożone przez `dfdy()`.

```
 1.     def residual_loss(self, pinn: PINN):
 2.         x, y, t = get_interior_points(
 3.             self.x_domain, self.y_domain, self.t_domain,
 4.             self.n_points, pinn.device())
 5.         loss = dfdt(pinn, x, y, t).to(device)\
 6.             - self.dTy(y, t) * dfdy(pinn, x, y, t).to(device)\
 7.             - self.Kx * dfdx(pinn, x, y, t,order=2).to(device)\
 8.             - self.Ky * dfdy(pinn, x, y, t, order=2).to(device)\
 9.             - self.source(y,t).to(device)\
10.             - bx(pinn, x, y, t).to(device) * dfdx(pinn, x, y, t).to(device)\
11.             - by(pinn, x, y, t).to(device) * dfdy(pinn, x, y, t).to(device)
12.
13.         return loss.pow(2).mean()
14.
```

Dodane linijki zaznaczono kolorem zielonym.

Kod pochodnych działka

Funkcje `bx()` oraz `by()`, powstały analogicznie do funkcji `f()`, którą zastąpiono funkcją `cannon()`.

Funkcje wywołując funkcje `cannon()`, a następnie obliczają pochodną względem `y` lub `x`.

Podobnie jak w wersji cpp, stałe mocy działka pozwalając zwiększać lub zmiejszać wartości zwracanego tensora.

```
1. def bx(pinn: PINN, x: torch.Tensor, y: torch.Tensor, t: torch.Tensor, order: int = 1):
2.     f_value = cannon(pinn, x, y, t)
3.     return df(f_value, y, order=order) * cannon_strength_x
4.
5. def by(pinn: PINN, x: torch.Tensor, y: torch.Tensor, t: torch.Tensor, order: int = 1):
6.     f_value = cannon(pinn, x, y, t)
7.     return df(f_value, y, order=order) * cannon_strength_y
8.
```

Kod działka

- Kod funkcji działka zwraca tensor dla podanych tensorów `x`, `y` i `t`.
- Zmienna `zero_tensor`, to tensor o wartościach zero,
- Zmienna `valid_i_denom` Tworzy tensor wartości logicznych, sprawdzając, czy i_denom jest większe od 0.
- W linijce 9 ustawiana jest wartość `time` na `0.0` w miejscach, gdzie jest `valid_i_denom` fałszywe.
- W linijce 11 `valid_y = y <= time` tworzy tensor wartości logicznych, sprawdzając, czy `y` jest mniejsze lub równe `time`.

Obliczamy w którym momencie wystrzału jesteśmy i możemy przez rozmiar siatki by traktować `time` jako współrzędną osi Y. Jeżeli `y > time` znaczy to tyle, że y jest nad falą wystrzału, zatem powinno zwrócić tam wartość 0.

0 zostanie zwrócone w przypadku gdy y jest nad y', lub gdy kąt α rozchodzenia się fali jest więszky od zdefiniowanego maksymalnego kąta.

Jeżeli wszystkie te warunki są spełnione to ostatecznie zwracany tensor to różnica między y a y', pomnożona przez cosinus α kąta.

```
 1. def cannon(pinn: PINN, x: torch.Tensor, y: torch.Tensor, t: torch.Tensor) -> torch.Tensor:
 2.     iter = t
 3.     zero_tensor = torch.zeros_like(iter)
 4.
 5.     i_denom = torch.tensor((iterations / wave_speed) - cannon_shot_time, device=iter.device)
 6.     valid_i_denom = i_denom > 0
 7.
 8.     time = (iter - cannon_shot_time) * grid_size / i_denom
 9.     time = torch.where(valid_i_denom, time, zero_tensor)
10.
11.     valid_y = y <= time
12.
13.     x_prim = torch.abs(cannon_x_loc - x)
14.     alpha_rad = torch.atan(x_prim / time)
15.     valid_alpha = alpha_rad < max_alpha
16.
17.     y_prim = torch.sqrt(time * time - x_prim * x_prim)
18.     valid_y_prim = y <= y_prim
19.
20.     cos_term = torch.cos(alpha_rad * cone_limiter * 0.5)
21.
22.     cannon_value = (y_prim - y) * cos_term
23.     valid_condition = valid_i_denom & valid_y & valid_alpha & valid_y_prim
24.     cannon_value = torch.where(valid_condition, cannon_value, zero_tensor)
25.
26.     return cannon_value
27.
```

Stałe działka

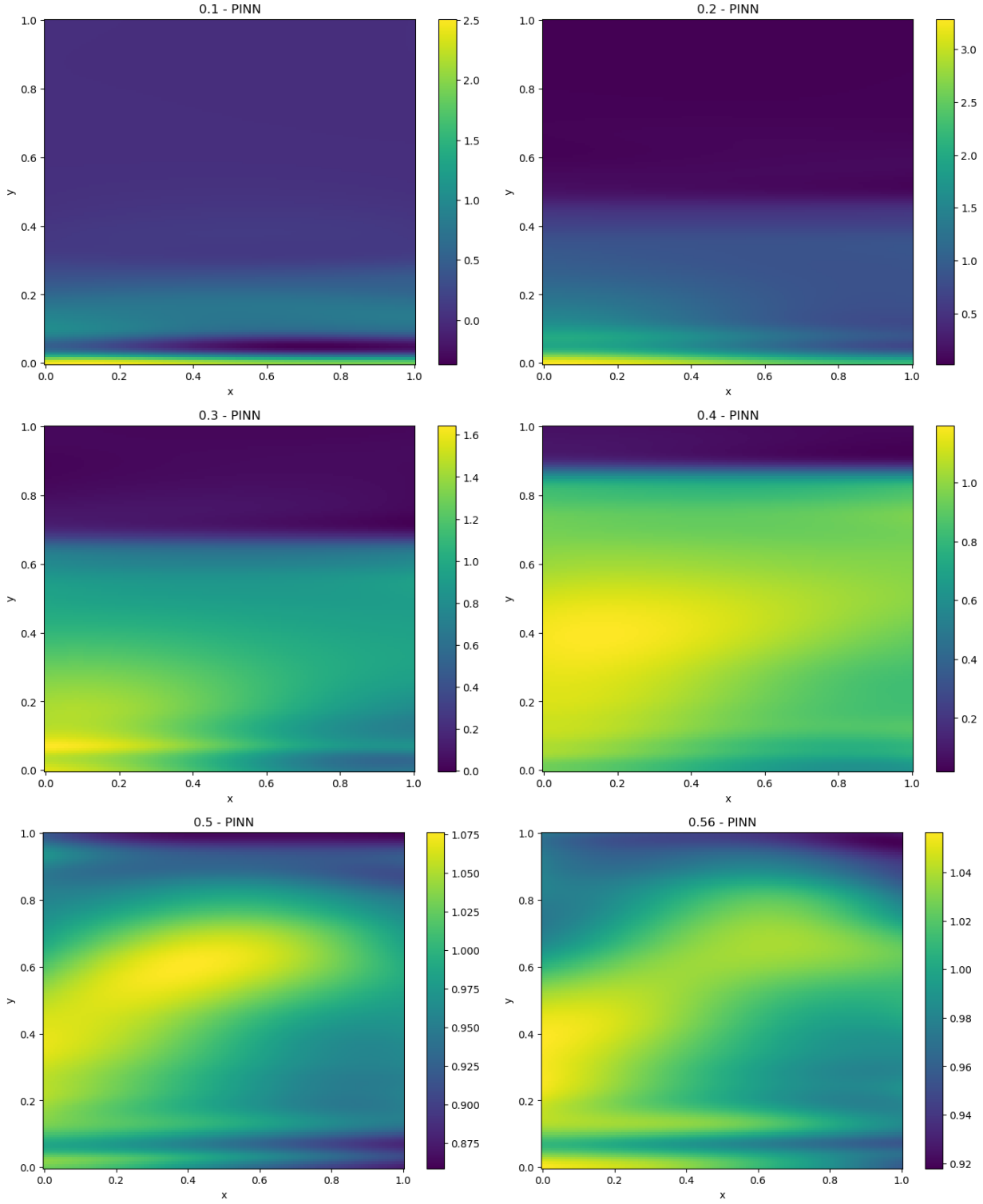
Stałę działka wykorzystywane przez funkcje działka.

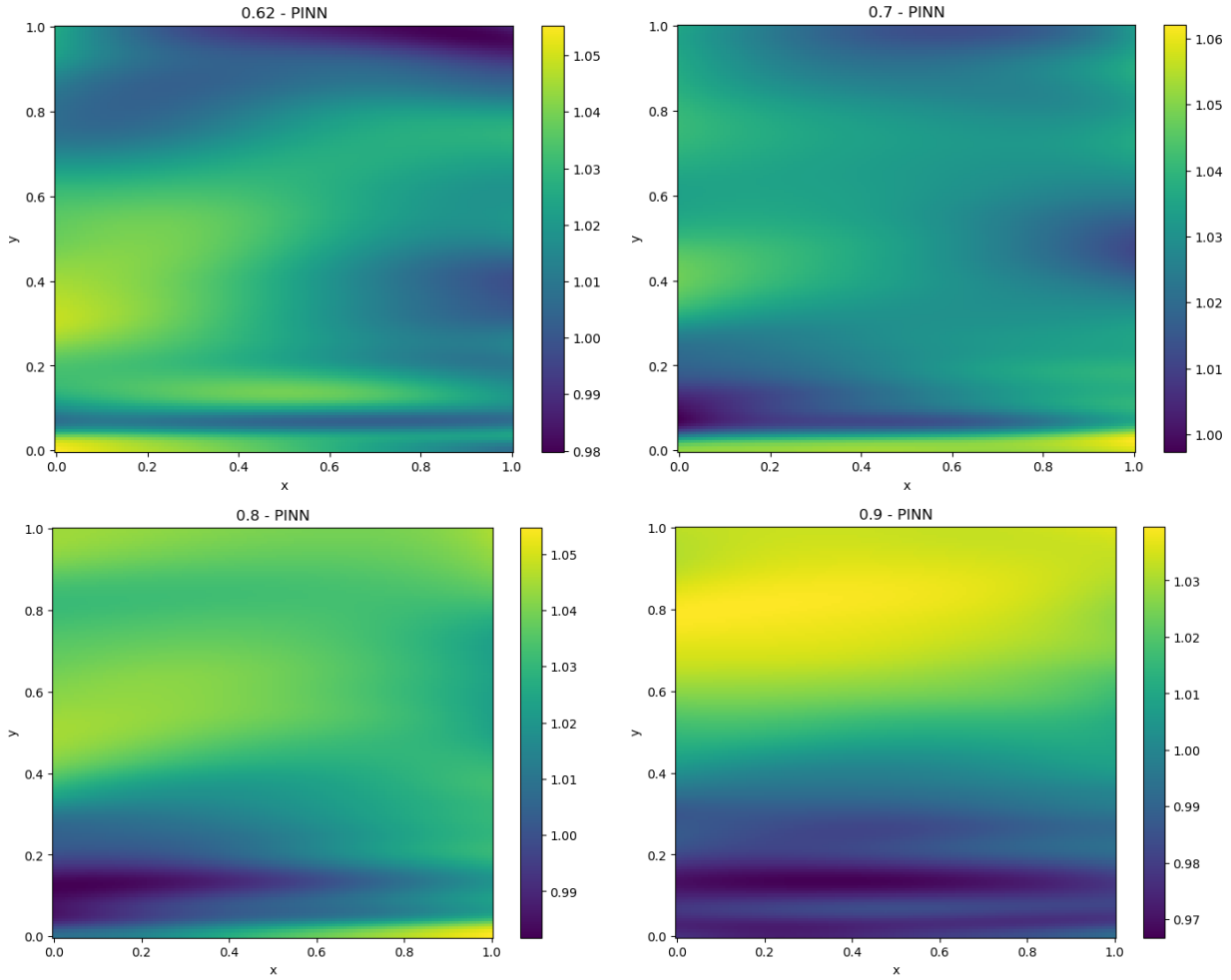
```
grid_size = 1.0
cannon_x_loc = grid_size / 2
cannon_shot_time = 0.4
cannon_strength_x = 3.5
cannon_strength_y = 4.5
cone_limiter = 6.0
max_alpha = np.pi / cone_limiter
wave_speed = 2.0
iterations = 1.0

```

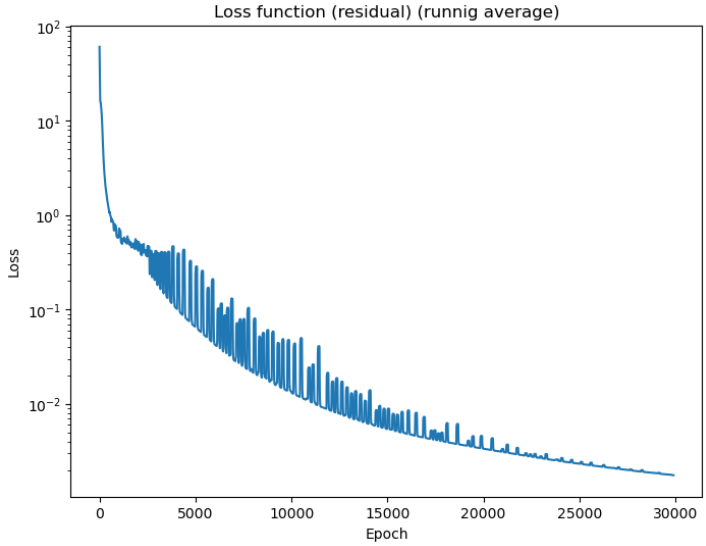
- `grid_size` – rozmiar siatki, w której przeprowadzana jest symulacja,
- `cannon_x_loc` – lokalizacja działa na osi X, w symulacji działo znajduje się na środku siatki,
- `cannon_shot_time` – klatka, w której działo strzela,
- `cannon_strength_x` – mnożnik siły działa w osi X,
- `cannon_strength_y` – mnożnik siły działa w osi Y,
- `cone_limiter` – zmienna ograniczenia maksymalnego kąta α rozchodzenia się wystrzelonej fali, równego π / `cone_limiter`,
- `wave_speed` – szybkość rozchodzenia się fali.

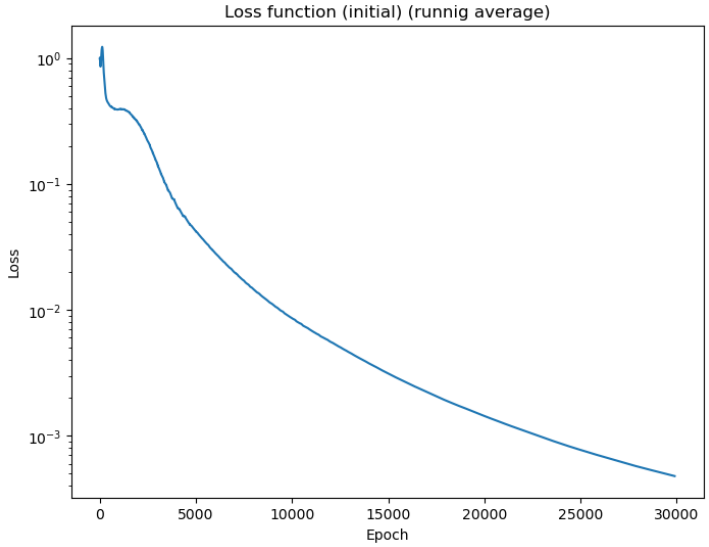
Uzyskany efekt

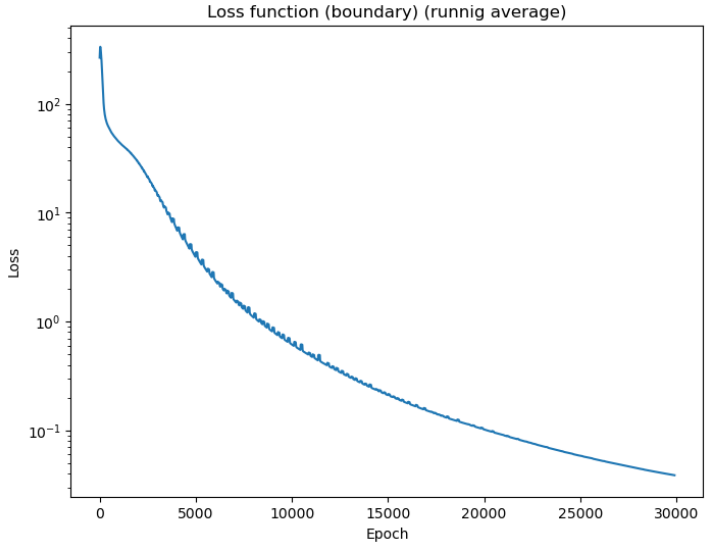




Wykresy zbieżności procesu uczenia




Loss function (initial) (runnig average)
Loss
Epoch
10^{0}
10^{-1}
10^{-2}
10^{-3}
0
5000
10000
15000
20000
25000
30000


Loss function (boundary) (runnig average)
Loss
Epoch
10^{2}
10^{1}
10^{0}
10^{-1}
0
5000
10000
15000
20000
25000
30000