Wprowadzenie do obliczeń numerycznych dla równań różniczkowych stochastycznych oraz wykorzystanie obliczeń zrównoleglonych

Natalia Czyżewska, Marcelina Studzińska-Wrona
Wydział Matematyki Stosowanej,
Grupa Stochastycznej Analizy Numerycznej

12.01.2021

Równanie różniczkowe

zwyczajne (ODE)

Równanie różniczkowe zwyczajne pierwszego rzędu w postaci normalnej oraz zagadnienie Cauchy'ego (problem początkowy – Initial Value Problem)

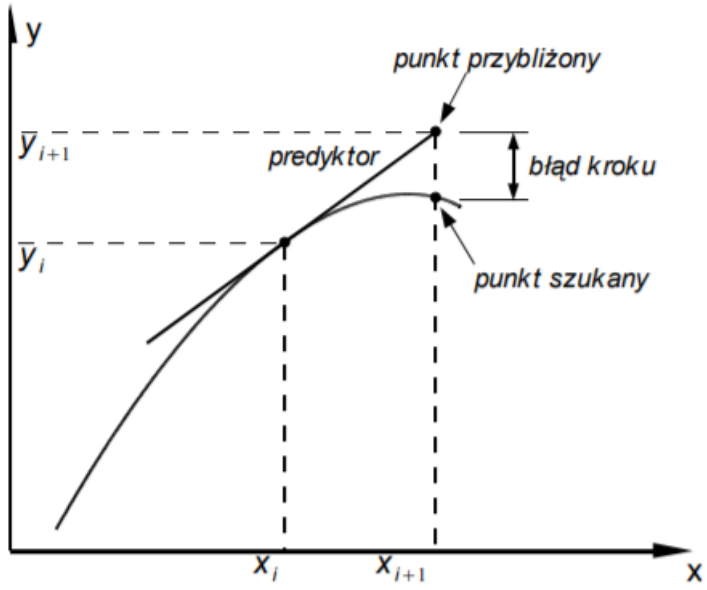
$$\frac{dy}{dx} = f(x,y)$$

$$y(x_0) = y_0$$

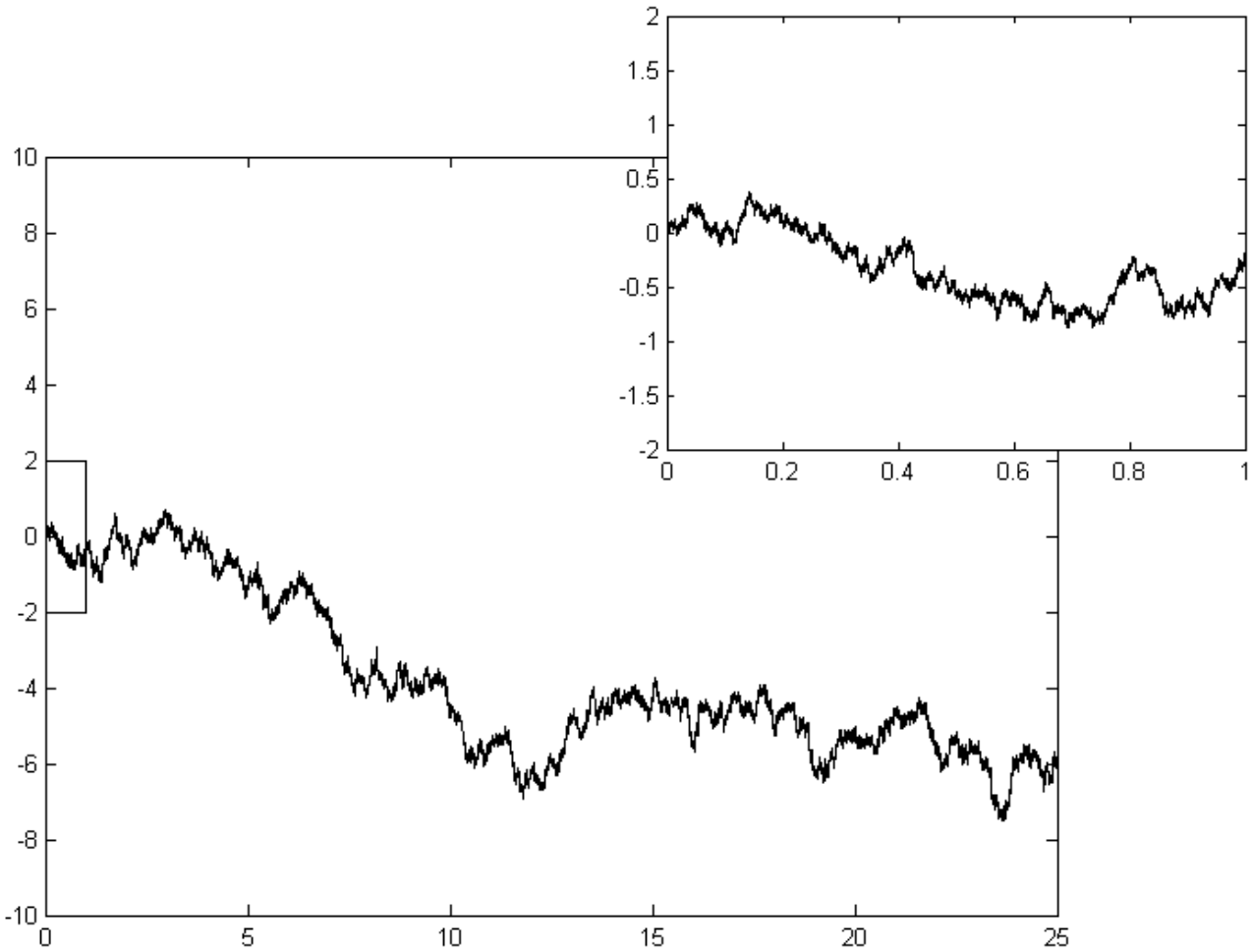
Metoda Eulera

$$y_{i+1} = y_i + f(x_i, y_i)h$$

$$h = x_{i+1} - x_i$$



Proces stochastyczny



Rys. Ruch Browna – proces Wienera

Równanie różniczkowe

stochastyczne (SDE)

dryf dyfuzja

$$\begin{cases} dX(t) = a(t, X(t-))dt + b(t, X(t-))dW(t) \\ X(0) = \xi, \end{cases}$$

Rów. Stochastyczne równanie różniczkowe

$$\begin{cases} dX(t) = a(t, X(t-))dt + b(t, X(t-))dW(t) + c(t, X(t-))dN(t) \\ X(0) = \xi, \end{cases}$$

Rów. Stochastyczne równanie różniczkowe ze skokami

Metody numeryczne

równań różniczkowych stochastycznych

Co to jest proces Wienera?

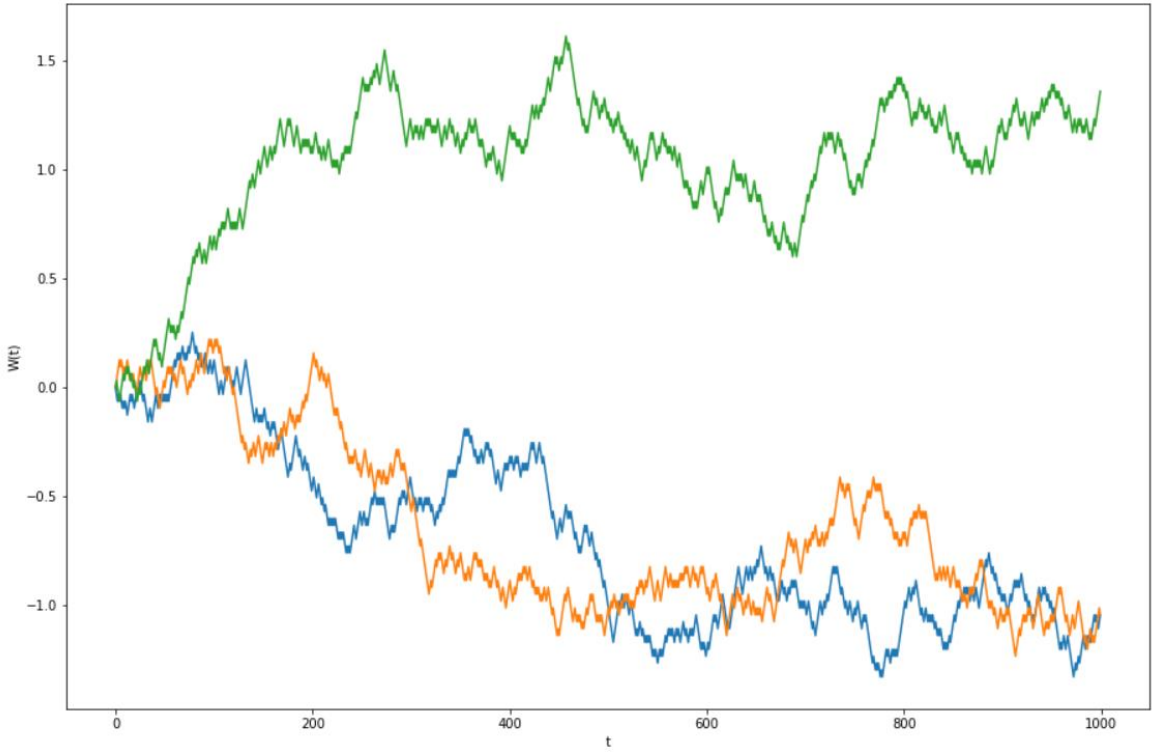
Cechy procesu Wienera:

1. $W_0 = 0$

2. Przyrosty $W_t - W_s$ są niezależne oraz mają rozkład

N(0, t-s), gdzie t>s.

3. Proces jest ciągły, ale nieróżniczkowalny.



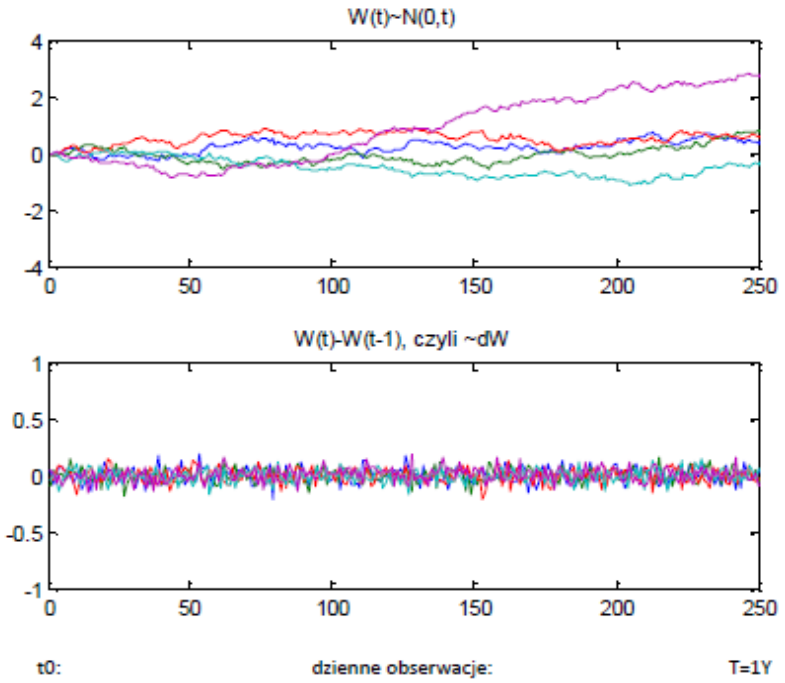
Metody numeryczne

równań różniczkowych stochastycznych

Jak generować pojedynczą trajektorię procesu Wienera?

- Chcemy symulować wartości $(W(t_1), \ldots, W(t_n))$ w ustalonym zbiorze punktów $0<t_1<\ldots<t_n$.
- Niech $Z_1, \ldots, Z_n$ będą niezależnymi zmiennymi losowymi N(0,1).
- Dla standardowego procesu Wienera W(0) = 0 oraz

$$W(t_{i+1}) = W(t_i) + (t_{i+1} - t_i)^{1/2} \cdot Z_i$$



Metody numeryczne

równań różniczkowych stochastycznych

```python
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_theme()

def plot_wiener(T,n):
    x=np.zeros(n+1)
    y=np.zeros(n+1)
    x[0] = 0.0
    y[0] = 0.0
    h = float(T/n)
    h_sqrt = np.sqrt(h)
    for i in range(n):
        x[i+1] = x[i] + h
        y[i+1] = y[i] + np.random.normal(0, h_sqrt)
    plt.plot(x,y)
    plt.show()
T = float(input("Długość przedziału T : "))
n = int(input("Liczba punktów siatki N : "))
plot_wiener(T,n)
```

```
Długość przedziału T : 1.2
Liczba punktów siatki N : 1000
```

Metody numeryczne

równań różniczkowych stochastycznych

$$X_n^E(t_{k+1}) = X_n^E(t_k) + a(U_{k,n}^E) \cdot \Delta t_k + b(U_{k,n}^E) \cdot \Delta W_k + c(U_{k,n}^E) \cdot \Delta N_k,$$

$$U_{k,n}^E = (t_k, X_n^E(t_k)),$$

Rów. Schemat Eulera dla stochastycznego równania różniczkowego

UWAGA: To jest przybliżenie tylko jednej realizacji trajektorii!

Metody numeryczne

równań różniczkowych stochastycznych

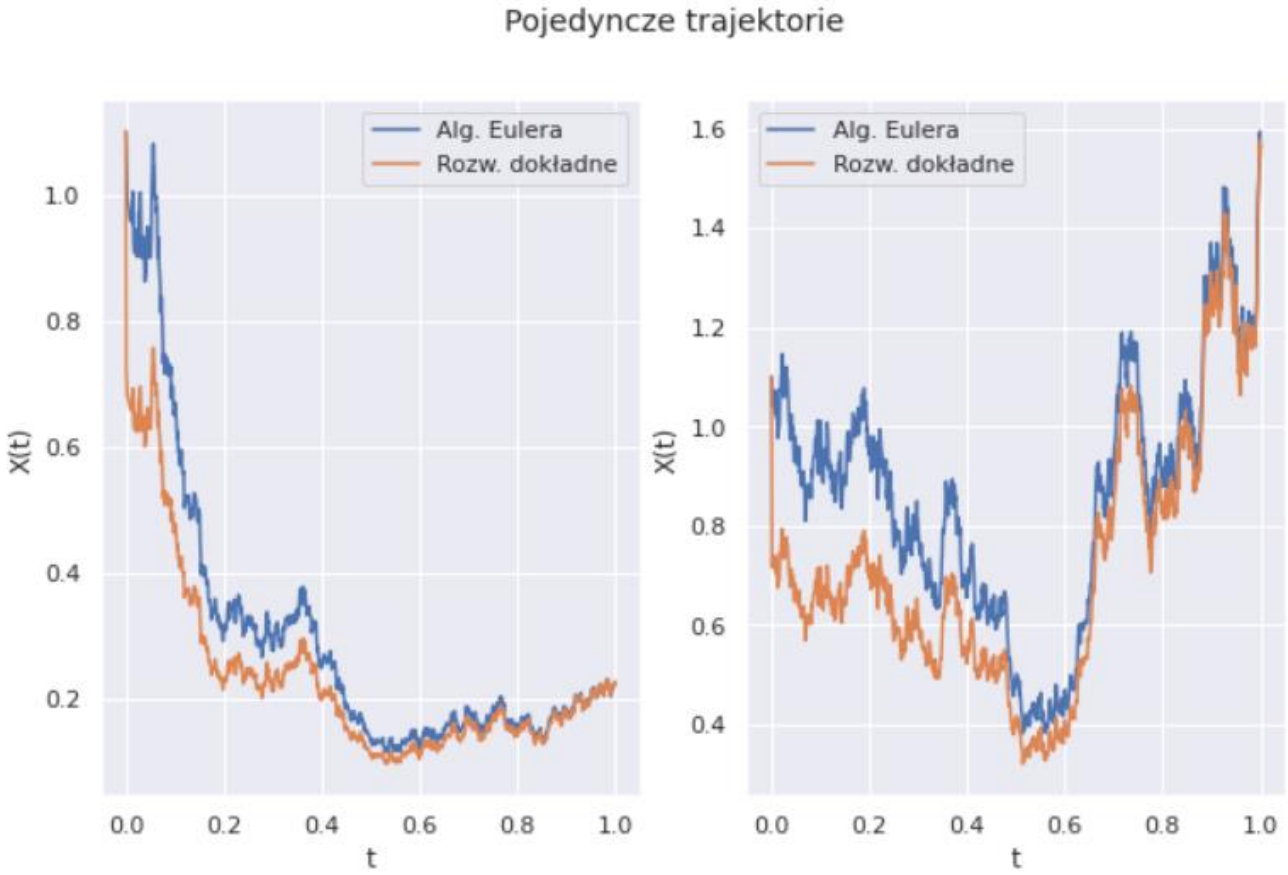
```
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_theme()

def plot_wiener(T,n):
    x=np.zeros(n+1)
    y=np.zeros(n+1)
    x[0] = 0.0
    y[0] = 0.0
    h = float(T/n)
    h_sqrt = np.sqrt(h)
    for i in range(n):
        x[i+1] = x[i] + h
        y[i+1] = y[i] + np.random.normal(0, h_sqrt)
    plt.plot(x,y)
    plt.show()
T = float(input("Długość przedziału T : "))
n = int(input("Liczba punktów siatki N : "))
plot_wiener(T,n)
```

```
Długość przedziału T : 1.2
Liczba punktów siatki N : 1000
```

Metody numeryczne

równań różniczkowych stochastycznych

Jak porównywać wynik rozwiązania?



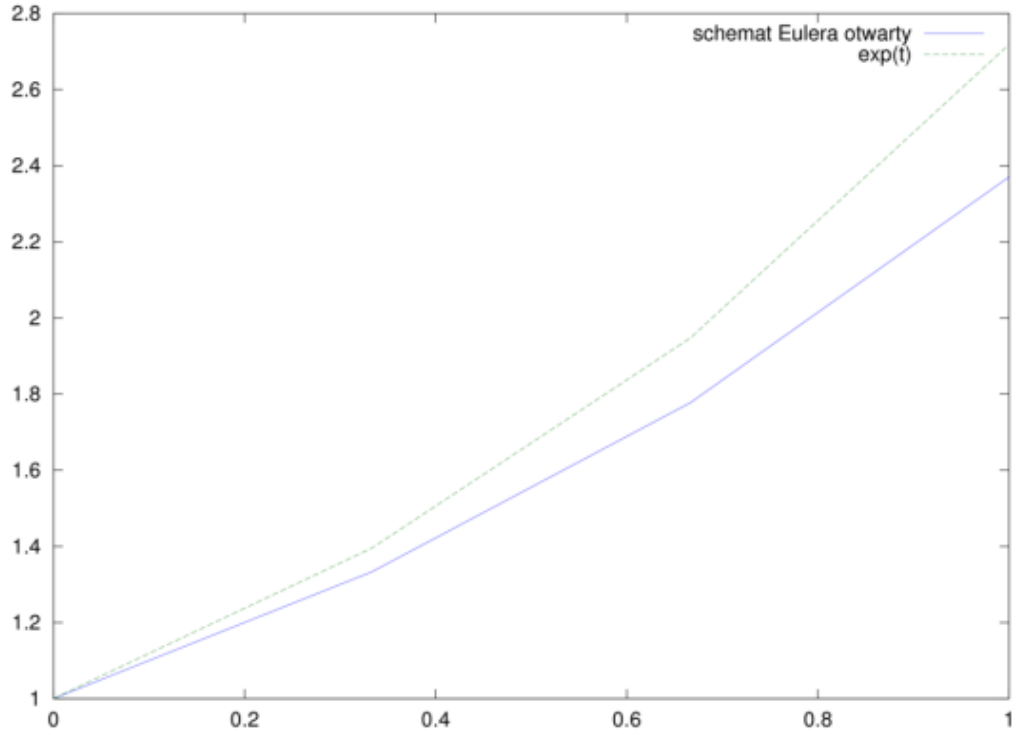
Metody numeryczne

równań różniczkowych stochastycznych

Porównanie wyników dla ODE

$$y_{i+1} = y_i + f(x_i, y_i)h$$

$$h = x_{i+1} - x_i$$

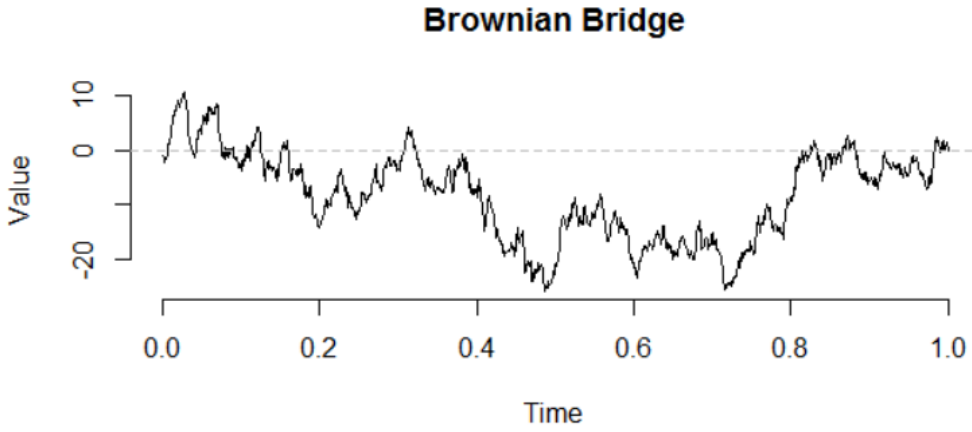


Metody numeryczne

równań różniczkowych stochastycznych

Jak porównywać wynik rozwiązania, kiedy:

- Znamy rozwiązanie dokładne?
- Nie znamy rozwiązania dokładnego?



Niech W będzie procesem Wienera oraz $t_1 < t_2 < t_3$. Wtedy $W(t_2)$ pod warunkiem $W(t_1)$ i $W(t_3)$ ma rozkład normalny $N(\mu, \sigma^2)$, gdzie

$$\mu = \frac{t_3 - t_2}{t_3 - t_1} W(t_1) + \frac{t_2 - t_1}{t_3 - t_1} W(t_3)$$

oraz

$$\sigma^2 = \frac{(t_3 - t_2)(t_2 - t_1)}{t_3 - t_1}.$$

Metody numeryczne

równań różniczkowych stochastycznych

» Istnieje wiele pakietów do aproksymacji równań różniczkowych zwyczajnych

Matlab, Mathematica, C++ i inne

» Aproksymacja równań różniczkowych stochastycznych

Biblioteka cuSTOCH – CUDA C

Podsumowanie

metod numerycznych dla SDE

- Potrafimy opisywać losowe zjawiska oraz aproksymować ich rozwiązania.
- Należy być jednak ostrożnym na losowość, tj.
  - Generowanie wielu trajektorii,
  - Zachowanie własności: generowanie niezależnych procesów Wienera (oraz Poissona),
  - Sensowna ocena wyników: generowanie mostów Browna.
- Możemy wykorzystać przyspieszenie sprzętowe (GPU) do obliczeń równoległych.

Wykorzystanie algorytmów sztucznej inteligencji w wycenie opcji

Natalia Czyżewska, Marcelina Studzińska-Wrona
Wydział Matematyki Stosowanej,
Grupa Stochastycznej Analizy Numerycznej

12.01.2021

Opcja

Prawo do przeprowadzenia w przyszłości transakcji na warunkach ustalonych dziś. Prawa tego nie trzeba realizować. Tak więc opcja daje prawo do sprzedaży/kupna instrumentu bazowego (akcji, obligacji, waluty, papieru wartościowego o stałym oprocentowaniu lub indeksu akcji) w ustalonym terminie w przyszłości po ustalonej cenie realizacji opcji.

Proces ewolucji ceny jest pewnym procesem losowym.

Model dyskretny

Przykład: dwumianowy model wyceny opcji *(Cox, Ross, Rubinstein, „Option pricing: Simplified Approach", Journal of Financial Economics- September 1979)*

Mamy pewne aktywo S, które w chwili początkowej $t = 0$ posiada wartość $S(t = 0) = S_0$. Po czasie T dopuszczamy jeden z dwóch możliwych scenariuszy:

- aktywo drożeje do wartości S_{up} z prawdopodobieństwem p albo
- tanieje do wartości S_{down} z prawdopodobieństwem $1 - p$.

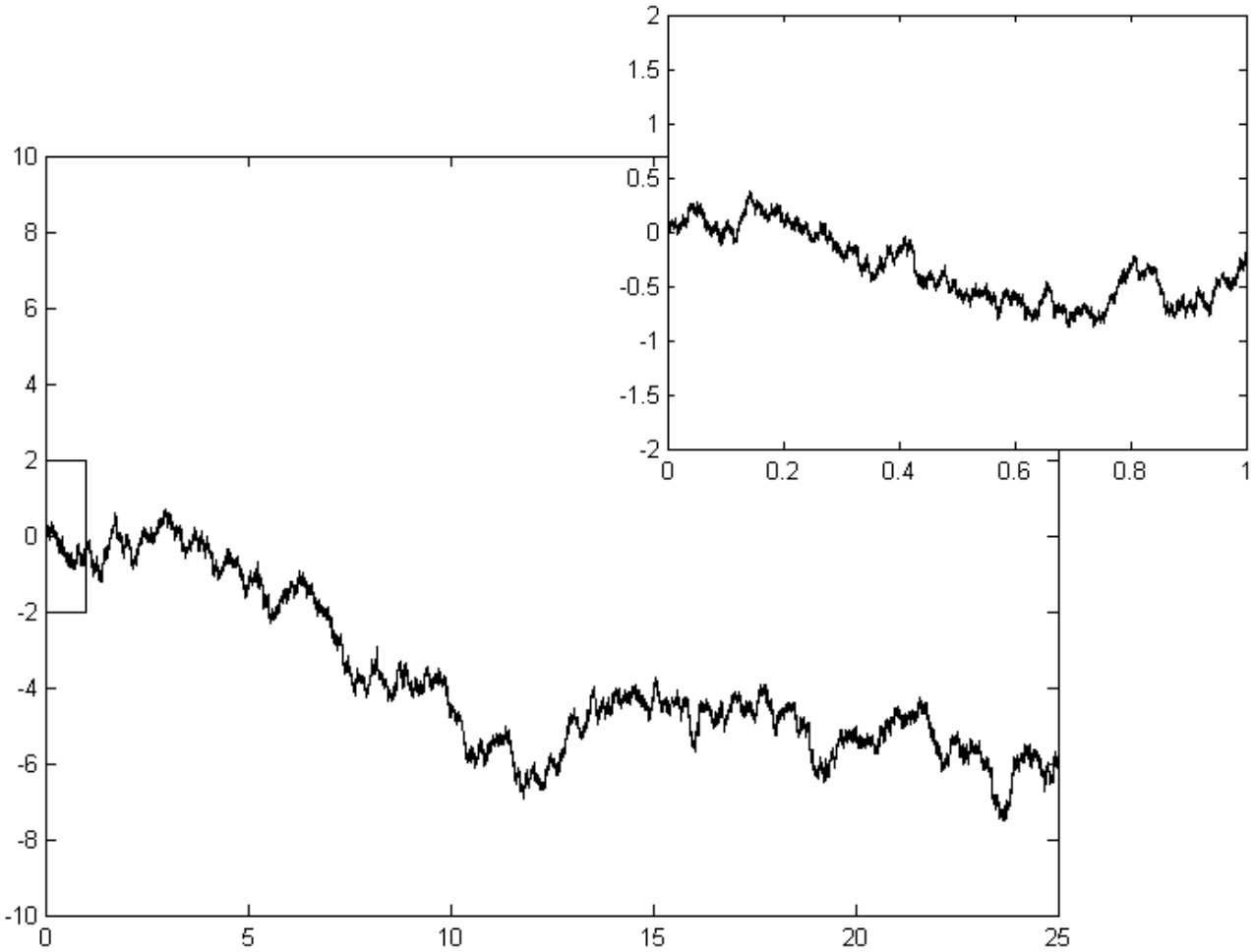
Rynek jest jednookresowy, co oznacza, że rozważamy tylko dwie chwile czasu: początkową: $t = 0$ i przyszłą: $t = T$.

Z drugiej strony po czasie T depozyt bankowy gwarantuje nam, że nasz kapitał będzie wynosił $S_0 e^{rT}$, gdzie r to stopa procentowa.

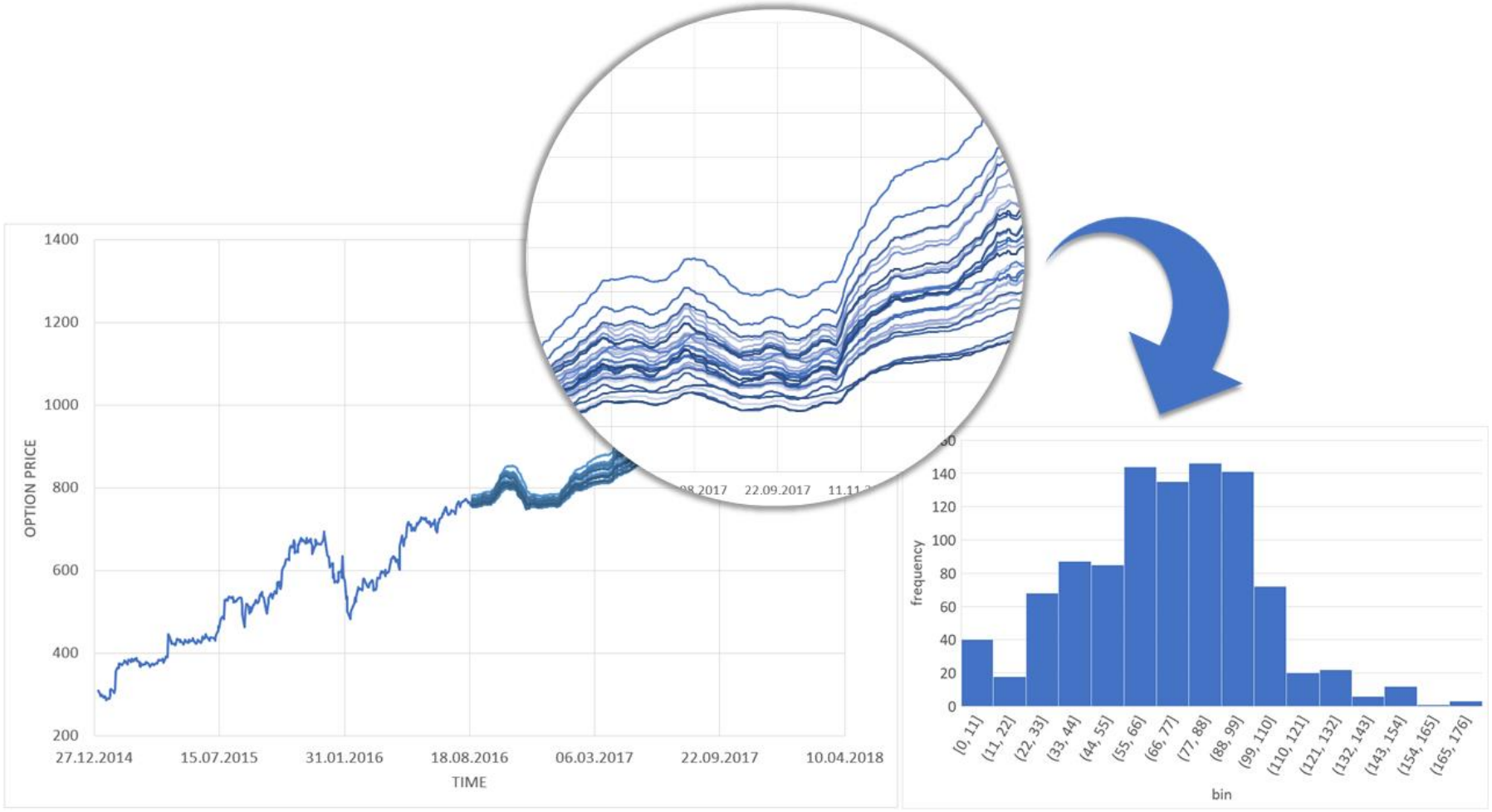
Stąd:

$$pS_u + (1 - p)S_{down} = S_0 e^{rT}$$

Model ciągły



Wycena opcji



Model Mertona

$$\begin{cases} \mathrm{d}X(t) = (r - c_0\lambda)X(t)\mathrm{d}t + \sigma X(t)\mathrm{d}W(t) + c_0 X(t-)\mathrm{d}N(t), \\ \qquad\qquad t \in [0, T], \\ X(0) = 100, \end{cases}$$

Schemat Eulera i zrandomizowany schemat Eulera

$$X_n^E(t_{k+1}) = X_n^E(t_k) + a(U_{k,n}^E) \cdot \Delta t_k + b(U_{k,n}^E) \cdot \Delta W_k + c(U_{k,n}^E) \cdot \Delta N_k,$$

$$U_{k,n}^E = (t_k, X_n^E(t_k)), \qquad X_n^E(0) = \xi,$$

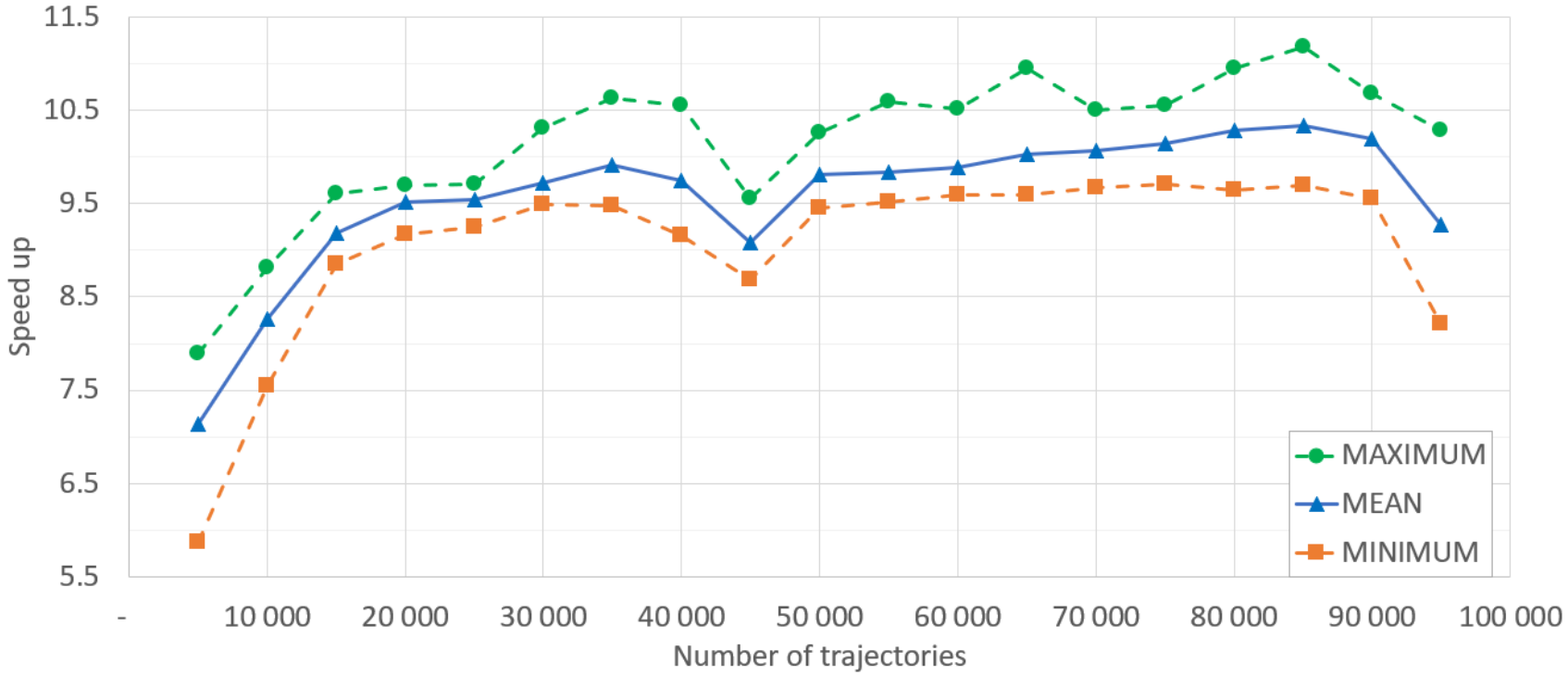
Rów. Schemat Eulera dla stochastycznego równania różniczkowego

$$X_n^{RE}(t_{k+1}) = X_n^{RE}(t_k) + a(\theta_k, X_n^{RE}(t_k)) \cdot \Delta t_k + b(U_{k,n}^{RE}) \cdot \Delta W_k + c(U_{k,n}^{RE}) \cdot \Delta N_k,$$

$$\theta_k = t_k + \tau_k \cdot \Delta t_k, \qquad \tau_k \sim U[0,1]$$
$$U_{k,n}^{RE} = (t_k, X_n^{RE}(t_k)), \qquad X_n^{RE}(0) = \xi,$$

Rów. Zrandomizowany schemat Eulera dla stochastycznego równania różniczkowego

Model Mertona
NVIDIA Titan V vs. Intel Broadwell



Szereg czasowy a RNN

Szereg czasowy to ciąg danych liczbowych, w którym każda obserwacja związana jest z konkretnym momentem w czasie.

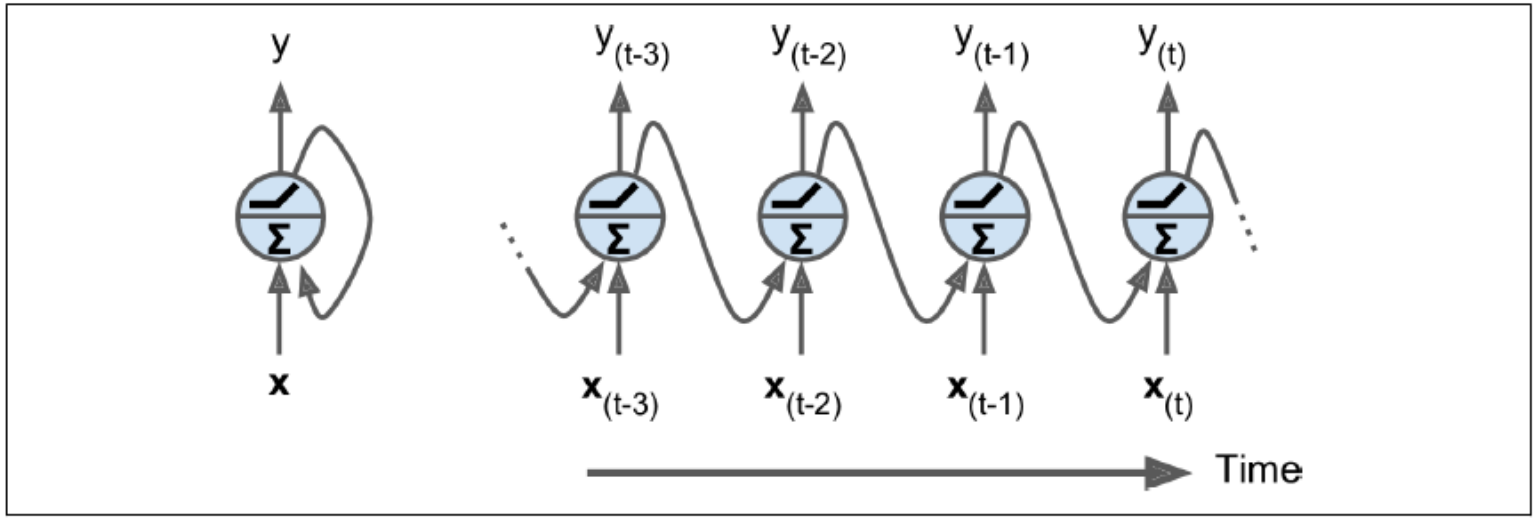
Szereg czasowy to realizacje pewnego procesu stochastycznego.

Sieci RNN - Architektury sieci neuronowych przystowane do przetwarzania danych w dziedzinie czasu.

Ideą rekurencyjnych sieci neuronowych (ang. Recursive Neural Networks, RNNs) jest wykorzystanie informacji sekwencyjnych. W przypadku tradycyjnej sieci neuronowej zakładamy, że wszystkie wejścia (i wyjścia) są niezależne od siebie. Często jest to złe założenie np. chcąc przewidzieć następny wyraz w zdaniu, lepiej zorientować się, które wyrazy pojawiły się przed nim.

Najprostszy neuron RNN

W każdym kroku czasowym t (zwanym również ramką) ten neuron rekurencyjny odbiera wejścia $x_{(t)}$ oraz własne wyjście z poprzedniego kroku czasowego $y_{(t-1)}$.



Z lewej: Rekurencyjny neuron; Z prawej: Rekurencyjny neuron rozwinięty w czasie

źródło: Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 2017.

Dlaczego nie RNN?

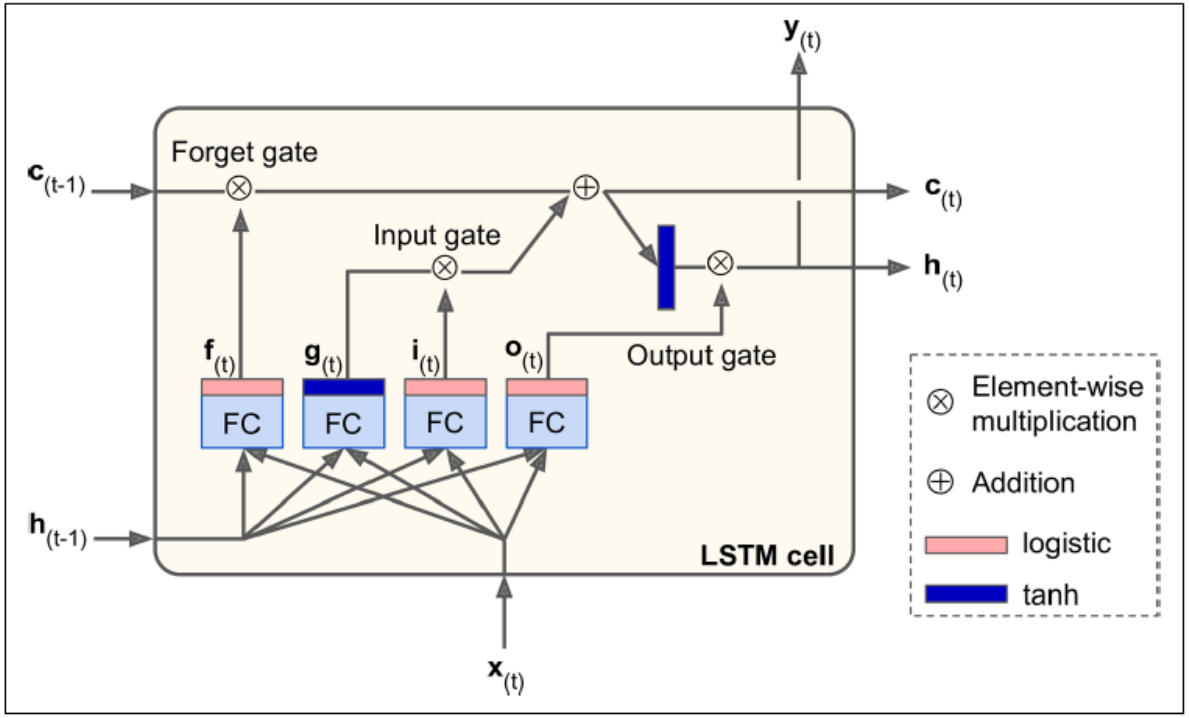
Problem zanikającego gradientu.

Wady RNN:

- długi czas trenowania;
- pamięć pierwszych wejść stopniowo zanika, a po pewnym czasie stan RNN nie zawiera praktycznie żadnych śladów pierwszych wejść

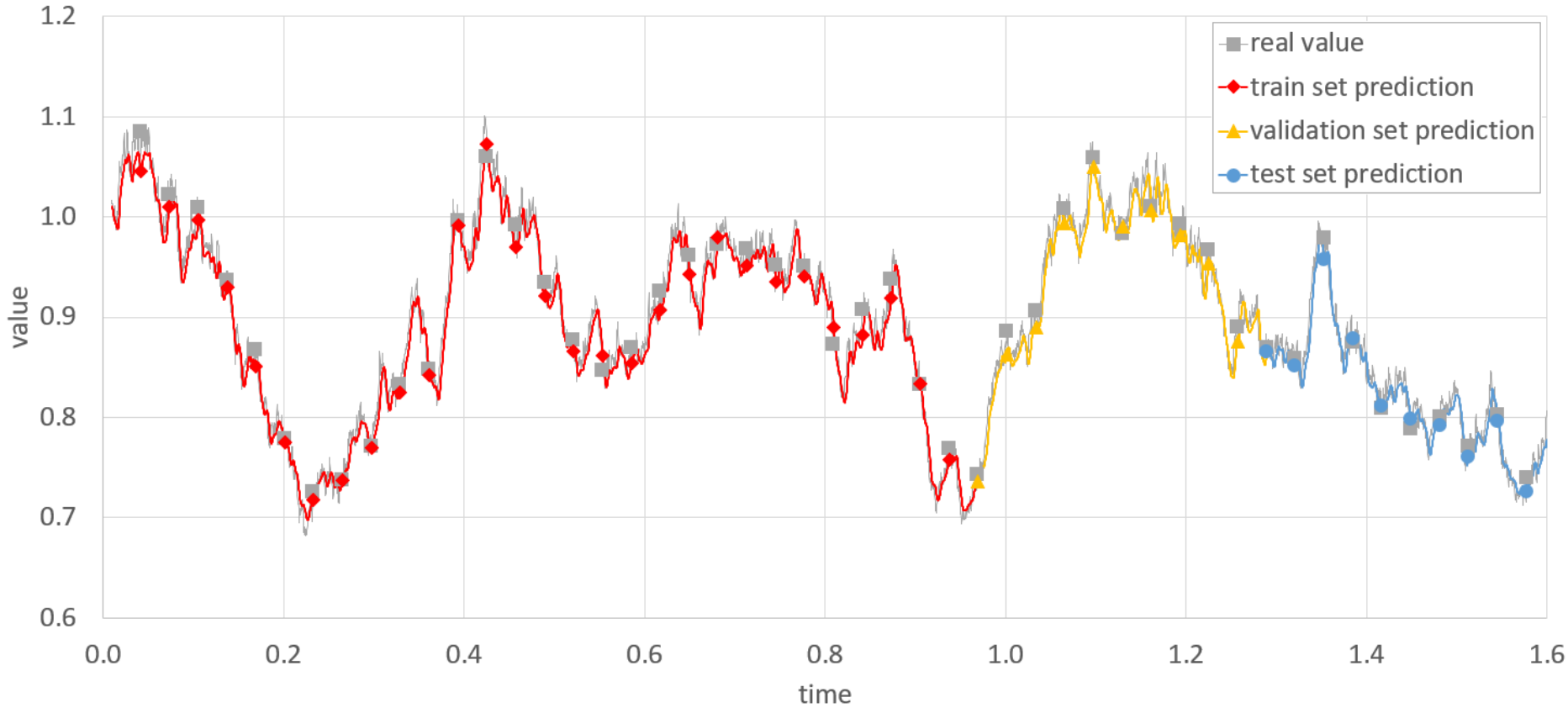
LSTM

Traktując komórkę LSTM jako ”czarną skrzynkę”, to wygląda ona dokładnie jak zwykła komórka, z tym że jej stan jest podzielony na dwa wektory:
$h_{(t)}$ i $c_{(t)}$ (”c” oznacza "komórkę", cell):
$h_{(t)}$ odpowiada stanowi krótkoterminowemu
$c_{(t)}$ odpowiada stanowi długoterminowemu



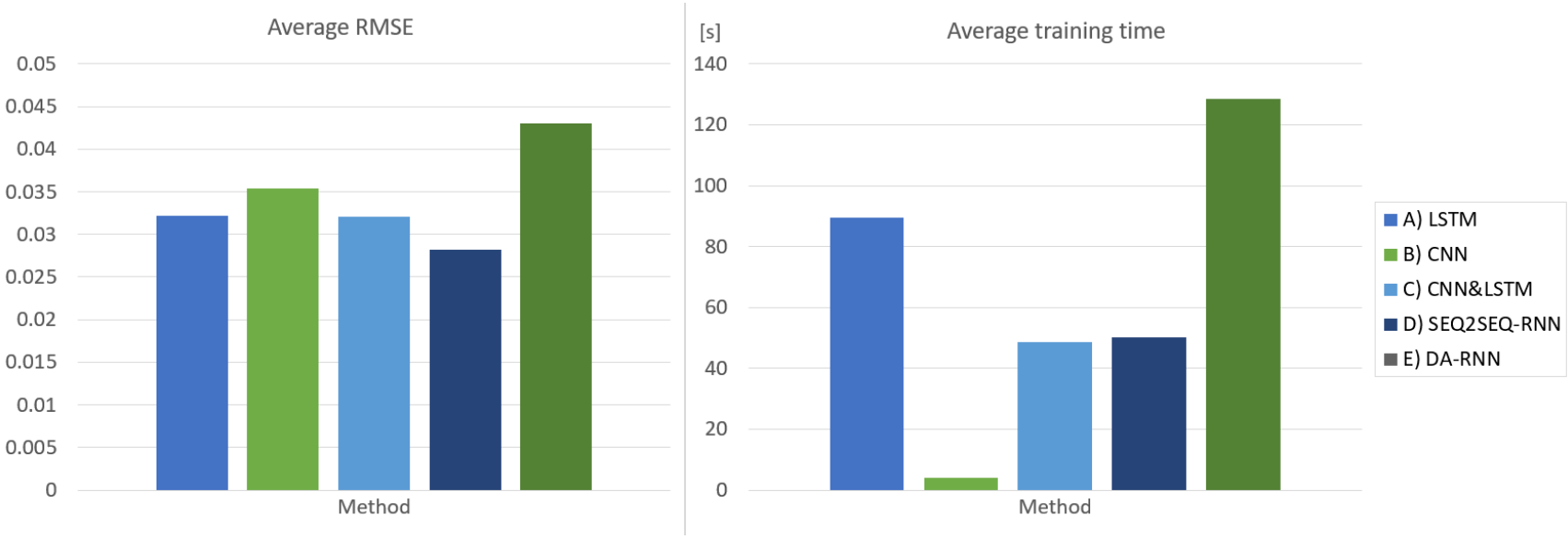
źródło: Géron, Aurélien. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O’Reilly Media, 2017.

Sieci neuronowe



Sieci neuronowe

A. LSTM (około 88 tys. parametrów).
B. CNN (około 1 tys. parametrów).
C. CNN&RNN (około 100 tys. parametrów).
D. Sequence-to-sequence RNN (około 200 tys. parametrów).
E. Dual-Stage Attention-Based RNN (około 200 tys. parametrów).



Bibliografia

[1] T. Bochacik, N. Czyżewska, A. Kałuża, D. Majchrowski, P. M. Morkisz, P. Przybyłowicz, M. Studzińska-Wrona, „Hybrid option pricing through AI and GPU-powered SDEs solvers.” poster, *GTC 2020, NVIDIA's GPU Technology conference* (2020)

[2] M. Jeanblanc, M. Yor, and M. Chesney. „Mathematical Methods for Financial Markets”. *Springer* (2009).

[3] A. Kałuża, P. M. Morkisz, and P. Przybyłowicz. „Optimal approximation of stochastic integrals in analytic noise model”. *Applied Mathematics and Computation 356* (2019), pp. 74-91.

[4] A. Kałuża and P. Przybyłowicz. „Optimal global approximation of jump-diffusion SDEs via path-independent step-size control”. *Applied Numerical Mathematics 128* (2018), pp. 24-42.

[5] P. M. Morkisz and P. Przybyłowicz. „Optimal pointwise approximation of SDE's from inexact information”. *Journal of Computational and Applied Mathematics 324* (2017), pp. 85-100.

[6] P. M. Morkisz and P. Przybyłowicz. „Randomized derivative-free Milstein algorithm for efficient approximation of solutions of SDEs under noisy information”. (2019). arXiv: https://arxiv.org/abs/1912.06865.

[7] P. Przybyłowicz. „Efficient approximate solution of jump-diffusion SDEs via path-dependent adaptive step-size control”. *Journal of Computational and Applied Mathematics 350* (2019), pp. 396-411.

Projekt wykonany w ramach grantu "Najlepsi z najlepszych! 4.0" oraz współpracy AGH – NVIDIA.

Sieci neuronowe

A. Two layers of LSTM (first with 128 output units, second with 32 output units) and a single dense layer with an output being the forecast entry (about 88k parameters).
B. CNN. The smallest analyzed architecture consisting of the convolution layer with 64 output filters followed by max pooling layer with the size of the pool window equal 2 (about 1k parameters).
C. Combination of CNN and RNN. The first layer is the same convolution as in the network B, then at the top there is one stacked layer of the LSTM with 128 hidden units (about 100k parameters).
D. Sequence-to-sequence RNN based on the encoder-decoder architecture. The encoder part of the network (one layer LSTM with 128 hidden units) is used to calculate an encoder vector, which in turn is used as an initial hidden state of the decoder part, which has the same structure as the encoder in this case (about 200k parameters).
E. Dual-Stage Attention-Based RNN. Roughly speaking, the encoder-decoder architecture with the attention mechanism for both parts, which once again consist of one layer LSTM with 128 hidden units (about 200k parameters).