Metody wyznaczania liczby e

Jakub Grobelny

Analiza Numeryczna (M) - Zadanie P1.4
18 listopada 2018

Streszczenie

W matematyce istnieje wiele liczb, które są w pewnym sensie interesujące, powtarzają się w różnych kontekstach i gałęziach tejże nauki. Jedną z takich liczb jest liczba e, zwana też liczbą Eulera lub stałą Napiera. Stała ta jest podstawą logarytmu naturalnego, czyli jedyną taką liczbą, dla której spełnione jest równanie $\ln e = 1$.

Ze względu na częstotliwość występowania liczby e w matematyce, istotne jest to, żeby móc ją wystarczająco dokładnie aproksymować. Niniejsze sprawozdanie jest podsumowaniem eksperymentu numerycznego polegającego na obliczaniu liczby e dwoma sposobami – korzystając z definicji $e = \lim_{n\to\infty}(1 + \frac{1}{n})^n$ oraz z rozwinięcia funkcji e^x w szereg $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$. Porównana została precyzja obu sposobów, oraz ich efektywność.

Spis treści

1 Definicja liczby e

Definicja 1: Logarytm naturalny

Dla $x \in \mathbb{R}, x > 0$ logarytm naturalny z x definiujemy następująco:

$$\ln x = \int_1^x \frac{1}{t} dt$$

Definicja 2: Podstawa logarytmu naturalnego

e jest jedyną taką liczbą rzeczywistą, że $\ln e = 1$.

Znając definicję 1 możemy sformułować definicję e, równoważną definicji 2.

Definicja 3: Podstawa logarytmu naturalnego

e jest jedyną taką liczbą rzeczywistą, że

$$1 = \int_1^e \frac{1}{t} dt$$

1.1 Obliczanie e jako granicy $\lim_{n\to\infty} (1 + \frac{1}{n})^n$

Twierdzenie 1

Liczbę e możemy zdefiniować następująco:

$$e = \lim_{n\to\infty} (1 + \frac{1}{n})^n.$$

Dowód. Niech $t \in [1; 1 + \frac{1}{n}]$. Wtedy

$$\frac{1}{1 + \frac{1}{n}} \leq \frac{1}{t} \leq 1.$$

Całkując stronami mamy

$$\int_1^{1+\frac{1}{n}} \frac{1}{1 + \frac{1}{n}}\, dt \leq \int_1^{1+\frac{1}{n}} \frac{1}{t}\, dt \leq \int_1^{1+\frac{1}{n}} 1\, dt.$$

Pierwsza całka jest równa $\frac{1}{n+1}$, druga z definicji 1 wynosi $\ln(1 + \frac{1}{n})$, trzecia zaś $\frac{1}{n}$. Mamy więc nierówność

$$\frac{1}{n+1} \leq \ln(1 + \frac{1}{n}) \leq \frac{1}{n},$$

którą możemy przekształcić dalej

$$e^{\frac{1}{n+1}} \leq 1 + \frac{1}{n} \leq e^{\frac{1}{n}}.$$

Podnosząc lewą stronę nierówności do potęgi $(n+1)$ otrzymujemy

$$e \leq (1+\frac{1}{n})^{n+1},$$

podnosząc zaś prawą stronę do potęgi n-tej mamy

$$(1+\frac{1}{n})^{n} \leq e.$$

Z tego wynika, że

$$(1+\frac{1}{n})^{n} \leq e \leq (1+\frac{1}{n})^{n+1}.$$

Dzieląc prawą stronę nierówności przez $1+\frac{1}{n}$ otrzymujemy

$$\frac{e}{1+\frac{1}{n}} \leq (1+\frac{1}{n})^{n}$$

co w połączeniu z wcześniejszą nierównością daje nam

$$\frac{e}{1+\frac{1}{n}} \leq (1+\frac{1}{n})^{n} \leq e.$$

Oczywistym jest, że $\lim\limits_{n\to\infty} \frac{e}{1+\frac{1}{n}} = e$ oraz $\lim\limits_{n\to\infty} e = e$, więc z twierdzenia o trzech ciągach wynika, że

$$\lim_{n\to\infty} (1+\frac{1}{n})^{n} = e.$$

□

Wiedząc, że $e = \lim\limits_{n\to\infty} (1+\frac{1}{n})^{n}$, wiemy, że dla dużych wartości n zachodzi

$$e \approx (1+\frac{1}{n})^{n}.$$

Wykorzystując tę obserwację można sformułować algorytm[1], który będzie wyznaczał przybliżoną wartość liczby e.

Program 1: Funkcja w języku Julia obliczająca $(1+\frac{1}{n})^{n}$

```
1 function calc_e_lim(n)
2     return (1 + 1/n)^n
3 end
```

[1]Wszystkie programy w niniejszym sprawozdaniu napisane są w języku Julia w wersji 1.0.1.

1.2 Rozwinięcie e w szereg

Definicja 4: Wielomian Taylora [1]

Niech funkcja f ma w punkcie x_0 pochodną k-tego rzędu, gdzie $k \in \mathbb{N}$. Wielomian

$$f(x_0) + \frac{f'(x_0)}{1!}(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + ... + \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

nazywamy wielomianem Taylora rzędu k funkcji f w punkcie x_0 i oznaczamy symbolem $P_k(x)$. Dla $x_0 = 0$ wielomian ten nazywamy wielomianem Maclaurina.

Wielomian P_k jest jedynym wielomianem stopnia k, który spełnia warunki:

$$P_k(x_0) = f(x_0),\ P_k'(x_0) = f'(x_0),\ ...,\ P_k^{(k)}(x_0) = f^{(k)}(x_0)$$

Twierdzenie 2: Wzór Taylora z resztą Lagrange'a [1]

Jeżeli funkcja f ma:
1. ciągłą pochodną rzędu $n-1$ na przedziale $[x_0; x]$,
2. pochodną właściwą $f^{(n)}$ na przedziale $(x_0; x)$,
to istnieje punkt $\xi \in (x_0; x)$ taki, że

$$f(x) = \sum_{k=0}^{n-1} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \frac{f^{(n)}(\xi)}{n!}(x - x_0)^n.$$

Powyższe twierdzenie jest prawdziwe także dla przedziału $[x; x_0]$, wtedy $\xi \in (x; x_0)$.
Równość występującą w tezie twierdzenia nazywamy wzorem Taylora, a wyrażenie

$$R_n(x) \stackrel{def}{=} \frac{f^{(n)}(\xi)}{n!}(x - x_0)^n$$

n-tą resztą Lagrange'a. Resztę Lagrange'a można także zapisać w postaci

$$R_n(x) \stackrel{def}{=} \frac{f^{(n)}(x_0 + \Theta\Delta x)}{n!}(\Delta x)^n,$$

gdzie $0 < \Theta < 1$ oraz $\Delta x = x - x_0$.
Dla $x_0 = 0$ wzór Taylora przyjmuje postać

$$f(x) = \sum_{k=0}^{n-1} \frac{f^{(k)}(0)}{k!}x^k + \frac{f^{(n)}(\xi)}{n!}x^n,$$

gdzie $\xi \in (0; x)$ dla $x > 0$ lub $\xi \in (x; 0)$ dla $x < 0$. Równość tę nazywamy wzorem Maclaurina.

Dla $n \to \infty$ mamy

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!}x^k$$

Dowód. Dowód w [1], na stronie 125. □

Znając powyższe twierdzenie można zapisać funkcję $f(x) = e^x$ w postaci szeregu przy użyciu wzoru Maclaurina. Najpierw można zaobserwować, że

$$\forall n \in \mathbb{N}\ (f^{(n)}(x) = e^x).$$

Powyższa obserwacja umożliwa rozwinięcie $f(x)$ w szereg

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(0)}{k!} x^k = \sum_{k=0}^{\infty} \frac{e^0}{k!} x^k = \sum_{k=0}^{\infty} \frac{x^k}{k!}.$$

$e = e^1 = f(1)$, więc ostatecznie można wyznaczyć następujący wzór na e podstawiając $x = 1$:

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}. \tag{1}$$

Program 2: Funkcja w języku Julia obliczająca sumę n wyrazów szeregu (1)

```
1  function calc_e_series(n)
2      fact = 1.0 # kolejne wartosci k!
3      sum = fact # suma szeregu
4      # pomijane jest obliczanie pierwszego wyrazu rownego 1/0!
5      for k in 1:n
6          fact *= k
7          sum += 1/fact
8      end
9      return sum
10 end
```

2 Opis eksperymentu

Obliczenia zostały wykonane w języku Julia (wersja 1.0.1) przy użyciu 256-bitowej arytmetyki (256 bitów przeznaczonych na reprezentację mantysy).

Programy 1 i 2 zostały uruchomione dla różnych wartości n, a następnie została porównana precyzja, z jaką przybliżają liczbę e poprzez porównanie wyniku do wbudowanej stałej `MathConstants.e`.

Obliczona została liczba cyfr znaczących każdego przybliżenia przy użyciu wzoru

$$\lfloor -log_{10}(\delta_e) \rfloor,$$

gdzie

$$\delta_e = |\frac{\Delta e}{e}|$$

i

$$\Delta e = e - \widetilde{e},$$

gdzie $\widetilde{e}$ oznacza otrzymane przybliżenie e.

2.1 Oszacowanie szybkości zbieżności obu programów

Przed wykonaniem obliczeń można spróbować oszacować ile iteracji każdej metody jest potrzebne, aby uzyskać maksymalnie dokładne przybliżenie e.

Definicja 5: Wykładnik zbieżności ciągu

Niech ciąg a_n będzie zbieżny do g. Jeżeli istnieją takie, liczby $p, C \in \mathbb{R},\ C > 0$, że,

$$\lim_{n\to\infty} \frac{|a_{n+1} - g|}{|a_n - g|^p} = C,$$

to p nazywamy **wykładnikiem zbieżności** ciągu a_n, a C – **stałą asymptotyczną błędu**. Dla $p = 1$ oraz $0 < C < 1$ zbieżność jest **liniowa**, dla $p = 2$, **kwadratowa**, dla $p = 3$ – **sześcienna**.
Dla $p = 1$ i $C = 1$ zbieżność jest **podliniowa**.
Jeżeli ciąg a_n zbiega do g podliniowo, oraz

$$\lim_{n\to\infty} \frac{|a_{n+2} - a_{n+1}|}{|a_{n+1} - a_n|} = 1,$$

to mówimy, że ciąg a_n zbiega **logarytmicznie** do g.

Można obliczyć wykładnik zbieżności ciągu $(1 + \frac{1}{n})^n$ korzystając z definicji 5. Okazuje się, że ten **ciąg jest zbieżny do** e **logarytmicznie**, więc kolejne przbliżenia polepszają się nieznacznie.

W przypadku drugiej metody oszacowanie potrzebnej wartości n, która zagwarantuje maksymalnie dużą precyzję wyniku, okazuje się być prostszym zadaniem.

Definicja 6: Precyzja arytmetyki

Precyzją arytmetyki danego komputera nazywamy liczbę

$$u = \frac{1}{2}\, 2^{-t},$$

gdzie $t \in \mathbb{N}$ jest liczbą bitów przeznaczonych na mantysę.

Można zauważyć, że korzystając ze wzoru (1) od pewnego momentu kolejne składniki sumy będą mniejsze niż precyzja arytmetyki. Oznacza to, że od tego momentu przybliżenie nie będzie się już zmieniać, gdyż $1 + u = 1$.

Przyjęto, że liczba bitów przeznaczonych na mantysę wynosi 256, mamy więc

$$u = \frac{1}{2}\, 2^{-256}$$

$$u = 2^{-257}$$

Dla ostatniego znaczącego składnika sumy szeregu ze wzoru (1) spełniona musi więc być nierów-

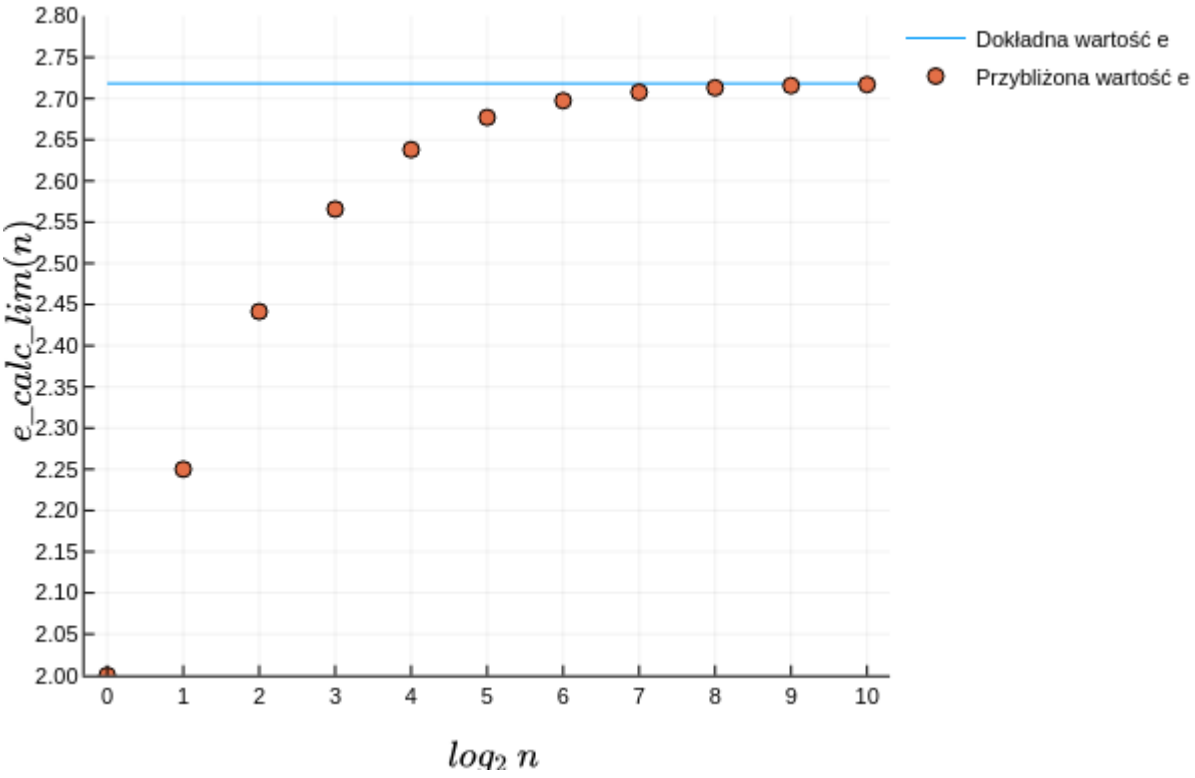
ność

$$\frac{1}{k!} \leq 2^{-257}.$$

$$k! \geq 2^{257}.$$

Rozwiązaniem jest $k \geq 57$. Można spodziewać się, że liczba iteracji większa niż k nie polepszy już otrzymanego przybliżenia e, jeżeli do jego obliczenia użyty zostanie program 2. Nie oznacza to jednak, że reszta szeregu (1) jest mniejsza niż używana precyzja arytmetyki. W zaproponowanym algorytmie wyrazy są jednak sumowane po kolei, w taki sposób, że każdy kolejny jest dodawany do dotychczasowej sumy, przez co można przypuszczać, że powyższa obserwacja będzie miała odzwierciedlenie w praktyce.

3 Wyniki obliczeń i ich analiza



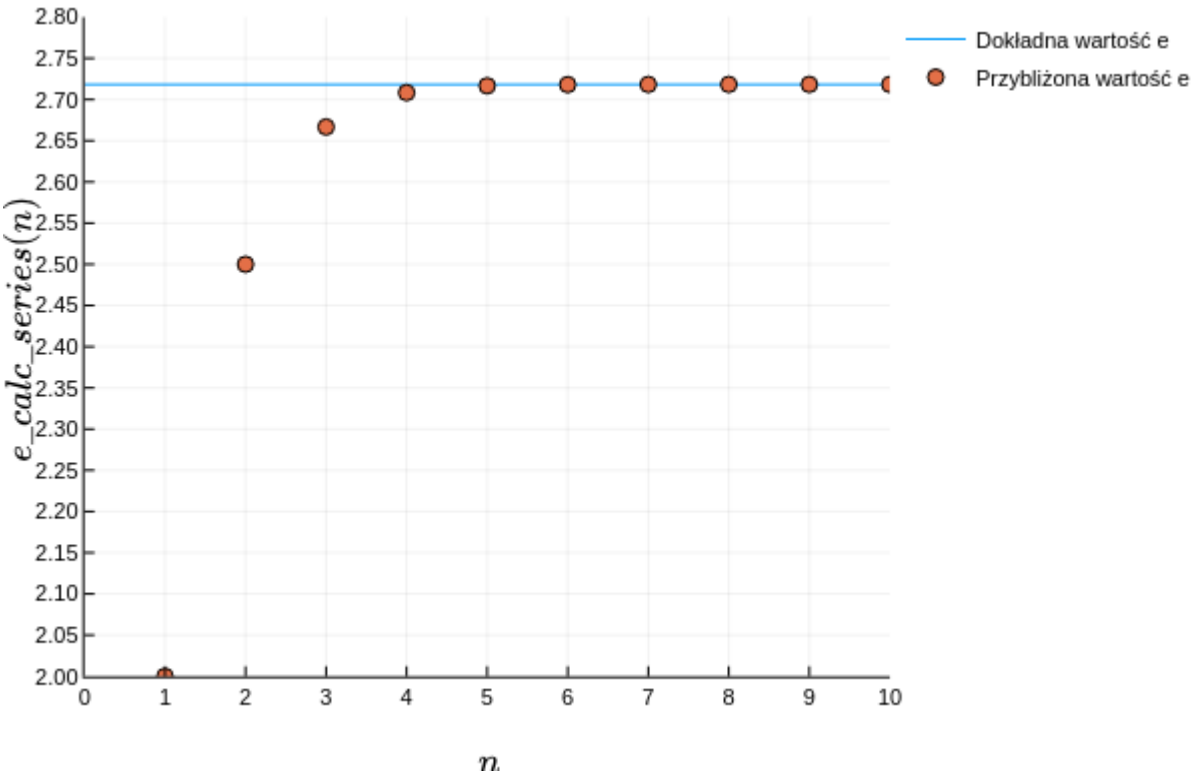
Rysunek 1: Wykres przedstawiający wyniki programu 1 dla $n = 2^0, 2^1, ..., 2^{10}$

n	Liczba cyfr znaczących przybliżenia e
2^1	0
2^2	0
2^3	1
2^4	1
2^5	1
2^6	2
2^7	2
2^8	2
2^9	3
2^{10}	3
$\vdots$	$\vdots$
2^{252}	76

Tablica 1: Wyniku programu 1 dla wybranych wartości n

n	Liczba cyfr znaczących przybliżenia e
1	0
2	1
3	1
4	2
5	3
6	4
7	4
8	5
9	6
10	7
11	9
21	21
31	35
41	51
51	58
55	75
56	76
57	76
58	76
61	76
71	76
81	76
91	76

Tablica 2: Wyniki programu 2 dla wybranych wartości n



Rysunek 2: Wykres przedstawiający wyniki programu 2 dla $n = 1, 2, ..., 10$

Jak widać w tablicy 1, otrzymane przybliżenie e przy użyciu programu 1 jest bardzo niedokładne nawet dla $n = 2^{10}$. Wyniki są zgodne z oczekiwaniami, gdyż ciąg $(1+\frac{1}{n})^n$ zbiega do e logarytmicznie. W tablicy 1 wyszczególniony został wynik dla $n = 2^{252}$, gdyż jest to pierwsza wartość n, dla której uzyskana została (w przybliżeniu) taka sama precyzja, jaką udało się uzyskać przy wykorzystaniu programu 2. Przy założeniu, że operacja potęgowania w Julii wykonuje się w czasie $O(log_2\, b)$, gdzie b jest to wykładnik potęgi, można oszacować złożoność programu 1 na $O(log_2\, n)$. Widać wtedy dysproporcje pomiędzy dwoma algorytmami. Pierwszy z nich potrzebuje 252 iteracje aby osiągnąć wynik, który drugi zwraca już przy 56 iteracjach (program 2 oczywiście ma złożoność $O(n)$).

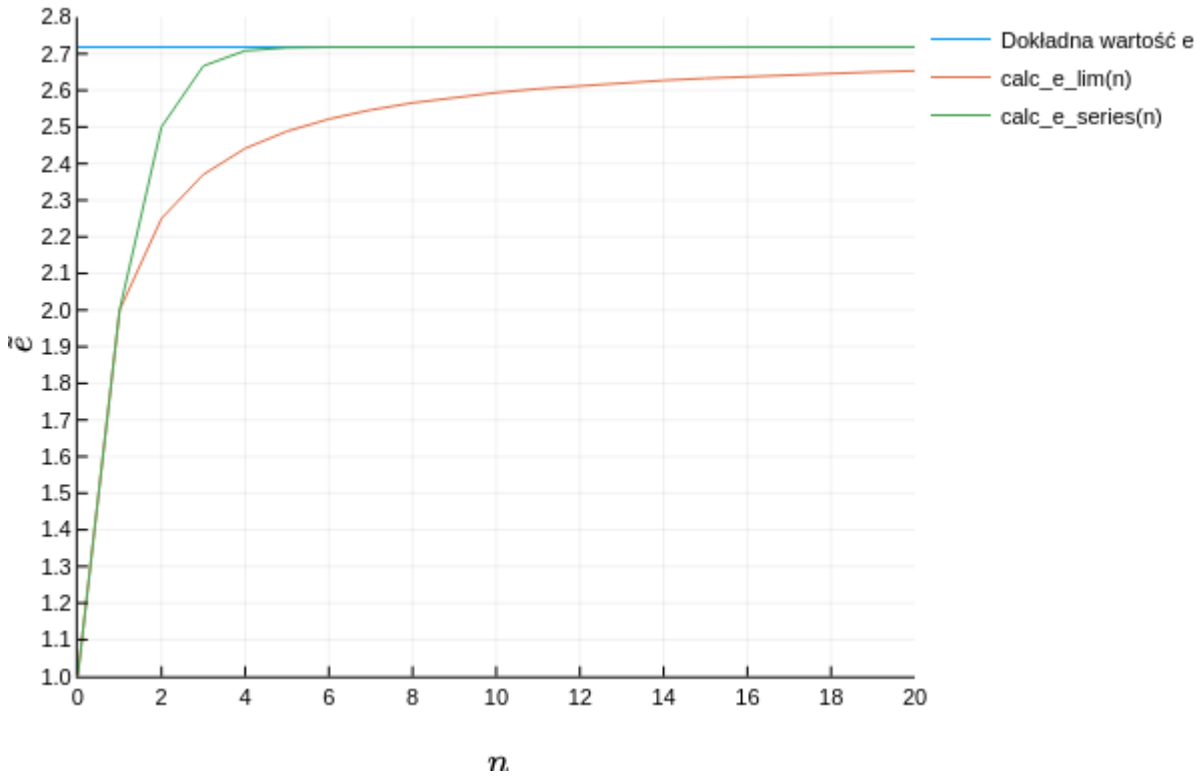
Tablica 2 potwierdza również wcześniejsze przypuszczenia, że liczba iteracji większa bądź równa 57 nie polepszy już dokładności wyniku przy zastosowaniu 256-bitowej precyzji.

Uzyskane wyniki pokazują, że obliczanie e z definicji

$$e = \lim_{n\to\infty} (1 + \frac{1}{n})^n$$

jest gorszym sposobem numerycznego obliczania tej stałej niż przy wykorzystaniu wzoru

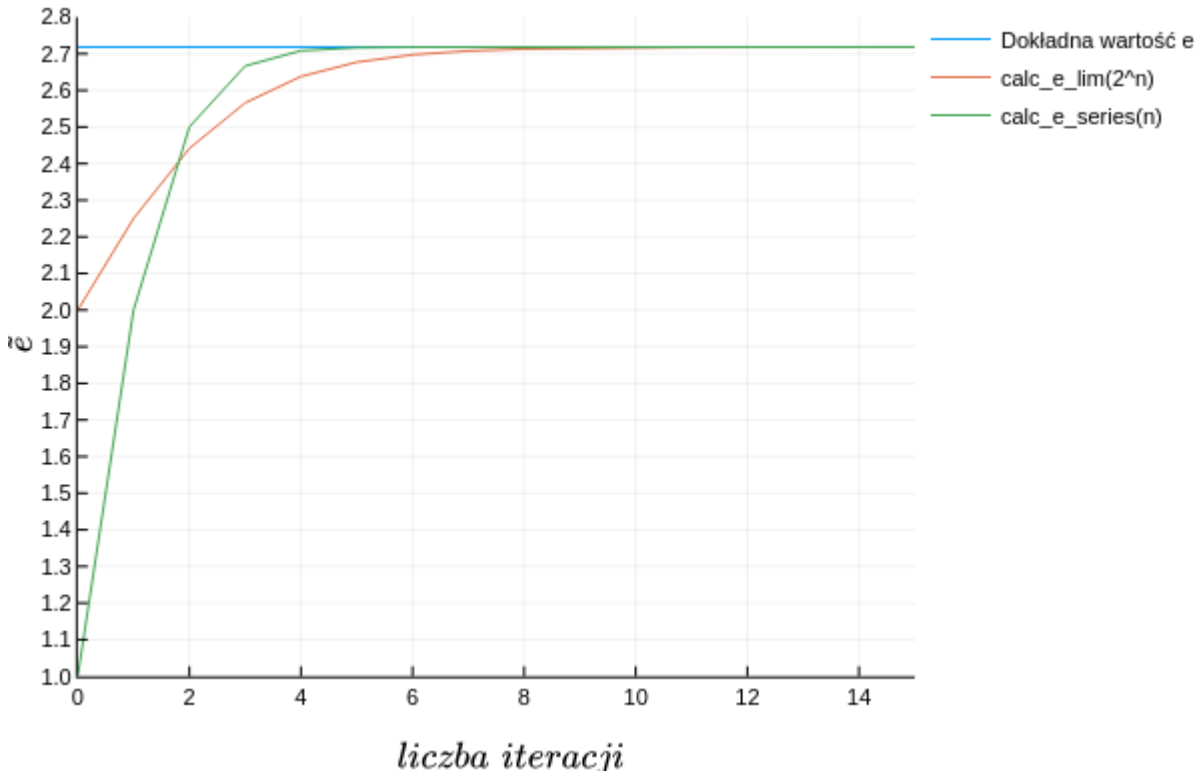
$$e = \sum_{k=0}^{\infty} \frac{1}{k!}.$$



Rysunek 3: Wykres porównujący wyniki obu programów w jednakowej skali

Dane przedstawione na rysunkach 1 i 2 mogą być nieco mylące, ze względu na różną skalę osi x, więc zostały one przedstawione również na rysunku 3, który pozwala na lepszą analizę otrzymanych wyników. Widać, że wykres dla funkcji `calc_e_lim` jest o wiele bardziej spłaszczony, niż ten dla funkcji `calc_e_series`.

Zakładając, że `calc_e_lim` ma złożoność $O(log_2\, n)$ można przedstawić wyniki tak jak na rysunku 4, gdzie na osi x znajduje się przybliżona liczba iteracji algorytmu (w przypadku programu 1 iteracje rozumiemy jako liczbę kroków potrzebnych do podniesienia wyrażenia $(1 + \frac{1}{n})$ do potęgi n).



Rysunek 4: Wykres porównujący wyniki obu programów dla danej liczby iteracji

Widać, że nawet przy takim spojrzeniu na otrzymane rezultaty, funkcja `calc_e_series` dalej jest zauważalnie lepsza niż funkcja `calc_e_lim`.

Literatura

[1] Zbigniew Skoczylas Marian Gewert. *Analiza matematyczna 1, Definicje, twierdzenia, wzory*. Oficyna Wydawnicza GiS, 1991.