Powtórka - liczby w pamięci komputera

Liczby całkowite

Liczby naturalne

Liczby zapisujemy w systemie pozycyjnym – zazwyczaj jako potęgi liczby 10

```
3 * 10^2 + 9 * 10^1 + 2 * 10^0
```

W informatyce – stosuje się system dwójkowy

In[∘]:=
```
1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0
```

- Zakres: 0 do $2^{\text{liczba bitów}}-1$

In[∘]:=
```
BaseForm[153, 2]
```
zapis w system liczbowym

Out[∘]//BaseForm=

10011001_2

Liczby całkowite

Liczby ujemne

kod uzupełnień do dwóch:

In[ ]:=
```
0*(-2^6) + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0
```

Out[ ]=

42

przepis na “–42”: zamień wszystkie bity na przeci-wne i dodaj 1

In[ ]:=
```
1*(-2^6) + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 1
```

Out[ ]=

-42

- jednoznaczne zero

Liczby zmiennoprzecinkowe

- W systemie dziesiętnym 454.574=

In[ ]:=
```
N[4 * 10^2 + 5 * 10^1 + 4 * 10^0 + 5 * 10^-1 + 7 * 10^-2 + 4 * 10^-3]
```

Out[ ]=
454.574

Ale też 1/3

$= 0 * 10^0 + 3 * 10^{-1} + 3 * 10^{-2} + 3 * 10^{-3} + ...$

- W systemie binarnym 12.625 =

In[ ]:=
```
N[1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0 + 1 * 2^-1 + 0 * 2^-2 + 1 * 2^-3 ]
```

In[ ]:=
```
BaseForm[13.625, 2]
```
zapis w system liczbowym

Out[ ]//BaseForm=
1101.101_2

Ale 0.1 =

In[ ]:=
```
BaseForm[0.1, 2]
```
zapis w system liczbowym

Out[ ]//BaseForm=
$0.0001100110011001100110011001101_2$

In[ ]:=
```
N[0 * 2^-1 + 0 * 2^-2 + 0 * 2^-3 + 1 * 2^-4 + 1 * 2^-5 + 0 * 2^-6 + 0 * 2^-7 +
  1 * 2^-8 + 1 * 2^-9 + 0 * 2^-10 + 0 * 2^-11 + 1 * 2^-12 + 1 * 2^-13 + 0 * 2^-14 + 0 * 2^-15 +
  1 * 2^-16 + 1 * 2^-17 + 0 * 2^-18 + 0 * 2^-19 + 1 * 2^-20 + 1 * 2^-21 + 0 * 2^-22 + 1 * 2^-23, 10]
```
przybliżenie numeryczne

Liczby zmiennoprzecinkowe: IEEE 754

Reprezentacja liczb: ±mantysa * 2^{cecha}

In[◦]:=
```
BaseForm[145.1135, 2]
zapis w system liczbowym
```

Out[◦]//BaseForm=

$1.0010001000111010001_2 \times 2^7$

In[◦]:=
```
rand = RandomChoice[{0, 1}, 32];
       losowy wybór
Graphics[Table[{EdgeForm[Black], If[i == 0, FaceForm[Red],
grafika  tabela  rodzaj kr···  czarny  operator ···  styl na z···  czerwony
    If[i > 0 && i < 9, FaceForm[Green], FaceForm[Yellow]]], Rectangle[{i, 0}, {i + 1, 1}],
    operator warunkowy  styl na z···  zielony  styl na z···  żółty  prostokąt
   Black, Text[ToString[rand[[i + 1]]], {i + 0.5, 0.5}]}, {i, 0, 31}], ImageSize → Full]
   czarny  tekst  przemień na ciąg znaków                         rozmiar obrazu  kompletny
```

Out[◦]=

0	0	1	1	1	0	1	1	0	0	0	0	1	0	0	1	1	1	0	0	0	1	0	1	1	0	0	1	0	0	0	1

In[◦]:=
```
 If[rand[[1]]==1,sign =-1,sign =1]
wykladnik = Total[Table[rand[[2+i]]*2^(7-i),{i,0,7}]]-127
mantysa = 1+N@Total[Table[rand[[10+i]]*2^(-i-1),{i,0,22}]]
N[sign*mantysa * 2^wykladnik]
```

Out[◦]=

1

Out[◦]=

-9

Out[◦]=

1.07634

Out[◦]=

0.00210223

- Przypadki specjalne
  - 0^+

Out[◦]=

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

  - 0^-

Out[◦]=

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- ±∞ (mantysa = same zera)

Out[•]=

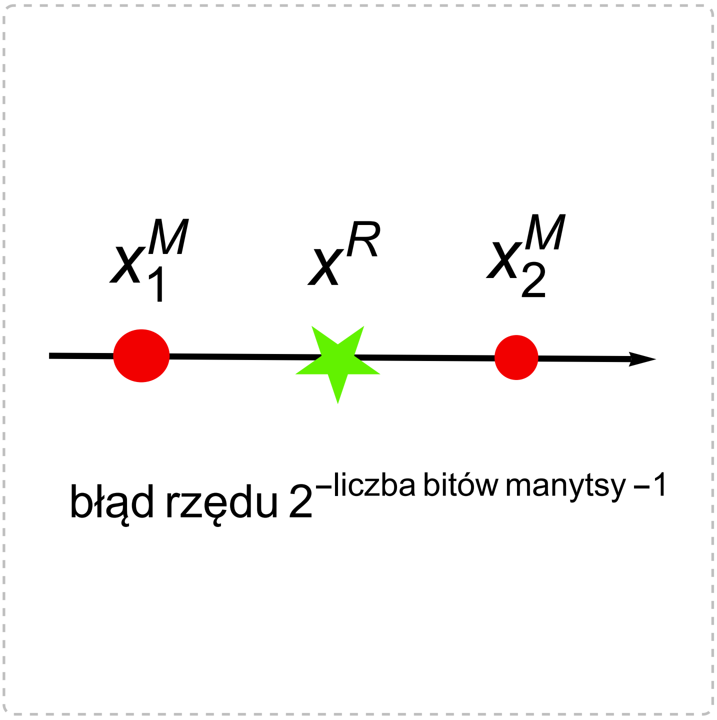
±1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- NaN (mantysa może być niezerowa)

Out[•]=

±1	1	1	1	1	1	1	1	1	1	0	1	1	0	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	1	1

Liczba maszynowa vs liczba rzeczywista



In[∘]:=

```
N[2^-24]
```

przybliżenie numeryczne

5.96046×10^{-8}

Błąd względny i bezwzględny

Błąd bezwzględny: $| x - x_p |$ (x_p: przybliżenie liczby x)

Błąd względny: $\frac{|x-x_p|}{x}$

Przykład: wzór Stirlinga: n! ≈ $\left(\frac{n}{e}\right)^n \sqrt{2 \pi n}$

In[ ]:=
```
stir[n_] := Sqrt[2 π n] (n/E)^n
```

In[ ]:=
```
m = 100
N[m!]
N[stir[m]]
Abs[N[m!] - N[stir[m]]] / N[m!] * 100  (*błąd względny*)
Abs[N[m!] - N[stir[m]]]  (*błąd bezwzględny*)
```

przybliżenie numeryczne

Out[ ]= 100

Out[ ]= 9.33262×10^{157}

Out[ ]= 9.32485×10^{157}

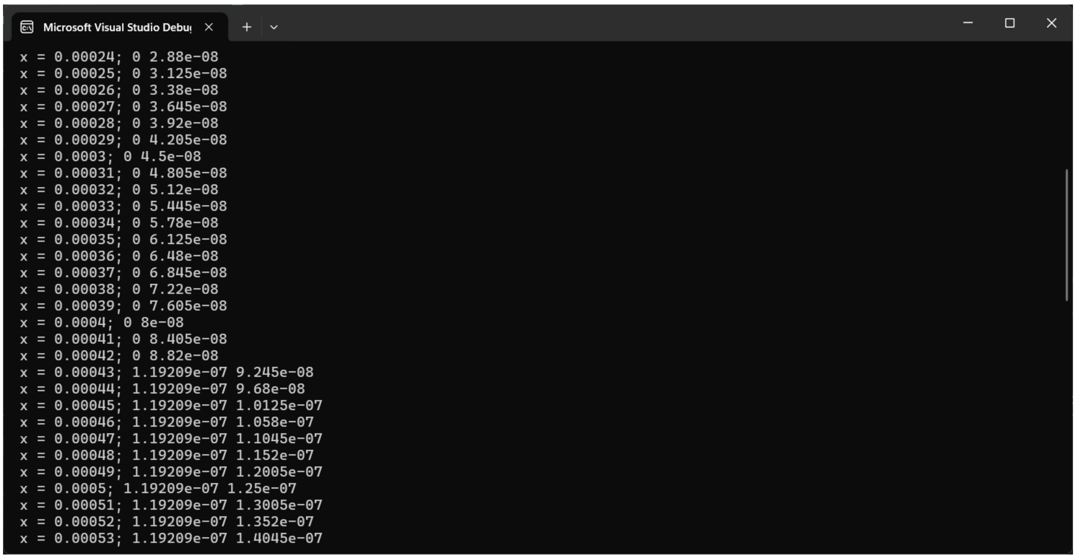
Out[ ]= 0.0832983

Out[ ]= 7.77392×10^{154}

Utrata liczb znaczących

Należy być też ostrożnym przy odejmowaniu bliskich sobie wartości.

$$\sqrt{x^2+1}-1 \text{ vs } \frac{x^2}{\sqrt{x^2+1}+1}$$



```
Microsoft Visual Studio Debu
x = 0.00024; 0 2.88e-08
x = 0.00025; 0 3.125e-08
x = 0.00026; 0 3.38e-08
x = 0.00027; 0 3.645e-08
x = 0.00028; 0 3.92e-08
x = 0.00029; 0 4.205e-08
x = 0.0003; 0 4.5e-08
x = 0.00031; 0 4.805e-08
x = 0.00032; 0 5.12e-08
x = 0.00033; 0 5.445e-08
x = 0.00034; 0 5.78e-08
x = 0.00035; 0 6.125e-08
x = 0.00036; 0 6.48e-08
x = 0.00037; 0 6.845e-08
x = 0.00038; 0 7.22e-08
x = 0.00039; 0 7.605e-08
x = 0.0004; 0 8e-08
x = 0.00041; 0 8.405e-08
x = 0.00042; 0 8.82e-08
x = 0.00043; 1.19209e-07 9.245e-08
x = 0.00044; 1.19209e-07 9.68e-08
x = 0.00045; 1.19209e-07 1.0125e-07
x = 0.00046; 1.19209e-07 1.058e-07
x = 0.00047; 1.19209e-07 1.1045e-07
x = 0.00048; 1.19209e-07 1.152e-07
x = 0.00049; 1.19209e-07 1.2005e-07
x = 0.0005; 1.19209e-07 1.25e-07
x = 0.00051; 1.19209e-07 1.3005e-07
x = 0.00052; 1.19209e-07 1.352e-07
x = 0.00053; 1.19209e-07 1.4045e-07
```

(*Mathematica jest na tyle inteligentna, że sobie poradzi z takim problemem*)

Kod źródłowy w C++

```
#include < iostream >
#include < cmath >

float f1 (float x) {
    return std::sqrtf (x*x + 1) - 1;
    }

float f2 (float x) {
    return x*x/(std::sqrtf (x*x + 1) + 1);
    }

int main () {
    float val {}, step = 0.00001;
    for (int i = 0; i < 100; i++) {
        val = step*i;
        std::cout << " x = " << val << "; " << f1 (val) << " " << f2 (val) << std::endl;
        }
    return 0;
    }
```

Możemy też skorzystać z szeregu Taylora:

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

Obliczyć wartość wyrażenia Log[x+1]-x dla x bliskich zera .

In[◦]:=
```
Normal[Series[Log[x+1]-x,{x,0,5}]]
```

Out[◦]=

$$-\frac{x^2}{2}+\frac{x^3}{3}-\frac{x^4}{4}+\frac{x^5}{5}$$

Microsoft Visual Studio Debu

```
x = 0; 0 0
x = 1e-05; 1.35306e-08 -4.99997e-11
x = 2e-05; 2.69611e-08 -1.99997e-10
x = 3e-05; 4.02906e-08 -4.49991e-10
x = 4e-05; 5.35219e-08 -7.99979e-10
x = 5e-05; -5.25542e-08 -1.24996e-09
x = 6e-05; -3.9523e-08 -1.79993e-09
x = 7e-05; -2.65863e-08 -2.44989e-09
x = 8e-05; -1.37661e-08 -3.19983e-09
x = 9e-05; -1.04046e-09 -4.04976e-09
x = 0.0001; 1.15979e-08 -4.99967e-09
x = 0.00011; 2.41271e-08 -6.04956e-09
x = 0.00012; 3.65544e-08 -7.19942e-09
x = 0.00013; 4.88799e-08 -8.44927e-09
x = 0.00014; -5.80767e-08 -9.79908e-09
x = 0.00015; -4.59549e-08 -1.12489e-08
x = 0.00016; -3.39205e-08 -1.27986e-08
x = 0.00017; -2.20025e-08 -1.44484e-08
x = 0.00018; -1.01718e-08 -1.61981e-08
x = 0.00019; 1.57161e-09 -1.80477e-08
x = 0.0002; 1.3184e-08 -1.99973e-08
x = 0.00021; 2.47092e-08 -2.20469e-08
x = 0.00022; -8.30332e-08 -2.41964e-08
x = 0.00023; -7.17118e-08 -2.64459e-08
x = 0.00024; -6.04778e-08 -2.87954e-08
x = 0.00025; -4.9331e-08 -3.12448e-08
x = 0.00026; -3.83297e-08 -3.37941e-08
x = 0.00027; -2.73867e-08 -3.64434e-08
x = 0.00028; -1.65601e-08 -3.91927e-08
x = 0.00029; -5.84987e-09 -4.20419e-08
```

In[◦]:=
```
Table[{N@x, N[Log[x + 1] - x]}, {x, 10^-5, 0.00029, 10^-5}]
```
tabela przy·· ·· logarytm

Out[◦]=

$\{\{0.00001, -4.99996\times10^{-11}\}, \{0.00002, -1.99997\times10^{-10}\},$
$\{0.00003, -4.49991\times10^{-10}\}, \{0.00004, -7.99979\times10^{-10}\}, \{0.00005, -1.24996\times10^{-9}\},$
$\{0.00006, -1.79993\times10^{-9}\}, \{0.00007, -2.44989\times10^{-9}\}, \{0.00008, -3.19983\times10^{-9}\},$
$\{0.00009, -4.04976\times10^{-9}\}, \{0.0001, -4.99967\times10^{-9}\}, \{0.00011, -6.04956\times10^{-9}\},$
$\{0.00012, -7.19942\times10^{-9}\}, \{0.00013, -8.44927\times10^{-9}\}, \{0.00014, -9.79909\times10^{-9}\},$
$\{0.00015, -1.12489\times10^{-8}\}, \{0.00016, -1.27986\times10^{-8}\}, \{0.00017, -1.44484\times10^{-8}\},$
$\{0.00018, -1.61981\times10^{-8}\}, \{0.00019, -1.80477\times10^{-8}\}, \{0.0002, -1.99973\times10^{-8}\},$
$\{0.00021, -2.20469\times10^{-8}\}, \{0.00022, -2.41965\times10^{-8}\}, \{0.00023, -2.64459\times10^{-8}\},$
$\{0.00024, -2.87954\times10^{-8}\}, \{0.00025, -3.12448\times10^{-8}\}, \{0.00026, -3.37941\times10^{-8}\},$
$\{0.00027, -3.64434\times10^{-8}\}, \{0.00028, -3.91927\times10^{-8}\}, \{0.00029, -4.20419\times10^{-8}\}\}$

```
(*Mathematica znowu sobie radzi*)
```

Oblicz wartość funkcji sinus dla argumentu typu float (4 bajty) x = 24684.1516

Mathematica daje wynik:

In[ ]:=
```
N[Sin[24 684.1516]]
```
sinus

Out[ ]=
```
-0.611631
```

W C++ funkcja: std::sin(24684.1516f) daje −0.612219

- Rozwiązania problemu:
  - użyć zmiennej o większej dokładności (double'a): std::sin(24684.1516) = −0.611631
  - Wykorzystać periodyczność funkcji sinus: $24\,684.1516 = 3928 * 2\,\pi + 3.7997$ =>std::sin(3.7997f) = −0.611621

In[ ]:=
```
Mod[24 684.1516, 2 π]
```
modulo

Out[ ]=
```
3.79971
```

Znaj swoje szeregi

Spróbujmy obliczyć liczbę π z dokładnością do piątego miejsca po przecinku

Sposób naiwny: $\frac{\pi^2}{6} = \sum_{k=1}^{\infty} \frac{1}{k^2}$ więc $\pi = \sqrt{6 \sum_{k=1}^{\infty} \frac{1}{k^2}}$

In[∘]:=
```
AbsoluteTiming@N[Sqrt[6 * Sum[1/i^2, {i, 1, 100000}]], 10]
N[π, 10]
Abs[N[π, 10] - N[Sqrt[6 * Sum[1/i^2, {i, 1, 100000}]], 10]]
```
bezwzględna długo··· przybliżenie numeryczne
przybliżenie numeryczne
wa·· przybliżeni··· przybliżenie numeryczne

Out[∘]= {3.92185, 3.141583104}

Out[∘]= 3.141592654

Out[∘]= 9.549×10^{-6}

Sposób lepszy: $\frac{1}{\pi} = \frac{2\sqrt{2}}{99^2} \sum_{k=0}^{\infty} \frac{(4k)!}{(k!)^4} \frac{26390\,k + 1103}{396^{4k}}$

In[∘]:=
```
N[π, 40]
AbsoluteTiming@N[1 / Sum[(2 Sqrt[2])/99^2 (4 k)!/(k!)^4 * (26390 k + 1103)/396^(4 k), {k, 0, 10}], 40]
Abs[N[π, 40] - N[1 / Sum[(2 Sqrt[2])/99^2 (4 k)!/(k!)^4 * (26390 k + 1103)/396^(4 k), {k, 0, 3}], 40]]
```
przybliżenie numeryczne
bezwzględna długo··· przyb·· sumowanie
wa·· przybliżeni··· przyb·· sumowanie

Out[∘]= 3.141592653589793238462643383279502884197

Out[∘]= {0.0007701, 3.141592653589793238462643383279502884197}

Out[∘]= $5.2388963 \times 10^{-32}$

Inny szereg: $\pi = \sum_{k=0}^{\infty} \frac{(-1)^k}{4^k} \left(\frac{2}{4k+1} + \frac{2}{4k+2} + \frac{1}{4k+3} \right)$

In[ ]:=

```
AbsoluteTiming[N[Sum[(-1)^k/4^k (2/(4 k + 1) + 2/(4 k + 2) + 1/(4 k + 3)), {k, 0, 10}], 10]]
```

bezwzględna dług··· ··· sumowanie

```
Abs[N[π, 10] - N[Sum[(-1)^k/4^k (2/(4 k + 1) + 2/(4 k + 2) + 1/(4 k + 3)), {k, 0, 10}], 10]]
```

wa·· przybliżeni··· ·· sumowanie

Out[ ]=

{0.0001779, 3.141592675}

Out[ ]=

2.1×10^{-8}

2.1×10^{-8}

Ciekawostka - mnożenie macierzy

Mnożenie jest “droższe” od dodawania

In[◦]:=
```
A = (a1 a2
     a3 a4)
B = (b1 b2
     b3 b4)
A.B
```

Out[◦]=
```
{{a1, a2}, {a3, a4}}
```

Out[◦]=
```
{{b1, b2}, {b3, b4}}
```

Out[◦]=
```
{{a1 b1 + a2 b3, a1 b2 + a2 b4}, {a3 b1 + a4 b3, a3 b2 + a4 b4}}
```

Algorytm Strassena

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}); \\
M_2 &= (A_{21} + A_{22}) \times B_{11}; \\
M_3 &= A_{11} \times (B_{12} - B_{22}); \\
M_4 &= A_{22} \times (B_{21} - B_{11}); \\
M_5 &= (A_{11} + A_{12}) \times B_{22}; \\
M_6 &= (A_{21} - A_{11}) \times (B_{11} + B_{12}); \\
M_7 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}),
\end{aligned}
$$

$$
\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 - M_2 + M_3 + M_6 \end{bmatrix}
$$

Dla macierzy 4x4 algorytm Strassena potrzebuje 49 mnożeń.

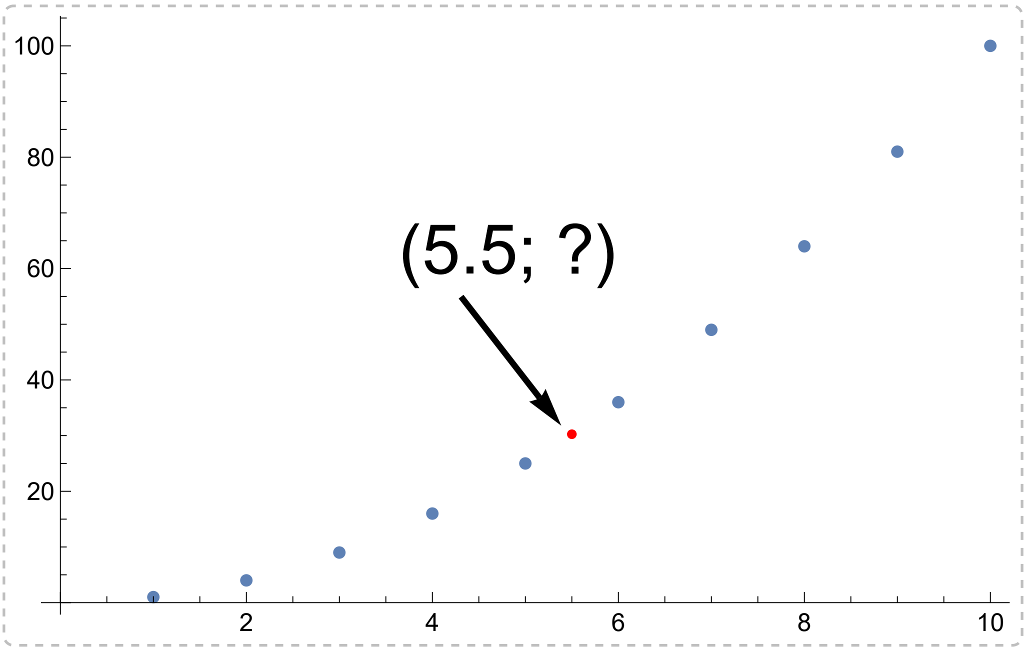
AlphaTensor znalazł(?a) sposób z 47 mnożeniami

Interpolacja i ekstrapolacja

Interpolacja - znajdowanie nowych wartości dla znanego zbioru danych

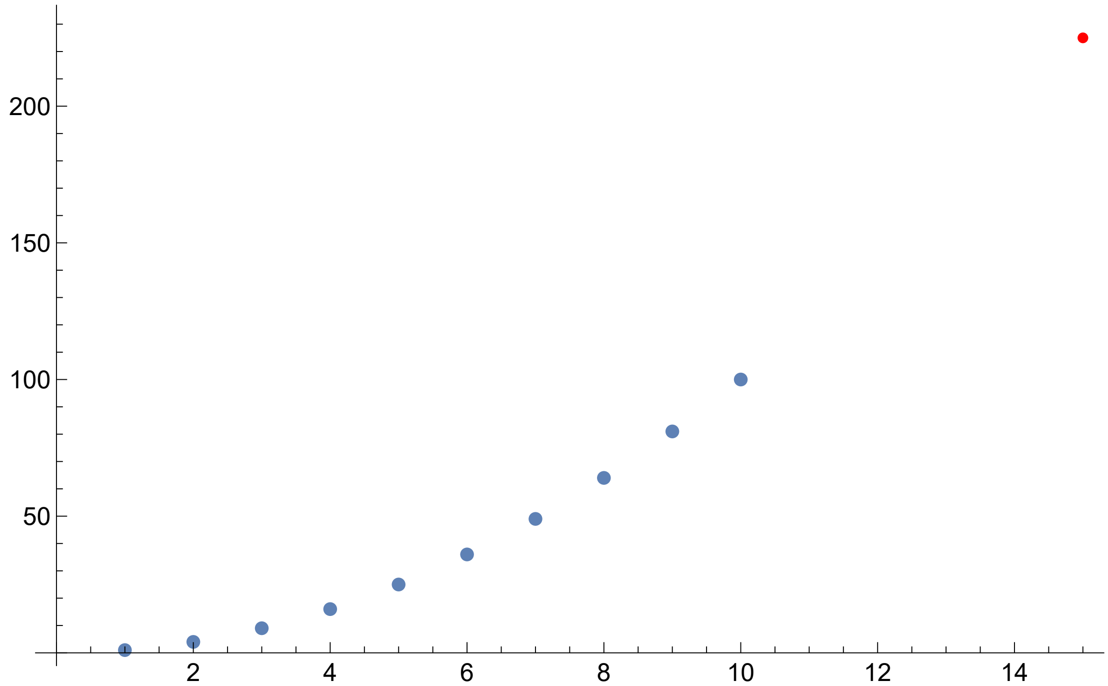
In[∘]:=
```
Show[{ListPlot[Table[i^2, {i, 1, 10}]],
  Graphics[{PointSize[0.01], Red, Point[{5.5, 5.5^2}]}]},
 LabelStyle → {13, GrayLevel[0]}]
```

⌊pokaż ⌊wykres d⋯ ⌊tabela ⌊grafika ⌊rozmiar kropki ⌊cz⋯ ⌊punkt ⌊styl etykiety ⌊poziom szarości



Ekstrapolacja - znajdowanie nowych wartości dla znanego zbioru danych, ale poza zakresem danych

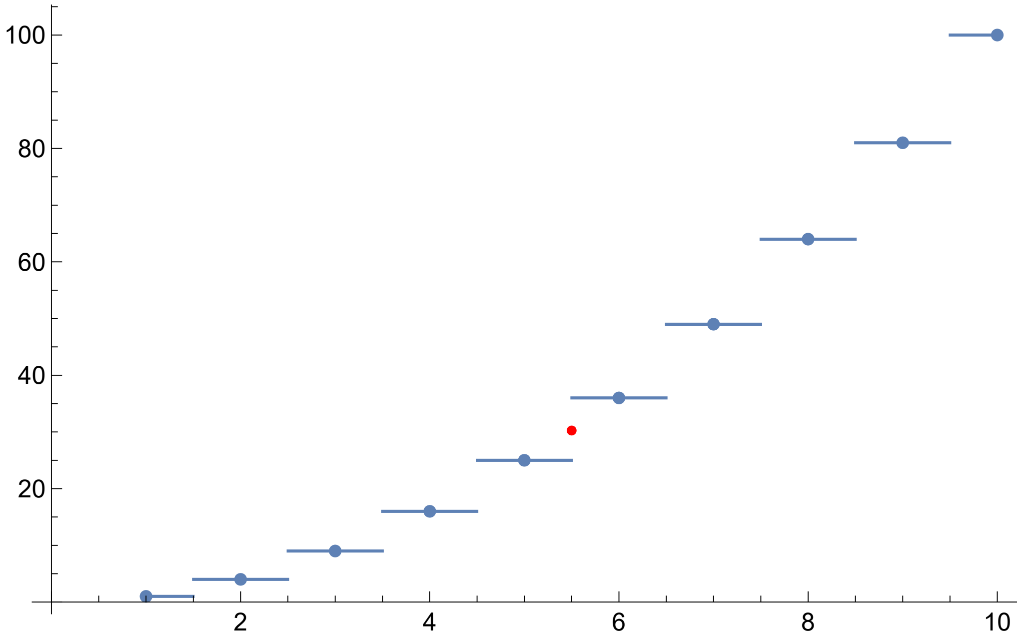
Out[ ]=



Interpolacja stała - wartość pomiędzy punktami jest taka sama jak najbliższego elementu

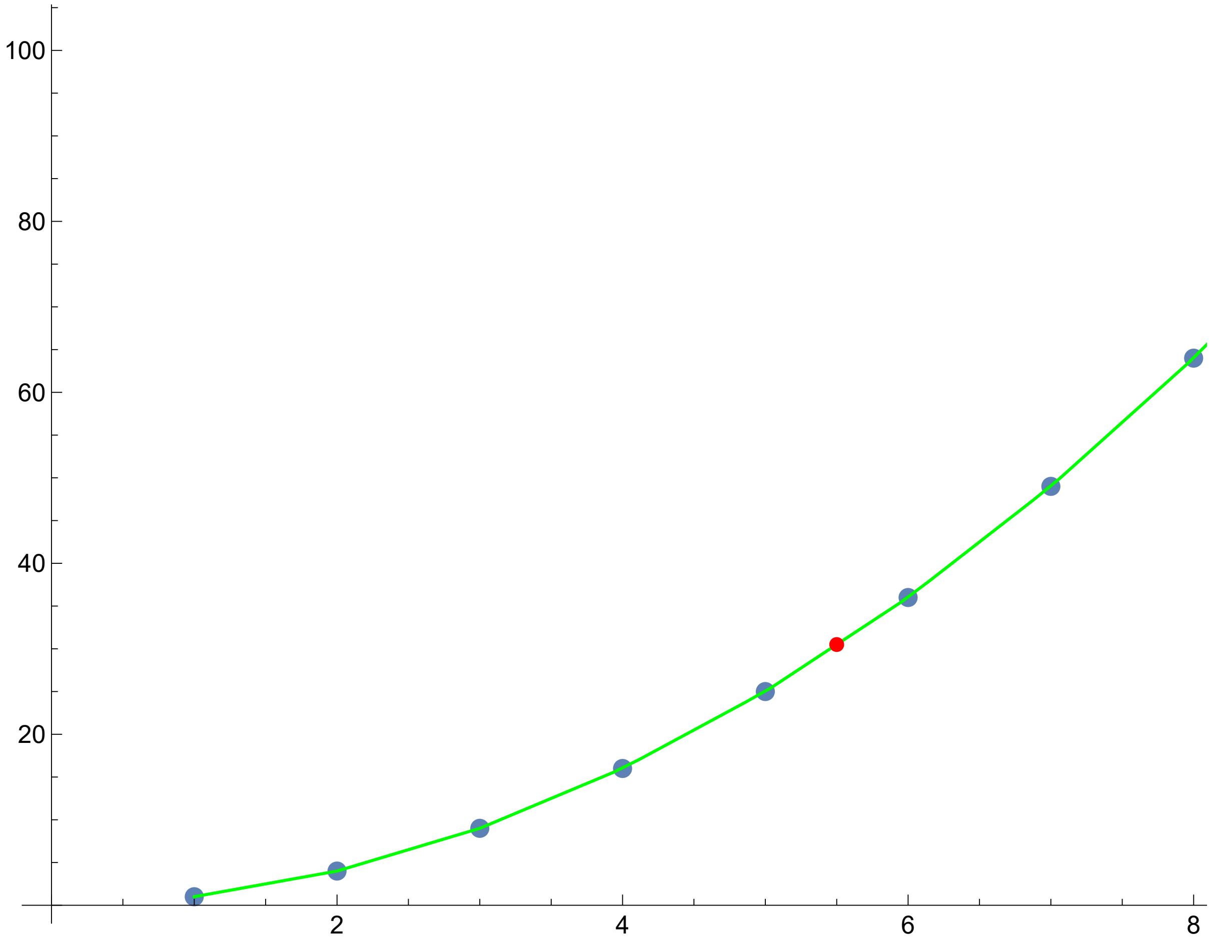
In[◦]:=
```
Show[{ListPlot[Table[i^2, {i, 1, 10}]], Plot[Ceiling[x - .5]^2, {x, 1, 10}],
  Graphics[{PointSize[0.01], Red, Point[{5.5, 5.5^2}]}]},
 LabelStyle → {13, GrayLevel[0]}]
```
pokaż, wykres d⋯, tabela, wykres, grafika, rozmiar kropki, cz⋯, punkt, styl etykiety, poziom szarości

Out[◦]=



Interpolacja liniowa - przybliżamy wartość funkcji między dwoma punktami funkcją liniową

Out[∘]=



Znajdźmy wartość funkcji dla x = 5.5

Znamy wartości funkcji f dla f(5) = 25 oraz f(6) = 36. Znajdujemy prostą przechodzącą przez oba punkty:

In[∘]:=

```
line=SolveValues[ { 25 == a * 5 + b, 36 == a * 6 + b    },{a,b}][[1]]
```

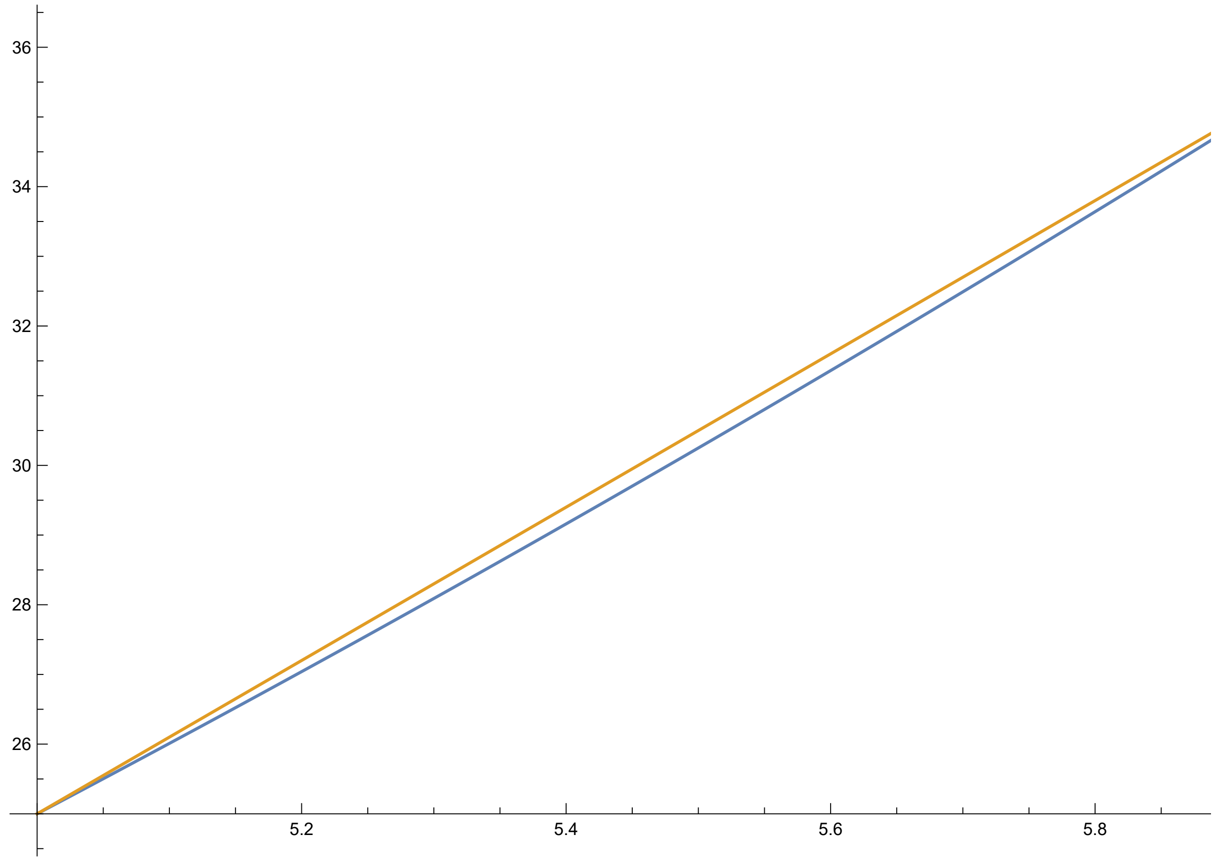
Out[∘]=

{11, -30}

Więc nasza interpolacja dla x∈ [a,b] to y = 11 x - 30. Dla x = 5.5, y = 30.5 (prawdziwa wartość to 30.25).

In[•]:= Plot[{x^2, 11 x - 30}, {x, 5, 6}]
wykres

Out[•]=



Interpolacja wielomianowa

Interpolacja Lagrange'a

Wielomianem Lagrange'a nazywamy wielomian $L(x) = \sum_{j=0}^{k} y_j * l_j(x)$, gdzie $l_j(x) = \prod_{\substack{0\leq m\leq k \\ m\neq j}} \frac{x-x_m}{x_j-x_m}$

Przykład: znajdź wielomian Lagrange'a dla podanych 4 punktów:

In[∘]:=

```
{x0, y0} = {0, 2};
{x1, y1} = {0.5, 1.125};
{x2, y2} = {1.5, -0.625};
{x3, y3} = {3, 8};
```

In[∘]:=

$$l0 = \frac{x - x1}{x0 - x1}\frac{x - x2}{x0 - x2}\frac{x - x3}{x0 - x3}$$
$$l1 = \frac{x - x0}{x1 - x0}\frac{x - x2}{x1 - x2}\frac{x - x3}{x1 - x3}$$
$$l2 = \frac{x - x0}{x2 - x0}\frac{x - x1}{x2 - x1}\frac{x - x3}{x2 - x3}$$
$$l3 = \frac{x - x0}{x3 - x0}\frac{x - x1}{x3 - x1}\frac{x - x2}{x3 - x2}$$

```
L = y0 * l0 + y1 * l1 + y2 * l2 + y3 * l3
Simplify[L]
```

uprość

Out[∘]=

0.444444 (3 – x) (–1.5 + x) (–0.5 + x)

Out[∘]=

0.8 (–3 + x) (–1.5 + x) x

Out[∘]=

–0.444444 (–3 + x) (–0.5 + x) x

Out[∘]=

0.0888889 (–1.5 + x) (–0.5 + x) x

Out[∘]=

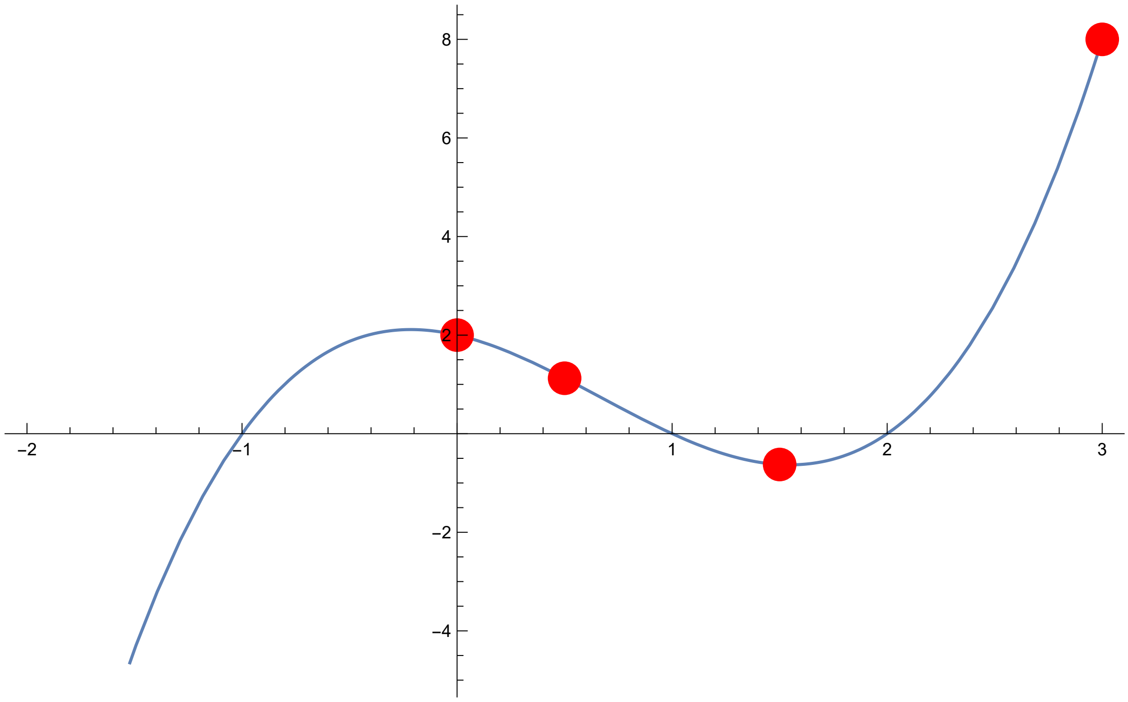
0.888889 (3 – x) (–1.5 + x) (–0.5 + x) + 0.9 (–3 + x) (–1.5 + x) x +
0.277778 (–3 + x) (–0.5 + x) x + 0.711111 (–1.5 + x) (–0.5 + x) x

Out[∘]=

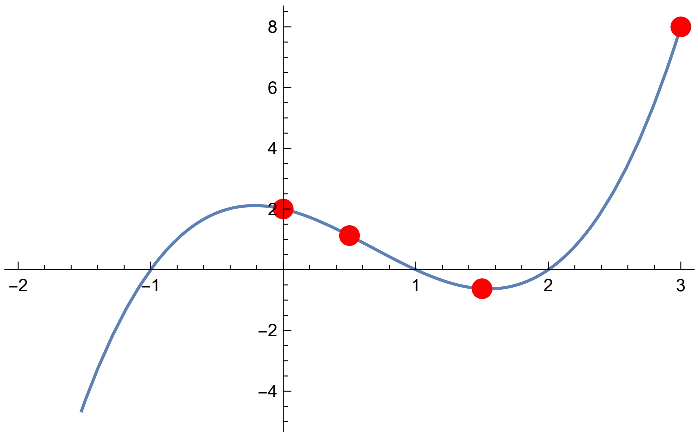
$2. - 1.\,x - 2.\,x^2 + 1.\,x^3$

In[ ]:=
```
Show[{Plot[{L}, {x, -2, 3}],
  Graphics[{Red, PointSize[0.03], Point[{{x1, y1}, {x2, y2}, {x3, y3}, {x0, y0}}]}]}]
```
pokaż wykres grafika cz··· rozmiar kropki punkt

Out[ ]=



Out[ ]=



Algorytm Neville'a

Alternatywny sposób obliczania wielomianu Lagrange'a:

$p_{ii} = y_i$

$p_{ij} = \frac{(x - x_i)\, p_{i+1\, j} - (x - x_j)\, p_{i\, j-1}}{x_j - x_i}$

poszukujemy $p_{0\, n}$

Out[ ]=

p00	p01	p02	p03	p04
	p11	p12	p13	p14
		p22	p23	p24
			p33	p34
				p44

In[ ]:=

```
{x0, y0} = {0, 2};
 {x1, y1} = {0.5, 1.125};
{x2, y2} = {1.5, -0.625};
{x3, y3} = {3, 8};
```

In[•]:=
```
p00 = y0
p11 = y1
p22 = y2
p33 = y3
```
$$p01 = \frac{(x - x0)\ p11 - (x - x1)\ p00}{x1 - x0}$$
$$p12 = \frac{(x - x1)\ p22 - (x - x2)\ p11}{x2 - x1}$$
$$p02 = \frac{(x - x0)\ p12 - (x - x2)\ p01}{x2 - x0}$$
$$p23 = \frac{(x - x2)\ p33 - (x - x3)\ p22}{x3 - x2}$$
$$p13 = \frac{(x - x1)\ p23 - (x - x3)\ p12}{x3 - x1}$$
$$p03 = \frac{(x - x0)\ p13 - (x - x3)\ p02}{x3 - x0}$$

Out[•]=
2

Out[•]=
1.125

Out[•]=
-0.625

Out[•]=
8

Out[•]=
2. (-2 (-0.5 + x) + 1.125 x)

Out[•]=
1. (-1.125 (-1.5 + x) - 0.625 (-0.5 + x))

Out[•]=
0.666667 (1. (-1.125 (-1.5 + x) - 0.625 (-0.5 + x)) x - 2. (-1.5 + x) (-2 (-0.5 + x) + 1.125 x))

Out[•]=
0.666667 (0.625 (-3 + x) + 8 (-1.5 + x))

Out[•]=
0.4 (-1. (-1.125 (-1.5 + x) - 0.625 (-0.5 + x)) (-3 + x) +
0.666667 (0.625 (-3 + x) + 8 (-1.5 + x)) (-0.5 + x))

Out[•]=
$\frac{1}{3}$ (0.4 (-1. (-1.125 (-1.5 + x) - 0.625 (-0.5 + x)) (-3 + x) +
0.666667 (0.625 (-3 + x) + 8 (-1.5 + x)) (-0.5 + x)) x - 0.666667 (-3 + x)
(1. (-1.125 (-1.5 + x) - 0.625 (-0.5 + x)) x - 2. (-1.5 + x) (-2 (-0.5 + x) + 1.125 x)))

In[•]:=
```
Simplify[p03]
```
uprość

Out[•]=
$2. - 1.\ x - 2.\ x^2 + 1.\ x^3$