Wolfram Mathematica

Technologie Informacyjne i Komunikacyjne R

Michał Parniak

mparniak@fuw.edu.pl www.qodl.eu

Na podstawie materiałów: Katarzyna Grzelak, Paweł Klimczewski

semestr zimowy 2026/27

Plan wykładu

Notatnik i składnia (zajęcia 1)
Algebra i analiza (zajęcia 1)
Listy i macierze (zajęcia 2)
Wykresy (zajęcia 2)
Równania różniczkowe i dane (zajęcia 2)
Wszystko jest wyrażeniem (zajęcia 3)
Programowanie funkcyjne (zajęcia 3)
Podsumowanie

Czym jest Mathematica

- Program do obliczeń **symbolicznych** (wzory, całki, równania) i **numerycznych**, z grafiką i własnym językiem programowania (*Wolfram Language*).
- Stephen Wolfram, 1988; dziś wersja 14.
- Licencja dla studentów UW (do zainstalowania na własnym komputerze); w pracowni OKWF zainstalowana.

Uruchamianie

`mathematica` — notatnik (okna, wykresy)
`math` — tryb tekstowy w terminalu
`math -script plik.wl` — skrypt

- Notatnik (`.nb`) = komórki z poleceniami, wynikami, tekstem i wykresami.
- Obliczenia wykonuje **jądro** (*kernel*) — pamięta wszystkie definicje z całej sesji, niezależnie od kolejności komórek na ekranie.

Praca w notatniku

Shift+Enter	wykonaj komórkę
Alt+.	przerwij obliczenia
F1 / `?Plot`	pomoc / opis funkcji
Esc a Esc	α (litery greckie)
Ctrl+/, **Ctrl+^**	ułamek, potęga (2D)
Ctrl+L	poprzednie polecenie
%, %5	ostatni wynik, `Out[5]`
; na końcu	nie wypisuj wyniku
`(* ... *)`	komentarz

- Komórki tekstowe i tytuły: *Format* → *Style* albo **Alt+1**. . . **Alt+7** — notatnik może być raportem.
- *Evaluation* → *Quit Kernel* — czysty start (zapomina wszystkie definicje).
- Dokumentacja (F1) ma przykłady dla każdej funkcji — zwykle najszybsza droga do rozwiązania.

Składnia: cztery rodzaje nawiasów i trzy znaki równości

`(a + b) c`	grupowanie
`Sin[x]`	argumenty funkcji
`{1, 2, 3}`	lista
`lista[[2]]`	element listy
`x = 5`	przypisanie
`f[x_] := x^2`	definicja (opóźniona)
`x^2 == 4`	równanie / porównanie
`x -> 3`	reguła podstawienia

- Funkcje wbudowane: **wielka litera**, nawiasy kwadratowe: `Sin[x]`, `Exp[x]`, `Sqrt[x]`, `Log[x]` (naturalny).
- Stałe: `Pi`, `E`, `I`, `Infinity`, `Degree`.
- Mnożenie: `a*b` albo `a b` (spacja!); `ab` to **jedna** nazwa.
- Własne nazwy zaczynamy **małą** literą (`omega`, `f`) — nie zderzą się z wbudowanymi (`N`, `E`, `C`, `D` są zajęte!).

Wynik dokładny i przybliżony

```
1/2 + 3/4           (* 5/4 *)
1/2 + 0.75          (* 1.25 *)
Sqrt[8]             (* 2 Sqrt[2] *)
Sqrt[-4]            (* 2 I *)
Cos[60 Degree]      (* 1/2 *)
E^(I Pi)            (* -1 *)
Cos[2]              (* Cos[2] *)
Cos[2.]             (* -0.416147 *)
N[Pi, 20]           (* 3.1415926535897932385 *)
Sqrt[2] // N        (* 1.41421 *)
Rationalize[0.125] (* 1/8 *)
```

- Liczby całkowite i ułamki są **dokładne**; Mathematica nie zaokrągla bez potrzeby.
- Kropka (`2.`) albo `N[...]` — przybliżenie zmiennoprzecinkowe.
- `N[wyr, 100]` — dowolna liczba cyfr.
- `wyr // f` to to samo co `f[wyr]` (zapis „na końcu").
- Dokładnie liczą się też duże liczby: `2026!` ma 5822 cyfry.
- Funkcje trygonometryczne biorą **radiany**; stopnie: `60 Degree`.

Zmienne i własne funkcje

```
omega = 2 Pi 10^6;      (* przypisanie *)
omega = .               (* usuń wartość *)
Clear[omega]            (* to samo *)
ClearAll["Global`*"]    (* usuń wszystko *)

f[x_] := x^2 + 5 x        (* funkcja *)
f[3]                      (* 24 *)
g[x_, y_] := Sqrt[x^2 + y^2]
g[3, 4]                   (* 5 *)
```

- `x_` — „dowolne wyrażenie, nazwijmy je `x`" (*wzorzec*).

`=` czy `:=`?

```
a = 2;
y1[x_] = a x;    (* teraz: 2 x *)
y2[x_] := a x;   (* przy użyciu *)
a = 10;
{y1[3], y2[3]}   (* {6, 30} *)
```

Najczęstszy błąd

Zmienna `x = 5` ustawiona godzinę temu psuje teraz `Solve[x^2 == 4, x]`. W razie dziwnych wyników: `ClearAll["Global`*"]` albo *Quit Kernel*.

Przekształcenia algebraiczne i podstawienia

```
Expand[(a + b)^3]
  (* a^3 + 3 a^2 b + 3 a b^2 + b^3 *)
Factor[x^3 + 6 x^2 + 11 x + 6]
  (* (1 + x) (2 + x) (3 + x) *)
Simplify[Sin[x]^2 + Cos[x]^2]    (* 1 *)
Simplify[Sqrt[x^2], x > 0]       (* x *)
Apart[1/(x (x + 1))]   (* 1/x - 1/(1 + x) *)
Together[1/x + 1/(x + 1)]
  (* (1 + 2 x)/(x (1 + x)) *)

x^2 + y /. {x -> 3, y -> 2}      (* 11 *)
```

- `Simplify`, `FullSimplify` (wolniej, dokładniej).
- Założenia: `Simplify[wyr, x > 0]` — bez nich $\sqrt{x^2} \neq x$!
- `wyr /. x -> 3` — podstaw 3 za x tylko w tym wyrażeniu, **bez** trwałego przypisania. Bezpieczniejsze niż `x = 3`.
- Inne: `Collect`, `Coefficient`, `Numerator`, `ComplexExpand`.

Równania

```
sol = Solve[x^2 - 3 x + 2 == 0, x]
  (* {{x -> 1}, {x -> 2}} *)
x /. sol                    (* {1, 2} *)
x^2 - 3 x + 2 /. sol        (* {0, 0}: sprawdzenie *)

Solve[{a x + y == 7, b x - y == 1}, {x, y}]
  (* {{x -> 8/(a+b), y -> -(a-7 b)/(a+b)}} *)

NSolve[x^5 - x - 1 == 0, x, Reals]
  (* {{x -> 1.1673}} *)
FindRoot[Exp[-x] == Sin[x], {x, 0.5}]
  (* {x -> 0.588533} *)
```

- W równaniach **podwójne** ==.
- Wynik to lista **reguł** — wartość wyciągamy przez `/.`.
- `Solve` — dokładnie, `NSolve` — numerycznie (wielomiany), `FindRoot` — jeden pierwiastek w pobliżu punktu startowego.
- Dla `FindRoot` najpierw **narysuj** obie strony, żeby wybrać punkty startowe.

Granice, sumy, pochodne, całki, szeregi

```
Limit[Sin[x]/x, x -> 0]                    (* 1 *)
Limit[1/x, x -> 0, Direction -> "FromAbove"] (* Infinity *)
Sum[k^2, {k, 1, n}]          (* n (1+n) (1+2 n)/6 *)
Sum[1/k^2, {k, 1, Infinity}]               (* Pi^2/6 *)
D[x^2 Sin[x], x]        (* x^2 Cos[x] + 2 x Sin[x] *)
D[Exp[-a t], {t, 2}]                (* a^2 E^(-a t) *)
Integrate[x Exp[-x], x]           (* -E^-x (1 + x) *)
Integrate[Exp[-x^2], {x, -Infinity, Infinity}]
                                        (* Sqrt[Pi] *)
Integrate[Exp[-a x^2], {x, -Infinity, Infinity},
  Assumptions -> a > 0]             (* Sqrt[Pi/a] *)
NIntegrate[Sin[x]/x, {x, 1, 10}]        (* 0.712265 *)
Series[Sin[x], {x, 0, 5}] // Normal
                          (* x - x^3/6 + x^5/120 *)
```

- `D[f, {x, n}]` — n-ta pochodna; `f'[x]` dla własnych funkcji.
- Całka nieoznaczona — bez stałej; sprawdzamy przez `D[...]` i `Simplify`.
- Parametry w całkach często wymagają `Assumptions`.
- Brak wzoru? `NIntegrate`, `NSum`.
- `Series` daje `O[x]^6`; `Normal` zamienia na wielomian.

Listy

```
Range[5]                (* {1, 2, 3, 4, 5} *)
Range[0, 1, 0.25]       (* {0., 0.25, ..., 1.} *)
Table[n^2, {n, 1, 5}]   (* {1, 4, 9, 16, 25} *)
Table[i + j, {i, 2}, {j, 3}]
                       (* {{2,3,4}, {3,4,5}} *)
v = {10, 20, 30, 40};
v[[2]]                  (* 20 *)
v[[-1]]                 (* 40: ostatni *)
v[[2 ;; 3]]             (* {20, 30} *)
m = {{1, 2}, {3, 4}};
m[[All, 2]]             (* {2, 4}: kolumna *)
```

- Numeracja od **1**.
- Działania na listach element po elemencie: `{1, 2, 3}^2` $\rightarrow$ `{1, 4, 9}`.
- `Length`, `Total`, `Mean`, `Max`, `Sort`, `Reverse`, `Append`, `Join`, `Flatten`, `Transpose`, `Dimensions`.
- Macierz = lista wierszy; tabela danych = lista punktów `{x, y}`.
- Pętlę `Do` + `AppendTo` prawie zawsze lepiej zastąpić przez `Table`.

Macierze i wektory

```
sx = PauliMatrix[1]; sy = PauliMatrix[2];
sx . sy                        (* {{I, 0}, {0, -I}} *)
sx . sy - sy . sx        (* {{2 I, 0}, {0, -2 I}} *)
m = {{1, 2}, {3, 4}};
m * m              (* {{1, 4}, {9, 16}}: NIE m.m! *)
Det[m]                                      (* -2 *)
Inverse[m]        (* {{-2, 1}, {3/2, -1/2}} *)
Eigensystem[sx] (* {{-1, 1}, {{-1, 1}, {1, 1}}} *)
{1, 2, 3} . {4, 5, 6}                       (* 32 *)
Cross[{1, 0, 0}, {0, 1, 0}]          (* {0, 0, 1} *)
Norm[{3, 4}]                                 (* 5 *)
MatrixExp[-I th sx/2] // FullSimplify
  (* {{Cos[th/2], -I Sin[th/2]},
      {-I Sin[th/2], Cos[th/2]}} *)
```

- Iloczyn macierzowy: **kropka** `.` ; gwiazdka mnoży element po elemencie.
- `IdentityMatrix[n]`, `Transpose`, `ConjugateTranspose`, `Tr`, `KroneckerProduct` (iloczyn tensorowy).
- `MatrixExp` — ewolucja $e^{-iHt/\hbar}$: obrót kubitu o kąt θ.

`MatrixForm` tylko do oglądania

Po `m = {{1,2},{3,4}} // MatrixForm` `m` przestaje być macierzą. Pisz:

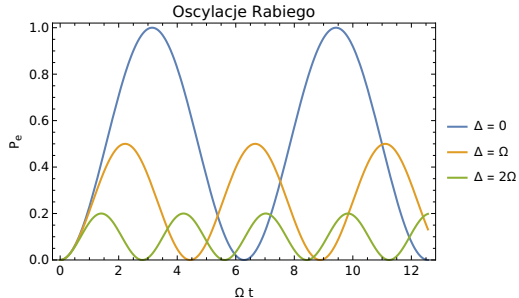
```
m = {{1,2},{3,4}};
m // MatrixForm
```

`Plot` i jego opcje

```
pe[t_, d_] := 1/(1 + d^2) *
              Sin[Sqrt[1 + d^2] t/2]^2

Plot[{pe[t, 0], pe[t, 1], pe[t, 2]},
  {t, 0, 4 Pi},
  PlotLegends -> {"\[CapitalDelta] = 0",
    "\[CapitalDelta] = \[CapitalOmega]",
    "\[CapitalDelta] = 2\[CapitalOmega]"},
  Frame -> True, FrameLabel -> {"t", "Pe"},
  PlotRange -> {0, 1.02},
  PlotLabel -> "Oscylacje Rabiego"]
```

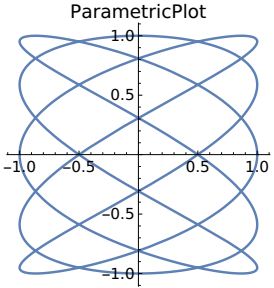
- Kilka funkcji: lista `{f, g, ...}`. Litery greckie: **Esc D Esc** (w tekście pliku: `\[CapitalDelta]`).
- Opcje: `PlotStyle`, `PlotRange`, `AxesLabel`, `GridLines`, `ImageSize`; wszystkie: `Options[Plot]`.



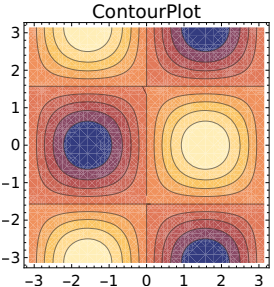
`PlotRange`

Domyślnie Mathematica **obcina** wysokie piki. Wierzchołek ucięty?
`PlotRange -> All`.

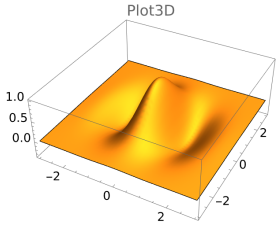
Inne wykresy



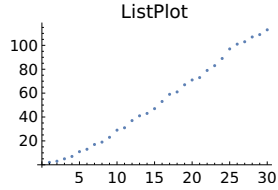

```
ParametricPlot[{Cos[5t],
Sin[3t]}, {t,0,2Pi}]
```




```
ContourPlot[Sin[x]Cos[y],
{x,-Pi,Pi}, {y,-Pi,Pi}]
```




```
Plot3D[f[x,y],
{x,-3,3}, {y,-3,3}]
```




```
ListPlot[Table[
{n,Prime[n]}, {n,30}]]
```

- Dane: `ListPlot`, `ListLinePlot`; skale: `LogPlot`, `LogLogPlot`.
- Krzywa uwikłana: `ContourPlot[x^2 + y^2 == 9, ...]`.
- Pola wektorowe: `VectorPlot`, `StreamPlot`.

```
p1 = Plot[...]; p2 = ListPlot[...];
Show[p1, p2]          (* nałóż wykresy *)
Export["wykres.pdf", Show[p1, p2]]
```

`Manipulate`: interaktywny wykres

```
Manipulate[
  Plot[n Exp[-a x] Sin[w x + phi], {x, 0, 100},
    PlotRange -> {-100, 100}],
  {{n, 100}, 0, 100},
  {{a, 0.03}, 0, 0.1},
  {{w, 0.5}, 0, 1},
  {phi, 0, Pi}]
```

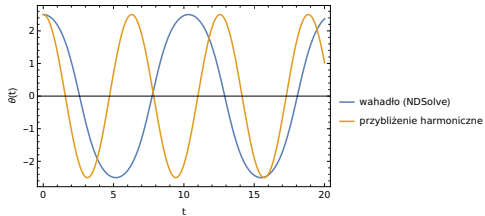
- Każdy parametr dostaje suwak: `{a, min, max}`; z wartością początkową: `{{a, a0}, min, max}`.
- Przydatne do zrozumienia wzoru i do znalezienia **wartości startowych** dopasowania.
- Ustaw `PlotRange` na sztywno — inaczej osie skaczą przy każdym ruchu suwaka.

Równania różniczkowe

```
DSolve[{x''[t] == -w^2 x[t],
  x[0] == 1, x'[0] == 0}, x[t], t]
                    (* {{x[t] -> Cos[t w]}} *)
DSolveValue[{y'[t] == -k y[t], y[0] == y0},
  y[t], t]                  (* y0 E^(-k t) *)

(* wahadło: brak wzoru -> numerycznie *)
th = NDSolveValue[{th''[t] == -Sin[th[t]],
  th[0] == 2.5, th'[0] == 0}, th, {t, 0, 20}];
Plot[{th[t], 2.5 Cos[t]}, {t, 0, 20}]
```

- Funkcja niewiadoma: `x[t]`, pochodne `x'[t]`, `x''[t]`; warunki też z `==`.
- `NDSolve` zwraca `InterpolatingFunction` — używa się jej jak zwykłej funkcji.

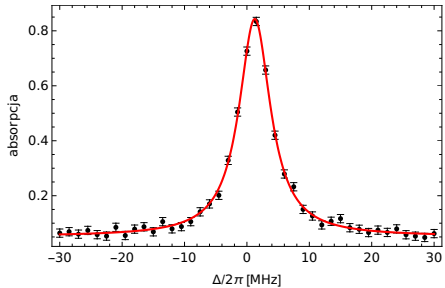


Duża amplituda: wahadło jest wolniejsze niż oscylator harmoniczny.

Dane z pliku i dopasowanie

```
SetDirectory[NotebookDirectory[]];
dane = Import["widmo.dat", "Table"]; (* x,y,dy *)
Dimensions[dane]                      (* {41, 3} *)
pkt = ListPlot[{#1, Around[#2, #3]} & @@@ dane];

fit = NonlinearModelFit[dane[[All, {1, 2}]],
   a/(1 + 4 (x - x0)^2/g^2) + c,
   {{a, 1}, {x0, 0}, {g, 5}, {c, 0}}, x,
   Weights -> 1/dane[[All, 3]]^2,
   VarianceEstimatorFunction -> (1 &)];
fit["ParameterTable"]
Show[pkt,
  Plot[fit[x], {x, -30, 30}, PlotRange -> All]]
```



linia D2: $\Gamma/2\pi = 6{,}15 \pm 0{,}15$ MHz

- `{{a, 1}, ...}` — wartości startowe (dobrane np. w `Manipulate`).
- Wagi $1/\sigma^2$ + `VarianceEstimatorFunction` $\rightarrow$ niepewności parametrów z niepewności punktów.
- Linia prosta: `LinearModelFit`; zapis: `Export`.

Wyrażenia, głowy, wzorce

```
FullForm[a + b c]
  (* Plus[a, Times[b, c]] *)
Head[x + y]          (* Plus *)
Head[{1, 2}]         (* List *)
Head[3.5]            (* Real *)

Cases[{1, a, 2.5, b, 3}, _Integer]
                     (* {1, 3} *)
silnia[0] = 1;
silnia[n_Integer?Positive] := n silnia[n - 1]
silnia[10]           (* 3628800 *)
silnia[2.5]          (* silnia[2.5]: nie pasuje *)
```

- Każde wyrażenie to `głowa[argumenty]` — także suma, lista, wykres.
- Dlatego te same narzędzia (`/.`, `Map`, `Apply`) działają na wszystkim.
- **Wzorce**: `_` dowolne wyrażenie, `x_` nazwane, `_Integer` liczba całkowita, `_?OddQ` spełniające warunek.
- Definicja pasuje tylko do argumentów zgodnych ze wzorcem — reszta zostaje nieobliczona.

Funkcje czyste (anonimowe, „lambdy”)

```
Function[x, x^2][3]       (* 9 *)
(#^2 &)[3]                (* 9 *)
(x |-> x^2)[4]            (* 16 *)
(#1 + 2 #2 &)[1, 10]      (* 21 *)

Select[Range[10], # > 6 &]
              (* {7, 8, 9, 10} *)
Select[Range[10], EvenQ[#] && # > 4 &]
              (* {6, 8, 10} *)
Plot[#, {x, 0, 1}] & /@ {x, x^2}
```

- **Funkcja bez nazwy**, tworzona tam, gdzie jest potrzebna.
- `#` — argument, `#1`, `#2` — kolejne argumenty, `&` — koniec funkcji.
- Trzy zapisy tego samego: `Function[x, ...]`, `... &`, `x |-> ...` (jak `lambda x: ...` w Pythonie).
- Typowe użycie: argument innych funkcji — `Select`, `Map`, `SortBy`, `NestList`.

Map, Apply i krótkie zapisy

```
f /@ {a, b, c}      (* {f[a],f[b],f[c]} *)
#^2 & /@ {1, 2, 3}        (* {1, 4, 9} *)
f @@ {a, b, c}            (* f[a, b, c] *)
Plus @@ {1, 2, 3}                  (* 6 *)
f @@@ {{1, 2}, {3, 4}}
                     (* {f[1,2], f[3,4]} *)
f @ x                            (* f[x] *)
{1, 2, 3} // Total                 (* 6 *)
Thread[{a, b} -> {1, 2}]
                     (* {a -> 1, b -> 2} *)
```

zapis	funkcja	działanie
`f /@ l`	`Map`	na każdy element
`f @@ l`	`Apply`	elementy jako argumenty
`f @@@ l`		`Apply` na każdym elemencie
`f @ x`		`f[x]`
`x // f`		`f[x]`
`wyr /. r`	`ReplaceAll`	podstawienie

- `@@@` przydaje się przy danych:
  `{#1, Around[#2, #3]} & @@@ dane`.

Iteracja: `Nest`, `FixedPoint`, `Fold`

```
Nest[f, x, 3]                      (* f[f[f[x]]] *)
NestList[2 # &, 1, 5]       (* {1, 2, 4, 8, 16, 32} *)

(* metoda Newtona dla Sqrt[2] *)
FixedPointList[(# + 2/#)/2 &, 1.]
  (* {1., 1.5, 1.41667, 1.41422, 1.41421, ...} *)
FixedPoint[(# + 2/#)/2 &, 1.]            (* 1.41421 *)

Fold[f, x, {a, b, c}]       (* f[f[f[x, a], b], c] *)
FoldList[Plus, 0, {1, 2, 3, 4}]  (* {0, 1, 3, 6, 10} *)
Accumulate[{1, 2, 3, 4}]             (* {1, 3, 6, 10} *)
```

- `Nest[f, x, n]` — złóż `f` *n* razy; `NestList` — wszystkie kroki (np. odwzorowanie logistyczne, mapy iterowane).
- `FixedPoint` — powtarzaj, aż wynik przestanie się zmieniać.
- `Fold` — „zwiń” listę funkcją dwóch argumentów; `FoldList` — z wynikami pośrednimi.

Proceduralnie czy funkcyjnie?

Proceduralnie (jak w C)

```
s = 0;
Do[s += k^2, {k, 100}];
s                                   (* 338350 *)

Module[{s = 0},      (* s lokalne *)
  Do[s += k^2, {k, 100}]; s]
If[3 > 2, "tak", "nie"]
Which[x < 0, "ujemna", True, "nieujemna"]
```

Funkcyjnie

```
Total[Range[100]^2]                 (* 338350 *)
```

- Krócej, szybciej, mniej miejsc na błąd (brak zmiennej `s`, którą trzeba wyzerować).
- `Do`, `For`, `While` są dostępne, ale zwykle zastępuje je `Table`, `Map`, `Nest`, `Total`.
- `Module[{zmienne}, ...]` — zmienne lokalne, które nie zaśmiecają notatnika.

Rekurencja i zapamiętywanie

```
fib[0] = 0; fib[1] = 1;
fib[n_] := fib[n] = fib[n - 1] + fib[n - 2]
fib[100]
  (* 354224848179261915075 *)
```

- `fib[n_] := fib[n] = ...` — policzona wartość zapisuje się jako nowa definicja (*memoizacja*).
- Bez tego `fib[100]` wymagałoby $\sim 10^{21}$ wywołań.

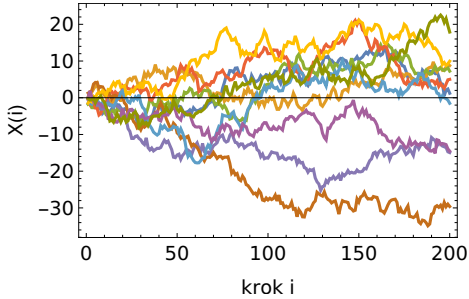
- Szczególne przypadki (`fib[0] = 0`) definiujemy osobno — Mathematica wybiera najbardziej szczegółową pasującą definicję.
- Głęboka rekurencja: `$RecursionLimit = 10^5`.
- Podgląd definicji: `?fib`.

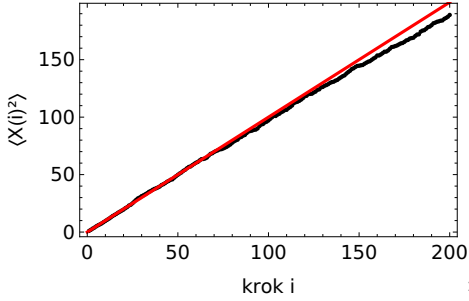
Przykład: błądzenie losowe w czterech linijkach

```
krok := RandomVariate[NormalDistribution[0, 1], 200]
walks = Table[Accumulate[krok], {10}];
ListLinePlot[walks]

many = Table[Accumulate[krok], {2000}];
Show[ListPlot[Mean[many^2]], Plot[i, {i, 0, 200}]]
```

- `Accumulate` — sumy częściowe kroków = położenie.
- `Mean[many^2]` — średnia po realizacjach, krok po kroku; teoria: $\langle X(i)^2 \rangle = i\sigma^2$.
- `krok :=` (nie =) — przy każdym użyciu nowe losowanie.





Pułapki i dobre praktyki

- = przypisuje, == porównuje, := definiuje, -> podstawia.
- `Sin(x)` to `Sin*x`; poprawnie `Sin[x]`.
- `ab` $\neq$ `a b`.
- Stare definicje psują nowe obliczenia: `ClearAll["Global`*"]` na początku notatnika.
- `MatrixForm`, `//N` tylko do wyświetlania na końcu.
- `N[...]` możliwie późno — dokładne obliczenia nie gubią cyfr.
- Notatnik porządkuj tytułami sekcji i komórkami tekstu — to także raport.
- Sprawdzaj wyniki: podstaw rozwiązanie (`/.`), zróżniczkuj całkę, narysuj funkcję.
- **Literatura**: dokumentacja (F1, sekcje *Examples*); S. Wolfram, *An Elementary Introduction to the Wolfram Language* (bezpłatnie online, `wolfram.com/language/elementary-introduction`).