Interpolacje - ciąg dalszy

Efekt Rungego - problemy interpolacja wielomianowej

In[ ]:=
```
f[x_] := 1/(1 + 25 x^2)
```

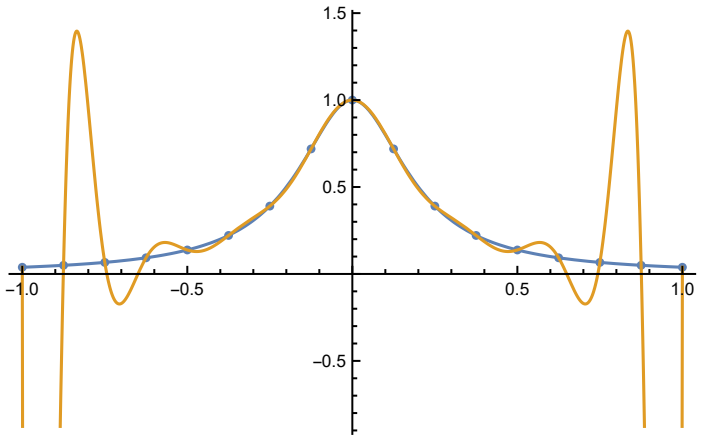
In[ ]:=
```
n = 16;
points = Table[{2 i/n - 1, f[2 i/n - 1]}, {i, 0, n}];
```

In[ ]:=
```
Lagrange = Simplify[
  Sum[points[[j, 2]] *
    Product[If[m ≠ j, (x - points[[m, 1]])/(points[[j, 1]] - points[[m, 1]]), 1], {m, 1, n + 1}], {j, 1, n + 1}]]
```

Out[ ]=
$$\left(170\,189\,292\,876\,156\,034 - 3\,876\,218\,985\,722\,260\,225\,x^2 + 59\,903\,899\,173\,085\,802\,500\,x^4 - 527\,646\,780\,474\,606\,000\,000\,x^6 + 2\,588\,025\,468\,896\,400\,000\,000\,x^8 - 7\,125\,820\,316\,160\,000\,000\,000\,x^{10} + 10\,848\,507\,904\,000\,000\,000\,000\,x^{12} - 8\,460\,697\,600\,000\,000\,000\,000\,x^{14} + 2\,621\,440\,000\,000\,000\,000\,000\,x^{16}\right) / 170\,189\,292\,876\,156\,034$$

In[ ]:=
```
Show[{ListPlot[points], Plot[{f[x], Lagrange}, {x, -1, 1}]}, PlotRange → All]
```

Out[ ]=



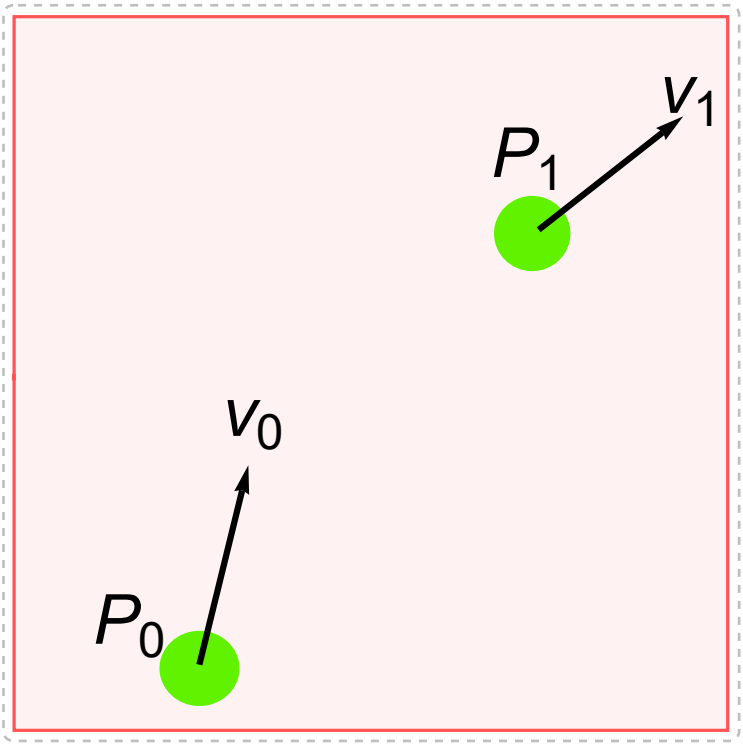
Zwiększenie ilości punktów użytych do interpolacji pogarsza (zwłaszcza na brzegach) jakość przybliżenia

Interpolacje - cubic Hermite spline

Spline - funkcja generująca krzywe

zbiór punktów -----> splines ---> krzywa

Problem: Mając dwa punkty oraz prędkości w tych punktach wyznacz trajektorię ruchu



Zakładamy, że trajektoria jest wielomianem 3 stopnia (zgodnie z nazwą "cubic Hermite spline"). Parametr t∈ [0,1] (analogicznie dla jak dla prostej, przechodzącej przez punkty P_p i P_k, którą możemy parametryzować przez $P_k\, t\, +\, (1\, -t\,)\, P_p$)

In[ ]:=
```
f[t_] := a t^3 + b t^2 + c t + d
```

Nasze warunki to:

In[ ]:=
```
Solve[{
   f[0] == P0,
   f[1] == P1,
   f'[0] == v0,
   f'[1] == v1

  }, {a, b, c, d}]
```

Out[ ]=

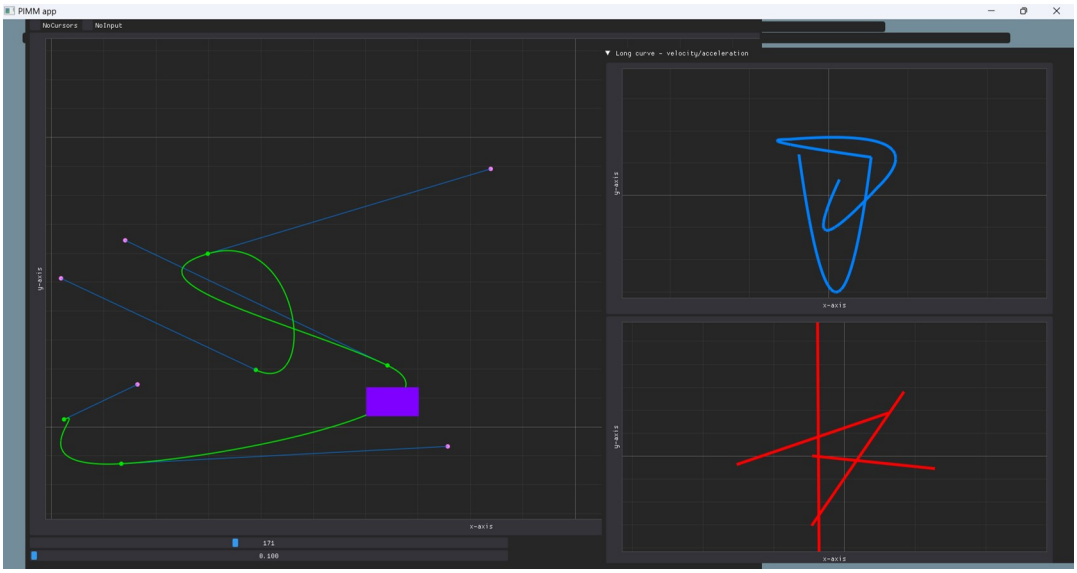
$$\{\{a \to 2\,P0 - 2\,P1 + v0 + v1,\ b \to -3\,P0 + 3\,P1 - 2\,v0 - v1,\ c \to v0,\ d \to P0\}\}$$

Wstawiając powyższe do funkcji f dostajemy krzywą spełniającą nasze warunki.
Nasz wynik możemy zapisać w postaci macierzowej:

In[ ]:=
```
Clear[P0, P1]
Collect[{1, t, t^2, t^3}.{{1, 0, 0, 0}, {0, 1, 0, 0}, {-3, -2, 3, -1}, {2, 1, -2, 1}}.{P0, v0, P1, v1}, t]
```

Out[ ]=
$$\{\{P0 + t\, v0 + t^2\, (-3\, P0 + 3\, P1 - 2\, v0 - v1) + t^3\, (2\, P0 - 2\, P1 + v0 + v1)\}\}$$

Dodatkowo, w załączonym programie można pobawić się krzywymi Hermite'a. W panelu po prawej pokazane są krzywe kreślone przez wektory prędkości i przyśpieszenia. W panelu dolnym pokazane są wykresy długości tych wektorów zależne od parametru t (tutaj jest on przeskalowany); Jak widać prędkość jest ciągła (wynika to z konstrukcji), a przyśpieszenie nie jest ciągłe.



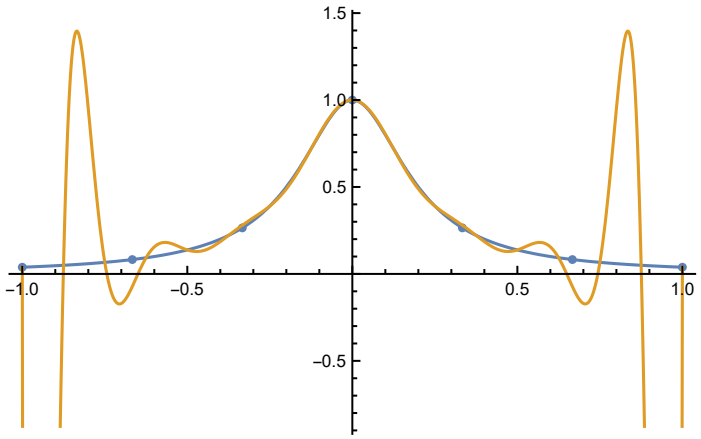
Efekt Runge'go II

Poprzednio mieliśmy problem z interpolacją wielomianową:

In[•]:=
$$f[x\_] := \frac{1}{1 + 25\, x^2}$$

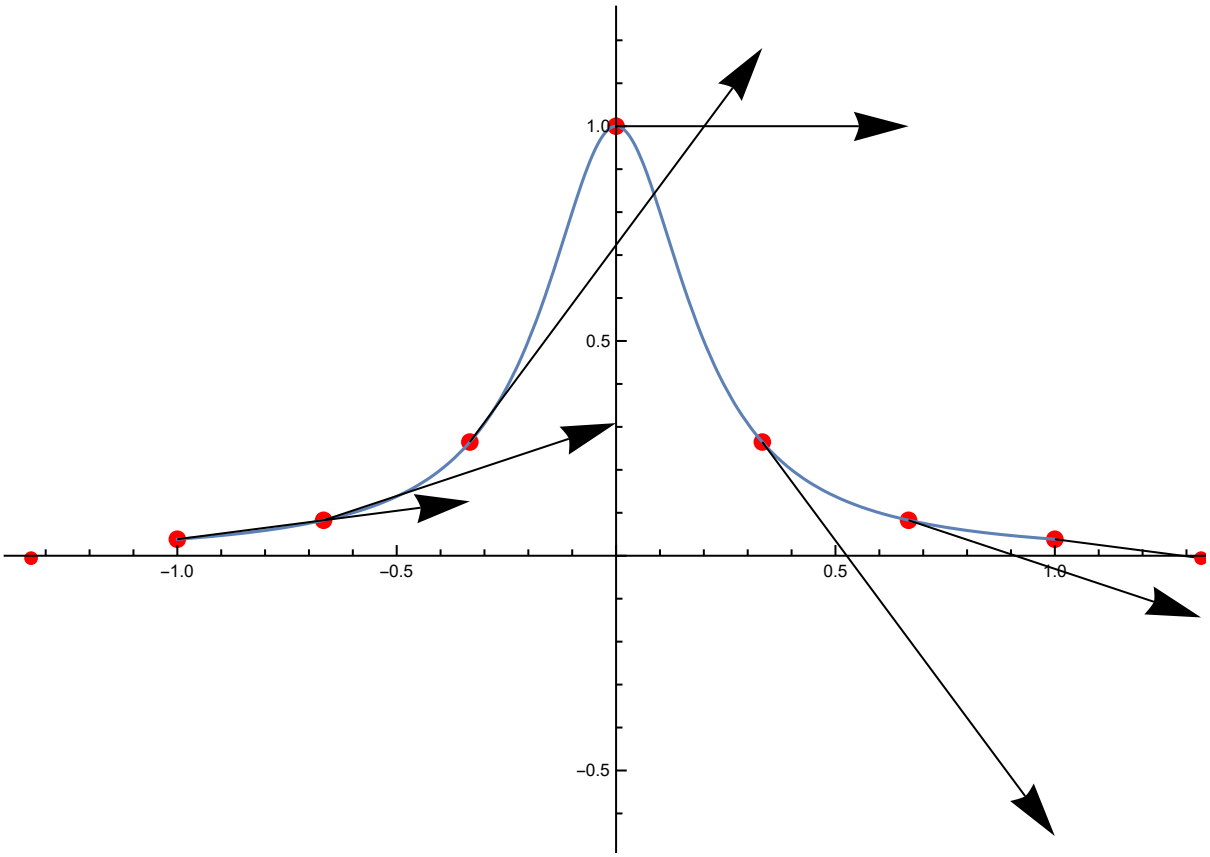
In[•]:=
```
n = 6;
points = Table[{2 i/n - 1, f[2 i/n - 1]}, {i, 0, n}];
Show[{ListPlot[points], Plot[{f[x], Lagrange}, {x, -1, 1}]}, PlotRange → All]
```

Out[•]=



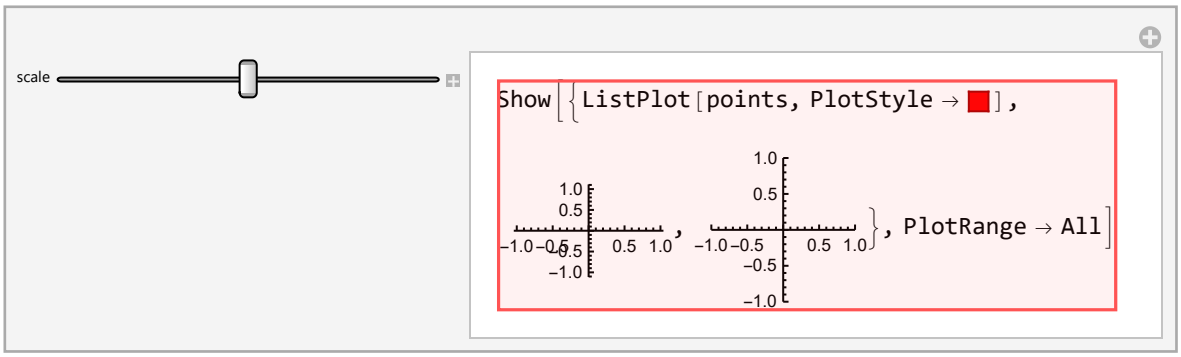
Mając zestaw punktów (bez prędkości), nadal możemy zastosować interpolację Hermita pomiędzy punktami. Musimy tylko oszacować prędkości początkowe. Najprostszy sposób osiągnięcia tego jest wyznaczenie wektora łączącego sąsiadów danego punktu i "przyczepienie" go do danego punktu. Problemem są tylko pierwszy i ostatni punkt (mają tylko po jednym sąsiedzie). Rozwiązaniem jest odbicie sąsiada względem punktu. Mając zestaw punktów i prędkości, możemy wyznaczyć krzywą Hermite'a pomiędzy każdą parą punktów.

Out[ ]=



Ostatnią modyfikacją jest wprowadzenie parametru (tutaj nazwanego “scale”), który skaluje wszystkie prędkości. Dla scale=1 mamy interpolację liniową. (Dla scale = 1/2 mamy tzw. Catmull−Rom spline)

Out[ ]=



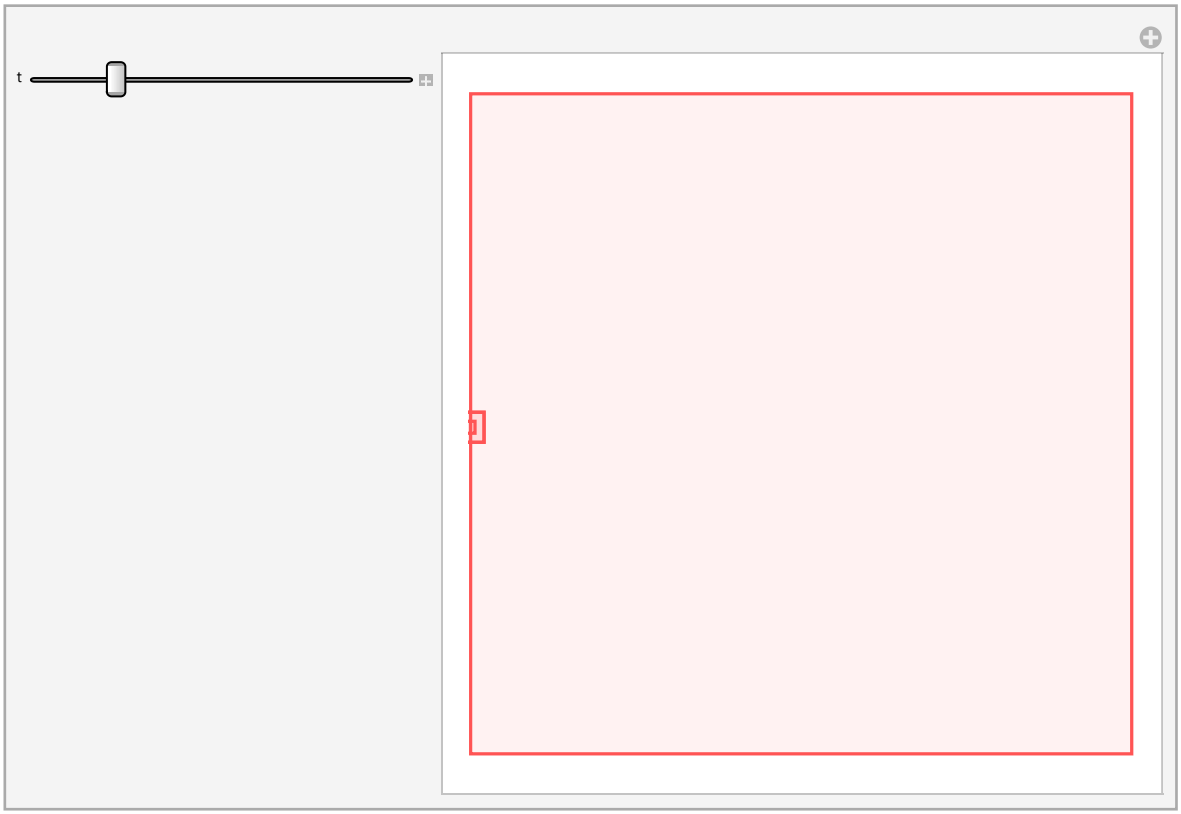
Krzywe Beziera

Ciekawym zastosowaniem interpolacji liniowej jest wyznaczanie krzywych Beziera (szeroko stosowanych w grafice komputerowej itp.)

In[ ]:=
```
{x, y} = {{1, 3}, {5, 0}}
```

Out[ ]=
```
{{1, 3}, {5, 0}}
```

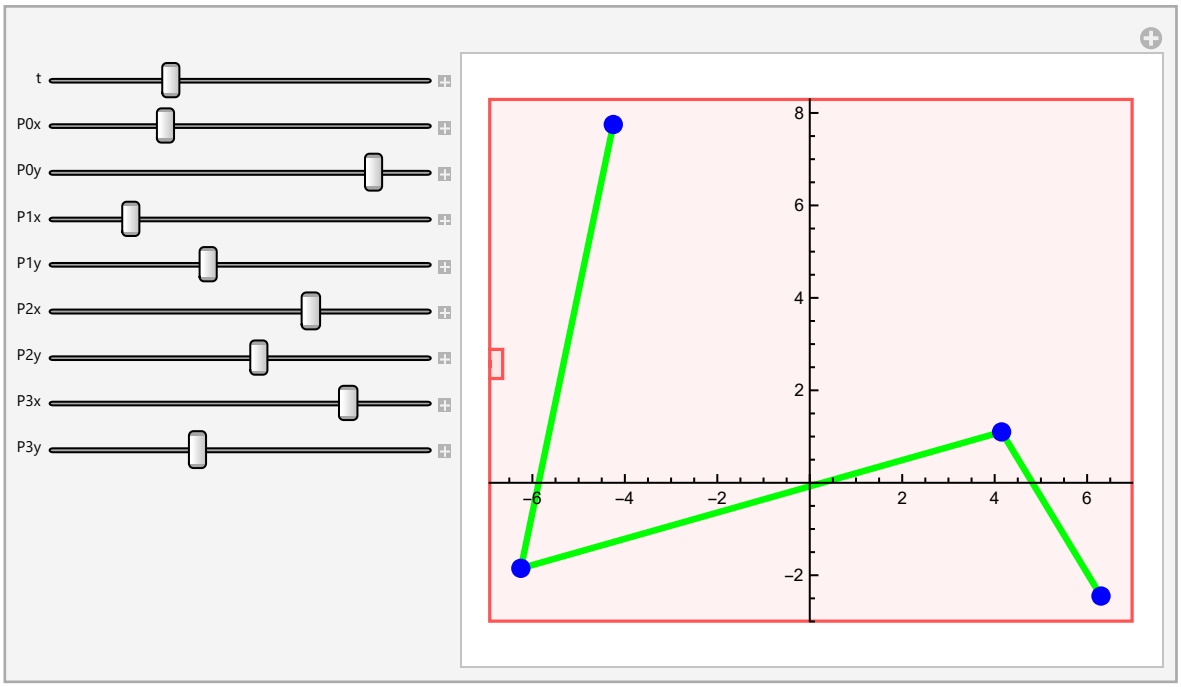
Out[ ]=



In[ ]:=
```
Lerp[t_, P0_, P1_] := t P1 + (1 - t) P0
```

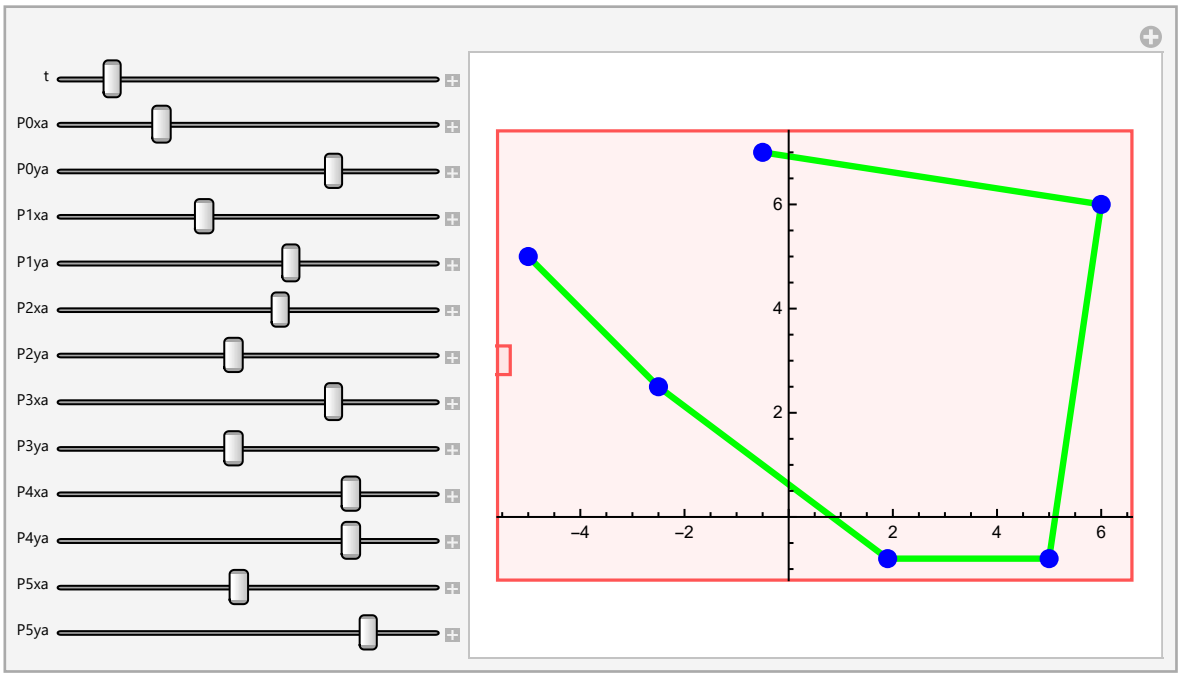
Powyżej mamy przykład zastosowania interpolacji linowej (t P_1+(1-t) P_0) do dwóch punktów. Poniżej mamy wyznaczoną krzywą Beziera (niebieska krzywa) daną przez 4 punkty. Krzywa ta powstaje przez interpolację pomiędzy kolejnymi punktami (zielone proste). Wszystkie te proste zależą od parametru t. Następnie stosujemy interpolację liniową pomiędzy otrzymanymi w poprzednim kroku punktami (czarne proste). Następnie stosujemy interpolację liniową na otrzymanych punktach (pomarańczowa prosta). Ruch punktu na tej prostej wyznacza krzywą Beziera stopnia 3. Powyższą strategię można nazwać: “interpoluj liniowo, dopóki nie skończą Ci się punkty do interpolowania”.

Out[ ]=



Zauważmy, że użyliśmy 6 interpolowań liniowych. Dodając więcej punktów liczba ta rośnie. Np. dla 6 punktów musimy użyć 15 interpolowań. Poniżej: krzywa Beziera stopnia 5.

Out[ ]=



Zauważmy, że krzywe Beziera przechodzą tylko przez punkt początkowy i końcowy, a reszta punktów kontroluje kształt krzywej.

Zauważmy, że stosujemy tu "rekurencyjną" metodę otrzymywania krzywej Bezier'a. Możemy jednak rozwinąć wyrażenie na krzywą i otrzymać postać wielomianową

In[•]:=
```
Clear[P0, P1, P2, P3]
a = Lerp[t, P0, P1];
b = Lerp[t, P1, P2];
c = Lerp[t, P2, P3];
d = Lerp[t, a, b];
e = Lerp[t, b, c];
Collect[Lerp[t, d, e], t]
```

Out[•]=
$$P0 + (-3\,P0 + 3\,P1)\,t + (3\,P0 - 6\,P1 + 3\,P2)\,t^2 + (-P0 + 3\,P1 - 3\,P2 + P3)\,t^3$$

lub macierzową

In[•]:=
$$\text{Collect}\left[(1\ t\ t^2\ t^3).\begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}.\begin{pmatrix} P0 \\ P1 \\ P2 \\ P3 \end{pmatrix}, t\right]$$

Out[•]=
$$\left\{\left\{P0 + (-3\,P0 + 3\,P1)\,t + (3\,P0 - 6\,P1 + 3\,P2)\,t^2 + (-P0 + 3\,P1 - 3\,P2 + P3)\,t^3\right\}\right\}$$

W ogólności $B_n(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i}\, t^i\, P_i$

W szczególności dla 4 punktów (liczymy od zera) mamy:

In[•]:=
$$\text{Collect}\left[\sum_{i=0}^{3} \text{Binomial}[3, i]\ (1-t)^{3-i}\ t^i\ P_i, t\right]$$

Out[•]=
$$P_0 + t\,(-3\,P_0 + 3\,P_1) + t^2\,(3\,P_0 - 6\,P_1 + 3\,P_2) + t^3\,(-P_0 + 3\,P_1 - 3\,P_2 + P_3)$$

In[•]:=
```
P[t_] := P0 + (-3 P0 + 3 P1) t + (3 P0 - 6 P1 + 3 P2) t^2 + (-P0 + 3 P1 - 3 P2 + P3) t^3
R[t_] := P4 + (-3 P4 + 3 P5) t + (3 P4 - 6 P5 + 3 P6) t^2 + (-P4 + 3 P5 - 3 P6 + P7) t^3
```

In[•]:=
```
Clear[P0, P1, P2, P3, P4, P5, P6, P7]
Solve[{
  P[1] == R[0]
 }, P4]
```

Out[•]=
$$\{\{P4 \to P3\}\}$$

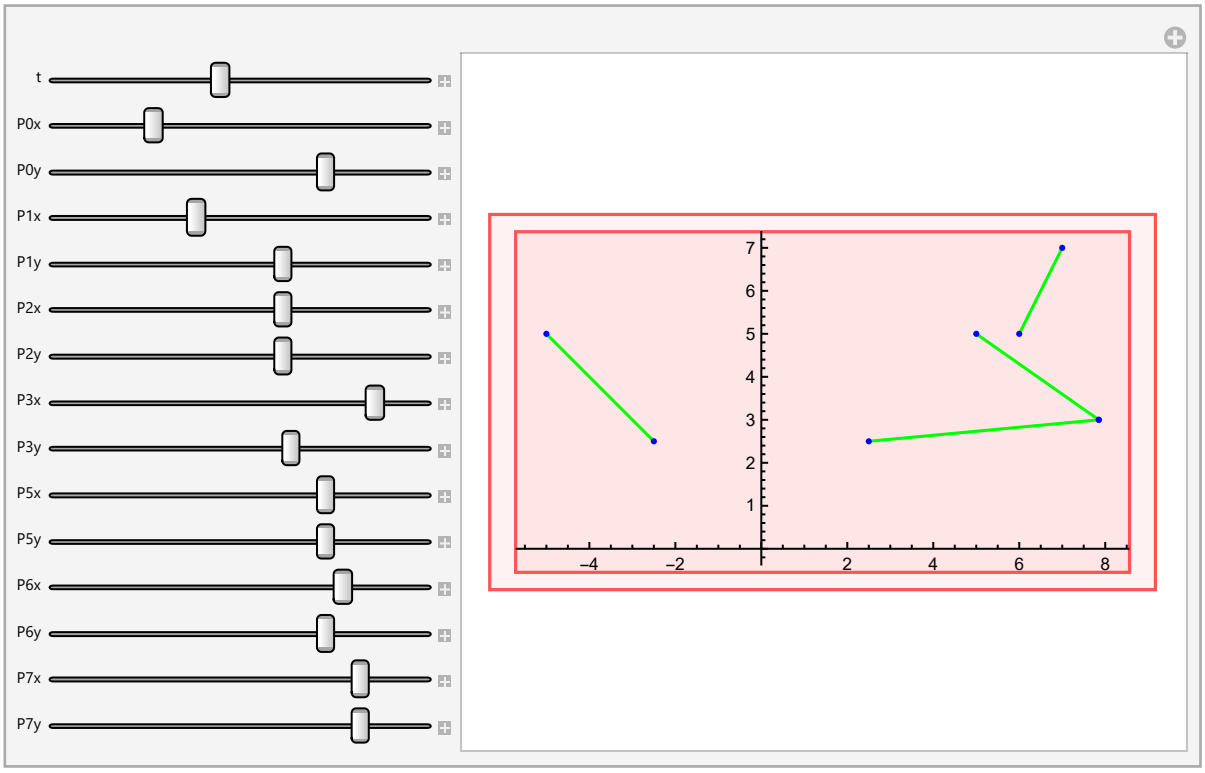
In[ ]:=
```
Manipulate[

 points2 = {P0, P1, P2, P3, P4, P5, P6, P7} = {{P0x, P0y}, {P1x, P1y},
    {P2x, P2y}, {P3x, P3y}, {P4x, P4y}, {P5x, P5y}, {P6x, P6y}, {P7x, P7y}};
 points2[[5]] = P3;
 P4 = P3;
 GraphicsColumn[{
   Show[{
     ParametricPlot[P[t1], {t1, 0, 1}],
     ParametricPlot[R[t1], {t1, 0, 1}],
     Graphics[{Green, Thickness[0.005], Line[{points2[[1]], points2[[2]]}]}],
     Graphics[{Green, Thickness[0.005], Line[{points2[[3]], points2[[4]]}]}],
     Graphics[{Green, Thickness[0.005], Line[{points2[[5]], points2[[6]]}]}],
     Graphics[{Green, Thickness[0.005], Line[{points2[[7]], points2[[8]]}]}],
     Graphics[{Blue, Thickness[0.01], PointSize[0.01], Point[points2]}],

     Graphics[{Red, PointSize[0.02], Point[{If[t < 1, P[t], R[t - 1]]}]}]
    }, PlotRange → All]

  }], {t, 0, 2},
 {{P0x, -5}, -10, 10}, {{P0y, 5}, -10, 10},
 {{P1x, -2.5}, -10, 10}, {{P1y, 2.5}, -10, 10},
 {{P2x, 2.5}, -10, 10}, {{P2y, 2.5}, -10, 10},
 {{P3x, 3}, -10, 10}, {{P3y, 3}, -10, 10},
 {{P5x, 5}, -10, 10}, {{P5y, 5}, -10, 10},
 {{P6x, 6}, -10, 10}, {{P6y, 5}, -10, 10},
 {{P7x, 7}, -10, 10}, {{P7y, 7}, -10, 10}]
```

Out[ ]=



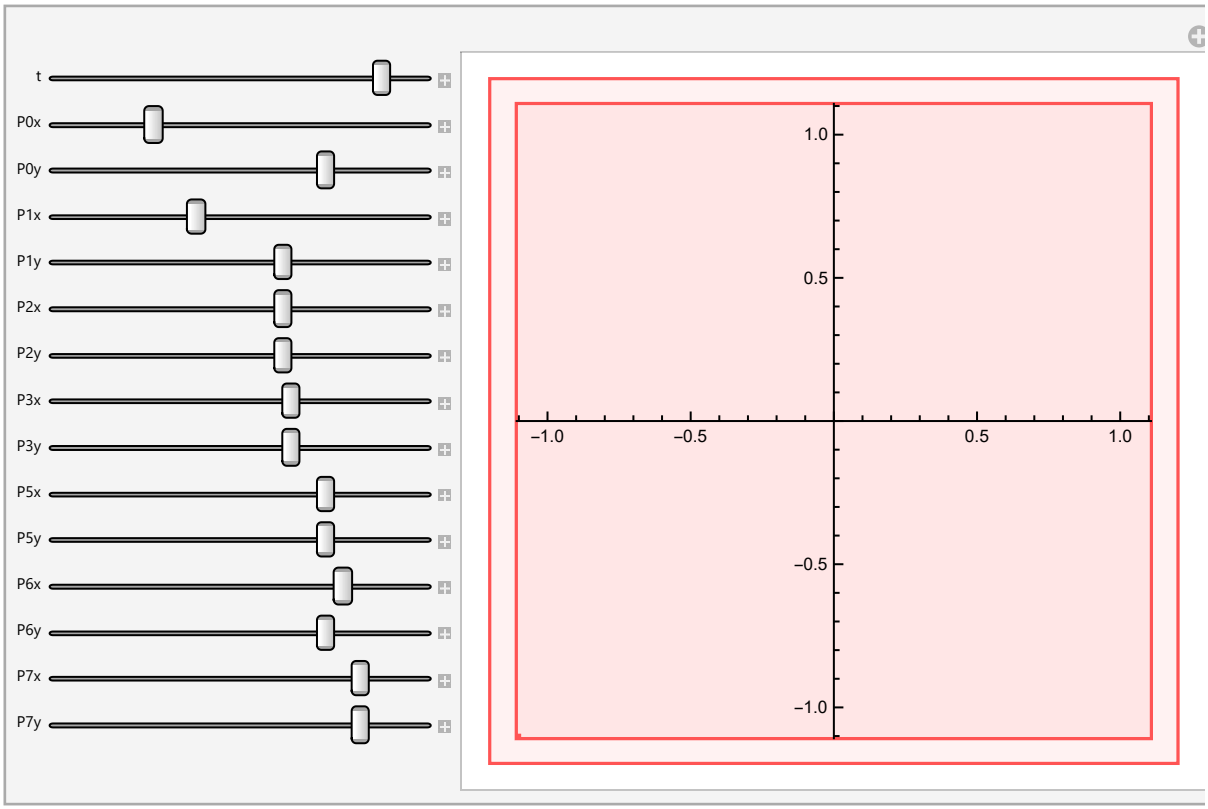
In[ ]:=
```
Manipulate[

 points2 = {P0, P1, P2, P3, P4, P5, P6, P7} = {{P0x, P0y}, {P1x, P1y},
    {P2x, P2y}, {P3x, P3y}, {P4x, P4y}, {P5x, P5y}, {P6x, P6y}, {P7x, P7y}};
 points2[[5]] = P3;
 P4 = P3;
 GraphicsColumn[{
   Show[{
     ParametricPlot[{P'[t1]}, {t1, 0, 1}],
     ParametricPlot[{R'[t1]}, {t1, 0, 1}],
     ParametricPlot[{P[t1], R[t1]}, {t1, 0, 1}, PlotStyle → Green],

     Graphics[{Red, PointSize[0.02], Point[{If[t < 1, P'[t], R'[t - 1]]}]}],
     Graphics[{Black, PointSize[0.02], Point[{If[t < 1, P[t], R[t - 1]]}]}]
    }, PlotRange → All]

  }], {t, 0, 2},
 {{P0x, -5}, -10, 10}, {{P0y, 5}, -10, 10},
 {{P1x, -2.5}, -10, 10}, {{P1y, 2.5}, -10, 10},
 {{P2x, 2.5}, -10, 10}, {{P2y, 2.5}, -10, 10},
 {{P3x, 3}, -10, 10}, {{P3y, 3}, -10, 10},
 {{P5x, 5}, -10, 10}, {{P5y, 5}, -10, 10},
 {{P6x, 6}, -10, 10}, {{P6y, 5}, -10, 10},
 {{P7x, 7}, -10, 10}, {{P7y, 7}, -10, 10}]
```

Out[ ]=

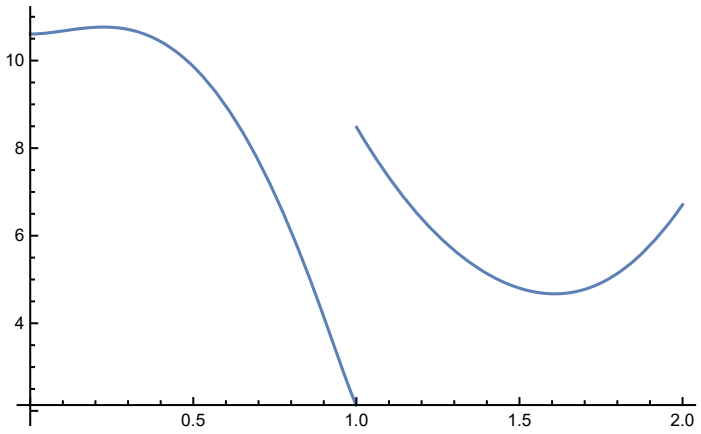


Zielona krzywa jest naszą krzywą Beziera. Niebieska krzywa pokazuje ruch wektora prędkości - widać nieciągłość. Poniższy wykres pokazuje normę wektora prędkości $\left(\sqrt{v^2}\right)$ od parametru afinicznego

(czasu), gdzie także widać nieciągłość

In[•]:=
```
Plot[If[t ≤ 1, Sqrt[P'[t][[1]]^2 + P'[t][[2]]^2],
  Sqrt[R'[t - 1][[1]]^2 + R'[t - 1][[2]]^2]], {t, 0, 2}]
```

Out[•]=



Możemy wymóc ciągłość prędkości:

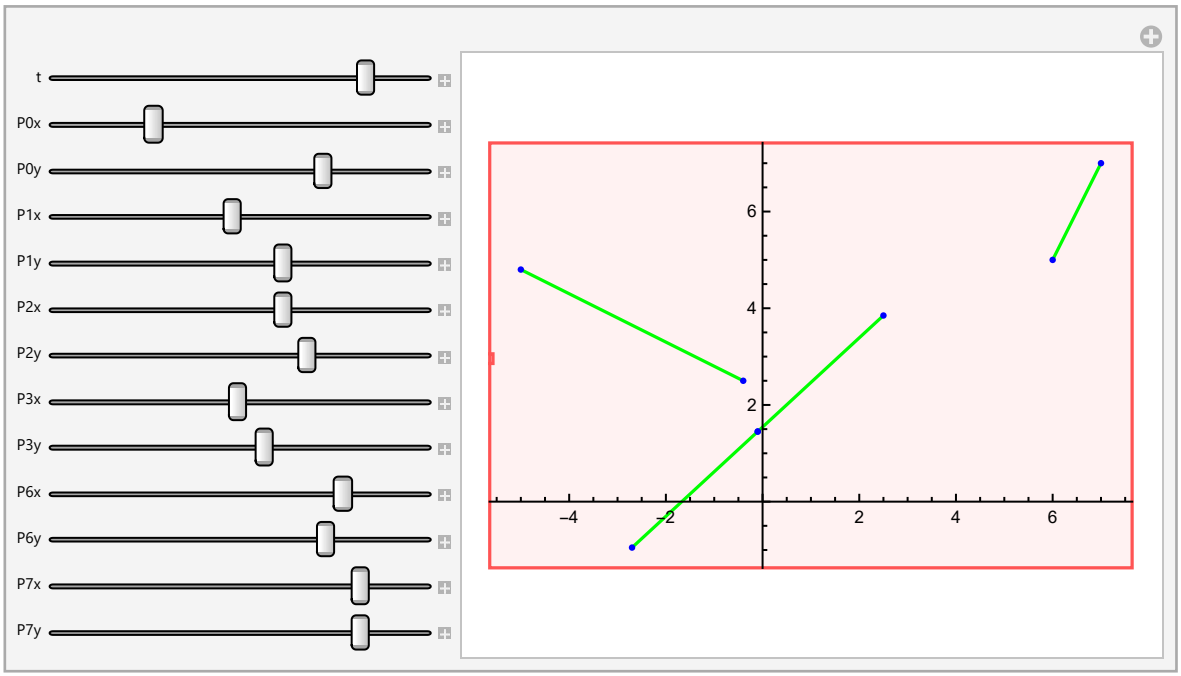
In[•]:=
```
Clear[P0, P1, P2, P3, P4, P5, P6, P7]

Solve[{
  P[1] == R[0],
  P'[1] == R'[0]
 }, {P4, P5}]
```

Out[•]=

$$\{\{P4 \to P3,\ P5 \to -P2 + 2\,P3\}\}$$

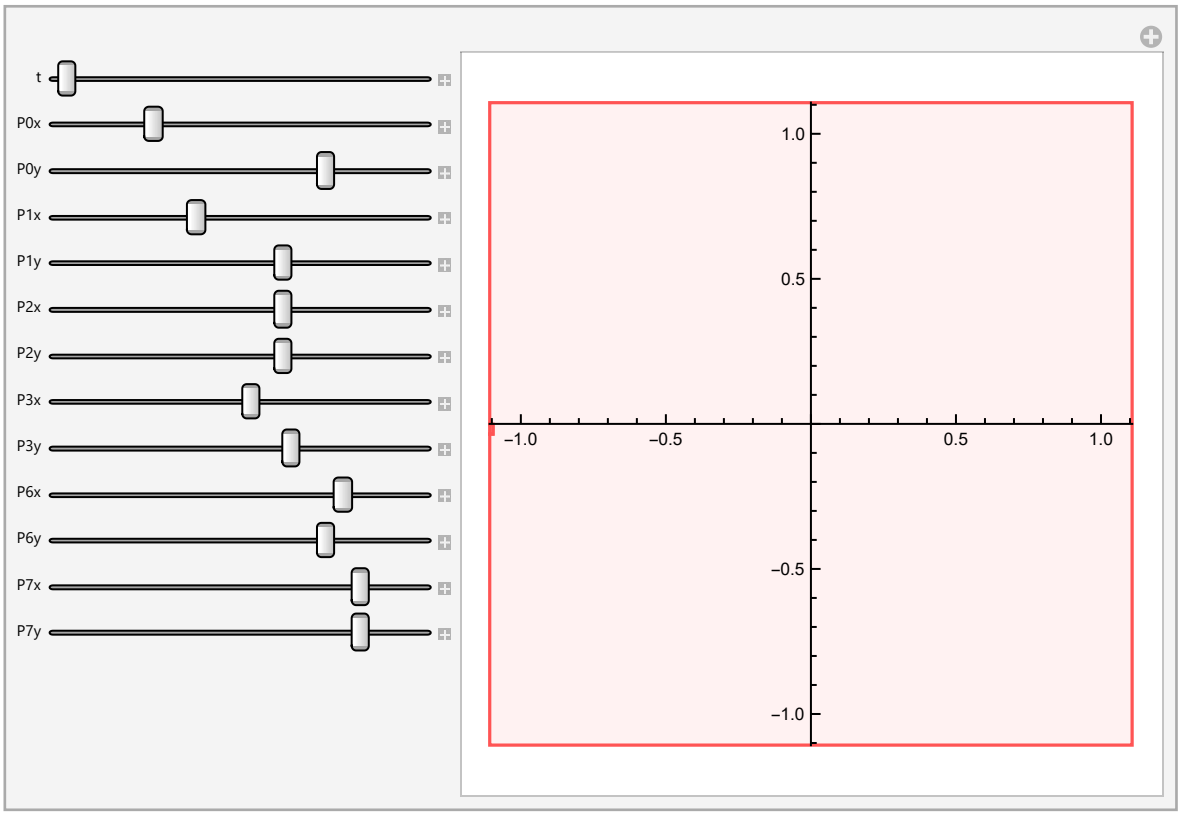
Out[•]=



In[ ]:=
```
Manipulate[
 points2 = {P0, P1, P2, P3, P4, P5, P6, P7} = {{P0x, P0y}, {P1x, P1y},
    {P2x, P2y}, {P3x, P3y}, {P4x, P4y}, {P5x, P5y}, {P6x, P6y}, {P7x, P7y}};
 points2[[5]] = P3;
 P4 = P3;
 points2[[6]] = -P2 + 2 P3;
 P5 = -P2 + 2 P3;

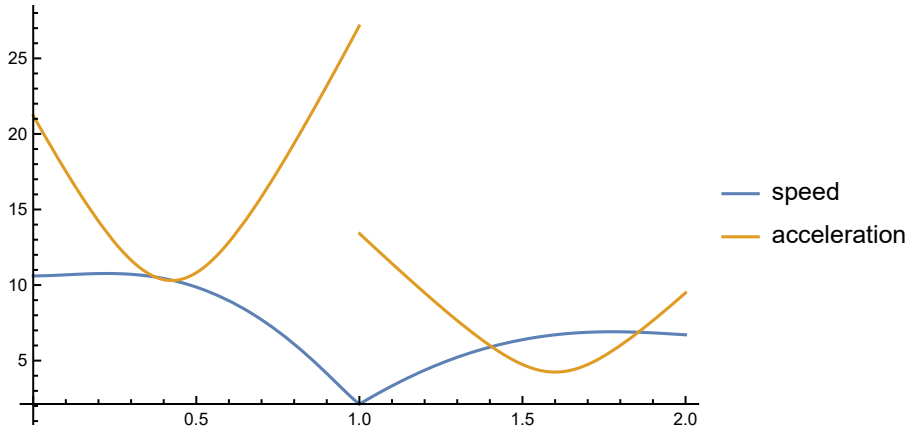
 Show[{
   ParametricPlot[P'[t1], {t1, 0, 1}],
   ParametricPlot[R'[t1], {t1, 0, 1}],
   ParametricPlot[{P[t1], R[t1]}, {t1, 0, 1}, PlotStyle → Green],
   Graphics[{Red, PointSize[0.02], Point[{If[t < 1, P'[t], R'[t - 1]]}]}],
   Graphics[{Black, PointSize[0.02], Point[{If[t < 1, P[t], R[t - 1]]}]}]
  }, PlotRange → All], {t, 0, 2},
 {{P0x, -5}, -10, 10}, {{P0y, 5}, -10, 10},
 {{P1x, -2.5}, -10, 10}, {{P1y, 2.5}, -10, 10},
 {{P2x, 2.5}, -10, 10}, {{P2y, 2.5}, -10, 10},
 {{P3x, 3}, -10, 10}, {{P3y, 3}, -10, 10},
 {{P6x, 6}, -10, 10}, {{P6y, 5}, -10, 10},
 {{P7x, 7}, -10, 10}, {{P7y, 7}, -10, 10}]
```

Out[ ]=



In[ ]:=
```
Plot[{If[t ≤ 1, Sqrt[P'[t][[1]]^2 + P'[t][[2]]^2], Sqrt[R'[t - 1][[1]]^2 + R'[t - 1][[2]]^2]],
  If[t ≤ 1, Sqrt[P''[t][[1]]^2 + P''[t][[2]]^2], Sqrt[R''[t - 1][[1]]^2 + R''[t - 1][[2]]^2]]},
 {t, 0, 2}, PlotLegends → {"speed", "acceleration"}]
```

Out[ ]=



Jak widać powyżej nasza krzywa stała się krzywą typu C^1 (ciągłość pierwszych pochodnych), ale drugie pochodne nie są ciągłe.

Możemy dodać kolejny warunek na ciągłość przyśpieszenia

```
Clear[P0, P1, P2, P3, P4, P5, P6, P7]
Solve[{
  P[1] == R[0],
  P'[1] == R'[0],
  P''[1] == R''[0]
 }, {P4, P5, P6}]
```

Out[ ]=

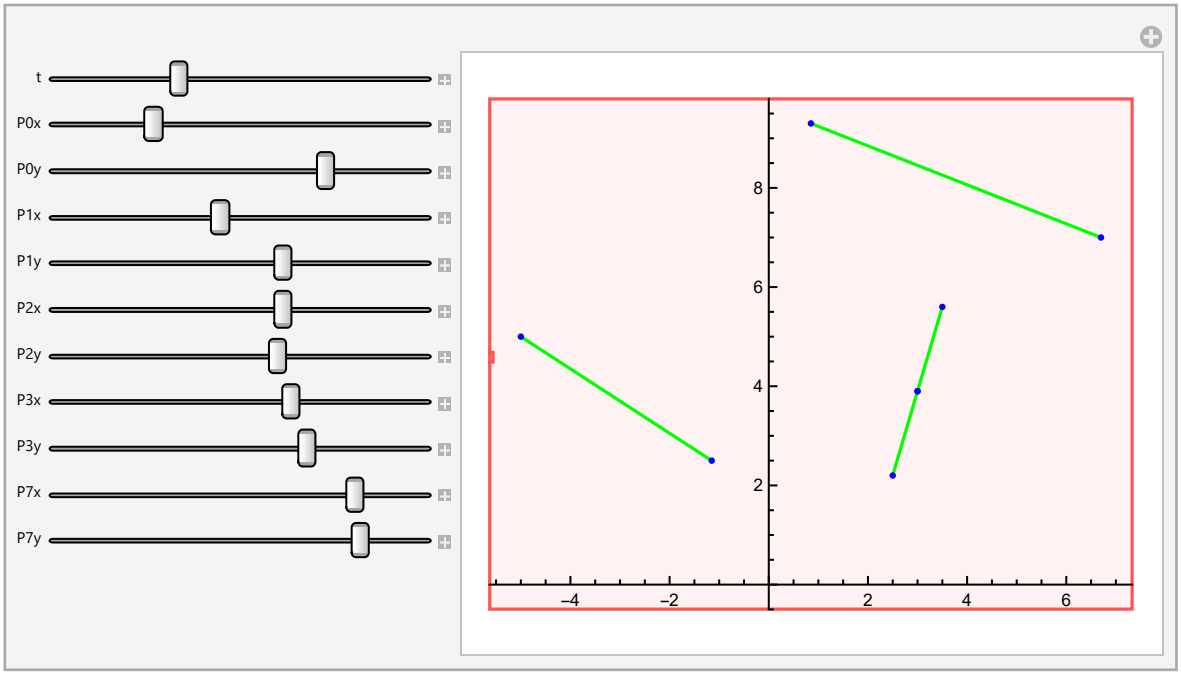
```
{{P4 → P3, P5 → -P2 + 2 P3, P6 → P1 - 4 P2 + 4 P3}}
```

In[ ]:=
```
Manipulate[

 points2 = {P0, P1, P2, P3, P4, P5, P6, P7} = {{P0x, P0y}, {P1x, P1y},
    {P2x, P2y}, {P3x, P3y}, {P4x, P4y}, {P5x, P5y}, {P6x, P6y}, {P7x, P7y}};
 points2[[5]] = P3;
 P4 = P3;
 points2[[6]] = -P2 + 2 P3;
 P5 = -P2 + 2 P3;
 points2[[7]] = P1 - 4 P2 + 4 P3;
 P6 = P1 - 4 P2 + 4 P3;
 Show[{
   ParametricPlot[P[t1], {t1, 0, 1}],
   ParametricPlot[R[t1], {t1, 0, 1}],
   Graphics[{Green, Thickness[0.005], Line[{points2[[1]], points2[[2]]}]}],
   Graphics[{Green, Thickness[0.005], Line[{points2[[3]], points2[[4]]}]}],
   Graphics[{Green, Thickness[0.005], Line[{points2[[5]], points2[[6]]}]}],
   Graphics[{Green, Thickness[0.005], Line[{points2[[7]], points2[[8]]}]}],
   Graphics[{Blue, Thickness[0.01], PointSize[0.01], Point[points2]}],

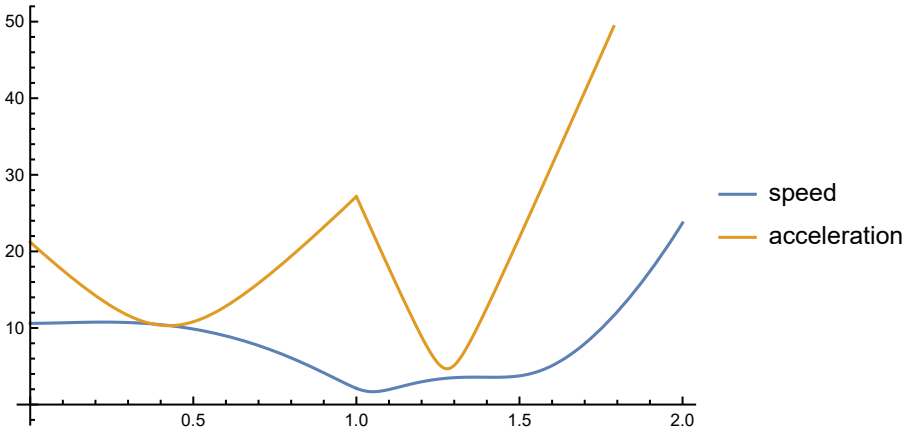
   Graphics[{Red, PointSize[0.02], Point[{If[t ≤ 1, P[t], R[t - 1]]}]}]
  }, PlotRange → All], {t, 0, 2},
 {{P0x, -5}, -10, 10}, {{P0y, 5}, -10, 10},
 {{P1x, -2.5}, -10, 10}, {{P1y, 2.5}, -10, 10},
 {{P2x, 2.5}, -10, 10}, {{P2y, 2.5}, -10, 10},
 {{P3x, 3}, -10, 10}, {{P3y, 3}, -10, 10},
 {{P7x, 7}, -10, 10}, {{P7y, 7}, -10, 10}]
```

Out[ ]=



Zwiększając gładkość krzywej tracimy lokalną kontrolę nad kształtem! Zmiana położenia punktu P_1 powoduje globalną zmianę na krzywej, a nie jak poprzednio tylko na pierwszym kawałku. Zazwyczaj C^1 jest wystarczająca.

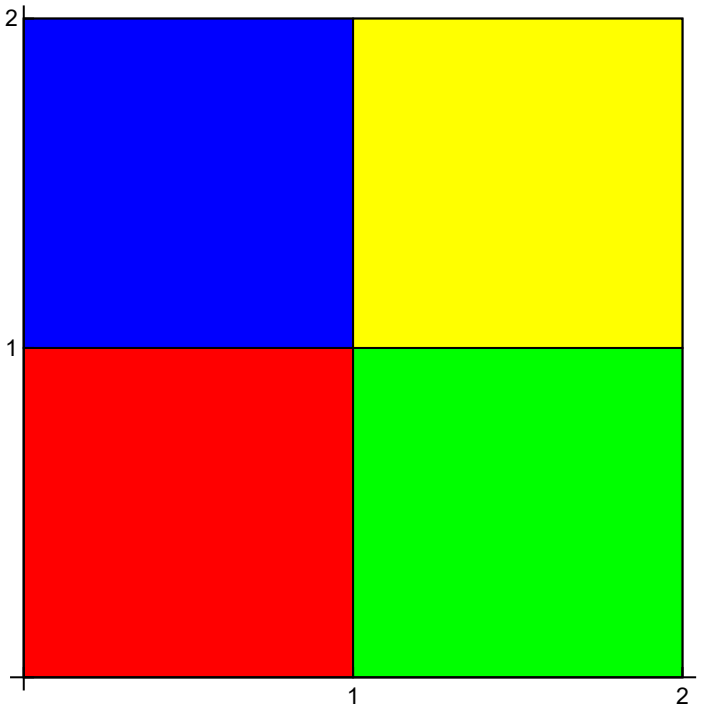
Out[ ]=



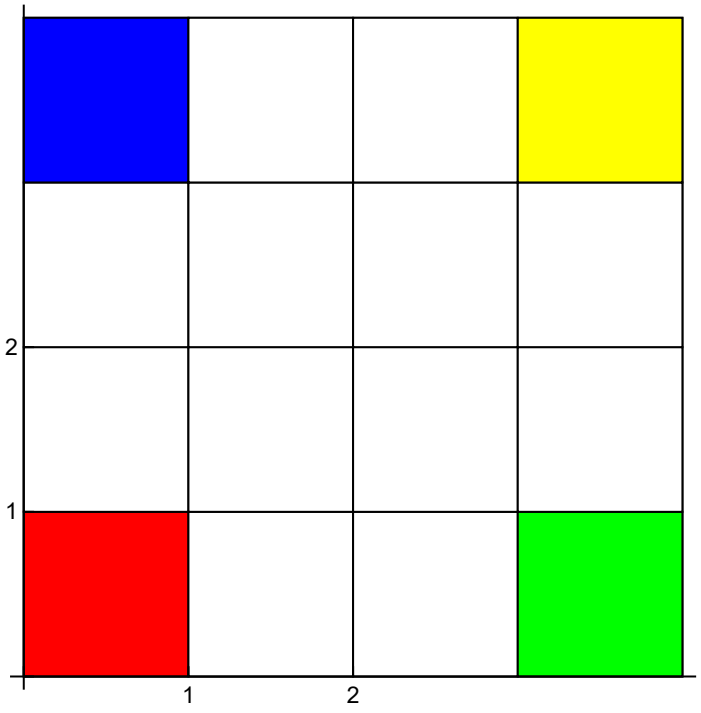
Zaawansowane przykłady interpolacji - DLSS (Deep learning super sampling)

- Super-rozdzielczość - renderuje klatkę w mniejszej rozdzielczości, a następnie zwiększa rozdzielczość obrazu przy użyciu AI

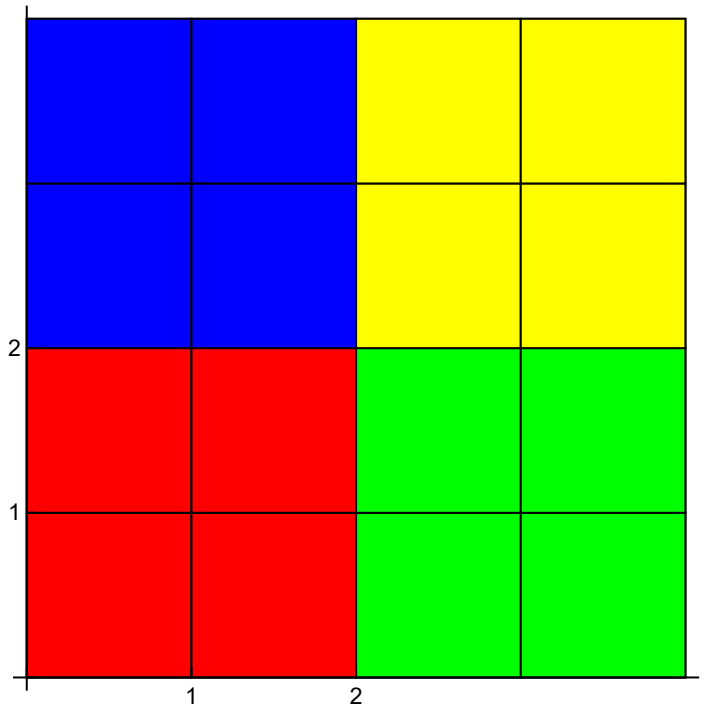
Out[ ]=



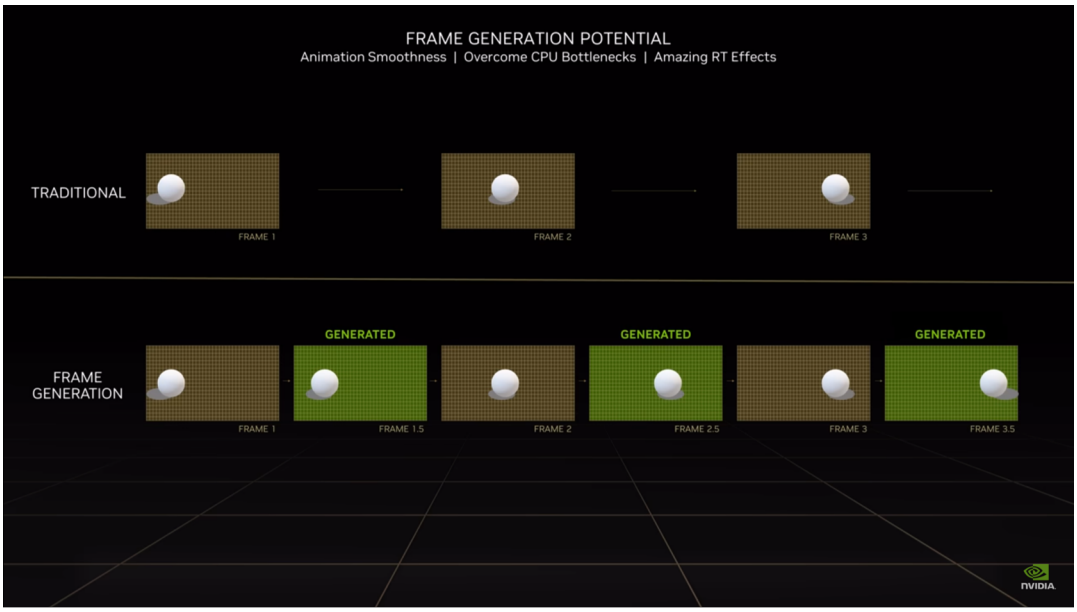
Out[ ]=



Out[ ]=



- Generowanie klatek - generowanie większej ilości klatek bez dużych strat na wydajności



In[ ]:= `Hyperlink["https://www.nvidia.com/pl-pl/geforce/technologies/dlss/"]`

Rozwiązywanie równań nieliniowych

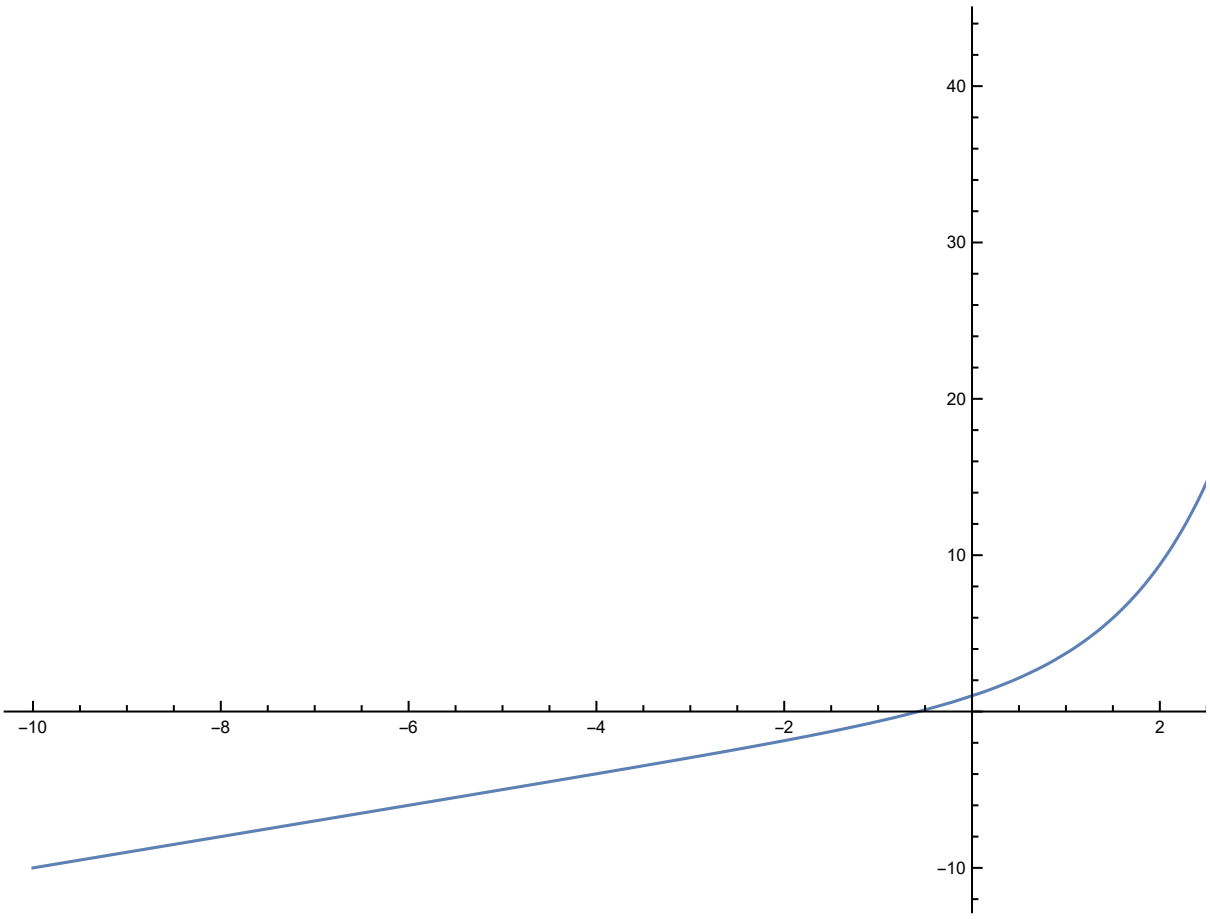
Problem: mając funkcję f (x), znajdź jej miejsca zerowe tj. takie x_0, że f $(x_0) = 0$

- Łatwe: $f(x) = x^2 + 5x - 4$
- Trudniejsze: f(x) = $x^3 - 3x^2 - 6x + 8$, g(x) = sin(2x-1) - $\frac{\pi}{2}$
- Rozwiązywalne w szczególnym przypadku: f(x) = $-24 + 2x + 29x^2 - 3x^3 - 5x^4 + x^5$
- Brak rozwiązania analitycznego f(x) = Log[6+x] $- x^2$

Twierdzenie Darboux:
Jeśli $f : [a, b] \to \mathbb{R}$ jest funkcją ciągłą na przedziale $[a, b]$ oraz jeśli wartości funkcji na krańcach przedziału ($f(a)$ i $f(b)$) mają różne znaki, to istnieje punkt $x_0 \in]a, b[$, taki że $f(x_0) = 0$.

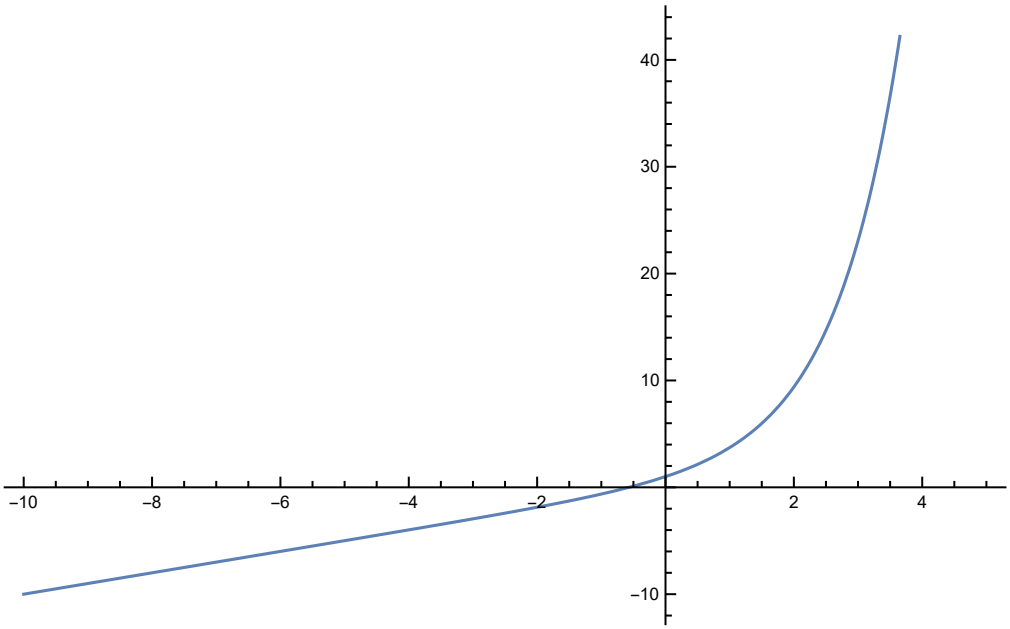
In[ ]:=
```
Plot[x + Exp[x], {x, -10, 5.}]
```

Out[ ]=



Problem: Rozwiąż równanie x + E^x = 0

Out[ ]=



Z analizy funkcji $f(x) = x + E^x$ wiemy, że:

- f(x) jest ściśle rosnąca oraz $\lim_{x\to-\infty} f(x) = -\infty$ oraz $\lim_{x\to\infty} f(x) = \infty$ więc równanie ma tylko jedno rozwiązanie
- f(0) = 1 > 0
- f(-2) = -2 + $\frac{1}{E^2}$ < 0

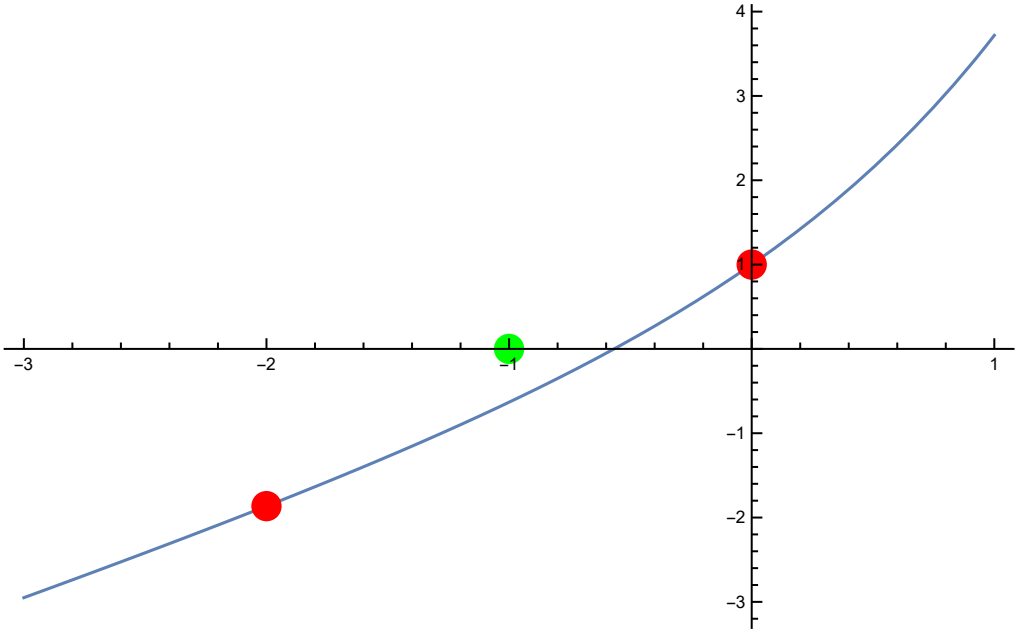
Zatem $f(x)$ ma miejsce zerowe na przedziale] − 2, 0[

Metoda bisekcji

- Znajdź przedział [a, b] gdzie f(a) f(b) < 0
- oblicz punkt środkowy: $x_{\text{mid}} = \frac{a+b}{2}$
- oblicz $f(x_{\text{mid}})$; jeśli dokładność ($|f(x_{\text{mid}})| < \epsilon$ albo $|b-a| < \epsilon$) jest wystarczająca to przerwij algorytm, miejsce zerowe to $x_0 \simeq x_{\text{mid}}$
- skróć przedział [a, b]; w zależności od znaku $f(x_{\text{mid}})$, a = x_{mid} albo $b = x_{\text{mid}}$

In[•]:=
```
f[x_] := x + Exp[x]
```

Out[•]=



W naszym przypadku, a = -2 i b =0, więc w pierwszym przybliżeniu $x_0 \simeq -1$, ale dokładność przybliżenia jest słaba

In[•]:=
```
N@f[-1]
```

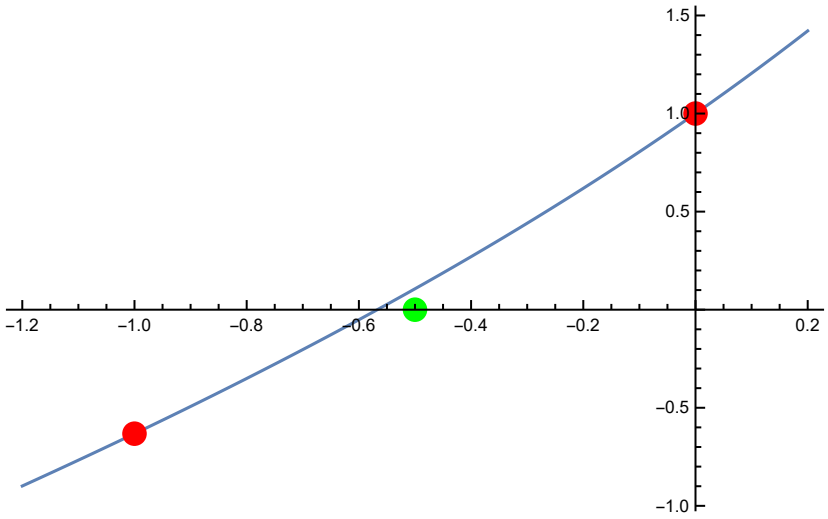
Out[•]=
```
-0.632121
```

- Naszym zmniejszonym przedziałem musi być]-1, 0[ponieważ f(-1) * f(0) < 0
- Następne przybliżenie: $x_{\text{mid}} = \frac{-1+0}{2} = -\frac{1}{2}$

In[•]:=
```
N@f[-1 / 2]
```

Out[•]=
```
0.106531
```

Out[ ]=

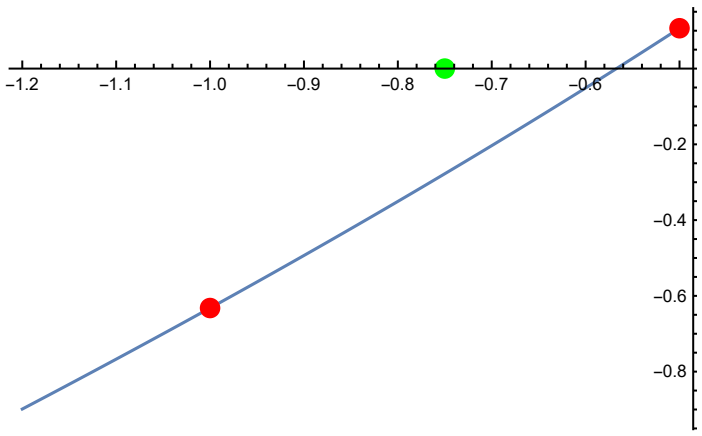


- Zmniejszamy przedział; tym razem f(-1/2) * f(-1)<0, więc nowy przedział to]-1, $\frac{-1}{2}$[
- $x_{\text{mid}} = \frac{-1+(-1/2)}{2} = -3/4$

In[ ]:=
```
N@f[-3 / 4]
```

Out[ ]=
```
-0.277633
```

Out[ ]=



In[ ]:=
```
Clear[a, b, c]
a = -2;
b = 0;
c = -1;
sol = {};
While[Abs[f[c]] > 0.0001,
 c = (b + a)/2;
 If[f[c] * f[a] < 0, b = c];
 If[f[c] * f[b] < 0, a = c];
 AppendTo[sol, c];
]
```

In[•]:=
```
N@c
N@f[c]
Length[sol]
N@sol
```

Out[•]=
```
-0.567139
```

Out[•]=
$$7.23791 \times 10^{-6}$$

Out[•]=
```
13
```

Out[•]=
```
{-1., -0.5, -0.75, -0.625, -0.5625, -0.59375, -0.578125,
 -0.570313, -0.566406, -0.568359, -0.567383, -0.566895, -0.567139}
```

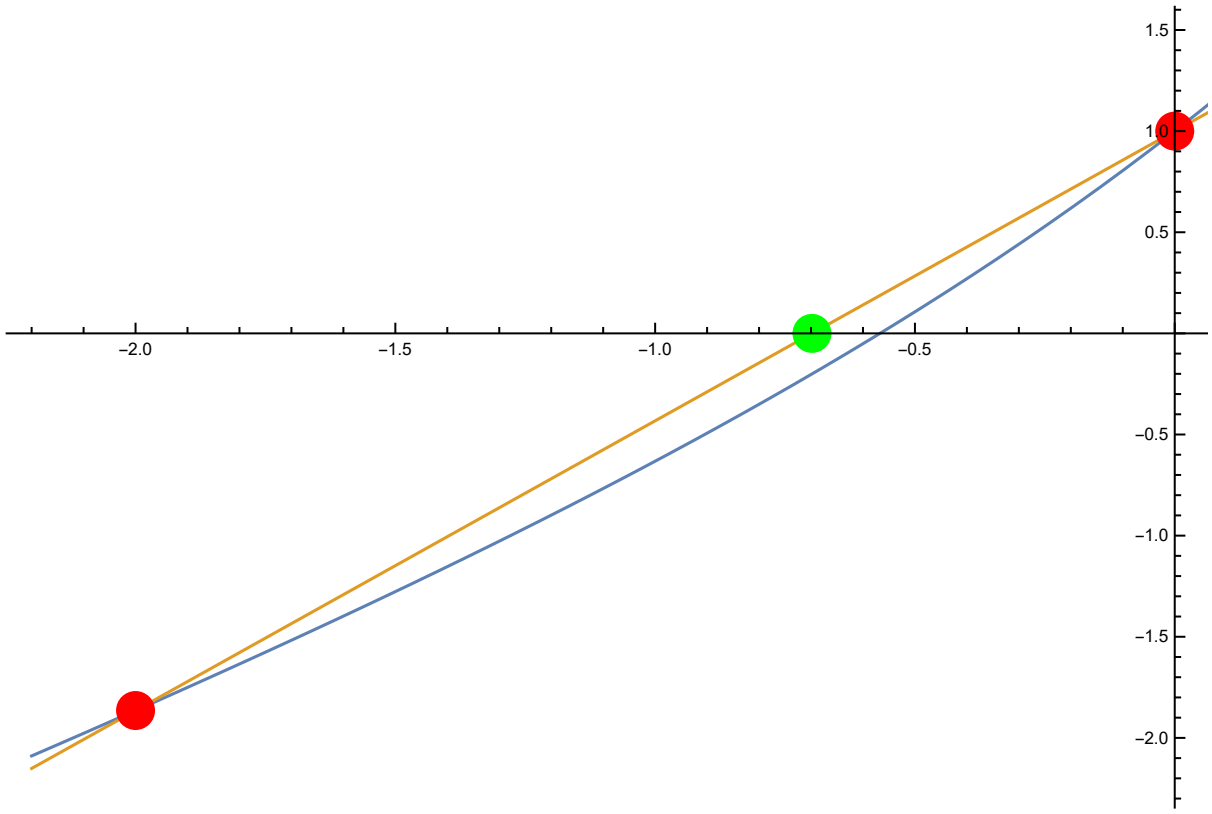
- Prosta i bezpieczna
- Wolna

Metoda falsi

- Znajdź przedział [a, b] gdzie f (a) f (b) < 0
- Interpoluj liniowo funkcję f między punktami a i b
  - Rozwiąż układ równań $\begin{cases} f(a) = A\,a + B \\ f(b) = A\,b + B \end{cases}$
  - funkcja interpolująca g(x) = $\frac{f(b)-f(a)}{a-b}\,x - \frac{b\,f(a)-a\,f(b)}{a-b}$
- Znajdź miejsce zerowe x_{lin} otrzymanej funkcji liniowej
  - g(x)=0 => $\frac{f(a)-f(b)}{a-b}\,x_{\text{lin}} - \frac{b\,f(a)-a\,f(b)}{a-b}$ =0 =>$x_{\text{lin}} = \frac{b\,f(a)-a\,f(b)}{f(a)-f(b)}$
- oblicz $f(x_{\text{lin}})$; jeśli dokładność ($|f(x_{\text{lin}})| < \epsilon$ albo $|b-a| < \epsilon$) jest wystarczająca to przerwij algorytm, miejsce zerowe to $x_0 \simeq x_{\text{mid}}$
- skróć przedział [a, b]; w zależności od znaku $f(x_{\text{lin}})$, a = x_{lin} albo $b = x_{\text{lin}}$

In[ ]:=
```
a = -2;
b = 0;
Show[{
  Plot[{x + Exp[x], (f[a] - f[b])/(a - b) x - (b f[a] - a f[b])/(a - b)}, {x, -2.2, 0.2}],
  Graphics[{Red, PointSize[0.03], Point[{{-2, f[-2]}, {0, f[0]}}]}],
  Graphics[{Green, PointSize[0.03], Point[{{(b f[a] - a f[b])/(f[a] - f[b]), 0}}]}]
 }]
```

Out[ ]=



In[ ]:=
```
N@f[(b f[a] - a f[b])/(f[a] - f[b])]
```

Out[ ]=
```
-0.200663
```

- Skracanie przedziału działa tak jak w metodzie bisekcji; $f(x_{lin})\, f(0) < 0$ więc a = x_{lin}

In[ ]:=
```
Clear[a, b, c]
a = -2;
b = 0;
c = -2;
sol = {};
While[Abs[f[c]] > 0.0001,
 c = (b f[a] - a f[b])/(f[a] - f[b]);
 If[f[c] * f[a] < 0, b = c];
 If[f[c] * f[b] < 0, a = c];
 AppendTo[sol, c];
]
```

In[ ]:=
```
N@c
N@f[c]
Length[sol]
N@sol
```

Out[ ]=
```
-0.567163
```

Out[ ]=
```
-0.0000308386
```

Out[ ]=
```
5
```

Out[ ]=
```
{-0.698162, -0.58148, -0.568735, -0.56732, -0.567163}
```

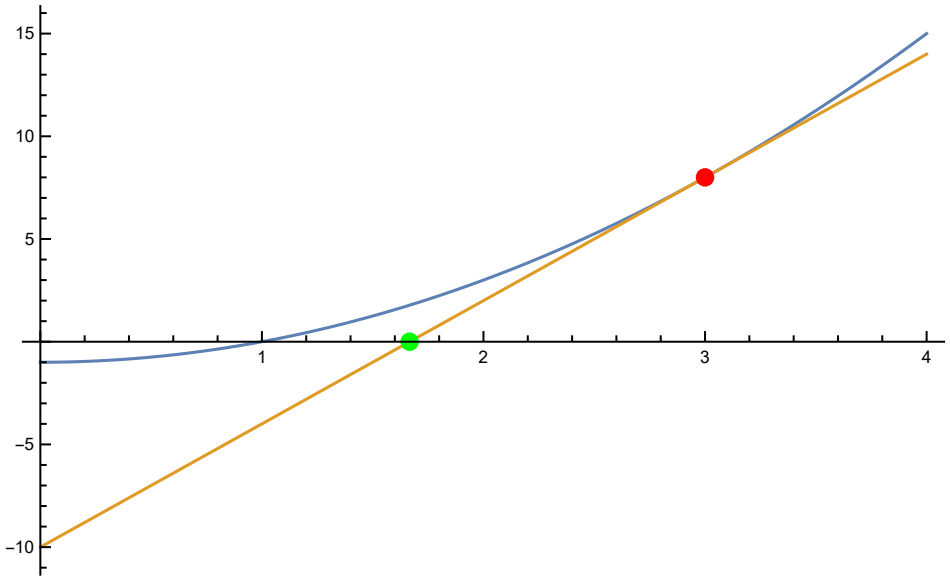
Metoda Newtona

- zgadnij początkowe miejsce zerowe x_0
- wyznacz styczną do funkcji f w punkcie x_0
- miejsce zerowe stycznej jest przybliżeniem miejsca zerowego funkcji f
  - $y = f'(x_0)\, x + f(x_0) - f'(x_0)\, x_0 \Rightarrow$
  - $0 = f'(x_0)\, x + f(x_0) - f'(x_0)\, x_0 \Rightarrow x = x_0 - \frac{f(x_0)}{f'(x_0)}$
  - $x_{n+1} = x_n - \frac{f[x_n]}{f'[x_n]}$

In[ ]:=
```
x0 = 3
Show[{
  Plot[{x^2 - 1, 2 x0 x + x0^2 - 1 - x0 * 2 * x0}, {x, 0, 4}],
  Graphics[{Red, PointSize[0.02], Point[{x0, x0^2 - 1}]}],
  Graphics[{Green, PointSize[0.02], Point[{x0 - (x0^2 - 1)/(2 x0), 0}]}]
 }]
```

Out[ ]=
```
3
```

Out[ ]=



Wracamy do naszej funkcji f(x) = Exp[x] + x

In[ ]:=
```
x0 = 0;
sol = {};
While[Abs[f[x0]] > 0.0001,
 x0 = x0 - (Exp[x0] + x0)/(Exp[x0] + 1);
 AppendTo[sol, x0];
]
```

In[•]:=
```
N@x0
N[Exp[x0] + x0]
Length[sol]
N@sol
```

Out[•]=
```
-0.567143
```

Out[•]=
$$1.9648 \times 10^{-7}$$

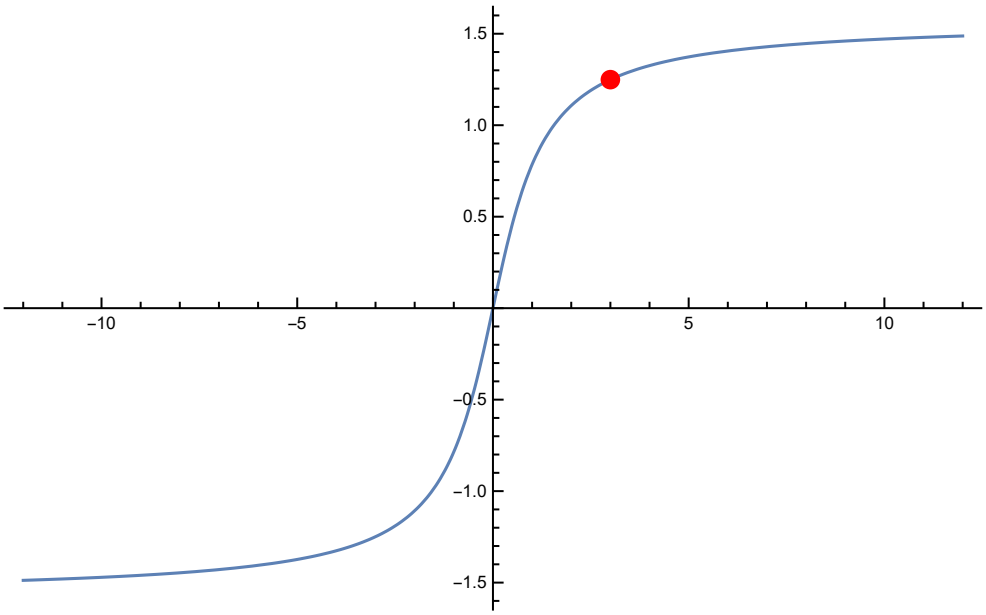
Out[•]=
```
3
```

Out[•]=
```
{-0.5, -0.566311, -0.567143}
```

- Szybka (zbieżność kwadratowa)
- Nie zawsze jest zbieżna (np. punkt początkowy blisko ekstremum lokalnego)
- Wymaga znajomości pochodnej

In[•]:=
```
x0 = 3;
Show[{
  Plot[ArcTan[x], {x, -12., 12.}],
  Graphics[{Red, PointSize[0.02], Point[{x0, ArcTan[x0]}]}]
 }]
```

Out[•]=



In[ ]:=

```
sol = {};
n = 0;
While[n < 10,
 x0 = x0 - ArcTan[x0]/(1/(1+x0^2));
 AppendTo[sol, x0];
 n++;
]
N@sol
```

Out[ ]=

$$\{-9.49046,\ 124.,\ -23\,905.9,\ 8.97653\times 10^{8},\ -1.26572\times 10^{18},\ 2.51648\times 10^{36},\ -9.94733\times 10^{72},\ 1.55429\times 10^{146},\ -3.79477\times 10^{292},\ 2.261988093610309\times 10^{585}\}$$

Fast Inverse Square Root

Problem: jak szybko obliczyć $\frac{1}{\sqrt{x}}$

```
// Quake III Arena
float Q_rsqrt (float number)
{
        long i;
        float x2, y;
        const float threehalfs = 1.5 F;
        x2 = number*0.5 F;
        y = number;
        i = *(long*) & y; // evil floating point bit level hacking
        i = 0x5f3759df - (i >> 1); // what the fuck?
        y = *(float*) & i;

        y = y*(threehalfs - (x2*y*y)); // 1 st iteration
    //y =  y*(threehalfs - (x2*y*y)); // 2 nd iteration, this can be removed return y;
    }
```

Out[•]=

https://www.youtube.com/watch?v=p8u_k2LIZyo

Różniczkowanie

Definicja :

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

Najprostsze rozwiązanie:

f'(x) = $\frac{f(x+h)-f(x)}{h}$, h - mała liczba

- Jaki jest błąd przybliżenia?
  Korzystamy z szeregu Taylor'a:

$$f(x+h) = f(x) + h f'(x) + \frac{1}{2} h^2 f''(\xi), \quad \xi \in [x, x+h]$$

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \frac{1}{2} h f''(\xi), \quad \xi \in [x, x+h]$$

Przykład: f(x) = Log[x], h = 0.001, $x_0 = 2$

In[•]:=
```
h = 0.001;
x0 = 2;
(Log[x0 + h] - Log[x0])/h
```

Out[•]=
```
0.499875
```

In[•]:=
```
D[Log[x], x] /. {x → x0}
```

Out[•]=
$$\frac{1}{2}$$

- Błąd: $\left|\frac{1}{2} h f''(x0)\right| < \left|\frac{1}{2} h f''(\xi)\right| < \left|\frac{1}{2} h f''(x0+h)\right|$

In[•]:=
```
Abs[1/2 h D[Log[x], x, x] /. {x → 2}]
Abs[1/2 h D[Log[x], x, x] /. {x → 2.001}]
```

Out[•]=
```
0.000125
```

Out[•]=
```
0.000124875
```

- Uwaga na numerykę!
  Przykład: f(x) = arctg(x) w $x_0 = \sqrt{2}$
  - f'(x) = $\frac{1}{1+x^2}$ => $f'\left(\sqrt{2}\right) = \frac{1}{3}$
- ```
  arctan(sqrt(2)) 0.955317 h = 0.0001 derivative 0.33319
  arctan(sqrt(2)) 0.955317 h = 1e-05 derivative 0.333786
  arctan(sqrt(2)) 0.955317 h = 1e-06 derivative 0.298023
  arctan(sqrt(2)) 0.955317 h = 1e-07 derivative 0.596046
  arctan(sqrt(2)) 0.955317 h = 1e-08 derivative 0
  ```

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

```
float f(float x) {
   return std::atan(x);
}

float diff(float x, float (*fptr) (float), float h = 0.001) {
   return (fptr(x + h) - fptr(x)) / h;

}

int main(){


   std::cout << " h = " << 0.0001 << " " << diff(sqrtf(2), f, 0.0001) << std::endl;
   std::cout << " h = " << 0.00001 << " derivative " << diff(sqrtf(2), f, 0.00001) << std::endl;

   std::cout << " h = " << 0.000001 << " derivative " << diff(sqrtf(2), f, 0.000001) << std::endl;

   std::cout << " h = " << 0.0000001 << " derivative " << diff(sqrtf(2), f, 0.0000001) << std::endl;

   std::cout << " h = " << 0.00000001 << " derivative " << diff(sqrtf(2), f, 0.00000001) << std::endl;

}
```

Metoda “central difference”

$$f'_{+}(x) = \frac{f(x+h) - f(x)}{h} + O(h)$$

$$f'_{-}(x) = \frac{f(x) - f(x-h)}{h} + O(h)$$

$$f'(x) \approx \frac{f'_{+}(x) + f'_{-}(x)}{2} = \frac{f(x+h) - f(x-h)}{2h}$$

Błąd:

$$f(x+h) = f(x) + h f'(x) + \frac{1}{2} h^2 f''(x) + \frac{1}{6} h^3 f'''(\xi_{+})$$

$$f(x-h) = f(x) - h f'(x) + \frac{1}{2} h^2 f''(x) - \frac{1}{6} h^3 f'''(\xi_{-})$$

$$f(x+h) - f(x-h) = f(x) + h f'(x) + \frac{1}{2} h^2 f''(x) + \frac{1}{6} h^3 f'''(\xi_{+}) - f(x) + h f'(x) - \frac{1}{2} h^2 f''(x) + \frac{1}{6} h^3 f'''(\xi_{-})$$

$$f(x+h) - f(x-h) = 2h f'(x) + \frac{1}{6} h^3 \left(f'''(\xi_{+}) + f'''(\xi_{-})\right)$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + \frac{1}{12} h^2 \left(f'''(\xi_{+}) + f'''(\xi_{-})\right)$$

Pochodne wyższego rzędu

- Szereg Tylor'a:

$$f(x+h) = f(x) + h f'(x) + \frac{1}{2} h^2 f''(x)$$

$$f''(x) = \frac{2}{h^2} (f(x+h) - f(x) - h f'(x))$$

$$f''(x) = \frac{2}{h^2} \left(f(x+h) - f(x) - h \frac{f(x+h) - f(x-h)}{2h} \right)$$

$$f''(x) = \frac{1}{h^2} (2 f(x+h) - 2 f(x) - f(x+h) + f(x-h))$$

$$f''(x) = \frac{1}{h^2} (f(x+h) - 2 f(x) + f(x-h)) + O\left(h^2\right)$$