O1: Materiały dydaktyczne do 20 lekcji matematyki i fizyki

Praca zbiorowa pod redakcją
dr hab. Marcina Kostura

Spis treści

ROZDZIAŁ 1

Wstęp

Książka ta jest dostępna w interaktywnej wersji online pod adresem:

- W wersji polskiej http://visual.icse.us.edu.pl/metodologia
- W wersji angielskiej http://visual.icse.us.edu.pl/methodology

Podczas realizacji projektu nauczyciele i uczniowie wytworzyli kilka tysięcy dokumentów - tzw. notatników w systemie SageMath. Dokumenty te znajdują się na dwóch dedykowanych szkołom serwerach i niektóre z nich zostały opublikowane pod adresami:

- https://sage01.icse.us.edu.pl/pub/
- https://sage03.icse.us.edu.pl/pub/

Rezultaty projektu zostały też podsumowane w filmie z realizacji projektu dostępnym na Youtube pod adresem:

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

ROZDZIAŁ 2

Matematyka

2.1 Granica i ciągłość funkcji w punkcie

2.1.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenie 2h zajęć lekcyjnych.

Został on opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w XXXIII LO M. Kopernika w Warszawie.

Opracowanie scenariusza i prowadzenie lekcji: Mirosław Malinowski.

2.1.2 Ciągi liczbowe - powtórzenie

1. Co to jest ciąg liczbowy?
2. Podaj definicję Cauchy'ego granicy ciągu liczbowego.
3. Korzystając z własności granic ciągów liczbowych oblicz następujące granice:

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right),\ \lim_{n\to\infty}\left(1-\frac{1}{n}\right),\ \lim_{n\to\infty}\frac{2n^2+n+1}{n^2-\frac{1}{2}n},\ \lim_{n\to\infty}\frac{4n^2-n-1}{2n^2+3n}$$

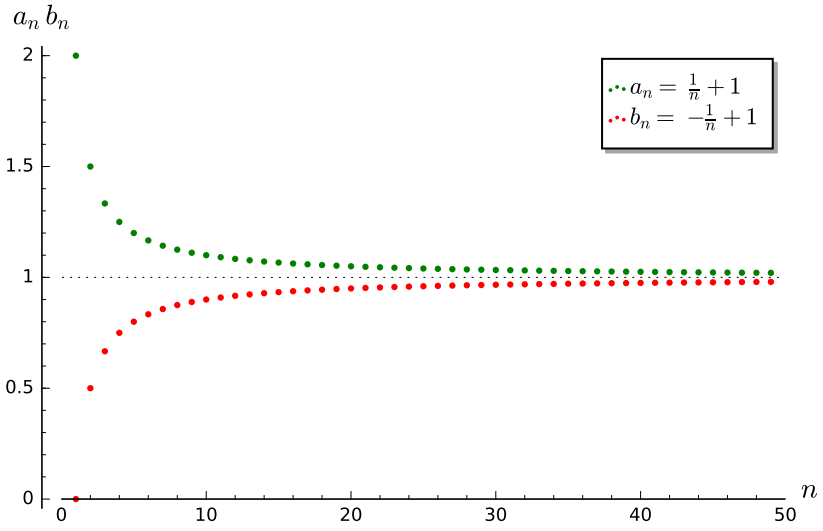
4. Przy pomocy pakietu SAGE oblicz granice ciągów podanych wyżej oraz narysuj ich wykresy.

Przykład:

Sprawdzimy zbieżność dwóch ciągów: $a_n = 1 + \frac{1}{n}$ oraz $b_n = 1 - \frac{1}{n}$

```
var('a_n')
a(n) = 1 + 1/n
b(n) = 1 - 1/n
c = limit(a(n), n=oo);
d = limit(b(n), n=oo)
plt = points([(m, a(m)) for m in srange (1, 50)],
 ↪ axes_labels=['$n$', '$a_n \, b_n$'], legend_label="$a_n = $
 ↪ $%s$" %latex(a(n)), color = 'green', figsize = (6, 4),
 ↪ gridlines = [None, [c]], fontsize=9)
plt += points([(m, b(m)) for m in srange (1, 50)],
 ↪ legend_label="$b_n = $ $%s$" %latex(b(n)),color = 'red',
 ↪ gridlines = [None, [d]])
plt
```

Wynik działania powyższej procedury:



Rys. 2.1: Dwa ciągi mające tę samą granicę.

2.1.3 Wprowadzenie definicji Heinego granicy funkcji w punkcie.

Funkcja $f(x)$ ma **granicę lewostronną** g w punkcie x_0 (używamy zapisu $\lim\limits_{x \to x_0^-} f(x) = g$), jeżeli dla każdego ciągu argumentów (x_n) o wyrazach należących do przedziału $x_n < x_0$), zbieżnego do x_0, ciąg wartości $(f(x_n))$ jest zbieżny do g .

$$\lim_{x \to x_0^-} f(x) = g \iff \forall (x_n) \in (a; x_0) : \lim_{n \to \infty} a_n = x_0 \implies \lim_{n \to \infty} f(a_n) = g$$

Funkcja $f(x)$ ma **granicę prawostronną** g w punkcie x_0 (używamy zapisu $\lim\limits_{x \to x_0^-} f(x) = g$), jeżeli dla każdego ciągu argumentów (x_n) o wyrazach należących do przedziału $x_n > x_0$), zbieżnego do x_0, ciąg wartości $(f(x_n))$ jest zbieżny do g .

$$\lim_{x \to x_0^+} f(x) = g \iff \forall (x_n) \in (x_0; a) : \lim_{n \to \infty} a_n = x_0 \implies \lim_{n \to \infty} f(a_n) = g$$

Informacja: Funkcja $f(x)$ ma granicę g w punkcie x_0, jeśli istnieją granice lewostronna i prawostronna tej funkcji w punkcie x_0 i $\lim\limits_{x\to x_0^-} f(x) = \lim\limits_{x\to x_0^+} f(x) = g$.

Przykład

Korzystając z definicji Heinego obliczymy granicę funkcji $f(x) = \frac{x}{x+1}$ w punkcie $x_0 = 1$.

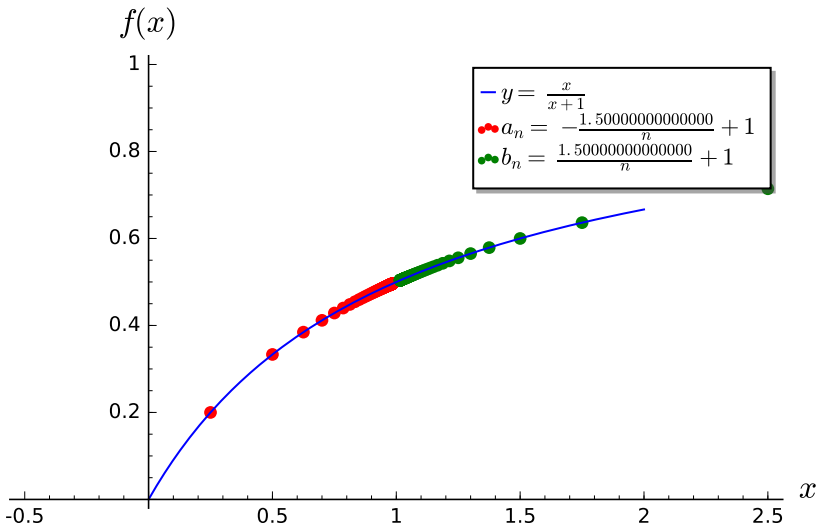
```
var('x0 x_0')
x0 = 1
f(x) = x/(x+1) # Badana funkcja
a(n) = x0 - 1.5/n # Ciąg lewostronnie zbieżny do x0
b(n) = x0 + 1.5/n # Ciąg prawostronnie zbieżny do x0
gl = limit(f(a(n)), n=oo) # Granica lewostronna ciągu wartości funkcji f(x)
gr = limit(f(b(n)), n=oo) # Granica prawostronna ciągu wartości funkcji f(x)
la = [a(k) for k in srange(1, 100)] # Wartości wyrazów ciągu zbieżnego lewostronnie do x0
lb = [b(k) for k in srange(1, 100)] # Wartości wyrazów ciągu zbieżnego prawostronnie do x0
plt = plot (f(x), (x, x0-1, x0+1), axes_labels=['$x$','$f(x)$'], legend_label="$y = $ $%s$"%latex(f(x)))
plt += points([(m, f(m)) for m in la], legend_label="$a_n = $ $%s$" %latex(a(n)), color='red', size=40, ymin = 0, ymax = 1, figsize=(6,4))
plt += point([(x, f(x)) for x in lb], color='green', size=40, legend_label="$b_n = $ $%s$" %latex(b(n)))
plt
```

Wynik działania powyższej procedury:

Zadanie 1

Wykorzystując podaną wyżej procedurę oblicz granice następujących funkcji:

1. $f(x) = \frac{x+2}{x-1}$ w punkcie $x_0 = 2$
2. $f(x) = \frac{x^2-3}{2x^2-1}$ w punkcie $x_0 = 1$
3. $f(x) = \frac{sinx}{x+1}$ w punkcie $x_0 = \frac{\pi}{2}$
4. $f(x) = \frac{x}{|x|}$ w punkcie $x_0 = 0$
5. $f(x) = \begin{cases} x^2 & \text{for } x \leq 0 \\ 2^x & \text{for } x > 0 \end{cases}$ w punkcie $x_0 = 0$



Rys. 2.2: Dwa ciągi mające tę samą granicę.

2.1.4 Definicja ciągłości funkcji w punkcie.

Funkcja f jest ciągła w punkcie x_0 wtedy, gdy dla każdego ciągu (x_n) o wyrazach należących do pewnego otoczenia liczby x_0, zbieżnego do x_0:

1. Istnieje granica $\lim\limits_{x \to x_0} f(x)$.
2. $\lim\limits_{x \to x_0} f(x) = f(x_0)$

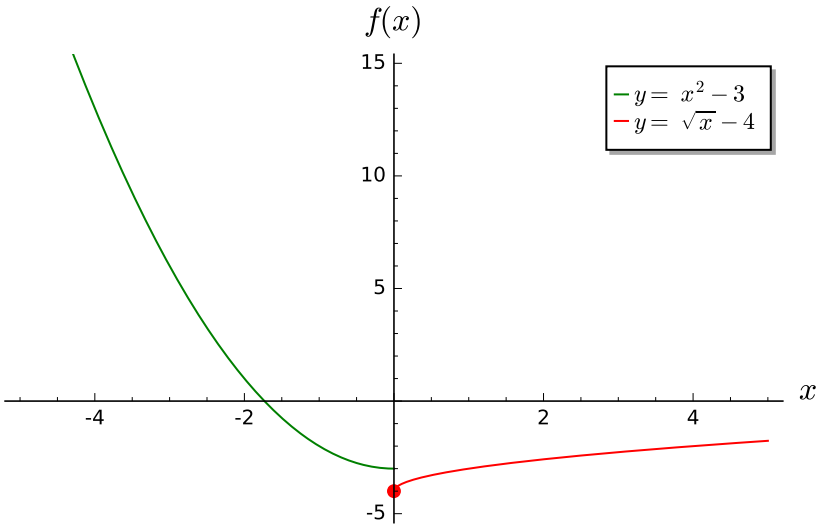
Przykład

Zbadaj, czy funkcja $f(x) = \begin{cases} x^2 - 4 & \text{dla } x < x_0 \\ \sqrt{x} - 4 & \text{dla } x \geq x_0 \end{cases}$ jest ciągła w punkcie $x_0 = 0$. Sprawdź, czy dana funkcja jest ciągła w innych punktach x_0?

```
var('x0')
x0 = 0
fl(x) = x^2 - 4
fr(x) = sqrt(x) - 4
def f(x):
    if x < x0: return fl(x)
    if x == x0: return fr(x)
    if x > x0: return fr(x)
a = limit(fl(x), x = x0, dir = 'left')
b = limit(fr(x), x = x0, dir = 'right')
if a == b == f(x0):
    print("Funkcja jest ciagla w punkcie ",x0)
else:
    print("Funkcja NIE JEST ciagla w punkcie ",x0)
plt = plot (fl, (x, x0-5, x0), axes_labels=['$x$','$f(x)$'], ymin =
 ↪  -5, ymax = 15, figsize = (6, 4), color = 'green',
 ↪  legend_label="$y =$ $%s$"%latex(fl(x)))
```

```
16 plt += plot (fr, (x, x0, x0+5), color = 'red', legend_label="$y=$
     ↪ $%s$"%latex(fr(x))) + point((x0, f(x0)), color = 'red', size =
     ↪ 48)
17 plt
```

Wynik działania powyższej procedury:



Rys. 2.3: Dwie funkcje.

Zadanie 2

Zbadaj czy następujące funkcje są ciągłe w podanych punktach.

1. $f(x) = |x+1| - x$ w punkcie $x_0 = -1$
2. $f(x) = \begin{cases} |x+3| - 1 & \text{for } x < x_0 \\ \cos x & \text{for } x \geq x_0 \end{cases}$ w punkcie $x_0 = 0$.
3. $f(x) = \begin{cases} \frac{x^2+x-6}{x-2} & \text{for } x < x_0 \\ 3x - 1 & \text{for } x \geq x_0 \end{cases}$ w punkcie $x_0 = 2$
4. $f(x) = \begin{cases} -2\sin x & \text{for } x < x_0 \\ \cos x & \text{for } x \geq x_0 \end{cases}$ w punkcie $x_0 = \pi$
5. $f(x) = \begin{cases} x \sin \frac{1}{x} & \text{for } x \neq x_0 \\ 0 & \text{for } x = x_0 \end{cases}$ w punkcie $x_0 = 1$

2.2 Od punktu do punktu

2.2.1 O scenariuszu

Prezentowany scenariusz jest wynikiem zajęć prowadzonych w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie. Przeznaczony jest do realizacji na dwóch godzinach lekcyjnych (po jednej na każdą z części), przy czym - co pokazały m.in. szkolne testowania na grupach uczniów oraz warszaty prowadzone dla grup nauczycieli - bardzo istotna jest możliwość przedyskutowania efektów zmian przedstawionych w części drugiej kodów. Struktura materiału jest swoistym dialogiem z uczestnikami zajęć, dopuszczamy więc podejście typu nieformalnego - najważniejsza jest dla nas bowiem możliwość eksperymentowania.

Lekcje prowadził Krzysztof Oleś.

2.2.2 Wstęp

W roku 2005 wydawnictwo TIKKUN zaprezentowało Polakom książkę, która w roku 1976 zdobyła światowy rozgłos (ze względów politycznych – w Polsce była rodzajem podziemnej klasyki dla matematyków). Piszemy tutaj o „Dowodach i refutacjach" Imre Lakatosa, w których pokazano, istotność powątpiewania i stawiania hipotez. Wspomniany rok 1976 pojawia się także jeszcze w innym kontekście. Otóż właśnie w tym roku na łamach New York Timesa pojawiła się informacja o udowodnieniu (?) twierdzenia o czterech barwach – wykorzystano programy komputerowe i ostateczne wyniki pochodziły z zaprogramowanych obliczeń. Można oczywiście dyskutować nad tego typu dowodem. Wydaje się jednak, że nad potrzebą wykorzystywania komputera do stawiania hipotez dyskutować nie trzeba. I to chcemy naszym wychowankom pokazać, łącząc matematykę z całkowicie dla nich naturalnym środowiskiem komputerowym. **Powątpiewanie może być twórcze**, a zabawa (!) może wywoływać uśmiech odkrywania.

Przedstawiamy ponadto dodatek, który może być punktem wyjścia do rozważań związanych z próbą definiowania wymiaru obiektów fraktalnych.

2.2.3 Scenariusz zajęć

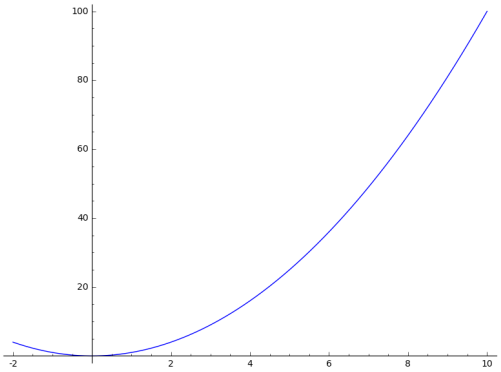
Część pierwsza

Pomyślmy przez chwilę o wykresie funkcji, np. $f(x) = \log_x \left|4 \sin\left(\frac{\pi}{2} - 3x\right) - 6\right|$. To oczywiście ciągła linia, którą wyobrażamy sobie bez jakiegokolwiek problemu. Czyżby?

Stosując w naszych matematycznych podróżach SAGE'a, użyć możemy m.in.:

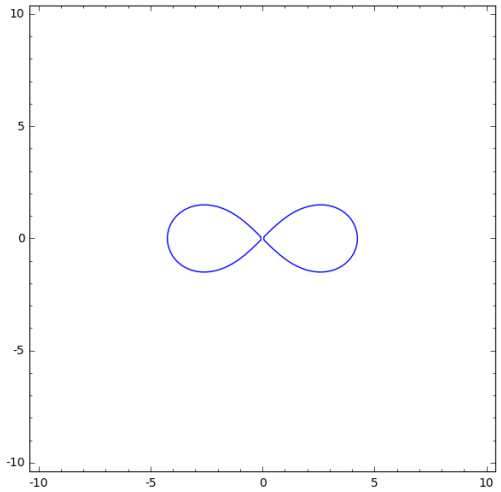
1. wzoru konkretnej funkcji

```
a=-2 #zmień to
plot(x^2, (x, a, 10))
```



2. równania nazywanego „uwikłanym”

```
var('y')
a=3 #zmień to
implicit_plot(((x^2)+(y^2))^2==2*(a^2)*((x^2)-(y^2)),(x,-10,10),
              (y,-10,10))
```



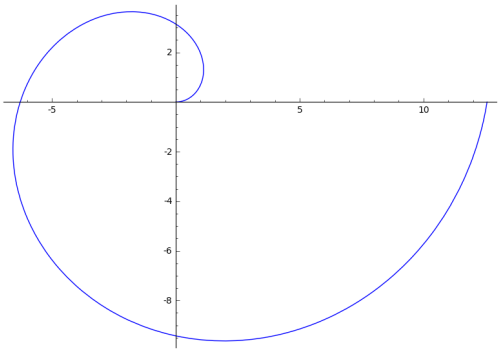
3. postaci biegunowej

```
a=2 #zmień to
polar_plot(a*x, (x, 0, 2*pi))
```

Czasem dostrzegamy pewne podobieństwa w wymienionych tu powyżej podejściach: w każdym z przypadków na ekranie komputera pojawia się zestaw punktów - jest on jednak ukryty w mniej lub bardziej skomplikowanej formule.

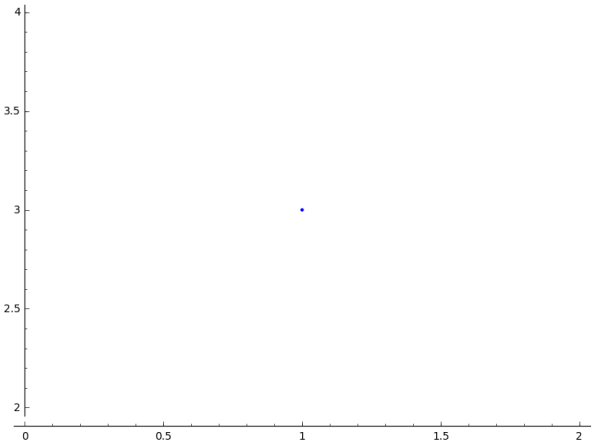
A może powinniśmy użyć najprostszej metody: od pukntu do punktu? Korzystając z rekurencji?

Pomyślmy o punkcie umieszczonym w układzie współrzędnych - da nam to możliwość połączenia geometrii z rozważaniami numerycznymi. Ważne będą dla nas możliwości eksperymentowania i komputerowych zabaw - prezentowane przykłady nie będą natomiast programistycznie skomplikowane.



Rozpocznijmy więc od umieszczenia na ekranie jednego punktu.

```
fig=point((1,3))
fig
```



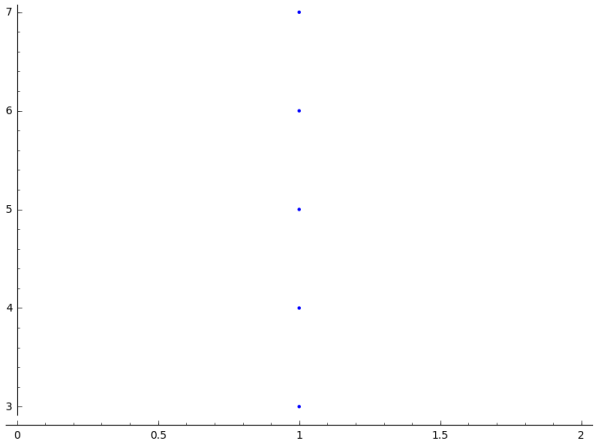
Właściwie to nic takiego - umieśćmy więc punktów pięć...

```
fig=point((1,3),(1,4),(1,5),(1,6),(1,5))
fig
```

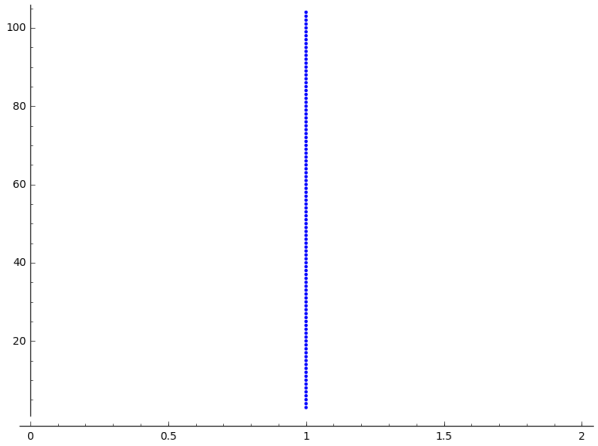
Po tym małym zaplanowanym błędzie punkty próbujemy dodać.

```
fig=point((1,3))+point((1,4))+point((1,5))+point((1,6))+point((1,7))
fig
```

Zauważmy, że nawet przy użyciu ctr+c+ctrl+v zabiera to sporo czasu i aż ciężko jest nam myśleć o umieszczaniu na ekranie takim sposobem stu punktów - tym bardziej w sytuacji, w której możemy znaleźć pewną **regularność** w opisie ich drugich współrzędnych. Zatem: użyjmy jej.




```
fig=point((1,3))
for i in range(4,105):
    fig=fig+point((1,i))
fig
```

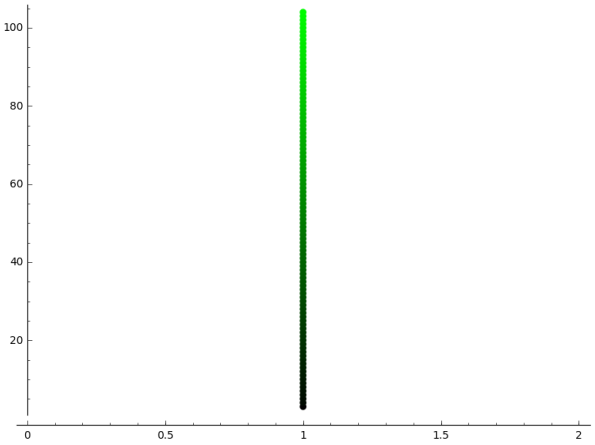


Spróbujmy zmienić też rozmiar puktów oraz ich odcień.

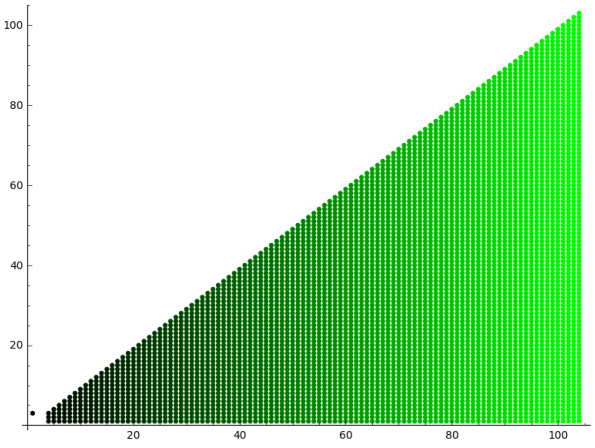
```
s=40 #zmień rozmiar
fig=point((1,3),rgbcolor=(0,0,0),size=s) #co oznacza (0,0,0)?
for n in range(4,105):
    fig=fig+point((1,n),rgbcolor=(0,n/105,0),size=s)
fig
```

Nie zapominajmy o możliwości umieszczenia pętli w pętli.

```
a=1
b=3
c=105
d=20
fig=point((a,b),rgbcolor=(0,0,0),size=d)
```




```
for n in range(4,c):
    for k in range(1,n):
        fig=fig+point((n,k),rgbcolor=(0,n/c,0),size=d)
fig
```



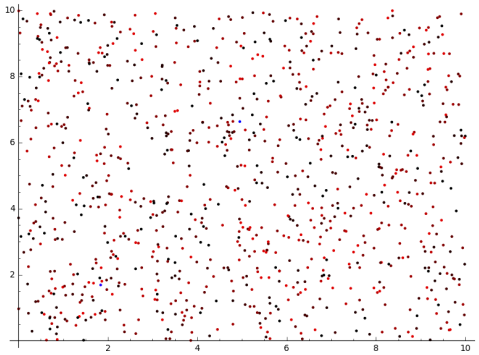
Patrząc na uzyskany efet, zauważamy pewien problem w „lewym” wierzchołku trójkąta - usuńmy go, poprawnie manipulując liczbami.

Część druga

Przejdźmy do losowania.

```
n=1001 #zmień to
a=10*random() #dlaczego używamy mnożenia?
b=10*random()
fig=point((a,b))
for k in range(1,n):
    a=10*random()
    b=10*random()
```

```
    fig=fig+point((a,b),color=((1/8)*k,2*k,k)) #zmień sposób
     ↪ kolorowania
fig
```



W powyższym przykładzie zauważamy specyficzny rodzaj chaosu... Czy możemy jednak punkty bardziej kontrolować?

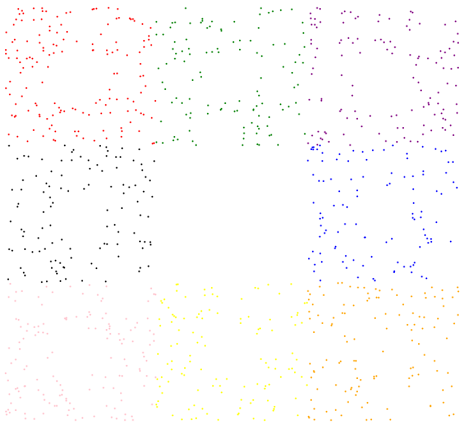
Wyobraźmy sobie sytuację, w którym określony punkt początkowy (a, b) przekształcany jest w wybranym losowo jednym z ośmiu przekształceń. Każde z nich składa się z dwóch części: liniowej operacji na pierwszej współrzędnej (trzy liczby a_i, b_i, c_i) oraz liniowej operacji na drugiej współrzędnej (trzy liczby d_i, e_i, f_i). Po przekształceniu otrzymujemy nowy punkt (a, b) który przetwarzamy analogicznie - oczywiście nie poprzestajemy na dwóch punktach, komputer dokona setek powtórzeń.

Spójrzmy uważnie na kod źródłowy (szczególnie na linie zawierające #).

```
a=0 #pierwsza współrzędna punktu startowego
b=0 #druga współrzędna punktu startowego
d=1001 #liczba powtórzeń i długie listy czynników poniżej...
a1=0.333;b1=0;c1=-0.333;d1=0;e1=0.333;f1=0.333
a2=0.333;b2=0;c2=0;d2=0;e2=0.333;f2=0.333
a3=0.333;b3=0;c3=0.333;d3=0;e3=0.333;f3=0.333
a4=0.333;b4=0;c4=0.333;d4=0;e4=0.333;f4=0
a5=0.333;b5=0;c5=0.333;d5=0;e5=0.333;f5=-0.333
a6=0.333;b6=0;c6=0;d6=0;e6=0.333;f6=-0.333
a7=0.333;b7=0;c7=-0.333;d7=0;e7=0.333;f7=-0.333
a8=0.333;b8=0;c8=-0.333;d8=0;e8=0.333;f8=0 #i wreszcie koniec listy
r=point((a,b),axes=False, frame=False,size=0)
for c in range(1,d):
  n=randint(1,8) #wybór jednego z ośmiu przekształceń
  if n==1:
      a=(a1*a)+(b1*b)+c1
      b=(d1*a)+(e1*b)+f1
      r=r+point((a,b),axes=False, frame=False,size=5,color='red')
  if n==2:
      a=(a2*a)+(b2*b)+c2
      b=(d2*a)+(e2*b)+f2
      r=r+point((a,b),axes=False, frame=False,size=5,color='green')
```

```
  if n==3:
      a=(a3*a)+(b3*b)+c3
      b=(d3*a)+(e3*b)+f3
      r=r+point((a,b),axes=False, frame=False,size=5,color='purple')
  if n==4:
      a=(a4*a)+(b4*b)+c4
      b=(d4*a)+(e4*b)+f4
      r=r+point((a,b),axes=False, frame=False,size=5,color='blue')
  if n==5:
      a=(a5*a)+(b5*b)+c5
      b=(d5*a)+(e5*b)+f5
      r=r+point((a,b),axes=False, frame=False,size=5,color='orange')
  if n==6:
      a=(a6*a)+(b6*b)+c6
      b=(d6*a)+(e6*b)+f6
      r=r+point((a,b),axes=False, frame=False,size=5,color='yellow')
  if n==7:
      a=(a7*a)+(b7*b)+c7
      b=(d7*a)+(e7*b)+f7
      r=r+point((a,b),axes=False, frame=False,size=5,color='pink')
  if n==8:
      a=(a8*a)+(b8*b)+c8
      b=(d8*a)+(e8*b)+f8
      r=r+point((a,b),axes=False, frame=False,size=5,color='black')
show (r, figsize=(8.75,8))
```

Przy stu powtórzeniach otrzymana figura wydaje się chaotyczna, dlatego też wykonaliśmy większą liczbę powtórzeń (proponujemy dalsze zwiększanie liczby d).



Czy układ punktów nie zaczyna nam czegoś przypominać?

Gdzieś w głowie powinien pojawić się nam dywan Sierpińskiego.

Poeksperymentujmy z tym tworem i spróbujmy odpowiedzieć na niełatwe pytania:

- Czy generowana figura zależy od doboru punktu startowego?
- Co stanie się, gdy zmieniać będziemy liczby $a_i, b_i, c_i, d_i, e_i, f_i$?

- Co stanie się, jeśli np. jedno z ośmiu przekształceń „wyłączymy” i czy na pewno umiemy to w kodzie źródłowym zrobić?
- Dlaczego dywan pokolorowany jest w taki a nie inny sposób?

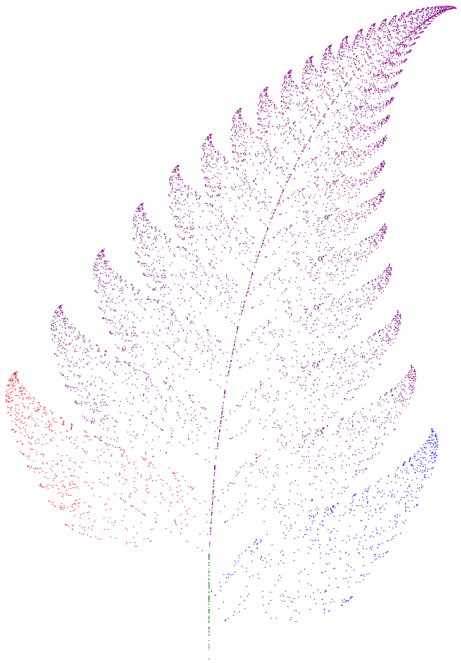
Wydaje się, że odpowiedzi na te pytania (oparte na pewnych założeniach) będą zaskakujące, ale komputer się nie męczy - stawiajmy więc hipotezy...

Być może ciekawszym od dywanu będzie znany (prawie wszystkim) liść.

```
c=10001 #liczba powtórzeń
a=0 #pierwsza współrzędna punktu startowego
b=0 #druga współrzędna punktu startowego
p=7 #szerokość obrazu
q=10 #wysokość obrazu
r=point((a,b),size=1, axes=false, frame=false) #zmieniając 'false' na 'true' możemy uzyskać osie i ramkę
for m in range (0,c):
  n=random()
  if n<0.01: #a co to takiego?!
      o=0.0*a + 0.0*b + 0.0
      b=0.0*a + 0.16*b + 0.0
      a=o
      r=r+point((a,b), axes=false, frame=false, color='green', size=1)
  elif n<0.08: #dlaczego elif?
      o=0.2*a - 0.26*b + 0.0
      b=0.23*a + 0.22*b + 1.6
      a=o
      r=r+point((a,b), axes=false, frame=false,color='red', size=1)
  elif n<0.15:
      o=-0.15*a + 0.28*b + 0.0
      b=0.26*a + 0.24*b + 0.44
      a=o
      r=r+point((a,b), axes=false, frame=false,color='blue',size=1)
  elif n<1:
      o=0.85*a + 0.04*b + 0.0
      b=-0.04*a + 0.85*b + 1.6
      a=o
      r=r+point((a,b), axes=false, frame=false,color='purple', size=1)
show(r, figsize=(p,q))
```

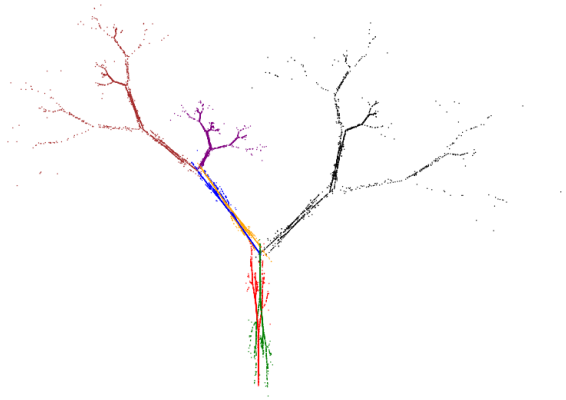
Przypuszczalnie dywan i liść **pociągną** nas do dalszego eksperymentu, w którym spróbujemy zapisać współczynniki w tabelach (poprzednie sposoby zapisu zachęcają do takiego rozwiązania).

```
a1=[0.05,0,-0.06,0,0.4,-0.47]
a2=[-0.05,0,-0.06,0,-0.4,-0.47]
a3=[0.03,-0.14,-0.16,0,0.26,-0.01]
a4=[-0.03,0.14,-0.16,0,-0.26,-0.01]
```



```
a5=[0.56,0.44,0.3,-0.37,0.51,0.15]
a6=[0.19,0.07,-0.2,-0.1,0.15,0.28]
a7=[-0.33,-0.34,-0.54,-0.33,0.34,0.39]
c=1
d=1
t=10001
r=point((c,d),axes=False, frame=False,size=0.1,)
for u in range(1,t):
  n=randint(1,7)
  if n==1:
      i=(a1[0]*c)+(a1[1]*d)+a1[2]
      o=(a1[3]*c)+(a1[4]*d)+a1[5]
      c=i
      d=o
      r=r+point((c,d),axes=False, frame=False,size=1,color='red')
  if n==2:
      i=(a2[0]*c)+(a2[1]*d)+a2[2]
      o=(a2[3]*c)+(a2[4]*d)+a2[5]
      c=i
      d=o
      r=r+point((c,d),axes=False, frame=False,size=1,color='green')
  if n==3:
      i=(a3[0]*c)+(a3[1]*d)+a3[2]
      o=(a3[3]*c)+(a3[4]*d)+a3[5]
      c=i
      d=o
      r=r+point((c,d),axes=False, frame=False,size=1,color='blue')
  if n==4:
      i=(a4[0]*c)+(a4[1]*d)+a4[2]
      o=(a4[3]*c)+(a4[4]*d)+a4[5]
```

```
        c=i
        d=o
        r=r+point((c,d),axes=False, frame=False,size=1,color='orange')
    if n==5:
        i=(a5[0]*c)+(a5[1]*d)+a5[2]
        o=(a5[3]*c)+(a5[4]*d)+a5[5]
        c=i
        d=o
        r=r+point((c,d),axes=False, frame=False,size=1,color='black')
    if n==6:
        i=(a6[0]*c)+(a6[1]*d)+a6[2]
        o=(a6[3]*c)+(a6[4]*d)+a6[5]
        c=i
        d=o
        r=r+point((c,d),axes=False, frame=False,size=1,color='purple')
    if n==7:
        i=(a7[0]*c)+(a7[1]*d)+a7[2]
        o=(a7[3]*c)+(a7[4]*d)+a7[5]
        c=i
        d=o
        r=r+point((c,d),axes=False, frame=False,size=1,color='brown')
r
```



A można by jakoś te nużące zapisy skrócić? Może np. tak:

```
A = [[0.05,0,-0.06,0,0.4,-0.47],
  [-0.05,0,-0.06,0,-0.4,-0.47],
  [0.03,-0.14,-0.16,0,0.26,-0.01],
  [-0.03,0.14,-0.16,0,-0.26,-0.01],
  [0.56,0.44,0.3,-0.37,0.51,0.15],
  [0.19,0.07,-0.2,-0.1,0.15,0.28],
  [-0.33,-0.34,-0.54,-0.33,0.34,0.39]]
c = 1
d = 1
t = 100001
pkt_lst = []
```

```
for u in range(1,t):
  n = randint(0,6)
  a = A[n]
  i = (a[0]*c)+(a[1]*d)+a[2]
  o = (a[3]*c)+(a[4]*d)+a[5]
  c = i
  d = o
  pkt_lst.append( (c,d) )

point(pkt_lst,axes=False, frame=False,size=1,color='red')
```

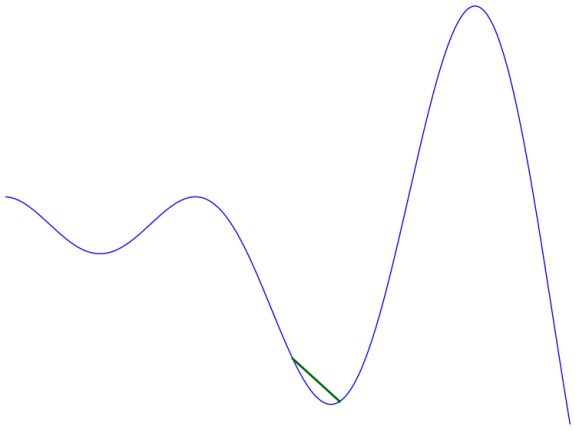
Eksperymentujmy!

2.2.4 Dodatek

Wróćmy do dywanu Sierpińskiego. Czy jest to raczej rodzaj linii lub też coś w rodzaju połączonych kwadratów? Czy nieustanne (w naszej głowie) powtarzanie iteracji zbliża nas do bardziej normalnych kwadratów? Co oznacza „bardziej”?

Popatrzmy na niebieską linię poniżej - chciejmy ją zmierzyć zieloną linijką.

```
plot(x * sin(x), (x, -2, 10), axes=false)+line([(4.1,4.1*sin(4.1)),
  ↪  (5.1,5.1*sin(5.1))], color='darkgreen', thickness=2)
```



Oszacujmy długość niebieskiej linii. Niech $M(\epsilon)$ oznacza długość mierzonej krzywej linijką długości ϵ oraz $L(\epsilon)$ przyłożeń tejże linijki. Zauważmy, że im mniejsze ϵ tym szacowanie dokładniejsze. Zauważmy, że $M(\epsilon) \approx \epsilon \cdot L(\epsilon)$ i

$$L(\epsilon) \sim \frac{1}{\epsilon}$$

(krótsza linijka oznacza większą liczbę jej przyłożeń). Jeśli powtórzymy nasze rozumowanie, rozważając pole zamiast długości, to „linijka” będzie kwadratem o boku długości ϵ i

$$L(\epsilon) \sim \frac{1}{\epsilon^2}.$$

A co z objętością? Być może „linijka" będzie sześcianem i

$$L(\epsilon) \sim \frac{1}{\epsilon^3}.$$

Zatem

$$L(\epsilon) \sim \frac{1}{\epsilon^d}$$

i $d = 1$ (jeśli próbujemy oszacować długość), $d = 2$ (jeśli próbujemy oszacować pole), $d = 3$ (jeśli próbujemy oszacować objętość).

Spróbujmy dobrać się do d.

$$L(\epsilon) \approx \left(\frac{1}{\epsilon}\right)^d,$$

$$\log L(\epsilon) \approx \log \left(\frac{1}{\epsilon}\right)^d = d \log \left(\frac{1}{\epsilon}\right),$$

i

$$d \approx \frac{\log L(\epsilon)}{\log \frac{1}{\epsilon}},$$

może zapiszmy to tak:

$$d = \lim_{\epsilon \to 0} \frac{\log L(\epsilon)}{\log \frac{1}{\epsilon}}?$$

(czy jest jakiś błąd w zamiennym użyciu znaków: $\sim, \approx, =$?).

Wygląda to dość dramatycznie - zobaczmy jak zadziała w przypadku dywanu Sierpińskiego. Figurę tą możemy (**na pewno**?!) pokryć 1 kwadratem o boku długości 1, 8 kwadratami o boku dłogości $\frac{1}{3}$, 64 kwadratami o boku długości $\frac{1}{9}$,..., 8^n kwadratami o boku długości $\left(\frac{1}{3}\right)^n$ i

$$d = \lim_{n \to \infty} \frac{\log 8^n}{\log 3^n} = \frac{\log 8}{\log 3} \approx 1.893.$$

Dywan Sierpińskiego jest czymś między linią a powierzchnią: być może - przy okazji - zbliżyliśmy się trochę do pojęcia wymiaru... Rozwinięcie powyższej idei można znaleźć w scenariuszu *Wymiar fraktalny*.

2.2.5 Podsumowanie

W powyższym tekście trzy wyrażenia zapisaliśmy czcionką pogrubioną - chcielibyśmy do nich powrócić.

Regularność

Działania wspomagane Sage'em mogą uczniom pomóc w badaniu rekurencji (pewnej regularności powtarzanej wielokrotnie, dzięki komputerom bardzo wiele razy).

Pociągną

Warto naszym zdaniem pociągnąć (angielskie „to attract") uczniów do koncepcji atraktora: fraktale to często atraktory - komputer może przecież pomóc w małych matematycznych odkryciach.

Na pewno

Należy podkreślić, że powyższe rozważania dotyczące wymiaru są tylko zasygnalizowaniem problemu - ale mogą budować uczniowską intuicję (która nie powinna być natychmiast zafrapowana problemem istnienia $\lim_{\epsilon \to 0} \ldots$).

A przy okazji - na samym końcu - warto postawić pytanie: gdzie jest granica między intuicyjną zabawą ucznia a poważnym rozumowaniem matematycznym?

2.3 Wymiar fraktalny

2.3.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenia jednej godziny zajęć lekcyjnych i jest uzupełnieniem do scenariusza *Od punktu do punktu*.

Został on opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Krzysztofa Olesia.

2.3.2 Wstęp - generacja samopodobnych objektów

W dalszej analizie potrzebujemy algorytmów do generacji samopodobnych obiektów geometrycznych. Rozważymy trzy obiekty, chociaż przedstawione rozwiązania mogą służyć badaniu dowolnych danych, zarówno pomiarowych jak i sztucznie wygenerowanych. Ponieważ analiza poniższych algorytmów nie jest dla nas celem, to poprzestaniemy na wypróbowaniu ich działania.

```
import numpy as np

def kochenize(a,b):
    HFACTOR = (3**0.5)/6
    dx = b[0] - a[0]
    dy = b[1] - a[1]
    mid = ( (a[0]+b[0])/2, (a[1]+b[1])/2 )
    p1 = ( a[0]+dx/3, a[1]+dy/3 )
    p3 = ( b[0]-dx/3, b[1]-dy/3 )
    p2 = ( mid[0]-dy*HFACTOR, mid[1]+dx*HFACTOR )
    return p1, p2, p3

def koch(steps, width=1):
    arraysize = 4**steps + 1
    points = [(0.0,0.0)]*arraysize
    points[0] = (-width/2., 0.0)
    points[-1] = (width/2., 0.0)
    stepwidth = arraysize - 1
    for n in xrange(steps):
        segment = (arraysize-1)/stepwidth
        for s in xrange(segment):
            st = s*stepwidth
            a = (points[st][0], points[st][1])
            b = (points[st+stepwidth][0], points[st+stepwidth][1])
            index1 = st + (stepwidth)/4
            index2 = st + (stepwidth)/2
            index3 = st + ((stepwidth)/4)*3
            result = kochenize(a,b)
            points[index1], points[index2], points[index3] = result
```

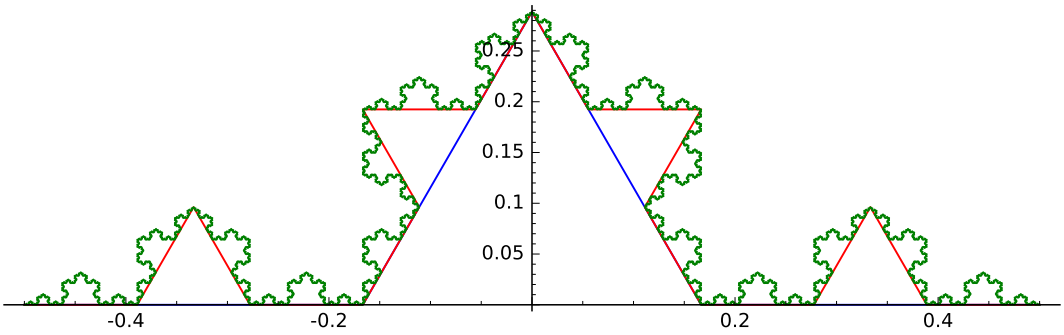
```
        stepwidth /= 4
    return np.array(points)
def hilbert(n=6):
    L=[]
    M = matrix([[2,3],[1,4]])
    for i in range(1,n):
        M1=M.antitranspose()
        M2=M+4^i*ones_matrix(2^i)
        M3=M+2*4^i*ones_matrix(2^i)
        M4=M.transpose()+3*4^i*ones_matrix(2^i)
        P=block_matrix([[M2,M3],[M1,M4]])
        M=P
        L.append(P)

    A = M.numpy()
    nx,ny = A.shape
    Z = np.argsort(A.flatten())
    X,Y = Z%nx,Z/ny
    X,Y = X/(2^n-1.),Y/(2^n-1.)
    xy = np.vstack([X,Y]).T
    return xy
```

Wypróbujmy działanie tych funkcji. Na początek możemy narysować krzywą Kocha. Algorytm jej tworzenia jest dwuetapowy. Najpierw należy odcinek podzielić na trzy równe części. Następnie środkową zastąpić dwoma bokami trójkąta równobocznego. Powtarzając wiele razy wspomnianą operację, otrzymujemy nieskończenie długą linię mieszczącą się w obszarze o skończonym polu. Narysujmy pierwszą, drugą i szóstą iterację.

```
line(koch(1),aspect_ratio=1) + line(koch(2),color='red') + \
  line(koch(3),color='green')
```

Wynikiem działania powyższego kodu jest wykres Rys. 2.4.



Rys. 2.4: Przybliżenie krzywej Kocha.

Widzimy, że każde kolejne zwiększenie liczby iteracji (argumentu) powoduje skomplikowanie wykresu.

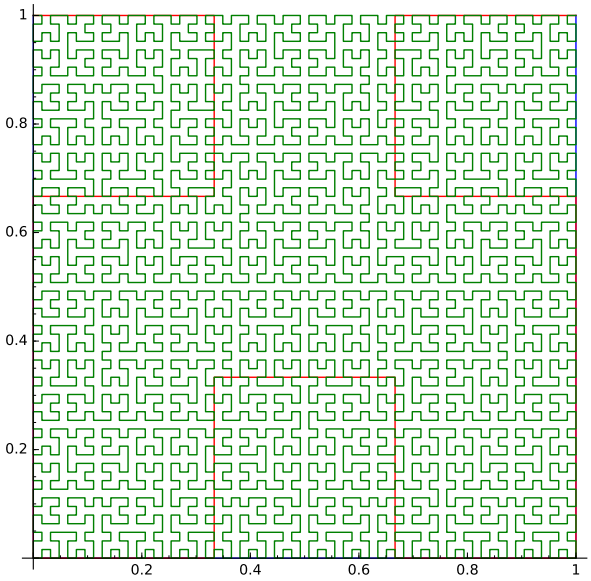
Należy pamiętać, że ilość danych rośnie potęgowo z liczbą pokoleń, więc badzo łatwo przekroczyć zasoby komputera, na którym wykonujemy powyższy algorytm. Warto sprawdzić ile czasu zajmuje wyenerowanie danej krzywej.

```
%%time
data_koch = koch(9)
```

Podobnie z krzywą Hilberta - narysujmy pierwszą, drugą i szóstą iteracje.

```
line(hilbert(1),aspect_ratio=1) + line(hilbert(2),color='red') + \
  line(hilbert(6),color='green')
```

Wynikiem działania powyższego kodu jest wykres Rys. 2.5.



Rys. 2.5: Przybliżenie krzywej Hilberta.

```
%%time
data_hilbert = hilbert(9)
```

Na samym końcu wygenerujemy dane będące punktami na okręgu - czyli obiekcie wymiarze równym jeden.

```
phi = np.linspace(0,2*3.14,1000000)
data_circle =  np.vstack([0.3*np.cos(phi),0.3*np.sin(phi)]).T
```

2.3.3 Wymiar pudełkowy (Mińkowskiego)

Wymiar Minkowskiego można określić rozważając to, jak długość zależy od skali, tzn. „linijki", którą przeprowadzamy pomiar:

$$l(\epsilon) \sim e^{(1-d)},$$

gdzie d jest wymiarem obiektu.

Ponieważ znamy procedurę tworzenia niektórych obiektów, to możemy dla nich otrzymać dokładne wyniki. I tak dla:

- Krzywej Kocha:

$$d = \frac{\log(4)}{\log(3)} \simeq 1.2618$$

- okręgu:

$$d = 1$$

- Krzywej Hilberta:

$$d = 2.$$

Spróbujmy obliczyć wymiar obiektu, zapominając skąd mamy dane: weźmy je (np. 1 milion punktów leżących na krzywej Kocha) i zmierzmy długość łamanej. Następnie wyrzućmy co drugi punkt i powtórzmy pomiar. Taką procedurę możemy zastosować dla dowolnego obiektu będącego łamaną.

Operacje na tablicach:

Pozornie skomplikowana linijka w Python/Sage `np.mean(np.sqrt(np.sum(np.diff(l,axis=0)**2,axis=1)))` jest równoznaczna z matematycznym zapisem:

$$\frac{1}{N}\sum_{i=0}^{N-1}\sqrt{\sum_{j=1}^{2}(l_{i,j}-l_{i-1,j})^2},$$

a `np.sum(np.sqrt(np.sum(np.diff(l,axis=0)**2,axis=1)))`

oznacza:

$$\sum_{i=0}^{N-1}\sqrt{\sum_{j=1}^{2}(l_{i,j}-l_{i-1,j})^2}.$$

Informacja: W poniższej komórce można „odkomentować" inne przypadki, powtórzyć obliczenia i przeanalizować wyniki.

```
# l = data_hilbert
# l = data_circle
l = data_koch
```

```
scal=[]
for i in range(100):
    epsilon = np.mean(np.sqrt(np.sum(np.diff(l,axis=0)**2,axis=1)))
    length = np.sum(np.sqrt(np.sum(np.diff(l,axis=0)**2,axis=1)))
    scal.append( (epsilon,length) )

    l = l[0::2,:]
    if l.shape[0]<=2:
        break
```

Wystarczy dopasować tak otrzymane dane do modelu $l(\epsilon) \sim e^{(1-d)}$ i powinniśmy otrzymać przybliżenie wymiaru.

```
var('a,d,x')
model(x)=a*x^(1-d)
scal_sel = [(eps,length) for eps,length in scal if eps>0.0009 and
 ↪  eps<0.01]
fit = find_fit(scal_sel,model)
fit
```

Narysujmy dopasowanie.

```
plot_loglog(model(x).subs(fit),(x,0.001,1),title=r"$l(\epsilon)= b
 ↪  \epsilon^{(1-%0.4f)}}$"%(d.subs(fit))) +\
 point(scal,size=30) + point(scal_sel,size=30,color='red')
```

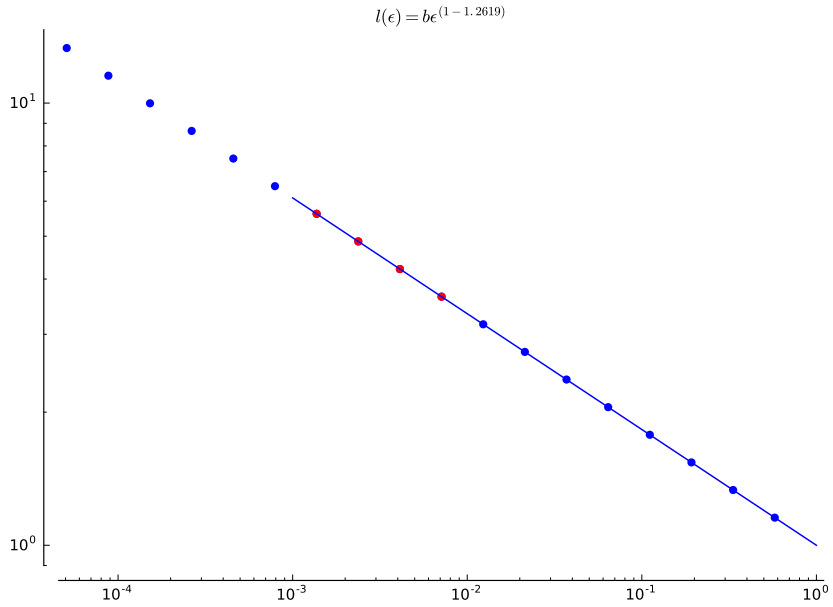
Wynikiem działania powyższego kodu jest wykres Rys. 2.6.

Otrzymujemy liczbę zbliżoną do wyniku dokładnego. Zaletą tego postępowania jest jego niezależność od źródła danych.

2.3.4 Box counting

Wyobraźmy sobie, ze robimy zdjęcie naszego obiektu aparatem cyfrowym o pewnej rozdzielczości i zliczamy tylko te piksele, na których widać obiekt. Jak zmienia się liczba pikseli, na których znajduje się nasz obiekt z rozmiarem piksela aparatu? Taka procedura nazywa się box-counting.

Wykorzystujemy histogram wbudowany w numpy: `np.histogramdd`



Rys. 2.6: Wykres w skali logarytmicznej (tzw. log-log) długości łamanej od średniej dlugości segmentu.

Piksel - lub voxel (3d) - może być n-wymiarowym pudełkiem, jednak takim, by mógł on pokrywać cały obiekt: np. dla krzywej Kocha musimy rozważyć co najmniej piksele 2d.

Zaletą box-countingu jest to, że wystarczy dysponować punktami należącymi do obiektu w dowolnej kolejności, np. takimi jak te generowane w grze w chaos.

```
# xy = data_circle
# xy = data_hilbert
xy = data_koch
```

```
scal = []
# np.logspace(1.2,3.3,10)
for bs in [15, 27, 46, 79, 135, 232, 398, 681, 1165, 1995]:
    H = np.histogramdd(xy,bins=[np.linspace(-1.,1.0,int(bs))]*2 )[0]
    scal.append( (2*bs,np.sum(H>0) ))
    print np.sum(H>0),bs**2
```

W wyniku działania powyższego kodu otrzymamy następujące liczby pikseli (boxów):

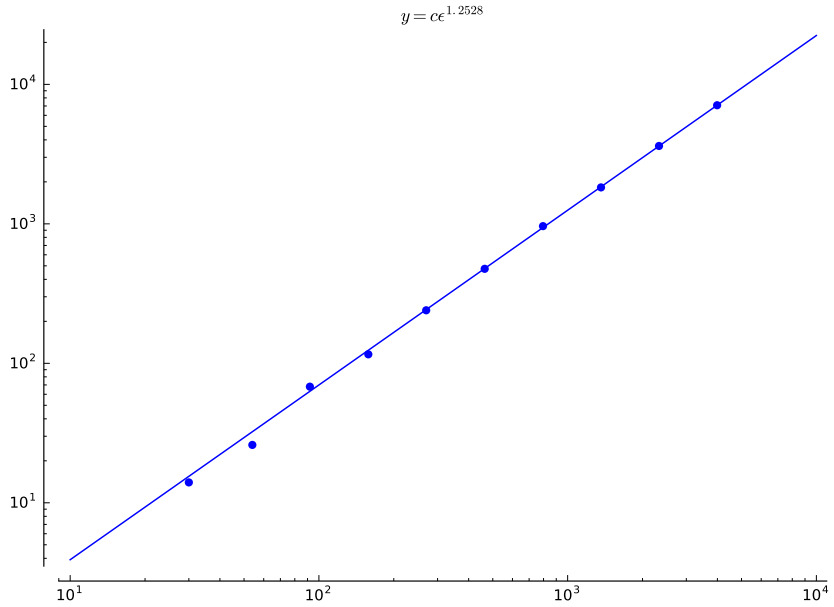
```
14 225
26 729
68 2116
116 6241
240 18225
476 53824
963 158404
1825 463761
```

```
3617 1357225
7086 3980025
```

```
var('a,d,x')
model(x)=a*x^d
fit = find_fit(scal,model)
print fit
plt = plot_loglog(model(x).subs(fit),(x,1,1e5),title="$y= c
  ↪ \epsilon^{%0.4f}$"%(d.subs(fit))) + point(scal,size=30)

plt.show()
```

Wynikiem działania powyższego kodu jest wykres Rys. 2.6 oraz dopasownie $\simeq 1.25$.



Rys. 2.7: Wykres w skali logarytmicznej liczby pikseli, którą zajmuje obiekt do całkowitej liczby pikseli (lub liczby boxów).

2.3.5 Podsumowanie

Powyższe przykłady zachęcają do przeprowadzania eksperymetnów z własnymi danymi. Można na przykład wykorzystać dane geodezyjne linii brzegowej rzek i zbadać ich wymiar fraktalny. Szczególnie prostą i uniwersalną wydaje się metoda box-counting, która w języku Python - wykorzystującym biblioteki zawarte w SageMath - zawiera się w kilku liniach kodu.

2.4 Zadziwiające przybliżenie

2.4.1 O scenariuszu

Prezentowany scenariusz jest wynikiem zajęć (zainspirowanych podręcznikiem „Matematyka się liczy" pod redakcją prof. Wacława Zawadowskiego) prowadzonych w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie. Przeznaczony jest do realizacji na dwóch godzinach lekcyjnych (po jednej na każdą z części).

Struktura materiału jest swego rodzaju rozmową z uczestnikiem zajęć: dopuszczamy więc podejście - także językowe - typu nieformalnego. Użyjemy komputera do zilustrowania **prostego** problemu, używając **podstawowego** programowania.

- Dlaczego podstawowego?

Nie znamy się bowiem na Sage'u zbyt dobrze.

- Dlaczego prostego?

Chcemy bowiem **zafrapować** zagadnieniem matematycznym uważanym za nudne. Większość uczniów nie lubi rachunków - my spróbujemy więc użyć komputera do powtórzeń żmudnych obliczeń. Być może doprowadzą nas one do zasygnalizowania pewnych nieporozumień w używaniu liczb.

Lekcje prowadził Krzysztof Oleś.

2.4.2 Wstęp

Matematyka, coraz częściej uznawana za jedną z podstaw kształcenia, jest przedmiotem diagnozy związanej z osiągnięciami wychowanków wszystkich poziomów nauczania. Przeprowadzane także i w Polsce badania w ramach Programu Międzynarodowej Oceny Umiejętności Uczniów OECD PISA sugerują, że nierzadko polscy słabsi uczniowie radzą sobie z zadaniami lepiej niż słabsi uczniowie krajów OECD; lepsi uczniowie polscy jednak rozwiązują zadania gorzej niż lepsi uczniowie OECD. Tego typu sytuacja dotyczy najczęściej zadań o średnim poziomie trudności. Konieczną potrzebą jest więc budowanie i pogłębianie aparatu matematycznego, tak by nabywane umiejętności mogły być solidnym fundamentem rozwiązywania problemów matematycznych – przede wszystkim tych, które rozszerzają przewidziane w podstawie programowej treści nauczania. Komputer może naszym zdaniem w pozytywny sposób wpływać na analizowanie pojęć i struktur wprowadzanych na lekcjach klasycznych - prowadzonych (niestety?) jedynie za pomocą przysłowiowych: kredy i tablicy...

2.4.3 Scenariusz zajęć

Część pierwsza

Rozpocznijmy przybliżaniem pierwiastka kwadratowego liczby 2. Wykorzystamy algorytm (oparty na metodzie Newtona znajdowania miejsc zerowych funkcji) znany pod hasłem: metoda babilońska.

```
rot=2
for n in range(1,6):    #zmień przedział
        rot=0.5*(rot+2/rot)
        print rot
        print "error =",abs(N(sqrt(2)-rot))
```

```
1.50000000000000
error = 0.0857864376269049
1.41666666666667
error = 0.00245310429357137
1.41421568627451
error = 2.12390141451912e-6
1.41421356237469
error = 1.59472435257157e-12
1.41421356237309
error = 2.22044604925031e-16
```

Czy wiemy

- co znaczy zapis 2.12390141451912e-6?
- jak zmieni się błąd przy zmianie n?

Zabaczmy jak szybki jest podany algorytm.

```
rot=2
graph=point((0,sqrt(2)))
for n in range(1,21):    #czy przedział (1,51) zmieni cokolwiek?
        rot=0.5*(rot+2/rot)
        graph=graph+point((n,rot))
plot(graph)
```

Zaraz, zaraz... Co mamy na myśli, używając słowa „szybki”?

Zróbmy pewnego rodzaju porównanie. Jedną z najbardziej popularnych liczb jest π, użyjemy zatem algorytmu ją przybliżającego. Oprzemy się na wzorze podanym przez Wallisa w roku 1655:

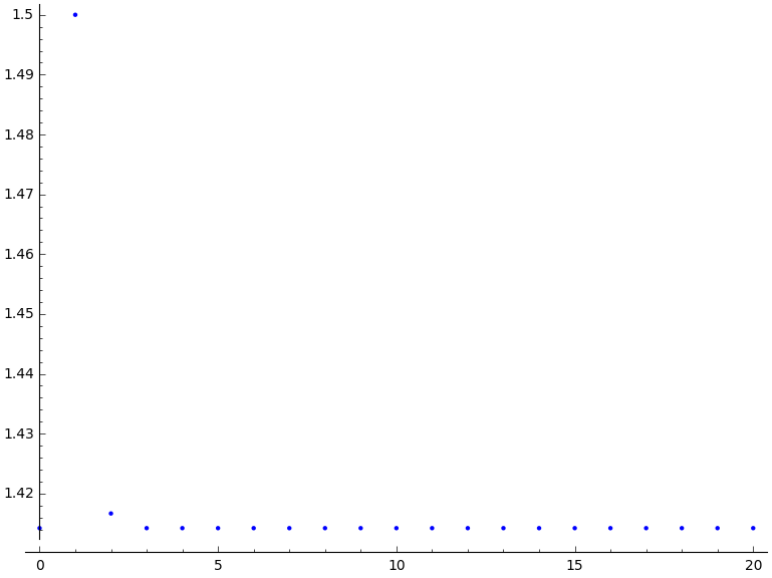
$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \left(\frac{2n}{2n-1} \cdot \frac{2n}{2n+1} \right).$$

Wydaje się to dosyć skomplikowane - prawdopodobnie z powodu użycia dużego π. A może poniższy zapis

$$\frac{\pi}{2} = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdot \frac{8}{7} \cdot \frac{8}{9} \cdot \ldots$$

jest prostszy?

Po prostu: duże π oznacza iloczyn nieskończony (coś w rodzaju wielu, wielu mnożeń...).



Jak to działa?

Męczące (można to sprawdzić na kartce papieru...) obliczenia przeprowadzi komputer.

```
w=1
for i in range(1,6):
        w=w*((2*i)/(2*i-1))*((2*i)/(2*i+1))
        print 2*w      #wolimy ułamki zwykłe czy dziesiętne?
        print "error =",abs(N(pi-2*w))
```

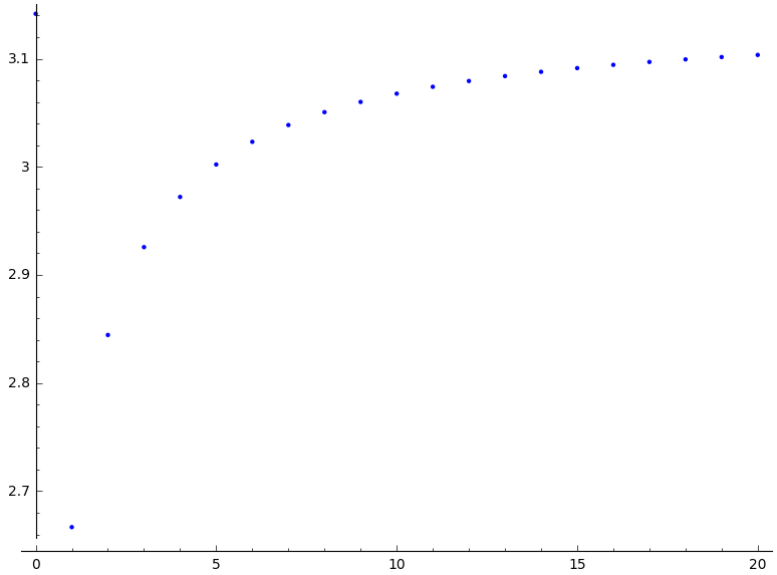
```
8/3
error = 0.474925986923127
128/45
error = 0.297148209145349
512/175
error = 0.215878367875507
32768/11025
error = 0.169438458578455
131072/43659
error = 0.139416699032886
```

Czy wiemy

- co znaczy abs()?
- co znaczy N()?
- jak zmieni się błąd przy zmianie n?
- ilu powtórzeń musimy użyć, by osiągnąć 3,14?

Zabaczmy jak szybki jest podany algorytm.

```
w=1
graph=point((0,pi))
for i in range(1,21):
        w=w*((2*i)/(2*i-1))*((2*i)/(2*i+1))
        graph=graph+point((i,2*w))
plot(graph)
```



Możemy teraz porównać szybkość pierwszego i drugiego algorytmu oraz zadać niewygodne pytania:

- Czy kiedykolwiek zastanawialiśmy się nad tym, jak nasz kalkulator przybliża liczby?
- Może kalkulator kolegi robi to lepiej? Co znaczy „lepiej”?
- Obliczaliśmy błędy - Sage musiał pierwiastek kwadratowy liczby 2 oraz π przybliżyć (nie są to liczby wymierne): czy Sage popełnił błąd? Jak duży?

Część druga

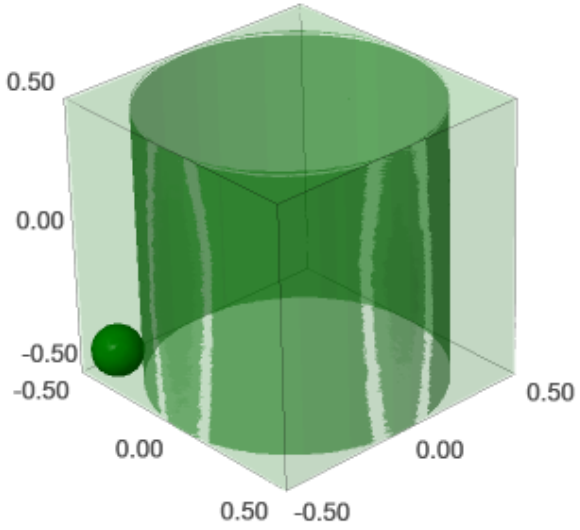
No dobrze, ale kto jest zainteresowany różnicami w przybliżeniach np. na piętnastym miejscu po przecinku?

Zajmijmy się zatem pewnym problemem geometrycznym.

Rozważmy walec wpisany w sześcian (podstawy walca są wpisane w równoległe ściany sześcianu). W rogu tegoż sześcianu umieszczamy stycznie do walca kulkę o maksymalnej objętości. Jaka jest ta objętość?

```
var('x,y,z')
r=(sqrt(2)-1)/(2*sqrt(2)+2)      #a skąd to?
a=implicit_plot3d(x^2+y^2-0.25,(x,-0.5,0.5),(y,-0.5,0.5),
(z,-0.5,0.5), color = "green", opacity = 0.4)
```

```
b=cube(center=(0, 0, 0), opacity=0.1, color = "green")
c=sphere(center=(-0.5+r,-0.5+r,-0.5+r), opacity=0.9, color =
  ↪ "green", size=r)
graph=a+b+c
graph
```



Jak widzimy długość krawędzi sześcianu wynosi 1

$$x, y, z \in (-0,5; 0,5),$$

a walec związany jest z okręgiem o równaniu

$$x^2 + y^2 = 0,25.$$

Ale skąd wzięto

$$r = \frac{\sqrt{2}-1}{2\sqrt{2}+2}?$$

Niech r oznacza promień szukanej kulki. Z prostego związku pomiędzy przekątną kwadratu oraz promieniami odpowiednich okręgów otrzymujemy:

$$\frac{1}{2}\sqrt{2} = r\sqrt{2} + r + \frac{1}{2}$$

$$\frac{1}{2}\sqrt{2} - \frac{1}{2} = r\left(1+\sqrt{2}\right)$$

$$r = \frac{\frac{1}{2}\sqrt{2} - \frac{1}{2}}{1+\sqrt{2}} = \frac{\sqrt{2}-1}{2\sqrt{2}+2}$$

a szukana objętość jest równa

$$\frac{4}{3}\pi r^3 = \frac{4}{3}\pi\left(\frac{1}{2}\right)^3\left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^3 = \frac{\pi}{6}\left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^3.$$

Wszyscy słyszeliśmy o przekształcaniu wyrażeń zawierających liczby niewymierne, zabierzmy się więc do żmudnej roboty...

$$\left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^3 = \left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\cdot\frac{\sqrt{2}-1}{\sqrt{2}-1}\right)^3 = \left(\sqrt{2}-1\right)^6,$$

ale

$$\left(\sqrt{2}-1\right)^6 = \left(\left(\sqrt{2}-1\right)^2\right)^3 = \left(3-2\sqrt{2}\right)^3,$$

$$\left(\sqrt{2}-1\right)^6 = \left(\left(\sqrt{2}-1\right)^3\right)^2 = \left(5\sqrt{2}-7\right)^2,$$

i ostatecznie

$$\left(\sqrt{2}-1\right)^6 = \left(5\sqrt{2}-7\right)^2 = 99-70\sqrt{2}.$$

Niech

$$w_1 = 99-70\sqrt{2}, \quad w_2 = \left(5\sqrt{2}-7\right)^2, \quad w_3 = \left(3-2\sqrt{2}\right)^3,$$

$$w_4 = \left(\sqrt{2}-1\right)^6, \quad w_5 = \left(\frac{\sqrt{2}-1}{\sqrt{2}+1}\right)^3.$$

Oczywiście $w_1 = w_2 = w_3 = w_4 = w_5$, ale: czy jest jakaś różnica między $w_1, \ldots, w_5$ jeśli do pierwiastka kwadratowego liczby 2 będziemy się zbliżać? Zobaczmy...

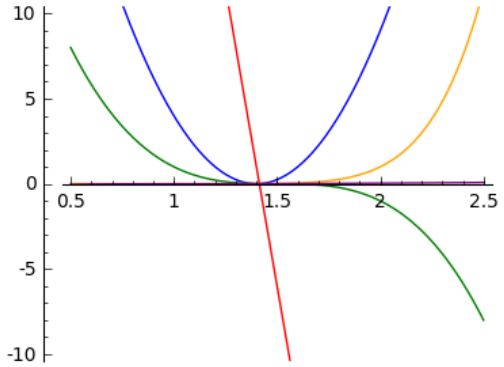
```
p=1.44      #zmień p
print 'apr=',p
print 'w_1=',N(99-70*p)
print 'w_2=',N((5*p-7)^2)
print 'w_3=',N((3-2*p)^3)
print'w_4=',N((p-1)^6)
print 'w_5=',N(((p-1)/(p+1))^3)
```

Okazuje się, że różnice pomiędzy przybliżeniami są duże, jeśli za pierwiastek kwadratowy liczby 2 przyjmiemy 1,41 (wielu ludzi przyjmuje takie przybliżenie do części setnych). A co z: 1,414; 1,4142; 1,41421; 1,414213 i tak dalej?

Różnice wciąż są duże, co możemy zobaczyć także na wykresach zwązanych z $w_1, \ldots, w_5$ funkcji.

```
apr= 1.44000000000000
w_1= -1.80000000000000
w_2= 0.0399999999999997
w_3= 0.00172800000000000
w_4= 0.00725631385599999
w_5= 0.00586392693661584
```

```
@interact
def _(xlimits=range_slider(0.5, 2.5, 0.1, default=(0.5, 2.5),
 ↪  label="horizontal range"),
    ylimits=range_slider(-10, 10, 0.1, default=(-10, 10),
     ↪  label="vertical range")):
    plt = plot(99-70*x, xlimits, color="red")
    plt = plt+plot((5*x-7)^2, xlimits, color="blue")
    plt = plt+plot((3-2*x)^3, xlimits, color="green")
    plt = plt+plot((x-1)^6, xlimits, color="orange")
    plt = plt+plot(((x-1)/(x+1))^3, xlimits, color="purple")
    show(plt, xmin=xlimits[0], xmax=xlimits[1], ymin=ylimits[0],
     ↪  ymax=ylimits[1], figsize=(4, 3))
```



Wróćmy zatem do rysunku zawierającego poszukiwaną kulkę.

```
var('x,y,z')
p=1.41      #zmień p
r1=N(0.5*((99-70*p)^(1/3)))      #skąd "^(1/3)"?
r2=N(0.5*(((5*p-7)^2)^(1/3)))
r3=N(0.5*((3-2*p)^3)^(1/3))
r4=N(0.5*((p-1)^6)^(1/3))
r5=N(0.5*(((p-1)/(p+1))^3)^(1/3))
r=r5      #zmień r
a=implicit_plot3d(x^2+y^2-0.25,(x,-0.5,0.5),(y,-0.5,0.5),
(z,-0.5,0.5), color = "green", opacity = 0.4)
b=cube(center=(0, 0, 0), opacity=0.1, color = "green")
c=sphere(center=(-0.5+r,-0.5+r,-0.5+r), opacity=0.9, color =
 ↪  "green", size=r)
```

```
graph=b+a+c
graph
```

Powinniśmy zmieniać

- p: 1.414, 1.4142, 1.41421, 1.414213; nie zapomnijmy o sqrt(2),
- r: $r_1, \ldots, r_5$.

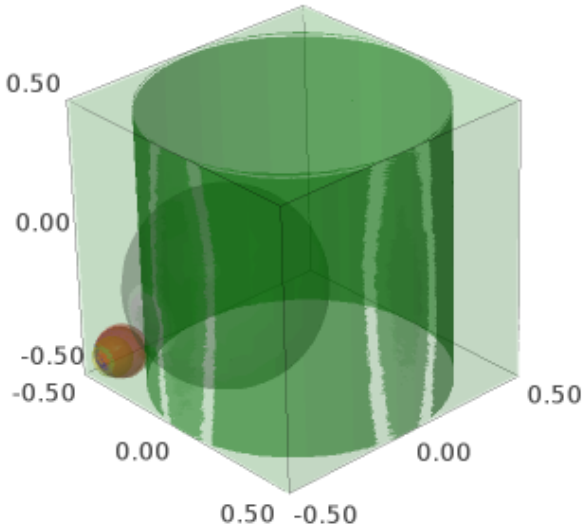
Zabaczmy pięć kulek jednocześnie.

```
var('x,y,z')
p=1.41      #dlaczego mamy problem z p=1.44?
r1=N(0.5*((99-70*p)^(1/3)))
r2=N(0.5*(((5*p-7)^2)^(1/3)))
r3=N(0.5*((3-2*p)^3)^(1/3))
r4=N(0.5*((p-1)^6)^(1/3))
r5=N(0.5*(((p-1)/(p+1))^3)^(1/3))
a=implicit_plot3d(x^2+y^2-0.25,(x,-0.5,0.5),(y,-0.5,0.5),
(z,-0.5,0.5), color = "green", opacity = 0.4)
b=cube(center=(0, 0, 0), opacity=0.1, color = "green")
c=sphere(center=(-0.5+r1,-0.5+r1,-0.5+r1), opacity=0.2, color =
 ↪  "grey", size=r1)
d=sphere(center=(-0.5+r2,-0.5+r2,-0.5+r2), opacity=0.2, color =
 ↪  "yellow", size=r2)
e=sphere(center=(-0.5+r3,-0.5+r3,-0.5+r3), opacity=0.2, color =
 ↪  "red", size=r3)
f=sphere(center=(-0.5+r4,-0.5+r4,-0.5+r4), opacity=0.2, color =
 ↪  "blue", size=r4)
g=sphere(center=(-0.5+r5,-0.5+r5,-0.5+r5), opacity=0.2, color =
 ↪  "orange", size=r5)
graph=a+b+c+d+e+f+g
graph
```

Czy to nie dziwne?

Zakończmy nasze rachunki rozważeniem poszukiwanej objętości - ponieważ mamy dość przyglądania się dalekim miejscom po przecinku, przyjmijmy, że długość krawędzi sześcianu wynosi 60.

```
p=1.41      #zmień p
print 'apr=',p
w_1=N(99-70*p)
w_2=N((5*p-7)^2)
w_3=N((3-2*p)^3)
w_4=N((p-1)^6)
w_5=N(((p-1)/(p+1))^3)
print 'volume 1=',N(pi)*36000*w_1      #skąd 36000?
print 'volume 2=',N(pi)*36000*w_2
```




```
print 'volume 3=',N(pi)*36000*w_3
print 'volume 4=',N(pi)*36000*w_4
print 'volume 5=',N(pi)*36000*w_5
```

```
apr= 1.41000000000000
volume 1= 33929.2006587711
volume 2= 282.743338823079
volume 3= 659.583660806486
volume 4= 537.224133143207
volume 5= 556.868709967303
```

I znowu - powinniśmy zmieniać p: 1,414; 1,4142; 1,41421; 1,414213; nie zapomnijmy o sqrt(2).

I po raz kolejny: czy to nie dziwne? Może nie (?!), ale powyższy przykład pokazuje, jak bardzo należy uważać, wykorzystując w rachunkach przybliżenia.

2.4.4 Podsumowanie

Chcieliśmy pokazać, jak ważna jest różnica w użyciu wyrażenia algebraicznego w rodzaju

$$\frac{\sqrt{2}-1}{2\sqrt{2}+2}$$

a jego przybliżeń. Dlaczego?

Po pierwsze: ponieważ używamy liczb, które nie są wymierne, a oznacza to konieczność ich przybliżania. Próbowaliśmy pokazać dwa różne - jeśli chodzi o liczbę koniecznych do odpowiedniego przybliżenia powtórzeń - algorytmy. Zasugerowaliśmy znalezienie niemałej liczby

koniecznych powtórzeń, by uzyskać przybliżenia liczby π przysłowiowym 3,14. Ponieważ jednak dziesiętne przybliżenia mogą nie wydawać się interesujące - zdecydowaliśmy się zobaczyć (!) ich wagę w problemie geometrycznym, w którym szczególną rolę odegrały przybliżenia pierwiastka kwadratowego liczby 2.

Po drugie: ponieważ w szkołach polskich mamy do czynienia z przewagą rozwiązań (np. równań) w postaci algebraicznej. Oznacza to, że maturalne rozwiązanie równania

$$7x^2 + 27x - 31 = 0$$

powinno mieć postać

$$x_1 = \frac{-27 - \sqrt{1597}}{14}, \quad x_2 = \frac{-27 + \sqrt{1597}}{14}.$$

Wydaje się, że warto czasem zwrócić uwagę na mentalną przepaść pomiędzy powyższymi „obrazkami" a poniższymi „liczbami"

$$x_1 \approx -4,78303; \quad x_2 \approx 0,92589.$$

Być może nasze rozważania dotyczą jedynie (?) różnic między znakami

$$= \qquad \text{oraz} \qquad \approx$$

2.5 RSA szyfrowanie asymetryczne

2.5.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenie 3h zajęć lekcyjnych.

Został on opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Krzysztofa Jarczewskiego.

2.5.2 Wstęp

Uczniowie powinni znać i rozumieć:

- działania na potęgach, NWD, dzielenie z resztą. *(1.1, 1.4, 1.5 mat_p)*,
- podstawowy algorytm Euklidesa a najlepiej rozszerzoną jego wersję, *(1.0-II-5.11.a inf_r)*,
- podstawowe komendy programistyczne w SageMath: działania, funkcję warunkową, pętle *(1.0-II-5.22-23 inf_r)*,
- potrzebę szyfrowania wiadomości *(1.0-II-2.5, 1.0-II-5.11.e inf_r)*,
- kodowanie znaków ASCI *(1.0-II-5.11.d inf_r)*.

Uczniowie na poniższych zajęciach poznają:

- własności działania modulo (kongruencję), chińskie twierdzenie o resztach,
- małe twierdzenie Fermata, proste szyfrowanie asymetryczne,
- szyfrowanie asymetryczne RSA – oparte na liczbach pierwszych *(1.0-II-5.11.e inf_r)*.

Powyżej w nawiasach jest wpisany szczegółowy zakres nauczanych treści.

mat_p – matematyka poziom podstawowy, inf_r – informatyka poziom rozszerzony.

2.5.3 Część teoretyczna

Definicja kongruencji.

Dwie liczby całkowite : a, b przystają do siebie **modulo** *n*, jeśli: $(a - b) \cdot k = n, \;\; k \in Z.$

Kongruencję zapisujemy symbolicznie: $a = b(mod \;\; n)$.

Przykłady:

2 = 2 (mod 8), ponieważ 2 - 2 = 0, 0 jest podzielne przez 8,

3 = 18 (mod 5), ponieważ 3 - 18 = -15, -15 jest podzielne przez 5,

100 = 1 (mod 9), ponieważ 100 - 1 = 99, 99 jest podzielne przez 9,

250 = 206 (mod 22), ponieważ 250 - 206 = 44, 44 jest podzielne przez 22.

Ćwiczenie 1.

Znajdź x takie, że: $3x = 1(mod\ 4),\ 5 < x < 10$.

```
for x in range(5,11):          #You can change this range
    if (3*x - 1) % 4 == 0:     #You can change this equation
        print "x =",x
```

x=7

Ćwiczenie 2.

Znajdź x takie, że: $3x + 2 = 1(mod\ 5)$.

```
for x in range(40):
    if (3*x+2 - 1) % 5 == 0:
        print x
```

```
3 8 13 18 23 28 33 38
```

Oczywiście istnieje nieskończenie takich rozwiązań. Dodatkowo te rozwiązania wyznaczają ciąg arytmetyczny.

Ćwiczenie 3.

Znajdź x takie, że: 3x = 1 (mod 6).

```
odp  = "?"
for x in range(100):
    if (3*x-1) % 6 == 0:
        odp = x
print odp
```

W powyższym ćwiczeniu nie istnieje żadna liczba, która spełnia powyższą kongruencję.

Informacja: Chińskie twierdzenie o resztach.

Poniższe ćwiczenie można rozwiązać przy użyciu chińskiego twierdzenia o resztach. Jedno z najważniejszych twierdzeń z teorii liczb i kryptografii. Twierdzenie to pozwala dzielić sekret wśród kilku osób (ważne hasło liczbowe).

Ćwiczenie 4.

Tabliczka czekolady składa się z mniej niż 100 kawałków. Przy podziale na trzy równe części, pozostaje 1 kawałek czekolady. Dzieląc na 5 równych części, zostają 3 kawałki czekolady, a przy podziale na 7 równych części, pozostają 2 kawałki.

Wiemy, że liczba kawałków czekolady musi spełniać poniższe kongruencje:

x = 1 mod 3,

x = 3 mod 5,

x = 2 mod 7.

```
for x in range(100):
    if (x-1) % 3 == 0 and (x-3) % 5 == 0 and (x-2) % 7 == 0:
        print x
```

Otrzymujemy 58.

Małe twierdzenie Fermata.

Jeśli p jest liczbą pierwszą oraz a, p są względnie pierwsze, **wtedy** $a^{p-1}-1$ jest wielokrotnością liczby p. Zapisujemy to symbolicznie: $a^{p-1} = 1(mod\ p)$.

Sprawdźmy poprawność powyższego twierdzenia, dla kolejnych liczb pierwszych, numerycznie z wykorzystaniem języka Python.

Dla a = 35 i p = 5 lub p = 7 liczby nie spełniają założeń twierdzenia. Możemy dodatkowo stwierdzić, że liczba $a^{p-1} - 1$ jest podzielna przez p.

```
for x in range (1, 30):
    p = nth_prime(x)
    print(p, 35^(p-1) % p)
```

2.5.4 Część informatyczna

Informacja: Szyfrowanie wiadomości.

Pierwsze wzmianki o kryptografii pochodzą już ze starożytności. Można stwierdzić, że szyfrowanie powstało równocześnie z wynalezieniem pisma. Szyfrowanie było stosowne przy

przekazywaniu wiadomości wojskowych lub politycznych. Na lekcjach informatyki poznaliśmy (lub poznamy) szyfr Cezara. Jest to prosty szyfr, w którym zamieniamy litery. Co prawda zaszyfrowana wiadomość jest niezrozumiała, ale także prosta do odszyfrowania. Inne metody starożytnych były bardziej wyrafinowane i trudniejsze do odszyfrowania. Do lat sześćdziesiątych dwudziestego wieku znane były tylko szyfry symetryczne, to znaczy takie, które mają jeden klucz (jedną metodę) dzięki, któremu szyfrujemy i deszyfrujemy wiadomości.

W latach siedemdziesiątych dwudziestego wieku kryptografowie dzięki informatyzacji, zwiększeniu mocy obliczeniowej komputerów oraz potrzebie zabezpieczenia danych wymyślili szyfr asymetryczny, czyli taki, w którym używamy dwóch różnych kluczy – jeden do zaszyfrowania, a drugi do odszyfrowania (kolejność kluczy jest nieważna). Jeden z kluczy udostępniamy osobie, która ma przesłać nam tajną wiadomość. Możemy nawet udostępnić klucz na naszej stronie internetowej (dostępny dla wszystkich - klucz publiczny). Drugi klucz jest tajny (klucz prywatny) znamy go tylko my i nie możemy go nikomu udostępnić. Tylko i wyłącznie dzięki kluczowi prywatnemu możemy odszyfrować wiadomość.

Poniżej opiszemy prosty szyfr asymetryczny, który można złamać (czyli znając liczby d, n można szybko znaleść liczbę e). Będzie to Wasze zadanie dodatkowe.

Jak matematycznie stworzyć szyfr asymetryczny?

Do stworzenia prostego szyfru asymetrycznego będą nam potrzebne różne liczby naturalne: $a, b, a1, b1$.

Czym większe liczby tym szyfr jest bezpieczniejszy - trudniejszy do odszyfrowania bez znajomości odpowiedniego klucza.

Dla naszego przykładu wystarczą liczby dwu, trzy cyfrowe.

Obliczamy: $M = a \cdot b - 1$, wtedy: $e = a1 \cdot M + a, \quad d = b1 \cdot M + b, \quad n = (e \cdot d - 1)/M$

Otrzymujemy parę kluczy, klucz publiczny: (d, n) i klucz prywatny: (e, n).

Poniżej przykład generowania kluczy oraz zaszyfrowania liczby.

```
number = 1234567   #You can change this number (message). What will
 ↪  be if number larger then n?
a = 89             #you can change the numbers: a, b, a1, b1
b = 45
a1 = 98
b1 = 55
M = a*b-1
e = a1*M+a
d = b1*M+b
n = (e*d-1)/M
print "public key:", (d, n)
print "private key:",(e, n)
 # encryption
szyfr = (number*d) % n
print "encryption:", szyfr
# decryption
deszyfr = (szyfr*e) % n
print "decryption:", deszyfr
```

```
public key: (220265, 21590866)
private key: (392481, 21590866)
encryption: 16533851
decryption: 1234567
```

Co zrobić gdy liczba jest więsza od n?

1. Obliczamy resztę z dzielenia przez n (otrzymujemy „porcję” do zaszyfrowania).
2. Szyfrujemy otrzymaną „porcję”.
3. Do szyfru dodajemy zaszyfrowaną „porcję” w kolejnej potędze liczby n.
4. Dzielimy liczbę przez n.
5. Jeśli otrzymana liczba jest większa od 0, to powtarzamy kroki 1-4

```
number=12345656767563535236421387987979799674354678943534 52 #Big
 ↪  number(message)
szyfr = 0
i=0
while number>0:                                  # 5
    pomoc = number%n                             # 1
    szyfr = szyfr + ((pomoc*d) % n)*n^i          # 2, 3
    i=i+1
    number = int(number/n)                       # 4
print szyfr
```

W podobny sposób deszyfrujemy wiadomość:

Pomoc:

number → szyfr	szyfr → deszyfr	d→e

Spróbuj poniżej odszyfrować liczbę:

```
i=0
while number>0:                                  # 5
    pomoc = number%n                             # 1
    szyfr = szyfr + ((pomoc*d) % n)*n^i          # 2, 3
    i=i+1
    number = int(number/n)                       # 4
print szyfr
```

Zazwyczaj chcemy zaszyfrować tekst, a nie liczbę, czyli musimy zamienić litery (znaki) na liczbę. Do tego posłużymy się kodem ASCII.

Każdej literze, znakowi przyporządkowana jest liczba z przedziału od 1 do 128.

Poniżej algorytm szyfrowania wiadomości tekstowej (ten kod został napisany i wprowadzony przez uczniów na zajęciach).

```
number=0
i=0
tekst="This is the secret message or anything."
for x in tekst:
    i=i+1
    print x,"->", ord(x)," ",
    if (i%10==0):
        print
    number=number + ord(x)*128^i
print
print "number =", number
```

Pełny algorytm szyfrujący

Po złożeniu powyższych programów otrzymujemy pełny algorytm szyfrowania i deszyfrowania wiadomości tekstowych.

```
number = 0
i = 0
tekst = "This is the secret message or anything." #message
tekst2 = ""
print "message:", tekst
# change text to number
for x in tekst:
    i = i + 1
    number = number + ord(x)*128^i
print "number:", number
print ""
# encription
szyfr = 0
i=0
while number>0:
    pomoc=number%n
    szyfr = szyfr + ((pomoc*d) % n)*n^i
    i = i + 1
    number = int(number/n)
print "encription:", szyfr
```

```
message: This is the secret message or anything.
number: 710462119235500194958769552333505678558759290␣
↪2
↪5684299925302283649808043559662611097␣
↪6
encription: 247771732970102709758504535275676311805105␣
↪6145804692906609710645765611862711721717856778
```

Pełny algorytm deszyfrujący

```
tekst2 = ""
deszyfr = 0
i = 0
print "encription:", szyfr
# decription
while szyfr>0:
    pomoc = szyfr%n
    deszyfr = deszyfr + ((pomoc*e) % n)*n^i
    i = i+1
    szyfr = int(szyfr/n)
print "decription: ", deszyfr
## change number to text
i = 0
while deszyfr>0:
    i = i+1
    deszyfr = int(deszyfr/128)
    tekst2 = tekst2 + chr(deszyfr%128)
print "message: ", tekst2
```

```
encription: 24777173297010270975850453527567631180510␣
↪56145804692906609710645765611862711721717856778
decription: 710462119235500194958769552333505678558 75␣
↪92902568429992530228364980804355966261109 76
message:  This is the secret message or anything.
```

2.5.5 Szyfrowanie asymetryczne RSA

RSA jeden z pierwszych i najpopularniejszy asymetryczny algorytm kryptograficzny z kluczem publicznym, zaprojektowany w 1977 przez Rona Rivesta, Adi Szamira oraz Leonarda Adlemana (jego nazwa pochodzi od pierwszych liter nazwisk jego twórców).

Bezpieczeństwo szyfru RSA opiera się na rozkładzie dużych (ponad dwustucyfrowych) liczb złożonych na liczby pierwsze (trudność faktoryzacji).

Poniżej przykład

1. Wybieramy liczby pierwsze 20-34 cyfrowe.
2. Mnożymy je i wyznaczamy podział otrzymanej liczby złożonej na czynniki pierwsze (to trwa bardzo długo).

```
%time
@interact
def _(n=slider( srange(20,32,2))):
```

```
    a = int(random()*10^n)
    a = next_prime(a)
    print a
    b = int(random()*10^n)
    b = next_prime(b)
    print b
    n = a*b
    print(factor(n))
```

Zobacz jeszcze przewidywania dla dłuższych liczb.

```
@interact
def _(n=slider( range(34,101,2))):
    t = 2^((n-34)/2)
    print n,"-digits prime numbers, factoring time:", t, "minutes"
    if t>100 and t<60*24:
        print n,"-digits prime numbers, factoring time:", int(t/60),
          ↪ "hours"
    elif t>60*24 and t<60*24*365:
        print n,"-digits prime numbers, factoring time:",
          ↪ int(t/60/24), "days"
    elif t>60*24*365:
        print n,"-digits prime numbers, factoring time:",
          ↪ int(t/60/24/365), "year"
```

Generowanie szyfru RSA

1. Wybieramy dwie duże liczby pierwsze: p, q (w praktyce wykorzystuje się liczby ponad stocyfrowe, ale dla naszych porzeb wystarczą liczby trzycyfrowe).
2. Obliczamy: $n = p \cdot q, \;\; f = (p-1)(q-1)$.
3. Wybieramy dowolną nieparzystą liczbę e, taką że:$1 < e < f$ and $gcd(d, \;\; f) = 1$.
4. Wyznaczamy liczbę d as: $de = 1 \; (mod \; f)$.

Klucz publiczny to para liczb: (d, n).

Klucz prywatny to para liczb: (e, n).

```
los = int(100*random())
p = nth_prime(30+los)
los = int(100*random())
q = nth_prime(30+los)
n = p*q
f = (p-1)*(q-1)
los = int(f*random())
e = next_prime(los)
print "p =",p, ", q =",q, ", e =",e, ", n =", n, ", f =", f
```

Ostatecznie należy wyznaczyć liczbę e taką, że: $(d \cdot e)\ modf = 1$.

Możemy użyć rozszerzonego algorytmu Euklidesa do wyznaczenia liczby e. Moi uczniowie zmieniając istniejący program w Internecie napisali poniższy program, ale nie zawsze generuje on prawidłową liczbę. Spróbuj poprawić ten kod!

```
a = e
p0 = 0
p1 = 1
a0 = a
n0 = f
q  = int(n0/a0)
r  = n0 % a0
while (r > 0):
    t = p0 - q * p1
    if (t >= 0):
        t = t % n
    else:
        t = n - ((-t) % n)
    p0 = p1
    p1 = t
    n0 = a0
    a0 = r
    q  = int(n0/a0)
    r  = n0 % a0
d = p1
print "verification : (d*e)%f =", (d*e)%f
print " public key:", d, n
print "private key:", e, n
```

Pełny algorytm szyfrowania RSA

Wystarczy skopiować algorytm szyfrowania z punktu 2 i zamienić: pomoc*d na pomoc^d.

```
number = 0
i = 0
tekst = "This is secret message or anything." #message
tekst2 = ""
print "message:", tekst
# change text to number
for x in tekst:
    i = i+1
    number = number + ord(x)*128^i
print "number:", number
print ""
# encription
szyfr = 0
i = 0
while number>0:
```

```
    pomoc = number%n
    szyfr = szyfr + ((pomoc^d) % n)*n^i
    i = i + 1
    number = int(number/n)
print "encription:", szyfr
```

Pełen algorym deszyfrujący RSA

Wystarczy skopiować algorytm deszyfrowania z punktu 2 i zamienić: pomoc*d na pomoc^d.

```
tekst2 = ""
deszyfr = 0
i = 0
print "encription:", szyfr
# decription
while szyfr>0:
    pomoc = szyfr%n
    deszyfr = deszyfr + ((pomoc^e) % n)*n^i
    i = i + 1
    szyfr = int(szyfr/n)
print "decription: ", deszyfr
## change number to text
i = 0
while deszyfr>0:
    i = i + 1
    deszyfr = int(deszyfr/128)
    tekst2 = tekst2 + chr(deszyfr%128)
print "message: ", tekst2
```

2.5.6 Wnioski

Uczniowie naszej szkoły przed projektem iCSE mogli usłyszeć wykład o metodach szyfrowania. Wykazali oni duże zainteresowanie tą sprawą. Dlatego zdecydowałem się zorganizować lekcje z asymetrycznego szyfrowania przy użyciu języka programowania Python. Język SageMath umożliwia pracę na dużych liczbach przekraczających zakres zmiennych typu float, double, a jednocześnie szybkość obliczeniowa jest naprawdę imponująca. W ten sposób uczniowie mieli możliwość praktycznego sposobu szyfrowania i deszyfrowania wiadomości przy użyciu publicznych i prywatnych kluczy. Zajęcia odbywały się na dodatkowych godzinach w ramach iCSE for school w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie. Celem zajęć było rozszerzenie nauczania matematyki i informatyki w drugiej klasie liceum. Powyższy temat nadaje się również jako praca projektowa, która łączy wiedzę matematyczno-informatyczną. Jak wiadomo powyższe elementy są istotne w dziedzinie kryptografii, która łączy teorię liczb z praktyką programistyczną. Nie przekracza to zakresu materiału przewidzianego na rozszerzeniu z informatyki liceum lub technikum. Dlatego też postanowiłem przeprowadzić lekcje dotyczące asymetrycznego szyfrowania wiadomości **RSA**.

Materiał dla uczniów jest podzielony na trzy rozdziały (trzy godziny dydaktyczne). Pierwszy z nich wprowadza pojęcia kongruencji oraz istotne matematyczne twierdzenia, które są wykorzystywane w kryptografii. Co prawda dowody i szczegóły zagadnień są pominięte, ale zainteresowani uczniowie bez problemu znajdą te informacje w internecie. Drugi rozdział to szczegółowe wprowadzenie szyfrowania asymetrycznego stosowanego na początku lat 70 poprzedniego stulecia (obecnie stosowanego już tylko w celach dydaktycznych). Trzeci rozdział to już pełne szyfrowanie RSA. W każdej części są wyszczególnione ćwiczenia i zadania dla uczniów.

Zadaniem uczniów było uzyskanie matematycznej znajomości kongruencji, małego twierdzenia Fermata i algorytmu euklidesowego. Te kwestie zostały zaprezentowane na początku, a uczniowie rozwiązywali swoje zadania podczas warsztatów. Każdy uczeń wygenerował własną parę kluczy, szyfrował i deszyfrował własne wiadomości. Pomimo wiedzy teoretycznej uczniów, było dość zaskakujące dla nich, że nie można odszyfrować wiadomości z tym samym kluczem i że klucze można zamienić. Oznacza to, że prywatny klucz może stać się publicznym i odwrotnie. Największą niespodzianką dla uczniów była symulacja złamania hasła RSA - dla liczby dwustucyfrowej szacowany podział na czynniki pierwsze dla szybkiego komputera zająłby to ponad 3000 lat.

2.6 Przybliżanie wielomianami

2.6.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenia 3h zajęć lekcyjnych.

Został on opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Krzysztofa Jarczewskiego.

2.6.2 Wstęp

Uczniowie powinni znać:

> Funkcję liniową i kwadratową *(4.6–10 mat_p)*, pojęcie wielomianu *(3.6 mat_r)*, definicję silnia *(10.1 mat_r)*.
>
> Podstawowe komendy programistyczne w SageMath: działania, funkcję warunkową, pętle *(1.0-II-5.22-23 inf_r)*.

Uczniowie na poniższych zajęciach poznają:

- sposoby implementacji i obliczania silni,
- pochodną funkcji i sposoby jej obliczania *(11.2 mat_r)*,
- wyznaczanie prostej, paraboli i wielomianu przechodzącego przez dane punkty *(3.2 mat_p)*,
- wzór Taylora oraz jego interpretację geometryczną.

Powyżej w nawiasach jest wpisany szczegółowy zakres nauczanych treści.

> *mat_p – matematyka poziom podstawowy,*
>
> *mat_r – matematyka poziom rozszerzony,*
>
> *inf_r – informatyka poziom rozszerzony.*

2.6.3 Część teoretyczna

Definicja silni.

Silnia z liczby naturalnej n to iloczyn wszystkich liczb naturalnych mniejszych lub równej n. Symbolicznie zapisujemy $n!$.

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)!, \quad n > 0 \end{cases}$$

Przykład.

$$4! = 4 \cdot 3! = ... = 4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24$$

Silnia w SageMath.

Pierwszy przykład liczy silnię zgodnie z definicją.

```
silnia=1
for i in range(1,6):
     silnia=silnia*i
     print i, '!=', silnia
```

```
1!= 1
2!= 2
3!= 6
4!= 24
5!= 120
```

Drugi przykład przy obliczaniu silni korzysta z wbudowanej funkcji w SageMath.

```
print 5, '!=', factorial(5)
```

Poniżej przykłady obliczania pochodnej w SageMath z wykorzystaniem instrukcji *diff*.

```
f=x^5  #you can change this function
show("f(x)=",f)
show("f'(x)=",f.diff(x))
```

```
f=sin(x)
show("f(x)=",f)
show("f'(x)=",f.diff(x))
```

Kolejne wzory dotyczące pochodnej funkcji.

Poniżej wzory na pochodną sumy, różnicy, iloczynu i ilorazu funkcji.

$$f, g - funkcje, \qquad c - liczba\ rzeczywista$$
$$(c \cdot f)' = c \cdot f'$$
$$(f + g)' = f' + g'$$
$$(f - g)' = f' - g'$$
$$(f \cdot g)' = f' \cdot g + f \cdot g'$$
$$(f/g)' = (f' \cdot g - f \cdot g')/g^2$$

Informacja: Liczba przed zmienną nie zmienia operacji na pochodnej.

Wyrażenia algebraiczne oddzielone + lub - liczą się oddzielnie.

Przykłady

```
f=x^3-2*x^2+3*x-4   #you can change this function
show("1. f(x)=",f,",         f'(x)=",f.diff(x))
f=x*cos(x)
show("2. f(x)=",f,",        f'(x)=",f.diff(x))
f=x^2*sin(x)
show("3. f(x)=",f,",        f'(x)=",f.diff(x))
f=sin(x)/x
show("4. f(x)=",f,",       f'(x)=",f.diff(x))
```

Pochodne z pochodnych - pochodne wyższych rzędów.

Oczywiście, możemy obliczyć pochodną z pochodnej. Pochodne wyższego rzędu zapisujemy w następujący sposób:

$$f''(x), \qquad f'''(x), \qquad f''''(x), \qquad \ldots$$
$$f^{(2)}(x), \qquad f^{(3)}(x), \qquad f^{(4)}(x), \qquad \ldots$$

Poniżej obliczenia wyższych rzędów pochodnej w SageMath:

```
f=x^3-3*x^2  #you can change this function
show ("      f(x)=",f, "         f'(x)=", f.diff(x))
show ("f''(x)=",f.diff(x,2),"          f'''(x)=", f.diff(x,3))
```

```
f=sin(x)
show('f(x)=',f)
show("f'(x)=",f.diff(x))
show("f''(x)=",f.diff(x,2))
show("f'''(x)=",f.diff(x,3))
show("f''''(x)=",f.diff(x,4))
```

Obliczanie wartości pochodnej w punkcie.

Pochodna funkcji jest oczywiście funkcją, więc możemy obliczyć wartość pochodnej dla argumentu.

Przykłady

```
f=sin(x) #you can change this function
w1=f.diff(x).substitute(x = 0)
w2=f.diff(x).substitute(x = pi/3)
show("f(x)=", f, ",         f'(x)=",f.diff(x), ",         f'(0)=" ,
 ↪  w1, ",         f'(pi/3)=", w2)
```

```
g=x^4+3-2*x^3+5*x  #you can change this function
w1=g.diff(x,2).subs(x = 1)
w2=g.diff(x,2).subs(x = 2)
show("g(x)=", g, ",       g''(x)=",g.diff(x,2), ",       g''(1)=" ,
 ↪  w1, ",       g''(2)=", w2)
```

Definicja wielomianu.

Wielomianem stopnia n zmiennej x nazywamy funkcję:

$$W(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_n \cdot x^n, \qquad a_0, a_1, a_2, ..., a_n - wspczynniki.$$

Informacja: Funkcja liniowa i funkcja kwadratowa jest wielomianem.

$$W_1(x) = a_0 + a_1 \cdot x$$
$$W_2(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2$$

2.6.4 Informatyczne obliczanie wielomianów

Informacja: Funkcja liniowa.

Wiemy, że przez dwa punkty przechodzi dokładnie jedna prosta. Ponadto znając współrzędne powyższych punktów, możemy określić wzór tej prostej. Przypomnijmy, że wzór jest funkcję liniową:

$$y = ax + b$$

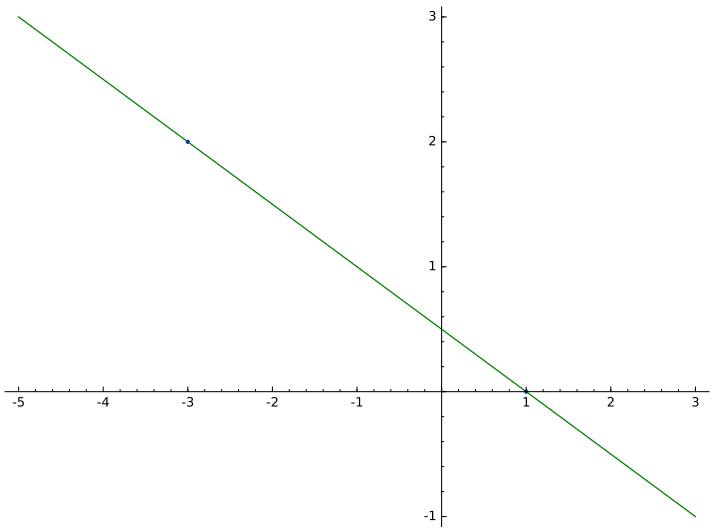
Współczynnik kierunkowy i wyraz wolny możemy obliczyć z poniższych wzorów:

$$a = \frac{y_2 - y_1}{x_2 - x_1}$$
$$b = y_1 - ax_1$$

Wpisując odpowiednie równania, możemy narysować linię prostą przechodzącą przez dwa punkty.

```
x1=-int(random()*4)
y1=int(random()*9-4)
x2=int(random()*4)+1
y2=int(random()*9-4)
p1=point((x1,y1),size=10)
p2=point((x2,y2),size=10)
a=(y2-y1)/(x2-x1)
b=y1-a*x1
f=a*x+b
show ('y=',f)
g=plot(a*x+b,xmin=x1-2, xmax=x2+2, color="green")
show(p1+p2+g,figsize=4)
```

a plot as in Rys. 2.6.4.



Informacja: Parabola.

Poniżej znajduje się przykład dotyczący trzech punktów, które nie są współliniowe. Możemy wyznaczyć funkcję kwadratową do której należą te punkty. Więc musimy wyznaczyć z poniższych równań współczynniki *a, b, c* funkcji kwadratowej.

$$\begin{cases} y_1 = ax_1^2 + bx_1 + c \\ y_2 = ax_2^2 + bx_2 + c \\ y_3 = ax_3^2 + bx_3 + c \end{cases}$$

Te obliczenia są żmudne i czasochłonne, nawet dla konkretnego przykładu. Gdybyśmy chcieli wyznaczyć odpowiednie wzory, jak powyżej dla funkcji liniowej, to zajęłoby to nam dużo czasu.

Poniżej wykorzystamy możliwości SageMath.

```
x1=-1
y1=0
x2=1
y2=4
x3=3
y3=-1
p1=point((x1,y1),size=10)
p2=point((x2,y2),size=10)
p3=point((x3,y3),size=10)
show(p1+p2+p3,figsize=3)
```

Obliczamy następujące równania, z których szukamy współczynniki: *a, b, c*.

$$\begin{cases} y_1 = ax_1^2 + bx_1 + c \\ y_2 = ax_2^2 + bx_2 + c \\ y_3 = ax_3^2 + bx_3 + c \end{cases}$$

Zamieniamy powyższy układ równań na odpowiednie równanie macierzowe.

$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

W SageMath możemy łatwo rozwiązać powyższe równanie, wystarczy zastosować poniższe działanie:

$$Mv, \quad where \quad M - matrix, \quad v - vector \quad [y1, y2, y3]$$

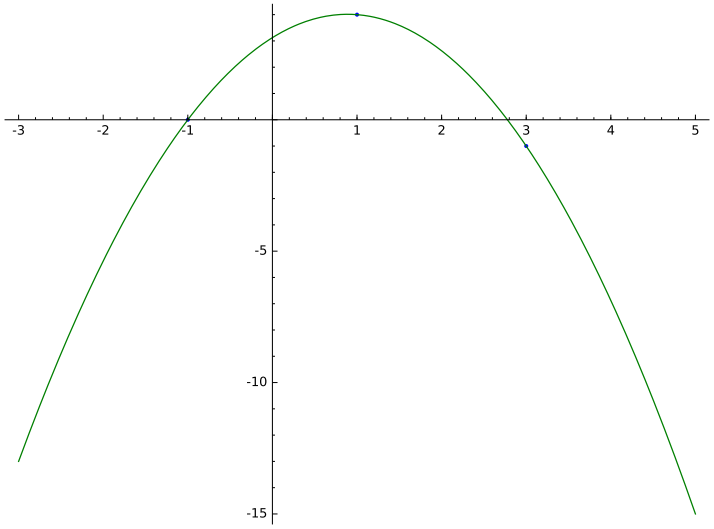
```
M = matrix(3,3,[[x1^2,x1,1],[x2^2,x2,1],[x3^2,x3,1]])
v = vector([y1,y2,y3])
wynik = M\v
[a,b,c]=wynik
show("a=",a,",  b=",b, ",  c=",c)
f=a*x^2+b*x+c
show('y=',f)
g=plot(f,xmin=-3, xmax=5, color="green")
show(p1+p2+p3+g,ymin=-7, ymax=8, figsize=4)
```

a plot as in Rys. 2.6.4.

2.6.5 Wielomian

Oto przykład dla kilku losowych punktów. Otrzymana funkcja jest wielomianem.

Jeśli podasz n punktów, to na pewno przechodzi przez te punkty wielomianem stopnia mniejszego od n.




```
points={}
vector_x=[]
vector_y=[]
k=6                    #number of points
y=int(random()*7-3)
vector_y=[y]
points=point((0,y),size=10)
print '(',0,',',y,')'
for i in range(k-1):
    vector_x=vector_x+[0]
vector_x=vector_x+[1]
for n in range(k-1):
    x=n+1
    for i in range(k):
        vector_x=vector_x+[x^(k-i-1)]
    y=int(random()*7-3)
    vector_y=vector_y+[y]
    print '(',x,',',y,')'
    points = points + point((x,y),size=10)
show(points,ymin=-2,ymax=6,figsize=4)
```

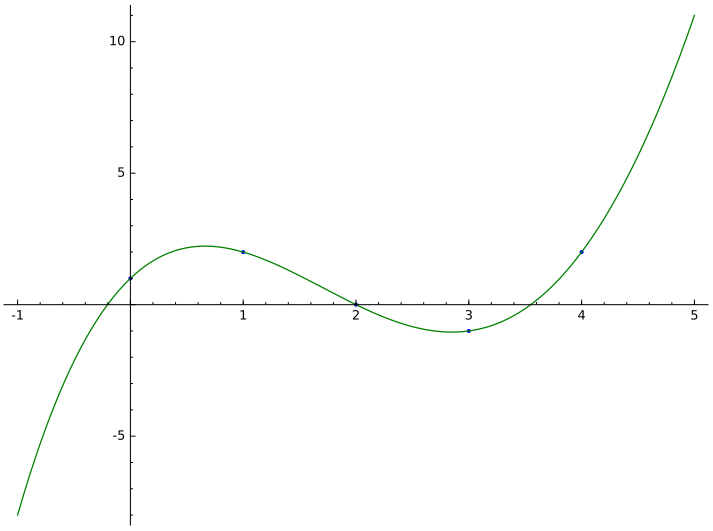
Dla losowych punktów obliczamy współczynniki wielomianu.

```
M = matrix(k,k,vector_x)
v=vector(vector_y)
wynik = M\v
show(M)
show(wynik)
```

Rysujemy wielomian, który przechodzi przez podane punkty.

```
var('x')
vector_x=[]
for i in range(k):
    vector_x=vector_x+[x^(k-i-1)]
w=vector(vector_x)
f=w*wynik
show("f(x)=",f)
f=plot(f,xmin=-1, xmax=k, color="green")
show(points+f,ymin=-7,ymax=8,figsize=6)
```

a plot as in Rys. 2.6.5.



2.6.6 Wzór Taylora

Z analizy matematyczna znany poniższy jest wzór, który przybliża dowolną funkcję pewnym odpowiadającym tej funkcji wielomianem.

Wzór Taylora

$$f(x) = f(a) + \frac{x-a}{1!}f^{(1)}(a) + \frac{(x-a)^2}{2!}f^{(2)}(a) + \ldots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \ldots$$

Możemy uprościć powyższy wzór podstawiajac za a=0.

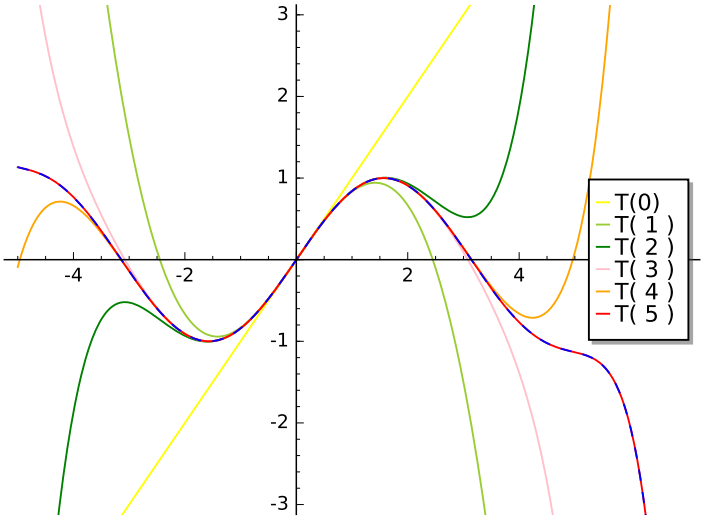
Wzór Taylora-Maclaurina.

$$f(x) = f(0) + \frac{x}{1!}f^{(1)}(0) + \frac{x^2}{2!}f^{(2)}(0) + \ldots + \frac{x^n}{n!}f^{(n)}(0) + \ldots$$

To jest przykład dla funkcji $f(x) = sin(x)$.

```
kolor=[]
kolor=["yellowgreen","green","pink","orange","red","brown","black"]
n=6
f=x
q=plot(f,xmin=-4,xmax=6, ymin=-3, ymax=3,color="yellow",
 ↪  legend_label="T(0)")
for i in range(1, n):
    k=2*i+1
    f=f+(-1)^i*(1/factorial(k))*x^k
    q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3,
     ↪  color=kolor[(i-1)%7], legend_label=r"T( %d )" % i)
show(sin(x),"=",f)
q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3, linestyle="--",
 ↪  figsize=5.5)
show(q)
```

a plot as in Rys. 2.6.6.



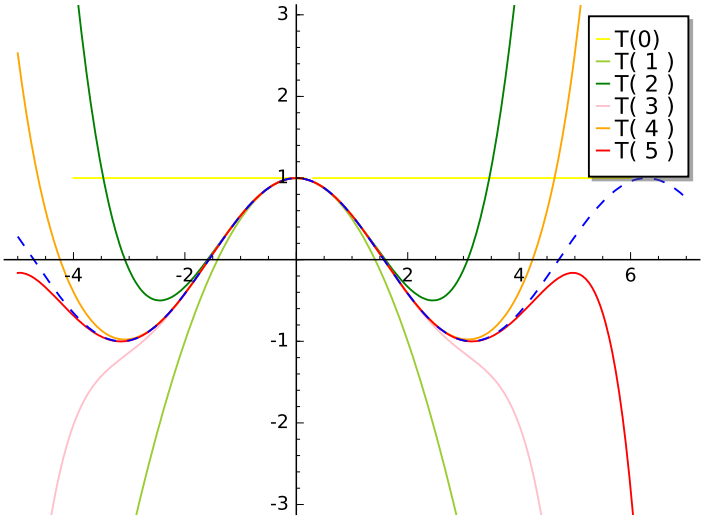
Ćwiczenia dla uczniów.

Dla funkcji $f(x) = cos(x)$ znajdź odpowiadający wielomian ze wzoru Taylora-Maclaurina.

```
kolor=[]
kolor=["yellowgreen","green","pink","orange","red","brown","black"]
n=6
f=1
q=plot(f,xmin=-4,xmax=6, ymin=-3, ymax=3,color="yellow",
 ↪  legend_label="T(0)")
for i in range(1, n):
    k=2*i
    f=f+(-1)^i*(1/factorial(k))*x^k
    q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3,
     ↪  color=kolor[(i-1)%7], legend_label=r"T( %d )" % i)
show(cos(x),"=",f)
```

```
f=cos(x)
q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3, linestyle="--",
 ↪  figsize=5.5)
show(q)
```

a plot as in Rys. 2.6.6.



Zastosuj wzór Taylora-Maclaurina dla funkcji $f(x) = e^x$.

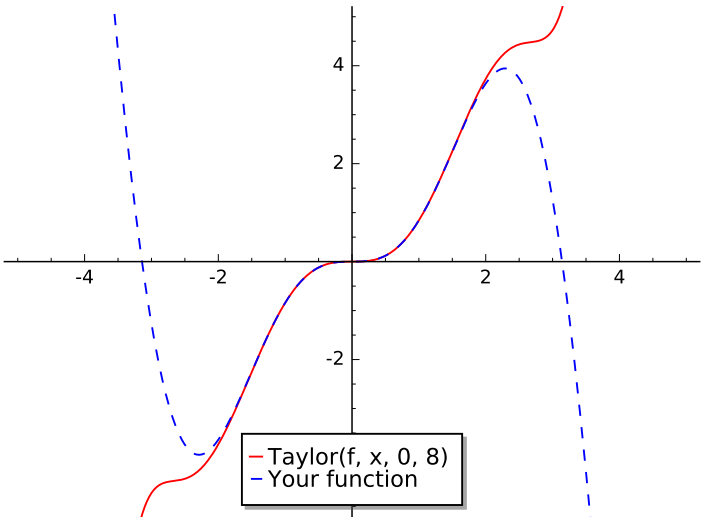
```
kolor=[]
kolor=["yellowgreen","green","pink","orange","red","brown","black"]
n=8
f=1
q=plot(f,xmin=-4,xmax=6, ymin=-3, ymax=3,color="yellow",
 ↪  legend_label="T(0)")
for i in range(0, n):
    k=i+1
    f=f+(1/factorial(k))*x^k
    #print(f(x))
    q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3,
     ↪  color=kolor[(i-1)%7], legend_label=r"T( %d )" % i)
show(e^x,"=",f)
f=e^x
q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=10, linestyle="--",
 ↪  figsize=5.5)
show(q)
```

Znamy już wzór Taylora. Teraz możemy uprościć nasze obliczenia i użyć wbudowanego wzoru Taylora w SageMath.

```
f=sin(x)*x^2           #your function
k=8                    #level iteration
```

```
t=taylor(f,x,0,k)      #Taylor function in Sage
q=plot(t, xmin=-5, xmax=5, ymin=-5, ymax=5, color="red",
 ↪  legend_label=r"Taylor(f, x, 0, %d)" % k)
show(f,"=",t)
q=q+plot(f, xmin=-5, xmax=5, ymin=-5, ymax=5, linestyle="--",
 ↪  figsize=5.5, legend_label=r"Your function")
show(q)
```

a plot as in Rys. 2.6.6.



2.6.7 Wnioski

Zajęcia odbywały się na dodatkowych godzinach w ramach iCSE4school w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie. Celem zajęć było rozszerzenie nauczania matematyki i informatyki w drugiej klasie liceum. Powyższy temat w drugiej grupie testowej był prowadzony metodą „flip teaching", czyli uczniowie musieli się przygotować do zajęć z wykorzystaniem internetu. Pierwsze zajęcia były poświęcone silni i pochodnej funkcji. Drugie zajęcia to wielomiany i wyznaczanie wielomianu przechodzącego przez dane punkty. Według programu nauczania na lekcjach matematyki podobne zadania dotyczą szczególnych przypadków na prostej i paraboli. Ja sam spotkałem się z pytaniami uczniów, czy da się wyznaczyć odpowiednie wzory dotyczące paraboli i czy da się to uogólnić na dowolną ilość punktów. Tak więc powstała idea napisania przeze mnie programu w SageMath, który przy zadanych punktach wyznaczy wielomian przechodzący przez te punkty oraz narysuje to na wykresie. Praca domowa uczniów to zapoznanie się z pojęciem macierzy, mnożeniem macierzy przez wektor i wyznaczaniem jej wyznacznika. Trzecie zajęcia to wyznaczanie przybliżenia funkcji wielomianem przy użyciu wzoru Taylora. Po omówieniu moich przykładów uczniowie mieli w podobny sposób wyznaczyć wielomiany dla podanych funkcji. Jeżeli zauważyli pewną prawidłowość w kolejnych współczynnikach wielomianu to mieli podać hipotezę, a następnie sprawdzić ją w internecie, czy jest ona prawdziwa.

Według mnie zajęcia te mogą być dobrym uzupełnieniem i ugruntowaniem wiedzy uczniów z matematyki w trzeciej klasie liceum na poziomie rozszerzonym lub na zajęciach dodatkowych

w klasie drugiej. Ponadto każdy rozdział można traktować niezależnie, czyli przeprowadzać go w czasie przeprowadzania danego materiału na lekcjach matematyki.

ROZDZIAŁ 3

Fizyka

3.1 Badanie drgań struny

3.1.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenie co najmniej 2h zajęć lekcyjnych z czego:

- 1h w pracowni fizycznej:
  - wykład
  - wykonanie doświadczenia
- 1h lub 2h w pracowni komputerowej:
  - zademonstrowanie odpowiednich metod numerycznych
  - omówienie wyników ćwiczeń i konsultacje

Materiał został opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Adama Ogazę.

3.1.2 Wstęp

Cele lekcji:

- Doświaczalne zbadanie zależności częstotliwości drgań struny od jej długości.
- Użycie Pythona do przeprowadzenia graficznej analizy danych.

Pomoce naukowe:

- Struna rozpieta na linijce/ pudło rezonansowe / dowolny instrument strunowy
- Generator akustyczny
- Głośnik
- Aplikacja mobilna do pomiaru częstotliwości dźwięku
- Aplikacja - generator dźwieku o okreslonej częstotliwości

Użyte metody:

- Wstępne ćwiczenia komputerowe dotyczące tworzenia wykresów w Pythonie oraz dopasowywania prostych i krzywych do danych pomiarowych.
- Wykład jako wstęp teoretyczny.
- Ćwiczenia w grupach - pomiary częstotliwosci drgań struny o różnej długości.

Lekcja przeznaczona jest dla uczniów w wieku 17 lat, zgłębiających fizykę na poziomie rozszerzonym.Spełnia następujące wymagania podstawy programowej:

- Cel ogólny V: “planowanie i przeprowadzanie prostych eksperymentów oraz analizowanie ich wyników”.
- Wymaganie szczegółowe 6.8: “uczeń stosuje w obliczeniach związek między parametrami fali: długością, częstotliwością, okresem, prędkością ”
- Wymaganie szczegółowe 6.12: “uczeń opisuje fale stojące i ich związek z falami biegnącymi przeciwbieżnie”.
- Wymaganie przekrojowe 12.2: “uczeń samodzielnie wykonuje poprawne wykresy”.
- Wymaganie przekrojowe 12.5: “uczeń dopasowuje prostą y = ax + b do wykresu i oblicza współczynniki a i b”.
- Wymaganie doświadczalne 13.6: “uczeń bada drgania striny (na przykład zalezność częstotliwości drgań od długosci struny), wykonuje pomiary, opis, analizę danych i interpretacje graficzną.

Opisywany temat jest przykładem doskonałej integracji nauk ścisłych z informatyką i nauką języka angielskiego. Dlatego musi być przeprowadzony zarówno w pracowni informatycznej, jak i pracowni komputerowej oraz językowej.

3.1.3 Część informatyczna

Część doświadczalna musi zostać poprzedzona podstawowym kursem Pythona, obejmującym rysowanie i formatowanie wykresów oraz dopasowywanie funkcji do danych doświadczalnych. Polska podstawa programowa wymaga jedynie dopasowywania prostych y = ax + b i obliczania współczynników a i b. Python umożliwia łatwe dopasowywanie dowolnej krzywej, co stanowi dobrą okazję do zabawy z hiperbolą. Dodatkowe umiejętności niezbędne do wykonania zadania domowego zadanego pod koniec lekcji eksperymentalnej są następujące: formatowanie tekstu w Sage, wstawianie zdjęć i tabel do notatnika. Zaleca się przeprowadzenie z uczniami nastęujących ćwiczeń:

1. Wstaw dowolne zdjęcie do notatnika:

2. Utwórz i sformatuj tabelę z przykładowymi danymi:

LP.	1	2	3	4	5	6	7	8	9
l	5	10	15	20	30	40	50	70	90
y	1750	879	589	444	309	243	185	122	99

3. Utwórz i ładnie sformatuj na ich podstawie wykres.

```
# Tworzenie wykresu punktowego
l = [5, 10, 15, 20, 30, 40, 50, 70, 90]
y = [1750, 879, 589, 444, 309, 243, 185, 122, 99]
point(zip(l,y))
```

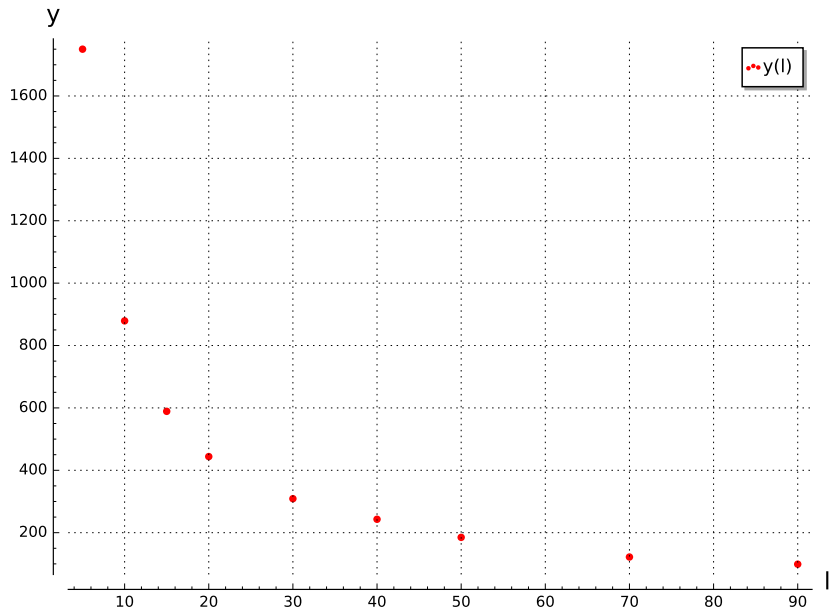
```
# Metody ozdabiania wykresu
pkt=[(l[i],y[i]) for i in range(len(l))]
point(pkt, gridlines=True, size=25, color='red', axes_labels=['l',
 ↪  'y'], legend_label='y(l)')
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.1.

Tabelaryczne wypisanie danych (zamiast drukowania)

```
data = [['l', 'y']]
data.extend(zip(l, y))
table(data)
```

4. Dopasuj hiperbolę do powyższych punktów



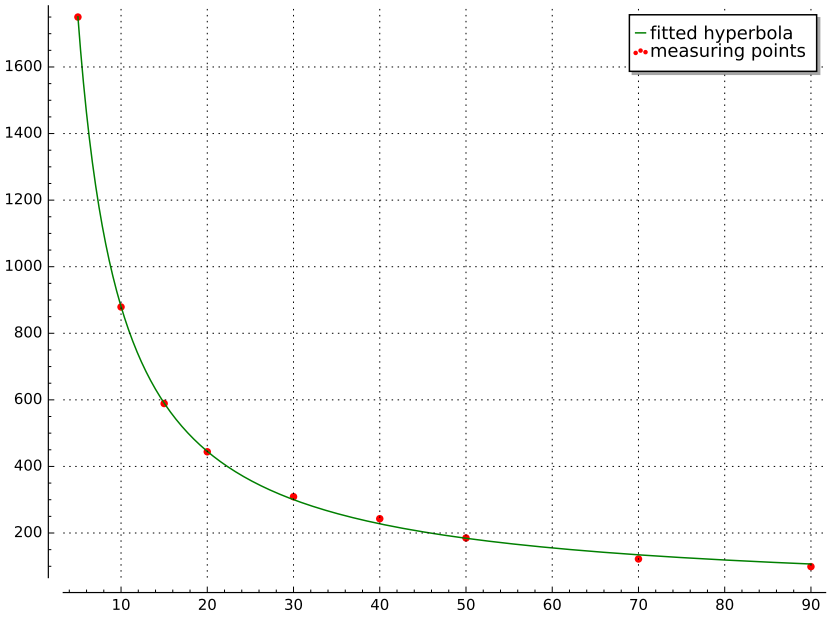
Rys. 3.1: Wykres danych z legendą oraz siatką.

```
# Dopasowywanie hiperboli
var ('a, b')
hyper(x) = a/x+b
fit = find_fit(pkt, hyper,solution_dict=True)
print fit
rys1=plot(hyper.subs(fit), x, 5, 90, color="green",
 ↪ legend_label='fitted hyperbola')
rys2=point(pkt, gridlines=True, size=25, color='red',
 ↪ legend_label='measuring points')
rys1+rys2
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.2.

5. Przyjmij, że l oznacza długość struny, natomiast y to częstotliwość jej drgań. Sporządź wykres T(l), gdzie T jest okresem oraz dopasuj do niego prostą. Zbadaj, czy wymuszenie przejścia prostej przez początek układu współrzędnych wpływa na wartość współczynników w sposób istotny.

```
pktinv=[(l[i],N(1/y[i], digits=4)) for i in range(len(l))]
print pktinv
var ('a, b, c')
straight(x) = a*x+b
straight0(x) = c*x
fit = find_fit(pktinv, straight,solution_dict=True)
print fit
fit0 = find_fit(pktinv, straight0,solution_dict=True)
print fit0
rys1=plot(straight.subs(fit), (x, 0, 90), color="green",
 ↪ legend_label='fitted straight line')
```



Rys. 3.2: Wykres danych doswiadczalnych wraz z dopasowaniem do modelu.

```
11 rys0=plot(straight0.subs(fit0), (x, 0, 90), color="yellow",
    ↪ legend_label='going through 0')
12 rys2=point(pktinv, gridlines=True, size=25, color='red',
    ↪ legend_label='measuring points', axes_labels=['l [cm]','T
    ↪ [s]'])
13 rys1+rys0+rys2
```

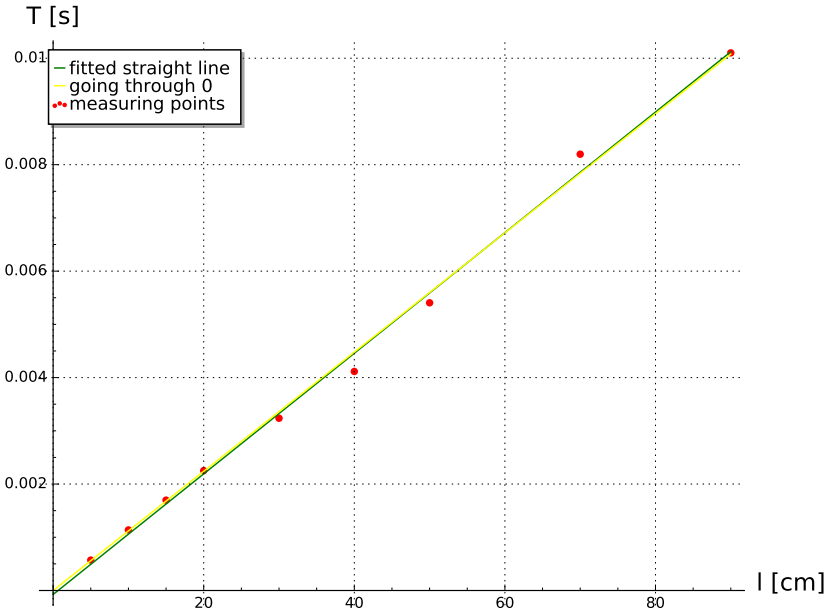
Wynikiem działania powyższego kodu jest wykres Rys. 3.3.

3.1.4 Część doświadczalna

Na początku nauczyciel prosi uczniów o przypomnienie podstawowych faktów dotyczących fal stojacych, częstotliwości, długosci fali i prędkości fazowej. Następnie wyprowadza wzór*f* (l), gdzie l to długość struny. Wzór ten pokazuje, że te dwie zmienne są odwrotnie proporcjonalne. Zbadanie tej zależnosci jest głównym celem niniejszej lekcji.

Następnie uczniowie dzielą się na grupy. Każda grupa wybiera jeden instrument. Może nim być struna rozpięta na linijce lub pudle rezonansowym albo dowolny strunowy instrument muzyczny. Grupy oddalają się maksymalnie, by jak najmniej sobie przeszkadzać.

W każdej grupie ktoś odpowiada za wprawianie struny w drgania. Jednocześnie ktoś inny generuje dźwięk za pomocą aplikacji mobilnej lub komputerowej. Ma ona możliwość płynnej zmiany częstotliwości. Gdy grupa uzna, że aplikacja „stroi”, czyli wydaje dźwięk taki sam jak struna, grupa odnotowuje długość struny i wyświetlaną częstotliwość. Pomiar powtarzany jest dla różnych długości oscylatora w najszerszym możliwym zakresie. Alternatywnie, częstotliwość drgań struny może być mierzona bezpośrednio przez odpowiednią aplikacje na smartfonie.



Rys. 3.3: Wykres okresu drgań od długości struny wraz z dopasowaniem do modelli $y = ax+b$ i $y = ax$.

Wszystkie zebrane dane są zapisywane w tabeli w notatniku. Uczniowie są zachęcania do robienia zdjęć układowi pomiarowemu. Nauczyciel zapowiada, że zebrane dane oraz zdjęcia będą potrzebne do wykonania pracy domowej.

Zadanie domowe

Napisz w Sage krótkie sprawozdanie o przeprowadzonym eksperymencie, zawierające opis istoty problemu, układu pomiarowego (ze zdjęciem), użytych narzędzi, wykonanych czynności, uzyskanych wyników oraz wnioski. W szczególności sprawozdanie powinno potwierdzić lub obalic hipotezę, że częstotliwość drgań jest odwrotnie proporcjonalna do długości struny.

Dane są wspólne dla całej grupy, niemniej sprawozdania musza być napisane niezależnie i indywidualnie. Autorzy najlepszych prac zyskują prawo do ich przetłumaczenia na angielski i opublikowania za dodatkowe punkty. Nauczyciele języka angielskiego są gotowi do pomocy i nadzoru w procesie tłumaczenia

Uwagi o realizacji

Pierwszy raz niniejszą lekcję przeprowadzono w maju 2015. Wszyscy uczniowie wykonali te same pomiary. cała grupa (14 osób) widoczna jest na zamieszczonym zdjęciu. Wprawdzie wszystkie sprawozdania oparte są na tych samych danych, lecz poprosiłem o pracę indywidualną i napisanie w domu unikalnych sprawozdań. Faktycznie, przedstawione do oceny prace różniły sie poziomem i użytymi środkami. Uczniowie zazwyczaj przyznawali, że użycie Sage bardzo im pomogło. Odkryli w tym środowisku wygodne narzędzie do realizacji podobnych zadań.

W pierwszym roku realizacji projektu nie było mozliwości tłumaczenia prac na język angielski. Wpadłem na ten pomysł dopiero rok później.

Druga edycja lekcji *Badanie drgań struny* została przeprowadzona 5 kwietnia 2016 (obie części - informatyczna i fizyczna). Uczniowie zostali podzieleni na 5 grup 4-osobowych. Jedna z uczennic przyniosła własne skrzypce, ktoś inny gitarę. Pozostałe grupy zostały wyposażone w instrumenty z mojego laboratorium, tzn strunę na pudle rezonansowym oraz dwie w struny rozpięte na linijce. Wszystkie grupy dysponowały generatorami akustycznymi bądź mobilnymi aplikacjami do pomiaru dominującej częstotliwości odbieranego dźwięku. Niektóre grupy pozostały w klasie, zaś inne wyszły na korytarz, by przeprowadzić badania daleko od zakłóceń powodowanych przez inne grupy.

Po dokonaniu wstępnych obliczeń okazało się, że 4 grupy odniosły sukces w pomiarach, natomiast jedna napotkała zakłocenia ze strony dźwięków wytwarzanych przez sąsiednią grupę. Poprosili mnie o możliwość powtórzenia pomiarów na zajęciach pozalekcyjnych. Ostatecznie wszyscy uczniowie otrzymali sensowne wyniki i napisali całkiem dobre sprawozdania.

Po dwóch tygodniach spotkalismy sie ponownie w pracowni komputerowej. Do tego czasu dokonałem wstępnej oceny prac. Uzasadniłem swoje oceny oraz wskazałem, co można było poprawić. Po upływie tygodnia dokonałem ostatecznej oceny. Pięć prac uzyskało najwyższą notę i przeszło do następnego etapu. Ich autorzy otrzymali przywilej przetłumaczenia ich na angielski i opublikowania za dodatkowe punkty. Wcześnierj poprosiłem anglistów o współpracę. Nadzorowali oni tłumaczenia, a także postawili własne oceny ze swojego przedmiotu. Wsszystkie wyróżnione prace zostały opublikowane na serwerze sage01.

Lekcja odniosła wielki sukces. Bardzo spodobała się uczniom. Zdołałem zintegrować fizykę nie tylko z informatyką, ale również z językiem angielskim. Językowcy docenili moją inicjatywę i przyznali, że było to ciekawe doświadczenie zarówno dla nich, jak i dla uczniów. Uczą oni przedmiotu o nazwie *język angielski techniczny dla inżynierów,* a nasza współpraca przyniosła żywe, praktyczne zastosowanie tego przedmiotu.

Nie będę opisywać indywidualnych wniosków uczniów, ponieważ są one wszystkie zawarte w opublikowanych oryginalnych pracach.

Ostatnia edycja tej lekcji (kwiecień 2017) odniosła największy sukces. Kolejny rocznik uczniów posiadł umiejętność dodawania do wykresu prostokątów błędów przy użyciu biblioteki numpy. Niektórzy uczniowie przyniesli na lekcję swój profesjonalny sprzęt muzyczny i przeprowadzili naprawdę poważne badania naukowe. Podobnie jak rok wcześniej, zakwalifikowałem najlepsze prace do tłumaczenia, z którego uczniowie wywiązali się doskonale. Najlepsze prace domowe zawarte są w oddzielnych plikach.

Praca 1, wersja polska: https://sage01.icse.us.edu.pl/home/pub/184/

Praca 1, wersja angielska: https://sage01.icse.us.edu.pl/home/pub/179/

Praca 2, wersja polska: https://sage01.icse.us.edu.pl/home/pub/170/

Praca 2, wersia angielska: https://sage01.icse.us.edu.pl/home/pub/172/

3.2 Fale dźwiękowe

3.2.1 O scenariuszu

Scenariusz ten jest materiałem do przeprowadzenie co najmniej 2h zajęć lekcyjnych z czego:

- 1h w pracowni fizycznej:
  - wykład
  - wykonanie doświadczenia
- 1h lub 2h w pracowni komputerowej:
  - zademonstrowanie odpowiednich metod numerycznych
  - omówienie wyników ćwiczeń i konsultacje

Materiał został opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Adama Ogazę.

Niniejsza lekcja została przetestowana trzykrotnie na uczniach zgłębiających fizykę na poziomie rozszerzonym (wiek 17 lat). W roku 2015 liczebność grupy wynosiła 14 osób, w 2016: 21 osób, a w roku 2017: 16 osób. Lekcja została poprzedzona kursem podstaw programowania w języku Python i składa się z 2 części: wstępu teoretycznego i warsztatów komputerowych. Obie części zostały sfilmowane w roku 2015 i opublikowane na Youtube z angielskimi napisami.

3.2.2 Wstęp

Główne cele lekcji są następujące:

- Wyjaśnienie, czym jest dźwięk.
- Wyjaśnienie, czym jest akustyka, na jakie działy się dzieli i dlaczego?
- Zdefiniowanie wszystkich wielkości fizycznych i fizjologicznych opisujących dźwięk.
- Przedstawienie podstawowej wiedzy dotyczącej widma dźwięku.
- Ćwiczenia w rysowaniu wykresów w Pythonie.
- Pokazanie, że każda funkcja okresowa może być przedstawiona jako kombinacja liniowa funkcji sinus.
- Przećwiczenie interaktów i suwaków w Pythonie.

Przygotowując tą lekcję należy wziąć pod uwagę następujące okoliczności:

- Na poziomie szkoły średniej uczniowie nie potrafią całkować. Słyszeli już o całkowaniu i jego zastosowaniach, ale przeprowadzanie obliczeń na poziomie wymaganym przez transformatę Fouriera daleko wykracza poza ich możliwości. Dlatego rzeczywiste obliczenia należy zastąpić prostą zabawą z amplitudami poszczególnych harmonicznych.

- Jest to pierwszy kontakt uczniów z takimi pojęciami jak @interact i suwak. Problemy techniczne same w sobie są trudne do przezwyciężenia, dlatego nie ma sensu przesadnie komplikować fizycznej strony zagadnienia.
- Niniejsza lekcja (podobnie jak wszystkie inne) powinna być atrakcyjna, dlatego należy dążyć do uzyskiwania spektakularnych rezultatów możliwie prostymi środkami.
- Poziom trudności przykładów powinien stopniowo wzrastać w miarę upływu lekcji. Dobrym pomysłem jest pokazywanie alternatywnych rozwiazań tego samego problemu.

3.2.3 Część teoretyczna

Ćwiczenia w programowaniu zostały poprzedzone lekcją teoretyczną o falach dźwiękowych, opublikowaną pod adresem: https://youtu.be/dp-ajKHs6WU

Główne problemy dyskutowane na wykładzie są następujące:

- Definicja fali dźwiękowej.
- Zapowiedź, że pewne idee przedstawione na wykładzie zostaną później rozwinięte na zajęciach komputerowych z użyciem Sage i Pythona.
- Definicja akustyki i wyjaśnienie istoty akustyki fizycznej i fizjologicznej.
- Infradźwięki i ultradźwięki.
- Częstotliwość / długość fali i ich związek z wysokością tonu.
- Widmo dźwięku i jego związek z barwą.
- Prawo Webera - Fechnera.
- Natężenie dźwięku i jego związek z głośnością (audiogram).
- Faza dźwięku i jej związek z wrażeniami przestrzennymi.

Podczas lekcji wykorzystywano komputerowy generator akustyczny (program dołączony do jednego z podręczników) do przedstawiania związku wysokości tonu z częstotliwością. Posiadał również opcje demonstrowania, jak zmiana kształtu sygnału wpływa na widmo i barwę dźwięku. Uczniowie zostali poinformowani, że będą używać Pythona do ilustrowania kształtu sygnału złożonego, w zależności od amplitud indywidualnych harmonicznych.

3.2.4 Część informatyczna

Część informatyczna lekcji została przeprowadzona w pracowni komputerowej, sfilmowana i umieszczona pod adresem: https://youtu.be/0fVgRy6CpWQ

Film ten, nakręcony w roku 2015, pokazuje wcześniejszą wersję programu. Poniżej przedstawiona jest wersja udoskonalona.

```
1 wall = 20 # granica dziedziny = koniec obszaru dostępnego dla fali
```

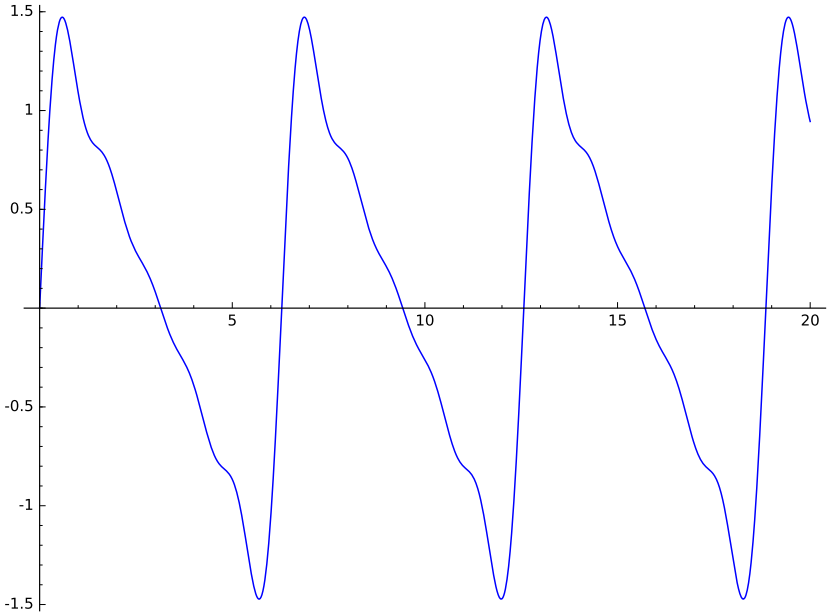
Przykładowa funkcja złożona, zdefiniowana w sposób statyczny

```
amplitudes = (1, 1/2, 3/10, 1/5, 1/10)
WaveComplex(t) = sum(a*sin((n+1)*t) for n, a in
 ↪ enumerate(amplitudes))
WaveComplex(t)
```

```
1/10*sin(5*t) + 1/5*sin(4*t) + 3/10*sin(3*t) + 1/2*sin(2*t) +
sin(t)
```

```
plot(WaveComplex, (t, 0, wall))
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.4.



Rys. 3.4: Wykres.

Funkcja ta sama, jak wyżej, lecz uzyskana z użyciem innych środków.

```
def WaveComplexPlot(amplitudes=(1, 0.5, 0.3, 0.2, 0.1), tmin=0,
 ↪ tmax=20, **kwargs):
    WaveComplex(t) = sum(a*sin((n+1)*t) for n, a in
     ↪ enumerate(amplitudes))
    plt = plot(WaveComplex, (t, tmin, tmax), **kwargs)
    show(plt)
WaveComplexPlot(tmax=wall, figsize=(6, 3))
```

Inna wersja tej samej funkcji, lecz teraz poszczególne amplitudy są sterowane za pomocą suwaków.

Z uwagi na to, że uczniowie pracowali z wcześniejszą wersją Sage, nie mogliśmy użyć gotowej funkcji histogram. W zamian zaproponowałem jej własną wersję.

```
def WaveComplexPlot(A1=1, A2=0.5, A3=0.3, A4=0.2, A5=0.1, **kwargs):
    WaveComplex(t) = A1*sin(t) + A2*sin(t*2) + A3*sin(t*3) +
     ↪ A4*sin(t*4) + A5*sin(t*5)
    return plot(WaveComplex, t, 0, wall,**kwargs)
WaveComplexPlot(figsize=(6,3))
```

```
@interact
def _(A1_=slider(0,1,0.01), A2_=slider(0,1,0.01),
 ↪ A3_=slider(0,1,0.01), A4_=slider(0,1,0.01),
 ↪ A5_=slider(0,1,0.01)):
    plt = WaveComplexPlot(A1=A1_,A2=A2_, A3=A3_, A4=A4_, A5=A5_,
     ↪ figsize=(8,3))
    show(plt)
    histogram = line([(1,0), (1,A1_)], thickness=10) + line([(2,0),
     ↪ (2,A2_)], thickness=10) + line([(3,0), (3,A3_)],
     ↪ thickness=10) + line([(4,0), (4,A4_)], thickness=10) +
     ↪ line([(5,0), (5,A5_)], thickness=10,figsize=(8,3))
    show(histogram)
```

Wynikiem działania powyższego kodu jest program interaktywny Rys. 3.5.

Dekompozycja przykładowej funkcji z użyciem transformaty Fouriera.

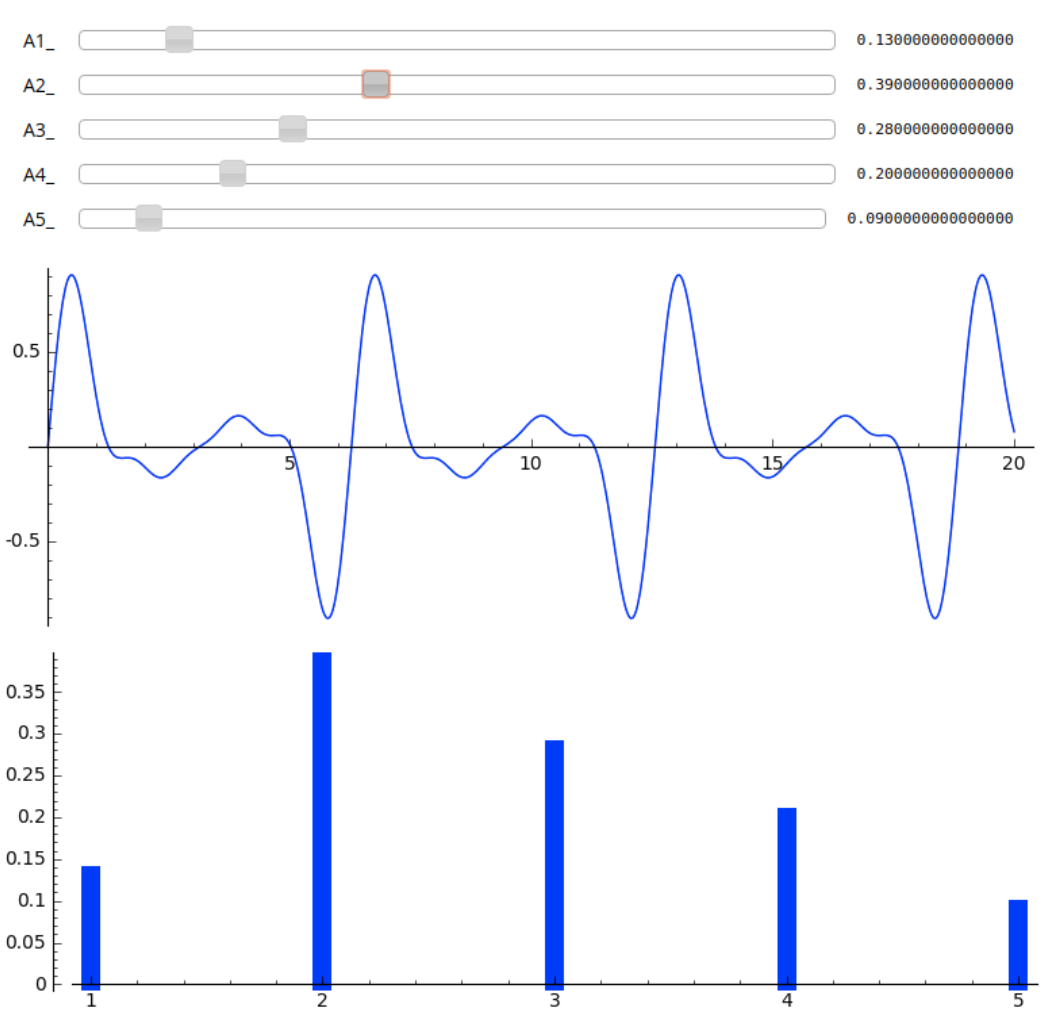
```
f(t) = sum(sin(n*t)/n for n in range(1, 6))
f = Piecewise([[(0, 2*pi), f]])
show(f.plot(), figsize=(4, 2))
sine_coeffs = [N(f.fourier_series_sine_coefficient(i, pi), digits=8)
 ↪ for i in range(20)]
show(bar_chart(sine_coeffs), figsize=(4, 2))
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.6.

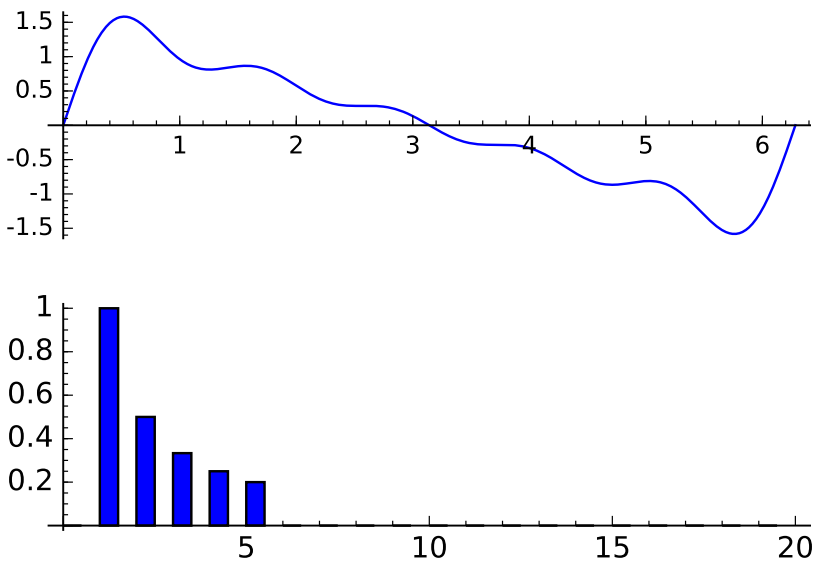
```
Piecewise?
```

Podobna analiza zastosowana dla funkcji piłokształtnej i pozostałych funkcji dostępnych w generatorze akustycznym pokazanym na filmie.

```
sawtooth(t) = (pi-t)/2
sawtooth = piecewise([[(0, 2*pi), sawtooth]])
show(sawtooth.plot(), figsize=(4, 2))
sine_coeffs = [N(sawtooth.fourier_series_sine_coefficient(i, pi),
 ↪ digits=8) for i in range(20)]
show(bar_chart(sine_coeffs), figsize=(4, 2))
```



Rys. 3.5: Dekompozycja sygnału z użyciem transformaty Fouriera.



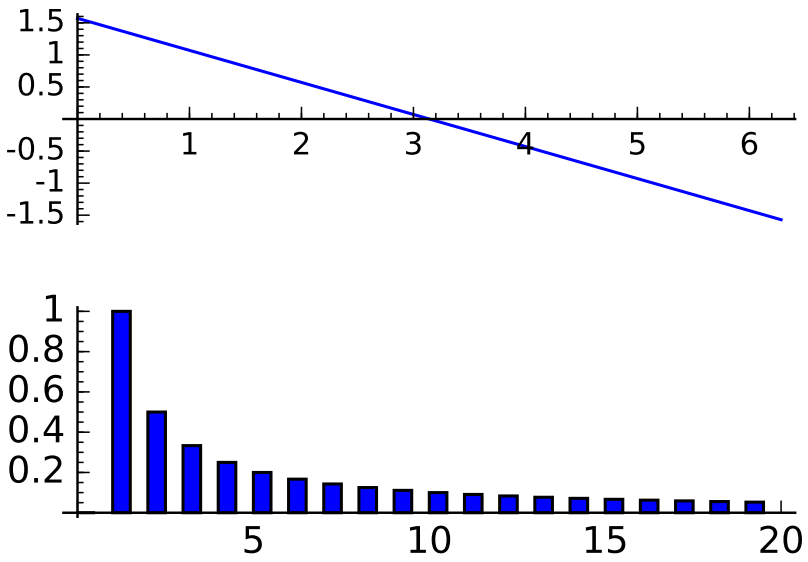
Rys. 3.6: Dekompozycja sygnału typu „piła” z użyciem transformaty Fouriera.

Wynikiem działania powyższego kodu jest wykres Rys. 3.7.

```
triangle1(t) = pi/4*t
triangle2(t) = pi/4*(pi/2-(t-pi/2))
triangle3(t) = pi/4*((t-2*pi))
triangle = Piecewise([[(0, pi/2), triangle1],
                      [(pi/2, 3*pi/2), triangle2],
                      [(3*pi/2, 2*pi), triangle3]])
show(triangle.plot(), figsize=(4, 2))
sine_coeffs = [N(triangle.fourier_series_sine_coefficient(i, pi),
 ↪ digits=8) for i in range(20)]
show(bar_chart(sine_coeffs), figsize=(4, 2))
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.8.

```
upper(t) = 1
lower(t) = -1
rectangle = Piecewise([[(0, pi), upper],
                       [(pi, 2*pi), lower]])
show(rectangle.plot(), figsize=(4, 2))
sine_coeffs = [N(rectangle.fourier_series_sine_coefficient(i, pi),
 ↪ digits=8) for i in range(20)]
show(bar_chart(sine_coeffs), figsize=(4, 2))
```



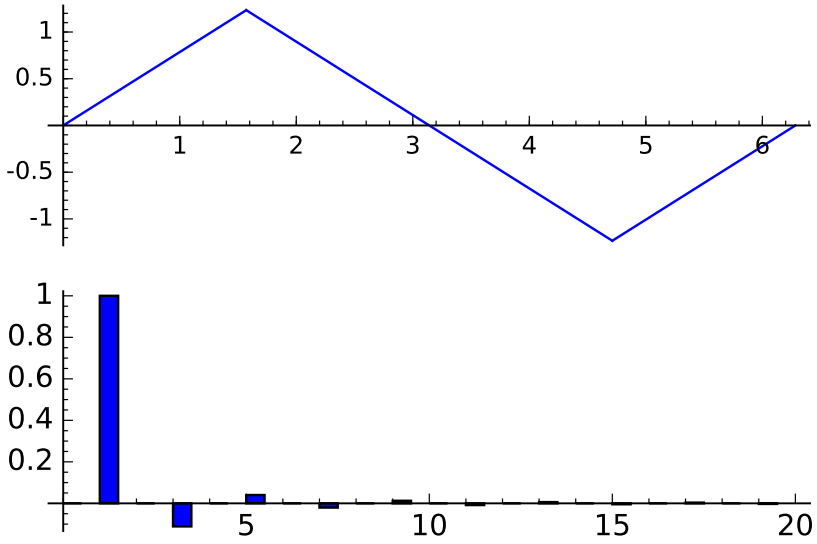
Rys. 3.7: Dekompozycja sygnału trójkątnego z użyciem transformaty Fouriera.

Z użyciem elementu `@interact`, powyższe segmenty kodu można połączyć jak poniżej. Teraz uczniowie mogą dokonywać prób z różnymi postaciami sygnału:

```
def pw_sawtooth():
    sawtooth(t) = (pi-t)/2
    return Piecewise([[(0, 2*pi), sawtooth]])

def pw_triangle():
    triangle1(t) = pi/4*t
    triangle2(t) = pi/4*(pi/2-(t-pi/2))
    triangle3(t) = pi/4*((t-2*pi))
    return Piecewise([[(0, pi/2), triangle1],
                      [(pi/2, 3*pi/2), triangle2],
                      [(3*pi/2, 2*pi), triangle3]])

def pw_rectangle():
    upper(t) = 1
    lower(t) = -1
    return Piecewise([[(0, pi), upper],
                      [(pi, 2*pi), lower]])
@interact
def fourier_sine_trafo(signalname=selector(['sawtooth', 'triangle',
  ↪  'rectangle'])):
```



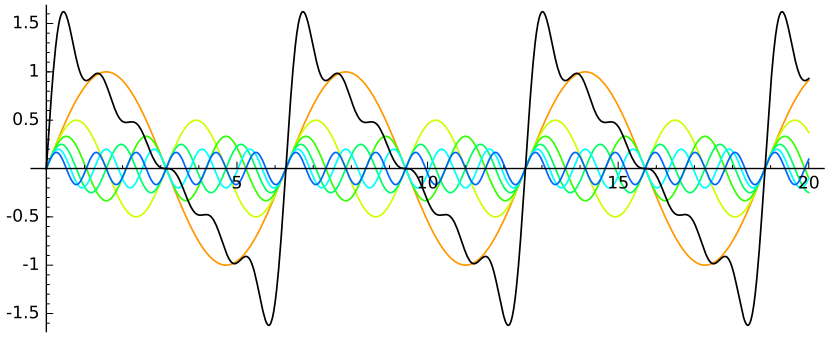
Rys. 3.8: Dekompozycja sygnału prostokatnego z użyciem transformaty Fouriera.

```
    signaldict = {'sawtooth': pw_sawtooth,
                  'triangle': pw_triangle,
                  'rectangle': pw_rectangle}
    signal = signaldict[signalname]()
    show(signal.plot(), figsize=(4, 2))
    sine_coeffs = [N(signal.fourier_series_sine_coefficient(i, pi),
     ↪ digits=8) for i in range(20)]
    show(bar_chart(sine_coeffs), figsize=(4, 2))
```

Sterowaniu może też podlegać ilość iteracji. W poniższym przykładzie, pętla nakłada na siebie wykresy funkcji i wyświetla ich sumę wraz ze składnikami:

```
@interact
def _(n=slider(1, 10, 1)):
    plt = sum(plot(sin(i*t)/i, (t, 0, wall), color=hue(i/10)) for i
     ↪ in range(1, n+1))
    plt = plt+plot(sum(sin(i*t)/i for i in range(1, n+1)), (t, 0,
     ↪ wall), color='black')
    show(plt)
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.9.

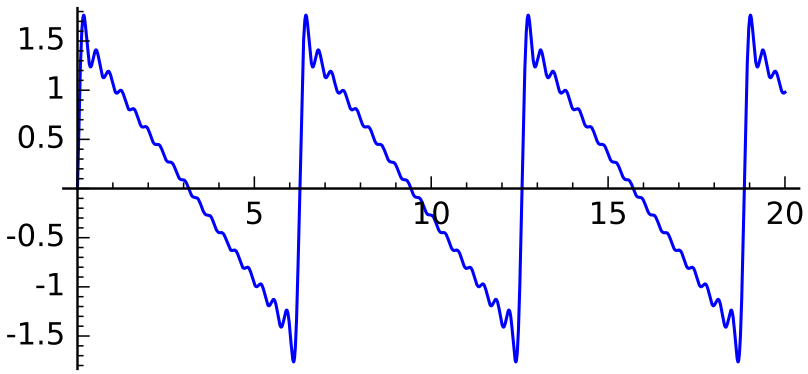


Rys. 3.9: Sumą fal wraz ze składnikami otrzymana w elemencie interaktywnym z $n = 6$

Poniższa pętla tworzy falę złożoną, zbudowaną z harmonicznych o amplitudach odwrotnie proporcjonalnych do ich częstotliwości. Liczba iteracji jest sterowana suwakiem. Funkcje pokazane wyżej są do siebie dodawane, a wyświetlana jest ich suma.

```
@interact
def _(n=slider(1, 20, 1)):
    wave(t) = sum(sin(i*t)/i for i in range(1, n+1))
    plot(wave, (t, 0, wall), figsize=(4, 2)).show()
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.10.



Rys. 3.10: Sumą fal o amplitudach odwrotnie proporcjonalnych do ich częstotliwości wraz ze składnikami otrzymana w elemencie interaktywnym dla $n = 17$

Ten sam efekt uzyskany bez iteracji.

```
def WaveCmplx(t):
        w=0
        for i in range(1,10):
            w=w+1/i*sin(i*t)
        return w
plot(WaveCmplx,  (t, 0, wall), figsize=(4,2))
```

3.2.5 Wnioski

Użycie Pythona doprowadziło uczniów do lepszego zrozumienia istoty widma dźwięku. W szkole średniej uczniowie nie znaja transformaty Fouriera - temat ten wykracza daleko poza podstawę programową. Dzięki powyższym programom, w zasadzie bawiąc się, dokonują wielkiego odkrycia, że każda funkcja okresowa o poprawnej symetrii (co ma miejsce w kontekście fal akustycznych), może zostać przedstawiona jako kombinacja liniowa funkcji sinus.

Z drugiej strony, uczestnicy lekcji mieli możliwość nauczenia się podstawowych metod tworzenia wykresów w sposób interaktywny, co było dla nich nowością. Temat „widmo dźwięku" stał sie mniej abstrakcyjny, ponieważ uczniowie bawili sie nim własnymi rękami.

Z moich obserwacji wynika, że uczniowie z radością używają gotowych narzędzi zawartych w tym dokumencie, natomiast nie są zdolni do ich twórczego rozwinięcia. Moje próby zachęcenia uczniów do udoskonalenia powyższego kodu nie powiodły się.

3.3 Zjawiska akustyczne

3.3.1 O scenariuszu

Materiał przedstawiony poniżej może być zrealizowany podczas 2 godzin lekcyjnych: jedna w pracowni fizycznej, druga w sali zaopatrzonej w wystarczającą ilość komputerów.

Scenariusz został opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Adama Ogazę.

3.3.2 Wstęp

Główne cele lekcji są następujące:

- Wyjaśnienie, czym jest echo i pogłos.
- Zademonstrowanie rezonansu akustycznego.
- Wyjaśnienie zasady działania instrumentów muzycznych.
- Wyjaśnienie, czym są dudnienia,
- Ćwiczenia w tworzeniu wykresów w Pythonie.
- Ćwiczenia w uzyciu interaktów i suwaków w Pythonie.
- Ćwiczenia w tworzeniu animacji w Sage

Uwarunkowania, które trzeba uwzględnić:

- Uczniowie posiadają już podstawową wiedzę o falach w ogólności. W szczególnosci znają równanie fali, koncepcję fali stojącej i rezonansu.
- Jest to drugi kontakt uczniów z animacjami w Pythonie. Uczestniczyli już w lekcji informatyki specjalnie poświęconej temu zagadnieniu.
- Należy pamiętać, że uczniowie nie znają matematyki wyższej. Prezentowane rozwiązania programistyczne nie mogą być zbyt wysublimowane.
- Zalecana jest gradacja trudności. Należy rozpocząć od prostych przykładów i stopniowo je komplikować (na przykład przez dodawanie nowych parametrów lub uzmiennianie stałych).

3.3.3 Część teoretyczna

Ćwiczenia w programowaniu zostały poprzedzone lekcją teoretyczną o zjawiskach akustycznych. Wykład został sfilmowany i opublikowany po polsku, ale z napisami w języku angielskim.

https://youtu.be/jWGTTD5-mFA

Główne zagadnienia dyskutowane na wykładzie:

- Echo
- Pogłos
- Rezonns akustyczny
- Fale stojące (w szczególności w instrumentach)
- Dudnienia

3.3.4 Część informatyczna

Zasadnicze umiejętnosci ćwiczone na lekcji:

- Sterowanie zjawiskami za pomocą suwaków.
- Wprawianie wykresów w ruch.
- Badanie ruchu impulsu falowego.
- Składanie drgań.
- Badanie fal stojących i dudnień.

Problemy fizyczne dyskutowane w poniższym kodzie i wykresach są następujące:

- Dudnienia - jak kształt fali złożonej zależy od częstotliwości fal składowych
- Jak dokonać animacji impulsu falowego i jego odbicia.
- Pokazanie, że fala stojąca jest naprawdę rezultatem nałożenia się na siebie dwóch fal biegnących w przeciwne strony.

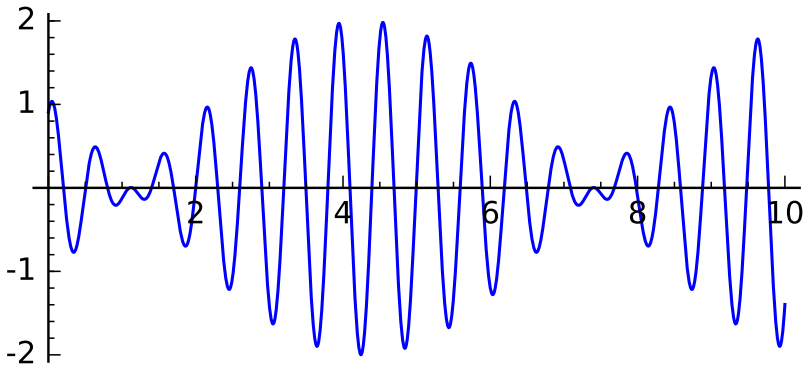
Dudnienia

Prosty przykład wykresu z ustalonymi częstotliwościami i fazami początkowymi, które mogą być zmieniane wewnątrz kodu. Rezultaty tych zmian obserwujemy na wykresie.

```
# **kwarg pozwala na dodanie kolejnych "nazwanych" parametrów
# dudnienia to funkcja zwracająca wykres nałożonych na siebie
  ↪ funkcji y1 i y2
def dudnienia(omega1=10,omega2=11,A1=1,A2=1,fi=2,t0=0,**kwarg):
    y1(t)=A1*sin(omega1*(t-t0))
    y2(t)=A2*sin(omega2*(t-t0)+fi)
    y(t)=y1(t)+y2(t)
    return   plot(y, (t, 0, 10),ymax=A1+A2,ymin=-(A1+A2),**kwarg )
dudnienia(figsize=(4,2))
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.11.

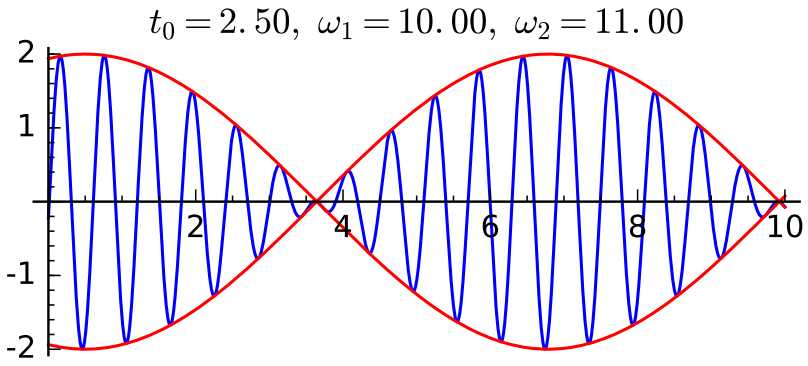
Bardziej złożony wykres zawierający parametry funkcji i obwiedni. Dla prostoty wszystkie amplitudy sa równe.



Rys. 3.11: Wykres nałożonych na siebie funkcji $y_1(t)$ i $y_2(t)$.

```
def envelope(omega1, omega2, a, phase, t0):
    f(t) = 2*a*cos((omega1-omega2)*(t-t0)/2-phase/2)
    return f
def beat(omega1=10, omega2=11, a=1, phase=2, t0=0,
 ↪ plot_envelope=True, **kwargs):
    y1(t) = a*sin(omega1*(t-t0))
    y2(t) = a*sin(omega2*(t-t0)+phase)
    y(t) = y1(t)+y2(t)
    title = '$t_0 = %4.2f,\ \omega_1 = %5.2f,\ \omega_2 = %5.2f$'%\
        (t0, omega1, omega2)
    range = (t, 0, 10)
    plt = plot(y, range, ymin=-2*a, ymax=2*a, title=title, **kwargs)
    if plot_envelope:
        envelope_func = envelope(omega1, omega2, a, phase, t0)
        plt = plt+plot(envelope_func, range, color='red', **kwargs)
        plt = plt+plot(-envelope_func, range, color='red', **kwargs)
    return plt
beat(t0=2.5, figsize=(4, 2))
```

Wynikiem działania powyższego kodu jest wykres `akustyka_envelope`.



Rys. 3.12: Wykres nałożonych na siebie funkcji $y_1(t)$ i $y_2(t)$ wraz z ich obwiednią.

Łatwo skonstruowac narzędzie z suwak zmieniającym argument czasowy. Pozwala na ręczne przesuwanie fali.

```
@interact
def _(t0=slider(0, 2*pi, 0.01, label="$t_0$")):
    plt = beat(t0=t0, figsize=(4,2))
    show(plt)
```

Możliwa jest też regulacja częstotliwości drugiej fali. Można prześledzić, że im częstotliwości obu fal mniej się różnią, tym większy okres pulsacji.

```
@interact
def _(t0=slider(0, 10, 0.01, label="$t_0$"),
      omega2=slider(10, 12, 0.01, label="$\omega_2$")):
    plt = beat(t0=t0 , omega2=omega2, figsize=(4,2))
    show(plt)
```

Informacja: Wynikiem działania powyższych kodów są elementy interaktywne, które najlepiej jest samodzielnie wypróbować w wersji online tego podręcznika.

Kolejne obliczenia zbliżają nas krok po kroku do stworzenia animacji.

```
# plts - zbiór wykresów dla argumentu czasowego t0 iterowanego w
 ↪ petli.
plts = [dudnienia(t0=t0_,figsize=(4,2)) for t0_ in
 ↪ srange(0,6.3,0.2)]
# przykładowe dwa wykresy
show(plts[0])
show(plts[10])
# Przygotowanie dla kolejnych obliczeń.
anim = animate(plts)
```

Tablica wykresów fali dla kolejno zmienianego argumentu t0

```
plots = [beat(t0=t0, figsize=(4, 2)) for t0 in sxrange(0, 2*pi,
 ↪ pi/10)]
graphics_array(plots, ncols=4).show()
```

Gotowa animacja. Widzimy sekwencję wykresów dla różnych wartości argumentu czasowego.

```
%time
anim.show()
```

Animacja wskazująca wpływ drugiej częstotliwości na częstość pulsacji.

```
plots = [beat(omega2=omega2, plot_envelope=False, figsize=(4, 2)) \
 for omega2 in sxrange(5, 15, 0.2)]
animate(plots).show()
```

Odbicie i fala stojąca

Fala stojąca jako złożenie dwóch fal biegnących w przeciwne strony. Uczniowie mogą manipulować wartościami wszystkich parametrów (co było przedmiotem ćwiczeń).

```
# Półautomatyczne sterowanie kolorami w pętli
A=1
omega=6
v=13
delay=30
t_max= 7
sum( [plot(A*sin(omega*(t/delay-x/v))+\
 A*sin(omega*(t/delay+x/v)),(x,0,20),figsize=6,color=hue(t/t_max))\
  for t in srange(0,t_max,1.0)] )
```

Dla lepszego umaocznienia ewolucji czasowej fali stojącej, warto zastosować animację. Dodatkowy walor dydaktyczny stanowi wyświetlenie fal biegnących. Zastosowano argument w postaci *x-vt* zamiast *t-x/v* by uniknąć problemów matematycznych dla prędkości zmierzającej do zera.

```
def running_wave(a=1, omega=1, v=10, t=0):
    wave(x) = a*sin(omega*(x-v*t))
    return wave
range = (x, 0, 20)
figsize = (4, 2)
v = 2
plts = [plot(running_wave(v=v, t=t), range, figsize=figsize)
        + plot(running_wave(v=-v, t=t), range, color='green',
          ↪ figsize=figsize)
        + plot(running_wave(v=v, t=t)+running_wave(v=-v, t=t),\
          range, color='red', ymin=-2, ymax=2, figsize=figsize)
        for t in sxrange(0, 2*pi/v, pi/(10*v))]
animate(plts).show()
```

Tablica grafik pomocna do bardziej szczegółowej analizy ewolucji czasowej.

```
graphics_array(plts[0:6], ncols=3).show()
```

Impuls falowy i jego odbicie

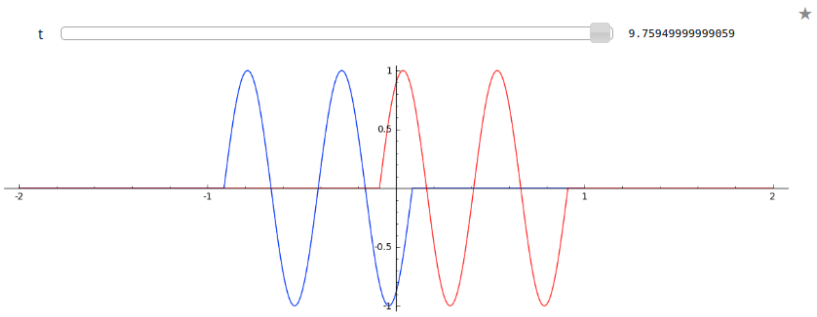
Ruch impulsu falowego. Zdaję sobie sprawę, że tak zdefiniowany impuls jest niefizyczny z powodu nieciągłości w pochodnej, lecz jest to pierwsze podejście do tego typu animacji. Uczniowie zostali poinformowani o watpliwościach natury fizycznej; przykład ma raczej ilustrować jak radzić sobie z takimi animacjami z punktu widzenia samego programowania.

```
def pulse1(x):
    if x>=0 and x<=4*pi:
        return A1*sin(x)
    else:
        return 0.0
```

Dwa impulsy biegnące w przeciwnych kierunkach.

```
var('x')
A1 = 1
c = 1.4
nl = 2
nL = 4
k = 4*pi # 2pi/wavelenght
@interact
def _(t=slider(0,10,0.0001,default=1/c*(nL-nl)*2*pi)):
    x0 = -nL*2*pi/k
    x1 = (nL-nl)*2*pi/k
    plt = Graphics()
    plt += plot( lambda x:pulse1(k*(x-x0)-c*t),(x,x0,1),figsize=(12,4),thickness=1)
    plt += plot( lambda x:pulse1(k*(x-x1)+c*t),(x,x0,2),color='red',thickness=1)
    plt.show()
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.13.



Rys. 3.13: Wykres nałożonych na siebie funkcji $y_1(t)$ i $y_2(t)$ wraz z ich obwiednią.

Złożenie impulsu biegnącego i odbitego.

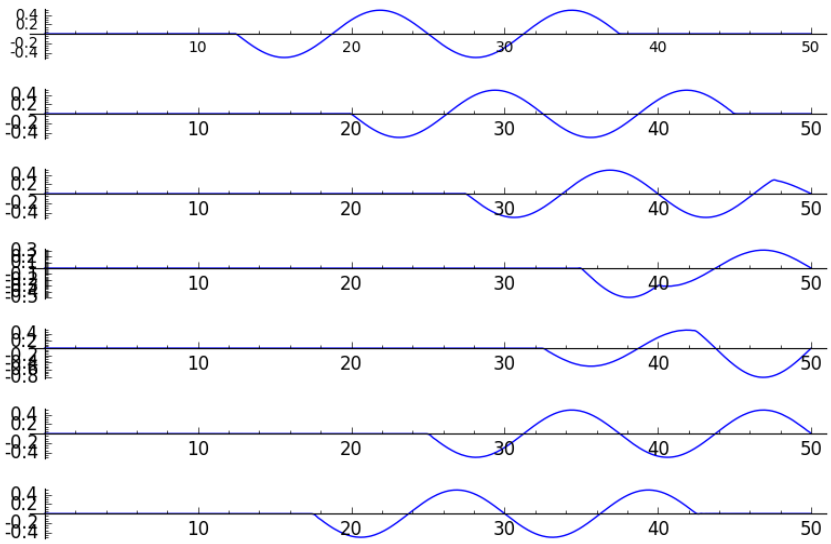
```
var('x')
A1 = 1
```

```
c = 3.4
nl = 2
nL = 4
k = 4*pi # 2pi/wavelenght
@interact
def _(t=slider(0,10,2*pi/k/64)):
    x0 = -nL*2*pi/k
    x1 = (nL-nl)*2*pi/k
    plt = Graphics()
    plt += plot( lambda x:pulse1(k*(x-x0)-c*t)+\
     pulse1(k*(x-x1)+c*t),(x,x0,0),figsize=(12,4),\
     thickness=1,ymin=-2,ymax=2)
    plt.show()
```

Aby precyzyjnie obliczyć przebieg procesu odbicia fali można numerycznie rozwiązać równanie falowe. Poniższy kod demostruje przykład takiego algorytmu opartego o bibliotekę `numpy`.

```
%time
import numpy as np
N = 4048
l = 50.
dx = float(l)/(N-1)
c2 = np.ones(N)
dt = 0.005
print np.sqrt(np.max(c2))*dt/dx
x = np.linspace(0,l,N)
u = np.zeros(N)
u0 = np.zeros(N)
unew = np.zeros(N)
ulst=[u.copy()]
n=4.
T = 1.*l/n
for i in range(25000):
    unew[1:-1] = 2.*u[1:-1] - u0[1:-1] + dt**2
     ↪ *(c2[1:-1]/dx**2*np.diff(u,2))
    u0=u.copy()
    u=unew.copy()
    u[-1] = u[-2]
    u[0] = u[1]
    u[-1] = 0
    u[0] = 0
    if dt*i/T*2.0*np.pi< 4*np.pi:
        u[0] = 0.5*np.sin(dt*i/T*2.0*np.pi)
    if i%50==0:
        ulst.append(u.copy())
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.14.



Rys. 3.14: Odbicie fali biegnącej otrzymane poprzez numeryczne rozwiązanie równania falowego. W wersji interaktywnej dostępne są również animacje tego procesu.

```
@interact
def _(ith=slider(range(len(ulst)))):
    u = ulst[ith]
    plt =  line(zip(x,u),figsize=(12,5),ymin=-1,ymax=1)
    plt.show()
```

```
plts = [line(zip(x,u),figsize=(6,2),ymin=-1,ymax=1) for u in
 ↪ ulst[::8]]
animate(plts).show()
```

Wnioski

Programowanie w Pythonie okazało się interesującym uzupełnieniem lekcji fizyki. Korzyści polegają na możliwości mniej lub bardziej łatwej wizualizacji zjawisk przy zmianie parametrów. Dzięki temu, wzory przedstawione na wykładach stają się mniej abstrakcyjne i każdy mógł się własnoręcznie przekonać, jak wynik np. interferencji fal zależy od ich częstotliwości, kierunku, prędkości itd.

Animacje same w sobie okazały się nowym wyzwaniem dla uczniów. W pierwszych edycjach lekcji prosiłem uczniów o dokonanie prób jakiegokolwiek rozwinięcia przedstawionych idei poprzez ulepszenie kodu. Okazało się to niewykonalne, ponieważ kod jest dla nich zbyt trudny. Jedyne, do czego byli zdolni, to manipulacja wartościami parametrów - a to i tak dużo.

Niemniej przebieg lekcji oceniam wysoko. Uczniowie byli zainteresowani i zaangażowani. Ich kompetencje, zarówno w zakresie fizyki, jak i informatyki, wzrosły.

3.4 Badanie ruchu przyspieszonego

3.4.1 O scenariuszu

Materiał przedstawiony poniżej może być zrealizowany podczas 2 godzin lekcyjnych: jedna w pracowni fizycznej, druga w sali zaopatrzonej w wystarczającą ilość komputerów.

Scenariusz został opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Adama Ogazę.

3.4.2 Wstęp

Celem lekcji jest zbadanie ruchu przyspieszonego, a w szczególności rozstrzygnięcie, czy ruch jest jednostajnie przyspieszony, tzn. czy przyspieszenie jest w nim stałe oraz wyliczenie tego przyspieszenia. Wymaga to przeprowadzenia szeregu pomiarów, najlepiej całych serii, w powtarzalnych warunkach. Python posłuży nam do algebraicznego przetworzenia wielkiej liczby danych i przeprowadzenia analizy graficznej. Koncepcja tematu przewiduje następujące etapy postępowania:

1. Przeprowadzenie lekcji w pracowni fizycznej, obejmującej wykład teoretyczny dotyczący ruchu jednostajnie przyspieszonego oraz demonstrację odpowiedniego doświadczenia.
2. Przeprowadzenie lekcji w pracowni komputerowej na temat graficznej analizy danych: dopasowywania prostych i krzywych do danych pomiarowych i rysowanie prostokątów błędu.
3. Zadanie uczniom pracy domowej polegającej na wykonaniu analogicznego doświadczenia i opracowania wyników z użyciem metod poznanych na lekcji informatyki.

Kolejność dwóch pierwszych punktów może być zamieniona.

Niniejszy dokument zawiera rozważania o charakterze ogólnym. Szczegóły lekcji doświadczalnej zawarte są w załączonym filmie (wyposażonym w napisy angielskie). Przykładowe prace uczniów znajdują się w oddzielnych plikach.

3.4.3 Część doświadczalna

Pomoce naukowe

1. Dowolny układ do powtarzalnego badania ruchu przyspieszonego - najlepiej tor powietrzny nachylony pod niewielkim kątem. Można użyć też zwykłej równi pochyłej i kulki / wózka itp.
2. Przyrząd do pomiaru drogi.
3. Komplet przyrządów do pomiaru czasu - moga to być telefony komórkowe uczniów.

Tok lekcji

Cały przebieg lekcji, opis eksperymentu oraz oczekiwań dotyczących sposobu opracowania wyników znajduje się na filmie:

https://youtu.be/deTJ4i1V0dg

W szczególności przedstawiony w nim został wzór karty pomiarowej, wskazówki dotyczące sposobu przeprowadzenia doświadczenia i omówione zostały formuły niezbędne do obliczeń. Zastosowane metody obliczeniowe w zasadzie wykraczają poza podstawę programową, ale dzięki Pythonowi ich użycie jest bardzo łatwe i uczeń natychmiast uzyskuje spektakularne wyniki bez konieczności dogłębnego rozumienia wszystkich mechanizmów.

Sugerowana metoda doświadczalna polega na zaznaczeniu na równi pochyłej szeregu kresek w ustalonych odległościach od punktu startowego a następnie wykonniu kilku serii pomiarów czasu ruchu od startu do każdej z kresek. Dokładność pomiaru drogi (pomiar jednokrotny) szacujemy metodą działki elementarnej (w warunkach lekcji jest ona zdeterminowana grubością kredy), natomiast czasy dotarcia ciała (w warunkach lekcji - ślizgacza) do konkretnej kreski uśredniamy, a jako niepewność przyjmujemy odchylenie standardowe serii pomiarów czasu dla tej kreski. Niepewności pomiarów złożonych (kwadrat czasu, przyspieszenie) obliczamy za pomocą Uproszczonej Metody Logarytmicznej (UML).

Z wykonanego doświadcznia należy sporządzić sprawozdanie według ogólnych zaleceń, opublikowanych w formie PDF na stronie internetowej szkoły.

3.4.4 Część informatyczna

Poniższa sekcja nie stanowi kompletnego rozwiązania problemu, a jedynie zbiór wskazówek, jak sie pewne rzeczy robi. Resztę pozostawiamy inwencji uczniów. W razie problemów, jako wzór mogą posłużyć przykładowe prace uczniów, dołączone do niniejszej dokumentacji. Wszystkie poniższe dane są zmyślone i mają na celu zaprezentowanie sposobu działania poszczególnych komend.

Przykładowe obliczanie odchylenia standardowego. Obliczanie go „na piechotę” jest dosyć żmudne; w Pythonie sprowadza się do wykonania banalnej instrukcji:

```
 # przykładowe dane
anydata = (3.4, 3.6, 3.3, 3.3, 3.5, 3.7, 3.6, 3.6, 3.6, 3.5)
mean_data = mean(anydata) # wartość średnia
std_data = std(anydata) # odchylenie standardowe
print "Wartość średnia = ", N(mean_data, digits=3)
print "Odchylenie std = ", N(std_data, digits=3)
```

W wyniku działania powyższego kodu otrzymamy:

Wartość średnia = 3.51
Odchylenie std = 0.137

Dane doświadczalne możemy łatwo narysować za pomocą np. `point`. Jeśli danych jest mało to można wpisać je bezpośrednio do tablicy w Python-ie, tak jak poniżej:

```
s = (0, 0.03, 0.1, 0.39, 0.88, 1.62, 2.44, 3.55)
t = (0, 0.5, 1.01, 2.1, 2.97, 3.88, 5.02, 5.95)
delta_s = (0, 0.02, 0.02, 0.02, 0.05, 0.05, 0.05, 0.1)
delta_t = (0.1, 0.1, 0.1, 0.2, 0.2, 0.2, 0.2, 0.3)
pkt = zip(t, s)
point(pkt, gridlines=True, size=25, color='red', axes_labels=['l',
 ↪  's'], legend_label='s(t)')
```

Należy jednak wykonać dopasowanie paraboli do powyższych danych. Bez użycia środowiska Sage jest to dla ucznia czynność absolutnie „praktycznie" niewykonalna. Tutaj sprowadza sie do wykonania kilku prostych instrukcji. Wzór opisujący parabolę został tak sformułowany, by w wyniku otrzymać wprost wartość przyspieszenia. Uczeń nie musi rozumieć, jak to się dzieje z matematycznego punktu widzenia. O tym, że otrzymał właściwy rezultat przekonuje go wykres, możliwie ściśle przylegający do punktów pomiarowych.

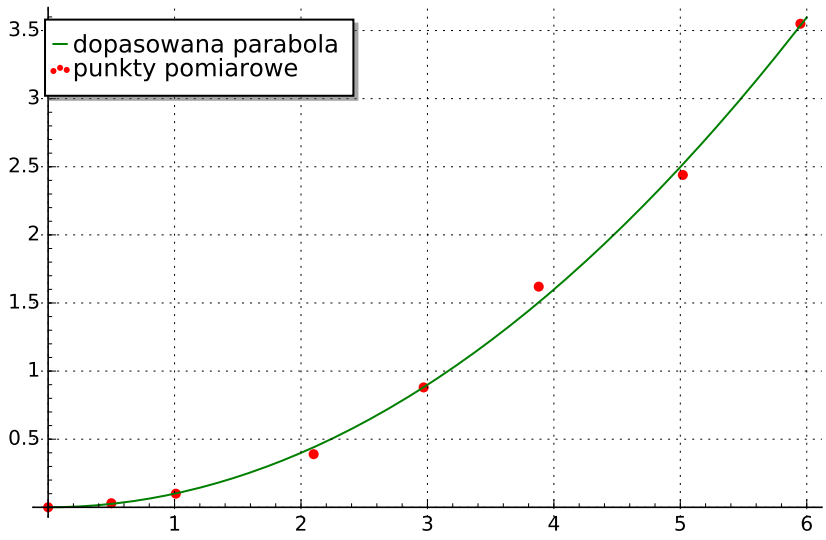
```
var ('a')
parabola(x) = a/2*x^2 # Czas oznaczono jako x, aby uniknąć dalej
 ↪  kolizji oznaczeń
fit = find_fit(pkt, parabola,solution_dict=True)
print fit
rys1=plot(parabola.subs(fit), x, 0, 6, color="green", \
  legend_label='dopasowana parabola')
rys2=point(pkt, gridlines=True, size=25, color='red', \
  legend_label='punkty pomiarowe')
rys1+rys2
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.15.

Środowisko SageMath nie jest wyposażone w dogodną funkcję nanoszącą prostokąty błędów na wykres. Możemy jednak bezpośrednio skorzystać z zawartej w SageMath biblioteki `matplotlib`, która jest z resztą wewnętrznie używana przez Sage do generowania wszystkich wykresów. Punkty połączone są łamaną (fmt=»o-«).

```
import matplotlib.pyplot as plt
plt.clf()
plt.errorbar(t, s, xerr=delta_t, yerr=delta_s, fmt='o-')
plt.xlabel("t [s]")
plt.ylabel("s [m]")
plt.savefig('1.png')
```

Niestety, wspomniana biblioteka nie rysuje wprost wykresów funkcji danych wzorem algebraicznym. Aby umieścić na jednym wykresie prostokąty błędów i dopasowaną parabolę, należy samodzielnie wypróbkować formułe analityczną. Funkcja `np.linspace` wygeneruje nam



Rys. 3.15: Wykres nałożonych na siebie danych i dopasowanej funkcji $y = \frac{at^2}{2}$

100 (w kodzie poniżej) równomiernie oddalonych punktów z przedziału od 0 do ostatniego punktu pomiarowego `t[7]`.

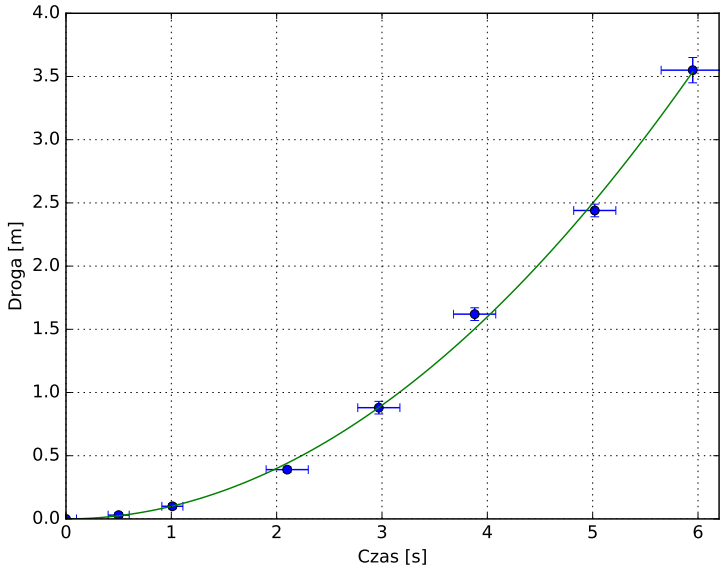
```
import matplotlib.pyplot as plt
import numpy as np
plt.clf()
plt.errorbar(t, s, xerr=delta_t, yerr=delta_s, fmt='o')
plt.xlabel("Czas [s]")
plt.ylabel("Droga [m]")
t_ = np.linspace(0,t[7],100)
plt.plot(t_,a.subs(fit)/2*t_**2)
plt.grid()
plt.xlim(0, 6.2)
plt.ylim(0, 4)
plt.savefig('1.png')
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.16.

3.4.5 Zadanie domowe

Skonstruować dowolny układ do obserwacji ruchu przyspieszonego. Wykonać serie pomiarów czasu przebycia różnych dróg w tym ruchu. Napisać sprawozdanie w notatniku Sage zawierające:

1. Wstęp teoretyczny
2. Opis układu doświadczalnego (ze zdjęciem) i wykonanych czynności
3. Wyniki pomiarów i obliczeń



Rys. 3.16: Wykres nałożonych na siebie danych i dopasowanej funkcji $s = \frac{a}{2}t^2$ wraz z błędami. Wykres wykonany za pomocą biblioteki matplotlib.

4. Graficzną analizę danych - wykresy $s(t)$, $s(t^2)$ i $a(t)$ wraz z prostokątami błędów i dopasowanymi prostymi / krzywymi
5. Dyskusję uzyskanych wyników - należy rozstrzygnąć różnymi metodami, czy przyspieszenie w obserwowanym ruchu było stałe.

Szczegółowe zalecenia dotyczące pisania sprawozdań są uczniom znane, gdyż zostały opublikowane w formie pliku PDF na stronie internetowej szkoły.

3.4.6 Uwagi o realizacji

Niniejszy scenariusz został przetestowany w latach 2015-2017 na trzech rocznikach uczniów klas drugich LO (fizyka, poziom rozszerzony). Wcześniej, przed rozpoczęciem projektu, podobne doświadczenie było już realizowane od wielu lat, ale sprawozdania były oddawane w wersji papierowej. Python ułatwił przeliczanie danych i rysowanie wykresów oraz umożliwił obliczanie parametrów prostych / krzywych dopasowanych do punktów pomiarowych. Wcześniej było to niemożliwe. W odniesieniu do wielkości wprost proporcjonalnych uczeń mógł jedynie przyłożyć linijkę do narysowanych punktów i sprawdzić, czy da się tak poprowadzić odcinek, by przechodził przez wszystkie prostokąty błędów.

Nastawienie uczniów do wymagań projektu było różne. Najgorzej wypadł rocznik środkowy. Była to jedyna klasa o profilu ścisłym, która na skutek zmian w siatce godzin, nie realizowała pełnego rozszerzenia z informatyki. Niektórzy deklarowali jawną niechęć do programowania. Nie przekonywały ich oczywiste korzyści ze stosowania technologii informatycznych. Dlatego też przystałem na rozwiązanie hybrydowe - dokonanie obliczeń w Pythonie i oddanie sprawozdania papierowego (z wykresami Pythona jako załącznikami).

Uczniowie rocznika pierwszego i trzeciego nie mieli żadnych oporów w stosowaniu techno-

logii komputerowych w pełnym wymiarze. Nie widzieli również przeszkody w tym, że do pisania wzorów w notatniku Sage (niezbędnych np. we wstępie teoretycznych) konieczne jest opanowanie Latexa. Nawiązałem współpracę z anglistami i skłoniłem autorów najlepszych sprawozdań do przetłumaczenia ich na język angielski. Za ten wysiłek uczniowie otrzymali dodatkowe punkty zarówno z fizyki, jak i z języka angielskiego.

Linki do przykładowych prac:

Wersja polska: https://sage01.icse.us.edu.pl/home/pub/148/

Wersja angielska: https://sage01.icse.us.edu.pl/home/pub/177/

Cechy dobrego sprawozdania z fizyki: http://3lo.edu.pl/?p=306

3.5 Zderzenia

3.5.1 O scenariuszu

Materiał przedstawiony poniżej może być zrealizowany podczas 2 godzin lekcyjnych: jedna w pracowni fizycznej, druga w sali zaopatrzonej w wystarczającą ilość komputerów.

Przedstawiony w niniejszym dokumencie eksperyment realizowany był na zajęciach pozalekcyjnych i miał status zadania dodatkowego. Zainteresowani uczniowie mogli go przeprowadzić, sfilmować i opisać w formie sprawozdania, za które mogli otrzymać dodatkową ocenę.

Scenariusz został opracowany w ramach projektu iCSE4school na podstawie lekcji prowadzonych w latach 2015-2017 w III Liceum Ogólnokształcącym im. Stefana Batorego w Chorzowie przez Adama Ogazę.

Wartością dodatkową była owocna współpraca z anglistami, pod okiem których autorzy najlepszych prac dokonali ich tłumaczenia. Uczniowie klas ścisłych realizują w naszej szkole dodatkowy przedmiot o nazwie *język angielski dla inżynierów*. Doświadczenie z fizyki stworzyło możliwość przećwiczenia języka technicznego na żywym przykładzie i otrzymania dodatkowej oceny z tego przedmiotu.

3.5.2 Opis problemu

Eksperyment polega na przeanalizowaniu zderzenia centralnego 2 ciał o znanych masach na torze powietrznym. Oryginalne slizgacze na torze wyposażone były w spężyste metalowe zderzaki. Ponieważ zauważyłem, że podczas zderzeń część energii rozprasza się, powodując trwałe odkształcenia zderzaków, zastąpiłem je odpychającymi się magnesami neodymowymi. Można więc oczekiwać, że obecnie zderzenia będą doskonale sprężyste, a więc zachowany będzie zarówno pęd, jak i energia kinetyczna. Tak zaprojektowany eksperyment stwarza również możliwość dokładniejszego zbadania charakteru siły magnetycznej.

Zderzenia trwały około 3 sekund i zostały sfilmowane kamerami HD. Przykładowy film znajduje się pod adresem: *https://youtu.be/JSpKctrX_YM*

Zadaniem uczniów było odczytywanie (klatka po klatce) położenia obydwu ślizgaczy na tle skali toru powietrznego. Ślizgaczy było kilka do wyboru, każdy o innej masie. Zmienne x_1 i x_2 oznaczają współrzędne konców podstawy ślizgaczy zwróconych ku sobie. Przyklejone do ślizgaczy magnesy wystawały o d_1 i d_2, tak więc odległość między ich biegunami była mniejsza od $|x_2 - x_1|$ o $d_1 + d_2$. Zbiór danych eksperymentalnych jest więc następujący:

- Masy ślizgaczy: m_1 i m_2 od około 0,17 kg do ponad 0,4 kg
- Rozmiary magnesów i ich mocowania: d_1 i d_2, około 0,01 m
- Czas był obliczany iteracyjnie i, w zależności od kamery, narastał z krokiem 1/25 s lub 1/30 s
- Współrzędne ślizgaczy $x_1(t)$ i $x_2(t)$ odczytywane z filmu i wyrażone w metrach. Jeśli ślizgacze poruszały się w kierunku malejących wartości na skali, ich prędkości i pędy są ujemne.

Na podstawie powyższych danych należało wykonać i zinterpretować następujące wykresy:

- $x_1(t)$ i $x_2(t)$ - położenia ślizgaczy w funkcji czasu
- $v_1(t)$ i $v_2(t)$ - prędkości chwilowe ślizgaczy w poszczególnych klatkach. Jesli punkty wykresu tworzyły chaotyczną chmurę (zaszumienie danych), radziłem, by uśredniać prędkości z 2 sąsiednich klatek.
- $p_1(t), p_2(t)$ i $p_c(t)$ - pędy poszczególnych ślizgaczy i całego układu. Należało zbadać, czy pęd układu został zachowany.
- $E_1(t), E_2(t)$ i $E_c(t)$ - energie kinetyczne ślizgaczy i ich suma. Należało zbadać, czy energia kinetyczna jest zachowana.
- $a_1(t)$ i $a_2(t)$ - przyspieszenia, jakim podlegają ślizgacze.
- $F_1(t)$ i $F_2(t)$ - siły działające na ślizgacze
- $F(r)$ - zależność siły magnetycznej od odległości między biegunami (z uwzględnieniem rozmiarów wystających z wózków magnesów).

3.5.3 Uwagi o realizacji

Dane pierwszego ucznia

Było to pionierskie wykonanie tego doświadczenia w kwietniu 2016 przez pojedynczego ochotnika z pierwszego rocznika uczniów realizujących projekt.

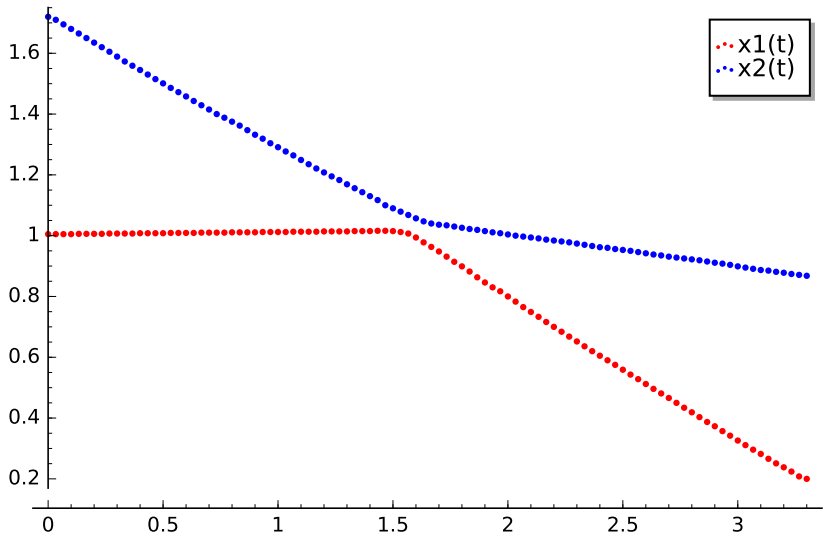
Wyniki pomiarów i wykresy $x_1(t)$ i $x_2(t)$

```
m1 = 0.1793
m2 = 0.3197
d1 = 0.01
d2 = 0.011
delta_t = 1/30
t = [(i*delta_t) for i in range(0,100)]
x1 = [1.005,1.005,1.005,...
x2 = [1.720,1.710,1.695,...
x1t = [(t[i],x1[i]) for i in range(0,100)]
x2t = [(t[i],x2[i]) for i in range(0,100)]
xt = point(x1t,color = "red",legend_label='x1(t)')+\
     point(x2t,color="blue",legend_label='x2(t)')
xt
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.17.

Wykresy $v_1(t)$ i $v_2(t)$

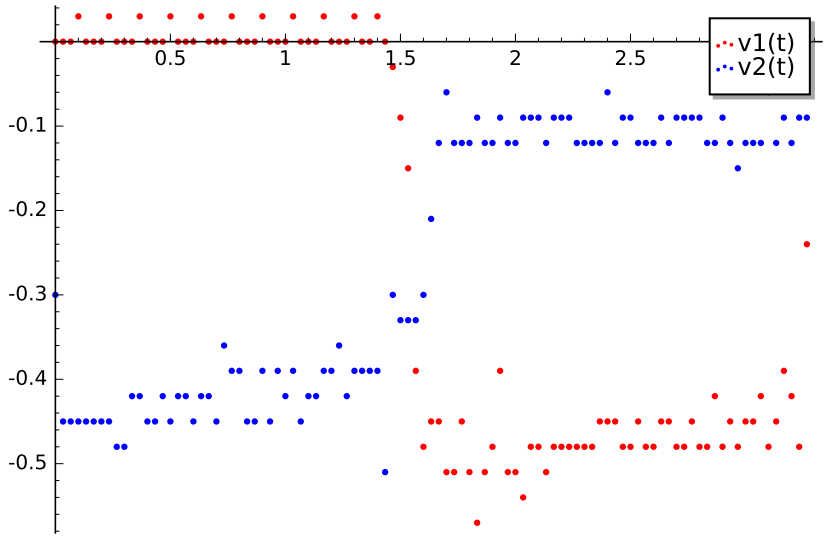
```
v1 = [((x1[i+1]-x1[i])/(delta_t)) for i in range(0,99)]
v2 = [((x2[i+1]-x2[i])/(delta_t)) for i in range(0,99)]
v1t = [(t[i],v1[i]) for i in range(0,99)]
v2t = [(t[i],v2[i]) for i in range(0,99)]
```



Rys. 3.17: Wykres drogi od czasu wykonany z danych doświadczalnych.

```
vt = point(v1t,color="red",legend_label='v1(t)')+\
     point(v2t,color="blue",legend_label='v2(t)')
vt
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.18.

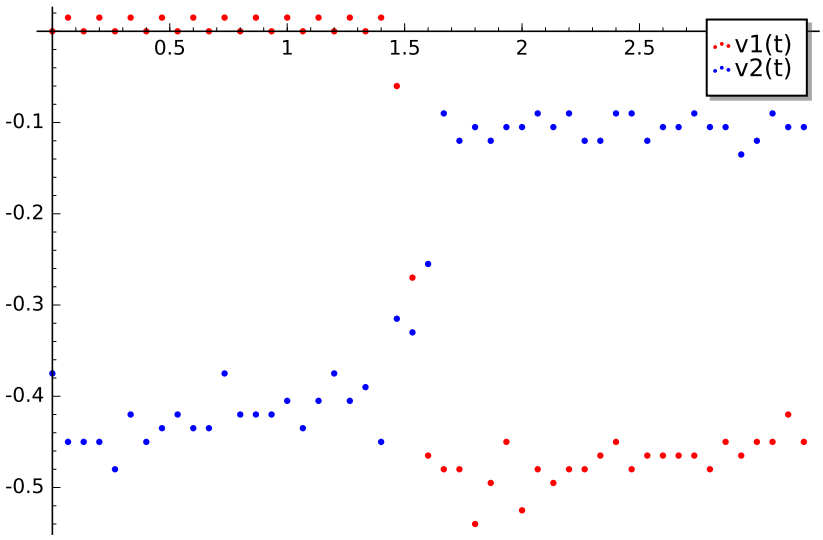


Rys. 3.18: Wykres prędkości od czasu wykonany na podstawie pomiaru drogi.

Widać na nich duży szum spowodowany ograniczoną rozdzielczością odczytu położenia i czasu. Wystarczą niewielkie fluktuacje przyrostów położenia w poszczególnych klatkach filmu, niewidoczne na wykresach z położeniem, a wykresy prędkości rozsypują się. Szum danych przenosi się (i potęguje) na wykresach pędów, enegii kinetycznych (tu mamy kwadrat prędkości!), przyspieszeń i sił. Poradziłem więc uśredniać prędkości na dwóch sąsiednich przedziałach (klatkach filmu):

```
v1 = [((x1[i+1]-x1[i-1])/(2*delta_t)) for i in range(1,99,2)]
v2 = [((x2[i+1]-x2[i-1])/(2*delta_t)) for i in range(1,99,2)]
v1t = [(t[2*i],v1[i]) for i in range(0,49)]
v2t = [(t[2*i],v2[i]) for i in range(0,49)]
vt = point(v1t,color="red",legend_label='v1(t)')+\
   point(v2t,color="blue",legend_label='v2(t)')
vt
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.19.



Rys. 3.19: Wykres prędkości od czasu wykonany na podstawie pomiaru drogi z uśrednianiem.

Szum się zmmiejszył, ale zmalała też ilość punktów wykresu i w ogóle rozdzielczość czasowa, z jaką badamy zjawisko. A najbardziej interesujące procesy zachodzą w krótkiej chwili największego zbliżenia ślizgaczy.

Uczeń sporządził pozostałe wykresy, napisał sprawozdanie i wyciągnął wnioski, ale w chmurze punktów pomiarowych trudno było dopatrzyć się ciekawych szczegółów. Dane można odszumić, stosując bardziej zaawansowane metody matematyczne, daleko wykraczające poza możliwości ucznia szkoły średniej.

Dane drugiego ucznia

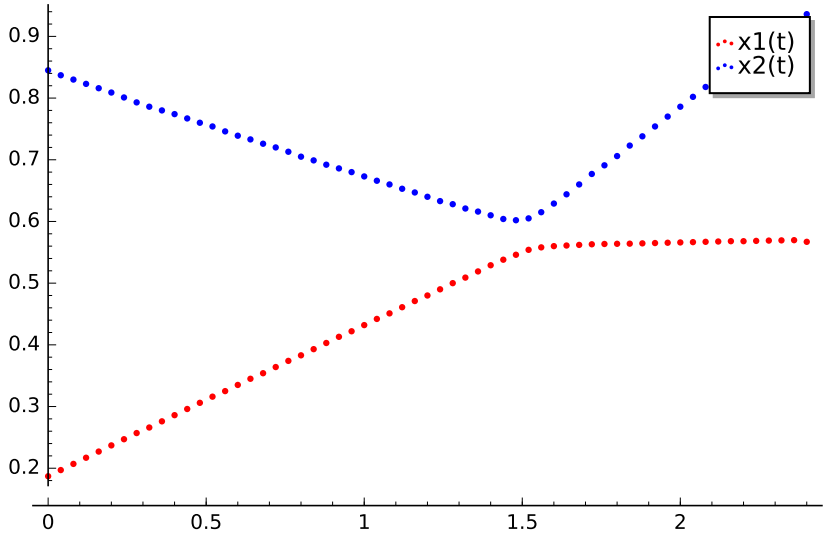
W październiku 2016 roku cała grupa kolejnego rocznika uczniów sfilmowała swoje zderzenia. Starano się zbadać odmienne przypadki, zmieniając masy wózków, wartości i zwroty predkości lub ustawiając jeden ze ślizgaczy nieruchomo (jako tarczę). Przytaczam dane autora najlepszego opracowania (dotyczą filmu cytowanego wyżej).

Wyniki pomiarów i wykresy $x_1(t)$ i $x_2(t)$

```
m1 = 0.4093
m2 = 0.17195
d1 = 0.011
d2 = 0.01
```

```
delta_t = 1/25
t = [(i*delta_t) for i in range(0, 61)]
x1 = [0.187, 0.197, ...       0.569666,   0.567]
x2 = [0.845, 0.837, ....     0.906, 0.92, 0.936]
x1t = [(t[i], x1[i]) for i in range(0, 61)]
x2t = [(t[i],x2[i]) for i in range(0, 61)]
xt = point(x1t, color = "red", legend_label = 'x1(t)')+\
     point(x2t, color = "blue",    legend_label = 'x2(t)')
xt
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.20.



Rys. 3.20: Wykres położenia od czasu wykonany na podstawie doświadczenia.

Wykresy $v_1(t)$ i $v_2(t)$

Od razu zastosowano uśrednianie po sąsiednich klatkach, by zmniejszyć szum.

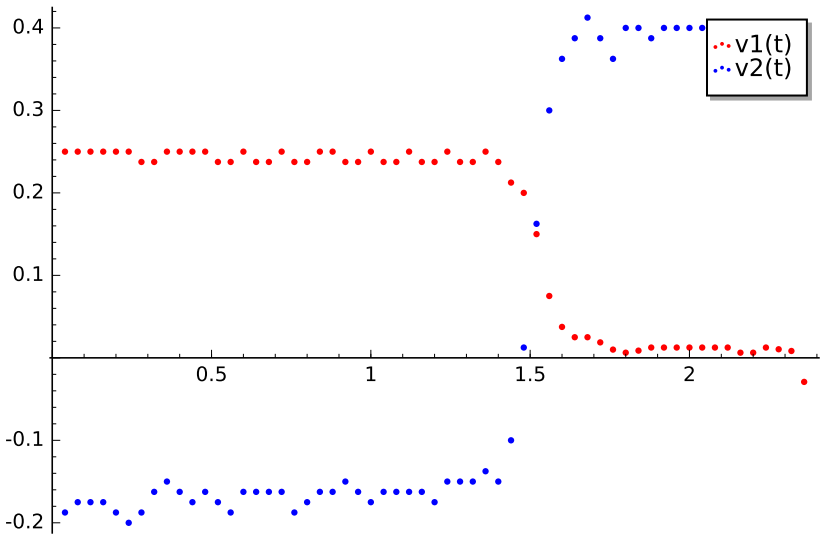
```
v1 = [((x1[i+1]-x1[i-1])/(2*delta_t)) for i in range(0,60,1)]
v2 = [((x2[i+1]-x2[i-1])/(2*delta_t)) for i in range(0,60,1)]
v1t = [(t[i], v1[i]) for i in range(1, 60)]
v2t = [(t[i], v2[i]) for i in range(1, 60)]
vt = point(v1t, color = "red", legend_label='v1(t)') + \
     point(v2t, color = "blue", legend_label = 'v2(t)')
vt
```

Wynikiem działania powyższego kodu jest wykres Rys. 3.21.

Szczegóły analizy zawarte są w orginalnej pracy ucznia:

wersja polska: https://sage01.icse.us.edu.pl/home/pub/146/

wersja angielska: https://sage01.icse.us.edu.pl/home/pub/147/



Rys. 3.21: Wykres prędkości od czasu wykonany na podstawie pomiaru drogi z uśrednianiem.

3.5.4 Wnioski

Przedstawione doświadczenie było jednym z ciekawszych w mojej karierze zawodowej. Ze względu na mnogość różnych sytuacji (dowolne prędkości, kilka mas ślizgaczy do wyboru), można było uzyskać zupełnie różne rezultaty. Uczniowie mieli też swobodę w wyciąganiu wniosków, była to ich samodzielna praca badawcza. Na przykład w cytowanej pracy, na wykresie energii kinetycznej widać wyraźne minimum w chwili największego zbliżenia wózków. Uczeń zinterpretował to jako błąd pomiarowy spowodowany zbyt szybko zmieniającymi się prędkościami. Moim zdaniem jest to moment, w którym energia kinetyczna częściowo zamieniła sie w energię oddziaływań magnetycznych. Ale dlaczego, w takim razie, widać też załamanie na wykresie pędu całkowitego?

Doświadczenie dotyczyło w zasadzie czystej mechaniki, ale przy okazji dało sposobność zbadania charakteru siły magnetycznej. Uczniowie mogli dopasowywać do wykresu $F(r)$ dowolne krzywe - nie narzucałem tutaj rozwiązań. Należało spojrzeć na dane i domyślić się, jakiego typu krzywa będzie najbardziej odpowiednia.

Jak widać z powyższej analizy, kluczowe znaczenie ma precyzyjny odczyt położeń ślizgaczy na poszczególnych klatkach filmu. Nie jest to łatwe i wymaga zastosowania kamery o dobrych parametrach. Szum danych można częściowo usunąć, ale odbywa sie to kosztem obniżenia rozdzielczości, z jaką widzimy całe zjawisko.