O1: Teaching and learning materials in mathematics and physics

Collective work edited by dr hab. Marcin Kostur

Contents

CHAPTER 1

Introduction

Attention: This document has online version which contains live code cells allowing for explorations without need of installation of any software. it can be found at:

- http://visual.icse.us.edu.pl/methodology/

This publication contains scenarios and materials for over 20h lessons in mathematics and physics. It is mostly intended for use in highschools.

The scripts developed in the project using Python are recommended for use in a variety of ways, depending on the needs and capabilities of students and teachers. Thus, they can be used directly during class activities or in the form of workshops, provided that the number of teaching hours is at the teacher's disposal. Materials or parts can also be used in home work using the innovative "flipped teaching" method, where pupils in homework analyze the underlying knowledge. This is a new topic and the lesson is devoted to consolidating and deepening the acquired knowledge and skills (as opposed to the classical methods whereby the basic knowledge is introduced during the lesson and the self-development of the material is done at home). Finally, our materials can be used to explore and broaden the knowledge and skills of students and students with special educational needs (gifted students and those who have the need for self-development or who need to use numerical methods for solving problems using a publicly accessible tool) learn outside the school for self-study, as well as by all those who are interested in repetition, consolidation and extension of knowledge.

The lesson scenarios from the iCSE4school project can also be used to work with students with less potential in the arts by presenting pieces of our materials by teachers, as well as in the form of workshop work, where students modify the source texts themselves, watching the effects of these changes and formulating appropriate conclusions. This way of using materials not only creates opportunities for teachers and young people, but also effectively uses classroom scenarios using Python in working with younger learners at earlier stages of education.

The materials from the iCSE4school project can be used in e-learning, blended-learning, and m-learning, using computers or other devices (eg smartphones, tablets); with access to the internet.

During the project students and teachers from two highschools worked on SageMath servers. They have created almost 3000 documents, some of them have been "published" and can be viewed:

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

CHAPTER 2

Teaching materials: Mathematics

2.1 Limit and continuity of a function at a point

2.1.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at XXXIII LO M. Kopernika w Warszawie.

It was prepared by Mirosław Malinowski based on his lesson.

2.1.2 Number sequences - review

1. What do we call a number sequence?
2. Recall the Cauchy definition of the limit of a number sequence.
3. Calculate the following limits:

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

4. Using SAGE calculate limits of the sequences above and sketch their graphs.

Example:

Let's check out convergence of the following sequences: $a_n = 1 + \frac{1}{n}$ 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
```

Running above code should produce the following code:

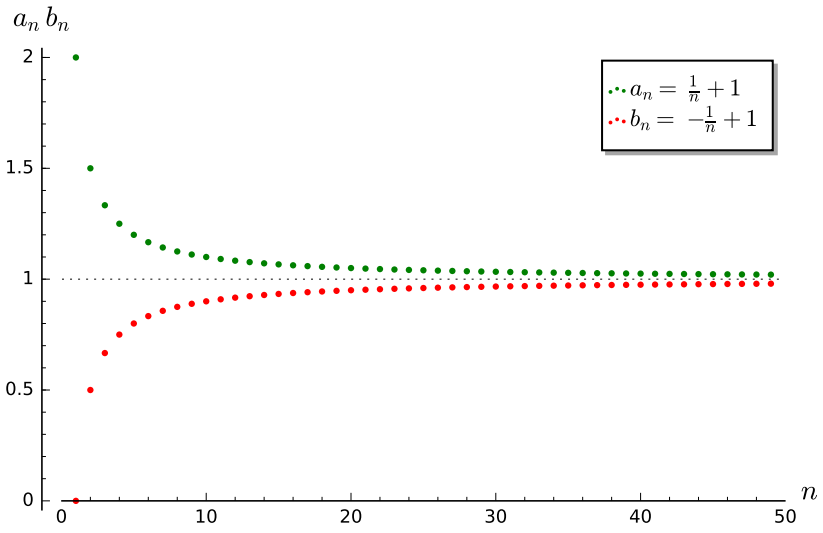


Fig. 2.1: Two sequences convering to a number.

2.1.3 Introduction of Heine definition of the limit of a function at a point

A function $f(x)$ has a **left-hand limit** g at a point x_0 (denoted $\lim\limits_{x \to x_0^-} f(x) = g$), if for every sequence (x_n) such that $x_n < x_0$), convergent to x_0, the sequence of values $(f(x_n))$ is convergent to g .

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

A function $f(x)$ has a **right-hand limit** g at a point x_0 (denoted $\lim\limits_{x \to x_0^-} f(x) = g$), if for every sequence (x_n) such that $x_n > x_0$), convergent to x_0, the sequence of values $(f(x_n))$ is convergent

to g .

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

Note: The function $f(x)$ has limit g at the point x_0, if the left-hand limit and the right-hand limit exist at x_0 and $\lim\limits_{x\to x_0^-} f(x) = \lim\limits_{x\to x_0^+} f(x) = g$.

Example

Using the Heine definition we will calculate the limit of the function $f(x) = \frac{x}{x+1}$ at the point $x_0 = 1$.

```
var('x0 x_0')
x0 = 1
f(x) = x/(x+1) # Given function
a(n) = x0 - 1.5/n # A sequence convergent to x0 from below
b(n) = x0 + 1.5/n # A sequence convergent to do x0 from above
gl = limit(f(a(n)), n=oo) # Left-hand limit of a sequence of values of the function f(x)
gr = limit(f(b(n)), n=oo) # Right-hand limit of a sequence of values of the function f(x)
la = [a(k) for k in srange(1, 100)] # Terms of a sequence convergent to x0 from below
lb = [b(k) for k in srange(1, 100)] # Terms of a sequence convergent to x0 from above
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
```

Running above code should produce the following code:

Excercise 1

Using the procedure given above find the following limits:

1. $f(x) = \frac{x+2}{x-1}$ at $x_0 = 2$
2. $f(x) = \frac{x^2-3}{2x^2-1}$ at $x_0 = 1$
3. $f(x) = \frac{sinx}{x+1}$ at $x_0 = \frac{\pi}{2}$

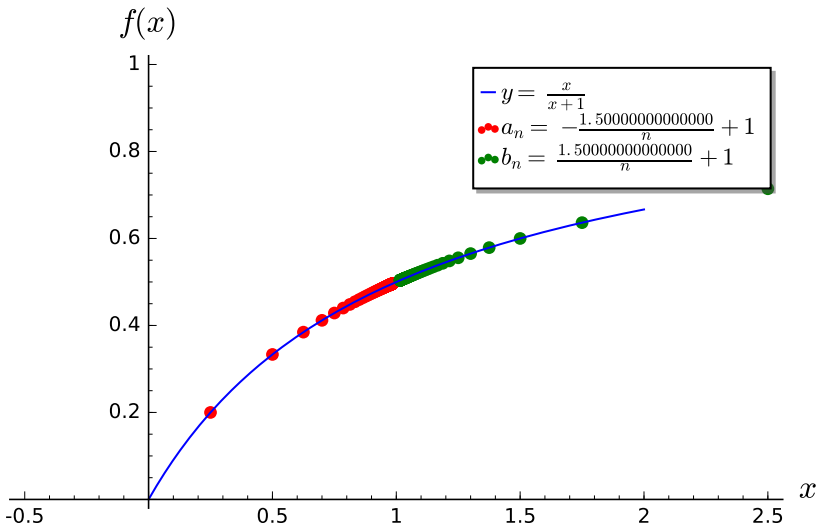


Fig. 2.2: Two sequences convering to a number.

4. $f(x) = \frac{x}{|x|}$ at $x_0 = 0$

5. $f(x) = \begin{cases} x^2 & \text{for } x \leq 0 \\ 2^x & \text{for } x > 0 \end{cases}$ at $x_0 = 0$

2.1.4 Definition of continuity of a function at a point

A function f is continuous at a point x_0 if for any sequence (x_n) with terms belonging to a neighbourhood of x_0, convergent to x_0:

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

Example

Check whether the function $f(x) = \begin{cases} x^2 - 4 & \text{dla } x < x_0 \\ \sqrt{x} - 4 & \text{dla } x \geq x_0 \end{cases}$ is continuous at the point $x_0 = 0$.
Is the given function continuous at other points 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("The function is continuous at the point ",x0)
else:
    print("The function is NOT continuous at the point ",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)))
plt += plot (fr, (x, x0, x0+5), color = 'red', legend_label="$y=$
 ↪  $%s$"%latex(fr(x))) + point((x0, f(x0)), color = 'red', size =
 ↪  48)
plt
```

Running above code should produce the following code:

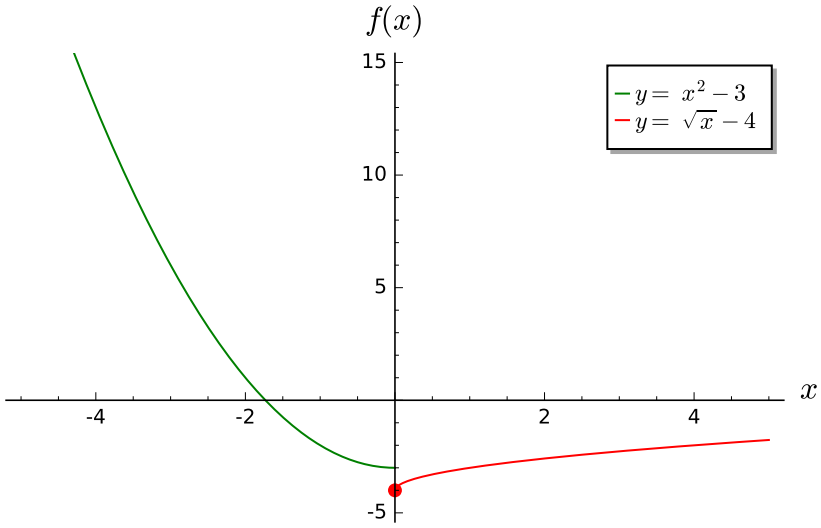


Fig. 2.3: Two functions

Excercise 2

Verify if the following functions are continuous at the given points.

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

5. $f(x) = \begin{cases} x \sin \frac{1}{x} & \text{for } x \neq x_0 \\ 0 & \text{for } x = x_0 \end{cases}$ at $x_0 = 1$

2.2 From point to point

2.2.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Krzysztof Oleś based on his lesson.

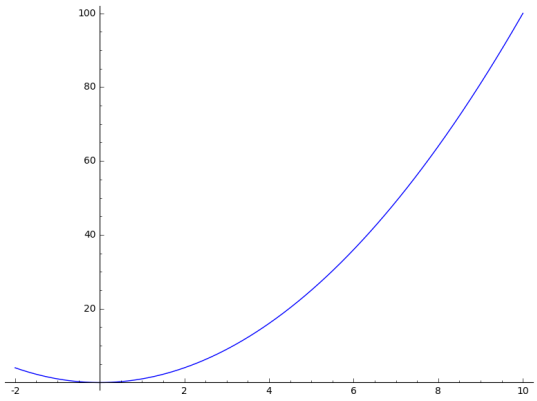
2.2.2 From point to point

Let's think about a graph connected with a function, i.e. $f(x) = \log_x \left|4 \sin\left(\frac{\pi}{2} - 3x\right) - 6\right|$. That's of cource a continuous line. Really?

Using SAGE for the preparation of drawings, we can rely on:

1. the description of the function

```
a=-2 #change it
plot(x^2, (x, a, 10))
```

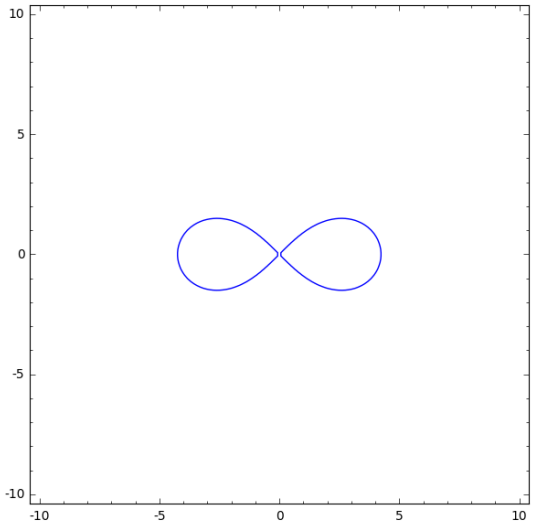


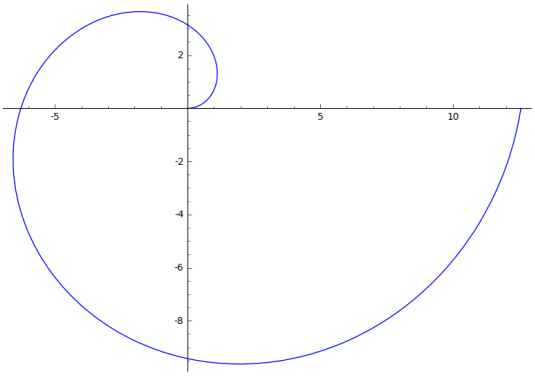
2. equation, which we can call "entangled"

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

3. polar formula

```
a=2 #change it
polar_plot(a*x, (x, 0, 2*pi))
```


10
5
0
-5
-10
-10
-5
0
5
10


2
-5
5
10
-2
-4
-6
-8

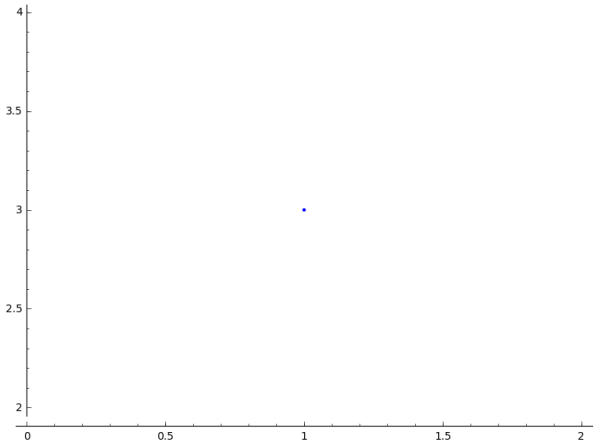
Sometimes we can see some similarities among the effects of using these different approaches: in each of these cases a set of points appears on computer screen - it is fitted in more or less complex formula.

Maybe should we use the simplest method: from point to point? Using recursion?

Let's think about the point placed in the coordinate system - it gives the opportunity to trace the connection between geometry and numerical operations. Important for us is the possibility of student's experimentation and computer fun - examples are presented in their programming layer easy.

So we start by placing the point on the screen.

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



Apparently nothing - so let's put on the screen five points...

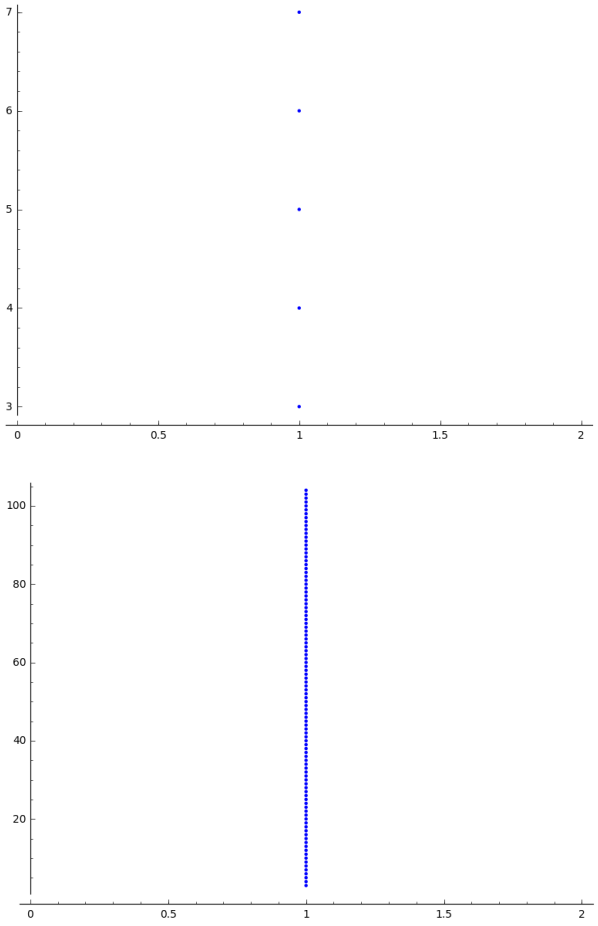
```
fig=point((1,3),(1,4),(1,5),(1,6),(1,5))
fig
```

After this small planned error we try to add points.

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

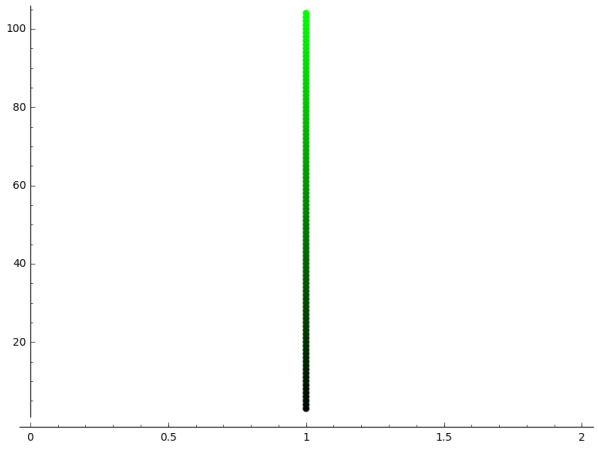
Note that even with ctr+c+ctrl+v it takes time and it's scary to think about boredom of the placement in this way a hundred points - in a situation where we can see a certain **REGULARITY** in the second coordinate points considered. Therefore, let's use it.

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



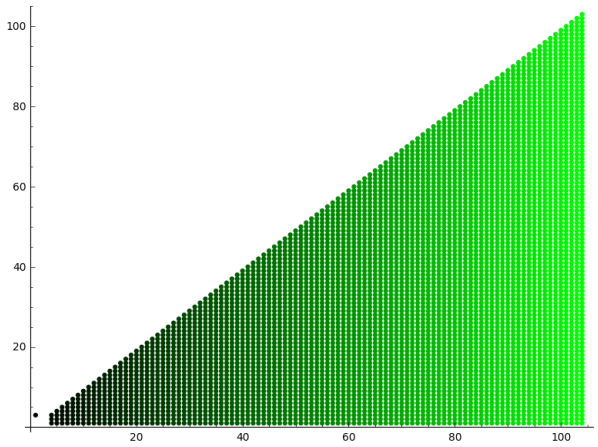
Change the point size, playing tinge.

```
s=40 #change size
fig=point((1,3),rgbcolor=(0,0,0),size=s) #what does (0,0,0) mean?
for n in range(4,105):
    fig=fig+point((1,n),rgbcolor=(0,n/105,0),size=s)
fig
```



Do not forget about the possibility of placing the loop in the loop.

```
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
```



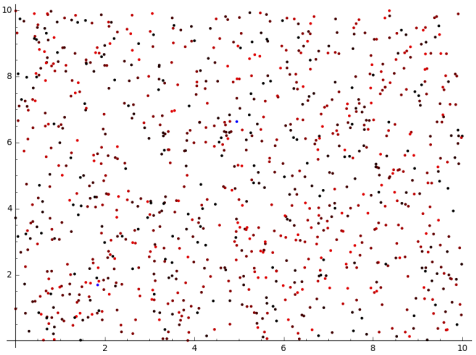
Looking at the obtained effect we see a problem with the "left" apex of a triangle - try to remove it properly manipulating numbers.

If we did, then we can go to the draw.

```
n=1001 #change it
a=10*random() #why do we use multiplication?
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)) #change the way of
     ↪ coloring
fig
```

In the example above, you can see a kind of chaos... Can you control over the points?

Imagine a situation in which the specified starting point (a, b) is transformed in one of eight randomly selected transformations. Each of them consists of two parts: a linear operation on the first coordinate (three numbers a_i, b_i, c_i), and linear operations on the second coordinate (three numbers d_i, e_i, f_i). After the transformation we obtain a new point (a, b), which is thrown into the described formulas again - and of course, not satisfied with two points, we (computer?) repeat it hundreds of times.

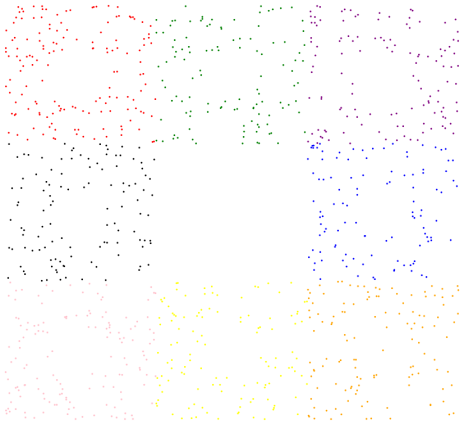


Let look carefully at the source code (specially on lines containing #).

```
a=0 #the first coordinate of the starting point
b=0 #the second coordinate of the starting point
d=1001 #number of repetitions and lengthy list of factors below...
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 #and finally the end of
 ↪  the list
r=point((a,b),axes=False, frame=False,size=0)
for c in range(1,d):
  n=randint(1,8) #select one of the eight maps
  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))
```

With a hundred repetitions figure seems chaotic - therefore we've repeated our experiment more times (you should change d again).



Do our points put us in something familiar?

With the seeming chaos should emerge Sierpinski's carpet.

But let's try to experiment and answer the following not easy questions:

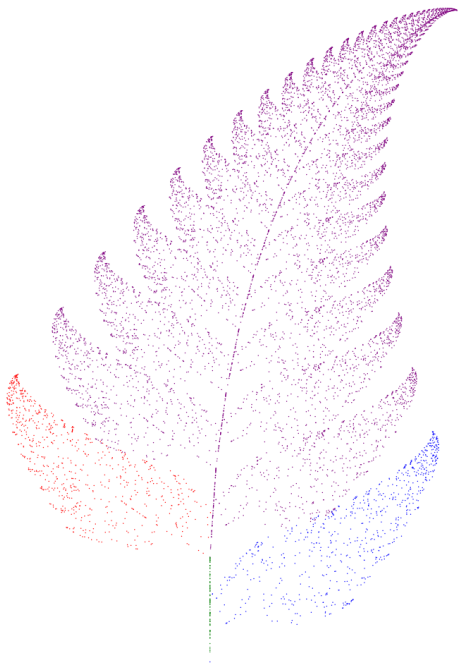
- Does built figure depend on the choice of the starting point?
- What happens to the built figure, if we change $a_i, b_i, c_i, d_i, e_i, f_i$?
- What happens to the built figure, if one of the eight maps we exclude - how can we quickly in the source code do it?
- Why is carpet colored in such a way and not another?

We believe that the answers to these questions - based on tested assumptions - will be surprising...

Perhaps more than a carpet fascinates us known (almost all) leaf.

```
c=10001 #number of repetitions
a=0 #the first coordinate of the starting point
b=0 #the second coordinate of the starting point
p=7 #the width of the picture
q=10 #the height of the picture
```

```
r=point((a,b),size=1, axes=false, frame=false) #by changing the
 ↪  'false' to 'true' you can generate axes and frame
for m in range (0,c):
  n=random()
  if n<0.01: #what is it for?!
      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: #why 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))
```

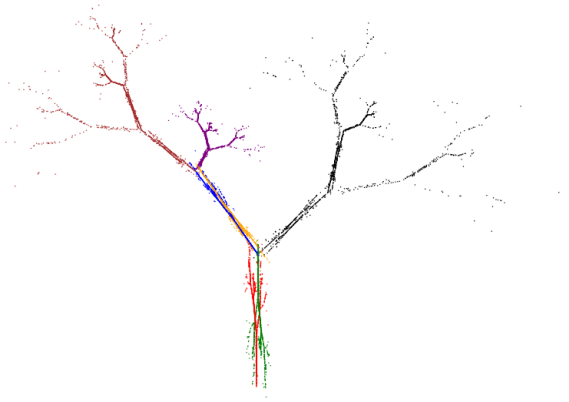


Perhaps the carpet and the leaf **ATTRACT** us to further experiment, in which we will try to

arrange the coefficients in the tables (various methods for introducing the coefficients in the above two examples encourage such arrangement).

```
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
```

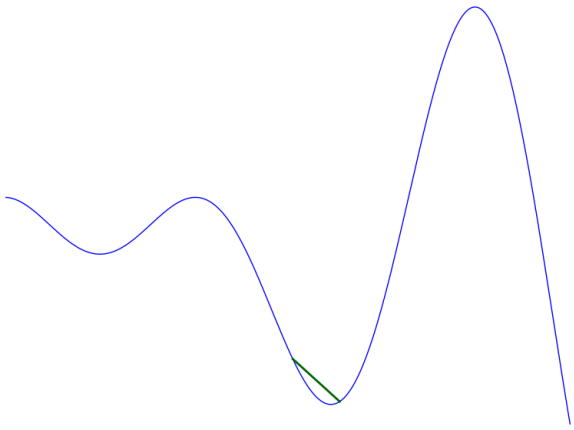
```
        r=r+point((c,d),axes=False, frame=False,size=1,color='purple')
    if n==7:
        i=(a7[0]*c)+(a7[1]*d)+a7[2]
        o=(a7[3]*c)+(a7[4]*d)+a7[5]
        c=i
        d=o
        r=r+point((c,d),axes=False, frame=False,size=1,color='brown')
r
```



Let's go back to the Sierpinski's carpet. Or is it rather a kind of line or something like connected squares? Is repeating iterations indefinitely (in our head...) move closer to a more normal squares? What does "more" mean?

Look at the blue line below - we want to measure it with a green ruler.

```
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)
```



Let's estimate the length of the blue line. Let $M(\epsilon)$ means the length of the measured curve by a ruler legth of ϵ, and $L(\epsilon)$ number of touchdowns rulers into the curve. Note that the smaller

ϵ, the estimation more accurate. Note that $M(\epsilon) \approx \epsilon \cdot L(\epsilon)$ and

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

(if the ruler is shorter, the more times we have to apply it). If we repeat this reasoning, considering the area instead of the length, a "ruler" would be a square with a side length of ϵ and

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

What about the volume? Perhaps a "ruler" woud be a cube and

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

So

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

and $d = 1$ (when we try to estimate the length), $d = 2$ (when we try to estimate the area), $d = 3$ (when we try to estimate the volume).

Let's try to get to d.

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

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

and

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

maybe can we write a formula like this

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

(are there any mistake in replacing signs: $\sim, \approx, =$ above?).

It looks quite dramatically. Let's see how this works in the case of the Sierpinski's carpet. This figure we can (**SURELY**?!) cover by 1 square with a side length of 1, 8 squares with a side length of $\frac{1}{3}$, 64 squares with a side length of $\frac{1}{9}$,..., 8^n squares with a side length $\left(\frac{1}{3}\right)^n$ and

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

Sierpinski's carpet is something between a line and a square - perhaps we came a little closer to the concept of dimension...

2.2.3 Summary

In the text above three words are bold. Finally, we would like to return to them.

REGULARITY Actions based on SAGE can help students explore recursion (a kind of regularity repeated over and over again, thanks to computers very many times).

ATTRACT It is worth mentioning in the context of using by students SAGE attention to the concept of attractor (not only in the mathematical sense: fractals are often attractors) - a computer may able to attract them to small discoveries.

SURELY It should be emphasized that the above considerations concerning dimension are only a signal of a problem - but they can build a student intuition (which should not be immediately deal with the problem of the existence $\lim_{\epsilon \to 0} \ldots$).

And by the way the last word: where is the limit between a student's intuitive fantasizing and a strict mathematical waffle?

2.3 Amazing approximation

2.3.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Krzysztof Oleś based on his lesson.

__This project is the result of activities (inspired by the textbook__ "Matematyka się liczy" edited by prof. Wacław Zawadowski)

2.3.2 Introduction

We want to use a computer to illustrate a SIMPLE problem by using BASIC programs.

- Why BASIC?

Because we don't know if we are familiar enough with SAGE.

- Why SIMPLE?

Because we are keen on making some people surprised by this part of mathematics which is told to be boring... Most of people don't like counting, so we'll repeat not exciting calculations and computer will lead us (perhaps) to some misunderstandings in using numbers...

2.3.3 Part 1

Let's start with approximation of square-root of two. We'll use square-root algorithm (which is based on Newton's method for finding zeros of a function, this case is known as Babylonian method).

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

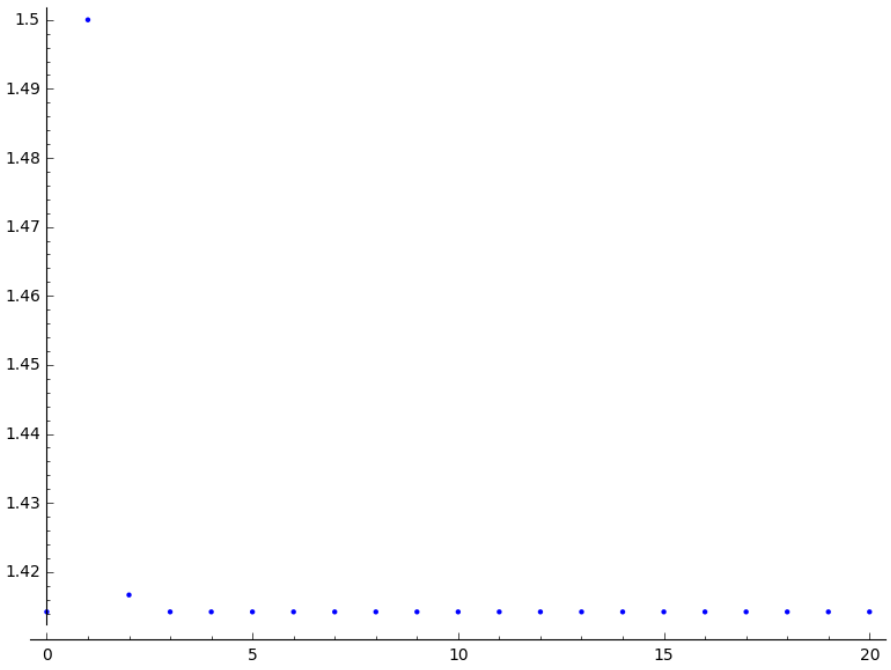
Do you know

- what does 2.12390141451912e-6 mean?
- how does error change while n changes?

Let's see how fast this algorithm is.

```
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
```

```
rot=2
graph=point((0,sqrt(2)))
for n in range(1,21):     #does range (1,51) change anything?
        rot=0.5*(rot+2/rot)
        graph=graph+point((n,rot))
plot(graph)
```



Wait a minute... What are we thinking about using a word "fast"?

Let's make a comparison. One of the most popular number is π, so we'll use an algorithm of approximation of this object. It'll be based on Wallis' product for π, written down in 1655:

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

It's quite complicated - perhaps a big π makes us confused. What about a formula like that

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

Easier?

A big π stands for a product (something like many, many multiplications...).

How does it work?

Tiring (you can check on the paper...) calculations will do the computer.

```
w=1
for i in range(1,6):
        w=w*((2*i)/(2*i-1))*((2*i)/(2*i+1))
        print 2*w      #do you prefer fractions or decimals?
        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
```

Do you know

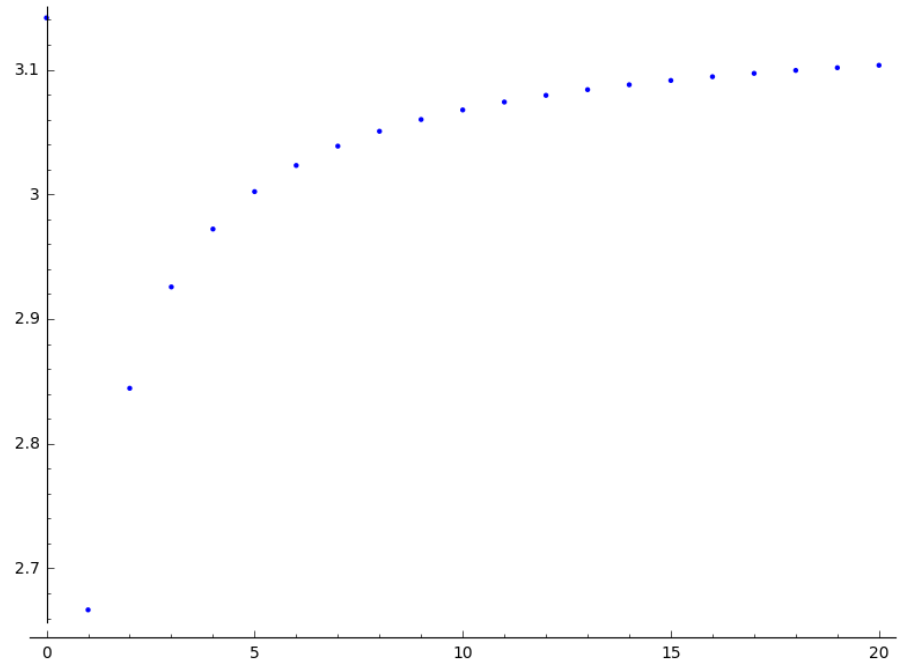
- what does abs() mean?
- what does N() mean?
- how does error change while n changes?
- how many reps do we have to do to get to 3.14?

Let's see how fast this algorthm is.

```
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)
```

Now we can compare the speed of the first and second algorithm and ask very awkward questions:

- Have we ever wondered about how our calculator approximates the numbers?
- Maybe our colleague's calculator makes it better. What does "better" mean?



- We counted some errors - SAGE had to approximate square-root of two and π (these numbers are not rational): did SAGE make a mistake? How big?

2.3.4 Part 2

Okay, but who is interested in differences in approximations of numbers, for example on the fifteenth decimal place?

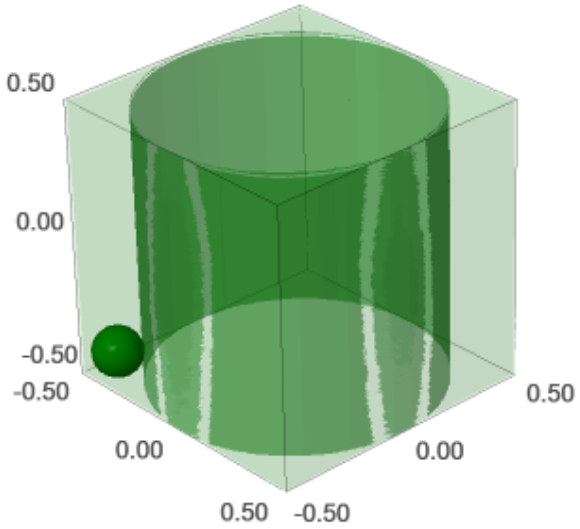
Let's try to take care of some geometrical problem.

Consider the cylinder inscribed in a cube (bases of the cylinder are inscribed in the two parallel faces of the cube). In the corner of the cube put the ball of maximum volume tangent to the cylinder. What is this volume?

```
var('x,y,z')
r=(sqrt(2)-1)/(2*sqrt(2)+2)      #where did it come from?
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
```

As we can see the edge of the cube has a length of 1

$$x, y, z \in (-0.5, 0.5),$$



and the cylinder is connected with the circle pattern of

$$x^2 + y^2 = 0.25.$$

And where was taken

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

Denoted by r the radius of the search ball. With simple relationship between the diagonal of the square and the rays of both circles we receive:

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

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

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

and searched volume is equal to

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

Everyone has heard about the transformation of expressions that contain numbers that are not rational, so let's get to the tedious work...

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

but

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

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

and finally

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

Let

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

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

Of course $w_1=w_2=w_3=w_4=w_5$, but: are there any difference among $w_1,\ldots,w_5$ when when we approach the square-root of two? Let's see...

```
p=1.44      #change 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)
```

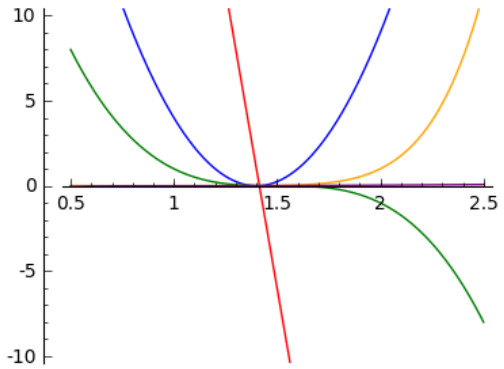
```
apr= 1.44000000000000
w_1= -1.80000000000000
w_2= 0.0399999999999997
w_3= 0.00172800000000000
w_4= 0.00725631385599999
w_5= 0.00586392693661584
```

It turns out that the differences among approximations are big, if we accept the square-root of two as 1.41 (and yet many people think of approximations only to parts hundredths). What about: 1.414, 1.4142, 1.41421, 1.414213 and so on?

The differences are large, what we can see, considering graphs related functions connected with $w_1,\ldots,w_5$.

```
@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))
```



Let's go back to the drawing containing our ball.

```
var('x,y,z')
p=1.41      #change p
r1=N(0.5*((99-70*p)^(1/3)))      #why here "^(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      #change 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
```

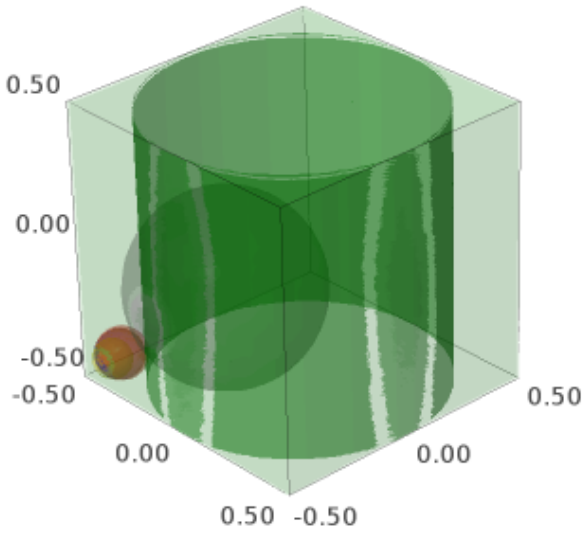
You should change

- p: 1.414, 1.4142, 1.41421, 1.414213; don't forget about sqrt(2),
- r: $r_1, \ldots, r_5$.

Let's look at all five balls at once.

```
var('x,y,z')
p=1.41      #why is there a problem with 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
```



Is not that strange?

Let's finish our calculations, considering we're looking for volume, whereby because we already have enough of looking at distant places after the decimal assume that the edge of the cube has a length of 60.

```
p=1.41      #change 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      #why here 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
```

Again - you should change p: 1.414, 1.4142, 1.41421, 1.414213; don't forget about sqrt(2).

And again: is not that strange? Perhaps not, but considered above example shows how much we have to be careful using approximations.

2.3.5 Summary

We wanted to show how important is the difference between manipulating algebraic expression in kind

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

and its approximation.

Why?

Firstly, due to the fact that we usually use the numbers which are not rational and that means the need to use their approximations. We tried to show two different - in the sense necessary number of the implementation iterations - algorithms. We suggested finding no small number of iterations leading to approximations of π proverbial 3.14. Because the differences of decimals can not really interest many people - we decided to see (!) in a geometric problem a specific significance of the adopted approximations square-root of two.

Secondly, in Polish schools, the advantage is algebraic problem solving (for example equations). This means that the matriculation examination solution of the equation

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

should look like

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

There is a mental gap between writing the above "pictures" and the following "numbers"

$$x_1 \approx -4.78303, \quad x_2 \approx 0.92589.$$

Perhaps the whole project should be considered as draw attention to the difference between the signs

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

2.4 RSA asymmetric cipher.

2.4.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Krzysztof Jarczewski based on his lesson.

2.4.2 Definition of congruence relation.

Two integers a and b are said to be **congruent modulo** n, written: $a = b(mod\ n)$, when: $(a-b) \cdot k = n,\ \ k \in Z$.

Examples:

2 = 2 (mod 8), because 2 - 2 = 0, which is a multiple of 8,

3 = 18 (mod 5), because 3 - 18 = -15, which is a multiple of 5,

100 = 1 (mod 9), because 100 - 1 = 99, which is a multiple of 9,

250 = 206 (mod 22), because 250 - 206 = 44, which is a multiple of 22.

Exercise 1.

Find x if you know: $3x = 1(mod\ 4),\ \ 5 < x < 10$.

```
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
```

Exercise 2.

Find x if you know: $3x + 2 = 1(mod\ 5)$.

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

We can realise there are infinitely many solutions. What is more, the solutions determine an arithmetic progression.

Exercise 3.

Find x if you know: $3x = 1(mod\ 6)$.

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

In the exercise above there is not a number which satisfies the given congruity.

2.4.3 The Chinese remainder theorem.

The following exercise can be solved using the Chinese remainder theorem. One of the most important theorems in number theory and cryptography. This theorem allows to share a secret among several people (valid numeric password).

Exercise 4.

A bar of chocolate consists of less than 100 pieces. While dividing it into three equal parts, there remains 1 piece of chocolate. While dividing it into 5 parts, what remains are 3 pieces of chocolate but when dividing into 7 equal parts, 2 pieces remain.

We know that the number of chocolate pieces must satisfy the below congruence:

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
```

2.4.4 Fermet's Little Theorem.

If p is a prime number and a is not divisible by p,

then $a^{p-1} - 1$ is an integer multiple of p, or in symbols: $a^{p-1} = 1(mod\ p)$

Let's check the correctness of the Fermet's Little Theorem basing on the Python language.

For a we substitute 35, so p=5 and p=7, the am. theory is not satisfied. We can even state that it must be dividable by p.

This code was written and posted the students in the classroom.

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

2.4.5 Message Encryption.

Cryptography was mentioned in the Antique Times for the first time. So, we can conclude that encryption and writing were invented at the same time. Encryption was used to send military and political messages. During the IT lessons we acquired (or will acquire) the Caesar cipher. It is a simple encryption where letters are substituted. Although the ciphered message is not understandable, but simple to decryption . Other methods of encryption applied in the Antique Times were much more sophisticated and more difficult to de cryption . Until 1960s of the 20th century only symmetric encryptions had been well-known. They are the encryptions which have just one method of ciphering and deciphering the message.

In the 1970s of the 20th century, the power of computing and the need for data protection led the cryptographers to invent an asymmetric encryption, where two different keys are used – one to encrypt and the other to decrypt the message (the order of keys is of no importance). One of the keys is available to a person who is to send the secret message. You can even make the key available to the public on your website (available to everyone – a public key). The other key is a secret one (a private key) which is only known to us and cannot be made available to anyone. Only the private key allows us to decipher the message.

Below, you can find a simple asymmetric encryption which can be cracked (if you know the digits: d, n, define number e) it is your task to score extra points.

How to create asymmetrical encryption mathematically?

To create a simple asymmetrical encryption you need various natural numbers: $a, b, a1, b1$.

The bigger the numbers is, the safer the encryption becomes. It is more difficult to decript if you don't know the proper key.

For our task we take only two-digit and three-digit numbers.

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

The key of the cipher are pairs of numbers: a public key (d, n) and a private key (e, n).

Below you can find an example of the number cither:

```
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
encrypted = (number*d) % n
print "encryption:", encrypted
# decryption
decrypted = (encrypted*e) % n
print "decryption:", decrypted
```

What to do when the number is larger than n?

1. We calculate the remainder of division by n (we receive a “portion” to cipher)
2. We cipher the “portion”
3. We add the ciphered “portion” in the next power of number n to the code.
4. We divide the number by n
5. If the result is larger than 0, repeat the steps from 1 – 4.

```
number=12345656767563535236421387987979799674354678943534524 1 #Big
 ↪  number(message)
encrypted = 0
i=0
while number>0:                                 # 5
    pomoc=number%n                              # 1
    encrypted = encrypted + ((pomoc*d) % n)*n^i    # 2, 3
    i=i+1
    number=int(number/n)                        # 4
print encrypted
```

In the similar way the message is decripted.

Help:

number → encrypted	encrypted → decrypted	d→e

Try to decription the number (message) below.

```
i=0
while number>0:                                 # 5
    pomoc=number%n                              # 1
    encrypted = encrypted + ((pomoc*d) % n)*n^i    # 2, 3
    i=i+1
```

```
    number=int(number/n)                    # 4
print encrypted
```

What we usually want to do is to cipher a text not a number, so we have to substitute letters into numbers. We shall use ASCII code. Each letter, symbol is given a number from 1 to 128.

Below, you can find the algorithm of the encryption (this code was written and posted by the students in the classroom).

```
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
```

The full algorithm of encryption.

Following the submission of these algorithms we get full algorithm to encrypt and decrypt text messages.

```
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
encrypted = 0
i=0
while number>0:
    pomoc=number%n
    encrypted = encrypted + ((pomoc*d) % n)*n^i
    i=i+1
    number=int(number/n)
print "encription:", encrypted
```

The full algorithm of decryption.

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

2.4.6 RSA asymmetric cipher.

RSA is one of the first and most popular algorithm cryptosystems with a public key. It was designed in 1977 by Ron Rivest, Adi Szamir and Leonard Adleman (RSA name derives from the first letters of the creators' surnames).

The security of the RSA cryptosystem is based on the decomposition of large complex numbers (more than two-digit numbers) into prime numbers (factoring problem).

Example below.

```
@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"
```

Notice how time-consuming the calculation of the distribution of prime factors.

Generating RSA cryptosystem.

1. Choose two large prime numbers: p, q (in practice you use numbers which are more than a hundred digit, but we use three-digit numbers).
2. Compute: $n = p \cdot q, \;\; f = (p-1)(q-1)$.
3. Choose an integer d such that: $1 < d < f$ and $gcd(d, \;\; f) = 1$ (You can choose a prime number).
4. Determine e as: $de = 1 \; (mod \; f)$.

Public key: (d, n)

Private key: (e, n)

It is enough to copy the algorithm of cither from previous lessons and substitute them.

```
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
```

Determine e as: $(d \cdot e) \; modf = 1$.

We can use expanded Euclidean algorithm, to find e number. My students changed the existing program on the Internet, but not always, does it generate the correct number. Can you improve this code!

```
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
```

The full algorithm of encryption RSA.

It is enough to copy the algorithm of coding from the previous lessons and substitute pomoc*d them 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
encrypted = 0
i=0
while number>0:
    pomoc=number%n
    encrypted = encrypted + ((pomoc^d) % n)*n^i
    i=i+1
    number=int(number/n)
print "encription:", encrypted
```

The full algorithm of decryption RSA.

It is enough to copy the algorithm of coding from the previous lessons and substitute pomoc*e them pomoc^e.

```
tekst2=""
decrypted = 0
i=0
print "encription:", encrypted
# decription
while encrypted>0:
```

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

2.5 Polynomial approximation

2.5.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Krzysztof Jarczewski based on his lesson.

2.5.2 Factorial

The **factorial** of a integer n , denoted by n !, is the product of all positive integers smaller than or equal n.

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

For example,

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

The **factorial** in SageMath.

The first example counts factorial according to the definition.

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

The second example uses a built-in function in SageMath.

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

2.5.3 Derivative

Derivative will be interpreted as a mathematical operation on a function.

Basic formulas:

$$\begin{array}{ll} n' = 0 & \\ x' = 1 & \\ (x^n)' = n \cdot x^{n-1}, & n > 1 \\ (sin(x))' = cos(x) & \\ (cos(x))' = -sin(x) & \end{array}$$

The following examples in Sage with operation *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))
```

Next formulas for the derivative.

The following formulas for the derivative of the sum, difference, product and quotient of functions.

$$f, g - functions, \qquad c - real\ \ number$$
$$(c \cdot f)' = c \cdot f'$$
$$(f + g)' = f' + g'$$
$$(f - g)' = f' - g'$$
$$(f \cdot g)' = f' \cdot g + f \cdot g'$$
$$(f/g)' = (f' \cdot g - f \cdot g')/g^2$$

Comments

The number before the variable does not change operations on the derivative.

Algebraic expressions separated by + or – count separately.

Examples

$$(c \cdot f)' = c \cdot f'$$
$$(f + g)' = f' + g'$$
$$(f - g)' = f' - g'$$

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

$$(f \cdot g)' = f' \cdot g + f \cdot g'$$

```
f=x*cos(x)
show("f(x)=",f,",          f'(x)=",f.diff(x))
g=x^2*sin(x)
show("g(x)=",g,",          g'(x)=",g.diff(x))
```

$$(f \cdot g)' = f' \cdot g + f \cdot g'$$

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

Derivatives of derivatives - Derivatives of higher orders.

We can calculate the derivative of a derivative.

Derivatives of higher orders are written in the following way:

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

Below, the calculations of higher order derivatives of the 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))
```

Calculating the value of the derivative at the point.

Derivative of a function is a function so we can calculate the value of the derivative for the argument.

Below suitable examples.

```
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)
```

2.5.4 Polynomial.

Polynomial of *n* degree and *x* variable is called function:

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

Conclusion

Linear function and quadratic function are polynomial.

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

2.5.5 Line.

We know are line goes through two points. Besides, knowing the coordinates of the points above, we can determine the formula of this line. We should remember that the formula is a linear function:

$$y = ax + b$$

Directional factor and the intercept can be calculated from these formulas:

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

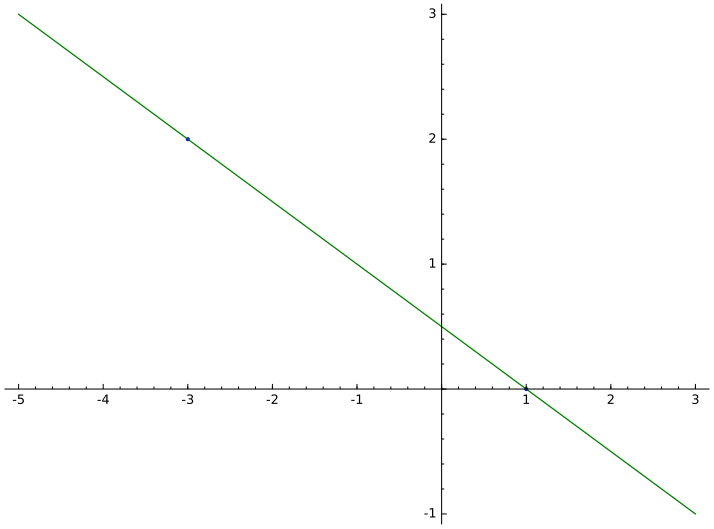
Typing the appropriate equations, we can draw a straight line through two points.

```
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 `f_liniowa`.

2.5.6 Parabola.

Below, there is an example for three points which are not collinear. Then we can determine the quadratic function, which includes these points. So we have to determine *a, b, c* coefficients



from the following equation quadratic function.

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

This work is tedious, even for a specific example. If we wanted to determine appropriate models as above for the linear function that probably it would probably take us a long time.

Below we use the capabilities of Sage.

```
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)
```

We calculate the following equations, where he search coefficients: *a, b, c*.

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

Change the above system of equations for the corresponding matrix equation.

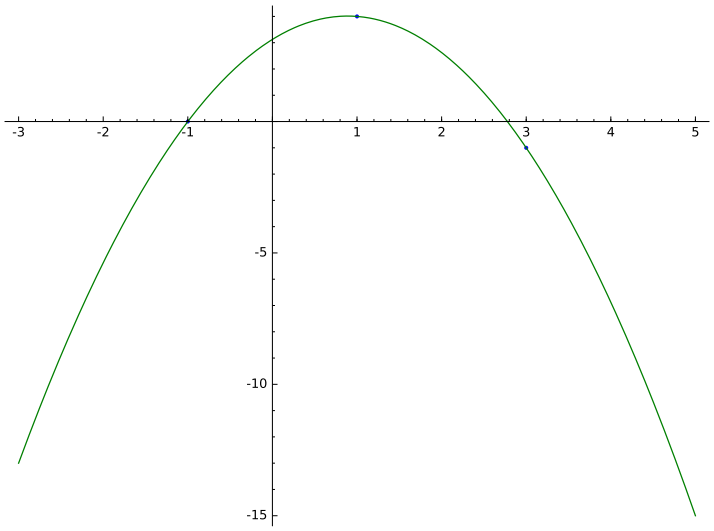
$$\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

In the SageMath we can easily solve this equation is enough to apply the following operation:

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

```
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 `parabola`.



2.5.7 Polynomial.

Here is an example for a few random points. The resulting function is a polynomial.

If you specify *n* points, it certainly passes through these points a polynomial of degree less than *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)
```

For the random points calculate polynomial coefficients.

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

We draw a polynomial that goes passes through the given points.

```
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 `wielomian`.

2.5.8 Taylor's formula.

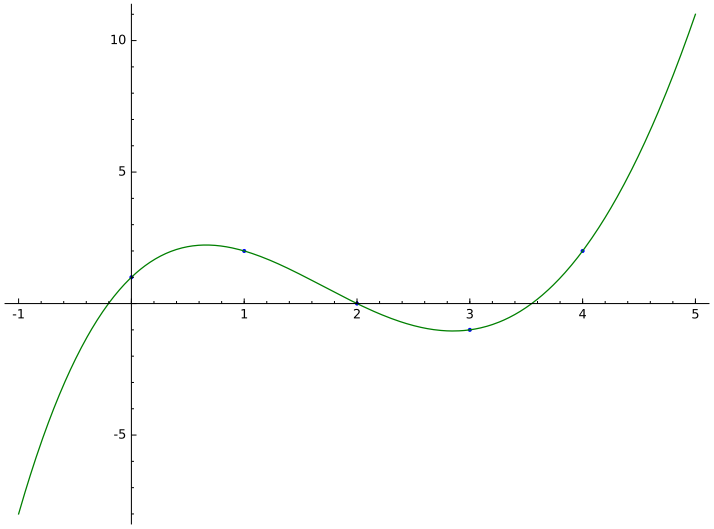
From the mathematical analysis is known below the following formula is known. It approximates any function corresponding polynomial.

Taylor's formula

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

We can simplify the above formula substituting for a = 0. We get **The Taylor-Maclaurin formula**.

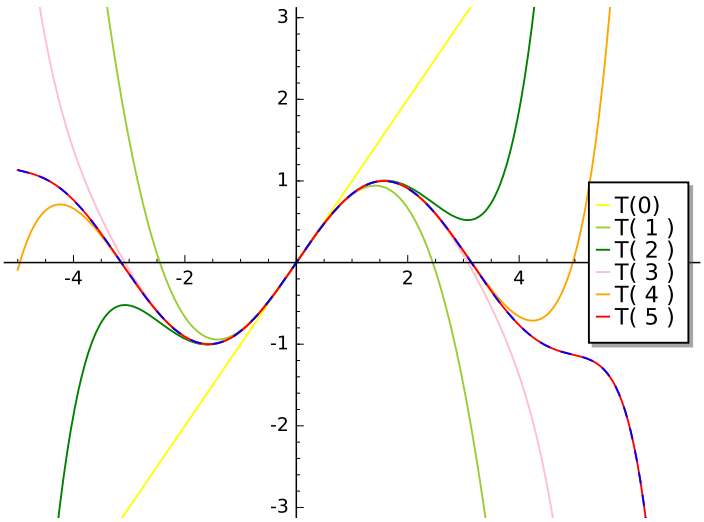
$$f(x) = f(0) + \frac{x}{1!}f^{(1)}(0) + \frac{x^2}{2!}f^{(2)}(0) + \ldots + \frac{x^n}{n!}f^{(n)}(0) + \ldots$$



Here is an example for the function $f(x) = sin(x)$.

```
f=sin(x)  # You can change this function
n=8       # You can change this number
q=plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3, linestyle="--",
 ↪  figsize=5.5)
kolor=[]
kolor=["yellowgreen","green","pink","orange","red","brown","black"]
g=f(0)
for i in range(1, n):
    g=g+(x^i/factorial(i))* diff(f,i).subs(x=0)
    q=q+plot(g,xmin=-5, xmax=7, ymin=-3, ymax=3,
     ↪  color=kolor[(i-1)%7], legend_label=r"T( %d )" % i)
show(q)
```

a plot as in `t_sin`.

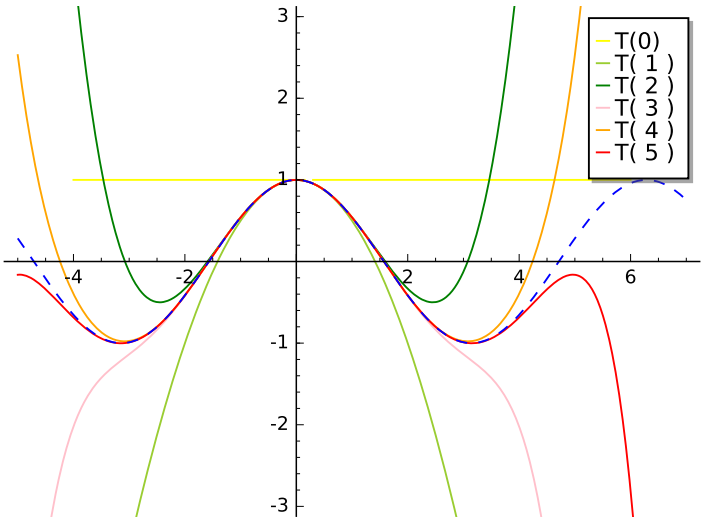


Exercise for students.

For the function $f(x) = cos(x)$ find the corresponding polynomial formula of Taylor-Maclaurin.

```
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 `t_cos`.



Use the Taylor-Maclaurin's formula for function $f(x) = e^x$.

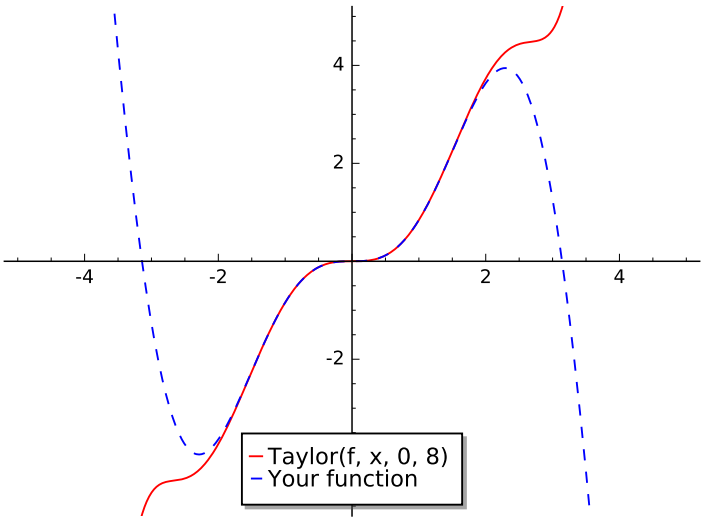
```
kolor=[]
kolor=["yellowgreen","green","pink","orange","red","brown","black"]
n=8
f=1
q=plot(f,xmin=-4,xmax=6, ymin=-3, ymax=3,color="yellow",
 ↪  legend_label="T(0)")
for i in range(0, n):
    k=i+1
    f=f+(1/factorial(k))*x^k
    q=q+plot(f,xmin=-5, xmax=7, ymin=-3, ymax=3,
     ↪  color=kolor[(i-1)%7],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

We already know the Taylor's formula. Now we can simplify our calculations, and use the built-in Taylor's formula in 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 Fig. 2.5.8.



2.6 First order differential equations

2.6.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at XXXIII LO M. Kopernika w Warszawie.

It was prepared by Mirosław Malinowski based on his lesson.

2.6.2 Lesson Plan

Based on the book C. Quinn, C. Sangwin, R. Haese. M. Haese, "Mathematics for the international student. Mathematics HL (Option): Calculus, Haese and Harris Publications 2013.

We will only consider differential equations of the form:

$f(x,y)\frac{dy}{dx} + g(x,y) = 0$, where $y = y(x)$.

These are known as first order differential equations since there is only one derivative in the equation, and it is a first derivative.

A function $y = y(x)$ **is said to be a solution** of a differential equation if it satisfies the differential equation for all values of x ** in the domain of y .

2.6.3 Slope Fields

Given any first order differential equation of the form $f(x,y)\frac{dy}{dx} + g(x,y) = 0$, where $y = y(x)$ we can write

$\frac{dy}{dx} = \frac{-g(x,y)}{f(x,y)} = h(x,y)$

We may therefore deduce the gradient of the solution curves to the differential equation at any point :math:‘(x,y)‘in the plane where :math:‘h(x,y)‘is defined, and hence the equations of the tangents to the solution curves.

The set of tangents at all points :math:‘(x,y)‘ is called the slope field of the differential equation .

By representing the gradients at many different grid points as line segments, we obtain a slope field of the tangents to the solution curves. Now the tangent to a curve approximates that curve at and near the points of tangency. Therefore, by following the direction given by the slope field at a given point, we can approximate solution curves of the differential equation.

Example 1

Consider the differential equation $\frac{dy}{dx} = x(y-1)$.

1. Construct the slope field using integer grid points for $x, y \in [-4, 4]$.

2. Sketch the particular solution curve through $P(1, 2)$. Sketch the particular solution curves through several other points of your choice - adjust the parameters of the graph respectively.

3. An **isocline** is the set of all points on a slope field at which the tangents to the solution curves have the same gradient. Identify any isoclines on your slope fields.

Solution:

```
sage: a,b,x,x0,y0 = var('a b x x0 y0')
sage: y = function('y', x)
sage: de = diff(y,x) == x*(1-y)
sage: general = desolve(de, y).expand()
sage: x0 = 1
sage: y0 = 2
sage: particular = desolve(de, [y,x], [x0,y0]).expand()
sage: html('General solution of the equation is $y=$'
 ↪  "$%s$"%latex(general) + "$")
sage: html('The particular solution of the equation for $y=$'
 ↪  +latex(y0)+ ' when $x=$' +latex(x0)+ ' is $y=$ '
 ↪  "$%s$"%latex(particular) + "$")
sage: p1 = plot_slope_field(a*(1-b), (a,-2,2), (b,-2,3), figsize =
 ↪  (6,4), headaxislength = 3, headlength = 3)
sage: p2 = plot (particular, (x, -2, 2))
sage: p1+p2
```

EXERCISES

Using SAGE solve the following problems. You can modify the procedure given above or write your own.

1. Consider the differential equation $\frac{dy}{dx} = 2x - y$.

1. Find the general solution to the differential equation.

2. Construct the slope field of the differential equation.

3. Find the solution curve which passes through $P(0, 1)$ and its equation.

2. Consider the differential equation $\frac{dy}{dx} = 10y\tan(x)$, where x is measured in degrees. Draw the slope field using integer grid points, where $x, y \in [-2, 2]$.

3. Draw the slope field for the differential equation $\frac{dy}{dx} = \frac{-1+x^2+4y^2}{y-5x+10}$.

1. Sketch the particular solution curve passing through the origin.

2. Sketch the isocline corresponding to:

1. $\frac{dy}{dx}$ being undefined.

2. $\frac{dy}{dx} = 0$.

2.6.4 Separable Differential Equations

Differential equations which can be written in the form $\frac{dy}{dx} = \frac{f(x)}{g(y)}$, where $y = y(x)$, are known as separable differential equations.

We can notice that if $\frac{dy}{dx} = \frac{f(x)}{g(y)}$, then $g(y)\frac{dy}{dx} = f(x)$. Now, integrating both sides of the equation with respect to x we get $\int g(y)\frac{dy}{dx}dx = \int f(x)dx$ and by the Chain Rule we can reduce the problem of solving the differential equation to the problem of finding two separate integrals

$$\int g(y)dy = \int f(x)dx$$

Example 2:

Find the general solution of the differential equation $\frac{dy}{dx} = \frac{x^2y+y}{x^2-1}$. Verify your results with SAGE and sketch the solution curve passing through the point $P(x_0, y_0)$ if:

1. $(x_0, y_0) = (2, 3)$
2. $(x_0, y_0) = (0, 0)$
3. $(x_0, y_0) = (-2, 1)$

Solution:

```
sage: x, x0, y0 = var('x x0 y0')
sage: y = function('y', x)
sage: de = diff(y,x) == (x^2*y+y)/(x^2-1)
sage: general = desolve(de, y)
sage: html('General solution of the equation is
 ↪  $y=$'"$%s$"%latex(general) + '.' + "$")
sage: x0 = -2
sage: y0 = 1
sage: particular = desolve(de, [y,x], [x0, y0]).simplify()
sage: html('The particular solution of the equation is
 ↪  $y=$'"$%s$"%latex(particular) + '.' + "$")
sage: p1 = plot (particular, x, xmin = -3, xmax = 4, ymin = -10,
 ↪  ymax = 10, axes_labels=['$x$','$f(x)$'], exclude = [-1],
 ↪  detect_poles = 'show', figsize = (6, 4), color = 'blue',
 ↪  legend_label="$y =$ $%s$"%latex(particular))
sage: p1
```

EXERCISES

1. Solve the following initial value problems:

1. $(2 - x)\frac{dy}{dx} = 1, y(4) = 3.$
2. $\frac{dy}{dx} - 3x\sec(x) = 0, y(1) = 0.$
3. $e^y(2x^2 + 4x + 1)\frac{dy}{dx} = (x + 1)(e^y + 3), y(0) = 2.$
4. $x\frac{dy}{dx} = \cos^2(y), y(e) = \frac{\pi}{4}.$

2. Solve $\frac{dy}{dx} = \frac{3y-xy}{x^2-1}$, $y(0) = 1$.

2.6.5 Homogeneous Differential Equations

Differential equations of the form $\frac{dy}{dx} = f\left(\frac{y}{x}\right)$, where $y = y(x)$ are known as homogeneous differential equations .

They can be solved using the substitution $y = vx$, where $v = v(x)$. The substitution will always reduce the differential equation to a separable differentiable equation.

Example 3:

Use the substitution $y = vx$, where $v = v(x)$, to find general solution of the differential equation $\frac{dy}{dx} = \frac{x+2y}{x}$. Verify your results with SAGE and find the particular solution if $y = \frac{3}{2}$ when $x = 3$.

Solution:

```
sage: x = var('x')
sage: y = function('y',x)
sage: de = (diff(y,x) == (x+2*y)/x)
sage: general = desolve (de, y).expand()
sage: html('General solution of the equation is
 ↪  $y=$'"$%s$"%latex(general) + '.' + "$")
sage: particular = desolve(de, [y,x], [3,1.5]).expand()
sage: html('The particular solution of the equation is
 ↪  $y=$'"$%s$"%latex(particular) + '.' + "$")
sage: p1 = plot (particular, x, xmin = -3, xmax = 4, ymin = -3, ymax
 ↪  = 10, axes_labels=['$x$','$f(x)$'], detect_poles = 'show',
 ↪  figsize = (6, 4), color = 'blue', legend_label="$y =$
 ↪  $%s$"%latex(particular))
sage: p1
```

2.6.6 The Integrating Factor Method

Suppose a first order linear differential equation is of the form $\frac{dy}{dx} + P(x)y = Q(x)$, where $y = y(x)$.

1. Calculate the integrating factor $I(x) = e^{\int P(x)dx}$. You do not need a constant of integration.
2. Multiply the differential equation through by $I(x)$.
3. Simplify the LHS and hence obtain $I(x)y = \int I(x)Q(x)dx + C$, where C is a constant.
4. Integrate to obtain the general solution.

Example 4:

Solve the differential equation $\frac{dy}{dx} + 3x^2y = 6x^2$. Find the particular solution of the equation for the initial condition $y(0) = 1$.

Solution:

```
sage: var('x y C')
sage: var('dy,dx')
sage: var('x0 y0')
sage: Y = function('Y', x)
sage: de = diff(Y,x) + 3*x^2*Y == 6*x^2
sage: I = e^(integral(3*x^2, x)) #integrating factor
sage: html('1. The integrating factor of the equation is $I(x)=$'
 ↪  "$%s$"%latex(I) + '.' + "$")
sage: de1 =
 ↪  ((I*de).subs({diff(Y,x):dy/dx,Y:y})*dx).full_simplify().expand()
sage: html('2. Multiplying both sides of the equation by the
 ↪  integrating factor ' "$%s$"%latex(I) + ' we get '
 ↪  "$%s$"%latex(de1) + '.' + "$")
sage: RHS = integral(de1.rhs(),x).coefficient(dx) # RHS.show()
sage: LHS = y*I
sage: html('3. Taking integrals on both sides of the equation we
 ↪  get: ')
sage: eqn = (LHS == RHS + C)
sage: eqn.show()
sage: solution = solve(eqn, y)[0].expand()
sage: html('4. Therefore, general solution of the equation is '
 ↪  "$%s$"%latex(solution) + '.' + "$")
sage: x0 = 0
sage: y0 = 1
sage: particular = desolve(de, [Y,x], [x0,y0]).expand()
sage: html('The particular solution of the equation for $y=$
 ↪  '+latex(y0)+ ' when $x=$ ' +latex(x0)+ 'is
 ↪  $y=$'"$%s$"%latex(particular) + '.' + "$")
```

PROBLEM

Using the above procedure solve the initial value problem $\cos x\frac{dy}{dx} = y\sin x + \sin 2x$, $y(0) = 1$.

EXERCISES

1. Solve the following using the integrating factor method:

1. $\frac{dy}{dx} + 4y = 12$.
2. $x\frac{dy}{dx} + y = x\cos x$.
3. $\frac{dy}{dx} - 3y = e^x$, $y(0) = 2$.
4. $\frac{dy}{dx} + y = x + e^x$, $y(1) = 1$.

2. Solve the differential equation $(x + 1)y + x\frac{dy}{dx} = x - x^2$.

CHAPTER 3

Teaching materials: Physics

3.1 Examining string vibrations

3.1.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Adam Ogaza based on his lesson.

3.1.2 Introduction

The aims of lesson:

- Experimental determination the dependence of frequency on the length of the string.
- Use Python to carry out the graphical data analysis.

Educational aids:

- String stretched on a ruler / resonance box / any string instrument
- Acoustic generator
- Loudspeaker
- Phone application measuring frequency of sound
- Phone application generating sound of a given frequency

Used methods:

- Preliminary computer exercises in drawing plots in Python and fitting straits and curves to experimental data.
- Lecture - as a theoretical introduction.
- Exercises in groups - measurements of vibration frequencies for string of different lengths.

This lesson is addressed to students of age 17, learning at advanced level. It fulfills the following requirements included in National Curriculum:

- General requirement V: "planning and executing simple experiments and analyzing their results".
- Detailed requirement 6.8: "student uses in calculations the relations between wave parameters: length, frequency, period and velocity"
- Detailed requirement 6.12: "student describes standing waves and their dependence on waves running backwardly".
- Intersection requirement 12.2: "student single-handedly makes correct graphs".
- Intersection requirement 12.5: "student fits a straight line y = ax + b to the graph and calculates the coefficients a and b".
- Experimental requirement 13.6: "student examines string vibrations (for example the dependence of frequency on the length), does the measurements, description, data analysis and graph interpretation"

The described topic is a simple example of a perfect integration science with IT and teaching English. Therefore it must be carried out both in computer laboratory, physics laboratory and English class-room.

3.1.3 IT Part

The experimental lesson must be preceded by a basic course of Python, including drawing and formatting graphs and fitting functions to measuring points. Polish National Curriculum requires only fitting straight lines y = ax + b and counting the coefficients a and b. Python allows to fit any curve without any effort, so it is a good opportunity to play with hyperbola. The additional skills needed to complete the homework assigned at the end of experimental lesson are: formatting text in SAGE, inserting pictures and making tables in the text. It is advisable to perform with students the following exercises:

1. Insert any photo into SAGE document:

2. Create and format a hypothetical table:

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. Create and decorate a dot diagram of it.

```
sage: # Creation the dot diagram
sage: l = [5, 10, 15, 20, 30, 40, 50, 70, 90]
sage: y = [1750, 879, 589, 444, 309, 243, 185, 122, 99]
sage: point(zip(l,y))
```

```
sage: # Methods of decorating the diagram
sage: pkt=[(l[i],y[i]) for i in range(len(l))]
sage: point(pkt, gridlines=True, size=25, color='red',
 ↪  axes_labels=['l', 'y'], legend_label='y(l)')
```

Tabular output of data (instead of print)

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

4. Fit a hyperbola to above points

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

5. Assume, that l represents the length of string and y stands for the frequency. Draw the graph T(l), where T is the period and fit a straight line to it. Examine, whether forcing the straight to go through the origin of coordinates changes much.

```
sage: pktinv=[(l[i],N(1/y[i], digits=4)) for i in range(len(l))]
sage: print pktinv
sage: var ('a, b, c')
sage: straight(x) = a*x+b
sage: straight0(x) = c*x
sage: fit = find_fit(pktinv, straight,solution_dict=True)
sage: print fit
sage: fit0 = find_fit(pktinv, straight0,solution_dict=True)
sage: print fit0
sage: rys1=plot(straight.subs(fit), (x, 0, 90), color="green",
 ↪ legend_label='fitted straight line')
sage: rys0=plot(straight0.subs(fit0), (x, 0, 90), color="yellow",
 ↪ legend_label='going through 0')
sage: rys2=point(pktinv, gridlines=True, size=25, color='red',
 ↪ legend_label='measuring points', axes_labels=['l [cm]','T
 ↪ [s]'])
sage: rys1+rys0+rys2
```

3.1.4 Experimental Part

At the beginning the teacher ask pupils to recall the ideas of standing waves, frequency, wave length and phase velocity. Next, he derives the formula $f\ (\ l\ )$, where l is the length of string. It shows, that this two variables are inversely proportional. Checking this dependence is the main aim of this lesson.

Next, students are divided into groups. Each group chooses one instrument. It may be a string stretched on a ruler, string stretched on a resonance box or any real music string instrument. Groups move far away each from other to minimize the mutual disturbance.

In each group somebody is responsible for putting the string into vibrations. Simultaneously, somebody else generates sound using software generator. It has a property of fluent adjusting the frequency. When the group judge, that the tone coming out from the loudspeaker is the same as from the string, they note the length of string and frequency. This measurement is repeated

for different length of the oscillator, in the most possible range. Alternatively, students may make independent measurements, using software applications in their smartphones.

All collected data are noted in a table in notebooks. Students are encouraged to make photos. The teacher announces, that this data and photos will be used in a homework.

3.1.5 Homework

Write in SAGE a short report of the executed experiment, including description of the essence of the matter, measuring system (with photo), used tools, performed activities, obtained results and inferences. In particular, the report should prove or refute the hypothesis, that frequency of vibrations is reciprocally proportional to the length of string.

The data are shared for the whole groups, but all reports must be independent and individual. The authors of best reports will have the right to translate their works into English and publish them, gaining additional points for it. English teachers are ready to assist and supervise pupils in translation.

3.1.6 Evaluation Report

At first time this lesson was carried out in the spring 2015. All students made the same measurement. The whole group (14 person) is visible in the above picture. All homeworks were based on the same set of data, but I asked students to work individually at home and create unique reports. In fact, obtained reports differed in level and tools used. Students usually admitted, that using Sage helped them much. They discovered that Sage is a convenient environment for that tasks.

The second edition of the lesson *Examining String Vibrations* was accomplished 5-th April 2016 (both IT and experimental parts). Students were divided into 5 groups of four people each. One girl brought her own violin, somebody else used her own guitar. The remaining 3 groups were equipped with instruments from my laboratory, i.e. a string stretched on resonance box and two strings stretched on a ruler. All groups were equipped with software acoustic generators and applications measuring the dominant frequency of heared sound. Some groups stayed in the classroom, whereas the others went to corridor to carry out their measurements far away from noises generated by other groups.

After tentative calculations it occurred, that 4 groups succeeded in their measurements, whereas one encountered disturbances by voices coming from the neighbor group. They asked me to repeat this experiment during additional classes. Finally, all students obtained reasonable results and wrote quite good reports.

After two weeks we met again in the IT laboratory. By this time I had made a preliminary assessments of the works. I explained my remarks and advised, what could be improved. After next week I set the final assessments. Five works got the highest grades and moved on to the second step. Their authors were granted a possibility to translate the works into English and publish them for extra points. Earlier I asked English teacher for cooperation. They supervised the translations and also made their own assessments of the language. All the awarded homeworks are published at sage01 server.

This lesson was a great success. Students liked it very much. I managed to integrate physics not only with IT, but also with English. Language teachers appreciated my initiative and reported, that it was an interesting experience both for them and for the students. They teach a subject called *technical English for engineers* and this was its live, practical application.

I will not describe the individual student's conclusions, because they are all included in the original works.

The last edition (in April 2017) was the most successfull. Next generation of students became skilled in attaching errorbars to graphs, using Numpy library. Some of them brought their own professional music equipment and made really serious scientific investigation. Like one year earlier, I qualified the best works for translation, which students did on their own perfect. The best homeworks are attached in separate files.

Work 1, Polish version: https://sage01.icse.us.edu.pl/home/pub/184/

Work 1, English version: https://sage01.icse.us.edu.pl/home/pub/179/

Work 2, Polish version: https://sage01.icse.us.edu.pl/home/pub/170/

Work 2, English version: https://sage01.icse.us.edu.pl/home/pub/172/

3.2 Sound Waves

3.2.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Adam Ogaza based on his lesson.

3.2.2 Introduction

The lesson described below was tested three times on students learning physics at advanced level (age 17). In 2015 the numerical amount of group was 14, in 2016: 21 and in 2017: 16. This lesson was preceded by a short course of programming in Python and consists of two parts: theoretical introduction and IT laboratory. Both parts were filmed in 2015 and published on Youtube with English subtitles.

The main aims of this lesson are:

- Explaining, what is sound.
- Explaining, what is acoustics, how it is divided and why?
- Defining all physical and physiological quantities describing sound.
- Giving a basic knowledge of sound spectrum.
- Exercising in plotting diagrams in Python.
- Showing, that any periodical function can be presented as a linear combination of sine functions.
- Exercising interaction and sliders in Python.

Preparing this lesson we should take into consideration the following circumstances:

- At high school level pupils cannot integrate. They have heard about integral and its applications, but performing integrations at the level required by Fourier transform is far beyond their abilities. Therefore real calculations must be replaced by simple playing with amplitudes of harmonics.
- It is the first students' contact with @interact and sliders. Technical problems themselves are heavy to overcome, so there is no point in making the physical side of the topic more difficult.
- This lesson (like any other) should be attractive, so spectacular results should be reached by relatively simple means.
- The level of difficulties of the examples should gradually grow during this lesson. A good idea is to show different solutions for the same problem.

3.2.3 Theoretical part

Exercises in programming were preceded by a theoretical lecture about sound waves, published at: https://youtu.be/dp-ajKHs6WU

Main problems discussed in this lecture are the following:

- Definition of sound wave.
- Announcement, that some ideas from this lecture will be developed further during IT lesson with the use of SAGE and Python
- Definition of acoustics and explanation the essence of physical and physiological acoustics.
- Infrasounds and ultrasounds.
- Frequency / wavelength and their relation to the tone
- Spectrum of sound and its relation to the timbre.
- Weber - Fechner law.
- Sound intensity level and its relation to the loudness (audiogram).
- Phase of the sound wave and its relation to the spatial impressions.

During this lesson a software acoustic generator is used to demonstrate the tone output with respect to the frequency. It can also show, how changes of shape of signal influences the sound spectrum and timbre of voice. Students are informed, that they will use Python to illustrate the shape of complex signal with respect to the amplitudes of individual harmonics.

3.2.4 IT part

The IT part of this lesson was executed in IT laboratory, filmed and published at: https://youtu.be/0fVgRy6CpWQ

This film, made in 2015, shows the previous version of programming. After some improvement it looks as follows:

```
sage: wall = 20 # limit of domain = end of space accessible for
 ↪  waves
```

Hypothetical complex wave, defined in a static manner

```
sage: amplitudes = (1, 1/2, 3/10, 1/5, 1/10)
sage: WaveComplex(t) = sum(a*sin((n+1)*t) for n, a in
 ↪  enumerate(amplitudes))
sage: WaveComplex(t)
1/10*sin(5*t) + 1/5*sin(4*t) + 3/10*sin(3*t) + 1/2*sin(2*t) + sin(t)
```

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

The same as above, but obtained with different tools.

```
sage: 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)
sage: WaveComplexPlot(tmax=wall, figsize=(6, 3))
```

Another version of the same function, but the individual amplitudes are now controlled by sliders.

Because students are working on server sage03 with version 6.4.1 of SAGE, we cannot use the ready function histogram. Instead I propose my own version.

```
sage: def WaveComplexPlot(A1=1, A2=0.5, A3=0.3, A4=0.2, A5=0.1,
 ↪ **kwarg):
....:     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)
sage: WaveComplexPlot(figsize=(2,1))
```

```
sage: @interact
sage: 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=(1,1))
....:     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)
....:     show(histogram)
```

Decomposition of an example function with the use of Fourier transform.

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

```
sage: Piecewise?
```

Similar analysis extended to a sawtooth-like function as well as the functions used with the sound generator as shown in the video.

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

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

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

With the use of interact, the code segments above could be combined as follows. Students may now play with different input signal forms:

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

We can also control the number of iteration. In the example below, the loop superimposes plots of functions and displays the sum together with the components:

```
sage: @interact
sage: 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)
```

Now the loop creates a complex wave built with harmonics of amplitudes inversely proportional to their frequencies. The number of iterations is controlled by a slider. The functions shown above are added each to other and the sum is plotted.

```
sage: @interact
sage: 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()
```

The same effect, but without any interaction.

```
sage: def WaveCmplx(t):
....:         w=0
....:         for i in range(1,10):
....:             w=w+1/i*sin(i*t)
```

```
....:         return w
sage: plot(WaveCmplx,  (t, 0, wall), figsize=(4,2))
```

3.2.5 Conclusions

Using Python led pupils to better understanding the essence of sound spectrum. In high school pupils don't know Fourier transform - it is beyond the curriculum. It is a big discovery for them, that any periodic function, which has the correct symmetries (which is the case in the context of acoustic waves), can be presented as a combination of sine functions.

From the other side, the participants of the lesson had an opportunity to learn the basic methods of plotting in interactive manner, which was new for them. The topic "sound spectrum" became less abstract, because students had it "in their hands".

My observation is, that students are happy when using ready tools included in this worksheet, but are not able to develop them. My attempts to encourage students to improve above code as a homework failed.

3.3 Acoustic Effects

3.3.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Adam Ogaza based on his lesson.

3.3.2 Introduction

The main aims of this lesson are:

- Explaining, what is echo and after-sound.
- Demonstration of acoustic resonance.
- Explaining the principle of operation of music instruments.
- Explaining, what are beats,
- Exercising in plotting diagrams in Python.
- Exercising interaction and sliders in Python.
- Exercising animations in SAGE

Circumstances taken into consideration:

- Students have already basic knowledge of waves in general. In particular they know the equantion of wave, idea of standing wave and resonance.
- It is the second students' contact with animations in Python. They participated in one IT lesson devoted to this problem.
- One should remember, that student don't know higher mathematics, so presented IT solutions must not be too sophisticated.
- Gradation of difficulty level is recommended. One should start with simple examples and gradually complicate them (for example by adding more parameters or varying constants)

All described below can be carried out during two hours: one in physics laboratory, the second in a class-room equiped with sufficient amount of computers

3.3.3 Theoretical part

Exercises in programming were preceded by a theoretical lecture about acoustic effects. This lecture was filmed and published in Polish but with English subtitles.

https://youtu.be/jWGTTD5-mFA

Main problems discussed in this lecture are the following:

- Echo
- After-sound
- Acoustic resonance
- Standing waves (especially in instruments)
- Beats

I will not describe the details here, because all of them are visible in the film.

3.3.4 IT part

Main skills exercised during lesson:

- Processes controlling with sliders.
- Puting diagrams into motion.
- Examining the motion of wave impulses.
- Composition of vibrations.
- Examining standing waves and beats.

The physical problems discussed in the code and diagrams below, are the following:

- Beats - how does the shape of composite wave depend on component frequencies.
- How to animate a wave impulse and its reflection.
- Showing, that standing wave is really a result of interference betwee two waves travelling in opposite directions.

Beats

Easy example of plot with fixed frequencies and initial phase, which can be changed inside the code. Results of those changes can be observed on the plot.

```
# **kwarg allows adding additional "called" arguments
# dudnienia (beats) is a function returning a plot of superimposed
  ↪  functions y1 and 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))
```

More complex drawing including the parameters of functions and envelope. For simplicity the amplitudes are chosen to be equal.

```
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)
    trange = (t, 0, 10)
    plt = plot(y, trange, 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, trange, color='red', **kwargs)
        plt = plt+plot(-envelope_func, trange, color='red',
         ↪  **kwargs)
    return plt
beat(t0=2.5, figsize=(4, 2))
```

A slider controlling the time shift. Enables moving the picture.

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

Controlling the frequency of second wave. One can onserve the dependence of pulsation period on the difference between frequencies of component waves.

```
@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)
```

Creation of graphics array for further animations.

```
# plts - it is a collection of plots for time argument t0 iterated
 ↪  in a loop.
plts = [dudnienia(t0=t0_,figsize=(4,2)) for t0_ in
 ↪  srange(0,6.3,0.2)]
# two plots shown as an example
show(plts[0])
show(plts[10])
```

```
# Preparation for the next box
anim = animate(plts)
```

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

Ready animation. We see the sequence of plots for different time arguments.

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

Another approach, making use of plots defined above

```
animate(plots).show()
```

Animation showing the impact of second frequency on the pulsation period.

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

Reflection and standing waves

Standing wave as an interference of two waves travelling the opposite side. Students can manipulate all the parameters (it was an exercise).

```
# Semi-automatic color control in the loop
A=1
omega=6
v=13
delay=30
t_max= 7
# This plots show time evolution of standing wave. Students may
 ↪ manipulate the constants
sum(
 ↪ [plot(A*sin(omega*(t/delay-x/v))+A*sin(omega*(t/delay+x/v)),(x,0,20),figsiz
 ↪ for t in srange(0,t_max,1.0)] )
```

In order to make the time evolution of the standing wave more apparent, an animation is very useful.It might also be instructive to display the travelling waves.The variable delay seems to be unnecessary and the argument *x-vt* is used instead of *t-x/v* so that no problem arises when v goes to zero.

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

Graphics array useful for a more detailed analysis.

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

Wave pulse and its reflection

Motion of wave impulse. I know, that so defined impulse is not physical because of differential discontinuity, but it is the first approach to such kind of animations. Students were informed of the physical doubts.

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

Two incoming wave impulses coming from opposite directions

```
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)
```

```
14
15      plt.show()
```

Superposition of impuls and its reflection

```
1  var('x')
2  A1 = 1
3  c = 3.4
4  nl = 2
5  nL = 4
6  k = 4*pi # 2pi/wavelenght
7  @interact
8  def _(t=slider(0,10,2*pi/k/64)):
9      x0 = -nL*2*pi/k
10     x1 = (nL-nl)*2*pi/k
11     plt = Graphics()
12     plt += plot( lambda
         ↪  x:pulse1(k*(x-x0)-c*t)+pulse1(k*(x-x1)+c*t),(x,x0,0),figsize=(12,4),thi
13     plt.show()
```

Numerical wave reflection

```
1  %time
2  import numpy as np
3  N = 4048
4  l = 50.
5  dx = float(l)/(N-1)
6  c2 = np.ones(N)
7  dt = 0.005
8  print np.sqrt(np.max(c2))*dt/dx
9  x = np.linspace(0,l,N)
10 u = np.zeros(N)
11 u0 = np.zeros(N)
12 unew = np.zeros(N)
13 ulst=[u.copy()]
14 n=4.
15 T = 1.*l/n
16 for i in range(25000):
17     unew[1:-1] = 2.*u[1:-1] - u0[1:-1] + dt**2
         ↪  *(c2[1:-1]/dx**2*np.diff(u,2))
18     u0=u.copy()
19     u=unew.copy()
20
21     u[-1] = u[-2]
22     u[0] = u[1]
23
24     u[-1] = 0
25     u[0] = 0
```

```
26
27      if dt*i/T*2.0*np.pi< 4*np.pi:
28          u[0] = 0.5*np.sin(dt*i/T*2.0*np.pi)
29
30      if i%50==0:
31          ulst.append(u.copy())
```

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

```
1  len(ulst)
```

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

3.3.5 Conclusions

Programming in Python became an interesting supplement of physics lesson. The benefit lies in the possibility of (more or less) easy visualisation of processes with the change of parameters. Thank this, the formulae presented during lectures are less abstract and everybody can single-handed check, how the result of interference depends on frequencies, direction of speed and so on.

Animations itself were a new challenge for students. In the previous version I asked student to develop somehow the code to create some new ideas. It failed, because the code was too difficult for them. All they are able to do is to manipulate the parameters.

Nevertheless, I judge this lesson high. Students were very interested and engaged. Their skills, both in physics and informatics, increased.

3.4 Accelerated motion

3.4.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Adam Ogaza based on his lesson.

3.4.2 Introduction

The aim of the lesson is examining the accelerated motion, in particular sampling, whether this motion is uniformly accelerated, i.e. the acceleration is constant, and determining the value of acceleration. It requires a sequence of measurements in repeatable conditions. Python will be used for processing a big amount of data and conducting graphical analysis. The concept of the topic predicts the following stages of proceeding:

1. Carrying out a lesson in physical classroom, including theoretical lecture about uniformly accelerated motion and demonstration of proper experiment.
2. Carrying out a lesson in IT classroom about graphical data analysis, fitting straights and curves to measuring points and drawing error-bars.
3. Setting a homework: performing a similar exercise and work out data using methods introduced during IT lesson.

The order of first two points may be switched.

This document includes general speculations. Details of experimental lesson are contained in attached film (English subtitles available). Exemplary homeworks are enclosed in separate files.

3.4.3 Experimental part

Teaching resources

1. Any system for repeatable examining accelerated motion. The best is slightly inclined air track. An ordinary inclined plane and globule or trolley may work as well.
2. A device for distance measurements.
3. Set of devices measuring time. Student's smartphones are good for this purpose.

The course of the lesson

The whole course of lesson, description of experiment and expectation towards the way of data processing are contained in the film: https://youtu.be/deTJ4i1V0dg In particular it shows the pattern of measuring chart, tips on the way of conducting the experiment and formulas necessary for calculations. Presented calculation methods basically exceeds the Curriculum,

but thanks to Python they are very easy in use and students immediately gain spectacular results without any need of deep understanding all the mechanisms.

In simple words, the suggested experimental method requires marking on the plane a sequence of scores in a certain distances from the starting point and conducting several series of measurements of motion time from starting point to each particular score. The distance measurement accuracy (single measurement) is estimated as a scale interval (thickness of the chalk). Times for each score are averaged, and as their uncertainty we assume the standard deviation for each score. The uncertainty of complex measurements (square of time, acceleration) is calculated by logarithmic derivative method.

This experiment should be reported in writing, according to general recommendations published in PDF form on the school web site.

3.4.4 IT part

The following section does not contain a complex problem solving, but only a set of tips, how to do different things. The rest remains for student's invention. In case of problems, enclosed exemplary student's homeworks may be used as a pattern. All data below are fabricated and their purpose is to show, how certain Python instructions work.

Exemplary calculation of standard deviation. Traditional calculation is arduous, in Python it comes down to execute a banal instruction:

```
anydata = (3.4, 3.6, 3.3, 3.3, 3.5, 3.7, 3.6, 3.6, 3.6, 3.5) #
 ↪ przykładowe dane
mean_data = mean(anydata) # mean value
std_data = std(anydata) # standard deviation
print anydata
print "Mean value = ", N(mean_data, digits=3)
print "Std deviation = ", N(std_data, digits=3)
```

Measuring data simulation and plot $s(t)$

```
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',
 ↪ 'y'], legend_label='s(t)')
```

Fitting parabola to data. Without Sage environment it is absolutely infeasible for students. Here it comes down to a few simple instructions. The formula describing the parabola is written in such a manner, that the result of fitting gives directly the acceleration value. Pupil doesn't need to understand, how it works. He is convinced of proper result, seeing plot closely adhesive to measurement points.

```
var ('a')
parabola(x) = a/2*x^2 # Time is marked as x, to avoid collision in
 ↪ variables
fit = find_fit(pkt, parabola,solution_dict=True)
print fit
rys1=plot(parabola.subs(fit), x, 0, 6, color="green",
 ↪ legend_label='fitted parabola')
rys2=point(pkt, gridlines=True, size=25, color='red',
 ↪ legend_label='measurement points')
rys1+rys2
```

Python is not equipped with mechanisms for automatic drawing error-bars. Additional library *matplotlib* is needed. Points are connected by broken line (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')
```

Unfortunately, mentioned library does not draw function plots. To put on one plot both error-bars and fitted parabola, the following construction must be used. Function *linspace* scales the time axis (in this case from 0 to the last point t[7]) and determines the sampling density (here: 100 points - the more the curve is smoother).

```
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')
plt.savefig('1.pdf')
```

3.4.5 Homework

Construct any system for observing the accelerated motion. Conduct series of measurements of travelling time for different distances in this motion. Write a report in Sage notebook, including:

1. Theoretical introduction
2. Description of experimental system (with picture) and performed actions
3. Results of measurements and calculations
4. Graphical data analysis - plots $s(t), s(t^2)$ and $a(t)$ with error-bars and fitted straights / curves
5. Discussion of results - one should adjudicate by different means, whether the acceleration in observed motion was constant.

Detailed recommendations concerning writing reports are familiar for pupils, as they were published on school website in an PDF document many years ago.

3.4.6 Remarks concerning implementation

This scenario was tested in the years 2015-2017 on three age groups of pupils of second class high school (age 17) learning physics at extended level. Earlier, before starting the project, similar experiment has been accomplished for many years, but reports were written in paper version. Python made easier the data processing and chart plotting and made feasible the calculation of parameters of straights / curves fitted to experimental data. Earlier it was impossible. If quantities were directly proportional, student could only apply a ruler to plotted points and check, whether the line goes through all error-bars.

The pupil's approach to project requirements was various. The middle age-group was the worst. It was the only class of science profile, which, because of some changes in the schedule, didn't realize the full extension in IT science. Some of the pupils declared open aversion to programming. Evident benefits of IT use didn't convince them. Therefore I agreed for hybrid solution - making calculations in Python and writing paper reports (with Python charts as attachments).

Pupils of first and third age-group didn't demonstrate any resistance in using IT technologies in full range. They didn't see also any obstacle in the necessity of learning Latex, needed in writing formulas in the reports. I also established cooperation with English teachers and encouraged the authors of the best reports to translate their works into English. For their effort pupils were granted additional points both in physics and in English.

Links to exemplary works:

Polish version: https://sage01.icse.us.edu.pl/home/pub/148/

English version: https://sage01.icse.us.edu.pl/home/pub/177/

Features of good report: http://3lo.edu.pl/?p=306

3.5 Collisions

3.5.1 About this lesson plan

This is a lesson plan for indented for realization during 2h lesson activities.

It has been developed during work in iCSE4school project based on lesson carried out in 2015-2017 at The Stefan Batory High School in Chorzów.

It was prepared by Adam Ogaza based on his lesson.

3.5.2 Description of the problem

The experiment presented in this document has been accomplished during facultative activities and has the status of optional task. Interested pupils might conduct it, film it and describe in a report, additionally evaluated.

The task is to analyze a central collision of 2 bodies of known masses on air truck. The original gliders of air track were equipped with elastic metal bumpers. Because I noticed, that during collisions part of the energy dissipates, causing permanent deformations of bumpers, I replaced them by repelling neodymium magnets. One can expect, that now the collisions will be perfectly elastic, so that both momentum and kinetic energy will be conserved. Such experiment gives also opportunity for detailed investigation the nature of magnetic forces.

Collisions lasted about 3 seconds and were filmed using HD cameras. Exemplary film is available here: https://youtu.be/JSpKctrX_YM

The pupil's task was to read out (frame by frame) positions of both gliders from the air track's scale. There were a few gliders to choose from, each of different mass. The variables x_1 and x_2 denote coordinates of the ends of gliders' grounds turned to each other. The magnets glued to gliders jutted by d_1 and d_2, so that the distance between poles was less from $|x_2 - x_1|$ by $d_1 + d_2$. The set of experimental data was the following:

- Gliders' masses: m_1 and m_2 from about 0,17 kg to over 0,4 kg
- Sizes of magnets and their fastening: d_1 and d_2, about 0,01 m
- Time was calculated iteratively and, depending on the camera, was increasing with the step 1/25 s or 1/30 s
- Gliders' coordinates $x_1(t)$ i $x_2(t)$ red out from the film and expressed in meters. If gliders were moving towards decreasing values on the scale, their velocities and momentums were negative.

On the basis of above data, one should draw and interpret the following charts:

- $x_1(t)$ and $x_2(t)$ - gliders' positions as a function of time.
- $v_1(t)$ and $v_2(t)$ - gliders' instantaneous velocities in particular frames. When points of charts created a chaotic cloud (data noising), I advised to take averages from 2 neighbor frames.

- $p_1(t), p_2(t)$ and $p_c(t)$ - momentums of particular gliders and the whole system. One should check, whether the total momentum was conserved.
- $E_1(t), E_2(t)$ and $E_c(t)$ - gliders' kinetic energies and their sum. One should check, whether the total kinetic energy was conserved.
- $a_1(t)$ and $a_2(t)$ - gliders' accelerations.
- $F_1(t)$ and $F_2(t)$ - forces applied to gliders.
- $F(r)$ - dependence of magnetic force on the distance between poles.

3.5.3 Remarks concerning implementation

Data of first student

It was a trailblazing performance of this experiment in April 2016, done by a single volunteer from the firs age-group of pupils involved in the project.

Results of measurements and charts $x_1(t)$ and $x_2(t)$:

```
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=[1.005,1.005,1.005,1.005,1.006,1.006,1.006,1.006,1.007,1.007,1.007,1.007,1.00
x2=[1.720,1.710,1.695,1.680,1.665,1.650,1.635,1.620,1.605,1.589,1.573,1.559,1.54
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_lab
xt
```

Charts $v_1(t)$ and $v_2(t)$:

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

We see a big noise caused by a limited resolution in reading out the coordinates in time. A small fluctuations of position increases in respective frames, invisible on position charts, are enough to spill the chart of velocity. The data noise devolves (and increases) into charts of momentums, kinetic energies (here we have a square of speed), accelerations and forces. I advised to take average speed for every two neighbor frames:

```
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_lab
vt
```

The noise decreased, but it refers also to the amount of chart points and time resolution of investigated phenomenon. We lose the most interesting processes going on in a short while during the closest gliders' approach.

The pupil made the rest of charts, wrote the report and drew conclusions, but in the cloud of measurement points it was difficult to discern any interesting details. The noise can be reduced by advanced mathematical means, far away from high school pupil's capabilities.

Data of second student

In October 2016 the whole group of next pupils age-group filmed their collisions. They tried to investigate different cases, varying the gliders' masses, values and senses of velocity or putting one glider motionless (as a target). I dish up data from the best elaboration (referring to film quoted above).

Results of measurements and charts $x_1(t)$ and $x_2(t)$

```
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, 0.207, 0.217, 0.227, 0.237, 0.247, 0.257, 0.266,
 ↪ 0.276,
0.286, 0.296, 0.306, 0.316, 0.325, 0.335, 0.345, 0.354, 0.364,
 ↪ 0.374,
0.383, 0.393, 0.403, 0.413, 0.422, 0.432, 0.442, 0.451, 0.461,
 ↪ 0.471,
0.480, 0.490, 0.500, 0.509, 0.519, 0.529, 0.538, 0.546, 0.554,
 ↪ 0.558,
0.560, 0.561, 0.562, 0.563, 0.5635, 0.5638, 0.564, 0.5645, 0.565,
 ↪ 0.5655,
0.566, 0.5665, 0.567, 0.5675, 0.568, 0.568, 0.5685, 0.569, 0.569333,
 ↪ 0.569666,   0.567]
x2 = [0.845, 0.837, 0.83, 0.823, 0.816, 0.809, 0.801, 0.793, 0.786,
 ↪ 0.78,
0.774, 0.767, 0.76, 0.754, 0.746, 0.739, 0.733, 0.726, 0.72, 0.713,
 ↪ 0.705,
0.699, 0.692, 0.686, 0.68, 0.673, 0.666, 0.66, 0.653, 0.647, 0.64,
 ↪ 0.633,
0.628, 0.621, 0.616, 0.61, 0.604, 0.602, 0.605, 0.615, 0.629, 0.644,
 ↪ 0.66,
```

```
0.677, 0.691, 0.706, 0.723, 0.738, 0.754, 0.77, 0.786, 0.802, 0.818,
 ↪ 0.833,
0.848, 0.863, 0.878, 0.892, 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
```

Charts $v_1(t)$ and $v_2(t)$

The averaging over neighbor frames was applied immediately, to reduce the noise.

```
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
```

I will not depict the way of creating further charts, because from the IT point of view there is nothing revelatory in it. Pupils are able to write the proper code by themselves. It is enough to know, how loops work, how to create charts and fit functions to measurement points. In case of doubts, the original student's homework will help:

Polish version: https://sage01.icse.us.edu.pl/home/pub/146/

English version: https://sage01.icse.us.edu.pl/home/pub/147/

I will add only, that I do not share all the presented there final conclusions. Furthermore, in my opinion, the author quite unnecessary fitted 12th degree polynomials to the charts of velocity. I do not know, what was it for.

3.5.4 Conclusions

The presented experiment was one of most interesting in my professional career. Because of the multiplicity of different situations (arbitrary velocities, more gliders to choose, one could obtain quite different results. Students had freedom in drawing conclusions, it was their independed research work. For example, in the quoted work, at the chart of kinetic energy there is an evident minimum at the moment of closest approach of gliders. Student interpretent it as a measuring error caused by too rapid change of speeds. In my opinion, it is a moment, when kinetic energy partially remolded into energy of magnetic interactions. But why, in such case, there is also a visible small breakdown at the total momentum chart?

This experiment basically pertained to pure mechanics, but by the way it gave an opportunity to explore the nature of magnetic force. Students could fit any curves to the chart $F(r)$ - I did

not impose ready solutions. One should look at the data and figure out, what type of curve will be the most suitable.

How it can be seen from above analysis, the precision in reading out the positions of gliders at individual frames, is essential. It is not easy and requires use of good camera. The data noise may be partially removed, but it decreases the resolution, in which we see the whole phenomenon.

The added value of the whole enterprise was a fruitful cooperation with English teachers. They supervised translations done by authors of the best homeworks. Students in science profile classes realize an additional subject called *English for engineers*. Experiment in physics created opportunity to exercise technical language on living example and get additional assessment for it.