Funkcja 3-D

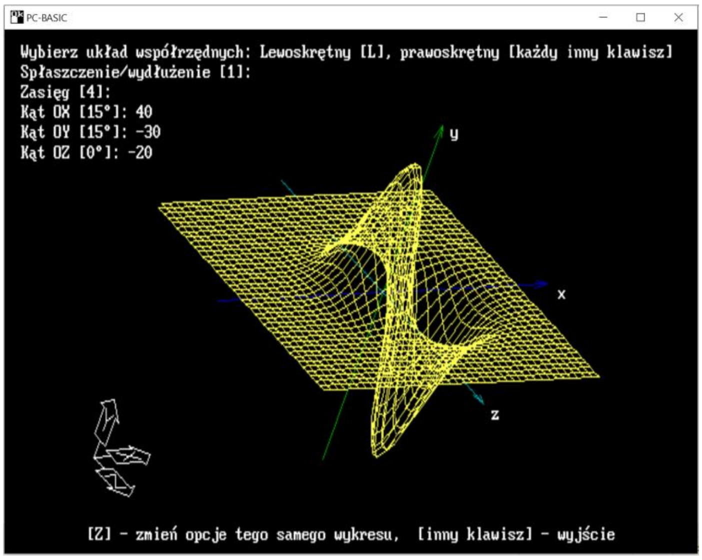
To mój pierwszy program na PC z 1991 roku. Wykonanie poniższej grafiki zajęło procesorowi Intel 80283 pełne dwie minuty. Program wysłałem w tym samym roku do czasopisma *Bajtek* (dodałem przykład rysunku sprzęgła Oldhama). Odebrano go, ale nie publikowano. Była to moja reakcja na nieudolne próby tworzenia grafiki 3-D w tym czasopiśmie.

Poniżej jest obraz graficzny funkcji:

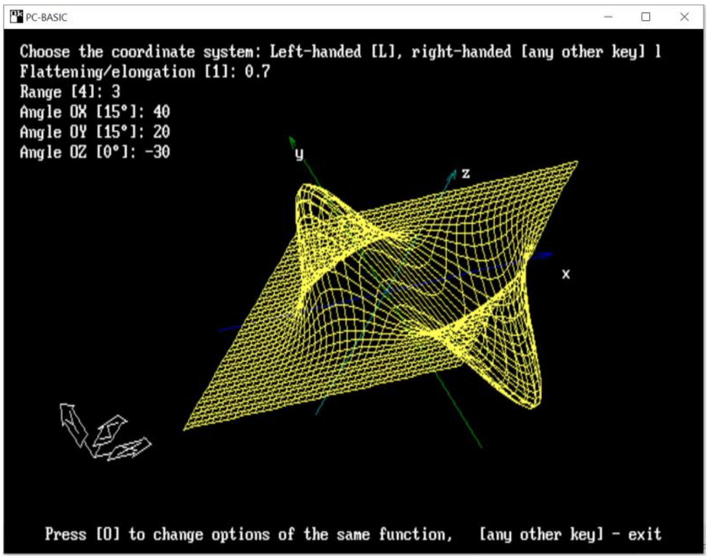
```
V=-(X*X+Z*Z)
VV=-((X-2)*X+(Z-.5)*Z)
Y=5*EXP(V)-2*EXP(VV)
```

czyli $y = f(x,z) = 5*e^{-(x*x+z*z)}-2*e^{-((x-2)*x+(z-.5)*z)}$

Wersja polska:



Wersja angielska:



Niestety proces obsługi błędów nie jest w GW-Basic'u dopracowany. Podaję dwie wersje kodu spodziewając się takich samych wyników:

```
10      <-- linia FOR poniżej - dwie wersje
20       PRINT "I=" I, "log(i)=" LOG(I);
30       ON ERROR GOTO 60
40       PRINT "        Nie ma bledu (No error)"
50       GOTO 90
60       A=ERR: B=ERL
70       PRINT: PRINT "Numer bledu (Error number)=" A "   Linia bledu (Error line)=" B
80       RESUME 90
90 NEXT
100 PRINT: PRINT "Proces zakonczony pomyslnie (Process completed successfully)"
110 END
```

Pierwsza wersja:

```
10 FOR I=2 TO -2 STEP -1
```

<---->

Wynik:

```
I= 2          log(i)= .6931472   Nie ma bledu (No error)
I= 1          log(i)= 0          Nie ma bledu (No error)
I= 0          log(i)=
Numer bledu (Error number)= 5    Linia bledu (Error line)= 20
I=-1          log(i)=
Numer bledu (Error number)= 5    Linia bledu (Error line)= 20
I=-0          log(i)=
Numer bledu (Error number)= 5    Linia bledu (Error line)= 20
```

Druga wersja:

```
10 FOR I=-2 TO 2 STEP 1
```

Wynik:

```
I=-2          log(i)=
Illegal function call in 20
OK
```

```
Process zakonczony pomyslnie (Process completed successfully)
OK
```

Bardzo dobrze - wszystko jak oczekiwano

Bardzo źle - nieoczekiwany rezultat !

Przy wpisaniu wyrażenia algebraicznego funkcji nie można liczyć na przypadek poprawnego procesu przetwarzania błędów. Dlatego też koniecznym jest wpisanie tuż nad linią funkcji f(x,z) linii omijających niewykonalne operacje matematyczne (pokolorowane linie poniżej), poprzez:

IF (*warunek do ominięcia*) THEN BLAD=1: GOTO 14050

Przykład:

```
12090 IF (X+Z <= 0) THEN BLAD=1: GOTO 14050 ' ta linia omija błąd
13000 Y=LOG(X+Z)                            ' <-- tu już błędu nie będzie
```

Inny przykład:

```
12080 IF (X-2*Z <= 0) THEN BLAD=1: GOTO 14050  ' ta linia omija błąd
12090 IF (X + LOG(X-2*Z) < 0) THEN BLAD=1: GOTO 14050 'linia omija błąd
13000 Y= 2 + SQR(X + LOG(X-2*Z))    ' <-- tu już błędu nie będzie
```

Koniecznym jest tu podanie BLAD=1 informujący dalszą część procesu, że wystąpiła niepoprawna operacja matematyczna.
Jestem przekonany, że jest jakaś metoda wychwycenia błędu nr 5 czyli 'illegal function call' ale musi ona jakoś sztucznie oszukiwać program. Z błędem nr 11 czyli 'Division by zero' (dzielenie przez zero) nie ma tego kłopotu.

Należy pamiętać, że każda wartość spoza zakresu liczb całkowitych (od -32768 do 32767) powoduje błąd "OVERFLOW" i wstrzymanie działania programu.

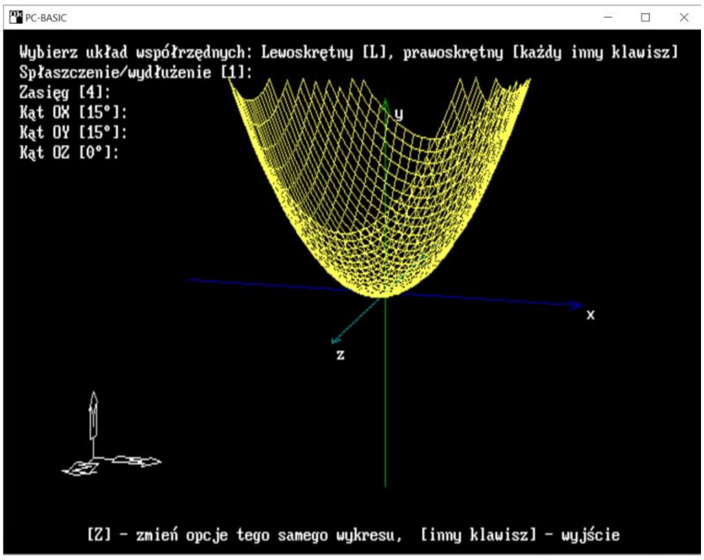
- - - - - - -

W poniższych przykładach piszę X*X*X chociaż zawsze można napisać X^3

13000 Y = X * X / 2 + Z * Z / 6

$y = x^2/a^2 + z^2/b^2$ Elliptic paraboloid

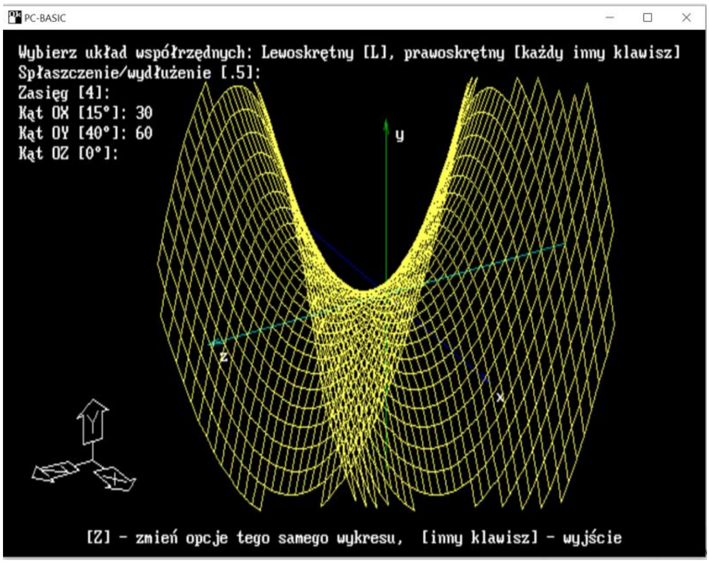
str. 716 i 719, (2) Y=4x2+z2, str. 687 i 688, (4) str. 756



13000 Y = Z * Z - X * X

$y = z^2/b^2 - x^2/a^2$ Hyperbolic paraboloid

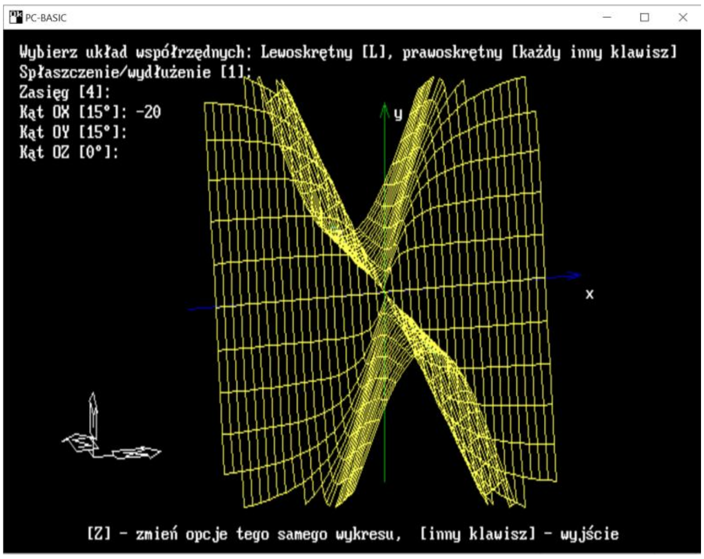
(1) str. 716 i 775, (2) str. 687 i 688 + 812, (4) str. 756 i 801



13000 Y = (5*X*X*Z) / (X*X+Z*Z)

$y = 5x^2z/(x^2+z^2)$

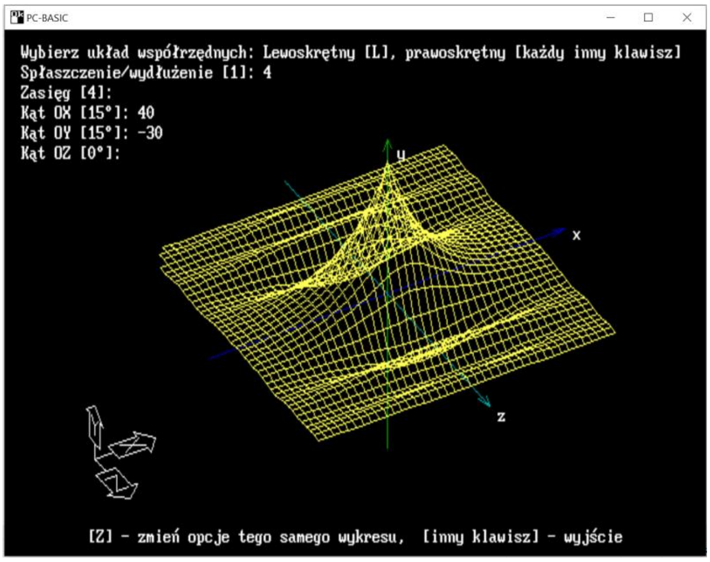
(1) str. 785



E = 2.71828 <-- linia 194

13000 Y = COS(Z*Z)*E^(-SQR(X*X+Z*Z))

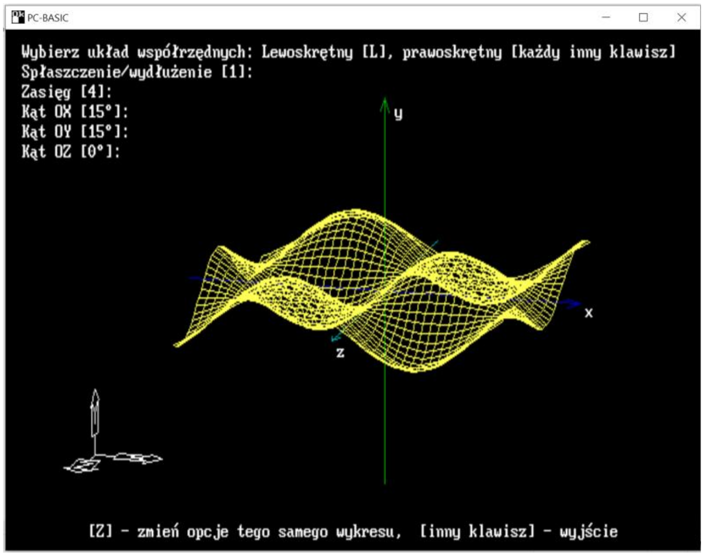
(1) str.783



13000 Y = SIN(X) * COS(Z)

y = sin x * cos z

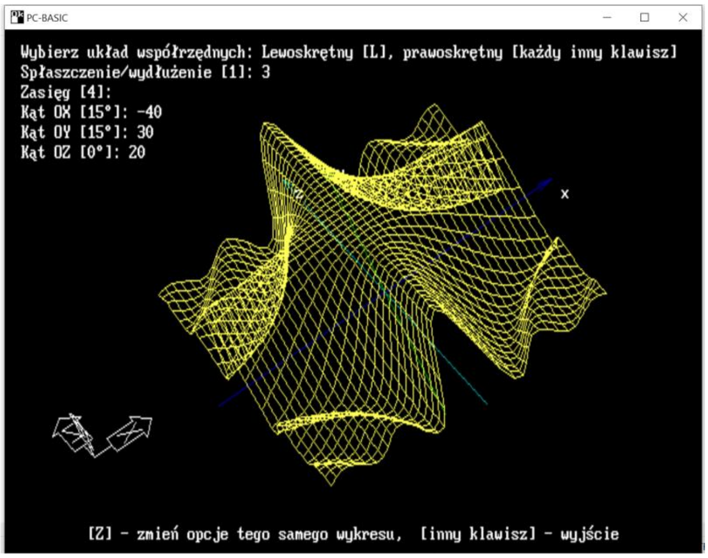
tego nie było w książkach



13000 Y = SIN(X*Z) / (X*Z)

y = sin (xz) / (xz)

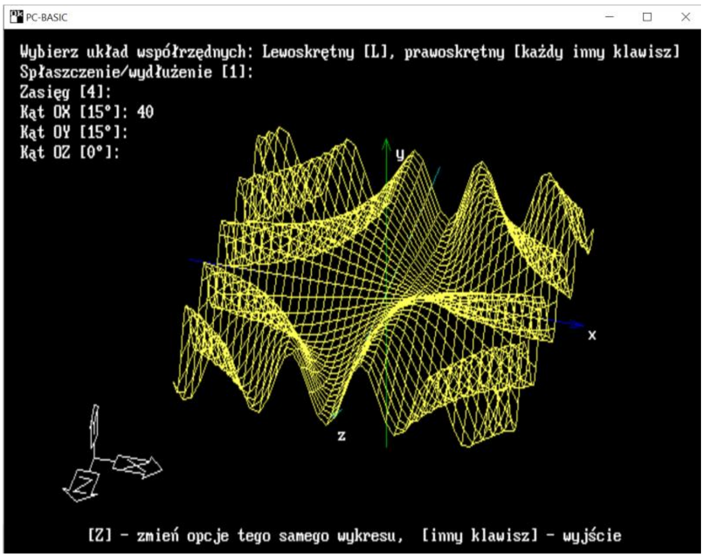
(1) str.785



13000 Y = SIN(X * Z)

y = sin xz

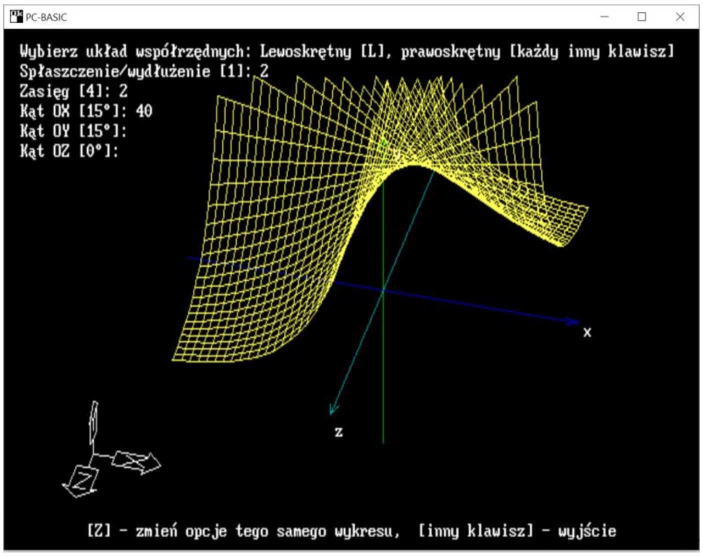
(1) str 783



E = 2.71828 <-- linia 194

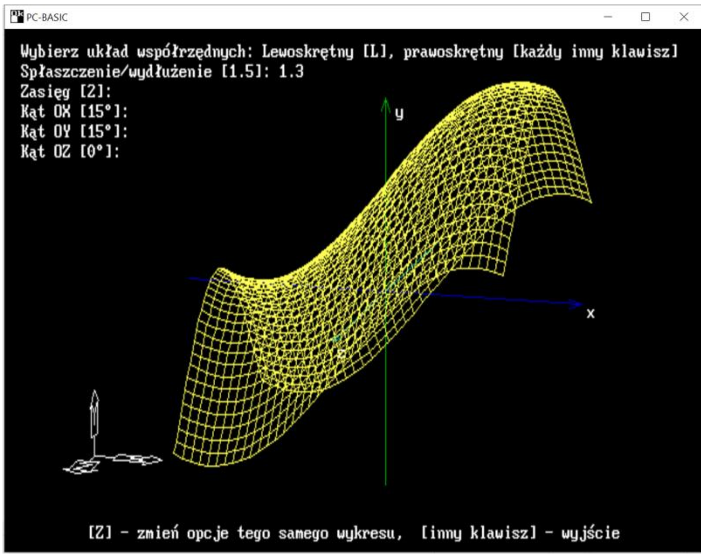
13000 Y = E^(X*Z)

$y = e^{xz}$ (1) str.786



13000 Y = SIN(X) + COS(Z)

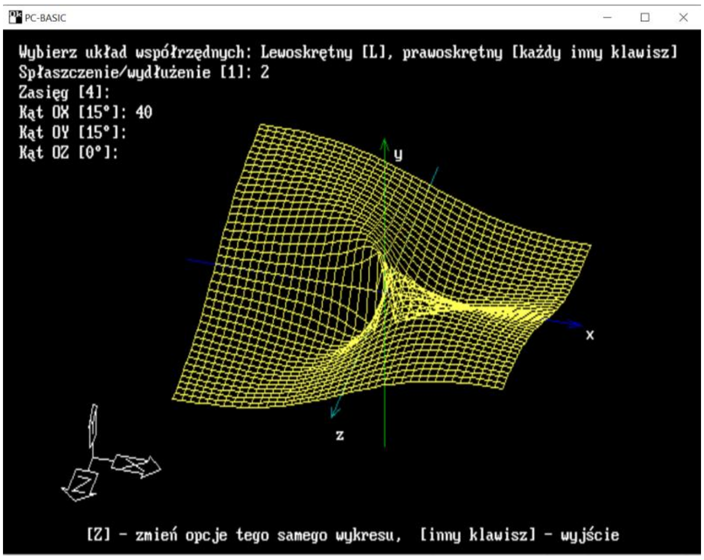
(1) str.786



13000 Y = (X*Z) / (X*X+Z*Z)

y = $(xz)/(x^2+z^2)$

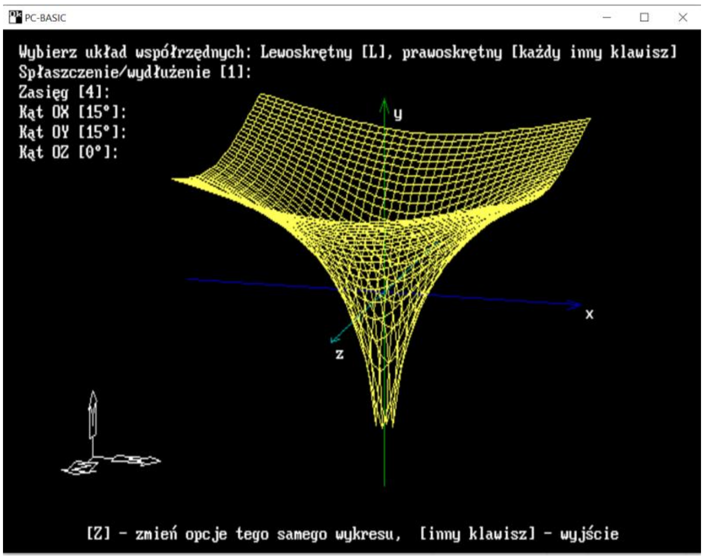
(1) str.786



13000 Y = LOG(X*X+Z*Z)

y = $\log(x^2+z^2)$

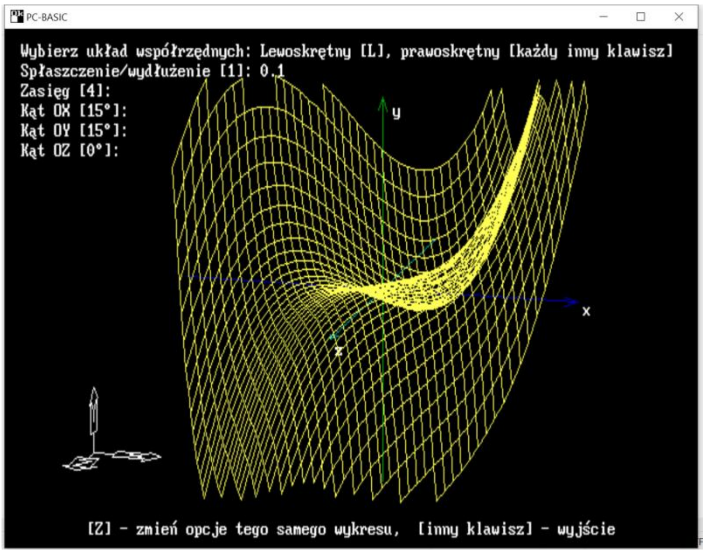
(1) str.787



13000 Y = X*X*X -3*X*Z + Z*Z*Z

y = $x^3-3xz+z^3$

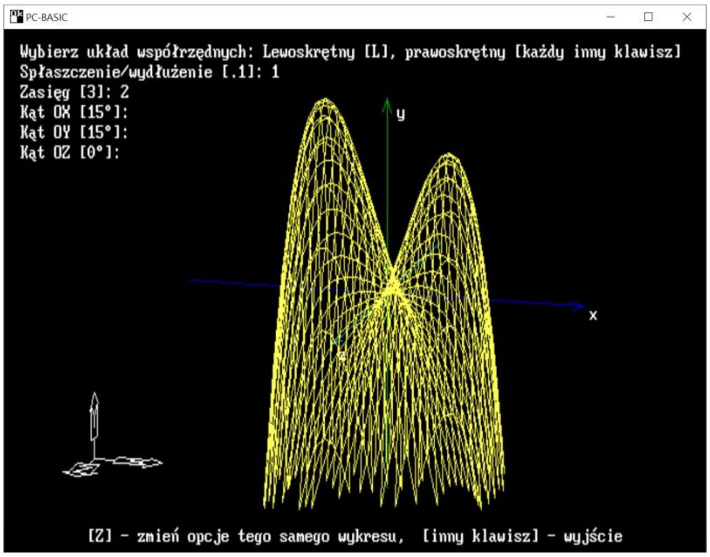
(1) str.836



13000 Y = 4*X*Z-X*X*X*X-Z*Z*Z*Z

y = $4xz-x^4-z^4$

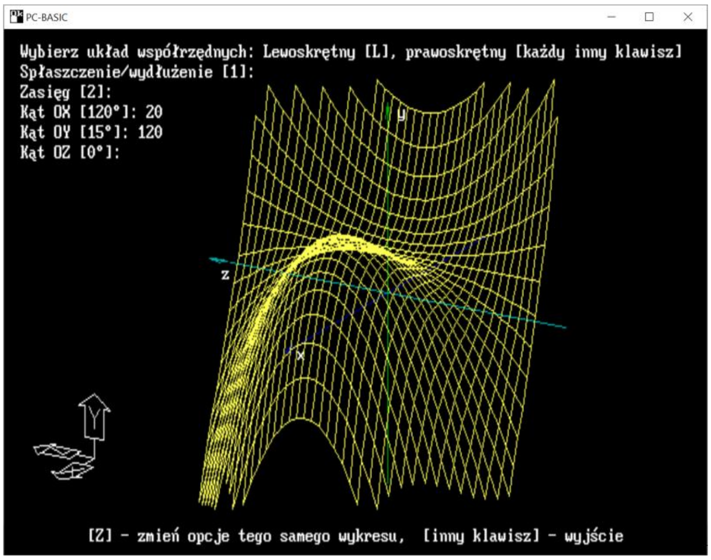
(1) str.836



13000 Y = (3*X*X+1)/2 - X*(X*X+Z*Z)

y = $(3x^2+1)/2 - x(x^2+z^2)$

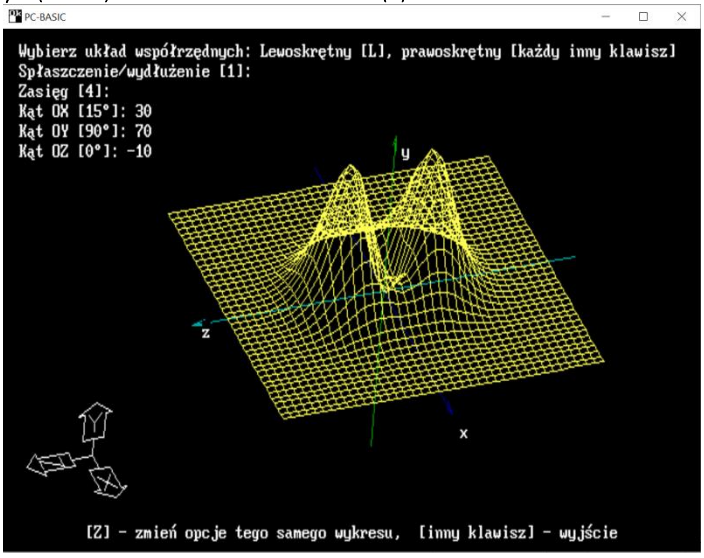
(1) str.836



E = 2.71828 <-- linia 194

13000 Y = (X*X+4*Z*Z)*E^(1-X*X-Z*Z)

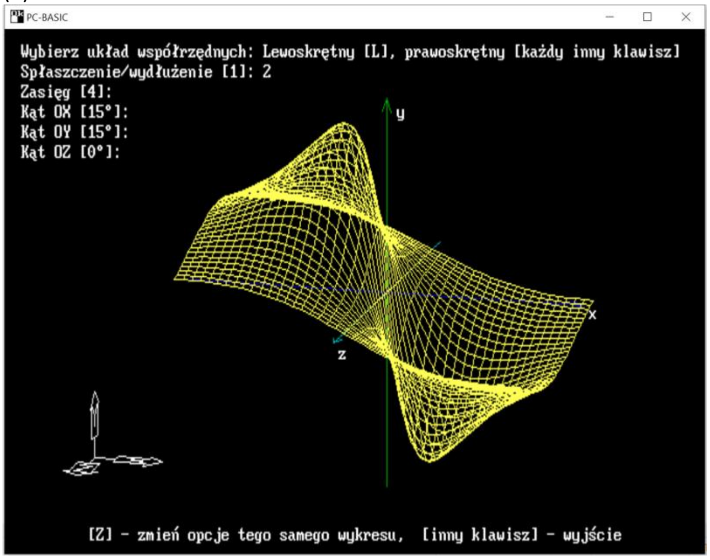
$y = (x^2+4z^2)e^{1-xx-zz}$ (1) str.836



13000 Y = (-4*X)/(X*X+Z*Z+1)

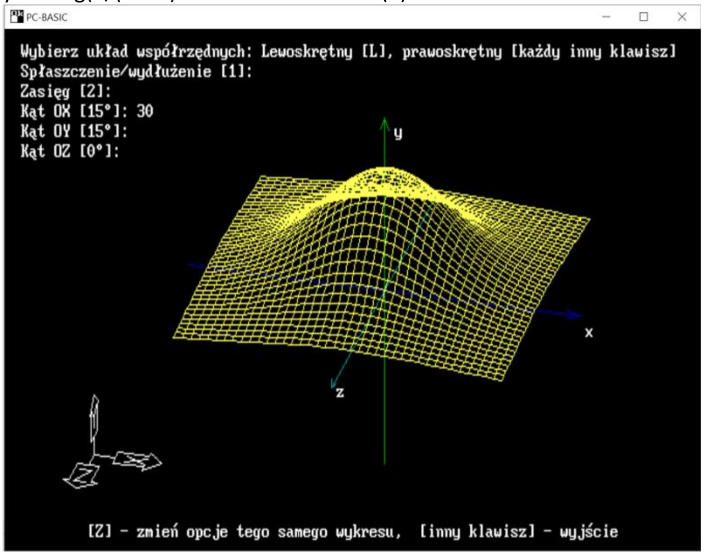
$y = -4x/(x^2+z^2+1)$

(1) str.837



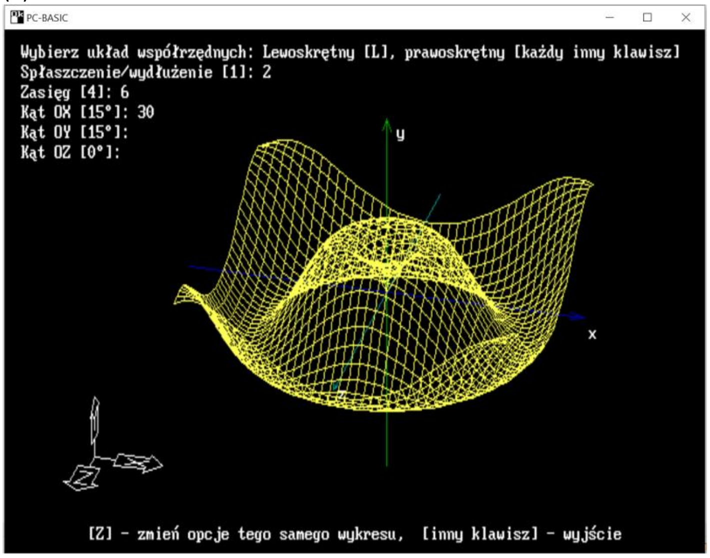
13000 Y = ATN(1/(X*X+Z*Z))

$y = \mathrm{arctg}(1/(x^2+z^2)$ (1) str.837



13000 Y = SIN(SQR(X*X+Z*Z))

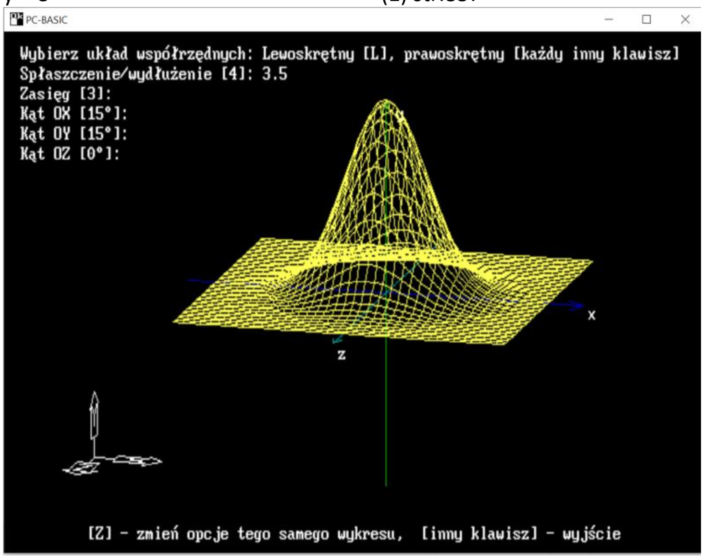
(2) str.758 i 759



E = 2.71828 <-- linia 194

13000 Y = E^(-(X*X+Z*Z))

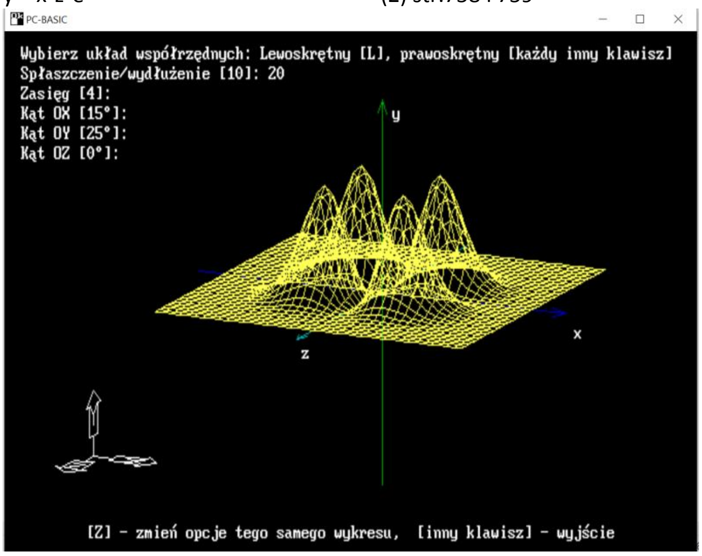
$y = e^{-(xx+zz)}$ (1) str.837



E = 2.71828 <-- linia 194

13000 Y = X*X*Z*Z*E^(-X*X-Z*Z)

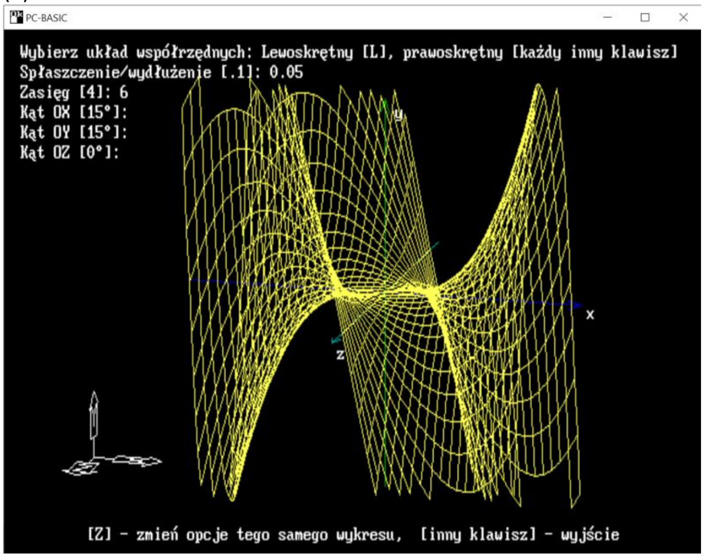
$y = x^2z^2e^{-xx-zz}$ (2) str.758 i 759



13000 Y = X*X*X-3*X*Z*Z

$y = x^3 - 3xz^2$

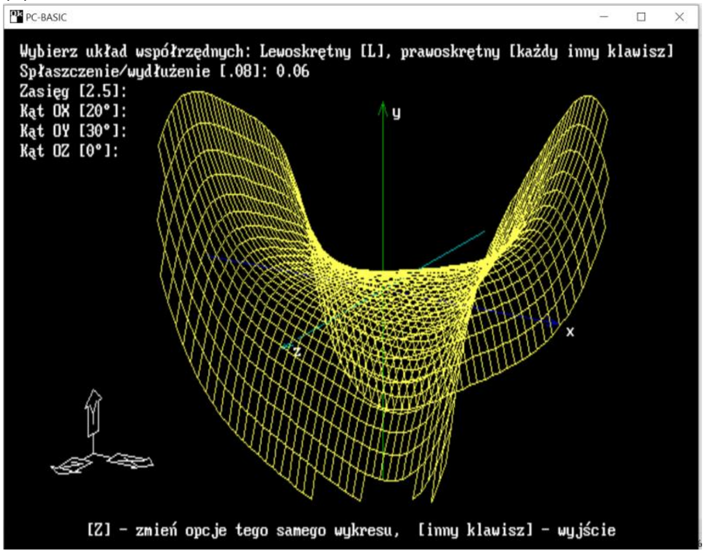
(2) str.758 i 759



13000 Y = X*X*X*X - Z*Z*Z*Z - 4*X*Z + 1

$y = x^4 - z^4 - 4xz + 1$

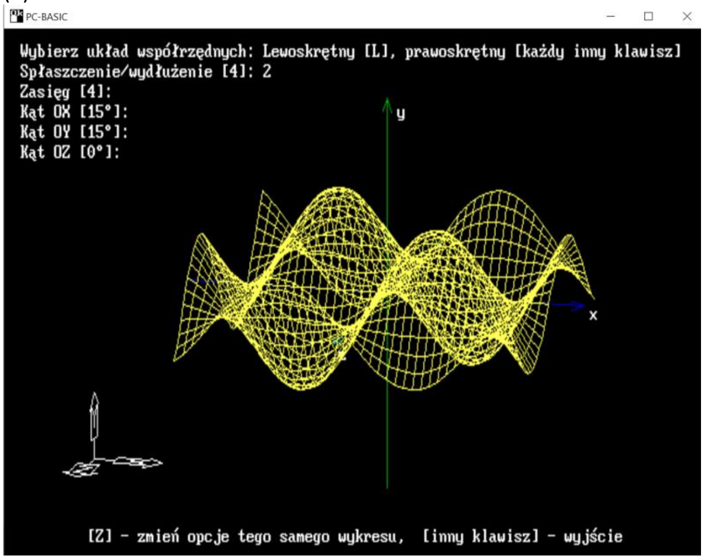
(2) str.813



13000 Y = SIN(X) * SIN(Z)

$y = \sin x * \sin z$

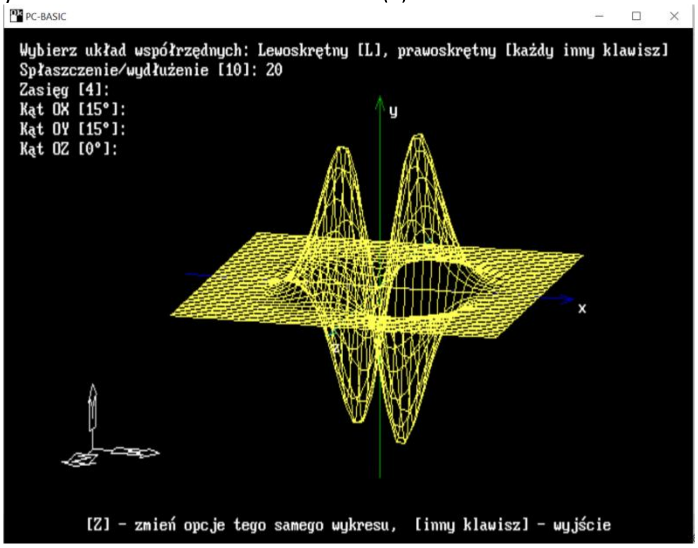
(2) str.758 i 759



E = 2.71828 <-- linia 194

13000 Y = -X*Z*E^(-X*X-Z*Z)

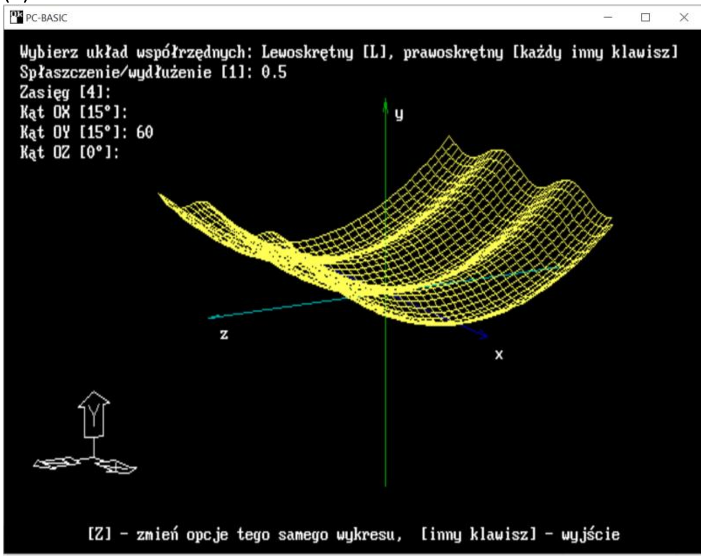
$y = -xze^{-x*x-z*z}$ (2) str.754



13000 Y = (SIN(X))^2 + 0.25*Z*Z

$y = \sin^2 x + 1/4\, z^2$

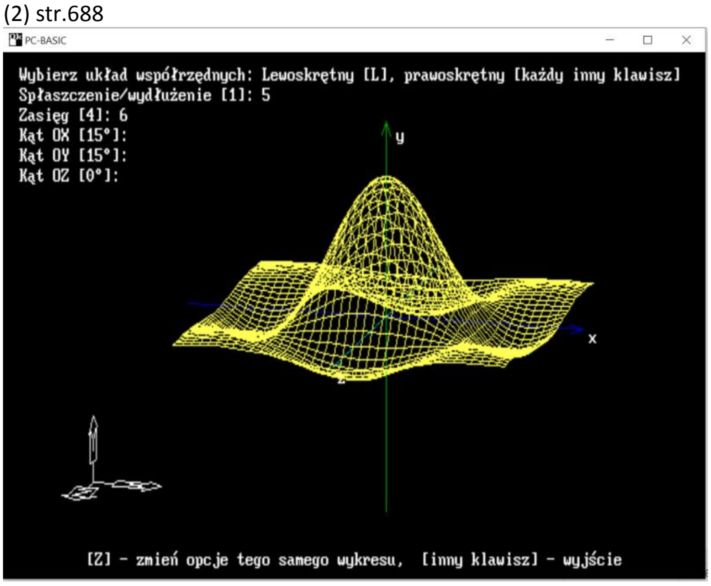
(2) str.758 i 759



13000 Y = (SIN(X)*SIN(Z)) / (X*Z)

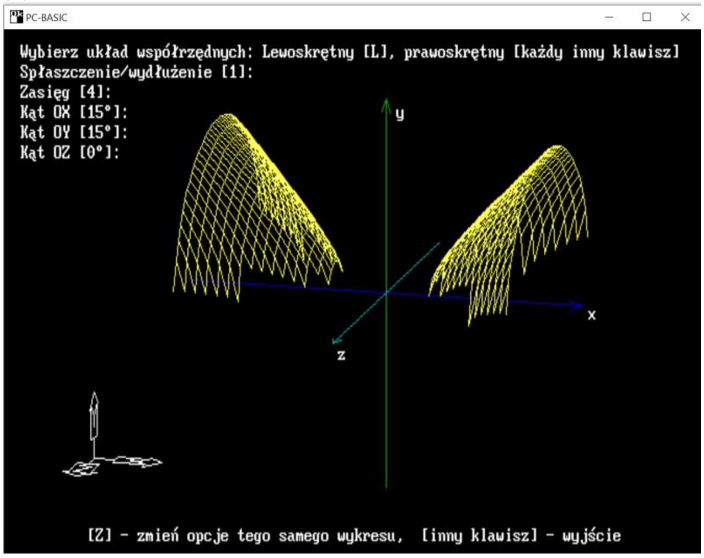
$y = \sin x \sin z / (xz)$

(2) str.688



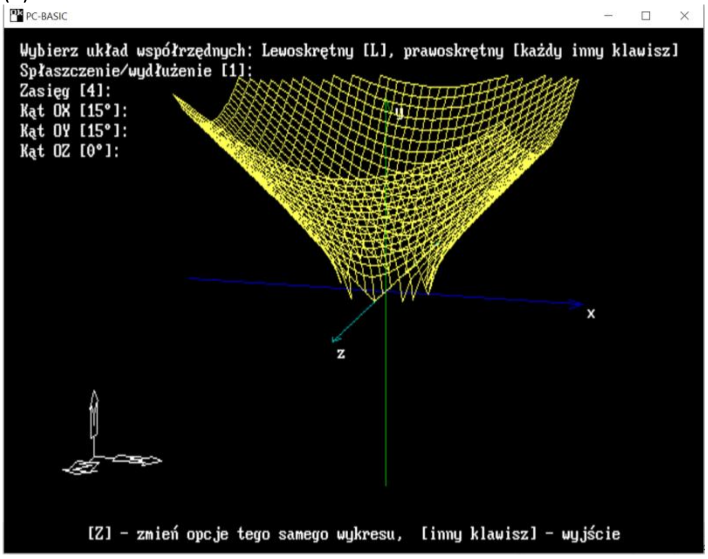
12990 IF X*X-Z*Z-1 < 0 THEN BLAD=1: GOTO 14050
13000 Y = SQR(X*X-Z*Z-1)

(3) str.636



12990 IF X*X+Z*Z-1 < 0 THEN BLAD=1: GOTO 14050
13000 Y = SQR(X*X+Z*Z-1)

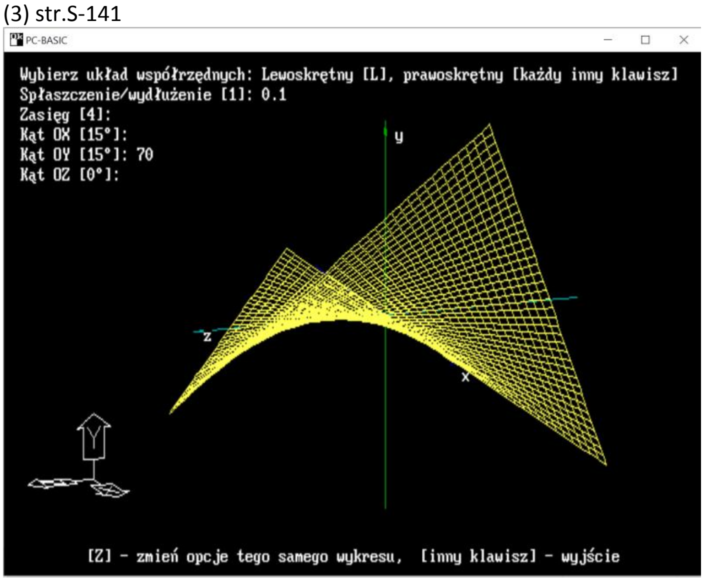
(3) str.635



13000 Y = 2*X*Z <-- saddle (siodło)

y = 2xz

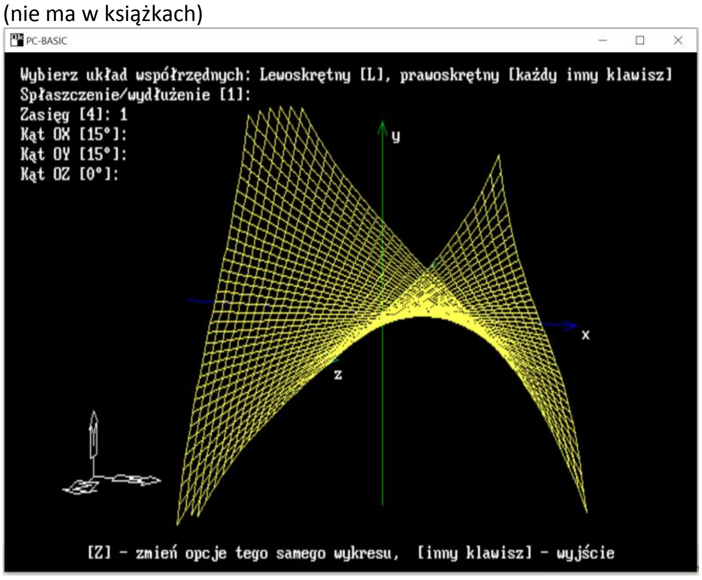
(3) str.S-141



13000 Y = TAN(X*Z)

y = Tan(xz)

(nie ma w książkach)



- - - - - - - - - - - - - -

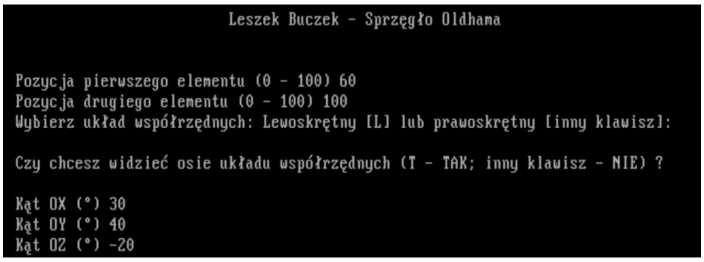
(1) Roland E. Larson, Robert P. Hostetler: Calculus with Analytic Geometry; The Pennsylvania State University, The Behrend College; Third edition; 1986, D.C. Heath and Company, ISBN: 0-669-09568-0

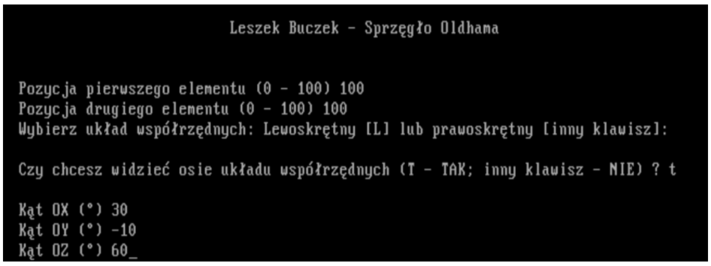
(2) James Stewart: Calculus, Concepts and Contexts; McMaster University; 1998, Brooks Cole Publishing Company, ISBN: 0-534-34330-9

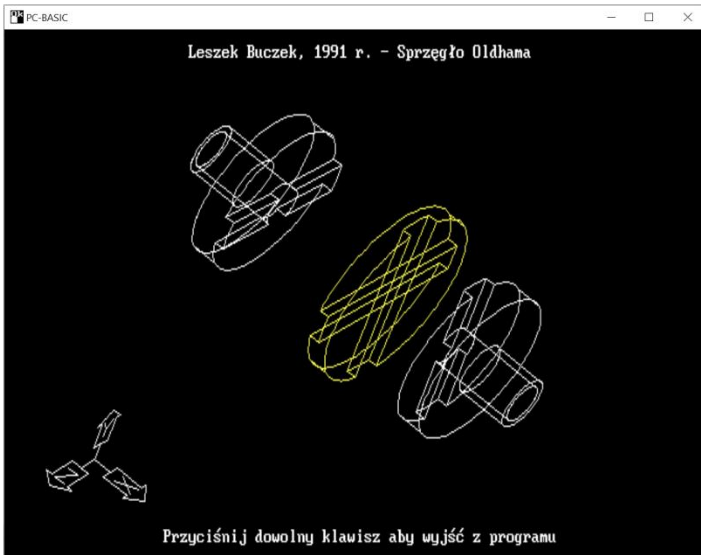
(3) Sherman K. Stein: Calculus and Analytic Geometry; University of California, Davis; Fourth edition; 1987, McGraw-Hill Book Company, ISBN: 0-07-061159-9

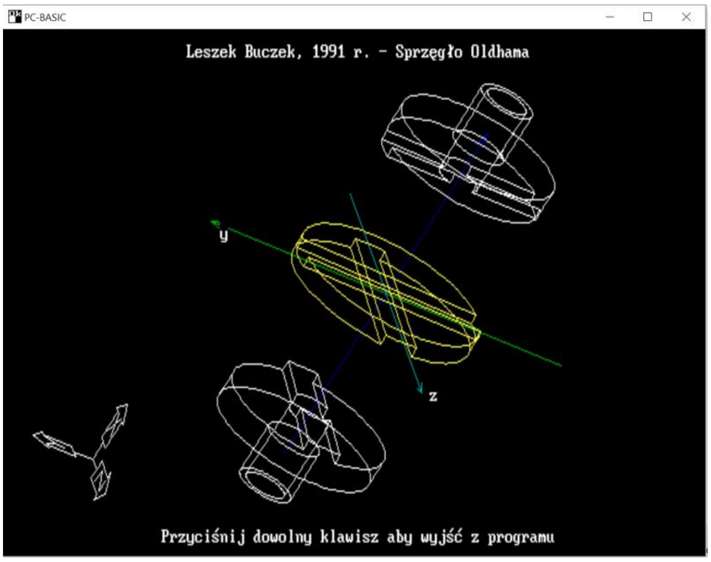
(4) Harley Flanders: Calculus; Florida Atlantic University; 1985, W. H. Freeman and Company, ISBN: 0-7167-1643-7

Poniżej jest przykład zastosowania programu lecz nie do wykresu funkcji ale do ogólnie pojętej grafiki przestrzennej.









Jak to robią inne aplikacje?

Na przykładzie powyżej podanej pierwszej funkcji

```
' --- Równanie funkcji Y=f(X,Z) ---
V=-(X*X+Z*Z)
VV=-((X-2)*X+(Z-.5)*Z)
Y=5*EXP(V)-2*EXP(VV)
' ---------------------------------
```

Jak to robi *GeoGebra*

GeoGebra Wykres 3D ← https://www.geogebra.org/3d?lang=pl

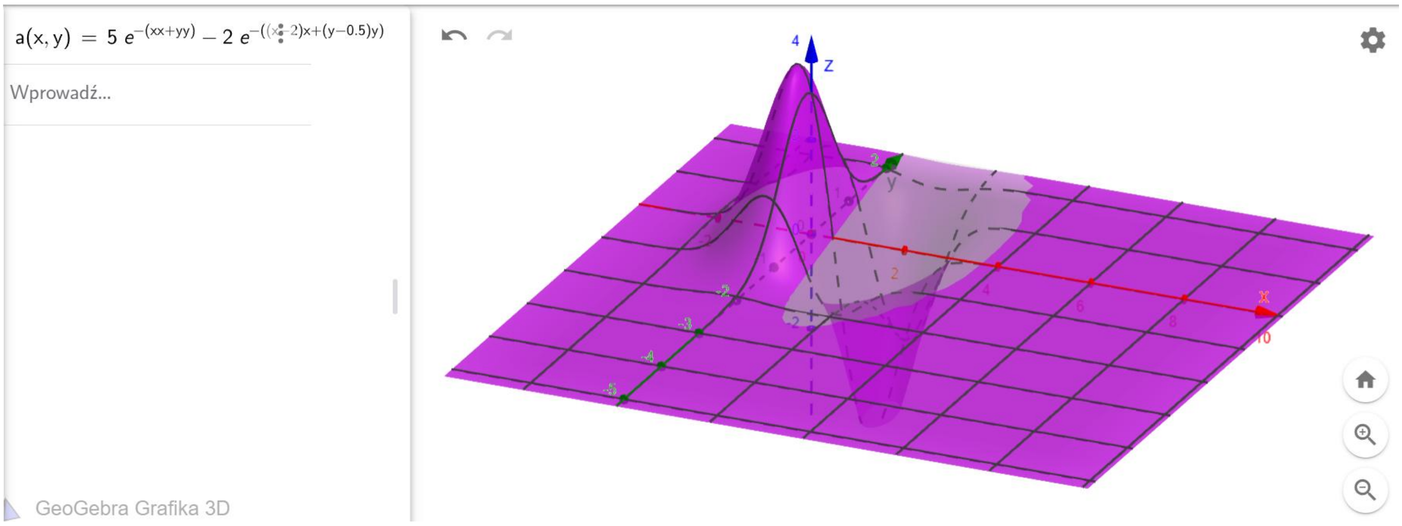
Wspaniałe narzędzie do przedstawiania grafiki przestrzennej.

- ✓ aby dostać grafikę nic nie trzeba ‘importować’, dołączać (*include*), robić ‘download’ czegokolwiek – od razu można wpisać zależności, nie tylko funkcyjne,
- ✓ aplikacja posiada sporo narzędzi co bardzo wzbogaca możliwości prezentacji obrazu graficznego.

Pomimo fascynacji jaką doznaję bawiąc się aplikacją GeoGebry, pozostaje ona poza tematem tej prezentacji, która ma podawać pełny kod w wybranym języku programowania dla tworzenia grafiki.

Na jakim etapie rozwoju była GeoGebra w 1991 roku, w którym to roku po raz pierwszy udostępniałem innym swój program? Nie mogłem tego zrobić wtedy dla wszystkich, więc robię to teraz.

$z = f(x,y) = 5*e^{-(x*x+y*y)}-2*e^{-((x-2)*x+(y-.5)*y)}$



Jak to robi *Python*

Aby otrzymać obraz figury przestrzennej w dowolnym jej ustawieniu trzeba importować takie cudotwórcze mechanizmy jak:

- OpenGL.GL/GLU/GLUT
- vpython

lub coś podobnego.

Jeżeli z nich zrezygnuję to zostaje mi *matplotlib* z *numpy*. Jakkolwiek wynik jest dobry, ale tylko w jednym ustawieniu - typowo jednostronna prezentacja.

Poniżej kod do skopiowania do Python'a i wynik:

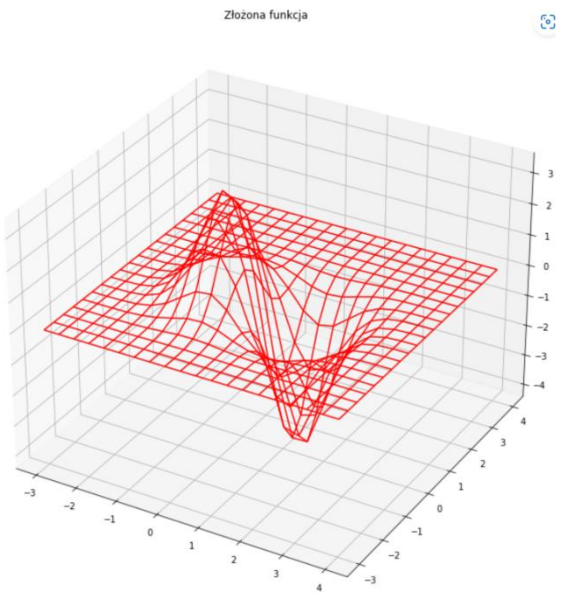
```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return 5*np.exp(-(x*x+y*y)) - 2*np.exp(-((x-2)*x+(y-0.5)*y))
# V=-(X*X+Y*Y)
# VV=-((X-2)*X+(Y-.5)*Y)
# Z=5*EXP(V)-2*EXP(VV)

x = np.linspace(-3, 4, 20)
y = np.linspace(-3, 4, 20)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)

fig = plt.figure(figsize=(12, 18))
ax = plt.axes(projection ='3d')
ax.plot_wireframe(X, Y, Z, color ='red')
ax.set_title('Złożona funkcja');
```



Takie rzeczy wykonuje także Excel.

Jak najprościej zrobić taką jednostronną prezentację figury przestrzennej

Załóżmy, że chcemy otrzymać wykres funkcji `y=f(x,z)` z osiami **X** i **Y** na ekranie jakby nie było osi **Z**, ale ona jest: prostopadła do ekranu a więc do płaszczyzny wyznaczonej przez osie **X** i **Y**.
Wszelkie liczby dotyczące przesunięcia określonego punktu o ilość pikseli na ekranie, są tu tylko przykładowe.

Zacznijmy od rysowania łamanej linii dla z=-10.
I obliczamy **y** jako `f(x,z)`, gdzie **z**=-10 a **x** zmienia się od -10 do 10 co 1. Otrzymamy na ekranie 21 punktów. W trakcie tych 21 obliczeń:

- zapamiętujemy pikselową pozycję każdego z punktów w tablicy `DIM POZX(21)` i `DIM POZY(21)` (nazwy sugerują: pozycja x i pozycja y),
- punkty te kolejno łączymy jeden za drugim instrukcją `LINE -( POZX(i), POZY(i) )` otrzymując krzywą łamaną z połączonych 21 punktów. Tu **i** od 1 do 20 (**i**=0 to pierwszy punkt każdego **z**)

Teraz prowadzimy identyczne obliczenia ale dla **z**=-9, ale:

- **każdą obliczoną pozycję piksela przesuwamy o 5 pikseli w dół i 3 piksele w prawo**.
- zaraz po tym obliczeniu każdego piksela:
  - łączymy go z odpowiednim punktem `(POZX(i),POZY(i))` obliczonym wcześniej dla **z**=-10,
  - zmieniamy wartości `POZX(i)` i `POZY(i)` z tych obliczonych dla **z**=-10 na te wartości dla **z**=-9 (aby przygotować dane dla następnej linii, czyli dla **z**=-8),
  - łączymy każdy punkt obliczony dla **z**=-9, tak, jak to zrobiliśmy wcześniej dla **z**=-10.

Otrzymujemy ciąg małych 21 trapezów (chyba, że linie się ze sobą przecinają).

(W tym miejscu możesz podjąć się wypełnienia kolorem każdego z trapezów, co drugi innym kolorem – najlepiej innego odcienia, np. jasnoszary i ciemnoszary. Takie powielanie kolorami następnej linii trapezów – dla '**z**' większego o 1 – da efekt pokrycia niewidocznych linii i płaszczyzn nowymi liniami i płaszczyznami (stąd **z** od -10 do 10 a nie odwrotnie). Może to zadanie nie być takie proste, bo nowe punkty mogą być poniżej albo powyżej starych punktów na ekranie. W dodatku nowe linie mogą przecinać się ze starymi.)

Jak osiągniemy etap, w którym zakończy się rysowanie łamanej linii dla **z**=10, wykres uznajemy za ukończony.

Jeżeli chcesz mieć ten wykres przekręcony, tak jak na ostatnim wyniku Pythona, to dla kolejnego wyniku dla ‘**y**’ podnieś punkt o następne dwa piksele. Powtarzaj to osobno dla każdego ‘**z**’, podnosząc obliczone ‘**y**’ o następne 2 piksele dla kolejnego ‘**x**’.

W wyniku otrzymujemy dobre ale tylko poglądowe przedstawienie funkcji przestrzennej i nie ma to wiele wspólnego z obrotami figury o dowolne kąty. Nie wymaga znajomości matematyki. Jest to zadanie dla 14-letniego pasjonata IT.