Аналитическая геометрия

Пример 1

In [1]:
```
import numpy as np
from numpy.linalg import norm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from sympy.abc import x, y
from sympy import *
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
a = np.array([1,2,3,4,5])
a
```

Out[2]:
```
array([1, 2, 3, 4, 5])
```

In [3]:
```
a[1]
```

Out[3]:
```
2
```

In [4]:
```
a[1:4]
```

Out[4]:
```
array([2, 3, 4])
```

In [5]:
```
a[-1]
```

Out[5]:
```
5
```

In [6]:
```
a = np.array([1,2,3,4])
b = np.array([11,12,13,14])
c = a+b
d = a-b
e = 4*a
print(f'a+b: {c}, a-b: {d}, 4a: {e}')
```

```
a+b: [12 14 16 18], a-b: [-10 -10 -10 -10], 4a: [ 4  8 12 16]
```

Пример 2

In [7]:
```
a = np.array([3,4])
norm(a)
```

Out[7]:
```
5.0
```

Пример 3

In [8]:
```
f = np.array([3,1,4])
g = np.array([0,-2,1])
np.dot(f, g)
```

Out[8]: 2

Пример 4

In [9]:
```
a = np.array([1,2])
b = np.array([3,4])
np.dot(a,b)/norm(b)
```

Out[9]: 2.2

Пример 5

In [10]:
```
f = np.array([3,1,4])
g = np.array([0,-2,1])
np.cross(f,g)
```

Out[10]: array([9, -3, -6])

In [11]:
```
a = np.array([1,2,3])
b = np.array([4,0,-1])
c = np.array([0,5,-2])
np.dot(a,np.cross(b,c))
```

Out[11]: 81

In [12]:
```
A = np.array([a,b,c])
np.linalg.det(A)
```

Out[12]: 80.99999999999996

In [13]:
```
P1P = np.array([8-2,5-2])
P2P = np.array([8-10,5-6])
np.cross(P1P,P2P)
```

Out[13]: array(0)

Пример 6

In [14]:
```
a = Matrix([[1,2,3], [0,-1, 1]])
a
```

Out[14]:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -1 & 1 \end{bmatrix}$$

In [15]:
```
A = Matrix([[1,2,3]])
A
```

Out[15]:

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$$

In [16]:
```
A = Matrix([[1],[2],[3]])
A
```

Out[16]:

$$\begin{bmatrix}1\\2\\3\end{bmatrix}$$

In [17]:

```
A = Matrix([1,2,3])
A
```

Out[17]:

$$\begin{bmatrix}1\\2\\3\end{bmatrix}$$

Точки

In [18]:

```
Point(1, 2, 3)
```

Out[18]: $\mathrm{Point3D}(1, 2, 3)$

In [19]:

```
Point([1, 2])
```

Out[19]: $\mathrm{Point2D}(1, 2)$

In [20]:

```
p = Point(dim=4)
```

In [21]:

```
p1, p2 = Point(1, 1), Point(4, 5)
p1.distance(p2)
```

Out[21]: 5

In [22]:

```
p3 = Point(x, y)
p3.distance(Point(0, 0))
```

Out[22]: $\sqrt{x^2 + y^2}$

In [23]:

```
p1, p2 = Point(1, 1), Point(13, 5)
p1.midpoint(p2)
```

Out[23]: $\mathrm{Point2D}(7, 3)$

In [24]:

```
p1 = Point3D(1, 2, 2)
p2 = Point3D(2, 7, 2)
p3 = Point3D(0, 0, 2)
p4 = Point3D(1, 1, 2)
Point3D.are_coplanar(p1, p2, p3, p4)
```

Out[24]:

```
True
```

In [25]:

```
p1, p2 = Point(0, 0), Point(1, 1)
p3, p4, p5 = Point(2, 2), Point(x, x), Point(1, 2)
Point.is_collinear(p1, p2, p3, p4)
```

Out[25]:

```
False
```

In [26]:

```
pl, p2, p3, P4 = Point(1, 0), (0, 1), (-1, 0), (0, -1)
pl.is_concyclic()
```

Out[26]: True

Прямые на плоскости и в пространстве

In [27]:
```
Line((0, 0), (1, 1))
```

Out[27]:

In [28]:
```
p1, p2 = Point(1, 1), Point(3, 0)
Line(p1, p2)
```

Out[28]:

In [29]:
```
p1, p2 = Point(1,0), Point(5,3)
l1 = Line(p1, p2)
l1.equation()
```

Out[29]: $-3x + 4y + 3$

In [30]:
```
l1.coefficients
```

Out[30]: (-3, 4, 3)

In [31]:
```
p1, p2 = Point(1,0,0), Point(5,3,2)
l2 = Line(p1, p2)
l2.equation()
```

Out[31]: $(-3x + 4y + 3, \ -x + 2z + 1)$

In [32]:
```
p1, p2 = Point(1, 0), Point(5, 3)
l1 = Line(p1, p2)
l1.arbitrary_point()
```

Out[32]: $\mathrm{Point2D}(4t + 1, 3t)$

In [33]:
```
p1, p2 = Point3D(1, 0, 0), Point3D(5, 3, 1)
l1 = Line3D(p1, p2)
l1.arbitrary_point()
```

Out[33]: $\mathrm{Point3D}(4t + 1, 3t, t)$

In [34]:
```
A = Point(-2,3)
k = 2
l = Line(A, slope = k)
l.equation()
```

Out[34]: $-2x + y - 7$

In [35]:
```
A = Point(-2,3,0)
l = Line(A, direction_ratio=[1,2,0])
l.equation()
```

Out[35]: $(-2x + y - 7,\ z)$

In [36]:
```
a, b = (0,0), (3,3)
Line(a, b).direction
```

Out[36]: $\mathrm{Point2D}(3, 3)$

In [37]:
```
Line(a, b).direction.unit
```

Out[37]: $\mathrm{Point2D}\left(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$

In [38]:
```
p1, p2 = Point(0, 0, 0), Point(1, 1, 1)
s = Line(p1, p2)
s.distance(Point(-1, 1, 1))
```

Out[38]: $\frac{2\sqrt{6}}{3}$

In [39]:
```
e = Line((0, 0), (1, 0))
sw = Line((1, 1), (0, 0))
sw.angle_between(e)
```

Out[39]: $\frac{3\pi}{4}$

In [40]:
```
sw.smallest_angle_between(e)
```

Out[40]: $\frac{\pi}{4}$

In [41]:
```
p1, p2 = Point(0, 0), Point(3, 5)
p3, p4 = Point(-2, -2), Point(0, 2)
l1, l2, l3 = Line(p1, p2), Line(p1, p3), Line(p1, p4)
Line.are_concurrent(l1, l2, l3)
```

Out[41]:
```
True
```

In [42]:
```
l1 = Line3D(Point3D(4,19,12), Point3D(5,25,17))
l2 = Line3D(Point3D(-3, -15, -19), direction_ratio=[2,8,8])
l1.intersection(l2)
```

Out[42]:
```
[Point3D(1, 1, -3)]
```

In [43]:
```
p1, p2 = Point3D(0, 0, 0), Point3D(3, 4, 5)
p3, p4 = Point3D(2, 1, 1), Point3D(8, 9, 11)
l1, l2 = Line3D(p1, p2), Line3D(p3, p4)
Line3D.is_parallel(l1, l2)
```

Out[43]:
```
True
```

In [44]:
```
p1, p2, p3 = Point(0, 1), Point(3, 4), Point(2, 3)
l1 = Line(p1, p2)
```

```python
l2 = Line(p1, p3)
l1.is_similar(l2)
```

Out[44]: True

In [45]:
```python
p1, p2, p3 = Point(0, 0), Point(2,3), Point(-2, 2)
l1 = Line(p1, p2)
l2 = l1.parallel_line(p3)
p3 in l2
```

Out[45]: True

In [46]:
```python
p1, p2, p3 = Point(0, 0), Point(2, 3), Point(-2, 2)
l1 = Line(p1, p2)
l2 = l1.perpendicular_line(p3)
p3 in l2
```

Out[46]: True

In [47]:
```python
p1, p2, p3 = Point(0, 0), Point(1, 1), Point(Rational(1, 2), 0)
l1 = Line(p1, p2)
l1.projection(p3)
```

Out[47]: $\mathrm{Point2D}\left(\frac{54}{25}, \frac{72}{25}\right)$

Плоскости

In [48]:
```python
a = Plane(Point3D(1, 1, 2), Point3D(2, 4, 7), Point3D(3, 5, 1))
a.equation()
```

Out[48]: $-23x + 11y - 2z + 16$

In [49]:
```python
a = Plane(Point3D(1, 4, 2), normal_vector=(6, 6, 6))
a.equation()
```

Out[49]: $6x + 6y + 6z - 42$

Пример 7

In [50]:
```python
Plane((1, 1, 1), (1, 4, 7))
```

Out[50]: $\mathrm{Plane}(\mathrm{Point3D}(1, 1, 1), (1,\ 4,\ 7))$

In [51]:
```python
a = Plane((1, 1, 1), (0, 0, 1))
p = a.p1
p
```

Out[51]: $\mathrm{Point3D}(1, 1, 1)$

In [52]:
```python
a = Plane(Point3D(1, 1, 2), Point3D(2, 4, 7), Point3D(3, 5, 1))
a.equation()
```

Out[52]: $-23x + 11y - 2z + 16$

In [53]:
```
a = Plane(Point3D(1, 1, 1), Point3D(2, 3, 4), Point3D(2, 2, 2))
a.normal_vector
```
Out[53]: $\left(-1,\ 2,\ -1\right)$

In [54]:
```
a = Plane(Point3D(1, 4, 6), normal_vector=(2, 4, 6))
a.parallel_plane(Point3D(2, 3, 5))
```
Out[54]: $\mathrm{Plane}\left(\mathrm{Point3D}\left(2, 3, 5\right), \left(2,\ 4,\ 6\right)\right)$

In [55]:
```
a, b = Point3D(0, 0, 0), Point3D(0, 1, 0)
Z = (0, 0, 1)
p = Plane(a, normal_vector=Z)
p.perpendicular_plane(a, b)
```
Out[55]: $\mathrm{Plane}\left(\mathrm{Point3D}\left(0, 0, 0\right), \left(1,\ 0,\ 0\right)\right)$

In [56]:
```
a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
a.perpendicular_line(Point3D(9, 8, 7))
```
Out[56]: $\mathrm{Line3D}\left(\mathrm{Point3D}\left(9, 8, 7\right), \mathrm{Point3D}\left(11, 12, 13\right)\right)$

In [57]:
```
a = Plane(Point3D(1, 1, 1), normal_vector=(1, 1, 1))
b = Point3D(1, 2, 3)
a.distance(b)
```
Out[57]: $\frac{13\sqrt{14}}{14}$

In [58]:
```
a = Plane(Point3D(1, 2, 2), normal_vector=(1, 2, 3))
b = Line3D(Point3D(1, 3, 4), Point3D(2, 2, 2))
a.angle_between(b)
```
Out[58]: $- \operatorname{asin}{\left(\frac{\sqrt{21}}{6} \right)}$

In [59]:
```
a = Plane(Point3D(1, 2, 3), normal_vector=(1, 1, 1))
l = Line((-1, 2, 0), (5, -3, 0))
a.intersection(l)
```
Out[59]:
```
[Point3D(-37, 32, 0)]
```

In [60]:
```
d = Plane(Point3D(6, 0, 0), normal_vector=(1, 1, 1))
e = Plane(Point3D(2, 0, 0), normal_vector=(3, 4, -3))
d.intersection(e)
```
Out[60]:
```
[Line3D(Point3D(18, -12, 0), Point3D(11, -6, 1))]
```

In [61]:
```
a = Plane(Point3D(5, 0, 0), normal_vector=(1, -1, 1))
b = Plane(Point3D(0, -2, 0), normal_vector=(3, 1, 1))
Plane.are_concurrent(a, b)
```
Out[61]:
```
True
```

In [62]:
```
a = Plane(Point3D(1, 2, 3), normal_vector=(1, 1, 1))
b = Plane(Point3D(1, 2, 3), normal_vector=(2, 2, 2))
a.equals(b)
```

Out[62]: False

In [63]:
```
a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
b = Plane(Point3D(3,1,3), normal_vector=(4, 8, 12))
a.is_parallel(b)
```

Out[63]: False

In [64]:
```
a = Plane((1,4,6), (2, 4, 6))
l = Line(Point(1, 3, 4), Point(2, 2, 2))
a.is_parallel(l)
```

Out[64]: True

In [65]:
```
a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
b = Plane(Point3D(2, 2, 2), normal_vector=(-1, 2, -1))
a.is_perpendicular(b)
```

Out[65]: False

In [66]:
```
a = Plane(Point3D(1,4,6), normal_vector=(2, 4, 6))
l = Line3D(Point3D(1, 3, 4), Point3D(2, 2, 2))
a.is_perpendicular(l)
```

Out[66]: False

In [67]:
```
a = Plane(Point3D(1, 1, 1), normal_vector=(1, 1, 1))
c = Line3D(Point3D(1, 1, 1), Point3D(2, 2, 2))
a.projection_line(c)
```

Out[67]: $\mathrm{Point3D}(1, 1, 1)$

In [68]:
```
a = Plane(Point3D(1, 1, 1), normal_vector=(1, 1, 1))
c = Line3D(Point3D(0, 0, 0), Point3D(1, 0, 1))
a.projection_line(c)
```

Out[68]: $\mathrm{Line3D}\left(\mathrm{Point3D}(1, 1, 1), \mathrm{Point3D}\left(\frac{4}{3}, \frac{1}{3}, \frac{4}{3}\right)\right)$

In [69]:
```
a = Plane(Point3D(1, 1, 2), normal_vector=(1, 1, 1))
b = Point3D(0, 1, 2)
a.projection(b)
```

Out[69]: $\mathrm{Point3D}\left(\frac{1}{3}, \frac{4}{3}, \frac{7}{3}\right)$

Пример 8

In [70]:
```
def distance_line(A,B,M,N):
    AB = Point(B.x-A.x,B.y-A.y,B.z-A.z)
    l1 = Line(A,B)
    l2 = Line(M,N)
    p = Plane(A, normal_vector=AB)
    pl2 = p.projection_line(l2)
    d = pl2.distance(A)
    return(d)
```

In [71]:
```
A = Point(0,0,0)
D = Point(1,1,0)
E = Point(0,1,0)
B1 = Point(1,0,1)

distance_line(A,E,D, B1)
```

Out[71]: 1

Отрезок

In [72]:
```
s = Segment((1,0), (4,4))
s
```

Out[72]:

In [73]:
```
s.points
```

Out[73]:
```
(Point2D(1, 0), Point2D(4, 4))
```

In [74]:
```
s.slope
```

Out[74]: $\frac{4}{3}$

In [75]:
```
s.length
```

Out[75]: 5

In [76]:
```
s.midpoint
```

Out[76]: $\mathrm{Point2D}\left(\frac{5}{2}, 2\right)$

In [77]:
```
A = Point(0,2)
s.distance(A)
```

Out[77]: 2

In [78]:
```
s = Segment((0,2), (2,0))
s.perpendicular_bisector()
```

Out[78]:

In [79]:
```
Line(s)
```

Out[79]:

In [80]:
```
Line(s).equation()
```

Out[80]: $2x + 2y - 4$

Пример 9

In [81]:
```
def Point_oneside_L(A,B,l):
    s = Segment(A,B)
    return not(Line.are_concurrent(l, s))
```

In [82]:
```
l = Line((2,0), (0,2))
A = Point(1,0)
B = Point(0,1)
D = Point(1,2)
Point_oneside_L(A,B,l)
```

Out[82]:
```
True
```

Пример 10

In [83]:
```
Point_oneside_L(A,D,l)
```

Out[83]:
```
False
```

Пример 11

In [84]:
```
def Point_oneside_P(A,B,P):
    s = Segment(A,B)
    p = P.intersection(s)
    if len(p) == 0:
        return True
    else:
        return not(s.contains(p[0]))
```

In [85]:
```
P = Plane((3,0,0), (0,3,0), (0,0,3))
A = Point(0,0,0)
B = Point(5,5,5)
D = Point(1,0,0)
Point_oneside_P(A,D,P)
```

Out[85]:
```
True
```

In [86]:
```
Point_oneside_P(A,B,P)
```

Out[86]:
```
False
```

Пример 12

In [87]:
```
def Point_opposite_l(A,l):
    A0 = l.projection(A)
    x = 2*A0.x - A.x
    y = 2*A0.y - A.y
    if len(A) == 2:
        return Point(x,y)
    elif len(A) == 3:
        z = 2*A0.z - A.z
        return Point(x,y,z)
```

In [88]:
```
A = Point(0,7)
l = Line((0,-3),(2,1))
Point_opposite_l(A,l)
```

Out[88]: $\mathrm{Point2D}(8,3)$

Пример 13

In [89]:
```
def Point_opposite_P(A,P):
    A0 = P.projection(A)
    x =2*A0.x - A.x
    y =2*A0.y - A.y
    z =2*A0.z - A.z
    return Point(x,y,z)
```

In [90]:
```
A = Point(0,0,0)
P = Plane((3,0,0),(0,3,0),(0,0,3))
Point_opposite_P(A,l)
```

Out[90]: $\mathrm{Point3D}\left(\frac{12}{5}, -\frac{6}{5}, 0\right)$

Луч

In [91]:
```
Ray((0,0,0), (1,1,1))
```

Out[91]: $\mathrm{Ray3D}(\mathrm{Point3D}(0,0,0), \mathrm{Point3D}(1,1,1))$

In [92]:
```
r = Ray((0,0), (1,0))
r.rotate(-pi/2)
```

Out[92]:

Треугольник

In [93]:
```
A = Point(0,0)
B = Point(0,4)
D = Point(3,0)
Triangle(A,B,D)
```

Out[93]:

In [94]:
```
Triangle(sss=(1,1,1))
```

Out[94]:

In [95]:
```
Triangle(sas=(1,45,2))
```

Out[95]:

In [96]:
```
Triangle(asa=(90, 1, 30))
```

Out[96]:

In [97]:
```
T = Triangle(sss=(3,4,5))
A, B, D = T.vertices
B
```

Out[97]: $\mathrm{Point2D}(3,0)$

In [98]:
```
T.vertices[2]
```

Out[98]: $\mathrm{Point2D}(3,4)$

In [99]:
```
T.altitudes[T.vertices[0]]
```

Out[99]:

In [100…
```
T.orthocenter
```

Out[100]: $\mathrm{Point2D}(3,0)$

In [101…
```
T.medians[T.vertices[2]]
```

Out[101]:

In [102…
```
T.circumcircle
T.circumcenter
T.circumradius
```

Out[102]: $\frac{5}{2}$

In [103…
```
T.incircle
T. incenter
T.inradius
```

Out[103]: 1

In [104…
```
T.medial
```

Out[104]:

Пример 14

In [105…
```
T = Triangle(sas=(3,90,4))
O1 = T.incenter
O2 = T.circumcenter
O1.distance(O2)
```

Out[105]: $\frac{\sqrt{5}}{2}$

Многоугольник

In [106…
```
pl, p2, pz, p4 = [(0, 0), (4, 0), (5, 4), (0, 2)]
Polygon(pl, p2, p3, p4)
```

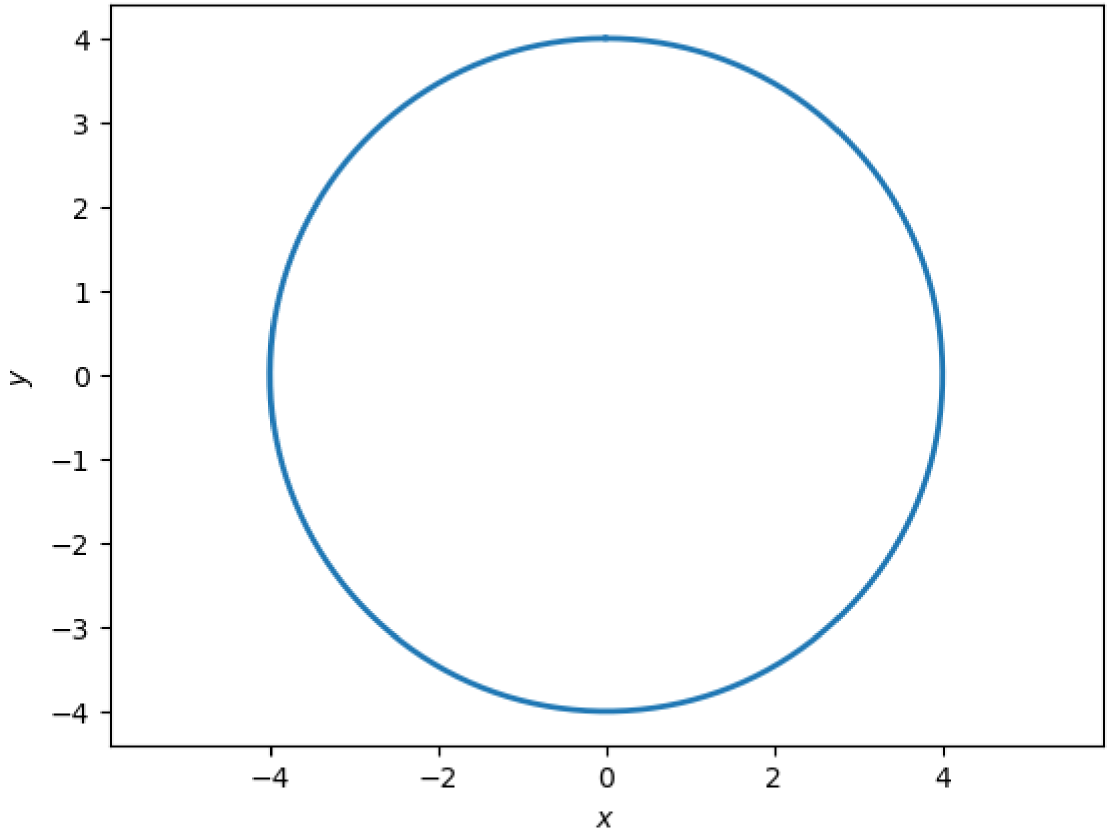
Out[106]:

Кривые второго порядка

Окружность

In [107…
```
r = 4
t = np.arange(0, 2*np.pi, 0.01)
```

```
x = r*np.sin(t)
y = r*np.cos(t)
plt.plot(x, y, lw=2)
plt.axis('equal')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.show()
```



In [108…

```
Circle(Point(0,0), 5)
```

Out[108]:

In [109…

```
Circle(Point(0,0), Point(0,1), Point(1,0))
```

Out[109]:

In [110…

```
c3 = Circle(Point(0, 0), 5)
c3.equation()
```

Out[110]: $x^2 + y^2 - 25$

In [111…

```
c = Circle(Point(0, 0), Point(1, 1), Point(1, 0))
c.hradius, c.vradius, c.radius
```

Out[111]:

```
(sqrt(2)/2, sqrt(2)/2, sqrt(2)/2)
```

In [112…

```
c.center
```

Out[112]: $\text{Point2D}\left(\frac{1}{2}, \frac{1}{2}\right)$

In [113…

```
s = c3.area
s
```

Out[113]: 25π

In [114…

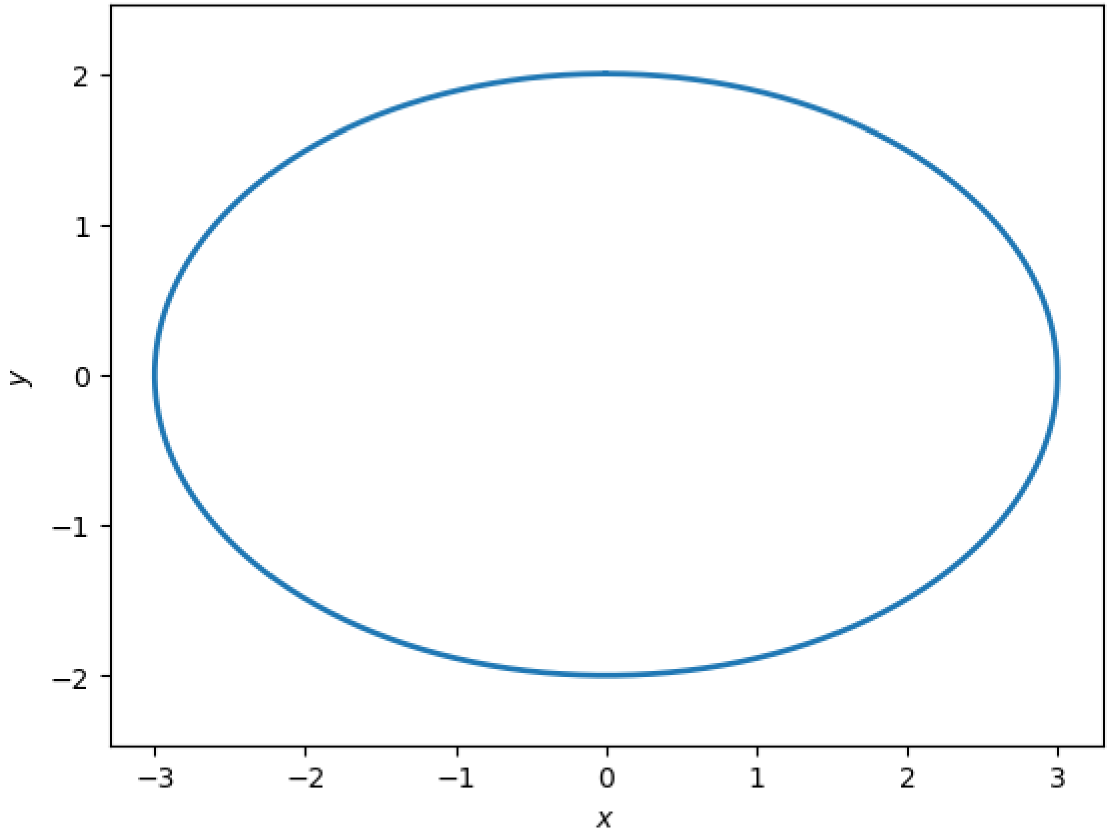
```
c4 = Circle(Point(3, 4), 6)
c4.circumference
```

Out[114]: 12π

Эллипс

In [115…

```
a = 3
b = 2
t = np.arange(0, 2*np.pi, 0.01)
x = a*np.sin(t)
y = b*np.cos(t)
plt.plot(x, y, lw=2)
plt.axis('equal')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.show()
```



In [116…

```
e1 = Ellipse(Point(0, 0), 5, 2)
e1
```

Out[116]:

In [117…
```
e2 = Ellipse(Point(3, 1), hradius=3, eccentricity=Rational(4, 5))

e2
```

Out[117]:

In [118…
```
e1 = Ellipse(Point(1, 0), 3, 2)
e1.equation()
```

Out[118]: $$\frac{y^2}{4} + \left(\frac{x}{3} - \frac{1}{3}\right)^2 - 1$$

In [119…
```
el = Ellipse(Point(0, 0), 3, 2)
el.arbitrary_point()
```

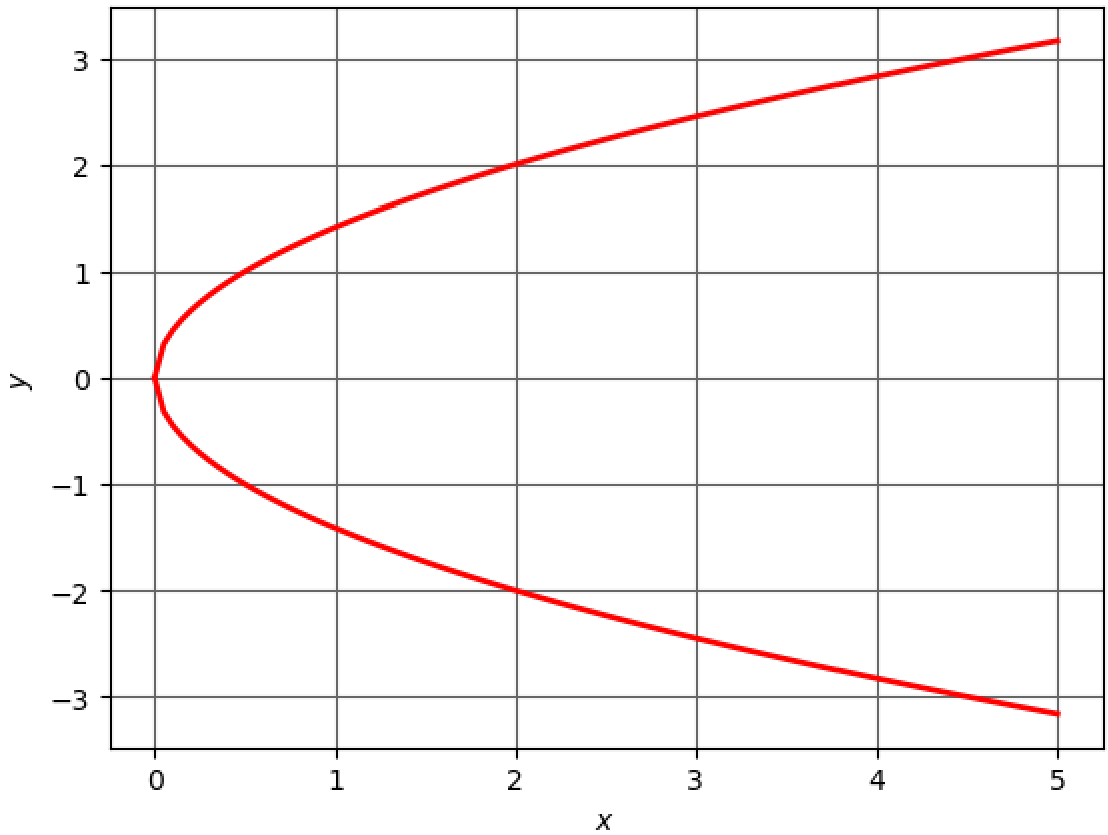
Out[119]: $\mathrm{Point2D}(3\cos(t), 2\sin(t))$

In [120…
```
p1 = Point(0, 0)
e1 = Ellipse(p1, 3, 1)
e1.area
```

Out[120]: 3π

Парабола

In [121…
```
x = np.linspace(0,5,100)
y1 = np.sqrt(2*x)
y2 = -np.sqrt(2*x)

plt.plot(x, y1, lw=2, color='r')
plt.plot(x, y2, lw=2, color='r')

plt .grid(True, linestyle='-', color='0.4')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.show()
```



In [122…

```
pl = Parabola(Point(0, 0), Line(Point(5, 8), Point(7,8)))
```

In [123…

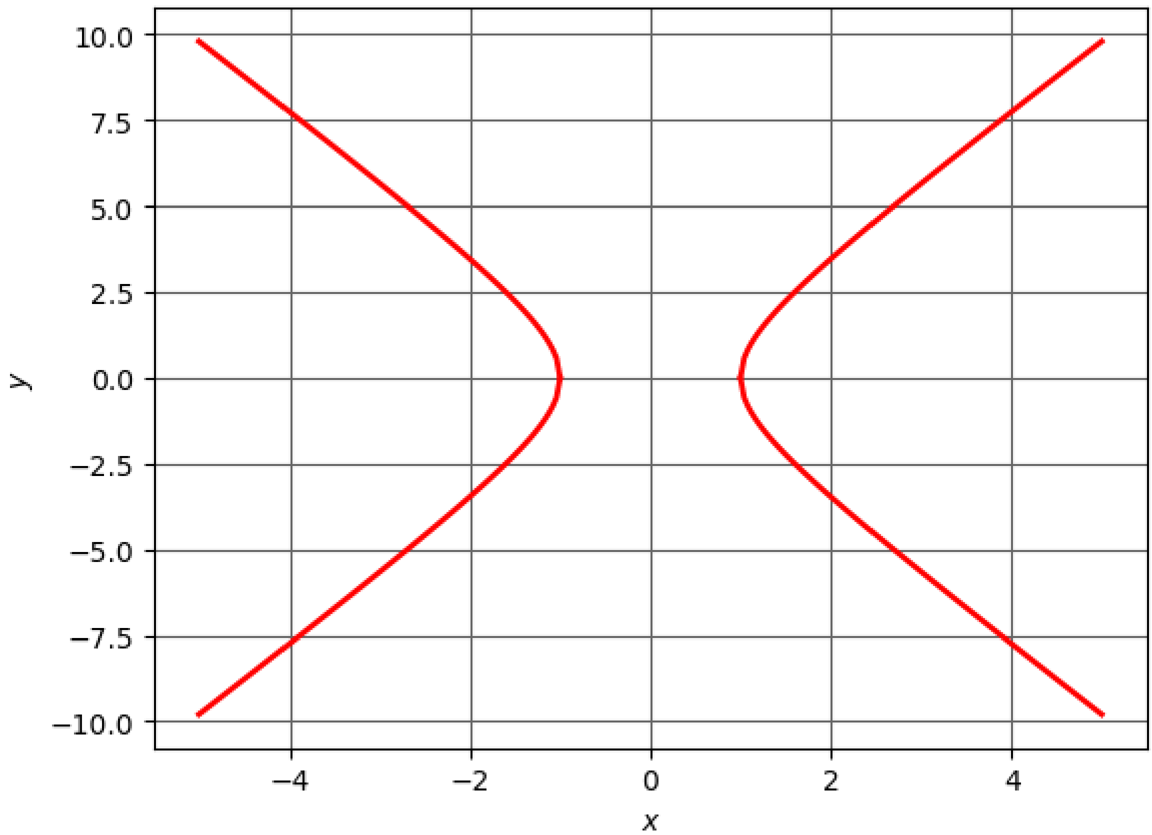
```
pl = Parabola(Point(0, 0), Line(Point(5, 8), Point(7, 8)))
pl.equation()
```

Out[123]: $-x^2 - 16y + 64$

Гипербола

In [124…

```
x1 = np.linspace(-5,-1,100)
y1 = 2*np.sqrt(x1**2-1)
y2 = -2*np.sqrt(x1**2-1)
x2 = np.linspace(1,5,100)
y3 = 2*np.sqrt(x2**2-1)
y4 = -2*np.sqrt(x2**2-1)
plt.plot(x1, y1, lw=2, color='r')
plt.plot(x1, y2, lw=2, color='r')
plt.plot(x2, y3, lw=2, color='r')
plt.plot(x2, y4, lw=2, color='r')
plt.grid(True, linestyle='-', color='0.4')
plt.ylabel('$y$')
plt.xlabel('$x$')
plt.show()
```



Приведение кривой второго порядка к каноническому виду

Пример 15

In [125…

```
x = Symbol('x')
y = Symbol('y')

M = Matrix([[x**2,x*y,y**2,x,y,1],
            [1**2,1*1,1**2,1,1,1],
            [0**2,0*1,1**2,0,1,1],
            [1**2,1*0,0**2,1,0,1],
            [2**2,2*3,3**2,2,3,1],
            [3**2,3*2,2**2,3,2,1]])

det(M)
```

Out[125]: $-8x^2 + 32xy - 24x - 8y^2 - 24y + 32$

Пример 16

In [126…

```
def conic_curve(A,a,f_transform=0):
    if (A.shape != (2,2)) or (len(a) != 3):
        raise ValueError('Invalid size of Aya matrices')

    a11 = A[0,0]; a12 = A[0,1]; a22 = A[1,1]
    a1 = a[0]; a2 = a[1]; a0 = a[2]
    D = det(A)
    Delta = det(Matrix([[a11,a12,a1],
    [a12,a22,a2],
    [a1, a2, a0]]))
```

```
    I = a11+a22
    B = det(Matrix([[a11,a1],
                    [a1, a0]])) + \
        det(Matrix([[a22,a2],
                    [a2, a0]]))

    if (Delta*I < 0) and (D > 0):
        print('ellipse')
    if (Delta != 0) and (D < 0):
        print('Hyperbola')
    if (Delta != 0) and (D == 0):
        print('Parabola')
    if (Delta == 0) and (D < 0):
        print(' pair of intersecting spindles')
    if (Delta == 0) and (D == 0) and (c < 0):
        print(' pair of parallel strands')
    if (Delta == 0) and (D == 0) and (B == 0):
        print(' Poyai')
    if (Delta == 0) and (D > e) and (B == e):
        print(' Point')
    if (Delta*I > 0) and (D > e):
        print(' min ellipse')
    if (Delta == 0) and (D == 0) and (B > e):
        print(' pair of imaginary parallel lines')

    T, _ = A.diagonalize()
    T1 = T.inv()
    n1 = sqrt(T1[0,0]**2+T1[1,0]**2)
    n2 = sqrt(T1[0,1]**2+T1[1,1]**2)
    x,y,x1,y1 = symbols('x y x1 y1')

    Q0 = a11*x**2 + 2*a12*x*y + a22*y**2 + \
        2*a1*x + 2*a2*y + a0
    x0 = (T1[0,0]/n1)*x1+(T1[1,0]/n1)*y1
    y0 = (T1[0,1]/n2)*x1+(T1[1,1]/n2)*y1

    Q = Q0.subs({x: x0, y: y0}).simplify()
    if (f_transform == 0):
        print('Equation: %s' % Q)
    else:
        print('Equation: %s' % Q)
        print('transition equation:')
        print('x = %s' % x0)
        print('y = %s' % y0)
```

In [127…

```
A = Matrix([[1,0],
[0,-4]])
a = Matrix([-2,0,-8])
conic_curve(A,a)
```

```
Hyperbola
Equation: -4*x1**2 + y1**2 - 4*y1 - 8
```

Пример 17

In [128…

```
A = Matrix([[1,1],
            [1,1]])
a = Matrix([2,0,1])
conic_curve(A,a,f_transform=1)
```

```
Parabola
Equation: -2*sqrt(2)*x1 + 2*y1**2 + 2*sqrt(2)*y1 + 1
transition equation:
x = -sqrt(2)*x1/2 + sqrt(2)*y1/2
y = sqrt(2)*x1/2 + sqrt(2)*y1/2
```

Кривая в пространстве

In [129…

```
t = symbols('t')
C = Curve((sin(t), cos(t)), (t, -pi, pi))
C.functions
```

Out[129]: $(\sin(t),\ \cos(t))$

In [130…

```
C.limits
```

Out[130]: $(t,\ -\pi,\ \pi)$

In [131…

```
C.parameter
```

Out[131]: t

Длина кривой

In [132…

```
x = symbols('x')
Curve((x, x**2), (x, 0, 1)).length
```

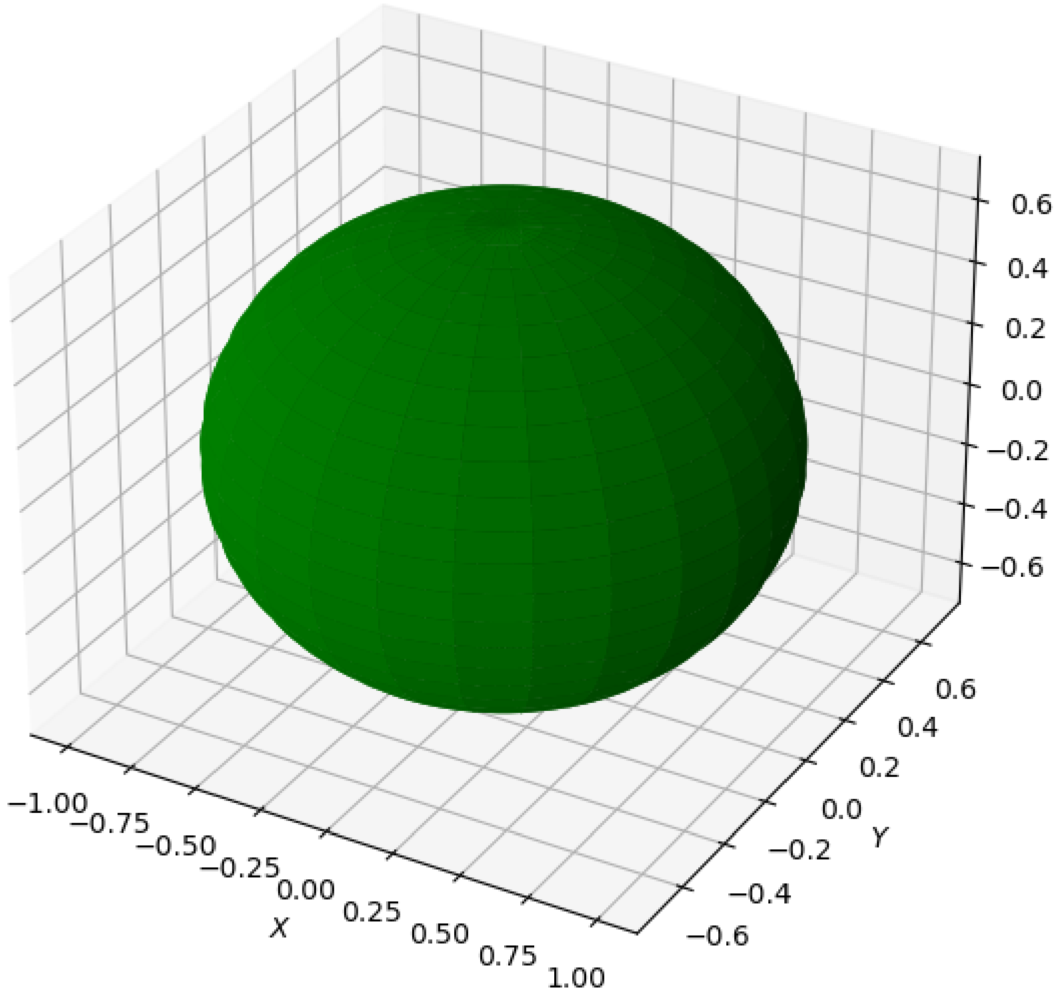
Out[132]: $\frac{\operatorname{asinh}(2)}{4} + \frac{\sqrt{5}}{2}$

Поверхности второго порядка

Элипсоид

In [133…

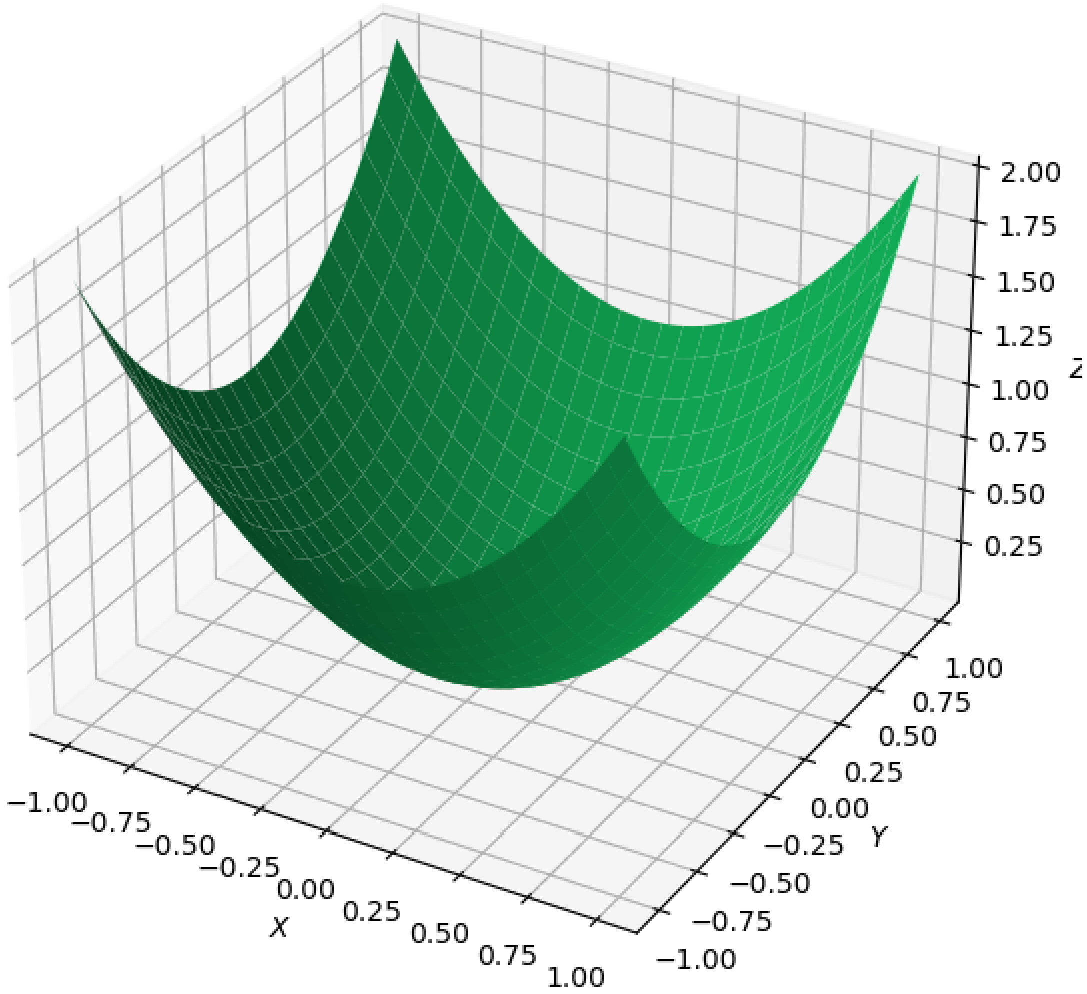
```
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(projection='3d')

coefs = (1, 2, 2)

rx, ry, rz = 1/np.sqrt(coefs)

u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)

x = rx * np.outer(np.cos(u), np.sin(v))
y = ry * np.outer(np.sin(u), np.sin(v))
z = rz * np.outer(np.ones_like(u), np.cos(v))

ax.plot_surface(x, y, z, rstride=4, cstride=4, color='g')

ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")

plt.show()
```



Элиптический параболоид

In [134…

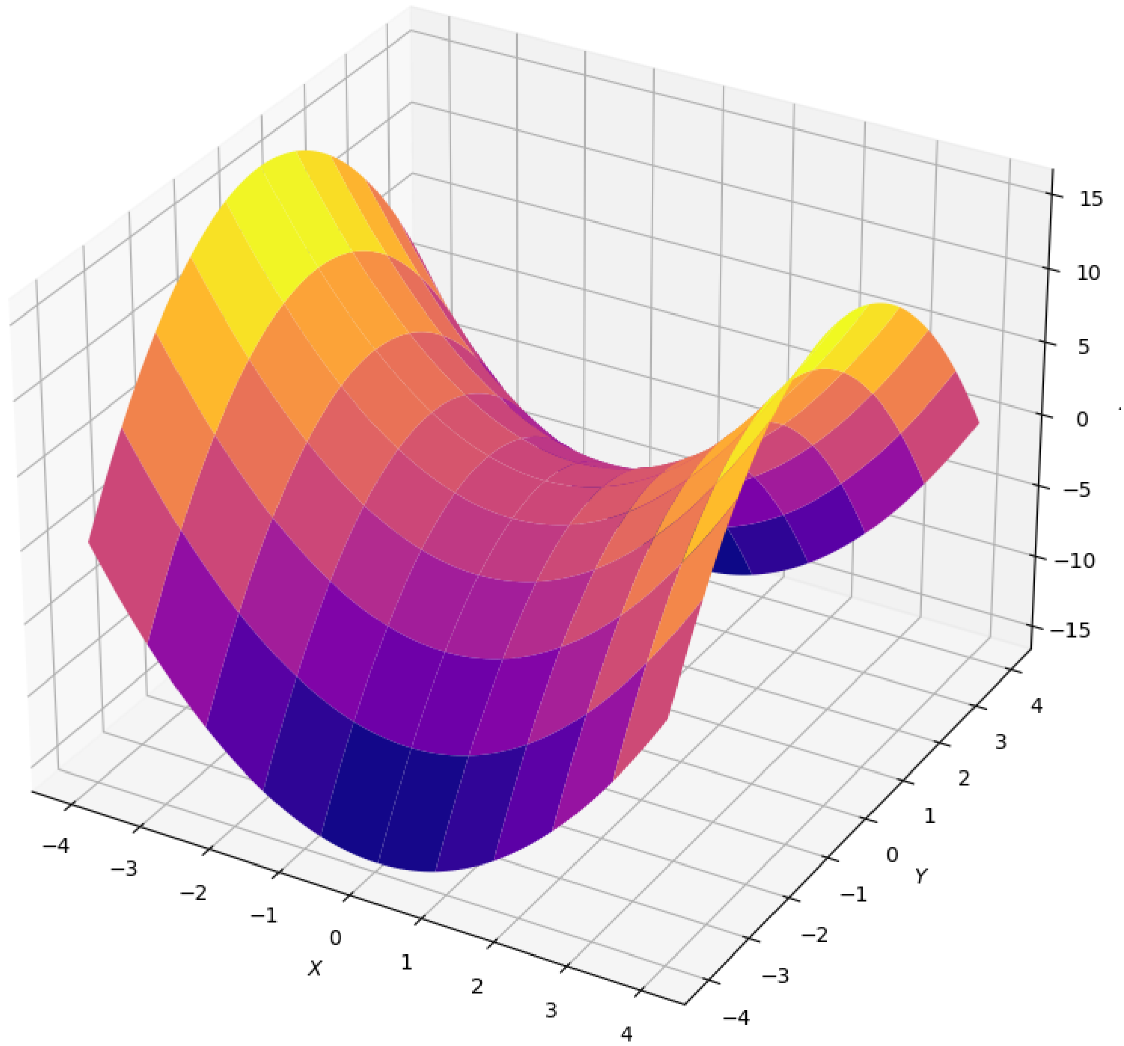
```
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(projection='3d')
x = np.linspace(-1,1,100);
y = np.linspace(-1,1,100);
[x,y] = np.meshgrid(x,y);
z = lambda w: w[0]**2 + w[1]**2
Z = z((x,y))

ax.plot_surface(x,y, Z, rstride=4, cstride=4, color='#11aa55')

ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")
plt.show()
```



Гиперболический параболоид

In [135…

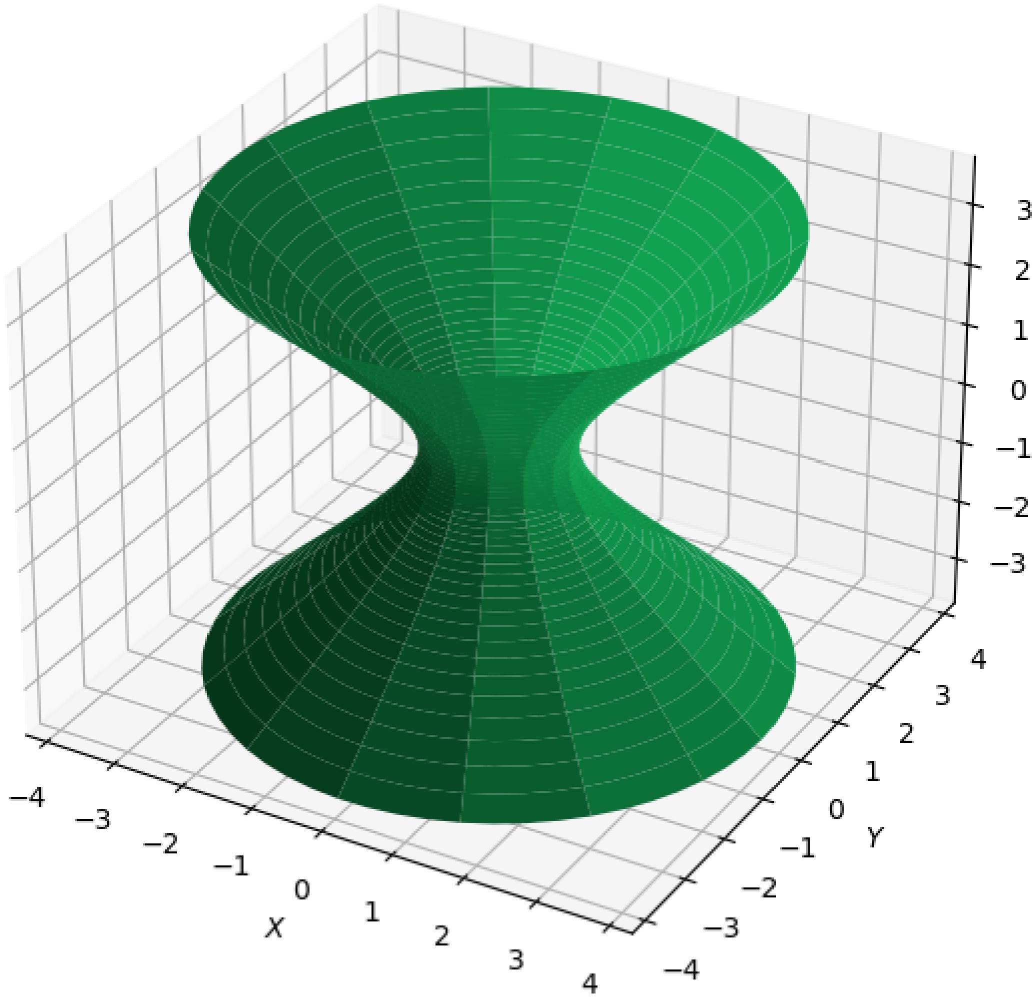
```
f = lambda x, y: x ** 2 - y ** 2

fig = plt.figure(figsize = (10, 10))
ax = fig.add_subplot(1, 1, 1, projection = '3d')
xval = np.linspace(-4, 4, 100)
yval = np.linspace(-4, 4, 100)
x, y = np.meshgrid(xval, yval)
z = f(x, y)

surf = ax.plot_surface(x, y, z, rstride = 10,\
                    cstride = 10,cmap = cm.plasma)

ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")

plt.show()
```



Однополосный гиперболоид

In [136…

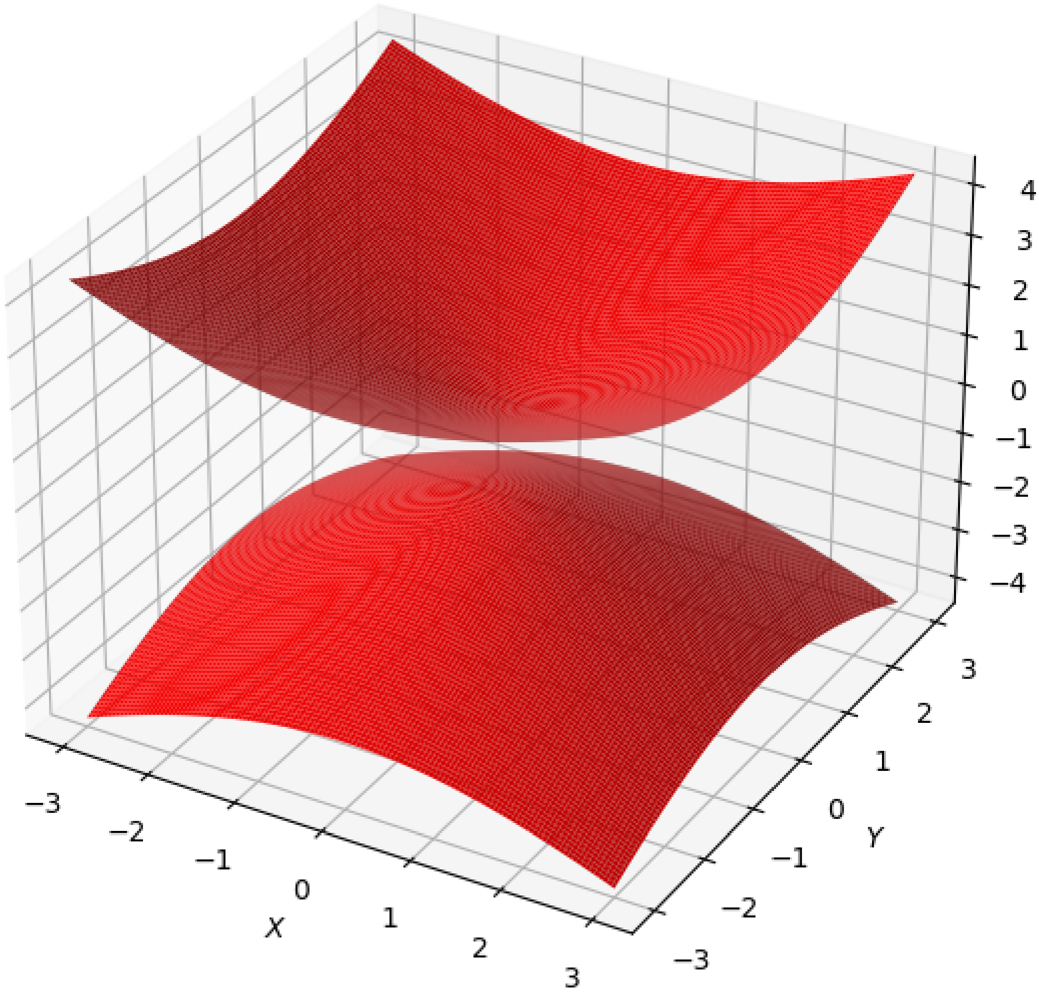
```
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(projection='3d')

u=np.linspace(-2,2,200);
v=np.linspace(0,2*np.pi,60);
[u,v]=np.meshgrid(u,v);

x = np.cosh(u)*np.cos(v)
y = np.cosh(u)*np.sin(v)
z = np.sinh(u)

ax.plot_surface(x, y, z, rstride=4, cstride=4, color='#11aa55')

ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")

plt.show()
```



Двухполостный гиперболоид

In [137…

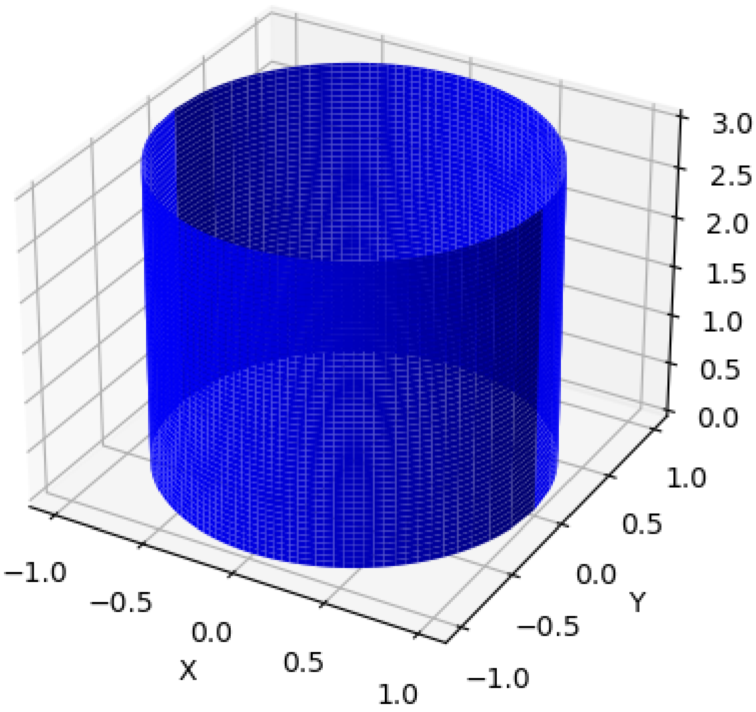
```
fig = plt.figure(figsize=(7,7))
ax = fig.add_subplot(projection='3d')
x = np.linspace(-3,3,500);
y = np.linspace(-3,3,500);
[x,y] = np.meshgrid(x,y);
z1 = lambda w: np.sqrt(w[0]**2 + w[1]**2 + 1)
Z1 = z1((x,y))
z2 = lambda w: -np.sqrt(w[0]**2 + w[1]**2 + 1)
Z2 = z2((x,y))
ax.plot_surface(x, y, Z1, rstride=4, cstride=4, color='r')
ax.plot_surface(x, y, Z2, rstride=4, cstride=4, color='r')
ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")
plt.show()
```



Элиптический цилиндр

In [138…

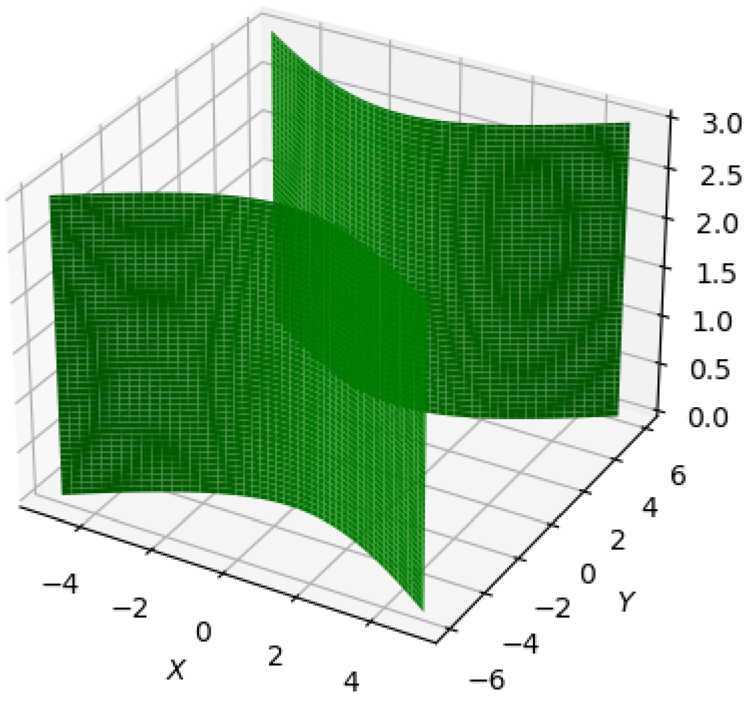
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-1, 1, 100)
z = np.linspace(0, 3, 100)
x, z = np.meshgrid(x, z)

y = np.sqrt(1-x**2)
ax.plot_surface(x, y, z, color='b')
ax.plot_surface(x, -y, z, color='b')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

plt.show()
```



Гиперболический цилиндр

In [139…

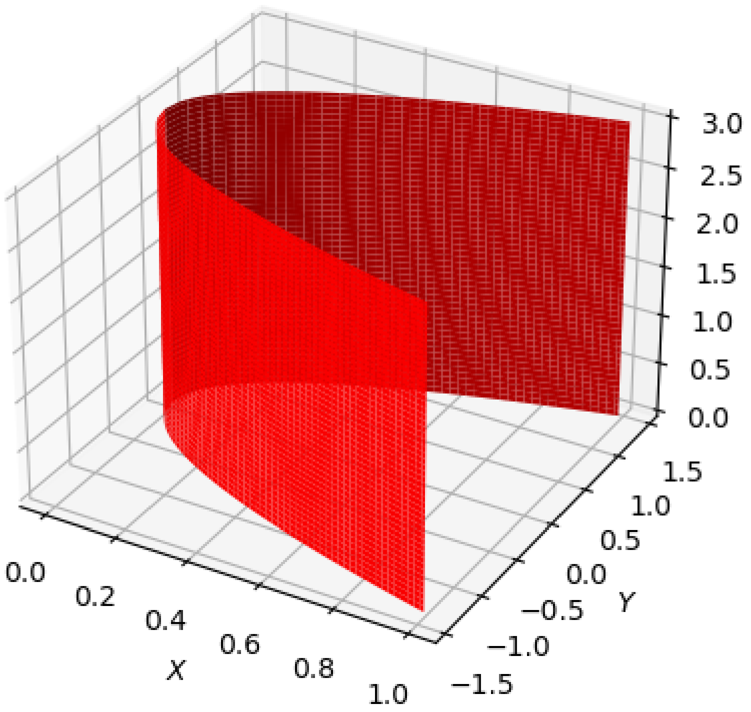
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(-5, 5, 100)
z = np.linspace(0, 3, 100)
x, z = np.meshgrid(x, z)
y = np.sqrt(10+x**2)
ax.plot_surface(x, y, z, color='g')
ax.plot_surface(x, -y, z, color='g')
ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")
plt.show()
```



Параболический цилиндр

In [140…

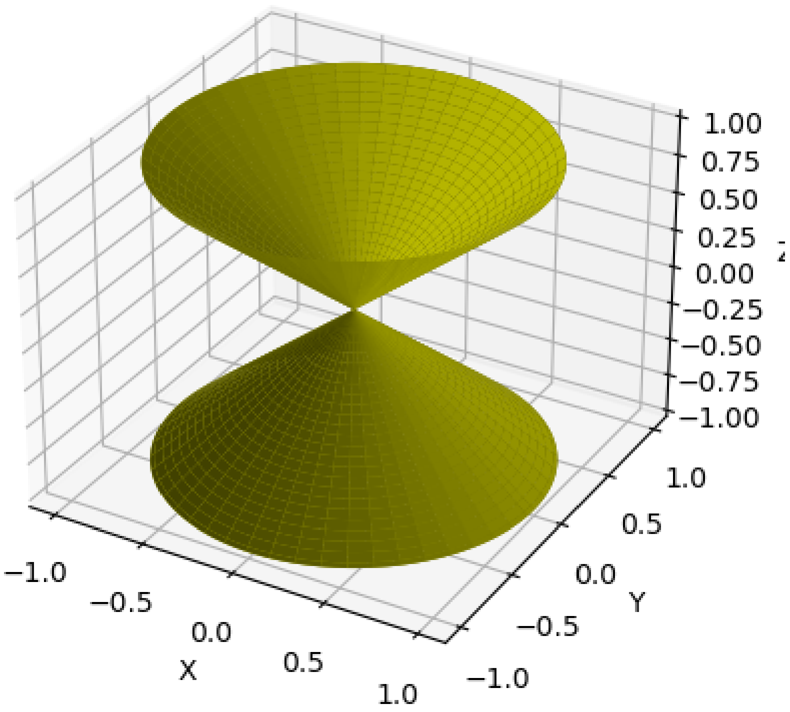
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.linspace(0, 1, 100)
z = np.linspace(0, 3, 100)
x, z = np.meshgrid(x, z)
y = np.sqrt(2*x)
ax.plot_surface(x, y, z, color='r')
ax.plot_surface(x, -y, z, color='r')
ax.set_xlabel("$X$")
ax.set_ylabel("$Y$")
ax.set_zlabel("$Z$")
plt.show()
```



Конус

In [141…

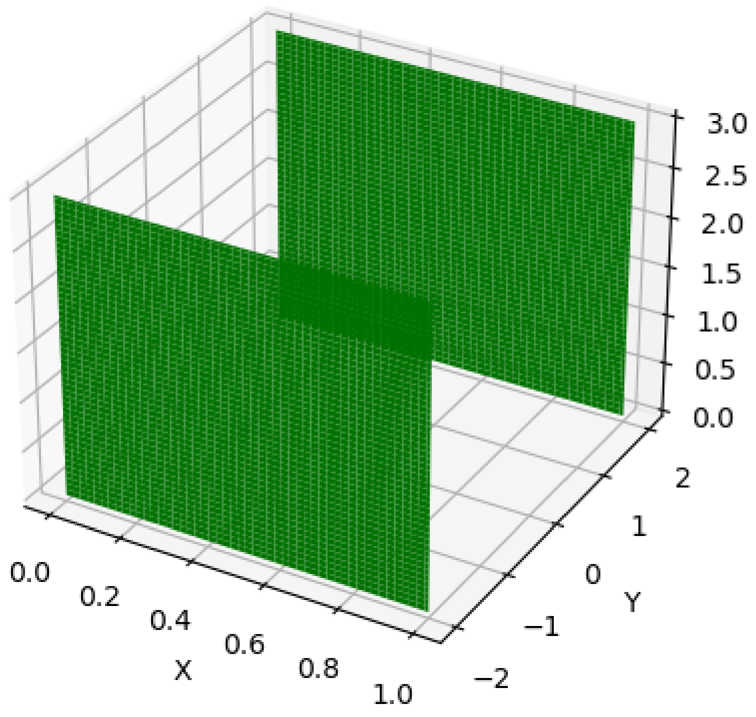
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
theta = np.linspace(0, 2*np.pi, 100)
r = np.linspace(-1, 1, 100)
t, R = np.meshgrid(theta, r)
x = R*np.cos(t)
y = R*np.sin(t)
z1 = R
z2 = -R
ax.plot_surface(x, y, z1, color='y')
ax.plot_surface(x, y, z2, color='y')
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")
plt.show()
```



Пара параллельных плоскостей

In [142…

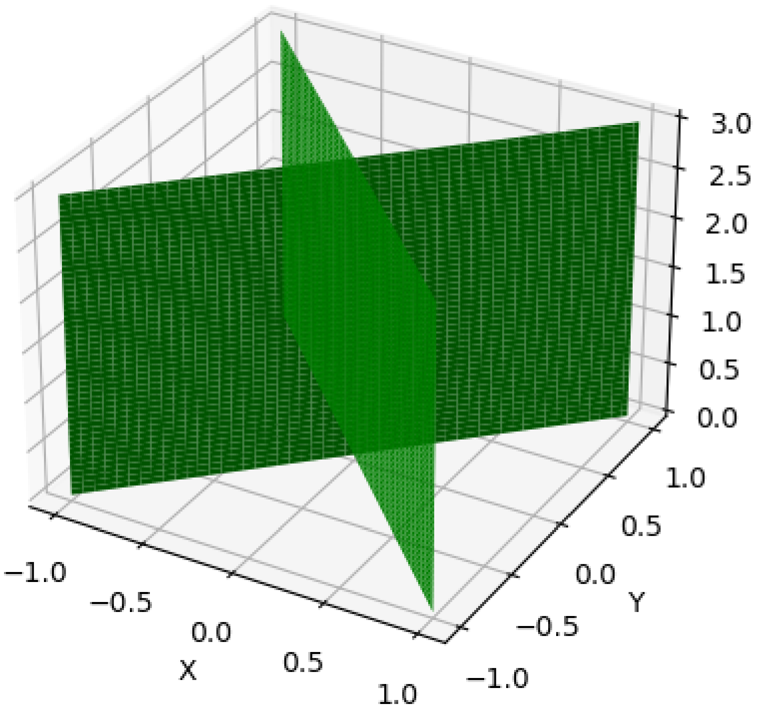
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(0, 1, 100)
z = np.linspace(0, 3, 100)
x, z = np.meshgrid(x, z)

y = 2

ax.plot_surface(x, y, z, color='g')
ax.plot_surface(x, -y, z, color='g')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

plt.show()
```



Пара пересекающихся плоскостей

In [143…

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x = np.linspace(-1, 1, 100)
z = np.linspace(0, 3, 100)
x, z = np.meshgrid(x, z)

y = x

ax.plot_surface(x, y, z, color='g')
ax.plot_surface(x, -y, z, color='g')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

plt.show()
```



Примеры решения задач

Даны векторы

$$a = (2, -3, 4, 1), b = (-6, 9, -12, -3), p = (3, 2, -1, 4).$$

Вычислить:

$$x = 2(a, b) \cdot p + 3b(p, p) - |b| \cdot b.$$

In [144…

```
a = np.array([2,-3,4,1])
b = np.array([-6,9,-12,-3])
p = np.array([3,2,-1,4])
x = 2 * np.dot(a,b) * p + 3 * b * np.dot(p,p)
x
```

Out[144]:

```
array([-1080,   450,  -900,  -990])
```

Определить, какой угол между векторами

$$a = (2, -3, 4, 1), b = (-6, 9, -12, -3):$$

острый, тупой или прямой.

In [145…

```
np.dot(a,b)
```

Out[145]:

```
-90
```

Являются ли ортогональными векторы

$$a = (2, 0, -3), b = (6, 1, 4)?$$

In [146…

```
a = (2, 0, -3)
b = (6, 1, 4)
np.dot(a,b)
```

Out[146]:

```
0
```

Прямую l, заданную общим уравнением

$$4x - 5y + 20 = 0,$$

записать в параметрическом виде.

In [147…

```
s = Line((2,8), (-3,4))
s.equation()
```

Out[147]: $4x - 5y + 32$

Найти направляющий вектор прямой, заданной уравнением

$$4x - 7y - 14 = 0.$$

In [148…

```
l = Line((0,4), (-5,0))
l.arbitrary_point()
```

Out[148]: $\mathrm{Point2D}(-5t, 4 - 4t)$

Найти направляющий вектор прямой, заданной уравнением

$$4x - 7y - 14 = 0$$

In [149…

```
s = Line((3.5,0), (0,-2))
s.direction
```

Out[149]: $\mathrm{Point2D}\left(-\frac{7}{2}, -2\right)$

Прямую s, представленную общим уравнением:

$$5x - 2y + 10 = 0$$

записать в каноническом виде.

In [150…

```
s = Line((-2,0), (0,5))
p = s.direction
P
```

Out[150]: $\mathrm{Plane}(\mathrm{Point3D}(3, 0, 0), (9,\ 9,\ 9))$

Прямую s, представленную общим уравнением:

$$x + y - 2 = 0,$$

записать в параметрическом виде.

In [151…

```
s = Line((2,0), (0,2))
s.arbitrary_point()
```

Out[151]: $\text{Point2D}(2 - 2t, 2t)$

Вычислить расстояние от точки $\mathrm{M}(5;4)$ до прямой, проходящей через точки $A(1;-2)$ и $B(0;3)$.

In [152…
```
M = Point(5,4)
s = Line((1,-2), (0,3))
s.distance(M)
```

Out[152]: $\sqrt{26}$

Написать уравнение прямой, проходящей через точку M и параллельной прямой l, если $M(-2;1), l : 3x - 2y + 12 = 0.$

In [153…
```
M = Point(-2,1)
l = Line((0,6), (-4,0))
l1 = l.parallel_line(M)
l1.equation()
```

Out[153]: $6x - 4y - 14$

Написать уравнение прямой, проходящей через точку M и перпендикулярной прямой l, если $M(3;-3), l : x + 2y - 4 = 0.$

In [154…
```
M = Point(3,-3)
l = Line((0,2), (4,0))
l1 = l.perpendicular_line(M)
l1.equation()
```

Out[154]: $-4x + 2y + 18$

Найти точку пересечения прямых

$$s_1 : 2x - 3y + 12 = 0 \text{ и } s_2 : x + y - 2 = 0.$$

In [155…
```
s1 = Line((0,4), (-6,0))
s2 = Line((0,2), (2,0))
s1.intersection(s2)
```

Out[155]:
```
[Point2D(-6/5, 16/5)]
```

Написать уравнение прямой, проходящей через точку (M) и точку пересечения прямых l_1 и l_2, если $M(2;0), l_1 : 2x - y - 1 = 0$, $l_2 : x + 3y - 4 = 0$

In [156…
```
M = Point(2,0)
l1 = Line((0,-1), (0.5,0))
l2 = Line((1,1), (4,0))
A = l1.intersection(l2)
s = Line(M,A[0])
s.equation()
```

Out[156]: $-x - y + 2$

Найти расстояние от точки $P(2;2)$ до прямой l, заданной в каноническом виде: $\frac{x-2}{3} = \frac{y+1}{2}$.

In [157…

```
P = Point(2,2)
l = Line((2,-1), (5,1))
l.distance(P)
```

Out[157]: $\frac{9\sqrt{13}}{13}$

Найти расстояние от точки $P(-2;2)$ до прямой l, записанной в параметрической форме:

$$x = 2t - 3, y = t + 2.$$

In [158…

```
p = Point(-2,2)
l = Line((-3,2), (-1,3))
l. distance(P)
```

Out[158]: $\sqrt{5}$

Найти угол пересечения прямой $x - 2y + 4 = 0$ с осью Ox.

In [159…

```
l1 = Line((0,2), (-4,0))
l2 = Line((0,0), (1,0))
l1. smallest_angle_between(l2)
```

Out[159]: $\operatorname{acos}\left(\frac{2\sqrt{5}}{5}\right)$

Найти координаты основания перпендикуляра, опущенного из начала координат на прямую l, заданную уравнением:

$$x - y - 17 = 0$$

In [160…

```
l = Line((17,0), (0,-17))
s = l.perpendicular_line((0,0))
l.intersection(s)
```

Out[160]:

```
[Point2D(17/2, -17/2)]
```

Найти расстояние между параллельными прямыми l_1, l_2, заданными уравнениями $3x - 4y - 2 = 0$ и $3x - 4y + 1 = 0$.

In [161…

```
l1 = Line((2,1), (6,4))
l2 = Line((1,1), (5,4))
l2.distance((2,1))
```

Out[161]: $\frac{3}{5}$

Дан куб $ABDEA_1B_1D_1E_1$ со стороной, равной 1. Найти угол между диагоналями AD_1 и B_1E.

In [162…

```
A = Point(0,0,0)
D1 = Point(1,1,1)
B1 = Point(1,1,0)
```

```
E = Point(0,1,0)
AD1 = Line(A,D1)
B1E = Line(B1,E)
AD1.angle_between(B1E)
```

Out[162]: $\operatorname{acos}\left(-\frac{\sqrt{3}}{3}\right)$

Найти проекцию точки $A(-1;1)$ на прямую $s: 2x+3y=6$.

In [163…
```
A = Point(-1,1)
s = Line((3,0), (0,2))
s.projection(A)
```

Out[163]: $\text{Point2D}\left(-\frac{3}{13}, \frac{28}{13}\right)$

Написать уравнение медианы и высоты, проведенных из вершины A треугольника ABC, если заданы вершины:

$$A(-1;-5), B(3;-1), C(1;-2)$$

In [164…
```
A, B, C = Point(-3, -2), Point(0, 4), Point(6, 0)
M = B.midpoint(C)
s = Line(A, M)
s.equation()
```

Out[164]: $-4x+6y$

In [165…
```
s = Line(B, C)
l2 = s.perpendicular_line(A)
l2
```

Out[165]:

In [166…
```
l2.equation()
```

Out[166]: $-6x+4y-10$

Найти уравнение прямой l, являющейся пересечением плоскостей p_1 и p_2, если p_1 проходит через точки $(0;1;2), (2;1;3)$ и $2;-2;5)$, а p2 проходит через точки $(-1;3;-2), (4;0;1)$ и $(5;1;0)$.

In [167…
```
p1 = Plane((0,1,2), (2,1,3), (2,-2,5))
p2 = Plane((-1,3,-2), (4,0,1), (5,1,0))
l = p1.intersection(p2)
l
```

Out[167]:
```
[Line3D(Point3D(-4, 1, 0), Point3D(12, -23, 24))]
```

In [168…
```
s = l[0]
s.equation()
```

Out[168]: $(3x + 2y + 10, \ -3x + 2z - 12)$

In [169…
```
p1.equation()
```
Out[169]: $-x^2 - 16y + 64$

In [170…
```
p2.equation()
```
Out[170]: $8y + 8z - 8$

Выяснить характер расположения прямых AB и CD (пересекаются, параллельны или скрещиваются), где $A(1;1;1)$, $B(3;-2;0)$, $C(1;0;1)$, $D(2;1;0)$

In [171…
```
s1 = Line3D((1,1,1), (3,-2,0))
s2 = Line3D((1,0,1), (2,1,0))
Line.are_concurrent(s1, s2)
```
Out[171]:
```
False
```

In [172…
```
Line.is_parallel(s1, s2)
```
Out[172]:
```
False
```

In [173…
```
s1.is_similar(s2)
```
Out[173]:
```
False
```

Доказать, что три медианы треугольника пересекаются в одной точке.

In [174…
```
a1, a2, b1, b2, d1, d2 = symbols('al a2 y b2 dl d2')

A = Point(a1,a2)
B = Point(b1,b2)
D = Point(d1,d2)

M = B.midpoint(D)
N = D.midpoint(A)
K = A.midpoint(B)
```

In [175…
```
AM = Line(A, M)
BN = Line(B, N)
DK = Line(D, K)

Line.are_concurrent(AM, BN, DK)
```
Out[175]:
```
True
```

Определить тип кривой второго порядка

$$13x^2 + 18xy + 37y^2 - 26x - 18y - 27 = 0$$

In [176…
```
A = Matrix([[13,9],
            [9, 37]])
a = Matrix([-13,-9,-27])
conic_curve(A,a,f_transform=1)
```

```
ellipse
Equation: 10*x1**2 + 6*sqrt(10)*x1 + 40*y1**2 - 8*sqrt(10)*y1 - 27
transition equation:
x = -3*sqrt(10)*x1/10 + sqrt(10)*y1/10
y = sqrt(10)*x1/10 + 3*sqrt(10)*y1/10
```

Применение аналитической геометрии в решении собственной задачи

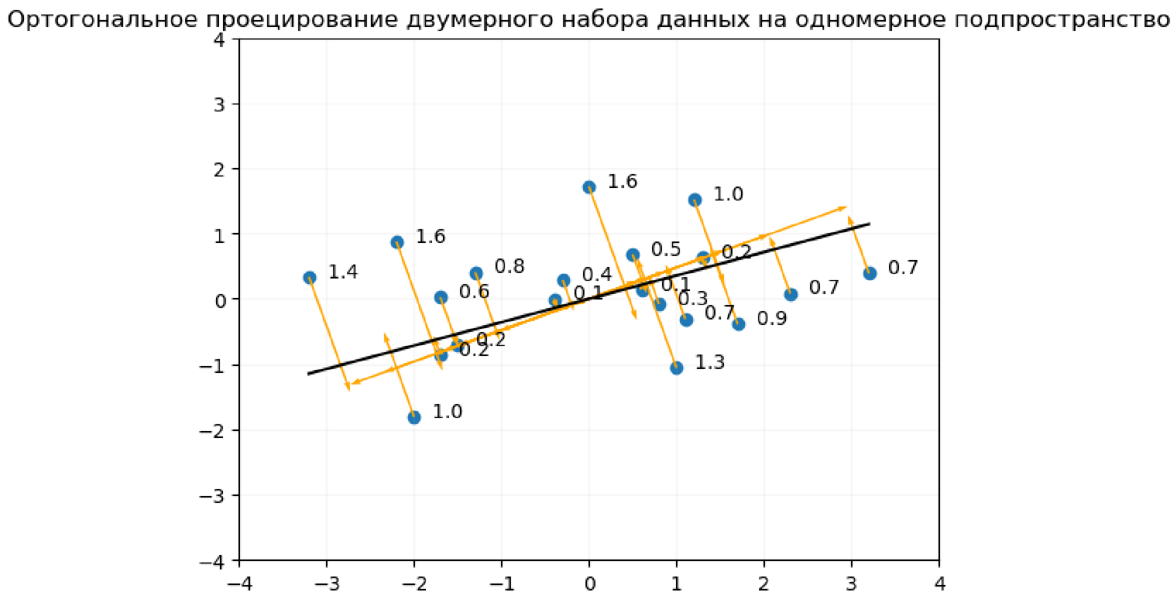
Предположим, что нам необходимо совершить ортогональное проецирование двумерного набора данных на одномерное подпространство. Данная операция применяется в машинном обучении. Причина, по которой ортогональная проекция из всех возможных проекций отвечает нашим интересам, заключается в свойстве ближайшего вектора. Согласно ему,среди всех векторов в пространстве W вектор, ближайший к U это ортогональная проекция U на W. Другими словами, мы хотим получить проекцию, наиболее близкую к исходному набору данных, чтобы сохранить как можно больше информации после уменьшения измерения.

In [177…

```
# Инициализация входных данных и примененеие метода наименьших квадратов в качестве
origin2D = np.vstack([0,0])
xs = np.vstack([-3.2,-2.2,-2,-1.7,-1.7,-1.5,-1.3,-.4,-.3,0,.5,.6,.8,1,1.1,1.2,1.3,
ys = np.vstack(np.random.normal(0,.8,len(xs)))
points = np.hstack([xs,ys])
lsq = np.linalg.lstsq(points,np.random.randn(20))[0]

posPts = [p for p in points if p[0]>=0] # Разделение данных на положительные и отр
negPts = [p for p in points if p[0]<0]
maxNeg = np.hstack([xs[0],[xs*(lsq[0]/lsq[1])][0][0]])
maxPos = np.hstack([xs[::-1][0],[xs*(lsq[0]/lsq[1])][0][::-1][0]])
```

In [178…

```
# Настройка графика
plt.axis([-4,4,-4,4])
plt.plot(xs,xs*(lsq[0]/lsq[1]),c="k")
plt.scatter(points[:,0],points[:,1])
plt.grid(alpha=.1)

# Итеративный поиск какждой проекции
for i in negPts:
    newQ = (np.dot(i,maxNeg)/norm(maxNeg)**2)*maxNeg
    plt.quiver(*origin2D,*newQ,scale=8,width=.003,color="orange")
    currV = newQ-i
    plt.quiver(*i,*currV,scale=8,width=.003,color="orange")
    plt.annotate("    " + str(round(norm(currV),1)),i)
for i in posPts:
    newQ = (np.dot(i,maxPos)/norm(maxPos)**2)*maxPos
    plt.quiver(*origin2D,*newQ,scale=8,width=.003,color="orange")
    currV = newQ-i
    plt.quiver(*i,*currV,scale=8,width=.003,color="orange")
    plt.annotate("    " + str(round(norm(currV),1)),i)

plt.title("Ортогональное проецирование двумерного набора данных на одномерное подп
```



Примеры проекций на одномерные подпространства.

In [179…
```
v1 = np.vstack([1,2])
v2 = np.vstack([2,1])
projec = lambda v1,v2: (np.dot(v2.T,v1.T)/norm(v1.T)**2) * v2
v3 = projec(v1.ravel(),v2.ravel())
v4 = v3.ravel() - v1.ravel()
Matrix(v1)
```

Out[179]:
$$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

In [180…
```
Matrix(v2)
```

Out[180]:
$$\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

In [181…
```
Matrix(v3)
```

Out[181]:
$$\begin{bmatrix} 1.6 \\ 0.8 \end{bmatrix}$$

In [182…
```
Matrix(v4)
```

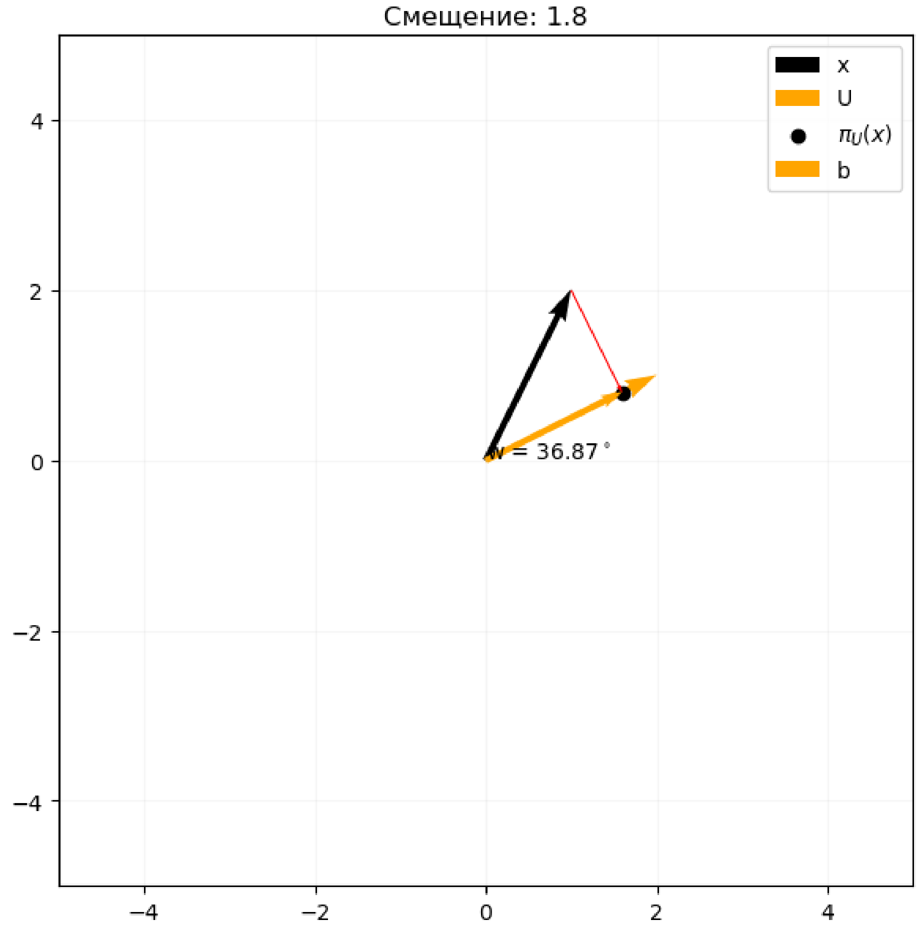
Out[182]:
$$\begin{bmatrix} 0.6 \\ -1.2 \end{bmatrix}$$

In [183…
```
plt.rcParams[ "figure.figsize" ] = (7,7)
```

In [184…
```
scale = 10
angle = lambda cos: np.arccos(cos)*(180/np.pi)
cosAngle = lambda x, y: (np.dot(x.T,y)/np.sqrt(np.dot(np.dot(x.T,x),np.dot(y.T,y)))
plt.axis([-scale/2,scale/2,-scale/2,scale/2])
plt.suptitle(r"(a) Проекция $x \in \mathbb{R}^2$ на подпространство $U$ с базисным
plt.title(f"Смещение: {norm(v3):.1f}")
plt.grid(alpha=.1)

plt.quiver(*origin2D, *v1, scale=scale, color = "k")
plt.quiver(*origin2D, *v2, scale=scale, color = "orange")
plt.scatter(*v3,color="k")
```

```python
plt.quiver(*origin2D,*v3,scale=scale, width = .005,color="orange")
plt.quiver(*v1,*v4,scale=scale,width=.002,color="red")

plt.annotate("w = " + str(round(angle(cosAngle(v1,v2)),2))+r"$^\circ$",xy=origin2D
plt.legend(["x","U",r"$\pi_U(x)$","b"]);
```

(a) Проекция $x \in \mathbb{R}^2$ на подпространство U с базисным вектором b.



Проекция $\pi_U(x)$ ближе всего к x, где "ближе всего" означает, что расстояние $||x - \pi_U(x)||$ минимально. Отсюда следует, что (красный) отрезок ортогонален U, а значит и базисному вектору b из U. Условие ортогональности дает $\langle \pi_U(x) - x, b \rangle = 0$, так как углы между векторами определяются через внутреннее произведение.

In [185…
```python
round(np.inner(v3 - v1.T,v2.T)[0][0],5) # Проверка условия ортогональности
```

Out[185]:
```
-0.0
```

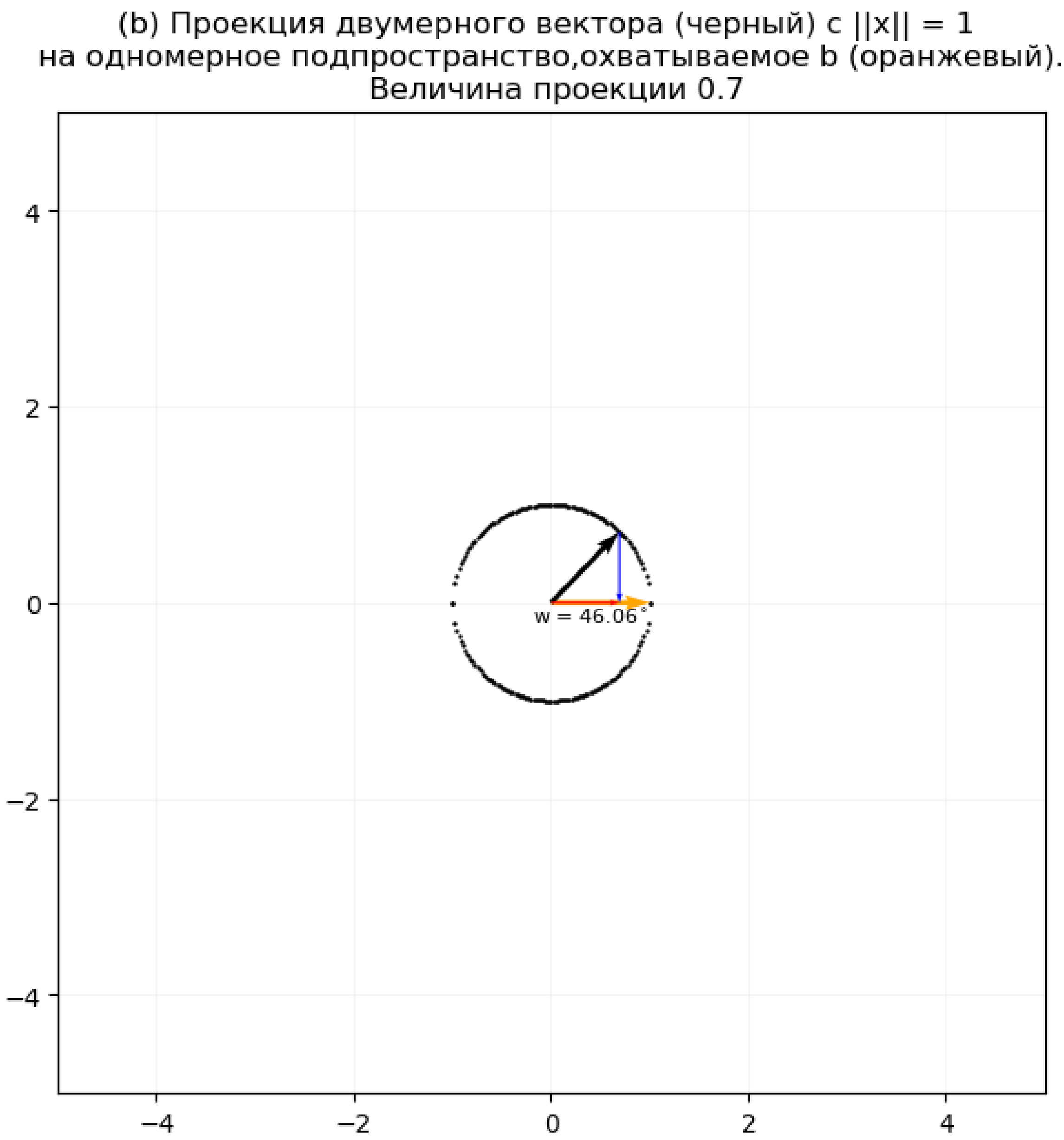
Проекция $\pi_U(x)$ от x на U должна быть леммой U и, следовательно, кратной базисному вектору b, который охватываетU. Следовательно, $\pi_U(x) = \lambda b$, для некоторой $\lambda \in \mathbb{R}$.

In [186…
```python
xRight = np.linspace(0,1,50)
xLeft = np.linspace(-1,0,50)
l2normData = np.hstack([
    np.vstack(np.vstack([xRight,xLeft,xLeft,xRight]).ravel()),
    np.vstack(np.vstack([np.sqrt(1-xRight**2), np.sqrt(-xLeft**2+1),
        -np.sqrt(-xLeft**2+1), -np.sqrt(1-xRight**2)]).ravel())
])
v1 = np.vstack(np.array([1,0]))
v2 = np.vstack(l2normData[34:35][0])
```

```
v3 = projec(v2.ravel(),v1.ravel()) #
v4 = v3.ravel()-v2.ravel()
```

In [187…

```
plt.axis([-scale/2,scale/2,-scale/2,scale/2])
plt.grid(alpha=.1)
plt.scatter(l2normData[:,0],l2normData[:,1],s=.5,color="k")
plt.title(
    "(b) Проекция двумерного вектора (черный) с ||x|| = 1 \n на одномерное подпрос
        "охватываемое b (оранжевый)." + f" \n Величина проекции {norm(v3):.1f}"
)

plt.quiver(*origin2D, *v1,scale=scale, color="orange", width=.005)
plt.quiver(*origin2D, *v2,scale=scale, color="k", width=.005)
plt.annotate("w = " + str(round(angle(cosAngle(v1,v2)),2))+r"$^\circ$",xy=origin2D
plt.quiver(*origin2D,*v3,scale=scale,width=.002,color="red")
plt.quiver(*v2,*v4,scale=scale,width=.002,color="blue");
```



Три шага для определения проекции между любым вектором x и некоторым базисным вектором b для подпространства U:

1. Найдите скаляр, λ для b. Учитывая, что некоторая проекция p существует, внутреннее произведение между x-p и b будет равно 0.

   a. Мы знаем, что проекция p также может быть записана как скалярная операция

над нашим базисным вектором b, поэтому мы говорим $\langle x - \lambda b, b \rangle = 0$.
b. Затем мы можем выделить скаляр, λ, увидев, что он равен внутреннему произведению x и b минус $\lambda \langle b, b \rangle = 0$
c. $\lambda = \frac{\langle x,b \rangle}{\langle b,b \rangle}$ что также можно записать как внутреннее произведение b и x, деленное на квадрат нормы b, $\frac{\langle b,x \rangle}{||b||^2}$.

2. Найдите точку на U, которую создаст проекция p. Так как $p = \lambda b$, то по формуле (c) сверху находим $p = \frac{\langle x,b \rangle}{\langle b,b \rangle} b$.
3. Найдите матрицу проекции P_p. Мы знаем, что проекция - это просто линейное отображение, поэтому мы должны быть в состоянии найти матрицу, P, которая берет любой вектор, x, и отображает его, или создает новый проецируемый вектор, p, на наше подпространство или линию, так что $p = P_p x$, и $P_p = \frac{p}{x}$.
a. Так как $p = \frac{\langle x,b \rangle}{\langle b,b \rangle} b$, из 2. выше, $P_p = \frac{1}{x} \frac{\langle x,b \rangle}{\langle b,b \rangle} b = \frac{bb^T}{\langle b,b \rangle}$.