lab1_m_n

May 20, 2025

[104]:
```
#importy
import numpy as np
import os
import math as m
import cmath
import time
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import tkinter as tk
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from tkinter import Menu
from scipy.optimize import newton
from scipy.interpolate import interp1d
import sympy as sp
from scipy.integrate import solve_ivp
```

[2]:
```
#zadanie 1.1
A=np.array([[2,-7],[5,4]])
B=np.array([[6,1],[4,-3]])
f=np.array([4,1])
print(A)
print(B)
print(f)
```

```
[[ 2 -7]
 [ 5  4]]
[[ 6  1]
 [ 4 -3]]
[4 1]
```

[5]:
```
#zadanie 1.2
print("a)")
print("Wielkość macierzy A: ",A.shape)
print("Ilość elementów macierzy A: ",A.size)
print("Wielkość wektora f: ",len(f))
print("b)")
B_T=B.T
print(B_T)
```

```
print("c)")
C=(A+B)@(A+B)+2*(A-B)
print(C)
print("d)")
C=np.hstack([A,B])
print(C)
h=np.hstack([f,f])
print(h)
print("e)")
D=C*h
print(D)
```

```
a)
Wielkość macierzy A:  (2, 2)
Ilość elementów macierzy A:  4
Wielkość wektora f:  2
b)
[[ 6  4]
 [ 1 -3]]
c)
[[  2 -70]
 [ 83 -39]]
d)
[[ 2 -7  6  1]
 [ 5  4  4 -3]]
[4 1 4 1]
e)
[[ 8 -7 24  1]
 [20  4 16 -3]]
```

[7]:
```
#zadanie 1.3
print("a)",locals())

with open("dane.txt", "w") as f:
    for var, val in locals().items():
        f.write(f"{var} = {val}\n")
```

```
a) {'__name__': '__main__', '__doc__': 'Automatically created module for IPython
interactive environment', '__package__': None, '__loader__': None, '__spec__':
None, '__builtin__': <module 'builtins' (built-in)>, '__builtins__': <module
'builtins' (built-in)>, '_ih': ['', '#importy\nimport numpy as np\nimport
os\nimport math as m\nimport cmath\nimport time\nimport matplotlib.pyplot as
plt\nfrom mpl_toolkits.mplot3d import Axes3D\nimport tkinter as tk\nfrom
matplotlib.backends.backend_tkagg import FigureCanvasTkAgg', '#zadanie 1.1\nA=np
.array([[2,-7],[5,4]])\nB=np.array([[6,1],[4,-
3]])\nf=np.array([4,1])\nprint(A)\nprint(B)\nprint(f)', 'import json\nimport
getpass\nimport hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
```

```
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values =
get_ipython().run_line_magic(\'who_ls\', \'\')\n    dataframes = [\n        {\n
"name": var,\n            "type": type(eval(var)).__name__,\n            "hash":
dataframe_hash(var),\n            "cols": dataframe_columns(var),\n
"dtypesStr": dtypes_str(var),\n        }\n        for var in values if
is_data_frame(var)\n    ]\n    result = {"dataframes": dataframes, "user":
user}\n    return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()',
'import json\nimport getpass\nimport hashlib\n\ndef import_pandas_safely():\n
try:\n        return __import__(\'pandas\')\n    except ImportError:\n
return False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v:
str):\n    obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame)
or isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values =
get_ipython().run_line_magic(\'who_ls\', \'\')\n    dataframes = [\n        {\n
"name": var,\n            "type": type(eval(var)).__name__,\n            "hash":
dataframe_hash(var),\n            "cols": dataframe_columns(var),\n
"dtypesStr": dtypes_str(var),\n        }\n        for var in values if
is_data_frame(var)\n    ]\n    result = {"dataframes": dataframes, "user":
user}\n    return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()',
'#zadanie 1.2\nprint("a")")\nprint("Wielkość macierzy A: ",A.shape)\nprint("Ilość
elementów macierzy A: ",A.size)\nprint("Wielkość wektora f: ",len(f))\nprint("b)
")\nB_T=B.T\nprint(B_T)\nprint("c)")\nC=(A+B)@(A+B)+2*(A-
B)\nprint(C)\nprint("d)")\nC=np.hstack([A,B])\nprint(C)\nh=np.hstack([f,f])\npri
nt(h)\nprint("e)")\nD=C*h\nprint(D)', 'import json\nimport getpass\nimport
hashlib\n\ndef import_pandas_safely():\n    try:\n        return
```

__import__(\'pandas\')\n   except ImportError:\n   return False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n obj = eval(v)\n   if isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):\n   return True\n\n\ndef dataframe_columns(var):\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return [[df.name, str(df.dtype)]]\n return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef dtypes_str(frame):\n   return str(eval(frame).dtypes)\n\ndef dataframe_hash(var):\n   # Return a hash including the column names and number of rows\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-8\')).hexdigest()\n\ndef get_dataframes():\n   if __pandas is None:\n return []\n   user = getpass.getuser()\n   values = get_ipython().run_line_magic(\'who_ls\', \'\')\n   dataframes = [\n   {\n "name": var,\n   "type": type(eval(var)).__name__,\n   "hash": dataframe_hash(var),\n   "cols": dataframe_columns(var),\n "dtypesStr": dtypes_str(var),\n   }\n   for var in values if is_data_frame(var)\n   ]\n   result = {"dataframes": dataframes, "user": user}\n   return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()', '#zadanie 1.3\nprint("a)",locals())\n\nwith open("dane.txt", "w") as f:\n   for var, val in locals().items():\n   f.write(f"{var} = {val}\\n")'], '_oh': {3: '{"dataframes": [], "user": "skier"}', 4: '{"dataframes": [], "user": "skier"}', 6: '{"dataframes": [], "user": "skier"}'}, '_dh': [WindowsPath('C:/Users/skier')], 'In': ['', '#importy\nimport numpy as np\nimport os\nimport math as m\nimport cmath\nimport time\nimport matplotlib.pyplot as plt\nfrom mpl_toolkits.mplot3d import Axes3D\nimport tkinter as tk\nfrom matplotlib.backends.backend_tkagg import FigureCanvasTkAgg', '#zadanie 1.1\nA=np.array([[2,-7],[5,4]])\nB=np.array([[6,1],[4,-3]])\nf=np.array([4,1])\nprint(A)\nprint(B)\nprint(f)', 'import json\nimport getpass\nimport hashlib\n\ndef import_pandas_safely():\n   try:\n   return __import__(\'pandas\')\n   except ImportError:\n   return False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n obj = eval(v)\n   if isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):\n   return True\n\n\ndef dataframe_columns(var):\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return [[df.name, str(df.dtype)]]\n return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef dtypes_str(frame):\n   return str(eval(frame).dtypes)\n\ndef dataframe_hash(var):\n   # Return a hash including the column names and number of rows\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-8\')).hexdigest()\n\ndef get_dataframes():\n   if __pandas is None:\n return []\n   user = getpass.getuser()\n   values = get_ipython().run_line_magic(\'who_ls\', \'\')\n   dataframes = [\n   {\n

"name": var,\n   "type": type(eval(var)).__name__,\n   "hash": dataframe_hash(var),\n   "cols": dataframe_columns(var),\n "dtypesStr": dtypes_str(var),\n   }\n   for var in values if is_data_frame(var)\n   ]\n   result = {"dataframes": dataframes, "user": user}\n   return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()', 'import json\nimport getpass\nimport hashlib\n\ndef import_pandas_safely():\n try:\n   return __import__(\'pandas\')\n   except ImportError:\n return False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n   obj = eval(v)\n   if isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):\n   return True\n\n\ndef dataframe_columns(var):\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return [[df.name, str(df.dtype)]]\n return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef dtypes_str(frame):\n   return str(eval(frame).dtypes)\n\ndef dataframe_hash(var):\n   # Return a hash including the column names and number of rows\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-8\')).hexdigest()\n\ndef get_dataframes():\n   if __pandas is None:\n return []\n   user = getpass.getuser()\n   values = get_ipython().run_line_magic(\'who_ls\', \'\')\n   dataframes = [\n   {\n "name": var,\n   "type": type(eval(var)).__name__,\n   "hash": dataframe_hash(var),\n   "cols": dataframe_columns(var),\n "dtypesStr": dtypes_str(var),\n   }\n   for var in values if is_data_frame(var)\n   ]\n   result = {"dataframes": dataframes, "user": user}\n   return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()', '#zadanie 1.2\nprint("a")")\nprint("Wielkość macierzy A: ",A.shape)\nprint("Ilość elementów macierzy A: ",A.size)\nprint("Wielkość wektora f: ",len(f))\nprint("b")")\nB_T=B.T\nprint(B_T)\nprint("c)")\nC=(A+B)@(A+B)+2*(A-B)\nprint(C)\nprint("d)")\nC=np.hstack([A,B])\nprint(C)\nh=np.hstack([f,f])\nprint(h)\nprint("e)")\nD=C*h\nprint(D)', 'import json\nimport getpass\nimport hashlib\n\ndef import_pandas_safely():\n   try:\n   return __import__(\'pandas\')\n   except ImportError:\n   return False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n obj = eval(v)\n   if isinstance(obj, __pandas.core.frame.DataFrame) or isinstance(obj, __pandas.core.series.Series):\n   return True\n\n\ndef dataframe_columns(var):\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return [[df.name, str(df.dtype)]]\n return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef dtypes_str(frame):\n   return str(eval(frame).dtypes)\n\ndef dataframe_hash(var):\n   # Return a hash including the column names and number of rows\n   df = eval(var)\n   if isinstance(df, __pandas.core.series.Series):\n   return hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-8\')).hexdigest()\n\ndef get_dataframes():\n   if __pandas is None:\n return []\n   user = getpass.getuser()\n   values =

```
get_ipython().run_line_magic(\'who_ls\', \'\')\n    dataframes = [\n        {\n
"name": var,\n            "type": type(eval(var)).__name__,\n            "hash":
dataframe_hash(var),\n            "cols": dataframe_columns(var),\n
"dtypesStr": dtypes_str(var),\n        }\n        for var in values if
is_data_frame(var)\n    ]\n    result = {"dataframes": dataframes, "user":
user}\n    return json.dumps(result, ensure_ascii=False)\n\n\nget_dataframes()',
'#zadanie 1.3\nprint("a)",locals())\n\nwith open("dane.txt", "w") as f:\n    for
var, val in locals().items():\n        f.write(f"{var} = {val}\\n")'], 'Out':
{3: '{"dataframes": [], "user": "skier"}', 4: '{"dataframes": [], "user":
"skier"}', 6: '{"dataframes": [], "user": "skier"}'}, 'get_ipython': <bound
method InteractiveShell.get_ipython of <ipykernel.zmqshell.ZMQInteractiveShell
object at 0x0000022CBB8931D0>>, 'exit': <IPython.core.autocall.ZMQExitAutocall
object at 0x0000022CBB8DAC60>, 'quit': <IPython.core.autocall.ZMQExitAutocall
object at 0x0000022CBB8DAC60>, 'open': <function open at 0x0000022CB98E3240>,
'_': '{"dataframes": [], "user": "skier"}', '__': '{"dataframes": [], "user":
"skier"}', '___': '{"dataframes": [], "user": "skier"}', '__session__':
'C:\\Users\\skier\\lab1_m_n.ipynb', '_i': '\nimport json\nimport getpass\nimport
hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values = %who_ls\n    dataframes =
[\n        {\n            "name": var,\n            "type":
type(eval(var)).__name__,\n            "hash": dataframe_hash(var),\n
"cols": dataframe_columns(var),\n            "dtypesStr": dtypes_str(var),\n
}\n        for var in values if is_data_frame(var)\n    ]\n    result =
{"dataframes": dataframes, "user": user}\n    return json.dumps(result,
ensure_ascii=False)\n\n\nget_dataframes()', '_ii': '#zadanie
1.2\nprint("a)")\nprint("Wielkość macierzy A: ",A.shape)\nprint("Ilość elementów
macierzy A: ",A.size)\nprint("Wielkość wektora f: ",len(f))\nprint("b)")\nB_T=B.
T\nprint(B_T)\nprint("c)")\nC=(A+B)@(A+B)+2*(A-
B)\nprint(C)\nprint("d)")\nC=np.hstack([A,B])\nprint(C)\nh=np.hstack([f,f])\npri
nt(h)\nprint("e)")\nD=C*h\nprint(D)', '_iii': '\nimport json\nimport
getpass\nimport hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
```

```
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values = %who_ls\n    dataframes =
[\n        {\n            "name": var,\n            "type":
type(eval(var)).__name__,\n            "hash": dataframe_hash(var),\n
"cols": dataframe_columns(var),\n            "dtypesStr": dtypes_str(var),\n
}\n        for var in values if is_data_frame(var)\n    ]\n    result =
{"dataframes": dataframes, "user": user}\n    return json.dumps(result,
ensure_ascii=False)\n\n\nget_dataframes()', '_i1': '#importy\nimport numpy as
np\nimport os\nimport math as m\nimport cmath\nimport time\nimport
matplotlib.pyplot as plt\nfrom mpl_toolkits.mplot3d import Axes3D\nimport
tkinter as tk\nfrom matplotlib.backends.backend_tkagg import FigureCanvasTkAgg',
'np': <module 'numpy' from 'C:\\Users\\skier\\anaconda3\\Lib\\site-
packages\\numpy\\__init__.py'>, 'os': <module 'os' (frozen)>, 'm': <module
'math' (built-in)>, 'cmath': <module 'cmath' (built-in)>, 'time': <module 'time'
(built-in)>, 'plt': <module 'matplotlib.pyplot' from
'C:\\Users\\skier\\anaconda3\\Lib\\site-packages\\matplotlib\\pyplot.py'>,
'Axes3D': <class 'mpl_toolkits.mplot3d.axes3d.Axes3D'>, 'tk': <module 'tkinter'
from 'C:\\Users\\skier\\anaconda3\\Lib\\tkinter\\__init__.py'>,
'FigureCanvasTkAgg': <class
'matplotlib.backends.backend_tkagg.FigureCanvasTkAgg'>, '_i2': '#zadanie 1.1\nA=
np.array([[2,-7],[5,4]])\nB=np.array([[6,1],[4,-
3]])\nf=np.array([4,1])\nprint(A)\nprint(B)\nprint(f)', 'A': array([[ 2, -7],
       [ 5,  4]]), 'B': array([[ 6,  1],
       [ 4, -3]]), 'f': array([4, 1]), '_i3': '\nimport json\nimport
getpass\nimport hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
```

```
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values = %who_ls\n    dataframes =
[\n        {\n            "name": var,\n            "type":
type(eval(var)).__name__,\n            "hash": dataframe_hash(var),\n
"cols": dataframe_columns(var),\n            "dtypesStr": dtypes_str(var),\n
}\n        for var in values if is_data_frame(var)\n    ]\n    result =
{"dataframes": dataframes, "user": user}\n    return json.dumps(result,
ensure_ascii=False)\n\n\nget_dataframes()', 'json': <module 'json' from
'C:\\Users\\skier\\anaconda3\\Lib\\json\\__init__.py'>, 'getpass': <module
'getpass' from 'C:\\Users\\skier\\anaconda3\\Lib\\getpass.py'>, 'hashlib':
<module 'hashlib' from 'C:\\Users\\skier\\anaconda3\\Lib\\hashlib.py'>,
'import_pandas_safely': <function import_pandas_safely at 0x0000022CBB912F20>,
'__pandas': <module 'pandas' from 'C:\\Users\\skier\\anaconda3\\Lib\\site-
packages\\pandas\\__init__.py'>, 'is_data_frame': <function is_data_frame at
0x0000022CBB913920>, 'dataframe_columns': <function dataframe_columns at
0x0000022CBD860040>, 'dtypes_str': <function dtypes_str at 0x0000022CBD8605E0>,
'dataframe_hash': <function dataframe_hash at 0x0000022CBB913EC0>,
'get_dataframes': <function get_dataframes at 0x0000022CBCCD6DE0>, '_3':
'{"dataframes": [], "user": "skier"}', '_i4': '\nimport json\nimport
getpass\nimport hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values = %who_ls\n    dataframes =
[\n        {\n            "name": var,\n            "type":
type(eval(var)).__name__,\n            "hash": dataframe_hash(var),\n
"cols": dataframe_columns(var),\n            "dtypesStr": dtypes_str(var),\n
}\n        for var in values if is_data_frame(var)\n    ]\n    result =
{"dataframes": dataframes, "user": user}\n    return json.dumps(result,
ensure_ascii=False)\n\n\nget_dataframes()', '_4': '{"dataframes": [], "user":
"skier"}', '_i5': '#zadanie 1.2\nprint("a)")\nprint("Wielkość macierzy A:
",A.shape)\nprint("Ilość elementów macierzy A: ",A.size)\nprint("Wielkość
wektora f: ",len(f))\nprint("b)")\nB_T=B.T\nprint(B_T)\nprint("c)")\nC=(A+B)@(A+
B)+2*(A-
B)\nprint(C)\nprint("d)")\nC=np.hstack([A,B])\nprint(C)\nh=np.hstack([f,f])\npri
nt(h)\nprint("e)")\nD=C*h\nprint(D)', 'B_T': array([[ 6,  4],
       [ 1, -3]]), 'C': array([[ 2, -7,  6,  1],
```

```
       [ 5,  4,  4, -3]]), 'h': array([4, 1, 4, 1]), 'D': array([[ 8, -7, 24,
1],
       [20,  4, 16, -3]]), '_i6': '\nimport json\nimport getpass\nimport
hashlib\n\ndef import_pandas_safely():\n    try:\n        return
__import__(\'pandas\')\n    except ImportError:\n        return
False\n\n\n__pandas = import_pandas_safely()\n\n\ndef is_data_frame(v: str):\n
obj = eval(v)\n    if  isinstance(obj, __pandas.core.frame.DataFrame) or
isinstance(obj, __pandas.core.series.Series):\n        return True\n\n\ndef
dataframe_columns(var):\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return [[df.name, str(df.dtype)]]\n
return list(map(lambda col: [col, str(df[col].dtype)], df.columns))\n\n\ndef
dtypes_str(frame):\n    return str(eval(frame).dtypes)\n\ndef
dataframe_hash(var):\n    # Return a hash including the column names and number
of rows\n    df = eval(var)\n    if isinstance(df,
__pandas.core.series.Series):\n        return
hashlib.sha256(f"{var}-{df.name},{len(df)}".encode(\'utf-8\')).hexdigest()\n
return hashlib.sha256(f"{var}-{\',\'.join(df.columns)},{len(df)}".encode(\'utf-
8\')).hexdigest()\n\ndef get_dataframes():\n    if __pandas is None:\n
return []\n    user = getpass.getuser()\n    values = %who_ls\n    dataframes =
[\n        {\n            "name": var,\n            "type":
type(eval(var)).__name__,\n            "hash": dataframe_hash(var),\n
"cols": dataframe_columns(var),\n            "dtypesStr": dtypes_str(var),\n
}\n        for var in values if is_data_frame(var)\n    ]\n    result =
{"dataframes": dataframes, "user": user}\n    return json.dumps(result,
ensure_ascii=False)\n\n\nget_dataframes()', '_6': '{"dataframes": [], "user":
"skier"}', '_i7': '#zadanie 1.3\nprint("a)",locals())\n\nwith open("dane.txt",
"w") as f:\n    for var, val in locals().items():\n        f.write(f"{var} =
{val}\\n")'}
```


```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
Cell In[7], line 5
      2 print("a)",locals())
      4 with open("dane.txt", "w") as f:
----> 5     for var, val in locals().items():
      6         f.write(f"{var} = {val}\n")

RuntimeError: dictionary changed size during iteration
```

[9]:
```
#zadanie 1.4
print("a)")
x = np.arange(1, 25)
print(x)
print("b)")
Y=x.reshape(6,4,order='F')
print(Y)
```

```
a)
[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24]
b)
[[ 1  7 13 19]
 [ 2  8 14 20]
 [ 3  9 15 21]
 [ 4 10 16 22]
 [ 5 11 17 23]
 [ 6 12 18 24]]
```

[11]:
```
#zadanie 1.5
print("a)")
res=m.exp(2*m.sin(2*m.pi))
print(res)
print("b)")
res=m.cos(m.pi/3)**4
print(res)
print("c)")
res=m.log(m.sqrt(5))
print(res)
```

```
a)
0.9999999999999996
b)
0.06250000000000006
c)
0.8047189562170503
```

[15]:
```
#zadanie 1.6
z=3-2j
print(abs(z))
print(cmath.phase(z),m.degrees(cmath.phase(z)))
print(z.conjugate())
```

```
3.605551275463989
-0.5880026035475675 -33.690067525979785
(3+2j)
```

[17]:
```
#zadanie 1.7
E=np.ones((3,4))
F=np.zeros((3,4))
G=np.random.rand(3,4)
print(E)
print(F)
print(G)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[0.87342194 0.08787475 0.00890878 0.89700339]
 [0.73316883 0.04753934 0.78491897 0.59127939]
 [0.92093044 0.98462991 0.70596491 0.48952745]]
```

[19]:
```
#zadanie 2.1
A=np.zeros((6,6),dtype=float)
for i in range(6):
    for j in range(6):
        if i!=j:
            A[i,j]=1/((i+1)-(j+1))
print(A)
```

```
[[ 0.         -1.         -0.5        -0.33333333 -0.25       -0.2       ]
 [ 1.          0.         -1.         -0.5        -0.33333333 -0.25      ]
 [ 0.5         1.          0.         -1.         -0.5        -0.33333333]
 [ 0.33333333  0.5         1.          0.         -1.         -0.5       ]
 [ 0.25        0.33333333  0.5         1.          0.         -1.        ]
 [ 0.2         0.25        0.33333333  0.5         1.          0.        ]]
```

[21]:
```
#zadanie 2.2
A=np.random.rand(10,10)
for i in range (10):
    for j in range (10):
        if A[i,j]<0.5 and A[i,j]>0.2:
            print(A[i,j])
```

```
0.264591313796227
0.2850587421413634
0.259290804621257
0.3417733358028311
0.49917777050012846
0.3113398620260531
0.21754538537162071
0.33639397689464523
0.326414332945943
0.3697827495427054
0.22204800913187317
0.406565349203754
0.4769165213960317
0.25222194303013845
0.35423785797143603
0.24903219241536378
0.32436399008457506
0.31242336328090126
0.42032668220585345
0.4049266627353425
```

```
0.2273144759806608
0.26321684851451155
0.21909854733549228
0.4799884187457989
0.32890566373934027
0.4956019491023308
0.27243640370711364
0.4520535400461212
0.3885083010125442
0.28041934600713236
0.31402091119312925
0.32341294830842326
0.3255916620534074
0.290666089883017
0.3956972526316559
```

[25]:
```
#zadanie 2.3
def zadanie_2_3(a,b):
   return m.sin(a)*m.cos(b)

print(zadanie_2_3(1,2))
```

```
-0.35017548837401463
```

[27]:
```
#zadanie 2.4
def zadanie_2_4(n,A,B):
    A_inv=np.linalg.inv(A)
    B_T=B.T
    S=A+B
    R=(A-B)
    I=A*B
    C=A*3
    print("A_inv:",A_inv)
    print("B_T:",B_T)
    print("S:",S)
    print("R:",R)
    print("I:",I)
    print("C:",C)

n=3
A=np.random.randn(3,3)
B=np.random.randn(3,3)
print("A:",A)
print("B:",B)
zadanie_2_4(n,A,B)
```

```
A: [[-0.77455296 -0.41768144  0.33097259]
 [-1.10347912 -0.05604999  0.83704839]
 [ 0.23333234  1.28124544 -0.90602106]]
```

```
B: [[ 1.03615759 -1.02983238 -0.40005381]
 [ 0.0280338   0.83193316 -0.60496342]
 [-0.0893473  -0.54702933  0.38548552]]
A_inv: [[-1.5392637   0.06874446 -0.49878714]
 [-1.2120049   0.94092251  0.42654337]
 [-2.11036568  1.34830528 -0.62898805]]
B_T: [[ 1.03615759  0.0280338  -0.0893473 ]
 [-1.02983238  0.83193316 -0.54702933]
 [-0.40005381 -0.60496342  0.38548552]]
S: [[ 0.26160464 -1.44751382 -0.06908122]
 [-1.07544532  0.77588317  0.23208498]
 [ 0.14398505  0.7342161  -0.52053554]]
R: [[-1.81071055  0.61215095  0.73102639]
 [-1.13151291 -0.88798315  1.44201181]
 [ 0.32267964  1.82827477 -1.29150658]]
I: [[-0.80255892  0.43014187 -0.13240684]
 [-0.03093471 -0.04662985 -0.50638365]
 [-0.02084761 -0.70087884 -0.349258  ]]
C: [[-2.32365887 -1.25304431  0.99291776]
 [-3.31043735 -0.16814997  2.51114518]
 [ 0.69999702  3.84373632 -2.71806317]]
```

[29]:
```python
#zadanie 2.5
def porownaj_czas_cos():
    x = np.arange(-10, 10.001, 0.001)

    start_loop = time.time()
    y_loop = []
    for xi in x:
        y_loop.append(np.cos(xi))
    end_loop = time.time()
    czas_petla = end_loop - start_loop

    start_vector = time.time()
    y_vector = np.cos(x)
    end_vector = time.time()
    czas_tablica = end_vector - start_vector

    # Wyniki
    print(f"Czas obliczeń w pętli:     {czas_petla:.5f} s")
    print(f"Czas obliczeń tablicowych: {czas_tablica:.5f} s")

porownaj_czas_cos()
```

```
Czas obliczeń w pętli:     0.01800 s
Czas obliczeń tablicowych: 0.00000 s
```

[31]:
```
#zadanie 2.6
def metrs(kmh):
    return kmh*10/36

print(metrs(36))
```

```
10.0
```

[33]:
```
#zadanie 3.1/2
wektor = np.random.rand(50)
wektor.tofile("dane.bin")
dane = np.fromfile("dane.bin", dtype=np.float64)
A = dane[:20].reshape((4, 5))
print(A)
```

```
[[0.92004542 0.65506606 0.39056134 0.60333604 0.00390868]
 [0.44517685 0.88046411 0.86828721 0.13627132 0.17753263]
 [0.6571506  0.74830653 0.89720365 0.02549351 0.30450774]
 [0.95865198 0.62827154 0.84214016 0.52973909 0.82511417]]
```

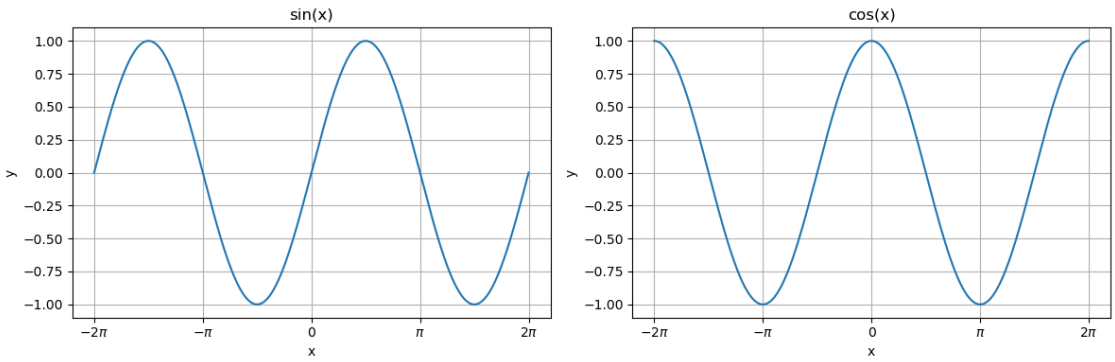
[35]:
```
#zadanie 3.3
def fahr(n):
    return n*9/5+32

v=np.arange(0, 301, 20)
w=fahr(v)

dane = np.concatenate((v, w))
dane.tofile("temperatura.txt")
```

[37]:
```
#zadanie 3.4
def temp_print():
    dane = np.fromfile("temperatura.txt", dtype=np.float64)
    srodek = len(dane) // 2
    v_plik = dane[:srodek]
    w_plik = dane[srodek:]
    print(v_plik)
    print(w_plik)

temp_print()
```

```
[  0.  20.  40.  60.  80. 100. 120. 140. 160. 180. 200. 220. 240. 260.
 280. 300.]
[ 32.  68. 104. 140. 176. 212. 248. 284. 320. 356. 392. 428. 464. 500.
 536. 572.]
```

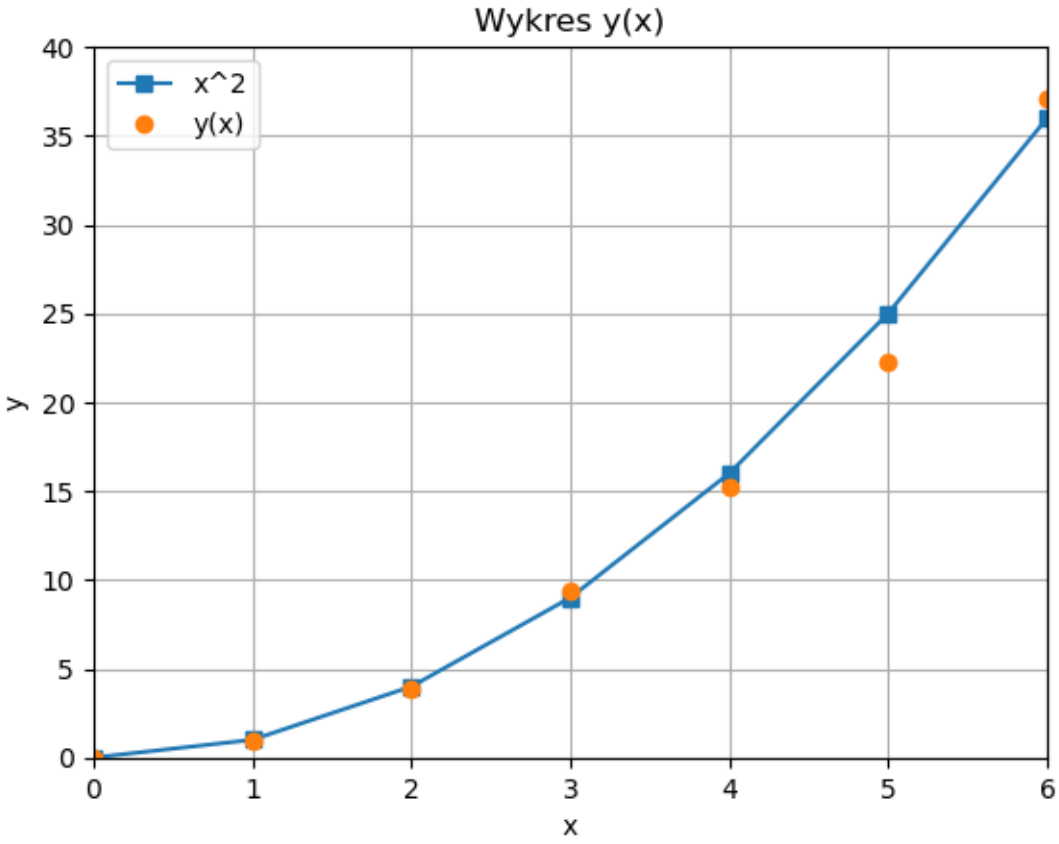
[39]:
```
#zadanie 4.1
x=np.linspace(-2*np.pi,2*np.pi,1000)
y1=np.sin(x)
```

```
y2=np.cos(x)

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(x, y1)
plt.title("sin(x)")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)

xticks = [-2*np.pi, -np.pi, 0, np.pi, 2*np.pi]
xtick_labels = [r'$-2\pi$', r'$-\pi$', '0', r'$\pi$', r'$2\pi$']
plt.xticks(xticks, xtick_labels)

plt.subplot(1, 2, 2)
plt.plot(x, y2)
plt.title("cos(x)")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)

xticks = [-2*np.pi, -np.pi, 0, np.pi, 2*np.pi]
xtick_labels = [r'$-2\pi$', r'$-\pi$', '0', r'$\pi$', r'$2\pi$']
plt.xticks(xticks, xtick_labels)

plt.tight_layout()
plt.show()
```



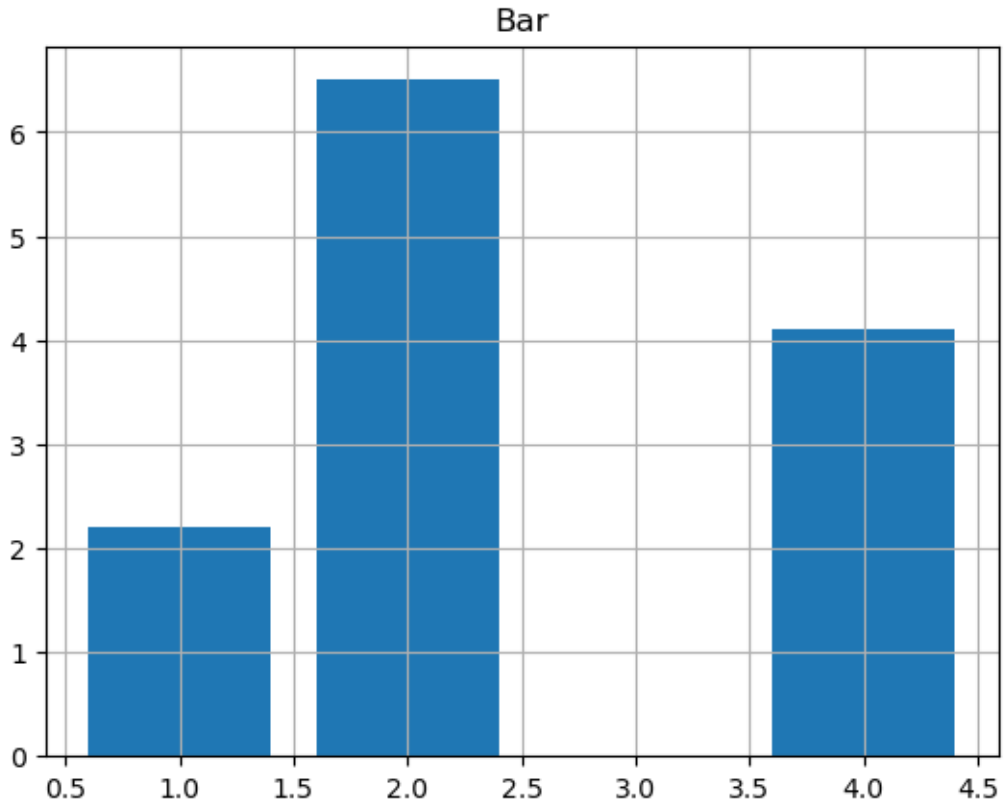
[41]:
```
#zadanie 4.2
x=np.array([0,1,2,3,4,5,6])
y=np.array([0,0.95,3.9,9.4,15.2,22.3,37.1])
b=x**2
```

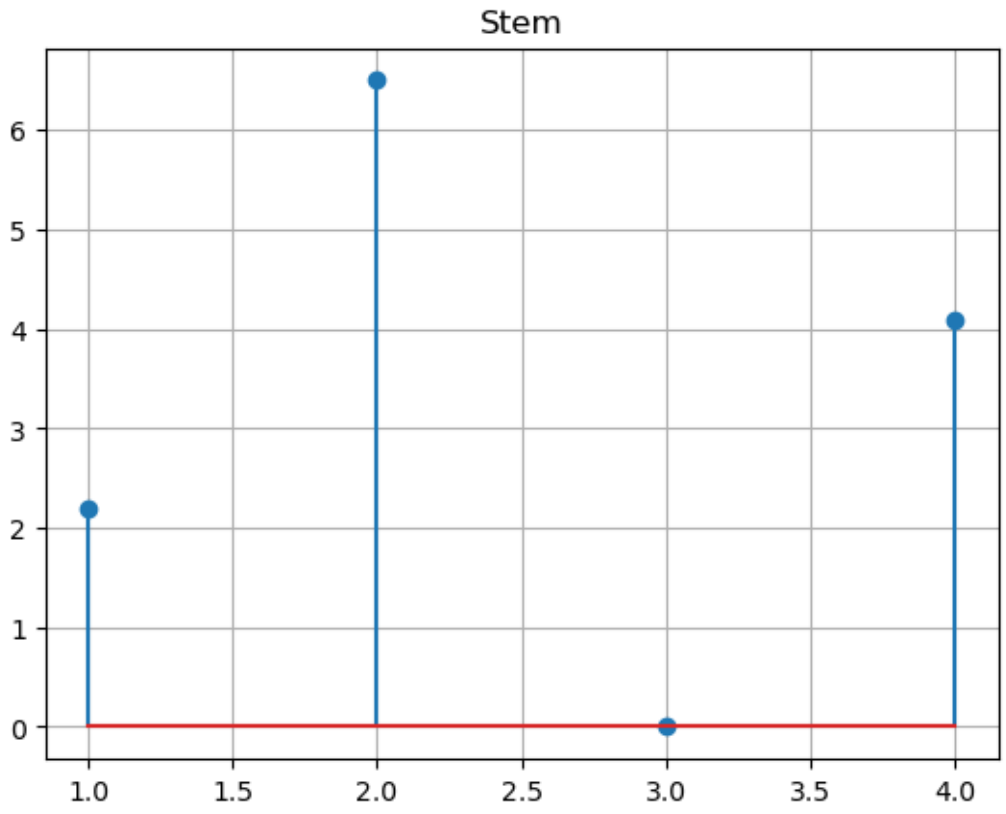
```
plt.plot(x, b, 's-', label='x^2')
plt.plot(x, y, 'o', label='y(x)')
plt.xlim(0,6)
plt.ylim(0,40)
plt.title('Wykres y(x)')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True)
plt.legend()
plt.show()
```

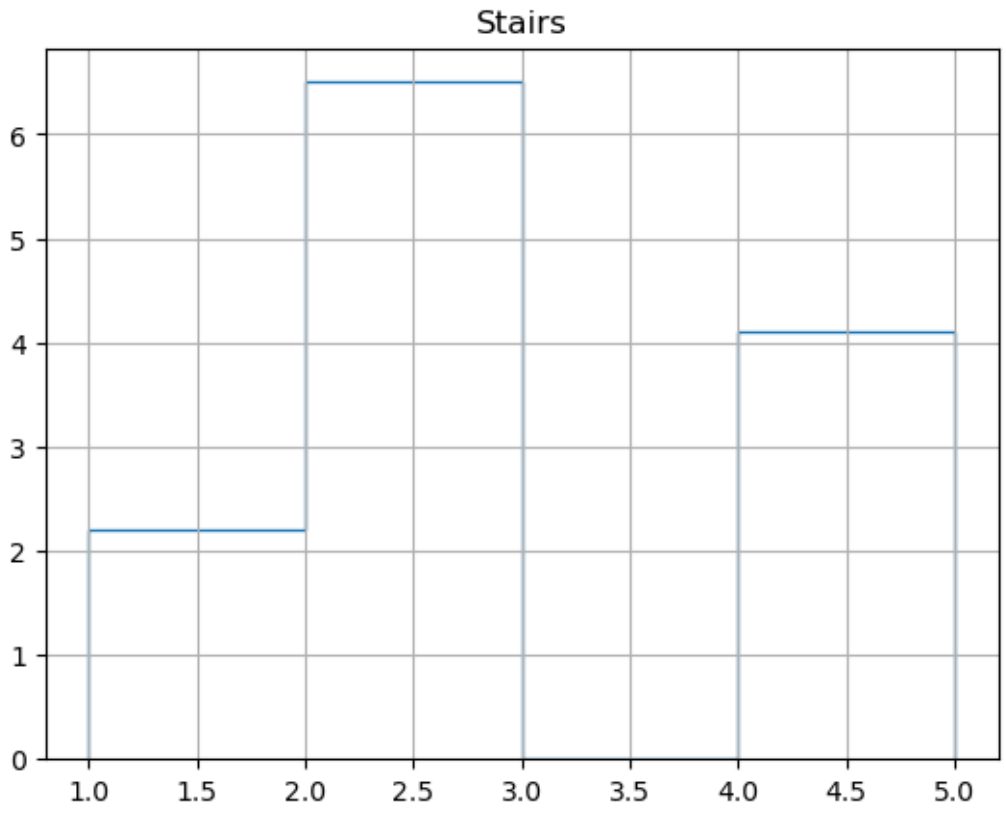


[43]:
```
#zadanie 4.3
x = [1, 2, 3, 4]
y = [2.2, 6.5, 0, 4.1]

plt.figure()
plt.bar(x, y)
plt.title('Bar')
plt.grid(True)
```

```
plt.figure()
plt.stem(x, y)
plt.title('Stem')
plt.grid(True)

x_stairs = x + [x[-1] + 1]
plt.figure()
plt.stairs(y, x_stairs)
plt.title('Stairs')
plt.grid(True)

plt.show()
```

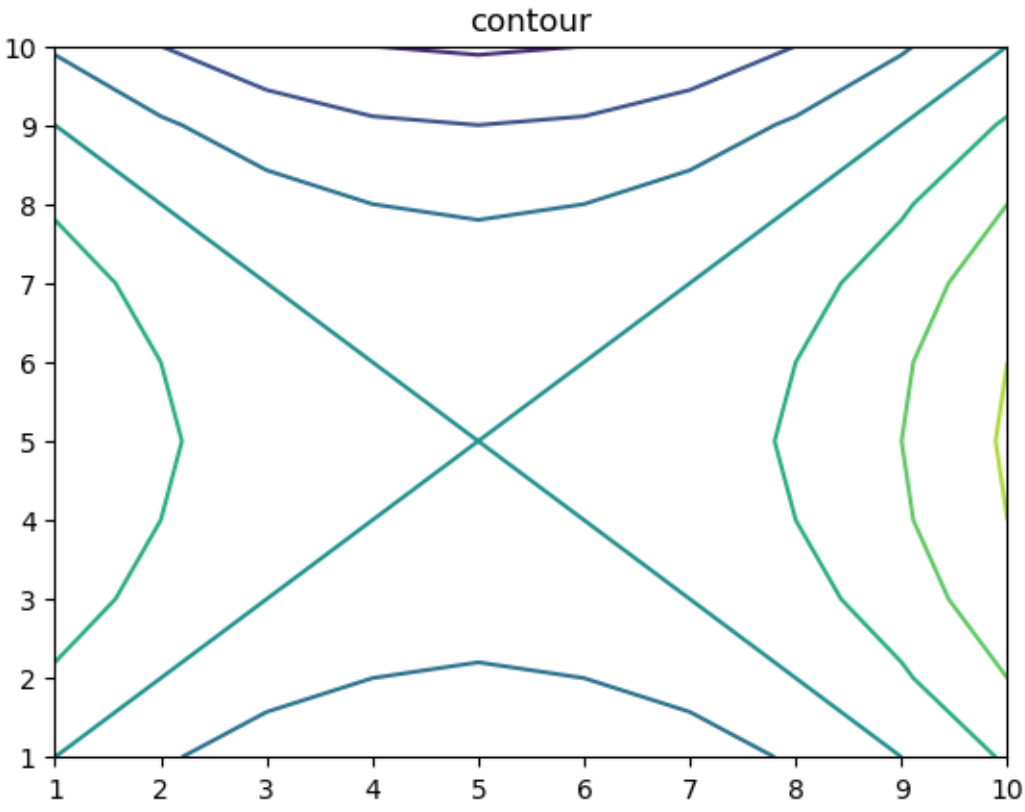


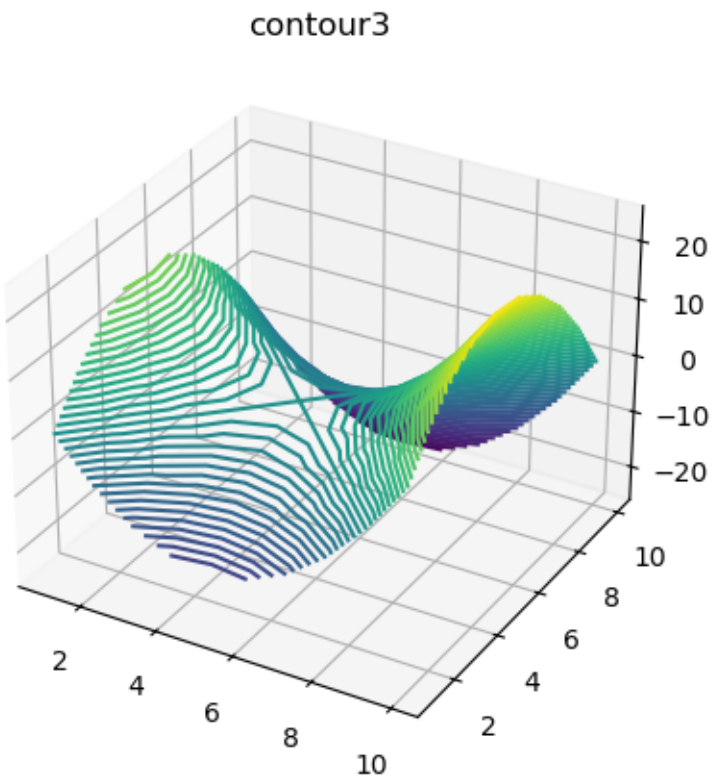

Stem
6
5
4
3
2
1
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0

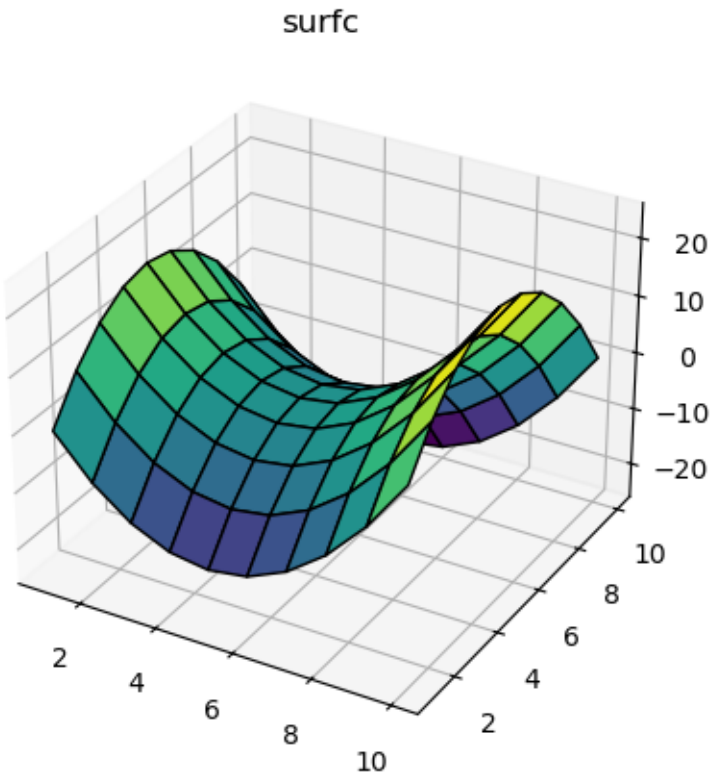


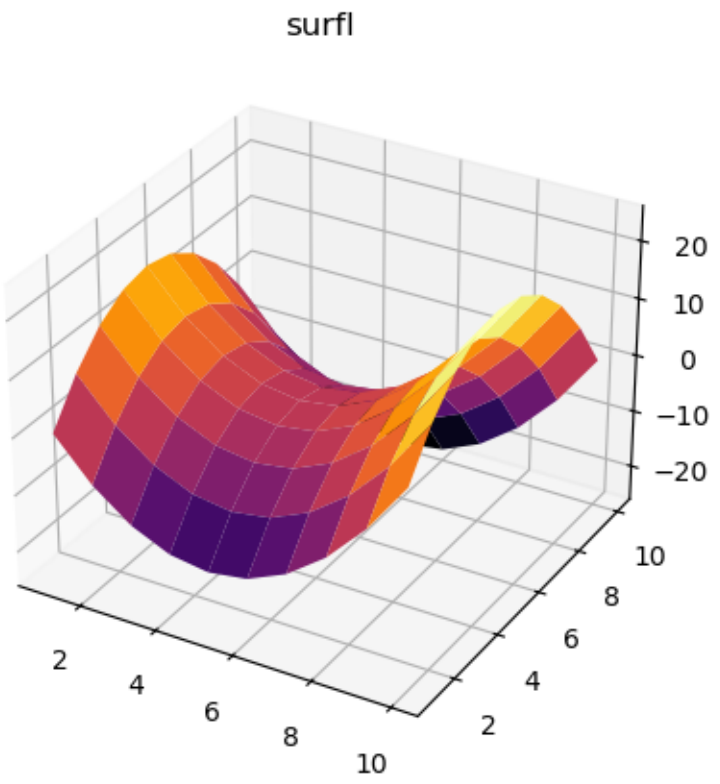
[45]:
```
#zadanie 4.4
x = np.arange(1, 11)
y = np.arange(1, 11)
X, Y = np.meshgrid(x, y)
Z = (X - 5)**2 - (Y - 5)**2

plt.figure()
plt.contour(X, Y, Z)
plt.title('contour')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.contour3D(X, Y, Z, 50)
ax.set_title('contour3')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='k')
ax.set_title('surfc')

fig = plt.figure()
```

```
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, Z, cmap='inferno', lightsource=None)
ax.set_title('surfl')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)
ax.set_title('meshz')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X, Y, Z)
ax.contour(X, Y, Z, zdir='z', offset=np.min(Z)-10)
ax.set_title('meshc')

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
for i in range(len(y)):
    ax.plot(x, Z[i, :], y[i])
ax.set_title('waterfall')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

plt.show()
```

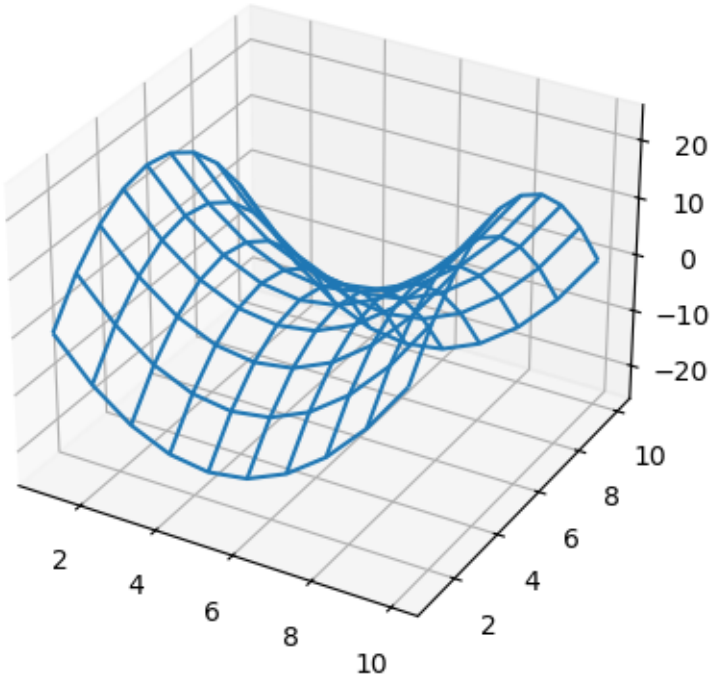

contour
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
7
8
9
10


contour3
20
10
0
−10
−20
2
4
6
8
10
2
4
6
8
10

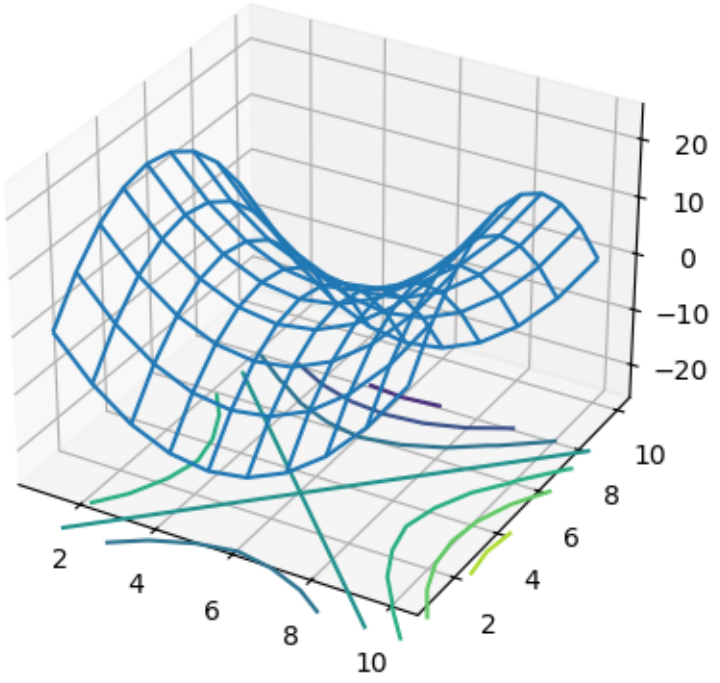

surfc
20
10
0
−10
−20
2
4
6
8
10
2
4
6
8
10

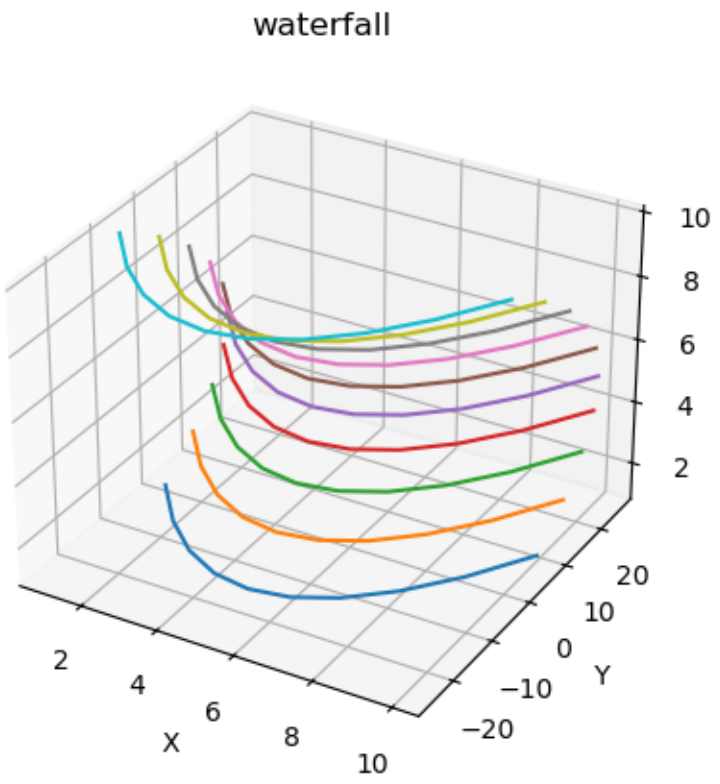

surfl
20
10
0
−10
−20
2
4
6
8
10
2
4
6
8
10

meshz

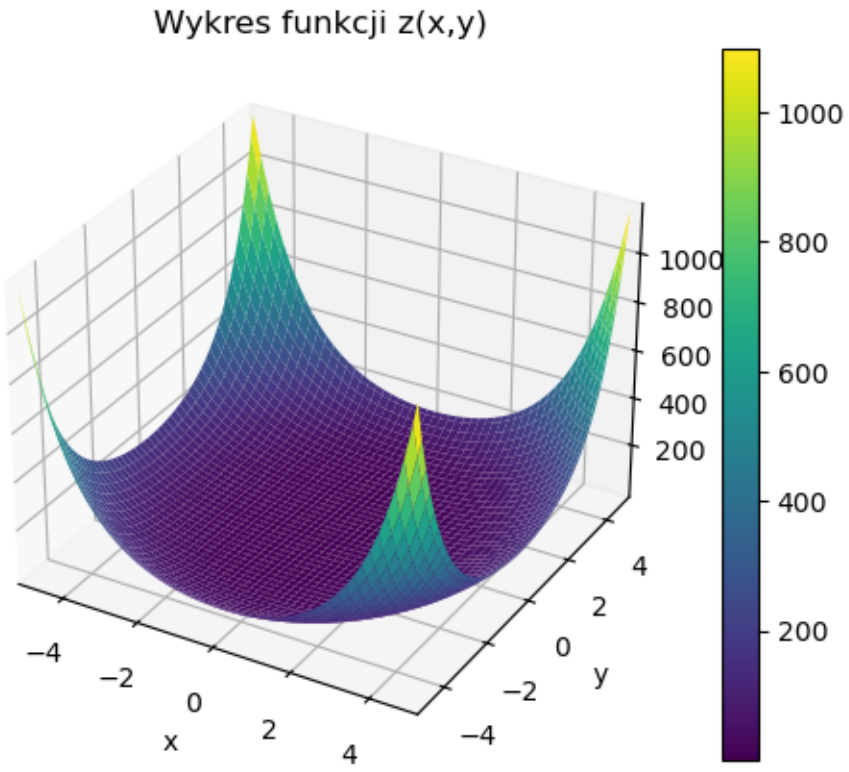


meshc



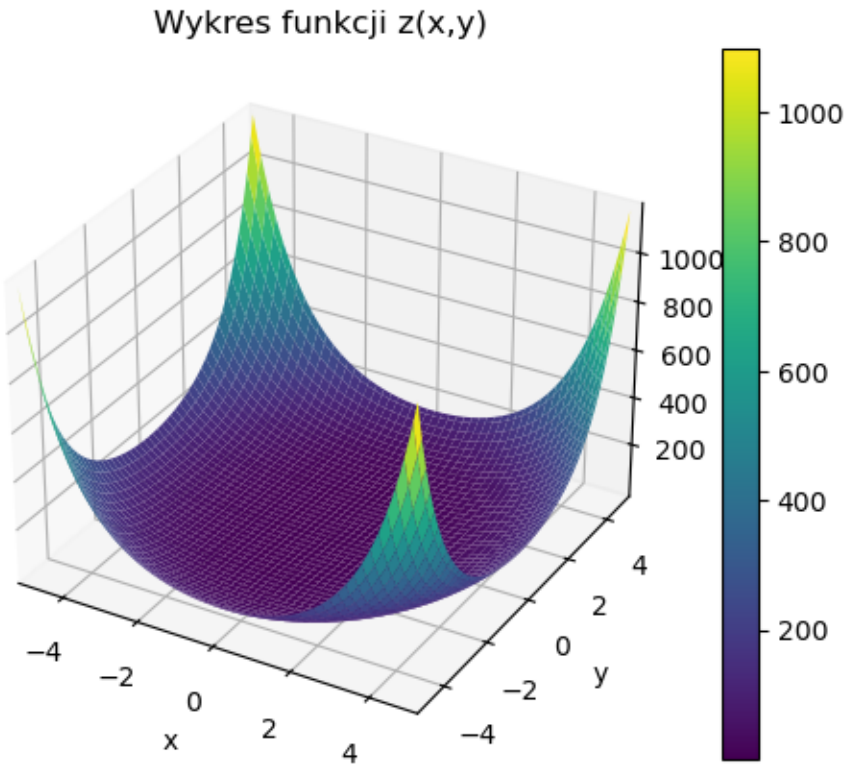


[47]:
```
#zadanie 4.5
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = (X - Y) * (X + Y) + np.exp(np.sqrt(X**2 + Y**2))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis')
fig.colorbar(surf)

ax.set_xlim(-5, 5)
ax.set_ylim(-5, 5)
ax.grid(True)
ax.set_title('Wykres funkcji z(x,y)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

plt.show()
```



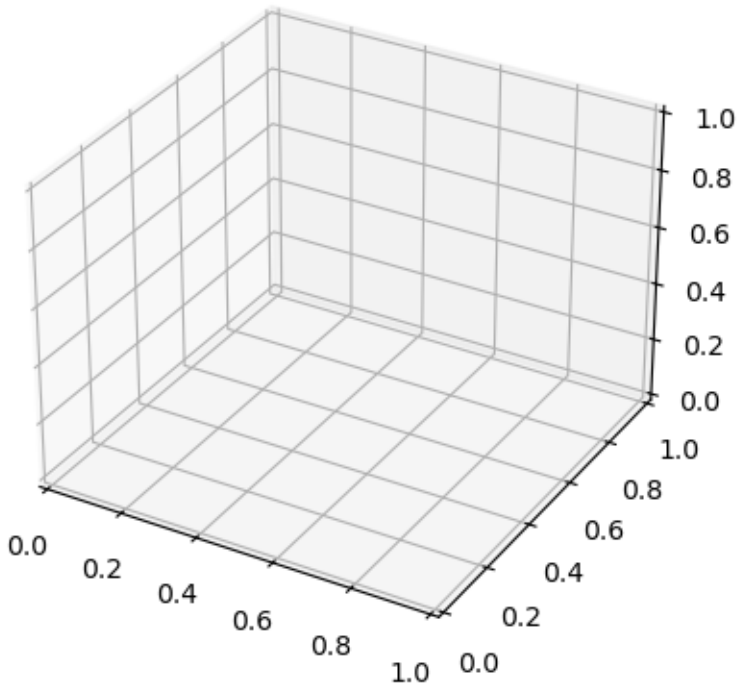
[49]:
```
#zadanie 4.6
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, y)
Z = (X - Y) * (X + Y) + np.exp(np.sqrt(X**2 + Y**2))
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis')
fig.colorbar(surf)

ax.set_xlim(-5, 5)
ax.set_ylim(-5, 5)

ax.grid(True, linestyle=':')
ax.set_xticks([-4, -2, 0, 2, 4])
ax.set_yticks([-4, -2, 0, 2, 4])

ax.set_title('Wykres funkcji z(x,y)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
```

```
plt.show()
```



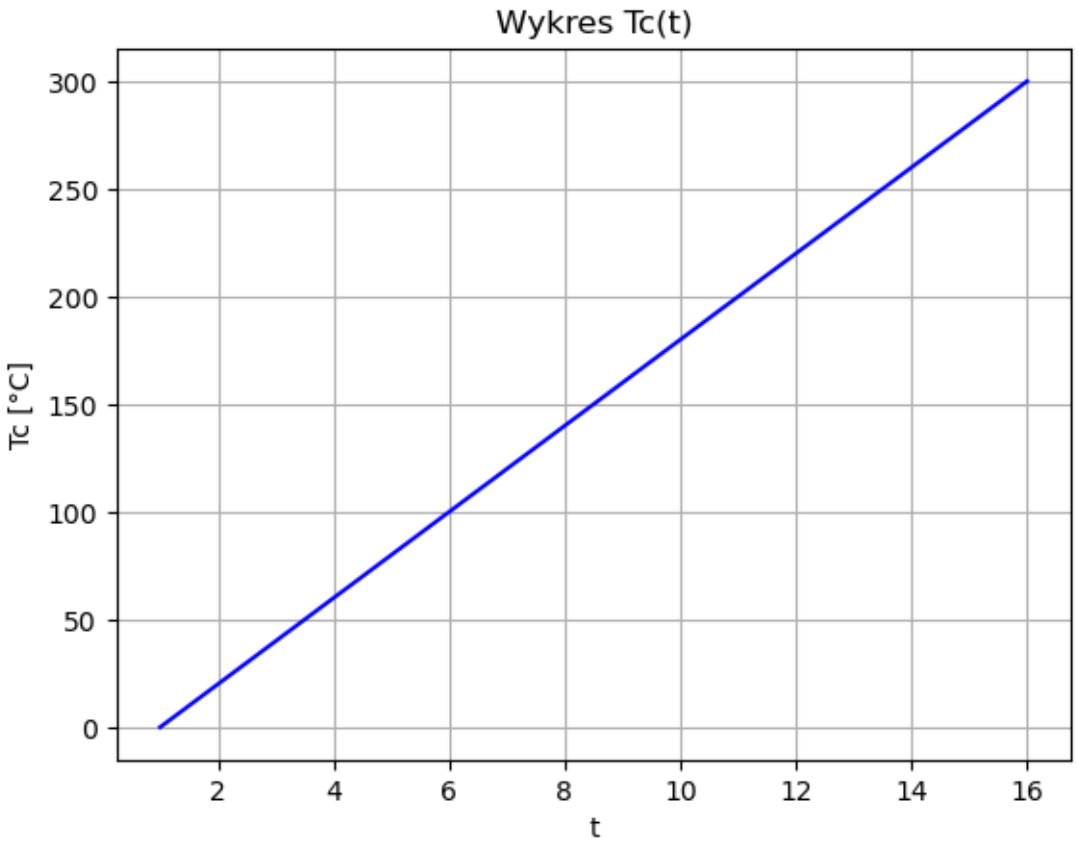
[60]:
```
#zadanie 4.7
def rysuj_wykres(ax):
    x = np.linspace(-5, 5, 100)
    y = np.linspace(-5, 5, 100)
    X, Y = np.meshgrid(x, y)
    Z = (X - Y) * (X + Y) + np.exp(np.sqrt(X**2 + Y**2))
    ax.plot_surface(X, Y, Z, cmap='viridis')
    ax.grid(True, linestyle=':')
    ax.set_xticks([-4, -2, 0, 2, 4])
    ax.set_yticks([-4, -2, 0, 2, 4])

def czysc():
    ax.clear()
    canvas.draw()

root = tk.Tk()
root.title("Czyść")

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

```
rysuj_wykres(ax)

canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)
canvas.draw()

btn = tk.Button(root, text="Czyść", command=czysc)
btn.pack(side=tk.BOTTOM)

root.mainloop()
```



[64]:
```
#zadanie 4.8
dane = np.fromfile("temperatura.txt", dtype=np.float64)
Tc = dane[:len(dane)//2]
Tk = dane[len(dane)//2:]
t = np.arange(1, len(Tc)+1)

def rysuj_tc():
    ax.clear()
    ax.plot(t, Tc, 'b-', label='Temperatura °C')
    ax.set_title('Wykres Tc(t)')
    ax.set_xlabel('t')
```

```
    ax.set_ylabel('Tc [°C]')
    ax.grid(True)
    canvas.draw()

def rysuj_tk():
    ax.clear()
    ax.plot(t, Tk, 'r-', label='Temperatura °F')
    ax.set_title('Wykres Tk(t)')
    ax.set_xlabel('t')
    ax.set_ylabel('Tk [°F]')
    ax.grid(True)
    canvas.draw()

root = tk.Tk()
root.title("Wykresy Temperatury")

fig, ax = plt.subplots()
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)
canvas.draw()

menubar = Menu(root)
menu_przebiegi = Menu(menubar, tearoff=0)
menu_przebiegi.add_command(label="Tc(t)", command=rysuj_tc)
menu_przebiegi.add_command(label="Tk(t)", command=rysuj_tk)
menubar.add_cascade(label="Przebiegi", menu=menu_przebiegi, underline=0)
root.config(menu=menubar)

rysuj_tc()

root.mainloop()
```



[76]:
```
#zadanie 5.1
def f(x):
    return np.sin(x)
def f_p(x):
    return np.cos(x)

x0=3
m0=newton(f,x0,fprime=f_p)
print(m0)
```

```
3.141592653589793
```

[80]:
```
#zadanie 5.2
print("a)")
wsp=[1,3,-4]
roots=np.roots(wsp)
print(roots)
print("b)")
wsp=[-2,0,3,-1,-6]
roots=np.roots(wsp)
```

```
print(roots)
```

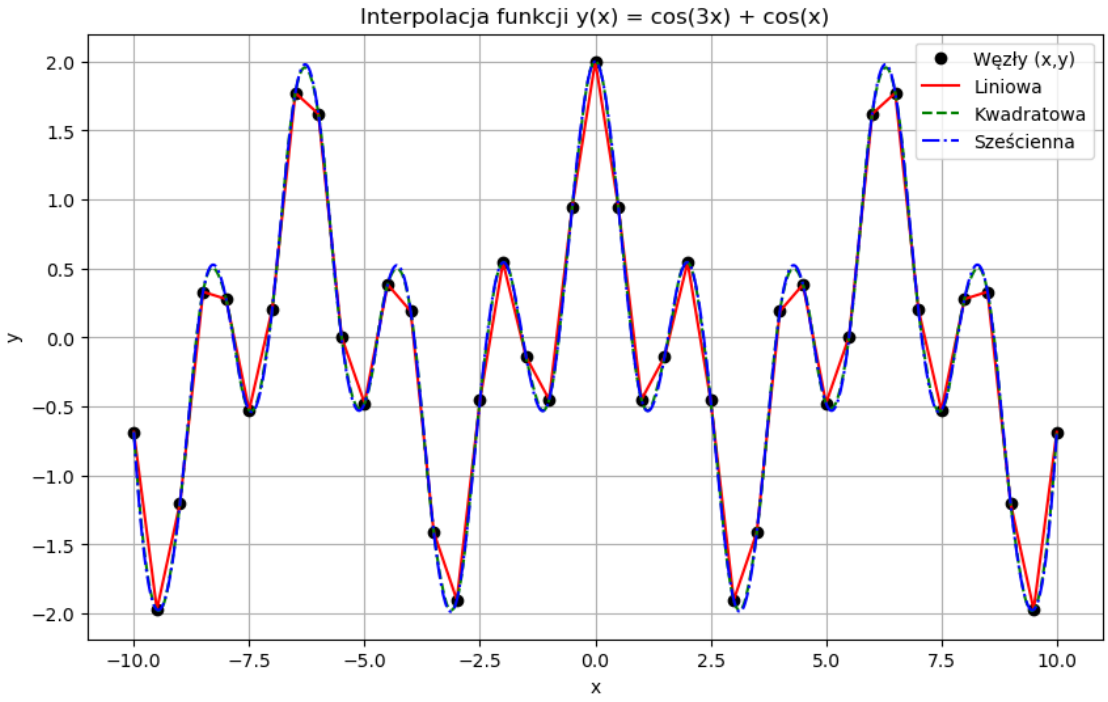
```
a)
[-4.  1.]
b)
[ 1.1148265+0.7777938j   1.1148265-0.7777938j  -1.1148265+0.61701949j
 -1.1148265-0.61701949j]
```

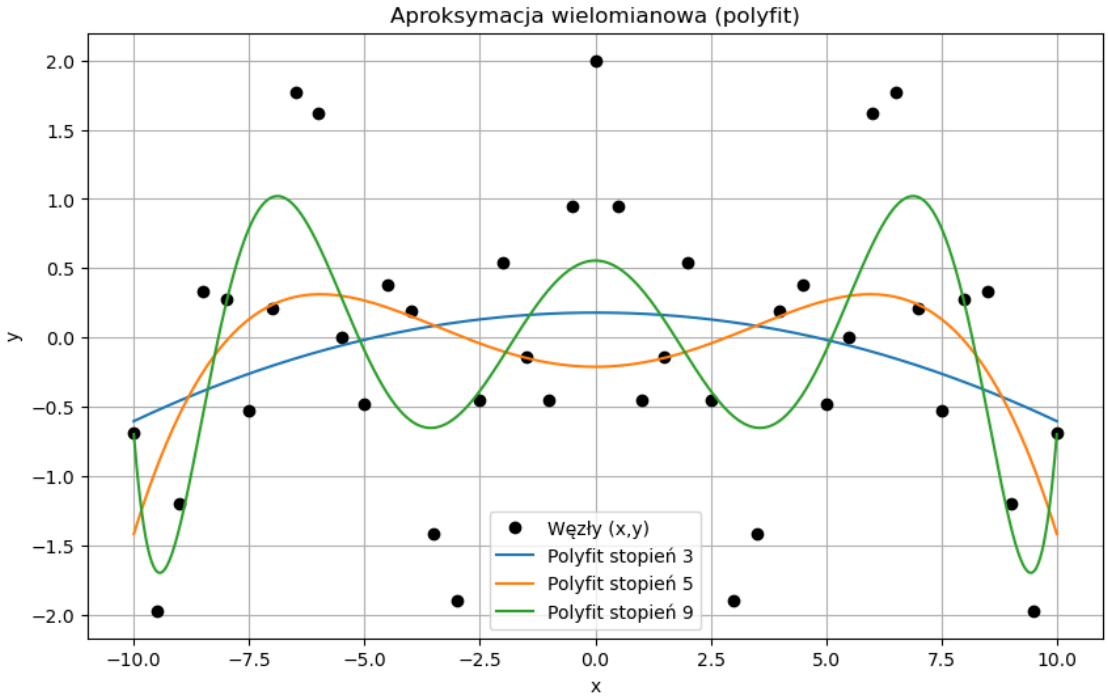
[88]:
```
#zadanie 5.3
A=np.array([[1,2,3,4],[3,4,5,6],[2,3,5,0],[3,5,2,1]])
b=np.array([1,2,4,6])
x=np.linalg.solve(A,b)
print("a =",x[0])
print("b =",x[1])
print("c =",x[2])
print("d =",x[3])
```

```
a = -0.8939393939393941
b = 1.8030303030303032
c = 0.07575757575757582
d = -0.48484848484848486
```

[92]:
```
#zadanie 5.5
x = np.arange(-10, 10.5, 0.5)
y = np.cos(3 * x) + np.cos(x)

x_interp = np.linspace(-10, 10, 1000)

linear_interp = interp1d(x, y, kind='linear')
quad_interp = interp1d(x, y, kind='quadratic')
cubic_interp = interp1d(x, y, kind='cubic')

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label='Węzły (x,y)')
plt.plot(x_interp, linear_interp(x_interp), 'r-', label='Liniowa')
plt.plot(x_interp, quad_interp(x_interp), 'g--', label='Kwadratowa')
plt.plot(x_interp, cubic_interp(x_interp), 'b-.', label='Sześcienna')

plt.title("Interpolacja funkcji y(x) = cos(3x) + cos(x)")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()
plt.show()

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label='Węzły (x,y)')
```

```
for degree in [3, 5, 9]:
    coeffs = np.polyfit(x, y, degree)
    y_poly = np.polyval(coeffs, x_interp)
    plt.plot(x_interp, y_poly, label=f'Polyfit stopień {degree}')

plt.title("Aproksymacja wielomianowa (polyfit)")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
plt.legend()
plt.show()
```



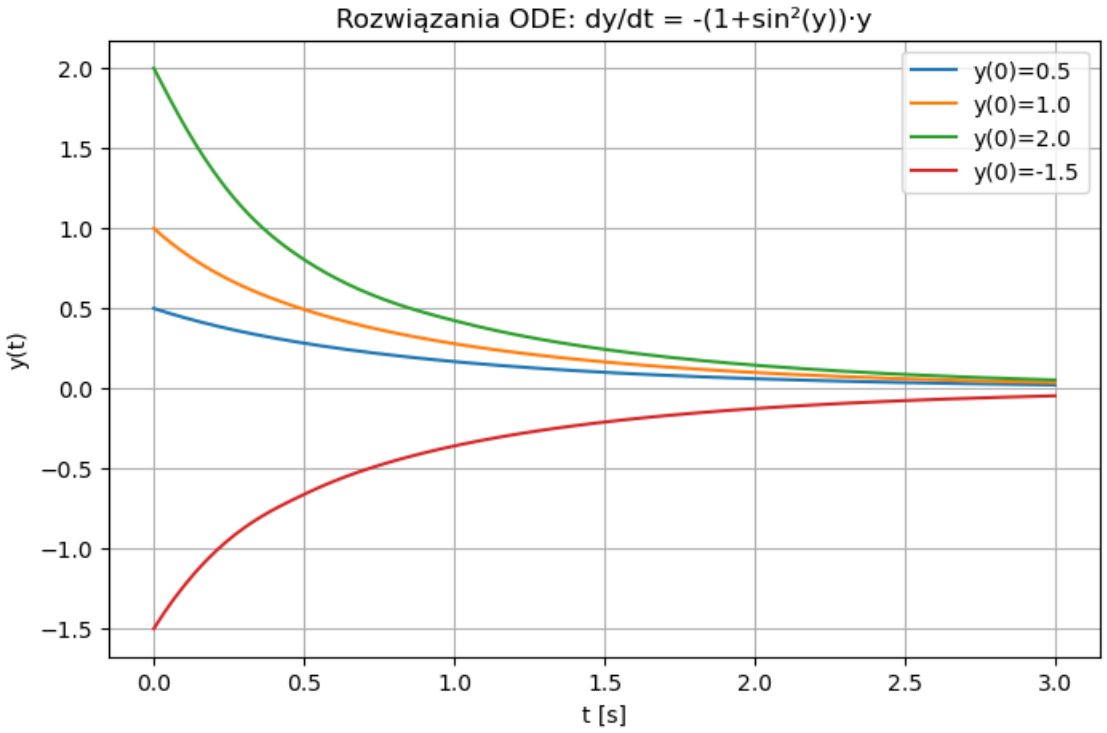


[100]:
```
#zadanie 5.6
x=sp.symbols('x')
f=(x-1)**2
C=sp.integrate(f,(x,-2,2))
g=x+2*sp.sin(x)
D=sp.integrate(g,(x,-sp.pi,sp.pi))
print("a)",C)
print("b)",D)
```

a) 28/3
b) 0

[116]:
```
#zadanie 5.7
def f(t, y):
    return -(1 + (np.sin(y))**2) * y

t_span = (0, 3)
t_eval = np.linspace(*t_span, 300)
init = [0.5, 1.0, 2.0, -1.5]

plt.figure(figsize=(8, 5))

for y0 in init:
    s = solve_ivp(f, t_span, [y0], t_eval=t_eval)
```

```
    plt.plot(s.t, s.y[0], label=f'y(0)={y0}')

plt.xlabel('t [s]')
plt.ylabel('y(t)')
plt.title('Rozwiązania ODE: dy/dt = -(1+sin²(y))·y')
plt.legend()
plt.grid(True)
plt.show()
```



[ ]: