[ ]:
```
import jax
import numpy

from numpy.linalg import inv
from jax import numpy as jnp
from jax import grad
```

1 Prolem №1

You will work with the following function for exercise, $f(x, y) = e^{-(sin(x)+cos(y))^2}$

Draw the computational graph for the function. Note, that it should contain only primitive operations - you need to do it automatically.

[ ]:
```
#Function of first problem
def func_p1(x, y):
  return jnp.exp(- jnp.power((jnp.sin(x[0]) + jnp.cos(y[0])), 2))


def dfunc_p1(x, y):
  return grad(func_p1, argnums=(0, 1))(x, y)
```

[ ]:
```
z=jax.xla_computation(dfunc_p1)(numpy.random.rand(1), numpy.random.rand(1))

with open("t1.txt", "w") as f:
  f.write(z.as_hlo_text())

with open("t1.dot", "w") as f:
  f.write(z.as_hlo_dot_graph())
```

2 Problem №2

Compare analytic and autograd approach for the hessian of: $f(x) = \frac{1}{2}x^T Ax + b^T x + c$

[ ]:
```
from jax import jacfwd, jacrev
```

[ ]:
```
A = numpy.random.rand(100, 100)
b = numpy.random.rand(100)
c = 1

def func_p2(x):
 return 0.5 * x.T @ A @ x + b @ x + c

def hessian(f):
  return jax.jacfwd(jax.grad(f))
```

```
def d2func_p2(x):
  return hessian(func_p2)(x)

hessian_auto2 = d2func_p2(numpy.random.rand(100))
hessian_anal2 = (A + A.T) / 2
```

Difference between autograde and analytical solution:

[ ]:
```
numpy.linalg.norm(hessian_anal2 - hessian_auto2)
```

[ ]:
```
2.1407686e-06
```

Cringe moment for visualising it

[ ]:
```
z = jax.xla_computation(d2func_p2)(numpy.random.rand(100))

with open("t2.txt", "w") as f:
   f.write(z.as_hlo_text())

with open("t2.dot", "w") as f:
  f.write(z.as_hlo_dot_graph())
```

3 Problem №3

Suppose we have the following function $f(x) = \frac{1}{2}||x||^2$, select a random point $x_0 \in \mathbb{B}^{1000} = \{0x_i1|i\}$. Consider 10 steps of the gradient descent starting from the point x_0:

$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$.

Your goal in this problem to write the function, that takes 10 scalar values α_i and return the result of the gradient descent on function $L = f(x_{10})$. And optimize this function using gradient descent on $\alpha \in \mathbb{R}^{10}$. Suppose, $\alpha_0 = 1$.

$\alpha_{k+1} = \alpha_k - \beta \cdot \frac{\partial L}{\partial \alpha}$

Choose any β and the number of steps your need. Describe obtained results.

[ ]:
```
def func_p3(x):
  return 0.5 * x.T @ x

def dfunc_p3(x):
  return grad(func_p3)(x)
```

[ ]:
```
# Do it later...
def gradient(x0, alpha0, num_steps=10):
  x = x0
  alpha = alpha0
  for i in range(0, num_steps):
    x = x - alpha * dfunc_p3(x)
```

4 Problem №4

Compare analytic and autograd approach for the gradient of: $f(X) = -log(det(X))$

Analytical gradient: $df = -\frac{1}{det(X)} \cdot det(X)\langle X^{-T}, dX\rangle$

$df = -\langle X^{-T}, dX\rangle$

$\nabla f = -X^{-T}$

[ ]:
```
X = numpy.random.rand(100, 100)

func_p4 = lambda X: -jnp.log(jnp.linalg.det(X))
dfunc_p4 = lambda X: grad(func_p4)(X)

grad_auto4 = dfunc_p4(X)
grad_anal4 = -(inv(X)).T

print("Difference between analytical and autograde methods:", numpy.linalg.
 ↪norm(grad_auto4 - grad_anal4))
```

```
Difference between analytical and autograde methods: 0.00039553354
```

5 Problem №5

Compare analytic and autograd approach for the gradient and hessian of: $f(x) = x^T x x^T x$

[ ]:
```
def func_p5(x):
  return jnp.dot(x.T, x)* jnp.dot(x.T, x)

def dfunc_p5(x):
  return grad(func_p5)(x)

def d2func_p5(x):
  return hessian(func_p5)(x)
```

Analytical gradient: \$df = 4 \langle x, x\rangle \cdot \langle x, dx\rangle\$

$\nabla f = 4\langle x, x\rangle \cdot x$

Analytical hessian: $d^2 f = 4 \cdot \left(\langle dx_2, x\rangle\langle x, dx_1\rangle + \langle x, dx_2\rangle\langle x, dx_1\rangle + \langle x, x\rangle\langle dx_2, dx_1\rangle\right)$

$d^2 f = 4 \cdot x^T\left(3x \cdot dx_2^T\right)dx_1 = 12x^T \cdot x dx_2^T \cdot dx_1$

$hessian(f) = 12x \cdot x^T$ - it will be matrix...

[ ]:
```
x = numpy.random.rand(2)
#x = numpy.ones(2)
grad_auto5 = grad(func_p5)(x)
grad_anal5 = 4 * jnp.dot(x, x) * x
```

```
print("Difference between analytic and auto gradient",numpy.linalg.
 ↪norm(grad_auto5 - grad_anal5))
```

```
Difference between analytic and auto gradient 0.0
```

[ ]:
```
hessian_auto5 = d2func_p5(x)
hessian_anal5 = 12 * jnp.outer(x, x.T)

print("Difference between analytic and auto hessian:",numpy.linalg.
 ↪norm(hessian_auto5 - hessian_anal5))

#print("Hessian auto: ", hessian_auto5, hessian_auto5.shape)
#print("Hessian anal: ", hessian_anal5, hessian_anal5.shape)
```

```
Difference between analytic and auto hessian: 5.4479795
```