Численные методы решения систем нелинейных уравнений

Дз №7

Варинт 5 Доскоч Роман 3 курс 13 группа

In [1]:
```
import numpy as np
from numpy import array
import matplotlib.pyplot as plt
from matplotlib.pyplot import plot
from numpy import meshgrid
import random
import math
```

$$\begin{cases} 6(x-5)^2 + 9y - 5 = 0 \\ (x-10)^2 + (y+4)^2 - 21^2 = 0 \end{cases}$$

Найдем центр параболы

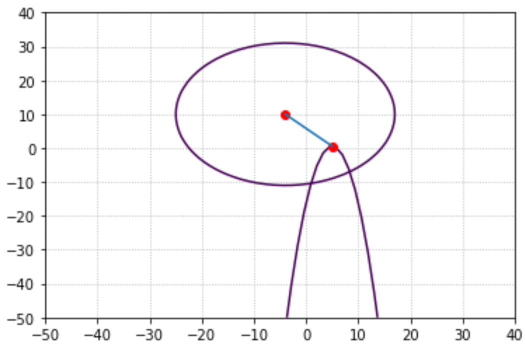
$$dy = (-(6(x-5)^2 - 5)/9)' = 0$$
$$-12(x-5)/9 = 0$$
$$x = 5 \; y = \frac{5}{9}$$

так как центр оружности находиться в точке $O(-4, 10)$, а центр паработы в точке $O'(5, \frac{5}{9})$растояниемеждуточками$\sqrt{(5+4)^2 + (5/9-10)^2} = 13.0459775 < 21$

Значит, корней всего 2 и будут они в близи вершины параболы.

In [2]:
```
f1 = lambda x,y: 6*(x - 5)**2 + 9*y - 5
f2 = lambda x,y: (x - 10)**2 + (y + 4)**2 - 21**2
```

In [3]:
```
x, y = meshgrid(np.linspace(-20,40,50), np.linspace(-50,30,50))
plt.contour(x,y,f1(x,y),[0])
plt.contour(y,x,f2(x,y),[0])
plot(-4, 10, 'ro', 5, 5/9, 'ro', [-4,5], [10,5/9])
plt.grid(ls=':')
```



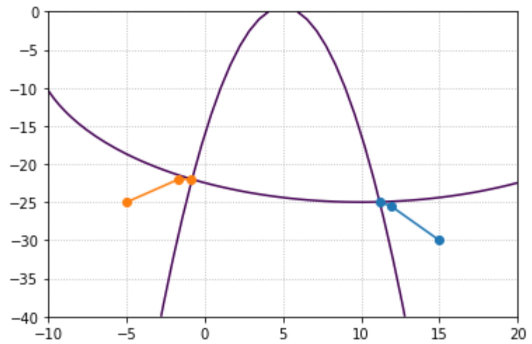
In [4]:
```
#метод крамера для решения 2-матрицы
def Cramer(X,f):
    a1, a2, a3, a4 = X[0][0], X[0][1], X[1][0], X[1][1]
    b1, b2 = f[0], f[1]
    y = (a3*b1-a1*b2)/(a3*a2-a1*a4)
    x = (b2 - a4*y)/a3
    return array([x,y])
```

Метод ньютона

In [5]:
```
def solver(x_, y_, iterations=3):
    x_curr = array([x_,y_])
    path_X, path_Y = [], []
    for i in range(iterations):
        x,y = x_curr[0], x_curr[1]
        J = [[12*(x-5),       9],
             [2*(x-10), 2*(y+4)]]
        x_curr += Cramer(J,[-f1(x,y),-f2(x,y)])

        path_X.append(x)
        path_Y.append(y)

    return x_curr, path_X, path_Y
```

In [6]:
```
x,y = meshgrid(np.linspace(-10, 20, 50), np.linspace(-40, 0, 50))
plt.contour(x,y,f1(x,y),[0])
plt.contour(x,y,f2(x,y),[0])
plt.grid(ls=':')

r1, path_X, path_Y = solver(15.0,-30.0)
plot(path_X,path_Y,'-o')
r2, path_X, path_Y = solver(-5.0,-25.0)
plot(path_X,path_Y,'-o')

print(f'(x1,y1)={[r1[0],r1[1]]}')
print(f'(x2,y2)={[r2[0],r2[1]]}')
print(f'f1(x1,y1) = {f1(r1[0],r1[1])} ≈ 0')
print(f'f2(x2,y2) = {f2(r2[0],r2[1])} ≈ 0')
```

```
(x1,y1)=[11.18748904356222, -24.966454635154953]
(x2,y2)=[-0.8168988581780426, -21.999943891219807]
f1(x1,y1) = 0.01203226882051922 ≈ 0
f2(x2,y2) = 0.00328099511466462 ≈ 0
```

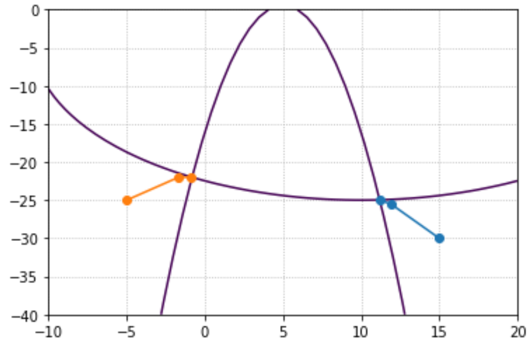


Дискретный метод Ньютона

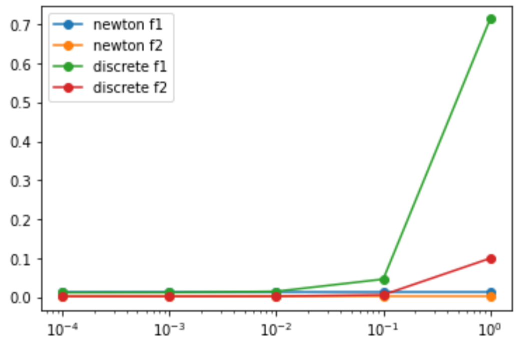
In [7]:
```
def discrete_solver(x_, y_, h=.001, iterations=3):
    x_curr = array([x_,y_])
    path_X, path_Y = [], []
    for i in range(iterations):
        x,y = x_curr[0], x_curr[1]
        J = [[(f1(x+h,y) - f1(x,y))/h, (f1(x,y+h)-f1(x,y))/h],
             [(f2(x+h,y) - f2(x,y))/h, (f2(x,y+h)-f2(x,y))/h]]
        x_curr += Cramer(J,[-f1(x,y),-f2(x,y)])

        path_X.append(x)
        path_Y.append(y)

    return x_curr, path_X, path_Y
```

In [8]:
```
x,y = meshgrid(np.linspace(-10,20, 50), np.linspace(-40,0, 50))
plt.contour(x,y,f1(x,y),[0])
plt.contour(x,y,f2(x,y),[0])
plt.grid(ls=':')

dr1, path_X, path_Y = discrete_solver(15.0,-30.0)
plot(path_X,path_Y,'-o')
dr2, path_X, path_Y, = discrete_solver(-5.0,-25.0)
plot(path_X,path_Y,'-o')

print(f'(x1,y1)={[dr1[0],dr1[1]]}')
print(f'(x2,y2)={[dr2[0],dr2[1]]}')
print(f'f1(x1,y1) = {f1(dr1[0],dr1[1])} ≈ 0')
print(f'f2(x2,y2) = {f2(dr2[0],dr2[1])} ≈ 0')
```

```
(x1,y1)=[11.187493192480334, -24.966455162664637]
(x2,y2)=[-0.8168936297634727, -21.999945663730106]
f1(x1,y1) = 0.0123355779611245 ≈ 0
f2(x2,y2) = 0.0032316948538095858 ≈ 0
```



In [9]:
```
start_point = (15.0,-30.0)
H = [ .0001, .001, .01, .1, 1]

discrete = [[f1(*solver(*start_point)[0]),
             f2(*solver(*start_point)[0]),
             f1(*discrete_solver(*start_point, h)[0]),
             f2(*discrete_solver(*start_point, h)[0])] for h in H]

pl = plot(H, discrete, '-o')
plt.legend(iter(pl), ('newton f1','newton f2','discrete f1','discrete f2'));
plt.xscale('log',base=10)
```



Вывод: точность дискретного метода зависит от h (дельты), хотя если выбрать достаточно маленькую h то ответ будет не хуже настоящего.