Untitled

March 6, 2024

1 Tarea 5

Jairo Saul Diaz Soto

Dr. Joaquin Peña Acevedo

Optimizacion I

2024 / 03 / 06

2 Ejercicio 1

2.1 Punto 1:

Encuentre y clasifique los puntos estacionarios para la funcion:

$$f(\mathbf{x}) = x_1^2 - x_2^2 + x_3^2 - 2x_1x_3 - x_2x_3 + 4x_1 + 12$$

Para determinar los puntos estacionarios primero determinamos las derivadas parciales tal que:

$$\frac{\partial f(\mathbf{x})}{\partial x_1} = 2x_1 - 2x_3 + 4$$
$$\frac{\partial f(\mathbf{x})}{\partial x_2} = -2x_2 - x_3$$
$$\frac{\partial f(\mathbf{x})}{\partial x_3} = 2x_3 - 2x_1 - x_2$$

Igualamos cada una de las parciales a 0 y resolvemos el sistema de ecuaciones siguiente

$$2x_1 - 2x_3 + 4 = 0$$

$$2x_2 = -x_3$$
$$2x_3 - 2x_1 - x_2 = 0$$

Resolviendo se obtiene lo siguiente

$$\mathbf{x} = [-10, 4, -8]$$

2.2 Punto 2:

Sea $\mathbf{x}_0 = (1, 0, 0)^T$. Calcule el punto $\mathbf{x}_1$ usando la direccion de descenso maximo con paso exacto.

Obtenemos la direccion del maximo descenso el cual es el negativo del gradiente tal que

$$\nabla f(\mathbf{x}) = [2x_1 - 2x_3 + 4, -2x_2 - x_3, -2x_1 - x_2 + 2x_3]$$

De la forma en que si expresamos la funcion original de la siguiente forma

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + \mathbf{c}$$

Y su gradiente

$$\nabla f(\mathbf{x}) = A\mathbf{x} - b$$

Se puede determinar que

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

$$\mathbf{b} = [-4, 0, 0]$$

Por tal motivo el tamanio de paso exacto que se debe dar es el siguiente:

$$\alpha = \frac{\mathbf{d}^T\mathbf{d}}{\mathbf{d}^T A\mathbf{d}}$$

Donde $\mathbf{d}$ es la direccion de descenso, en este caso entonces $\mathbf{d} = -\nabla f(\mathbf{x}_0)$ a lo cual el tamanio de paso seria

$$\mathbf{d} = [-6, 0, 2]$$

$$\mathbf{d}^T\mathbf{d} = 40$$

$$\mathbf{d}^T A\mathbf{d} = 128$$

$$\alpha = \frac{5}{16}$$

Por lo tanto

$$\mathbf{x}_1 = \mathbf{x}_0 + \alpha\mathbf{d}$$

$$\mathbf{x}_1 = \frac{1}{8}[-7, 0, 5]$$

3 Ejercicio 2

Considere la funcion

$$f(\mathbf{x}) = 2x_1^2 + x_2^2 - 2x_1x_2 + x_1^3 + x_1^4$$

Sea $\mathbf{x}_0 = (0, 1)^T$:

3.1 Paso 1:

Aplique un paso del metodo de Newton a partir del punto si la Hessiana en $\mathbf{x}_0$ es definida positiva. Sino, aplique el algoritmo de descenso maximo con un tamanio de paso apropiado.

Bien. para poder aplicar el metodo de Newton, primeramente necesitamos determinar el gradiente de la funcion y la matriz Hessiana asociada, comenzamos determinando las derivadas parciales para lo anterior entonces

$$\frac{\partial f(\mathbf{x})}{\partial x_1} = 4x_1 - 2x_2 + 3x_1^2 + 4x_1^3$$

$$\frac{\partial f(\mathbf{x})}{\partial x_2} = 2x_2 - 2x_1$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1^2} = 4 + 6x_1 + 12x_1^2$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_2^2} = 2$$

$$\frac{\partial^2 f(\mathbf{x})}{\partial x_1 \partial x_2} = \frac{\partial^2 f(\mathbf{x})}{\partial x_2 \partial x_1} = -2$$

Por lo tanto se tiene que el gradiente es

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 4x_1 - 2x_2 + 3x_1^2 + 4x_1^3 \\ 2x_2 - 2x_1 \end{pmatrix}$$

Y la hessiana asociada

$$H_f(\mathbf{x}) = \begin{pmatrix} 4 + 6x_1 + 12x_1^2 & -2 \\ -2 & 2 \end{pmatrix}$$

Obtenidos estos valores queda determinar si la matriz Hessiana es definida psotiva tal que se cumpla que A es definida positiva si para cualquier $\mathbf{x} \neq 0$ se tiene que

$$\mathbf{x}^T A \mathbf{x} > 0$$

Entonces para nuestra matriz en particular en el punto de interes entonces se tiene que

$$H_f(\mathbf{x}_0) = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix}$$

Aplicando el cirterio de *Sylvester*, el cual nos pide verificar que el determinante de todos los menores principales (superiores o inferiores) de la matriz sean psotivios incluido el determinante de la matriz, probando se tiene que

$$M_1 = H[1] = 4, \quad detM_1 > 0$$

$$M_2 = H[1,2] = H = 4, \quad detM_2 = detH > 0$$

De esta forma entonces, el metodo de Newton nos indica que

$$\mathbf{x}_1 = \mathbf{x}_0 - H_f(\mathbf{x}_0)^{-1} \nabla f(\mathbf{x}_0))$$

De lo cual, la inversa de la matriz Hessiana seria

$$H_f(\mathbf{x}_0)^{-1} = \frac{1}{4} \begin{pmatrix} 2 & 2 \\ 2 & 4 \end{pmatrix}$$

Mientras que el gradiente evaluado en el punto dado seria

$$\nabla f(\mathbf{x}_0) = \begin{pmatrix} -2 \\ 2 \end{pmatrix}$$

Entonces

$$\mathbf{x}_1 = (0,1) - (0,1)$$
$$\mathbf{x}_1 = (0,0)$$

3.2 Paso 2:

Calcule el cambio de la funcion objetivo: $f(\mathbf{x}_1) - f(\mathbf{x}_0)$

Entonces evaluando las funciones se tiene que

$$f(\mathbf{x}_0) = 1$$
$$f(\mathbf{x}_1) = 0$$

A lo que la diferencia $f(\mathbf{x}_1) - f(\mathbf{x}_0) = -1$

4 Ejercicio 3

Supongamos que $f_1, f_2 : \mathbb{R}^n \to \mathbb{R}$ son funciones convexas.

4.1 Paso 1:

Muestre que tambien es convexa la funcion $f(\mathbf{x})$ definida como

$$f(\mathbf{x}) = \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$$

Bien, se dice una funcion es convexa en un dominio si para cuales quiera dos puntos $\mathbf{x}, \mathbf{y}$ y $\lambda \in [0,1]$ se cumple que

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y})$$

Para nuestro caso particular se tiene lo siguiente

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) = \max\{f_1(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}), f_2(\lambda\mathbf{x} + (1-\lambda)\mathbf{y})\}$$

Mientras que la otra parte seria

$$\lambda f(\mathbf{x}) + (1-\lambda) f(\mathbf{y}) = \lambda \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\} + (1-\lambda)\max\{f_1(\mathbf{y}), f_2(\mathbf{y})\}$$

De esta ultima parte se puede reescribir como

$\lambda \max\{f_1(\mathbf{x}), f_2(\mathbf{x})\} + (1-\lambda)\max\{f_1(\mathbf{y}), f_2(\mathbf{y})\} = \max\{\lambda f_1(\mathbf{x}), \lambda f_2(\mathbf{x})\} + \max\{(1-\lambda) f_1(\mathbf{y}), (1-\lambda) f_2(\mathbf{y})\}$

De lo cual, entonces quedaria mostrar la desigualdad, en el caso que supongamos que

$$\max\{f_1(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}), f_2(\lambda\mathbf{x} + (1-\lambda)\mathbf{y})\} = f_1(\lambda\mathbf{x} + (1-\lambda)\mathbf{y})$$

Sabemos dada que es convexa

$$f_1(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) \leq \lambda f_1(\mathbf{x}) + (1-\lambda) f_1(\mathbf{y})$$

Entonces, se deberia cumplir que

$$\lambda f_1(\mathbf{x}) + (1-\lambda) f_1(\mathbf{y}) \leq \max\{\lambda f_1(\mathbf{x}), \lambda f_2(\mathbf{x})\} + \max\{(1-\lambda) f_1(\mathbf{y}), (1-\lambda) f_2(\mathbf{y})\}$$

Lo cual es correcto, ya que siempre se alcanza esta igualdad.

4.2 Paso 2:

Si $n = 1$ y $f_1(-0.4) = 0.36$, $f_1(0.6) = 2.56$, $f_2(-0.4) = 3.66$, $f_2(1) = 2$, identifique el intervalo mas pequenio en el que se puede garantizar que se encuentra el minimizador de la funcion $f(\mathbf{x})$. Explique su respuesta

Dado que f_1 y f_2 son convexas entonces

$$f_1(\lambda(-0.4) + (1-\lambda)(0.6)) \leq \lambda f_1(-0.4) + (1-\lambda) f_1(0.6)$$

$$f_1(0.6 - \lambda) \leq 2.56 - 2.2\lambda$$

$$f_2(\lambda(-0.4) + (1-\lambda)(1)) \leq \lambda f_2(-0.4) + (1-\lambda) f_2(1)$$

$$f_2(1 - 1.4\lambda) \leq 1.66\lambda + 2$$

$$2.56 - 2.2\lambda = 1.66\lambda + 2$$

$$3.86\lambda = 0.56$$

$$\lambda = 0.145$$

$$f_1(0.455) \leq 2.241$$

$$f_2(0.797) \leq 2.241$$

Por lo tanto el intervalo esta definido entre 0.455 y 0.797

5 Ejercicio 4

5.1 Programando el metodo

[54]:
```
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
```

[55]:
```
def m_k(p, fk, gk, hk):
    return fk + (p@gk) + ((p@hk@p)/2)
```

[56]:
```
def met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta):
    eps_m = np.finfo(float).eps
    n = len(x0)
    if n == 2:
        record = []
    else:
        record = None
    x = x0
```

```
    r = (rmax-rmin)/4
    for k in range(NMax):
        if n == 2:
            record.append(x)
        f = fun(x)
        g = grad(x)
        normg = np.sqrt(g@g)
        if normg < tol:
            return k, x, normg, record, True
        B = hess(x)
        den = g@B@g
        if den < eps_m:
            t = 1.0
        else:
            t = min(1, pow(normg, 3)/(den*r))
        p = -t*(r/normg)*g
        rho = (fun(x) - fun(x+p))/(m_k(np.zeros(n), f, g, B) - m_k(p, f, g, B))
        if rho < 1/4 and r > 4*rmin:
            r = r/4
        elif rho>3/4 and np.sqrt(p@p) == r:
            r = min(rmax, 2*r)
        else:
            r = r
        if rho > eta:
            x = x+p
        else:
            x = x
    return k, x, normg, record, False
```

5.2 Probando el metodo

[57]:
```
## Funciones con sus Hessianas y sus Gradientes
def himmelblau(x):
    return ((x[0]**2) + x[1] - 11)**2 + (x[0] + (x[1]**2) - 7)**2

def himmelblau_grad(x):
    return np.array([(4*x[0]*((x[0]**2) + x[1] - 11)) + (2*(x[0] + (x[1]**2) -␣
 ↪7)),
                     (2*((x[0]**2) + x[1] - 11)) + (4*x[1]*(x[0] + (x[1]**2) -␣
 ↪7))])

def himmelblau_hess(x):
    dxx = (12*(x[0]**2)) + (4*x[1]) - 42
    dxy = 4*(x[0]+x[1])
    dyy = (12*(x[1]**2)) + (4*x[0]) - 26
    return np.array([[dxx, dxy],
                     [dxy, dyy]])
```

```
def beale(x):
    return (1.5 - x[0] + (x[0]*x[1]))**2 + (2.25 - x[0] + (x[0]*x[1]**2))**2 +␣
 ↪(2.625 - x[0] + (x[0]*x[1]**3))**2

def beale_grad(x):
    return np.array([ (2*(x[1] - 1) * (1.5 - x[0] + (x[0]*x[1]))) + (2*(x[1]**2␣
 ↪- 1)*(2.25 - x[0] + (x[0]*x[1]**2))) + (2*(x[1]**3 - 1)*(2.625 - x[0] +␣
 ↪(x[0]*x[1]**3))),
                    (2*x[0]*(1.5 - x[0] + (x[0]*x[1]))) + (4*(x[0]*x[1])*(2.25␣
 ↪- x[0] + (x[0]*x[1]**2))) + (6*(x[0]*x[1]**2)*(2.625 - x[0] +␣
 ↪(x[0]*x[1]**3)))])

def beale_hess(x):
    dxx = 2*(((x[1]-1)**2) + (((x[1]**2)-1)**2) + (((x[1]**3)-1)**2))
    dyy = (2*x[0]**2) + (4*x[0]*(2.25-x[0]+(x[0]*x[1]**2))) +␣
 ↪(8*(x[0]*x[1])**2) + (12*x[0]*x[1]*(2.625-x[0]+(x[0]*x[1]**3)))␣
 ↪+(18*(x[0]*x[1]**2)**2)
    dxy = (2*(1.5-x[0]+(x[0]*x[1]))) + (2*x[0]*(x[1]-1)) + (4*x[1]*(2.
 ↪25-x[0]+(x[0]*x[1]**2))) + (4*x[0]*x[1]*(x[1]**2 - 1)) + (6*x[1]**2 *(2.
 ↪625-x[0] + (x[0]*x[1]**3))) + (6*x[0]*x[1]**2*(x[1]**3 - 1))
    return np.array([[dxx, dxy],
                     [dxy, dyy]])


def rosenbrock(x):
    n = len(x)
    res = 0
    for k in range(n-1):
        res += (100 * (x[k+1] - (x[k]**2))**2 ) + (1-x[k])**2
    return res

def rosenbrock_grad(x):
    gradient = np.zeros_like(x)
    n = len(x)
    for i in range(n-1):
        gradient[i] += -400 * x[i] * (x[i+1] - x[i]**2) - 2 * (1 - x[i])
        gradient[i+1] += 200 * (x[i+1] - x[i]**2)
    return gradient

def rosenbrock_hess(x):
    n = len(x)
    sol = np.zeros((n, n))
    for i in range(n-1):
        sol[i, i] += 1200 * x[i]**2 - 400 * x[i+1] + 2
        sol[i, i+1] = -400 * x[i]
```

```
        sol[i+1, i] = -400 * x[i]
        sol[i+1, i+1] += 200
    return sol

def hartmann(x):
    A = np.array([[10, 3, 17, 3.5, 1.7, 8],
                  [0.05, 10, 17, 0.1, 8, 14],
                  [3, 3.5, 1.7, 10, 17, 8],
                  [17, 8, 0.05, 10, 0.1, 14]])
    P = np.array([[1312, 1696, 5569, 124, 8283, 5886],
                  [2329, 4135, 8307, 3736, 1004, 9991],
                  [2348, 1451, 3522, 2883, 3047, 6650],
                  [4047, 8828, 8732, 5743, 1091, 381]])
    P = P / 10000
    a = np.array([1.0, 1.2, 3.0, 3.2])

    n = len(x)
    if n != 6:
        print("Error de dimensionalidad.\n")
        return 0
    sum1 = 0
    for i in range(4):
        sum2 = 0
        for j in range(n):
            sum2 += A[i,j]*(x[j] - P[i,j])**2
        sum1 += a[i] * np.exp(-sum2)

    return -(2.58 + sum1) / 1.94

def hartmann_grad(x):
    A = np.array([[10, 3, 17, 3.5, 1.7, 8],
                  [0.05, 10, 17, 0.1, 8, 14],
                  [3, 3.5, 1.7, 10, 17, 8],
                  [17, 8, 0.05, 10, 0.1, 14]])
    P = np.array([[1312, 1696, 5569, 124, 8283, 5886],
                  [2329, 4135, 8307, 3736, 1004, 9991],
                  [2348, 1451, 3522, 2883, 3047, 6650],
                  [4047, 8828, 8732, 5743, 1091, 381]])
    P = P / 10000
    a = np.array([1.0, 1.2, 3.0, 3.2])
    n = len(x)
    sol = []
    if n != 6:
        print("Error de dimensionalidad.\n")
        return 0
    for k in range(n):
        sum1 = 0
```

```
        for i in range(4):
            sum2 = 0
            for j in range(n):
                sum2 += A[i,j]*((x[j] - P[i,j])**2)
            sum1 += a[i]* A[i,k]*(x[k]-P[i,k])*np.exp(-sum2)
        sol.append((2*sum1)/1.94)
    return np.asarray(sol)

def hartmann_hess(x):
    A = np.array([[10, 3, 17, 3.5, 1.7, 8],
                  [0.05, 10, 17, 0.1, 8, 14],
                  [3, 3.5, 1.7, 10, 17, 8],
                  [17, 8, 0.05, 10, 0.1, 14]])
    P = np.array([[1312, 1696, 5569, 124, 8283, 5886],
                  [2329, 4135, 8307, 3736, 1004, 9991],
                  [2348, 1451, 3522, 2883, 3047, 6650],
                  [4047, 8828, 8732, 5743, 1091, 381]])
    P = P / 10000
    a = np.array([1.0, 1.2, 3.0, 3.2])
    n = len(x)
    if n != 6:
        print("Error de dimensionalidad.\n")
        return 0
    sol =  np.zeros((n,n))
    for k in range(n):
        for l in range(n):
            sum1 = 0
            for i in range(4):
                sum2 = 0
                for j in range(n):
                    sum2 += A[i,j]*(x[j] - P[i,j])**2
                if k != l:
                    sum1 += a[i]* A[i,k]*(x[k]-P[i,k])*A[i,l]*(x[l]-P[i,l])*np.
 ↪exp(-sum2)
                else:
                    sum1 += a[i]*A[i,k]*np.exp(-sum2)*(1 +␣
 ↪(2*A[i,k]*((x[k]-P[i,k])**2)))
            sol[k,l] = (-4*sum1)/1.94
    return sol
```

[58]:
```
def contornosFnc2D(fncf, xleft, xright, ybottom, ytop, levels, puntos=None):
    # Crea una discretización uniforme del intervalo [xleft, xright]
    ax = np.linspace(xleft, xright, 250)
    # Crea una discretización uniforme del intervalo [ybottom, ytop]
    ay = np.linspace(ybottom, ytop, 200)
    # La matriz mX que tiene las abscisas
    mX, mY = np.meshgrid(ax, ay)
```

```
    # Se crea el arreglo mZ con los valores de la función en cada nodo
    mZ = mX.copy()
    for i, y in enumerate(ay):
        for j, x in enumerate(ax):
            mZ[i, j] = fncf(np.array([x, y]))

    # Grafica de las curvas de nivel
    fig, ax = plt.subplots()
    CS = ax.contour(mX, mY, mZ, levels, cmap='Wistia')

    # Grafica los puntos y conecta la secuencia con líneas
    if puntos is not None:
        puntos = np.array(puntos)
        ax.plot(puntos[:, 0], puntos[:, 1], color='red', marker='o',␣
↪linestyle='-', linewidth=2, label='Secuencia de puntos')

    ax.legend()  # Muestra la leyenda si se han graficado puntos
    plt.show()
```

[59]:
```
eps_m = np.finfo(float).eps
NMax = 50000
rmin = 10E-5
eta = 0.25
```

5.2.1 Funcion de Himemblau $\Delta_{max} = 4.0$

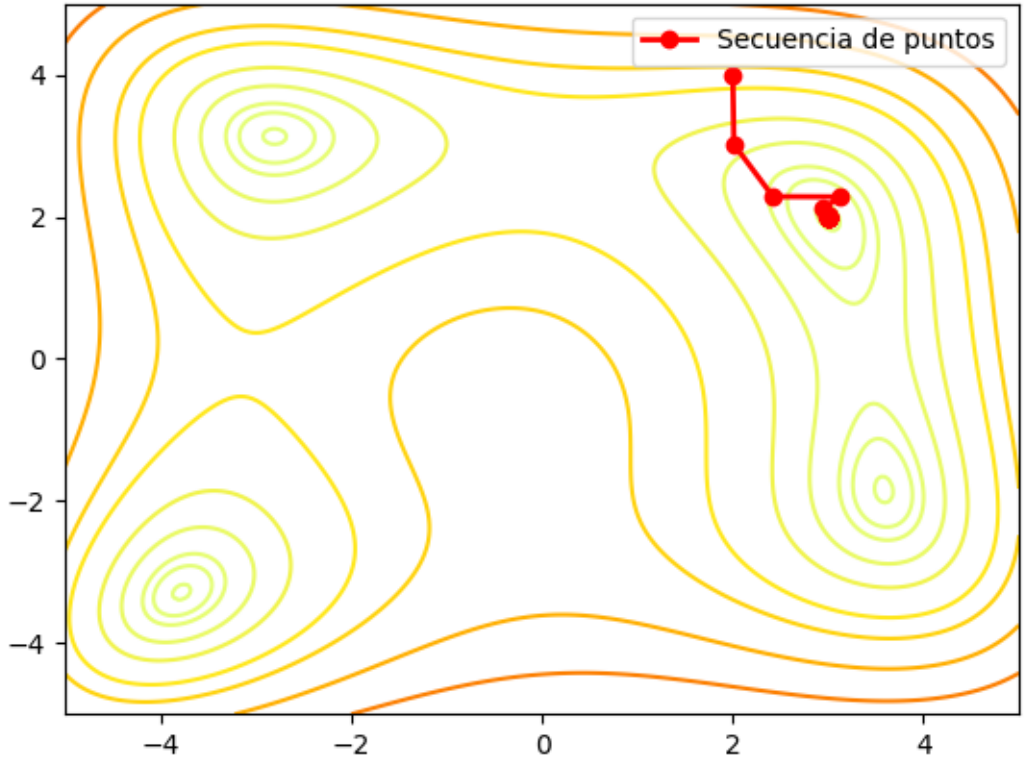
[60]:
```
fun = himmelblau
grad = himmelblau_grad
hess =  himmelblau_hess
x0 = np.asarray([2,4])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-5, 5]
ysearch = [-5, 5]
rmax = 4.0

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  26
El punto optimo encontrado es:  [3. 2.]
```

```
La funcion en el punto optimo es: 5.239958757106325e-18
El modulo del gradiente en el punto optimo es:  1.7654431294742216e-08
El algortimo tuvo el estatus de convergencia:  True
```



5.2.2 Funcion de Himemblau $\Delta_{max} = 0.25$

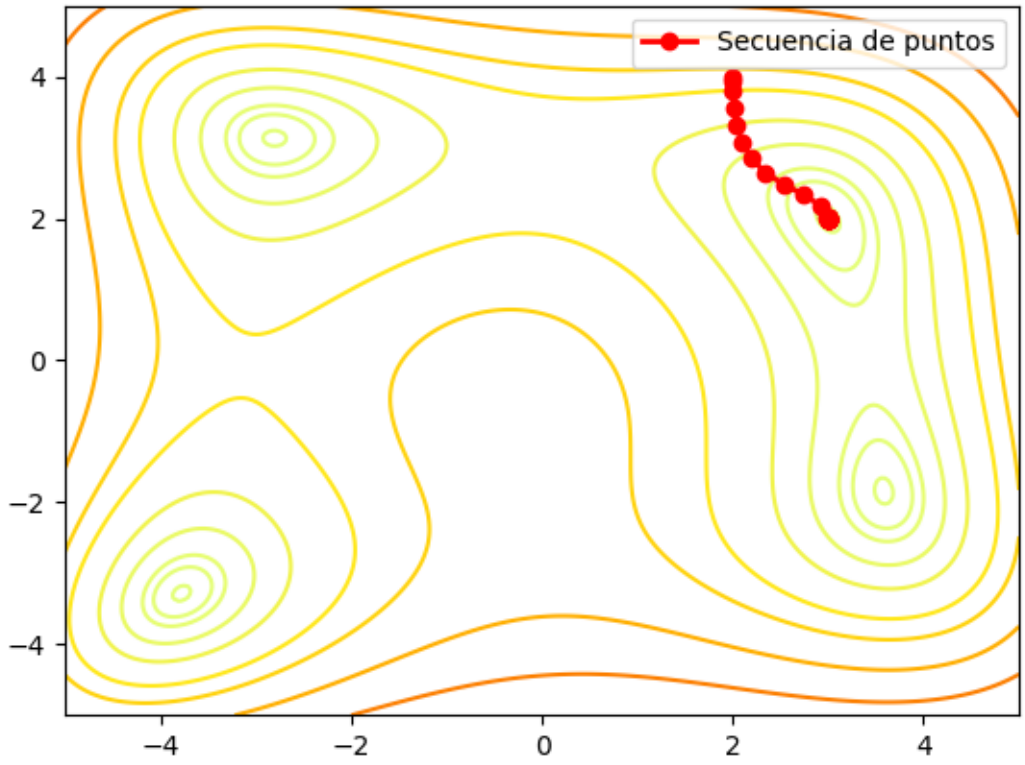
[61]:
```
fun = himmelblau
grad = himmelblau_grad
hess =  himmelblau_hess
x0 = np.asarray([2,4])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-5, 5]
ysearch = [-5, 5]
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
```

```
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  36
El punto optimo encontrado es:  [3. 2.]
La funcion en el punto optimo es: 1.6118850347762392e-18
El modulo del gradiente en el punto optimo es:  1.0097641556169548e-08
El algortimo tuvo el estatus de convergencia:  True
```



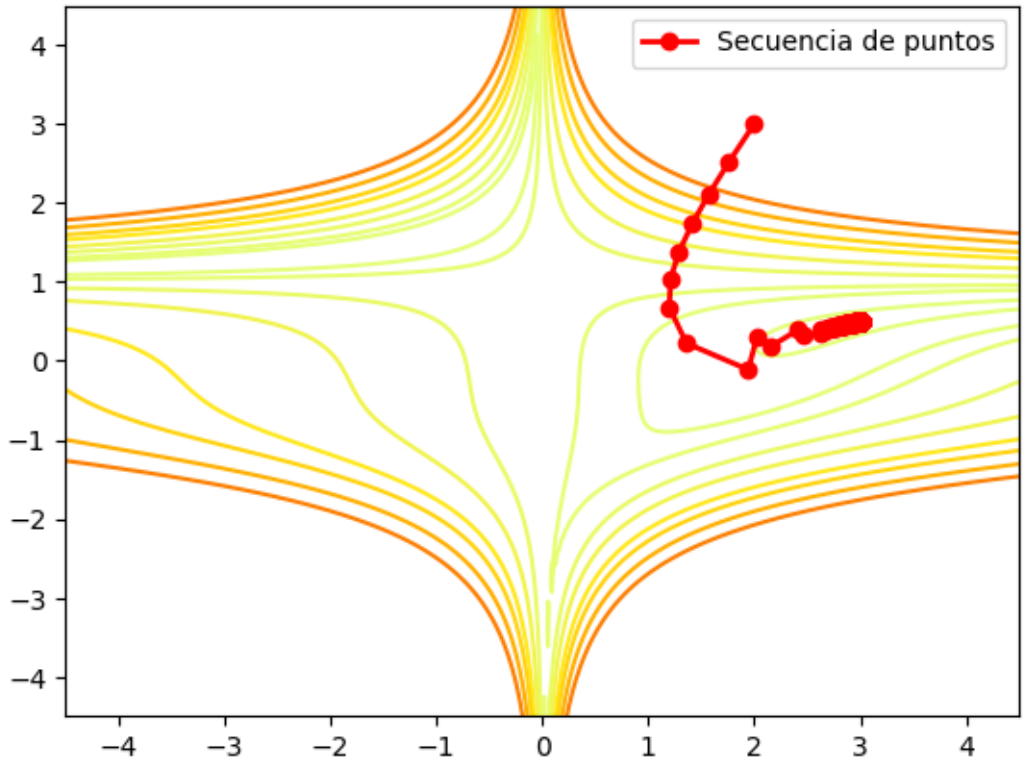
5.2.3 Funcion de Beale $\Delta_{max} = 4.0$

[62]:
```
fun = beale
grad = beale_grad
hess =  beale_hess
x0 = np.asarray([2,3])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-4.5, 4.5]
ysearch = [-4.5, 4.5]
rmax = 4.0

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)
```

```
print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  457
El punto optimo encontrado es:  [2.99999994 0.49999998]
La funcion en el punto optimo es: 5.871164901273395e-16
El modulo del gradiente en el punto optimo es:  1.9644898370792116e-08
El algortimo tuvo el estatus de convergencia:  True
```



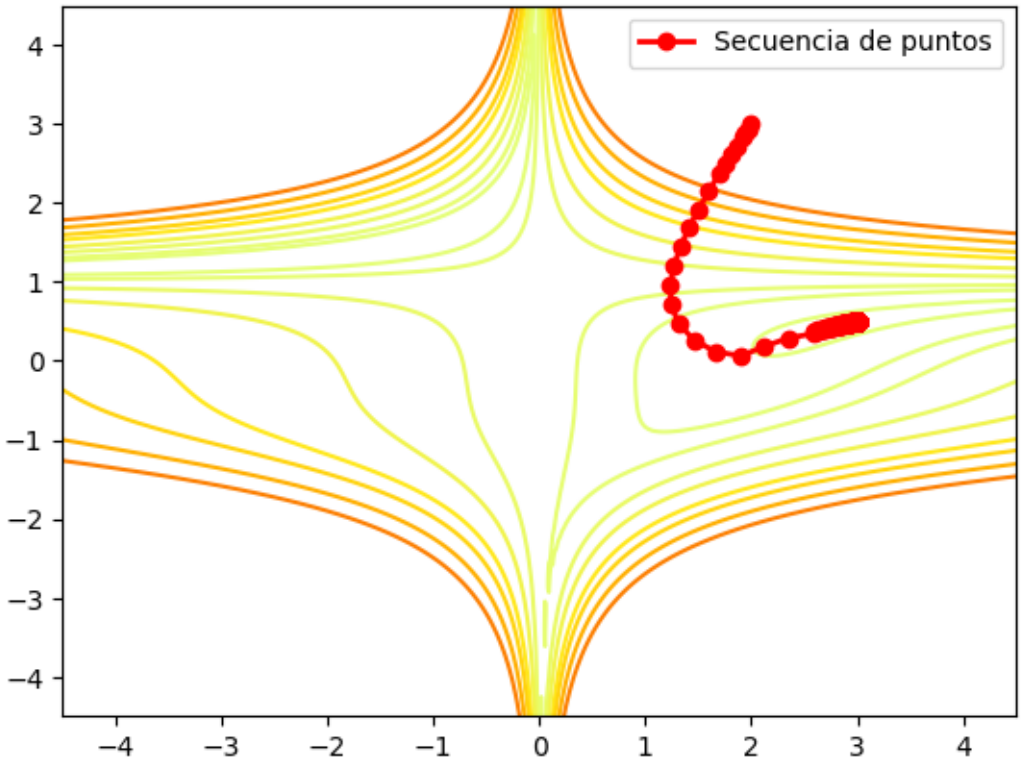
5.2.4 Funcion de Beale $\Delta_{max} = 0.25$

[63]:
```
fun = beale
grad = beale_grad
hess =  beale_hess
x0 = np.asarray([2,3])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-4.5, 4.5]
ysearch = [-4.5, 4.5]
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  339
El punto optimo encontrado es:  [2.99999994 0.49999998]
La funcion en el punto optimo es: 6.729192938507521e-16
El modulo del gradiente en el punto optimo es:  2.0685310361654457e-08
El algortimo tuvo el estatus de convergencia:  True
```



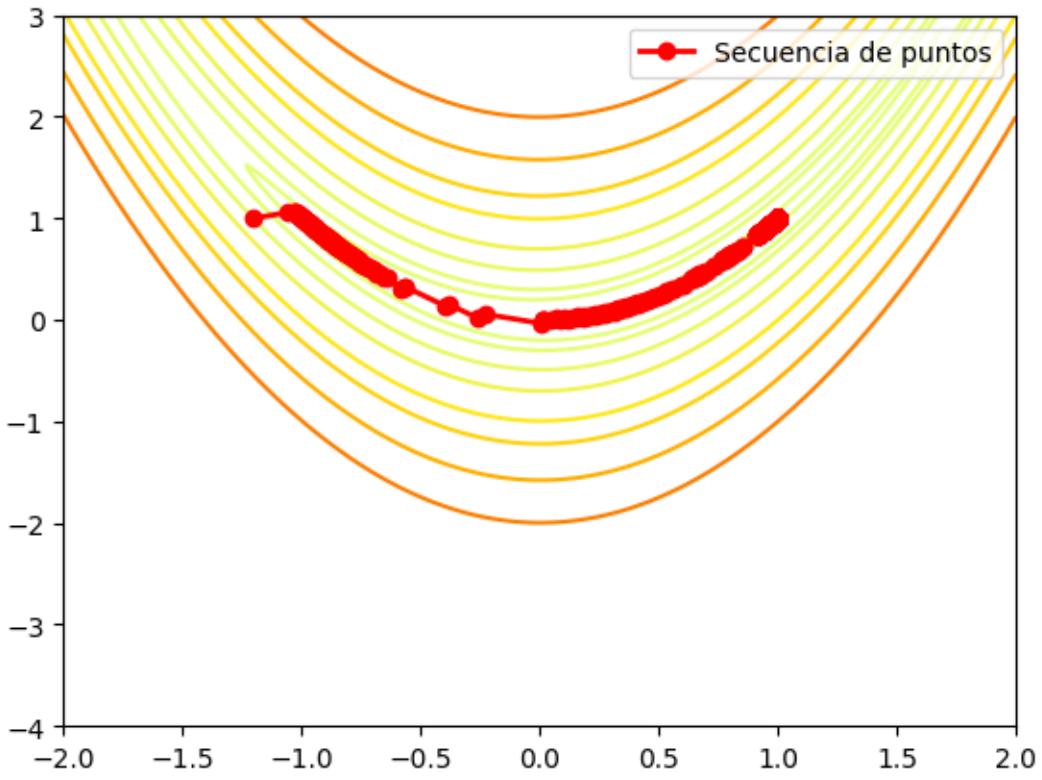
5.2.5 Funcion de Rosenbrock 2D $\Delta_{max} = 4.0$

[64]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
x0 = np.asarray([-1.2,1.0])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 4.0

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  12476
El punto optimo encontrado es:  [0.99999998 0.99999996]
La funcion en el punto optimo es: 4.360379006682842e-16
El modulo del gradiente en el punto optimo es:  2.107317848333999e-08
El algortimo tuvo el estatus de convergencia:  True
```



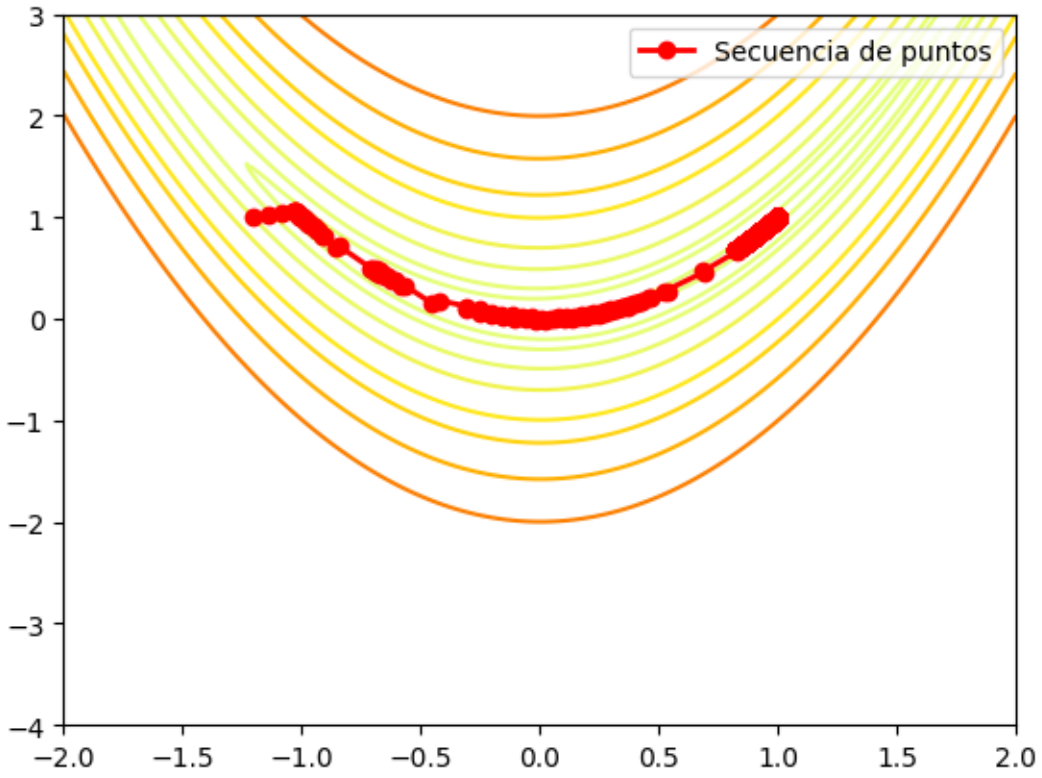
5.2.6 Funcion de Rosenbrock 2D $\Delta_{max} = 0.25$

[65]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
x0 = np.asarray([-1.2,1.0])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  18466
El punto optimo encontrado es:  [0.99999998 0.99999996]
La funcion en el punto optimo es: 3.471928515248036e-16
El modulo del gradiente en el punto optimo es:  2.103873718920268e-08
El algortimo tuvo el estatus de convergencia:  True
```



5.2.7 Funcion de Rosenbrock 10D $\Delta_{max} = 4.0$

[66]:
```
from itertools import cycle
```

[67]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
cic = cycle([-1.2, 1.0])
x0 = np.asarray([next(cic) for _ in range(10)])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 4.0
```

```
k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  28797
El punto optimo encontrado es:  [1.         1.         1.         1.         1.
1.
 0.99999999 0.99999998 0.99999997 0.99999994]
La funcion en el punto optimo es: 1.234992402788919e-15
El modulo del gradiente en el punto optimo es:  4.7079280929432985e-08
El algortimo tuvo el estatus de convergencia:  True
```

5.2.8 Funcion de Rosenbrock 10D $\Delta_{max} = 0.25$

[68]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
cic = cycle([-1.2, 1.0])
x0 = np.asarray([next(cic) for _ in range(10)])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  27351
El punto optimo encontrado es:  [1.         1.         1.         1.         1.
1.
 0.99999999 0.99999998 0.99999997 0.99999994]
```

```
La funcion en el punto optimo es: 1.3854560401019269e-15
El modulo del gradiente en el punto optimo es:  4.7074841776941555e-08
El algortimo tuvo el estatus de convergencia:  True
```

5.2.9 Funcion de Rosenbrock 20D $\Delta_{max} = 4.0$

[69]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
cic = cycle([-1.2, 1.0])
x0 = np.asarray([next(cic) for _ in range(20)])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 4.0

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  29140
El punto optimo encontrado es:  [1.         1.         1.         1.         1.
1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.99999999 0.99999999 0.99999998
 0.99999996 0.99999991]
La funcion en el punto optimo es: 2.693844299012893e-15
El modulo del gradiente en el punto optimo es:  6.659254473273622e-08
El algortimo tuvo el estatus de convergencia:  True
```

5.2.10 Funcion de Rosenbrock 20D $\Delta_{max} = 0.25$

[70]:
```
fun = rosenbrock
grad = rosenbrock_grad
hess =  rosenbrock_hess
cic = cycle([-1.2, 1.0])
x0 = np.asarray([next(cic) for _ in range(20)])
tol = np.sqrt(len(x0)*eps_m)
xsearch = [-2, 2]
ysearch = [-4, 3]
```

```
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  30471
El punto optimo encontrado es:  [1.         1.         1.         1.         1.
1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         0.99999999 0.99999999 0.99999998
 0.99999996 0.99999992]
La funcion en el punto optimo es: 2.390481249878516e-15
El modulo del gradiente en el punto optimo es:  6.66327955969921e-08
El algortimo tuvo el estatus de convergencia:  True
```

5.2.11 Funcion de Hartman 6D $\Delta_{max} = 4.0$

[71]:
```
NMax = 60000
```

[72]:
```
fun = hartmann
grad = hartmann_grad
hess =  hartmann_hess
cic = cycle([0])
x0 = np.asarray([next(cic) for _ in range(6)])
tol = np.sqrt(len(x0)) *  pow(eps_m,1/3)
tol = tol/2
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 4.0

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
```

```
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  59999
El punto optimo encontrado es:  [0.20175888 0.14997227 0.47664287 0.27534581
0.31157408 0.65730226]
La funcion en el punto optimo es: -3.0424573235180583
El modulo del gradiente en el punto optimo es:  0.0042953355073941804
El algortimo tuvo el estatus de convergencia:  False
```

5.2.12 Funcion de Hartman 6D $\Delta_{max} = 0.25$

[73]:
```
fun = hartmann
grad = hartmann_grad
hess =  hartmann_hess
cic = cycle([0])
x0 = np.asarray([next(cic) for _ in range(6)])
tol = np.sqrt(len(x0)) *  pow(eps_m,1/3)
tol = tol/2
xsearch = [-2, 2]
ysearch = [-4, 3]
rmax = 0.25

k, x, g, rcd, bl = met_reg_conf(fun, grad, hess, x0, NMax, tol, rmax, rmin, eta)

print("El algoritmo realizo el siguiente numero de iteraciones: ", k)
print("El punto optimo encontrado es: ", x)
print("La funcion en el punto optimo es:", fun(x))
print("El modulo del gradiente en el punto optimo es: ", g)
print("El algortimo tuvo el estatus de convergencia: ", bl)
if rcd is not None:
    contornosFnc2D(fun, xsearch[0], xsearch[1], ysearch[0], ysearch[1], [0.5,␣
 ↪5, 10, 25, 50, 100, 150, 250, 400], rcd)
```

```
El algoritmo realizo el siguiente numero de iteraciones:  59999
El punto optimo encontrado es:  [0.20171875 0.14999416 0.47677814 0.27533926
0.31165677 0.65730216]
La funcion en el punto optimo es: -3.042457687701415
El modulo del gradiente en el punto optimo es:  0.0010369926185050914
El algortimo tuvo el estatus de convergencia:  False
```

[74]:
```
tol
```

[74]:
```
7.416386784514109e-06
```