PROBLEMA 6.5 - Müller-Brown surface

Gisela Martí Guerrero

Müller-Brown surface in the domain $-1.5 \leq x \leq 1.0$ and $-0.5 \leq y \leq 2.0$ computed with Runge-Kutta 4 and three different Euler methods (simple, modified and improved).

Starting in the first order saddle point (transition state) located in $(-0.822, 0.624)$. The initial condition is the eigenvector of the Hessian matrix of this point with negative eigenvalue, $(x(0), y(0)) = (-0.822, 0.624)$; $g(x(0), y(0)) = v_0$.

Müller-Brwon surface:

$$E(x,y) = \sum_{i=1}^{4} A_i exp(a_i(x - x_i^0)^2 + b_i(x - x_i^0)(y - y_i^0) + c_i(y - y_i^0)^2)$$

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# Muller-Brown parameters
Ai = np.array((-200, -100, -170, 15))
ai = np.array((-1, -1, -6.5, 0.7))
bi = np.array((0, 0, 11, 0.6))
ci = np.array((-10, -10, -6.5, 0.7))
xoi = np.array((1, 0, -0.5, -1))
yoi = np.array((0, 0.5, 1.5, 1))

# Initial point (TS)
r0 = np.array([-0.822,0.624])
```

In [ ]:
```
def E(x, y):
    energy = np.sum(Ai[:, np.newaxis, np.newaxis] * \
             np.exp(ai[:, np.newaxis, np.newaxis] * \
             (x - xoi[:, np.newaxis, np.newaxis])**2 + \
             bi[:, np.newaxis, np.newaxis] * \
             (x - xoi[:, np.newaxis, np.newaxis]) * \
             (y - yoi[:, np.newaxis, np.newaxis]) + \
             ci[:, np.newaxis, np.newaxis] * \
             (y - yoi[:, np.newaxis, np.newaxis])**2), axis=0)
    return energy
```

In [ ]:
```
def gradient(x, y):
    dx = sum(Ai[i]*(2*ai[i]*(x-xoi[i])+bi[i]*(y-yoi[i]))*np.exp(ai[i]* \
         (x-xoi[i])**2+bi[i]*(x-xoi[i])*(y-yoi[i])+ci[i]* \
         (y-yoi[i])**2) for i in range(4))
    dy = sum(Ai[i]*(bi[i]*(x-xoi[i])+2*ci[i]*(y-yoi[i]))*np.exp(ai[i]* \
         (x-xoi[i])**2+bi[i]*(x-xoi[i])*(y-yoi[i])+ci[i]* \
         (y-yoi[i])**2) for i in range(4))
    return np.array([dx, dy])
```

In [ ]:
```
def hessian(x, y):
    dxdx = sum(Ai[i]*(2*ai[i]+4*ai[i]**2*(x-xoi[i])**2+bi[i]**2* \
           (y-yoi[i])**2+2*ai[i]*bi[i]*(x-xoi[i])*(y-yoi[i]))*np.exp(ai[i]* \
           (x-xoi[i])**2+bi[i]*(x-xoi[i])*(y-yoi[i])+ci[i]* \
           (y-yoi[i])**2) for i in range(4))
    dydy = sum(Ai[i]*(2*ci[i]+4*ci[i]**2*(y-yoi[i])**2+bi[i]**2* \
           (x-xoi[i])**2+2*bi[i]*ci[i]*(x-xoi[i])*(y-yoi[i]))*np.exp(ai[i]* \
           (x-xoi[i])**2+bi[i]*(x-xoi[i])*(y-yoi[i])+ci[i]* \
            (y-yoi[i])**2) for i in range(4))
    dxdy = sum(Ai[i]*(bi[i]*(x-xoi[i])+2*ci[i]*(y-yoi[i])+2*ai[i]*bi[i]*(x-xoi[i])*
           (y - yoi[i]))*np.exp(ai[i]*(x-xoi[i])**2+bi[i]*(x-xoi[i])*(y-yoi[i])+ \
           ci[i]*(y-yoi[i])**2) for i in range(4))
    return np.array([[dxdx, dxdy], [dxdy, dydy]])
```

In [3]:
```
# Runge-Kutta4 method
def RK4(x,dt,f):
    f0 = f(x)
    f1 = f(x + f0*dt/2)
    f2 = f(x + f1*dt/2)
    f3 = f(x + f2*dt)
    xt = x + dt/6*(f0 + 2*f1 + 2*f2 + f3)
    return xt

# Euler methods
def euler(x, f, dt, mode):
    if mode.lower() == "simple":
        a, b, d, g = 1, 0, 0, 0
    elif mode.lower() == "modified":
        a, b, d, g = 0, 1, 0.5, 0.5
    elif mode.lower() == "improved":
        a, b, d, g = 0.5, 0.5, 1, 1

    xt = x + dt*(a*f(x) + b*f(x+f(x)*d*dt))
    return xt

def integrate_grad(r):
    return -gradient(r[0], r[1])
```

In [4]:
```
E_init = E(r0[0],r0[1])
G_init = gradient(r0[0],r0[1])
H_init = hessian(r0[0],r0[1])

print("Energy at the initial point:", float(E_init))
print("Gradient at the initial point:", G_init)
print("Hessian at the initial point:")
print(H_init)

eigenvalues, eigenvectors = np.linalg.eig(H_init)
print("Eigenvalues:",eigenvalues)
print("Eigenvectors:")
print(eigenvectors)
```

```
Energy at the initial point: -40.66484530104902
Gradient at the initial point: [-0.19188051  0.01037288]
Hessian at the initial point:
[[-758.89569697  529.66607449]
 [ 529.66607449 -558.51124521]]
Eigenvalues: [-1197.76249624  -119.64444594]
Eigenvectors:
[[-0.77002112 -0.6380184 ]
 [ 0.6380184  -0.77002112]]
```

In [5]:
```
dt = 0.00001
# First vector
rdir = eigenvectors[:,0]*200*dt + r0
positions_RK = [r0,rdir]           # Positions (Runge-Kutta)
positions_ES = [r0,rdir]           # Positions (Euler simple)
positions_EM = [r0,rdir]           # Positions (Euler modified)
positions_EI = [r0,rdir]           # Positions (Euler improveed)

# Second vector
rdir2 = r0 - eigenvectors[:,1]*200*dt
positions_RK2 = [r0,rdir2]         # Positions (Runge-Kutta)
positions_ES2 = [r0,rdir2]         # Positions (Euler simple)
positions_EM2 = [r0,rdir2]         # Positions (Euler modified)
positions_EI2 = [r0,rdir2]         # Positions (Euler improveed)
```

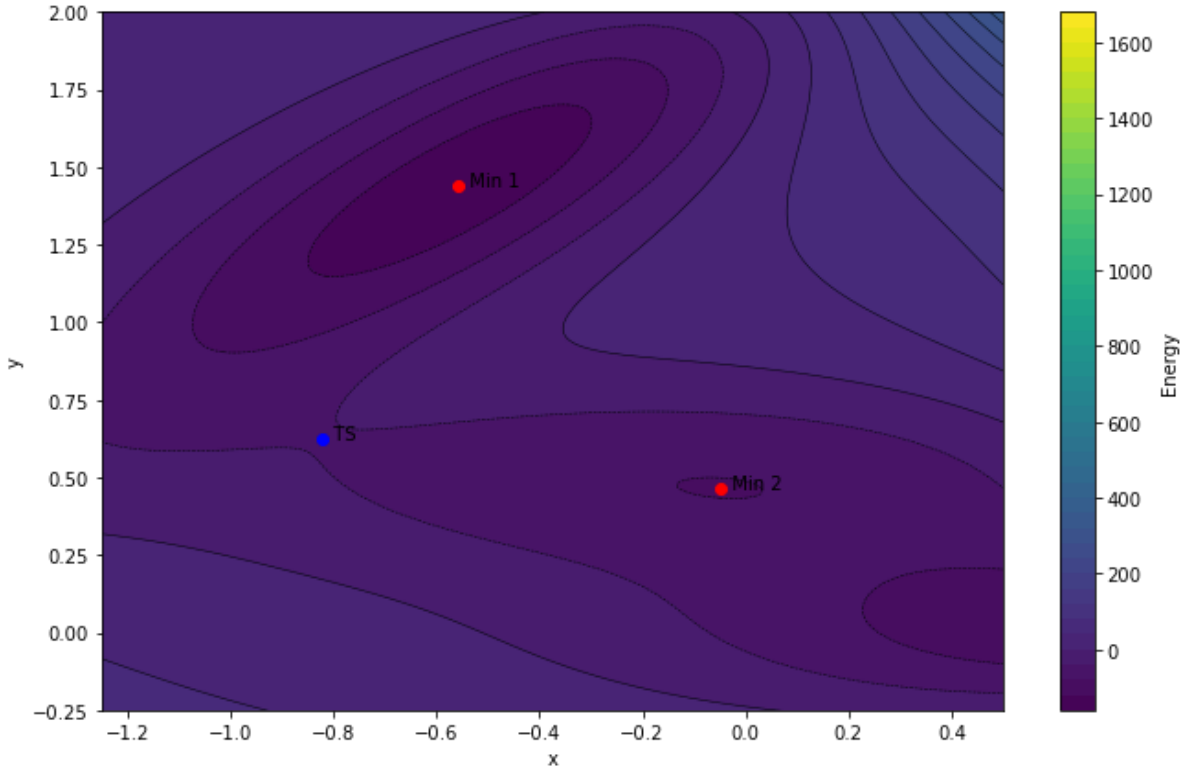
In [6]:
```
# MAIN LOOP
for i in range(10000):
    # Runge-Kutta4 method
    r_RK = RK4(positions_RK[-1], dt, integrate_grad)
    r_RK2 = RK4(positions_RK2[-1], dt, integrate_grad)
    positions_RK.append(r_RK)
    positions_RK2.append(r_RK2)

    # Euler Methods
    r_ES = euler(positions_ES[-1], integrate_grad, dt, mode="simple")
    r_EM = euler(positions_EM[-1], integrate_grad, dt, mode="modified")
    r_EI = euler(positions_EI[-1], integrate_grad, dt, mode="improved")
    r_ES2 = euler(positions_ES2[-1], integrate_grad, dt, mode="simple")
    r_EM2 = euler(positions_EM2[-1], integrate_grad, dt, mode="modified")
    r_EI2 = euler(positions_EI2[-1], integrate_grad, dt, mode="improved")
    positions_ES.append(r_ES)
    positions_EM.append(r_EM)
    positions_EI.append(r_EI)
    positions_ES2.append(r_ES2)
    positions_EM2.append(r_EM2)
    positions_EI2.append(r_EI2)

positions_RK = np.array(positions_RK)
positions_RK2 = np.array(positions_RK2)
positions_ES = np.array(positions_ES)
positions_EM = np.array(positions_EM)
positions_EI = np.array(positions_EI)
positions_ES2 = np.array(positions_ES2)
positions_EM2 = np.array(positions_EM2)
positions_EI2 = np.array(positions_EI2)
```

In [7]:
```
# Energy surface calculation
x = np.linspace(-1.5,1.,1000)
y = np.linspace(-0.5,2.,1000)
XX,YY = np.meshgrid(x,y)
ener = E(XX,YY)
```

In [13]:
```
# Plot
fig, ax = plt.subplots(figsize=(11, 7))
c = ax.contourf(XX, YY, ener, levels=50, cmap='viridis')
contour = ax.contour(XX, YY, ener, levels=50, colors='black', linewidths=0.5)
plt.colorbar(c, label='Energy')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.plot(*positions_RK2[-1], 'ro'), plt.text(*positions_RK2[-1], "  Min 2")
plt.plot(*positions_RK[-1], 'ro'), plt.text(*positions_RK[-1], "  Min 1")
plt.plot(*r0, 'bo'), plt.text(*r0, "  TS")
plt.xlim([-1.25,0.5])
plt.ylim([-0.25,2.0])
plt.show()
```



In [9]:
```
print("Transition state position (x,y):",r0[0],",",r0[1])
print("        ------------ First minimum ------------")
print("Runge-Kutta minimum position (x,y):",positions_RK[-1,0],",",positions_RK[-1,
print("Simple Euler minimum position (x,y):",positions_ES[-1,0],",",positions_ES[-1
print("Modified Euler minimum position (x,y):",positions_EM[-1,0],",",positions_EM[
print("Improved Euler minimum position (x,y):",positions_EI[-1,0],",",positions_EI[
print("        ------------ Second minimum ------------")
print("Runge-Kutta minimum position (x,y):",positions_RK2[-1,0],",",positions_RK2[-
print("Simple Euler minimum position (x,y):",positions_ES2[-1,0],",",positions_ES2[
print("Modified Euler minimum position (x,y):",positions_EM2[-1,0],",",positions_EM
print("Improved Euler minimum position (x,y):",positions_EI2[-1,0],",",positions_EI
```

```
Transition state position (x,y): -0.822 , 0.624
        ------------ First minimum ------------
Runge-Kutta minimum position (x,y): -0.5582236346330437 , 1.441725841804648
Simple Euler minimum position (x,y): -0.5582236346330435 , 1.4417258418046481
Modified Euler minimum position (x,y): -0.5582236346330435 , 1.4417258418046481
Improved Euler minimum position (x,y): -0.5582236346330437 , 1.441725841804648
        ------------ Second minimum ------------
Runge-Kutta minimum position (x,y): -0.0500108251362117 , 0.4666941051335285
Simple Euler minimum position (x,y): -0.05001082510701608 , 0.46669410512995696
Modified Euler minimum position (x,y): -0.05001082513627538 , 0.46669410513353626
Improved Euler minimum position (x,y): -0.05001082513628125 , 0.466694105133537
```