Assignment 3 Unconstrained Nonlinear Numerical Optimization: Gradient Free and Approximation Methods

A) Consider an objective function to be minimized:

$$f(x) = \Sigma_{k=1}^{2} [(-1)^k (x_k - k)^{2k}]$$

with $x_k \in [-5, 5]$; k = 1, 2.

Consider Nelder–Mead's Downhill Simplex Algorithm with parameters:

- $\alpha = 1$
- $\beta = 2$
- $\gamma = 1/2$
- $\delta = 1/2$

Construct an initial simplex such that:

- $x_1 = [-5, 5]^T$
- $x_2 = [0, 0]^T$
- $x_3 = [5, -5]^T$

For 3 Iteration

In [12]:
```
from sympy import symbols, lambdify
from math import isfinite
from scipy.optimize import minimize

# Define symbolic variables and function: f(x) = sum_{k=1}^2 (-1)^k * (x_k - k)^(2k)
x1, x2 = symbols('x1 x2')
f_sym = (-1)**1 * (x1 - 1)**2 + (-1)**2 * (x2 - 2)**4  # simplifies to -(x1-1)^2 + (x2-2)^4
f_num = lambdify((x1, x2), f_sym, 'math')  # numeric function using Python's math

def f_vec(x):
    # x is an array-like of length 2
    return float(f_num(x[0], x[1]))

# Given initial simplex vertices
x1_init = [-5.0,  5.0]
x2_init = [ 0.0,  0.0]
x3_init = [ 5.0, -5.0]

initial_simplex = [x1_init, x2_init, x3_init]

# Use SciPy's minimize with Nelder-Mead
res = minimize(f_vec, x0=x2_init, method='Nelder-Mead',
               options={
                   'initial_simplex': initial_simplex,
                   'maxiter': 3,
                   'xatol': 1e-12,
                   'fatol': 1e-12,
                   'disp': True
               })

print("Result after iterations:")
print("x:", res.x)
print("fun:", res.fun)
print("nit (iterations):", res.nit)
print("message:", res.message)
print("success:", res.success)
```

```
Result after iterations:
x: [-0.625  0.625]
fun: 0.933837890625
nit (iterations): 3
message: Maximum number of iterations has been exceeded.
success: False
C:\Users\shiva\AppData\Local\Temp\ipykernel_18120\2058112449.py:22: RuntimeWarning: Maximum number of iterations has been exceeded.
  res = minimize(f_vec, x0=x2_init, method='Nelder-Mead',
```

For 5 Iteration

In [13]:
```
from sympy import symbols, lambdify
from math import isfinite
from scipy.optimize import minimize

# Define symbolic variables and function: f(x) = sum_{k=1}^2 (-1)^k * (x_k - k)^(2k)
x1, x2 = symbols('x1 x2')
f_sym = (-1)**1 * (x1 - 1)**2 + (-1)**2 * (x2 - 2)**4  # simplifies to -(x1-1)^2 + (x2-2)^4
f_num = lambdify((x1, x2), f_sym, 'math')  # numeric function using Python's math

def f_vec(x):
    # x is an array-like of length 2
    return float(f_num(x[0], x[1]))

# Given initial simplex vertices
x1_init = [-5.0,  5.0]
x2_init = [ 0.0,  0.0]
x3_init = [ 5.0, -5.0]

initial_simplex = [x1_init, x2_init, x3_init]

# Use SciPy's minimize with Nelder-Mead
res = minimize(f_vec, x0=x2_init, method='Nelder-Mead',
               options={
                   'initial_simplex': initial_simplex,
                   'maxiter': 5,
                   'xatol': 1e-12,
                   'fatol': 1e-12,
                   'disp': True
               })

print("Result after iterations:")
print("x:", res.x)
print("fun:", res.fun)
print("nit (iterations):", res.nit)
print("message:", res.message)
print("success:", res.success)
```

```
Result after iterations:
x: [-3.28125  3.28125]
fun: -15.634245872497559
nit (iterations): 5
message: Maximum number of iterations has been exceeded.
success: False
```

```
C:\Users\shiva\AppData\Local\Temp\ipykernel_18120\4038032105.py:22: RuntimeWarning: Maximum number of iterations
has been exceeded.
  res = minimize(f_vec, x0=x2_init, method='Nelder-Mead',
```

For 100 Iteration

In [14]:
```
from sympy import symbols, lambdify
from math import isfinite
from scipy.optimize import minimize

# Define symbolic variables and function: f(x) = sum_{k=1}^2 (-1)^k * (x_k - k)^(2k)
x1, x2 = symbols('x1 x2')
f_sym = (-1)**1 * (x1 - 1)**2 + (-1)**2 * (x2 - 2)**4  # simplifies to -(x1-1)^2 + (x2-2)^4
f_num = lambdify((x1, x2), f_sym, 'math')  # numeric function using Python's math

def f_vec(x):
    # x is an array-like of length 2
    return float(f_num(x[0], x[1]))

# Given initial simplex vertices
x1_init = [-5.0,  5.0]
x2_init = [ 0.0,  0.0]
x3_init = [ 5.0, -5.0]

initial_simplex = [x1_init, x2_init, x3_init]

# Use SciPy's minimize with Nelder-Mead
res = minimize(f_vec, x0=x2_init, method='Nelder-Mead',
               options={
                   'initial_simplex': initial_simplex,
                   'maxiter': 100,
                   'xatol': 1e-12,
                   'fatol': 1e-12,
                   'disp': True
```

```
             })

print("Result after iterations:")
print("x:", res.x)
print("fun:", res.fun)
print("nit (iterations):", res.nit)
print("message:", res.message)
print("success:", res.success)
```

```
Optimization terminated successfully.
         Current function value: -15.634872
         Iterations: 64
         Function evaluations: 138
Result after iterations:
x: [-3.28962389  3.28962389]
fun: -15.634872460323553
nit (iterations): 64
message: Optimization terminated successfully.
success: True
```

B) Find the global minimum of Eason's function using the BFGS method.

$$f(x) = \cos(x) \cdot e^{-(x-\pi)^2}, \quad x \in [-5, 5]$$

Investigate the effect of starting from different initial solutions, say $x_0 = -5$ and $x_0 = 5$.

For 3 Iteration

In [16]:
```python
from math import pi, cos, exp
from scipy.optimize import minimize

# Define Easom-like function
def easom(x):
    x = x[0]  # SciPy passes x as an array
    return cos(x) * exp(-(x - pi)**2)

# Run BFGS starting from x0 = -5
res1 = minimize(easom, x0=[-5.0], method="BFGS", options={"disp": True,"maxiter":3}) # Added maxiter to limit ite
print("Start -5 →", res1.x, res1.fun)

# Run BFGS starting from x0 = 5
res2 = minimize(easom, x0=[5.0], method="BFGS", options={"disp": True,"maxiter":3}) # Added maxiter to limit iter
print("Start 5 →", res2.x, res2.fun)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 2
         Gradient evaluations: 1
Start -5 → [-5.] 4.62765664695O5575e-30
         Current function value: 0.000001
         Iterations: 3
         Function evaluations: 18
         Gradient evaluations: 9
Start 5 → [6.82085293] 1.1348018149256301e-06
```

Unconstrained problem

In [5]:
```python
from math import pi, cos, exp
from scipy.optimize import minimize

# Define Easom-like function
def easom(x):
    x = x[0]  # SciPy passes x as an array
    return cos(x) * exp(-(x - pi)**2)

# Run BFGS starting from x0 = -5
res1 = minimize(easom, x0=[-5.0], method="BFGS", options={"disp": True})
print("Start -5 →", res1.x, res1.fun)

# Run BFGS starting from x0 = 5
res2 = minimize(easom, x0=[5.0], method="BFGS", options={"disp": True})
print("Start 5 →", res2.x, res2.fun)
```

```
Optimization terminated successfully.
         Current function value: 0.000000
         Iterations: 0
         Function evaluations: 2
         Gradient evaluations: 1
Start -5 → [-5.] 4.6276566469505575e-30
Optimization terminated successfully.
         Current function value: 0.000001
         Iterations: 3
         Function evaluations: 18
         Gradient evaluations: 9
Start 5 → [6.82085293] 1.1348018149256301e-06
```

The global minimum $f_* = -1$ occurs at $x_* = \pi$. You may notice, starting from $x_0 = 5$ may obtain your optimal solution much quicker than starting from $x_0 = -5$.

C) Consider an objective function to be minimized:

$$f(x) = \Sigma_{k=1}^{2} [(-1)^k (x_k - k)^{2k}]$$

Apply the Trust Region Algorithm Let the initial point be $[0, 0]^T$ and an initial trust region radius of 1.

Algorithm parameters:

- $\alpha_1 = 0.01$
- $\alpha_2 = 0.9$
- $\beta_1 = \beta_2 = 0.5$

For 3 Iteration, uses Numpy

In [17]:
```python
from math import pi
import numpy as np
from scipy.optimize import minimize

# Define the function
def f(x):
    x1, x2 = x
    return -(x1 - 1)**2 + (x2 - 2)**4

# Gradient
def grad(x):
    x1, x2 = x
    return np.array([-2*(x1 - 1), 4*(x2 - 2)**3])

# Hessian
def hess(x):
    x1, x2 = x
    return np.array([[-2, 0], [0, 12*(x2 - 2)**2]])

# Initial point
x0 = np.array([0.0, 0.0])

# Trust Region Solve (trust-constr)
res = minimize(f, x0, method='trust-constr',
               jac=grad, hess=hess,
               options={'initial_tr_radius': 1.0,
                        'maxiter': 3,   # only 3 iterations
                        'verbose': 3})

print("\nFinal result after 3 iterations:")
print("x:", res.x)
print("f(x):", res.fun)
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  | penalty  |CG stop|
|-------|-------|-------|-------------|----------|----------|----------|----------|-------|
|   1   |   1   |   0   | +1.5000e+01 | 1.00e+00 | 3.20e+01 | 0.00e+00 | 1.00e+00 |   0   |
|   2   |   2   |   2   | +8.5679e-02 | 7.00e+00 | 9.39e+00 | 0.00e+00 | 1.00e+00 |   3   |
|   3   |   3   |   4   | -7.5597e+01 | 4.90e+01 | 1.74e+01 | 0.00e+00 | 1.00e+00 |   3   |

The maximum number of function evaluations is exceeded.
Number of iterations: 3, function evaluations: 3, CG iterations: 4, optimality: 1.74e+01, constraint violation: 0.00e+00, execution time: 0.024 s.

Final result after 3 iterations:
x: [-7.70122791  1.41890305]
f(x): -75.59734370789705
```

For 3 Iteration, Used Jax

In [18]:
```python
import jax
import jax.numpy as jnp
from scipy.optimize import minimize

# Define Easom-like function (from part a)
def f(x):
    x1, x2 = x
    return -(x1 - 1)**2 + (x2 - 2)**4

# Wrap with JAX
f_jax = lambda x: f(x)

# Gradient and Hessian via JAX
grad_f = jax.grad(lambda x1, x2: f_jax((x1, x2)), argnums=(0,1))
hess_f = jax.hessian(lambda xy: f_jax((xy[0], xy[1])))

def fun(x):
    return float(f_jax(x))

def grad(x):
    g = grad_f(x[0], x[1])
    return jnp.array(g, dtype=float)

def hess(x):
    H = hess_f(x)
    return jnp.array(H, dtype=float)

x0 = jnp.array([0.0, 0.0])

res = minimize(fun, x0, method="trust-constr",
               jac=grad, hess=hess,
               options={"initial_tr_radius": 1.0,
                        "maxiter": 3, "verbose": 3})

print("\nFinal result after 3 iterations:")
print("x:", res.x)
print("f(x):", res.fun)
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  | penalty  |CG stop|
|-------|-------|-------|-------------|----------|----------|----------|----------|-------|
|   1   |   1   |   0   | +1.5000e+01 | 1.00e+00 | 3.20e+01 | 0.00e+00 | 1.00e+00 |   0   |
|   2   |   2   |   2   | +8.5679e-02 | 7.00e+00 | 9.39e+00 | 0.00e+00 | 1.00e+00 |   3   |
|   3   |   3   |   4   | -7.5597e+01 | 4.90e+01 | 1.74e+01 | 0.00e+00 | 1.00e+00 |   3   |

The maximum number of function evaluations is exceeded.
Number of iterations: 3, function evaluations: 3, CG iterations: 4, optimality: 1.74e+01, constraint violation:
0.00e+00, execution time: 0.086 s.

Final result after 3 iterations:
x: [-7.70122792  1.41890302]
f(x): -75.59734373651207
```

For 3 Iteration, uses Sympy

In [19]:
```python
import sympy as sp
from scipy.optimize import minimize

# Symbols
x1, x2 = sp.symbols("x1 x2")
f_expr = -(x1 - 1)**2 + (x2 - 2)**4

# Lambdify
f_fun = sp.lambdify((x1, x2), f_expr, "math")
grad_fun = sp.lambdify((x1, x2), [sp.diff(f_expr, x1), sp.diff(f_expr, x2)], "math")
hess_fun = sp.lambdify((x1, x2), [[sp.diff(f_expr, x1, x1), sp.diff(f_expr, x1, x2)],
                                  [sp.diff(f_expr, x2, x1), sp.diff(f_expr, x2, x2)]], "math")

def fun(x):
    return f_fun(x[0], x[1])

def grad(x):
    return grad_fun(x[0], x[1])

def hess(x):
    return hess_fun(x[0], x[1])

x0 = [0.0, 0.0]
```

```
res = minimize(fun, x0, method="trust-constr",
               jac=grad, hess=hess,
               options={"initial_tr_radius": 1.0,
                        "maxiter": 3, "verbose": 3})

print("\nFinal result after 3 iterations:")
print("x:", res.x)
print("f(x):", res.fun)
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  | penalty  |CG stop|
|-------|-------|-------|-------------|----------|----------|----------|----------|-------|
|   1   |   1   |   0   | +1.5000e+01 | 1.00e+00 | 3.20e+01 | 0.00e+00 | 1.00e+00 |   0   |
|   2   |   2   |   2   | +8.5679e-02 | 7.00e+00 | 9.39e+00 | 0.00e+00 | 1.00e+00 |   3   |
|   3   |   3   |   4   | -7.5597e+01 | 4.90e+01 | 1.74e+01 | 0.00e+00 | 1.00e+00 |   3   |

The maximum number of function evaluations is exceeded.
Number of iterations: 3, function evaluations: 3, CG iterations: 4, optimality: 1.74e+01, constraint violation:
0.00e+00, execution time: 0.011 s.

Final result after 3 iterations:
x: [-7.70122791  1.41890305]
f(x): -75.59734370789705
```

In [20]:

```
import sympy as sp
from scipy.optimize import minimize

# Symbols
x1, x2 = sp.symbols("x1 x2")
f_expr = -(x1 - 1)**2 + (x2 - 2)**4

# Lambdify
f_fun = sp.lambdify((x1, x2), f_expr, "math")
grad_fun = sp.lambdify((x1, x2), [sp.diff(f_expr, x1), sp.diff(f_expr, x2)], "math")
hess_fun = sp.lambdify((x1, x2), [[sp.diff(f_expr, x1, x1), sp.diff(f_expr, x1, x2)],
                                  [sp.diff(f_expr, x2, x1), sp.diff(f_expr, x2, x2)]], "math")

def fun(x):
    return f_fun(x[0], x[1])

def grad(x):
    return grad_fun(x[0], x[1])

def hess(x):
    return hess_fun(x[0], x[1])

x0 = [0.0, 0.0]

res = minimize(fun, x0, method="trust-constr",
               jac=grad, hess=hess,tol=1e-2,
               options={"initial_tr_radius": 1.0,
                        'maxiter': 100,
                        "verbose": 3,
                        'gtol': 1e-2,         # Gradient norm tolerance
                        'xtol': 1e-2,         # Step size tolerance
                        'barrier_tol': 1e-2,  # Optional for constraints
                        })

print("\nFinal result after 100 iterations:")
print("x:", res.x)
print("f(x):", res.fun)
```

```
| niter |f evals|CG iter|  obj func   |tr radius |   opt    |  c viol  | penalty  |CG stop|
|-------|-------|-------|-------------|----------|----------|----------|----------|-------|
|   1   |   1   |   0   | +1.5000e+01 | 1.00e+00 | 3.20e+01 | 0.00e+00 | 1.00e+00 |   0   |
|   2   |   2   |   2   | +8.5679e-02 | 7.00e+00 | 9.39e+00 | 0.00e+00 | 1.00e+00 |   3   |
|   3   |   3   |   4   | -7.5597e+01 | 4.90e+01 | 1.74e+01 | 0.00e+00 | 1.00e+00 |   3   |
|  98   |  89   |  178  | -1.1028e+50 | 1.85e+19 | 6.33e+32 | 0.00e+00 | 1.00e+00 |   1   |
|  99   |  90   |  180  | -1.1028e+50 | 3.71e+19 | 6.98e+33 | 0.00e+00 | 1.00e+00 |   2   |
|  100  |  91   |  182  | -1.1028e+50 | 3.71e+19 | 1.91e+33 | 0.00e+00 | 1.00e+00 |   1   |

The maximum number of function evaluations is exceeded.
Number of iterations: 100, function evaluations: 91, CG iterations: 182, optimality: 1.91e+33, constraint violati
on: 0.00e+00, execution time: 0.17 s.

Final result after 100 iterations:
x: [-1.05014601e+25  7.81761577e+10]
f(x): -1.1028062730902755e+50
```