Particle Swarm Optimization (PSO)

In [1]:
```
import numpy as np
from numpy import absolute
from numpy.random import uniform, choice
from random import randint

import seaborn as sns
import matplotlib.pyplot as plt

from IPython.display import HTML, display
from pprint import pprint
from tabulate import tabulate
```

In [2]:
```
from python_codes.evaluation import fitness_fx
from python_codes.particle import Particle
```

In [3]:
```
(b_lo, b_up) = (-1, 2)
n_dimensions = 5
n_particles = 100
swam_size = 2
n_iters = 500
g = None
omega = 1
phi_p = 0.5
phi_g = 0.5
```

In [4]:
```
particle_pop = []
for i in range(n_particles):
    part_velocity = uniform(-absolute(b_lo - b_up), absolute(b_lo - b_up), size=n
    part_position = uniform(low=b_lo, high=b_up, size=n_dimensions)
    best_position = part_position

    if g is not None:
        if fitness_fx(best_position).sum() > fitness_fx(g).sum():
            g = best_position.copy()
    else:
        g = best_position.copy()

    p = Particle(position=part_position, velocity=part_velocity, best_position=be
    particle_pop.append(p)

for particle in particle_pop:
    neighbors = choice(particle_pop, size=swam_size)
    particle.neighbors_particles = neighbors
```

In [5]:
```
print('Best known position (g) in the initial population:', g)
print('f(g): ', fitness_fx(g).sum())
```

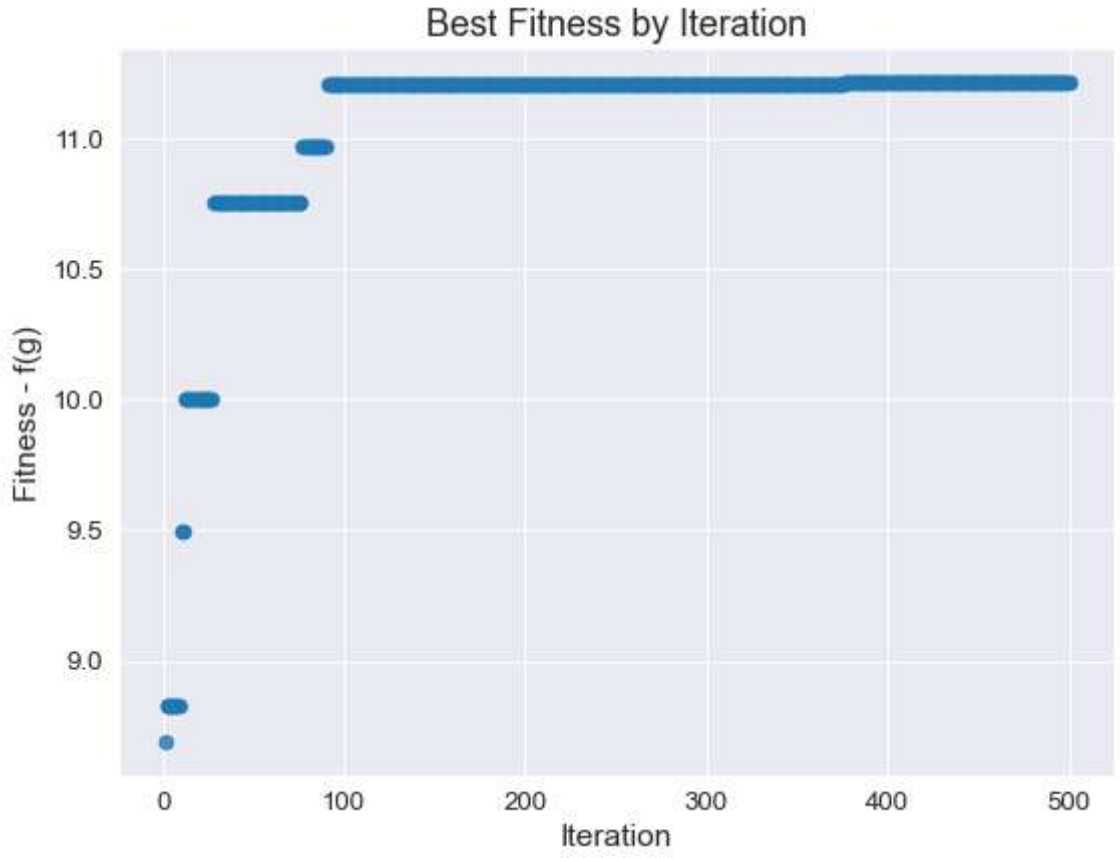
```
Best known position (g) in the initial population: [1.07300646 0.83583854 1.269
45272 0.87787407 1.50307677]
f(g):  8.01607912911538
```

In [6]:
```
best_per_it = []

for it in range(n_iters):
    for particle in particle_pop:

        tmp_vel = particle.velocity.copy()
        tmp_position = particle.position.copy()
        tmp_best_pos = particle.best_position.copy()


        new_velocity = []
        for d in range(n_dimensions):
            r_p, r_g = uniform(), uniform()
            v_id = omega * tmp_vel[d] + phi_p * r_p * (
                tmp_best_pos[d] - tmp_position[d]) + phi_g * r_g * (g[d] - tmp_pc
            new_velocity.append(v_id)
        particle.velocity = np.array(new_velocity)


        tmp_position += new_velocity

        tmp_position = [x if (-1 <= x <= 2) else uniform(low=b_lo, high=b_up) for
        particle.position = np.array(tmp_position)

        if fitness_fx(particle.position).sum() > fitness_fx(particle.best_positic

            particle.best_position = particle.position

            if fitness_fx(particle.best_position).sum() > fitness_fx(g).sum():

                g = particle.best_position

    best_per_it.append(g)
```

In [7]:
```
print('Best known position (g) after 500 iterations:', g)
print('f(g): ', fitness_fx(g).sum())
```

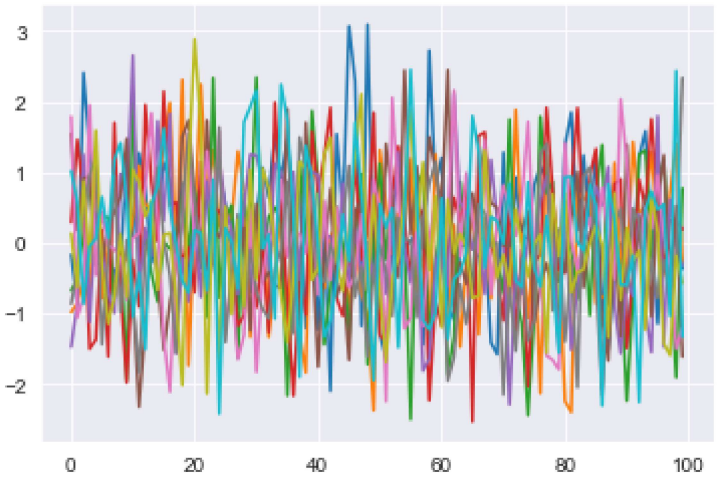
```
Best known position (g) after 500 iterations: [ 1.82418091 -0.64869243  1.86421
376  1.8457774  -0.86204845]
f(g):  11.21643200347268
```

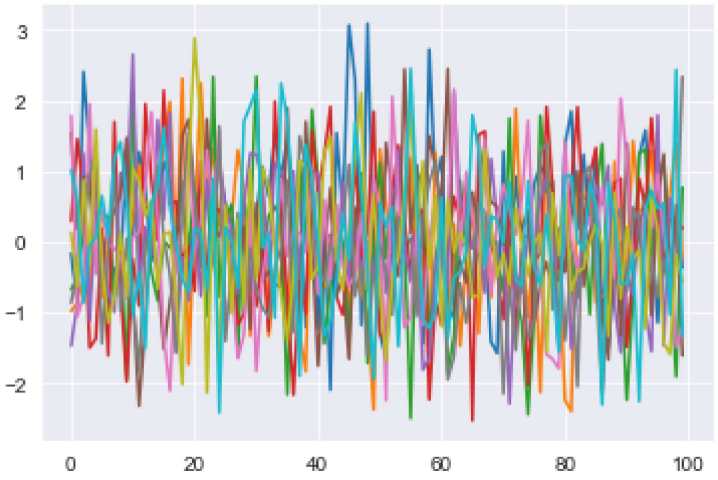
In [8]:
```
sns.set_style('darkgrid')

plt.figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')

y = np.array([fitness_fx(x).sum() for x in best_per_it])
x = np.array(range(1, len(y)+1))

sns.regplot(x=x, y=y, fit_reg=False)
plt.title('Best Fitness by Iteration', fontsize=16)
plt.xlabel('Iteration', fontsize=14)
plt.ylabel('Fitness - f(g)', fontsize=14)
plt.yticks(fontsize=12)
plt.xticks(fontsize=12)
plt.show()
```



WIP

In [9]:
```
%matplotlib inline
import time
import pylab as pl
from IPython import display

for i in range(10):
    pl.plot(pl.randn(100))
    display.clear_output(wait=True)
    display.display(pl.gcf())
    time.sleep(1.0)
```





In [ ]:
```

```