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]:
```
# Codes Implemented
from python_codes.evaluation import fitness_fx
from python_codes.particle import Particle
```

In [3]:
```
# Algorithm parameters
(b_lo, b_up) = (-1, 2)  # Lower and up boundaries
n_dimensions = 5        # Number of genes per particle
n_particles = 100       # Number of particles in population
swam_size = 2           # Number of neighbors for each particle (swarm)
n_iters = 500           # Number of iterations (criterion)
g = None                # Best known position (vector)
omega = 1               # Omega contant
phi_p = 0.5             # Phi p constant
phi_g = 0.5             # Phi g constant
```

Next we generate the initial population:

In [4]:
```
# Generate particle population
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)

# Define the swarm of each particle
for particle in particle_pop:
    neighbors = choice(particle_pop, size=swam_size)
    particle.neighbors_particles = neighbors
```

In [10]:
```
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: [1.66226588 1.63457626 1.825
89829 1.87030108 0.86681045]
f(g):  11.565157057023557
```

Now that we've a initial population, the algorithm its execute. That algorithm its based on the pseudocode, shown in the beginning of the notebook:

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

for it in range(n_iters):
    for particle in particle_pop:
        # Get the current values
        tmp_vel = particle.velocity.copy()
        tmp_position = particle.position.copy()
        tmp_best_pos = particle.best_position.copy()

        # Update particle velociy
        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_po
            new_velocity.append(v_id)
        particle.velocity = np.array(new_velocity)

        # Update particle position
        tmp_position += new_velocity
        # If any dimension overcome the limits,
        # other values in the limits its generate for this dimension.
        tmp_position = [x if (-1 <= x <= 2) else uniform(low=b_lo, high=b_up) for
        particle.position = np.array(tmp_position)

        # Update best positions
        if fitness_fx(particle.position).sum() > fitness_fx(particle.best_positic
            # Update the particles best known position
            particle.best_position = particle.position

            if fitness_fx(particle.best_position).sum() > fitness_fx(g).sum():
                # Update the swarms best known position
                g = particle.best_position

    best_per_it.append(g)
```

In [11]:
```
print('Best known position (g) after 500 iterations:', g)
print('f(g): ', fitness_fx(g).sum())
```

```
Best known position (g) after 500 iterations: [1.66226588 1.63457626 1.82589829
1.87030108 0.86681045]
f(g):  11.565157057023557
```

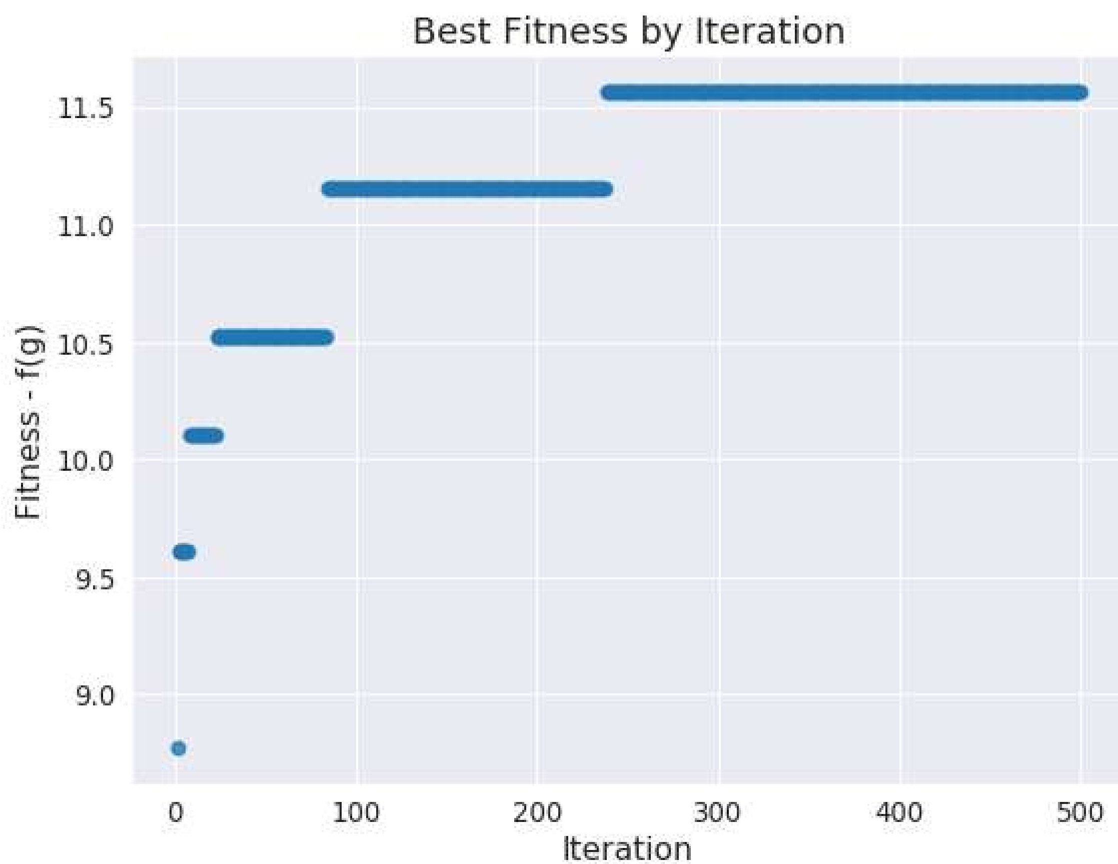
Next, we show the results by iteration. We can notice that the g position merge to a local minima.

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 [ ]:
```
# WIP: function to show points moving in each iteration...
%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)
```