# Further development of Particle Swarm Optimization method applied to accelerators

Ilyas Fatkhullin

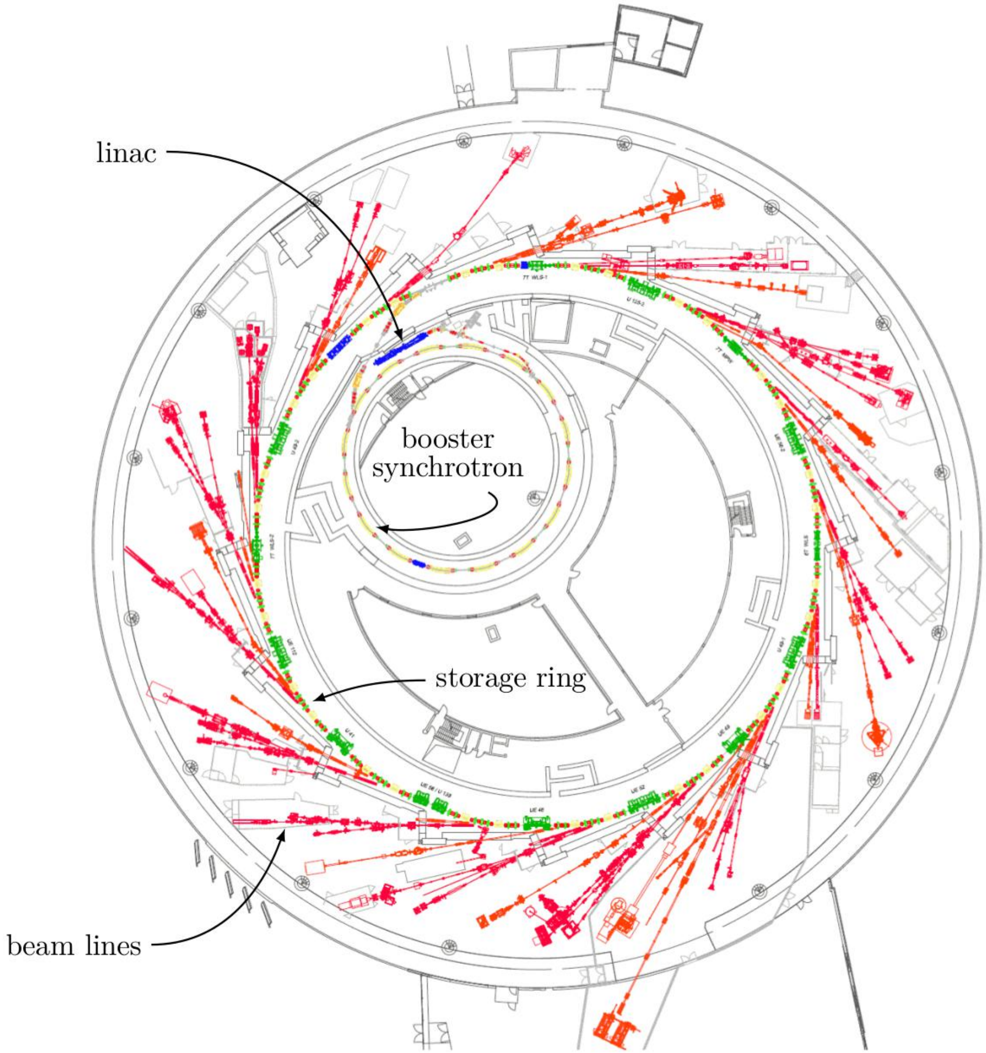
Supervisor: Dr. Ji Li

FG-IA

Helmholtz Zentrum Berlin, Germany

# OUTLINE

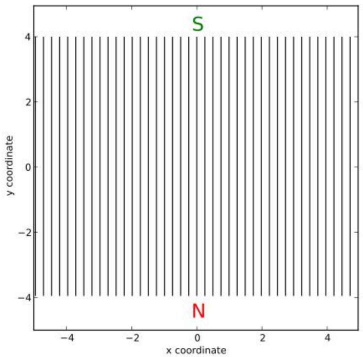
- Background and motivation
- Introduction of Particle Swarm Optimization (PSO)
- Improvements applied to PSO
- Experimental and simulation results
- Multi-objective PSO


linac
booster
synchrotron
storage ring
beam lines

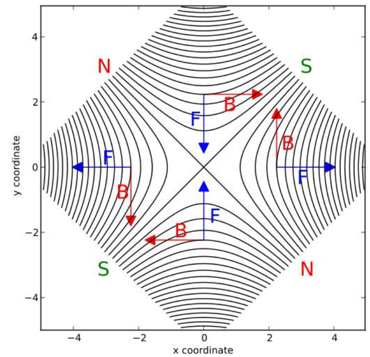
# MAGNETS IN STORAGE RING

Field lines of an idealized
**dipole**, **quadrupole** and **sextupole** magnet
in the plane transverse to the beam direction

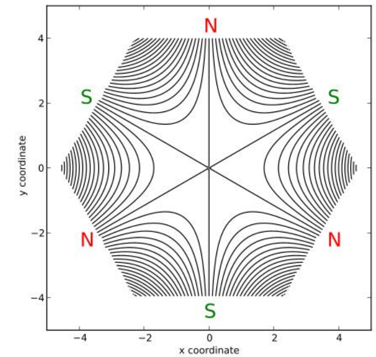
dipole

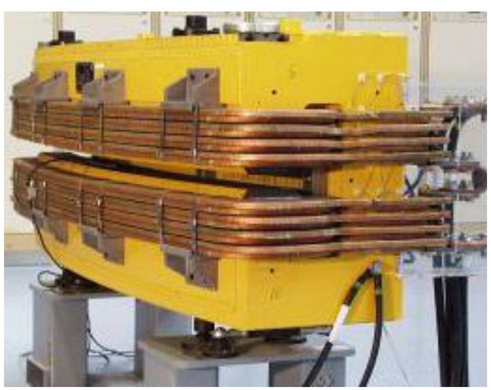


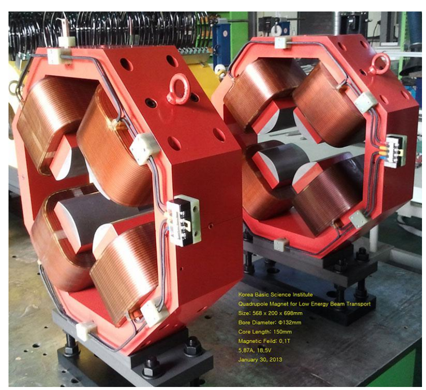

quadrupole

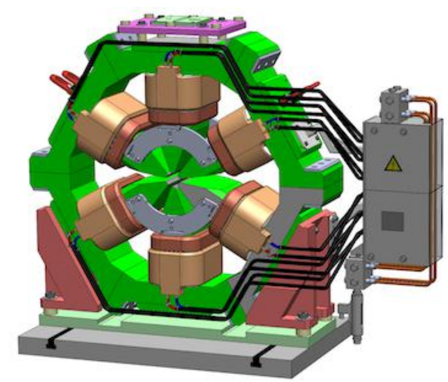



sextupole







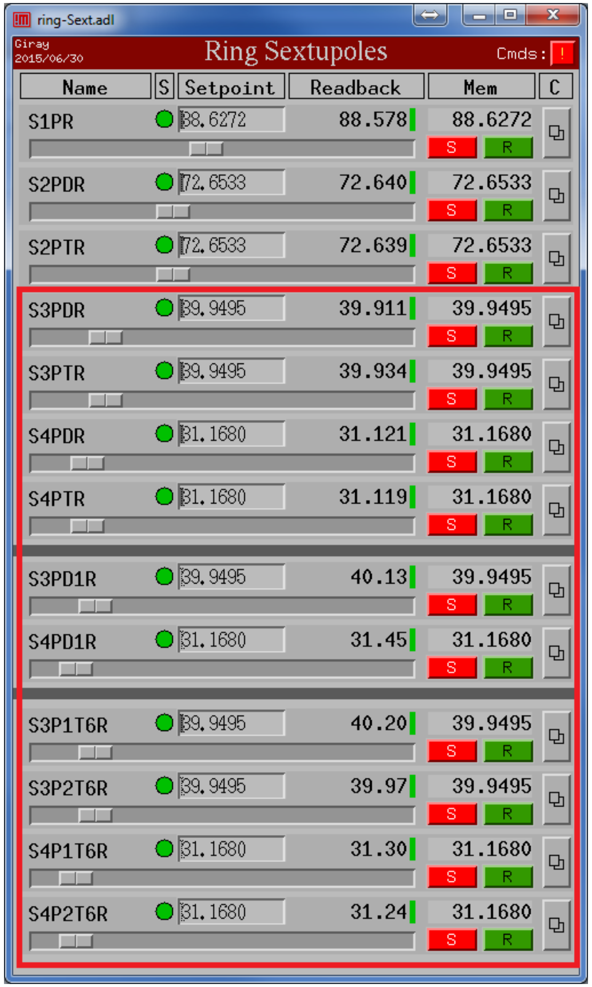
google.com

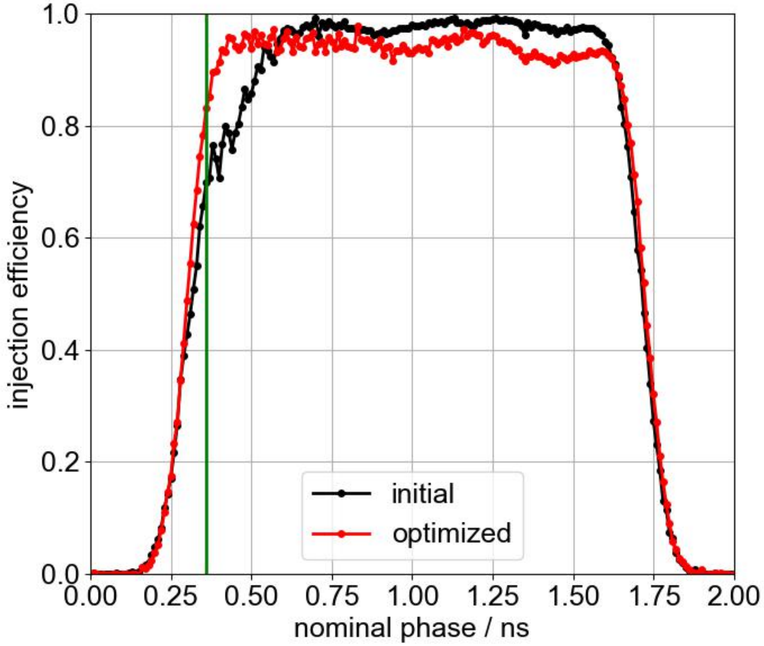
1)

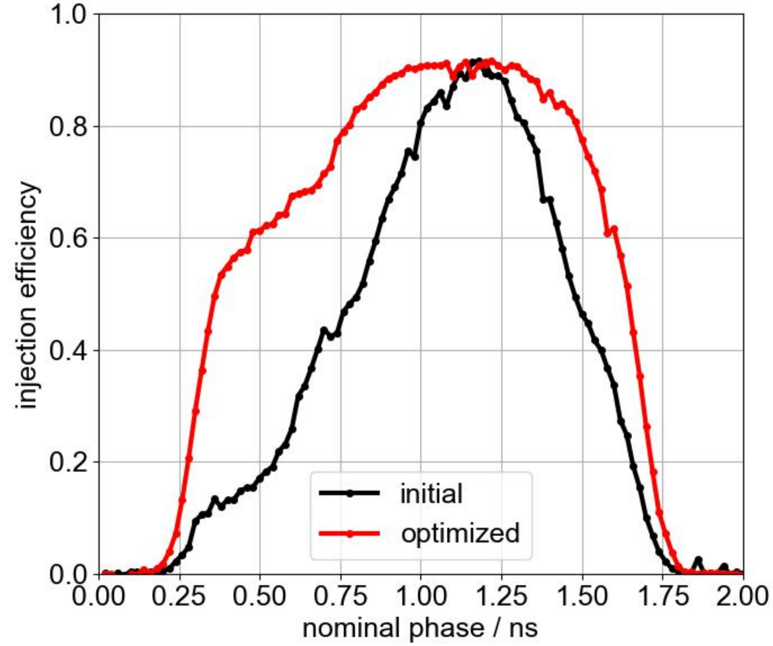
$$\text{Injection efficiency} = \frac{\text{Beam current injected to storage ring}}{\text{Beam current from injector}}$$

2) $$I = I_0 \cdot e^{-\frac{t}{\tau}}$$

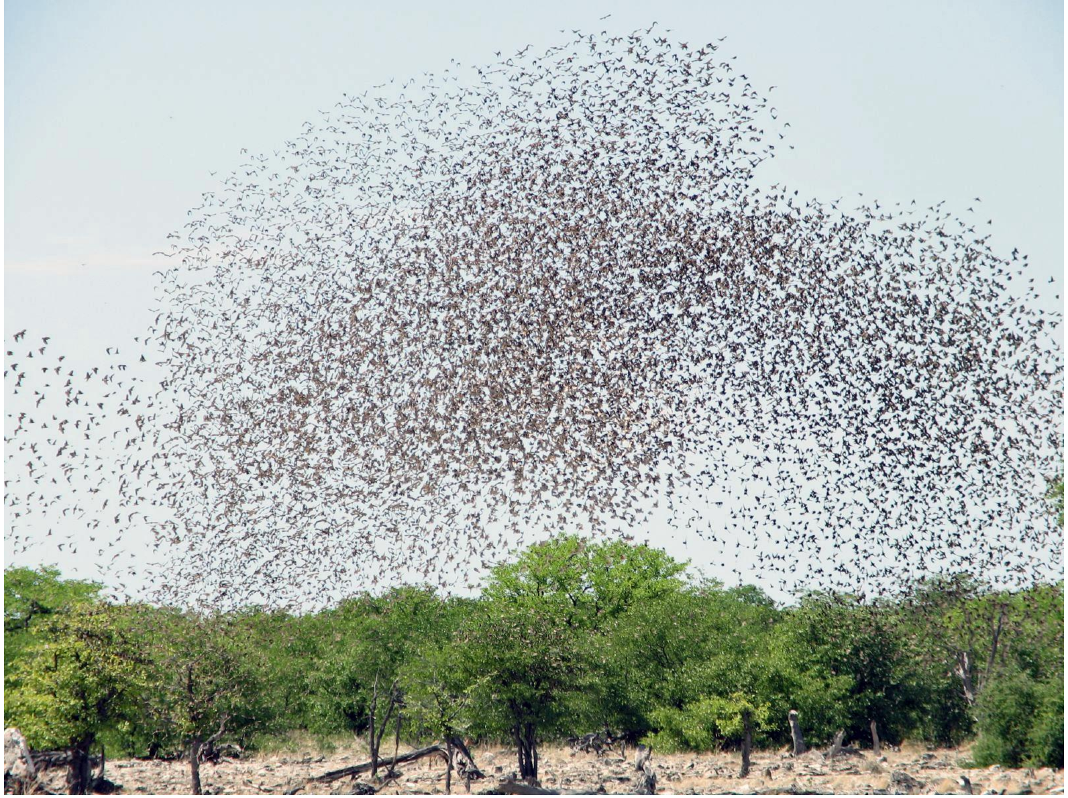
lifetime






- optimization of 10 sextupoles
- insertion devices deteriorate injection efficiency



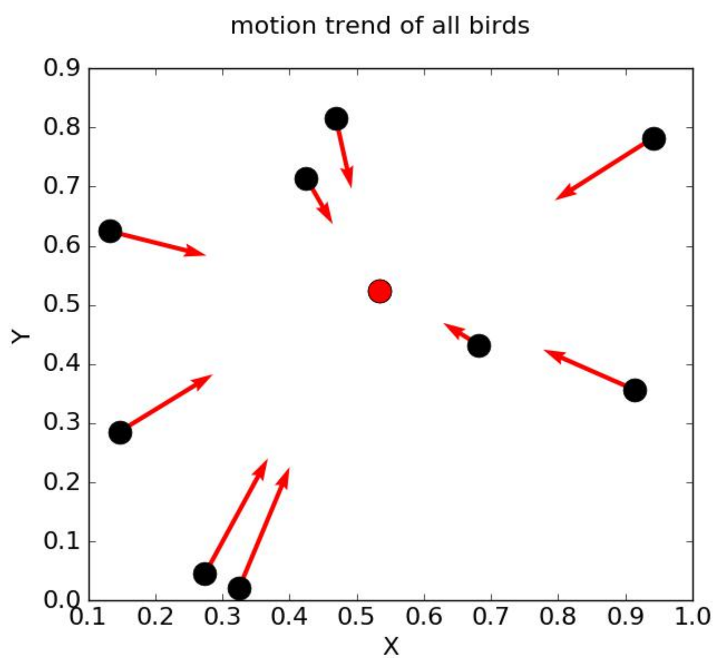
Birds in a flock searching for food

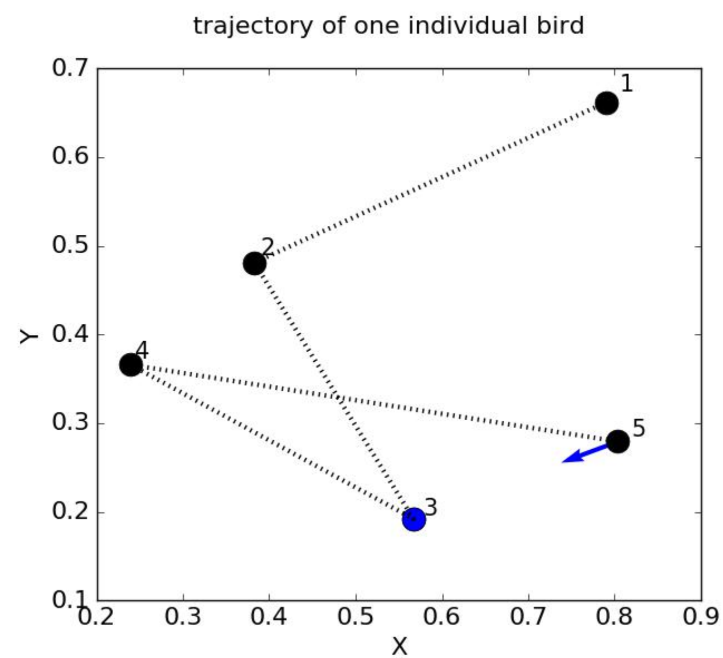
google.com

Developed by Dr. Eberhart and Dr. Kennedy in 1995

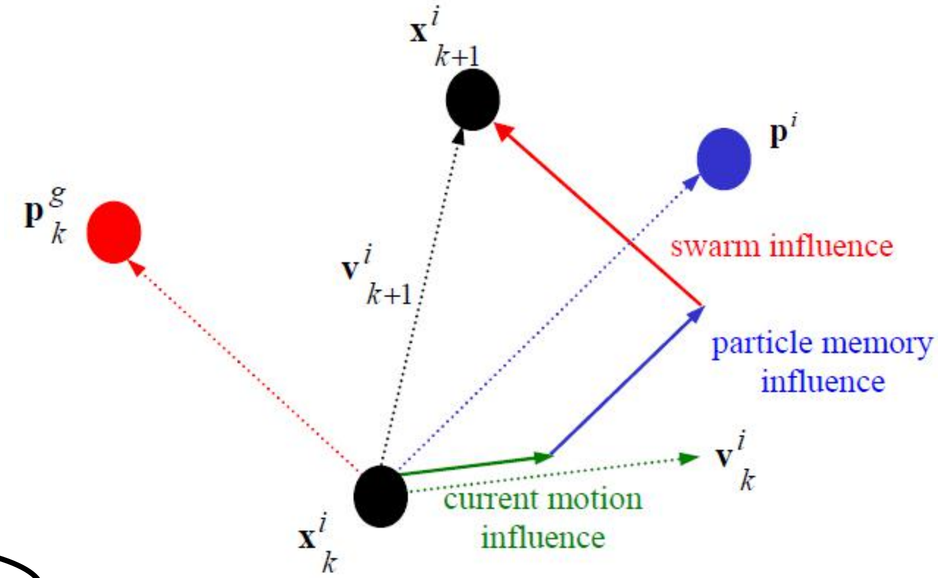
SCENARIO:

- A group of flying **black** birds searching the only one piece of food in a certain space
- No bird knows where the food is
- The bird closest to the food turns red and can be recognized (gbest)
- The birds have memory, every bird marks the best position in its flightpath with blue color (pbest)
- All birds follow the red bird and are influenced by their blue positions until the food is found



# MAIN FORMULA OF PSO




inertia term

cognitive term

$$v_{k+1} = \boxed{w \cdot v_k} + \boxed{c_1 \cdot rand() \cdot (pbest - x_k)} + \boxed{c_2 \cdot rand() \cdot (gbest - x_k)} \quad \text{(a)} \quad \text{(velocity update)}$$

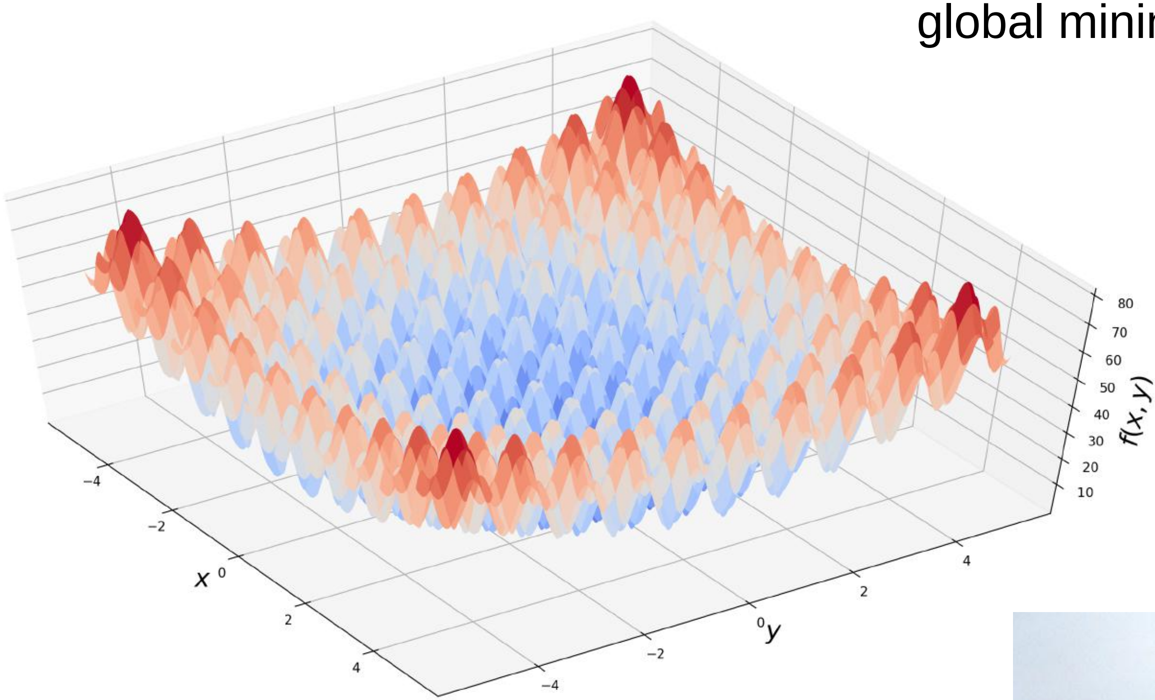
$$x_{k+1} = x_k + v_{k+1} \quad \text{(b)} \quad \text{(position update)}$$

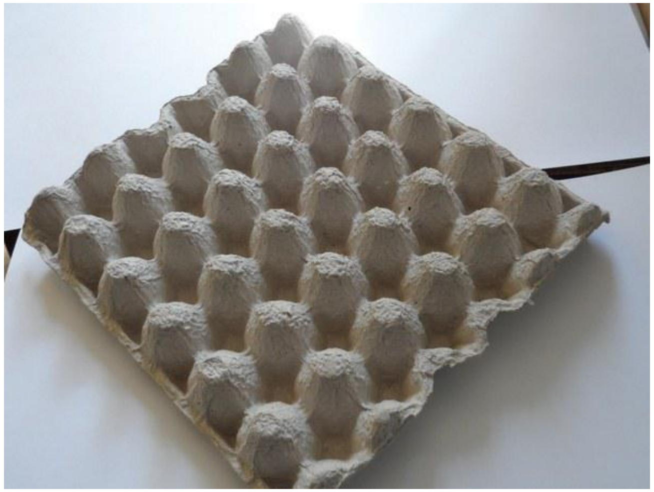
social learning term

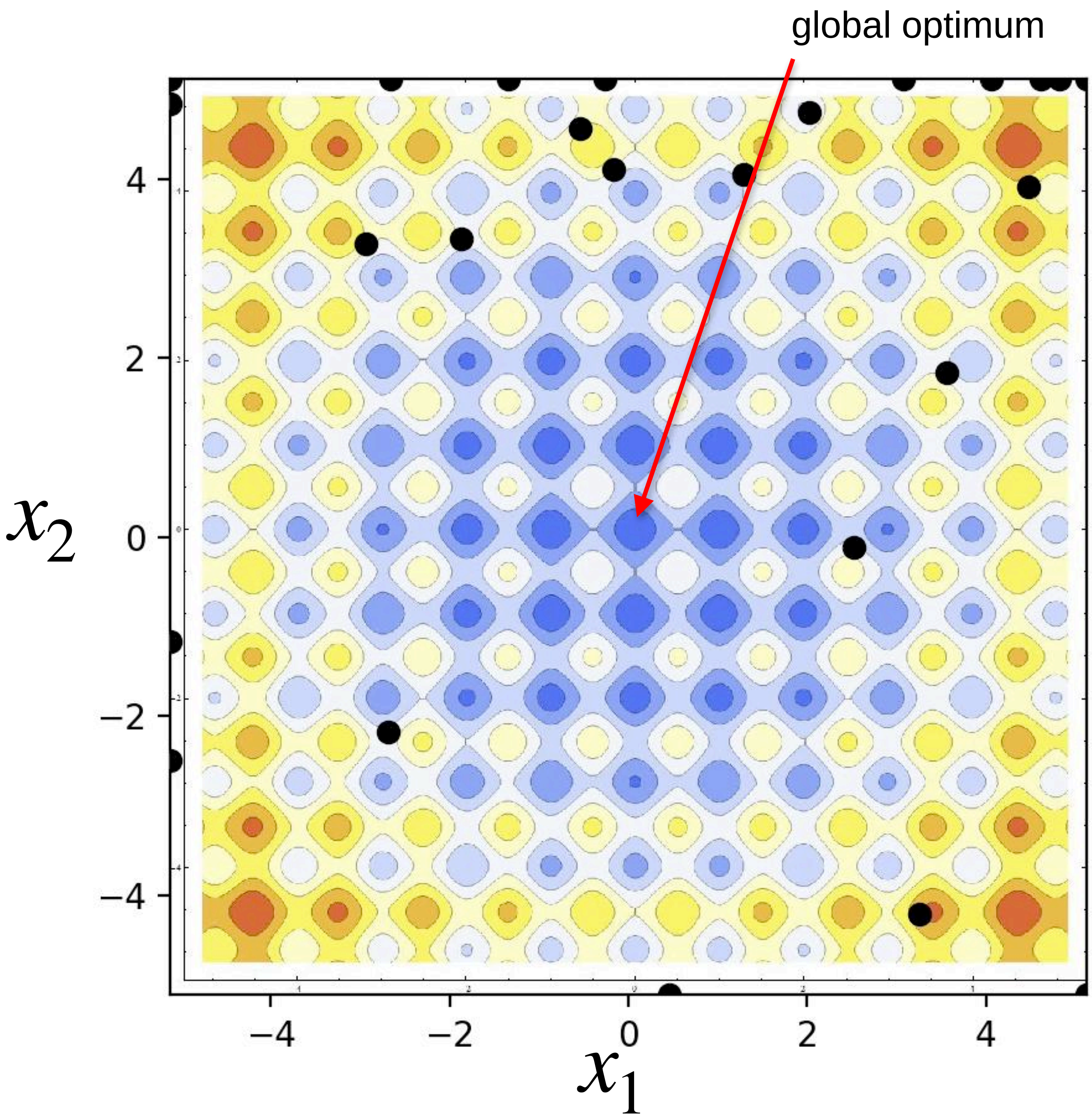
# TEST FUNCTION: RASTRIGIN

global minimum: $x_1, \ldots, x_n = 0,\ldots,0$

$$f(0,\ldots,0) = 0$$



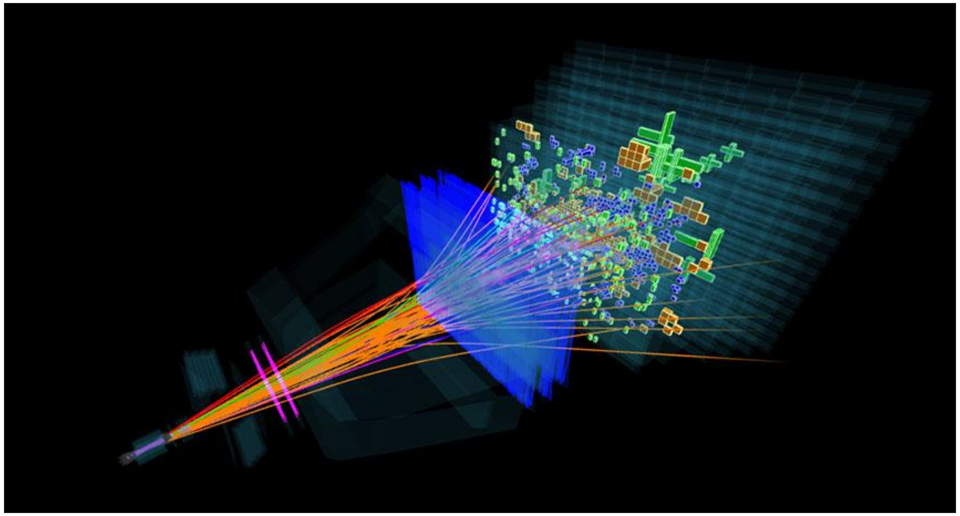




global optimum
4
2
0
−2
−4
$x_2$
−4
−2
0
2
4
$x_1$

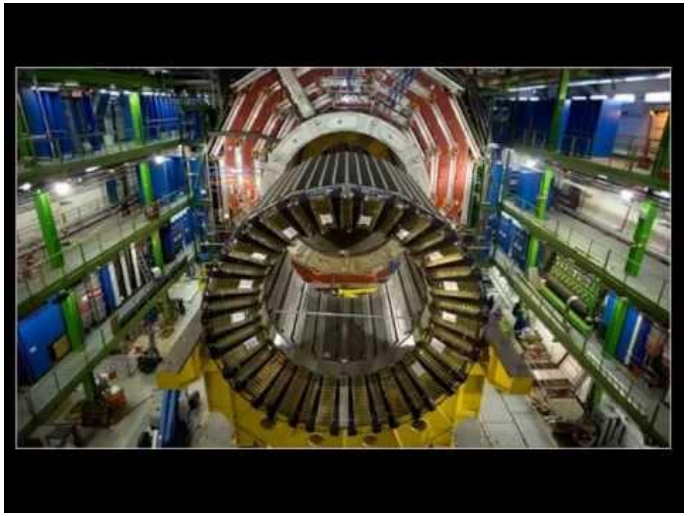
google.com

# EXPECTATIONS









google.com

# REALITY

HZB Helmholtz Zentrum Berlin

pso.py | pso_animate.py | functions.py

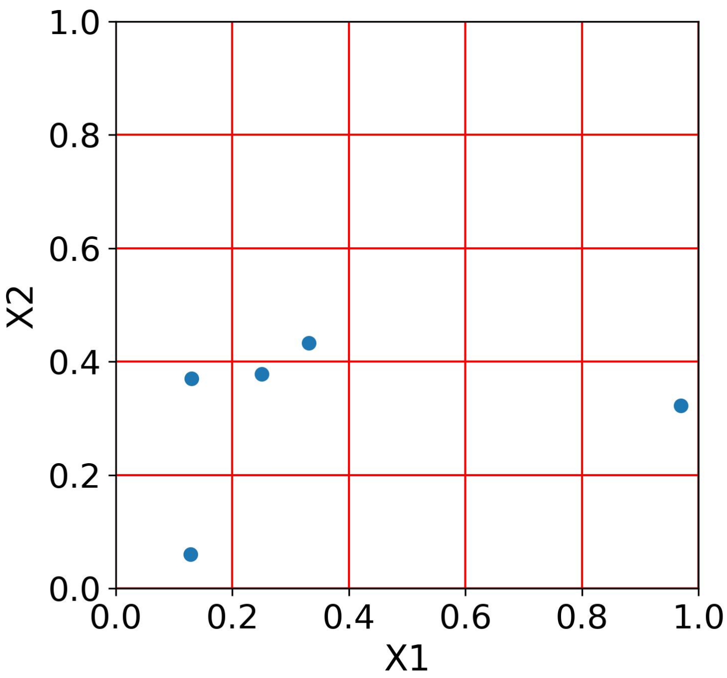
```python
#
class PSO:

    def __init__(self, func_name, lb, ub, Nobj = 1, nRep = 100, bait = np.array(
    def lhs(self):▭

    def _obj_wrapper(self, func, args, kwargs, x):
        return func(x, *args, **kwargs)

    def _is_feasible_wrapper(self, func, x):
        return np.all(self.func(x)>=0)

    def _cons_none_wrapper(self, x):
        return np.array([0])

    def _cons_ieqcons_wrapper(self, x):
        return np.array([y(x, *self.args, **self.kwargs) for y in self.ieqcons])

    def _cons_f_ieqcons_wrapper(self, x):
        return np.array(self.f_ieqcons(x, *self.args, **self.kwargs))

    def Mutate(self, x, pm, mu, lb, ub):▭

    def Mutate_many_dimensions(self, x, pm, mu, lb, ub):▭

    def Dominates(self, X, Y):▭

    def nondomSolutions(self, X, F):▭

    def inverse_permutation_crowding(self, p):
        #This function computes an "inverse permutation"
        return np.array([p.index(l) for l in range(len(p))])

    def crowding_sorting(self, archiveX, archiveF):
        nondomN = len(archiveF)
        crowdDist = np.zeros((nondomN, ))
        for i in range(self.Nobj):
            indexes = np.argsort(archiveF[:,i])
            archivetestF = archiveF[indexes, i]
            crowdDist = crowdDist[indexes] #(*)
```

pso.py 427:14

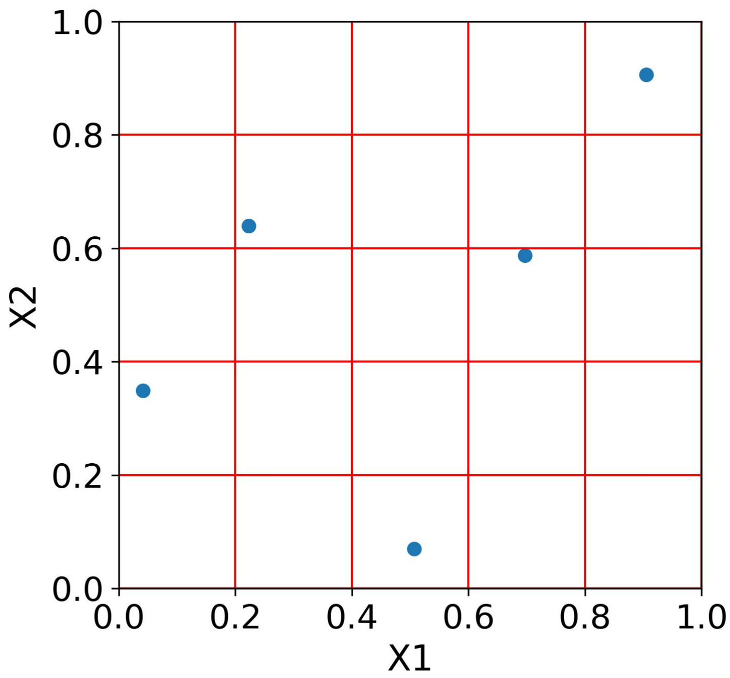
```python
    for c in x:
        firstSum += c**2.0
        secondSum += np.cos(2.0*math.pi*c)
    return -20.0*np.exp(-0.2*np.sqrt(firstSum/n)) - np.exp(secondSum/n) + 20 + m
def Sphere(x):
    Sum = 0.0
    n = float(len(x))
    for c in x:
        Sum += c**2
    return Sum
def Rastrigin(x, A = 10):
    n = float(len(x))
    Sum = 0.0
    for c in x:
        Sum += c**2 - A * np.cos(2 * math.pi * c)
    return A*n + Sum
def Holder(x):
    return 19.2085 - np.abs(np.sin(x[0]) * np.cos(x[1]) * np.exp(abs(1 - np.sqrt

def Rosenbrock(x):
    Sum = 0.0
    for i in range(len(x)-1):
        Sum += 100.0*(x[i+1]-x[i]**2)**2 + (1-x[i])**2
    return Sum

def ZDT1(X):
    #:::: ZDT1 ::::#
    n = X.s
    G  = 1.
    F1 = X[
    F2 = G*
    F = np.
    return

def Viennet
    F1 = 0.
    F2 = ((
    F3 = 1/
    F = np.
```

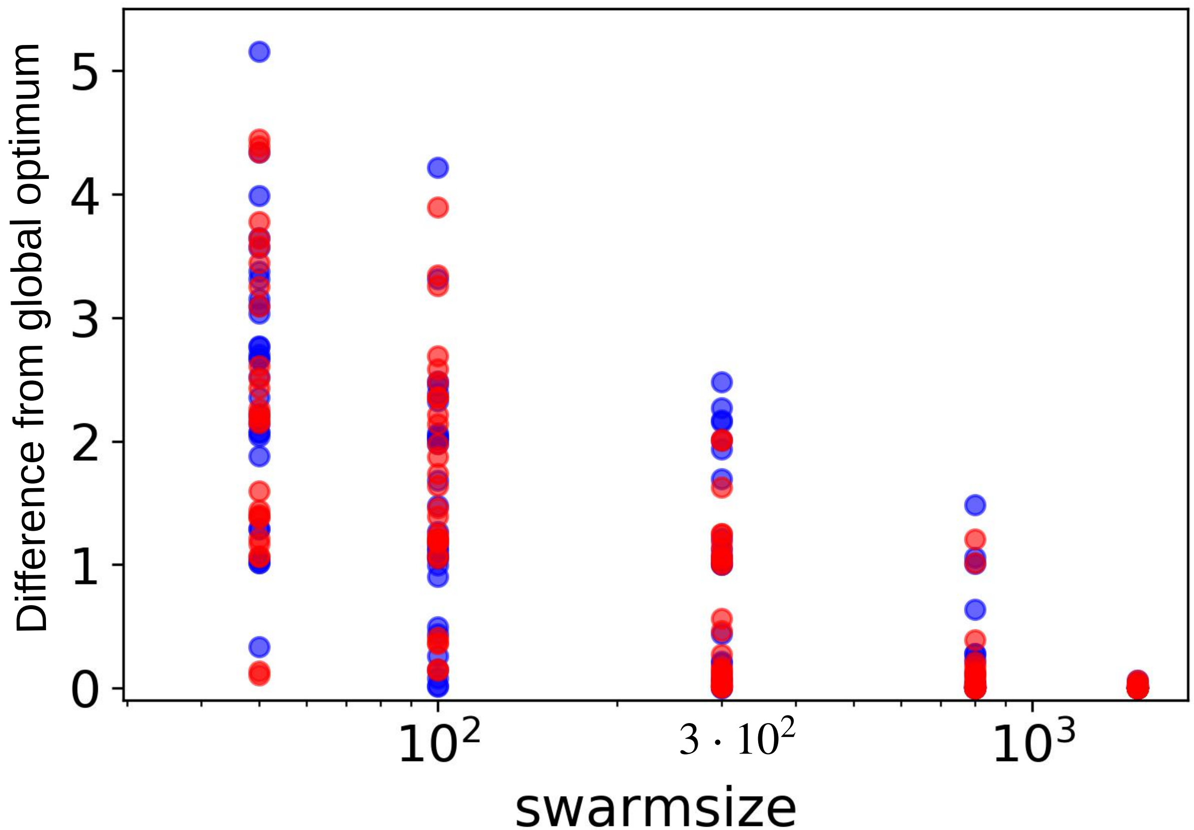
# HOW TO GENERATE STARTING POSITIONS

Random Sampling



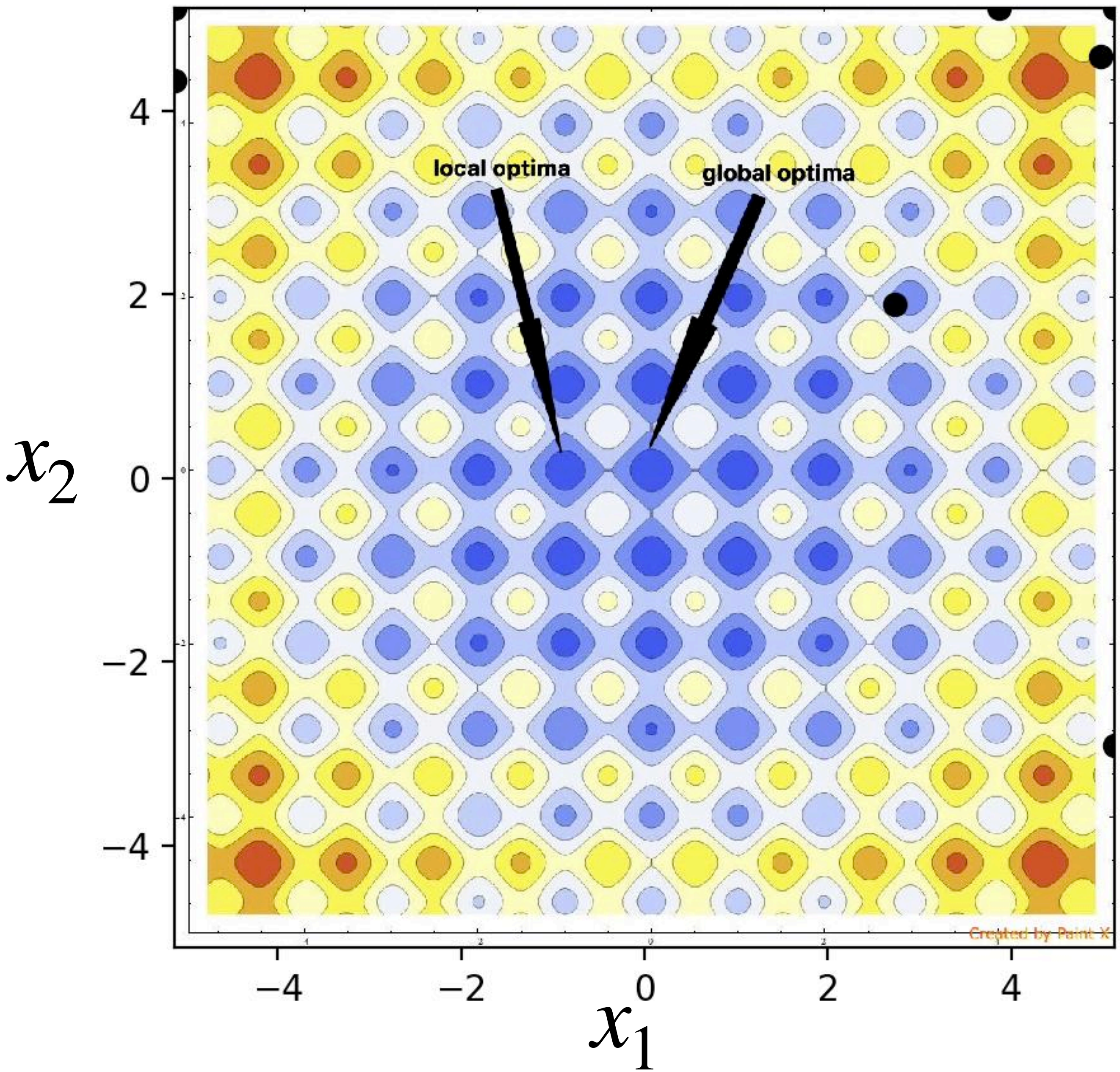

Latin Hypercube Sampling



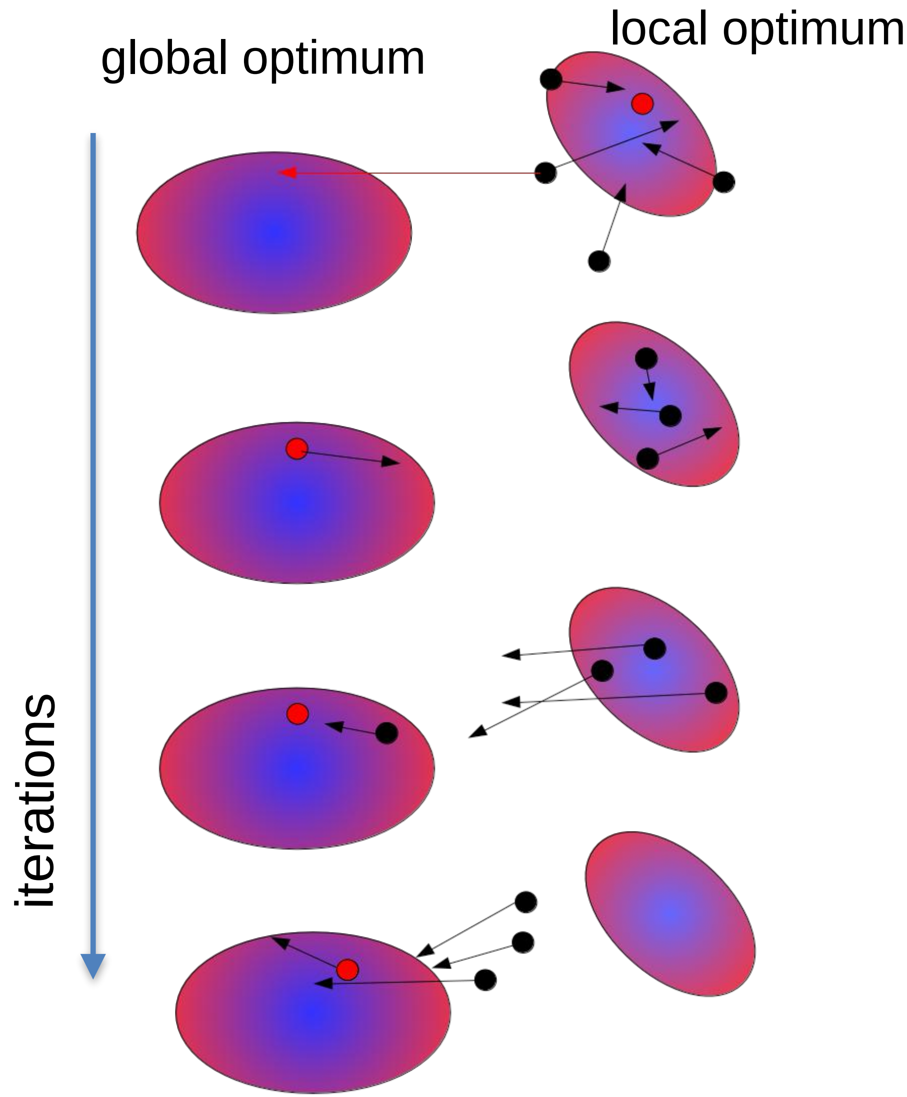


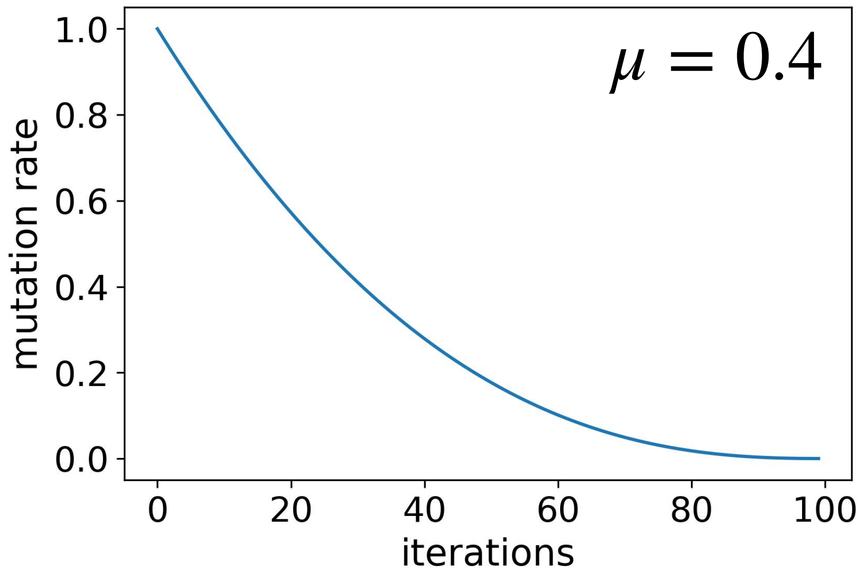

- – random
- – latin hypercube

- function: Rastrigin
- dimensions: 4
- iterations: 70



local optima
global optima
$x_2$
$x_1$
4
2
0
−2
−4
Created by Paint X



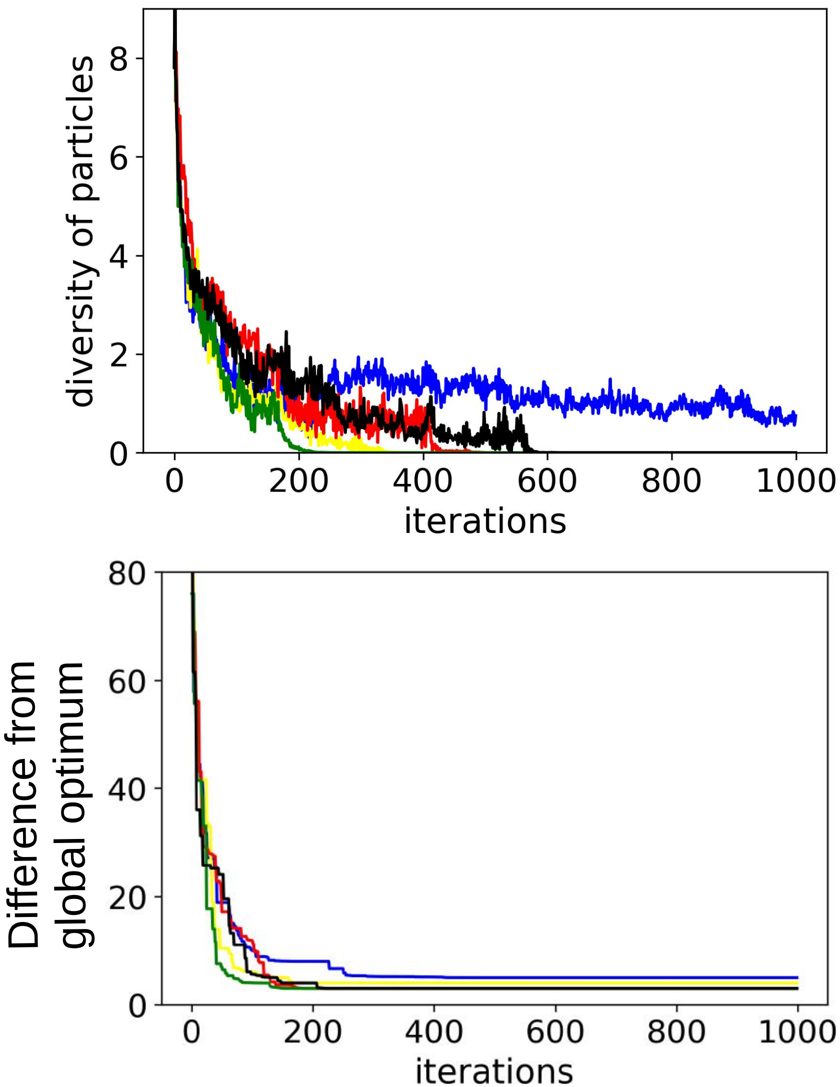

$$\text{Mut. rate} = \left(1 - \frac{n}{N}\right)^{\frac{1}{\mu}}$$

$\mu$ – mutation factor

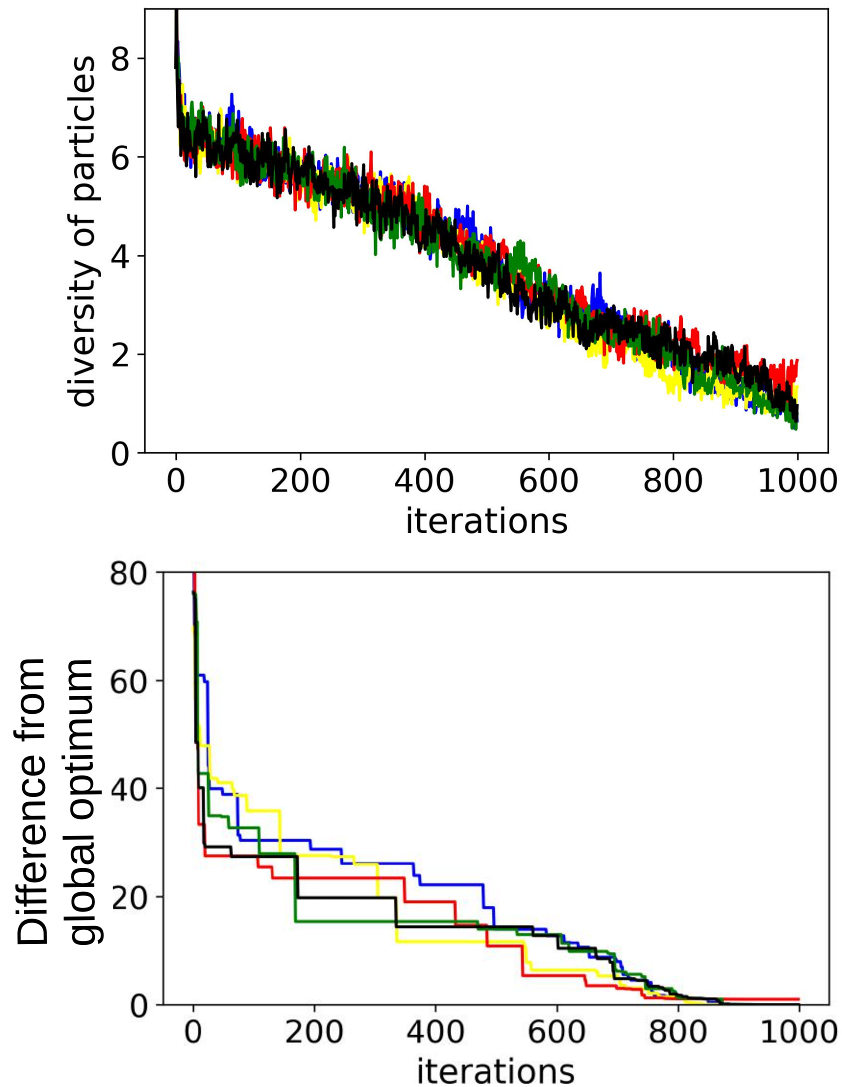
$n$ – iteration number

$N$ – maximum number of iterations

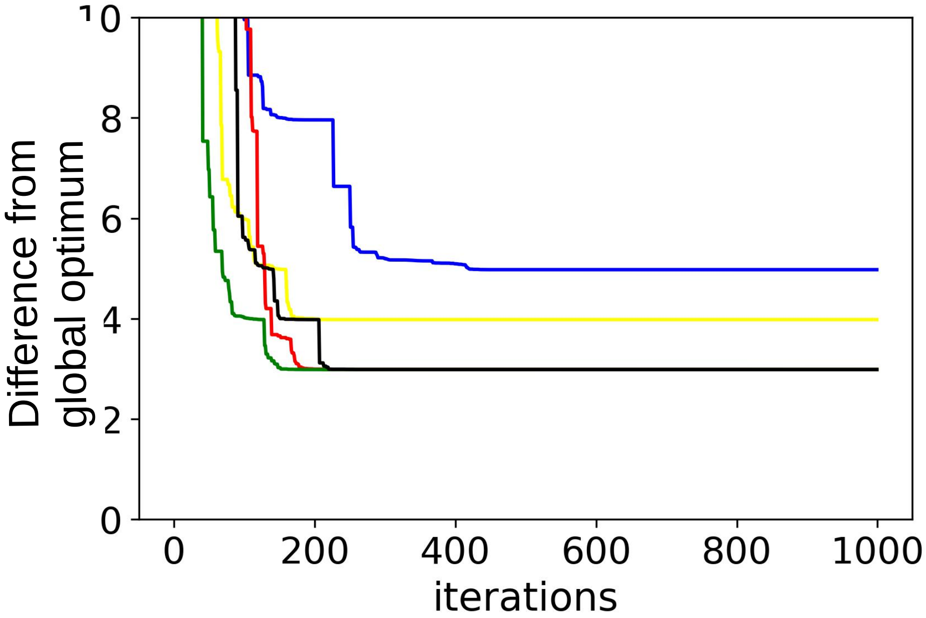
without mutation



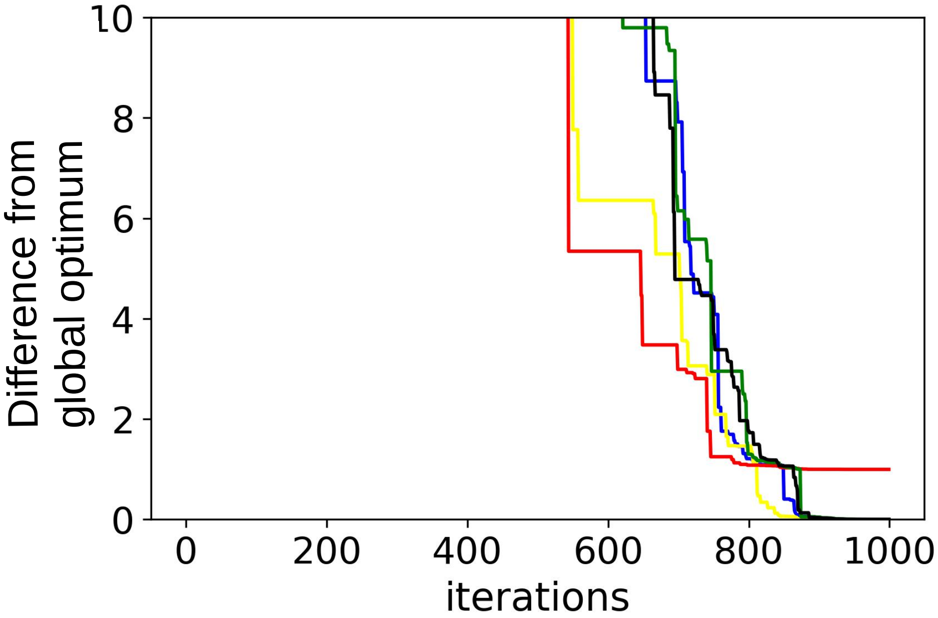

with mutation



without mutation

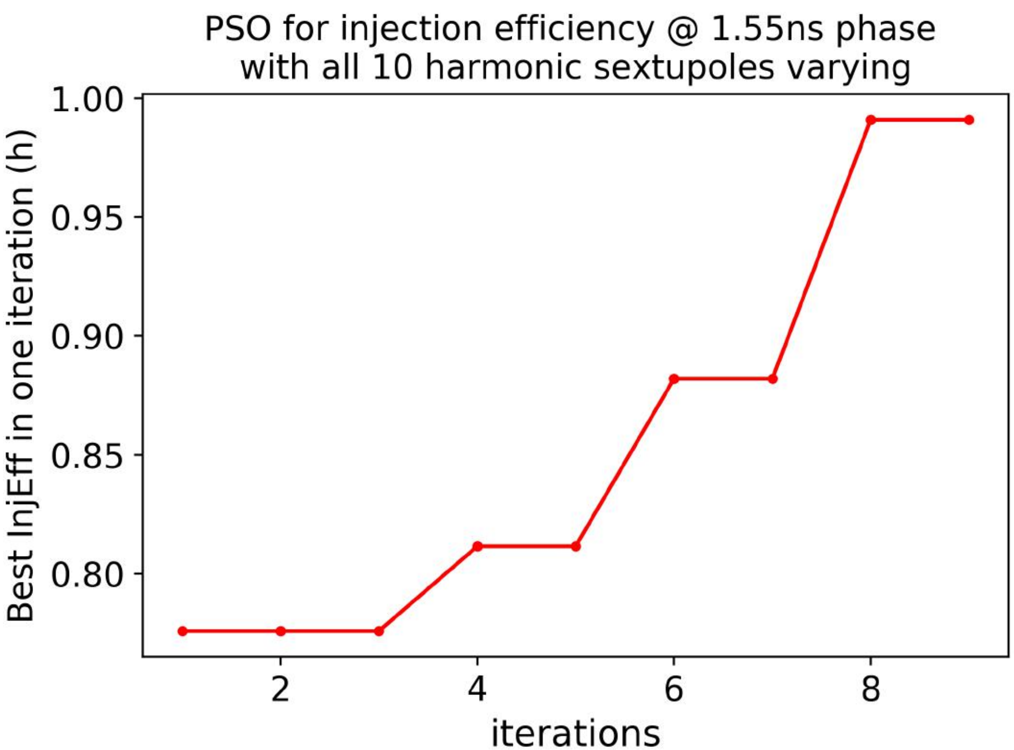


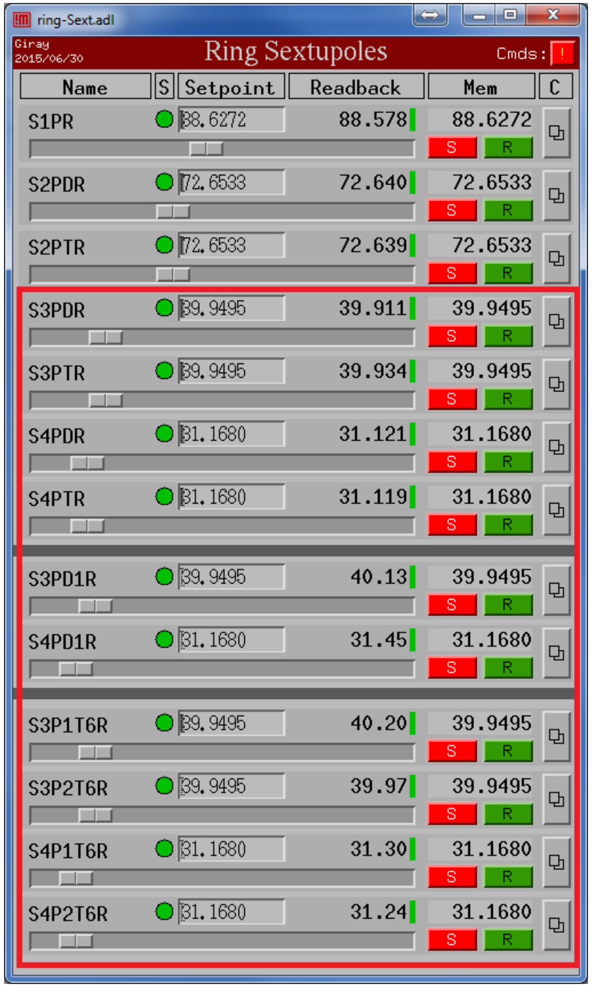

with mutation



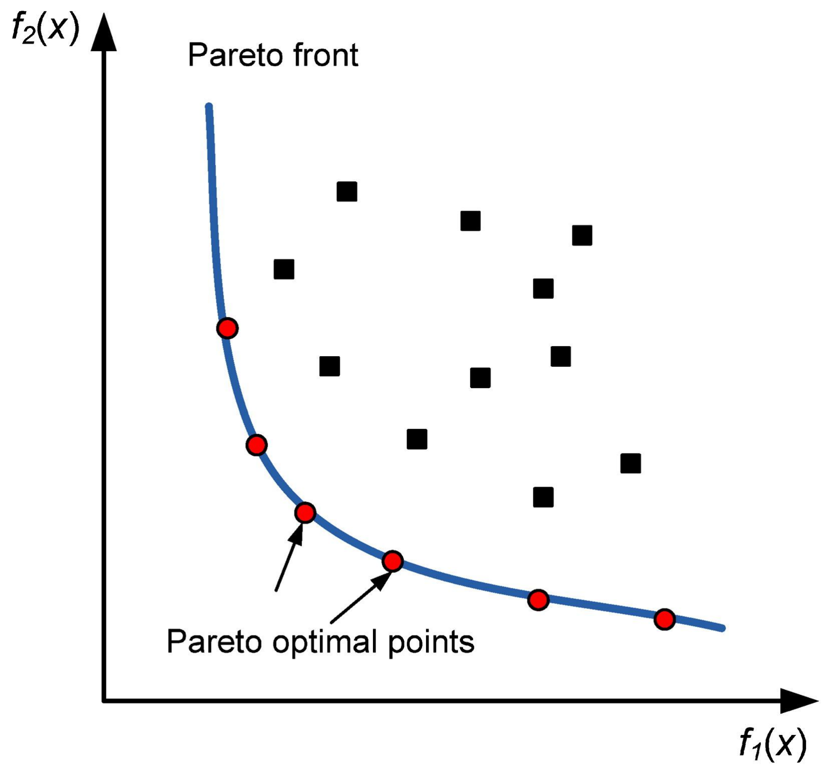

- Dimensions: 7
- Function: Rastrigin
- Swarm size: 30

HZB Helmholtz Zentrum Berlin



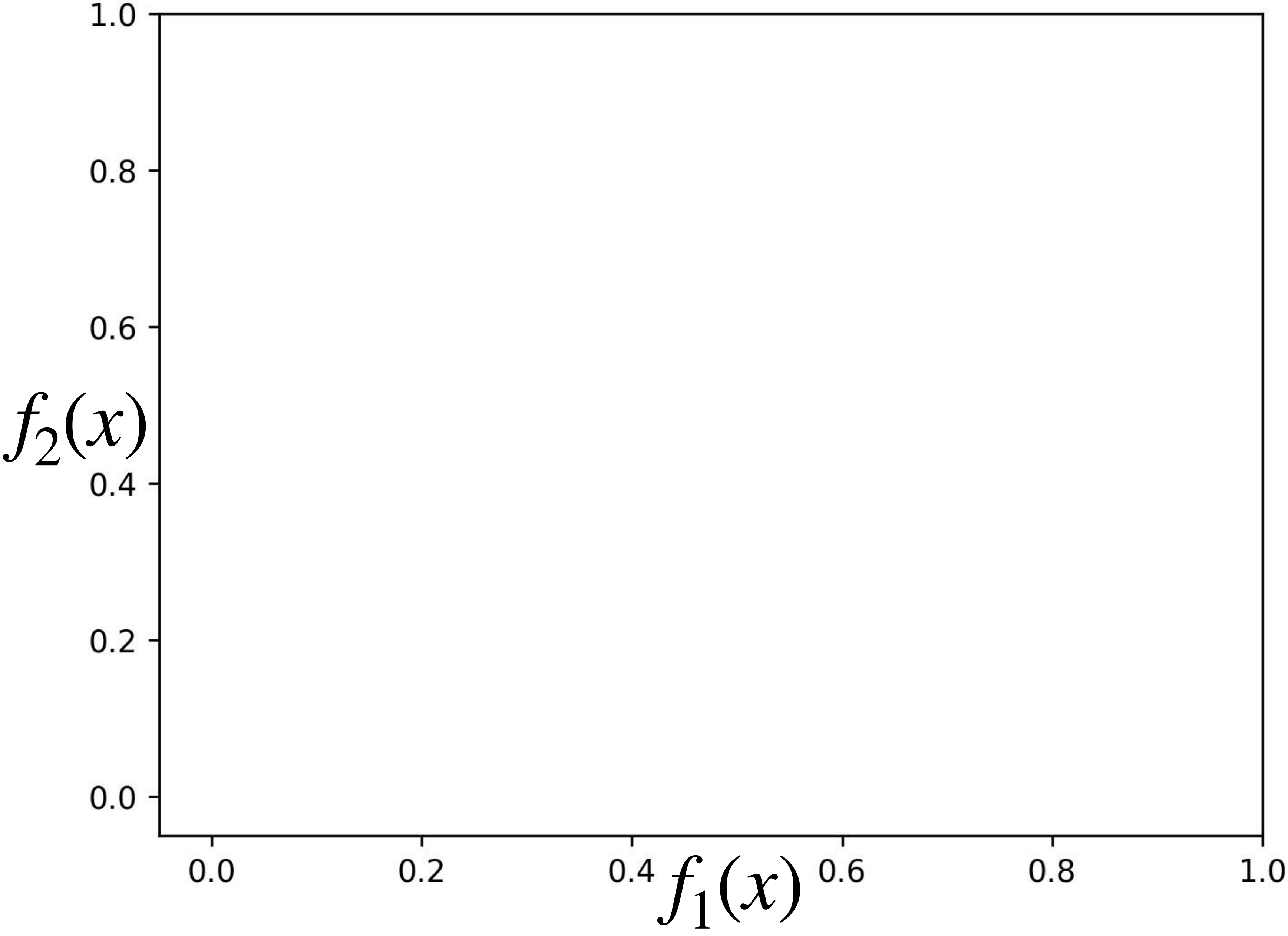
HZB Helmholtz Zentrum Berlin




*def.* ***Pareto front*** *is the best set of solutions*

https://github.com/txt/fss17/blob/master/review2.md

ZDT1 function



- Implementation of LHS and mutation in PSO
- Experimental work at BESSY II
- Debugging Multi-objective PSO

$$\overrightarrow{f_1} = (f_{11}, \ldots, f_{n1})$$

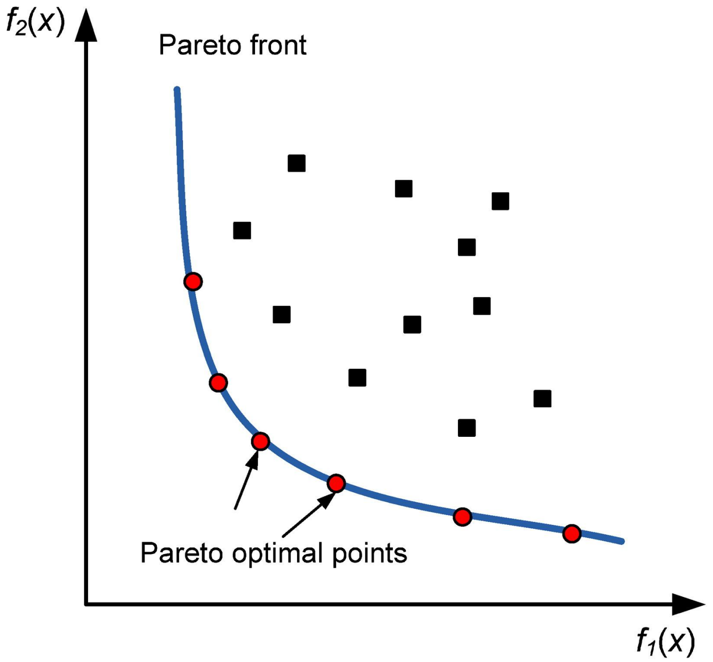
$$\overrightarrow{f_2} = (f_{12}, \ldots, f_{n2})$$

$\overrightarrow{f_1}$ ***dominates*** $\overrightarrow{f_2}$ *if:*

$\forall i \in 1,\ldots,n : f_{1i} \le f_{2i}$ *and* $\overrightarrow{f_1} \neq \overrightarrow{f_2}$

*def. Set is called **non-dominated set** of solutions if no point in the set dominates other points*

*def. **Pareto front** is the best non-dominated set*




Zitzler–Deb–Thiele's function 1