You will be using this package for benchmarking on a set of test problems

Gradient-Free-Optimizers

https://github.com/SimonBlanke/Gradient-Free-Optimizers

In [2]:
```python
#load benchmarks problems

import numpy as np

eqc = np.load('eqcS.npy',allow_pickle=True).item()
class prb:   #this will be your black box function
    def __init__(self,prob,noisy_level):
        # function for generating data
        self.func=prob['func']
        # number of constraints
        self.n_con=prob['n_con']
        # noise level
        self.nl=noisy_level
        # constraints
        self.con=[]
        for i in range(self.n_con):
            self.con.append(prob['con'+str(i+1)])
    def datcol(self,x):
        try:
            # eval takes string type input and evaluates python code. Input can be
            #a mathematical function or a python function which will be called.
            y = eval(self.func)
        except:
            x = np.array([x])
            y = eval(self.func)
        return float(y)
    def constraint(self,x):
        # Appears to evaluate if the constraints are satisfied
        #returns 1 if all constraints are satisfied, else -1
        try:
            m,n=np.shape(x)
        except:
            x = np.array([x])
            m,n=np.shape(x)
        y=np.empty((self.n_con))
        for i in range(self.n_con):
            y[i]=eval(self.con[i])
        if y.all() == 1.:
            label = 1.
        else:
            label = -1.
        return label

bcp=np.load('bcp8.npy',allow_pickle=True).item()

class benchmark:
    def __init__(self,func,noisy_level):
        self.func = str(func)
        self.nl = noisy_level
    def datcol(self,x):
        try:
            y = eval(self.func)
        except:
            x = np.array([x])
            y = eval(self.func)
        return float(y)
```

below are the plots for visualizing 2 variable examples

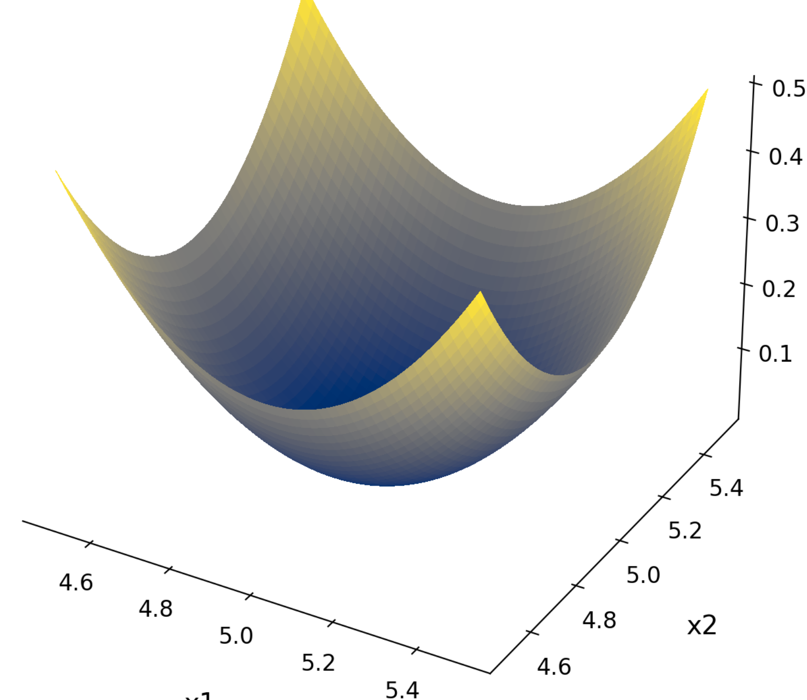
Dimensionality varies from 2-10

use the text files provided to you for categorizing based on dimensionality

You'll be generating performance curves to compare different algorithms, similar to the work in the paper that was sent to you recently by Dr. Boukouvala

In [3]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

fdpi = 200
# bcp contains problems as numpy array - really a dictionary
# params of bcp dictionary
# yopt - optimal value
# xopt - optimal solution
# xlb - lower bound
# xub - upper bound
# n - dimensionality
# func - function to be evaluated - exists as string version of argument where x[:,0] is first variable and x[:
names = list(bcp.keys())


#for i in names:
for i in ['BeckerLago']:
    # n defines dimensionality - ensures we are only plotting 2D problems
    if bcp[i]['n'] == 2:
        prob = bcp[i]
        xlb = prob['xlb']
        xub = prob['xub']
        # pass problem to benchmark class to enable evaluation
        prbf = benchmark(prob['func'],0)
        bounds = np.array([xlb, xub])
        x = np.linspace(bounds[0,0],bounds[1,0],100)
        y = np.linspace(bounds[0,1],bounds[1,1],100)

        X,Y = np.meshgrid(x,y)
        f = prob['func'].replace("x[:,0]", "X").replace("x[:,1]", "Y")
        Z = eval(f)

        fig = plt.figure(figsize=(8,8), dpi = fdpi)
        ax = fig.add_subplot(111, projection='3d')
        # plt.tight_layout(rect=(4, 1,1, 1))
        # Plot a 3D surface
        ax.plot_surface(X, Y, Z, cmap="cividis", antialiased=False)
        ax.set_xlabel('x1',labelpad= 20, fontsize=14)
        ax.set_ylabel('x2',labelpad= 20, fontsize=14)
        ax.set_zlabel('f(x1,x2)',labelpad= 20, fontsize=14)
        ax.set_title(i)
        ax.w_xaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
        ax.w_yaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
        ax.w_zaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
        ax.xaxis.set_tick_params(labelsize=12)
        ax.yaxis.set_tick_params(labelsize=12)
        ax.zaxis.set_tick_params(labelsize=12)
        ax.grid(False)
        plt.show()
```

```
/tmp/ipykernel_105901/53595514.py:47: MatplotlibDeprecationWarning: The w_xaxis attribute was deprecated in Mat
plotlib 3.1 and will be removed in 3.8. Use xaxis instead.
  ax.w_xaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
/tmp/ipykernel_105901/53595514.py:48: MatplotlibDeprecationWarning: The w_yaxis attribute was deprecated in Mat
plotlib 3.1 and will be removed in 3.8. Use yaxis instead.
  ax.w_yaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
/tmp/ipykernel_105901/53595514.py:49: MatplotlibDeprecationWarning: The w_zaxis attribute was deprecated in Mat
plotlib 3.1 and will be removed in 3.8. Use zaxis instead.
  ax.w_zaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
```



General function definition that will be used for rest of semester for solving and evaluating results

In [20]:
```python
# Import all optimizers explicitly into namespace
# exp_opt
from gradient_free_optimizers import EnsembleOptimizer
exp_opt = EnsembleOptimizer
#global_opt
from gradient_free_optimizers import RandomSearchOptimizer,RandomRestartHillClimbingOptimizer,RandomAnnealingOp
global_opt = [RandomSearchOptimizer,RandomRestartHillClimbingOptimizer,RandomAnnealingOptimizer,PatternSearch,P
#local_opt
from gradient_free_optimizers import HillClimbingOptimizer,StochasticHillClimbingOptimizer,RepulsingHillClimbin
local_opt = [HillClimbingOptimizer,StochasticHillClimbingOptimizer,RepulsingHillClimbingOptimizer,SimulatedAnne
#pop_opt
from gradient_free_optimizers import ParallelTemperingOptimizer,ParticleSwarmOptimizer,EvolutionStrategyOptimiz
pop_opt = [ParallelTemperingOptimizer,ParticleSwarmOptimizer,EvolutionStrategyOptimizer,SpiralOptimization]
#smb_opt
from gradient_free_optimizers import ForestOptimizer,BayesianOptimizer,TreeStructuredParzenEstimators,Lipschitz
smb_opt = [ForestOptimizer,BayesianOptimizer,TreeStructuredParzenEstimators,LipschitzOptimizer,DirectAlgorithm]

# Find and replace function for editing function strings
def multiple_replace(_string:str, find_replace:list):
    new_string = _string
    for find,replace in find_replace:
        new_string = new_string.replace(find,replace)
    return new_string

# Create usable form of the function to pass to optimizers
def get_usable_fxn(dimension: int, fxn_form: str):
    find_rep_list = [('x[:,{}]'.format(dimension-1-i),'x["{}"]'.format(dimension-1-i)) for i in range(dimension
    opt_form = multiple_replace(fxn_form, find_rep_list)
    return opt_form

# Create search space
def create_search_space(dimension: int, xlb: list, xub: list, resolution: int = 100):
    search_space = {str(i):np.linspace(xlb[i],xub[i],resolution) for i in range(dimension)}
    return search_space

# Create a container for objective function - need to use eval to evaluate string version of function, but can'
#directly to optimizer - there is probably a better way to do this, possibly using wrappers. Not worth time rig
def create_parent_fxn(search_space,opt_form):
    search_space = search_space
    opt_form = opt_form
    def objective_fxn(search_space):
        #clear_output(wait=True)
        x = search_space
        return -1*eval(opt_form)
    return objective_fxn
```

Simple test of above functions - can easily be wrapped in loops to automate evaluation of all problem and optimizers

- This will serve as the boiler plate code that can easily be extended.
- Only challenge will be optimizers with hyperparameters. Can start by using defaults

In [21]:
```python
test_fxn = bcp['BeckerLago']
fxn_form = test_fxn['func']
xlb = test_fxn['xlb']
xub = test_fxn['xub']
ndim = test_fxn['n']

opt_form = get_usable_fxn(ndim, fxn_form)
search_space = create_search_space(ndim, xlb, xub, resolution=100)
fxn = create_parent_fxn(search_space,opt_form)
# Use first optimizer in list of local optimizers
opt = local_opt[0](search_space)
opt.search(fxn,n_iter=2500)
```

```
Results: 'objective_fxn'
   Best score: -5.1015202530356945e-05
   Best parameter:
      '0' : 5.005050505050505
      '1' : 5.005050505050505

   Random seed: 1068919302

   Evaluation time   : 0.09051942825317383 sec    [26.87 %]
   Optimization time : 0.2463550567626953 sec    [73.13 %]
   Iteration time    : 0.33687448501586914 sec    [7421.16 iter/sec]
```

Becker Lago Problem

- Proof of concept code here: Not pretty or modular
- Evaluate the three criteria from the paper successfully
  - Ability to find global optimum within max(1.01*optimum, 0.01 + optimum)
  - Ability to improve a given starting point (not completely evaluated here - should be a fxn of tau)
  - Ability to improve near optimal solutions: handpicked here, will be defined as solution within 5% of global optimal for generalizability

In [4]:
```python
import numpy as np
from gradient_free_optimizers import RandomSearchOptimizer,HillClimbingOptimizer
from sympy.parsing.sympy_parser import parse_expr
from IPython.display import clear_output
fs = [RandomSearchOptimizer,HillClimbingOptimizer]
# example 2 dim problem to validate optimizer
test_fxn = bcp['BeckerLago']
fxn_form = test_fxn['func'].replace("x[:,0]", "X").replace("x[:,1]", "Y")
xlb = test_fxn['xlb']
xub = test_fxn['xub']

x = np.linspace(xlb[0],xub[0],100)
y = np.linspace(xlb[1],xub[1],100)

# Search space must be passed as a dictionary
search_space = {'X':x, 'Y':y}
optimum = {}

def parent_fxn(search_space):
    clear_output(wait=True)
    X, Y = search_space['X'], search_space['Y']
    # Note, the optimizers maximize the "objective" function, so we need to negate the function to find a minim
    return -1*eval(fxn_form)
for f in fs:
    optimum[f.__name__] = []
    for i in range(10):
        opt = f(search_space) # initialize optimizer
        opt.search(parent_fxn, n_iter=2500) # run optimizer
        #opt.search_data.to_csv('./{}.txt'.format(f.__name__), sep='\t', index=False)
        optimum[f.__name__].append(-1*opt.best_score)

opts = ['Solver\tBest_Optimum\tAverage_Optimum\tSuccess']
for i in optimum.keys():
    optimum[i].sort()
    medians = (np.median(optimum[i]))
    offset = abs(np.array(optimum[i]) - test_fxn['yopt'])
    tmp = min(offset)
    tmp_opt = optimum[i][offset.tolist().index(tmp)]
    success = abs(test_fxn['yopt']) <= max(abs(1.01*tmp_opt),abs(tmp_opt)+0.01)
    opts += ['{}\t{}\t{}\t{}'.format(i,tmp_opt,medians,success)]
with open('BeckerLago.txt', 'w') as f:
    print('\n'.join(opts),file=f)
```

```
Results: 'parent_fxn'
   Best score: -5.1015202530356945e-05
   Best parameter:
      'X' : 5.005050505050505
      'Y' : 5.005050505050505

   Random seed: 1048775505

   Evaluation time   : 0.3661315441131592 sec    [52.61 %]
   Optimization time : 0.3298063278198242 sec    [47.39 %]
   Iteration time    : 0.6959378719329834 sec    [3592.27 iter/sec]
```

Evaluate convergence from a specified starting point

In [26]:
```python
# pass initial guess as a dictionary to warm start optimizer
initial_guess = {'X':5.5, 'Y':5.5}
initialize={"warm_start":[initial_guess]}

optimum = {}
for f in fs:
    opt = f(search_space) # initialize optimizer
    opt.search(parent_fxn, n_iter=2500) # run optimizer
    #opt.search_data.to_csv('./{}.txt'.format(f.__name__), sep='\t', index=False)
    optimum[f.__name__] = -1*opt.best_score

opts = ['Solver\tOptimum']
for i in optimum.keys():
    tmp_opt = optimum[i]
    opts += ['{}\t{}'.format(i,tmp_opt)]
with open('BeckerLagoStartingPoint.txt', 'w') as f:
    print('\n'.join(opts),file=f)
```

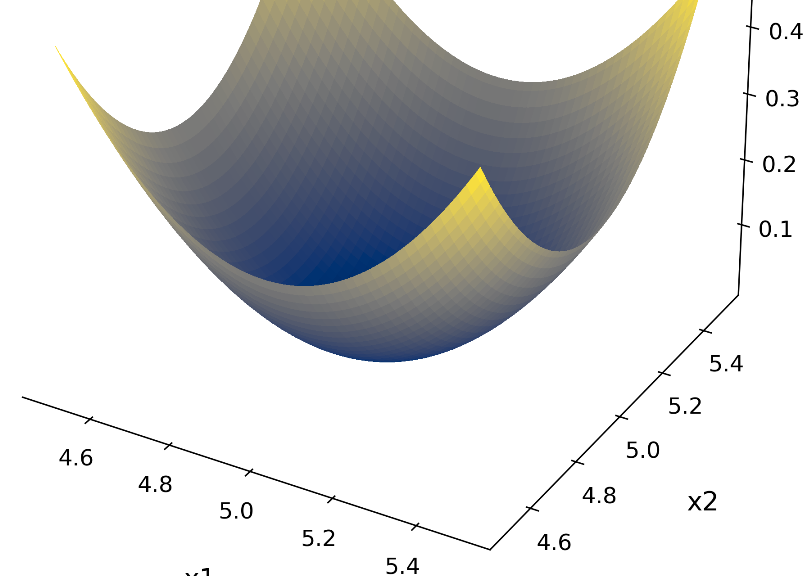
```
Results: 'parent_fxn'
   Best score: -5.1015202530356945e-05
   Best parameter:
      'X' : 5.005050505050505
      'Y' : 4.994949494949495

   Random seed: 1908877436

   Evaluation time   : 0.5716309547424316 sec    [60.93 %]
   Optimization time : 0.3665273189544678 sec    [39.07 %]
   Iteration time    : 0.9381582736968994 sec    [2664.8 iter/sec]
```

In [27]:
```python
# pass initial guess as a dictionary to warm start optimizer
initial_guess = {'X':5.05, 'Y':5.05}
initialize={"warm_start":[initial_guess]}

optimum = {}
for f in fs:
    opt = f(search_space) # initialize optimizer
    opt.search(parent_fxn, n_iter=2500) # run optimizer
    #opt.search_data.to_csv('./{}.txt'.format(f.__name__), sep='\t', index=False)
    optimum[f.__name__] = -1*opt.best_score

opts = ['Solver\tOptimum']
for i in optimum.keys():
    tmp_opt = optimum[i]
    opts += ['{}\t{}'.format(i,tmp_opt)]
with open('BeckerLagoNearOptimal.txt', 'w') as f:
    print('\n'.join(opts),file=f)
```

```
Results: 'parent_fxn'
   Best score: -5.1015202530356945e-05
   Best parameter:
      'X' : 4.994949494949495
      'Y' : 4.994949494949495

   Random seed: 1110658485

   Evaluation time   : 0.543109655380249 sec    [57.43 %]
   Optimization time : 0.4026529788970947 sec    [42.57 %]
   Iteration time    : 0.9457626342773438 sec    [2643.37 iter/sec]
```

In [5]:
```python
#Plot final configuration with all guesses. Highlight optimal guess
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import matplotlib.lines as mlines
import numpy as np

fig,ax = plt.subplots()
```

```
/tmp/ipykernel_105901/902782072.py:16: MatplotlibDeprecationWarning: The w_xaxis attribute was deprecated in Ma
tplotlib 3.1 and will be removed in 3.8. Use xaxis instead.
  ax.w_xaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
/tmp/ipykernel_105901/902782072.py:17: MatplotlibDeprecationWarning: The w_yaxis attribute was deprecated in Ma
tplotlib 3.1 and will be removed in 3.8. Use yaxis instead.
  ax.w_yaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
/tmp/ipykernel_105901/902782072.py:18: MatplotlibDeprecationWarning: The w_zaxis attribute was deprecated in Ma
tplotlib 3.1 and will be removed in 3.8. Use zaxis instead.
  ax.w_zaxis.set_pane_color((1.0, 1.0, 1.0, 1.0))
```



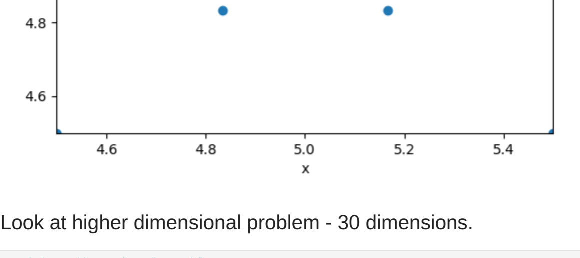
Make neat gif graphic

In [13]:
```python
import matplotlib.pyplot as plt
import numpy as np
import imageio
import  os

#catch and ignore warnings
import warnings
warnings.filterwarnings("ignore")

def generate_plots():
    images = []
    optimal_i = 0
    data = np.genfromtxt('HillClimbingOptimizer.txt', delimiter='\t', skip_header=1)
    for i in range(1, 2501):
        x = data[:i,1]
        y = data[:i,2]
        z = data[:i,0]
        optimal_x = x[np.argmax(z)]
        optimal_y = y[np.argmax(z)]
        fig, ax = plt.subplots()
        ax.plot(x, y, 'o')
        ax.plot([optimal_x], [optimal_y], 'ro')
        ax.set_title(f"Plot {i}: Optimal ({optimal_x:.3f}, {optimal_y:.3f})")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_xlim([4.5, 5.5])
        ax.set_ylim([4.5, 5.5])
        plt.savefig(f"./figures/plot_{i}.png")
        plt.close()
        if i % 100 == 0:
            print(f"Generated {i} plots")
        if i == optimal_i + 10:
            optimal_i = i
            images.append(imageio.imread(f"./figures/plot_{i}.png"))
    imageio.mimsave("HillClimbing.gif", images, fps=10)
    # delete all the images
    for i in range(1, 2501):
        os.remove(f"./figures/plot_{i}.png")

generate_plots()
```

```
Generated 100 plots
Generated 200 plots
Generated 300 plots
Generated 400 plots
Generated 500 plots
Generated 600 plots
Generated 700 plots
Generated 800 plots
Generated 900 plots
Generated 1000 plots
Generated 1100 plots
Generated 1200 plots
Generated 1300 plots
Generated 1400 plots
Generated 1500 plots
Generated 1600 plots
Generated 1700 plots
Generated 1800 plots
Generated 1900 plots
Generated 2000 plots
Generated 2100 plots
Generated 2200 plots
Generated 2300 plots
Generated 2400 plots
Generated 2500 plots
```

Generate final plot

In [8]:
```python
import matplotlib.pyplot as plt
import numpy as np
import imageio
import  os

#catch and ignore warnings
import warnings
warnings.filterwarnings("ignore")

def generate_final_plots(opt: list):
    for i in opt:
        data = np.genfromtxt(f'{i}.txt', delimiter='\t', skip_header=1)
        x = data[:,1]
        y = data[:,2]
        z = data[:,0]
        optimal_x = x[np.argmax(z)]
        optimal_y = y[np.argmax(z)]
        fig, ax = plt.subplots()
        ax.plot(x, y, 'o')
        ax.plot([optimal_x], [optimal_y], 'ro')
        ax.set_title(f"Plot {i}: Optimal ({optimal_x:.3f}, {optimal_y:.3f})")
        ax.set_xlabel("x")
        ax.set_ylabel("y")
        ax.set_xlim([4.5, 5.5])
        ax.set_ylim([4.5, 5.5])
        plt.savefig(f"./figures/{i}_Final.png")
    plt.close()

generate_final_plots(['HillClimbingOptimizer','RandomSearchOptimizer'])
```



Look at higher dimensional problem - 30 dimensions.

In [ ]:
```python
# Higher dimensional problems
names = list(bcp.keys())
for i in names:
    # n defines dimensionality - ensures we are only plotting 2D problems
    if bcp[i]['n'] > 2:
        print(i,'\t',bcp[i]['n'])
```

In [ ]:
```python
test_fxn = bcp['3pk']
fxn_form = test_fxn['func']
xlb = test_fxn['xlb']
xub = test_fxn['xub']
print(test_fxn)
print(fxn_form)
print(xlb)
print(xub)
```

In [24]:
```python
# systematically setup search space as dictionary - this is basically dictionary comprehension
#creates a dictionary with keys i and values np.linspace(xlb[i],xub[i],100) for i in range(30)
search_space = {str(i):np.linspace(xlb[i],xub[i],100) for i in range(30)}
```

In [ ]:
```python
from gradient_free_optimizers import RandomSearchOptimizer
from IPython.display import clear_output

# create looped funtion to do replacement in string version of functions
def multiple_replace(_string:str, find_replace:list):
    new_string = _string
    for find,replace in find_replace:
        new_string = new_string.replace(find,replace)
    return new_string

# Create find and replace pairs as tuples using list comprehension - do in reverse order to avoid replacing onl
find_rep_list = [('x[:,{}]'.format(29-i),'x["{}"]'.format(29-i)) for i in range(30)]
opt_form = multiple_replace(fxn_form, find_rep_list)

# Create a container for objective function - need to use eval to evaluate string version of function, but can'
#directly to optimizer
def parent_fxn(search_space):
    #clear_output(wait=True)
    x = search_space
    return -1*eval(opt_form)

initial_guess = {str(i):90 for i in range(30)}
#opt = RandomSearchOptimizer(search_space) # initialize optimizer
opt = HillClimbingOptimizer(search_space, epsilon=0.1,initialize={"warm_start":[initial_guess]})
opt.search(parent_fxn, n_iter=100)
#opt.search(parent_fxn, n_iter=1000000) # run optimizer
```

In [36]:
```python
from gradient_free_optimizers import DirectAlgorithm

opt = DirectAlgorithm(search_space)
opt.search(parent_fxn, n_iter=10000)
```

```
Results: 'parent_fxn'
   Best score: -15.912166966465017
   Best parameter:
      '0'  : 0.43333333333333335
      '1'  : 0.5555555555555556
      '2'  : 1.1
      '3'  : 0.2888888888888889
      '4'  : 0.5777777777777778
      '5'  : 0.8555555555555556
      '6'  : 0.2555555555555556
      '7'  : 0.3666666666666667
      '8'  : 1.1
      '9'  : 0.41111111111111115
      '10' : 0.6444444444444445
      '11' : 0.5555555555555556
      '12' : 0.07777777777777778
      '13' : 0.5111111111111112
      '14' : 0.03333333333333333
      '15' : 61.510614737757685
      '16' : 70.60371198425999
      '17' : 83.90442117807
      '18' : 18.832469908387775
      '19' : 18.02205847713788
      '20' : 98.04982949408989
      '21' : 102.69354233774747
      '22' : 18.2007357678697
      '23' : 290.84516518886664
      '24' : 37.29377704620414
      '25' : 34.5141904667303
      '26' : 24.051691822349696
      '27' : 63.43344982989737
      '28' : 66.78069598434
      '29' : 17.811591848918283

   Random seed: 2013140372

   Evaluation time   : 8.293853998184204 sec    [53.78 %]
   Optimization time : 7.128984689712524 sec    [46.22 %]
   Iteration time    : 15.422838687896729 sec    [648.39 iter/sec]
```

In [ ]:
```python
from gradient_free_optimizers import RandomRestartHillClimbingOptimizer

opt = RandomRestartHillClimbingOptimizer(search_space, n_iter_restart=200)
opt.search(parent_fxn, n_iter=1000)
```

In [ ]:
```python
test_fxn = bcp['arglinb']
fxn_form = test_fxn['func']
xlb = test_fxn['xlb']
xub = test_fxn['xub']
print(test_fxn)
print(fxn_form)
print(xlb)
print(xub)

search_space = {str(i):np.linspace(xlb[i],xub[i],100) for i in range(10)}
```

In [ ]:
```python
find_rep_list = [('x[:,{}]'.format(9-i),'x["{}"]'.format(9-i)) for i in range(10)]
opt_form = multiple_replace(fxn_form, find_rep_list)

# Create a container for objective function - need to use eval to evaluate string version of function, but can'
#directly to optimizer
def parent_fxn(search_space):
    clear_output(wait=True)
    x = search_space
    return -1*eval(opt_form)

opt = RandomSearchOptimizer(search_space) # initialize optimizer
opt.search(parent_fxn, n_iter=100000) # run optimizer
```

In [ ]:
```python
from gradient_free_optimizers import DirectAlgorithm

opt = DirectAlgorithm(search_space)
opt.search(parent_fxn, n_iter=10000)
```