Example_EA

April 19, 2019

1 Preliminaries

2 Importing packages

```
In [1]: %matplotlib inline
        import seaborn as sns
        sns.set()
        sns.set_style('whitegrid')
        sns.set_context("notebook")

        #Plotting packages
        import matplotlib.cm as cm
        import matplotlib.mlab as mlab
        import matplotlib.pyplot as plt

        from matplotlib import colors
        from mpl_toolkits import mplot3d
        from mpl_toolkits.mplot3d import Axes3D

        #Data packages
        import numpy as np
        import pandas as pd

        plt.rcParams['figure.figsize'] = [10, 5]
```

3 Defining some visualization functions

```
In [2]: def plt_land(a=15, b=-80):
            #temporal data for surface triangulation
            X_g = X.flatten()
            Y_g = Y.flatten()
            Z_g = f(X_g,Y_g)

            #plotting surface
            fig = plt.figure(figsize=(15,5))
            ax = fig.add_subplot(1, 2, 1, projection='3d')
```

```
        ax.view_init(a,b)
        ax.plot_trisurf(X_g, Y_g, Z_g,
                        color='c', alpha=0.3, linewidth=0)
        ax.set_aspect('auto')
        ax.autoscale_view()
        ax.scatter( -1, -1, 0, s=20)
        ax.set_xlabel('x')
        ax.set_ylabel('y')
        ax.set_zlabel('z')


        #plotting level curves
        levels = 15
        ay = fig.add_subplot(1,2,2)
        CS = ay.contour(X, Y, Z, levels)
        ay.scatter(-1, -1)
        ay.clabel(CS, inline=True, fontsize=8)
        ay.set_aspect('auto')

        #adjusting
        plt.tight_layout()
        plt.show()


    def update_par_coord():
        global xp, yp, zp
        xp = parents[:,3]
        yp = parents[:,4]
        zp = parents[:,2]


    def update_pro_coord():
        global xr, yr, zr
        xr = progeny[:, 3]
        yr = progeny[:, 4]
        zr = progeny[:, 2]
```

```
In [3]: def plt_par(gen, a=15, b=-80):
            #temporal dataframe with a generation's parents
            query = (generations['function']=='parent') & (generations['generation']==gen)
            par_g = generations[query]

            xp_g = par_g['gen_x'].values
            yp_g = par_g['gen_y'].values
            zp_g = par_g['fitness'].values

            #temporal values for surface triangulation
```

```python
    X_g = X.flatten()
    Y_g = Y.flatten()
    Z_g = f(X_g,Y_g)

    #plotting surface with points
    ##fig = plt.figure(figsize=(plt.figaspect(0.5)))
    fig = plt.figure(figsize=(15,5))
    ax = fig.add_subplot(1, 2, 1, projection='3d')
    ax.view_init(a,b)
    ax.plot_trisurf(X_g, Y_g, Z_g,
                    color='c', alpha=0.3, linewidth=0)
    ax.set_aspect('auto')
    ax.autoscale_view()
    ax.scatter(xp_g, yp_g, zp_g)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ##ax.set_aspect("equal")

    #plotting level curves
    levels = 15
    ay = fig.add_subplot(1,2,2)
    ay.contour(X, Y, Z, levels)
    ay.set_aspect('auto')
    ay.scatter(xp_g, yp_g)

    plt.tight_layout()

    plt.show()


def plt_par_3D():
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(
        X, Y, Z,  rstride=1, cstride=1, color='c', alpha=0.3, linewidth=0)

    ax.autoscale(False)

    ax.scatter( xp, yp, zp, color="k", s=20)

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    ax.set_aspect("equal")
    plt.tight_layout()
    plt.show()
```

```
    ax.view_init(30, 30);


def plt_par_2Dcc():
    fig = plt.figure()
    ax = fig.add_subplot(111)
    levels = 15
    ax.contour(X, Y, Z, levels)
    ax.autoscale(False)
    ax.scatter(xp, yp)
```

```
In [4]: def plt_gen(gen, a=15, b=-80):
            #temporal dataframe with a generation's parents
            query = (generations['function']=='parent') & (generations['generation']==gen)
            par_g = generations[query]

            xp_g = par_g['gen_x'].values
            yp_g = par_g['gen_y'].values
            zp_g = par_g['fitness'].values

            #temporal dataframe with a generation's progeny
            query = (generations['function']=='progeny') & (generations['generation']==gen)
            prg_g = generations[query]

            xr_g = prg_g['gen_x'].values
            yr_g = prg_g['gen_y'].values
            zr_g = prg_g['fitness'].values

            #temporal values for surface triangulation
            X_g = X.flatten()
            Y_g = Y.flatten()
            Z_g = f(X_g,Y_g)

            #plotting surface with points
            ##fig = plt.figure(figsize=(plt.figaspect(0.5)))
            fig = plt.figure(figsize=(15,5))
            ax = fig.add_subplot(1, 2, 1, projection='3d')
            ax.view_init(a,b)
            ax.plot_trisurf(X_g, Y_g, Z_g,
                            color='c', alpha=0.3, linewidth=0)
            ax.set_aspect('auto')
            ax.autoscale_view()
            ax.scatter(xp_g, yp_g, zp_g)
            ax.scatter(xr_g, yr_g, zr_g)
            ax.set_xlabel('x')
```

```
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    ##ax.set_aspect("equal")

    #plotting level curves
    levels = 15
    ay = fig.add_subplot(1,2,2)
    ay.contour(X, Y, Z, levels)
    ay.set_aspect('auto')
    ay.scatter(xp_g, yp_g)
    ay.scatter(xr_g, yr_g)

    plt.tight_layout()

    plt.show()


def plt_gen_3D():
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(
        X, Y, Z,  rstride=1, cstride=1, color='c', alpha=0.3, linewidth=0)

    ax.autoscale(False)

    ax.scatter( xp, yp, zp, s=20)
    ax.scatter( xr, yr, zr, s=20)

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    ax.set_aspect("equal")
    plt.tight_layout()
    plt.show()

    ax.view_init(30, 30);


def plt_gen_2Dcc():
    fig = plt.figure()
    ax = fig.add_subplot(111)
    levels = 15
    ax.contour(X, Y, Z, levels)
    ax.autoscale(False)
    ax.scatter(xp, yp)
    ax.scatter(xr, yr)
```

4 Startup

4.1 Defining Fitness Landscape

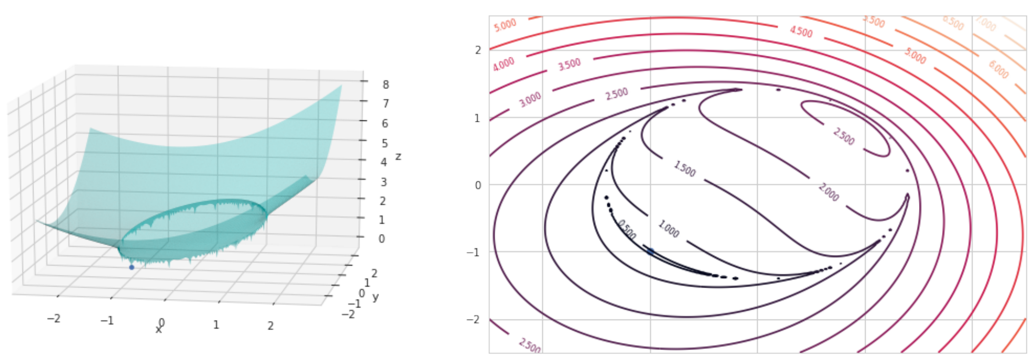
2-D function "Happy cat" (Name because of the peculiar shape of its countour curves) It has a minimum in (x,y) = (-1, -1) that is difficult to achieve.

In [5]:
```
# Function definition
def f(x, y):
    D = 2
    alpha = 1/8
    a = np.abs(x ** 2 + y ** 2 - D) ** (alpha * D)
    b = ( 0.5 * (x ** 2 + y ** 2) + (x + y) ) / D

    return (a + b + 0.5)

# Sampling between the domains of the problem
x = np.linspace(-2.5, 2.5, 251)
y = np.linspace(-2.5, 2.5, 251)

X, Y = np.meshgrid(x, y)
Z = f(X, Y)
```

In [6]:
```
#Testing the minimum
f(-1,-1)
```

Out[6]: 0.0

In [7]:
```
# Graphing the landscape
plt_land(15,-80)
```



4.2 Generating initial population randomly

In [8]:
```
# Let's pick ten random numbers in the problem domain (with uniform probability)
np.random.seed(654321)
```

```
#Population size
pop_s = 10

initial = np.random.uniform(-2.5000, 2.5000, (pop_s,5))

# Fitness in column 3
initial[:,2] = f(initial[:,3],initial[:,4])
# Column 0: birthdate, column 1: generation
#initial[:,0] = np.arange(10)
initial[:,0] = np.zeros(pop_s)
initial[:,1] = np.ones(pop_s)*2
```

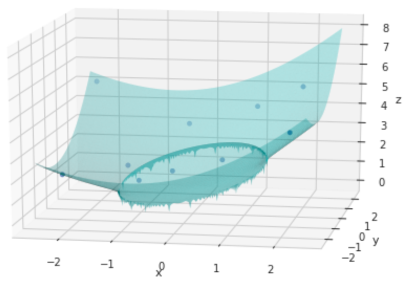
In [9]:
```
#We'll store it as a dataframe
generation = 0
cols = ['generation','function','fitness', 'gen_x', 'gen_y' ]

generations = pd.DataFrame(initial, columns=cols)

query = (generations['generation']==generation) & (generations['function']==2)

generations.loc[query, "function"] = "parent"
generations

#We set the parents population as the
parents = np.copy(initial)

generations
```

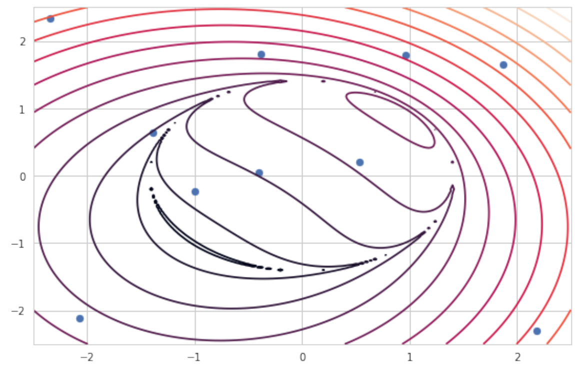
Out[9]:

	generation	function	fitness	gen_x	gen_y
0	0.0	parent	4.135337	0.965858	1.797411
1	0.0	parent	4.969311	-2.338178	2.341816
2	0.0	parent	1.132752	-0.998618	-0.235342
3	0.0	parent	1.526142	-0.400435	0.042969
4	0.0	parent	5.252818	1.872661	1.650052
5	0.0	parent	1.457118	-1.382985	0.636565
6	0.0	parent	2.086830	0.538902	0.196349
7	0.0	parent	4.638118	2.181788	-2.300281
8	0.0	parent	2.206452	-2.068859	-2.112496
9	0.0	parent	3.157188	-0.383263	1.808036

4.3 Visualizing

In [10]:
```
plt_par(generation, a=15, b=-80)
```





5 Simple Evolutionary Algorithm

5.1 One detailed run

5.1.1 Selection of parents

```
In [11]: ## Select the top 5 fittest individuals of the generation

         qty_prog = 5

         def select_parents(qty_prog):
             global progeny
             a = np.copy(parents)
             a = a[a[:,2].argsort()]
             progeny = a
             progeny = np.delete(progeny, list(range(qty_prog,len(parents))), axis=0)

         select_parents(qty_prog)
```

```
In [12]: progeny
```

```
Out[12]: array([[ 0.        ,  2.        ,  1.13275208, -0.99861801, -0.23534206],
                [ 0.        ,  2.        ,  1.45711755, -1.38298524,  0.636565  ],
                [ 0.        ,  2.        ,  1.52614213, -0.40043528,  0.04296868],
                [ 0.        ,  2.        ,  2.08683027,  0.53890207,  0.19634913],
                [ 0.        ,  2.        ,  2.20645243, -2.06885883, -2.11249592]])
```

5.1.2 Mutating the progeny

```
In [13]: mut_p = 0.5
         step_s = 0.5

         def mut(mut_prob, s_size):
             #Mutation probability
```

```
    #Stepsize of the mutation
    global progeny

    #We modify the x and y values of the progeny
    a = list( [(i,j) for i in range(qty_prog) for j in range(3,5) ] )

    for (i,j) in a:
        r = ( np.random.random() < mut_prob)
        if r == True:
            progeny[i,j] = progeny[i,j] + (2 * (np.random.random() - 1)) * s_size
            if progeny[i,j] > 2.5:
                progeny[i,j] = 2.5
            if progeny[i,j] < -2.5:
                progeny[i,j] = -2.5

    # Fitness in column 3
    progeny[:,2] = f(progeny[:,3],progeny[:,4])

    # Column 0: generation; column 1: function
    progeny[:,0] = np.ones(qty_prog) * int(generation)
    progeny[:,1] = np.ones(qty_prog)*5

mut(mut_p, step_s)
```

```
In [14]: progeny
```

```
Out[14]: array([[ 0.        ,  5.        ,  1.10740752, -1.04533378, -0.23534206],
                [ 0.        ,  5.        ,  2.29933341, -2.13081124,  0.636565  ],
                [ 0.        ,  5.        ,  1.00774277, -0.61864341, -0.86261098],
                [ 0.        ,  5.        ,  1.81706357,  0.53890207, -0.35674031],
                [ 0.        ,  5.        ,  2.44862541, -2.35279223, -2.11249592]])
```

5.1.3 Updating the dataframe

```
In [15]: def df_add_prog():
             global generations

             prog = pd.DataFrame(progeny, columns=cols)
             generations = generations.append(prog, ignore_index = True)

             query = (generations['generation']==generation) & (generations['function']==5)
             generations.loc[query, "function"] = "progeny"

         df_add_prog()
```

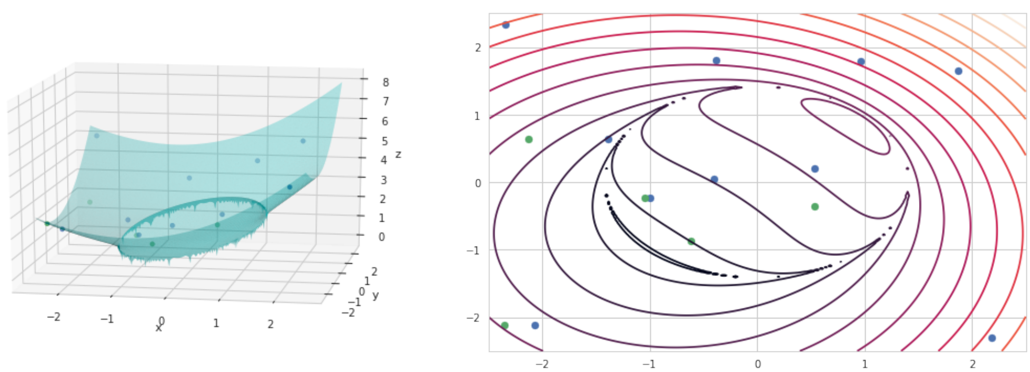
```
In [16]: generations
```

Out[16]:

	generation	function	fitness	gen_x	gen_y
0	0.0	parent	4.135337	0.965858	1.797411
1	0.0	parent	4.969311	-2.338178	2.341816
2	0.0	parent	1.132752	-0.998618	-0.235342
3	0.0	parent	1.526142	-0.400435	0.042969
4	0.0	parent	5.252818	1.872661	1.650052
5	0.0	parent	1.457118	-1.382985	0.636565
6	0.0	parent	2.086830	0.538902	0.196349
7	0.0	parent	4.638118	2.181788	-2.300281
8	0.0	parent	2.206452	-2.068859	-2.112496
9	0.0	parent	3.157188	-0.383263	1.808036
10	0.0	progeny	1.107408	-1.045334	-0.235342
11	0.0	progeny	2.299333	-2.130811	0.636565
12	0.0	progeny	1.007743	-0.618643	-0.862611
13	0.0	progeny	1.817064	0.538902	-0.356740
14	0.0	progeny	2.448625	-2.352792	-2.112496

5.1.4 Visualizing

In [17]:
```
plt_gen(generation, a=15, b=-80)
```



5.1.5 Selecting members of the new population

In [18]:
```
generation += 1
```

In [19]:
```
def new_parents():
    global parents, progeny
    parents = np.append(parents, progeny, axis=0)

    a = parents
    a = a[a[:,2].argsort()]

    parents = a
    parents = np.delete(parents, list(range(pop_s,len(parents))), axis=0)
```

```
            parents[:,2] = f(parents[:,3],parents[:,4])

            parents[:,0] = np.ones(pop_s) * int(generation)
            parents[:,1] = np.ones(pop_s)*2

        new_parents()
```

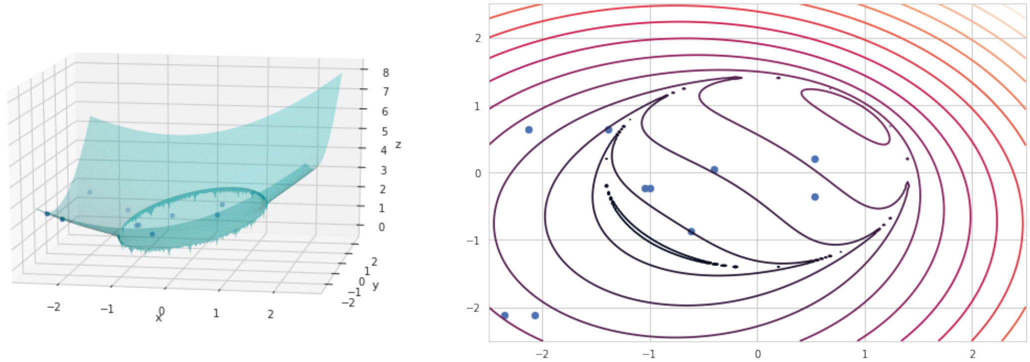
```
In [20]: def df_add_par():
             global generations
             par = pd.DataFrame(parents, columns=cols)
             generations = generations.append(par, ignore_index = True)

             query = (generations['generation']==generation) & (generations['function']==2)

             generations.loc[query, "function"] = "parent"

         df_add_par()
```

```
In [21]: with pd.option_context('display.max_rows',30):
             print(generations)
```

```
    generation function   fitness     gen_x     gen_y
0          0.0   parent  4.135337  0.965858  1.797411
1          0.0   parent  4.969311 -2.338178  2.341816
2          0.0   parent  1.132752 -0.998618 -0.235342
3          0.0   parent  1.526142 -0.400435  0.042969
4          0.0   parent  5.252818  1.872661  1.650052
5          0.0   parent  1.457118 -1.382985  0.636565
6          0.0   parent  2.086830  0.538902  0.196349
7          0.0   parent  4.638118  2.181788 -2.300281
8          0.0   parent  2.206452 -2.068859 -2.112496
9          0.0   parent  3.157188 -0.383263  1.808036
10         0.0  progeny  1.107408 -1.045334 -0.235342
11         0.0  progeny  2.299333 -2.130811  0.636565
12         0.0  progeny  1.007743 -0.618643 -0.862611
13         0.0  progeny  1.817064  0.538902 -0.356740
14         0.0  progeny  2.448625 -2.352792 -2.112496
15         1.0   parent  1.007743 -0.618643 -0.862611
16         1.0   parent  1.107408 -1.045334 -0.235342
17         1.0   parent  1.132752 -0.998618 -0.235342
18         1.0   parent  1.457118 -1.382985  0.636565
19         1.0   parent  1.526142 -0.400435  0.042969
20         1.0   parent  1.817064  0.538902 -0.356740
21         1.0   parent  2.086830  0.538902  0.196349
22         1.0   parent  2.206452 -2.068859 -2.112496
23         1.0   parent  2.299333 -2.130811  0.636565
24         1.0   parent  2.448625 -2.352792 -2.112496
```

5.1.6 Visualizing

```
In [22]: plt_par(generation)
```



6 EA for n generations

```
In [23]: qty_prog = 5
         mut_p = 0.5
         step_s = 0.5

         for i in range(57):
             select_parents(qty_prog)
             mut(mut_p, step_s)
             df_add_prog()

             generation += 1
             new_parents()
             df_add_par()
```

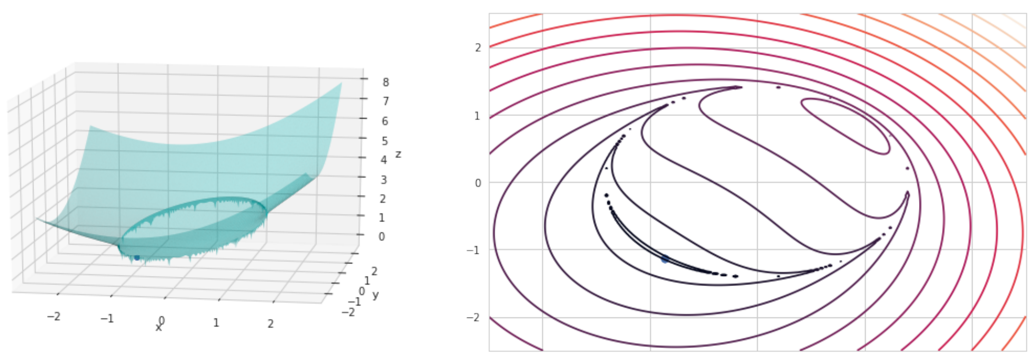
```
In [24]: with pd.option_context('display.max_rows',30):
             print(generations)
```

```
     generation function   fitness     gen_x     gen_y
0           0.0   parent  4.135337  0.965858  1.797411
1           0.0   parent  4.969311 -2.338178  2.341816
2           0.0   parent  1.132752 -0.998618 -0.235342
3           0.0   parent  1.526142 -0.400435  0.042969
4           0.0   parent  5.252818  1.872661  1.650052
5           0.0   parent  1.457118 -1.382985  0.636565
6           0.0   parent  2.086830  0.538902  0.196349
7           0.0   parent  4.638118  2.181788 -2.300281
8           0.0   parent  2.206452 -2.068859 -2.112496
9           0.0   parent  3.157188 -0.383263  1.808036
10          0.0  progeny  1.107408 -1.045334 -0.235342
```

```
11          0.0  progeny  2.299333 -2.130811  0.636565
12          0.0  progeny  1.007743 -0.618643 -0.862611
13          0.0  progeny  1.817064  0.538902 -0.356740
14          0.0  progeny  2.448625 -2.352792 -2.112496
..          ...      ...       ...       ...       ...
865        57.0  progeny  1.381239 -1.747399 -1.135008
866        57.0  progeny  0.712683 -0.860418 -1.222142
867        57.0  progeny  1.247497 -1.033460 -1.632299
868        57.0  progeny  0.420534 -0.860418 -1.135008
869        57.0  progeny  0.741781 -0.860418 -1.237963
870        58.0   parent  0.420534 -0.860418 -1.135008
871        58.0   parent  0.420534 -0.860418 -1.135008
872        58.0   parent  0.420534 -0.860418 -1.135008
873        58.0   parent  0.420534 -0.860418 -1.135008
874        58.0   parent  0.420534 -0.860418 -1.135008
875        58.0   parent  0.420534 -0.860418 -1.135008
876        58.0   parent  0.420534 -0.860418 -1.135008
877        58.0   parent  0.420534 -0.860418 -1.135008
878        58.0   parent  0.420534 -0.860418 -1.135008
879        58.0   parent  0.420534 -0.860418 -1.135008

[880 rows x 5 columns]
```

6.1 Visualize last generation

```
In [25]: plt_par(generation)
```



7 Data exploration

7.1 Evolution of fitness

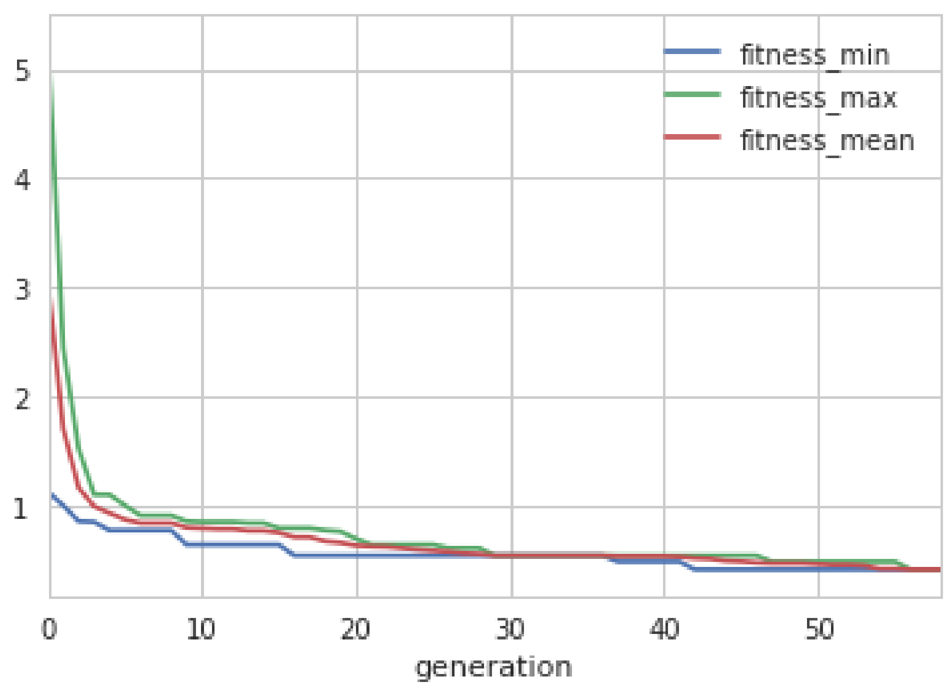
Minimum, maximum and mean fitness vs the population generation.

In [26]:
```
grouped = generations[generations['function']=='parent'].groupby('generation').agg({'fi
grouped.columns = ["_".join(x) for x in grouped.columns.ravel()]

with pd.option_context('display.max_rows',10):
    print(grouped)
```

```
            fitness_min  fitness_max  fitness_mean
generation
0.0            1.132752     5.252818      3.056207
1.0            1.007743     2.448625      1.708947
2.0            0.862333     1.526142      1.169870
3.0            0.856666     1.108174      0.999856
4.0            0.780174     1.107408      0.937872
...                 ...          ...           ...
54.0           0.420534     0.492769      0.427757
55.0           0.420534     0.492769      0.427757
56.0           0.420534     0.420534      0.420534
57.0           0.420534     0.420534      0.420534
58.0           0.420534     0.420534      0.420534

[59 rows x 3 columns]
```

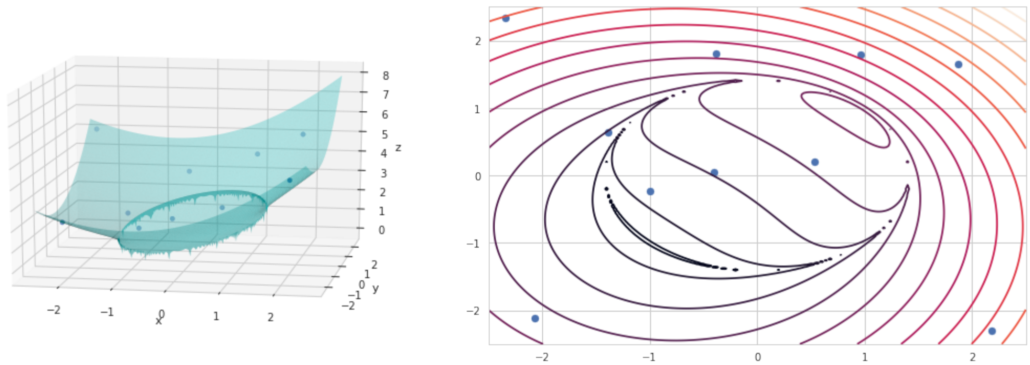
In [27]:
```
grouped.plot()
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2e945ba860>

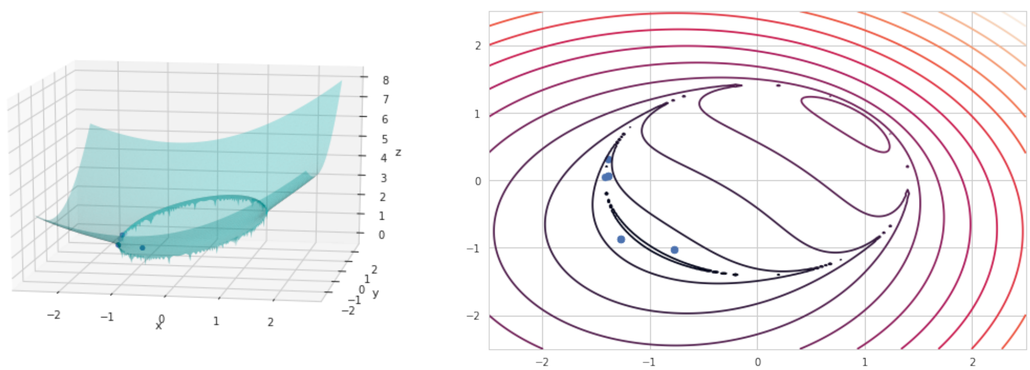


7.2 Migration of Population

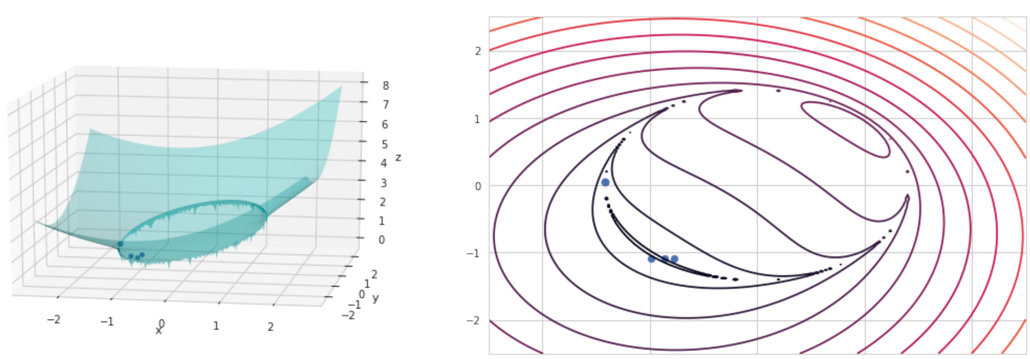
```
In [28]: plt_par(0)
```



```
In [29]: plt_par(10)
```



```
In [ ]: plt_par(20)
```



In [ ]: plt_par(30)

In [ ]: plt_par(40)

In [ ]: plt_par(50)

8 Scratchpad

In [ ]:
```
#ploting a specific generation
gen = 0

#temporal dataframe with a generation's parents
query = (generations['function']=='parent') & (generations['generation']==gen)
par_g = generations[query]

xp_g = par_g['gen_x'].values
yp_g = par_g['gen_y'].values
zp_g = par_g['fitness'].values

#temporal dataframe with a generation's progeny
query = (generations['function']=='progeny') & (generations['generation']==gen)
prg_g = generations[query]

xr_g = prg_g['gen_x'].values
yr_g = prg_g['gen_y'].values
zr_g = prg_g['fitness'].values

#temporal values for surface triangulation
X_g = X.flatten()
Y_g = Y.flatten()
Z_g = f(X_g,Y_g)

#plotting surface with points
##fig = plt.figure(figsize=(plt.figaspect(0.5)))
fig = plt.figure(figsize=(15,5))
ax = fig.add_subplot(1, 2, 1, projection='3d')
ax.plot_trisurf(X_g, Y_g, Z_g,
               color='c', alpha=0.3, linewidth=0)
ax.set_aspect('auto')
#ax.autoscale(False)
ax.scatter(xp_g, yp_g, zp_g)
ax.scatter(xr_g, yr_g, zr_g)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

#ax.set_aspect("equal")
```

```
#plotting level curves
ay = fig.add_subplot(1,2,2)
levels = 15
ay.contour(X, Y, Z, levels)
ay.set_aspect('auto')
#ay.autoscale(False)
ay.scatter(xp_g, yp_g)
ay.scatter(xr_g, yr_g)

plt.tight_layout()

plt.show()
```

```
In [ ]: X.shape
```