این تابع با نام `initialize_population` دریافتی‌های زیر را می‌گیرد:

`lower_limit`: - حد پایین محدوده مقادیر ممکن برای هر بعد (dimension) در جمعیت اولیه.

`upper_limit`: - حد بالا محدوده مقادیر ممکن برای هر بعد (dimension) در جمعیت اولیه.

`pop_size`: - اندازه جمعیت اولیه، یعنی تعداد افراد (individuals) موجود در جمعیت.

`dimensions`: -تعداد بعدها (dimensions) برای هر فرد (individual) در جمعیت.

در ابتدا، یک آرایه به نام `pop` برای نگهداری جمعیت ایجاد می‌شود.

سپس، به ازای هر عضو در جمعیتبا استفاده از حلقه for ، دو بخش اصلی تشکیل دهنده آن ایجاد می‌شود:

`individual`: .1یک فرد جدید که در واقع یک لیست است که دارای دو عنصر است.

- عنصر اول لیست(`np.random.uniform(lower_limit, upper_limit, size=dimensions)`) ، یک آرایه با اندازه `dimensions`است که با استفاده از تابع`np.random.uniform` ، از توزیع یکنواخت در محدوده `lower_limit` تا `upper_limit`مقادیر تصادفی دریافت می‌کند. این آرایه مقادیر مربوط به هر بعد (dimension) را برای فرد مشخص می‌کند.

- عنصر دوم لیست(`np.random.uniform(upper_limit)`) ، یک عدد تصادفی در محدوده 0 تا `upper_limit` است که برای فرد مشخص می‌کند.

در نهایت، جمعیت (`pop`) به صورت یک آرایه numpy با نوع داده `object` برگشت داده می‌شود.

برای استفاده از این تابع، باید مقادیر مناسب برای ورودی‌ها مانند`lower_limit` ، `upper_limit`، `pop_size`و `dimensions` را تعیین کنید و سپس با فراخوانی تابع، جمعیت اولیه را دریافت کنید.

تابع `schwefel` به عنوان یک تابع هدف (objective function) برای محاسبه مقدار فیتنس (fitness) یک فرد در الگوریتم‌های بهینه‌سازی استفاده می‌شود. این تابع با دریافت یک آرایه به نام `a`، ابعاد مختلف، مقدار فیتنس را برگردانده و در متغیر `result` ذخیره می‌کند. برای این منظور، ابتدا متغیر `alpha` را به عنوان یک عدد ثابت مقداردهی می‌کند. سپس با استفاده از توابع numpy مانند ``np.sin`، ``np.sqrt`` و `np.sum`، مقادیر مختلف را محاسبه کرده و در نهایت مقدار فیتنس را بر اساس فرمول محاسبه‌شده برمی‌گرداند.

تابع `ackley` نیز یک تابع هدف دیگر است که برای محاسبه مقدار فیتنس استفاده می‌شود. این تابع نیز با دریافت یک آرایه به نام `a`، ابعاد مختلف، مقدار فیتنس را برگردانده و در متغیر `result` ذخیره می‌کند. در این تابع، متغیرهای ``a`، ``b` و `c` مقادیر ثابتی هستند که می‌توانند با استفاده از کامنت‌ها فعال شوند. سپس با استفاده از توابع numpy مانند ``np.sum`، ``np.exp` و `np.cos`، مقادیر مختلف را محاسبه کرده و مقدار فیتنس را بر اساس فرمول محاسبه‌شده برمی‌گرداند.

در هر دو تابع، مقدار فیتنس محاسبه شده در متغیر `result` قرار دارد و با استفاده از دستور `return` به عنوان خروجی تابع برگشت داده می‌شود.

تابع `fitness_for_schwefel` به عنوان یک تابع برای محاسبه مقدار فیتنس برای هر عضو (individual) در جمعیتی به نام `population` استفاده می‌شود. این تابع با گرفتن جمعیتی به عنوان ورودی، مقادیر فیتنس مربوطه را برای هر عضو محاسبه می‌کند و در لیست `fitness_values` ذخیره می‌کند. برای هر عضو در جمعیت، مقدار `x` از بخش اول عضو (individual) استخراج می‌شود و سپس با استفاده از تابع `schwefel`، مقدار فیتنس محاسبه می‌شود و در لیست `fitness_values` قرار می‌گیرد. در نهایت، لیست `fitness_values` حاوی مقادیر فیتنس برای تمام اعضا برگشت داده می‌شود.

تابع `fitness_for_ackley` نیز به عنوان یک تابع برای محاسبه مقدار فیتنس برای هر عضو (individual) در جمعیتی به نام `population` استفاده می‌شود. این تابع با گرفتن جمعیتی به عنوان ورودی، مقادیر فیتنس مربوطه را برای هر عضو محاسبه می‌کند و در لیست `fitness_values` ذخیره می‌کند. برای هر عضو در جمعیت، مقدار `x` از بخش اول عضو (individual) استخراج

می‌شود و سپس با استفاده از تابع `ackley`، مقدار فیتنس محاسبه می‌شود و در لیست `fitness_values` قرار می‌گیرد. در نهایت، لیست `fitness_values` حاوی مقادیر فیتنس برای تمام اعضا برگشت داده می‌شود.

تابع `parent_selection` برای انتخاب والدین (parents) از جمعیتی به نام `population` استفاده می‌شود. این تابع با گرفتن جمعیت و تعداد والدین مورد نیاز به عنوان ورودی، والدین انتخاب شده را برمی‌گرداند.

در این تابع، با استفاده از تابع `np.random.choice` والدین به صورت تصادفی از بین اندیس‌های جمعیت انتخاب می‌شوند. تابع `np.random.choice` با پارامترهای `len(population)` برای تعداد اندیس‌ها و `size=num_parents` برای تعداد والدین مورد نیاز، اندیس‌های تصادفی را از جمعیت انتخاب کرده و در متغیر `selected_parents` ذخیره می‌کند.

سپس با استفاده از این اندیس‌های انتخاب شده، والدین متناظر را از جمعیت `population` با استفاده از عملگر ایندکسینگ انتخاب کرده و در نهایت در قالب یک آرایه برمی‌گرداند.

به طور خلاصه، تابع `parent_selection` تعداد `num_parents` والدین را تصادفی از جمعیت `population` انتخاب کرده و آن‌ها را در یک آرایه برمی‌گرداند.

تابع `local_intermediary` برای تولید فرزند محلی (local intermediary) با استفاده از والدینی که به عنوان ورودی به تابع داده شده‌اند، استفاده می‌شود.

در این تابع، با استفاده از `len(parents[0])` طول یک والدین (parents) را به عنوان `chromosome_size` استخراج می‌کنیم. سپس با استفاده از تابع `np.sum` و عملگر محور 0 (`axis=0`)، مجموع تمام والدین را در جهت محور عمودی محاسبه می‌کنیم.

سپس نتیجه محاسبه را بر تعداد والدین (`len(parents)`) تقسیم می‌کنیم تا مقدار میانگین را به دست آوریم. در نهایت، مقدار میانگین به عنوان فرزند محلی (child) برگردانده می‌شود.

به طور خلاصه، تابع `local_intermediary` با محاسبه میانگین والدین و تقسیم بر تعداد والدین، فرزند محلی را تولید می‌کند و آن را برمی‌گرداند.

تابع `mutation` برای اعمال جهش به فرزندی که به عنوان ورودی به تابع داده شده‌است، استفاده می‌شود. در این تابع، با استفاده از عملگر ایندکسینگ، آرایه `child` را به دو قسمت تقسیم می‌کنیم. قسمت اول (`x`) شامل مقدار فرزند و قسمت دوم (`sigma`) شامل مقدار انحراف معیار فرزند است.

سپس با استفاده از توابع `np.random.normal` و عملگر `+` و `*`، جهش‌های مورد نیاز را برای مقادیر `sigma` و `x` محاسبه می‌کنیم. مقدار جهش برای `sigma` با استفاده از توزیع نرمال با پارامترهای (0, t) و (0, tau_prime) و با استفاده از عملگر `np.exp` محاسبه می‌شود. مقدار جهش برای `x` نیز با استفاده از توزیع نرمال با پارامترهای (0, sigma_star) و با استفاده از عملگر `np.random.normal` محاسبه می‌شود.

در ادامه، با استفاده از عملگر ایندکسینگ و استفاده از عملگرهای مقایسه‌ای (`<`, `>`) و عملگرهای حسابی (`+`, `-`)، مقدار `mutated` را محدود می‌کنیم تا در محدوده مشخص شده باشد. با استفاده از دستورات شرطی، اگر `mutated` بزرگتر از `high` یا کوچکتر از `low` باشد، مقدار آن را تصحیح می‌کنیم.

در نهایت، آرایه‌ای حاوی مقادیر `mutated` و `sigma` را با استفاده از تابع `np.array` و با نوع داده `object` برمی‌گردانیم.

به طور خلاصه، تابع `mutation` با استفاده از توابع تصادفی و عملگرهای مقایسه‌ای و حسابی، جهش را برای فرزند محاسبه کرده و فرزند جدید را برمی‌گرداند.

تابع `non_adaptive_mutation` برای اعمال جهش غیر تطبیقی (non-adaptive mutation) به فرزندی که به عنوان ورودی به تابع داده شده است، استفاده می‌شود.

در این تابع، با استفاده از عملگر ایندکسینگ، آرایه `child` را به دو قسمت تقسیم می‌کنیم. قسمت اول (`x`) شامل مقدار فرزند و قسمت دوم (`sigma`) شامل مقدار انحراف معیار فرزند است. سپس با استفاده از تابع `np.random.normal` و عملگر `+`، جهش را برای مقدار `x` محاسبه می‌کنیم. مقدار جهش برابر با `x` اصلی به اضافه یک مقدار تصادفی است که با استفاده از توزیع نرمال با پارامترهای (0, np.abs(fixed_sigma)) تولید می‌شود. سپس با استفاده از تابع `np.clip` و عملگرهای مقایسه‌ای (`>`, `<`)، مقدار `mutated_x` را محدود می‌کنیم تا در محدوده مشخص شده باشد. در نهایت، آرایه‌ای حاوی مقادیر `mutated_x` و `sigma` را با استفاده از تابع `np.array` و با نوع داده `object` برمی‌گردانیم.

به طور خلاصه، تابع `non_adaptive_mutation` با استفاده از توابع تصادفی و عملگرهای مقایسه‌ای و حسابی، جهش غیر تطبیقی را برای فرزند محاسبه کرده و فرزند جدید را برمی‌گرداند.

تابع `generational_selection` برای انتخاب نسل جدید از فرزندانی که به عنوان ورودی به تابع داده شده‌اند، استفاده می‌شود.

در این تابع، با استفاده از دستور شرطی (`if-elif-else`) بررسی می‌شود که نوع تابع فیتنس (`fit`) چیست. اگر مقدار `fit` برابر با "schwefel" باشد، تابع `fitness_for_schwefel` بر روی آرایه `children` اعمال می‌شود و نتایج فیتنس برای فرزندان به دست می‌آید. اگر مقدار `fit` برابر با "ackley" باشد، تابع `fitness_for_ackley` بر روی آرایه `children` اعمال می‌شود و نتایج فیتنس برای فرزندان به دست می‌آید. در غیر این صورت، پیامی نمایش داده می‌شود و مقدار `None` برگردانده می‌شود.

سپس با استفاده از تابع `np.argsort` و تابع `[:survivors]`، اندیس‌هایی از آرایه فیتنس به دست می‌آید که مقادیر فیتنس آن‌ها کمینه است و تعداد آن‌ها برابر با `survivors` است.

در نهایت، نسل جدیدی از فرزندان با استفاده از اندیس‌های کمینه‌فیتنس به دست آمده، از آرایه `children` انتخاب می‌شود و به عنوان خروجی تابع برگردانده می‌شود.

به طور خلاصه، تابع `generational_selection` با بررسی نوع تابع فیتنس و انتخاب فرزندان با مقادیر کمینه‌فیتنس، نسل جدیدی از فرزندان را برمی‌گرداند.

تابع `elitist_selection` برای انتخاب نسل جدید از جمعیت اصلی و فرزندانی که به عنوان ورودی به تابع داده شده‌اند، استفاده می‌شود.

در این تابع، با استفاده از دستور `global`، متغیر `fit` را به عنوان متغیر جهانی تعریف می‌کنیم.

سپس با استفاده از تابع `np.vstack`، آرایه `population` و `children` را در هم می‌چسبانیم تا جمعیت ترکیبی بسازیم.

سپس با استفاده از دستور شرطی (`if-elif-else`) بررسی می‌شود که نوع تابع فیتنس (`fit`) چیست. اگر مقدار `fit` برابر با "schwefel" باشد، تابع `fitness_for_schwefel` بر روی آرایه `combined_population` اعمال می‌شود و نتایج فیتنس برای جمعیت ترکیبی به دست می‌آید. اگر مقدار `fit` برابر با "ackley" باشد، تابع `fitness_for_ackley` بر روی آرایه `combined_population` اعمال می‌شود و نتایج فیتنس برای جمعیت ترکیبی به دست می‌آید. در غیر این صورت، پیامی نمایش داده می‌شود و مقدار `None` برگردانده می‌شود.

سپس با استفاده از تابع `np.argsort` و تابع `[:survivors]`، اندیس‌هایی از آرایه فیتنس به دست می‌آید که مقادیر فیتنس آن‌ها کمینه است و تعداد آن‌ها برابر با `survivors` است.

در نهایت، نسل جدیدی از جمعیت با استفاده از اندیس‌های کمینه‌فیتنس به دست آمده، از آرایه `combined_population` انتخاب می‌شود و به عنوان خروجی تابع برگردانده می‌شود.

به طور خلاصه، تابع `elitist_selection` با ترکیب جمعیت اصلی و فرزندان، بررسی نوع تابع فیتنس و انتخاب جمعیت با مقادیر کمینه‌فیتنس، نسل جدیدی از جمعیت را برمی‌گرداند.

تابع `offsprings` برای تولید فرزندان جدید براساس جمعیت اولیه، انتخاب والدین، ترکیب ژن‌ها و جهش استفاده می‌شود.

در این تابع، یک لیست خالی به نام `children` تعریف می‌شود.

سپس با استفاده از حلقه `for`، برای تعداد `7 * len(population)` فرزند جدید تولید می‌شود. در هر مرحله از حلقه، با استفاده از دستور شرطی (`if-else`) بررسی می‌شود که آیا انتخاب والدین (`parent_selections`) روشن است یا خیر. اگر روشن باشد، تابع `parent_selection` بر روی جمعیت اولیه فراخوانی می‌شود و دو والدین برگردانده می‌شود. در غیر این صورت، والدین برابر با جمعیت اولیه قرار می‌گیرند.

سپس با استفاده از دستور شرطی دیگر، بررسی می‌شود که آیا ترکیب ژن‌ها (`recombination`) روشن است یا خیر. اگر روشن باشد، تابع `local_intermediary` بر روی والدین صدا زده می‌شود و یک فرزند جدید به دست می‌آید. در غیر این صورت، فرزند برابر با والدین قرار می‌گیرد.

در قدم بعد، با استفاده از دستور شرطی دیگر، بررسی می‌شود که آیا جهش (`mutate`) روشن است یا خیر و از چه نوعی استفاده شود. اگر روشن باشد و نوع آن "adaptive" باشد، تابع `mutation` بر روی فرزند اعمال می‌شود. اگر نوع جهش "non adaptive" باشد، تابع `non_adaptive_mutation` بر روی فرزند اعمال می‌شود. در صورتی که جهش غیرفعال باشد، ادامه‌ی حلقه برای تولید فرزند بعدی انجام می‌شود.

فرزند تولید شده به لیست `children` اضافه می‌شود. در نهایت، آرایه `children` به عنوان خروجی تابع برگردانده می‌شود.

به طور خلاصه، تابع `offsprings` با استفاده از انتخاب والدین، ترکیب ژن‌ها و جهش، فرزندان جدیدی را براساس جمعیت اولیه تولید می‌کند و آن‌ها را به عنوان خروجی برمی‌گرداند.

این کد یک الگوریتم تکامل استراتژی‌محور (Evolution Strategy) را پیاده‌سازی می‌کند. الگوریتم به صورت زیر است:

1. ابتدا جمعیت اولیه با استفاده از تابع `initialize_population` تولید می‌شود.

2. سپس بر اساس نوع تابع فیتنس (Schwefel یا Ackley)، مقادیر فیتنس اولیه برای جمعیت محاسبه می‌شوند و در متغیر `first_fitness` قرار می‌گیرند.

3. لیست‌های `max_fit` و `avg_fit` برای ذخیره کردن مقادیر فیتنس بیشینه و میانگین در طول نسل‌ها ایجاد می‌شوند.

4. برای تعداد نسل‌های مشخص شده، مراحل زیر تکرار می‌شوند:

   - فرزندان جدید با استفاده از تابع `offsprings` تولید می‌شوند.

   - براساس روش انتخاب بقا (generational یا elitist)، جمعیت بروزرسانی می‌شود.

   - بر اساس نوع تابع فیتنس، مقادیر فیتنس برای جمعیت محاسبه می‌شوند و در متغیر `fitness_values` قرار می‌گیرند.

   - مقادیر بیشینه و میانگین فیتنس در لیست‌های `max_fit` و `avg_fit` اضافه می‌شوند.

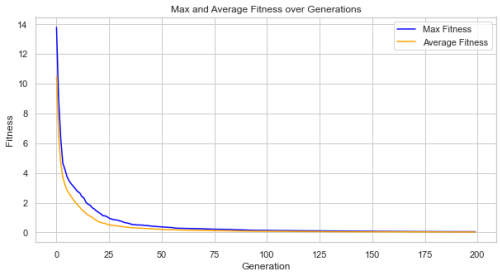
   - اگر مقدار کمینه‌ی فیتنس کمتر یا مساوی e-4.1 شود، جایگاه فرد با کمترین فیتنس پیدا شده و نتیجه نمایش داده می‌شود.

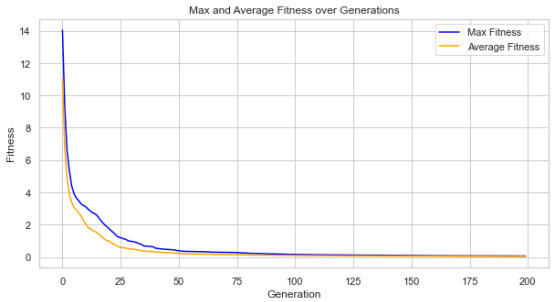
5. مقادیر بیشینه و میانگین فیتنس در طول دسته‌ای از اجراها (برابر با `num_of_run`) محاسبه شده و میانگین آن‌ها نمایش داده می‌شود.

Ackley : Survival : elitisem , Mutate : adebtive , Recombination : on , ndim 2

min fit : 0.06207992288807462 , Generation 200 - Best Fitness: 0.06207992288807462, Average Fitness: 0.0369418887992577

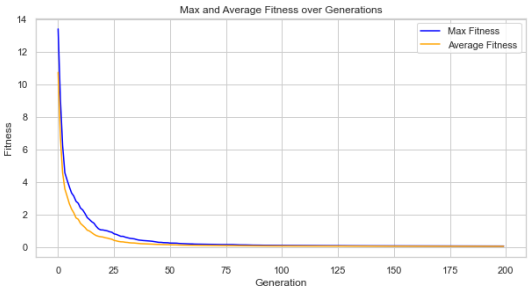
min fit : 0.048829998034439104 , Generation 200 - Best Fitness: 0.048829998034439104, Average Fitness: 0.028712657266629353

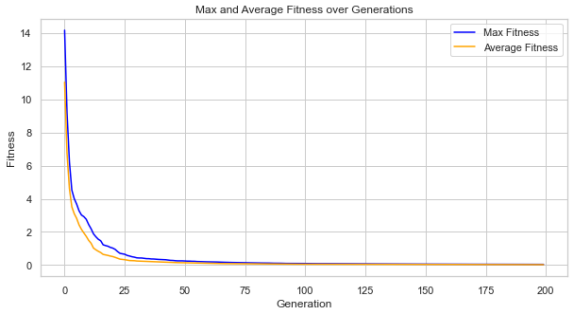




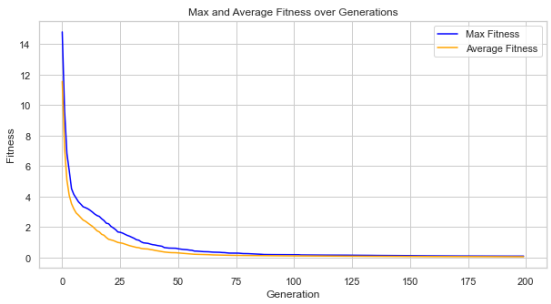
min fit : 0.044900147551260705 , Generation 200 - Best Fitness: 0.044900147551260705, Average Fitness: 0.028052322464173204

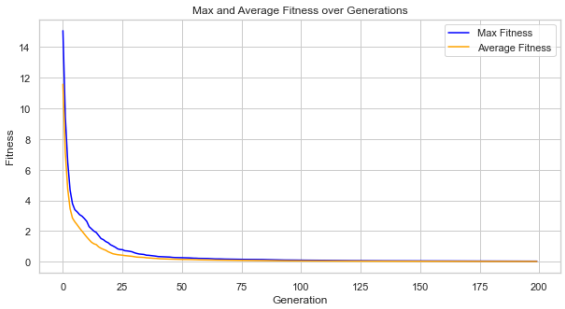
min fit : 0.04800050014688084 , Generation 200 - Best Fitness: 0.04800050014688084, Average Fitness: 0.032542841411164695



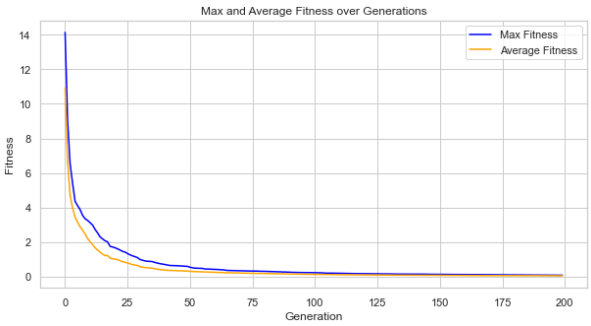


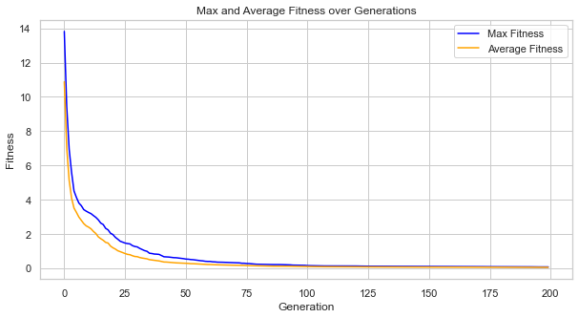
min fit : 0.04047997586283758 , Generation 200 - Best Fitness: 0.04047997586283758, Average Fitness: 0.023254179634493877
min fit : 0.08796651137608835 , Generation 200 - Best Fitness: 0.08796651137608835, Average Fitness: 0.050484363258932914





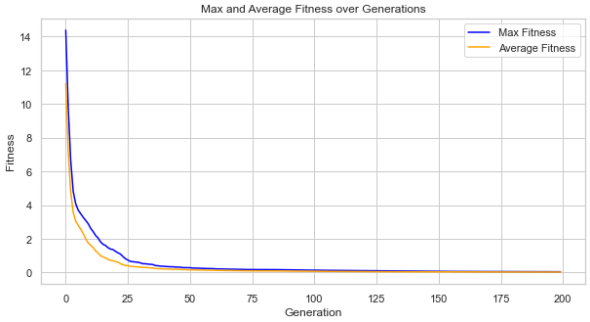
min fit : 0.06187964555508474 , Generation 200 - Best Fitness: 0.06187964555508474, Average Fitness: 0.03803713093311627
min fit : 0.0749315607864216 , Generation 200 - Best Fitness: 0.0749315607864216, Average Fitness: 0.04483780926620877

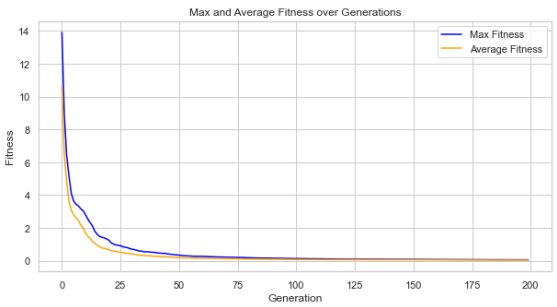




min fit : 0.05444093808507988 , Generation 200 - Best Fitness: 0.05444093808507988, Average Fitness: 0.03415440505455757

min fit : 0.04925721297780017 , Generation 200 - Best Fitness: 0.04925721297780017, Average Fitness: 0.03090802427667592

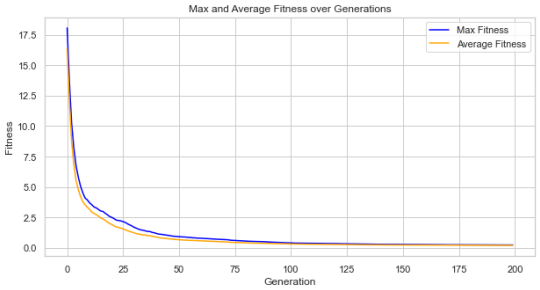


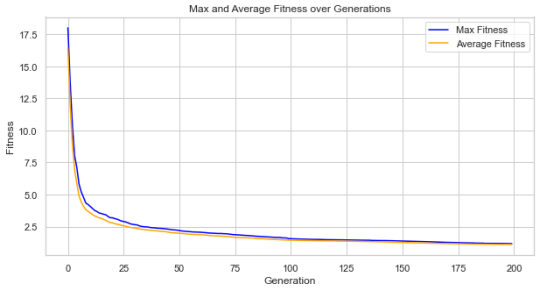


max fit average for 2 runs: 0.05727664132639676
avg fit average for 2 runs: 0.03479256223652103

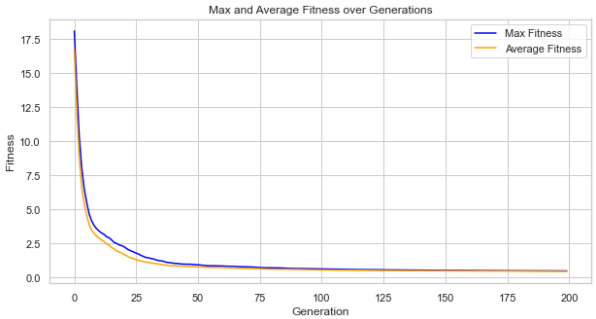
Ackley : Survival : elitisem , Mutate : adebtive , Recombination : on , ndim 5

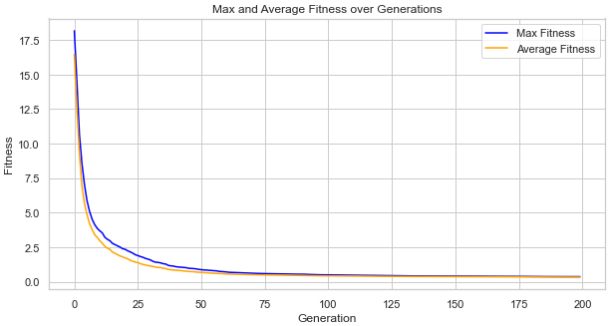
```
min fit : 1.177314483048772 , Generation 200 - Best Fitness: 1.17731448304
8772, Average Fitness: 1.1094680112472504
min fit : 0.2274903687358436 , Generation 200 - Best Fitness: 0.2274903687
358436, Average Fitness: 0.20002084157570477
```



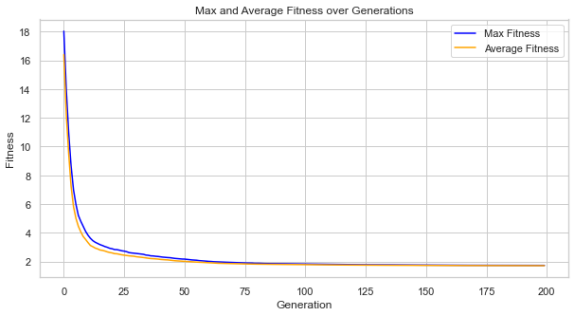


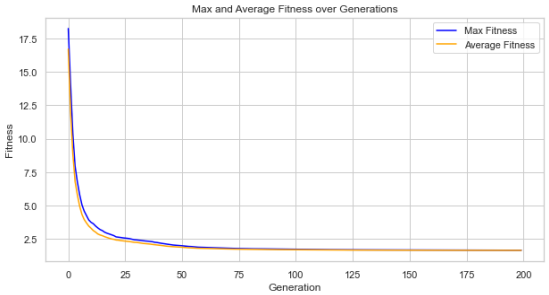

```
min fit : 0.372759714028025 , Generation 200 - Best Fitness: 0.37275971402802
5, Average Fitness: 0.34216896245536554
min fit : 0.460660845450072 , Generation 200 - Best Fitness: 0.46066084545
0072, Average Fitness: 0.4273188656891934
```



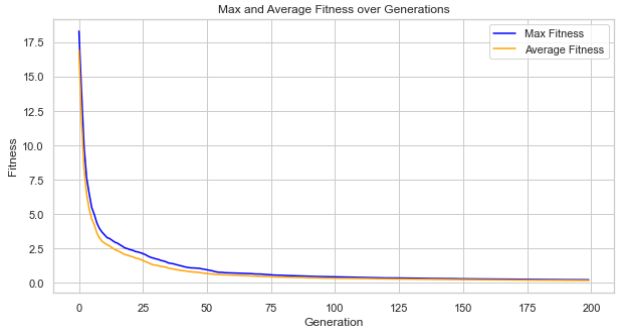


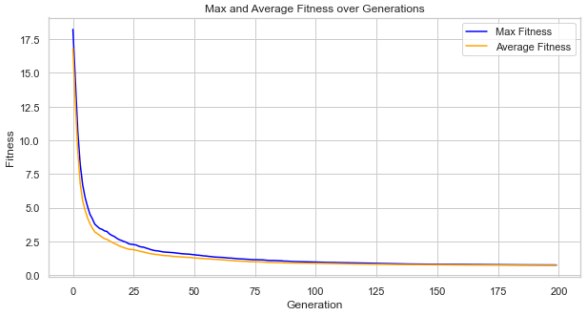

```
min fit : 1.6655054637522828 , Generation 200 - Best Fitness: 1.6655054637
522828, Average Fitness: 1.6559340276048986
min fit : 1.7364763493417184 , Generation 200 - Best Fitness: 1.7364763493
417184, Average Fitness: 1.7180458426634972
```



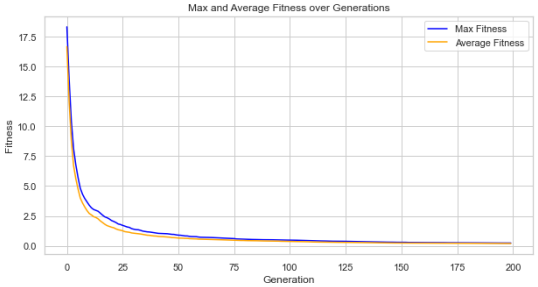


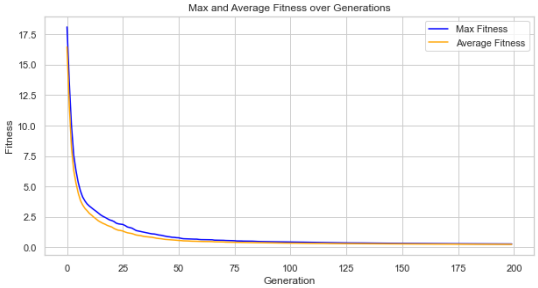
```
min fit : 0.7538949909135941 , Generation 200 - Best Fitness: 0.7538949909
135941, Average Fitness: 0.7284413104476413
min fit : 0.23016620723739623 , Generation 200 - Best Fitness: 0.230166207
23739623, Average Fitness: 0.1796956339062457
```






```
min fit : 0.27633976081720535 , Generation 200 - Best Fitness: 0.276339760
81720535, Average Fitness: 0.2465211931109061
min fit : 0.23213347428001585 , Generation 200 - Best Fitness: 0.232133474
28001585, Average Fitness: 0.193986635662569
```

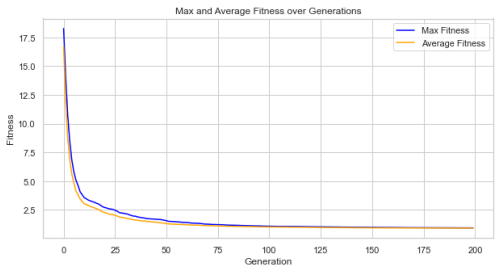


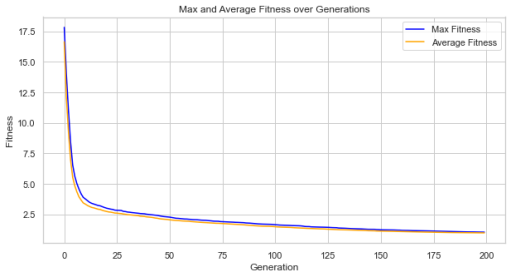



```
max fit average for 2 runs: 0.7132741657604924
avg fit average for 2 runs: 0.6801601324363272
```

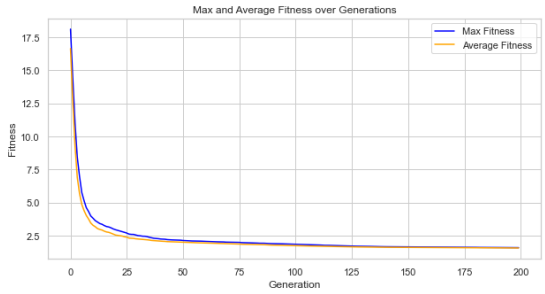
Ackley : Survival : elititstm, Mutate : non adebtive , Recombination : on , ndim 5

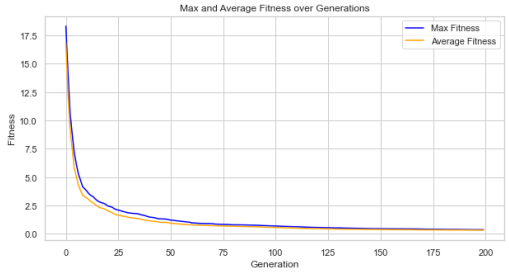
```
min fit : 1.0343669258510535 , Generation 200 - Best Fitness: 1.0343669258
510535, Average Fitness: 0.973965867768742
min fit : 0.9342590517448248 , Generation 200 - Best Fitness: 0.9342590517
448248, Average Fitness: 0.9140080448098451
```



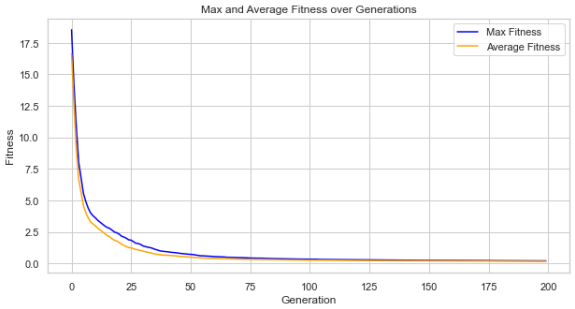


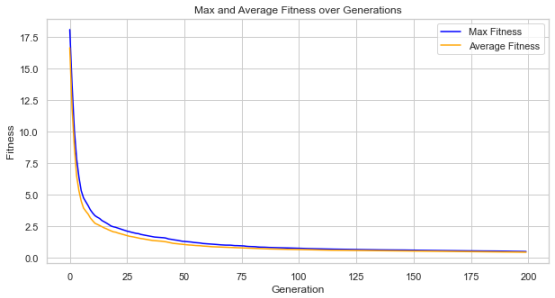
min fit : 0.37200531927657154 , Generation 200 - Best Fitness: 0.372005319
27657154, Average Fitness: 0.33145547500208505
min fit : 1.6005150661013237 , Generation 200 - Best Fitness: 1.6005150661
013237, Average Fitness: 1.575167526931248



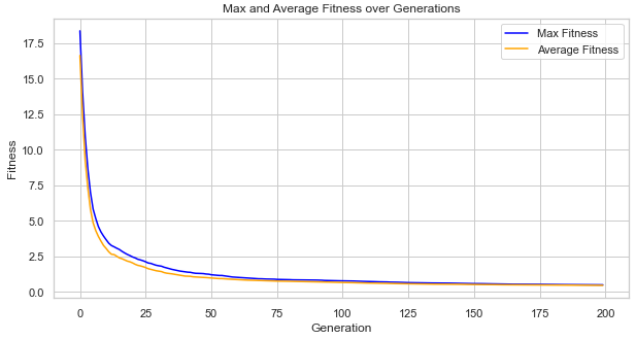


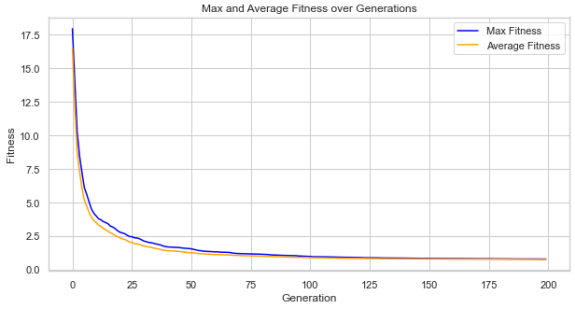
min fit : 0.477705348087206 , Generation 200 - Best Fitness: 0.47770534808
7206, Average Fitness: 0.4223058826152611
min fit : 0.20578910102210957 , Generation 200 - Best Fitness: 0.205789101
02210957, Average Fitness: 0.17297369122490186



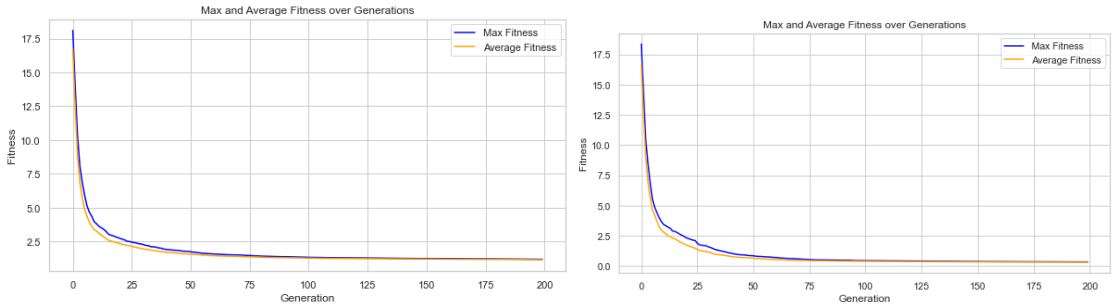


min fit : 0.7717226411567393 , Generation 200 - Best Fitness: 0.7717226411
567393, Average Fitness: 0.7427179687868828
min fit : 0.4824102727632291 , Generation 200 - Best Fitness: 0.4824102727
632291, Average Fitness: 0.44531293209914524





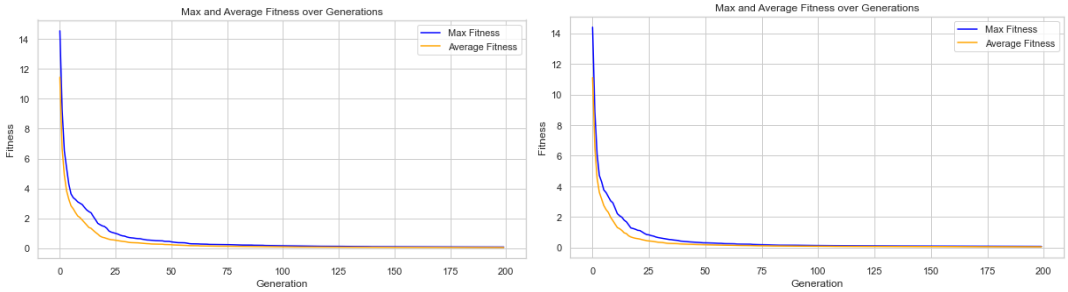
```
min fit : 0.3468701890156969 , Generation 200 - Best Fitness: 0.3468701890
156969, Average Fitness: 0.32495466834567033
min fit : 1.1954769884748946 , Generation 200 - Best Fitness: 1.1954769884
748946, Average Fitness: 1.1644741012309296
```



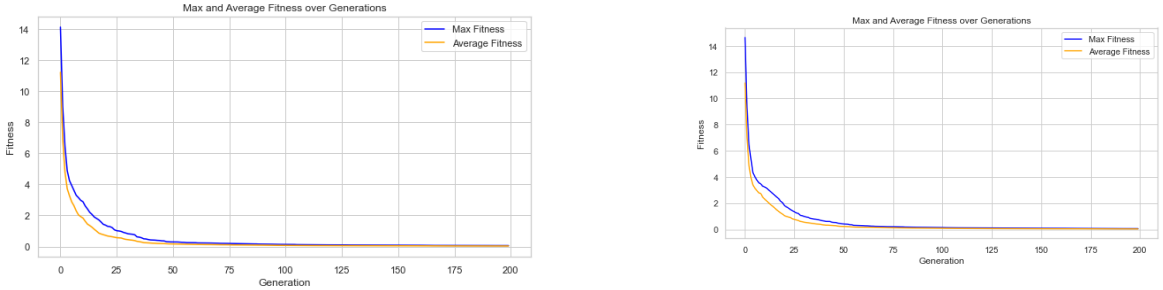

```
max fit average for 2 runs: 0.7421120903493648
avg fit average for 2 runs: 0.7067336158814711
```

Ackley : Survival : elititstm, Mutate : non adebtive , Recombination : on , ndim 2

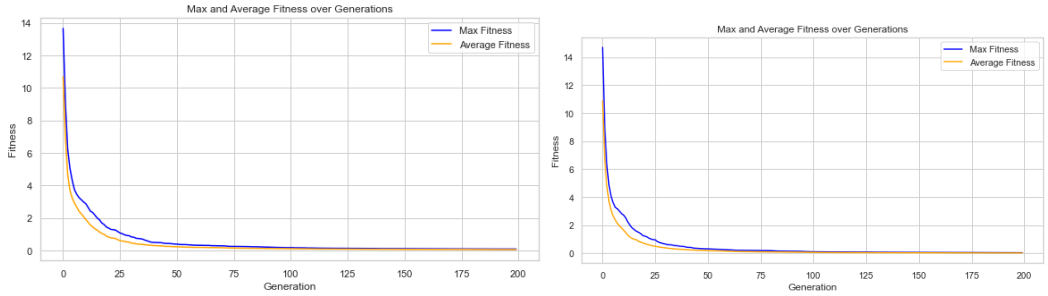
```
min fit : 0.06530849221248092 , Generation 200 - Best Fitness: 0.065308492
21248092, Average Fitness: 0.040059452750422705
min fit : 0.06703701943949314 , Generation 200 - Best Fitness: 0.067037019
43949314, Average Fitness: 0.037398530055482465
```



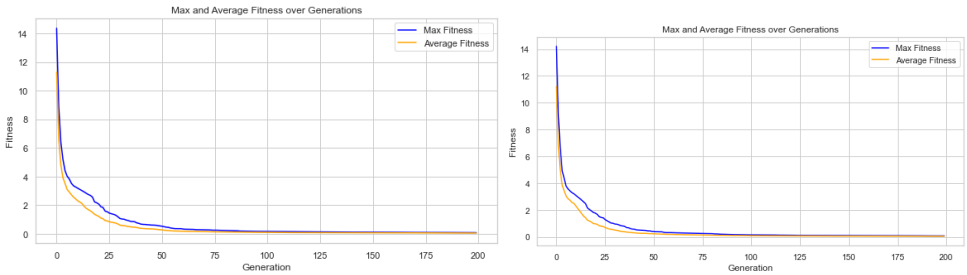

```
min fit : 0.059665664258411244 , Generation 200 - Best Fitness: 0.05966566
4258411244, Average Fitness: 0.03408655340026831
min fit : 0.06240475578952642 , Generation 200 - Best Fitness: 0.062404755
78952642, Average Fitness: 0.03722999127934186
```



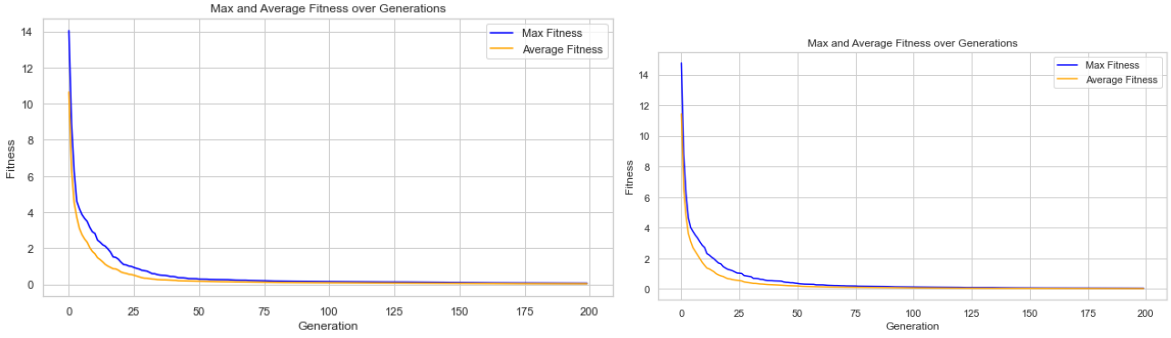
min fit : 0.04495650418506303 , Generation 200 - Best Fitness: 0.044956504
18506303, Average Fitness: 0.028036904783446184
min fit : 0.07844490121438286 , Generation 200 - Best Fitness: 0.078444901214
38286, Average Fitness: 0.04819669576918308



min fit : 0.07282500527818359 , Generation 200 - Best Fitness: 0.072825005
27818359, Average Fitness: 0.04620655383466712
min fit : 0.08158253144235461 , Generation 200 - Best Fitness: 0.081582531
44235461, Average Fitness: 0.05446056059622599

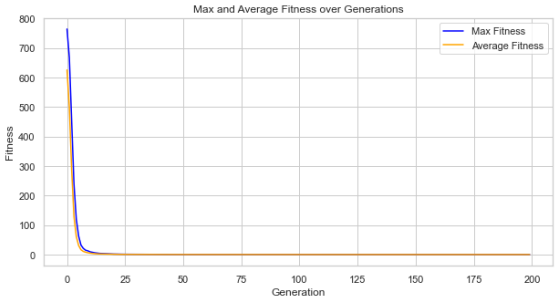


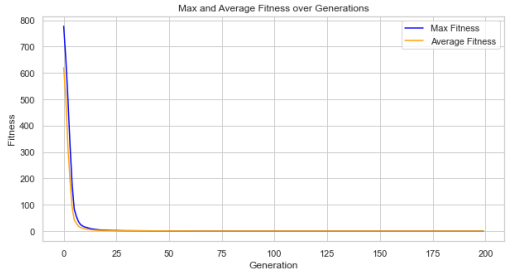
min fit : 0.05405976269825219 , Generation 200 - Best Fitness: 0.054059762
69825219, Average Fitness: 0.03178674379207967
min fit : 0.06450717148448293 , Generation 200 - Best Fitness: 0.064507171
48448293, Average Fitness: 0.03866976991937664



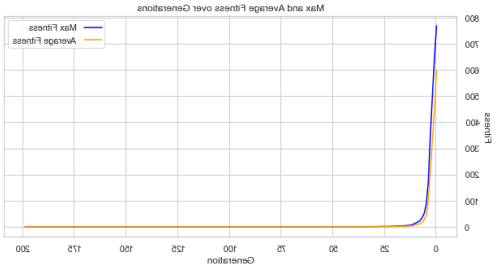
schwefel: Survival : elititstm, Mutate : non adebtive , Recombination : on , ndim 3

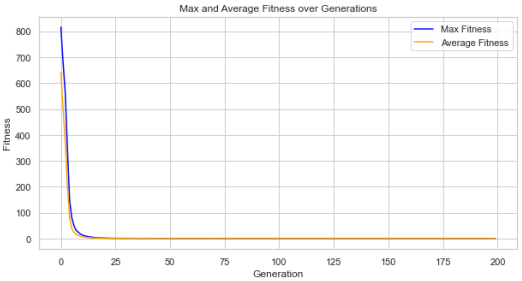
```
min fit : 0.12101439489765653 , Generation 200 - Best Fitness: 0.121014394
89765653, Average Fitness: 0.11146994042337383
min fit : 0.11320157275486054 , Generation 200 - Best Fitness: 0.113201572
75486054, Average Fitness: 0.10802607925211305
```



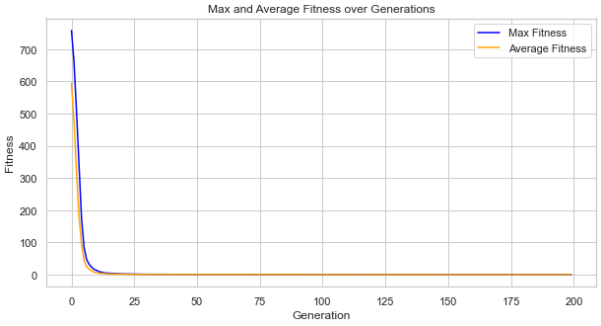


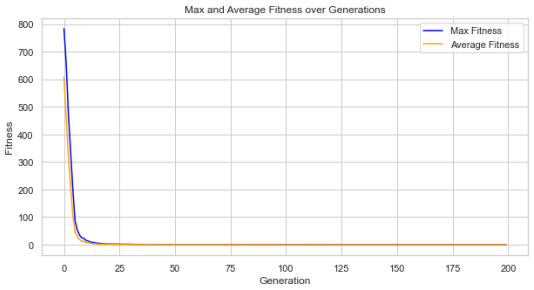

```
min fit : 0.11718781354579733 , Generation 200 - Best Fitness: 0.117187813
54579733, Average Fitness: 0.10949025472981021
min fit : 0.12023716062640233 , Generation 200 - Best Fitness: 0.120237160
62640233, Average Fitness: 0.11174264735107045
```



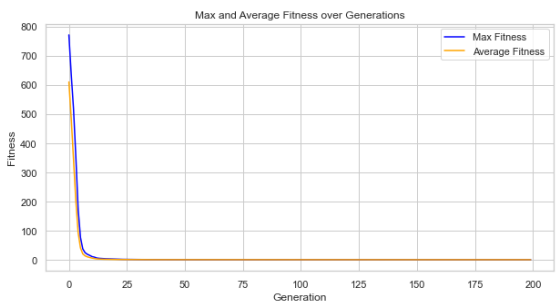


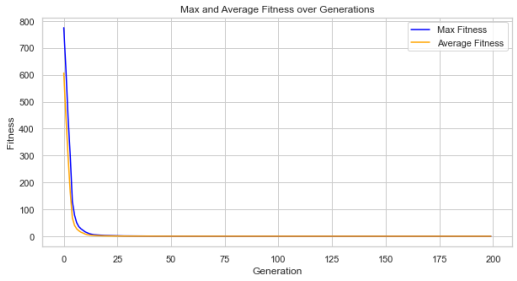

```
min fit : 0.1192397013480786 , Generation 200 - Best Fitness: 0.1192397013
480786, Average Fitness: 0.11041364387501516
min fit : 0.12043573154551268 , Generation 200 - Best Fitness: 0.120435731
54551268, Average Fitness: 0.11127211081194673
```



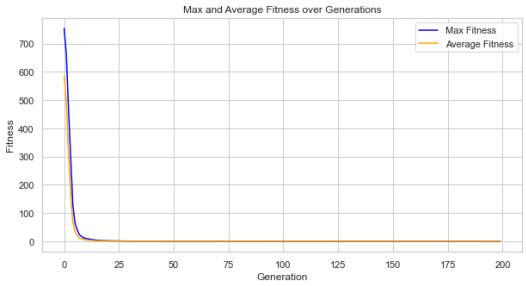


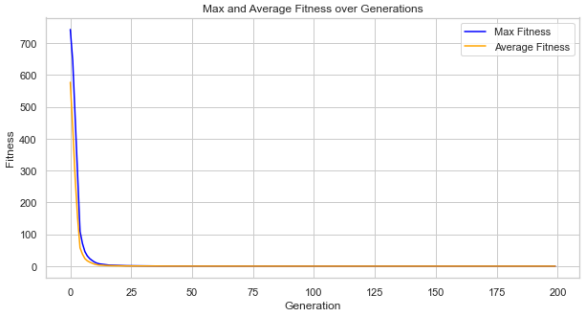
```
min fit : 0.11787740034606031 , Generation 200 - Best Fitness: 0.117877400
34606031, Average Fitness: 0.11025730908484775
min fit : 0.11714097616663821 , Generation 200 - Best Fitness: 0.117140976
16663821, Average Fitness: 0.10941944488881063
```






```
min fit : 0.12218503377675916 , Generation 200 - Best Fitness: 0.122185033
77675916, Average Fitness: 0.11182537425153896
min fit : 0.11589940469525573 , Generation 200 - Best Fitness: 0.115899404
69525573, Average Fitness: 0.10962625447504706
```

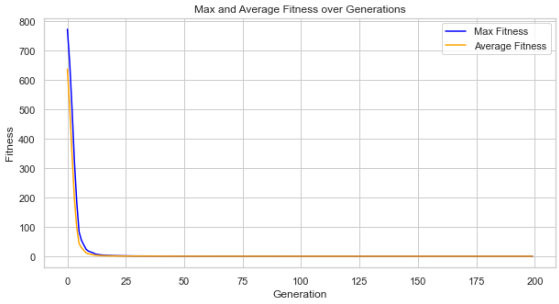


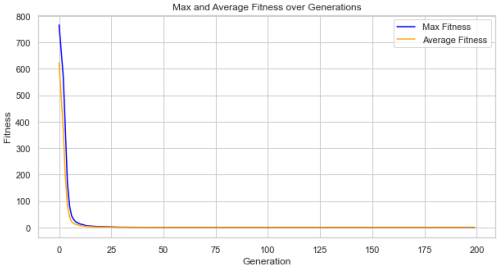



```
max fit average for 2 runs: 0.11844191897030214
avg fit average for 2 runs: 0.11035430591435738
```

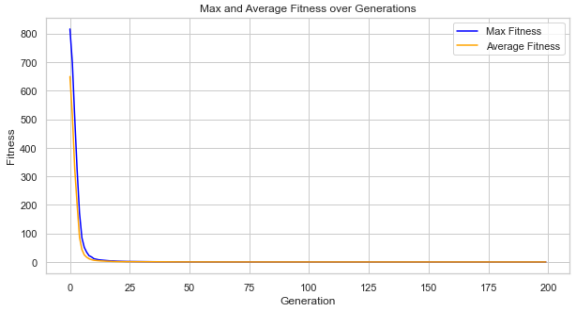
schwefel: Survival : elitistm, Mutate : non adebtive , Recombination : on , ndim 6

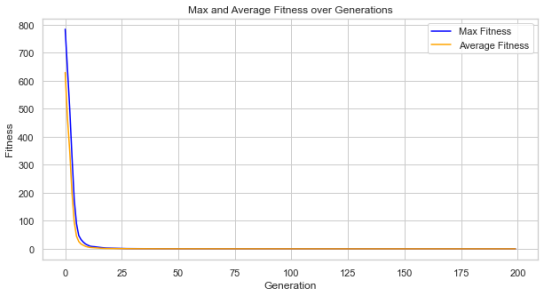
```
min fit : 0.12145701305507828 , Generation 200 - Best Fitness: 0.121457013
05507828, Average Fitness: 0.11147484620052638
min fit : 0.11643759707521895 , Generation 200 - Best Fitness: 0.116437597
07521895, Average Fitness: 0.1090117553195455
```



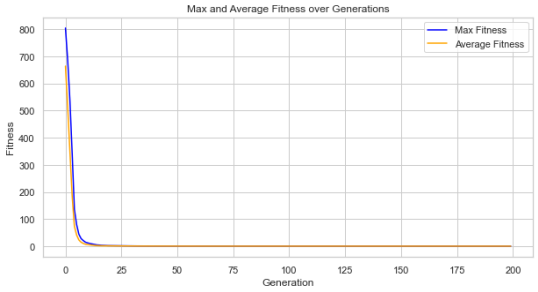


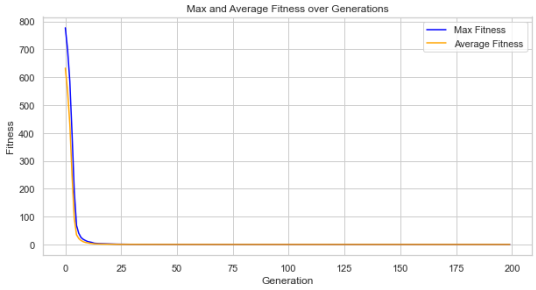
min fit : 0.11919899950044055 , Generation 200 - Best Fitness: 0.119198999
50044055, Average Fitness: 0.11024343128835198
min fit : 0.11746020641635369 , Generation 200 - Best Fitness: 0.117460206
41635369, Average Fitness: 0.10964251776594892



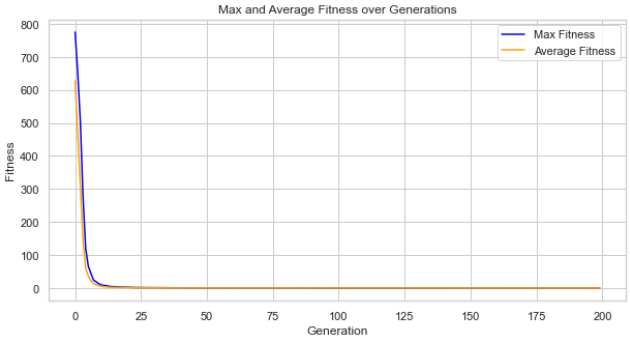


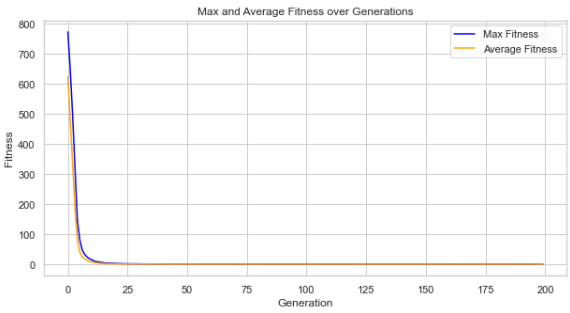
min fit : 0.11676988397766763 , Generation 200 - Best Fitness: 0.116769883
97766763, Average Fitness: 0.10908293096027818
min fit : 0.11637626169886062 , Generation 200 - Best Fitness: 0.116376261
69886062, Average Fitness: 0.10931448432125535





min fit : 0.11969748286924187 , Generation 200 - Best Fitness: 0.119697482
86924187, Average Fitness: 0.11013478421373975
min fit : 0.11571548791289388 , Generation 200 - Best Fitness: 0.115715487
91289388, Average Fitness: 0.10899034992186216





```
min fit : 0.1127592046227619 , Generation 200 - Best Fitness: 0.1127592046
227619, Average Fitness: 0.10822449584654123
min fit : 0.11479006813328851 , Generation 200 - Best Fitness: 0.114790068
13328851, Average Fitness: 0.10841415219187411
```