به نام خدا

هوش مصنوعی **پروژه اول** **نیمسال اول 1401-00**

نام و نام خانوادگی: شایان طلایی **شماره دانشجویی: ۹۸۱۰۹۵۴۸**

هدف پروژه

در این پروژه ما قصد داریم تا به کمک الگوریتم ژنتیک توابع ریاضی را تخمین بزنیم. گونه‌های ما در این پروژه درخت محاسبه توابع هستند که نسل به نسل غربال می‌شوند و آن‌هایی که با هدف ما سازگاری بیشتری دارند، به جا می‌مانند.

طراحی گونه

در طراحی درخت‌های توابع، از کلاس Node استفاده شده است، که در ادامه جزئیات طراحی آن را توضیح می‌دهیم.

برای اینکه برنامه ما نسبت به طراحی درخت و استفاده از توابع آن منعطف باشد، در ابتدا مفاهیم رئوس را در بالاترین سطح انتزاع کلاس بندی می‌کنیم.

هر راس، قابلیت محاسبه مقدار تابعی که خودش در ریشه آن قرار دارد، را باید داشته باشد. همچنین، فرم نوشتاری تابع مورد نظر رابابد برگرداند. میتوان از رئوس انتظار داشت که فرزندانش و یا تمام زیردرخت مربوط به خودش را برگرداند.

در ادامه دو کلاس برای رئوس تک عملوندی و دو عملوندی طراحی کرده‌ام. رئوس تک عملوندی پوینتر به تنها فرزندشان و رئوس دوعملوندی پوینتر به فرزند چپ و راستشان را نگه می‌دارند.

با کلاس‌های بالا همه شرایط محیاست تا برای تک تک عملگرهای ریاضی از یکی از دو کلاس بالا ارث ببریم. همان‌طور که در کد به جزئیات نوشته شده است، عملگرهای Constant, Parameter, Sinus, Cosinus، جزو عملگرهای Unary و Addition, Subtraction, Multiplication, Division, Power جزو عملگرهای دوعملوندی هستند.

طراحی دیتاست

برای ساخت دیتاست، تابع و بازه‌ای از دامنه را از ورودی می‌گیریم و به تعداد نقاط مشخص شده در بازه مورد نظر از نمودار تابع نقطه برمی‌داریم. با تابع add_dataset می‌توان مجموعه نقاط دو دیتاست را روی هم ریخت و در نتیجه از توابع چند ضابطه‌ای نمونه گیری کرد. اگر تابعی برای دیتاست مشخص نشود، نمونه‌های خود را به صورت تصادفی انتخاب می‌کند.

شیوه گزارش نتایج آزمایش‌ها

به ازای هر آزمایش انجام شده، قطعه کد مربوط به آن قرار داده شده است. همچنین در زیر آن نمودار تغییرات loss توابع در طول نسل‌ها وجود دارد. برای تست کارایی و نزدیکی جواب ما به تابع اصلی، تعداد نقاط زیادی را از نمودار هردو نمونه برداری کرده‌ایم. نقاط تابع اصلی با رنگآبی و تخمین تابع ما با رنگ نارنجی مشخص شده است. در نمودارهایی که رنگ آبی دیده نمی‌شود، در واقع تابع ما تا حد خوبی به تابع اصلی منطبق است. دقت کنید که بازه‌ای که برای تست انتخاب کرده‌ایم مطابق با رفتار تابع بسیار بزرگتر از بازه یادگیری انتخاب شده است.

طراحی درخت

با توجه به ساختار رئوسی که طراحی کردیم، هر درخت با شناختن ریشه خود و صدا زدن توابع آن، همه اطلاعات خود را خواهد داشت.

اما به عنوان گونه‌ای که باید مورد تکامل قرار بگیرد، باید نحوه جهش Mutation و Breed جفت‌گیری دو نمونه از آن تعریف شود. در ادامه برای هر ساختار درختی که طراحی کرده‌ایم این موارد را مشخص می‌کنیم.

درخت XPolyTree

با الهام گیری از بسط تیلور توابع پیوسته، می‌خواهیم به کمک چندجمله‌ای توابع را تخمین بزنیم.
این نوع درخت با داشتن درجه و بازه ضرایب چندجمله‌ای، ساخته می‌شود. ضرایب در آن بازه به صورت یکنواخت نمونه برداری می‌شوند.

- شیوه Breed

در ورودی، دو چندجمله ای با درجه برابر خواهیم داشت. فرزند آن‌ها به گونه‌ای تعیین می‌شود که هر ضریب آن با احتمال برابر از یکی ازوالدین انتخاب می‌شود.

- چالش و نحوه رفع آن

در ابتدا، با توجه به میزان loss دو درخت، میانگین وزن دار ضرایب را می‌گرفتم ولی با توجه به غیریکنوانت بودن loss ها، عملا تاثیر والدضعیف از بین می‌رفت. سپس با توابع سیگموید loss را نرمال کردم ولی در این حال هم عملا برای والدین در یک نسل وزن ضرایب برابرمی‌شد. سپس میانگین گیری معمولی را جایگزین کردم. اشکال این روش، در این بود که مجموعه فرزندان قبل از جهش در پوش محدبفضای تشکیل شده توسط والدین قرار می‌گرفت. در نهایت از روش بالا برای جفت‌گیری این درختان استفاده کردم.

- شیوه Mutation

یکی از ضرایب چندجمله‌ای را به صورت تصادفی انتخاب و با مقدار تصادفی کوچکی جمع یا از آن کم می‌کنیم.

- چالش و نحوه رفع آن

در ابتدا برای جهش یک گونه، مقداری تصادفی برای تمام ضرایب آن در نظر می‌گرفتم و آن‌ها را تغییر می‌دادم. اما به تجربه فهمیدم کههرچه تغییرات کوچک‌تر باشد، احتمال پیشرفت و پیدا کردن گونه بهتر، بیشتر خواهد بود.

- آزمایش‌ها

در تست‌هایی که از این شیوه مشاهده کردم، به نظر الگوریتم خیلی هوشمندانه ابتدا ضریبی را تنظیم می‌کرد که در رفتار آن تابع در بازهمشخص شده تاثیر بیشتری داشت.
در آزمایش‌هایی که چندجمله‌ای را تقریب میزدیم، تمام ضرایب تا حد خوبی به ضرایب چندجمله‌ای منطبق شده اند.
در ادامه تابع سینوس را به کمک چمدجمله‌ای تقریب زده‌ایم که همان طور که میبینید، ضریب هر جمله به سمت بسط تیلور سینوس میلکرده است. برای افزایش دقت، می‌توان تعداد نسل‌ها و یا بازه ای که برای یادگیری مشخص می‌کنیم را افزایش دهیم.

نام و نام خانوادگی: شایان طلایی | شماره دانشجویی: ۹۸۱۰۹۵۴۸

درخت ComplexTree

این درخت را با هدف امکان تولید همه توابع ممکن با عملیات‌های گفته شده طراحی شده است. یکی از ایده‌های اصلی در تمام مراحل ساخت این درخت، استفاده از انتخاب‌های تصادفی است. هر جا که از متغیر coin استفاده شده است، در واقع یک تصمیم تصادفی گرفته می‌شود که با توجه با میزان مناسب بودن هر تصمیم، احتمال آن پیشامد را تنظیم کرده‌ام. مزیت استفاده از این شیوه برای تصمیمگیری این است که همه حالت‌های ممکن که بر سر دوراهی قرار می‌گیریم امکان به وجود آمدن خواهند داشت و در صورت نامطلوب بودندر نسل بعدی حذف می‌شوند. علاوه بر آن، از یک شکل شدن نسل بعد جلوگیری می‌کند.

- شیوه Breed

در صورت بچه نداشتن، یا با سکه‌ای که می‌اندازیم، بچه تولید شده، یک راس دوعملوندی در بالای ریشه‌های والدین خواهد بود. در صورتی که یکی از والدین بیش از حد پیچیده باشد (بیش از ۱۵ راس)، بچه آن‌ها یکی از دو سمت آن خواهد بود. در غیراین صورت، یکی از دو ریشه را تصادفا انتخاب می‌کنیم و از هر یک از والدین، یک راس بچه در زیر آن قرار می‌دهیم. باز هم به صورت تصادفی، فرزندان را انتخاب می‌کنیم. البته به این نکته توجه می‌کنیم که فرزندی که از یک ریشه می‌آید، از همان سمت انتخاب شود.

- عناصر سازنده درخت :Simplex

از آنجایی که رئوس اغلب برای تعریف شدن به متغیر و یا عدد ثابت نیاز دارند، کوچکترین اجزای سازنده را به صورت Simplex تعریف می‌کنیم. این اجزا می‌توانند، ثابت، متغیر، سینوس، کسینوس و یا اعمال یک عملگر دوتایی روی یک متغیر و عدد ثابت باشند. مجددا نحوه انتخاب Simplex تماما با انداختن سکه‌های ورن دار انجام می‌شود.

- چالش‌ها و نحوه رفع آن:

در ابتدا Simplex ها را درختان با عمق حداکثر سه تعریف کرده بودم ولی با کم پیرو نتیجه قبل، هرچه عناصر و تغییرات ساده تر باشند ساختن به کمک آن‌ها بهتر می‌شود. تابع تقسیم، در اثر جهش ممکن بود مخرجی پیدا کند که به لحاظ پارامتری صفر باشد، احتمال وقوع آن را کم کردم و مخرج را عدد ثابت انتخاب کردم. در توابع تست شده مشکل بالا به وجود نیامد. تابع توان، در صورت ورودی منفی و توان اعشاری، به عدد مختلط منجر می‌شد که آن را از تعریف توان حذف کردم، در توابع تست شده، برنامه به کمک ضرب و استفاده از ضرایب بسط تیلور، آن‌ها را می‌ساخت.

- شیوه Mutation

برای جهش دادن یک درخت، از بالا به صورت تصادفی به سمت عمق می‌رویم. در هر مرحله سکه می‌اندازیم، تا مشخص کنیم پایین ترمی‌رویم یا خیر و سمت پایین رفتن را مشخص کنیم. و یا اینکه راسی برای پایین رفتن وجود نداشته باشد.
در نتیجه فرایند بالا، راسی که می‌خواهیم تغییر دهیم، انتخاب می‌شود. اگر راس عدد ثابت بود یا مقدار آن را کمی تغییر می‌دهیم و یا مثل بقیه راس‌ها با آن رفتار می‌کنیم. برای تغییر یک راس، آن را با یک Simplex جایگزین می‌کنیم.

- چالش ها و نحوه رفع آن‌ها:

در نتیجه زاد و ولد درختان، فرزندان بزرگتری به وجود می‌آمد و به دلیل پیچیدگی محاسباتی الگوریتم کند می‌شد. با اضافه کردن محدودیت پیچیدگی این مشکل برطرف شد.

شیوه تکامل Evolution

الگوریتم تکاملی پیاده‌سازی شده در هر نسل سه روند Selection، CrossOver، Mutation را طی می‌کند تا نسل بعد را بسازد. شرط پایان تکامل یا رسیدن به حداکثر تعداد نسل‌های مشخص شده است و یا پیدا کردن گونه‌ای که loss آن از ۰,۰۰۰۱ کمتر باشد.

- تابع ارزیاب

برای برسی عملکرد هر درخت از تابع MSELoss روی تمام دیتاست استفاده می‌کنیم. به گونه‌ای که میانگین توان دو فاصله جواب تابع درهر نقطه، تا میزان واقعی محاسبه می‌شود. در نهایت از جواب رادیکال گرفته می‌شود.

- شیوه Selection

ابتدا میزان loss تمام نسل محاسبه می‌شود. سپس افراد بر اساس میزان loss مرتب شده و یک نسبتی از کل آن‌ها selection_ratio که به عنوان پارامتر اجرا مشخص می‌شود، انتخاب می‌شود. در ادامه میزان loss افراد به کمک تابع Sigmoid نرمال سازی می‌شوند.

- شیوه Cross Over

به میزان مشخص شده برای نسل بعد، در هر گام دو والد متناسب با احتمال‌های نرمال شده، انتخاب می‌شوند و تابع Breed آن گونه رویشان صدا زده می‌شود تا فرزندشان برای نسل بعد تولید شود.
ممکن است نسبتی از نسل بعد را با گونه‌های ابتدایی پر کنیم، تا از منقرض شدن آن‌ها جلوگیری کنیم.

- شیوه Mutation

روی تمام فرندان نسل جدید تابع mutation آن گونه صدا زده می‌شود.

جمع بندی و نتایج عملی

در ادامه توابع مختلفی را از برنامه خواسته‌ایم که تخمین بزند و جزئیات هایپر پارامتر ها و عملکرد برنامه را مشخص کرده‌ایم.
ابتدا توابع تک متغیر متفاوت را بررسی کرده ایم. در بین تست‌ها تابع دو ضابطه‌ای و تابع رندوم نیز بررسی شده است. در ادامه برای اینکه قدرت الگوریتم را محک بزنیم، تخمین توابع دو متغیره را برسی کردیم.
جزئیات هر آزمایش در کنار آن مشخص شده است.

نتیجه گیری

منطق اصلی بهبود نسل‌ها، جست‌وجوی شبه تصادفی در بین گونه‌های برتر و حذف گونه‌های ضعیف است.
با تکیه بر منطق بالا، هرچه جمعیت اولیه و تعداد نسل‌ها را بیشتر کنیم، احتمال پدید آمدن یک گونه برتر بیشتر خواهد شد.
با توجه به تغییراتی که در برنامه دادم، متوجه شدم که هرچه تغییرات و جهش‌ها ساده تر باشند، نسل‌ها با احتمال بیشتری نسبت به نسل قبل خود پیشرفت می‌کنند.

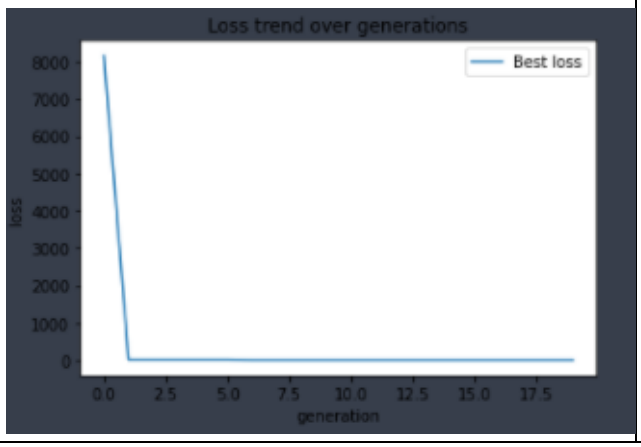
البته بعد از گذشتن نسل‌های زیاد، جمعیت حول گونه برتر متمرکز می‌شود و اگر با جست‌وجوهای محلی نتوان به جواب رسید، الگوریتم قفل می‌شود. برای حل مشکل بالا بهتر است، در صورت گذشت چند نسل و پیشرفت نکردن گونه‌ها، الگوریتم را از ابتدا شروع کنیم تا با یک مجموعه اولیه جدید، شانس پیدا کردن جواب را امتحان کنیم.

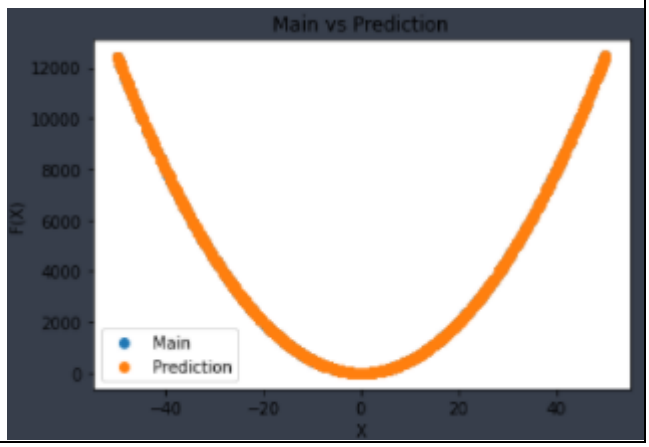
1: $F(x) = 5x^2$

```
func = Mul(Const(5), Pow(Param(0), Const(2)))
dataset = Dataset(50, mul, 1, (-10, 10))
population = [XpolyTree(2, (3, -3)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, XpolyTree.breed, XpolyTree.mutate)
trainer.evolute(20, 0.4)

trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: 5.0 * (x ^ 2) + -0.28 * (x ^ 1) + -1.12
Loss: 3.3409722842202147



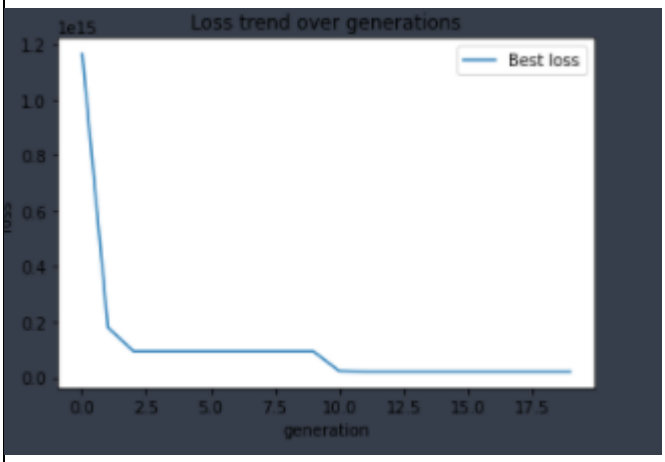


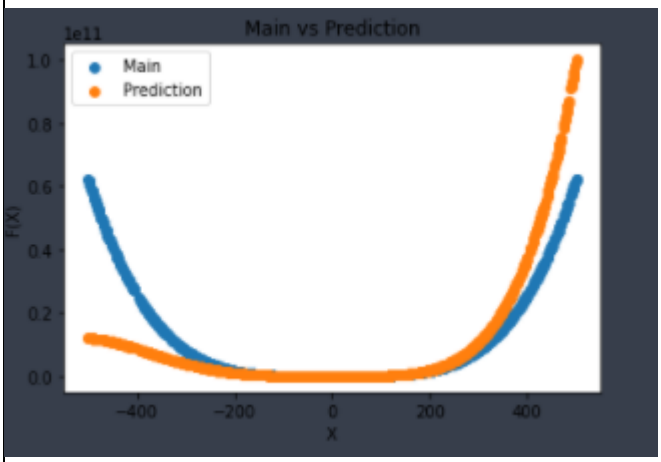
2: $F(x) = x^4 + 2x^2 - 3$

```
func = Add(Pow(Param(0), Const(4)), Add(Mul(Const(2), Pow(Param(0), Const(2))), Const(-3)))
dataset = Dataset(100, func, 1, (-100, 100))
population = [XpolyTree(5, (3, -3)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, XpolyTree.breed, XpolyTree.mutate)
trainer.evolute(20, 0.1)

trainer.show_trend()
compare(func, trainer.best_sample, (-500, 500))
```

Best sample: 0.0 * (x ^ 5) + 0.9 * (x ^ 4) + -2.65 * (x ^ 3) + -1.66 * (x ^ 2) + 2.29 * (x ^ 1) + -0.98
Loss: 20808208165186.684



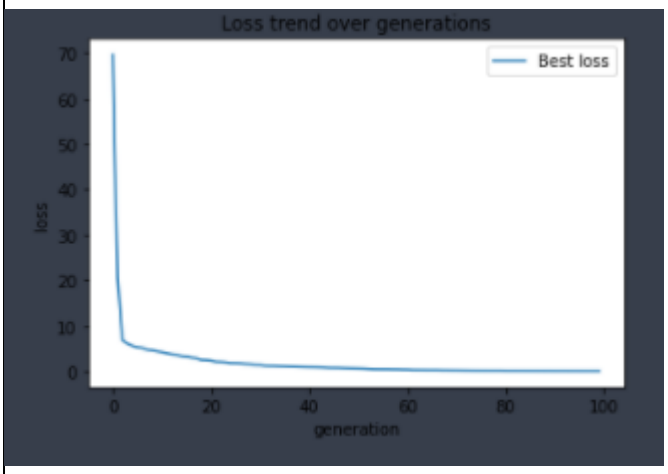


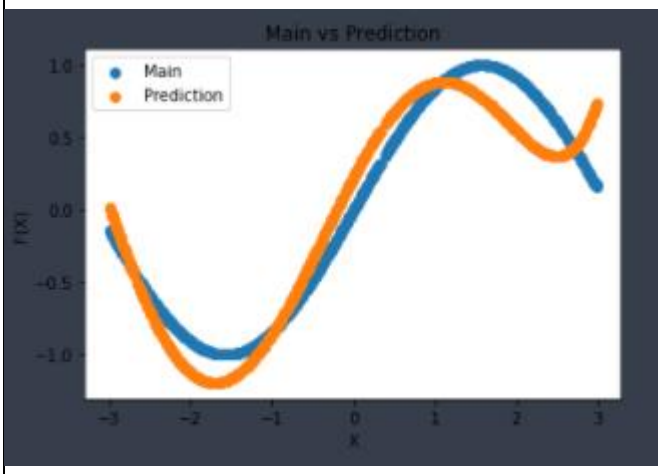
3: $F(x) = \sin(x)$

```
func = Sin(Param(0))
dataset = Dataset(100, func, 1, (-3.14, 3.14))
population = [XpolyTree(5, (3, -3)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, XpolyTree.breed, XpolyTree.mutate)
trainer.evolute(100, 0.1)

trainer.show_trend()
compare(func, trainer.best_sample, (-1, 1))
```

Best sample: 0.01 * (x ^ 5) + 0.03 * (x ^ 4) + -0.21 * (x ^ 3) + -0.25 * (x ^ 2) + 1.06 * (x ^ 1) + 0.23
Loss: 0.05115290351814221

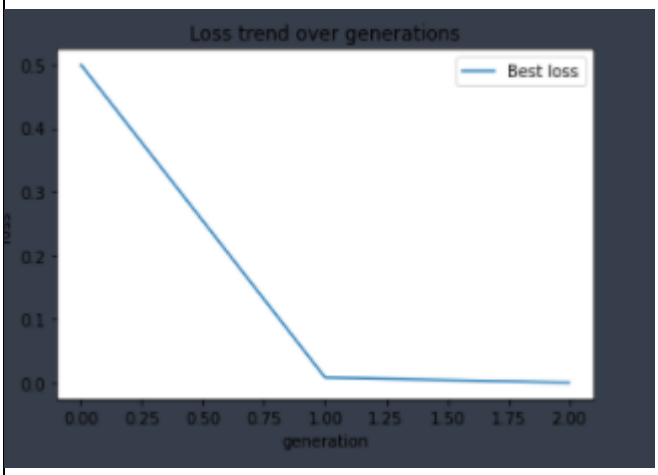


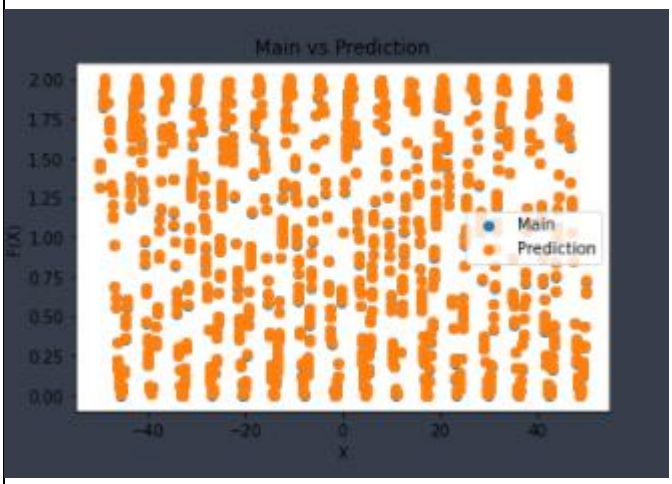


4: $F(x) = \sin(x) + 1$

```
func = Add(Sin(Param(0)), Const(1))
dataset = Dataset(50, func, 1, (-10, 10))
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(20, 0.1)
# trainer.evolute(20, 0.01)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: sin(-0.96) + 0.91 + sin(x) + 0.91
Loss: 1.0480092049044508e-05

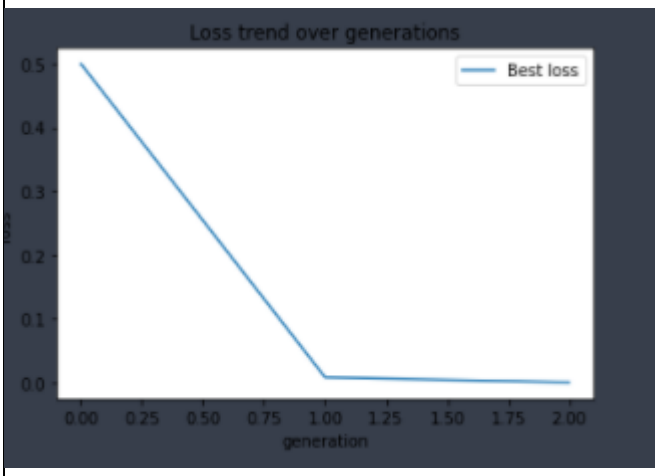


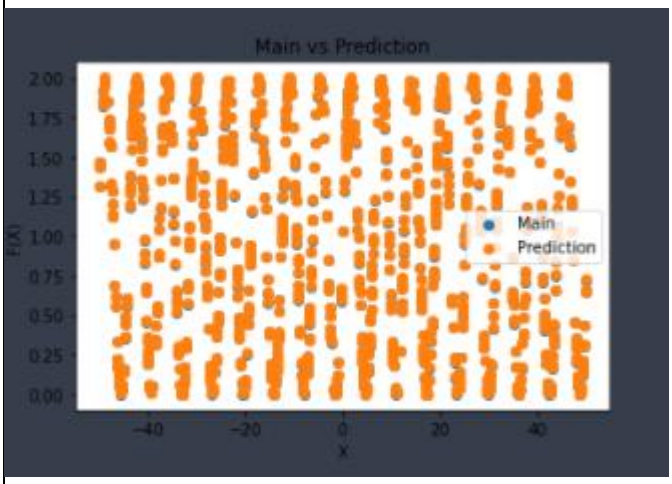


5: $F(x) = \sin(x) + \cos(x)$

```
func = Add(Sin(Param(0)), Cos(Param(0)))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(5, 0.1)
# trainer.evolute(20, 0.01)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: cos(x - 0.57 + -1.0) + cos(x)
Loss: 5.518427663390709e-07

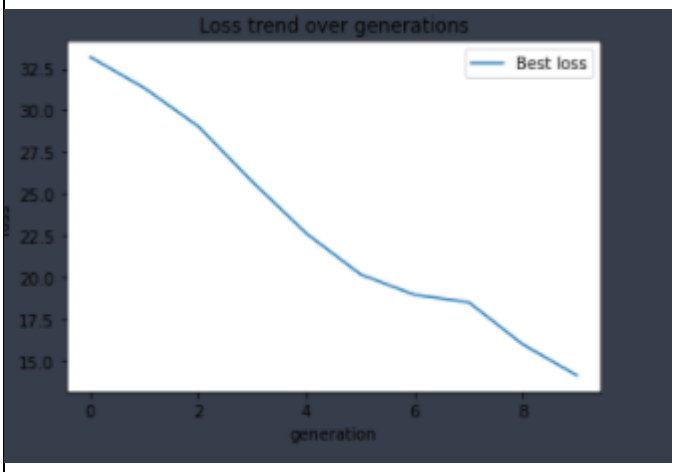


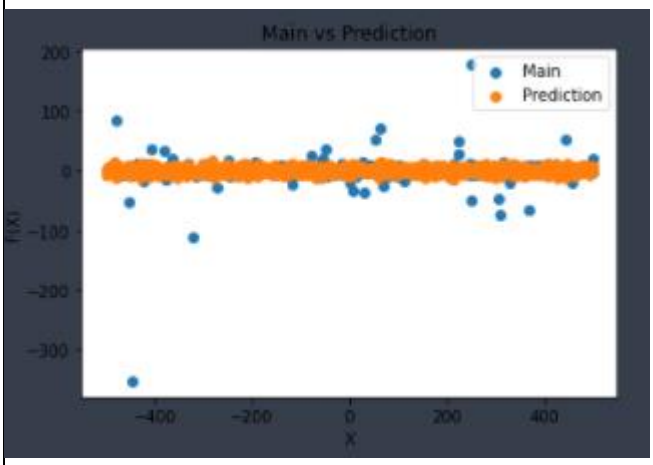


6: $F(x) = \sin(x)/\cos(x)$

```
func = Div(Sin(Param(0)), Cos(Param(0)))
dataset = Dataset(50, func, 1, (-100, 100))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(1000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.4)
trainer.show_trend()
compare(func, trainer.best_sample, (-500, 500))
```

Best sample: sin(x * 0.12) + sin(x) + sin(x * 0.13) + sin(x) + sin(x * 0.13) + sin(x) + sin(x * x) + sin(x)
Loss: 525.4847904046533
Number of generations: 10
Number of examinations: 4000.0
Execution time: 4.73558497428894

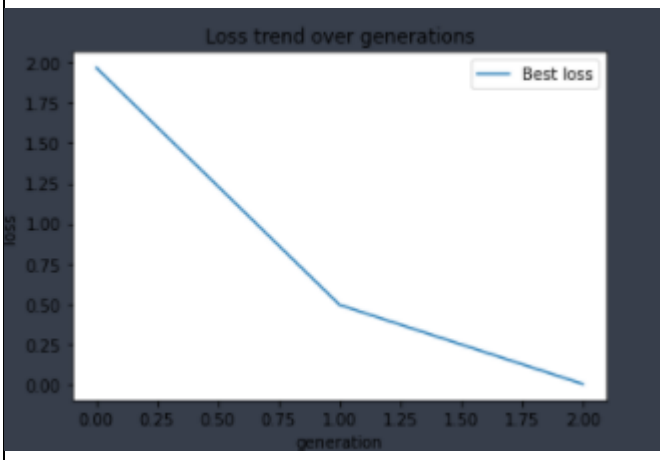


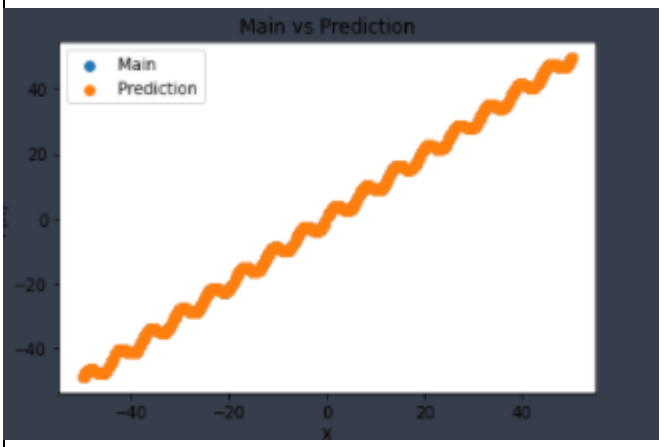


7: $F(x) = x + 2\sin(x)$

```
func = Add(Param(0), Mul(Const(2), Sin(Param(0))))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(20, 0.1)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: x + sin(x) + 0.0 + sin(x)
Loss: 1.3113569160915428e-05

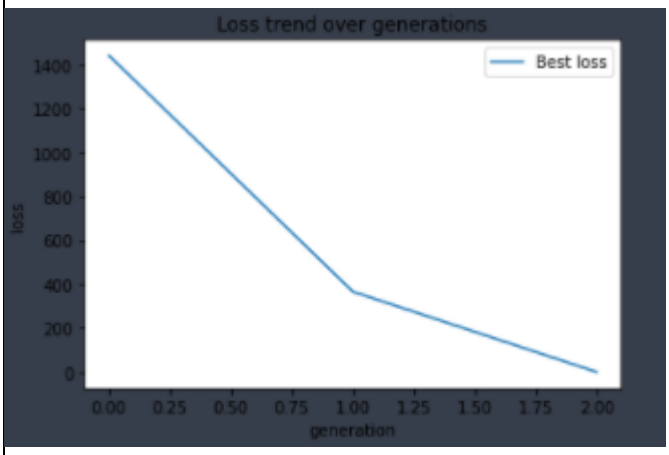


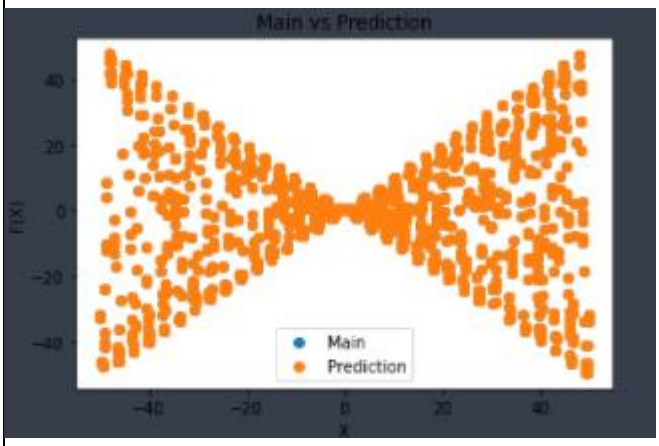


8: $F(x) = x\sin(2x)$

```
func = Mul(Param(0), Sin(Mul(Const(2), Param(0))))
dataset = Dataset(50, func, 1, (-100, 100))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.1)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: (x * cos(x)) * sin(x) + (x * cos(x)) * sin(x)
Loss: 2.524354896707238e-29
Number of generations: 1
Number of examinations: 500.0
Execution time: 1.910367727279663

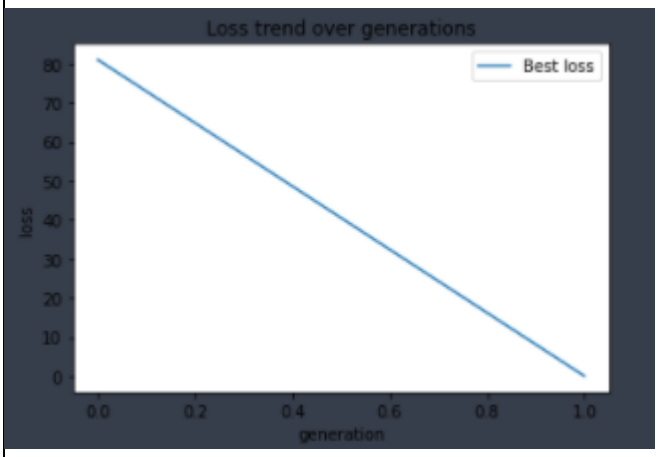


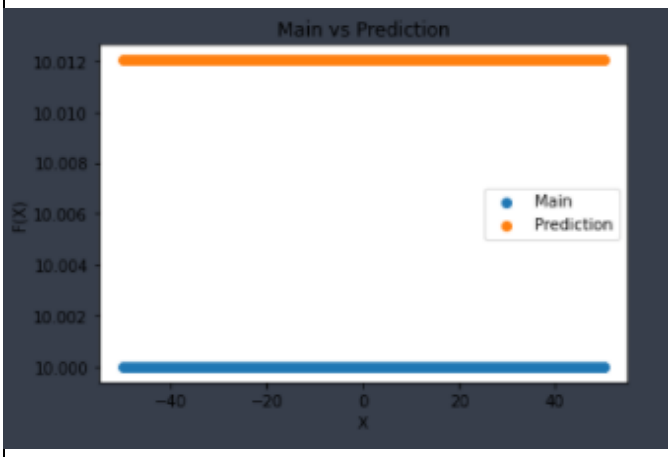


9: $F(x) = 10$

```
func = Const(10)
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(20, 0.1)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: 1.0 + 9.02
Loss: 0.00014510096358425276
Number of generations: 3
Number of examinations: 6000.000000000001
Execution time: 6.697886943817139

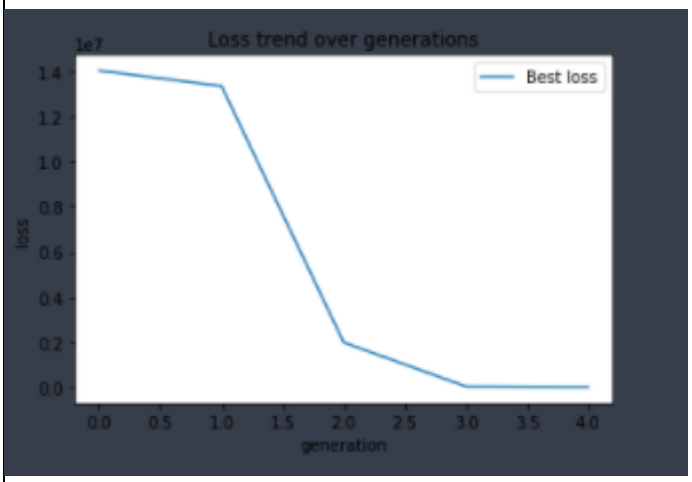


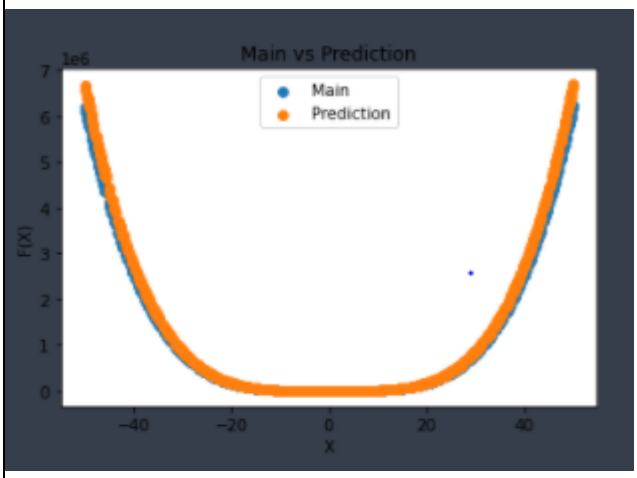


10: $F(x) = -200$

```
func = Const(-200)
dataset = Dataset(50, func, 1, (-100, 100))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(5, 0.4)
trainer.show_trend()
compare(func, trainer.best_sample, (-500, 500))
```

Best sample: ((x - (-0.81 - x)) * (x - 2.03)) * (((x * -0.46) - (0.4 - x)) * (x - -2.34))
Loss: 6043.995977156814

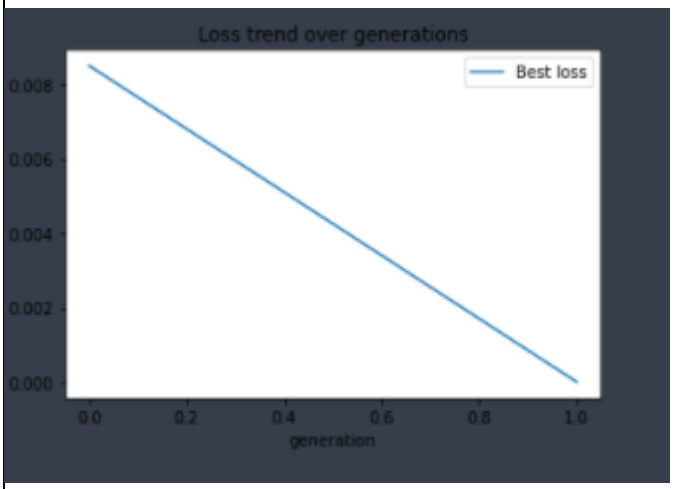


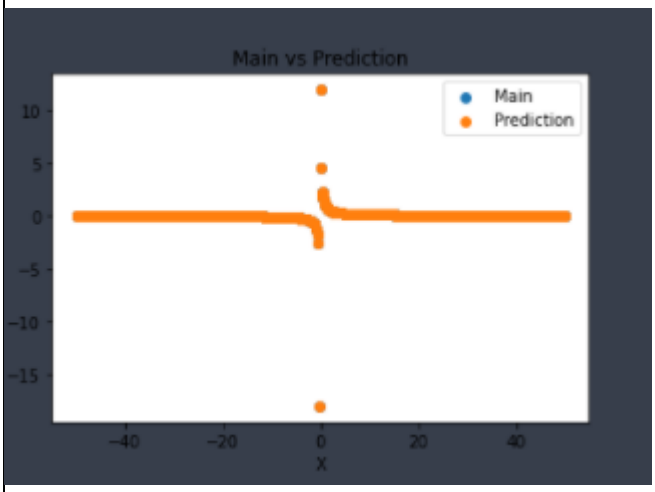


11: $F(x) = -1/x$

```
func = Div(Const(1), Param(0))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(5000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(5, 0.4)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: 1.0 / x
Loss: 1.1599300390642692e-05

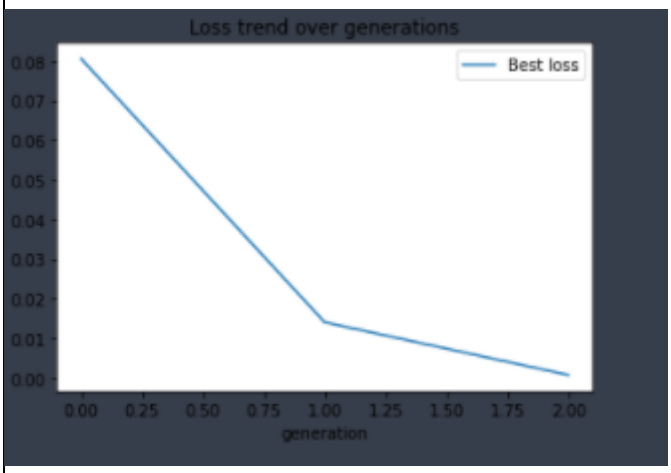


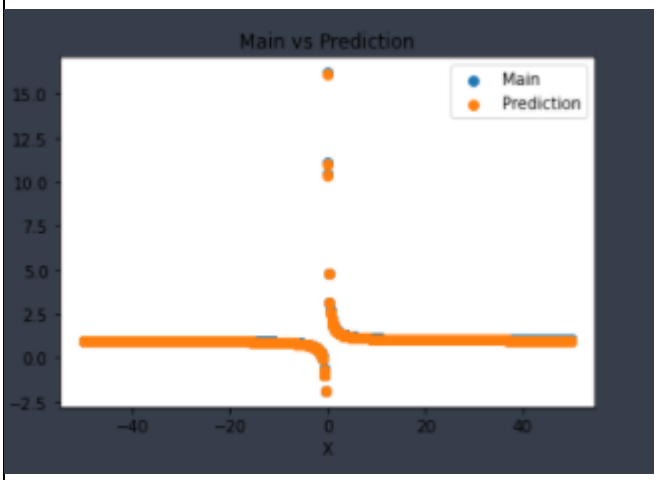


12: $F(x) = (x + 1)/x$

```
func = Div(Add(Param(0), Const(1)), Param(0))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: 0.97 + 1.0 / x
Loss: 0.0007462879218998378

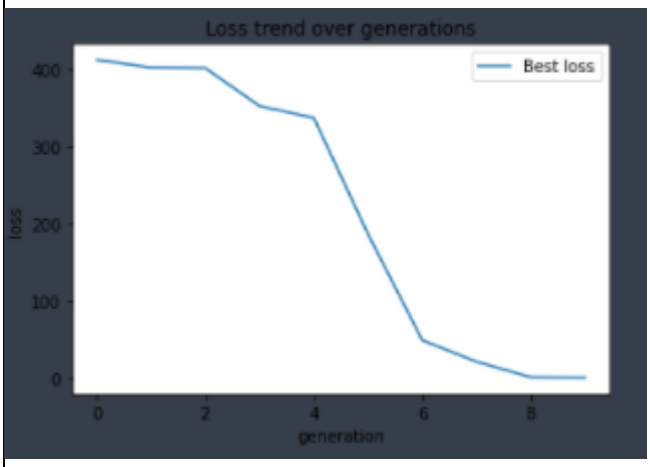


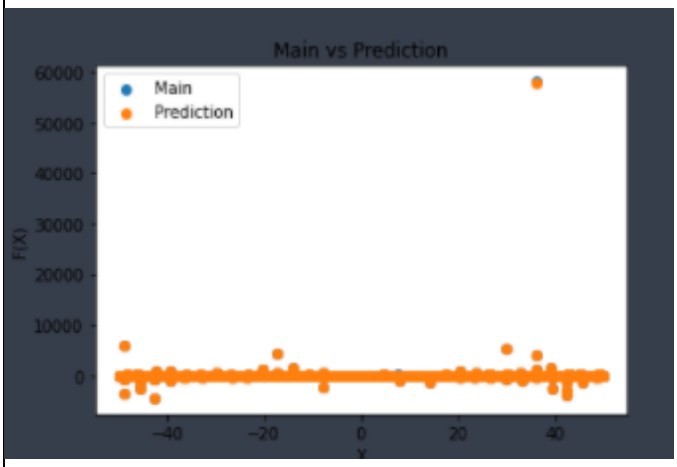


13: $F(x) = (x + 1)/\cos(x)$

```
func = Div(Add(Param(0), Const(1)), Cos(Param(0)))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: (x + 0.86) / cos(x) + cos(x) * (sin(x) + 0.74) + cos(x) * (x + (-0.89 - x) - -0.62)
Loss: 0.7202603019524146

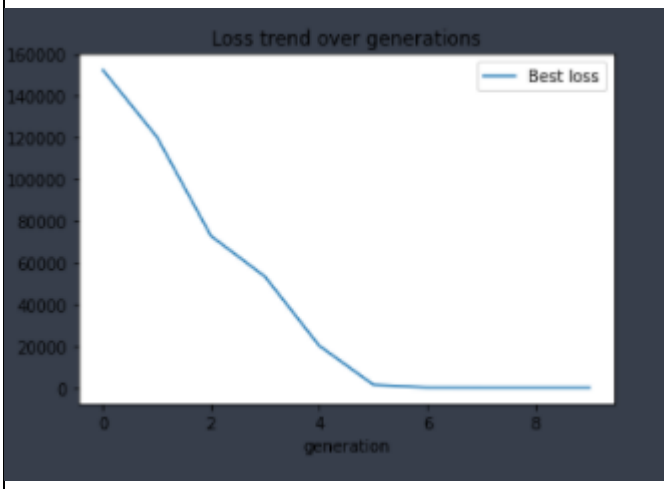


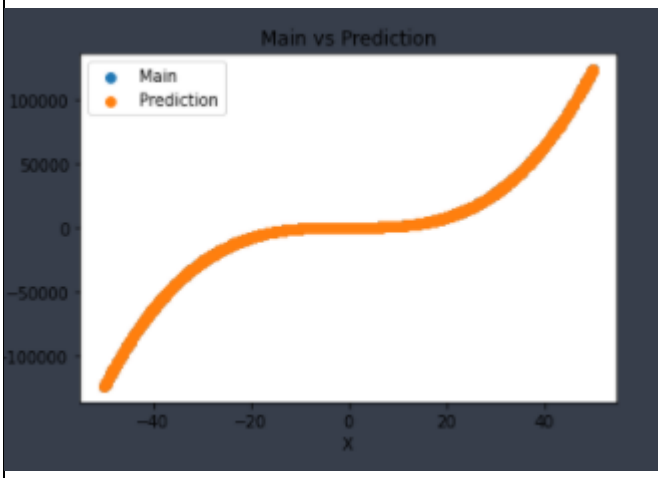


14: $F(x) = x^3$

```
func = Pow(Param(0), Const(3))
dataset = Dataset(50, func, 1, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50))
```

Best sample: ((x - 0.85) - -1.95) * ((x * (x + -0.12)) - ((x - 0.95) - -0.07))
Loss: 4.727760535015854

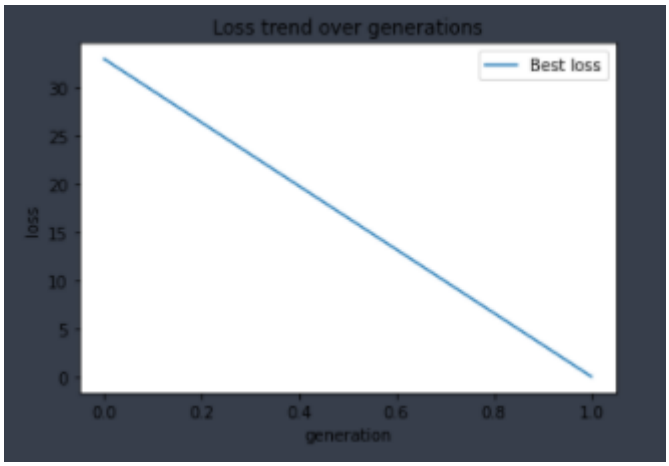


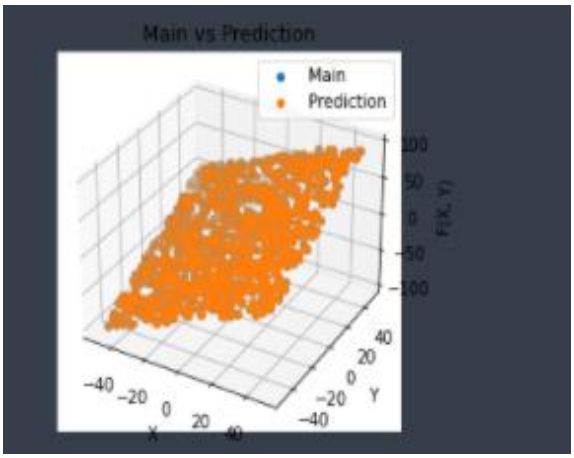


15: $F(x) = x + y$

```
func = Add(Param(0), Param(1))
dataset = Dataset(500, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-50, 50))
```

Best sample: y + x
Loss: 0.0

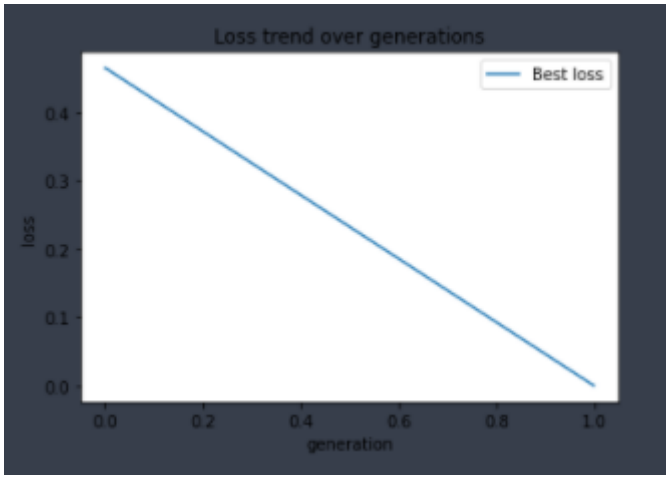


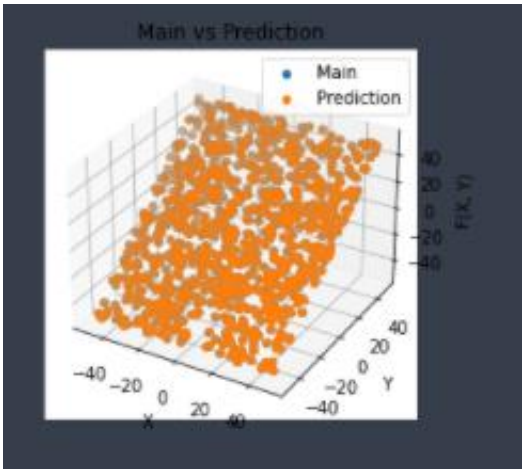


16: $F(x) = \sin(x) + y$

```
func = Add(Sin(Param(0)), Param(1))
dataset = Dataset(500, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-50, 50))
```

Best sample: sin(x) + y
Loss: 0.0

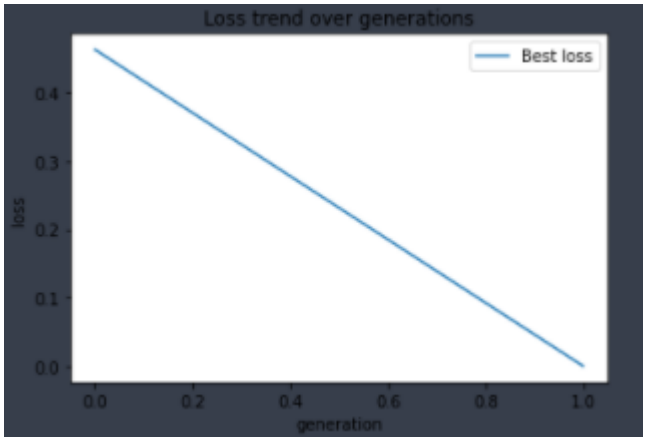


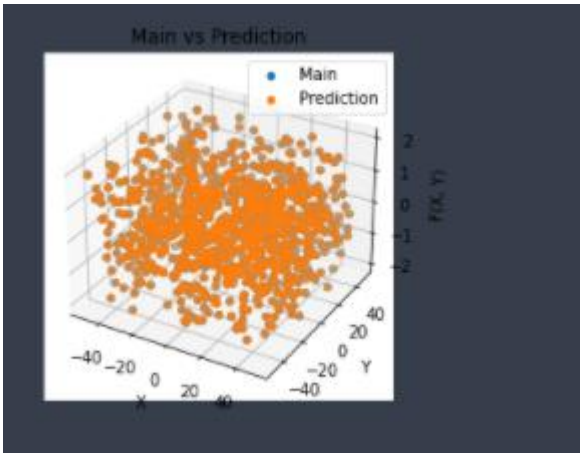


17: $F(x) = \sin(x) + \sin(y)$

```
func = Add(Sin(Param(0)), Param(1))
dataset = Dataset(500, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-50, 50))
```

Best sample: sin(x) + sin(y)
Loss: 0.0

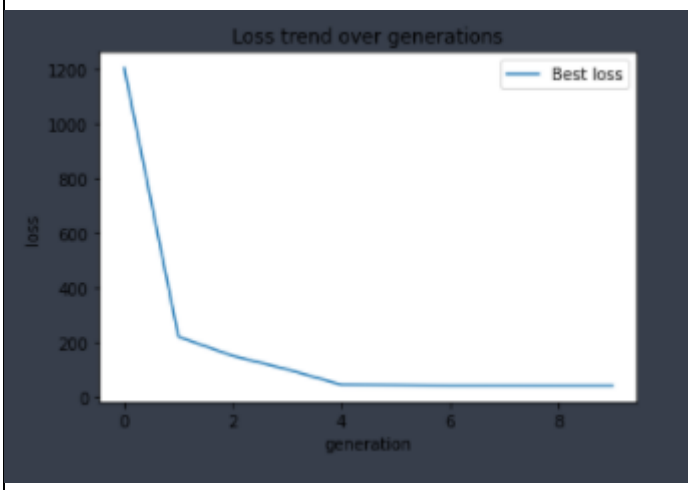


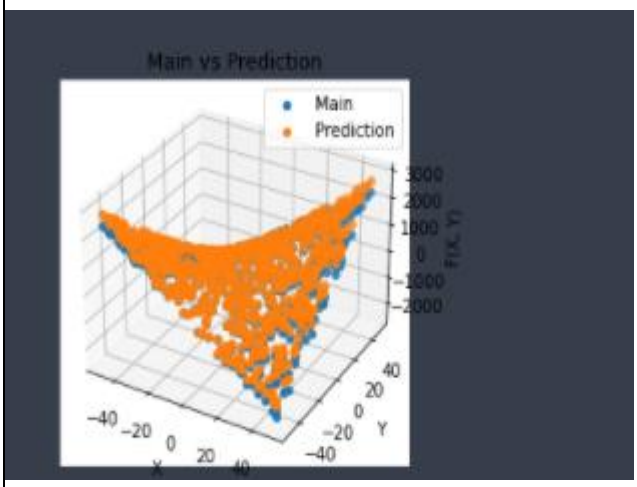


18: $F(x) = xy + 10$

```
func = Add(Mul(Param(0), Param(1)), Const(10))
dataset = Dataset(500, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.3)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-50, 50))
```

Best sample: (y - ((x - ((x + x * 0.83) * -0.13)) * -0.14)) * (((x - -0.08) - 2.06) - -1.93)
Loss: 43.864410955905164

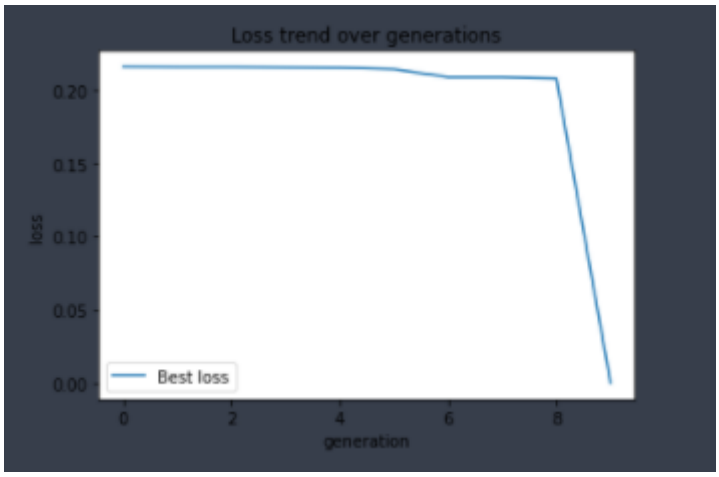


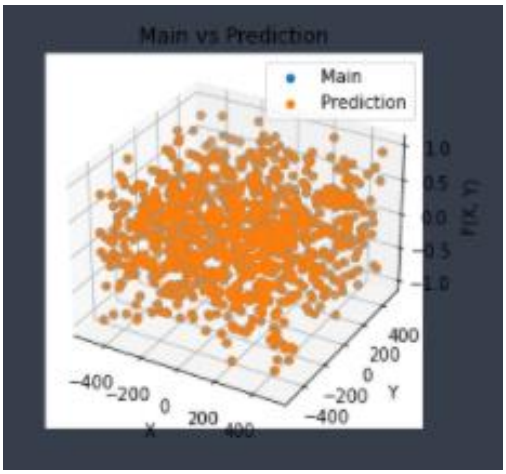


19: $F(x) = \sin(x) + \cos(y)$

```
func = Mul(Param(0), Sin(Param(1)))
dataset = Dataset(50, func, 2, (-7, 7))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(15, 0.3, basic_constrcutor= ComplexTree)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-500, 500))
```

Best sample: cos(y) * sin(x)
Loss: 0.0

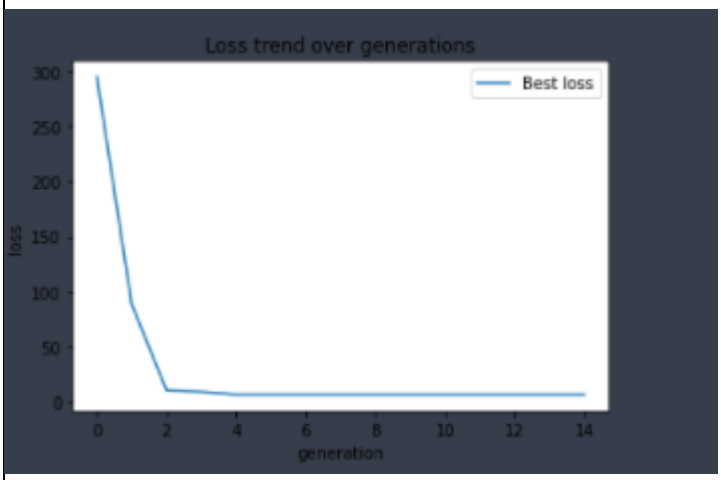


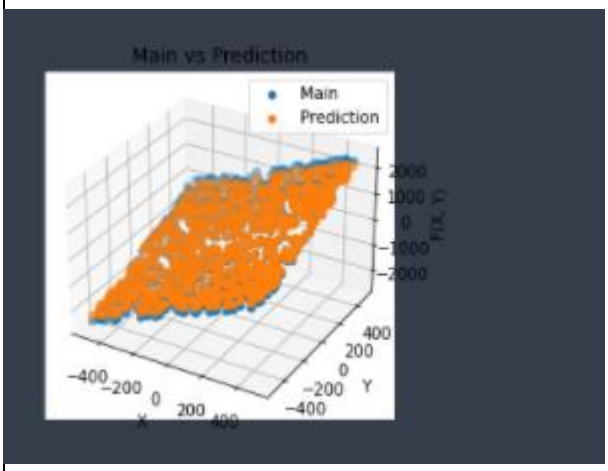


20: $F(x) = 2.65x + 2.41y$

```
func = Add(Mul(Const(2.65), Param(0)), Mul(Const(2.41), Param(1)))
dataset = Dataset(100, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(15, 0.3, basic_constrcutor= ComplexTree)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-500, 500))
```

Best sample: x + x + -0.66 + y - -1.22 + y - (0.78 - (x * 0.63))
Loss: 6.288078134969598

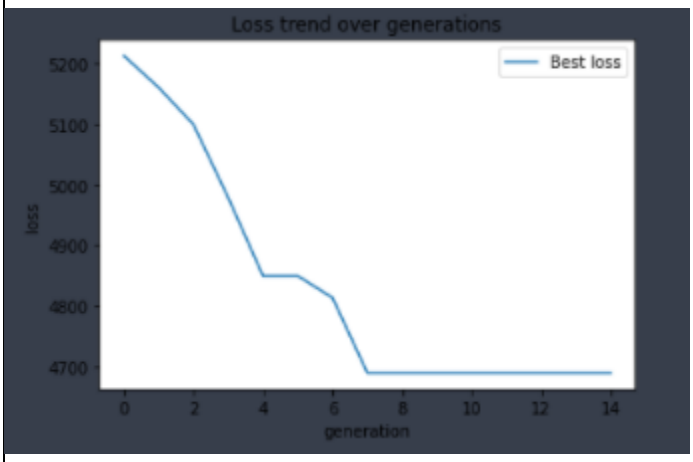


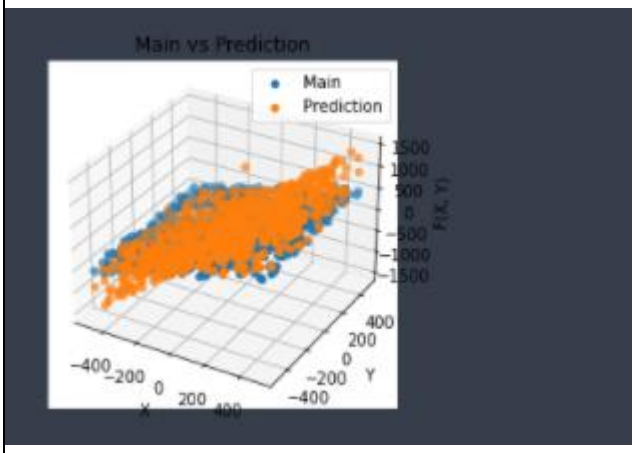


21: $F(x, y) = x + 100\sin(xy)$

```
func = Add(Param(0), Mul(Const(100), Sin(Mul(Param(0), Param(1)))))
dataset = Dataset(100, func, 2, (-10, 10))
data = dataset.point_value[0]
population = [ComplexTree((-2, 2), dim= 2) for _ in range(1000)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(15, 0.3, basic_constrcutor= ComplexTree)
trainer.show_trend()
compare_3d(func, trainer.best_sample, (-500, 500))
```

Best sample: ((9.55 / (x * (x + 0.8))) - -2.0) * (x - -2.0) + 12.32 / (0.74 - x) + ((1.47 / x) - sin(x)) * (x - -4.96)
Loss: 4689.619799997204

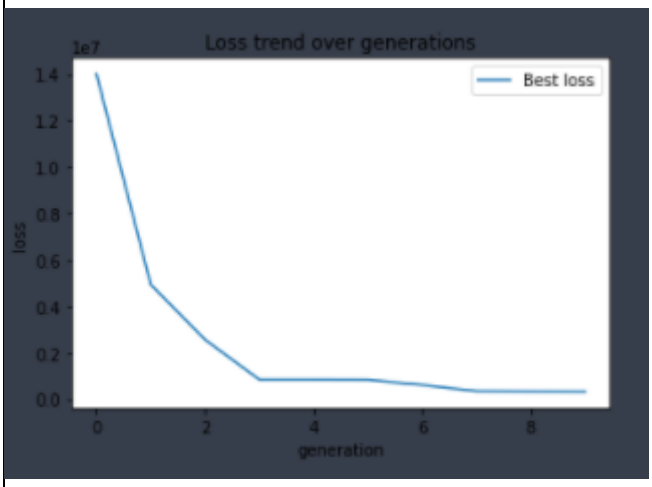


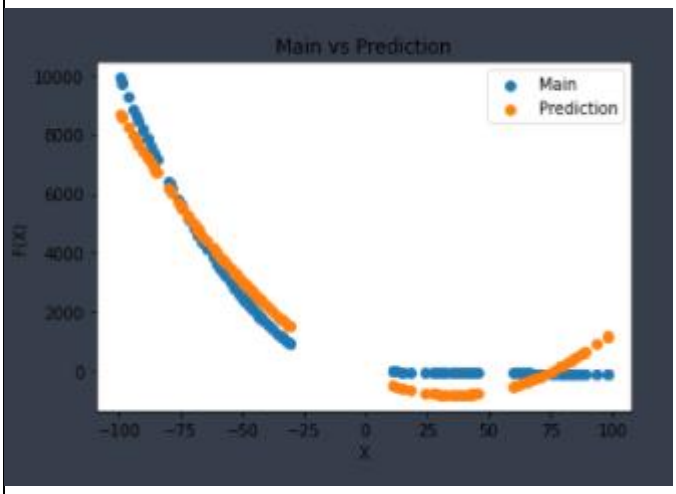


$$22:\ F(x) = \begin{cases} x^2, & -100 < x < -30 \\ -x - 3, & 10 < x < 100 \end{cases}$$

```
func = Pow(Param(0), Const(2))
dataset = Dataset(80, func, 1, (-100, -30))
func = Sub(Sub(Const(0), Param(0)), Const(3))
dataset.add_dataset(Dataset(50, func, 1, (10, 100)))
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(10, 0.1)
# trainer.evolute(20, 0.01)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50), dataset)
```

Best sample: x * ((x * 0.83) / 1.62) + (x + 3.32) * -37.28
Loss: 324651.58590126317
Number of generations: 10
Number of examinations: 50
Execution time: 5.611530065536499





23: $F(x) = random\ function$

```
func = None
dataset = Dataset(80, func, 1, (-100, 100))
population = [ComplexTree((-2, 2)) for _ in range(500)]
trainer = EvolutionaryTrainer(dataset, population, ComplexTree.breed, ComplexTree.mutate)
trainer.evolute(20, 0.1)
# trainer.evolute(20, 0.01)
trainer.show_trend()
compare(func, trainer.best_sample, (-50, 50), dataset)
```

Best sample: cos((x - -0.02) - 0.47) + cos(x * (-3.92 - x)) + cos(x - 0.47) + cos(x * (-0.43 - sin(x)))
Loss: 30.732209960584886
Number of generations: 20
Number of examinations: 1000.0
Execution time: 8.258622407913208