بسمه تعالی

تقریب تابع با کمک برنامه نویسی ژنتیک (GP)

تهیه کننده: هلیا وفایی

مدرس: دکتر عبدی

پروژه ی اول

نیم سال اول سال

توضیح فایل main.py :

در ابتدا آرایه ای تحت عنوان FUNCTIONS شامل عملگر های جمع، تفریق، ضرب، تقسیم تعریف می کنیم.

تابعی به نام func تعریف می کنیم که ضابطه ی مورد نظر را در آن تعریف می کنیم.

آرایه ای به نام X و Y تعریف می کنیم به طوری که آرایه ی X شامل اعداد 0 تا 10 به فاصله ی 0.01 و آرایه ی Y شامل جواب تابع func به ازای آرایه ی X می باشد.

در ادامه به صورت تصادفی عددی را به عنوان تعداد ردیف های موجود (عمق) در گراف انتخاب می کنیم و در ادامه با صدا زدن تابع RandomGraphs در فایل make_random_graphs.py شروع به ساختن 200 گراف (جمعیت اولیه) می کنیم.

سپس به ازای هر گراف و مقادیر مختلفی از اعداد تحت عنوان x، خروجی گراف به وسیله ی تابع eval در فایل evaulte.py محاسبه می شود که خروجی این اعداد در آرایه ای به نام evals ذخیره می شوند.

بعد از پر کردن آرایه ی evals باید میزان شایستگی هر گراف بررسی شود که بدین منظور از آرایه ی fit در فایل fitness استفاده می کنیم.

در قدم بعدی باید جمعیت جدید را محاسبه کنیم که برای این کار می توان به تعداد 200 (تعداد کل گراف ها) بار مادر، پدر و در نتیجه فرزند تعریف کرد. برای تعریف مادر و پدر از تابع select در فایل selection و برای تعریف فرزند از تابع cross در فایل cross_over بهره می بریم.

باید توجه داشت که آرایه ی گره های فرزند را به یک گراف جدید تحت عنوان g_child اضافه کنیم. در ادامه مشابه کاری که برای جمعیت اولیه انجام دادیم به محاسبه ی fitness گراف g_child می پردازیم. بعد از این کار گراف g_child را به یک آرایه تحت عنوان population اضافه می کنیم.

در مرحله ی بعد باید بهترین گراف از آرایه ی population را انتخاب کرد که برای این کار می توان از تابع best در فایل return_best استفاده کرد.

سپس گره های گراف مورد نظر را چاپ می کنیم.

در آخر باید نمودار های ضابطه ی اصلی و ضابطه ای که محاسبه کردیم را بکشیم.

توضیح فایل graph.py :

این فایل شامل 2 کلاس برای ساخت گره (Node) و گراف (Graph) می باشد. به طوری که Constructor کلاس گره شامل 2 پارمتر data و children می شود.
Constructor کلاس گراف شامل 2 پارمتر fitness و nodes می باشد. همچنین شامل 2 تابع برای اضافه کردن گره ی ریشه (add_root) و اضافه کردن بچه های یک گره (add_child) می باشد.

توضیح فایل make_random_graphs.py :

ابتدا شروع به ساختن 200 گراف می کنیم. برای ساخت گراف به صورت تصادفی یکی از عناصر موجود در آرایه ی FUNCTIONS را انتخاب می کنیم. سپس همان عملگر را به عنوان ریشه ی اولین گراف خود قرار می دهیم و تعداد کل گره های موجود در گراف را تحت عنوان num_of_nodes حساب می کنیم.

در مرحله ی بعد تابع make را صدا می زنیم که کار این تابع ساخت گراف می باشد. بعد از ساخت هر گراف آن را در یک آرایه ذخیره و آن را return می کنیم.

همان طور که اشاره شد کار تابع make ساخت گراف می باشد بدین حالت که در ابتدا 2 گره از آرایه ی FUNCTIONS به صورت تصادفی انتخاب و به گراف مورد نظر اضافه می شوند و این کار تا زمانی که تعداد گره های گراف به اندازه ی num_of_nodes برسد ادامه پیدا می کند. در مرحله ی آخر هم به تعداد مورد نیاز x به گراف اضافه خواهد شد.

توضیح فایل evaluate.py :

هدف این فایل محاسبه ی خروجی یک گراف به ازای اعداد مختلف می باشد. در تابع موجود در این فایل به عمیق ترین گره های موجود در گراف رفته و از پایین شروع به محاسبه ی ضابطه می کنیم.

توضیح فایل fitness.py :

هدف این فایل محاسبه ی میزان شایستگی هر گراف به کمک فرمول انحراف از معیار می باشد که این محاسبه را به وسیله ی آرایه ی evals (مقدار خروجی هر گراف به ازای مقادیر مختلف x) و Y (مقدار صحیح خروجی به ازای مقادیر مختلف x) انجام می دهد.

توضیح فایل selection.py :

هدف این فایل ساخت یک والد برای تولید جمعیت جدید می باشد. ابتدا به طور تصادفی یک آرایه به اندازه ی 100 (نصف اندازه ی گراف ها) انتخاب می کنیم. سپس اولین عضو از آرایه را به عنوان بهترین عضو برمی گزینیم. در آخر به تعداد 100 بار بررسی می کنیم تا بهترین گراف از نظر fitness را پیدا و برگردانیم.

توضیح فایل cross_over.py :

هدف این فایل ساخت آرایه ی فرزند می باشد. به طوری که یک انتها تحت عنوان my_len_m و my_len_f برای آرایه ی گره های مادر و پدر در نظر می گیریم. باید توجه داشت که این انتها طول آرایه ی گره های مادر و پدر بدون احتساب گره های شامل x می باشد. در قدم بعد برای تولید فرزند یک شروع و پایان از طول والدین تحت عنوان start_m، stsart_f، end_m، end_f در نظر می گیریم. حال برای تولید فرزند بدین صورت عمل می کنیم که آرایه ی مادر از ابتدا تا start_m، آرایه ی پدر از start_f تا end_f و در آخر آرایه ی مادر از end_m تا my_len_m را کنار یکدیگر قرار می دهیم و این کار را تا زمانی ادامه می دهیم که مطمئن شویم آرایه ی فرزند متوان می باشد. از آنجایی که آرایه ی مادر و پدر را بدون نظر گرفتن گره های شامل x انجام دادیم، در مرحله آخر باید به تعداد طول آرایه ی فرزند، x به انتهای آرایه، x اضافه کنیم. در آخر آرایه ی فرزند را برمی گردانیم.

توضیح فایل return_best.py :

هدف این فایل پیدا کردن گرافی در آرایه ی population می باشد که دارای بهترین fitness است. برای این کار اولین عضو از آرایه را به عنوان بهترین گراف انتخاب می کنیم و در ادامه به تعداد طول آرایه بررسی می کنیم تا بهترین گراف از نظر fitness را پیدا و برگردانیم.

چالش های مواجه شده در زمان حل پروژه و روش حل آن ها :

یکی از بزرگترین چالش هایی که در زمان حل پروژه با آن رو به رو شدم چگونگی ساخت گراف فرزند بود که با امتحان کردن روش های مختلف متوجه شدم می توان در ابتدا از آرایه ی گره های مادر و پدر، آرایه ای جدید ساخت و در آخر آن آرایه را به گراف تبدیل کرد.

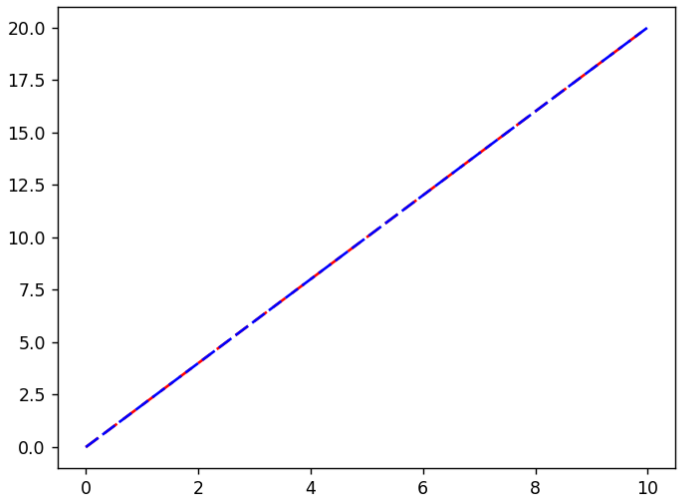
نمونه ای از آزمایش های انجام شده :

توجه :

نمودار آبی رنگ مربوط به ضابطه ی اصلی و نمودار قرمز رنگ مربوط به ضابه ی به دست آمده می باشد

مثال 1 : ضابطه ی اصلی : x+x

ضابطه ی به دست آمده : x+x



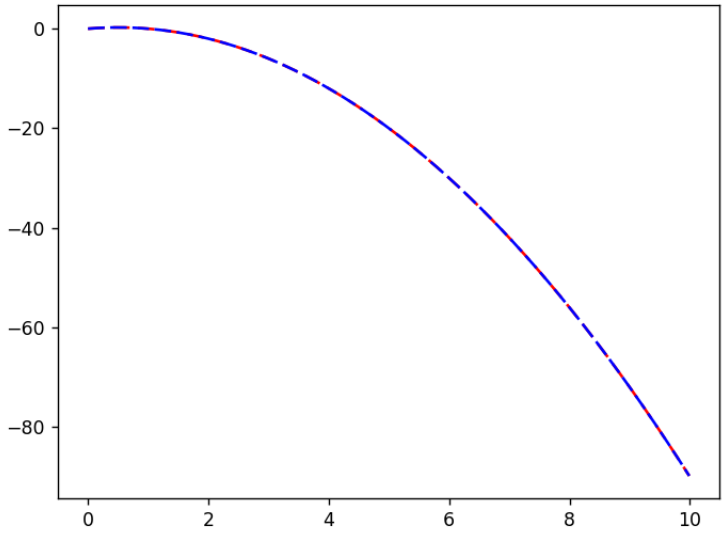
شکل 1-1 : نمودار های کشیده شده از ضابطه ها

```
+
x
x
fitness is 0.0
amount of calculating fitness is 400
amount of descendant is 2
Runtime of the program is 2.8622219562530518
```

شکل 1-2 : خروجی چاپ شده

مثال 2 : ضابطه ی اصلی : x-x*x

ضابطه ی به دست آمده : x-x*x/x



شکل 2-1 : نمودار های کشیده شده از ضابطه ها

```
*
-
/
x
x
x
x
fitness is 1.1152408404734431e-29
amount of calculating fitness is 400
amount of descendant is 2
Runtime of the program is 186.6873426437378
```

شکل 2-2 : خروجی چاپ شده

جمع بندی و نتایج علمی درباره ی برنامه نویسی ژنتیک (GP) :

برنامه نویسی ژنتیکی یک شاخه از الگوریتم ژنتیک می باشد. تفاوت اصلی آن با الگوریتم ژنتیک در نحوهٔ پردازش و نمایش جواب ها می باشد. الگوریتم ژنتیک یک رشته از اعداد را به عنوان جواب نمایش داده می دهد. این الگوریتم از 4 گام برای حل مسائل استفاده می کند: یک جمعیت اولیه از ترکیب های تصادفی عملگر ها و پایانه های مسئله ایجاد می کند.

هر نمونه حل (برنامه) که در جمعیت می باشد را اجرا می کند و براساس اینکه چقدر با هدف منظور سازگاری دارد به آن یک میزان از سودمندی می دهیم. یک جمعیت جدید از برنامه های کامپیوتری (نمونه حل ها) ایجاد می کنیم. بهترین برنامه های موجود را کپی می کنیم. برنامه های کامپیوتری جدید را توسط عمل جهش ایجاد می کنیم. برنامه های کامپیوتری جدید را توسط عمل ادغام بهترین نمونه برنامهٔ پیدا شده را به عنوان خروجی انتخاب می کنیم. یکی از مسائلی که در علوم کامپیوتر مطرح است، این است که چگونه به کامپیوتر این قابلیت را بدهیم که تنها به آن بگوییم که یک کار خاص را انجام دهد، و کامپیوتر آن کار را بدون اینکه به آن اعلام کنیم که چگونه آن را انجام بده، انجام دهد. برنامه نویسی ژنتیک این مشکل را حل کرده است، به این صورت که یک برنامه کامپیوتری برای یک مسائله سطح بالا تولید می‌کند. برنامه نویسی ژنتیک در واقع به هدف ما، یعنی برنامه نویسی اتوماتیک نیز شناخته می‌شود. پرورش این جمعیت با استفاده از نسخه الهام گرفته از عمل های انتخاب و تولید مثل بیولوژیکی صورت می‌گیرد. همانطور که در قسمت الگوریتم‌های ژنتیک نیز بیان کردیم، این اعمال بازترکیبی و جهش می‌باشد، که با ترکیب با اعمال انتخاب والدین و انتخاب بازماندگان به این مهم دست پیدا می‌کنند.