گزارش

اسم درس

1401/8/17

- نام و نام خانوادگی :
- نام استاد :
- گروه :

گزارش کار

توضیح اجرای برنامه:

برنامه به دو صورت نوشته شده است:

1. تابع ریاضی دلخواه را در قسمت f method نوشته و سپس مقادیر نقاطی در محدوده ی قابل مشخص به دست آمده و سپس با استفاده از این نقاط و الگوریتم ، تابعی شبیه به تابع اصلی توسط کد حدس زده میشود.

2. نقاط تابع مورد نظر به برنامه داده میشود ، سپس بر اساس آن نقاط و محدوده ی آن ، با استفاده از الگوریتم ، تابعی شبیه به تابع اصلی توسط کد حدس زده میشود.

کتابخانه های مورد نیاز:

1. matplotlib : ترسیم توابع به صورت گرافیکی
2. tqdm : نشان دادن progressbar
3. numpy : توابع ریاضی مورد نیاز

- تمامی کتابخانه از طریق pip قابل نصب هستند.

متغییر های مهم:

1. Terminal_set :

یک لیست از متغییر های استفاده شده ، برای سادگی کار یک بعدی در نظر گرفته میشود.

2. Max_depth :

میزانی که در یک درخت عمیق میشود. هر چه عدد بیشتر باشد زمان بیشتری برای اجرای برنامه نیاز است. عددی بین 20 تا 40 بهترین حالت ها بود. کمتر از 20 تابع پیش بینی شده از واقعیت دور میشود ، بیشتر از 40 هم زمان بیشتر برده و گاها نتایج ناخواسته به وجود میاورد.

3. Population_num :

تعداد جمعیت را تعیین میکند. مقادیر بین 6 هزار تا 10 هزار بهترین حالت ها را به وجود آوردند. هر چی جمعیت کمتر باشد نتایج تفاوت بیشتری پیدا میکنند. ولی برعکس این موضوع صادق نیست ، هر چی جمعیت بیشتر شود لزوما به معنای بهتر شدن و نزدیک شدن تابع پیش بینی شده نیست. از جایی به بعد جمعیت بیشتر فقط سرعت اجرای برنامه را کند میکند و روی تابع خروجی تاثیری ندارد.

4. Selected_chromosomes :

تعداد کروموزوم های انتخابی برای جمعیت است. تعداد زیاد آن سرعت برنامه را کم میکند ولی تغییر مقدار آن تفاوت بسیاری در خروجی ها میدهد.

5. Epoch_feedback :

هر چی تعداد آن کمتر باشد ، سرعت برنامه نیز کمتر میشود. این مقدار برای بررسی جمعیت با نتایج به دست آمده استفاده میشود. اگر روی 1 باشد یعنی با تمام جمعیت نیاز است که بررسی شود. اگر روی 10 باشد ، به معنای بررسی یک دهم از جمعیت است. کمتر بودن مقدار این متغییر باعث نتایج بهتر میشود و تفاوت در بعضی توابع بسیار فاحش است.

6. Functions :

توابع و عملگر ها را نگه داری میکند. ترتیب اعضای هر لیست در خروجی تابع تاثیر گذار نیست. با تغییر ترتیب index توابع یا عملگر ها نتایج متفاوتی میتوان به دست آورد اما این تفاوت از مقدار رندومی هست که به وجود آمده و به index آن مربوط نیست. هر چند در بسیار از توابع ، گاها مقدار هایی به صورت اتفاقی به وجود می آیند که نتیجه ی آن بسیار نزدیک به تابع اصلی است.

7. در صورت استفاده از use_method :

7.1. X_start_range و X_end_range : بازه ی x ها ، هر چه فاصله ی این دو بیشتر باشد نقاط بیشتری باید پردازش شود.

7.2. X_step : مقداری که هر بار به x اضافه میشود تا x بعدی را پیدا کند. هر چه مقدار این عدد کمتر باشد ، نقاط بیشتری پیدا شده و سرعت برنامه نیز افت پیدا میکند (دقت تابع به دست آمده بالا میرود.)

گزارش کار

جزییات پروژه:

تولید جمعیت اولیه ، را بر اساس تعداد واحدی که در نظر گرفته ایم انجام میدهیم. سپس برای الگوریتم ، به صورت رندوم از قسمتی از جمعیت پدر و مادر را به صورت جدا مشخص میکنیم. سپس ژنوم پدر را بین ژنوم های مادر قرار میدهیم تا فرزند را از آن ها بسازیم.

نتایج به دست آمده:

- سرعت اجرای برنامه به شدت به متغییر های بالا بستگی دارد.
- سرعت برنامه همچنین به تعداد نقاط هم بستگی دارد.
- جمعیت ، اعمال الگوریتم ، انتخاب تست و ... به صورت تصادفی اتفاق میافتد. ممکن در تستی به صورت کاملا اتفاقی یک درخت بسیار شبیه پیدا شود که در دیگر تست ها اتفاق نیفتد ؛ هر چند که تمامی متغییر ها کاملا یکسان باشند.
- بعضی توابع کاملا درست پیدا میشود ولی مقدار عددی آن ها درست نیست زیرا ضریب عددی در آن ها اعمال نمیشود. به عنوان مثال:

2x^2 + 5x >> x^2 + x

گزارش کار

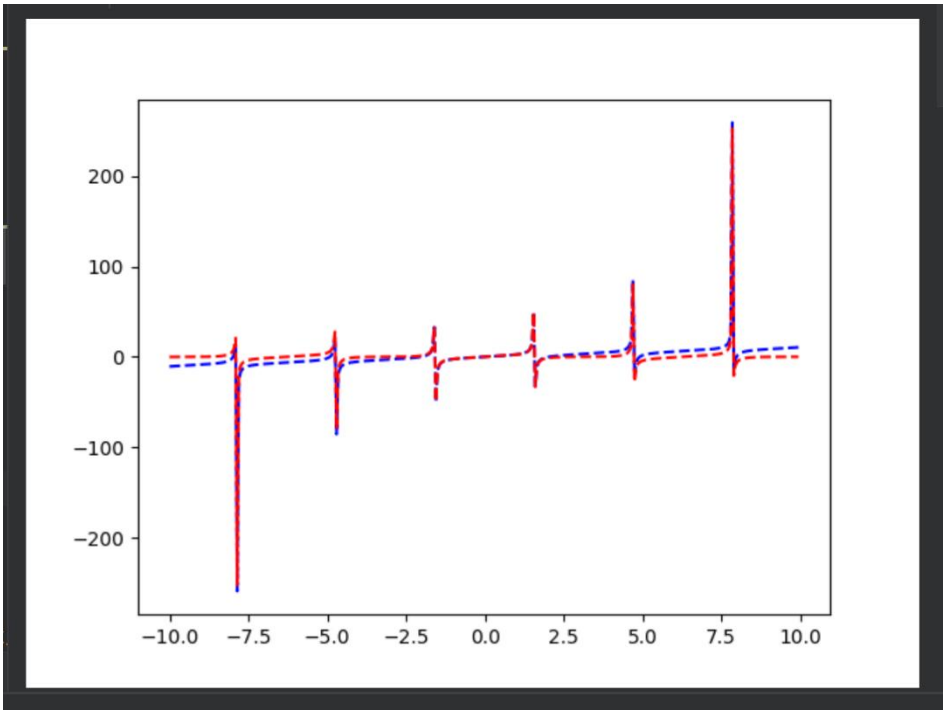
تبصره 1:

برنامه به طریقی نوشته شده است که میتوان توابع یا عملگر های اضافی را اضافه یا حذف کرد. در صورتی که نیاز به تست آن باشد ، فقط با کامنت کردن یک خط قسمت های اضافی و خارج از متن پروژه را میتوان از برنامه خارج کرد.

مثال:

y = tan(x) + x

خروجی بدون داشتن tan در توابع:



درخت:

```
['+', 'sin', 'x0', '/', 'sin', 'x0', 'cos', 'x0']
```

خروجی:

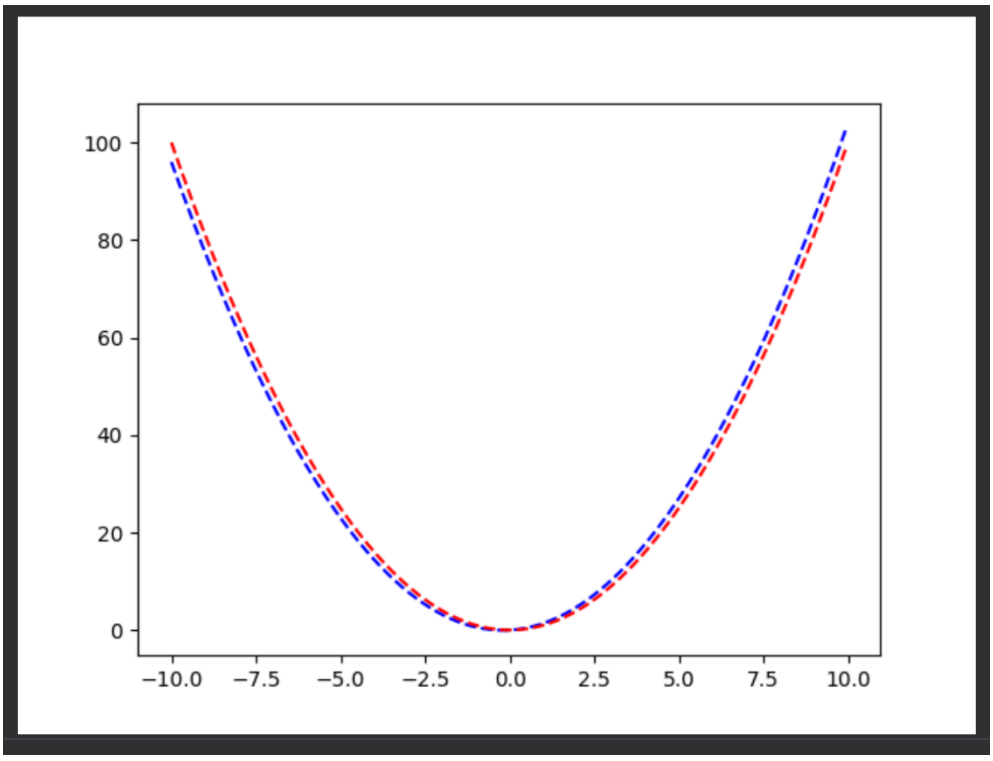
y = sin(x) + sin(x) / cos(x)

گزارش کار

مثال:

y = x ^ 2 + 2 * x / 5

خروجی بدون داشتن عملگر ضرب و تقسیم:



درخت:

```
['ln', 'abs', '^', 'e', 'x0', 'x0']
```

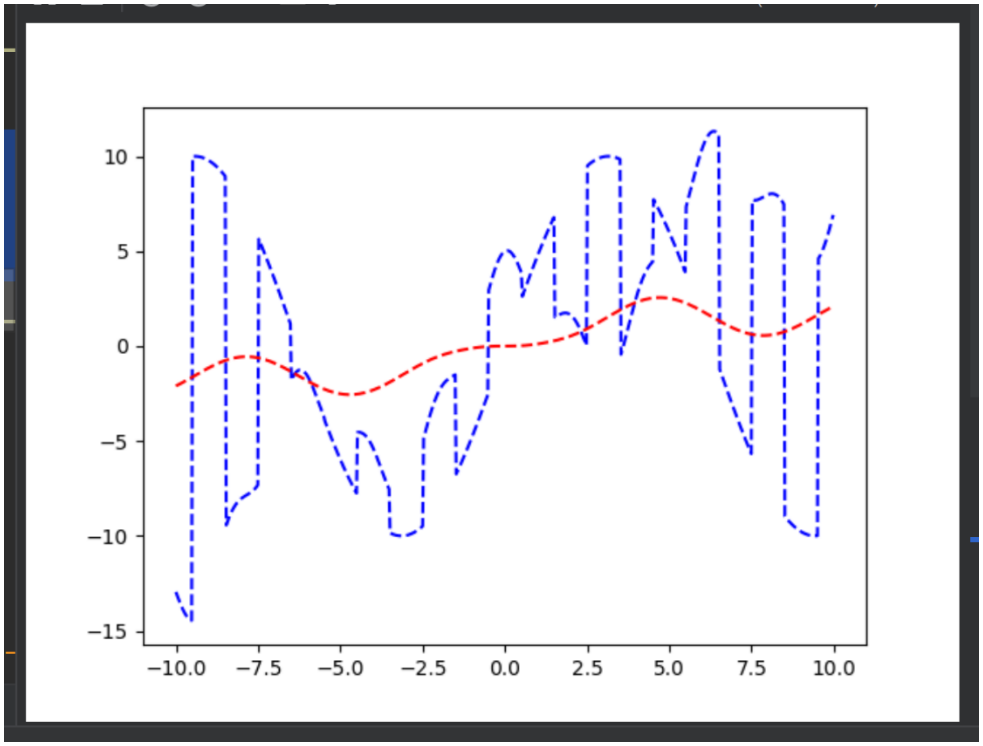
خروجی:

y = ln(|e ^ x|)x

تبصره 2: مثال:

```
if int(round(x)) % 2 == 0:
    y = x + 5 * cos(x) ^ 3
else:
    y = 10 * sin(x / 2)
```

خروجی تابع خط خطی:



درخت:

```
['-', 'tg', 'tanh', 'x0', 'sin', 'x0']
```

خروجی:

y = tan(tanh(x)) – sin(x)

تبصره 3:

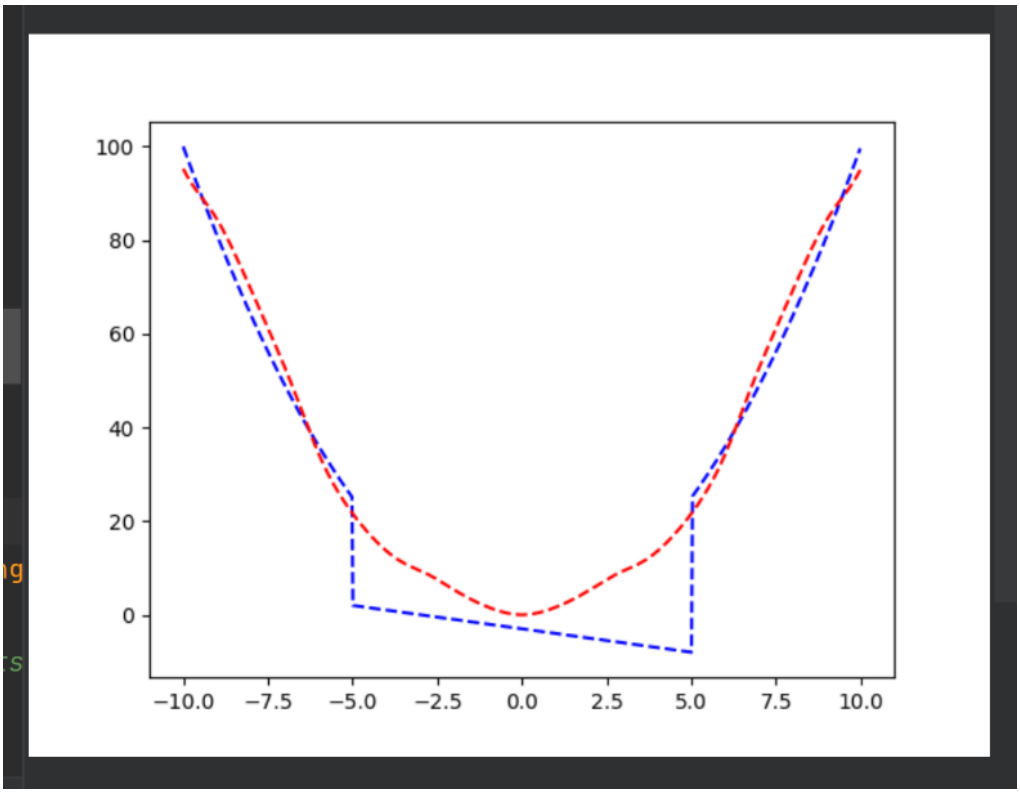
```
if x < -5:
    return x ** 2

elif -5 ⩽ x ⩽ 5:
    return -x - 3

else:
    return x ** 2
```

مثال:

خروجی تابع:



درخت:

```
['*', '+', 'tanh', 'sin', 'sin', 'x0', 'x0', 'x0']
```

خروجی:

y = (tan(sin(sin(x))) + x) * x

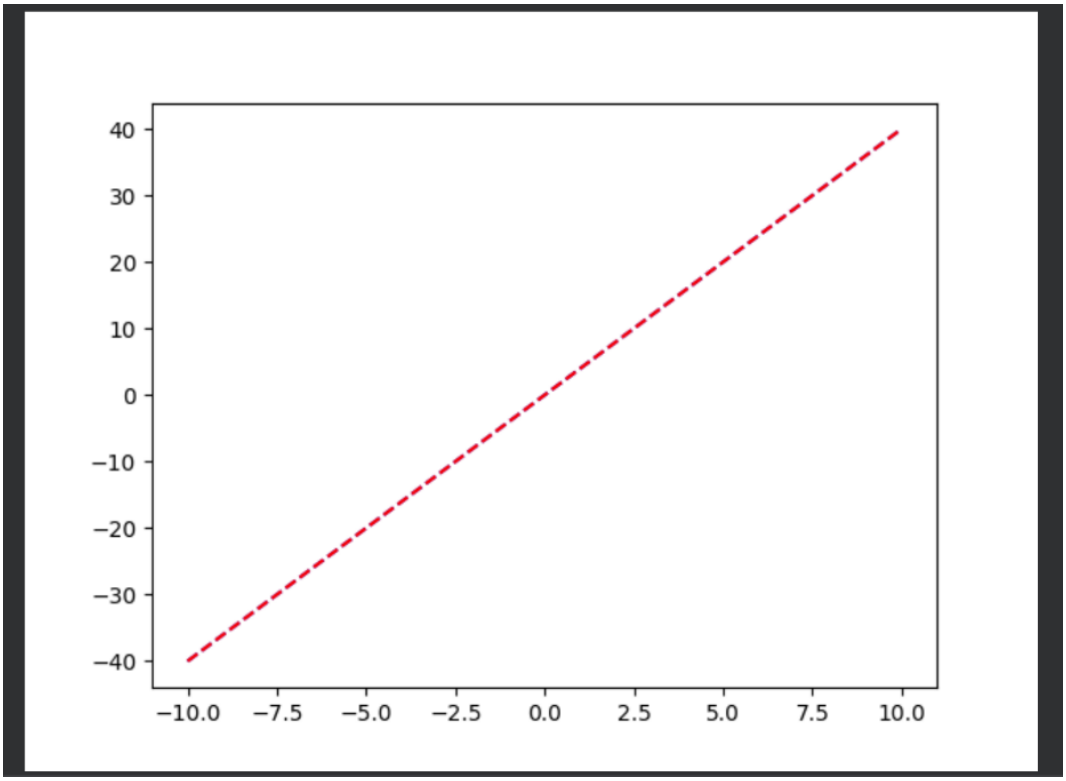
گزارش کار

تبصره 4: **تست انواع تابع**

1. تابع خطی

خروجی تابع خطی:

Y = X * 4



درخت:

```
['+', 'x0', '+', 'x0', '+', 'x0', 'x0']
```

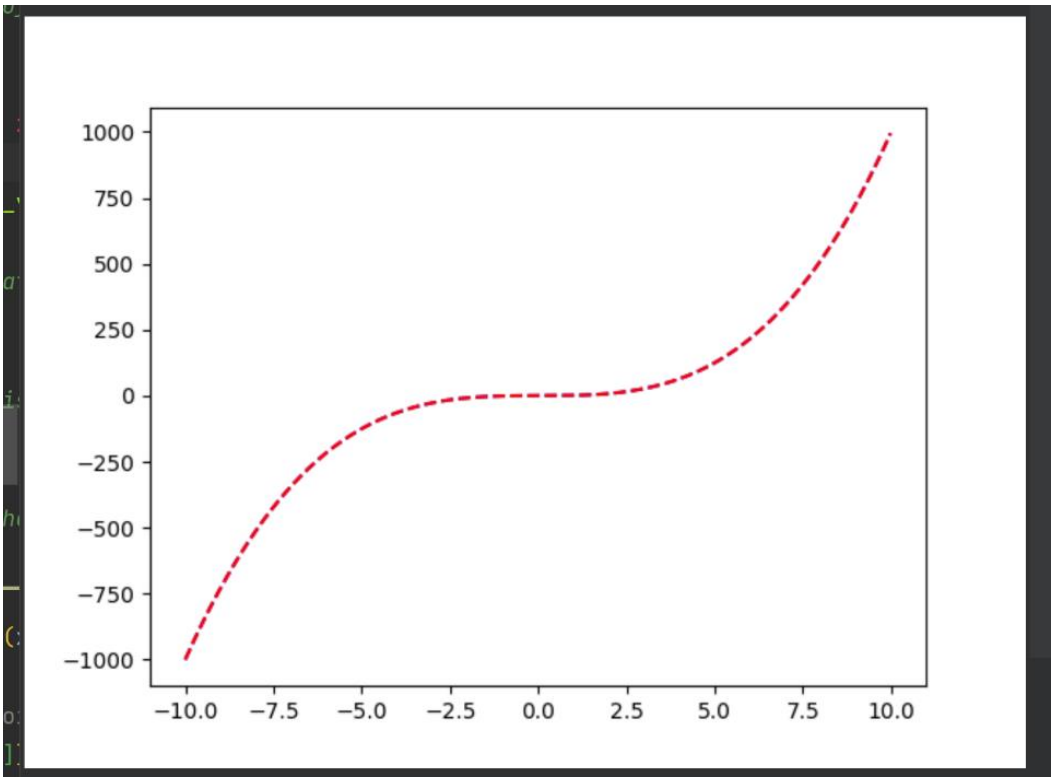
خروجی:

y = (x) + (x + (x+x))

2. تابع مربعی: **تابع استفاده شده در تبصره 1**
3. تابع مکعبی

خروجی تابع مکعبی:

Y = X ^ 3



درخت:

```
['*', 'x0', '*', 'x0', 'x0']
```

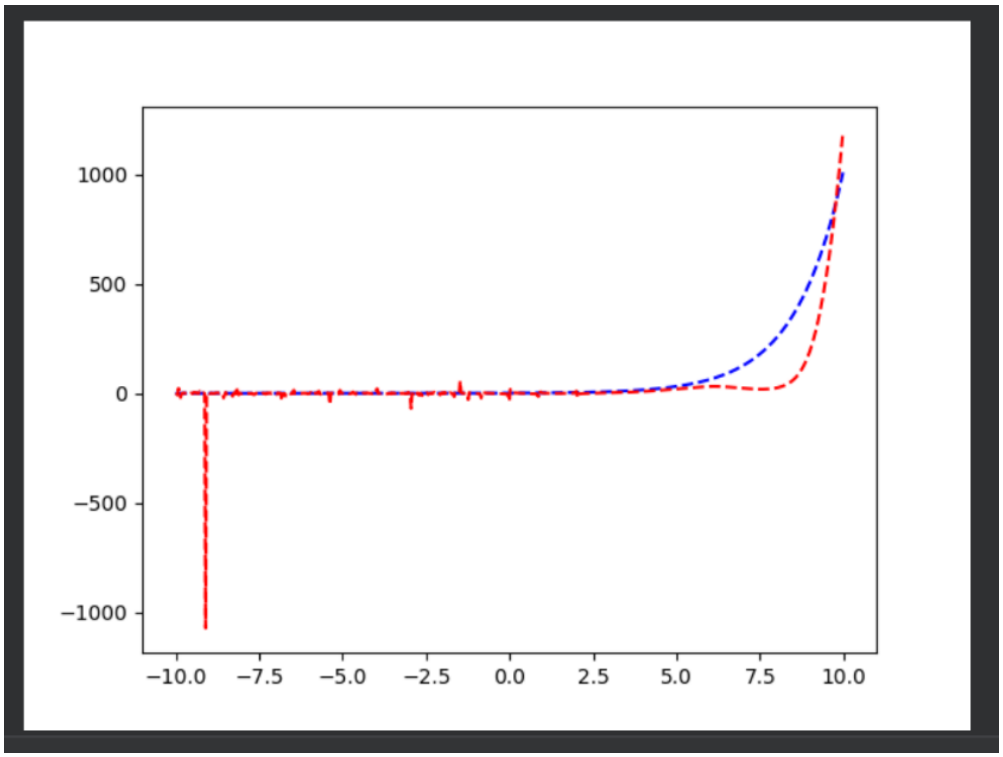
خروجی:

y = (x) * (x * x)

4. تابع نمایی:

خروجی تابع نمایی:

Y = 2 ^ x



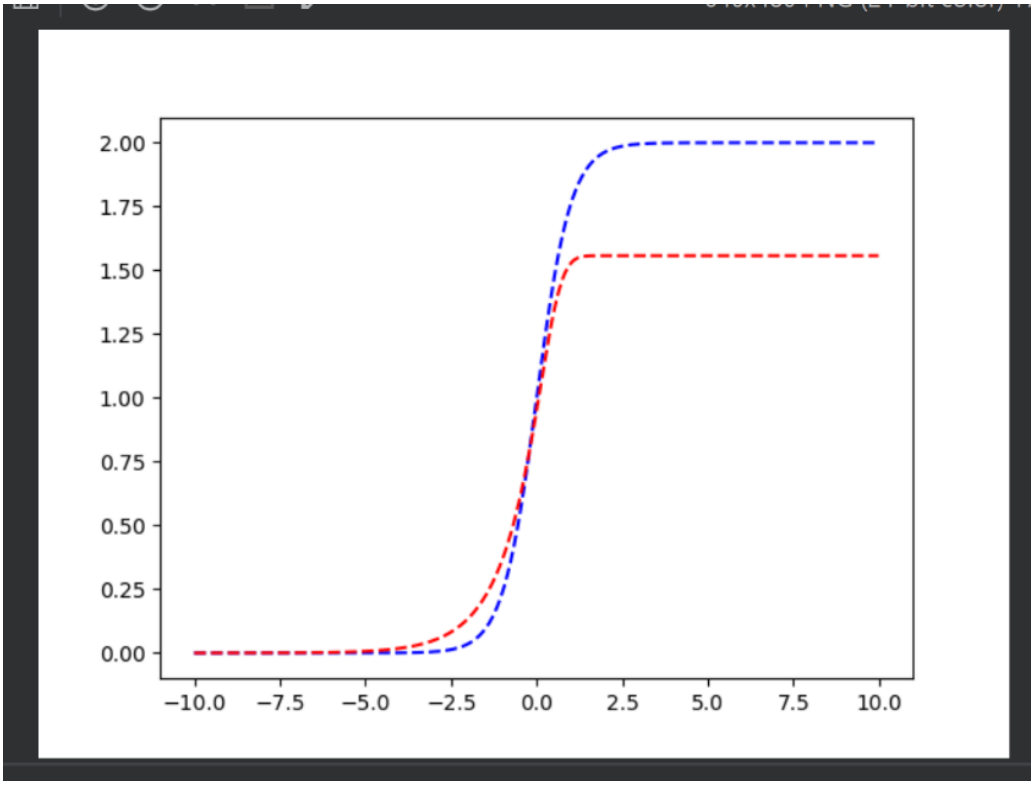
درخت:

```
['ctg', '*', 'x0', 'e', '-', 'e', 'sin', 'x0', 'x0']
```

5. تابع مثلثاتی: تابع استفاده شده در تبصره 2
6. تابع هذلولی:

خروجی تابع هذلولی:

Y = tanh(x) + 1



درخت:

```
['tg', 'tanh', 'e', 'x0']
```

خروجی:

y = tan(tanh(e)x)