درس رایانش تکاملی

استاد درس: جناب آقای دکتر ملک

بهمن ۱۴۰۰

الگوریتم بهینه‌سازی گوزن یال‌دار

نغمه فرزی

شماره دانشجویی: ۹۹۴۴۳۱۷۴

فهرست

۱. مقدمه

در این پروژه مطلوب پیاده سازی مقاله الگوریتم بهینه سازی بر اساس رفتار های گله‌ای گوزن‌های یالدار بوده. این الگوریتم بر اساس ۵ خصیصه رفتاری این حیوانات پیاده شده است. خصوصیات رفتاری این حیوانات شامل موارد زیر است:

- حرکت محلی :بدلیل محدودیت دید جستجو برای غذا فقط بصورت محلی
- غریزه ی گله ای :به صورت تصادفی به یک گوزن در گله که در مراتع پر علف تر قرار دارد نزدیک میشود
- حافظه ی جمعی :گله زمین های حاصلخیز را از مهاجرتهای قبلی به یاد دارد
- فشار جمعیت :حرکت به سوی زمین های حاصلخیز با در نظر گرفتن تمایل به عدم تداخلهای ارضی متعادل میشود .
- اجتناب از قحطی زدگی :گوزن از مکانهای خشک و بی حاصل دوری میکند

۲.پیاده سازی

در این مرحله پس از import کردن کتابخانه‌های مورد نظر به پیاده سازی بدنه‌ی الگوریتم می‌پردازیم. بخش‌های متفاوت الگوریتم اصلی بر اساس هر یک از خصوصیات رفتاری گفته شده شکل گرفته است. این الگوریتم برای هر یک از عضو های جمعیت اولیه و به تعداد تکرار مشخص شده ادامه پیدا می‌کند. بخش های متفاوت این الگوریتم در زیر توضیح داده شده اند.

متغیر های استفاده شده در بخش‌های این الگوریتم به شرح زیر است:

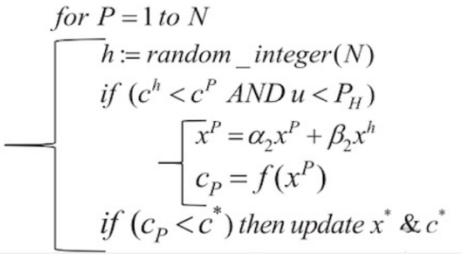
N	تعداد تکرارهای الگوریتم
D	بعد فضای جستجو
f	تابع هدف
N	تعداد جمعیت اولیه
X^p	موقعیت pامین عضو جمعیت
η	نرخ یادگیری
U	متغیر تصادفی (standard uniform)
$\alpha 1 , \beta 1$	پارامترهای کنترلی حرکت محلی
$\alpha 2 , \beta 2$	پارامترهای حرکت سراسری
δ_w	آستانه‌ی فاصله تا کمترین شایستگی
δ_c	آستانه‌ی فاصله برای جلوگیری از تداخل اعضا در مناطقی که بیشترین شایستگی را دارند
n_s	تعداد قدم‌های کوچک تصادفی
n_e	تعداد قدم‌های حول بهترین جواب
P_H	احتمال مربوط به حرکت یک عضو

در بخش اول الگوریتم حرکت محلی بر اساس شبه کد زیر پیاده شده است که در آن n_s مقدار ثابتی است برای مشخص کردن تعداد قدم‌های کوچک تصادفی، u یک راستای تصادفی برای بردار یکه بوده متغیر η طول قدم است که برای هر عضو متفاوت بوده و در نهایت هر عضو از جمعیت طی حرکت محلی به بهترین مکان یافته شده حرکت میکند.

$$
\begin{aligned}
&\textit{for } P = 1 \textit{ to } N \\
&\quad \textit{for } s = 1 \textit{ to } n_s \\
&\qquad y^s = x^P + \eta u v \\
&\qquad c_s^{'} = f(y^s) \\
&\quad s^* = \arg\min_s c_s \\
&\quad x^P = \alpha_1 y^{s^*} + \beta_1 (x^P - y^{s^*}) \\
&\quad c_P = f(x^P) \\
&\quad \textit{if } (c_P < c^*) \textit{ then update } x^* \ \& \ c^*
\end{aligned}
$$

```
[16] def local_movement(xp_cp_dataframe,ns,argmin_star,c_p_star):
       for idx,row in xp_cp_dataframe.iterrows():
         xp=row['x_p']
         cp=row['c_p']
         df = pd.DataFrame(columns=['y_s','c_s'])
         for s in range(ns):
           y_s= xp+etha*make_rand_unit_vector(dim)
           c_s= fitness(y_s)
           df.loc[len(df)]=[y_s,c_s]
         argmin=df['c_s'].idxmin()
         s_star=df['y_s'].loc[argmin]
         xp=alpha1*s_star+betta1*(xp-s_star)
         xp_cp_dataframe.at[idx,'x_p']=xp
         cp=fitness(xp)
         xp_cp_dataframe.at[idx,'c_p']=cp
         if cp < c_p_star:
           argmin_star=idx
           c_p_star=cp
       return xp_cp_dataframe,argmin_star,c_p_star
```

در بخش دوم الگوریتم، غریزه‌ی گله‌ای طبق شبه کد زیر پیاده شده که در آن به ازای هر عضو از جمعیت یک عضو از جمعیت بصورت تصادفی انتخاب شده و در صورتیکه شایستگی بالاتری داشته باشد با احتمال P_H به آن نزدیک تر میشویم.




```
def herd_distinction(xp_cp_dataframe,argmin_star,c_p_star):
  for idx,row in xp_cp_dataframe.iterrows():
    xp=row['x_p']
    cp=row['c_p']
    h=np.random.randint(0,len(xp_cp_dataframe))
    xh=xp_cp_dataframe['x_p'].loc[h]
    ch=xp_cp_dataframe['c_p'].loc[h]
    u = np.random.uniform(0,1)
    if ch<cp and u<ph:
      xp=alpha2*xp+betta2*xh
      cp=fitness(xp)
      xp_cp_dataframe.at[idx,'x_p']=xp
      xp_cp_dataframe.at[idx,'c_p']=cp
    if cp < c_p_star:
      argmin_star=idx
      c_p_star=cp
  return xp_cp_dataframe,argmin_star,c_p_star
```

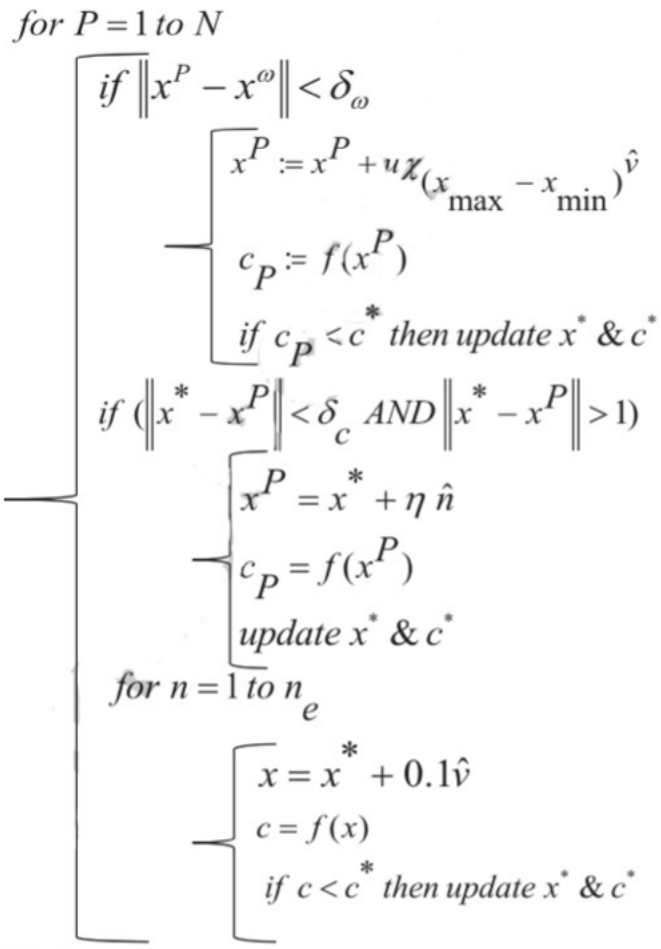
در بخش بعد به ازای هر عضو از جمعیت به ترتیب بخش‌هایی از الگوریتم که مربوط به اجتناب از قحطی زدگی بر اساس آستانه‌ی فاصله مشخص شده، فشار جمعیت بر اساس آستانه‌ی فاصله مربوط به جلوگیری از تداخل و همچنین مربوط به بخش حافظه‌ی جمعی است را انجام میدهیم.

```
def starvation_avoidance(xp_cp_dataframe,x_worst,dim,etha,distance_from_crowding_threshold,distance_from_worst_threshold,ne):
  argmin_star=xp_cp_dataframe['c_p'].idxmin()
  cp_star=xp_cp_dataframe['c_p'].loc[argmin_star]
  for idx,row in xp_cp_dataframe.iterrows():
    xp=row['x_p']
    cp=row['c_p']
    x_min=xp_cp_dataframe['x_p'].loc[argmin_star]
    argmax=xp_cp_dataframe['c_p'].idxmax()
    x_max=xp_cp_dataframe['x_p'].loc[argmax]
    dist= np.linalg.norm((xp-x_worst))
    # print( dist)
    if dist< distance_from_worst_threshold:
      xp=xp+np.random.uniform(0,1)*(x_max-x_min)*make_rand_unit_vector(dim)
      cp=fitness(xp)
      xp_cp_dataframe.at[idx,'x_p']=xp
      xp_cp_dataframe.at[idx,'c_p']=cp
      if cp<cp_star:
        argmin_star=idx
        cp_star=cp
    dist2 = np.linalg.norm((x_min-xp))
    if dist2<distance_from_crowding_threshold and dist2>1:
      xp=x_min+etha*np.ones_like(x_min) #n vec
      cp=fitness(xp)
      xp_cp_dataframe.at[idx,'x_p']=xp
      xp_cp_dataframe.at[idx,'c_p']=cp
      if cp<cp_star:
        argmin_star=idx
        cp_star=cp

    for _ in range(ne):
      x=x_min+.1*make_rand_unit_vector(dim)
      c=fitness(x)
      argmin_star=xp_cp_dataframe['c_p'].idxmin()
      cp_star=xp_cp_dataframe['c_p'].loc[argmin_star]
      if c<cp_star:
        cp_star=c
        xp_cp_dataframe.at[argmin_star,'x_p']=x
        xp_cp_dataframe.at[argmin_star,'c_p']=c

  return xp_cp_dataframe,argmin_star,cp_star
```

این قسمت نیز طبق شبه کد زیر پیاده شده است.



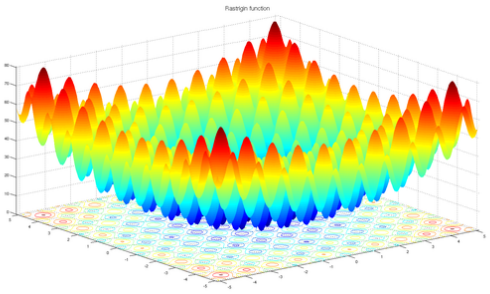
روند گفته شده را به ازای بیشینه تکرارهای مشخص شده انجام میدهیم. تابع اصلی پارامترهای ذکر شده را بعنوان ورودی گرفته و شایسته ترین عضوها در هر نسل را بعنوان خروجی بر میگرداند.

```
wildbeast_optimizaiton(xp_cp_dataframe,ns,dim,etha,distance_from_crowdi
ng_threshold,distance_from_worst_threshold,ne,ph,alpha1,betta1,alpha2,bett
a2,function='weierstrass')
```

۳. نتایج

برای مقایسه، عملکرد این الگوریتم را در کنار الگوریتم‌های Genetic Algorithm، ABC، SA، GSA و PSO روی benchmarkهای متفاوت بررسی کرده و نمودارها را برای ۲۰ جمعیت اولیه و در ۵۰۰ تکرار برای ۳۰ بعد نمایش داده‌ایم:

۱.Rastrigin:



- نمودار مربوط به مقاله:

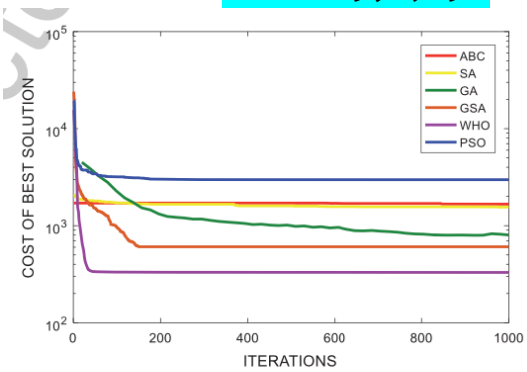
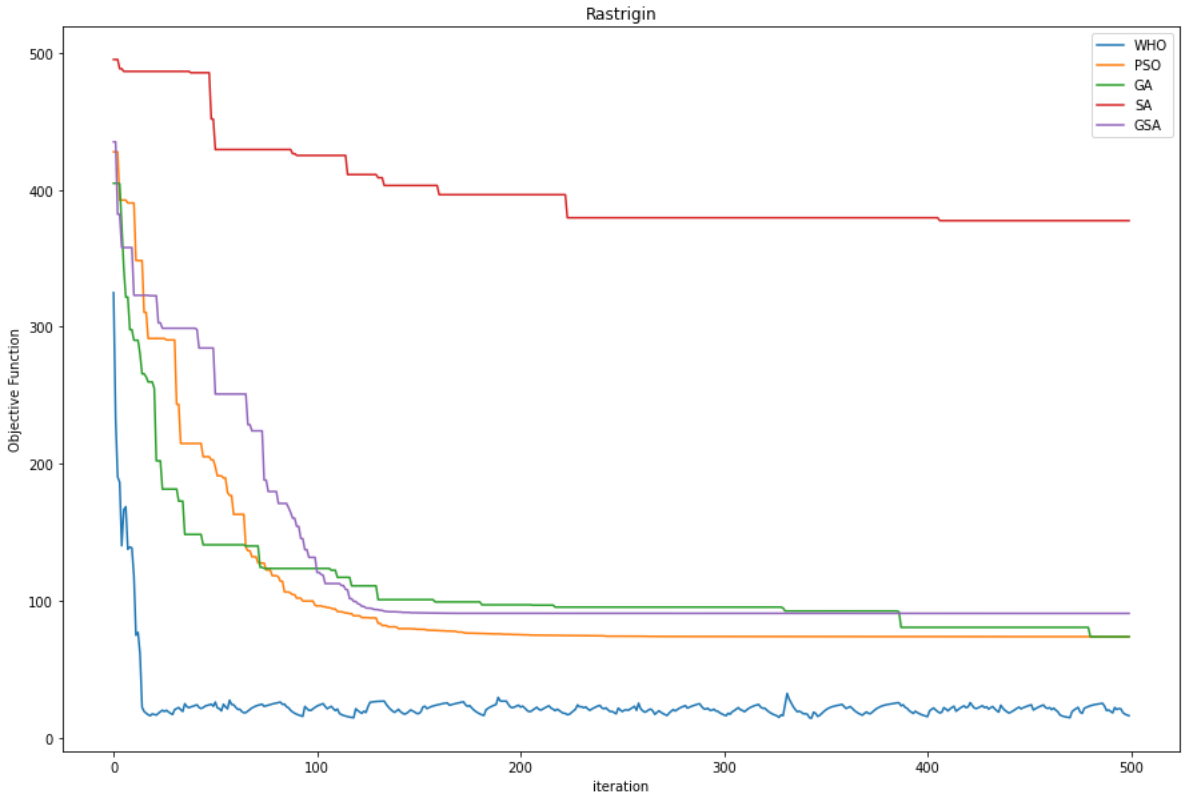


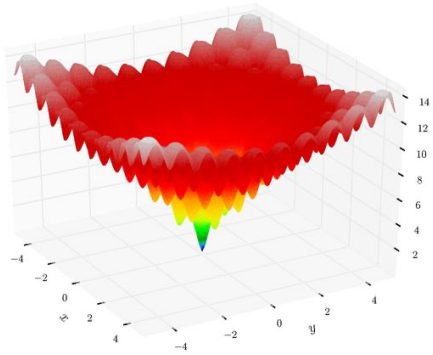
Fig. 9. Convergence characteristics of Rastrigin function.

- نمودار بدست آمده:



Ackley.۲:

- نمودار مربوط به مقاله:



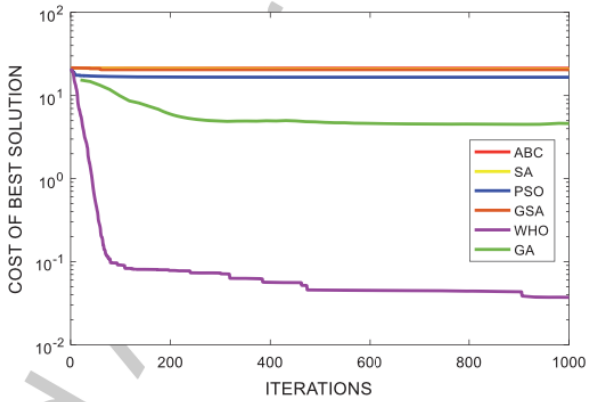
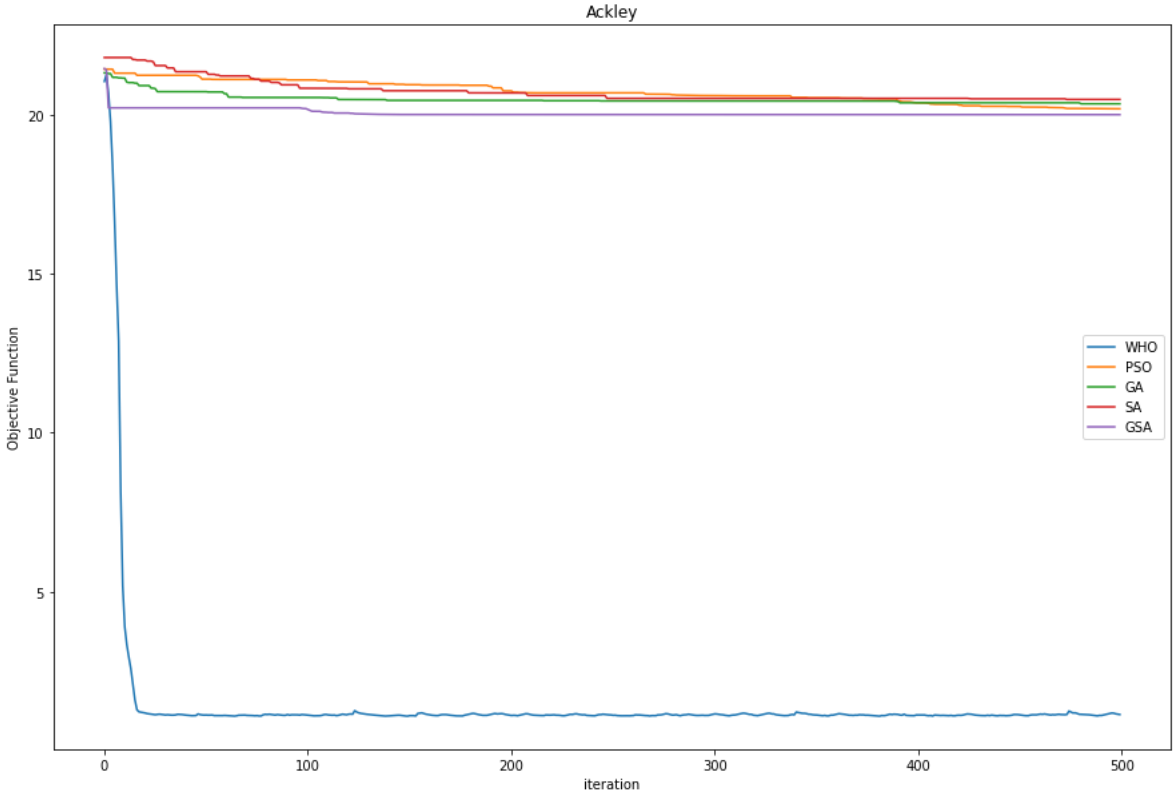


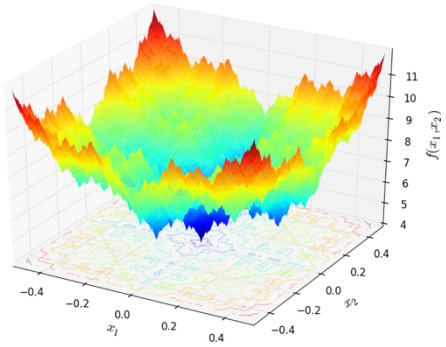
Fig. 8. Convergence characteristics of Ackley function.

- نمودار بدست آمده:

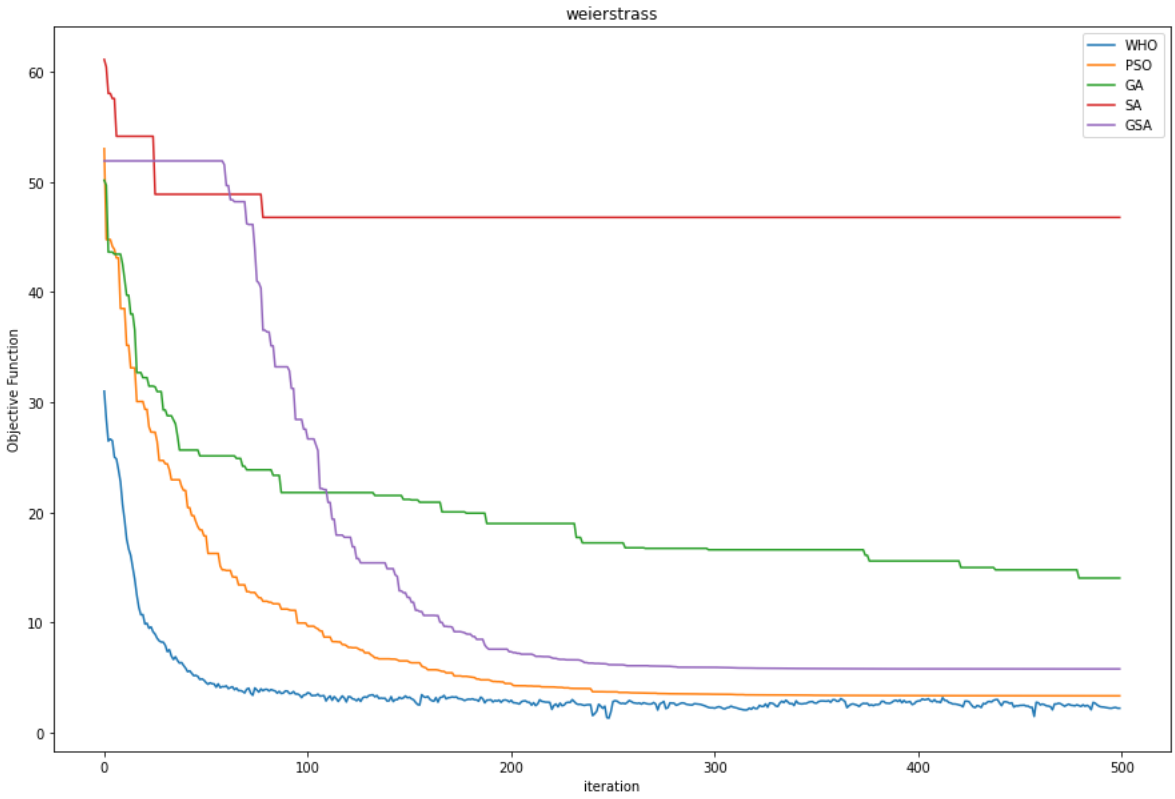


۳.weierstras :

- مقاله این تابع هدف را مورد بررسی قرار نداده بود.

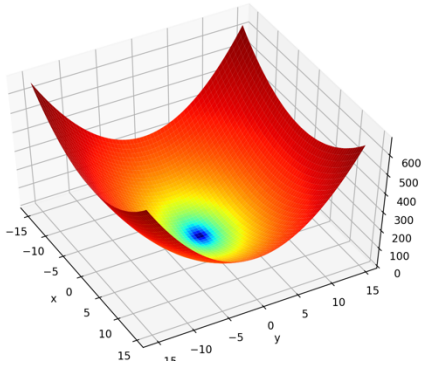


- نمودار بدست آمده:



sphere.۴:

- نمودار مربوط به مقاله:



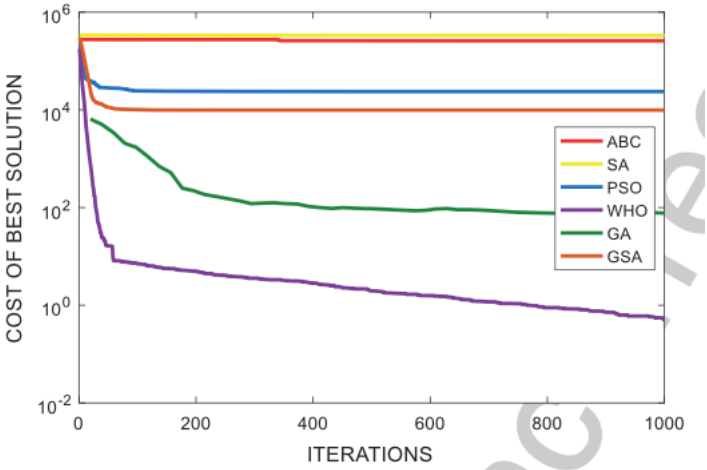
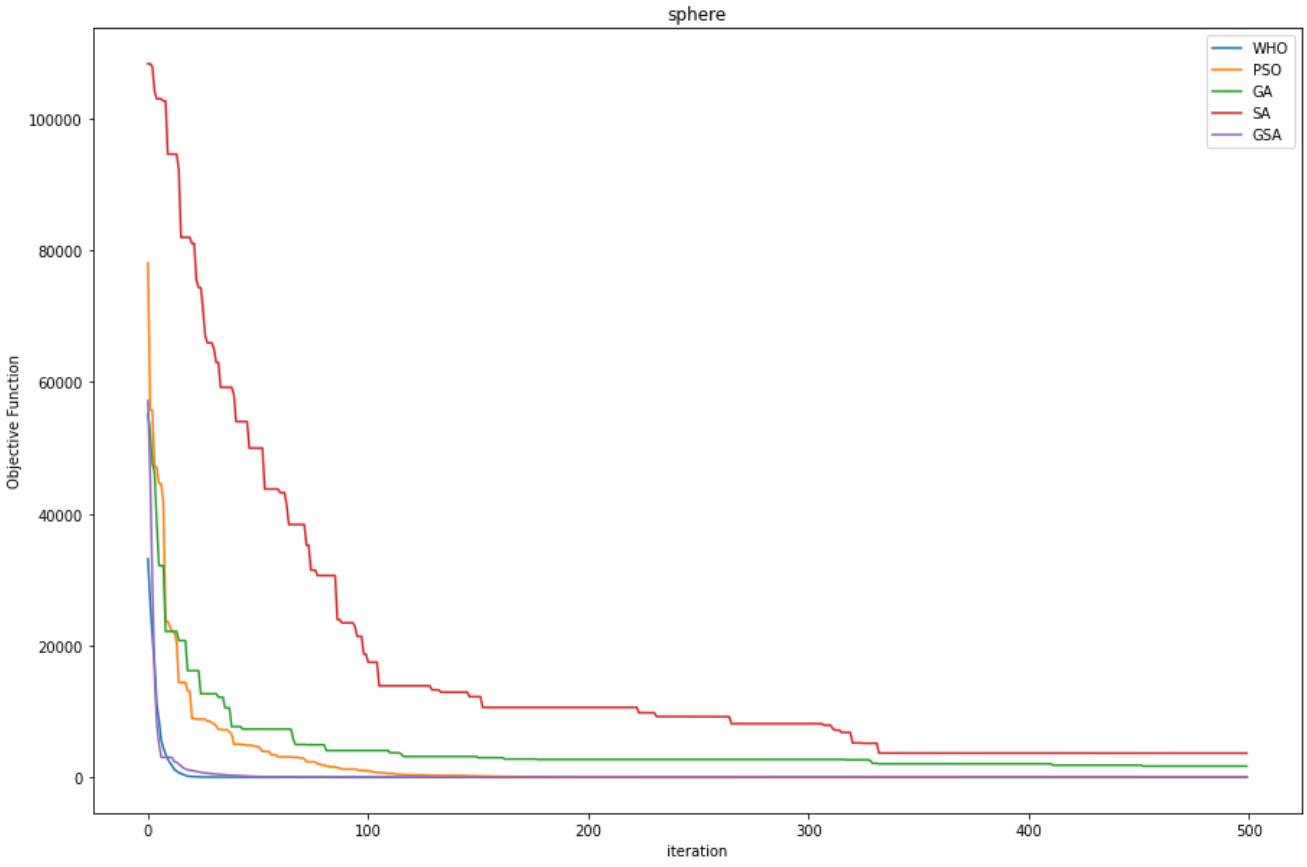


Fig. 6. Convergence characteristics of Noisy Sphere function.

- نمودار بدست آمده:



این الگوریتم در اجراهای متفاوت روی توابع ذکر شده همگرایی سریعتری نسبت به الگوریتم‌های دیگر دارد.

در نهایت بدلیل اینکه زمان اجرای الگوریتم بسیار بالا بود mean median وSTD را برای تابع هدف sphere در ۵ بعد، ۱۰۰ تکرار و ۵۰ اجرا بدست آورده و مقایسه کردیم.نتایج کمی با مقاله متفاوت است و این شاید به دلیل این باشد که ابعاد و تعداد تکرار ها از آزمایش های مقاله متفاوت است.
بعد از PSO الگوریتم WHO در ۱۰۰ تکرار بهتر عمل کرده است.

Algorithm	mean	median	STD
WHO	**0.02892827781798943**	**0.02853166738256003**	**0.0065312712058163**
PSO	**1.6825364242556e-09**	**1.6825364242556e-09**	**0.01**
SA	30.488334505807792	1.897808403180754	14.20745485358927
GSA	54.880756339339456	19.072088155832496	122.12059895267

لینک کد در گوگل کولب:

https://colab.research.google.com/drive/1STdQkGXgwvH4O3io8k6yjVZkiWIqArjK?usp=sharing

۴. راهنمای اجرای کد

برای اجرای کد هر cell به ترتیب باید اجرا شود. ابتدا cell مربوط به کتابخانه‌ها بوده سپس در بخش WHO الگوریتم اصلی نوشته شده است و پس از آن الگوریتم های بهینه سازی دیگر تعریف شده تا در مرحله‌ی مقایسه کارایی الگوریتم از آنها استفاده شود.
در مراحل بعدی توابع هدف متفاوتی با الگوریتمهای تعریف شده بهینه شده‌اند. در هر بخش متغیرهای مربوط به الگوریتم‌های تعریف شده مقدار دهی شده و پس از اجرای الگوریتم ها و یافتن شایسته ترین جواب در هر تکرار نمودار مربوطه رسم شده است. جواب های بدست آمده در هر تکرار در درایو ذخیره شده‌اند چون مدت اجرای الگوریتم بسیار طولانی بود.
در نهایت برای مقایسه‌یmean ، median و STD برای هر الگوریتم ۵۰ بار اجرا در نظر گرفته شده و موارد مذکور با کتابخانه‌ی statistics محاسبه شده است.