# Evolución diferencial y LSHADE aplicado a función de 10 variables

Ciro Fabian Bermudez Marquez
INAOE
Mexico, Puebla
cirofabian.bermudez@gmail.com

***Resumen*—En este trabajo se pone a prueba el algoritmo de evolución diferencial (ED) para un problema de minimización de una función de 10 variables implementado en python.**

***Index Terms*—ED, python, heuristica.**

## I. Introducción

La evolución diferencial usa números reales en su presentación de las variables, en general realiza un mejor trabajo que el algoritmo genético y se pude usar una condición automática de paro.

Las heurísticas se deben usar si el problema tiene tres o más variables y en este trabajo se estudiará la evolución diferencial (ED) aplicado a una función de 10 variables.

$$f_1(x) = 0.1 \sum_{i=1}^{10} (x_i - 2)(x_i - 5) + \sin(1.5\pi x_i) \qquad (1)$$

teniendo en cuenta las siguientes características para el algoritmo $C = 0.8$, $R = 0.6$, $\mu = 100$, $g = 50, 100, 150$, espacio de búsqueda $x_i = [0, 7]$ para todas las $i = \{1, 2, \ldots, 10\}$ .

## II. La evolución diferencial

El algoritmo requiere las siguientes variables para su funcionamiento:

- Tamaño de población $\mu$
- Numero de generaciones $g$
- Valor del constante de recombinación $R$
- Valor del constante de diferencias $F$
- Valor del umbral $s$.

Para un problema de $n$ variables el tamaño de población debe ser igual a $10n$, el número de generaciones a probar serán 30, 50, 100, 150.

Si $R, F = [0, 1]$ son iguales a 1 se está usando una búsqueda aleatoria. $s$ está definido en el espacio de la función. Si se tiene un umbral $u$ significa $0.1u$ en las variables. Si se tiene una precisión $1 \times 10^{-5}$, equivale a $1 \times 10^{-6}$ en las variables.

## III. Resultados ED

Lo primero que hay que hacer es cambiar la función objetivo del archivo **evalua.py** de la siguiente manera:

```
import math

def Evalua( _n, vpar ) :
  suma = 0
  for i in range(_n):
    suma = suma +  (vpar[i]-2)*(vpar[i]-5) +
    math.sin( 1.5*math.pi*vpar[i])
    v = 0.1*suma
  return v
```

Código 1. Generar número aleatorio flotante.

Y haciendo las modificaciones al archivo **corre.py** con las especificaciones del problema se obtuvo lo siguiente:

Tabla I
Resultados de algoritmo ED.

| # Generaciones | Mínimo |
|---|---|
| 50 | -2.270374746463768 |
| 100 | -2.8987574063291177 |
| 150 | -3.1971969909333335 |
| 200 | -3.219405568513535 |
| 500 | -3.224518002922993 |
| 1000 | -3.224518002922993 |

Sabemos que el mínimo se encuentra en $x = 3.652887442162$ y $f_1(x) =$ -3.224518019 y con 150 generaciones ya nos aproximamos bastante al resultado.

## IV. Resultados LSHADE

Para el algoritmo de LSHADE se requieren los siguientes datos:

- g_problem_size = 10
- g_max_num_evaluations = 20000
- seed, se ingresa manualmente
- g_pop_size = 180

Los datos anteriores se ingresaron de esa manera dadas las recomendaciones del algoritmo

Como el algoritmo esta escrito en c++ se modifico el archivo **evaluate.cc** para ajustar la función objetivo de la siguiente manera:

```
#include <math.h>
#define PI
    3.14159265358979323846264338327950 29

double evaluate( double *vx, int n )
{
  //double x = vx[0];
  double y = 0.0;
  for (int i = 0; i <= 9; i++) {
    y = y + (vx[i]-2.0)*(vx[i]-5.0) + sin(1.5*
    PI*vx[i]);
      }

  y = 0.1*y;
```

```
13
14    return( y );
15  }
```

Código 2. Generar número aleatorio flotante.

y se obtuvieron los siguientes resultados:

Tabla II
RESULTADOS DE ALGORITMO LSHADE.

| # seed | Mínimo |
|---|---|
| 3 | -3.22451802e+00 |
| 20 | -3.22451802e+00 |
| 100 | -3.22451802e+00 |

Variando el número máximo de evaluaciones a valores mayores no afecta el resultado y es necesario disminuirlo considerablemente para que el algoritmo empiece a alejarse de una buena solución.

## V. CONCLUSIONES

La ED necesito de al menos 500 generaciones para encontrar un buen resultado, esto depende de igual manera de la sintonización que uno haga a los parámetros del algoritmo sin embargo pese a esto hizo un buen trabajo en un tiempo relativamente corto para encontrar una solución aceptable, esto también se puede ver opacado debido a que este algoritmo se ejecuto en python. En cuanto a LSHADE, este algoritmo fue más sencillo de utilizar, las mismas instrucciones del código fuente facilitaron de gran manera su aplicación, debido a que este último esta escrito en c++ su ejecución fue más rápida y con las recomendaciones rápidamente se encontró una solución aceptable.

## REFERENCIAS

[1] Dr. Luis Gerardo de la Fraga. “Apuntes de clase” .