Computación evolutiva: segunda práctica

Andrés Mañas Mañas

February 16, 2017

Abstract

En este documento se desarrolla y prueba una estrategia evolutiva para la búsqueda de los mínimos globales de las funciones Hiper Elipsoide Rotado y función de Rastrigin para 10 dimensiones.

Se representan las funciones para el caso de 2 dimensiones.

Se adjunta y comenta el código de la estrategia evolutiva y se describen las características de la aplicación implementada.

Se aplica la estrategia para buscar los mínimos de las funciones anteriores en el caso de 10 dimensiones.

Se describen los experimentos realizados para determinar los mejores parámetros del algoritmo para la búsqueda de la solución.

Se adjunta tabla de resultados y conclusiones a las que se llega al amparo de los mismos.

Tabla de contenidos

1 Ejercicio 1: Representación gráfica de las funciones para dos dimensiones

Primero vamos a definir una función que nos permite dibujar en 3 dimensiones. Utilizamos la librería matplotlib, bastante conocida en el mundo python.

```
In [1]: %matplotlib inline

        import numpy as np
        from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.pyplot as plt
        import random
        import math
        from matplotlib import cm


        def plot_3d(f,a,b,step,figsize=(8, 8)):
            """
            Dibuja en tres dimensiones una función
            de dos variables

            Args:
                f (function): la función
                a (float): extremo izquierdo del
                    dominio para las dos variables de f
                b (float): extremo derecho del dominio
                    para  las dos variables de f
                step (float): granularidad con la
                    que se  muestrea f
                figsize (int,int): tamaño de la figura
            """
            x = y = np.arange(a,b,step)
            X, Y = np.meshgrid(x, y)
            z = np.array([f([x,y]) for x,y in zip(np.ravel(X),
                                                  np.ravel(Y))])
            Z = z.reshape(X.shape)

            fig = plt.figure(figsize=figsize)
            ax = fig.add_subplot(111, projection='3d')
            ax.set_xlabel('X')
            ax.set_ylabel('Y')
            ax.set_zlabel('Z')
            ax.plot_surface(X, Y, Z, cmap=cm.coolwarm_r)
```

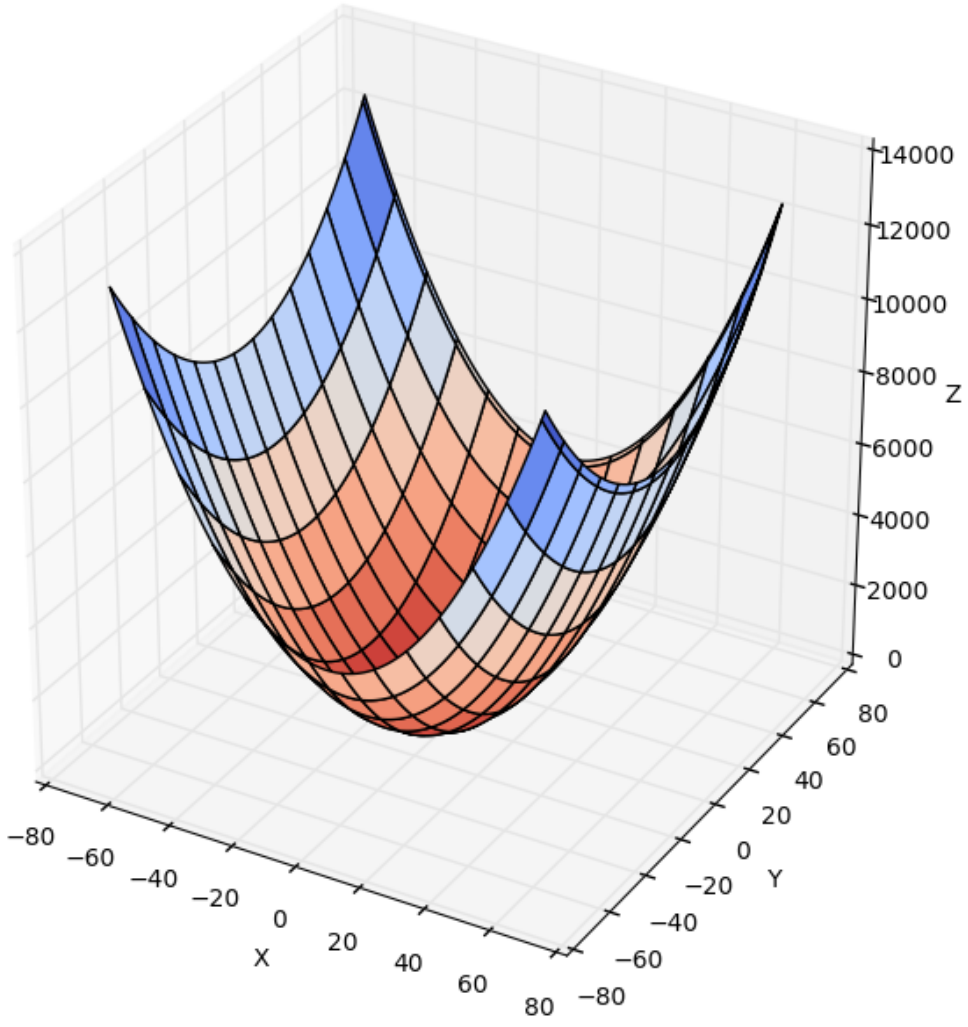
Defino ahora la función hiper-elipsoide rotado y la función de Rastrgin.

```
In [2]: def hiper_elipsoide_rotado(coords):
            """
            Devuelve el valor del hiper-elipsoide
            rotado en el punto dado por coords.

            Args:
                coords: las coordenadas de un
                    punto en un espacio real de
```

```
                dimensión n
        """
        result = 0
        for i in range(1, len(coords)+1):
            for j in range(1, i+1):
                result += coords[j-1]**2
        return result

    def rastrigin(coords):
        """
        Devuelve el valor de la función de
        Rastrigin en el punto dado por coords.

        Args:
            coords: las coordenadas de un punto
                en un espacio real de dimensión n
        """
        result = 0
        for xi in coords:
            result += (xi**2 - 10*math.cos(2*math.pi*xi))
        return 10*len(coords) + result
```

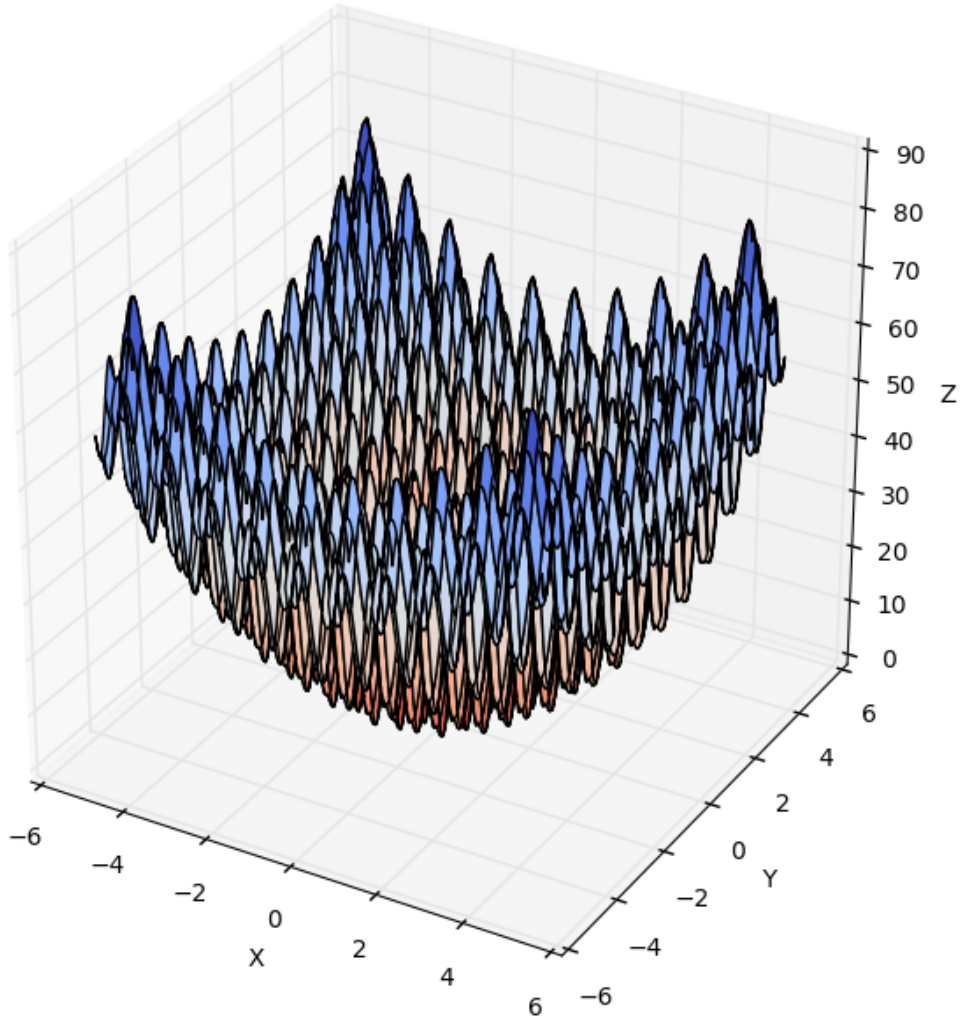
Podemos dibujar ahora el hiper-elipsoide rotado de 2 dimensiones:

```
In [30]: plot_3d(hiper_elipsoide_rotado, -65.54 , 65.54 , 1)
```



Y la función de Rastrigin:

```
In [31]: plot_3d(rastrigin, -5.12 , 5.12 , 0.02)
```



En el caso de la función de Rastrigin, claramente vemos que el renderizado en 3 dimensiones no nos dá suficiente información visual (tiene tantos máximos y mínimos que acaba uno por no saber como es la función).

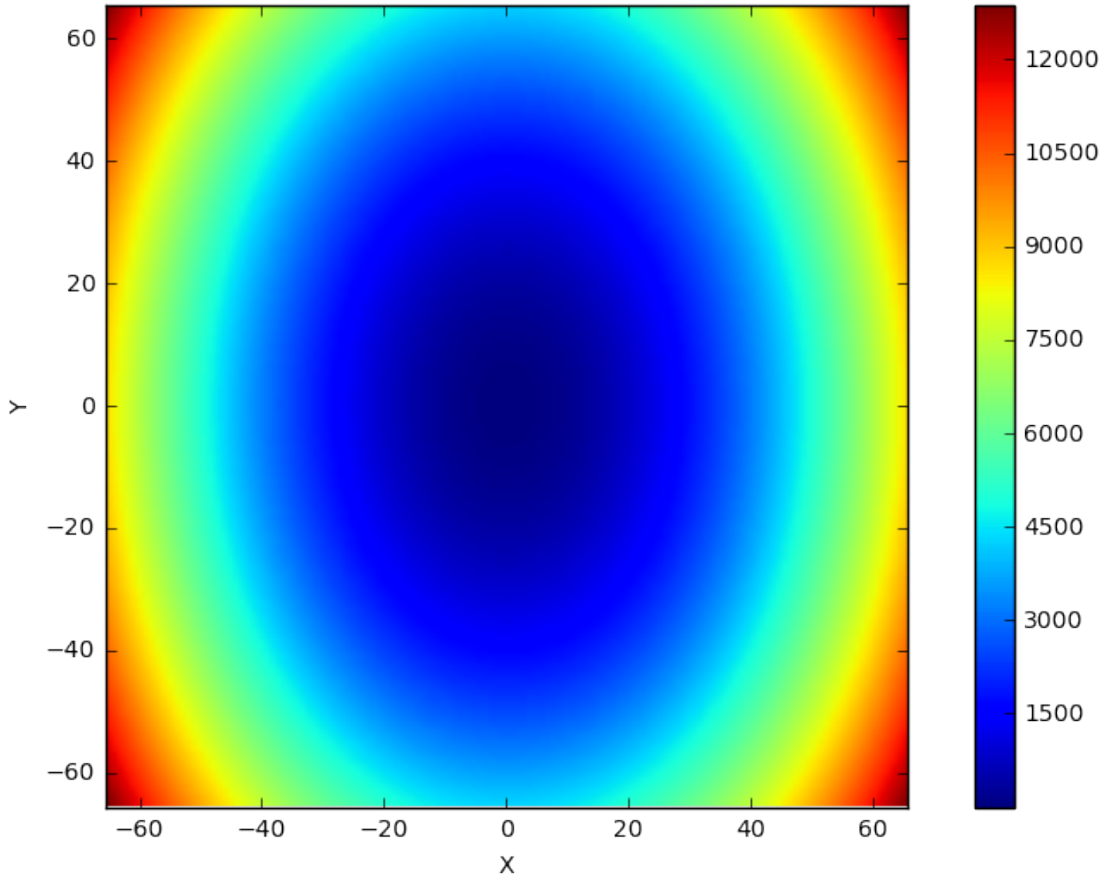
Por eso, será conveniente disponer de la siguiente función que nos permite tener una vista cenital de las funciones representando el valor que adoptan las mismas como un mapa de calor.

```
In [3]: def plot_color_map(f,a,b,step,figsize=(10, 6)):
            """
            Dibuja en dos dimensiones la vista
            cenital de una función de dos variables
            vista como un mapa de intensidad.

            Args:
                f: la función
                a: extremo izquierdo del dominio
```

```
            para las dos variables de f
        b: extremo derecho del dominio
            para las dos variables de f
        step: granularidad con la que se
            muestrea f
        figsize: tamaño de la figura
    """
    x = y = np.arange(a,b,step)
    X, Y = np.meshgrid(x, y)
    z = np.array([f([x,y]) for x,y
                  in zip(np.ravel(X), np.ravel(Y))])
    Z = z.reshape(X.shape)

    fig = plt.figure(figsize=figsize)
    ax = fig.add_subplot(111)
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    plt.imshow(Z,extent=[a,b,a,b])
    plt.colorbar(orientation='vertical')
```

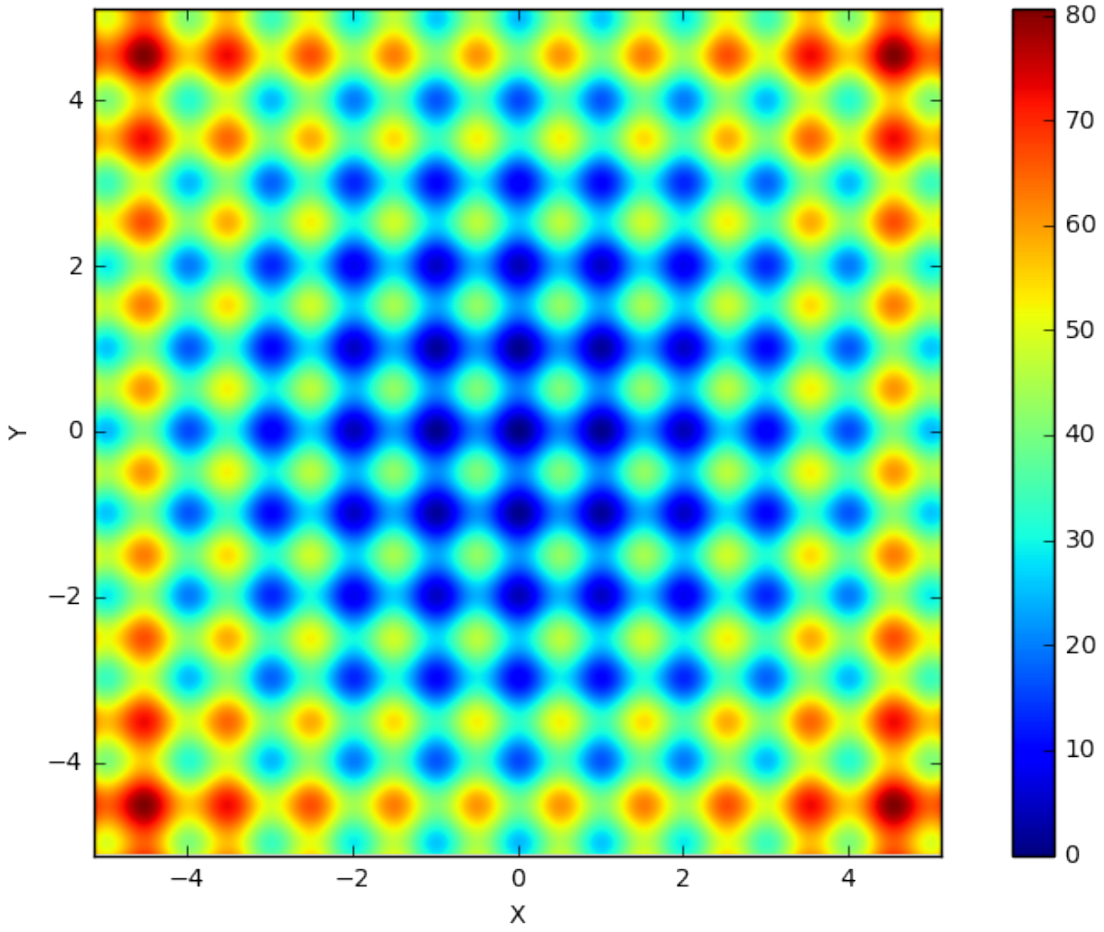
Con esta función podemos ver que efectivamente el hiper elipsoide rotado se corresponde con una superficie con un único mínimo en el origen:

```
In [33]: plot_color_map(hiper_elipsoide_rotado, -65.54 , 65.54 , 1)
```



Del mismo modo, podemos tener una vista más intuitiva del aspecto de la función de Rastrigin para 2 dimensiones. Observamos que en el intervalo en el que se dibuja tiene muchos mínimos locales, cuyo valor se hace más pequeño a medida que se aproximan al origen de coordenadas donde se encuentra su mínimo global.

```
In [34]: plot_color_map(rastrigin, -5.12 , 5.12 , 0.02)
```



2 Ejercicio 2: Implementación de la estrategia evolutiva de búsqueda de mínimos globales

Siguiendo las indicaciones de la práctica, para la búsqueda de los mínimos globales de las funciones hiper elipsoide rotado y función de Rastrigin, se ha inplementado una estrategia evolutiva.

** Este tipo de estratedias están principalmente indicadas en problemas de optimización numérica (como es el caso de los problemas que queremos resolver). Son rápidas y suelen funcionar bien en problemas en los que se pueden representar las soluciones con números reales. **

A lo largo de la evolución, en este tipo de estrategias, no sólo va evolucionando la solución sino también los parámetros que controlan la evolución de la misma.

2.1 Introducción

Mi implemetación es lo suficientemente genérica como para resolver problemas de cualquier dimensión n.

En el caso de mi implementación, represento los individuos como diccionarios con las propiedades: - coords (coordenadas del individuo) - sigmas (parámetros sigma para el operador de mutación).

Dado que en la práctica no se piden mutaciones correlacionadas, no agrego en la representación los parámetros alfa.

Además, dado que en la práctica se piden experimentos de paso único y de n pasos determino que la propiedad sigmas de un individuo puede: - tener un único elemento numérico, en cuyo caso el comportamiento del algoritmo será de paso único - o bien tener un array con el mismo número de elementos que coords, en cuyo caso el comportamiento del algoritmo será de n pasos

Será responsabilidad de la implementación de mis algoritmos interpretar, según la representación de los individuos, si la computación debe realizarse desde el enfoque de paso único (misma sigma controla la variabilidad de todas las coordenadas) o de n pasos (un sigma independiente para el control de la variabilidad de cada coordenada).

Por ejemplo, para el caso de 3 dimensiones, un ejemplo de individuo perteneciente a una ejecución configurada con paso único sería:

```
individual_1 = {'coords':[1,2,3],'sigmas':[1]}
```

Y para el mismo caso de 3 dimensiones, un ejemplo de individuo perteneciente a una ejecución configurada con n pasos sería:

```
individual_2 = {'coords':[1,2,3],'sigmas':[2,3,1]}
```

2.2 Implementación

```
In [4]: from collections import Iterable
        import math
        import inspect

        def rand_individual(domain,dimension,step='n',sigma=1):
            """
            Genera un inviduo aleatoriamente.
            Para conocer la representación utilizada para
            los individuos, léanse los explicaciones
            anteriores.

            Args:
                domain: intervalo real de definición de la
                    función cuyos óptimos buscamos
                dimension: dimensión del experimento
                step: 'n' si el individuo va a participar
                    en una estrategia de n pasos o 'one'
                    si la estrategia es de paso único
                sigma: valor que adoptará/n el/los sigma/s
                    del individuo dependiendo de que la
```

```
            estrategia sea de paso único o de n pasos
        """
        coords=np.random.uniform(domain[0],domain[1],dimension)
        if step=='one':
            sigmas=[sigma]
        elif step=='n':
            sigmas=[sigma]*dimension
        else:
            raise ValueError('Not a valid step')
        return {'coords':coords,'sigmas':sigmas}


    def one_step_mutation(individual,min_sigma,tau,tau_prima=None):
        """
        Muta un individuo en experimentos
        de paso único.

        Args:
            individual: el individuo a mutar
            min_sigma: el valor mínimo de sigma
                permitido en el individuo mutado
            generation: generacion en curso
            tau: intensidad de mutación
            tau_prima: se ignora
        """
        coords,[sigma]=individual['coords'],individual['sigmas']
        sigma_mut=sigma*math.exp(tau*np.random.normal())
        sigma_mut=sigma_mut if min_sigma<sigma_mut else sigma
        coords_mut=[x+sigma_mut*np.random.normal() for x in coords]
        return {'coords':coords_mut,'sigmas':[sigma_mut]}


    def n_step_mutation(individual,min_sigma,tau,tau_prima):
        """
        Muta un individuo en experimentos de n pasos.

        Args:
            individual: el individuo a mutar
            min_sigma: el valor mínimo de sigma
                permitido en los sigmas del
                individuo mutado
            generation: generacion en curso
            tau: intensidad de mutación común
            tau_prima: intensidad de mutación
                por componente
        """
        coords,sigmas=individual['coords'],individual['sigmas']
        common_norm=np.random.normal()
        sigmas_mut=[]
        coords_mut=[]
        for sigma,coord in zip(sigmas,coords):
            sigma_mut=sigma*math.exp(tau_prima*common_norm +
                                     tau*np.random.normal())
            sigma_mut=sigma_mut if min_sigma<sigma_mut else sigma
```

```
            coord_mut=coord + sigma_mut*np.random.normal()
            sigmas_mut += [sigma_mut]
            coords_mut += [coord_mut]
        return {'coords':coords_mut,'sigmas':sigmas_mut}


def discrete_recombination(ind_1,ind_2):
    """
    Recombina dos individuos por el método
    discreto (tomando una a una la coordenada
    y el sigma de uno u otro individuo,
    según el azar).
    Este método acepta tanto individuos que
    participan en estrategias de paso único
    como de n pasos.

    Args:
        ind_1: primer individuo
        ind_2: segundo individuo

    Returns:
        la recombinación de ambos individuos
    """
    sigmas_comb=[]
    coords_comb=[]
    both=[ind_1,ind_2]
    one_step= 1==len(ind_1['sigmas'])
    for i in range(len(ind_1['coords'])):
        j=np.random.choice([0,1])
        coords_comb+=[both[j]['coords'][i]]
        if one_step:
            sigmas_comb=[both[j]['sigmas'][0]]
        else:
            sigmas_comb+=[both[j]['sigmas'][i]]
    return {'coords':coords_comb,'sigmas':sigmas_comb}


def intermediate_recombination(ind_1,ind_2):
    """
    Recombina dos individuos por el método
    intermedio (tomando una a una la
    coordenada y el sigma mediosde ambos
    individuos).
    Este método acepta tanto individuos que
    participan en estrategias de paso único
    como de n pasos.

    Args:
        ind_1: primer individuo
        ind_2: segundo individuo
    """
    return {'coords':[(a+b)/2 for a,b in
                      zip(ind_1['coords'],ind_2['coords'])],
            'sigmas':[(a+b)/2 for a,b in
```

```
                        zip(ind_1['sigmas'],ind_2['sigmas'])]}


def mu_comma_lambda(mus,lambdas,f,mu):
    """
    Selección de tipo mu comma lambda
    (selecciona los mejores individuos del
    conjunto de lambas/hijos).

    Args:
        mus: los padres de la generación
            en curso
        lambdas: los hijos de la generación
            en curso
        f: la función de evaluación de la
            calidad de los individuos
            (función fitness)
        mu: número de individuos a seleccionar
    """
    return sorted(lambdas, key=lambda x:f(x['coords']))[0:mu]


def mu_plus_lambda(mus,lambdas,f,mu):
    """
    Selección de tipo mu plus lambda
    (selecciona los mejores individuos del
    conjunto de mus/padres y lambas/hijos).

    Args:
        mus: los padres de la generación en curso
        lambdas: los hijos de la generación en curso
        f: la función de evaluación de la calidad
            de los individuos (función fitness)
        mu: número de individuos a seleccionar
    """
    return sorted(mus+lambdas,
                  key=lambda x:f(x['coords']))[0:mu]


def termination_cond_met(best,generation,max_generations,
                         termination_delta,f):
    """
    Determina si se dan las condiciones para que
    la evolución acabe por
     - haber evolucionado durante demasiadas
         generaciones
     - haber alcanzado el mejor individuo de la
         generación en curso un fitness
         suficientemente bueno

    Args:
        best: array con el histórico de los
            mejores individuos de cada generación
```

```
            generation: generación en curso
            max_generations: máximo número de
                generaciones permitido en el experimento
            termination_delta: error mínimo del mejor
                individuo. Si el fitness del mejor
                individuo (cercanía al mínimo global
                que sabemos que es 0) es inferior
                a este valor, el experimento se da
                por concluido.
            f: la función de evaluación de la calidad
                de los individuos (función fitness)
        """
        return max_generations<=generation \
               or 0<len(best) \
                  and abs(f(best[-1]['coords']))<termination_delta


    def live(domain, dimension, f,
             step='n',
             selection='mu_comma_lambda',
             recombination='intermediate',
             mu=30,
             lamb=200,
             sigma=1,
             min_sigma=.0,
             max_generations=1000,
             termination_delta=.0,
             tau_factor=1,
             tau_prima_factor=1,
             mutate_sigmas_every=1):
        """
        Lleva a cabo la estrategia evolutiva de
        búsqueda de mínimos globales de las funciones
        hiper elipsoide rotado o función de Rastrigin,
        tal como se ha descrito  en el presente
        documento.

        Args:
            domain: dominio de definición de la
                función cuyos mínimos globales
                se buscan
            dimension: demensión del espacio en el
                que se define la función f
            f: función cuyos mínimos globales
                buscamos con la estrategia evolutiva
            step: determina si la estrategia será
                de paso único 'one' o de n pasos 'n'.
                El valor por defecto es 'n'.
            selection: determina si la selección
                de hijos  se hará con el método
                mu comma lambda 'mu_comma_lambda'
                o mu plus lambda  'mu_plus_lambda'.
                Valor por defecto 'mu_comma_lambda'.
            recombination: determina si la recombinación
```

```
                de individuos se hará por el método
                discreto 'discrete' o intermedio
                'intermediate'.
                Valor por defecto 'intermediate'
            mu: número de padres en cada iteración.
                El valor por defecto es de 30.
            lamb: número de hijos en cada interación.
                El valor por defecto es de 200.
            sigma: valor por defecto que se aplicará
                en la  inicialización de los sigmas
                de todo individuo.
                Valor por defecto 1.
            min_sigma: valor mínimo permitido de los
                sigmas de un individuo.
                Valor por defecto 0.0001.
            max_generations: número de generaciones
                máximo permitido en el experimento.
                Valor por defecto 1000:
            termination_delta: error mínimo del mejor
                individuo.
                Si el fitness del mejor individuo
                (cercanía al mínimo global que sabemos
                que es 0) es inferior a este valor,
                el experimento se da por concluido.
                Valor por defecto .0
            tau_factor: factor de multiplicación del
                valor por defecto de tau, 1.
            tau_prima_factor: factor de multiplicación
                del valor por defecto de tau prima, 1.

        Returns:
            - el mejor individuo encontrado tras la evolución
            - y un array con el histórico del mejor inviduo
                en cada generación.
        """

    #     print [locals()[arg] for arg in inspect.getargspec(live).args[3:]]

        if step=='n':
            mut_f=n_step_mutation
            tau=tau_factor/math.sqrt(2*dimension)
            tau_prima=tau_prima_factor/math.sqrt(2*math.sqrt(dimension))
        elif step=='one':
            mut_f=one_step_mutation
            tau=tau_factor/math.sqrt(dimension)
            tau_prima=None

        if recombination=='intermediate':
            rec_f=intermediate_recombination
        elif recombination=='discrete':
            rec_f=discrete_recombination

        if selection=='mu_comma_lambda':
            sel_f=mu_comma_lambda
```

```
        elif selection=='mu_plus_lambda':
            sel_f=mu_plus_lambda

        t,best=0,[]
        mus=[rand_individual(domain,dimension,step,sigma)
             for i in range(mu)]
        while not termination_cond_met(best,t,max_generations,
                                       termination_delta,f):
            recombinations=[rec_f(*random.sample(mus,2))
                            for i in range(lamb)]
            mutations=[mut_f(ind,min_sigma,tau,tau_prima)
                       for ind in recombinations]
            mus=sel_f(mus,mutations,f,mu)
            best+=[mus[0]]
            t+=1
        return best[-1],best
```

Defino además una función que nos permite lanzar experimentos en paralelo para reducir el tiempo de espera en completarse un conjunto de experimentos.

```
In [10]: from functools import partial
         from multiprocessing import Pool

         def p_exps(name,f,n=20):
             """
                 Ejecuta en paralelo una batería
                 de experimentos.
             """
             p = Pool()
             result = {name:p.map(f,range(n))}
             p.close()
             p.terminate()
             return result
```

2.3 Descripción de la estrategia

Observando la función live, que es la que coordina la evolución, se pueden apreciar los aspectos que siguen.

Dado un experimento, es obligatorio indicar el número de dimensiones que se van a utilizar, la función f (hiperelipsoide rotada o rastrigin) y el dominio de definición de la función f.

Todo experimento puede realizarse de paso único o de n pasos, con recombinación intermedia o discreta y con selección de descendencia por (μ, λ) o $(\mu + \lambda)$.

Los valores de mu, lamb, sigma, min_sigma, max_generations o termination_delta se dan por defecto, aunque pueden ser configurados en cada ejecución del experimento.

La estrategia en sí es bastante simple, un individuo se representa por unas coordenadas en el espacion n-dimensional y por uno ('one' step) o tantos sigmas como coordenadas tenga el individuo ('n' step), que gobierna la intensidad de la mutación tanto de las coordenadas como de los propios sigmas.

En esta implementación, por hacerla más sencilla de entender y de mantener, se toman los valores de tau por defecto que se indican en el documento base de referencia. No se permite parametrizar tales valores sino que todos los experimentos tendrán los mismos en función del número de dimensiones.

Con estas explicaciones, el algoritmo simplemente hace esto: - genera una población inicial al azar en el intervalo de definición de la función - mientras el fitness del mejor individuo no sea suficientemente bueno o no se haya alcanzado el máximo de generaciones permitido - generar lamd individuos recombinando la población actual - mutar los individuos lamb anteriores - establecer la nueva población como los mejores individuos seleccionados de entre los individuos mutados y la población actual (dependiendo de la función de selección) - guardar el mejor de los resultantes en el paso anterior en el histórico de mejores individuos por generación - repetir hasta que no se tenga un individuo con suficiente fitness o no se hayan agotado el máximo de generaciones permitidas - devolvel el mejor individuo y el histórico de mejores individuos por generación

2.4 Pruebas en dos dimensiones

Antes de pasar a hacer los experimentos en 10 dimensiones que se piden en la práctica es mejor ver como se comporta el algoritmo en el caso de 2 dimensiones.

La ventaja de hacerlo en 2 dimensiones es que podemos disponer de un gráfico qeu muestre visualmente "la traza" de la evolución.

A tal efecto, la siguiente función nos será muy útil junto a la función plot_color_map que se definión anteriormente.

```
In [6]: def add_scatter(individuals):
            """
            Dibuja un scatter con la secuencia de
            coordenadas que se corresponden con una
            secuencia de individuos según los
            represento en la presente práctica
            (se explica más arriba).
            Permite representar el camino que sigue la
            evolución de la mejor solución que el
            algoritmo encuentra en cada generación.

            Args:
                individuals: array de individuos
            """
            x,y=[],[]
            for ind in individuals:
                x+=[ind['coords'][0]]
                y+=[ind['coords'][1]]
            colors=cm.rainbow(np.linspace(0, 1, len(x)))
            plt.scatter(x, y, c=colors, s=100)
```

Igualmente, nos será uy útil una función que dibuje como de próximo está un individuo al óptimo global buscado, que sabemos que en ambos casos es 0. Algo que nos sirva para representar la evolución del fitness del mejor individuo.

Nótese que la siguiente función hace una transformación logarítmica de los valores por f de cada individuo, para evitar que los gráficos salgan aberrados y por lo tanto que no sirvan para interpretarse con provecho.

```
In [7]: def plot_inv_log_fitness(f,individuals):
            """
            Dibuja el valor dado por f en una población
            de individuos en escala logarítmica.
            Puede interpretarse con el inverso
```

```
        del fitness de los individuos.

        Se utiliza escala logarítmica porque si no
        los gráficos salen un poco aberrados.

        Args:
            f: una función n variada
            individuals: array de individuos con
                n coordenadas cada uno.
        """
        plt.plot([math.log(f(ind['coords'])+1)
                  for ind in individuals])
```

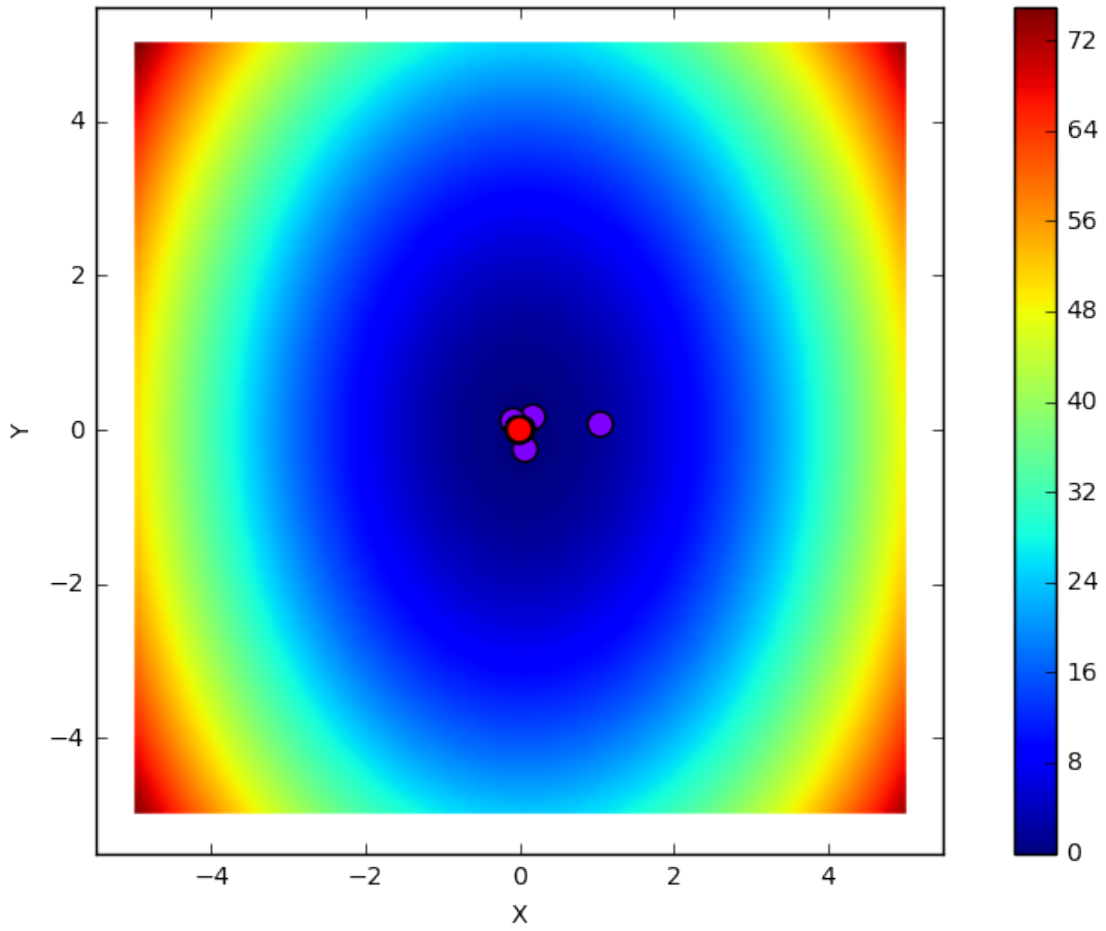
2.4.1 Hiperelipsoide rotado en 2 dimensiones

De este modo, para el hiper elipsoide rotado de 2 dimensiones, con todos los valores por defecto, podemos llevar a cabo una búsqueda del mínimo global:

```
In [38]: best,all=live([-65.54 , 65.54],2,hiper_elipsoide_rotado)
```

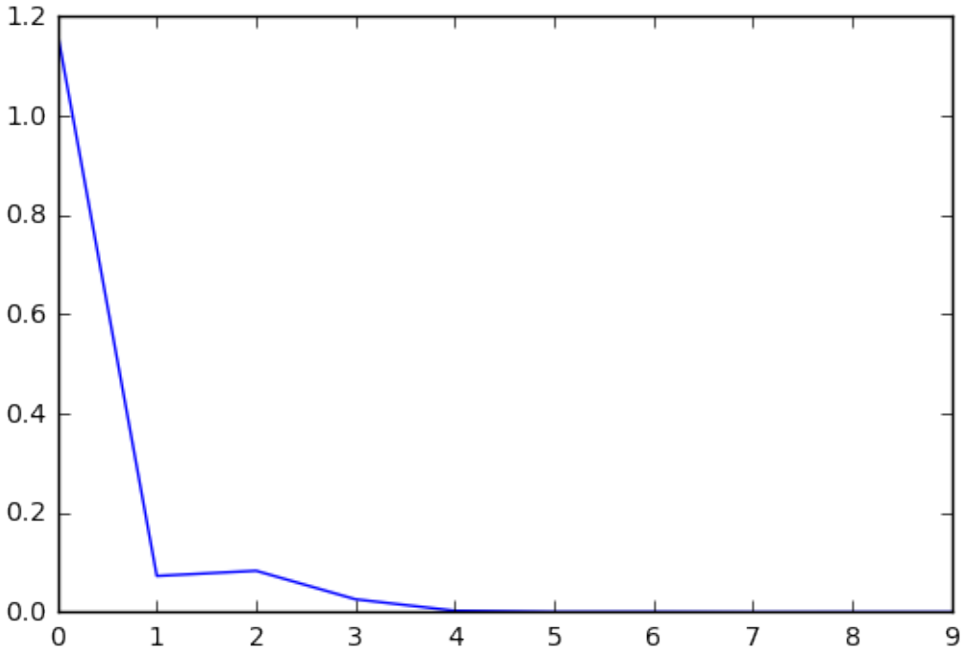
Y ver gráficamente como el mejor individuo de cada generación se va aproximando al origen de coordenadas. Nótese que el aprendizaje es tan rápido que los individuos muy rápidamente se ubican muy próximos al origen de coordenadas. Por eso, reduzco a [-5,5] el intervalo en el que dibujo los gráficos.

```
In [39]: plot_color_map(hiper_elipsoide_rotado,-5,5,0.1)
         add_scatter(all)
```



Podemos ver la evolución del fitness del mejor individuo (en escala logarítmica e inversa):

```
In [42]: plot_inv_log_fitness(hiper_elipsoide_rotado,all[:10])
```

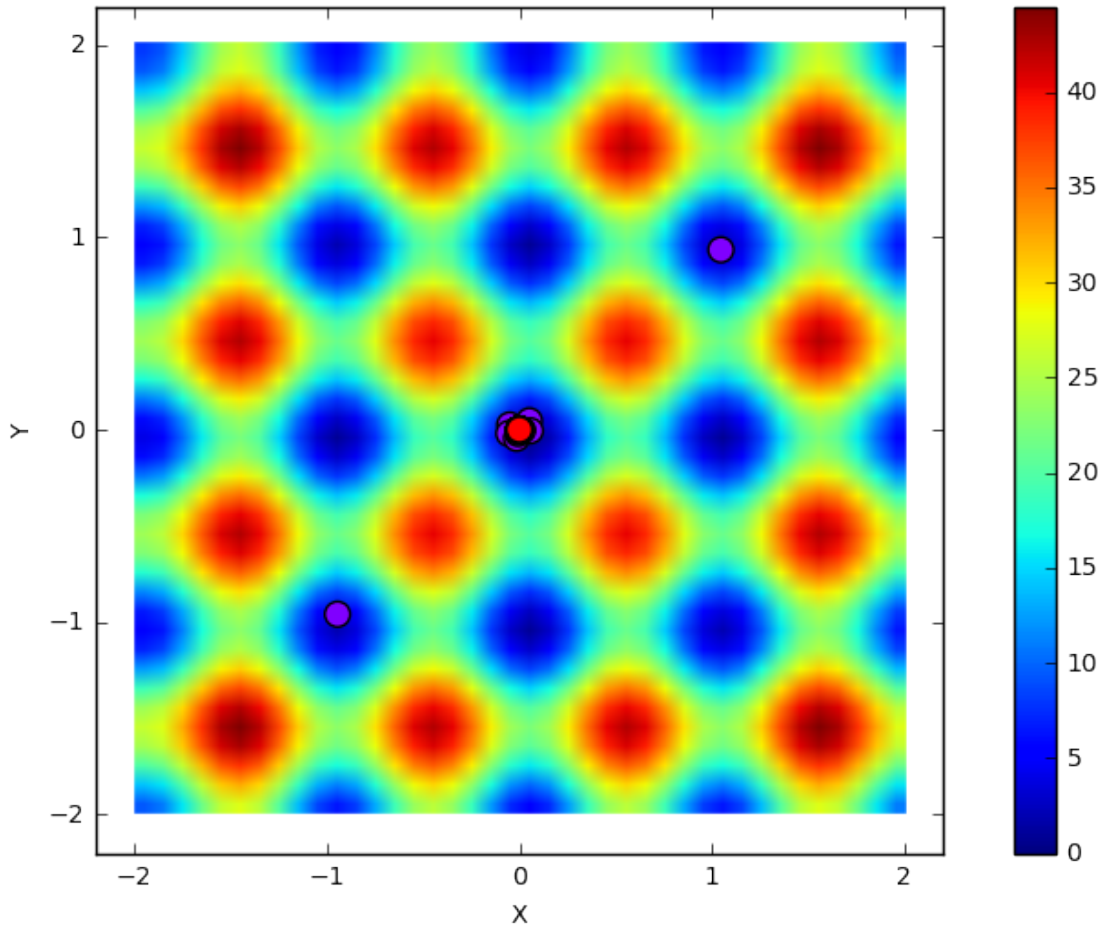


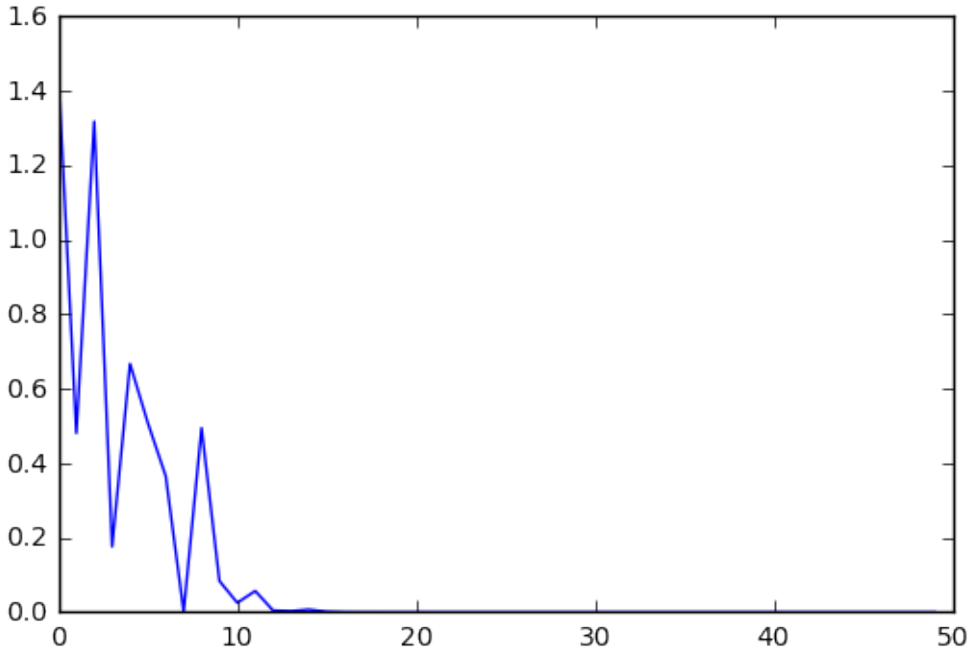
Observamos que encuentra el mínimo muy rapido, en 5 generaciones.

2.4.2 Rastrigin en 2 dimensiones

Equivalenteme, para el caso de la función de Rastrigin, en dos dimensiones, tenemos:

```
In [44]: best,all=live([-5.12 , 5.12],2,rastrigin)

         # Reducimos el intervalo en el que dibujamos
         # para mayor claridad del dibujo
         plot_color_map(rastrigin,-2,2,0.1)
         add_scatter(all)
         plt.show()
         plot_inv_log_fitness(rastrigin,all[:50])
```


2
1
0
−1
−2
Y
−2
−1
0
1
2
X
40
35
30
25
20
15
10
5
0



Nótese que en el caso de dos dimensiones, el mínimo de la función de Rasttrigin también se encuentra muy rápido.

Y son esto, más o menos, podemos hacernos una idea del herramentaje que tenemos a nuestra disposición para continuar con la práctica.

3 Ejercicio 3: Búsqueda del mínimo del hiper-elipsoide rotado de 10 dimensiones con selección mu coma lambda y paso único

Realizamos 20 ejecuciones y guardamos los resultados en la variable **hiper_one_comma**.

Analizaremos estos resultados posteriormente.

```
In [9]: def partial(i):
            return live([-65.54 , 65.54],10,
                        hiper_elipsoide_rotado,
                        step='one',
                        selection='mu_comma_lambda')[1]
        hiper_one_comma =  p_exps('Hiper one comma',partial)
```

4 Ejercicio 4: Búsqueda del mínimo del hiper-elipsoide rotado de 10 dimensiones con selección mu coma lambda y paso n

Realizamos 20 ejecuciones y guardamos los resultados en la variable **hiper_n_comma**.

Analizaremos estos resultados posteriormente.

```
In [102]: def partial(i):
              return live([-65.54 , 65.54],10,
                          hiper_elipsoide_rotado,
                          step='n',
                          selection='mu_comma_lambda')[1]
          hiper_n_comma =  p_exps('Hiper n   comma',partial,n=4)
```

5 Ejercicio 5: Búsqueda del mínimo del hiper-elipsoide rotado de 10 dimensiones con selección mu plus lambda y paso único

Realizamos 20 ejecuciones y guardamos los resultados en la variable **hiper_one_plus**.

Analizaremos estos resultados posteriormente.

```
In [103]: def partial(i):
              return live([-65.54 , 65.54],10,
                          hiper_elipsoide_rotado,
                          step='one',
                          selection='mu_plus_lambda')[1]
          hiper_one_plus =  p_exps('Hiper one plus ',partial)
```

6 Ejercicio 6: Búsqueda del mínimo del hiper-elipsoide rotado de 10 dimensiones con selección mu plus lambda y paso n

Realizamos 20 ejecuciones y guardamos los resultados en la variable **hiper_n_plus**.

Analizaremos estos resultados posteriormente.

```
In [104]: def partial(i):
              return live([-65.54 , 65.54],10,
                          hiper_elipsoide_rotado,
                          step='n',
                          selection='mu_plus_lambda')[1]
          hiper_n_plus =  p_exps('Hiper n   plus ',partial)
```

7 Métricas de evaluación de configuraciones

Para justificar la conveniencia de una otra configuración de la estrategia evolutiva utilizada para la búsqueda de los mínimos del hiper-elipsoide rotado, primero tenemos que definir las funciones de rendimiento con las que medir los resultados.

Lógicamente, estas funciones serán igualmente utilizables en el caso de la función de Rastrigin.

```
In [11]: def SR(results,f,epsilon=1e-6):
             """
             Calcula el porcentaje de ejecuciones que
             acaban con éxito.
             El éxito se define como la situación en
             la que el fitness del mejor individuo
             dista del mínimo global en menos de epsilon.
```

```
        Args:
            results (array): un array cuyos elementos
                son arrays de individuos (el histórico
                de los mejores individuos de cada
                generación de una ejecución del
                algoritmo)
            f (function): la función que cuyos
                se buscan en la ejecución que
                generara los resultados de entrada
            epsilon (float): distancia máxima al
                mínimo global de la función que
                determina si una ejecución es
                considerada éxito o no.
                valor por defecto 1e-5

        Returns:
            float: el porcentaje de ejecuciones que
                acaban con éxito
        """
        best_inds=[hist[-1] for hist in results]
        fitnesses=[f(ind['coords']) for ind in best_inds]
        successes=filter(lambda x: x < epsilon, fitnesses)
        return float(len(successes))/len(fitnesses)


    def AES(results):
        """
        Calcula el número medio de evaluaciones
        necesario hasta dar con la solución.
        Esta métrica tiene sentido dado que
        nuestra evaluaciones se dotan de
        condiciones de terminación sin
        necesariamente agotar el máximo
        de generaciones permitidas.

        Args:
            results (array): un array cuyos elementos
                son arrays de individuos (el histórico
                de los mejores individuos de cada
                generación de una ejecución del
                algoritmo)

        Returns:
            float: el número medio de evaluaciones
                hasta dar con la solución
        """
        return float(sum([len(x) for x in results]))/len(results)


    def MBF(results,f):
        """
        Calcula la media de los valores de fitness
        del mejor individuo encontrado (el individuo
```

```
        devuelto por el algoritmo) en un
        conjunto de ejecuciones del algoritmo
        evolutivo.

        Args:
            results (array): un array cuyos elementos
                son arrays de individuos (el histórico
                de los mejores individuos de cada
                generación de una ejecución del
                algoritmo)
            f (function): la función que cuyos
                se buscan en la ejecución que
                generara los resultados de entrada

        Returns:
            float: la media de los valores de fitness
                de los mejores individuos encontrados
                en varias ejecuciones del algortimo
        """
        best_inds=[hist[-1] for hist in results]
        fitnesses=[f(ind['coords']) for ind in best_inds]
        return sum(fitnesses)/len(fitnesses)
```

Por último, necesitamos también una función que dibuje la tabla de resultados de un conjunto de experimentos y que funcione bién cuando este documento se exporta a latex. Ésta es:

```
In [12]: import pandas as pd
         pd.set_option('display.notebook_repr_html', True)

         def _repr_latex_(self):
         #     return "\centering{%s}" % self.to_latex()
             return self.to_latex()

         pd.DataFrame._repr_latex_ = _repr_latex_


         def render_metrics(metrics):
             """
             Dibuja de forma tabulada las métricas de
             rendimiento de una lista de experimentos.

             Args:
                 metrics (array): array donde cada
                     elemento es un array compuesto de
                     ['nombre del experimento',
                      SR del exp,
                      AES del exp, ,
                      MBF del exp]
             """
             data = {'SR' : [x[1] for x in metrics],
                     'AES': [x[2] for x in metrics],
                     'MBF': [x[3] for x in metrics]}
             return pd.DataFrame(data,
                                 index=[x[0] for x in metrics],
                                 columns=['SR','AES','MBF'])
```

```
def render_experiments(experiments,f,epsilon):
    """
    Dibuja de forma tabulada un diccionario
    de experimentos.
    """
    metrics = [[k, SR(v, f, epsilon=epsilon),
                AES(v), MBF(v, f)]
               for k,v in experiments.iteritems()]
    return render_metrics(metrics)

def merge_dicts(*dict_args):
    """
    Given any number of dicts, shallow copy and merge into a new dict,
    precedence goes to key value pairs in latter dicts.
    """
    result = {}
    for dictionary in dict_args:
        result.update(dictionary)
    return result
```

8 Ejercicio 7: Resultados para la función hiper-elipsoide rotado

Una vez disponemos de las funciones anteriores, podemos calcular y mostrar la tabla de resultados para la búsqueda del mínimo global del hiper-elipsoide rotado de 10 dimensiones, tomando 30 mus y 200 lambdas en las distintas configuraciones que hemos probado.

```
In [105]: render_experiments(merge_dicts(hiper_one_comma,
                                          hiper_n_comma,
                                          hiper_one_plus,
                                          hiper_n_plus),
                              hiper_elipsoide_rotado, 1e-10)
```

Out[105]:

	SR	AES	MBF
Hiper one plus	1.0	1000.0	1.105618e-77
Hiper one comma	1.0	1000.0	3.186863e-64
Hiper n comma	1.0	1000.0	6.218410e-233
Hiper n plus	1.0	1000.0	7.882819e-241

Los resultados son en general muy satisfactorios. Todos se quedan a una distancia menor de 1e-10 del mínimo. En realidad, a menos de 4e-64.

Se observa claramente que la estrategia de n pasos profundiza mucho más en la solución. En parte, es de entender, al tratarse de una función con un único mínimo.

Dado que como veremos más tarde, la búsqueda del óptimo de la función de Rastrigin ha sido bastante más complicada, prefiero retrasar hasta los comentarios de ésta las distintas experimentaciones que he hecho para intentar ajustar al máximo los parámetros de la estrategia. O dicho de otro modo, para

la función hiper-elipsoide rotado los resultados son tan buenos que prefiero centrar todos los esfuerzos en la función de Rastrigin.

Si acaso, para comprobar que efectivamente indicando un error de terminación, la convergencia se produce en pocas generaciones, podemos verlo con:

```
In [95]: def partial(i):
             return live([-65.54 , 65.54],10,
                         hiper_elipsoide_rotado,
                         step='one',
                         selection='mu_comma_lambda',
                         termination_delta=1e-10)[1]
         hiper_one_comma_delta =  p_exps('Hiper one comma delta ',partial)

         def partial(i):
             return live([-65.54 , 65.54],10,
                         hiper_elipsoide_rotado,
                         step='n',
                         selection='mu_comma_lambda',
                         termination_delta=1e-10)[1]
         hiper_n_comma_delta =  p_exps('Hiper n   comma delta ',partial)

         def partial(i):
             return live([-65.54 , 65.54],10,
                         hiper_elipsoide_rotado,
                         step='one',
                         selection='mu_plus_lambda',
                         termination_delta=1e-10)[1]
         hiper_one_plus_delta =  p_exps('Hiper one plus  delta ',partial)

         def partial(i):
             return live([-65.54 , 65.54],10,
                         hiper_elipsoide_rotado,
                         step='n',
                         selection='mu_plus_lambda',
                         termination_delta=1e-10)[1]
         hiper_n_plus_delta =  p_exps('Hiper n   comma delta ',partial)

         render_experiments(merge_dicts(hiper_one_comma_delta,
                                        hiper_n_comma_delta,
                                        hiper_one_plus_delta,
                                        hiper_n_plus_delta),
                            hiper_elipsoide_rotado, 1e-10)
```

Out[95]:

	SR	AES	MBF
Hiper n comma delta	1.0	79.10	7.006832e-11
Hiper one comma delta	1.0	165.15	9.256651e-11
Hiper one plus delta	1.0	134.10	8.620310e-11
Hiper n plus delta	1.0	84.65	6.995538e-11

Nuevamente comprobamos que la estrategia de n pasos es más rápida y que en todos los experimentos se

encuentra una solucíon que satisfaga el delta de terminación (error mínimo). Es decir, la tasa de éxito es del 100% en todos los casos.

9 Ejercicio 9: Análisis equivalente para la función de Rastrigin

Ya que tenemos todas las herramientas necesarias, basta simplemente con realizar los experimentos y mostrar los resultados:

```
In [106]: def partial(i):
              return live([-5.12,5.12],10,
                          rastrigin,
                          step='one',
                          selection='mu_comma_lambda')[1]
          rast_one_comma =  p_exps('Rastrigin one comma ',partial)

          def partial(i):
              return live([-5.12,5.12],10,
                          rastrigin,
                          step='n',
                          selection='mu_comma_lambda')[1]
          rast_n_comma =  p_exps('Rastrigin n   comma ',partial)

          def partial(i):
              return live([-5.12,5.12],10,
                          rastrigin,
                          step='one',
                          selection='mu_plus_lambda')[1]
          rast_one_plus =  p_exps('Rastrigin one plus ',partial)

          def partial(i):
              return live([-5.12,5.12],10,
                          rastrigin,
                          step='n',
                          selection='mu_plus_lambda')[1]
          rast_n_plus =  p_exps('Rastrigin n   plus ',partial)

          render_experiments(merge_dicts(rast_one_comma,
                                         rast_n_comma,
                                         rast_one_plus,
                                         rast_n_plus),
                             rastrigin, 1e-10)
```

Out[106]:

	SR	AES	MBF
Rastrigin one comma	0.3	1000.0	1.542187
Rastrigin n plus	0.0	1000.0	3.333113
Rastrigin n comma	0.1	1000.0	1.840674
Rastrigin one plus	0.0	1000.0	2.686389

En el caso de Rastrigin vemos que los resultados son en general decepcionantes con los valores por defecto de los parámetros del algoritmo.

Las estrategias que descartan los mus en la selección ofrecen mejor rendimiento, pero en general los rendimientos son malos.

De hecho, la media de distancias al mínimo global no baja de 1.54, cosa decepcionante por completo.

He hecho miles de experimentos cambiando uno a uno todos los posibles parámetros de la estrategia: sigmas, valor mínimo de sigma, taus, mus, lambdas...

10 Ejercicio 8: Búsqueda de la mejor configuración de parámetros

A continuación muestro el modo de búsqueda de la mejor configuración de parámetros que he seguido para la función de Rastrigin. Con dije antes, en el caso del hiperelipsoide rotado, los resultados obtenidos con los valores por defecto son tan buenos que he decidido no proceder a tal análisis.

Básicamente, he preparado un script que realiza una batería de pruebas explorando todas las combinaciones de paso, selección y combinación y probando con factores de multiplicación de taus y taus primas 0.25, 0.5, 1, 2 y 4.

Obsérvese que el ataque por fuerza siguiente para descubrir la mejor parametrización lo limito a sólo 250 generaciones. La razón es que después de acabar esta práctica me gustaría seguir conservando el ordenador en el que la realizo y todas sus CPUs. Por favor, disculpe la libertad que me he tomado en esta limitación.

```
In [43]: import itertools as it

         def p_live(step,s,r,t,t_p,i):
             return live([-5.12,5.12],
                         10,
                         rastrigin,
                         step=step,
                         selection=s,
                         recombination=r,
                         tau_factor=t,
                         tau_prima_factor=t_p,
                         max_generations=250)[1]

         def do_exploration():
             result = []
             sels = ['mu_comma_lambda','mu_plus_lambda']
             recs = ['intermediate','discrete']
             taus = [0.25, 0.5, 1, 2, 4]
             taus_p = [0.25, 0.5, 1, 2, 4]
             template = 'Rast n {} {} tau={} tau prima={}'
             for s,r,t,t_p in it.product(sels,recs,taus,taus_p):
                 name = template.format(r,s,t,t_p)
                 print name
                 exps = p_exps(name,partial(p_live,'n',s,r,t,t_p),n=20)
                 metrics = render_experiments(exps,rastrigin,epsilon=0.1)
                 result += [metrics]

             template = 'Rast one {} {} tau={}'
             for s,r,t in it.product(sels,recs,taus):
```

```
            name = template.format(r,s,t)
            print name
            exps = p_exps(name,partial(p_live,'one',s,r,t,0),n=20)
            metrics = render_experiments(exps,rastrigin,epsilon=0.1)
            result += [metrics]
        return result


exploration = do_exploration()
pd.set_option('display.max_rows', 500)
result = pd.concat(exploration)
result.sort_values('SR')
```

Out[43]:

	SR	AES	MBF
Rast n inter plus tau=0.5 tau prima=4	0.00	250.0	4.328072
Rast n disc plus tau=0.25 tau prima=2	0.00	250.0	4.129081
Rast n disc plus tau=0.25 tau prima=1	0.00	250.0	3.731096
Rast n disc plus tau=0.25 tau prima=0.25	0.00	250.0	28.410628
Rast n inter plus tau=4 tau prima=4	0.00	250.0	3.830592
Rast n inter plus tau=4 tau prima=2	0.00	250.0	3.581852
Rast n inter plus tau=4 tau prima=1	0.00	250.0	4.118031
Rast n inter plus tau=4 tau prima=0.5	0.00	250.0	8.075069
Rast n inter plus tau=4 tau prima=0.25	0.00	250.0	18.080988
Rast n inter plus tau=2 tau prima=4	0.00	250.0	5.124038
Rast n inter plus tau=2 tau prima=2	0.00	250.0	4.278324
Rast n inter plus tau=2 tau prima=0.5	0.00	250.0	16.547862
Rast n inter plus tau=2 tau prima=0.25	0.00	250.0	26.946989
Rast n inter plus tau=1 tau prima=4	0.00	250.0	4.925047
Rast n inter plus tau=1 tau prima=1	0.00	250.0	4.427567
Rast n inter plus tau=1 tau prima=0.5	0.00	250.0	20.888121
Rast n inter plus tau=1 tau prima=0.25	0.00	250.0	27.522825
Rast n inter plus tau=0.5 tau prima=1	0.00	250.0	3.681348
Rast n inter plus tau=0.5 tau prima=0.5	0.00	250.0	22.831933
Rast n inter plus tau=0.5 tau prima=0.25	0.00	250.0	25.594012
Rast n inter plus tau=0.25 tau prima=4	0.00	250.0	4.328071
Rast n inter plus tau=0.25 tau prima=2	0.00	250.0	4.129080
Rast n disc plus tau=0.5 tau prima=0.25	0.00	250.0	25.855713
Rast n disc plus tau=0.5 tau prima=0.5	0.00	250.0	4.278323
Rast n disc plus tau=0.5 tau prima=1	0.00	250.0	4.328071
Rast n disc plus tau=0.5 tau prima=2	0.00	250.0	4.726054
Rast one disc plus tau=1	0.00	250.0	2.984877
Rast one disc plus tau=0.25	0.00	250.0	27.031063
Rast one inter plus tau=4	0.00	250.0	4.129080
Rast one inter plus tau=0.5	0.00	250.0	23.401600
Rast one inter plus tau=0.25	0.00	250.0	25.288749
Rast one disc comma tau=4	0.00	250.0	5.024542
Rast one disc comma tau=2	0.00	250.0	2.935129
Rast one disc comma tau=1	0.00	250.0	3.183867
Rast one inter comma tau=2	0.00	250.0	2.736137
Rast n disc plus tau=4 tau prima=4	0.00	250.0	4.136850

	SR	AES	MBF
Rast n inter plus tau=0.25 tau prima=0.5	0.00	250.0	22.236333
Rast n disc plus tau=4 tau prima=0.5	0.00	250.0	4.228575
Rast n disc plus tau=2 tau prima=2	0.00	250.0	5.522023
Rast n disc plus tau=2 tau prima=1	0.00	250.0	3.979835
Rast n disc plus tau=2 tau prima=0.5	0.00	250.0	5.770760
Rast n disc plus tau=2 tau prima=0.25	0.00	250.0	4.974795
Rast n disc plus tau=1 tau prima=4	0.00	250.0	4.990099
Rast n disc plus tau=1 tau prima=2	0.00	250.0	5.671265
Rast n disc plus tau=1 tau prima=1	0.00	250.0	3.581853
Rast n disc plus tau=1 tau prima=0.5	0.00	250.0	4.377818
Rast n disc plus tau=1 tau prima=0.25	0.00	250.0	6.613558
Rast n disc plus tau=0.5 tau prima=4	0.00	250.0	4.463781
Rast n disc plus tau=2 tau prima=4	0.00	250.0	4.957643
Rast n inter plus tau=0.25 tau prima=0.25	0.00	250.0	24.736600
Rast one disc plus tau=4	0.00	250.0	4.676306
Rast n disc comma tau=4 tau prima=2	0.00	250.0	5.323029
Rast n disc comma tau=0.5 tau prima=0.25	0.00	250.0	4.029583
Rast n disc comma tau=0.25 tau prima=1	0.00	250.0	3.134120
Rast n disc comma tau=0.25 tau prima=0.25	0.00	250.0	4.029584
Rast n inter comma tau=4 tau prima=2	0.00	250.0	3.183869
Rast n disc comma tau=4 tau prima=4	0.00	250.0	4.620339
Rast n inter comma tau=4 tau prima=0.25	0.00	250.0	69.340123
Rast n inter comma tau=2 tau prima=4	0.00	250.0	3.979836
Rast n disc comma tau=0.5 tau prima=1	0.00	250.0	3.830591
Rast n inter comma tau=2 tau prima=2	0.00	250.0	2.885381
Rast n inter comma tau=1 tau prima=4	0.00	250.0	3.532104
Rast n inter comma tau=1 tau prima=2	0.00	250.0	2.785885
Rast n inter comma tau=1 tau prima=0.25	0.00	250.0	38.389069
Rast n inter comma tau=0.5 tau prima=4	0.00	250.0	3.930088
Rast n inter comma tau=0.5 tau prima=2	0.00	250.0	3.034625
Rast n inter comma tau=0.25 tau prima=4	0.00	250.0	4.079331
Rast n inter comma tau=0.25 tau prima=2	0.00	250.0	2.785885
Rast n inter comma tau=2 tau prima=0.25	0.00	250.0	60.468894
Rast n disc comma tau=0.5 tau prima=2	0.00	250.0	4.626559
Rast n inter comma tau=4 tau prima=0.5	0.00	250.0	56.879541
Rast n disc comma tau=1 tau prima=0.25	0.00	250.0	3.830592
Rast n disc comma tau=2 tau prima=4	0.00	250.0	4.186123
Rast n disc comma tau=2 tau prima=1	0.00	250.0	3.731096
Rast n disc comma tau=2 tau prima=0.5	0.00	250.0	3.731096
Rast n disc comma tau=2 tau prima=0.25	0.00	250.0	4.228575
Rast n disc comma tau=4 tau prima=0.25	0.00	250.0	4.278323
Rast n disc comma tau=1 tau prima=4	0.00	250.0	5.176250
Rast n disc comma tau=0.5 tau prima=4	0.00	250.0	4.242211
Rast n disc comma tau=1 tau prima=2	0.00	250.0	4.079332
Rast n disc comma tau=2 tau prima=2	0.00	250.0	3.631600
Rast n disc comma tau=4 tau prima=1	0.00	250.0	4.576810
Rast n disc comma tau=1 tau prima=1	0.00	250.0	3.233616
Rast n disc comma tau=1 tau prima=0.5	0.00	250.0	4.477315
Rast n disc comma tau=4 tau prima=0.5	0.00	250.0	4.676307
Rast one inter plus tau=1	0.05	250.0	4.911691

	SR	AES	MBF
Rast one inter comma tau=4	0.05	250.0	3.681348
Rast one inter plus tau=2	0.05	250.0	3.034625
Rast n inter plus tau=0.25 tau prima=1	0.05	250.0	2.984877
Rast n inter comma tau=0.25 tau prima=1	0.05	250.0	2.139162
Rast one disc plus tau=0.5	0.05	250.0	2.785885
Rast one inter comma tau=0.25	0.05	250.0	28.578686
Rast n disc plus tau=4 tau prima=2	0.05	250.0	3.432608
Rast one disc comma tau=0.25	0.05	250.0	2.039666
Rast n disc plus tau=4 tau prima=0.25	0.05	250.0	4.228575
Rast n inter plus tau=0.5 tau prima=2	0.05	250.0	4.328071
Rast n disc comma tau=0.5 tau prima=0.5	0.05	250.0	3.432609
Rast n inter plus tau=2 tau prima=1	0.05	250.0	3.681348
Rast n disc comma tau=0.25 tau prima=4	0.05	250.0	3.930058
Rast n disc comma tau=0.25 tau prima=2	0.05	250.0	3.681348
Rast n disc plus tau=0.25 tau prima=4	0.05	250.0	4.525109
Rast n inter comma tau=4 tau prima=4	0.05	250.0	3.034625
Rast n inter comma tau=4 tau prima=1	0.05	250.0	2.139162
Rast n disc comma tau=0.25 tau prima=0.5	0.05	250.0	3.532104
Rast n inter comma tau=2 tau prima=1	0.05	250.0	2.387902
Rast one disc plus tau=2	0.05	250.0	3.532104
Rast n inter plus tau=1 tau prima=2	0.05	250.0	4.477315
Rast one inter comma tau=1	0.10	250.0	1.741178
Rast n disc plus tau=0.25 tau prima=0.5	0.10	250.0	4.029583
Rast n disc plus tau=4 tau prima=1	0.10	250.0	3.780844
Rast n inter comma tau=1 tau prima=0.5	0.10	250.0	1.492439
Rast n inter comma tau=0.5 tau prima=1	0.15	250.0	2.288406
Rast n inter comma tau=1 tau prima=1	0.20	250.0	1.989918
Rast n inter comma tau=0.25 tau prima=0.5	0.20	250.0	1.392943
Rast one disc comma tau=0.5	0.25	250.0	2.288404
Rast n inter comma tau=0.5 tau prima=0.5	0.25	250.0	1.343195
Rast n inter comma tau=0.25 tau prima=0.25	0.25	250.0	5.263288
Rast one inter comma tau=0.5	0.30	250.0	1.094465
Rast n inter comma tau=0.5 tau prima=0.25	0.30	250.0	10.345627
Rast n inter comma tau=2 tau prima=0.5	0.40	250.0	0.696471

11 Resultados, análisis y comparación

Los resultados los he ido poniendo en las distintas secciones de esta práctica. También los he ido comentando.

Sin embargo, aquí me interesaría proceder a un análisis del ataque por fuerza bruta anterior para determinar la mejor configuración de la estrategia para la función de Rastrigin.

Podemos concluir, de la tabla anterior:

- El paso n suele ir mejor que el sencillo
- La selección mu_comma_lambda sin ninguna duda van mejor que la mu_plus_lambda
- La recombinación intermedia ofrece mejores resultados que la discreta
- El tau más apropiado parece situarse en algún punto entre 0.25 y 0.5 aunque curiosamente el mejor resultado se obtiene con un tau igual a 2

- Los mejores taus primas parece que están también en algún punto entre 0.25 y 0.5

Y en cualquier caso, y aunque contraviene un poco las observaciones anteriores, la mejor configuracion parece que es paso n, selección mu_comma_lambda, recombinación intermediate, tau=2 y tau prima=0.5.

12 Conclusiones

Después de todo el trabajo que lleva esta práctica llego a la conclusión de que las estrategias evolutivas pueden ser una buena herramiento para búsqueda de óptimos en problemas de naturaleza numérica o que puedan modelarse a esa naturaleza.

La cantidad de distintas parametrizaciones que pueden hacerse en estas estrategias es elevada y dar con un algoritmo útil para cada problema puede conllevar mucho más trabajo de lo que en principio aparenta.

Es muy importante comprender el problema real que se quiere resolver para conducirse por el camino de parametrizar apropiadamente el algoritmo. Por ejemplo, en el caso de la función de Rastrigin ha sido muy complicado comprender por qué la mayoría de las veces el algoritmo se quedaba "atrapado" en un óptimo cercano al origen de coordenadas. Tras dibujar la función en dos dimensiones es más fácil comprender que realmente lo que ocurre en vista de la gran cantidad de óptimos locales que tiene la función y su escasa diferencia en valor.Si se tiene en cuenta que el problema está planteado en 10 dimensiones, puedo comprender que los resultados no siempre hayan sido buenos. En el caso de 6 dimensiones, las pruebas que he hecho siempre encuentran el óptimo global. Y en el caso de 8 dimensiones, la mayoría de las veces lo encuentran.

Tengo la sensación de no haber conseguido exprimir al máximo las posibilidades que dan los parámetros tau y tau prima, y no será porque no he hecho miles de pruebas. Pero en cualquier caso me quedo con la idea de que gobirnar la intensidad de la variabilidad de las sigmas.

En el caso de la función hiperelipsoide rotado, el tener un único óptimo, todas las configuraciones del algoritmo han dado resultados muy buenos. Nuevamente, esto se comprenden muy fácilmente representando la función en el caso de dos dimensiones .

Lo realmente importante, o lo que más importante me ha resultado a mí de esta práctica, es recabar en la necesidad de utilizar métricas de estimación de la calidad de un algoritmo y su parametrización y que estas métricas se basen en la repetición de ejecucuiones idénticas y en el análisis estadístico de los resultados. De este modo, no podria haber hecho comparaciones que realmente valgan para algo sin haber dispuesto de las tres métricas SR, AES y MBF.

Igualmente, dibujar gráficas de la evolución del fitness considero que es de lo más necesario. Me ha sido muy útil acudir al concepto de escala logarítmica (inversa en este caso), para que las gráficas realmente tubieran sentido y fueran interpretables.