# Convergence_Analysis

January 29, 2020

## 1 Pareto-like Sequential Sampling (PSS) Algorithm - Convergence Analysis

### 1.1 ## 2D Schwefel function as an example

Authors: Mahmoud Shaqfa, Katrin Beyer

---

This additional sheet demonstrates the convergence analysis of Schwefel function as a 2D optimization problem ($n = 2$).

$f(x_1, x_2) = 418.9829 \times 2 - x_1 sin(\sqrt{|x_1|}) - x_2 sin(\sqrt{|x_2|})$.

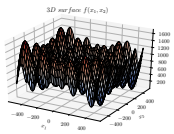
### 1.2 Plot Schwefel function and find the true prominent region

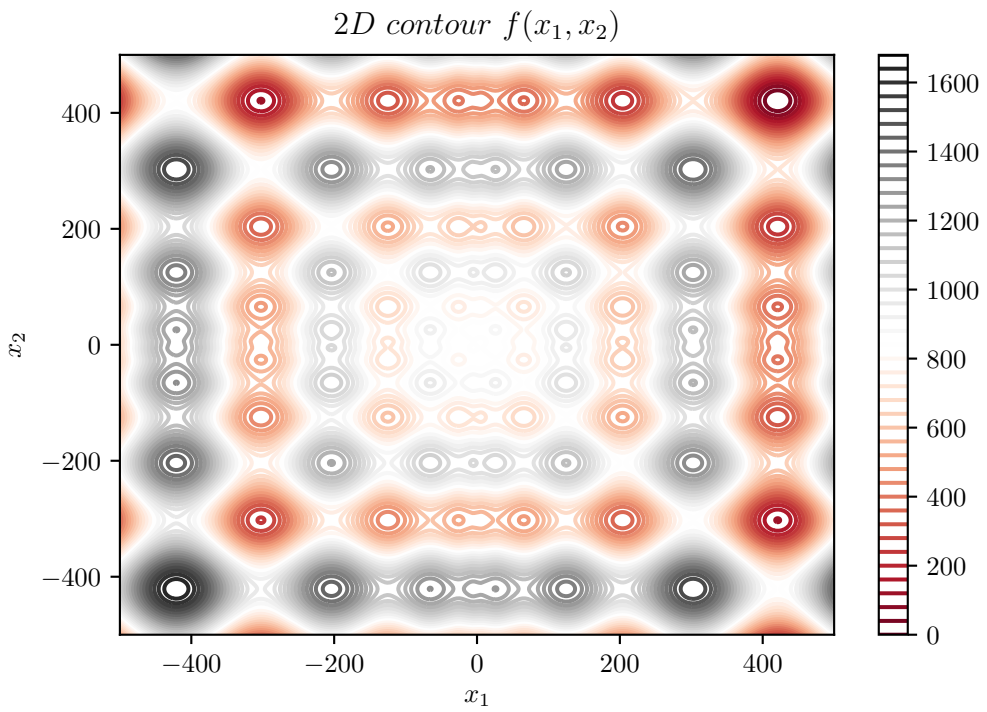
Schwefel function in a multimodal non-convex function with n-dimensions. It is considered as one of the hard standard benchmarks for global optimization algorithms. Normally, this function is defined on the hypercube domain $\in [-500, 500]^n$. The exact answer for this function is when $x_j = 420.9687, \ \forall \, j = 1, \ldots, n$.

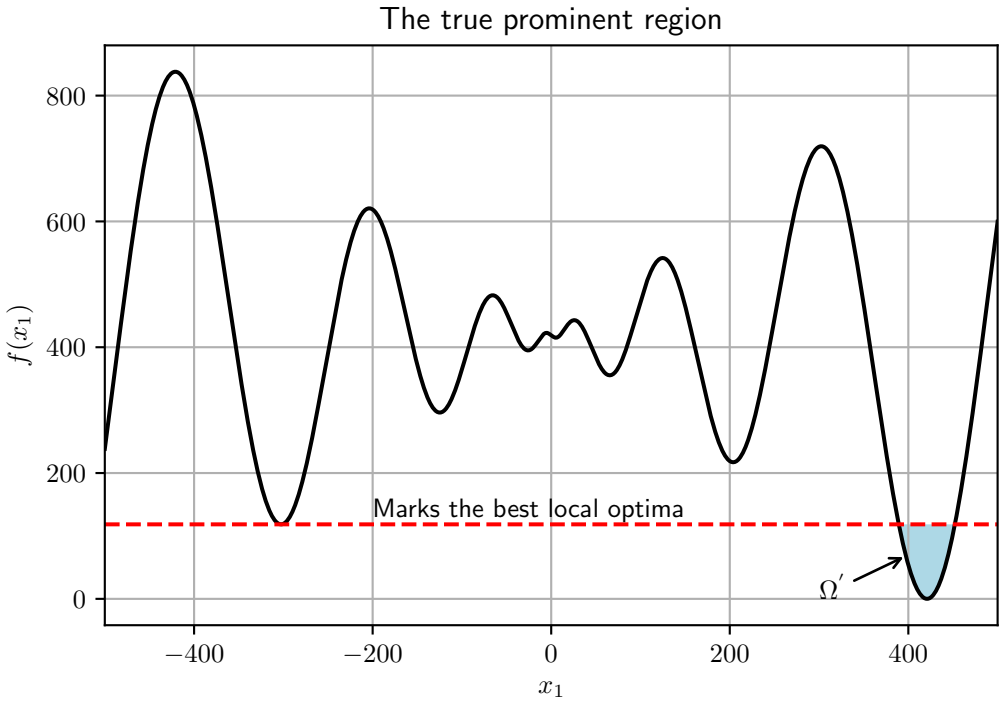
In [7]:
```
# Load Python 3.6 preambles
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from matplotlib import ticker, cm
from matplotlib.ticker import MaxNLocator
plt.rcParams['text.usetex'] = True
from IPython.display import set_matplotlib_formats
set_matplotlib_formats('pdf')

# Define Schwefel function -- 2D only
def schwefel(x, y):
    return 418.9829 * 2. - x * np.sin(np.sqrt(np.abs(x))) -\
        y * np.sin(np.sqrt(np.abs(y)))

# Prepare for plots - initialization and pre-allocation
```

```
step = 1.; # Step size (the plot accuracy)
grid = np.linspace(-500., 500., np.int((500. - -500.)/step))
X, Y = np.meshgrid(grid, grid)
Z = schwefel(X, Y)
y = 418.9829 - grid * np.sin(np.sqrt(np.abs(grid)))
y2 = np.ones([len(y), 1]) * 118.4384 # The line that marks the best local optimum
```

In [8]:
```
# Plot 3D surface
fig = plt.figure(dpi= 200)
ax = plt.axes(projection = '3d')
ax.plot_surface(X, Y, Z,cmap=cm.coolwarm, edgecolor='black',\
                linewidth = 0.3, alpha = 0.7, shade = True)
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.title(r'$3D~surface~f(x_1, x_2)$')
plt.show()

# Plot 2D contour
fig = plt.figure(dpi= 200)
cp = plt.contour(X, Y, Z, 50, cmap='RdGy')
plt.xlabel(r'$x_1$')
plt.ylabel(r'$x_2$')
plt.title(r'$2D~contour~f(x_1, x_2)$')
plt.colorbar()
plt.show()

# Plot the distribution of one x_1 vs the f(x1)
fig = plt.figure(dpi= 200)
plt.xlabel(r'$x_1$')
plt.ylabel(r'$f(x_1)$')
plt.xlim([-500., 500.])
plt.grid()
plt.plot(grid, y, 'k')
plt.plot(grid, y2, 'r--')
plt.fill_between(grid, y, y2[:,0]\
    , facecolor='lightblue', where= y < y2[:,0])
plt.text(-200, 130, 'Marks the best local optima')
plt.title('The true prominent region')
plt.annotate(r"$\Omega^{'}$", xy=(400, 70), xytext=(300, 0),
    arrowprops=dict(arrowstyle="->"))
plt.show()
```




2D contour $f(x_1, x_2)$
400
200
0
−200
−400
$x_2$
−400
−200
0
200
400
$x_1$
1600
1400
1200
1000
800
600
400
200
0


The true prominent region
800
600
400
200
0
$f(x_1)$
Marks the best local optima
$\Omega'$
−400
−200
0
200
400
$x_1$

## 1.3 The probability of finding good solutions at the initial population step

### 1.3.1 The effects of scalability and population size ($\beta$) on the initial population sampling

After we define the location of the best local minimum, we can accordingly define the true prominent domain for the design variable. The true prominent domain in Schwefel is $\Omega^{'} \in [389.33, 452.16]$ per design variable $x_j$.

The probability of getting any solution in the true prominent region is $P(x \in \Omega^{'}) = \frac{n^{'}}{N} = \frac{||\Omega^{'}||}{||\Omega||} = \frac{452.16 - 389.33}{500 - (-500)} = 0.062827$.

The probability of at least one out of $\beta$ solutions will be in the prominent region (event $A_0$): $P(A_0) = 1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}$. This is a significant parameter for the "gbest" topology used with the PSS algorithm.

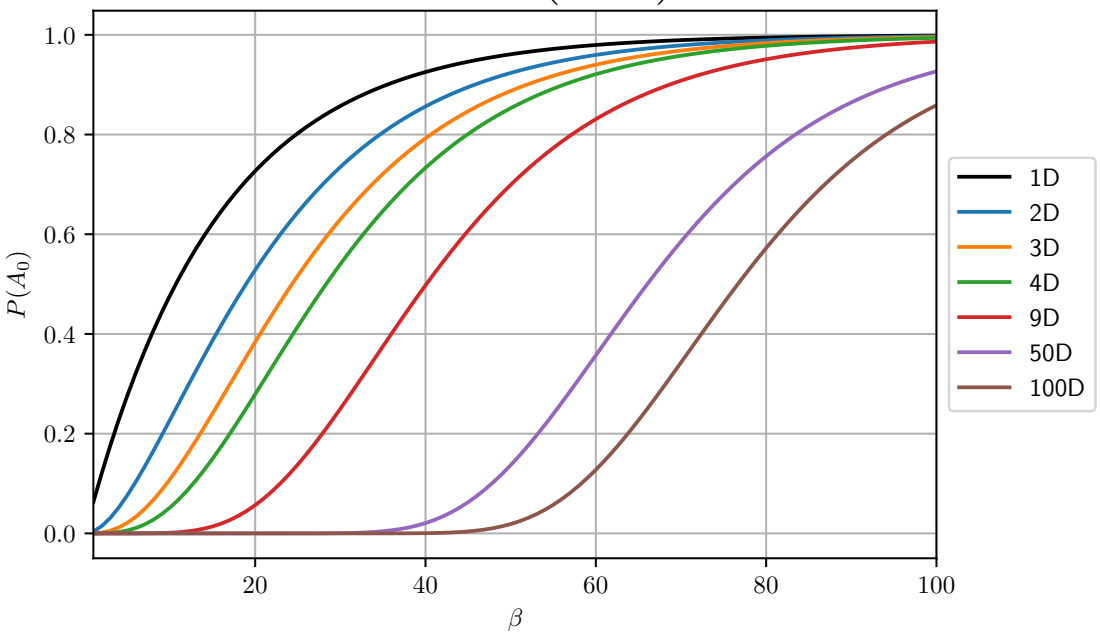
This analysis is important to determine the initial size of the population and the needed computational capacity for the algorithm to run. The following figure explains the effect of the size of the population and the problem vs. the probability of sampling at least one solution in the true prominent domain $\Omega^{'}$. As we can see from the figure, the curve converges after a considerable population size to 1.0. The more the population we use the better the quality of the solutions can be obtained.

Note: If we need to find the probability for $n > 1$ design variables it will be simply by multiplying the probabilities (independent events) of each design variable (Multiplication rule of conditional probabilities).

```
In [9]: population_size_vector = np.linspace(1, 100, 100)

        def probability_vs_size(p, vector):
            return (1. - np.power((1. - p), vector))

        p = 0.062827
        probability_vs_size_vector = probability_vs_size(p, population_size_vector)
        probability_vs_size_vector_2D = np.power(probability_vs_size_vector, 2.)
        probability_vs_size_vector_3D = np.power(probability_vs_size_vector, 3.)
        probability_vs_size_vector_4D = np.power(probability_vs_size_vector, 4.)
        probability_vs_size_vector_9D = np.power(probability_vs_size_vector, 9.)
        probability_vs_size_vector_50D = np.power(probability_vs_size_vector, 50.)
        probability_vs_size_vector_100D = np.power(probability_vs_size_vector, 100.)

        fig, ax = plt.subplots(dpi= 200)
        plt.plot(population_size_vector, probability_vs_size_vector, 'k', label='1D')
        plt.plot(population_size_vector, probability_vs_size_vector_2D, label='2D')
        plt.plot(population_size_vector, probability_vs_size_vector_3D, label='3D')
        plt.plot(population_size_vector, probability_vs_size_vector_4D, label='4D')
        plt.plot(population_size_vector, probability_vs_size_vector_9D, label='9D')
        plt.plot(population_size_vector, probability_vs_size_vector_50D, label='50D')
        plt.plot(population_size_vector, probability_vs_size_vector_100D, label='100D')

        plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
```

```
plt.xlabel(r"$\beta$")
plt.ylabel(r"$P(A_0)$")
ax.set_title(r"$\displaystyle P(A_0) = 1 - \Bigg(1 - \frac{n^{'}}{N}\Bigg)^{\beta}$"\
             , fontsize = 14)
plt.xlim([1, 100])
plt.grid(True)
plt.show()
```

$$P(A_0) = 1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}$$




---

## 1.4 The balance between the diversification and the intensification search steps

### 1.4.1 exploration vs. exploitation

The diversification probability: $P^i\left(x \in \Omega^{'} | r^i > \alpha\right) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right](1 - \alpha)$. This formula is constant with time and gives the algorithm free accessability to the all parts of the landscape.

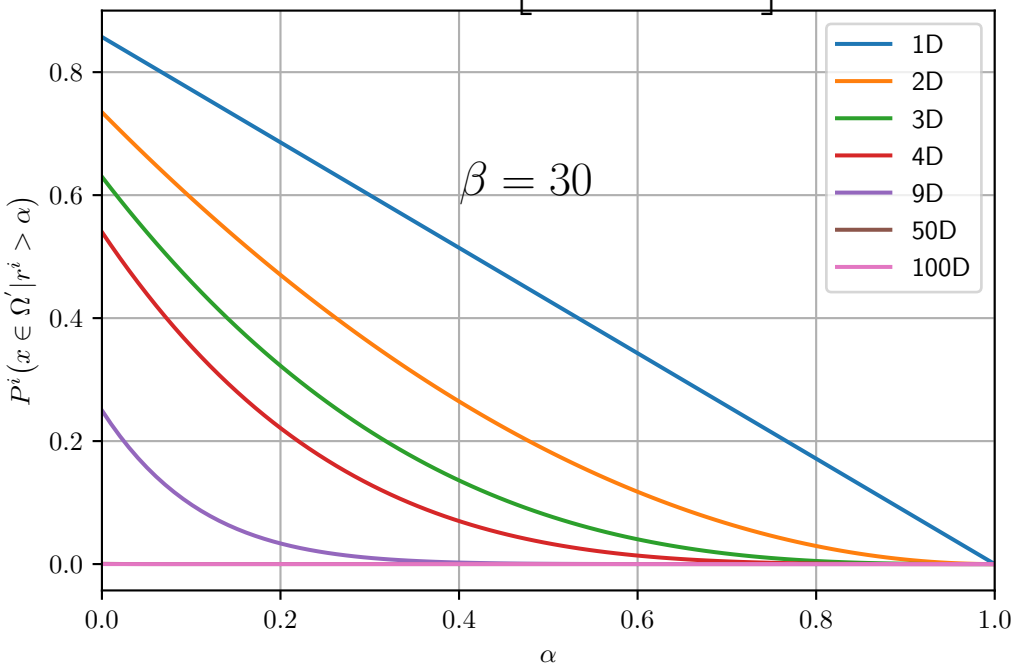
The intensification probability: $P^i\left(x_{best}^{i+1} < x_{best}^{i} | r^i \leq \alpha\right) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right]\alpha$. This formula is time-dependent and depends on the current location of the fictious prminent region(Inside $\Omega^{'}$ only). In this section we only investigate it inside the true prominent domain.

The the size of the population considered here: $\beta = 30$ and 100.

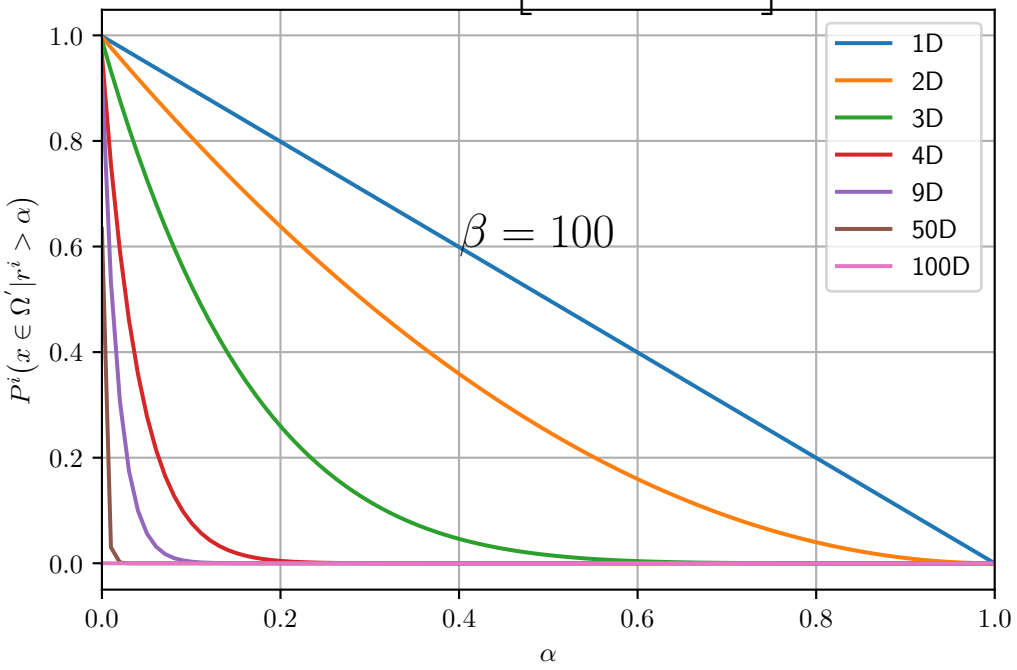
In [10]:
```
# The diversification probability analysis - Beta = 30
alpha_size_vector = np.linspace(0., 1., 100)

def diversification_probability(p, alpha_vector, beta):
    return (1. - np.power((1. - p), beta)) * (1-alpha_vector)

p = 0.062827
beta1 = 30
probability_vs_alpha_div_1D = diversification_probability(p, alpha_size_vector, beta1)
probability_vs_alpha_div_2D = np.power(probability_vs_alpha_div_1D, 2.)
probability_vs_alpha_div_3D = np.power(probability_vs_alpha_div_1D, 3.)
probability_vs_alpha_div_4D = np.power(probability_vs_alpha_div_1D, 4.)
probability_vs_alpha_div_9D = np.power(probability_vs_alpha_div_1D, 9.)
probability_vs_alpha_div_50D = np.power(probability_vs_alpha_div_1D, 50.)
probability_vs_alpha_div_100D = np.power(probability_vs_alpha_div_1D, 100.)

fig, ax = plt.subplots(dpi= 200)
plt.plot(alpha_size_vector, probability_vs_alpha_div_1D, label='1D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_2D, label='2D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_3D, label='3D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_4D, label='4D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_9D, label='9D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_50D, label='50D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_100D, label='100D')

plt.legend()
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)$")
ax.set_title(r"$\displaystyle P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)= \left[1 - 
plt.xlim([0, 1])
plt.grid(True)
plt.text(0.4, 0.6, r"$\beta = 30$", fontsize = 18)
plt.show()

# The diversification probability analysis - Beta = 100
beta2 = 100
probability_vs_alpha_div_1D = diversification_probability(p, alpha_size_vector, beta2)
probability_vs_alpha_div_2D = np.power(probability_vs_alpha_div_1D, 2.)
probability_vs_alpha_div_3D = np.power(probability_vs_alpha_div_2D, 3.)
probability_vs_alpha_div_4D = np.power(probability_vs_alpha_div_3D, 4.)
probability_vs_alpha_div_9D = np.power(probability_vs_alpha_div_3D, 9.)
probability_vs_alpha_div_50D = np.power(probability_vs_alpha_div_3D, 50.)
probability_vs_alpha_div_100D = np.power(probability_vs_alpha_div_9D, 100.)

fig, ax = plt.subplots(dpi= 200)
plt.plot(alpha_size_vector, probability_vs_alpha_div_1D, label='1D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_2D, label='2D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_3D, label='3D')
```

```
plt.plot(alpha_size_vector, probability_vs_alpha_div_4D, label='4D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_9D, label='9D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_50D, label='50D')
plt.plot(alpha_size_vector, probability_vs_alpha_div_100D, label='100D')

plt.legend()
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)$")
ax.set_title(r"$\displaystyle P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)= \left[  1 -
plt.xlim([0, 1])
plt.grid(True)
plt.text(0.4, 0.6, r"$\beta = 100$", fontsize = 18)
plt.show()
```

$$P^i(x \in \Omega^{'}|r^i > \alpha) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right](1 - \alpha).$$



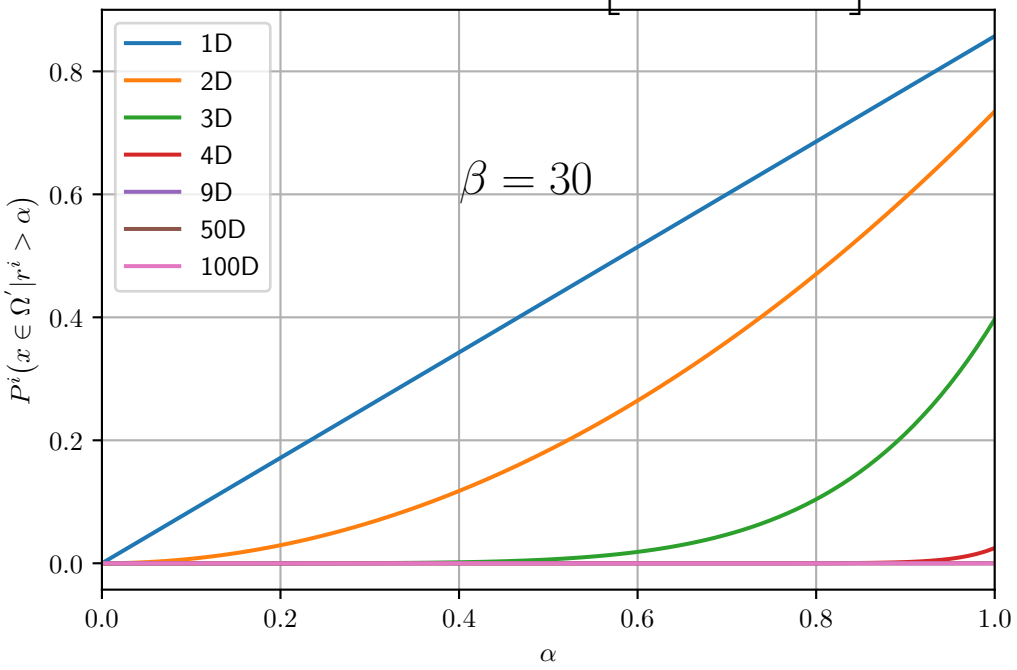
$$P^i(x \in \Omega^{'} | r^i > \alpha) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right] (1 - \alpha).$$



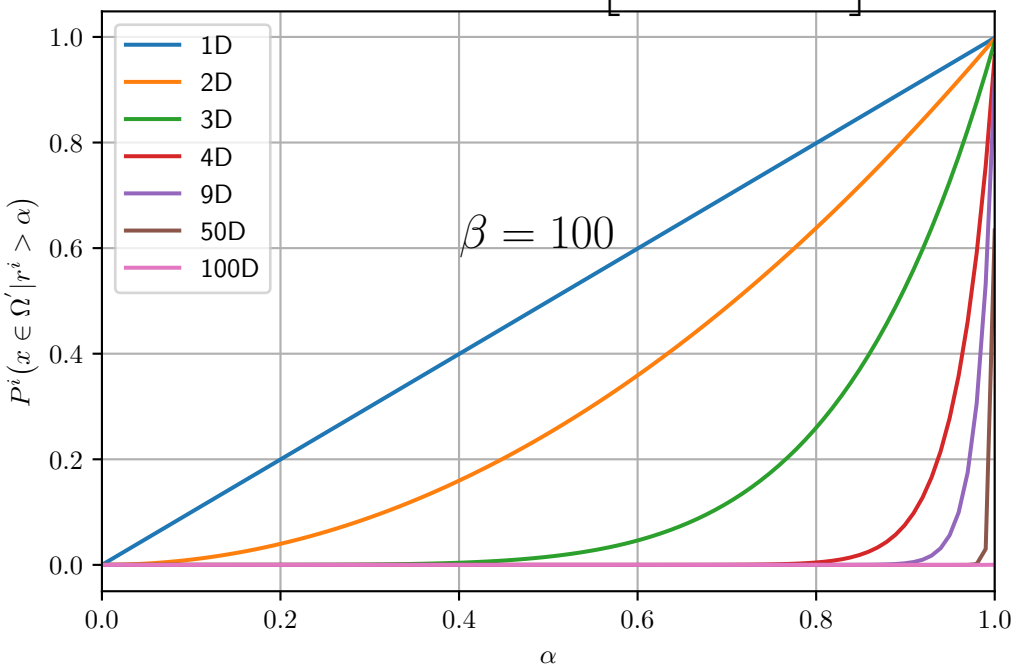

```
In [11]: # The intensification probability analysis - Beta = 30
         alpha_size_vector = np.linspace(0., 1., 100)

         def intensification_probability(p, alpha_vector, beta):
             return (1. - np.power((1. - p), beta)) * (alpha_vector)

         p = 0.062827
         beta1 = 30
         probability_vs_alpha_int_1D = intensification_probability(p, alpha_size_vector, beta1)
         probability_vs_alpha_int_2D = np.power(probability_vs_alpha_int_1D, 2.)
         probability_vs_alpha_int_3D = np.power(probability_vs_alpha_int_2D, 3.)
         probability_vs_alpha_int_4D = np.power(probability_vs_alpha_int_3D, 4.)
         probability_vs_alpha_int_9D = np.power(probability_vs_alpha_int_3D, 9.)
         probability_vs_alpha_int_50D = np.power(probability_vs_alpha_int_9D, 50.)
         probability_vs_alpha_int_100D = np.power(probability_vs_alpha_int_9D, 100.)

         fig, ax = plt.subplots(dpi= 200)
         plt.plot(alpha_size_vector, probability_vs_alpha_int_1D, label='1D')
         plt.plot(alpha_size_vector, probability_vs_alpha_int_2D, label='2D')
         plt.plot(alpha_size_vector, probability_vs_alpha_int_3D, label='3D')
         plt.plot(alpha_size_vector, probability_vs_alpha_int_4D, label='4D')
         plt.plot(alpha_size_vector, probability_vs_alpha_int_9D, label='9D')
```

```
plt.plot(alpha_size_vector, probability_vs_alpha_int_50D, label='50D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_100D, label='100D')

plt.legend()
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)$")
ax.set_title(r"$\displaystyle P^i\big(x^{i+1}_{best} < x^{i}_{best}|r^{i}\leq \alpha\
plt.xlim([0, 1])
plt.grid(True)
plt.text(0.4, 0.6, r"$\beta = 30$", fontsize = 18)
plt.show()

# The intensification probability analysis - Beta = 100
beta2 = 100
probability_vs_alpha_int_1D = intensification_probability(p, alpha_size_vector, beta2)
probability_vs_alpha_int_2D = np.power(probability_vs_alpha_int_1D, 2.)
probability_vs_alpha_int_3D = np.power(probability_vs_alpha_int_2D, 3.)
probability_vs_alpha_int_4D = np.power(probability_vs_alpha_int_3D, 4.)
probability_vs_alpha_int_9D = np.power(probability_vs_alpha_int_3D, 9.)
probability_vs_alpha_int_50D = np.power(probability_vs_alpha_int_3D, 50.)
probability_vs_alpha_int_100D = np.power(probability_vs_alpha_int_9D, 100.)

fig, ax = plt.subplots(dpi= 200)
plt.plot(alpha_size_vector, probability_vs_alpha_int_1D, label='1D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_2D, label='2D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_3D, label='3D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_4D, label='4D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_9D, label='9D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_50D, label='50D')
plt.plot(alpha_size_vector, probability_vs_alpha_int_100D, label='100D')

plt.legend()
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)$")
ax.set_title(r"$\displaystyle P^i\big(x^{i+1}_{best} < x^{i}_{best}|r^{i}\leq \alpha\
plt.xlim([0, 1])
plt.grid(True)
plt.text(0.4, 0.6, r"$\beta = 100$", fontsize = 18)
plt.show()
```

$$P^i(x^{i+1}_{best} < x^i_{best} | r^i \leq \alpha) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right] \alpha.$$




$$P^i(x^{i+1}_{best} < x^i_{best} | r^i \leq \alpha) = \left[1 - \left(1 - \frac{n^{'}}{N}\right)^{\beta}\right] \alpha.$$



In [12]:
```
# Diversification vs. intensification - Beta = 100
fig, ax = plt.subplots(dpi= 200)
plt.plot(alpha_size_vector, probability_vs_alpha_int_1D, label='1D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_2D, label='2D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_3D, label='3D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_4D, label='4D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_9D, label='9D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_50D, label='50D-intensification')
plt.plot(alpha_size_vector, probability_vs_alpha_int_100D, label='100D-intensificatio

plt.plot(alpha_size_vector, probability_vs_alpha_div_1D, label='1D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_2D, label='2D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_3D, label='3D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_4D, label='4D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_9D, label='9D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_50D, label='50D-diversification')
plt.plot(alpha_size_vector, probability_vs_alpha_div_100D, label='100D-diversificatio

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.xlabel(r"$\alpha$")
plt.ylabel(r"$P^i\big(x\in \Omega^{'}|r^{i} > \alpha\big)$")
ax.set_title(r"Diversification vs. intensification", fontsize = 14)
plt.xlim([0, 1])
plt.grid(True)
plt.text(0.4, 0.6, r"$\beta = 100$", fontsize = 18)
plt.show()
```