

**Universidad Tecnológica Nacional**

**Facultad Regional Rosario**

Especialidad: Ing. en Sistemas de Información.

Asignatura: Algoritmos Genéticos

Comisión: 3EK03

Ciclo lectivo: 2025

Fecha: 12/06/2025

**TRABAJO PRÁCTICO N°1:**

**Maximización de una función objetivo**

|  |  |  |
| --- | --- | --- |
| **Alumno** | **Correo** | **Legajo** |
| Mondino, Juan Cruz | juancm.2000@hotmail.com | 51922 |
| Giampietro, Gustavo | gustgiam2001@gmail.com | 50671 |
| Mateo, Alexis | alexisjoelmateo@gmail.com | 51191 |

Contenido

[Enunciado 1](#_heading=h.lybgvo3r0zx)

[Metodología de desarrollo 2](#_heading=h.4hmhg567zcv)

[Forma de trabajo abordada en equipo 3](#_heading=h.r2czlj8nflc2)

[Herramientas de programación 4](#_heading=h.s76c30279g8k)

[Código 4](#_heading=h.ag53b4wup50c)

[Salidas por pantalla 12](#_heading=h.hu2pxhskwth)

[Gráficas 12](#_heading=h.91iprlcf9upm)

[Sin Elitismo 12](#_heading=h.blnrixyedymm)

[Ruleta 12](#_heading=h.xrj3jhulo6p)

[Torneo 15](#_heading=h.ccfkzqk2hbm8)

[Con elitismo 17](#_heading=h.1spm1tclppi2)

[Ruleta 18](#_heading=h.cyxxghhtcuay)

[Torneo 19](#_heading=h.5f65a05r363h)

[Conclusiones 19](#_heading=h.iild6b4tupzu)

# 

# Enunciado

Hacer un programa que utilice un Algoritmo Genético Canónico para buscar un máximo de la función:

en el dominio [0 , 230-1], donde:

Teniendo en cuenta los siguientes datos:

* Probabilidad de Crossover = 0,75.
* Probabilidad de Mutación = 0,05.
* Población Inicial: 10 individuos.
* Ciclos del programa: 20.
* Métodos de selección: Ruleta, Torneo y Elitismo.
* Método de Crossover: 1 Punto.
* Método de Mutación: invertida.
* Cantidad de individuos seleccionados por elitismo: 2.

El programa debe mostrar los siguientes datos para los diferentes métodos de selección:

1. El cromosoma correspondiente al valor máximo, el valor máximo, mínimo y promedio obtenido de cada población.
2. La impresión de las tablas de mínimos, promedios y máximos para 20, 100 y 200 corridas (para elitismo solamente los valores de 100 iteraciones del algoritmo).
3. Las gráficas de los valores Máximos, Mínimos y Promedios de la función objetivo por cada generación luego de correr el algoritmo genético 20, 100 y 200 iteraciones (para elitismo solamente las gráficas de 100 iteraciones del algoritmo).
4. Realizar comparaciones de las salidas corriendo el mismo programa en distintos ciclos de corridas y además realizar todos los cambios que considere oportunos en los parámetros de entrada de manera de enriquecer sus conclusiones.

# Metodología de desarrollo

El programa consiste en ejecutarlo con la siguiente línea:

*Python TP1\_AG\_G2.py -c N -s M -e B*

Donde:

* N es un número entero que representa la cantidad de ciclos, o generaciones, para las que uno quiere que se ejecute el programa.
* M es uno de entre dos valores, la letra “r” o la “t”. De esta forma se representa si se quiere efectuar el método de ruleta o el método de torneo respectivamente.
* Por último, B puede ser un valor entre 1 o 0, siendo 1 el que indique que se usará elitismo y siendo 0 para el caso contrario.

Con esta información, inicializa las variables del algoritmo, como la probabilidad de crossover, la probabilidad de mutación, la cantidad de individuos, la cantidad de genes por individuo, el coeficiente de normalización (*coef*) y, en caso de usarse torneo, cuántos competidores participan por ronda.

Luego se genera la población inicial: una lista de cromosomas binarios aleatorios. Cada cromosoma es una lista de 0s y 1s. A esta población se le calcula el fitness de cada cromosoma evaluando su valor decimal con la función objetivo enunciada anteriormente, y finalmente calcula su fitness relativo dentro de la población, es decir, su proporción respecto de la suma total de objetivos. Paralelamente, se calcula el máximo, mínimo, promedio y mejor individuo.

A partir de aquí, se inicia el ciclo evolutivo. El programa decide si usará elitismo o no. En caso afirmativo, se seleccionan los mejores individuos según fitness y se los aparta para preservarlos (los llamaremos elitistas). El resto de la nueva generación se obtiene por uno de los dos métodos de selección: ruleta, que asigna probabilidad proporcional al fitness, o torneo, donde se eligen aleatoriamente grupos de competidores y se selecciona el mejor de cada grupo.

Una vez seleccionados los individuos para la siguiente generación (sin contar a los elitistas), el algoritmo procede a evaluar la probabilidad de crossover. Si se termina dando, toma pares de individuos y aplica el operador de cruce en un punto, generando dos hijos. Luego, con otra probabilidad, analiza si debe aplicar una mutación a cada uno de los hijos, alterando aleatoriamente un gen de su cromosoma. Cuando se termina este proceso, si hay elitismo, los mejores individuos originales se reincorporan a la población sin modificaciones.

Con la nueva población formada, se vuelve a calcular el fitness y las estadísticas. Los valores máximos, mínimos, promedios y el mejor cromosoma de esa generación se almacenan para su posterior análisis. Este proceso se repite durante la cantidad de ciclos especificada por el usuario.

Una vez finalizados todos los ciclos, se generan los gráficos representando la evolución del valor máximo, mínimo y promedio de fitness a lo largo de las generaciones. Por último, se crea un archivo Excel, que contiene los datos obtenidos ciclo a ciclo, incluyendo el mejor cromosoma en formato binario y decimal.

El objetivo del programa es simular un algoritmo genético que permita analizar y comparar el comportamiento evolutivo de distintas estrategias de selección (ruleta y torneo), con y sin elitismo, observando cómo afectan a la convergencia de la población hacia la solución óptima (el mayor número posible dentro del dominio [0; 230-1] que sea un máximo de la función objetivo), a través de la observación de las estadísticas en los gráficos.

# Forma de trabajo abordada en equipo

Para realizar las corridas de la simulación nuestro grupo eligió programar en Python el algoritmo, para ello fue utilizado el editor de código fuente Visual Studio Code.

Como cada uno de nosotros trabajó desde su propia computadora fueron utilizadas herramientas como Google Drive, en donde fueron creadas carpetas y documentos compartidos para que cada uno pueda editar libremente, y GitHub, para poder codear sin necesidad de ir pasando el documento entre nosotros, además de aprovechar la seguridad que aporta este sistema de control de versiones online.

# Herramientas de programación

Como ya fue mencionado se utilizó Python para programar los algoritmos. Pero dentro de este lenguaje fueron utilizadas diferentes librerías con el fin de implementar las siguientes funciones:

* OS: Se utiliza para realizar operaciones sobre el sistema de archivos, como verificar si existe un archivo con un nombre determinado y eliminarlo en caso de ser necesario. También permite limpiar la pantalla del terminal, dependiendo del sistema operativo (Windows, Linux, etc.).
* RANDOM: Su utilización surge por la necesidad de tener un generador de números aleatorios confiable. Además, uno puede limitar el dominio de posibles números aleatorios o especificar si es entero o real.
* SYS: Se emplea para manejar argumentos desde la línea de comandos, lo que permite ejecutar el programa con distintos parámetros (por ejemplo, cantidad de ciclos, tipo de selección, uso o no de elitismo) sin modificar el código fuente.
* MATPLOTLIB.PYPLOT: Esta librería permite generar gráficos durante o después de la ejecución del programa. Se usa para visualizar la evolución de las métricas del algoritmo (como el fitness máximo, mínimo y promedio), lo cual facilita el análisis y comprensión de los resultados.
* PANDAS: Es una herramienta poderosa para la manipulación de datos en forma de tablas. Se utiliza para construir DataFrames y exportarlos a archivos Excel (.xlsx), lo que permite guardar los resultados del algoritmo en un formato claro y accesible.
* OPENPYXL: Es el motor que PANDAS usa por defecto para escribir archivos Excel. Es necesario tenerla instalada para que la función to\_excel() de PANDAS funcione correctamente con archivos .xlsx.

# Código

En este apartado se explicará parte por parte el código Python utilizado para correr las simulaciones:

En la parte superior del código se especifican las librerías que se utilizarán y deben ser instaladas para correr el programa.

import random  
import os  
import sys  
import matplotlib.pyplot as plt  
import pandas as pd  
*# Utiliza openpyxl tambien*

Luego, se definieron dos funciones. La primera es utilizada para obtener un número aleatorio entre 0 y 1 de una manera más sencilla.

def aleatorio():  
    return random.randint(0, 1)

def completoCromosoma(cantidad):  
   cromosoma = [aleatorio() for \_ in range(cantidad)]  
   return cromosoma

La segunda, se la llama cuando se requiere obtener un cromosoma completo, uno solamente debe pasarle la cantidad de genes que debe tener dicho cromosoma y la función creará el cromosoma con la longitud especificada completo por ceros y unos aleatoriamente. Aquí se busca obtener una población de una cantidad de individuos o cromosomas especificada, ésta es la función que llama a la explicada anteriormente para que complete los genes de cada cromosoma.

def generarPoblacion(cantidadCromosomas, cantidadGenes):  
    poblacion = []  
    for i in range(cantidadCromosomas):  
        cromosoma = completoCromosoma(cantidadGenes)  
        poblacion.append(cromosoma)  
    return poblacion

binarioADecimal es una función que utilizamos para convertir a un cromosoma, que se puede tomar como un número binario, a su correspondiente número decimal.

def binarioADecimal(cromosoma):  
 decimal = 0  
 exponente=0  
 for i in range(len(cromosoma)-1,-1,-1):  
 if cromosoma[i] == 1:  
 decimal = decimal + pow(2,exponente)  
 exponente += 1   
 return decimal

funcionObjetivo se llama para evaluar qué tan buena es una solución candidata. Toma un valor (o conjunto de valores) como parámetro y devuelve una medida numérica que representa su calidad o desempeño según el problema que se busca resolver.

def funcionObjetivo(x):

coef = (2 \*\* 30) - 1

return (x / coef) \*\* 2

En la función crossover, se reciben dos parámetros que son los padres por combinar para obtener dos nuevos individuos de la población. Entonces, se elige un punto aleatorio dentro de cada individuo para “cortarlo” y unirlo a la otra parte del segundo individuo, obteniendo así los dos nuevos cromosomas de la nueva generación.

def crossover1Punto(padre, madre):  
    puntoCorte = random.randint(1,len(padre)-1)  
    h1 = padre[:puntoCorte] + madre[puntoCorte:]  
    h2 = madre[:puntoCorte] + padre[puntoCorte:]  
    return h1, h2

Esta es la función encargada de devolver las métricas de rendimiento de cada generación, sirve para monitorear el progreso.

def calculadorEstadisticos(poblacion):     
 objetivos = [funcionObjetivo(binarioADecimal(ind)) for ind in poblacion]  
    max\_objetivos = max(objetivos)  
    min\_objetivos = min(objetivos)  
    mejor\_cromosoma = poblacion[objetivos.index(max\_objetivos)]  
 avg\_objetivos = round((sum(objetivos)/len(objetivos)),4)  
    return [max\_objetivos,min\_objetivos, avg\_objetivos, mejor\_cromosoma]

La medida devuelta por la funcionObjetivo es luego utilizada acá, en la función calculadorFitness, donde se almacena el fitness de cada individuo. Este fitness es lo que nos permite identificar que un número (individuo) sea mejor que otro.

def calculadorFitness(poblacion):  
 fitness = []  
 for individuo in poblacion:   
 decimal = binarioADecimal(individuo)  
 obj = funcionObjetivo(decimal)  
 fitness.append(obj)  
 return fitness

La función operadorMutacion se utiliza para aplicar variaciones aleatorias a los individuos de la población. Su objetivo es mantener la evolución genética como lo es en la realidad (al menos lo más cercano posible) modificando ligeramente los cromosomas dependiendo de una probabilidad establecida. Esta implementa un tipo de mutación por inversión, donde se selecciona un segmento aleatorio del cromosoma y se invierte el orden de los genes dentro de ese segmento.

# Mutacion  
def mutacionInvertida(poblacion, probabilidad):  
 for i in range(len(poblacion)):  
 if random.random() < probabilidad:  
 individuo = poblacion[i]  
 pos1 = random.randint(0, len(individuo) - 1)  
 pos2 = random.randint(0, len(individuo) - 1)  
 # Ordenar para que pos1 < pos2  
 if pos1 > pos2:  
 pos1, pos2 = pos2, pos1  
 segmento\_invertido = individuo[pos1:pos2+1][::-1]  
 poblacion[i] = individuo[:pos1] + segmento\_invertido + individuo[pos2+1:]  
 return poblacion

La función seleccionRuleta implementa un método de selección por ruleta (o ruleta proporcional) dentro de un algoritmo genético. Dada una población y una lista de valores de aptitud (fitnessValores), selecciona una cantidad específica de individuos (cantidad) de forma probabilística, favoreciendo a aquellos con mayor fitness.

Primero, calcula la probabilidad relativa de cada individuo dividiendo su fitness entre el total. Luego genera una lista de probabilidades acumuladas, lo que permite simular una ruleta: se generan números aleatorios entre 0 y 1, y se selecciona el primer individuo cuya probabilidad acumulada sea mayor o igual al número generado. Esto garantiza que los individuos más aptos tengan mayor probabilidad de ser seleccionados, pero sin excluir totalmente a los menos aptos.

# Ruleta  
def seleccionRuleta(poblacion, fitnessValores, cantidad):  
 total\_fitness = sum(fitnessValores)  
 probabilidades = [f / total\_fitness for f in fitnessValores]  
 # Generar acumuladas  
 acumuladas = []  
 acum = 0  
 for p in probabilidades:  
 acum += p  
 acumuladas.append(acum)  
 seleccionados = []  
 for \_ in range(cantidad):  
 r = random.random()  
 for i, limite in enumerate(acumuladas):  
 if r <= limite:  
 seleccionados.append(poblacion[i])  
 break  
 return seleccionados

Luego se realiza la selección por torneo utilizando seleccionTorneo, donde se eligen varios competidores al azar y gana el que tiene mayor fitness. Se repite hasta completar la cantidad de individuos requeridos. Es un método de selección que introduce competencia directa entre cromosomas.

# Torneo  
def seleccionTorneo(poblacion, fitnessValores, cantidadIndividuos, cantidadCompetidores):  
 ganadores = []  
 for j in range(cantidadIndividuos):  
 competidores = []  
 fitness\_competidores = []  
 for i in range(cantidadCompetidores):  
 c=random.randint(0,len(poblacion)-1)  
 competidores.append(poblacion[c])  
 fitness\_competidores.append(fitnessValores[c])  
 ganador = competidores[fitness\_competidores.index(max(fitness\_competidores))]  
 ganadores.append(ganador)   
 return ganadores

La función ciclos\_con\_elitismo ejecuta con elitismo durante una cantidad determinada de generaciones. En cada ciclo, se parte de una población de individuos, se calcula su aptitud (fitness), y se seleccionan los mejores según este criterio. Estos individuos, denominados elitistas, son apartados temporalmente para asegurarse de que pasen directamente a la siguiente generación sin sufrir modificaciones.

Luego, el resto de la población es reemplazada por nuevos individuos generados mediante un proceso de selección (ruleta o torneo, según se indique), seguido de operadores genéticos: crossover entre pares de individuos y mutación invertida. Finalmente, se reincorporan los individuos elitistas a la población y se actualizan las métricas de desempeño (máximo, mínimo, promedio y mejor individuo), que se almacenan en listas para ser devueltas al finalizar todas las generaciones.

# Elitismo  
def ciclos\_con\_elitismo(ciclos, prob\_crossover, prob\_mutacion, cant\_individuos, cant\_genes, metodo\_seleccion, cantidadElitismo, cantidadCompetidores=None):  
 maximos=[]

minimos=[]

promedios=[]

mejores=[]

pob = generarPoblacion(cant\_individuos,cant\_genes) #Poblacion inicial random

fit = calculadorFitness(pob)

rta = calculadorEstadisticos(pob)

for j in range (ciclos):

#De la poblacion me quedo con los de elite.

elitistas = []

indices\_elitistas = []

fit\_ordenados = sorted(fit, reverse=True)

for i in range(cantidadElitismo):

indice = fit.index(fit\_ordenados[i])

elitistas.append(pob[indice])

indices\_elitistas.append(indice)

fit[indice] = -1

pob[indice] = -1

indice = 0

borrados = 0

while borrados < 2:

if fit[indice] == -1:

fit.pop(indice)

pob.pop(indice)

indice = 0

borrados = borrados + 1

else: indice = indice + 1

#elitistas contiene los i mejores individuos de pob

if metodo\_seleccion == 'r':

pob = seleccionRuleta(pob,fit,cant\_individuos - cantidadElitismo) #selecciono una nueva poblacion con los individuos faltantes.

else:

pob = seleccionTorneo(pob, fit, cant\_individuos - cantidadElitismo, cantidadCompetidores)

#REALIZO CROSSOVER Y MUTACION EN LA POBLACION

for i in range (0,len(pob),2):

padre = pob[i]

madre = pob[i+1]

if random.random() < prob\_crossover :

hijo1, hijo2 = crossover1Punto(padre,madre)

pob[i], pob[i+1] = hijo1, hijo2

pob = mutacionInvertida(pob, prob\_mutacion)

pob = pob + elitistas

fit = calculadorFitness(pob)

rta = calculadorEstadisticos(pob)

#GUARDAR VALORES NECESARIOS PARA LA GRAFICA

maximos.append(rta[0])

minimos.append(rta[1])

promedios.append(rta[2])

mejores.append(rta[3])

return maximos, minimos, promedios, mejores

La función ciclos\_sin\_elitismo ejecuta una corrida sin elitismo a lo largo de una cantidad especificada de generaciones. A diferencia del enfoque con elitismo, en este caso ningún individuo es preservado automáticamente, lo que implica que incluso los mejores pueden ser descartados en el proceso evolutivo.

En cada ciclo, se aplica un método de selección (ruleta o torneo) para conformar la nueva población completa. Luego, se realiza el crossover entre pares de individuos con una probabilidad determinada, seguido de la mutación inversa aplicada de forma probabilística a cada individuo. Tras estas operaciones, se recalcula el fitness de la población y se registran las estadísticas relevantes (máximo, mínimo, promedio y mejor individuo). Esta información se guarda en listas para ser devuelta al finalizar todas las generaciones.

# Sin elitismo

def ciclos\_sin\_elitismo(ciclos, prob\_crossover, prob\_mutacion, cantidadIndividuos, cant\_genes, metodo\_seleccion, cantidadCompetidores=None):

maximos=[]

minimos=[]

promedios=[]

mejores=[]

pob = generarPoblacion(cantidadIndividuos,cant\_genes)

fit = calculadorFitness(pob)

rta = calculadorEstadisticos(pob)

maximos.append(rta[0])

minimos.append(rta[1])

promedios.append(rta[2])

mejores.append(rta[3])

for j in range (ciclos):

if metodo\_seleccion == 'r':

pob = seleccionRuleta(pob,fit, cantidadIndividuos)

else:

pob = seleccionTorneo(pob, fit, cantidadIndividuos, cantidadCompetidores)

for i in range (0,len(pob),2):

padre = pob[i]

madre = pob[i+1]

if random.random() < prob\_crossover :

hijo1, hijo2 = crossover1Punto(padre,madre)

pob[i], pob[i+1] = hijo1, hijo2

pob = mutacionInvertida(pob, prob\_mutacion)

fit = calculadorFitness(pob)

rta = calculadorEstadisticos(pob)

#GUARDAR VALORES NECESARIOS PARA LA GRAFICA

maximos.append(rta[0])

minimos.append(rta[1])

promedios.append(rta[2])

mejores.append(rta[3])

return maximos, minimos, promedios, mejores

La función generar\_grafico se encarga de visualizar la evolución del rendimiento del algoritmo genético a lo largo de las generaciones. Toma como entrada listas con los valores máximos, mínimos y promedios de aptitud por generación, así como el mejor individuo, un título y la cantidad de ciclos. Utiliza la biblioteca matplotlib para generar un gráfico de líneas con esos datos, donde se representa cómo cambian dichas métricas en el tiempo. El gráfico se guarda como una imagen .png con un nombre derivado del título proporcionado y también se muestra en pantalla.

Por otro lado, la función verificar\_maximo realiza una verificación simple sobre una lista de datos, evaluando si los valores son monótonamente crecientes. Recorre la lista y, si encuentra un valor menor que el anterior, imprime un mensaje indicando el índice y el valor problemático. En caso contrario, confirma que todos los valores son mayores o iguales a sus predecesores. Esta función es útil para corroborar el buen funcionamiento del método elitista para tantas generaciones.

# GRAFICOS

def generar\_grafico(maximos, minimos, promedios, mejores, titulo, ciclo):  
 x = list(range(len(maximos)))

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(x, maximos, label = 'Máximos', marker='o', linestyle='-', color='b', linewidth=1, markersize=2)

ax.plot(x, minimos, label = 'Mínimos', marker='o', linestyle='-', color='g', linewidth=1, markersize=2)

ax.plot(x, promedios, label = 'Promedios', marker='o', linestyle='-', color='r', linewidth=1, markersize=2)

ax.set\_title('Máximos, Mínimos y Promedios')

ax.set\_xlabel('CORRIDA',fontsize=12)

ax.set\_ylabel('APTITUD',fontsize=12)

ax.set\_ylim(0, 1.2)

ax.set\_xlim(0, ciclo + 2)

ax.grid(True)

ax.legend(fontsize = 10)

fig.suptitle(titulo, fontsize=15)

plt.tight\_layout(rect = [0, 0, 1, 0.95])

plt.savefig(titulo.replace(" ", "\_") + '.png')

plt.show()

def verificar\_maximo(datos):

for i in range(1, len(datos)):

if datos[i] < datos[i - 1]:

print(f"Dato menor encontrado en índice {i}: {datos[i]} < {datos[i - 1]}")

break

else:

print("Todos los datos son mayores o iguales a sus antecesores.")

La función crear\_tabla genera un archivo Excel que registra la evolución de los valores clave del algoritmo genético a lo largo de las generaciones. Recibe como entrada las listas de máximos, mínimos, promedios y mejores individuos por generación, así como el método de selección utilizado y si se aplicó elitismo. A partir de estos datos, crea una tabla en la que cada fila representa una generación e incluye el valor máximo, mínimo, promedio, la representación decimal del mejor individuo y su cadena binaria.

# TABLAS EXCEL  
def crear\_tabla(maximos, minimos, promedios, mejores, metodo\_seleccion, elitismo\_Bool):  
 cadenas = [''.join(str(num) for num in cromosoma) for cromosoma in mejores]  
 decimales = [str(binarioADecimal(cromosoma)) for cromosoma in mejores]  
 nombreMetodo = ''  
 nombreElitismo = ''  
 nombreCantidadCiclos = str(len(maximos)-1)  
 if metodo\_seleccion == 'r':  
 nombreMetodo = '\_Ruleta'  
 else:  
 nombreMetodo = '\_Torneo'  
 if elitismo\_Bool == 1:  
 nombreElitismo = '\_Elitismo'  
 df\_nuevo = pd.DataFrame({  
 'Corrida': range(len(maximos)),  
 'Max': maximos,  
 'Min': minimos,  
 'AVG': promedios,  
 'Decimal': decimales,  
 'Mejor Cromosoma': cadenas,  
 })  
 archivo\_excel = 'VALORES\_' + nombreCantidadCiclos + 'Ciclos' + nombreMetodo + nombreElitismo + '.xlsx'  
 if os.path.exists(archivo\_excel):  
 os.remove(archivo\_excel)  
 df\_nuevo.to\_excel(archivo\_excel, index=False)  
 else:  
 df\_nuevo.to\_excel(archivo\_excel, index=False)

Por último, el bloque principal del programa define los parámetros generales del algoritmo genético y gestiona su ejecución en función de los argumentos que se le pasen por línea de comandos. Recibe tres argumentos clave: la cantidad de ciclos (-c), el tipo de selección (-s, que puede ser ruleta 'r' o torneo 't'), y si se aplica o no elitismo (-e, 1 para sí, 0 para no).

Según si se aplica elitismo o no, se llama a la función correspondiente que ejecuta el algoritmo genético bajo esa configuración. Luego, se genera un gráfico de la evolución de las métricas y se crea una tabla Excel con los resultados por generación. Finalmente, se llama a verificar\_maximo.

# PROGRAMA PRINCIPAL

probCrossover = 0.75

probMutacion = 0.05

cantidadIndividuos = 10

cantidadElitismo = 2

cantidadCompetidores = int(cantidadIndividuos \* 0.4)

cantidadGenes = 30

coef = (2 \*\* cantidadGenes) - 1

maximosPorCiclo = []

minimosPorCiclo = []

promediosPorCiclo = []

if len(sys.argv) != 7 or sys.argv[1] != "-c" or sys.argv[3] != "-s" or sys.argv[5] != "-e":

print("Uso: python TP1\_AG\_G2.py -c <ciclos> -s <seleccion: r-ruleta t-torneo> -e <elitismo: 1-si 0-no>")

sys.exit(1)

if int(sys.argv[2]) < 0 or (int(sys.argv[6]) != 0 and int(sys.argv[6]) != 1) or (sys.argv[4] != "r" and sys.argv[4] != "t"):

print("Error: python TP1\_AG\_G2.py -c <ciclos> -s <seleccion: r-ruleta t-torneo> -e <elitismo: 1-si 0-no>")

sys.exit(1)

ciclosPrograma = int(sys.argv[2])

if int(sys.argv[6]) == 1:

maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores = ciclos\_con\_elitismo(ciclosPrograma,probCrossover, probMutacion, cantidadIndividuos, cantidadGenes, metodo\_seleccion=sys.argv[4], cantidadElitismo=cantidadElitismo, cantidadCompetidores=cantidadCompetidores)

if sys.argv[4] == 'r':

titulo = 'Seleccion RULETA ELITISTA - de '+ str(ciclosPrograma) + ' ciclos'

else:

titulo = 'Seleccion TORNEO ELITISTA - de '+ str(ciclosPrograma) + ' ciclos'

generar\_grafico(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, titulo, ciclosPrograma)

crear\_tabla(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, sys.argv[4], int(sys.argv[6]))

else:

maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores = ciclos\_sin\_elitismo(ciclosPrograma,probCrossover, probMutacion, cantidadIndividuos, cantidadGenes, sys.argv[4], cantidadCompetidores=cantidadCompetidores)

if sys.argv[4] == 'r':

titulo = 'Seleccion RULETA - de '+ str(ciclosPrograma) + ' ciclos'

else:

titulo = 'Seleccion TORNEO - de '+ str(ciclosPrograma) + ' ciclos'

generar\_grafico(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, titulo, ciclosPrograma)

crear\_tabla(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, sys.argv[4], int(sys.argv[6]))

verificar\_maximo(maximosPorCiclo)

# 

# Salidas por pantalla

Los datos obtenidos en cada generación fueron almacenados en archivos Excel. Como algunas de las corridas se realizaron para gran cantidad de generaciones, decidimos subir estos archivos a una carpeta de drive y compartir el link. Esto debido a que examinar directamente esa cantidad de valores numéricos podría ser tedioso y no le aportaría tanto a la conclusión, así como lo hacen los gráficos de la siguiente sección.

Link:

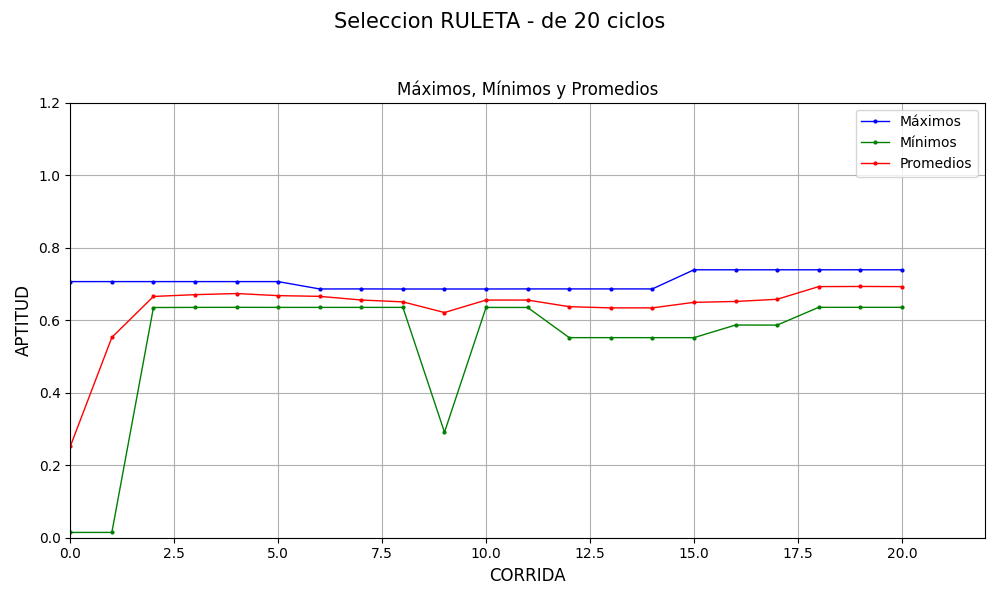
<https://drive.google.com/drive/folders/1Th12fWT9TuVWHyZnt1ePVbe9SKPeT2lQ?usp=sharing>

# Gráficas

## Sin Elitismo

### Ruleta

Para 20 corridas:

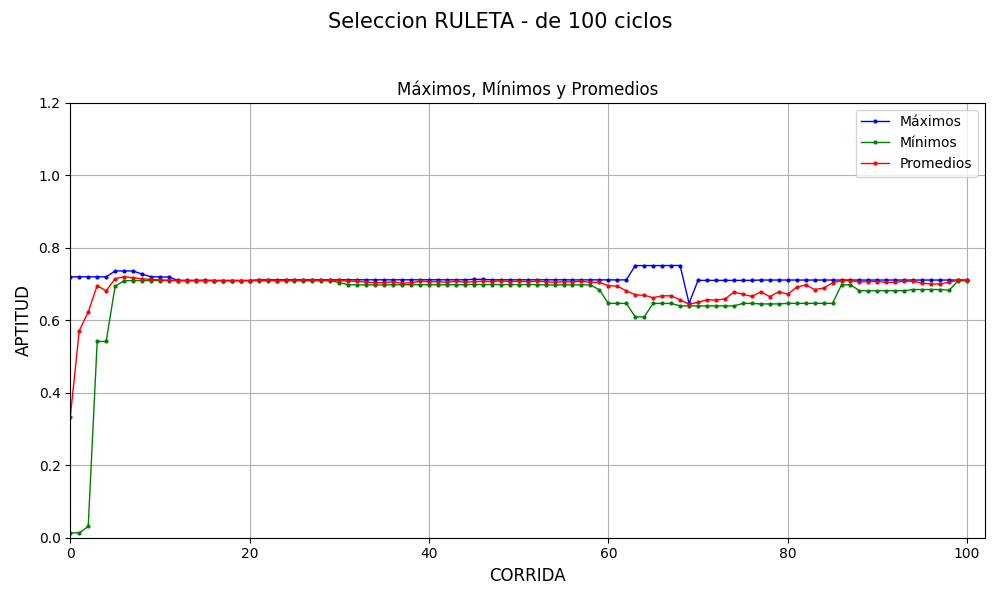


Como se puede observar en la imagen, es difícil que varíe la curva de los máximos a no ser que sufran una mutación, cosa que no sucede en este caso. Al tener más probabilidades de salir sorteados en la ruleta, se mantienen, aunque pasen por el crossover. Esto se da debido a que la parte de mayor peso del número binario es la que indica que tan grande el número es, y la parte de menor peso no modifica tanto el número. Otro suceso que es importante remarcar es que, como se ve en la sexta generación, disminuyó el valor del máximo en comparación a la generación anterior, esto remarca la posibilidad de perder un individuo de gran aptitud por el solo hecho de no haber salido sorteado.

En la curva de los mínimos se puede observar una gran variabilidad. El pico bajo de la novena generación se puede interpretar como el resultante de una mutación.

Por último, en la curva de promedios se observa una tendencia a seguir la forma de la gráfica de mínimos, ya que la de máximos no se mueve tanto.

Para 100 corridas:



Como se puede observar en la imagen, durante las primeras generaciones hay una mejora rápida tanto en los valores máximos como en los promedios, lo cual indica que el algoritmo encuentra buenas soluciones desde el inicio. Sin embargo, al no emplearse elitismo, los individuos más aptos no están garantizados para pasar a la siguiente generación, lo que se refleja claramente alrededor de la décima generación, donde se produce una disminución en el valor máximo respecto a la generación anterior. Esto, una vez más, es una evidencia directa del riesgo de perder soluciones óptimas por el simple hecho de no haber sido seleccionadas en la ruleta.

En cuanto a la curva de los mínimos, se observa una mayor variabilidad a lo largo de las generaciones. Hay caídas pronunciadas (por ejemplo, entre las generaciones 60 y 70) que podrían estar asociadas tanto a la pérdida de presión selectiva como a la introducción de individuos menos aptos por mutación o recombinación desfavorable. Esta inestabilidad en los mínimos también impacta en los promedios, que tienden a seguir el mismo patrón, sobre todo cuando los máximos se mantienen constantes.

Para 200 corridas:



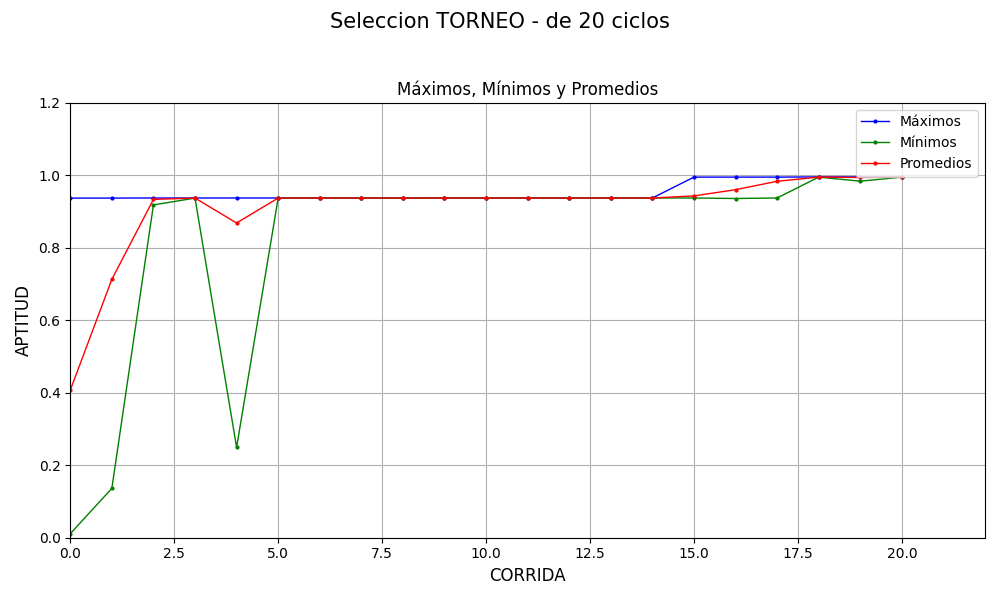
En esta ejecución de 200 ciclos, nuevamente se aprecia el comportamiento típico de un algoritmo genético sin elitismo, en el cual los valores de aptitud pueden experimentar oscilaciones importantes debido a la falta de preservación explícita de los mejores individuos. En la curva de máximos, se observa un empeoramiento en el máximo de una población, pero luego una rápida mejora, alcanzando valores cercanos al óptimo en menos de 25 ciclos. A lo largo de la simulación, el valor máximo se mantiene relativamente alto, aunque con algunas interrupciones momentáneas donde desciende. Esto indica que, si bien el algoritmo es capaz de generar soluciones de alta calidad, no tiene garantía de mantenerlas en el tiempo, como se evidencia en los descensos que ocurren.

La curva de mínimos muestra una inestabilidad mucho más marcada. Existen múltiples caídas abruptas a lo largo de todo el proceso, incluyendo picos negativos importantes incluso cerca del final, como en las generaciones 190 y 195. Estas caídas reflejan la introducción de individuos con baja aptitud, producto, una vez más, de mutaciones o combinaciones genéticas desfavorables.

Por su parte, la aptitud promedio sigue una tendencia ascendente durante las primeras generaciones, y logra estabilizarse en un valor alto en torno a los ciclos 80 a 100. Sin embargo, esta estabilidad no logra mantenerse ya que, como los máximos no son tan variables, tiende a seguir la curva de los mínimos. Además, los mínimos pueden llegar a tener picos muy bajos en comparación de los máximos, viéndose reflejados también en la curva del promedio.

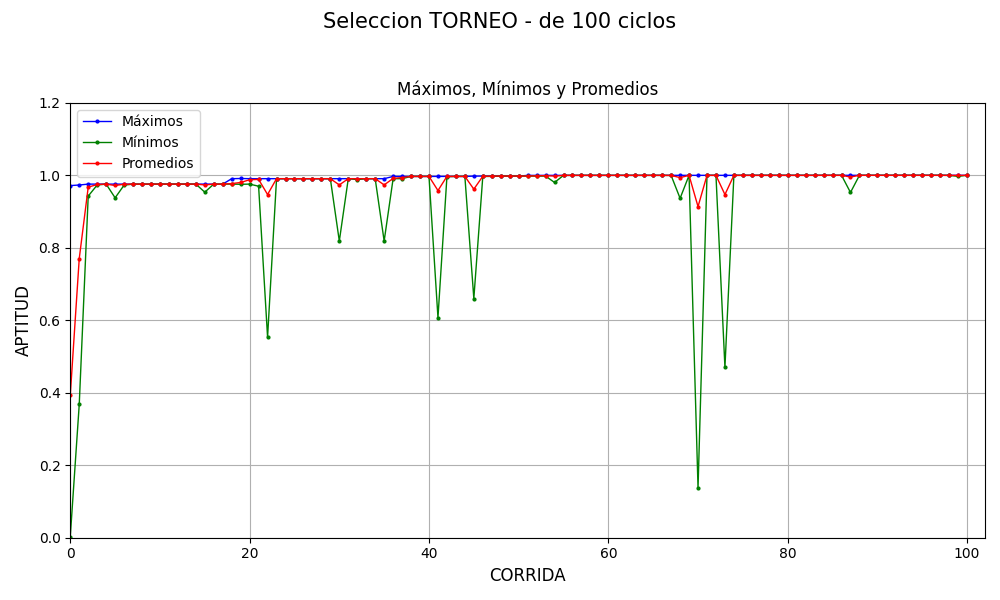
### Torneo

Para 20 corridas:



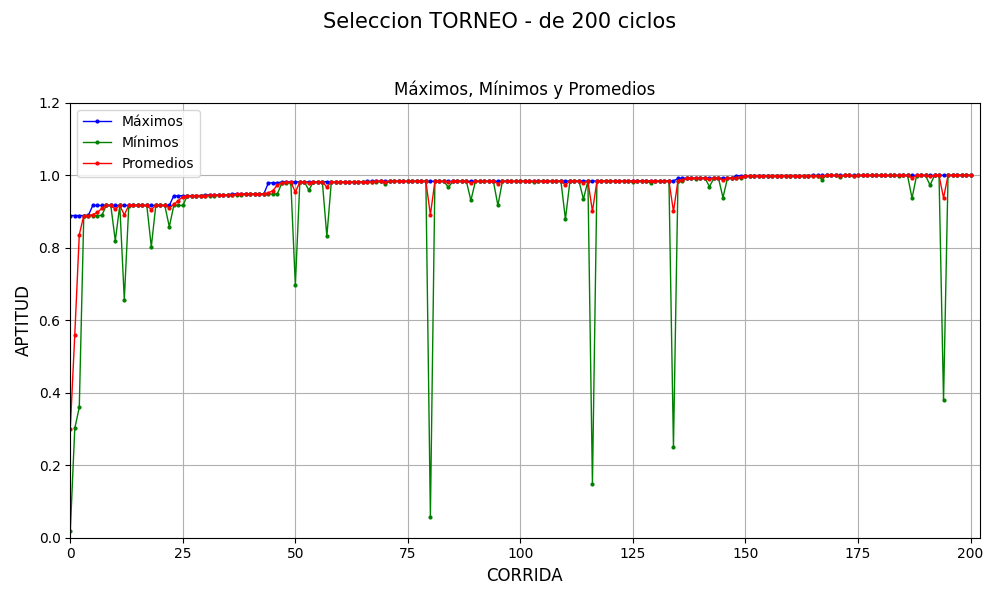
Aplicando la selección por torneo se puede observar que a medida que aumentan las generaciones, el promedio poblacional tiende al mejor individuo. En la gráfica superior se puede ver como en los primeros 5 ciclos hubo una variabilidad muy fuerte y luego se estabiliza alcanzando valores similares al individuo máximo.

Para 100 corridas:



En la gráfica correspondiente a los 100 ciclos con la selección de torneo se puede observar que el máximo asciende antes de los 40 ciclos hasta un valor cercano al 1 y luego se estabiliza, mientras que el mínimo asciende hasta casi alcanzar el valor máximo pero tiene caídas abruptas (debido a malas mutaciones o crossover) lo que lo hace más inestable.

Para 200 corridas:



La aptitud máxima comienza en valores cercanos a 0.9 y presenta un crecimiento gradual con algunos saltos marcados, alcanzando un valor cercano a 1 aproximadamente en la corrida 50. La curva de los valores mínimos presenta una gran variabilidad. Aunque en muchos ciclos se mantiene cerca del valor máximo, hay varias caídas bruscas y marcadas. Como ya mencionamos, la selección por torneo permite alcanzar soluciones de alta calidad de manera sostenida.

## Con elitismo

Para ambos casos métodos de selección se decidió realizar una corrida de cien generaciones.

### Ruleta

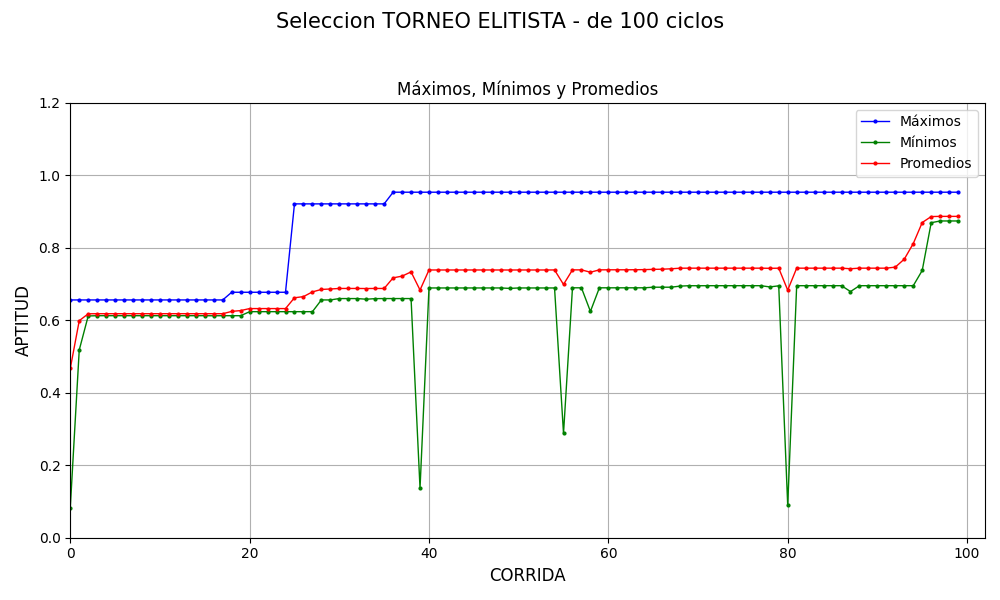


En esta simulación de 100 ciclos, se observa el efecto positivo del elitismo sobre el comportamiento de la corrida con ruleta como método de selección. A diferencia de los casos sin elitismo, la curva de máximos presenta una estabilidad sostenida, alrededor de la generación 40. Esto evidencia que las mejores soluciones se conservan de una generación a otra, protegiéndolas de los efectos aleatorios del crossover y la mutación. Además, el gran ascenso observado en la curva en la generación 40 refleja que posiblemente haya sido una mutación que dio como resultado tal individuo óptimo.

La curva de mínimos, por su parte, presenta una alta variabilidad con múltiples caídas pronunciadas, alcanzando valores cercanos a cero en generaciones como la 15, 60 y 95. Esto evidencia que, aunque aún pueden incorporarse individuos de baja aptitud a la población, su presencia no compromete la permanencia de las mejores soluciones gracias al efecto del elitismo. Además, se observa que estos individuos poco aptos tienden a desaparecer rápidamente, a diferencia de lo que ocurre en los escenarios sin elitismo. Esto se debe a que su probabilidad de ser seleccionados es muy baja en comparación con los individuos de alto rendimiento.

Por su parte, aunque la aptitud promedio coincide con algunas pequeñas caídas, se mantiene en un rango alto y bastante constante, lo que sugiere que la calidad general de la población se conserva con eficiencia. Estas caídas puntuales coinciden con descensos en los mínimos, pero sin impacto significativo sobre los valores máximos, lo cual confirma que el elitismo contribuye a una mejor estabilidad poblacional y a una convergencia más robusta del algoritmo.

### Torneo



En el gráfico de la selección por torneo con elitismo podemos ver como aumenta la separación del máximo respecto del mínimo. Los mínimos siguen mostrando caídas abruptas y el máximo crece rápidamente, aunque no alcanzan el valor perfecto (1).

También se muestra una evolución clara hacia la convergencia óptima. La curva de máximos comienza en un valor intermedio, ascendiendo en muy pocas ocasiones, estabilizándose en aproximadamente 0.99. Al igual que en el caso anterior, una vez alcanzado el valor máximo, el elitismo evita su pérdida, consolidando una población cada vez más apta. Con la implementación del elitismo se evita la pérdida de los mejores individuos por no haber sido seleccionados para competir.

La curva de mínimos también sigue una tendencia ascendente. Se observan algunos picos bruscos y breves hacia abajo, señal de la aparición esporádica de mutaciones. Sin embargo, al contar con elitismo, estos individuos no afectan la permanencia de los mejores, lo que evita que estos valores bajos se perpetúen.

La curva del promedio crece con una pendiente muy leve y constante, lo que indica una mejora colectiva sostenida de la población. Las oscilaciones son suaves, lo que refleja la efectividad del torneo (en este caso con elitismo) para preservar buenas soluciones y explorar nuevas al mismo tiempo.

# Conclusiones

A lo largo del desarrollo del trabajo práctico, se observaron diferencias marcadas en el comportamiento de los algoritmos genéticos según el método de selección utilizado y la presencia o ausencia de elitismo.

La selección por ruleta, si bien es sencilla de implementar y basa sus decisiones en una lógica probabilística proporcional al fitness, mostró una fuerte dependencia de las mutaciones para producir mejoras significativas. En varias simulaciones, los valores máximos se mantuvieron estancados hasta que una mutación afortunada introdujo una solución de mayor calidad, lo que evidencia una presión selectiva débil y una convergencia más lenta hacia el óptimo.

En contraste, la selección por torneo arrojó resultados más dinámicos, ya que favorece de manera directa a los individuos con mayor fitness dentro de subconjuntos aleatorios de la población. Esto permitió una mejora más rápida de los valores máximos, aunque también introdujo una mayor variabilidad en los mínimos. La falta de elitismo en ambos métodos permitió que individuos de alta calidad se pierdan ocasionalmente.

La lentitud de convergencia del método de ruleta se explica porque incluso los peores individuos conservan una probabilidad no nula de ser seleccionados, mientras que los mejores pueden quedar fuera del proceso de cruce. Por su parte, el método de torneo también presenta vulnerabilidades, si se enfrentan dos individuos de baja calidad en una misma competencia, uno de ellos avanzará; y si los mejores compiten entre sí, sólo uno continuará. Esto puede derivar en un deterioro de la calidad de la población. En este contexto, la incorporación de elitismo se reveló como un factor clave para estabilizar la evolución del algoritmo, ya que garantiza la conservación de las mejores soluciones y favorece tanto la convergencia como la mejora general de la población.

En resumen, mientras que la ruleta tiende a avanzar de forma más lenta y dependiente del azar, el torneo permite mejoras más rápidas, pero con menor estabilidad. El elitismo actúa como un mecanismo de protección que mitiga las debilidades de ambos métodos, permitiendo una evolución más sólida y sostenida hacia la solución óptima.