Universidad Galileo
Maestría en investigación de operaciones
Programación no lineal

PROYECTO FINAL

Nehemías Bejamín Lóez Macario 16005834
Luis José Soto 15000506
Carlos Alejandro Montiel Lorenzana 15000552

Objetivos

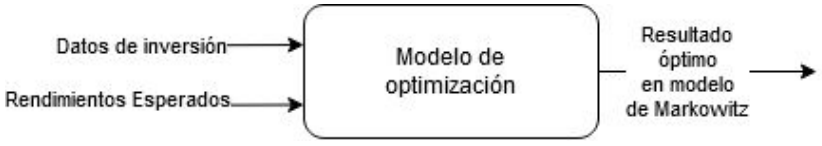
- Los estudiantes serán capaz de asimilar problemas financieros y resolver usando herramientas de programación no lineal
- Usar herramientas estadísticas para solucionar problemas reales
- Interpretar resultados obtenidos respaldados con software y análisis
- Resolver problemas cuadráticos orientados a la vida real

Planteamiento del Problema

Se realizará una simulación para el asesoramiento de inversión aplicando conceptos y herramientas de programación no lineal en donde, se tomará un portafolio de inversión por la cual minimizamos sus riesgos de inversión tomando en cuenta los rendimientos.
El inversionista desea determinar un portafolio óptimo de inversión, minimizando el riesgo de los rendimientos esperados, para ello vamos a hacer énfasis en la función objetivo y en en sus restricciones.

$$\begin{aligned} \text{mín} \quad & x^T \Sigma x \\ \text{s.a.} \quad & \mu^T x \geq R \\ & \sum_{i=1}^{n} x_i = 1 \\ & x_i \geq 0, \quad i = 1, \ldots, n. \end{aligned}$$

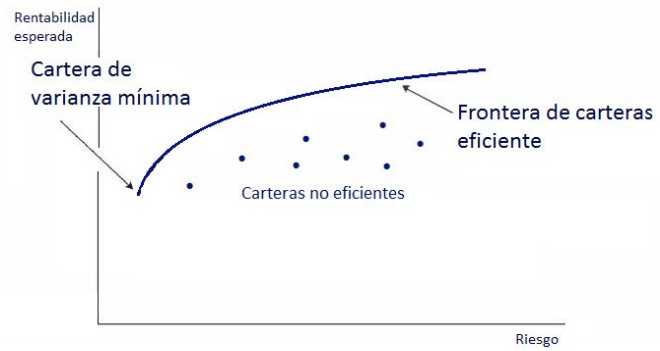
El inversionista desea resolver el problema mencionada según el modelo de Markowitz, el cual, iniciando de un set de datos de inversiones en el tiempo, se puede determinar el riesgo de todo el conjunto de inversiones tomando en cuenta cada inversión. Para este problema, se puede interpretar un modelo input/output como el siguiente:



Marco Teórico

1. **Modelo de Markowitz**

   Una cartera eficiente es una cartera que ofrece el mínimo riesgo para un valor de rentabilidad esperado. A través del siguiente gráfico lo veremos con más claridad:



   Se espera saber el riesgo de inversión para un rendimiento conocido cuyo objetivo es modelar un problema de optimización, así encontrando el punto óptimo (riesgo) para cada rentabilidad, en donde, se obtienen los siguientes resultados:

   1. Determinación del conjunto de carteras eficientes.
   2. Determinación de la actitud del inversor frente al riesgo.
   3. Determinar la cartera óptima.

   Para partir, se toman en cuenta los siguientes supuestos (López 2010):

   - La rentabilidad de una cartera viene dada por su esperanza matemática o media.
   - El riesgo de una cartera se mide a través de la volatilidad (según la varianza o desviación típica).
   - El inversor siempre prefiere la cartera con mayor rentabilidad y menor riesgo.

   Como ejemplo, podemos partir que en un inversionista tiene cierta información, de esa información se sabe que hay 10 años de los precios de dos bebidas energéticas A y B. Lo que hace el modelo de Markowitz, es comparar todos los precios, relacionarlos entre sí y conocer cuándo se obtiene más rendimiento con menos riesgo.

2. **Algoritmo KKT**

   Se ha decidido usar Karush Kuhn Tucker debido a que es capaz de resolver problemas de optimización con inecuaciones e igualdades.

   Para el problema de optimización en forma estándar:

$$\text{Optimizar:}$$
$$f(\mathbf{x})$$
$$\text{S.T :}$$
$$h_j(\mathbf{x}) = 0,$$
$$g_i(\mathbf{x}) \leq 0,$$

   Para que un punto x^* sea factible, es un punto regular y un minimizador local donde

todas las funciones son continuamente diferenciables, en donde, existe un $\lambda \in R^m$ y un no negativo $\mu \in R^p$ tal que,

A. $$f(x^*) + \sum_i^m \lambda_i \nabla h_i(x^*) + \sum_j^p \mu_j \nabla g_j(x^*)$$

B. $u_j g_j(x^*) = 0, j = [1:p]$ (complementary slackness)

C. $h(x^*) = 0, g(x^*) <= 0$ (Factibilidad)

En donde λ son los multiplicadores de lagrange y μ son los multiplicadores KKT.

3. SQP (Programación Cuadrática Secuencial)

Es un método iterativo el cual es muy comúnmente utilizado para la optimización de funciones no lineales restringidas y al mismo tiempo cumple con que su función objetivo y las restricciones de la misma, son diferenciables 2 veces y de forma continua.

Si tenemos la peculiaridad de que el problema no tiene restricciones, entonces el método se reduce al método de Newton para encontrar un punto en el cual notemos que el gradiente desaparece. Por otro lado, si bien el problema solo presenta restricciones de igualdad, entonces el método equivale a aplicar el método de Newton a las condiciones de primer orden, o condiciones de Karush-Kuhn-Tucker.

Algoritmo SLSQP

El algoritmo de programación secuencial de mínimos cuadrados descrito es un método cuasi-Newton (usando BFGS) aplicado a una función de lagrange que consta de una función de pérdida y restricciones de igualdad o desigualdad o ambas. Debido a que en cada iteración algunas de las restricciones de desigualdad se cumplen y en otras no, las desigualdades que no se cumplen se omiten para la siguiente iteración. Un problema con restricciones de igualdad se resuelve en cada paso utilizando el subconjunto de restricciones en la función de lagrange. Algo importante que cabe mencionar es que el optimizador utiliza una versión ligeramente modificada del solucionador de mínimos cuadrados no lineal NNLS de Lawson y Hanson.

Metodología

Se utilizarán paquetes del lenguaje de programación para la resolución del problema, se usará python 3.0+.
Para la documentación se usará google docs con la extensión de Auto-LaTeX para que todo el equipo pueda hacer ediciones en tiempo real.
Paquetes de software a utilizar:

1. Pandas
2. Sympy
3. numpy
4. matplotlib
5. csv

Software opcional para correr scripts

- Anaconda Navigator con Spyder

El árbol de archivos que se encontrarán disponibles dentro del .zip es el siguiente:

- informe.docx
- informe.pdf
- optimizacion_slsqp.py
- optimizacion_kkt.py
- tabla_resultados_slsqp.xslx
- tabla_resultados_kkt.xslx

Para hacer uso de los scripts instalar los paquetes en consola y correr normal con python o bien usar Spyder e insertar en el directorio Data1.csv y Data2.csv.

Resultados

- **Resultados con modelo KKT**
  Durante el desarrollo del algoritmo, se utilizó las gradientes de las ecuaciones de igualdad y desigualdad, formando así la siguiente ecuación de KKT:

$$\nabla f(x^*) + \Sigma\lambda \nabla h(x^*) + \Sigma\mu \nabla g(x^*) = 0$$

  Donde la función $h(x^*)$ representa las restricciones de igualdades y la función $g(x^*)$ representa las restricciones de desigualdad. Esto siempre y cuando se suman con el gradiente de la función.

  Tasa de rendimiento:
  Se utilizó durante en la implementación del algoritmo debido a que este nos da una ganancia o pérdida neta de una inversión durante un período de tiempo específico, que se expresa como un porcentaje.

```
12
13 #tasa de rendimiento
14 def Yielrate(P, Pt_1):
15     r_i = (P - Pt_1)/Pt_1
16     return r_i
17
```

$$r_i^t = \frac{P_i^t - P_i^{t-1}}{P_i{}^t - 1}, i = [1:n] t = [1:T]$$

Tasa de rendimiento promedio

Se utiliza para poder obtener un promedio en base a los valores que se obtuvieron en la tasa de rendimiento, esto es utilizado para poder formar la matriz de covarianza.

```
31 #tasa de rendimiento promedio
32 def Yielratemean(T, rprm_stocks, rprm_bonds, rprm_mm):
33     for i in range(len(list_stocks)):
34         rprm_stocks = list_stocks[i] + rprm_stocks
35         rprm_bonds = list_bonds[i] + rprm_bonds
36         rprm_mm = list_mm[i] + rprm_mm
37     rprm_stocks = rprm_stocks/T
38     rprm_bonds = rprm_bonds/T
39     rprm_mm = rprm_mm/T
40     return [rprm_stocks, rprm_bonds, rprm_mm]
```

$$r_i = \frac{1}{T} \sum_{t=1}^{T} r_i^t$$

Matriz de covarianza

Es una matriz que sus valores están formados por el grado de variación conjunta de varias variables aleatorias respecto a sus medias, es por eso que era necesario poder obtener el promedio de la tasa de rendimiento.

```
47 def Matrixcovariance(frame_rate):
48     result = 0
49     rows = []
50     for i in range(0,3):
51         column_j = frame_rate.columns[i]
52         for j in range(0,3):
53             column_k = frame_rate.columns[j]
54             for k in range(len(list_mm)):
55                 result += ((frame_rate[column_j][k] - rprm_array[i])*
56             rows.append(result/43)
57             result = 0
58         sigma.append(rows)
59         rows = []
60     return sigma
61
```

$$\Sigma_{ij} = \begin{cases} \sigma_i^2 & \text{si } \ i = j, \\ E[(r_i - \mu_i)(r_j - \mu_j)] & \text{si } \ i \neq j. \end{cases}$$

Seguidamente se comenzó por trabajar los gradientes de cada función:

Gradiente de la función objetivo:
Esta función retorna el gradiente del resultado del producto de las matrices:

```
77 #Devolvemos el gradiente de la funcion
78 def Gradientfunction(f, variables):
79     gradiente_function_array = []
80     for i in range(0,3):
81         gradiente_function_array.append(f[0].diff(variables[i]))
82     return (Matrix(gradiente_function_array))
83
```

$\nabla f(x^*)$

Gradiente de las funciones g(x) y h(x):
Ambas funciones nos retornan las ecuaciones necesarias para poder completar las restricciones que se nos solicitaron resolver:

$$\text{s.a.} \qquad \mu^T x \geq R$$

$$\sum_{i=1}^{n} x_i = 1$$

```
#Devolvemos el gradiente de la ecuación de desigualdad.
def Gradientinequality(equationsineq, variables):
    gradiente_ineq_array = []
    m = symbols('m')
    for i in range(0,3):
        gradiente_ineq_array.append(equationsineq[0].diff(variables[i
    return (Matrix(gradiente_ineq_array).T*m)

#Devolvemos el gradiente de la ecuación de igualdad.
def GradientEquality(equationseq, variables):
    gradiente_eq_array = []
    l = symbols('l')
    for i in range(0,3):
        gradiente_eq_array.append(equationseq.diff(variables[i]))
    return (Matrix(gradiente_eq_array).T*l)

```

Finalmente, se agregaron otro par de ecuaciones para con esto completar todas las necesarias para resolver en total las 6 ecuaciones encontradas, todo el proceso de elaboración del algoritmo se basó en las siguientes reglas:

$$\lambda \in \mathbb{R}^m,\ \mu \in \mathbb{R}^p,\ \mu \geq 0,$$

$$\nabla f(x^*) + \sum_{i=1}^{m} \lambda_i \nabla h_i(x^*) + \sum_{j=1}^{p} \mu_j \nabla g_j(x^*) = 0,$$

$$\mu_j g_j(x^*) = 0,\ j = 1, \ldots, p \quad \textit{(complementary slackness)},$$

$$h(x^*) = 0 \text{ and } g(x^*) \leq 0 \quad \textit{(feasibility)}.$$

Ahora, para el inversor tenemos una respuesta muy interesante, ya que hemos sido capaces de encontrar la frontera eficiente según el modelo de Markowitz, en donde, esta función a simple vista es convexa y tiene un punto óptimo.

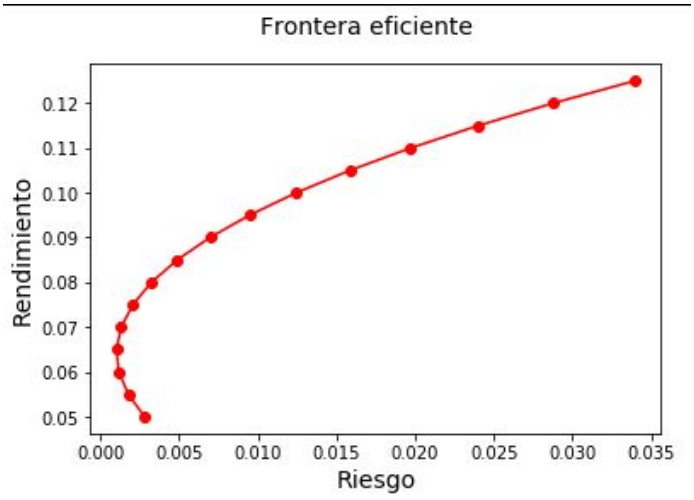
¿Es convexo el problema de optimización?

Para este problema solamente si $\nabla^2 f(x) > 0$ o bien si es positiva definida, para esto podemos obtener los eigenvalores de la matriz de covarianza y verificar si estos son positivos en su parte real.

$$\begin{pmatrix} 0.02778952824950455 & 0.0038708185299160657 & 0.00021097190345415958 \\ 0.0038708185299160657 & 0.011125194080315278 & -0.00019265422858954711 \\ 0.00021097190345415958 & -0.00019265422858954711 & 0.0011560612551425513 \end{pmatrix}$$

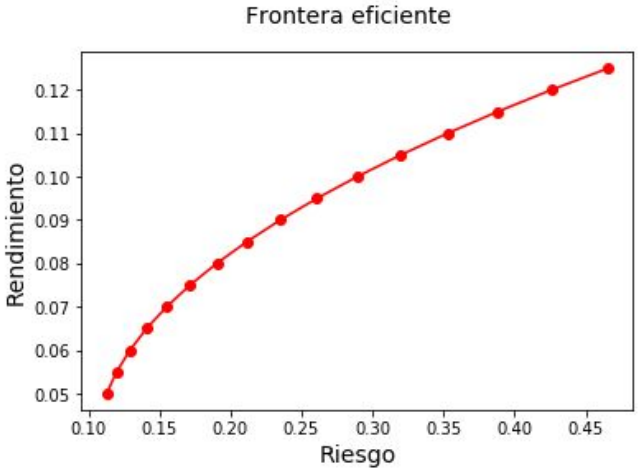
$\lambda_1 0.00114\ldots$
$\lambda_2 0.01027\ldots$
$\lambda_3 = \dfrac{25}{873}$

Se puede observar en los resultados, una variación de riesgo a la solución según el rendimiento esperado, (ver imágenes siguientes)



R Solución

R	Solución
0.05	0.00279183
0.055	0.00175337
0.06	0.00116021
0.065	0.00101234
0.07	0.00130978
0.075	0.00205251
0.08	0.00324055
0.085	0.00487388
0.09	0.00695251
0.095	0.00947644
0.1	0.01244567
0.105	0.0158602
0.11	0.01972003
0.115	0.02402515
0.12	0.02877558
0.125	0.0339713



R	Solución (Riesgo)
0.05	0.1128696
0.055	0.11989333
0.06	0.12927302
0.065	0.14100869
0.07	0.15510033
0.075	0.17154794
0.08	0.19035151
0.085	0.21151106
0.09	0.23502658
0.095	0.26089807
0.1	0.28912552
0.105	0.31970895
0.11	0.35264834
0.115	0.38794371
0.12	0.42559505
0.125	0.46560235

- Resultados con modelo SLSQP

Función Objetivo

```
76 def objetivo(x):
77     xStock = x[0]
78     xBound = x[1]
79     xMM = x[2]
80     aux1 = xStock*sigma[0,0] + xBound*sigma[0,1] + xMM*sigma[0,2]
81     aux2 = xStock*sigma[1,0] + xBound*sigma[1,1] + xMM*sigma[1,2]
82     aux3 = xStock*sigma[2,0] + xBound*sigma[2,1] + xMM*sigma[2,2]
83     return aux1*xStock + aux2*xBound + aux3*xMM
84
85
```

$$x^T \sum x$$

¿Qué se hace aquí?
Se están multiplicando toda la información del dataset [Stocks, Bonds, MM] por la matriz de covarianza y luego se vuelve a multiplicar la información del dataset, este comportamiento se puede observar qué es una función cuadrática debido a lo siguiente,
Sí, aux1*xStock
aux1 = xStock*sigma[0,0] + xBound*sigma[0,1] + xMM*sigma[0,2]
Entonces,
xStock^2*sigma[0,0]

Restricciones:

```
86 #Restruccción 1
87 def restriccion_1(x):
88     return (mu[0]*x[0] + mu[1]*x[1] + mu[2]*x[2] - R)
89
90 #Restriccion 2
91 def restriccion_2(x):
92     return (x[0] + x[1] + x[2] - 1)
93
```

$$\text{s.a.} \qquad \mu^T x \geq R$$

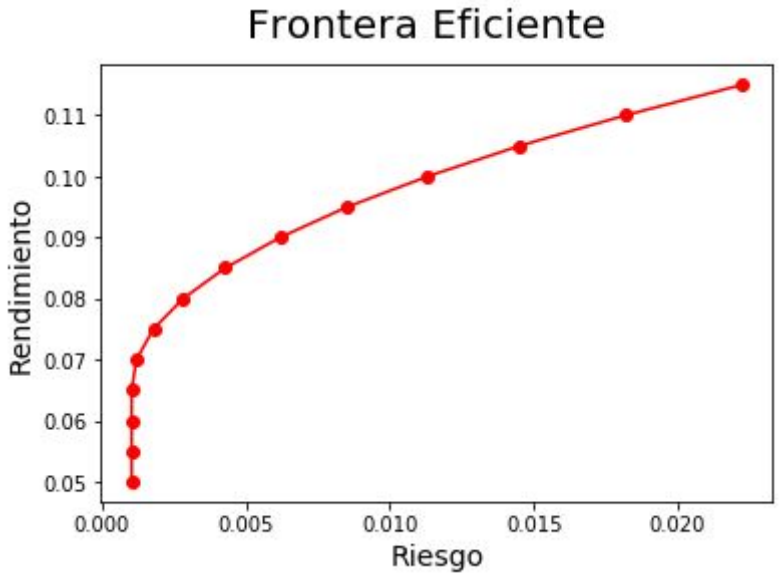
$$\sum_{i=1}^{n} x_i = 1$$

Soluciones:

```
107 R_arr = [0.05, 0.055, 0.06, 0.065, 0.07, 0.075, 0.08, 0.085, 0.09,
108 #R_arr = [0.05, 0.055, 0.06, 0.065, 0.07, 0.075, 0.08, 0.085, 0.09,
109 for i in R_arr:
110     R = i
111     solucion = minimize(objetivo, x0, method='SLSQP', constraints=r
112     #print(R, solucion.fun, sep=",")
113     Solucion_arr.append(solucion.fun)
114
```

Se genera una solución para cada rendimiento esperado, así obteniendo el riesgo para ese rendimiento.

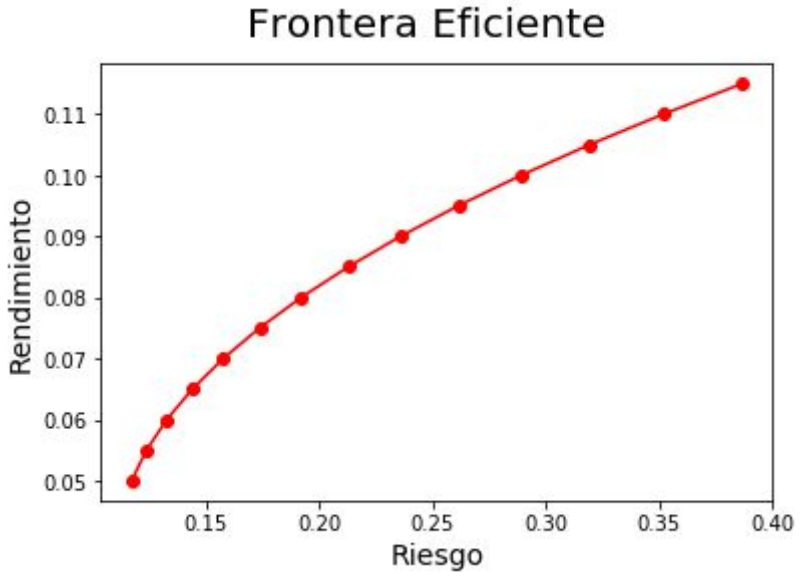
Resultados:
Usando SLSQP, y el Data1.csv



R	Solución (Riesgo)
0.05	0.001028386
0.055	0.001028386
0.06	0.001028386
0.065	0.001028386
0.07	0.00119203
0.075	0.001787957
0.08	0.002819272

0.085	0.004285975
0.09	0.006188065
0.095	0.008525542
0.1	0.011318465
0.105	0.014506831
0.11	0.018178456
0.115	0.022229327

Ahora, con Data2.csv



R	Solución (Riesgo)
0.05	0.117015076
0.055	0.123620495
0.06	0.132583507
0.065	0.143903172
0.07	0.157580429
0.075	0.173614964
0.08	0.192006777
0.085	0.212755869

0.09	0.23586224
0.095	0.261325889
0.1	0.289146816
0.105	0.319325022
0.11	0.351860506
0.115	0.386753269

Interpretación de resultados KKT y SLSQP

Para el inversor, podemos estar seguros de 2 cosas, según los algoritmos utilizados para el Dataset1 podemos ver claramente que el algoritmo KKT puro da una mejor interpretación de la frontera eficiente, tanto que a simple vista se puede ver el punto óptimo de la función frontera eficiente.

Punto óptimo en KKT (menor riesgo) vs SLSQP

R	riesgo en KKT	riesgo en SLSQP
0.065	0.00101234	0.143903172

Sin embargo, para el inversor lo mejor es presentarle los resultados para que el decida qué Rendimiento R utilizar.

Referencias y bibliografías

- (2010) Modelo de Markowitz. *Economipedia.* Recuperado de https://economipedia.com/definiciones/modelo-de-markowitz.html
- The Journal of Finance, Vol. 7, No. 1. (Mar., 1952), pp. 77-91.
- (2020) Karush Kuhn Tucker conditions. *Wikipedia.* https://en.wikipedia.org/w/index.php?title=Karush%E2%80%93Kuhn%E2%80%93Tucker_conditions
- S. Boyd and L. Vandenberghe (2004),Convex Optimization,Cambridge University Press, Chapter 5
- R. T. Rockafellar (1970),Convex Analysis, PrincetonUniversity Press, Chapters 28–30
- (2014). SLSQP - Sequential Least Squares Programming. *pyOpt.* Recuperado de http://www.pyopt.org/reference/optimizers.slsqp.html#slsqp-sequential-least-squares-programming
- (2016) SLSQP. *Degeneratonic.* Recuperado de http://degenerateconic.com/slsqp/
- (2019) ¿Qué es la matriz de covarianzas?. *minitab* Recuperado de https://support.minitab.com/es-mx/minitab/18/help-and-how-to/modeling-statistics/anova/supporting-topics/anova-statistics/what-is-the-variance-covariance-matrix/