Informe Trabajo Final Modelo de Optimización 2


```
Participantes:
  Daniel Orlando Ortiz Pacheco: C-412
  Aldo Javier Verdesia Delgado: C-412
  Yansaro Rodríguez Paez: C-412
  Javier Alejandro Valdés González: C-411
```


Estado del Arte

Al momento del desarrollo de este proyecto, el contenido "algorítmico" que se imparte en la asignatura de Modelo de Optimización se encuentra mayormente relacionado a todos los matices del algoritmo Simplex. Referente a este tema a lo largo del curso de desarrolla múltiples tareas, tanto teóricas como prácticas. En la parte práctica de este contenido, que son mayormente laboratorios evaluativos, los estudiantes deben resolver diversos problema y emplear el algoritmo de distintas formas. Pero no se cuenta con una herramienta que permita tanto la explotación de los algoritmos, como la consolidación de sus distintas definiciones teóricas. Posibles aproximaciones serian algunos páginas en internet y librerías de distintos lenguaje, que junto a la resolución del problema dado, aportan el paso a paso de la solución.Dicha aproximación se queda corto para los intereses del cliente, mientras que el desarrollo total del algoritmo por el estudiante se ajusta más a la idea, pero se excede, pues no tampoco es la idea la profundización en detalles demasiado técnicos

Problema

El objetivo de este trabajo es diseñar e implementar un lenguaje de programación que permita describir e implementar algoritmos para la solución del problema de programación lineal inspirados al estilo del Simplex. Eso quiere decir que en el lenguaje se debe poder implementar un método simplex o dual simplex con pivoteo tradicional (como se enseña en las clases), pero también las versiones del simplex revisado. También debería ser posible describir e implementar métodos basados en planos cortantes, en ramificación y acotación o algunas otras variantes que se puedan deducir de ellos, por ejemplo, una combinación de estas últimas. Usando el lenguaje debería ser capaz de implementar un método que dado un problema de programación lineal, devuelva la tercera mejor solución fáctible básica, o un método de planos cortantes en los que resolver un problema de programación lineal (con un método basado en Simplex) sea una instrucción.

Desarrollo

Primeras Observaciones

Luego de debatir varias ideas con el cliente; se terminó concluyendo que se ajustaba más un pequeño framework, que este orientado hacia el dominio y ter-

minología del `simplex` y toda la teoría que este contiene, que un nuevo lenguaje implementado desde cero. El *DSL* que necesita el cliente debe tener las siguientes características:

- *Debe ser un lenguaje imperativo*: Al ser un framework, y no un nuevo lenguaje, el *DSL* ya de inicio cuenta con todas las herramientas necesarias para cumplir con estas características provenientes del lenguaje base
- *El código resultante debe ser lo más cercano posible a la descripción teórica del algoritmo*: Para lo cual el desarrollo se apoya en algunas características especiales del lenguaje para simular, sistemas de ecuaciones, definiciones de funciones de cambios y pivoteos , etc …
- *Debe tener la características de no ser imprescindible*: El fin del proyecto es, lograr un producto final que sirva de apoyo para el estudio y la evaluación de los contenidos relacionados con el algoritmos `simplex` en la carrera de ciencias de la computación. Debido al método evacuativo de la asignatura, la acumulación de créditos a lo largo del curso que puedan facilitar las evaluaciones finales, por tanto si el *DSL* resultante formará parte de dichas evaluaciones los estudiantes con pocos o ningún crédito también deben poderse evaluar. Razón que refuerza la idea de un framework frente a un nuevo lenguaje, pues siempre se podría prescindir del framework e implementar los algoritmos en el lenguaje base

Implementación

Apoyados en las consideraciones iniciales, se selecciono como lenguaje base para el desarrollo del framework, al lenguaje **Python**, lenguaje imperativo, con el que los estudiantes de ciencia de la computación tiene bastante contacto a lo largo de la carrera, además de contar con los famosos "métodos mágicos" que brinda grandes facilidades para el desarrollo de dsl internos. Además **Python** cuenta con la librería **numpy** que será de gran apoyo para el trabajo con matrices, vectores y demás. El desarrollo de dividió de en varias fases

- *Formato*: en esta sección de desarrollo todas las herramientas que para lograr un dsl interno que brindara la sensación de estar definiendo un problema de optimización linea y su posterior transformación de la forma estándar, entrada requerida de los algoritmos `simplex`. Ejemplo:

```
pol.min_z = 2 * x[0] + 1 * x[1] + 3 * x[2] + 2 *x[3] +1* x[4]
pol.s_a = (
     1 * x[0] + 1 * x[1] + 1 * x[2] + 1* x[3] == 9,
    -1 * x[0] + 1 * x[1] + 1 * x[2] + 1 * x[4] == 3
    )
```

- *Forma Explicita*: se definen varias estructuras para facilitar al desarrollador la definición, de forma matemática de los distintos componentes de los algoritmos `simplex`, como pueden ser *y0* y *rj*

```
y0 = mo.inverse_matrix(B) * stand_eq.result_vector
ztr = ctb * mo.inverse_matrix(B) * R
rj = [ (ctr[ir] - ztr[ir]) if index_in_base else 0
        for index_in_base, _, ir in stand_eq.list_var_index_by(base) ]
```

- *Simplex*: se implementan las diferentes herramientas para facilitar tanto la definición de algoritmo `simplex` como su version de dos fases

```
while mo.check_optimal_condition(simplex_to_eq):
  aq = mo.find_input_column(simplex_to_eq)
  ap = mo.find_swap_column_to(aq, simplex_to_eq,
          _min = lambda yio, yiq: yiq/yio if yio > 0 else None)

  if ap is None: break # problema no acotado

  ypq = simplex_to_eq.Ax[ap][aq] # pivote
  simplex_to_eq = mo.swap_column_base(aq, ap, simplex_to_eq,
      formule= lambda i, j, yij, yiq, ypj: ypj/ypq if i == ap
                                              else yij - yiq * ypj / ypq)
```

- *Problema Dual*: Apoyado en la etapa anterior (Simplex), contiene varios mecanismos para realizar el análisis del problema dual y atravez del mismo lograr el resultado del problema principal. La estructura es muy parecida a la anterior salvo que los métodos para seleccionar las columnas de la operación serán:

```
ap = mo.dual_find_output_column(simplex_to_eq)
aq = mo.dual_find_input_column(ap, simplex_to_eq,
            _min = lambda rj, ypj: -rj/ypj if ypj < 0 else None)
```

- *Cortes*: Como etapa final se desarrollo algunos métodos para complementar y enriquecer, la gama de problemas que el algoritmo simplex puede resolver, ya sea la acotación de problemas no acotado o la búsqueda de soluciones enteras para ciertos sistemas:

```
# búsqueda de la columna base para generar un corte
curt_file = mo.plane_cut_row_selection(simplex_to_eq,
                            formule= lambda y: ((mo.zp(y) - y) * -1))
# generación del corte
curt, symbol, result = mo.cut_plane(simplex_to_eq, curt_file,
                            formule= lambda y: ((mo.zp(y) - y) * -1) )
# inclusión de la nueva fila
simplex_to_eq = mo.include_row(simplex_to_eq, curt, symbol, result)
```

Finalizando la implementación con un decorador para todos los métodos que servirá de control de que herramientas, se podrán usar en el desarrollo de algún ejercicio y cuales no. Este control se llevará a cabo a partir del archivo `sale.txt` que se encuentra junto a este informe y a los ejemplos aportados. En este archivo debe encontrarse los distintos nombres de los métodos el desarrollador

haya comprado y tenga derecho a usar, de lo contrario el framework lanzara errores en respuesta al uso ilegal de sus métodos. Este mecanismo cuenta com una sistema neutro que es la palabra reservada `all`, si esta palabra se encuentra en el archivo señalado, el sistema reconocerá toda llamada a sus método como legitima

Documentación del DSL

El resultado final del trabaja es un framework con múltiples funciones y herramientas que facilitara la definición y el procesamiento de un **Problema de Optimización Linea**, así como su resolución mediante el algoritmo `simplex` y sus diferente interpretaciones. Dicho framework se documenta a continuación (Ver ejemplo 1: conf_simplex_example_1.py)

El framework se conforma de un gran módulo de funciones estáticas, que el desarrollador debe importa:

```
import simplex_framework as mo
```

La `format_componet` es la función inicial, esta tiene como resultado una tupla de dos elementos una "lista infinita de variables" y un POL, el cual espera que se le asignen las propiedades min_z o max_z y s_a haciendo uso de la "lista infinita de variables". El método `format_componet`siempre debe acompañarse del método `get_stand_form` que dado un POL genera otro en la forma estándar, requisito necesario para el algoritmo `simplex`. De la explotación de este método se obtiene la siguiente sintaxis:

```
x, pol = mo.format_componet()
pol.min_z = 2 * x[0] + 1 * x[1] + 3 * x[2] + 2 *x[3] +1* x[4]
pol.s_a = (
       1 * x[0] + 1 * x[1] + 1 * x[2] + 1* x[3] == 9,
      -1 * x[0] + 1 * x[1] + 1 * x[2] + 1 * x[4] == 3
      )

stand_pol = mo.get_stand_form(pol)
```

Seguido de la definición del problema que se desea optimizar comienza los distintos pasos del algoritmo, comenzando por la selección de la base de la matriz se restricciones. En este pasos comienzan las bifurcación pues en dependencia de si la matriz presenta una base canónica o no, se pueden decidir varias opciones en la resolución del ejercicio. Por tanto el framework aporta una lista de funciones para cubrir todas las posibles decisiones del desarrollador.(Ver ejemplo 1: conf_simplex_example_2.py)

```
# retorna una lista de indices de las columnas canónicas
canonical_columns = mo.get_canonical_columns(pol)
# predicado afirmativo si las columnas de la lista son una base de Ax
is_base = mo.is_base(canonical_columns, pol)
# completa una base de Ax que contenga los vectores de la lista
```

```
# del segundo argumento, este es opcional y por default = []
base = mo.get_base(pol, canonical_columns)

pol_to_firts_fase, base = mo.complet_canonical_base(pol, canonical_columns)
# crea un nuevo problema al que se le agregan
# vectores canónicos que faltaban en el problema inicial
# además retorna la base canonica completada
```

Una vez seleccionada la base con la que se comenzará el algoritmo, independiente de la bifurcación seleccionada, el siguiente pasos sería realizar las transformaciones necesarias para expresar el problema de manera explicita respecto a la base seleccionada. En pos de ganar en expresividad y lenguaje matemático se definió la interfaz `Operador`, al que se le modificaron los principales operadores aritméticos y que internamente utilicé las operaciones que brinda `numpy` en el trabajo con matrices y vectores:

```
ctb, ctr, B, R = mo.explicit_descompose(stand_eq, base)
y0 = mo.inverse_matrix(B) * stand_eq.result_vector
ztr = ctb * mo.inverse_matrix(B) * R

# el método list_var_index_by es un método de la clase LinealOptimizationProblem
# esta clase es la interface de entrada a todos los métodos antes expuestos
# el método es un iterador de 0 a la dimension del problema
# en cada paso resuelve una terna de
# 1- booleano que responde a si el indice pertenece a la base
# 2- indice de iteración
# 3- indice de iteración de los vectores que no pertenecen a la base
rj = [(ctr[ir] - ztr[ir]) if index_in_base
      else 0 for index_in_base, _, ir in stand_eq.list_var_index_by(base) ]
```

Una vez se tienen las herramientas para iniciar el algoritmo el framework ofrece las herramientas para escribir el mismo lo más cerca posible a la propia definición, tomando como base la interface `Simplex`, definida de la siguiente manera:

```
class Simplex:
    def __init__(self, base, ctx, Ax, y0, rj) -> None:
        self.base = base
        self.ctx = ctx
        self.Ax = np.array(Ax)
        self.y0 = y0
        self.rj = rj
```

Apoyado en dicha interface y en varios métodos del framework se optiene la siguiente sintaxis :

```
# instancia Simplex
simplex_to_pol = mo.simplex_build(base, ctb, ctr, B, R, y0, rj)

while not mo.check_optimal_condition(simplex_to_pol):
```

```
# busca el menor rj negativo y retorna la columna
aq = mo.find_input_column(simplex_to_pol)
# el method find_swap_column_to busca la fila p de Ax
# tal que la function _min sea mínima y distinta de None
# siempre que exista aq
ap = mo.find_swap_column_to(aq, simplex_to_pol,
                _min = lambda yio, yiq: yiq/yio if yio > 0 else None)

if ap is None: break # problema no acotado

ypq = simplex_to_eq.Ax[ap][aq] # pivote

# realiza el cambio de base aplicando a lo largo de
# toda la tabla del simplex la función formule
simplex_to_eq = mo.swap_column_base(aq, ap, simplex_to_eq,
    formule= lambda i, j, yij, yiq, ypj: ypj/ypq if i == ap
                                        else yij - yiq * ypj / ypq)
```

De manera similar se puede trabajar sobre la interfaz **Simplex** para realizar el desarrollo de problema dual. (Ver ejemplo 3: conf_dualsimplex_example_1.py)

```
while not mo.is_dual_faceable(simplex_to_eq):
    # busca el menor y0 negativo y retorna la fila
    ap = mo.dual_find_output_column(simplex_to_eq)
    # el method find_swap_column_to busca la columna q de Ax
    # tal que la function _min sea mínima y distinta de None
    # siempre que exista aq
    aq = mo.dual_find_input_column(ap, simplex_to_eq,
              _min = lambda rj, ypj: -rj/ypj if ypj < 0 else None)

    if aq is None: break

    ap = simplex_to_eq.base.index(ap)
    ypq = simplex_to_eq.Ax[ap][aq]
    # realiza el cambio de base aplicando a lo largo de toda la
    # tabla del simplex la función formule
    simplex_to_eq = mo.swap_column_base(aq, ap, simplex_to_eq,
        formule= lambda i, j, yij, yiq, ypj: ypj/ypq if i == ap
                                            else yij - yiq * ypj / ypq)
```

Por lo contrario se puede realizar el mismo análisis partiendo de el problema de optimización lineal inicial, empleando el siguiente método para obtener el problema dual y luego seguir el desarrollo del simplex para este nuevo problema

```
dual_pol =get_dual_problem(pol)
```

Para complementar estos desarrollos, el framework incluye una serie de métodos para editar y acotar el problema para encontrar soluciones mas especificas: (Ver

ejemplo 4: conf_curt_example.py)

```
# facilita la búsqueda de una columna optima, según formule,
# para realizar algún corte
# mo.zp es un redondeador aportado por el framework
# más acorde a las necesidades de estos algoritmos
curt_file = mo.plane_cut_row_selection(simplex_to_eq,
                formule= lambda y: ((mo.zp(y) - y) * -1))
# Construye un corte apoyado en una columna, y utiliza formule para computar los
# valores del corte según el algoritmo de "Planos cortantes"
curt, symbol, result = mo.cut_plane(simplex_to_eq, curt_file,
                           formule= lambda y: ((mo.zp(y) - y) * -1) )
# Construye un corte apoyado en una columna
# con valores de corte según el algoritmo de "Dual Todos Enteros"
curt, symbol, result = mo.cut_plane_z(simplex_to_eq, curt_file )
# Construye un corte apoyado en una columna,
# con valores de corte según el algoritmo de "Corte Primal Todo Enteros"
curt, symbol, result = mo.cut_primal_z(simplex_to_eq, curt_file )

# Construye una nueva interfaz de simplex agregando el nuevo corte
simplex_to_eq = mo.include_row(simplex_to_eq, curt, symbol, result)
```

Distribución de los Participantes

- Daniel Orlando Ortiz Pacheco: Diseño y desarrollo de las herramientas para la definición del los conceptos variable, innecuación y problema de optimización lineal. Además de facilitar la trasformación de este ultimo a la interfaz de **Simplex** definida por sus compañeros. Investigación general del problema y sus posibles soluciones, si como toma de contactos con el cliente para concretar ideas
- Aldo Javier Verdesia Delgado: Desarrollo y diseño de toda la face “Método Simplex”, implementación de todas las herramientas para los movimientos y operaciones tabulares. Diseño de la interface de simplex utilizada a lo largo del proyecto
- Yansaro Rodríguez Paez: Implementación de los métodos necesarios para realizar el desarrollo del algoritmo DualSimplex, coordinación con las faces anteriores y posteriores para lograr un desarrollo coherente y de fácil acoplamiento
- Javier Alejandro Valdés González:Desarrollo y diseño de todas las interfaces, que faciliten el uso de los distintos cortes que brinda el framework y la futura definición de nuevos cortes