

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG-HCM**  
**KHOA TOÁN – TIN HỌC**



**BÁO CÁO ĐỒ ÁN**  
**BÀI TẬP CODE MÔN QUY HOẠCH**  
**TUYẾN TÍNH**

**Môn Học:     QUY HOẠCH TUYẾN TÍNH.**

**Giảng viên:   NGUYỄN LÊ HOÀNG ANH.**

## MỤC LỤC

I.	Bảng thông tin sinh viên .....	3
II.	Giới thiệu sản phẩm .....	4
1.	Thông tin .....	4
2.	Ngôn ngữ lập trình .....	4
3.	Kết quả đạt được .....	5
4.	Hạn chế .....	6
III.	Mô tả .....	6
1.	Mô tả thuật toán .....	6
2.	Mô tả cách sử dụng .....	27
IV.	Ví dụ .....	31
1.	Thuật toán đơn hình .....	31
2.	Thuật toán Bland .....	34
3.	Thuật toán hai pha .....	36
V.	Tài liệu tham khảo .....	38

## I. Bảng thông tin sinh viên

STT	MSSV	Tên	Nhiệm vụ	Đánh giá
1	21280016	Trần Minh Hiền	<ul style="list-style-type: none"><li>- Tìm hiểu và xây dựng các hàm xử lý cho phương pháp hai pha.</li><li>- Đưa ra các bài toán ví dụ, kiểm tra tính đúng đắn của các hàm xử lý.</li></ul>	100%
2	21280040	Trần Ngọc Khánh Như	<ul style="list-style-type: none"><li>- Tìm hiểu và xây dựng các hàm xử lý cho phương pháp đơn hình và Bland.</li><li>- Phân tích cách thực thi các hàm trong hệ thống.</li></ul>	100%
3	21280104	Lâm Gia Phú ( <b>Nhóm trưởng</b> )	<ul style="list-style-type: none"><li>- Xây dựng website, triển khai hoạt động 24/7.</li><li>- Xử lý thông tin input, truyền dữ liệu về cho hệ thống xử lý. Hiển thị kết quả sau khi đã xử lý thông tin.</li></ul>	100%

Chữ ký sinh viên 1

Trần Minh Hiền Trần

Chữ ký sinh viên 2

Trần Ngọc Khánh Như

Chữ ký sinh viên 3

Lâm Gia Phú

## II. Giới thiệu sản phẩm

### 1. Thông tin

- Quy Hoạch Tuyến Tính từ lâu đã trở thành bài toán học búa, đòi hỏi tư duy logic và kỹ năng giải quyết vấn đề phức tạp. Nhằm đáp ứng nhu cầu ngày càng cao của việc học tập và giải quyết các bài toán quy hoạch tuyến tính hiệu quả, website HNP ra đời như một "cánh tay đắc lực" cho các bạn sinh viên trên con đường tìm tòi học hỏi của mình.
- Điểm nổi bật của website là tích hợp đầy đủ 3 thuật toán tối ưu hóa tiên tiến cho bài toán Quy Hoạch Tuyến Tính:
  - o Thuật toán Đơn hình: Là thuật toán giải quyết QHTT theo phương pháp cổ điển, được sử dụng rộng rãi bởi tính đơn giản và hiệu quả cao. Thuật toán này hoạt động bằng cách lặp đi lặp lại các bước để tìm ra điểm cực đại hoặc cực tiểu của hàm mục tiêu, tuân theo các điều kiện ràng buộc của bài toán.
  - o Thuật toán Bland: Khắc phục nhược điểm của thuật toán Đơn hình, thuật toán Bland sử dụng quy tắc chọn biến đi ra khỏi cơ sở một cách linh hoạt, giúp giải quyết các bài toán quy hoạch tuyến tính phức tạp hơn, đặc biệt là khi có nhiều biến và ràng buộc.
  - o Thuật toán Hai pha: Phù hợp cho các bài toán quy hoạch tuyến tính có nhiều biến và ràng buộc, thuật toán Hai pha chia bài toán thành hai giai đoạn: Giai đoạn 1 biến đổi bài toán về dạng chuẩn và Giai đoạn 2 sử dụng thuật toán Đơn hình để giải quyết.
- **Giao diện thân thiện, hướng dẫn chi tiết:** Website được thiết kế với giao diện trực quan, dễ sử dụng, phù hợp với mọi đối tượng người dùng, kể cả người mới bắt đầu. Các bước giải toán được hướng dẫn chi tiết, rõ ràng, giúp người dùng dễ dàng thao tác và hiểu rõ quy trình giải quyết bài toán.
- **Hiệu quả và tin cậy:** Website hoạt động 24/7, đảm bảo tính ổn định và khả năng tiếp cận mọi lúc mọi nơi.
- **Tiết kiệm thời gian và công sức:** Website giúp bạn tự động hóa quá trình giải quyết các bài toán, tiết kiệm thời gian và công sức so với việc giải toán thủ công.
- **Nâng cao hiệu quả công việc:** Website giúp bạn đưa ra quyết định tối ưu cho bài toán quy hoạch tuyến tính, góp phần nâng cao hiệu quả và chất lượng học tập cho các bạn sinh viên.
- ➔ **Kết luận:** Website hỗ trợ giải bài toán Quy Hoạch Tuyến Tính là một công cụ đắc lực cho các nhà nghiên cứu, học viên và chuyên gia trong nhiều lĩnh vực. Với hệ thống thuật toán tối ưu hóa mạnh mẽ, giao diện thân thiện và hướng dẫn chi tiết, website giúp bạn giải quyết các bài toán QHT một cách nhanh chóng, hiệu quả và chính xác nhất.

### 2. Ngôn ngữ lập trình

- **Ngôn ngữ lập trình:**
  - o **Python:** Lựa chọn Python là ngôn ngữ lập trình cốt lõi cho website mang đến nhiều ưu điểm:
    - **Mạnh mẽ và linh hoạt:** Python sở hữu khả năng xử lý dữ liệu phức tạp, đáp ứng tốt các yêu cầu của website.
    - **Cộng đồng rộng lớn:** Python được sử dụng phổ biến, do đó dễ dàng tìm kiếm tài liệu hướng dẫn và hỗ trợ khi cần thiết.
    - **Dễ học, dễ sử dụng:** Ngôn ngữ Python có cú pháp đơn giản, dễ hiểu, giúp giảm thiểu thời gian và chi phí phát triển.

- **Công cụ và nền tảng triển khai web:**

- **Flask:** Lựa chọn Flask làm framework web cho website mang lại nhiều lợi ích:
  - **Nhẹ và hiệu quả:** Flask là framework web nhẹ, giúp tối ưu hóa hiệu suất website.
  - **Dễ sử dụng:** Flask có cú pháp đơn giản, dễ học, giúp giảm thời gian phát triển.
  - **Linh hoạt:** Flask cho phép tùy chỉnh cao, đáp ứng đa dạng nhu cầu của website.
- **Render:** Lựa chọn Render làm nền tảng triển khai web mang đến nhiều ưu điểm:
  - **Dễ dàng triển khai:** Render cung cấp quy trình triển khai đơn giản, giúp website hoạt động nhanh chóng.
  - **Quản lý hiệu quả:** Render hỗ trợ quản lý website hiệu quả, bao gồm theo dõi hiệu suất, bảo mật và cập nhật.
  - **Mở rộng linh hoạt:** Render cho phép website mở rộng dễ dàng khi nhu cầu sử dụng tăng cao.

- **Các tính năng kỹ thuật:**

- **Tích hợp Flask và Python:** Sự kết hợp giữa Flask và Python tạo nên nền tảng vững chắc cho website, giúp xây dựng trang web hiệu quả, dễ dàng bảo trì và nâng cấp.
- **Xử lý dữ liệu động:** Website có khả năng xử lý dữ liệu đầu vào từ người dùng một cách linh hoạt và hiệu quả, đáp ứng mọi yêu cầu tính toán phức tạp.
- **Tích hợp Git và CI/CD:** Việc tích hợp Git và CI/CD giúp quản lý mã nguồn hiệu quả, tự động hóa quy trình triển khai, đảm bảo website luôn hoạt động ổn định và cập nhật liên tục.

➔ **Kết luận:** Sự kết hợp giữa các công nghệ tiên tiến như Python, Flask, Render và Git/CI/CD đã tạo nên website HNP với hiệu suất cao, khả năng xử lý dữ liệu mạnh mẽ và tính bảo trì dễ dàng. Website hoạt động 24/7 trên nền tảng Render, đảm bảo tính ổn định và sẵn sàng đáp ứng mọi nhu cầu của người dùng.

3. **Kết quả đạt được**

- **Giao diện thân thiện:** Website được thiết kế với giao diện trực quan, dễ sử dụng, giúp người dùng dễ dàng nhập dữ liệu bài toán và thao tác giải quyết. Hệ thống hướng dẫn chi tiết và cảnh báo lỗi khi nhập sai đảm bảo tính chính xác của dữ liệu đầu vào.
- **Hoạt động liên tục:** Website hoạt động 24/7 trên cả thiết bị di động và máy tính, cho phép người dùng truy cập và sử dụng mọi lúc mọi nơi.
- **Hỗ trợ đa phương pháp:** Website tích hợp 3 phương pháp giải bài toán Quy Hoạch Tuyến Tính phổ biến: Đơn hình, Bland và Hai pha, đáp ứng nhu cầu giải quyết đa dạng của người dùng.
- **Hiển thị chi tiết quá trình giải:** Website cung cấp tên phương pháp sử dụng, giá trị biến và kết quả sau mỗi bước lặp, giúp người dùng dễ dàng theo dõi và kiểm tra tính chính xác của giải pháp.
- **Giải quyết nhiều trường hợp:** Website có khả năng giải quyết các bài toán Quy Hoạch Tuyến Tính với nhiều trường hợp khác nhau như tồn tại nghiệm duy nhất, vô nghiệm, vô số nghiệm và bài toán giới nội.

#### 4. Hạn chế

- **Thời gian tải trang:** Website có thể mất khoảng 5 phút để tải khi truy cập lần đầu do hạn chế về gói VPS.
- **Hiển thị bảng giá trị:** Hiện tại, website chưa hiển thị bảng giá trị một cách trực quan, người dùng cần tự kiểm tra cột giá trị tương ứng với biến trong bảng.

### III. Mô tả

#### 1. Mô tả thuật toán

##### a. Hàm `__init__`

```
def __init__(self, num_vars, num_cons, is_min, obj_coeffs,
constraint_matrix, constraint_rhs, constraint_signs, variable_signs):
    self.num_vars = num_vars
    self.num_cons = num_cons
    self.is_min = is_min
    self.obj_coeffs = obj_coeffs
    self.constraint_matrix = constraint_matrix
    self.constraint_rhs = constraint_rhs
    self.constraint_signs = constraint_signs
    self.variable_signs = variable_signs
    self.num_new_vars = 0
```

- **Công dụng:** Khởi tạo đối tượng `LinearProgrammingProblem`.
- Tham số truyền vào:
  - o `num_vars`: Số biến của bài toán.
  - o `num_cons`: Số ràng buộc của bài toán.
  - o `is_min`: Biến boolean, nếu là `True` (hoặc 1) thì bài toán là min, nếu `False` (hoặc 0) thì bài toán là max.
  - o `obj_coeffs`: Mảng chứa các hệ số của hàm mục tiêu.
  - o `constraint_matrix`: Ma trận hệ số của các ràng buộc.
  - o `constraint_rhs`: Mảng chứa các giá trị bên phải của các ràng buộc.
  - o `constraint_signs`: Mảng chứa các dấu của các ràng buộc (1: ">=", 0: "=", -1: "<=").
  - o `variable_signs`: Mảng chứa các dấu của các biến (1: ">= 0", -1: "<= 0", 0: "không ràng buộc dấu").
- Giá trị trả về: Không có giá trị trả về vì đây là hàm khởi tạo.

##### b. Hàm `solve`

```
def solve(self):
    self.convert_to_standard_form()
    tableau = np.zeros((self.num_cons + 1, self.num_vars + self.num_cons +
1))
    tableau = self.initialize_tableau(tableau)
```

```

check = 0

algorithm_choice = self.choose_algorithm()
if algorithm_choice == 0:
    tableau, check = self.dantzig_method(tableau)
elif algorithm_choice == 1:
    tableau, check = self.bland_method(tableau)
else:
    tableau, check = self.two_phase_method(tableau)

return self.process_output(tableau, check)

```

- Công dụng của hàm: Hàm solve thực hiện quá trình giải bài toán quy hoạch tuyến tính bằng cách chuyển đổi nó sang dạng chuẩn, khởi tạo bảng tableau và chọn thuật toán thích hợp để giải.
- Tham số truyền vào self: Tham chiếu đến đối tượng hiện tại của lớp.
- Giá trị trả về: Kết quả của hàm process\_output, một chuỗi HTML chứa thông tin về kết quả giải bài toán.
- Giải thích code:
  - Chuyển đổi sang dạng chuẩn: Gọi phương thức convert\_to\_standard\_form để chuyển đổi bài toán quy hoạch tuyến tính sang dạng chuẩn (standard form).
  - Khởi tạo bảng tableau: Tạo một mảng numpy có kích thước (self.num\_cons + 1, self.num\_vars + self.num\_cons + 1) với tất cả các phần tử bằng 0. Số hàng là self.num\_cons + 1 (số ràng buộc cộng thêm một hàng cho hàm mục tiêu). Số cột là self.num\_vars + self.num\_cons + 1 (số biến cộng thêm số biến slack và một cột cho các giá trị tự do).
  - Khởi tạo nội dung bảng tableau: Gọi phương thức initialize\_tableau để khởi tạo nội dung cho bảng tableau với các hệ số của hàm mục tiêu, các ràng buộc và các biến. Khởi tạo biến kiểm tra: Đặt biến check bằng 0. Biến này sẽ lưu trạng thái kết quả của quá trình giải.
  - Chọn thuật toán: Gọi phương thức choose\_algorithm để chọn thuật toán thích hợp dựa trên các ràng buộc của bài toán. Giá trị trả về của phương thức này được lưu vào biến algorithm\_choice.
  - Giải bài toán bằng thuật toán thích hợp:
    - Nếu algorithm\_choice bằng 0, gọi phương thức dantzig\_method để giải bài toán bằng phương pháp Dantzig.
    - Nếu algorithm\_choice bằng 1, gọi phương thức bland\_method để giải bài toán bằng phương pháp Bland.
    - Nếu algorithm\_choice không phải 0 hoặc 1, gọi phương thức two\_phase\_method để giải bài toán bằng phương pháp hai pha.
  - Xử lý và trả về kết quả: Gọi phương thức process\_output để xử lý bảng tableau cuối cùng và trạng thái check, trả về kết quả dưới dạng chuỗi HTML chứa thông tin về kết quả giải bài toán.

#### c. Hàm display

```

def display(self):
    objective_function = ""
    if self.is_min:

```

```

        objective_function = "Min z = "
    else:
        objective_function = "Max z = "

    for j in range(self.num_vars):
        coefficient = self.obj_coeffs[j]
        if coefficient >= 0 and j != 0:
            objective_function += " + " + str(coefficient) + "x" + str(j + 1) +
1)
        else:
            objective_function += str(coefficient) + "x" + str(j + 1)

    print(objective_function)

    for i in range(self.num_cons):
        constraint = ""
        for j in range(self.num_vars):
            coefficient = self.constraint_matrix[i, j]
            if coefficient >= 0 and j != 0:
                constraint += " + " + str(coefficient) + "x" + str(j + 1)
            else:
                constraint += str(coefficient) + "x" + str(j + 1)

        if self.constraint_signs[i] == 1:
            constraint += ">= "
        elif self.constraint_signs[i] == 0:
            constraint += "= "
        else:
            constraint += "<= "

        constraint += str(self.constraint_rhs[i])
        print(constraint)

    for j in range(self.num_vars):
        variable = "x" + str(j + 1)
        if self.variable_signs[j] == 1:
            variable += " >= 0"
        elif self.variable_signs[j] == -1:
            variable += " <= 0"
        else:
            break
        print(variable)

```



- Công dụng của hàm: Hàm display được dùng để hiển thị bài toán quy hoạch tuyến dưới dạng có thể đọc được. Nó in ra hàm mục tiêu, các ràng buộc và điều kiện của các biến.
- Tham số truyền vào self: Tham chiếu đến đối tượng hiện tại của lớp.
- Giá trị trả về: Hàm này không trả về giá trị. Nó in trực tiếp các thông tin ra màn hình.
- Giải thích code:
  - Khởi tạo hàm mục tiêu: Khởi tạo chuỗi objective\_function để lưu hàm mục tiêu. Nếu bài toán là tối thiểu hóa, chuỗi sẽ bắt đầu bằng "Min z = ". Nếu là tối đa hóa, chuỗi sẽ bắt đầu bằng "Max z = ".
  - Xây dựng hàm mục tiêu:
    - Duyệt qua từng hệ số của biến trong hàm mục tiêu (self.obj\_coeffs).
    - Nếu hệ số lớn hơn hoặc bằng 0 và không phải là hệ số đầu tiên, thêm "+ coefficient x(j+1)" vào chuỗi objective\_function.
    - Nếu là hệ số đầu tiên hoặc nhỏ hơn 0, chỉ thêm "coefficient x(j+1)".
  - In hàm mục tiêu: In chuỗi objective\_function ra màn hình.
  - Xây dựng các ràng buộc:
    - Duyệt qua từng ràng buộc (self.num\_cons).
    - Trong mỗi ràng buộc, duyệt qua từng hệ số của biến (self.constraint\_matrix).
    - Nếu hệ số lớn hơn hoặc bằng 0 và không phải là hệ số đầu tiên, thêm "+ coefficient x(j+1)" vào chuỗi constraint.
    - Nếu là hệ số đầu tiên hoặc nhỏ hơn 0, chỉ thêm "coefficient x(j+1)".
  - Thêm dấu bất đẳng thức hoặc dấu bằng vào ràng buộc:
    - Nếu dấu của ràng buộc là 1, thêm ">= " vào chuỗi constraint.
    - Nếu là 0, thêm "= ".
    - Nếu là -1, thêm "<= ".
  - Thêm giá trị tự do và in ràng buộc:
    - Thêm giá trị tự do của ràng buộc (self.constraint\_rhs) vào chuỗi constraint.
    - In chuỗi constraint ra màn hình.
  - Hiển thị điều kiện của các biến:
    - Duyệt qua từng biến (self.num\_vars).
    - Tạo chuỗi variable với tên biến ("x" + số thứ tự).
    - Nếu dấu của biến là 1, thêm ">= 0" vào chuỗi variable.
    - Nếu dấu là -1, thêm "<= 0".
    - Nếu dấu khác 1 hoặc -1, dùng vòng lặp.
    - In chuỗi variable ra màn hình.

d. Hàm convert\_to\_standard\_form

```
def convert_to_standard_form(self):
    # Objective Function
    if not self.is_min:
        self.obj_coeffs = -self.obj_coeffs

    # Variable Signs
    for i in range(self.num_vars - self.num_new_vars):
        if self.variable_signs[i] == -1:
```

```

        self.obj_coeffs[i] = -self.obj_coeffs[i]
        self.constraint_matrix[:, i] = -self.constraint_matrix[:, i]
    elif self.variable_signs[i] == 0:
        self.num_vars += 1
        self.num_new_vars += 1
        self.variable_signs = np.append(self.variable_signs, 0)
        self.obj_coeffs = np.append(self.obj_coeffs, -
self.obj_coeffs[i])
        self.constraint_matrix =
np.concatenate((self.constraint_matrix, -np.array([self.constraint_matrix[:,
i]]).T), axis=1)

# Constraint Signs
for i in range(self.num_cons):
    if self.constraint_signs[i] == 1:
        self.constraint_matrix[i] = -self.constraint_matrix[i]
        self.constraint_rhs[i] = -self.constraint_rhs[i]
        self.constraint_signs[i] = -1
    elif self.constraint_signs[i] == 0:
        self.num_cons += 1
        self.constraint_signs[i] = -1
        self.constraint_signs = np.append(self.constraint_signs, -1)
        self.constraint_matrix =
np.concatenate((self.constraint_matrix, -
np.array([self.constraint_matrix[i]])), axis=0)
        self.constraint_rhs = np.append(self.constraint_rhs, -
self.constraint_rhs[i])

```

- Công dụng của hàm: Hàm `convert_to_standard_form` được sử dụng để chuyển đổi bài toán quy hoạch tuyến tính về dạng chuẩn. Dạng chuẩn này yêu cầu hàm mục tiêu là hàm tối thiểu, tất cả các biến đều không âm và tất cả các ràng buộc đều là bất đẳng thức nhỏ hơn hoặc bằng.
  - o Tham số truyền vào `self`: Tham chiếu đến đối tượng hiện tại của lớp.
- Giá trị trả về: Hàm này không trả về giá trị. Nó thực hiện chuyển đổi trên các thuộc tính của đối tượng hiện tại.
- Giải thích code:
  - o Chuyển đổi hàm mục tiêu: Nếu bài toán không phải là bài toán tối thiểu hóa (`self.is_min` là `False`), thì đổi dấu tất cả các hệ số trong hàm mục tiêu (`self.obj_coeffs`). Điều này biến bài toán tối đa hóa thành bài toán tối thiểu hóa.
  - o Chuyển đổi dấu của các biến:
    - Duyệt qua từng biến.
    - Nếu biến có dấu âm (`self.variable_signs[i] == -1`), đổi dấu hệ số tương ứng trong hàm mục tiêu và đổi dấu các hệ số tương ứng trong ma trận ràng buộc (`self.constraint_matrix`).
    - Chuyển đổi biến tự do thành hai biến không âm:

- Nếu biến là tự do (`self.variable_signs[i] == 0`), thêm một biến mới vào mô hình.
- Tăng số lượng biến (`self.num_vars`) và số lượng biến mới (`self.num_new_vars`).
- Thêm dấu 0 cho biến mới vào danh sách dấu biến (`self.variable_signs`).
- Thêm hệ số âm của hệ số hiện tại vào hàm mục tiêu (`self.obj_coeffs`).
- Thêm cột hệ số âm của biến hiện tại vào ma trận ràng buộc (`self.constraint_matrix`).
- Chuyển đổi dấu của các ràng buộc:
  - Duyệt qua từng ràng buộc.
  - Nếu ràng buộc là lớn hơn hoặc bằng (`self.constraint_signs[i] == 1`), đổi dấu toàn bộ hệ số trong ràng buộc và đổi dấu giá trị tự do (`self.constraint_rhs`). Sau đó, đổi dấu của ràng buộc thành nhỏ hơn hoặc bằng (`self.constraint_signs[i] = -1`).
- Chuyển đổi ràng buộc bằng thành hai ràng buộc nhỏ hơn hoặc bằng:
  - Nếu ràng buộc là bằng (`self.constraint_signs[i] == 0`), tăng số lượng ràng buộc (`self.num_cons`).
  - Đổi dấu của ràng buộc thành nhỏ hơn hoặc bằng (`self.constraint_signs[i] = -1`).
  - Thêm một ràng buộc mới với dấu âm của ràng buộc hiện tại vào danh sách dấu ràng buộc (`self.constraint_signs`).
  - Thêm hàng hệ số âm của ràng buộc hiện tại vào ma trận ràng buộc (`self.constraint_matrix`).
  - Thêm giá trị tự do âm của ràng buộc hiện tại vào danh sách giá trị tự do (`self.constraint_rhs`).

e. Hàm `print_table`

```
def print_table(self, tableau):
    print(tableau)
```

- Công dụng: In tableau ra màn hình.
- Tham số truyền vào: tableau, Bảng đơn hình.
- Giá trị trả về: Không có

f. Hàm `choose_pivot_dantzig`

```
def choose_pivot_dantzig(self, tableau, xPivot, yPivot, phase1):
    minC = 0
    yPivot = -1
    for i in range(tableau.shape[1] - 1):
        if (tableau[0, i] < 0) and (tableau[0, i] < minC):
            minC = tableau[0, i]
            yPivot = i
    if yPivot == -1:
        return xPivot, yPivot, 0
    xPivot = self.find_arg_min_ratio(tableau, yPivot, phase1)
```

```

if xPivot == -1:
    return xPivot, yPivot, -1
return xPivot, yPivot, 1

```

- Công dụng: Hàm `choose_pivot_dantzig` chọn trục pivot cho phương pháp Dantzig trong bài toán quy hoạch tuyến tính. Nó duyệt qua các cột để tìm cột có giá trị nhỏ nhất (âm nhất) trong hàng đầu tiên, sau đó tìm hàng tương ứng với tỷ lệ nhỏ nhất. Nếu tìm thấy trục pivot hợp lệ, hàm trả về chỉ số hàng và cột của trục pivot cùng với giá trị trạng thái là 1. Nếu không, nó trả về các giá trị và trạng thái thích hợp (0 hoặc -1) để chỉ ra rằng bài toán đã tối ưu hoặc không giới hạn.
- Tham số truyền vào:
  - o `self`: Tham chiếu đến đối tượng hiện tại của lớp.
  - o `tableau`: Ma trận tableau hiện tại của bài toán quy hoạch tuyến tính.
  - o `xPivot`: Chỉ số hàng hiện tại của trục pivot.
  - o `yPivot`: Chỉ số cột hiện tại của trục pivot.
  - o `phase1`: Boolean xác định nếu bài toán đang trong giai đoạn 1 của phương pháp hai pha (two-phase method).
- Giá trị trả về:
  - o `xPivot`: Chỉ số hàng của trục pivot được chọn.
  - o `yPivot`: Chỉ số cột của trục pivot được chọn.
  - o Một giá trị chỉ trạng thái:
    - 0 nếu không tìm thấy cột pivot phù hợp (nghĩa là tối ưu hoặc không có lời giải).
    - -1 nếu không tìm thấy hàng pivot phù hợp (nghĩa là bài toán không giới hạn).
    - 1 nếu tìm thấy cả hàng và cột pivot phù hợp.
- Giải thích từng đoạn code:
  - o Khởi tạo giá trị:
    - `minC`: Lưu giá trị nhỏ nhất của hệ số trong hàng đầu tiên của tableau.
    - `yPivot`: Chỉ số cột của trục pivot, ban đầu được đặt là -1 (không hợp lệ).
  - o Tìm cột pivot:
    - Duyệt qua các cột của hàng đầu tiên trong bảng tableau (trừ cột cuối cùng chứa giá trị RHS).
    - Tìm giá trị nhỏ nhất (âm nhất) trong hàng đầu tiên.
    - Cập nhật `minC` và `yPivot` với giá trị và chỉ số cột tương ứng.
  - o Kiểm tra nếu không tìm thấy cột pivot hợp lệ: Nếu không có cột nào thỏa mãn điều kiện (tất cả các giá trị trong hàng đầu tiên đều không âm), trả về `xPivot`, `yPivot` và 0 (nghĩa là bài toán đã tối ưu).
  - o Tìm hàng pivot: Gọi hàm `find_arg_min_ratio` để tìm hàng pivot `xPivot` dựa trên cột pivot `yPivot` đã chọn và xem xét nếu đang trong giai đoạn 1 của phương pháp hai pha.
  - o Kiểm tra nếu không tìm thấy hàng pivot hợp lệ: Nếu không có hàng nào thỏa mãn điều kiện (nghĩa là bài toán không giới hạn), trả về `xPivot`, `yPivot` và -1.
  - o Trả về kết quả: Nếu tìm thấy cả hàng và cột pivot hợp lệ, trả về `xPivot`, `yPivot` và 1.

g. Hàm `dantzig_method`

```

def dantzig_method(self, tableau, phase1=False):
    xPivot, yPivot = -1, -1

```

```

while True:
    xPivot, yPivot, check = self.choose_pivot_dantzig(tableau, xPivot,
yPivot, phase1)
    if check == 1:
        tableau, xPivot, yPivot = self.rotate_pivot(tableau, xPivot,
yPivot)
    else:
        return tableau, -check

```

- Công dụng: Hàm dantzig\_method thực hiện việc xoay bảng đơn hình bằng cách liên tục chọn và xoay trục xoay theo phương pháp Dantzig cho đến khi đạt được nghiệm tối ưu hoặc xác định rằng bài toán không có nghiệm tối ưu. Các chỉ số trục xoay (xPivot, yPivot) và bảng đơn hình (tableau) được cập nhật trong mỗi bước xoay. Khi không còn trục xoay hợp lệ, hàm trả về bảng đơn hình cuối cùng và trạng thái của bài toán.
- Tham số truyền vào
  - o tableau: Bảng đơn hình hiện tại (ma trận chứa các hệ số của hàm mục tiêu và các ràng buộc).
  - o phase1 (mặc định là False): Biến boolean xác định liệu hàm có đang được gọi trong pha 1 của phương pháp hai pha hay không.
- Giá trị trả về
  - o tableau: Bảng đơn hình sau khi thực hiện các bước xoay trục xoay và đạt đến trạng thái kết thúc (tìm được nghiệm tối ưu hoặc xác định rằng không có nghiệm tối ưu).
  - o -check: Trạng thái của bài toán sau khi kết thúc:
    - 0: Đã tìm được nghiệm tối ưu.
    - 1: Bài toán không có nghiệm tối ưu (không có giới hạn trên hoặc dưới).
    - 2: Có thể có các trường hợp khác dựa trên việc triển khai cụ thể của choose\_pivot\_dantzig.
  - o Giải thích code:
    - xPivot và yPivot được khởi tạo là -1. Đây là các chỉ số của hàng và cột trục xoay, ban đầu được đặt là -1 để chỉ ra rằng chưa có trục xoay nào được chọn.
    - while True: Vòng lặp vô hạn, sẽ tiếp tục cho đến khi gặp điều kiện dừng bên trong.
    - xPivot, yPivot, check = self.choose\_pivot\_dantzig(tableau, xPivot, yPivot, phase1): Gọi hàm choose\_pivot\_dantzig để chọn trục xoay tiếp theo. Hàm này trả về chỉ số hàng (xPivot), chỉ số cột (yPivot), và trạng thái (check):
      - 1: Nếu tìm thấy trục xoay hợp lệ.
      - Khác 1: Nếu không tìm thấy trục xoay hợp lệ hoặc xác định rằng bài toán không có nghiệm tối ưu.
    - if check == 1: Nếu tìm thấy trục xoay hợp lệ:
      - tableau, xPivot, yPivot = self.rotate\_pivot(tableau, xPivot, yPivot): Gọi hàm rotate\_pivot để thực hiện xoay bảng đơn hình tại trục xoay được chọn. Hàm này trả về bảng đơn hình đã xoay và cập nhật các chỉ số trục xoay.
    - else: Nếu không tìm thấy trục xoay hợp lệ:

- return tableau, -check: Trả về bảng đơn hình và trạng thái kết thúc, đảo dấu của check để biểu thị trạng thái dừng của bài toán.

#### h. Hàm choose\_pivot\_bland

```
def choose_pivot_bland(self, tableau, xPivot, yPivot):
    yPivot = -1
    for i in range(tableau.shape[1] - 1):
        if tableau[0, i] < 0:
            yPivot = i
            break
    if yPivot == -1:
        return xPivot, yPivot, 0
    xPivot = self.find_arg_min_ratio(tableau, yPivot, False)
    if xPivot == -1:
        return xPivot, yPivot, -1
    return xPivot, yPivot, 1
```

- Công dụng: Hàm choose\_pivot\_bland thực hiện việc chọn trục xoay tiếp theo trong bảng đơn hình theo quy tắc Bland. Nó duyệt qua các cột để tìm cột có hệ số nhỏ hơn 0, sau đó tìm hàng có tỷ lệ nhỏ nhất giữa giá trị bên phải và hệ số trong cột đó. Nếu tìm thấy trục xoay hợp lệ, hàm trả về chỉ số hàng và cột của trục xoay cùng với trạng thái 1. Nếu không, hàm trả về trạng thái tương ứng chỉ ra rằng không có trục xoay hợp lệ.
- Tham số truyền vào
  - o tableau: Bảng đơn hình hiện tại (ma trận chứa các hệ số của hàm mục tiêu và các ràng buộc).
  - o xPivot: Chỉ số hàng của trục xoay hiện tại.
  - o yPivot: Chỉ số cột của trục xoay hiện tại.
- Giá trị trả về
  - o xPivot: Chỉ số hàng của trục xoay được chọn.
  - o yPivot: Chỉ số cột của trục xoay được chọn.
  - o check: Trạng thái của việc chọn trục xoay:
    - 1: Nếu tìm thấy trục xoay hợp lệ.
    - 0: Nếu không tìm thấy trục xoay hợp lệ (nghĩa là không có biến nào để xoay).
    - -1: Nếu không tìm thấy chỉ số hàng hợp lệ cho trục xoay (nghĩa là không có tỷ lệ nhỏ nhất).
- Giải thích code:
  - o yPivot được khởi tạo là -1. Đây là chỉ số cột của trục xoay, ban đầu được đặt là -1 để chỉ ra rằng chưa có cột nào được chọn.
  - o Vòng lặp for duyệt qua các cột của bảng đơn hình (trừ cột cuối cùng chứa các giá trị bên phải của ràng buộc).
    - Nếu tìm thấy một cột i có hệ số trong hàm mục tiêu (hàng đầu tiên của bảng đơn hình) nhỏ hơn 0, chỉ số i sẽ được gán cho yPivot, và vòng lặp dừng lại (break). Điều này có nghĩa là biến tương ứng với cột i là ứng viên để làm trục xoay.

- Nếu không tìm thấy cột nào có hệ số nhỏ hơn 0, yPivot vẫn là -1, và hàm trả về xPivot, yPivot, và 0, chỉ ra rằng không có trục xoay hợp lệ.
- Gọi hàm `find_arg_min_ratio` để tìm chỉ số hàng xPivot của trục xoay dựa trên cột yPivot vừa chọn. Hàm này sẽ tìm hàng có tỷ lệ nhỏ nhất giữa giá trị bên phải và hệ số trong cột yPivot. Nếu không tìm thấy hàng hợp lệ, xPivot sẽ được gán giá trị -1.
  - Nếu xPivot là -1, hàm trả về xPivot, yPivot, và -1, chỉ ra rằng không tìm thấy chỉ số hàng hợp lệ cho trục xoay.
  - Nếu tìm thấy cả chỉ số hàng xPivot và chỉ số cột yPivot hợp lệ, hàm trả về xPivot, yPivot, và 1, chỉ ra rằng đã tìm thấy trục xoay hợp lệ.

i. Hàm `bland_method`

```
def bland_method(self, tableau):
    xPivot, yPivot = -1, -1
    while True:
        xPivot, yPivot, check = self.choose_pivot_bland(tableau, xPivot,
yPivot)
        if check != 1:
            return tableau, -check
        else:
            tableau, xPivot, yPivot = self.rotate_pivot(tableau, xPivot,
yPivot)
    return tableau, 0
```

- Công dụng: Hàm `bland_method` giải bài toán quy hoạch tuyến tính bằng phương pháp đơn hình sử dụng quy tắc Bland. Nó liên tục chọn trục xoay và xoay bảng đơn hình cho đến khi tìm được nghiệm hoặc xác định rằng bài toán không có nghiệm. Nếu không tìm thấy trục xoay hợp lệ hoặc bài toán không có nghiệm, hàm trả về trạng thái tương ứng. Nếu tìm được nghiệm tối ưu, hàm trả về bảng đơn hình đã giải và trạng thái 0.
- Tham số truyền vào:
  - `tableau`: Bảng đơn hình hiện tại (ma trận chứa các hệ số của hàm mục tiêu và các ràng buộc).
- Giá trị trả về
  - `tableau`: Bảng đơn hình sau khi áp dụng phương pháp đơn hình theo quy tắc Bland.
  - `check`: Trạng thái của việc giải bài toán:
    - 0: Nếu tìm được nghiệm tối ưu.
    - 1: Nếu bài toán không giới hạn.
    - -1: Nếu không có nghiệm (do không tìm được chỉ số hàng hợp lệ cho trục xoay).
- Giải thích code:
  - `xPivot` và `yPivot` được khởi tạo là -1. Đây là chỉ số hàng và cột của trục xoay, ban đầu được đặt là -1 để chỉ ra rằng chưa có trục xoay nào được chọn.
  - Vòng lặp `while True` bắt đầu một vòng lặp vô hạn để liên tục chọn trục xoay và xoay bảng đơn hình cho đến khi tìm được nghiệm hoặc xác định rằng bài toán không có nghiệm.
  - Gọi hàm `choose_pivot_bland` để chọn trục xoay tiếp theo theo quy tắc Bland. Hàm này trả về chỉ số hàng (`xPivot`), chỉ số cột (`yPivot`), và trạng thái (`check`).

- Nếu trạng thái check không bằng 1 (tức là không tìm thấy trục xoay hợp lệ hoặc bài toán không có nghiệm), hàm trả về bảng đơn hình hiện tại và trạng thái -check.
- Nếu check là 0, tức là không có cột nào có hệ số nhỏ hơn 0 trong hàm mục tiêu, bài toán đã tìm được nghiệm tối ưu.
- Nếu check là -1, tức là không tìm thấy chỉ số hàng hợp lệ cho trục xoay, bài toán không có nghiệm.
- Nếu check bằng 1 (tức là tìm thấy trục xoay hợp lệ), gọi hàm rotate\_pivot để thực hiện phép xoay trên bảng đơn hình. Hàm này cập nhật bảng đơn hình và trả về bảng đơn hình đã xoay cùng với chỉ số hàng và cột của trục xoay mới.

j. Hàm find\_pivot\_column

```
def find_pivot_column(self, tableau, col):
    xPivot = -1
    flag = False
    for i in range(1, tableau.shape[0]):
        if tableau[i, col] == 0:
            continue

        if tableau[i, col] == 1:
            if flag is False:
                xPivot = i
                flag = True
            else:
                return -1
        else:
            return -1

    return xPivot
```

- Công dụng: Hàm find\_pivot\_column tìm kiếm hàng duy nhất trong cột đã chỉ định của bảng đơn hình mà có giá trị 1 và các hàng khác đều có giá trị 0. Nếu tìm thấy, nó trả về chỉ số của hàng đó. Nếu không, hàm trả về -1, cho biết không có hàng phù hợp. Điều này giúp xác định biến cơ bản trong phương pháp đơn hình.
- Tham số truyền vào
  - o tableau: Bảng đơn hình hiện tại, dạng ma trận numpy chứa các hệ số của hàm mục tiêu và các ràng buộc.
  - o col: Chỉ số cột cần kiểm tra để tìm trục xoay.
- Giá trị trả về:
  - o xPivot: Chỉ số hàng của trục xoay trong cột đã chỉ định. Nếu không tìm thấy hàng phù hợp, hàm trả về -1.
- Giải thích code
  - o xPivot được khởi tạo là -1, đại diện cho việc chưa có hàng nào được chọn.
  - o flag được khởi tạo là False để đánh dấu xem đã tìm thấy một hàng có giá trị 1 hay chưa.
  - o Nếu giá trị tại cột col của hàng i bằng 1:



- Nếu flag là False (chưa tìm thấy hàng nào có giá trị 1 trước đó), gán xPivot là i và đặt flag là True.
- Nếu flag là True (đã tìm thấy một hàng có giá trị 1 trước đó), trả về -1 vì có nhiều hơn một hàng có giá trị 1 trong cùng cột.
- Nếu giá trị tại cột col của hàng i không phải là 0 hoặc 1, trả về -1 vì cột này không phù hợp để làm trục xoay.
- Trả về chỉ số hàng của trục xoay (xPivot). Nếu không tìm thấy hàng phù hợp, hàm trả về giá trị mặc định -1.

k. Hàm two\_phase\_method

```
def two_phase_method(self, tableau):
    tableauP1 = np.zeros((tableau.shape[0], tableau.shape[1] + 1))
    tableauP1[0, -2] = 1
    tableauP1[1:, -2] = -np.ones((tableau.shape[0] - 1, 1)).ravel()
    tableauP1[1:, :tableau.shape[1] - 1] = tableau[1:, :tableau.shape[1] - 1]
    tableauP1[1:, -1] = tableau[1:, -1]

    xPivot, yPivot = -1, tableauP1.shape[1] - 2
    minB = 0
    for i in range(tableauP1.shape[0]):
        if tableau[i, yPivot] < minB:
            minB = tableau[i, yPivot]
            xPivot = i
    tableauP1, xPivot, yPivot = self.rotate_pivot(tableauP1, xPivot, yPivot)
    tableauP1, check = self.dantzig_method(tableauP1, 1)

    for j in range(tableauP1.shape[1] - 2):
        if tableauP1[0, j] != 0:
            return tableau, -1 # No solution

    # Phase 2
    tableau[1:, :tableau.shape[1] - 1] = tableauP1[1:, :tableau.shape[1] - 1]
    tableau[1:, -1] = tableauP1[1:, -1]

    for j in range(tableau.shape[1]):
        xPivot = self.find_pivot_column(tableau, j)
        if xPivot == -1:
            continue
        tableau, xPivot, j = self.rotate_pivot(tableau, xPivot, j)

    tableau, check = self.dantzig_method(tableau)
```

```

        return tableau, check

    def check_unique_solution(self, tableau, pivots):
        for i in range(tableau.shape[1] - 1):
            if (i >= self.num_vars - self.num_new_vars) and (i < self.num_vars):
                continue
            if ((pivots[i] == -1) and (abs(tableau[0, i]) < 1e-6)) and
(self.variable_signs[i] != 0):
                return False
        return True

```

- Công dụng: Hàm `two_phase_method` thực hiện phương pháp hai giai đoạn để giải bài toán quy hoạch tuyến tính. Phương pháp này bao gồm hai giai đoạn:
    - o Giai đoạn 1: Tìm giải pháp khả thi ban đầu nếu không có sẵn.
    - o Giai đoạn 2: Tối ưu hóa giải pháp tìm được từ giai đoạn 1 để đạt được giá trị tối ưu của hàm mục tiêu.
  - Tham số truyền vào
    - o `tableau`: Bảng đơn hình hiện tại, là một ma trận numpy chứa các hệ số của hàm mục tiêu và các ràng buộc.
  - Giá trị trả về
    - o `tableau`: Bảng đơn hình cuối cùng sau khi thực hiện phương pháp hai giai đoạn.
    - o `check`: Trạng thái kết quả của phương pháp:
      - 0: Giải pháp tối ưu được tìm thấy.
      - -1: Không có giải pháp khả thi.
  - Giải thích code:
    - o Tạo `tableauP1`, bảng đơn hình mới để thực hiện giai đoạn 1 với kích thước thêm một cột.
    - o Cột mới được thêm vào với các giá trị: hàng đầu tiên đặt 1 và các hàng còn lại đặt -1.
    - o Sao chép các giá trị từ bảng `tableau` vào `tableauP1`, ngoại trừ hàm mục tiêu ban đầu.
    - o Khởi tạo `xPivot` và `yPivot` để xác định vị trí trục xoay.
    - o Tìm hàng có giá trị nhỏ nhất trong cột mới thêm vào để chọn trục xoay ban đầu.
    - o Thực hiện phép quay trục tại vị trí tìm được.
    - o Thực hiện phương pháp Dantzig để tìm giải pháp khả thi ban đầu.
    - o Kiểm tra nếu bất kỳ biến nào khác không bằng 0, kết luận không có giải pháp khả thi và trả về -1.
    - o Sao chép các giá trị từ `tableauP1` vào `tableau` để chuẩn bị cho giai đoạn 2.
    - o Tìm các cột để thực hiện phép quay trục và tối ưu hóa giải pháp.
    - o Thực hiện phương pháp Dantzig một lần nữa để tối ưu hóa giải pháp và trả về kết quả.
1. Hàm `check_unique_solution`

```

def check_unique_solution(self, tableau, pivots):
    for i in range(tableau.shape[1] - 1):
        if (i >= self.num_vars - self.num_new_vars) and (i < self.num_vars):
            continue

```

```

        if ((pivots[i] == -1) and (abs(tableau[0, i]) < 1e-6)) and
(self.variable_signs[i] != 0):
            return False
        return True

```

- Công dụng của hàm
  - o Hàm check\_unique\_solution kiểm tra xem giải pháp tối ưu có duy nhất hay không.
- Tham số truyền vào
  - o tableau: Bảng đơn hình hiện tại, là một ma trận numpy chứa các hệ số của hàm mục tiêu và các ràng buộc.
  - o pivots: Danh sách các chỉ số trục xoay cho mỗi biến.
- Giá trị trả về
  - o True: Nếu giải pháp là duy nhất.
  - o False: Nếu giải pháp không duy nhất.
  - o Duyệt qua các cột của bảng đơn hình, bỏ qua các cột tương ứng với các biến mới được thêm vào.
  - o Kiểm tra nếu một biến không là biến cơ bản (pivot bằng -1), giá trị của nó trong hàm mục tiêu rất nhỏ (gần 0), và dấu của biến không phải là 0.
  - o Nếu đúng, giải pháp không là duy nhất, trả về False. Nếu không có điều kiện nào vi phạm, trả về True, xác nhận giải pháp là duy nhất.

#### m. Hàm find\_variable\_name

```

def find_variable_name(self, tableau, index):
    name = ""
    if index < self.num_vars - self.num_new_vars:
        name = "x" + str(index + 1)
        return 1, name
    elif (index + 1 > self.num_vars) and (index + 1 < tableau.shape[1]):
        name = "w" + str(index + 1 - self.num_vars)
        return 1, name
    return 0, name

```

- Công dụng: Hàm find\_variable\_name xác định tên của một biến trong bài toán quy hoạch tuyến tính dựa trên chỉ số của biến đó trong bảng đơn hình.
- Tham số truyền vào
  - o tableau: Bảng đơn hình hiện tại, là một ma trận numpy chứa các hệ số của hàm mục tiêu và các ràng buộc.
  - o index: Chỉ số của biến cần tìm tên.
- Giá trị trả về
  - o name: Tên của biến được xác định từ chỉ số truyền vào.
  - o Một cặp giá trị gồm:
    - 1: Nếu biến được tìm thấy và tên được xác định.
    - 0: Nếu không xác định được tên của biến.
- Giải thích code:
  - o Khởi tạo biến name là chuỗi rỗng. Đây sẽ là biến chứa tên của biến sau khi xác định.

- Nếu index nhỏ hơn số lượng biến ban đầu trừ đi số biến mới được thêm vào, xác định đây là một biến ban đầu (biến gốc) trong bài toán.
- Tên của biến sẽ là "x" kèm theo chỉ số của biến (tăng thêm 1 vì chỉ số bắt đầu từ 0).
- Trả về 1 kèm tên của biến, cho biết biến này đã được xác định.
- Nếu index + 1 lớn hơn số lượng biến ban đầu (`self.num_vars`) và nhỏ hơn tổng số cột trong tableau, xác định đây là một biến bổ sung (biến w) trong bài toán.
- Tên của biến sẽ là "w" kèm theo chỉ số của biến (trừ đi số lượng biến ban đầu).
- Trả về 1 kèm tên của biến, cho biết biến này đã được xác định.
- Nếu không rơi vào hai trường hợp trên, trả về 0 kèm tên (vẫn là chuỗi rỗng), cho biết biến này không được xác định tên.

n. Hàm `initialize_tableau`

```
def initialize_tableau(self, tableau):
    tableau[0, :self.num_vars] = self.obj_coeffs
    tableau[0, -1] = 0
    tableau[1:, :self.num_vars] = self.constraint_matrix
    tableau[1:, self.num_vars:-1] = np.identity(self.num_cons)
    tableau[1:, -1] = self.constraint_rhs

    self.variable_signs = np.append(self.variable_signs,
np.ones(self.num_cons))
    return tableau
```

- Công dụng của hàm
  - Hàm `initialize_tableau` khởi tạo bảng đơn hình (tableau) dùng trong phương pháp đơn hình để giải bài toán quy hoạch tuyến tính. Bảng này sẽ chứa các hệ số của hàm mục tiêu, các ràng buộc, và các biến bổ sung cần thiết.
- Tham số truyền vào
  - `tableau`: Bảng đơn hình ban đầu, là một ma trận numpy sẽ được khởi tạo và trả về sau khi được thiết lập.
- Giá trị trả về
  - `tableau`: Bảng đơn hình sau khi đã được khởi tạo với các hệ số của hàm mục tiêu và các ràng buộc
- Giải thích code
  - Dòng đầu tiên của bảng đơn hình (dòng 0) được gán các hệ số của hàm mục tiêu (`self.obj_coeffs`). Các hệ số này tương ứng với các biến quyết định trong bài toán.
  - Phần tử cuối cùng của dòng đầu tiên (giá trị tự do của hàm mục tiêu) được gán bằng 0. Điều này thường biểu diễn giá trị của hàm mục tiêu tại điểm bắt đầu.
  - Các dòng còn lại của bảng đơn hình (từ dòng 1 trở đi) được gán các hệ số của ma trận ràng buộc (`self.constraint_matrix`). Mỗi dòng tương ứng với một ràng buộc tuyến tính trong bài toán.
  - Phần tiếp theo của các dòng (từ vị trí `self.num_vars` đến vị trí kế cuối) được gán bằng ma trận đơn vị (`np.identity(self.num_cons)`). Điều này thường biểu diễn các biến bổ sung để chuyển các ràng buộc bất đẳng thức thành đẳng thức.

- Phần tử cuối cùng của các dòng (giá trị tự do của các ràng buộc) được gán bằng các giá trị của vế phải của các ràng buộc (`self.constraint_rhs`).
- Cập nhật các dấu hiệu của các biến bằng cách thêm vào một mảng các giá trị 1 (biểu diễn các biến bổ sung). Điều này giúp theo dõi trạng thái của các biến trong bài toán.
- Trả về bảng đơn hình đã được khởi tạo đầy đủ.

o. Hàm `process_output`

```
def process_output(self, tableau, result):
    print("Tableau:", tableau)
    print("Result:", result)
    output = "<hr><h1>RESULT</h1><hr>"

    if result == 1:
        if self.is_min:
            output += " => The problem is <b>UNBOUNDED</b>. <br> MIN z = -<br>" + str(np.inf) + "</b> <br>"
        else:
            output += " => The problem is <b>UNBOUNDED</b>. <br> MAX z = +<br>" + str(np.inf) + "</b> <br>"
    elif result == 0:
        if self.is_min:
            output += "<u>MIN z = <b>" + str(-tableau[0, -1]) + "</b></u><br>"
        else:
            output += "<u>MAX z = <b>" + str(tableau[0, -1]) + "</b></u><br>"

    pivots = np.array([self.find_pivot_column(tableau, i) for i in range(tableau.shape[1] - 1)])
    if self.check_unique_solution(tableau, pivots):
        output += '<b> => UNIQUE SOLUTION.</b> The optimal solution is: <br>'

    for j in range(self.num_vars - self.num_new_vars):
        if tableau[0, j] != 0:
            output += f"x<sub>{j + 1}</sub> = 0<br>"
            continue
        count = 0
        index = 0
        for i in range(1, tableau.shape[0]):
            if tableau[i, j] != 0:
                count += 1
                index = i
        if self.variable_signs[j] == -1:
            output += f"x<sub>{j + 1}</sub> = {-tableau[index, -1]}<br>"
```

```

        else:
            output += f"x<sub>{j + 1}</sub> = {tableau[index, -
1]]}<br>"
        else:
            output += "<b>=> MULTIPLE SOLUTIONS.</b> <br>"
            output += "The optimal solution set is: <br>"
            sign = np.array([-1 if ((self.variable_signs[i] < 0) & (i <
self.num_vars - self.num_new_vars)) else 1 for i in range(tableau.shape[1] -
1)])
            for i in range(self.num_vars - self.num_new_vars):
                if pivots[i] == -1:
                    if abs(tableau[0, i]) > 1e-4:
                        output += f"x<sub>{i + 1}</sub> = 0<br>"
                    else:
                        if self.variable_signs[i] == 0:
                            output += f"x<sub>{i + 1}</sub> is free<br>"
                        elif self.variable_signs[i] == 1:
                            output += f"x<sub>{i + 1}</sub> >= 0<br>"
                        else:
                            output += f"x<sub>{i + 1}</sub> <= 0<br>"
                else:
                    root = f"x<sub>{i + 1}</sub> = {sign[i] *
tableau[pivots[i], -1]} <br>"
                    for j in range(tableau.shape[1] - 1):
                        if ((abs(tableau[0, j]) > 1e-4) | (pivots[j] != -
1)) | (j == i):
                            continue
                        check_root, name = self.find_variable_name(tableau,
j)
                        if check_root == 0:
                            continue
                        else:
                            if -sign[i] * sign[j] * tableau[pivots[i], j]
== 0:
                                continue
                            if -sign[i] * sign[j] * tableau[pivots[i], j]
> 0:
                                root += f"+ {-sign[i] * sign[j] *
tableau[pivots[i], j]}{name} <br>"
                            else:
                                root += f"{-sign[i] * sign[j] *
tableau[pivots[i], j]}{name} <br>"
                        output += root
                    output += "With:<br>"
                    for i in range(tableau.shape[1] - 1):

```



```

else:
    output += "<b> => NO SOLUTION</b>.<br>"
    print(output)
    return output

```

- Công dụng của hàm: Hàm process\_output xử lý và định dạng kết quả từ bảng đơn hình (tableau) của bài toán quy hoạch tuyến tính sau khi chạy thuật toán đơn hình. Nó sẽ in bảng đơn hình và kết quả ra màn hình, đồng thời tạo một chuỗi HTML chứa thông tin chi tiết về kết quả tối ưu, bao gồm giá trị của hàm mục tiêu và các giá trị của biến quyết định.
  - Tham số truyền vào
    - o tableau: Bảng đơn hình sau khi đã thực hiện xong thuật toán Simplex.
    - o result: Kết quả của thuật toán Simplex, có thể là:
      - 1: Bài toán không bị chặn (unbounded).
      - 0: Bài toán có nghiệm tối ưu.
      - Các giá trị khác: Bài toán không có nghiệm.
  - Giá trị trả về
    - o output: Chuỗi HTML chứa thông tin chi tiết về kết quả bài toán quy hoạch tuyến tính.
- p. Hàm choose\_algorithm

```

def choose_algorithm(self):
    flag = 0
    for i in range(self.num_cons):
        if self.constraint_rhs[i] == 0: # Bland
            flag = 1
        if self.constraint_rhs[i] < 0: # Two-phase
            return 2
    return flag

```

- Công dụng của hàm: Hàm choose\_algorithm xác định thuật toán nào sẽ được sử dụng để giải bài toán quy hoạch tuyến tính, dựa trên các điều kiện trong các ràng buộc của bài toán.
- Tham số truyền vào
  - o Hàm này không nhận tham số nào ngoài self, là tham chiếu đến đối tượng hiện tại của lớp.
- Giá trị trả về
  - o 2: Nếu có ít nhất một ràng buộc với vế phải âm (self.constraint\_rhs[i] < 0).
  - o 1: Nếu có ít nhất một ràng buộc với vế phải bằng không (self.constraint\_rhs[i] == 0) và không có ràng buộc nào với vế phải âm.
  - o 0: Nếu không có ràng buộc nào với vế phải bằng không hoặc âm.
- Giải thích code
  - o Khởi tạo biến flag với giá trị 0. Biến này sẽ lưu trữ giá trị trả về, trừ khi gặp điều kiện đặc biệt.
  - o Bắt đầu vòng lặp qua tất cả các ràng buộc (self.num\_cons là số lượng ràng buộc).
    - Nếu vế phải của ràng buộc thứ i bằng không (self.constraint\_rhs[i] == 0), đặt flag thành 1. Điều này gợi ý rằng thuật toán Bland có thể được sử dụng.



- Nếu vế phải của ràng buộc thứ  $i$  nhỏ hơn không ( $\text{self.constraint\_rhs}[i] < 0$ ), trả về 2 ngay lập tức. Điều này cho biết cần sử dụng thuật toán hai giai đoạn ( $\text{two\_phase\_method}$ ) vì có ràng buộc âm.
- Sau khi kiểm tra tất cả các ràng buộc, trả về giá trị của  $\text{flag}$ . Nếu  $\text{flag}$  là 1, điều đó nghĩa là có ít nhất một ràng buộc với vế phải bằng không. Nếu  $\text{flag}$  vẫn là 0, nghĩa là tất cả các ràng buộc đều có vế phải dương và thuật toán chuẩn ( $\text{dantzig\_method}$ ) có thể được sử dụng.

q. Hàm `rotate_pivot`

```
def rotate_pivot(self, tableau, xPivot, yPivot):
    for i in range(tableau.shape[0]):
        if i != xPivot:
            coef = -tableau[i, yPivot] / tableau[xPivot, yPivot]
            tableau[i, :] += coef * tableau[xPivot, :]
        else:
            coef = tableau[xPivot, yPivot]
            tableau[xPivot, :] /= coef
    return tableau, xPivot, yPivot
```

- Công dụng của hàm: Hàm `rotate_pivot` thực hiện các phép biến đổi hàng trên bảng `tableau` để đưa bài toán về dạng chuẩn tắc, giúp cho phương pháp đơn hình tiến tới nghiệm tối ưu. Các phép biến đổi này bao gồm:
  - Đưa các phần tử ngoài hàng pivot của cột pivot về 0.
  - Chuẩn hóa hàng pivot để phần tử pivot trở thành 1.
- Tham số truyền vào
  - `self`: Tham chiếu đến đối tượng hiện tại của lớp.
  - `tableau`: Ma trận `tableau` đại diện cho bài toán quy hoạch tuyến tính.
  - `xPivot`: Chỉ số hàng của phần tử pivot.
  - `yPivot`: Chỉ số cột của phần tử pivot.
- Giá trị trả về
  - `tableau`: Ma trận `tableau` sau khi đã thực hiện phép xoay vòng trục.
  - `xPivot`: Chỉ số hàng của phần tử pivot (không thay đổi).
  - `yPivot`: Chỉ số cột của phần tử pivot (không thay đổi).
- Giải thích:
  - Bắt đầu vòng lặp qua tất cả các hàng của bảng `tableau`.
  - Nếu hàng hiện tại không phải là hàng pivot ( $i \neq \text{xPivot}$ ), thực hiện phép biến đổi hàng để đưa các phần tử ngoài hàng pivot của cột pivot về 0.
  - Tính hệ số `coef` bằng cách lấy phần tử ở cột pivot của hàng hiện tại chia cho phần tử pivot. Sau đó, thực hiện phép biến đổi hàng: hàng hiện tại được cộng với `coef` nhân với hàng pivot. Điều này đảm bảo rằng phần tử ở cột pivot của hàng hiện tại sẽ trở về 0.
  - Nếu hàng hiện tại là hàng pivot ( $i == \text{xPivot}$ ), chuẩn hóa hàng pivot bằng cách chia tất cả các phần tử trong hàng pivot cho phần tử pivot. Điều này làm cho phần tử pivot trở thành 1.
  - Trả về bảng `tableau` sau khi đã thực hiện phép xoay vòng trục, cùng với chỉ số hàng và cột của phần tử pivot.

r. Hàm `find_arg_min_ratio`

```

def find_arg_min_ratio(self, tableau, yPivot, phase1):
    i = 0
    xPivot = -1
    minRatio = -1
    ratio = 0
    for i in range(tableau.shape[0]):
        if tableau[i, yPivot] > 0:
            minRatio = tableau[i, -1] / tableau[i, yPivot]
            xPivot = i
            break
    if xPivot == -1:
        return -1
    for i in range(1, tableau.shape[0]):
        if tableau[i, yPivot] > 0:
            ratio = tableau[i, -1] / tableau[i, yPivot]
            if ratio < minRatio:
                minRatio = ratio
                xPivot = i
            if phase1 is True:
                if (ratio == minRatio) and (tableau[i, -2] == 1):
                    xPivot = i
    return xPivot

```

- Công dụng của hàm: Hàm `find_arg_min_ratio` được sử dụng để tìm chỉ số hàng của phần tử pivot dựa trên tỉ số tối thiểu (minimum ratio). Đây là một bước quan trọng trong quá trình chọn phần tử pivot trong phương pháp đơn hình.
- Tham số truyền vào
  - o `self`: Tham chiếu đến đối tượng hiện tại của lớp.
  - o `tableau`: Ma trận tableau đại diện cho bài toán quy hoạch tuyến tính.
  - o `yPivot`: Chỉ số cột của phần tử pivot.
  - o `phase1`: Biến boolean cho biết có đang ở giai đoạn 1 của phương pháp hai pha (two-phase method) hay không.
- Giá trị trả về
  - o `xPivot`: Chỉ số hàng của phần tử pivot có tỉ số tối thiểu, hoặc -1 nếu không tìm thấy phần tử pivot hợp lệ.
- Giải thích code:
  - o Khởi tạo các biến `i`, `xPivot`, `minRatio`, và `ratio`. `xPivot` lưu trữ chỉ số hàng của phần tử pivot. `minRatio` lưu trữ giá trị tỉ số tối thiểu. `ratio` lưu trữ giá trị tỉ số tại mỗi bước.
  - o Duyệt qua từng hàng của bảng tableau. Nếu phần tử tại cột `yPivot` của hàng `i` lớn hơn 0, tính tỉ số giữa phần tử ở cột cuối cùng (cột giá trị tự do) và phần tử tại cột `yPivot`. Gán giá trị này cho `minRatio` và gán chỉ số hàng `i` cho `xPivot`. Thoát khỏi vòng lặp sau khi tìm thấy phần tử hợp lệ đầu tiên.
    - Nếu không tìm thấy phần tử hợp lệ (không có phần tử nào tại cột `yPivot` lớn hơn 0), trả về -1.

- Duyệt qua từng hàng từ hàng thứ hai trở đi (bỏ qua hàng đầu tiên). Nếu phần tử tại cột yPivot của hàng i lớn hơn 0, tính tỉ số giữa phần tử ở cột cuối cùng và phần tử tại cột yPivot. Nếu tỉ số này nhỏ hơn minRatio, cập nhật minRatio và xPivot với giá trị và chỉ số hàng mới.
  - Nếu đang ở giai đoạn 1 (phase1 là True) và tỉ số bằng minRatio, kiểm tra nếu phần tử tại cột thứ hai từ cuối của hàng i bằng 1, cập nhật xPivot với chỉ số hàng i.
- Trả về chỉ số hàng của phần tử pivot có tỉ số tối thiểu. Nếu không có phần tử hợp lệ nào được tìm thấy, trả về -1.

## 2. Mô tả cách sử dụng

- Bước 1: Truy cập vào trang web
  - Mở trình duyệt và truy cập vào địa chỉ <https://hcmus-linear-programming.onrender.com/>
- Bước 2: Nhập số lượng biến (Number variables):
  - Giao diện sẽ hiển thị các ô nhập liệu như sau:
    - Number variables: Nhập số lượng biến quyết định của bài toán. Ví dụ, nếu bài toán có hai biến  $x$  và  $y$ , nhập số "2" vào ô này.
    - Number constraints: Nhập số lượng ràng buộc của bài toán. Ví dụ, nếu bài toán có ba ràng buộc, nhập số "3" vào ô này.
  - Sau khi nhập xong, nhấn nút "Submit" để tiếp tục.
- Bước 3: Nhập hàm mục tiêu (Target Function)
  - Chọn loại hàm mục tiêu: Ở ô thả xuống "Target function", chọn "Minimize" nếu bạn muốn tối thiểu hóa hoặc "Maximize" nếu bạn muốn tối đa hóa hàm mục tiêu.
  - Nhập hệ số cho các biến:
    - Ô đầu tiên sau "X1": Nhập hệ số của biến  $x_1$ . Ví dụ, nếu hàm mục tiêu là  $x_1 + 2x_2$ , nhập "1".
    - Ô thứ hai sau "X2": Nhập hệ số của biến  $x_2$ . Ví dụ, nếu hàm mục tiêu là  $x_1 + 2x_2$ , nhập "2".

**HNP**

### Linear Programming Problem Input

Number variables:

Number constraints:

#### Target function

Target function Maximize  X1 +  X2

#### Constraints

Constraint 1  X1 +  X2 =

Constraint 2  X1 +  X2 =

#### Variable constraints

X1 free

X2 free

- Bước 4: Nhập các ràng buộc (Constraints):
  - Nhập dữ liệu cho từng ràng buộc theo các ô nhập liệu tương ứng:

- Ô đầu tiên sau "X1": Nhập hệ số của biến  $x_1$  trong ràng buộc đầu tiên. Ví dụ, nếu ràng buộc là  $x_1 + 2x_2 \leq 9$ , nhập "1".
- Ô thứ hai sau "X2": Nhập hệ số của biến  $x_2$  trong ràng buộc đầu tiên. Ví dụ, nếu ràng buộc là  $x_1 + 2x_2 \leq 9$ , nhập "2".
- Ô thả xuống: Chọn dấu bất đẳng thức hoặc dấu bằng ( $\leq, =, \geq$ ) phù hợp với ràng buộc. Ví dụ, chọn  $\leq$
- Ô cuối cùng: Nhập giá trị hằng số bên phải của ràng buộc. Ví dụ, nếu ràng buộc là  $x_1 + 2x_2 \leq 9$ , nhập "9".
- Tương tự với ràng buộc còn lại.

HNP

## Linear Programming Problem Input

Number variables:

Number constraints :

### Target function

Target function   X1 +  X2

### Constraints

Constraint 1  X1 +  X2

Constraint 2  X1 +  X2

### Variable constraints

X1

X2

- Bước 5: Nhập ràng buộc cho các biến (Variable Constraints)
  - Ràng buộc cho  $x_1$ , Chọn giá trị trong ô thả xuống bên cạnh  $x_1$ :
    - "free" nếu  $x_1$  là biến tự do.
    - " $\geq$ " nếu  $x_1$  không âm.
    - " $\leq$ " nếu  $x_1$  không dương.
  - Ràng buộc cho  $x_2$ , Chọn giá trị trong ô thả xuống bên cạnh  $x_2$ :
    - "free" nếu  $x_2$  là biến tự do.
    - " $\geq$ " nếu  $x_2$  không âm.

- " $\leq$ " nếu  $x_2$  không dương.

**HNP**

## Linear Programming Problem Input

Number variables:

Number constraints :

### Target function

Target function   X1 +  X2

### Constaints

Constaint 1  X1 +  X2

Constaint 2  X1 +  X2


### Variable constaints

X1

X2

- Bước 6: Nhấn nút "Submit" để hệ thống giải bài toán. Hệ thống sẽ trả về kết quả như sau
- Bước 7: Xem kết quả. Hệ thống sẽ xử lý và hiển thị kết quả của bài toán, bao gồm:
  - Phương pháp giải (Method): Ở ví dụ này bài toán được giải theo phương pháp đơn hình (Dantzig).
  - Danh sách bảng (Tableau list): Ở mục này sẽ hiển thị các danh sách các bước lặp trong quá trình xoay bảng, ở mỗi bước lặp (Iteration) sẽ có một bảng giá trị để người dùng có thể đối chiếu khi thực hiện giải tay trên giấy.
  - Kết quả (Result): Nghiệm tối ưu và giá trị tối ưu (nếu có). Ở trong ví dụ này giá trị tối ưu là  $\max z = 6$  và điểm tối ưu là  $(0, 3)$ .

- Kết quả trả về từ hệ thống:

 **HNP**

**Method: Dantzig method**

**Tableau list:**  
**Iteration 1:**

**[-1. -2. 0. 0. 0.]**

**[1. 2. 1. 0. 9.]**

**[1. 1. 0. 1. 3.]**

 **HNP**

**[-1. -2. 0. 0. 0.]**

**[1. 1. 0. 1. 3.]**

---

**RESULT**

---

**MAX z = 6.0**

**⇒ UNIQUE SOLUTION.** The optimal solution is:

**$x_1 = 0$**

**$x_2 = 3.0$**

- *Lưu ý:*

- Trong quá trình nhập dữ liệu hãy điền đầy đủ các thông tin, nếu không điền đầy đủ thông tin khi ấn nút “Submit” sẽ hiện thông báo như sau:

The screenshot shows a web browser window with the URL `hcmus-linear-programming.onrender.com`. The page title is "HNP". A modal error message is displayed at the top, stating: "hcmus-linear-programming.onrender.com says Invalid input in constraint 1, variable 1. Please enter a number." Below the error message, the form fields are visible: "Number variables: 2", "Number constraints: 2", "Target function: Minimize 1 X1 + 2 X2", "Constraint 1: 1 X1 + 1 X2 ≤ 1", "Constraint 2: 1 X1 + 2 X2 ≤ 1", "Variable constraints: X1 ≥ 0, X2 ≥ 0", and a "Submit" button.

- Sau khi sử dụng, nếu muốn nhập một bài toán khác, hãy truy cập lại hay tải lại trang website để nhập thêm bài toán mới.

## IV. Ví dụ

### 1. Thuật toán đơn hình

- Xét một bài toán như sau:

$$\max z = 2x_1 + x_2$$

$$x_1 + x_2 \leq 12$$

$$x_1 + 2x_2 \leq 8$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

- Ta sẽ nhập vào website bài toán trên như sau:

# Linear Programming Problem Input

Number variables:

Number constraints :

## Target function

Target function   X1 +  X2

## Constraints

Constraint 1  X1 +  X2

Constraint 2  X1 +  X2

## Variable constraints

X1

X2

- Kết quả thu được như sau:



## Method: Dantzig method

Tableau list:  
Iteration 1:

$[-2. \ -1. \ 0. \ 0. \ 0.]$

$[ \ 1. \ 1. \ 1. \ 0. \ 12.]$

$[1. \ 2. \ 0. \ 1. \ 8.]$

---

## RESULT

---

**MAX  $z = 16.0$**

**$\Rightarrow$  UNIQUE SOLUTION.** The optimal solution is:

**$x_1 = 8.0$**

**$x_2 = 0$**

- Với bài toán trên hệ thống đề xuất giải bằng phương pháp đơn hình (Dantzig). Lần lượt bên dưới là bảng giá trị sau mỗi lần xoay. Kết quả cuối cùng thu được, bài toán có nghiệm duy nhất, điểm tối ưu là  $(x_1, x_2) = (8, 0)$  với giá trị tối ưu là  $\max z = 16$ .

## 2. Thuật toán Bland

- Xét bài toán:

$$\min x_1 + 2x_2$$

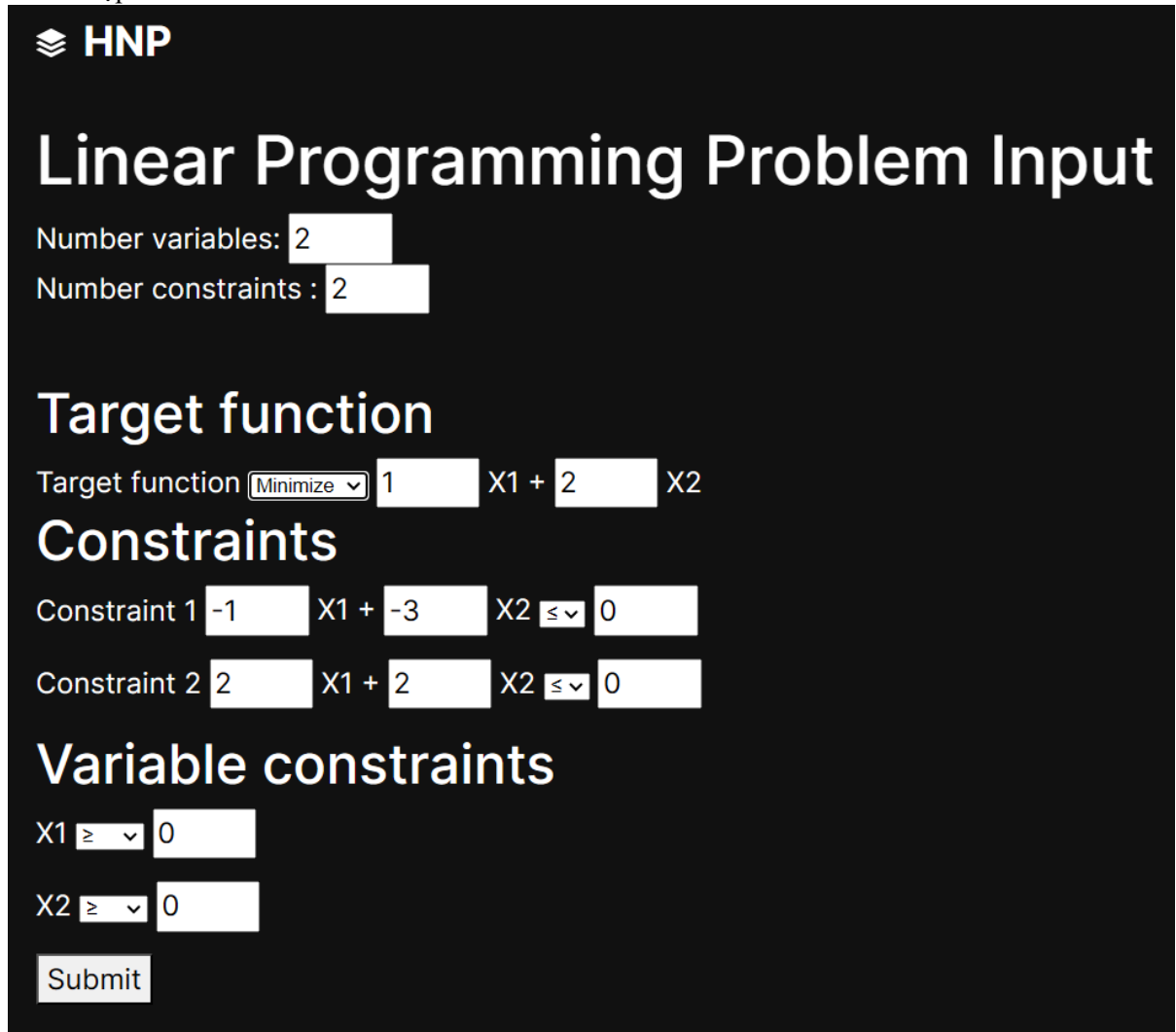
$$-x_1 - 3x_2 \leq 0$$

$$2x_1 + 2x_2 \geq 0$$

$$x_1 \geq 0$$

$$x_2 \geq 0$$

- Ta sẽ nhập vào website bài toán trên như sau:



**HNP**

# Linear Programming Problem Input

Number variables:

Number constraints :

## Target function

Target function   X1 +  X2

## Constraints

Constraint 1  X1 +  X2

Constraint 2  X1 +  X2

## Variable constraints

X1

X2

- Kết quả thu được như sau:

Method: Bland method

Tableau list:  
Iteration 1:

[1. 2. 0. 0. 0.]

[-1. -3. 1. 0. 0.]

[2. 2. 0. 1. 0.]

---

RESULT

---

MIN  $z = -0.0$

⇒ **UNIQUE SOLUTION.** The optimal solution is:

$$x_1 = 0$$

$$x_2 = 0$$

- Với bài toán trên hệ thống đề xuất giải bằng phương pháp Bland. Lần lượt bên dưới là bảng giá trị sau mỗi lần xoay. Kết quả cuối cùng thu được, bài toán có nghiệm duy nhất, điểm tối ưu là  $(x_1, x_2) = (0, 0)$  với giá trị tối ưu là  $\max z = 0$ .

### 3. Thuật toán hai pha

- Xét bài toán như sau:

$$\max 3x_1 + x_2$$

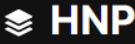
$$-x_1 + x_2 \geq 1,$$

$$-x_1 - x_2 \leq -3$$

$$2x_1 + x_2 \leq 4$$

$$x_1, x_2 \geq 0$$

- Nhập bài toán vào website như sau:



## Linear Programming Problem Input

Number variables:

Number constraints :

### Target function

Target function Maximize  X1 +  X2

### Constraints

Constraint 1  X1 +  X2 ≥

Constraint 2  X1 +  X2 ≤

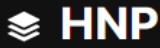
Constraint 3  X1 +  X2 ≤

### Variable constraints

X1 ≥

X2 ≥

- Kết quả thu được như sau:



## Method: Two-phase method

### Tableau list:

#### Iteration 1:

$[-1. -1. 0. 1. 0. 0. -3.]$

$[ 1. -1. 1. 0. 0. -1. -1.]$

$[-1. -1. 0. 1. 0. -1. -3.]$

$[ 2. 1. 0. 0. 1. -1. 4.]$

#### Iteration 2:

### RESULT

MAX  $z = 5.0$   
⇒ **UNIQUE SOLUTION.** The optimal solution is:  
 $x_1 = 1.0$   
 $x_2 = 2.0$

- Với bài toán trên hệ thống đề xuất giải bằng phương pháp hai pha. Lần lượt bên dưới là bảng giá trị sau mỗi lần xoay. Kết quả cuối cùng thu được, bài toán có nghiệm duy nhất, điểm tối ưu là  $(x_1, x_2) = (1, 2)$  với giá trị tối ưu là  $\max z = 5$ .

## V. Tài liệu tham khảo

- Phan Quốc Khánh, Trần Huệ Nương. (2003). Quy hoạch tuyến tính. Nhà xuất bản Giáo dục, Nhà in Thanh Niên, 62 Trần Huy Liệu, Q.PN, TP.HCM.
- Khalibartan, "simplex.py", Simplex Method, GitHub, truy cập ngày 10 tháng 5 năm 2024, <https://github.com/khalibartan/simplex-method/blob/master/simplex.py>.
- Vanderbei, R. J. (2021). Linear Programming: Foundations and Extensions. Cham: Springer.