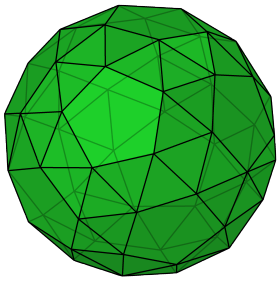
Projects in Mathematics and Applications

Phương pháp điểm trong Interior Point Method

Ngày 8 tháng 8 năm 2024

Phạm Quốc Bình* †Bảo Quý Định Tân
Nguyễn Trần Hà Phương ‡ §Lê Minh Quý



*Trường THPT Thống Nhất B - Đồng Nai
†Trường THPT chuyên Lê Quý Đôn - Khánh Hòa
‡Wilbraham and Monson Academy
§Trường THPT Gia Định

Lời cảm ơn

Lời đầu tiên, chúng em xin gửi lời cám ơn chân thành nhất đến với các nhà sáng lập của trại hè, anh Cấn Trần Thành Trung, anh Trần Hoàng Bảo Linh, anh Lê Việt Hải cùng ban tổ chức PiMA đã xây dựng, duy trì và phát triển chương trình 8 năm liên tục để chúng em có cơ hội trải nghiệm trại hè Toán học và ứng dụng bổ ích và thú vị.

Bên cạnh đó, chúng xin cám ơn anh, chị hướng dẫn đứng lớp giảng dạy kiến thức về Đại số tuyến tính, Vi tích phân và Quy hoạch tuyến tính trong phần đầu của trại. Sự nhiệt huyết của anh, chị đã tiếp thêm năng lượng cho chúng em nỗ lực hết mình trong thời gian trại hè diễn ra.

Tiếp đến, chúng em muốn gửi lời cảm ơn đặc biệt đến các anh hướng dẫn, anh Trần Phan Anh Danh, anh Nguyễn Hoàng Khang và anh Vũ Lê Thế Anh đã theo dõi và giúp đỡ rất tận tình nhóm trong quá trình hoàn thành dự án. Đề tài của chúng em sẽ không thể hoàn thiện nếu thiếu đi sự giúp đỡ của các anh.

Chúng em xin cám ơn trường Đại học Khoa học Tự nhiên, ĐHQG-HCM đã tài trợ phòng học và trang thiết bị để chúng em có thể học tập trong điều kiện tốt nhất có thể. Tụi em rất vinh dự khi được tham gia trại hè tại một trong những trường đại học nghiên cứu hàng đầu ở Việt Nam.

Trong những ngày hè vừa rồi, được đồng hàng cùng các bạn trại sinh và các anh chị hướng dẫn từ khắp mọi miền tổ quốc, từ khắp nơi trên thế giới, chúng em xin cám ơn mọi người vì những trải nghiệm đầy ý nghĩa và khó quên này. Hi vọng trong những năm tới, PiMA sẽ tiếp tục phát triển và truyền nguồn cảm hứng và động lực cho những bạn học sinh THPT khác.

Xin cám ơn.

Ngày 8 tháng 8 năm 2024.
Nhóm 6.

Tóm tắt nội dung

Báo cáo này tập trung nghiên cứu và ứng dụng phương pháp điểm trong, đặc biệt là phiên bản primal-dual, trong giải quyết bài toán quy hoạch tuyến tính. Phương pháp điểm trong là một công cụ, khác biệt so với phương pháp đơn hình truyền thống nhờ việc di chuyển bên trong miền nghiệm của bài toán. Từ đó, phương pháp này giảm thiểu số bước để tìm ra nghiệm tối ưu.

Trong báo cáo, chúng tôi trình bày những cơ sở lý thuyết của phương pháp điểm trong, cài đặt và so sánh với các thuật toán khác để chứng minh độ hiệu quả của phương pháp này. Ở phần cuối cùng, báo cáo cũng trình bày ứng dụng thực tiễn của phương pháp điểm trong qua bài toán vận tải và minh họa độ hiệu quả của thuật toán thông qua các ví dụ cụ thể.

Mục lục

Danh pháp tiếng Anh

Những thuật ngữ được dùng trong bài báo cáo vào danh pháp tiếng Anh của chúng:

1. Bài toán quy hoạch tuyến tính: Linear Programming Problem
2. Bài toán chắn: Barrier Problem
3. Biến: Variables
4. Đường trung tâm: Central Path
5. Điểm cực đại: Maximum Point
6. Điểm tối ưu: Optimal Point
7. Giá trị tối ưu: Optimal Value
8. Hàm chắn logarithm: Logarithmic Barrier Function
9. Hàm Lagrange: Lagrangian Function
10. Hàm mục tiêu: Objective Function
11. Hệ điều kiện Karush-Kuhn-Tucker (KKT): Karush-Kuhn-Tucker (KKT) Conditions
12. Ma trận: Matrix
13. Nghiệm: Root
14. Phương pháp giải tích: Analytical Method
15. Phương pháp điểm trong: Interior Point Method
16. Phương pháp Newton: Newton Method
17. Ràng buộc tuyến tính: Linear Constraints
18. Trọng số: Weight
19. Tập nghiệm chấp nhận được: Feasible Region
20. Thuật toán: Algorithm
21. Bài toán vận tải: Transportation Problem

1 Giới thiệu

1.1 Bài toán quy hoạch tuyến tính

Bài toán quy hoạch tuyến tính là một công cụ quan trọng để giải quyết các bài toán tối ưu hóa. Nó được sử dụng để tìm giá trị tối ưu của một hàm mục tiêu tuyến tính, đồng thời phải thỏa những ràng buộc tuyến tính đi kèm.

Bài toán quy hoạch tuyến tính ở dạng chuẩn có dạng

$$\begin{aligned} \text{Maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad \forall\, i = 1, \ldots, m \\ & x_j \geq 0, \quad \forall\, j = 1, \ldots, n \end{aligned}$$

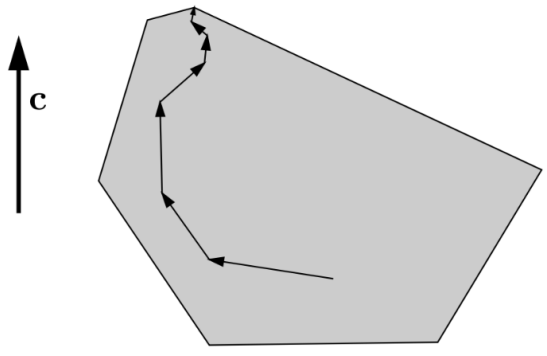
Trong đó

- $\sum_{j=1}^{n} c_j x_j$ là hàm mục tiêu
- $\sum_{j=1}^{n} a_{ij} x_j \leq b_i$ và $x_j \geq 0$ là những ràng buộc

1.2 Phương pháp điểm trong

Phương pháp điểm trong là một phương pháp tiên tiến để giải quyết bài toán quy hoạch tuyến tính. Khác với phương pháp đơn hình - di chuyển từ đỉnh này qua đỉnh khác trên biên của tập nghiệm khả thi của bài toán - phương pháp điểm trong di chuyển bên trong của tập nghiệm này. Điều này giúp giảm thiểu số bước để tìm ra nghiệm tối ưu của bài toán, từ đó xử lý các bài toán quy hoạch tuyến tính lớn và phức tạp. Đó cũng là ưu điểm của phương pháp điểm trong so với các phương pháp khác như phương pháp đơn hình hay phương pháp elip.

Hướng tiếp cận cơ bản của phương pháp điểm trong là di chuyển trong miền nghiệm của bài toán và từ từ tiếp cận đến nghiệm tối ưu. Trong khi di chuyển, nó tránh khỏi biên của miền nghiệm tạo bởi các ràng buộc của bài toán gốc, từ đó tránh những vấn đề liên quan đến biên của bài toán, điều mà phương pháp đơn hình hay gặp phải. Đây chính là điểm nổi bật của phương pháp điểm trong.



Hình 1: Hình minh họa về cách tiếp cận của phương pháp điểm trong

Phương pháp điểm trong được giới thiệu lần đầu vào năm 1984 bởi Narendra Kramarkar, một nhà nghiên cứu ở IBM. Kể từ đó, nhiều phiên bản và cải tiến của phương pháp này đã được phát triển nhằm tăng cường độ hiệu quả của nó trong nhiều ứng dụng thực tế. Trong bài báo cáo này, chúng tôi tập trung nghiên cứu vào phương pháp đường trung tâm, vì nó là phương pháp mang lại hiệu quả về mặt tính toán tốt nhất. Đồng thời, chúng tôi tập trung nghiên cứu phiên bản primal-dual, vì nó kết hợp thông tin của cả bài toán gốc và bài toán đối ngẫu, rất quan trọng trong việc giải quyết những bài toán lớn.

2 Thiết kế thuật toán

2.1 Bài toán chắn

Trước hết, giả sử ta có bài toán quy hoạch tuyến tính ở dạng chuẩn như sau:

$$\begin{aligned} &\text{Maximize } c^\intercal x \\ &\text{Subject to } Ax \leq b \\ &x \geq 0 \end{aligned}$$

Và tương ứng ta có bài toán đối ngẫu là:

$$\begin{aligned} &\text{Minimize } b^\intercal y \\ &\text{Subject to } A^\intercal y \geq c \\ &y \geq 0 \end{aligned}$$

Ta sẽ thêm các biến bù w và z tương ứng cho hai bài toán trên để đưa về dạng chính tắc, với bài toán gốc trở thành:

$$\begin{aligned} &\text{Maximize } c^\intercal x \\ &\text{Subject to } Ax + w = b \\ &x, w \geq 0 \end{aligned} \tag{1}$$

Tương tự, bài toán đối ngẫu khi đó là:

$$\begin{aligned} &\text{Minimize } b^\intercal x \\ &\text{Subject to } A^\intercal y - z = c \\ &y, z \geq 0 \end{aligned}$$

Ta mong muốn là có thể loại bỏ đi điều kiện ràng buộc về dấu bất đẳng thức của các biến trong (1), và ta có thể làm như vậy bằng cách xét một bài toán với hàm mục tiêu khác, chẳng hạn như:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \sum_{i=1}^{n} \mathrm{I}(x_i) + \sum_{i=1}^{m} \mathrm{I}(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{2}$$

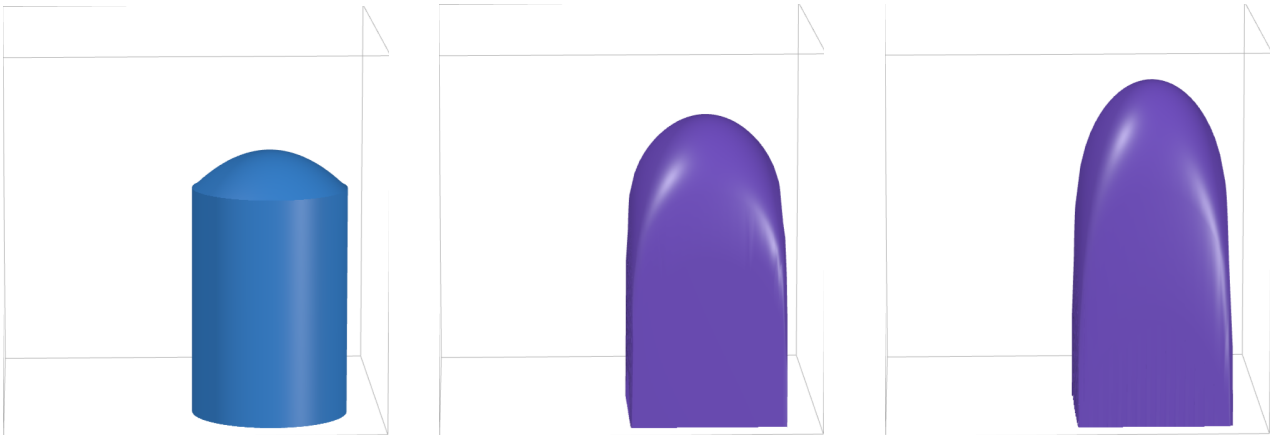
Trong đó hàm $I(x)$ được định nghĩa như sau:

$$I(x) = \begin{cases} -\infty & , x < 0 \\ 0 & , x \geq 0 \end{cases}$$

Xét bài toán (2), ta nhận thấy nếu x, $w \geq 0$ thì bài toán này sẽ trở thành bài toán (1), trái lại nếu tồn tại một thành phần nào đó trong các vector x, vector w nhỏ hơn 0 thì hàm mục tiêu ngay lập tức tiến đến $-\infty$ trái với việc ta đang cực đại hóa hàm mục tiêu. Do đó bài toán (2) hàm chứa bài toán (1), nhưng các ràng buộc về dấu của x và w đã được loại bỏ.
Nhưng với hàm như vậy thì rất khó để ta có thể nghiên cứu được bài toán, vì hàm mục tiêu không liên tục cho nên không tồn tại đạo hàm trên $\mathbb{R}$, do đó ta không thể áp dụng trực tiếp các phương pháp giải tích để nghiên cứu.
Tuy nhiên, ta xét một lựa chọn khác như sau:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{i=1}^{m} \ln(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{3}$$

trong đó $\mu > 0$ là một tham số thực.
Hàm mục tiêu mới này của ta có hàm logarithm vốn là một hàm có đạo hàm nhiều cấp, giúp ta dễ dàng áp dụng trực tiếp giải tích để nghiên cứu. Mặt khác hàm mục tiêu như trên đã bao gồm cả điều kiện các biến $x, w > 0$ và ngăn chặn việc x và w là các điểm bằng không, vì lẽ giả sử tồn tại x_i tiến đến 0 thì hàm mục tiêu sẽ tiến dần đến $-\infty$ đi ngược lại với việc cực đại hóa hàm mục tiêu. Vậy đối với hàm mục tiêu mới này của ta, điểm cực đại của mỗi bài toán tương ứng với mỗi μ sẽ luôn nằm bên trong mà không nằm trên hay vượt ra biên nghiệm chấp nhận được. Xem (Hình: 2) để hiểu hơn về các hàm mục tiêu chứa $I(x)$ và logarithm.



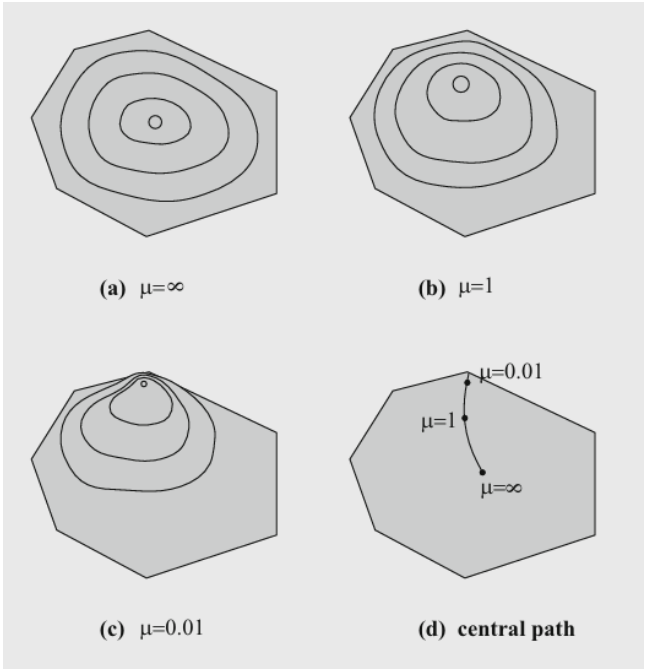
(a) Hàm mục tiêu chứa I(x) (b) $\mu = 0.1$ (c) $\mu = 0.2$

Hình 2: Ở **hình a** là hình minh họa cho hàm mục tiêu có chứa hàm $I(x)$, và **hình b**, **hình c** là hình minh họa cho hàm mục tiêu có hàm logarithm tương ứng với $\mu = 0.1$ và $\mu = 0.2$. Ở cả ba hình thì vùng nghiệm chấp nhận được đại diện là một hình tròn, phần tô màu đại diện cho giá trị của hàm mục tiêu, giá trị hàm mục tiêu càng bé khi càng ở những điểm càng thấp. Ta nhận thấy xung quanh vùng nghiệm chấp nhận được đối với hàm mục tiêu chứa $I(x)$ là một vách chắn, mà chỉ cần bước qua là sẽ rơi xuống những giá trị cực kì bé. Còn đối với **hình b** và **hình c**, thì khi càng tiến về vùng biên của vùng nghiệm chấp nhận được, sẽ có một độ dốc thoải xuống những miền giá trị sâu hơn nhưng không rơi xuống đột ngột như hàm mục tiêu chứa $I(x)$. Mặt khác với μ có giá trị càng nhỏ ta nhận thấy đỉnh của hàm mục tiêu logarithm càng trở về gần giống với đỉnh của hàm mục tiêu chứa $I(x)$.

Bài toán (3) ở trên được gọi là **bài toán chắn** tương ứng của bài toán (1). Và bài toán (3) ở trên thì không tương đương với bài toán (1). Tương ứng với mỗi μ thì ta sẽ được một bài toán riêng. Tuy không tương đương với bài toán số (1) nhưng nếu ta chọn μ đủ nhỏ thì từ việc giải bài toán số (3) ta có thể xấp xỉ được nghiệm tối ưu của bài toán (1)

Trong bài toán cụ thể số (3) ta đã dùng hàm logarithm để biểu diễn hàm mục tiêu, nhưng sẽ còn có rất nhiều cách chọn hàm khác để tạo ra biểu diễn hàm mục tiêu với cùng mục đích để giới hạn bài toán luôn nằm trong miền nghiệm chấp nhận được. Những hàm như thế được gọi là **hàm chắn**. Hàm được dùng trong bài toán số (3) được gọi là **hàm chắn logarithm**.

Ta có một cách để hình dung hàm chắn như sau, giả sử tập nghiệm chấp nhận được của bài toán là một đa giác. Giá trị hàm mục tiêu sẽ tiến đến $-\infty$ khi tiến đến cạnh của đa giác. Tương ứng với mỗi μ khác nhau ta sẽ tìm được giá trị lớn nhất của hàm mục tiêu đạt tại một điểm nào đó nằm trong đa giác (xem hình 3)



Hình 3: Hình (a), (b), (c) minh họa cho giá trị của hàm mục tiêu trên tập nghiệm chấp nhận được của bài toán. Các giá trị lớn nhất sẽ được đạt tại 1 điểm nào đó nằm trong đa giác. Hình (d) vẽ lại đường đi của những điểm cực trị khi μ thay đổi từ ∞ đến 0

Qua đó cũng cho thấy được tại sao phương pháp ta đang nghiên cứu được gọi là *Phương pháp điểm trong*, vì tương ứng với mỗi μ ở trên ta sẽ xác định được điểm lớn nhất chỉ nằm trong tập nghiệm chấp nhận được. Nhưng với việc μ dần tiến về 0 thì ta sẽ xấp xỉ được giá trị tối ưu của hàm mục tiêu ban đầu. Và nếu ta nối lại các điểm mà tại đó hàm mục tiêu lớn nhất trong các bài toán với μ khác nhau, thì ta sẽ được một đường đi, đường đi đó được gọi là **đường trung tâm**.

2.2 Đường trung tâm

Ý tưởng cơ bản của phương pháp trung tâm là ta xuất phát từ trung tâm miền nghiệm (hoặc một nghiệm chấp nhận được của bài toán) và xác định nghiệm tối ưu của từng bài toán chắn. Sau khi giải nghiệm tối ưu của n bài toán chắn - mỗi bài toán chắn ứng với từng giá trị μ, ta đến nghiệm tối ưu của bài toán gốc. Tập hợp các nghiệm tối ưu của từng bài toán chắn tương ứng với mỗi giá trị μ là đường trung tâm.

Bài toán chắn cần giải quyết của chúng ta là:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{i=1}^{m} \ln(w_i) \\ &\text{Subject to } Ax + w = b \end{aligned} \tag{4}$$

Chúng ta cần tối đa hóa hàm mục tiêu, kèm với điều kiện $Ax + w = b$ và $x, w \geq 0$. Khi đó, bài toán cần giải quyết của chúng ta được biến đổi thành bài toán chắn có dạng

$$\max_x \left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\ x, w > 0 \right)$$

Bên cạnh đó, như đã nêu, đường trung tâm là tập hợp các điểm. Vì thế, ta sẽ sử dụng hàm arg để trả về các giá trị x cần tìm với x là nghiệm tối ưu của từng bài toán chắn.

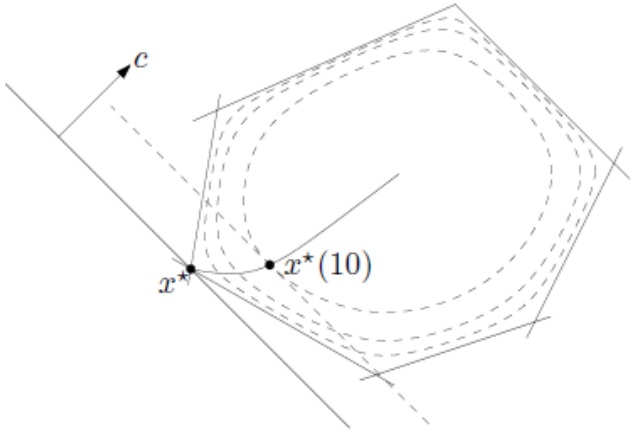
$$\arg\max_x \left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\ x, w > 0 \right)$$

Biểu diễn toán học của đường trung tâm là

$$x^*(\mu) = \arg\max_x \left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax + w = b,\, x, w > 0 \right)$$

Trong đó:

- $c^T x$: Hàm tối ưu của bài toán gốc.
- $Ax = b$: Ràng buộc của bài toán gốc.
- $\sum_{i=1}^{n} \ln(x_i)$ và $\sum_{j=1}^{n} \ln(w_j)$: Hàm chắn
- μ: Trọng số của hàm chắn. Ban đầu, μ rất lớn, tất cả các hàm chắn đồng thời ngăn nghiệm tối ưu bài toán chắn tiến ra biên, xảy ra việc nghiệm tối ưu bài toán chắn khi này nằm ở trung tâm miền nghiệm. Khi μ tiến đến 0^+, thì hàm chắn trở nên cực nhỏ, bài toán chắn sẽ dần tương đương với bài toán gốc. Từ đó, nghiệm tối ưu của hai bài toán sẽ gần tương đương nhau.



Hình 4: Khi μ tiến tới 0, ảnh hưởng của hàm chắn sẽ giảm, dẫn đến việc đi đến giá trị tối ưu dễ dàng hơn. Khi đó, bài toán chắn (4) sẽ dần trở về bài toán (1)

Với mỗi μ, ta tìm x và w để

$$\left(c^T x + \mu \sum_{i=1}^{n} \ln(x_i) + \mu \sum_{j=1}^{n} \ln(w_j) \middle| Ax = b,\, x, w > 0 \right)$$

đạt giá trị lớn nhất. Để tìm nghiệm tối ưu của từng bài toán chắn, dựa trên từng giá trị μ, ta sử dụng phương pháp Newton được trình bày ở Mục 2.4.

2.3 Hàm Lagrange

Để cho thuận tiện, ta sẽ quy ước ma trận x **in thường**:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

Còn ma trận X **in hoa** sẽ là:

$$X = \begin{pmatrix} x_1 & 0 & \cdots & 0 \\ 0 & x_2 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & x_n \end{pmatrix}$$

vậy ma trận X^{-1} sẽ là:

$$X^{-1} = \begin{pmatrix} \frac{1}{x_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{x_2} & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & \frac{1}{x_n} \end{pmatrix}$$

Ngoài ra còn có e kích thước $n \times 1$ là ma trận có dạng:

$$e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

ma trận này sẽ có kích thước tùy thuộc vào vị trí nó được viết nhằm hợp lệ hóa việc nhân ma trận.

Ta có thể chứng minh rằng với mỗi μ, chỉ có một lời giải duy nhất cho bài toán chắn. Và khi μ càng tiến tới 0 thì đáp án của bài toán chắn càng gần với bộ nghiệm của bài toán gốc. Ta xét bài toán chắn:

$$\begin{aligned} &\text{Maximize } c^\intercal x + \mu \sum_j \ln x_j + \mu \sum_i \ln w_i \\ &\text{Subject to } Ax + w = b. \end{aligned} \tag{5}$$

Đây là dạng bài toán tối ưu với với ràng buộc là các phương trình nên ta có thể sử dụng phương pháp nhân tử Lagrange để giải. Hàm Lagrange của bài toán là:

$$L(x, w, y) = c^\intercal x + \mu \sum_j \ln x_j + \mu \sum_i \ln w_i + y^\intercal (b - Ax - w). \tag{6}$$

Vì các điểm dừng của hàm Lagrange cũng sẽ là cực trị địa phương của bài toán ban đầu, nên ta có thể lấy ra được hệ phương trình cho đáp án của bài toán chắn. Xét đạo hàm theo từng biến của hàm Lagrange:

$$\begin{aligned} \frac{\partial L}{\partial x_j} &= c_j + \mu \frac{1}{x_j} - \sum_i y_i a_{ij} = 0, j = 1, 2, \ldots, n, \\ \frac{\partial L}{\partial w_i} &= \mu \frac{1}{w_i} - y_i = 0, i = 1, 2, \ldots, m, \\ \frac{\partial L}{\partial y_i} &= b_i - \sum_j a_{ij} x_j - w_i = 0, i = 1, 2, \ldots, m. \end{aligned} \tag{7}$$

Viết những đạo hàm trên dưới dạng ma trận, ta có hệ phương trình:

$$\begin{aligned} A^T y - \mu X^{-1} e &= c \\ y &= \mu W^{-1} e \\ Ax + w &= b \end{aligned} \tag{8}$$

Đặt $z = \mu X^{-1}e$ với X là ma trận vuông mà mọi số đều bằng 0 và có các biến x lần lượt trên đường chéo chính. Thế z vào, ta sẽ có hệ phương trình:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ z &= \mu X^{-1} e \\ y &= \mu W^{-1} e \end{aligned} \tag{9}$$

Ta nhân X vào hai vế của phương trình phương trình thứ 3, W vào hai vế của phương trình phương trình thứ 4 sẽ có được hệ phương trình:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned} \tag{10}$$

Nhìn vào 2 phương trình đầu của hệ phương trình trên, ta thấy phương trình 1 và 2 lần lượt là điều kiện của bài toán chính ban đầu và bài toán đối ngẫu tương ứng của nó. Còn khi μ nhỏ dần về 0, phương trình 3 và 4 sẽ dần trở về dạng:

$$\begin{aligned} XZ &= 0 \\ YW &= 0 \end{aligned} \tag{11}$$

Hai phương trình này, chú ý về độ lệch bù của bài toán chính và bài toán đối ngẫu, chính là điều kiện của nghiệm tối ưu của bài toán. Vậy nên hệ phương trình (10) có nghiệm là bộ nghiệm vừa tối ưu hàm mục tiêu, vừa thỏa mãn yêu cầu của bài toán ban đầu.

2.4 Thuật toán

2.4.1 Ý tưởng thuật toán

1. Bắt đầu với bộ (x, w, y, z) hợp lệ.
2. Ở mỗi vòng lặp, đi đến điểm mới $(x+\Delta x, w+\Delta w, y+\Delta y, z+\Delta z)$ nằm trên đường trung tâm đến nghiệm tối ưu.
3. Thuật toán gồm 2 pha: Tìm điểm bắt đầu và đi theo đường trung tâm

2.4.2 Đi theo đường trung tâm

1. (Pha I) Gán $\mu = 1$ và chọn bộ $(x, w, y, z) > 0$ thoả mãn hệ (10).
2. (Pha II) Trong khi $\mu \geq \epsilon$, lặp lại bước 3 và 4. Nếu $\mu < \epsilon$, trả x là một nghiệm tối ưu và dừng vòng lặp
3. Thay μ bằng $\left(1 - \frac{1}{2\sqrt{n}}\right)\mu$ [1]
4. (Bước Newton) Tìm $(\Delta x, \Delta w, \Delta y, \Delta z)$ là nghiệm của hệ (10):

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned} \tag{12}$$

5. Thay (x, w, y, z) bằng $(x + \Delta x, w + \Delta w, y + \Delta y, z + \Delta z)$ và lặp lại vòng lặp.

[1] Cách giảm μ này được tham khảo từ [3] mục **7.2 Interior Point Methods**

2.4.3 Pha II

Ở Pha II, bước Newton là bước quan trọng để ta giải quyết bài toán. Vì thế để được tìm được nghiệm $(\Delta x, \Delta w, \Delta y, \Delta z)$ ta sẽ sử dụng **phương pháp Newton**.

Trong giải tích số, phương pháp Newton là một phương pháp tìm giá trị gần đúng của một hàm số có tham số thực bằng phương pháp xấp xỉ tuyến tính. Ta thường dùng phương pháp Newton để tìm nghiệm của hàm số hay $F(\xi) = 0$.

Ta có thể tổng quát phương pháp Newton để tìm nghiệm của hệ phương trình phi tuyến tính $F(\xi) = 0$ bằng cách cập nhật ξ. Ta xét hàm $F : \mathbb{R}^n \to \mathbb{R}^n$ với

$$F(\xi) = \begin{bmatrix} F_1(\xi) \\ F_2(\xi) \\ \vdots \\ F_N(\xi) \end{bmatrix}, \quad \xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix}$$

Tìm nghiệm của hệ phương trình nghĩa là ta cần tìm ξ^* sao cho $F(\xi^*) = 0$. Dùng phương pháp xấp xỉ tuyến tính $F(\xi)$ bằng Khai triển Taylor, ta có:

$$F(\xi + \Delta\xi) \approx F(\xi) + F'(\xi)\Delta\xi,$$

trong đó $F'(\xi)$ là ma trận Jacobi của $F(\xi)$:

$$J_F(X) = \nabla F = F'(\xi) = \begin{bmatrix} \frac{\partial F_1}{\partial \xi_1} & \frac{\partial F_1}{\partial \xi_2} & \dots & \frac{\partial F_1}{\partial \xi_N} \\ \frac{\partial F_2}{\partial \xi_1} & \frac{\partial F_2}{\partial \xi_2} & \dots & \frac{\partial F_2}{\partial \xi_N} \\ \vdots & \vdots & & \vdots \\ \frac{\partial F_N}{\partial \xi_1} & \frac{\partial F_N}{\partial \xi_2} & \dots & \frac{\partial F_N}{\partial \xi_N} \end{bmatrix}$$

Giải phương trình $F(\xi) + F'(\xi)\Delta\xi = 0$, ta tìm được $\Delta\xi$; thay ξ bằng $\xi + \Delta\xi$ và lặp lại quá trình trên đến khi hàm $F(\xi)$ đạt xấp xỉ 0.

Quay trở lại với bài toán đường trung tâm, ta gọi:

$$\xi = \begin{bmatrix} x \\ w \\ y \\ z \end{bmatrix} \qquad F(\xi) = \begin{bmatrix} Ax + w - b \\ A^T y - z - c \\ XZe - \mu e \\ YWe - \mu e \end{bmatrix}.$$

Ta tìm được ma trận Jacobi $F'(\xi) = \begin{bmatrix} A & I & 0 & 0 \\ 0 & 0 & A^T & -I \\ Z & 0 & 0 & X \\ 0 & Y & W & 0 \end{bmatrix}$

Giải hệ phương trình $F'(\xi)\Delta\xi = -F(\xi)$:

$$\begin{aligned} A\Delta x + \Delta w &= b - Ax - w \\ A^T\Delta y - \Delta z &= z + c - A^T y \\ Z\Delta x + X\Delta z &= \mu e - XZe \\ Y\Delta w + W\Delta y &= \mu e - YWe \end{aligned} \tag{13}$$

ta tìm được $\Delta\xi$ với $\Delta\xi = \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \\ \Delta z \end{bmatrix}$

2.4.4 Pha I

Ở Pha I, ta cần tìm một bộ chọn bộ $(x, w, y, z) > 0$ thoả mãn hệ (10) với $\mu = 1$. Ta sẽ bắt đầu với một bộ $(x, w, y, z) > 0$ bất kỳ ví dụ như ma trận toàn bộ là số 1, sau đó thực hiện phương pháp Newton để giải quyết, đưa bộ ban đầu ta lấy bất kỳ thành bộ thoả mãn. Sở dĩ ta làm được như vậy là do hướng Newton của hệ (10) cũng định hướng vào trong miền xác định, vậy nên từ một điểm nằm ngoài miền, ta có thể đi dần dần đến điểm thoả mãn hệ (10) và nằm trong miền nghiệm. Điều này đã được chứng minh bởi Lustig [2], cụ thể ở mục 3 và 4.

2.5 Hệ điều kiện Karush-Kuhn-Tucker (KKT)

Như đã trình bày ở mục trước đó, hệ điều kiện (10) đã được sử dụng để giải bài toán quy hoạch tuyến tính, thực ra hệ điều kiện này chính là một hệ quả của một vấn đề tổng quát hơn trong tối ưu lồi.

Trong tối ưu lồi, các hàm ràng buộc và hàm mục tiêu là các hàm lồi và khả vi, điều đó mang tính tổng quát hơn so với bài toán quy hoạch tuyến tính khi mà các hệ điều kiện chỉ là các ràng buộc tuyến tính (cũng là các ràng buộc lồi). Một kết quả đáng chú ý trong tối ưu lồi chính là **hệ điều kiện Karush-Kuhn-Tucker (KKT)**[2], kết quả này mang một ý nghĩa lớn khi đây chính là điều kiện đủ để một nghiệm khả thi được khẳng định là nghiệm tối ưu của bài toán, từ đó ta có thể xây dựng nên thuật toán tìm nghiệm tối ưu cho các bài toán tối ưu lồi mà cũng vì vậy ta sẽ đồng thời giải được cả các bài toán quy hoạch tuyến tính.

2.5.1 Hệ điều kiện Karush-Kuhn-Tucker (KKT) trong bài toán tối ưu lồi

Ta xét bài toán gốc như sau trong tối ưu lồi:

$$\begin{aligned} &\text{Minimize } f_0(x) \\ &\text{Subject to } f_i(x) \le 0, \quad i = 1, \dots, m \\ &\qquad\qquad h_i(x) = 0, \quad i = 1, \dots, p. \end{aligned}$$

Bài toán đối ngẫu Lagrange với hàm mục tiêu là hàm đối ngẫu Lagrange $g(\lambda, \nu)$ [3] tương ứng của bài toán trên trong tối ưu lồi là:

$$\begin{aligned} &\text{Maximize } g(\lambda, \nu) = \inf_{x \in D} \left(f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \right) \\ &\text{Subject to } \lambda \ge 0 \end{aligned}$$

Giả sử rằng bài toán gốc của chúng ta là một bài toán lồi, tức là các hàm f_i, h_i đều lồi và khả vi thì nếu ta tìm được một bộ $(\tilde{x}, \tilde{\lambda}, \tilde{\nu})$ thỏa mãn hệ điều kiện sau:

$$\begin{aligned} f_i(\tilde{x}) &\le 0, \quad i = 1, \dots, m \\ h_i(\tilde{x}) &= 0, \quad i = 1, \dots, p \\ \tilde{\lambda}_i &\ge 0, \quad i = 1, \dots, m \\ \tilde{\lambda}_i f_i(\tilde{x}) &= 0, \quad i = 1, \dots, m \\ \nabla f_0(\tilde{x}) + \sum_{i=1}^{m} \tilde{\lambda}_i \nabla f_i(\tilde{x}) + \sum_{i=1}^{p} \tilde{\nu}_i \nabla h_i(\tilde{x}) &= 0 \end{aligned} \tag{14}$$

thì $\tilde{x}$ chính là nghiệm tối ưu của bài toán gốc, và $(\tilde{\lambda}, \tilde{\nu})$ là nghiệm tối ưu của bài toán đối ngẫu.

Hệ điều kiện (14) ở trên được gọi là **hệ điều kiện Karush-Kuhn-Tucker (KKT)**.

[2] Xem thêm về hệ điều kiện này trong [1] mục **5.5.3 KKT optimality conditions**

[3] Xem thêm [1] mục **5.2 The Lagrange dual problem**

Trong hệ điều kiện này, thì 2 dòng đầu chính là điều kiện để thỏa mãn $\tilde{x}$ là nghiệm khả thi của bài toán gốc, dòng thứ 3 là điều kiện để $\tilde{\lambda}_i$ là nghiệm khả thi của bài toán đối ngẫu, dòng thứ 4 chính là độ lệch bù của nghiệm tối ưu bài toán gốc và bài toán đối ngẫu là bằng 0, dòng thứ 5 là điều kiện gradient của hàm Lagrangian $L(x, \lambda, \nu)$ bằng 0.

Bởi vì ta đã giả sử bài toán gốc ta đang xét là bài toán lồi, mà gradient của hàm Lagrangian $L(\tilde{x}, \tilde{\lambda}, \tilde{\nu}) = 0$ cho nên ta mới có được khẳng định $\tilde{x}$ là nghiệm tối ưu của bài toán gốc. Cũng theo đó mà ta sẽ có được $(\tilde{\lambda}, \tilde{\nu})$ là nghiệm tối ưu của bài toán đối ngẫu. [4]

2.5.2 Hệ điều kiện Karush-Kuhn-Tucker trong bài toán quy hoạch tuyến tính

Quay trở lại với hệ điều kiện (10) và bài toán tối ưu tuyến tính (cũng là một lớp con của các bài toán lồi) ta đang xét:

$$\begin{aligned} Ax + w &= b \\ A^T y - z &= c \\ XZe &= \mu e \\ YWe &= \mu e \end{aligned}$$

Ta nhận thấy rằng nếu (x, w, y, z) là nghiệm của hệ phương trình trên thì cũng thỏa mãn điều kiện: gradient của hàm Lagrangian tương ứng của bài toán gốc bằng 0 (Tương ứng với điều kiện số 5 của hệ 14), đồng thời khi đó điều kiện số 1 và số 2 cũng được thỏa mãn (tương ứng với điều kiện 1, 2, 3 của hệ 14).

Xuyên suốt quá trình thuật toán của chúng ta đã đề cập ở trên, ta đã giảm μ nhỏ dần cho đến 0, mà các điều kiện đã cho không bị phá vỡ, khi đó điều kiện 3 và 4 trong hệ (10) tương ứng với điều kiện 4 trong hệ (14) đã bị điều chỉnh (thay vì XZe = 0 và YWe = 0 mà là XZe = μe và YWe = μe).

Từ đó dẫn đến việc bộ (x, w, y, z) của ta sẽ dần thỏa mãn hệ KKT đã nêu và trở thành nghiệm tối ưu của bài toán gốc.

Từ kết quả trên cho thấy, hệ điều kiện (10) thực chất chính là **hệ điều kiện Karush-Kuhn-Tucker** trong trường hợp các hàm lồi của bài toán là các hàm tuyến tính và tham số μ tiến tới 0.

3 Cài đặt và so sánh thuật toán

Dưới đây là mã giả của thuật toán:

[4] Chứng minh chi tiết xem qua [1] **Chapter 5 Duality**

Khởi tạo các biến ban đầu: $\mu \leftarrow 1$, $\varepsilon \leftarrow$ Một giá trị đủ nhỏ để nghiệm xấp xỉ được tốt.
Từ kết quả ở pha 1 của thuật toán ta thu về được các giá trị ban đầu của bộ (x, y, z, w) thỏa mãn hệ phương trình: $\begin{cases} Ax + w = b \\ A^T y - z = c \\ XZe = \mu e \\ YWe = \mu e. \end{cases}$
while $\mu \geq \varepsilon$ **do**
    $\Delta x, \Delta y, \Delta z, \Delta w \leftarrow \text{NewtonMethod}(\mu, x, y, z, w)$
    $x \leftarrow x + \Delta x$
    $y \leftarrow y + \Delta y$
    $z \leftarrow z + \Delta z$
    $w \leftarrow w + \Delta w$
    $\mu \leftarrow \left(1 - \frac{1}{2\sqrt{n}}\right) \mu$
end while
Quá trình lặp ở trên sau một số hữu hạn bước sẽ kết thúc, khi đó return lại kết quả xấp xỉ nghiệm tối ưu
return x

Dưới đây là cách cài đặt thuật toán bằng Python và vẽ ra đường mà thuật toán của chúng ta đi để đến được điểm tối ưu:

```python
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as widgets
from IPython.display import display, clear_output

def convertDiag(x):
    X = np.zeros((len(x), len(x)))
    for i in range(len(x)):
        X[i][i] = x[i, 0]  # Extract the single element from the array
    return X

def NewtonMethod(x, w, y, z, mu, c, A, b):
    X = convertDiag(x)
    Y = convertDiag(y)
    W = convertDiag(w)
    Z = convertDiag(z)

    AT = np.transpose(A)
    m, n = A.shape

    Fe = np.block([[np.dot(A, x) + w - b],
                   [np.dot(AT, y) - z - c],
                   [np.dot(X, z) - mu * np.ones((n, 1))],
                   [np.dot(Y, w) - mu * np.ones((m, 1))]])

    dFe = np.block([
        [A, np.identity(m), np.zeros((m, m)), np.zeros((m, n))],
        [np.zeros((n, n)), np.zeros((n, m)), AT, -np.identity(n)],
        [Z, np.zeros((n, m)), np.zeros((n, m)), X],
        [np.zeros((m, n)), Y, W, np.zeros((m, n))]
    ])

    res = np.linalg.solve(dFe, -1 * Fe)

    deltax = res[:n]
```

```
    deltaw = res[n:n + m]
    deltay = res[n + m:n + m + m]
    deltaz = res[n + m + m:]

    return deltax, deltaw, deltay, deltaz

epsilon = 1e-10
def phaseII(x, w, y, z, c, A, b):
    mu = 1

    AT = np.transpose(A)
    m, n = A.shape

    path = []

    path.append([float(x[0, 0]), float(x[1, 0])])  # Extract single elements

    while mu >= epsilon:
        mu = (1 - 1 / (2 * np.sqrt(n))) * mu

        deltax, deltaw, deltay, deltaz = NewtonMethod(x, w, y, z, mu, c, A,
    b)

        x = x + deltax
        y = y + deltay
        w = w + deltaw
        z = z + deltaz

        path.append([float(x[0, 0]), float(x[1, 0])])  # Extract single elements


    return x, path

def phaseI(c, A, b):
    m, n = A.shape

    x = np.ones((n, 1))
    w = np.ones((m, 1))
    y = np.ones((m, 1))
    z = np.ones((n, 1))

    for i in range(100):
        deltax, deltaw, deltay, deltaz = NewtonMethod(x, w, y, z, 1, c, A,
   b)

        x = x + deltax
        y = y + deltay
        w = w + deltaw
        z = z + deltaz

    return x, w, y, z

def solve(cT, A, bT):
    c = np.transpose(cT)
    b = np.transpose(bT)

    xstart, wstart, ystart, zstart = phaseI(c, A, b)

    xoptimal, path = phaseII(xstart, wstart, ystart, zstart, c, A, b)

    return np.round(np.dot(cT, xoptimal), 3), path
```

```
95
96  def visualize(cT, A, bT, cpathpoints, limx1=10, limx2=10):
97      x1 = np.linspace(0, limx1, 400)
98      x2 = np.linspace(0, limx2, 400)
99      X1, X2 = np.meshgrid(x1, x2)
100
101     m, n = A.shape
102
103     constraint = []
104     for i in range(m):
105         constraint.append(A[i][0]*X1 + A[i][1]*X2 - bT[i])
106
107     constraint = np.array(constraint)
108
109     mu = 0.01
110     epsilon = 1e-6
111
112     barrierfunction = np.log(-np.minimum(constraint[0], -epsilon))
113     for i in range(1, m):
114         barrierfunction = barrierfunction + np.log(-np.minimum(constraint[i],
    -epsilon))
115     barrierfunction = mu * barrierfunction
116     barrierfunction = barrierfunction + mu * (np.log(x1 + epsilon) + np.log(x2
    + epsilon))
117
118     goalfunction = cT[0]*X1 + cT[1]*X2 + barrierfunction
119
120     feasiblemask = (constraint[0] >= 0)
121     for i in range(1, m):
122         feasiblemask = feasiblemask & (constraint[i] >= 0)
123     goalfunction = np.where(feasiblemask, goalfunction, np.nan)
124
125     plt.figure(figsize=(10, 10))
126     contour = plt.contourf(X1, X2, goalfunction, cmap='viridis', levels=100,
    alpha=0.7, extend='both')
127
128     cbar = plt.colorbar(contour)
129     cbar.set_label('Objective function value')
130
131     for i in range(m):
132         plt.contour(X1, X2, constraint[i], levels=[0], colors='r', linestyles='
    --')
133
134     plt.xlabel('$x1$')
135     plt.ylabel('$x2$')
136     plt.title('Feasible solution range')
137
138     arrowsize = 0.03
139     for i in range(len(cpathpoints) - 1):
140         arrowsize = arrowsize * 0.5
141         vectorstart = (cpathpoints[i][0], cpathpoints[i][1])
142         vectorend = (cpathpoints[i + 1][0], cpathpoints[i + 1][1])
143         plt.arrow(vectorstart[0], vectorstart[1],
144                   vectorend[0] - vectorstart[0],
145                   vectorend[1] - vectorstart[1], width=arrowsize,
146                   headwidth=arrowsize * 3, headlength=arrowsize * 1.5,
147                   fc='black', ec='black')
148
149     plt.grid(True)
150     plt.axhline(0, color='black', linewidth=0.5)
151     plt.axvline(0, color='black', linewidth=0.5)
```

```
152
153     plt.show()
154
155 def solveAndVisualize(cT, A, bT, limx1 = 10, limx2 = 10):
156   ans, path = solve(cT, A, bT)
157
158   cT = np.asarray(cT)[0]
159   A = np.array(A)
160   bT = np.asarray(bT)[0]
161
162   visualize(cT, A, bT, path, limx1, limx2)
```

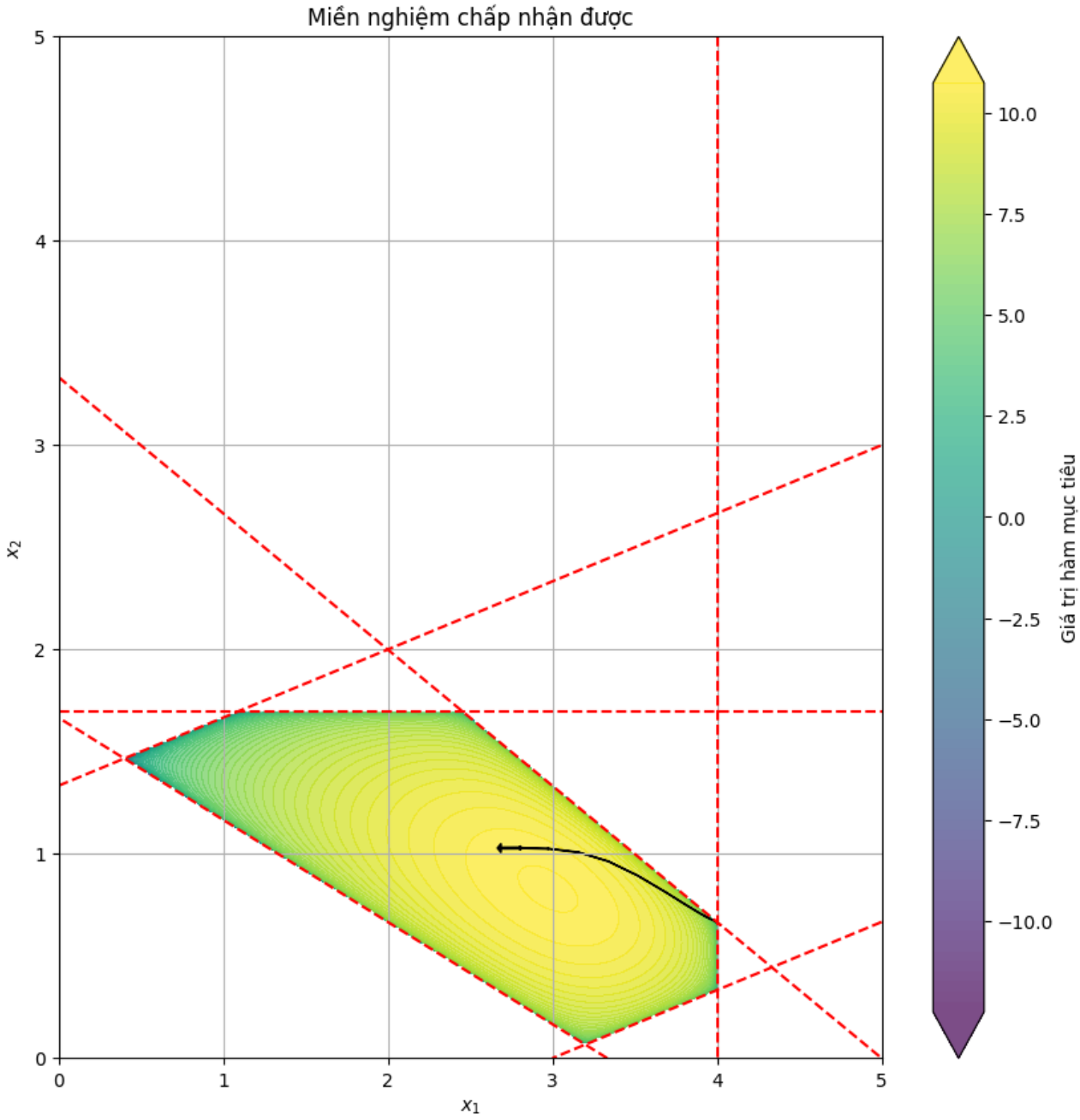
Chạy thử code với bài toán như sau:

$$\begin{aligned} \text{Maximize } & x_1 + x_2 \\ \text{Subject to } & x_1 \leq 4 \\ & x_2 \leq 1,7 \\ & 2x_1 + 3x_2 \leq 10 \\ & x_1 - 3x_2 \leq 3 \\ & -2x_1 + 6x_2 \leq 8 \\ & -3x_1 - 6x_2 \leq -10 \end{aligned} \tag{15}$$

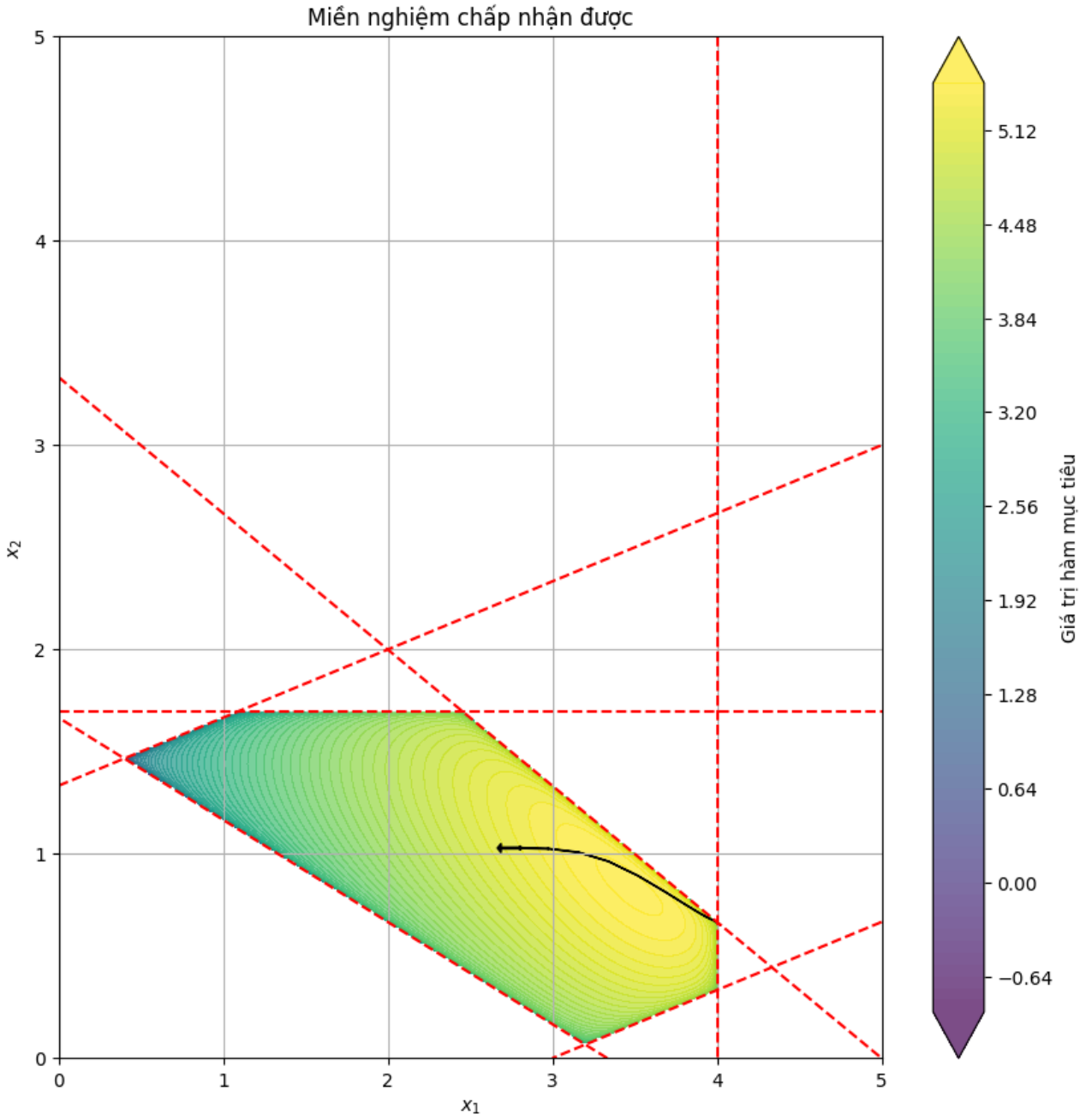
nhập các tham số tương ứng và chạy code như sau:

```
cT = np.matrix([1, 1])
A = np.matrix([[1, 0],
               [0, 1],
              [2, 3],
              [1, -3],
              [-2, 6],
              [-3, -6]])
bT = np.matrix([4, 1.7, 10, 3, 8, -10])
solveAndVisualize(cT, A, bT, 5, 5)
```

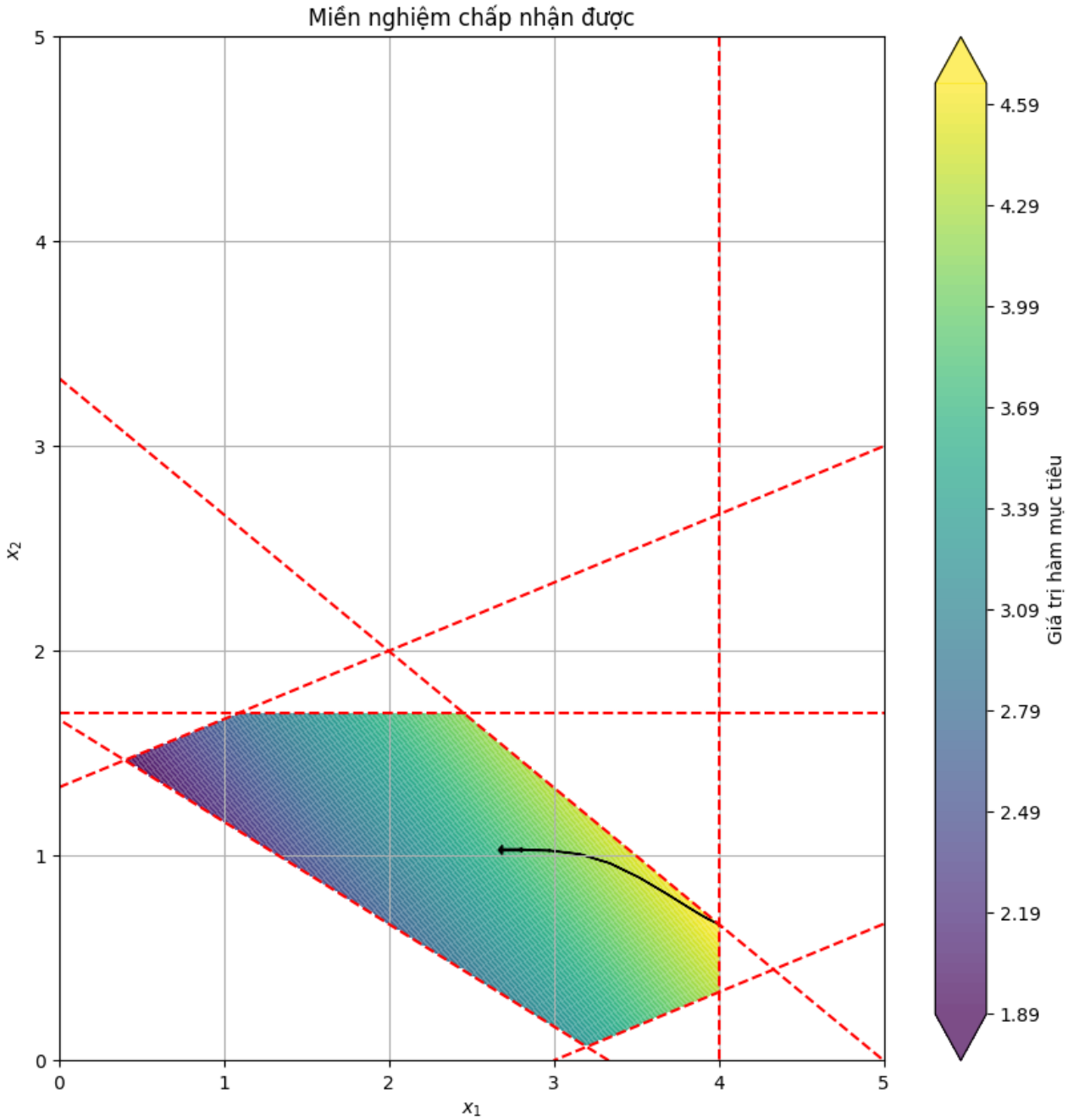
sau khi chạy code, ta sẽ được hình ảnh:


Miền nghiệm chấp nhận được
x_1
x_2
Giá trị hàm mục tiêu
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0

Hình 5: $\mu = 1$


Miền nghiệm chấp nhận được
x_1
x_2
Giá trị hàm mục tiêu
5.12
4.48
3.84
3.20
2.56
1.92
1.28
0.64
0.00
−0.64

Hình 6: $\mu = 0.2$



Hình 7: $\mu = 0.001$

Ở trong hình, ta vẽ ra đường đi được nối lại từ các điểm mà thuật toán ta đi qua và miền nghiệm hợp lệ với màu được tô là nhạt nếu tại đó giá trị hàm mục tiêu của bài toán chắn lớn, và màu được tô là đậm nếu tại đó giá trị hàm mục tiêu bài toán chắn nhỏ. Ta thấy rằng khi μ giảm dần (từ 1 về gần 0), vùng màu nhạt sẽ có xu hướng tập trung ở gần điểm tối ưu của bài toán gốc ban đầu, chứng tỏ khi μ càng nhỏ thì tác động của hàm chắn sẽ càng ít, nghiệm của bài toán chắn sẽ càng gần hơn với nghiệm của bài toán gốc và thuật toán của ta sẽ dần đi đến điểm tối ưu cần tìm.

4 Ứng dụng thuật toán

4.1 Ứng dụng của phương pháp điểm trong qua bài toán vận tải

4.1.1 Giới thiệu bài toán

Ta xét một ứng dụng thực tế của phương pháp điểm trong qua bài toán vận tải có bối cảnh như sau:
Nhằm chuẩn bị cho Hội xuân 2024 của trường đại học Nha Trang - NTU, n lớp đã được trưng dụng để làm nơi chế biến đồ ăn cho hội. Lớp i dự định chuẩn bị F_i đơn vị nguyên liệu, các nguyên liệu có thể mua được từ m siêu thị trên địa bàn thành phố: siêu thị j hiện đang có C_j đơn vị nguyên liệu. Chi phí để mua và chuyển một đơn vị nguyên liệu từ siêu thị j đến nhà của lớp i là P_{ij}.
Tóm tắt:

- Có n lớp. Mỗi lớp i cần F_i đơn vị nguyên liệu
- Có m siêu thị. Mỗi siêu thị j có C_j đơn vị nguyên liệu
- Chi phí để mua và chuyển 1 đơn vị nguyên liệu từ siêu thị j đến lớp i là P_{ij}

Mục tiêu của bài toán là

(1) mua và chuyển nguyên liệu sao cho tổng chi phí là nhỏ nhất

(2) đáp ứng nhu cầu của tất cả lớp

(3) không vượt quá lượng nguyên liệu ở mỗi siêu thị

4.1.2 Mô hình hóa bài toán

Bài toán vận tải trên được biểu diễn dưới dạng một bài toán quy hoạch tuyến tính như sau:

$$\begin{aligned} \text{Minimize } Z &= \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}x_{ij} \\ \text{Subject to } &\sum_{j=1}^{m} x_{ij} = F_i \forall i = 1, ..., n \\ &\sum_{i=1}^{n} x_{ij} \leq C_j \forall j = 1, ..., m \\ &x_{ij} \geq 0 \end{aligned}$$

Trong đó:

- x_{ij} là số đơn vị nguyên liệu mua từ siêu thị j đến lớp i với $x_{ij} \geq 0$
- Tối thiểu hóa hàm mục tiêu $Z = \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}x_{ij}$ là mục tiêu (1) của bài toán
- Ràng buộc $\sum_{j=1}^{m} x_{ij} = F_i \forall i = 1, ..., n$ là mục tiêu (2) của bài toán
- Ràng buộc $\sum_{i=1}^{n} x_{ij} \leq C_j \forall j = 1, ..., m$ là mục tiêu (3) của bài toán

4.2 Bài toán minh họa

4.2.1 Giới thiệu bài toán minh họa

Giả sử có 2 lớp và 2 siêu thị. Các lớp cần nguyên liệu chế biến như sau: Lớp thứ 1 cần 20 đơn vị nguyên liệu, lớp thứ 2 cần 30 đơn vị nguyên liệu.

Lượng nguyên liệu có sẵn tại các siêu thị như sau: Siêu thị thứ 1 có 40 đơn vị nguyên liệu, siêu thị thứ 2 có 60 đơn vị nguyên liệu.

Chi phí mua và vận chuyển như sau

	Siêu thị 1	Siêu thị 2
Lớp 1	30	20
Lớp 2	25	15

Bảng 1: Chi phí mua và vận chuyển (đơn vị: nghìn đồng)

Như vậy,

- $F_1 = 20$ và $F_2 = 30$
- $C_1 = 40$ và $F_2 = 60$
- $P_{11} = 30$, $P_{12} = 20$, $P_{21} = 25$,$P_{22} = 15$

Biểu diễn toán học của bài toán minh họa trên:

$$\begin{aligned} \text{Minimize } Z &= 30x_{11} + 20x_{12} + 25x_{21} + 15x_{22} \\ \text{Subject to } & x_{11} + x_{12} = 20 \\ & x_{21} + x_{22} = 30 \\ & x_{11} + x_{21} \leq 40 \\ & x_{12} + x_{22} \leq 60 \\ & x_{ij} \geq 0 \end{aligned}$$

Đưa biểu diễn trên về dạng bài toán cực đại hóa hàm mục tiêu, ta có:

$$\begin{aligned} \text{Maximize } Z &= -30x_{11} - 20x_{12} - 25x_{21} - 15x_{22} \\ \text{Subject to } & x_{11} + x_{12} = 20 \\ & x_{21} + x_{22} = 30 \\ & x_{11} + x_{21} \leq 40 \\ & x_{12} + x_{22} \leq 60 \\ & x_{ij} \geq 0 \end{aligned}$$

4.2.2 Sử dụng thuật toán đã cài đặt và giải bài toán minh họa

Ta kiểm tra nghiệm của bài toán trên bằng hàm linprog của thư viện `scipy`:

```
import scipy.optimize as so
from scipy.optimize import linprog
import numpy as np

# Transportation problem
c = np.matrix([30, 20, 25, 15])
Aub = np.matrix([
    [1, 0, 1, 0],
    [0, 1, 0, 1]
], dtype=float)
```

```
b_ub = np.matrix([40, 60])

# Define A_eq and B_eq
A_eq = np.matrix([
    [1, 1, 0, 0],
    [0, 0, 1, 1]
])
b_eq = np.matrix([20,30])

# Defind bounds
# Because the bounds is the same as default setting, so we don't have to define
    them
bounds = [
    (0, None)
    for i in range(len(c))
]

# With enough parameters, we use function scipy.optimize.linprog
# to solve the above linear programming problem
result = linprog(
    c=c,
    A_ub=A_ub,
    b_ub=b_ub,
    A_eq=A_eq,
    b_eq=b_eq,
    bounds=bounds,
)

if result.success:
    print( Optimal value: , result.fun)
    print( Optimal solution: , result.x)
else:
    print( Error: , result.message)
```

Sau khi chạy dòng code trên, ta đạt được kết quả:

```
    Optimal value: 850.0
    Optimal solution: [ 0. 20.  0. 30.]
```

Ở đây, chúng tôi sử dụng thư viện scipy để giải quyết bài toán, vì cài đặt của chúng tôi hoạt động không hiệu quả với những bài toán quy hoạch tuyến tính từ hai biến trở lên. Đây sẽ là mục tiêu khắc phục trong tương lai gần của nhóm.

Tài liệu

[1] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004.

[2] I.J. Lustig. Feasibility issues in a primal-dual interior-point method for linear programming. *Mathematical Programming*, 49(1-3):145–162, 1991.

[3] Jirí Matoušek and Bernd Gärtner. *Understanding and Using Linear Programming*. January 2007.

[4] Joel Sobel. Linear programming notes viii: The transportation problem. https://econweb.ucsd.edu/~jsobel/172aw02/notes8.pdf, 2002.