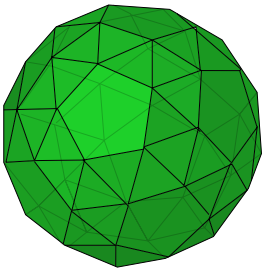
Projects in Mathematics and Applications

CONJUGATE GRADIENT DESCENT

Ngày 29 tháng 8 năm 2019

*Huỳnh Ngọc Tân
*Nguyễn Thiện Nhân
*Nguyễn Mạc Nam Trung
†Ngô Nhựt Bảo Trân



*Trường Phổ Thông Năng Khiếu, ĐHQG TPHCM
†Trường THPT Chuyên Thoại Ngọc Hầu, An Giang

Lời cảm ơn

Nhóm chúng em xin chân thành cảm ơn các thầy cô, các anh chị trong Ban Tổ chức trại hè Toán học và Ứng dụng PiMA đã tạo điều kiện cho chúng em cơ hội nghiên cứu về Toán Ứng dụng nói chung và lĩnh vực Deep Learning nói riêng, cũng như tạo cơ hội cho chúng em trải qua các hoạt động như chơi trò chơi lớn, làm dự án nhóm để mọi người có cơ hội tìm hiểu, học hỏi và gắn bó cùng nhau. Ngoài ra, chúng em xin cảm ơn các tổ chức tài trợ đã có những sự hỗ trợ quý giá cho PiMA. Đặc biệt, chúng em xin gửi lời cảm ơn đến trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Thành phố Hồ Chí Minh đã hỗ trợ chúng em rất nhiều trong hơn mười ngày diễn ra trại hè.

Nhóm chúng em hy vọng nhận được những góp ý, nhận xét từ mọi người về dự án này để dự án được hoàn thiện hơn.

Tóm tắt nội dung

Một hàm số bậc hai nhiều biến $f : \mathbb{R}^n \to \mathbb{R}$ luôn có thể được biểu diễn dưới dạng

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$$

trong đó A là ma trận đối xứng cấp n, b là vector thuộc $\mathbb{R}^n$ và c là số thực. Trong trường hợp ma trận A **xác định dương**, hàm số f xác định như trên là một hàm lồi nghiêm ngặt, có duy nhất một điểm cực trị địa phương đồng thời là cực trị toàn cục trên tập xác định $\mathbb{R}^n$:

$$\boldsymbol{x}_* = \arg\min_{\boldsymbol{x}\in\mathbb{R}^n} f(x).$$

Ta có thể xem xét bài toán tìm $\boldsymbol{x}_*$ dưới hai góc nhìn. Bên cạnh góc nhìn tối ưu hàm nhiều biến, việc tìm $\boldsymbol{x}_*$ tương đương với việc giải hệ phương trình tuyến tính xác định bởi phương trình

$$A\boldsymbol{x} = \boldsymbol{b}.$$

Bài báo cáo này trình bày phương pháp **Gradient liên hợp** (**Conjugate Gradient Descent** hay **CGD**) để tìm giá trị $\boldsymbol{x}_*$. Phương pháp **CGD**, với thuật toán Gradient liên hợp (**Conjugate Gradient Algorithm** hay **CGA**) đặc biệt hiệu quả trong việc tối ưu hàm lồi nghiêm ngặt bậc hai có dạng như trên. Điểm đặc biệt của phương pháp là xét cơ sở $\{d_1, \ldots, d_n\}$ trong $\mathbb{R}^n$ có tính chất liên hợp với ma trận đối xứng, xác định dương A. Thuật toán của ta sẽ tìm được giá trị $\boldsymbol{x}_*$ sau không quá n bước.

Ngoài ra, ở phần cuối bài báo cáo, nhóm cũng đưa ra các phiên bản của **CGD** cho trường hợp tổng quát là tối ưu hàm nhiều biến khả vi bất kì. Một số phiên bản thường được áp dụng là phương pháp Fletcher-Reeves và phương pháp Polak-Ribiere. Nhóm sẽ phân tích các thuật toán của những phiên bản khác nhau và đưa ra những nhận xét về ưu điểm, nhược điểm của từng phiên bản.

Tuy nhiên, trọng tâm của bài viết này là tìm hiểu thuật toán và ý nghĩa toán học của phương pháp **CGD** trong trường hợp hàm lồi nghiêm ngặt bậc hai.

Mục lục

1 Tổng quan về phương pháp Conjugate Gradient Descent

Mỗi hệ phương trình đại số tuyến tính tương ứng với một phương trình ma trận dạng $A\boldsymbol{x} = \boldsymbol{b}$. Nhiều phương pháp, thuật toán đã được đề xuất để giải phương trình ma trận này. Các thuật toán quen thuộc thường gặp là: thuật toán Gauss – Jordan cho ma trận A có kích thước bất kì, thuật toán Cramer đối với trường hợp A là ma trận vuông. Đặc điểm chung của các thuật toán trên là chúng thuộc loại phương pháp "trực tiếp", theo nghĩa nếu thực hiện tuần tự hữu hạn các bước trong thuật toán thì cuối cùng ta sẽ giải được phương trình $A\boldsymbol{x} = \boldsymbol{b}$.

Tuy nhiên, trong thực tế, đôi khi ma trận A có kích thước rất lớn và việc thực hiện các phương pháp trực tiếp trở nên khó khả thi. Một loại phương pháp khác để giải quyết vấn đề này là phương pháp lặp. Ý tưởng của phương pháp lặp là bắt đầu từ một điểm x_0, ta lần lượt cập nhật các x_i sao cho dãy $\{x_n\}$ sinh ra sẽ hội tụ về giá trị cần tìm.

Phương pháp Gradient liên hợp được đề xuất lần đầu bởi Hestenes và Stiefel vào những năm 1950. Nó được xem là một phương pháp lặp để giải các hệ phương trình tuyến tính với ma trận đối xứng, xác định dương.

Trong trường hợp ma trận A là xác định dương, việc giải hệ phương trình tuyến tính cũng tương đương với việc tối ưu hàm bậc hai nên về sau, phương pháp này được cải tiến và áp dụng trong các bài toán tối ưu.

2 Cơ sở toán học

Đây là danh sách những khái niệm, kiến thức nền để xây dựng các lí thuyết trong bài báo cáo này.

2.1 Đại số tuyến tính

- Vector và ma trận; Các phép tính cơ bản giữa vector với vector, ma trận với vector, ma trận với ma trận.
- Không gian vector hữu hạn chiều; Tổ hợp tuyến tính, không gian con, không gian con sinh bởi tập hợp; Độc lập tuyến tính, tập sinh và cơ sở.
- Không gian vector trang bị tích vô hướng; Cơ sở trực giao; Quá trình trực giao hóa Gram - Schmidt.
- Trị riêng và vector riêng.
- Ánh xạ tuyến tính; Dạng song tuyến tính; Dạng toàn phương.

2.2 Giải tích

- Vi phân hàm nhiều biến; Vector gradient.
- Cực trị hàm nhiều biến; Hàm lồi, hàm lồi nghiêm ngặt.

Trong phần báo cáo này, nhóm thường xuyên sử dụng một số thuật ngữ: Ma trận đối xứng, xác định dương; Cơ sở A - liên hợp (trong đó A đối xứng, xác định dương); Hàm lồi, hàm lồi nghiêm ngặt. Sau đây nhóm sẽ trình bày lại định nghĩa các khái niệm trên mà không thảo luận chi tiết về chúng.

Định nghĩa 2.1. Ma trận $A \in \mathbb{R}^{n \times n}$, đối xứng, được gọi là **xác định dương**, nếu với mọi vector $\boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{x} \neq \boldsymbol{0}$, ta luôn có:

$$\boldsymbol{x}^T A \boldsymbol{x} > 0.$$

Ví dụ 2.2. Cho $A = \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}$. Ta có A đối xứng. Mặt khác, với $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2, \boldsymbol{x} \neq \boldsymbol{0}$ tùy ý, ta có:

$$\boldsymbol{x}^T A \boldsymbol{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = x_1^2 - 2x_1x_2 + 2x_2^2 = (x_1 - x_2)^2 + x_2^2 > 0.$$

Vậy ma trận A đối xứng, xác định dương.

Định nghĩa 2.3. Cho $A \in \mathbb{R}^{n \times n}$ là ma trận đối xứng, xác định dương. Một tập hợp các vector $B = \{d_1, \ldots, d_n\}$ của $\mathbb{R}^n$ được gọi là **A - liên hợp** nếu:

$$d_i^T A d_j = 0, \forall 1 \leq i \neq j \leq n.$$

Từ định nghĩa, ta chứng minh được một tập A - liên hợp thì độc lập tuyến tính, do đó B cũng là một cơ sở của $\mathbb{R}^n$.

Nhận xét 2.4. *Nếu $A = I_n$ thì B trở thành một cơ sở trực giao.*

Định nghĩa 2.5. Hàm số $f : \mathbb{R}^n \to \mathbb{R}$ được gọi là **hàm lồi (convex function)** nếu:

$$f(\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1 - \theta) f(\boldsymbol{y})$$

với mọi $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, với mọi $0 \leq \theta \leq 1$.

Định nghĩa 2.6. Hàm số $g : \mathbb{R}^n \to \mathbb{R}$ được gọi là **hàm lồi nghiêm ngặt (strictly convex function)** nếu:

$$f(\theta \boldsymbol{x} + (1 - \theta)\boldsymbol{y}) < \theta f(\boldsymbol{x}) + (1 - \theta) f(\boldsymbol{y})$$

với mọi $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n : \boldsymbol{x} \neq \boldsymbol{y}$, với mọi $0 < \theta < 1$.

Định lý 2.7. *Nếu f và g là hàm lồi nghiêm ngặt thì $f + g$ cũng là hàm lồi nghiêm ngặt.*

Định lý 2.8. *Nếu f là hàm lồi nghiêm ngặt thì có tối đa một cực trị toàn cục.*

3 Tổng quan về thuật toán Gradient Descent

Gradient Descent là một thuật toán thường được sử dụng trong bài toán tối ưu hóa, cụ thể là tìm điểm cực tiểu của các hàm số nhiều biến và khả vi. Bắt đầu từ một điểm x_0, ta lần lượt cập nhật các giá trị x_i với mong muốn dãy số x_n của ta sẽ hội tụ đến điểm cực tiểu. Tư tưởng của phương pháp là đi ngược chiều đạo hàm.

Dữ liệu đầu vào: Một hàm J(θ) cần tìm cực tiểu theo biến θ, và một giá trị learning rate $\alpha(\alpha > 0)$ quyết định độ dài của bước đi (step size) theo hướng gradient.
Bước 1: Khởi tạo giá trị ngẫu nhiên cho biến θ
Bước 2: Cập nhật cho đến khi hội tụ hoặc đạt đến điều kiện dừng:

$$\theta \leftarrow \theta - \alpha \bigtriangledown_\theta J(\theta)$$

Một số đặc điểm của Gradient Descent:
Chọn learning rate: Việc chọn α có ý nghĩa quan trọng đối với kết quả của thuật toán. Nếu α quá lớn thì ta không hội tụ được về đích, nhưng nếu α quá nhỏ thì ta lại mất rất nhiều thời gian để chạy thuật toán này.
Chọn θ: Có thể cực tiểu tìm được là điểm cực tiểu địa phương hoặc thuật toán sẽ hội tụ tại một điểm yên ngựa, đây là điều ta thường không mong muốn.

4 Thuật toán Conjugate Gradient Descent

4.1 Ý tưởng thuật toán

Conjugate Gradient Descent (CGD) là một phiên bản phức tạp hơn của Gradient Descent, phỏng theo thuật toán xấp xỉ nghiệm của hệ phương trình tuyến tính :

$$\mathbf{Ax} = \mathbf{b}. \tag{1}$$

với $A \in \mathbb{R}^{n\times n}$, A đối xứng và xác định dương; $b \in \mathbb{R}^n$.
Ta không tính trực tiếp nghiệm x_* từ phương trình (1), thay vào đó ta sẽ xây dựng thuật toán tối ưu một hàm số nhận x_* là cực tiểu, ở đây ta xây dựng hàm số

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{Ax} - \mathbf{b}^\top \mathbf{x}.$$

Mệnh đề 4.1. Tồn tại duy nhất $\boldsymbol{x_*} \in \mathbb{R}^n$ sao cho f đạt giá trị nhỏ nhất tại $\boldsymbol{x_*}$. Hơn nữa, $\boldsymbol{x_*}$ chính là nghiệm của phương trình (1).

Chứng minh. Ta trình bày hai ý cần chứng minh:

- Hàm f lồi (tức hàm f có tối đa một cực trị toàn cục);
- Hàm f đạt cực tiểu tại x_* là nghiệm của phương trình (1).

a) **Hàm f lồi ngặt:**
Ta chứng minh $g(\mathbf{x}) = \mathbf{x}^\top \mathbf{Ax}$ là hàm lồi ngặt là đủ. Ta chứng minh bằng định nghĩa:

$$\begin{aligned} g(\mathbf{x}, \mathbf{y}, \lambda) &= \lambda g(\mathbf{x}) + (1-\lambda) g(\mathbf{y}) - g(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \\ &= \lambda \mathbf{x}^\top \mathbf{Ax} + (1-\lambda)\mathbf{y}^\top \mathbf{Ay} - (\lambda \mathbf{x} + (1-\lambda)\mathbf{y})^T \mathbf{A}(\lambda \mathbf{x} + (1-\lambda)\mathbf{y}) \\ &= \lambda(1-\lambda)(\mathbf{x}-\mathbf{y})^T A(\mathbf{x}-\mathbf{y}) \geq 0, \forall \lambda \in [0,1]. \end{aligned}$$

Dấu bằng xảy ra khi và chỉ khi $x = y$ nên g là hàm lồi nghiêm ngặt.

b) x_* **là điểm cực tiểu của** f:
Do f là hàm lồi ngặt nên ta chỉ cần chỉ ra $f(x) \geq f(x_*)$ với mọi $x \in \mathbb{R}^n$. Đặt $e = x - x_*$. Ta có:

$$\begin{aligned} f(x) = f(x_* + e) &= \frac{1}{2}(x_* + e)^T A(x_* + e) - b^T(x_* + e) \\ &= \frac{1}{2}(x_*)^T Ax_* + e^T Ax_* + \frac{1}{2}e^T Ae + b^T x_* - b^T e \\ &= f(x_*) - e^T b + b^T e + \frac{1}{2}e^T Ae \\ &= f(x_*) + \frac{1}{2}e^T Ae \geq f(x_*). \end{aligned}$$

□

Xây dựng thuật toán tối ưu f

Ta có $\nabla f(x) = Ax - b$. Ta xây dựng thuật toán ban đầu tương đối giống Gradient Descent:

- Chọn một điểm x_0 bất kỳ, di chuyển điểm theo hướng $d_0 = -\nabla f(x_0)$.
- Chọn độ dài đường đi là α_0 sao cho $f(x_0 + \alpha_0 d_0)$ đạt giá trị nhỏ nhất.
- Việc chọn hướng đi cũng khá quan trọng, ta chọn d_1 để kế thừa được f đạt cực tiểu tại α_0.

Ta có đạo hàm của $f(x_1)$ theo α_0 bằng 0. Theo quy tắc mắt xích,ta chứng minh được $\nabla f(x_1)$ vuông góc với d_0. như vậy $\nabla f(x_1 + \alpha_1 d_1)$ vuông góc với d_0, từ đây, ta thu được Ad_1 vuông góc với d_0. Từ đây, ta nghĩ tới việc xây dựng các vector hướng đi liên hợp với nhau theo ma trận A.

Như vậy so với Gradient Descent, ta đã có những điểm mới :

- Các hướng đi sẽ chọn liên hợp với nhau theo ma trận A.
- Learning rate cập nhật để tối ưu hàm theo hướng đi.

Tuy nhiên, ta vẫn chưa hình dung được hướng đi của d_k (trường hợp tổng quát hơn so với hai hướng đi đầu tiên) và chưa thấy được việc learning rate trong CGD sẽ hoạt động hiệu quả thế nào trong bài toán tối ưu hàm bậc 2 này.

4.2 Mối liên hệ giữa các đại lượng trong CGD

Đầu tiên, ta nhắc lại cách di chuyển giữa hai vector trong thuật toán GD:

$$x_{k+1} = x_k + \alpha_k d_k.$$

Do a_k được tính dựa vào việc tối thiểu $f(x_k + \alpha_k d_k)$ nên ta có thể viết :

$$\alpha_k = \underset{\alpha}{\operatorname{argmin}} f(x_k + \alpha d_k).$$

Đặt

$$\begin{aligned} F_k(\alpha) &= f(x_k + \alpha d_k) \\ &= \frac{1}{2}(x_k + \alpha d_k)^T A(x_k + \alpha d_k) - b^T(x_k + \alpha d_k) \\ &= \frac{1}{2} d_k^T A d_k \alpha^2 + d_k^T (Ax_0 - b)\alpha + \left(\frac{1}{2} x_k^T A x_k - b^T x_k\right). \end{aligned}$$

Do tính chất xác định dương của ma trận A nên F_k có cực tiểu và việc xác định cực tiểu dựa vào giải $\dfrac{\partial F_k(\alpha)}{\partial \alpha} = 0$.

Ta có : $0 = \dfrac{\partial F_k(\alpha)}{\partial \alpha} = (d_k^T A d_k)\alpha + d_k^T(Ax_k - b)$.

Từ đây, ta tính được learning rate : $\alpha_k = -\dfrac{d_k^T(Ax_k - b)}{d_k^T A d_k}$.

Vấn đề còn lại là xây dựng thuật toán cập nhật cho d_k, chú ý rằng tương tự với ý tưởng trong trường hợp d_0, d_1, ta mong muốn các cực trị tại $\alpha_0, \alpha_1, \cdots, \alpha_{k-1}$ được bảo toàn, nên ta mong muốn $d_k^T A d_i = 0$ với mọi $0 \leq i \leq k-1$. Ta xây dựng d_{k+1} có mối liên hệ với d_k và đại lượng gradient, nên ta có thể viết :

$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_{k+1} d_k.$$

Vấn đề của ta bây giờ là chọn β_{k+1} thế nào để các hướng đi liên hợp với nhau theo ma trận A. Ý tưởng tự nhiên là xử lí ta tính β_{k+1} dựa vào d_{k+1} và d_k và chứng minh $d_{k+1} A d_i = 0$, $\forall i \leq k+1$.

$$0 = d_{k+1}^T A d_k = -\nabla f(x_{k+1})^T A d_k + \beta_{k+1} d_k^T A d_k.$$

Từ đây,ta tính được $\beta_{k+1} = \dfrac{\nabla f(x_{k+1})^T A d_k}{d_k^T A d_k}$.

Để thuận tiện cho tính toán, ta đặt $r_k = \nabla f(x_k)$. Để chứng minh được $d_{k+1} A d_i = 0$, $\forall i \leq k+1$, ta cần bổ đề sau :

Bổ đề 4.2. Giả sử nghiệm ở lượt thứ k nào đó chưa là nghiệm, khi đó :

$$\begin{aligned} \text{span}\{r_0, r_1, \cdots, r_k\} &= \text{span}\ \{r_0, Ar_0, \cdots, A^k r_0\}\ . \\ \text{span}\{d_0, d_1, \cdots, d_k\} &= \text{span}\ \{r_0, Ar_0, \cdots, A^k r_0\}\ . \end{aligned}$$

Chứng minh. Ta chứng minh cả hai mệnh đề bằng quy nạp theo k, dễ thấy mệnh đề đúng với $k = 0$.

Giả sử mệnh đề đúng với k,ta có:

$$\text{span}\{r_0, r_1, \cdots, r_k, r_{k+1}\} = \text{span}\{r_0, r_1, \cdots, r_k, r_k + \alpha_k A d_k\} = \text{span}\{r_0, r_1, \cdots, r_k, A d_k\}$$

Từ đây, ta suy ra :

$$\text{span}\{r_0, r_1, \cdots, r_k, r_{k+1}\} \in \text{span}\{r_0, Ar_0, \cdots, A^{k+1} r_0\}$$

Lại có : $A^{k+1} r_0 \in \text{span}\{Ap_0, Ap_1, \cdots, Ap_k\}$ mà Ap_i có biểu diễn tuyến tính qua r_i và r_{i+1} nên ta suy ra $A^{k+1} r_0 \in \text{span}\{r_0, r_1, \cdots, r_{k+1}\}$ hay $\text{span}\{r_0, r_1, \cdots, r_{k+1}\} = \text{span}\ \{r_0, Ar_0, \cdots, A^{k+1} r_0\}$ (*). Ngoài ra, ta có : $\text{span}\{d_0, d_1, \cdots, d_k, d_{k+1}\} = \text{span}\{d_0, d_1, \cdots, d_k, -r_{k+1} + \beta_{k+1} d_k\} = \text{span}\{d_0, d_1, \cdots, d_k, r_{k+1}\} = \text{span}\{r_0, r_1, \cdots, r_k, r_{k+1}\} = \text{span}\ \{r_0, Ar_0, \cdots, A^{k+1} r_0\}$. (**)

Từ (*),(**), bổ đề được chứng minh. □

Trở lại bài toán, ta sẽ chứng minh $d_{k+1}^T A d_i = 0$, $\forall i \leq k+1$ bằng quy nạp theo k.
Với $k = 0$, ta đã xây dựng được.
Giả sử mệnh đề của ta đúng với k bằng l,ta sẽ chứng minh mệnh đề đúng đến $l+1$.

Ta có :

$$d_{l+1}^T A d_i = -(\nabla f(x_{l+1}))^T A d_i + \beta_{l+1} d_l^T A d_i = r_{l+1}^T A d_i + \beta_{l+1} d_l^T A d_i.$$

Chú ý rằng

$$r_{l+1}^T d_i = 0, \forall i = 0, 1, \cdots, l$$

mà $A d_i \in A$ span$\{r_0, Ar_0, \cdots, A^i r_0\}$ = span $\{Ar_0, A^2 r_0, \cdots, A^{i+1} r_0\} \subset \{d_0, d_1, \cdots, d_{i+1}\}$ nên ta suy ra được $r_{l+1}^T A d_i = 0$, $\forall i = 0, 1, \cdots, l-1$.

Mà theo mệnh đề quy nạp,ta có :

$$d_l^T A d_i = 0, \forall i = 0, 1, \cdots, l-1$$

nên ta có

$$d_{k+1}^T A d_i = 0, \forall i = 0, 1, \cdots, l-1$$

Mà do như cách tính β,ta đã có : $d_l^T A d_l = 0$, nên mệnh đề quy nạp được chứng minh xong.

Cuối cùng, ta sẽ đánh giá thuật toán, bằng một định lý :

Định lý 4.3. *Sau tối đa n bước thì x_0 sẽ đi đến x_n.*

Chứng minh. Ý tưởng chứng minh khá đơn giản như sau, ta chứng minh các hướng đi tạo thành một cơ sở, nên tồn tại một bộ $W = (w_1, w_2, \cdots, w_n)$ sao cho $x_* = x_0 + w_1 d_1 + w_2 d_2 + \cdots + w_n d_n$ và ta chứng minh $w_i = \alpha_i$ là đủ.
Phần 1 $d_0, d_1, \cdots, d_{n-1}$ tạo thành một cơ sở .
Ta chứng minh bằng phản chứng, giả sử ngược lại tồn tại một chỉ số h và các số thực r_i sao cho : $v_k = \sum_{i=1, i\neq h}^{n} r_i v_i$. Từ đây, ta tính được $v_h^T A v_k = \alpha_h v_h^T A v_h = 0$.
Nên $\alpha_h = 0$, từ đây,ta thu được điều vô lí.
Phần 2 $w_i = \alpha_i$, $i = 1, 2, \cdots, n$.
Đặt $e_k = x_k - x_*$ và $r_k = A e_k$ khi đó : $e_0 = -\sum_{i=0}^{n-1} w_i d_i$.
(1) Lại có : $e_k = x_k - x_* = x_k - x_0 + x_0 - x_* = \sum_{i=0}^{k-1} w_i d_i + e_0$
Nhân $d_k^T A$ vào hai vế của : $d_k^T A e_0 = -\sum_{i=0}^{n-1} w_i d_k^T A d_i = -w_k d_k^T A d_k$.
Từ các điều trên,ta thu được : $w_k = \dfrac{-d_k^T A e_k}{d_k^T A d_k} = -\dfrac{d_k^T r_k}{d_k^T A d_k} = \alpha_k$.

□

Như vậy, ta có các mối liên hệ giữa các đại lượng như sau:

$$\alpha_k = -\frac{r_k^T d_k}{d_k^T A d_k}$$
$$x_{k+1} = x_k + \alpha_k d_k$$
$$r_{k+1} = A x_{k+1} - b$$
$$\beta_{k+1} = \frac{r_{k+1}^T A d_k}{d_k^T A d_k}$$
$$d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$$

Ta cũng có biến đổi để thành công thức 2, các phép biến đổi khá cơ bản không trình bày trong khuôn khổ bài viết:

$$\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}$$
$$x_{k+1} = x_k + \alpha_k d_k$$
$$r_{k+1} = r_k + \alpha_k A d_k$$
$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$
$$d_{k+1} = -r_{k+1} + \beta_{k+1} d_k$$

4.3 Thuật toán

Dưới đây là thuật toán **CGD** cho trường hợp hàm lồi nghiêm ngặt bậc hai được viết bằng ngôn ngữ Python:

```
x = array([[-1000], [-200], [500]])
i_max = 100
i = 0
r = b - A@x
d = r
delta_new = np.dot(r.T, r)
delta0 = delta_new
q = 0
alpha = 0
beta = 0
while i < i_max and delta_new > 1e-15**2*delta0:
    q = A@d
    alpha = delta_new/np.dot(d.T,q)
    print(alpha)
    print("+")
    x = x + alpha*d
    if i % 50 == 0:
        r = b - A@x
    else:
        r = r - alpha*q
    delta_old = delta_new
    delta_new = np.dot(r.T,r)
    B = delta_new/delta_old
    d = r + B*d
    i += 1
return x
```

5 Tổng quát thuật toán CGD

Sau khi ta hiểu được thuật toán Conjugate Gradient Descent cho một **phương trình bậc 2** (Quadratic Function) bất kì, một cách tự nhiên ta sẽ nghĩ đến việc tổng quát hoá thuật toán trên với một hàm khả vi bất kì. Tuy nhiên, trước khi các bạn đọc phần này chúng mình xin phép được không trình bày quá chi tiết vào các cách chứng minh các định lý và chỉ điểm qua những điểm quan trọng.
Đầu tiên, ta sẽ ghi lại các bước cập nhật các tham số $\alpha_k(learning\ rate), \beta_k$ cho một hàm bậc 2 bất kì. Các bước cập nhật cho một hàm bậc 2 như sau

$x_0 \in \mathbb{R}$
$r_0 = Ax_0 - b$
$d_0 = -r_0$
$k = 0$
$while\ r_k \neq 0:$
$$\alpha_k = \frac{r_k^T r_k}{d_k^T A d_k}$$
$$x_{k+1} = x_k + \alpha_k d_k$$
$$r_{k+1} = r_k + \alpha_k A d_k$$
$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$
$$d_{k+1} = -r_k + \beta_k d_k$$
$$k = k + 1$$
$end.$

Dựa vào thuật toán CGD của phương trình bậc 2 ta có thể thấy các tham số α_k và β_k phụ thuộc rất nhiều vào khả năng đạo hàm của hàm này. Như vậy nếu ta có một hàm khả vi bất kì thì liệu việc tổng quát thuật toán CGD có gặp khó khăn hay không ? Theo [4] ta có 3 khó khăn chính khi tổng quát hoá thuật toán cho một hàm khả vi như sau

- (1) : Ta không thể sử dụng cách cập nhật r_k như trên.
- (2) : Sẽ phức tạp hơn khi tính toán α.
- (3) : Có rất nhiều cách khác nhau để cài đặt hàm β.

Ta sẽ nói qua các vấn đề và hướng giải quyết của các vấn đề đó.

5.1 Tìm kiếm theo đường thẳng (Line search)

Trong phần này ta sử dụng thuật toán line search để tìm kiếm α_k.
Ở đây, ta giải quyết bài toán cho một hàm khả vi bất kì khi biết rằng hàm này có tính giảm **(descent property)**
Ta sẽ sử dụng điều kiện Wolfe **(Wolfe condition)** để thoả mãn tính chất trên. Ở trên ta đã biết rằng

$$d_k = -r_k + \beta_k d_{k-1}$$
$$x_k = x_{k-1} + \alpha_k d_k$$

Điều kiện Wolfe được chia làm 2 loại chính là điều kiện Wolfe cơ bản **(Standard Wolfe condition)** và điều kiện Wolfe mạnh **(strong Wolfe condition)**. Được nêu ra như sau

Điều kiện Wolfe cơ bản (Standard Wolfe condition)

$$r(x_k + \alpha_k d_k)^T d_k \geq \sigma r_k^T d_k$$

Điều kiện Wolfe mạnh (strong Wolfe condition)

$$f(x_k + \alpha_k d_k) - f(x_k) \leq \rho \alpha_k r_k^T d_k$$
$$|r(x_k + \alpha_k d_k)^T d_k| \leq -\rho \alpha_k r_k^T d_k$$

Trong cả hai điều kiện trên thì ta đều phải có $0 < \rho < \sigma < 1$.
Nhắc lại điều kiện **tính giảm** thì ta có thể hiểu nó như sau :

$$r_k^T d_k < 0$$

Với thuật toán CGD thì điều này sẽ luôn đúng bởi vì ta biết rằng $d_k = -r_k + \beta_k d_{k-1}$ nên bằng cách nhân r_k^T vào hai vế ta có

$$r_k^T d_k = -r_k^T r_k + \beta r_k^T d_{k-1}$$

Vì vậy nếu thuật toán tìm kiếm đường thẳng của ta luôn thực hiện được với mỗi d_k thì ta sẽ có $r_k^T d_i = 0$ $(\forall i = \overline{0, k-1})$. Như vậy ta sẽ có

$$r_k^T d_k = -||r_k||^2$$

Vậy d_k sẽ luôn giảm vì $r_k \neq 0$.
Tuy nhiên trong thực nghiệm, không phải ở tất cả các bước ta đều tìm được d_k thoả $r_k^T d^i = 0$ $(\forall i = \overline{0, k-1})$. Vậy thì giả định d_k liên hợp với nhau sẽ không còn được thoả mãn. Thuật toán của ta sẽ chạy không còn chính xác.
Nhưng ta có cách để ta có thể sử dụng đó là dùng **strong Wolfe condition** để tìm kiếm. Điều kiện strong Wolfe được sử dụng để đánh giá một cách xấp xỉ giá trị α_k. Ở đây ta sẽ sử dụng **bisection** để tìm kiếm α_k ở bước thứ k. Thuật toán **bisection** là việc tìm kiếm nhị phân . Ở đây ta cần tìm α_k sao cho khi nhìn theo hướng của d_k thì ta có

$$g(\alpha) = d_k^T r_{+1} \approx 0$$

Để làm được điều này, ta đầu tiên chọn 2 điểm a, b sao cho $g(a)g(b) < 0$. Khi đó giá trị α làm cho hàm này xấp xỉ bằng 0 sẽ nằm trong đoạn [a,b] và ta sẽ chọn $c = \dfrac{a+b}{2}$. Ta tiếp tục kiểm tra dấu của $g(c)$. Nếu $g(c) > 0$ thì ta tiếp tục chọn $b = g(c)$ và lặp lại quá trình trên. Ngược lại, nếu $g(c) < 0$ thì ta chọn $a = g(c)$ và lặp lại quá trình trên. Ta có thể sử dụng ϵ : $g(\alpha) < \epsilon$ để đánh quá giá độ chính xác của $g(\alpha)$ (xét xem $g(\alpha)$ có đủ gần 0 hay không) trình cập nhật.

5.2 Phương pháp Polak-Ribière-Polyak

Phương pháp Polak-Ribière-Polyak được 2 nhóm Polak-Ribière và Polyak tìm ra độc lập với nhau nên đôi khi ta có thể gọi là phương pháp Polak-Ribière-Polyak hay cũng có thể gọi là phương pháp Polak-Ribière. Có rất nhiều phương pháp để điều chỉnh β_k, nhưng ở đây chúng tôi sẽ chỉ nêu phương pháp của Polak-Ribière-Polyak được sử dụng kết hợp với **inexact line search** cùng điều kiện Wolfe mạnh. Ta nêu công thức tính β_k như sau

$$\beta_{k+1}^{PRP} = \frac{r_{k-1}^T (r_k - r_k)}{r_k^T r_k}$$

Đối với công thức của Polak-Ribière-Polyak như trên ta có một số định lý để chứng minh được sự hội tự toàn cục (globally convergent) của các hàm số . Ở đây chúng tôi xin trình bày lại các định lý nhưng không chứng minh dùng để chứng minh sự hội tụ của phương pháp Polak-Ribière-Polyak và một số định lý khác

Định lý 1
Giả sử rằng chúng ta có $r_k^T d_k < 0$ thì ta sẽ chứng minh

$$\sum_{i=1}^{\infty} \frac{\|r_k\|^2}{\|d_k\|^2} = +\infty$$

Chúng ta tiếp tục định nghĩa một **tính chất (*)** như sau
Giả sử chúng ta có thể đã biết rằng

$$d_k = -r_k + \beta_k d_{k-1}$$
$$x_k = x_{k-1} + \alpha_k d_k$$

Khi đó ta nói một phương pháp có **tính chất (*)** khi tồn tại một hằng số $b > 1$ và $\zeta > 0$ sao cho $\forall k$:

$$|\beta_k| \leq b$$
$$\|x_{k+1} - x_k\| \leq \zeta \Rightarrow |\beta_k| \leq \frac{1}{2b}$$

Ta có thể thấy rằng phương pháp Polak-Ribière-Polyak đã có tính chất trên theo [3].
Ta có thêm một định lý cho phương pháp của Polak-Ribière-Polyak về việc **hội tụ mạnh** (strongly convergent) của phương pháp này là
Định lý 2
Với định lý này ta sẽ xét 3 điều kiện sau cần phải thỏa

- (1) $\beta \geq 0, \forall k$
- (2) **tính chất (*)** được thỏa
- (3) Điều kiện Wolfe thỏa mãn với mọi k

Khi đó ta có

$$\lim_{x \to \infty} (r_k) = 0$$

Ở hai định lý trên ta đã thấy được sự hội tụ của phương pháp Polak-Ribière-Polyak tuy nhiên tồn tại một số trường hợp về sự không hội tụ của phương pháp Polak-Ribière-Polyak và sau đây là các định lý về sự tồn tại đó .

Trước khi nói về **định lý 4** ta sẽ đề cập sơ qua đến công thức xây dựng cho một phương pháp khác (nhưng ta sẽ không bàn đến phương pháp đó ở đây, ta chỉ cần một định lý nhỏ được chứng minh trong công thức đó). Đó chính là một trong những phương pháp đầu tiên để ta có thể xây dựng và phát triển thêm thuật toán Conjugate Gradient Descent sau này đó là **phương pháp Fletcher-Reeves**. Chúng ta cùng nói qua cách cập nhật β ở định lý này.

$$\beta_k^{FR} = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$$

Từ đây, chúng ta cần các β_k thỏa mãn điều kiện sau

$$\sigma\beta_k \leq \overline{\sigma}\beta_k^{FR}$$

Trong đó σ được định nghĩa như ở điều kiện Wolfe mạnh như ở trên còn $\overline{\sigma} \in (0, \frac{1}{2}]$

Từ đây dẫn ta tới **định lý 3** như sau
Nếu ta có

$$||r_k^2|| \sum_{j=2}^{k} \prod_{i=j}^{k} (\frac{\beta_i}{\beta_i^{FR}})^2 \leq u \cdot k \quad (*1)$$

Trong đó u là một hằng số bất kì thỏa mãn $u > 0$ thì khi đó

$$\lim_{k \to \infty} inf(||r_k||) = 0$$

Note: r_k ở đây là của phương pháp Fletcher-Reeves nên sẽ khác với r_k ở thuật toán Polak-Ribière-Polyak. Tuy nhiên, tính chất $(*1)$ vẫn sẽ được sử dụng trong định lý 4

Định lý 4 Với mọi số $\epsilon > 0$, giả sử rằng

$$\beta_k = max\{\beta_k^{PRP}, -\epsilon\}$$

và với λ_k được chọn làm bất kì cực tiểu địa phương của $g_k(\lambda) = f(x_k + \lambda d_k), \lambda > 0$. Giả sử tính chất (**) được thỏa mãn cùng với tính chất $r_k^T d_k < 0$, khi đó tồn tại một hàm $f(x)$ sao cho $\{||r_k||\}$ được sinh bởi phương pháp Polak-Ribière-Polyak không tiến được tới gần điểm 0.

Các định lý trên chỉ được nêu tóm tắt và các chứng minh có thể được tìm thấy ở trong [3]. Có thể thấy rằng phương pháp Polak-Ribière-Polyak chỉ là một trong các phương pháp để tìm ra α và β cho một hàm khả vi bất kì. Có rất nhiều cách khác dựa trên nhiều định lý hơn cố gắng để chọn β sao cho bài toán tìm cực tiểu có thể được tối ưu hóa một cách tốt nhất nhưng đến nay vẫn chưa có phương pháp để giải quyết hoàn toàn bài toán.

6 Ưu, nhược điểm của thuật toán CGD

Thuật toán Conjugate Gradient Descent là một trong những thuật toán để tối ưu hóa hàm số. Trong thực tế, rất ít những thuật toán tối ưu hóa có thể dùng để tổng quát hầu hết các bài toán. Các thuật toán do con người làm ra cũng có những ưu điểm và nhược điểm nhất định và đôi khi những thuật toán ấy chỉ giải quyết được cho những bài toán cụ thể.
Điều tương tự cũng được áp dụng cho thuật toán Conjugate Gradient Descent (CGD). Sau đây chúng ta sẽ điểm qua một số điểm mạnh và yếu nhất định của thuật toán này.

6.1 Ưu điểm

Như ta đã thấy ở phần trình bày thì thuật toán Coujugate Gradient Descent chỉ thuận lợi nhất khi chúng ta giải quyết bài toán ở hàm bậc 2. Hơn nữa thuật toán này sẽ chỉ chạy nhanh được khi ta may mắn khởi tạo được điểm đầu tiên mà ta có thể khởi tạo các hướng tiếp theo hợp lý để việc tính toán phù hợp với những gì ta xây dựng trong lý thuyết.

6.2 Nhược điểm

Có thể thấy thuật toán Conjugate Gradient Descent không được sử dụng phổ biến hiện nay bởi vì thực tế từ thực nghiệm trong quá khứ và cũng như hiện nay thì thuật toán này vẫn còn gặp nhiều vấn đề. Ở đây ta sẽ liệt kê 3 vấn đề cơ bản như sau:

- (1) : Theo lý thuyết khi ta xây dựng hàm bậc 2 thì ta cần có các hướng d_i phải liên hợp (conjugate) với nhau nhưng thực tế sẽ có rất nhiều hàm khả vi bị phá vỡ sự liên hợp đó khi ta cập nhật các biến.

- (2) : Việc tìm kiếm learning rate α_k sẽ khó thực hiện hơn khi hàm đó là một hàm khả vi bất kì. Khi đó ta phải sử dụng các phương pháp khác để tìm giá trị của α nhưng cũng không quá chính xác.

- (3) : Các phiên bản của Conjugate Gradient Descent trong trường hợp tổng quát được biết đến hiện tại đều có những mặt hạn chế. Chẳng hạn, phương pháp Fletcher - Reeves có thể hội tụ chậm hơn cả thuật toán Steepest Gradient Descent. Trong khi đó, dù phương pháp Polak-Ribiere thường được sử dụng và cho ra tốc độ hội tụ nhanh hơn, nhưng có những hàm số mà áp dụng thuật toán Polak-Ribiere, ta sẽ không thể hội tụ được về điểm cực tiểu.

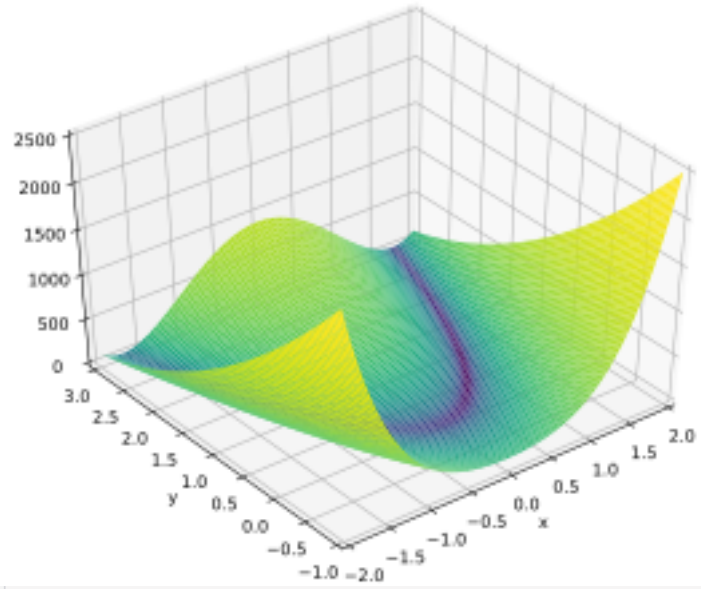
7 Áp dụng mô hình

Source code

Giới thiệu hàm Rosenbrock

Rosenbrock là hàm được dùng phổ biến để đánh giá hiệu quả của mô hình tìm cực trị hàm số. Ta sẽ dùng hàm này để đánh giá hiệu quả của phương pháp Conjugate Gradient Descent.

$f(x) = f(x_1, x_2, \cdots, x_N) = \sum_{i=1}^{\frac{N}{2}} [b(x_{2i-1}^2 - x_{2i})^2 + (x_{2i-1} - a)^2]$ với N chẵn, a, b là hằng số cho trước.




```
def fmin(f, l, h):
    mesh = np.linspace(l, h, 10000)
    return mesh[np.argmin(np.fromiter((f(mesh[i]) for i in range(len(mesh))), mesh.dtype))]
```

```
def linesearch(x, d, lim=0.1):
    f = lambda a: r(x[0][0] + a * d[0][0], x[1][0] + a * d[1][0])
    argmin = fmin(f, -lim, lim)
    return fmin(f, -lim, lim)
```

```
def gCGD(df, x):
    d = -df(x)
    while abs(r(x[0][0], x[1][0])) > 1e-7:
        grad_i = df(x)
        alpha = linesearch(x, d)
        x = x + alpha*d
        grad_i_plus_one = df(x)
        B = (grad_i_plus_one.T @ (grad_i_plus_one - grad_i)) / (d.T @ (grad_i_plus_one - grad_i))
        d = -grad_i_plus_one + B*d
    return x
```

Tài liệu tham khảo

1. J.Nocedal Stephen, J.Wright. *Numerical Optimization*. Springer, 1999.

2. Andrew Gibiansky, *Conjugate Gradient*,
   http://andrew.gibiansky.com/blog/machine-learning/conjugate-gradient/

3. YUHONG DAI, JIYE HAN, GUANGHUI LIU, DEFENG SUN, HONGXIA YIN, AND YA-XIANG YUAN. *Convergences properties of nonlinear conjugate gradient methods*
   http://www.math.nus.edu.sg/ matsundf/NonlinearCG.pdf

4. Jonathan Richard Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*, 1994.
   https://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.pdf

5. https://en.wikipedia.org/wiki/Conjugate-gradient-method

6. https://en.wikipedia.org/wiki/Rosenbrock-function