Phân tích suy biến (SVD)

Phân tích suy biến (Singular Value Decomposition (SVD)) là phương pháp phân tích ma trận A kích thước $m \times n$ thành dạng tích ba ma trận

$$A = U\Sigma V^T$$

trong đó:

- U là ma trận trực giao có kích thước $m \times m$.
- Σ là ma trận đường chéo kích thước $m \times n$.
- V là ma trận trực giao có kích thước $n \times n$.

Scipy và svd

Scipy (Scientific Python) là phần mềm nguồn mở cho toán học, khoa học và kỹ thuật. Thư viện SciPy được xây dựng dựa trên thư viện NumPy, cung cấp thao tác mảng N chiều thuận tiện và nhanh chóng. SciPy gồm các gói con (submodule) cho đại số tuyến tính, tối ưu hóa, tích hợp và thống kê.

- Scipy chứa nhiều loại gói phụ giúp giải quyết vấn đề phổ biến nhất liên quan đến tính toán khoa học.
- Scipy là thư viện Khoa học được sử dụng nhiều nhất chỉ sau Thư viện Khoa học GNU cho C/C ++ hoặc Matlab.
- Dễ sử dụng và dễ hiểu cũng như tính toán nhanh.
- Có thể hoạt động trên mảng (array) của thư viện NumPy.

Hàm svd trong Scipy: là một hàm trong gói con linalg của Scipy, được dùng để tìm phân tích suy biển (svd) một ma trận A cấp $m \times n$ bất kỳ.

$scipy.linalg.svd(A, full\_matrices = True, compute\_uv = True)$

- A: ma trận cần phân tích cấp $m \times n$.
- $full\_matrices$: nhập giá trị $True$ hoặc $False$, không bắt buộc. Nếu $True$ (mặc định), ma trận U và V^T có cấp $m \times m$ và $n \times n$. Nếu $False$, các ma trận U và V^T có cấp lần lượt là $m \times k$ và $k \times n$, trong đó $k = min(m, n)$.
- $compute\_uv$: nhập giá trị $True$ hoặc $False$, không bắt buộc. Nếu $True$, kết quả trả về sẽ bao gồm ma trận U, V^T và vector S_diag. Ngược lại, chỉ có vector S_diag được trả về. Giá trị mặc định là $True$.

Kết quả trả về:

- U: ma trận cấp $m \times m$ hay $m \times k$ phụ thuộc vào $full\_matrices = True$ or $False$.
- S_diag: vector chứa giá trị kỳ dị của ma trận A được sắp theo thứ tự không tăng, có kích thước $(k,)$.
- V^T: ma trận cấp $n \times n$ hay $n \times k$ phụ thuộc vào $full\_matrices = True$ or $False$.

Sinh viên đọc thêm về hàm svd của scipy tại đây

Bài tập 1 - Tìm phân tích suy biến svd của một ma trận

Sử dụng hàm **svd** của package **scipy** trong Python để viết một chương trình cho phép tính dạng phân tích suy biến của ma trận dưới đây:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

Thuật toán

- Đầu vào: Ma trận A.
- Đầu ra: Ma trận U, V^T và Σ.

Các bước làm bài:

1. Khai báo thư viện cần dùng, thư viện scipy.linalg
2. Nhập ma trận A
3. Dùng hàm svd lên ma trận A để tính ma trận U, V^T và vector S_diag. Lưu ý: các phần tử của vector S_diag đều không âm và được sắp xếp theo thứ tự giảm dần.
4. Tạo ra ma trận Σ từ vector S_diag
   - 4.1 Tạo ma trận không cấp $k \times k$.
   - 4.2 Thay đường chéo của ma trận không bằng các phần tử của S_diag.
   - 4.3 Thêm hàng/cột 0 vào Σ để cùng dạng với ma trận A ban đầu.
5. Xuất kết quả.

Compact SVD

Giả sử trên đường chéo chính của ma trận Σ có r giá trị khác 0. Nghĩa là

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0.$$

Ta có thể biểu diễn SVD dưới dạng tổng của các vector cột $u_i \in \mathbb{R}^n$ của U và vector dòng $v_i \in \mathbb{R}^n$ của V^T như sau:

$$A = \sum_{i=1}^{r} u_i \lambda_i v_i. \tag{2}$$

Từ đó, ta có một dạng SVD gọn hơn gọi là **compact SVD**

$$A = U_r \Sigma_r V_r^T \tag{3}$$

trong đó

- U_r và V_r lần lượt là ma trận được tạo bởi r cột đầu tiên của U và V.
- Σ_r là ma trận được tạo bởi r hàng đầu tiên và r cột đầu tiên của Σ.

Bài tập 2 - Tìm compact SVD của một ma trận

Viết chương trình cho phép nhập vào một ma trận, sau đó tính compact SVD của ma trận đó. Sử dụng ma trận dưới đây để kiểm tra.

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 2 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \end{bmatrix} \tag{4}$$

Thuật toán:

- Đầu vào: Ma trận A.
- Đầu ra: Ma trận U_r, V_r^T và Σ_r

Các bước làm bài:

1. Tìm SVD của ma trận A như trên (dùng scipy).
2. Tìm số giá trị λ khác 0 của ma trận A, đặt là r.
3. Giữ lại r hàng và r cột đầu tiên của ma trận Σ để tạo thành Σ_r.
4. Giữ lại r cột đầu tiên của ma trận U để tạo ra U_r.
5. Giữ lại r hàng đầu tiên của ma trận V^T để tạo ra V_r^T.
6. Xuất kết quả.

Lưu ý: Khi nhân $U_r \Sigma_r V_r^T$ với nhau, ta vẫn được ma trận A như ban đầu. Phép giảm chiều compact SVD không làm thay đổi ma trận A. Vì vậy, sau khi tìm được compact SVD của A, hãy thử nhân lại $U_r \Sigma_r V_r^T$ để kiểm tra kết quả.

Truncated SVD

Trong ma trận Σ, các giá trị trên đường chéo chính là không âm và giảm dần

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_r > 0.$$

Thông thường, chỉ có một số các trị riêng λ_i mang giá trị lớn, các giá trị còn lại thường nhỏ và gần 0. Khi đó, với phương pháp truncated SVD, ta có thể xấp xỉ ma trận A dưới dạng

$$A \approx A_k = U_k \Sigma_k V_k^T \text{ với } k < r \tag{5}$$

trong đó

- U_k và V_k lần lượt là ma trận được tạo bởi k cột đầu tiên của U và V.
- Σ_k là ma trận được tạo bởi k hàng đầu tiên và k cột đầu tiên của Σ.

Việc xóa bớt những giá trị λ nhỏ và gần bằng 0 khiến cho việc lưu trữ thông tin của ma trận A không thể y như ban đầu. Khi chúng ta nhân ba ma trận U_k, Σ_k, V_k^T lại với nhau, ta chỉ nhận được một ma trận gần đúng với ma trận A.
$\Longrightarrow$ Số giá trị λ được giữ lại ảnh hưởng đến lượng thông tin được giữ lại của ma trận A sau khi thực hiện phép giảm chiều truncated SVD.
Đây chính là điểm khác biệt giữa compact SVD và truncated SVD.

k và lượng thông tin trong ma trận A

Công thức tính phần trăm lượng thông tin giữ lại sau khi giữ lại k giá trị λ:

$$I = (1 - \frac{\|A - A_k\|_F^2}{\|A\|_F^2}) * 100 = (\frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{j=1}^{r} \lambda_j^2}) * 100 \tag{6}$$

Công thức tính số k cần giữ lại nếu muốn giữ lại ít nhất $I\%$ lượng thông tin trong A:
Chọn k nhỏ nhất sao cho

$$\frac{\sum_{i=1}^{k} \lambda_i^2}{\sum_{j=1}^{r} \lambda_j^2} \geq I\% \tag{7}$$

Bài tập 3 - Tìm truncated SVD khi biết số λ cần giữ lại

Viết chương trình cho phép nhập vào ma trận A dưới đây:

$$A = \begin{bmatrix} 1.01 & 0.9 & 0.2 & 1.001 & 0.3 \\ 0.2 & 1.01 & 0.3 & 0.8 & 0.4 \\ 1 & 1.002 & 2 & 0.98 & 2 \\ 0.3 & 2 & 0.4 & 1.01 & 0.9 \\ 1.1 & 0.2 & 0.03 & 2 & 0.87 \end{bmatrix} \tag{8}$$

Sau đó tính truncated SVD của ma trận A khi giữ lại 4 giá trị λ lớn nhất và xuất ra lượng thông tin còn giữ lại lúc này của ma trận.

Thuật toán

- Đầu vào: Ma trận A.
- Đầu ra: Ma trận U_k, V_k, Σ_k và phần trăm lượng thông tin còn giữ lại lúc này.

Các bước làm bài:

1. Tìm phân tích SVD của ma trận A như bài tập 1.
2. Giữ lại 4 hàng, 4 cột đầu tiên của ma trận Σ để tạo ra ma trận Σ_k
3. Giữ lại 4 cột đầu tiên của ma trận U để tạo ra U_k.
4. Giữ lại 4 hàng đầu tiên của ma trận V^T để tạo ra V_k^T.
5. Áp dụng công thức (6) để tính phần trăm lượng thông tin được giữ lại lúc này.
6. Xuất kết quả.

Bài tập 4 - Tìm truncated SVD khi biết phần trăm lượng thông tin muốn giữ

Viết chương trình cho phép nhập vào ma trận A dưới đây. Tìm truncated SVD của ma trận A khi muốn giữ lại ít nhất 90% lượng thông tin ban đầu.

$$A = \begin{bmatrix} 1.01 & 0.9 & 0.2 & 1.001 & 0.3 \\ 0.2 & 1.01 & 0.3 & 0.8 & 0.4 \\ 1 & 1.002 & 2 & 0.98 & 2 \\ 0.3 & 2 & 0.4 & 1.01 & 0.9 \\ 1.1 & 0.2 & 0.03 & 2 & 0.87 \end{bmatrix} \tag{9}$$

Thuật toán

- Đầu vào: Ma trận A và lượng thông tin cần giữ.
- Đầu ra: Ma trận U_k, V_k^T, Σ_k và số $k\lambda$ cần giữ.

Các bước làm bài:

1. Tìm SVD của ma trận A như bài tập 1.
2. Áp dụng công thức (15) để tìm ra số $k\lambda$ cần phải giữ.
3. Giữ lại k hàng, k cột đầu tiên của ma trận Σ để tạo ra ma trận Σ_k
4. Giữ lại k cột đầu tiên của ma trận U để tạo ra U_k.
5. Giữ lại k hàng đầu tiên của ma trận V^T để tạo ra V_k^T.
6. Xuất kết quả.

Bài tập 5 - Ứng dụng của truncated SVD trong việc nén ảnh

Viết chương trình cho phép nén ảnh theo phương pháp truncated SVD và xuất ảnh đã bị nén với 50, 100, 250 và 300 lần lượt là số λ lớn nhất để lưu giữ thông tin. Sau đó, xuất ra lượng thông tin còn giữ lại tương ứng.

Hình 1: Hình gốc

Lưu trữ ảnh dưới dạng mảng

Để thao tác với ảnh trong chương trình, chúng ta cần phải lưu trữ hình ảnh có kích thước $m \times n$ sang dạng mảng bằng câu lệnh:

$$\text{A} = \text{numpy.array(Image.open('file ảnh'))}$$

Lúc này, A là một mảng gồm 3 ma trận cấp $m \times n$ đại diện cho các cường độ khác nhau của 3 màu đỏ, xanh lá và xanh dương. Mảng A được gọi là ảnh RGB. Trong đó, màu sắc của mỗi pixel của ảnh được tạo ra từ sự kết hợp bởi các cường độ khác nhau của các màu đỏ, xanh dương và xanh lá. Các cường độ này có giá trị nằm trong khoảng từ 0 đến 255.
Ví dụ: Nhấp vào đường link sau để chọn màu RGB.

Làm xám ảnh

Để nén ảnh, mình cần tìm truncated SVD của mảng A dùng để lưu trữ ảnh.
Tuy nhiên, hàm svd của Scipy chỉ làm việc với một ma trận nên chúng ta cần phải chuyển A từ 3 ma trận sang 1 ma trận, tức là biến ảnh màu RGB thành ảnh xám (grayscale image) với màu sắc của mỗi pixel chỉ được tạo thành từ cường độ của ánh sáng, hay nói cách khác là chỉ các sắc thái của màu xám.
Cách chuyển đổi:

- **Dùng công thức:**

$$A = A.dot([0.299, 0.5870, 0.114])$$

- **Dùng OpenCV:** Sinh viên khai báo thư viện cv2 và áp dụng

$$A = cv2.cvtColor(A, cv2.COLOR\_RGB2GRAY)$$

Lưu ý:

- Khi sử dụng công thức trên và vẽ ảnh bằng imshow(), ảnh sẽ có màu xanh. Thế nhưng, đó vẫn là ảnh xám. Để vẽ ảnh ra màu xám đẹp, khi vẽ ảnh, ta làm như sau:

$$\text{plt.imshow(A, cmap} = \text{'gray'})$$

- Ngoài những cách này, còn rất nhiều cách khác giúp làm xám ảnh. Ví dụ như dùng Pillow,... Sinh viên có thể dùng bất cứ phương pháp nào tùy ý.

Thuật toán

- Đầu vào: Hình ảnh chùa Một Cột.
- Đầu ra: Hình ảnh chùa đã bị nén lần lượt với số giá trị λ được giữ lại theo đề bài và phần trăm lượng thông tin giữ lại tương ứng.

Các bước làm bài:

1. Khai báo các thư viện cần thiết.
2. Dùng hàm numpy.array(Image.open('file ảnh')) để đọc hình ảnh và biến hình ảnh thành dạng mảng, đặt là A.
3. Nhân A với mảng $(0.299, 0.5870, 0.1140)$ để tạo ảnh xám.
4. Tìm truncated SVD của A như trên với 50 số λ lớn nhất được giữ lại.
5. Tạo lại ma trận A từ 3 ma trận của truncated SVD. Đây chính là hình ảnh sau khi đã nén.
6. Áp dụng công thức (6) để tính phần trăm lượng thông tin giữ lại và xuất kết quả.
7. Vẽ ảnh sau khi đã nén và trước khi nén để so sánh. (dùng imshow)
8. Làm tương tự với số λ còn lại.

Bài tập 6 - Ứng dụng của SVD trong việc giải thuật toán hồi quy tuyến tính

Giả sử ta có số liệu thống kê các thông số RAM, bộ nhớ, PIN và giá tiền của 24 chiếc điện thoại như trong bảng 2:

	NAME	RAM	MEMORY	PIN	PRICE
0	Samsung Galaxy A01 Core	1	16	3000	1,850,000
1	Samsung Galaxy A11	3	32	4000	2,650,000
2	Samsung Galaxy A02s	4	64	5000	3,350,000
3	Samsung Galaxy J7 Prime	3	32	3300	3,790,000
4	Samsung Galaxy A21s	3	32	5000	4,250,000
5	Samsung Galaxy A22 4G	6	128	5000	4,700,000
6	Samsung Galaxy A30s	4	64	4000	4,150,000
7	Samsung Galaxy A31	6	128	5000	5,150,000
8	Samsung Galaxy A52	8	128	4500	6,600,000
9	Samsung Galaxy A72	8	256	5000	10,100,000
10	Samsung Galaxy Note 10 Lite	8	128	4500	10,500,000
11	Samsung Galaxy S10+	8	128	4100	12,400,000

Hình 2: Bảng số liệu thống kê số RAM, bộ nhớ, PIN và giá tiền của 24

	NAME	RAM	MEMORY	PIN	PRICE
12	Samsung Galaxy S21 FE 5G	8	256	4500	13,650,000
13	Samsung Galaxy S21 FE	8	128	4500	12,790,000
14	Samsung Galaxy S20+	8	128	4500	15,500,000
15	Samsung Galaxy S20 Ultra	12	128	5000	16,000,000
16	Samsung Galaxy Note 20 Ultra	12	256	4500	18,990,000
17	Samsung Galaxy Z Flip3 5G	8	256	3300	19,350,000
18	Samsung Galaxy Z Flip	8	256	3300	20,990,000
19	Samsung Galaxy S21 Plus 5G	8	256	4800	23,000,000
20	Samsung Galaxy Z Fold2	12	256	4500	23,000,000
21	Samsung Galaxy S21 Ultra 5G	12	256	5000	29,800,000
22	Samsung Galaxy S22 Ultra	12	512	5000	29,990,000
23	Samsung Galaxy Z Fold3	12	512	4400	33,990,000

Hình 3: Bảng số liệu thống kê số RAM, bộ nhớ, PIN và giá tiền của 24 chiếc điện thoại.

Hãy sử dụng lý thuyết giải bài toán hồi quy tuyến tính bằng SVD để tìm giá trị dự đoán giá tiền của một chiếc điện thoại mới dựa vào các thông số RAM, bộ nhớ và PIN lần lượt là 4, 64, 4000.

Thuật toán

- Đầu vào: số RAM, bộ nhớ và PIN của một chiếc điện thoại mới.
- Đầu ra: giá tiền của chiếc điện thoại mới.

Các bước làm bài:

1. Khởi tạo ma trận A cấp 24×4 và vector Y cấp 24×1 từ bảng số liệu như sau:

$$A = \begin{bmatrix} 1 & 16 & 3000 & 1 \\ 3 & 32 & 4000 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ 12 & 512 & 4400 & 1 \end{bmatrix} \quad Y = \begin{bmatrix} 1850000 \\ 2650000 \\ \vdots \\ 33990000 \end{bmatrix} \tag{10}$$

Lưu ý: Khuyến khích tạo ma trận từ file.

Thuật toán (tt)

2. Tính định thức của ma trận $A^T A$:

- Nếu định thức khác 0, ma trận X được tính bằng:

$$X = (A^T A)^{-1} A^T Y$$

- Nếu không, thực hiện các bước sau:
  1. Tìm SVD của ma trận A.
  2. Tìm ma trận $\Sigma^+ = diag(\lambda_1^{-1}, \cdots, \lambda_r^{-1}, 0, \cdots, 0)$
  3. Tính X bằng:

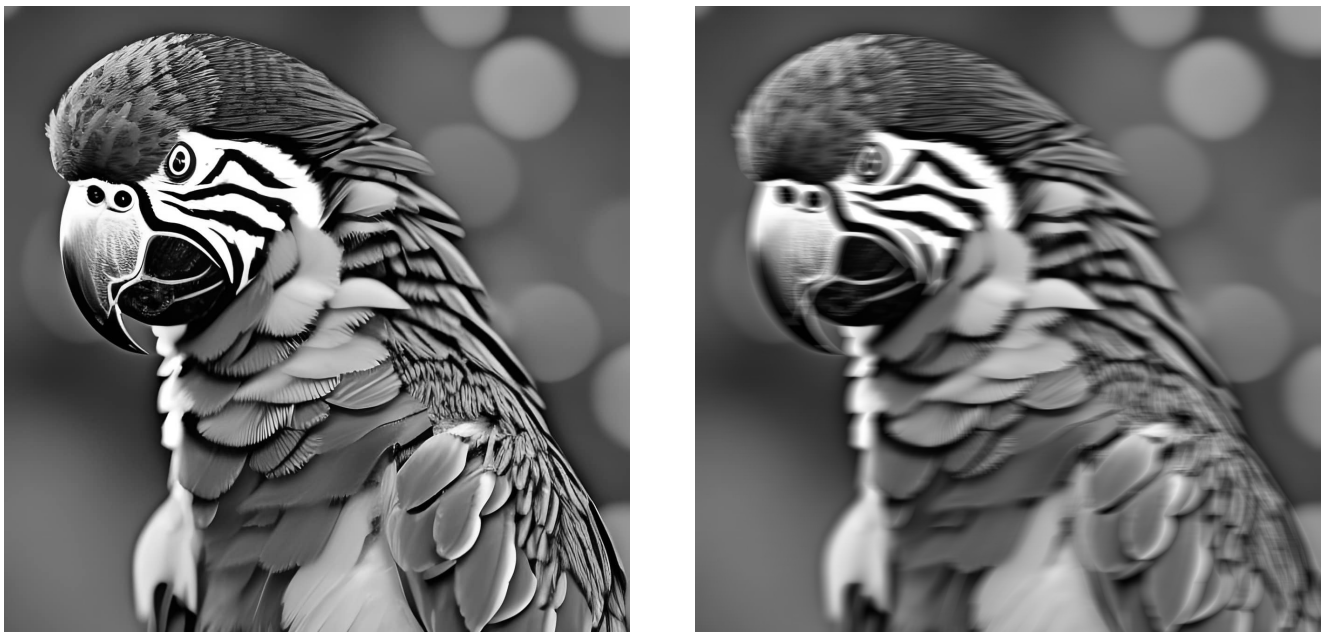
$$X = V \Sigma^+ U^T Y$$

3. Tính giá chiếc điện thoại mới:

$$\omega = (4, 64, 4000, 1) X$$

Ảnh nhòe chuyển động (motion blur)

Ảnh nhòe chuyển động xảy ra khi có một chuyển động có tốc độ không đổi và có hướng cố định giữa máy ảnh và vật.



Hình 4: Ảnh gốc và ảnh nhòe

Ảnh nhòe theo phương ngang và theo phương đứng

Hình 5: Ảnh nhòe theo phương ngang (trái) và ảnh nhòe theo phương đứng (phải)

Cách khôi phục ảnh nhòe bằng SVD

Độ dài của chuyển động nhòe tuyến tính ℓ:

Cách khôi phục ảnh nhòe bằng SVD

Ma trận giảm A**:** Giả sử ảnh nhòe có kích thước $r \times m$ và bị nhòe theo phương ngang. Ma trận giảm A là một ma trận Toepltiz gồm m dòng , $n = m + \ell - 1$ cột. Ma trận A có dòng đầu tiên $(a_{1,j})_{j=1}^{n}$ và cột đầu tiên $(a_{i,1})_{i=1}^{m}$ như sau:

$$a_{i,1} = \begin{cases} \frac{1}{\ell} & i = 1, \\ 0 & i = 2, \ldots, m \end{cases} \tag{11}$$

$$a_{1,j} = \begin{cases} \frac{1}{\ell} & j = 1, \ldots, \ell, \\ 0 & j = \ell + 1, \ldots, n \end{cases} \tag{12}$$

Cách khôi phục ảnh nhòe bằng SVD

Tính chất $A_{i,j} = A_{i+1,j+1}$ của ma trận Toepltiz giúp xác định rõ ràng những phần tử còn lại A.

$$A = \begin{pmatrix} \frac{1}{\ell} & \frac{1}{\ell} & \dots & \frac{1}{\ell} & 0 & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{\ell} & \frac{1}{\ell} & \dots & \frac{1}{\ell} & 0 & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{\ell} & \frac{1}{\ell} & \dots & \frac{1}{\ell} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{\ell} & \frac{1}{\ell} & \frac{1}{\ell} & \dots & \frac{1}{\ell} \end{pmatrix} \tag{13}$$

Cách khôi phục ảnh nhòe bằng SVD

Ví dụ: Từ ảnh nhòe theo phương ngang dưới đây ta có được ma trận B cấp 1344×1344 và độ dài của chuyển động nhòe tuyến tính $\ell = 30$. Vậy ma trận giảm A lúc này sẽ có kích thước 1344×1373:

$$\begin{pmatrix} \frac{1}{30} & \frac{1}{30} & \dots & \frac{1}{30} & 0 & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{30} & \frac{1}{30} & \dots & \frac{1}{30} & 0 & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{30} & \frac{1}{30} & \dots & \frac{1}{30} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \frac{1}{30} & \frac{1}{30} & \frac{1}{30} & \dots & \frac{1}{30} \end{pmatrix}$$

Cách khôi phục ảnh nhòe bằng SVD

Công thức khôi phục ảnh nhòe theo phương ngang:

$$X^+ = B(A^+)^T = B(V\Sigma^+U^T)^T \tag{14}$$

Trong đó:

- X^+ là ma trận của ảnh được khôi phục.
- B là ma trận của ảnh nhòe cấp $r \times m$.
- A^+ là ma trận giả nghịch đảo của ma trận A cấp $m \times n$, với $n = m + \ell - 1$.
- $\Sigma^+ = diag(\lambda_1^{-1}, \ldots, \lambda_r^{-1}, 0, \ldots, 0)$

Cách khôi phục ảnh nhòe bằng SVD

Công thức khôi phục ảnh nhòe theo phương thẳng đứng:

$$X^+ = A^+ B = V \Sigma^+ U^T B \tag{15}$$

Trong đó:

- X^+ là ma trận ảnh khôi phục.
- B là ma trận của ảnh nhòe cấp $r \times m$.
- A^+ là ma trận giả nghịch đảo của ma trận A cấp $r \times n$, với $n = r + \ell - 1$.
- $\Sigma^+ = diag(\lambda_1^{-1}, \ldots, \lambda_r^{-1}, 0, \ldots, 0)$

Bài tập 7

Cho ảnh nhòe theo phương ngang với $\ell = 30$ như dưới đây:

Hình 6: Ảnh nhòe theo phương ngang

Hãy viết chương trình cho phép khôi phục ảnh nhòe phía trên.

Thuật toán

- Đầu vào: Ảnh nhòe theo phương ngang và ℓ.
- Đầu ra: Ảnh đã được khôi phục.

Các bước làm bài:
1. Khai báo các thư viện cần thiết.
2. Dùng hàm numpy.array(Image.open('file ảnh')) để đọc ảnh nhòe và biến hình ảnh thành dạng mảng, đặt là B.
3. Tạo ra ma trận giảm A cấp $m \times n$ từ ℓ bằng công thức (11), (12), (13). (Có thể dùng hàm toepltiz của numpy).
4. Tìm $A^+ = V\Sigma^+ U^T$ từ SVD của A.
5. Tính ma trận X^+ từ công thức (14).
6. Xuất ma trận X^+ dưới dạng ảnh.

Bài tập 8

Cho ảnh nhòe theo phương thẳng đứng với $\ell = 30$ như dưới đây:

Hình 7: Ảnh nhòe theo phương thẳng đứng

Hãy viết chương trình cho phép khôi phục ảnh nhòe phía trên.

Thuật toán

- Đầu vào: Ảnh nhòe theo phương thẳng đứng và ℓ.
- Đầu ra: Ảnh đã được khôi phục.

Các bước làm bài:
1. Khai báo các thư viện cần thiết.
2. Dùng hàm numpy.array(Image.open('file ảnh')) để đọc ảnh nhòe và biến hình ảnh thành dạng mảng, đặt là B.
3. Tạo ra ma trận giảm A cấp $r \times n$ từ ℓ bằng công thức (11), (12), (13). (Có thể dùng hàm toepltiz của numpy).
4. Tìm $A^+ = V\Sigma^+ U^T$ từ SVD của A.
5. Tính ma trận X^+ từ công thức (15).
6. Xuất ma trận X^+ dưới dạng ảnh.

Tham khảo

Sinh viên có thể tham khảo hai bài báo dưới đây để tìm hiểu thêm về khôi phục ảnh nhòe bằng ma trận giả nghịch đảo.

[1] *Application of the pseudoinverse computation in reconstruction of blurred images*.

[2] *Blind image deconvolution: motion blur estimation*.