PROVA 2

SCC0205 - Métodos do Cálculo Numérico I

Amanda Araujo Silva - Nº USP: 10260441

Prof. Antonio Castelo Filho

ICMC-USP (2024)

1 Decomposição SVD: Compressão de imagem

A imagem escolhida para ilustrar o método numérico de decomposição SVD para compressão de imagens foi `'ovo.jpg'`:

Figura 1: Imagem original *Os ovos*. Resolução 1170 x 1549 pixels.

Após a leitura da imagem no MATLAB por meio do comando `imread` que carrega uma imagem representada na forma matricial indexada $[0, 255]$, a imagem original foi transformada de colorida para tons de cinza:

```
>> img = rgb2gray(img);
```

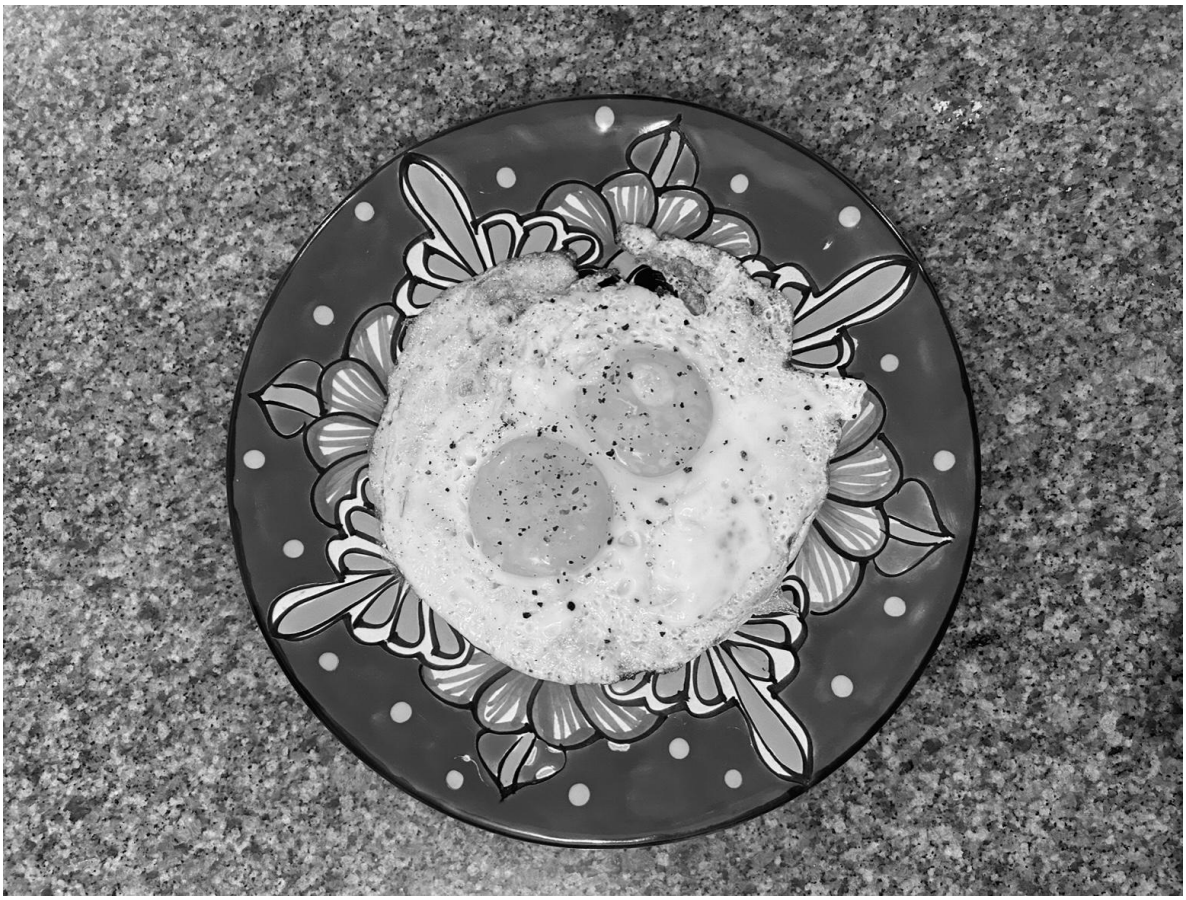

Figura 2: Imagem original em tons de cinza. Resolução 1170 x 1549 pixels.

De modo a permitir o processamento da imagem, foi realizada sua conversão para o formato `double`, transformando-a para valores reais no intervalo $[0, 1]$, pois a função de interesse `svd.m` do MATLAB não aceita outros formatos além de `simple` e `double`:

```
>> img = im2double(img);
```

O próximo passo consistiu na decomposição em valores singulares (SVD):

$$A_{m \times n} = U \Sigma V^T$$

onde U: matriz ortogonal de dimensão $m \times m$, Σ: matriz diagonal $m \times n$ e U: matriz ortogonal de dimensão $n \times n$. A imagem armazenada na matriz `img` de dimensões 1170 x 1549 foi fatorada nas matrizes dadas por U, S, V:

```
>> [U, S, V] = svd(img);
```

`>> size(U)`	`>> size(S)`	`>> size(V)`
ans =	ans =	ans =
1170 1170	1170 1549	1549 1549

Em posse da decomposição SVD da imagem, uma aplicação interessante é a resolução do seguinte problema: dada uma matriz A de posto p (nesse caso, $p = min\{n, m\}$),

qual é a matriz de posto $r < p$ que melhor aproxima A? Esse problema pode ser visto, em outras palavras, como um problema de compressão de imagem.

Solução: realizar a decomposição SVD da matriz `img` e pegar os r primeiros elementos de Σ (no caso, S), ou seja, os r primeiros valores singulares, e as r primeiras colunas de U e V.

```
1     img_r = U(:,1:r)*S(1:r,1:r)*V(:,1:r)';
```

Visualização das imagens comprimidas obtidas a diferentes taxas de compressão:

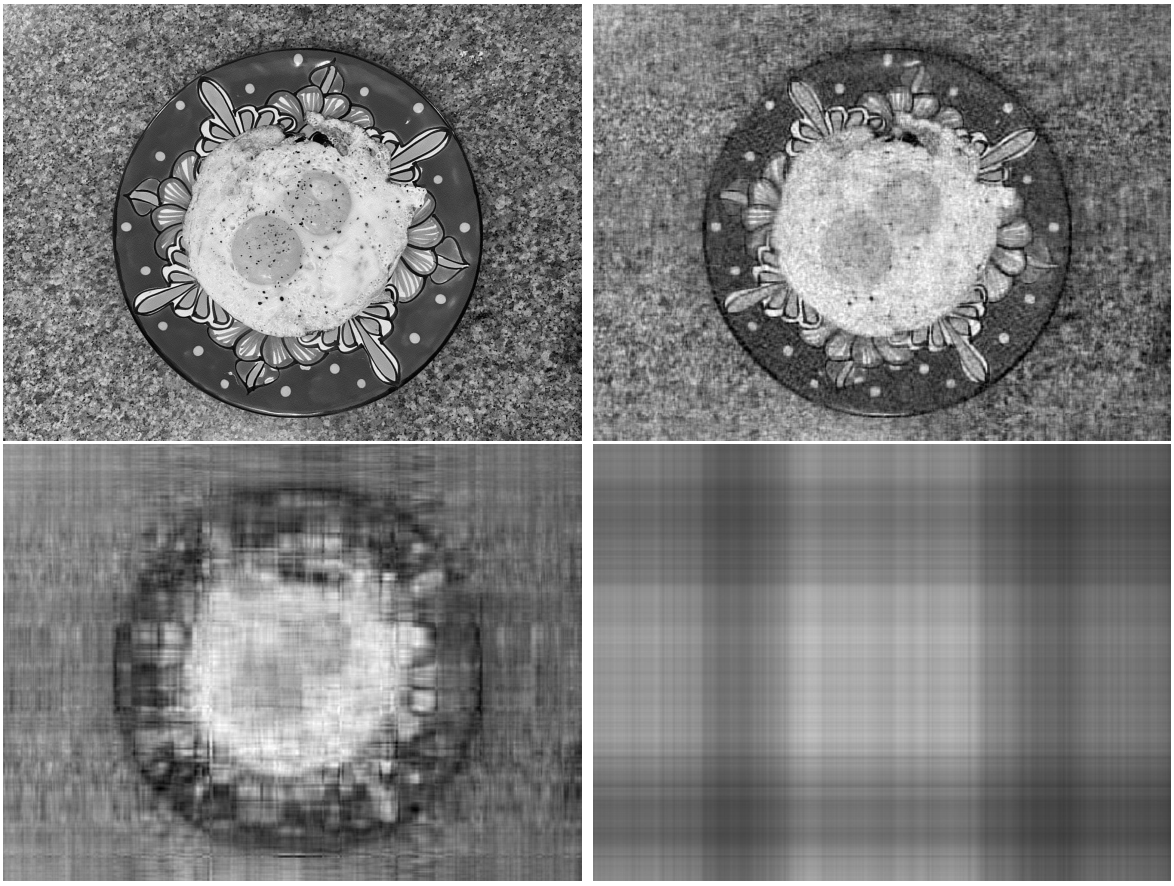

Figura 3: Série *Ovos* de imagens comprimidas obtida para o uso de diferentes quantidades de valores singulares r: 351 (referente a 30% do posto original), 49, 10 e 1, respectivamente (*sentido horário*).

Observa-se que mesmo com um r pequeno em relação ao posto original de 1170, $r = 351$, é gerada uma imagem facilmente identificável com apenas uma leve sensação de pixelamento em relação à original. Reduzindo ainda mais, nota-se para $r = 49$ um ruído visível, que aumenta ainda mais conforme r diminui, chegando até o extremo de um único valor singular $r = 1$ com a perda total da identificabilidade do objeto na figura.

Comparando as imagens obtidas com a original em tons de cinza para garantir a qualidade, foi calculado o erro cometido pela aproximação, através da norma matricial da diferença entre a imagem original e a imagem comprimida:

```
norm(img - img_r)
```

Nº de valores singulares r	Erro
351	3.1010
49	15.6638
10	28.1760
1	126.5818

Tabela 1: Erro obtido pela imagem comprida em aproximar a imagem original

Ao se comparar os erros obtidos, nota-se que o erro cresce com a diminuição do número de valores singulares usados para aproximar a imagem, de acordo com o observado empiricamente com a série *Ovos*. Para garantir a consistência do método, foi testado também a resposta dada a escolha de r igual ao posto da matriz `img` (o máximo possível usando a decomposição SVD), $r = 1170$, obtendo erro de $3.7378e - 12$, na prática zero. Ou seja, resultados de acordo com o esperado teoricamente.

É curioso olhar também para o espaço de armazenamento de cada uma das imagens obtidas *vs.* tamanho da imagem em tons de cinza *double* salva pelo MATLAB de 388KB, algo com implicações práticas de uso de memória:

Nº de valores singulares r	Memória (KB)	Taxa de compressão (%)
351	380	2,1
49	252	35,1
10	151	61,1
1	79	79,6

Tabela 2: Tamanho de armazenamento de memória da série *Ovos*

Nota-se diferentes taxas de compressão, em termos de 1 - razão do uso de memória da imagem comprimida pelo uso de memória da imagem original, cujo valor aumenta conforme se utiliza uma menor quantidade de valores singulares para representar a mesma imagem *Os ovos*. É interessante notar, no entanto, que não há uma relação necessariamente linear entre a quantidade de valores singulares empregados na aproximação e a taxa de compressão final obtida. De modo geral, foi possível ver como um resultado teórico dado pela decomposição SVD de matrizes é capaz de ser aplicada em problemas reais relevantes para o dia a dia, obtendo de maneira bem sucedida uma solução para a quetão de compressão de imagens.

2 PCA: Ajuste de curvas

Análise de componentes principais (PCA) é um procedimento matemático que permite extrair a informação importante dos dados, identificando a base mais significativa para reexpressar o conjunto de dados, filtrando ruído e capturando sua possível estrutura intrínseca. Nesse contexto, melhor expressar os dados está dentro da suposição de que a dinâmica de interesse ou a informação mais importante existe ao longo das direções com maior variância e presumivelmente maior razão sinal-ruído. O processo de PCA computa novas variáveis chamadas componentes principais em ordem crescente, obtidas como combinação linear das variáveis originais, de modo que a primeira componente principal representa a direção de maior variância possível dos dados e assim por diante, sendo as componentes ortogonais entre si.

No caso dos dados fornecidos, um conjunto de pontos em R^2, e do problema de obtenção da melhor reta que ajusta os dados, resolvê-lo por meio de PCA significa considerar apenas a primeira componente principal dos dados, tendo em vista que uma reta é definida por um único vetor (direção). Código em MATLAB da implementação do *fitting* linear dos dados baseado em PCA, calculando o principal componente (associado ao autovalor dominante) utilizando o método das potências e de Francis:

```
% PCA: Analise de Componentes Principais para fitting linear

% Intervalo de interesse
t = 0:pi/200:pi/2;

% Variaveis
x = cos(t);
y = sin(t);

% Criar a matriz A de dados: observacoes em colunas
A = [x; y];

% Centralizar dados
A = A - mean(A, 2); % media ao longo das linhas

% Matriz de covariancia: Cx = AA^T/n
C = A*A'/max(size(x));

% Calcular o autovalor dominante da matriz de covariancia
% Como queremos apenas um fitting linear, usamos apenas o
    autovetor associado ao primeiro autovalor

tol = 0.000001;

% Metodo das Potencias
[eigenvalue, eigenvector, k] = potencias(C, tol);
v_potencias = eigenvector;

% Metodo de Francis
[autovetores, autovalores] = francis(C, tol);
v_francis = autovetores(:, 1); % primeiro autovetor
```

```
31
32      % PLOT
33      hold on
34
35      % Plot dos dados centralizados
36      plot(A(1, :), A(2, :));
37
38      % Variaveis do fitting linear
39      t = t - mean(t); % centralizando intervalo de interesse
40      x_potencias = t * v_potencias(1);
41      y_potencias = t * v_potencias(2);
42      x_francis = t * v_francis(1);
43      y_francis = t * v_francis(2);
44
45      % Plot do fitting linear
46      plot(x_potencias, y_potencias, 'r');
47      plot(x_francis, y_francis, 'b--');
48
49      legend('Dados', 'Fitting Pot ncias', 'Fitting Francis');
50      xlabel('Eixo X');
51      ylabel('Eixo Y');
52      xlim([-0.75, 0.75]);
53      ylim([-0.75, 0.75]);
54      hold off
55
56
57      % FUNCOES
            ------------------------------------------------------------
58      % Metodo das Potencias
59      function [lambda,y,k] = potencias(A,tol)
60          k = 0; kmax = 1000; erro = inf;
61          n = size(A,1); y0 = zeros(n,1); y0(1) = 1;
62          while (erro>tol && k<kmax)
63              x = A*y0;
64              y = x/norm(x);
65              erro = abs(abs(y0'*y)-1);
66              y0 = y; k = k+1;
67          end
68          lambda = y'*A*y;
69      end
70
71      % Metodo de Francis
72      function [V,D] = francis(A,tol)
73          n = size(A,1);
74          V = eye(n);
75          erro = inf;
76          while erro>tol
77              [Q,R] = qr(A);
78              A = R*Q;
79              V = V*Q;
80              erro = max(max(abs(tril(A,-1))));
81          end
82          D = diag(A);
83      end
```

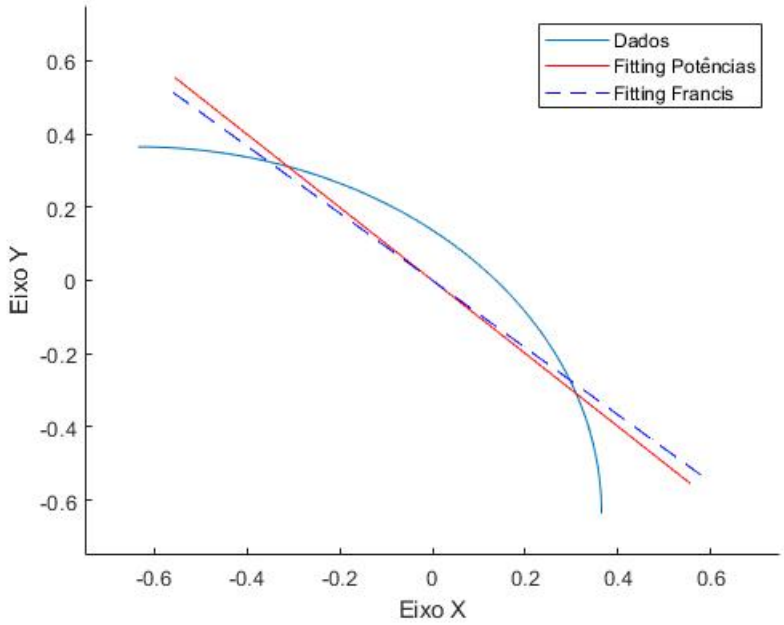

Figura 4: *Fitting* linear dos dados obtidos através de PCA com cálculo dos autovalores e respectivos autovetores realizado pelo método de Potências e método de Francis, sob tolerância `tol` $= 10^{-2}$.

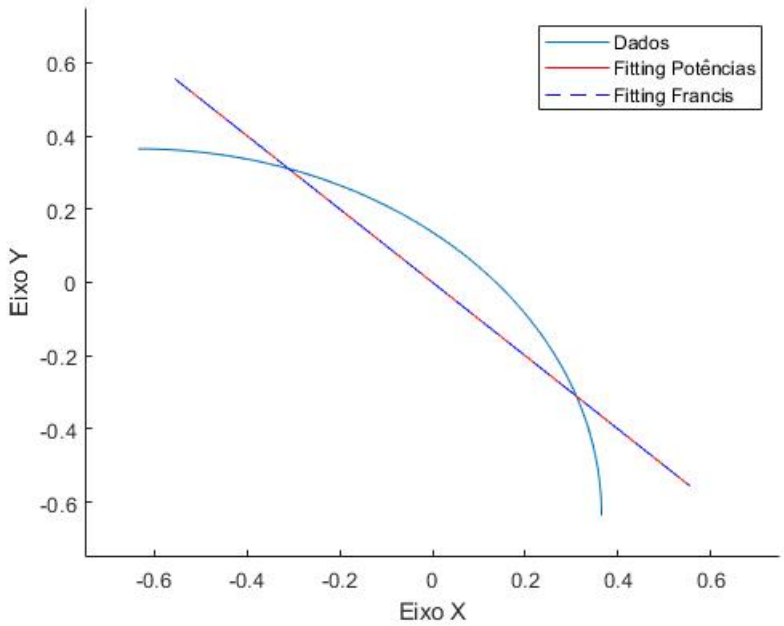

Figura 5: *Fitting* linear dos dados obtidos através de PCA com cálculo dos autovalores e respectivos autovetores realizado pelo método de Potências e método de Francis, sob tolerância `tol` $= 10^{-6}$.

Os gráficos (4) e (5) mostram a reta de melhor ajuste dos dados obtidas sob tolerâncias distintas para os métodos numéricos das potências e de Francis. Observa-se que de fato as retas encontradas capturam a direção de maior espalhamento, *i.e.* de maior variância dos dados, como proposto pelo PCA. Como tratam-se de métodos computacionais, estes estão sujeitos a erros e flutuações, e foi possível notar como ao se considerar uma tolerância maior, `tol =` 10^{-2}, para o cálculo do autovetor (4) os métodos apresentam resultados para o autovetor correspondente à primeira componente principal levemente distintos, enquanto em (5) para um tolerância mais baixa `tol =` 10^{-6} ambos os métodos convergem para a mesma reta.

Cálculo do erro quadrático médio cometido pela melhor aproximação linear dos dados:

```
>> err_potencias = immse(A(2, :), y_potencias)
>> err_francis = immse(A(2, :), y_francis)
```

```
err_potencias =
    0.3981

err_francis =
    0.3981
```

Ou seja, para um valor pequeno do parâmetro de tolerância, ambas as funções para os métodos de potência e de Francis retornaram uma reta que melhor ajusta os dados sujeitos a um mesmo valor de erro (dada a precisão de 4 casas decimais), de modo que ambos os métodos foram capazes de realizar o cálculo do autovetor de interesse e convergiram para o mesmo resultado, como esperado, possibilitando a realização do ajuste da curva com PCA.

Cálculo do maior e menor componente principal:

- Maior componente principal: associada ao maior λ da matriz de covariância;
- Menor componente principal: associada ao menor λ da matriz de covariância;

O método de Francis calcula de uma só vez todos os autovetores e autovalores da matriz dada e obtivemos como saída:

```
autovetores =
 0.7071 0.7071
-0.7071 0.7071

autovalores =
    0.1849
    0.0080
```

O método das potências, por sua vez, retorna o autovalor dominante (maior autovalor) e seu autovetor associado:

```
eigenvector =
    0.7071
   -0.7071

eigenvalue =
    0.1849
```

No entanto, por meio do uso do método das potências invertido, é possível calcular o menor componente principal:

```
% M. Potencia invertido
function [lambda,y,k] = potencia_inv(A,tol,alpha)
    if(nargin==2) alpha = 0; end
    k = 0; kmax = 1000; erro = inf;
    n = size(A,1); y0 = zeros(n,1); y0(1) = 1;
    B = A - alpha*eye(n);
    [L,U] = lu(B);
    while (erro>tol && k<kmax)
        x = U\(L\y0);
        y = x/norm(x);
        erro = abs(abs(y0'*y)-1);
        y0 = y; k = k+1;
    end
    lambda = y'*A*y;
end
```

```
% Menor M. Potencias
[lambda,y,k] = potencia_inv(C,tol, 0);
```

```
y =
    0.7071
    0.7071

lambda =
    0.0080
```

Ao se comparar os dois métodos, considerando o método de Francis *vs.* método das potências e das potências invertido, observa-se que os valores encontrados foram os mesmos: mesmos autovalores associados ao seu mesmo respectivo autovetor, um resultado consistente, como esperado.