Universidade de São Paulo
Instituto de Matemática e Estatistica
IME

EP2 - Laboratório de Métodos Numéricos

Patrícia da Silva Rodrigues (nºUSP 11315590)

Junho
2023

Parte 0 - Implementação do Código

Observações preliminares:

1. O nome dos programas estão diferentes do nome das funções do pdf, pois estava ocorrendo erros por terem nomes iguais. Por isso, decidimos mudar o nome dos programas para Decompress.m, Compress.m e CalculateError.m (primeira letra maiúscula.

2. O parâmetro "method"passado na decompress, estamos considerando como int, onde 1 é a bilinear e 2 é a bicubica.

decompress.m

Para fazer a interpolação dos pontos da imagem descomprimida foram utilizados dois métodos: a interpolação bilinear por partes e a interpolação bicubica. As implementações de ambos os métodos são bastante similares.

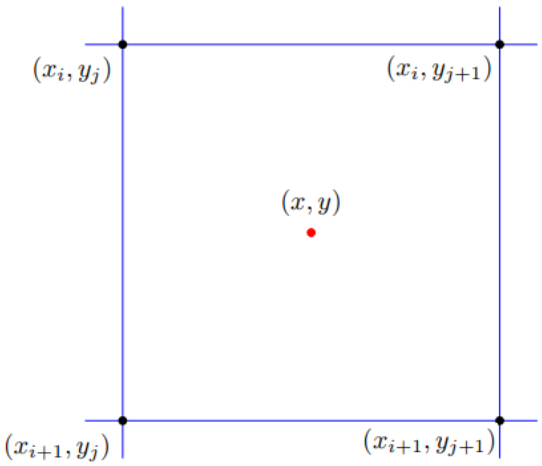
Em primeiro lugar, obtemos quatro pontos conhecidos da matriz descomprimida os quais denominamos $(x_i,\ y_j)$, $(x_i,\ y_{j+1})$, $(x_{i+1},\ y_j)$ e $(x_{i+1},\ y_{j+1})$. Observe que os índices i e j indicam os índices da matriz comprimida (pontos conhecidos) e, convertendo para a matriz descomprimida, temos que

$$x_{i+1} = x_i + k + 1$$

$$y_{j+1} = y_j + k + 1$$

onde k é o número de linhas e colunas adicionas entre cada valor da matriz comprimida.

Representação dos pontos (as coordenadas estão um pouco diferentes do pdf desse EP):



Após obter esses quatro pontos, é possível aplicar o método bilinear e bicubico para calcular os valores que faltam. Vale ressaltar que as interpolações calculadas foram baseadas nas fórmulas dadas no pdf. Assim, iteramos para obter todos os pontos da matriz e para cada quatro pontos obtidos fazemos as interpolações. As implementações de cada método seguem logo abaixo.

1. Interpolação Bilinear por Partes

Funções utilizadas:

- function resposta = Pij_bilinear(A,x,xi,y,yj)
- function decompress(compressedImg, method, k, h)

Dado quatro pontos conhecidos na matriz

$$\underbrace{\begin{bmatrix} f(x_i, y_j) \\ f(x_i, y_{j+1}) \\ f(x_i, y_{j+1}) \\ f(x_{i+1}, y_{j+1}) \end{bmatrix}}_{\mathbf{F}} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & h & 0 \\ 1 & h & 0 & 0 \\ 1 & h & h & h^2 \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}}_{\mathbf{A}} \tag{1}$$

Para isolar a matriz A da equação acima é necessário multiplicar os dois lados por B^{-1} pelo lado esquerdo. Desse modo, teremos a equação $A = B^{-1}.F$

Antes de resolver a multiplicação acima, precisamos encontrar os valores da matriz F utilizando os quatro pontos já conhecidos na matriz. Para isso utilizamos a interação para obter os valores desses pontos. Como a matriz possui 3 dimensões (RGB), então é importante iterar para cada coordenada o eixo z da matriz (terceira coordenada).

Por fim, é realizada uma iteração que percorre os pontos vazios da submatriz e preenchem conforme a fórmula abaixo (utilizada na função Pij_bilinear(A,x,xi,y,yj)):

$$p_{ij}(x,y) = a_0 + a_1(x - x_i) + a_2(y - y_j) + a_3(x - x_i)(y - y_j)$$

2. Interpolação Bicubica

Funções utilizadas:

- function resultado = derivate_fx(x,y,z,M,k,h)
- function resultado = derivate_fy(x,y,z,M,k,h)
- function resultado = derivate_fxy(x,y,z,M,k,h)
- function resposta = Pij(A,x,xi,y,yj)
- function decompress(compressedImg, method, k, h)

Este método apresenta uma implementação bastante similar ao método bilinear, porém envolve mais cálculos para obter os valores da matriz F. Logo abaixo temos a equação que será utilizada no programa:

$$\underbrace{\begin{bmatrix} f(x_i,y_j) & f(x_i,y_{j+1}) & \frac{\partial f}{\partial y}(x_i,y_j) & \frac{\partial f}{\partial y}(x_i,y_{j+1}) \\ f(x_i,y_{j+1}) & f(x_{i+1},y_{j+1}) & \frac{\partial f}{\partial y}(x_i,y_{j+1}) & \frac{\partial f}{\partial y}(x_{i+1},y_{j+1}) \\ \frac{\partial f}{\partial x}(x_i,y_j) & \frac{\partial f}{\partial x}(x_i,y_{j+1}) & \frac{\partial^2 f}{\partial x\,\partial y}(x_i,y_j) & \frac{\partial^2 f}{\partial x\,\partial y}(x_i,y_{j+1}) \\ \frac{\partial f}{\partial x}(x_i,y_{j+1}) & \frac{\partial f}{\partial x}f(x_{i+1},y_{j+1}) & \frac{\partial^2 f}{\partial x\,\partial y}(x_i,y_{j+1}) & \frac{\partial^2 f}{\partial x\,\partial y}(x_{i+1},y_{j+1}) \end{bmatrix}}_{\mathbf{F}} = B \underbrace{\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}}_{\mathbf{A}} B^T \tag{2}$$

onde

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & h & h^2 & h^3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2h & 3h^2 \end{bmatrix} \tag{3}$$

Multiplicando B^{-1} à esquerda e $(B^T)^{-1}$ à direta dos dois lados da equação, então obtemos a equação $A = B^{-1}.F.(B^T)^{-1}$ para encontrar os coeficientes da matriz A. Temos os valores da matriz B e alguns valores da matriz F, porém será necessário calcular as derivadas parciais e mistas da função f. Segue abaixo as fórmulas aplicadas nas funções do programa.

- function resultado = derivate_fx(x,y,z,M,k,h)

$$\frac{\partial f}{\partial x}(x_i,y_j) \approx \frac{f(x_{i+1},y_j) - f(x_{i-1},y_j)}{2h}$$

com as restrições:

$$\frac{\partial f}{\partial x}(x_0,y_j) \approx \frac{f(x_1,y_j) - f(x_0,y_j)}{h}$$

$$\frac{\partial f}{\partial x}(x_{p-1},y_j) \approx \frac{f(x_{p-1},y_j) - f(x_{p-2},y_j)}{h}$$

- function resultado = derivate_fy(x,y,z,M,k,h)

$$\frac{\partial f}{\partial y}(x_i, y_j) \approx \frac{f(x_i, y_{j+1}) - f(x_i, y_{j-1})}{2h}$$

com as restrições:

$$\frac{\partial f}{\partial y}(x_i, y_0) \approx \frac{f(x_i, y_1) - f(x_i, y_0)}{h}$$

$$\frac{\partial f}{\partial y}(x_i, y_{p-1}) \approx \frac{f(x_i, y_{p-1}) - f(x_i, y_{p-2})}{h}$$

• function resultado = derivate_fxy(x,y,z,M,k,h)

$$\frac{\partial^2 f}{\partial x\, \partial y}(x_i, y_j) \approx \frac{\frac{\partial f}{\partial y}(x_{i+1}, y_j) - \frac{\partial f}{\partial y}(x_{i-1}, y_j)}{2 * h}$$

onde a derivada parcial de f em relação a y é calculada chamando a função derivate_fy().

Após encontrar os valores da matriz F, é possível calcular os valores da matriz A e iterar a submatriz para obter os pontos vazios a partir da função abaixo:

$$p_{ij}(x,y) = \begin{bmatrix} 1 & (x-x_i) & (x-x_i)^2 & (x-x_i)^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ (y-y_j) \\ (y-y_j)^2 \\ (y-y_j)^3 \end{bmatrix} \tag{4}$$

compress.m

Funções utilizadas:

- function calculateError(originalImg, decompressedImg)
- function compress(originalImg, k)

Esta função tem como objetivo retirar algumas linhas e colunas da matriz de originalImg (imagem). Para isso, é passado um valor k nos parâmetros tal que obedeça a equação $p = n + (n-1)k$, onde p é o número de pixels da imagem original, n é o número de pixels da imagem comprimida e k é um valor inteiro.

Além disso, as linhas e colunas que serão mantidas são aquelas com o índice i tal que i $\equiv$ 0 mod k+1. Desse modo, é possível gerar a imagem comprimida por iteração dos índices da matriz. Vale ressaltar que é importante iterar para cada matriz RGB (coordenada z).

calculateError.m

Funções utilizadas:

- function calculateError(originalImg, decompressedImg)

Esta função basicamente calcula o erro entre a imagem original e a imagem que foi comprimida e descomprimida, voltando ao mesmo tamanho da imagem original.

Inicialmente, obtemos as matrizes origR, origG e origB de originalImg e, obtemos decR, decG e decB de decompressedImg. Após isso, calculamos errR, errG e errB com a seguinte equação:

$$errX = \frac{|origX - decX|}{|origR|}$$

onde X = R, G ou B.

Por fim, calculamos o erro:

$$err = \frac{errR + errG + errB}{3}$$

Parte 1 - O Zoológico

A função $f(x, y) = (sen(x), sen(y) + sen(x), sen(x))$ gerou a seguinte imagem:
Imagem gerada pela função:

Figura 1: Gráfico de $f(x, y) = (sen(x), sen(y) + sen(x), sen(x))$

Imagem ampliada em 1 vezes (k = 1) usando a interpolação bilinear (descompress):

Figura 2: Gráfico de $f(x, y) = (sen(x), sen(y) + sen(x), sen(x))$ ampliada

Agora iremos pegar a imagem original testar o método de compressão e depois de descompressão bilinear e calcular o erro.

Imagem comprimida 1 vez (k = 1) usando a interpolação bilinear(compress):

Figura 3: Gráfico de $f(x, y) = (sen(x), sen(y) + sen(x), sen(x))$ comprimida 1 vez (k = 1)

Agora iremos pegar a imagem acima descomprimi-la e ela deve voltar a ter a dimensão original, porém com distorções na resolução. Esse erro gerado será calculado a seguir. **Imagem comprimida 1 vez (k = 1) usando a interpolação bilinear(compress):**

Figura 4: Figura 3 descomprimida (k = 1) bilinear

Imagens/descompressed2.png

Figura 5: Descompimida usando a bicubica

Obbs: O erro está mais a baixo Podemos observar que a imagem ficou quadriculada, ou seja, não ficou suave como antes. Isso aconteceu porque, ao comprimir a imagem, perdemos alguns pontos que a figura tinha que a tornava tão suave e fidedigna a função f(x,y). Essa perda fez diferença quando tentamos expandi-la novamente. Os pontos que foram perdidos ao comprimir foram interpolados ao descomprimir e essa interpolação foi baseada nos pontos que haviam ao redor e claro esses novos pontos não correspondem exatamente aos de antes, apesar de se aproximarem. Foi possível observar também que, por se tratar de uma imagem suave (e com suave quero dizer sem grandes mudanças de tons em poucos pixels), a interpolação conseguiu se aproximar visualmente mais da imagem original do que em fotos reais, por exemplo, onde diversos tons geralmente estão presentes.

Testando com funções diversas para a interpolação Bilinear

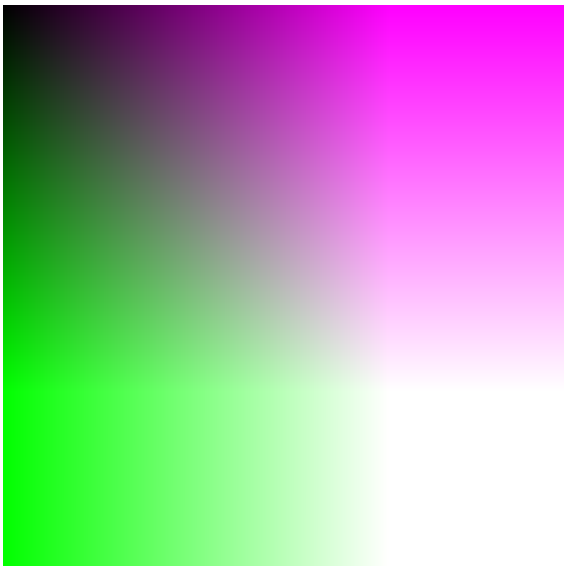


Figura 6: Gerada pela equação $f(x, y) = (x + y, x, y)$

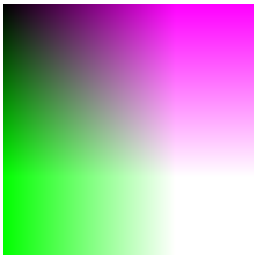


Figura 7: Comprimida

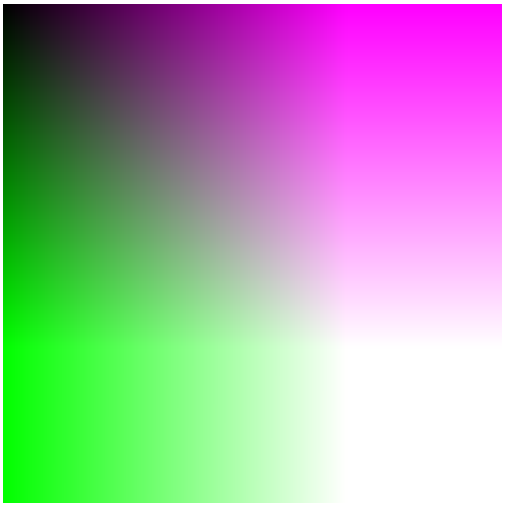


Figura 8: Descomprimida usando bilinear

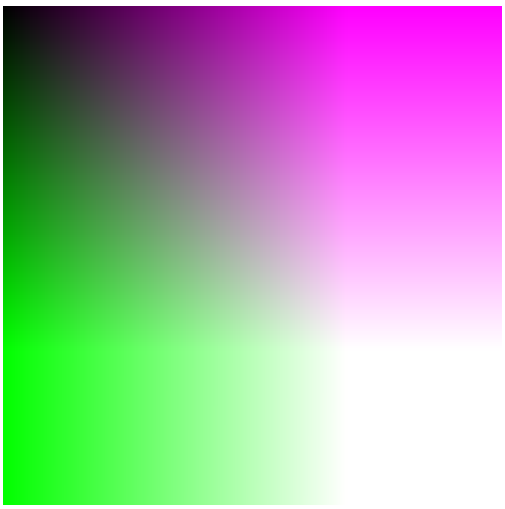


Figura 9: Descomprimida usando bicubica

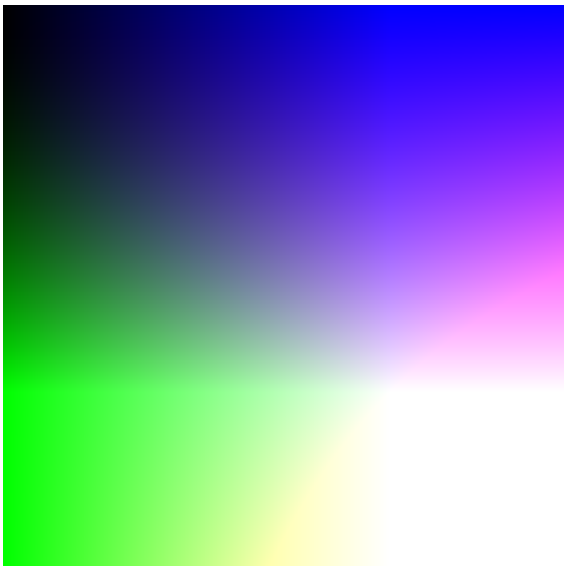


Figura 10: Gerada pela equação $f(x, y) = (xy, x, y)$

Figura 11: Comprimida

Figura 12: Descomprimida usando bilinear

Figura 13: Descomprimida usando bicubica

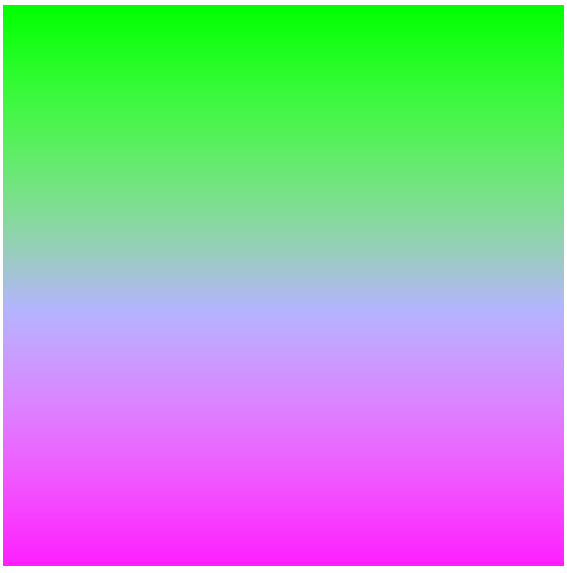


Figura 14: Gerada pela equação $f(x, y) = (sen(x), cos(x), tan(x))$

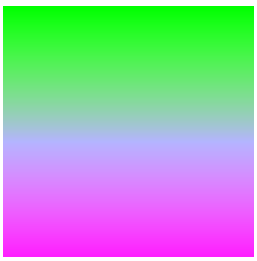


Figura 15: Comprimida

Figura 16: Descomprimida usando bilinear

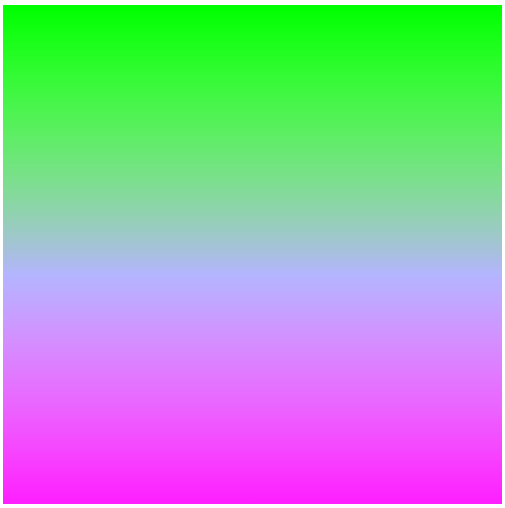


Figura 17: Descomprimida usando bicubica

Respondendo tópicos

• Funciona bem para imagens preto e branco?

Resposta: Depende, pois para que a interpolação funcione é importante que a matriz da imagem tenha x,y,3 dimensões (seja RGB), pois a interpolação trabalha nesses três níveis. Para o caso de imagens preto e branco, não é sempre que a imagem segue o padrão RGB. Logo, o programa pode até funcionar caso a imagem seja RGB, mas dependendo da imagem, não funcionará.

• Funciona bem para imagens coloridas?

Resposta: Sim, as imagens coloridas ficam aparentemente boas depois de comprimidas e interpoladas de acordo com os testes realizados. Além disso, as imagens coloridas garantem que a matriz imagem tenha 3

dimensões (RGB).

• Funciona bem para todas as funções de classe C^2?

Resposta: Para os testes realizados, aparentemente funciona, pois as imagens são geradas sem distorções.

• E para funções que não são de classe C^2 ?

Figura 18: Comprimida

Resposta: Aparentemente funciona sim.

• Como o valor de h muda a interpolação?
R: A imagem escurece quando conforme aumentando o h. Vai ficando pálida até mesmo fica preta e branca. Como no exemplo abaixo para h = 3.

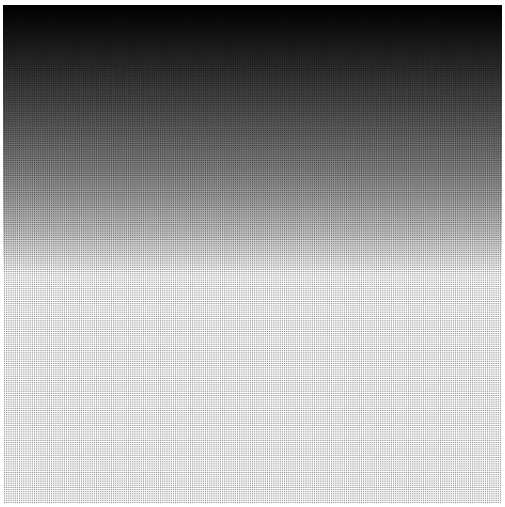


Figura 19: Comprimida

• Como se comporta o erro?
R: O erro, ao comprimir a sen e depois descomprimir usando o metodo bilinear é 0.001426 R: Usando a bi-cubica é 0.0021745

Responda também a esta questão:

Parte 2 - A Selva

Testes com imagens reais
Imagem preto e branco

Figura 20: Imagem em preto e branco

Figura 21: Comprimida

Figura 22: Descomprimida usando bilinear. Podemos observar aqui que para imagens pretas a interpolação bilinear não funciona tão bem e isso se acontece porque para gerar imagens pretas não utilizam geralmente o sistema RGB e isso no caso afeta bastante a interpolação bilinear

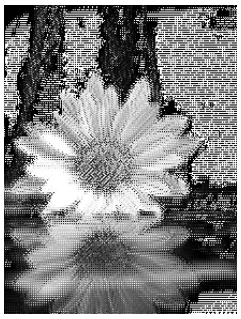


Figura 23: Descomprimida usando bi-cubica. Da mesma maneira e por razões parecidas a interpolação bi-cubica também não é boa para descomprimir uma imagem em preto e branco

Figura 24: Imagem colorida

Figura 25: Comprimida

Figura 26: Descomprimida usando bilinear

Figura 27: Descomprimida usando bicubica. Podemos observar que a resolução da bicubica é menos suave que a da interpolação bilinear

A interpolação bilinear deixa as imagens mais suaves, já a bicubica ficaram mais quadriculadinhas, menos suave.

Respondendo tópicos

• Funciona bem para imagens preto e branco?

Resposta: É a mesma razão da Parte 1 - Zoológico: Depende, pois para que a interpolação funcione é importante que a matriz da imagem tenha x,y,3 dimensões (seja RGB), pois a interpolação trabalha nesses três níveis. Para o caso de imagens preto e branco, não é sempre que a imagem segue o padrão RGB. Logo, o programa pode até funcionar caso a imagem seja RGB, mas dependendo da imagem, não funcionará.

• Funciona bem para imagens coloridas?

Resposta: É a mesma razão da Parte 1 - Zoológico: Sim, as imagens coloridas ficam aparentemente boas depois de comprimidas e interpoladas de acordo com os testes realizados. Além disso, as imagens coloridas garantem que a matriz imagem tenha 3 dimensões (RGB).

• Funciona bem para todas as funções de classe C^2?

Resposta: Para os testes realizados, aparentemente não funciona, pois as imagens são reais e geralmente não seguem uma função pré determinada.

• E para funções que não são de classe C^2 ?

Resposta: As imagens são reais, isto é, não obedecem uma função na maioria das vezes.

• Como o valor de h muda a interpolação?

R: A imagem pode distorcer mais conforme muda o h.

• Como se comporta o erro?

Resposta: O erro da bicubica geralmente é maior que o erro da bilinear.