**Nome: Gabriel Ferreira de Souza Araujo** **NUSP: 12718100**

# Relatório EP3 - Laboratório Numérico

## 1. CÓDIGO

Foram feitos quatro programas: “compress.m”, “decompress.m”, “get_func.m” e “calculateError.m”. O programa “compress” tem com ofunção principal a compressão de uma imagem, os parâmetros dessa função são o nome da imagem e um valor para taxa de amostragem “k”. O programa “decompress” tem funções que expande uma imagem para o tamanho original obedecendo a lei $p = n + (n - 1) * k$. Os parâmetros dessa função são o nome da imagem descomprimida, a taxa de amostragem “k”, a altura “h”, e o número do método (bilinear = 1 e bicubico = 2). O programa get_func gera as imagens do zoológico modificando o parametro “p” (tamanho da imagem) e o “funcao” (número da funcao que deseja rodar). Para rodar os programas, basta inserir os parâmetros da forma correta com um determinado “k” para cada imagem presente nos testes enviados, também, as imagens testes precisam estar no mesmo diretório dos programas e com o nome igual ao indicado no código (“imagem” para comprimir).

## 2. DECOMPRESS

### 2.1 Interporlação Bilinear Por Partes

A função principal é chamada **bilinear_interpolation** e a primeira parte do código extrai os canais de cor (vermelho, verde e azul) da imagem comprimida usando a indexação **(:,:,1), (:,:,2) e (:,:,3).** Esses canais são armazenados nas variáveis imgR, imgG e imgB, respectivamente.

O código entra em um loop while aninhado para percorrer os pixels da imagem original (imgR, imgG e imgB). O loop externo percorre as linhas da imagem original, enquanto o loop interno percorre as colunas. As variáveis p e q são usadas para rastrear a posição atual nos loops.

Dentro do loop, o código chama a função **_bilinear_interpolation** para realizar a interpolação bilinear em uma vizinhança 2x2 de pixels na imagem original. A função **_bilinear_interpolation** recebe o fator de interpolação k e um subconjunto da imagem original **imgR(p:p+1, q:q+1), imgG(p:p+1, q:q+1) e imgB(p:p+1, q:q+1).**

A função _bilinear_interpolation realiza a interpolação bilinear entre os quatro pixels da vizinhança. Inicialmente, os valores dos pixels são organizados em um vetor coluna usando a função reshape. Em seguida, uma matriz M é definida para realizar a interpolação. Os coeficientes de interpolação são calculados multiplicando a matriz M pelo vetor coluna dos valores dos pixels.

A variável interpolation é inicializada como uma matriz de zeros com tamanho **(k+2) x (k+2)** para armazenar a imagem interpolada.

Em seguida, o código entra em outro loop while aninhado para percorrer as células da matriz interpolation. O loop externo percorre as linhas, enquanto o loop interno percorre as colunas. As variáveis p e q são usadas para rastrear a posição atual nos loops.

Dentro do loop, o código calcula as coordenadas normalizadas x e y dos pontos de interpolação. Essas coordenadas variam de 0 a 1. Em seguida, um polinômio é construído usando as coordenadas x e y como base para a interpolação. O vetor de coeficientes a calculado anteriormente é multiplicado pelo polinômio para obter o valor interpolado do pixel. O valor interpolado é atribuído à posição correspondente na matriz interpolation.

Depois que todos os pixels da imagem original são interpolados e armazenados na matriz interpolation, a função **_bilinear_interpolation** retorna a matriz resultante. De volta à função **bilinear_interpolation**, a matriz interpolation resultante é atribuída à posição correta na matriz reconstructedImg. A indexação **(p*(k+1))-k:(p+1)*(k+1)-k, (q*(k+1))-k:(q+1)*(k+1)-k** é usada para calcular as posições corretas onde os pixels interpolados devem ser colocados na matriz reconstructedImg.

O loop continua até que todas as células da imagem original sejam percorridas, incrementando os valores de p e q a cada iteração. Após o término do loop, a função bilinear_interpolation retorna a matriz reconstructedImg, que contém a imagem reconstruída.

### 2.2 Interpolação Bicubica

A função principal é chamada **bicubic_interpolation,** o código atribui a matriz compressedImg à variável img. Em seguida, são criadas as matrizes img_dx, img_dy e img_dxdy chamando as funções get_img_dx, get_img_dy e get_img_dy passando a compressedImg como argumento. Essas funções calculam as derivadas parciais da imagem compressedImg ao longo dos eixos x e y.

A variável fsz é calculada como o tamanho da imagem reconstruída, levando em consideração o fator de interpolação k. A fórmula utilizada é **sz(1) + (sz(1)-1)*k,** que aumenta a altura e largura da imagem original como segue a lei **p = n + (n-1)*k**. A matriz reconstructedImg é inicializada como uma matriz de zeros com o tamanho fsz para armazenar a imagem reconstruída. Em seguida, o código entra em um loop while aninhado para percorrer os pixels da imagem original (img, img_dx, img_dy e img_dxdy). O loop externo percorre as linhas da imagem original, enquanto o loop interno percorre as colunas. As variáveis p e q são usadas para rastrear a posição atual nos loops. Dentro do loop, o código constrói uma matriz f que contém os valores dos pixels e suas derivadas parciais em uma vizinhança 2x2. A função cat é usada para concatenar as matrizes dos pixels e suas derivadas parciais nas dimensões corretas.

Em seguida, a função **_bicubic_interpolation** é chamada para realizar a interpolação bicúbica na matriz f. A interpolação é realizada separadamente para cada canal de cor (vermelho, verde e azul). Os resultados da interpolação são armazenados nas variáveis rimgR, rimgG e rimgB. Os valores interpolados são atribuídos às posições correspondentes na matriz reconstructedImg usando a indexação **reconstructedImg(p*(k+1):(p+1)*(k+1), q*(k+1):(q+1)*(k+1), :).**

Após atribuir os valores interpolados, as variáveis q são incrementadas para passar para a próxima coluna. Quando todas as colunas são percorridas no loop interno, as variáveis q são redefinidas para 1 e a variável p é incrementada para passar para a próxima linha.

O loop externo e o loop interno continuam até que todas as células da imagem original tenham sido percorridas. Finalmente, a função retorna a matriz reconstructedImg, que contém a imagem reconstruída. A segunda função, _bicubic_interpolation, realiza a interpolação bicúbica entre quatro pixels. A matriz de coeficientes M é definida, representando a matriz de interpolação bicúbica. A matriz f é multiplicada por M para obter os coeficientes intermediários. Em seguida, a matriz de coeficientes a é calculada multiplicando M pela transposta de f. Essa multiplicação completa o processo de interpolação bicúbica. Uma matriz interpolation é inicializada como uma matriz de zeros para armazenar os valores interpolados.

Os loops while percorrem as posições p e q na matriz interpolation, assim como na função **bicubic_interpolation**. Dentro do loop, as coordenadas normalizadas x e y são calculadas. Um vetor X é construído com os polinômios de base para a interpolação bicúbica na direção x, e um vetor Y é construído para a direção y. A interpolação bicúbica é calculada multiplicando X, a e Y e atribuindo o resultado à posição correspondente na matriz interpolation. Depois de percorrer todas as células da matriz interpolation, a matriz é transposta e, em seguida, a função retorna a matriz interpolation.

## 3. A SELVA

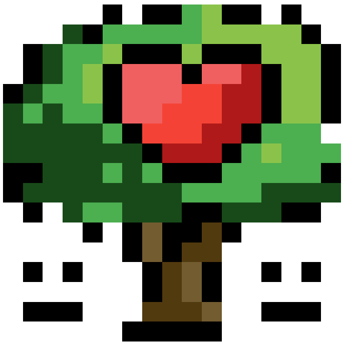

Figura 1: Original 17x17x3

Figura 1: Imagem Original 5x5x3 com K = 3

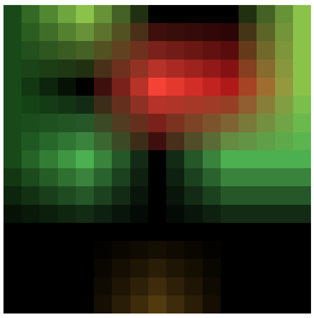

Figura 1: Imagem Descomprimida (I. Bilinear) 17x17x3 com K= 3 e Erro = 0.5319

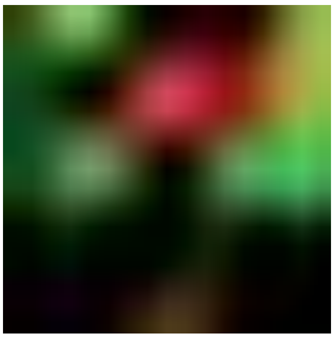

Figura 1: Imagem Descomprimida (I. Bicúbica) 17x17x3 com K= 3 e Erro = 0.5601

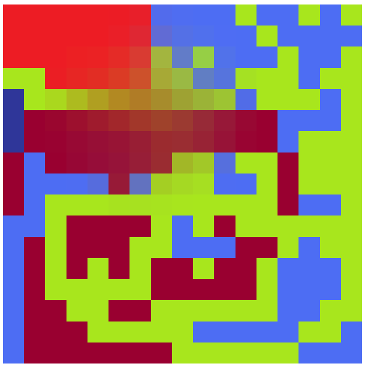

Figura 2: Imagem Original 17x17x3

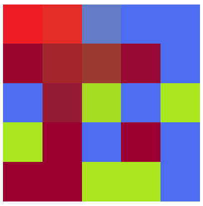

Figura 2: Imagem Comprimida 5x5x3 com K = 3

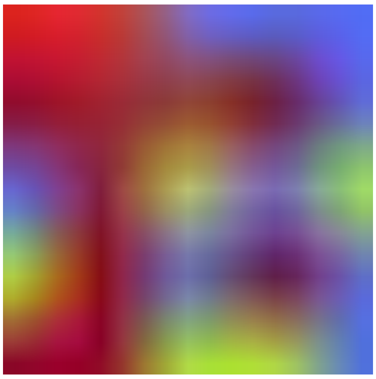

Figura 2: Figura 2: Imagem Descomprimida (I. Bilienear) 17x17x3 com K = 3 e Erro = 0.3653

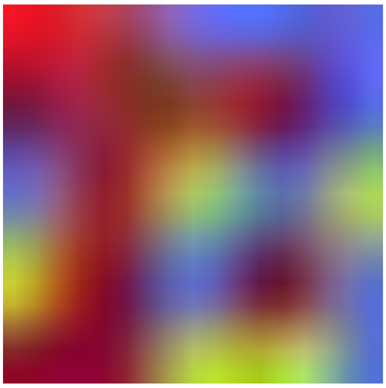

Figura 2: Imagem Descomprimida (I. Bicúbica) 17x17x3 com K = 3 e Erro = 0.3951

Figura 3: Imagem Original 49x49x3

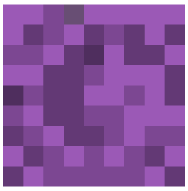

Figura 3: Imagem Comprimida 9x9x3 com K = 5

Figura 3: Imagem Descomprimida (I. Bilinear) 49x49x3 com K = 9 e Erro = 0.1015

Figura 3: Imagem Descomprimida (I. Cúbica) 49x49x3 com K = 9 e Erro = 0.1114

Figura 4: Imagem Original 105x105x3

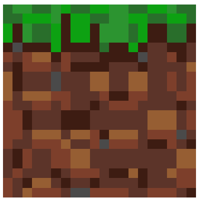

Figura 4: Imagem Comprimida 14x14x3 com K = 7

Figura 4: Imagem Descomprimida (I. Bilinear) 105x105x3 com K= 7 e Erro = 0.1851

Figura 4: Imagem Descomprimida (I. Bicúbica 105x105x3 com K= 7 e Erro = 0.1898

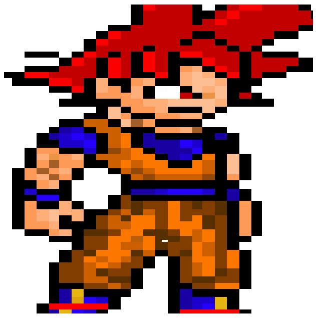

Figura 5: Imagem Original 151x151x3

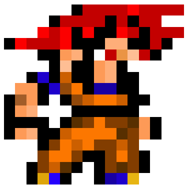

Figura 5: Imagem Comprimida 16x16x3 com K = 9

Figura 5: Imagem Descomprimida (I. Bilinear) 151x151x3 com K = 9 e Erro = 0.2462

Figura 5: Imagem Descomprimida (I. Bicúbica) 151x151x3 com K = 9 e Erro = 0.2549

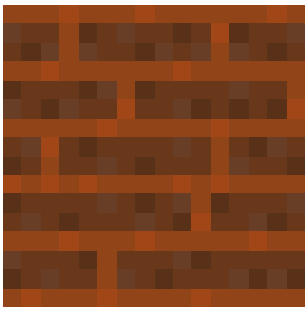

Figura 6: Imagem Original 151x151x3

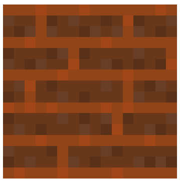

Figura 6: Imagem Comprimida 151x151x3 com K = 9

Figura 5: Imagem Descomprimida (I. Bilinear) 151x151x3 com K = 9 e Erro = 0.074855

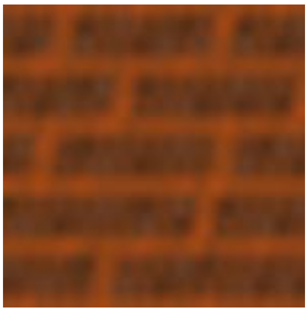

Figura 5: Imagem Descomprimida (I. Bicúbica) 151x151x3 com K = 9 e Erro =  0.074499

## 3.1 Questões

**1. Funciona bem para Imagens preto e branco?** R: Como o método de interpolação desenvonve-se com a manipulação de uma matriz com x, y, 3 dimenões (RGB), então para imagens preto e branco, não é garantido que nosso programa funcionará caso a imagem não siga o padrão RGB.

**2. Funciona bem para imagens coloridas?** R: Sim, para imagens coloridas o método da interpolação é capaz de gerar uma aproximação boa para descompressão, mas nem sempre é a ideal, pois isso depende o grau de amostragem da imagem. Vemos nos exemplos que quanto menor a imagem (menor amostra) temos um erro de descompressão maior, além disso, para imagens que possuem uma alta dispersão de pixels o erro também aumenta.

**3. Funciona bem para todas as funções de classe $C^2$?** R: Como as imagens são reais e não seguem uma função específica, temos que essa afirmação é falsa.

**4. E para funções que não são de classe $C^2$?** R: Novamente, as imagens não garantem uma função definida, portanto, pode ou não uma funcionalidade para esse caso.

**5. Como o valor de h muda a interpolação?** O “h” pode interfirir na distorção da imagem.

**6. Como se comporta o erro?** R: Com os exeplos acima, percebemos que geralmente, a interpolação bicúbica tem um erro amior do que a interpolação bilinear por partes, e nota-se que quanto menor a taxa de descompressão “k” , maior o tamanho da imagem e menor dispersão de pixels, menor o erro.

# 4. ZOOLÓGICO

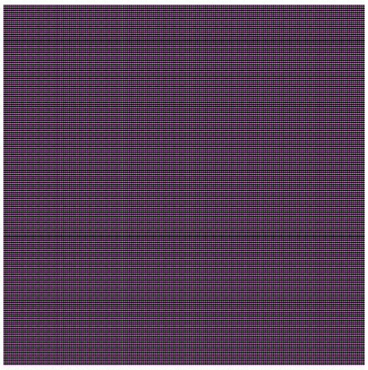

Figura 7: Imagem original da função $f(x, y) = \left( sen(x),\ \frac{(sen(x)+sen(y))}{2},\ sen(x) \right)$

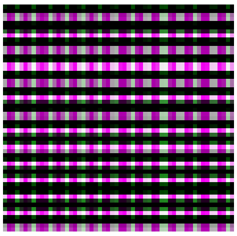

Figura 7: Imagem compimida com k = 9

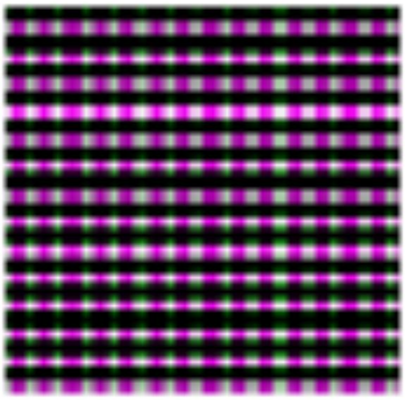

Figura 7: Imagem Descomprimida com k = 9

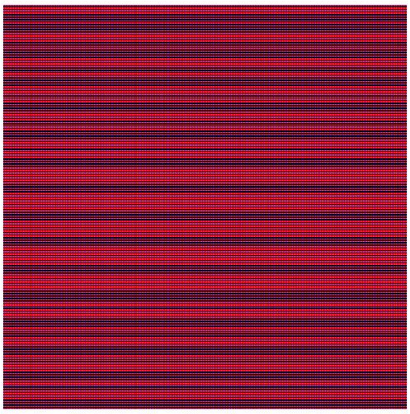

Figura 8: Imagem original da função: $f(x, y) = \left(sen(x) \cdot x,\ \frac{(sen(x)+sen(y))}{8},\ sen(x) + sen(y)\right)$

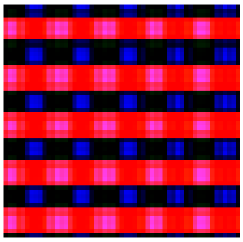

Figura 8: Imagem compimida com k = 9

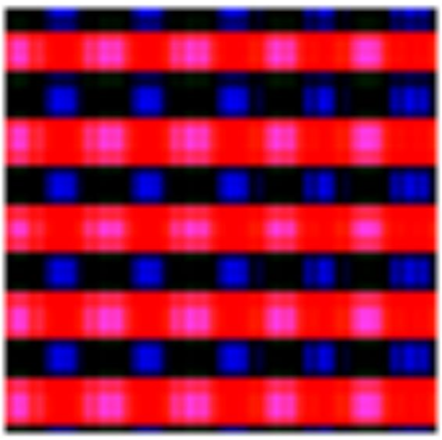

Figura 8: Imagem Descomprimida com k = 9

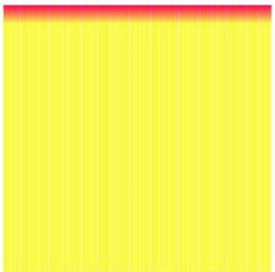

Figura 9: Imagem original da função: $f(x,y) = \left(80x^2,\ \frac{x}{70},\ \cos(y)\right)$



Figura 9: Imagem compimida com k = 7

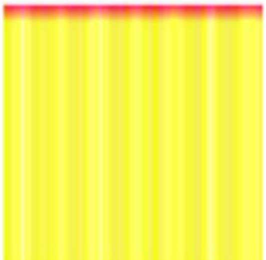

Figura 9: Imagem Descomprimida com k = 7

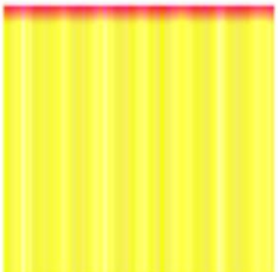

Figura 9: Imagem Descomprimida 3 vezes com k = 1

### 4.1 Questões

**1. Funciona bem para Imagens preto e branco?** R: Como o método de interpolação desenvonve-se com a manipulação de uma matriz com x, y, 3 dimenões (RGB), então para imagens preto e branco, não é garantido que nosso programa funcionará caso a imagem não siga o padrão RGB.

**2. Funciona bem para imagens coloridas?** R: Sim, para imagens coloridas o método da interpolação é capaz de gerar uma aproximação boa para descompressão, mas nem sempre é a ideal, pois isso depende o grau de amostragem da imagem. Vemos nos exemplos que quanto menor a imagem (menor amostra) temos um erro de descompressão maior, além disso, para imagens que possuem uma alta dispersão de pixels o erro também aumenta.

**3. Funciona bem para todas as funções de classe $C^2$?** R: Ao analisar os exemplos, onde temos funlções de classe C2, o método aparente funcionar.

**4. E para funções que não são de classe $C^2$?** R: Para funções de outras ordens, ao fazer alguns exemplos, tudo aparenta funionar.

**5. Como o valor de h muda a interpolação?** Conforme aumenta “h”, nota-se que a imagem tende a escurecer e aumentar os tons de preto.

**6. Como se comporta o erro?** R: Com os exeplos acima, percebemos que geralmente, a interpolação bicúbica tem um erro amior do que a interpolação bilinear por partes, e nota-se que quanto menor a taxa de descompressão “k” , maior o tamanho da imagem e menor dispersão de pixels, menor o erro.

**7. Coparação entre comprimir imagem com k=7 e comprimir 3 vezes com k=1:** Ao comparar as duas imagens resultantes, temos que os resultados são muito próximos, mostrando o quanto uma baixa taxa de compressão pode resultar numa boa qualidade de paroximação em comparação com altas taxas “k”.