Nomes: Cainã Setti Galante **Nº USP:** 10737115
Rubens Gomes Neto 09318484

Exercício-Programa II de MAC0210

1 Parte 0 - Laboratório

Nesta parte do problema tivemos que implementr as funções que servirão para o estudo da interpolação de polinômios aplicados a imagens. As decisões de projeto e detalhes da implementação estão descritos abaixo.

Vale lembrar que as funções trabalham com imagens com três canais de cores. Portanto, quando nos referirmos a um ponto de uma matriz, na verdade estaremos nos referindo a um vetor de três coordenadas, onde cada coordenada representa uma cor. As matrizes que usamos têm três dimensões.

1.1 compress.m

Esse arquivo contém apenas a função `compress`, com o seguinte protótipo:

```
function compress (originalImg, k)
```

Ela recebe um arquivo de imagem no formato *png* e devolve, em um outro arquivo *png*, a imagem comprimida com a taxa k.

A leitura da imagem recebida é armazenada em uma matriz grande. Comprimimos retirando todas as linhas e colunas i tal que $i\%(k+1) = 1$, onde % representa a operação de resto.

A compressão é feita selecionando os pontos que possuem um par linha, coluna que satisfazem os requisitos, o atribuímos à matriz reduzida. O ponto (x, y) da matriz grande é colocado no seguinte ponto da matriz reduzida: $(\left\lfloor \frac{x}{k+1} + 1 \right\rfloor, \left\lfloor \frac{y}{k+1} + 1 \right\rfloor)$.

Feito isso, a matriz pequena é escrita numa imagem *png*.

1.2 calculateError.m

O arquivo possui apenas a função `calculateError`, a qual tem o protótipo:

```
function calculateError(originalImg, decompressedImg)
```

Essa função calcula o erro relativo entre duas imagens usando as fórmulas fornecidas no enunciado (aqui, usamos para a imagem comprimida e uma imagem descomprimida).

Primeiramente, lemos as imagens e armazenamos em matrizes, então, fazemos a conta. O único detalhe da implementação é que usamos a função `norm` do Octave para calcular a norma euclidiana.

1.3 decompress.m

Esse é o arquivo mais importante dos três enviados. Ele faz a descompressão de uma imagem usando algum método de interpolação e nos devolve o arquivo com a imagem descomprimida. Ele possui as seguintes funções, com os protótipos:

```
function decompress (compressedImg, method, k, h)
```

A função `decompress` recebe uma imagem em *png* e a descomprime em uma razão k, utilizando o método bilinear ou o método bicúbico.

Ela lê a imagem e a armazena em uma matriz. A descompressão será feita inserindo k linhas e colunas entre as linhas e colunas da matriz. Calculamos o tamanho da p da imagem descomprimida usando a fórmula $p = n + (n - 1) \cdot k$, onde n é o tamanho da matriz da imagem pequena.

Dependendo do método escolhido, ela chama a função que desenvolverá o método da interpolação, que devolverá uma matriz com os pontos interpolados.

Feito isso, essa matriz é escrita em um arquivo *png*.

```
function B = bilinear (A, k, h, p)
```

Esse representa um dos métodos de interpolação descrito no enunciado, o método Bilinear por partes.

Para começar, chamamos a função `expande`, que nos devolve uma matriz com o tamanho que precisamos (mais detalhes abaixo).

Com essa matriz definimos quadrados de lado $k + 2$ e, então, armazenamos os vértices do quadrado. Usamos `X = inv(A)*B` para resolver o sistema linear do método da interpolação (no enunciado está na forma B = AX). Com a matriz X dos valores da solução, fazemos a interpolação para cada cor de todos os pontos dentro de cada quadrado definido, usando a fórmula dada:

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

No entanto, fizemos pequenas adaptações para a implementação funcionar:

`f = X(1) + X(2).*x + X(3).*y + X(4).*x.*y;`, onde:

- `f` é o resultado do polinômio interpolador.

- `X(i)` é o correspondente a a_{i-1}, proveniente da solução do sistema linear.
- `x` e `y` são as coordenadas do ponto no quadrado, definidos assim:
  - `x = ((m-i)/(k+1))*h;`, onde m é a real coordenada do ponto na matriz, i é o início do quadrado na matriz grande, k é a taxa de descompressão e h é referente ao lado do quadrado, definido no enunciado.
  - `y = ((n-j)/(k+1))*h;`, onde n é a real coordenada do ponto na matriz, j é o início do quadrado na matriz grande, e o resto é análogo.
- Isso faz com que os índices dentro do quadrado estejam entre 0 e h, e, por consequência, o quadrado tenha lado de tamanho k.

O cálculo é feito para todos os pontos que não são os vértices do quadrado.

Em nosso método, consideramos (x_0, y_0) como $(1, 1)$ da matriz e (x_{p-1}, y_{p-1}) como (p, p) da matriz.

```
function B = bicubico (A, k, h, p)
```

Essa função representa o outro método de interpolação descrito no enunciado, o método Bicubico.

Da mesma forma que no outro método, chamamos a função `expande` que devolve uma matriz com o tamanho necessário (mais detalhes abaixo).

Para o cálculo das derivadas parciais existem 3 funções: `derivax`, `derivay` e `derivaxy` que já verificam as condições para as derivadas na borda. Uma observação a se realizar é que consideramos que nos casos em que aparecem pontos que extrapolam a grade fizemos os cálculos utilizando a diferença unilateral com o ponto da borda e seu adjacente.

Para cada quadrado Q_{ij} descrito no enunciado calculamos a matriz com os 16 coeficientes de p_{ij}. Calculamos os valores das cores de cada pixel da imagem com a fórmula do polinômio interpolador para cada quadrado Q_{ij} do enunciado.

```
function B = expande (A, k, p)
```

Essa função faz algo parecido com o oposto do descrito em **compress.m**

Recebe uma matriz quadrada A de tamanho p e coloca k linhas e colunas de zeros entre suas linhas e colunas, atribuindo tudo isso em uma matriz B.

Vamos andando ponto a ponto, e caso esse ponto tenha o par (linha, coluna) $= (i, j)$ tal que ambos os números cumpram o requisito $x\%(k+1) = 1$, onde % representa a operação de resto, o correspondente $(\frac{i-1}{k+1}+1, \frac{j-1}{k+1}+1)$ da matriz A é atribuído na matriz B.