Laboratório de Métodos Numéricos
EP2

Daniel Silva Nunes - 10297612
Renan Tiago dos Santos Silva - 9793606

30 de junho de 2020

1 Introdução

Esse exercício-programa, feito em Octave, utiliza métodos de interpolação de polinômios para o caso bivariado, a fim de descomprimir imagens e observar os resultados do processo, comparando erros e efeitos conforme atributos da imagem também variam.

2 Decisões de projeto quanto a implementação dos métodos

Para comprimir uma imagem `img.png`, com parâmetro $k = x$, executar

```
compress("img.png", x)
```

Para descomprimir a imagem comprimida por `compress`, executar

```
decompress("compressed.png", m, k, h)
```

em que `m` é o método a ser utilizado (1 para bilinear e 2 para bicúbico), e `k` e `h` são os parâmetros descritos no enunciado.

Ao invés de codificar apenas três function files, conforme o enunciado, decidimos criar mais arquivos de modo a deixar o código mais modularizado, reutilizável e legível. Os arquivos extras são os seguintes:

- **bilinear.m** - function file contendo a implementação da interpolação bilinear.
- **bicubic.m** - function file contendo a implementação da interpolação bicúbica.
- **createEmptySquares.m** - função auxiliar que recebe a imagem comprimida e os parâmetros k e $finalSize$. O parâmetro k corresponde ao lado de um quadrado cujos pixels em seu interior serão interpolados a partir dos pixels em seus vértices, conforme o enunciado. $finalSize$ corresponde às dimensões da imagem final, e é calculado por $finalSize = n+(n-1)k$. A função não preenche os espaços vazios, já que esses serão preenchidos pelas interpolações a serem executadas posteriormente.
- **dfdx.m**, **dfdy.m**, **dfdxy.m** - funções que aproximam as derivadas parciais e mistas necessárias para as interpolações.
- **calculateError.m**: Function file que recebe o nome de dois arquivos, o arquivo original e o descomprimido, e calcula e retorna o erro entre seus pixels conforme definido no enunciado do exercício.

3 Experimentos

3.1 Demonstração da aplicação dos métodos

Primeiramente, geramos uma imagem em RGB a partir da função de classe C^2

$$f(x, y) = (\sin(x), \frac{\sin(x) + \sin(y)}{2}, \sin(x)$$

produzindo o seguinte resultado:

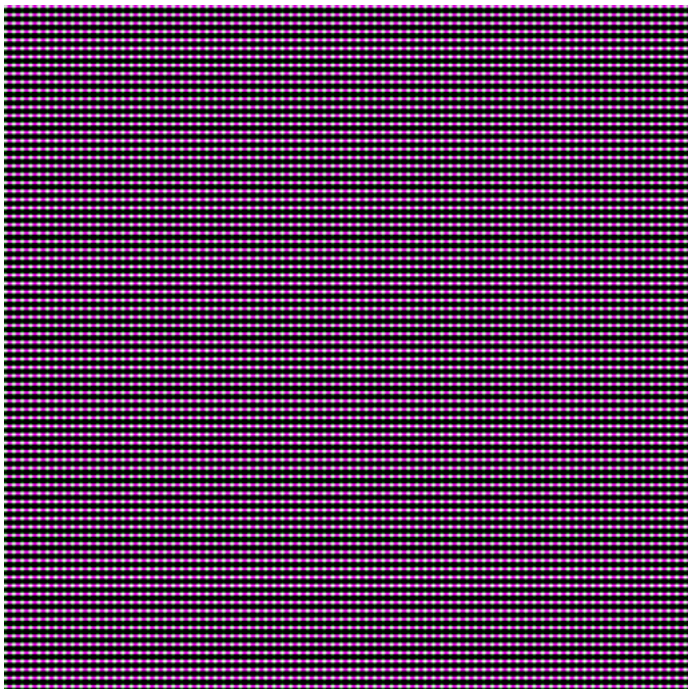

Figura 1: Imagem com dimensões 512×512 pixels, gerada por $f(x)$

O código para geração dessa figura pode ser encontrado no function file `generate_img.m`, que recebe parâmetros w e h, as dimensões da imagem em pixels, e também o parâmetro $f = \{1, 2, 3\}$, para definir qual função será utilizada na produção da imagem. O resultado é salvo no arquivo `c2.png`.

Possíveis funções para usar com `generate_img.m`:

1. $f(x, y) = (\sin(x), \frac{\sin(x)+\sin(y)}{2}, \sin(x))$

2. $g(x) = (\cos(x), \sin(x^2), 2\cos(x))$

3. $h(x) = (y^2, x^2, \sin(x))$

Em seguida, realizamos a compressão da imagem, conforme o enunciado, para $k = 4$, produzindo a seguinte imagem de dimensões 103×103 pixels:

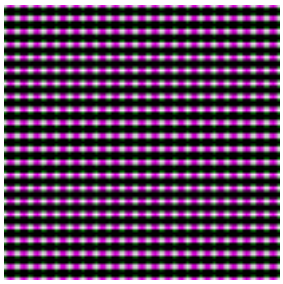

Figura 2: Imagem exibida na figura 1, comprimida com $k = 4$

Com a imagem comprimida, podemos testar ambos os métodos de interpolação bidimensional para descomprimí-la novamente e observar os resultados.

3.1.1 Interpolação Bilinear

A descompressão da imagem utilizando o método bilinear implementado recebe a imagem comprimida, o parâmetro k, que indica quantos pixels serão criados entre cada um dos pixels da imagem comprimida, e h, que indica o lado do quadrado cujos vértices serão utilizados na interpolação.

Executando o método bilinear com $k = 4$ e $h = 2$, obtivemos a seguinte imagem descomprimida:

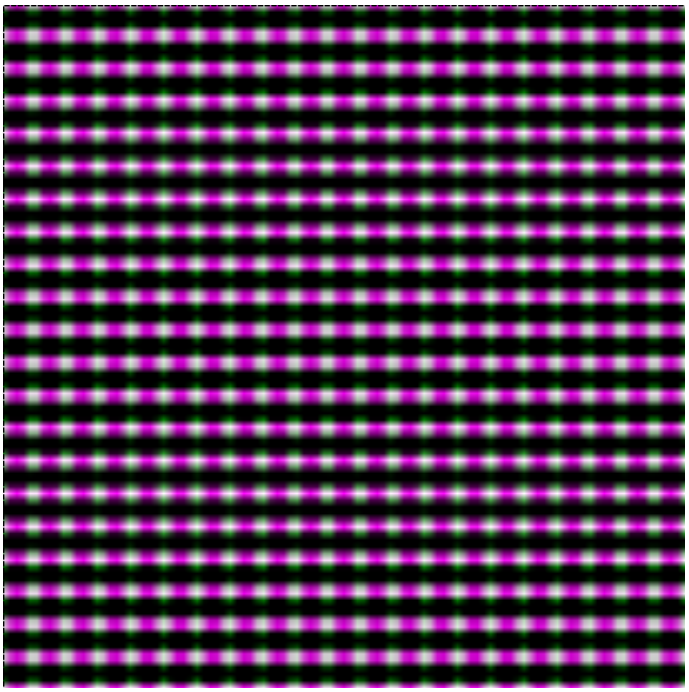

Figura 3: Imagem gerada por $f(x)$, descomprimida uma vez com $k = 4$ e $h = 2$ utilizando o método bilinear.

Observa-se que alguns ruídos foram introduzidos em relação à imagem original, sendo que, naturalmente, alguns dados foram perdidos no processo de descompressão.

Além disso, durante os experimentos, observamos que o processo de descompressão, dependendo do valor de k e do tamanho da imagem original, não gera uma imagem com as mesmas dimensões da original. Isso é esperado, pois sabemos que para $k \in \mathbb{N}$, não são todos valores de k para qual a igualdade $p = n+(n-1)k$ se mantém. Isto é, uma imagem quadrada, de lado 512,

comprimida com $k = 4$ gera uma imagem de lado 103. Ao descomprimí-la com $k = 4$ novamente, obtém-se o tamanho final $p = 103 + 102 * 4 = 511$. Ou seja, com uma coluna e uma linha a menos.

Com isso em mente, para os experimentos registrados apenas utilizamos apenas valores de k que fazem a descompressão gerar imagens com o tamanho original, de modo que o erro possa ser calculado corretamente pixel a pixel.

3.1.2 Interpolação Bicúbica

A mesma imagem `c2.png` comprimida com $k = 4$ e em seguida descomprimida com $k = 4$ e $h = 2$ no método bicúbico gerou o seguinte resultado:

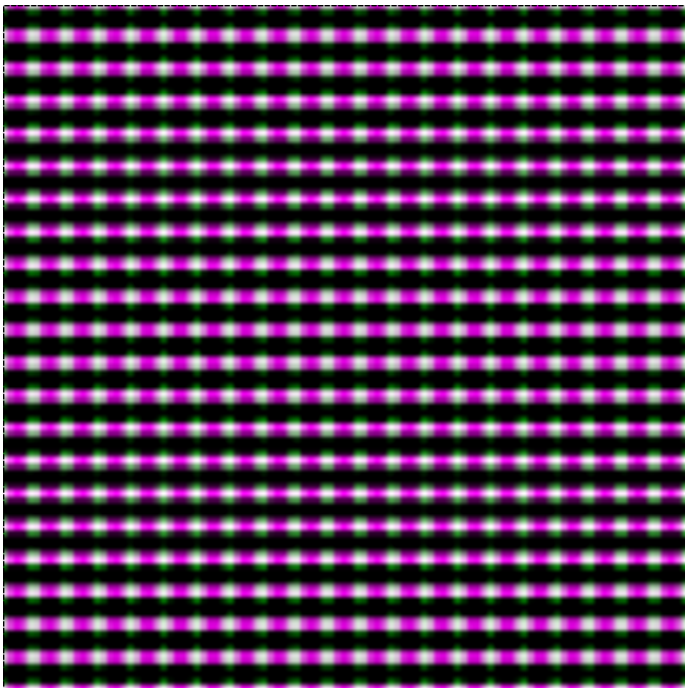

Figura 4: Imagem gerada por $f(x)$, descomprimida uma vez com $k = 4$ e $h = 2$ utilizando o método bicúbico.

Concluímos que para $f(x)$, a descompressão da imagem com os mesmos parâmetros k e h realizada uma só vez não apresenta diferenças significativas.

4 Exemplos ilustrativos dos resultados

4.1 Imagens no zoológico

Para esses experimentos, utilizaremos as seguintes imagens geradas a partir das funções $g(x)$ e $h(x)$, de classe C^2, definidas anteriormente:

Figura 5: Imagem gerada por $g(x)$.

Figura 6: Imagem gerada por $h(x)$

Além disso, também usaremos suas versões em preto e branco:

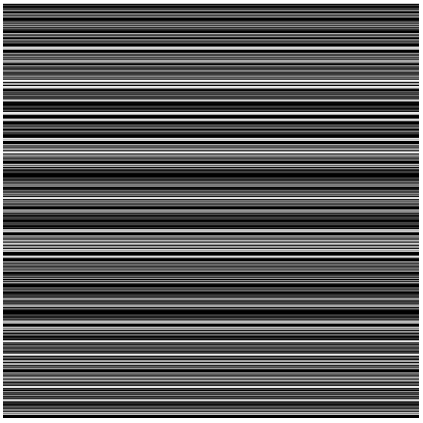

Figura 7: Imagem gerada por $g(x)$ em preto e branco.

Figura 8: Imagem gerada por $h(x)$ em preto e branco

4.1.1 Descompressão com método bilinear

As imagens foram geradas com tamanho 200×200 pixels, comprimidas com $k = 4$ e em seguida descomprimidas com $k = 4$ e $h = 2$. Os resultados obtidos pelo método bilinear são exibidos a seguir:

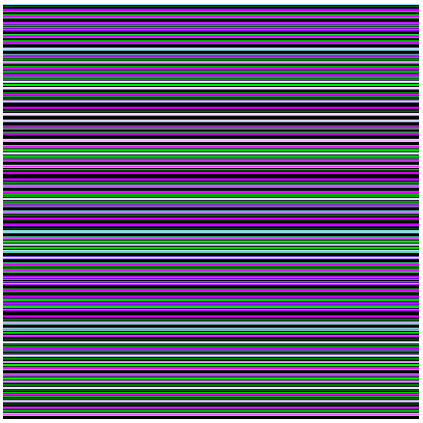

Figura 9: Imagem original gerada por $g(x)$

Figura 10: Imagem descomprimida com $k = 4$ e $h = 2$
Erro = 0.94172

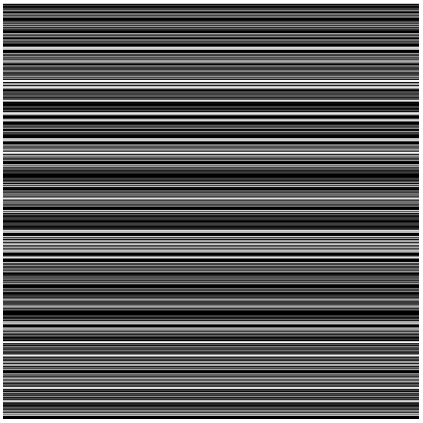

Figura 11: Imagem em preto e branco gerada por $g(x)$

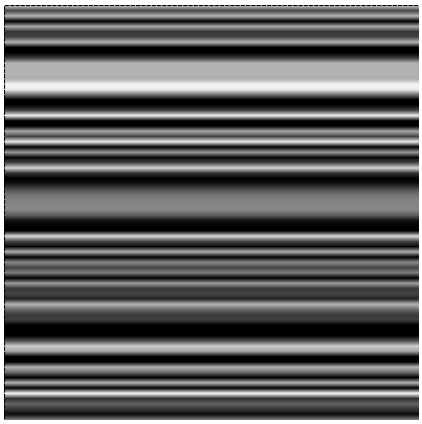

Figura 12: Imagem descomprimida com $k = 4$ e $h = 2$
Erro = 0.79257

Figura 13: Imagem gerada por $h(x)$

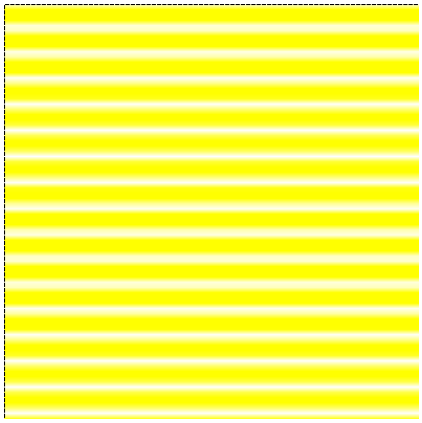

Figura 14: Imagem descomprimida com $k = 4$ e $h = 2$
Erro = 0.35273

Figura 15: Imagem em preto e branco gerada por $h(x)$

Figura 16: Imagem descomprimida com $k = 4$ e $h = 2$
Erro = 0.057840

É possível observar, sobretudo nas imagens em preto e branco que foram descomprimidas, que algumas colunas e pixels não foram processados por conta do tamanho da imagem resultante, que é calculado por $n + (n - 1)k$, não retornar para p, o tamanho original, a depender do k escolhido.

- **Funciona bem para imagens preto e branco?**

  Nos exemplos, podemos observar que a interpolação bilinear funciona bem para imagens em preto e branco, produzindo pixels com intensidades semelhantes aos das imagens coloridas, salvo, é claro, pelas cores.

- **Funciona bem para imagens coloridas?**

  Sim. Observamos que os pixels originais são bem amostrados e produzem resultados coerentes com os esperados.Em função do valor de h, faixas de cores que nas imagens originais eram menores tendem a ficar em maior destaque.

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

  Nos próximos exemplos, na seção "selva", teremos exemplos de imagens que não foram modeladas por funções de classes C^2. Veremos como os métodos se comportam a seguir para esses casos.

- **Como o valor de h muda a interpolação?**

  O valor de h descreve o lado do quadrado a ser utilizado para interpolar os pixels internos a um conjunto de quatro vértices (pixels) extraídos da imagem original. Quanto maior o h utilizado, maior a variância dos pixels dentro dos quadrados interpolados. Nas imagens geradas nessa seção, porém, não observamos grandes diferenças, talvez pelo aspecto linear que a imagem tem.

- **Como se comporta o erro?** Os valores de erro, calculados por `calculateError.m`, são indicados na exibição de cada uma das imagens. Em imagens em preto e branco, o erro tende a ser menor. Nossa hipótese é que isso é motivado pela menor variação de cores em cada pixel. Um erro menor também é observado na imagem de $h(x)$, que por sua vez também tem aparentemente menos cores.

- **Considere uma imagem de tamanho p^2 . Comprima-a com $k = 7$. Para obter a descompressão, podemos rodar decompress com $k = 7$. Experimente alternativamente usar decompress três vezes com k = 1 nas três. Compare os resultados.**

  Para essa questão, usamos a imagem gerada por $f(x)$ exibida anteriormente:

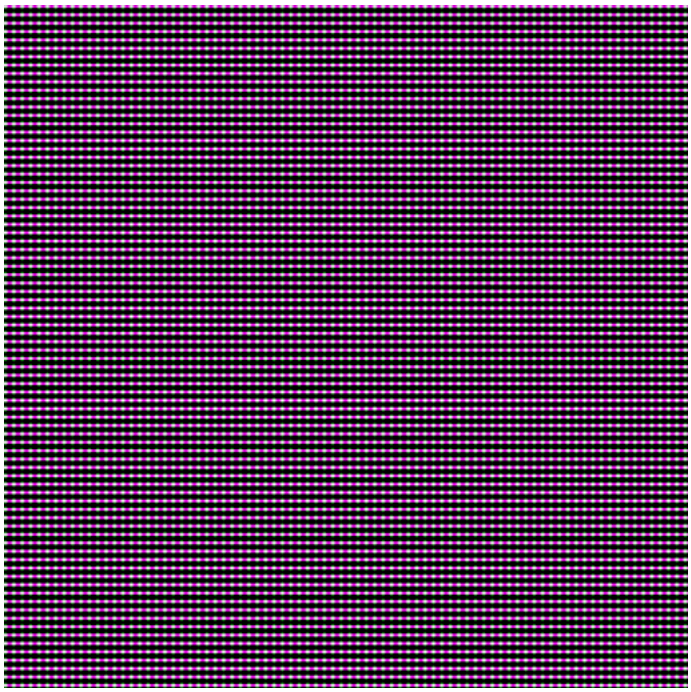

Figura 17: Imagem de tamanho 400×400 pixels gerada por $f(x)$

A imagem comprimida com $k = 7$ e depois descomprimida com $k = 7$ e $h = 2$, pelo método bilinear, é a seguinte:

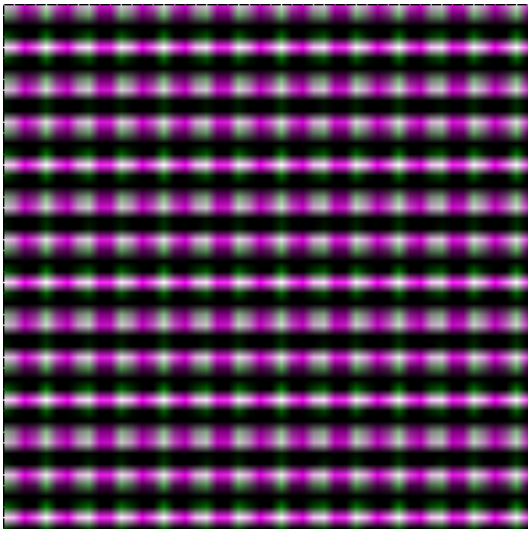

Figura 18: Imagem descomprimida com $k = 7$ e $h = 2$
Erro = 0.89017

E a imagem descomprimida três vezes com $k = 1$:

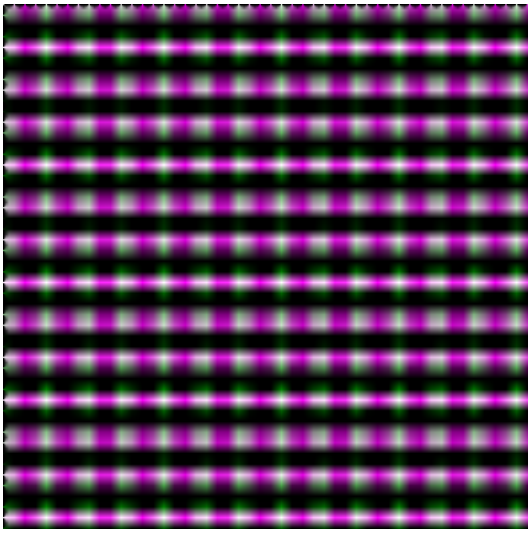

Figura 19: Imagem descomprimida três vezes com $k = 1$ e $h = 2$
Erro = 0.89007

Observa-se assim que o efeito sobre a imagem resultante é muito próximo

nos dois casos. Os erros são basicamente os mesmos e o resultado aparente também. Todavia, é mais rápido descomprimir a imagem três vezes com $k = 1$ do que executar a descompressão com $k = 7$.

4.1.2 Descompressão com método bicúbico

Similarmente, para o método bicúbico, fixando $k = 4$ e $h = 2$, obtivemos os seguintes resultados:

Figura 20: Imagem descomprimida
Erro = 0.95732

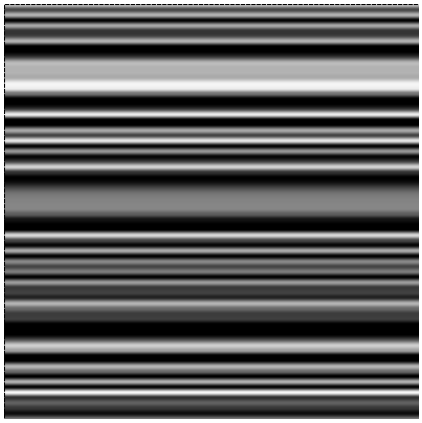

Figura 21: Imagem em preto e branco descomprimida
Erro = 0.81844

Observa-se uma transição mais suave entre as cores da imagem interpolada pelo método bicúbico quando se compara com o resultado do método bilinear.

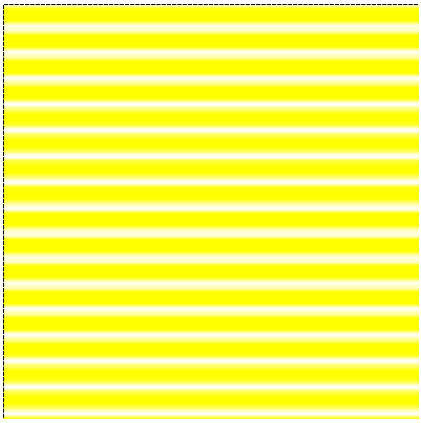

Figura 22: Imagem descomprimida
Erro = 0.35856

Figura 23: Imagem em preto e branco descomprimida
Erro = 0.058510

Observa-se que para imagens com variação menor de cores, o erro permanece menor.

- **Funciona bem para imagens preto e branco?**

  Como no método bilinear, a interpolação bicúbica a partir de imagens em preto e branco também gera bons resultados. Os erros para essas imagens são menores, dado que a variação de cores é menor.

- **Funciona bem para imagens coloridas?**

  Sim. A partir dos exemplos, é possível observar uma transição mais suave entre as cores da interpolação bicúbica.

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

  Será mostrado a seguir.

4.2 Imagens na selva

Para esses experimentos, utilizaremos as seguintes imagens:

Figura 24

Figura 25

Figura 26

Figura 27

A compressão foi aplicada em cada imagem com o parâmetro $k = 5$. As imagens têm dimensões padronizadas de 500×500 pixels. Em seguida, a descompressão foi feita com $k = 6$ e $h = 3$, para cada um dos exemplos. Os resultados são exibidos a seguir, em conjunto com o método utilizado e a taxa de erro obtida.

4.2.1 Figura 24

Figura 28: Método bilinear
$k = 6$ e $h = 3$
Erro = 0.054998

Figura 29: Método bicúbico
$k = 6$ e $h = 3$
Erro = 0.055020

4.2.2 Figura 25

Figura 30: Método bilinear
$k = 6$ e $h = 3$
Erro = 0.091013

Figura 31: Método bicúbico
$k = 6$ e $h = 3$
Erro = 0.092239

Observamos com os dois primeiros exemplos que tanto para imagens coloridas como para imagens em preto e branco, que não são geradas por funções

de classe C^2, que ambos os métodos apresentam boa performance de descompressão, mantendo erros baixos em relação às imagens originais.

4.2.3 Figura 26

Figura 32: Método bilinear
$k = 6$ e $h = 3$
Erro = 0.031576

Figura 33: Método bicúbico
$k = 6$ e $h = 3$
Erro = 0.031521

4.2.4 Figura 27

Figura 34: Método bilinear
$k = 6$ e $h = 3$
Erro = 0.055249

Figura 35: Método bicúbico
$k = 6$ e $h = 3$
Erro = 0.056027

A partir dos experimentos com imagens reais, constatamos que os métodos de interpolação apresentam baixos erros na descompressão, sendo que alguns deles foram menores do que os obtidos com imagens geradas por funções $\in C^2$, como é o caso da figura 22 na página 14.