EP2 - Comprimir e Conquistar

MAC0210 - Laboratrio de Mtodos Numricos

Setembro 2016

Este enunciado foi elaborado e escrito por Ernesto G. Birgin (professor), Gustavo Estrela (monitor) e pelos alunos Nathan Benedetto, Rodrigo Enju, Victor da Matta, Victor Sprengel e Victor Molero.

1 Introdução

Este exercício-programa tem como objetivo generalizar dois métodos de interpolação por polinômios para o caso bivariado. O experimento consiste em começar com uma imagem grande, fazer uma amostra pequena de seus pixels e depois reconstruí-la.

Você deve comparar os resultados de interpolação de duas maneiras. A primeira parte consistirá em observar os resultados desse experimento quando a imagem interpolada é gerada por uma função (o zoológico), e a outra em testá-lo com imagens reais (a selva).

2 Parte 0 - O Laboratório

Primeiro, vamos considerar que todo nosso trabalho de interpolação pode ser feito independentemente para cada uma das 3 cores da paleta RGB. Uma imagem de p^2 pixels fornece, então, 3 matrizes de pixels, as quais estamos interessados em estudar e interpolar. Para interpolar essa imagem, vamos imaginar que os pixels conhecidos foram amostrados de uma imagem original, nas seguintes condições:

$$\begin{array}{rl} \text{Dado} & h > 0\ ,\ h \in \mathbb{R} \\ \text{e uma função} & f : \mathbb{R}^2 \to \mathbb{R} \\ \text{amostrada nos pontos } (x_i, y_j) \text{ com} & x_i = ih,\ i = 0, ..., (p-1) \\ & y_j = jh,\ j = 0, ..., (p-1) \end{array}$$

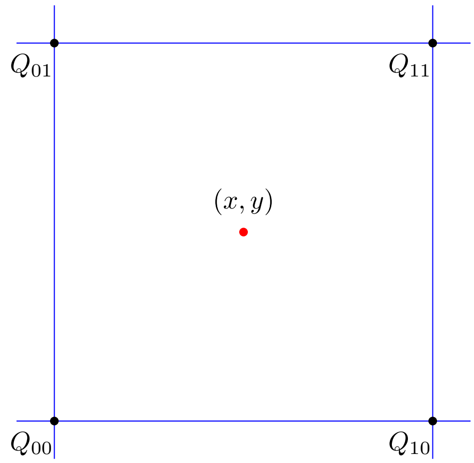
Então, a matriz de pixels F de tamanho $p \times p$ é tal que

$$(F)_{ij} = f(x_i, y_j) = f(ih, jh)\ ,\ 0 \leq i < p \text{ e } 0 \leq j < p.$$

Nossa função interpolante deve ser capaz de, dado $(x, y) \in [0, (p-1)h] \times [0, (p-1)h]$, aproximar $f(x, y)$. Iremos supor que (x, y) não é um dos pontos amostrados, isto é,

$$(x, y) \neq (x_i, y_j)\ \forall\ 0 \leq i < p, 0 \leq j < p.$$

Para aproximar $f(x, y)$, considere os quatro pontos amostrados que definem um quadrado de lado h que contém o ponto (x, y).



Definiremos esse quadrado da seguinte forma:

$$
\begin{aligned}
Q_{00} &= (x_i, y_j) \text{ para algum } 0 \leq i, j \leq p-1 \\
Q_{10} &= (x_{i+1}, y_j) \\
Q_{11} &= (x_{i+1}, y_{j+1}) \\
Q_{01} &= (x_i, y_{j+1})
\end{aligned}
$$

Note que, da maneira que definimos, existe mais de um quadrado que contém o ponto (x, y) se ele estiver na fronteira de um quadrado. Você tem a liberdade de criar uma regra para definir qual quadrado será usado para interpolar o ponto nessas condições.

Nos dois métodos de interpolação que implementaremos nesse exercício programa, o valor interpolado em (x, y) dependerá apenas de informações dos vértices do quadrado de lado h como definimos.

2.1 Interpolação Bilinear Por Partes

Podermos aproximar a função $f(x, y)$ deste modo:

$$f(x, y) \approx a_0 + a_1 x + a_2 y + a_3 xy$$

e expressar essa função interpolante por meio de um sistema de equações:

$$
\begin{bmatrix}
1 & x_0 & y_0 & x_0 y_0 \\
1 & x_0 & y_1 & x_0 y_1 \\
1 & x_1 & y_0 & x_1 y_0 \\
1 & x_1 & y_1 & x_1 y_1
\end{bmatrix}
\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}
=
\begin{bmatrix} f(Q_{00}) \\ f(Q_{01}) \\ f(Q_{10}) \\ f(Q_{11}) \end{bmatrix}
$$

que precisa ter solução única pelas definições de função interpolante.

Leia mais em:
https://en.wikipedia.org/wiki/Bilinear_interpolation

2.2 Interpolação Bicubica

Na interpolação bilinear cada ponto interpolado depende apenas dos valores dos 4 pontos mais próximos, como definimos anteriormente. Esse tipo de técnica pode causar uma aparência quadrada na imagem interpolada, como vemos abaixo.

Para diminuir esse efeito usamos a interpolação bicubica, que exige que a função interpoladora $v(x, y)$ seja contínua e também que as primeiras derivadas em x e y e a derivada primeira mista sejam contínuas. Note que em nosso exemplo não faz sentido exigir que a imagem de entrada tenha informação das derivadas parciais (e será que faz sentido perguntar o valor da derivada parcial em um ponto da imagem?) portanto teremos que aproximar essas derivadas parciais.

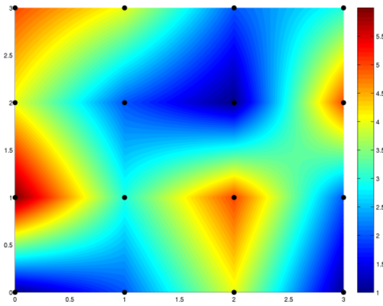


Figure 1: Figura gerada por uma interpolação bilinear. Fica aparente na imagem como a falta de suavidade na interpolação "mostra" quadrados delimitados pelas bordas de cada interpolação. Fonte: http://www.codecogs.com/library/maths/approximation/interpolation /multivariate.php

Iremos calcular as derivadas parciais de forma aproximada, por diferenças finitas. Dado um ponto (x_a, y_b), é interessante fazer nossa aproximação ao redor do ponto, de forma centralizada.

De forma até um pouco surpreendente, aproximar a derivada de uma função $g : \mathbb{R} \to \mathbb{R}$ por

$$g'(x) \approx \frac{g(x+h) - g(x-h)}{2h}$$

Garanto que o erro tenha ordem $O(h^3)$, enquanto a forma mais natural,

$$g'(x) \approx \frac{g(x+h) - g(x)}{h}$$

Apenas fornece uma garantia de erro da ordem $O(h)$. A primeira formulação se chama *diferença centralizada*, equanto a segunda se chama *diferença lateral* [1].

As derivadas parciais são calculadas variando apenas um dos termos, e, portanto, podem ser calculadas da mesma forma que se calcula no caso de funções de $\mathbb{R}$ pra $\mathbb{R}$. Assim, as fórmulas para as derivadas primeiras são:

$$\frac{\partial f}{\partial x}(x_a, y_b) \approx \frac{f(x_{a+1}, y_b) - f(x_{a-1}, y_b)}{2h}$$
$$\frac{\partial f}{\partial y}(x_a, y_b) \approx \frac{f(x_a, y_{b+1}) - f(x_a, y_{b-1})}{2h}$$

Resta então calcular as derivadas mistas. Isto será feito utilizando as já calculadas derivadas parciais. Ou seja, o método é equivalente a calcular a derivada parcial com relação à x na função dada pela derivada parcial com relação à y. Formalmente,

$$\frac{\partial f}{\partial x \partial y}(x_a, y_b) \approx \frac{\frac{\partial f}{\partial y}(x_{a+1}, y_b) - \frac{\partial f}{\partial y}(x_{a-1}, y_b)}{2h}$$

É importante observar que o método da diferença centralizada não pode ser aplicado nos pontos das bordas. Para obter estes valores, será necessário calcular utilizando a diferença unilateral correspondente. Os casos de borda são

$$\frac{\partial f}{\partial x}(x_0, y_b) \approx \frac{f(x_1, y_b) - f(x_0, y_b)}{h}$$
$$\frac{\partial f}{\partial x}(x_{p-1}, y_b) \approx \frac{f(x_{p-1}, y_b) - f(x_{p-2}, y_b)}{h}$$

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

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

Você tem agora tudo que precisa para calcular as derivadas parciais em Octave.

Dado os quatro cantos do qual o ponto (x, y) depende (os vértices do quadrado de lado h que definimos), a função interpoladora $v(x, y)$, neste caso, será descrita dessa forma:

$$v(x,y) = [1\ x\ x^2\ x^3] \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}$$

onde

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = B \begin{bmatrix} f(x_a, x_b) & f(x_a, x_{b+1}) & \frac{\partial f(x_a,x_b)}{\partial y} & \frac{\partial f(x_a,x_{b+1})}{\partial y} \\ f(x_{a+1}, x_b) & f(x_{a+1}, x_{b+1}) & \frac{\partial f(x_{a+1},x_b)}{\partial y} & \frac{\partial f(1,1)}{\partial y} \\ \frac{\partial f(x_a,x_b)}{\partial x} & \frac{\partial f(x_a,x_{b+1})}{\partial x} & \frac{\partial^2 f(x_a,x_b)}{\partial x \partial y} & \frac{\partial^2 f(x_a,x_{b+1})}{\partial x \partial y} \\ \frac{\partial f(x_{a+1},x_b)}{\partial x} & \frac{\partial f(1,1)}{\partial x} & \frac{\partial^2 f(x_{a+1},x_b)}{\partial x \partial y} & \frac{\partial^2 f(1,1)}{\partial x \partial y} \end{bmatrix} B^T$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix}$$

Leia mais em:
https://en.wikipedia.org/wiki/Bicubic_interpolation

3 O Programa

Você deve entregar três **function file**s do **Octave**. Segue a descrição de cada um.

Dica: Para ler e escrever imagens leia a documentação do **Octave** para **imread** e **imwrite**. Estas duas funções bastam para o pedido.

3.1 compress(originalImg, k)

Esta função deve comprimir uma imagem. Suponha que ela é quadrada e tem p^2 pixels e que $\exists n \in \mathbb{N}$ tal que $p = n + (n-1)k$. Comprima a imagem original para o tamanho $n \times n$ ao manter as linhas e colunas de índice i tais que i é da forma $i \equiv 1 \mod (k+1)$; remova o resto dos pixels.

3.1.1 Parâmetros

- **originalImg** - nome da imagem original que você irá comprimir
- **k** - taxa de compressão

3.1.2 Saída

Você deve gerar uma imagem chamada **compressed.png** como pedido acima.

3.2 decompress(compressedImg, method, k)

Esta função deve realizar o papel de decompressão. Você deve expandir uma imagem comprimida quadrada com tamanho de lado n de modo que o lado final tenha tamanho $n + (n-1)k$. Faça isso adcionando k linhas e k colunas entre cada uma das n linhas e colunas.

Depois, para cada ponto que você precisar preencher, interpole três funções (uma para cada cor RGB) usando o método especificado em **method** e descrito nesta seção.

3.2.1 Parâmetros

- **compressedImg** - nome do arquivo da imagem que comprimida
- **method** - método de interpolação a ser usado
  - **1** - Bilinear
  - **2** - Bicubico
- **k** - Taxa de compressão, como especificada na compressão e acima

3.2.2 Saída

Gere uma imagem chamada **decompressed.png**. Esta é a imagem gerada pela sua matriz de tamanho $n + (n-1)k$ após a interpolação das entradas criadas.

3.3 calculateError(originalImg, decompressedImg)

Este programa deve calcular o erro entre a imagem original e a imagem descomprimida do seguinte modo:

Considere $origR$ a matriz correspondente a cor RED da imagem original, e análogo para B e G. Considere também $decR$ a matriz correspondente a cor RED da sua imagem descomprimida, e análogo para G e B. Então definimos o erro err como

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

onde

$$errR = \frac{\|origR - decR\|_2}{\|origR\|_2}$$

e análogo para G e B.

3.3.1 Parâmetros

- **originalImg** - nome do arquivo que contém a imagem original
- **decompressedImg** - nome do arquivo que contém a imagem depois de comprimida e descomprimida

3.3.2 Saída

Imprima apenas um número na saída padrão: o erro calculado.

4 Parte 1 - O Zoológico

Nesta parte você deve usar uma função $f : \mathbb{R}^2 \to \mathbb{R}^3$ de classe C^2 para gerar uma imagem grande em RGB, no formato especificado na parte "Programa", para testar nosso método de compressão, isto é, comprimir, descomprimir e calcular o erro.

Considere que a matriz F que sua imagem dá vem de um h suficientemente pequeno (pelo menos ordem de 10^{-4}) para que a aproximação da derivada seja suficientemente boa.

Teste com exemplos próprios, mas certifique-se de testar pelo menos com a função:

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

(Ajustar esse teste para novo h)

Faça experimentos e responda questões como:

- Funciona bem para imagens preto e branco?
- Funciona bem para imagens coloridas?
- Funciona bem para todas as funções de classe C^2?
- E para funções que não são de classe C^2?
- Como se comporta o erro?

Responda também a esta questão:

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. Escreva no relatório suas conclusões.

5 Parte 2 - A Selva

Agora, na selva, sua tarefa é realizar algo muito parecido com a parte 1, mas com uma imagem real (foto ou desenho). Note que agora a existência de uma f de classe C^2 é uma suposição muito provavelmente falsa.

Responda no relatório todas as questões ainda pertinentes da Parte 1, e outras que você julgar boas.

6 Para entregar

Entregue um arquivo .tar com nome **ep2.tar**. Ao descompactá-lo devemos encontrar uma pasta **ep2** com os seguintes conteúdos:

- **decompress.m**, o programa em Octave como function file
- **compress.m**, o programa em Octave como function file
- **calculateError.m**, o programa em Octave como function file
- **relatorio.pdf**, um relatório completo do EP, em pdf, incluindo:
  - Decisões de projeto quanto à implementação dos métodos
  - Observações pedidas quanto aos experimentos
  - Exemplos ilustrativos dos resultados

References

[1] Dennis, J.E. and Schnabel, R.B. "Numerical Methods for Unconstrained Optimization and Nonlinear Equations." Smth, (1983).