CSE 4345 – Homework #8

Assigned: Tuesday, April 16, 2019
Due: Tuesday, April 23, 2019 at the end of class

Note the following about the homework:

1. You must show your work to receive credit.
2. If your submission has more than one page, staple the pages. If I have to staple it, the cost is 10 points.

Assignment:

1. (hand solution) Given the data {(3, 4), (-2, 24), (0, 4)}, determine the interpolating polynomial of degree two using
   (a) the monomial basis
   (b) the Newton basis
2. (hand solution) Find the natural cubic spline through the points (2, 5), (3, 3), (5, 4), and (6, 4). Give exact solutions, that is, keep the fractions produced. Be clear what the final solution is in terms of the polynomials.

   There are many ways to derive and write the polynomials representing the spline. I need an easy way to verify the accuracy of your answer, so you should follow the approach in [Sau12, pp. 170-171] and the polynomials should be in the form

   $$p_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

3. (Python solution) Sometimes we wish to resize an image. When the resizing produces a larger image, we have to make a decision as to how we produce the values that didn't exist in the original image. In our case, we will use bilinear interpolation.
   (a) On the course website is a skeleton file, `bilinear.py`. You will write the function `resizeImage()`. It's OK to decompose your logic into multiple functions, but I should be able to call your `resizeImage()` function with my own image name and it should work correctly.
   (b) On the course website is a file, `hw08-images.zip`, that contains the image `book_fausett_small.jpg`.
   (c) `resizeImage()` should resize the image whose name it is given using bilinear interpolation. The new image will be approximately twice as wide and approximately twice as tall (ask yourself why I said approximately). To do this, spread out the original pixels in the image and then use linear interpolation to calculate the values between the pixels. For example,

A	B	C
D	E	F
G	H	I
J	K	L

becomes

A	a	B	b	C
c	d	e	f	g
D	h	E	i	F
j	k	l	m	n
G	o	H	p	I
q	r	s	t	u
J	v	K	w	L

To find the new values in the new image, we would use the following equations:

- $a = (A + B)/2$
- $b = (B + C)/2$
- $c = (A + D)/2$
- $d = (A + B + D + E)/4$
- $e = (B + E)/2$
- $f = (B + C + E + F)/4$
- $g = (C + F)/2$

The same approach is used for the remaining values. Since the image is in color, apply this overall approach to each color component's layer.

(d) While you should not hard-code your logic to the image I have provided, you can assume the image will be a color image and therefore will have three layers that you will need to change.

For the sample image I have provided, the original image has dimensions $130 \times 73 \times 3$ and therefore the new image will have dimensions $259 \times 145 \times 3$.

(e) Write your own function to produce the new pixel values of the resized image instead of relying on any image processing libraries for doing this. Of course, you will have to use something (e.g., matplotlib) for reading and displaying the file.

(f) Your program should display the original image and the new image as subplots of the same figure. Put the original image on the left and the new image on the right.

4. (Python solution) Sometimes we may wish to create an animated GIF that morphs from one image into another. A simple way to create the intermediate images is to use interpolation.

(a) On the course website is a skeleton file, `morph.py`. You will write the function `intermediate()`. which has the signature

```
intermediate(name1, name2, N, display)
```

where

- `name1`: string of the name of the starting image
- `name2`: string of the name of the stopping image
- `N`: integer representing how many steps to go from the starting image to the stopping image; $N = 1$ means that you go straight from the starting image to the stopping image an no intermediate images are created

- `display`: a Boolean flag indicating if the intermediate images should be displayed or saved to files. If `display` is `True`, each intermediate image should be displayed in its own figure window. If `display` is `False`, each intermediate image should be saved to a file with the names being `mid01.jpg`, `mid02.jpg`, ... There will be $N-1$ images displayed or files created. They should be in order, that is, the first intermediate image created should be the one that follows the starting image, etc.

It's OK to decompose your logic into multiple functions, but I should be able to call your `intermediate()` function with my own arguments and it should work correctly.

(b) On the course website is a file, `hw08-images.zip`, that contains the images `darinYoung.jpg` and `darinOld.jpg`.

(c) While you should not hard-code your logic to the images I have provided, you can assume

- the images will be a color image and therefore will have three layers that you will need to change.
- the first and last images have the same dimensions.
- N will be at least one (this will result in NO images being produced)

(d) The intermediate images will be created using linear interpolation between the corresponding pixels in the beginning and ending images. We want to go from the first image to the last image in N steps, creating the $N-1$ intermediate images.

Assume that we have two 2×2 images with a single layer in each:

10	15
30	20

becomes

16	15
32	14

If we want to go from the left image to the right image in $N = 3$ steps, then the step size for the upper left corner is (16 - 10)/3 = 6/3 = 2. For the upper right corner the step size is (15 - 15)/3 = 0/3 = 0. For the lower left corner the step size is (32 - 30)/3 = 2/3. For the lower right corner, the step size is (14 - 20)/3 = -6/3 = -2.

(e) Note that these images will come in as type `uint8` and we don't want to round off the step sizes prematurely. For example, the lower left corner pixel value begins at 30 and changes to 32 in steps of 2/3 when $N = 3$. Therefore, we have the following pixel values:

- starting image: 30
- intermediate image: $30 + 1 \times 2/3 = 30.667$. After converting to `uint8`, this becomes 30.
- intermediate image: $30 + 2 \times 2/3 = 31.333$. After converting to `uint8`, this becomes 31.
- stopping image: $30 + 3 \times 2/3 = 32$. This image is never actually produced, but shows how the steps get us to it.

(f) Write your own function to produce the intermediate images instead of relying on any libraries that do the interpolation for you. Of course, you will have to use something (e.g., matplotlib) for reading the files.

General requirements about the Python problems:

a) As a comment in your source code, include your name.

b) The Python program should do the work. Don't perform the calculations and then hard-code the values in the code or look at the data and hard-code to this data unless instructed to do so.

c) The program should not prompt the user for values, read from files unless instructed to do so, or print things not specified to be printed in the requirements.

To submit the Python portion, do the following:

a) Create a directory using your net ID in lowercase characters plus the specific homework. For example, if your net ID is `abc1234` and the homework is `hw04`, then the directory should be named `abc1234-hw04` (zero-pad the number if necessary to make it two digits).

b) Place your .py files in this directory.

c) Do not submit the data files unless instructed to do so.

d) Zip the directory, not just the files within the directory. You must use the zip format and the name of the file (using the example above) will be `abc1234-hw04.zip`.

e) Upload the zip'd file to Blackboard.

References

[Sau12] Tim Sauer. *Numerical Analysis*. Pearson Education, Upper Saddle Riber, NJ, 2^{nd} edition, 2012.