Práctica 4: Normalización e Interpolación

Néstor Iván Martínez Ostoa
Visualización de la Información - 0605
I.I.M.A.S. - U.N.A.M.

14 de marzo del 2021

Primera parte

Visualizar los siguientes campos escalares usando isosuperficies. El valor en cada punto estará dado por:

$$v_1 = \frac{\sin\left(x^2+y^2+z^2\right)^{1/2}}{\left(x^2+y^2+z^2\right)^{1/2}}$$
$$v_2 = xe^{-x^2-y^2-z^2}$$

Campo escalar v_1

Para el campo escalar v_1 el rango de valores de la malla fue: $X, Y, Z \in [-5{,}5, 5{,}5]$ cada componente con 40 puntos. Para la función que grafica la isosuperficie, se emplearon los siguientes parámetros:

- `isomin` $= -0{,}2$
- `isomax` $= 0$
- `surface_count` $= 2$

```
min_ = -5.5
max_ = 5.5
steps = 40j
X, Y, Z = np.mgrid[min_:max_:steps, min_:max_:steps, min_:max_:steps]
values = np.sin(np.sqrt(X**2 + Y**2 + Z**2)) / np.sqrt(X**2 + Y**2 + Z**2)
# Funcion para graficar la isosuperficie
plot_isosurface(X,Y,Z,values,isomin=-0.2, isomax=0,surface_count=2, opacity=0.9)
```

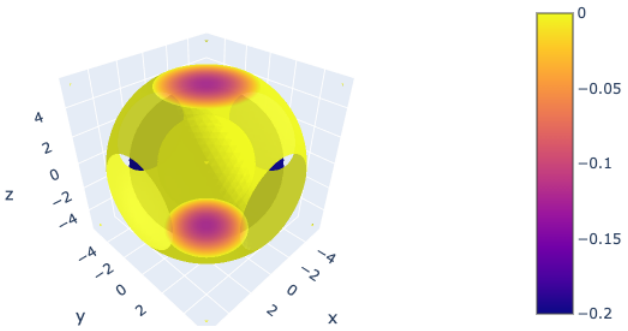


Figura 1: Visualización del campo escalar v_1

Campo escalar v_2

Para el campo escalar v_1 el rango de valores de la malla fue: $X, Y, Z \in [-1{,}5, 1{,}5]$ cada componente con 30 puntos. Para la función que grafica la isosuperficie, se emplearon los siguientes parámetros:

- `isomin` $= 0{,}2$
- `isomax` $= 0{,}2$
- `surface_count` $= 2$

```
min_ = -1.5
max_ = 1.5
steps = 30j
X, Y, Z = np.mgrid[min_:max_:steps, min_:max_:steps, min_:max_:steps]
values = X*np.power(np.e, -X**2-Y**2-Z**2)
# Funcion para graficar la isosuperficie
plot_isosurface(X,Y,Z,values,isomin=-0.2, isomax=0.2 ,surface_count=2, colorbar_ticks=10)
```

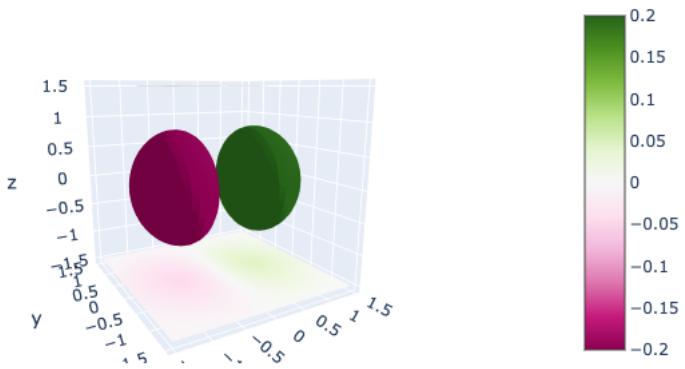


Figura 2: Visualización del campo escalar v_2

Segunda Parte

Normalización

La imagen *raw* (con dimensiones 485×485) tiene detalles que no se observan fácilmente. Cambie el rango en el que se despliegan algunos valores de la imagen (aquellos que hacen parte de los detalles ocultos).

Imagen Original

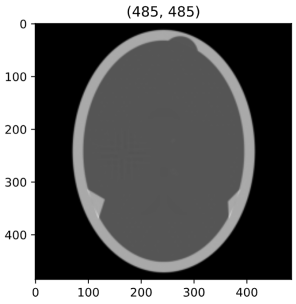


Figura 3: Imagen *raw* sin procesamiento

Procesamiento de la imagen

1. Obtenemos el histograma de la imagen original para obtener el rango de valores a modificar Del

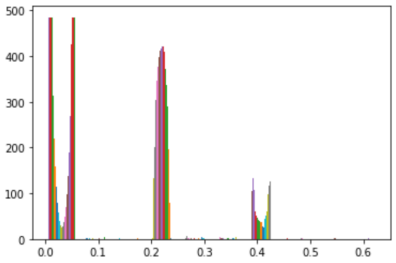


Figura 4: Histograma de la imagen *raw*

histograma de la figura 4 podemos observar que tenemos tres distribuciones de color:

- Valores cercanos a 0: representa el fondo negro de la imagen
- Valores cercanos a 0,2: son los valores que queremos visualizar que actualmente se encuentran ocultos
- Valores cercanos a 0,4: representan el anillo en gris claro que engloba la imagen

2. Obtenemos todos los valores cerca de la vecindad del 0,2 y los almacenamos en una lista

```
A = []
for row in range(raw_img.shape[0]):
    for col in range(raw_img.shape[1]):
        pixel = raw_img[row][col]
        if pixel > 0.2 and pixel < 0.25:
            A.append(pixel)
```

3. De la lista del punto anterior, obtenemos el mínimo y máximo

```
min_A = min(A)
max_A = max(A)
print(f'Minimo: {min_A}, Maximo: {max_A}')
```

Mínimo: 0,2008127
Máximo: 0,2497125

4. Realizamos la normalización con los valores en la imagen en la vecindad del 0,2:

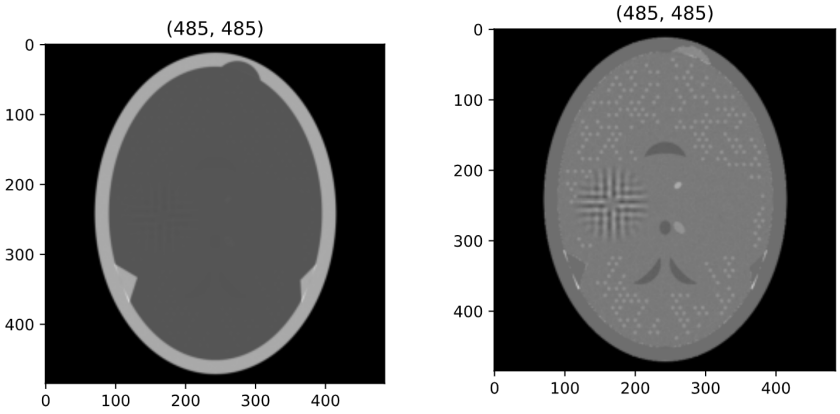
```
raw_img_cpy = np.copy(raw_img) # Copia de la imagen original
# Iteracion sobre los renglones y columnas de la matriz
for row in range(raw_img_cpy.shape[0]):
    for col in range(raw_img_cpy.shape[1]):
        pixel = raw_img_cpy[row][col]
        # Verificamos que el pixel este en la vecindad de 0.2
        if pixel > 0.2 and pixel < 0.225:
            # Normalizacion del valor del pixel
            norm = (pixel-min_A) / (max_A - min_A)
            # Igualamos a 0 cuando el pixel equivale al minimo
            if pixel == min_A:
                raw_img_cpy[row][col] = 0
                continue
            # De lo contrario, asignamos su valor normalizado + 0.1
            raw_img_cpy[row][col] = norm + 0.1
        # Incrementa el contraste del anillo en gris claro
        elif pixel > 0.395 and pixel < 0.43:
            raw_img_cpy[row][col] = 0.27
```

El proceso para normalizar fue:

$$\text{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

y una vez obtenido el pixel normalizado (norm) si el valor del pixel fue igual al mínimo de la lista A, el pixel lo igualamos a 0. De lo contrario, el pixel lo igualamos al valor normalizado y sumamos un valor de 0,1.

5. Desplegamos la imagen procesada



(a) Imagen *raw* sin procesamiento

(b) Imagen *raw* procesada seleccionando valores en la vecindad del 0,2

Figura 5: Comparación de imagen *raw* sin procesamiento y con procesamiento

Interpolación

En el anexo de esta práctica (ver sección 1.1) se encuentras las funciones detalladas por módulo y las empleadas en los resultados mostrados a continuación. Otro aspecto importante a mencionar es que las imágenes escaladas mantienen el mismo aspecto que la imagen original, sin embargo, hay que ver la escala mostrada para darse cuenta que en efecto, se está haciendo el escalamiento de manera efectiva. Esto se debe a que se ocupó el módulo `matplotlib` [1] para desplegar las imágenes.

Resultados

- **Imagen original**

```
rimage = np.array(Image.open('landscape.png').convert('L'), dtype=int)
display_raw(rimage)
```

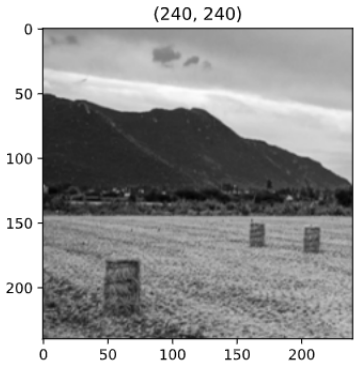


Figura 6: Imagen original

- **Imagen escalada 2x**

```
image_2x = resize_image(rimage, scaling_factor=2)
display_raw(image_2x)
```

[1] Concretamente, la función

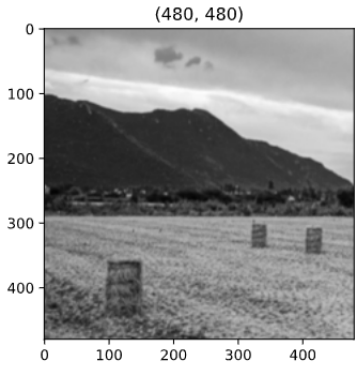


Figura 7: Imagen original escalada al doble

- **Imagen escalada 5x**

```
image_5x = resize_image(rimage, scaling_factor=5)
display_raw(image_5x)
```

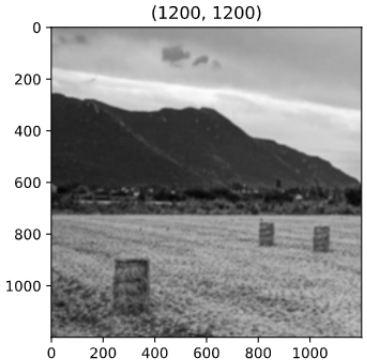


Figura 8: Imagen original escalada cinco veces

- **Zoom en imagen original**

```
zoomed_img = zoom_image(rimage, x=150, size_x=90, y=80, size_y=130)
display_raw(zoomed_img)
```

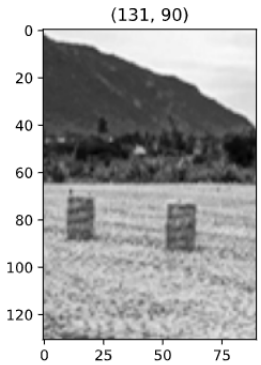


Figura 9: Zoom sobre la imagen original

- **Zoom en imagen 5x**

```
zoomed_img = zoom_image(image_5x, x=700, size_x=500, y=400, size_y=600)
display_raw(zoomed_img)
```

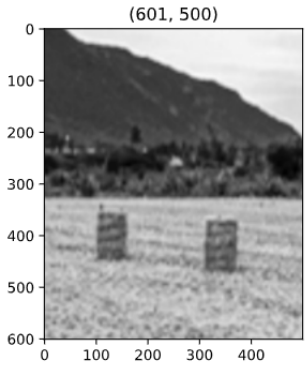


Figura 10: Zoom sobre la imagen escalada 5 veces

1. Anexo

1.1. Funciones Parte 2

1.1.1. Interpolación lineal y bilineal

```
def interpolate_image(image, num_to_ignore, axis):
    """
        Function that makes a linear interpolation in a given image through a given axis

        Parameters:
        -----------
        image: ndarray
            Matrix containing RGB values per each combination of row and column

        num_to_ignore: int
            Integer representing the empty value (non RGB value) inside the matrix

        axis: int
            axis = 0 <- rows
            axis = 1 <- columns

        Return:
        -------
        image: ndarray
            Iterpolated image
    """
    # either first row or first colum in image
    element = image[:,0] if axis == 0 else image[0]

    # indices of RGB values in element
    reference_points_indices = np.where(element != num_to_ignore)[0]
    num_rgb_values_in_element = reference_points_indices.shape[0]

    for i in range(1, num_rgb_values_in_element):
        # indices of point of reference 1 (x1) and point of reference two (x2)
        first_ref_idx = reference_points_indices[i-1]
        second_ref_idx = reference_points_indices[i]

        # getting whole columns and rows in image based on points of reference
        y1 = image[:,first_ref_idx] if axis == 1 else image[first_ref_idx]
        x1 = np.repeat(first_ref_idx, image.shape[axis])
        y2 = image[:,second_ref_idx] if axis == 1 else image[second_ref_idx]
        x2 = np.repeat(second_ref_idx, image.shape[axis])

        # Iterate in all the missing values between two points of references
        # and perform linear interpolation
        for x in range(first_ref_idx+1, second_ref_idx):
            y = linear_interpolation(x1,y1,x2,y2,x)
            if axis == 0: image[x] = y
            else: image[:,x] = y
    return image

def linear_interpolation(x1, y1, x2, y2, x):
    """
        Function that performs linear interpolation

        Parameters:
        -----------
        <x1, y1, x2, y2, x>: ndarray or int
            Given a ndarray as parameter, this function will perform vector operations

        Return:
        -------
        y: int
            Interpolated value
    """
    return abs(y1 + np.round((x-x1)*(y2-y1)/(x2-x1),0))
```

1.1.2. Escalamiento a un tamaño dado

```
def resize_image(image, scaling_factor):
    """
        Function that handles bilinear interpolation image rescaling
```

```
4
5           Parameters:
6           -----------
7           image: ndarray
8               Image matrix representing a given image where each value holds the pixel color
9
10          scaling_factor: int
11              Integer representing the scaling factor of the image to be rescaled
12
13          Returns:
14          --------
15          scaled_image: ndarray
16              Image matrix representing the new matrix with the scaling factor
17      """
18      # num_to_ignore is a value used to fill initial empty positions when
19      # rescaling the original image to the scaled image final size
20      num_to_ignore=-1
21
22      # new_empty_image is a matrix that will contain scaled image final dimensions
23      new_empty_image = build_empty_image(image, scaling_factor, num_to_ignore)
24
25      # Linear and bilinear interpolation
26      scaled_columns_image = interpolate_image(new_empty_image, num_to_ignore, axis=0)
27      scaled_image = interpolate_image(scaled_columns_image, num_to_ignore, axis=1)
28      return scaled_image
29
30  def build_empty_image(image, scaling_factor, num_to_ignore):
31      """
32          Function that adds the necessary empty columns and rows based on the scaling_factor
33
34          Parameters:
35          -----------
36          image: ndarray
37              Image matrix representing a given image where each value holds the pixel color
38
39          scaling_factor: int
40              Integer representing the scaling factor of the image to be rescaled
41
42          Returns:
43          --------
44          empty_image: ndarray
45              Image matrix representing the new matrix with the scaled new size and empty
        values
46      """
47      (M, N) = (image.shape[0], image.shape[1])
48      (new_M, new_N) = (M*scaling_factor, N*scaling_factor)
49      empty_image = np.copy(image)
50      # Rows addition
51      empty_image = add_elements(empty_image, M, new_M, num_to_ignore, axis=0)
52      # Columns addition
53      empty_image = add_elements(empty_image, N, new_N, num_to_ignore, axis=1)
54      return empty_image
```

1.1.3. Zoom sobre una imagen

```
1   def zoom_image(image, x, size_x, y, size_y):
2       """
3           Function that returns the zoomed image given a set of parameters
4
5           Parameters:
6           -----------
7           image: ndarray
8               Matrix representing the image with RGB values
9           <x,y>: int
10              Initial x or y position
11          <size_x, size_y>: int
12              Number of pixels to obtain starting at x or y
13
14          Return:
15          -------
16          Image at given positions
17      """
18      return image[y:y+size_y+1][:,x:x+size_x+1]
```

Referencias

[1] Garduño E, "Notas de Visualización de la Información: Interpolación", Marzo del 2021, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México.

[2] Garduño E, "Notas de Visualización de la Información: Normalización", Marzo del 2021, Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México.

[3] Plotly, "3D Isosurface Plots"', [En línea] Revisado el 14 de marzo del 2021 en: `https://plotly.com/python/3d-isosurface-plots/`