

# Isotropic Diffusion via Heat Equation

## import library

```
In [34]: import numpy as np  
import matplotlib.image as img  
import matplotlib.pyplot as plt  
import matplotlib.colors as colors
```

## load image

```
In [35]: I_color = img.imread('222.jpg') # load color image
```

## check the size of image and convert 3D to 2D if necessary

```
In [36]: def rgb2gray(rgb):
    r, g, b = rgb[:, :, 0], rgb[:, :, 1], rgb[:, :, 2]
    gray = 0.3333 * r + 0.3333 * g + 0.3333 * b
    return gray

n_dimension = l_color.shape[2]          # number of dimension of data
print(l_color.shape[2])

if n_dimension == 3:
    l = rgb2gray(l_color)

n_dimension = 2      # number of dimension of the converted data
n_row       = l.shape[1]      # number of rows of data
n_column    = l.shape[0]      # number of columns of data

print("number of dimension = ", n_dimension)
print("number of rows = ", n_row)
print("number of columns = ", n_column)

plt.figure(figsize=(16,8))

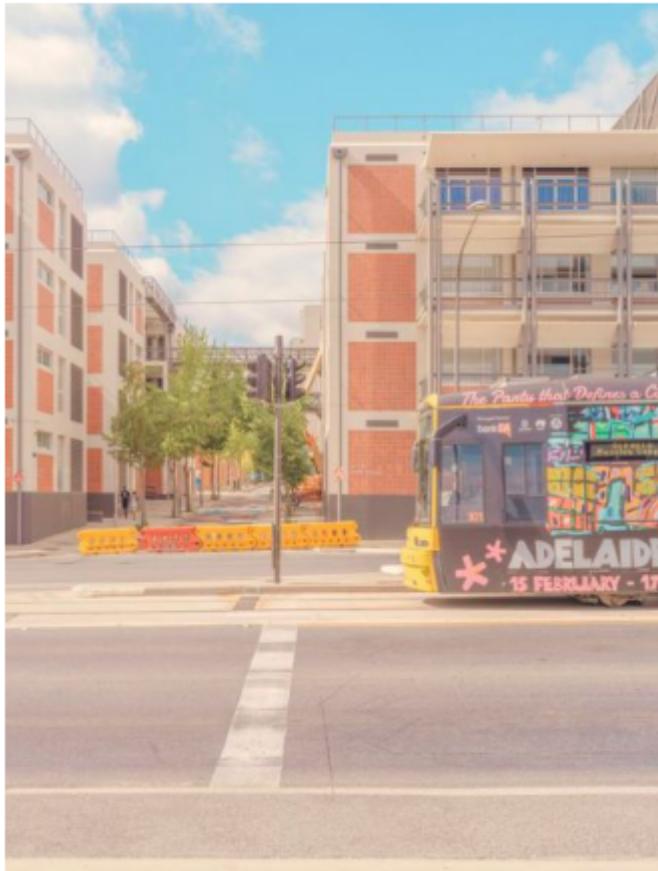
plt.subplot(121)
plt.imshow(l_color)
plt.title('input image in color')
plt.axis('off')

plt.subplot(122)
plt.imshow(l, cmap = 'gray')
plt.title('input image in grey')
plt.axis('off')
```

```
3
number of dimension = 2
number of rows = 433
number of columns = 577
```

```
Out[36]: (-0.5, 432.5, 576.5, -0.5)
```

input image in color



input image in grey



```
git commit -a -m "load image and check its size"  
git push origin master
```

## normalize input image so that the range of image is [0, 1]

```
In [37]: I /= 255  
print(I)  
  
[[0.89664235 0.89664235 0.89664235 ... 0.73848824 0.74502353 0.74502353]  
[0.89664235 0.89664235 0.89664235 ... 0.74633059 0.75155882 0.75286588]  
[0.8888 0.8888 0.8888 ... 0.75678706 0.76070824 0.75940118]  
...  
[0.85350941 0.85350941 0.84958824 ... 0.86527294 0.86919412 0.86658 ]  
[0.86135176 0.86135176 0.85743059 ... 0.86396588 0.86788706 0.86396588]  
[0.86527294 0.85743059 0.85350941 ... 0.85743059 0.85743059 0.85220235]]
```

```
git commit -a -m "normalise data"  
git push origin master
```

## define a function to compute the first-order derivatives with Neumann boundary condition

```
In [38]: print(I)
D = np.pad(I, (1,1), 'constant', constant_values=0)

# print(D)

# derivative = np.roll(D ,1, axis=0 ) -D

# print(derivative)
# y=derivative[1:-1,:]
# y2= y[:,1:-1]
# print(y2)
```

```
[ [0.89664235 0.89664235 0.89664235 ... 0.73848824 0.74502353 0.74502353]
[0.89664235 0.89664235 0.89664235 ... 0.74633059 0.75155882 0.75286588]
[0.8888     0.8888     0.8888     ... 0.75678706 0.76070824 0.75940118]
...
[0.85350941 0.85350941 0.84958824 ... 0.86527294 0.86919412 0.86658   ]
[0.86135176 0.86135176 0.85743059 ... 0.86396588 0.86788706 0.86396588]
[0.86527294 0.85743059 0.85350941 ... 0.85743059 0.85743059 0.85220235]]
```

```
In [39]: def compute_derivative_first_order(data, axis, scheme, boundary):

    if boundary == 'neumann':
        D = np.pad(data, ((1,1),(1,1)), 'constant', constant_values=0) # use numpy.pad function

    if axis == 'x':
        if scheme == 'forward':
            derivative = np.roll(D,1, axis=1) - D # use numpy.roll function
        elif scheme == 'backward':
            derivative = np.roll(D,-1, axis=1) - D # use numpy.roll function
        else: # scheme == central
            derivative = np.roll(D,-1, axis=1) - D # use numpy.roll function

    elif axis == 'y':
        if scheme == 'forward':
            derivative = np.roll(D,1, axis=0) - D # use numpy.roll function
        elif scheme == 'backward':
            derivative = np.roll(D,-1, axis=0) - D # use numpy.roll function
        else: # scheme == central
            derivative = np.roll(D,-1, axis=0) - D # use numpy.roll function

    dd = derivative[1:-1,:] # remove the first and the last rows and columns for the boundary condition
    dD= dd[:,1:-1]
    return(dD)
print()
```

```
[0.89664235 0.89664235 0.89664235 ... 0.73848824 0.74502353 0.74502353]
[0.89664235 0.89664235 0.89664235 ... 0.74633059 0.75155882 0.75286588]
```

```
[0.8888 0.8888 0.8888 ... 0.75678706 0.76070824 0.75940118]
...
[0.85350941 0.85350941 0.84958824 ... 0.86527294 0.86919412 0.86658 ]
[0.86135176 0.86135176 0.85743059 ... 0.86396588 0.86788706 0.86396588]
[0.86527294 0.85743059 0.85350941 ... 0.85743059 0.85743059 0.85220235]]
```

```
git commit -a -m "define a function for computing the first-order derivative"
git push origin master
```

## define a function to compute the second-order derivatives with Neumann boundary condition

```
In [40]: def compute_derivative_second_order(data, axis):

    D_forward = compute_derivative_first_order(data, axis, 'forward', 'neumann')
    D_backward = compute_derivative_first_order(data, axis, 'backward', 'neumann')

    dDdD = D_forward - D_backward # use the above D_forward and D_backward

    return(dDdD)
```

```
git commit -a -m "define a function for computing the second-order derivative"
git push origin master
```

## define a function to compute heat equation

```
In [41]: def heat_equation(data, delta_t, number_iteration):
    u = data    # initialisation
    for t in range(number_iteration):
        laplace = compute_derivative_second_order(data, axis='x') +compute_derivative_second_order(data, axis='y') # use the above
        u      += delta_t*laplace # compute the heat equation u(t+1) = u(t) + delta_t * laplace(u)
    return(u)
print(1)
```

```
[[0.89664235 0.89664235 0.89664235 ... 0.73848824 0.74502353 0.74502353]
 [0.89664235 0.89664235 0.89664235 ... 0.74633059 0.75155882 0.75286588]
 [0.8888     0.8888     0.8888     ... 0.75678706 0.76070824 0.75940118]
 ...
 [0.85350941 0.85350941 0.84958824 ... 0.86527294 0.86919412 0.86658    ]
 [0.86135176 0.86135176 0.85743059 ... 0.86396588 0.86788706 0.86396588]
 [0.86527294 0.85743059 0.85350941 ... 0.85743059 0.85743059 0.85220235]]
```

```
In [42]: delta_t = 0.1 # delta t in the computation of heat equation
print(l)

u10    = heat_equation(l, delta_t, 10)      # compute heat equation with 10 iterations

u100   = heat_equation(l, delta_t, 100)     # compute heat equation with 100 iterations

u1000  = heat_equation(l, delta_t, 1000)    # compute heat equation with 1000 iterations

plt.figure(figsize=(16,16))

# visualise the original image
plt.subplot(221)
plt.imshow(l, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')
print(l)
```

```
[0.89664235 0.89664235 0.89664235 ... 0.73848824 0.74502353 0.74502353]
[0.89664235 0.89664235 0.89664235 ... 0.74633059 0.75155882 0.75286588]
[0.8888 0.8888 0.8888 ... 0.75678706 0.76070824 0.75940118]
...
...
```

```
[0.85350941 0.85350941 0.84958824 ... 0.86527294 0.86919412 0.86658     ]
[0.86135176 0.86135176 0.85743059 ... 0.86396588 0.86788706 0.86396588]
[0.86527294 0.85743059 0.85350941 ... 0.85743059 0.85743059 0.85220235]]
[[-1.05252589e+29  3.26000268e+29  3.29646267e+29 ... -1.29209510e+30
  1.48277597e+30  4.31020981e+29]
[ 3.18451942e+29  4.41477519e+29 -8.38797256e+29 ...  4.45382912e+30
  1.68634335e+30 -1.49613439e+30]
[ 3.43176921e+29 -8.02892886e+29 -1.06751706e+30 ...  3.64801039e+30
 -4.51426890e+30 -1.21987836e+30]
...
[-1.26599277e+30  3.95571756e+31  2.81756280e+30 ... -4.56679281e+31
  1.27937214e+31  1.60037118e+31]
[ 1.35413566e+31  1.45661544e+30 -3.98875056e+31 ...  1.28104309e+31
  2.13122502e+31 -4.53002830e+30]
[ 3.97391208e+29 -1.36090030e+31 -8.52357615e+29 ...  1.56570543e+31
 -4.53254440e+30 -5.48379345e+30]]
```

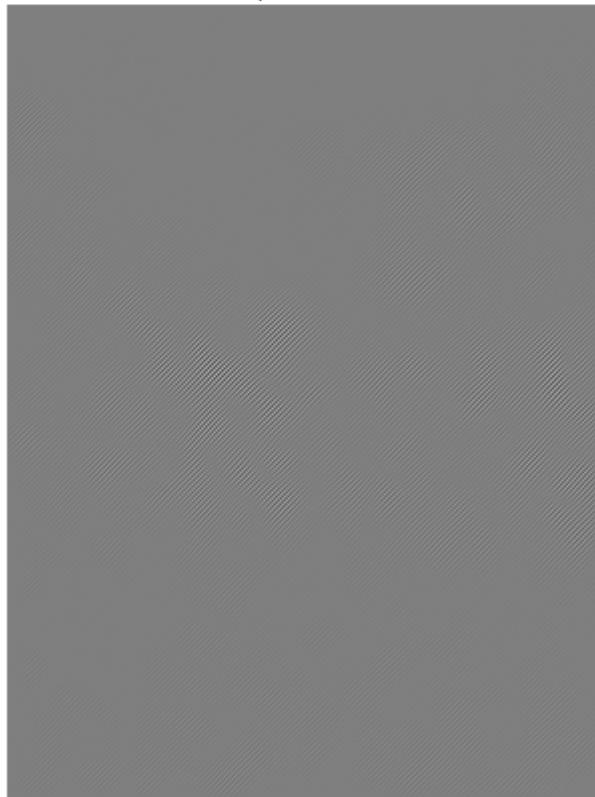
input image



heat equation t = 10



heat equation t = 100



heat equation t = 1000



```
git commit -a -m "compute the heat equation"
```

```
git push origin master
```

**compute the (forward) first-order derivative of the solutions obtained by heat equation**

```
In [43]: lx = compute_derivative_first_order(l, 'x', 'forward', 'neumann') # compute l_x using forward scheme
ly = compute_derivative_first_order(l, 'y', 'forward', 'neumann') # compute l_y using forward scheme

u10_x = compute_derivative_first_order(u10, 'x', 'forward', 'neumann') # compute u(10)_x using forward scheme
u10_y = compute_derivative_first_order(u10, 'y', 'forward', 'neumann') # compute u(10)_y using forward scheme

u100_x = compute_derivative_first_order(u100, 'x', 'forward', 'neumann') # compute u(100)_x using forward scheme
u100_y = compute_derivative_first_order(u100, 'y', 'forward', 'neumann') # compute u(100)_y using forward scheme

u1000_x = compute_derivative_first_order(u1000, 'x', 'forward', 'neumann') # compute u(1000)_x using forward scheme
u1000_y = compute_derivative_first_order(u1000, 'y', 'forward', 'neumann') # compute u(1000)_y using forward scheme

plt.figure(figsize=(16,16))

# visualise the first-order derivative in x-axis of the original image
plt.subplot(221)
plt.imshow(lx, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the first-order derivative in x-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_x, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the first-order derivative in x-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_x, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the first-order derivative in x-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_x, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')

plt.figure(figsize=(16,16))

# visualise the first-order derivative in y-axis of the original image
```

```
plt.subplot(221)
plt.imshow(Iy, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the first-order derivative in y-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_y, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the first-order derivative in y-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_y, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the first-order derivative in y-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_y, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')
```

Out[43]: (-0.5, 432.5, 576.5, -0.5)

```
git commit -a -m "compute the first-order derivative of the solution from the heat equations"
```

```
git push origin master
```

## **compute the second-order derivative of the solutions obtained by heat equation**

```
In [41]: lxx      = compute_derivative_second_order(l, 'x' ) # compute second derivative l_xx in x-direction
lyy      = compute_derivative_second_order(l, 'y' ) # compute second derivative l_yy in y-direction

u10_xx   = compute_derivative_second_order(u10, 'x' ) # compute second derivative u(10)_xx in x-direction
u10_yy   = compute_derivative_second_order(u10, 'y' ) # compute second derivative u(10)_yy in y-direction

u100_xx  = compute_derivative_second_order(u100, 'x' ) # compute second derivative u(100)_xx in x-direction
u100_yy  = compute_derivative_second_order(u100, ???) # compute second derivative u(100)_yy in y-direction

u1000_xx = compute_derivative_second_order(u1000, 'x' ) # compute second derivative u(1000)_xx in x-direction
u1000_yy = compute_derivative_second_order(u1000, ???) # compute second derivative u(1000)_yy in y-direction

plt.figure(figsize=(16,16))

# visualise the second-order derivative in x-axis of the original image
plt.subplot(221)
plt.imshow(lxx, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_xx, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_xx, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the second-order derivative in x-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_xx, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')

plt.figure(figsize=(16,16))

# visualise the second-order derivative in y-axis of the original image
```

```
plt.subplot(221)
plt.imshow(lyy, cmap='gray')
plt.title('input image')
plt.axis('off')

# visualise the second-order derivative in y-axis of the results of heat equation with 10 iterations
plt.subplot(222)
plt.imshow(u10_yy, cmap='gray')
plt.title('heat equation t = 10')
plt.axis('off')

# visualise the second-order derivative in y-axis of the results of heat equation with 100 iterations
plt.subplot(223)
plt.imshow(u100_yy, cmap='gray')
plt.title('heat equation t = 100')
plt.axis('off')

# visualise the second-order derivative in y-axis of the results of heat equation with 1000 iterations
plt.subplot(224)
plt.imshow(u1000_yy, cmap='gray')
plt.title('heat equation t = 1000')
plt.axis('off')
```

Out[41]: (-0.5, 511.5, 511.5, -0.5)

input image

heat equation t = 10

```
git commit -a -m "compute the second-order derivative of the solution from the heat equations"
```

```
git push origin master
```

## define functions for the visualization

```
In [42]: def plot_image_color(l, title = 'title'):

    plt.figure(figsize=(8,8))
    plt.imshow(l)
    plt.title(title)
    plt.axis('off')

def plot_image_gray(l, title = 'title'):

    plt.figure(figsize=(8,8))
    plt.imshow(l, cmap='gray')
    plt.title(title)
    plt.axis('off')

def plot_image_gray_2x1(l1, l2, title1 = 'title 1', title2 = 'title 2'):

    plt.figure(figsize=(16,8))

    plt.subplot(121)
    plt.imshow(l1, cmap = 'gray')
    plt.title(title1)
    plt.axis('off')

    plt.subplot(122)
    plt.imshow(l2, cmap = 'gray')
    plt.title(title2)
    plt.axis('off')

def plot_image_gray_2x2(l1, l2, l3, l4, title1 = 'title 1', title2 = 'title 2', title3 = 'title 3', title4 = 'title 4'):

    plt.figure(figsize=(16,16))

    plt.subplot(221)
    plt.imshow(l1, cmap = 'gray')
    plt.title(title1)
    plt.axis('off')

    plt.subplot(222)
    plt.imshow(l2, cmap = 'gray')
    plt.title(title2)
    plt.axis('off')
```

```
plt.subplot(223)
plt.imshow(I3, cmap = 'gray')
plt.title(title3)
plt.axis('off')

plt.subplot(224)
plt.imshow(I4, cmap = 'gray')
plt.title(title4)
plt.axis('off')
```

```
git commit -a -m "define functions for the visualization"
```

```
git push origin master
```

```
# ######
#
```

## # results

```
# ######
#
```

### # 01. plot the input image in color

```
In [43]: plot_image_color(I_color, 'input image in color')
```



# 02. plot the input image in gray

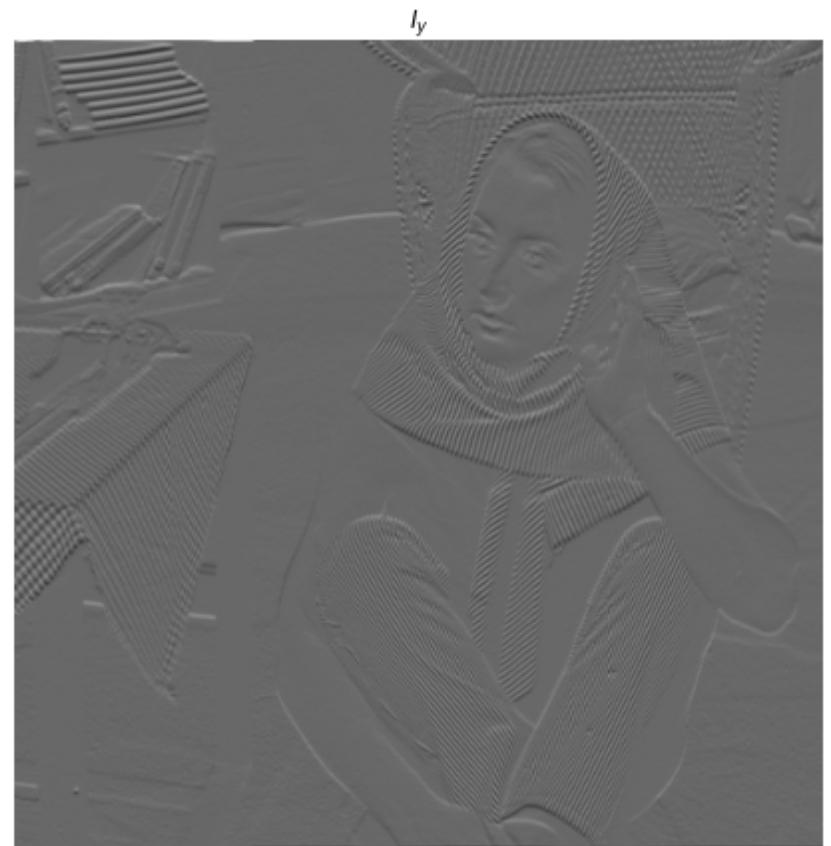
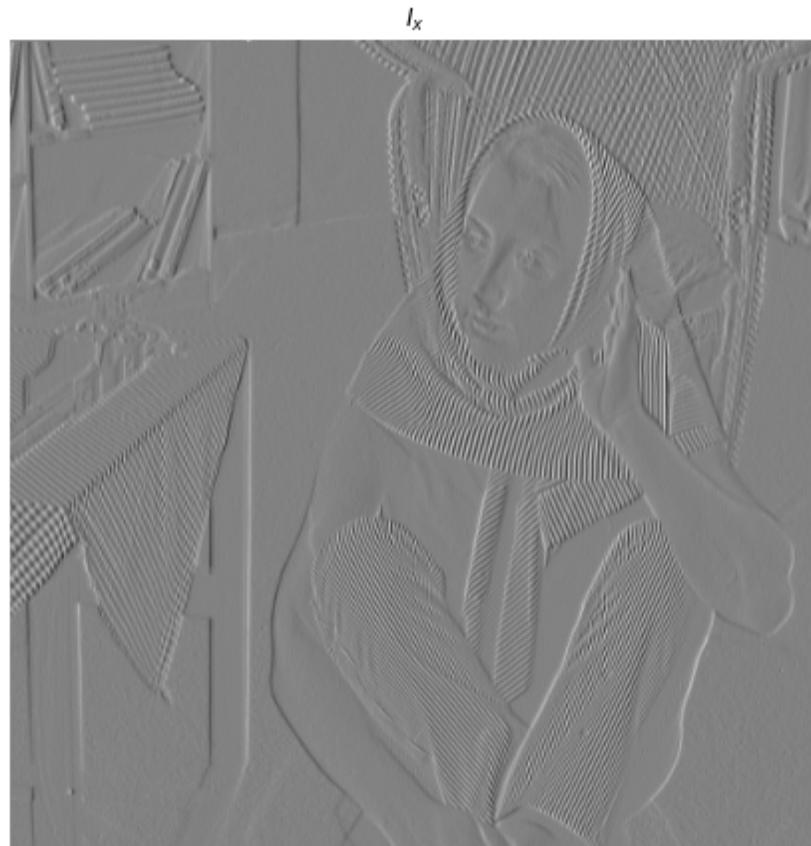
```
In [44]: plot_image_gray(I, 'input image in grey')
```

input image in grey



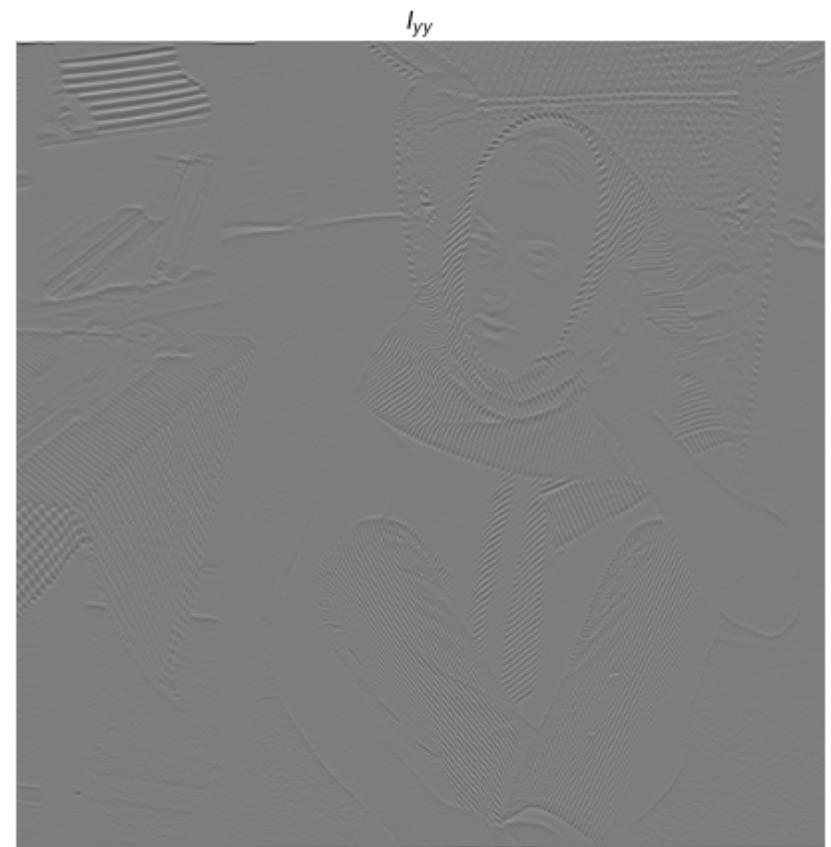
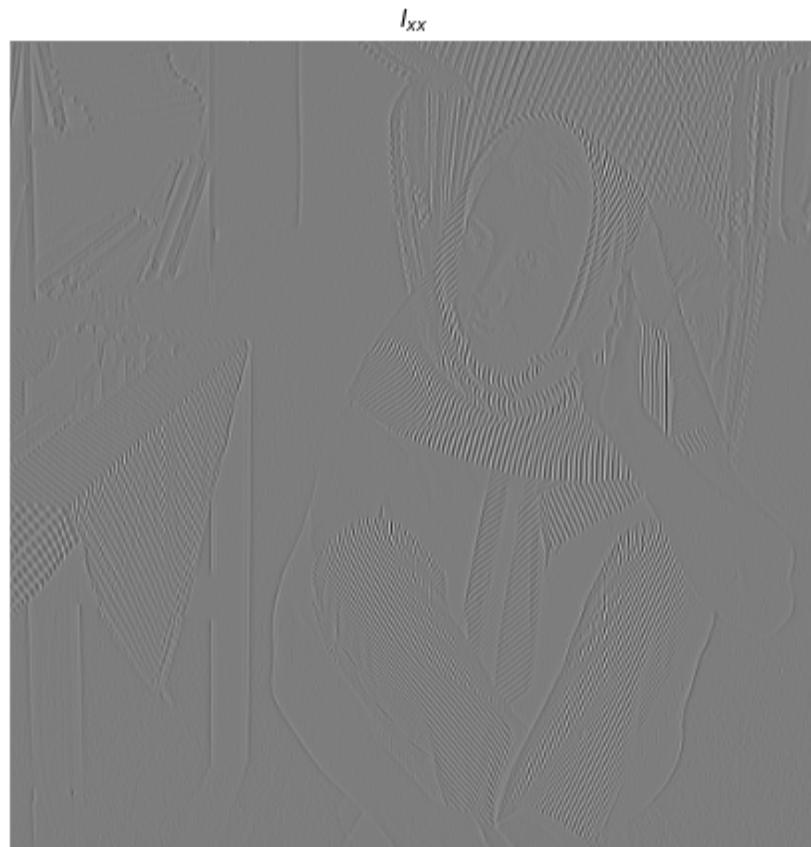
**# 03. plot the (forward) first derivative  $I_x$  and  $I_y$  of input gray image  $I$  with Neumann boundary condition**

```
In [45]: plot_image_gray_2x1(Ix, ly, '$I_x$', '$I_y$')
```



**# 04. plot the second derivative  $I_{xx}$  and  $I_{yy}$  of input gray image  $I$  with Neumann boundary condition**

```
In [46]: plot_image_gray_2x1(lxx, lyy, '$l_{xx}$', '$l_{yy}$')
```



**# 05. plot the original image and its solution of the heat equation with 10, 100, 1000 iterations**

```
In [47]: plot_image_gray_2x2(l, u10, u100, u1000, 'original $l$', '$u(t=10)$', '$u(t=100)$', '$u(t=1000)$')
```

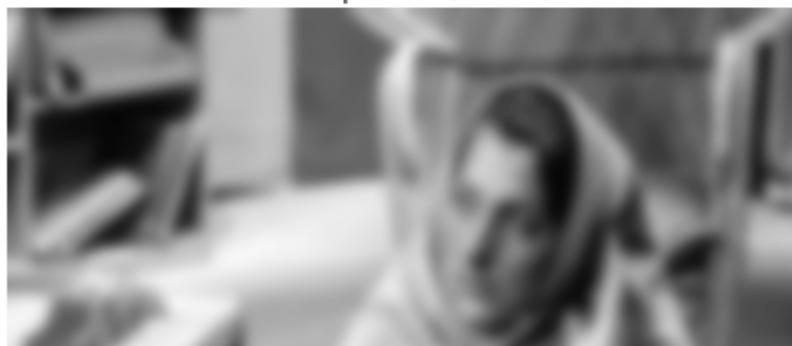
original  $l$



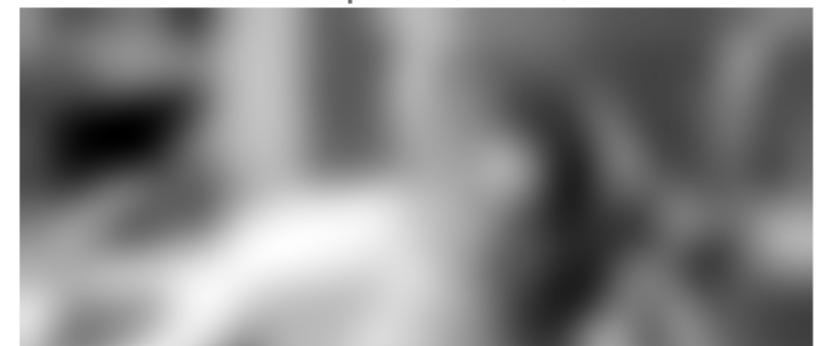
heat equation  $u(t = 10)$



heat equation  $u(t = 100)$



heat equation  $u(t = 1000)$





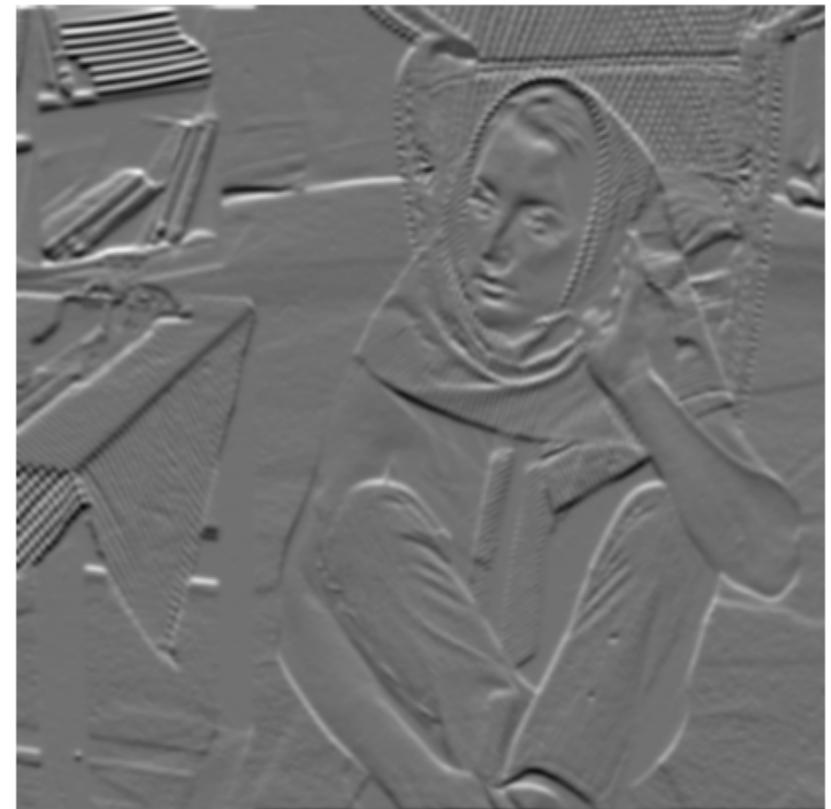
**# 06. plot the (forward) first derivative of the solution of the heat equation with 10 iterations**

```
In [48]: plot_image_gray_2x1(u10_x, u10_y, '$u_x(t=10)$', '$u_y(t=10)$')
```

$u_x(t = 10)$



$u_y(t = 10)$



**# 07. plot the (forward) first derivative of the solution of the heat equation with 100 iterations**

```
In [49]: plot_image_gray_2x1(u100_x, u100_y, '$u_x(t=100)$', '$u_y(t=100)$')
```

$u_x(t = 100)$



$u_y(t = 100)$



**# 08. plot the (forward) first derivative of the solution of the heat equation with 1000 iterations**

```
In [50]: plot_image_gray_2x1(u1000_x, u1000_y, '$u_x(t=1000)$', '$u_y(t=1000)$')
```

$u_x(t = 1000)$



$u_y(t = 1000)$



