COMP0115 Coursework 1

Report for Numerical Implementation of Diffusion

1 Introduction

Diffusion, is a time-dependent physical phenomenon which describes the movement of particles from a concentrated group to an unconcentrated group. Mathematically, a diffusion equation is a partial differential equaltion which encapsulates time and moving direction — gradient.

In terms of image processing, diffusion is a evolution process of an image over time t where the image get smoothed in all directions. In this report, I will show how I achieved such process based on numerical implementation of diffusion over three different aspects:

- 1-D Linear Diffusion
- 2-D Linear Diffusion
- 2-D Anisotropic Diffusion

All processing procedure are performed over a grey level image: Lenna

Figure 1: Lenna.jpg

2 Numerical Implementation of Diffusion

2.1 1-D Linear Diffusion

The equation for 1-D Linear Diffusion is:

$$\frac{\partial f}{\partial t} = \frac{\partial^2 f}{\partial x^2}$$

For image processing purpose, a discretize approximation is necessary. A general implementation procedure for such approximation can be seen as follows:

1. Choose N number of points.
2. Create N by N Laplacian Matrix A
3. Solve an iterative scheme :

Explicit scheme:	$f^{k+1} = f^k + \Delta t \mathsf{A} f^k$
Implicit scheme:	$(I - \Delta t \mathsf{A}) f^{k+1} = f^k$

Where Δt is an choosen time step.

Start with the first procedure, since we are developing in 1 dimension, we cannot use the input image directly. Therefore, I samples one horizontal line of pixels from that image. This line is also used for initialzation for f^0.

Next, I we need to construct Laplacian Matrix A, it is a N by N matrix with following looks:

$$\mathsf{A} = \begin{bmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ 0 & 0 & 1 & -2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

Where the diagonal of A is -2 except A_{11} and A_{NN} which are -1. These diagonal elements are neighboured by 1s. The rest of A are all 0s. The magnitude along diagonal represents the number of connected neighbours and those 1s represents diagonal's adjacents' location.

From now on, we can compute f iteratively based on two schemes. By keeping value of f for each iteration, we can visualize the evolution of diffusion over time step t. The general idea for iteration implementation is to create functions with resepct to each schemes, then put those functions into for-loop. The paramerters of functions is the output from the same function but from last iteration.

In explicit scheme, parameters are computed based on previous time step. So it is quite straightforward implementation by just following the given equation.

In terms of implicit scheme, where all parameters are computed simultaneously. In order to put two functions into same for-loop. The Implicit scheme now becomes:

$$f^{k+1} = f^k \backslash (I - \Delta t \mathsf{A}) f^{k+1}$$

Which invloves matrix inverse operation. Above gives all necessities for implementation for 1-D linear diffusion, put those implementation into matlab we get the following results as shown in Figure 2:

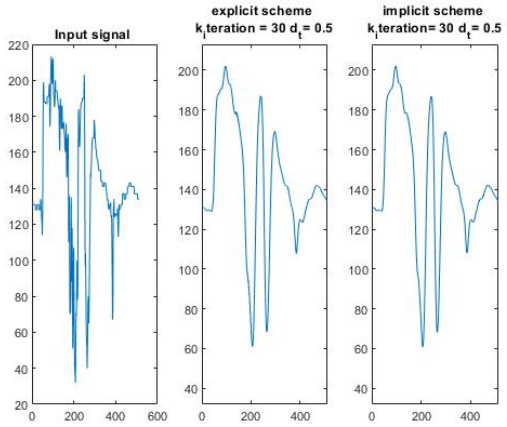
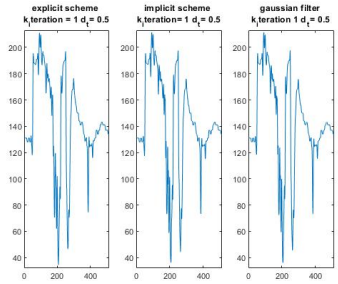


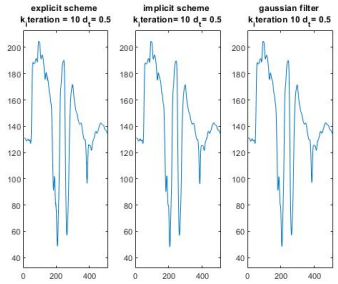
Figure 2: Scheme results comparison

As shown in the figure, I choose Δ to be 0.5 to make explicit scheme stable.

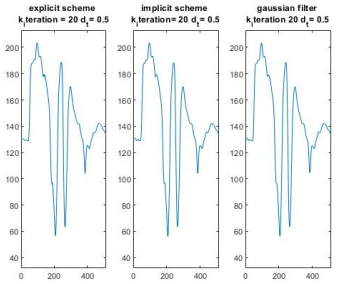
It is also possible that we can use convolution of f^0 with a Gaussian filter to get f^k. The standard deviation of such fiter is $\sigma = \sqrt{2k\Delta t}$. Matlab does a perfect job in implementing this by just using two functions: `fspecial` and `imfilter`.



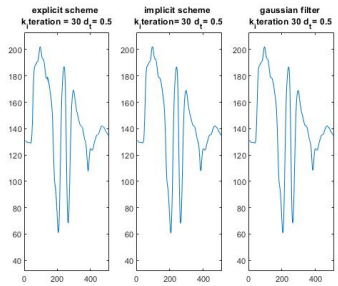
(a) k=1



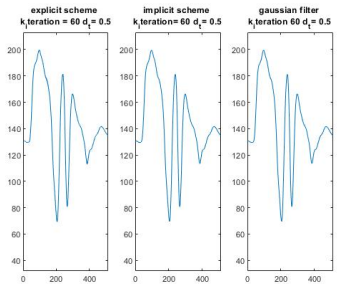
(b) k=5



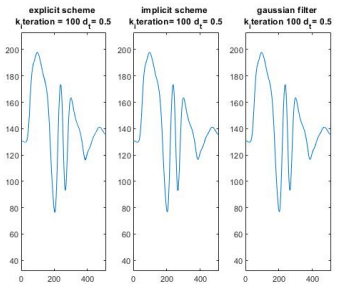
(c) k=10



(d) k=15



(e) k=20



(f) k=25

Figure 3: 1-D Linear Diffusion results at dt = 0.5

Figure 3 gives the 1-D linear filter results over different time steps, we can see the signals get smoothed over time and the use of convolution with Gaussian filter gives as identical result as the other two.

2.2 2-D Linear Diffusion

The equation for 2-D linear diffusion is:

$$\frac{\partial f}{\partial t} = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

The implementation of 2-D linear diffusion shares similar approach as 1-D, except the construction of matrix A is slightly different. This time, we need to reshape f in to N^2 by 1 vector first then forming a N^2 by N^2 matrix A. Since A now is extremely large, for the sake of memory concern, we should consider converting such dense matrix into a sparsed matrix.

The implicit implementation involves matrix inversion. However, in 2-D, it is computationally intensive for inversing A even it is sparsed. Thus, we need to split A with

respect to direction: A^{x} and A^{y}. Such implementation is named *semi-implicit scheme*:

$$\left(I - \frac{\Delta t}{2}\mathsf{A}^{\mathsf{x}}\right) f^{k+1/2} = \left(I + \frac{\Delta t}{2}\mathsf{A}^{\mathsf{y}}\right) f^{k}$$
$$\left(I - \frac{\Delta t}{2}\mathsf{A}^{\mathsf{y}}\right) f^{k+1} = \left(I + \frac{\Delta t}{2}\mathsf{A}^{\mathsf{x}}\right) f^{k+1/2}$$

For matlab implementation we can use function `spdiags` as suggensted by the coursework page:

```
IAx = speye(N*N) - deltaT/2 * spdiags(dd,range,N*N,N*N)
f = IAx \b
```

where `dd` is a N^2 by 3 matrix with each row of $[-1\ \ 2\ \ 1]$.`dd` is $[-1\ \ 0\ \ 1]$ for A^{x} and $[-N\ \ 0\ \ N]$ for A^{y}.

By following the same procedure as 1-D linear diffusion, we completes the implementation of 2-D linear diffusion.

(a) k=1 (b) k=5 (c) k=10

(d) k=15 (e) k=20 (f) k=25

Figure 4: 2-D Linear Diffusion at dt = 0.5

Figure 4 shows the results of 2-D linear diffusion over diffter time steps. For each diffusion, I also tracking the local maxima on top left quadrant labelled by red cross. For different we can see red cross slight movement. From k 5 to 10, the local maxima changes because the new local maxima annihalates the previous one.

2.3 2-D Anisotropic diffusion

The equation for 2-D Anisotropic diffusion is

$$\frac{\partial f}{\partial t} = \frac{\partial}{\partial x}\gamma\frac{\partial f}{\partial x} + \frac{\partial}{\partial y}\gamma\frac{\partial f}{\partial y}$$

So we can have different diffusivity for different directions. The difference between anisotropic and linear diffusion is that anisotropic introduces the γ term which is a diffusivity function, we use Perona-Malik function to implement this scheme:

$$\gamma = \frac{1}{1 + (\frac{|\nabla f|}{T})^2}$$

Where T is a threshold to control the response of γ for various size of edges. My choice of threshold, is to take the mean of input image.

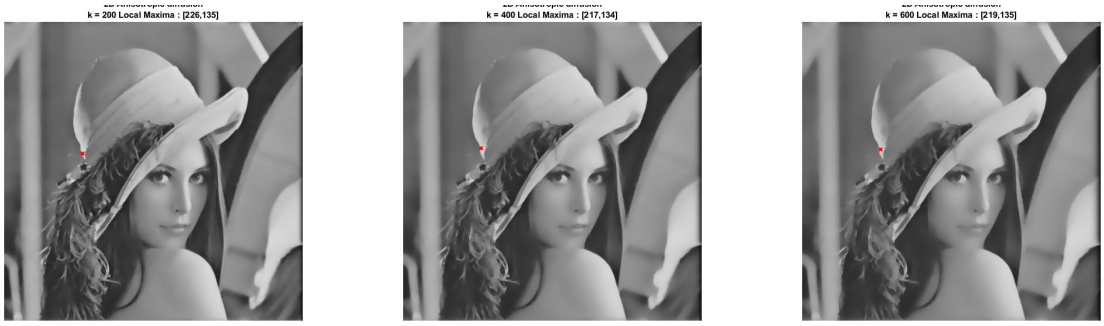
Before iteration, firstly we need to create explict 1^{st} order derivative operaters Dx and Dy. Both of them are N^2 by N^2 matrix so use `spdiags` to create the sparsed one.

Now we use the coursework suggested function to compute the diffusion:

```
gam = spdiags(reshape(gamma,[],1),0:0,N*N,N*N)
PM =  - Dx' * gam * Dx + Dy' * gam * Dy
```

As what we did in previous section, use explicit scheme to implement the 2-D anisotropic diffusion. Same as 2-D linear diffusion, I compute a series of f over different steps with $d_t = 0.05$ and track the local maxima. During the computation, derivative Dx and Dy should be computed for every iteration same applies to PM. However, I later find out that we can achieve indistinguishable results with just one computation of those terms but faster. Which is to take those terms out of loops. The reson for that is, the edge information is preserved for an image. The anisotropic diffusion does not wipe those edges, so for every iteration, the edge information remained is similar.

(a) k=1 (b) k=50 (c) k=100



(d) k=200 (e) k=400 (f) k=600

Figure 5: 2-D Anisotropic Diffusion at dt = 0.05

The Figure 5 gives the 2-D anisotropic results at different time steps. The image gets blurred gradually but not like the one in linear diffusion where it likes a global blur, anisotropic diffusion blurred image but preserves the edge contents, so the result looks shaper than linear diffusion.