R-refinement in image-processing via the PDE-approach

Kenery E. Oron
(9536574)

Universiteit Utrecht

July 5, 2011

Supervisor: Dr. Paul A. Zegeling

Second reader: Prof. dr. Rob H. Bisseling

Contents

Abstract

(Digital) Signal Processing plays a huge role in computer vision. We will use two related Partial Differential Equations (PDEs), known for their smoothing feature, to investigate the removal of noise in (digital) signals, namely: the Heat and Perona-Malik equation. This report explains how we can do (digital) signal processing on a bounded domain $\Omega \subset \mathbb{R}^n$ $(n = 1, 2)$, via a PDE approach. Depending on the type of noise present in the signal, the PDE approach gives *desirable* results. For faster iteration with the Perona-Malik equation we first need an (un)conditionally stable finite difference method or use a non uniform grid with R-refinement (adaptive grids) for a possibly better edge detection.

Keywords: Finite Difference methods, Heat equation, Perona-Malik[1] equation, Digital Signal Processing, Image Processing, noise reduction (denoising), (anisotropic) diffusion, refinement.

1 Introduction

For several years now, Partial Differential Equations (PDEs) are used to restore images. There are also other methods for doing this such as: linear filters and wavelets. PDEs are also used for image analysis. One of the well known models which has been used before *better* models emerged, are the heat equation and the inverse heat equation [P01]. In this report we will study image processing via a PDE approach. Image processing can be divided in three parts: *image compression*, *image restoration* and *image analysis*. We will do some image restoration, where our main focus will be comparing a *special* PDE to the heat equation. Since in practice we do not have the original (*good*) image when doing image processing, we will need stopping criteria. These stopping criteria mean that we have reached a point where the obtained image is clearer compared to the original *bad* image, and will not get better if we iterate further on.

Before doing image processing we will need to investigate or consider the following points concerning the PDEs to be used:

- a *good* and stable finite difference (FD) scheme for the PDE,
- measurable stopping criteria,
- using uniform or non-uniform grid, depending on the image.

In the following two chapters we will investigate each one of these points. We will consider the bounded domain $\Omega \subset \mathbb{R}^n$, with the smooth boundary $\partial\Omega$. Only $n = 1, 2$ will be investigated in chapter 2 respectively chapter 3.

[1] Pietro Perona (1961): his research interests are in computational and biological vision.
Jitendra Malik (1960): his research interests are in machine vision and computational modeling of early human vision. These include work on edge detection, texture segmentation, line drawing interpretation, and 3-D object recognition [P02]

2 Digital Signal Processing via the PDE Approach

In the last few years Digital Signal Processing (DSP) has grown rapidly. The application of DSP can be found in several technologies. We may define a signal as any time-varying or spatial-varying quantity. Sampling of an analog signal is required in order to use that signal on the computer, which usually is done in the following two steps [L01]:

- discretization; transferring continuous models to discrete counterparts,
- quantization; mapping of large set of input values to a smaller set.

In this report we will only use digital signals, and skip the digitizing process of analog signals. The signals used in this chapter are artificial ones, but real signals can also be used. Our aim is to use also an example where (jump) discontinuity is involved. The main focus will be to remove noise from signals, also known as denoising or noise reduction. This noise reduction procedure will be done via the Partial Differential Equation (PDE) approach.

In the coming sections we will discuss some Partial Differential Equations (PDEs) for noise reduction, and their pros and cons, which are applicable. The PDEs which will be discussed are: the well known one-dimensional diffusion equation of the form

$$u_t = k u_{xx}, \tag{1}$$

also known as the Heat Equation (HE) and a Perona-Malik Equation (PME). In (1) we have that k is a constant and $u = u(x,t)$ represents the (digital) signal. In the coming sections in this chapter we will investigate the function $u(x,t)$ bounded as follows

$$0 \leq u(x,t) \leq 1, \tag{2}$$

and $u(x,0) \in \{S_1, S_2\}$. Here S_i is a still to be defined (digital) signal, for $i \in \{1,2\}$.

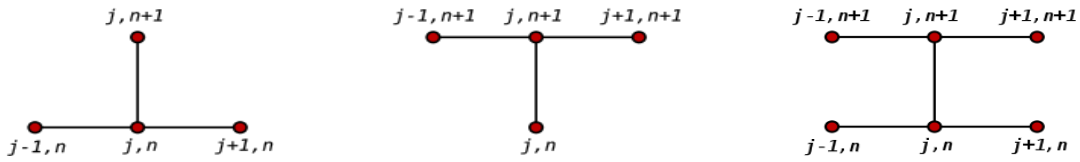
$$S_1 = \begin{cases} 1 & \text{for } x \in [0, 0.25\rangle \cup \langle 0.75, 1] \\ 0.4 & \text{for } x \in \langle 0.25, 0.4\rangle \cup \langle 0.6, 0.75\rangle \\ 0.1 & \text{for } x \in \langle 0.4, 0.6] \end{cases} \tag{3}$$

and S_2 is defined as the K-th column of the (digital) image in Figure 8(b). Some random noise will be added to these two signals[2] :

$$u_{noise} := u_{exact} \ + \ \textit{some random noise}. \tag{4}$$

Afterwards we will try to reduce the noise in the *noisy signal*, referred as u_{noise}, by applying the one-dimensional version of the HE or PME. To apply these PDEs we need to solve them and this will be done numerically. From the textbooks we know this can be done by using a Finite Difference Method (FDM) , such as the following:

[2]these two signals S_1 and S_2 will also be referred as $u_{exact} = u_{exact}(x,t)$



(a) (b) (c)

Figure 1: Some stencils for a specific Finite Difference (FD) scheme : the stencil for the (a) explicit method, (b) implicit method, (c) Crank-Nicolson method.

- Explicit method
  In this method u_j^{n+1} depends on the quantities u_{j-1}^n, u_j^n and u_{j+1}^n.
- Implicit method
  Now the quantities u_{j-1}^{n+1}, u_j^{n+1} and u_{j+1}^{n+1} all depend on u_j^n in an implicit manner.
- Crank-Nicolson method,
  In this FDM the quantities u_{j-1}^{n+1}, u_j^{n+1} and u_{j+1}^{n+1} depend on u_{j-1}^n, u_j^n and u_{j+1}^n,

where the stencils[3] of these methods can be seen in Figure 1. With the notation u_j^n we mean that

$$u_j^n := u(j\Delta x, n\Delta t),$$

is the representation of the numerical approximation from the solution we are seeking at the point $(j\Delta x, n\Delta t)$. We notice that the values of Δx and Δt are an uniform interval in the x-direction respectively t-direction. Before choosing a FDM we need to know what the pros and cons are. We first need to check the convergence, stability or consistency of the finite difference scheme. For the consistency we can use theorem 1 [B01] .

Theorem 1. ***Lax Equivalence Theorem*** *A consistent, two level difference scheme for a well-posed linear initial-value problem is convergent if and only if it is stable.*

2.1 The Heat Equation: $u_t = u_{xx}$

Now we can investigate how a Partial Differential Equation (PDE) can be applied in the denoising process of a noisy signal. Here we will consider the bounded domain $\Omega \subset \mathbb{R}^n$, with $n = 1$. By using the one-dimensional (1-D) version of the Heat Equation (HE) which has the form:

$$u_t = ku_{xx}. \tag{5}$$

We recall that

$$u_t = \frac{\partial}{\partial t}\, u(x,t) = \frac{\partial u(x,t)}{\partial t},$$

[3]Source: http://en.wikipedia.org/wiki/Finite_difference_method, last visited: May 2011.

and similarly we have that

$$u_{xx} = \frac{\partial}{\partial x}\left(\frac{\partial}{\partial x}\ u(x,t)\right) = \frac{\partial^2 u(x,t)}{\partial x^2}.$$

Since we know that k is the thermal diffusivity, we will only consider the situation $k = 1$, so we will work with the Heat equation:

$$u_t = u_{xx}. \tag{6}$$

One of the first questions that may arise is: *"why use the Heat Equation for denoising a noisy signal?"* One of the answers is the smoothing ability of the Heat equation, already made in the first few iteration steps, using any Finite Difference (FD) scheme consistent with the Heat Equation. This is only possible if we meet the stability requirements of that FD scheme.

Using an Explicit Method for approximating the Heat Equation

Knowing that Implicit methods can be expensive in computation time, we will use an Explicit method to approximate the Heat Equation. The Explicit Euler will be the method of our choice, which is also known as **E**uler **F**orward (**EF**). Firstly a uniform[4] discretization of time and space will be done, such that

$$\Delta t = \frac{1}{N} \qquad \text{and} \qquad \Delta x = \frac{1}{M}.$$

Therefore we can approximate the left-hand side (LHS) and the right-hand side (RHS) of (6). From the *mean value theorem* of calculus we know that

$$u_t = u_t(x,t) = \lim_{\Delta t \to 0} \frac{u(x, t+\Delta t) - u(x,t)}{\Delta t}, \tag{7}$$

similarly we have for the x-direction that

$$u_x = u_x(x,t) = \lim_{\Delta t \to 0} \frac{u(x+\Delta x, t) - u(x,t)}{\Delta x}, \tag{8}$$

this way[5] the LHS of (6) can be approximated by

$$u_t = \frac{\partial u(x,t)}{\partial x} \approx \frac{u_j^{n+1} - u_j^n}{\Delta t}, \tag{9}$$

and the RHS by

$$u_{xx} = \frac{\partial^2 u(x,t)}{\partial x^2} \approx \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2}, \tag{10}$$

which is a reasonable approximation [B01]. Solving (6) with **EF** leads to

$$u_j^{n+1} = u_j^n + \frac{\Delta t}{(\Delta x)^2}\Big(u_{j+1}^n - 2u_j^n + u_{j-1}^n\Big) \tag{11}$$

[4] using uniform grids in FD methods is not compulsory, but simplifies the computations

[5] we can also use $(u_x)^n_{k-\frac{1}{2}}$ and $(u_x)^n_{k+\frac{1}{2}}$ to approximate u_n^k but we can also see this by using the well known Taylor expansions, both are described in §A

By doing *Von Neumann stability analysis*, which is based on the decomposition of the errors into Fourier series [L01(d)], we find that, for solving (6) with (11), the requirement for convergence of this numerical method is

$$r = \frac{\Delta t}{(\Delta x)^2} \leq \frac{1}{2}, \tag{12}$$

indicating that this[6] FD scheme is conditionally stable. If we iterate *too long* with EF, the solution $u_{sol}(x,t)$ will get smoother, but the shape compared to $u(x,t)$ is lost. The results of the tests done with the Heat Equation are visible in §2.3.

2.2 The Perona-Malik Equation: $u_t = \frac{\partial}{\partial x}(D(u)u_x)$

In this section we will investigate the use of another PDE for noise reduction in *noisy* signals. Here we will also consider the bounded domain $\Omega \subset \mathbb{R}^n$, with $n = 1$, and use the so called one-dimensional Perona-Malik Equation (PME)

$$u_t = \frac{\partial}{\partial x}(D(u)u_x), \quad \text{with} \quad D(u) = \frac{1}{1+\epsilon|u_x|^2} \tag{13}$$

so

$$u_t = \left(\frac{u_x}{1+\epsilon|u_x|^2}\right)_x, \tag{14}$$

where $0 \leq \epsilon \leq 1$. Later on we will show the influence of ϵ on the convergence when using the PME for denoising a noisy signal. More details on the PME can be found in §3.2.

Using an Explicit Method for approximating the Heat Equation

We will use the same Explicit Method as for the HE to discretize the PME, which makes comparison easier, when setting $\epsilon = 0$.

$$\begin{aligned}
\left[\frac{u_x}{1+\epsilon|u_x|^2}\right]_x\Bigg|_{x_i} &\approx \frac{\dfrac{u_{x,i+\frac{1}{2}}}{1+\epsilon|u_{x,i+\frac{1}{2}}|^2} - \dfrac{u_{x,i-\frac{1}{2}}}{1+\epsilon|u_{x,i-\frac{1}{2}}|^2}}{\Delta x} \\
&= \frac{u_{x,i+\frac{1}{2}}}{\Delta x + \epsilon\Delta x|u_{x,i+\frac{1}{2}}|^2} - \frac{u_{x,i-\frac{1}{2}}}{\Delta x + \epsilon\Delta x|u_{x,i-\frac{1}{2}}|^2} \\
&\approx \frac{u_{i+1}-u_i}{(\Delta x)^2 + \epsilon(\Delta x)^2\left|\dfrac{u_{i+1}-u_i}{\Delta x}\right|^2} - \frac{u_i - u_{i-1}}{(\Delta x)^2 + \epsilon(\Delta x)^2\left|\dfrac{u_i-u_{i-1}}{\Delta x}\right|^2}
\end{aligned}$$

With this we find that the Euler-Forward approximation for the Perona-Malik equation is

$$u_j^{n+1} = u_j^n + \frac{\Delta t}{(\Delta x)^2}\left(\frac{u_{j+1}^n - u_j^n}{1+\epsilon\left|\dfrac{u_{j+1}^n-u_j^n}{\Delta x}\right|^2} - \frac{u_j^n - u_{j-1}^n}{1+\epsilon\left|\dfrac{u_j^n-u_{j-1}^n}{\Delta x}\right|^2}\right), \tag{15}$$

[6] the Implicit scheme and Crank-Nicolson scheme are both unconditionally stable

this can also be written as:

$$u_j^{n+1} = u_j^n + \frac{\Delta t}{(\Delta x)^2}\left(\frac{u_{j+1}^n - u_j^n}{1+\frac{\epsilon}{(\Delta x)^2}\left|u_{j+1}^n - u_j^n\right|^2} - \frac{u_j^n - u_{j-1}^n}{1+\frac{\epsilon}{(\Delta x)^2}\left|u_j^n - u_{j-1}^n\right|^2}\right). \tag{16}$$

Notice that for $\epsilon = 0$, the Perona-Malik equation becomes the heat equation (14) = (6) and also that (16) = (11).

2.3 Results

Now we can start some denoising tests by applying the Heat Equation (HE) and the Perona-Malik Equation (PME) to the signals S_1 and S_2. Before doing this we need to know when to abort the noise reducing routine using one of these Partial Differential Equations (PDEs).

Stopping criteria for the Heat and Perona-Malik PDE

If we iterate too long with both PDEs, we cannot see the original shape any longer. That's why we need to built in a stopping criterion especially for the heat equation, when solved with Euler-Forward. First we will use the signal $u_{exact} = S_1$ as described in (3). After adding some random noise to it we can apply the HE and PME to the noisy signal $u^*(x,t)$. The result is shown in Figure 2. For the error made by both used PDEs see Figure 3. For this example we can also see what the result would be by using the PME for different $\epsilon \in \{0, \frac{125}{10000}, \frac{25}{1000}, \frac{5}{1000}, \frac{1}{100}, \frac{2}{100}, \frac{8}{100}, \frac{32}{100}, \frac{64}{100}, 1\}$, see Figure 4 and Figure 5.

- **Stopping criterion for the Heat PDE, solved with Explicit Euler**
  We know that in each time-step the *local truncation error* τ, with EF is $\mathcal{O}((\Delta x)^2)$, so the *global error* in the nth iteration step is $n\tau$, so $\mathcal{O}(\Delta x)$. This since $n \sim \frac{1}{\Delta x}$. This tells us that we can stop the iteration as soon as

  $$\|u_{j+1}^{n+1} - u_j^{n+1}\|_2 \leq \Delta x,$$

  for Δx *small enough*. In Figure 6 we see the effect of this implementation. We do not get better results then for $\|u_{j+1}^{n+1} - u_j^{n+1}\|_2 \leq (\Delta x)^p$ with $p = 1$. So we are satisfied with p=1. Example: when setting $p \geq 1.2$ we can keep on iterating but will not find a better result which gives a *reasonable* approximation of the exact solution $u(x,t)$.

- **Stopping criterion for the Perona-Malik PDE, solved with Explicit Euler**
  For PM with EF we also found a suitable stopping criterion. We find that

  $$\|u_{j+1}^{n+1} - u_j^{n+1}\|_2 \leq (\Delta x)^q, \qquad \text{with} \quad q = 2,$$

  this depending on the choice of Δx. In Figure 7, it is visible that the implementation as above does help. Because of memory limitations we did not take Δx much smaller.

  Questions left to be answered are: i) Does this hold for each choice of ϵ? ii) Does this work for 2-D? The latter will be discussed, for both HE and PME, in chapter 3.

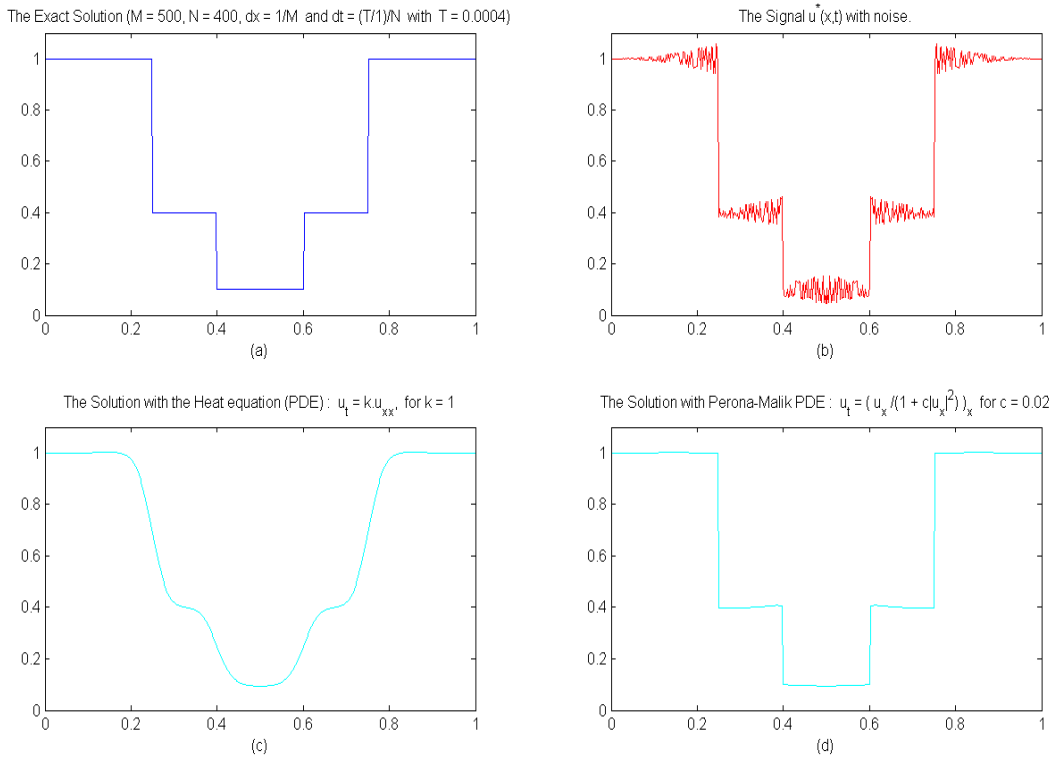


Figure 2: **(a)**: The signal $u(x,t)$, which is the exact solution for a PDE. **(b)**: The signal $u^*(x,t)$, which is $u(x,t)$ with some random generated noise. **(c)**: Solving the PDE with the Heat Equation, where the initial condition is $u_0^n = u^*(x,t)$, with $\Delta x = \frac{1}{500} = 2 \times 10^{-3}$, $\Delta t = \frac{0.0004}{400} = 10^{-6} \Rightarrow \frac{\Delta t}{(\Delta x)^2} = \frac{1}{4}$ and $k = 1$. **(d)**: Solving the PDE with the Perona-Malik PDE, where the initial condition also is $u_0^n = u^*(x,t)$, with $\Delta x = \frac{1}{500} = 2 \times 10^{-3}$, $\Delta t = \frac{0.0004}{400} = 10^{-6}$ and $c = \epsilon = 0.02$. The Heat Equation rapidly produces a smooth signal, but looses the details, which are the discontinuities in the signal. Notice that the Perona-Malik Equation nicely keep these details, and acts like a *local averaging filter* [P06].

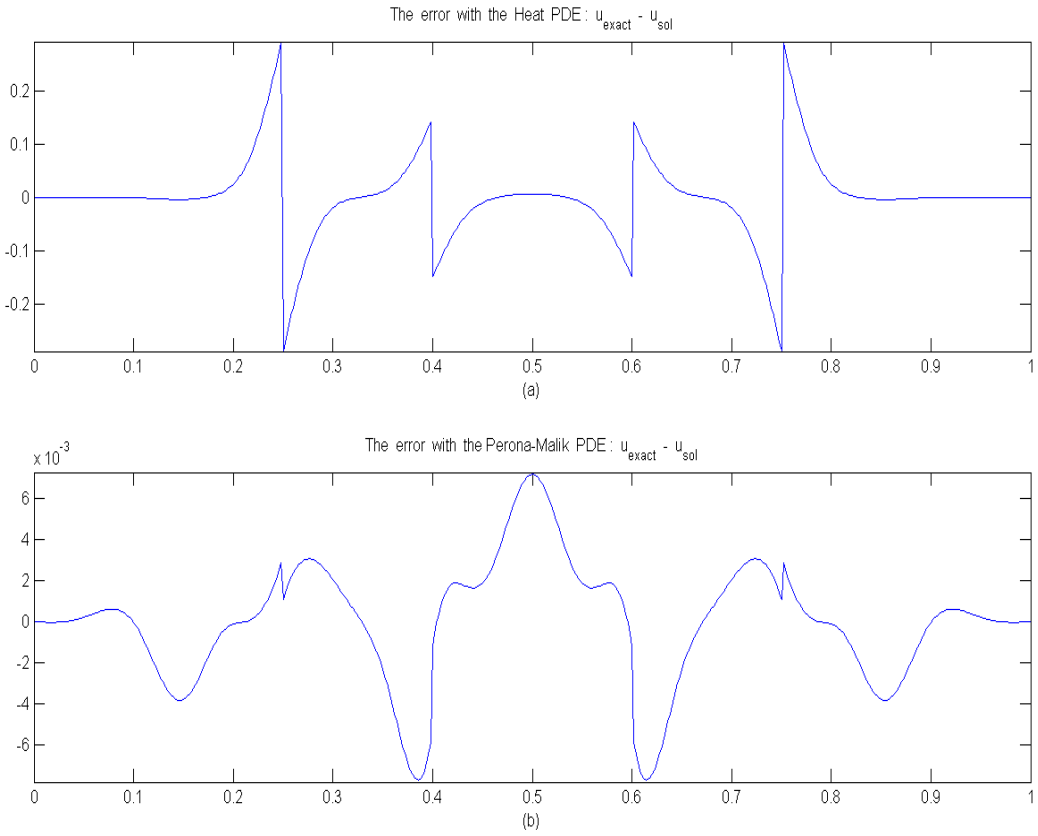


Figure 3: **(a)**: The error with the Heat PDE : $\|u_{exact} - u_{sol}\|_2 = 1.7176$. **(b)**: The error with the Perona-Malik PDE : $\|u_{exact} - u_{sol}\|_2 = 0.0655$. By only investigating the error of the PDEs used for denoising, we could guess that the exact solution has discontinuities at $x \in \{0.25,\ 0.4,\ 0.6,\ 0.75\}$. This tells us that reducing Δx might improve the solution, with the disadvantage that we need to iterate much longer, since we also need to reduce the value of Δt for stability reasons. Or that it's desirable to use a non-uniform discretization of the x-direction in this case, what will not be treated here.

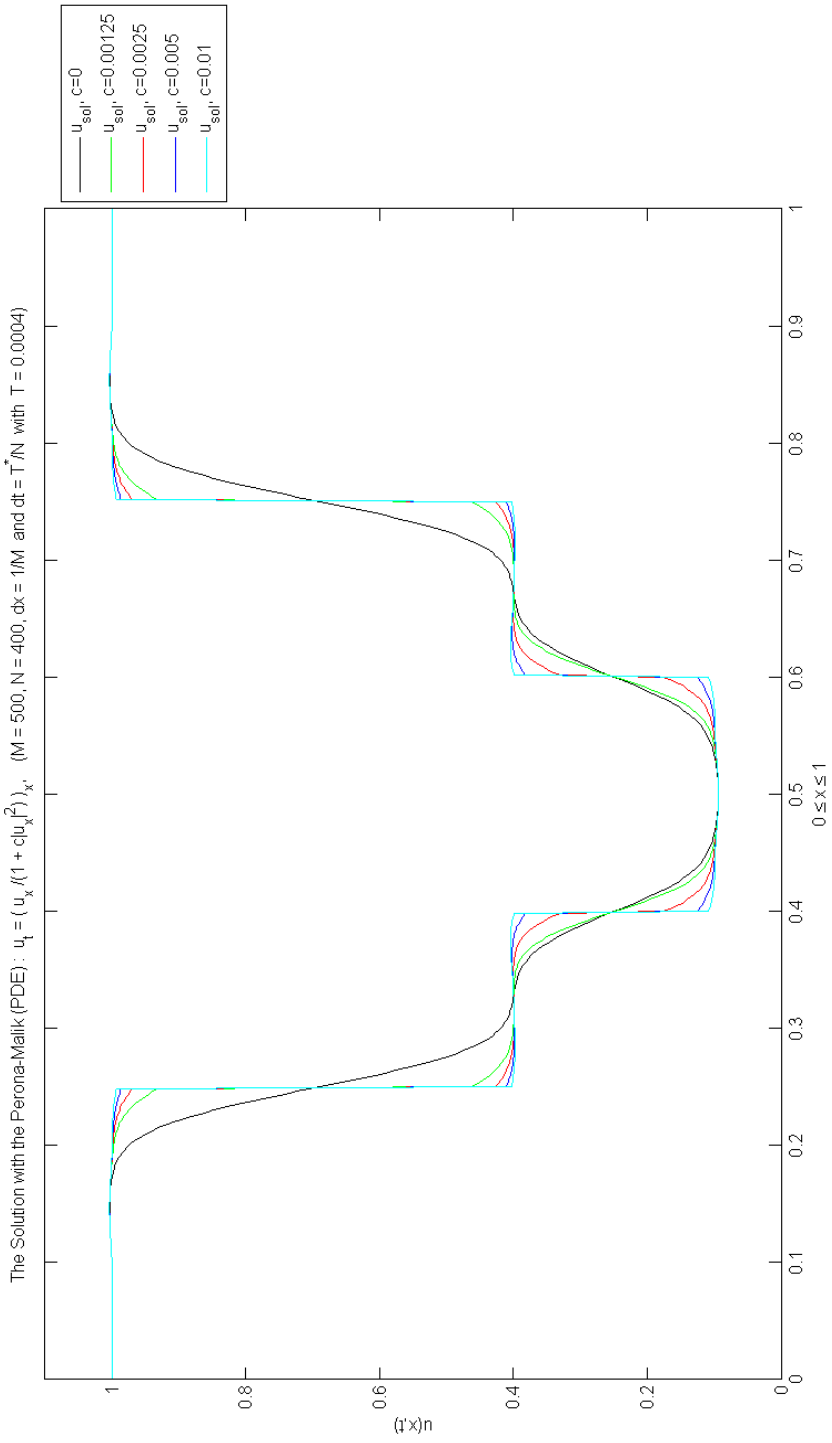


Figure 4: Solving the PDE with the Perona-Malik Equation for $M = 500$, $N = 400$, $\Delta x = \frac{1}{500} = 2 \times 10^{-3}$, $\Delta t = \frac{0.0004}{400} = 10^{-6}$, so we have that $\frac{\Delta t}{(\Delta x)^2} = \frac{1}{4}$ and use different values for $c = \epsilon \in \{0, \frac{125}{10000}, \frac{25}{1000}, \frac{5}{1000}, \frac{1}{100}\}$. $T = stop \times T^*$, with $stop = 1$ and $T^* = 0.0004$. Precisely 400 iteration steps are made.

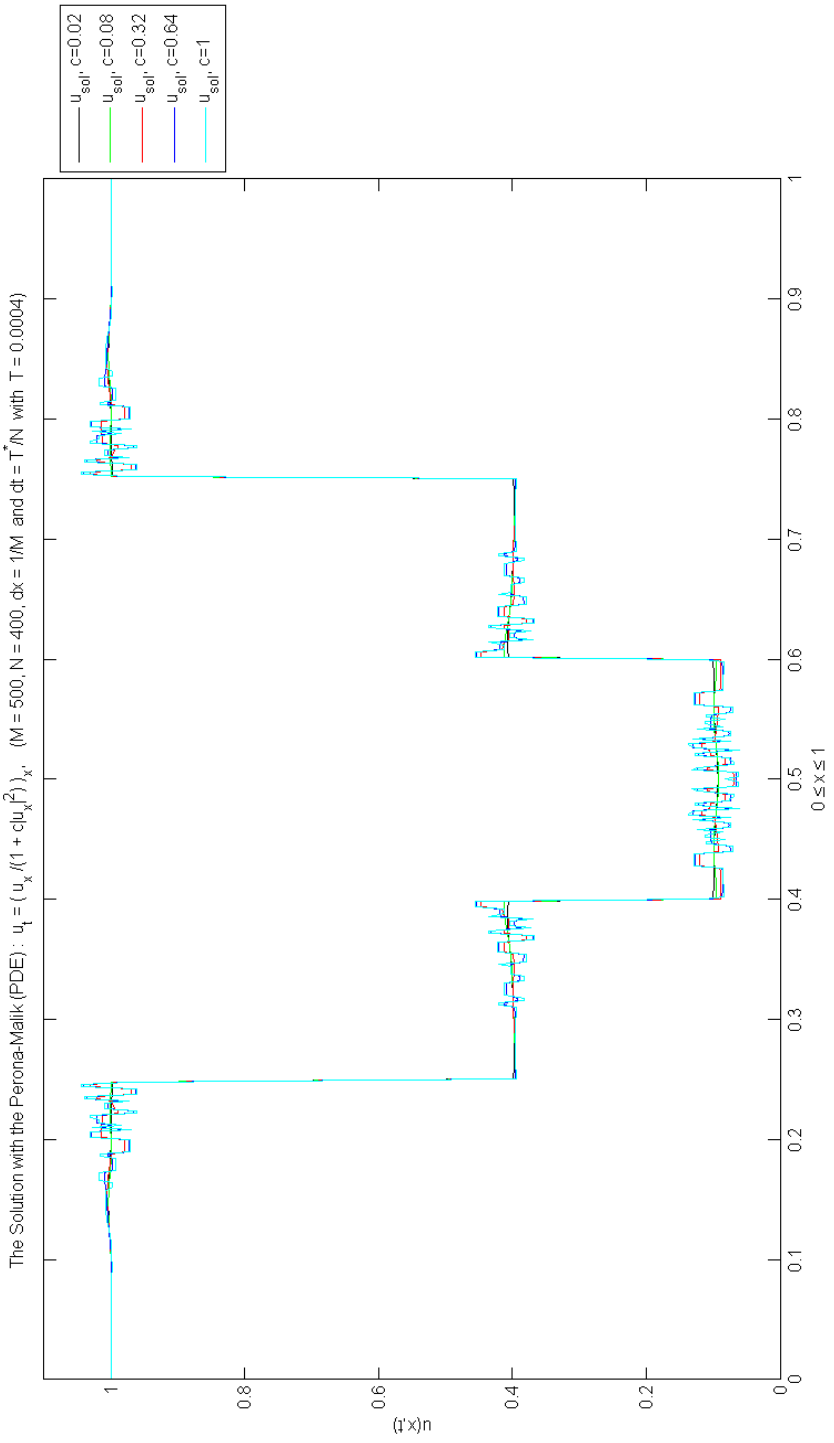


Figure 5: Solving the PDE with the Perona-Malik Equation for $M = 500$, $N = 400$, $\Delta x = \frac{1}{500} = 2 \times 10^{-3}$, $\Delta t = \frac{0.0004}{400} = 10^{-6}$, so we have that $\frac{\Delta t}{(\Delta x)^2} = \frac{1}{4}$ and use different values for $c = \epsilon \in \{\frac{2}{100}, \frac{8}{100}, \frac{32}{100}, \frac{64}{100}, 1\}$. $T = stop \times T^*$, with $stop = 1$ and $T^* = 0.0004$. Here also precisely 400 iteration steps are made.

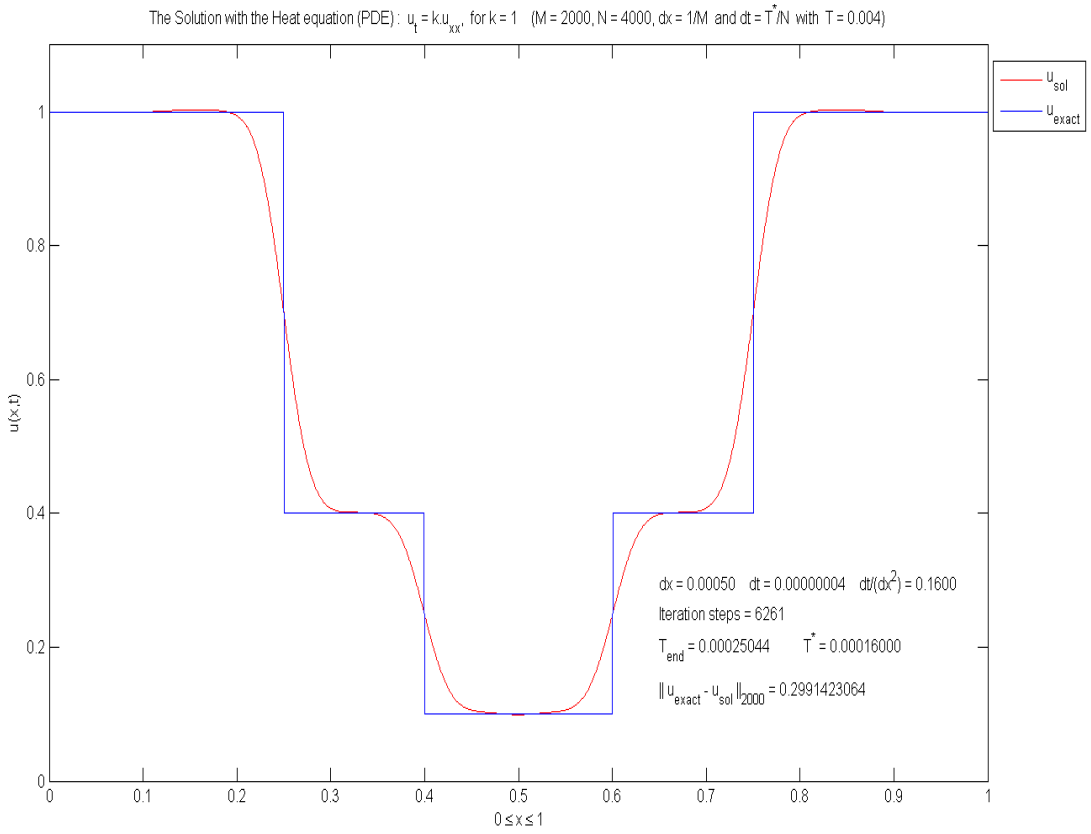


Figure 6: Solving the PDE with the Heat Equation, where we use the initial condition $u_0^n = u^*(x,t)$, with $\Delta x = \frac{1}{2000} = 5 \times 10^{-4}$, $\Delta t = \frac{0.00016}{4000} = 4 \times 10^{-8} \Rightarrow \frac{\Delta t}{(\Delta x)^2} = \frac{4}{25} = 0.16$ and $k = 1$. $T_{max} = T = 25T^*$, with $T^* = 0.00016$. With a built in variable *stop* we can expand the end-time T, which makes further iteration possible if the stopping criterion is not yet reached. Notice that the number of iteration steps needed is 6261; this is larger then $N = 4000$. We have that $stop = 25$ and the elapsed time is approximately 4 seconds.

Using a column of a (digital) image as a signal: S_2

First we will read in an image and store this as a matrix. Then we will take the K-th column from that matrix and consider this as a signal. The gray-scaled image in Figure 8(b) will be used for this purpose. In Figure 9 we have a signal, where $K = 17$ (see the red line in Figure 10), and apply the Heat and Perona-Malik PDE to it. Here we also see that the Perona-Malik PDE gives better results compared to the Heat PDE.

With the results above we hope to have put a step in the right direction by moving from the 1-D to the 2-D situation. This means that we will investigate denoising noisy images, instead of noisy signals.

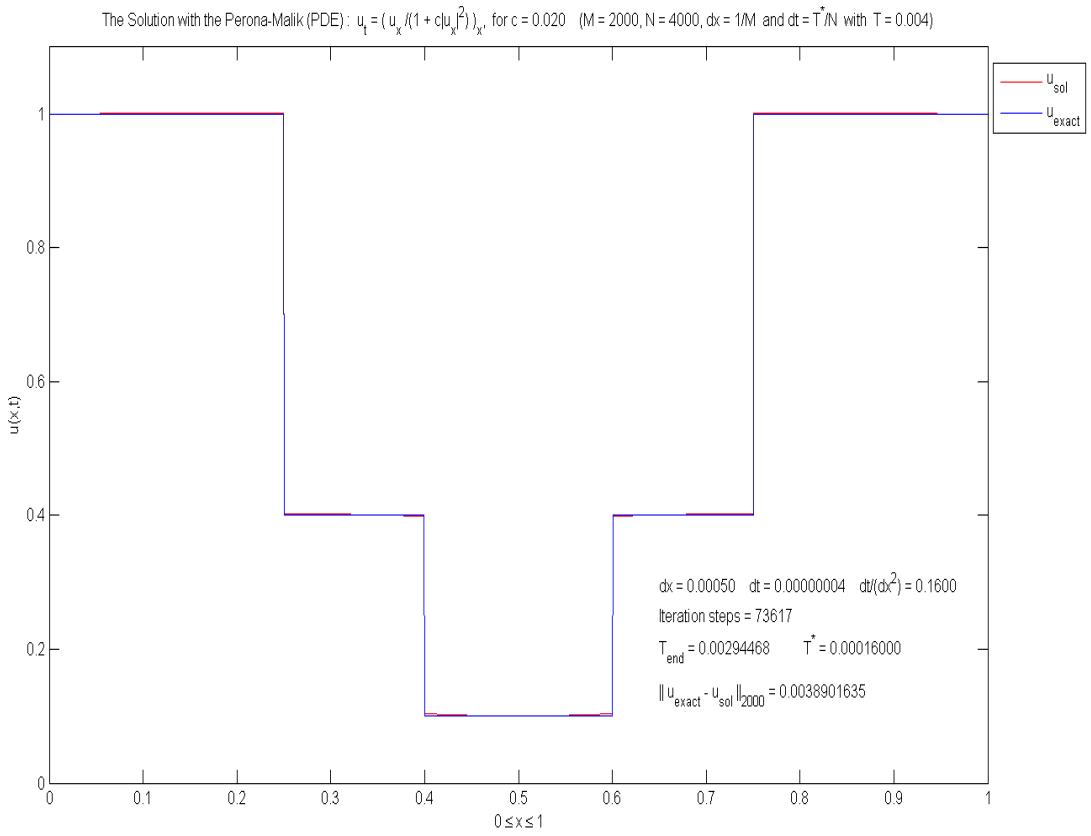
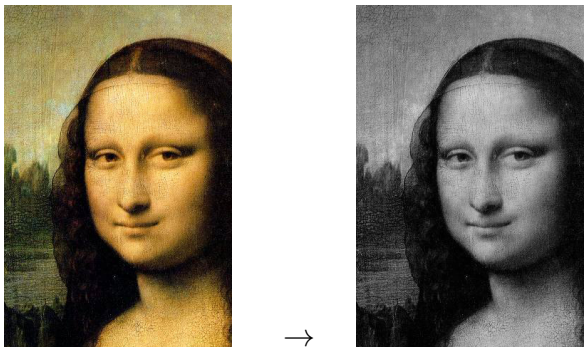


Figure 7: Solving the PDE with the Perona-Malik Equation, where the initial condition is $u_0^n = u^*(x,t)$, with $\Delta x = \frac{1}{2000} = 5 \times 10^{-4}$, $\Delta t = \frac{0.00016}{4000} = 4 \times 10^{-8} \Rightarrow \frac{\Delta t}{(\Delta x)^2} = \frac{4}{25} = 0.16$ and $c = \epsilon = 0.02$. $T_{max} = T = 25T^*$, with $T^* = 0.00016$. A built in variable *stop* expands the end-time T. Notice that the number of iteration steps needed is 73617; this is much larger then $N = 4000$. We have that $stop = 25$ and the elapsed time is approximately 3800 seconds. The stopping criterion used for this approximation is $\|u_j^n - u_{j-1}^n\|_2 \leq (\Delta x)^{1.8}$.



(a) (b)

Figure 8: Mona Lisa (a) A (digital) color image in the Red-Green-Blue (RGB) color space . (b) Converted version of the RGB Mona Lisa image to shades of gray. We calculated the gray scale of each pixel via Y=0.30R+0.59G+0.11B.

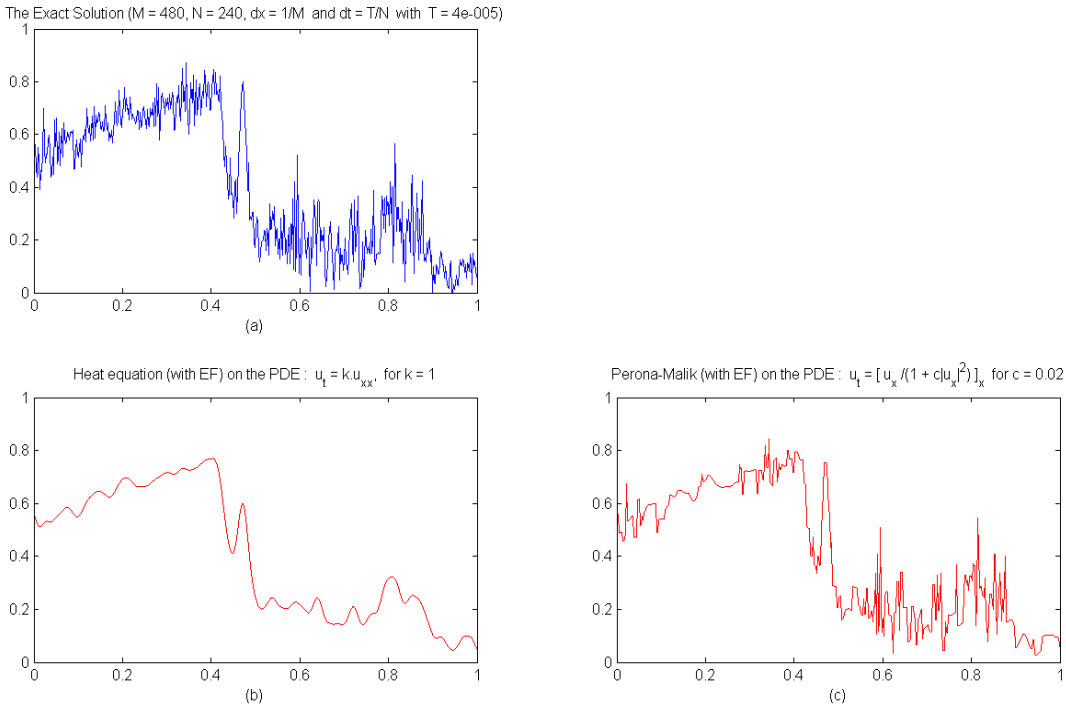


Figure 9: **(a)** Considering a column of an image as a signal (pde) with noise. **(b)** Applying the Heat equation on the signal in (a), using Euler-Forward. **(c)** Applying the Perona-Malik equation on the signal in (a), using Euler-Forward. **Parameters**: $k = 1$, $M = N = 24$, $T = 0,00004$ and $\epsilon = c = 0.02$, so $\Delta x = \frac{1}{M}$, $\Delta t = \frac{T}{N}$ and $\frac{\Delta t}{(\Delta x)^2} = \frac{48}{125} = 0.384$.

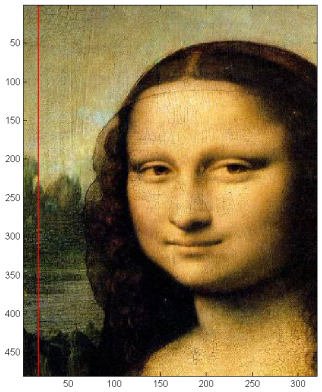


Figure 10: The red line is the 17-th Column of the (color) Mona Lisa image.

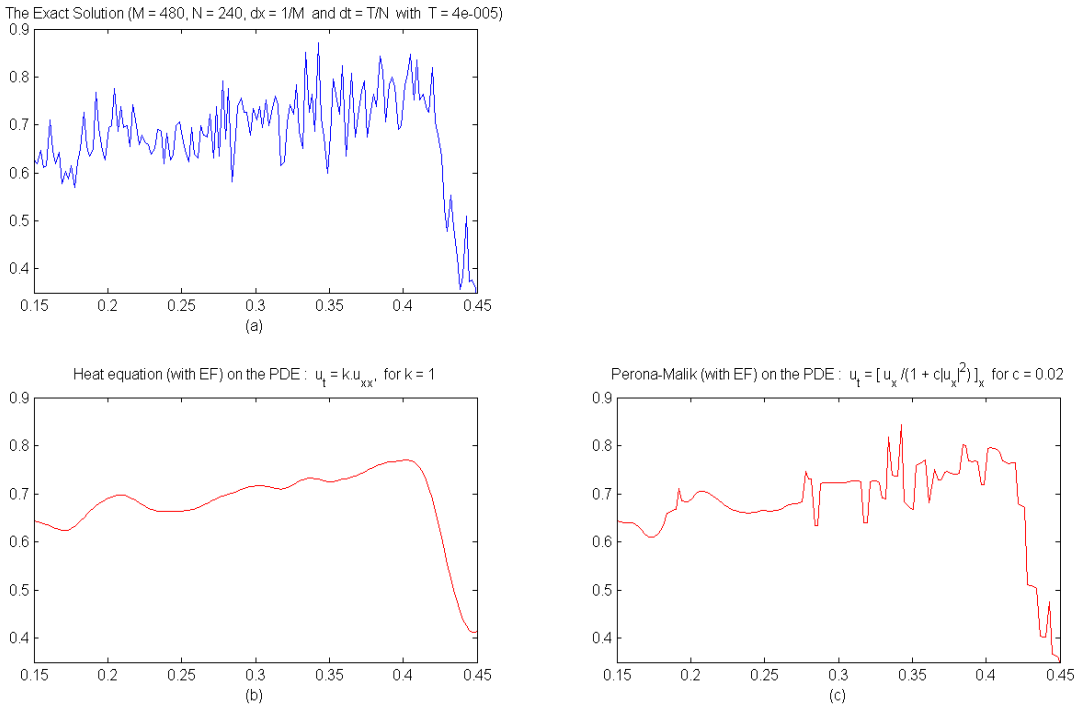


Figure 11: Zooming in on figure 9, this on the area $[0.15\ ,\ 0.45] \times [0.35\ ,\ 0.90]$.

2.4 Conclusion and Future Work

When we want to denoise a noisy signal, as a form of Digital Signal Processing (DSP), it is possible to use another numerical method such as the Implicit method or the Crank-Nicolson (CN) method, instead of the Explicit Euler, to solve the Heat Equation (HE). We wanted to try this and see if the results are better, compared to the Explicit method explained above. Recall that the HE will smoothen the signal quite rapidly, so detailed information might get lost, even when a better numerical method is used to solve the Heat Equation. This means for example that the Implicit Euler (EB), compared to the Explicit Euler (EF), is better when EF needs to compute much longer, in terms of time steps. Using a non-uniform discretization of the x-direction is desired especially when there are discontinuities involved. However when using a more stable numerical method compared to the EF, we might not gain speedups, in terms of computational cost for the whole denoising process.

The Perona-Malik Equation (PME) seems to be a better PDE for the denoising procedure within DSP, compared to the HE. When using the PME we need to choose a proper ϵ. If $\epsilon = 0$, we have that PME = HE. For $\epsilon > 0.25$ the denoising process is too slow or nothing seems to happen, in the *beginning* of this procedure. Iterating long enough will smoothen the noisy signal eventually see for example Figure 12. PME works like a local averaging filter, which localizes the discontinuities[7], and leaves them unchanged. But a locally fast alternating signal will be smoothened a little. This smoothening process does not happen as fast as what the HE would do to that (noisy) signal.

To handle fast alternating signals or signals with discontinuities, it would be better if we do not choose equally large step sizes Δx. This means that we need to take care of stability issues, when using EF, in each time-step. This since Δx might change in each time-step so that locally $\frac{\Delta t}{(\Delta x)^2} > \frac{1}{2}$, meaning that we also need to change Δt. To avoid this we can use EB, CN or some other unconditionally stable Finite Difference method which is consistent with the HE and PME.

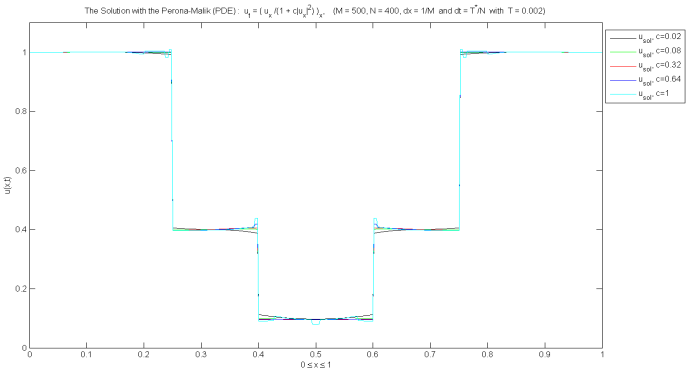


Figure 12: Set $stop = 5$ and the other parameters as the example in Figure 5.

[7] these discontinuities in the signal may be due to noise, but may also be the result of transition to locally detailed information

3 Image Processing via the PDE approach

In this chapter we will do some (digital) image processing. With this we mean image restoration in the form of noise reduction. The image analysis and decompression process will not be discussed in this report. This means that we will investigate the two-dimensional (2-D) version of what is discussed in chapter 2.

All images during the denoising process will be gray-scale[8] images, where each value of the pixel is the result of converting a color image, in the RGB colorspace; so this pixel value is the weighted sum

$$Y = 0.30R + 0.59G + 0.11B, \tag{17}$$

where R,G,B are respectively the red, green and blue pixel component from the color image to be processed. In the gray-scale image the value *zero* represents black, and the value *one* represents white.

Adding noise to an image

A brief description of the noise in the noisy image will be given. We will use the following kind of (random) noise [L01 (f)]:

- plain random noise;
  We set a value[9] σ and then for each pixel we randomly choose a value β; so on *pixel level* we have that
  $$Y_{noisy} = Y + \sigma\beta, \tag{18}$$
  or
  $$Y_{noisy} = Y - \sigma\beta, \tag{19}$$
  with $0 < \sigma < 1$ and $0 \leq \beta \leq 1$, depending[10] on the image to be investigated. For instance; if the background of the *test image* is black, we will use (18) and if the background is white we will use (19).

- salt-and-pepper noise;
  This can be recognized by the randomly occurring white and black pixels.

- white Gaussian noise;
  Adds Gaussian white noise of mean m and variance v to the *test image*. In all test cases we used $m = 0.0$, but we will vary the value of the parameter v. This type of noise will be used the most during the denoising procedures in this report, because of the uncorrelated property and is also commonly used in applications [L01 (g)].

More details on all of these type of noise can be found in [L01], and will not be discussed, since our main focus is denoising. And in practice we will need to denoise a noisy image,

[8] grayscale (or grey-scale) images are distinct from black-and-white images [L01 (e)]

[9] for most images we used $\sigma = 0.2345678901$ or $\sigma = 0.4$

[10] this is not a requirement, but only for better studying purpose of the applied denoising method

not knowing what the original image is. We can also say; not knowing what the added noise is.

There are more kinds of noise, such as *Quantization noise (uniform noise)* [L01], *Pop noise*, *Localized random noise* and *Random clutter noise* (*recognizable by unwanted dots and horizontal bars with maximum intensity*) [B05]. None of these four will be discussed in this report.

Removing noise from an image
One of the most important aspects in (digital) signal processing is denoising noisy signals [B05]. There are several kinds of noise removing techniques such as: *linear smoothing filters* (convolving the original image with a mask that represents a low-pass filter) [L01][B05], *non linear filters* (example: the median filter) [B04][L01], and *(An)isotropic diffusion* [B04][P01][P02][L01]. Only the latter will be discussed.

The Peak signal-to-noise ratio
The Peak signal-to-noise ratio (PSNR) is commonly used for measurement of the quality of reconstruction of lossy compression [L01 (c)]. With the PSNR we will try to give the quality of the denoised image [P03] compared to the *noisy image* or compared to the *original image*. The PSNR is defined via the mean squared error (MSE) which is given by

$$MSE = \frac{1}{M\ N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left[I(i,j) - K(i,j) \right]^2 , \tag{20}$$

where I and K are images and M, N are the size of the image respectively, in horizontal and in vertical direction. We will consider K the noisy approximation[11] of I. Let us define MAX_I as the maximum possible pixel value of the image. So in our case we have $MAX_I = 1$. With this the PSNR is defined as

$$\begin{aligned} PSNR &= 10. \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20. \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\ &= 20. \log_{10} \left(\frac{1}{\sqrt{\frac{1}{M\ N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left[I(i,j) - K(i,j) \right]^2}} \right) \end{aligned} \tag{21}$$

With the PSNR we can also see if further denoising is necessary or even required to gain a reasonable result for the quality of the denoised image. The comparison then should be done with the noisy image, instead of the original images, since in practice we do not have the noisy image.

We could also build in the PSNR into the denoising algorithm as a stopping criterion. This is not done in the algorithm we used. In section § 3.3 we can see what the PSNR can tell us about the convergence.

[11] notice; we need not to know which image is the noisy one since $(I-K)^2 = (K-I)^2$, for $I, K \in [-1, 1]$

3.1 The Heat Equation: $u_t = \Delta u$

In section §2.1 we saw that smoothening, of a locally fast alternating signal, with the Heat Equation (HE) occurs when solving a Partial Differential Equation (PDE). A well known effect for the 2-D situation is similar to the 1-D case. In 1-D we had smoothening and diffusion of a signal, now we have smoothening of the surface with a diffusion effect. Therefore the HE is also known as the diffusion equation [B02][L01 (h)].

We want to analyse what the effect is of the HE on a noisy image. Now we need to solve the 2-D initial boundary value problem and we will leave the boundary of the image during the denoising procedure unchanged. One may choose another approach for handling the boundaries during this process. The effect of this choice is visible in the examples in section §3.3.

2-D discretization of the Heat PDE

Here we will consider the initial boundary value problem for the HE on the domain $\Omega \subset \mathbb{R}^n$, with $n = 2$; the 2-D Heat equation

$$\begin{cases} u_t = k\Delta u = k\left(\dfrac{\partial^2 u}{\partial x^2} + \dfrac{\partial^2 u}{\partial y^2}\right), \\ \\ u(x, y, 0) = \Theta(x, y), \end{cases} \tag{22}$$

where $\Theta(x, y)$ is the noisy image, and $k = 1$. The finite difference method will be applied for approximation of (22) this by applying the Explicit Euler (EF) scheme. These approximations can be found in the appendix from (58)-(62). This gives us the approximation

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{\Delta t}{(\Delta x)^2}\Big(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\Big) + \frac{\Delta t}{(\Delta y)^2}\Big(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\Big) \tag{23}$$

For efficiency purposes when programming, we may set $r = \dfrac{\Delta t}{(\Delta x)^2}$ and $s = \dfrac{\Delta t}{(\Delta y)^2}$. Thus we can rewrite (23) as

$$u_{i,j}^{n+1} = (1 - 2r - 2s)u_{i,j}^n + r(u_{i+1,j}^n + u_{i-1,j}^n) + s(u_{i,j+1}^n + u_{i,j-1}^n) \tag{24}$$

Numerical Stability

As in the 1-D case we need to check the stability of HE using EF for the 2-D situation. The stability condition now is that both r and s should be less than $\frac{1}{2}$. On a uniform grid where $\Delta x = \Delta y$ we have that $\lambda := max(r, s)$, this way we have that the scheme is conditionally stable with

$$r + s \leq \frac{1}{2} \qquad \Rightarrow \qquad \lambda \leq \frac{1}{4}, \tag{25}$$

which is stricter than the one-dimensional case [B01].

3.2 The Perona-Malik Equation: $u_t = \nabla.(D(u)\nabla u)$

In this section we will investigate[12] the two-dimensional (2-D) version of the, in the previous chapter discussed, Perona-Malik Equation (PME). Here scale-space filtering, in terms of multi-scale descriptions of images, will have a important role. I quote; Perona and Malik [P02]: *"The essential idea of this approach is quite simple: embed the original image in a family of derived images $I(x, y, t)$ obtained by convolving the original image $I_0(x, y)$ with a Gaussian kernel G(x,y;t) of variance t:*

$$I(x, y, t) = I_0(x, y) * G(x, y; t). \tag{26}$$

Larger values of t, the scale-space parameter, correspond to images at coarser resolutions."

The used Gaussian kernel has the form [L01 (i)]:

$$\frac{1}{2\pi t} e^{-(x^2+y^2)/2t},$$

and the derived images $I(x, y, t)$ may be viewed as solutions of (22), the Heat diffusion equation, with

$$I_0(x, y) = u(x, y, 0) = \Theta(x, y).$$

Some important features are that there should not be any generation of spurious detailed information, when we diminish the resolution. That blurring is required to be space invariant. These criteria are respectively known as:

- *Causality*
- *Homogeneity and Isotropy*

Since the Gaussian kernel does not *respect* natural boundaries (edges) of objects, spatial distortion occurs when using this kernel. Thats why Perona and Malik enunciated the following criteria:

- *Causality*
- *Immediate Localization*
- *Piecewise Smoothing*

More details about this can be found in [P02]. Perona and Malik showed that a suitable choice for the constant k in (22), enables us to satisfy the last two conditions in the criteria above, this without giving up the first criterion; causality. The choice is; not to use a diffusion coefficient k as a constant, but a diffusion coefficient as function $k = k(x, y, t)$. Perona and Malik suggested the *anisotropic diffusion equation*[13] instead of the

[12] most of the text in this section is based on [P02]

[13] this is also known as the Perona-Malik equation. In [P08] they note that the PME should not be named anisotropic *in their terminology*, but isotropic

isotropic heat diffusion equation, where the subset of the plane is denoted as $\Omega \subset \mathbb{R}^2$ and $I(x, y, t) : \Omega \to \mathbb{R}$ is the family of gray-scaled images corresponding with $I_0(x, y)$. Thus the anisotropic diffusion equation is defined as

$$I_t = div\Big(k(x, y, t)\ \nabla I\Big) = k(x, y, t)\Delta I + \nabla k \cdot \nabla I, \tag{27}$$

where $I_t = \frac{\partial}{\partial t} I$ further div, ∇ and Δ respectively indicate the divergence, gradient and Laplacian operator, with respect to the space variables. Perona and Malik showed that the simplest estimate of the position of edges is the gradient of the brightness function

$$E(x, y, t) = \nabla I(x, y, t), \tag{28}$$

which gives excellent results, and that the conduction coefficient k can be chosen as

$$k = k(x, y, t) = g(\|E\|) = g(\|\nabla I\|), \tag{29}$$

where $g(\cdot)$ is a nonnegative monotonically decreasing function and $g(0) = 1$. The effect therefore is that diffusion occurs in the interior of regions and does not affect the edges of these regions. Perona and Malik also showed that in case of using edge estimate $E(x, y, t)$ there is a restricted choice, this will not be done here.

Perona and Malik proposed two functions for the diffusion coefficient

$$g(\|\nabla I\|) = e^{-(\|\nabla I\|/K)^2} \tag{30}$$

and

$$g(\|\nabla I\|) = \frac{1}{1 + \left(\frac{\|\nabla I\|}{K}\right)^{1+\alpha}} \qquad \alpha > 0, \tag{31}$$

where $\alpha = 1$ in [P02][P04][L01(b)]. We will use (31) for our numerical experiments and also set $\alpha = 1$, but we have that the constant $K = \dfrac{1}{\sqrt{\epsilon}}$. This way we will use

$$g(\|\nabla I\|) = \frac{1}{1 + \epsilon\|\nabla I\|^2}. \tag{32}$$

The constant ϵ, meaning K, can be fixed at some value and mostly $K \in \{1, 2, 3, \cdots\}$, but this is not a requirement. Notice that for $K \to \infty$ we have $\epsilon \to 0$ and for $\epsilon = 0$ we have the Heat PDE.

2-D discretization of the Perona-Malik PDE

Now we will investigate the 2-D Perona-Malik Equation (PME), which is of the form

$$u_t = div\Big(D(u)\nabla u\Big) = \nabla.(D(u)\nabla u), \tag{33}$$

so $D(u) = g(\|\nabla u\|)$ and u represents the image. In the previous section we have already given the discretization of u_t, so our focus will be on the right-hand side of (33).

$$u_t = \nabla \cdot \Big(\frac{1}{1+\epsilon|\nabla u|^2} \nabla u \Big) = \frac{\partial}{\partial x} \left[\frac{1}{1+\epsilon|\nabla u|^2} \frac{\partial u}{\partial x} \right] \Big|_{i,j} + \frac{\partial}{\partial y} \left[\frac{1}{1+\epsilon|\nabla u|^2} \frac{\partial u}{\partial y} \right] \Big|_{i,j}. \quad (34)$$

where

$$|\nabla u|^2 = \nabla u \cdot \nabla u = \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2. \quad (35)$$

We will now approximate both equations. Using the approximations in the appendix. We have that

$$\frac{\partial}{\partial x} \left[\frac{1}{1+\epsilon|\nabla u|^2} \frac{\partial u}{\partial x} \right] \Big|_{i,j} = \quad \rightarrow \quad \text{see Appendix : § A,} \quad (36)$$

and

$$\frac{\partial}{\partial y} \left[\frac{1}{1+\epsilon|\nabla u|^2} \frac{\partial u}{\partial y} \right] \Big|_{i,j} = \quad \rightarrow \quad \text{see Appendix : § A.} \quad (37)$$

This way we find that the FD-scheme for the Perona-Malik PDE becomes

$$u_{i,j}^{n+1} = u_{i,j}^n + \Delta t \Big((36) + (37) \Big). \quad (38)$$

3.3 Results

In this section we will put the Heat Equation (HE) and the Perona-Malik Equation (PME) to the denoising test, for different kind of images. There will also be different types of noise added to the original images, so we can analyze what the denoising effect of these two PDEs is on the different types of noisy images.

Stopping criteria for the Heat and Perona-Malik PDE

We did not build in a stopping criterion in the algorithm yet[14], but the Peak Signal-to-Noise Ratio (PSNR) can be used to abort iteration in an early state if there is no divergence. So for now our stopping criteria is a fixed number of iteration steps.

Experiments

We will investigate two kind of images (*simple* and *less simple*) and put the HE and the PME to the denoising test. The original *simple* image has only three grayshades and has only vertical and horizontal edges. The *less simple* images have more grayshades and all kind of edges. The errors in the Figures 14, 16 and 18 are the norm[15]: $\|u_{i,j}^{n+1} - u_{i,j}^n\|$ given in terms of the largest singular value.

[14] see the experiments under item Salt-and-Pepper noise for an example of the use of this possible criteria

[15] using the function norm(A) or norm(A,2) in MATLAB, where A is a matrix representing $u_{i,j}$

A. We will look at the following *simple*[16] test cases:

- Gaussian noise
  During the denoising procedures the boundary of the images itself is left unchanged. The denoising process is on an image where Gaussian noise is added, with mean $m = 0.0$:

  - We set variance $v = 0.1$, and we did only 500 steps with PM.
    In Figure 13 the result of the denoising process is visible with the PME. Further iteration is preferred here to remove the remaining *dark dots*. Figure 14, shows that removing these remaining *dark dots* will be a very slow process and therefore a lot of iteration steps are needed.

  - Here we also set variance $v = 0.1$, but now 2500 steps are done with PM.
    As we saw in the experiment above, further iteration was needed. In Figure 15 we see that all the *dark dots* are gone, but Figure 16 tells us that this effect was already reached after approximately 1000 steps. After 1500 steps there is almost no progress made.

  - Here we set variance $v = 0.15$, but now 1500 steps are done with PM.
    The result can be seen in Figure 17. We set $\epsilon = 0.08$ instead of the 0.16, what was the case in the experiments above. Because of this choice of ϵ we have faster convergence, see Figure 18. The now larger parameter v introduced more noise, such that the edges of the internal object (rectangle)
    is deformed.

- Plain random noise
  Here we add some plain random noise, and will see that after a few steps the HE gives bad results, see the Figure 19. Even when we can already guess what the original image should look like. For the same case the PME gives much better results. The smoothening effect of the HE causes the resulting denoised image to seem blurry, but the PME also keeps the edges of the internal rectangle *almost intact*.

- Salt-and-Pepper noise
  If this type of noise is involved, the PME is not a suitable denoising PDE for image processing. This since clusters of white or black pixels easily arise, which cannot be filtered by the PME. In Figure 20 we see this effect, where almost 30% of the pixels is changed in black or white. Here we also used $\epsilon = 0.02$ for the PME. In Figure 21 we see that the PSNR will work as a stopping criterion but the result may be not what we hoped. A good filter (image operator) when this kind of noise is involved is the so called *"Extrema Killer"* [P01]; I quote: "*This image operator simply removes all connected components of upper and lower level sets with area smaller than some fixed scale. This is not a PDE, actually it's much simpler!*".

[16] only vertical and horizontal edges are involved in the rectangles

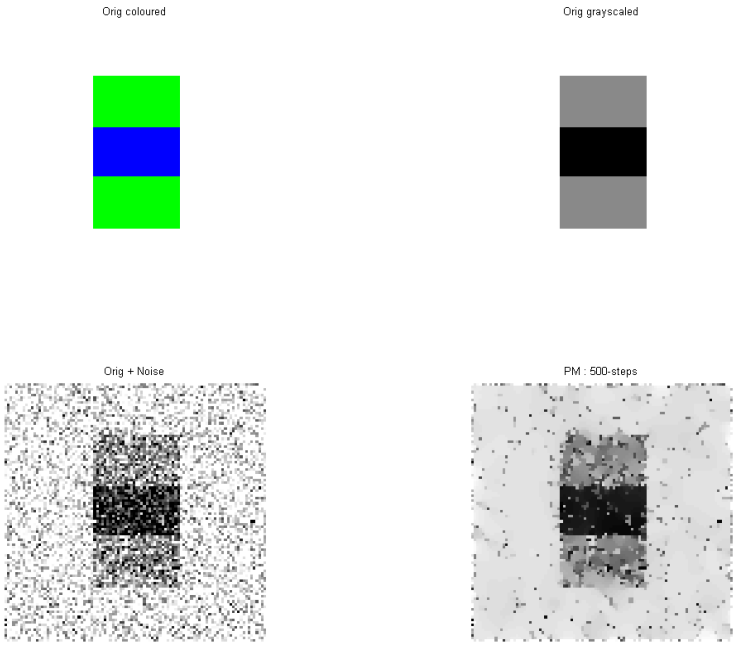


Figure 13: The left and right top figures are the original image, in RGB and gray-scaled respectively. The left lower image is the original gray-scaled image with Gaussian noise; the parameters are $m = 0.0$ and $v = 0.1$. The lower right image is the denoising result, after 500 steps, with the Perona-Malik PDE.

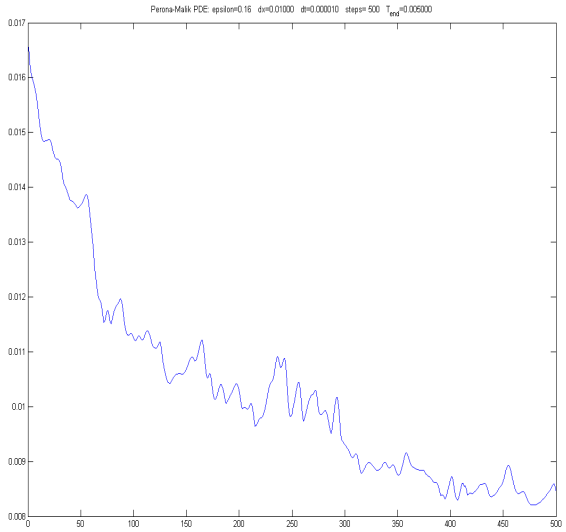


Figure 14: Here we see the error of the denoising procedure in Figure 13, with $\epsilon = 0.16$.

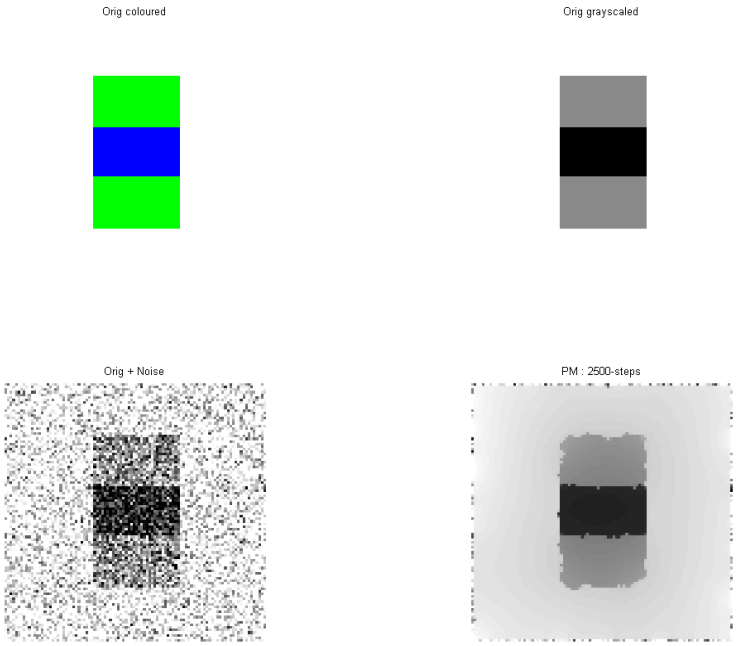


Figure 15: The left and right top figures are the original image, in RGB and gray-scaled respectively. The left lower image is the original gray-scaled image with Gaussian noise; the parameters are $m = 0.0$ and $v = 0.1$. The lower right image is the denoising result, after 2500 steps, with the Perona-Malik PDE.

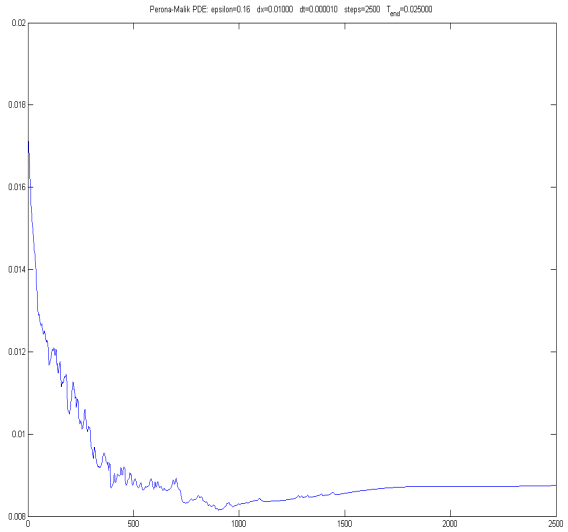


Figure 16: Here we see the error of the denoising procedure in Figure 15, with $\epsilon = 0.16$.

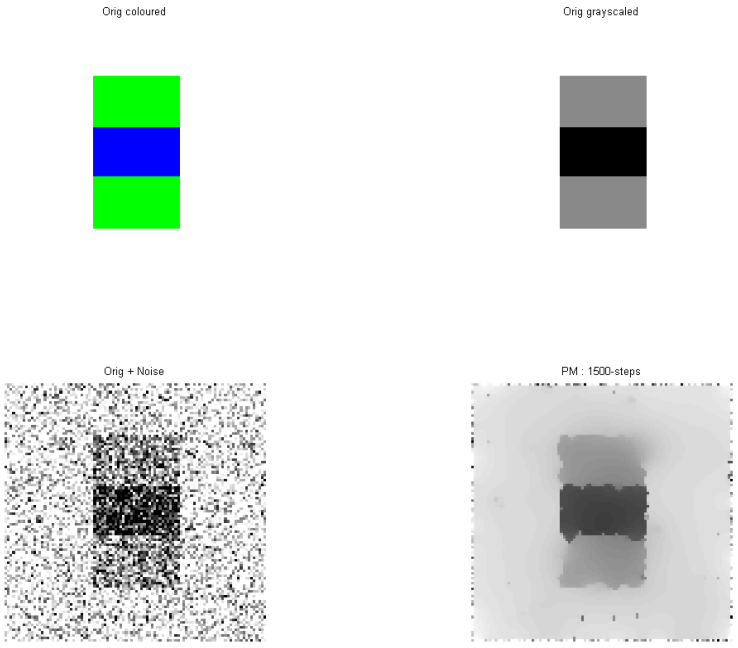


Figure 17: The left and right top figures are the original image, in RGB and gray-scaled respectively. The left lower image is the original gray-scaled image with Gaussian noise; the parameters are: $m = 0.0$ and $v = 0.15$. The lower right image is the denoising result, after 1500 steps, with the Perona-Malik PDE.

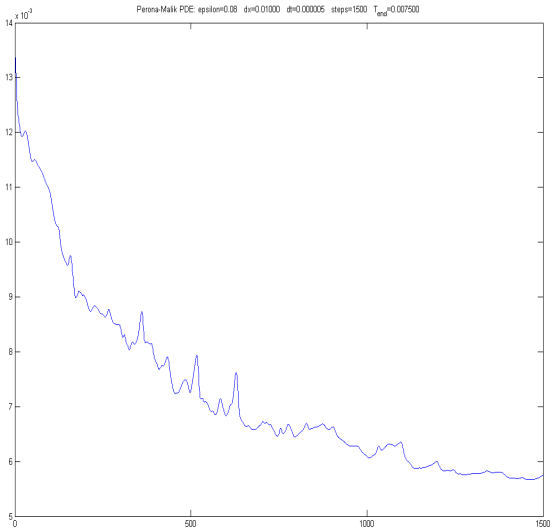


Figure 18: Here we see the error of the denoising procedure in Figure 17, with $\epsilon = 0.08$.

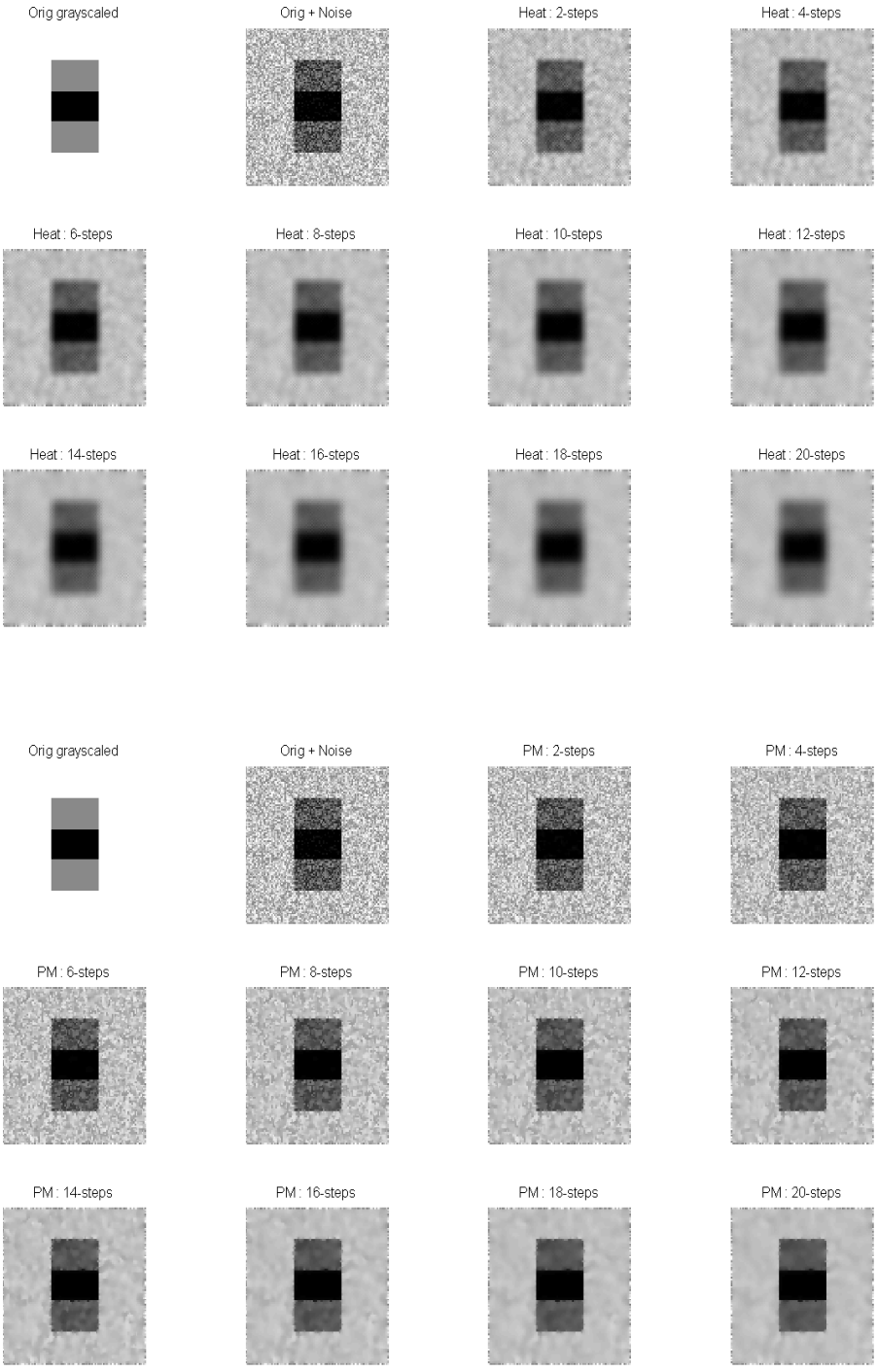


Figure 19: The first 12 images (top) represent the denoising procedure with the HE, and the following 12 images (below) the denoising procedure with the PME, where $\epsilon = 0.02$. The denoising result are after 2-20 steps respectively.

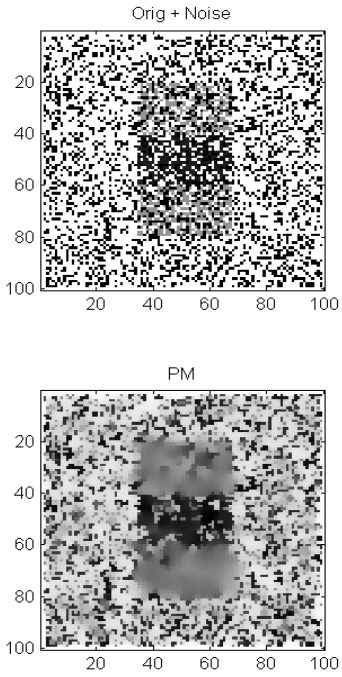


Figure 20: (Top): Original image with Salt-and-Pepper noise, where almost 30% of the pixels is turned black or white. (Lower): The denoised image after 500 iteration steps with the Perona-Malik equation.

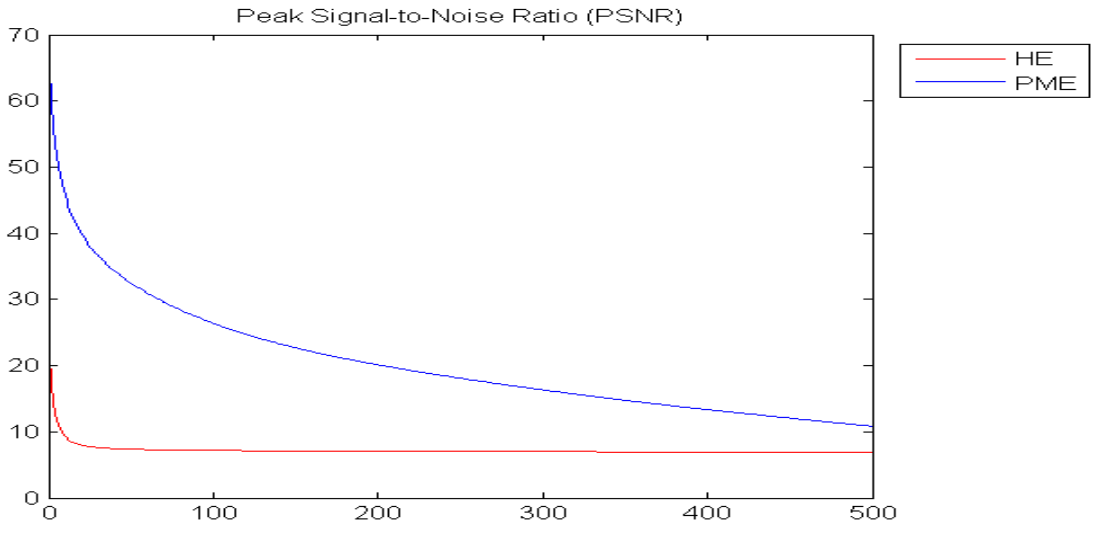


Figure 21: The PSNR curve for the Heat and Perona-Malik equations. We may conclude that after 10 iteration steps the HE for sure won't give better results. But the PME give some hope. The best it can do is visible in Figure 20.

B. The following images are less simple test cases.

- **Mona Lisa**; with Gaussian white noise
  Notice that the original Mona Lisa image has a lot of *noise-like* pixels. This was maybe the intention of the painter. That's why we do not add a lot of noise to this test image; parameter $m = 0.0$ and $v = 0.1$. The difference is therefore almost not visible; see Figure 22:

  - As we saw in the previous examples the HE will cause a blurry result as denoised image. We can also say PME with $\epsilon = 0$ causes this. This shows us that we need not take a too small ϵ which is almost zero, since the interior edges of objects such as the eyes, nose and lips are totally destroyed.

  - In Figure 22 we see that after 200 steps with PME the eyes, nose and lips are still intact. The forehead and clouds are smoothened without loosing the edges of these objects.

- **Face**; with Gaussian white noise
  After adding noise to this image we see that there is almost no visible difference between the background with noise disturbance and the contour of the face.

  - As expected the HE does not work and objects like nose and lips are localized but are totally blurred and have no logical meaning[17] at all.

  - Choosing $\epsilon = 0.02$ again we see that with the PME the eyes and lips are also localized but now we can recognize them. This is also the case in the noisy image, so this is no gain at all. Since there is no visible difference between the background with noise disturbance and the contour of the face we see that the contour almost totally disappeared after the 150 steps. Recall that the PME works as a locally averaging filter. To be able to recognize the contour we should have stopped earlier with the iteration, but then we would not lose that much noise.

    The question is; *"when should we stop iterating to gain a recognizable denoised image?"* Even the PSNR stopping criterion would not be suitable in this case. We really need to localize edges of objects in images to get better results when we have noisy images as described in this example. The more noise the less change we have in localizing and keeping these edges. Recall the causality feature of the PME; therefore we will not be able to reconstruct the edges that are lost because of the noise, since this would mean generating spurious detailed information.

    A solution to this problem is to use adaptive grids to localize edges in a early stage of the denoising process.

[17] a logical meaning is localizing the positions of the eyes and so on, but that is not what we want here

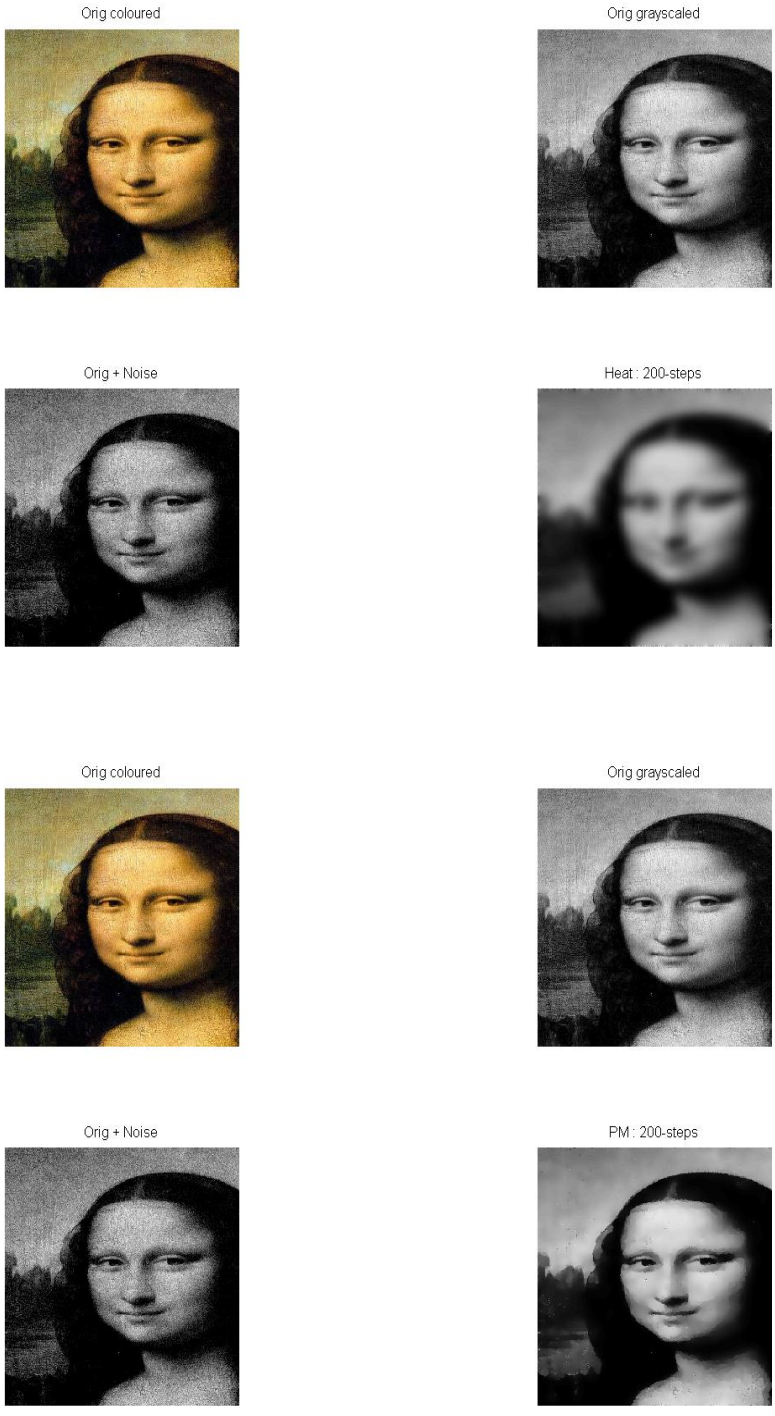


Figure 22: **Mona Lisa**: The first 4 images (top); represent the denoising procedure with the HE, and the following 4 images (below) the denoising procedure with the PME, where $\epsilon = 0.02$. In both cases we made 200 iteration steps.

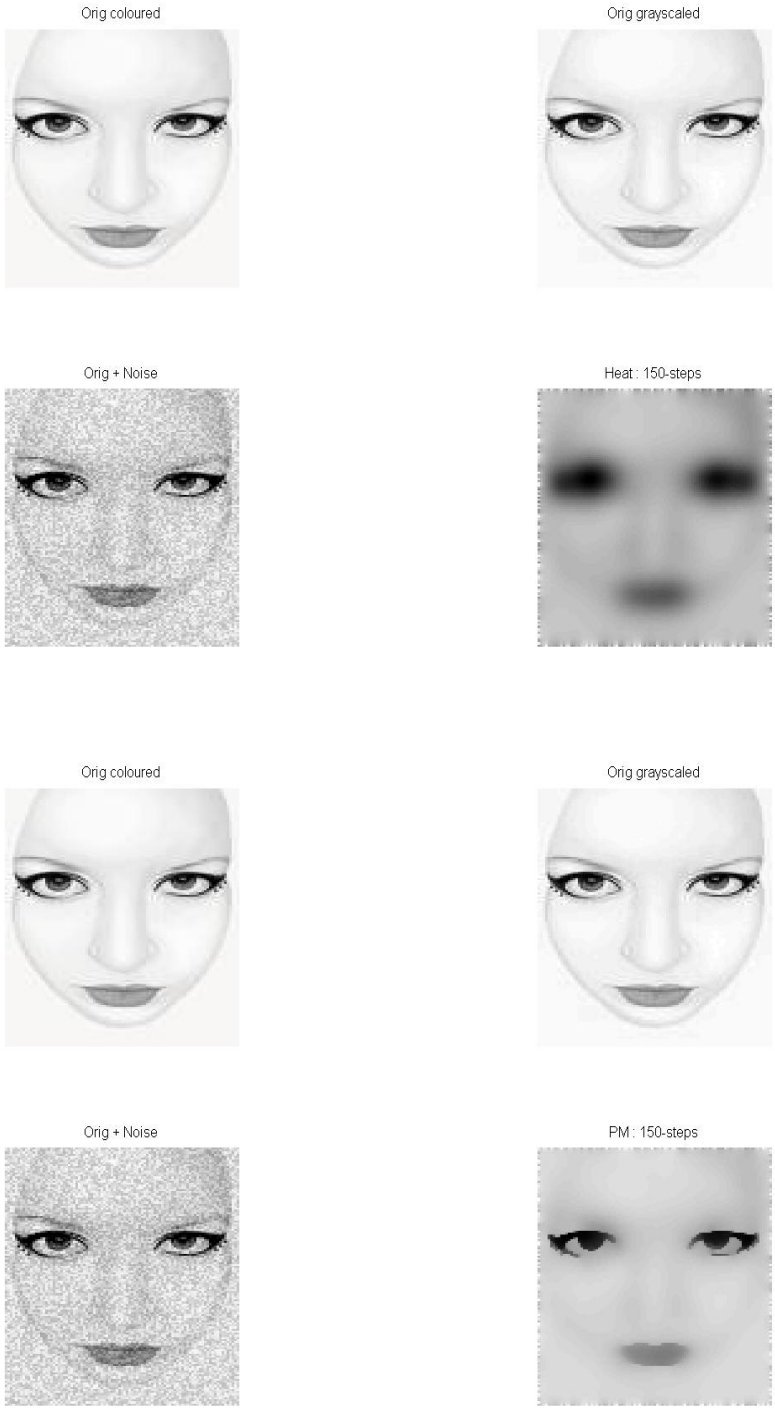


Figure 23: **Face**: The first 4 images (top); represent the denoising procedure with the HE, and the following 4 images (below) the denoising procedure with the PME, where $\epsilon = 0.02$. In both cases we made 150 iteration steps.

3.4 R-refinement in image Processing

For detecting edges in an early state of the denoising procedure, we need to switch from a uniformly distributed to a non-uniformed grid. To benefit from this approach the size of the image is of great importance. The larger the size the better adaptivity will take place. There are several ways to do refinement, using adaptive grids. In [P07] nice results are visible using the finite volume scheme for solving nonlinear diffusion equations. The approach here will be via a finite difference scheme.

First we need to perform a transformation of coordinates where $u(x, y, t)$ is transformed into $u(\xi, \eta, \theta)$, where $\xi = \xi(x, y, t)$, $\eta = \eta(x, y, t)$ and $\theta = t$. So that we can approximate

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{39}$$

Before doing this, notice that

$$\frac{\partial \theta}{\partial t} = 1 \qquad \text{and} \qquad \frac{\partial \theta}{\partial x} = \frac{\partial \theta}{\partial y} = 0,$$

as result of $\theta = t$, so

$$\begin{aligned}
u_t = \frac{\partial u}{\partial t} &= \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial t} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial t} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial t} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial t} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial t} + \frac{\partial u}{\partial \theta} = u_\xi \xi_t + u_\eta \eta_t + u_\theta \\
u_x = \frac{\partial u}{\partial x} &= \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial x} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial x} = u_\xi \xi_x + u_\eta \eta_x \\
u_y = \frac{\partial u}{\partial y} &= \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial y} + \frac{\partial u}{\partial \theta}\frac{\partial \theta}{\partial y} = \frac{\partial u}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial u}{\partial \eta}\frac{\partial \eta}{\partial y} = u_\xi \xi_y + u_\eta \eta_y
\end{aligned}$$

With these approximations we can compute the terms $u_{xx} = \frac{\partial^2 u}{\partial x^2}$ and $u_{yy} = \frac{\partial^2 u}{\partial y^2}$. This can be done by first eliminating the following terms, based on [P09]

$$\xi_t,\ \xi_x,\ \xi_y,\ \eta_t,\ \eta_x,\ \eta_y,\ \theta_t,\ \theta_x, \text{ and } \theta_y$$

in the future equation(s). This will be done by using the fact that for each matrix A for which there is an inverse A^{-1} we have that $AA^{-1} = A^{-1}A = I$, where I is the identity matrix. From the transformation we know that the Jacobian matrix and its (logical) inverse are

$$J_T^{-1} = \begin{pmatrix} \xi_x & \xi_y & \xi_t \\ \eta_x & \eta_y & \eta_t \\ 0 & 0 & 1 \end{pmatrix} \qquad \text{and} \qquad J_T = \begin{pmatrix} x_\xi & x_\eta & x_\theta \\ y_\xi & y_\eta & y_\theta \\ 0 & 0 & 1 \end{pmatrix}$$

So $J_T^{-1}J_T = I$. If we work this out we get the following system of equations (i) to (vi)

$$\begin{cases} \xi_x x_\xi + \xi_y y_\xi &= 1, & (i) \\ \eta_x x_\xi + \eta_y y_\xi &= 0, & (ii) \\ \xi_x x_\eta + \xi_y y_\eta &= 0, & (iii) \\ \eta_x x_\eta + \eta_y y_\eta &= 1, & (iv) \\ \xi_x x_\theta + \xi_y y_\theta + \xi_t &= 0, & (v) \\ \eta_x x_\theta + \eta_y y_\theta + \eta_t &= 0, & (vi) \end{cases}$$

This system can be solved, such that the each ξ, η and θ derivative is expressed in the terms of the coordinates of the physical domain. We can eliminate y_ξ and y_η in (i) and (iii) and find that

$$\left.\begin{aligned} -y_\eta\left(\xi_x x_\xi + \xi_y y_\xi\right) &= -y_\eta \\ y_\xi\left(\xi_x x_\eta + \xi_y y_\eta\right) &= 0 \end{aligned}\right\} \quad \Rightarrow \quad \xi_x = \frac{-y_\eta}{x_\eta y_\xi - x_\xi y_\eta}$$

In the same way we can eliminate x_ξ and x_η in (i) and (iii)

$$\left.\begin{aligned} -x_\eta\left(\xi_x x_\xi + \xi_y y_\xi\right) &= -x_\eta \\ x_\xi\left(\xi_x x_\eta + \xi_y y_\eta\right) &= 0 \end{aligned}\right\} \quad \Rightarrow \quad \xi_y = \frac{-x_\eta}{x_\xi y_\eta - x_\eta y_\xi}$$

Lets eliminate y_ξ and y_η in (ii) and (iv)

$$\left.\begin{aligned} -y_\eta\left(\eta_x x_\xi + \eta_y y_\xi\right) &= 0 \\ y_\xi\left(\eta_x x_\eta + \eta_y y_\eta\right) &= y_\xi \end{aligned}\right\} \quad \Rightarrow \quad \eta_x = \frac{y_\xi}{x_\eta y_\xi - x_\xi y_\eta}$$

Now we eliminate x_ξ and x_η in (ii) and (iv)

$$\left.\begin{aligned} -x_\eta\left(\eta_x x_\xi + \eta_y y_\xi\right) &= 0 \\ x_\xi\left(\eta_x x_\eta + \eta_y y_\eta\right) &= x_\xi \end{aligned}\right\} \quad \Rightarrow \quad \eta_y = \frac{x_\xi}{x_\xi y_\eta - x_\eta y_\xi}$$

Substituting the computed values above in (v) and (vi) we find that

$$\xi_t = -\frac{x_\theta y_\eta - x_\eta y_\theta}{x_\xi y_\eta - x_\eta y_\xi} \qquad \text{and} \qquad \eta_t = \frac{x_\theta y_\xi - x_\xi y_\theta}{x_\xi y_\eta - x_\eta y_\xi}$$

If we compute the determinant of the Jacobian matrix J_T we find, with for example the use of the Sarrus' rule [L01 (j)], that

$$\mathcal{J} = det(J_T) = |J_T| = x_\xi y_\eta - y_\xi x_\eta.$$

With this value we see that we can rewrite the computed terms as

$$\xi_x = \frac{y_\eta}{\mathcal{J}}, \qquad \xi_y = -\frac{x_\eta}{\mathcal{J}}, \qquad \xi_t = -\frac{x_\theta y_\eta - y_\theta x_\eta}{\mathcal{J}},$$

$$\eta_x = -\frac{y_\xi}{\mathcal{J}}, \qquad \eta_y = \frac{x_\xi}{\mathcal{J}}, \qquad \eta_t = \frac{x_\theta y_\xi - y_\theta x_\xi}{\mathcal{J}}.$$

We see that the first derivative of $u(x,y,t)$ with respect to x is:

$$\begin{aligned}\frac{\partial u}{\partial x} = u_x &= u_\xi \xi_x + u_\eta \eta_x \\ &= \frac{1}{\mathcal{J}}\Big[u_\xi y_\eta - u_\eta y_\xi\Big] \\ &= \frac{1}{\mathcal{J}}\Big[(u_\xi y_\eta + u y_{\eta\xi}) - (u_\eta y_\xi + u y_{\xi\eta})\Big] \\ &= \frac{1}{\mathcal{J}}\Big[(u y_\eta)_\xi - (u y_\xi)_\eta\Big]\end{aligned} \tag{40}$$

Notice that the cross terms $uy_{\eta\xi}$ and $uy_{\xi\eta}$ cancel in the previous equation. With this result we can write the second derivative of $u(x,y,t)$ with respect to x, by using the fact that

$$\frac{\partial^2 u}{\partial x^2} = u_{xx} = (u_x)_x = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right),$$

so we find

$$\begin{aligned}u_{xx} &= \frac{1}{\mathcal{J}}\Big[(u_x y_\eta)_\xi - (u_x y_\xi)_\eta\Big] \\ &= \frac{1}{\mathcal{J}}\Big[\left(\frac{1}{\mathcal{J}}\Big[(u y_\eta)_\xi - (u y_\xi)_\eta\Big] y_\eta\right)_\xi - \left(\frac{1}{\mathcal{J}}\Big[(u y_\eta)_\xi - (u y_\xi)_\eta\Big] y_\xi\right)_\eta\Big] \\ &= \frac{1}{\mathcal{J}}\left[\left(\frac{u_\xi y_\eta^2 - u_\eta y_\xi y_\eta}{\mathcal{J}}\right)_\xi - \left(\frac{u_\xi y_\eta y_\xi - u_\eta y_\xi^2}{\mathcal{J}}\right)_\eta\right] \\ &= \frac{1}{\mathcal{J}}\left[\left(\frac{u_\xi y_\eta^2}{\mathcal{J}}\right)_\xi - \left(\frac{u_\eta y_\xi y_\eta}{\mathcal{J}}\right)_\xi - \left(\frac{u_\xi y_\eta y_\xi}{\mathcal{J}}\right)_\eta + \left(\frac{u_\eta y_\xi^2}{\mathcal{J}}\right)_\eta\right].\end{aligned} \tag{41}$$

In the same way we can determine u_y and u_{yy}:

$$\begin{aligned}\frac{\partial u}{\partial y} = u_y &= u_\xi \xi_y + u_\eta \eta_y \\ &= \frac{1}{\mathcal{J}}\Big[-u_\xi x_\eta + u_\eta x_\xi\Big] \\ &= \frac{1}{\mathcal{J}}\Big[(u x_\xi)_\eta - (u x_\eta)_\xi\Big]\end{aligned} \tag{42}$$

and since $u_{yy} = (u_y)_y$ we determine that

$$
\begin{aligned}
u_{yy} &= \frac{1}{\mathcal{J}}\Big[(u_y x_\xi)_\eta - (u_y x_\eta)_\xi\Big] \\
&= \frac{1}{\mathcal{J}}\Big[\ \Big(\frac{1}{\mathcal{J}}\Big[(ux_\xi)_\eta - (ux_\eta)_\xi\Big]x_\xi\Big)_\eta\ -\ \Big(\frac{1}{\mathcal{J}}\Big[(ux_\xi)_\eta - (ux_\eta)_\xi\Big]x_\eta\Big)_\xi\ \Big] \\
&= \frac{1}{\mathcal{J}}\left[\ \left(\frac{u_\eta x_\xi^2 - u_\xi x_\eta x_\xi}{\mathcal{J}}\right)_\eta\ -\ \left(\frac{u_\eta x_\xi x_\eta - u_\xi x_\eta^2}{\mathcal{J}}\right)_\xi\ \right] \\
&= \frac{1}{\mathcal{J}}\left[\ \left(\frac{u_\eta x_\xi^2}{\mathcal{J}}\right)_\eta\ -\ \left(\frac{u_\xi x_\eta x_\xi}{\mathcal{J}}\right)_\eta\ -\ \left(\frac{u_\eta x_\xi x_\eta}{\mathcal{J}}\right)_\xi\ +\ \left(\frac{u_\xi x_\eta^2}{\mathcal{J}}\right)_\xi\ \right].
\end{aligned}
\tag{43}
$$

The left-hand-side of (39) becomes

$$
u_t = u_\theta + \frac{1}{\mathcal{J}}\Big[(x_\theta y_\xi - y_\theta x_\xi)\,u_\eta\ -\ (x_\theta y_\eta - y_\theta x_\eta)\,u_\xi\Big] \tag{44}
$$

So now we are able to write (39) in the transformed coordinates as

$$
\begin{aligned}
&\frac{1}{\mathcal{J}}\Big[(x_\theta y_\xi - y_\theta x_\xi)\,u_\eta\ -\ (x_\theta y_\eta - y_\theta x_\eta)\,u_\xi\Big] \\
&= \frac{1}{\mathcal{J}}\left[\ \left(\frac{u_\xi y_\eta^2}{\mathcal{J}}\right)_\xi\ -\ \left(\frac{u_\eta y_\xi y_\eta}{\mathcal{J}}\right)_\xi\ -\ \left(\frac{u_\xi y_\eta y_\xi}{\mathcal{J}}\right)_\eta\ +\ \left(\frac{u_\eta y_\xi^2}{\mathcal{J}}\right)_\eta\ \right] + \\
&\quad \frac{1}{\mathcal{J}}\left[\ \left(\frac{u_\eta x_\xi^2}{\mathcal{J}}\right)_\eta\ -\ \left(\frac{u_\xi x_\eta x_\xi}{\mathcal{J}}\right)_\eta\ -\ \left(\frac{u_\eta x_\xi x_\eta}{\mathcal{J}}\right)_\xi\ +\ \left(\frac{u_\xi x_\eta^2}{\mathcal{J}}\right)_\xi\ \right] \\
&= \frac{1}{\mathcal{J}}\left[\ \left(\frac{x_\eta^2 + y_\eta^2}{\mathcal{J}}u_\xi\right)_\xi\ -\ \left(\frac{x_\xi x_\eta + y_\xi y_\eta}{\mathcal{J}}u_\eta\right)_\xi\ + \right. \\
&\qquad \left.\left(\frac{x_\xi^2 + y_\xi^2}{\mathcal{J}}u_\eta\right)_\eta\ -\ \left(\frac{x_\xi x_\eta + y_\xi y_\eta}{\mathcal{J}}u_\xi\right)_\eta\ \right]
\end{aligned}
\tag{45}
$$

Discretization of the transformed heat equation

Equation (45) can be discretized. We know enough about the stability of the numerical method Euler Forward (EF). So we will try to investigate the influence of this method in

the discretization of the transformed HE. Lets define

$$\left.\begin{aligned}\Delta\mathcal{X}^{n+1}_{i,j} &\equiv \mathcal{X}^{n+1}_{i,j} - \mathcal{X}^{n}_{i,j}\\ \Delta\mathcal{X}^{n}_{i+1,j} &\equiv \mathcal{X}^{n}_{i+1,j} - \mathcal{X}^{n}_{i-1,j}\\ \Delta\mathcal{X}^{n}_{i,j+1} &\equiv \mathcal{X}^{n}_{i,j+1} - \mathcal{X}^{n}_{i,j-1}\end{aligned}\right\} \qquad \text{with} \quad \mathcal{X} \in \{u, x, y\} \tag{46}$$

We will use the following approximations for the derivatives:

$$\begin{aligned}
u_\theta &\approx \frac{u^{n+1}_{i,j} - u^{n}_{i,j}}{\theta^{n+1}_{i,j} - \theta^{n}_{i,j}} = \frac{\Delta u^{n+1}_{i,j}}{\Delta\theta}\\
x_\theta &\approx \frac{x^{n+1}_{i,j} - x^{n}_{i,j}}{\theta^{n+1}_{i,j} - \theta^{n}_{i,j}} = \frac{\Delta x^{n+1}_{i,j}}{\Delta\theta}\\
y_\theta &\approx \frac{y^{n+1}_{i,j} - y^{n}_{i,j}}{\theta^{n+1}_{i,j} - \theta^{n}_{i,j}} = \frac{\Delta y^{n+1}_{i,j}}{\Delta\theta}\\
u_\xi &\approx \frac{u^{n}_{i+1,j} - u^{n}_{i-1,j}}{\xi^{n}_{i+1,j} - \xi^{n}_{i-1,j}} = \frac{\Delta u^{n}_{i+1,j}}{2\Delta\xi}\\
x_\xi &\approx \frac{x^{n}_{i+1,j} - x^{n}_{i-1,j}}{\xi^{n}_{i+1,j} - \xi^{n}_{i-1,j}} = \frac{\Delta x^{n}_{i+1,j}}{2\Delta\xi}\\
y_\xi &\approx \frac{y^{n}_{i+1,j} - y^{n}_{i-1,j}}{\xi^{n}_{i+1,j} - \xi^{n}_{i-1,j}} = \frac{\Delta y^{n}_{i+1,j}}{2\Delta\xi}\\
u_\eta &\approx \frac{u^{n}_{i,j+1} - u^{n}_{i,j-1}}{\eta^{n}_{i,j+1} - \eta^{n}_{i,j-1}} = \frac{\Delta u^{n}_{i,j+1}}{2\Delta\eta}\\
x_\eta &\approx \frac{x^{n}_{i,j+1} - x^{n}_{i,j-1}}{\eta^{n}_{i,j+1} - \eta^{n}_{i,j-1}} = \frac{\Delta x^{n}_{i,j+1}}{2\Delta\eta}\\
y_\eta &\approx \frac{y^{n}_{i,j+1} - y^{n}_{i,j-1}}{\eta^{n}_{i,j+1} - \eta^{n}_{i,j-1}} = \frac{\Delta y^{n}_{i,j+1}}{2\Delta\eta}
\end{aligned}$$

With these approximations we are using a kind of centered approximation around a point (i, j) in the transformed grid (ξ, η), where the distance between two neighboring mesh point in the ξ-direction or η-direction is $\Delta\xi$ respectively $\Delta\eta$.

Now we are able to write the discretization of the u_t

$$\begin{aligned}
u_t &= u_\theta + u_\xi \xi_t + u_\eta \eta_t \\
&= u_\theta + \frac{1}{J}\Big[(x_\theta y_\xi - y_\theta x_\xi)\, u_\eta - (x_\theta y_\eta - y_\theta x_\eta)\, u_\xi \Big] \\
&\approx \frac{\Delta u_{i,j}^{n+1}}{\Delta\theta} + \frac{4\Delta\xi\Delta\eta}{\Delta x_{i+1,j}^n \Delta y_{i,j+1}^n + \Delta x_{i,j+1}^n \Delta y_{i+1,j}^n} \cdot \Big(\frac{1}{4\Delta\theta\Delta\xi\Delta\eta} \cdot \\
&\quad \Big\{ \Delta u_{i,j+1}^n \left[\Delta x_{i,j}^{n+1} \Delta y_{i+1,j}^n - \Delta y_{i,j}^{n+1} \Delta x_{i+1,j}^n \right] - \\
&\quad\quad \Delta u_{i+1,j}^n \left[\Delta x_{i,j}^{n+1} \Delta y_{i,j+1}^n - \Delta y_{i,j}^{n+1} \Delta x_{i,j+1}^n \right] \Big\} \Big) \\
&= \frac{1}{\Delta\theta}\Big\{ \Delta u_{i,j}^{n+1} + A \Delta u_{i,j+1}^n - B \Delta u_{i+1,j}^n \Big\},
\end{aligned} \tag{47}$$

with

$$A = \frac{\Delta x_{i,j}^{n+1} \Delta y_{i+1,j}^n - \Delta y_{i,j}^{n+1} \Delta x_{i+1,j}^n}{\Delta x_{i+1,j}^n \Delta y_{i,j+1}^n + \Delta x_{i,j+1}^n \Delta y_{i+1,j}^n} \quad \text{and} \quad B = \frac{\Delta x_{i,j}^{n+1} \Delta y_{i,j+1}^n - \Delta y_{i,j}^{n+1} \Delta x_{i,j+1}^n}{\Delta x_{i+1,j}^n \Delta y_{i,j+1}^n + \Delta x_{i,j+1}^n \Delta y_{i+1,j}^n}$$

Lets look at the discretization of the following derivatives:

$$\begin{aligned}
\left(C u_\xi\right)_\xi &\approx \frac{\left(C u_\xi\right)_{i+\frac{1}{2},j} - \left(C u_\xi\right)_{i-\frac{1}{2},j}}{\Delta\xi} \\
&\approx \frac{1}{\Delta\xi}\left(\frac{C_{i+\frac{1}{2},j}\; u_{i+1,j} - C_{i+\frac{1}{2},j}\; u_{i,j}}{\Delta\xi} - \frac{C_{i-\frac{1}{2},j}\; u_{i,j} - C_{i-\frac{1}{2},j}\; u_{i-1,j}}{\Delta\xi} \right) \\
&= \frac{1}{(\Delta\xi)^2}\left(\frac{C_{i+1,j} + C_{i,j}}{2}\; u_{i+1,j} - \left[\frac{C_{i+1,j} + C_{i,j}}{2} + \frac{C_{i,j} + C_{i-1,j}}{2} \right] u_{i,j} + \right. \\
&\qquad \left. \frac{C_{i,j} + C_{i-1,j}}{2}\; u_{i-1,j} \right) \\
&= \frac{1}{2\,(\Delta\xi)^2}\Big(\Big[C_{i+1,j} + C_{i,j} \Big]\; u_{i+1,j} - \Big[C_{i+1,j} + 2C_{i,j} + C_{i-1,j} \Big]\; u_{i,j} + \\
&\qquad \Big[C_{i,j} + C_{i-1,j} \Big]\; u_{i-1,j} \Big).
\end{aligned}$$

$$
\begin{aligned}
(Du_\eta)_\xi &\approx \frac{(Du_\eta)_{i+1,j} - (Du_\eta)_{i-1,j}}{2\Delta\xi} \\
&\approx \frac{1}{2\Delta\xi}\left(\frac{D_{i+1,j+1}\ u_{i+1,j+1} - D_{i+1,j-1}\ u_{i+1,j-1}}{2\Delta\eta} - \right. \\
&\qquad \left. \frac{D_{i-1,j+1}\ u_{i-1,j+1} - D_{i-1,j-1}\ u_{i-1,j-1}}{2\Delta\eta} \right) \\
&\approx \frac{1}{4\Delta\xi\Delta\eta}\Big(D_{i+1,j+1}\ u_{i+1,j+1} - D_{i+1,j-1}\ u_{i+1,j-1} - \\
&\qquad D_{i-1,j+1}\ u_{i-1,j+1} + D_{i-1,j-1}\ u_{i-1,j-1} \Big).
\end{aligned}
$$

$$
\begin{aligned}
\left(Eu_\eta\right)_\eta &\approx \frac{\left(Eu_\eta\right)_{i,j+\frac{1}{2}} - \left(Eu_\eta\right)_{i,j-\frac{1}{2}}}{\Delta\eta} \\
&\approx \frac{1}{\Delta\eta}\left(\frac{E_{i,j+\frac{1}{2}}\ u_{i,j+1} - E_{i,j+\frac{1}{2}}\ u_{i,j}}{\Delta\eta} - \frac{E_{i,j-\frac{1}{2}}\ u_{i,j} - E_{i,j-\frac{1}{2}}\ u_{i,j-1}}{\Delta\eta}\right) \\
&= \frac{1}{(\Delta\eta)^2}\left(\frac{E_{i,j+1}+E_{i,j}}{2}\ u_{i,j+1}\ -\ \left[\frac{E_{i,j+1}+E_{i,j}}{2}\ +\ \frac{E_{i,j}+E_{i,j-1}}{2}\right]\ u_{i,j}\ + \right. \\
&\qquad \left.\frac{E_{i,j}+E_{i,j-1}}{2}\ u_{i,j-1}\ \right) \\
&= \frac{1}{2\,(\Delta\eta)^2}\Big(\ \Big[E_{i,j+1}+E_{i,j}\Big]\ u_{i,j+1}\ -\ \Big[E_{i,j+1}+2E_{i,j}+E_{i,j-1}\Big]\ u_{i,j}\ + \\
&\qquad \Big[E_{i,j}+E_{i,j-1}\Big]\ u_{i,j-1}\ \Big).
\end{aligned}
$$

$$
\begin{aligned}
\left(Fu_\xi\right)_\eta &\approx \frac{\left(Fu_\eta\right)_{i,j+1} - \left(Fu_\eta\right)_{i,j-1}}{2\Delta\eta} \\
&\approx \frac{1}{2\Delta\eta}\left(\ \frac{F_{i+1,j+1}\ u_{i+1,j+1} - F_{i-1,j+1}\ u_{i-1,j+1}}{2\Delta\xi}\ - \right. \\
&\qquad \left.\frac{F_{i+1,j-1}\ u_{i+1,j-1} - F_{i-1,j-1}\ u_{i-1,j-1}}{2\Delta\xi}\ \right) \\
&\approx \frac{1}{4\Delta\xi\Delta\eta}\Big(\ F_{i+1,j+1}\ u_{i+1,j+1}\ -\ F_{i-1,j+1}\ u_{i-1,j+1}\ - \\
&\qquad F_{i+1,j-1}\ u_{i+1,j-1}\ +\ F_{i-1,j-1}\ u_{i-1,j-1}\ \Big).
\end{aligned}
$$

Now we can approximate the right hand side of (39) using (41) and (43):

$$
\begin{aligned}
& u_{xx} + u_{yy} \\
= \; & \frac{1}{\mathcal{J}} \left[\left(\frac{x_\eta^2 + y_\eta^2}{\mathcal{J}} u_\xi \right)_\xi - \left(\frac{x_\xi x_\eta + y_\xi y_\eta}{\mathcal{J}} u_\eta \right)_\xi + \right. \\
& \left. \left(\frac{x_\xi^2 + y_\xi^2}{\mathcal{J}} u_\eta \right)_\eta - \left(\frac{x_\xi x_\eta + y_\xi y_\eta}{\mathcal{J}} u_\xi \right)_\eta \right] \\
= \; & \frac{1}{\mathcal{J}} \left[\left(C u_\xi \right)_\xi - \left(D u_\eta \right)_\xi + \left(E u_\eta \right)_\eta - \left(F u_\xi \right)_\eta \right] \\
\approx \; & \frac{4 \Delta\xi \Delta\eta}{\Delta x_{i+1,j} \Delta y_{i,j+1} - \Delta x_{i,j+1} \Delta y_{i+1,j}} \left\{ \right. \\
& \frac{\left(C_{i+1,j} + C_{i,j} \right) u_{i+1,j} - \left(C_{i+1,j} + 2C_{i,j} + C_{i-1,j} \right) u_{i,j} + \left(C_{i,j} + C_{i-1,j} \right) u_{i-1,j}}{2(\Delta\xi)^2} - \\
& \frac{D_{i+1,j+1} u_{i+1,j+1} - D_{i+1,j-1} u_{i+1,j-1} - D_{i-1,j+1} u_{i-1,j+1} + D_{i-1,j-1} u_{i-1,j-1}}{4 \Delta\xi \Delta\eta} + \\
& \frac{\left(E_{i,j+1} + E_{i,j} \right) u_{i,j+1} - \left(E_{i,j+1} + 2E_{i,j} + E_{i,j-1} \right) u_{i,j} + \left(E_{i,j} + E_{i,j-1} \right) u_{i,j-1}}{2(\Delta\eta)^2} - \\
& \left. \frac{F_{i+1,j+1} u_{i+1,j+1} - F_{i-1,j+1} u_{i-1,j+1} - F_{i+1,j-1} u_{i+1,j-1} + F_{i-1,j-1} u_{i-1,j-1}}{4 \Delta\xi \Delta\eta} \right\}
\end{aligned}
\tag{48}
$$

Notice that we still have to approximate the derivatives[18] of $C_{i \pm p, j \pm q}$, even for D, E and F, where

$$
\begin{aligned}
C &= \frac{x_\eta^2 + y_\eta^2}{\mathcal{J}} \\
D &= \frac{x_\xi x_\eta + y_\xi y_\eta}{\mathcal{J}} = F \\
E &= \frac{x_\xi^2 + y_\xi^2}{\mathcal{J}}
\end{aligned}
$$

but these will not be given.

[18] for example $C_{i \pm p, j \pm q}$ where $p, q \in \{-1, 0, 1\}$

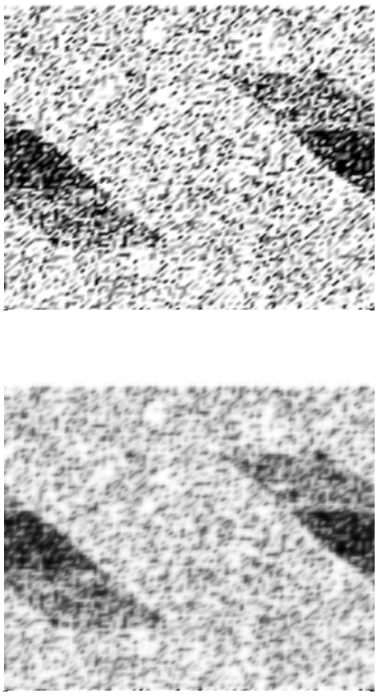


Figure 24: Top: the original image with added Salt-and-Pepper Noise. Lower: the denoised image with the Heat Equation (HE) on an R-refined adaptive grid.

Detailed information about the adaptive mesh PDEs and the involved *monitor function* can be found in [B03]. From this point on we used the C++ code from Anika Remorie, Thesis: *Adaptive numerical solutions of two-dimensional fingering patterns*, she solved the nonequilibrium Richards Equation (NERE) on an adaptive grid using the IMEX-θ scheme[19] given by:

$$\frac{\partial S}{\partial t} = \nabla.\left(D(S)\nabla\right) + \frac{\partial K(S)}{\partial z} + \tau\nabla.\left[K(S)\nabla\left(\frac{\partial S}{\partial t}\right)\right]. \tag{49}$$

By setting $\tau = 0$, $K(S) = 0$ and $D(S) = 1$ we have the Heat Equation (HE). Now we just need to set the correct parameters in the provided program. In Figure 24 we see the result of the HE. For the PME we need to plug in the correct $D(S)$, but this is not done in this report.

In Figures 25 - 26 we see that R-refinement technique works for the detection of edges using the HE. The more the noise in the image, the less grid points are available for detecting the edges to speed up the denoising process. Because grid points are also used for detecting noise in the image.

[19] this was solved with the Bi-CGSTAB (Bi-Conjugate Gradient Stabilized) method

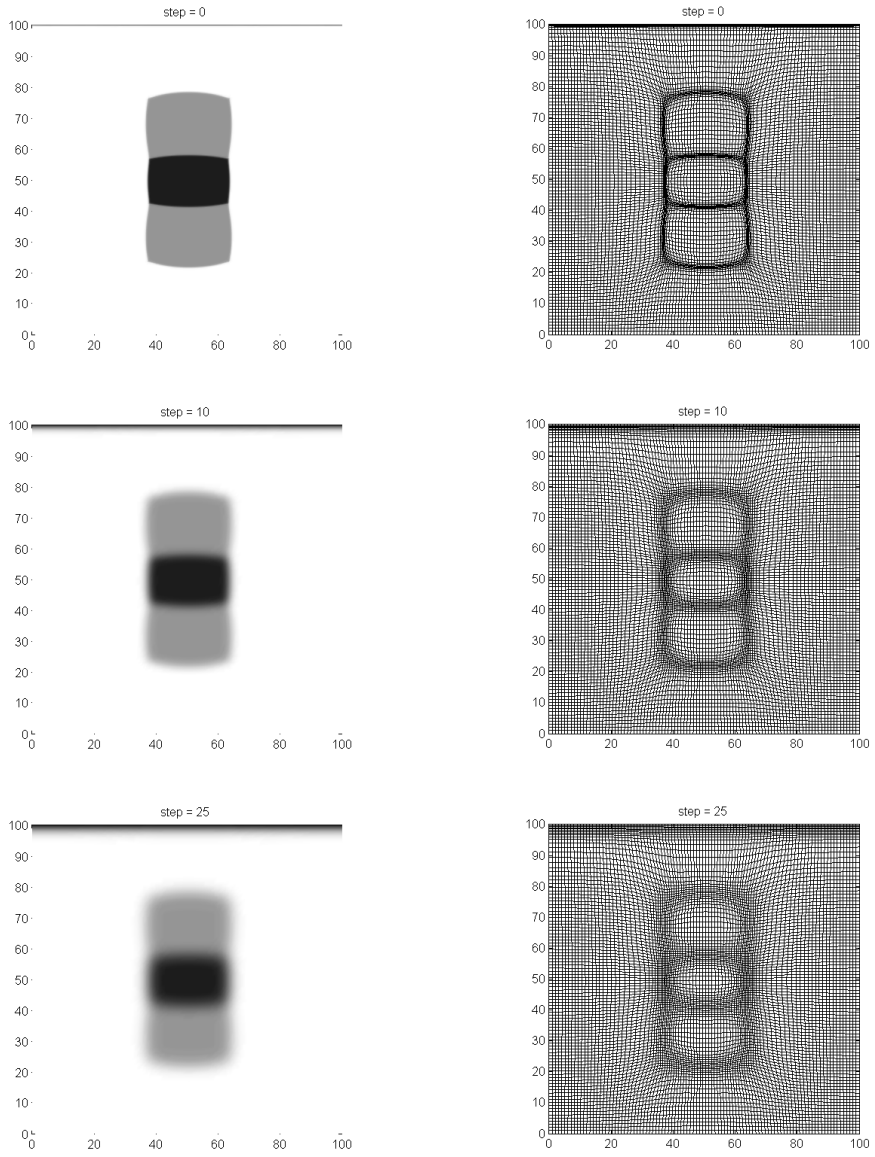


Figure 25: **No noise** The left column gives the resulting images after applying the Heat Equation. In the right column we see the adaptation of the mesh. The edges of the internal rectangles are well detected. The first row is the initial situation, the second and third row are corresponding with iteration step 10 respectively 25.

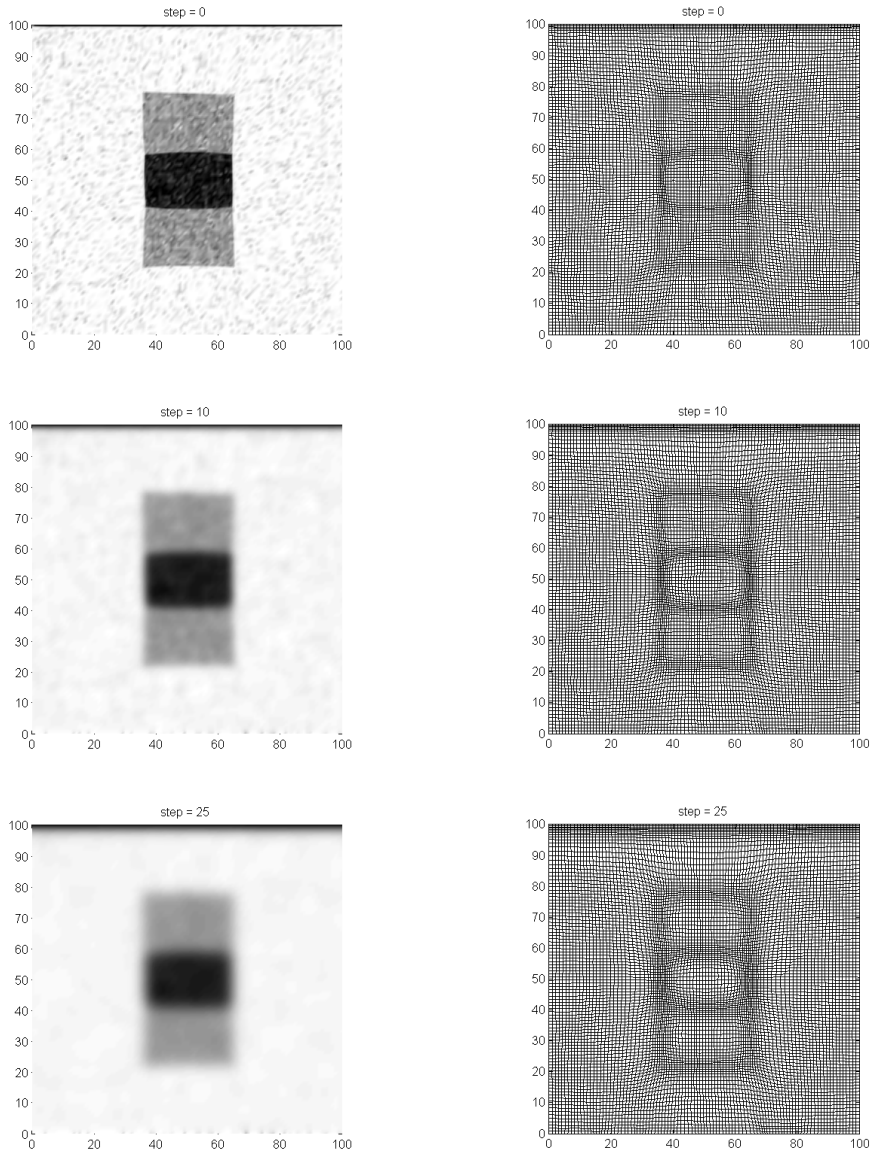


Figure 26: **Gaussian noise m=0, v=0.01** The left column gives the resulting images after applying the Heat Equation. In the right column we see the adaptation of the mesh. The edges of the internal rectangles are well detected. The first row is the initial situation, the second and third row are corresponding with iteration step 10 respectively 25.

3.5 Conclusion and Future Work

In the previous sections we have seen that the Heat Equation (HE) and the Perona-Malik Equation (PME) have a smoothening effect during image processing. Depending on the noise involved, the PME has desirable results. For example; the Salt-and-Pepper noise is the worst of the three possibilities mentioned, since it strongly depends on the percentages of changed pixels *new* structures might be formed. But when Gaussian white noise is involved the denoised image with the PME nicely resembles with the original image, meaning that edges are preserved. On the other hand, if there are clear structures formed by the added noise, neither the HE or PM will denoise these structures *properly*; meaning that these structures will be visible in the resulting denoised image when applying the PME, even with a proper value for the parameter ϵ.

As we saw with the Mona Lisa image; denoising with PME gives a nicely smoothened image where all the applied details in the cloud by the artist, are removed since they seem similar to noise. We will just have to do with this!

A major setback is the color of the background. In the previous chapter we also saw that the PME works like a locally averaging filter. This means that especially when there are white backgrounds involved the resulting denoised image seems to have a gray shade as background. This needs to be taken care of with other techniques; for example rescaling the resulted denoised image in the range $[0, 1]$, by increasing the brightness[20], see Figure 27.

(a) (b)

Figure 27: Increasing the brightness of the resulting denoised images: (a) from the *rectangle* in Figure 15, (b) from the **Face** in Figure 23.

This is a part of image reconstruction, but not a part of the treated PDE approach. Therefore we will not pay further attention to it.

To avoid many iteration steps with the PME we can use a unconditionally stable finite difference method such as an Implicit method which is consistent with the PME. Another approach is using adaptive grids. This way we can localize edges early in the iteration.

[20] for example in *Microsoft Office Picture Manager* by increasing the brightness from 0 to 20

4 References

[$B01$] Numerical Partial Differential Equations: Finite Difference Methods
Text in Applied Mathematics 22
J.W. Thomas, 1995 Springer-Verlag, ISBN: 0-387-97999-9

[$B02$] Partial Differential Equations: An Introduction
Walter A. Strauss, Wiley 1992, ISBN: 0-471-54868-5

[$B03$] Communications in Computational Physics
Balanced Monitoring of Flow Phenomena in Moving Mesh Methods
A. van Dam, P. A. Zegeling
Volume 7, Number 1, January 2010, ISSN: 1815-2406, pp. 138-170

[$B04$] Image Processing, Analysis, and Machine Vision
International Student Editon, Third Edition
Milan Sonka, Vaclav Hlavac, Roger Boyle
Cengage Learning, ISBN-13: 978-0-495-24438-7

[$B05$] A Primer On WAVELETS and Their Scientific Applications
James S. Walker, 2008 Second Edition, ISBN: 1-58488-745-1

[$P01$] A review of P.D.E. models in image processing and image analysis
F. Guichard, L. Moisan, J.-M. Morel
http://www.math-info.univ-paris5.fr/~moisan/papers/CMLA2001-09.pdf
Last visited: April 2011

[$P02$] Scale-Space and Edge Detection Using Anisotropic Diffusion
Pietro Perona and Jitendra Malik; IEEE Trans. Vol. 12. July 1990; 629-639

[$P03$] Generalized Perona-Malik Equation Based on Entropy Maximization
Hesan Z. Rafi, Hamid Soltanian-Zadeh, *2003*

[$P04$] Medical Image Analysis, Computer Science and Electrical Engineering Dept.
West Virginia University, CS 593/791, 20th January 2006
http://www.sci.utah.edu/~gerig/CS7960-S2010/materials/Perona-Malik/lecture4.pdf
Last visited: June 2011

[$P05$] Perona-Malik equation and its numerical properties
Maciej Wielgus, Lipiec 2010
http://students.mimuw.edu.pl/~mwielgus/files/PM_equation.pdf
Last visited: June 2011

[$P06$] A non-local algorithm for image denoising
Antoni Buades, Bartumeu Coll, Jean-Michel Morel
http://perso.telecom-paristech.fr/~gousseau/ATSI/projets/nlmeans.pdf
Last visited: June 2011

[P07] An Adaptive Finite Volume Scheme for Solving Nonlinear Diffusion Equations in Image Processing
Zuzana Krivä and Karol Mikula
Journal of Visual Communication and Image Representation 13, 22 - 35 (2002)

[P08] Anisotropic Diffusion in Image Processing
Joachim Weickert, Department of Computer Science University of Copenhagen, 1998

[P09] Three-dimensional adaptive moving meshes and their visualization
Matthijs Sypkens Smit (Advisor: Paul A. Zegeling), Utrecht University, April 5, 2005
http://matthijs.mired.nl/thesis/thesis_hyperref.pdf
Last visited: June 2011

[L01] http://en.wikipedia.org/wiki (Last visited: June 2011)
Topics: (a) Digital signal processing, (b) Anisotropic diffusion, (c) PSNR,
(d) Von Neumann stability analysis, (e) Grayscale, (f) Image noise,
(g) Gaussian noise, (h) Heat Equation, (i) Scale space, (j) Sarrus' rule

Appendix

A List of Approximations

Approximating u_{xx} via the Taylor Expansions

If $u(x,t)$ is smooth enough, then the approximation of u in the point $(x+\Delta x, t)$ is equal to

$$u(x \pm \Delta x, t) := T^{\pm}_{n,t}(u(x,t)) + R^{\pm}_{n,t}(u(x,t)) = T^{\pm}_{n,t}(u) + R^{\pm}_{n,t}(u) \tag{50}$$

$$u(x+\Delta x, t) = \sum_{j=0}^{n} \frac{(\Delta x)^j}{j!} \frac{\partial^j}{\partial x^j} u(x,t) + \sum_{j=n+1}^{\infty} \frac{(\Delta x)^j}{j!} \frac{\partial^j}{\partial x^j} u(x,t), \tag{51}$$

$$u(x-\Delta x, t) = \sum_{j=0}^{n} \frac{(-\Delta x)^j}{j!} \frac{\partial^j}{\partial x^j} u(x,t) + \sum_{j=n+1}^{\infty} \frac{(-\Delta x)^j}{j!} \frac{\partial^j}{\partial x^j} u(x,t). \tag{52}$$

So we have that

$$u_{xx} = \frac{\partial^2 u(x,t)}{\partial x^2} = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{(\Delta x)^2} + R^{+}_{3,t}(u) + R^{-}_{3,t}(u), \tag{53}$$

recall that if $\Delta x \to 0$ then $[R^{+}_{3,t}(u) + R^{-}_{3,t}(u)] \to 0$, which results in

$$u_{xx} \approx \frac{u^n_{i+1} - 2u^n_i + u^n_{i-1}}{(\Delta x)^2}. \tag{54}$$

Approximation for u_t, u_x and u_{xx} in 1-D

For the 1-D case we have that

$$u_t = \left.\frac{\partial u(x,t)}{\partial t}\right|_{x_i} \approx \frac{u^{n+1}_i - u^n_i}{\Delta t}, \tag{55}$$

$$u_x = \left.\frac{\partial u(x,t)}{\partial x}\right|_{x_i} \approx \frac{u^n_{i+1} - u^n_i}{\Delta x}. \tag{56}$$

$$\begin{aligned} u_{xx} &= \frac{\partial}{\partial x_i}\left(\frac{\partial u}{\partial x_i}\right) = \frac{\partial}{\partial x_i} u_x = \Big[u_x\Big]_x \Big|_{x_i} \\ &\approx \frac{\Big[u_x\Big]^n_{i+\frac{1}{2}} - \Big[u_x\Big]^n_{i-\frac{1}{2}}}{\Delta x} \approx \frac{\frac{u^n_{i+1} - u^n_i}{\Delta x} - \frac{u^n_i - u^n_{i-1}}{\Delta x}}{\Delta x} = \frac{u^n_{i+1} - 2u^n_i + u^n_{i-1}}{(\Delta x)^2}. \end{aligned} \tag{57}$$

Approximation for u_t, u_x and u_y in 2-D

For the 2-D case we have that

$$u_t = \left.\frac{\partial u(x,y,t)}{\partial t}\right|_{i,j} \approx \frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t}, \tag{58}$$

$$u_x = \left.\frac{\partial u(x,y,t)}{\partial x}\right|_{i,j} \approx \frac{u_{i+1,j}^{n} - u_{i,j}^{n}}{\Delta x}, \tag{59}$$

$$u_y = \left.\frac{\partial u(x,y,t)}{\partial y}\right|_{i,j} \approx \frac{u_{i,j+1}^{n} - u_{i,j}^{n}}{\Delta y}. \tag{60}$$

Approximation for u_{xx} and u_{yy} in 2-D

For the 2-D case we have that

$$\begin{aligned} u_{xx} &= \left.\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right)\right|_{i,j} = \left.\Big[u_x\Big]_x\right|_{i,j} \\ &\approx \frac{\Big[u_x\Big]_{i+\frac{1}{2},j}^{n} - \Big[u_x\Big]_{i-\frac{1}{2},j}^{n}}{\Delta x} = \frac{\dfrac{u_{i+1,j}^{n} - u_{i,j}^{n}}{\Delta x} - \dfrac{u_{i,j}^{n} - u_{i-1,j}^{n}}{\Delta x}}{\Delta x} = \frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2}. \end{aligned} \tag{61}$$

Similar we can approximate

$$\begin{aligned} u_{yy} &= \left.\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}\right)\right|_{i,j} = \left.\Big[u_y\Big]_y\right|_{i,j} \\ &\approx \frac{\Big[u_y\Big]_{i,j+\frac{1}{2}}^{n} - \Big[u_y\Big]_{i,j-\frac{1}{2}}^{n}}{\Delta y} = \frac{\dfrac{u_{i,j+1}^{n} - u_{i,j}^{n}}{\Delta y} - \dfrac{u_{i,j}^{n} - u_{i,j-1}^{n}}{\Delta y}}{\Delta y} = \frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2}. \end{aligned} \tag{62}$$

Approximation of mixed derivatives $u_{xy} = u_{yx}$ in 2D

For the 2D case we have that $u_{xy} = u_{yx}$

$$\left.\frac{\partial^2 u}{\partial x \partial y}\right|_{i,j} = \left.\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}\right)\right|_{i,j} = \left.\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right)\right|_{i,j} = \left.\frac{\partial^2 u}{\partial y \partial x}\right|_{i,j}$$

$$\begin{aligned} &\approx \frac{\left.\frac{\partial u}{\partial y}\right|_{i+1,j} - \left.\frac{\partial u}{\partial y}\right|_{i-1,j}}{2\Delta x} \approx \frac{\frac{u^n_{i+1,j+1} - u^n_{i+1,j-1}}{2\Delta y} - \frac{u^n_{i-1,j+1} - u^n_{i-1,j-1}}{2\Delta y}}{2\Delta x} \\ &= \frac{u^n_{i+1,j+1} - u^n_{i+1,j-1} - u^n_{i-1,j+1} + u^n_{i-1,j-1}}{4\Delta x \Delta y}. \end{aligned} \tag{63}$$

The stecil form of this approximation is

$$\frac{1}{4\Delta x \Delta y}\begin{pmatrix} -1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & -1 \end{pmatrix}. \tag{64}$$

We also can compute $u_{xy} = u_{yx}$ via

$$\left.\frac{\partial^2 u}{\partial x \partial y}\right|_{i,j} = \left.\frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}\right)\right|_{i,j} = \left.\frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right)\right|_{i,j} = \left.\frac{\partial^2 u}{\partial y \partial x}\right|_{i,j}$$

$$\begin{aligned} &\approx \frac{\left.\frac{\partial u}{\partial y}\right|_{i+\frac{1}{2},j} - \left.\frac{\partial u}{\partial y}\right|_{i-\frac{1}{2},j}}{\Delta x} \approx \frac{\frac{u^n_{i+\frac{1}{2},j+1} - u^n_{i+\frac{1}{2},j-1}}{2\Delta y} - \frac{u^n_{i-\frac{1}{2},j+1} - u^n_{i-\frac{1}{2},j-1}}{2\Delta y}}{\Delta x} \\ &\approx \frac{\frac{\frac{u^n_{i+1,j+1} + u^n_{i,j+1}}{2} - \frac{u^n_{i+1,j-1} + u^n_{i,j-1}}{2}}{2\Delta y} - \frac{\frac{u^n_{i,j+1} + u^n_{i-1,j+1}}{2} - \frac{u^n_{i,j-1} + u^n_{i-1,j-1}}{2}}{2\Delta y}}{\Delta x} \\ &= \frac{\frac{u^n_{i+1,j+1}}{2} + \frac{u^n_{i,j+1}}{2} - \frac{u^n_{i+1,j-1}}{2} - \frac{u^n_{i,j-1}}{2} - \frac{u^n_{i,j+1}}{2} - \frac{u^n_{i-1,j+1}}{2} + \frac{u^n_{i,j-1}}{2} + \frac{u^n_{i-1,j-1}}{2}}{2\Delta x \Delta y} \\ &= \frac{u^n_{i+1,j+1} + u^n_{i,j+1} - u^n_{i+1,j-1} - u^n_{i,j-1} - u^n_{i,j+1} - u^n_{i-1,j+1} + u^n_{i,j-1} + u^n_{i-1,j-1}}{4\Delta x \Delta y}. \end{aligned} \tag{65}$$

We see that both approximations of u_{xy} in (63) and (65) have the same stencil (64). We can also construct other approximations for u_{xy} if needed, which have another stencil.

Approximating the 2D heat equation

$$u_t = \Delta u = u_{xx} + u_{yy} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial y}\right) \tag{66}$$

Using the approximations in (58), (61) and (62) we find that (66) becomes

$$u_{i,j}^{n+1} = u_{i,j}^n + \frac{\Delta t}{(\Delta x)^2}\left(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\right) + \frac{\Delta t}{(\Delta y)^2}\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right). \tag{67}$$

For efficiency purposes we set $r = \frac{\Delta t}{(\Delta x)^2}$ and $s = \frac{\Delta t}{(\Delta y)^2}$. Thus we can rewrite (67) as

$$u_{i,j}^{n+1} = (1 - 2r - 2s)u_{i,j}^n + r(u_{i+1,j}^n + u_{i-1,j}^n) + s(u_{i,j+1}^n + u_{i,j-1}^n). \tag{68}$$

Approximating the 2D Perona-Malik PDE

$$u_t = \nabla \cdot \left(\frac{1}{1+\epsilon|\nabla u|^2}\nabla u\right) = \frac{\partial}{\partial x}\left[\frac{1}{1+\epsilon|\nabla u|^2}\frac{\partial u}{\partial x}\right]\Bigg|_{i,j} + \frac{\partial}{\partial y}\left[\frac{1}{1+\epsilon|\nabla u|^2}\frac{\partial u}{\partial y}\right]\Bigg|_{i,j}. \tag{69}$$

where

$$|\nabla u|^2 = \nabla u \cdot \nabla u = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2. \tag{70}$$

The following approximations are used:

$$u_x\Big|_{i+\frac{1}{2},j} = \frac{\partial u}{\partial x}\Big|_{i+\frac{1}{2},j} \approx \frac{u_{i+1,j}^n - u_{i,j}^n}{\Delta x}, \tag{71}$$

$$u_x\Big|_{i-\frac{1}{2},j} = \frac{\partial u}{\partial x}\Big|_{i-\frac{1}{2},j} \approx \frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}, \tag{72}$$

$$u_y\Big|_{i,j+\frac{1}{2}} = \frac{\partial u}{\partial y}\Big|_{i,j+\frac{1}{2}} \approx \frac{u_{i,j+1}^n - u_{i,j}^n}{\Delta y}, \tag{73}$$

$$u_y\Big|_{i,j-\frac{1}{2}} = \frac{\partial u}{\partial y}\Big|_{i,j-\frac{1}{2}} \approx \frac{u_{i,j}^n - u_{i,j-1}^n}{\Delta y}, \tag{74}$$

$$\left[u_x^2\right]\Big|_{i+\frac{1}{2},j} = \left(u_x\Big|_{i+\frac{1}{2},j}\right)^2 \approx \left(\frac{u_{i+1,j}^n - u_{i,j}^n}{\Delta x}\right)^2, \tag{75}$$

$$\left[u_x^2\right]\Big|_{i-\frac{1}{2},j} = \left(u_x\Big|_{i-\frac{1}{2},j}\right)^2 \approx \left(\frac{u_{i,j}^n - u_{i-1,j}^n}{\Delta x}\right)^2, \tag{76}$$

$$\left[u_y^2\right]\Big|_{i,j+\frac{1}{2}} = \left(u_y\Big|_{i,j+\frac{1}{2}}\right)^2 \approx \left(\frac{u_{i,j+1}^n - u_{i,j}^n}{\Delta y}\right)^2, \tag{77}$$

$$\left[u_y^2\right]\Big|_{i,j-\frac{1}{2}} = \left(u_y\Big|_{i,j-\frac{1}{2}}\right)^2 \approx \left(\frac{u^n_{i,j}-u^n_{i,j-1}}{\Delta y}\right)^2, \tag{78}$$

$$u^n_{i+\frac{1}{2},j+1} \approx \frac{u^n_{i+1,j+1}+u^n_{i,j+1}}{2} \quad \text{and} \quad u^n_{i+\frac{1}{2},j-1} \approx \frac{u^n_{i+1,j-1}+u^n_{i,j-1}}{2}, \tag{79}$$

$$u^n_{i+1,j+\frac{1}{2}} \approx \frac{u^n_{i+1,j+1}+u^n_{i+1,j}}{2} \quad \text{and} \quad u^n_{i-1,j+\frac{1}{2}} \approx \frac{u^n_{i-1,j+1}+u^n_{i-1,j}}{2}, \tag{80}$$

$$u^n_{i-\frac{1}{2},j+1} \approx \frac{u^n_{i,j+1}+u^n_{i-1,j+1}}{2} \quad \text{and} \quad u^n_{i-\frac{1}{2},j-1} \approx \frac{u^n_{i,j-1}+u^n_{i-1,j-1}}{2}, \tag{81}$$

$$u^n_{i+1,j-\frac{1}{2}} \approx \frac{u^n_{i+1,j}+u^n_{i+1,j-1}}{2} \quad \text{and} \quad u^n_{i-1,j-\frac{1}{2}} \approx \frac{u^n_{i-1,j}+u^n_{i-1,j-1}}{2}, \tag{82}$$

$$\begin{aligned}\left[u_y^2\right]\Big|_{i+\frac{1}{2},j} &= \left(u_y\Big|_{i+\frac{1}{2},j}\right)^2 \approx \left(\frac{u^n_{i+\frac{1}{2},j+1}-u^n_{i+\frac{1}{2},j-1}}{2\Delta y}\right)^2 \\ &\approx \left(\frac{u^n_{i+1,j+1}+u^n_{i,j+1}-u^n_{i+1,j-1}-u^n_{i,j-1}}{4\Delta y}\right)^2,\end{aligned} \tag{83}$$

$$\begin{aligned}\left[u_x^2\right]\Big|_{i,j+\frac{1}{2}} &= \left(u_x\Big|_{i,j+\frac{1}{2}}\right)^2 \approx \left(\frac{u^n_{i+1,j+\frac{1}{2}}-u^n_{i-1,j+\frac{1}{2}}}{2\Delta x}\right)^2 \\ &\approx \left(\frac{u^n_{i+1,j+1}+u^n_{i+1,j}-u^n_{i-1,j+1}-u^n_{i-1,j}}{4\Delta x}\right)^2,\end{aligned} \tag{84}$$

$$\begin{aligned}\left[u_y^2\right]\Big|_{i-\frac{1}{2},j} &= \left(u_y\Big|_{i-\frac{1}{2},j}\right)^2 \approx \left(\frac{u^n_{i-\frac{1}{2},j+1}-u^n_{i-\frac{1}{2},j-1}}{2\Delta y}\right)^2 \\ &\approx \left(\frac{u^n_{i,j+1}+u^n_{i-1,j+1}-u^n_{i,j-1}-u^n_{i-1,j-1}}{4\Delta y}\right)^2,\end{aligned} \tag{85}$$

$$\begin{aligned}\left[u_x^2\right]\Big|_{i,j-\frac{1}{2}} &= \left(u_x\Big|_{i,j-\frac{1}{2}}\right)^2 \approx \left(\frac{u^n_{i+1,j-\frac{1}{2}}-u^n_{i-1,j-\frac{1}{2}}}{2\Delta x}\right)^2 \\ &\approx \left(\frac{u^n_{i+1,j}+u^n_{i+1,j-1}-u^n_{i-1,j}-u^n_{i-1,j-1}}{4\Delta x}\right)^2.\end{aligned} \tag{86}$$

First we will approximate

$$\frac{\partial}{\partial x}\left[\frac{1}{1+\epsilon|\nabla u|^2}\frac{\partial u}{\partial x}\right]\Bigg|_{i,j}$$

$$\approx \frac{\dfrac{1}{1+\epsilon\left[u_x^2+u_y^2\right]\Big|_{i+\frac{1}{2},j}}\left.\dfrac{\partial u}{\partial x}\right|_{i+\frac{1}{2},j} - \dfrac{1}{1+\epsilon\left[u_x^2+u_y^2\right]\Big|_{i-\frac{1}{2},j}}\left.\dfrac{\partial u}{\partial x}\right|_{i-\frac{1}{2},j}}{\Delta x}$$

$$\approx \frac{\dfrac{(u^n_{i+1,j}-u^n_{i,j})/\Delta x}{1+\epsilon\left[\left(\frac{u^n_{i+1,j}-u^n_{i,j}}{\Delta x}\right)^2+\left(\frac{u^n_{i+1,j+1}+u^n_{i,j+1}-u^n_{i+1,j-1}-u^n_{i,j-1}}{4\Delta y}\right)^2\right]} - \dfrac{\dfrac{(u^n_{i,j}-u^n_{i-1,j})/\Delta x}{1+\epsilon\left[\left(\frac{u^n_{i,j}-u^n_{i-1,j}}{\Delta x}\right)^2+\left(\frac{u^n_{i,j+1}+u^n_{i-1,j+1}-u^n_{i,j-1}-u^n_{i-1,j-1}}{4\Delta y}\right)^2\right]}{\Delta x} \tag{87}$$

$$= \frac{u^n_{i+1,j}-u^n_{i,j}}{(\Delta x)^2+\epsilon\left(u^n_{i+1,j}-u^n_{i,j}\right)^2+\frac{\epsilon(\Delta x)^2}{16(\Delta y)^2}\left(u^n_{i+1,j+1}+u^n_{i,j+1}-u^n_{i+1,j-1}-u^n_{i,j-1}\right)^2} - \frac{u^n_{i,j}-u^n_{i-1,j}}{(\Delta x)^2+\epsilon\left(u^n_{i,j}-u^n_{i-1,j}\right)^2+\frac{\epsilon(\Delta x)^2}{16(\Delta y)^2}\left(u^n_{i,j+1}+u^n_{i-1,j+1}-u^n_{i,j-1}-u^n_{i-1,j-1}\right)^2}.$$

Secondly we will approximate

$$\frac{\partial}{\partial y}\left[\frac{1}{1+\epsilon|\nabla u|^2}\frac{\partial u}{\partial y}\right]\Bigg|_{i,j}$$

$$\approx \frac{\dfrac{1}{1+\epsilon\left[u_x^2+u_y^2\right]\Big|_{i,j+\frac{1}{2}}}\left.\dfrac{\partial u}{\partial y}\right|_{i,j+\frac{1}{2}} - \dfrac{1}{1+\epsilon\left[u_x^2+u_y^2\right]\Big|_{i,j-\frac{1}{2}}}\left.\dfrac{\partial u}{\partial y}\right|_{i,j-\frac{1}{2}}}{\Delta y}$$

$$\approx \frac{\dfrac{(u^n_{i,j+1}-u^n_{i,j})/\Delta y}{1+\epsilon\left[\left(\frac{u^n_{i,j+1}-u^n_{i,j}}{\Delta y}\right)^2+\left(\frac{u^n_{i+1,j+1}+u^n_{i+1,j}-u^n_{i-1,j+1}-u^n_{i-1,j}}{4\Delta x}\right)^2\right]} - \frac{\dfrac{(u^n_{i,j}-u^n_{i,j-1})/\Delta y}{1+\epsilon\left[\left(\frac{u^n_{i,j}-u^n_{i,j-1}}{\Delta y}\right)^2+\left(\frac{u^n_{i+1,j}+u^n_{i+1,j-1}-u^n_{i-1,j}-u^n_{i-1,j-1}}{4\Delta x}\right)^2\right]}{\Delta y} \tag{88}$$

$$= \frac{u^n_{i,j+1}-u^n_{i,j}}{(\Delta y)^2+\epsilon\left(u^n_{i,j+1}-u^n_{i,j}\right)^2+\frac{\epsilon(\Delta y)^2}{16(\Delta x)^2}\left(u^n_{i+1,j+1}+u^n_{i+1,j}-u^n_{i-1,j+1}-u^n_{i-1,j}\right)^2} - \frac{u^n_{i,j}-u^n_{i,j-1}}{(\Delta y)^2+\epsilon\left(u^n_{i,j}-u^n_{i,j-1}\right)^2+\frac{\epsilon(\Delta y)^2}{16(\Delta x)^2}\left(u^n_{i+1,j}+u^n_{i+1,j-1}-u^n_{i-1,j}-u^n_{i-1,j-1}\right)^2}.$$

With this we find that the FD scheme for the Perona-Malik PDE becomes:

$$u^{n+1}_{i,j} = u^n_{i,j} + \Delta t\Big((87)+(88)\Big). \tag{89}$$

B List of Matlab Source code

In this chapter we have MATLAB code for denoising 1-D or 2-D signals. Please read the caution before using this code!

B.1 Code for 1-D: Heat and Perona-Malik PDE

Listing 1: For computing S_1 in (3) – **f1.m**

```
function u_x0 = f1(x)
%
%  Function u(x,0)=f(x) is the initial condition, with  x in [0,1].
%

a = 0.10;  b = 0.40;  c = 1.00;  u_x0 = 0.00;

% For the exact solution u = f(x - ct) we need f to be periodic.
x = x - floor(x); %periodic f(x) --> ...=f(33.1)=...=f(2.1)=f(1.1)=f(0.1)

if x < 0.25 | x > 0.75
    u_x0=c;
elseif x >= 0.25 & x < 0.40 | x > 0.6  & x <= 0.75
    u_x0=b;
elseif x >= 0.40 & x <= 0.6
    u_x0=a;
end

clear a b c x
```

Listing 2: For computing S_1 in (3) with some noise – **f1_EF.m**

```
function u_x0 = f1_EF(x)
%
%  Function u(x,0)=f(x) is the initial condition, with x in [0,1].
%

a = 0.10;   b = 0.40;   c = 1.00;   u_x0 = 0.00;

% For the exact solution u = f(x - ct) we need f to be periodic.
x = x - floor(x); %periodic f(x) --> ...=f(33.1)=...=f(2.1)=f(1.1)=f(0.1)

if x == 0 | x == 1
    u_x0=c;
elseif x > 0 & x < 0.25
    u_x0= c+(x^2)*sin((251*x)^(4));
elseif  x > 0.75 & x < 1
    y = 1 - x;
    u_x0= c+(y^2)*sin((251*y)^(4));
elseif x >= 0.25 & x < 0.40
    u_x0= b+(x^3)*sin((83*x)^(4));
elseif x > 0.6  & x <= 0.75
    y = 1 - x;
    u_x0= b+(y^3)*sin((83*y)^(4));
elseif x >= 0.40 & x <= 0.5
    u_x0= a+(x^4)*sin((179*x)^(4));
elseif x >  0.50 & x <= 0.6
    y = 1 - x;
    u_x0= a+(y^4)*sin((179*y)^(4));
end

clear a b c x t1 t2 t3 y
```

Listing 3: For denoising with the Heat Equation – **Heat1D.m**

```
function [usol,res_ef,totstep] = Heat1D(M,N,T,c,stop)
%
% Finite Difference scheme Heat PDE (with)
%                 u_t = c u_xx,     where  c is a constant.
%
% Input  : M        -> such that dx = 1/M,      x = [0,1].
%          N        -> such that dt = T/N,      t = [0,T].
%          T        -> see input N.
%          c        -> constant 'c' in the heat equation.
%          stop     -> for extending the T_end (choose an integer)
%
% Output : usol     -> The numerical solution u, with the Euler Forward scheme.
%          res_ef   -> Residual (using 2-norm).
%          totstep  -> Total number of steps needed.
%
% See also PERONAMALIK1D
%

tic

dt  = T/abs(N);
dx  = 1/abs(M); xx = linspace(0,1,M+1);
lmd = c*(dt/(dx^2));      % Courant number

usol = zeros(M+1,3);      % Set dimension usol (numerical solution of u(x,t)).

% Set initial condition
for i=1:M+1, usol(i,1) = f1_EF(xx(i)); usol(i,2) = usol(i,1); end

utemp   = usol(:,2);      % array for the temporary solution
totstep = -1;
mm      = 1.0;            % power for the norm %1.125 zie steps6261

for j=1:(stop*N)+1
    for i=1:M+1
        if i == 1
            usol(i,3) = usol(i,2)+(c*lmd)*(usol(i,2)-2*usol(i,2));
        elseif i > 1 && i < M+1
            usol(i,3) = usol(i,2)+(c*lmd)*(usol(i+1,2)-2*usol(i,2)+usol(i-1,2));
        elseif i == M+1
            usol(i,3) = usol(i,2)+(c*lmd)*(-2*usol(i,2)+usol(i-1,2));
        end
    end
    % Set boundary conditions u(0,t) = u(1,t) ==> usol(1,j) = usol(M+1,j).
    usol(1,3)       = usol(1,2);
    usol(M+1,3)     = usol(M+1,2);
    usol([2:M+1],2) = usol([2:M+1],3);

    res_ef(j) = norm(utemp - usol(:,3),2);  utemp = usol(:,3);

    if (res_ef(j) < (dx^(mm))), dx2 = dx^(mm) , rsj = res_ef(j), jj = j,  break, end
    totstep = totstep +1;
end

toc
toc_EF = toc
figure(121); plot(log(res_ef))

clear M N T c stop                          % Clearing the input variables (memory).
clear dx dt lmd tt xx i j mm t toc_EF utemp  % Clearing other variables     (memory).
```

Listing 4: For denoising with the Perona-Malik Equation – **PeronaMalik1D.m**

```
function [usol,res_pm,totstep] = PeronaMalik1D(M,N,T,c,stop)
%
% Finite Difference scheme Perona-Mailik (with EF)
%                 u_t = ( u_x /(1 + c|u_x|^2) )_x,
% where  c = epsilon, is a constant. For c=0, we have the heat equation
%                 u_t = u_xx.
%
% Input  : M        -> such that dx = 1/M,      x = [0,1].
%          N        -> such that dt = T/N,      t = [0,T].
%          T        -> see input N.
%          c        -> constant 'c'.
%          stop     -> for extending the T_end (choose an integer)
%
% Output : usol     -> The numerical solution u, with the Perona-Malik PDE.
%          res_PM   -> Residual (using 2-norm).
%          totstep  -> Total number of steps needed.
%
% See also HEAT1D
%

tic

if (nargin < 4)
    error('MATLAB:PeronaMalik','Not enough input arguments.');
end
if (nargin == 4)
    stop    = 1
end
if (nargin > 5)
    error('MATLAB:PeronaMalik','Too many input arguments.');
end

dt  = T/abs(N);
dx  = 1/abs(M); xx = linspace(0,1,M+1);

usol = zeros(M+1,3);   % Set dimension usol (numerical of solution u(x,t)).

% Set initial condition
for i=1:M+1, usol(i,1) = f1_EF(xx(i)); usol(i,2) = usol(i,1); end

utemp   = usol(:,2);
totstep = -1;
mm      = 1.8;  % power for the norm  %1.7 see steps24499 /  1.8 see steps73617

for j=1:(stop*N)+1
    for i=1:M+1

        if i == 1
            usol(i,3) = usol(i,2)+ (dt/dx^2)*h_ux(usol,i,2,dx,0,c);
        elseif i > 1 && i < M+1
            usol(i,3) = usol(i,2)+ (dt/dx^2)*h_ux(usol,i,2,dx,2,c);
        elseif i == M+1
            usol(i,3) = usol(i,2)+ (dt/dx^2)*h_ux(usol,i,2,dx,1,c);
        end
    end
    % Set boundary conditions u(0,t) = u(1,t) ==> usol(1,j) = usol(M+1,j).
    usol(1,3)       = usol(1,2);
    usol(M+1,3)     = usol(M+1,2);
    usol([2:M+1],2) = usol([2:M+1],3);

    res_pm(j) = norm(utemp - usol(:,3),2);  utemp = usol(:,3);

    if (res_pm(j) < (dx)^(mm)), dx2 = dx^(mm) , rsj = res_pm(j), jj = j,  break, end
    totstep = totstep +1;
end


```

```
toc
toc_PM = toc

clear M N T c stop                    % Clearing the input variables (memory).
clear dx dt lmd tt xx i j  mm toc_PM % Clearing other variables      (memory).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%    Built-in functions    %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [u_xx] = Vxx(u,i,j,dx,bound)
%
% bound==0 => u(i-1,j)=0, bound==1 => u(i+1,j)=0
%

if bound == 0
    u_xx = (u(i+1,j) -2*u(i,j))/(dx^2);
elseif bound==1
    u_xx = (-2*u(i,j) + u(i-1,j))/(dx^2);
elseif bound > 1
    u_xx = (u(i+1,j) -2*u(i,j) + u(i-1,j))/(dx^2);
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_x] = Vx(u,i,j,dx,bound)
%
% bound==1 => u(i+1,j)=0
%

if bound == 1
    u_x = u(i,j)/dx;
elseif bound < 1 || bound > 1
    u_x = (u(i+1,j)-u(i,j))/dx;
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [fux] = f_ux(u,i,j,dx,bound,c)

vx  = Vx(u,i,j,dx,bound);
vxx = Vxx(u,i,j,dx,bound);
t1  = vxx*(1 + (c*(abs(vx)^(2))) );
t2  = vx*(2*c*abs(vx)*abs(vxx));
n   = 1 + (2*c*abs(vx)^(2)) + ((c^2)*abs(vx)^(4));
fux = (t1 - t2)/n;

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_x] = Wx1(u,i,j,dx,bound)
%
% bound==1 => u(i+1,j)=0
%

if bound == 1
    u_x = u(i,j)/dx;
elseif bound == 0 || bound > 1
    u_x = (u(i+1,j)-u(i,j))/dx;
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
```

```
function [u_x] = Wx2(u,i,j,dx,bound)
%
% bound==1 => u(i-1,j)=0
%

if bound == 0
    u_x = u(i,j)/dx;
elseif bound > 0
    u_x = (u(i,j)-u(i-1,j))/dx;
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [gux] = g_ux(u,i,j,dx,bound,c)

wx1 = Wx1(u,i,j,dx,bound);
wx2 = Wx2(u,i,j,dx,bound);
t1  = wx1/(1 + (c*(abs(wx1)^(2))) );
t2  = wx2/(1 + (c*(abs(wx2)^(2))) );
gux = (t1 - t2)/dx;

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_x] = Ux1(u,i,j,dx,bound)
%
% bound==1 => u(i+1,j)=0
%

if bound == 1
    u_x = u(i,j)/dx;
elseif bound == 0 || bound > 1
    u_x = u(i+1,j)-u(i,j);
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_x] = Ux2(u,i,j,dx,bound)
%
% bound==1 => u(i-1,j)=0
%

if bound == 0
    u_x = u(i,j)/dx;
elseif bound > 0
    u_x = u(i,j)-u(i-1,j);
end

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [hux] = h_ux(u,i,j,dx,bound,c)

ux1 = Ux1(u,i,j,dx,bound);              % u(i+1,j) - u(i,j)
ux2 = Ux2(u,i,j,dx,bound);              % u(i,j)   - u(i-1,j)
t1  = ux1/(1 + ((c/(dx^2))*(abs(ux1)^(2))) );
t2  = ux2/(1 + ((c/(dx^2))*(abs(ux2)^(2))) );
hux = (t1 - t2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Listing 5: Testing the Heat and Perona-Malik PDE on a noisy signal – **test_HTPM.m**

```
addpath('./NumericalMethods');
addpath('./Functions');

M  = 500;       %1000;          % 500;
N  = 400;       %4000;          % 400;
T  = 0.0004;    %0.00016;       % 0.0004;
c1 = 1;
c2 = 0.02;

dx=1/M; dt=T/N; lmd = (c1*dt)/(dx^2)

a   = linspace(0,1,M+1);
fa  = zeros(M+1,1);
fef = zeros(M+1,1);

for i=1:M+1
    fa(i)  = f1(a(i));
    fef(i) = f1_EF(a(i));
end

stop = 1;  %25                  % variable for extending the end-time T_end

[usol_EF,rs_ef,stepEF] = Heat1D(M,N,T,c1,stop);
[usol_PM,rs_pm,stepPM] = PeronaMalik1D(M,N,T,c2,stop);

T = stop*T;

figure(1);
subplot(2,2,1);
  plot(a,fa,'-b');  axis([0 1 0 1.1]);
  title(sprintf("The Exact Solution (M = %d, N = %d, dx = 1/M  and dt = (T/%d)/N
                 with  T = %0.3g)",M,N,stop,T));
  xlabel('(a)');

subplot(2,2,2);
  plot(a,fef,'-r'); axis([0 1 0 1.1]);
  title(sprintf("The Signal u^{*}(x,t) with noise."));
  xlabel('(b)');

subplot(2,2,3);
  plot(a,usol_EF(:,3),'-c'); axis([0 1 0 1.1]);
  title(sprintf("The Solution with the Heat equation (PDE) :  u_{t} = k.u_{xx},
                 for k = %d",c1));
  xlabel('(c)');

subplot(2,2,4);
  plot(a,usol_PM(:,3),'-c'); axis([0 1 0 1.1]);
  title(sprintf("The Solution with Perona-Malik PDE :  u_{t} = ( u_x /(1 + c|u_x|^2) )_x
                 for c = %0.3g", c2 ));
  xlabel('(d)');

lineobj = findobj('type', 'line'); set(lineobj, 'linewidth', 1.2);
dddt    = 0.0001;
n1      = norm((fa - usol_EF(:,3)),2);

figure(2);
subplot(2,1,1);
  plot(a,(fa - usol_EF(:,3)) ,'-b');
  axis([0 1 (min((fa - usol_EF(:,3)))- dddt) (max((fa - usol_EF(:,3)))+ dddt)]);
  title(sprintf("The error with the Heat  PDE :  u_{exact} - u_{sol}"));
  xlabel('(a)');

subplot(2,1,2);
  plot(a,(fa - usol_PM(:,3)),'-b');
  axis([0 1 (min((fa - usol_PM(:,3)))- dddt) (max((fa - usol_PM(:,3)))+ dddt)]);
  title(sprintf("The error with the Perona-Malik  PDE :  u_{exact} - u_{sol}"));
  xlabel('(b)');
```

```
68
figure(17);
70   plot(a,usol_EF(:,3),'-r'); hold on; plot(a,fa,'-b'); axis([0 1 0 1.1]);
     title(sprintf("The Solution with the Heat equation (PDE) :  u_{t} = k.u_{xx},
72                  for k = %d     (M = %d, N = %d, dx = 1/M  and dt = T^{*}/N
                    with  T = %0.3g)",c1,M,N,T));
74   text(0.65,0.30, sprintf("dx = %2.5f     dt = %2.8f     dt/(dx^2) = %2.4f",dx,dt,lmd) );
     text(0.65,0.25, sprintf("Iteration steps = %d",stepEF) );
76   text(0.65,0.20, sprintf("T_{end} = %2.8f
                             T^{*} = %2.8f",((stepEF*1.00)*dt), (T/stop) ));
78   text(0.65,0.13, sprintf("|| u_{exact} - u_{sol} ||_{%d} = %2.10f",
                             M,norm((fa - usol_EF(:,3)),M))); %% the M-norm
80
     ylabel('u(x,t)');
82   xlabel("0 \leq x \leq 1");

84   lineobj = findobj('type', 'line'); set(lineobj, 'linewidth', 1.20);
     h       = legend('u_{sol}','u_{exact}',-1);
86
     figure(19);
88   plot(a,usol_PM(:,3),'-r');  hold on;
     plot(a,fa,'-b');            axis([0 1 0 1.1]);
90
     title(sprintf("The Solution with the Perona-Malik (PDE) :
92                  u_{t} = ( u_x /(1 + c|u_x|^2) )_x,
                    for c = %1.3f     (M = %d, N = %d, dx = 1/M
94                  and dt = T^{*}/N  with  T = %0.3g)",c2,M,N,T));

96   text(0.65,0.30, sprintf("dx = %2.5f     dt = %2.8f     dt/(dx^2) = %2.4f",dx,dt,lmd) );
     text(0.65,0.25, sprintf("Iteration steps = %d",stepPM) );
98   text(0.65,0.20, sprintf("T_{end} = %2.8f
                             T^{*} = %2.8f",((stepPM*1.00)*dt), (T/stop) ));
100  text(0.65,0.13, sprintf("|| u_{exact} - u_{sol} ||_{%d} = %2.10f",
                             M,norm((fa - usol_PM(:,3)),M))); %% the M-norm
102
     ylabel('u(x,t)');
104  xlabel(" 0 \leq x \leq 1");

106  lineobj = findobj('type', 'line'); set(lineobj, 'linewidth', 1.20);
     h       = legend('u_{sol}','u_{exact}',-1);
108
clear lineobj n1 n2 M N T c1 c2 dddt dt dx fa fef i lmd usol_EF usol_PM a
110 clear h rs_ef rs_pm stepEF stepPM stop tol
```

B.2 Code for 2-D: Heat and Perona-Malik PDE

Listing 6: For computing the PSNR – **compute_psnr.m**

```
1  function [psnr] = compute_psnr(A,B,max_i)
   %
3  % function for computing the Peak Signal-to-Noise Ratio (PSNR)
   %
5
   M = size(A,1);
7  N = size(A,2);
   C = A - B;
9
   % Computing the Mean Square Error %
11 mse = (1/(M*N))*sum(sum(C.*C));

13 % Compute the PSNR %
   psnr = 20*log10(max_i/sqrt(mse));
15
   clear M N C mse
```

Listing 7: The 2-D Perona-Malik PDE – **PeronaMalik2D.m**

```
function [B,totstep,Tend,psnr1,psnr2] = PeronaMalik2D(A,c,dt,steps,AA)
%
% 2-D Perona-Malik Equation (PME) using Euler Forward (EF) as Finite Difference scheme
%
% Input  : A      -> Matrix containing the noisy gray-scaled values of the image.
%          c      -> constant 'epsilon' in the Perona-Malik equation.
%          dt     -> A fictive timestep
%          steps  -> The number of iteration steps to be done
%          AA     -> Matrix containing the gray-scaled values of the original image.
%
% Output : B      -> The numerical solution B, where the PME is applied to denoise
%                    the noisy image.
%
% See also HEAT2D
%

tic

M  = size(A,2);                 % the number of columns in the matrix
N  = size(A,1);                 % the number of rows in the matrix
dx = 1/M; r = (c*dt)/(dx^2);    % Courant number for the x-direction
dy = 1/N; s = (c*dt)/(dy^2);    % Courant number for the y-direction


% Set initial condition
B = A; C = A;

totstep = 0;                    % total number of iteration steps
tt1     = clock;
NN      = (steps*N*M)/100;
TT      = 0;

psnr1 = zeros(steps,1);
psnr2 = zeros(steps,1);
AAA   = zeros(N,M);

for i=1:M
    for j=N
        AAA(j,i) =  double(((AA(j,i,1)*0.30)/255.0) + ((AA(j,i,2)*0.59)/255.0) + ...
                           ((AA(j,i,3)*0.11)/255.0));
    end
end

for k=1:steps
    for j=1:N
        for i=1:M
            C(j,i) = B(j,i) + ...
                     dt*(pm_x(B,i,j,dx,dy,c,M,N) + ...
                         pm_y(B,i,j,dx,dy,c,M,N));
            TT = TT + 1;
        end
    end

    B = C;

    psnr1(k) = compute_psnr(A,B,1);            % compute psnr compared to noisy image
    psnr2(k) = compute_psnr(AAA,B,1);          % compute psnr compared to original image

    % Setting the boundary conditions
    B(:,1)=A(:,1); B(1,:)=A(1,:); B(:,M)=A(:,M); B(N,:)=A(N,:);

    totstep = totstep +1;
    tt1 = clock;
end

rearange = 0

```

```
if rearange == 1

% Setting all C(j,i) in the range [0,1]
for j=1:M
    for i=1:N
        if C(j,i) < 0
            C(j,i) = 0;
        elseif C(j,i) > 1
            C(j,i) = 1;
        end
    end
end

end

toc
toc_PM = toc

totstep
Tend    = totstep*dt

makefigures = 1;

if makefigures == 1
    figure(117);
      subplot(2,1,1); imagesc(A);colormap(gray);
      title('Orig + Noise'); axis square; axis on;
      subplot(2,1,2); imagesc(C);colormap(gray);
      title('PM'); axis square; axis on;
end

% Clearing input variables (memory).
clear A c dt steps AA

% Clearing other variables     (memory).
clear AAA C M N NN TT dx dy mm tt1 toc_PM makefigures rearange  i j k r s h w


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%    Built-in functions    %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_x] = pm_x(A,i,j,dx,dy,c,w,h)
%
% u1 = A(j,i-1)
% u2 = A(j,i+1)
% u3 = A(j-1,i)
% u4 = A(j+1,i)
% u5 = A(j,i)
% u6 = A(j-1,i-1)
% u7 = A(j+1,i+1)
%

dx2 = dx^2;
dy2 = dy^2;
t   = (c/16.0)*(dx2/dy2);

%T1  = A(j,i+1) - A(j,i);
%T2  = A(j,i)   - A(j,i-1);
%v1  = A(j+1,i+1) + A(j+1,i)   - A(j-1,i+1) - A(j-1,i);
%v2  = A(j+1,i)   + A(j+1,i-1) - A(j-1,i)   - A(j-1,i-1);
%N1  = dx2 + c*(T1^2)+ t*(v1^2);
%N2  = dx2 + c*(T2^2)+ t*(v2^2);


```

```
[u1,u2,u3,u4,u5,u6,u7,u8,u9] = pm_A(A,i,j,w,h);

T1  = u6 - u5;
T2  = u5 - u4;
v1  = u3 + u2 - u9 - u8;
v2  = u2 + u1 - u8 - u7;
N1  = dx2 + (c*(T1^2)) + (t*(v1^2));
N2  = dx2 + (c*(T2^2)) + (t*(v2^2));
u_x = (T1/N1) - (T2/N2);

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u_y] = pm_y(A,i,j,dx,dy,c,w,h)
%
% u1 = A(j,i-1)
% u2 = A(j,i+1)
% u3 = A(j-1,i)
% u4 = A(j+1,i)
% u5 = A(j,i)

dx2 = dx^2;
dy2 = dy^2;
t   = (c/16.0)*(dy2/dx2);

%T1  = A(j+1,i) - A(j,i);
%T2  = A(j,i)   - A(j-1,i);
%v1  = A(j+1,i+1) + A(j,i+1)   - A(j+1,i-1) - A(j,i-1);
%v2  = A(j,i+1)   + A(j-1,i+1) - A(j,i-1)   - A(j-1,i-1);
%N1  = dy2 + c*(T1^2)+ t*(v1^2);
%N2  = dy2 + c*(T2^2)+ t*(v2^2);

[u1,u2,u3,u4,u5,u6,u7,u8,u9] = pm_A(A,i,j,w,h);

T1  = u2 - u5;
T2  = u5 - u8;
v1  = u3 + u6 - u1 - u4;
v2  = u6 + u9 - u4 - u7;
N1  = dy2 + (c*(T1^2)) + (t*(v1^2));
N2  = dy2 + (c*(T2^2)) + (t*(v2^2));

u_y = (T1/N1) - (T2/N2);

%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

function [u1,u2,u3,u4,u5,u6,u7,u8,u9] = pm_A(A,i,j,w,h)
%
%  Input
% - w : width(A)  using index i
% - h : height(A) using index j
%
% u1 = A(j+1,i-1);    u2 = A(j+1,i);    u3 = A(j+1,i+1);
% u4 = A(j  ,i-1);    u5 = A(j  ,i);    u6 = A(j  ,i+1);
% u7 = A(j-1,i-1);    u8 = A(j-1,i);    u9 = A(j-1,i+1);

if      i==1 && j==1                                      % u1
     u1 = 0;              u2 = A(j+1,i); u3 = A(j+1,i+1);
     u4 = 0;              u5 = A(j  ,i); u6 = A(j  ,i+1);
     u7 = 0;              u8 = 0;        u9 = 0;

elseif  (i>1 && i<w) && j==1                              % u2
     u1 = A(j+1,i-1); u2 = A(j+1,i); u3 = A(j+1,i+1);
     u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = A(j  ,i+1);
     u7 = 0;          u8 = 0;        u9 = 0;

```

```
elseif i==w && j==1                                              % u3
    u1 = A(j+1,i-1); u2 = A(j+1,i); u3 = 0;
    u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = 0;
    u7 = 0;          u8 = 0;        u9 = 0;

elseif i==1 && (j>1 && j<h)                                      % u4
    u1 = 0;          u2 = A(j+1,i); u3 = A(j+1,i+1);
    u4 = 0;          u5 = A(j  ,i); u6 = A(j  ,i+1);
    u7 = 0;          u8 = A(j-1,i); u9 = A(j-1,i+1);

elseif (i>1 && i<w) && (j>1 && j<h)                              % u5
    u1 = A(j+1,i-1); u2 = A(j+1,i); u3 = A(j+1,i+1);
    u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = A(j  ,i+1);
    u7 = A(j-1,i-1); u8 = A(j-1,i); u9 = A(j-1,i+1);

elseif i==w && (j>1 && j<h)                                      % u6
    u1 = A(j+1,i-1); u2 = A(j+1,i); u3 = 0;
    u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = 0;
    u7 = A(j-1,i-1); u8 = A(j-1,i); u9 = 0;

elseif i==1 && j==h                                              % u7
    u1 = 0;          u2 = 0;        u3 = 0;
    u4 = 0;          u5 = A(j  ,i); u6 = A(j  ,i+1);
    u7 = 0;          u8 = A(j-1,i); u9 = A(j-1,i+1);

elseif (i>1 && i<w) && j==h                                      % u8
    u1 = 0;          u2 = 0;        u3 = 0;
    u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = A(j  ,i+1);
    u7 = A(j-1,i-1); u8 = A(j-1,i); u9 = A(j-1,i+1);

elseif i==w && j==h                                              % u9
    u1 = 0;          u2 = 0;        u3 = 0;
    u4 = A(j  ,i-1); u5 = A(j  ,i); u6 = 0;
    u7 = A(j-1,i-1); u8 = A(j-1,i); u9 = 0;
end
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Caution

To make the code readable we choose to change the syntax of some MATLAB functions at some places in the code. Spaces in the text, as visible in the *error* message on line 24 of Listing 4, have a strange notation. For example instead of using *title('Test Message')* we used *title("Test Message")*. So beware for all functions where quotes are used " " , such as :

- xlabel(" ... ") or ylabel(" ... ")
- text(x y, sprintf(" ... "))
- title(" ... ")
- error(" ... ")

Consult the MATLAB help function for the correct syntax if the code crashes and you get the following message

Error: Missing variable or function.

Listing 8: The 2-D Heat PDE – **Heat2D.m**

```
function [B,totstep,Tend,psnr1,psnr2] = Heat2D2_test(A,c,dt,steps,AA)
%
%  2-D Heat Equation(HE) using Euler Forward (EF) as Finite Difference scheme
%
% Input  : A      -> A matrix containing the grayscaled values of an image.
%          c      -> constant 'c' in the heat equation.
%          dt     -> A fictive timestep
%          steps  -> The number of iteration steps to be done
%          AA     -> Matrix containing the gray-scaled values of the original image.
%
% Output : B      -> The numerical solution B, where the heat equation is
%                    applied for denoising the noisy image with noise.
%
% See also PERONAMALIK2D
%

tic

M  = size(A,2);                 % the number of columns in the matrix
N  = size(A,1);                 % the number of rows in the matrix
T  = 0.0004;
c1 = 1;
c2 = 0.02;
dx = 1/M;
dy = 1/N;
r = (c*dt)/(dx^2);              % Courant number for the x-direction
s = (c*dt)/(dy^2);              % Courant number for the y-direction

% Set initial condition
B = A; C = A;


% variables for psnr computing!!

psnr1 = zeros(steps,1);
psnr2 = zeros(steps,1);

AAA = zeros(N,M);

for i=1:M
    for j=N
        AAA(j,i) =  double(((AA(j,i,1)*0.30)/255.0) + ((AA(j,i,2)*0.59)/255.0) + ...
                           ((AA(j,i,3)*0.11)/255.0));
    end
end


totstep = 0;

for k=1:steps
    for j=1:N
        for i=1:M
            if      i == 1  && j==1
                C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                         r*(0         + B(j,i+1)) + ...
                         s*(0         + B(j+1,i));

            elseif i == 1  && (j>1  && j<N)
                C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                         r*(0         + B(j,i+1)) + ...
                         s*(B(j-1,i) + B(j+1,i));

            elseif i == 1 && j==N
                C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                         r*(0         + B(j,i+1)) + ...
                         s*(B(j-1,i) + 0        );
```

```
68              elseif (i>1  && i<M) && j == 1
                    C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
70                           r*(B(j,i-1) + B(j,i+1)) + ...
                             s*(0        + B(j+1,i));
72
                elseif (i>1  && i<M) && j == N
74                  C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                             r*(B(j,i-1) + B(j,i+1)) + ...
76                           s*(B(j-1,i) + 0        );

78              elseif i == M && j== N
                    C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
80                           r*(B(j,i-1) + 0        ) + ...
                             s*(B(j-1,i) + 0        );
82
                elseif i == M && j== 1
84                  C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                             r*(B(j,i-1) + 0        ) + ...
86                           s*(0        + B(j+1,i));

88              elseif i == M  && (j>1  && j<N)
                    C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
90                           r*(B(j,i-1) + 0        ) + ...
                             s*(B(j-1,i) + B(j+1,i));
92
                elseif (i > 1 && i < M) && (j > 1 && j < N)
94                  C(j,i) = (1 - 2*r - 2*s)*B(j,i)  + ...
                             r*(B(j,i-1) + B(j,i+1)) + ...
96                           s*(B(j-1,i) + B(j+1,i));
                end
98          end
        end
100
        B = C;
102
        % Set boundary conditions
104     B(:,1) = A(:,1);  B(1,:) = A(1,:); B(:,M) = A(:,M); B(N,:) = A(N,:);

106     psnr1(k) = compute_psnr(A,B,1);
        psnr2(k) = compute_psnr(AAA,B,1);
108     totstep  = totstep +1;
    end
110
    toc
112 toc_EF = toc
    Tend   = totstep*dt
114
    makefigures = 0;
116
    if makefigures == 1
118     figure(117);
          subplot(2,1,1); imagesc(A);colormap(gray); title('Orig');
120          axis square; axis on;
          subplot(2,1,2); imagesc(C);colormap(gray); title('Heat');
122          axis square; axis on;
    end
124
    % Clearing the input variables (memory).
126  clear A c dt steps AA

128 % Clearing other variables      (memory).
     clear dx dt r s i j  c1 c2 dy k M N T makefigures toc_EF AAA C
```