Numerical resolution of inpainting methods

Carmen Mayora Cebollero

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Directores del trabajo:
Francisco José Gaspar Lorenz
Carmen Rodrigo Cardiel

Julio 2020

Prologue

In this Undergraduate Dissertation we introduce the concept of inpainting, explaining three inpainting methods that are used to restore grey value images and another one used to restore binary images. This work is structured in the following way:

- A summary (written in Spanish).
- Contents.
- Chapter 1. *Introduction.* We introduce the concept of inpainting.
- Chapter 2. *The good continuation principle.* We explain how to solve the issue of the lack of unicity in the inpainting problem, focusing on the historical knowledge and on the application of Gestalt theory to our context (amodal completion and the good continuation principle).
- Chapter 3. *Inpainting methods based on second-order equations.* We present two inpainting methods governed by second-order equations. We explain their numerical resolutions and we implement them in MATLAB. We prove that they do not fulfill the good continuation principle, therefore they are not good enough.
- Chapter 4. *Inpainting methods based on the Cahn-Hilliard equation.* We present an inpainting method based on a variation of the Cahn-Hilliard equation. It only works for binary images, but we can extend it to obtain a method for grey value images (TV-H^{-1} inpainting). We explain the numerical resolution of both methods and we implement them in MATLAB.
- Appendix A. *Mathematical facts.* We sketch out some mathematical results that are used along the work.
- Appendix B. *Discretization of the Bilaplacian.* We explain the discretization of the Bilaplacian.
- Appendix C. *Numerical resolution with MATLAB.* MATLAB programs used to obtain the inpainted images for each inpainting method can be found here.
- Bibliography.

Resumen

Hoy en día una de las principales fuentes de información es la imagen digital. Esta se obtiene mediante el muestreo y cuantificación de una representación analógica de la realidad. Es decir, superponiendo una malla regular sobre la imagen analógica y asignando a cada uno de sus elementos (píxel) un valor. Esto nos permite interpretar una imagen digital como una matriz y asociarla a una función $f : \mathbb{R}^2 \to \mathbb{R}^i$, siendo $i = 1$ si consideramos imágenes en blanco y negro (escala de grises) e $i = 3$ si son en color (en este trabajo estudiaremos el caso $i = 1$). Podemos realizar operaciones con f, conocidas como técnicas de procesamiento de imágenes. Las aplicaremos a la restauración de imágenes dañadas o *inpainting*.

La restauración de imágenes dañadas es un tipo de interpolación que sirve para sustituir las partes estropeadas de una imagen digital usando la información obtenida de las zonas intactas de la misma. La formulación matemática del problema es la siguiente:

Sea f una imagen dañada definida en el dominio imagen $\Omega \subseteq \mathbb{R}^2$. Denotemos por $\mathcal{D} \subseteq \Omega$ a la zona estropeada, que llamaremos dominio de inpainting*, agujero o hueco. El objetivo es encontrar una imagen u, similar a la original (sin desperfectos), conocida como imagen restaurada. Normalmente u se obtiene resolviendo un problema de contorno.*

El término *inpainting* fue acuñado por los restauradores de arte. Para diseñar métodos de *inpainting* hay que tener en cuenta obstáculos que presenta el proceso de restauración como son la inexistencia de solución única o la imposibilidad de reconstruir estructura y textura al mismo tiempo (los métodos presentados en este trabajo únicamente reconstruyen la estructura).

El problema relacionado con la no unicidad de solución puede resolverse haciendo uso del conocimiento histórico (proporcionado por el contexto de la imagen) y de la teoría de la Gestalt (teoría que estudia las leyes que sigue nuestro cerebro para hacer la reconstrucción visual de una imagen dañada) aplicada al procesamiento de imágenes. Las leyes proporcionadas por la teoría de la Gestalt son conocidas en el campo del procesamiento de imágenes como leyes de agrupación. La más importante de todas ellas en la restauración de imágenes es la compleción amodal, que consiste en la capacidad de reconstruir un objeto a pesar de que esté oculto parcialmente por otros. La base de esta ley de agrupación es el principio de buena continuación, que da prioridad a la conexión de estructuras y a la preservación de su curvatura.

Los métodos de restauración locales y estructurales son aquellos que utilizan únicamente la información estructural de las zonas de alrededor del agujero para cubrirlo. Por eso, la EDP que nos permite obtener la imagen restaurada únicamente se resuelve en el dominio de *inpainting* $\mathcal{D}$, considerando que tras los procesos de restauración el resto de la imagen es idéntico a la imagen dañada. A partir de ahora, a menos que se especifique lo contrario, al hablar de imágenes nos referiremos a imágenes en escala de grises.

Consideraremos aquellos operadores de interpolación tales que dada una curva (borde de la zona estropeada $\mathcal{D}$) y una función definida en ella, les asocian una función definida en el dominio de *inpainting* (que será la imagen restaurada). En un principio buscamos operadores de interpolación gobernados por EDPs de orden 2, obteniendo el método *harmonic inpainting* y *TV inpainting*.

El *harmonic inpainting* es un método de restauración de imágenes cuyo proceso de interpo-

lación se basa en calcular los valores correspondientes a la parte dañada como el promedio de los valores de los elementos de alrededor. El problema correspondiente es

$$\begin{cases} \Delta u = 0 & \text{en } \mathcal{D}, \\ u = f & \text{en } \partial\mathcal{D}, \end{cases}$$

cuya solución numérica puede obtenerse aplicando el método de diferencias finitas.

La calidad de este método puede estudiarse descomponiendo la imagen original (sin dañar) como suma de su parte armónica y de su parte inarmónica en un punto que corresponderá al dominio dañado (cuando la imagen ya haya sufrido el proceso de degradación). Identificando la parte armónica con la solución débil del problema anterior, puede demostrarse que cuando el dominio de *inpainting* es relativamente grande este método no cumple el principio de buena continuación. Por tanto, no puede considerarse que proporciona buenos resultados.

El *TV inpainting* es un método de restauración de imágenes cuyo proceso de interpolación se basa en calcular el valor de los píxeles dañados de la imagen como la mediana de los valores de los elementos de alrededor. El problema correspondiente es

$$\begin{cases} \nabla \cdot \left(\dfrac{\nabla u}{\|\nabla u\|} \right) = 0 & \text{en } \mathcal{D}, \\ u = f & \text{en } \partial\mathcal{D}, \end{cases}$$

cuya solución numérica puede obtenerse sustituyendo las derivadas del problema de evolución correspondiente por fórmulas de diferencias finitas.

El principal problema de este método es que su solución puede no ser única. Además, puede demostrarse, reduciendo el problema a una sola línea de nivel, que la interpolación es lineal, por lo que no se conserva la curvatura y no se cumple el principio de buena continuación. En este caso tampoco podemos considerar que este método nos devuelva una imagen restaurada correctamente.

Hemos observado que realizar la restauración de imágenes con ecuaciones de segundo orden no nos proporciona buenos resultados. Para mejorarlos debemos tener en cuenta el gradiente de la imagen en la frontera del dominio de *inpainting*. Por este motivo utilizaremos métodos gobernados por ecuaciones de cuarto orden.

Una imagen binaria es aquella cuyos píxeles solo pueden tomar dos valores, por simplicidad escogeremos el blanco y el negro. El *Cahn-Hilliard inpainting* para imágenes binarias es un método de restauración basado en una modificación de la ecuación de Cahn-Hilliard, que es de cuarto orden. Puede parecer un método restrictivo en el sentido de que únicamente puede ser utilizado para imágenes binarias, sin embargo, si es usado para reconstruir la parte estructural de la imagen y se combina con otros métodos de *inpainting* puede dar muy buenos resultados. La imagen restaurada será la solución estacionaria de

$$\begin{cases} u_t = \Delta \left(-\epsilon \Delta u + \dfrac{1}{\epsilon} F'(u) \right) & \text{en } \mathcal{D}, \\ u = f & \text{en } \partial\mathcal{D}, \\ \nabla u \cdot \vec{n} = \nabla f \cdot \vec{n} & \text{en } \partial\mathcal{D}, \end{cases}$$

donde $F(u) = u^2(u-1)^2$ y $\epsilon > 0$. Para la resolución numérica de este problema aplicamos el método *convexity splitting* para ecuaciones de evolución, obteniendo esquemas de paso en tiempo que podemos discretizar con el uso de fórmulas de diferencias finitas.

Podemos extender el *Cahn-Hilliard inpainting* para obtener un método de *inpainting* que dé buenos resultados para la restauración de imágenes en escala de grises. Este método es conocido como $TV - H^{-1}$ *inpainting* y la imagen restaurada es la solución estacionaria de

$$\begin{cases} u_t &= \Delta p & \text{en } \mathcal{D}, \\ u &= f & \text{en } \partial\mathcal{D}, \\ \nabla u \cdot \vec{n} &= \nabla f \cdot \vec{n} & \text{en } \partial\mathcal{D}, \end{cases}$$

donde $p = -\nabla \cdot \left(\frac{\nabla u}{\|\nabla u\|_\delta} \right) \in \partial \mathrm{TV}_\infty(u) = \left\{ \hat{p} \in \mathcal{L}^2(\mathcal{D}) \mid \langle v - u, \hat{p} \rangle \leq \mathrm{TV}_\infty(v) - \mathrm{TV}_\infty(u),\ \forall v \in \mathcal{L}^2(\mathcal{D}) \right\}$, siendo $\|\nabla u\|_\delta = \sqrt{\|\nabla u\|^2 + \delta}$, $0 < \delta \ll 1$. Su solución numérica puede obtenerse, al igual que en el método anterior, aplicando el método de *convexity splitting* para ecuaciones de evolución, obteniéndose esquemas de paso en tiempo que podemos discretizar con el uso de fórmulas de diferencias finitas.

Contents

Chapter 1

Introduction

Nowadays, digital images are one of the main sources of information. A digital image is obtained from an analogue representation of the continuous world by sampling and quantization: a regular grid is superimposed on the analogue image and a value is assigned to each grid element. These elements are called pixels (*pict*ure *el*ements). If it is a black and white image, the values of the pixels (grey values) are scalars in the interval $[0, 255]$ where 0 represents black and 255, white (these values can also be scaled to $[0, 1]$). If it is a colour image, the values (colour values) are represented by vectors (r, g, b) (r means red, g is green and b corresponds to blue) where $r, g, b \in [0, 255]$ ($r, g, b \in [0, 1]$ if the values are scaled). For a digital image the typical size goes from 2000×2000 pixels (if the photo is taken using a simple digital camera) to 10000×10000 pixels (if a high-resolution camera is used to take the image).

Since we have reinterpreted the digital image as a regular grid (a matrix) we can treat it as a function $f : \mathbb{R}^2 \to \mathbb{R}^i$ where $i = 1$ if we work with grey value images (black and white images) and $i = 3$ if it is a colour image (in this work we study the case $i = 1$). In some contexts the domain of f can be $\mathbb{R}^3$ or even $\mathbb{R}^4$. We can make operations with f, that is to say, we can apply image processing techniques. In this work we apply them to image inpainting.

Image inpainting is a type of interpolation used for the restoration of virtual images: the damaged parts of the picture are filled using the information obtained from the intact parts. This is a process similar to the one used by art restorers who utilize the information around the damaged part to reconstruct it. Some applications of image inpainting are digital restoration of old paintings for conservation purposes, text erasing from advertising photographs and removal of wires needed to create special effects.

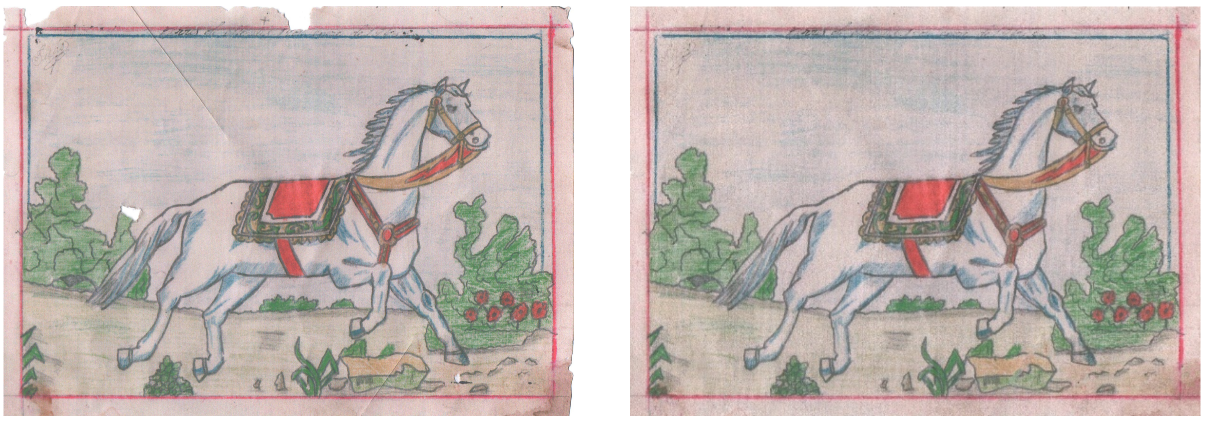

Figure 1.1: On the left-hand side we have a picture of 1943 that went through a degradation process and on the right-hand side we have inverted this process using inpainting methods.

On the left-hand side of Figure 1.1 we can see a draw which was damaged by time. Using inpainting methods we have reconstructed it, obtaining the image on the right-hand side of Figure 1.1 (for the reconstruction we have used [1]).

Mathematically, we have a function f defined on $\Omega \subseteq \mathbb{R}^2$ (known as image domain) which represents the damaged image and our goal is to reconstruct it over the damaged domain $\mathcal{D} \subseteq \Omega$ (which is called inpainting domain, hole or gap) to obtain a picture u similar to the original one (u is known as inpainted image). In the past, image inpainting was made manually, today it is automated thanks to mathematical techniques.

The term inpainting was invented by art restoration workers. In the framework of digital image restoration this word appeared for the first time in the work of Bertalmio et al. [2]. With the objective of imitating the work of art restorers they explain an inpainting model based on the resolution of a discrete approximation of the PDE $u_t = \nabla^{\perp} u \cdot \nabla\Delta\, u$ in the gap $\mathcal{D}$ taking into account the information of its boundary $\partial\mathcal{D}$ (in the previous equation $\nabla^{\perp} u$ represents $(-u_y, u_x)$). The idea behind this method follows the principle of prolonging the image intensity in the direction of the level lines which go inside the damaged part. In order to avoid the crossing of these level lines, a non-linear diffusion term can be added, so the equation is $u_t = \nabla^{\perp} u \cdot \nabla\Delta u + \nu\nabla \cdot (d(\|\nabla u\|)\nabla u)$, where $d(s)$ is the diffusion coefficient and $0 < \nu \ll 1$. In this inpainting process the user only takes part to select the inpainting domain.

Although PDEs are very common in inpainting methods, sometimes they cannot be used. For instance, if we have a repetitive image, we have to apply the exemplar-based inpainting which is based on the replacement of the missing element by a copy of another one equal to it that is not damaged.

To design inpainting methods we have to take into account some obstacles that we can find in the process of restoration:

- Unicity problem: the damaged parts of our image represent areas where the information is irrecoverable. If we do not take into account some additional knowledge (for example, historical facts) the solution of the inpainting problem may not be unique.

- Problem of the automation of the process: inpainting mathematical methods are designed with the objective of avoid the supervision of the user. This implies that they have to simulate the answer that the human brain would give.

- Texture-structure problem: one of the biggest problems is the inability of the methods to reconstruct structure and texture at the same time. It is necessary to differentiate between texture inpainting and non-texture or geometrical/structural inpainting. In this work we study the last one (also known as low-level inpainting).

- Size problem: the bigger the hole is, the harder the problem becomes. If the gap is small, it is enough to use lower-order inpainting methods. However, if it is bigger, it is necessary to use non-linear PDEs of higher order.

In this work we study four inpainting methods. They are local structural inpainting methods, that is to say, only local structural information is used to fill the missing parts of the damaged image. All of them are governed by PDEs that we solve numerically. We present examples of all of them.

In Chapter 3 we explain two inpainting methods which are based on second-order equations. According to the good continuation principle (which is explained in Chapter 2) they are not good enough. In Chapter 4 we present two inpainting methods which are based on the Cahn-Hilliard equation. It is a fourth-order equation, so we obtain better results.

For more information about these four methods and to learn about other inpainting methods we refer to the reader to [3].

Chapter 2

The good continuation principle

In the work of Bertalmio et al. [2] some conservators at the Minneapolis Institute of Arts were asked about their job. They asserted that *inpainting is a very subjective procedure (...) There is not such thing as "the" way to solve the problem.* The process depends on the object and on the professional. The same happens with image inpainting: it does not have unique solution. We have two main strategies which help us to determine it: historical knowledge and visual perception.

2.1 Historical knowledge

Historical knowledge can help us to reconstruct parts of the damaged region using information provided by the image context. So, this strategy depends on the image: if we work with an old painting (art restoration), it is useful to know information about the techniques and the materials used by the painter; but if we work with a medical image, it is more useful to have knowledge about anatomy.

For example, this strategy is very useful in the continuation of roads in aerial images. Let us imagine that we have an aerial image of a street which has trees on both sides of the road (the tops of the trees hide part of the street). We have to reconstruct the road to have the possibility of following it in the aerial image. Our historical knowledge in this case is a geometric knowledge: we know how the shape of a road is. This information helps us to obtain a unique solution.

2.2 Visual perception. Gestalt theory

Visual perception is a context-free continuation. It is a human ability which consists in interpolating broken or occluded structures automatically.

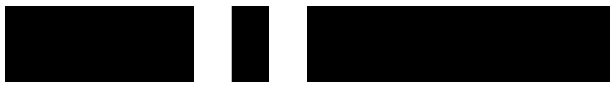

Figure 2.1: One straight line. Two straight lines. Which is the right solution?

Let us observe the image in Figure 2.1. It is a damaged image, but we do not know if before the damage it was a unique straight line or there were two straight lines. The solution depends on the way of perception and on the experience of the viewer. We need a theory that studies all the possible solutions, assigns probabilities to each of them and gives us a unique solution. That is to say, a theory which works in a similar way to our natural perception is needed. It is Gestalt theory.

Gestalt is a german word that can be translated to English as shape. Gestalt theory studies the laws followed by our brain to do the visual reconstruction of a damaged image. Gestaltists (scientists who work on this theory) interprete shapes as clouds of points with different hues of grey and they study the laws followed by our perception mechanism to obtain continuous shapes (called *gestalten*) reconstructing these collections of points.

Those clouds of points can be identified with the pixels of an image. Therefore, gestalt laws can be formalized to apply them to our case. We can find the explanation of these concepts applied to digital image processing in the work of Kanizsa [4].

Considering a group as a set of points with the same characteristics, the principal gestalt laws needed in image processing are called grouping laws. The elementary grouping laws that we can find in Kanizsa's work are:

- Vicinity: perception tends to group near objects.
- Similarity: similar shapes tend to be grouped.
- Continuity of direction: objects which follow the same direction tend to be interpreted as a unique one.
- Amodal completion: it consists in the capability of reconstructing an entire object despite the fact that some of its parts are occluded by other objects.
- Closure: perception tends to close shapes which seem to be interrupted.
- Constant width: objects with the same width tend to be grouped.
- Tendency to convexity: visual perception prefers to group points in convex shapes rather than in concave ones.
- Symmetry: visual perception tends to recognize symmetry and to understand symmetric shapes as a group.
- Common motion: we tend to group objects which seem to move together.
- Past experience: shapes can be recognized if something similar has been seen before.

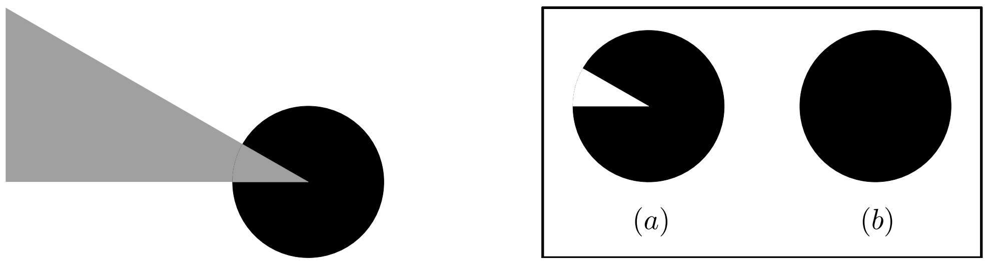

Figure 2.2: What about the black piece? The good continuation principle gives us the right solution: (b).

When we study these laws in a concrete image, we can have an ambiguous situation because each law may group the objects in a different way. In image inpainting we consider that amodal completion is the strongest grouping law and we choose its configuration as the correct one.

The basis of amodal completion is the good continuation principle. It prefers connection over disconnection, that means, if an object seems to be partially occluded or broken, according to this principle it continues under the other objects that hide it. This law also considers that objects are extended preserving their curvature.

In Figure 2.2 we can see an example of how the good continuation principle acts. On the left picture we can see an image, we deduce that it is formed by two different shapes, but which is the correct solution for the black piece? Is it (a)? Is it (b)? According to the good continuation principle the correct one is (b) because in (a) the curvature is not preserved and disconnection is preferred over connection.

The good continuation principle has influenced many inpainting methods, so we use it as a criterion to determine if one method is good enough.

Chapter 3

Inpainting methods based on second-order equations

In the following, when we define an inpainting method we denote the damaged image by f. It is defined on $\Omega \subseteq \mathbb{R}^2$ and the damaged part is the inpainting domain $\mathcal{D}$, which fulfills that $\mathcal{D} \subseteq \Omega$ and $\partial\mathcal{D} \cap \partial\Omega = \emptyset$. The inpainted image that we obtain when we have covered the gap is denoted by u. In this work, the PDEs of the inpainting methods are solved in $\mathcal{D}$, obtaining a solution u defined on $\mathcal{D}$ that can be extended to Ω considering that in $\Omega \setminus \mathcal{D}$ the inpainted image u coincides with the damaged one ($\Omega \setminus \mathcal{D}$ is the part of the image which was intact).

Notation 3.1. Let $u : \mathcal{D} \subseteq \mathbb{R}^2 \to \mathbb{R}$. We denote by D^2u the Hessian of u.

Notation 3.2. Let $A \in \mathbb{R}^{2\times 2}$ and $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^2$. Let us denote the quadratic form $\boldsymbol{v}^t A \boldsymbol{w}$ by

$$A(\boldsymbol{v}, \boldsymbol{w}) := \boldsymbol{v}^t A \boldsymbol{w} = \sum_{i,j} A_{ij} \boldsymbol{v}_i \boldsymbol{w}_j .$$

Example 3.1. We apply Notation 3.2 to the matrix $A = D^2u$ and to the vectors $\nabla u = (u_x, u_y)^t$ and $\nabla^\perp u = (-u_y, u_x)^t$, then

- $D^2u\left(\frac{\nabla u}{\|\nabla u\|}, \frac{\nabla u}{\|\nabla u\|}\right) = \frac{1}{\|\nabla u\|^2}\left(u_{xx}u_x^2 + 2u_{xy}u_xu_y + u_{yy}u_y^2\right).$
- $D^2u\left(\frac{\nabla^\perp u}{\|\nabla u\|}, \frac{\nabla^\perp u}{\|\nabla u\|}\right) = \frac{1}{\|\nabla u\|^2}\left(u_{xx}u_y^2 - 2u_{xy}u_xu_y + u_{yy}u_x^2\right).$
- $D^2u\left(\frac{\nabla^\perp u}{\|\nabla u\|}, \frac{\nabla u}{\|\nabla u\|}\right) = D^2u\left(\frac{\nabla u}{\|\nabla u\|}, \frac{\nabla^\perp u}{\|\nabla u\|}\right) = \frac{1}{\|\nabla u\|^2}\left(u_xu_y(u_{yy} - u_{xx}) + u_{xy}(u_x^2 - u_y^2)\right).$

Notation 3.3. We denote by $\mathcal{C}$ the set of continuous and simple Jordan curves, that is to say,

$$\mathcal{C} = \{\Gamma \mid \Gamma \text{ is a continuous and simple Jordan curve}\}.$$

Notation 3.4. The set of continuous functions defined on Γ is

$$\mathcal{F}(\Gamma) = \{f : \Gamma \to \mathbb{R} \mid f \text{ is continuous}\}.$$

Definition 3.1. Interpolation operator.

An interpolation operator (also called inpainting operator) is the map

$$\begin{aligned} E : \mathcal{F}(\Gamma) \times \mathcal{C} &\longrightarrow \{\text{Functions on the inpainting domain } \mathcal{D}(\Gamma)\} \\ (f, \Gamma) &\longmapsto E(f, \Gamma) \end{aligned}$$

where $E(f, \Gamma)$ is called interpolant and $\mathcal{D}(\Gamma)$ denotes the inpainting domain $\mathcal{D}$ whose boundary is the continuous simple Jordan curve Γ.

Lemma 3.1. *Interpolation axioms.*
The interpolation operator in Definition 3.1 has to fulfill the following axioms:

- *Comparison Principle (monotony). Let $\Gamma \in \mathcal{C}$ and f, $g \in \mathcal{F}(\Gamma)$ such that $f \leq g$. Then,*
$$E(f,\Gamma) \leq E(g,\Gamma).$$
- *Stability Principle (the iteration of the process of extraction of the interpolant does not give us additional information). Let $\Gamma_1 \in \mathcal{C}$ and $f \in \mathcal{F}(\Gamma_1)$. For any $\Gamma_2 \in \mathcal{C}$ such that $\mathcal{D}(\Gamma_2) \subseteq \mathcal{D}(\Gamma_1)$ we have that*
$$E(E(f,\Gamma_1)|_{\Gamma_2},\Gamma_2) = E(f,\Gamma_1)|_{\mathcal{D}(\Gamma_2)}.$$
- *Regularity Principle. Let $A \in SM(2,\mathbb{R}) = \{M \in \mathbb{R}^{2\times 2} \mid M = M^t\}$, $\boldsymbol{p} \in \mathbb{R}^2\backslash 0$, $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^2$ with the scalar product $\langle \boldsymbol{v}, \boldsymbol{w}\rangle = v_1 w_1 + v_2 w_2$ and $c : \mathbb{R}^2 \to \mathbb{R}$. Following Notation 3.2, we define*
$$Q(\boldsymbol{w}) = \frac{A(\boldsymbol{w}-\boldsymbol{v}, \boldsymbol{w}-\boldsymbol{v})}{2} + \langle \boldsymbol{p}, \boldsymbol{w}-\boldsymbol{v}\rangle + c(\boldsymbol{v}).$$
Then,
$$\frac{E(Q|_{\partial B(\boldsymbol{x},r)}, \partial B(\boldsymbol{x},r))(\boldsymbol{x}) - Q(\boldsymbol{x})}{r^2/2} \to F(A,\boldsymbol{p},c\,(\cdot)\,,\boldsymbol{x}) \quad \text{as } r \to 0^+,$$
where $F : SM(2,\mathbb{R}) \times \mathbb{R}^2 \setminus 0 \times \mathbb{R} \times \mathbb{R}^2 \to \mathbb{R}$ is a continuous function and $B(\boldsymbol{x}, r)$ is the ball in $\mathbb{R}^2$ with center $\boldsymbol{x}$ and radius r.

  In the particular case of inpainting we take $A = D^2 u$, $\boldsymbol{p} = \nabla u$ and $c = u$.
- *Translation Invariance (the inpainting operator is invariant under the translation of f by a vector $\boldsymbol{h}$). Let $\Gamma \in \mathcal{C}$ and $f \in \mathcal{F}(\Gamma)$. Let us denote the translation of f by $\boldsymbol{h} \in \mathbb{R}^2$ as $\tau_{\boldsymbol{h}} f(\boldsymbol{x}) := f(\boldsymbol{x}+\boldsymbol{h})$, $\forall \boldsymbol{x} \in \Gamma$. Then,*
$$E(\tau_{\boldsymbol{h}} f, \Gamma - \boldsymbol{h}) = \tau_{\boldsymbol{h}} E(f,\Gamma).$$
- *Rotation Invariance (the inpainting operator is invariant under rotations). Let $\Gamma \in \mathcal{C}$, $f \in \mathcal{F}(\Gamma)$, $\mathcal{R} \in \mathbb{R}^{2\times 2}$ be a rotation matrix such that $\mathcal{R}f(\boldsymbol{x}) := f(\mathcal{R}^t \boldsymbol{x})$, $\forall \boldsymbol{x} \in \Gamma$, and $\mathcal{R}\Gamma \in \mathcal{C}$. Then,*
$$E(\mathcal{R}f, \mathcal{R}\Gamma) = \mathcal{R}E(f,\Gamma).$$
- *Grey Scale Shift Invariance. Let $\Gamma \in \mathcal{C}$, $f \in \mathcal{F}(\Gamma)$ and $C \in \mathbb{R}$. Then*
$$E(f + C, \Gamma) = E(f,\Gamma) + C.$$
- *Linear Grey Scale Invariance. Let $\Gamma \in \mathcal{C}$, $f \in \mathcal{F}(\Gamma)$ and $\lambda \in \mathbb{R}$. Then*
$$E(\lambda f, \Gamma) = \lambda E(f,\Gamma).$$
- *Zoom Invariance (invariance under zooming). Let $\Gamma \in \mathcal{C}$, $f \in \mathcal{F}(\Gamma)$ and $\delta_\lambda f(\boldsymbol{x}) := f(\lambda \boldsymbol{x})$, $\forall \boldsymbol{x} \in \Gamma$, with $\lambda > 0$. Then*
$$E(\delta_\lambda f, \lambda^{-1}\Gamma) = \delta_\lambda E(f,\Gamma).$$

Theorem 3.1. *Let E be an interpolation operator which satisfies the interpolation axioms described in Lemma 3.1. Let f be a continuous function that is defined on $\partial\mathcal{D}$ (the boundary of the inpainting domain $\mathcal{D}$). Let $u = E(f, \partial\mathcal{D})$ be the interpolant in $\mathcal{D}$.*

Then, the continuous function defined on $\mathcal{D}$ that is denoted by u is a solution of the problem

$$\left\{\begin{array}{rcll} G(A) & = & 0 & \text{in } \mathcal{D}, \\ u & = & f & \text{on } \partial\mathcal{D}, \end{array}\right. \tag{3.1}$$

where $G(A)$ is a non-decreasing function that satisfies $G(\lambda A) = \lambda G(A)$, $\forall\lambda \in \mathbb{R}$, and $A = D^2u$.

Proposition 3.1. *Let us consider the assumptions of Theorem 3.1 and let us add that the function G is differentiable at the origin.*

Then, problem (3.1) can be written as

$$\left\{\begin{array}{rcll} \alpha D^2u\left(\frac{\nabla u}{\|\nabla u\|}, \frac{\nabla u}{\|\nabla u\|}\right) + 2\beta D^2u\left(\frac{\nabla u}{\|\nabla u\|}, \frac{\nabla^\perp u}{\|\nabla u\|}\right) + \gamma D^2u\left(\frac{\nabla^\perp u}{\|\nabla u\|}, \frac{\nabla^\perp u}{\|\nabla u\|}\right) & = & 0 & \text{in } \mathcal{D}, \\ u & = & f & \text{on } \partial\mathcal{D}, \end{array}\right. \tag{3.2}$$

where $\alpha, \gamma \geq 0$ and $\alpha\gamma - \beta^2 \geq 0$.

Corollary 3.1. *Applying Example 3.1 to the previous problem (3.2), we can rewrite it as*

$$\left\{\begin{array}{rcll} \frac{1}{\|\nabla u\|^2}((\alpha u_x^2 - 2\beta u_x u_y + \gamma u_y^2)u_{xx} + (\alpha u_y^2 + 2\beta u_x u_y + \gamma u_x^2)u_{yy} + & & & \\ +(2u_x u_y(\alpha - \gamma) + 2\beta(u_x^2 - u_y^2))u_{xy}) & = & 0 & \text{in } \mathcal{D}, \\ u & = & f & \text{on } \partial\mathcal{D}, \end{array}\right. \tag{3.3}$$

where $\alpha, \gamma \geq 0$ and $\alpha\gamma - \beta^2 \geq 0$.

Now we choose possible values for α, β and γ in (3.3) to obtain inpainting methods. We present two examples: harmonic inpainting and TV inpainting. Harmonic inpainting is explained because of its simplicity and TV inpainting, because the concept of TV introduced to study it is useful to understand a more complex method that we will introduce in the next chapter.

3.1 Harmonic inpainting

Let us take $\beta = 0$ and $\alpha = \gamma$ in (3.3). We obtain

$$\left\{\begin{array}{rcll} \Delta u & = & 0 & \text{in } \mathcal{D}, \\ u & = & f & \text{on } \partial\mathcal{D}, \end{array}\right. \tag{3.4}$$

where $\Delta u = u_{xx} + u_{yy}$ is the Laplacian of u.

This inpainting method is called harmonic inpainting since the inpainted image is obtained solving the Laplace equation in (3.4) whose solution is known as harmonic function. It is one of the best understood inpainting methods. The interpolation process consists in computing the missing grey values as the average of the values of neighbouring pixels.

Formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be the given image defined on Ω which has some grey values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. We look for an image $u \in \mathcal{L}^2(\mathcal{D})$ that satisfies (3.4).*

3.1.1 Numerical resolution

We have to solve problem (3.4). It is governed by an elliptic equation and it has Dirichlet boundary conditions. To solve it we apply the finite difference method, that is to say, we consider a mesh in our domain and in each point of the grid we replace the derivatives of the PDE by the corresponding finite difference formulas in Definition A.1, obtaining a system of equations that we can solve.

We suppose that the inpainting domain is a square $\mathcal{D} = (a, b) \times (c, d) \subset \Omega$. Let us take a mesh in $\mathcal{D}$ like the following (we can see a representation in Figure 3.1):

$$\mathcal{D}_h = \{(x_i, y_j) = (a + ih, c + jh) \mid i, j = 0, ..., n + 1\},$$

where $h = \dfrac{b-a}{n+1} = \dfrac{d-c}{n+1}$ is the step of the grid and $n \in \mathbb{N}$.

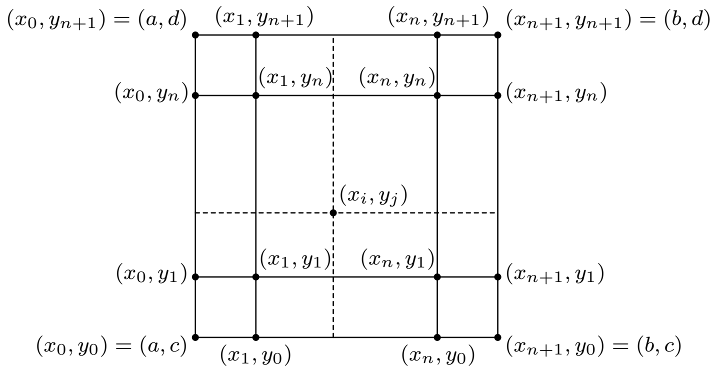

Figure 3.1: Representation of a grid in $\mathcal{D}$

Our objective is to obtain a grid function

$$\{u_{0,0}, u_{0,1}, ..., u_{0,n+1}, u_{1,0}, ..., u_{1,n+1}, ..., u_{n+1,0}, ..., u_{n+1,n+1}\}$$

such that $u_{i,j} \approx u(x_i, y_j)$ for $i, j = 0, ..., n + 1$.

Using the Dirichlet boundary conditions ($u = f$ on $\partial\mathcal{D}$) we have that

- $u_{0,j} = f(x_0, y_j)$, $j = 0, ..., n + 1$;
- $u_{n+1,j} = f(x_{n+1}, y_j)$, $j = 0, ..., n + 1$;
- $u_{i,0} = f(x_i, y_0)$, $i = 0, ..., n + 1$;
- $u_{i,n+1} = f(x_i, y_{n+1})$, $i = 0, ..., n + 1$.

To compute the remaining $u_{i,j}$ (which correspond with the inner nodes) we replace the derivatives of the equation in (3.4) by the corresponding centered finite difference formulas in Definition A.1 and we obtain (expressing it in terms of the grid function) that

$$\frac{1}{h^2}(u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}) = 0; \quad i, j = 1, ..., n. \tag{3.5}$$

The finite difference scheme can be represented by a five-point stencil (see Figure 3.2).

We have a linear system of n^2 equations with n^2 unknowns. We want to write it as $Au^h = b$. To have a matrix A with a good structure (a block tridiagonal matrix) we order the unknowns in a specific way called *natural rowwise ordering*. The vector of unknowns u^h is

$$u^h = \begin{bmatrix} u^{[1]} \\ \vdots \\ u^{[n]} \end{bmatrix} \text{ where } u^{[j]} = \begin{bmatrix} u_{1,j} \\ \vdots \\ u_{n,j} \end{bmatrix}; j = 1, ..., n.$$

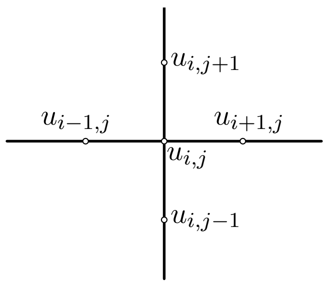

Figure 3.2: Graphic representation of the five-point stencil

Matrix A is sparse (we know that each row has at most 5 nonzero elements). In addition, the equations at points near the boundary involve the known values $u_{0,j}, u_{n+1,j}, u_{i,0}$ and $u_{i,n+1}$ $(i,j=0,...,n+1)$, so we can move them to the right-hand side b of the system.

The matrix A can be written as

$$A=\frac{1}{h^2}\begin{bmatrix} T & I & & & & \\ I & T & I & & & \\ & I & T & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & I \\ & & & & I & T \end{bmatrix},$$

where I is the $n\times n$ identity matrix and T is a tridiagonal $n\times n$ matrix of the form

$$T=\begin{bmatrix} -4 & 1 & & & \\ 1 & -4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & -4 \end{bmatrix}.$$

The vector of independent terms b is

$$b=-\frac{1}{h^2}\begin{bmatrix} b^{[1]} \\ b^{[2]} \\ \vdots \\ b^{[m]} \\ \vdots \\ b^{[n-1]} \\ b^{[n]} \end{bmatrix},$$

where $b^{[1]}$, $b^{[m]}$ $(m=2,...,n-1)$, $b^{[n]}$ are vectors of dimension n that can be written as

$$b^{[1]}=\begin{bmatrix} u_{0,1}+u_{1,0} \\ u_{2,0} \\ u_{3,0} \\ \vdots \\ u_{n-2,0} \\ u_{n-1,0} \\ u_{n+1,1}+u_{n,0} \end{bmatrix},\quad b^{[m]}=\begin{bmatrix} u_{0,m} \\ 0 \\ \vdots \\ 0 \\ u_{n+1,m} \end{bmatrix},\quad b^{[n]}=\begin{bmatrix} u_{0,n}+u_{1,n+1} \\ u_{2,n+1} \\ u_{3,n+1} \\ \vdots \\ u_{n-2,n+1} \\ u_{n-1,n+1} \\ u_{n+1,n}+u_{n,n+1} \end{bmatrix}.$$

The numerical solution of (3.4) is the solution of the system $Au^h = b$.

We program the resolution with MATLAB (see Section C.1 of Appendix C). Let us consider picture `Damaged image` in Figure 3.3, its inpainting domain is a white square (see `Damaged image detail` in Figure 3.3). Its damaged columns go from pixel 750 to pixel 790 and its damaged rows go from pixel 160 to pixel 200. Running the program, `harmonicInpainting(750,790,160,200)`, with this damaged image we obtain the image `Inpainted image` in Figure 3.3.

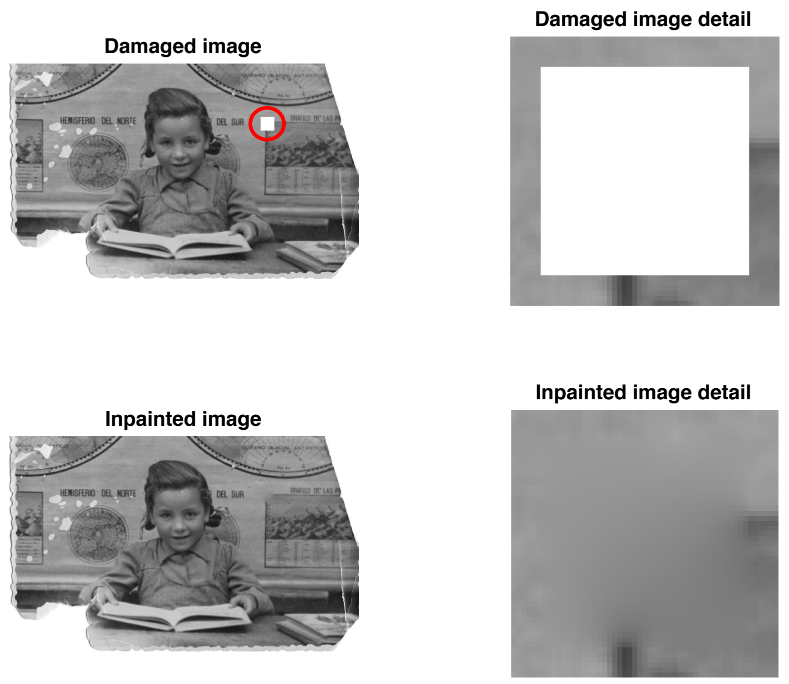

Figure 3.3: On the top we have a damaged image and we can see in detail its inpainting domain (white square). On the bottom we have the inpainted image (it has been obtained applying harmonic inpainting) and we can see in detail how the inpainting domain has been inpainted.

3.1.2 Quality of harmonic inpainting

The quality of harmonic inpainting is studied using the weak solution of (3.4).

Variational formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be the given image defined on Ω which has some grey values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. We look for an image $u \in \mathrm{H}^1(\mathcal{D})$ which fulfills*

$$\begin{cases} \int_{\mathcal{D}} \nabla u \cdot \nabla \psi \, \mathrm{d}\boldsymbol{x} & = & 0 & \forall \psi \in \mathrm{H}_0^1(\mathcal{D}), \\ u & = & f & \text{on } \partial\mathcal{D}. \end{cases}$$

This weak solution $u \in \mathrm{H}^1(\mathcal{D})$ is known as the harmonic extension u of f from $\Omega \setminus \mathcal{D}$ to $\mathcal{D}$.

$\mathrm{H}^1(\mathcal{D})$ and $\mathrm{H}_0^1(\mathcal{D})$ denote Sobolev spaces, for more detail see Definition A.4.

Now we start the study of the quality of this inpainting method. It is based on [5].

Let u^0 be the original image (that is, the image before the degradation process) defined on Ω. We suppose that u^0 is a smooth function. Let us fix a point $\boldsymbol{x_0} = (x_0, y_0) \in \mathcal{D}$ and let $\boldsymbol{x} = (x, y)$ be whatever other point in $\mathcal{D}$. Let $G(\boldsymbol{x}, \boldsymbol{x_0})$ be the Green's function (see Definition A.5) of (3.4) (understanding it as a function of $\boldsymbol{x}$). So, G is the solution of

$$\begin{cases} -\Delta G & = & \delta_{\boldsymbol{x}-\boldsymbol{x_0}} & \text{in } \mathcal{D}, \\ G & = & 0 & \text{on } \partial\mathcal{D}. \end{cases} \tag{3.6}$$

Applying Theorem A.1 (Green's Second Formula with $v_1 = u^0(\boldsymbol{x})$ and $v_2 = -G(\boldsymbol{x},\boldsymbol{x_0})$) we have that

$$\int_{\mathcal{D}} \left(u^0(\boldsymbol{x})(-\Delta G(\boldsymbol{x},\boldsymbol{x_0})) + G(\boldsymbol{x},\boldsymbol{x_0})\Delta u^0(\boldsymbol{x})\right) \mathrm{d}\boldsymbol{x} = \int_{\partial\mathcal{D}} \left(u^0(\boldsymbol{x})\frac{\partial(-G(\boldsymbol{x},\boldsymbol{x_0}))}{\partial\vec{n}} + G(\boldsymbol{x},\boldsymbol{x_0})\frac{\partial u^0(\boldsymbol{x})}{\partial\vec{n}}\right) \mathrm{d}s.$$

Using (3.6) we have that

$$u^0(\boldsymbol{x_0}) = \int_{\partial\mathcal{D}} u^0(\boldsymbol{x})\frac{\partial(-G(\boldsymbol{x},\boldsymbol{x_0}))}{\partial\vec{n}}\mathrm{d}s + \int_{\mathcal{D}} G(\boldsymbol{x},\boldsymbol{x_0})(-\Delta u^0(\boldsymbol{x}))\mathrm{d}\boldsymbol{x}.$$

So, we have a decomposition of u^0 of the form

$$u^0(\boldsymbol{x_0}) = u^h(\boldsymbol{x_0}) + u^a(\boldsymbol{x_0}).$$

The first term $u^h(\boldsymbol{x_0}) = \int_{\partial\mathcal{D}} u^0(\boldsymbol{x})\frac{\partial(-G(\boldsymbol{x},\boldsymbol{x_0}))}{\partial\vec{n}}\mathrm{d}s$ is the harmonic part. It is the extension of $u^0_{|\partial\mathcal{D}}$ respect to the harmonic measure $\frac{-\partial G(\boldsymbol{x},\boldsymbol{x_0})}{\partial\vec{n}}\mathrm{d}s$. It can be understood as the harmonic extension and we have that $u^h(\boldsymbol{x_0}) = u(\boldsymbol{x_0})$. The second term $u^a(\boldsymbol{x_0}) = \int_{\mathcal{D}} G(\boldsymbol{x},\boldsymbol{x_0})(-\Delta u^0(\boldsymbol{x}))\mathrm{d}\boldsymbol{x}$ is the difference between the real solution and the harmonic extension, so it is the error made with this inpainting method.

Theorem 3.2. *Let $d = \max_{r,s\in\partial\mathcal{D}} \|r-s\|$ be the diameter of $\mathcal{D}$ (inpainting domain) and let G be the Green's function (see Definition A.5) of the Laplacian (the one obtained solving (3.6)). Then,*

$$\int_{\mathcal{D}} |G(\boldsymbol{x},\boldsymbol{x_0})|\mathrm{d}\boldsymbol{x} \leq \frac{d^2}{4}.$$

As u^0 is a smooth function, $|\Delta u^0|$ is bounded by a constant $C > 0$. Using this fact and Theorem 3.2 we have that

$$|u^a(\boldsymbol{x_0})| \leq \int_{\mathcal{D}} |G(\boldsymbol{x},\boldsymbol{x_0})||-\Delta u^0(\boldsymbol{x})|\mathrm{d}\boldsymbol{x} \leq C\int_{\mathcal{D}} |G(\boldsymbol{x},\boldsymbol{x_0})|\mathrm{d}\boldsymbol{x} \leq C\frac{d^2}{4}.$$

We conclude that the error at point $\boldsymbol{x_0}$ satisfies that $|u^a(\boldsymbol{x_0})| \leq C\frac{d^2}{4} \equiv \mathcal{O}(d^2)$. So, if the inpainting domain is a big hole, the bound of the error is big enough to support the idea that harmonic inpainting is not good enough (the good continuation principle is not fulfilled).

In `Inpainted image detail` in Figure 3.3 we can see in detail the region that has been inpainted in `Damaged image` in Figure 3.3. It can be clearly seen that the edges are poorly propagated inside the gap because it is considerably large.

3.2 TV inpainting

Total Variation (TV) is a well-known image processing method since Rudin, Osher and Fatemi presented it in their work [6]. It was proposed with the aim of preserve sharp image structures like edges using smooth methods. TV is applied in image denoising, image deblurring and image segmentation, so it makes sense to try to use it in image inpainting.

TV inpainting arises taking $\alpha = \beta = 0$ and $\gamma \neq 0$ in (3.3):

$$\left\{\begin{array}{rcll} \frac{1}{\|\nabla u\|^2}\left(u_{xx}u_y^2 - 2u_{xy}u_xu_y + u_{yy}u_x^2\right) &=& 0 & \text{in } \mathcal{D}, \\ u &=& f & \text{on } \partial\mathcal{D}. \end{array}\right. \tag{3.7}$$

It is easy to prove that the following problem is equivalent to the previous one in (3.7):

$$\begin{cases} \nabla \cdot \left(\dfrac{\nabla u}{\|\nabla u\|} \right) & = & 0 & \text{in } \mathcal{D}, \\ \\ u & = & f & \text{on } \partial\mathcal{D}. \end{cases} \tag{3.8}$$

The interpolation process consists in computing the values of the missing pixels as the median value of neighbouring pixels.

Formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be the given image defined on Ω which has some grey values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. We look for an image $u \in \mathcal{L}^2(\mathcal{D})$ that satisfies* (3.8).

3.2.1 Numerical resolution

We have to solve problem (3.8). To obtain the numerical solution we work with the following evolution problem:

$$\begin{cases} u_t & = & \nabla \cdot \left(\dfrac{\nabla u}{\|\nabla u\|_\delta} \right) & \text{in } \mathcal{D}, \\ \\ u & = & f & \text{on } \partial\mathcal{D}. \end{cases} \tag{3.9}$$

To avoid problems with 0 in the denominator we have taken $\|\nabla u\|_\delta = \sqrt{\|\nabla u\|^2 + \delta}$ instead of $\|\nabla u\|$, where $0 < \delta \ll 1$.

We suppose that we have the same spatial inpainting domain $\mathcal{D}$ and the same spatial grid that we had in the numerical resolution of harmonic inpainting method in Section 3.1.1 Numerical resolution. In the temporary direction we consider a grid with step size $\tau = (T_{final} - 0) \,/\, (m+1)$ with $m \in \mathbb{N}$. We define $t_k = k\tau$, $k = 0, ..., m+1$. So the resultant mesh is

$$\mathcal{D}_{h,\tau} = \{(x_i, y_j, t_k) = (a + ih, c + jh, k\tau) \mid i,j = 0, ..., n+1; k = 0, ..., m+1\}.$$

We denote by $u_{i,j}^k$ the approximation of $u(x_i, y_j)$ at time level t_k $(i,j = 0, ..., n+1,\ k = 0, ..., m+1)$. Our objective is to compute a spatial grid function

$$\{u_{0,0}^k, u_{0,1}^k, ..., u_{0,n+1}^k, u_{1,0}^k, ..., u_{1,n+1}^k, ..., u_{n+1,0}^k, ..., u_{n+1,n+1}^k\}$$

for $k = 1, ..., m+1$. For $k = 0$ (initial condition) the spatial grid function is given by the values of the pixels of $\mathcal{D}$ of the damaged image. The spatial grid function that we look for is the one obtained at time level t_{m+1}.

Note that at any time t_k the values of $u_{0,j}^k$, $u_{n+1,j}^k$, $u_{i,0}^k$ and $u_{i,n+1}^k$ $(i,j = 0, ..., n+1)$ are known thanks to the Dirichlet boundary conditions ($u = f$ on $\partial\mathcal{D}$) and these values are the same for any k:

- $u_{0,j}^k = f(x_0, y_j)$, $j = 0, ..., n+1$;
- $u_{n+1,j}^k = f(x_{n+1}, y_j)$, $j = 0, ..., n+1$;
- $u_{i,0}^k = f(x_i, y_0)$, $i = 0, ..., n+1$;
- $u_{i,n+1}^k = f(x_i, y_{n+1})$, $i = 0, ..., n+1$.

In the following we focus our calculations on the inner nodes of the spatial mesh for each time level. Let us start the discretization of our problem. The temporary derivative and the spatial derivatives are approximated with the corresponding finite difference formulas in Definition A.1. Denoting by $\left(\nabla_h \cdot \left(\dfrac{\nabla_h u}{\|\nabla_h u\|_\delta}\right)\right)_{i,j}^{k-1}$ the discretization of $\nabla \cdot \left(\dfrac{\nabla u}{\|\nabla u\|_\delta}\right)$ at (x_i, y_j, t_{k-1}) we have that our scheme in differences is

$$\frac{u_{i,j}^k - u_{i,j}^{k-1}}{\tau} = \left(\nabla_h \cdot \left(\frac{\nabla_h u}{\|\nabla_h u\|_\delta}\right)\right)_{i,j}^{k-1},$$

and expressing $u^k_{i,j}$ in terms of the values of the approximations of the previous level we obtain an explicit scheme

$$u^k_{i,j} = \tau \left(\nabla_h \cdot \left(\frac{\nabla_h u}{\|\nabla_h u\|_\delta} \right) \right)^{k-1}_{i,j} + u^{k-1}_{i,j}.$$

Discretization of $\nabla \cdot \left(\frac{\nabla u}{\|\nabla u\|_\delta} \right)$**.** To do this discretization we consider the same spatial grid that we have been using in this section and we use the corresponding finite difference formulas in Definition A.1.

Calling $\mathcal{K}_\delta(x,y) = \dfrac{1}{\|\nabla_h u(x,y)\|_\delta}$, for $i,j = 1,...,n$ we have

$$\left(\nabla_h \cdot \left(\frac{\nabla_h u}{\|\nabla_h u\|_\delta} \right) \right)(x_i,y_j) \approx \frac{\mathcal{K}_\delta(x_{i+1/2},y_j)u_x(x_{i+1/2},y_j) - \mathcal{K}_\delta(x_{i-1/2},y_j)u_x(x_{i-1/2},y_j)}{h} +$$
$$+ \frac{\mathcal{K}_\delta(x_i,y_{j+1/2})u_y(x_i,y_{j+1/2}) - \mathcal{K}_\delta(x_i,y_{j-1/2})u_y(x_i,y_{j-1/2})}{h} \approx$$
$$\approx \frac{\mathcal{K}_\delta(x_{i+1/2},y_j)\dfrac{u_{i+1,j}-u_{i,j}}{h} - \mathcal{K}_\delta(x_{i-1/2},y_j)\dfrac{u_{i,j}-u_{i-1,j}}{h}}{h} +$$
$$+ \frac{\mathcal{K}_\delta(x_i,y_{j+1/2})\dfrac{u_{i,j+1}-u_{i,j}}{h} - \mathcal{K}_\delta(x_i,y_{j-1/2})\dfrac{u_{i,j}-u_{i,j-1}}{h}}{h}.$$

Finally, we obtain

$$\left(\nabla_h \cdot \left(\frac{\nabla_h u}{\|\nabla_h u\|_\delta} \right) \right)(x_i,y_j) \approx$$
$$\approx \frac{1}{h^2}[\mathcal{K}_\delta(x_{i+1/2},y_j)u_{i+1,j} + \mathcal{K}_\delta(x_{i-1/2},y_j)u_{i-1,j} + \mathcal{K}_\delta(x_i,y_{j+1/2})u_{i,j+1} + \mathcal{K}_\delta(x_i,y_{j-1/2})u_{i,j-1} -$$
$$- \left[\mathcal{K}_\delta(x_{i+1/2},y_j) + \mathcal{K}_\delta(x_{i-1/2},y_j) + \mathcal{K}_\delta(x_i,y_{j+1/2}) + \mathcal{K}_\delta(x_i,y_{j-1/2})\right] u_{i,j}] \quad \text{for } i,j = 1,...,n. \tag{3.10}$$

Let us compute the expressions of $\mathcal{K}_\delta(x_{i+1/2},y_j)$, $\mathcal{K}_\delta(x_{i-1/2},y_j)$, $\mathcal{K}_\delta(x_i,y_{j+1/2})$ and $\mathcal{K}_\delta(x_i,y_{j-1/2})$ for $i,j = 1,...,n$.

$$\mathcal{K}_\delta(x_{i+1/2},y_j) \approx \frac{1}{\sqrt{\left(\dfrac{u_{i+1,j}-u_{i,j}}{h}\right)^2 + \left(\dfrac{u_{i+1/2,j+1/2}-u_{i+1/2,j-1/2}}{h}\right)^2 + \delta}} =$$
$$= \frac{h}{\sqrt{\left(u_{i+1,j}-u_{i,j}\right)^2 + \left(u_{i+1/2,j+1/2}-u_{i+1/2,j-1/2}\right)^2 + \delta h^2}}.$$

Realize that $u_{i+1/2,j+1/2}$ and $u_{i+1/2,j-1/2}$ are not nodes of our spatial grid, so these are unknown values. We approximate these values in the following way:

$$u_{i+1/2,j+1/2} = \frac{u_{i,j+1}+u_{i+1,j+1}}{2}, \quad u_{i+1/2,j-1/2} = \frac{u_{i,j-1}+u_{i+1,j-1}}{2}.$$

So, $$\mathcal{K}_\delta(x_{i+1/2}, y_j) \approx \frac{h}{\sqrt{(u_{i+1,j}-u_{i,j})^2 + ((u_{i,j+1}+u_{i+1,j+1}-u_{i,j-1}-u_{i+1,j-1})/2)^2 + \delta h^2}}. \tag{3.11}$$

Proceeding in a similar way we obtain that

$$\mathcal{K}_\delta(x_{i-1/2}, y_j) \approx \frac{h}{\sqrt{(u_{i,j}-u_{i-1,j})^2 + ((u_{i-1,j+1}+u_{i,j+1}-u_{i-1,j-1}-u_{i,j-1})/2)^2 + \delta h^2}}, \tag{3.12}$$

$$\mathcal{K}_\delta(x_i, y_{j+1/2}) \approx \frac{h}{\sqrt{(u_{i,j+1}-u_{i,j})^2 + ((u_{i+1,j+1}-u_{i-1,j+1})/2)^2 + \delta h^2}} \quad \text{and} \tag{3.13}$$

$$\mathcal{K}_\delta(x_i, y_{j-1/2}) \approx \frac{h}{\sqrt{(u_{i,j}-u_{i,j-1})^2 + ((u_{i+1,j-1}-u_{i-1,j-1})/2)^2 + \delta h^2}}. \tag{3.14}$$

The finite difference scheme in this case is (3.10) with the expressions in (3.11), (3.12), (3.13) and (3.14). It can be represented by a nine-point stencil (see Figure 3.4).

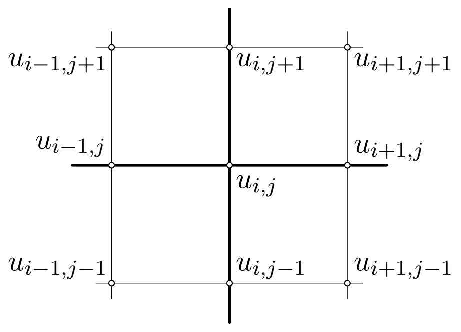

Figure 3.4: Graphic representation of the nine-point stencil

In conclusion, we have that the discretization of (3.9) for each time level is

$$u_{i,j}^k = \frac{\tau}{h^2}(\mathcal{K}_\delta(x_{i+1/2}, y_j, t_{k-1})u_{i+1,j}^{k-1} + \mathcal{K}_\delta(x_{i-1/2}, y_j, t_{k-1})u_{i-1,j}^{k-1} + \mathcal{K}_\delta(x_i, y_{j+1/2}, t_{k-1})u_{i,j+1}^{k-1} +$$

$$+\mathcal{K}_\delta(x_i, y_{j-1/2}, t_{k-1})u_{i,j-1}^{k-1} - [\mathcal{K}_\delta(x_{i+1/2}, y_j, t_{k-1}) + \mathcal{K}_\delta(x_{i-1/2}, y_j, t_{k-1}) + \mathcal{K}_\delta(x_i, y_{j+1/2}, t_{k-1}) +$$

$$+\mathcal{K}_\delta(x_i, y_{j-1/2}, t_{k-1})]u_{i,j}^{k-1}) + u_{i,j}^{k-1} \quad \text{for } i,j = 1, ..., n.$$

Realize that if $i, j = 1, n$, then the Dirichlet boundary conditions are involved in the computations.

We program the numerical resolution of this problem with MATLAB (see Section C.2 of Appendix C). Let us consider the image `Damaged image` in Figure 3.5. Its inpainting domain is a white square (see `Damaged image detail` in Figure 3.5). Its damaged columns go from pixel 760 to pixel 770 and its damaged rows go from pixel 170 to pixel 180. We run the program, `TVInpainting(760,770,170,180,100000)`, with this damaged image and we obtain the picture `Inpainted image` in Figure 3.5.

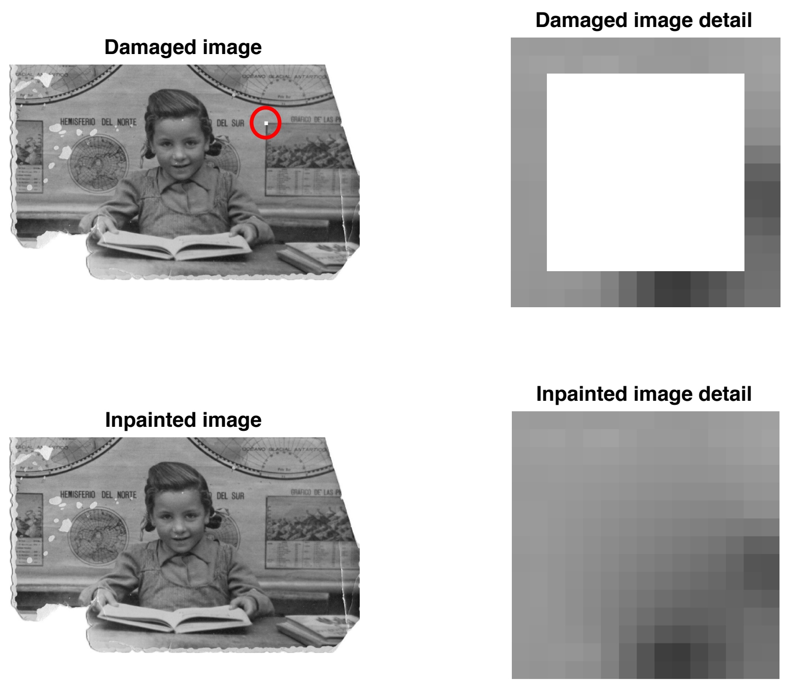

Figure 3.5: On the top we have a damaged image and we can see in detail its inpainting domain (white square). On the bottom we have the inpainted image (it has been obtained applying TV inpainting) and we can see in detail how the inpainting domain has been inpainted.

3.2.2 Issues that TV inpainting presents

The principal problem of TV inpainting is that the solution may not be unique.

Example 3.2. Let $\mathcal{D}$ be the unit disk and let us take the function $\phi(\boldsymbol{x}) = \lambda_1 x_1^2 + \lambda_2 x_2^2$ where λ_1 and λ_2 are two parameters such that $\lambda_1 > \lambda_2$. Then,

$$u_1(\boldsymbol{x}) = \phi\left(\sqrt{1-x_2^2}, x_2\right) \quad \text{and} \quad u_2(\boldsymbol{x}) = \phi\left(x_1, \sqrt{1-x_1^2}\right)$$

are two different solutions of (3.7). This example can be found in [7].

This method also presents problems related to its quality. To study this we have to discuss how edge information of f is propagated into $\mathcal{D}$. To do it we need to obtain the weak formulation of the problem. However, before we need to introduce a new space (we have a method that preserves sharp structures, so we need a less smooth space than Sobolev space H^1 (see Definition A.4)): the space of functions of bounded variation (BV).

Definition 3.2. Total variation (TV) for integrable functions.
Let $u \in \mathcal{L}^1_{loc}(\mathcal{D})$, we define the total variation (TV) of u by duality, that is to say, TV is a functional which associates a scalar value $\mathrm{TV}(u)$ to each u.

$$\mathrm{TV}(u) = |Du|(\mathcal{D}) = \sup\left\{-\int_{\mathcal{D}} u\ \nabla\cdot\phi\ \mathrm{d}\boldsymbol{x} \mid \phi \in \mathcal{C}_c^\infty(\mathcal{D};\mathbb{R}^2), |\phi(\boldsymbol{x})| \leq 1\ \forall \boldsymbol{x} \in \mathcal{D}\right\}.$$

$\mathcal{L}^1_{loc}(\mathcal{D})$ denotes the set of locally integrable functions defined on $\mathcal{D}$ (see Definition A.6).

Example 3.3. TV of some functions.

- If $u \in W^{1,1}(\mathcal{D})$ (see Definition A.4), the total variation of u is $\mathrm{TV}(u) = |Du|(\mathcal{D}) = \|\nabla u\|_1$.

- If $u = \chi_A$ with $A \subset \mathcal{D}$, the total variation of u is $\mathrm{TV}(u) = |Du|(\mathcal{D}) = \mathcal{H}^1(\partial A)$, where $\mathcal{H}^1$ is the one-dimensional Hausdorff measure.

Definition 3.3. Space of functions of bounded variation (BV).
The space of functions of bounded variation in $\mathcal{D}$, denoted by $\mathrm{BV}(\mathcal{D})$, is

$$\mathrm{BV}(\mathcal{D}) = \left\{ u \in \mathcal{L}^1(\mathcal{D}) \mid \mathrm{TV}(u) < \infty \right\}.$$

It is a Banach space with the norm $\|u\|_{\mathrm{BV}} = \|u\|_1 + \mathrm{TV}(u)$.

Variational formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be the given image defined on Ω which has some grey values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. We look for*

$$\min_{u \in \mathrm{BV}(\mathcal{D})} \{|Du|(\mathcal{D}) \text{ such that } u = f \text{ on } \partial\mathcal{D}\}. \tag{3.15}$$

Definition 3.4. Perimeter.
Let $E \subset \mathcal{D}$ be a measurable set in $\mathbb{R}^2$. E is a set of finite perimeter if and only if its characteristic function (χ_E) is a function of bounded variation in $\mathcal{D}$. $Per(E;\mathcal{D}) := |D\chi_E|(\mathcal{D})$ is the perimeter of E in $\mathcal{D}$.

Theorem 3.3. *Co-area formula.*
Let $u \in \mathrm{BV}(\mathcal{D})$ and let us consider the so-called s-sup level set of u defined by $\{u > s\} := \{\boldsymbol{x} \in \mathcal{D} \mid u(\boldsymbol{x}) > s\}$ with $s \in \mathbb{R}$. Then,

$$|Du|(\mathcal{D}) = \int_{-\infty}^{\infty} Per(\{u > s\};\mathcal{D})\mathrm{d}s.$$

Let us assume that the original image u^0 (the one that is not damaged and such that $u^0_{|\Omega\setminus\mathcal{D}} = f_{|\Omega\setminus\mathcal{D}}$) is smooth and is not constant inside $\mathcal{D}$. That allows us to assert that there exists $\boldsymbol{x}_0$ close to $\mathcal{D}$ such that $\nabla f(\boldsymbol{x}_0) \neq \mathbf{0}$. Applying Theorem A.2 (inverse function theorem), we have that the level lines of $\mathcal{D}$ have to be well-defined.

Let us consider the s-level line $\Gamma_s = \{\boldsymbol{x} \in \mathcal{D} \mid u^0(\boldsymbol{x}) = s\}$. Our inpainting problem is reduced to interpolate Γ_s by a curve γ_s that joins two points of the boundary, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, such that $f(\boldsymbol{x}_1) = f(\boldsymbol{x}_2) = s$. Thanks to the weak formulation in (3.15) we know that we have to look for a solution that minimizes $|Du|(\mathcal{D})$. Using co-area formula (that is, Theorem 3.3) it can be proved that the level line is interpolated by a straight line (for a more detailed explanation see [3]).

We conclude that in TV inpainting the interpolation is done linearly. That means that curvature is not preserved. This contradicts the good continuation principle, so TV inpainting is not good enough.

In picture `Inpainted image detail` in Figure 3.5 we can see in detail the part of the damaged image (see `Damaged image` in Figure 3.5) that has been inpainted. If we look the image carefully we can see that the darkest pixels of the inpainting domain are located on a line (not on a right angle).

For more information about the inpainting methods described in this chapter see [7].

Chapter 4

Inpainting methods based on the Cahn-Hilliard equation

As we have seen in the previous chapter, second-order inpainting methods present problems related to the connection of edges over large distances and the propagation of these edges in the right direction. To solve these drawbacks we need a method which has to take into account the gradient of the image on the boundary of the inpainting domain (Neumann boundary conditions). We have to increase the order of the equations we consider.

In this chapter we present a fourth-order inpainting method for binary images based on a modification of the Cahn-Hilliard equation and other method which is a generalization for grey value images.

The Cahn-Hilliard equation is a non-linear fourth-order equation which describes the process of the phase separation in binary alloys. It can be written in the following way:

$$\begin{cases} u_t = \nabla \cdot (M(u)\nabla\mu(u)) & \text{in } \mathcal{D}, \\ \nabla u \cdot \vec{n} = \nabla\mu \cdot \vec{n} = 0 & \text{on } \partial\mathcal{D}, \end{cases}$$

where $\mu(u) = F'(u) - \epsilon^2\Delta u$, $\mathcal{D} \subset \mathbb{R}^n$ $(n = 1, 2, 3)$ is a bounded domain, $F(u) = 0.25(u^2 - 1)^2$ is a double-well potential, $M(u)$ is a positive mobility and $\epsilon > 0$. u represents the phase-field.

Some basic principles and practical applications of the Cahn-Hilliard equation can be found in [8].

Inpainting methods based on the Cahn-Hilliard equation are good methods because the equations are relatively simple (although they do not involve curvature terms, they fulfill the good continuation principle) and their numerical solutions have good properties.

4.1 Cahn-Hilliard inpainting for binary images

A binary image is an image whose pixels can only take two values. To make it more simple we consider that these values are the ones which correspond to black and white.

In this section we present a variation of the Cahn-Hilliard equation for inpainting which can be applied to binary images. This inpainting method is known as Cahn-Hilliard inpainting. It seems a bit restrictive method because of the fact that it only works well for binary images. However, it is a good method for reconstructing the structural part of an image. So, combining it with other inpainting methods we can obtain good results. In addition, we can extend it to formulate an inpainting method for grey value images (it is called TV-H^{-1} inpainting and we will study it later).

Formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be a given binary image defined on Ω which has some values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. The inpainted image u we are looking for is constructed* *following the evolution of*

$$\begin{cases} u_t = \Delta\left(-\epsilon\Delta u + \frac{1}{\epsilon}F'(u)\right) & \text{in } \mathcal{D}, \\ u = f & \text{on } \partial\mathcal{D}, \\ \nabla u\cdot\vec{n} = \nabla f\cdot\vec{n} & \text{on } \partial\mathcal{D}, \end{cases} \tag{4.1}$$

where $F(u)$ is the double-well potential (we can take $F(u) = u^2(u-1)^2$ because the wells of F are the two values that the image can take: black or white) and $\epsilon > 0$.

4.1.1 Numerical resolution

Our inpainted image u is the stationary solution of (4.1). To obtain such u we use a method for evolution equations known as convexity splitting (to learn more about it and its applications on inpainting see [9]). Using it we obtain a time-stepping scheme, to which we apply the finite difference method.

Lemma 4.1. *General procedure needed to obtain the expression used in each time level.*
Let u be a function defined on $\mathcal{D}\times[0,T_{final}]$ where $\mathcal{D}\subset\Omega$ and $T_{final} > 0$. Let G be a real function and let us consider $u_t = G(u, D^\alpha u)$ (D^α means that it depends on the derivatives, $\alpha = 1,2,3,4$) as our PDE. The time-stepping scheme at time level t_k is

$$\frac{u^k - u^{k-1}}{\tau} = G_k(u^{k-1}, u^k, D^\alpha u^{k-1}, D^\alpha u^k),$$

where τ is the size of the time step (which is defined in the same way as in Section 3.2.1 Numerical resolution), G_k is an approximation of G at time level t_k and $\alpha = 1,2,3,4$.

Looking at our problem (4.1), we have that $G(u, D^\alpha u) = \Delta\left(-\epsilon\Delta u + \frac{1}{\epsilon}F'(u)\right)$ and the approximation at time level t_k is

$$G_k(u^{k-1}, u^k, D^\alpha u^{k-1}, D^\alpha u^k) = -\epsilon\Delta\Delta u^k + \frac{1}{\epsilon}\Delta F'(u^{k-1}) + C_1(\Delta u^k - \Delta u^{k-1}) + C_2(u^{k-1} - u^k),$$

where C_1 is comparable to $1/\epsilon$ and $C_2 \gg 1$. For each time level we have the following scheme:

$$\frac{u^k - u^{k-1}}{\tau} = -\epsilon\Delta\Delta u^k + \frac{1}{\epsilon}\Delta F'(u^{k-1}) + C_1(\Delta u^k - \Delta u^{k-1}) + C_2(u^{k-1} - u^k).$$

Putting the terms which depend on the previous time level t_{k-1} on the right-hand side and the terms which are involved in this time level t_k on the left-hand side we obtain the problem that we have to solve for each k:

$$\begin{cases} \epsilon\Delta\Delta u^k - C_1\Delta u^k + (1/\tau + C_2)\,u^k = g(u^{k-1}) & \text{in } \mathcal{D}, \\ u^k = f & \text{on } \partial\mathcal{D}, \\ \nabla u^k\cdot\vec{n} = \nabla f\cdot\vec{n} & \text{on } \partial\mathcal{D}, \end{cases} \tag{4.2}$$

where $g(u^{k-1}) = \left(\Delta F'(u^{k-1})\right)/\epsilon - C_1\Delta u^{k-1} + (C_2 + 1/\tau)\,u^{k-1}$.

To solve the previous problem (4.2) we apply the finite difference method. We suppose that the inpainting domain is a square $\mathcal{D} = (a,b)\times(c,d)\subset\Omega$. For each time level we take the same spatial grid that we used in Section 3.1.1 Numerical resolution, that is,

$$\mathcal{D}_h = \{(x_i, y_j) = (a + ih, c + jh) \mid i,j = 0,...,n+1\}.$$

We denote by $u^k_{i,j}$ the approximation of $u(x_i, y_j)$ at time level t_k $(i, j = 0, ..., n+1,\ k = 0, ..., m+1)$. We have to compute a spatial grid function

$$\{u^k_{0,0}, u^k_{0,1}, ..., u^k_{0,n+1}, u^k_{1,0}, ..., u^k_{1,n+1}, ..., u^k_{n+1,0}, ..., u^k_{n+1,n+1}\}$$

for $k = 1, ..., m+1$. For $k = 0$ the spatial grid function is given by the values of the pixels of $\mathcal{D}$ of the damaged image. The spatial grid function at t_{m+1} is the one we are looking for.

Using that $u^k = f$ on $\partial\mathcal{D}$ (Dirichlet boundary conditions) we have that at any time level we know the value of the following elements of the grid function:

- $u^k_{0,j} = f(x_0, y_j)$, $j = 0, ..., n+1$;
- $u^k_{n+1,j} = f(x_{n+1}, y_j)$, $j = 0, ..., n+1$;
- $u^k_{i,0} = f(x_i, y_0)$, $i = 0, ..., n+1$;
- $u^k_{i,n+1} = f(x_i, y_{n+1})$, $i = 0, ..., n+1$.

Note that these values are the same for any k.

To compute the remaining $u^k_{i,j}$ (which correspond with the inner nodes) we have to discretize problem (4.2). The idea is to solve a system of n^2 equations with n^2 unknowns of the form $A^k(u^k)^h = b^k$, where A^k is the matrix of coefficients obtained with the approximation of the left-hand side of the equation in problem (4.2) by the corresponding centered finite difference formulas in Definition A.1, $(u^k)^h$ is the vector of unknowns ordered in the same way as in Section 3.1.1 Numerical resolution and b^k is the vector of independent terms which corresponds with the sum of $g(u^{k-1})$ and the known values of $u^k_{i,j}$ that appear because of the discretization of the left-hand side (they are related to the boundary conditions).

To obtain A^k we follow these steps:

- We compute the discretization of $\Delta\Delta\hat{u}^k_1 = 0$ with the same boundary conditions as in (4.2). We obtain a system of the form $A^k_1(\hat{u}^k_1)^h = b^k_1$ where $(\hat{u}^k_1)^h$ is the vector of unknowns ordered in the same way as $(u^k)^h$. We can find the computations in Appendix B. We need the matrix of coefficients that we denote by A^k_1.

- We compute the discretization of $\Delta\hat{u}^k_2 = 0$ with the same Dirichlet boundary conditions as in (4.2) (in this case Neumann boundary conditions are not needed). We obtain a system of the form $A^k_2(\hat{u}^k_2)^h = b^k_2$ where $(\hat{u}^k_2)^h$ is the vector of unknowns ordered in the same way as $(u^k)^h$. We have computed this case in Section 3.1.1 Numerical resolution. We need the matrix of coefficients that we denote by A^k_2.

- We compute A^k as $A^k = \epsilon A^k_1 - C_1 A^k_2 + (1/\tau + C_2)\, I_{n^2}$.

To obtain the expression of b^k we have to do the following:

- When we have computed A^k_1 and A^k_2 we have obtained in each case a vector of independent terms (nodes on the boundary and ghost nodes are involved). We have to consider these vectors b^k_1 (the vector of independent terms in Appendix B) and b^k_2 (the vector of independent terms in Section 3.1.1 Numerical resolution) as part of the vector b^k.

- We have to discretize the expression $g(\hat{u}) = (\Delta F'(\hat{u}))/\epsilon - C_1\Delta\hat{u} + (C_2 + 1/\tau)\,\hat{u}$ in order to evaluate it with the vector u^{k-1} computed in the previous time level.

  To do the discretization of $g(\hat{u})$ we compute the discretization of each term (last term does not need any discretization):

  - $(\Delta F'(\hat{u}))/\epsilon$. To compute the discretization of this term we use the discretization of the Laplacian in (3.5). Adjusting it to our particular case we have that the discretization we are looking for is

$$\left(F'(\hat{u}_{i+1,j}) + F'(\hat{u}_{i-1,j}) + F'(\hat{u}_{i,j+1}) + F'(\hat{u}_{i,j-1}) - 4F'(\hat{u}_{i,j})\right) / \left(\epsilon h^2\right) \quad \text{for } i, j = 1, ..., n.$$

    Realize that if $i, j = 1, n$, then Dirichlet boundary conditions are needed.

- $-C_1 \Delta \hat{u}$. To compute the discretization of this expression we need equation (3.5) as in the previous term. Adjusting it to this case, we obtain

$$-C_1 \left(\hat{u}_{i+1,j} + \hat{u}_{i-1,j} + \hat{u}_{i,j+1} + \hat{u}_{i,j-1} - 4\hat{u}_{i,j} \right) / h^2 \quad \text{for } i, j = 1, ..., n.$$

  Note that if $i, j = 1, n$, then Dirichlet boundary conditions are needed.

- We compute b^k as $b^k = (\textit{discretization of } g(\hat{u}) \textit{ evaluated at } u^{k-1}) + \epsilon b_1^k - C_1 b_2^k$.

We program the numerical resolution of this method with MATLAB (see Section C.3 of Appendix C). We use the binary image `Damaged image` in Figure 4.1. The inpainted domain is a white square (see `Damaged image detail` in Figure 4.1). The damaged columns go from pixel 1020 to pixel 1040 and the damaged rows go from pixel 1510 to pixel 1530. Running the program, `CHBinary(1020,1040,1510,1530,50000)`, we obtain the picture `Inpainted image` in Figure 4.1.

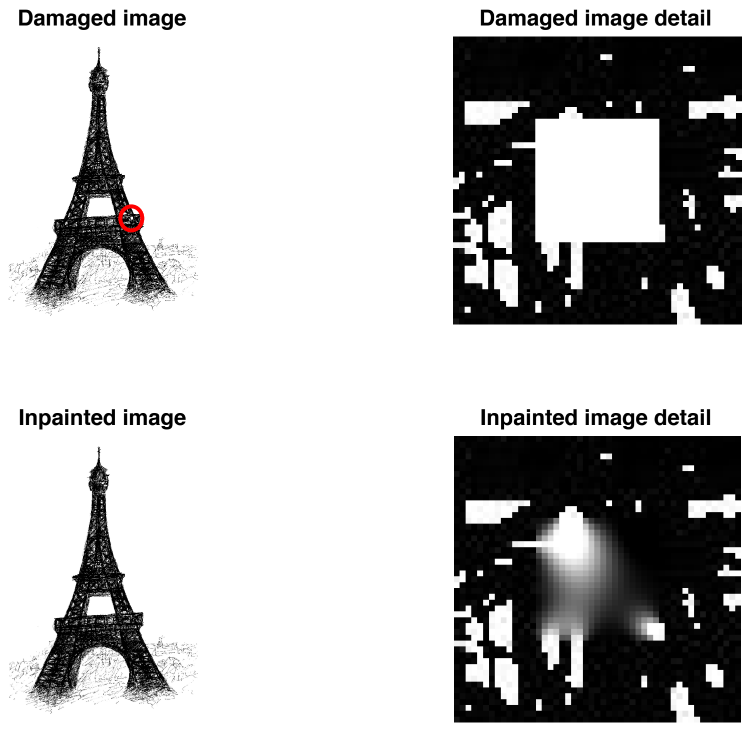

Figure 4.1: On the top we have a damaged image and we can see in detail its inpainting domain (white square). On the bottom we have the inpainted image (it has been obtained applying Cahn-Hilliard inpainting) and we can see in detail how the inpainting domain has been inpainted.

4.2 Extension of Cahn-Hilliard inpainting to grey value images: TV-H^{-1} inpainting

We can extend the inpainting method in Section 4.1 to obtain another one that can be applied to grey value images (also known as black and white images). This new method is called TV-H^{-1} inpainting.

Before presenting the formulation of this method we need to explain some new mathematical concepts related to the concept of total variation explained in Definition 3.2.

Definition 4.1. Total variation with infinite bound.

Let $u \in \mathcal{L}^1(\mathcal{D})$. We can define total variation with infinite bound (TV_∞) as

$$\mathrm{TV}_\infty(u) = \begin{cases} \mathrm{TV}(u) & \text{if } 0 \leq u(\boldsymbol{x}) \leq 1 \text{ a.e. in } \mathcal{D}, \\ +\infty & \text{otherwise.} \end{cases}$$

Definition 4.2. Subdifferential of the functional $\mathrm{TV}_\infty(u)$.

The subdifferential of the functional $\mathrm{TV}_\infty(u)$ is

$$\partial \mathrm{TV}_\infty(u) = \left\{ p \in \mathcal{L}^2(\mathcal{D}) \mid \langle v-u, p \rangle \leq \mathrm{TV}_\infty(v) - \mathrm{TV}_\infty(u),\ \forall v \in \mathcal{L}^2(\mathcal{D}) \right\}.$$

Formulation. *Let $f \in \mathcal{L}^2(\Omega)$ be a given grey value image defined on Ω which has some values lost inside the inpainting domain $\mathcal{D} \subset \Omega$. The inpainted image u is the evolution of*

$$\begin{cases} u_t &= \Delta p & \text{in } \mathcal{D}, \\ u &= f & \text{on } \partial\mathcal{D}, \\ \nabla u \cdot \vec{n} &= \nabla f \cdot \vec{n} & \text{on } \partial\mathcal{D}, \end{cases} \tag{4.3}$$

where $p \in \partial \mathrm{TV}_\infty(u)$. We take $p = -\nabla \cdot \left(\dfrac{\nabla u}{\|\nabla u\|_\delta} \right)$, where $\|\nabla u\|_\delta = \sqrt{\|\nabla u\|^2 + \delta}$, $0 < \delta \ll 1$.

4.2.1 Numerical resolution

The inpainted image u is the stationary solution of (4.3). To obtain u we use the method for evolution equations that we have used for the numerical resolution of Cahn-Hilliard inpainting for binary images (see Section 4.1.1 Numerical resolution), that is, convexity splitting. We obtain a time-stepping scheme and we apply the finite difference method.

In this case, looking at (4.3) we deduce that the function G defined in Lemma 4.1 is $G(u, D^\alpha u) = \Delta p$ and the approximation at time level t_k is

$$G_k(u^{k-1}, u^k, D^\alpha u^{k-1}, D^\alpha u^k) = -\Delta \left(\nabla \cdot \left(\frac{\nabla u^{k-1}}{\|\nabla u^{k-1}\|_\delta} \right) \right) + C_1 \left(\Delta\Delta u^{k-1} - \Delta\Delta u^k \right) + C_2 \left(u^{k-1} - u^k \right),$$

where C_1 is comparable to $1/\delta$ and $C_2 \gg 1$. The scheme at each time level is

$$\frac{u^k - u^{k-1}}{\tau} = -\Delta \left(\nabla \cdot \left(\frac{\nabla u^{k-1}}{\|\nabla u^{k-1}\|_\delta} \right) \right) + C_1(\Delta\Delta u^{k-1} - \Delta\Delta u^k) + C_2(u^{k-1} - u^k).$$

We reorder the terms of our PDE to put the ones that depend on time level t_{k-1} on the right-hand side and the ones that correspond to time level t_k on the left-hand side. We obtain the problem that we have to solve for each k:

$$\begin{cases} C_1 \Delta\Delta u^k + (1/\tau + C_2)\, u^k &= g(u^{k-1}) & \text{in } \mathcal{D}, \\ u^k &= f & \text{on } \partial\mathcal{D}, \\ \nabla u^k \cdot \vec{n} &= \nabla f \cdot \vec{n} & \text{on } \partial\mathcal{D}, \end{cases} \tag{4.4}$$

where $g(u^{k-1}) = -\Delta \left(\nabla \cdot \left(\nabla u^{k-1} / \|\nabla u^{k-1}\|_\delta \right) \right) + C_1 \Delta\Delta u^{k-1} + (1/\tau + C_2)\, u^{k-1}$.

To solve problem (4.4) we apply the finite difference method. We suppose that the inpainting domain is a square $\mathcal{D} = (a,b) \times (c,d) \subset \Omega$. At each time level we take the same spatial grid that we used in Section 3.1.1 Numerical resolution, that is,

$$\mathcal{D}_h = \{(x_i, y_j) = (a + ih, c + jh) \mid i, j = 0, ..., n+1\}.$$

We denote by $u^k_{i,j}$ the approximation of $u(x_i, y_j)$ at time level t_k $(i, j = 0, ..., n+1,\ k = 0, ..., m+1)$. Our goal is to compute a spatial grid function

$$\{u^k_{0,0}, u^k_{0,1}, ..., u^k_{0,n+1}, u^k_{1,0}, ..., u^k_{1,n+1}, ..., u^k_{n+1,0}, ..., u^k_{n+1,n+1}\}$$

for $k = 1, ..., m+1$. For $k = 0$ the spatial grid function is given by the values of the pixels of $\mathcal{D}$ of the damaged image. The spatial grid function at t_{m+1} is the numerical solution of our problem.

At any time level t_k we know the value of some elements of the spatial grid function thanks to the Dirichlet boundary conditions ($u^k = f$ on $\partial\mathcal{D}$):

- $u^k_{0,j} = f(x_0, y_j)$, $j = 0, ..., n+1$;
- $u^k_{n+1,j} = f(x_{n+1}, y_j)$, $j = 0, ..., n+1$;
- $u^k_{i,0} = f(x_i, y_0)$, $i = 0, ..., n+1$;
- $u^k_{i,n+1} = f(x_i, y_{n+1})$, $i = 0, ..., n+1$.

These values coincide for any k.

The remaining $u^k_{i,j}$ correspond with the inner nodes. To compute them we have to discretize the left-hand side of the equation in (4.4). We also have to discretize the right-hand side in order to be able to evaluate it with the known spatial grid function of the previous time level. The idea is to solve a system of n^2 equations with n^2 unknowns of the form $A^k(u^k)^h = b^k$, where A^k is the matrix of coefficients obtained with the discretization of the left-hand side of the equation in (4.4) by the corresponding centered finite difference formulas in Definition A.1, $(u^k)^h$ is the vector of unknowns ordered in the same way as in Section 3.1.1 Numerical resolution and b^k is the vector of independent terms which is $g(u^{k-1})$ added to the known values of $u^k_{i,j}$ that appear because of the discretization of the left-hand side (these values are known because we have boundary conditions).

In the following we explain how to obtain A^k:

- We have to compute the discretization of $\Delta\Delta\hat{u}^k_1 = 0$ with the same boundary conditions as in (4.4). Our goal is to obtain a system of the form $A^k_1(\hat{u}^k_1)^h = b^k_1$ where $(\hat{u}^k_1)^h$ is the vector of unknowns ordered in the same way as $(u^k)^h$. The computations needed to obtain this system can be found in Appendix B. We denote the matrix of coefficients by A^k_1.
- We compute A^k as $A^k = C_1 A^k_1 + (1/\tau + C_2)\, I_{n^2}$.

The vector of independent terms b^k is obtained in the following way:

- When we have computed A^k_1 we have obtained a vector b^k_1 of independent terms (it is the vector of independent terms in Appendix B). It is part of the vector b^k.
- We have to discretize $g(\hat{u}) = -\Delta\left(\nabla \cdot (\nabla\hat{u}/\|\nabla\hat{u}\|_\delta)\right) + C_1\Delta\Delta\hat{u} + (1/\tau + C_2)\,\hat{u}$ and to evaluate it with the vector u^{k-1}.

  To compute the discretization of $g(\hat{u})$ we calculate the discretization of each term (last term does not need any discretization):

  - $-\Delta\left(\nabla \cdot (\nabla\hat{u}/\|\nabla\hat{u}\|_\delta)\right)$. To obtain the discretization of this term first of all we have to apply the discretization of the Laplacian (left-hand side of (3.5)) and then we have to apply the discretization of $\nabla \cdot (\nabla\hat{u}/\|\nabla\hat{u}\|_\delta)$ (it is equation in (3.10)) to each term of the previous discretization.

    If $i, j = 1, 2, n-1, n$, then Dirichlet or Neumann boundary conditions are needed.
  - $C_1\Delta\Delta\hat{u}$. To compute the discretization of this expression we use formula (B.2) obtained in the discretization of the Bilaplacian. Adjusting it to our case we have that the discretization we are looking for is

$$C_1(\hat{u}_{i+2,j} + 2\hat{u}_{i+1,j-1} - 8\hat{u}_{i+1,j} + 2\hat{u}_{i+1,j+1} + \hat{u}_{i,j-2} - 8\hat{u}_{i,j-1} + 20\hat{u}_{i,j} - 8\hat{u}_{i,j+1} + \\ +\hat{u}_{i,j+2} + 2\hat{u}_{i-1,j-1} - 8\hat{u}_{i-1,j} + 2\hat{u}_{i-1,j+1} + \hat{u}_{i-2,j})/h^4 \quad \text{for } i, j = 1, ..., n.$$

    If $i, j = 1, 2, n-1, n$, then Dirichlet or Neumann boundary conditions are needed.

- We compute b^k as $b^k = (\textit{discretization of } g(\hat{u}) \textit{ evaluated at } u^{k-1}) + C_1 b_1^k$.

We program the numerical resolution of this method with MATLAB (see Section C.4 of Appendix C). We use the picture `Damaged image` in Figure 4.2 whose inpainting domain is a grey square (see `Damaged image detail` in Figure 4.2). The damaged columns go from pixel 760 to pixel 770 and the damaged rows go from pixel 170 to pixel 180. Running the program, `CHGreyValues(760,770,170,180,100000)`, we obtain the image `Inpainted image` in Figure 4.2.

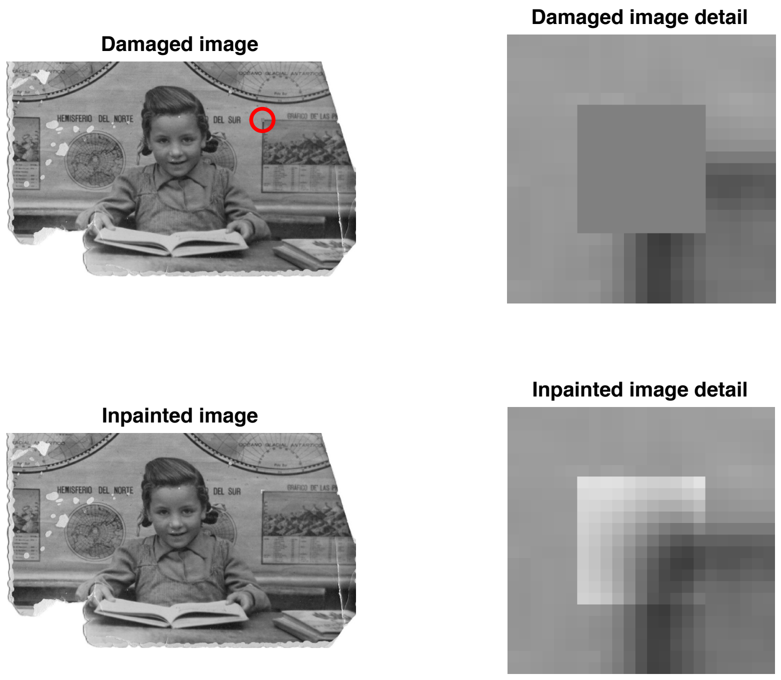

Figure 4.2: On the top we have a damaged image and we can see in detail its inpainting domain (grey square). On the bottom we have the inpainted image (it has been obtained applying TV-H^{-1} inpainting) and we can see in detail how the inpainting domain has been inpainted.

In picture `Inpainted image detail` in Figure 4.2 we can see the region of picture `Damaged image` in Figure 4.2 that has been inpainted. We can realize that this solution is better than the one obtained with TV inpainting (it is `Inpainted image detail` in Figure 3.5) because in the case of TV-H^{-1} inpainting we can observe that the darkest pixels form the right angle.

For more information about these two methods we refer to the reader to [10].

Appendix A

Mathematical facts

Definition A.1. Finite difference formulas.

Let $u(x, y)$ be a function sufficiently differentiable and let h be a number small enough.

The first partial derivative u_x at point (x_i, y_j) can be approximated by the centered finite difference formula

$$u_x(x_i, y_j) \approx \frac{1}{2h}\left(u(x_{i+1}, y_j) - u(x_{i-1}, y_j)\right),$$

and the first partial derivative u_y at that point can be approximated by the centered finite difference formula

$$u_y(x_i, y_j) \approx \frac{1}{2h}\left(u(x_i, y_{j+1}) - u(x_i, y_{j-1})\right).$$

The second partial derivate u_{xx} at point (x_i, y_j) can be approximated by the centered finite difference formula

$$u_{xx}(x_i, y_j) \approx \frac{1}{h^2}\left(u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)\right),$$

and the second partial derivative u_{yy} at that point can be approximated by the centered finite difference formula

$$u_{yy}(x_i, y_j) \approx \frac{1}{h^2}\left(u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})\right).$$

Let τ be a number small enough and let us suppose that the function u also depends on time, that is to say, $u(x, y, t)$. We can approximate the first partial derivative u_t at point (x_i, y_j, t_k) by

$$u_t(x_i, y_j, t_k) \approx \frac{1}{\tau}\left(u(x_i, y_j, t_k) - u(x_i, y_j, t_{k-1})\right).$$

Note: $x_{i+1} \equiv x_i + h,\ x_{i-1} \equiv x_i - h,\ y_{j+1} \equiv y_j + h,\ y_{j-1} \equiv y_j - h,\ t_{k-1} \equiv t_k - \tau.$

Definition A.2. Space of distributions.

Let $\Omega \subseteq \mathbb{R}^2$. We denote by $D(\Omega)$ the set of functions indefinitely differentiable with compact support in Ω.

The space of distributions on Ω is the topological dual space of $D(\Omega)$ (only for real functionals). It is denoted by $D'(\Omega)$.

Definition A.3. Partial derivative of a distribution ($\partial^\alpha u$).

Let $\Omega \subseteq \mathbb{R}^2$. Let $u \in D'(\Omega)$ and let $\alpha = (\alpha_1, ..., \alpha_n)$ be a multi-index such that $|\alpha| = \alpha_1 + ... + \alpha_n$. Then,

$$\langle \partial^\alpha u, \varphi \rangle := (-1)^{|\alpha|} \langle u, \partial^\alpha \varphi \rangle,\ \forall\ \varphi \in D(\Omega).$$

See Definition A.2 for the definitions of the spaces $D(\Omega)$ and $D'(\Omega)$.

Definition A.4. Sobolev space.

Let $\Omega \subseteq \mathbb{R}^2$. The Sobolev space $W^{m,p}(\Omega)$ with m a non-negative integer and $p \in [1, \infty)$ is

$$W^{m,p}(\Omega) := \{u \in \mathcal{L}^p(\Omega) \mid \partial^\alpha u \in \mathcal{L}^p(\Omega),\, \forall |\alpha| \leq m,\, \alpha \in \mathbb{N}\}.$$

It is a Banach space with the norm

$$\|u\|_{W^{m,p}(\Omega)} = \left[\sum_{0 \leq |\alpha| \leq m} \|\partial^\alpha u\|^p_{\mathcal{L}^p(\Omega)}\right]^{\frac{1}{p}}.$$

We denote by $W_0^{m,p}(\Omega)$ the subset of elements of $W^{m,p}(\Omega)$ which take the value 0 on the boundary (in a weak sense).

When $p = 2$ the Sobolev spaces are denoted by $\mathrm{H}^m(\Omega) = W^{m,2}(\Omega)$ and $\mathrm{H}_0^m(\Omega) = W_0^{m,2}(\Omega)$. These are Hilbert spaces.

Note: All the derivatives are taken in the sense of distributions (see Definition A.3).

Definition A.5. Green's function. [11]

Let $\Omega \subseteq \mathbb{R}^2$, $u : \Omega \to \mathbb{R}$ and $\boldsymbol{x}, \boldsymbol{s} \in \Omega$. Let $L[u] = f$ be a differential equation. The Green's function $G(\boldsymbol{x}, \boldsymbol{s})$ is any solution of $L[G] = -\delta_{\boldsymbol{x}-\boldsymbol{s}}$.

Theorem A.1. *Green's second formula.*

Let v_1 and v_2 be $\mathcal{C}^2$ functions defined on the set $\Omega \subseteq \mathbb{R}^2$. The boundary of Ω is denoted by $\partial\Omega$. Let $\vec{n}$ be the outward normal direction on $\partial\Omega$ and let s be the length parameter. Then,

$$\int_\Omega (v_1 \Delta v_2 - v_2 \Delta v_1) \mathrm{d}x \mathrm{d}y = \int_{\partial\Omega} \left(v_1 \frac{\partial v_2}{\partial \vec{n}} - v_2 \frac{\partial v_1}{\partial \vec{n}}\right) \mathrm{d}s.$$

Definition A.6. Locally integrable.

A function f defined on a set Ω is called locally integrable ($f \in \mathcal{L}^1_{loc}(\Omega)$) if it is integrable on each compact of Ω.

Theorem A.2. *Inverse function theorem.*

Let $\Omega \subseteq \mathbb{R}^2$ be an open set, $f : \Omega \to \mathbb{R}$ such that $f \in \mathcal{C}^p(\Omega)$ $(1 \leq p \leq \infty)$ and $\boldsymbol{x}_0 \in \Omega$ such that $\nabla f(\boldsymbol{x}_0) \neq \boldsymbol{0}$. Then, there exist two open sets $U \subseteq \mathbb{R}^2$ and $V \subseteq \mathbb{R}$ such that $\boldsymbol{x}_0 \in U \subseteq \Omega$, $f(\boldsymbol{x}_0) \in V \subseteq f(\Omega)$ and $f_{|_U} : U \to V$ is bijective.

More mathematical facts related to PDEs that can be useful to understand this work can be found in [12], [13] and [3, Appendix B].

Appendix B

Discretization of the Bilaplacian

We have the following problem:

$$\begin{cases} \Delta\Delta u &= 0 & \text{in } \mathcal{D}, \\ u &= f & \text{on } \partial\mathcal{D}, \\ \nabla u \cdot \vec{n} &= \nabla f \cdot \vec{n} & \text{on } \partial\mathcal{D}. \end{cases} \tag{B.1}$$

To solve it we apply the finite difference method.

We suppose that the inpainting domain $\mathcal{D} \subset \Omega$ is a square $(a,b) \times (c,d)$. We take a mesh as the one used in Section 3.1.1 Numerical resolution, that is to say,

$$\mathcal{D}_h = \{(x_i, y_j) = (a + ih, c + jh) \mid i, j = 0, ..., n+1\}.$$

Our goal is to obtain a grid function

$$\{u_{0,0}, u_{0,1}, ..., u_{0,n+1}, u_{1,0}, ..., u_{1,n+1}, ..., u_{n+1,0}, ..., u_{n+1,n+1}\}$$

such that $u_{i,j} \approx u(x_i, y_j)$ for $i, j = 0, ..., n+1$.

Realize that we have Dirichlet boundary conditions ($u = f$ on $\partial\mathcal{D}$), so we already know the values of some elements of the grid function. That is, we know that

- $u_{0,j} = f(x_0, y_j)$, $j = 0, ..., n+1$;
- $u_{n+1,j} = f(x_{n+1}, y_j)$, $j = 0, ..., n+1$;
- $u_{i,0} = f(x_i, y_0)$, $i = 0, ..., n+1$;
- $u_{i,n+1} = f(x_i, y_{n+1})$, $i = 0, ..., n+1$.

To obtain the remaining nodes, that is, the inner nodes, we apply the corresponding centered finite difference formulas in Definition A.1 two times, obtaining a formula which involves 13 nodes. Let us do it.

We know that

$$\Delta\Delta u = \frac{\partial^2 \Delta u}{\partial x^2} + \frac{\partial^2 \Delta u}{\partial y^2}.$$

We apply the corresponding centered finite difference formulas in Definition A.1 one time and we obtain the following approximation for $i, j = 1, ..., n$:

$$\frac{1}{h^2}\left(\Delta_h u_{i+1,j} + \Delta_h u_{i-1,j} + \Delta_h u_{i,j+1} + \Delta_h u_{i,j-1} - 4\Delta_h u_{i,j}\right) = 0.$$

Now we apply the corresponding centered finite difference formulas in Definition A.1 to each term and we have

$$\frac{1}{h^4}(u_{i+2,j} + 2u_{i+1,j-1} - 8u_{i+1,j} + 2u_{i+1,j+1} + u_{i,j-2} - 8u_{i,j-1} + 20u_{i,j} - 8u_{i,j+1} + u_{i,j+2} +$$
$$+ 2u_{i-1,j-1} - 8u_{i-1,j} + 2u_{i-1,j+1} + u_{i-2,j}) = 0 \tag{B.2}$$

for $i, j = 1, ..., n$. As we said, 13 nodes are involved, so the scheme can be represented by a thirteen-point stencil (see Figure B.1).

We have a linear system of n^2 equations with n^2 unknowns. Ordering the unknowns in the same way as we have done in Section 3.1.1 Numerical resolution, we can write the system as $Au^h = b$. Note that each equation involves 13 nodes, this implies that each row of the matrix A has at most 13 non-zero elements.

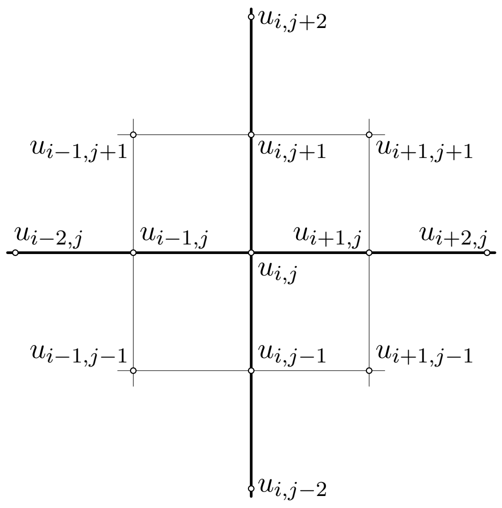

Figure B.1: Graphic representation of the thirteen-point stencil

Realize that when we apply the equation with these indices

- $i = 2, ..., n-1,\ j = 1$;
- $i = 2, ..., n-1,\ j = n$;
- $i = 1,\ j = 2, ..., n-1$;
- $i = n,\ j = 2, ..., n-1$;

we need a ghost node which is out of our domain. When we use these other indices

- $i = 1,\ j = 1$;
- $i = n,\ j = 1$;
- $i = 1,\ j = n$;
- $i = n,\ j = n$;

we need two ghost nodes which are outside the domain. Using the Neumann boundary conditions ($\nabla u \cdot \vec{n} = \nabla f \cdot \vec{n}$ on $\partial\mathcal{D}$) we can obtain the expressions of the ghost nodes in terms of the nodes of our mesh and of the derivatives of f (we have to use the corresponding centered finite difference formulas in Definition A.1 for the u derivatives and we have to isolate the ghost nodes). These expressions are:

- $u_{i,-1} = u_{i,1} - 2hf_y(x_i, y_0),\ i = 0, ..., n+1$;
- $u_{i,n+2} = u_{i,n} + 2hf_y(x_i, y_{n+1}),\ i = 0, ..., n+1$;
- $u_{-1,j} = u_{1,j} - 2hf_x(x_0, y_j),\ j = 0, ..., n+1$;
- $u_{n+2,j} = u_{n,j} + 2hf_x(x_{n+1}, y_j),\ j = 0, ..., n+1$.

To sum up, using formula (B.2) for $i, j = 1, ..., n$, applying the formulas for the ghost nodes when the special values of i and j are involved and moving the known values ($u_{i,j}$ of the boundary and evaluations of the functions f_x and f_y) to the right-hand side we obtain (ordering the vector of unknowns as always) that the matrix A is

$$A = \frac{1}{h^4}\begin{bmatrix} S_1 & T & I & & & & & \\ T & S_2 & T & I & & & & \\ I & T & S_2 & T & I & & & \\ & I & T & S_2 & T & I & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & I & T & S_2 & T & I \\ & & & & I & T & S_2 & T \\ & & & & & I & T & S_1 \end{bmatrix},$$

where I is the $n \times n$ identity matrix, T is a tridiagonal $n \times n$ matrix of the form

$$T = \begin{bmatrix} -8 & 2 & & & \\ 2 & -8 & 2 & & \\ & \ddots & \ddots & \ddots & \\ & & 2 & -8 & 2 \\ & & & 2 & -8 \end{bmatrix},$$

and S_1, S_2 are $n \times n$ matrices as follows

$$S_2 = S_1 - I = \begin{bmatrix} 21 & -8 & 1 & & & & \\ -8 & 20 & -8 & 1 & & & \\ 1 & -8 & 20 & -8 & 1 & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & 1 & -8 & 20 & -8 & 1 \\ & & & 1 & -8 & 20 & -8 \\ & & & & 1 & -8 & 21 \end{bmatrix}.$$

The vector of independent terms b is

$$b = -\frac{1}{h^4}\begin{bmatrix} b^{[1]} \\ b^{[2]} \\ b^{[3]} \\ \vdots \\ b^{[m]} \\ \vdots \\ b^{[n-2]} \\ b^{[n-1]} \\ b^{[n]} \end{bmatrix},$$

where $b^{[1]}$, $b^{[2]}$, $b^{[m]}$ $(m = 3, ..., n-2)$, $b^{[n-1]}$, $b^{[n]}$ are vectors of dimension n that can be written as

$$b^{[1]} = \begin{bmatrix} 2u_{2,0} - 8u_{1,0} + 2u_{0,0} - 8u_{0,1} + 2u_{0,2} - 2h(f_y(x_1, y_0) + f_x(x_0, y_1)) \\ 2u_{3,0} - 8u_{2,0} + 2u_{1,0} + u_{0,1} - 2hf_y(x_2, y_0) \\ 2u_{4,0} - 8u_{3,0} + 2u_{2,0} - 2hf_y(x_3, y_0) \\ \vdots \\ 2u_{n-1,0} - 8u_{n-2,0} + 2u_{n-3,0} - 2hf_y(x_{n-2}, y_0) \\ u_{n+1,1} + 2u_{n,0} - 8u_{n-1,0} + 2u_{n-2,0} - 2hf_y(x_{n-1}, y_0) \\ 2u_{n+1,0} - 8u_{n+1,1} + 2u_{n+1,2} - 8u_{n,0} + 2u_{n-1,0} - 2h(f_y(x_n, y_0) - f_x(x_{n+1}, y_1)) \end{bmatrix},$$

$$b^{[2]} = \begin{bmatrix} u_{1,0} + 2u_{0,1} - 8u_{0,2} + 2u_{0,3} - 2hf_x(x_0, y_2) \\ \\ u_{2,0} + u_{0,2} \\ \\ u_{3,0} \\ \vdots \\ u_{n-2,0} \\ \\ u_{n+1,2} + u_{n-1,0} \\ \\ 2u_{n+1,1} - 8u_{n+1,2} + 2u_{n+1,3} + u_{n,0} + 2hf_x(x_{n+1}, y_2) \end{bmatrix},$$

$$b^{[m]} = \begin{bmatrix} 2u_{0,m-1} - 8u_{0,m} + 2u_{0,m+1} - 2hf_x(x_0, y_m) \\ \\ u_{0,m} \\ \\ 0 \\ \vdots \\ 0 \\ \\ u_{n+1,m} \\ \\ 2u_{n+1,m-1} - 8u_{n+1,m} + 2u_{n+1,m+1} + 2hf_x(x_{n+1}, y_m) \end{bmatrix},$$

$$b^{[n-1]} = \begin{bmatrix} u_{1,n+1} + 2u_{0,n} - 8u_{0,n-1} + 2u_{0,n-2} - 2hf_x(x_0, y_{n-1}) \\ \\ u_{2,n+1} + u_{0,n-1} \\ \\ u_{3,n+1} \\ \vdots \\ u_{n-2,n+1} \\ \\ u_{n+1,n-1} + u_{n-1,n+1} \\ \\ 2u_{n+1,n-2} - 8u_{n+1,n-1} + 2u_{n+1,n} + u_{n,n+1} + 2hf_x(x_{n+1}, y_{n-1}) \end{bmatrix},$$

$$b^{[n]} = \begin{bmatrix} 2u_{2,n+1} - 8u_{1,n+1} + 2u_{0,n-1} - 8u_{0,n} + 2u_{0,n+1} - \\ \qquad -2h(f_x(x_0, y_n) - f_y(x_1, y_{n+1})) \\ \\ 2u_{3,n+1} - 8u_{2,n+1} + 2u_{1,n+1} + u_{0,n} + 2hf_y(x_2, y_{n+1}) \\ \\ 2u_{4,n+1} - 8u_{3,n+1} + 2u_{2,n+1} + 2hf_y(x_3, y_{n+1}) \\ \vdots \\ 2u_{n-1,n+1} - 8u_{n-2,n+1} + 2u_{n-3,n+1} + 2hf_y(x_{n-2}, y_{n+1}) \\ \\ u_{n+1,n} + 2u_{n,n+1} - 8u_{n-1,n+1} + 2u_{n-2,n+1} + 2hf_y(x_{n-1}, y_{n+1}) \\ \\ 2u_{n+1,n-1} - 8u_{n+1,n} + 2u_{n+1,n+1} - 8u_{n,n+1} + 2u_{n-1,n+1} + \\ \qquad +2h(f_y(x_n, y_{n+1}) + f_x(x_{n+1}, y_n)) \end{bmatrix}.$$

The numerical solution of problem (B.1) is the solution of the system $Au^h = b$ with the matrix A and the vector b described above.

Appendix C

Numerical resolution with MATLAB

In this appendix we can find several MATLAB programs. Given a damaged image, these programs allow us to obtain the inpainted one. We only have to indicate the damaged pixels, that is, the inpainting domain (it has to be a square) and, in the case that the PDE is solved as an evolution problem, a parameter m related to the number of subdivisions of the temporary grid has to be introduced.

In the first four sections we have the main programs needed to apply harmonic inpainting, TV inpainting, Cahn-Hilliard inpainting for binary images and TV-H^{-1} inpainting. In the last section there are auxiliary functions used in the programs presented in the previous sections. In each program there are an introduction about its running and some complementary explanations.

C.1 Harmonic inpainting. Numerical resolution with MATLAB

```
function harmonicInpainting(a1,b1,c1,d1)

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  harmonicInpainting solves the inpainting method known as harmonic
                                                              %  inpainting.

%  It has four inputs:
        %  -> a1 is the first column of pixels which is damaged.
        %  -> b1 is the last column of pixels which is damaged.
        %  -> c1 is the first row of pixels which is damaged.
        %  -> d1 is the last row of pixels which is damaged.

%  It returns the damaged image and the inpainted one.

%  NOTE: We suppose that our inpainting domain is a square, so b1-a1 =
                                                                 %  = d1-c1.
%  The inpainting domain with its boundary is a submatrix of the matrix of
        %  the image. The indices of the rows (respect to the image matrix)
           %  are c1-1,...,d1+1. The indices of the columns (respect to the
                                             %  image matrix) are a1-1,...,b1+1.

%  n is the number of pixels which are damaged in each spatial direction.
global n;
n = b1-a1+1;
%  n can also be computed as n = d1-c1+1.
%  As we work with images, the usual step of the spatial grid is h = 1
                       %  in both directions (x-direction and y-direction).
h = 1;

%  We read the original image.
f = double(imread('IMAGE.jpg'))/255;

%  We damage the image.
f(c1:d1,a1:b1) = 1;

%  In our problem we have Dirichlet boundary conditions, so we store the
   %  values of the pixels of the boundary in a vector D of dimension 4n+4.
global D;
for i =1:n+2
    D(i) = f(d1+1,a1+i-2);
end
for i = 0:n-1
    D(n+2+2*i+1) = f(d1-i,a1-1);
    D(n+2+2*i+2) = f(d1-i,b1+1);
end
for i = 1:n+2
    D(3*n+2+i) = f(c1-1,a1+i-2);
end

%  A is a matrix with a lot of zeros, so it is sparse:
A = sparse(n*n, n*n);

%  aux represents a number which is used a lot in the program:
aux = 1/(h^2);

%  We start filling the matrix A:

%%Diagonal of the matrix.
for i = 1:n*n
    A(i,i) = -4*aux;
end

%%Elements above and below the diagonal.
for ind = 0:n-1
    for i = 1:n-1
        A(i+n*ind,(i+1)+n*ind) = aux;
        A((i+1)+n*ind,i+n*ind) = aux;
    end
end

%%Elements of the diagonal sub-matrices.
for ind = 0:n-2
    for i = 1:n
        A(i+n*ind,(i+n*ind)+n) = aux;
        A((i+n*ind)+n,i+n*ind) = aux;
    end
end

%  We fill the vector b (independent terms):
b = zeros(1,n*n);

%%1. Independent terms which correspond with the n first elements of
                                                               %%the vector b.
b(1) = -aux*D(n+3)-aux*D(2);
b(n) = -aux*D(n+4)-aux*D(n+1);
for i = 2:n-1
    b(i) = -aux*D(i+1);
end

%%2,...,n-1. Independent terms which correspond with the middle elements
                                                             %%of the vector b.
for j = 2:n-1
    b((j-1)*n+1) = -aux*D(n+3+2*(j-1));
    b(j*n) = -aux*D(n+2+2*j);
end

%%n. Independent terms which correspond with the n last elements of
                                                               %%the vector b.
b(n*(n-1)+1) = -aux*D(3*n+1)-aux*D(3*n+4);
b(n*n) = -aux*D(3*n+2)-aux*D(4*n+3);
for i = 2:n-1
    b((n-1)*n+i) = -aux*D(3*n+3+i);
end

%  Now we solve the system Au=b:
u(1:n*n)=A(1:n*n,1:n*n)\b(1:n*n)';

%  We have that u is a vector of dimension n*n, we have to convert it
                        %  into a square matrix with n rows and n columns:
u = reshape(u,n,n);

%  The order in which we have stored the values of the grid function is not
          %  the order in which we have to print them, we make the change:
u = u';
ulast = zeros(n,n);

for i = 1:n
    for j = 1:n
        ulast(n-(i-1),j) = u(i,j);
    end
end

%  We print the damaged image and the inpainted one:
uinpainting = f;
uinpainting(c1:d1,a1:b1) = ulast;
subplot(2,2,1); imshow(f); title 'Damaged image';
subplot(2,2,3); imshow(uinpainting); title 'Inpainted image';
subplot(2,2,2); imshow(f(c1-6:d1+6,a1-6:b1+6)); title 'Damaged image
                                                                detail';
subplot(2,2,4); imshow(uinpainting(c1-6:d1+6,a1-6:b1+6)); title 'Inpainted
                                                          image detail';

end
```

C.2 TV inpainting. Numerical resolution with MATLAB

```
function TVInpainting(a1,b1,c1,d1,m)

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  TVInpainting solves the inpainting method known as TV inpainting.

%  It has five inputs:
        %  -> a1 is the first column of pixels which is damaged.
        %  -> b1 is the last column of pixels which is damaged.
        %  -> c1 is the first row of pixels which is damaged.
        %  -> d1 is the last row of pixels which is damaged.
        %  -> m, being m+1 the number of subdivisions of the temporary grid.

%  It returns the damaged image and the inpainted one.

%  This program uses an auxiliary function:
        %  -> auxiliary.

%  NOTE: We suppose that our inpainting domain is a square, so b1-a1 =
                                                              %  = d1-c1.
%  The inpainting domain with its boundary is a submatrix of the matrix
 %  of the image. The indices of the rows (respect to the image matrix) are
 %  c1-1,...,d1+1. The indices of the columns (respect to the image matrix)
                                                       %  are a1-1,...,b1+1.

%  n is the number of pixels which are damaged in each spatial direction.
global n;
n = b1-a1+1;
%  n can also be computed as n = d1-c1+1.

%  As we work with images, the usual step of the spatial grid is h = 1
                       %  in both directions (x-direction and y-direction).
global h;
h = 1;

%  k is the step of the temporary grid.
k = 0.01;

%  The value of delta (the term used to avoid problems with zero in the
                                                         % denominator) is:
global delta;
```

```
delta = 0.01;

%  We read the original image.
f = double(imread('IMAGE.jpg'))/255;

%  We damage the image.
f(c1:d1,a1:b1) = 1;

%  We store the initial condition.
g = f(c1:d1,a1:b1);

%  The order in which we have read the pixels is not the order in which
       %  we work with the values of the grid function, we make the change:
unew = zeros(n,n);

for i = 1:n
    for j = 1:n
        unew(n-(i-1),j) = g(i,j);
    end
end

unew = unew';

%  In our problem we have Dirichlet boundary conditions, so we store the
   %  values of the pixels of the boundary in a vector D of dimension 4n+4.
global D;
for i =1:n+2
    D(i) = f(d1+1,a1+i-2);
end
for i = 0:n-1
    D(n+2+2*i+1) = f(d1-i,a1-1);
    D(n+2+2*i+2) = f(d1-i,b1+1);
end
for i = 1:n+2
    D(3*n+2+i) = f(c1-1,a1+i-2);
end

%  We solve the evolution problem.
global uold;

for time = 1:m+1
    uold = unew;
    for j = 1:n
        for i = 1:n
            unew(i,j) = k*auxiliary(i,j) + uold(i,j);
        end
    end
end

%  The last unew (that is the one that solves our problem) is a matrix, but
%  the order in which we have stored the values of the grid function is not
```

```
          %  the order in which we have to print them, we make the change:
unew = unew';

ulast = zeros(n,n);

for i = 1:n
    for j = 1:n
        ulast(n-(i-1),j) = unew(i,j);
    end
end

%  We print the damaged image and the inpainted one:
uinpainting = f;
uinpainting(c1:d1,a1:b1) = ulast;
subplot(2,2,1); imshow(f); title 'Damaged image';
subplot(2,2,3); imshow(uinpainting); title 'Inpainted image';
subplot(2,2,2); imshow(f(c1-2:d1+2,a1-2:b1+2)); title 'Damaged image
                                                                 detail';
subplot(2,2,4); imshow(uinpainting(c1-2:d1+2,a1-2:b1+2)); title 'Inpainted
                                                           image detail';

end
```

C.3 Cahn-Hilliard inpainting for binary images. Numerical resolution with MATLAB

```
function CHBinary(a1,b1,c1,d1,m)

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  CHBinary solves the inpainting method known as Cahn-Hilliard inpainting
                                                          %  for binary images.

%  It has five inputs:
        %  -> a1 is the first column of pixels which is damaged.
        %  -> b1 is the last column of pixels which is damaged.
        %  -> c1 is the first row of pixels which is damaged.
        %  -> d1 is the last row of pixels which is damaged.
        %  -> m, being m+1 the number of subdivisions of the temporary grid.

%  It returns the damaged image and the inpainted one.

%  This program uses the following auxiliary functions:
        %  -> laplacian.
        %  -> bilaplacianA.
        %  -> bilaplacianb.
        %  -> laplacianF.

%  NOTE: We suppose that our inpainting domain is a square, so b1-a1 =
                                                                %  = d1-c1.
% The inpainting domain with its boundary is a submatrix of the matrix of
    % the image. The indices of the rows (respect to the image matrix) are
 % c1-1,...,d1+1. The indices of the columns (respect to the image matrix)
                                                      % are a1-1,...,b1+1.

% n is the number of pixels which are damaged in each spatial direction.
global n;
n = b1-a1+1;
% n can also be computed as n = d1-c1+1.

% As we work with images, the usual step of the spatial grid is h = 1
                       % in both directions (x-direction and y-direction).
global h;
h = 1;

% k is the step of the temporary grid.
k = 0.01;

% Constants.
C1 = 3/4;
C2 = 2;
epsilon = 4;

% We read the original image.
f = double(imread('IMAGE010.jpg'))/255;

% We damage the image.
f(c1:d1,a1:b1) = 1;

% We store the initial condition.
g = f(c1:d1,a1:b1);

% The order in which we have read the pixels is not the order in which
       % we work with the values of the grid function, we make the change:
unew = zeros(n,n);

for i = 1:n
    for j = 1:n
        unew(n-(i-1),j) = g(i,j);
    end
end

unew = unew';

unew = reshape(unew,[],1);

% In our problem we have Dirichlet boundary conditions, so we store the
   % values of the pixels of the boundary in a vector D of dimension 4n+4.
global D;
for i =1:n+2
    D(i) = f(d1+1,a1+i-2);
```

```
end
for i = 0:n-1
    D(n+2+2*i+1) = f(d1-i,a1-1);
    D(n+2+2*i+2) = f(d1-i,b1+1);
end
for i = 1:n+2
    D(3*n+2+i) = f(c1-1,a1+i-2);
end

%  We also have Neumann boundary conditions, so we have to approximate the
     %  derivatives of the damaged image on the boundary of the inpainting
                 %  domain by finite difference formulas. For that we need
                              %  the pixels that surround the boundary.
global Neumann;
Neumann = zeros(4*n+8,1);
for i = 1:n+2
    Neumann(i) = f(d1+2,a1+i-2);
end
for i = 0:n+1
    Neumann(n+3+2*i) = f(d1-j+1,a1-2);
    Neumann(n+4+2*i) = f(d1-j+1,b1+2);
end
for i = 1:n+2
    Neumann(3*n+6+i) = f(c1-2,a1+i-2);
end

%  We solve the problem.
global uold;

[mlapl,vlapl] = laplacian();
mbilapl = bilaplacianA();
vbilapl = bilaplacianb();

for j = 1:m+1
    uold = unew;
    dlaplFder = laplacianF();
    matrixA = epsilon*mbilapl-C1*mlapl+((1/k)+C2)*eye(n*n);
    vectorb = epsilon*vbilapl'-C1*vlapl'+(1/epsilon)*dlaplFder
                              +(C2+(1/k))*uold-C1*(mlapl*uold-vbilapl');
    unew = matrixA\vectorb;
end

%  We have that unew is a vector of dimension n*n, we have to convert it
                        %  into a square matrix with n rows and n columns:
unew = reshape(unew,n,n);

%  The order in which we have stored the values of the grid function is not
           %  the order in which we have to print them, we make the change:
unew = unew';

ulast = zeros(n,n);
```

```
for i = 1:n
    for j = 1:n
        ulast(n-(i-1),j) = unew(i,j);
    end
end

% We print the damaged image and the inpainted one:
uinpainting = f;
uinpainting(c1:d1,a1:b1) = ulast;
subplot(2,2,1); imshow(f); title 'Damaged image';
subplot(2,2,3); imshow(uinpainting); title 'Inpainted image';
subplot(2,2,2); imshow(f(c1-14:d1+14,a1-14:b1+14)); title 'Damaged image
                                                                   detail';
subplot(2,2,4); imshow(uinpainting(c1-14:d1+14,a1-14:b1+14));
                                            title 'Inpainted image detail';

end
```

C.4 TV-H^{-1} inpainting. Numerical resolution with MATLAB

```
function CHGreyValues(a1,b1,c1,d1,m)

% Name: CARMEN MAYORA CEBOLLERO
% Date: 2020
% CHGreyValues solves the inpainting method known as TV-H^{-1} inpainting.

% It has five inputs:
        % -> a1 is the first column of pixels which is damaged.
        % -> b1 is the last column of pixels which is damaged.
        % -> c1 is the first row of pixels which is damaged.
        % -> d1 is the last row of pixels which is damaged.
        % -> m, being m+1 the number of subdivisions of the temporary grid.

% It returns the damaged image and the inpainted one.

% This program uses the following auxiliary functions:
        % -> bilaplacianA.
        % -> bilaplacianb.
        % -> laplacianaux.

% NOTE: We suppose that our inpainting domain is a square, so b1-a1 =
                                                              % = d1-c1.
% The inpainting domain with its boundary is a submatrix of the matrix of
    % the image. The indices of the rows (respect to the image matrix) are
 % c1-1,...,d1+1. The indices of the columns (respect to the image matrix)
                                                    % are a1-1,...,b1+1.

% n is the number of pixels which are damaged in each spatial direction.
global n;
n = b1-a1+1;
```

```
% n can also be computed as n = d1-c1+1.

% As we work with images, the usual step of the spatial grid is h = 1
                       % in both directions (x-direction and y-direction).
global h;
h = 1;

% k is the step of the temporary grid.
k = 0.001;

% Constants.
C1 = 0.0001;
C2 = 2;

% The value of delta is:
global delta;
delta = 0.1;

% We read the original image.
f = double(imread('IMAGE.jpg'))/255;

% We damage the image.
f(c1:d1,a1:b1)=0.5;

% We store the initial condition.
g = f(c1:d1,a1:b1);

% The order in which we have read the pixels is not the order in which
       % we work with the values of the grid function, we make the change:
unew = zeros(n,n);

for i = 1:n
    for j = 1:n
        unew(n-(i-1),j) = g(i,j);
    end
end

unew = unew';

unew = reshape(unew,[],1);

% In our problem we have Dirichlet boundary conditions, so we store the
   % values of the pixels of the boundary in a vector D of dimension 4n+4.
global D;
for i =1:n+2
    D(i) = f(d1+1,a1+i-2);
end
for i = 0:n-1
    D(n+2+2*i+1) = f(d1-i,a1-1);
    D(n+2+2*i+2) = f(d1-i,b1+1);
end
```

```
for i = 1:n+2
    D(3*n+2+i) = f(c1-1,a1+i-2);
end

% We also have Neumann boundary conditions, so we have to approximate the
     % derivatives of the damaged image on the boundary of the inpainting
                 % domain by finite difference formulas. For that we need
                               % the pixels that surround the boundary.
global Neumann;
Neumann = zeros(4*n+8,1);
for i = 1:n+2
    Neumann(i) = f(d1+2,a1+i-2);
end
for i = 0:n+1
    Neumann(n+3+2*i) = f(d1-j+1,a1-2);
    Neumann(n+4+2*i) = f(d1-j+1,b1+2);
end
for i = 1:n+2
    Neumann(3*n+6+i) = f(c1-2,a1+i-2);
end

% We solve the problem.
mbilapl = bilaplacianA();
vbilapl = bilaplacianb();

global uold;

for j = 1:m+1
    uold = unew;
    uold = reshape(uold,n,n);
    vlaplaux = laplacianaux();
    uold = reshape(uold,[],1);
    matrixA = ((1/k)+C2)*eye(n*n) + C1*mbilapl;
    vectorb = C1*(mbilapl*uold-vbilapl') - vlaplaux + (C2+(1/k))*uold
                                                           + vbilapl';
    unew = matrixA\vectorb;
end

% We have that unew is a vector of dimension n*n, we have to convert it
                         % into a square matrix with n rows and n columns:
unew = reshape(unew,n,n);

% The order in which we have stored the values of the grid function is not
          % the order in which we have to print them, we make the change:
unew = unew';

ulast = zeros(n,n);

for i = 1:n
    for j = 1:n
        ulast(n-(i-1),j) = unew(i,j);
    end
end

%  We print the damaged image and the inpainted one:
uinpainting = f;
uinpainting(c1:d1,a1:b1) = ulast;
subplot(2,2,1); imshow(f); title 'Damaged image';
subplot(2,2,3); imshow(uinpainting); title 'Inpainted image';
subplot(2,2,2); imshow(f(c1-6:d1+6,a1-6:b1+6)); title 'Damaged image
                                                                 detail';
subplot(2,2,4); imshow(uinpainting(c1-6:d1+6,a1-6:b1+6)); title 'Inpainted
                                                            image detail';

end
```

C.5 Auxiliary programs

res = auxiliary(a,b)

```
function res = auxiliary(a,b)

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program computes the evaluation of the discretization of
               %  div(grad(u)/normdelta(grad(u))) at a point of the grid.

%  It has two inputs:
          % -> a is the index of the x-coordinate.
          % -> b is the index of the y-coordinate.

%  This program uses one auxiliary function:
      %  -> N.

%  This auxiliary program is used in the program TVInpainting and in
                                    %  the auxiliary function laplacianaux.

global n;
global h;
global delta;
global D;
global uold;

if 2<=a && a<=n-1 && 2<=b && b<=n-1
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                   +uold(a+1,b+1)-uold(a,b-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)
                     +uold(a,b+1)-uold(a-1,b-1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(uold(a+1,b+1)
                                         -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                          -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==1 && 2<=b && b<=n-1
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                 +uold(a+1,b+1)-uold(a,b-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-D(n+3+2*(b-1)))^2)+(((1/2)*(D(n+3+2*b)
                 +uold(a,b+1)-D(n+3+2*(b-2))-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*D(n+3+2*(b-1));
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(uold(a+1,b+1)
                                             -D(n+3+2*b)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                         -D(n+3+2*(b-2))))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n && 2<=b && b<=n-1
    kdpa = h/sqrt(((D(n+4+2*(b-1))-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                  +D(n+4+2*b)-uold(a,b-1)-D(n+4+2*(b-2))))^2)+delta*(h^2));
    sumap1b = kdpa*D(n+4+2*(b-1));
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)
                  +uold(a,b+1)-uold(a-1,b-1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(D(n+4+2*b)
                                          -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(D(n+4+2*(b-2))
                                          -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif b==1 && 2<=a && a<=n-1
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                            +uold(a+1,b+1)-D(a+1)-D(a+2)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)
                                +uold(a,b+1)-D(a)-D(a+1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(uold(a+1,b+1)
                                          -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-D(a+1))^2)+(((1/2)*(D(a+2)
                                                  -D(a)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(a+1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif b==n && 2<=a && a<=n-1
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(D(3*n+3+a)
                       +D(3*n+4+a)-uold(a,b-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(D(3*n+2+a)
                       +D(3*n+3+a)-uold(a-1,b-1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((D(3*n+3+a)-uold(a,b))^2)+(((1/2)*(D(3*n+4+a)
                                                 -D(3*n+2+a)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(3*n+3+a);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                              -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==1 && b==1
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                                  +uold(a+1,b+1)-D(2)-D(3)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-D(n+3))^2)+(((1/2)*(D(n+5)
                                    +uold(a,b+1)-D(1)-D(2)))^2)+delta*(h^2));
    sumam1b = kdma*D(n+3);
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(uold(a+1,b+1)
                                                     -D(n+5)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-D(2))^2)+(((1/2)*(D(3)
                                                       -D(1)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(2);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==1 && b==n
    kdpa = h/sqrt(((uold(a+1,b)-uold(a,b))^2)+(((1/2)*(D(3*n+4)
                         +D(3*n+5)-uold(a,b-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((uold(a,b)-D(3*n+1))^2)+(((1/2)*(D(3*n+3)
                              +D(3*n+4)-D(3*n-1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*D(3*n+1);
    kdpb = h/sqrt(((D(3*n+4)-uold(a,b))^2)+(((1/2)*(D(3*n+5)
                                                   -D(3*n+3)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(3*n+4);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                                   -D(3*n-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);
```

```
elseif a==n && b==1
    kdpa = h/sqrt(((D(n+4)-uold(a,b))^2)+(((1/2)*(uold(a,b+1)
                                       +D(n+6)-D(n+1)-D(n+2)))^2)+delta*(h^2));
    sumap1b = kdpa*D(n+4);
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)
                                    +uold(a,b+1)-D(n)-D(n+1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((uold(a,b+1)-uold(a,b))^2)+(((1/2)*(D(n+6)
                                              -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((uold(a,b)-D(n+1))^2)+(((1/2)*(D(n+2)
                                                      -D(n)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(n+1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n && b==n
    kdpa = h/sqrt(((D(3*n+2)-uold(a,b))^2)+(((1/2)*(D(4*n+3)
                                +D(4*n+4)-uold(a,b-1)-D(3*n)))^2)+delta*(h^2));
    sumap1b = kdpa*D(3*n+2);
    kdma = h/sqrt(((uold(a,b)-uold(a-1,b))^2)+(((1/2)*(D(4*n+2)
                         +D(4*n+3)-uold(a-1,b-1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((D(4*n+3)-uold(a,b))^2)+(((1/2)*(D(4*n+4)
                                                  -D(4*n+2)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(4*n+3);
    kdmb = h/sqrt(((uold(a,b)-uold(a,b-1))^2)+(((1/2)*(D(3*n)
                                              -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*uold(a,b);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==0 && 2<=b && b<=n-1
    kdpa = h/sqrt(((uold(a+1,b)-D(n+3+2*(b-1)))^2)+(((1/2)*(D(n+3+2*b)
                 +uold(a+1,b+1)-D(n+3+2*(b-2))-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((D(n+3+2*(b-1))-(-2*h*N(0,b,1)+uold(1,b)))^2)
                            +(((1/2)*((-2*h*N(0,b+1,1)+uold(1,b+1))+D(n+3+2*b)
                -(-2*h*N(0,b-1,1)+uold(1,b-1))-D(n+3+2*(b-2))))^2)+delta*(h^2));
    sumam1b = kdma*(-2*h*N(0,b,1)+uold(1,b));
    kdpb = h/sqrt(((D(n+3+2*b)-D(n+3+2*(b-1)))^2)+(((1/2)*(uold(a+1,b+1)
                             -(-2*h*N(0,b+1,1)+uold(1,b+1))))^2)+delta*(h^2));
    sumabp1 = kdpb*D(n+3+2*b);
    kdmb = h/sqrt(((D(n+3+2*(b-1))-D(n+3+2*(b-2)))^2)+(((1/2)*(uold(a+1,b-1)
                             -(-2*h*N(0,b-1,1)+uold(1,b-1))))^2)+delta*(h^2));
    sumabm1 = kdmb*D(n+3+2*(b-2));
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(n+3+2*(b-1));
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n+1 && 2<=b && b<=n-1
```

```
    kdpa = h/sqrt((((2*h*N(n+1,b,1)+uold(n,b))-D(n+4+2*(b-1)))^2)
             +(((1/2)*(D(n+4+2*b)+(2*h*N(n+1,b+1,1)+uold(n,b+1))-D(n+4+2*(b-2))
                              -(2*h*N(n+1,b-1,1)+uold(n,b-1))))^2)+delta*(h^2));
    sumap1b = kdpa*(2*h*N(n+1,b,1)+uold(n,b));
    kdma = h/sqrt(((D(n+4+2*(b-1))-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)
                     +D(n+4+2*b)-uold(a-1,b-1)-D(n+4+2*(b-2))))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((D(n+4+2*b)-D(n+4+2*(b-1)))^2)+(((1/2)*((2*h*N(n+1,b+1,1)
                                +uold(n,b+1))-uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(n+4+2*b);
    kdmb = h/sqrt(((D(n+4+2*(b-1))-D(n+4+2*(b-2)))^2)
      +(((1/2)*((2*h*N(n+1,b-1,1)+uold(n,b-1))-uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(n+4+2*(b-2));
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(n+4+2*(b-1));
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif b==0 && 2<=a && a<=n-1
    kdpa = h/sqrt(((D(a+2)-D(a+1))^2)+(((1/2)*(uold(a,b+1)+uold(a+1,b+1)
                 -(-2*h*N(a,0,2)+uold(a,1))-(-2*h*N(a+1,0,2)+uold(a+1,1))))^2)
                                                              +delta*(h^2));
    sumap1b = kdpa*D(a+2);
    kdma = h/sqrt(((D(a+1)-D(a))^2)+(((1/2)*(uold(a-1,b+1)+uold(a,b+1)
                 -(-2*h*N(a-1,0,2)+uold(a-1,1))-(-2*h*N(a,0,2)+uold(a,1))))^2)
                                                              +delta*(h^2));
    sumam1b = kdma*D(a);
    kdpb = h/sqrt(((uold(a,b+1)-D(a+1))^2)+(((1/2)*(uold(a+1,b+1)
                                         -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((D(a+1)-(-2*h*N(a,0,2)+uold(a,1)))^2)
     +(((1/2)*((-2*h*N(a+1,0,2)+uold(a+1,1))-(-2*h*N(a-1,0,2)+uold(a-1,1))))^2)
                                                              +delta*(h^2));
    sumabm1 = kdmb*(-2*h*N(a,0,2)+uold(a,1));
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(a+1);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif b==n+1 && 2<=a && a<=n-1
    kdpa = h/sqrt(((D(3*n+4+a)-D(3*n+3+a))^2)+(((1/2)*((2*h*N(a,n+1,2)
      +uold(a,n))+(2*h*N(a+1,n+1,2)+uold(a+1,n))-uold(a,b-1)-uold(a+1,b-1)))^2)
                                                              +delta*(h^2));
    sumap1b = kdpa*D(3*n+4+a);
    kdma = h/sqrt(((D(3*n+3+a)-D(3*n+2+a))^2)+(((1/2)*((2*h*N(a-1,n+1,2)
        +uold(a-1,n))+(2*h*N(a,n+1,2)+uold(a,n))-uold(a-1,b-1)-uold(a,b-1)))^2)
                                                              +delta*(h^2));
    sumam1b = kdma*D(3*n+2+a);
    kdpb = h/sqrt((((2*h*N(a,n+1,2)+uold(a,n))-D(3*n+3+a))^2)
                     +(((1/2)*((2*h*N(a+1,n+1,2)+uold(a+1,n))-(2*h*N(a-1,n+1,2)
                                          +uold(a-1,n))))^2)+delta*(h^2));
    sumabp1 = kdpb*(2*h*N(a,n+1,2)+uold(a,n));
    kdmb = h/sqrt(((D(3*n+3+a)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                         -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(3*n+3+a);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==0 && b==1
    kdpa = h/sqrt(((uold(a+1,b)-D(n+3))^2)+(((1/2)*(D(n+5)+uold(a+1,b+1)-D(1)
                                                          -D(2)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((D(n+3)-(-2*h*N(0,1,1)+uold(1,1)))^2)
      +(((1/2)*((-2*h*N(0,2,1)+uold(1,2))+D(n+5)-D(1)-(-2*h*N(0,0,1)+D(2))))^2)
                                                                  +delta*(h^2));
    sumam1b = kdma*(-2*h*N(0,1,1)+uold(1,1));
    kdpb = h/sqrt(((D(n+5)-D(n+3))^2)+(((1/2)*(uold(a+1,b+1)
                                -(-2*h*N(0,2,1)+uold(1,2))))^2)+delta*(h^2));
    sumabp1 = kdpb*D(n+5);
    kdmb = h/sqrt(((D(n+3)-D(1))^2)+(((1/2)*(D(2)-(-2*h*N(0,0,1)+D(2))))^2)
                                                                  +delta*(h^2));
    sumabm1 = kdmb*D(1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(n+3);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==0 && b==n
    kdpa = h/sqrt(((uold(a+1,b)-D(3*n+1))^2)+(((1/2)*(D(3*n+3)+D(3*n+4)
                                   -D(3*n-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*uold(a+1,b);
    kdma = h/sqrt(((D(3*n+1)-(-2*h*N(0,n,1)+uold(1,n)))^2)
        +(((1/2)*((-2*h*N(0,n+1,1)+D(3*n+4))+D(3*n+3)-D(3*n-1)-(-2*h*N(0,n-1,1)
                                             +uold(1,n-1))))^2)+delta*(h^2));
    sumam1b = kdma*(-2*h*N(0,n,1)+uold(1,n));
    kdpb = h/sqrt(((D(3*n+3)-D(3*n+1))^2)+(((1/2)*(D(3*n+4)
                                -(-2*h*N(0,n+1,1)+D(3*n+4))))^2)+delta*(h^2));
    sumabp1 = kdpb*D(3*n+3);
    kdmb = h/sqrt(((D(3*n+1)-D(3*n-1))^2)+(((1/2)*(uold(a+1,b-1)
                             -(-2*h*N(0,n-1,1)+uold(1,n-1))))^2)+delta*(h^2));
    sumabm1 = kdmb*D(3*n-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(3*n+1);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n+1 && b==1
    kdpa = h/sqrt((((2*h*N(n+1,1,1)+uold(n,1))-D(n+4))^2)+(((1/2)*(D(n+6)
            +(2*h*N(n+1,2,1)+uold(n,2))-D(n+2)-(2*h*N(n+1,0,1)+D(n+1))))^2)
                                                                  +delta*(h^2));
    sumap1b = kdpa*(2*h*N(n+1,1,1)+uold(n,1));
    kdma = h/sqrt(((D(n+4)-uold(a-1,b))^2)+(((1/2)*(uold(a-1,b+1)+D(n+6)
                                           -D(n+2)-D(n+1)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((D(n+6)-D(n+4))^2)+(((1/2)*((2*h*N(n+1,2,1)+uold(n,2))
                                           -uold(a-1,b+1)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(n+6);
    kdmb = h/sqrt(((D(n+4)-D(n+2))^2)+(((1/2)*((2*h*N(n+1,0,1)+D(n+1))
                                                  -D(n+1)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(n+2);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(n+4);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n+1 && b==n
    kdpa = h/sqrt((((2*h*N(n+1,n,1)+uold(n,n))-D(3*n+2))^2)
        +(((1/2)*(D(4*n+4)+(2*h*N(n+1,n+1,1)+D(4*n+3))-D(3*n)-(2*h*N(n+1,n-1,1)
                                             +uold(n,n-1))))^2)+delta*(h^2));
    sumap1b = kdpa*(2*h*N(n+1,n,1)+uold(n,n));
    kdma = h/sqrt(((D(3*n+2)-uold(a-1,b))^2)+(((1/2)*(D(4*n+3)+D(4*n+4)
                                 -uold(a-1,b-1)-D(3*n)))^2)+delta*(h^2));
    sumam1b = kdma*uold(a-1,b);
    kdpb = h/sqrt(((D(4*n+4)-D(3*n+2))^2)+(((1/2)*((2*h*N(n+1,n+1,1)+D(4*n+3))
                                                -D(4*n+3)))^2)+delta*(h^2));
    sumabp1 = kdpb*D(4*n+4);
    kdmb = h/sqrt(((D(3*n+2)-D(3*n))^2)+(((1/2)*((2*h*N(n+1,n-1,1)+uold(n,n-1))
                                            -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*D(3*n);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(3*n+2);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==1 && b==0
    kdpa = h/sqrt(((D(3)-D(2))^2)+(((1/2)*(uold(a,b+1)+uold(a+1,b+1)
        -(-2*h*N(1,0,2)+uold(1,1))-(-2*h*N(2,0,2)+uold(2,1))))^2)+delta*(h^2));
    sumap1b = kdpa*D(3);
    kdma = h/sqrt(((D(2)-D(1))^2)+(((1/2)*(D(n+3)+uold(a,b+1)-(-2*h*N(0,0,2)
                      +D(n+3))-(-2*h*N(1,0,2)+uold(1,1))))^2)+delta*(h^2));
    sumam1b = kdma*D(1);
    kdpb = h/sqrt(((uold(a,b+1)-D(2))^2)+(((1/2)*(uold(a+1,b+1)-D(n+3)))^2)
                                                         +delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((D(2)-(-2*h*N(1,0,2)+uold(1,1)))^2)+(((1/2)*((-2*h*N(2,0,2)
                      +uold(2,1))-(-2*h*N(0,0,2)+D(n+3))))^2)+delta*(h^2));
    sumabm1 = kdmb*(-2*h*N(1,0,2)+uold(1,1));
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(2);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n && b==0
    kdpa = h/sqrt(((D(n+2)-D(n+1))^2)+(((1/2)*(uold(a,b+1)+D(n+4)
         -(-2*h*N(n,0,2)+uold(n,1))-(-2*h*N(n+1,0,2)+D(n+4))))^2)+delta*(h^2));
    sumap1b = kdpa*D(n+2);
    kdma = h/sqrt(((D(n+1)-D(n))^2)+(((1/2)*(uold(a-1,b+1)+uold(a,b+1)
    -(-2*h*N(n,0,2)+uold(n,1))-(-2*h*N(n-1,0,2)+uold(n-1,1))))^2)+delta*(h^2));
    sumam1b = kdma*D(n);
    kdpb = h/sqrt(((uold(a,b+1)-D(n+1))^2)+(((1/2)*(D(n+4)-uold(a-1,b+1)))^2)
                                                         +delta*(h^2));
    sumabp1 = kdpb*uold(a,b+1);
    kdmb = h/sqrt(((D(n+1)-(-2*h*N(n,0,2)+uold(n,1)))^2)+(((1/2)*(D(n+2)
                          -(-2*h*N(n-1,0,2)+uold(n-1,1))))^2)+delta*(h^2));
    sumabm1 = kdmb*(-2*h*N(n,0,2)+uold(n,1));
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(n+1);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);
```

```
elseif a==1 && b==n+1
    kdpa = h/sqrt(((D(3*n+5)-D(3*n+4))^2)+(((1/2)*((2*h*N(1,n+1,2)+uold(1,n))
       +(2*h*N(2,n+1,2)+uold(2,n))-uold(a,b-1)-uold(a+1,b-1)))^2)+delta*(h^2));
    sumap1b = kdpa*D(3*n+5);
    kdma = h/sqrt(((D(3*n+4)-D(3*n+3))^2)+(((1/2)*((2*h*N(0,n+1,2)+D(3*n+1))
            +(2*h*N(1,n+1,2)+uold(1,n))-D(3*n+1)-uold(a,b-1)))^2)+delta*(h^2));
    sumam1b = kdma*D(3*n+3);
    kdpb = h/sqrt((((2*h*N(1,n+1,2)+uold(1,n))-D(3*n+4))^2)
            +(((1/2)*((2*h*N(2,n+1,2)+uold(2,n))-(2*h*N(0,n+1,2)+D(3*n+1))))^2)
                                                            +delta*(h^2));
    sumabp1 = kdpb*(2*h*N(1,n+1,2)+uold(1,n));
    kdmb = h/sqrt(((D(3*n+4)-uold(a,b-1))^2)+(((1/2)*(uold(a+1,b-1)
                                              -D(3*n+1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(3*n+4);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);

elseif a==n && b==n+1
    kdpa = h/sqrt(((D(4*n+4)-D(4*n+3))^2)+(((1/2)*((2*h*N(n,n+1,2)+uold(n,n))
           +(2*h*N(n+1,n+1,2)+D(3*n+2))-uold(a,b-1)-D(3*n+2)))^2)+delta*(h^2));
    sumap1b = D(4*n+4);
    kdma = h/sqrt(((D(4*n+3)-D(4*n+2))^2)+(((1/2)*((2*h*N(n-1,n+1,2)
        +uold(n-1,n))+(2*h*N(n,n+1,2)+uold(n,n))-uold(a-1,b-1)-uold(a,b-1)))^2)
                                                            +delta*(h^2));
    sumam1b = kdma*D(4*n+2);
    kdpb = h/sqrt((((2*h*N(n,n+1,2)+uold(n,n))-D(4*n+3))^2)
      +(((1/2)*((2*h*N(n+1,n+1,2)+D(3*n+2))-(2*h*N(n-1,n+1,2)+uold(n-1,n))))^2)
                                                            +delta*(h^2));
    sumabp1 = kdpb*(2*h*N(n,n+1,2)+uold(n,n));
    kdmb = h/sqrt(((D(4*n+3)-uold(a,b-1))^2)+(((1/2)*(D(3*n+2)
                                           -uold(a-1,b-1)))^2)+delta*(h^2));
    sumabm1 = kdmb*uold(a,b-1);
    sumab = -(kdpa+kdma+kdpb+kdmb)*D(4*n+3);
    res = (1/(h^2))*(sumap1b+sumam1b+sumabp1+sumabm1+sumab);
end

end
```

sol = N(x,y,der)

```
function sol = N(x,y,der)

% Name: CARMEN MAYORA CEBOLLERO
% Date: 2020
% This program computes the approximation of the derivatives on the
                                 % boundary (Neumann boundary conditions).

% It has three inputs:
            % -> x is the index of the x-coordinate.
            % -> y is the index of the y-coordinate.
            % -> der=1 is x-derivative and der=2 is y-derivative.
% This auxiliary program is used in the program auxiliary and in
                                                                % bilaplacianb.

global Neumann;
global D;
global h;
global n;

if der==1
    if x==0
        if y==0
            sol = (1/h)*(D(1)-Neumann(n+3));
        elseif y==n+1
            sol = (1/h)*(D(3*n+3)-Neumann(3*n+5));
        else
            sol = (1/h)*(D(n+3+2*(y-1))-Neumann(n+3+2*y));
        end
    elseif x==n+1
        if y==0
            sol = (1/h)*(Neumann(n+4)-D(n+2));
        elseif y==n+1
            sol = (1/h)*(Neumann(3*(n+2))-D(4*n+4));
        else
            sol = (1/h)*(Neumann(n+4+2*y)-D(n+2+2*y));
        end
    end
elseif der==2
    if y==0
        sol = (1/h)*(D(x+1)-Neumann(x+1));
    elseif y==n+1
        sol = (1/h)*(Neumann(3*(n+2)+1+x)-D(3*n+3+x));
    end
end

end
```

[A,b] = laplacian()

```
function [A,b] = laplacian()

% Name: CARMEN MAYORA CEBOLLERO
% Date: 2020
% This program computes the matrix and the vector of independent terms of
            % the system of equations obtained with the discretization of
                                                        % laplacian(u) = 0.
% It is an adaptation of the program harmonicInpainting.

% This auxiliary program is used in the program CHBinary.

global n;
global h;
global D;

%  A is a matrix with a lot of zeros, so it is sparse:
A = sparse(n*n, n*n);

%  aux represents a number which is used a lot in the program:
aux = 1/(h^2);

%  We fill the matrix A:

%%Diagonal of the matrix.
for i = 1:n*n
    A(i,i) = -4*aux;
end

%%Elements above and below the diagonal.
for ind = 0:n-1
    for i = 1:n-1
        A(i+n*ind,(i+1)+n*ind) = aux;
        A((i+1)+n*ind,i+n*ind) = aux;
    end
end

%%Elements of the diagonal sub-matrices.
for ind = 0:n-2
    for i = 1:n
        A(i+n*ind,(i+n*ind)+n) = aux;
        A((i+n*ind)+n,i+n*ind) = aux;
    end
end

%  We fill the vector b (independent terms):
b = zeros(1,n*n);

%%1. Independent terms which correspond with the n first elements of
                                                            %%the vector b.
b(1) = -aux*D(n+3)-aux*D(2);
b(n) = -aux*D(n+4)-aux*D(n+1);
for i = 2:n-1
    b(i) = -aux*D(i+1);
end

%%2,...,n-1. Independent terms which correspond with the middle elements
                                                          %%of the vector b.
for j = 2:n-1
    b((j-1)*n+1) = -aux*D(n+3+2*(j-1));
    b(j*n) = -aux*D(n+2+2*j);
end

%%n. Independent terms which correspond with the n last elements of
                                                            %%the vector b.
```

```
b(n*(n-1)+1) = -aux*D(3*n+1)-aux*D(3*n+4);
b(n*n) = -aux*D(3*n+2)-aux*D(4*n+3);
for i = 2:n-1
    b((n-1)*n+i) = -aux*D(3*n+3+i);
end

end
```

A = bilaplacianA()

```
function A = bilaplacianA()

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program computes the matrix of the system of equations obtained
                          %  with the discretization of bilaplacian(u) = 0.

%  This auxiliary program is used in the program CHBinary and in
                                                                 %  CHGreyValues.

global n;
global h;

%  A is a matrix with a lot of zeros, so it is sparse:
A = sparse(n*n, n*n);

%  aux represents a number which is used a lot in the program:
aux = 1/(h^4);

%  We fill the matrix A:

%%Diagonal of the matrix:

%%%The first and the last n elements of the diagonal.
A(1,1) = 22*aux;
A(n,n) = 22*aux;
A(n*(n-1)+1,n*(n-1)+1) = 22*aux;
A(n*n,n*n) = 22*aux;
for i = 2:n-1
    A(i,i) = 21*aux;
    A(n*(n-1)+i,n*(n-1)+i) = 21*aux;
end

%%%Remaining elements.
for ind = 1:n-2
    A(n*ind+1,n*ind+1) = 21*aux;
    A((ind+1)*n,(ind+1)*n) = 21*aux;
    for i = 2:n-1
        A(n*ind+i,n*ind+i) = 20*aux;
    end
end
```

```
%%Elements that are above and below the diagonal.
for ind = 0:n-1
    for i = 1:n-1
        A(i+n*ind,(i+1)+n*ind) = -8*aux;
        A((i+1)+n*ind,i+n*ind) = -8*aux;
    end
end

%%Remaining elements of the diagonal blocks of dimension nxn.
for ind = 0:n-1
    for i = 1:n-2
        A(i+n*ind,(i+2)+n*ind) = aux;
        A((i+2)+n*ind,i+n*ind) = aux;
    end
end

%%Blocks above and below the diagonal blocks.
for ind = 0:n-2
    for i = 1:n
        A(i+n*ind,(i+n*ind)+n) = -8*aux;
        A((i+n*ind)+n,i+n*ind) = -8*aux;
    end
end
for ind = 0:n-2
    for i = 1:n-1
        A(i+n*ind,(i+n*ind)+n+1) = 2*aux;
        A((i+1)+n*ind,(i+n*ind)+n) = 2*aux;
        A((i+n*ind)+n+1,i+n*ind) = 2*aux;
        A((i+n*ind)+n,(i+1)+n*ind) = 2*aux;
    end
end

%%Identity matrices.
for ind = 0:n-3
    for i = 1:n
        A(i+n*ind,(i+n*ind)+2*n) = aux;
        A((i+n*ind)+2*n,i+n*ind) = aux;
    end
end

end
```

b = bilaplacianb()

```
function b = bilaplacianb()

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program computes the vector of independent terms of the system of
       %  equations obtained with the discretization of bilaplacian(u) = 0.

%  This program uses one auxiliary function:
        %  -> N.

%  This auxiliary program is used in the program CHBinary and in
                                                        %  CHGreyValues.

global n;
global h;
global D;

%  aux represents a number which is used a lot in the program:
aux = -1/(h^4);

%  We fill the vector b:
b = zeros(1,n*n);

%%1. Independent terms which correspond with the n first elements of
                                                         %%the vector b.
b(1) = 2*aux*D(3)-8*aux*D(2)+2*aux*D(1)-8*aux*D(n+3)+2*aux*D(n+5)
                                            -2*aux*h*(N(1,0,2)+N(0,1,1));
b(2) = 2*aux*D(4)-8*aux*D(3)+2*aux*D(2)+aux*D(n+3)-2*aux*h*N(2,0,2);
for i = 3:n-2
    b(i) = 2*aux*D(i+2)-8*aux*D(i+1)+2*aux*D(i)-2*aux*h*N(i,0,2);
end
b(n-1) = aux*D(n+4)+2*aux*D(n+1)-8*aux*D(n)+2*aux*D(n-1)
                                                     -2*h*aux*N(n-1,0,2);
b(n) = 2*aux*D(n+2)-8*aux*D(n+4)+2*aux*D(n+6)-8*aux*D(n+1)+2*aux*D(n)
                                            -2*h*aux*(N(n,0,2)-N(n+1,1,1));

%%2. Independent terms which correspond with the n elements of
                                            %%the vector b from n+1 to 2n.
b(n+1) = aux*D(2)+2*aux*D(n+3)-8*aux*D(n+5)+2*aux*D(n+7)-2*h*aux*N(0,2,1);
b(n+2) = aux*D(3)+aux*D(n+5);
for i = 3:n-2
    b(n+i) = aux*D(i+1);
end
b(2*n-1) = aux*D(n+6)+aux*D(n);
b(2*n) = 2*aux*D(n+4)-8*aux*D(n+6)+2*aux*D(n+8)+aux*D(n+1)
                                                     +2*h*aux*N(n+1,2,1);

%%3,...,n-2. Independent terms which correspond with the middle elements
                                                       %%of the vector b.
for j = 3:n-2
    b((j-1)*n+1) = 2*aux*D(n+3+2*(j-2))-8*aux*D(n+3+2*(j-1))
                                      +2*aux*D(n+3+2*j)-2*h*aux*N(0,j,1);
    b((j-1)*n+2) = aux*D(n+3+2*(j-1));
    b(j*n-1) = aux*D(n+4+2*(j-1));
    b(j*n) = 2*aux*D(n+4+2*(j-2))-8*aux*D(n+4+2*(j-1))+2*aux*D(n+4+2*j)
                                                     +2*h*aux*N(n+1,j,1);
end

%%n-1. Independent terms which correspond with the n elements of
```

```
                                        %%the vector b from n(n-2)+1 to n(n-1).
b(n*(n-2)+1) = aux*D(3*n+4)+2*aux*D(3*n+1)-8*aux*D(3*n-1)+2*aux*D(3*n-3)
                                                        -2*h*aux*N(0,n-1,1);
b(n*(n-2)+2) = aux*D(3*n+5)+aux*D(3*n-1);
for i = 3:n-2
    b(n*(n-2)+i) = aux*D(3*n+3+i);
end
b(n*(n-1)-1) = aux*D(3*n)+aux*D(4*n+2);
b(n*(n-1)) = 2*aux*D(3*n-2)-8*aux*D(3*n)+2*aux*D(3*n+2)+aux*D(4*n+3)
                                                      +2*h*aux*N(n+1,n-1,1);

%%n. Independent terms which correspond with the n last elements of
                                                             %%the vector b.
b(n*(n-1)+1) = 2*aux*D(3*n+5)-8*aux*D(3*n+4)+2*aux*D(3*n-1)-8*aux*D(3*n+1)
                             +2*aux*D(3*n+3)-2*h*aux*(N(0,n,1)-N(1,n+1,2));
b(n*(n-1)+2) = 2*aux*D(3*n+6)-8*aux*D(3*n+5)+2*aux*D(3*n+4)+aux*D(3*n+1)
                                                        +2*h*aux*N(2,n+1,2);
for i = 3:n-2
    b(n*(n-1)+i) = 2*aux*D(3*n+4+i)-8*aux*D(3*n+3+i)+2*aux*D(3*n+2+i)
                                                        +2*h*aux*N(i,n+1,2);
end
b(n*n-1) = aux*D(3*n+2)+2*aux*D(4*n+3)-8*aux*D(4*n+2)+2*aux*D(4*n+1)
                                                      +2*h*aux*N(n-1,n+1,2);
b(n*n) = 2*aux*D(3*n)-8*aux*D(3*n+2)+2*aux*D(4*n+4)-8*aux*D(4*n+3)
                           +2*aux*D(4*n+2)+2*h*aux*(N(n,n+1,2)+N(n+1,n,1));

end
```

sol = laplacianF()

```
function sol = laplacianF()

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program computes a vector which stores the discretization of
                                                   %  laplacian(F'(uold)).

%  This program uses one auxiliary function:
       %  -> Fder.

% This auxiliary program is used in the program CHBinary.

global n;
global h;
global D;
global uold;

uold = reshape(uold,n,n);

%  aux represents a number which is used a lot in the program:
aux = 1/(h^2);
```

```
%  We fill the vector sol.
sol = zeros(n*n,1);

for j = 2:n-1
    for i = 2:n-1
        sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(uold(i-1,j))
                   +Fder(uold(i,j+1))+Fder(uold(i,j-1))-4*Fder(uold(i,j)));
    end
end

i = 1;
for j = 2:n-1
    sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(D(n+3+2*(j-1)))
                   +Fder(uold(i,j+1))+Fder(uold(i,j-1))-4*Fder(uold(i,j)));
end

i = n;
for j = 2:n-1
    sol((j-1)*n+i) = aux*(Fder(D(n+4+2*(j-1)))+Fder(uold(i-1,j))
                   +Fder(uold(i,j+1))+Fder(uold(i,j-1))-4*Fder(uold(i,j)));
end

j = 1;
for i = 2:n-1
    sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(uold(i-1,j))
                        +Fder(uold(i,j+1))+Fder(D(i+1))-4*Fder(uold(i,j)));
end

j = n;
for i = 2:n-1
    sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(uold(i-1,j))
                    +Fder(D(3*n+3+i))+Fder(uold(i,j-1))-4*Fder(uold(i,j)));
end

i = 1; j = 1;
sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(D(n+3))+Fder(uold(i,j+1))
                                           +Fder(D(2))-4*Fder(uold(i,j)));

i = n; j = 1;
sol((j-1)*n+i) = aux*(Fder(D(n+4))+Fder(uold(i-1,j))+Fder(uold(i,j+1))
                                         +Fder(D(n+1))-4*Fder(uold(i,j)));

i = 1; j = n;
sol((j-1)*n+i) = aux*(Fder(uold(i+1,j))+Fder(D(3*n+1))+Fder(D(3*n+4))
                                    +Fder(uold(i,j-1))-4*Fder(uold(i,j)));

i = n; j = n;
sol((j-1)*n+i) = aux*(Fder(D(3*n+2))+Fder(uold(i-1,j))+Fder(D(4*n+3))
                                    +Fder(uold(i,j-1))-4*Fder(uold(i,j)));

uold = reshape(uold,[],1);
```

```
end
```

res = Fder(a)

```
function res = Fder(a)

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program evaluates the function F'(x)=4x^3-6x^2+2x with the input
                                                                 %  value a.

%  This auxiliary program is used in the program laplacianF.

res = 4*(a^3)-6*(a^2)+2*a;

end
```

sol = laplacianaux()

```
function sol = laplacianaux()

%  Name: CARMEN MAYORA CEBOLLERO
%  Date: 2020
%  This program computes a vector which stores the discretization of
                  %  laplacian(div((grad(uold))/(normdelta(grad(uold))))).

%  This program uses one auxiliary function:
       %  -> auxiliary.

%  This auxiliary program is used in the program CHGreyValues.

global n;
global h;

%  aux represents a number which is used a lot in the program:
aux = 1/(h^2);

%  We fill the vector sol
sol = zeros(n*n,1);

for j = 1:n
    for i = 1:n
        sol((j-1)*n+i) = aux*(auxiliary(i+1,j)+auxiliary(i-1,j)
                      +auxiliary(i,j+1)+auxiliary(i,j-1)-4*auxiliary(i,j));
    end
end

end
```

Bibliography

[1] Inpaint, `https://online.theinpaint.com/myimages`.

[2] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles and Coloma Ballester, *Image Inpainting* (proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques), ACM Press/Addison-Wesley, Reading, MA, 2000, 417–424.

[3] Carola-Bibiane Schönlieb, *Partial Differential Equation Methods for Image Inpainting*, Cambridge University Press, **29** (2015).

[4] Gaetano Kanizsa, *Organization in Vision: Essays on Gestalt Perception*, Praeger, New York, 1979.

[5] Jianhong Shen and Tony F. Chan, *Mathematical Models for Local Non-texture Inpaintings*, SIAM J. Appl. Math., **62** (2002), 1019–1043.

[6] Leonid I. Rudin, Stanley Osher and Emad Fatemi, *Nonlinear Total Variation Based Noise Removal Algorithms*, Physica D: Nonlinear Phenomena, **60** (1992), 259–268.

[7] Vicent Caselles, J.-M. Morel and Catalina Sbert, *An Axiomatic Approach to Image Interpolation*, IEEE transactions on image processing, **7** (1998), 376–386.

[8] Junseok Kim, Seunggyu Lee, Yongho Choi, Seok-Min Lee and Darae Jeong, *Basic Principles and Practical Applications of the Cahn-Hilliard Equation*, Mathematical Problems in Engineering, 2016.

[9] Carola-Bibiane Schönlieb and Andrea Bertozzi, *Unconditionally Stable Schemes for Higher Order Inpainting*, Communications in Mathematical Sciences, **9** (2011), 413–457.

[10] Martin Burger, Line He and Carola-Bibiane Schönlieb, *Cahn-Hilliard Inpainting and a Generalization for Grayvalue Images*, SIAM J. Imaging Sciences, **2** (2009), 1129-1167.

[11] Christopher Stover, *Green's Function*, MathWorld – A Wolfram Web Resource (created by Eric W. Weisstein), `https://mathworld.wolfram.com/GreensFunction.html`.

[12] Lawrence C. Evans, *Partial Differential Equations: Graduate Studies in Mathematics*, American Mathematical Society, **19** (1998).

[13] Randall J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*, SIAM, 2007.