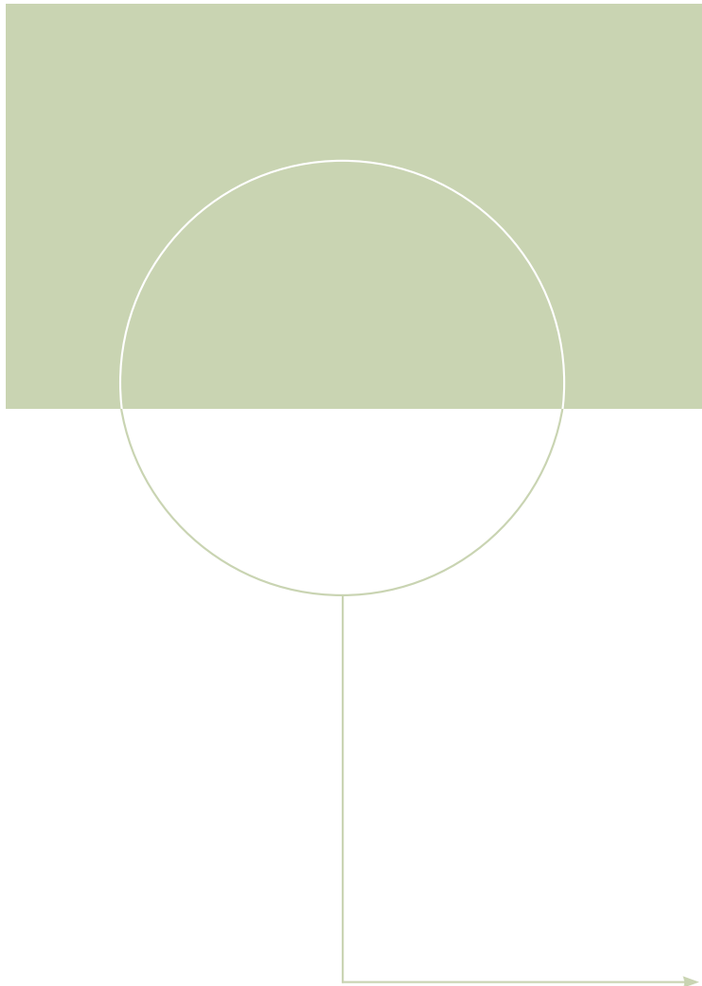


Master's thesis

2019

Paul Gabriel Duffin

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Mathematical Sciences

Paul Gabriel Duffin

Image Restoration and Super-Resolution using Total Variation

June 2019

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU

Norwegian University of
Science and Technology

Image Restoration and Super-Resolution using Total Variation

Mathematical Sciences
Submission date: June 2019
Supervisor: Markus Grasmair

Norwegian University of Science and Technology
Department of Mathematical Sciences

Abstract

One of the main goals of image restoration is the ability to extract information from corrupted sources of image data. The main goal of this thesis is to investigate one powerful method of image restoration, and that is using variational algorithms to optimize a functional. High-quality images are needed in a wide variety of practical applications so that appropriate information can be extracted from them, including in medical imaging, satellite imaging, astronomy, sonar, radar, among other applications. These images tend to have various sources of noise and other imperfections, and we investigate different ways to model and correct this.

In this thesis, we will make use of a modified Chambolle-Pock algorithm, which was originally developed for image denoising, but will now being used for more general applications, including super-resolution. This algorithm uses total variation and regularization to provide a stable solution to an otherwise highly unstable problem. In this thesis, we will investigate these types of problems in depth. We will further investigate this by implementing the primal-dual algorithm on images both for deblurring as well as for image super-resolution.

Sammendrag

Hovedformålet med bildegjenoppretting er å kunne hente informasjon fra ødelagte kilder til bildedata. Hovedmålet med denne oppgaven er å undersøke en kraftig metode for bildegjenoppretting, og det er å bruke variasjonsalgoritmer for å optimalisere en funksjonell. Bilder av høy kvalitet er nødvendig i et bredt spekter av praktiske applikasjoner, slik at relevant informasjon kan hentes fra dem, blant annet i medisinsk bildebehandling, satellittbildebehandling, astronomi, sonar og radar. Disse bildene har en tendens til å ha forskjellige støykilder og andre mangler, og vi undersøker forskjellige måter å modellere og å rette på dette.

I denne masteroppgaven vil vi bruke en modifisert Chambolle-Pock-algoritme, som opprinnelig ble utviklet for å fjerne støy fra bilder, men vil nå brukes til mer generelle applikasjoner, som for eksempel superoppløsning. Denne algoritmen bruker total variasjon og regularisering for å gi en stabil løsning på et problem som er ellers ustabilt. Vi vil undersøke og gi eksempler på disse typer problemer. Vi vil også undersøke dette ved å implementere den primale-duale algoritmen på bilder både for å skarpe bilder og for superoppløsning av bildene.

Preface

This thesis fulfills the requirement for the International Master's Degree in Mathematical Sciences, with a specialization in Applied Mathematics. It was completed in collaboration with my supervisor Markus Grasmair, Associate Professor in the Department of Mathematical Sciences at the Norwegian University of Science and Technology in the city of Trondheim, Norway.

I would like to express gratitude to the many people that have helped me on this journey. Thanks to Markus Grasmair for providing helpful insights and feedback into this project, as well as for proposing this interesting problem. Many thanks are also due to my family back home for their great support as I undertook this journey. I would also like to extend a great thanks to all of the Norwegian people for welcoming me into their community, and enhancing my time here.

Special thanks to Orbit NTNU, a student-run organization at this university that will be launching a selfie-taking CubeSat into low-earth orbit in 2020. I am forever grateful to them for including me in their many adventures and projects, and for providing me with a lot of great experience. Congratulations on the recent successful balloon mission, and thanks for allowing me to be a part of that team and use images obtained from that satellite balloon test in this master thesis.

Contents

List of Figures

Abbreviations

RGB	=	Red, Green, Blue
ROF	=	Rudin-Osher-Fatemi
TV	=	Total Variation

Chapter 1

Introduction

Whenever cameras take images, or when digital images are transferred or compressed, there is typically some blurring, noise, or other loss of information. There are various ways that one can attempt to restore this loss of information, including image sharpening, wavelet transforms, and optimization with regularization. The reason we want to do this is so that we can restore these lost features to a recognizable form. This has many practical applications in a wide variety of fields, including photography, medical imaging, satellite imagery, and computer vision.

We seek to process raw image data in order to restore some of their finer characteristics that may have been lost due to natural noise or other processes on the image. The problem then is to reverse what we call "camera artifacts" to produce the best possible original image. From a practical standpoint, this enables us to restore the image to make its original features more easily recognizable. From a linear algebra standpoint, this is not a well-posed problem, but as we will see in later chapters, we can get some results if we use regularization and optimization techniques. It is assumed that the reader is familiar with standard linear algebra notation and operators.

In Chapter 2, we will look at the mathematical formulations for these types of problems. Some examples of artifacts or other known operations that need to be reversed are static noise, blurring, or down-sampling. We start out by describing the mathematical properties of an image, from both a continuous and discrete perspective. We will further define the noise, blurring, down-sampling, and up-sampling operators in a mathematically robust way. In Chapter 3, we will discuss total variation concepts from a theoretical perspective and investigate certain properties of the gradient and divergence. In Chapter 4, we investigate saddle-point optimization problems and look at ways to solve these problems using convex optimization techniques. Here we must also lay a groundwork for convex analysis. For the purposes of this thesis, we will mostly consider grey-scale images. We will then describe some of the problems we wish to look at in this thesis in Chapter 5. Implemented Matlab functions will be appended to Appendix A. Test cases are given in Appendix B.

Chapter 2

Image Restoration

2.1 Mathematical Images

In the continuous bounded domain $\Omega \subset \mathbb{R}^n$, we will define a continuous image to be a function $u : \Omega \to \mathbb{R}^d$ from a spatial domain of dimension n to an output range of dimension d. This is not to say that the function is necessarily continuous, but that it is defined over a continuous domain of real numbers. We should not confuse this with the more common definition of an image, which is the mapping of a subset of a functional domain. In the case of common gray-scale images, we use $n = 2, d = 1$. That is to say the image function maps points in $\mathbb{R}^2$ to a single brightness value in $\mathbb{R}$. For RGB color images, we have $n = 2, d = 3$, and the output value is a vector which contains integer values for the three basic color channels (RGB). These types of images are commonly used in photography to detail a visual scene. Some examples of higher-dimensional images may include medical scans, computer graphics, subsurface imaging for oil exploration or seismic data, and so on. For simplicity, we restrict ourselves to two spacial dimensions and assume that Ω is a rectangular domain $[0, x] \times [0, y]$.

2.1.1 Discrete Images

To discretize an image, and thus make it something we can perform computations with, we sample the image in an evenly distributed manner. This type of discretization is generally not the same as that performed by a camera, which relies on taking the averages over a small area. This is therefore more of a theoretical construction. Later on, we will see that this is essentially analogous to down-sampling from an infinite resolution. We impose these sample points onto a rectangular sampling grid, at well defined intervals of length Δx and Δy along the respective axes, and call those values pixels. We specify a resolution $m \times n$, which tells us the number of pixels in each of the respective columns and rows, and choose Δx and Δy accordingly to be the continuous spacing between each discrete interval. We then impose the values of these pixels into a rectangular matrix $U \in \mathbb{R}^{m \times n}$.

We sample the function as follows, where we assume that the origins are identical:

$$U_{i,j} = u(i\Delta x, j\Delta y) \tag{2.1}$$

In the case of color images, we end up with a $m \times n \times 3$ array, where the red, blue, and green components each have their own separate $m \times n$ matrices. In this case, each channel of the RGB color scheme is also governed by this equation (2.1).

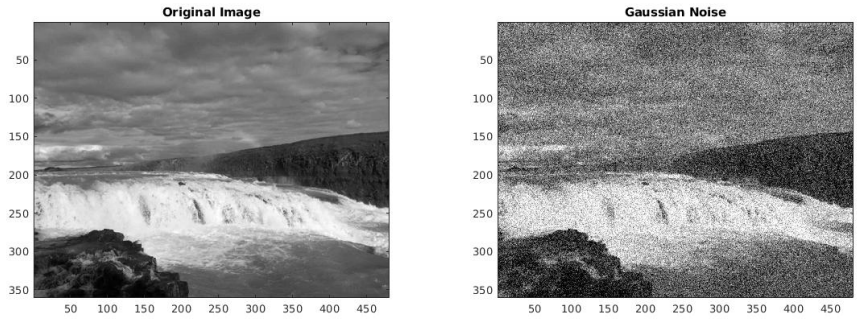
2.2 Image Noise

The problem of image denoising arises from the fact that many cameras and other imaging instruments introduce some small errors into the image, known as noise. This can be caused by high temperature, radiation, sensor vibration, or other physical considerations that affect the camera. This is usually modelled as what we call Gaussian noise, where the amount of noise disturbance in each pixel of the image is assumed to be modelled by the normal distributed. The following probability function is generally used to model this noise, where x is a random variable.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{2.2}$$

The standard deviation σ controls how much variance the noise will have, and the mean value μ represents the average noise. In the context of image processing, we will generally assume $\mu = 0$. The noise is modelled additively by the equation $V = U + N_G$, where U is an unknown noise-free image, and V is the known noisy data. The array N_G is randomly Gaussian distributed, where each element of is an independent Gaussian variable, and is of the same size as V. In later chapters we will investigate the use of mathematical optimization techniques and total variation to minimize this noise, and use this to estimate the noise-free image u. See Figure 2.1 for an example of an artificially generated noisy image with zero mean. The standard deviation in this figure is $\sigma = 40$ while the values range from -177 to 185.

Figure 2.1 Left: Original 360x480 Image Right: Gaussian noise added with $\sigma = 40$



2.3 Image Operations

The problem of deblurring is very similar to that of denoising. Blurring can be caused by many of the same things related to noise, including motion and vibration as well as lens focus and the atmosphere. In practice, we will typically see a combination of both blurring and noise. The main difference is that the values of any given pixel can be affected by the values of nearby pixels. We model this blurring operation of a discrete image mathematically by convoluting a small matrix with the larger image. This convolution operation is mathematically formulated as follows.

2.3.1 Convolution

Suppose we have a continuous image $u(x, y)$ on a rectangular domain Ω. Then define k to be another continuous function where the following is satisfied:

$$\iint_{\Omega} k(p, q)dpdq = 1 \tag{2.3}$$

The function k is also known as a kernel, and should also satisfy the following property.

$$\iint_{\Omega} |k(p, q)|dpdq \leq K \tag{2.4}$$

This makes the kernel much nicer to work with and will allow us to better define the convolution. The convolution of u with the kernel k is defined in the following way.

$$k * u(x, y) = \iint k(p, q)u(x - p, y - q)dpdq \tag{2.5}$$

If Ω is a bounded domain, then it can either be extended with zeros, or else each point can be reflected across the boundary so that $u(x - p, y - q)$ is well-defined. In the case where u is represented as a discrete image, or a matrix, then we can rewrite the convolution in the following matrix notation, where the edges of u are padded with zeros or other values as needed.

$$(k * u)_{ij} = \sum_{p} \sum_{q} (k_{pq} u_{i-p,j-q}) \tag{2.6}$$

The matrix k is a convolution matrix corresponding to its continuous counterpart, where the center of the matrix represents the origin of the continuous domain of the kernel function. This kernel will generally be represented as a relatively small matrix whose elements are normalized such that they sum to 1. The reason for this, as well as for the condition (2.3) is so that the image values don't get changed in scale. Additionally, we note that the convolution operator is self-adjoint, that is to say:

$$\langle k * x, y \rangle = \langle x, k * y \rangle \tag{2.7}$$

This follows directly from the definitions, and because this convolution with k is a symmetric operator.

2.3.2 Gaussian Blur

The simplest type of blurring kernel we can implement is the averaging kernel. In this case, all the values of the kernel are the same, and the kernel is normalized so that its total sum is 1. If we convolute this type of kernel with itself a few times, then the Central Limit Theorem suggests that we will approach what is called a Gaussian kernel, which is more widely applicable [1]. The following suggests an example of this:

$$B = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \Rightarrow \qquad B*B = \begin{bmatrix} 0.0123 & 0.0247 & 0.0370 & 0.0247 & 0.0123 \\ 0.0247 & 0.0494 & 0.0741 & 0.0494 & 0.0247 \\ 0.0370 & 0.0741 & 0.1111 & 0.0741 & 0.0370 \\ 0.0247 & 0.0494 & 0.0741 & 0.0494 & 0.0247 \\ 0.0123 & 0.0247 & 0.0370 & 0.0247 & 0.0123 \end{bmatrix}$$

See Figure 2.2 for an example of this type of kernel. In the figure, the blurring kernel is a symmetric 11x11 Gaussian matrix, where the distance from the center represents a normally distributed variable. This type of blurring kernel may be used for smoothing or pre-processing an image, but in our case we may also consider it as a description or source of natural blur, which we must then unblur. Assume that we are given v and k such that. The problem is then to solve the following equation for u

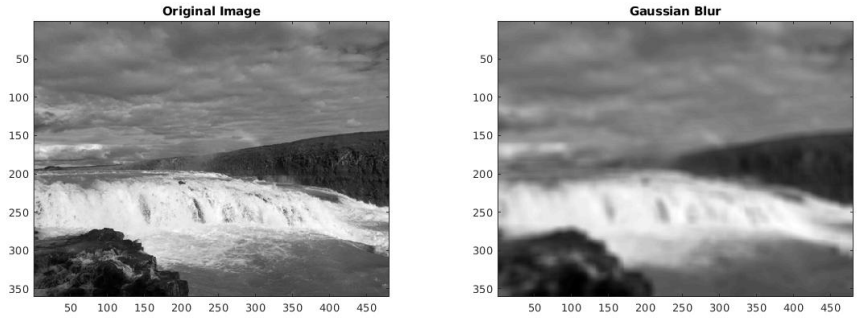
$$v = k * u \tag{2.8}$$

We may also assume that there is some additive noise, in which case the model becomes

$$v = k * u + n \tag{2.9}$$

In Chapter 5, we will explore some examples of these types of problems. In Chapter 4, we will describe a way to solve this problem by optimizing.

Figure 2.2 Left: Original 360x480 Image Right: Gaussian blur with $\sigma = 40$



2.4 Image Super-resolution

The problem of image super-resolution can be described as follows. Suppose we have several different images of the same scene, or a video, taken from slightly different positions. We then take several different frames or still images, and then plot them together on a higher resolution grid. If we look at Figure 2.4, we can imagine that one of the images on the right is taken from one position, while the other image is taken from a slightly different position. We must then optimize to choose a higher resolution image that best fits this solution, and later we will use total variation to accomplish this.

2.4.1 Down-sampling

In order to mathematically formulate this problem, let us begin by defining the operator A_J as the down-sampling operator on a point set J by a scaling factor of F. The set J consists of integer shifts (i, j) in the discrete image, where $1 \leq i, j \leq F$. This set must also have no duplicate values. For our purposes, we will assume M and N to be evenly divisible by F, though this may not always be the case. We denote the k-th shift vector in the set by $J(k)$. That is to say, the operator $A_J : \mathbb{R}^{M \times N} \to \mathbb{R}^{M' \times N' \times |J|}$ maps a full $M \times N$ image to a stack of $|J|$ lower-resolution images. We mathematically formulate this operation as follows:

$$(A_J u)_{i,j,k} = u_{p,q} \tag{2.10}$$

where p and q are indices in the larger image that we wish to down-sample:

$$p = [J(k)]_x + (i-1)F \tag{2.11}$$

$$q = [J(k)]_y + (j-1)F \tag{2.12}$$

where $[J(k)]_x$ and $[J(k)]_y$ are respectively the x and y components of the point $J(k)$. We can easily observe that the operator A_J is a linear map. This is due to the following observations. If we scale the right-hand side of (2.10) by a constant $c \in \mathbb{R}$, then the same scaling happens to the left-hand side, and we get the following (2.13). Furthermore, if we split the operand into a sum of two matrices, then we end up with the sum of two down-sampled image sets.

$$A_J(cu) = c(A_J u) \qquad A_J(u+v) = (A_J u) + (A_J v) \tag{2.13}$$

See Figure 2.4 for an illustration of this operator. Here we have an image on the left that gets down-sampled to two smaller images on the right, where the pixels are represented by letters. We will also see an example of a down-sampled image, with the resolution made intentionally low (36×48) so as to see the effect that this operator has. Here

Figure 2.3 Down-sampling operator A_J, where $J = \{(1, 1), (3, 3)\}$, $F = 4$

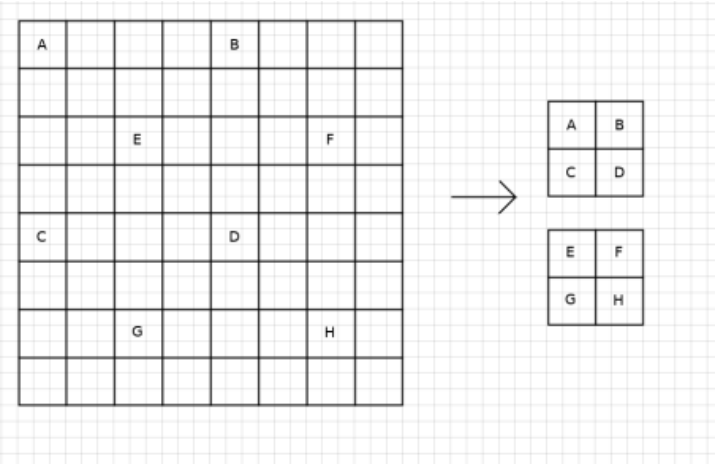
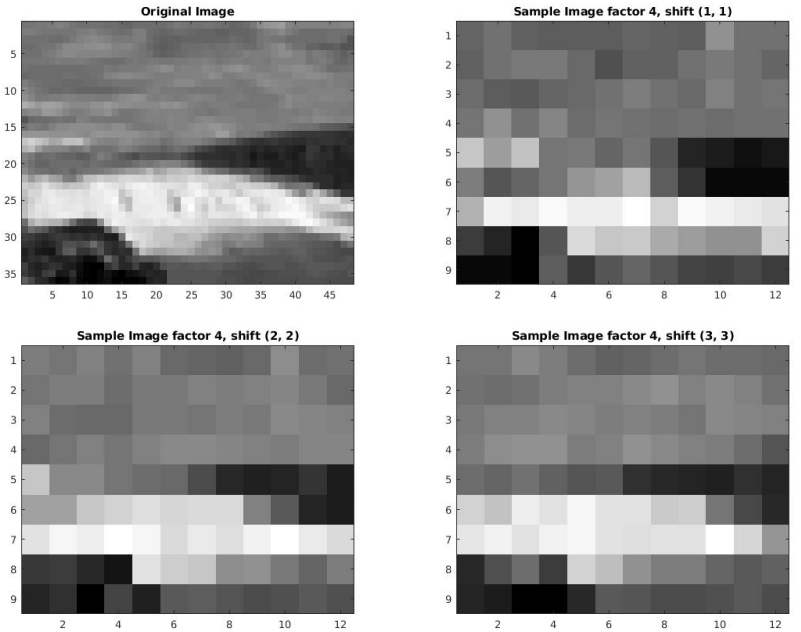


Figure 2.4 Original and down-sampled images



2.4.2 Kronecker Product

Before we introduce the up-sampling operator, it is useful to know about the Kronecker product of two matrices. What this operation does is take the product of every possible entry of A multiplied by every possible entry of B. The Kronecker product of two real matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ is defined as the following $mp \times nq$ matrix, $A \otimes B$:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1n}B \\ a_{21}B & a_{22}B & \dots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \dots & a_{mn}B \end{bmatrix} \tag{2.14}$$

where $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{p\times q}$ are given as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} b_{11} & b_{12} & \dots & b_{1q} \\ b_{21} & b_{22} & \dots & b_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pq} \end{bmatrix} \tag{2.15}$$

Now let us define the matrix $(\mathbf{1})_{i'j'}$ as the following $m \times n$ matrix for $1 \le i' \le m$ and $1 \le j' \le n$:

$$[(\mathbf{1})_{i'j'}]_{ij} = \begin{cases} 1 & \text{when } i = i' \ \&\ j = j', \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

This essentially tells us that only the (i', j') entry of this matrix is 1, while all other entries are zero. When we define the up-sampling operator, this matrix will be the second operand of the Kronecker product.

2.4.3 Up-sampling

The up-sampling operator $A'_J : \mathbb{R}^{M\times N\times |J|} \to \mathbb{R}^{MF\times NF}$ is defined as the operator where a stack of images such as the one on the right of Figure 2.4 is inserted into a larger grid at regularly spaced intervals, such as the one on the left, with zeros in all other spaces. This can be represented mathematically as follows. Suppose we have a set of images (or video) v, represented as a 3D-array. Let $(\mathbf{1})_{J(k)}$ be the $F \times F$ matrix as defined in the previous paragraph, where F is our scaling factor. Then the up-sampling operator A'_J of v with respect to the set J is defined as follows:

$$(A'_J v) = \sum_{k=1}^{|J|} v \otimes (\mathbf{1})_{J(k)} \tag{2.17}$$

where $J(k)$ is the k-th point in the ordered set J. This can also be thought of as an insertion operation, where we insert each of the J images into their proper place on the grid.

Notice that this operator A'_J is both a right-inverse and adjoint of the down-sampling operator A_J. This is due to the following observation, as well as the basic definitions of each of these operators:

$$A_J(A'_J v) = A_J \sum_{k=1}^{|J|} v \otimes (\mathbf{1})_{J(k)} = v \tag{2.18}$$

Though these computations can easily get very messy if we tried to write them out, one can look at the definitions to get an intuition of why this is the case. We are simply inserting the values of v into a large matrix, and then resampling those values from the same points that they were inserted into. On the other hand, the up-sampling operator is not a left-inverse of A_J, due to the inherent information loss in the down-sampling operation, unless the set J consists of all possible shift vectors, i.e. $|J| = F^2$. The up-sampling operator A'_J is also in fact the adjoint operator to the down-sampling operator A_J.

$$\langle A_J u, v\rangle = \langle u, A'_J v\rangle \tag{2.19}$$

This is also an intuitive result, because if we reduce these inner products to sums, we would be able to see that the non-zero terms are all the same. We provide an example of this operator with Figure 2.5.

2.4.4 Super-resolution

In Chapters 4 and 5, we will consider the operator A_J as an operator that we wish to invert. Here, we have v as a given stack of images, and the set J will be chosen visually based on the actual image data. This is made easier by performing an edge-detection operation on each of the image frames before-hand. Edge detection is essentially a way of detecting steep changes or discontinuities in an image, and in our case is accomplished by performing a convolution of the image with a heavily weighted kernel. A slightly more complicated method of accomplishing this is by estimating the optical flow to determine which direction the scene is moving. For additional information about image super-resolution, see [8]. We will try to find an unknown $u \in \mathbb{R}^{M\times N}$ such that the following equation is approximately solved:

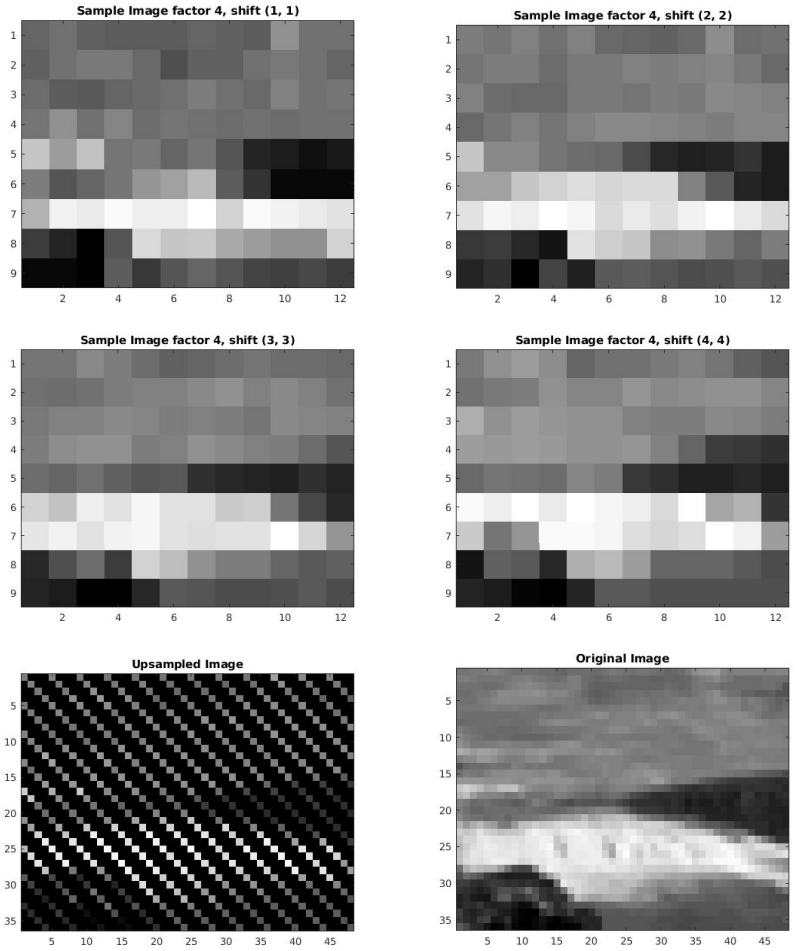
$$A_J u = v \tag{2.20}$$

which is not a very well defined system, so we should instead minimize the following functional:

$$\|A_J u - v\|_{L^2}^2 \to \min_u \tag{2.21}$$

Later on, we will also introduce regularization and total variation, which in practice is necessary due to the ill-posedness of this problem. The norm of the 3-D array of images is defined to be the Euclidean norm of the vector of norms given by each individual layer. Simply stated, this is the square root of the sum of all possible entries in the array. Note that these methods assume that when we down-sample, we are actually sampling one single pixel from the unknown higher resolution image. In practice this will likely not be true, though in theory it will still provide a good basis for this problem.

Figure 2.5 Top four images: Low-resolution images at their indicated shifts
Bottom-left: Upsampled image Bottom-right: Original.



Chapter 3

Total Variation and Variational Calculus

3.1 General Theory

Consider that we have some image u. In this chapter, we will discuss the mathematics of total variation. This is a particularly useful construct for image optimization as it serves as a nice regularization for solving many inverse problems in image restoration. It has use in a wide variety of applications, and tends to have a smoothing effect on the images that it restores [3]. In comparison to wavelet-based methods which rely on noise thresholding, total variation has the advantage of being more intuitive, and better suited to use in natural smooth images. Thus we will use this method of solving deblurring and denoising problems in later chapters. Here we will mathematically formulate total variation as well as the gradient and divergence, and show some important results from this.

In the continuous case, consider that we have a continuous 2D image function $u : \Omega \to \mathbb{R}$, where $\Omega \subset \mathbb{R}^2$. The total variation is defined as follows [2]:

$$R(u) = \sup \left\{ \iint_\Omega u(x,y) \text{div}\, \omega(x,y) dxdy : \quad \omega \in C_0^1(\Omega; \mathbb{R}^2), |\omega(x,y)| \leq 1 \quad \forall (x,y) \in \Omega \right\} \tag{3.1}$$

where we have zero boundary conditions on Ω, where $\omega : \mathbb{R}^2 \to \mathbb{R}^2$ is a vector-valued function, and where divergence is defined in the following notation:

$$\text{div}\, \omega = \nabla \cdot \omega = \frac{\partial \omega_x}{\partial x} + \frac{\partial \omega_y}{\partial y} \tag{3.2}$$

3.1.1 Divergence Theorem

One particularly useful fact in regards to this is that the divergence and the negative gradient operators are adjoint of one another. That is to say,

$$\langle u, -\text{div}\omega \rangle_{L^2} = \langle \nabla u, \omega \rangle_{L^2} \qquad \forall u \in C^1(\Omega), \quad \omega \in C_0^1(\Omega; \mathbb{R}^2) \tag{3.3}$$

where $\omega : \mathbb{R}^2 \to \mathbb{R}^2$ is defined to be some dual variable function, $u : \mathbb{R}^2 \to \mathbb{R}$ is a real-valued image function, and the gradient ∇u is defined to be as follows:

$$\nabla u = \left(\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y} \right) \tag{3.4}$$

In the continuous domain, this is a direct result of the Divergence Theorem, where $\mathbf{n}$ is the normal vector to the boundary $\partial\Omega$

$$\int_\Omega \nabla \cdot F dx = \int_{\partial\Omega} F \cdot \mathbf{n}\, ds \tag{3.5}$$

The product rule for the divergence can be stated as follows, where $u(x, y) \in \mathbb{R}$ and $\omega(x, y) \in \mathbb{R}^2$:

$$\nabla \cdot (u\omega) = u \cdot \text{div}\omega + \omega \cdot \nabla u \tag{3.6}$$

We also know that since $\omega \in C_0^1(\Omega; \mathbb{R}^2)$, that it must be zero everywhere at the boundary $\partial\Omega$. We can substitute $F(x, y)$ for the product of scalar $u(x, y)$ and vector $\omega(x, y)$ and use this to obtain

$$\int_{\partial\Omega} (u\omega) \cdot \mathbf{n}\, ds = 0 \tag{3.7}$$

Furthermore, we can use the product rule and Divergence Theorem to obtain the following result:

$$\langle u, \text{div}\omega \rangle_{L^2} + \langle \nabla u, \omega \rangle_{L^2} = \int_\Omega \nabla \cdot (u\omega) dx = \int_{\partial\Omega} (u\omega) \cdot \mathbf{n}\, ds = 0 \tag{3.8}$$

This shows us that the result (3.3) holds. In the following section, we will discretize the gradient and divergence and develop a similar result from there. We will see that these results are closely related to one another because the discrete definitions of the divergence and gradient are close approximations of their continuous counterparts. This will also give us a theoretical basis for solving inverse problems using the total variation.

3.2 Discrete Total Variation

Next we will define $R(u)$ to be the total variation of the image u, where u is a discrete image in a two-dimensional plane. For simplicity, we are only considering the case where we have a grey-scale image. Ultimately, the goal behind having a discrete formulation such as this is that it becomes possible to perform numerical computations with. We define the discrete total variation as follows:

$$R(u) = \sum_{i=1}^{M} \sum_{j=1}^{N} \|(\nabla u)_{i,j}\|_{\mathbb{R}^2} \tag{3.9}$$

where the discrete gradient $(\nabla u)_{i,j}$ of the image u is defined as follows [2]:

$$(\nabla u)^x_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j}, & \text{if } i < M \\ 0, & \text{if } i = M \end{cases} \tag{3.10}$$

$$(\nabla u)^y_{i,j} = \begin{cases} u_{i,j+1} - u_{i,j}, & \text{if } j < N \\ 0, & \text{if } j = N \end{cases} \tag{3.11}$$

and we combine these two components to define the discrete gradient ∇u as follows:

$$(\nabla u)_{i,j} = ((\nabla u)^x_{i,j}, (\nabla u)^y_{i,j}) \tag{3.12}$$

Equivalently, we can also extend the boundaries by setting

$$u_{M+1,j} = u_{M,j} \tag{3.13}$$

$$u_{i,N+1} = u_{i,N} \tag{3.14}$$

to obtain the following simple formula

$$(\nabla u)^x_{i,j} = u_{i+1,j} - u_{i,j} \tag{3.15}$$

$$(\nabla u)^y_{i,j} = u_{i,j+1} - u_{i,j} \tag{3.16}$$

The discrete divergence for $p = (p^x, p^y)$ is defined in the following way, where p^x and p^y are dual variables in $\mathbb{R}^{M \times N}$, and where p is a dual variable in the set $V = U \times U$:

$$(\text{div}p)^x_{i,j} = \begin{cases} p^x_{i,j} - p^x_{i-1,j}, & \text{if } 1 < i < M, \\ p^x_{i,j}, & \text{if } i = 1, \\ -p^x_{i-1,j}, & \text{if } i = M \end{cases} \tag{3.17}$$

$$(\text{div}p)^y_{i,j} = \begin{cases} p^y_{i,j} - p^y_{i,j-1}, & \text{if } 1 < j < N, \\ p^y_{i,j}, & \text{if } j = 1, \\ -p^y_{i,j-1}, & \text{if } j = N \end{cases} \tag{3.18}$$

We sum these two components to define the divergence of p in the following way:

$$\text{div}p = (\text{div}p)^x_{i,j} + (\text{div}p)^y_{i,j} \tag{3.19}$$

3.2.1 Discrete Divergence Theorem

What we wish to establish now is that the negative divergence and the discrete gradient are adjoint in the real inner product domain. The motivation for knowing this is that in Chapter 4, it will allow us to transform our minimization problem into a problem that we can use our algorithm to solve. This is not a trivial result, so we will briefly show here that the following equality described in (3.3) holds by using the discrete definitions of divergence and gradient and expanding the sums.

$$\begin{aligned}
\langle u, -(\text{div}p)^x \rangle &= \sum_{i,j} -u_{i,j}(\text{div}p)^x_{i,j} \\
&= \sum_{j=1}^{N}\sum_{i=1}^{M} -u_{i,j}(\text{div}p)^x_{i,j} \\
&= \sum_{j=1}^{N}\left(u_{M,j}p^x_{M-1,j} - u_{1,j}p^x_{1,j} - \sum_{i=2}^{M-1} u_{i,j}(p^x_{i,j} - p^x_{i-1,j}) \right) \\
&= \sum_{j=1}^{N}\left(u_{M,j}p^x_{M-1,j} - u_{1,j}p^x_{1,j} + \sum_{i=2}^{M-1} u_{i,j}p^x_{i-1,j} - \sum_{i=2}^{M-1} u_{i,j}p^x_{i,j} \right)
\end{aligned}$$

And now, noting that we can split the sums, we can further rewrite this in terms of the discrete gradient:

$$\begin{aligned}
&= \sum_{j=1}^{N}\left(\sum_{i=2}^{M} u_{i,j}p^x_{i-1,j} - \sum_{i=1}^{M-1} u_{i,j}p^x_{i,j} \right) \\
&= \sum_{j=1}^{N}\left(\sum_{i=1}^{M-1} u_{i+1,j}p^x_{i,j} - \sum_{i=1}^{M-1} u_{i,j}p^x_{i,j} \right) \\
&= \sum_{j=1}^{N}\left(\sum_{i=1}^{M-1} (u_{i+1,j} - u_{i,j})p^x_{i,j} \right) \\
&= \sum_{i,j} p^x_{i,j}(\nabla u)^x_{i,j} \\
&= \langle (\nabla u)^x, p^x \rangle
\end{aligned}$$

The proof for showing the result in the y-dimension is quite similar:

$$\begin{aligned}
\langle u, -(\text{div}p)^y \rangle &= \sum_{i=1}^{M} \sum_{j=1}^{N} -u_{i,j} (\text{div}p)^y_{i,j} \\
&= \sum_{i=1}^{M} \left(u_{i,N} p^y_{i,N-1} - u_{i,1} p^y_{i,1} - \sum_{j=2}^{N-1} u_{i,j} (p^y_{i,j} - p^y_{i,j-1}) \right) \\
&= \sum_{i=1}^{M} \left(\sum_{j=2}^{N} u_{i,j} p^y_{i,j-1} - \sum_{j=1}^{N-1} u_{i,j} p^y_{i,j} \right) \\
&= \sum_{i=1}^{M} \left(\sum_{j=1}^{N-1} (u_{i,j+1} - u_{i,j}) p^y_{i,j} \right) \\
&= \sum_{i,j} p^y_{i,j} (\nabla u)^y_{i,j} \\
&= \langle (\nabla u)^y, p^y \rangle
\end{aligned}$$

And the two results are easily combined as follows:

$$\begin{aligned}
\langle u, -\text{div}p \rangle &= \langle u, -(\text{div}p)^x \rangle + \langle u, -(\text{div}p)^y \rangle \\
&= \langle (\nabla u)^x, p^x \rangle + \langle (\nabla u)^y, p^y \rangle \\
&= \langle \nabla u, p \rangle
\end{aligned}$$

Thus, we can also rewrite the discrete total variation as follows:

$$\begin{aligned}
R(u) &= \sum_{i,j} \| (\nabla u)_{ij} \| \\
&= \sup_{\|p\|_\infty \leq 1} \langle \nabla u, p \rangle \\
&= \sum_{i,j} \sup_{\|p_{ij}\| \leq 1} \langle (\nabla u)_{ij}, p_{ij} \rangle \\
&= \sup_{\|p\|_\infty \leq 1} \left(- \sum_{i,j} u_{ij} (\text{div}\, p)_{ij} \right) \\
&= \sup_{\|p\|_\infty \leq 1} \langle u, \text{div}\, p \rangle
\end{aligned}$$

which as we can now see, is the same definition that we had for total variation in the continuous case. In the following chapter, these results will enable us to formulate the algorithms needed for image reconstruction and optimization.

Chapter 4

Primal-dual Optimization Methods

4.1 General Theory

Here we will develop the algorithms that are used in Chapter 5. We must start by introducing the general theory of primal-dual problems for convex functions. The problem of image restoration with total variation is fortunately a convex problem, so this will be particularly useful for that development. Refer to [6] for more background on convex analysis, and refer to [5] for a development of the primal-dual problem. To put it simply, a function $f : \mathbb{R}^n \to \mathbb{R}$ is a convex function if for $\lambda \in [0, 1]$ and $x, y \in \mathbb{R}^n$, we have

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y) \tag{4.1}$$

Moreover, a function f is defined as concave if $-f$ is convex. A function is proper convex or proper concave if the inequality in the definition is proper. Furthermore, we want our functions to be coercive. A coercive function is defined as a function f such that

$$f(x) \to \infty \text{ as } \|x\| \to \infty \tag{4.2}$$

Convex functions have many useful theorems and properties which we will briefly summarize here. The most important result is that if f is continuous, coercive, and convex, then it has at least one global minimum. Now let us define X and Y to be two vector spaces, each with an associated norm and inner product. We assume $K : X \to Y$ to be a continuous linear map with norm $\|K\|$:

$$\|K\| = \sup_{x \in X} \{\|Kx\|_Y : x \in X \text{ where } \|x\|_X \leq 1\} \tag{4.3}$$

Consider $F : Y \to \mathbb{R} \cup \{+\infty\}$ and $G : X \to \mathbb{R} \cup \{+\infty\}$ to be two continuous, coercive, convex functions on their respective domains. This means that at least global minimum exists for each of them. Since the sum of two convex, continuous, and coercive functions holds those same properties itself, this means the sum $F + G : X \times Y \to \mathbb{R} \cup \{+\infty\}$ also

obtains a global minimum. What we wish to consider is how to reformulate and solve the following minimization problem, written in primal form:

$$\min_{x \in X} F(Kx) + G(x) \tag{4.4}$$

Before this is done, we need to introduce the concept of duality and convex conjugates. We define the convex conjugate of F to be the function $F^* : Y \to \mathbb{R}$:

$$F^*(y^*) = \sup_{y \in Y} \{\langle y^*, y \rangle_Y - F(y)\} \tag{4.5}$$

The convex conjugate of G is similarly defined:

$$G^*(x^*) = \sup_{x \in X} \{\langle x^*, x \rangle_X - G(x)\} \tag{4.6}$$

The operator $K^* : Y \to X$ is defined to be the adjoint of K, defined by the following:

$$\langle Kx, y \rangle_Y = \langle x, K^* y \rangle_X \tag{4.7}$$

We can now rewrite F^* and G^* as follows, based on the fact that K is linear and bounded and F and G are continuous and proper convex.

$$F^*(y) = \max_{p \in Y} \{\langle p, y \rangle_Y - F(p)\} \tag{4.8}$$

$$G^*(x) = \max_{q \in X} \{\langle x, q \rangle_X - G(q)\} \tag{4.9}$$

From here, it makes sense to introduce Fenchel's dual problem, which is a reformulation of (4.4) based on the given definitions of F^* and G^*:

$$\max_{y \in Y} \{-F^*(y) - G^*(-K^* y)\} \tag{4.10}$$

More importantly, we can now write it as a primal-dual problem, where the solution is a saddle-point:

$$\min_{x \in X} \max_{y \in Y} \langle Kx, y \rangle_Y - F^*(y) + G(x) \tag{4.11}$$

This is the formulation that we will use to develop the Chambolle-Pock Algorithm in the following section.

4.2 Chambolle-Pock Algorithm

The Chambolle-Pock Algorithm as defined in [5] is an algorithm which should converge to the solution of the min-max problem (4.11) if the proper conditions are satisfied. We must first define the sub-gradient set ∂G in the following way. Assuming that G is convex, the sub-gradient $\partial G(x)$ is the set of all values v in the convex open set of $G : X \to \mathbb{R}$ such that:

$$\{v \in X : G(y) \geq G(x) + \langle v, y - x \rangle_X \quad \forall y \in X\} \tag{4.12}$$

In the algorithm however, we will always select one element, and this will also be denoted $\partial G(x)$. If G is a differentiable function, then $\partial G(x)$ will only consist of the one element $\nabla G(x)$.

We initialize the algorithm by selecting parameters $\tau, \theta > 0$ and $\theta \in [0, 1]$. We also pick starting points $x^0 \in X$ and $y^0 \in Y$, with $\tilde{x}^0 = x^0$. The operator I is defined to be the identity operator. For $n \geq 0$, we iterate each step as follows:

$$\begin{cases} y^{n+1} = (I + \sigma \partial F^*)^{-1}(y^n + \sigma K \tilde{x}^n), \\ x^{n+1} = (I + \tau \partial G)^{-1}(x^n - \tau K^* y^{n+1}), \\ \tilde{x}^{n+1} = x^{n+1} + \theta(x^{n+1} - x^n) \end{cases} \tag{4.13}$$

For additional clarity, we can rewrite the above iterative algorithm as follows:

$$\begin{cases} (I + \sigma \partial F^*) y^{n+1} &= (y^n + \sigma K \tilde{x}^n), \\ (I + \tau \partial G) x^{n+1} &= (x^n - \tau K^* y^{n+1}), \\ \tilde{x}^{n+1} &= x^{n+1} + \theta(x^{n+1} - x^n) \end{cases} \tag{4.14}$$

These are not necessarily trivial problems to solve. Now suppose we assume that (4.11) has a saddle-point solution $(\hat{x}, \hat{y})$. As was shown in [5], if we let $\theta = 1$, then we know that if X and Y are finite-dimensional, the algorithm converges to a saddle-point (x^*, y^*) such that $x^n \to x^*$ and $y^n \to y^*$ if the following regularity condition is satisfied:

$$\tau \sigma \|K\|^2 < 1 \tag{4.15}$$

These are indeed very interesting results, and we will use them to develop our denoising and deconvolution algorithms in the following sections.

4.3 Denoising Applications

The methods we use to denoise an image intend to make an image less noisy by minimizing the total variation and the residual, with regularization parameter λ. It must therefore provide a best solution to the problem (4.16), where v is the given data, and u is the unknown. To do this, we start by defining it as follows:

$$\frac{\lambda}{2} \|u - v\|_{L^2}^2 + R(u) \to \min_u \tag{4.16}$$

where $R(u)$ is the total variation defined in (3.9). This total variation is a regularization functional which we need in order for the algorithm to function properly. We can approximate the minimization problem (4.16) by using the discrete divergence operator, and thereby use a projected dual gradient descent method to solve the problem. From this construction, we can use our gradient descent method to minimize (4.16)

We will be using a discrete version of the ROF constrained optimization problem [3]. In our case, we write the optimization problem in primal form as in (4.4), where $X = \mathbb{R}^{M\times N}$, $Y = X \times X$, $K = \nabla$, and the following definitions are made:

$$F(q) = \sum_{i,j} \|q_{i,j}\| \tag{4.17}$$

$$G(u) = \frac{\lambda}{2} \|u - v\|^2 \tag{4.18}$$

Furthermore, we can see that the following relationship between F and the total variation R holds:

$$F(Ku) = R(u) = \sum_{i,j} \|(\nabla u)_{i,j}\| \tag{4.19}$$

In order to utilize the algorithms suggested by [5], we write the saddle-point problem (4.11) in its general form, where the operator A is the gradient operator.

$$\langle \nabla u, p\rangle + G(u) - F^*(p) \to \min_{u\in U} \max_{p\in V} \tag{4.20}$$

We also define $F^* : V \to \mathbb{R}$ to be the convex conjugate or dual of $F : V \to \mathbb{R}$. This means that

$$F(q) = \sum_{i,j} \sqrt{(q^x_{i,j})^2 + (q^y_{i,j})^2} \tag{4.21}$$

$$F^*(p) = \sup_{q\in V}(\langle q, p\rangle - F(q)) \tag{4.22}$$

$$= \sup_{q\in V} \sum_{i,j} \left(q^x_{i,j}p^x_{i,j} + q^y_{i,j}p^y_{i,j} - \sqrt{(q^x_{i,j})^2 + (q^y_{i,j})^2} \right) \tag{4.23}$$

$$= \sum_{i,j} \sup_{q\in V} \left(q^x_{i,j}p^x_{i,j} + q^y_{i,j}p^y_{i,j} - \sqrt{(q^x_{i,j})^2 + (q^y_{i,j})^2} \right) \tag{4.24}$$

$$= \sum_{i,j} \begin{cases} 0 & \text{if} \quad \|p_{i,j}\|_2 \le 1 \\ +\infty & \text{otherwise} \end{cases} \tag{4.25}$$

$$= \delta_P(p) \tag{4.26}$$

where we define $\delta_P(p)$ to be the following indicator function:

$$\delta_P(p) = \begin{cases} 0 & \text{if} \quad p \in P, \\ +\infty & \text{otherwise} \end{cases} \tag{4.27}$$

Here, we have the condition that P is the ball of radius 1 with respect to the $\|\cdot\|_\infty$ norm. As we have shown in Chapter 3, the adjoint of the divergence is the negative gradient. Accordingly, we have the primal-dual problem written as follows:

$$\frac{\lambda}{2}\|u - v\|^2_{L^2} - \langle u, \text{div}p\rangle_U - \delta_P(p) \to \max_{p\in V} \min_{u\in U} \tag{4.28}$$

The reason that this is equivalent to the previous definition is that this construction restricts $p \in P$, and also because $R(u) = F(\nabla u)$, and $\langle \nabla u, p\rangle = -\langle u, \operatorname{div} p\rangle$. We could also write this as a constrained optimization problem in the following way, based on (4.27):

$$\frac{\lambda}{2}\|u - v\|_{L^2}^2 - \langle u, \operatorname{div} p\rangle_U \to \min_{u\in U} \max_{p\in V} \qquad \text{subject to } \|p\|_\infty \le 1 \tag{4.29}$$

where we define the $\|\cdot\|_\infty$ norm as follows:

$$\|p\|_\infty^2 = \max_{i,j}\left((p_{i,j}^x)^2 + (p_{i,j}^y)^2\right) \tag{4.30}$$

where in our case, we have that $G(u)$ is the residual term and $F(q)$ is the total variation of u, where $q = \nabla u$. Note that $U = \mathbb{R}^{M\times N}$ and $V = U \times U$. The supremum of the above equation is obtained when $q = p$, if $p \in P$.
Since we know that $\|q_{i,j}\|_2^2 - \|q_{i,j}\|_2^2 > 0$ whenever $\|q_{i,j}\|_2^2 > 1$, the sum will diverge.

When G is differentiable, then ∂G is simply the unique gradient. The following holds:

$$G(u) = \frac{\lambda}{2}\|u - b\|_{L^2}^2 \qquad \partial G(u) = \lambda(u - b) \tag{4.31}$$

To select $v \in \partial F^*(p)$, consider the following condition, based on the definition of the subgradient (4.12). If we have $\|p\|_\infty = \max \|p_{ij}\| \le 1$, then

$$\begin{aligned} v \in \partial F^*(p) \qquad &\iff \qquad 0 \ge 0 + \langle v, q - p\rangle \quad \forall \|q\|_\infty \le 1 \\ &\iff \qquad \langle v, q\rangle \le \langle v, p\rangle \quad \forall \|q\|_\infty \le 1 \\ &\iff \qquad \langle v_{ij}, q_{ij}\rangle \le \langle v_{ij}, p_{ij}\rangle \qquad \forall i, j,\ \|q_{i,j}\| \le 1 \end{aligned}$$

Thus, if at a given point (i, j) we have $\|p_{ij}\| \le 1$, then we should choose $q_{ij} = p_{ij}$ and obtain $v_{ij} = 0$. Otherwise, if we have $\|p_{ij}\| \ge 1$, then we must project it pointwise. To do that, we must select some $\lambda_{ij} \ge 0$ and obtain $v_{ij} = \lambda_{ij} p_{ij}$ so that we can then use the Cauchy-Schwartz inequality to have the following, while restricting $\|q_{ij}\| \le 1$:

$$\langle v_{ij}, q_{ij}\rangle = \lambda_{ij}\langle p_{ij}, q_{ij}\rangle \le \lambda_{ij}\|p_{ij}\|\,\|q_{ij}\| \le \lambda_{ij}\|p_{ij}\| = \lambda_{ij}\langle p_{ij}, p_{ij}\rangle = \langle v_{ij}, p_{ij}\rangle$$

The algorithm (4.13) is then as follows:

Given initial $\tau, \sigma > 0$, $\theta \in [0, 1]$, $u_0 \in U$, $p_0 \in V$, with $\tilde{u}_0 = u_0$. Iterate with $n \ge 0$:

$$\begin{aligned} \tilde{p}_n &= (p_n + \sigma \cdot \nabla \tilde{u}_n) \\ p_{n+1} &= (I + \sigma \partial F^*)^{-1}(\tilde{p}_n) \\ u_{n+1} &= (I + \tau \partial G)^{-1}(u_n + \tau \cdot \operatorname{div} p_{n+1}) \\ \tilde{u}_{n+1} &= (1 + \theta)u_{n+1} - \theta u_n \end{aligned}$$

The above equations can also be rewritten as follows, by taking the identity and subgradient operators to both sides of the second and third equations above.

$$\begin{aligned} (p_n + \sigma \cdot \nabla \tilde{u}_n) &= \tilde{p}_n \\ p_{n+1} + \sigma \partial F^*(p_{n+1}) &= \tilde{p}_n \\ u_{n+1} + \tau \partial G(u_{n+1}) &= (u_n + \tau \cdot \operatorname{div} p_{n+1}) \\ \tilde{u}_{n+1} &= (1 + \theta)u_{n+1} - \theta u_n \end{aligned}$$

This can further be written in the following way, substituting ∂G for its well defined functional value

$$\begin{aligned} p_{n+1} + \sigma \partial F^*(p_{n+1}) &= \tilde{p}_n \\ (1 + \lambda\tau)u_{n+1} &= (u_n + \tau \cdot \text{div} p_{n+1} + \tau\lambda b) \\ \tilde{u}_{n+1} &= (1+\theta)u_{n+1} - \theta u_n \end{aligned}$$

where $b \in U$ is some given data. Based on the previous derivation of ∂F^*, the operator $(I + \sigma\partial F^*)^{-1}$ ultimately becomes a point-wise projection operator. This means that element of p^{n+1} either remains the same, or is normalized and projected onto a unit ball. and we have the following implementation of the algorithm, which we can easily implement on the computer:

$$\begin{aligned} \tilde{p}_n &= (p_n + \sigma \cdot \nabla \tilde{u}_n) \\ p_{i,j}^{n+1} &= \frac{\tilde{p}_{i,j}^n}{\max_{i,j}(1, |\tilde{p}_{i,j}^n|)} \\ u_{n+1} &= \frac{u_n + \tau \cdot \text{div} p_{n+1} + \lambda\tau b}{1 + \lambda\tau} \\ \tilde{u}_{n+1} &= (1+\theta)u_{n+1} - \theta u_n \end{aligned}$$

The parameter τ determines the step-size. If it is too large, then the method may not converge at all. The parameter θ determines whether this is an implicit or explicit method. If $\theta = 0$, then no intermediate update or averaging step is performed, and this may also impact the algorithm's stability. If $\theta = -\frac{1}{2}$, then it will average the last two iterations of u, and so on. The parameter σ determines the dual-step. It thus plays a role in how the dual variable p is computed.

In Chapter 5, we will see some examples of this algorithm implemented in MATLAB, which will converge to a saddle point solution of (4.28) so long as the regularity condition described in (4.15) is satisfied.

$$\tau\sigma\|\nabla\|^2 < 1 \tag{4.32}$$

Furthermore, we have an upper bound for the value of $\|\nabla\|^2$. Using the definition of the gradient (3.10) and the operator norm (4.3), we obtain the following result:

$$\begin{aligned} \|\nabla u\|^2 &= \sum_{i,j} \left\| \begin{pmatrix} u_{i+1,j} - u_{i,j} \\ u_{i,j+1} - u_{i,j} \end{pmatrix} \right\|^2 \\ &\leq \left(\left\| \begin{matrix} u \\ u \end{matrix} \right\| + \left\| \begin{matrix} u \\ u \end{matrix} \right\| \right)^2 \\ &= 4 \left\| \begin{matrix} u \\ u \end{matrix} \right\|^2 = 8 \|u\|^2 \end{aligned}$$

and we thus have the operator norm satisfying the following condition

$$\|\nabla\|^2 = \sup_{x \in X} \left\{ \frac{\|\nabla x\|}{\|x\|} : x \in X \right\}^2 \leq 8 \tag{4.33}$$

This means that the following is a sufficient condition for convergence of the algorithm:

$$\tau\sigma < \frac{1}{8} \tag{4.34}$$

Furthermore, a lower bound for the operator norm can be found numerically by computing the norm of the gradient of any chosen image. For example, a very large checkerboard image with alternating zeros has a squared norm of approximately 4. This means for example that the following is a necessary condition for regularity:

$$\tau\sigma < \frac{1}{4} \tag{4.35}$$

We will see this in Chapter 5 when we look at how we select the value of τ.

4.4 Applications to General Image Restoration

The methods we use to deconvolute an image seek to provide a best solution to an equation such as $Au = v$, where A is some given operator, v is the given data, and u is the unknown. In the context of imaging, u and v will be images of some given size, and A will be some image operation such as blurring or sub-sampling. The function $R(u)$ is still defined as (3.9). A natural extension to the problem (4.16) is therefore the following:

$$\frac{\lambda}{2}\|Au - v\|_2^2 + R(u) \to \min_u \tag{4.36}$$

We can imagine A to be some type of a blurring or down-sampling operator, where v is the blurred image. The algorithm for solving this problem is similar to the previous algorithm with the following modifications to the function G:

$$G(u) = \frac{\lambda}{2}\|Au - b\|_{L^2}^2 \qquad\qquad \partial G(u) = \lambda A^*(Au - b)$$

where F and F^* are still defined as they were in the previous section. This then leads to the following algorithm, which parts of which are taken from the previous algorithm:

$$\begin{aligned} p_{n+1} + \sigma\partial F^*(p_{n+1}) = \tilde{p}_n &= (p_n + \sigma \cdot \nabla\tilde{u}_n) \\ u_{n+1} + \lambda\tau(A^*(Au_{n+1} - b)) &= (u_n + \tau \cdot \text{div}p_{n+1}) \\ \tilde{u}_{n+1} &= (1+\theta)u_{n+1} - \theta u_n \end{aligned}$$

which can further be written as:

$$\begin{aligned} p_{n+1} + \sigma\partial F^*(p_{n+1}) = \tilde{p}_n &= (p_n + \sigma \cdot \nabla\tilde{u}_n) \\ (I + \lambda\tau A^*A)u_{n+1} &= (u_n + \tau \cdot \text{div}p_{n+1} + \lambda\tau A^*b) \\ \tilde{u}_{n+1} &= (1+\theta)u_{n+1} - \theta u_n \end{aligned}$$

where $b \in U$ is some given data. Thus we have:

$$\begin{aligned} p_{i,j}^{n+1} &= \frac{\tilde{p}_{i,j}^{n}}{\max_{i,j}(1, |\tilde{p}_{i,j}^{n}|)} \\ u_{n+1} &= (I + \lambda\tau A^*A)^{-1}(u_n + \tau \cdot \text{div}p_{n+1} + \lambda\tau A^*b) \\ \tilde{u}_{n+1} &= (1+\theta)u_{n+1} - \theta u_n \end{aligned}$$

The operator $(I + \lambda\tau A^*A)^{-1}$ is certainly not a trivial one. Since there is not always a simple analytical way to resolve it, we will have to solve another separate linear equation, using the conjugate gradient method. We can apply this conjugate gradient method to find the value u^{n+1}. See the following section for a brief description of this method. For further information on these and other gradient descent methods, see [7, Chap. 5]

4.4.1 Conjugate Gradient Descent

The conjugate gradient method is iterative in nature and involves solving the equation

$$Cu^{n+1} = d \tag{4.37}$$

where C is an operator, and d is a vector. In the image restoration problem that we are dealing with, these can be defined as the following:

$$C = (I + \lambda\tau A^*A) \tag{4.38}$$

$$d = (u_n + \tau \cdot \text{div}p_{n+1} + \lambda\tau A^*b) \tag{4.39}$$

We start with an initial guess, for instance $x_0 = u^n$, and proceed with the algorithm as follows:

Algorithm 1 Conjugate Gradient Method

$\mathbf{r_0} \leftarrow \mathbf{d} - C\mathbf{x_0}$
$\mathbf{p_0} \leftarrow \mathbf{r_0}$
$k \leftarrow 0$
while $\|\mathbf{r_0}\| > tol$ **do**
  $\alpha_k \leftarrow \langle \mathbf{r_k}, \mathbf{r_k} \rangle / \langle C\mathbf{p_k}, \mathbf{p_k} \rangle$
  $\mathbf{x_{k+1}} \leftarrow \mathbf{x_k} + \alpha_k \mathbf{p_k}$
  $\mathbf{r_{k+1}} \leftarrow \mathbf{r_k} - \alpha_k C\mathbf{p_k}$
  $\beta_k \leftarrow \langle \mathbf{r_{k+1}}, \mathbf{r_{k+1}} \rangle / \langle \mathbf{r_k}, \mathbf{r_k} \rangle$
  $\mathbf{p_{k+1}} \leftarrow \mathbf{r_{k+1}} + \beta_k \mathbf{p_k}$
  $k \leftarrow k + 1$
end while
return x_{k+1}

What this algorithm essentially does is minimizes the residuals $\mathbf{r_0}$. For this algorithm, $\|r_0\|$ does reach zero for finite dimensional vector spaces. With the discrete images however, these spaces have a lot of dimensions, and thus we set a tolerance that is small but greater than zero, so that it doesn't take too long and so that the solution doesn't become unstable from numerical rounding errors.

4.4.2 Super-resolution

On the other hand, in the case of $A = A_J$ being the down-sampling operator (2.10) with scaling factor F, we do have an exact solution for $(I + \lambda\tau A^*A)^{-1}$. Given that A^* is the up-sampling operator, we can describe A^*A as an operator from where some of the pixels of an image are set to zero, while others remain unchanged. Given this, we can define the operator $(I + \lambda\tau A^*A)$ as follows, where p and q are integers:

$$[(I + \lambda\tau A^*A)u]_{ij} = \begin{cases} (1 + \lambda\tau)u_{i,j} & \text{if } \exists k, \quad \begin{pmatrix} i+1 \\ j+1 \end{pmatrix} - J(k) = F \begin{pmatrix} p \\ q \end{pmatrix} \\ u_{ij} & \text{otherwise} \end{cases} \tag{4.40}$$

This is a pointwise linear operator that can easily be inverted as follows.

$$[(I + \lambda\tau A^*A)^{-1}v]_{ij} = \begin{cases} \frac{1}{1+\lambda\tau} v_{i,j} & \text{if } \exists k, \quad \begin{pmatrix} i+1 \\ j+1 \end{pmatrix} - J(k) = F \begin{pmatrix} p \\ q \end{pmatrix} \\ v_{ij} & \text{otherwise} \end{cases} \tag{4.41}$$

In this case, we would not necessarily need the conjugate gradient method to perform this inversion operation. We can formulate this pointwise definition more efficiently in the following way.

$$(I + \lambda\tau A^*A)^{-1} = I - \frac{\lambda\tau}{1 + \lambda\tau} A^*A \tag{4.42}$$

This will ultimately enable faster computations in our numerical implementation.

Chapter 5

Numerical Results

5.1 Description

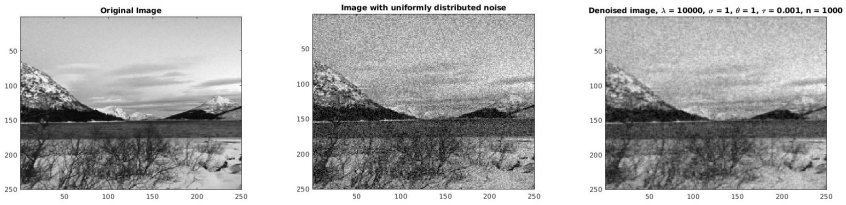
In this chapter, we will describe and implement several numerical experiments. We will visit the use of total variation for solving problems in denoising, deblurring, and image super-resolution. We will explore several different problems for each of these using different parameters and images. We will also try to implement some of these algorithms on real images that are not intentionally convoluted to see how they function. For the deblurring problem we will look at some satellite images, and for the super-resolution problem, we will look at some burst shots and/or videos. Finally we will look at the L-Curve method, which is a method for selecting the parameter λ based on having both the residual and total variation terms as small as possible.

5.2 Denoising

In this section, we will implement some TV-denoising problems to solve the minimization problem in Equation (4.16). We will adopt the algorithm from section 4.3 in the previous chapter to a MATLAB implementation. We start by performing one of our simplest examples of image restoration.

Figure 5.1 illustrates an example of this algorithm. See the code (that will be) appended in Appendix A. Here, we use the algorithm. We see that when applied to uniform noise, the algorithm works moderately well, but is in fact more optimal for other types of noise, such as Gaussian noise. We shall also explore additional examples with other types of noise and different values for the parameters. In this figure, we have an original image of a lake on the left, with artificial noise added in the middle. The algorithm is then applied and the result is on the right. This shows that the denoising algorithm is not perfect, but it does manage to smooth the image somewhat.

Figure 5.1 Minimizing $\frac{\lambda}{2}\|u-v\|_2^2 + R(u)$.
Left: Original Image Center: Noisy image Right: Denoised image, $\lambda = 10^4$



5.3 Deblurring

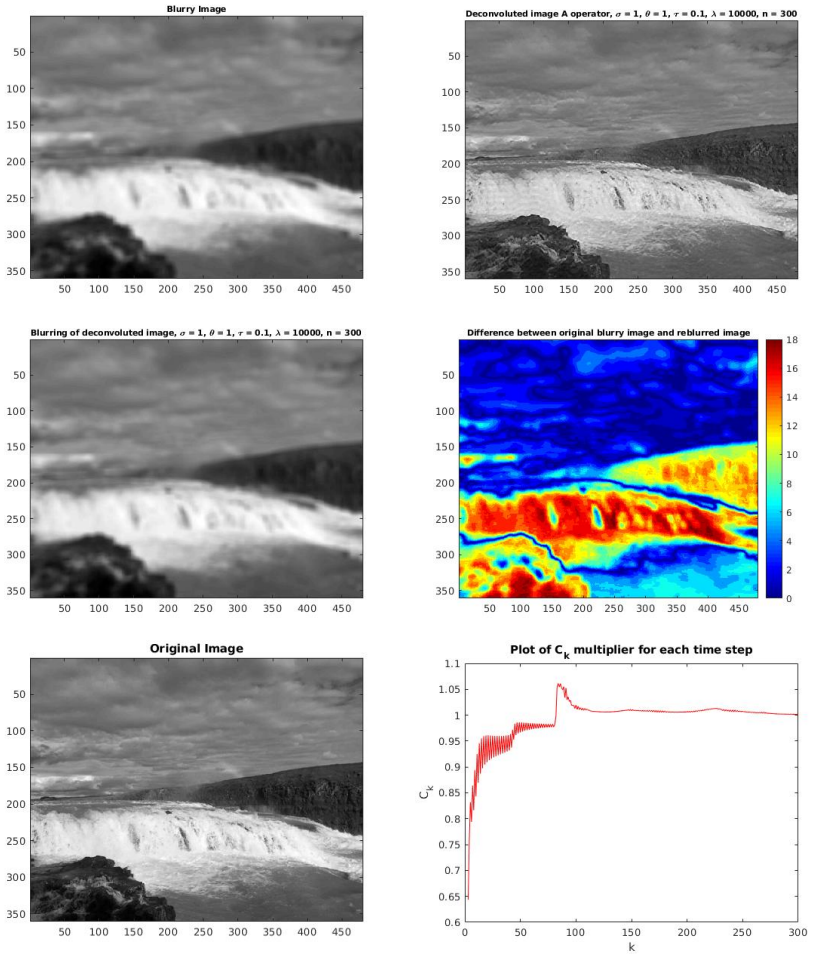
In this section, we will explore some deblurring problems. Here we will adopt the algorithm for solving the minimization problem (4.36) where the operator A is an image convolution operator. For our first numerical test, we will look at this natural image of an Iceland waterfall, as in Figure 5.2. Since the original image was already of a very high quality, we intentionally applied a blurring operator with convolution kernel B to the image in MATLAB. The convolution kernel in this case was a 9x9 uniform averaging matrix. This is therefore a test of the algorithm's functionality. The parameter λ is an important part of the minimization problem and represents how heavily we will weight the total variation term against the residual term. We chose this by using the L-curve method, which will be described later. See Appendix for the MATLAB code.

The results of Figure 5.2 can be thus interpreted. The blurry image is passed into the algorithm and represents the value v given in the problem. The operator A is the convolution of the image with the matrix B, so we have

$$Au = B * u$$

The deconvoluted image (representing u in the equation) gives us a result that more or less accurately depicts all the features of the original image, including some very fine features of the rocks and water that are not all that apparent in the blurry image. One noticeable artifact in the deconvoluted image is a slight kaleidoscope tiling effect. This is likely due to the fact that we have to average each pixel over a 9x9 area, which is large enough to see in the picture. This does however demonstrate that the algorithm works as expected. The colormap image simply shows where the blurred and re-blurred images differ. The red part of the image is where the residuals are high and thus the total variation needed to be optimized more. We can see that the waterfall and the rocks is where it is most red, and that is to be expected because those features had a lot of very fine details that needed to be restored. The plot on the bottom right indicates the value of the amplification factor at each iteration. By this we mean that the value C_k is the norm of the new image divided by the norm of the previous image. Thus we should see that this value approaches 1 as the algorithm converges to its solution.

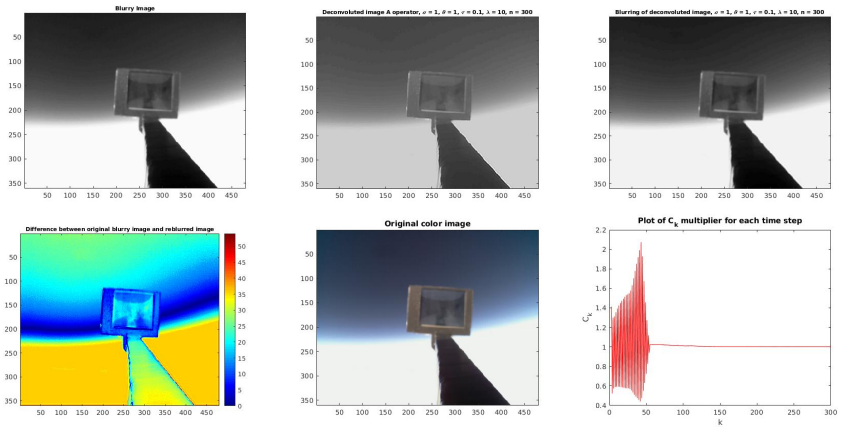
Figure 5.2 Minimizing $\frac{\lambda}{2}\|Au - v\|_2^2 + R(u)$.
Top-left: Original blurry image Top-right: Deconvoluted image with specified λ
Center-left: Reblurred image Center-right: Difference of blurry images
Bottom-left: Original Image Bottom-right: The ratio of consecutive norms.



We will now explore another example of an actual raw image. This image was taken from a small digital camera while attached to a stratospheric balloon in connection with a balloon test that was performed here at NTNU. The LCD screen visible from the camera is displaying an image of a person. Figure 5.3 shows us the results of the algorithm being executed on this image without any initial blurring being performed. The image was taken from about 20 km altitude, and thus was subject to considerable freezing temperatures and condensation, as well as back-lighting from the sun. These images were thus a little blurry, out of focus, and not well lit. We were interested in seeing if we could deblur it. Note that for this particular problem, we are much more interested in the foreground than the background, and have no concern for the background sky.

As we see in the deconvoluted image on the top-right hand corner of Figure 5.3, the edges of the frame and the arm become sharper. The blurring kernel used was that 5x5 kernel which was described in (2.3.2). The reason that we use this kernel is because we determined numerically that kernels don't sharpen the image quite as well. This removes seems to remove some of the blur, but not the lighting glare problems. This is not likely to be solved by these methods, and we might further need to resort to the use of photo-shop in order to accomplish what we really want to do here. This is therefore beyond the scope of what experiments we can do here.

Figure 5.3 Minimizing $\frac{\lambda}{2}\|Au - v\|_2^2 + R(u)$.
Top-left: Original greyscale image Top-right: Deconvoluted image
Center-left: Reblurred grey image Center-right: Difference of blurry images
Bottom-left: Original color image Bottom-right: Norm-multiplier plot

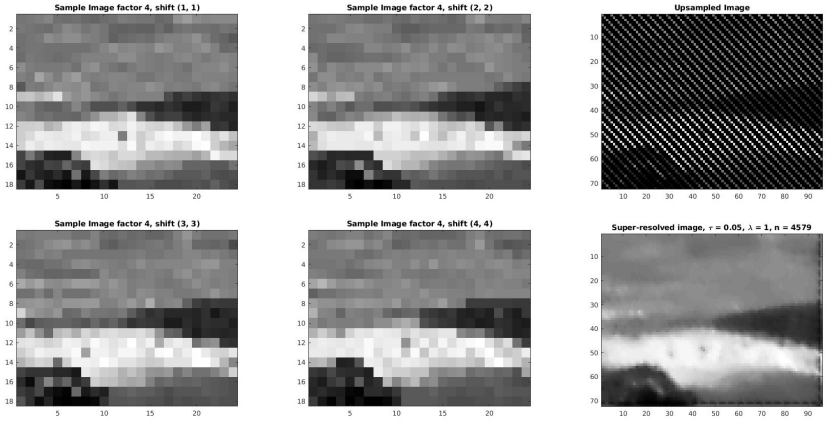


5.4 Image super-resolution

One of our overarching goals here was to solve the problem of image super-resolution. To start, we will show that we can successfully down-sample an image, and then restore those images even with the accompanying loss of information. We refer back to Chapter 2 for

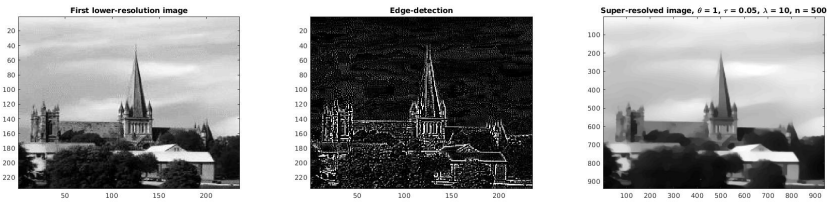
the description of down-sampling and up-sampling. We consider the following example of an image that was sub-sampled along the main diagonal of a 4x4 grid, the scaling factor being 4. In that case, we have $J = \{(1,1),(2,2),(3,3),(4,4)\}$. It is fairly simple then to define the operators and implement the algorithm as follows. As we can see, the algorithm seems to successfully paint in the missing pixels, though we do see some artifacts in the second-to-last rows and columns of the image. This may be due to the way the outer boundaries were defined.

Figure 5.4 Minimizing $\frac{\lambda}{2}\|Au - v\|_2^2 + R(u)$, where A is the down-sampler. The left four images are lower-resolution sub-samples. On the right, we have the up-sampling and in-painting



Now let us look at a brief example of this algorithm applied to an actual video that was not previously down-sampled. In the following example, a video was taken of a cathedral where the camera was moving approximately diagonally in the downward-right facing direction. We then processed this video into a GIF image, or a 3D array. Additionally, we convoluted it with an edge-detection kernel that would help us see where the movement was occuring. Ultimately, we estimated that we could upsample it by a sampling factor of 4 with $J = \{(4,4),(3,3),(2,2),(1,1)\}$. This resulted in the following display. Here we can see that although this resulted in a higher-resolution image, going from 235x235 to 940x940, it did not in fact result in a clearer image. This is not a perfect solution since it relies on imprecise camera movement. For details on the edge-detection, see Appendix B.

Figure 5.5 Left: First frame Center: Edge-highlighted frame Right: Result



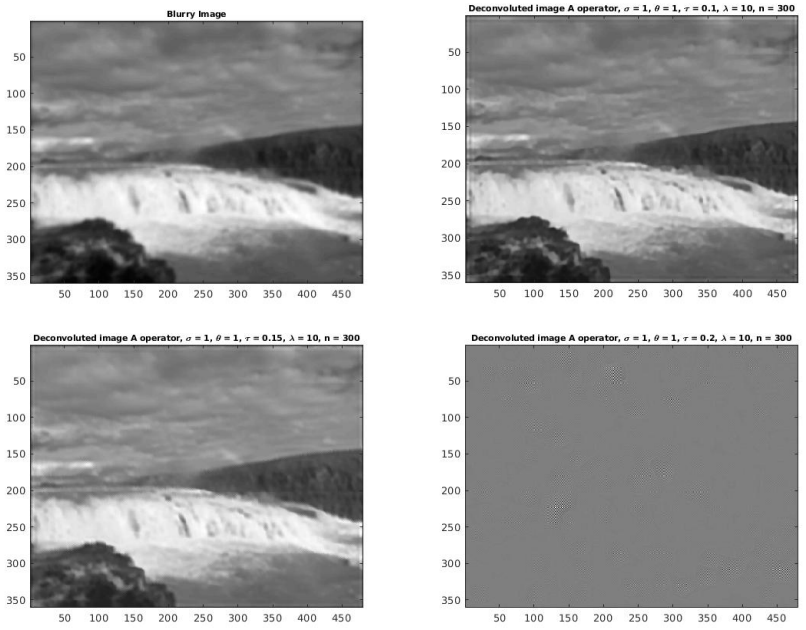
5.5 Parameter Selection

The primal-dual algorithm we use requires a lot of different parameters, including σ, θ, τ, and λ. How we select these parameters is dependent on a variety of factors, including how accurate we want our solution to be and how much time we want it to take. We already know from the previous chapter that (4.15) is a sufficient condition for convergence given that $\theta = 1$. That is, we have the following condition:

$$\tau\sigma\|K\|^2 < 1 \tag{5.1}$$

where we know from previously that $4 \leq \|K\|^2 \leq 8$. We also know that convergence is required if we wish for our numerical solutions to make any sense. A look at the following figure results illustrates just how hard this line is. Through further experimentation, we have found that when $\sigma = 1$, the threshold for τ is actually between 0.16 and 0.17. What this means is that if τ is less than that threshold, then the solution converges perfectly well, but if not then the deconvolution algorithm fails to provide a reasonable result, as we can see in the bottom-right image of Figure 5.6.

Figure 5.6 Parameter selection for τ.
Top-left: Original blurred image Top-right: $\tau = 0.10$
Bottom-left: $\lambda = 0.15$ Bottom-right: $\tau = 0.20$



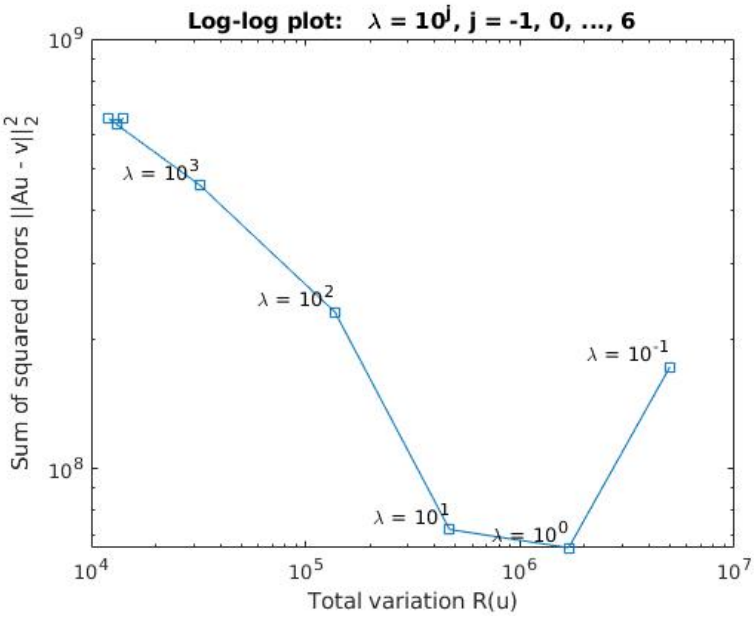
5.5.1 L-Curve Method for selecting λ

A big question we have for these optimization methods is how to select our parameter λ. The main purpose of this parameter is to control how much weight we give to regularization of the optimization problem. Suppose we have the following minimization problem with given data v, given operator A, and regularization parameter λ. Then we have the following equation (4.36)

$$\frac{\lambda}{2}\|Au - v\|_2^2 + R(u) \to \min_u \tag{5.2}$$

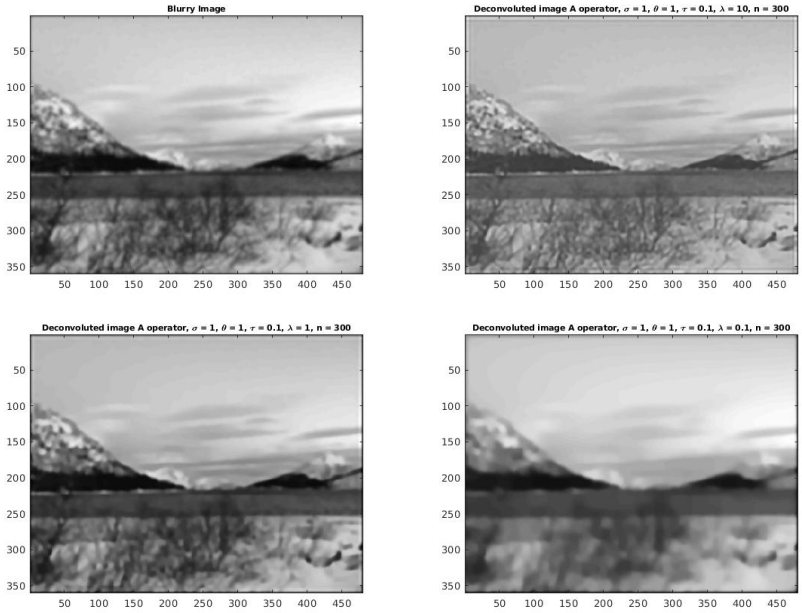
In this case, how would we choose the regularization parameter λ, given that it could lead to vastly different solutions? This is where we introduce the L-Curve method, which we can read more about in [4]. The first thing we do is plot a log-log plot of the residual sums with respect to the total variation (Figure 5.7). Here we will use the image in (5.8), and we will see why this is important. Here, we have that A is a blurring operator that convolutes a Gaussian kernel, and we perform the deblurring algorithm for multiple values of λ.

Figure 5.7 L-curve plot



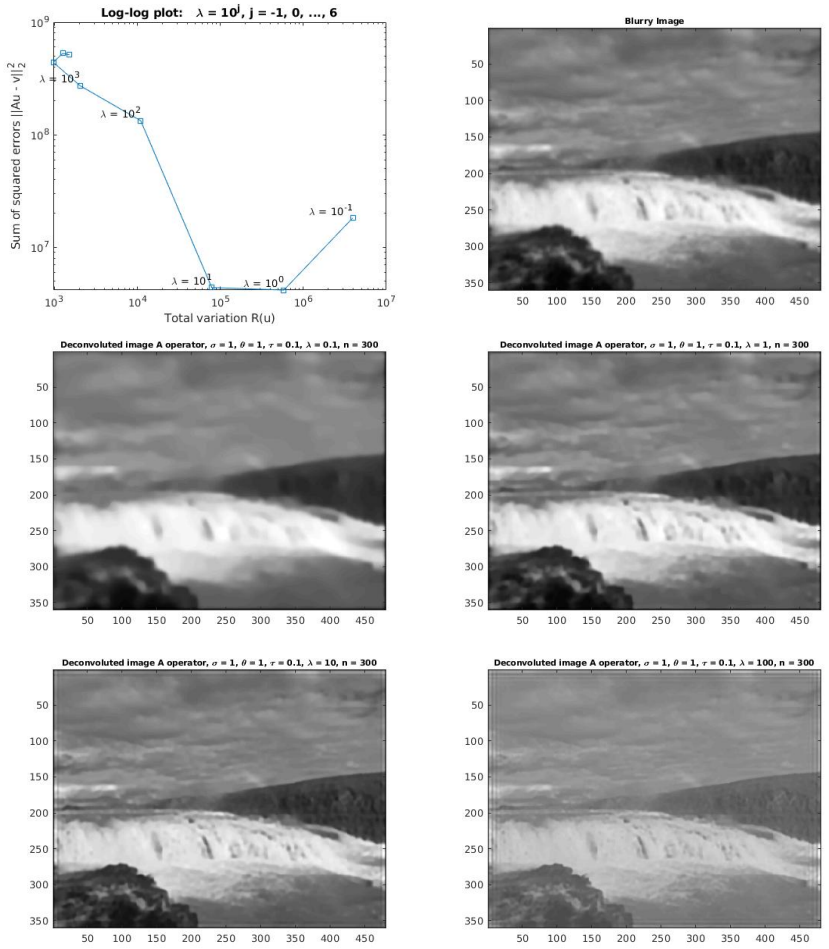
At this stage, we could theoretically compute the curvature at each of these label points in the figure, as they have done in [4]. The data point in this set with the maximum curvature is therefore the point we would want to use. We can estimate visually from the plot that the point with the maximum curvature in this instance occurs when $\lambda = 10^1$. The following figure is meant to showcase the different possible choices. As we can see in Figure 5.8, this does give a better, clearer result. On the other hand $\lambda = 10^0$ seems to do nothing, while $\lambda = 10^{-1}$ seems to make the result even more blurry.

Figure 5.8 Parameter selection for λ.
Top-left: Original blurred image Top-right: $\lambda = 10^1$
Bottom-left: $\lambda = 10^0$ Bottom-right: $\lambda = 10^{-1}$



If we look at another example image, we will see that there are not many significant differences in the outcome of the L-curve method, so the optimal parameter mostly to depend on the operator that is used. In both of these cases, we used the same blurring kernel, a 9x9 Gaussian matrix. We see that when λ is very small, it puts more weight on minimizing the total variation, thus making it more blurry. When λ is larger, then it lays greater weight upon minimizing the residual, but the contrast and border stability can suffer as a result. Ultimately, the L-curve method provides a good way to regularize the solution of this problem in a correct way.

Figure 5.9 Minimizing $\frac{\lambda}{2}\|Au - v\|_2^2 + R(u)$. Maximizing curvature
Top-left: Log-log plot Top-right: Original blurry image
Center-left: $\lambda = 10^{-1}$ Center-right: $\lambda = 10^{0}$
Bottom-left: $\lambda = 10^{1}$ Bottom-right: $\lambda = 10^{2}$



Chapter 6

Conclusion

To conclude, we have successfully developed some of the theory and applications behind the use of total variation in image processing. We have developed the algorithm and have proven a few theories about it. There are many ways to go about this particular type of problem, and this is just one of them. We applied the total variational algorithm to three completely different types of image processing problems, denoising, deblurring, and super-resolution.

With deblurring, we were able to sharpen both natural images as well as artificially blurred images. We defined the convolution of a blurring kernel in Chapter 2, and applied this to our numerical experiments in Chapter 5. With denoising, we never completely get rid of any noise, but we were able to smooth it by using the total variation. With super-resolution, we were able to up-sample a set of previously down-sampled images, but had much difficulty in actually applying the super-resolution algorithm to raw videos. An idea for future work on this is to try and compute the optical flow, and use this to adjust the images and carefully select the super-resolution grid. Though that is easier said than done, because there is a lot of guessing to that.

In Chapter 2, we defined and formulated several concepts including what we mean by images, blurring, noise, down-sampling, and up-sampling. In Chapter 3, we defined divergence, gradient, and total variation. In Chapter 4, we developed the saddle-point algorithm by using total variation, and also expanded on some additional theory for convex functions. In Chapter 5, we provided some examples of denoising, deblurring, and super-resolution. In addition to this we played with different parameter values. As we know, the value of a good algorithm depends heavily on the parameters used and we took great care to fit them properly. The figures in this thesis were implemented using code in the appendices.

Bibliography

[1] Image processing fundamentals. `http://www.mif.vu.lt/atpazinimas/dip/FIP/fip-Smoothin.html`. Accessed Online 24 May 2019.

[2] A. Chambolle. An algorithm for total variation minimization and applications. *Journal of Mathematical Imaging and Vision 20*, 20(3):89–97, Mar 2004.

[3] L. Rudin; S. J. Osher; E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60(1):259–268, Jan 1992.

[4] P.C. Hansen. The l-curve and its use in thenumerical treatment of inverse problems. *Technical University of Denmark*, (856876):1–21, 2005.

[5] A. Chambolle; T. Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *hal-00490826*, pages 1–48, Jun 2010.

[6] R. T. Rockafellar. *Convex Analysis*. Princeton Landmarks in Mathematics. Princeton, 1970.

[7] J. Nocedal; S. J. Wright. *Numerical Optimization, Second Edition*. Springer Series in Operations Research, 2006.

[8] Tony F. Chan; Michael K. Ng; Andy C. Yau; Andy M. Yip. Superresolution image reconstruction using fast inpainting algorithms. *Applied and Computational Harmonic Analysis*, 23:3–24, 2007.

Appendix A - Matlab

Primal-dual deblurring algorithm

```
function [P, MSE, R] = primal_dual_conv(U, B, N_steps,
    lambda, sigma, theta, tau)
%   Input: N_steps = number of steps
%       U:          Given MxN blurry image
%       B:          Blurring kernel
%       N_steps:    Number of steps in the iteration
%       lambda:     Regularization parameter
%       sigma:      Dual-step parameter
%       theta:      Update step parameter
%       tau:        Step-size parameter
%   Output:
%       P:          Smoothed image
%       MSE:        Mean-squared error
%       R:          Total variation

% Choose some default values
if (nargin < 2)
    s = 9;
    B = 1/(s^2) * ones(s);
end
if (nargin < 3)
    N_steps = 1000;
end
if (nargin < 4)
    lambda = 10000;
end
if (nargin < 5)
    sigma = 1;
end
if (nargin < 6)
    theta = 1;
end
if (nargin < 7)
    tau = 1/10;
end
```

```
35  % Get size of original image
36  M = size(U, 1);
37  N = size(U, 2);
38
39  % Operator that must be inverted in CG method
40  BB = conv2(B,B);
41  op = @(u) u + lambda*tau*reshape(imfilter(reshape(u,[M,N]),
        BB, 'conv', 'symmetric'),[M*N, 1]);
42
43  % Gradient operators
44  Dx = spdiags([ones(M-1, 1);0], 1, M, M) - spdiags([ones(M -
        1, 1); 0], 0, M, M);
45  Dy = spdiags([ones(N-1, 1);0], 1, N, N) - spdiags([ones(N -
        1, 1); 0], 0, N, N);
46
47  %Algorithm Initialization
48  P1 = zeros(M, N);    P2 = zeros(M, N);
49  x0 = U;
50  Ax0 = imfilter(x0, B,'conv','symmetric');
51  x_tild = x0;
52  C = []; N_vec = [];
53  Nprev = 1;
54  eps_update = 10^(-5);
55
56  %Iteration
57  for i=1:N_steps
58      % Gradient operation on x_tild
59      Ux = Dx * x_tild;
60      Uy = x_tild * Dy';
61
62      % Projected dual step
63      P1_t = P1 + sigma*Ux;
64      P2_t = P2 + sigma*Uy;
65      p_norm = sqrt(P1.^2 + P2.^2);
66      p_norm = max(p_norm, 1);
67      P1 = P1_t./p_norm;
68      P2 = P2_t./p_norm;
69      % Divergence of P
70      p_div = Dx'*P1 + P2*Dy;
71
72      % Update step on x, run pcg method
73      x_new = (x0 - (tau * p_div) + (lambda*tau * Ax0));
74      [x_new_rs, flag] = pcg(op,reshape(x_new,[M*N,1])
            ,10^(-12),40,[],[],reshape(x_new,[M*N,1]));
75      x_new = reshape(x_new_rs,[M,N]);
```

```
76
77      % Check size of update, stop the iteration if necessary
78      if(sum(sum(abs(x_new - x0))) < eps_update)
79          x0 = x_new;
80          N_steps = i;
81          break;
82      end
83
84      %Compute norm multipliers
85      Nnew = norm(x_new - x0, 2);
86      C = [C, Nnew/Nprev];
87      N_vec = [N_vec, Nnew];
88      Nprev = Nnew;
89
90      %Update step
91      x_tild = (1 + theta) * x_new - theta * x0;
92      x0 = x_new;
93  end
94
95  % Take consecutive products on C
96  CC = C(1:end-1) .* C(2:end);
97
98  % Plots of norm values and comparisons
99  figure; plot(3:N_steps, C(3:end), '-r');
100  title('Plot of C_k multiplier for each time step');
101  xlabel('k');
102  ylabel('C_k');
103
104  figure; plot(2:N_steps - 1, CC(2:end), '-r');
105  title('Plot of CC_k multiplier for each time step');
106  xlabel('k');
107  ylabel('CC_k');
108
109  figure; semilogy(3:N_steps, N_vec(3:end), '-r');
110  title('Plot of Cu_k multiplier for each time step');
111  xlabel('k');
112  ylabel('Cu_k');
113
114  % Make x0 displayable image
115  x0 = real(reshape(x0, [M,N]));
116  x0_min = min(x0(:))
117  x0_max = max(x0(:))
118  if (x0_max > x0_min)
119      x0 = 255*(x0-x0_min)/(x0_max-x0_min);
120  end
```

```
121
122 % Image plots
123 figure; imagesc(uint8(U)); title('Blurry Image', 'FontSize'
        , 8); colormap gray;
124
125 figure; imagesc(uint8(x0)); title(['Deconvoluted image, \
        sigma = ', num2str(sigma), ', \theta = ', num2str(theta)
        , ', \tau = ', num2str(tau), ', \lambda = ', num2str(
        lambda), ', n = ', num2str(N_steps)], 'FontSize', 8);
        colormap gray;
126
127 Ax0 = imfilter(x0, B,'conv','symmetric');
128 figure; imagesc(uint8(Ax0)); title(['Blurring of
        deconvoluted image, \sigma = ', num2str(sigma), ', \
        theta = ', num2str(theta), ', \tau = ', num2str(tau), ',
         \lambda = ', num2str(lambda), ', n = ', num2str(N_steps
        )], 'FontSize', 8); colormap gray;
129
130 figure; colormap jet; imagesc(uint8(abs(Ax0 - U))); title([
        'Difference between original blurry image and reblurred
        image'], 'FontSize', 8); colorbar;
131
132 % Send output
133 MSE = sum(sum((Ax0 - U).^2)) / numel(U)
134 R = sum(sum((Ux.^2 + Uy.^2).^(1/2)))
135 P = x0;
136
137 end
```

Up-sampling tensor array

```
function I_star = Istar(I_vals, F)
%   Input:
%       I_vals:     Nx2 matrix with N grid points
%       F:          Scaling factor
%   Output:
%       I_star:     3D array of tensor operands
    I_star = zeros(F, F, size(I_vals, 1));
    ind = 0;
    for ij = I_vals'
        ind = ind + 1;
        I_star(ij(1), ij(2), ind) = 1;
    end
end
```

Up-sampling operator

```
function Astaru = A_star(g, I_star)
%   Input:
%       g:          3D array of images
%       I_star:     3D array output by Istar.m
%       Both g and I_star must be 3D matrices
%       Each layer of I_star is a Kronecker tensor.
%   Output:
%       Astaru:     Resulting sum of these block products
    n_g1 = size(g, 1);
    n_g2 = size(g, 2);
    n_I1 = size(I_star, 1);
    n_I2 = size(I_star, 2);
    n_el = size(I_star, 3);

    Astaru = zeros(n_g1 * n_I1, n_g2 * n_I2);
    for i = 1:n_el
        Astaru = Astaru + kron(g(:,:,i), I_star(:,:,i));
    end
end
```

Down-sampling operator

```
function Au = A_op(u, F, I_vals)
%    Input:
%        u:          Original image
%        I_vals:     Nx2 matrix with N grid points
%        F:          Integer scaling factor
%    Output:
%        Au:         3D array of smaller images
    ind = 0;
    for ij = I_vals'
        ind = ind + 1;
        Au(:, :, ind) = u(ij(1):F:end, ij(2):F:end);
    end
end
```

Frobenius inner product

```
function C = fro(A, B)
%    Input:
%        A, B:   Two real-valued arrays of the same size
%    Output:
%        C:      Real value inner product of A and B
    dA = size(size(A), 2);
    dB = size(size(B), 2);
    if (da ~= db)
        error('Error, arrays are not compatible')
    end

    C = A .* B;
    for i = 1:dA
        C = sum(C);
    end
end
```

Super-resolution algorithm

```
function [P, MSE, R] = super_res(G, F_size, I_vals, I_star,
    N_steps, lambda, sigma, theta, tau)
%   Input:
%       G:          Given array of images, video, animation
%       F_size:     Scaling factor
%       I_vals:     Set of insertion points
%       I_star:     3D tensor array output of Istar.m
%       N_steps:    Number of iterations in the algorithm
%       lambda:     Regularization parameter
%       sigma:      Dual-step parameter
%       theta:      Parameter which determines update step
%       tau:        Step-size parameter
%
%   Output:
%       P:          Super-resolved image
%       MSE:        Mean-squared error
%       R:          Total variation

if (nargin < 2)
    F_size = 4;
end
if (nargin < 3)
    I_vals = [1, 1; 2, 2; 3, 3; 4, 4];
end
if (nargin < 4)
    I_star = Istar(I_vals);
end
if (nargin < 5)
    N_steps = 500;
end
if (nargin < 6)
    lambda = 10000;
end
if (nargin < 7)
    sigma = 1;
end
if (nargin < 8)
    theta = 1;
end
if (nargin < 9)
    tau = 1/20;
end


```

```
44  % Size of the animation
45  M = F_size * size(G, 1);
46  N = F_size * size(G, 2);
47  N_samples = size(G, 3);
48
49  % Gradient operators
50  Dx = spdiags([ones(M-1,1);0],1,M,M) - spdiags([ones(M-1,1);
        0],0,M,M);
51  Dy = spdiags([ones(N-1,1);0],1,N,N) - spdiags([ones(N-1,1);
        0],0,N,N);
52
53  % Operators for the algorithm
54  AA = @(u, F_size, I_vals, I_star) A_star(A_op(u, F_size,
        I_vals), I_star);
55  op_inv = @(u) u - (lambda*tau/(1 + lambda*tau))*AA(u,
        I_vals, I_star);
56
57  % Algorithm Initialization
58  Astarg = A_star(G, I_star);
59  U = Astarg;
60  U_vector = reshape(U', [], 1);
61  x0 = U;
62  x_tild = x0;
63  Ax0 = A_op(x0, F_size, I_vals);
64  eps_update = 10^(-5);
65  C = [];      N_vec = [];
66  Nprev = 1;
67
68  % Dual-variables
69  P1 = zeros(M, N);
70  P2 = zeros(M, N);
71
72  %Iteration
73  for i=1:N_steps
74      % Gradient operation on x_tild
75      Ux = Dx * x_tild;
76      Uy = x_tild * Dy';
77
78      % Projected dual step
79      P1_t = P1 + sigma*Ux;
80      P2_t = P2 + sigma*Uy;
81      p_norm = sqrt(P1.^2 + P2.^2);
82      p_norm = max(p_norm, 1);
83      P1 = P1_t./p_norm;
84      P2 = P2_t./p_norm;
```

```
85        % Compute divergence of P
86        p_div = Dx'*P1 + P2*Dy;
87
88        % Update step for x
89        x_new = op_inv(x0 - (tau*p_div) + (lambda*tau*Astarg));
90
91        % Check size of update, stop the iteration if necessary
92        if(sum(sum(abs(x_new - x0))) < eps_update)
93            x0 = x_new;
94            N_steps = i;
95            break;
96        end
97
98        % Update x_tild
99        x_tild = (1 + theta) * x_new - theta * x0;
100
101       % Compute multiplier terms for the figures
102       Nnew = norm(x_new - x0, 2);
103       C = [C, Nnew/Nprev];
104       N_vec = [N_vec, Nnew];
105       Nprev = Nnew;
106
107       % Final update
108       x0 = x_new;
109   end
110
111   % Take consecutive products on C
112   CC = C(1:end-1) .* C(2:end);
113
114   figure; plot(3:N_steps, C(3:end), '-r');
115    title('Plot of C_k multiplier for each time step');
116    xlabel('k');
117    ylabel('C_k');
118
119   figure; plot(2:N_steps - 1, CC(2:end), '-r');
120    title('Plot of CC_k multiplier for each time step');
121    xlabel('k');
122    ylabel('CC_k');
123
124
125   figure; semilogy(3:N_steps, N_vec(3:end), '-r');
126    title('Plot of difference norm for each time step');
127    xlabel('k');
128    ylabel('Cu_k');
129
```

```
130  % Make x0 displayable image
131  x0 = real(reshape(x0, [M,N]));
132  x0_min = min(x0(:))
133  x0_max = max(x0(:))
134  if (x0_max > x0_min)
135       x0 = 255*(x0 - x0_min)/(x0_max - x0_min);
136  end
137
138  figure; imagesc(uint8(x0)); title(['Super-resolved image,
         \theta = ', num2str(theta), ', \tau = ', num2str(tau), '
         , \lambda = ', num2str(lambda), ', n = ', num2str(
         N_steps)]);
139  colormap gray;
140
141  Ax0 = A_op(x0, F_size, I_vals);
142
143  % Send output
144  MSE = sum(sum(sum((Ax0 - G).^2))) / numel(G)
145  R = sum(sum((Ux.^2 + Uy.^2).^(1/2)))
146  P = x0;
147  end
```

Appendix B - Test Cases

Blurring test case - L-curve

```
%% L-curve

% Choose the dimensions of the image, and define the image
 M = 360;    N = 480;
 Norge = imresize(double(rgb2gray(imread('kabelvag.jpg'))),
     [M, N]);

% Blurring kernel (9x9 Gaussian)
 s = 5;
 B = 1/(s^2)*ones(s);
 BB = conv2(B, B);
 Norge = conv2(Norge, BB,'same');

% Initialize both terms of objective function
 MSE = [];
 R = [];

for j = -1:6
    [x0, s, r] = primal_dual_conv(Norge, BB, 300, 10^j, 1,
        1, 0.1);
    MSE = [MSE, s * M * N];
    R = [R, r];
end

figure; loglog(R, MSE, '-s');
title('Log-log plot:   \lambda = 10^j, j = -1, 0, ..., 6');
labels = {'\lambda = 10^{-1}', '\lambda = 10^0','\lambda =
    10^1','\lambda = 10^2','\lambda = 10^3','\lambda = 10^4'
    ,'\lambda = 10^5','\lambda = 10^6'};
text(R, MSE , labels, 'VerticalAlignment','bottom','
    HorizontalAlignment','right');
xlabel('Total variation R(u)');
ylabel('Sum of squared errors ||Au - v||_2^2');
```

Adjoint test case - Down-sampling

```
%% Adjoint Differences Check
% Test to show the operators A_op and A_star are adjoint

I_vals = [1, 2; 2, 1; 3, 3; 4, 4];
I_star = Istar(I_vals);

AU = A_op(U, I_vals);
AAU = AA(U, I_vals, I_star);

% Should show very small number close to zero
adjoint_difference_check = fro(AU, AU) - fro(U, AAU)
```

Super-resolution - Test case

```
% Choose the dimensions of the image, and initialize image
 M = 360;    N = 480;
 Island = imresize(double(rgb2gray(imread('iceland.jpg'))),
      [M, N]);

% Select a grid for down-sampling
 I_vals = [1, 1; 2, 2; 3, 3; 4, 4];
 I_star = Istar(I_vals);
 Island_samples = A_op(Island, I_vals);

 for (i = 1:size(Island_samples, 3))
     figure;
     imagesc(uint8(Island_samples(:, :, i)));
     title(['Sample Image number ', num2str(i)]);
     colormap gray;
 end

% Find the super-resolution given a simple insertion
 [x0, s, r] = super_res(Island_samples, I_vals, I_star,
     500, 10^j);
```

Video resolution - Test case

```
%% TEST CASE: Video/Image Super-resolution

% Initialize the GIF file in black/white
% Online image access: https://imgur.com/a/UsZ9ZVE
gif = im2double(imread('nidaros.gif', 'Frames', 1:4));

%From 4D array to 3D array
 ts = size(gif);
 gif = reshape(gif, ts(1), ts(2), ts(4));

%Edge-detection
 B = [-1, -1, -1; -1, 8, -1; -1, -1, -1];
 gif_edges = imfilter(gif, B, 'conv', 'symmetric');
 implay(gif_edges);

% Based on the edge-detection, choose appropriate I_vals
 F_size = 4;
 I_vals = [F_size, F_size];
 for i = 1:(F_size-1)
    I_vals = [I_vals; F_size - i, F_size - i];
 end
 I_star = Istar(I_vals, F_size);

% Run the super-resolution algorithm with the given
    parameters, and plot the figure.
 [x0, s, r] = super_res(gif, F_size, I_vals, I_star, 500,
     10);
```