Optimisation in computer vision: Inpainting

Group members: José Manuel López Camuñas, Marcos V. Conde, Alex Martin Martinez

Index

The problem

- Inpainting is the technique of modifying an image in an undetectable way, with the purpose of removing or reconstructing missing regions. In addition, no constraints are imposed on these regions' topology. [1]
- Removing objects from an image may be an easy task but it results in white pixels that makes it very obvious that the image has been modified.
- The information behind the object can not be recovered but with the information that remains in the image pixels can be generated to reproduce how the original image looked like.

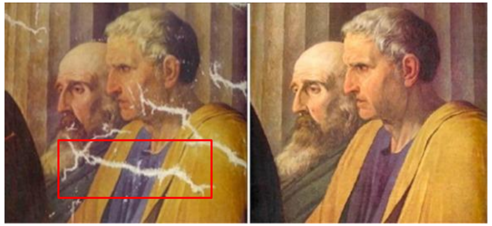
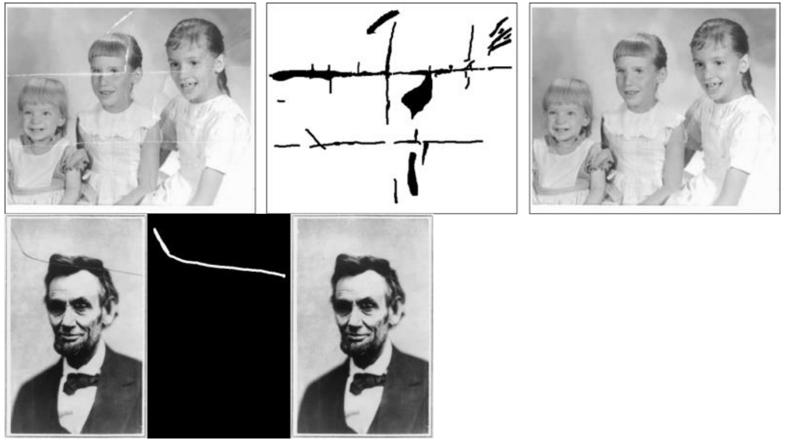


*Image Inpainting sample. We show the available original image **U** (left) and the post-processed image **V** (right). We also show one of the damaged region of interest (red bounding box) that we aim to recover.*

[1] Marcelo Bertalmio, Guillermo Sapiro, Vicent Caselles, and Coloma Ballester, ¨Image Inpainting”, SIGGRAPH 2000.

The problem

- Image Inpainting is a crucial task for Image Restoration and Image Retouching applications.



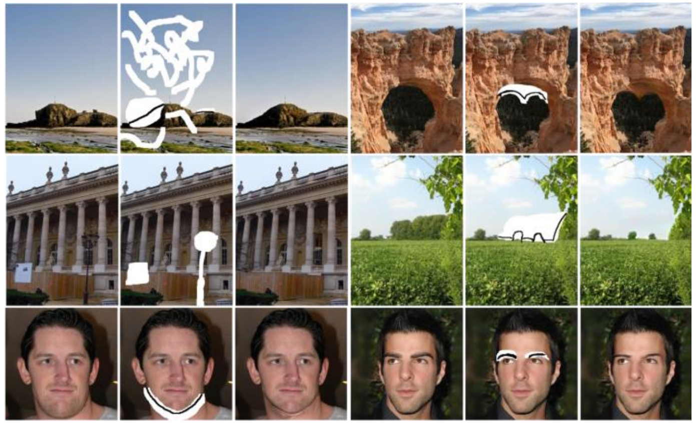
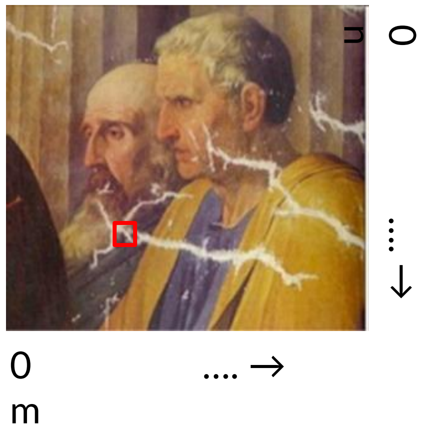


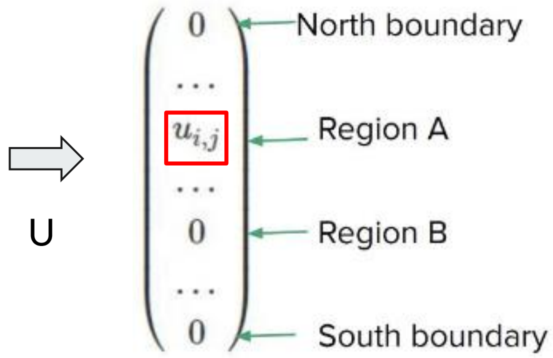
*Image Inpainting samples. We can see the original image **U** (left), the*

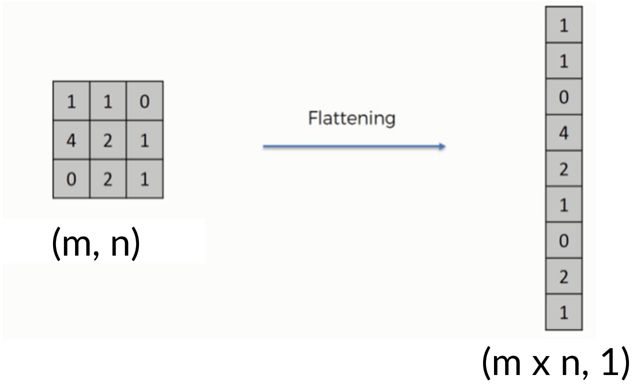
SOTA Image Inpainting Examples. Yu et.al. “Free-Form Image Inpainting with Gated Convolution”, ICCV 2019

The Solution

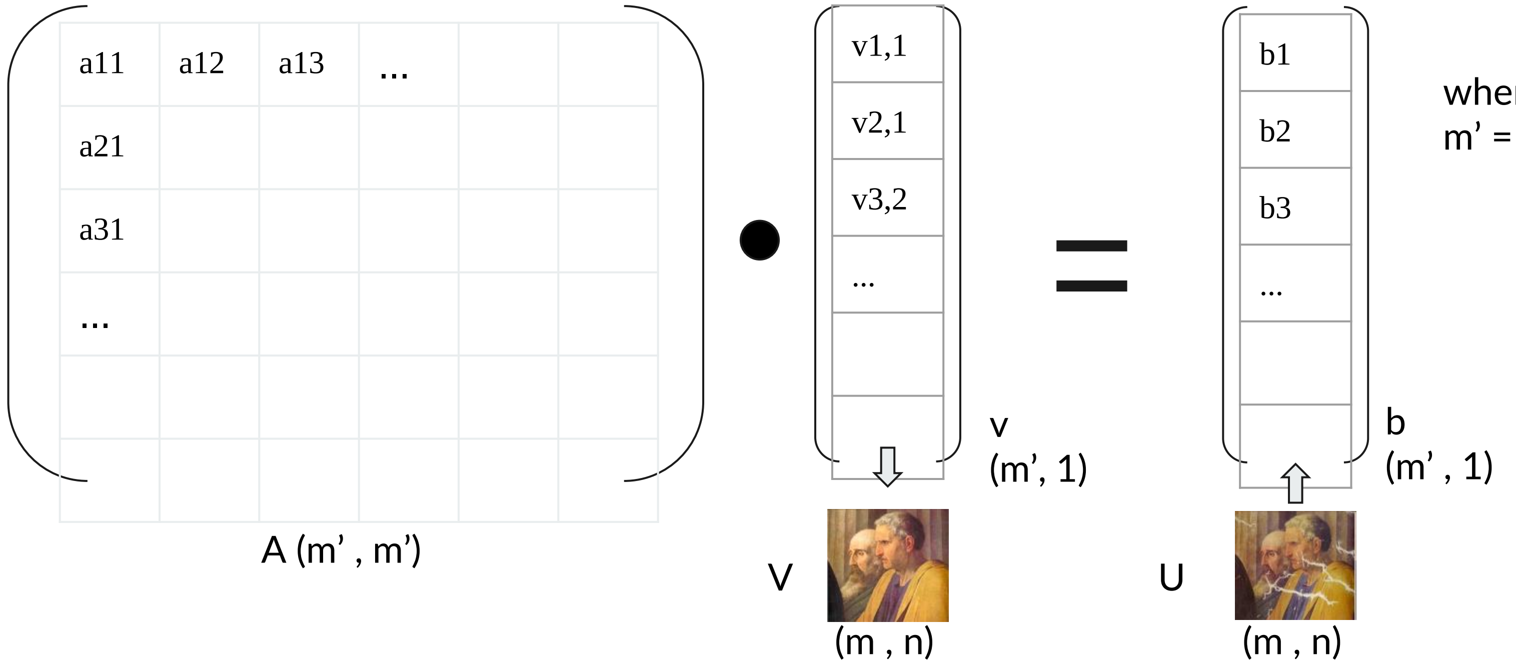
We define the available damaged image as U and the resultant unknown image V, which should be (**A**) similar to U, and (**B**) smoother. U has a dimension m x n (eg. 64 x 64 = 4096). However, we use ghost boundaries (padding), thus, we work with a dimension (m+2) x (n+2). The output unknown image V will have the same dimension as U. Both images as matrices are flattened into vectors b and v respectively (for instance [m+2,n+2] → [(m+2) x (n+2), 1]). A pixel U(i,j) is represented by a red box.







The Solution



where
m’ = (m+2) x (n+2)

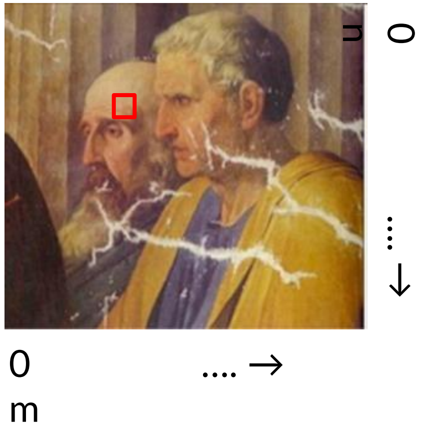
The Solution

Under only the **condition A**, the image V is the same as U, therefore, the matrix A in our system **Av=b** is the identity matrix. Each element (pixel, red box) V(i,j) = U(i,j). V has to be essentially as U in order to keep the general appearance and content semantic, so the changes are undetectable.

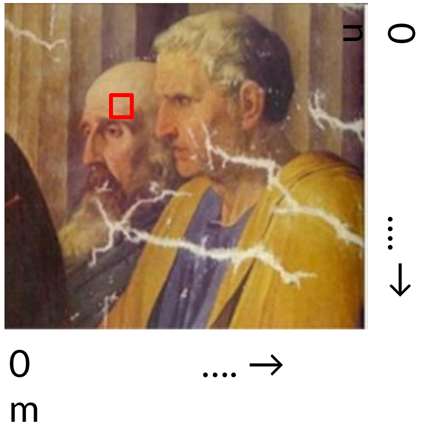
$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

A

X



=



The Solution

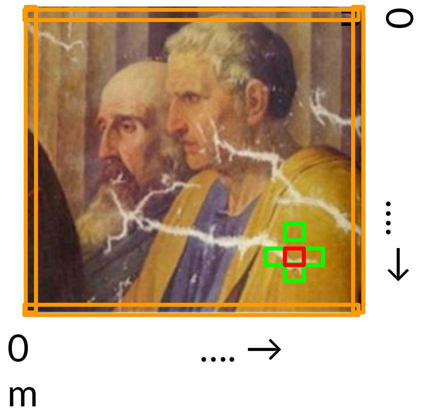
For the image to be coherent **and smooth** the following conditions have to be fulfilled:

A) In the positions i,j where U(i,j) the image was NOT damaged (region A) the value of V(i,j) at the position will remain the same
B) In the positions U(i,j) where the image was damaged (region B) the pixel (red) will be replaced with the mean of the 4 neighbouring pixels (green).

Mathematically this would translate to:

1. $V(i,j) = U(i,j)$ in region A
2. $4 \cdot V(i,j) - (V(i-1,j)+V(i+1,j)+V(i,j-1)+V(i,j+1)) = 0$ in region B

Note that for the pixels in the boundaries of the image (orange), we do not have the 4 neighbours, for this reason we add **ghost boundaries.**

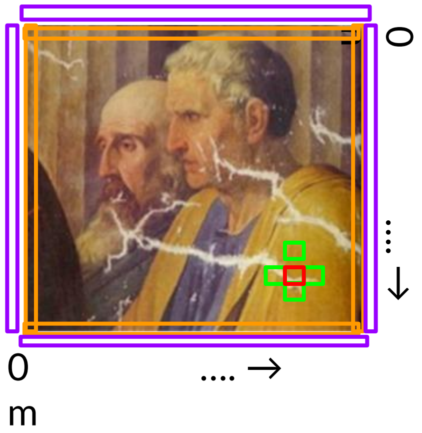


The Solution

Ghost boundaries also have to fulfil a condition, In this case the ghost boundaries will be equal to the neighbouring pixel. Being the image of (m+2,n+2) size it would translate to:

- V(1,j) = U(2,j) for any j (North Boundary)
- V(m+2,j) = U(m+1,j) for any j (South Boundary)
- V(i,m+2) = U(i,m+1) for any i (East Boundary)
- V(i,1) = U(i,1) for any i (West Boundary)

To accomplish these set of conditions the A matrix have to be filled with a certain value of numbers in each position. It we'll be exemplified which values has to have the A matrix for each condition separately in the i,j pixel. Combining all of this the matrix A can be constructed to solve the problem.



The Solution

1st: V(i,j)= U(i,j)   in region A

$$\begin{array}{c} \\ \\ \\ \text{i row} \rightarrow \\ \\ \\ \\ \end{array}\left(\begin{array}{cccccc} & & \text{j column} \downarrow & & & \\ 0 & 0 & 0 & \dots & & \\ 0 & \dots & & & & \\ 0 & & 1 & & & \\ \dots & & & & & \\ & & & & & \\ & & & & & \end{array}\right) \bullet \left(\begin{array}{c} v1,1 \\ v2,1 \\ \dots \\ v\ i,j \\ \\ \\ \end{array}\right) = \left(\begin{array}{c} 0 \\ 0 \\ \dots \\ u\ i,j \\ \\ \\ \end{array}\right)$$

The Solution

2nd: $4 \cdot V(i,j) - (V(i-1,j)+V(i+1,j)+V(i,j-1)+V(i,j+1)) = 0$ in region B

$$\begin{array}{r|ccccccccc|} & & j-(m+2)\text{column} & & & j\text{ column} & & & j+(m+2)\text{ column} & \\ \hline & 0 & & & 0 & 0 & \cdots & & & \\ & 0 & & & \cdots & & & & & \\ & 0 & & & & & & & & \\ i\text{ row} & \cdots & -1 & \cdots & -1 & 4 & -1 & \cdots & -1 & \cdots \\ & & & & & & & & & \\ & & & & & & & & & \\ \hline \end{array} \bullet \begin{pmatrix} v1,1 \\ v2,1 \\ \cdots \\ v\ i,j \\ \cdots \\ \ \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \cdots \\ 0 \\ \cdots \\ \ \end{pmatrix}$$

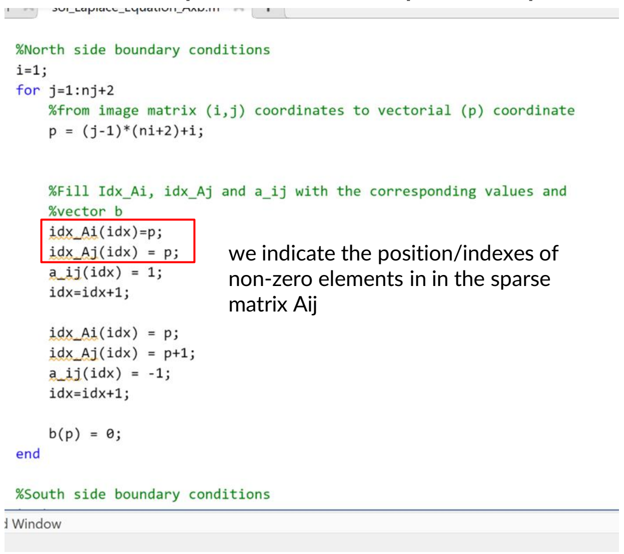
The Solution

Ghost Boundaries e.g North

$$\begin{array}{cc}
 & \begin{array}{cccccc} & & m+2\ \text{column} & & & \end{array} \\
\begin{array}{c} \\ \\ m+2\ \text{row} \\ \\ \\ \\ \end{array} & \begin{pmatrix} 1 & -1 & 0 & \dots & & \\ 0 & \dots & & & & \\ 0 & & 1 & -1 & & \\ \dots & & & \dots & & \\ & & & & & \\ & & & & & \end{pmatrix}
\end{array}
\bullet
\begin{pmatrix} v1,1 \\ v2,1 \\ \dots \\ v\ i,j \\ v\ i+1,j \\ \ \end{pmatrix}
=
\begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ \ \end{pmatrix}$$

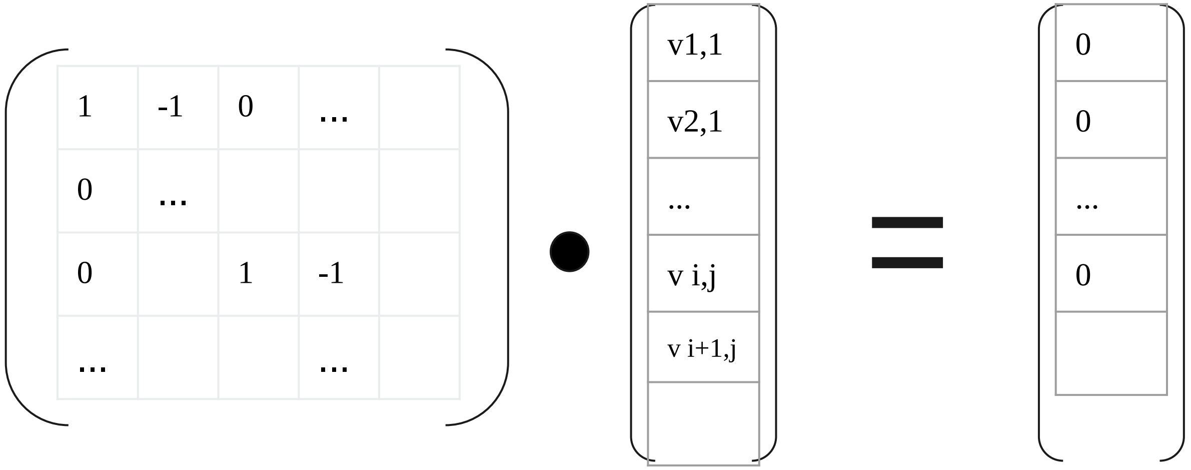
Implementation Details

We provide the important parts of the algorithm implemented using Matlab.



```
%North side boundary conditions
i=1;
for j=1:nj+2
    %from image matrix (i,j) coordinates to vectorial (p) coordinate
    p = (j-1)*(ni+2)+i;


    %Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
    %vector b
    idx_Ai(idx)=p;
    idx_Aj(idx) = p;
    a_ij(idx) = 1;
    idx=idx+1;

    idx_Ai(idx) = p;
    idx_Aj(idx) = p+1;
    a_ij(idx) = -1;
    idx=idx+1;

    b(p) = 0;
end

%South side boundary conditions
```

we indicate the position/indexes of non-zero elements in in the sparse matrix Aij



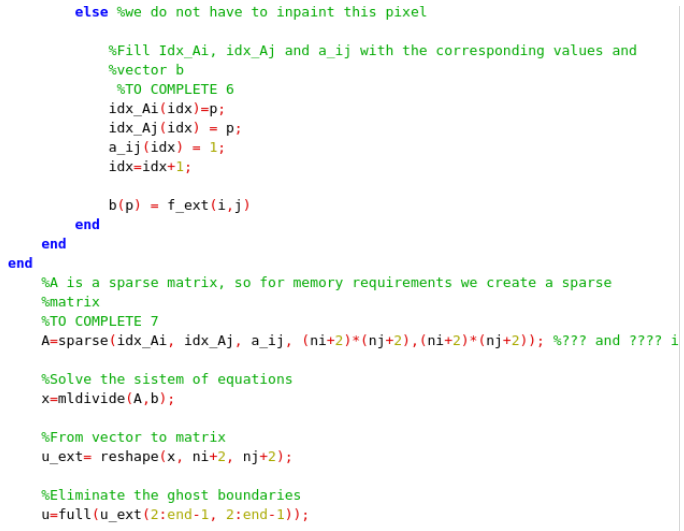
$$\begin{pmatrix} 1 & -1 & 0 & \dots & \\ 0 & \dots & & & \\ 0 & & 1 & -1 & \\ \dots & & & \dots & \end{pmatrix} \bullet \begin{pmatrix} v1,1 \\ v2,1 \\ \dots \\ v\ i,j \\ v\ i+1,j \\ \ \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ \ \end{pmatrix}$$

Implementation Details

```
%Inner points
for j=2:nj+1
    for i=2:ni+1

        %from image matrix (i,j) coordinates to vectorial (p) coordinate
        p = (j-1)*(ni+2)+i;

        if (dom2Inp_ext(i,j)==1) %If we have to inpaint this pixel

            %Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
            %vector b
            %TO COMPLETE 5
            idx_Ai(idx)=p;
            idx_Aj(idx) = p;
            a_ij(idx) = 4;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p-(ni+2);
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p+ni+2;
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p-1;
            a_ij(idx) = -1;
            idx=idx+1;

            idx_Ai(idx) = p;
            idx_Aj(idx) = p+1;
            a_ij(idx) = -1;
            idx=idx+1;

            b(p) = 0;
```

$$\begin{pmatrix} 0 & 0 & 0 & \dots & \\ 0 & \dots & & & \\ -1 & -1 & 4 & -1 & \dots \\ \dots & & & \dots & \end{pmatrix} \bullet \begin{pmatrix} v1,1 \\ v2,1 \\ \dots \\ v\ i,j \\ v\ i+1,j \\ \ \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ 0 \\ \ \end{pmatrix}$$

Implementation Details

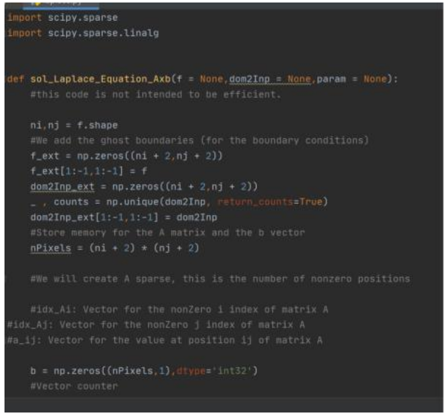


```
        else %we do not have to inpaint this pixel

            %Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
            %vector b
             %TO COMPLETE 6
            idx_Ai(idx)=p;
            idx_Aj(idx) = p;
            a_ij(idx) = 1;
            idx=idx+1;

            b(p) = f_ext(i,j)
        end
    end
end
    %A is a sparse matrix, so for memory requirements we create a sparse
    %matrix
    %TO COMPLETE 7
    A=sparse(idx_Ai, idx_Aj, a_ij, (ni+2)*(nj+2),(ni+2)*(nj+2)); %??? and ???? i

    %Solve the sistem of equations
    x=mldivide(A,b);

    %From vector to matrix
    u_ext= reshape(x, ni+2, nj+2);

    %Eliminate the ghost boundaries
    u=full(u_ext(2:end-1, 2:end-1));
```

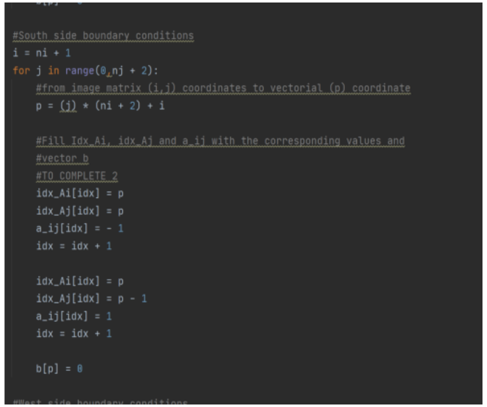
$$\begin{pmatrix} 0 & 0 & 0 & \dots & \\ 0 & \dots & & & \\ 0 & 0 & 1 & \dots & \\ \dots & & & \dots & \end{pmatrix} \bullet \begin{pmatrix} v1,1 \\ v2,1 \\ \dots \\ v\ i,j \\ v\ i+1,j \\ \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dots \\ u\ i,j \\ \end{pmatrix}$$

Implementation Details

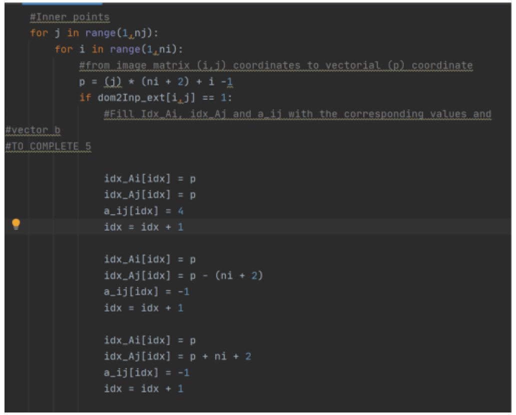
We also tried to adapt the algorithm to python.



```python
import scipy.sparse
import scipy.sparse.linalg


def sol_Laplace_Equation_Axb(f = None,dom2Inp = None,param = None):
    #this code is not intended to be efficient.

    ni,nj = f.shape
    #We add the ghost boundaries (for the boundary conditions)
    f_ext = np.zeros((ni + 2,nj + 2))
    f_ext[1:-1,1:-1] = f
    dom2Inp_ext = np.zeros((ni + 2,nj + 2))
    _ , counts = np.unique(dom2Inp, return_counts=True)
    dom2Inp_ext[1:-1,1:-1] = dom2Inp
    #Store memory for the A matrix and the b vector
    nPixels = (ni + 2) * (nj + 2)

    #We will create A sparse, this is the number of nonzero positions

    #idx_Ai: Vector for the nonZero i index of matrix A
#idx_Aj: Vector for the nonZero j index of matrix A
#a_ij: Vector for the value at position ij of matrix A

    b = np.zeros((nPixels,1),dtype='int32')
    #Vector counter
```



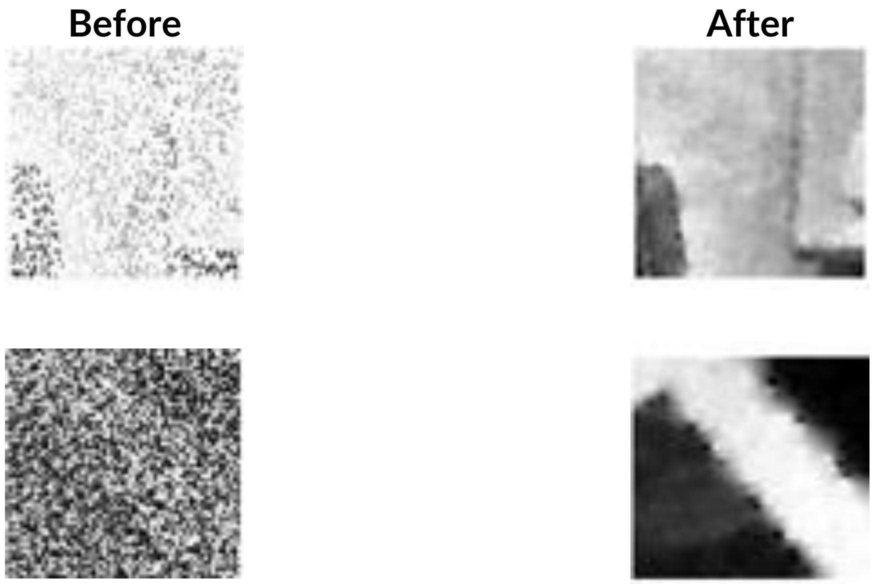
```python
#South side boundary conditions
i = ni + 1
for j in range(0,nj + 2):
    #from image matrix (i,j) coordinates to vectorial (p) coordinate
    p = (j) * (ni + 2) + i

    #Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
    #vector b
    #TO COMPLETE 2
    idx_Ai[idx] = p
    idx_Aj[idx] = p
    a_ij[idx] = - 1
    idx = idx + 1

    idx_Ai[idx] = p
    idx_Aj[idx] = p - 1
    a_ij[idx] = 1
    idx = idx + 1

    b[p] = 0

#West side boundary conditions
```



```python
    #Inner points
    for j in range(1,nj):
        for i in range(1,ni):
            #from image matrix (i,j) coordinates to vectorial (p) coordinate
            p = (j) * (ni + 2) + i -1
            if dom2Inp_ext[i,j] == 1:
                #Fill Idx_Ai, idx_Aj and a_ij with the corresponding values and
#vector b
#TO COMPLETE 5

                idx_Ai[idx] = p
                idx_Aj[idx] = p
                a_ij[idx] = 4
                idx = idx + 1

                idx_Ai[idx] = p
                idx_Aj[idx] = p - (ni + 2)
                a_ij[idx] = -1
                idx = idx + 1

                idx_Ai[idx] = p
                idx_Aj[idx] = p + ni + 2
                a_ij[idx] = -1
                idx = idx + 1
```

But it didn’t worked because of an error with the function that solves A·v=b.

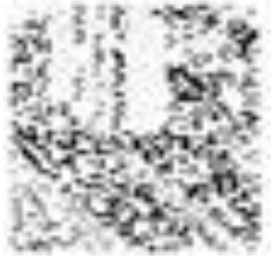
Results

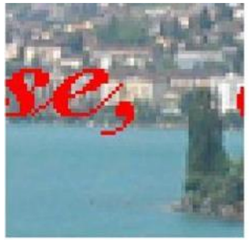


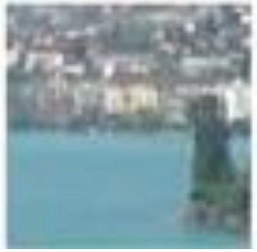
Results

Before

After

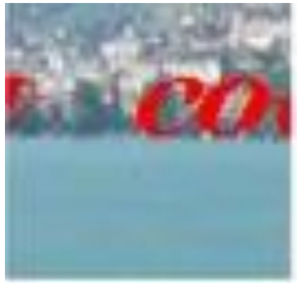


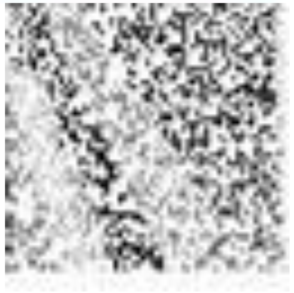




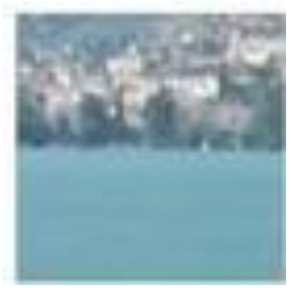
Results

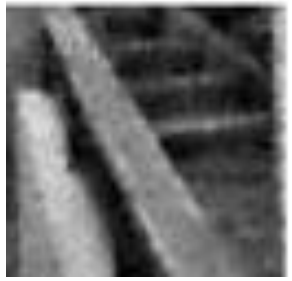
Before



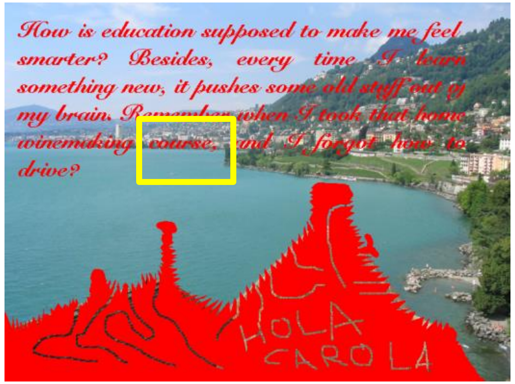


After

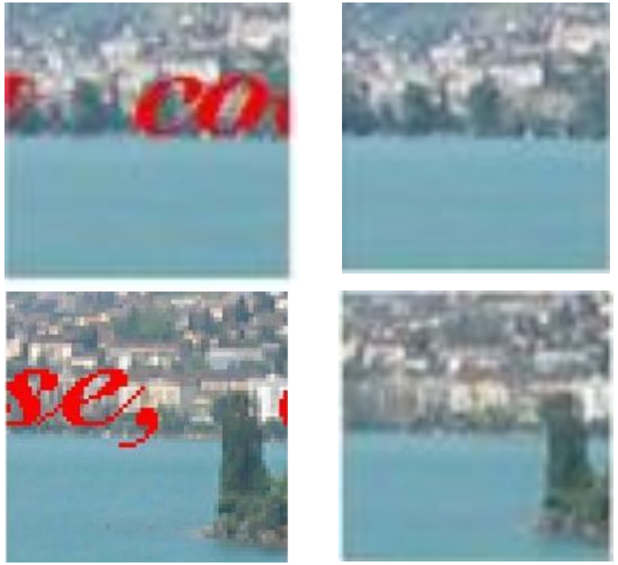




Results


How is education supposed to make me feel smarter? Besides, every time I learn something new, it pushes some old stuff out of my brain. Remember when I took that home winemaking course, and I forgot how to drive?
HOLA
CAROLA

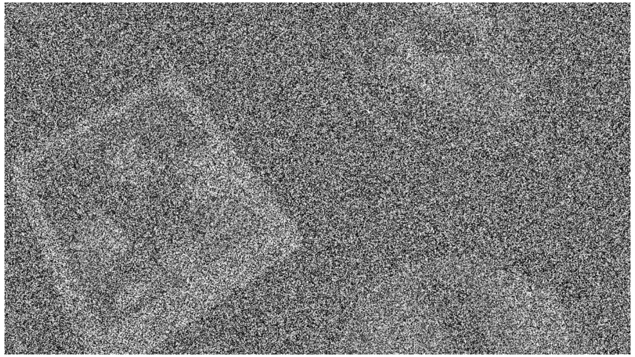
Before **After**

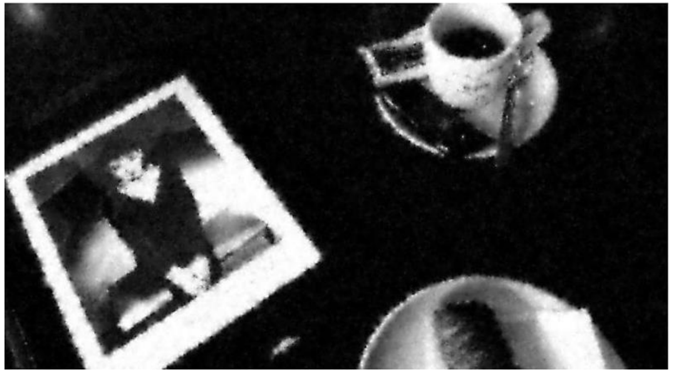


Due to the algorithm inefficiency we could not manage to process full-resolution RGB images, therefore we processed 64x64x3 crops

Results

Last minute we have seen what was the problem with the script and why it was going so slow. We could manage to create one full grayscale image from the examples.





Discussion and Conclusions

Solving the Inpainting problem via Optimisation presents the following advantages (+) and disadvantages (-) in comparison to Deep Learning methods:

- + No ground-truth is required, we only need a damaged and noisy image U as baseline.
- + No training is required, therefore, the data acquisition cost is notably smaller.
- + The method is easily interpretable and improvable.
- - Depending on the algorithm, the inference time is not competitive (i.e. using a normal computer the algorithm implemented in Matlab ran for 2 minutes, using 64x64 images).
- - We need to process each RGB channel separately.
- - In both cases, we need the (human-annotated) masks of the regions of interest.