PDEs for Fixing Bad Pictures

Prof. Anil Kokaram : Assignment

1 Mar 2023

This assignment is worth 7% for your final mark for this module. You will experiment with using a PDE to solve one of the oldest problems in cinema post-production. Please read this document COMPLETELY before starting your assignment. There are useful hints at the end. Please submit your final Matlab script with the name `<lastname>_<firstname>_<studentid>.m`.

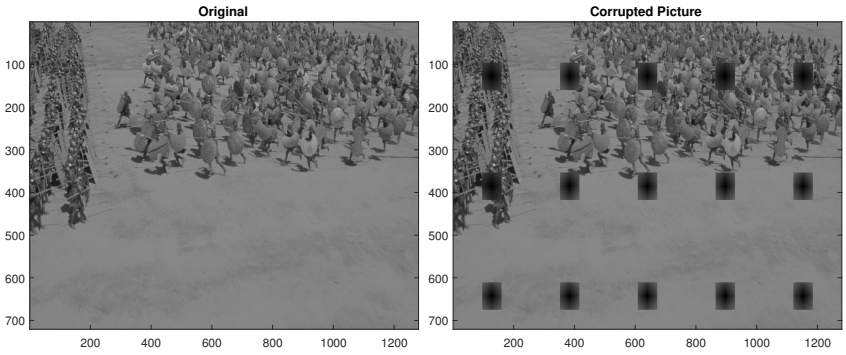

Figure 1: Original Frame (left), Corrupted Frame (right). These are stored in `greece.tif` and `badpicture.mat` respectively.

When film is stored for many years it degrades .. physically. When we convert these pictures to a digital file we often find dirt stuck to the film or bits just plain missing. Film is still used by certain directors today[1]. Digital cameras also have this problem when sensors go *bad*. In this modern time the problem of missing data for digital material is not as big of a problem, but when processing archived material, it still is an issue.[2]

Figure 1 shows a version of this archive film problem[3]. The frame on the left (`greece.tif`) has been corrupted over time so that blocks are missing, and the frame we have now is shown on the right (stored in `badpicture.mat`). Let us call these pictures O and G. The right way to solve this problem is to treat it like a video processing problem, but you are going to treat this like a 2D problem. The task is to design an algorithm that can fill-in the holes in the corrupted frame G so that it looks like the original O. You are not allowed to cheat and use the original of course.

The task of infilling holes has come to be known as *Inpainting*[4], although many people were working on the problem many

[1] Christopher Nolan used film for Interstellar for some shots at least.

[2] It might interest you to know that the master copies of famous movies like Goldfinger or Jungle Book are still on film. When these are remastered, the film has to be taken out of the vaults and rescanned at higher and higher resolutions every ten years or so as better and better TV sets get released.

[3] It turns out that this is test footage from the set of a summer blockbuster in 2004 that www.sigmedia.tv worked on.

[4] All cinema compositing software now has some kind of tool for inpainting built in. This includes Adobe Photoshop.

years before it had a name. One of the simplest ideas for hole filling is to assume that all images obey a partial differential equation in local areas. You will use this idea now.

Consider that the image $I(x, y)$ follows the PDE as follows.

$$\frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} = f(x, y) \tag{1}$$

Here $I(x, y)$ is the grey scale value of an image pixel at site (x, y). The idea is to solve for $I(x, y)$ wherever the pixels are missing in the corrupted image. You will find an array `badpixels.tif` which is set to 1 where a pixel is corrupted in G.

We can use a Finite Difference Method to generate a numerical solution to this problem. Using the usual 2nd order finite difference approximation we have the expression as follows.

$$\frac{I[m-1,n] - 2I[m,n] + I[m+1,n]}{h^2} + \frac{I[m,n-1] - 2I[m,n] + I[m,n+1]}{h^2} = f[m,n] \tag{2}$$

here $I[m, n]$ is the pixel value at site (m, n) (row, column) in the picture, and $f[m, n]$ is the same thing for the forcing function. In image processing we typically assume $h = 1$. This means we can write instead

$$I[m-1,n] + I[m+1,n] + I[m,n-1] + I[m,n+1] - 4I[m,n] - f[m,n] = 0 \tag{3}$$

One simple iterative solution therefore is just to update the new value of the corrupted image in-place using the existing values of the rest of the pixels as follows.

$$I^{k+1}[m,n] = (I^k[m-1,n] + I^k[m+1,n] + I^k[m,n-1] + I^k[m,n+1] - f[m,n])/4 \tag{4}$$

Here I^k is the value of the image solution at the kth iteration. But this performs rather poorly wrt convergence.

A much more efficient scheme (but still slow) is called Successive Overrelaxation (SOR). In this scheme we measure the error between the PDE equation and the actual value of the

pixel at the current site, then generate an update which is some fraction of that error.

$$E^k[m,n] = I^k[m-1,n] + I^k[m+1,n] + \\ I^k[m,n-1] + I^k[m,n+1] - 4I^k[m,n] - f[m,n] \quad (5)$$

$$I^{k+1}[m,n] = I^k[m,n] + \alpha\frac{E^k[m,n]}{4}$$

In this SOR scheme you can choose to update the kth iteration *in place* or *synchronously*. For *in place* updating, you simply replace every restored pixel with your new estimate as you iterate from site to site. In *synchronous* updating you update all the sites in the $k+1$th restored image using the data only from the k th iteration. In this assignment you must use *synchronous* updating. It turns out that synchronous updating does not work as well as in-place updating but we want to see how well it works anyway.

Unlike your previous Matlab labs and assignments, this algorithm needs to be implemented site-wise. Therefore you must iterate over every missing pixel updating as you go. This means having to access a 2D array of pixels one at a time. You need to use two loops to do this, one across columns, and one across rows.

Your overall task is to write a Matlab program which applies the SOR scheme to the pixels in the missing patches and so generate a *restored* image. You will have to use a few thousand iterations of SOR to get a decent result. A value of $\alpha = 0.8$ is a good guess but you might find a better value. Generate two restored versions, using $\alpha = 0.8$ as follows.

1. Using $f[m,n] = 0$
2. Using $f[m,n]$ as read from an array `f` stored in `forcing.mat`.

Follow the instructions below.

1. Use the variable `total_iterations` to store the total iterations you use. This should be at least 2500. The number should be the same for each restoration above. Use `alpha` to store the relevant constant you use.
2. For testing purposes store the output of your restoration after 20 iterations in variables `restored20` and `restored20_2`

corresponding to without and with the forcing function respectively. This can be done in your iteration loop just by using an `if` statement which does this assignment of variables at 20 iterations and then continues on with iterations.

3. Display the original image `original.tif` and corrupted image `badpicture.mat` in Figures 1 and 2 respectively. Display the restored images in Figures 3 and 4 respectively. Title all your images as shown. Your restored images should look something as shown below. Your final restored images (after all your iterations) should be stored in variables `restored` and `restored2` respectively.

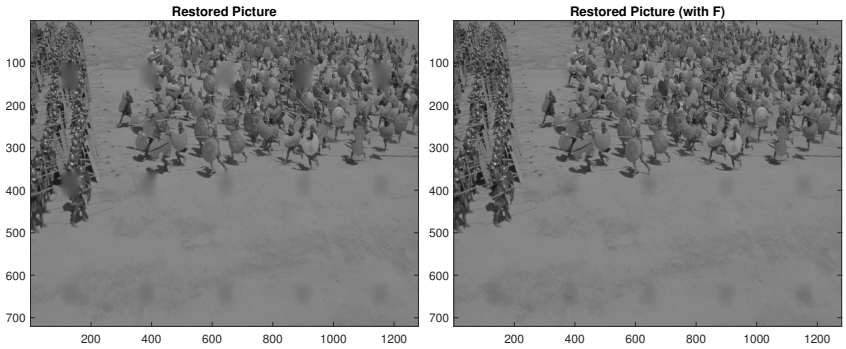

Figure 2: Restored Frames $F = 0$ (left) and F from a file (right).

4. At the end of each iteration calculate the standard deviation of the error between the original image and the restored version *in the missing patches only*. Assign the error vectors to the variables `err` and `err2` corresponding to without and with the forcing function respectively. Display your plot of this error measure versus iteration in Figure 5. The plot should look something like that below. Label your axes and use legends to label your curves and a linewidth of 3.

Notes

The data you will need is in the files as follows.

```
greece.tif      This contains the original uncorrupted picture as a TIFF file.
badpicture.mat  A matlab data file containing the corrupted image "badpic"
badpixels.tif   A TIFF image indicating the pixel sites in badpicture.mat
                that are corrupted. Each pixel is either 1 (corrupted site)
                or 0 (not corrupted).
```

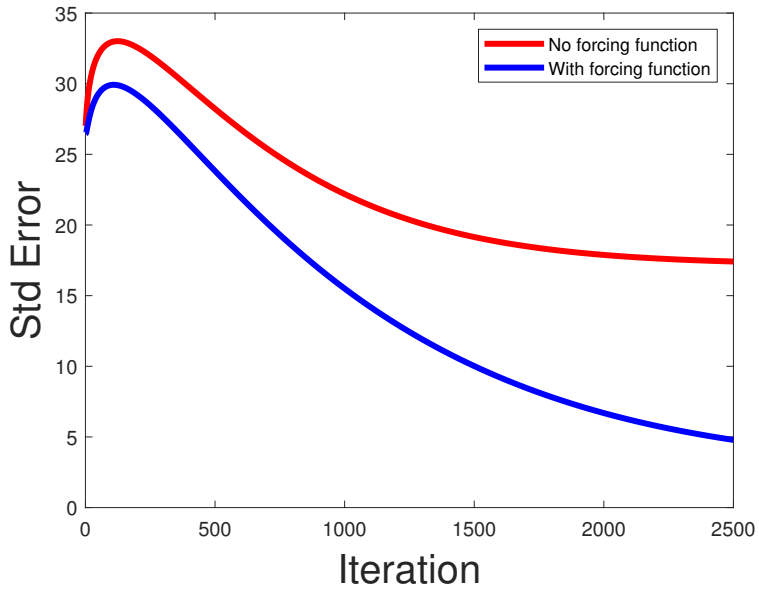

Figure 3: Convergence. Note that your curves may not look like this because there are a couple of different variants on the way you can perform the iterations.

`forcing.mat` Containing "f" the forcing function

To load a greyscale 'tif' image in Matlab and display it you can use the lines below.

```
pic = imread('fname.tif');
figure(1);
image(pic);
colormap(gray(256));
```

In Matlab if you want to get the pixel value at site $[m, n]$ in an array A, say, you can use just $A(m, n)$.

Some of the files you will use are stored as .mat files. Use `load filename.mat` to load up the data in those files in Matlab. Try `help load` to see what that does.

The file `badpixels.mat` contains a mask which is 1 where there is a bad pixel and 0 otherwise. You will need to iterate over each bad pixel site in the corrupted picture `corrupted.tif`. To find the bad pixel sites automatically you can use the `find` function in Matlab. You could also try to find each missing block manually but that is tedious and you'll probably get it wrong.

`[j,i]=find(mask==0)` finds all the sites in the 2D array `mask` which satisfy the condition that the pixel value at that site is 0, and returns the row, column index in the vectors `j`

and `i`.

`[p]=find(mask==0)` finds all the sites in the 2D array `mask` which satisfy the condition that the pixel value at that site is 0, and returns the vector index in the vector `p`. You can think of `p` as the pixel site locations indexed in raster scan order. So the element $(10, 6)$ for instance, in a 2D array would have a 1-D index $(10-1) \times H + 6$ into the 2D array. If the pixel value at $(10, 6)$ is 0 in this case then the corresponding value entered in `p` $= (10-1) \times H + 6$

To see how to use `find` you should try it yourself on a matrix which you make up for testing.

Final Comments

Using PDEs alone to infill pictures isn't that successful. You will have seen that its relatively easy to fill in the holes where they are not surrounded by much texture, but textured areas are tricky. It turns out that things like stochastic image models and patch based or Bayesian methods work quite alot better. Recently there has been interesting success using direct Machine Learning.

If on the other hand you can find some way to access that function $f(x, y)$ then you're on to winner. As you will have seen, with $f() = 0$ the PDE can't really infill any detail in the hole. However, with the right $f()$, the PDE is able to infill detail better. It turns out that using synchronous updating in this case is not a good idea. Some recent ideas in image editing allow users to play with mixing $f()$ functions (or details) with different pictures. The results are quite stunning when applied correctly. See the work of Patrick Perez in *Poisson Image Editing* and Francois Pitie in *Colour Transfer*.