Lab 3: Poisson Editing

Quim Llimona (146662)

UPF

In this lab, we will implement the Poisson editing, an image edition tool that merges images based on a binary mask in a smooth way. As it will be shown later, this can be done by means of minimizing a quadratic energy. Let's see how to do this first.

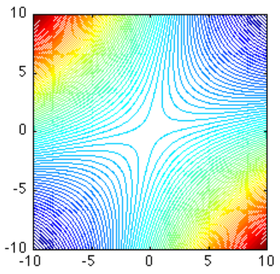
1 Quadratic functions

A generic expression of a quadratic function is as follows:

$$f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + c$$

Where the 1/2 factor is added for convenience and A is square, symmetric and positive definite. Let's recall what definiteness is with some example symmetric matrices, together with a representation of the most basic quadratic function that can be defined with them: $f(x) = \langle x, Ax\rangle$.

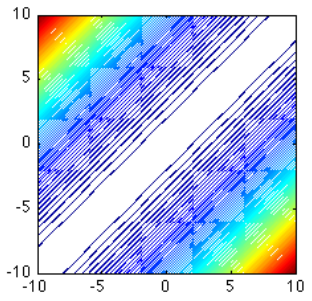
$$A_1 = \begin{bmatrix} 1 & -3 \\ -3 & 1 \end{bmatrix}$$



The eigenvalues of this matrix are $(-2, 4)$. Since some of them are lower than 0 and some are higher, the matrix is indefinite.

As it can be seen on the plot, the function hasn't got any minima; it goes to $-\infty$ when $|x+y| \to \infty$.

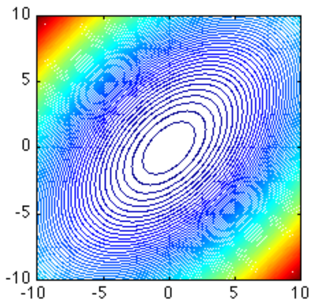
$$A_2 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$



The eigenvalues of this matrix are $(0, 2)$. Since they are non-negative, the matrix is semidefinite positive.

As it can be seen on the plot, the function has got infinite minima. All points on the line $x = y$ are absolute minima.

$$A_3 = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$



The eigenvalues of this matrix are $(0.5, 1.5)$. Since they are strictly positive, the matrix is definite positive.

As it can be seen on the plot, the function has got a single minimum: the point $(0, 0)$.

In general, the above function satistifes that if λ, v is an eigenvalue, eigenvector pair of the matrix A, then:

$$f(v) = \langle v, Av \rangle = \langle v, \lambda v \rangle = \lambda \|v\|^2$$

Notice that the eigenvectors don't specify a single point, but a direction. Thus, all vectors parallel to an eigenvector satisfy the equation above. This means that if an eigenvalue is 0 all the function on all points on its line are 0. If an eigenvalue is negative, all points on its line go towards $-\infty$. If an eigenvalue is positive, all points on its line go towards $+\infty$. This is what happened with the example matrices. Since the eigenvectors are a basis of the function domain, it's possible to get an idea of the shape of the contours from only the eigenvalues and eigenvectors.

1.1 Minimization of quadratic functions

It's easy to show that minimizing the above generic quadratic function is equivalent to minimizing this simpler one:

$$f(x) = \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$$

Its gradient can be found using the identity $D_v f(x) = \langle \nabla f(x), v\rangle$:

$$D_v f(x) = \lim_{h\to 0} \frac{f(x+hv) - f(x)}{h}$$

Let's develop on the upper term, $f(x+hv) - f(x)$.

$$\begin{aligned}
f(x+hv) - f(x) &= 0.5\langle x+hv, A(x+hv)\rangle - \langle b, x+hv\rangle - 0.5\langle x, Ax\rangle + \langle b, x\rangle \\
&= 0.5\left(\langle x, Ax\rangle + \langle hv, Ax\rangle + \langle x, Ahv\rangle + \langle hv, Ahv\rangle\right) \\
&\quad - \langle b, x\rangle - \langle b, hv\rangle - 0.5\langle x, Ax\rangle + \langle b, x\rangle \\
&= 0.5\left(h\langle v, Ax\rangle + h\langle x, Av\rangle + h^2\langle v, Av\rangle\right) - h\langle b, v\rangle \\
&= h\langle Ax, v\rangle - h\langle b, v\rangle + 0.5h^2\langle v, Av\rangle
\end{aligned}$$

Now we can switch to the limit:

$$\begin{aligned}
\lim_{h\to 0} \frac{f(x+hv) - f(x)}{h} &= \lim_{h\to 0} \frac{h\langle Ax, v\rangle - h\langle b, v\rangle + 0.5h^2\langle v, Av\rangle}{h} \\
&= \lim_{h\to 0} \langle Ax, v\rangle - \langle b, v\rangle + 0.5h\langle v, Av\rangle \\
&= \langle Ax, v\rangle - \langle b, v\rangle \\
&= \langle Ax - b, v\rangle
\end{aligned}$$

Therefore, $\nabla f(x) = Ax - b$.

Since we want to minimize it without resitrictions, we can simply set its gradient to 0 and solve for x. This means solving a linear system. It could be solved directly, but for large matrices iterative methods are more effective. Let's see some of them.

1.2 The Gradient Descent method

We already saw the gradient descent in a previous assignment. This time we will use it with an optimal step size; that is, for each iteration the step size is such that the resulting point is the minimum across the search line defined by the gradient. Let's compute this step.

The step can be found by minimizing the energy across the search line. To this, we will make a variable change such that we get this restriction; we will minimize the function $h(t) = f(x_k - tv)$ on the t variable. There are no external bounds, so let's compute its gradient in order to set it to 0 using the chain rule.

$$\begin{aligned}\frac{d}{dt} f(x_k - tv) &= \langle -v, \nabla f(x_k - tv) \rangle \\ &= \langle -v, A(x_k - tv) - b \rangle \\ &= t\langle v, Av \rangle - \langle v, Ax_k - b \rangle\end{aligned}$$

$$\begin{aligned}t_k\langle v, Av \rangle - \langle v, Ax_k - b \rangle = 0 \\ t_k\langle v, Av \rangle = \langle v, Ax_k - b \rangle \\ t_k = \frac{\langle v, Ax_k - b \rangle}{\langle v, Av \rangle}\end{aligned}$$

The convergence of this method, as I checked empirically, is exponential; when plotting a logarithmic plot, the line decreases linearly. However, when the minimum is close it gets slower.

Notice that if A is the identity matrix, the method converges to the exact solution in a single step. This happens because the matrix is not making any transformation to the space, so the function is equivalent to minimizing the distance to a point. Therefore, the gradient is always pointing to the solution.

If more steps are needed, it happens that the angle between consecutive search directions is 90^o. That's because the step is optimal, which implies that at each iteration the best point is that where the gradient of the function is orthogonal to the search direction. Therefore, the next search direction, made of the gradient of the function, will be orthogonal to the previous one. This may seem clever, but it usually makes the algorithm follow a zig-zag pattern and converge slowly in cases where the gradient of the function differs a lot from the direction one should follow to find the minimum.

1.3 The Conjugate Gradient method

The Conjugate Gradient method overcomes the issue presented above by defining the orthogonality of the search directions in a way that takes into account how the A matrix transforms the space. It works in the same way, by choosing an optimized step that finds the minimum across a search direction iteratively. However, the search direction is not the gradient of the function (which is orthogonal to the previous one) but only the components of the gradient that haven't been optimized yet. This generates a sequence of search directions that are conjugate with respect to the matrix A, that is, $\langle v_1, Av_2 \rangle = 0$.

As a result, the method (in theory) converges in at most n steps, being n the number of dimenisions of the search space, because it's the number of vectors needed to span it. In the case of two-dimensional functions, the algorithm converges in only 2 steps; in the first one, the function is minimized across the direction of the gradient in the initial posiction. In the second one, the function is minimized across the direction of the conjugate of the previous gradient. This new direction is independent from the first one, and the two together span the plane, so the algorithm stops there.

With fixed-point arithmetic often more steps are required because of the lack of precision when computing the conjugate directions. However, the test I run with two-dimensional functions didn't show this issue.

2 Poisson Editing

In photo editing, it's common to make patches; that is, to fix a part of an image by replacing it with a part from another image. If this is done directly (i.e. by specifying which pixels come from which image), the result is not very good because the edges will be obvious and there might be changes on the illumination of the involved objects. A good approach to solve this problem is Poisson editing.

With Poisson editing, instead of mixing the gray values we mix the gradients of each pixel in order to preserve the shapes. Then, we could integrate the result to get a gray-scale image back. This would eliminate any offset, but still might leave some artifacts and would remove the offset of the original image as well. A better approximation is to find the image by minimizing an energy that makes the gradient of the result close to the desired mixture while sticking selected pixels from the result to the original image. This would allow us to leave the pixels from the image being edited unmodified and modify only the ones from the patch.

Before formuating the energy, it is necessary to define the gradient mixture formally. Let u_1, u_2 be the images being mixes and m the mask that defines from which image comes each pixel (0 means form u_2 and 1 from u_1), the gradient mixture is a vector-valued image defined as:

$$v_{ij} = m_{ij}\nabla^+ u_{1,ij} + (1 - m_{ij})\nabla^+ u_{2,ij}$$

We will define the energy by adding up the square errors of the things we want to maintain:

$$E(u) = \sum_{i=1}^{N}\sum_{j=1}^{M} \left|\nabla^+ u_{ij} - v_{ij}\right|^2 + \sum_{i=1}^{N}\sum_{j=1}^{M} \beta_{ij}\left(u_{ij} - u_{2,ij}\right)^2$$

Which can be expressed in matrix form as (adding a 0.5 factor for convenience):

$$E(u) = \frac{1}{2}\langle \nabla^+ u - v, \nabla^+ u - v\rangle + \frac{1}{2}\langle B(u -_2), u - u_2\rangle$$

Where B is a diagonal matrix with the β coeffcients on the diagonal, and all images have been vectorized.

It would be convininent to find a matrix A, a vector b and a scalar c such that the energy can be expressed as follows, because we already know how to minimize it:

$$E(u) = \frac{1}{2}\langle Au, u\rangle - \langle b, u\rangle + c$$

The energy can be expressed as (recall that B is symmetric):

$$\begin{aligned}
E(u) &= 0.5\langle \nabla^+ u - v, \nabla^+ u - v\rangle + 0.5\langle B(u-u_2), u-u_2\rangle \\
&= 0.5\left(\langle \nabla^+ u, \nabla^+ u\rangle - 2\langle \nabla^+ u, v\rangle + \langle v, v\rangle + \langle Bu, u\rangle - 2\langle Bu, u_2\rangle + \langle Bu_2, u_2\rangle\right) \\
&= \langle 0.5(\nabla^+)^T \nabla^+ u, u\rangle + \langle 0.5Bu, u\rangle - \langle (\nabla^+)^T v, u\rangle - \langle Bu_2, u\rangle + \langle v, v\rangle + \langle Bu_2, u_2\rangle \\
&= 0.5\langle \left(B + (\nabla^+)^T \nabla^+\right) u, u\rangle - \langle Bu_2 + (\nabla^+)^T v, u\rangle + (\langle v, v\rangle + \langle Bu_2, u_2\rangle)
\end{aligned}$$

Therefore,

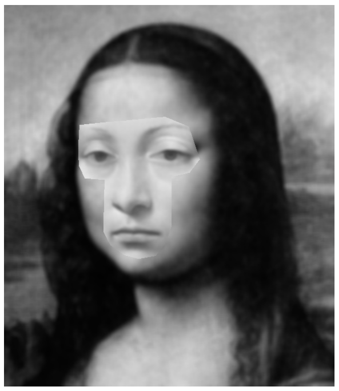
$$\begin{aligned}
A &= B + (\nabla^+)^T \nabla^+ = B - \operatorname{div}^- \nabla^+ \\
b &= Bu_2 + (\nabla^+)^T v = Bu_2 - \operatorname{div}^- v \\
c &= \langle v, v\rangle + \langle Bu_2, u_2\rangle
\end{aligned}$$

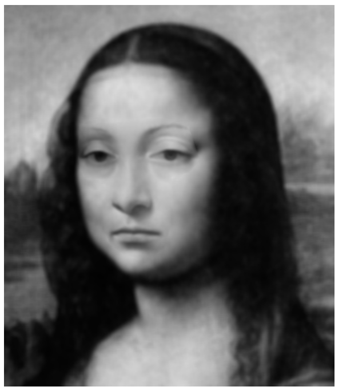
Its gradient is given by:

$$\begin{aligned}
\nabla E(u) &= Au - b \\
&= \left(B - \operatorname{div}^- \nabla^+\right) u - Bu_2 + \operatorname{div}^- v \\
&= B(u - u_2) - \operatorname{div}^-(v - \nabla^+ u)
\end{aligned}$$

However, the method has been implemented directly with A and b because the implementation is generic.

The algorithm converges very fast, and the actual number of iterations depends on the initial conditions. Starting from zeros, it takes about 500 iterations. It's more efficient to initialize x_0 to the background image, or even to the rough composition made by taking the pixels from one of the images according to the mask, which can be computed directly. Then, the algorithm takes about 400 iterations to converge. Furthermore, with smart initizalization the image already looks good with less 100 iterations, while when starting from 0 it takes a lot more. These are the results:


Direct composition


Gradient composition