Norwegian University of Science
and Technology
Department of Mathematical
Sciences

TMA4205 Numerical Linear Algebra Fall 2016

Semester project – part 2

1 Optical Flow

When a three-dimensional scene is observed through a camera lense (or an eye), the movements in the scene together with the relative movement of the lense give rise to an apparent motion within the image plane. This apparent motion is called the *optical flow* and can be used for example for object segmentation or collision detection. The goal of this project is the efficient implementation of a particular method for the estimation of this optical flow.

Denote in the following by $\Omega := [0, L_x] \times [0, L_y] \subset \mathbb{R}^2$ the image plane, and assume that we are given an image sequence (or movie) on Ω, modelled as a function $I \colon \Omega \times \mathbb{R} \to \mathbb{R}$. We are interested in estimating the optical flow at some fixed time $t_0 \in \mathbb{R}$, which we write as vector field $w \colon \Omega \to \mathbb{R}^2$ with components u and v, that is,

$$w(x, y) = \begin{pmatrix} u(x, y) \\ v(x, y) \end{pmatrix}.$$

Here u denotes the horizontal and v the vertical apparent movement in the image sequence. The basis for the estimation of this apparent movement is now the assumption that intensity values of the scene remain approximately constant along the flow. That is, for sufficiently small time steps Δt we have

$$I(x, y, t_0) = I(x + \Delta t\, u(x, y), y + \Delta t\, v(x, y), t_0 + \Delta t) + o(\Delta t).$$

Dividing by Δt and taking a limit $\Delta t \to 0$, this implies that

$$0 = \lim_{\Delta t \to 0} \frac{1}{\Delta t} \big(I(x + \Delta t\, u, y + \Delta t\, v, t_0 + \Delta t) - I(x, y, t_0) \big) = u \partial_x I + v \partial_y I + \partial_t I.$$

That is, the functions u and v solve the optical flow equation

$$u \partial_x I + v \partial_y I = -\partial_t I. \tag{1}$$

Because this is a single equation for the two functions u and v, we cannot expect a unique solution. Roughly spoken, the equation (1) only provides a description of the optical flow across edges in the image, but not along edges. This non-uniqueness of the solution of the optical flow equation is called the *aperture problem* in optical flow. In order to tackle this problem and at the same time deal with approximation errors, it is necessary to include additional assumptions concerning the behaviour of the optical flow. One of the simplest assumption is smoothness: the values of u and v should change only slowly in space. One

possibility to realise this assumption is to require that the functions u and v solve the optimisation problem

$$\frac{1}{2}\|u\partial_x I + v\partial_y I + \partial_t I\|^2 + \frac{\lambda}{2}\big(\|\nabla u\|^2 + \|\nabla v\|^2\big) \to \min,$$

where $\lambda > 0$ is some regularisation parameter trading off smoothness of u and v against accuracy of the solution of the optical flow equation.

The Euler–Lagrange equations for this variational problem (that is, the optimality conditions for this optimisation problem) are the coupled PDEs

$$\begin{aligned}(u\partial_x I + v\partial_y I)\partial_x I - \lambda\Delta u &= -\partial_x I\,\partial_t I,\\ (u\partial_x I + v\partial_y I)\partial_y I - \lambda\Delta v &= -\partial_y I\,\partial_t I,\end{aligned} \qquad \text{in } (0, L_x)\times(0, L_y), \tag{2}$$

with either homogeneous Neumann boundary conditions both for u and v, that is,

$$\begin{aligned}\partial_x u = \partial_x v = 0 \quad &\text{on } \{0, L_x\}\times[0, L_y],\\ \partial_y u = \partial_y v = 0 \quad &\text{on } [0, L_x]\times\{0, L_y\},\end{aligned}$$

or homogeneous Dirichlet boundary conditions

$$u = v = 0 \qquad \text{on } \{0, L_x\}\times[0, L_y] \cup [0, L_x]\times\{0, L_y\} \tag{3}$$

if one wants to include the assumption that the flow at the boundary of the image plane is zero.

See Figure 1 for an example of the resulting optical flow computed from two consecutive images in an image sequence.

2 Discretisation

For the discretisation, we assume that we are given two consecutive frames I_0 and I_1 of an image sequence at times $t = 0$ and $t = \Delta t$. The images I_0 and I_1 themselves are greyscale images given on a rectangular (pixel) grid of size $m \times n$. For simplicity, we assume that $\Delta t = 1$ and the grid size of the pixel grid is $\Delta x = \Delta y = h := 1$. In this case, the time derivative of I at $t = 0$ can be roughly approximated by

$$\partial_t I(x_i, y_j) := I_1(x_i, y_j) - I_0(x_i, y_j),$$

and the space derivatives by

$$\partial_x I(x_i, y_j) := \begin{cases} I_0(x_{i+1}, y_j) - I_0(x_i, y_j) & \text{if } i < m,\\ 0 & \text{if } i = m,\end{cases}$$

and

$$\partial_y I(x_i, y_j) := \begin{cases} I_0(x_i, y_{j+1}) - I_0(x_i, y_j) & \text{if } j < n,\\ 0 & \text{if } j = n.\end{cases}$$

The flow field $w = (u, v)$ is discretised on the same pixel grid as the given image sequence, and for the discretisation of the Laplacian of u and v, we use the usual 5-point stencil, e.g.,

$$(\Delta u)_{ij} \approx \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij}}{h^2}.$$

Moreover we assume in the following that there should be no flow across the image boundaries, and therefore we use homogeneous Dirichlet boundary conditions (3).

Figure 1: Result of an optical flow computation. *First row:* Two consecutive frames in an image sequence. *Second row:* The resulting flow *(right)* and the first frame of the sequence overlaid with the flow *(left)*. The colourwheel on the bottom right indicates the direction and intensity of the flow field at each pixel. The test images have been taken from `http://vision.middlebury.edu/flow/`. The optical flow has been computed with a regularisation parameter of $\lambda = 1000$, and the images were pre-smoothed using a Gaussian kernel with standard deviation $\sigma = 5$.

3 Problem Setting

1 Implement a version of the conjugate gradient (CG) method for the solution of the optical flow problem (2) with Dirichlet boundary conditions (3). The method should work directly on the grid and should contain a convergence test so that it terminates when

$$\frac{\|r_k\|_2}{\|r_0\|_2} < \text{tol},$$

where r_k denotes the residual at step k.

The implementation should consist of two functions, first a main function where the images are imported and their derivatives computed and then a function where the actual CG method is implemented. The function header for the CG method could for instance look like this:

```
function [u,v] = OF_cg(u0, v0, Ix, Iy, lambda, rhsu, rhsv, tol, maxit)
%
% [u,v] = OF_cg(u0, v0, Ix, Iy, rhsu, rhsv, tol, maxit) performs
% the CG method for solving the optical flow problem.
%
% input:
% u0     - initial guess for u
% v0     - initial guess for v
% Ix     - x-derivative of the first frame
% Iy     - y-derivative of the first frame
% lambda - regularisation parameter
% rhsu   - right-hand side in the equation for u
% rhsv   - right-hand side in the equation for v
% tol    - relative residual tolerance
% maxit - maximum number of iterations
%
% output:
% u      - numerical solution for u
% v      - numerical solution for v
```

Test the method using the images `frame10.png` and `frame11.png` from the webpage of the course. Reasonable estimates of the optical flow should be attainable with regularisation parameters λ ranging from 500 to 10000 (or even larger). In order to obtain better results, it is helpful to pre-smooth the images I_0 and I_1 (see the *Hints* below). Additional test sequences can be downloaded from `http://vision.middlebury.edu/flow/`.

2 Implement a multigrid V-cycle for the solution of the optical flow problem (2) with Dirichlet boundary conditions (3). Again, implement a main function from which the V-cycle is called, and another function containing the actual implementation of the V-cycle. This function should take as input the initial guess, the right-hand side, the current level and maximal level of the grid, the number of pre-smoothings, and the number of post-smoothings. Use the CG algorithm from **1)** to solve the problem on the coarsest level. Both weighted Jacobi and red-black Gauss–Seidel should be implemented as smoothers.

It is advantageous to further break down the function where the V-cycle is performed into several parts. The main routine could for instance look like this:

```
function [u,v] = mg_OF(u0,v0, Ix, Iy, lambda, rhsu, rhsv, s1, s2, ...
                                                level, max_level)
%
% [u,v] = mg_OF(u0,v0, Ix, Iy, lambda, rhsu, rhsv, s1, s2,...
%                 ..., level, max_level) performs one multigrid
% V-cycle for the optical flow problem.
%
% input:
% u0        - initial guess for u
% v0        - initial guess for v
% Ix        - x-derivative of the first frame
% Iy        - y-derivative of the first frame
% lambda    - regularisation parameter
% rhsu      - right-hand side in the equation for u
% rhsv      - right-hand side in the equation for v
% s1        - number of pre-smoothings
% s2        - number of post-smoothings
% level     - current level
% max_level - total number of levels
%
% output:
% u      - numerical solution for u
% v      - numerical solution for v

if level == max_level
    [u,v] = OF_cg(u0, v0, Ix, Iy, lambda, rhsu, rhsv, 1e-6, 1000)
else
    [u,v] = jacobi(u0, v0, Ix, Iy, lambda, rhsu, rhsv, level, ...
                                                          2/3, nu1);
    [rhu,rhv] = residual(u, v, Ix, Iy, lambda, rhsu, rhsv);
    [r2hu,r2hv,Ix2h,Iy2h] = restriction(rhu, rhv, Ix2h, Iy2h);
    [e2hu,e2hv] = mg_OF(zeros(size(r2hu)), zeros(size(r2hv)), ...
         Ix2h, Iy2h, lambda, r2hu, r2hv, s1, s2, level+1, max_level);
    [ehu,ehv]  = interpolation(e2hu, e2hv);
    u = u + ehu;
    v = v + ehv;
    [u,v] = jacobi(u, v, Ix, Iy, lambda, rhsu, rhsv, level, 2/3, nu2);
end
```

Notice that the routine is defined recursively. What is left then is to implement the routines `jacobi.m`, `residual.m`, `restriction.m` and `interpolation.m`. In addition, red-black Gauss–Seidel (`gs_rb.m`) must be implemented.

Use the same test images as for problem **1)**, and test the program for different choices of the number of pre- and post-smoothings, as well as for different regularisation parameters λ. Are there any differences between using weighted Jacobi and red-black Gauss–Seidel as relaxation method? Does the number of iterations to convergence depend on λ?

3 Modify the routine in **1)** to solve the optical flow with a PCG method using the multigrid V-cycle from **2)** as preconditioner. That is, instead of solving the linear system $Mz = r$ that is usually required in each PCG iteration, we compute z by applying the multigrid solver (with a reasonably small number of smoothing steps s_1 and s_2) with right hand side r. This preconditioner, however, has to be applied with some care. What properties must a preconditioner for the CG method have, and what does this mean for the multigrid V-cycle?

Test the algorithm on the same problem as in **1)**. How does the number of iterations to reach convergence depend on the different parameters in the problem, and how does the computation time changes with the parameters?

The report should include a short description of the discretisation, and for the different problems at least:

1) A plot of the solution and the convergence history, that is, a plot of the Euclidean norm of the residual against the number of iterations. This should be done for different values of λ.

2) A plot of the approximate solution after each of the first five multigrid V-cycles. Include a plot of the convergence history. This should be done with both weighted Jacobi and red-black Gauss–Seidel as relaxation method for different numbers of smoothing iterations.

3) A plot of the initial guess and the solution after each of the first five iterations in the PCG method. Again, include a plot of the convergence history and report how the number of iterations until convergence depends on λ.

Additionally, compare the different methods to each other: Which computation times can be achieved with the different methods for different parameter settings?

Use the condition $\|r_k\|_2/\|r_0\|_2 < 10^{-8}$ as convergence criterion in all the computations.

A Dealing with Images

In MATLAB, images can be exported using the functions `imread` and `imwrite`. Note that `imread` will load standard greyscale images as unsigned integers; in order to perform most numerical computations, it is therefore often necessary to convert the imported images to double precision numbers. In order to load the image *frame10.png*, it is thus possible to use the command `I0 = double(imread('frame10.png'));`. In order to display RGB images, one may use the function `imshow`. At the webpage of the course, a MATLAB program can be found for converting two-dimensional flow fields into RGB-images (`mycomputeColor.m`).

In case of fast movements in the images (or a low frame rate), the results of the optical flow computations improve, if one applies some smoothing to the input images I_0 and I_1. This can, for instance, be done by calling the function `imgaussfilt(I0,sigma)`, which convolves the image `I0` with a Gaussian kernel with standard deviation σ. As an example, the results in Figure 1 were obtained with $\sigma = 5$.

B Matrix Operations

Use matrix- or vector-operations whenever possible. E.g., for solving the discretised Poisson equation with a 5-point stencil, the Jacobi iteration reads as

$$u_{i,j}^{(k+1)} = \tfrac{1}{4}\big(u_{i,j-1}^{(k)} + u_{i-1,j}^{(k)} + u_{i,j+1}^{(k)} + u_{i+1,j}^{(k)} + h^2 f_{i,j}\big), \quad 2 \le i, j \le N.$$

In MATLAB this can be written as the matrix operation

```
index = 2 : N;
u(index, index) = 0.25 * (  u(index, index - 1) + u(index - 1, index) ...
                  + u(index, index + 1) + u(index + 1, index) ...
                  + h^2 * f(index, index));
```

which is much more efficient than

```
unew = u;
for i = 2 : N
  for j = 2 : N
    unew(i, j) = 0.25 * (u(i, j - 1) + u(i - 1, j) + u(i, j + 1) ...
                 + u(i + 1, j) + h^2 * f(i, j));
    end
  end
u = unew;
```

In case of the optical flow equation, the situation is similar.