Interpolation surface

We have a matrix V that represents the values the values of a function $f(i,j) = V_{ij}$ for integer arguments (i,j). We want to define a smooth (continuously differentiable) function $f(x,y)$ that interpolates the values of f for real arguments (x,y). In other words, we want to be able to take some data as shown in Figure 1 and produce a smooth surface as shown in Figure 2.

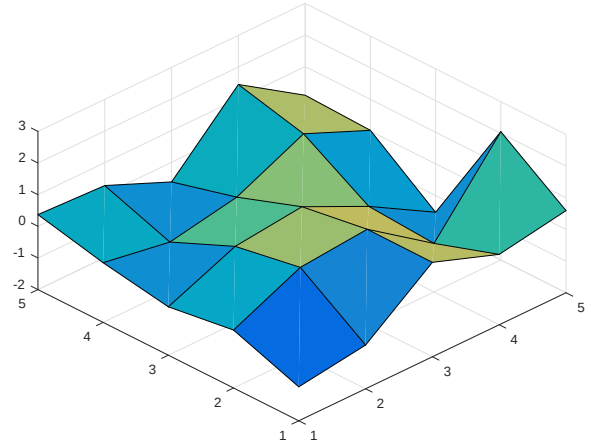

Figure 1: A random 5×5 matrix of values as plotted by the `surf` command which automatically interpolates the data using linear functions.

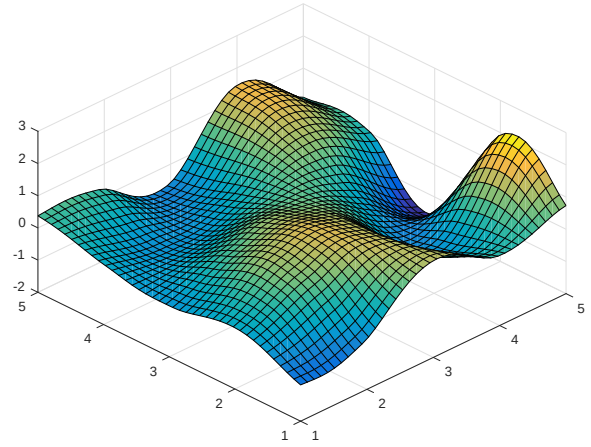

Figure 2: A smooth function $f(x,y)$ that is defined everywhere on $[1,5] \times [1,5]$ and agrees with the values shown in Figure 1 for integer arguments.

The idea is to define the function on the entire domain $[1,n] \times [1,m]$ by pasting together functions defined on the individual squares of unit size. While doing this we must of course ensure that the values of the function f and also the values of its

derivatives everywhere on the edges of the individual squares agree with the values and derivatives of the function on the edges of the adjacent squares. This can be done in several ways, but the idea of the solution you must implement is based on the concept of the so-called *partition of unity*.

1. task: (1 point) Find the 3. order polynomial $p(x)$ which satisfies the conditions

$$p(0) = 1, p'(0) = 0, p(1) = 0, p'(1) = 0 \tag{1}$$

Let us first concentrate on defining the interpolation function on the unit square $K = [0,1] \times [0,1]$.

Assume we are given the values of the function on the vertices of the unit square $A(0,0)$,$B(1,0)$,$C(0,1)$,$D(1,1)$ and that we denote these values by a, b, c, d. Assume also that we are given the desired values of the partial derivatives in the x and y directions for the function in the vertices A, B, C, D. Denote these values by a_x, b_x, c_x, d_x and a_y, b_y, c_y, d_y. This means we have 12 values altogether for a given square.
First we define the *weights*

$$\begin{aligned} w_A(x,y) &= p(x)p(y) \\ w_B(x,y) &= (1-p(x))p(y) \\ w_C(x,y) &= p(x)(1-p(y)) \\ w_D(x,y) &= (1-p(x))(1-p(y)) \end{aligned} \tag{2}$$

where p is the solution to (1). Then we define auxiliary functions which have exactly the desired values and derivatives in the respective vertices

$$\begin{aligned} f_A(x,y) &= a + a_x x + a_y y \\ f_B(x,y) &= b + b_x(x-1) + b_y y \\ f_C(x,y) &= c + c_x x + c_y(y-1) \\ f_D(x,y) &= d + d_x(x-1) + d_y(y-1) \end{aligned} \tag{3}$$

and lastly we define the function f (where we omit writing the argument (x,y) for simplicity)

$$f = f_A w_A + f_B w_B + f_C w_C + f_D w_D \tag{4}$$

2. task: (4 points) Write an efficient function `Z=interpolationFunction(data,len)` that computes the values of the function (4) on a `len` $\times$ `len` mesh of points in the unit square K and returns them in the `len` $\times$ `len` matrix `Z`. The 12 values needed are contained in the variable `data`. How you arrange these values is up to you (`data` can be a row, column, matrix or ...).

The next task is with regard to the theoretical explanation of the interpolation function in (4). It is not necessary for the implementation of the solution, it is only important for understanding how this type of interpolation works.

3. task: (4 points)

- Show that for the functions (2) we have

$$w_A + w_B + w_C + w_D = 1$$

everywhere on K (this is where the name partition of unity comes from). This equality ensures that in the case of constant data (or that the data is a linear function of (i, j)) we get a constant (resp. linear) function f. Justify this last statement.

- Compute the values and values of the partial derivatives of f from (4) in the vertices A, B, C, D. It enough to show what we get for one vertex.
- Show also that the values of f and its derivatives on a given edge of the square K depend only the data that refers to the vertices of that edge. Here it also suffices to show this for one of the edges.

4. task: (5 points) Write the function `interpolation(V,len)` that plots the data from `V` (using the `surf` or `mesh` commands) like in Figure 1 and then plots the interpolation surface as in Figure 2 by pasting together the functions from (4) for the individual squares. Here `len` is the same argument as for `interpolationFunction` (in Figure 2 we have `len=10` for example). While calling the function `interpolationFunction` you must define a sensible way of determining the desired partial derivatives at the vertices of the squares. It is essential that the arguments for the edge of a given square agree with the arguments for the function call for the same edge in the adjacent square. A natural choice for the partial derivative at a given vertex in a given direction is to take the derivative of the linear function that passes through the adjacent vertices in that direction. For instance, for the derivative in the x direction at the (i, j) vertex you can take

$$\frac{\partial f}{\partial x}(i, j) = \frac{1}{2}\left(f(i+1, j) - f(i-1, j)\right)$$

(of course for vertices (i, j) on the the edge of the domain other arrangements must be made).

Homework submission

Submit the following files

1. the files `interpolationFunction.m` and `interpolation.m` containing the code and comments.
2. the file (report) **solution.pdf** which includes the theoretical explanations and some illustrations of the obtained surfaces for chosen data.