Fakulteta za računalništvo in informatiko, Univerza v Ljubljani

Mathematical Modeling

Second Home Assignment

Report

Author

Jernej Vivod

Lecturer

Dr. Neža Mramor Kosta

April 17, 2018

Contents

1 Problem

1.1 Problem Statement

Given two parametric curves in R^3 the task is to find two points, one on each curve, such that the distance between them is minimal. The curves are given as

$$\begin{aligned} r_1 &= F(t);\ t \in R \\ r_2 &= G(s);\ s \in R \end{aligned} \tag{1}$$

We are looking for parameters t and s for which the vector

$$\vec{r} = F(t) - G(s) \tag{2}$$

has the smallest possible magnitude.

A MATLAB/Octave function that solves such a problem should also be written and generalized to work for parametric surfaces as well.

2 Solution

2.1 Finding the Minimum Distance between the Curves

There are many ways to compute the minimum distance between two points on the curves and to find the parameters at which such shortest distance is achieved. One way would be to write a simple distance formula

$$\vec{r} = ||F(t) - G(s)|| \tag{3}$$

This can be rewritten as

$$\vec{r} = \sqrt{(F(t) - G(s)) \cdot (F(t) - G(t))} \tag{4}$$

This expression can now be minimized using the Newton optimization method.

The problem with this method is that the second derivatives needed for the Hessian matrix become really long expressions that are impractical for programming purposes. We can use the observation, that the shortest distance vector will always be perpendicular to both curves - that is, the vector r, as it attains its shortest distance, will be perpendicular to the tangent lines at points $F(t)$ and $G(t)$. This fact can be used to write a function that attains a zero value, when the distance vector is perpendicular to both curves. The distance vector can be written as $\vec{r} = F(t) - G(t)$. When this vector is perpendicular to the curve given by F, the equation

$$\vec{r} \cdot F'(t) = (F(t) - G(s)) \cdot F'(t) = \vec{0} \tag{5}$$

will hold. We can reason similarly for the curve G.

This fact can be used to write a vector function that attains zero value when the distance vector is perpendicular to both tangent lines:

$$slopes(t, s) = \begin{bmatrix} \vec{r} \cdot F'(t) \\ \vec{r} \cdot G'(s) \end{bmatrix} = \begin{bmatrix} (F(t) - G(t)) \cdot F'(t) \\ (F(t) - G(t)) \cdot G'(s) \end{bmatrix} \tag{6}$$

The solution can be found by setting this function equal to $\vec{0}$ ($slopes(t, s) = \vec{0}$) and solving for t and s.

2.2 Approximating the Root of the slopes Function

One way to solve this equation is to use the Newton iteration method to approximate the root. The Newton iteration step can be written as

$$parameters^{(k+1)} = parameters^{(k)} - J_{slopes}^{-1} slopes(x^{(k)}) \tag{7}$$

Where *parameters* is a column vector representing the input parameters t and s, J_{slopes}^{-1} is the inverse of the Jacobian matrix of the slopes function and *slopes* is the function that attains zero value when distance vector $\vec{r}$ is minimal.

The Jacobian matrix of a vector function is defined as:

$$J = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{8}$$

The Jacobian matrix of the *slopes* function can be written as:

$$J_{slopes} = \begin{bmatrix} (F(t) - G(s)) \cdot F''(t) + F'(t) \cdot F'(t) & (-G'(s)) \cdot F'(t) \\ F'(t) \cdot G'(s) & (F(t) - g(s)) \cdot G''(s) + (-G'(s)) \cdot G'(s) \end{bmatrix} \tag{9}$$

2.3 Generalizing the Problem to include Parametric Surfaces

Parametric surfaces are defined as surfaces in the Euclidean space R^3 defined by a parametric equation with two parameters $\vec{r} : R^2 \to R^3$. Typically the function describing such a surface is written as

$$F(u, v) = \begin{bmatrix} f_1(u, v) \\ f_2(u, v) \\ f_3(u, v) \end{bmatrix}.$$

Similarly as with curves, the vector spanning the shortest distance between two parametric surfaces will be perpendicular to both tangent planes at points $F(t, s)$ and $G(u, v)$.

The tangent plane at point $F(t = a, s = b)$ is spanned by the vectors representing the partial derivatives $\frac{\partial F}{\partial t}$ and $\frac{\partial F}{\partial s}$ evaluated at $t = a,\ s = b$.

A vector $\vec{r}$ is perpendicular to this normal plane if it is perpendicular to both of the vectors spanning the tangent plane.

We can now again try to write a function (similar to *slopes* from previous subsection) that attains zero value when the distance vector $\vec{r}$ is perpendicular to both tangent planes.

For the distance vector $\vec{r}$ to be perpendicular to both tangent planes, it must be perpendicular to both vectors spanning the tangent plane at $F(t,s)$ and both vectors spanning the tangent plane at $G(u,v)$. As we discussed above, these vectors are evaluations of partial derivatives $\frac{\partial F}{\partial t}$, $\frac{\partial F}{\partial s}$, $\frac{\partial G}{\partial u}$, $\frac{\partial G}{\partial v}$ at certain values of parameters t, s, v and u.

This function (let's again call it *slopes*) can be written as:

$$slopes(F,\ G) = \begin{bmatrix} \vec{r} \cdot \frac{\partial F}{\partial t} \\ \vec{r} \cdot \frac{\partial F}{\partial s} \\ \vec{r} \cdot \frac{\partial G}{\partial u} \\ \vec{r} \cdot \frac{\partial G}{\partial v} \end{bmatrix} \tag{10}$$

Where $\vec{r} = F(t,\ s) - G(u,\ v)$ (the distance vector).

We can again write the Jacobian matrix (which will now be a 4×4 matrix) and use it to approximate the roots of this vector function to find values of parameters at which the distance vector is perpendicular to both tangent planes and hence has the smallest magnitude.

2.4 Generalizing the Problem to Arbitrary Dimensions

We can generalize the derived equations and write their general forms that work in arbitrary dimensions where the functions are given by $F\ :\ R^m \to R^k$ and $G\ :\ R^n \to R^k$ for arbitrary natural numbers m, n and k.

For the shortest distance to be achieved, the distance vector connecting the surfaces at $F(t_1, ..., t_n)$ and $G(s_1, ..., s_m)$ must be perpendicular to all vectors spanning the k dimensional tangent hyperplane at both points.

The familiar *slopes* function discussed at length in previous sections can be generalized to n dimensions as

$$slopes(t_1, ..., t_n, s_1, ..., s_m) = \begin{bmatrix} \vec{r} \cdot \frac{\partial F}{\partial t_1} \\ \vdots \\ \vec{r} \cdot \frac{\partial F}{\partial t_n} \\ \vec{r} \cdot \frac{\partial G}{\partial s_1} \\ \vdots \\ \vec{r} \cdot \frac{\partial G}{\partial s_m} \end{bmatrix} \tag{11}$$

Take n and m parameters respectively, the Jacobian will be constructed by differentiating all $n + m$ functions by all $n + m$ parameters. Following the algorithm for constructing the Jacobian, generalized *slopes* function will yield a $(n + m) \times (n + m)$ Jacobian matrix.

The parameters that give the points with shortest distance between them can again be computed by solving the equation

$$slopes(t_1, ..., t_n, s_1, ..., s_m) = \vec{0} \tag{12}$$

using the familiar Newton method generalized to work for $n + m$ input parameters.

The solution can be checked by computing the dot product of distance vector $\vec{r} = F(t_1, ..., t_n) - G(s_1, ..., s_m)$ with the tangent hyperplanes spanned by the derivatives of the parametric functions with respect to their parameters. If the distance vector is perpendicular to both surfaces, the dot products should both be zero.

3 Writing the MATLAB/Octave Code

Our task is to write a function given by the header:

```
function [d, t, s] = razdalja(F, G, t0, s0)
```

which returns the parameters t and s (at which the magnitude of the vector $\vec{r} = F(t) - G(t)$ is the smallest possible) and the distance d. The function should take function handles for the functions representing the two curves and the initial estimates of the parameters.

The function handles are functions of the form:

```
 function [r, dr, ddr] = curve(parameter)
```

The vector r is a column vector containing the evaluation of the curve at the passed parameter. Likewise, dr and ddr are column vectors containing the first and second derivative evaluations of the curve at the passed parameter.

We can use MATLAB/Octave's functionality to quickly and elegantly derive functions that only return the value that we want:

```
F = @(t) F(t);
dF = @(t) [~, res] = F(t);
ddF = @(t) [~, ~, res] = F(t);
G = @(t) G(t)
dG = @(t) [~, res] = G(t)
ddG = @(t) [~, ~, res] = G(t)
```

These functions can then be used to construct the slopes function and its Jacobian matrix:

```
function [slopes] = make_slopes(F, dF, G, dG)
slopes = @(t, s) [dot((F(t) - G(s)) , dF(t)) ;
dot((F(t) - G(s)), dG(s))];
endfunction

function [Jslopes] = make_jacobian(F, dF, ddF, G, dG, ddG)
Jslopes = @(t, s) [dot((F(t) - G(s)), ddF(t)) + dot(dF(t), dF(t)),
     dot(-dG(s), dF(t));
                       dot(dF(t), dG(s)),
                       dot((F(t) - G(s)),
                       ddG(s) + dot(-dG(s), dG(s))];
endfunction
```

These two functions are all we need to implement an algorithm for finding the root of the slopes function using the Newton's method:

```
% function [params, num_it] = newton(slopes, Jslopes, tolerance, max_it, params_0)
%
% Approximates the root of the function slopes. Finds parameters t and s for which
% the value of the function slopes is approximately [0, 0].

function [params, num_it] = newton(slopes, Jslopes, tolerance, max_it, params_0)
  % Set parameters to initial parameters
  params = params_0;
  % iterate max_it times
  for it = 1:max_it
    % Newton iteration step
    params = params - pinv(Jslopes(params(1), params(2)))*slopes(params(1), params(2));
    % if difference between old and new parameters is small enough
    if norm(params - params_0) < tolerance
      break
    endif
    % set old parameters to current parameters
    params_0 = params;
  endfor
  % save number of iterations performed
  num_it = it;
endfunction
```

Once we have approximated the parameters t and s, we can easily compute the distance between the two points as:

```
d = norm(F(t) - G(s));
```

3.1 Generalization to Include Parametric Surfaces

Using the ideas discussed in the previous section discussing this topic, we can modify the functions written above so they solve this same problem stated in terms of parametric surfaces.

The *slopes* function will again be a function, which achieves zero value when the distance vector represented as $F(t,\ s) - G(u,\ v)$ is perpendicular to both tangent planes.

```
function [slopes] = make_slopes_surf(F, d10F, d01F, G, d10G, d01G)
slopes = @(t, s, u, v)...
[dot(F(t, s) - G(u, v), d10F(t, s));
 dot(F(t, s) - G(u, v), d01F(t, s));
 dot(F(t, s) - G(u, v), d10G(u, v));
 dot(F(t, s) - G(u, v), d01G(u, v))
];
endfunction
```

The function that implements the Newton method to find the roots of the *slopes* function can easily be modified to work with parametric surfaces. All we need to do is change the input to the *slopes* function and its Jacobian as we are now dealing with 4 parameters.

After computing the parameters, the magnitude of the found distance vector can again simply be computed using the MATLAB/Octave *norm* function.

```
d = norm(F(params(1), params(2)), G(params(2), params(3)));
```

3.2 A Simple Test Case

This subsection demonstrates the functionality of the *razdalja* MATLAB/Octave function on a simple example. Consider a parametrized unit circle with center at the origin and a parametrized line going through $(2, 0) and (0, 2)$.

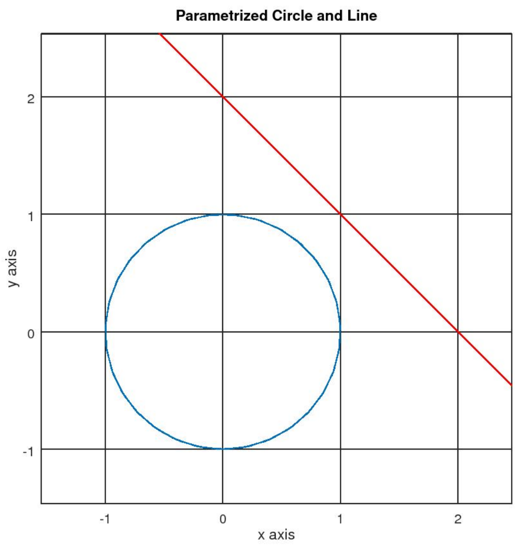

Figure 1: Parametrized unit Circle and Line

The input functions to *razdalja* representing the two curves can be written follows:

```
function [r, dr, ddr] = line(s)
    r = [s; 2 - s];
    dr = [1; -1];
    ddr = [0; 0];
endfunction
```

```
function [r, dr, ddr] = circle(t)
    r = [sin(t); cos(t)];
    dr = [cos(t); -sin(t)];
    ddr = [-sin(t); -cos(t)];
endfunction
```

Consider $t = 0$ and $s = 0$ to be the initial estimates for parameters t and s. Calling the *razdalja* function as

```
[d, t, s] = razdalja(@circle, @line, 0, 0)
```

yield the results

```
>> t
t =  0.78540
>> s
s =  1
>> d
d =  0.41421
```

We can check that the vector distance vector is orthogonal to both curves at these points by computing the dot product of the distance vector with the derivatives $F'(t)$ and $G'(s)$.

```
>> r = circle(t) - line(s)
r =


  -0.29289
  -0.29289


>> dot(r, [~, res] = circle(t))
ans =   -1.1102e-16
>> dot(r, [~, res] = line(s))
ans =   -1.1102e-16
```

We can see that the dot product is practically equal to zero which implies orthogonality.

The fact that the found parameters really represent the closes points can be further reaffirmed by plotting the distance vector between them.

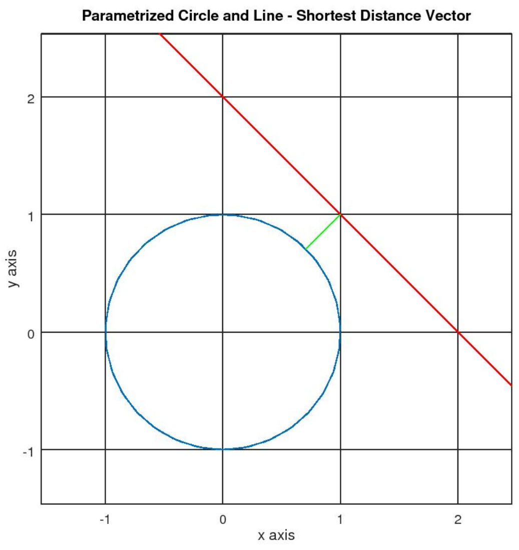

Figure 2: Distance vector through $F(t = 0.78540)$ and $G(s = 1)$