# Intersection of two implicit surfaces

A surface in $\mathbb{R}^3$ can be viewed as a solution set of an equation $f(\mathbf{x}) = 0$, where $\mathbf{x} = [x_1, x_2, x_3]^\mathsf{T} \in \mathbb{R}^3$, and $f$ is a function of three variables. Suppose we have two surfaces given by $f_1(\mathbf{x}) = 0$ and $f_2(\mathbf{x}) = 0$. The intersection of these two surfaces is the solution set of the nonlinear system

$$\begin{aligned} f_1(\mathbf{x}) &= 0, \\ f_2(\mathbf{x}) &= 0. \end{aligned}$$

If $f_1$ and $f_2$ are smooth functions and some additional conditions are satisfied the intersection of these two surfaces is a smooth curve $K$. Your objective is to find this curve.

### Construction of the curve $K$

View equations $f_1(\mathbf{x}) = 0$ and $f_2(\mathbf{x}) = 0$ as equations of level sets of $f_1$ and $f_2$. The curve $K$ is the intersection of these two level sets. Gradients of $f_1$ and $f_2$ are orthogonal to $K$ at each point of $K$. In other words, the vector $(\operatorname{grad} f_1) \times (\operatorname{grad} f_2)$ is tangent to $K$. Set

$$\mathbf{F}(\mathbf{x}) = \frac{(\operatorname{grad} f_1(\mathbf{x})) \times (\operatorname{grad} f_2(\mathbf{x}))}{\|(\operatorname{grad} f_1(\mathbf{x})) \times (\operatorname{grad} f_2(\mathbf{x}))\|}.$$

This is now a vector–valued function $\mathbf{F}\colon \mathbb{R}^3 \to \mathbb{R}^3$.

We will describe the curve $K$ with the following geometric approach. Suppose we start at $\mathbf{x}_0 \in K$, i.e., $f_1(\mathbf{x}_0) = 0$ and $f_2(\mathbf{x}_0) = 0$. Take a small step in the direction tangent to $K$ from $\mathbf{x}_0$ to get a new point $\mathbf{y}$. According to the definition of $\mathbf{F}$, that means $\mathbf{y} = \mathbf{x}_0 + h\mathbf{F}(\mathbf{x}_0)$ for some small $h > 0$. This $\mathbf{y}$ does *not* (necessarily) lie on the curve $K$, but it is sensible to assume that it is close to $K$ (since $h$ is small). Note that $\mathbf{F}(\mathbf{y}) \cdot \mathbf{x} = \mathbf{F}(\mathbf{y}) \cdot \mathbf{y}$ is the equation of a plane, which is 'close' to a plane normal to the curve $K$. We can now use $\mathbf{y}$ to obtain $\mathbf{x}_1$, which actually lies on $K$ by solving the following system of nonlinear equations (in the unknown $\mathbf{x}$)

$$\begin{aligned} f_1(\mathbf{x}) &= 0, \\ f_2(\mathbf{x}) &= 0, \\ \mathbf{F}(\mathbf{y}) \cdot \mathbf{x} - \mathbf{F}(\mathbf{y}) \cdot \mathbf{y} &= 0. \end{aligned} \tag{1}$$

We will obtain the solution $\mathbf{x}_1$ of this system by using the Newton's iteration with initial guess $\mathbf{y}$. Expectation is, of course, that $\mathbf{x}_1$ is close $\mathbf{y}$. (If it is not, our choice of $h$ was too large.)

Let's summarize the construction of consecutive points on the curve $K$:

(i) Take a small step in the direction of $\mathbf{F}(\mathbf{x}_0)$ from $\mathbf{x}_0 \in K$ to get an intermediate approximation $\mathbf{y}$,

(ii) then use the system (1) to find a next point $\mathbf{x}_1 \in K$ on the curve.

Repeat this for $\mathbf{x}_1$ and (again through the intermediate approximation and system (1)) get $\mathbf{x}_2 \in K$, etc. What we get is a sequence of points

$$\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n,$$

describing the curve $K$.

A minor problem regarding the sensibility of this construction: In practice the starting point $\mathbf{x}_0 \in K$ is not known, we only know an approximation $\mathbf{y}$ for $\mathbf{x}_0$. A quick remedy: Solve the system (1) with this approximation, get $\mathbf{x}_0 \in K$, and use the method described above.

## Extra credit: A better 'guess' for the intermediate approximation y

The method described is in fact the explicit Euler method for solving systems of ordinary differential equations. Indeed, the natural parametrization $\mathbf{x}(t)$ of $K$ is the solution of a system of autonomous differential equations

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}).$$

Picking $\mathbf{x}(0) = \mathbf{x}_0$ as our initial condition, where $\mathbf{x}_0 \in K$, the solution is precisely the natural parametrization of $K$.

Use one step of the Runge–Kutta method of $4^{\text{th}}$ order with step size $h > 0$ to determine the intermediate approximation $\mathbf{y}$, and then use system (1) as before to find the next point on the curve $K$.

## Task

1. Write down the vector–valued function $\mathbf{G}\colon \mathbb{R}^3 \to \mathbb{R}^3$ and its Jacobi matrix $J\mathbf{G}$ corresponding to the system (1). (Both can be expressed using $f_i$, grad $f_i$, and $\mathbf{y}$.)

2. Write a Julia function

   `intersectionOfSurfaces(f1, gradf1, f2, gradf2, X0, h, n; tol, maxit)`,

   which given arguments:

   - functions `f1, f2`: $\mathbb{R}^3 \to \mathbb{R}$ of a vector argument `X` $\in \mathbb{R}^3$,
   - gradients `gradf1, gradf2`: $\mathbb{R}^3 \to \mathbb{R}^3$, vector–valued functions $\mathbb{R}^3 \to \mathbb{R}^3$,
   - approximation (a Vector) `X0` for the position vector of the initial point on the curve (this has to be 'adjusted' using the system (1) first),
   - step length `h`,
   - the number `n` of consecutive points on the curve to be constructed,

   and keyword arguments (with default values)

- the tolerance `tol=1e-8` for Newton's iteration, and
- maximum allowed number of iteration steps `maxit=100` for Newton's iteration

returns a Vector of size $n+1$ containing Vectors of size 3 representing the position vectors of points on the curve $K$. *Stick to specifications!*

3. If you do the extra credit task also write a Julia function

   `intersectionOfSurfacesRK4(f1, gradf1, f2, gradf2, X0, h, n; tol, maxit),`

   which accepts the same kind of arguments and returns the same type outputs as the function `intersectionOfSurfaces`, but uses the Runge–Kutta method of $4^{\text{th}}$ order (instead of the Euler method) to find intermediate approximations. Compare the performance of both functions.

4. Find some nice examples for $f_1$ and $f_2$, use your function(s) to find the corresponding curves $K$, and include the plots of these curves in your report.

Additional requirement: Assume that the file `supportfunctions.jl` (with the function `newton`) and the *Julia* `LinearAlgebra` package will be included in the test environment. This is *not a 'soft' assumption*; prior to testing your submission all methods for the function `newton` will be deleted and default method from that file will be used instead.

## Submission

Use the online classroom to submit the following:

1. A Julia file **intersectionOfSurfaces.jl** containing the function `intersectionOfSurfaces` (and, if you do the extra credit task, `intersectionOfSurfacesRK4`). Both functions should be well commented.

2. A report file **solution.pdf** which contains the necessary derivations and answers to questions above. Your report should also include a statement explaining:

   - Who or what (AI tool) did you consult while solving this home assignment?
   - Did you use AI tools for coding? To what extent is the code your own work and to what extent is it AI-generated?

While you can discuss solutions of the problems with your colleagues, the programs and report must be your own creation. You can use all Julia functions from lab sessions.
Full solution to this assignment (without the extra credit task) is worth 15 points. The assignment will first be reviewed and graded by teaching assistants – (maximum) 10 points. Then, at an appropriate time, you will take a short quiz in the online classroom regarding your solution to the assignment – (maximum) 5 points.