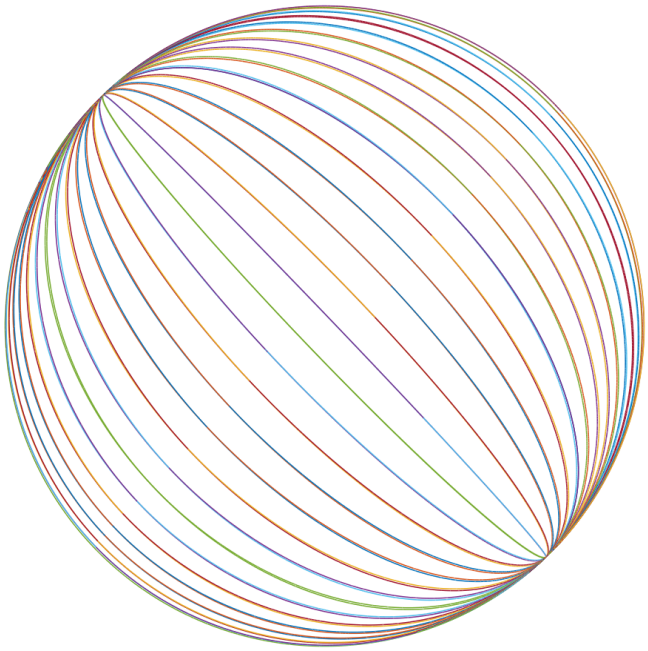
Geodesics

Mathematical Modelling



Vesna Trajcevska, Domen Vilar, Marcel Martinšek
Mentor: asist. dr. Peter Marijan Kink

University of Ljubljana
Faculty of Computer and Information Science
6th June 2020

1 Introduction

In euclidean space, the shortest path from point A to point B is a straight line, but in curved spaces, like surfaces in $\mathbb{R}^3$, there are no straight lines. The shortest path is actually a curve called a geodesic. The goal of this project was to find the geodesics on a 3-dimensional surface given with an implicit function and to write a function in Octave, which will numerically calculate and plot the geodesic in the given direction starting from the given point on a given surface.

2 Derivation of Equations

An implicit surface in $\mathbb{R}^3$ is given with an equation:

$$f(\mathbf{x}) = 0 \tag{1}$$

where x = (x, y, z), is a point that lays on the surface. So if the curve x(t) is a geodesic, then it must satisfy equation (1). By deriving this equation we get the following condition:

$$gradf(\mathbf{x}(t)) \cdot \mathbf{v}(t) = 0 \tag{2}$$

This tells us that the gradient vector and tangent vector in every point (x, y, z) on surface are perpendicular. Where $\mathbf{v}(t) = \dot{\mathbf{x}}(t) = (\dot{x}(t), \dot{y}(t), \dot{z}(t))$ is the tangent vector. We also define the normalised normal vector in point x

$$\mathbf{n}(\mathbf{x}) = \frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|} \tag{3}$$

where both *gradf* and n are row vectors. We can then find Q, a projection matrix of the normal plane and P a projection matrix of the tangent plane of the given curve.

$$Q = \mathbf{n}\mathbf{n}^\top, P = I - Q = I - \mathbf{n}\mathbf{n}^\top \tag{4}$$

If v is indeed a tangent vector in point x it should already be laying on the tangent plane, so multiplying it with P (the projection matrix of the tangent plane), returns the same vector.

$$\mathbf{v} = P\mathbf{v} \tag{5}$$

By deriving $\mathbf{v}$ by t we get the equation of a geodesic:

$$\dot{\mathbf{v}} = \dot{P}\mathbf{v} + P\dot{\mathbf{v}} \tag{6}$$

Geometrically a curve is considered a geodesic if the tangential acceleration is equal to zero. So the projection of the acceleration vector onto the tangent plane must equal zero:

$$P\dot{\mathbf{v}} = 0 \tag{7}$$

In physics this means that if an object is moving along the geodesic line it is accelerated only perpendicularly to surface.
We can now, from the equations (4),(6) and (7), see that:

$$\dot{\mathbf{v}} = \dot{P}\mathbf{v} = -\dot{\mathbf{Q}}\mathbf{v} \tag{8}$$

Which along with $\mathbf{v} = \dot{\mathbf{x}}$ create a system of 6 differential equations. Solving this system numerically allows us to find the geodesic starting in a given point x_0 in the direction of a given tangent vector v_0. However, even though equation (8) may seem simple we still need to find the derivative of Q = Q(x(t)) and express it only through derivatives of the function f, which defines our surface, and tangent vector v in point x. Here begins our first task which was to derive the next equation given in the instructions. We begin by writing Q as given in (4), and writing n as given in (3).

$$\mathbf{Q} = \mathbf{n}\mathbf{n}^\top = \frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|}\frac{gradf(\mathbf{x})^T}{\|gradf(\mathbf{x})\|} = \frac{gradf(\mathbf{x})gradf(\mathbf{x})^T}{\|gradf(\mathbf{x})\|^2}$$

Once we have this equation we can apply the quotient rule for derivatives.

$$\dot{\mathbf{Q}} = (\frac{gradf(\mathbf{x})gradf(\mathbf{x})^T}{\|gradf(\mathbf{x})\|^2})' =$$

$$\frac{(gradf(\mathbf{x})gradf(\mathbf{x})^T)'\|gradf(\mathbf{x})\|^2 - gradf(\mathbf{x})gradf(\mathbf{x})^T(\|gradf(\mathbf{x})\|^2)'}{\|gradf(\mathbf{x})\|^4}$$

From here by multiplying with the tangent vector we can simplify the equation, since $gradf(x)^T\mathbf{v} = gradf(\mathbf{x}(t)) \cdot \mathbf{v}(t) = 0$, meaning we can cut certain parts of the equation.

$$\dot{\mathbf{Q}}\mathbf{v} = \frac{(gradf(\mathbf{x})gradf(\mathbf{x})^T)'\|gradf(\mathbf{x})\|^2}{\|gradf(\mathbf{x})\|^4}\mathbf{v} - \frac{gradf(\mathbf{x})gradf(\mathbf{x})^T(\|gradf(\mathbf{x})\|^2)'}{\|gradf(\mathbf{x})\|^4}\mathbf{v}$$

$$gradf(\mathbf{x})^T\mathbf{v} = 0 \implies \frac{gradf(\mathbf{x})gradf(\mathbf{x})^T(\|gradf(\mathbf{x})\|^2)'}{\|gradf(\mathbf{x})\|^4}\mathbf{v} = 0$$

Next we apply the multiplication rule for derivatives which again divides the function into two components, one of which we can again cut seeing as condition (2) again applies.

$$\dot{\mathbf{Q}}\mathbf{v} = \frac{(gradf(\mathbf{x})gradf(\mathbf{x})^T)'}{\|gradf(\mathbf{x})\|^2}\mathbf{v} = \frac{((gradf(\mathbf{x}))'gradf(\mathbf{x})^T + gradf(\mathbf{x})(gradf(\mathbf{x})^T)')\mathbf{v}}{\|gradf(\mathbf{x})\|^2}$$

$$gradf(\mathbf{x})^T\mathbf{v} = 0 \implies (gradf(\mathbf{x}))'gradf(\mathbf{x})^T\mathbf{v} = 0$$

Lastly by using the chain rule we derive gradf(x).

$$\dot{\mathbf{Q}}\mathbf{v} = \frac{gradf(\mathbf{x})(gradf(\mathbf{x})^T)'}{\|gradf(\mathbf{x})\|^2}\mathbf{v} = \frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|^2}(\frac{\partial gradf(\mathbf{x})}{\partial x(t)}\frac{\partial x(t)}{\partial t})^T\mathbf{v}$$

The derivative of the gradient by x gives us the Hessian matrix. And the derivative of x by t is, as defined above, equal to v.

$$\frac{\partial gradf(\mathbf{x})}{\partial x(t)} = \begin{bmatrix} f_{xx} & f_{xy} & f_{xz} \\ f_{yx} & f_{yy} & f_{yz} \\ f_{zx} & f_{zy} & f_{zz} \end{bmatrix} = Hessf(x) \qquad \frac{\partial \mathbf{x}(t)}{\partial t} = \mathbf{v}$$

Once we input these derivatives back into the function and simply transpose we get the desired equation. The Hessian matrix is symmetric, seeing as for any twice differentiable function, partial derivatives commute, meaning $f_{xy} = f_{yx}$, so $Hessf(x)^T = Hessf(x)$

$$\dot{\mathbf{v}} = -\dot{\mathbf{Q}}\mathbf{v} = -\frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|^2}(Hessf(x)\mathbf{v})^T\mathbf{v} = -\frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|^2}\mathbf{v}^T Hessf(x)\mathbf{v}$$

3 Solution

3.1 geodesic.m

As we mentioned above our main goal in this project was to write a function that would calculate and plot a set of geodesics on a given 3D surface. With that intention in mind, we wrote `geodesic.m`. Given a point that lays on the surface and a direction vector, tangent to the surface, this function calculates geodesic curves from the starting point, rotating the direction vector between each calculation. The function, naturally also plots the curves, in the end the result is a set of "geodesic rays" stretching out from the starting point.

In this function, we first assemble the names of the gradient and hessian functions and initialize num_points, which represents the number of points on each of those geodesics respectively. If perpendicular circles to the geodesics are to be plotted, we also allocate space to store all the points of the geodesics. Next we calculate the normal vector to the tangent plane in the point x_0, project v_0 to the tangent plane and calculate the cross product $z = v_0 \times n$. These, along with the array of radian values theta, are going to be used for rotating the tangent vector around the gradient of the surface to create tangent vectors going in all directions from x_0.

We then have a for loop with i going from 1 to num_directions in which we have the process for creating each geodesic curve. First, we rotate v_0 theta(i)

radians counterclockwise in the tangent plane. We do this rotation by using the fact that z, v_0 and the rotated v are all on the tangential plane. Therefore, v can be expressed as a linear combination of v_0 and z:

$$v = \alpha v_0 + \beta z$$

Multiplying both sides with v_0 and knowing the dot product $z \cdot v_0 = 0$ we get

$$v_0 \cdot v = \alpha v_0^2 + \beta z \cdot v_0 = \alpha \cdot v_0^2 \tag{9}$$

Since v is just v_0 rotated, we know that $||v|| = ||v_0||$. With the definition of the dot product this gives us $v \cdot v_0 = ||v||^2 cos(\theta)$. This, combined with (9) gives us $\alpha = cos(\theta)$. Using the fact that $||z|| = ||v||$ it can easily be shown that $\beta = sin(\theta)$, bringing us to the final equation for the vector v_0 rotated by θ :

$$v = cos(\theta) \cdot v_0 + sin(\theta) \cdot z \tag{10}$$

Next we initialize a vector y to have the form $(x_1, x_2, x_3, v_1, v_2, v_3)^T$ and pass it, along with the other necessary parameters to our implementation of the 4th order Runge Kutta method (explained in more detail below). After we get all the points on the geodesic curve in the variable y, we can plot a new geodesic curve using a call to plot3() and "hold on". If the perpendicular circles to the geodesics are to be plotted, we also store all the components x,y and z of the calculated points into the i-th column of X, Y and Z respectively. After the loop stops, the only thing left to do is plot the aforementioned perpendicular circles if the input argument circles is equal to 1. At this point the X, Y and Z matrices have all the components of the plotted geodesics stored so that the j-th column of X has all the x components of the points of the j-th geodesic. if we just plot a curve by passing a vector of all the i-th points of the geodesics we'll end up with num_points curves connecting all the i-th points forming warped circles on the surface around the starting point x_0.

Listing 1: geodesic.m

```
function geodesic(x0,v0,t0,t1,h,object,num_directions=101,circles=0)
  %x0 - starting point
  %v0 - tangent vector at x0
  %t0,t1 - start and end of interval on which we will calculate points
  %h - length of a single step
  %object - name of surface
  %cirles - flag to set display of perpendicular circles
  %num_directions - number of directions to calculate geodesics in

  num_points = ceil((t1 - t0)/h); %num of points for each geodesic
  grad = [object,'gradf']; %name of gradient function for object
  hess = [object,'hess']; %name of hessian function for object
  addpath("./shapes");
```

```
  if (circles) %initializations of storage for geodesics
    X = zeros(num_points, num_directions);
    Y = zeros(num_points, num_directions);
    Z = zeros(num_points, num_directions);
  endif

  n = feval(grad, x0); %calculate normal vector in point x0
  v0 = project(x0, v0, grad); %projecting v0 in case it wasnt tangential
  z = cross(v0, n)/norm(n); %normalized cross product of v0 and n
  theta = linspace(0, 2*pi, num_directions);

  for i=1:num_directions %calculating num_direction geodesics
    v = cos(theta(i))*v0 + sin(theta(i))*z; %rotating by theta(i) radians
    y = [x0; v];
    y = rk4(y, num_points, h, grad, hess); %callculate one geodesic

    %for the circles
    if (circles)
      X(:,i) = y(1,:);
      Y(:,i) = y(2,:);
      Z(:,i) = y(3,:);
    endif

    plot3(y(1,:), y(2,:), y(3,:)); %plot geodesic on figure
    hold on
  end

  %for the circles
  if (circles)
    %surf(X,Y,Z);
    for i=1:15:num_points
      plot3(X(i,:), Y(i,:), Z(i,:));
    end
  endif
  axis equal;
endfunction
```

We used the `geodesic.m` function on numerous implicitly given functions. At the end of the document is a gallery of all the images our `geodesic.m` function plotted.

This function though, can't work independently. It relies on functions that calculate the gradient, Hessian, and projection matrix of the tangent plane for each surface. We wrote these functions for a few surfaces (sphere, torus etc.). But most notably it relies on a function called rk4 or the Runge Kutta method.

3.2 4th order Runge Kutta

In purely mathematical terms our rk4 numerically solves the system:

$$\dot{\mathbf{x}} = \mathbf{v}$$
$$\dot{\mathbf{v}} = -\frac{gradf(\mathbf{x})}{\|gradf(\mathbf{x})\|^2}\mathbf{v}^T Hessf(x)\mathbf{v}$$

Going forward, we will denote $\dot{v}$ as $a(x_i, v_i)$ and points and vectors that refer to the geodesic as capital letters (X and V).

The Runge Kutta method of 4th order in our case calculates the weighted average velocity $v_{avg} = h(v_1 + 2v_2 + 2v_3 + v_4)/6$ and the weighted average acceleration $a_{avg} = h(a_1 + 2a_2 + 2a_3 + a_4)/6$ vectors of the curve in 4 points denoted as x_1, x_2, x_3, x_4. The next point X_{i+1} and velocity vector V_{i+1} on the geodesic are calculated by adding these averages to the initial point X_i and velocity vector V_i.

The initial point x_1 on the surface and the slope v_1 are the given X_i and V_i. The sum of the tangent vector v_1 and the acceleration a_i tells us in which direction we move from point x_1 to get the point x_{i+1}.

More precisely:

1. In the given initial point $x_1 = X_i$ in which the tangent vector is, $v_1 = V_i$ we calculate the acceleration vector $a_1 = a(x_1, v_1)$. After we have both a_1 and v_1 we scale them by h.

2. We calculate the next point $x_2 = X_i + \frac{1}{2}v_1$ and in it we calculate the tangent vector $v_2 = V_i + \frac{1}{2}a_1$ and $a_2 = a(x_2, v_2)$, same as previously we scale them by h once calculated.

3. In the same way we calculate $x_3 = X_i + \frac{1}{2}v_2$ and in it we calculate the tangent vector $v_3 = V_i + \frac{1}{2}a_2$ and $a_2 = a(x_3, v_3)$, scaling by h afterwards.

4. Lastly we calculate $x_4 = X_i + v_3$, in it the tangent vector $v_4 = V_i + a_3$ and $a_4 = a(x_4, v_4)$, as always scaling by h after all calculations are done.

Once we have all four slopes v_1, v_2, v_3, v_4 we calculate the average slope v_{avg} and take a step in it's direction from the starting point X_i. This gives us the next approximated point X_{i+1} on the geodesic. Similarly we calculate the average velocity a_{avg}. The tangent vector V_{i+1} in the point X_{i+1} is the sum of V_i and the acceleration vector a_{avg}. Meaning the next point and tangent vector on the geodesic are calculated as follows:

$$X_{i+1} = X_i + v_{avg} \qquad V_{i+1} = V_i + a_{avg}$$

Our `rk4.m` function is based on the above explanation. And as a result it returns an array of estimated points on the geodesic starting at the initial point in the direction of the given vector.

```
function y = rk4(y0, n, h, gradf, hess)
  %y0 - [starting point; tangent vector]
  %n - num of points to be found
  %h - length of a single step
  %gradf,hess - gradient and hessian functions

  y = zeros(length(y0), n); %allocate space for n points
  y(:,1) = y0; %set first point
  for i=1:n-1 %calclating n-1 points with 4th ord. Runge Kutta
    k1 = h*equat( y(:,i) ,gradf,hess);
    k2 = h*equat( y(:,i)+k1/2 ,gradf,hess);
    k3 = h*equat( y(:,i)+k2/2 ,gradf,hess);
    k4 = h*equat( y(:,i)+k3 ,gradf,hess);

    %calculating next point approximation
    y(:, i+1) = y(:, i) + (k1+ 2*k2 + 2*k3 + k4)/6;
  end
endfunction

%function to calculate the tangent vector and acceleration in point x
function rez = equat(y,gradf,hess)
  x = y(1:3);
  v = y(4:6);
  grad = feval(gradf,x); %gradient in point x
  ngrad = grad'*grad; %||grad||^2
  a = -(grad/ngrad) * v' * feval(hess,x) * v; %dv(x)/dx = a(x,v)
  rez = [v; a];
endfunction
```

3.3 Results and conclusions

So now that everything is defined and coded it's time to see if we've managed to plot some geodesics. For a start we tried this with one of the simplest and most basic implicit surfaces we could think of - a sphere. After all the troubleshooting was done the results were very pleasing. Below, there are photos of a 101 geodesics plotted on a sphere with it's center at the origin point and radius 2. We have also plotted perpendicular circles to create a grid so the shape can be more clearly seen. Generally, these photos plainly show the geodesics of the sphere which look much like the meridians on Earth. It's interesting to note that when looking at these lines head on they look straight, and one can imagine standing on top of a ball similar to this one and walking a "straight" path along one of it's geodesics.

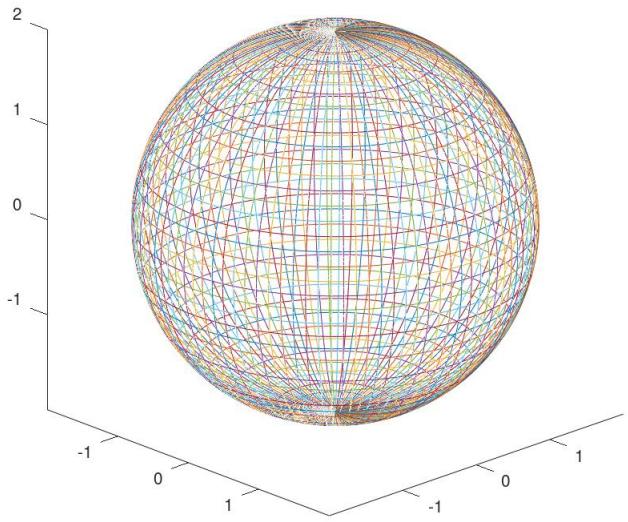


Figure 1: Geodesic curves on a sphere

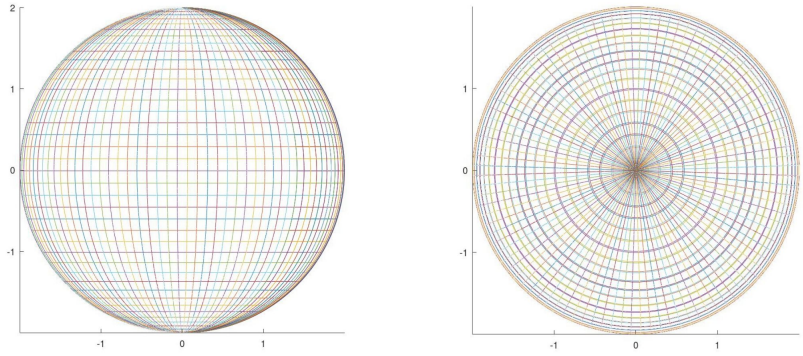


Figure 2: The side and top view of the sphere.

After we were assured that our function works, at least with a sphere, we wrote in functions for other, more interesting implicit surfaces. Below are images of geodesics on part of a torus plotted by calling the function `geodesic([3;0;1],[1;1;0],0,3.2,0.01,'torus', 101, 1)`. In this case we chose to represent only part of the torus because, if the geodesics are set to be

long enough to define the whole figure, they intertwine, and the image becomes messy, such images can be seen in the gallery below.

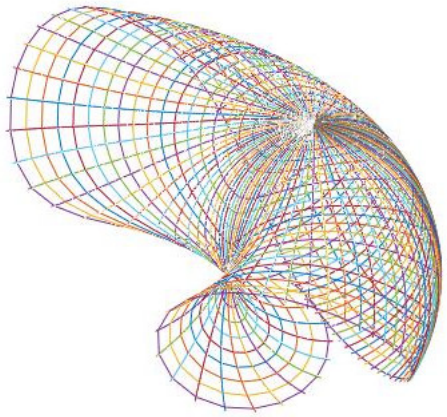


Figure 3: Geodesic curves on part of a torus

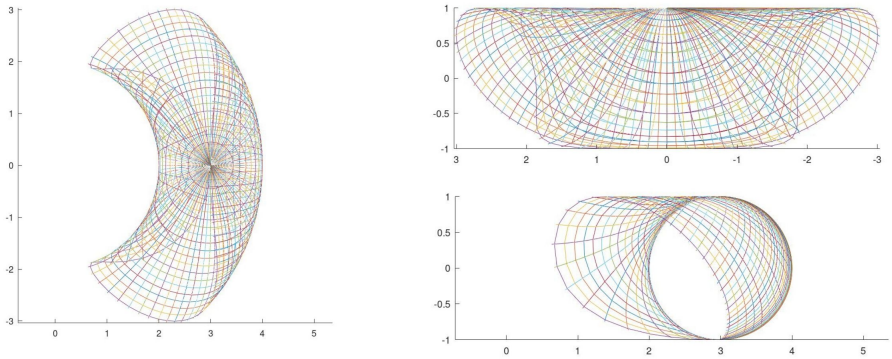


Figure 4: The top(left), front(upper right) and side(lower right) view of the torus.

Instead, to better represent the torus we thought it might be a wise choice to plot only one very long geodesic line. We edited the number of geodesics, so that by calling the function `geodesic([3;0;1], [1;1;0], 0, 1000, 0.01, 'torus',1)` we get the results as shown below. One long curve that goes around the torus, surpris-

ingly never reaching it's middle. Seeing as the results were quite interesting we decided to do the same thing with an ellipsoid, calling the function `geodesic([0;0;2],[1;1;0],0,2000,0.01,'ellipsoid',1)`, the results of this can be seen in the gallery at the end of the report.

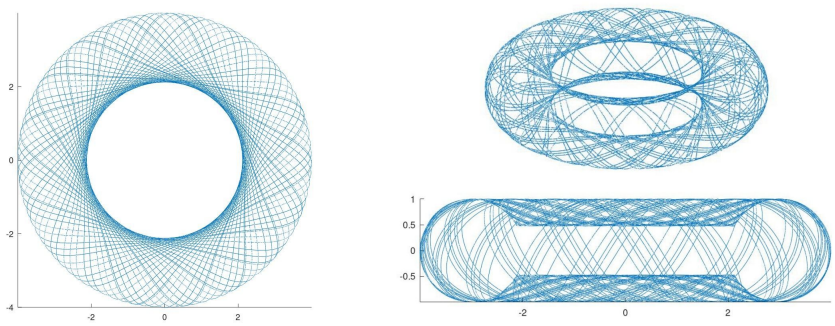


Figure 5: The top(left), angled(upper right) and side(lower right) view of one geodesic on the torus.

Although the results we had seen were promising, we had no concrete evidence that our function was accurate. To prove the accuracy of our geodesics, we used the built in sphere() and surf() functions to plot a sphere with it's center at [0,0,0] and a radius of 2 and then called our function with `geodesic([1;1;sqrt(2)],[1;1;0],0,2*pi,0.01,'sphere')` to draw over it. To add to the demonstration, we inputted a point at an angled position and not right on top of the sphere and a direction vector not exactly tangent to the surface. We can do this, because we project our starting vector onto the tangential plane in the starting point(and normalize it) before we start our calculations, so it doesn't matter if the vector isn't exactly tangent to the surface, as long as it is not collinear to the gradient vector. The result(6) was, as expected, a representation of 101 geodesics, starting from the point $[1,1,\sqrt{2}]$, properly fitting the sphere plotted by surf(). For a clearer image, we set the sphere's color to gray.

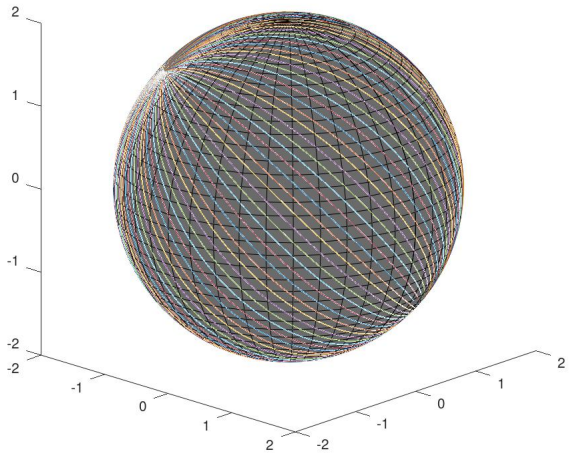


Figure 6: Sphere and 101 geodesics on it

We took the same approach with the ellipsoid and used the built in ellipsoid() function along with surf() to plot an ellipsoid with its center at [0,0,0] and the semi-major axis lengths xr=0.5, yr=1 and zr=2. The results (7) below were achieved by calling our function with `geodesic([0;0;4],[1;0;0],0,10,0.01,'ellipsoid')`.

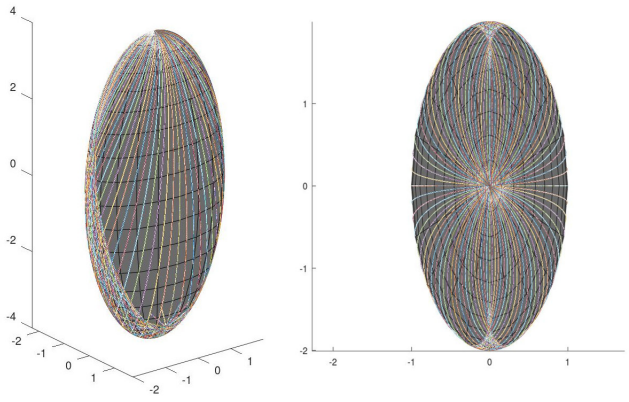


Figure 7: Angled(left) and top-down(right) view of the geodesics on an ellipsoid

We noticed an interesting detail when viewing the results from the top down view. The geodesics we've plotted don't seem to match the ellipsoid plotted by the implemented function. But we believe that in fact, it is the ellipsoid() function that's inaccurate in this case, as the mesh-grid returned by ellipsoid() does not in fact plot a smooth surface, instead patching together small plains that can only approximate the shape. Whereas our geodesics stick more closely to the mathematical formula because it calculates many points and the steps between them are much smaller.

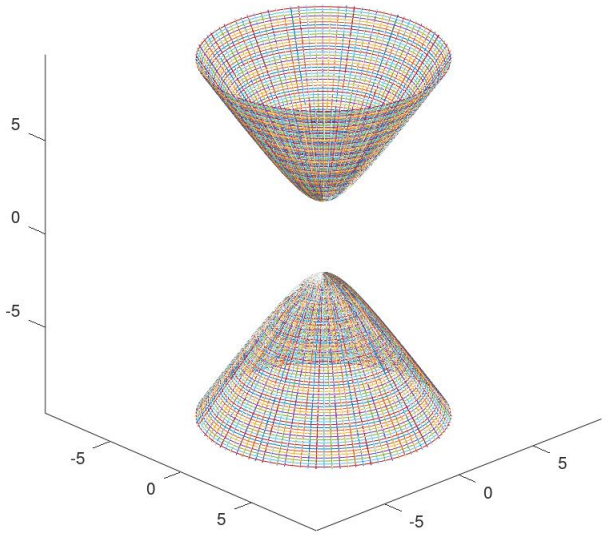


Figure 8: Hyperboloid

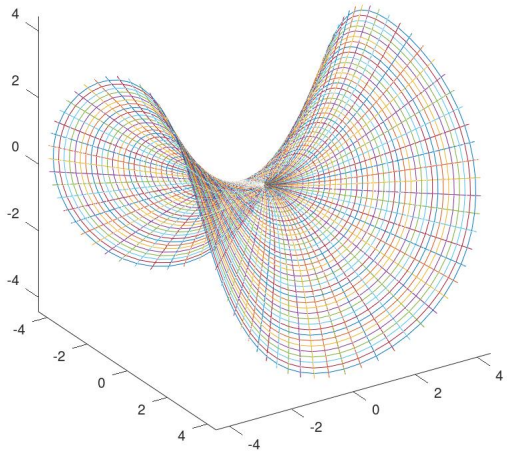


Figure 9: Hyperbolic paraboloid

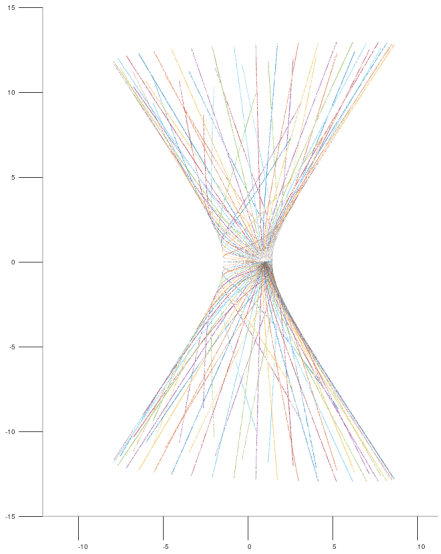


Figure 10: Hyperboloid

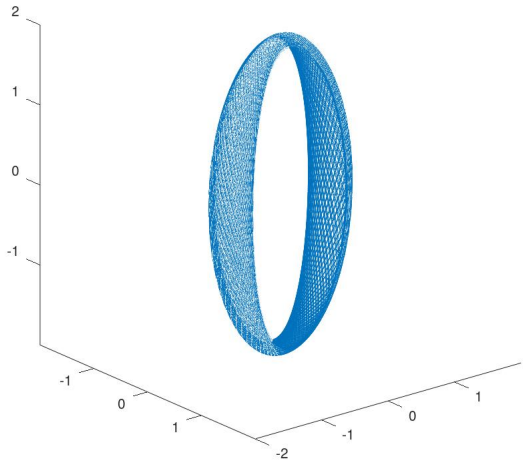


Figure 11: Ellipsoid - one long geodesic

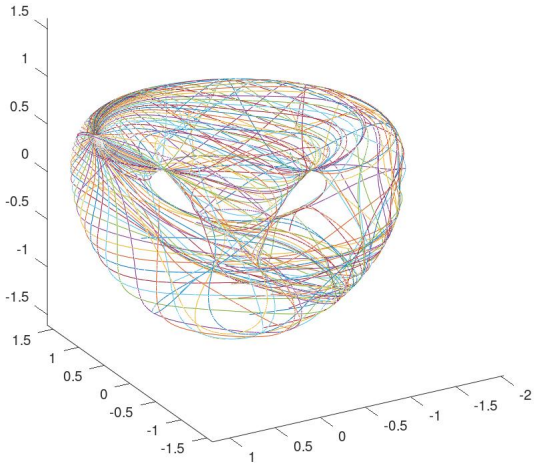


Figure 12: Genus

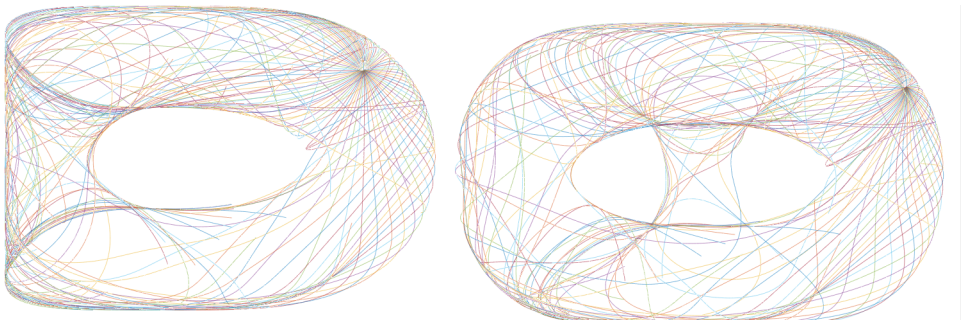


Figure 13: Genus from side view

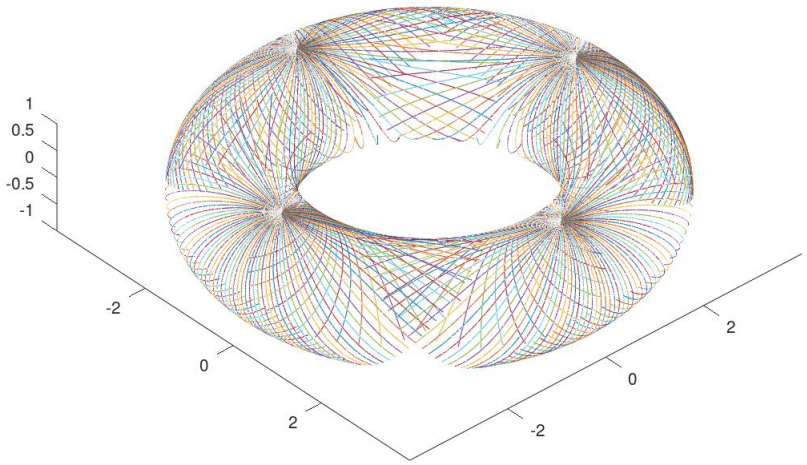


Figure 14: Geodesics from 4 points on torus

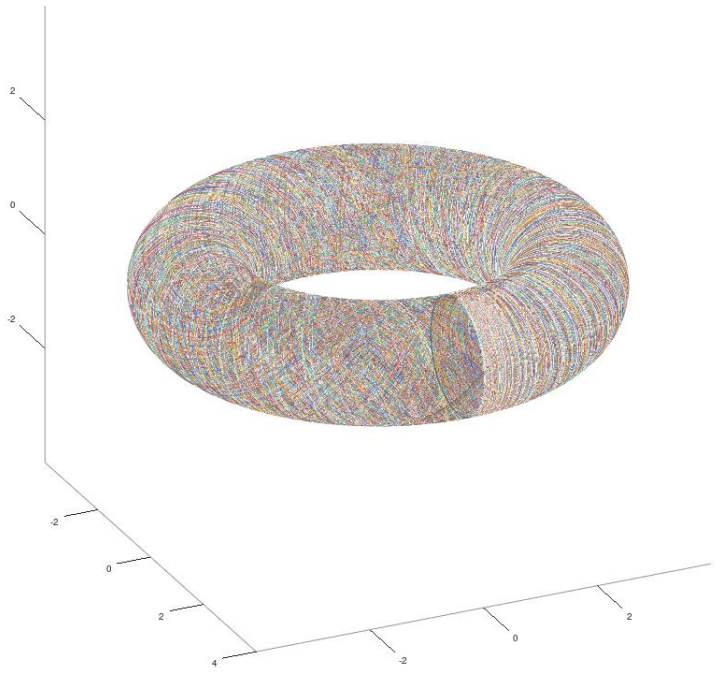


Figure 15: Goedesics on whole torus

3.4 Working in a team

We tackled the problem together as a team and divided work amongst all members of the group. Vesna Trajcevska did the theoretical part of the project, deriving the equations we needed for the Runge-Kutta method and thoroughly explained them in the report. Domen Vilar wrote the `rk4.m` function needed for numerically calculating geodesics. And lastly Marcel Martinšek wrote `geodesic.m` function that plots the geodesics for any given point and direction as long it lays on the surface. Marcel also wrote functions that return the Hessian matrix and gradf(x) needed for the Runge-Kutta iteration for chosen implicit surfaces. Our group ultimately worked together very well, we collaborated on everything, no one was ever really working on their own. So, although the work was divided fairly everyone still had a hand in everything, this way we ensured that all members of the group thoroughly understand each part of the project.