Numerical differential geometry in Matlab

John Kerl

Department of Mathematics, University of Arizona

Graduate Student Colloquium

January 16, 2008

Topics

- Why bother?
- 1D and 2D meshes and numerical differentiation
- Coordinates and parameterizations
- Surfaces embedded in $\mathbb{R}^3$
- Numerical estimation of tangent and unit-normal vectors
- Metric coefficients
- Christoffel symbols, curvature tensors, scalar curvature . . .

I care not so much about how far we get today (I can finish in a part-two talk, or you can read the rest of the Matlab code yourself), and more about identifying the essential concepts you need to translate between pure math and numerical methods.

Why?

The soul cannot think without a picture.
— Aristotle (384-322 B.C.).

Level-curve problem (Jan. 2001 geometry qual problem #3):

Let $g : \mathbb{S}^2 \to \mathbb{R} : (x, y, z) \mapsto y^2 - z$. *Determine the critical points, the critical values, and qualitatively describe the level sets.*

Solution

Use the method of Lagrange multipliers with constraint curve $f(x, y, z) = x^2 + y^2 + z^2 = 1$. There are four critical points:

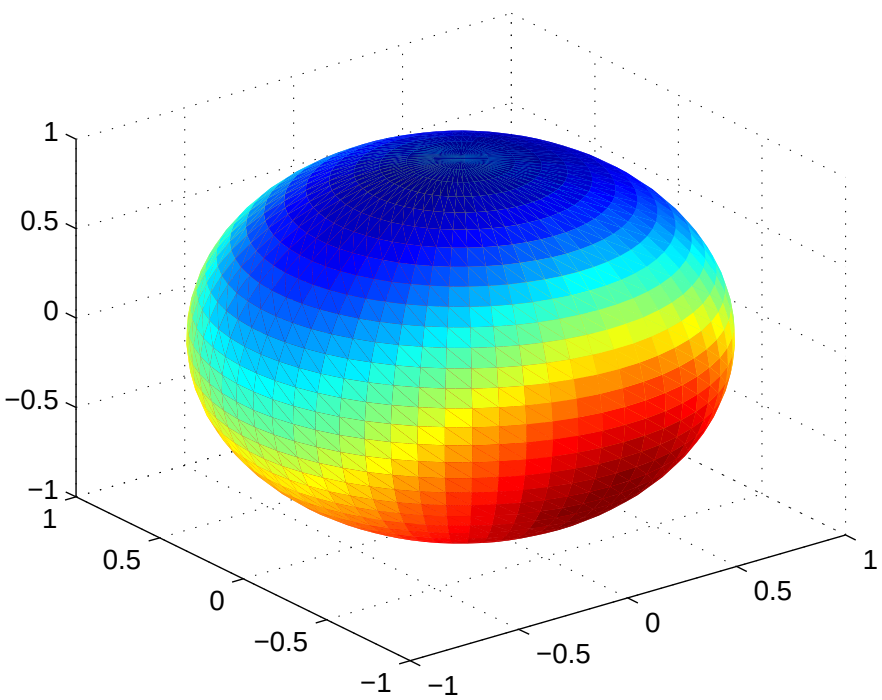
$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}, \begin{pmatrix} 0 \\ \sqrt{3}/2 \\ -1/2 \end{pmatrix}, \quad \text{and} \quad \begin{pmatrix} 0 \\ -\sqrt{3}/2 \\ -1/2 \end{pmatrix},$$

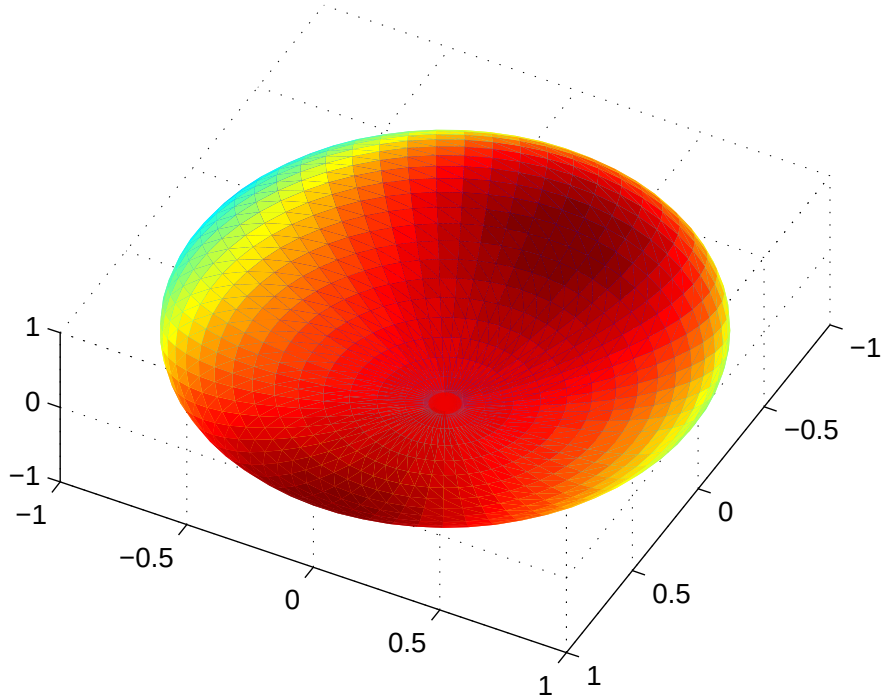
The critical values are found by $y^2 - z$. We have

$$-1, 1, 5/4, \quad \text{and} \quad 5/4,$$

respectively. Level sets take a bit more work. (See `http://math.arizona.edu/~kerl/doc/prolrevqual.pdf`.)

Here's a pair of Matlab plots of g on the sphere. (See `y2z.m` in the directory where you found this file.) The first plot is a top view (positive z is up); the second plot is a bottom view (negative z is up). The level curves are simply isotherms; you can read off their topologies at a glance: point, loop, figure eight, pair of loops, pair of points. This doesn't replace the explicit computations . . . but it certainly adds another perspective which I want to be able to see.





1D mesh

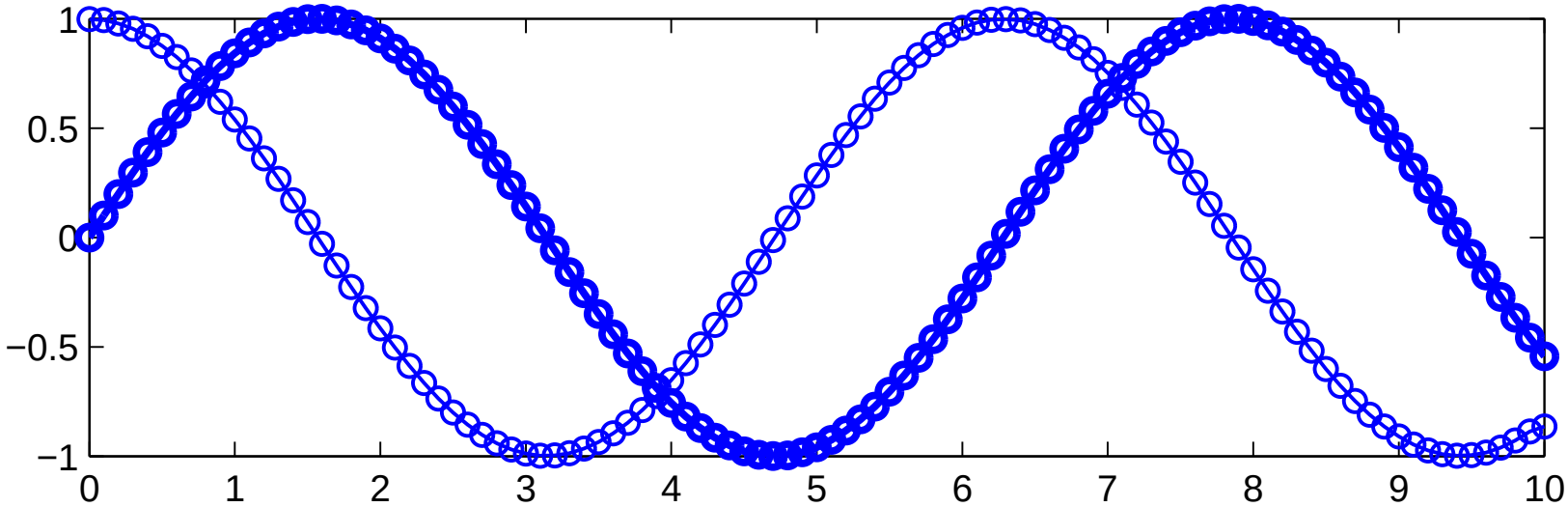
```
dx = 0.1; x = [0: dx: 10];
y = sin(x);
y = sin(x)  + cos(x); plot(x,y)
y = sin(x) .* cos(x); plot(x,y)
```

Key points:

- Multiple input and output points are contained in *vectors*. (This is not the way we use vectors in linear algebra.)
- Matlab supports vector operations, e.g. `y=sin(x)`.
- Arrays are added componentwise, but the `*` symbol means matrix multiplication (which isn't even defined if the dimensions aren't compatible). Componentwise multiplication, exponentiation, etc. are done using `.*`, `.^`, etc.

1D numerical differentiation

```
y = sin(x);
plot(x, y, '-o', 'LineWidth', 2);

yp = gradient(y, dx);
hold on
plot(x, yp, '-o', 'LineWidth', 1);
```



1D numerical differentiation (continued)

How would you numerically estimate a derivative?

- Forward difference: $\frac{f(x+h)-f(x)}{h}$
- Backward difference: $\frac{f(x)-f(x-h)}{h}$
- Centered difference: $\frac{f(x+h)-f(x-h)}{2h}$

1D numerical differentiation (continued)

Matlab does:

- forward differences on the left edge,
- backward differences on the right edge,
- centered differences in the middle.

The numerical estimates do not exactly match the analytical results (indeed, they match rather poorly here since $h = 1$):

```
>> x=[1:5]
x   =  1     2     3     4     5
>> y = x.^2
y   =  1     4     9    16    25
>> gradient(y, 1)
ans =  3     4     6     8     9
```

1D numerical differentiation (continued)

What we learn (or teach) in calculus class: accuracy of numerical estimates increases with smaller mesh size (h or `dx`) *if* arbitrary precision is available.

What we don't learn until numerical analysis *if* we ever take/teach it at all (!): subtraction of nearly equal numbers — e.g. $f(x+h) - f(x)$ for continuous f — results in loss of significant digits. You *cannot* just try to "take the limit" as h goes to zero. You have to stop sooner.

There is a trade-off: smaller h increases ideal accuracy but worsens cancellation error.

Example: base-10 arithmetic with 9 significant digits (cf. base-2 with 53 significant digits for double precision). Inputs have 9 significant digits; output has only one:

$$1.23456789 - 1.23456788 = 0.00000001$$

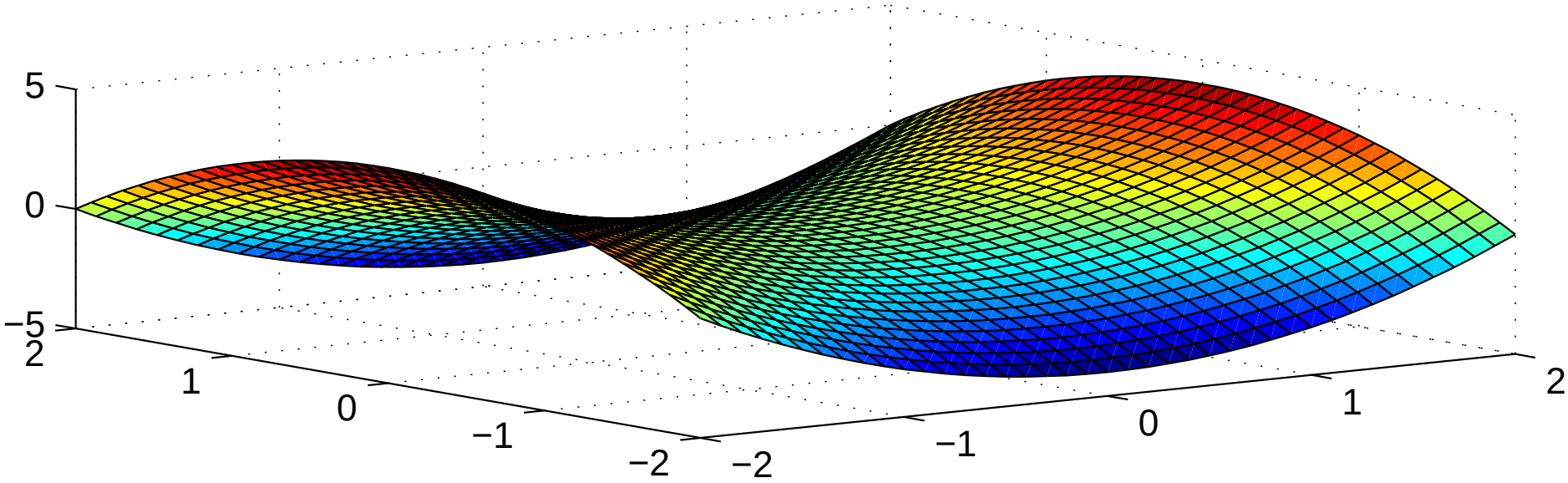
Numerical analysis is a non-trivial subject, but a rule of thumb (not important for this talk since I'm using very coarse meshes) is to keep h greater than square root of machine epsilon (10^{-9} for double precision).

2D meshgrid

```
>> [x,y]=meshgrid(1:4,5:9)
x =  1     2     3     4  |  y = 5     5     5     5
     1     2     3     4  |       6     6     6     6
     1     2     3     4  |       7     7     7     7
     1     2     3     4  |       8     8     8     8
     1     2     3     4  |       9     9     9     9
                          |
>> z = x + y              | >> z = x.^2 .* y
z =  6     7     8     9  | z =  5    20    45    80
     7     8     9    10  |       6    24    54    96
     8     9    10    11  |       7    28    63   112
     9    10    11    12  |       8    32    72   128
    10    11    12    13  |       9    36    81   144
```

Surface plots

```
dx = 0.1; dy = 0.1;
[x,y] = meshgrid(-2:dx:2,-2:dy:2);
z = x.^2 - y.^2;
surf(x,y,z);
```



2D numerical differentiation

$$\begin{aligned} \frac{\partial f}{\partial x} &\approx \frac{f(x+h,y)-f(x-h,y)}{2h} \\ \frac{\partial f}{\partial y} &\approx \frac{f(x,y+h)-f(x,y-h)}{2h}. \end{aligned}$$

```
>> [dzdx,dzdy]=gradient(z,dx,dy)
dzdx = 15   20   30   35   |  dzdy =  1   4   9   16
       18   24   36   42   |          1   4   9   16
       21   28   42   49   |          1   4   9   16
       24   32   48   56   |          1   4   9   16
       27   36   54   63   |          1   4   9   16
```

Again, numerical estimates don't exactly match analytical results.

Dual roles for vectors and matrices

Purely, we tend to think of matrices as implementing linear transformations.

Here, a matrix is an $M \times N$ container of sample points — just as in the 1D case, a vector served as a container of sample points.

Given $z = f(x, y)$, e.g. $z = x^2 - y^2$, the x, y, and z are 2D arrays. Think of this z as a *discretely sampled scalar field* on the plane.

Likewise

$$u = f(x, y), \quad v = g(x, y), \quad w = h(x, y)$$

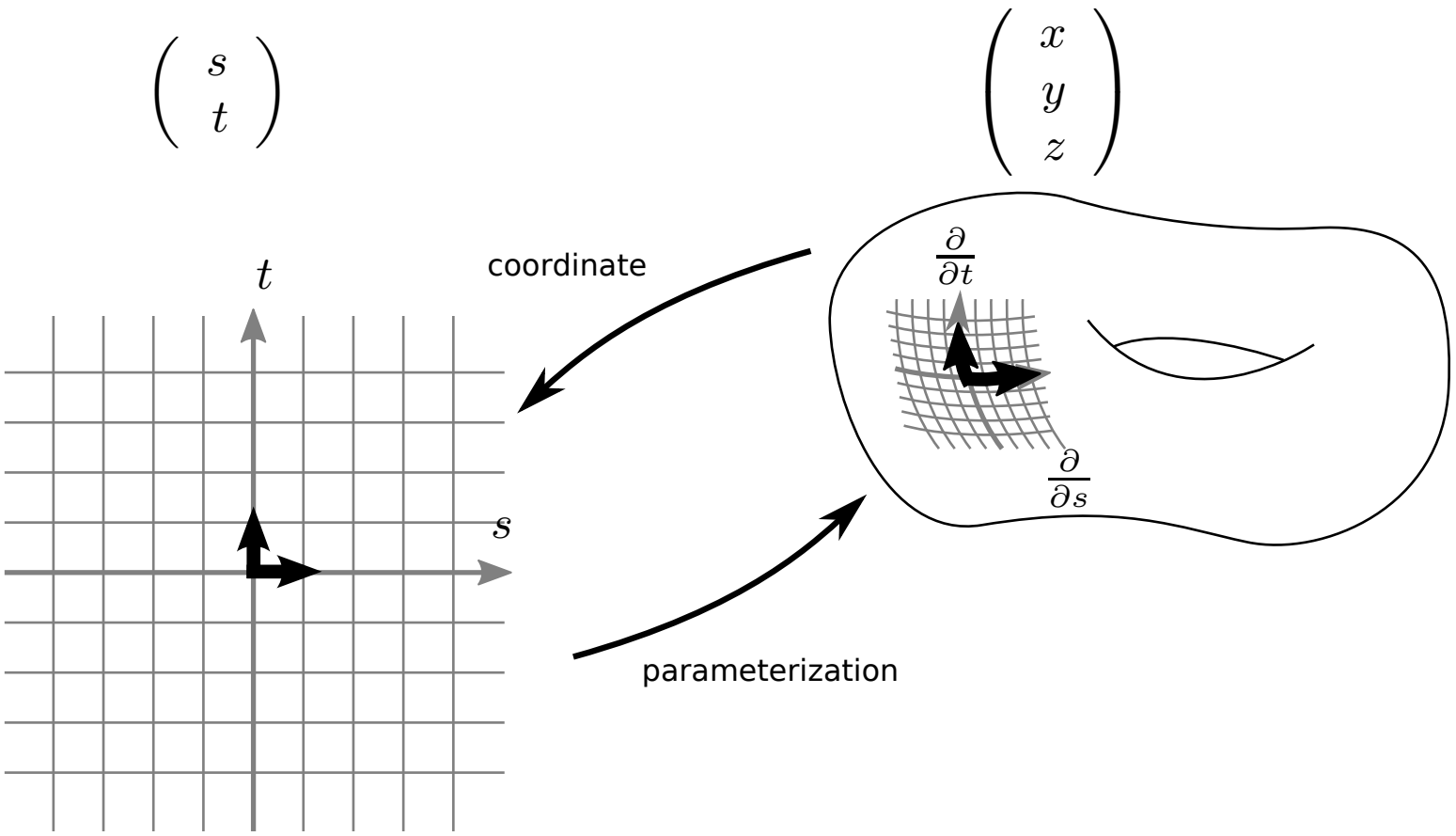
would be a discretely sampled vector field.

In Matlab, you can loop over array indices, or you can have the looping done for you. Furthermore, that implicit looping can be done on the entire array (e.g. `z=x+y`), or on slices (using the colon).

Examples:

```
>> x
x =   1     2     3     4
      1     2     3     4
      1     2     3     4
      1     2     3     4
      1     2     3     4

>> x(1,1)=400; x(2,:)=77; x(:,3)=999
x = 400     2   999     4
     77    77   999    77
      1     2   999     4
      1     2   999     4
      1     2   999     4
```

Coordinates and parameterizations



Surfaces embedded in $\mathbb{R}^3$

Parameterization:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{pmatrix}$$

Coordinates:

- Graph coordinates: x and y.
- Surface of revolution: θ and y.
- Cylinder: r and z.
- Spherical/toroidal: ϕ and θ.

Generically (in `gdgproj.m`) think of these as s and t.

For the hyperboloid of one sheet, use $s = z$ and $t = \theta$ coordinates. Find the hyperbola in r and z; form a surface of revolution by angle θ about the z axis.

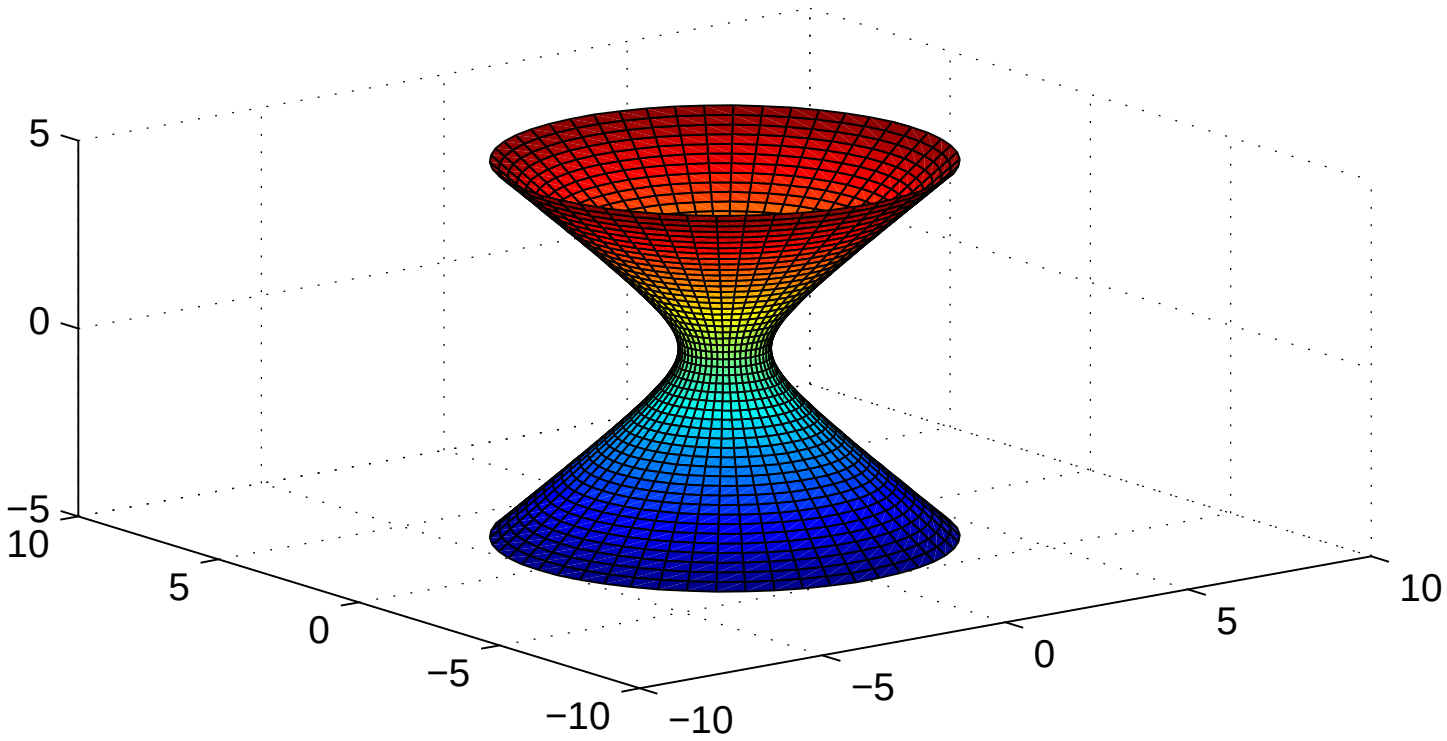
$$x^2 + y^2 - z^2 = 1$$
$$r^2 - z^2 = 1$$
$$r = \sqrt{1 + z^2}$$

Doing that in Matlab

```
zmin     =  -5.0; zmax     =   5.0;
nz       =  50;   dz       = (zmax-zmin)/nz;

thetamin =   0.0; thetamax =   2.0*pi;
ntheta   =  50;   dtheta   = (thetamax-thetamin)/ntheta;

[paramz,theta] = meshgrid(zmin:dz:zmax, ...
    thetamin:dtheta:thetamax+dtheta);

% Apply the parameterization to obtain R^3 coordinates
r = sqrt(1 + paramz.^2);
x = r .* cos(theta);
y = r .* sin(theta);
z = paramz;
```

Doing that in Matlab (continued)

```
surf(x,y,z)
```



Numerical estimation of tangent vectors

$$\left.\frac{\partial x}{\partial s}\right|_{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} \approx \frac{x(s+h,t) - x(s-h,t)}{2h}$$

$$\left.\frac{\partial y}{\partial s}\right|_{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} \approx \frac{y(s+h,t) - y(s-h,t)}{2h}$$

$$\left.\frac{\partial z}{\partial s}\right|_{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} \approx \frac{z(s+h,t) - z(s-h,t)}{2h}$$

So,

$$\left.\frac{\partial}{\partial s}\right|_{\begin{pmatrix} x \\ y \\ z \end{pmatrix}} = \left.\begin{pmatrix} \partial x/\partial s \\ \partial y/\partial s \\ \partial z/\partial s \end{pmatrix}\right|_{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}$$

$$\approx \frac{1}{2h}\left[\begin{pmatrix} x(s+h,t) \\ y(s+h,t) \\ z(s+h,t) \end{pmatrix} - \begin{pmatrix} x(s,t) \\ y(s,t) \\ z(s,t) \end{pmatrix}\right]$$

Unit-normal vectors

$$\mathbf{n} = \frac{\partial}{\partial s} \times \frac{\partial}{\partial t}$$

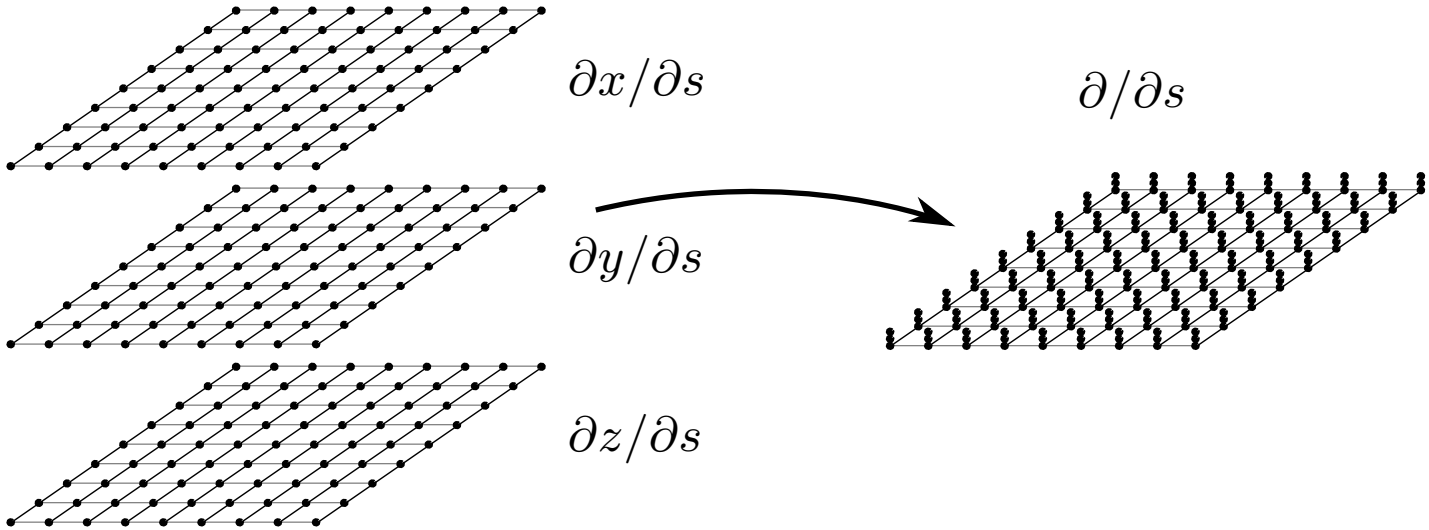
$$\hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|}$$

Matlab:

```
N1    = cross(DDs1, DDt1);
N1hat = N1 / norm(N1);
```

One problem

I had the x, y, and z coordinates of a surface as three separate 2D arrays. Likewise, the partials $\partial x/\partial s$, $\partial y/\partial s$, $\partial z/\partial s$, $\partial x/\partial t$, $\partial y/\partial t$, and $\partial z/\partial t$ are six separate 2D arrays.



Solution

Snippet from `gdgproj.m`:

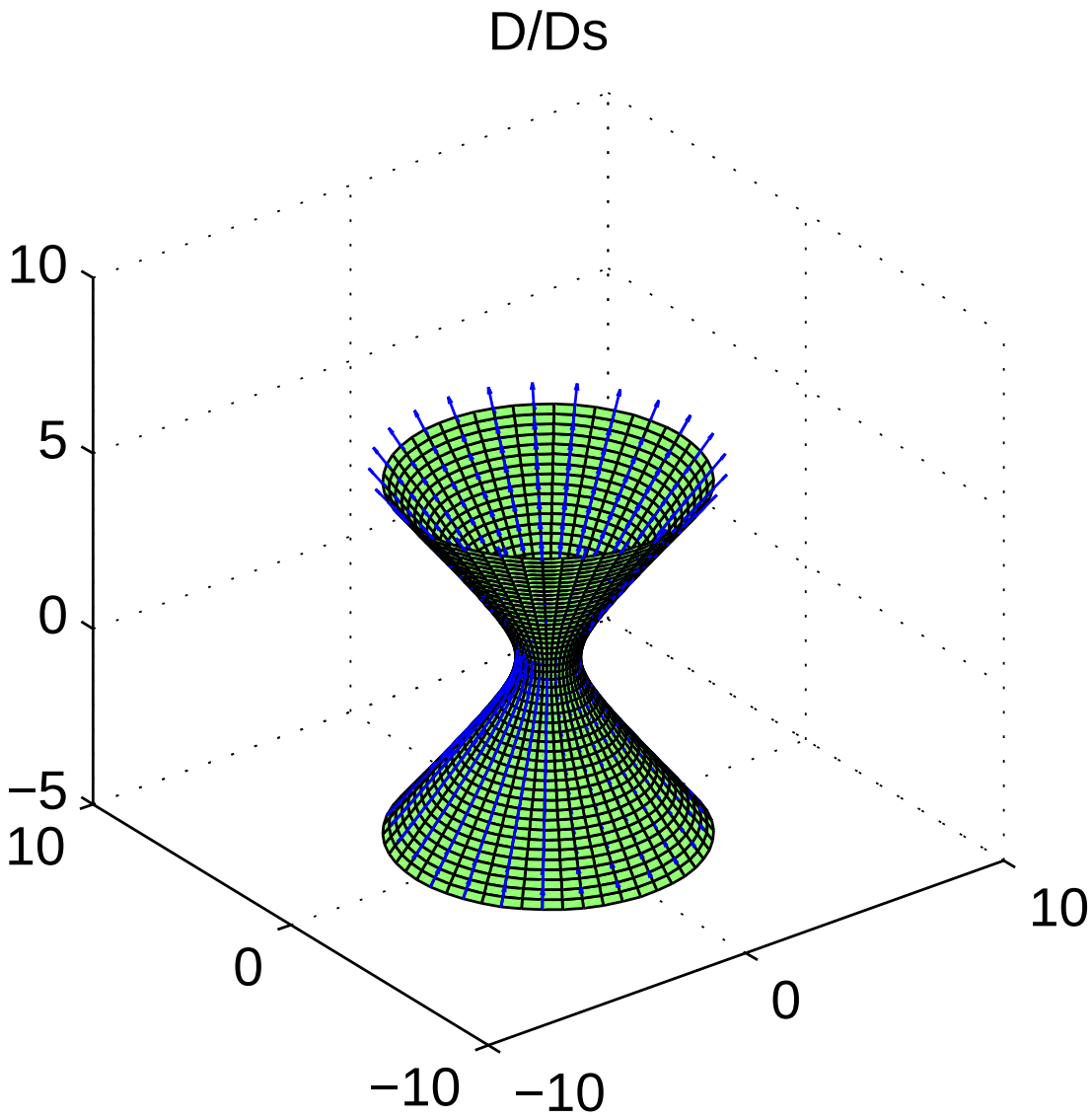
```
[DxDs,DxDt] = gradient(x,ds,dt); % Three scalar fields
[DyDs,DyDt] = gradient(y,ds,dt);
[DzDs,DzDt] = gradient(z,ds,dt);
DDs1 = [0 0 0]; DDt1 = [0 0 0]; % Single vectors

% Allocate space for vector fields.
% Put the s and t indices first so that
% size(DDs(:,:,3) is ns x nt.
DDs  = zeros(ns, nt, 3);
DDt  = zeros(ns, nt, 3);
Nhat = zeros(ns, nt, 3);
% continued on next slide ...
```

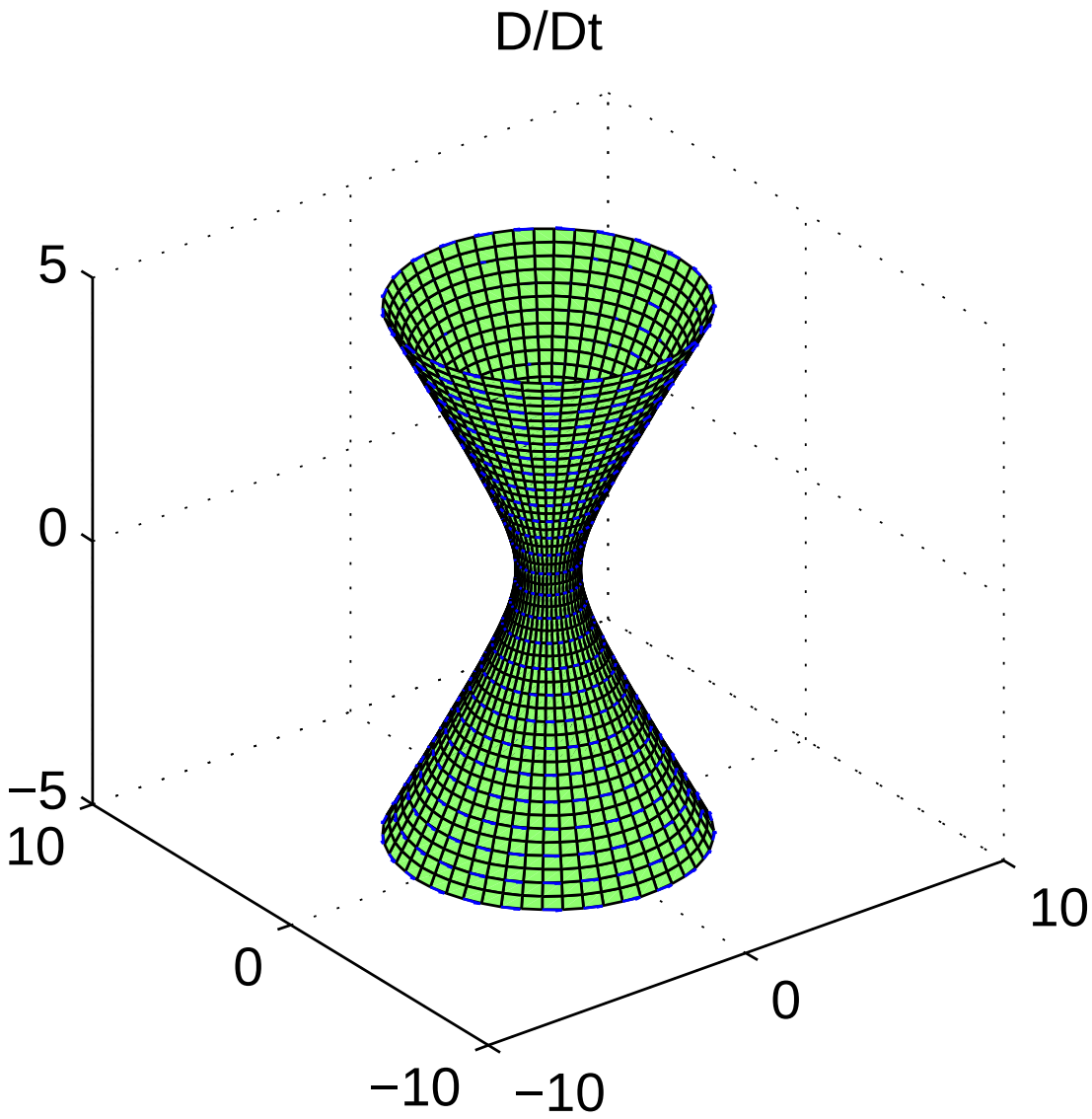
```
% ... continued from next slide.
% Here is the one place in this file where I loop over
% s and t meshgrid indices.
for ss = 1:ns
    for tt = 1:nt
        DDs1(1) = DxDs(ss,tt); DDt1(1) = DxDt(ss,tt);
        DDs1(2) = DyDs(ss,tt); DDt1(2) = DyDt(ss,tt);
        DDs1(3) = DzDs(ss,tt); DDt1(3) = DzDt(ss,tt);

        N1 = cross(DDs1, DDt1); N1hat  = N1 / norm(N1);

        % Store the tangent basis and unit normals for later.
        DDs (ss,tt,:) = DDs1; % Now a vector field
        DDt (ss,tt,:) = DDt1;
        Nhat(ss,tt,:) = N1hat;
    end
end
```

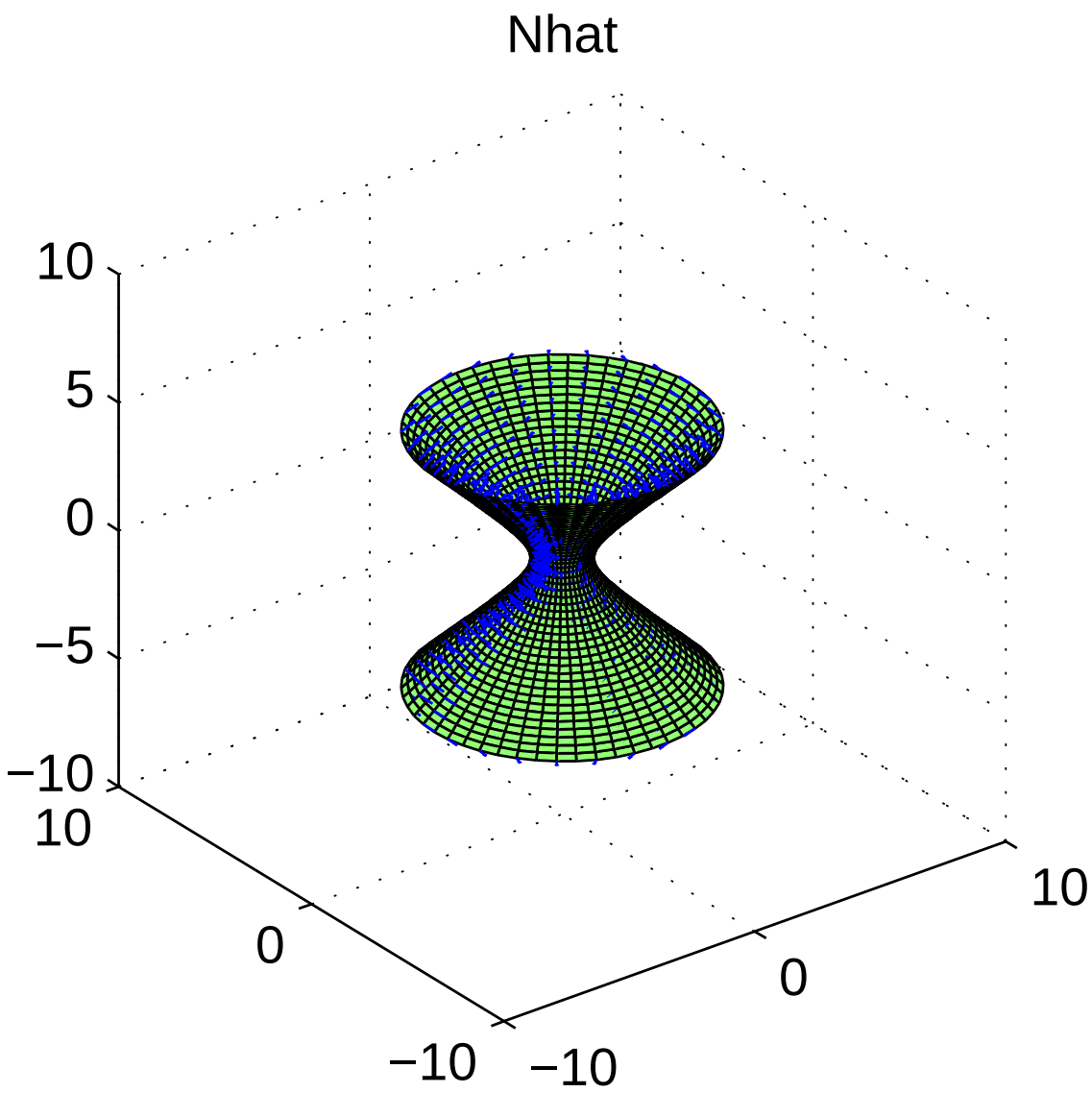
Quiver plots

The first three arguments to Matlab's `quiver3` are the foot coordinates; the last three are the vector coordinates.

```
% Draw the manifold for background
surf(x,y,z,zeros(ns,nt)); shading faceted; hold on;

% Draw the vector field
quiver3(x, y, z, DxDs, DyDs DzDs));
title 'D/Ds';
```


D/Ds
10
5
0
−5
10
0
−10
−10
0
10


D/Dt
5
0
−5
10
0
−10
−10
0
10


Nhat
10
5
0
−5
−10
10
0
−10
−10
0
10

Metric coefficients

The Riemannian metric is a twice-covariant tensor field which is symmetric and positive-definite at each point. (Think "field of dot products"). Due to bilinearity, it is specified by the *metric coefficients*:

$$g_{ij} \quad = \quad g\left(\frac{\partial}{\partial s_i}, \frac{\partial}{\partial s_j}\right)$$

For surfaces:

$$g_{11} = g\left(\frac{\partial}{\partial s}, \frac{\partial}{\partial s}\right) \qquad g_{12} = g\left(\frac{\partial}{\partial s}, \frac{\partial}{\partial t}\right)$$

$$g_{21} = g\left(\frac{\partial}{\partial t}, \frac{\partial}{\partial s}\right) \qquad g_{22} = g\left(\frac{\partial}{\partial t}, \frac{\partial}{\partial t}\right)$$

Dot products

Remember that we multiply vectors componentwise using `.*`. For dot products, we want plain old $*$, along with transpose. The dot product of $\mathbf{u}$ and $\mathbf{v}$, if $\mathbf{u}$ and $\mathbf{v}$ are written as row vectors, is the matrix product

$$\mathbf{u} \cdot \mathbf{v} = \mathbf{u}\, \mathbf{v}^t$$

e.g.

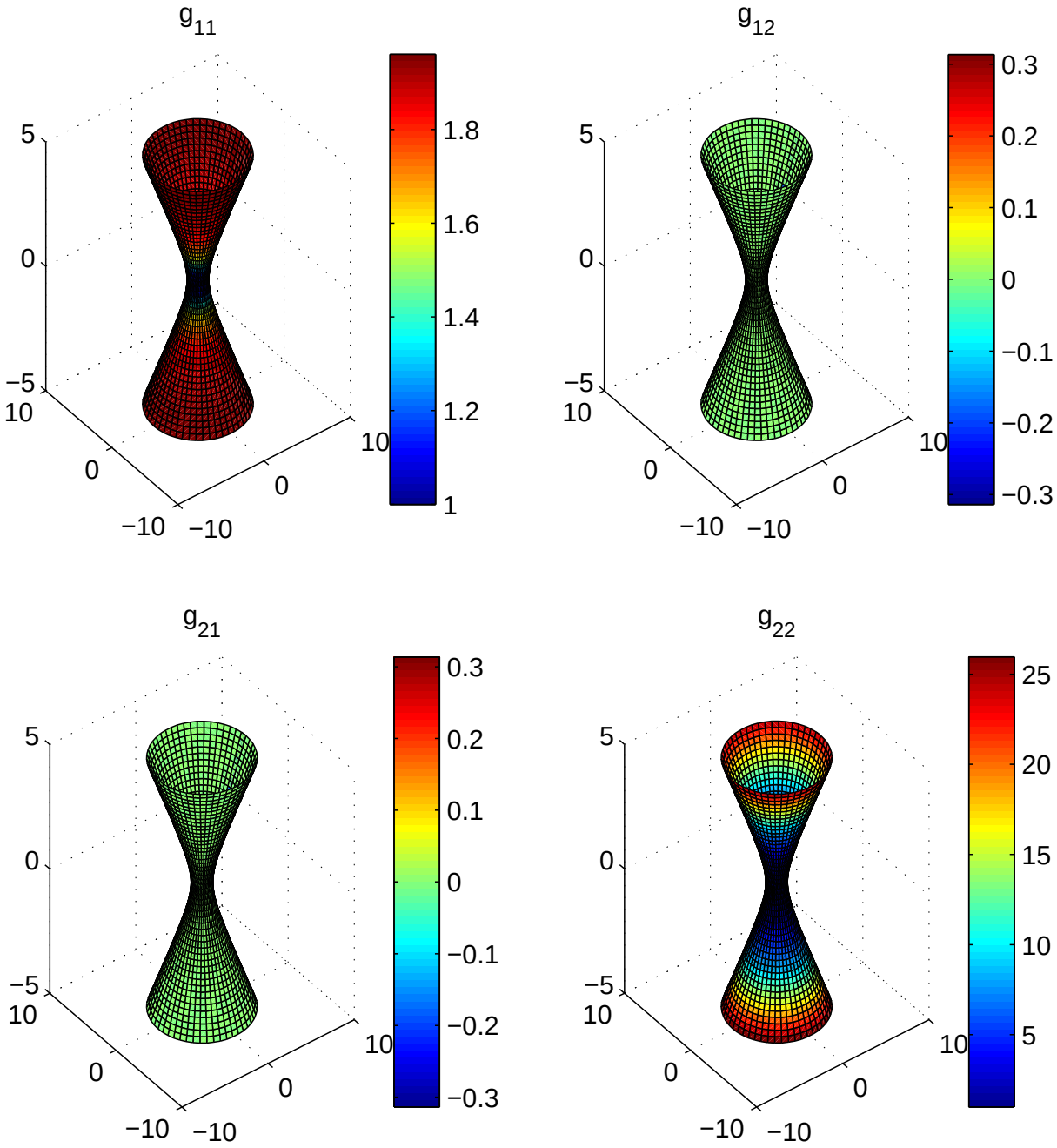
$$\begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \cdot \begin{pmatrix} 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

$$= 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 \qquad = 32.$$

Metric coefficients in Matlab

```
g = zeros(ns,nt,2,2); invg = zeros(ns,nt,2,2);
for ss = 1:ns
  for tt = 1:nt
    % The above DDs1 & DDt1 computations were here.
    a = DDs1 * DDs1'; b = DDs1 * DDt1';
    c = DDt1 * DDs1'; d = DDt1 * DDt1';
    g(ss,tt,1,1) = a; g(ss,tt,1,2) = b;
    g(ss,tt,2,1) = c; g(ss,tt,2,2) = d;
    gdet = a*d - b*c; % Inverse metric coefficients
    invg(ss,tt,1,1) =  d/gdet; invg(ss,tt,1,2) = -b/gdet;
    invg(ss,tt,2,1) = -c/gdet; invg(ss,tt,2,2) =  a/gdet;
  end
end
```

For the hyperboloid of one sheet, I used $s = z$, $t = \theta$.


g_{11}
5
0
−5
10
0
−10
−10
0
10
1.8
1.6
1.4
1.2
1
g_{12}
5
0
−5
10
0
−10
−10
0
10
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
g_{21}
5
0
−5
10
0
−10
−10
0
10
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
g_{22}
5
0
−5
10
0
−10
−10
0
10
25
20
15
10
5

Christoffel symbols

For surfaces, there are eight Christoffel symbols, with $i, j, k = 1, 2$:

$$\Gamma_{ij}^k = \frac{1}{2}\, g^{km} \left(\frac{\partial g_{jm}}{\partial i} + \frac{\partial g_{im}}{\partial j} - \frac{\partial g_{ij}}{\partial m} \right).$$

Here the Einstein summation convention (which Einstein, tongue-in-cheek, called his "one great contribution to mathematics") is used. That is, we mean

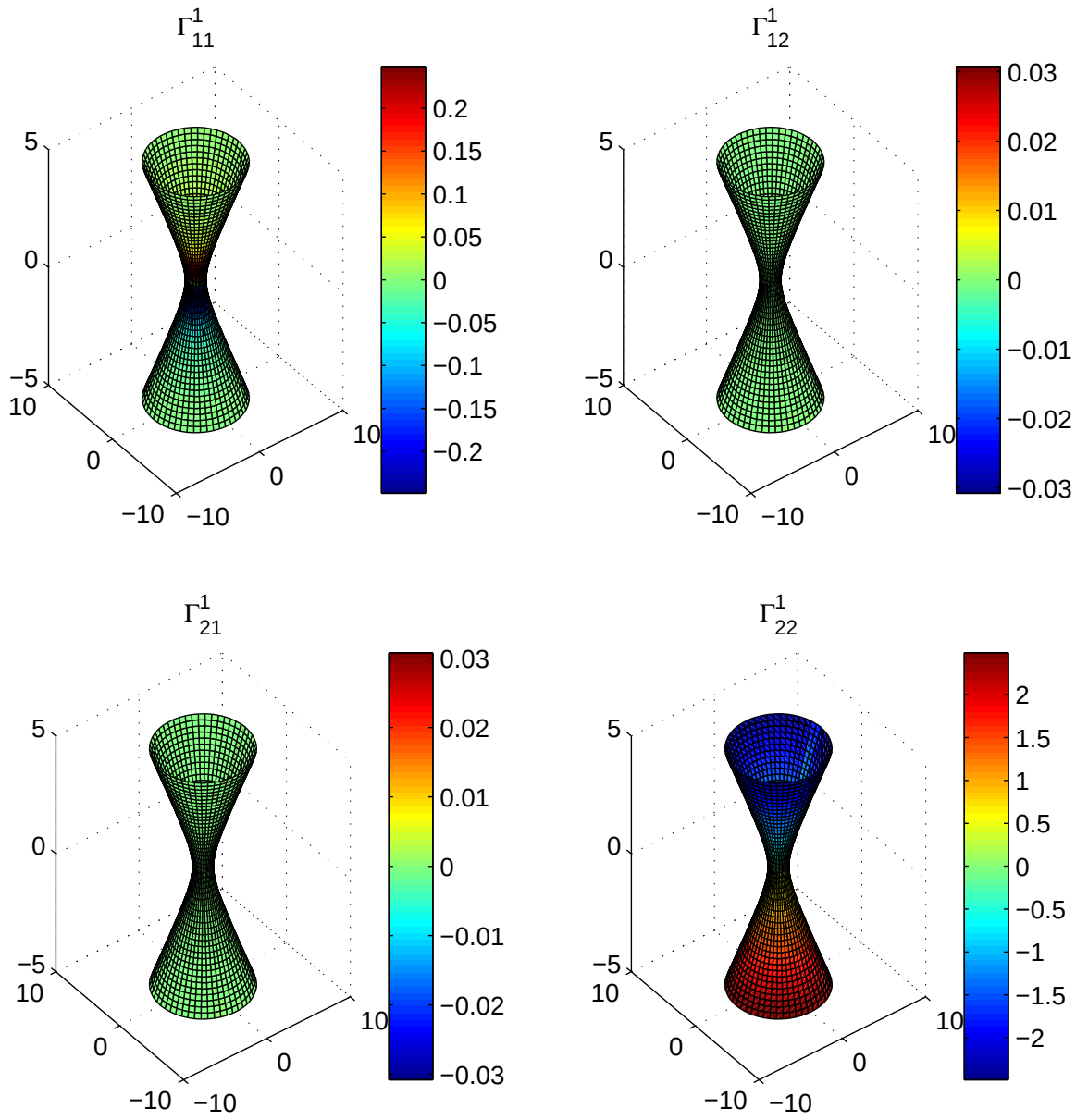
$$\Gamma_{ij}^k = \sum_{m=1}^{2} \frac{1}{2}\, g^{km} \left(\frac{\partial g_{jm}}{\partial i} + \frac{\partial g_{im}}{\partial j} - \frac{\partial g_{ij}}{\partial m} \right).$$

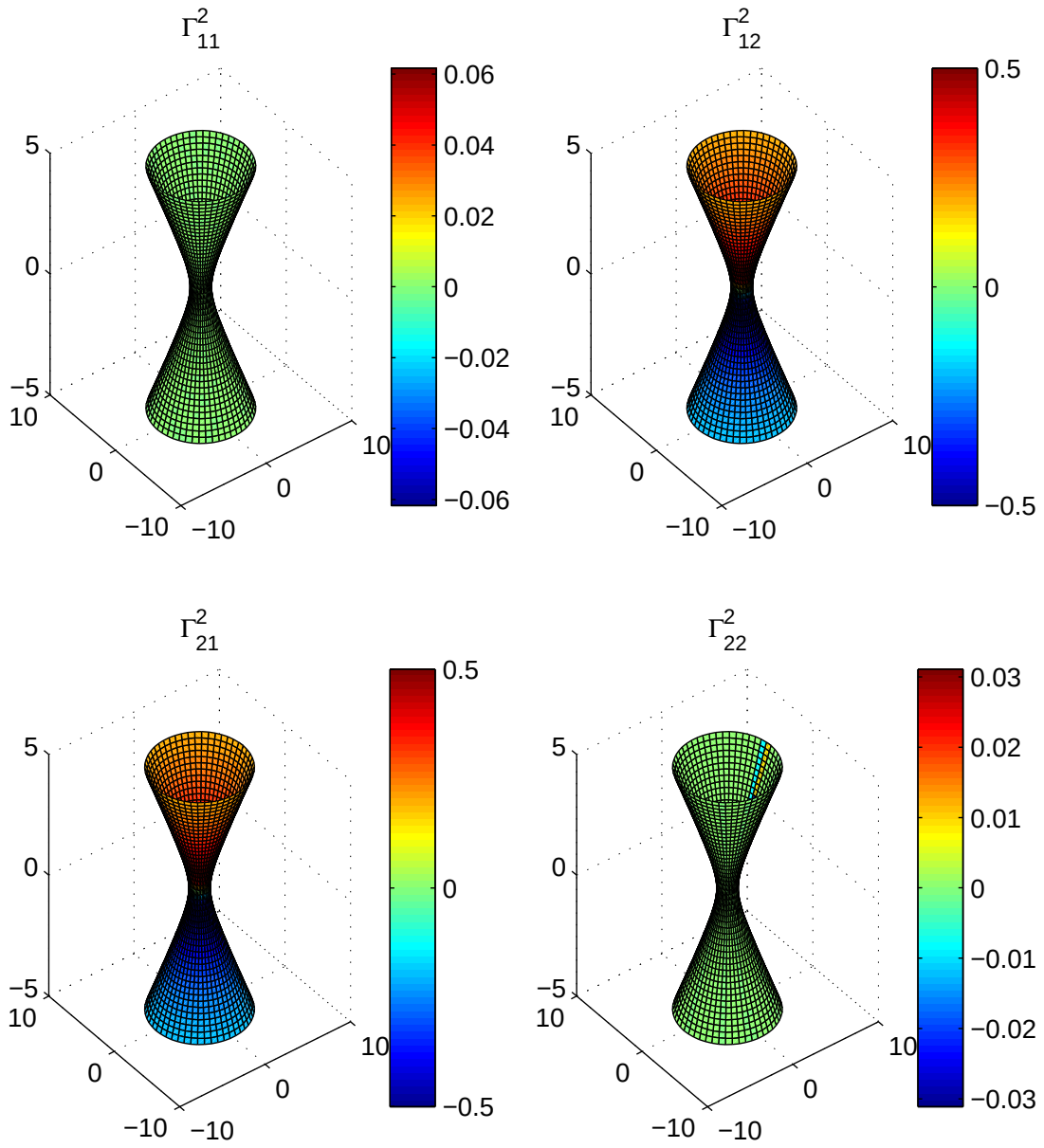
In Matlab

First compute the $\frac{\partial g_{jk}}{\partial i}$'s; Christoffel symbols on the next slide.

```
DD_g = zeros(ns,nt,2,2,2);

for j = 1:2
  for k = 1:2
    [DD_g(:,:,1,j,k), DD_g(:,:,2,j,k)] =
      gradient(g(:,:,j,k),ds,dt);
  end
end
```

```
Gamma = zeros(ns,nt,2,2,2);
for i = 1:2
  for j = 1:2
    for k = 1:2
      for m = 1:2
        Gamma(:,:,i,j,k) = Gamma(:,:,i,j,k) ...
          + 0.5 * invg(:,:,k,m) .* ( ...
          DD_g(:,:,i,j,m) + ...
          DD_g(:,:,j,i,m) - ...
          DD_g(:,:,m,i,j));
      end
    end
  end
end
```


Γ^1_{11}
5
0
−5
10
0
−10
−10
0
10
0.2
0.15
0.1
0.05
0
−0.05
−0.1
−0.15
−0.2
Γ^1_{12}
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
Γ^1_{21}
Γ^1_{22}
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2


Γ^2_{11}
0.06
0.04
0.02
0
−0.02
−0.04
−0.06
5
0
−5
10
0
−10
−10
0
10
Γ^2_{12}
0.5
0
−0.5
Γ^2_{21}
Γ^2_{22}
0.03
0.02
0.01
−0.01
−0.02
−0.03

Curvature tensors

The 1,3 curvature tensor:

$$R^l_{ijk} = \frac{\partial}{\partial i}\Gamma^l_{jk} + \Gamma^m_{ik}\Gamma^l_{jm} - \frac{\partial}{\partial j}\Gamma^l_{ik} - \Gamma^m_{jk}\Gamma^l_{im}.$$

The 0,4 curvature tensor:

$$R_{ijkl} = R_{ijk}{}^m g_{lm}.$$

The 0,2 curvature tensor:

$$R_{ij} = g^{kl} R_{kijl}.$$

Scalar curvature:

$$S = g^{ij} R_{ij}$$

which in turn is

$$g^{kl} g^{ij} R_{kijl}.$$

In Matlab

Compute $\frac{\partial}{\partial i}\Gamma^l_{jk}$'s.

```
DD_Gamma = zeros(ns,nt,2,2,2,2);
for j = 1:2
  for k = 1:2
    for l = 1:2
      [DD_Gamma(:,:,1,j,k,l), DD_Gamma(:,:,2,j,k,l)] = ...
        gradient(Gamma(:,:,j,k,l),ds,dt);
    end
  end
end
```

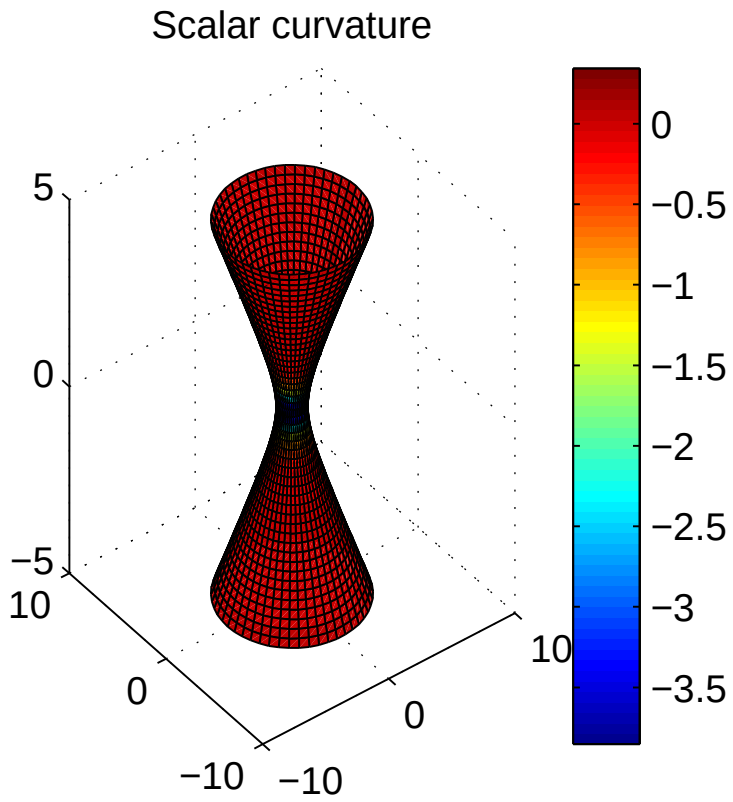
```
R13 = zeros(ns,nt,2,2,2,2);
for i = 1:2
  for j = 1:2
    for k = 1:2
      for l = 1:2
        for m = 1:2
          R13(:,:,i,j,k,l) = R13(:,:,i,j,k,l) ...
            + DD_Gamma(:,:,i,j,k,l) - DD_Gamma(:,:,j,i,k,l) ...
            + Gamma(:,:,i,k,m) .* Gamma(:,:,j,m,l) ...
            - Gamma(:,:,j,k,m) .* Gamma(:,:,i,m,l);
        end
      end
    end
  end
end
```

Matlab doesn't support index contraction by a built-in function (as far as I know), but you certainly can do it by summing.

```
R04 = zeros(ns,nt,2,2,2,2);
for i = 1:2
  for j = 1:2
    for k = 1:2
      for l = 1:2
        for m = 1:2
          R04(:,:,i,j,k,l) = R04(:,:,i,j,k,l) ...
            + R13(:,:,i,j,k,m) .* g(:,:,l,m);
        end
      end
    end
  end
end
```

```
R02 = zeros(ns,nt,2,2);
for i = 1:2
  for j = 1:2
    for k = 1:2
      for l = 1:2
        R02(:,:,i,j) = R02(:,:,i,j) ...
          + invg(:,:,k,l) .* R04(:,:,k,i,j,l);
      end
    end
  end
end
```

```
S = zeros(ns,nt);
for i = 1:2
  for j = 1:2
    S(:,:) = S(:,:) + invg(:,:,i,j) .* R02(:,:,i,j);
  end
end
```



More information

The level-curve problem (January 2001 #3):

`http://math.arizona.edu/~kerl/doc/prolrevqual.pdf`

Matlab code:

`http://math.arizona.edu/~kerl/gdg/gdgproj.m`

`http://math.arizona.edu/~kerl/gdg/y2z.m`

This file:

`http://math.arizona.edu/~kerl/gdg/gdgprojnotes.pdf`