ESS 212
SPRING 2020: *Homework # 1*
Due on April 15.

Exercise 1 Using the *ETOPO1 Global Relief Model*, compute the area of the Earth that is covered by oceans. Assume for simplicity that the Earth is a perfect sphere so that the area element is $dA = r^2 \cos(\phi) d\phi d\lambda$.

Solution: My script is called `hw1ex1_sol.jl`, which I put in the same folder (directory) as the file `/Users/fprimeau/.emacs.d/julia-emacs`.
`hw1ex1_sol.jl`:

```
using NetCDF,  Printf

cd(dirname(@__FILE__));

fname = "ETOPO1_Ice_g_gmt4.grd";
x = ncread(fname,"x"); # longitude (degrees)
y = ncread(fname,"y"); # latitude (degrees)
z = ncread(fname,"z"); # depth (m)

# compute the geometric average of the polar and equatorial radii
b = 6378e3; # equatorial radius [m]
c = 6356e3; # polar radius [m]
a = (b*b*c)^(1/3); # geometrical mean radius [m]

# convert longitude from degrees to radians
phi = pi*y/180;      # [rads]
dphi = phi[2]-phi[1];    # [rads]
# convert latitude from degrees to radians
lambda = pi*x/180;   #[rads]
dlambda = lambda[2]-lambda[1]; #[rads]

# compute the surface area element
dA = [ a^2*cos(p)*dphi*dlambda for l in lambda, p in phi];
AO = sum(dA[:].*(z[:].<0)); # Area of Ocean
AE = sum(dA[:]);            # Area of whole Earth
@printf("Area of ocean:       %e m2 \n",AO)
@printf("Area of whole Earth: %e m2 \n",AE)
@printf("Fraction of Earth covered by oceans: %f \n",AO/AE);
```

And here is the output it produces:

```
julia> include("hw1ex1_sol.jl")
Area of ocean:       3.619075e+14 m2
Area of whole Earth: 5.100334e+14 m2
Fraction of Earth covered by oceans: 0.709576
```

WARNING! Here is a bit of code that looks correct but actually isn't:

```
# compute the surface area element
dA = [ a^2*cos(p)*dphi*dlambda for p in phi, l in lambda];
AO = sum(dA[:].*(z[:].<0)); # Area of Ocean
AE = sum(dA[:]);            # Area of whole Earth
@printf("Area of ocean:       %e m2 WRONG!\n",AO)
@printf("Area of whole Earth: %e m2 OK\n",AE)
@printf("Fraction of Earth covered by oceans: %f WRONG!\n",AO/AE);
```

```
Area of ocean:       3.194574e+14 m2 WRONG!
Area of whole Earth: 5.100334e+14 m2 OK
Fraction of Earth covered by oceans: 0.626346 WRONG!
```

Can you figure out what's wrong?

Exercise 2 Again, using the *ETOPO1* data compute the volume of the world oceans. Assume for simplicity that the volume element is $\mathrm{d}V = a^2 \cos\phi \, \mathrm{d}\phi \mathrm{d}\lambda \mathrm{d}z$, where a is the radius of the Earth.

Solution: The total volume of the ocean is given by

$$\begin{aligned} V_{ocn} &= \int_{-\pi/2}^{\pi/2} \int_{-\pi}^{\pi} \int_{z_{bot}}^{0} a^2 \cos(\phi) dz d\lambda d\phi \\ &= \int_{-\pi/2}^{\pi/2} \int_{-\pi}^{\pi} -z_{bot} a^2 \cos(\phi) d\lambda d\phi \\ &\approx \sum_k -z_{bot,k} dA_k \end{aligned} \tag{1}$$

where k runs over every pixel in the *ETOPO1* field and where $z_{bot,k}$ is set to zero if the variable `z[:]` in the *ETOPO1* model is greater than zero.
Here is the extra bit of code that computes the volume of the ocean:

```
# compute the surface area element
dA = [ a^2*cos(p)*dphi*dlambda for l in lambda, p in phi];
# compute the total volume of the ocean
VO = sum(-dA[:].*z[:].*(z[:].<0));
@printf("Volume of of ocean: %e m3 \n",VO)
```

And here is the output it produces:

```
Volume of of ocean: 1.334050e+18 m3
```

Exercise 3 Use the 2018 World Ocean Atlas objectively analyzed annually averaged temperature and salinity data to compute the volumetrically averaged temperature and salinity of the ocean.

Solution: The volumetrically averaged fields are given by

$$\langle X \rangle = \frac{\int_V X(\lambda, \phi, z) dV}{\int_V dV}. \tag{2}$$

I compute the numerator by first integrating X over z for each water column and then I sum $X_k dA_k$ over k where k is the water column index. The vertical integral with respect to z is approximated using the trapezoidal rule.

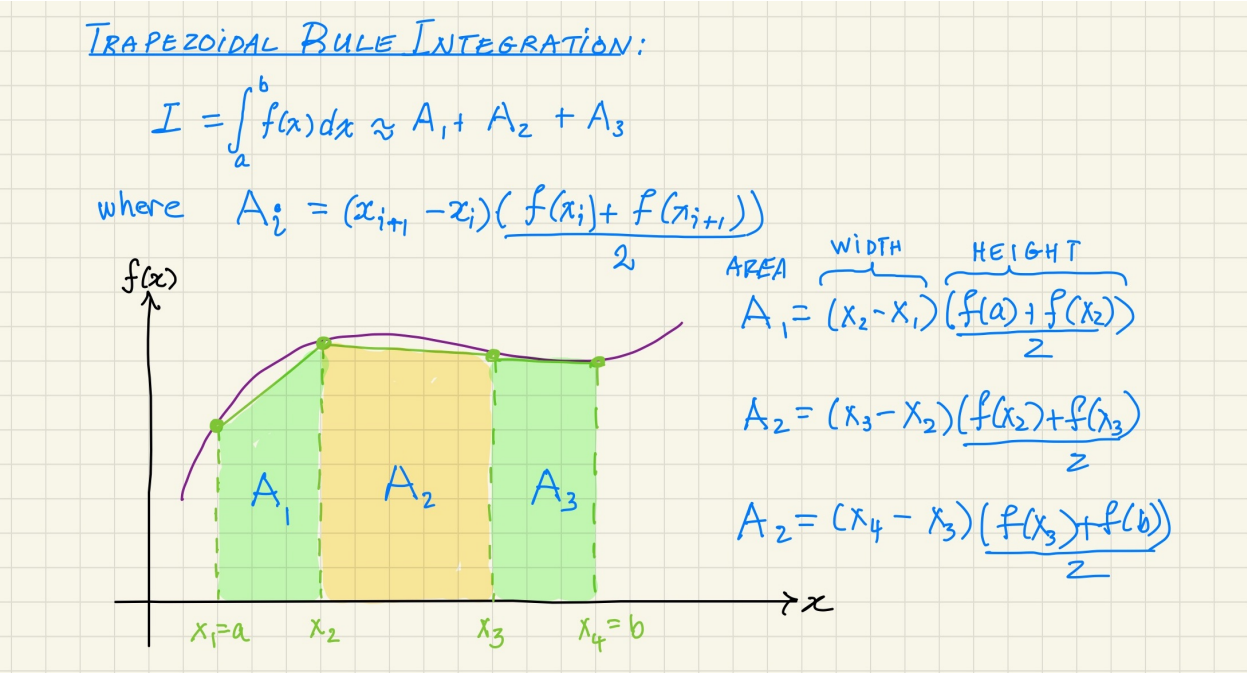


Figure 1: The trapezoidal integration rule approximates an integral using the area of a finite set of discrete trapezoids whose areas are easy to compute.

The script is called `hw1ex3_sol.jl`:

```
using NetCDF, Printf
cd(dirname(@__FILE__));
ft= "woa18_decav_t00_01.nc";
fs= "woa18_decav_s00_01.nc";
lat = Float64.(ncread(ft,"lat"));
lon = Float64.(ncread(ft,"lon"));
depth = Float64.(ncread(ft,"depth"));
t_an = Float64.(ncread(ft,"t_an"));
s_an = Float64.(ncread(fs,"s_an"));
```

```
t_an[t_an.>1f20].=NaN;
s_an[s_an.>1f20].=NaN;

# trapezoidal integration in the vertical
function trapz(f,z)
    I = 0
    for k in 2:length(z)
        dz = z[k] - z[k-1]
        h = 0.5*( f[k] + f[k-1] )
        I = I+dz*h
    end
    return I
end

# compute the geometric average of the polar and equatorial radii
b = 6378e3; # equatorial radius [m]
c = 6356e3; # polar radius [m]
a = (b*b*c)^(1/3); # geometrical mean radius [m]

# the woa18 atlas has a 1 deg x 1 deg horizontal resolution
dphi = 1*pi/180 # (radians)
dlam = 1*pi/180 # (radians)

# area element for every water column
dA = [a^2*cos(lat[j]*pi/180)*dphi*dlam for i in 1:length(lon), j in 1:length(lat)]

# maximum depth index for every water column
Kmx = [sum(.~isnan.(t_an[i,j,:,1])) for i in 1:length(lon), j in 1:length(lat)]

# volume of every water column
dV = [dA[i,j]*trapz(ones(Kmx[i,j]),depth[1:Kmx[i,j]]) for i in 1:length(lon),
      j in 1:length(lat)]

# integrated temperature for every water column
dIT = [dA[i,j]*trapz(t_an[i,j,1:Kmx[i,j]],depth[1:Kmx[i,j]]) for i in 1:length(lon),
      j in 1:length(lat)]

# integrated salinity for every water column
dIS = [dA[i,j]*trapz(s_an[i,j,1:Kmx[i,j]],depth[1:Kmx[i,j]]) for i in 1:length(lon),
      j in 1:length(lat)]

# sum the voume of every water column
V = sum(dV[:])
# sum the integrated temperature and salinity of every water column
IT = sum(dIT[:])
```

```
IS = sum(dIS[:])
# compute the volumetrically weighted averages
Tbar = IT/V
Sbar = IS/V

@printf("Total volume: %e \n",V)
@printf("Volumetrically averaged temperature: %f (deg C)\n",Tbar);
@printf("Volumetrically averaged salinity:    %f (psu)\n",Sbar);
```

The output produced is

```
julia> include("hw1ex3_sol.jl")
Total volume: 1.358832e+18
Volumetrically averaged temperature: 3.695175 (deg C)
Volumetrically averaged salinity:    34.720148 (psu)
```

Exercise 4 Using the TEOS-10 Thermodynamic Equation of Seawater - 2010 and 2018 World Ocean Atlas objectively analyzed temperature and salinity, compute

1. the volumetrically average thermal expansion coefficient for seawater in the top 200 m of the ocean,
2. the volumetrically average haline contraction coefficient for seawater in the top 200 m of the ocean,
3. the total mass of salt in the ocean,
4. the total mass of freshwater in the ocean.

Solution:
1. The 25th level of the 2018 World Ocean Atlas is at a depth of 200 m. We can therefore use the same method as Exercise 3, except now we set the maximum depth to be the minimum of `Kmx` and 25.
The resulting script is `hw3ex1_sol.jl`, whose listing (omitting lines that are in common with Exercise 3) is:

```
# maximum depth index for every water column
Kmx = [sum(.~isnan.(t_an[i,j,:,1])) for i in 1:length(lon), j in 1:length(lat)]

# minimum of 200 m and maximum water column depth
K200 = [minimum([Kmx[i,j],25]) for i in 1:length(lon), j in 1:length(lat)];

# compute the pressure for every grid point in the WOA
p = [gsw_p_from_z(-Float64(depth[k]),lat[j]) for i in 1:length(lon),
     j in 1:length(lat), k in 1:length(depth) ]

# compute the absolute salinity for every grid point in the WOA
sa = [gsw_sa_from_sp(s_an[i,j,k],p[i,j,k],lon[i],lat[j]) for i in 1:length(lon),
      j in 1:length(lat), k in 1:length(depth)]

# compute the conservative temperature for every grid point in the WOA
ct = [gsw_ct_from_t(sa[i,j,k],t_an[i,j,k],p[i,j,k]) for i in 1:length(lon),
      j in 1:length(lat), k in 1:length(depth)]

# compute the density of sea water for every grid point in the WOA
rho = [gsw_rho(sa[i,j,k],ct[i,j,k],p[i,j,k]) for i in 1:length(lon),
       j in 1:length(lat), k in 1:length(depth)]

# compute the thermal expansion coefficient for every grid point in the top 200 m
alpha = [gsw_alpha(sa[i,j,k],ct[i,j,k],p[i,j,k]) for i in 1:length(lon),
         j in 1:length(lat), k in 1:25]

# compute the thermal expansion coefficient for every grid point in the top 200 m
```

```
beta = [gsw_beta(sa[i,j,k],ct[i,j,k],p[i,j,k]) for i in 1:length(lon),
         j in 1:length(lat), k in 1:25]

# compute the total mass of seawater in each water column
dM = [dA[i,j]*trapz(rho[i,j,1:Kmx[i,j]],depth[1:Kmx[i,j]]) for i in 1:length(lon), j in 1:leng

# compute the total mass of seawater in the ocean
Mtotal = sum(dM[:]);


# compute the total mass of salt in each water column
dMsalt = [dA[i,j]*trapz(rho[i,j,1:Kmx[i,j]].*sa[i,j,1:Kmx[i,j]]/1000,depth[1:Kmx[i,j]]) for i
          j in 1:length(lat)]
Msalt = sum(dMsalt[:]);

# compute the total mass of fresh water in the ocean
Mfresh = Mtotal-Msalt;

# volume of every water column (above 200 m)
dV200 = [dA[i,j]*trapz(ones(K200[i,j]),depth[1:K200[i,j]]) for i in 1:length(lon),
      j in 1:length(lat)]

# integrated temperature for every water column
dIalpha = [dA[i,j]*trapz(alpha[i,j,1:K200[i,j]],depth[1:K200[i,j]])
           for i in 1:length(lon), j in 1:length(lat)]

# integrated salinity for every water column
dIbeta = [dA[i,j]*trapz(beta[i,j,1:K200[i,j]],depth[1:K200[i,j]])
          for i in 1:length(lon), j in 1:length(lat)]

# sum the voume of water in the top 200 m of the ocean
V200 = sum(dV200[:])

# sum the integrated alpha and beta for every water column
Ialpha = sum(dIalpha[:])
Ibeta = sum(dIbeta[:])
# compute the volumetrically weighted averages
alphabar = Ialpha/V200
betabar = Ibeta/V200

@printf("Total volume with depth < 200 m: %e \n",V200)
@printf("Volumetrically averaged thermal expansion coefficient: %e (1/K)\n",alphabar);
@printf("Volumetrically averaged saline contraction coefficient: %e (kg/g)\n",betabar);
@printf("Total mass of seawater in the ocean: %e (kg)\n",Mtotal);
@printf("Total mass of salt in the ocean: %e (kg)\n",Msalt);
```

```
@printf("Total mass of fresh water in the ocean: %e (kg)\n",Mfresh);
```

And the output is

```
julia> include("hw1ex4_sol.jl")
Total volume with depth < 200 m: 6.799311e+16
Volumetrically averaged thermal expansion coefficient: 2.262311e-04 (1/K)
Volumetrically averaged saline contraction coefficient: 7.329867e-04 (kg/g)
Total mass of seawater in the ocean: 1.416410e+21 (kg)
Total mass of salt in the ocean: 4.942560e+19 (kg)
Total mass of fresh water in the ocean: 1.366984e+21 (kg)
```

Problem 1 (Optional problem. Do it if you are an aspiring geodesist!) Repeat exercise 1 but this time use a more accurate area element that takes into account the fact that the Earth is shaped like an oblate spheroid rather than a sphere. In other words, instead of using the spherical coordinate defined in the lecture,

$$\begin{aligned} x &= a\cos\phi\cos\lambda, \\ y &= a\cos\phi\sin\lambda, \\ z &= a\sin\phi, \end{aligned} \tag{3}$$

use coordinates defined by,

$$\begin{aligned} x &= \frac{b\cos\phi\cos\lambda}{\sqrt{1-\epsilon^2\sin^2\phi}} \\ y &= \frac{b\cos\phi\sin\lambda}{\sqrt{1-\epsilon^2\sin^2\phi}} \\ z &= \frac{b(1-\epsilon^2)\sin\phi}{\sqrt{1-\epsilon^2\sin^2\phi}} \end{aligned} \tag{4}$$

where

$$\epsilon^2 = \frac{b^2-c^2}{b^2}, \tag{5}$$

with $b = 6378.137$ km the equatorial radius and $c = 6356.752$ km the polar radius. (Note that if we set $\lambda = 0$ and $\phi = 0$ then $x = b$, the equatorial radius, and that if we set $\phi = \pi/2$ then $z = b\sqrt{1-\epsilon^2} = c$, the polar radius, as expected.)
The area element is then given by

$$\mathrm{d}A = h_\phi h_\lambda \mathrm{d}\phi\mathrm{d}\lambda, \tag{6}$$

where

$$h_\phi = \left|\left|\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\phi}\right|\right| \qquad \text{and} \qquad h_\lambda = \left|\left|\frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\lambda}\right|\right| \tag{7}$$

with $\mathbf{r} = \begin{pmatrix} x & y & z \end{pmatrix}'$ being the position vector.

Solution: The derivative of the position vector with respect to ϕ is given by

$$\begin{aligned} \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\phi} &= \begin{bmatrix} \frac{\mathrm{d}x}{\mathrm{d}\phi} \\ \frac{\mathrm{d}y}{\mathrm{d}\phi} \\ \frac{\mathrm{d}z}{\mathrm{d}\phi} \end{bmatrix} \\ &= \frac{b(1-\epsilon^2)}{(1-\epsilon^2\sin^2(\phi))^{3/2}} \begin{bmatrix} \cos(\lambda)\sin(\phi) \\ \sin(\lambda)\sin(\phi) \\ -\cos(\phi) \end{bmatrix} \end{aligned} \tag{8}$$

Thus

$$\begin{aligned} h_\phi &= \left|\left| \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\phi} \right|\right| \\ &= \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}\phi}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}\phi}\right)^2 + \left(\frac{\mathrm{d}z}{\mathrm{d}\phi}\right)^2} \\ &= \frac{b(1-\epsilon^2)}{\left(1-\epsilon^2\sin^2(\phi)\right)^{3/2}} \end{aligned} \tag{9}$$

The derivative of the position vector with respect to λ is given by

$$\begin{aligned} \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\lambda} &= \begin{bmatrix} \frac{\mathrm{d}x}{\mathrm{d}\lambda} \\ \frac{\mathrm{d}y}{\mathrm{d}\lambda} \\ \frac{\mathrm{d}z}{\mathrm{d}\lambda} \end{bmatrix} \\ &= \frac{b}{(1-\epsilon^2\sin^2(\phi))^{1/2}} \begin{bmatrix} -\cos(\phi)\sin(\lambda) \\ \cos(\phi)\cos(\lambda) \\ 0 \end{bmatrix} \end{aligned} \tag{10}$$

Thus

$$\begin{aligned} h_\lambda &= \left|\left| \frac{\mathrm{d}\mathbf{r}}{\mathrm{d}\lambda} \right|\right| \\ &= \sqrt{\left(\frac{\mathrm{d}x}{\mathrm{d}\lambda}\right)^2 + \left(\frac{\mathrm{d}y}{\mathrm{d}\lambda}\right)^2 + \left(\frac{\mathrm{d}z}{\mathrm{d}\lambda}\right)^2} \\ &= \frac{b\cos(\phi)}{\left(1-\epsilon^2\sin^2(\phi)\right)^{1/2}} \end{aligned} \tag{11}$$

Thus the area element for this model of the Earth is

$$\begin{aligned} \mathrm{d}A &= h_\phi h_\lambda \mathrm{d}\phi\mathrm{d}\lambda \\ &= \frac{b^2\cos(\phi)(1-\epsilon^2)}{(1-\epsilon^2\sin^2(\phi))^2} \end{aligned} \tag{12}$$

Although the final result is simple enough, the calculations are a bit tedious and it's hard not to make mistakes. I used Matlab's symbolic toolbox to check my calculations. The Matlab script is

```
syms x y z b phi lambda epsilon
tmp = sqrt(1-epsilon^2*sin(phi)^2);
x = b*cos(phi)*cos(lambda)/tmp;
y = b*cos(phi)*sin(lambda)/tmp;
z = b*(1-epsilon^2)*sin(phi)/tmp;

dxdphi = simplify(diff(x,phi));
dydphi = simplify(diff(y,phi));
dzdphi = simplify(diff(z,phi));
```

```
dxdlambda = simplify(diff(x,lambda));
dydlambda = simplify(diff(y,lambda));
dzdlambda = simplify(diff(z,lambda));

fprintf('dxdphi = \n')
pretty(dxdphi)
fprintf('dydphi = \n')
pretty(dydphi)
fprintf('dzdphi = \n')
pretty(dzdphi)

fprintf('dxdlambda = \n')
pretty(dxdlambda)
fprintf('dydlambda = \n')
pretty(dydlambda)
fprintf('dzdlambda = \n')
pretty(dzdlambda)

hphi = sqrt(simplify(dxdphi^2+dydphi^2+dzdphi^2));
hlambda = sqrt(simplify(dxdlambda^2+dydlambda^2+dzdlambda^2));

fprintf('hphi = \n')
pretty(hphi)
fprintf('hlambda = \n')
pretty(hlambda)
```

And the output it produced is

```
>> hfactors
dxdphi =
                                  2
 b cos(lambda) sin(phi) (epsilon  - 1)
---------------------------------------
        2         2          2     3/2
(epsilon  cos(phi)  - epsilon  + 1)

dydphi =
                                 2
b sin(lambda) sin(phi) (epsilon  - 1)
---------------------------------------
                  2         2 3/2
     (1 - epsilon  sin(phi) )

dzdphi =
                             2
         b cos(phi) (epsilon  - 1)
- ---------------------------------------
```

```
          2         2          2     3/2
  (epsilon  cos(phi)  - epsilon  + 1)

dxdlambda =
     b cos(phi) sin(lambda)
- ---------------------------
                  2         2
  sqrt(1 - epsilon  sin(phi) )

dydlambda =
   b cos(lambda) cos(phi)
---------------------------
                2         2
sqrt(1 - epsilon  sin(phi) )

dzdlambda =
0

hphi =
    /        2         2     2    \
    |       b  (epsilon  - 1)     |
sqrt| - ------------------------- |
    |           2         2     3 |
    \   (epsilon  sin(phi)  - 1)  /

hlambda =
    /    2          2        \
    |   b  (sin(phi)  - 1)    |
sqrt| ---------------------- |
    |        2         2     |
    \ epsilon  sin(phi)  - 1 /

>>
```

The Julia script for Exercise 1 using this more accurate area element is called `hw1prob1_sol.jl`

```
using NetCDF,  Printf
cd(dirname(@__FILE__));
fname = "ETOPO1_Ice_g_gmt4.grd";
x = ncread(fname,"x"); # longitude (degrees)
y = ncread(fname,"y"); # latitude (degrees)
z = ncread(fname,"z"); # depth (m)
# compute the geometric average of the polar and equatorial radii
b = 6378e3; # equatorial radius [m]
c = 6356e3; # polar radius [m]
a = (b*b*c)^(1/3); # geometrical mean radius [m]
```

```
eps2 = 1-(c/b)^2;
# convert longitude from degrees to radians
phi = pi*y/180;      # [rads]
dphi = x[2]-x[1];    # [rads]
# convert latitude from degrees to radians
lambda = pi*x/180;   #[rads]
dlambda = y[2]-y[1]; #[rads]
# compute the surface area element
dA =[(b^2*cos(phi[j])*(1-eps2)/((1-eps2*sin(phi[j])^2)^2))*dphi*dlambda
      for i in 1:length(x),  j in 1:length(y)]
AO = sum(dA[:].*(z[:].<0)); # Area of Ocean
AE = sum(dA[:]);            # Area of whole Earth
@printf("Area of ocean:       %e m2 \n",AO)
@printf("Area of whole Earth: %e m2 \n",AE)
@printf("Fraction of Earth covered by oceans: %f \n",AO/AE);
```

It produces the following output:

```
julia> include("hw1prob1_sol.jl")
Area of ocean:       3.617876e+14 m2
Area of whole Earth: 5.100344e+14 m2
Fraction of Earth covered by oceans: 0.709340
```

While the volumes of the Earth models based on the sphere and the oblate spheroid are the same (by definition of the geometric mean), the surface area of the oblate spheriod is a bit bigger and the fraction of the area covered by the ocean is a bit smaller.
Also compute the volume of the ocean using the volume element

$$\mathrm{d}V = h_\phi h_\lambda \mathrm{d}\phi \mathrm{d}\lambda \mathrm{d}\xi, \tag{13}$$

where ξ is a vertical coordinate in the direction of a local plumb line near the surface of the Earth, i.e. ξ corresponds to the variable `z` in the *ETOPO1* data, whereas $(\lambda\,,\,\phi)$ corresponds to the geographic longitude and latitudes, i.e. the `x` and `y` variables in *ETOPO1* data.

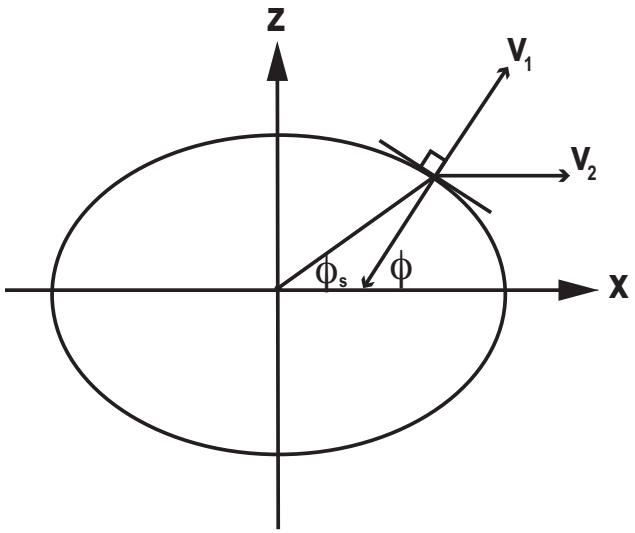
The volume of the ocean can be computed by adding the following bit of code to `hw1prob1_sol.jl` listed above

```
# compute the total volume of the ocean
VO = sum(-dA[:].*z[:].*(z[:].<0));
@printf("Volume of of ocean: %e m3 \n",VO)
```

to produce the following output:

```
Volume of of ocean: 1.333079e+18 m3
```

Note that the geographic latitude corresponds to the angle between the equatorial plane and a local vertical plumb line. As such it is slightly different than the geocentric latitude, which corresponds to the angle between the equatorial plane and a ray connecting points at the surface and center of the Earth. By setting $y = 0$ and $\lambda = 0$ and varying ϕ between $-\pi/2$ and $\pi/2$ the coordinates x



and z in Eq. (2) trace out a curve on the surface of a reference oblate spheroid that approximates a geopotential surface near sea level. The curve corresponds to a meridian. We often say that the meridians are 'great circles', but in fact they are ellipses. The angle between a vector, $\mathbf{v}_1$, perpendicular to a line tangent to the surface, (i.e. aligned with a vertical plumb line) and a vector, $\mathbf{v}_2$, along the intersection between the Equatorial plane and the plane $y = 0$ is defined as the geographic latitude. The angle between a ray connecting a point on the surface of the Earth and the center of the Earth and the equatorial plane is called the geocentric latitude (i.e. the latitude as defined in the lecture on spherical coordinates). Make a plot of $\Delta\phi \equiv \phi - \phi_s$, the difference between the geographic latitude and the geocentric latitude as a function of distance along the reference ellipsoid. ϕ_s denotes the geocentric latitude and ϕ denotes the geographic latitude. Use the arclength formula to compute the distance

$$s = \int_{-\pi/2}^{\pi/2} \sqrt{\left(\left.\frac{\mathrm{d}x}{\mathrm{d}\phi}\right|_{\lambda=0}\right)^2 + \left(\left.\frac{\mathrm{d}z}{\mathrm{d}\phi}\right|_{\lambda=0}\right)^2}\,\mathrm{d}\phi. \tag{14}$$

I don't believe that the above integral can be done analytically. Therefore approximate it numerically using the trapezoidal rule:

$$s_j = \sum_{k=1}^{j} \Delta s_k \tag{15}$$

where

$$\Delta s_k = \frac{f(\phi_{k-1}) + f(\phi_k)}{2}(\phi_k - \phi_{k-1}) \tag{16}$$

and

$$f(\phi_k) = \sqrt{\left(\left.\frac{\mathrm{d}x}{\mathrm{d}\phi}\right|_{\lambda=0,\phi=\phi_k}\right)^2 + \left(\left.\frac{\mathrm{d}z}{\mathrm{d}\phi}\right|_{\lambda=0,\phi=\phi_k}\right)^2}. \tag{17}$$

For the numerical integration you can choose 180 equally spaced ϕ_k points starting from $\phi_0 = -\pi/2$ and $\phi_{180} = \pi/2$. For each of these discrete points you can compute the value ϕ_s by first computing

the dot product between $\mathbf{v}_1 = (\ x(\phi_k) \quad 0 \quad z(\phi_k)\ )'$ and $\mathbf{v}_2 = (\ b \quad 0 \quad 0\ )'$ and then using the fact that

$$\phi_{s,k} = \arccos\left(\frac{\mathbf{v}_1 \cdot \mathbf{v}_2}{||\mathbf{v}_1||\,||\mathbf{v}_2||}\right). \tag{18}$$

To compute the arclength element along a meridian is computed using

$$\left[\frac{\mathrm{d}x}{\mathrm{d}\phi}\right]^2_{\lambda=0} = \frac{b^2\sin^2(\phi)(1-\epsilon^2)^2}{(1-\epsilon^2\sin^2(\phi))^3} \tag{19}$$

and

$$\left[\frac{\mathrm{d}z}{\mathrm{d}\phi}\right]^2_{\lambda=0} = \frac{b^2(1-\epsilon^2)^2\cos^2(\phi)}{(1-\epsilon^2\sin^2(\phi))^3} \tag{20}$$

so that

$$ds = \frac{b(1-\epsilon^2)}{(1-\epsilon^2\sin^2(\phi))^{3/2}}d\phi \tag{21}$$

(It turns out that there is a closed form expression for the integral of ds, but the expression is a mess and not particularly enlightening. The numerical solution based on the trapezoidal integration rule is simpler to implement.
The Julia code snippet to do the integral and plot the difference between ϕ and ϕ_s is

```
# compute phi_s
X = [b*cos(phi[j])/sqrt(1-eps2*sin(phi[j])^2) for j in 1:length(y)]
Z = [b*(1-eps2)*sin(phi[j])/sqrt(1-eps2*sin(phi[j])^2) for j in 1:length(y)]
r = sqrt.(X.^2+Z.^2)
phi_s = sign.(Z).*acos.(X./r);
# compute the arclength element
ds = [b*(1-eps2)/((1-eps2*sin(phi[j])^2)^(3/2)) for j in 1:length(y)]

# trapezoidal integration in the vertical
function trapz(f,z)
    I = 0
    for k in 2:length(z)
        dz = z[k] - z[k-1]
        h = 0.5*( f[k] + f[k-1] )
        I = I+dz*h
    end
    return I
end

s = [trapz(ds[1:j],phi[1:j]) for j in 1:length(y)]
delta_phi = (phi .- phi_s)*pi/180;
plot(s/1e6,delta_phi*10^5,
     xlabel = "Distance from South Pole  (10^3 km)",
     ylabel = "phi - phi_s  (10^-5 degrees) ",
     legend = :false)
```

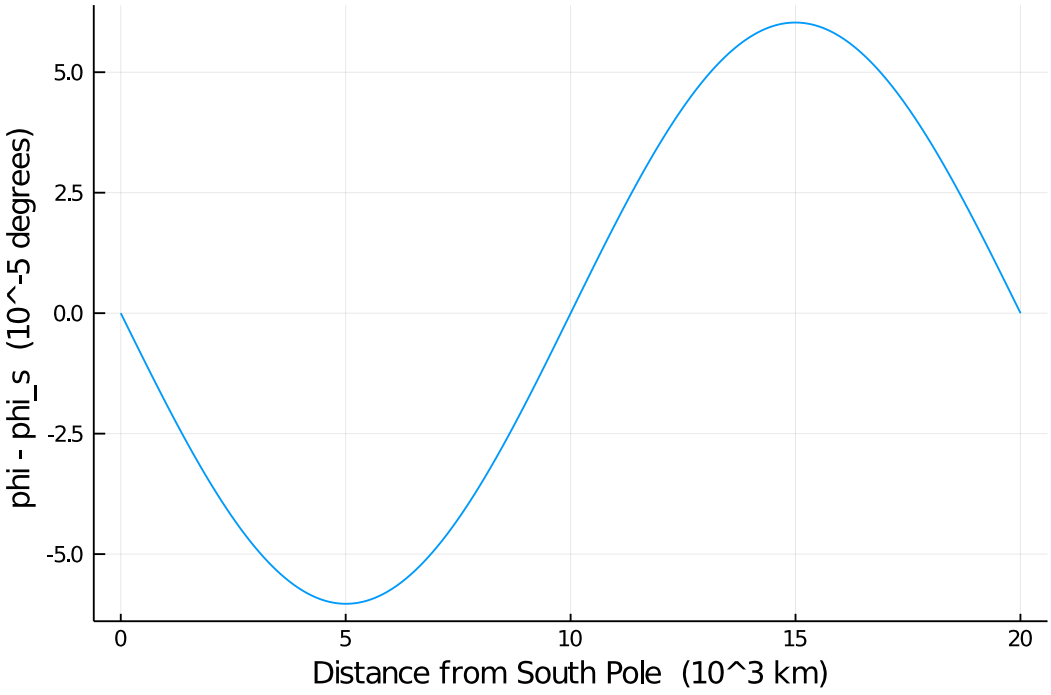


Figure 2: Graph of $\phi - \phi_s$, the angle between $\mathbf{g}$ and a vector pointing towards the center of the Earth for the particular oblate spheroid model used¡ in this problem. The maximum angle is 6.03×10^{-5} degrees at 45°S and 45°N.

[There are several different models for the reference geopotential. The one used to compute the hypsometric curve from *ETOPO1* data set on the NOAA website (`https://www.ngdc.noaa.gov/mgg/global/etopo1_surface_histogram.html`) uses the discretized area element

$$\Delta A = \frac{b^2 \cos\left[\phi(1-\epsilon^2)\right] \Delta\phi\Delta\lambda}{(1-\epsilon^2 \sin^2\phi)^2}. \tag{22}$$

I could not find the reference describing the equation of the ellipsoid that produces the above discretized area element. If anyone can find the reference please let me know.]