Introduction to using the SDWBA in acousticTS

Brandyn M. Lucca

2020-04-01

General Introduction

The purpose of `acousticTS` is the calculate the acoustic target strength (TS, dB re: m^2) of a given scatterer, such as bubbles or euphausiids. Euphausiid TS is generally modelled using the distorted Born wave approximation (DWBA), which is a function of acoustic frequency (Hz), material properties (soundspeed, h, and density, g, contrast compared to ambient seawater), body morphometrics (shape, r_0, length, L, and curvature, ρ_c), and orientation relative to the sea surface (θ). This model is used because it can be calculated for all frequencies, orientations, and animal shapes. The model treats each animal as a series of of bent cylinders that are generally weakly scattering ($\pm$ 5%). We use a special formulation of the model known as the SDWBA, which adds an additional phase deviation term (ϕ) that accounts for imperfections in how sound scatters off of each cylinder. This model is mathematically expressed as:

$$f_{bs}(\theta) = \sum_{j=1}^{N}[\frac{k_1 \ \rho_c}{4} \ e^{(2 \ i \ k_2 \ \rho_c)} \int M \ e^{(-2 \ i \ \vec{k_2} \ cos \ \beta_{tilt})} \frac{a_j J_1(2 \ k_2 \ a_j cos \ \beta_{tilt})}{cos \ \beta_{tilt}} \ d\beta_{tilt}] e^{(i \ \phi_j)}$$

where N is the number of cylinders comprising each animal shape, i is an imaginary number, $\vec{k_1}$ is the acoustic wave number of seawater adjusted for animal orientation θ whereby broadside incidence is 90°/270°, $\vec{k_2}$ is the acoustic wave number inside each cylinder adjusted for orientation, $\vec{r_0}$ is the position matrix which represents the x, y, and z coordinates of the animal shape, a_j is the radius for cylinder j, M is equal to $\frac{1}{gh^2} + \frac{1}{g-2}$, g is the density contrast ($\frac{\rho_{animal}}{\rho_{seawater}}$), h is the soundspeed contrast ($\frac{c_{animal}}{c_{seawater}}$), J_1 is the Bessel function of the first kind of order 1, ρ_c is the radius of curvature, and ϕ_j is the stochastic phase variability term for each cylinder which provides a multiplicative term drawn from a standard normal distribution: $N(0, \phi_j)$. The TS of the scatterer is then calculated via $TS = 20 \ log_{10}(f_{bs}(\theta))$.

Package installation

The `acousticTS` package can be found on GitHub at https://github.com/brandynlucca/acousticTS and was built using R version 3.6.1. The package is actively updated, so updates and new features can be tracked via this repository. R is an open-source, statistical programming-language commonly used for data analysis across many fields. Updated versions of R can be installed on Windows, MacOS, and UNIX platforms from https://r-project.org. This package has four dependencies: `elliptic` (integration of complex numbers) and `parallel`/`foreach`/`doParallel` (parallelized functions for improved model performance).

To install and call the latest version of `acousticTS` from the development GitHub repository:

```
#install.packages("devtools") #if needed, uncomment this line and install devtools
require(devtools) #call in devtools
install_github("brandynlucca/acousticTS@test-branch")
```

Full documentation is provided for all functions (e.g., `?jl` provides the help documentation for the built-in spherical Bessel function of the first kind).

Model syntax and setup

The TS of an individual scatterer can be calculated using the `SDWBA(...)` (see: `?SDWBA`) function which requires a `fluid-filled scatterer (FFS)` (see: `?FFS`) class object, which is a S4 object that comprises the target's position matrix (r_0), radius (a), material properties (g, h), orientation (θ), body curvature (ρ_c), body curve (a boolean `TRUE` or `FALSE`), body length (L), and number of cylinders (N). A sample shape can be called via `data(mcgehee)` (see: `?mcgehee`), which calls in the shape file and other model parameters reported in McGehee *et al.* (1998) for *Euphausia superba*, which has been commonly used for TS modelling efforts for krill. There are four ways to calculate TS using `SDWBA(...)`, which provides flexibility depending on users' needs, data format, and coding preferences:

1. Manually parameterizing the model.

```
#Let's call in the package
require(acousticTS)

#First we will create vectors for the x-, y-, and z-axis.
x <- seq(1,10,1)*1e-3; y <- rep(0,10); z <- c(seq(1,5,1),rev(seq(1,5,1)))*1e-4

#Now we will assign a radius vector
a <- z/2

#Material properties
g <- 1.036; h <- 1.0279

#Fill out the remaining model parameters
c <- 1500 #sound speed, m/s
freq <- 120e3 #frequency, Hz

#Now let's calculate the TS of this hypothetical shape at 120 kHz and ambient sound speed of 1500 m/s
SDWBA(c=c, frequency=freq, x=x, y=y, z=z, a=a, g=g, h=h)
#> [1] -114.0107
```

2. Manually creating a FFS-object using `FFSgenerate(...)` (see: `?FFSgenerate`) and `FFSwrite(...)` (see: `?FFSwrite`).

```
#We will use the same information we used before to generate our new scatterer
target <- FFSgenerate(x=x,y=y,z=z,a=a,g=g,h=h)

#We can look at the structure to get an idea about what the FFS scatterer object looks like
str(target)
#> Formal class 'FFS' [package "acousticTS"] with 9 slots
#>   ..@ rpos : num [1:3, 1:10] 1e-03 0e+00 1e-04 2e-03 0e+00 2e-04 3e-03 0e+00 3e-04 4e-03 ...
#>   .. ..- attr(*, "dimnames")=List of 2
#>   .. .. ..$ : chr [1:3] "x" "y" "z"
#>   .. .. ..$ : NULL
#>   ..@ a    : num [1:10] 0.00005 0.0001 0.00015 0.0002 0.00025 0.00025 0.0002 0.00015 0.0001 0.00005
#>   ..@ g    : num 1.04
#>   ..@ h    : num 1.03
#>   ..@ theta: num 1.57
#>   ..@ curve: logi FALSE
#>   ..@ pc   : num 0
#>   ..@ L    : num 0.01
```

```
#>   ..@ ncyl : int 10

#Now let's run the model on the object
SDWBA(target, c=c, frequency=freq)
#> [1] -114.0107

#If we want to save this object for later modelling efforts, we can create a data.frame and write a *.c
#Note that the FFSwrite(..., filename) defaults to saving this file in your current working directory.
#FFSwrite(target)
```

3. Calling in a *.csv formatted file to create a FFS-object using `FFSread(...)` (see: `?FFSread`).

```
#We will call in a sample krill shape reported by McGehee et al. (1998) from the GitHub repository, but
target_premade <- FFSread("https://raw.githubusercontent.com/brandynlucca/acousticTS/test-branch/data/m

#We can see that the file structure is the same as the previous "target" scatterer, just filled with di
str(target_premade)
#> Formal class 'FFS' [package "acousticTS"] with 9 slots
#>   ..@ rpos : num [1:3, 1:15] 0.0411 0 0 0.0395 0 ...
#>   ..@ a    : num [1:15] 0 0.000233 0.0007 0.001217 0.001455 ...
#>   ..@ g    : num 1.04
#>   ..@ h    : num 1.03
#>   ..@ theta: num 1.57
#>   ..@ curve: logi FALSE
#>   ..@ pc   : num 0
#>   ..@ L    : num 0.0411
#>   ..@ ncyl : int 15

#Now let's run the model on the object
SDWBA(target_premade, c=c, frequency=freq)
#> [1] -67.40053
```

4. Using both arbitrary and pre-generated FFS-object parameters.

```
#Let's say we are interested in a very different set of material properties!
SDWBA(target_premade, c=c, frequency=freq, g=1.01, h=1.005)
#> [1] -79.56342
```

FFS-object manipulation

In the previous section it was assumed that all of the scatterers we not curved and were straight from rostrum-to-telson. In these cases, a more traditional form of the model was used that discounts ρ_c. An animal can be curved by either setting `SDWBA(..., curve=T)` or using the `Shapely(...)` (see: `?Shapely(...)`). If `curve=TRUE`, the default ρ_c (i.e., `pc`) is 3.0 unless it is otherwise defined.

```
#Let's take a look at the str of our target. We can see that "curve" is set to False, and the radius of
str(target_premade)
#> Formal class 'FFS' [package "acousticTS"] with 9 slots
#>   ..@ rpos : num [1:3, 1:15] 0.0411 0 0 0.0395 0 ...
#>   ..@ a    : num [1:15] 0 0.000233 0.0007 0.001217 0.001455 ...
#>   ..@ g    : num 1.04
```

```
#>   ..@ h    : num 1.03
#>   ..@ theta: num 1.57
#>   ..@ curve: logi FALSE
#>   ..@ pc   : num 0
#>   ..@ L    : num 0.0411
#>   ..@ ncyl : int 15

#So we can force the object to do our bidding and curve it.
target_curved <- Shapely(target_premade, curve=T, pc=3.3)

#Now let's calculate the TS using both of the described methods, which will yield the same answer
SDWBA(target_curved, c=c, frequency=freq)
#> [1] -68.92558
SDWBA(target_premade, c=c, frequency=freq, curve=T, pc=3.3)
#> [1] -68.92558
```

The `Shapely(...)` function can also be used to change the orientation angle (`Shapely(..., theta=angle)`) and resize both the r_0 matrix and a vector based on a new length (`Shapely(..., length=new_length)`). The identical alternative is available by manually inputting a new orientation value into `SDWBA(...)`, but the shape manipulation requires `Shapely(...)`. For the purpose of convenience, the `degrad(...)` (see: `?degrad`) function is available for readily converting degrees to radians, and vice versa.

```
#First let's change the orientation of the object
target_angle <- Shapely(target_curved, theta=pi)
SDWBA(target_angle, c=c, frequency=freq)
#> [1] -75.02295
SDWBA(target_curved, c=c, frequency=freq, theta=pi)
#> [1] -75.02295

#If our orientation values are in degrees, we can use degrad(...) to convert!
angle_radians <- degrad(90,"deg") #degrees to radians
angle_radians
#> [1] 1.570796
angle_degrees <- degrad(angle_radians,"rad") #degrees to radians
angle_degrees
#> [1] 90

#We can also resize our target shape; the new length is in m
target_resize <- Shapely(target_angle, length=30e-3)
target_resize
#> An object of class "FFS"
#> Slot "rpos":
#>      [,1]         [,2]       [,3]        [,4]        [,5]       [,6]
#> [1,] 0.03 0.0288278843 0.02663194 0.023006878 0.020824876 0.01840187
#> [2,] 0.00 0.0000000000 0.00000000 0.000000000 0.000000000 0.00000000
#> [3,] 0.00 0.0007205438 0.00140502 0.001926098 0.001910328 0.00192566
#>            [,7]        [,8]       [,9]       [,10]       [,11]       [,12]
#> [1,] 0.01619190 0.013848692 0.01188047 0.010046800 0.008240512 0.006625002
#> [2,] 0.00000000 0.000000000 0.00000000 0.000000000 0.000000000 0.000000000
#> [3,] 0.00180271 0.001762846 0.00160872 0.001452185 0.001322226 0.001323467
#>            [,13]       [,14]       [,15]
#> [1,] 0.005203116 0.002327658 0.000000000
#> [2,] 0.000000000 0.000000000 0.000000000
```

```
#> [3,] 0.001617555 0.001936977 0.002785363
#>
#> Slot "a":
#>  [1] 0.0000000000 0.0001702612 0.0005107837 0.0008888337 0.0010623074
#>  [6] 0.0011358293 0.0012488501 0.0012141699 0.0012933137 0.0014894207
#> [11] 0.0013753778 0.0012925106 0.0010822394 0.0008652512 0.0004341223
#>
#> Slot "g":
#> [1] 1.0357
#>
#> Slot "h":
#> [1] 1.0279
#>
#> Slot "theta":
#> [1] 3.141593
#>
#> Slot "curve":
#> [1] TRUE
#>
#> Slot "pc":
#> [1] 3.3
#>
#> Slot "L":
#> [1] 0.0410898
#>
#> Slot "ncyl":
#> [1] 15

#So we have resized our target from a 41 mm krill to 30 mm, which adjusted both the position matrix, rp
SDWBA(target_resize, c=c, frequency=freq)
#> [1] -77.20959
```

Stochastic/vectorized model inputs

Up until now, we have not added in the phase deviation term (ϕ) that is set via `SDWBA(..., phase=0.0)`. As noted, this sets the standard deviation term to draw from a standard normal distribution that is drawn from `rnorm(1,0,phase)`. We can see that this can have a negligible-to-large effect depending on how large ϕ is set.

```
set.seed(1000)
#Normal DWBA, phase = 0.0
SDWBA(target_resize, c=c, frequency=freq)
#> [1] -77.20959
#phase = 0.04
SDWBA(target_resize, c=c, frequency=freq, phase=0.04)
#> [1] -77.13713
#phase = 0.07
SDWBA(target_resize, c=c, frequency=freq, phase=0.07)
#> [1] -76.86836
#phase = 0.22
SDWBA(target_resize, c=c, frequency=freq, phase=0.22)
#> [1] -77.51262
```

```
#phase = 0.50
SDWBA(target_resize, c=c, frequency=freq, phase=0.50)
#> [1] -79.27305
#phase = 0.77
SDWBA(target_resize, c=c, frequency=freq, phase=0.77)
#> [1] -76.9252
```

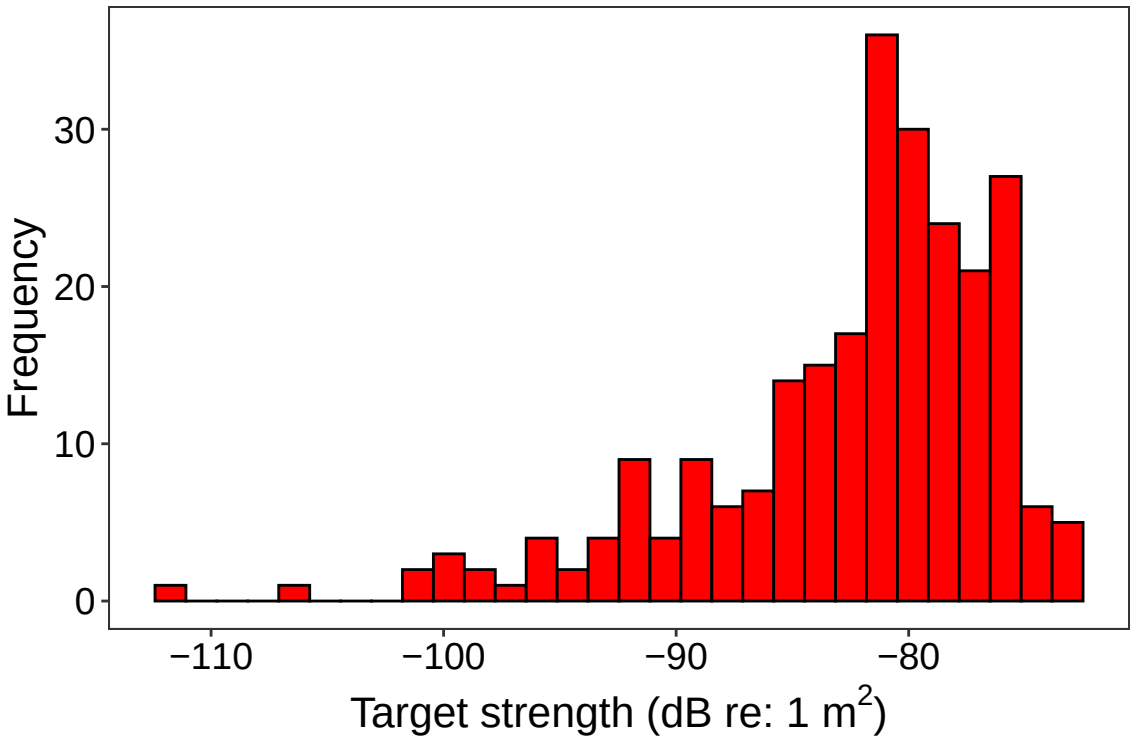
Because this is a stochastic variable, it makes more sense to simulate over more than a single iteration. This can be achieved using `SDWBA.sim(...)` (see: `?SDWBA.sim`), specifically defining `SDWBA.sim(..., nrep=NULL)`:

```
#Let's generate 10 iterations using the phase deviation, phi, of 0.77. This will print a bunch of value
set.seed(1000)
SDWBA.sim(target_resize, c=c, frequency=freq, phase=0.77, nrep=5)
#>   iteration    c frequency      g      h    theta  pc curve phase
#> 1         1 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 2         2 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 3         3 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 4         4 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 5         5 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#>      length        TS
#> 1 0.0410898 -77.15729
#> 2 0.0410898 -77.62018
#> 3 0.0410898 -80.10818
#> 4 0.0410898 -81.54719
#> 5 0.0410898 -76.92520

#This can also be parallelized, if desired and available to your computer by setting SDWBA.sim(..., par
set.seed(1000)
SDWBA.sim(target_resize, c=c, frequency=freq, phase=0.77, nrep=5, parallel=T, n.cores=6)
#>   iteration    c frequency      g      h    theta  pc curve phase
#> 1         1 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 2         2 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 3         3 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 4         4 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#> 5         5 1500    120000 1.0357 1.0279 3.141593 3.3  TRUE  0.77
#>      length        TS
#> 1 0.0410898 -78.28152
#> 2 0.0410898 -76.79851
#> 3 0.0410898 -75.71157
#> 4 0.0410898 -79.74627
#> 5 0.0410898 -77.26472
```

Note that `SDWBA.sim(...)` requires a FFS-class object input. Of course not all targets/animals comprise the same model parameters, or comprise a distribution of values such as various orientations, body curvatures, and material properties. We can also use `SDWBA.sim(...)` to draw from distributions for these other parameters:

```
#Using our same curved shape above, let's play around with a few distributions
n <- 5 #number of animal parameters we want to simulate
lengths <- rnorm(n,mean=30,sd=5)*1e-3 #Normal distribution of lengths
g <- rnorm(n,mean=target_resize@g,sd=1e-3) #Normal distribution for g
h <- rnorm(n,mean=target_resize@h,sd=1e-3) #Normal distribution for h
```

```
theta <- c(pi/2,pi/4) #Play around with two different orienation angles
pd <- 0.77 #phase deviation

#Now let's simulate over every combination of these values
ts_sim <- SDWBA.sim(target_resize, c=c, frequency=freq, phase=pd, g=g, h=h, length=lengths, theta=theta
head(ts_sim)
#>   iteration    c frequency        g        h    theta  pc curve phase
#> 1         1 1500    120000 1.035315 1.026918 1.570796 3.3  TRUE  0.77
#> 2         1 1500    120000 1.035224 1.026918 1.570796 3.3  TRUE  0.77
#> 3         1 1500    120000 1.036420 1.026918 1.570796 3.3  TRUE  0.77
#> 4         1 1500    120000 1.035681 1.026918 1.570796 3.3  TRUE  0.77
#> 5         1 1500    120000 1.034327 1.026918 1.570796 3.3  TRUE  0.77
#> 6         1 1500    120000 1.035315 1.027346 1.570796 3.3  TRUE  0.77
#>       length        TS
#> 1 0.02777111 -80.26574
#> 2 0.02777111 -79.30540
#> 3 0.02777111 -78.04290
#> 4 0.02777111 -79.63667
#> 5 0.02777111 -79.27447
#> 6 0.02777111 -78.44479

#We can then look at the distribution of values to see our simulated TS results!
require(ggplot2)

ggplot(data=ts_sim, aes(x=TS)) + geom_histogram(fill="red", color="black") +
  theme_bw() + theme(text=element_text(size=16), axis.text=element_text(size=14, color="black"),
                     panel.grid=element_blank()) +
  labs(x=expression(paste("Target strength (dB re: 1 ",m^2,")")),
       y="Frequency")
#> `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

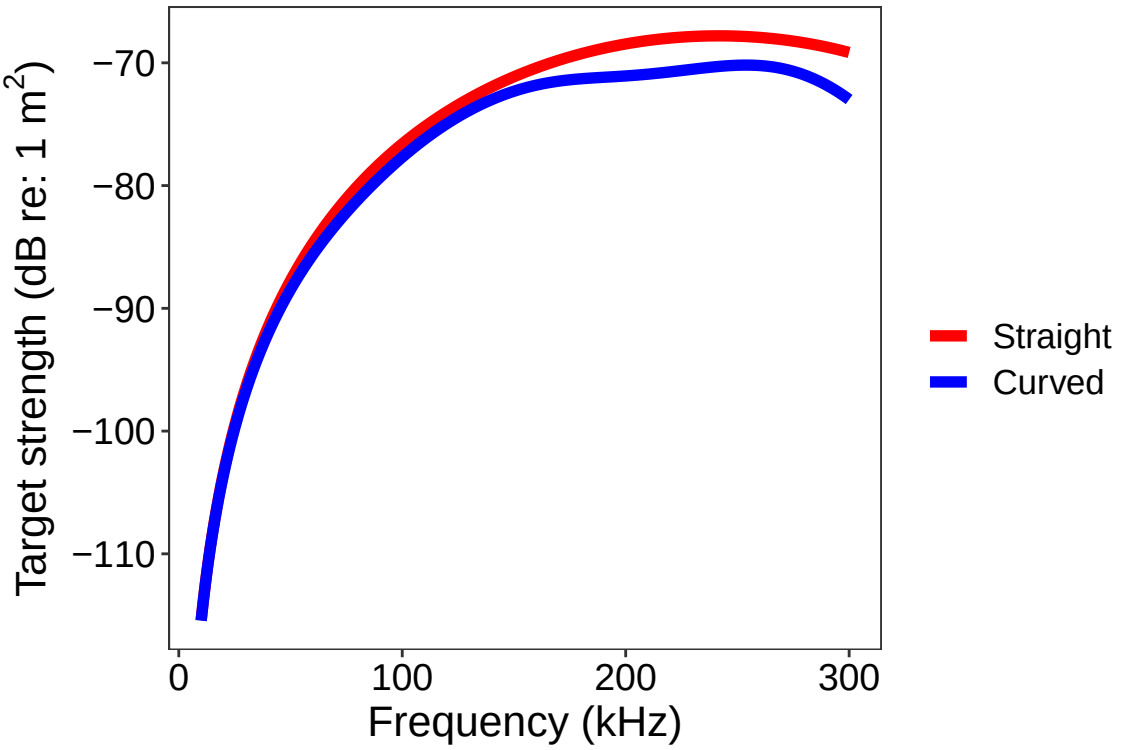


The output of `SDWBA.sim(...)` is by default a data.frame; however, if only a single aggregate statistic is needed, then `SDWBA.sim(..., aggregate=c("mean","median","maximum","minimum"))` can be used to generate just the mean, median, maximum, minimum, or any combination of those descriptive statistics.

```
SDWBA.sim(target_resize, c=c, frequency=freq, phase=pd, g=g, h=h, length=lengths, theta=theta, parallel=
#>      Stat        TS
#> 2   Mean -79.60582
#> 3 Median -81.10827
```

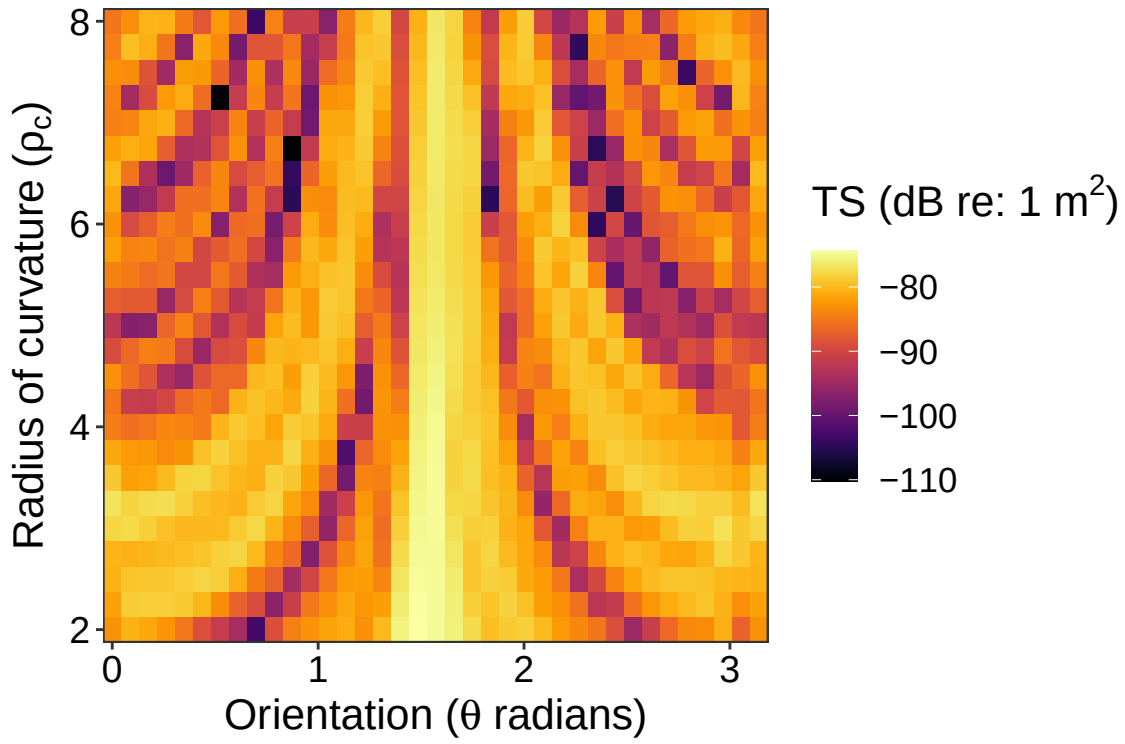
Example: TS-Frequency spectrum comparison between curved and non-curved animal

```
#We will compare the TS-frequency spectra of the same target when it is curved and non-curved
curve = c(T,F) #curved and non-curved
freq <- seq(10,300,1)*1e3 #frequencies to simulate, Hz
theta <- pi/2 #broadside incidence
ts_freq <- SDWBA.sim(target_resize, c=c, frequency=freq, curve=curve, theta=theta, parallel=T)

#And we can plot!
ggplot(data=ts_freq, aes(y=TS,x=frequency*1e-3)) + geom_path(aes(color=curve), size=2) +
  theme_bw() + theme(text=element_text(size=16), axis.text=element_text(size=14, color="black"),
                     panel.grid=element_blank()) +
  scale_color_manual(values=c("red","blue"), labels=c("Straight","Curved")) +
  labs(y=expression(paste("Target strength (dB re: 1 ",m^2,")")),
       x="Frequency (kHz)", color="")
```



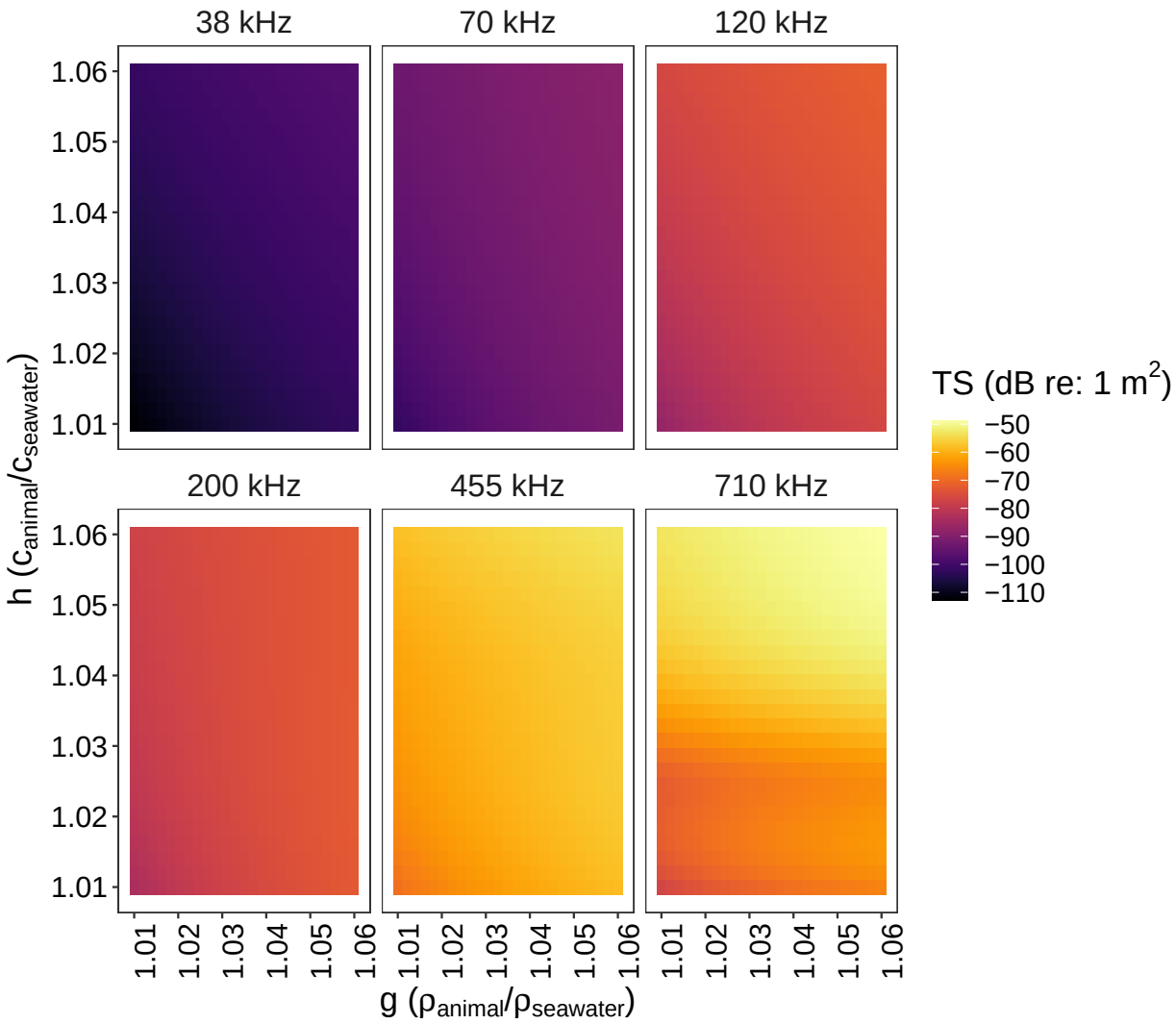
Example: Effect of orientation and body curvature on TS.

```
#Let's see how body curvature, pc, and orientation, theta, influences TS at 120 kHz
freq <- 120e3
theta <- seq(0,180,5) #degrees, 0 to 180 degrees at 5 degree resolution, or 0 to pi radians
thetar <- degrad(theta, "deg") #convert to radians
curve = T #set animal to be curved
pc <- seq(2.0,8.0,0.25) #curvature
ts_orfreq <- SDWBA.sim(target_resize, c=c, frequency=freq, theta=thetar, curve=curve, pc=pc, parallel=T

#Now let's plot
require(viridis)
ggplot(data=ts_orfreq, aes(x=theta, y=pc, fill=TS)) + geom_tile() + theme_bw() +
  theme(text=element_text(size=16), axis.text=element_text(size=14, color="black"),
        panel.grid=element_blank()) +
  scale_fill_viridis(option="B") +
  coord_cartesian(expand=c(0,0)) +
  labs(x=expression(paste("Orientation (",theta," radians)")),
       y=expression(paste("Radius of curvature (",rho[c],")")),
       fill=expression(paste("TS (dB re: 1 ",m^2,")")))
#> Warning in if (!expand) {: the condition has length > 1 and only the first
#> element will be used

#> Warning in if (!expand) {: the condition has length > 1 and only the first
#> element will be used
```



Example: Effect of material properties on TS at four discrete frequencies.

```
#Let's see how variation in g and h influences a target at 38, 70, 120, 200, 455, and 710 kHz.
freq <- c(38, 70, 120, 200, 455, 710)*1e3 #Hz
g <- seq(1.01,1.06,length.out=25) #g
h <- seq(1.01,1.06,length.out=25) #h
ts_gh <- SDWBA.sim(target_resize, c=c, frequency=freq, g=g, h=h, parallel=T)

#Now let's plot
ts_gh$frequency <- ts_gh$frequency * 1e-3 #convert from Hz to kHz
ts_gh$label <- paste(ts_gh$frequency," kHz",sep="")
ts_gh$label <- factor(ts_gh$label, levels=c("38 kHz","70 kHz","120 kHz","200 kHz","455 kHz","710 kHz"))
ggplot(data=ts_gh, aes(x=g, y=h, fill=TS)) + geom_tile() + theme_bw() +
  theme(text=element_text(size=16), axis.text=element_text(size=14, color="black"),
        panel.grid=element_blank(), strip.background=element_blank(), strip.text=element_text(size=15),
        axis.text.x=element_text(angle=90)) +
  scale_fill_viridis(option="B") +
  facet_wrap(~label) +
  labs(x=expression(paste("g (",rho[animal],"/",rho[seawater],")")),
       y=expression(paste("h (",c[animal],"/",c[seawater],")")),
       fill=expression(paste("TS (dB re: 1 ",m^2,")")))
```


38 kHz
70 kHz
120 kHz
200 kHz
455 kHz
710 kHz
h (c_animal/c_seawater)
g (ρ_animal/ρ_seawater)
1.01
1.02
1.03
1.04
1.05
1.06
TS (dB re: 1 m²)
−50
−60
−70
−80
−90
−100
−110