fishmechr

```
# library(fishmechr)
devtools::load_all()
#> i Loading fishmechr
```

```
library(tidyverse)
library(patchwork)
```

Computing kinematic parameters

To compute midline swimming kinematics accurately, follow the steps below.

1. Compute the arc length along the curve of the body.
2. If the points are not consistently at the same arc length, interpolate the points so that each one is at a consistent point on the body. It is often useful to do some smoothing as part of this step.
3. Compute a center location. Ideally this is the center of mass, but it could be approximated in a variety of ways.
4. Compute the body curvature or excursion relative to a central axis, or both.
5. Estimate the phase of the oscillation at each point. You can either detect peaks and zero crossings or use a mathematical technique called the Hilbert transform.
6. Use temporal and spatial derivatives of the phase to estimate the oscillation frequency, body wavelength, and body wave speed.
7. Based on the phase, identify individual undulation cycles. Within each cycle, you can identify the range of body excursion or curvature to compute the amplitude.

1. Compute arc length

Most kinematic variables are best specified in terms of arc length s, the distance along the body from the head to a particular point i:

$$s_i = \sum_{j=2}^{i} \left[(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2 \right]$$

Using arc length is better than something like the x coordinate for two reasons. First, often a fish does not swim precisely along the x axis, which means that the points would need to be rotated. Second, many fish swim with relatively large amplitude motions, which means that the distance along the curve is larger than the distance along the swimming direction, particularly near the tail where amplitudes are higher.

2. Interpolate points for a consistent position

Ideally, we want each point to represent a consistent location on the fish's body. Particularly with fish that do not have clear landmarks along the body, we may be able to mark the middle easily, but at points that may slide along the fish's length. We can use a spline to interpolate points at a consistent location.

Digitized points often have some error. As part of the interpolation process, we can use a smoothing spline to smooth out some of that error.

3. Estimate a center location

The center location is a weighted average location. For a midline defined by x and y coordinates (x_i, y_i), there are four main ways to identify the center location.

1. If m_i is the mass of segment located between (x_i, y_i) and (x_{i+1}, y_{i+1}), then

$$x_{com} = \frac{1}{2M} \sum_{i=1}^{n-1} m_i \left(x_{i+1} + x_i\right)$$

$$y_{com} = \frac{1}{2M} \sum_{i=1}^{n-1} m_i \left(y_{i+1} + y_i\right)$$

   where M is the total mass of the fish. If you know the mass distribution but not the true masses of segments, you could also use m_i as the fraction of the total mass in segment i and M would be 1.

2. If you do not know the mass distribution, then you can approximate m_i in several ways. The best is to use the width and height of the body. If the body, without the fins, has an elliptical cross section, where w_i is the horizontal width and h_i is the dorso-ventral height at point i, then the volume of the segment from (x_i, y_i) and (x_{i+1}, y_{i+1}) is

$$V_i = \pi \Delta s_i \left(w_i h_i + \tfrac{1}{2}\Delta w_i h_i + \tfrac{1}{2}\Delta h_i w_i + \tfrac{1}{3}\Delta w_i \Delta h_i\right)$$

   where $\Delta s_i = s_{i+1} - s_i$, $\Delta w_i = w_{i+1} - w_i$, and $\Delta h_i = h_{i+1} - h_i$. Then we approximate $m_i \approx \rho V_i$, where ρ is the density of the fish, which we assume here to be constant, so that

$$x_{com} = \frac{\sum_{i=1}^{n-1} \left[V_i \left(x_{i+1} + x_i\right)\right]}{2\sum_{i=1}^{n-1} V_i}$$

$$y_{com} = \frac{\sum_{i=1}^{n-1} \left[V_i \left(y_{i+1} + y_i\right)\right]}{2\sum_{i=1}^{n-1} V_i}$$

3. Often the width is visible from a camera from above or below, but the height is not known. In this case, a reasonable approximation is that $m_i \propto w_i$, so that

$$x_{com} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$$

$$y_{com} = \frac{\sum_{i=1}^{n} w_i y_i}{\sum_{i=1}^{n} w_i}$$

4. Choose one point along the body to use as the center. This could be the snout or a point close to the center of mass. This has been called the "stretched-straight center of mass", but tends to give inaccurate estimates of velocity or acceleration.

4. Estimate curvature or lateral excursion

Curvature

The 2D curvature κ of the midline in the horizontal plane is often a useful variable to compute. It can be thought of in two different ways. First, it is the inverse of the radius of curvature: the radius of a circle drawn through three successive points. The smaller the radius of curvature, the sharper the body bend, and the larger the value of κ. This estimate for curvature is defined by the following equation

$$\kappa = \left[\frac{\partial x}{\partial s}\frac{\partial^2 y}{\partial s^2} - \frac{\partial y}{\partial s}\frac{\partial^2 x}{\partial s^2}\right]\left[\left(\frac{\partial x}{\partial s}\right)^2 + \left(\frac{\partial y}{\partial s}\right)^2\right]^{-3/2}$$

Second, it is the spatial derivative of the angle of each segment. If a segment at arc length s has an angle θ to the horizontal axis, then the curvature is

$$\kappa = \frac{\partial \theta}{\partial s}$$

The angle for segment i is $\theta_i = \tan^{-1}(y_{i+1} - y_i, x_{i+1} - x_i)$.

Although both formulas are mathematically equivalent, they have slightly different properties depending on the measurement error on the x and y positions.

Lateral excursion

One can also estimate the body phase, and then the wavelength and wave speed, based on the excursion of the body relative to a primary axis. We suggest using the singular value decomposition (SVD) to estimate the primary axis, then using a low-pass filter to remove any oscillations at the tail beat frequency or higher. If you have a matrix of x and y coordinates of points along the body at a specific time, $\mathbf{X}$

$$\mathbf{X}_{n\times 2} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}$$

where the subscript indicates the size of the matrix (n points along the body by 2).

First, center each axis by subtracting the location of the center of mass or the mean of each column, to produce a matrix $\mathbf{X^C}$, centered around 0. Then the singular value decomposition allows you to write the matrix in the form

$$\mathbf{X}^C_{n\times 2} = \mathbf{U}_{n\times n}\mathbf{\Sigma}_{n\times 2}\mathbf{V}^T_{2\times 2}.$$

The matrix $\mathbf{V}_{2\times 2}$ then represents the principal axes of the body in that frame. The matrix can be estimated at each time point, to produce a time-varying matrix $\mathbf{V}(t)$.

Assuming the amplitude is relatively small, the first column of $\mathbf{V}$ represents a unit vector pointing along the primary axis of the body (which we call $\hat{\mathbf{a}}(t)$ and the second column is a unit vector normal to the primary axis.

We suggest using a low pass filter with a cutoff frequency below the tail beat frequency to smooth the components of the $\hat{\mathbf{a}}(t)$ vector, making sure to normalize it after smoothing. See implementation details below.

5. Estimate phase

To estimate the undulation frequency, body wavelength, and wave speed, consider a simple equation for the midline. Fish swim using a traveling wave of curvature κ_s (or lateral excursion) defined at an arc length s along the body as

$$\kappa(s,t) = A(s)\cos\left(2\pi\left[\frac{s}{\lambda} - ft\right]\right)$$

where $A(s)$ is the wave amplitude, k is the wave length, f is the oscillation frequency, and t is time. In this case, the phase of the oscillation $\phi(s,t)$ at a particular point along the body is approximately equal to the argument of the cosine, $2\pi\left(s/\lambda - ft\right)$. If the amplitude A varies over quickly over time or space, this approximation may not be exactly correct.

If we can estimate the phase accurately, then we can use a spatial (i.e., with respect to s) or temporal derivative to estimate λ, the body wavelength, or f the undulation frequency.

Below, we describe two methods for estimating phase.

Use the Hilbert transform

One convenient and relatively robust way to estimate the phase of each body point involves the use of the Hilbert transform **H**, a procedure that uses the Fourier transform to estimate a periodic signal that is 90deg shifted relative to another. In essence, given a cosine signal, the Hilbert transform returns the sine with the same amplitude and frequency. The utility of this operation is that it lets us estimate the "analytic signal' ', a complex-valued signal where the magnitude of the complex number is the amplitude of the wave and the phase angle of the complex number is the phase of the signal.

We can then take the Hilbert transform to estimate the analytic signal for curvature or lateral excursion

$$\kappa^*(s,t) = \kappa(s,t) + i\mathbf{H}\{\kappa(s,t)\}$$

where i is the imaginary number. (Note that the `hilbert` function in R and Python returns the full analytic signal, not just the imaginary component as written above). For a complex number $C = a + ib$, the magnitude is denoted by $||C|| = \left[a^2 + b^2\right]^{1/2}$ and the phase angle is denoted by $\angle C = \tan^{-1}(b/a)$.

One can also estimate the analytic signal for the lateral excursion of the body $z(s,t)$, where z is the lateral position of a point on the body, relative to the overall axis of the body. We suggest using the singular value decomposition to estimate the central body axis and then using a low-pass filter with a cutoff frequency lower than the tail beat frequency to remove the tail beat oscillation, as described above.

The analytic signal thus provides an estimate of the phase can be estimated as a continuous function of both time and position along the body. Other techniques for estimating phase require identifying particular features in the signal (such as peaks or zero crossings) and therefore do not estimate phase as a continuous signal. The estimated phase $\hat{\phi}$ is thus

$$\hat{\phi}(s,t) = \angle\kappa^*(s,t).$$

For a traveling wave, this phase, as estimated here, is equal to the argument of the cosine function from the traveling wave equation above, $\hat{\phi}(s,t) = 2\pi[s/\lambda - ft]$.

Numerical considerations for the Hilbert transform

The Hilbert transform only works well for this analysis with signals that are centered around zero and consist of many relatively smooth tailbeats. This is why we suggest using the curvature κ, rather than something like lateral position or the y coordinate of the body. To use the Hilbert transform on a lateral position, it is important to subtract a baseline value or use a high pass filter to ensure that the signal is centered around zero.

Similarly, if the signal is noisy, the phase $\hat{\phi}$ will not increase steadily and the derivatives used to estimate $\hat{f}$ and $\hat{\lambda}$ will not be meaningful. It is best to filter the input signal using a bandpass or low pass filter so that the oscillations are smooth.

In most programming languages, one should use a function `atan2` to estimate phase (not `atan`), because it gives an angle that ranges around the full circle (rather than 0 to 180 degrees). However, the output of `atan2` will jump (usually from π to $-\pi$) as t or s increase. To estimate frequency or wavelength, before performing the derivatives, one should estimate a smoothly increasing phase using a function `unwrap`, which searches for jumps and removes them.

Detect peaks and zero crossings

We can also estimate a phase by detecting specific features in the oscillation, such as peaks or zero crossings, and then interpolating a continuous value for the phase using a spline curve. For a cosine function, as above, a positive peak has phase $\hat{\phi} = 0$, a downward zero crossing has phase $\pi/2$, a negative peak (i.e., a trough) has phase π, and an upward zero crossing has phase $3\pi/2$. By identifying these features and their corresponding phases, one can then interpolate a continuous phase.

Using this method requires careful error checking. In particular, peaks can be identified erroneously, particularly if there is noise in the signal. We recommend using a strong smoothing filter on the curvature or lateral excursion before estimating phase. See examples and numerical details below.

6. Use the phase to estimate frequency and wavelength

We can then use the estimated phase to compute the frequency and wavelength by taking derivatives in time or space, respectively,

$$\hat{f}(t) = \frac{1}{2\pi}\frac{\partial}{\partial t}\hat{\phi}(s,t)$$

Estimated this way, frequency should be the same at every point along the body. Therefore, you can average $\hat{f}$ across the body or choose a single point along the body (usually the tail) to use as the estimate of frequency.

$$\hat{\lambda}_s(t) = 2\pi\left(\frac{\partial}{\partial s}\hat{\phi}(s,t)\right)^{-1}$$

The body wave speed $\hat{V}$ is the product of the two:

$$\hat{V}(t) = \hat{\lambda}(s,t)\hat{f}(t)$$

7. Quantify swimming parameters on a cycle-by-cycle basis

Once you have estimated a good undulation phase, it is straightforward to quantify other parameters that vary every cycle. Usually, we need to define an overall phase of the entire body oscillation. It's best to choose the most reliable phase estimate, which is usually the phase of the tail tip or a point near the tail (e.g., $\hat{phi}(L,t)$, where L is the body length).

As phase increases past 2π, it will jump back to 0. Most programming languages have a functioned called `unwrap` that looks for those jumps and removes them, producing a steadily increasing phase, which we will refer to as $\hat{\Phi}_c(t)$. Then the cycle number is

$$C(t) = \left\lfloor \hat{\Phi}_c(t)/(2\pi) \right\rfloor$$

where the L brackets denote the `floor` operation, or rounding down to the next lowest integer value.

For example, to find the body amplitude, search within each cycle to find the range of motion for each body point, and divide by two. See details below.

Example analysis of a lamprey data set

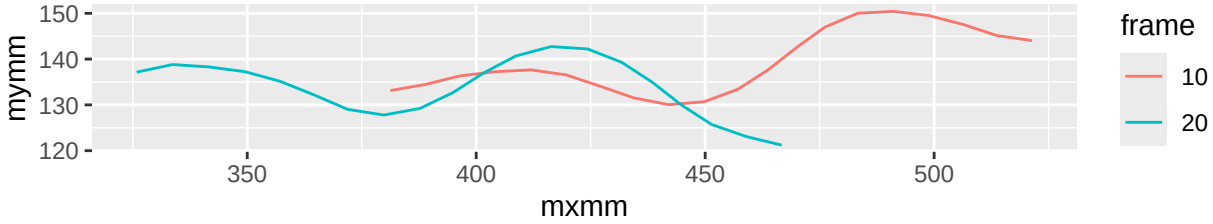
The data set `lampreydata` is included in the `fishmechr` package. Most swimming kinematics data sets should have a similar structure: * A time or frame column. It's often good to keep both in the data set. (Here, these are in `t` and `frame`) * A column that identifies the point on the body. This could be a factor variable, with names of each body part (like returned by Sleap or DeepLabCut), as long as there is a clear order from head to tail. Or it could be a numeric variable, with points numbered from 1 to n, where 1 is the snout and n is the tail. (here this is in `point`). * x and y coordinates of the point. Currently, these algorithms are only designed to work on 2D movements in the horizontal plane. (here these are in `mxmm` and `mymm`, indicating "midline" x and y points in mm)

```
head(lampreydata)
#> # A tibble: 6 x 23
#> # Groups:   point [6]
#>       t frame point  mxmm  mymm arclen0 arclen mxmm_s mymm_s width  xcom  ycom
#>   <dbl> <int> <int> <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>
#> 1  0.02     1     1    NA    NA      NA   0        NA     NA  2.48    NA    NA
#> 2  0.02     1     2    NA    NA      NA   8.13     NA     NA  2.61    NA    NA
#> 3  0.02     1     3    NA    NA      NA  16.3      NA     NA  2.75    NA    NA
#> 4  0.02     1     4    NA    NA      NA  24.4      NA     NA  2.81    NA    NA
#> 5  0.02     1     5    NA    NA      NA  32.5      NA     NA  2.83    NA    NA
```

```
#> 6  0.02     1     6    NA    NA      NA  40.7      NA     NA  2.85    NA    NA
#> # i 11 more variables: curve_ang <dbl>, curve_xy <dbl>, mxmm_ctr <dbl>,
#> #   mymm_ctr <dbl>, swimaxis_x <dbl>, swimaxis_y <dbl>, exc_x <dbl>, exc <dbl>,
#> #   ph_c <dbl>, ph_e <dbl>, cycle <dbl>
```

This plot shows the midlines in two frames.

```
lampreydata |>
  filter(frame %in% c(10, 20)) |>
  mutate(frame = factor(frame)) |>
  ggplot(aes(x = mxmm, mymm, color = frame, group = frame)) +
  geom_path() +
  coord_fixed()
```

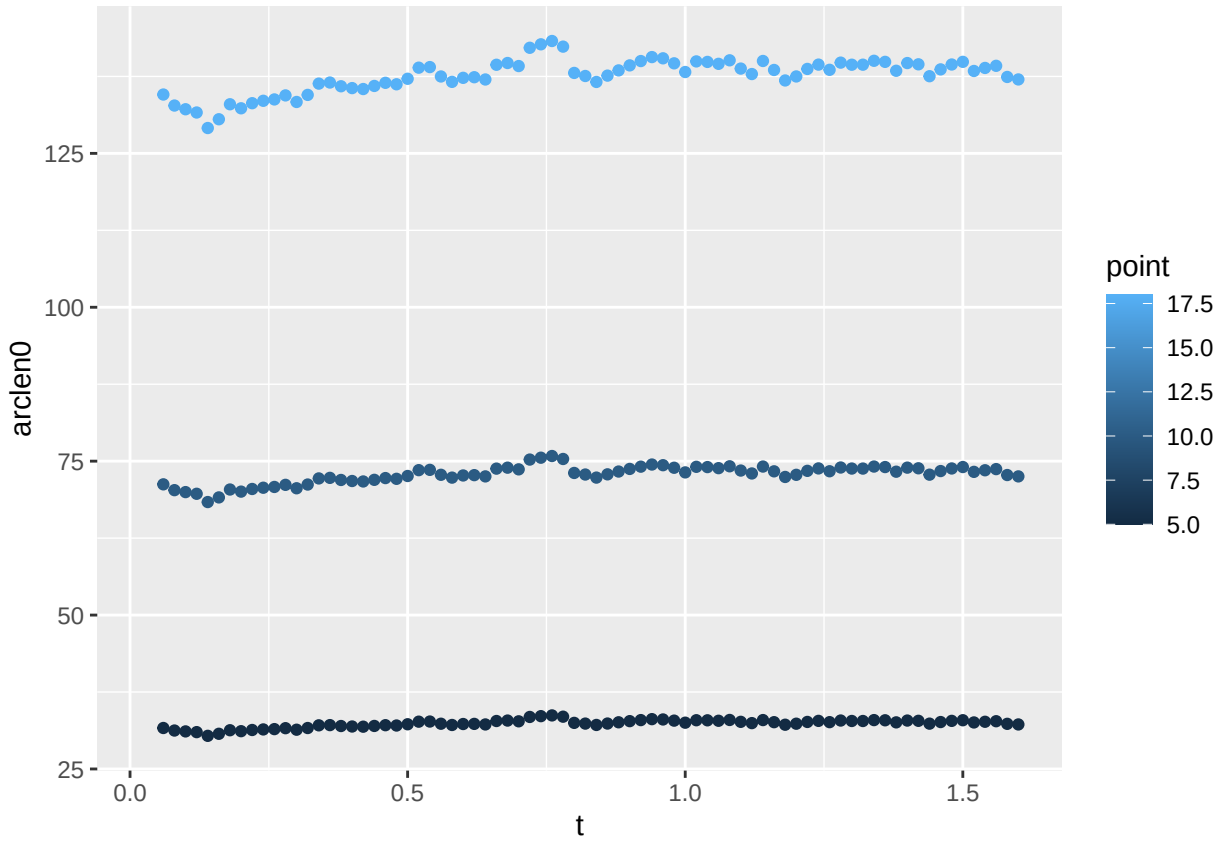


Compute arc length

```
lampreydata <-
  lampreydata |>
  group_by(frame) |>
  mutate(arclen0 = arclength(mxmm, mymm))
```

Here we plot the arc length of a few points along the body. We can see that they're generally in the same location, but not exactly, so we'll need to interpolate so that they're in the same place.

```
lampreydata |>
  ungroup() |>
  filter(point %in% c(5, 10, 18)) |>
  ggplot(aes(x = t, y = arclen0, color = point)) +
  geom_point()
#> Warning: Removed 6 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```

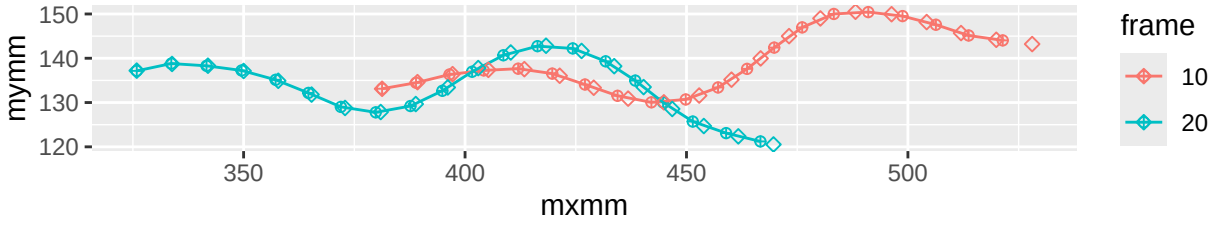


Here's the interpolation.

```
lampreydata <- lampreydata |>
  interpolate_points_df(arclen0, mxmm, mymm, spar = 0.2,
                        tailmethod = 'extrapolate',
                        .out = c(arclen='arclen', xs='mxmm_s', ys='mymm_s'))
#> Warning in check.out(.data, .out, .out_default = c(arclen = "arclen_s", : Columns
#> (arclen,mxmm_s,mymm_s) will be overwritten
```

Now the points are always at the same arc length. Sometimes this requires extrapolating to get the tail position.

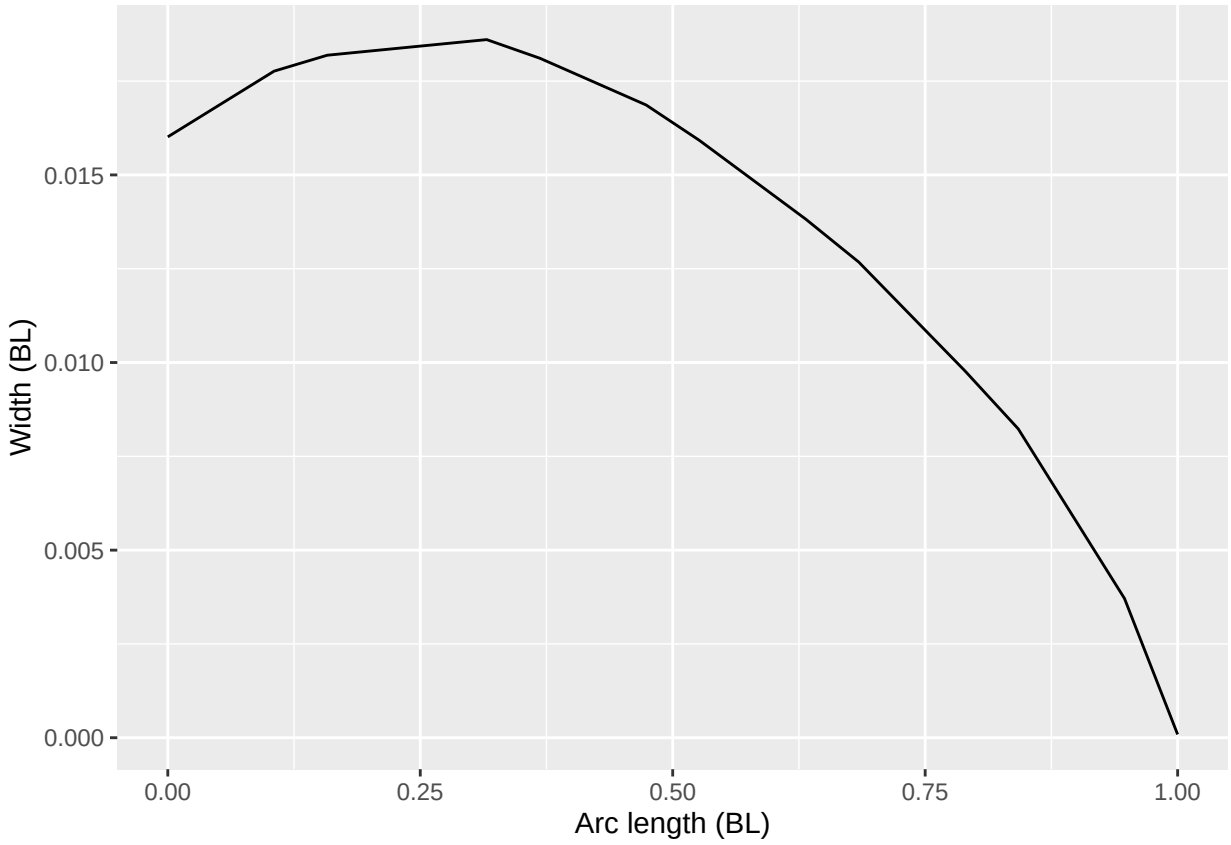
```
lampreydata |>
  filter(frame %in% c(10, 20)) |>
  mutate(frame = factor(frame)) |>
  ggplot(aes(x = mxmm, mymm, color = frame, group = frame)) +
  geom_point(shape = 10) +
  geom_path() +
  geom_point(aes(x = mxmm_s, y = mymm_s), shape = 5) +
  coord_fixed()
```



Get the width

I have digitized the width of an ammocoete from the ventral view. We assume here that the width is the distance from one lateral edge to the other (like a diameter), not the distance from the center to an edge (like a radius). Here it's expressed in terms of body lengths, both along the body and for the width itself.

```
fishwidth |>
  ggplot(aes(x = s, y = ammowidth)) +
  geom_path() +
  labs(x = 'Arc length (BL)', y = 'Width (BL)')
```



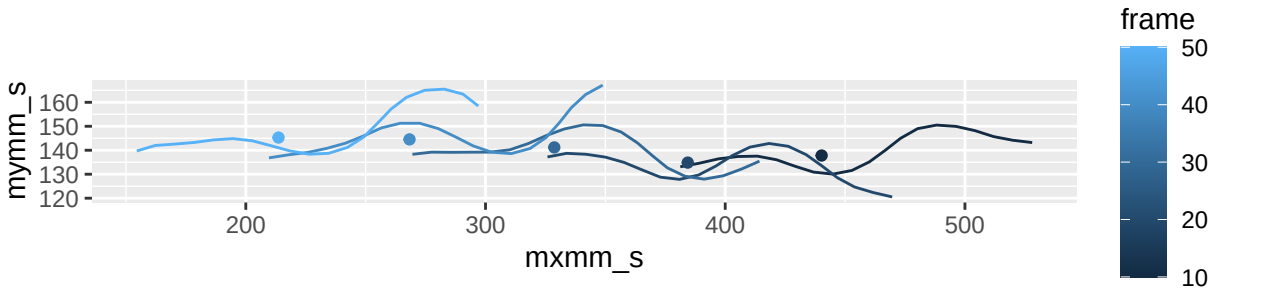

```
lampreydata <-
  lampreydata |>
  group_by(frame) |>
  mutate(width = interpolate_width(fishwidth$s, fishwidth$ammowidth, arclen))
```

Compute the center of mass

```
lampreydata <-
  lampreydata |>
  get_midline_center_df(arclen0, mxmm_s,mymm_s, width=width)
#> Warning in check.out(.data, .out, .out_default = c(xctr = "xcom", yctr = "ycom"), :
#> Columns (xcom,ycom) will be overwritten
#> Estimating center of mass based on width
```

```
lampreydata |>
  filter(frame %in% c(10, 20, 30, 40, 50)) |>
  ggplot(aes(x = mxmm_s, y = mymm_s, color = frame)) +
```

```
  geom_path(aes(group = frame)) +
  geom_point(data = ~filter(.x, point == 1), aes(x = xcom, y = ycom)) +
  coord_fixed()
```



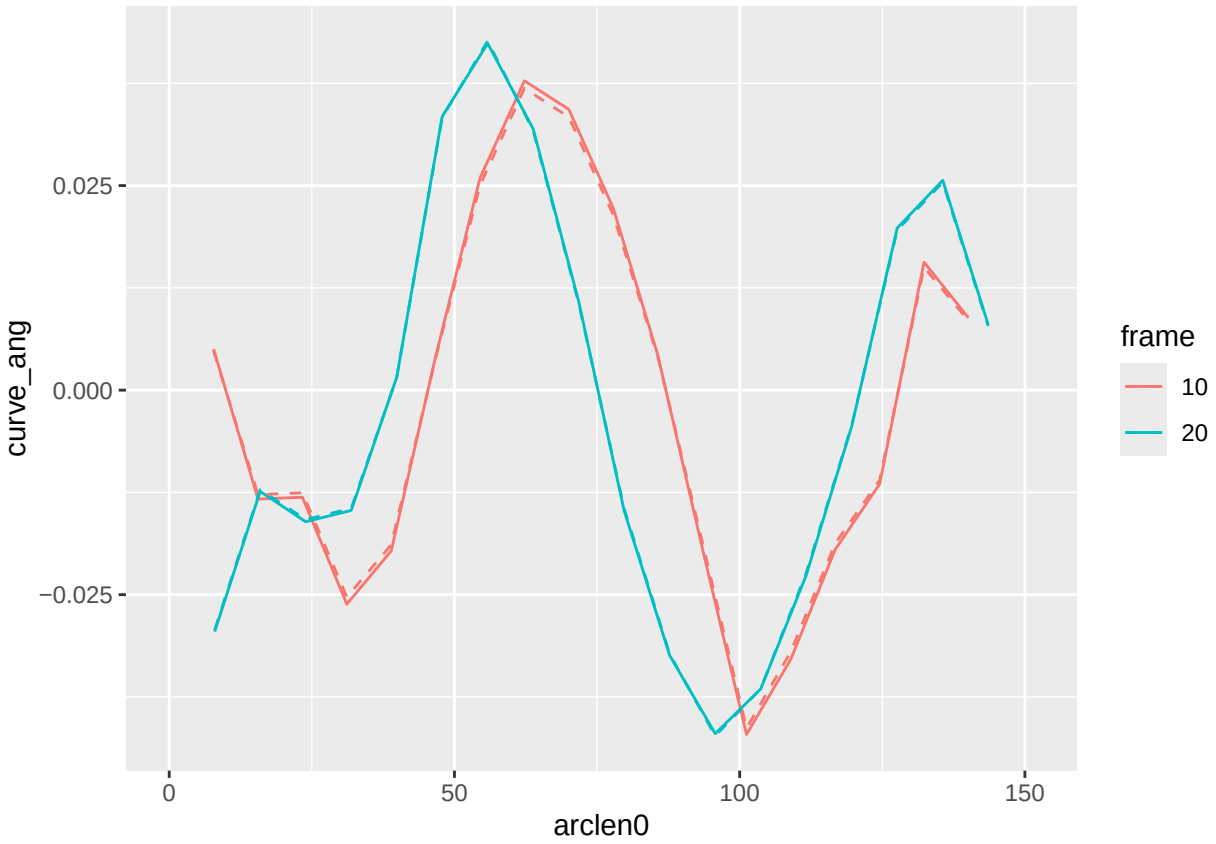
Compute curvature

This computes the curvature by the two different methods.

```
lampreydata <-
  lampreydata |>
  group_by(frame) |>
  mutate(curve_ang = curvature(arclen0, mxmm_s, mymm_s),
         curve_xy = curvature(arclen0, mxmm_s, mymm_s, method="xy"))
```

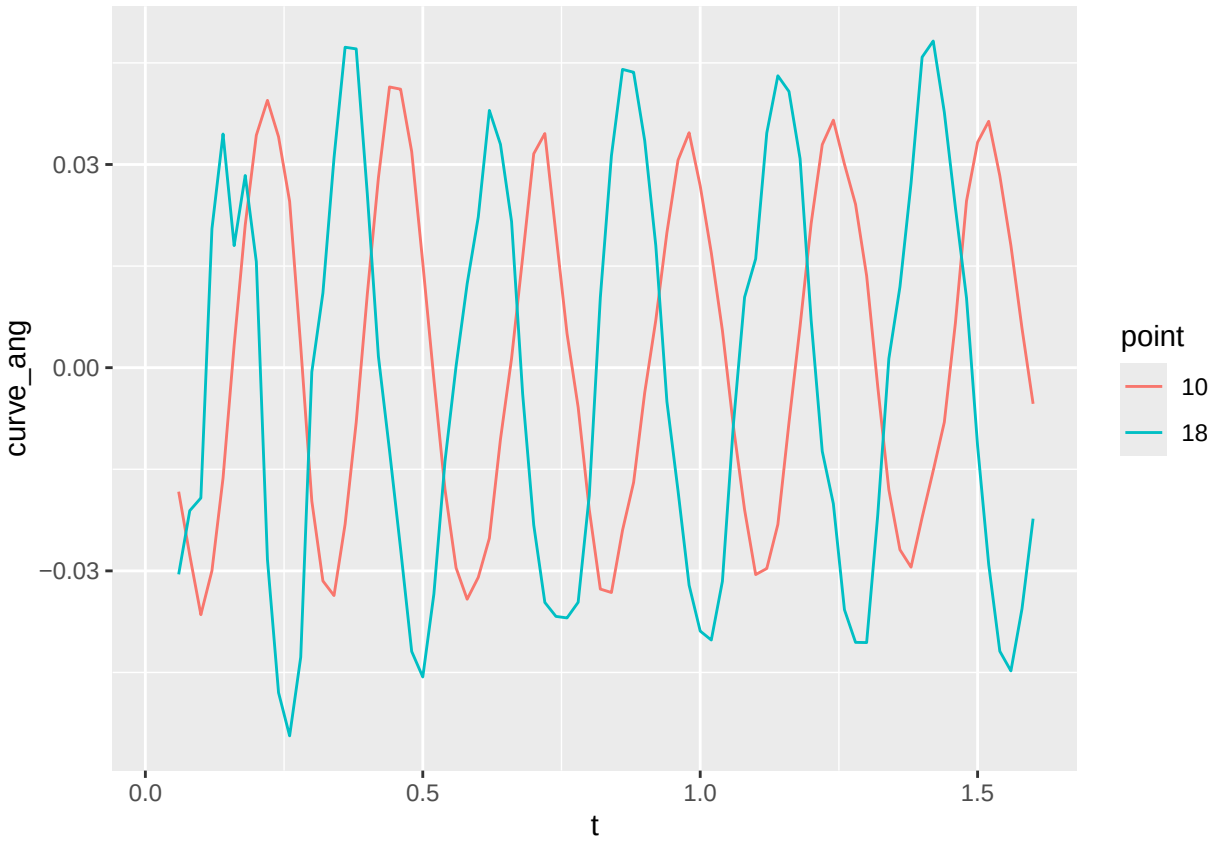
Compare the results. Here they aren't very different, although the "xy" method gives very slightly higher peaks.

```
lampreydata |>
  filter(frame %in% c(10, 20)) |>
  mutate(frame = factor(frame)) |>
  ggplot(aes(x = arclen0, color = frame, group = frame)) +
  geom_path(aes(y = curve_ang), linetype='solid') +
  geom_path(aes(y = curve_xy), linetype='dashed')
#> Warning: Removed 4 rows containing missing values or values outside the scale range
#> (`geom_path()`).
#> Removed 4 rows containing missing values or values outside the scale range
#> (`geom_path()`).
```



Plot curvature as a function of time for two different points along the body. For the Hilbert analysis to work well, these should be close to sinusoidal and centered around 0.

```
lampreydata |>
  filter(point %in% c(10, 18)) |>
  mutate(point = factor(point)) |>
  ggplot(aes(x = t, y = curve_ang, color = point)) +
  geom_path()
#> Warning: Removed 4 rows containing missing values or values outside the scale range
#> (`geom_path()`).
```

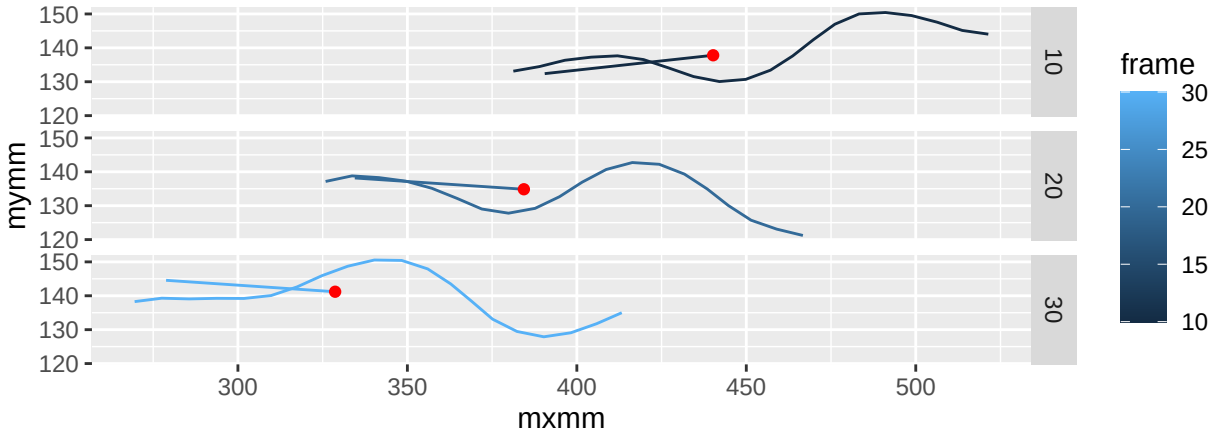


Get excursion

This extracts the central swimming axis.

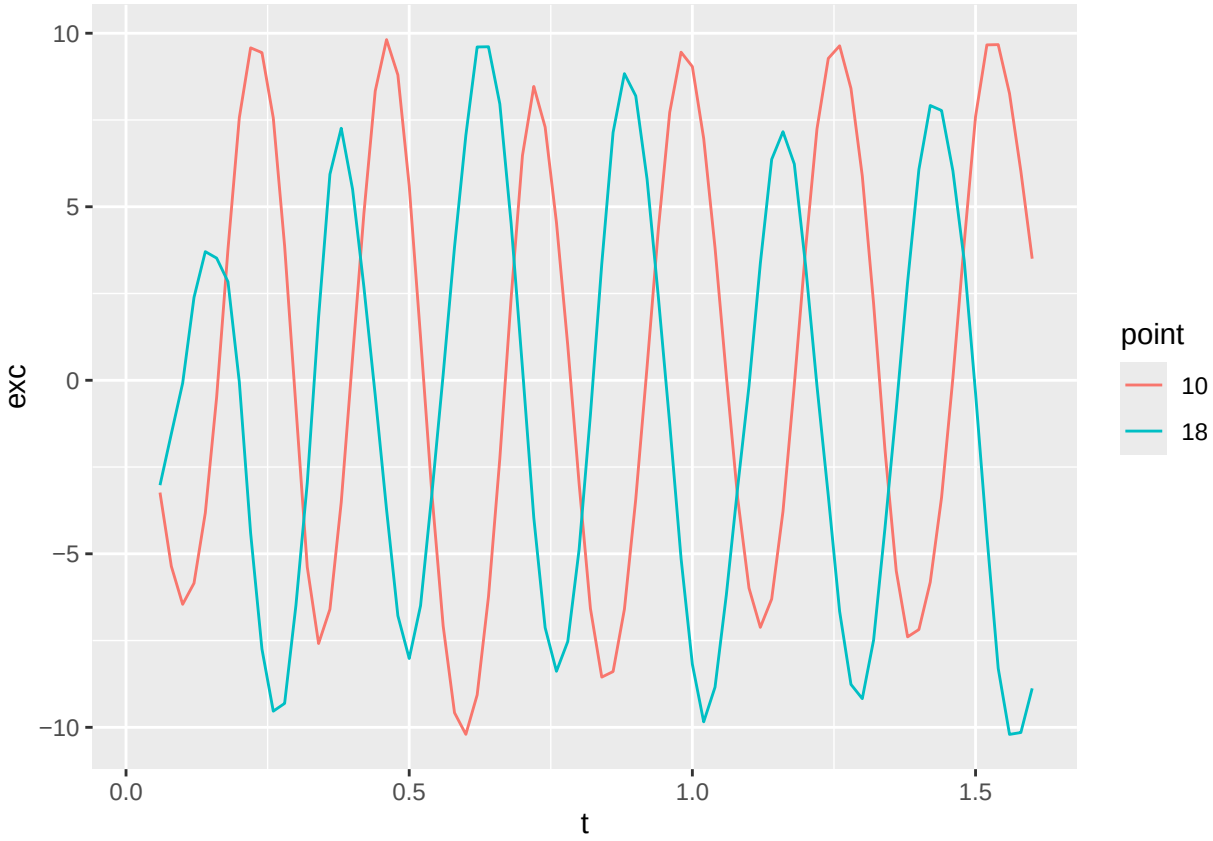
```
lampreydata <- lampreydata |>
  mutate(mxmm_ctr = mxmm_s - xcom,
         mymm_ctr = mymm_s - ycom) |>
  get_primary_swimming_axis_df(t, mxmm_ctr, mymm_ctr, .frame=frame)
#> Warning in check.out(.data, .out, .out_default = c(swimaxis_x = "swimaxis_x", :
#> Columns (swimaxis_x,swimaxis_y,exc_x,exc) will be overwritten
```

```
lampreydata |>
  filter(frame %in% c(10, 20, 30)) |>
  ggplot(aes(x = mxmm, y = mymm, color = frame)) +
  geom_path(aes(group = frame)) +
  geom_segment(data = ~ filter(.x, point == 20),
               aes(x = xcom, y = ycom,
                   xend = xcom - 50*swimaxis_x,
                   yend = ycom - 50*swimaxis_y)) +
  geom_point(data = ~ filter(.x, point == 20),
               aes(x = xcom, y = ycom), color = 'red') +
  facet_grid(frame ~ .) +
  coord_fixed()
```



Similar to curvature, we can compute phases based on the lateral excursions (`b`). As above, they need to be mostly sinusoidal and centered around zero.

```
lampreydata |>
  filter(point %in% c(10, 18)) |>
  mutate(point = factor(point)) |>
  ggplot(aes(x = t, y = exc, color = point)) +
  geom_path()
#> Warning: Removed 4 rows containing missing values or values outside the scale range
#> (`geom_path()`).
```



Phase

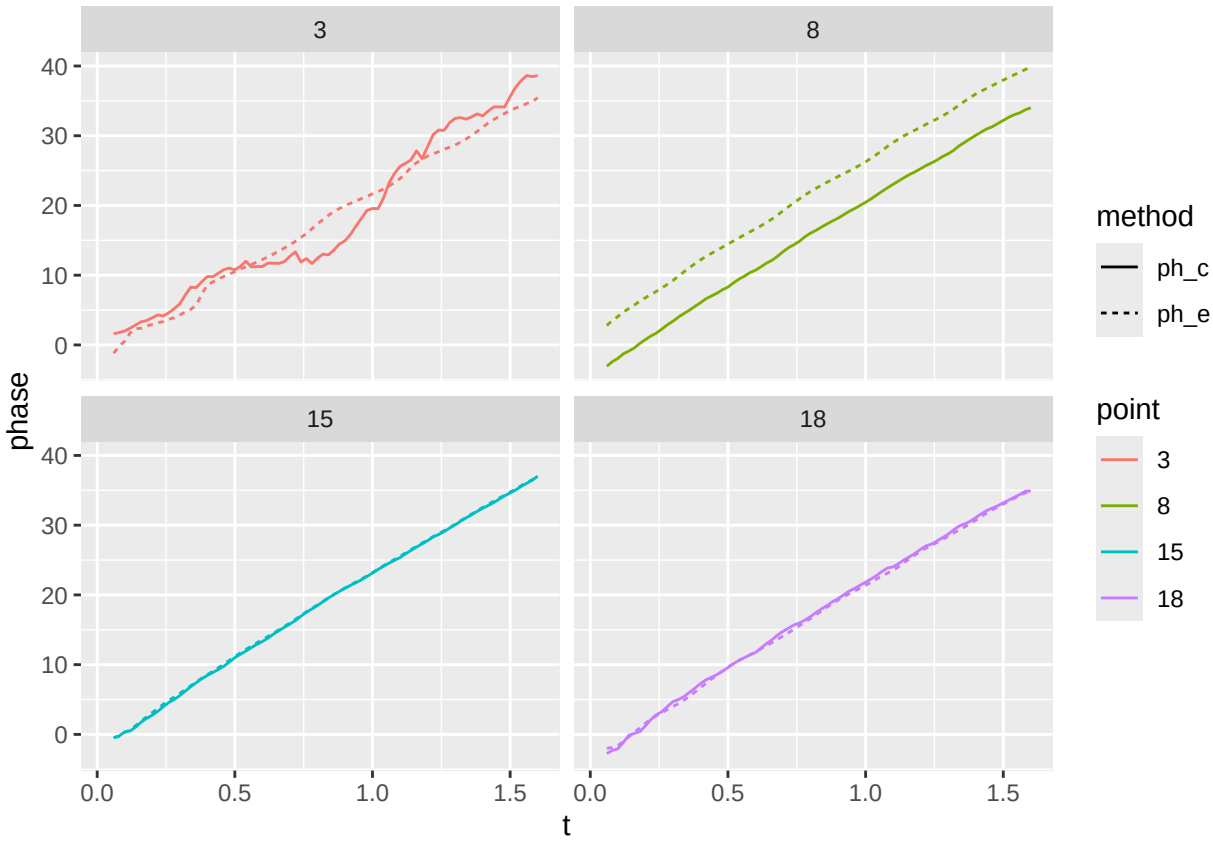
Now we compute the phase of each body point as it oscillates over time, looking at either the curvature `curve_ang` or the excursion `b`.

```
lampreydata <-
  lampreydata |>
  group_by(point) |>
  mutate(ph_c = hilbert_phase(curve_ang),
         ph_e = hilbert_phase(exc))
#> Warning: There were 43 warnings in `mutate()`.
#> The first warning was:
#> i In argument: `ph_c = hilbert_phase(curve_ang)`.
#> i In group 2: `point = 2`.
#> Caused by warning in `hilbert_phase()`:
#> ! Phase seems to go backwards a lot, which may indicate an overly noisy signal
#> i Run `dplyr::last_dplyr_warnings()` to see the 42 remaining warnings.
```

`hilbert_phase` does a few checks on the results and gives warnings if it finds things that might cause problems. * It checks to make sure that most of the input data is not NA. It does not give a warning if all of the input data is NA, only if at least some of the data is not NA. * It checks that the input signal seems to oscillate around 0. * It looks whether the phase mostly advances over time. If we have cases when the phase seems to run backwards, that's often a good indication that the data isn't smoothed enough. In the case above, the warning is for point 2, which is close to the head, where curvature tends to be quite low, so we can ignore the warning.

Compare the two phase estimates. We expect the slopes to be the same, but there could be an offset of 2π in places. Here solid lines are phase based on curvature and dashed are phase based on excursion.

```
lampreydata |>
  ungroup() |>
  filter(point %in% c(3, 8, 15, 18)) |>
  mutate(point = factor(point)) |>
  pivot_longer(cols = c(ph_c, ph_e), names_to = "method", values_to = "phase") |>
  ggplot(aes(x = t, y = phase, color = point, linetype = method)) +
  geom_path() +
  facet_wrap(~point)
#> Warning: Removed 16 rows containing missing values or values outside the scale range
#> (`geom_path()`).
```



Note that the curvature based phase (solid line) tends to fluctuate a lot for point 3, which is near the head where curvature is small.

Frequency

As described above, the time derivative of the phase is the frequency. Here, we can compute a frequency at each point along the body, although they ought to be the same.

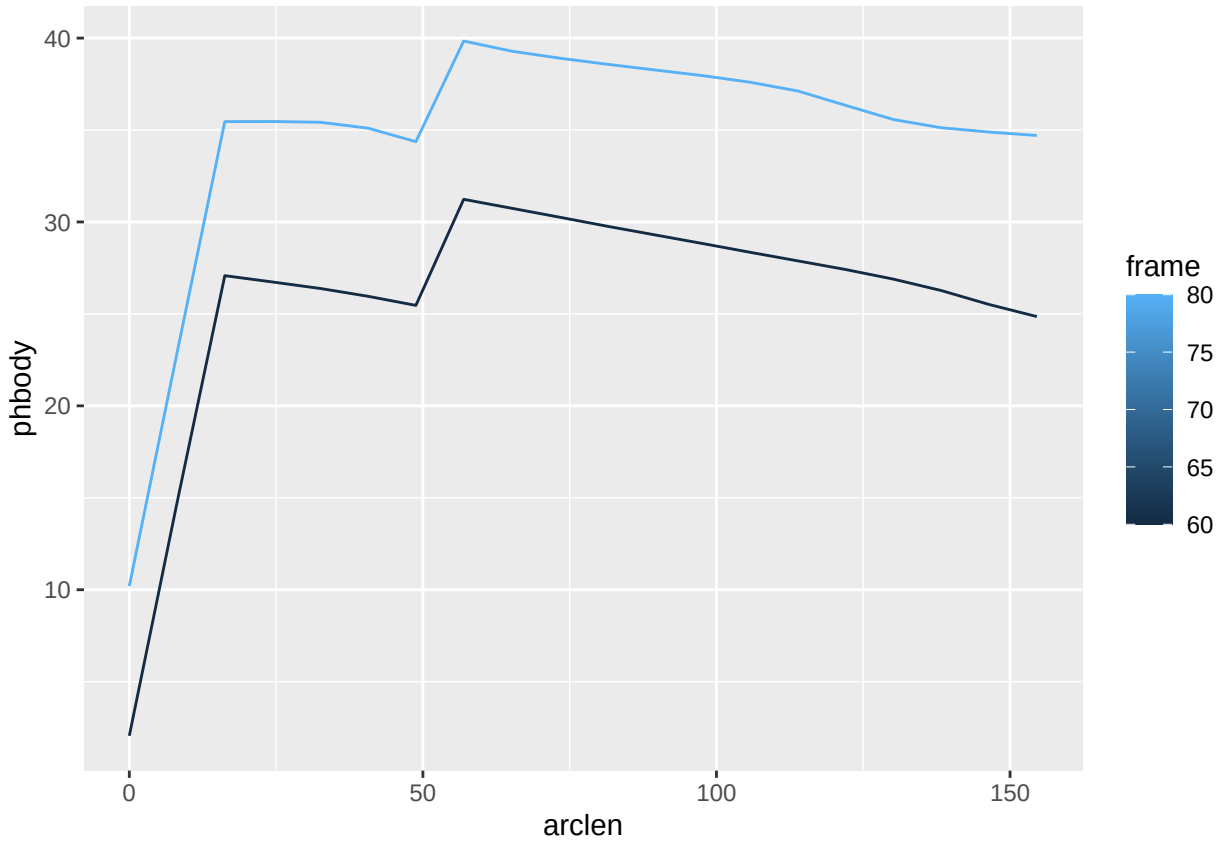
```
lampreydata |>
  group_by(point) |>
  mutate(freq_c = get_frequency(t, ph_c, method='deriv'),
         freq_e = get_frequency(t, ph_e, method='deriv')) |>
  pivot_longer(cols = c(freq_c, freq_e), names_to = "method", values_to = "freq") |>
  filter(point %in% c(3, 15, 18)) |>
  mutate(point = factor(point)) |>
  ggplot(aes(x = t, y = freq, color = point, shape = method)) +
  scale_shape_manual(values = c(1, 17)) +
  geom_point() +
  facet_wrap(~point)
#> Warning: There were 4 warnings in `mutate()`.
#> The first warning was:
#> i In argument: `freq_c = get_frequency(t, ph_c, method = "deriv")`.
#> i In group 2: `point = 2`.
#> Caused by warning in `get_frequency()`:
#> ! Phase seems to go backwards a lot, which may indicate an overly noisy signal
#> i Run `dplyr::last_dplyr_warnings()` to see the 3 remaining warnings.
#> Warning: Removed 18 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```

#

Body wavelength

Body wavelength, as described above, is the derivative of the phase of each point along the body with respect to arc length. Phase typically decreases along the body, which indicates a backward traveling wave.

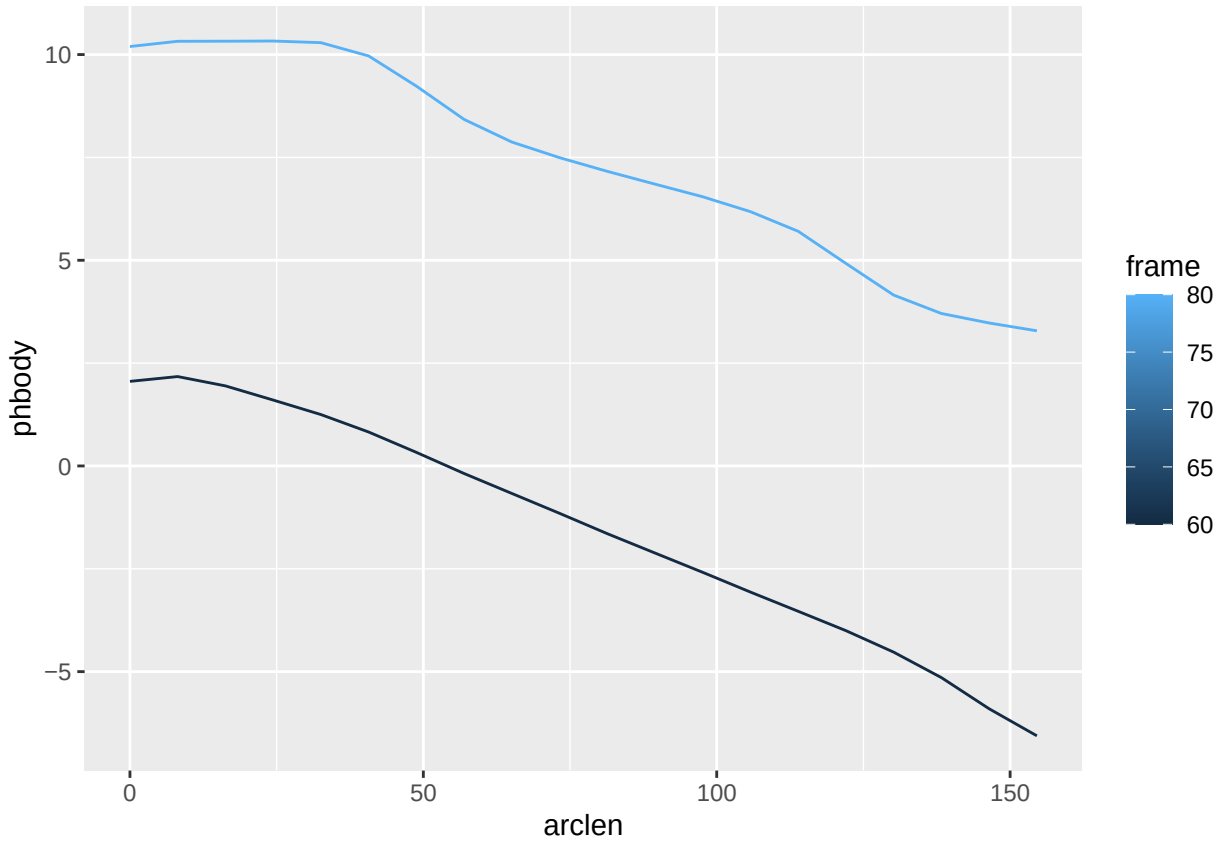
```
lampreydata |>
  filter(frame %in% c(60, 80)) |>
  group_by(frame) |>
  mutate(phbody = ph_e) |>
  ggplot(aes(x = arclen, y = phbody, color = frame)) +
  geom_path(aes(group = frame))
```



The apparent discontinuity at $s = 50$mm is not real. It just reflects the fact that phase goes from 0 to 2π and then wraps around back to 0. We get rid of that by using the `unwrap` function, applied across the spatial dimension (equivalently, using `group_by` with time or frame number).

This is the phase across the body with the discontinuities removed.

```
lampreydata |>
  filter(frame %in% c(60, 80)) |>
  group_by(frame) |>
  mutate(phbody = gsignal::unwrap(ph_e)) |>
  ggplot(aes(x = arclen, y = phbody, color = frame)) +
  geom_path(aes(group = frame))
```



Methods of estimating wavelength

There are several ways to estimate body wavelength based on the body phase, which have different advantages and disadvantages. The function `get_wavelength` can run each of them.

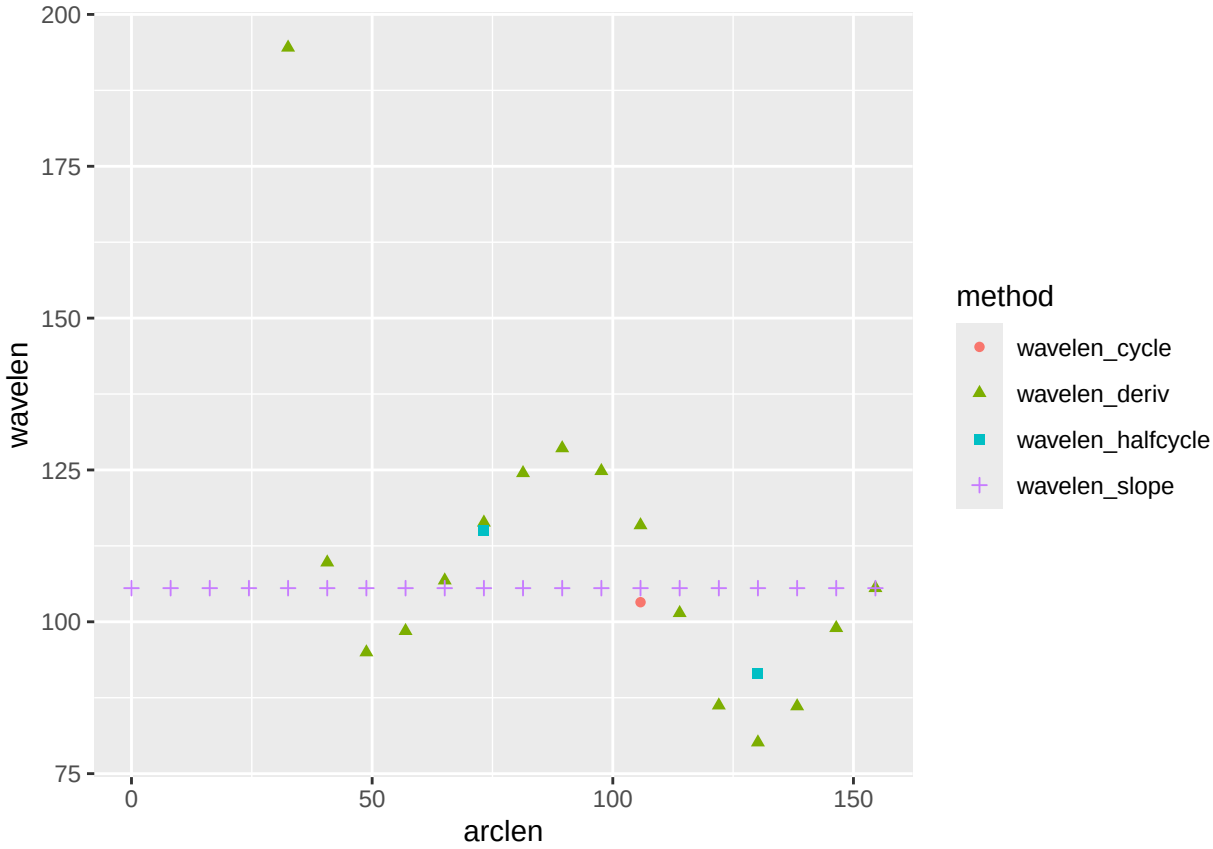
1. At each point along the body, you can estimate the derivative of the phase with respect to arc length. This gives the most comprehensive results, and, in particular, makes it fairly simple to consider whether wavelength varies along the body, but is also very sensitive to noise.
2. You can fit a line to all of the data along the body and take the slope of that line. This method is the least sensitive to noise, but only gives a single value along the entire body.
3. You can look for the distance along the body in which the phase changes by a full or a half cycle. If you look for the phase at the tail and then step backward along the body until you find a point that's either a full or half cycle before the tail, then that distance represents a full or half wave, respectively. `get_wavelength` makes this process more accurate by using a simple linear interpolation to find the distance that represents exactly 1 or 0.5 waves.

The phase tends to be poorly estimated for body points that are more anterior. `get_wavelength` allows you to specify a formula to ignore certain locations for the estimate.

Here is an example for a single frame. We're ignoring points that are less than 30mm from the head.

```
w <-
  lampreydata |>
  filter(frame %in% c(50)) |>
  group_by(frame) |>
  mutate(ph2 = gsignal::unwrap(ph_e),
         wavelen_deriv = get_wavelength(arclen, ph2, method="deriv",
                                        ignore_arclen_vals = \(s) s < 30),
```

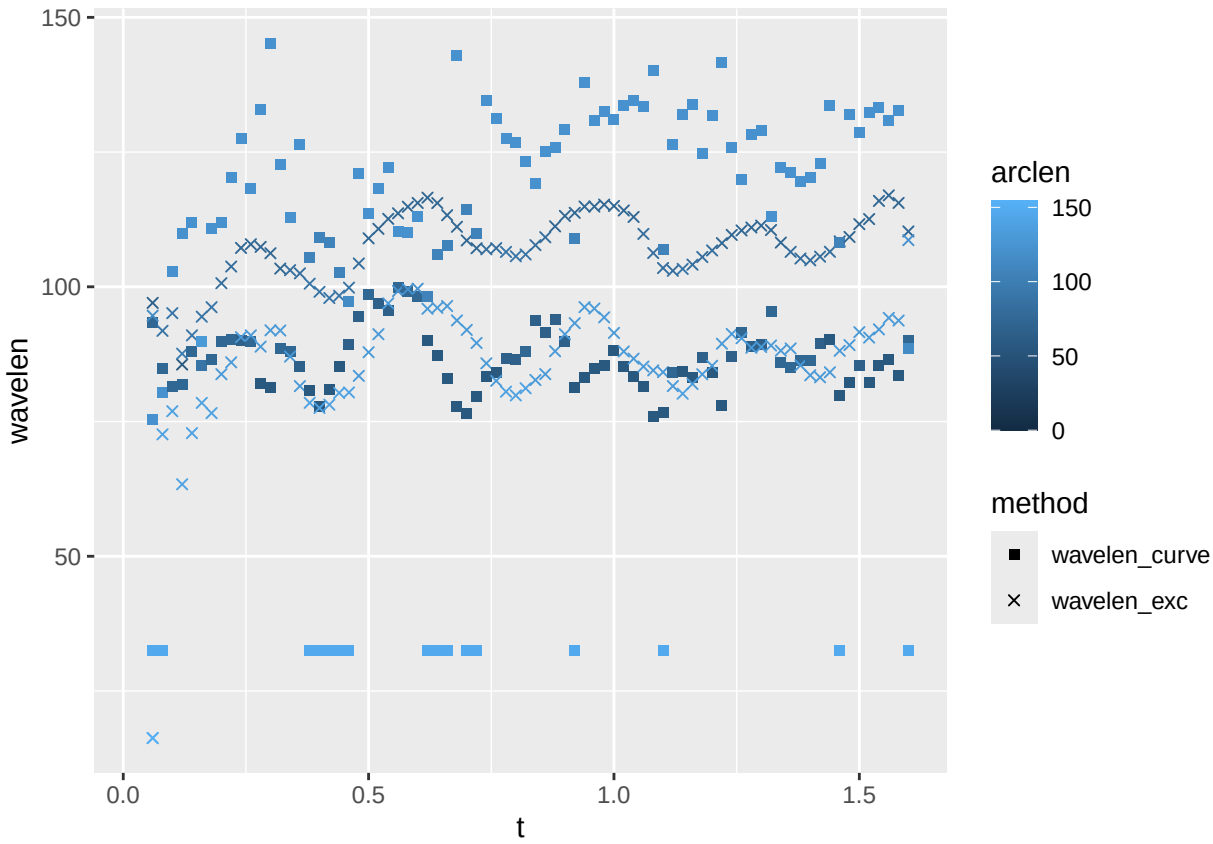
```
        wavelen_slope = get_wavelength(arclen, ph2, method="slope",
                                       ignore_arclen_vals = \(s) s < 30),
        wavelen_cycle = get_wavelength(arclen, ph2, method="cycle",
                                       ignore_arclen_vals = \(s) s < 30),
        wavelen_halfcycle = get_wavelength(arclen, ph2, method="halfcycle",
                                       ignore_arclen_vals = \(s) s < 30))
w |>
  pivot_longer(cols = contains("wavelen"), names_to = "method", values_to = "wavelen") |>
  ggplot(aes(x = arclen, y = wavelen, color = method, shape = method)) +
  geom_point()
#> Warning: Removed 41 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```



Here, you can see that all of the methods give similar results, but there are different numbers of values and different resolution along the body. We suggest the "halfcycle" option as a good, relatively robust compromise.

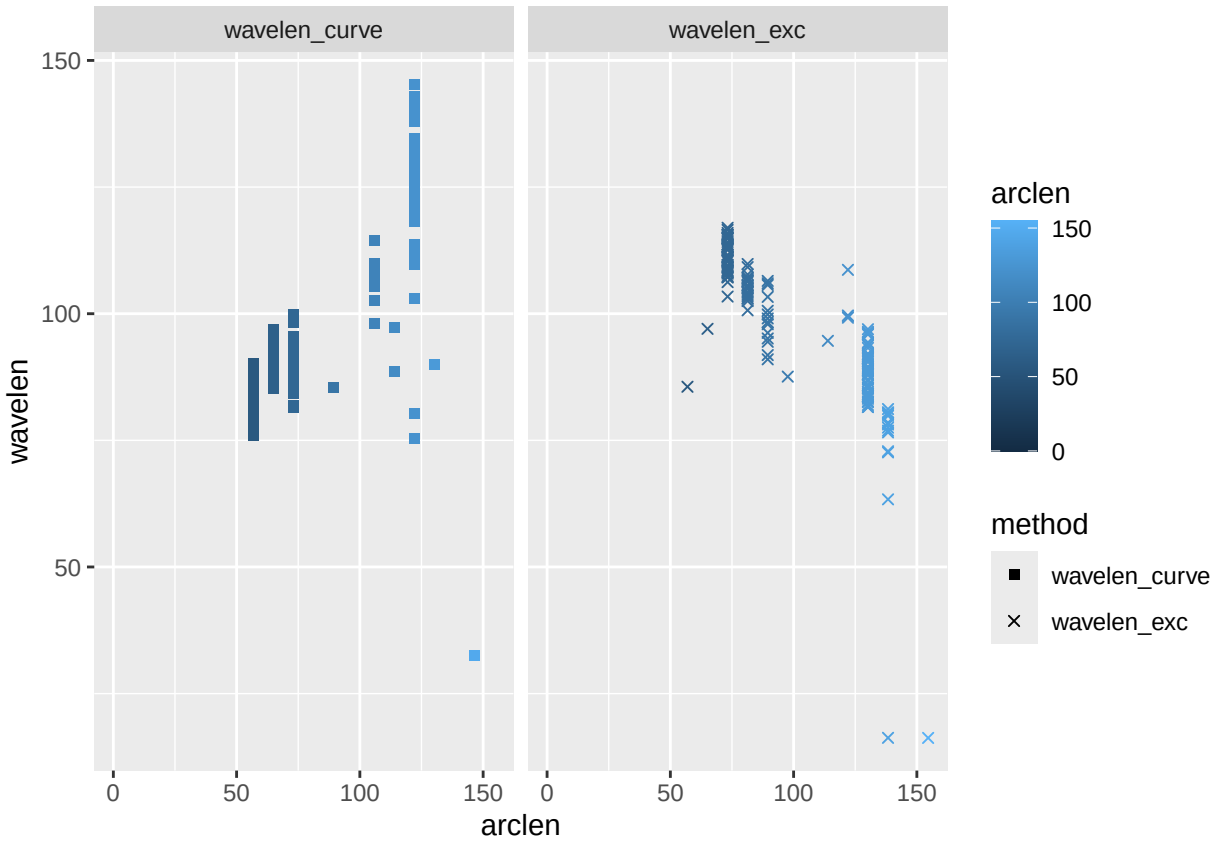
```
lampreydata |>
  group_by(frame) |>
  mutate(phbody_e = gsignal::unwrap(ph_e),
         phbody_c = gsignal::unwrap(ph_c),
         wavelen_exc = get_wavelength(arclen, phbody_e, method="halfcycle",
                                      ignore_arclen_vals = \(s) s < 30),
         wavelen_curve = get_wavelength(arclen, phbody_c, method="halfcycle",
                                      ignore_arclen_vals = \(s) s < 30)) |>
  pivot_longer(cols = contains("wavelen"), names_to = "method", values_to = "wavelen") |>
  ggplot(aes(x = t)) +
  geom_point(aes(y = wavelen, color = arclen, shape=method)) +
```

```
  scale_shape_manual(values = c(15, 4))
#> Warning: Removed 2869 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```



Here we see that wavelength estimated based on curvature is slightly noisier than curvature based on excursion, and that both fluctuate over time.

```
lampreydata |>
  group_by(frame) |>
  mutate(phbody_e = gsignal::unwrap(ph_e),
         phbody_c = gsignal::unwrap(ph_c),
         wavelen_exc = get_wavelength(arclen, phbody_e, method="halfcycle",
                                      ignore_arclen_vals = \(s) s < 30),
         wavelen_curve = get_wavelength(arclen, phbody_c, method="halfcycle",
                                      ignore_arclen_vals = \(s) s < 30)) |>
  pivot_longer(cols = contains("wavelen"), names_to = "method", values_to = "wavelen") |>
  ggplot(aes(x = arclen)) +
  geom_point(aes(y = wavelen, color = arclen, shape=method)) +
  scale_shape_manual(values = c(15, 4)) +
  facet_wrap(~method)
#> Warning: Removed 2869 rows containing missing values or values outside the scale range
#> (`geom_point()`).
```



This plot shows wavelength as a function of body position and estimation method. The curvature-based wavelengths seem to increase along the body (aside from some erroneous very low values very close to the tail), while the excursion-based wavelengths tend to decrease moving toward the tail.

Amplitude

To estimate amplitude, it's helpful to set up an overall cycle number for the whole body. We define this based on the excursion of the tail point. `get_body_cycle_numbers` also excludes partial cycles.

```
lampreydata <-
  lampreydata |>
  get_body_cycle_numbers_df(ph_e, 20) |>
  arrange(t, point)
#> Warning in check.out(.data, .out, .out_default = c(cycle = "cycle"), overwrite =
#> overwrite): Columns (cycle) will be overwritten
```

```
lampreydata
#> # A tibble: 1,600 x 23
#> # Groups:   point [20]
#>       t frame point  mxmm  mymm arclen0 arclen mxmm_s mymm_s width  xcom  ycom
#>   <dbl> <int> <int> <dbl> <dbl>   <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>
#>  1  0.02     1     1    NA    NA      NA   0        NA     NA  2.48    NA    NA
#>  2  0.02     1     2    NA    NA      NA   8.13     NA     NA  2.61    NA    NA
#>  3  0.02     1     3    NA    NA      NA  16.3      NA     NA  2.75    NA    NA
#>  4  0.02     1     4    NA    NA      NA  24.4      NA     NA  2.81    NA    NA
#>  5  0.02     1     5    NA    NA      NA  32.5      NA     NA  2.83    NA    NA
#>  6  0.02     1     6    NA    NA      NA  40.7      NA     NA  2.85    NA    NA
```

```
#>  7  0.02     1     7    NA    NA      NA  48.8      NA      NA  2.88    NA    NA
#>  8  0.02     1     8    NA    NA      NA  56.9      NA      NA  2.80    NA    NA
#>  9  0.02     1     9    NA    NA      NA  65.1      NA      NA  2.70    NA    NA
#> 10  0.02     1    10    NA    NA      NA  73.2      NA      NA  2.61    NA    NA
#> # i 1,590 more rows
#> # i 11 more variables: curve_ang <dbl>, curve_xy <dbl>, mxmm_ctr <dbl>,
#> #   mymm_ctr <dbl>, ph_c <dbl>, ph_e <dbl>, swimaxis_x <dbl>, swimaxis_y <dbl>,
#> #   exc_x <dbl>, exc <dbl>, cycle <dbl>
```

Now, we can use `group_by` on cycles to find cycle-by-cycle values. Here, we look for the range of the excursion variable `b`. Half of that range is the amplitude, which we can define at each point on the body, and over the four cycles that are present in this trial.

```
lampreydata |>
  group_by(point, cycle) |>
  summarize(amp = (max(exc) - min(exc)) / 2,
            arclen = mean(arclen)) |>
  ggplot(aes(x = arclen, y = amp, color = cycle)) +
  geom_path(aes(group = cycle))
#> `summarise()` has grouped output by 'point'. You can override using the `.groups`
#> argument.
#> Warning: Removed 20 rows containing missing values or values outside the scale range
#> (`geom_path()`).
```