Periodic Modeling

Developed by: Martin Modrak

https://discourse.mc-stan.org/t/mathy-folks-thoughts-on-the-asymptotic-representation-of-this-k-hmm-model-for-periodic-signals/22038/24

25 April 2021

Clear Out Console Script

```
cat("\014")
```

So we have a bounded periodic function f and I'll assume the period is 2π , i.e. that $f(x) = f(x + 2k\pi), -1 \leq f(x) \leq 1$ for all $k \in \mathrm{Z}, x \in \mathrm{R}$. We observe a scaled and shifted realization: $y \sim Normal(f(\nu x + p), \sigma)$ and we want to determine ν, p .

We'll pick two arbitrary numbers x_1, x_2 as data. The only requirement is that those points are quite far from each other, but are sorrounded by actually observed values. We'll parametrize our model in terms of the value of the function at those points:

$$
\begin{aligned}
-1 < z_i < 1 \\
z_1 = f(\nu x_1 + p) \\
z_2 = f(\nu x_2 + p)
\end{aligned}
$$

The main advantage is that $z_1, z_2 \in (-1, 1)$ will almost always be well constrained by data, so no multimodality problems here. But how do we get from z to ν, p ?

We'll start by looking at preimages of z within a single period. We'll assume that each point has exactly s distinct preimages in each period (some care would be needed to handle cases where some values have more preimages than others, but it is IMHO possible). For the $\sin$ function $s = 2$ (we'll never have exactly $z_i = \pm 1$).

$$
\begin{aligned}
0 \leq w_{i,j} < 2\pi \\
\{w_{1,1}, \ldots, w_{1,s}\} = f^{-1}(z_1) \\
\{w_{2,1}, \ldots, w_{2,s}\} = f^{-1}(z_2)
\end{aligned}
$$

We then introduce additional parameter k that says how many full periods of the function are between x_1 and x_2 . So we have a set of pairs of preimages and we can use those to find all solutions for ν, p given z .

**EDIT: ** There was previously a sign error in this equation.

$$
\begin{aligned}
\nu_{i,j,k} = \frac{w_{2,i} + 2k\pi - w_{1,j}}{x_2 - x_1} \\
p_{i,j,k} = w_{1,j} - \nu_{i,j,k} x_1
\end{aligned}
$$

So beyond z_1, z_2 we need 3 discrete parameters: i, j, k . If we can put an upper bound on k (which we IMHO always can as e.g. period shorter then distance between two consecutive points can be ruled out), we can marginalize all the discrete parameters out! This way, we sum out all the modes. The necessary background is at 7.2 Change point models | Stan User's Guide

Figure 1: Description.

Setting Work Directory

```
setwd("~/Dropbox/My Mac (Jonathan's MacBook Pro)/Desktop")
getwd()
```

```
## [1] "/Users/jonathan.h.morgan/Dropbox/My Mac (Jonathan's MacBook Pro)/Desktop"
```

Options

```
options(stringsAsFactors = FALSE)
options(mc.cores = parallel::detectCores())
options(mc.cores = 4)
```

PACKAGES

```
library(cmdstanr)     #Used to run cmdstan from R
```

```
## This is cmdstanr version 0.3.0
## - Online documentation and vignettes at mc-stan.org/cmdstanr
## - CmdStan path set to: /Users/jonathan.h.morgan/.cmdstanr/cmdstan-2.26.1
## - Use set_cmdstan_path() to change the path
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.1     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Checking that I am pointing to cmdstan

```
set_cmdstan_path("/Users/jonathan.h.morgan/cmdstan")
```

```
## CmdStan path set to: /Users/jonathan.h.morgan/cmdstan
```

```
cmdstan_path()
```

```
## [1] "/Users/jonathan.h.morgan/cmdstan"
```

```
cmdstan_version()
```

```
## [1] "2.26.1"
```

CREATING SYNTHETIC DATA

Parameters

```
n <- 30
frequency <- 13
phase <- 0.3
noise <- 0.5
amplitude <- 1.4
```

Constructing Data

```
x <- runif(n, max = 2 * pi)
f <- amplitude * sin(frequency * x + phase)
y <- rnorm(n, mean = f, sd = noise)
x_fixed = c(pi / 4, 7/4 * pi)
```

Creating Input List

```
data <- list(
  n = n,
  y = y,
  x = x,
  x_fixed = x_fixed,
  max_k = 10
)
```

ESTIMATING STAN MODEL & SAVING DATA OBJECTS

Setting Seed

```
set.seed(10271998)
```

Setting Temporary Directory

```
temp <- tempdir()
```

Compiling Model

```
mod <- cmdstan_model("~/Dropbox/My Mac (Jonathan's MacBook Pro)/Desktop/periodic.stan")
```

```
## Model executable is up to date!
```

Checking Model

```
mod$print()
```

```
## //  Example Periodic Function
## //  Martin Modrak: https://discourse.mc-stan.org/t/mathy-folks-thoughts-on-the-asymptotic-representa
## //  25 April 2021
##
## data{
##    int n ;
##    vector[n] y ;
##    vector[n] x ;
##    int<lower=0> max_k;
##    real x_fixed[2];
## }
##
## transformed data {
##   real log_prior[max_k + 1] = rep_array(-log(max_k + 1.0), max_k + 1); //prior prob 1/(max_k + 1)
## }
##
## parameters{
##    real<lower=0> noise ;
##    real<lower=0> amplitude;
##    real<lower=-1, upper=1> z[2];
## }
##
## transformed parameters{
##   real frequency[2, 2, max_k + 1];
##   real phase[2, 2, max_k + 1];
##   real log_lik[2, 2, max_k + 1];
##   {
##     real asinz[2] = asin(z);
##     real w[2,2];
##     w[1,]= { asinz[1], pi() - asinz[1]};
##     w[2,]= { asinz[2], pi() - asinz[2]};
##     for(i in 1:2) {
##       for(j in 1:2) {
##         for(k in 1:(max_k + 1)) {
##           frequency[i, j, k] = (w[2, i] + 2 * (k-1) * pi() - w[1, j]) / (x_fixed[2] - x_fixed[1]);
##           phase[i, j , k] = w[1,i] - frequency[i, j, k] * x_fixed[1];
##           {
##             vector[n] f = amplitude * sin(frequency[i, j, k] * x + phase[i, j, k]);
##             log_lik[i, j, k] = normal_lpdf(y | f, noise)
##               + log_prior[k]
##               -log(2.0) //Uniform prior over the 2 options
##               ;
##           }
##         }
##       }
##     }
##   }
## }
##
## model{
```

```
##    noise ~ weibull(2,1) ; //peaked at ~.8, zero-at-zero, ~2% mass >2
##    amplitude ~ weibull(2,1) ; //ditto
##    target += log_sum_exp(to_array_1d(log_lik));
## }
##
## generated quantities {
##   // Discrete sampling is inefficient, directly computing weights
##   // And taking weighted expectation values would work better
##   int index = categorical_logit_rng(to_vector(to_array_1d(log_lik)));
##   real freq_chosen = to_array_1d(frequency)[index];
##   real phase_chosen = to_array_1d(phase)[index];
##   //Force phase to lie between -pi + pi the stupid way
##   while(phase_chosen < -pi()) {
##     phase_chosen += 2 * pi();
##   }
##   while(phase_chosen > pi()) {
##     phase_chosen -= 2 * pi();
##   }
## }
```

Sampling

```
fit <- mod$sample(
  data = data,
  output_dir = temp,
  seed = 123,
  chains = 4,
  thin = 1,
  parallel_chains = 4,
  iter_warmup = 2000,
  iter_sampling = 2000,
  adapt_delta = 0.99,
  max_treedepth = 15,
  refresh = 500
)
```

```
## Running MCMC with 4 parallel chains...
##
## Chain 1 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 2 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 2 Exception: weibull_lpdf: Random variable is inf, but must be finite! (in '/var/folders/dl/_9
## Chain 2 If this warning occurs sporadically, such as for highly constrained variable types like covar
## Chain 2 but if this warning occurs often then your model may be either severely ill-conditioned or mi
## Chain 2
## Chain 3 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3 Informational Message: The current Metropolis proposal is about to be rejected because of the
## Chain 3 Exception: normal_lpdf: Location parameter[1] is -inf, but must be finite! (in '/var/folders/
## Chain 3 If this warning occurs sporadically, such as for highly constrained variable types like covar
```

```
## Chain 3 but if this warning occurs often then your model may be either severely ill-conditioned or mi
## Chain 3
## Chain 4 Iteration:    1 / 4000 [  0%]  (Warmup)
## Chain 3 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 2 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 4 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 1 Iteration:  500 / 4000 [ 12%]  (Warmup)
## Chain 3 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 4 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 1 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 2 Iteration: 1000 / 4000 [ 25%]  (Warmup)
## Chain 3 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 1 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 4 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 2 Iteration: 1500 / 4000 [ 37%]  (Warmup)
## Chain 3 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 3 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 1 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 1 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 4 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 4 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 2 Iteration: 2000 / 4000 [ 50%]  (Warmup)
## Chain 2 Iteration: 2001 / 4000 [ 50%]  (Sampling)
## Chain 3 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 1 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 4 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 2 Iteration: 2500 / 4000 [ 62%]  (Sampling)
## Chain 3 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 1 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 4 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 2 Iteration: 3000 / 4000 [ 75%]  (Sampling)
## Chain 3 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 1 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 4 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 2 Iteration: 3500 / 4000 [ 87%]  (Sampling)
## Chain 3 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 3 finished in 11.2 seconds.
## Chain 1 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 1 finished in 11.4 seconds.
## Chain 4 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 4 finished in 11.5 seconds.
## Chain 2 Iteration: 4000 / 4000 [100%]  (Sampling)
## Chain 2 finished in 12.9 seconds.
##
## All 4 chains finished successfully.
## Mean chain execution time: 11.8 seconds.
## Total execution time: 13.1 seconds.
```

Checking that the Model Passes: E-BFMI, Treedepth, Sample Size, & R-Hat Cursory Checks

```
fit$cmdstan_diagnose()
```

```
## Processing csv files: /var/folders/dl/_9shwqkn19s9d3lvpyfcmkf00000gn/T/RtmpsID7jm/periodic-202104261
##
## Checking sampler transitions treedepth.
## Treedepth satisfactory for all transitions.
##
## Checking sampler transitions for divergences.
## No divergent transitions found.
##
## Checking E-BFMI - sampler transitions HMC potential energy.
## E-BFMI satisfactory for all transitions.
##
## Effective sample size satisfactory.
##
## Split R-hat values satisfactory all parameters.
##
## Processing complete, no problems detected.
```

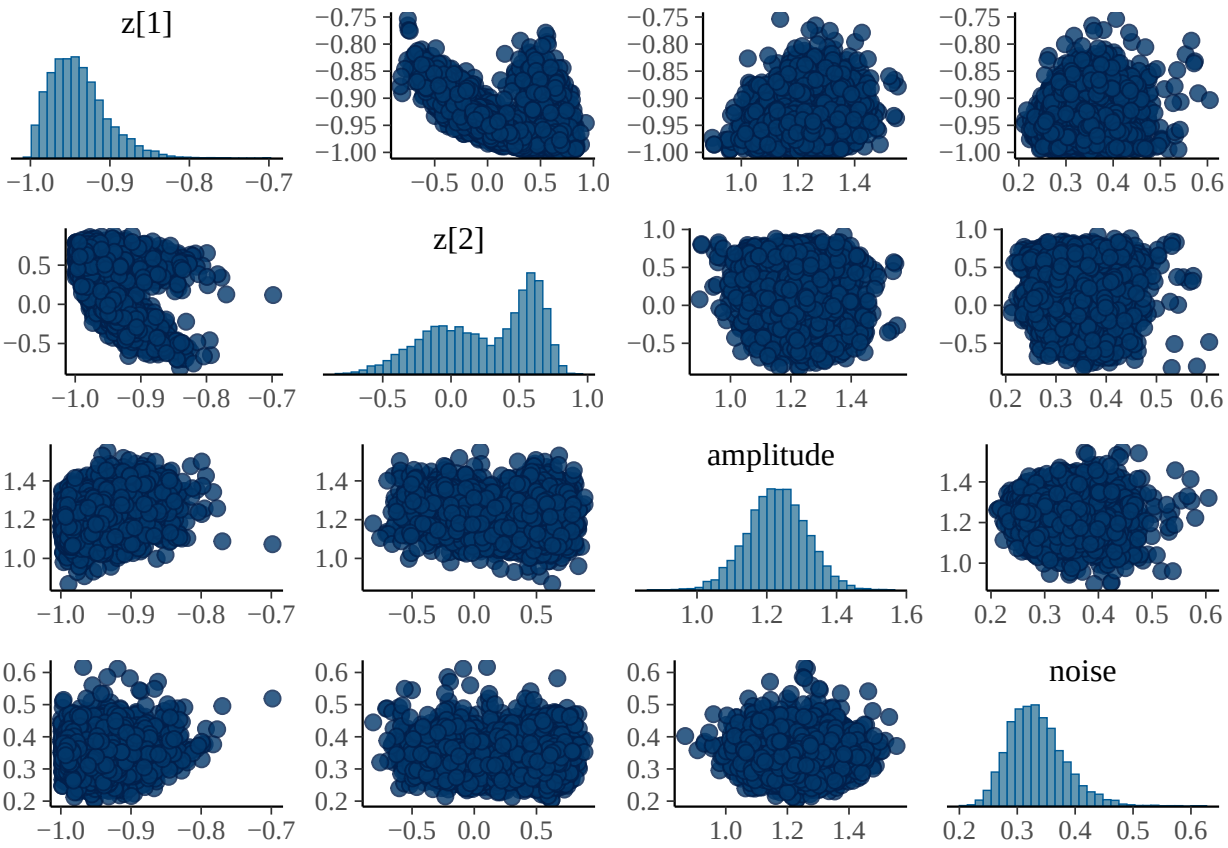
PLOTTING

Creating Data Objects

```
fit$summary(c("noise", "z", "freq_chosen", "phase_chosen", "amplitude", "index"))
```

```
## # A tibble: 7 x 10
##   variable       mean median     sd    mad     q5    q95  rhat ess_bulk ess_tail
##   <chr>         <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl>    <dbl>    <dbl>
## 1 noise         0.334  0.329 0.0491 0.0461  0.264  0.422  1.00    4003.    4183.
## 2 z[1]         -0.939 -0.944 0.0355 0.0335 -0.987 -0.873  1.00    3190.    2094.
## 3 z[2]          0.249  0.327 0.370  0.432  -0.396  0.716  1.00    1062.    3956.
## 4 freq_chosen  12.9   12.9   0.0428 0.0418 12.9   13.0    1.00    4115.    4216.
## 5 phase_chosen  0.502  0.500 0.133  0.132   0.280  0.720  1.00    3193.    2295.
## 6 amplitude     1.23   1.23  0.0856 0.0826  1.09   1.37   1.00    4107.    4016.
## 7 index        37.4   32     5.97   0      32     44      1.00     820.      NA
```

```
bayesplot::mcmc_pairs(fit$draws(), pars = c("z[1]","z[2]", "amplitude", "noise"))
```

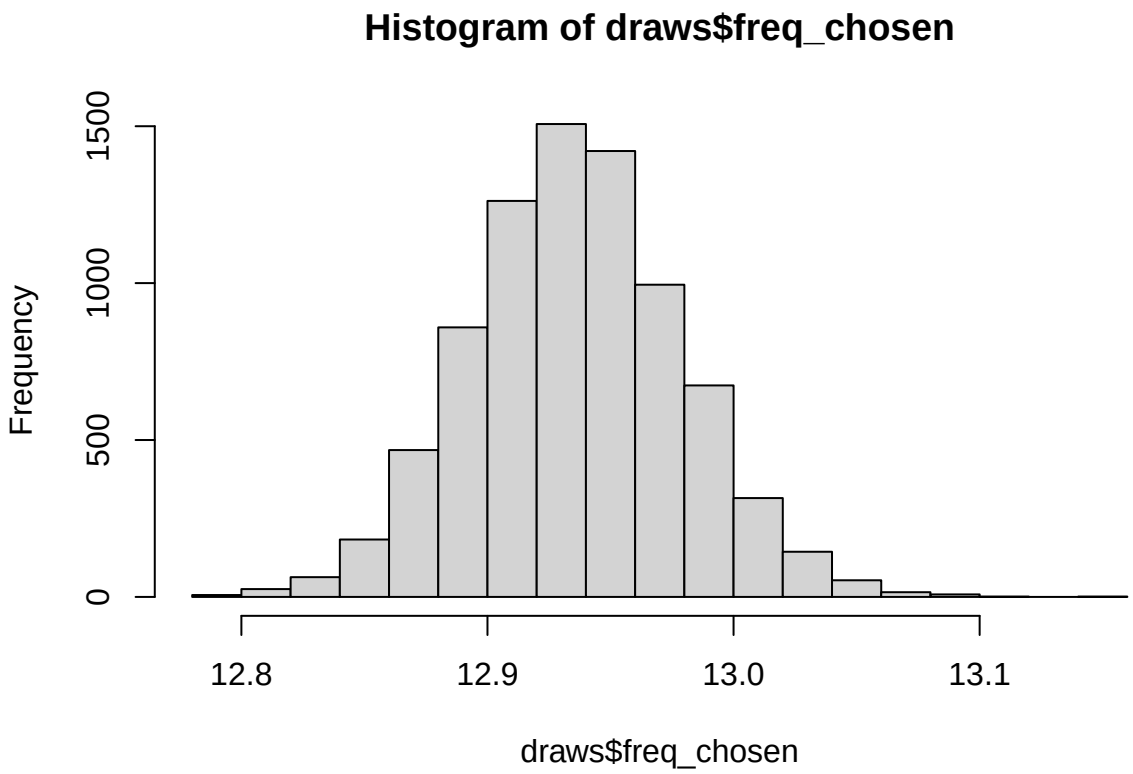


Pulling Out Draws for Frequency, Phase, and Amplitude

```
draws <- as.data.frame(posterior::as_draws_matrix(fit$draws(c("freq_chosen", "phase_chosen", "amplitude
```

Checking the frequency

```
hist(draws$freq_chosen)
```



Constructing True Model

```
x_adj <- seq(0, 2 * pi, length.out = 101)
y_adj <- amplitude * sin(frequency * x_adj + phase)
true <- as.data.frame(cbind(x_adj,y_adj))
colnames(true) <- c('x', 'y')
rm(x_adj,y_adj)
```

Observed Data

```
observed <- tibble(x = x, y = y)
```

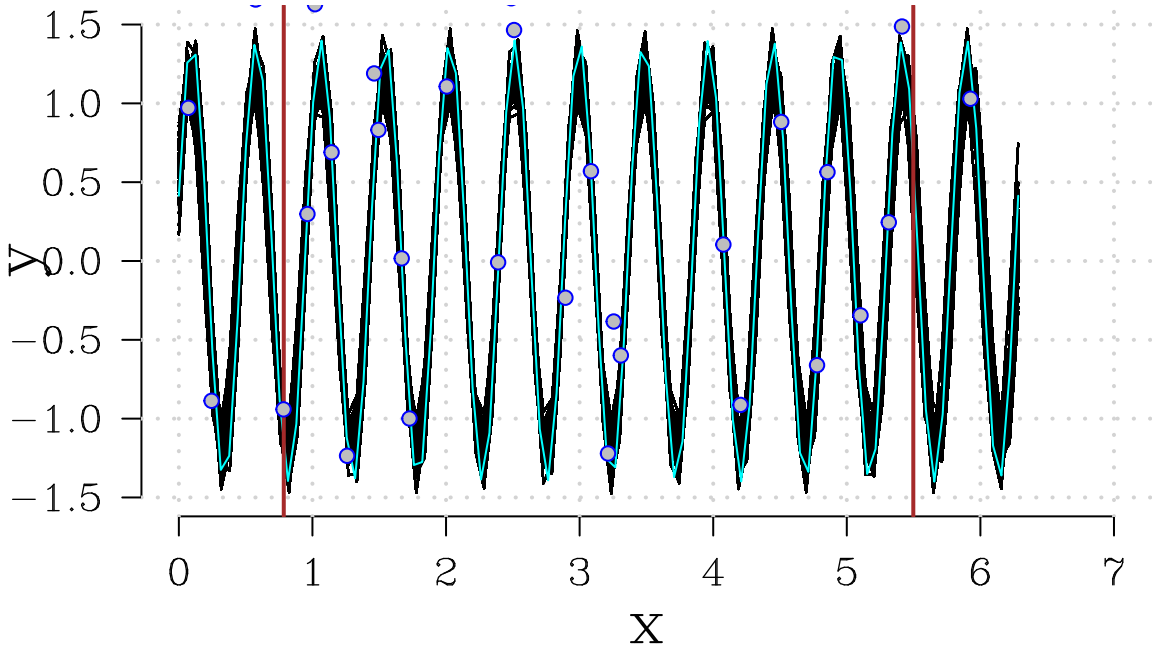
Constructing Draws

```
z <-  tibble(x = seq(0, 2 * pi, length.out = 100)) %>% crossing(as_tibble(draws) %>% mutate(id = 1:dim(
  mutate(y = amplitude * sin(x * freq_chosen + phase_chosen)) %>%
  filter(id %in% sample(unique(id), 100))

rm(amplitude,f,frequency,n,noise, phase, temp,x,x_fixed,y)
```

Plotting

```
x11(width=10.6806, height=7.30556)
periodic_visual <- function() {
  x_values <- pretty(z$x)
  y_values <- pretty(z$y)
```

```
  x_fixed = c(pi / 4, 7/4 * pi)

  plot(0, type='n', xlab=' ', ylab=' ', xlim=c(min(x_values), max(x_values)), ylim=c(min(y_values), max
  grid(lwd = 2)

  ids <- sort(z$id)
  for(i in seq_along(z$id)){
    iteration <- z[(z$id == ids[[i]]), ]
    lines(iteration$x, iteration$y, col="black")
  }

  lines(true, col="cyan")

  abline(v=x_fixed[[1]],col='brown',lwd=2, lty=1)
  abline(v=x_fixed[[2]],col='brown',lwd=2, lty=1)

  points(observed, pch=21, bg="grey", col="blue")

  mtext(side = 1, text = 'x', col = "black", line = 2.5, cex = 2, family='HersheySerif')
  mtext(side = 2, text = 'y', col = "black", line = 2.5, cex = 2, family='HersheySerif')
}

g <- cowplot::as_grob(periodic_visual)
p_1 <- cowplot::ggdraw(g)

p_1
```



NOTES

Parameters where is not fully contrained

```
set.seed(24855)
n <- 8
frequency <- 4
phase <- -0.5
noise <- 0.5
amplitude <- 1.4
```

Constructing Data

```
x <- runif(n, max = 2 * pi)
f <- amplitude * sin(frequency * x + phase)
y <- rnorm(n, mean = f, sd = noise)
x_fixed = c(pi / 4, 7/4 * pi)
```

Creating Input List

```
data <- list(
  n = n,
  y = y,
  x = x,
  x_fixed = x_fixed,
  max_k = 10
)
```