Collapsed Gibbs Sampler with Slice Sampling

RJMCMC-burnin Gibbs Sampler (fits exactly) >> Gibbs Sampler

Jan Meißner[1]

1 Data

```
library(spectralmc)

true_kp <- 9
fit_kp <- 9
noise_sigma <- 0.001
signal_model <- voigt.model

data <- data.synthetic.stormyclouds(seed = 53252, kp = true_kp, noise = 0.001)
x <- data$x
y <- data$y
```

2 Gibbs Sampler

Uses a collapsed (marginalizes over `a`, `b`, `amp` numerically) gibbs sampler which approximates the conditionals with slice sampling. Also re-parameterizes density with `(sqrt(gwidth^2+lwidth^2), gwidth-lwidth)` instead of `(gwidth, lwitdh)` if `min(gwidth/lwidth, lwidth/gwidth) > 0.05` (otherwise transformation becomes unstable). Uses a Likelihood-ratio test to stop slice sampling early. Slice sampling rerun a few times (3-5) to generate draws from which one is sampled according to their posterior density (prevents slice sampling failure).

- Takes 1-4s per peak and iteration
  - **30-40h runtime** for this notebook
  - Thus only usable with a remote cluster (I wont use anymore of my google.colab quota for this)
- short burn-in
- start_point is essentially irrelevant (modulo the label switching problem, gelman-rubin will be terrible due to that)
- small auto-correlation (looks like it in traces at least)
- fits ok
- proposal free
- non-informative priors
  - prior_a = Unif(1e-2, 1e2)
  - prior_b = Unif(-Inf, Inf)
  - prior_amp = Unif(0, Inf)
  - prior_pos = Unif(100, 900)
  - prior_lwidth = Unif(0, Inf)
  - prior_gwidth = Unif(0, Inf)
- pos, lwidth, gwidth prior can be chosen arbitrarily; not done here due to time constraints

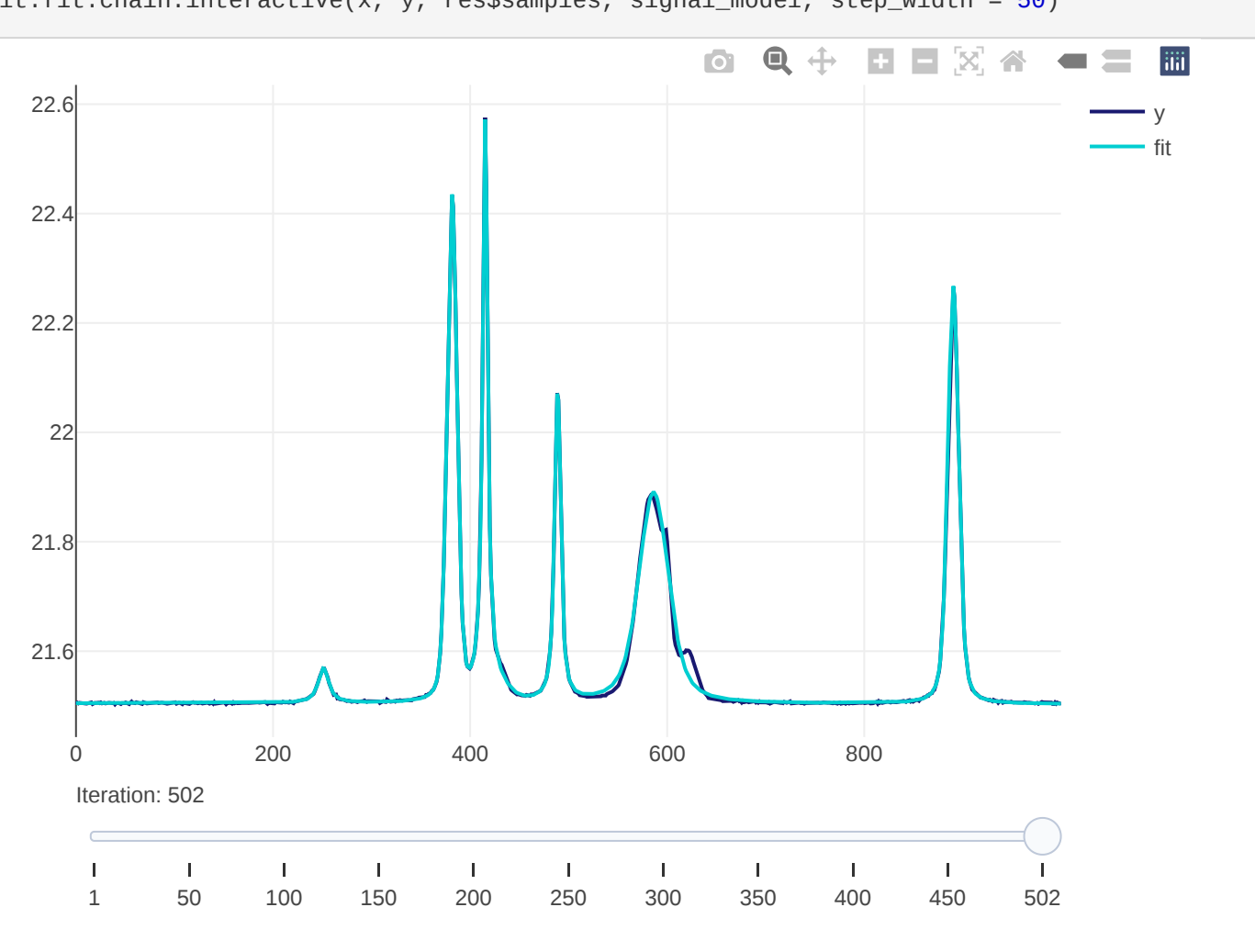
```
max_a <- 1e-2
lower <- c(-Inf, -max_a, 0) #lower_b, lower_a, lower_amp; encodes lower bounds of prior.Un
if
upper <- c(Inf, max_a, Inf) #upper_b, upper_a, upper_amp; encodes lower bounds of prior.Un
if

start_params <- prop.voigt.rnd_start(seed = 33, kp = fit_kp)

res <- chains.collapsed_slicer_gibbs(
        x, y, signal_model,
        start_params, lower, upper, noise_sigma,
        iter = 1000, print_bar = TRUE, half_steps = TRUE, decorr_trafo = TRUE
)
```

2.1 Results

2.1.1 Fit

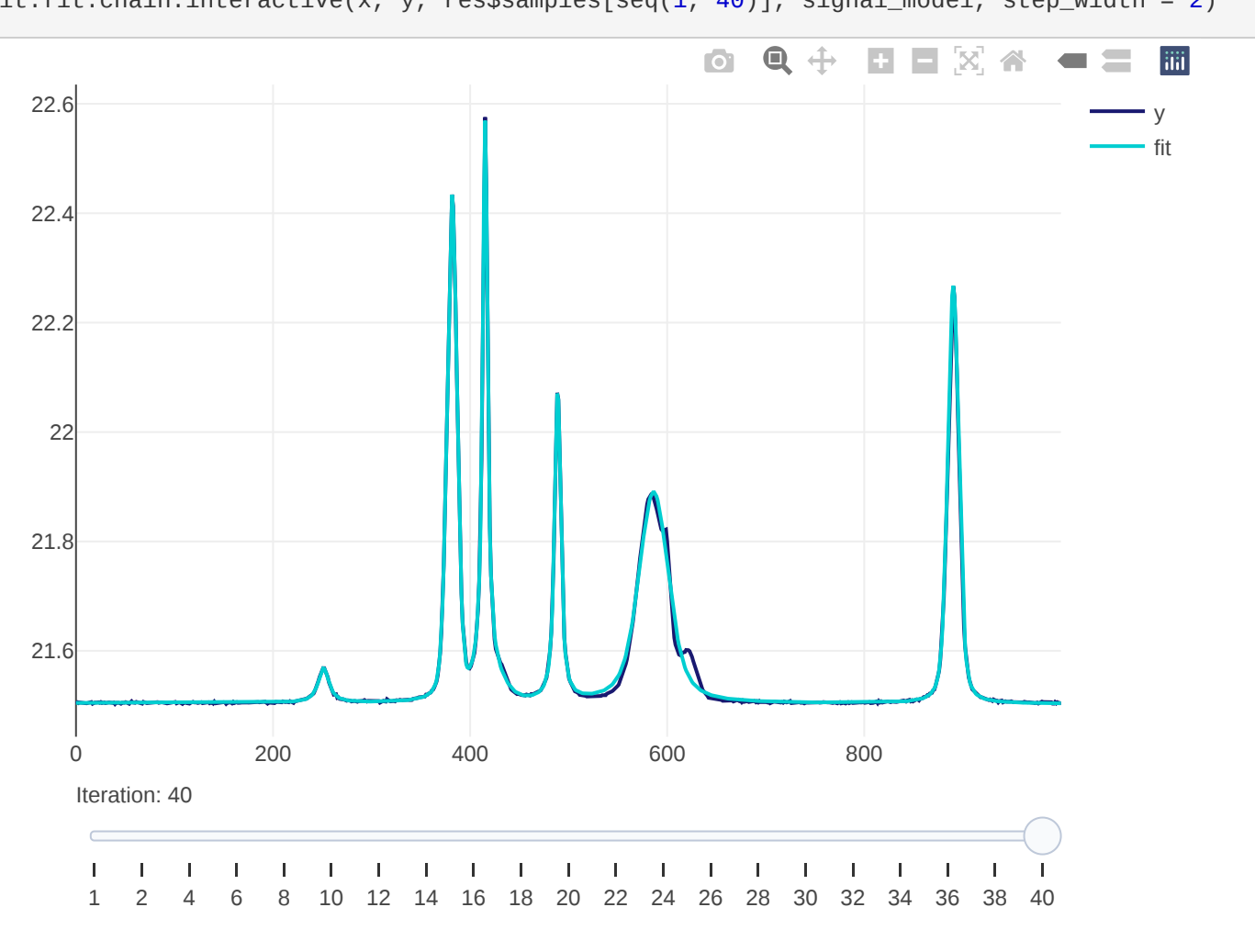
2.1.1.1 All

```
plt.fit.chain.interactive(x, y, res$samples, signal_model, step_width = 50)
```



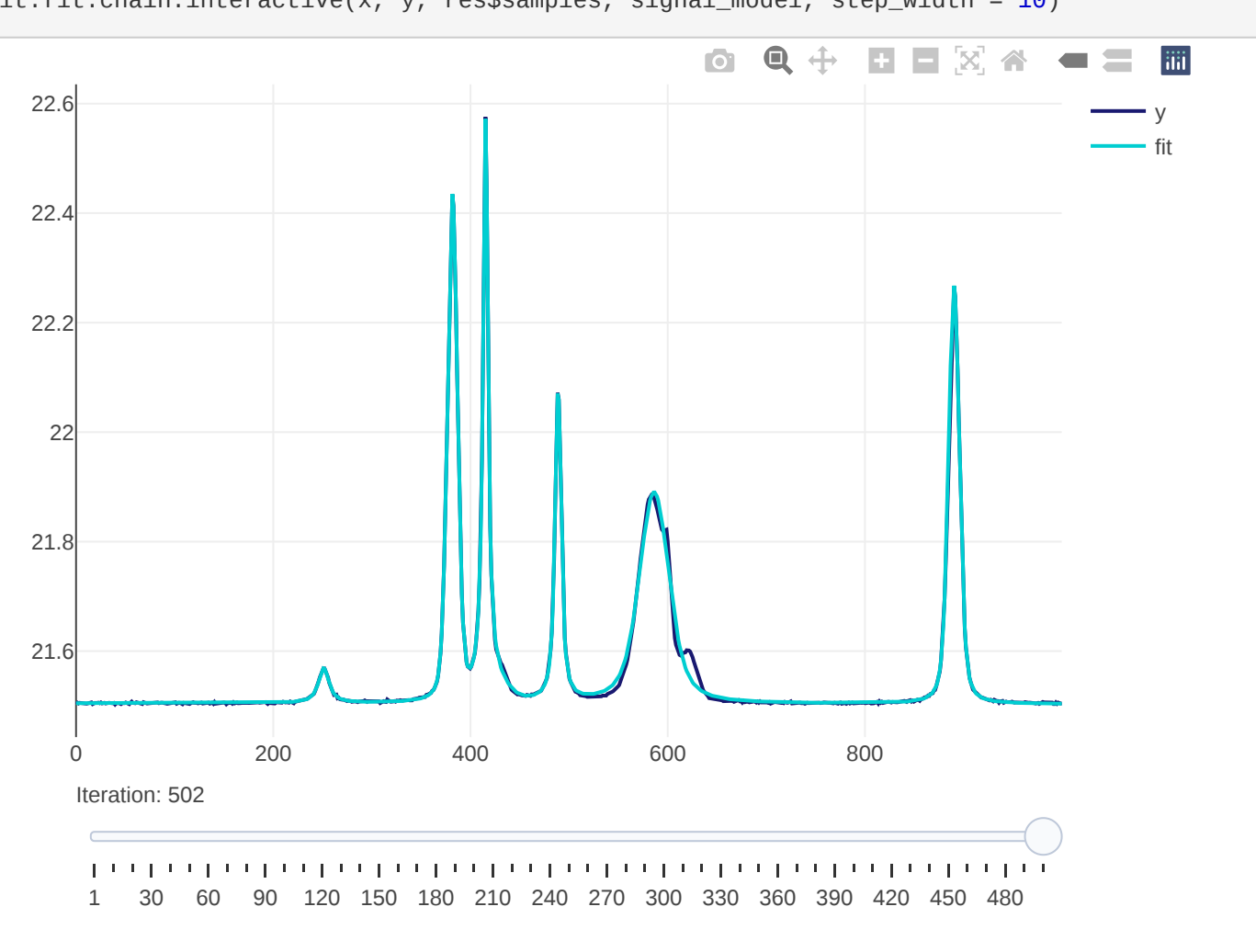
2.1.1.2 Burn-in

```
plt.fit.chain.interactive(x, y, res$samples[seq(1, 40)], signal_model, step_width = 2)
```



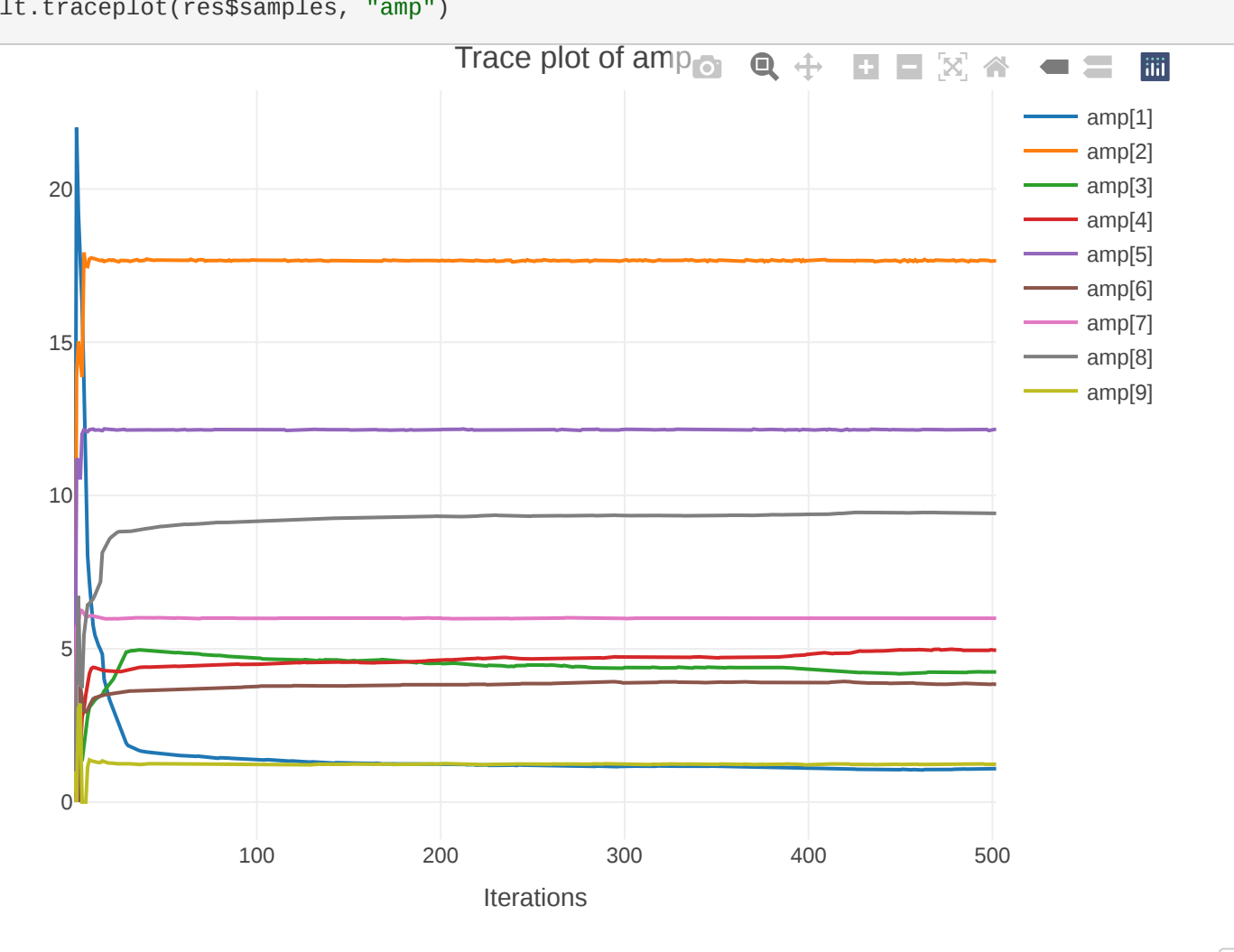
2.1.2 Traceplots

```
plt.fit.chain.interactive(x, y, res$samples, signal_model, step_width = 10)
```

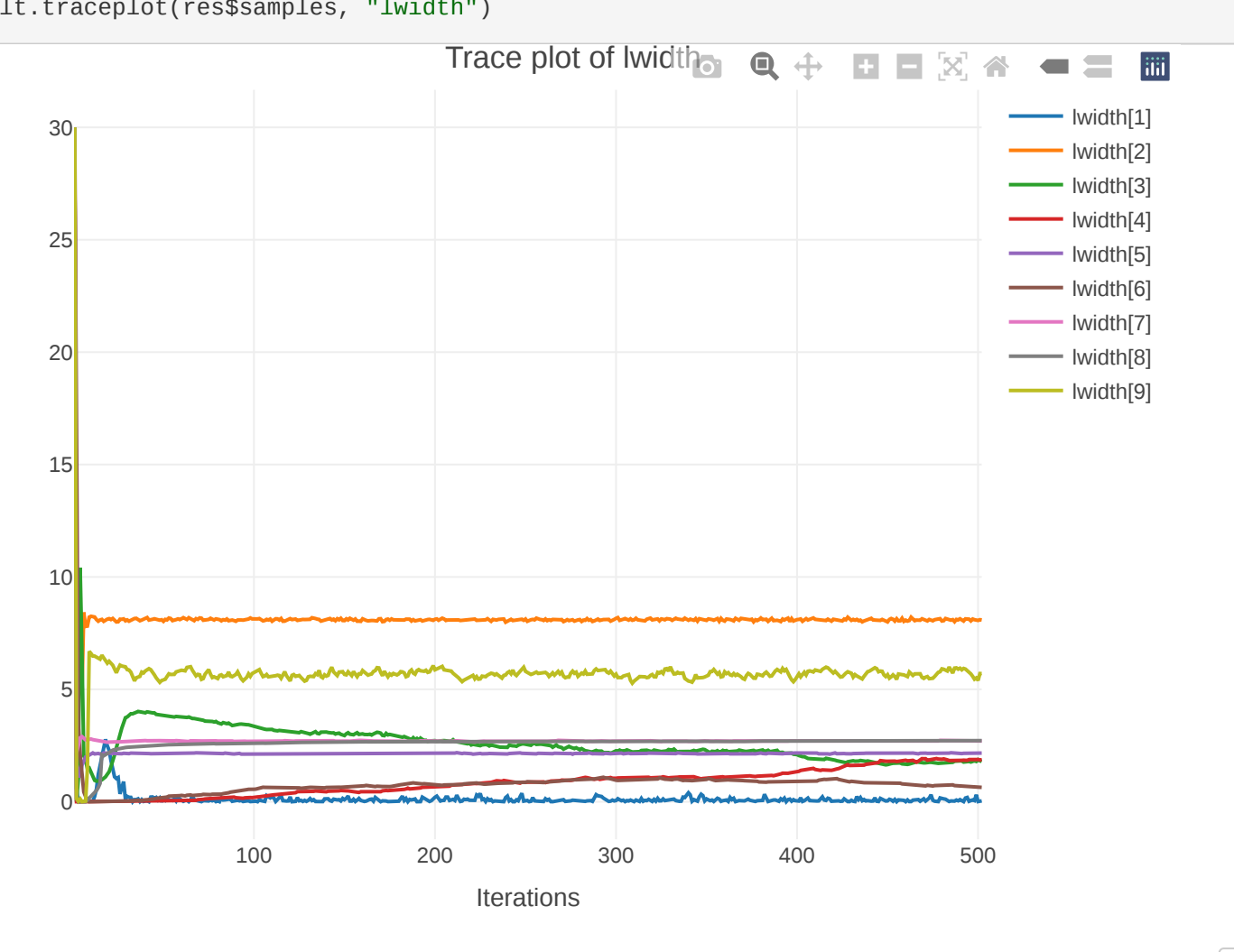



```
plt.traceplot(res$samples, "pos")
```

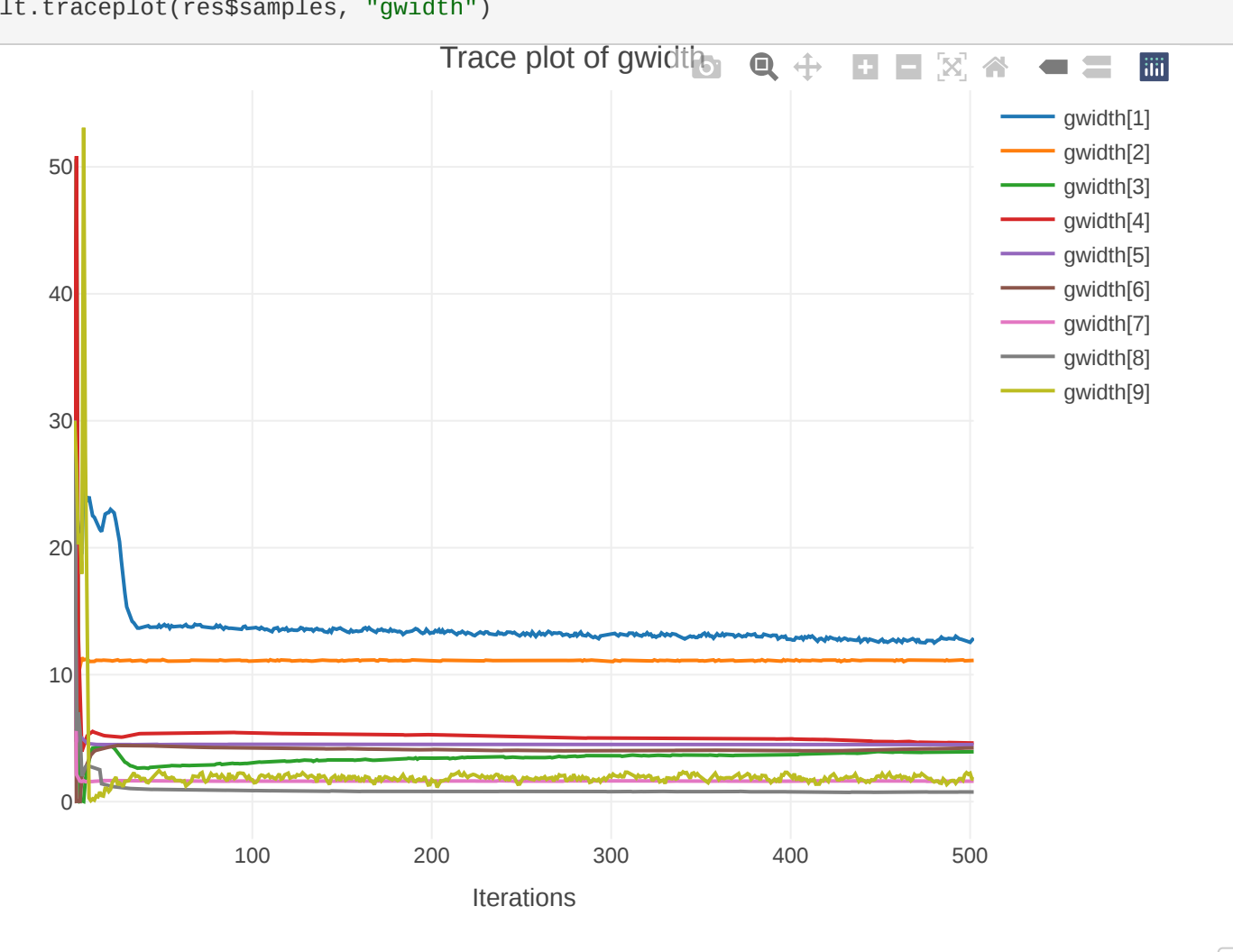

```
plt.traceplot(res$samples, "amp")
```



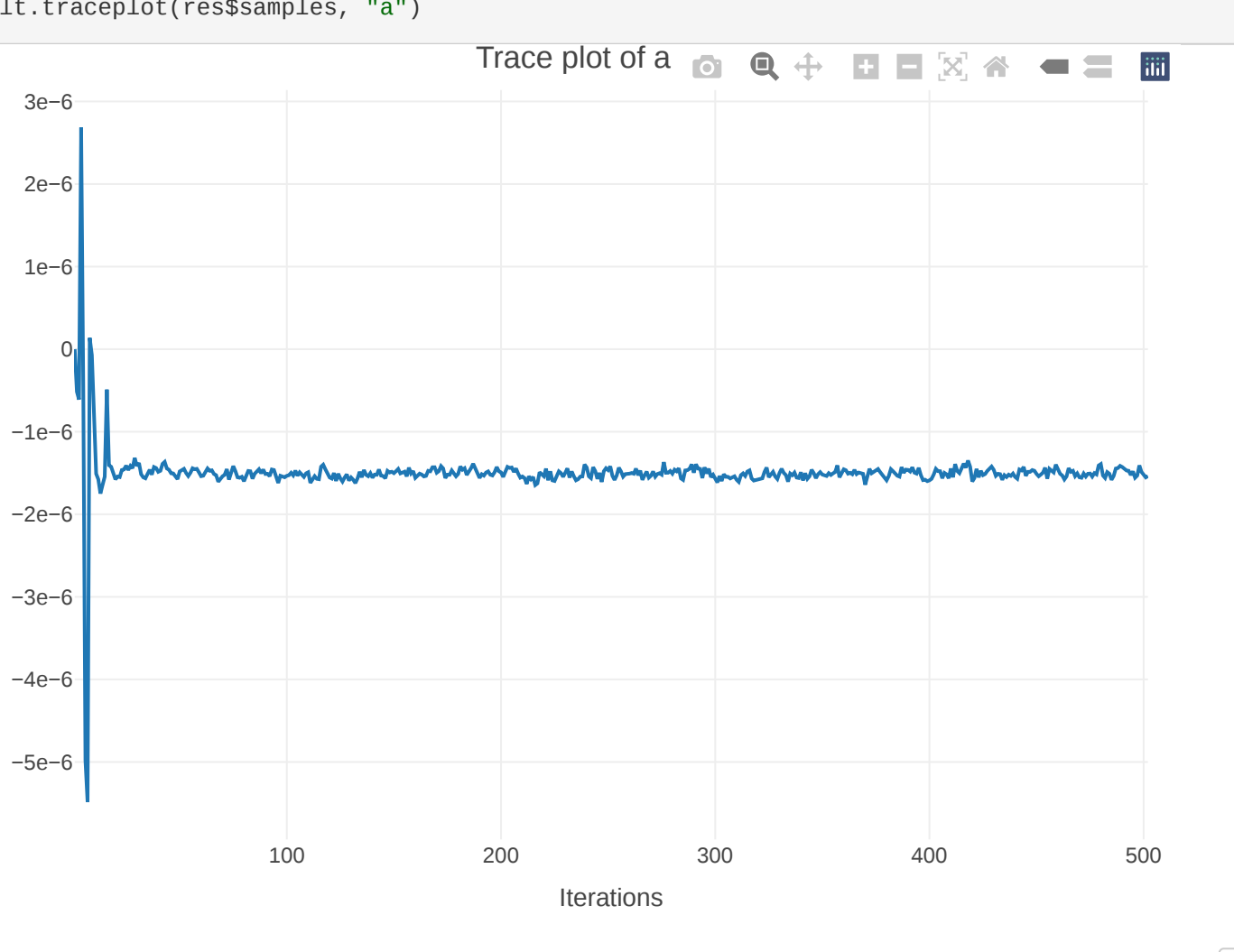

```
plt.traceplot(res$samples, "lwidth")
```



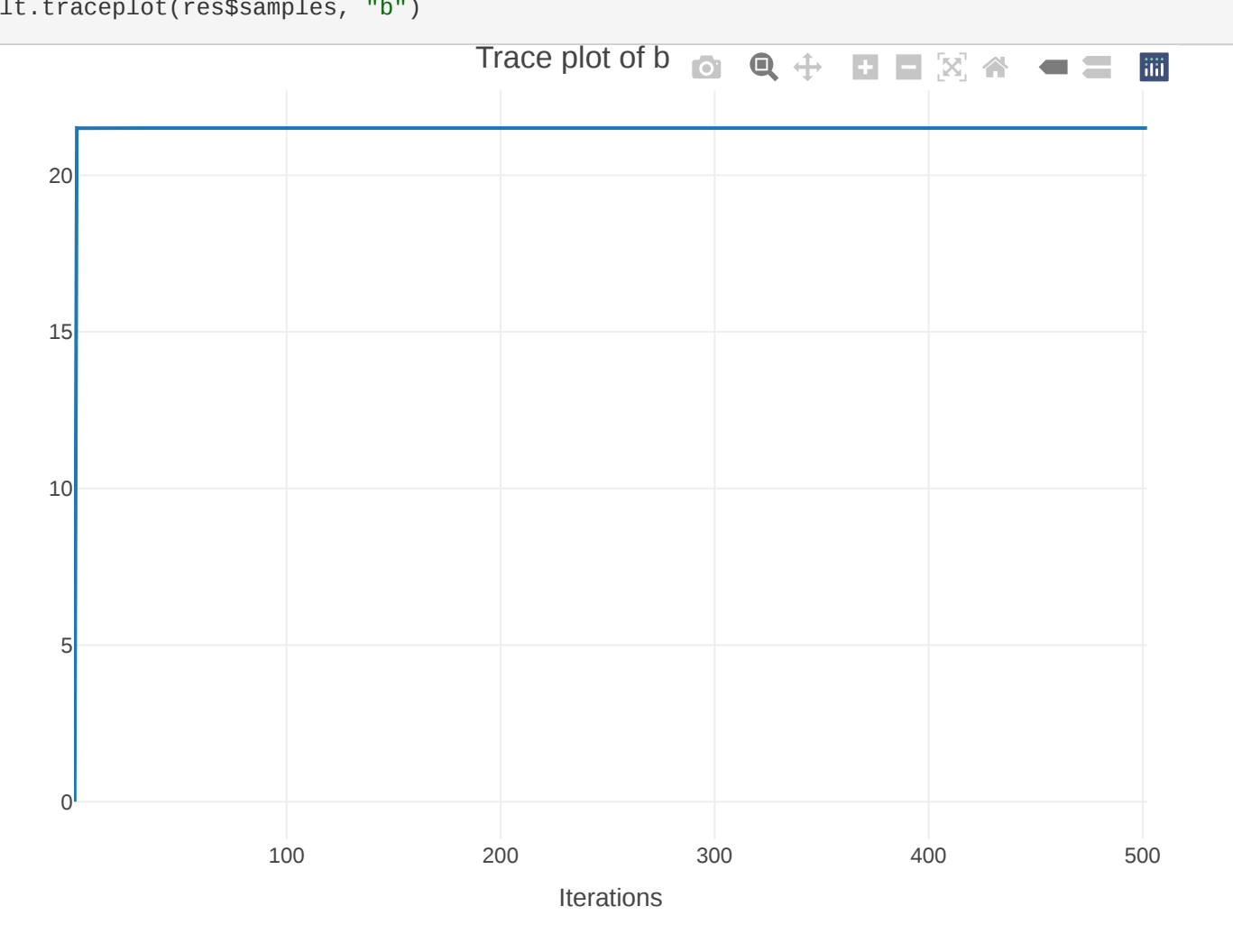

```
plt.traceplot(res$samples, "gwidth")
```




```
plt.traceplot(res$samples, "a")
```




```
plt.traceplot(res$samples, "b")
```



3 RJMCMC-burnin Gibbs Sampler

Gibbs Sampler starts by fitting one peak and adds a peak every `add_peak_every` iterations until `fit_kp` is reached at which point it just becomes a normal gibbs sampler. Helps resolving the peak deconvolution problem seen above and is an strong hint on the power of a full RJMCMC implementation.

- Better fit than EM
- Peak deconvolution (`pos[1]` and `pos[7]`) takes ages to resolve

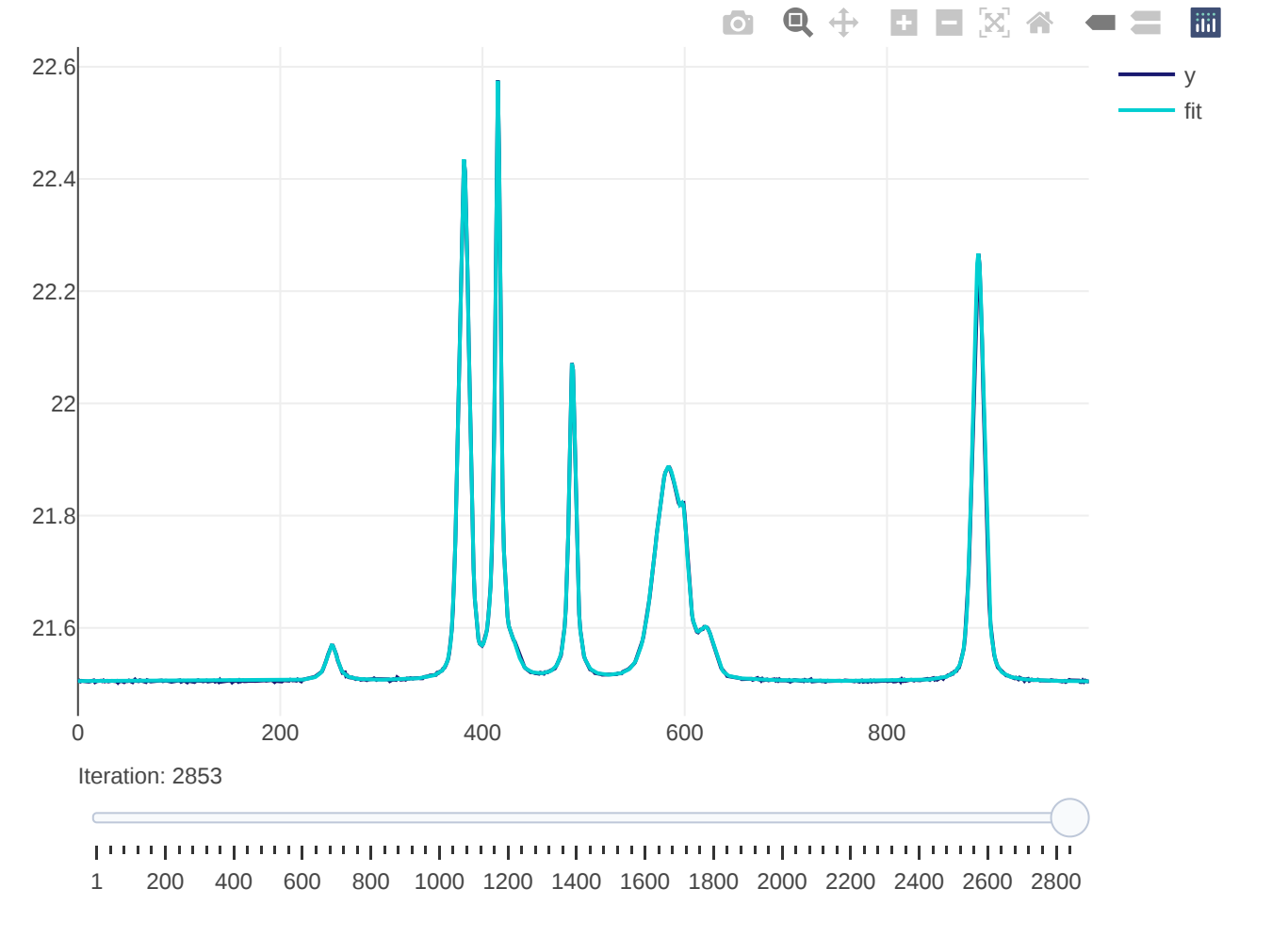
```
peak_spawner <- function(){
  params <- list(
          amp = rnorm(1, 1, 0.001),
          lwidth = rnorm(1, 0.1, 0.001),
          gwidth = rnorm(1, 4, 0.001),
          pos = runif(1, 100, 900),
          a = 0,
          b = 0
  )
  return(params)
}

res <- chains.rjmcmc_like_collapsed_slicer_gibbs(
        x,
        y,
        signal_model,
        peak_spawner,
        lower,
        upper,
        noise_sigma,
        fit_kp = fit_kp,
        add_peak_every = 10,
        iter = 5000,
        print_bar = TRUE,
        half_steps = TRUE,
        decorr_trafo = TRUE,
)
```

3.1 Results

3.1.1 Fit

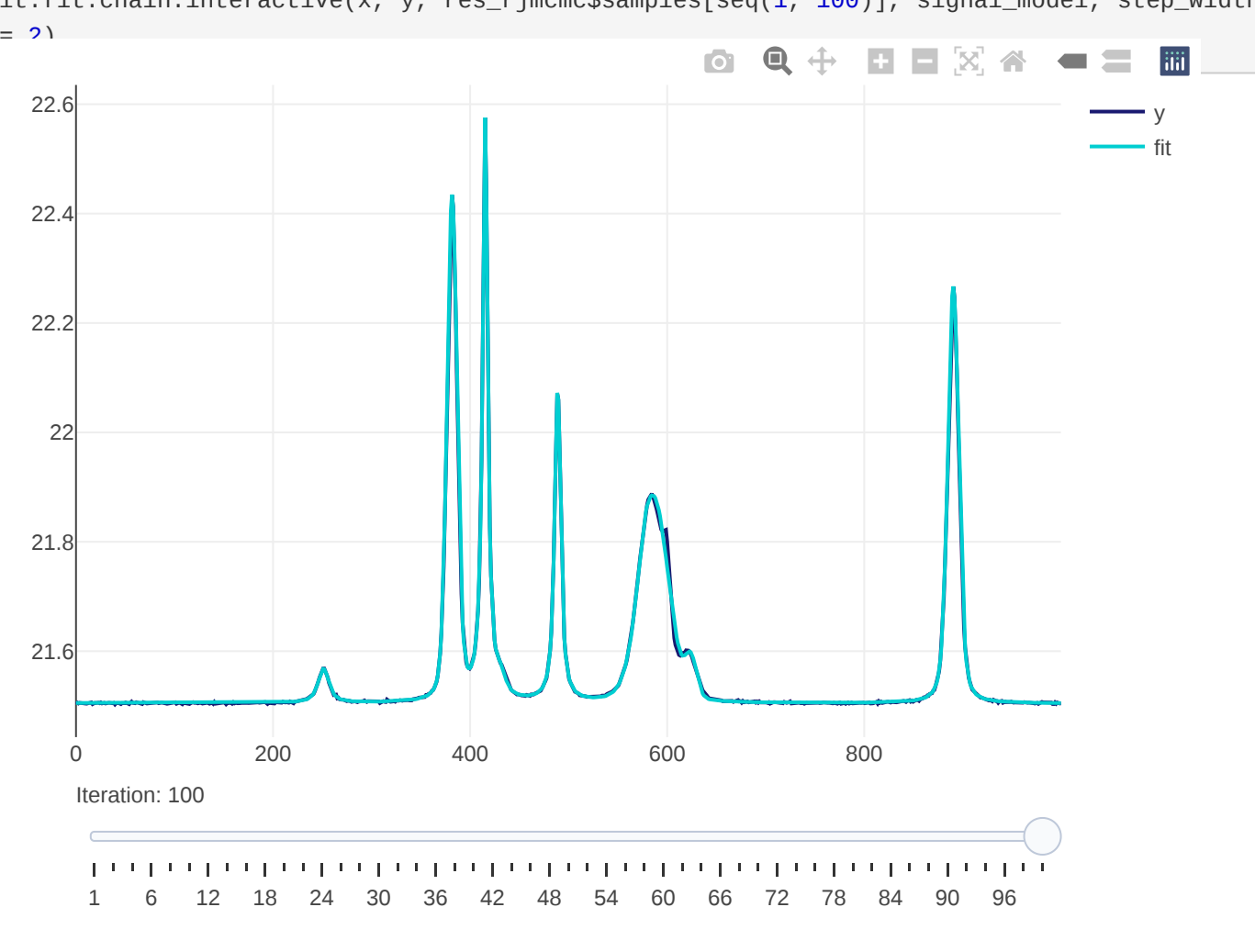
3.1.1.1 All

```
plt.fit.chain.interactive(x, y, res_rjmcmc$samples, signal_model, step_width = 40)
```



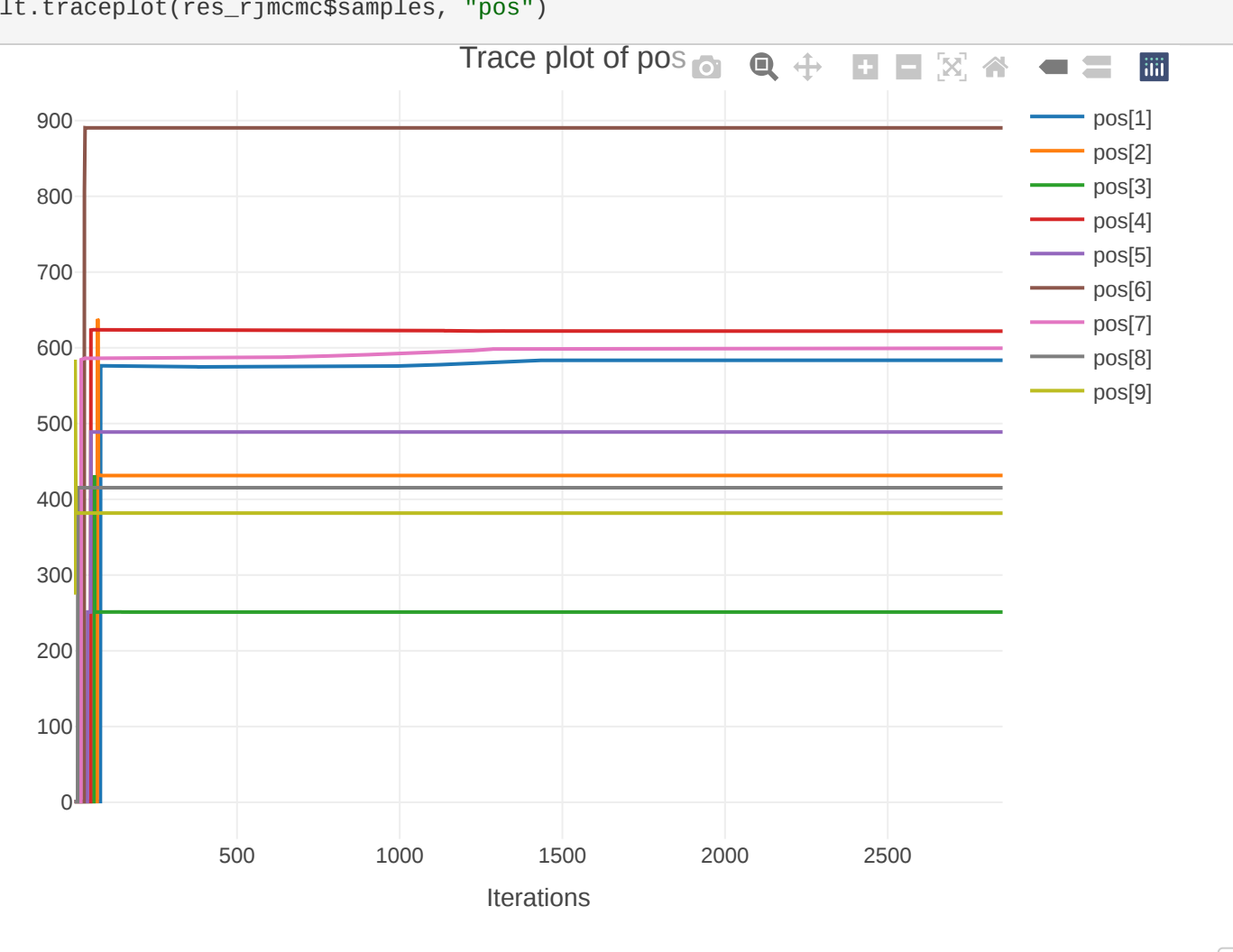
3.1.1.2 Burn-in

```
plt.fit.chain.interactive(x, y, res_rjmcmc$samples[seq(1, 100)], signal_model, step_width
 = 2)
```

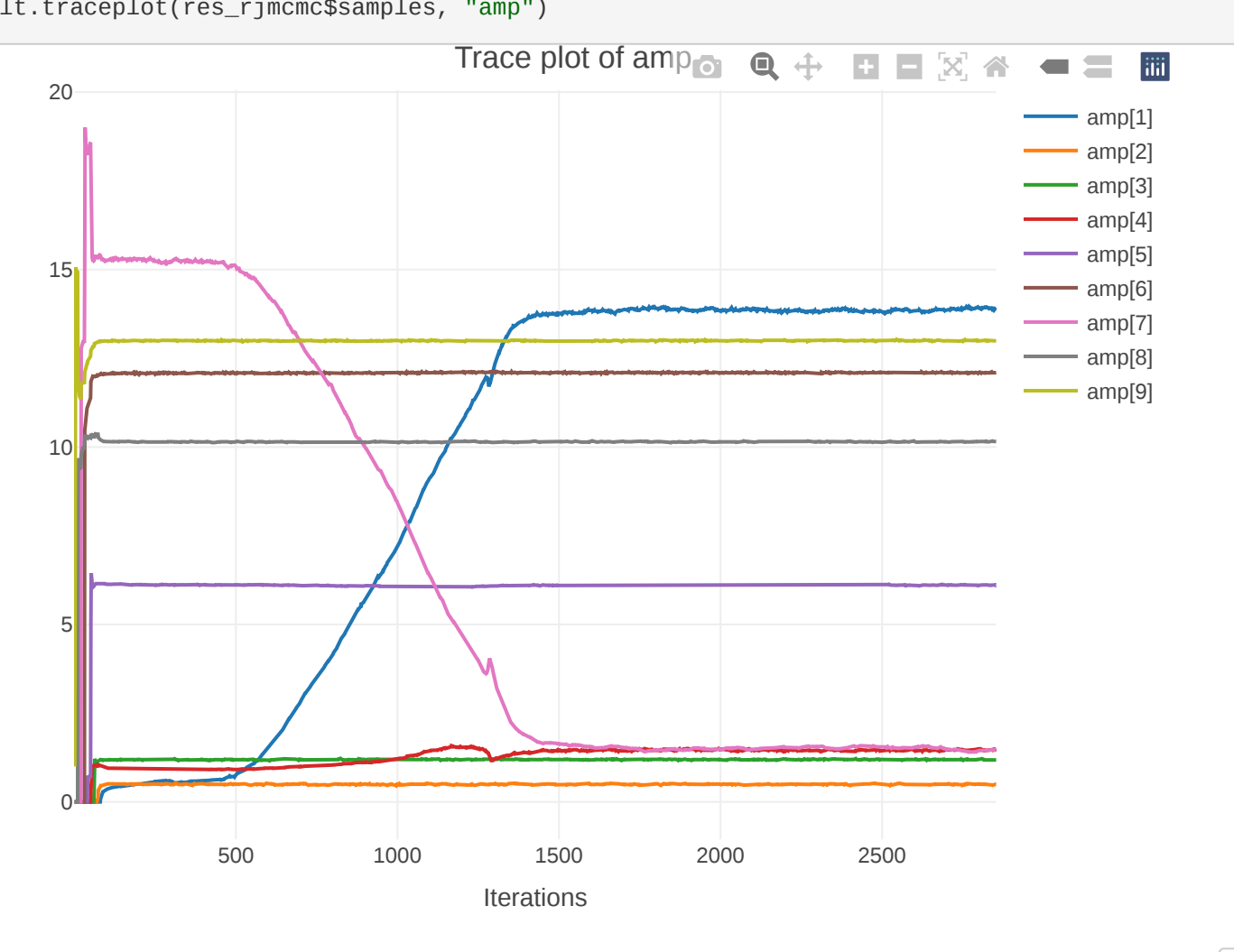


3.1.2 Traceplots

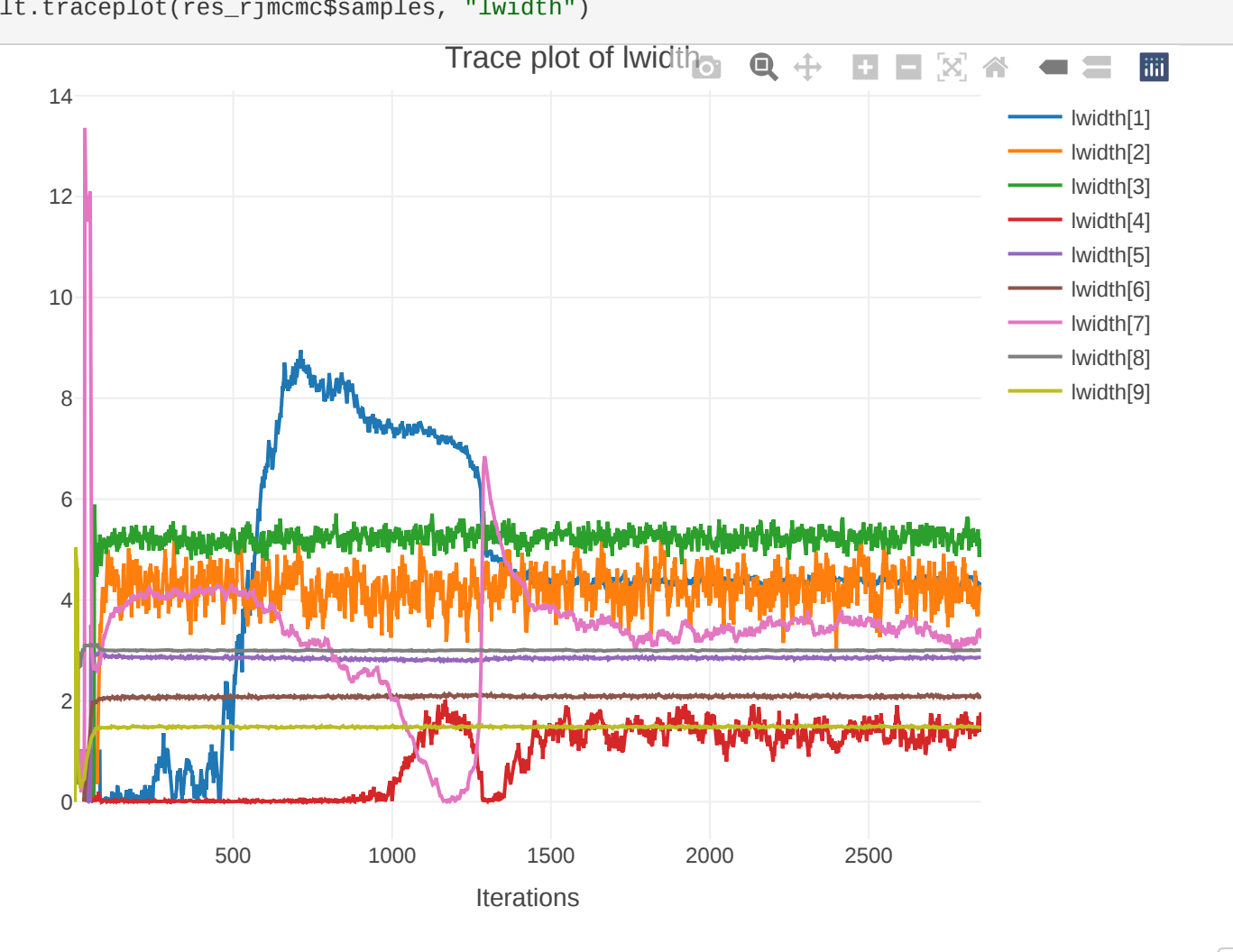
```
plt.traceplot(res_rjmcmc$samples, "pos")
```



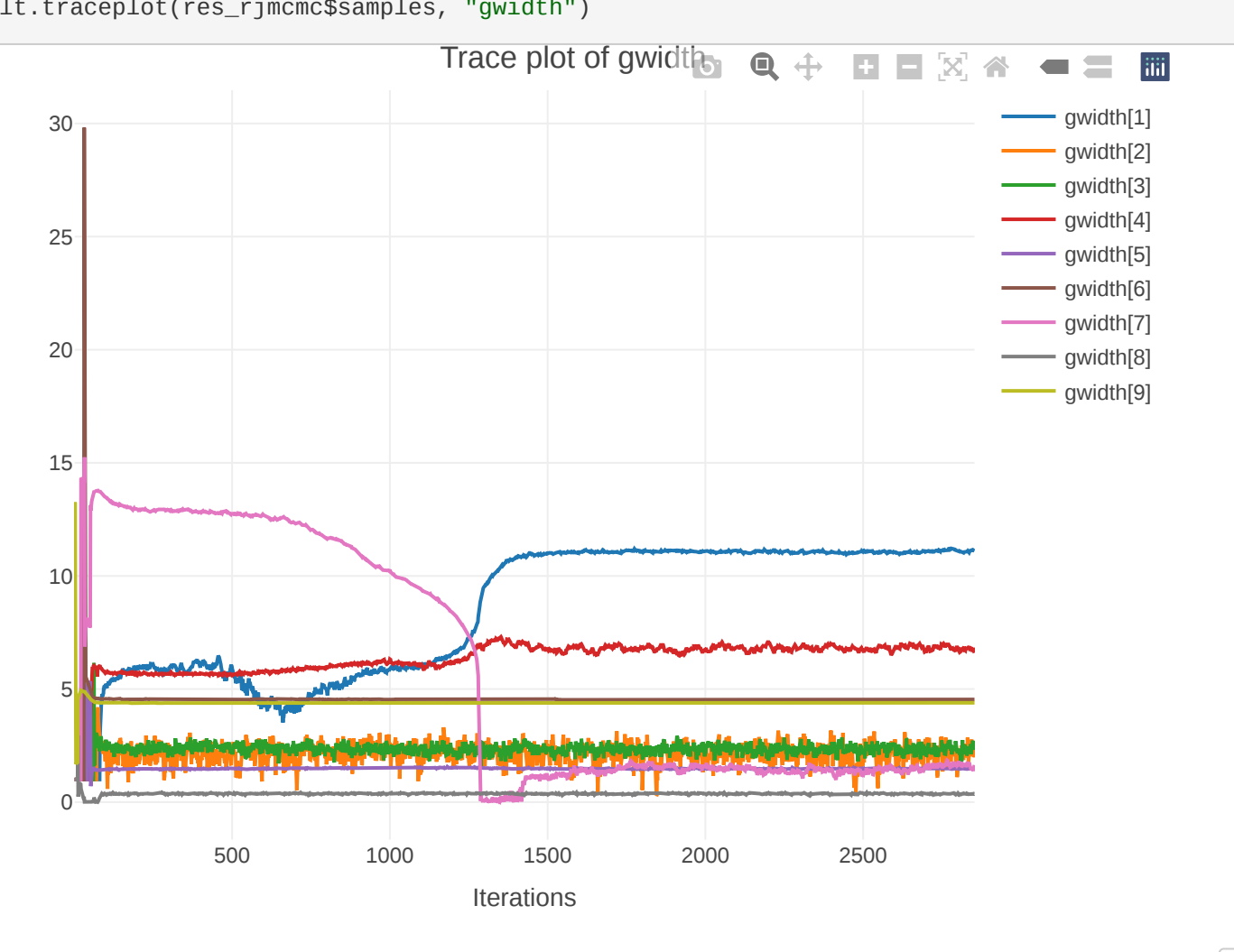

```
plt.traceplot(res_rjmcmc$samples, "amp")
```



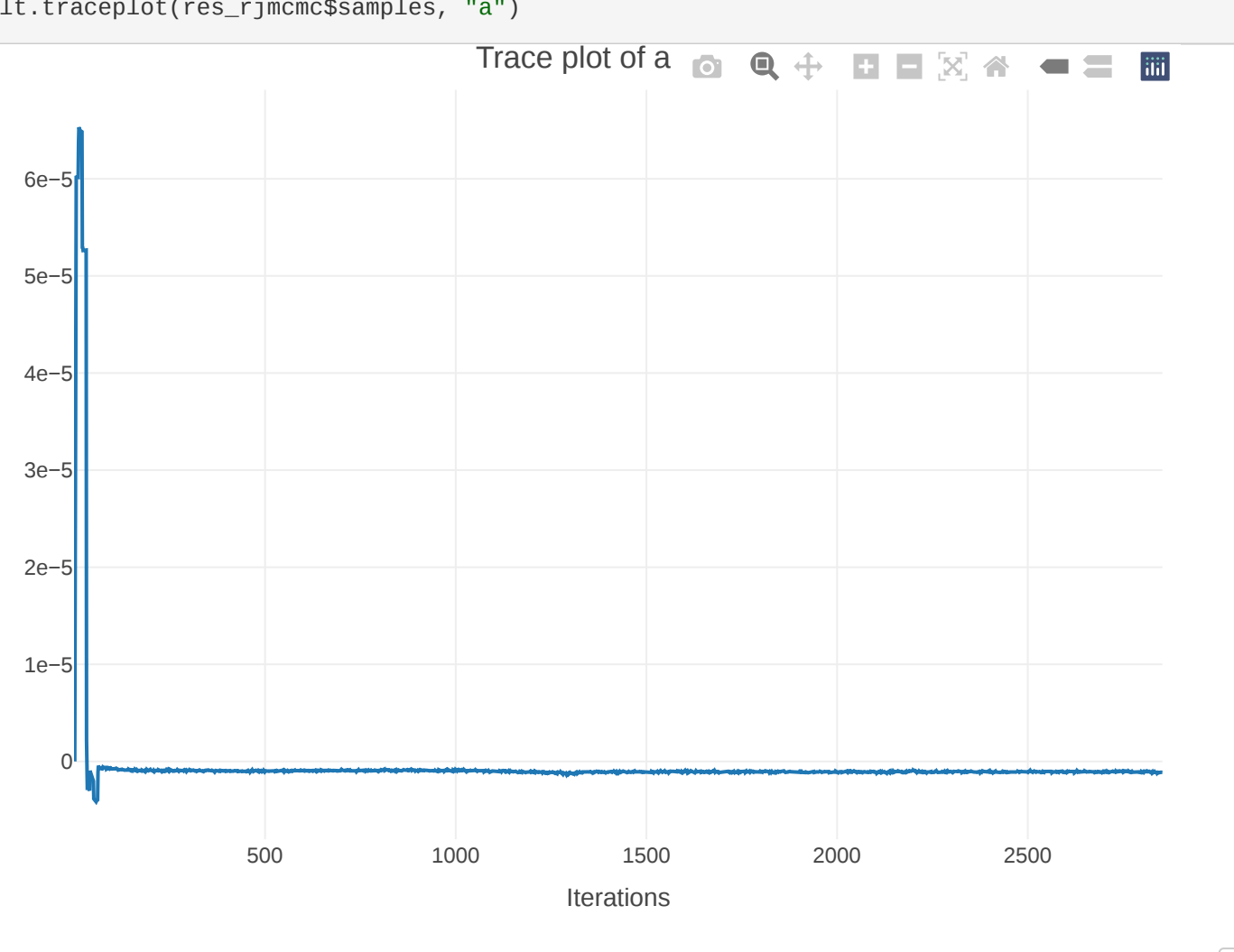

```
plt.traceplot(res_rjmcmc$samples, "lwidth")
```



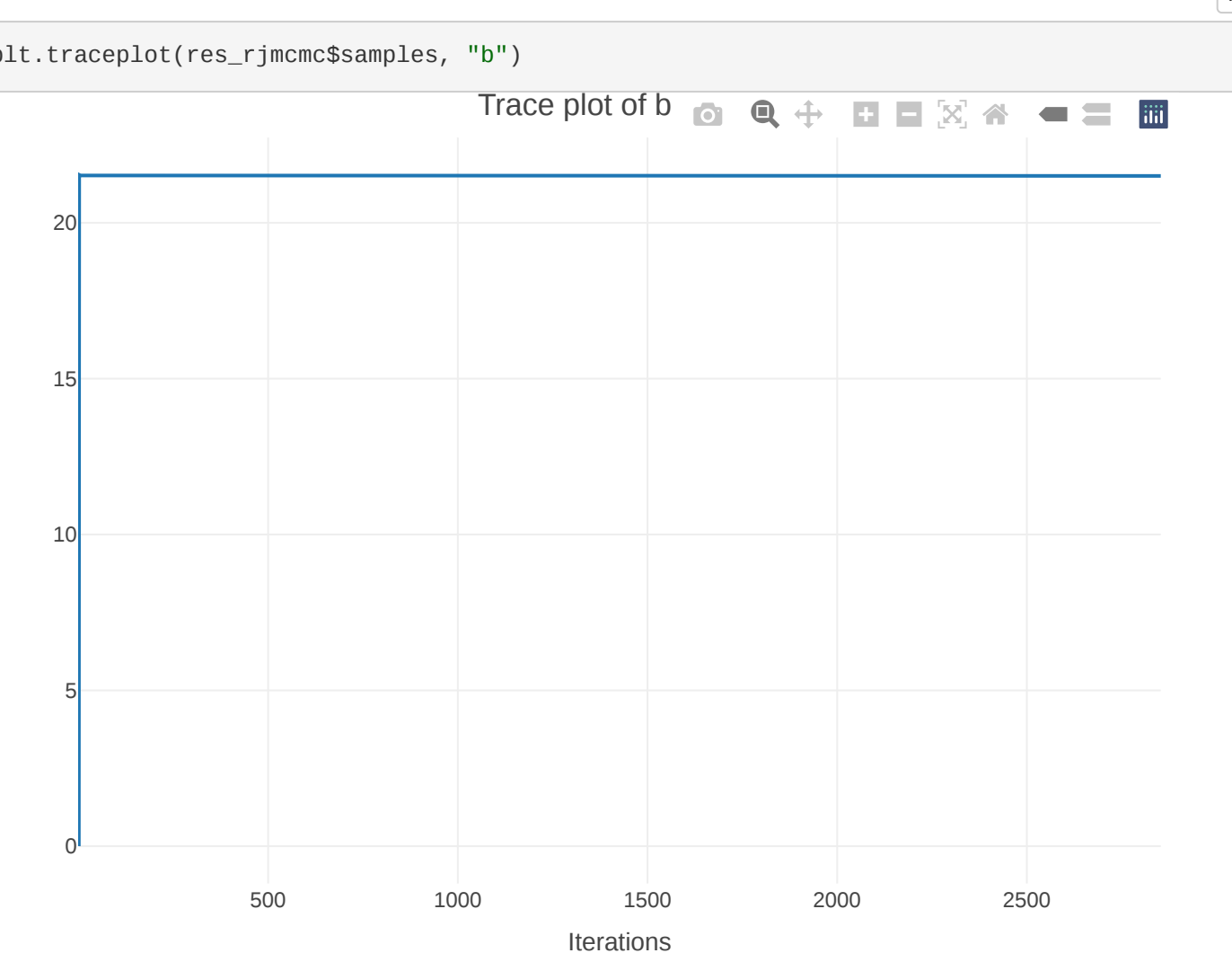

```
plt.traceplot(res_rjmcmc$samples, "gwidth")
```




```
plt.traceplot(res_rjmcmc$samples, "a")
```




```
plt.traceplot(res_rjmcmc$samples, "b")
```



4 Todo

- **Critical Bug** amp, a, b are not sampled instead they are set to roughly their means (not really either); Easy to fix.
- **Not a bug**: Uses a safeguard which prevents messing up the fit if the slice sampler fails, this however destroys the gibbs sampler property and causes rejection plateaus.
- **Done**: nu, be transformation becomes bad (why?) if either lwidth or gwidth is really small compared to the other, should switch to nu,be only if `min(lwidth/gwidth, gwidth/lwidth) > 0.05`
- Peak deconvolution is the central problem (i.e. fitting strongly overlapping peaks). See the second fit where only 1 Peak is misplaced. RJMCMC would most likely instantly solve this problem as it would *add* an peak on the right flank shortly increasing kp to 10 and then removing the misplaced peak putting kp to 9 again. This is an massive advantage that seems to be impossible to model with fixed kp. If a given kp is needed the kp prior could be slowly changed to degenerate to only the given value of kp.
- **Done** prop.voigt.rnd_start spawns to wide peaks for gibbs (the tuning for the spawner is very impactful)
- **Do Speed ups** Develop heuristic to evaluate pmvnorm fast; better slice sampler; better early termination of slice sampler. Run profiler to figure out if pmvnorm is actually the bottleneck or X construction. If it is X could use caching and speed up code by 90%

1. RWTH Aachen, philipp.meissner@rwth-aachen.de↩