ODEs in Stan

StanCon 2018 Helsinki

August 26, 2018

Daniel Lee

daniel@generable.com

Generable

Before we begin

Overall Structure

1. Write ODEs in Stan; simulate from ODE
2. Estimate parameters of ODE for a single observation
3. Estimate parameters of ODE for multiple observations
4. Parallelization

What you should be comfortable with

- Writing Stan programs, including functions
- Running Stan from interface of choice.
  For today:
    - CmdStan v2.18.0: running Stan
    - RStan: exporting data, reading fits
- Important: familiarity with Stan program blocks and Stan types

What we don't have time to cover. =(

- Full tutorial in Bayesian inference
- Full tutorial in Stan
- Full tutorial in ODEs
- Expert-level debugging through ODEs

Some Notes

- Working example is simple: simple harmonic oscillator
  - Highlights mechanics of ordinary differential equations (ODEs) in Stan
  - There are more complications with complex examples.
    Feel free to use your own example.

- To get the most out of this course:
  - Work together!
  - Ask questions.

Last thing... install CmdStan v2.18.0

- Download the latest CmdStan:
  https://github.com/stan-dev/cmdstan/releases/tag/v2.18.0
- Unzip the directory. Then type:
  - `make build -j4`

    (or however many cores)

- Also clone https://github.com/generable/stan-ode-workshop

Write ODEs in Stan

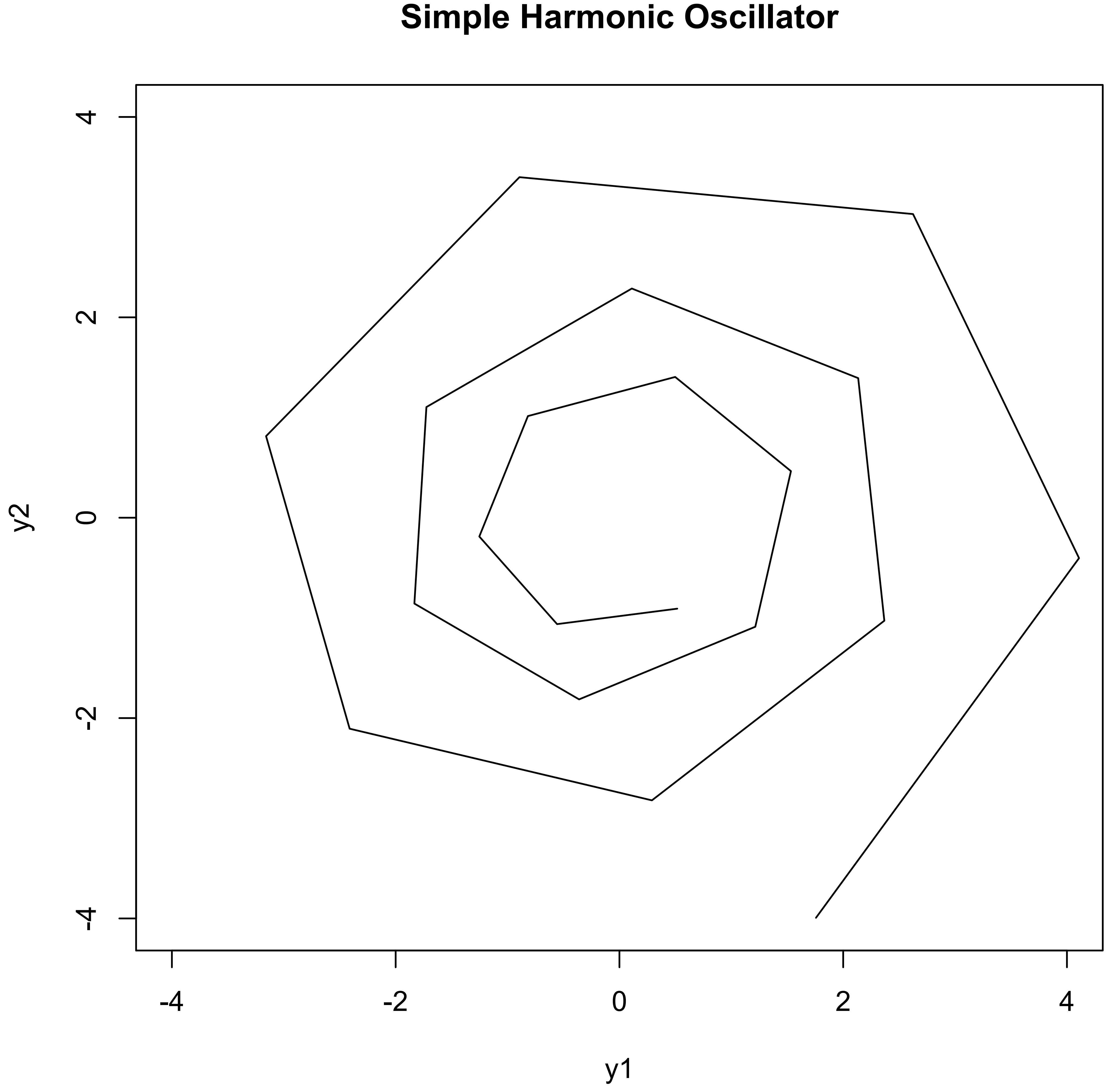
What's an ODE?

Differential equation describing the derivatives of a function with respect to an independent variable.

- What does that mean?
  - There's an independent variable, **t**.
    Usually time, but could represent anything.
  - There's a (multivariate) independent variable, **y**, that depends on t.
    We want **y(t)** for some set of t, but we don't know how to compute it directly.
  - We have a function that describes the derivative of the dependent variable with respect to the independent variable:
    $\mathbf{f'(t)} = dy(t)/dt = g(t, x, \theta)$
    where x are data, θ are parameters

Example: simple harmonic oscillator

- Two states.
- Equilibrium position: (0, 0)
- Restoring force proportional to displacement (with friction)

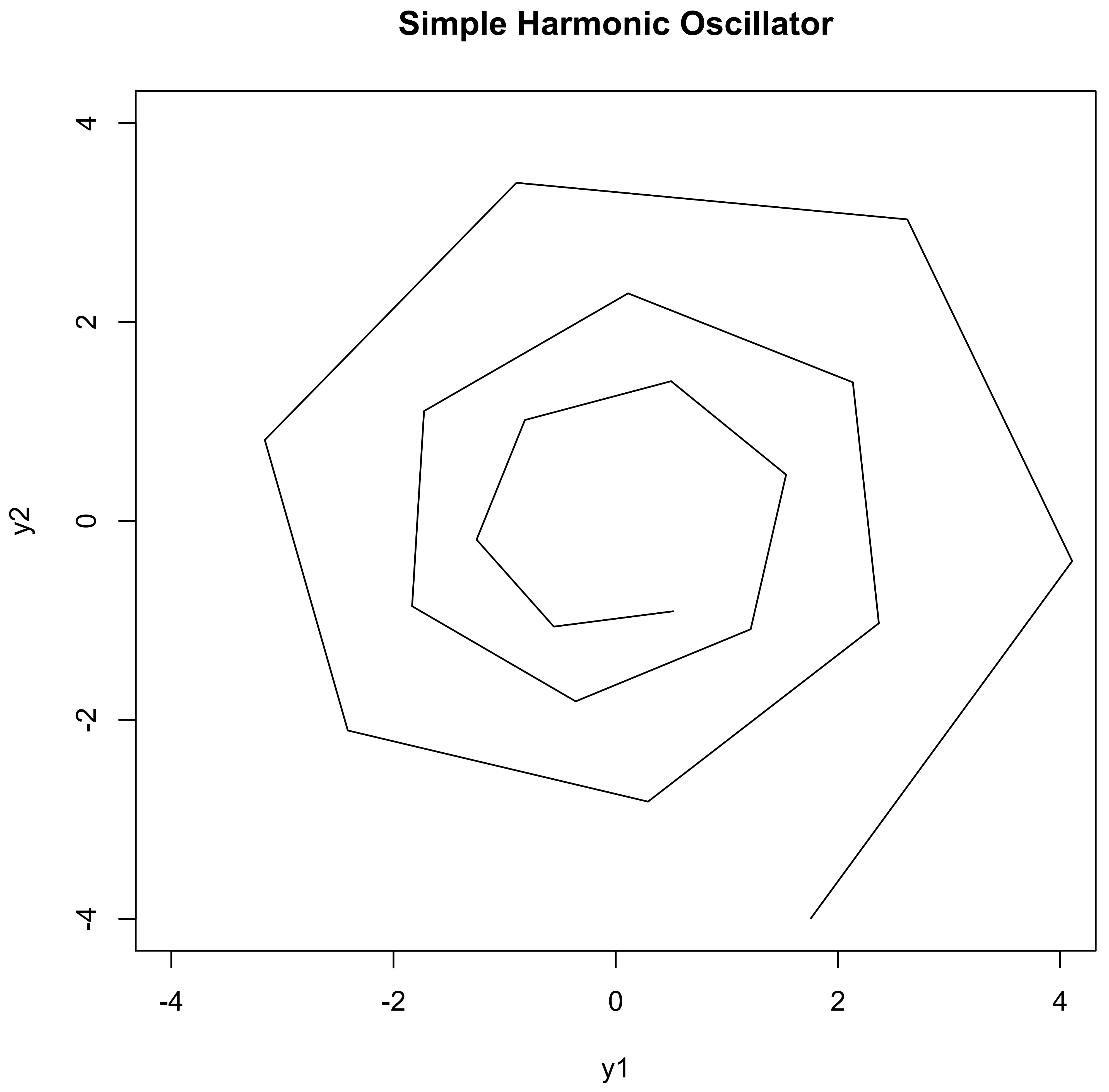


Example: simple harmonic oscillator

- Two states. y1, y2.
- $$\frac{dy_1(t)}{dt} = y2$$
- $$\frac{dy_2(t)}{dt} = -y_1 - \theta * y_2$$

- If we know the initial conditions, $y_1(t_0)$ $y_2(t_0)$, at time t_0, and θ,

  we can solve for times $t_0 < t_1 < t_2 < \dots$



Stan ODE function

- The Stan function must have this signature:
  - `real[] ode_system(real t, real[] y, real[] theta,`
    `real[] x_r, int[]x_i)`

- Simple harmonic oscillator:

```
real[] sho(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {
   real dy1_dt = y[2];
   real dy2_dt = -y[1] - theta[1] * y[2];
   return { dy1_dt, dy2_dt };
}
```

$$\frac{dy_1(t)}{dt} = y2$$

$$\frac{dy_2(t)}{dt} = -y_1 - \theta * y_2$$

Let's break it down

```
real[] sho(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {
  real dy1_dt = y[2];
  real dy2_dt = -y[1] - theta[1] * y[2];
  return { dy1_dt, dy2_dt };
}
```

- Number of states: 2.
- Return: dy1/dt, dy2/dt, evaluated at time t with state y
- Arguments:
    - t, independent variable
    - theta, ODE arguments that depend on parameters
    - x_r, ODE arguments that depend only on data
    - x_i, integer ODE arguments that depend only on data

integrate_ode_bdf() and integrate_ode_rk45()

- These functions solve for the states at the times given.
  They take the same arguments.
  - BDF: backward-differentiation formula for stiff systems
  - RK45: 4th and 5th order Runga-Kutta method for non-stiff systems

- ```
  real[] integrate_ode_*(ode_function, real[] initial_states, real t0,
    real[] times, real[] theta, real[] x_r, int[] X_i)
  ```

- The new arguments:
  - `ode_function: real[] f(real, real[], real[], int[])`
  - `real[] times. Times to solve for.`
  - `return: states at the times specified`
  ```

Generate data from this ODE

- From CmdStan:
  - make ../stan-ode-workshop/sho_sim
  - make ../stan-ode-workshop/sho_sim.exe
- From stan-ode-workshop
  - ./sho_sim sample
- Read from R:
  - library(rstan)
  - fit <- read_stan_csv("output.csv")
  - plot(extract(fit)$y[1,,])
- Why is every "row" of y the same?
- Exercise: adjust program so it accepts times as data
- Exercise: add noise in the generation process

Finish simulation

- Create a new variable, y_hat, with normal(0, 0.5) noise for both states.

- Generate dataset
    - data <-list()
    - data$T <- 20
    - data$ts <- 1:20
    - data$t0 <- 0
    - data$y0 <- c(1, 0)
    - data$y_hat <- y_hat[1,,]

    - stan_rdump(ls(data), "sho.data.R", envir = list2env(data))

Questions

- Can the ODE be discontinuous?
- Can a model have more than one differential equation?
- If I can solve the ODE analytically, should I?

Estimating from a single observation

Write a Stan program to fit the data

- What should be estimated?
  - sigma
  - theta
  - y0?

- Set reasonable priors (or you'll get in trouble)
  - sigma ~ normal(0.5, 0.1);
  - theta ~ normal(0, 0.5);
  - y0 ~ normal(0, 1);

Run

- ./sho_fit sample data file=sho.data.R output refresh=10
- fit <- read_stan_csv("output.csv")
- print(fit, c("sigma", "y0", "theta"))

More specifics for integrate_*()

- Times:
  - Must be sorted
  - times > t0
  - times must be data only
- x_r must be data only
- x_i must be data only

- Three optional parameters
  - relative tolerance: default 1e-10
  - absolute tolerance: default 1e-10
  - max_steps: default 1e8

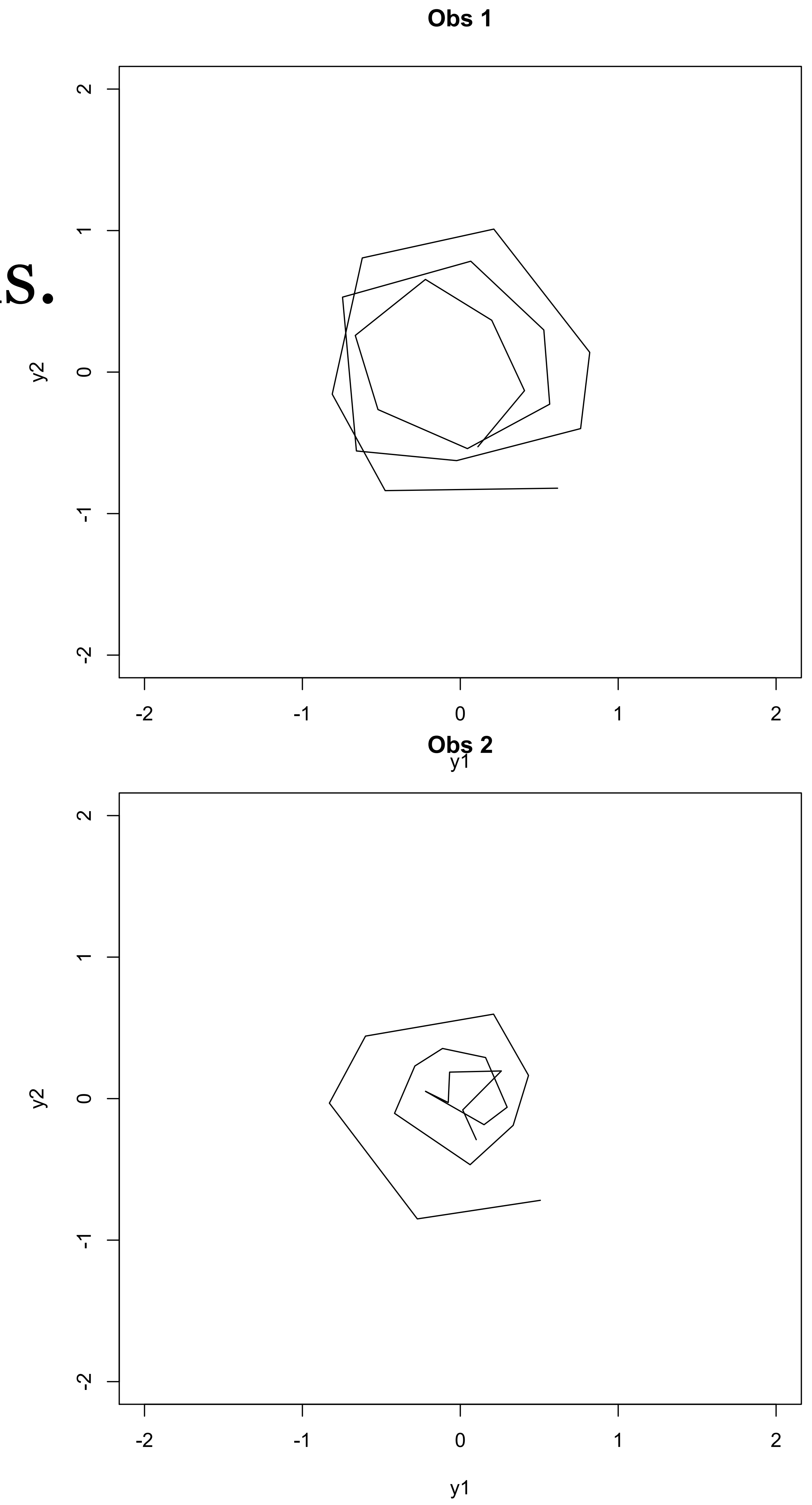
Some more implementation details

- We've only written $\frac{dy}{dt}$
- We solve for y(t), for some set of ts that we specify. That is, we get:
  - y(ts[1]), y(ts[2]), ..., y(ts[T])
  - we've solved for the y(t) numerically
- But wait... Stan needs gradients with respect to parameters!
  - We also need $\frac{dy(ts[1])}{d\theta}, \frac{dy(ts[2])}{d\theta}, \ldots \frac{dy(ts[T])}{d\theta}$
  - The ODE system we're actually solving for in Stan has this many states:
    - (# of states) x (# of thetas)
    - If we need to solve for initial conditions, we also need (# of thetas)

Estimate from multiple observations

Let's update simulation

- We want 10 sets of observations
- Let's draw 10 thetas, then generate observations.
- What does the data look like?
  - real y_hat[10, T, 2];
- Generate data.
  If we're behind, use sho_multiple.data.R



Exercise: extend sho_fit.stan to multiple

- Hints:
  - You'll want a loop over each individual
  - real theta[10];
  - Run with iter=0

- theta ~ normal(0.15, 0.1);

Review

- ODE function is the same
  - simulation
  - fitting to one set of observations
  - fitting to multiple set of observations
- ODEs are part of the model
  - Statistical model built on top
  - Estimates of parameters of the model happen simultaneously

Parallelization

Overview

- We write a function to handle each observational unit separately
    - Key requirement: the function is able to handle an independent part
- We pass this function into the **map_rect(...)** function
    - This "maps" the arguments to the function
    - All data is rectangular, so we'll need to pad
- From the language, this is all that's necessary
- We need separate build instructions for CmdStan (or interface of choice)

The function we need to write

- (vector, vector, data real[], data int[]) : vector
  - 1st vector: shared parameters
  - 2nd vector: individual parameters (both of these would have been theta)
  - data real[]: real data
  - data int[]: int data
  - return: vector. We have the flexibility to return what we want.

- This function will handle each observation

map_rect

- ```
  vector map_rect( (vector, vector, real[], int[]):vector f,
                   vector phi,
                   vector[] thetas,
                   data real[,] x_rs,
                   data int[,] x_is)
  ```
- Stan will loop over all the parameters.
- The output will just be concatenated into one vector.
  ```

Let's write the function for one observation

```
vector individual_ode(vector phi, vector theta,
                      real[] x_r, int[] x_i) {
  int T = x_i[1];
  real y[T, 2] = integrate_ode_bdf(sho, x_r[1:2], 0.0, x_r[3:(T+2)],
    to_array_1d(theta), x_r, x_i);
  return append_row(to_vector(y[, 1]), to_vector(y[, 2]));
}
```

- Still calling integrate_ode_bdf()
- Return is a single vector of y[, 1] and y[, 2] appended.

Make the single function work in a loop

- See: sho_fit_multiple_parallel_start.stan
- Important parts:
  - Setting up data

```
transformed data {
  int x_i[N] = rep_array(T, N);
  real x_r[N, T + 2];
  vector[0] phi;

  for (n in 1:N) {
    x_r[n, 1:2] = y0[n];
    x_r[n, 3:(T + 2)] = ts;
  }
}
```

  - Unpacking the return

```
transformed parameters {
  real y[N, T, 2];

  for (n in 1:N) {
    vector[T * 2] result = individual_ode(phi, to_vector(theta[n]), x_r[n], x_i);
    y[n, , 1] = to_array_1d(result[1:T]);
    y[n, , 2] = to_array_1d(result[(T + 1) : (2 * T)]);
  }
}
```

Using map_rect

- See: sho_fit_multiple_parallel.stan
- Important parts
  - Single call to map_rect():

```
transformed parameters {
  real y[N, T, 2];

  {
    vector[N * T * 2] result = map_rect(individual_ode, phi, theta, x_r, x_i);
    for (n in 1:N) {
      int start = T * (n  - 1);
      y[n, , 1] = to_array_1d(result[(start + 1) : (start + T)]);
      y[n, , 2] = to_array_1d(result[(start + T + 1) : (start + 2 * T)]);
    }
  }
}
```

To enable threading

- We need to rebuild the Stan program. Threads is easier to set up under linux + Mac
  - Open a file, make/local

    ```
    CXXFLAGS += -DSTAN_THREADS
    ```
  - Rebuild program

    ```
    touch ../stan-ode-workshop/sho_fit_multiple_parallel.stan
    make ../stan-ode-workshop/sho_fit_multiple_parallel
    ```
  - Set environment variable

    ```
    export STAN_NUM_THREADS=-1
    ```
  - Run! It'll run on multiple threads.

What we didn't cover

- Ragged data
- Censoring
- Speeding up the code
- Debugging through an ODE

Thank you