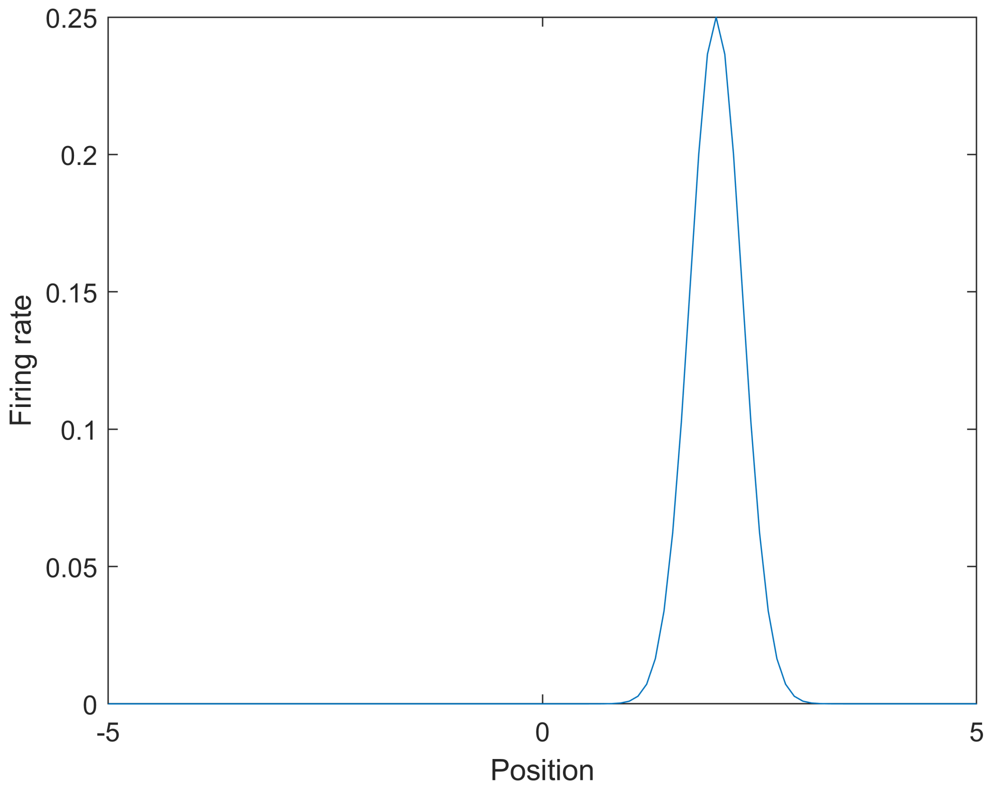
Simulated information-limiting correlations

Defining the place fields

Start with the functional profile of the place fields as a gaussian function:

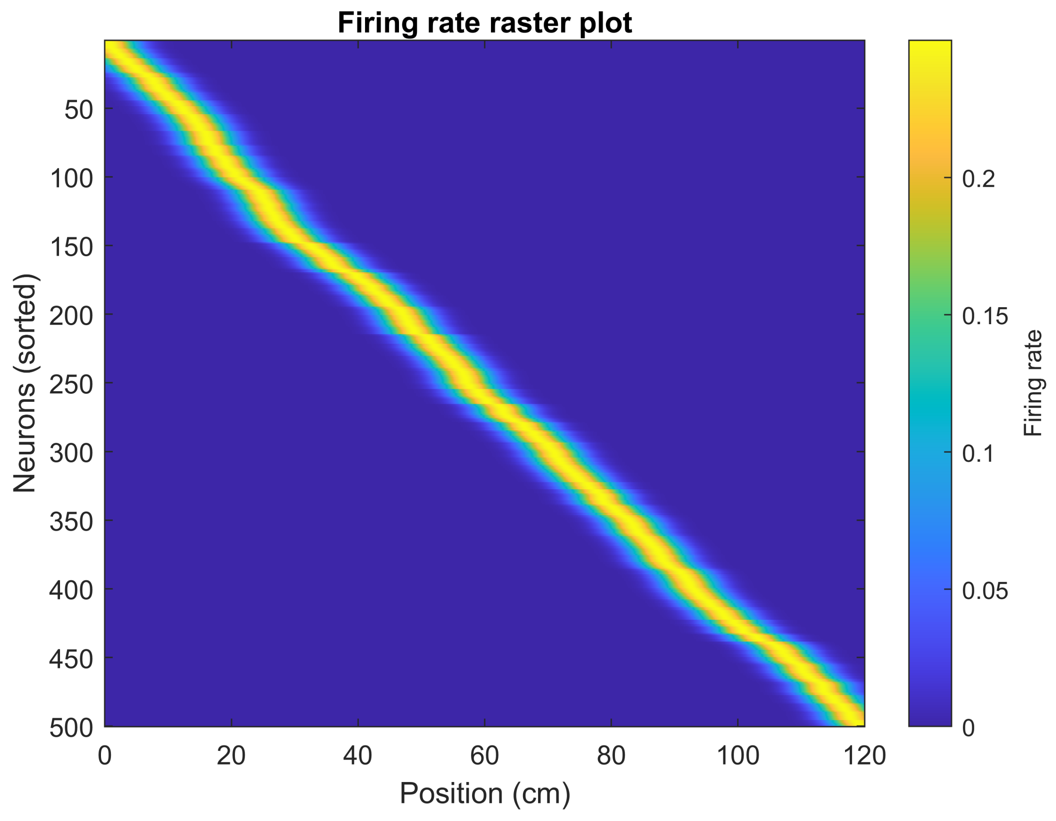
```
max_rate = 0.25;
g_field = @(x, m, s) max_rate.*exp(-((x-m)./s).^2 ./ 2);

figure;
x = -5:0.1:5;
plot(x, g_field(x,2,0.3));
xlabel 'Position'
ylabel 'Firing rate'
```



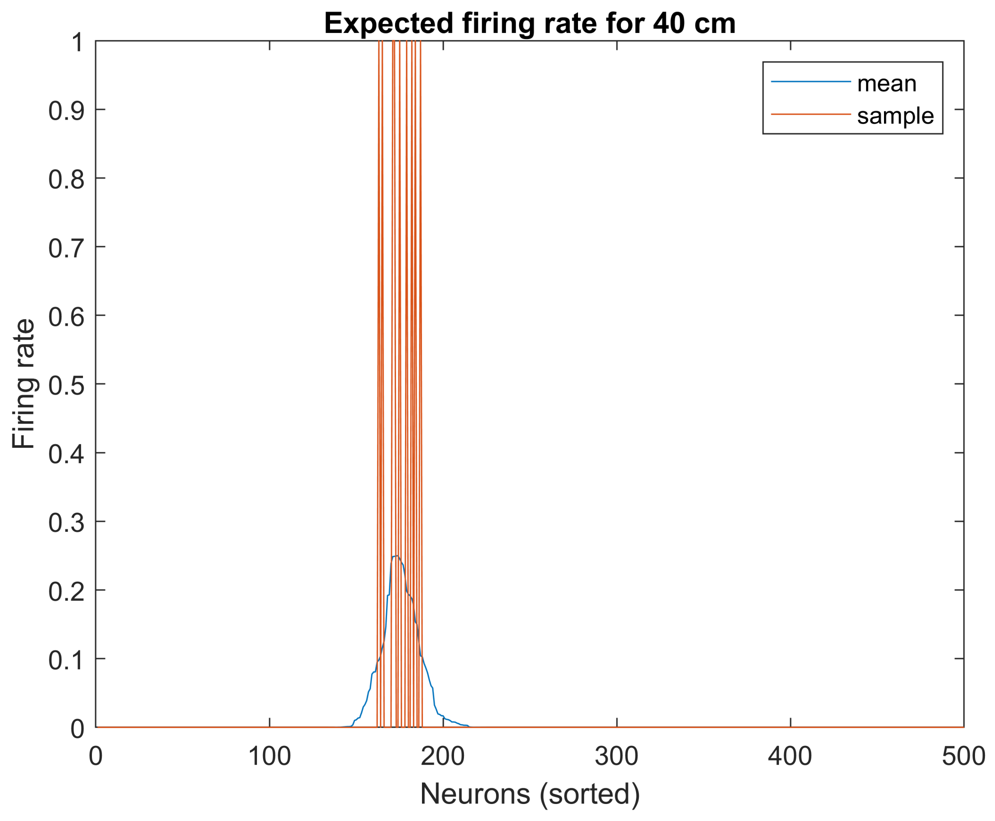
Define a 120cm track with uniformly centered 3cm std place fields, showing the Poisson firing rate per position:

```
n_cells = 500;
track_len = 120;
field_std = 3;

x = 0:0.1:track_len; % cm
m = sort(rand(n_cells, 1) * track_len);

R = g_field(x, m, field_std);
figure;
imagesc(x, 1:n_cells, R);
```

```
xlabel 'Position (cm)'
ylabel 'Neurons (sorted)'
h = colorbar;
h.Label.String = 'Firing rate';
title 'Firing rate raster plot'
```

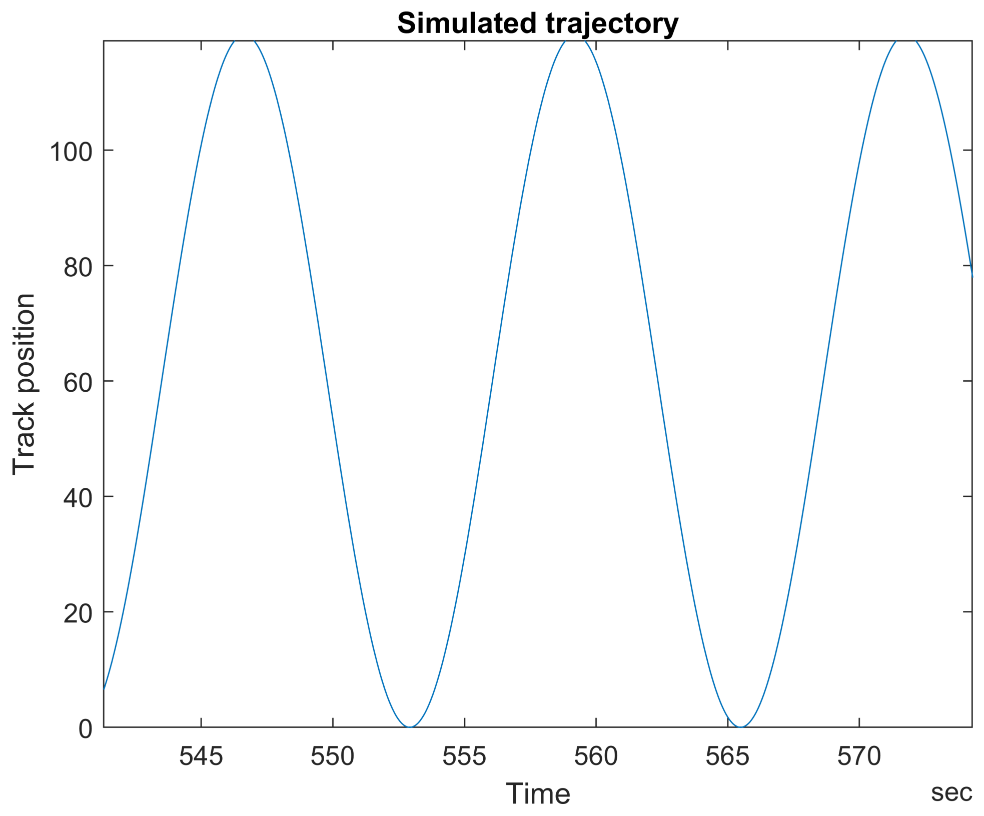


```
rate_func = @(x) g_field(x, m, field_std);
x_example = 40;
figure;
plot(rate_func(x_example));
xlabel 'Neurons (sorted)'
ylabel 'Firing rate'
title(sprintf('Expected firing rate for %d cm', x_example));

hold on;
response_sample = @(x) poissrnd(rate_func(x));
plot(response_sample(x_example));
legend mean sample
```



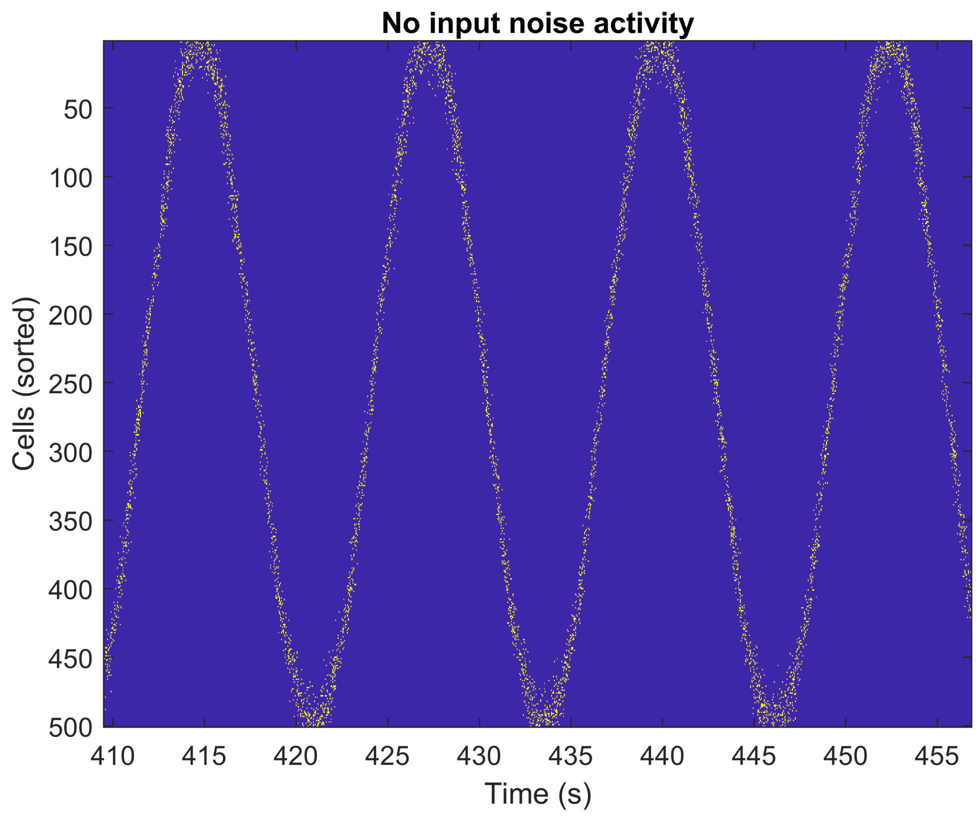
Simulate a trajectory oscillating back and forth, with a maximum velocity of 30cm/s. For now, assume directionally independent place fields:

```
t = seconds(seconds(0):seconds(1/20):hours(0.5)); %s
max_v = 30; %cm/s
x_traj = track_len .* sin(max_v./track_len .* t).^2;

figure;
plot(seconds(t), x_traj);
xlabel 'Time'
ylabel 'Track position'
title 'Simulated trajectory'

xlim([duration(0,9,1.3)...
      duration(0,9,34.3)])
ylim([0 119])
```

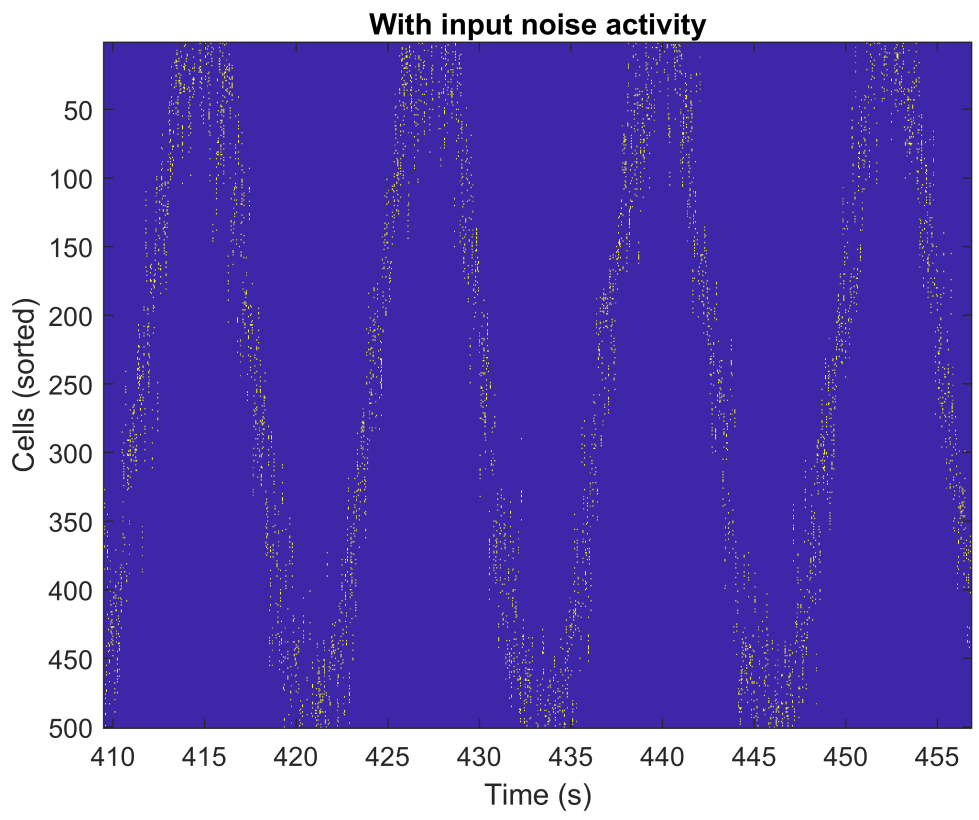


Generate the sample neural activity, first without noise, then with gaussian input noise of σ=10cm.

```
X_no_noise = response_sample(x_traj);

figure;
imagesc(t, 1:n_cells, X_no_noise, [0 1]);
xlabel 'Time (s)'
ylabel 'Cells (sorted)'
title 'No input noise activity'
xlim([409.5 456.9])
ylim([1 501])
```



```
noise_sigma = 10;
input_noise = noise_sigma*randn(size(x_traj));

X_noise = response_sample(x_traj + input_noise);
figure;
imagesc(t, 1:n_cells, X_noise, [0 1]);
xlabel 'Time (s)'
ylabel 'Cells (sorted)'
title 'With input noise activity'
xlim([409.5 456.9])
ylim([1 501])
```



Create a fake dateset:

```
tracesEvents.simulated_noise = X_noise.';
tracesEvents.simulated_no_noise = X_no_noise.';
tracesEvents.position = [x_traj(:), 0*x_traj(:)];

savefile = 'Mouse-2099-20210604_999999-linear-track-TracesAndEvents.mat';
save(savefile, 'tracesEvents');
```

Create a DecodeTensor object from it:

```
d_sim_noise = DecodeTensor({savefile, 'Fake'}, 'simulated_noise');
```

```
Total trials: 286     Throwing away 0     keeping 286
```

```
d_sim_no_noise = DecodeTensor({savefile, 'Fake'}, 'simulated_no_noise');
```

```
Total trials: 286     Throwing away 0     keeping 286
```

Test out decoding performances:

```
tic;
mean_err_real_noise = d_sim_noise.basic_decode(false, [], [])
```

```
mean_err_real_noise = 4.5392
```

```
mean_err_shuf_noise = d_sim_noise.basic_decode(true, [], [])
```

```
mean_err_shuf_noise = 1.9155
```

```

mean_err_real_no_noise = d_sim_no_noise.basic_decode(false, [], [])
```

```
mean_err_real_no_noise = 0.0052
```

```
mean_err_shuf_no_noise = d_sim_no_noise.basic_decode(true, [], [])
```

```
mean_err_shuf_no_noise = 0.0072
```

```
toc
```

```
Elapsed time is 234.913099 seconds.
```

```
t_ = tic;
opt = DecodeTensor.default_opt;
opt.neural_data_type = 'simulated_noise';
DecodeTensor.decode_series(savefile, 'Fake', opt);
toc(t_)
```

```
Elapsed time is 3266.103285 seconds.
```

```
db_name = 'sim_decoding.db';
DecodeTensor.aggregate_results('db_file', db_name);
```

```
conn = sqlite(db_name);
samp_size = 1;
sess = '999999';
mouse_name = 'Fake';
[n_sizes, imse] = PanelGenerator.db_imse_reader(conn,...
    'unshuffled', {{mouse_name, sess}}, samp_size);
[n_sizes_s, imse_s] = PanelGenerator.db_imse_reader(conn,...
    'shuffled', {{mouse_name, sess}}, samp_size);
```

Looking at the fitted curve parameters, they are consistent with what we found on experimental data (*N*~100 (real) ~1000 (shuffled)):

The following are the fitted curve's I_0 and *N* parameters for the unshuffled simulated data:

```
[fit_res, gof] = createFit_infoSaturation(n_sizes{1}(:), imse{1}(:))
```

```
fit_res = 
     General model:
     fit_res(x) = I_0*x/(1+x/N)
     Coefficients (with 95% confidence bounds):
       I_0 =   0.0002928  (0.0002547, 0.0003308)
       N =         104  (85.56, 122.5)
gof = struct with fields:
           sse: 9.5114e-05
       rsquare: 0.9592
```

```
          dfe: 49
   adjrsquare: 0.9583
         rmse: 0.0014
```

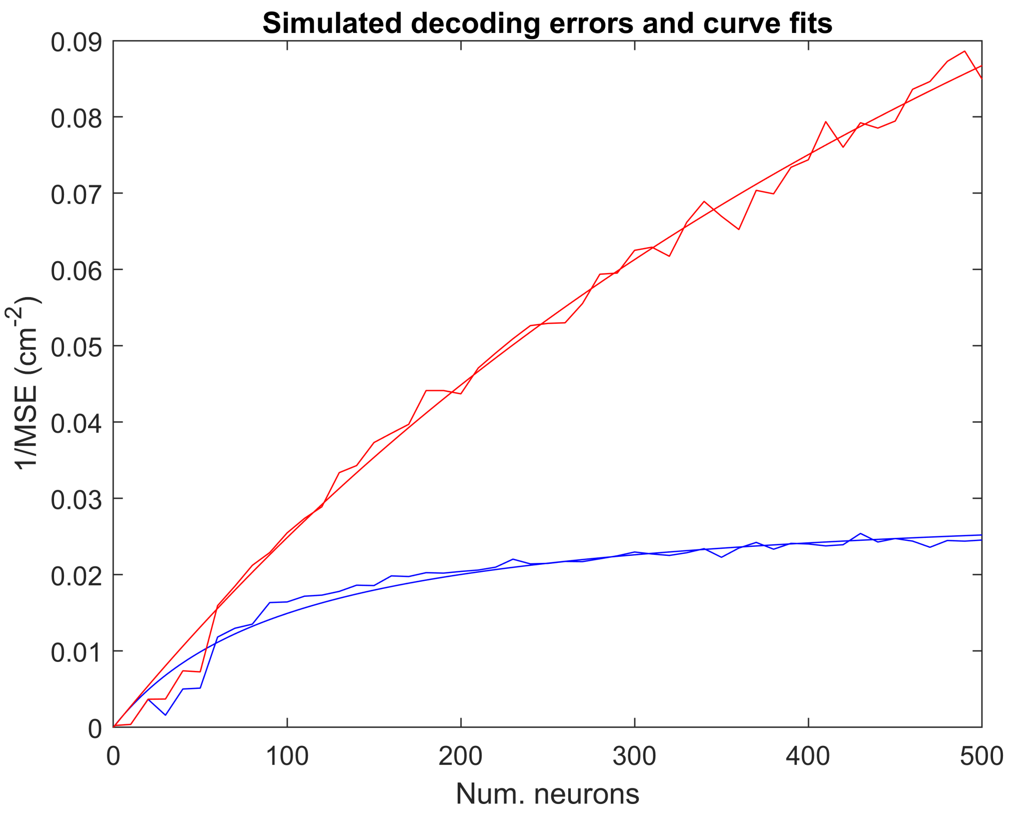
And for the shuffled simulated data:

```
[fit_res_sh, gof_sh] = createFit_infoSaturation(n_sizes_s{1}(:), imse_s{1}(:))
```

```
fit_res_sh = 
     General model:
     fit_res_sh(x) = I_0*x/(1+x/N)
     Coefficients (with 95% confidence bounds):
       I_0 =   0.0002788  (0.0002664, 0.0002913)
       N =       823.2  (703.4, 943)
gof_sh = struct with fields:
          sse: 1.8781e-04
      rsquare: 0.9944
          dfe: 49
   adjrsquare: 0.9943
         rmse: 0.0020
```

The inverse mean squared error curves with the curve fits are as follows:

```
figure;
plot(n_sizes{1}, imse{1}, 'b'); hold on;
plot(fit_res, 'b');
plot(n_sizes_s{1}, imse_s{1}, 'r');
plot(fit_res_sh, 'r');
xlabel 'Num. neurons'
ylabel '1/MSE (cm^{-2})'
legend off
title 'Simulated decoding errors and curve fits'
```


Simulated decoding errors and curve fits
1/MSE (cm^{-2})
Num. neurons
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
100
200
300
400
500