UNIVERSITY OF CAMBRIDGE

High-dimensional prior truncation

*On-going work / work in progress

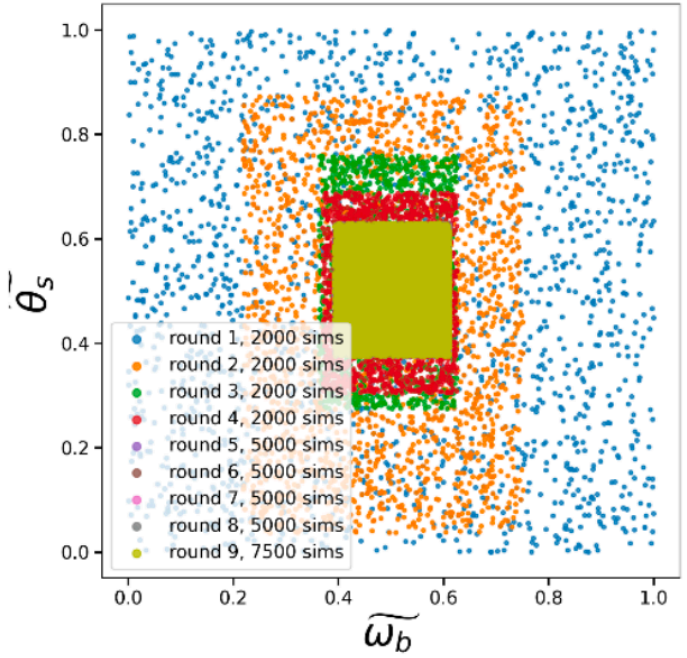
Kilian Hikaru Scheutwinkel, 3rd year Physics Ph.D, University of Cambridge

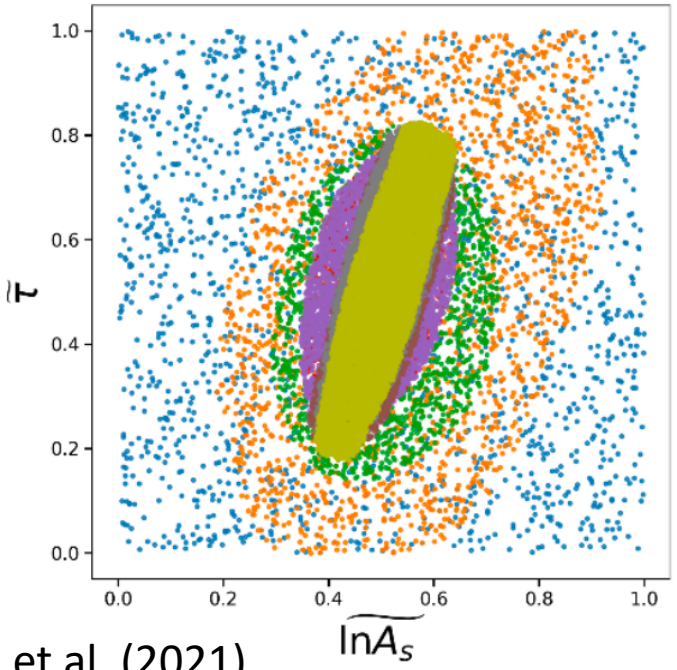
So far….

We saw what the Swyft algorithm does;
an implementation of (TM)NRE

We also saw the nested sampling algorithm just now

For the continuation of this talk: we are using TMNRE → NRE





Cole et al. (2021)

The idea of nested sampling…

NS solves:

$$Z = \int_{\Theta} L(\theta)\pi(\theta)d\theta = \int_0^1 \mathcal{L}dX,$$ by creating prior mass shells $dX = \pi(\theta)d\theta$

$dX :=$ fraction of prior mass $L > L^*$

These shells look familiar?

Skilling (2006)

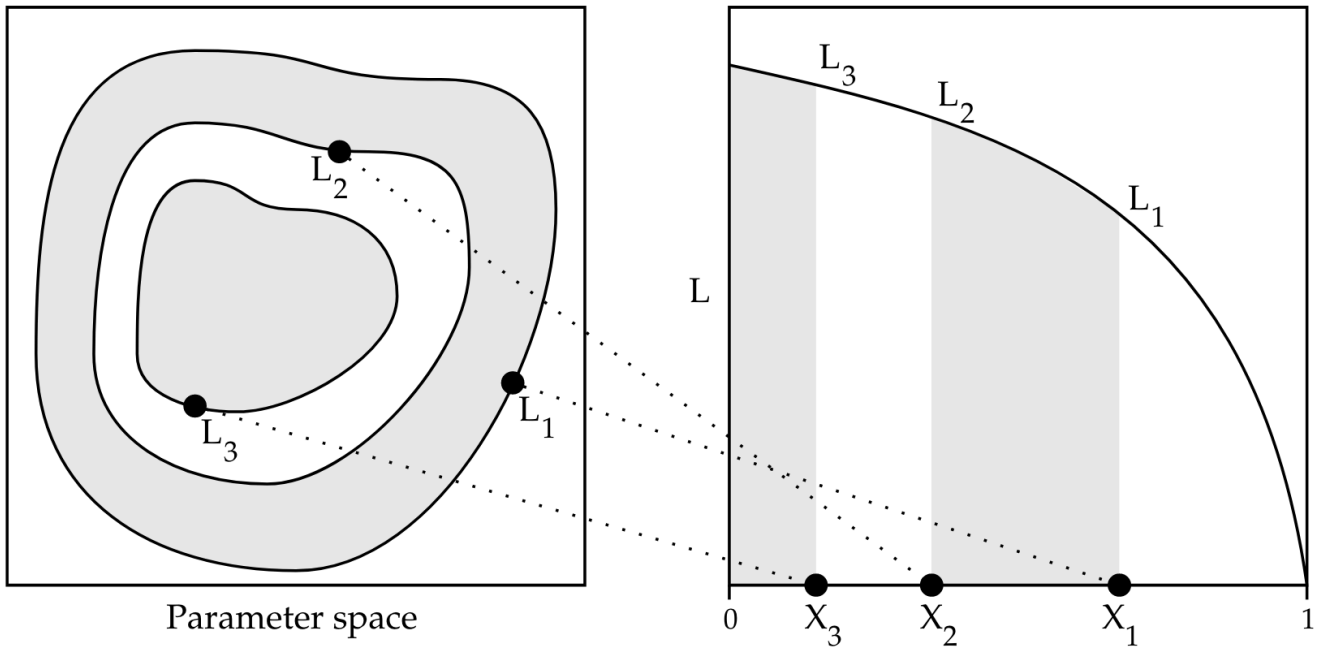


Figure 3: Nested likelihood contours are sorted to enclosed prior mass X.

NS vs TMNRE

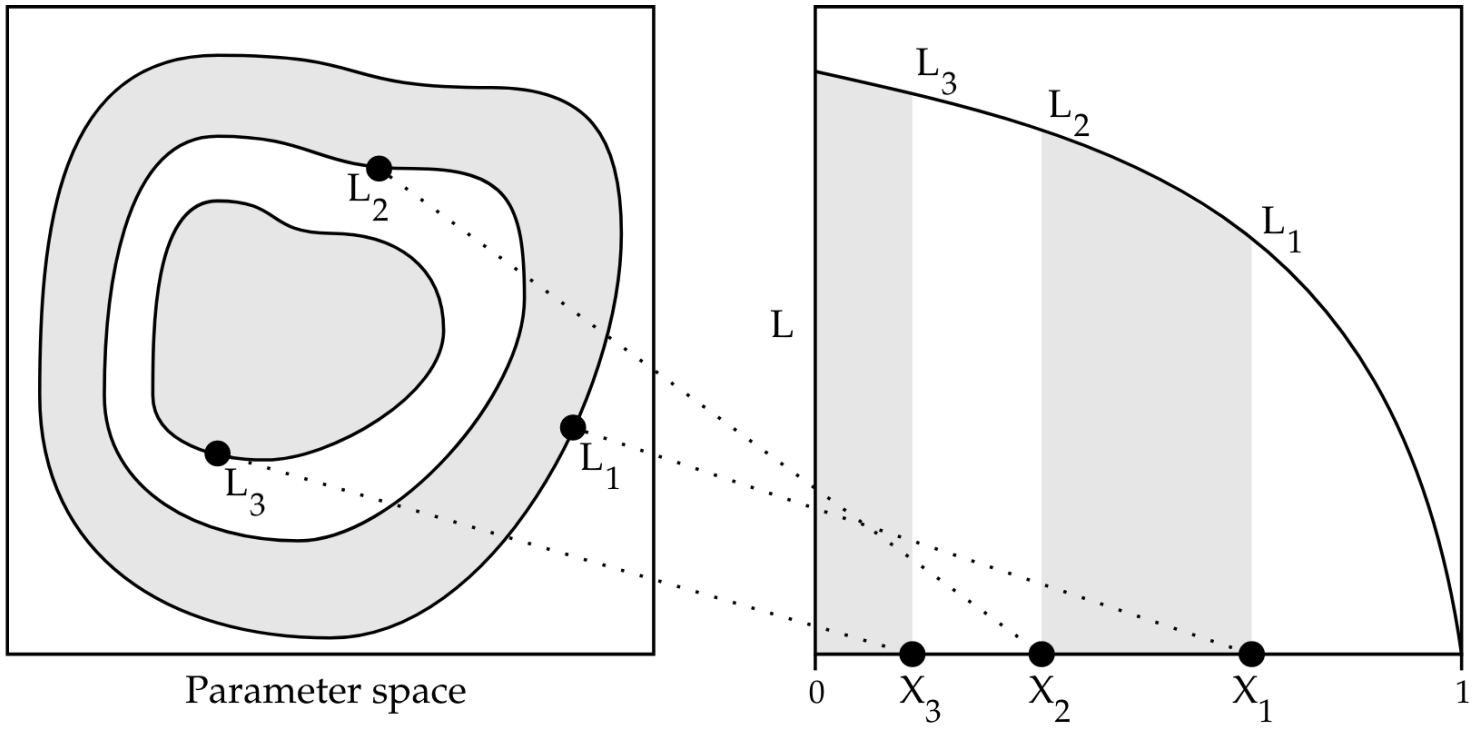
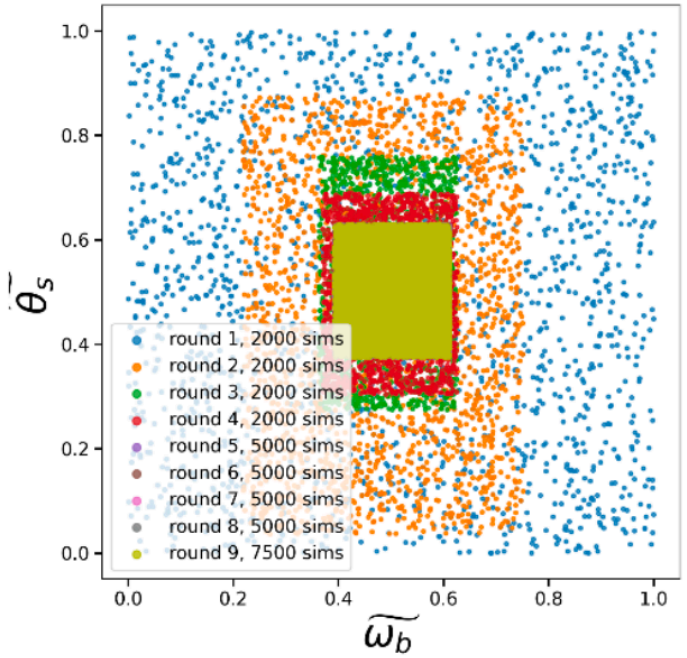
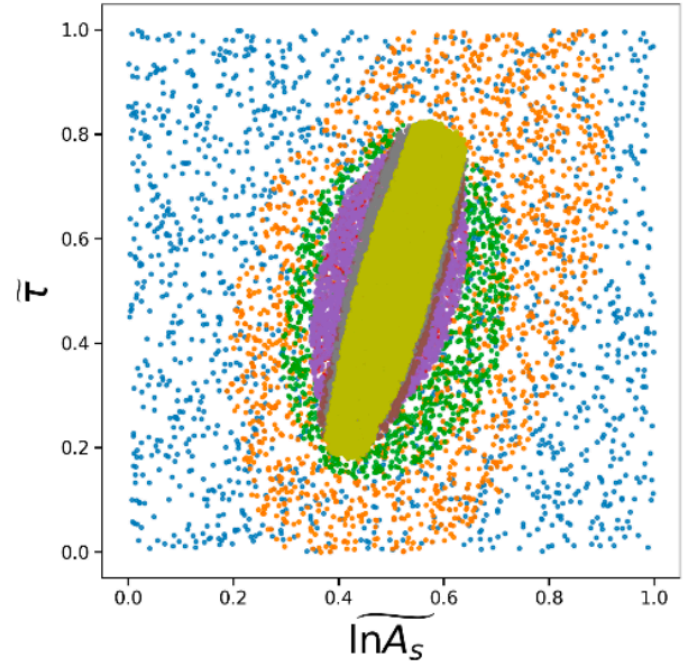


Figure 3: Nested likelihood contours are sorted to enclosed prior mass X.





Some computational fun…

What happens if we plug into the NS computation an NRE?

$$\int L(\theta)\pi(\theta)d\theta \rightarrow \int \underbrace{\frac{L(\theta)}{Z}}_{=r(x,\theta)} \pi(\theta)d\theta = \int p(\theta\,|\,x)d\theta = 1$$

It is an integration of the posterior distribution!

In practice…

NS finds proposal samples from prior subject to $L > L^*$

This dynamic likelihood constraint define the contours (shells) within the prior

In standard NS: $L_i^* = \min(L_{1:N,i})$

```
while logEvidenceIncrease > np.log(stop_criterion):
    # identifying lowest likelihood point
    minlogLike = logLikelihoods.min()
    # find new sample satisfying likelihood constraint
    proposal_sample = sampler.sample(livepoints=livepoints.copy(), minlogLike=minlogLike,
                                     livelikes=logLikelihoods)
```

Could also do: $L_i^* = \text{median}(L_{1:N,i})$

```
# find new sample satisfying likelihood constraint
medianlogLike = np.median(logLikelihoods)
for it in range(iter):
    # find new sample satisfying likelihood constraint
    proposal_sample = sampler.sample(livepoints=livepoints.copy(), minlogLike=medianlogLike,
                                     livelikes=logLikelihoods)
```

or other conditions to define “truncation”

Now stick all concepts together

NSNRE algorithm procedure:

1) Use a simulator to generate data $x \sim p(x \mid z)p(z)$
2) Train an NRE with swyft for an observation x_0 using simulator

3) Use NS computation to find samples $L_{i+1} > L_i^*$ where $L_i^* = \text{median}(L_{1:N,i})$
4) Train new NRE on newly found samples L_{i+1}
5) Repeat $i = 1...M$ times to find samples with higher likelihoods in subsequent rounds

Key point: The NRE can evaluate the “likelihood” (weights) of a sample

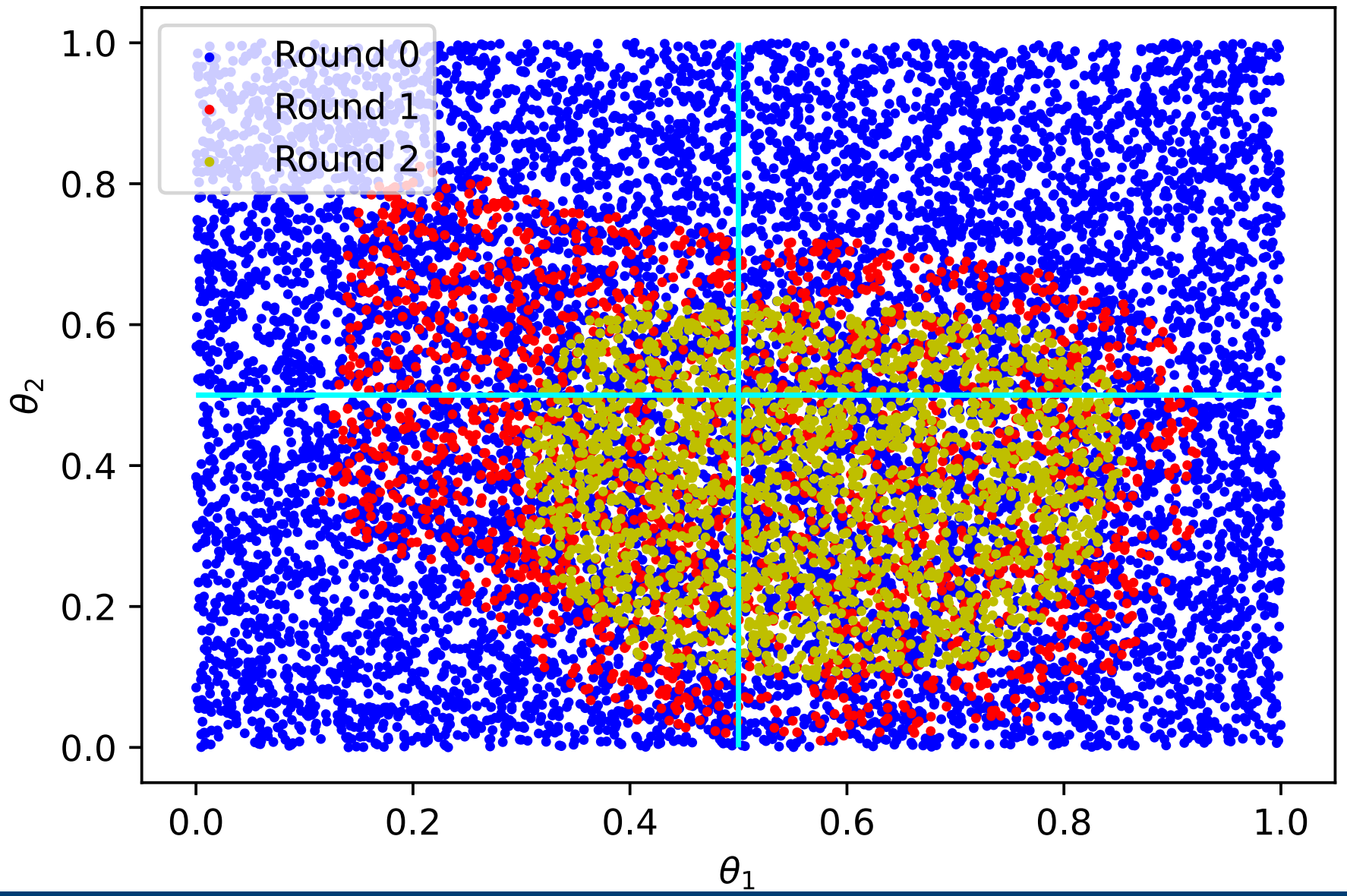
Toy Problem

```
class Simulator(swyft.Simulator):
    def __init__(self, nParam, F, cov):
        super().__init__()
        self.nParam = nParam
        self.F = F
        self.cov = cov
        self.transform_samples = swyft.to_numpy32

    def build(self, graph):
        means = graph.node('means', lambda: theta_prior.rvs(size=self.nParam))
        x = graph.node('x',
                       lambda means: multivariate_normal.rvs(self.F @ means, self.cov),
                       means)
```

Here: Estimate 2-d mean θ of multivariate Normal with known covariance

Swyft network

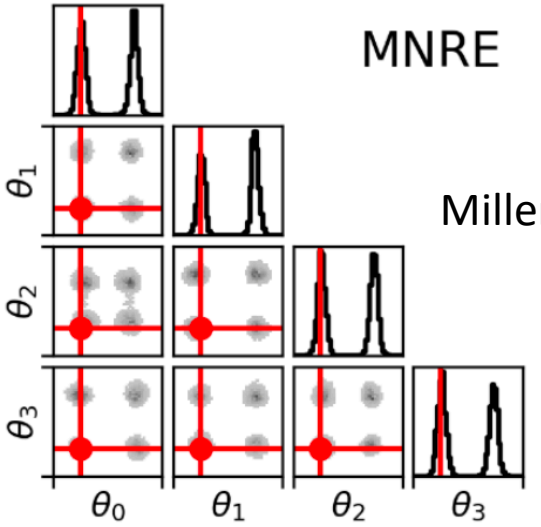
```
# initialize swyft network
class Network(swyft.SwyftModule):
    def __init__(self):
        super().__init__()
        marginals = (tuple(x for x in range(nParam)),)
        # self.logratios1 = swyft.LogRatioEstimator_1dim(num_features = 1, num_params = 3, varnames = 'z')
        self.logratios2 = swyft.LogRatioEstimator_Ndim(num_features=nData, marginals=marginals,
                                                       varnames='means',
                                                       hidden_features=32)

    def forward(self, A, B):
        # logratios1 = self.logratios1(A['x'], B['z'])
        logratios2 = self.logratios2(A['x'], B['means'])
        return logratios2
```


Log-weights contours using NS rounds, $\mathcal{L} > \mathcal{L}_{\text{median}}$
Round 0
Round 1
Round 2
1.0
0.8
0.6
0.4
0.2
0.0
θ_2
0.0
0.2
0.4
0.6
0.8
1.0
θ_1

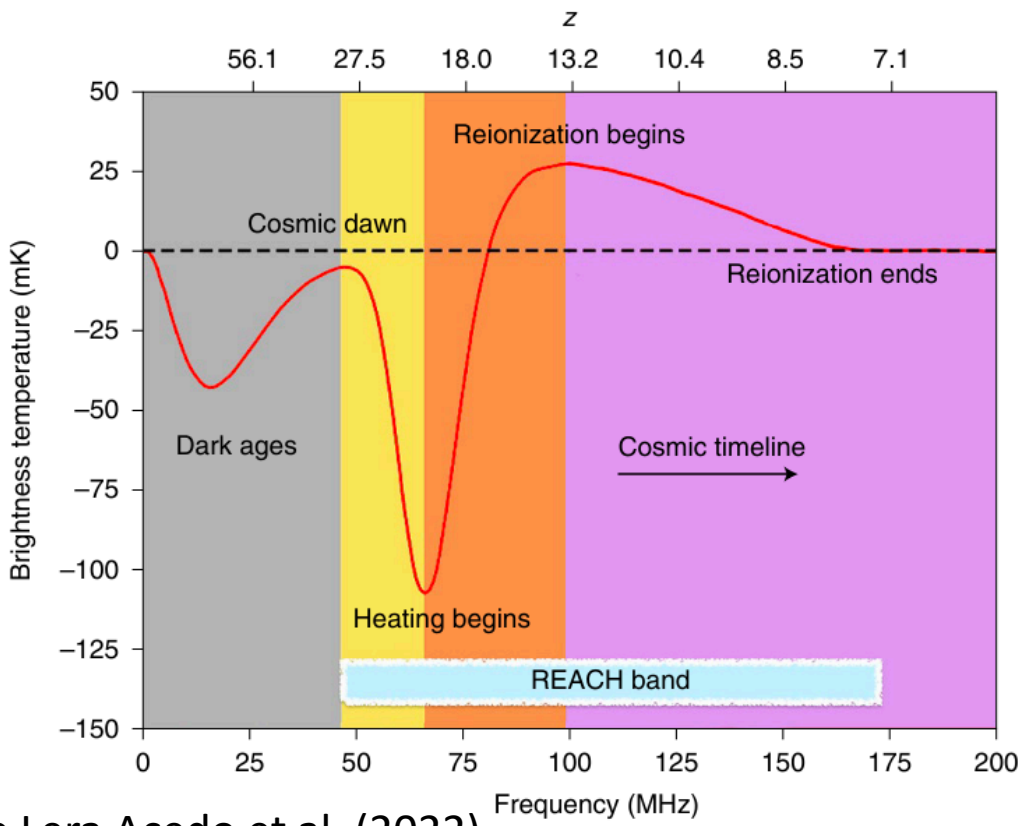
Future plans

1) Implement this method for NSNRE →NSMNRE
   - Why? Leveraging the estimation of 2d marginals instead of full joint inference

2) Use this method for 21-cm Cosmology e.g. for REACH (The Radio Experiment for the Analysis of Cosmic Hydrogen)
   - Where? The global 21-cm signal shape is uncertain
   - but there exists emulators (simulators) trained on cosmological parameters



Miller et al. (2021)



de Lera Acedo et al. (2022)

Research supported by

The Alan Turing Institute