ELFI

Engine for Likelihood-Free Inference

Henri Pesonen

ELFI Development Team

Aalto University
School of Science

UNIVERSITY OF HELSINKI

THE UNIVERSITY of EDINBURGH

UiO : University of Oslo

elfi.readthedocs.io

@henri_pesonen

Engine for Likelihood-free Inference

- Statistical software package for LFI
  - Written in Python (≥ 3.5)

```
pip install elfi
```

- Aim is to make it easy for
  - practitioners to apply LFI
  - developers to contribute to the package

ELFI Highlights (2018-19)

- Pneumococcal quorum sensing drives an asymmetric owner–intruder competitive strategy during carriage via the competence regulon
  - P. Shen, J. A. Lees, G. C. W. Bee, S. P. Brown and J. N. Weiser, Nature Microbiology vol 4, pp198–208, 2019
- Machine learning accelerated likelihood-free event reconstruction in dark matter direct detection
  - U. Simola, B. Pelssers, D. Barge, J. Conrad and J. Corander, Journal of Instrumentation, 14:3, 2019
- Resolving outbreak dynamics using approximate Bayesian computation for stochastic birth-death models
  - J. Lintusaari, P. Blomstedt, T. Sivula, M. U. Gutmann, S. Kaski, J. Corander, Wellcome Open Res, 4:14, 2019
- Adversarial Variational Optimization of Non-Differentiable Simulators
  - G. Louppe, J. Hermans and K. Cranmer, arXiv, Oct 2018

Main features of ELFI

- Implementations of LFI/ABC algorithms
- Easy syntax for defining model structure called ELFI-graph
- Storage and re-use of results
- Reporting and diagnostic tools
- Designed to be extensible
- Automatic parallellization
  - `ipyparallel, multiprocessing,...`

For methodologists & developers

- Modular design of the library
- Well-defined API for implementing own modules
- Maintenance of the library (Aalto PML)
- Growing user community
- Built-in simulators, ELFI graphs and data for method development and assessment

For methodologists & developers

- Source is hosted in Github
- Code development uses the continuous integration practices
  - Code review
  - Automated tests

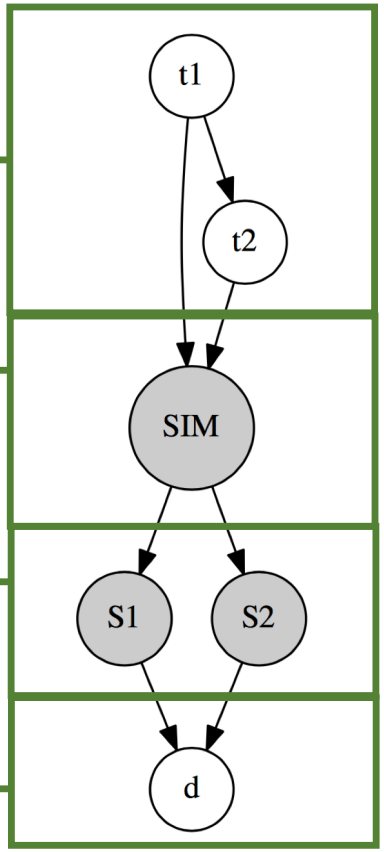
ELFI graph

Definition of the model is via ELFI-graph

Components

- Can be implemented in wide variety of languages

- Priors
- Simulators
- Summaries
- Distances

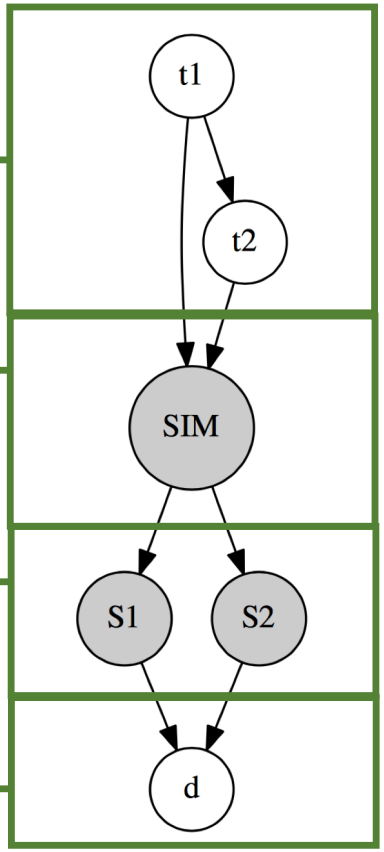


Definition of the model is via ELFI-graph

Components

- Can be implemented in wide variety of languages

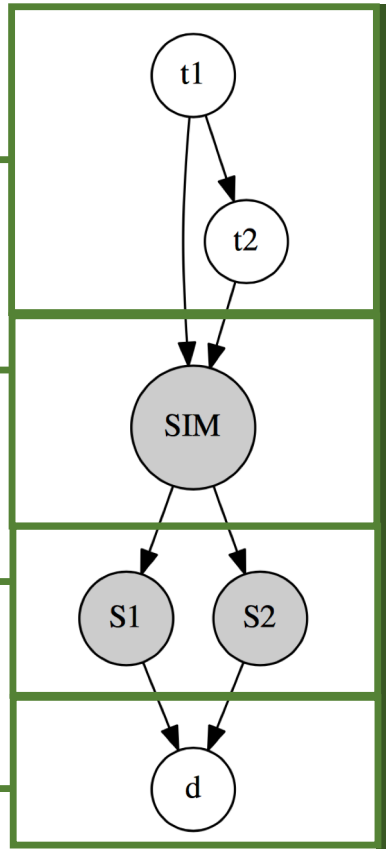
- Priors
- Simulators
- Summaries
- Distances



DAG structure facilitates detailed hierarchies

ELFI syntax

- Priors
- Simulators
- Summaries
- Distances




```
t1 = elfi.Prior('uniform',-2,4)
t2 = elfi.Prior('normal',t1,5)

SIM = elfi.Simulator(simulator,t1,t2,observed=y)

S1 = elfi.Summary(summary,SIM)
S2 = elfi.Summary(summary,SIM,2)

d = elfi.Distance('euclidean',S1,S2)
```

Priors

- `scipy.stats` distributions are supported

```
t1 = elfi.Prior('uniform',-2,4)
t2 = elfi.Prior('normal',t1,5)
```

- Custom priors can be defined
  - Define API similar to `scipy.stats`-distributions
  - Implement at least `rvs` method
  - Necessary keywords `random_state` and `size`

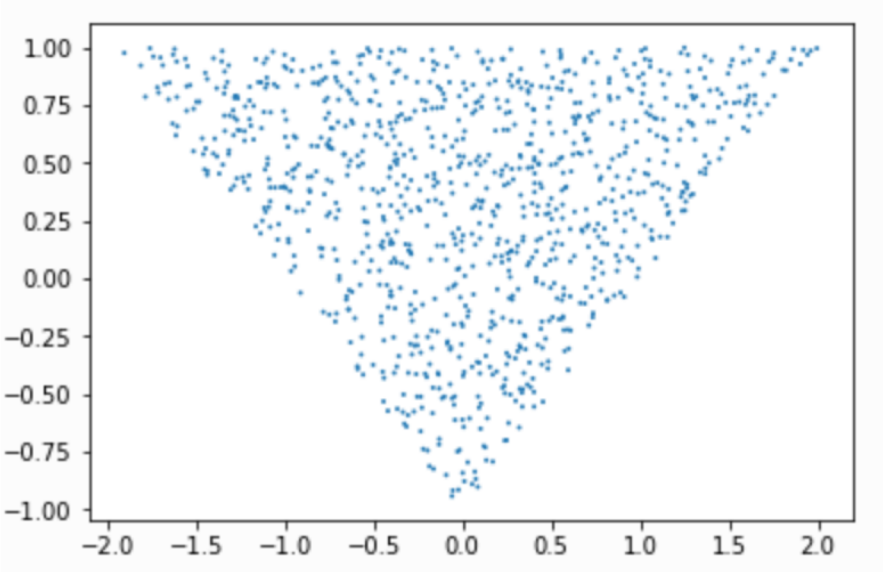
Custom priors

```
class CustomPrior_t1(elfi.Distribution):
    def rvs(b, size=1, random_state=None):
        u = scipy.stats.uniform.rvs(loc=0, scale=1,
                                    size=size,
                                    random_state=random_state)
        t1 = np.where(u<0.5, np.sqrt(2.*u)*b-b, -np.sqrt(2.*(1.-u))*b+b)
        return t1


class CustomPrior_t2(elfi.Distribution):
    def rvs(t1, a, size=1, random_state=None):
        locs = np.maximum(-a-t1, t1-a)
        scales = a - locs
        t2 = scipy.stats.uniform.rvs(loc=locs, scale=scales,
                                     size=size, random_state=random_state)
        return t2
```

Custom priors

```
t1_1000 = CustomPrior_t1.rvs(2, 1000)
t2_1000 = CustomPrior_t2.rvs(t1_1000, 1, 1000)
plt.scatter(t1_1000, t2_1000, s=4, edgecolor='none')
```



Simulators

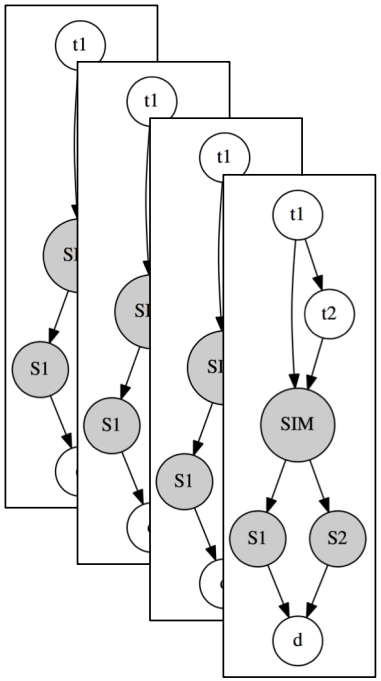
- ELFI requires some structure from the simulator
  - `batch_size`
  - `random_state`

```
def simulator(t1, t2, n_obs=100, batch_size=1,
              random_state=None):
    t1 = np.asanyarray(t1).reshape((-1, 1))
    t2 = np.asanyarray(t2).reshape((-1, 1))
    random_state = random_state or np.random
    w = random_state.randn(batch_size, n_obs+2)

    x = w[:, 2:] + t1*w[:, 1:-1] + t2*w[:, :-2]
    return x
```

batch_size

- Number of evaluations of a node before moving on to the next



batch_size

- Number of evaluations of a node before moving on to the next
- Essential for efficient parallellization
- Operations are vectorized
  - ELFI simulates a batch of data simultaneously
  - Built-in `elfi.tools.vectorize` for automatic vectorization

```
def simulator(t1, t2, n_obs=100, batch_size=1,
              random_state=None):
    t1 = np.asanyarray(t1).reshape((-1, 1))
    t2 = np.asanyarray(t2).reshape((-1, 1))
    random_state = random_state or np.random
    w = random_state.randn(batch_size, n_obs+2)

    x = w[:, 2:] + t1*w[:, 1:-1] + t2*w[:, :-2]
    return x
```

`random_state`

- `numpy.RandomState` object
- Used for ensuring consistent results
- Handles random number generation in parallel settings

```
def simulator(t1, t2, n_obs=100, batch_size=1,
              random_state=None):
    t1 = np.asanyarray(t1).reshape((-1, 1))
    t2 = np.asanyarray(t2).reshape((-1, 1))
    random_state = random_state or np.random
    w = random_state.randn(batch_size, n_obs+2)

    x = w[:, 2:] + t1*w[:, 1:-1] + t2*w[:, :-2]
    return x
```

Summary statistics

- Input: N batches of data
- Output: `numpy.array` of shape (N,M1,...,Mn)
- Recommended to use vectorized calculations if possible

Distance

- ELFI can use common distances from `scipy.spatial.distance`
- Custom distance/discrepancy functions
  - `elfi.Distance` and `elfi.Discrepancy`

Alternatively - define an ElfiModel object

```
m = elfi.ElfiModel()
elfi.Prior(CustomPrior1, 2, model=m, name='t1')
elfi.Prior(CustomPrior2, m['t1'], 1, name='t2')
elfi.Simulator(simulator, m['t1'], m['t2'], observed=y, name='MA2')
elfi.Summary(autocov, m['MA2'], name='S1')
elfi.Summary(autocov, m['MA2'], 2, name='S2')
elfi.Distance('euclidean', m['S1'], m['S2'], name='d')
```

Inference

Current inference methods in ELFI

ABC methods are initialized with the final node (usually distance) (or with `ElfiModel` and the name of the final node, e.g. `m['d']`)

- Rejection-ABC

```
elfi.Rejection(d, batch_size=N)
```

- SMC-ABC

```
elfi.SMC(d, batch_size=N)
```

- Bayesian Optimization for Likelihood-free Inference (BOLFI)

```
elfi.BOLFI(d, batch_size=N, …)
```

Rejection-ABC

- Simplest ABC algorithm
  - Generate data from the ELFI graph & reject samples too far away from observed data
  - By default, rejection-ABC works in `quantile`-mode
    - e.g. `quantile=0.1` of samples with the smallest distances are accepted
  - Run rejection-ABC via

```
rej = elfi.Rejection(d, batch_size=N)
result = rej.sample(M, quantile = 0.01)
```

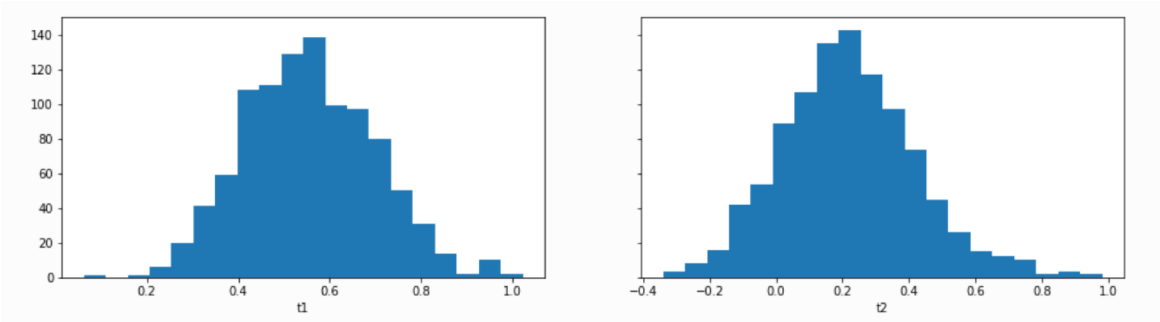
    - Simulator is run `M/quantile` times and `M` samples are kept

Rejection-ABC

- `sample` returns a `Sample` object
  - Attribute samples contain an `OrderedDict` of the posterior numpy arrays for all the model parameters (`elfi.Priors`)
  - e.g `t1_mean = result.samples['t1'].mean()`

  - Visualization

```
result.plot_marginals()
```

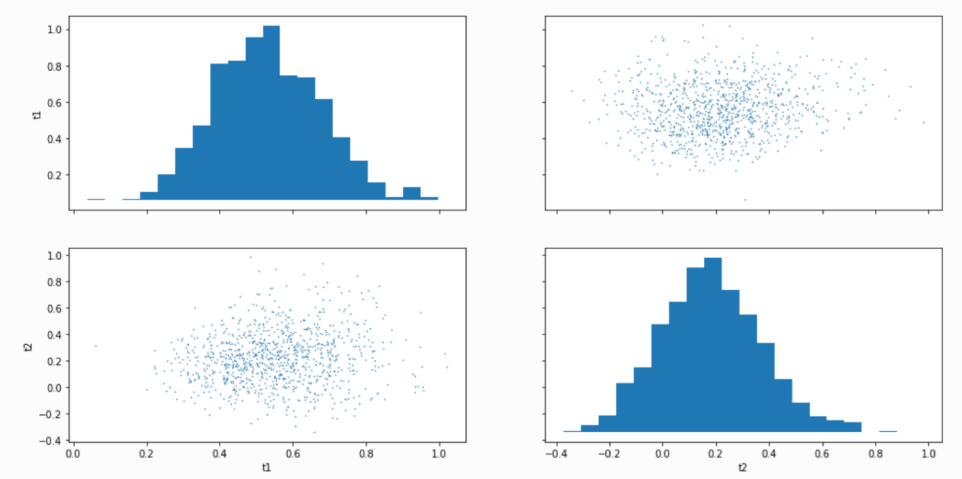


Rejection-ABC

- `sample` returns a `Sample` object
  - Attribute samples contain an `OrderedDict` of the posterior numpy arrays for all the model parameters (`elfi.Priors`)
  - e.g `t1_mean = result.samples['t1'].mean()`

  - Visualization

```
result.plot_pairs()
```



SMC-ABC

- Similar to rejection ABC
- Main difference is `schedule`-parameters

```
smc = elfi.SMC(d, batch_size=10000)
schedule = [0.7, 0.2, 0.05]
result_smc = smc.sample(N,schedule)
```

Bayesian Optimization for Likelihood-free Inference

- BOLFI framework is likely to provide useful in such situation
  - Complicated and computationally heavy simulators
    - Unable to run it for millions of times
  - Statistical model (GP) is created for discrepancy
    - Minimun is inferred using Bayesian optimization

Bayesian Optimization for Likelihood-free Inference

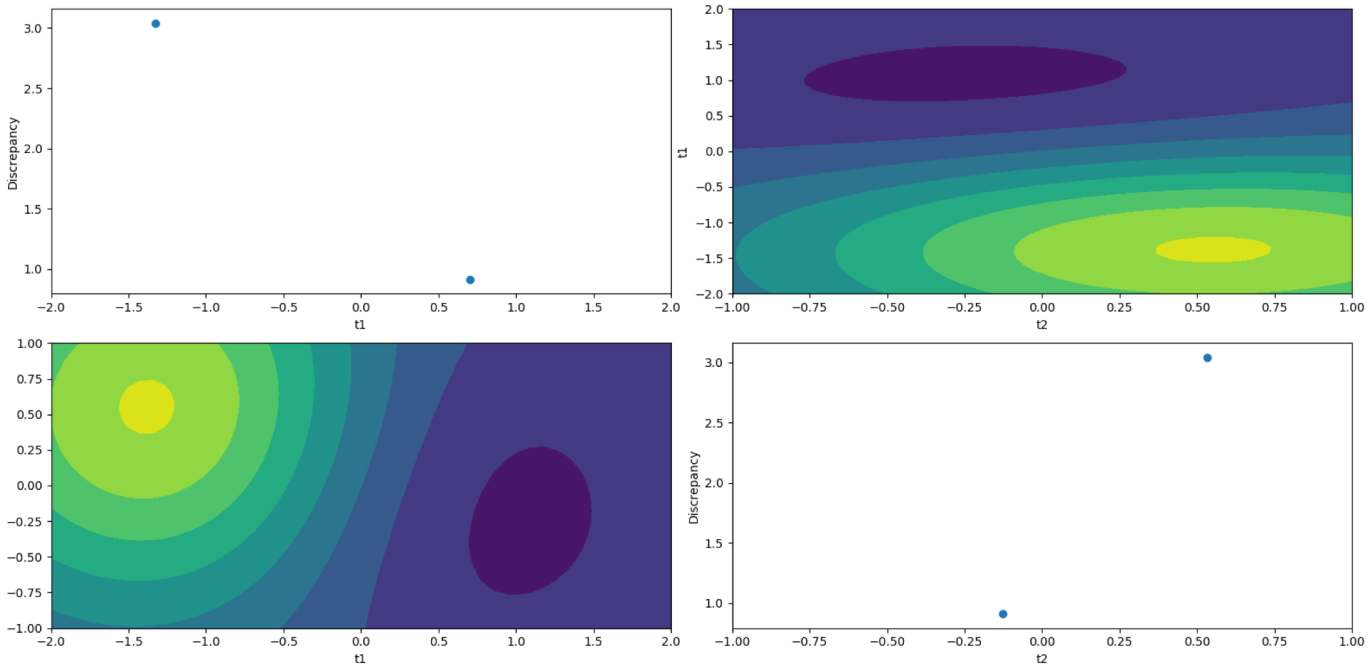
- More advanced inference method
  - Interface is also a bit more involved
  - Use the same ELFI graph
  - `target_model=GPyRegression`
    - Default `acquisition_method=LCBSC`
    - Defaults are often ok
      - Provide each parameter `bounds` as a dictionary
      - Provide `acq_noise_var` for acquisition function

Initial evidence from the prior

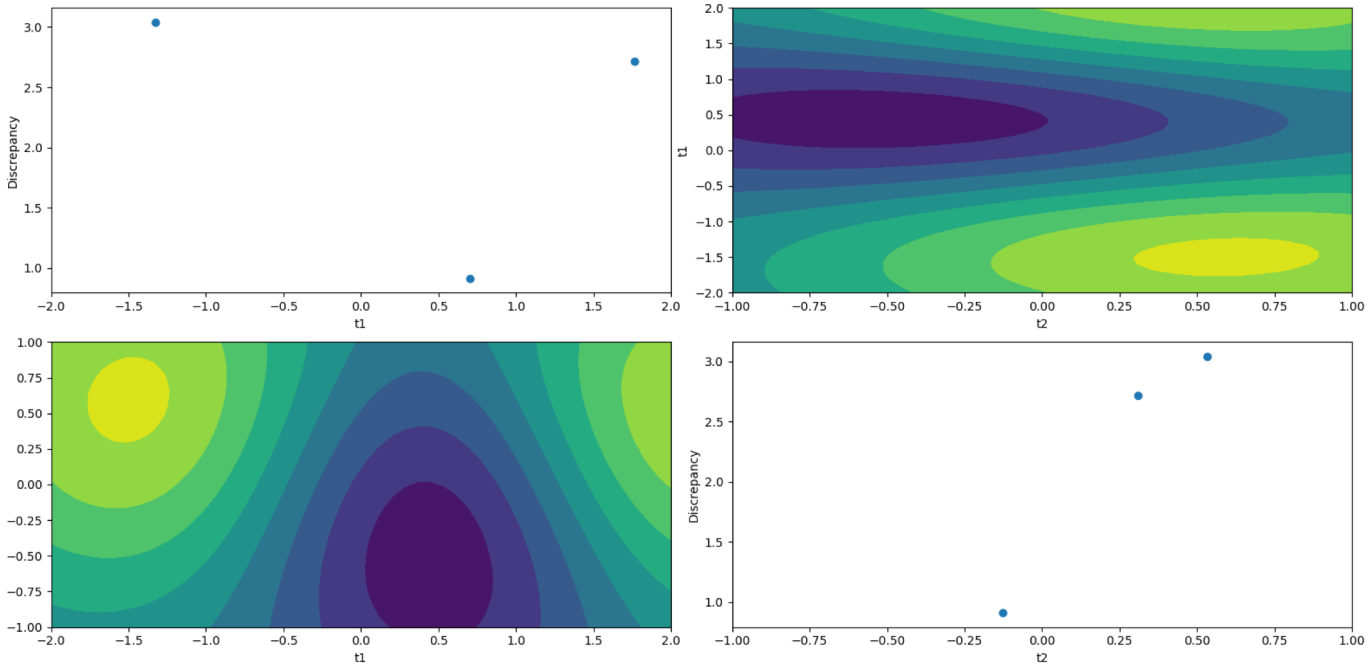
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
```

```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
```

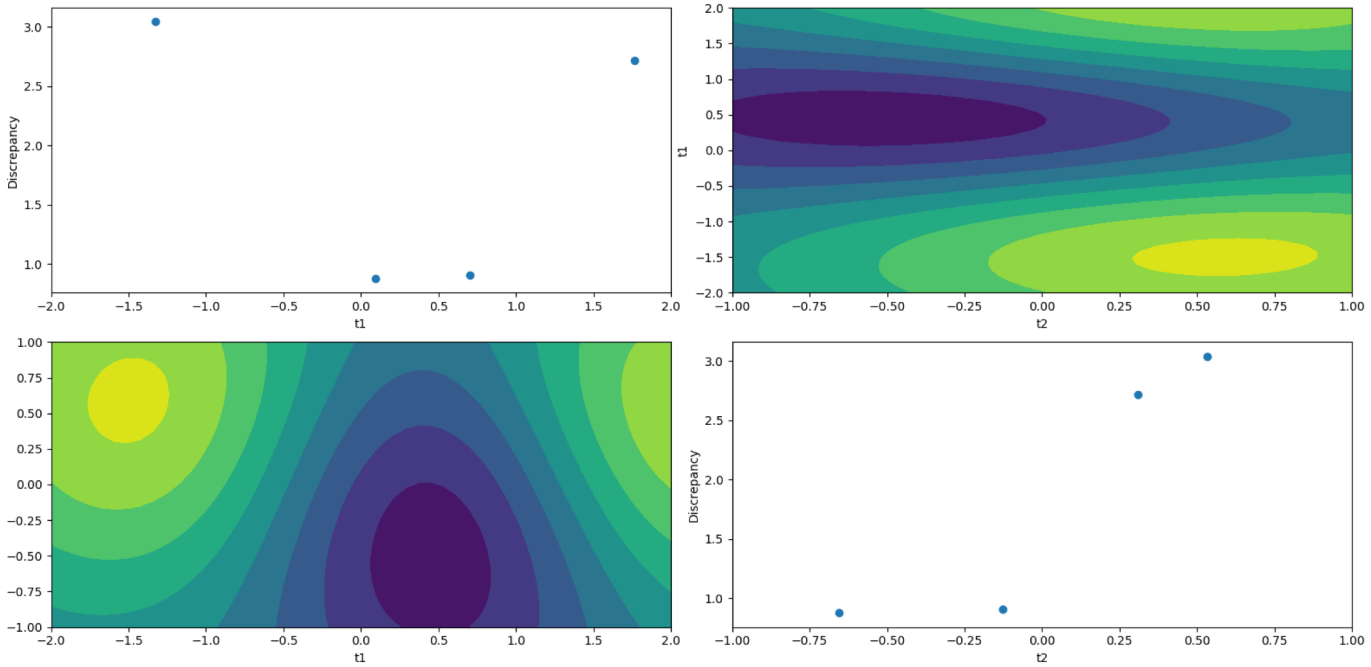
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 2)
```



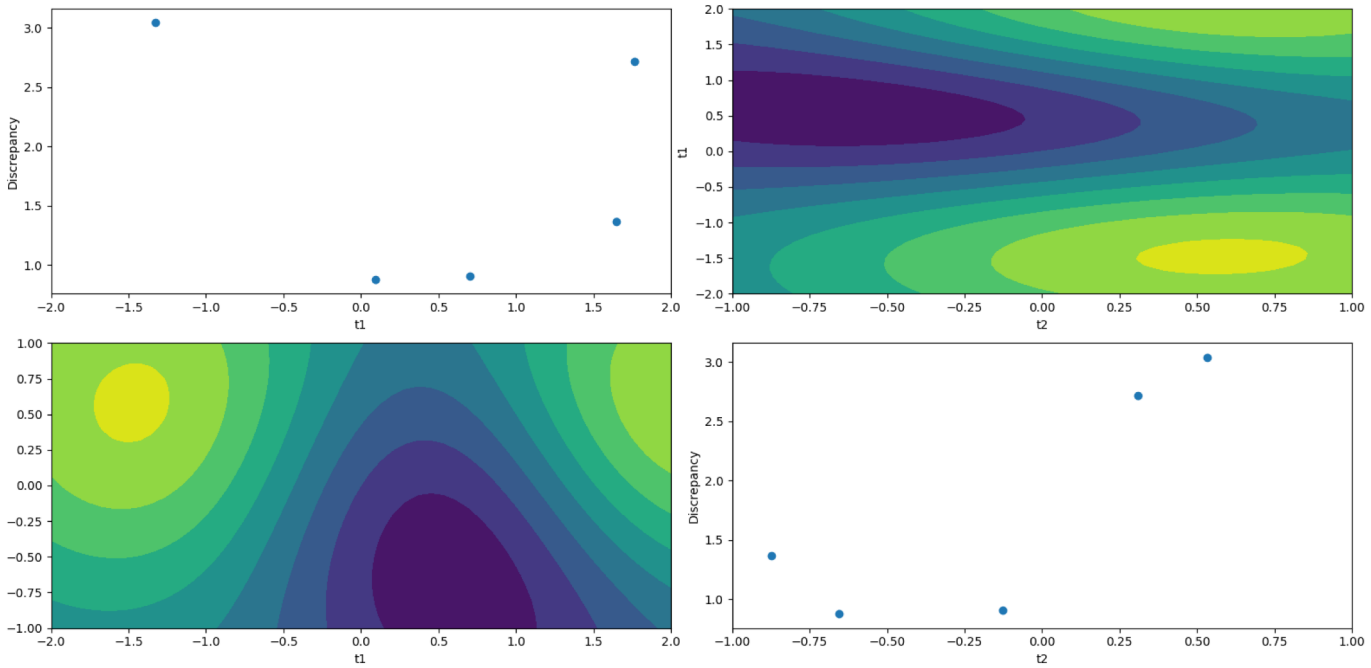
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 3)
```



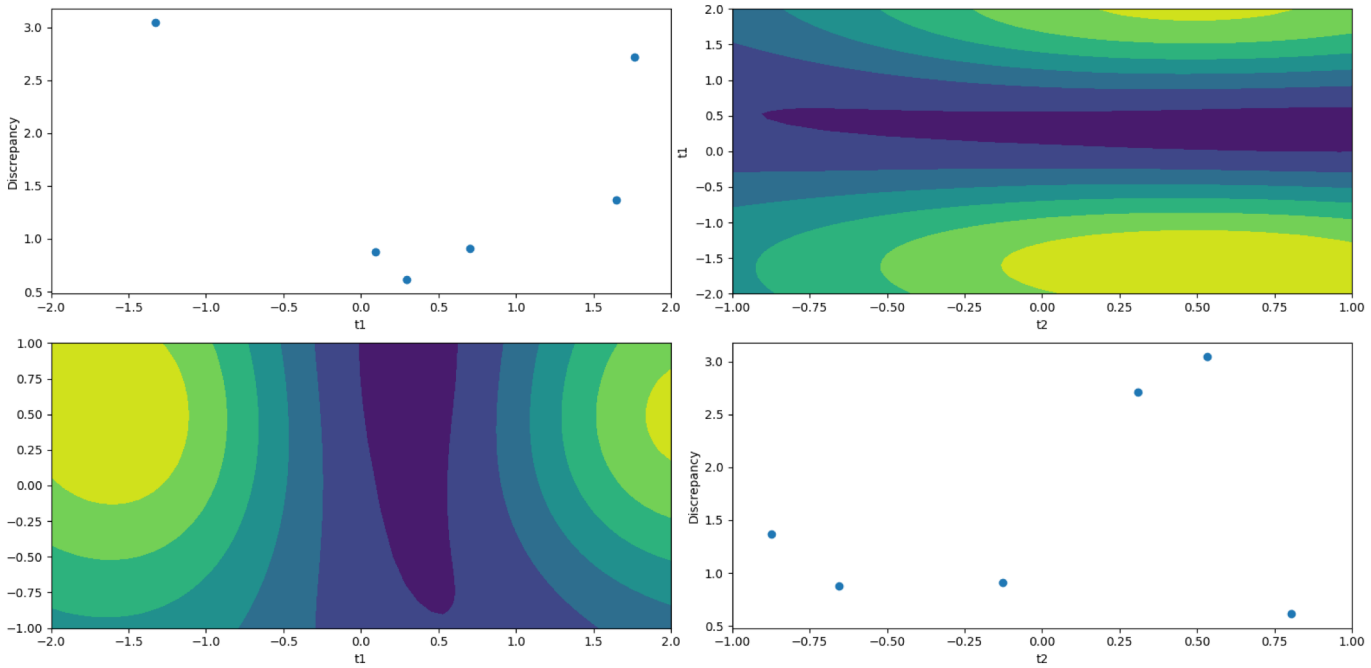
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 4)
```



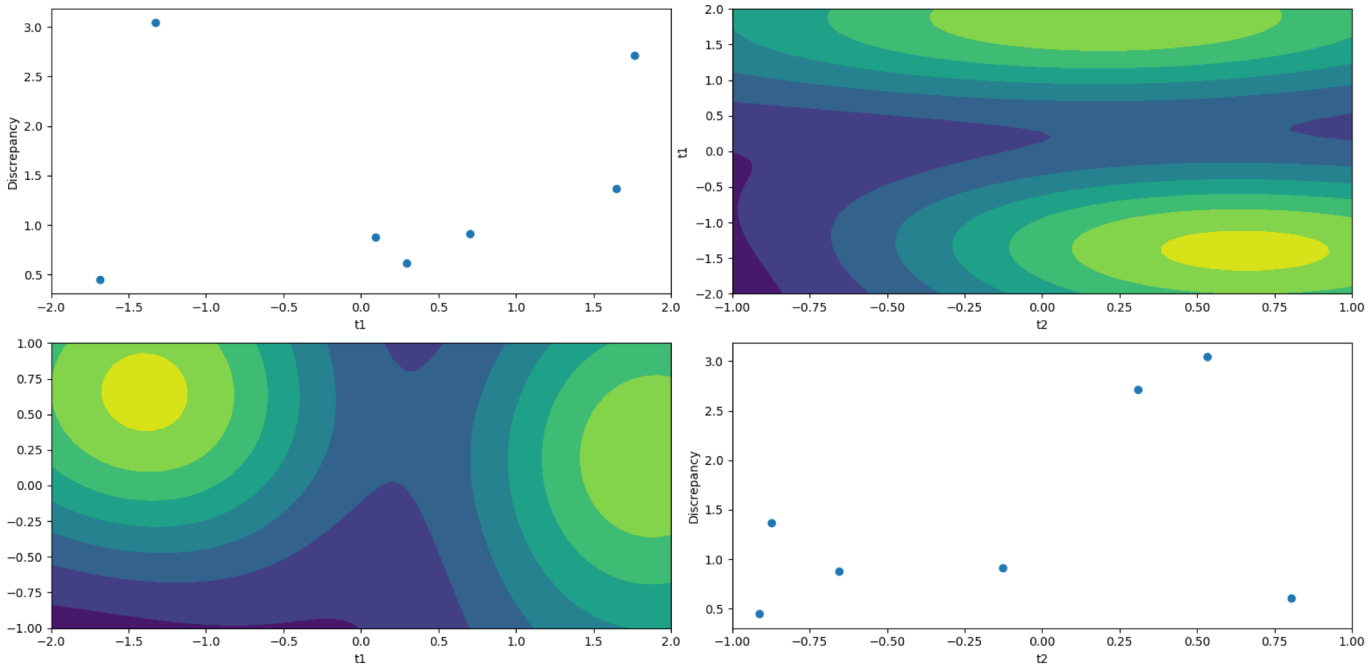
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 5)
```



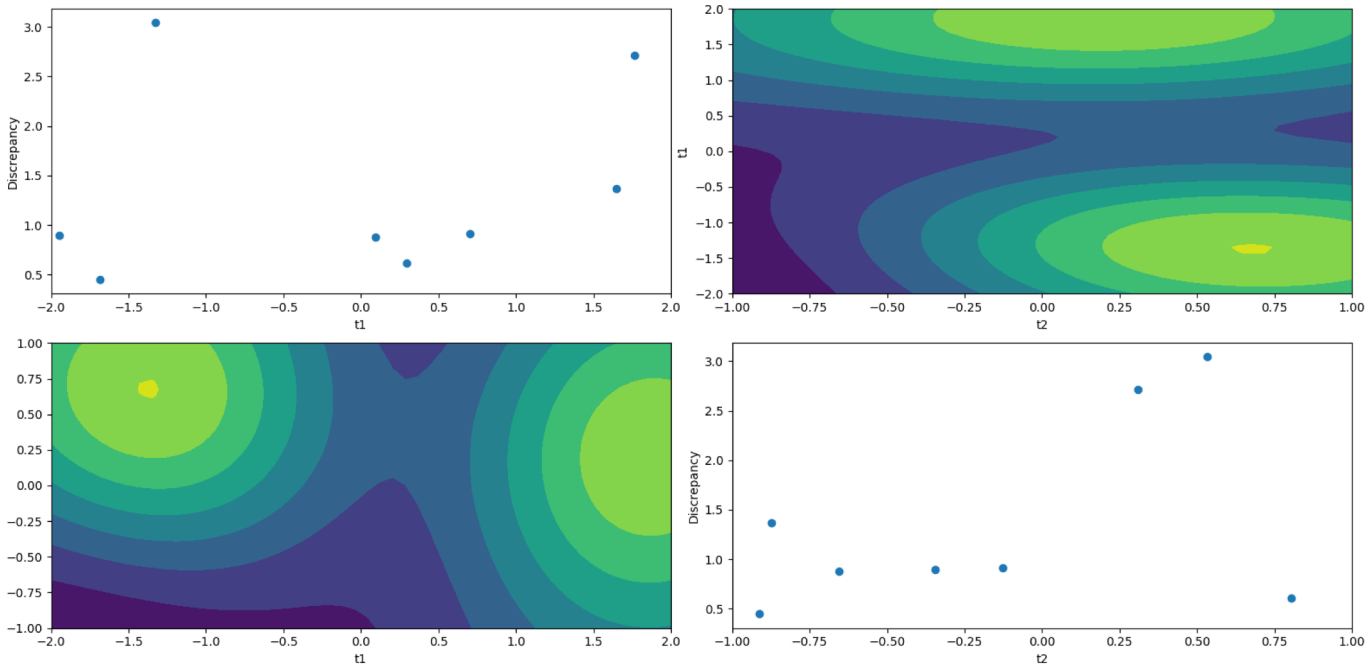
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 6)
```



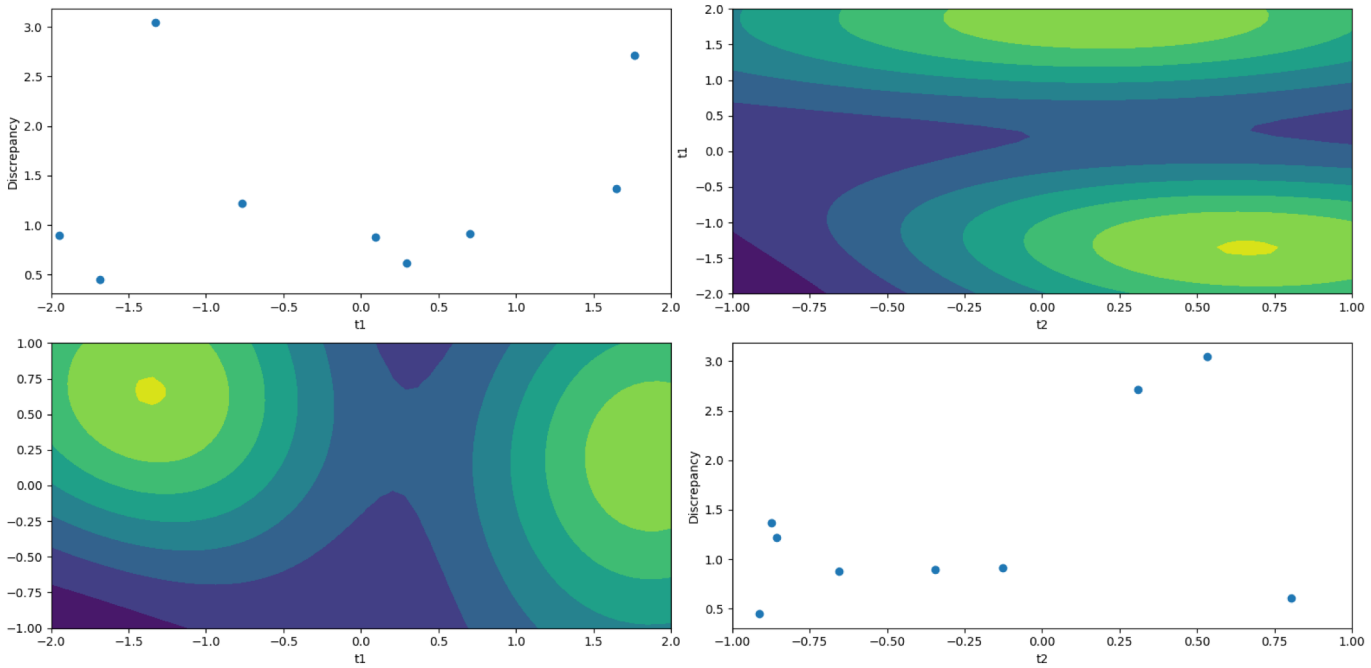
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 7)
```



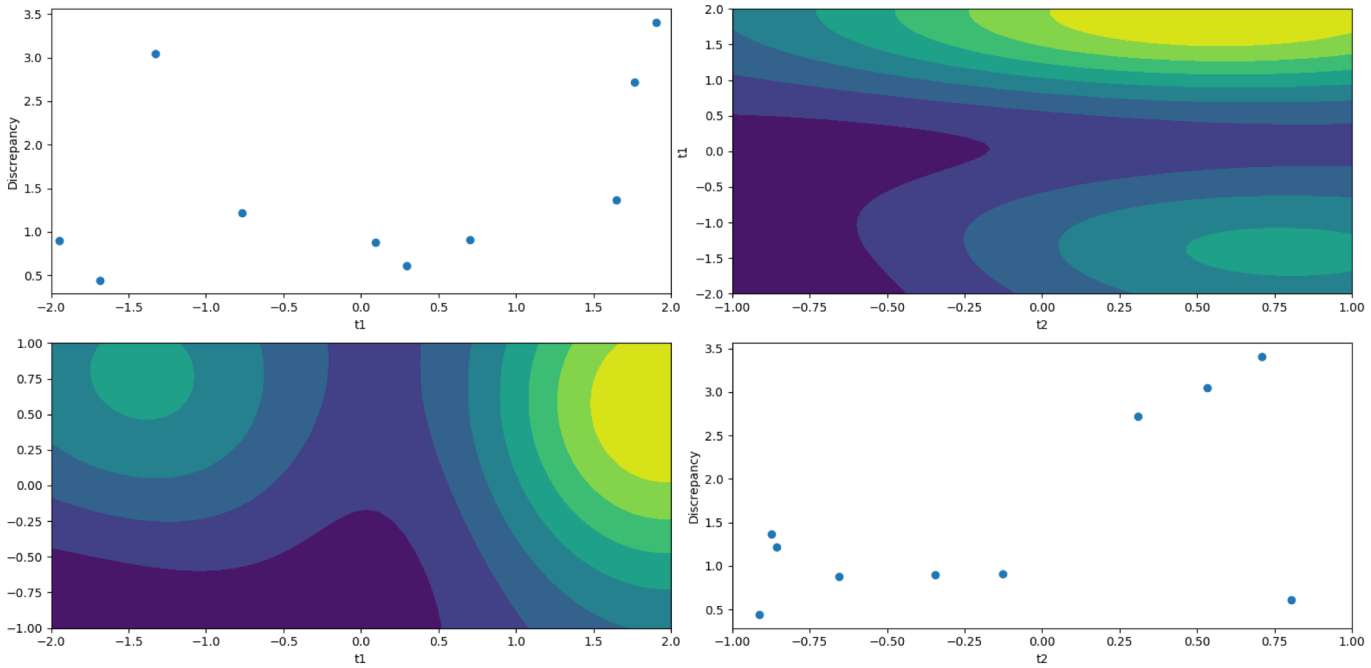
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 8)
```



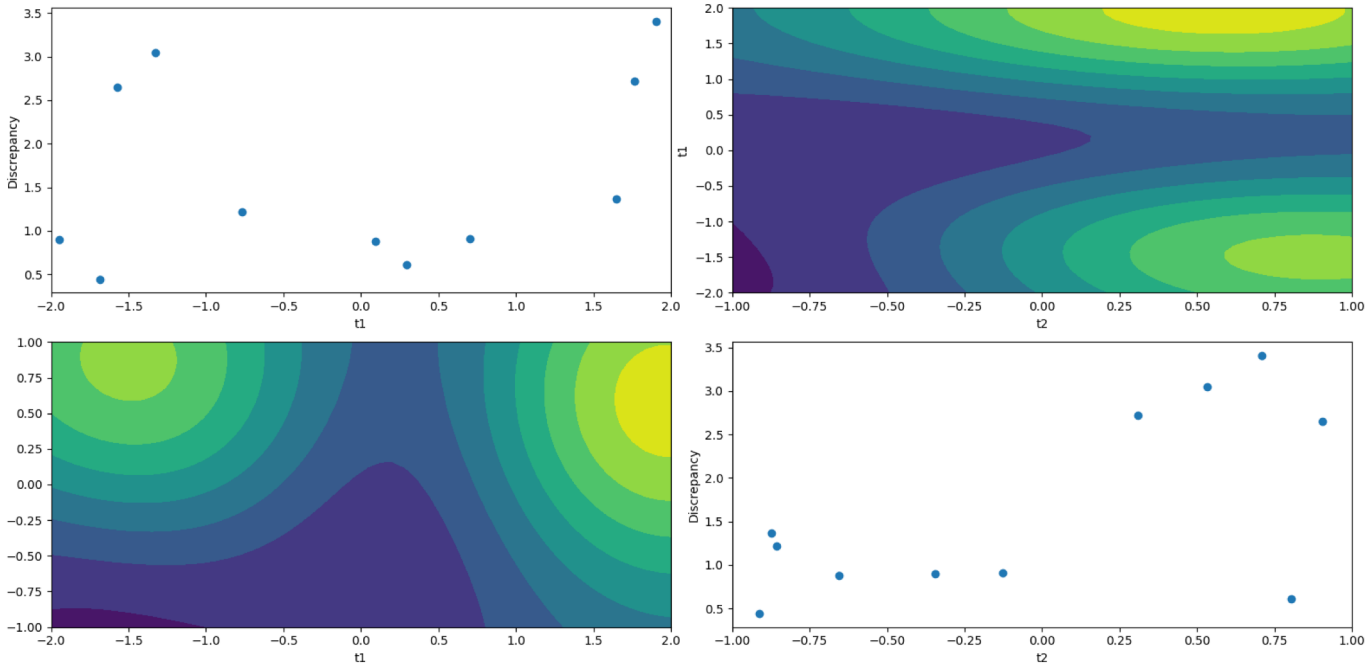
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 9)
```



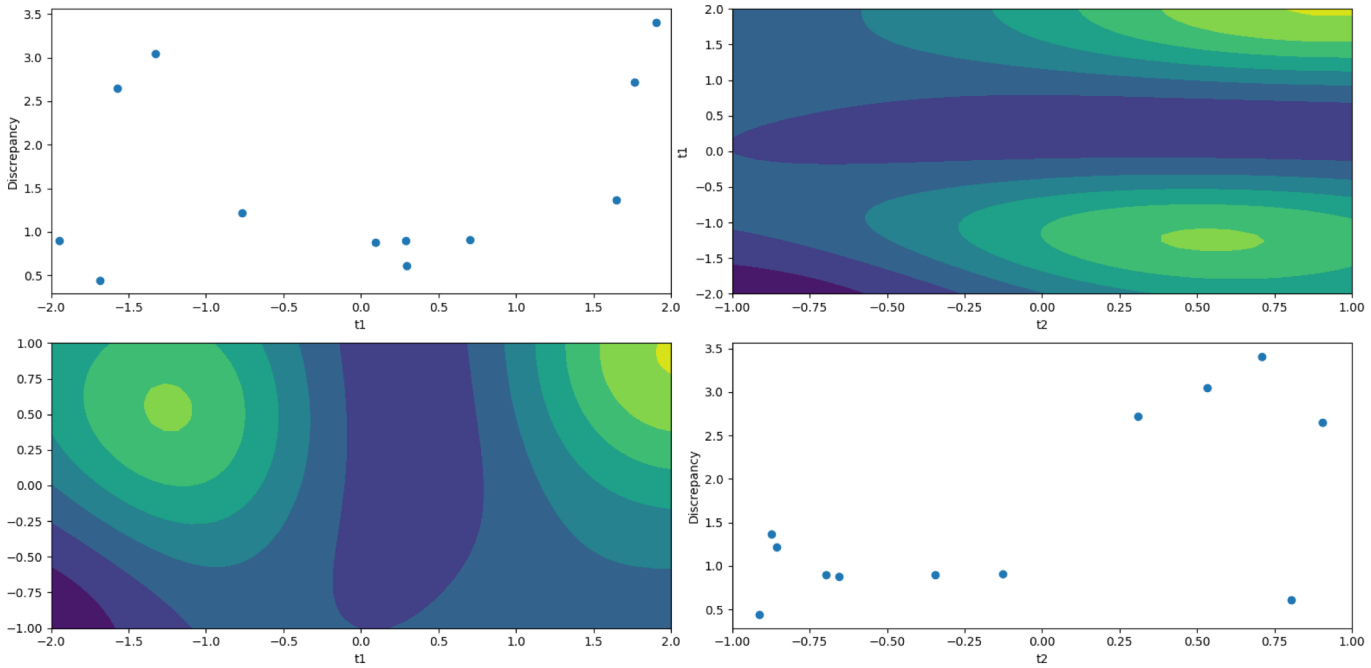
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 10)
```



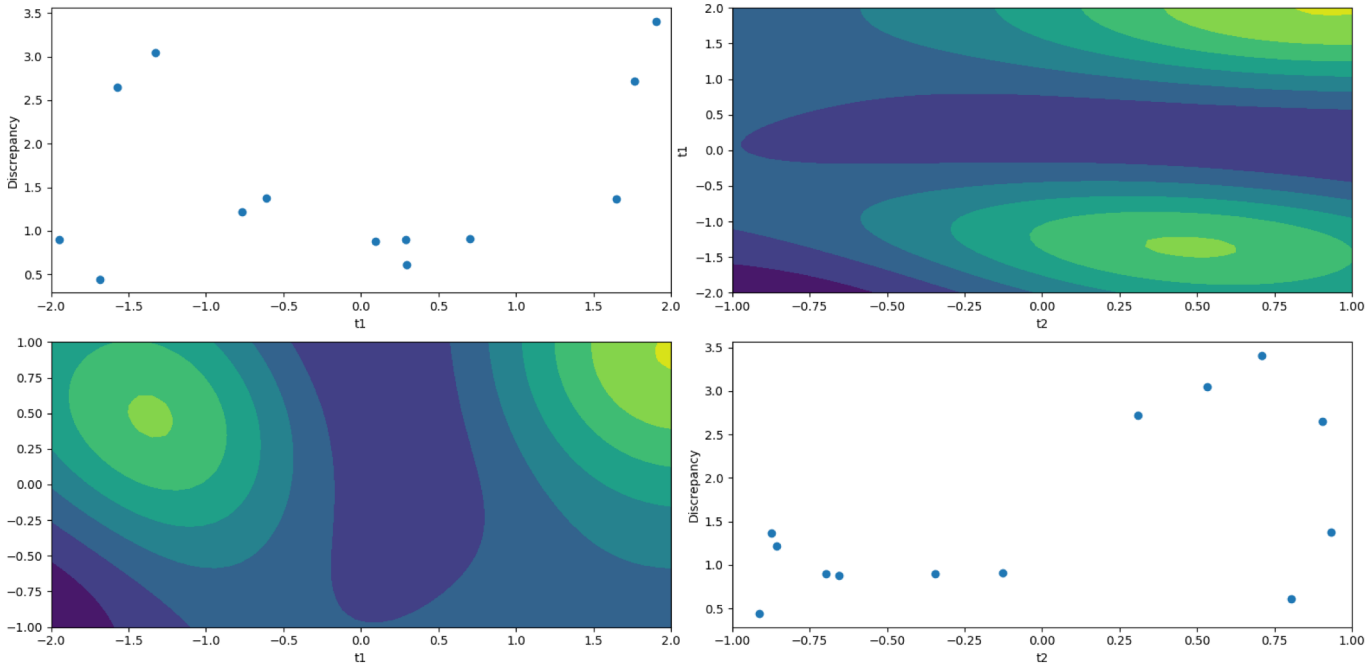
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 11)
```



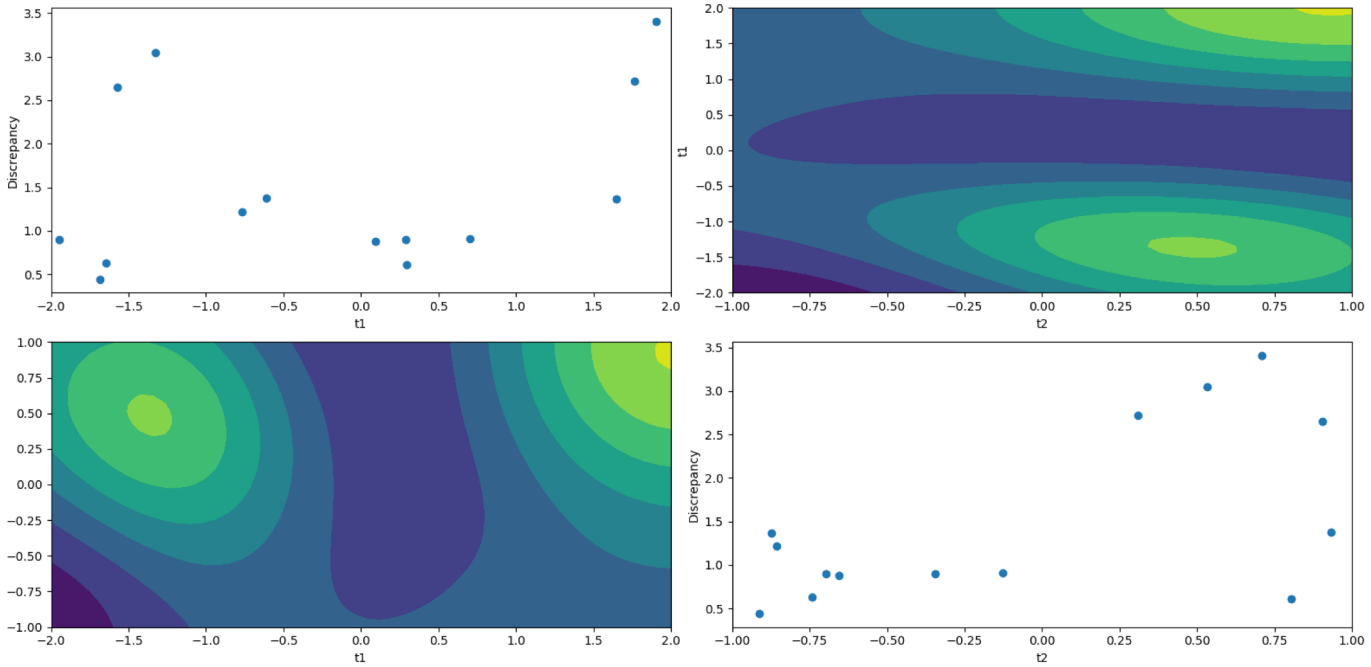
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 12)
```



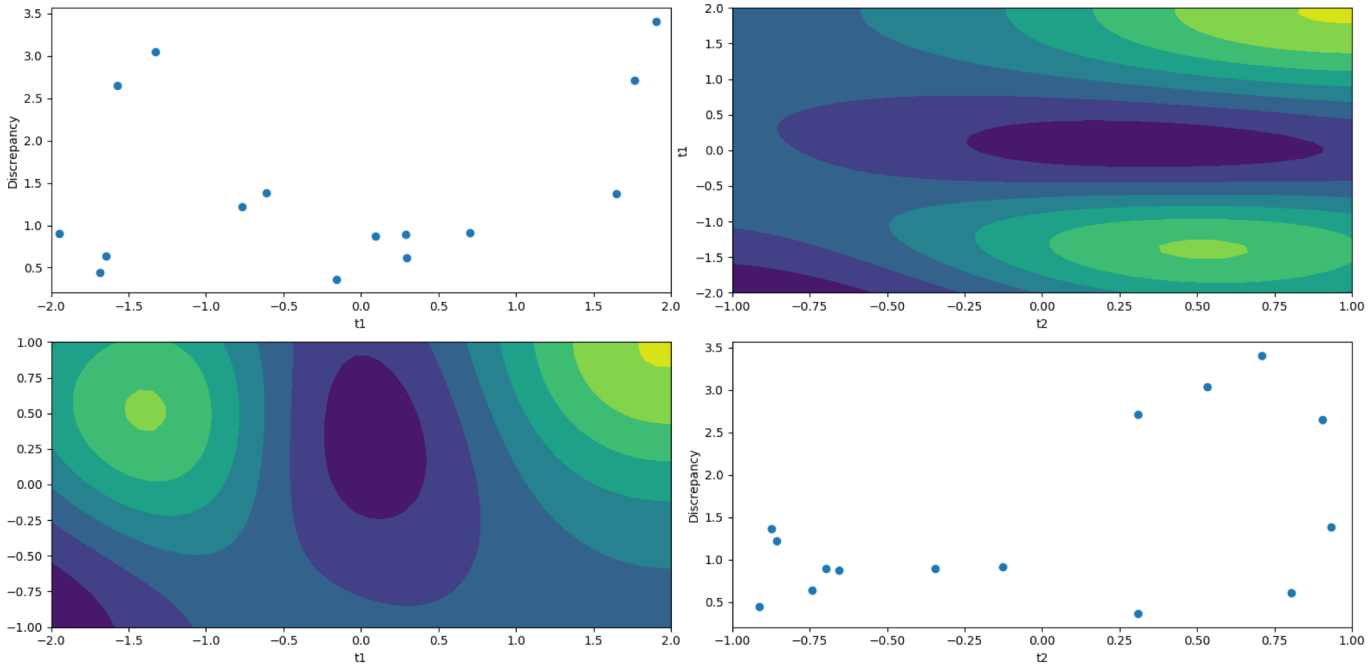
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 13)
```



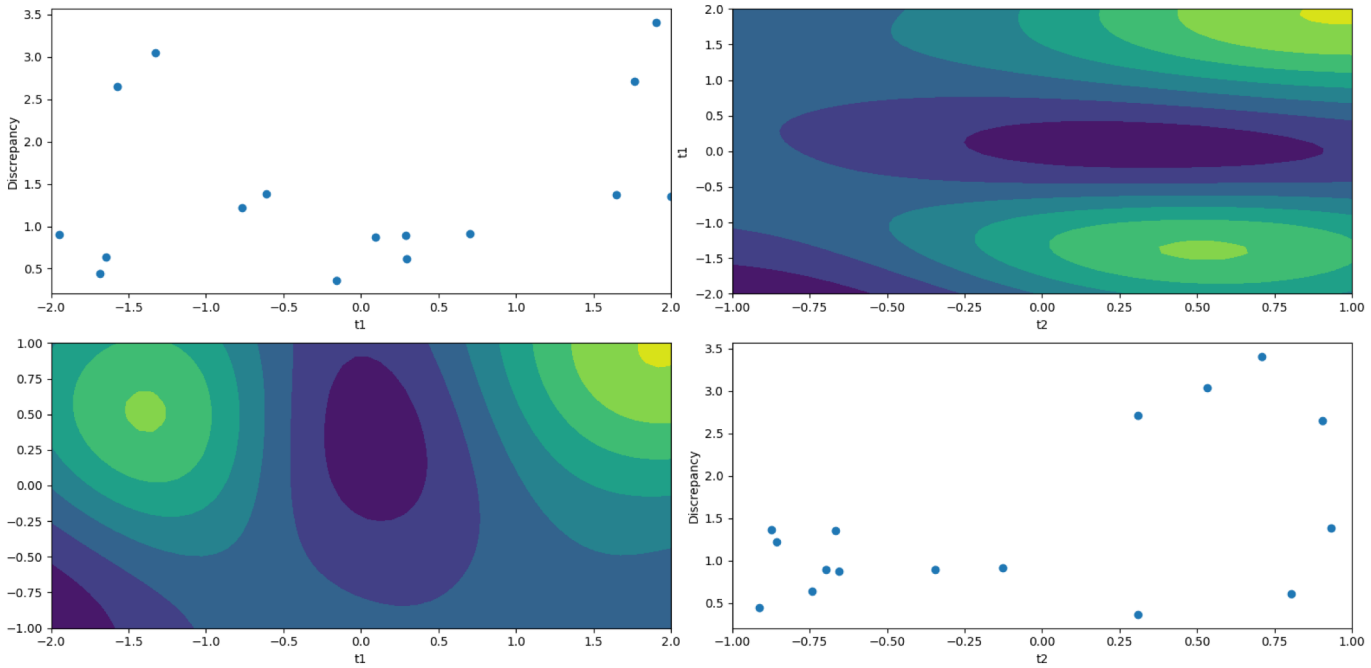
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 14)
```



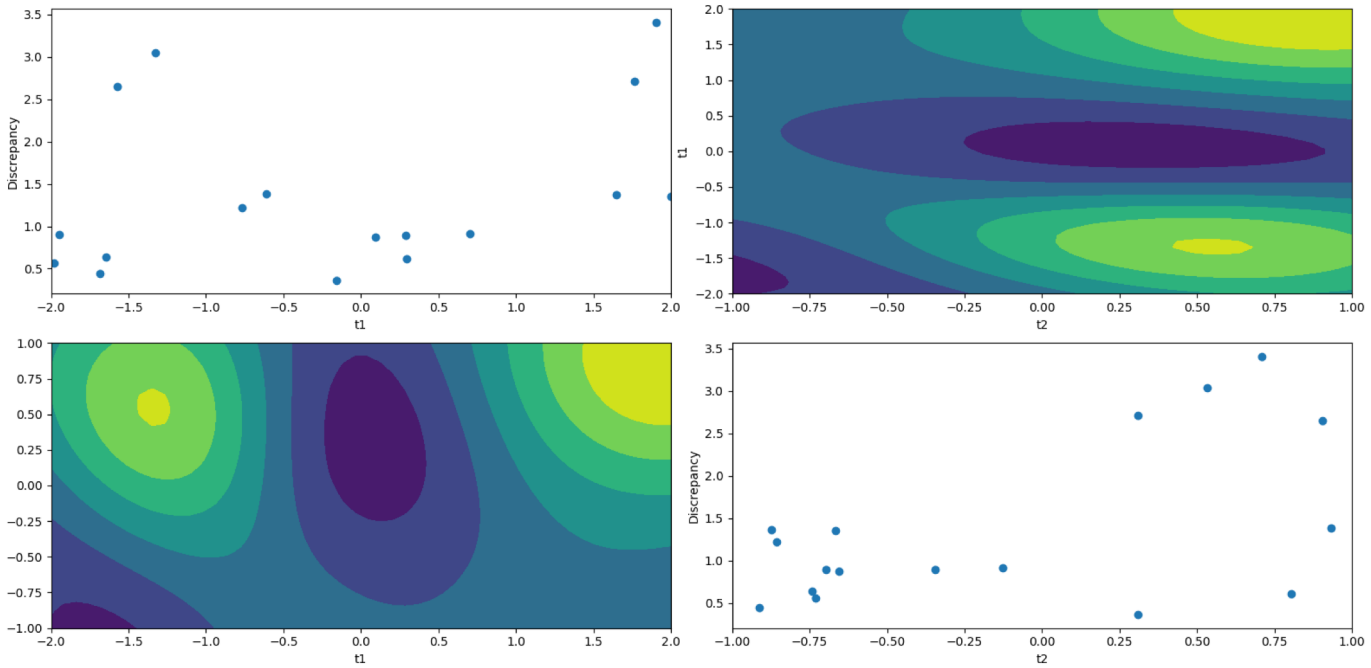
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 15)
```



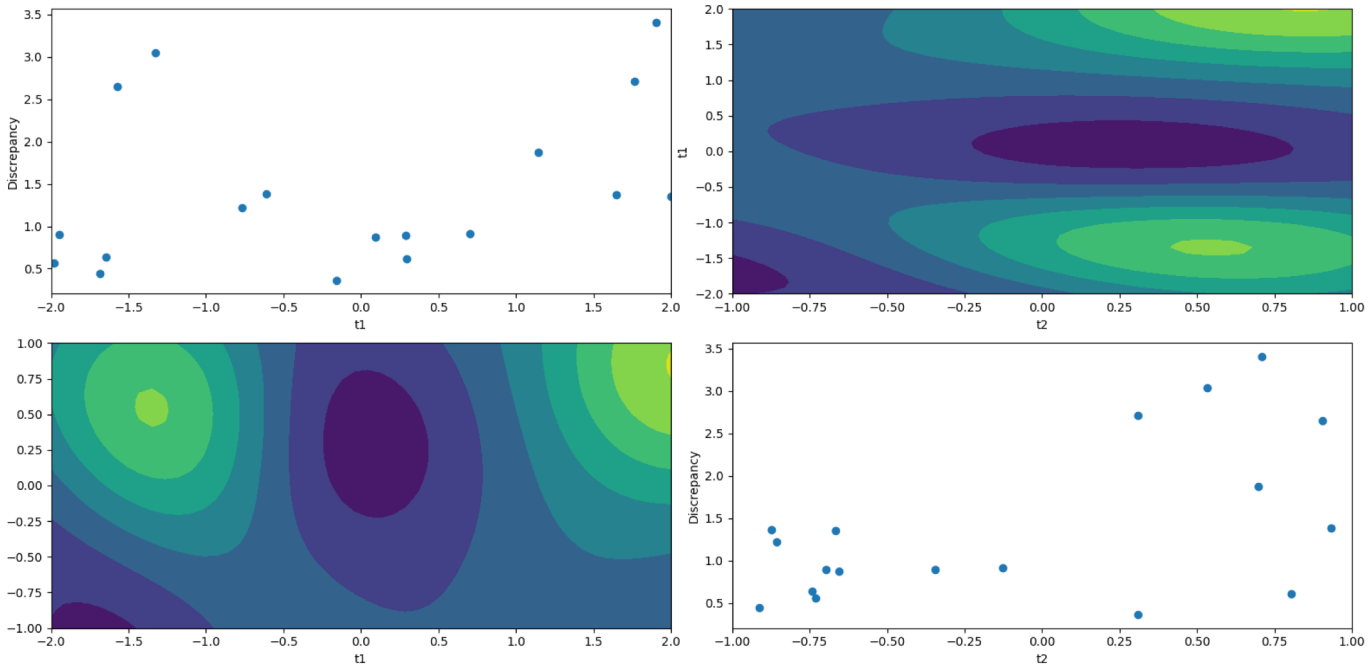
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 16)
```



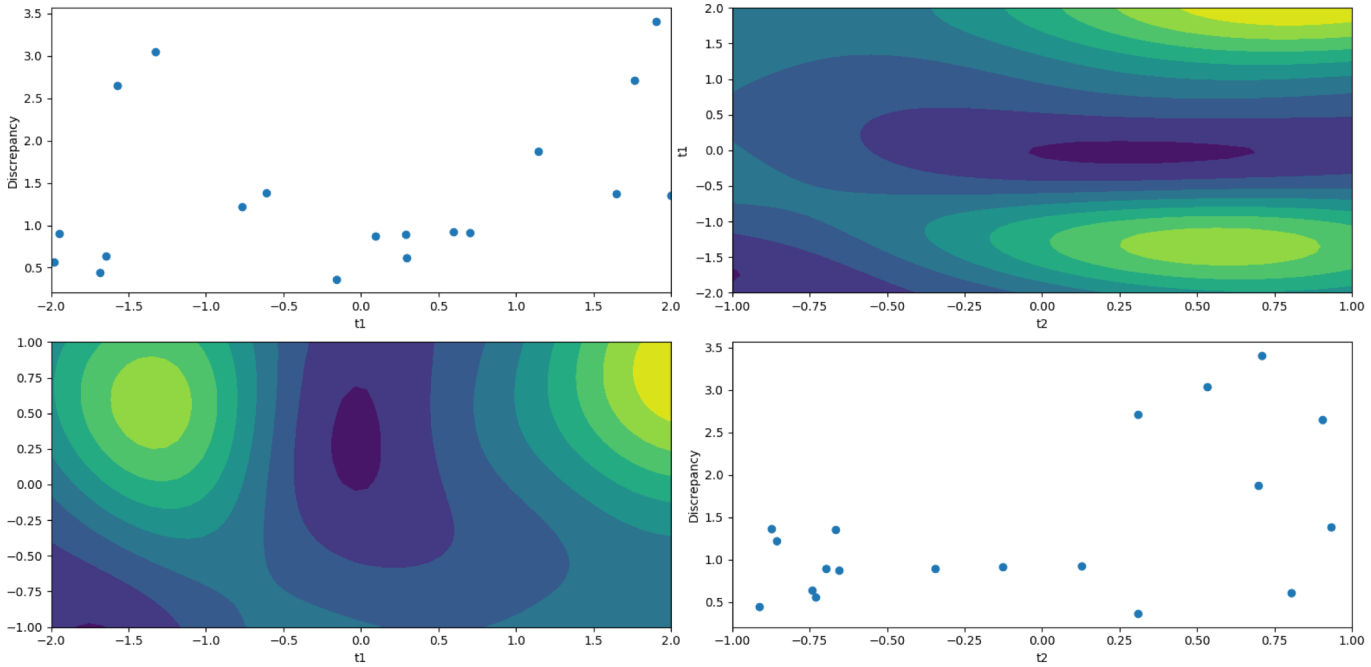
```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 17)
```



```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 18)
```



```
log_d = elfi.Operation(np.log, d)
bolfi = elfi.BOLFI(log_d, batch_size=1,
                   initial_evidence=2,
                   update_interval=5,
                   bounds={'t1':(-2, 2), 't2':(-1, 1)},
                   acq_noise_var=[0.1, 0.1], seed=seed)
post = bolfi.fit(n_evidence = 19)
```



Bayesian Optimization for Likelihood-free Inference

- Usually one wants to obtain a posterior sample from the bolfi-fit
- NUTS (default) and metropolis-samplers are available in ELFI

```
result_BOLFI = bolfi.sample(1000)
```

Bayesian Optimization for Likelihood-free Inference

- Usually one wants to obtain a posterior sample from the bolfi-fit
- NUTS (default) and metropolis-samplers are available in ELFI

```
result_BOLFI = bolfi.sample(1000)

4 chains of 1000 iterations acquired.
Effective sample size and Rhat for each parameter:
t1 2222.1197791 1.00106816947
t2 2256.93599184 1.0003364409
CPU times: user 1min 45s, sys: 1.29 s, total: 1min 47s Wall time:
55.1 s
```

References

- ELFI- Engine for Likelihood-free Inference. J. Lintusaari et al. JMLR, (16):1-7, 2018

- Bayesian Optimization for Likelihood-free Inference, M. Gutmann, J. Corander, 17(125):1-47, 2016.

Acknowledgements

- ELFI- core dev team
  - Past and present
- ELFI user community

elfi.readthedocs.io

gitter.im/elfi-dev/elfi

@henri_pesonen

Aalto University
School of Science

UNIVERSITY OF HELSINKI

UiO : University of Oslo

THE UNIVERSITY of EDINBURGH

Main features of ELFI

- Implementations of LFI/ABC algorithms
- Easy syntax for defining model structure called ELFI-graph
- Storage and re-use of results
- Reporting and diagnostic tools
- Designed to be extensible
- Automatic parallellization
  - `ipyparallel, multiprocessing,...`

ELFI

Engine for Likelihood-Free Inference

Want to contribute?

- Resources
  - elfi.readthedocs.io
  - github.com/elfi-dev/elfi

Want to contribute?

- Ideally you could be interested in LFI, but even that's not necessary
  - Code reviewing
  - Visualization
  - Code optimization & profiling
  - Improvements to API
- Code development uses the continuous integration practices
  - You have to try very hard to mess things up irreversibly!