

# Introduction to Numerical Optimization

Janoś Gabler and Tim Mensinger, University of Bonn



estimagic

# Installation

We assume you have done the following

- Installed miniconda or anaconda
- Executed:

```
git clone https://github.com/OpenSourceEconomics/euroscipy-estimagic.git  
cd euroscipy-estimagic  
conda env create -f environment.yml  
conda activate euroscipy-estimagic
```

- If you haven't done so, please do so until the first practice session
- Details: <https://github.com/OpenSourceEconomics/euroscipy-estimagic>

# About Us



- Website: [janosg.com](http://janosg.com)
- GitHub: [janosg](https://github.com/janosg)
- Started estimagic in 2019
- Postdoc in Econ, University of Bonn
- Open for interesting jobs



- Website: [tmensinger.com](http://tmensinger.com)
- GitHub: [timmens](https://github.com/timmens)
- estimagic core contributor
- PhD student in Econ, University of Bonn

# Sections

1. Introduction to **scipy.optimize**
2. Introduction to **estimagic**
3. Choosing algorithms
4. Bounds and constraints
5. Global optimization

# Structure of each topic

1. Summary of exercise you will solve
2. Some theory
3. Syntax in very simplified example
4. You solve a more difficult example in a notebook
5. Discuss one possible solution

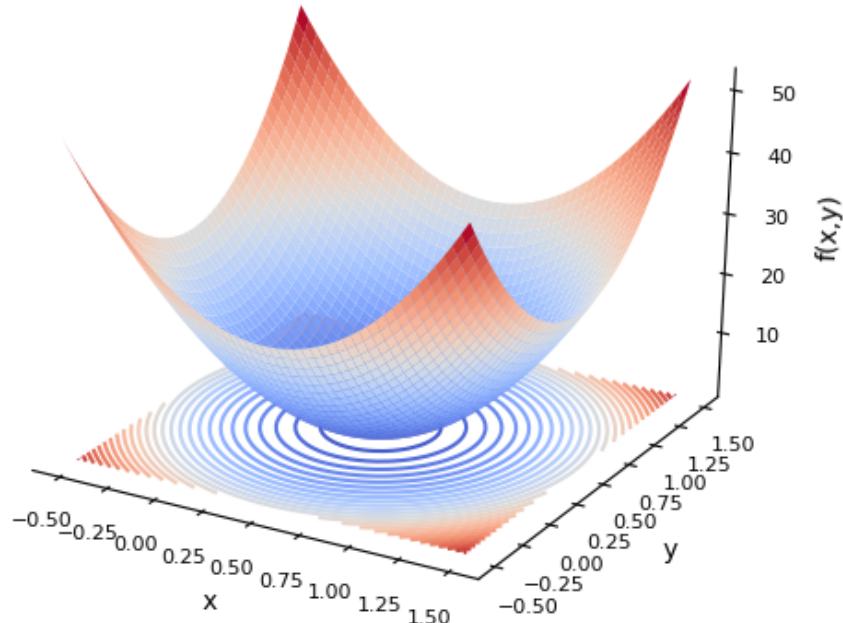
# Introduction to `scipy.optimize`

# Preview of practice session

- Translate an objective criterion function from math to code
- Use **scipy.optimize** to minimize the function

# Example problem

- Criterion  $f(a, b) = a^2 + b^2$
- Parameters  $a, b$
- Want:  $a^*, b^* = \operatorname{argmin} f(a, b)$
- Possible extensions:
  - Constraints
  - Bounds
- Optimum at  $a^* = 0, b^* = 0$ ,  
 $f(a^*, b^*) = 0$



# Optimization with `scipy.optimize`

```
import numpy as np
from scipy.optimize import minimize

def sphere(x):
    a, b = x
    return a ** 2 + b ** 2

x0 = np.ones(2)
res = minimize(sphere, x0)
res.fun

0.0

res.x

array([0.0, 0.0])
```

- **minimize** as unified interface to 14 local optimizers
- some support bounds or constraints
- Parameters are 1d arrays
- Maximize by minimizing  $-f(x)$
- Different interfaces for:
  - global optimization
  - nonlinear least-squares

# Practice Session

First optimization with `scipy.optimize` (12 min)

# Pros

- Very mature and reliable
- No additional dependencies
- Low overhead
- Enough algorithms for many use-cases

# Cons

- Relatively few algorithms
- No parallelization
- Maximization via sign flipping
- Feedback only at end
- No feedback in case of crash
- Parameters are flat arrays

→ **In practice, you often wish for more convenience**

# You hit exceptions ...

```
>>> scipy.optimize.minimize(func, x0)
-----
LinAlgError                                     Traceback (most recent call last)
<ipython-input-17-7459e5b4d8d4> in <module>
    1 scipy.optimize.minimize(func, x0)

    95
    96 def _raise_linalgerror_singular(err, flag):
→ 97     raise LinAlgError("Singular matrix")
    98

LinAlgError: Singular matrix
```

After 5 hours and with no additional information

# You write boilerplate code ...

```
def parse_parameters(x):
    """Parse the parameter vector into quantities we need."""
    num_types = int(len(x[54:]) / 6) + 1
    params = {
        'delta': x[0:1],
        'level': x[1:2],
        'coeffs_common': x[2:4],
        'coeffs_a': x[4:19],
        'coeffs_b': x[19:34],
        'coeffs_edu': x[34:41],
        'coeffs_home': x[41:44],
        'type_shares': x[44:44 + (num_types - 1) * 2],
        'type_shifts': x[44 + (num_types - 1) * 2:]
    }
    return params
```

# You make mistakes ...



Also, in the model solution (modelsol.R) the authors parameterize the fixed cost share and the Taylor rule inflation feedback parameter to the same thing. This is obviously a typo as cfc should be parameter 26 in the vector. I have no idea how this impacts their results.

dajmcodon initial commit

Latest commit 28d7874 on Mar 29, 2021 History

1 contributor

754 lines (653 sloc) | 22.6 KB

```
1 gensolution <- function(pvec) {
2   # Function takes in a lengthy vector of parameters and returns matrices nearly ready for Kalman input
3   stderr.ee <- pvec[1]
4   stderr.eo <- pvec[2]
5   stderr.ed <- pvec[3]
6   stderr.es <- pvec[4]
7   stderr.en <- pvec[5]
8   stderr,epinl <- pvec[6]
9   stderr,evn <- pvec[7]
10  ##
11  crhoe <- pvec[8]
12  crhob <- pvec[9]
13  crrho <- pvec[10]
14  crhose <- pvec[11]
15  crhono <- pvec[12]
16  crhopinf <- pvec[13]
17  crhow <- pvec[14]
18  cmu <- pvec[15]
19  cmu <- pvec[16]
20  csaedcost <- pvec[17]
21  csigma <- pvec[18]
22  chbar <- pvec[19]
23  cpdrobe <- pvec[20]
24  csig1 <- pvec[21]
25  cpdrob <- pvec[22]
26  cindw <- pvec[23]
27  cindp <- pvec[24]
28  czcap <- pvec[25]
29  cfc <- pvec[27]
30  crpi <- pvec[27]
31  crr <- pvec[28]
32  cry <- pvec[29]
```

...

26 cindw <- pvec[23]  
27 cindp <- pvec[24]  
28 czcap <- pvec[25]  
29 cfc <- pvec[27]  
30 crpi <- pvec[27]  
31 crr <- pvec[28]  
32 cry <- pvec[29]

czcap = pvec[25]

# Switching packages is painful ...



The SciPy optimizers and NLOpt have different conventions for the signature of the objective function. The [documentation for the objective function in NLOpt](#) says

0



The function `f` should be of the form:



```
def f(x, grad):  
    if grad.size > 0:
```



etc.

So you'll need to create an objective function that accepts two arguments, `x` and `grad`. If `grad.size > 0`, the function must fill in the array with the gradient.

Share Edit Follow

answered Apr 14, 2020 at 15:20



Warren Weckesser

110k • 19 • 194 • 214

# Introduction to estimagic

# Preview of practice session

- Translate a **scipy** optimization to **estimagic.minimize**
- Use dictionaries instead of flat arrays as parameters in the optimization
- Plot the optimization history (criterion and parameter)

# What is estimagic?



# estimagic

- Wraps optimizers from:
  - Scipy, Nlopt, TAO, Pygmo, ...
- Unified interface
- Additional functionality and convenience
- Tools for nonlinear estimation

# You can use it like scipy

```
import estimagic as em

def sphere(x):
    a, b = x
    return a ** 2 + b ** 2

res = em.minimize(
    criterion=sphere,
    params=np.ones(2),
    algorithm="scipy_lbfgsb",
)

res.params

array([ 0.,  0.])
```

- `minimize` works almost the same
- Supports local and global scipy algorithms
- Difference: No default algorithm

# Params can be almost anything

numpy arrays

```
params = np.array([1, 2, 3])
```

pandas objects

```
params = pd.Series([1, 2], index=["a", "b"])
```

scalars

```
params = 3.14
```

lists

```
params = [1, 2, 3]
```

dicts

```
params = {"a": 1, "b": 2, "c": [3, 4]}
```

NamedTuples

```
class Params(NamedTuple):
    beta: float = 0.95
    h: np.ndarray = np.ones(3)
```

→ You can nest these as deep as you want!

# Example: Dict of scalars and pandas Series

```
params = {"a": 0, "b": 1, "c": pd.Series([2, 3, 4])}

def dict_sphere(x):
    return x["a"] ** 2 + x["b"] ** 2 + (x["c"] ** 2).sum()

res = em.minimize(
    criterion=dict_sphere,
    params=params,
    algorithm="scipy_neldermead",
)
res.params
```

```
{'a': 0.,
 'b': 0.,
 'c': 0    0.
      1    0.
      2    0.
      dtype: float64}
```

# OptimizeResult

```
>>> res
```

Minimize with 5 free parameters terminated successfully after 805 criterion evaluations and 507 iterations.

The value of criterion improved from 30.0 to 1.6760003634613059e-16.

The `scipy_neldermead` algorithm reported: Optimization terminated successfully.

Independent of the convergence criteria used by `scipy_neldermead`, the strength of convergence can be assessed by the following criteria:

	one_step	five_steps
relative_criterion_change	1.968e-15**	2.746e-15**
relative_params_change	9.834e-08*	1.525e-07*
absolute_criterion_change	1.968e-16**	2.746e-16**
absolute_params_change	9.834e-09**	1.525e-08*

(\*\*\*: change  $\leq 1e-10$ , \*\*: change  $\leq 1e-8$ , \*: change  $\leq 1e-5$ . Change refers to a change between accepted steps. The first column only considers the last step. The second column considers the last five steps.)

# Access OptimizeResult's attributes

```
>>> res.criterion  
.  
>>> res.n_criterion_evaluations  
805  
>>> res.success  
True  
>>> res.message  
'Optimization terminated successfully.'  
>>> res.history.keys()  
dict_keys(['params', 'criterion', 'runtime'])
```

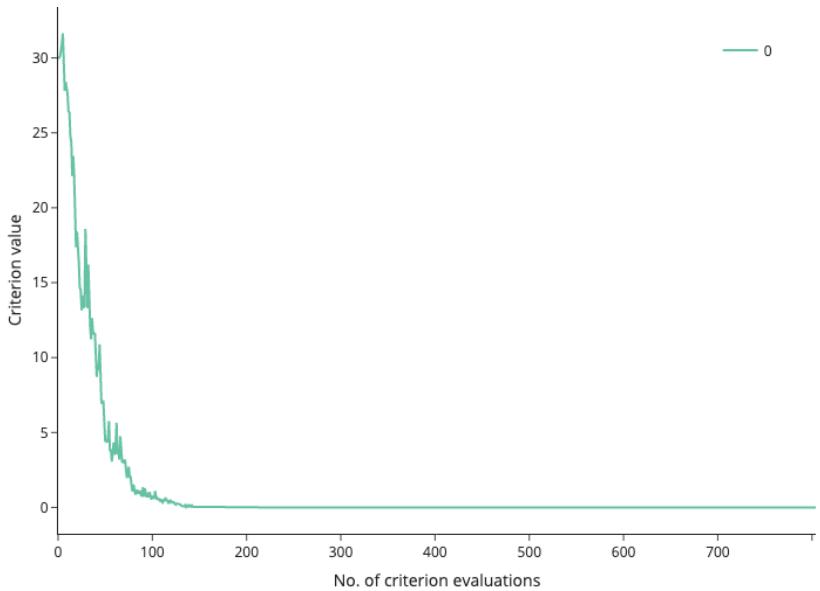
# Logging and Dashboard

```
res = em.minimize(  
    criterion=sphere,  
    params=np.arange(5),  
    algorithm="scipy_lbfgsb",  
    logging="my_log.db",  
)  
  
from estimagic import OptimizeLogReader  
  
reader = OptimizeLogReader("my_log.db")  
reader.read_history().keys()  
  
dict_keys(['params', 'criterion', 'runtime'])  
  
reader.read_iteration(1)[ "params" ]  
  
array([0., 0.817, 1.635, 2.452, 3.27 ])
```

- Persistent log in sqlite database
- No data loss ever
- Can be read during optimization
- Provides data for dashboard
- No SQL knowledge needed

# Criterion plot

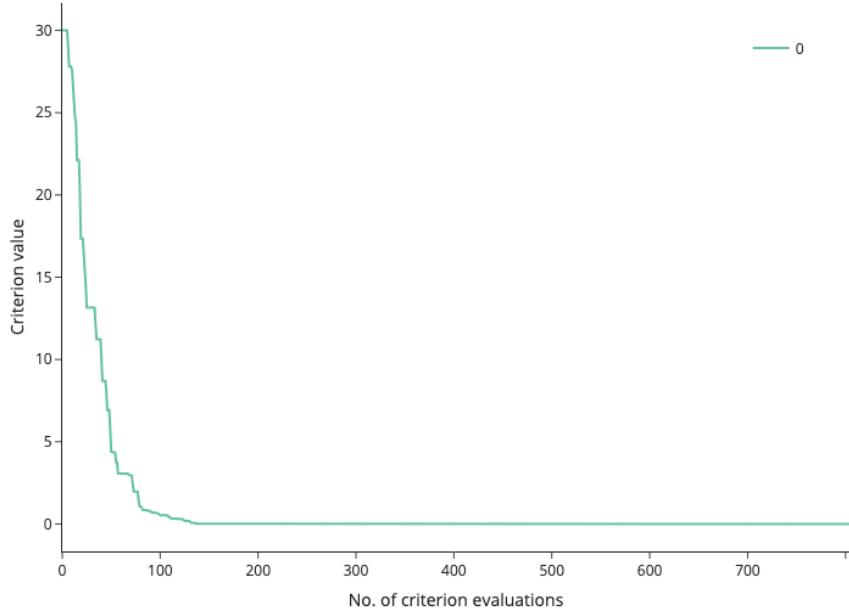
```
em.criterion_plot(res)
```



- First argument can be:
  - `OptimizeResult`
  - path to log file
  - list or dict thereof
- Dictionary keys are used for legend

# Criterion plot

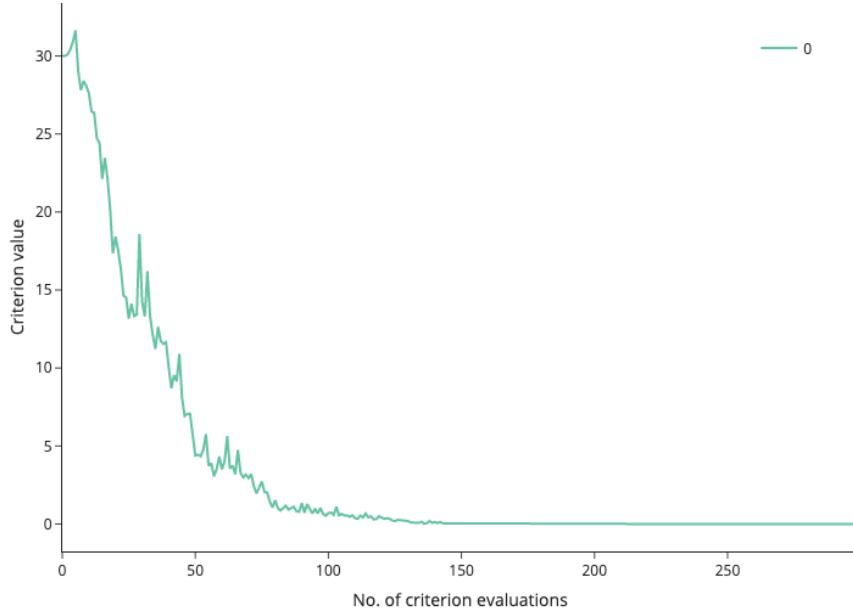
```
em.criterion_plot(res, monotone=True)
```



- **monotone=True** shows the current best value
- useful if there are extreme values in history

# Criterion plot

```
em.criterion_plot(res, max_evaluations=300)
```



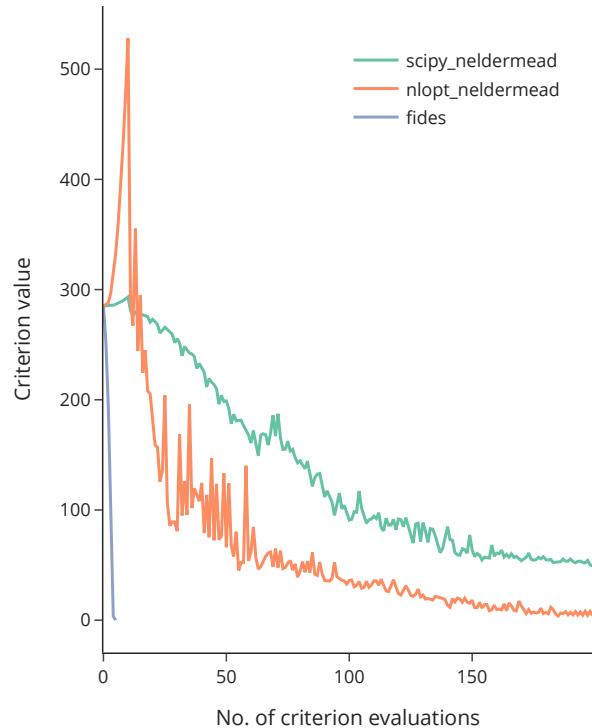
- **max\_evaluations** limits the x-axis

# Criterion plot for multiple optimizations

```
def sphere(x):
    return x @ x

results = {}
for algo in ["scipy_neldermead", "nlopt_neldermead", "fides"]:
    results[algo] = em.minimize(
        sphere,
        np.arange(10),
        algorithm=algo,
    )

em.criterion_plot(results, max_evaluations=200)
```



# Other features

- Parallel numerical derivatives and optimizers
- There is **maximize**
- Simple syntax for bounds and constraints
- Compatible with JAX automatic differentiation
- Sensible error handling
- Algorithms for nonlinear least-squares problems

# estimagic.readthedocs.io

## estimagic

[Getting Started](#) [How-to Guides](#) [Explanations](#) [API](#) [Development](#) [Optimizers](#)

Previous topic

[estimagic](#)

Next topic

[Installation](#)

Quick search

Go

## Getting Started

This section contains quickstart guides for new estimagic users. It can also serve as a reference for more experienced users.



Optimization

Learn numerical optimization with  
estimagic



Differentiation

Learn numerical differentiation with  
estimagic



Estimation

Learn maximum likelihood and  
methods of simulated moments  
estimation with estimagic



Installation

Installation instructions for estimagic  
and optional dependencies

## Practice Session

Convert previous example to estimagic (12 min)

# Choosing algorithms

# Preview of practice session

You will get an optimization problem that fails

- Figure out why it fails
- Choose an algorithm that works

# Relevant problem properties

- **Smoothness:** Differentiable? Kinks? Discontinuities? Stochastic?
  - **Convexity:** Are there local optima?
  - **Goal:** Do you need a global solution? How precise?
  - **Size:** 2 parameters? 10? 100? 1000? More?
  - **Constraints:** Bounds? Linear constraints? Nonlinear constraints?
  - **Structure:** Nonlinear least-squares, Log-likelihood function
- > Properties guide selection but experimentation is important

# Estimagic has many optimizers

The diagram illustrates the relationships between various optimization methods, categorized by package:

- nlopt**: nlopt\_praxis, nlopt\_ls\_dogbox, nlopt\_ls\_trf, nlopt\_ls\_lm, nlopt\_sbplx, nlopt\_esch, nlopt\_tnewton, nlopt\_bobyqa, nlopt\_ccsaq, ipopt, scipy\_basinhopping, simopt\_strong.
- scipy**: scipy\_ls\_dogbox, scipy\_ls\_trf, scipy\_neldermead, scipy\_ls\_lm, scipy\_bfgs, scipy\_neldermead, pygmo\_de1220, pygmo\_sea, pygmo\_bee\_colony, pygmo\_simulated\_annealing, scipy\_trust\_constr, pygmo\_compass\_search, scipy\_cobyla, nlopt\_slsqp, nlopt\_mma, nlopt\_lbfgs, nlopt\_var, nlopt\_slsqb, nlopt\_mma, nlopt\_lbfgs, nlopt\_lbfgsb, pygmo\_cmaes, simopt\_cmaes, scipy\_dual\_annealing, nlopt\_newuo, nlopt\_parallel, scipy\_conjugate\_gradient, scipy\_differential\_evolution, pygmo\_gwo, tao\_pounders, nlopt\_isres, pygmo\_mbh, pygmo\_simulated\_annealing, pygmo\_psdo, pygmo\_pso, pygmo\_gaco, pygmo\_xnes, pygmo\_ga, simopt\_spsa, pygmo\_ihs, pygmo\_sga, scipy\_brute, pygmo\_pso\_gen, nlopt\_crs2\_lm, nlopt\_direct, scipy\_shgo, scipy\_direct, scipy\_truncated\_newton, simopt\_astrodf, nlopt\_cobyla, scipy\_lbfgsb, nlopt\_cobyla, simopt\_adam, nag\_pybobyqa.
- pygmo**: pygmo\_sea, pygmo\_bee\_colony, pygmo\_simulated\_annealing, pygmo\_psdo, pygmo\_pso, pygmo\_gaco, pygmo\_xnes, pygmo\_ga, pygmo\_ihs, pygmo\_sga, pygmo\_pso\_gen.
- fides**: nlopt\_ls\_dogbox.
- ipopt**: ipopt.
- simopt**: simopt\_strong, simopt\_cmaes, simopt\_spsa.

# Optimization and derivatives

- The negative gradient of  $f$  is the direction of steepest descent
  - most optimizers use something gradient-like to find a search direction
- If the function is very curved, the descent direction is only valid locally
  - many optimizers use something hessian-like to determine the step size
- Below we give you a rough intuition for our go-to algorithms:
  - How do they work?
  - When should you use them

# scipy\_lbfgsb

- How it works:
  - Uses gradient for search direction
  - Uses approximated hessian (from gradient history) for step length
- When to use:
  - Differentiable functions with or without box constraints
  - Scales up to a few thousand parameters
  - Low overhead
  - Beats other BFGS implementations in many benchmarks

# fides

- How it works:
  - Uses gradient and approximated hessian to form taylor approximation
  - Optimum of taylor approximation is next candidate
  - This is called a *trustregion method*
- When to use:
  - Same situations as L-BFGS-B
  - More customizable
  - Good solution if **scipy\_lbfgsb** is too aggressive
  - Very underrated algorithm!

# **nlopt\_bobyqa, nag\_pybobyqa**

- How it works:
  - Uses interpolation to form a quadratic approximation
  - Optimum of approximation is next candidate
- When to use:
  - Go to algorithm if you do not have a closed form derivative
  - **nlopt** has less overhead
  - **nag** has options to deal with noise
  - Requires well scaled problems

# scipy\_neldermead, nlopt\_neldermead

- How it works:
  - Evaluate function on a set of points
  - Drop worst point and add new one
- When to use
  - No closed form derivative available
  - Badly scaled problems
  - Slightly noisy problems

# scipy\_ls\_lm, scipy\_ls\_trf

- Derivative based optimizers for nonlinear least squares
- Criterion needs structure:  $F(x) = \sum_i f_i(x)^2$
- In estimagic, criterion returns a dict:

```
def sphere_ls(x):
    # x are the least squares residuals in the sphere function
    return {"root_contributions": x, "value": x @ x}
```

- **scipy\_ls\_lm** is better for small problems without bounds
- **scipy\_ls\_trf** is better for problems with many parameters

→ **Always exploit least-squares structure if you can!**

# **nag\_dfols, pounders**

- Derivative free trust region methods for nonlinear least-squares
- Both beat bobyqa for least-squares problems!
- **nag\_dfols** is usually the fastest
- **nag\_dfols** has advanced options to deal with noise
- **pounders** can do criterion evaluations in parallel

# ipopt

- Probably best open source optimizer for nonlinear constraints

# Practice Session

Fix a failing optimization (10 min)

# Bounds and constraints

# Preview of practice session

- Solve optimization problem with
  - parameter bounds
  - fixed parameters

# Bounds

- Lower and upper bounds on parameters
- Also called box constraints
- Handled by most algorithms
- Need to hold for start parameters
- Guaranteed to hold during entire optimization
- Specification depends on **params** format

# How to specify bounds for array params

```
>>> def sphere(x):
...     return x @ x

>>> res = em.minimize(
...     criterion=sphere,
...     params=np.arange(3) + 1,
...     lower_bounds=np.ones(3),
...     algorithm="scipy_lbfgsb",
... )
>>> res.params
array([1., 1., 1.])
```

- Specify **lower\_bounds** and **upper\_bounds**
- Use **np.inf** or **-np.inf** to represent no bounds

# How to specify bounds for DataFrame params

```
>>> params = pd.DataFrame({  
...     "value": [1, 2, 3],  
...     "lower_bound": [1, 1, 1],  
...     "upper_bound": [3, 3, 3],  
... },  
...     index=["a", "b", "c"],  
... )  
  
>>> def criterion(p):  
...     return (p["value"] ** 2).sum()  
  
>>> em.minimize(criterion, params, algorithm="scipy_lbfgsb")
```

# How to specify bounds for pytree params

```
params = {"x": np.arange(3), "intercept": 3}

def criterion(p):
    return p["x"] @ p["x"] + p["intercept"]

res = em.minimize(
    criterion,
    params=params,
    algorithm="scipy_lbfgsb",
    lower_bounds={"intercept": -2},
)
```

- Enough to specify the subset of params that actually has bounds
- We try to match your bounds with params
- Raise **InvalidBoundsError** in case of ambiguity

# Constraints

- Constraints are conditions on parameters
- Linear constraints
  - $\min_x f(x) \text{ s.t. } l \leq Ax \leq u$
  - $\min_x f(x) \text{ s.t. } Ax = v$
- Nonlinear constraints:
  - $\min_x f(x) \text{ s.t. } c_1(x) = 0, c_2(x) \geq 0$
- "estimagic-constraints":
  - E.g. find probabilities or covariance matrix, fix parameters, ...

# Example: Find valid probabilities

```
>>> res = em.minimize(  
...     criterion=sphere,  
...     params=np.array([0.1, 0.5, 0.4, 4, 5]),  
...     algorithm="scipy_lbfgsb",  
...     constraints=[{  
...         "loc": [0, 1, 2],  
...         "type": "probability"  
...     }],  
... )  
  
>>> res.params  
array([0.33334, 0.33333, 0.33333, -0., 0.])
```

- Restrict first 3 parameters to be probabilities
  - Between 0 and 1
  - Sum to 1
- But "scipy\_lbfgsb" is unconstrained?!

# What happened

- Estimagic can implement some types of constraints via reparametrization
- Transforms a constrained problem into an unconstrained one
- Constraints must hold in start params
- Guaranteed to hold during entire optimization

# Which constraints can be handled via reparametrization?

- Fixing parameters (simple but useful)
- Finding valid covariance and correlation matrices
- Finding valid probabilities
- Linear constraints (as long as there are not too many)
  - $\min f(x) \text{ s.t. } A_1x = 0 \text{ and } A_2x \leq 0$

# Fixing parameters

```
>>> def criterion(params):
...     offset = np.linspace(1, 0, len(params))
...     x = params - offset
...     return x @ x

unconstrained_optimum = [1, 0.8, 0.6, 0.4, 0.2, 0]

>>> res = em.minimize(
...     criterion=criterion,
...     params=np.array([2.5, 1, 1, 1, 1, -2.5]),
...     algorithm="scipy_lbfgsb",
...     constraints={"loc": [0, 5], "type": "fixed"},
... )
>>> res.params
array([ 2.5,  0.8,  0.6,  0.4,  0.2, -2.5])
```

- **loc** selects location 0 and 5 of the parameters
- **type** states that they are fixed

# Linear constraints

```
>>> res = em.minimize(  
...     criterion=criterion,  
...     params=np.ones(6),  
...     algorithm="scipy_lbfgsb",  
...     constraints={  
...         "loc": [0, 1, 2, 3],  
...         "type": "linear",  
...         "lower_bound": 0.95,  
...         "weights": 0.25,  
...     },  
... )  
>>> res.params  
array([ 1.25,  1.05,  0.85,  0.65,  0.2 , -0.])
```

- Impose that average of first 4 parameters is larger than 0.95
- Weights can be scalars or same length as selected parameters
- Use "value" instead of "lower\_bound" for linear equality constraint

# Constraints have to hold

```
>>> em.minimize(  
...     criterion=sphere,  
...     params=np.array([1, 2, 3, 4, 5]),  
...     algorithm="scipy_lbfgsb",  
...     constraints={"loc": [0, 1, 2], "type": "probability"},  
... )
```

```
InvalidParamsError: A constraint of type 'probability' is not fulfilled in params,  
please make sure that it holds for the starting values. The problem arose because:  
Probabilities do not sum to 1. The names of the involved parameters are: ['0', '1', '2']  
The relevant constraint is:
```

```
{"loc": [0, 1, 2], "type": "probability", "index": array([0, 1, 2])}.
```

# Nonlinear constraints

```
>>> res = em.minimize(  
...     criterion=criterion,  
...     params=np.ones(6),  
...     algorithm="scipy_slsqp",  
...     constraints={  
...         "type": "nonlinear",  
...         "loc": [0, 1, 2, 3, 4],  
...         "func": lambda x: np.prod(x),  
...         "value": 1.0,  
...     },  
... )  
>>> res.params  
array([1.31, 1.16, 1.01, 0.87, 0.75, -0.])
```

- Restrict the product of first five params to 1
- Only work with some optimizers
- **func** can be an arbitrary python function of params that returns a number, array or pytree
- Use "lower\_bound" and "upper\_bound" instead of "value" for inequality constraints

# Parameter selection methods

- "**loc**" can be replaced other things
  - If params is a DataFrame with "value" column
    - "**query    - "**locDataFrame.loc****
  - Always
    - "**selector**
  - More in the documentation
-

# Practice Session

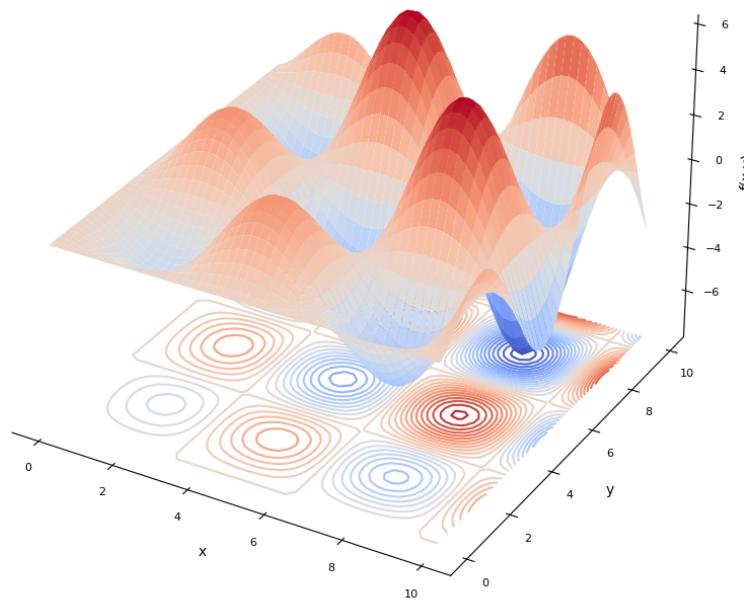
## Constrained optimization (8 min)

# Global optimization

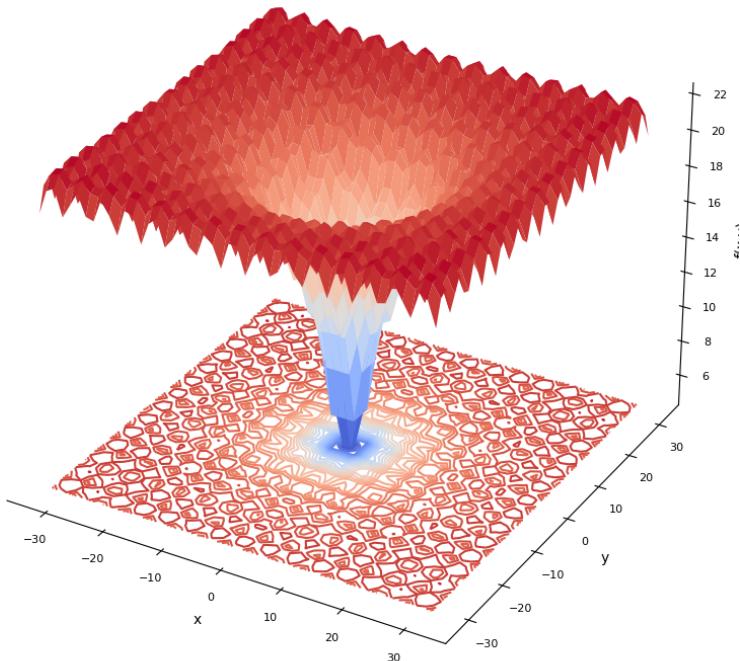
# Global vs local optimization

- Local: Find any local optimum
  - All we have done so far
- Global: Find best local optimum
  - Needs bounds to be well defined
  - Extremely challenging in high dimensions
- Local = global for convex problems

# Examples

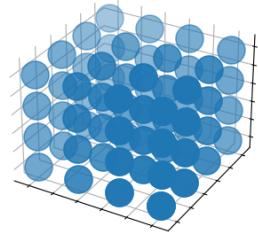
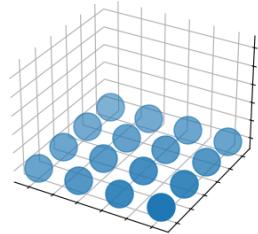
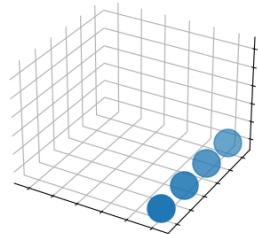


Alpine



Ackley

# How about brute force?



Dimension	Runtime (1 ms per evaluation, 100 points per dimension)
1	100 ms
2	10 s
3	16 min
4	27 hours
5	16 weeks
6	30 years

# Genetic algorithms

- Heuristic inspired by natural selection
- Random initial population of parameters
- In each evolution step:
  - Evaluate "fitness" of all population members
  - Replace worst by combinations of good ones
- Converge when max iterations are reached
- Examples: "`pygmo_gaco`", "`pygmo_bee_colony`", "`nlopt_crs2_lm`", ...

# Bayesian optimization

- Evaluate criterion on grid or sample of parameters
- Build surrogate model of criterion
- In each iteration
  - Do new criterion evaluations at promising points
  - Improve surrogate model
- Converge when max iterations is reached

# Multistart optimization:

- Evaluate criterion on random exploration sample
- Run local optimization from best point
- In each iteration:
  - Combine best parameter and next best exploration point
  - Run local optimization from there
- Converge if current best optimum is rediscovered several times

# Multistart example

```
>>> res = em.minimize(  
...     criterion=sphere,  
...     params=np.arange(5),  
...     algorithm="scipy_neldermead",  
...     soft_lower_bounds=np.full(5, -5),  
...     soft_upper_bounds=np.full(5, 15),  
...     multistart=True,  
...     multistart_options={  
...         "convergence.max_discoveries": 5,  
...         "n_samples": 1000  
...     },  
... )  
>>> res.params  
array([0., 0., 0., 0., 0.])
```

- Turn local optimizers global
- Inspired by tiktak algorithm
- Use any optimizer
- Distinguish hard and soft bounds

# How to choose

- Extremely expensive criterion (i.e. can only do a few evaluations):
  - Bayesian optimization
- Well behaved function:
  - Multistart with local optimizer tailored to function properties
- Rugged function with extremely many local optima
  - Genetic optimizer
  - Consider refining the result with local optimizer
- All are equally parallelizable

# Summary

# Summary

- You have solved a simple problem with `scipy.optimize`
- For larger problems, `estimagic` provides more convenience
  - ``params`` don't have to be flat arrays
  - Support for many more algorithms
  - ``criterion_plot`` and ``params_plot`` help you select the right algorithm
  - Logging and error handling help to deal with crashes
  - Multistart to make local optimizers global
- Many more features to explore in the documentation

# The team behind estimagic



Janoś Gabler



Mariam Petrosyan

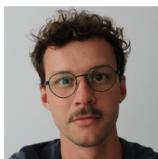


Tim Mensinger



Klara Röhrl

**NUMFOCUS**  
OPEN CODE = BETTER SCIENCE



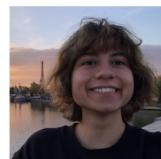
Tobias Raabe



Annica Gehlen



Sebastian Gsell



Bahar Coskun



UNIVERSITÄT BONN



Aida  
Takhmazova



Hans-Martin von  
Gaudecker



Kenneth L. Judd



Stanford University

**HOOVER**  
**INSTITUTION**

# How to contribute

- Make issues or provide feedback
- Improve or extend the documentation
- Suggest, wrap or implement new optimizers
- Teach estimagic to colleagues, students and friends
- Make us happy by giving us a star on

[github.com/OpenSourceEconomics/estimagic](https://github.com/OpenSourceEconomics/estimagic)

---