

Practical Numerical Optimization with Scipy, Estimagic and JAXopt

Scipy Conference 2022

Janos Gabler & Tim Mensinger

University of Bonn

About Us



- Website: janosg.com
- GitHub: [jano](https://github.com/janosg)
- Original author of estimagic
- Submitted PhD thesis, looking for interesting jobs soon
- Website: tmensinger.com
- GitHub: [timmens](https://github.com/timmens)
- estimagic core contributor
- PhD student in Econ, University of Bonn

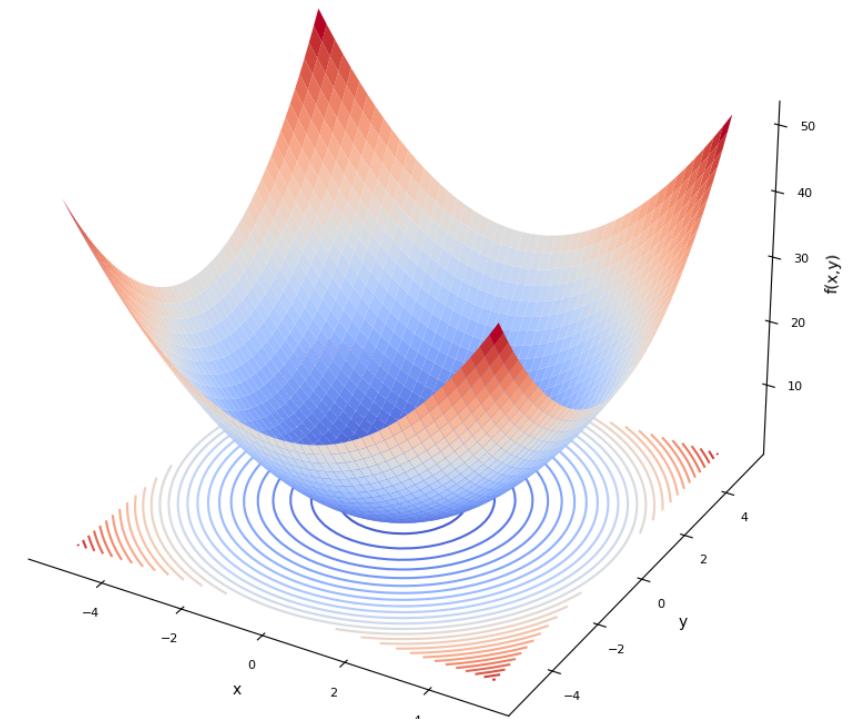
Index

1. What is numerical optimization
2. Introduction to `scipy.optimize`
3. Introduction to `estimagic`
4. Choosing algorithms
5. Advanced `estimagic`
6. Jax and Jaxopt

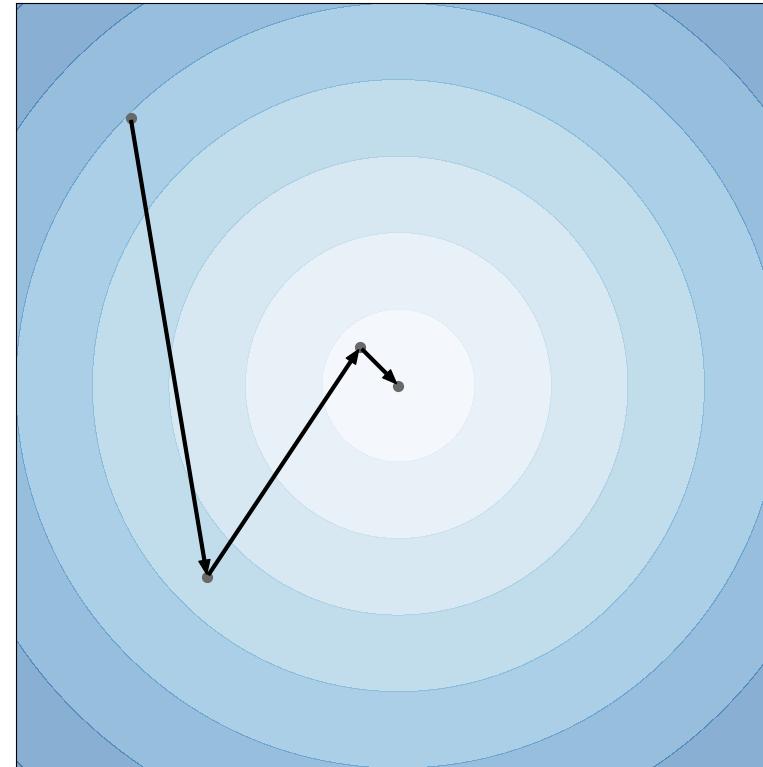
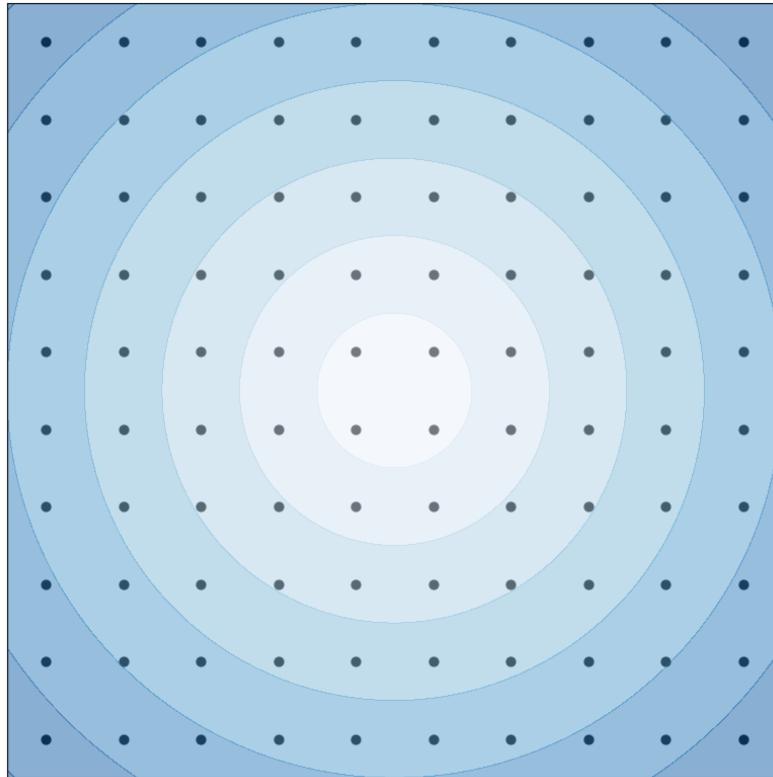
What is numerical optimization

Example problem

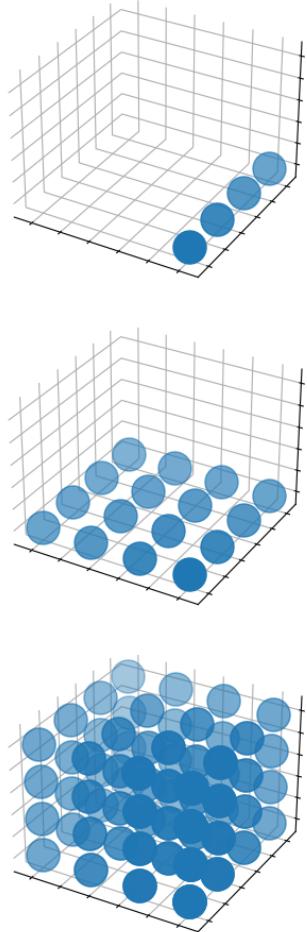
- Parameters x_1, x_2
- Criterion $f(x_1, x_2) = x_1^2 + x_2^2$
- Want: $x_1^*, x_2^* = \operatorname{argmin} f(x_1, x_2)$
- Possible extensions:
 - Constraints
 - Bounds
- Optimum at $(0, 0)$ with function value 0



Brute force vs. smarter algorithm



Complexity of brute force



Number of Dimensions	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

In this talk

- Nonlinear optimization with continuous parameters
- Linear and nonlinear constraints
- Global optimization
- Diagnostics and strategies for difficult problems

Not Covered

- Linear programming
- Mixed integer programming
- Stochastic gradient descent

Introduction to `scipy.optimize`

Solve example problem with `scipy.optimize`

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

>>> def sphere(x):
...     return np.sum(x ** 2)

>>> x0 = np.ones(2)
>>> res = minimize(f, x0)
>>> res.fun
0.0
>>> res.x
array([0.0, 0.0])
```

Features of `scipy.optimize`

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

Practice Session 1: First optimization with `scipy.optimize` (15 min)

Shortcomings of `scipy.optimize`

- Very few algorithms
- No parallelization
- Maximization via sign flipping
- No diagnostics tools
- No feedback before optimization or in case of crash
- No built-in multistart, benchmarking, scaling, or logging
- Parameters are 1d numpy arrays

Examples from real projects I

```
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
```

Examples from real projects II

```
>>> 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

Introduction to estimagic

What is estimagic?

- Library for difficult numerical optimization
- Additional tools for nonlinear estimation
- Wraps many other optimizer libraries:
 - Scipy, Nlopt, TAO, Pygmo, ...
- Harmonized interface
- A lot of additional functionality

You can use it like scipy

```
>>> import estimagic as em

>>> def sphere(x):
...     return np.sum(x ** 2)

>>> res = em.minimize(
...     criterion=sphere,
...     params=np.arange(5),
...     algorithm="scipy_lbfgsb",
... )

>>> res.params
array([ 0., -0., -0., -0., -0.])
```

- No default algorithm
- Many optional arguments scipy does not have

Params can be (almost) anything

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

>>> res = em.minimize(
...     criterion=dict_sphere,
...     params={"a": 0, "b": 1, "c": pd.Series([2, 3, 4])},
...     algorithm="scipy_powell",
... )

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

- `params` can be a [pytree](#)
- includes (nested) dicts, lists, tuples or namedtuples containing numbers, arrays, Series and DataFrames
- Special case: DataFrame with columns "value" , "lower_bound" and "upper_bound"

OptimizeResult is very informative

```
>>> res = em.minimize(dict_sphere, params={"a": 0, "b": 1, "c": pd.Series([2, 3, 4])}, algorithm="scipy_neldermead")
>>> 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 <= 1e-10, **: change <= 1e-8, *: change <= 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.)

OptimizeResult has useful attributes

```
>>> res.criterion
.0
>>> res.n_criterion_evaluations
805
>>> res.success
True
>>> res.message
'Optimization terminated successfully.'
>>> res.history.keys():
dict_keys(['params', 'criterion', 'runtime'])
```

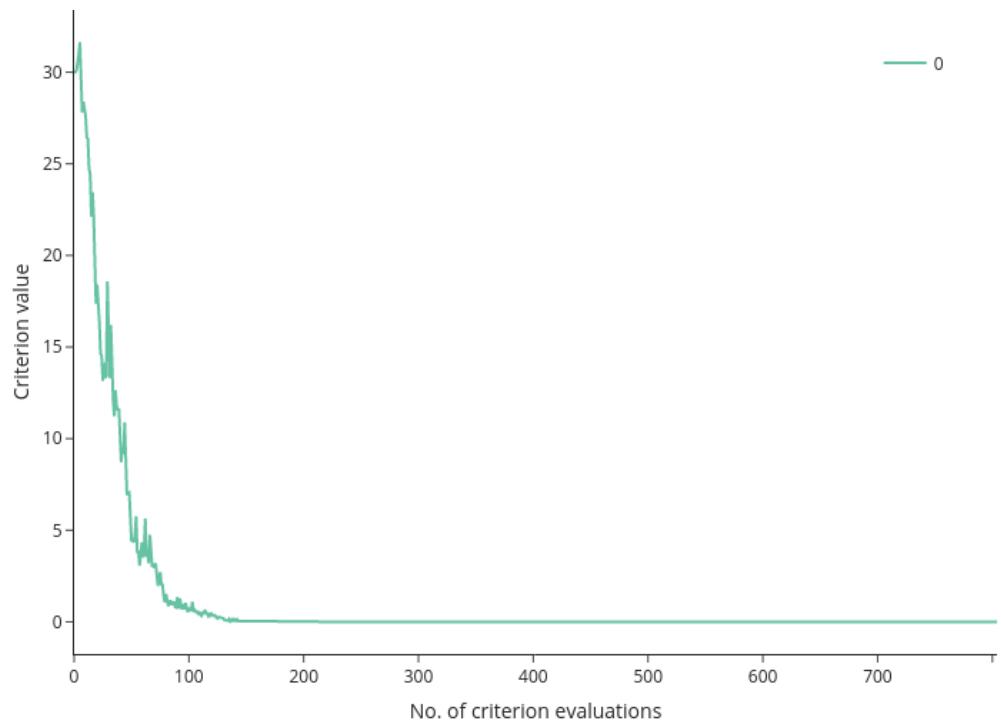
Logging and Dashboard

```
>>> res = em.minimize(  
...     criterion=sphere,  
...     params=np.arange(5),  
...     algorithm="scipy_lbfgsb",  
...     logging="my_log.db",  
...     log_options={  
...         "if_database_exists": "replace"  
...     },  
... )  
  
>>> 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 of parameters and criterion values
- Thread safe sqlite database
- No data loss, even if computer crashes
- Can be read during optimization
- Provides data for real-time dashboard

Criterion plot

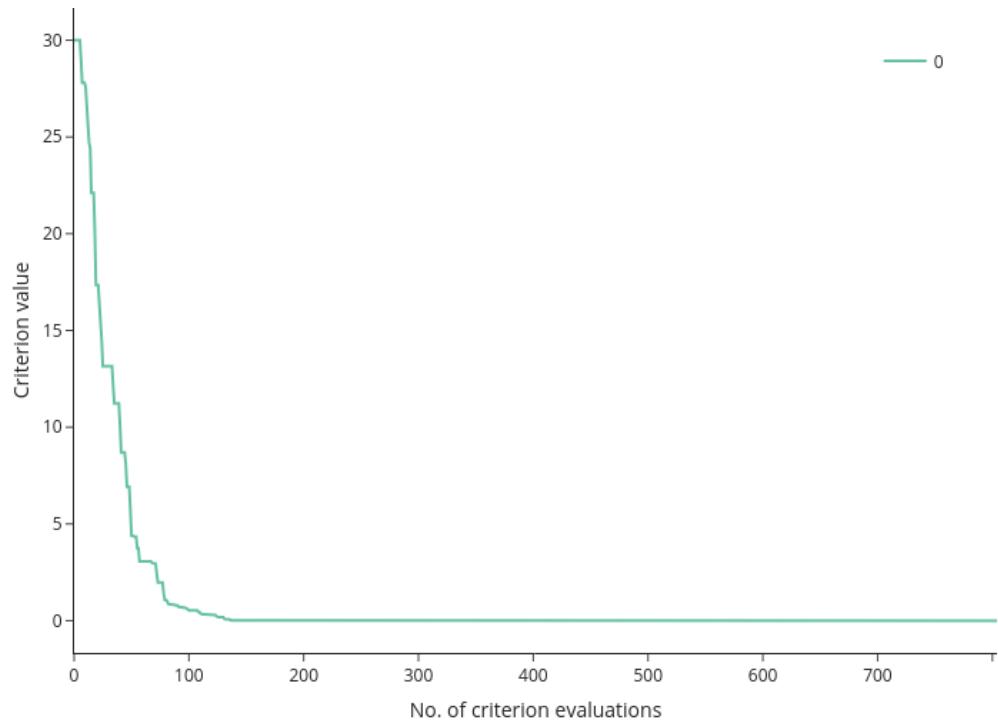
```
em.criterion_plot(res)
```



- First argument can be a OptimizeResult , path to log file or list/dict thereof

Criterion plot (2)

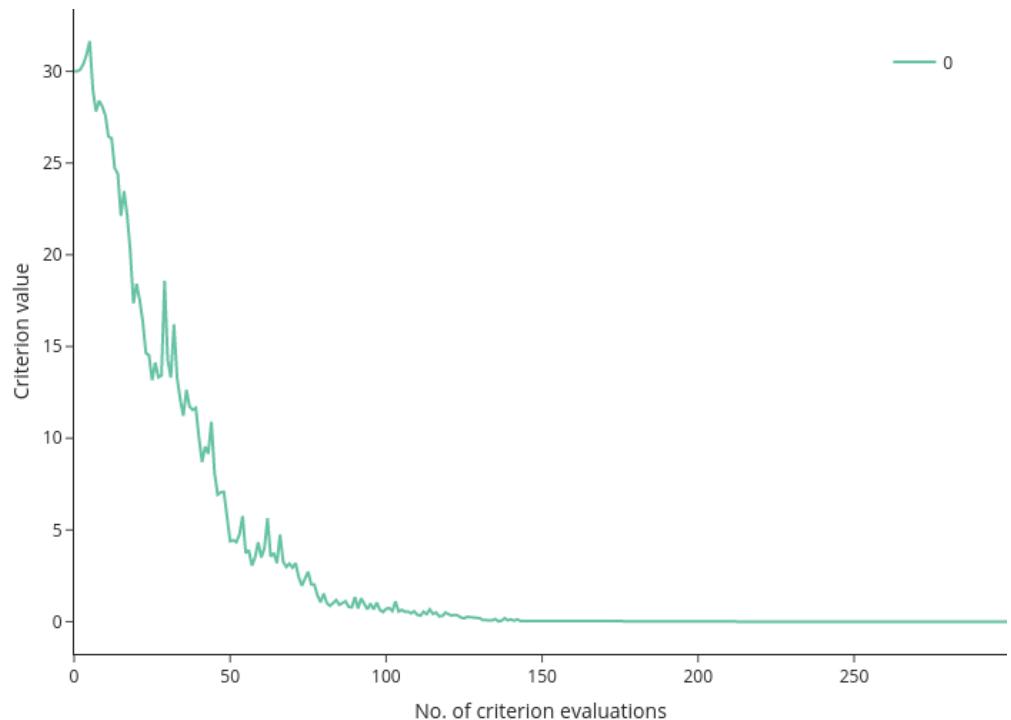
```
em.criterion_plot(res, monotone=True)
```



- First argument can be a `OptimizeResult`, path to log file or list/dict thereof
- `monotone=True` shows the current best value

Criterion plot (3)

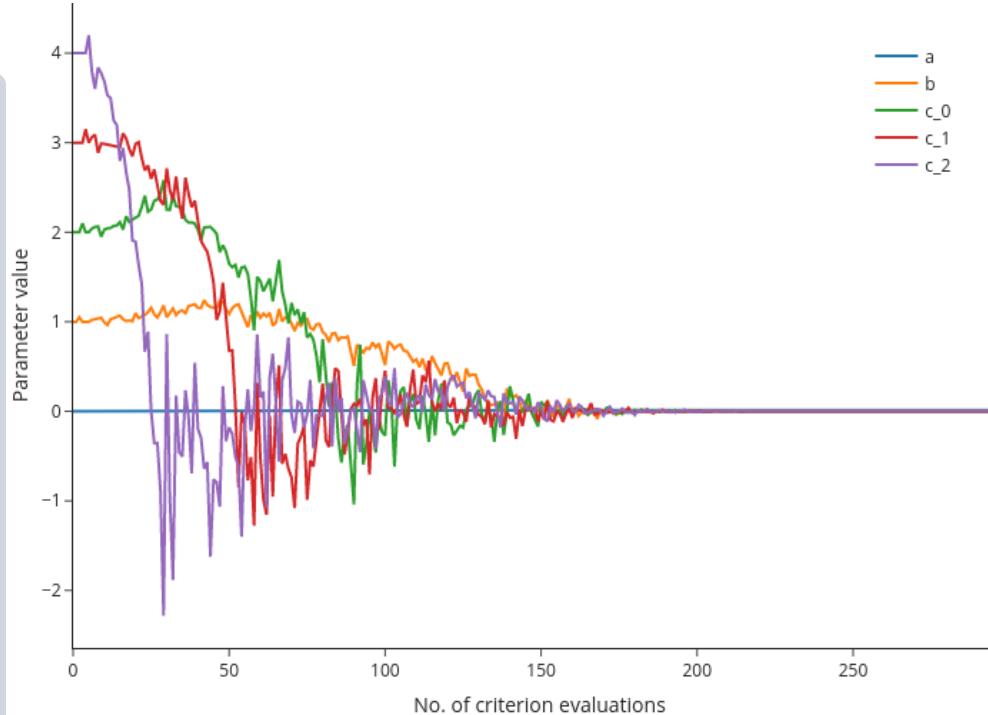
```
em.criterion_plot(res, max_evaluations=300)
```



- First argument can be a `OptimizeResult`, path to log file or list/dict thereof
 - `monotone=True` shows the current best value
 - `max_evaluations` sets upper limit of x-axis

Params plot

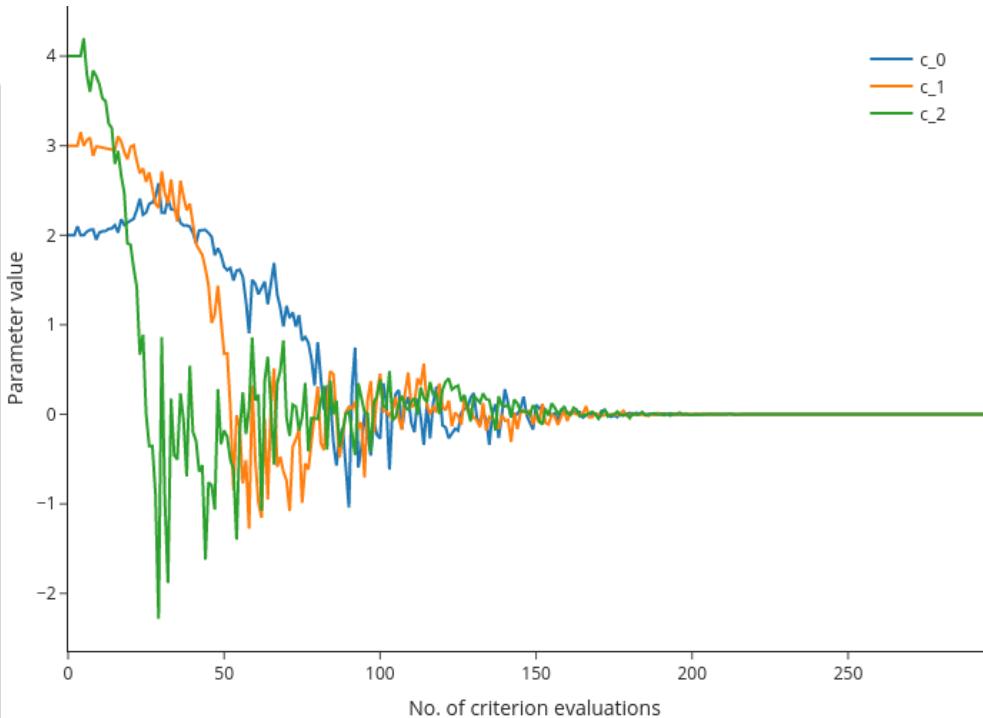
```
params = {
    "a": 0,
    "b": 1,
    "c": pd.Series([2, 3, 4])
}
em.params_plot(
    res,
    max_evaluations=300,
)
```



- Similar options as `criterion_plot`

Params plot (2)

```
params = {  
    "a": 0,  
    "b": 1,  
    "c": pd.Series([2, 3, 4])  
}  
em.params_plot(  
    res,  
    max_evaluations=300,  
    selector=lambda x: x["c"],  
)
```



- Similar options as `criterion_plot`
- `selector` is a function returning a subset of `params`

Unified interface to (collections of) algorithms

- **Nlopt**: Local and global algorithms by Stephen Johnson (MIT)
- **Pygmo**: Global algorithms by Francesco Biscani and Dario Izzo (ESA)
- **fides**: Pure Python algorithm by Fabian Fröhlich (Harvard)
- **TAO**: Toolkit for advanced optimization (Argonne national lab)
- **Cyipopt**: Python binding by Jason Moore to **IPOPT** algorithm by Andreas Wächter (Northwestern)
- **NAG**: Numerical algorithms group (Oxford)
- **estimagic**: Python implementations of some important algorithms
- **List not exhaustive and will add more soon (it is quite easy)**

Constraints via reparametrizations

```
>>> 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.])
```

- constraints is a list of dicts
- specify subset of parameters via loc , query or selector
- specify type of constraint
 - linear
 - probability
 - covariance
 - ...

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])}.
```

Closed-form or parallel numerical derivatives

```
>>> def sphere_gradient(params):
...     return 2 * params

>>> em.minimize(
...     criterion=sphere,
...     params=np.arange(5),
...     algorithm="scipy_lbfgsb",
...     derivative=sphere_gradient,
... )

>>> em.minimize(
...     criterion=sphere,
...     params=np.arange(5),
...     algorithm="scipy_lbfgsb",
...     numdiff_options={"n_cores": 6},
... )
```

- Numerical derivatives are calculated if user does not provide a derivative
- Efficient parallelization on (up to) as many cores as parameters

There is maximize

```
>>> def upside_down_sphere(params):
...     return -params @ params

>>> res = em.maximize(
...     criterion=upside_down_sphere,
...     params=np.arange(5),
...     algorithm="scipy_lbfgs",
... )
>>> res.params
array([ 0.,  0.,  0.,  0.,  0.])
```

Built-in multistart framework

```
>>> 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](#) by Fatih Guvenen and Serdar Ozkan
- Exploration phase on random sample
- Local optimizations from best points

Exploiting special structure of F

```
>>> def general_sphere(params):
...     contribs = params**2
...     out = {
...         "root_contributions": params,
...         "contributions": contribs,
...         "value": contribs.sum(),
...     }
...     return out

>>> res = em.minimize(
...     criterion=general_sphere,
...     params=np.arange(5),
...     algorithm="pounders",
... )
>>> res.params
array([0., 0., 0., 0., 0.])
```

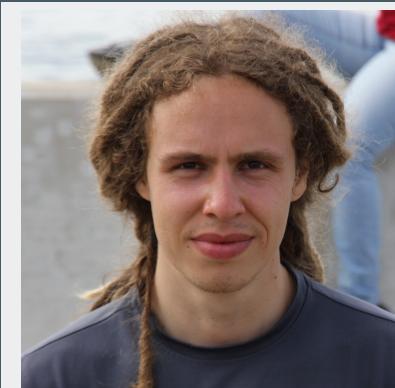
- Exploit structure of the problem
- Common structures
 - least-squares: $F(x) = \sum_i f_i(x)^2$
 - sum: $F(x) = \sum_i f_i(x)$, e.g. log-likelihood
- Huge speed-ups
- Increased robustness

Harmonized as much as possible but not more

```
>>> algo_options = {  
...     "convergence.relative_criterion_tolerance": 1e-9,  
...     "stopping.max_iterations": 100_000,  
...     "trustregion.initial_radius": 10.0,  
... }  
  
>>> res = em.minimize(  
...     criterion=sphere,  
...     params=np.arange(5),  
...     algorithm="nag_pybobyqa",  
...     algo_options=algo_options,  
... )  
>>> res.params  
array([0., 0., 0., 0., 0.])
```

- Same options have same name
- Different options have different names (e.g. not one tol argument)
- Ignore options that don't apply

The estimagic Team



Janos



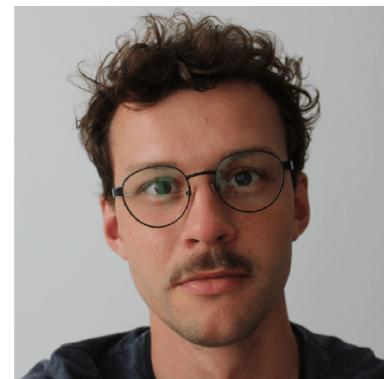
Tim



Klara



Sebastian



Tobias



Hans-Martin

Thanks to

- All [contributors](#) to estimagic
- [Kenneth Judd](#) for feedback and funding of a research visit
- [Gregor Reich](#) for feedback
- All authors of the amazing algorithms we are wrapping
- The University of Bonn and [TRA-1 Modelling](#) for funding
- [Collaborative Research Center Transregio 224](#) for funding

Break (5 min)

Practice Session 2: Convert previous example to estimagic (15 min)

Choosing algorithms

Relevant problem properties

- Smoothness: Differentiable? Kinks? Discontinuities? Stochastic?
- Convexity: Are there local optima?
- Size: 2 parameters? 10? 100? 1000? More?
- Constraints: Bounds? Linear constraints? Nonlinear constraints?
- Special structure: Nonlinear least-squares, Log-likelihood function
- Goal: Do you need a global solution? How precise?
- -> Properties guide the selection but do not replace benchmarking

`scipy_lbfgsb`

- Limited memory BFGS
- BFGS is a method to approximate hessians from multiple gradients
- Supports bounds
- Criterion must be differentiable
- Scales to a few thousand parameters
- Beats other BFGS implementations in many benchmarks
- Low overhead

fides

- Derivative based trust-region algorithm
- Supports bounds
- Developed by Fabian Fröhlich as a Python package
- Many advanced options to customize the optimization!
- Criterion must be differentiable
- Good solution if `scipy_lbfgsb` picks too extreme parameters that cause numerical overflow

nlopt_bobyqa , nag_pybobyqa

- **Bound Optimization by Quadratic Approximation**
- Derivative free trust region algorithm
- nlopt version has less overhead
- nag version has advanced options to deal with noise
- Good choice for non-differentiable but not too noisy functions
- Slower than derivative based methods but faster than neldermead

scipy_neldermead , nlopt_neldermead

- Popular direct search method
- nlopt version supports bounds
- nlopt version requires much fewer criterion evaluations in most benchmarks
- Never the best choice but often not the worst
- Can be very precise if run long enough

`scipy_ls_lm`, `scipy_ls_trf`

- Derivative based optimizers for least squares problems
- Criterion needs the structure: $F(x) = \sum_i f_i(x)^2$
- In estimagic, criterion function must return a dictionary:

```
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

nag_dfols , pounders

- Derivative free trust region method for nonlinear least-squares problems
- Both beat bobyqa for least-squares problems!
- `nag_dfols` is fastest and usually requires fewest criterion evaluations
- `nag_dfols` has advanced options to deal with noise
- `pounders` can do criterion evaluations in parallel

ipopt

- Interior point optimizer for problems with nonlinear constraints
- Probably the best open source optimizer for large constrained problems
- We wrap it via `cyipopt`
- Difficult to install on windows

Practice Session 3: Play with algorithm and algo_options (20 min)

What is benchmarking

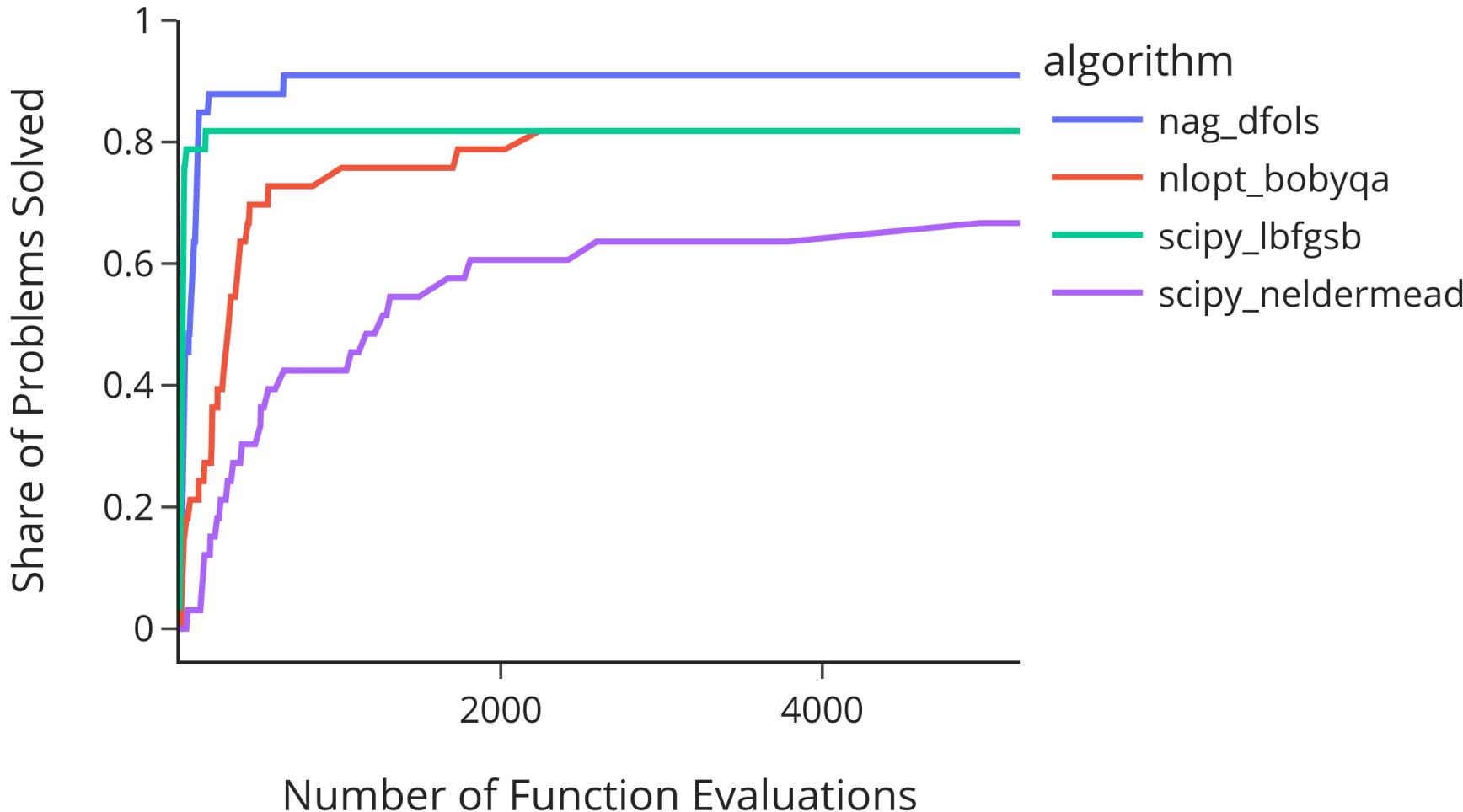
- Compare multiple algorithms on functions with known optimum
- Should mirror problems you actually want to solve
 - similar number of parameters
 - similar w.r.t. differentiability or noise
- Benchmark functions should be fast!
- Standardized sets of test problems and ways to visualize results

Running benchmarks in estimagic

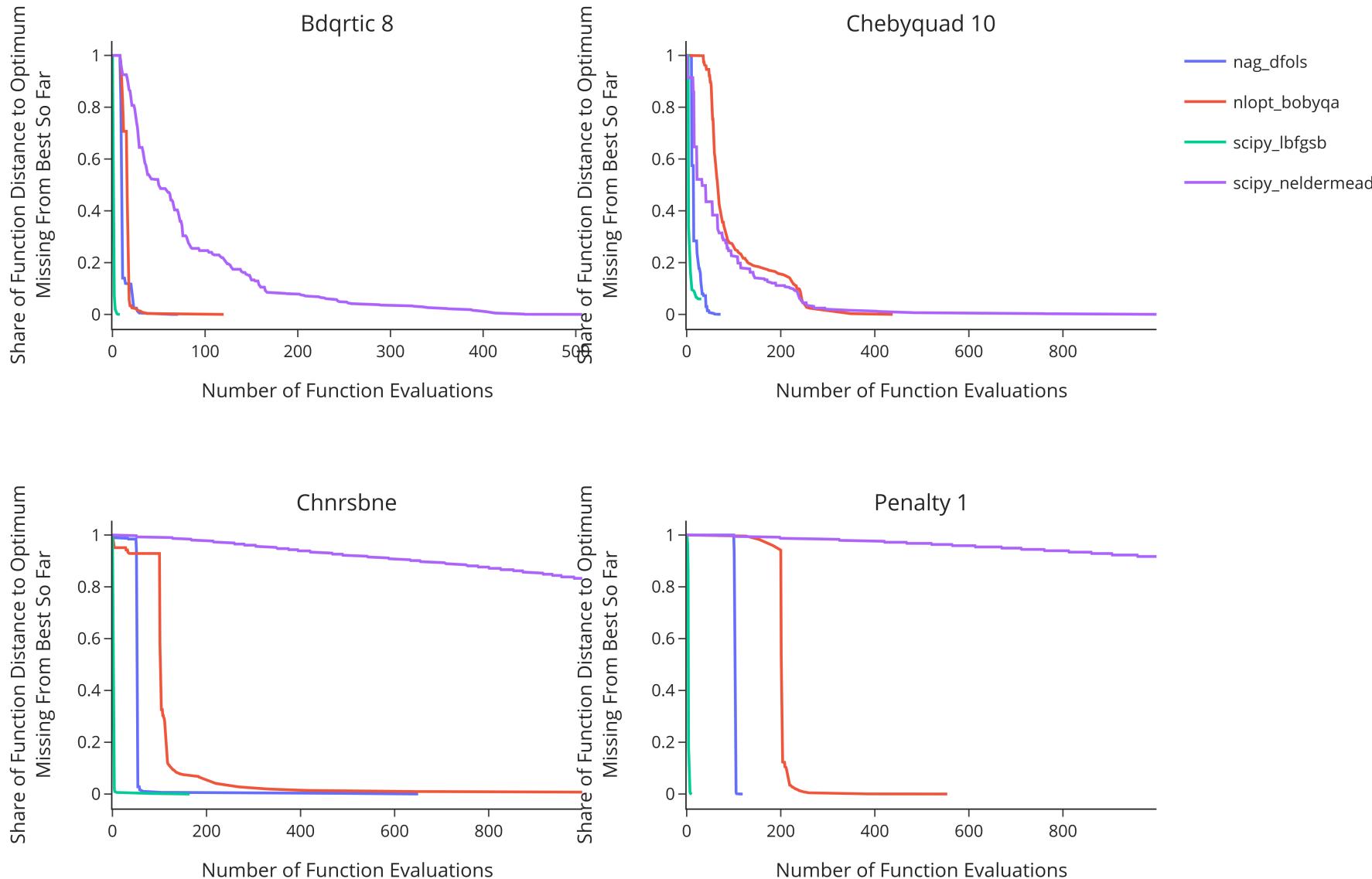
```
problems = em.get_benchmark_problems("estimagic")
optimizers = [
    "scipy_lbfgsb",
    "nag_dfols",
    "nlopt_bobyqa",
    "scipy_neldermead",
]
results = em.run_benchmark(
    problems=problems,
    optimize_options=optimizers,
    n_cores=4,
    max_criterion_evaluations=1000,
)
```

- Multiple sets of test problems
 - more_wild
 - estimagic
 - example
- Add noise or scaling problems
- Can pass additional options to govern minimization
- Benchmarks run in parallel

Profile plots



Convergence plots



Advanced options

```
problems = em.get_benchmark_problems(  
    name="example",  
    additive_noise=True,  
    additive_noise_options={  
        "distribution": "normal",  
        "std": 0.2,  
    },  
    scaling=True,  
    scaling_options={  
        "min_scale": 0.1,  
        "max_scale": 1000,  
    }  
)
```

- Add additive noise
- Add bad scaling
- This would be a very difficult set of problems

Practice Session 4: Benchmarking optimizers (10 min)

Break (10 min)

Advanced estimagic

Terminology of constraints in estimagic

- Constraints are conditions on parameters
- Can make problems simpler or more difficult
- bounds: $\min_x f(x)$ s.t. $l \leq x \leq u$
 - handled by most algorithms
- estimagic constraints:
 - handled by estimagic via reparametrization
- nonlinear constraints: $\min_x f(x)$ s.t. $c_1(x) = 0, c_2(x) \geq 0$
 - handled by some algorithms
 - can be violated during optimization

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_bounds": [1, 1, 1],  
...     "upper_bounds": [3, 3, 3],  
... },  
...     index=[ "a", "b", "c" ],  
... )  
  
>>> def criterion(p):  
...     return sphere(p.loc["value"])  
  
>>> 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

Reparametrization example

- Example: $f(x_1, x_2) = \sqrt{x_2 - x_1} + x_2^2$
- Only defined for $x_1 \leq x_2$
- Solve: $\min f(x_1, x_2)$ s.t. $x_1 \leq x_2$
- Not a simple bound!
- Reparametrization approach:
 - Define $\tilde{x}_2 = x_2 - x_1$ and $\tilde{f}(x_1, \tilde{x}_2) = \sqrt{\tilde{x}_2} + (x_1 + \tilde{x}_2)^2$
 - Solve: $\min \tilde{f}(x_1, \tilde{x}_2)$ s.t. $\tilde{x}_2 \geq 0$
 - Translate solution back into x_1 and x_2

Which constraints can be handled via reparametrization?

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

Do not try at home

- Easy to make mistakes when implementing this
 - forget to transform start parameters
 - forget to translate back
 - forget to adjust derivative
 - confuse directions
 - use non-differentiable transformations
- **Estimagic does reparametrizations for you!**
- **Completely hides transformed x**

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
- Other selection methods will be explained later

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.])
```

- Imposes 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

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 parameters to 1
- Only works 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

- The "loc" field can be replaced other things for some params formats
- If params is a DataFrame with "value" column
 - "query" : An arbitrary query string that selects the relevant rows
 - "loc" : Will be passed to DataFrame.loc
- With all params formats
 - "selector" : A python function that takes params as arguments and returns a subset of params

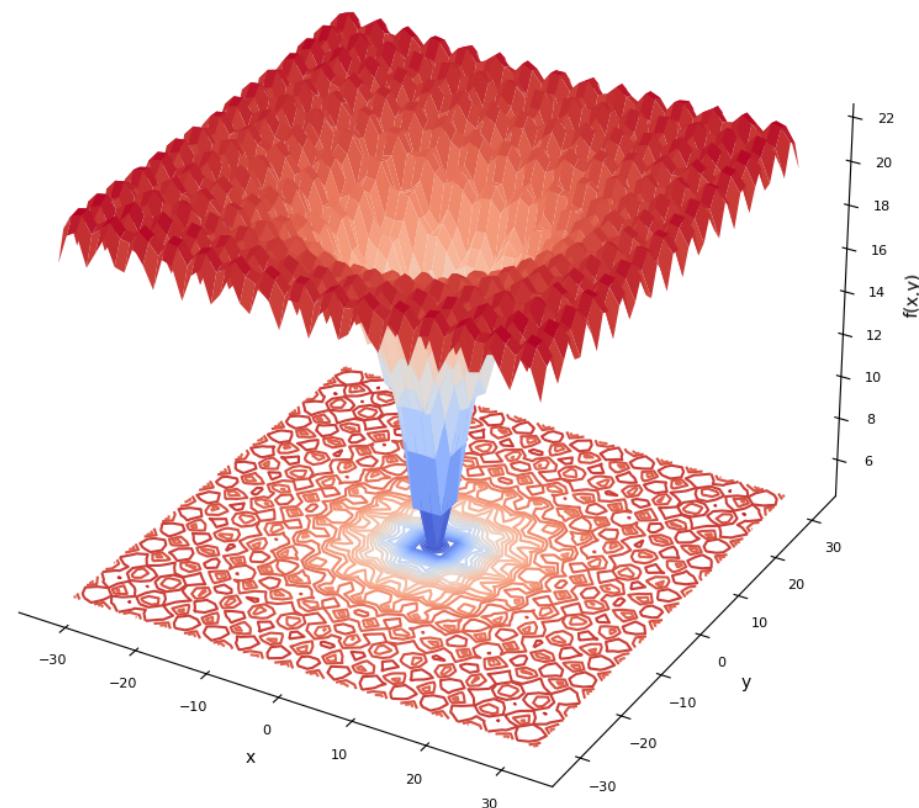
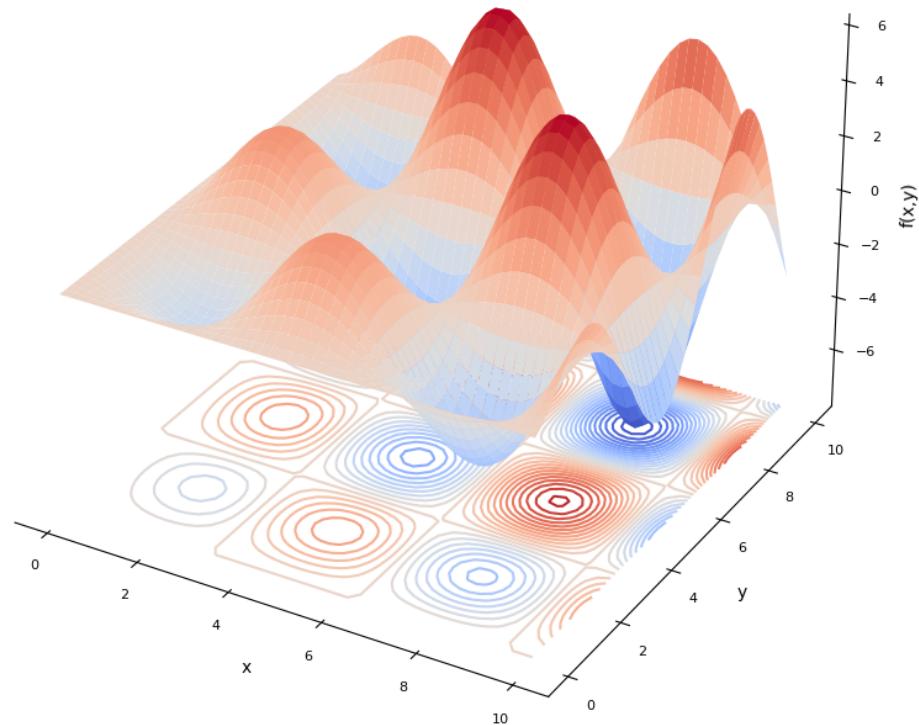
Practice Session 5: Constrained optimization (10 min)

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
- Global and local optimization are the same for convex problems

Examples



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 (in estimagic)

- Inspired by [tiktak algorithm](#) by Fatih Guvenen and Serdar Ozkan
- 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 was rediscovered several times
- Use any estimagic algorithm for local optimization
- Can distinguish soft and hard bounds

How to choose

- Extremely expensive criterion (i.e. can only do a few evaluations):
 - Bayesian optimization
- Differentiable function or least-squares structure and not too many local optima:
 - Multistart with a 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

Advanced multistart

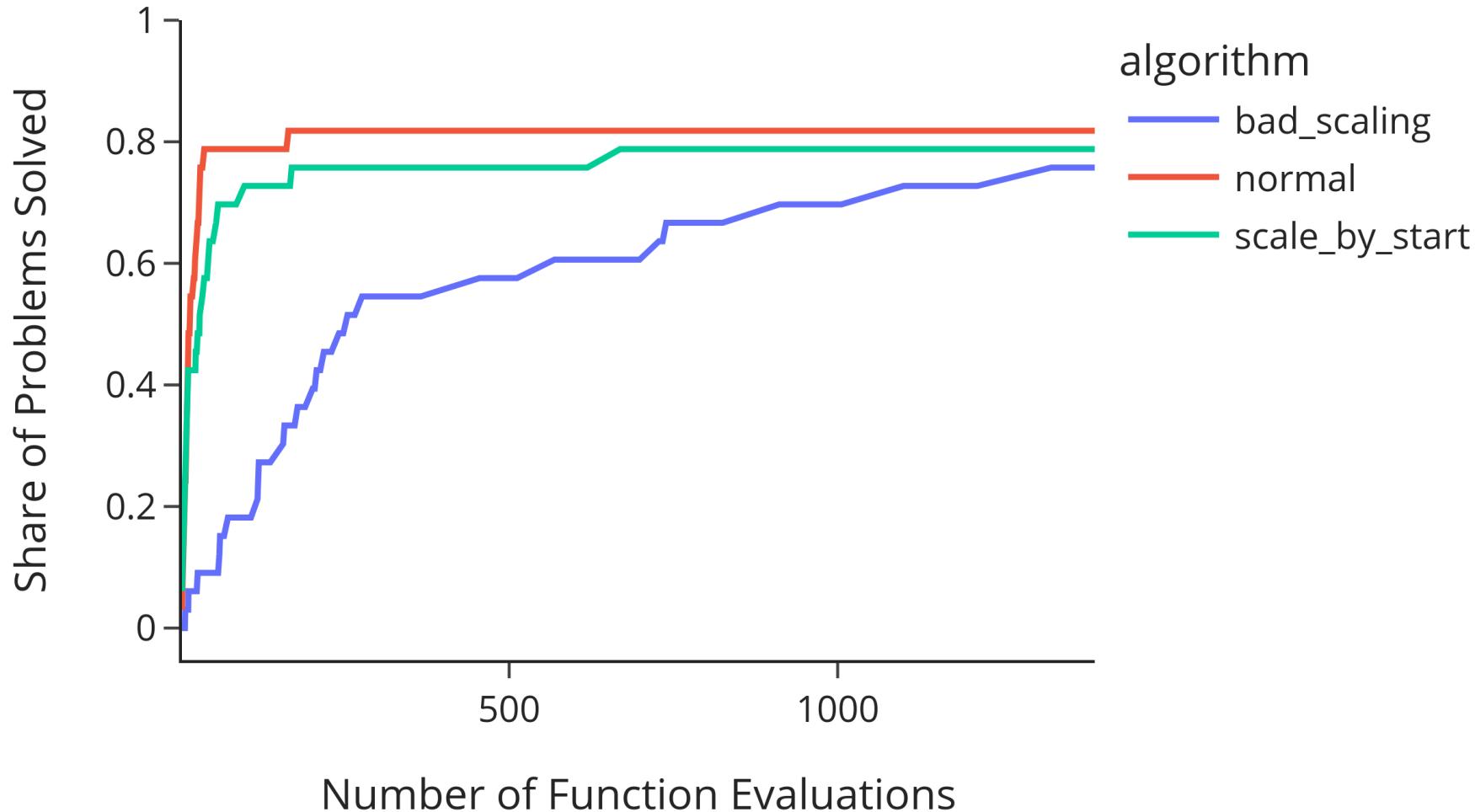
- Many local optima:
 - Large "n_sample"
 - Low "share_optimizations"
 - Weak convergence criteria
 - Refine result with stricter convergence criteria
- Few local optima:
 - Stick with defaults

Scaling

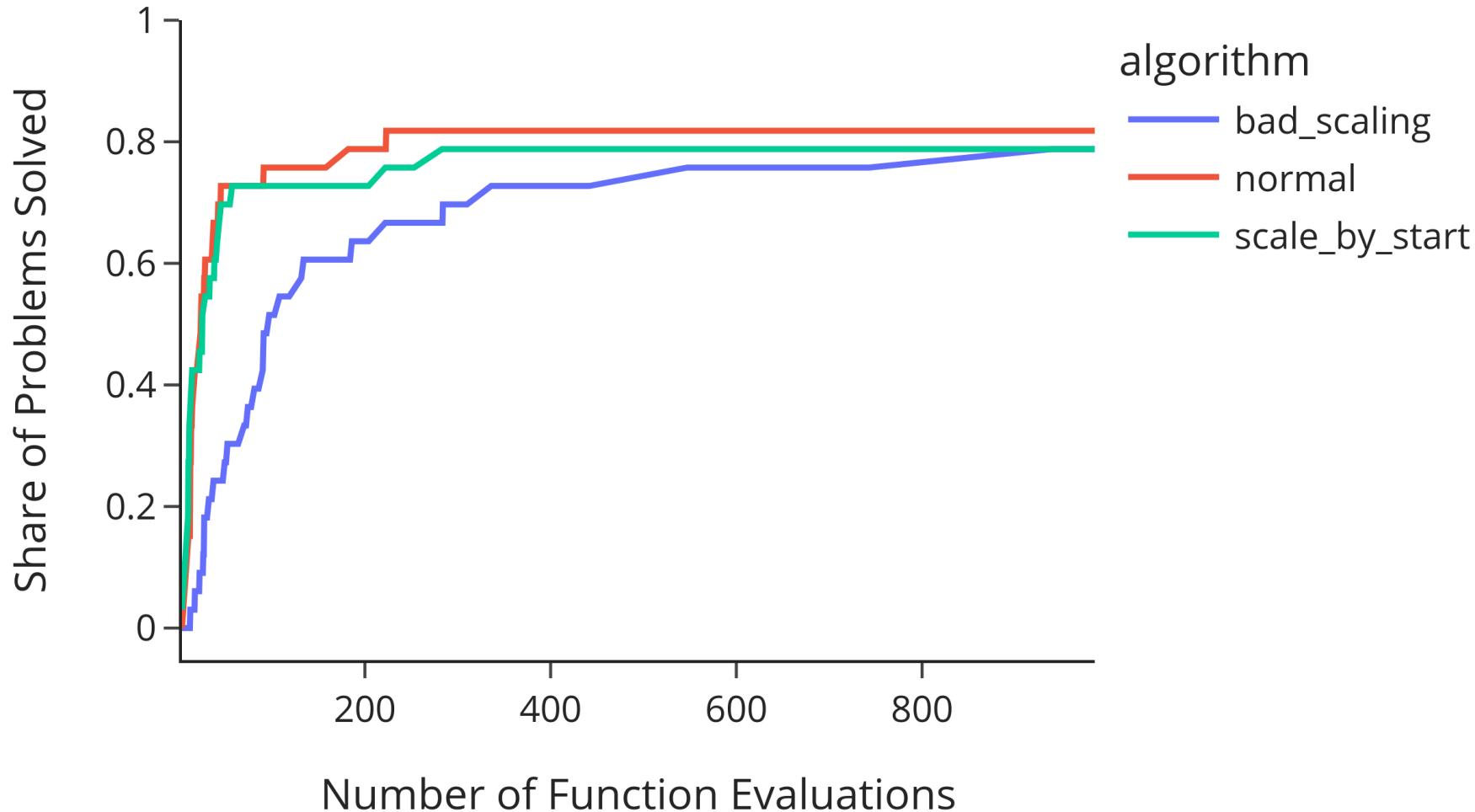
What is scaling

- Single most underrated topic among economists who do optimization!
- Well scaled: A fixed step in any parameter dimension yields roughly comparable changes in function value
 - $f(x_1, x_2) = 0.5x_1 + 0.8x_2$
- Badly scaled: Some parameters are much more influential
 - $f(x_1, x_2) = 1000x_1 + 0.2x_2$
 - $f(x_1, x_2) = e^{x_1} + \sqrt{x_2}$
- Often arises when parameters have very different units

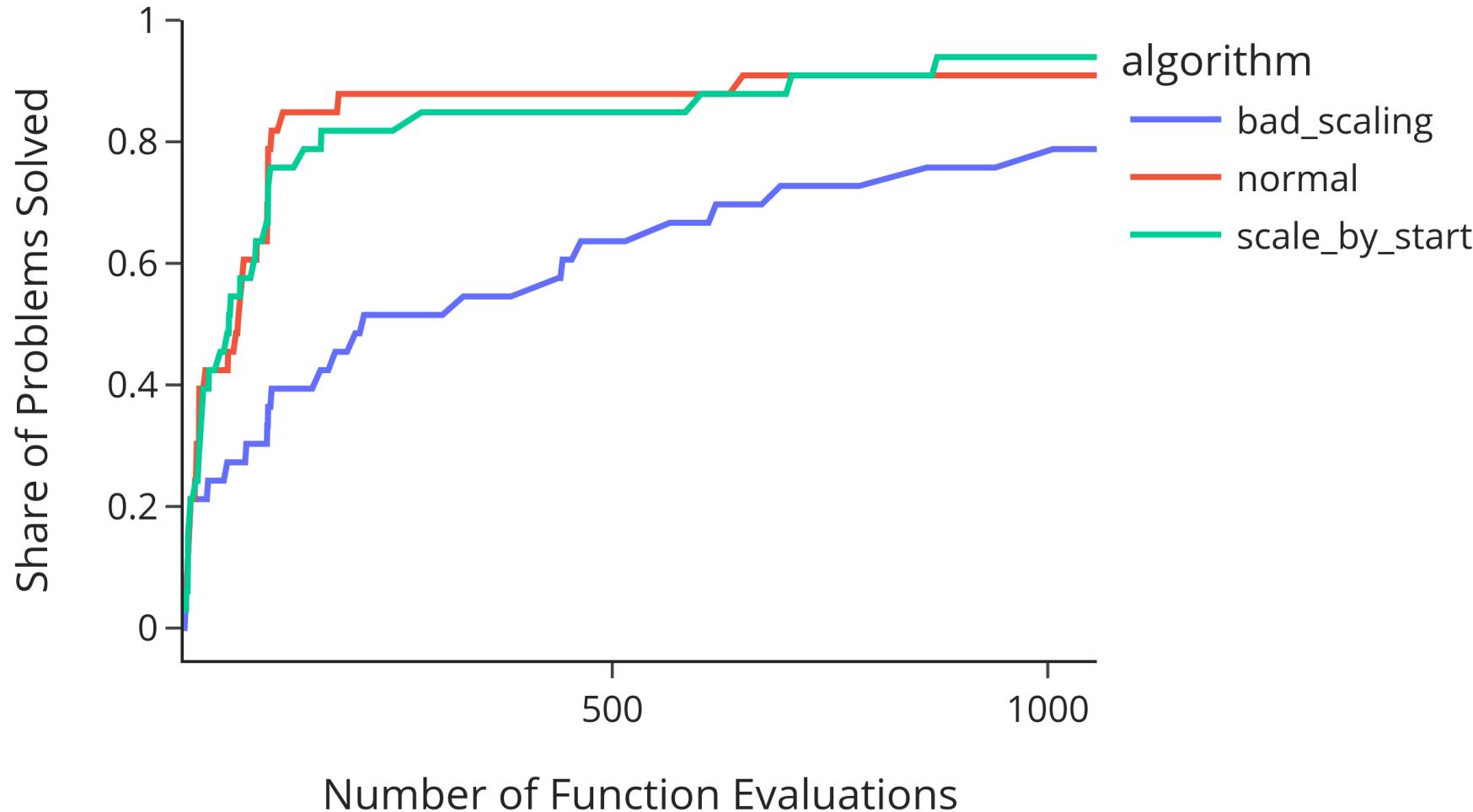
Effect of bad scaling: `scipy_lbfgsb`



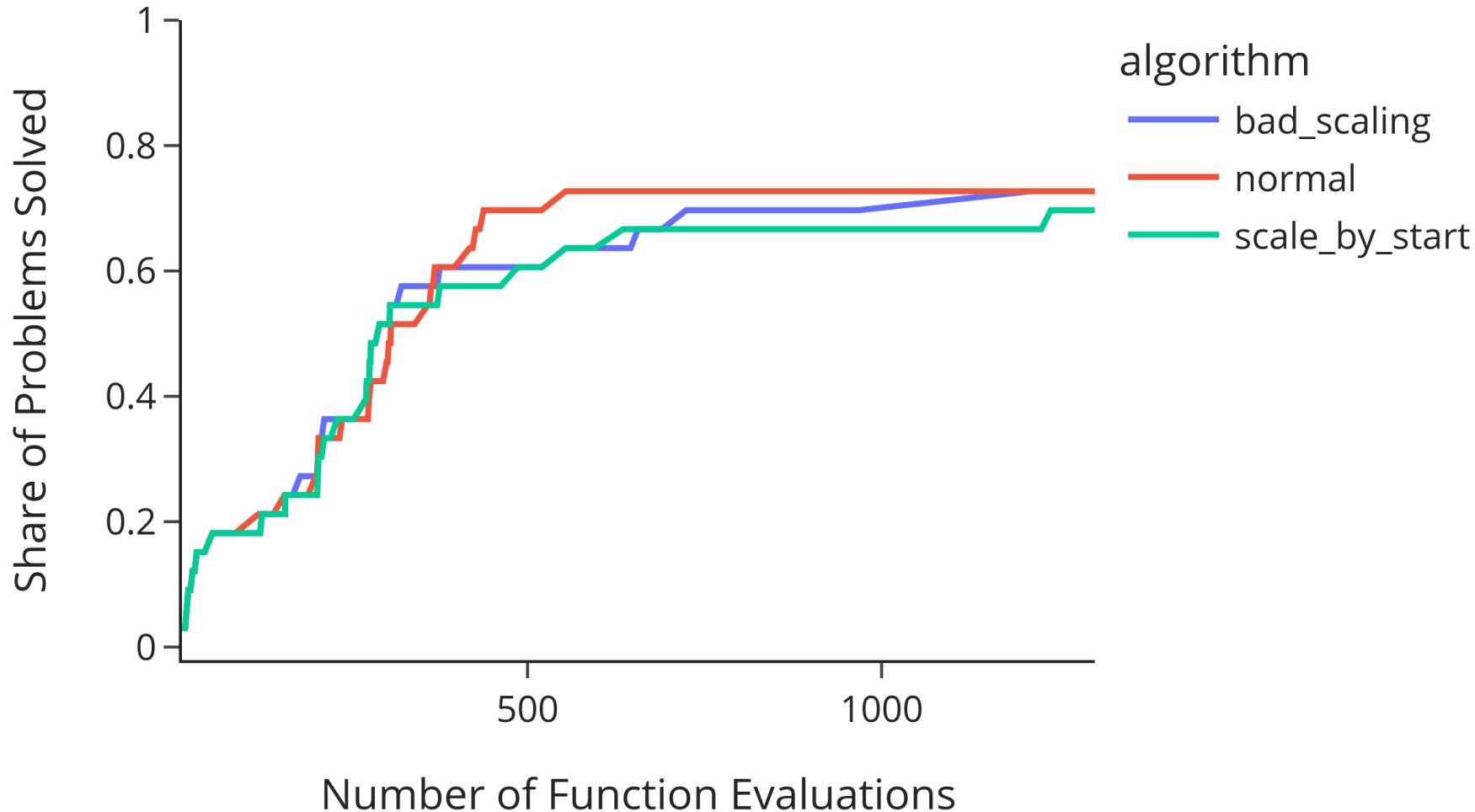
Effect of bad scaling: fides



Effect of bad scaling: nag_dfols



Effect of bad scaling: nlopt_bobyqa



Scaling in estimagic: By start params

```
def badly_scaled(x):
    return 0.01 * x[0] + x[1] + x[2] ** 6

em.minimize(
    criterion=badly_scaled,
    params=np.array([200, 1, 1]),
    scaling=True,
    # pick default method explicitly
    scaling_options={
        "method": "start_values",
    }
)
```

- Optimizer sees x / x_{start}
- Works without bounds
- Will recover that $x[0]$ needs large steps
- Won't recover that $x[2]$ needs tiny steps

Scaling in estimagic: By bounds

```
em.minimize(  
    criterion=badly_scaled,  
    params=np.array([200, 1, 1]),  
    scaling=True,  
    lower_bounds=np.array([-200, 0, 0.9]),  
    upper_bounds=np.array([500, 2, 1.1]),  
    scaling=True,  
    scaling_options={"method": "bounds"},  
)
```

- Internal parameter space mapped to $[0, 1]^n$
- Will work great in this case
- Requires careful specification of bounds

Very scale sensitive

- nag_pybobyqa
- tao_pounders
- pounders
- nag_dfols
- scipy_cobyla

Not scale sensitive

- scipy_neldermead
- nlopt_neldermead
- nlopt_bobyqa
- scipy_powell
- scipy_ls_lm
- scipy_ls_trf

Somewhat scale sensitive

- scipy_lbfgsb
- fides

Practice Session 6: Scaling of optimization problems (10 min)

Documentation of estimagic

- estimagic.readthedocs.io
- **Getting started:** Short tutorials
- **How to guides:** Explanations and examples for each argument of `maximize` and `minimize`
- **API Reference:** Interface of all public functions
- **Explanations:** Background information, tips and tricks, best practices

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  on
github.com/OpenSourceEconomics/estimagic

Numerical vs automatic differentiation (oversimplified)

- **Automatic differentiation**

- Magic way to calculate precise derivatives of Python functions
- Gradient calculation takes about 3 times longer than function
- Runtime independent of parameters
- Code must be written in a certain way

- **Numerical differentiation**

- Finite step approximation to derivatives
- Less precise than automatic differentiation
- Runtime increases linearly with number of parameters

What is JAX

- GPU accelerated replacement for Numpy
- State of the art automatic differentiation
 - Gradients
 - Jacobians
 - Hessians
- Just in time compiler for python code
- Composable function transformations such as `vmap`
- It's a life changing library!

Calculating derivatives with JAX

```
>>> import jax.numpy as jnp
>>> import jax

>>> def sphere(x):
...     return jnp.sum(x ** 2)

>>> gradient = jax.grad(sphere)

>>> gradient(jnp.array([1., 1.5, 2.]))
DeviceArray([2., 3., 4.], dtype=float32)
```

Practice Session 7: Using JAX derivatives in estimagic (10 min)

What is JAXopt

- Library of optimizers written in JAX
- Hardware accelerated
- Batchable
- Differentiable

When to use it

- Simple optimization problems
 - But many
- Robustness to hyper-parameters

Simple optimization in JAXopt

```
>>> import jax
>>> import jax.numpy as jnp
>>> from jaxopt import LBFGS

>>> x0 = jnp.array([1.0, 2, 3])
>>> shift = x0.copy()

>>> def criterion(x, shift):
...     return jnp.vdot(x, x + shift)

>>> solver = LBFGS(fun=criterion)

>>> result = solver.run(init_params=x0, shift=shift)
>>> result.params
DeviceArray([ 0. , -0.5, -1. ], dtype=float64)
```

- import solver
- initialize solver with criterion
- run solver with starting parameters
- pass additional arguments of criterion to run method

Vmap in JAX

Vectorize an optimization in JAXopt

```
>>> from jax import jit, vmap

>>> def solve(x, shift):
...     return solver.run(init_params=x, shift=shift).params

>>> batch_solve = jit(vmap(solve, in_axes=(None, 0)))
>>> weights = jnp.array([
    [0.0, 1.0, 2.0],
    [3.0, 4.0, 5.0]
])
>>> batch_solve(x0, weights)
DeviceArray([[ 0. , -0.5, -1. ],
            [-1.5, -2. , -2.5]], dtype=float64)
```

- import jit and vmap
- define wrapper around solve
- call vmap on wrapper
 - in_axes=(None, 0) means that we map over the 0-axis of the second argument
 - call jit at the end

Differentiate a solution in JAXopt

```
>>> from jax import jacobian

>>> jacobian(solve, argnums=1)(x0, weight)
DeviceArray([[-0.5,  0.,  0.],
            [ 0., -0.5,  0.],
            [ 0.,  0., -0.5]], dtype=float64)

>>> solve(x0, weight)
DeviceArray([ 0., -0.5, -1.], dtype=float64)

>>> solve(x0, weight + 1)
DeviceArray([-0.5, -1., -1.5], dtype=float64)
```

- import jacobian or grad
- use argnums to specify by which argument we differentiate

Practice Session 8: Vectorized optimization in JAXopt (15 min)