Estimagic

Janoś Gabler and Estimagic Team

November 18, 2021

What is numerical optimization

- Optimization problem is defined by:
  - Criterion function
  - Sometimes constraints
  - Initial guess for optimal parameters
- Try a bunch of parameter vectors and take the best
- No universally best way to select the parameters to try

Not so good workflows for optimization

- Pick a random optimizer package, run default algorithm from random start values and pray
- Run massively parallel but brute algorithm and ignore curse of dimensionality
- Copy Nelder Mead from numerical recipes
- Always just use an algorithm than once worked well on a completely different problem

Better workflow for numerical optimization

- Try to make your problem simpler (scaling, derivatives, smoothing, speedup, reparametrization)
- Select promising algorithms (theory + problem properties)
- Benchmark several algorithms and tuning parameters on similar problems
- Run best algorithm from multiple starting points
- Plot and check the convergence histories

What is estimagic – High level

- Estimate likelihood models
- Estimate Method of Simulated Moments models
- Bootstrapping

What is estimagic – Low level

- Optimization
  - Huge collection of high quality optimizers
  - Same interface, even for different classes of optimizers
  - Constraints implemented via differentiable reparametrizations
  - Built in benchmark suites and multistart
  - Scaling of optimization problems
- Differentiation
  - Finite differences + Richardson extrapolation
  - Visualization of function evaluations
  - Parallel function evaluations

Vision

- Become No. 1 Python package for:
  - (Difficult) nonlinear optimization
  - Numerical differentiation
  - Inference for custom estimation functions
- Add own optimizers where we see the need
- Optimization researchers use estimagic for benchmarking and add their algorithms

Contributors

(a) Janos Gabler

(b) Klara Röhrl

(c) Tobias Raabe

(d) Tim Mensinger

(e) Sebastian Gsell

As well as Hans-Martin von Gaudecker, Mariam Petrosyan, Moritz Mendel, Max Blesch, Christian Zimpelmann, Robin Musolff, Annica Gehlen, Sofia Badini, Sofya Akimova, Xuefei Han, Leiqiong Wan, Andrew Souther, Luis Calderon, Linda Maokomatanda, Madhurima Chandra and Vijaybabu Gangaprasad

This talk

1. Estimagic overview
2. Estimagic for advanced optimizations

- Many examples
- Each with link to documentation notebook
- Use slides as cheat-sheet!

Estimagic overview

Installation

- Install conda: `anaconda.com/products/individual#Downloads`
- Execute the following in a terminal:
  - **`conda config -add channels conda-forge`**
  - **`conda install -c opensourceeconomics estimagic`**
- Optionally, install more optimizers
  - **`pip install Py-BOBYQA`**
  - **`pip install DFO-LS`**
  - **`conda install petsc4py`**

Define a criterion function

```python
def sphere(params):
    out = {
        "value": (params["value"] ** 2).sum(),
        "contributions": params["value"] ** 2,
        "root_contributions": params["value"],
    }
    return out

def simple_sphere(params):
  return (params["value"] ** 2).sum()
```

- Dict version is needed to switch between standard optimizers, BHHH and least squares optimizers
- Simple version is enough for most cases
- Details: `tinyurl.com/firstoptimization`

Define start parameters and bounds

```python
import pandas as pd

start_params = pd.DataFrame(
    data=[[1, -5, 10], [2, -5, 10], [3, -5, 10]],
    columns=["value", "lower_bound", "upper_bound"],
    index=["alpha", "beta", "gamma"]
)
```

	value	lower_bound	upper_bound
alpha	1	-5	10
beta	2	-5	10
gamma	3	-5	10

- Can use arbitrary single or multi-index
- Can have additional columns
- Encourages label based parameter handling
- Details: `tinyurl.com/startparams`

Run an optimization

```
from estimagic import minimize

res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
)
res["solution_params"]
```

	value	lower_bound	upper_bound
alpha	0.0	-5.0	10.0
beta	0.0	-5.0	10.0
gamma	-0.0	-5.0	10.0

- Also have **`maximize`**
- Result is a self explanatory dictionary that can be easily saved and inspected
  - Solution parameters
  - Solution criterion value
  - ...
- Details:

  `tinyurl.com/firstoptimization`

Calculate numerical derivatives

```python
from estimagic import first_derivative

res = first_derivative(
    func=simple_sphere,
    params=start_params,
    # from here optional
    method="central",
    n_steps=2,
    n_cores=4,
    scaling_factor=2.5,
    error_handling="continue",
)
```

- All but first two optional
- Many more options for step configuration
- Result is dictionary:
  - Actual derivative
  - Function evaluations
  - ...
- Function that returns dict also works
- Bounds are respected

Estimating models

- Maximum Likelihood:`tinyurl.com/mlestimation`
- Method of Simulated Moments: `tinyurl.com/msmestimation`

Estimagic for advanced optimizations

List of optimizers

- Local, global and least squares optimizers from scipy
- Genetic optimizers from pygmo
- Local and global algorithms from nlopt
- Fides
- Ipopt
- POUNDERS (from TAO)
- DF-OLS and Py-BOBYQA
- Details: `tinyurl.com/listofalgorithms`

Set algorithm and algo_options

```
options = {
    "convergence.relative_params_tolerance": 1e-6,
    "stopping.max_criterion_evaluations": 10_000,
    "trustregion.initial_radius": 1,
}

res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    constraints=constraints,
    algo_options=options
)
res["solution_params"]
```

- Options are harmonized where possible
- Options that do not apply are ignored (with warning)
- Prefixes group the options by topic
- Default values: `tinyurl.com/optiondefaults`

Add constraints

```
constraints = [{"loc": "alpha", "type": "fixed", "value": 1}]
```

```
res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    constraints=constraints,
)
res["solution_params"]
```

	value	lower_bound	upper_bound
alpha	1.0	-5.0	10.0
beta	0.0	-5.0	10.0
gamma	-0.0	-5.0	10.0

- Linear constraints
- Covariance and sdcorr
- Equality and pairwise equality
- Probabilities
- Increasing and decreasing
- Details: `tinyurl.com/addconstraints`
- Implementation: `tinyurl.com/reparametrizations`

Add a derivative

```python
def sphere_derivative(params):
    x = params["value"].to_numpy()
    out = {
        "value": 2 * x,
        "contributions": np.diag(2 * x),
        "root_contributions": np.eye(len(x)),
    }
    return out

def simple_sphere_derivative(params):
    return 2 * params["value"]

res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    derivative=sphere_derivative,
)
```

- Simple version enough for most optimizers
- Dict version allows easy switch between all optimizers
- Derivative is ignored by gradient free optimizers
- Fully compatible with reparametrizations (thanks to Tim)

Rescale your problem

```
options = {
    "method": "start_values",
    "clipping_value": 0.1
}

res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    scaling=True,
    # optional
    scaling_options=options,
)
```

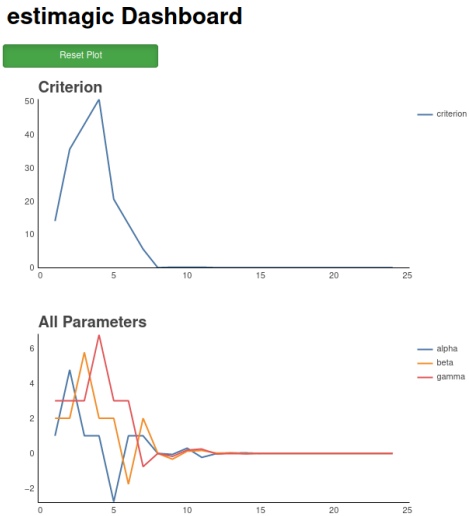
- Scaling is very important for derivative free optimizers
- Available methods:
  - Divide by absolute start values
  - Divide by gradient at start values
  - Rescale bounds to $[0, 1]^n$
- Details: `tinyurl.com/problemscaling`

Use logging

```
res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    logging="my_log.db",
)
```

- Evaluations, parameters, exceptions and more stored in sqlite database
- Logging is thread safe
- Convenient functions to read log

The dashboard

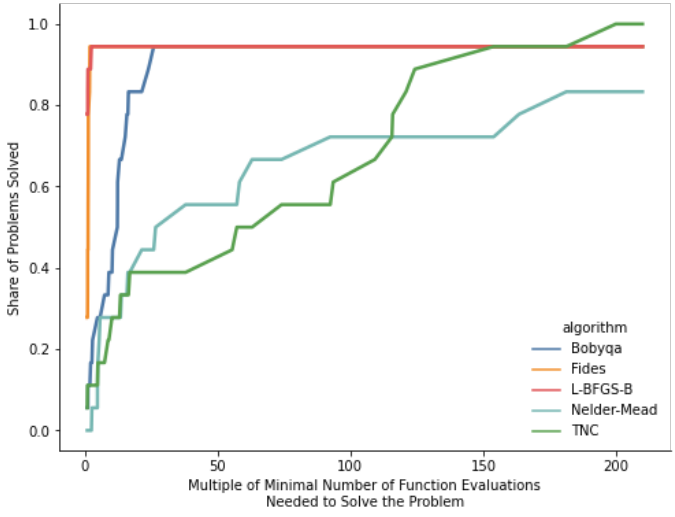


- **`estimagic dashboard my_log.db`**
- Options to
  - Skip entries
  - Jump to end or replay
  - Modify update frequency
  - . . .
- Grouping of parameters can be configured

Run optimizer benchmark

```python
from estimagic.examples.benchmarking import get_problems
from estimagic.examples.benchmarking import run_benchmark
from estimagic.visualization.profile_plot import profile_plot

problems = get_problems("more_wild")

optimizers = {
    "L-BFGS-B": {"algorithm": "scipy_lbfgsb"},
    "Nelder-Mead": {
        "algorithm": "scipy_neldermead",
        "algo_options": {"stopping_max_iterations": 1000},
    },
    "Fides": {"algorithm": "fides"},
    "Bobyqa": {
        "algorithm": "nlopt_bobyqa",
        "algo_options": {"stopping_max_criterion_evaluations": 1000},
    },
    "TNC": {
        "algorithm": "scipy_truncated_newton",
        "algo_options": {"stopping_max_criterion_evaluations": 500},
    },
}

results = run_benchmark(
    problems,
    optimizers,
    logging_directory="benchmark_logs",
)

fig, ax = profile_plot(
    problems=problems,
    results=results,
    runtime_measure="n_evaluations",
    y_precision=1e-3,
)
```

Plot benchmark results



Do multistart optimizations

```
options = {
    "n_samples": 100,
    "share_optimizations": 0.1,
    "n_cores": 12,
    "sampling_method": "sobol",
    "seed": 1234,
}

res = minimize(
    criterion=sphere,
    params=start_params,
    algorithm="scipy_lbfgsb",
    multistart=True,
    # optional
    multistart_options=options,
)
```

- Inspired by TikTak algorithm
- Exploration on random points
- Local optimizations from convex combination of current best and next best random point
- Convergence if current optimum is rediscovered multiple times
- Details: `tinyurl.com/multistartopt`

If you want to support us

- Give us a ☆ on GitHub
- Use estimagic and cite it
- Suggest new optimizers
- Improve the documentation
- Make Pull Requests