Tranquilo

A trustregion optimizer for economists by economists

Janoś Gabler, University of Bonn

Sebastian Gsell, LMU Munich

Tim Mensinger, University of Bonn

Mariam Petrosyan, University of Bonn

Prototypical optimization problem

- Discrete choice dynamic programming model
- Fit simulated choices to data
- Backwards induction is hard to parallelize
- Simulated choices are noisy
- 10 to 50 parameters
- Each simulation run takes a few minutes

Goals for an optimizer

- Robust to noise
- Parallel function evaluations
- Suitable for data fitting problems
- Designed for non-expert users
- Assumption: Criterion function is expensive!

Optimization Problems

Scalar Derministic

$\min_{l \leq x \leq u} F(x)$

Least-squares Deterministic

$\min_{l \leq x \leq u} F(x) = \sum_i f_i(x)^2$

Scalar Noisy

$\min_{l \leq x \leq u} \mathbb{E} F(x, \epsilon)$

Least-squares Noisy

$\min_{l \leq x \leq u} \mathbb{E} F(x, \epsilon) = \mathbb{E} \sum_i f_i(x, \epsilon_i)^2$

Existing optimizers

	Nelder-Mead	Bobyqa	PyBobyqa	DFO-LS	POUNDERS	Parallel NM
Library	Nlopt	Nlopt	NAG	NAG	TAO	(estimagic)
Class	simplex	trustregion	trustregion	trustregion	trustregion	simplex
Noisy	(yes)	no	yes	yes	no	(yes)
Parallel	no	no	(yes)	(yes)	no	yes
Least-squares	no	no	no	yes	yes	no

Recap: Trustregion optimizers

Derivative free trustregion optimization

- Define a region around x_k
- Maintain a sample of xs and corresponding function evaluations
- Fit a regression or interpolation model on the sample
- Optimize the surrogate model to create a candidate
- Evaluate the function at the candidate
- Accept or reject and adjust radius

Model quality, Rho, and Radius

F: criterion function

k: iteration counter

x_k: current x

M_k: surrogate model

s_k: candidate step

$\rho = \frac{F(x_k)-F(x_k+s_k)}{M_k(x_k)-M_k(x_k+s_k)}$

- Goal:
  - Sample few new points
  - Make large progress
- Model does not have to be great!
- Taylor like error bounds on M_k
  - Small ρ: decrease radius
  - Large ρ: increase radius
- Preview: This will fail in noisy case!

Least squares structure

- Surrogate should allow for internal minima
  - Quadratic model: $1 + n + \frac{n(n+1)}{2}$ points
  - $2n + 1$ points with regularization
- Underdetermined models often defeat intuition
- Least-square structure helps
  - Fit linear models $m_i(x) = a_i + b_i^T x$ for each residual $f_i(x)$
  - $M(x) = \sum_i m_i(x)^2 = \sum_i a_i^2 + \sum_i 2a_i b_i^T x + \sum_i x^T b_i b_i^T x = \alpha + g^T x + \frac{1}{2} x^T H x$
  - Fully determined model with just $n + 1$ points

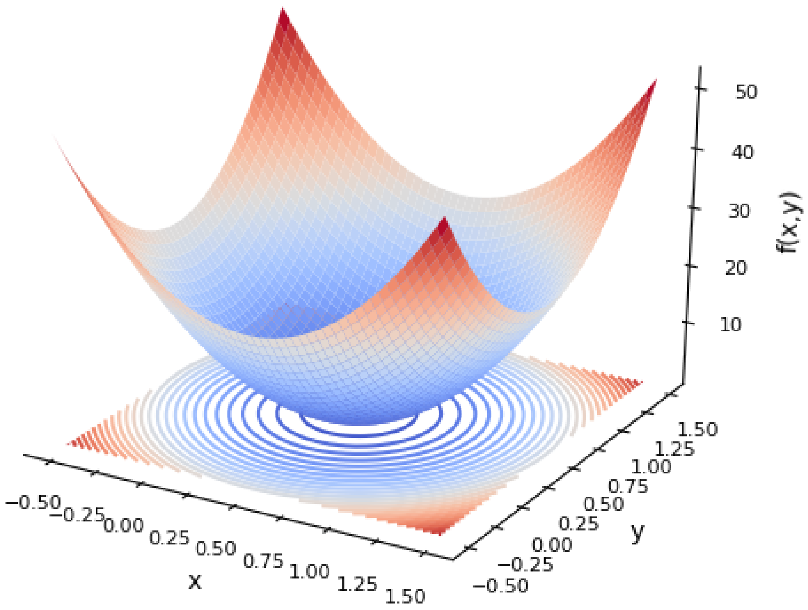
Noise-free and serial case

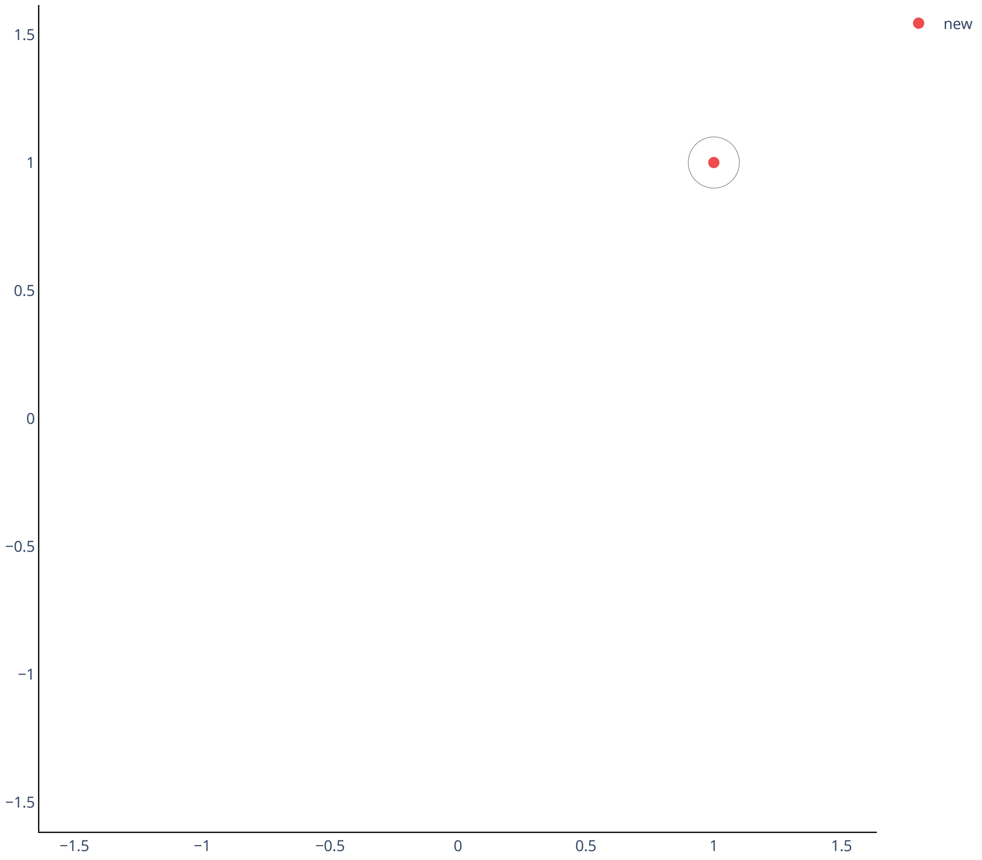
Tranquilo and Tranquilo-LS

- TrustRegion Adaptive Noise robust QUadratIc or Linear approximation Optimizer
- Fairly standard trustregion framework
  - Sampling: Approximate Fekete points
  - Subsolvers: GQTPAR or BNTR
  - Radius management: Same as POUNDERS
- Key differences
  - History search and variable sample size
  - Switch from round to cubic trustregions close to bounds
  - Same code for scalar and least-squares version!

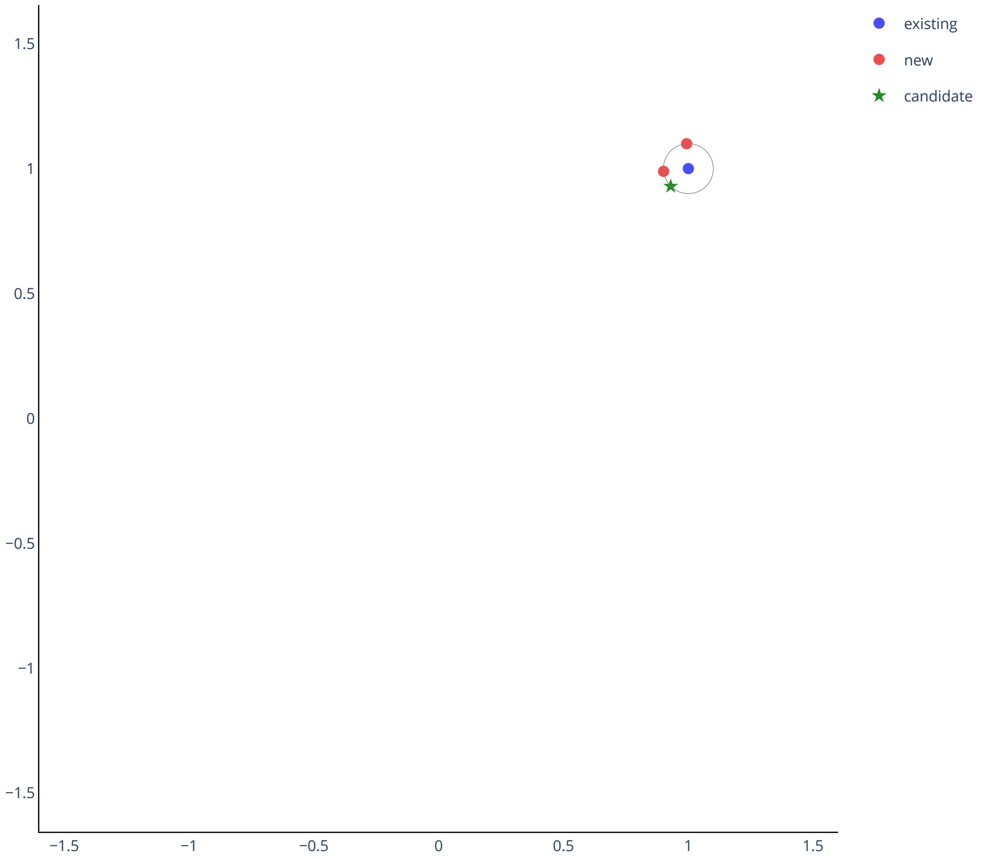
Tranquilo-LS in action

- Criterion function: $f(x) = \sum_i x_i^2$
- Start parameters: $x_0 = (1, 1)$
- Global optimum: $x^* = (0, 0)$

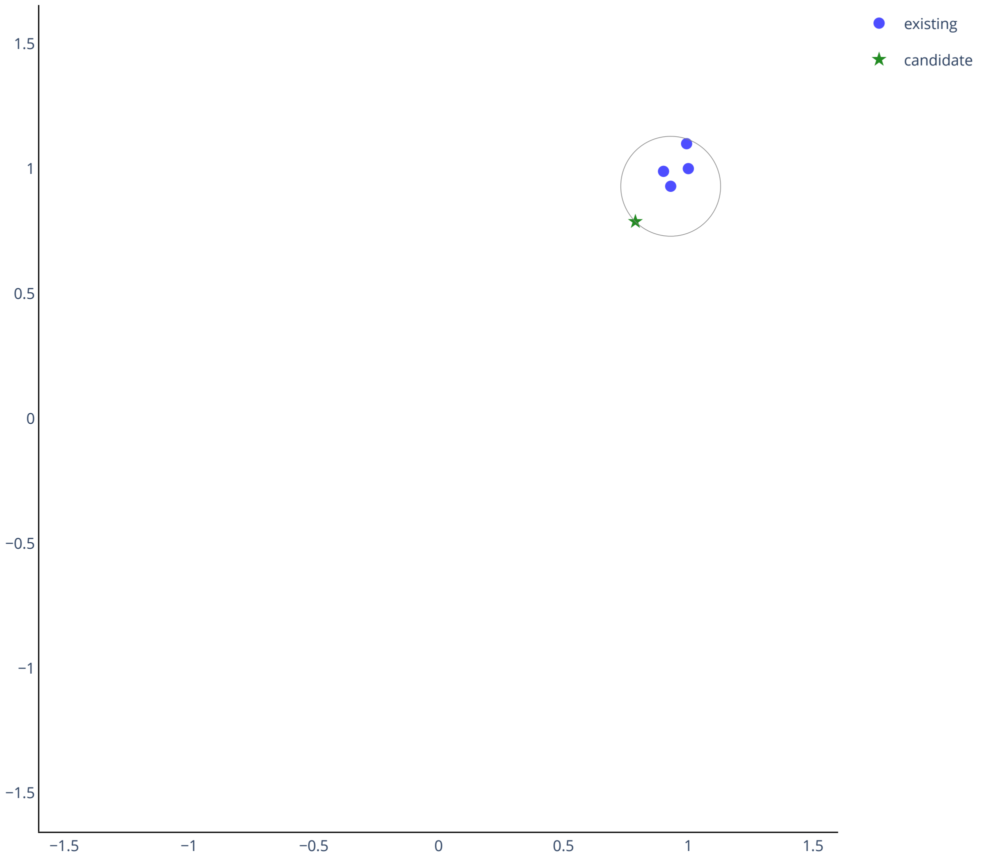




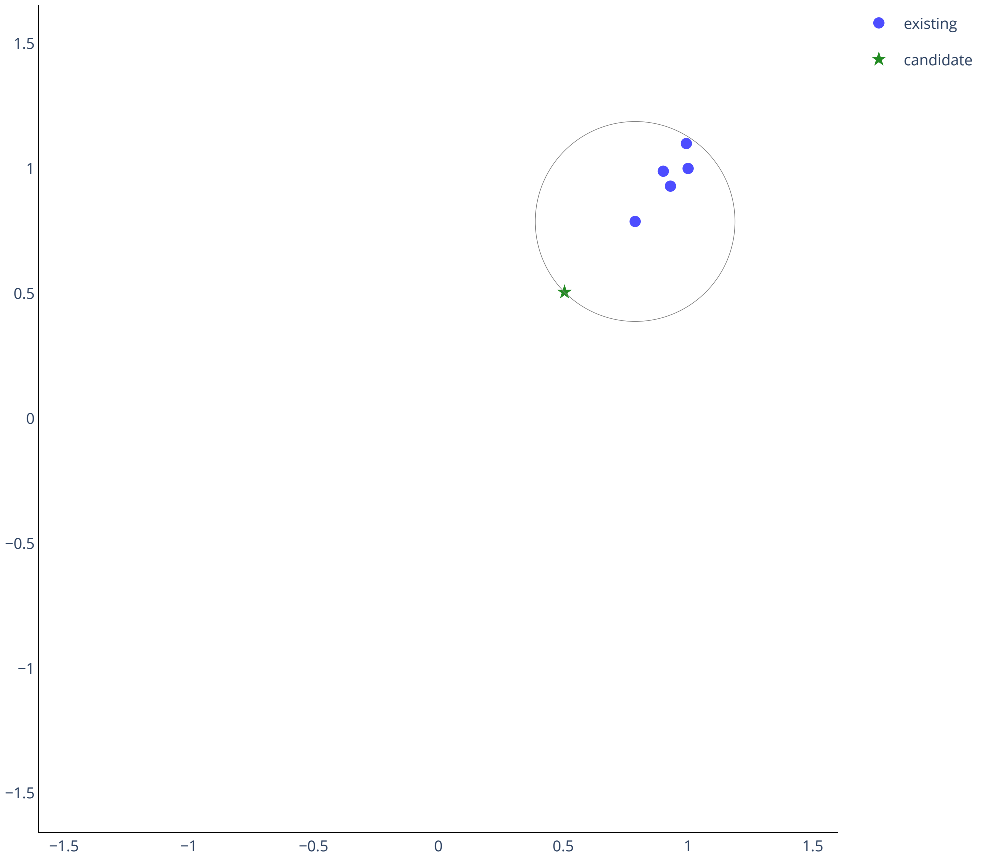
new
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5



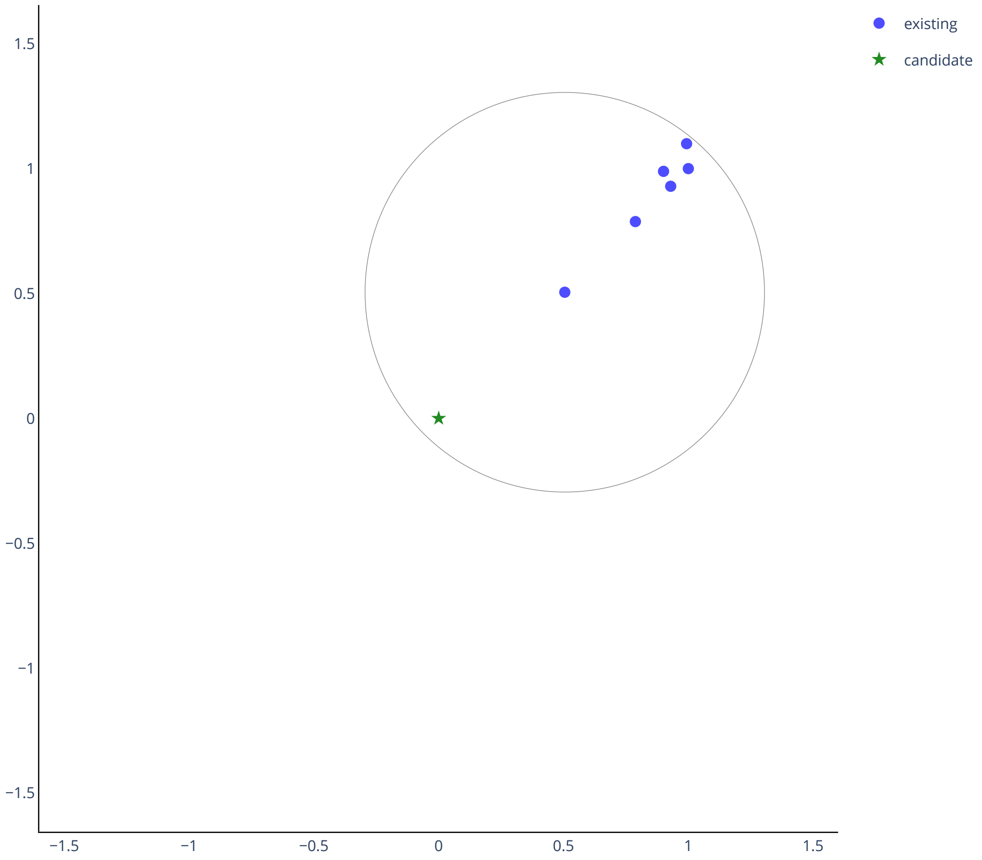
existing
new
candidate
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5

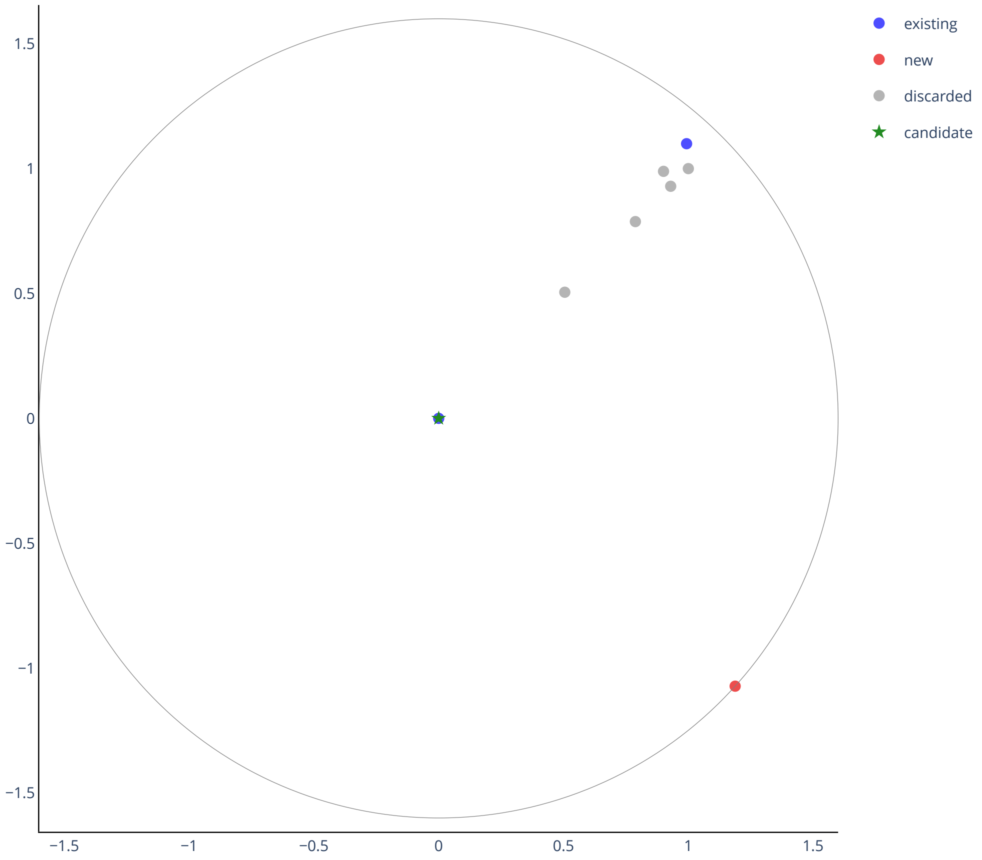


existing
candidate
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5



existing
candidate
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5


existing
candidate
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5

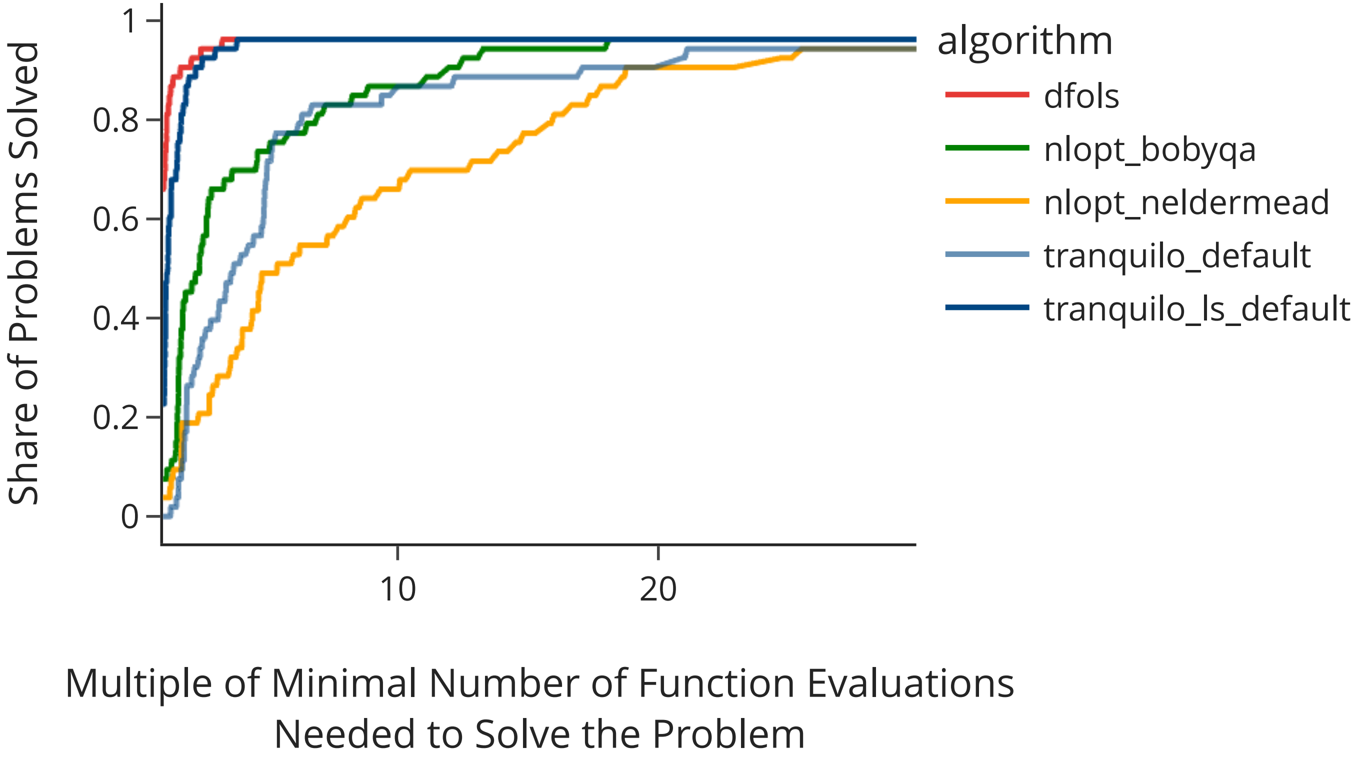


existing
new
discarded
candidate
1.5
1
0.5
0
−0.5
−1
−1.5
−1.5
−1
−0.5
0
0.5
1
1.5

Benchmarking

- Moré-Wild Benchmark set
- 52 leasts-squares problems with 2 to 12 parameters
- Used in POUNDERS, PyBobyqa and DFO-LS papers
- Differentiable (but we don’t use derivatives)
- Profile plots
  - Y-axis: share of solved problems
  - X-axis: computational cost in function evaluations
  - For each problem, cost is standardized by the cost of the best optimizer

Benchmark: Tranquilo vs. other optimizers

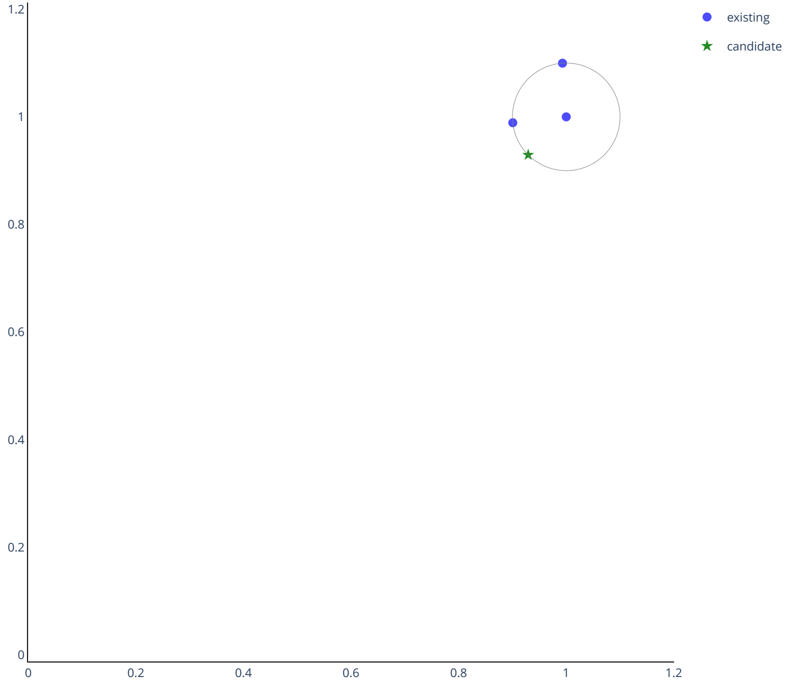


Parallel case

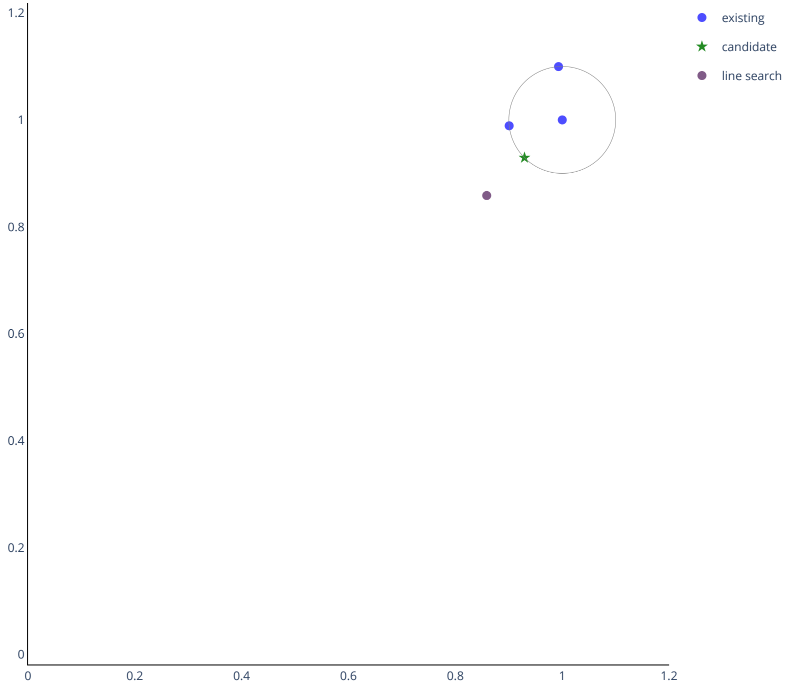
Cost model for parallel optimization

- Most economists have access to:
  - 4 to 8 cores on a laptop/desktop
  - 16 to 64 cores on a server
- In practice, criterion functions are often not parallelized
  - Lack of knowledge or time to write parallel code
  - Some problems are hard to parallelize
- Cost model with batch size b:
  - Want to avoid idle cores
  - b parallel evaluations have same cost as one

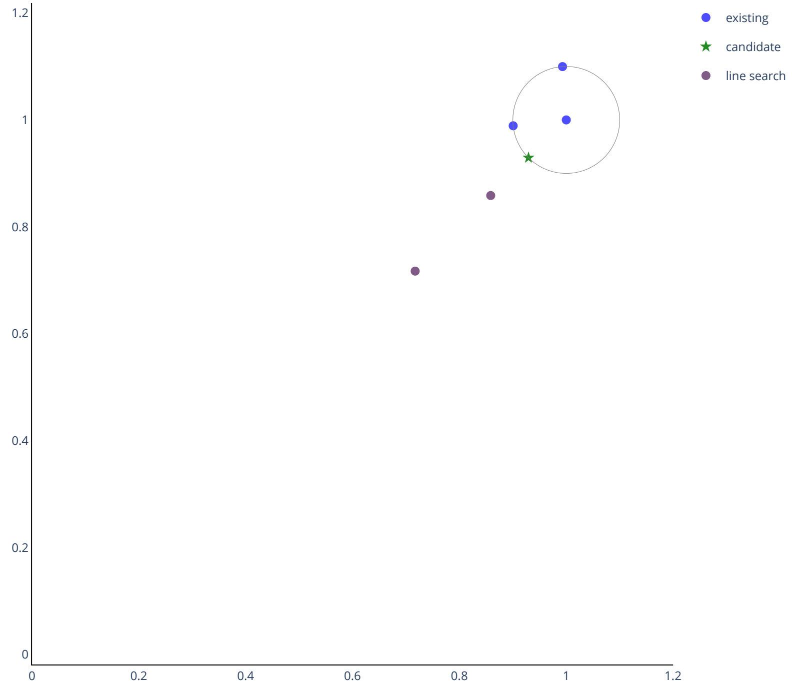
Idea 1: Parallel line search



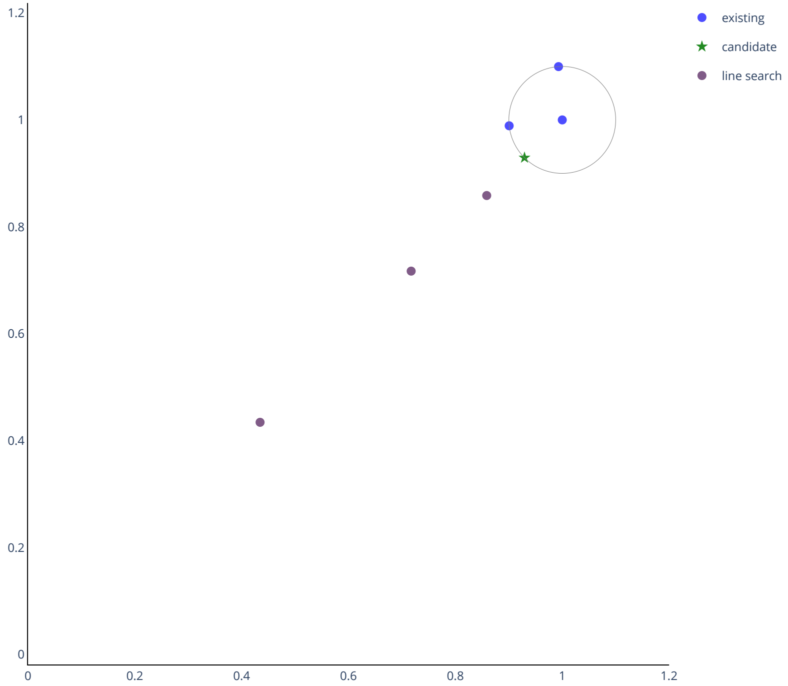
Idea 1: Parallel line search



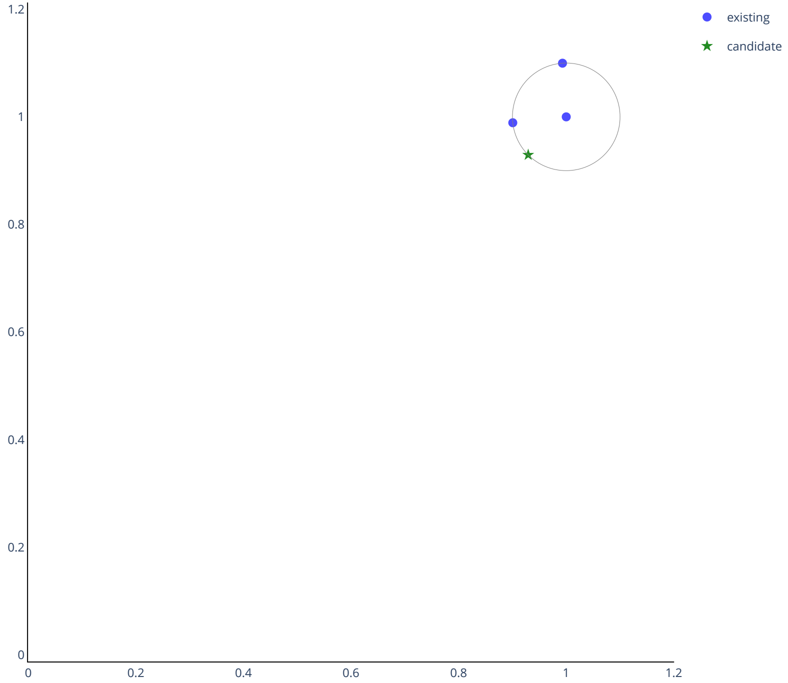
Idea 1: Parallel line search



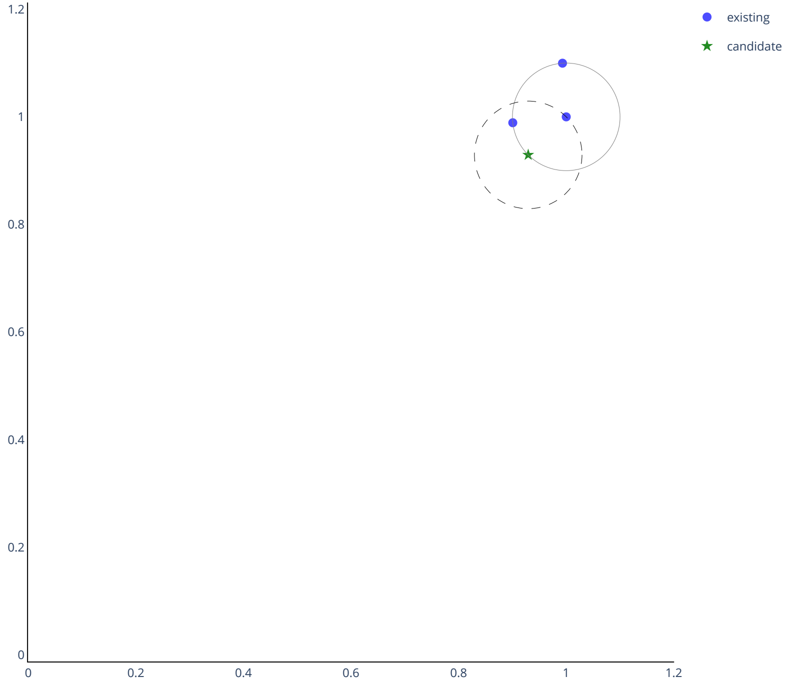
Idea 1: Parallel line search



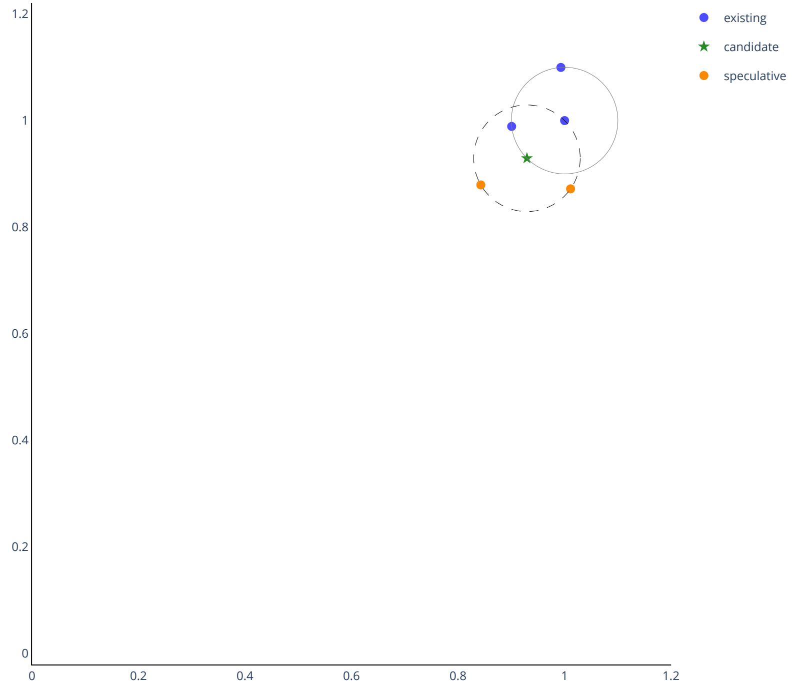
Idea 2: Speculative sampling



Idea 2: Speculative sampling



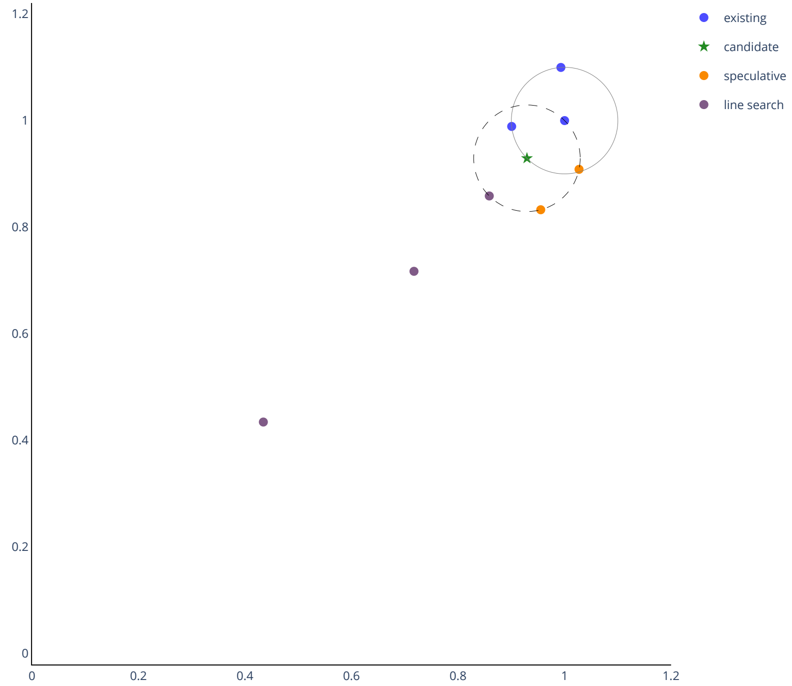
Idea 2: Speculative sampling



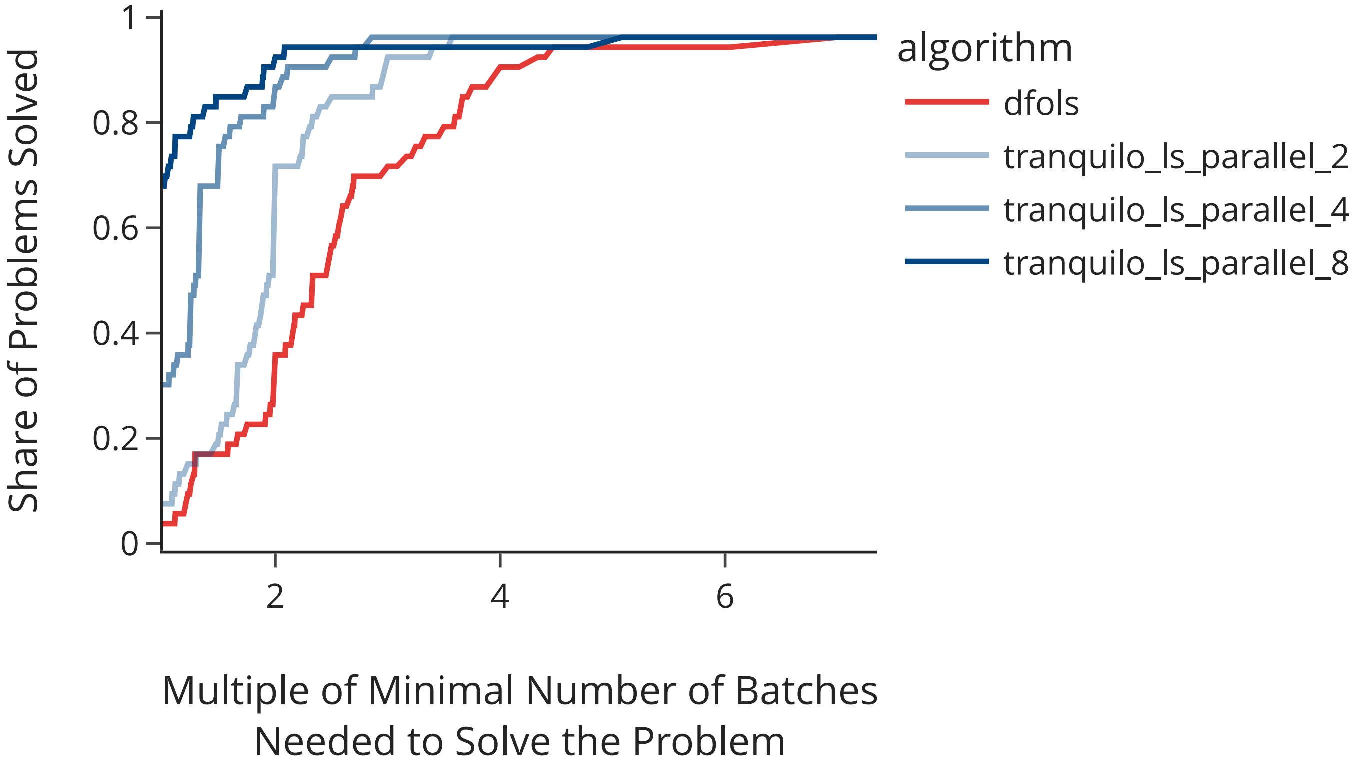
Combining the two

- If candidate is close to trustregion border:
  - Allocate up to three function evaluations to a line search
- If "free" function evaluations are left:
  - Do speculative sampling
- If any line-search or speculative point yields improvement
  - Accept them as new x

Line search + Speculation



Benchmark: Parallel tranquilo vs. DFO-LS



Noisy case

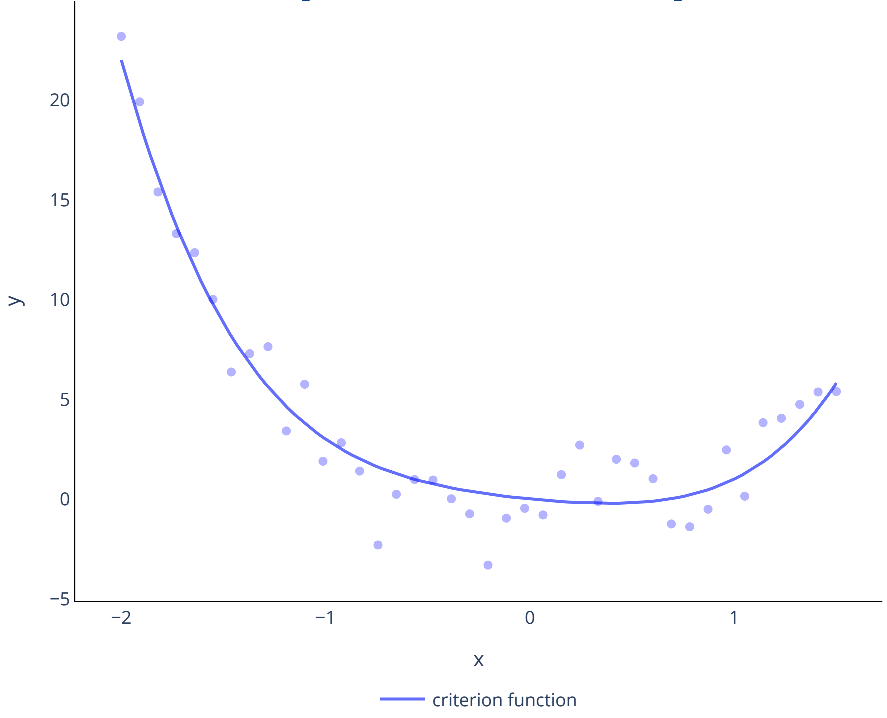
Problems caused by noise

- Model does not approximate well
- ρ is low in many iterations
- Radius shrinks to zero -> optimization fails

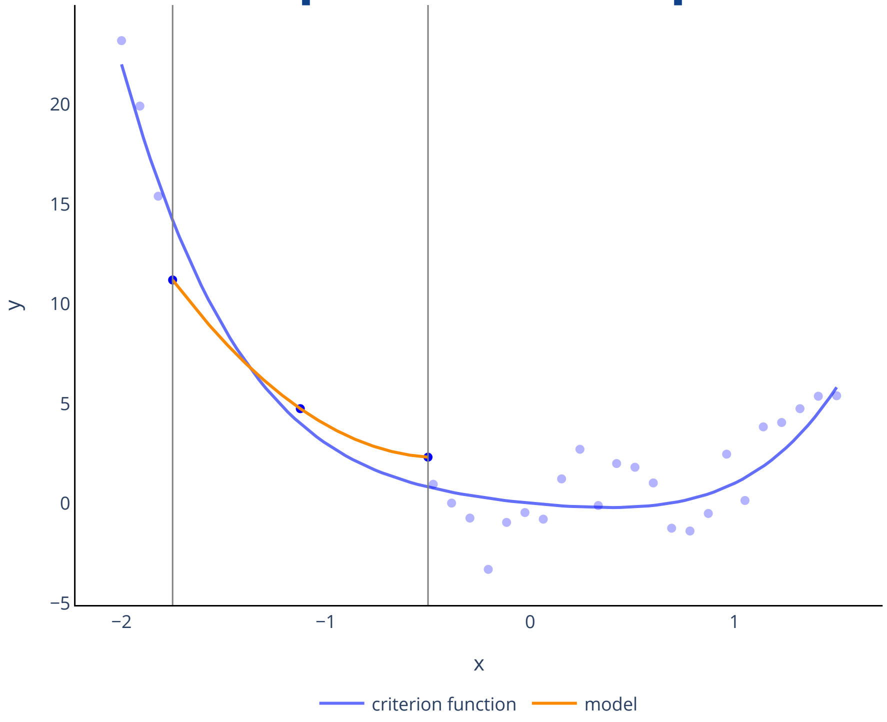
How DFO-LS handles noise

- Re-start if trustregion collapses
- Evaluate criterion multiple times at each point and average
- How many evaluations is decided by the user based on
  - Current radius
  - ρ
  - Iteration counter (k)
  - restart counter
- Very hard to get right!

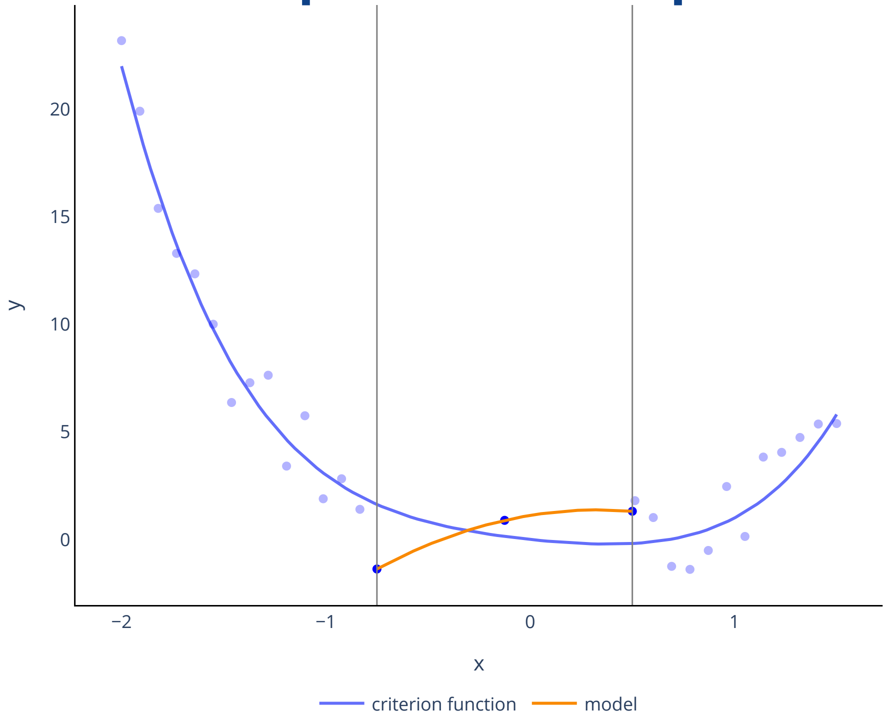
Why is it hard to pick sample sizes?



Why is it hard to pick sample sizes?



Why is it hard to pick sample sizes?



A different look on radius and ρ

Noise-free case

- Problem: Approximation error
- Tuning parameter: Radius
- Performance metric: ρ

Noisy case

- Problem: Random error
- Tuning parameter: Sample size
- Need: ρ_{noise}

Step 1: Estimate noise variance

- Scan history for all points with multiple evaluations of criterion
- Restrict to ones that are
  - close to current trustregion
  - have the most function evaluations
- Estimate
  - σ_k: variance of the noise on a scalar criterion function
  - Σ_k: covariance matrix of the noise on the least-squares residuals
- Locally constant approximation to an arbitrary noise term

Step 2: Simulate ρ_{noise}

- Surrogate model $M_k(x)$ approximates the criterion function
- Use M_k and σ_k to simulate a noisy sample
- Fit a model $\tilde{M}_k(x)$ on the simulated sample
- Optimize $\tilde{M}_k(x)$ to get a suggested step $\tilde{s}_k$
- $\rho_{noise} = \frac{M(x_k) - M(x_k + \tilde{s}_k)}{\tilde{M}_k(x_k) - \tilde{M}_k(x_k + \tilde{s}_k)}$
- Repeat the simulation
- Increase sample size if most rhos are small

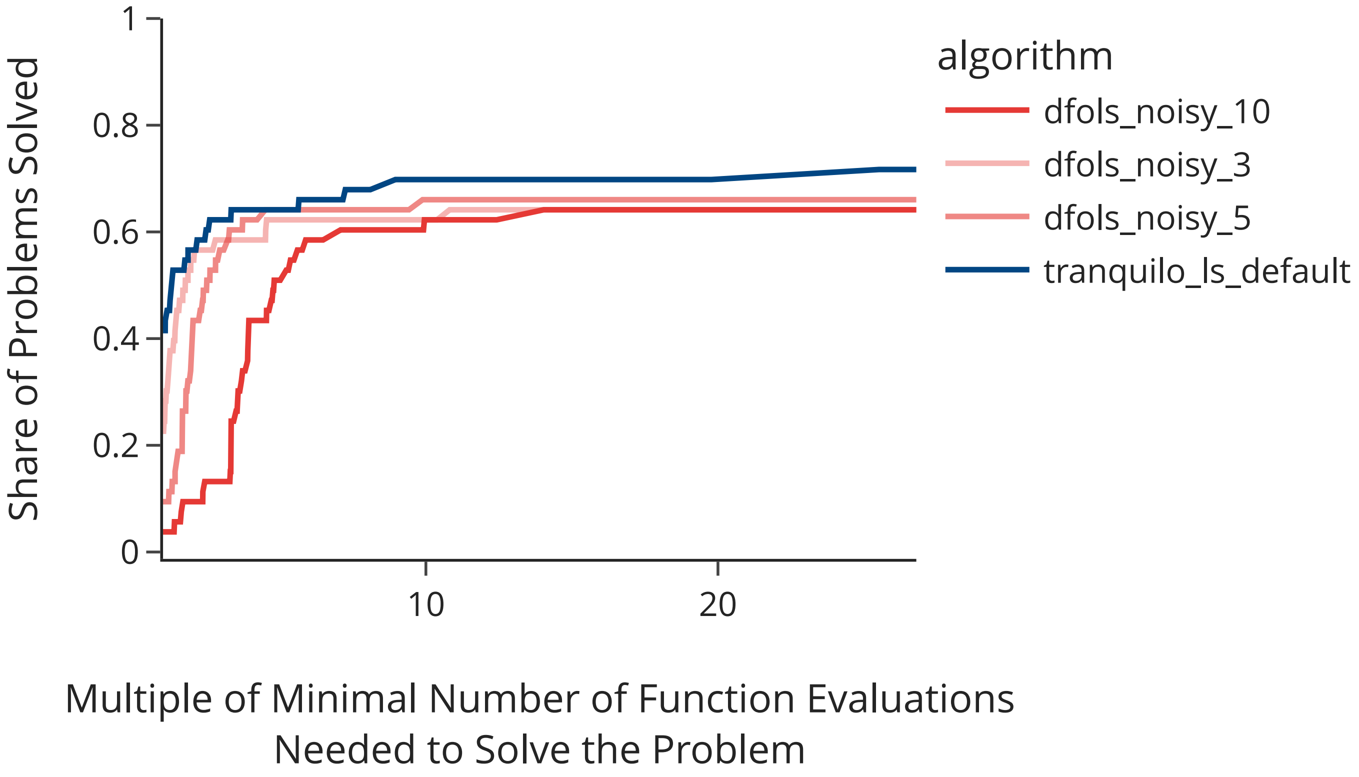
Noise in the acceptance step

- Noise free acceptance step is trivial
- Now: Does candidate have a lower expected value?
- Intuition: Needs large sample if values are close

Step 3: Power analysis

- Power analysis: $\frac{n_1 n_2}{n_1+n_2} \geq \sigma^2 \left[\frac{\Phi^{-1}(1-\alpha)+\Phi^{-1}(1-\beta)}{\Delta_{min}}\right]^2$
- n_1, n_2: number of evaluations at current and candidate x
- α: confidence level
- $1-\beta$: power level
- $\Delta_{min} = M_k(x_k) - M_k(x_k + s_k)$: Minimal detectable effect size
- Can calculate n_1 and n_2 that minimize new function evaluations

Benchmark: Noisy tranquilo vs. DFO-LS



Summary

- We created a modular framework for derivative free trustregion optimization
- Same code for scalar and least-squares version
- Performance in noise-free and serial setting is similar to existing optimizers
- Two ideas for parallelization:
  - Line search
  - Speculative sampling
- Two ideas for noise handling
  - Simulate ρ_{noise} in sampling step
  - Power analysis for acceptance step