Tuomo Valkonen

Proximal methods for point source localisation

The implementation

Regularised regression

The Lasso in $\mathbb{R}^n$

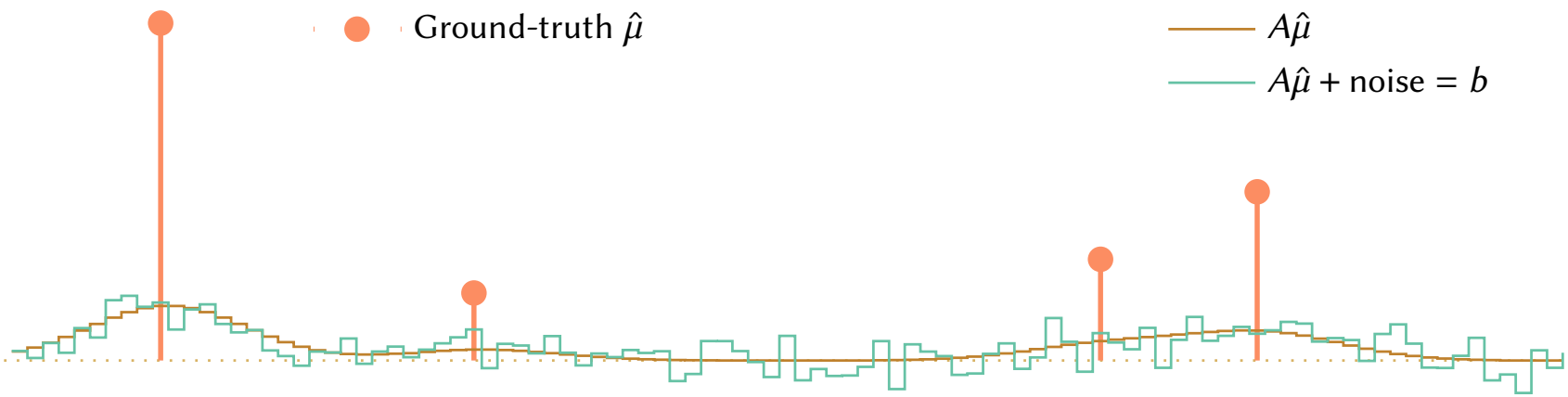
Solve

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2}\|Ax - b\|^2 + \alpha \|x\|_1.$$

The Lasso on[1] $\mathscr{M}_+(\Omega)$

In the space $\mathscr{M}(\Omega)$ of Radon (i.e., nice) measures on $\Omega \subset \mathbb{R}^n$, solve

$$\min_{\mu \in \mathscr{M}(\Omega)} \ \frac{1}{2}\|A\mu - b\|^2 + \alpha \|\mu\|_{\mathscr{M}(\Omega)} + \delta_{\geq 0}(\mu).$$



[1] Early formulations in Candès & Fernandez-Granda (2014), *Towards a Mathematical Theory of Super-resolution*

Forward-backward algorithm

Implicit step

- Pick a "particle-to-wave" operator $\mathscr{D} \in \mathbb{L}(\mathscr{M}(\Omega); C_0(\Omega))$ such that

$$A_* A \leq L\mathscr{D}.$$

- With

$$H(\mu) := \tau[A_*(A\mu^k - b) + \alpha] + \mathscr{D}(\mu - \mu^k) \in C_0(\Omega),$$

approximate implicit forward-backward step becomes

$$\text{on } \Omega \quad -\varepsilon_{k+1} \leq H(\mu^{k+1}) \leq \varepsilon_{k+1} \quad \text{on } \operatorname{supp} \mu^{k+1}.$$

Algorithm sketch for one FB step

1. **Optimise weights** β of $\mu^{k+1} = \sum_{x \in S} \beta_x \delta_x$ to have $H(\mu^{k+1}) \approx 0$ on $\operatorname{supp} \mu^{k+1}$. This may reduce $\operatorname{supp} \mu^{k+1}$ to satisfy right-hand-side
2. **Find minimiser** ξ of $H(\mu^{k+1})$.
3. **If in bounds**, $H(\mu^{k+1})(\xi) \geq -\varepsilon_{k+1}$, **finish**.
4. Otherwise **add** ξ **to support** of μ^{k+1}, and repeat.

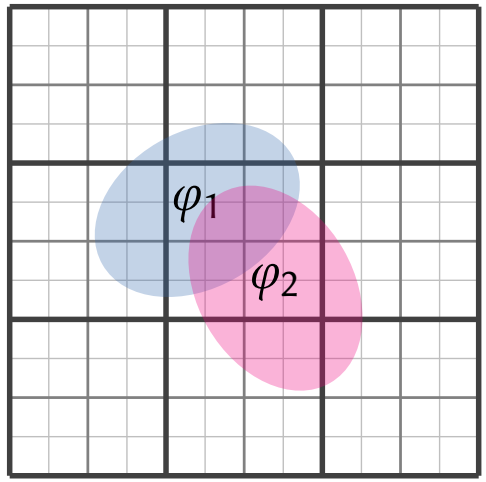
Implementation: main components

Weight optimisation subproblems

semismooth Newton ⇝ `nalgebra`.

Minimisation of H

branch-and-bound ⇝



- custom geometrical bisection tree structure for

$$\sum_k \varphi_k \quad \text{where each } \varphi_k \text{ has small support.}$$

  Maintains rough upper/lower bounds within each node/cube.
- multi-threaded with `rayon` pools and a priority queue.

Abstraction and tools

- Mappings, linear operators (A, $\mathscr{D}$), vector spaces, `GEMV`, `AXPY`, etc.
- `Float` with everything necessary; `linspace`; cubes; iteration;mapping…
- Iterative algorithm boilerplate

Iterative algorithm boilerplate separation

```
let mut logger = Logger::new();
let mut iter = AlgIteratorOptions {
    max_iter : 100,
    verbose_iter : Verbose::Every(10),
    .. Default::default()
}.timed()                       // Include time in output
 .into_log(&mut logger);        // Log the output

// ...

let mut x = 1 as float;

iter.iterate(|state|{
    x = x + x.sqrt();           // This is our computational step
    state.if_verbose(||{
        return x                // return current value when requested
    })
})
```

GATs for abstract functional analysis

```
// Trait for linear A: X → Y that have an adjoint A´: Y´ → X´
pub trait Adjointable<X, Y´> : Linear<X> {
    type AdjointCodomain; // X´
    type Adjoint<'a> : Linear<Y´, Codomain=Self::AdjointCodomain>
                       where Self : 'a;

    fn adjoint(&self) -> Self::Adjoint<'_>;
}

// ...

opA.adjoint().apply(x)  // No data copied to apply adjoint!
```

Challenges

nalgebra type piggybacking

`nalgebra` `RealField` has the same methods as `num_traits` `Float`
⇝ hell with `ToNalgebraRealField` piggybacking trait.

Standardisation? A way to require traits without importing their methods?

Trait bound bloat

Many of our functions have **20 lines** of *inherited* trait bounds.

Really? Do we really have to repeat inherited trait bounds?

Float literals

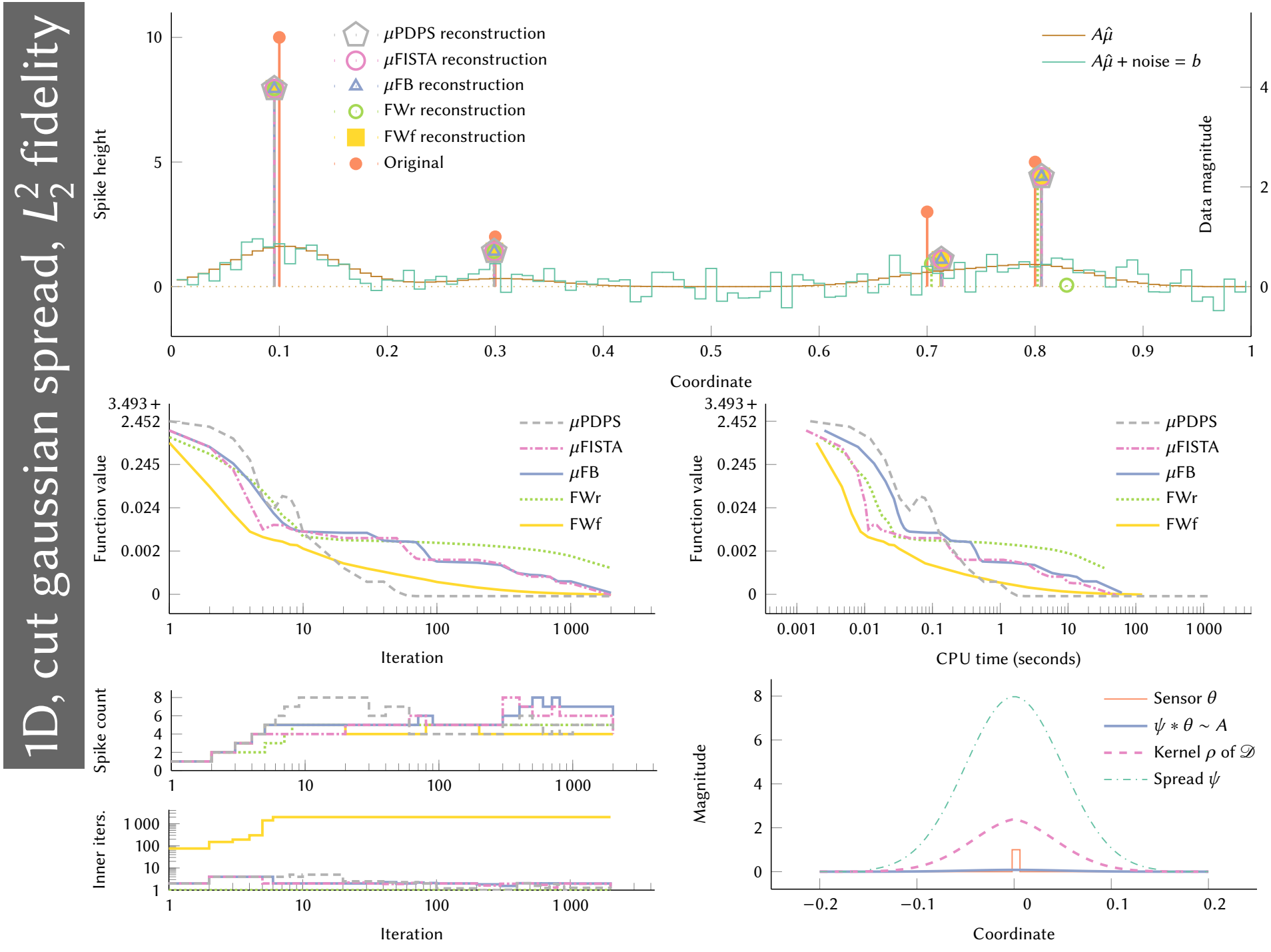
```
#[replace_float_literals(F::cast_from(literal))]
```

Returning closures and constructing iterators

Needed `Map` trait for methods instead of closures, and tons of boilerplate.

Documentation: math

Rust markdown stuck in the past: no KaTeX (without cumbersome workarounds).


1D, cut gaussian spread, L_2^2 fidelity
Spike height
Data magnitude
Coordinate
μPDPS reconstruction
μFISTA reconstruction
μFB reconstruction
FWr reconstruction
FWf reconstruction
Original
$A\hat{\mu}$
$A\hat{\mu}$ + noise = b
Function value
Iteration
CPU time (seconds)
μPDPS
μFISTA
μFB
FWr
FWf
Spike count
Inner iters.
Magnitude
Sensor θ
$\psi * \theta \sim A$
Kernel ρ of $\mathscr{D}$
Spread ψ

More details

Proximal methods for point source localisation

Manuscript: arXiv 2212.02991

Rust codes: Zenodo 7402055

Both: tuomov.iki.fi ⇝