A Very Short Survey of Solvers

Theo Diamandis

February 4, 2023
Review: Conic Programs

Most solvers work with the conic form of a problem:

\[
\begin{align*}
\text{minimize} & \quad c^T x \\
\text{subject to} & \quad Fx + g \preceq_K 0 \\
& \quad Ax = b
\end{align*}
\]

Modeling systems (e.g., Convex.jl) convert a problem to conic form by rewriting constraints \( f(x) \leq t \) as conic inequalities:

\[
\begin{align*}
x^T P x + & \leq t \iff \| (P^{1/2} x, (t - 1)/2) \|_2 \leq (t + 1)/2 \\
\| X \|_2 & \leq t \iff \begin{bmatrix} tl & X \\ X^T & tl \end{bmatrix} \succeq 0
\end{align*}
\]
Gradient method with step size $\alpha_k$ (constant or from line search):

$$x^{(k+1)} = x^{(k)} - \alpha_k \nabla f(x^{(k)})$$

Pros: inexpensive iterations

Cons: often very slow convergence (can’t get accurate solns); sensitive to scaling

Extensions (proximal gradient, accelerated gradient) are faster
Decomposition methods implement these ideas.

- **Very popular method**: Alternating Direction Method of Multipliers (ADMM)

- **Advantages**:
  - Works on huge problems; often can be parallelized/distributed
  - Often converges quickly to moderately accurate solution

- **Disadvantages**:
  - Very sensitive to problem scaling
  - Slow to get to high accuracy

- **Examples**: SCS, OSQP (QPs only), COSMO.jl
ADMM: split problem into easier problems

▶ Idea: \( \min f(x) + g(x) \rightarrow \min f(x) + g(z) \text{ s.t. } x = z \)

▶ Form the augmented Lagrangian (helps with smoothness & convergence)

\[
L(x, z, \nu) = f(x) + g(z) + \nu^T (x - z) + \left( \frac{\rho}{2} \right) \|x - z\|^2.
\]

▶ Iterate:

\[
\begin{align*}
x^{(k+1)} &= \arg\min_x L(x, z^{(k)}, \nu^{(k)}) \\
z^{(k+1)} &= \arg\min_z L(x^{(k+1)}, z, \nu^{(k)}) \\
\nu^{(k+1)} &= \nu^{(k)} + (x^{(k+1)} - z^{(k+1)})
\end{align*}
\]
Second-order algorithms use Hessian information

- Use Hessian (or approximate Hessian) information to accelerate convergence
- Converge very quickly for unconstrained or linearly constrained smooth problems
- Common method: L-BFGS(-B) (approximates Hessian)
  - See Optim.jl and LBFGSB.jl
- Key parameter: line search (which controls the step size)
Second-order methods solve the optimality conditions

- In the unconstrained case, solve $\nabla f_0(x) = 0$.

- The Newton step $\Delta x$ at current iterate $x^{(k)}$ solves the system with $\nabla f_0$ replaced by its first-order approximation around $x^{(k)}$:

  $$\nabla f_0(x^{(k)}) + \nabla^2 f_0(x^{(k)}) \Delta x = 0.$$  

- Alternative interpretation: $x + \Delta x$ minimizes the second order approximation to $f_0$ around $x^{(k)}$:

  $$\Delta x = \arg\min_v f_0(x^{(k)}) + \nabla f_0(x^{(k)})^T v + \frac{1}{2} v^T \nabla^2 f_0(x^{(k)}) v.$$
Interior point methods

- Interior point methods deal with (smooth) conic programs by turning the inequality constraints into part of the objective function

\[
\begin{align*}
\min \ f_0(x) & \quad \rightarrow \quad \min \ tf_0(x) - \sum_i \log(-f_i(x)) \\
\text{s.t.} \quad f_i(x) & \leq 0
\end{align*}
\]

- Approximation improves as \( t \to \infty \)

- Different cones have different associated barrier functions

- Must start with feasible point
  - Enough to know a feasible point for each inequality individually
  - Primal-dual methods get around this & are more robust
Interior point methods are very fast and accurate

- Often converge in <100 iterations (and good worst-case complexity)
- Quadratic convergence to machine precision when close to the optimal (very different than first order methods!)
- Implementations: Hypatia.jl, ECOS, Mosek, Gurobi (QCQPs)
- Usually needed for poorly scaled problems (e.g., many SDPs)
How to choose a solver?

▶ Choose an interior point method if you...
  – need a very accurate solution
  – have a poorly scaled problem
  – have a small to modest-sized problem

▶ Choose a first-order method if you...
  – have a very large problem (but if unconstrained, LBFGS works great)
  – don’t need an accurate solution (common in machine learning)

▶ https://jump.dev/JuMP.jl/stable/installation/#Supported-solvers