Log Barrier Method (contd.)

- Our objective becomes

$$\min_x f(x) + \sum_i \left(-\frac{1}{t}\right) \log\left(-g_i(x)\right)$$

$$\text{s.t. } Ax = b$$

- At different values of t, we get different $x^*(t)$
- Let $\lambda_i^*(t) =$ value that leads to satisfaction of necessary condition of original problem (assuming necessary condition of barrier problem is satisfied)
- First-order necessary conditions for optimality (and strong duality)[13] at $x^*(t), \lambda_i^*(t)$:
  1. .. Satisfying complementary slackness is the challenge
  2. .. - Addressed by iteratively solving
  3. ..
  4. ..
     - ★ ..
-

[13] of original problem

- Our objective becomes

$$\min_x f(x) + \sum_i \left(-\frac{1}{t}\right) \log\left(-g_i(x)\right)$$
$$\text{s.t. } Ax = b$$

- At different values of t, we get different x^*
- Let $\lambda_i^*(t) = \frac{-1}{t\, g_i\left(x^*(t)\right)}$
- First-order necessary conditions for optimality (and strong duality)[14] at $x^*(t), \lambda_i^*(t)$:
  1. $g_i\left(x^*(t)\right) \le 0$
  2. $Ax^*(t) = b$
  3. $\nabla f\left(x^*(t)\right) + \sum_{i=1}^m \lambda_i^*(t) \nabla g_i\left(x^*(t)\right) + \nu^*(t)^\top A = 0$
  4. $\lambda_i^*(t) \ge 0$
     - ★ Since $g_i\left(x^*(t)\right) \le 0$ and $t \ge 0$
- All above conditions hold at optimal solution $\mathbf{x}(t), \nu(t)$, of barrier problem $\Rightarrow$ $\left(\lambda_i^*(t), \nu^*(t)\right)$ are dual feasible. Feasibility is ensured. But since complementary slackness might be violated, can't yet talk of optimality

[14] of original problem

Log Barrier Method & Duality Gap

- If necessary conditions are satisfied and **if f and g_i's are convex, and g_i's strictly feasible**, the conditions are also sufficient. Thus, $\left(x^*(t), \lambda_i^*(t), \nu^*(t)\right)$ form a critical point for the Lagrangian

$$L(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \nu^\top (A\mathbf{x} - \mathbf{b})$$

- Lagrange dual function

$$L^*(\lambda, \nu) = \min_{\mathbf{x}} L(\mathbf{x}, \lambda, \nu)$$

$$L^*\left(\lambda^*(t), \nu^*(t)\right) = f\left(\mathbf{x}^*(t)\right) + \sum_{i=1}^{m} \lambda_i^*(t) g_i\left(\mathbf{x}^*(t)\right) + \nu^*(t)^\top \left(A\mathbf{x}^*(t) - \mathbf{b}\right)$$

$$= f(x^*(t)) - m/t <= p^*$$

  - ..m/t... is the *duality gap*
  - As $t \to \infty$, duality gap $\to$ .zero

Log Barrier Method & Duality Gap

- If necessary conditions are satisfied and **if f and g_i's are convex, and g_i's strictly feasible**, the conditions are also sufficient. Thus, $\left(x^*(t), \lambda_i^*(t), \nu^*(t)\right)$ form a critical point for the Lagrangian

$$L(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i g_i(\mathbf{x}) + \nu^\top (A\mathbf{x} - \mathbf{b})$$

- Lagrange dual function

$$L^*(\lambda, \nu) = \min_{\mathbf{x}} L(\mathbf{x}, \lambda, \nu)$$

$$\begin{aligned} L^*\left(\lambda^*(t), \nu^*(t)\right) &= f\left(\mathbf{x}^*(t)\right) + \sum_{i=1}^{m} \lambda_i^*(t) g_i\left(\mathbf{x}^*(t)\right) + \nu^*(t)^\top \left(A\mathbf{x}^*(t) - \mathbf{b}\right) \\ &= f\left(x^*(t)\right) - \frac{m}{t} \end{aligned}$$

  - $\frac{m}{t}$ here is called the *duality gap*
  - As $t \to \infty$, duality gap $\to 0$, but computing optimal solution $\mathbf{x}(t)$ to barrier problem will be that harder

Log Barrier Method & Duality Gap

- At optimality, primal optimal = dual optimal
  i.e. $p^* = d^*$
- From weak duality,

$$f\left(\mathbf{x}^*(t)\right) - \frac{m}{t} \leq p^*$$

$$\implies f\left(\mathbf{x}^*(t)\right) - p^* \leq \frac{m}{t}$$

  - The duality gap is always $\leq \frac{m}{t}$
  - The more we increase t, the smaller will be the duality gap

Iterative algorithm

1. Start with $t = t^{(0)}$, $\mu > 1$, and consider ϵ tolerance
2. **Repeat**

   t is multiplicatively scaled up by \mu in every iteration

   1. **Solve**

   $$x^*(t) = \underset{x}{\text{argmin}}\ f(x) + \sum_{i=1}^{m} \left(-\frac{1}{t}\right) \log\left(-g_i(x)\right)$$
   $$\text{s.t.}\ \ Ax = b$$

   A little later we discuss issues/technique effective for solving this

   2. If $\frac{m}{t} < \epsilon$, **Quit**
      else, **set** $t = \mu t$

- In the process, we can also obtain $\lambda^*(t)$ and $\nu^*(t)$
- **Convergence of outer iterations:**
  We get ϵ accuracy after $\log\left(\frac{\left(m/\epsilon t^{(0)}\right)}{log(\mu)}\right)$ updates of t

Log Barrier Method & Strictly Feasible Starting Point

Issue 1) Initially feasible solution
Issue 2) Solving efficiently

- The inner optimization in the iterative algorithm using a barrier method,

$$x^*(t) = \operatorname*{argmin}_x f(x) + \sum_i \left(-\frac{1}{t}\right) \log\left(-g_i(x)\right)$$

$$\text{s.t. } Ax = b$$

  can be solved using (sub)gradient descent starting from older value of x from previous iteration

- We must start with a strictly feasible x, otherwise $-\log\left(-g_i(x)\right) \to \infty$

How to find a strictly feasible $x^{(0)}$?

HINT: The same Barrier algorithm, applied on a suitable modification of the original optimization problem can be employed to find the initial strictly feasible x

How to find a strictly feasible $x^{(0)}$?

- *Basic Phase I method*

$$x^{(0)} = \underset{x}{\operatorname{argmin}}\, \Gamma$$

$$\text{s.t. } g_i(x) \leq \Gamma$$

- We solve this using the barrier method, and thus will also need a strictly feasible starting $\hat{x}^{(0)}$
- Here,

$$\Gamma = \max_{i=1\ldots m} g_i(\hat{x}^{(0)}) + \delta$$

where, $\delta > 0$

- *i.e.* Γ is slightly larger than the largest $g_i(\hat{x}^{(0)})$

- On solving this optimization for finding $x^{(0)}$,
  - If $\Gamma^* < 0$, $x^{(0)}$ is strictly feasible
  - If $\Gamma^* = 0$, $x^{(0)}$ is feasible (but not strictly)
  - If $\Gamma^* > 0$, $x^{(0)}$ is not feasible Deadend-problem is infeasible
- A slightly 'richer' problem can consider different Γ_i for each g_i, to improve numerical precision

$$x^{(0)} = \underset{x}{\text{argmin}}\, \Gamma_i$$

$$\text{s.t. } g_i(x) \leq \Gamma_i$$

Choice of a good $\hat{x}^{(0)}$ or $x^{(0)}$ depends on the nature/class of the problem, use domain knowledge to decide it

Log Barrier Method & Strictly Feasible Starting Point

Issue 2: Tradeoff between GOOD and SLOW inner solvers (for barrier problem) vs. FAST and SLOPPY inner solvers

- We need not obtain $x^*(t)$ exactly from each outer iteration
- If not solving for $x^*(t)$ exactly, we will get ϵ accuracy after *more than* $\log\left(\frac{(m/\epsilon t^{(0)})}{log(\mu)}\right)$ updates of t
  - However, solving the inner iteration exactly may take too much time
  - Fewer inner loop iterations correspond to more outer loop iterations

Can descent algorithms (that use Hessian) exploiting curvature information let us compute x(\mu t) from x(t) more efficiently

It turns out the curvature based algos work well for this kind of relatedness between successive subproblems

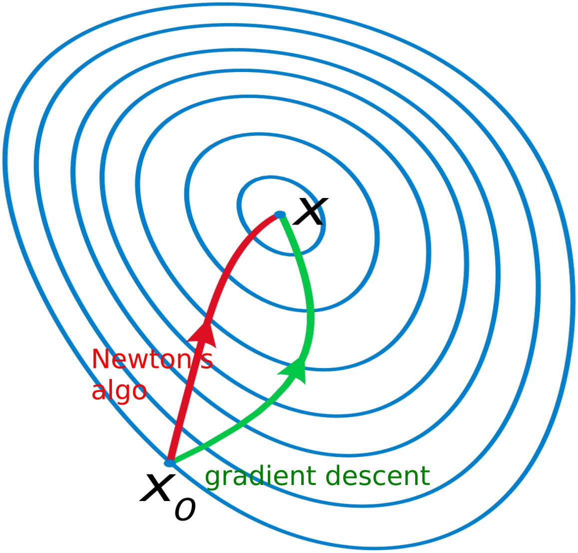
Log Barrier Method & Strictly Feasible Starting Point

- We need not obtain $x^*(t)$ exactly from each outer iteration
- If not solving for $x^*(t)$ exactly, we will get ϵ accuracy after *more than* $\log\left(\frac{\left(m/\epsilon t^{(0)}\right)}{log(\mu)}\right)$ updates of t
  - However, solving the inner iteration exactly may take too much time
  - Fewer inner loop iterations correspond to more outer loop iterations
- Second order descent algorithms (such as Newton Descent) found effective in such settings for following reasons:

Log Barrier Method & Strictly Feasible Starting Point

- We need not obtain $x^*(t)$ exactly from each outer iteration
- If not solving for $x^*(t)$ exactly, we will get ϵ accuracy after *more than* $\log\left(\frac{\left(m/\epsilon t^{(0)}\right)}{log(\mu)}\right)$ updates of t
  - However, solving the inner iteration exactly may take too much time
  - Fewer inner loop iterations correspond to more outer loop iterations
- Second order descent algorithms (such as Newton Descent) found effective in such settings for following reasons:
  - Accounts for curvature of the function; useful to converge to $\mathbf{x}(\mu t)$ quickly from $\mathbf{x}(t)$.
  - Quadratic convergence when close to $\mathbf{x}^*(t)$

Second Order Descent and Approximations Sections 4.5.2 - 4.5.6 of BasicsOfConvexOptimization.pdf


x
Newton's
algo
gradient descent
x_0

Newton's Algorithm as a Steepest Descent Method

- This choice of $\Delta\mathbf{x}^{k+1}$ corresponds to the direction of steepest descent under the matrix norm[15] induced by the Hessian $\nabla^2 f(\mathbf{x}^k)$:
  $\Delta\mathbf{x}^{(k)} = \text{argmin}\left\{\nabla^T f(\mathbf{x}^{(k)})\mathbf{v} \mid ||\mathbf{v}||_{\nabla^2 f(\mathbf{x}^k)} = 1\right\}.$
- Equivalently, based on approximating a function around the current iterate $\mathbf{x}^{(k)}$ using a second degree Taylor expansion.

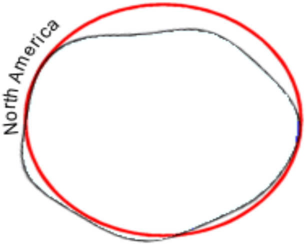
$$Q(\mathbf{x}) \approx \widetilde{f}(\mathbf{x}) = f(\mathbf{x}^{(k)}) + \nabla^T f(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x}-\mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x}-\mathbf{x}^{(k)})$$

- Convex $f \Rightarrow$

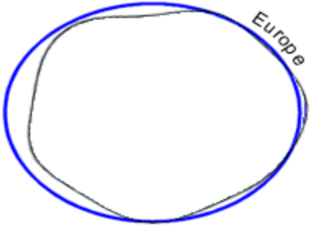
Recall: Gradient descent had an IDENTITY matrix in the quadratic part

Hessian is positive semi-definite
==> Quadratic approx Q(x) is convex

[15] $\left(\mathbf{v}^T \nabla^2 f(\mathbf{x}^k)\mathbf{v}\right)^{\frac{1}{2}}$

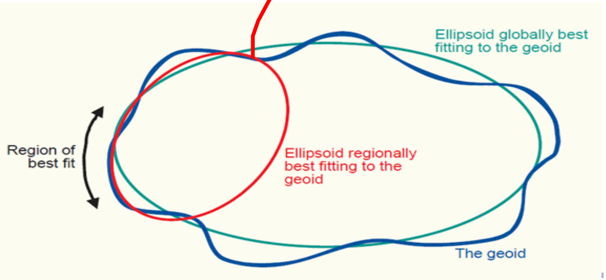


The red ellipsoid fits the geoid well in North America.



The blue ellipsoid fits the geoid well in Europe.

Local quadratic approximation to surface



Newton's Algorithm as a Steepest Descent Method

- This choice of $\Delta \mathbf{x}^{k+1}$ corresponds to the direction of steepest descent under the matrix norm[15] induced by the Hessian $\nabla^2 f(\mathbf{x}^k)$:
$\Delta \mathbf{x}^{(k)} = \text{argmin} \left\{ \nabla^T f(\mathbf{x}^{(k)})\mathbf{v} \mid ||\mathbf{v}||_{\nabla^2 f(\mathbf{x}^k)} = 1 \right\}$.
- Equivalently, based on approximating a function around the current iterate $\mathbf{x}^{(k)}$ using a second degree Taylor expansion.

$$Q(\mathbf{x}) \approx \tilde{f}(\mathbf{x}) = f(\mathbf{x}^{(k)}) + \nabla^T f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)})^T \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x} - \mathbf{x}^{(k)})$$

- Convex $f \Rightarrow$ convex quadratic approximation. Newton's method is based on solving the approximation exactly
- Setting gradient of quadratic approximation (with respect to $\mathbf{x}$) to $\mathbf{0}$ gives

$$\nabla^T f(\mathbf{x}^{(k)}) + \nabla^2 f(\mathbf{x}^{(k)})(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}) = 0$$

Assuming $\nabla^2 f(\mathbf{x}^k)$ is invertible, next iterate is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x}^{(k)})$

15 $\left(\mathbf{v}^T \nabla^2 f(\mathbf{x}^k)\mathbf{v}\right)^{\frac{1}{2}}$

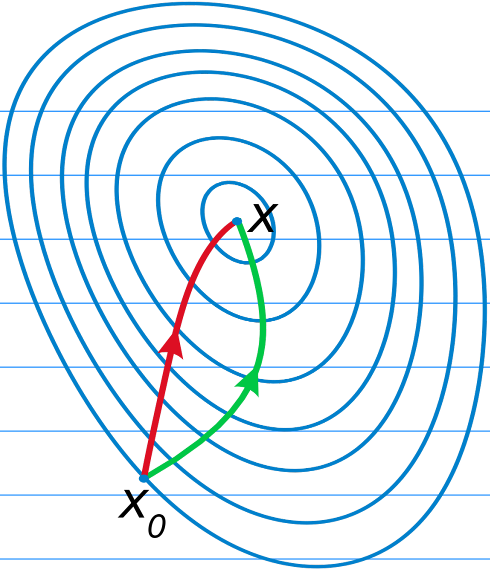
Newton's Algorithm as a Steepest Descent Method

Find a starting point $\mathbf{x}^{(0)} \in \mathcal{D}$.
Select an appropriate tolerance $\epsilon > 0$.
repeat

1. Set $\Delta\mathbf{x}^{(k)} = -\left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x})$.
2. Let $\lambda^2 = \nabla^T f(\mathbf{x}^{(k)}) \left(\nabla^2 f(\mathbf{x}^{(k)})\right)^{-1} \nabla f(\mathbf{x}^{(k)})$ $\Leftrightarrow$ Directional derivative in the Newton Direction
   Newton decrement
3. If $\frac{\lambda^2}{2} \leq \epsilon$, **quit**.
4. Set step size $t^{(k)} = 1$. Obtain $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t^{(k)}\Delta\mathbf{x}^{(k)}$. No line search!!
5. Set $k = k + 1$.

until

Figure 28: The Newton's method which typically uses a step size of 1. $\Delta\mathbf{x}^{(k)}$ can be shown to be always a Descent Direction (Theorem 83 of notes). For $\mathbf{x} \in \Re^n$, each Newton's step takes $O(n^3)$ time (without using any fast matrix multiplication methods).



Tend to converge in few iteration
However, each iteration itself
might be expensive

Quasi-newton algorithms
Problem specific methods
try to speed up each iteration

Variants of Newtons's Method

- **Special Cases:** When Objective function is a composition of two functions (such as Loss l over some Prediction function m): Gauss Newton Approximation (Section 4.5.4 of BasicsOfConvexOptimization.pdf) and Levenberg-Marquardt (Section 4.5.5) problem specific
- **Quasi-Newton Algorithms:** When Hessian inverse $\left(\nabla^2 f(\mathbf{x}^{k+1})\right)^{-1}$ is approximated by a matrix B^{k+1} such that
  - gradient of quadratic approximation $Q(\mathbf{x}^k)$ agrees at $\mathbf{x}^k$ and $\mathbf{x}^{k+1}$
  - B^{k+1} is as close as possible to B^k in some norm (such as the Frobenius norm)

  See BFGS (Section 4.5.6), LBFGS *etc.*

Cutting Plane Algorithm
(Invoking Linear Programs for Non-linear constraints)

Cutting Plane Algorithm

Consider amother general formulation of convex optimization problems[16]:

$$\begin{array}{lll} \text{minimize} & \mathbf{c}^T\mathbf{x} & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 & \text{for } i = 1, 2, \ldots, m \end{array} \tag{85}$$

where $g_i(\mathbf{x})$ are convex functions.

- How can every convex optimization problem be presented in this form?

[16] All convex optimization problems of the form discussed so far can be cast in this form.

Cutting Plane Algorithm

Consider amother general formulation of convex optimization problems[16]:

$$\begin{array}{lll} \text{minimize} & \mathbf{c}^T\mathbf{x} & \\ \text{subject to} & g_i(\mathbf{x}) \le 0 & \text{for } i = 1, 2, \dots, m \end{array} \tag{85}$$

where $g_i(\mathbf{x})$ are convex functions.

- How can every convex optimization problem be presented in this form? For objective function $f(\mathbf{x})$, translate it into a constraint $f(\mathbf{x}) - c \le 0$ and minimize c
- Let $\mathbf{s}_j(\mathbf{x}^i)$ be a subgradient for g_j at $\mathbf{x}^i$. By definition of subgradient

[16]All convex optimization problems of the form discussed so far can be cast in this form.

Cutting Plane Algorithm

Consider amother general formulation of convex optimization problems[16]:

$$\begin{array}{lll} \text{minimize} & \mathbf{c}^T\mathbf{x} & \\ \text{subject to} & g_i(\mathbf{x}) \leq 0 & \text{for } i = 1, 2, \ldots, m \end{array} \tag{85}$$

where $g_i(\mathbf{x})$ are convex functions.

- How can every convex optimization problem be presented in this form? For objective function $f(\mathbf{x})$, translate it into a constraint $f(\mathbf{x}) - c \leq 0$ and minimize c
- Let $\mathbf{s}_j(\mathbf{x}^i)$ be a subgradient for g_j at $\mathbf{x}^i$. By definition of subgradient $g_j(\mathbf{x}) \geq g_j(\mathbf{x}^i) + \mathbf{s}_j^T(\mathbf{x}^i)(\mathbf{x} - \mathbf{x}^i)$ for all $\mathbf{x} \in dom(g_j)$. [Eg: $\mathbf{s}_j(\mathbf{x}^i)$ could be $\nabla g_j(\mathbf{x}^i)$]

Use subgradient based linear lower bound function as a necessary linear inequality

[16] All convex optimization problems of the form discussed so far can be cast in this form.

Cutting Plane Algorithm (contd.)

- If point $\mathbf{x}^i$ is feasible, *i.e.*, $g_j(\mathbf{x}^i) \leq 0$ then

Cutting Plane Algorithm (contd.)

- If point $\mathbf{x}^i$ is feasible, *i.e.*, $g_j(\mathbf{x}^i) \leq 0$ then $0 \geq g_j(\mathbf{x}^i) + \mathbf{s}_j^T(\mathbf{x}^i)(\mathbf{x} - \mathbf{x}^i)$ for all $\mathbf{x} \in dom(g_j)$
- When the last inequality is enumerated for all values of i and j, we get several linear constraints:
  $\mathbf{s}_j^T(\mathbf{x}^i)\mathbf{x} \leq \mathbf{s}_j^T(\mathbf{x}^i)\mathbf{x}^i - g_j(\mathbf{x}^i)$ for fixed i and all j and $\mathbf{x} \in dom(g_j) \equiv A_i\mathbf{x} \leq A_i\mathbf{x}^i - \mathbf{g}_i$

$$\boxed{A_i = \begin{bmatrix} \mathbf{s}_1(\mathbf{x}^i) \\ \mathbf{s}_2(\mathbf{x}^i) \\ . \\ . \\ \mathbf{s}_m(\mathbf{x}^i) \end{bmatrix} \quad \mathbf{g}_i = \begin{bmatrix} g_1(\mathbf{x}^i) \\ g_2(\mathbf{x}^i) \\ . \\ . \\ g_m(\mathbf{x}^i) \end{bmatrix}} \tag{86}$$

All are for a fixed point x^i

Cutting Plane Algorithm (contd.)

- Stacking all the A_i's and $\mathbf{g}_i$'s together

Stack for different points xi

$$A^k = \begin{bmatrix} A_0 \\ A_1 \\ . \\ . \\ A_k \end{bmatrix} \quad \mathbf{b}^k = \begin{bmatrix} A_0\mathbf{x}^0 - \mathbf{g}_0 \\ A_1\mathbf{x}^1 - \mathbf{g}_1 \\ . \\ . \\ A_k\mathbf{x}^k - \mathbf{g}_k \end{bmatrix} \tag{87}$$

- With this, the necessary feasible conditions are: $A^k\mathbf{x} \leq \mathbf{b}^k$.
- Idea: Solve the following LP iteratively, until all original constraints are respected:

Need to solve iteratively because linear inequality was only necessary
Original constraints may still be violated

$$\mathbf{x}_*^k = \begin{array}{ll} \underset{\mathbf{x}}{\text{argmin}} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A^k\mathbf{x} \leq \mathbf{b}^k \end{array}$$

Kelly's Cutting Plane Algorithm (contd.)

Step 1
Input an initial feasible point, $\mathbf{x}^0$ and set $k = 0$.
Step 2: Evaluate A^k and $\mathbf{b}^k$
Step 3
Solve the LP problem

$$\mathbf{x}_*^k = \begin{array}{ll} \underset{\mathbf{x}}{\text{argmin}} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A^k\mathbf{x} \leq \mathbf{b}^k \end{array}$$

Look for violations
of original constraints

Step 4
If $\max\{g_j(\mathbf{x}_*^k),\ 1 \leq j \leq m\}$ $(\epsilon$ output $\mathbf{x}_* = \mathbf{x}_*^k$ as the point of optimality and stop. Otherwise, set $k = k + 1$, $\mathbf{x}^{k+1} = \mathbf{x}_*^k$, update A^k and $\mathbf{b}^k$ from (87) using (86) and repeat from **Step 3.**

Figure 29: Optimization for the convex problem in (85) using Kelly's cutting plane algorithm.