Quadratic Program Specification w/ Gradient Derivation

S. Vasserman and C. Margossian

June 2, 2018

We characterize the solution to a quadratic program with one linear equality constraint and a lower bound of zero on all elements.

1 Quadratic Program Description

We consider a constrained quadratic program specified in the standard form:

$$\min_{\mathbf{x}}\{\frac{1}{2}\mathbf{x}'H\mathbf{x} + \mathbf{f}'\mathbf{x}\} \tag{1}$$

$$\text{s.t. } A \cdot \mathbf{x} = b_{eq} \tag{2}$$

$$\mathbf{x} \geq \mathbf{0} \tag{3}$$

where

- H is an $n \times n$ positive semi-definite matrix
- $\mathbf{f}$ is an n-dimensional vector
- A is an n-dimensional vector[1]

[1] We might want to relax this for any number of linear constraints (up to n) at a later time

2 Semi-Analytical Gradient

In this section, we derive the solution to our quadratic program of interest using the KKT optimality constraints. Note that although we can write down the solution analytically, this solution will rely on Lagrange multipliers on the inequality constraints (that $x_i \geq 0$ for each $i = 1, \ldots, n$. Figuring out which of the constraints bind and which don't is what requires an iterative algorithm. Once we know which constraints bind at the optimum (meaning for which i, the optimal solution has that $x_i = 0$), the optimal solution using the remaining non-zero dimensions can be described analytically.

The quadratic program in equation (1) can be reformulated with the following Lagrangian:

$$\mathcal{L} = \frac{1}{2}\mathbf{x}'H\mathbf{x} + \mathbf{f}'\mathbf{x} - \lambda'\left(A \cdot \mathbf{x} - b_{eq}\right) - \sum_{i=1}^{n} \omega_i x_i \tag{4}$$

where λ is the Lagrangian multiplier on the equality constraint and ω_i is the Lagrangian multiplier on each inequality constraint (imposing that each bid is non-negative).

The optimal bid vector $\mathbf{x}^* = \arg\min_{\mathbf{x}} \mathcal{L}$ is therefore implicitly defined by the first order conditions:

$$\frac{\partial \mathcal{L}}{\partial x_i}(\mathbf{x}^*) = \frac{1}{2}\left[\sum_{j \neq i} x_j^* H_{ij} + 2x_i^* H_{ii}\right] + f_i - \lambda A_i - \omega_i = 0 \text{ for each } i = 1, \ldots, n \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial \lambda}(\mathbf{x}^*) = \sum_i A_i x_i - b = 0 \tag{6}$$

$$\frac{\partial \mathcal{L}}{\partial \omega_i}(\mathbf{x}^*) = -x_i^* = 0, \text{ if } \omega_i > 0. \tag{7}$$

Rearranging equation 5, we obtain:

$$\mathbf{x}_i^* = \frac{\omega_i + \lambda A_i - f_i - \frac{1}{2}\sum_{j \neq i} x_j^* H_{ij}}{H_{ii}} \text{ for each } i = 1, \ldots, n \tag{8}$$

Note that this still has the Lagrange multipliers λ and ω_i inside the definition of x_i^*. However, given a [numerical] solution to $\mathbf{x}^*$, the values of the Lagrange multipliers is pinned down (this is because the Lagrange multipliers are the solution to the dual problem of the quadratic

program).

In particular, by definition of the inequality constraints: $x_i \geq 0$, we have that for each i, either $x_i^* > 0$ (the constraint does not bind at i), in which case $\omega_i = 0$, or $x_i^* = 0$, in which case we obtain:

$$\omega_i = \frac{1}{2}\sum_{j \neq i} x_j^* H_{ij} + f_i - \lambda A_i$$

by plugging $\mathbf{x}^*$ into equation 5.

Getting λ is a bit trickier and may not be necessary as the numerical quadratic solver should be able to output the Lagrange multipliers on the equality constraints at the optimum together with $\mathbf{x}_i^*$ (note that this is an option in matlab's quadprog function). However, we need to be able to calculate the gradient of the Lagrange multipliers with respect to the parameters of the problem, and so we do this analytically below.

By plugging in the definition of $\mathbf{x}^*$ from equation 8 to the KKT condition in equation 6, we obtain:

$$\sum_i A_i \left[\frac{1}{H_{ii}} (\omega_i + \lambda A_i - f_i - \frac{1}{2}\sum_{j \neq i} x_j^* H_{ij}) = b \right]$$

Plugging in the definition of ω_i from above, and then rearranging and simplifying, we obtain:

$$\lambda = \frac{b + \sum\limits_{i:x_i^*>0} \frac{A_i}{H_{ii}} \left(f_i + \frac{1}{2}\sum_{j \neq i} x_j^* H_{ij} \right)}{\sum\limits_{i:x_i^*>0} \frac{A_i^2}{H_{ii}}}. \tag{9}$$

where the subscript $\{i : x_i^* > 0\}$ refers to those elements i for which the optimal x_i^* is strictly positive (i.e. the inequality constraint does not bind).

We can then plug the definitions of ω and λ into equation 8 to obtain an analytical solution to $\mathbf{x}_i^*$ for each element i without any Lagrange parameters (but still in terms of the solution of the remaining elements x_j^*, $j \neq i$, which are evaluated at the numerically computed optimum):

$$x_i^* = \frac{A_i \left(\frac{b + \sum\limits_{i:x_i^*>0} \frac{A_i}{H_{ii}} \left(f_i + \frac{1}{2}\sum_{j \neq i} x_j^* H_{ij} \right)}{\sum\limits_{i:x_i^*>0} \frac{A_i^2}{H_{ii}}} \right) - f_i - \frac{1}{2}\sum\limits_{j \neq i} x_j^* H_{ij}}{H_{ii}} \tag{10}$$

if $x_i^* > 0$. Otherwise, $x_i^* = 0$ and its derivative with respect to all parameters is zero as well.

3 Function Specification

For the general constraint conditions:

$$\mathbf{h}(\mathbf{x}) = \mathbf{0}$$
$$\mathbf{g}(\mathbf{x}) \geq \mathbf{0}$$

calling a numerical optimizer requires the user to pass H, f, and two functions h and g. The call may look as follows:

```
x = solver(H, f, A, h, g);
```

We would like to solve the optimization problem and compute the Jacobian matrix of the solution with respect to the auxiliary parameters θ. In Stan, variables may depend on model parameters (in this case θ) and fixed variables, δ, which can be stored in two arrays, `theta` and `delta`. The arguments of the solver then all depend on θ and δ. For example $\mathbf{H} = \mathbf{H}(\theta, \delta)$.

Given this, we require the user to pass the four original arguments as functions and two additional arguments which store θ and δ:

```
x = solver(H, f, h, g, theta, delta);
```

where the first arguments observe a strict signature. For example:

```
matrix H(theta, delta) {...}
vector f(theta, delta) {...}
```

From a user's perspective, the overhead is rather severe, as the optimizer requires the definition of 4 new functions, and construction of the arrays `theta` and `delta`. To alleviate this, we could overload the function to accept fixed variables arguments, instead of functions. We may also start with a more specialized function, as a proof of concept. For now, we focus on a special case of the KKT problem, as previously described. The inequality constraint is a

linear one of the form:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}_{\text{eq}}$$

and the inequality constraint simply requires x to have positive elements:

$$x_i \geq 0, \forall i$$

In this scenario, the function signature should look as

```
x = solver(H, f, A, b_eq, is_positive, theta, delta);
```

which unfortunately remains very cumbersome.

At a C++ level, we have (i) an evaluation function:

```
Evaluate(H, f, A, b_eq, is_positive, theta, delta) {
 MatrixXd H_value = value_of(H(theta, delta));
 VectorXd f_value = value_of(f(theta, delta));
 .  .  .
  x = quad_prog(H_value, f_value, A_value, is_positive);
 .  .  .
  return x;
}
```

and (ii) a new `vari` class with a custom chain method:

```
chain() {
 Optimizer_functor x_closed(x, delta);
 MatrixXd J;
 begin_nested();
 Jacobian(x_closed, theta, .  .  ., J);
 end_nested();
```

. . .

}

The `Optimizer_functor` defines a class of functions which return an analytical expression for x, for a given θ. Note that in order to construct this analytical expression, we first need to know which elements of x are non-zero, a task for which we require the numerical optimizer.