nb-2021-03-11

March 11, 2021

1 Supplementary notes for 2021-03-11

```
[1]: using LinearAlgebra
     using Plots
```

1.1 Consider constraints

So far, we have considered *unconstrained* optimization problems. The *constrained* problem is

$$\text{minimize } \phi(x) \text{ s.t. } x \in \Omega$$

where $\Omega \subset \mathbb{R}^n$. We usually define x in terms of a collection of constraint equations and inequalities:

$$\Omega = \{x \in \mathbb{R}^n : c_i(x) = 0, i \in \mathcal{E} \text{ and } c_i(x) \leq 0, i \in \mathcal{I}\}.$$

We will suppose throughout our discussions that both ϕ and all the functions c are differentiable.

If x_* is a solution to the constrained minimization problem, we say constraint $i \in \mathcal{I}$ is *active* if $c_i(x) = 0$. Often, the hard part of solving constrained optimization problems is figuring out which constraints are active. From this perspective, the equality constrained problem sits somewhere in difficulty between the unconstrained problem and the general constrained problem.

Our treatment of constrained optimization is necessarily brief; but in the next two lectures, I hope to lay out some of the big ideas. Today we will focus on formulations; next time, algorithms.

1.2 Three recipes

Most methods for constrained optimization involve a reduction to an unconstrained problem (or subproblem). There are three ways such a reduction might work:

- We might *remove* variables by eliminating constraints.
- We might keep the *same* number of variables and try to fold the constraints into the objective function.
- We might *add* variables to enforce constraints via the method of Lagrange multipliers.

These approaches are not mutually exclusive, and indeed one often alternates between perspectives in modern optimization algorithms.

1.3 Constraint elimination

The idea of constraint elimination is straightforward. Suppose we want to solve an optimization problem with only equality constraints: $c_i(x) = 0$ for $i \in \mathcal{E}$, where $|\mathcal{E}| < n$ and the constraints are independent – that is, the $|\mathcal{E}| \times n$ Jacobiam matrix $\partial c/\partial x$ has full row rank. Then we can think (locally) of x satisfying the constraints in terms of an implicitly defined function $x = g(y)$ for $y \in \mathbb{R}^{n-|\mathcal{E}|}$. If this characterization can be made global, then we can solve the unconstrained problem

$$\text{minimize } \phi(g(y))$$

over all $y \in \mathbb{R}^{n-|\mathcal{E}|}$.

The difficulty with constraint elimination is that it requires that we find a global parameterization of the solutions to the constraint equations. This is usually difficult. An exception is when the constraints are *linear*:

$$c(x) = A^T x - b$$

In this case, the feasible set $\Omega = \{x : A^T x - b = 0\}$ can be written as $x \in \{x^p + z : z \in \mathcal{N}(A)\}$, where x^p is a *particular solution* and $\mathcal{N}(A)$ is the null space of A. We can find both a particular solution and the null space by doing a full QR decomposition on A:

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

Then solutions to the constraint equations have the form

$$x = A^\dagger b + Q_2 y = Q_1 R_1^{-T} b + Q_2 y$$

where the first term is a particular solution and the second term gives a vector in the null space.

For problems with linear equality constraints, constraint elimination has some attractive properties. If there are many constraints, the problem after constraint elimination may be much smaller. And if the original problem was convex, then so is the reduced problem, and with a better-conditioned Hessian matrix. The main drawback is that we may lose sparsity of the original problem. Constraint elimination is also attractive for solving equality-constrained subproblems in optimization algorithms for problems with linear *inequality* constraints, particularly if those constraints are simple (e.g. elementwise non-negativity of the solution vector).

For problems with more complicated equality constraints, constraint elimination is hard. Moreover, it may not be worthwhile; in some cases, eliminating constraints results in problems that are smaller than the original formulation, but are harder to solve.

The idea of constraint elimination is not limited to equality constraints: one can also sometimes use an alternate parameterization to convert simple inequality-constrained problems to unconstrained problems. For example, if we want to solve a non-negative optimization problem (all $x_i \geq 0$), we might write $x_i = y_i^2$, or possibly $x_i = \exp(y_i)$ (though in this case we would need to let $y_i \to -\infty$ to exactly hit the constraint). But while they eliminate constraints, these re-parameterizations can also destroy nice features of the original problem (e.g. convexity). So while such transformations are a useful part of the computational arsenal, they should be treated as one tool among many, and not always as the best tool available.

1.4 Penalties and barriers

Constraint elimination methods convert a constrained to an unconstrained problem by changing the coordinate system in which the problem is posed. Penalty and barrier methods accomplish the same reduction to the unconstrained case by changing the function.

As an example of a *penalty* method, consider the problem

$$\text{minimize } \phi(x) + \frac{1}{2\mu}\sum_{i\in\mathcal{E}} c_i(x)^2 + \frac{1}{2\mu}\sum_{i\in\mathcal{I}} \max(c_i(x), 0)^2.$$

When the constraints are violated ($c_i > 0$ for inequality constraints and $c_i \neq 0$ for equality constraints), the extra terms (penalty terms) beyond the original objective function are positive; and as $\mu \to 0$, those penalty terms come to dominate the behavior outside the feasible region. Hence as we let $\mu \to 0$, the solutions to the penalized problem approach solutions to the original (true) problem. At the same time, as $\mu \to 0$ we have much wilder derivatives of ϕ, and the optimization problems become more and more problematic from the perspective of conditioning and numerical stability. Penalty methods also have the potentially undesirable property that if any constraints are active at the true solution, the solutions to the penalty problem tend to converge from *outside* the feasible region. This poses a significant problem if, for example, the original objective function ϕ is undefined outside the feasible region.

As an example of a *barrier* method, consider the purely inequality constrained case, and approximate the original constrained problem by the unconstrained problem

$$\text{minimize } \phi(x) - \mu \sum_{i\in\mathcal{I}} \log(-c_i(x)).$$

As $c_i(x)$ approaches zero from below, the barrier term $-\mu\log(-c_i(x))$ grows rapidly; but at any fixed x in the interior of the domain, the barrier goes to zero as μ goes to zero. Hence, as $\mu \to 0$ through positive values, the solution to the barrier problem approaches the solution to the true problem through a sequence of *feasible* points (i.e. approximate solutions that satisfy the constraints). Though the feasibility of the approximations is an advantage over penalty based formulations, interior formulations share with penalty formulations the disadvantage that the solutions for $\mu > 0$ lie at points with increasingly large derivatives (and bad conditioning) if the true solution has active constraints.

There are *exact penalty* formulations for which the solution to the penalized problem is an exact solution for the original problem. Suppose we have an inequality constrained problem in which the feasible region is closed and bounded, each constraint c_i has continuous derivatives, and $\nabla c_i(x) \neq 0$ at any boundary point x where constraint i is active. Then the solution to the problem

$$\text{minimize } \phi(x) + \frac{1}{\mu}\sum_i \max(c_i(x), 0)$$

is *exactly* the solution to the original constrained optimization problem for some $\mu > 0$. In this case, we used a *nondifferentiable* exact penalty, but there are also exact differentiable penalties.

1.5 Lagrange multipliers

Picture a function $\phi : \mathbb{R}^n \to \mathbb{R}$; if you'd like to be concrete, let $n = 2$. Absent a computer, we might optimize of ϕ by the physical experiment of dropping a tiny ball onto the surface and watching it roll downhill (in the steepest descent direction) until it reaches the minimum. If we wanted to solve a constrained minimization problem, we could build a great wall between the feasible and the infeasible region. A ball rolling into the wall would still roll freely in directions tangent to the wall (or away from the wall) if those directions were downhill; at a constrained miminizer, the force pulling the ball downhill would be perfectly balanced against an opposing force pushing into the feasible region in the direction of the normal to the wall. If the feasible region is $\{x : c(x) \leq 0\}$, the normal direction pointing inward at a boundary point x_* s.t. $c(x_*) = 0$ is proportional to $-\nabla c(x_*)$. Hence, if x_* is a constrained minimum, we expect the sum of the "rolling downhill" force $(-\nabla \phi)$ and something proportional to $-\nabla c(x_*)$ to be zero:

$$-\nabla \phi(x_*) - \mu \nabla c(x_*) = 0.$$

The *Lagrange multiplier* μ in this picture represents the magnitude of the restoring force from the wall balancing the tendency to roll downhill.

More abstractly, and more generally, suppose that we have a mix of equality and inequality constraints. We define the *Lagrangian*

$$L(x, \lambda, \mu) = \phi(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \sum_{i \in \mathcal{I}} \mu_i c_i(x).$$

The *Karush-Kuhn-Tucker (KKT) conditions* for x_* to be a constrained minimizer are

$$\begin{aligned}
\nabla_x L(x_*) &= 0 && && \\
c_i(x_*) &= 0, && i \in \mathcal{E} && \text{equality constraints} \\
c_i(x_*) &\leq 0, && i \in \mathcal{I} && \text{inequality constraints} \\
\mu_i &\geq 0, && i \in \mathcal{I} && \text{non-negativity of multipliers} \\
c_i(x_*)\mu_i &= 0, && i \in \mathcal{I} && \text{complementary slackness}
\end{aligned}$$

where the (negative of) the "total force" at x_* is

$$\nabla_x L(x_*) = \nabla \phi(x_*) + \sum_{i \in \mathcal{E}} \lambda_i \nabla c_i(x_*) + \sum_{i \in \mathcal{I}} \mu_i \nabla c_i(x_*).$$

The complementary slackness condition corresponds to the idea that a multiplier should be nonzero only if the corresponding constraint is active (a "restoring force" is only present if our test ball is pushed into a wall).

Like the critical point equation in the unconstrained case, the KKT conditions define a set of (necessary but not sufficient) nonlinear algebraic equations that must be satisfied at a minimizer. Because of the multipliers, we have *more* variables than were present in the original problem. However, the Jacobian matrix (KKT matrix)

$$J = \begin{bmatrix} \nabla_x^2 L(x_*) & \nabla c \\ (\nabla c)^T & 0 \end{bmatrix}$$

has a saddle point structure even when $\nabla_x^2 \phi$ is positive definite. Also, unlike the penalty and barrier approaches described before, the Lagrange multiplier approach requires that we figure out which multipliers are active or not — an approach that seems to lead to a combinatorial search in the worst case.

1.6 Lay of the Land

As we mentioned before, problems with *inequality* constraints tend to be more difficult than problems with *equality* constraints alone, because it involves the combinatorial subproblem of figuring out which constraints are *active* (a constraint $c_i(x) \leq 0$ is active if $c_i(x) = 0$ at the optimum). Once we have figured out the set of active constraints, we can reduce an inequality-constrained problem to an equality-constrained problem. Hence, the purely equality-constrained case is an important subproblem for inequality-constrained optimizers, as well as a useful problem class in its own right.

For problems with only equality constraints, there are several standard options:

- *Null space methods* deal with linear equality constraints by reducing to an unconstrained problem in a lower-dimensional space.
- *Projected gradient methods* deal with simple equality constraints by combining a (scaled) gradient step and a projection onto a constraint set.
- *Penalty methods* approximately solve an equality-constrained problem through an unconstrained problem with an extra term that penalizes proposed soutions that violate the constraints. That is, we use some constrained minimizer to solve

  $$\text{minimize } \phi(x) + \frac{1}{\mu} \sum_{i \in \mathcal{E}} c_i(x)^2.$$

  As $\mu \to 0$, the minimizers to these approximate problems approach the true minimizer, but the Hessians that we encounter along the way become increasingly ill-conditioned (with condition number proportional to μ^{-1}).
- *KKT solvers* directly tackle the first-order optimality conditions (the KKT conditions), simultaneously computing the constrained minimizer and the associated Lagrange multipliers.
- *Augmented Lagrangian* methods combine the advantages of penalty methods and the advantages of the penalty formulation. In an augmented Lagrangian solver, one finds critical points for the augmented Lagrangian

  $$\mathcal{L}(x, \lambda; \mu) = \phi(x) + \frac{1}{\mu} \sum_{i \in \mathcal{E}} c_i(x)^2 + \lambda^T c(x)$$

  by alternately adjusting the penalty parameter μ and the Lagrange multipliers.

In the inequality-constrained case, we have

- *Active set methods* solve (or approximately solve) a sequence of equality-constrained subproblems, shuffling constraints into and out of the proposed working set along the way. These methods are particularly attractive when one has a good initial estimate of the active set.
- *Projected gradient methods* deal with simple inequality constraints by combining a (scaled) gradient step and a projection onto a constraint set.
- *Barrier methods* and *penalty methods* add a term to the objective function in order to penalize constraint violations or near-violations; as in the equality-constrained case, a parameter μ governs a tradeoff between solution quality and conditioning of the Hessian matrix.
- *Interior point methods* solve a sequence of barrier subproblems using a continuation strategy, where the barrier or penalty parameter μ is the continuation parameter. This is one of the

most popular modern solver strategies, though active set methods may show better performance when one "warm starts" with a good initial guess for the solution and the active set of constraints.

As with augmented Lagrangian strategies in the equality-constrained case, state-of-the art strategies for inequality-constrained problems often combine approaches, using continuation with respect to a barrier parameters as a method of determining the active set of constraints in order to get to an equality-constrained subproblem with a good initial guess for the solution and the Lagrange multipliers.

The *sequential quadratic programming* (SQP) approach for nonlinear optimization solves a sequence of linearly-constrained quadratic optimization problems based on Taylor expansion of the objective and constraints about each iterate. This generalizes simple Newton iteration for unconstrained optimization, which similarly solves a sequence of quadratic optimization problems based on Taylor expansion of the objective. Linearly-constrained quadratic programming problems are hence an important subproblem in SQP solvers, as well as being an important problem class in their own right.

1.7 Quadratic programs with equality constraints

We begin with a simple case of a quadratic objective and linear equality constraints:

$$\begin{aligned} \phi(x) &= \frac{1}{2}x^T H x - x^T d \\ c(x) &= A^T x - b = 0, \end{aligned}$$

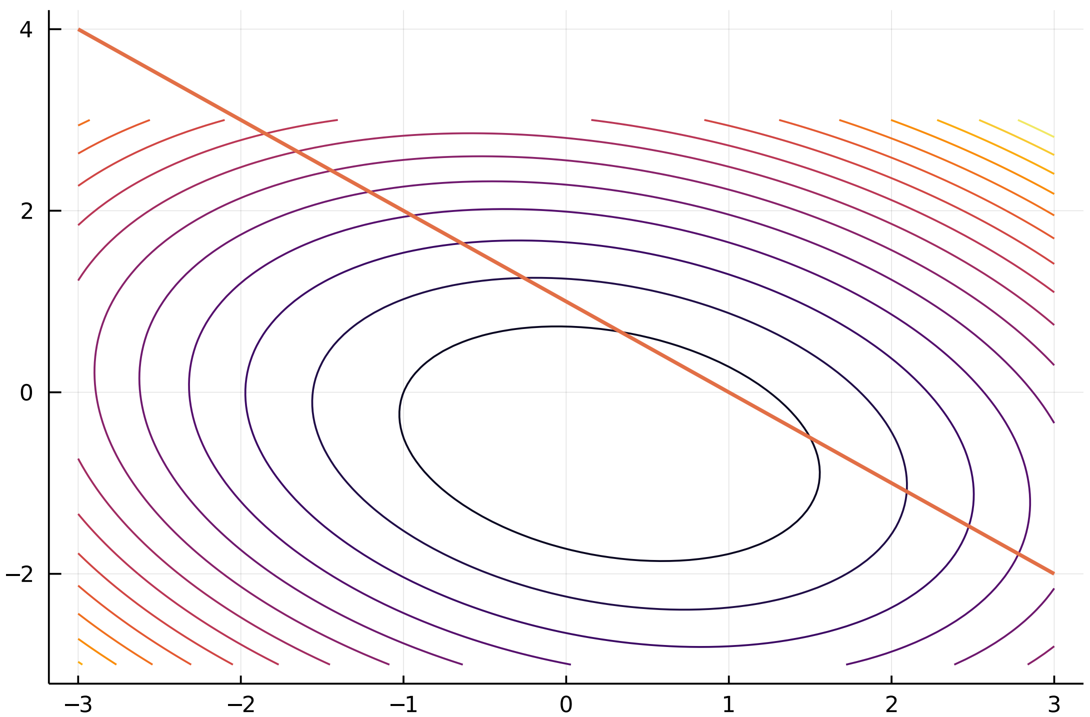
where $H \in \mathbb{R}^{n \times n}$ is symmetric and positive definite *on the null space of* A^T (it may be indefinite or singular overall), $A \in \mathbb{R}^{n \times m}$ is full rank with $m < n$, and $b \in \mathbb{R}^m$. Not only are such problems useful in their own right, solvers for these problems are also helpful building blocks for more sophisticated problems — just as minimizing an unconstrained quadratic can be seen as the starting point for Newton's method for unconstrained optimization.

[2]:
```
# Set up a test problem for linearly-constrained QP (2D so that we can plot)
H = [4.0  1.0 ;
     1.0  4.0 ]
d = [0.5 ; -2.0]
A = [1.0 ; 1.0]
b = [1.0]

 1(xy) = xy'*H*xy/2 - xy'*d
c1(xy) = A'*x - b

xx = range(-3, 3, length=100)
plot(xx, xx, (x,y) -> 1([x; y]), st=:contour, legend=false)
plot!(xx, 1.0 .- xx, linewidth=2)
```

[2]:



1.7.1 Constraint elimination (linear constraints)

As discussed last time, we can write the space of solutions to the constraint equations in terms of a (non-economy) QR decomposition of A:

$$A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

where Q_2 is a basis for the null space of A^T. The set of solutions satisfying the constraints $A^T x = b$ is

$$\Omega = \{u + Q_2 y : y \in \mathbb{R}^{(n-m)}, u = Q_1 R_1^{-T} b\};$$

here u is a *particular solution* to the problem. If we substitute this parameterization of Ω into the objective, we have the unconstrained problem

$$\text{minimize } \phi(u + Q_2 y).$$

While we can substitute directly to get a quadratic objective in terms of y, it is easier (and a good exercise in remembering the chain rule) to compute the stationary equations

$$\begin{aligned} 0 = \nabla_y \phi(u + Q_2 y) &= \left(\frac{\partial x}{\partial y}\right)^T \nabla_x \phi(u + Q_2 y) \\ &= Q_2^T (H(Q_2 y + u) - d) = (Q_2^T H Q_2) y - Q_2^T (d - Hu). \end{aligned}$$

In general, even if A is sparse, Q_2 may be dense, and so even if H is dense, we find that $Q_2^T H Q_2$ is dense.

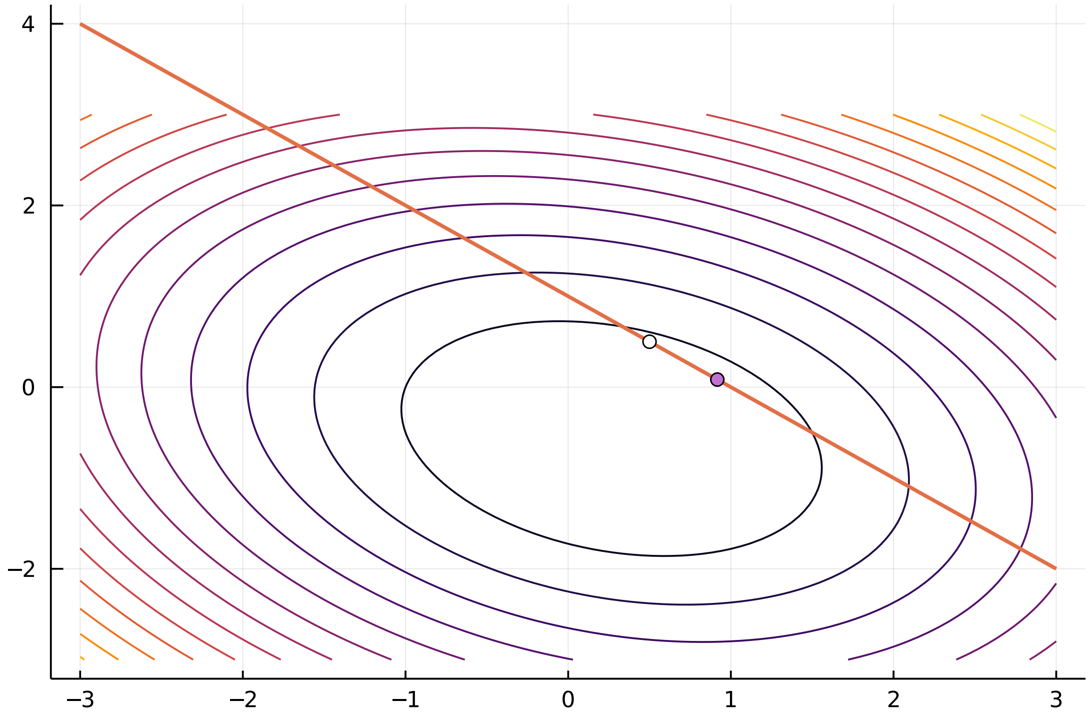
[3]:
```
# Solve the 2-by-2 problem via a null-space approach

F = qr(A)
Q = F.Q * I
Q1 = Q[:,[1]]
Q2 = Q[:,[2]]

u = Q1*(F.R'\b)
H22 = Q2'*H*Q2
r2  = Q2'*(d-H*u)
y   = H22\r2
x   = u + Q2*y

xx = range(-3, 3, length=100)
plot(xx, xx, (x,y) -> 1([x; y]), st=:contour, legend=false)
plot!(xx, 1.0 .- xx, linewidth=2)
plot!([u[1]], [u[2]], markercolor=:white, marker=true)
plot!([x[1]], [x[2]], marker=true)
```

[3]:



Finding a particular solution and a null space basis via QR is great for numerical stability, but it may not be ideal when the matrices involved are sparse or structured. An alternative is to use a sparse LU factorization of A^T:

$$PA^TQ = L\left[U_1 U_2\right].$$

where the U_1 submatrix is upper triangular. A particular solution is then

$$x = Q\begin{bmatrix} U_1^{-1}L^{-1}Pb \\ 0 \end{bmatrix}$$

and the null space is spanned by

$$Q^T\begin{bmatrix} -U_1^{-1}U_2 \\ I \end{bmatrix}$$

This reformulation may be particularly attractive if A is large, sparse, and close to square. Note that pivoting on rectangular constraint matrices needs to be done carefully, e.g. using so-called *rook pivoting* strategies that maintain numerical stability on full-rank matrices with rank-deficient submatrices.

1.7.2 Projected gradient and conjugate gradients

The *projected gradient* is a variant of gradient descent for constrained problem. One assumes that we have a projection operator P such that $P(x)$ is the closest point to x satisfying the constraint; the iteration is then

$$x_{k+1} = P\left(x_k - \alpha_k \nabla\phi(x_k)\right).$$

That is, we take an (unconstrained) gradient descent step, then project back to satisfy the constraint. It's an easy enough method to code, provided you have the projection P.

For our linear equality constraints the projection can be computed by a least squares type of solve:

$$\begin{aligned} P(x) &= x + A(A^TA)^{-1}(b - A^Tx) \\ &= (A^T)^\dagger b + (I - AA^\dagger)x \\ &= (A^T)^\dagger b + (I - \Pi)x \end{aligned}$$

Note that $(A^T)^\dagger b$ is the minimal norm solution to the constraint equation, and the range space of $I - \Pi = I - AA^\dagger$ is the null space of A^T, so this is similar to the picture we saw with the constraint elimination approach. And, of course, the gradient in this case is just the residual $r_k = Hx_k - d$.

If we start with a point x_0 that is consistent with the constraints, then each successive point remains on our linear constraint surface; in this case, we can simplify the iteration to

$$x_{k+1} = x_k - \alpha_k(I - \Pi)r_k$$

This is a stationary iteration for the underdetermined consistent equation

$$(I - \Pi)(Hx_k - d) = 0.$$

Unfortunately, the projected gradient iteration may converge rather slowly. A tempting thought is to use a scaled version of the gradient, but the naive version of this iteration will in general converge to the wrong point unless the projection operator is re-defined in terms of the distance associated with the same scaling matrix.

If the relevant projection is available, a potentially more attractive route for this problem is to write $x = u + z$ for some particular solution u (as in the null space approach) and then use a method like conjugate gradients on the system

$$(I - \Pi)H(I - \Pi)z = (I - \Pi)(d - Hu)$$

It turns out that the Krylov subspace generated by this iteration remains consistent with the constraint, and so -- somewhat surprisingly at first glance -- the method continues to work even though $(I - \Pi)H(I - \Pi)$ is singular.

1.7.3 Penalties and conditioning

Now consider a penalty formulation of the same equality-constrained optimization function, where the penalty is quadratic:

$$\text{minimize } \phi(x) + \frac{1}{2\mu}\|A^T x - b\|^2.$$

In fact, the augmented objective function is again quadratic, and the critical point equations are

$$(H + \mu^{-1}AA^T)x = d + \mu^{-1}Ab.$$

If μ is small enough and the equality-constrained quadratic program (QP) has a minimum, then $H+\mu^{-1}AA^T$ is guaranteed to be positive definite. This means we can solve via Cholesky; or (if the linear system is larger) we might use conjugate gradients.

We can analyze this more readily by changing to the Q basis from the QR decomposition of A that we saw in the constraint elimination approach:

$$\begin{bmatrix} Q_1^T H Q_1 + \mu^{-1} R_1 R_1^T & Q_1^T H Q_2 \\ Q_2^T H Q_1 & Q_2^T H Q_2 \end{bmatrix} (Q^T x) = \begin{bmatrix} Q_1^T d + \mu^{-1} R_1 b \\ Q_2^T d \end{bmatrix}$$

Taking a Schur complement, we have

$$(\mu^{-1} R_1 R_1^T + F)(Q_1^T x) = \mu^{-1} R_1 b - g$$

where

$$\begin{aligned} F &= Q_1^T H Q_1 - Q_1^T H Q_2 (Q_2^T H Q_2)^{-1} Q_2^T H Q_1 \\ g &= [I - Q_1^T H Q_2 (Q_2^T H Q_2)^{-1} Q_2^T] d \end{aligned}$$

As $\mu \rightarrow 0$, the first row of equations is dominated by the μ^{-1} terms, and we are left with

$$R_1 R_1^T (Q_1^T x) - R_1 b \rightarrow 0$$

i.e. $Q_1 Q_1^T x$ is converging to $u = Q_1 R_1^{-T} b$, the particular solution that we saw in the case of constraint elimination. Plugging this behavior into the second equation gives

$$(Q_2^T H Q_2)(Q_2^T x) - Q_2^T (d - Hu) \rightarrow 0,$$

i.e. $Q_2^T x$ asymptotically behaves like y in the previous example. We need large μ to get good results if the constraints are ill-posed or if $Q_2^T H Q_2$ is close to singular. But in general the condition number scales like $O(\mu^{-1})$, and so large values of μ correspond to problems that are numerically unattractive, as they may lead to large errors or (for iterative solvers) to slow convergence.

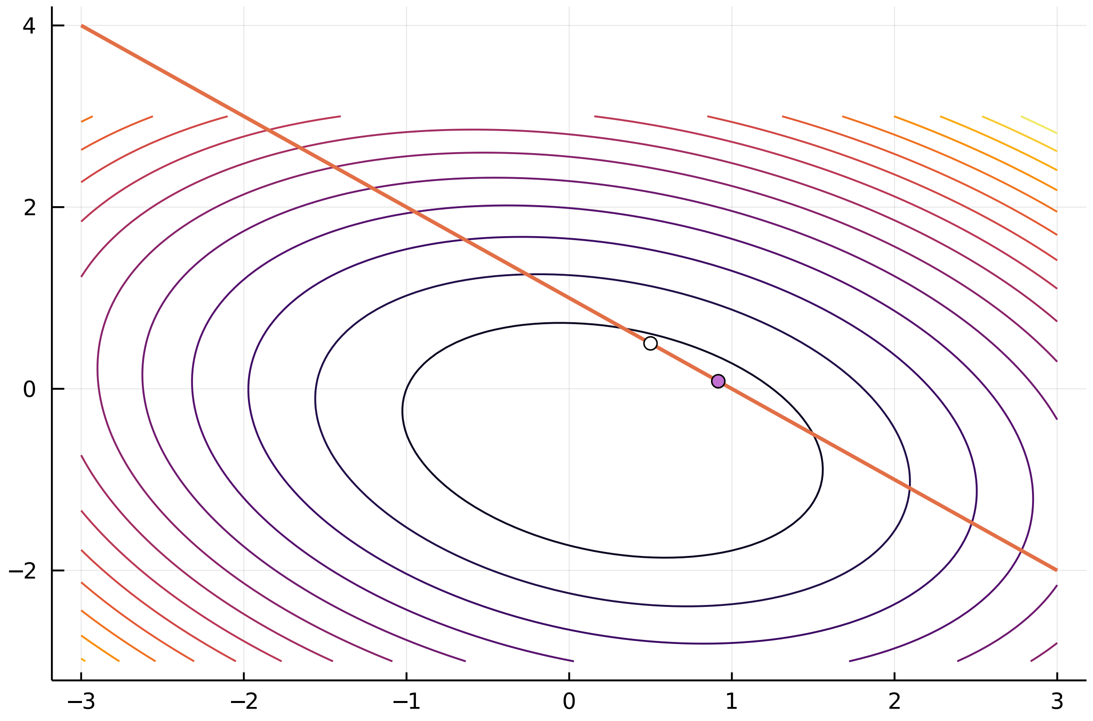
[4]:
```
# Demonstrate the solve with a moderate penalty

  = 1e-4
xhat = (H+A*A'/ )\(d+A*b[1]/ )
println("Error at  =$ : $(xhat-x)")

xx = range(-3, 3, length=100)
plot(xx, xx, (x,y) -> 1([x; y]), st=:contour, legend=false)
plot!(xx, 1.0 .- xx, linewidth=2)
plot!([u[1]], [u[2]], markercolor=:white, marker=true)
plot!([xhat[1]], [xhat[2]], marker=true)
```

```
Error at  =0.0001: [-0.00016245938500225598, -0.0001624593853049583]
```

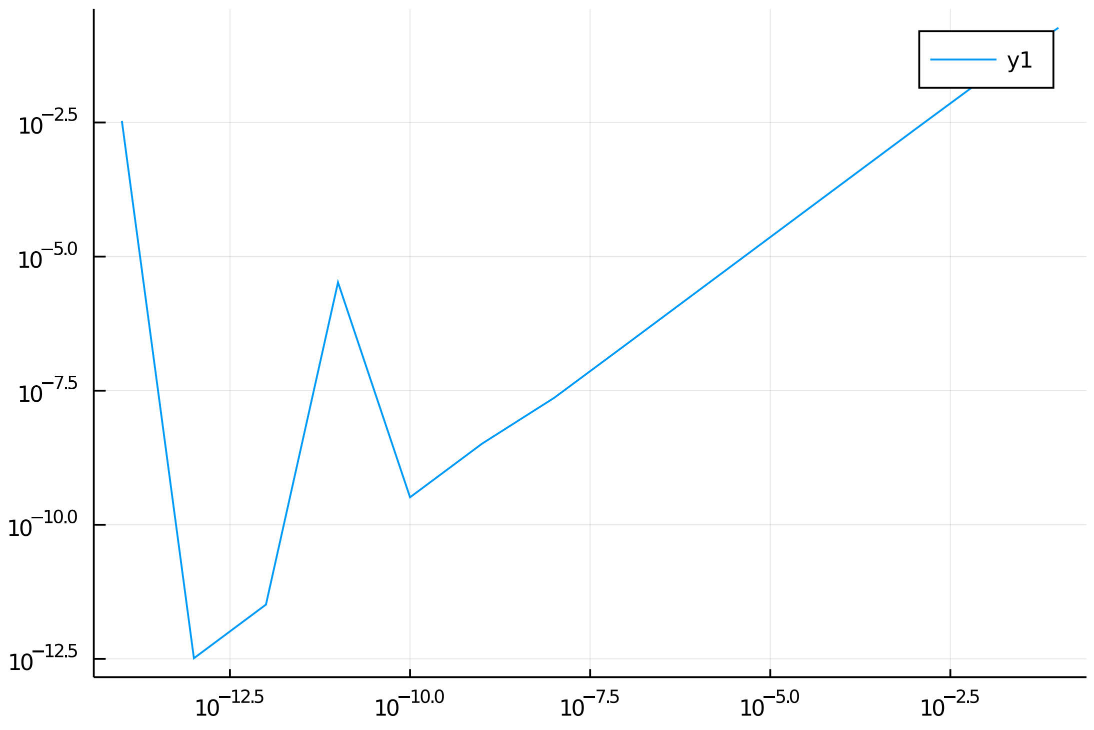
[4]:



[5]:
```
# Vary penalty to illustrate issues -- uniform improvement with smaller penalty␣
↪until ill-conditioning kills us
s = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11,␣
↪1e-12, 1e-13, 1e-14]
```

```
errs = []
for  in s
    xhat = (H+A*A'/ )\(d+A*b[1]/ )
    push!(errs, norm(xhat-x))
end
plot( s, errs, xscale=:log10, yscale=:log10)
```

[5]:



1.7.4 Lagrange multipliers and KKT systems

The KKT conditions for our equality-constrained problem say that the gradient of

$$L(x,\lambda) = \phi(x) + \lambda^T(A^T x - b)$$

should be zero. In matrix form, the KKT system (saddle point system)

$$\begin{bmatrix} H & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix}.$$

If A and H are well-conditioned, then so is this system, so there is no bad numerical behavior. The system also retains whatever sparsity was present in the original system matrices H and A. However, adding the Lagrange multipliers not only increases the number of variables, but the extended system lacks any positive definiteness that H may have.

When there are relatively few constraints and a fast solver with H is available, an attractive way to solve this KKT system is the so-called range-space method, which we recognize as just block Gaussian elimination:

$$\begin{aligned} A^T H^{-1} A \lambda &= A^T H^{-1} d - b \\ x &= H^{-1}(d - A\lambda) \end{aligned}$$

Rewritten as we might implement it, we have

$$\begin{aligned} Hx_0 &= d \\ HY &= A \\ (A^T Y)\lambda &= A^T x_0 - b \\ x &= x_0 - Y\lambda \end{aligned}$$

The KKT system is closely related to the penalty formulation that we saw in the previous subsection, in that if we use Gaussian elimination to remove the variable λ in

$$\begin{bmatrix} H & A \\ A^T & -\mu I \end{bmatrix} \begin{bmatrix} \hat{x} \\ \lambda \end{bmatrix} = \begin{bmatrix} d \\ b \end{bmatrix},$$

we have the Schur complement system

$$(H + \mu^{-1} A A^T)\hat{x} = d + \mu^{-1} A b,$$

which is identical to the stationary point condition for the quadratically penalized objective.

[6]:
```
x = [H A; A' 0.0] \ [d; b]
println("Error: $(x-x [1:end-1])")
println("  = $(x [end])")
println("  (x) = $(H*x [1:end-1]-d)")
println("  ×  c(x) = $(x [end]) × $A = $(x [end]*A)")
```

```
Error: [-2.220446049250313e-16, -6.938893903907228e-17]
 = -3.25
 (x) = [3.25, 3.25]
 ×  c(x) = -3.25 × [1.0, 1.0] = [-3.25, -3.25]
```

Note that the constrained stationarity condition

$$\nabla\phi(x_*) + \lambda \nabla c(x_*) = 0,$$

and we can use this to estimate the Lagrange multipliers from an approximation via a penalty method.

[7]:
```
  = 1e-4
xhat = (H+A*A'/ )\(d+A*b[1]/ )
r = H*xhat-d
println("   $(-norm(r)^2/(A'*r))")
```

```
   -3.2491877030742313
```

1.7.5 Uzawa iteration

Block Gaussian elimination is an attractive approach when we have a fast solver for H and there are not too many constraints. When there are a relatively large number of constraints, we might seek an alternate method. One such method is the *Uzawa iteration*

$$\begin{aligned} Hx_{k+1} &= d - A\lambda_k \\ \lambda_{k+1} &= \lambda_k + \omega(A^T x_{k+1} - b) \end{aligned}$$

where $\omega > 0$ is a relaxation parameter. We can eliminate x_{k+1} to get the iteration

$$\lambda_{k+1} = \lambda_k + \omega(A^T H^{-1}(d - A\lambda_k) - b) = (I - \omega A^T H^{-1} A)\lambda_k + \omega(A^T H^{-1} d - b),$$

which is a Richardson iteration on the Schur complement equation $(A^T H^{-1} A)\lambda = A^T H^{-1} d - b$. We can precondition and accelerate the Uzawa iteration in a variety of ways, as you might guess from our earlier discussion of iterative methods for solving linear systems.

1.7.6 Augmenting the Lagrangian

From a solver perspective, the block 2-by-2 structure of the KKT system looks highly attractive. Alas, we do *not* require that H be positive definite, nor even that it be nonsingular; to have a unique global minimum, we only need positive definiteness of the projection of H onto the null space (i.e. $Q_2^T H Q_2$ should be positive definite). This means we cannot assume that (for example) H will admit a Cholesky factorization.

The augmented Lagrangian approach can be seen as solving the constrained system

$$\text{minimize } \frac{1}{2}x^T Hx - d^T x + \frac{1}{2\mu}\|A^T x - b\|^2 \text{ s.t. } A^T x = b.$$

The term penalizing nonzero $\|A^T x - b\|$ is, of course, irrelevant at points satisfying the constraint $A^T x = b$. Hence, the constrained minimum for this augmented objective is identical to the constrained minimum of the original objective. However, if the KKT conditions for the modified objective take the form

$$\begin{bmatrix} H + \mu^{-1} AA^T & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} = \begin{bmatrix} d + \mu^{-1} Ab \\ b \end{bmatrix}.$$

Now we do not necessarily need to drive μ to zero to obtain a good solution; but by choosing μ small enough, we can ensure that $H + \mu^{-1} AA^T$ is positive definite (assuming that the problem is convex subject to the constraint). This can be helpful when we want to use a Cholesky factorization or a method like CG, but the original H matrix is indefinite or singular.

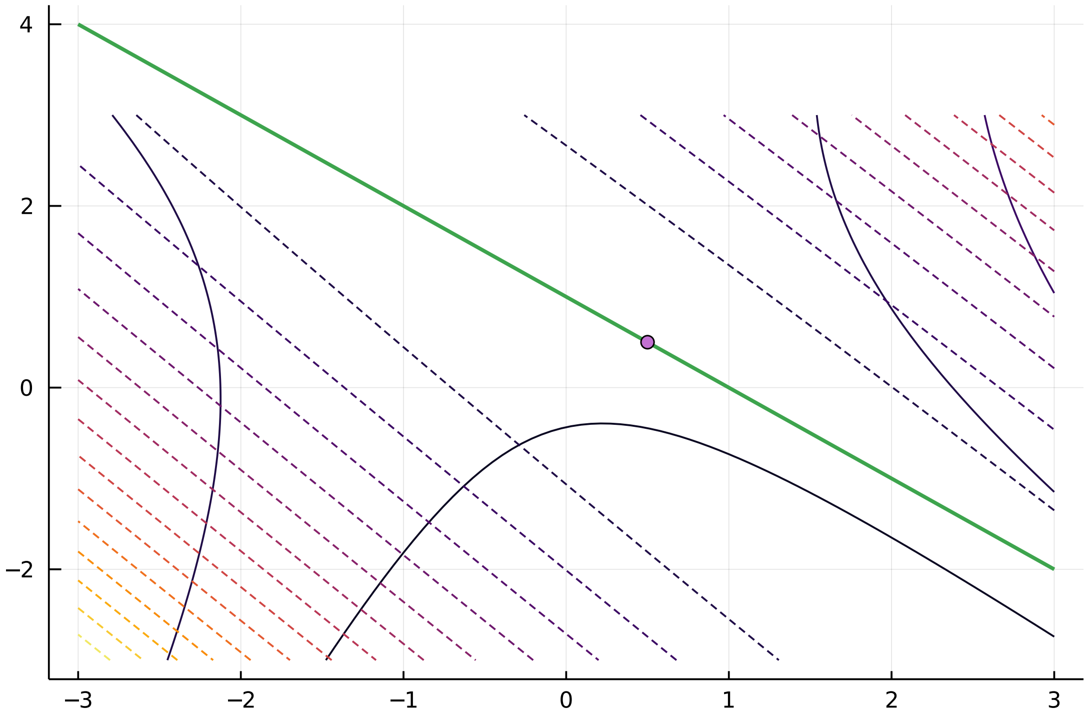
[8]:
```
# Set up a non-positive-definite problem for linearly-constrained QP (2D so␣
↪that we can plot)
H = [4.0  1.0 ;
     1.0  -1.0 ]
d = [0.5 ; -2.0]
A = [1.0 ; 1.0]
```

```
b = 1.0

 2(xy) = xy'*H*xy/2 - xy'*d
c2(xy) = A'*xy - b
 3(xy) =  2(xy) + 3*norm(c2(xy))^2
xy_sol = [H A; A' 0.0] \ [d; b]

xx = range(-3, 3, length=100)
plot(xx, xx, (x,y) ->  2([x; y]), st=:contour, legend=false)
plot!(xx, xx, (x,y) ->  3([x; y]), st=:contour, linestyle=:dash)
plot!(xx, 1.0 .- xx, linewidth=2)
plot!([xy_sol[1]], [xy_sol[2]], marker=true)
```

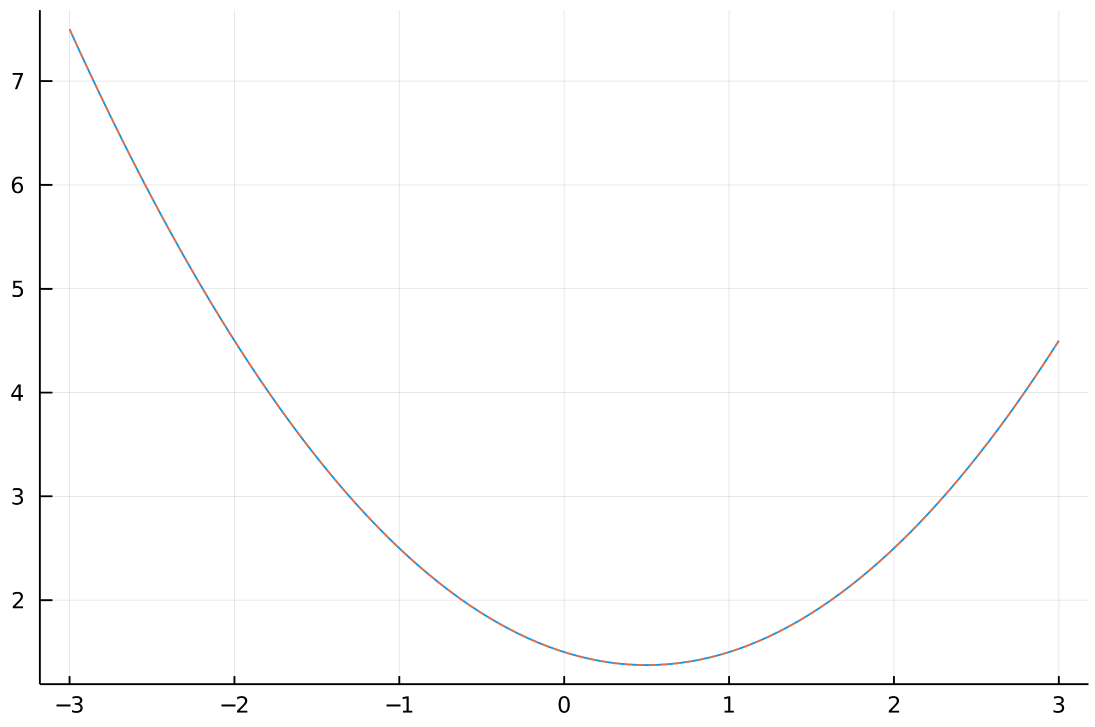
[8]:



[9]:

```
# Sanity check: do we have a constrained min (vs max or saddle)?
plot(xx, [ 2([x; 1.0-x]) for x in xx], legend=false)
plot!(xx, [ 3([x; 1.0-x]) for x in xx], linestyle=:dash)
```

[9]:



```
[10]: # Find the solution via method of Lagrange multipliers
      x  = [H A; A' 0.0] \ [d; b]
```

```
[10]: 3-element Array{Float64,1}:
        0.5
        0.5
       -2.0
```

```
[11]: # Augment the Lagrangian so the (1,1) submatrix is positive definite
        = 2.0
      x _augmented = [H+A*A'/  A; A' 0.0] \ [d+A*b[1]/ ; b]
```

```
[11]: 3-element Array{Float64,1}:
        0.4999999999999999
        0.5
       -1.9999999999999996
```

1.8 Quadratic programs with inequality constraints

We now consider a quadratic objective with linear inequality constraints:

$$\phi(x) = \frac{1}{2}x^T H x - x^T d$$
$$c(x) = A^T x - b \leq 0,$$

where $H \in \mathbb{R}^{n \times n}$ is symmetric and positive definite, $A \in \mathbb{R}^{n \times m}$ with $m < n$, and $b \in \mathbb{R}^m$. The KKT conditions for this problem are

$$\begin{aligned} Hx - d + A\lambda &= 0 \\ A^T x - b &\leq 0 \\ \lambda &\geq 0 \\ \lambda_i (A^T x - b)_i &= 0. \end{aligned}$$

The *active set* is the set of i such that $(A^T x - b)_i = 0$. We assume that the active columns of A are always linearly independent (e.g. $0 \leq x_i$ and $x_i \leq 1$ can co-exist, but it is not OK to have both $x_i \leq 1$ and $x_i \leq 2$).

Examples are always good, as are pictures. We will borrow the following 2D example from Nocedal and Wright (Example 16.3).

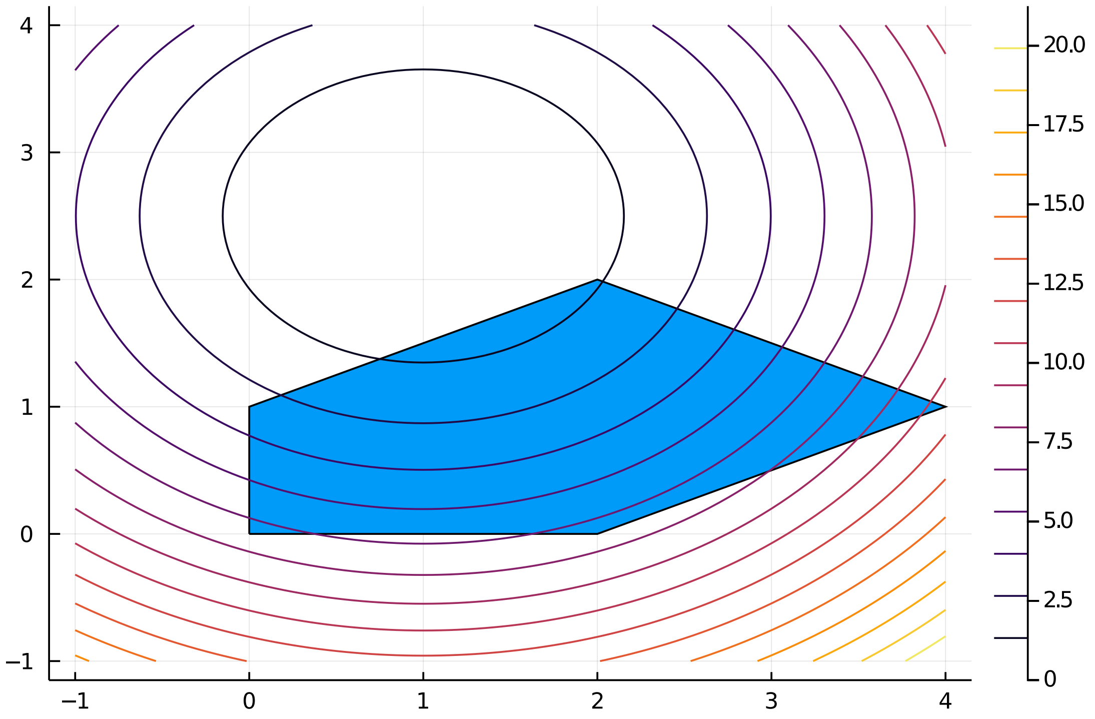
[12]:
```julia
# Objective in Nocedal and Wright:  (x) = (x[1]-1.0)^2 + (x[2]-2.5)^2
#   We will get rid of a constant term to get it to our usual form (and scale␣
 ↪by 1/2)
#
H = [1.0 0.0; 0.0 1.0]
d = [1.0; 2.5]

# Constraints per Nocedal and Wright -- we rewrite so the inequality goes the␣
 ↪other way
#  x1 - 2 x2 + 2   0
# -x1 - 2 x2 + 6   0
# -x1 + 2 x2 + 2   0
#  x1   0
#  x2   0
#
A = [-1.0  2.0 ;
      1.0  2.0 ;
      1.0 -2.0 ;
     -1.0  0.0 ;
      0.0 -1.0 ]'
b = [2.0; 6.0; 2.0; 0.0; 0.0]

# Draw a plot of the quadratic and the constraints
function plot_ex16_3()
    q(x,y) = (x-1.0)^2 + (y-2.5)^2
    corners = [0.0 0.0 ;
               2.0 0.0 ;
               4.0 1.0 ;
               2.0 2.0 ;
               0.0 1.0 ]
    xx = range(-1.0, 4.0, length=101)
    p = plot(corners[:,1], corners[:,2], st=:shape)
    plot!(xx, xx, q, st=:contour, legend=false)
    p
end

plot_ex16_3()
```

[12]:



1.8.1 An active set approach

At the kth step in an active set QP solver, we update an iterate x^k approximating the constrained minimizer *and* we update a corresponding working set $\mathcal{W}^k$ approximating the active set. A step of this solver looks like:

1. Choose a step p^k by minimizing the quadratic form assuming the constraints in $\mathcal{W}^k$ are the active constraints. This gives an equality-constrained subproblem.
2. If p^k is zero, then
    1. Compute the Lagrange multipliers associated with the set $\mathcal{W}^k$.
    2. If all the multipliers are non-negative, terminate.
    3. Otherwise, let λ_j be the most negative multiplier, and set $\mathcal{W}^{k+1} = \mathcal{W}^k \setminus \{j\}$
3. Otherwise $p^k \neq 0$.

1. Advance $x^{k+1} = x^k + \alpha_k p^k$ where the step length α_k is the largest allowed value (up to one) such that x^{k+1} is feasible.
2. If $\alpha_k < 1$, then there is (at least) one *blocking constraint* j such that $(A^T x^{k+1} - b)_j = 0$ and $j \notin \mathcal{W}^k$. Update $\mathcal{W}^{k+1} = \mathcal{W}^k \cup \{j\}$.

If we do not attempt any particular efficiency, this is mostly straightforward to code.

[13]:
```
function qp_as(x0, H, d, A, b; ptol=1e-8, W0=[], nsteps=100, monitor=(x,␣
↪W)->nothing)

    n = length(x0)
    m = length(b)
    W = zeros(Bool, m)

    x = copy(x0)
    p = zeros(n)
     = zeros(m)
    W[W0] .= true
    monitor(x, W)

    # Compute Cholesky factorization for range space solver
    F = cholesky(H)
    L = F.L
    Y = L\A
    c = L\d

    for k = 1:nsteps

        ## Solve the equality constrained subproblem (range space method)      ␣
↪
         [:] .= 0.0
         [W] = ( Y[:,W]'*Y[:,W] )\( Y[:,W]'*c - b[W] )
        p[:] = L'\(c-Y[:,W]* [W])-x

        if norm(p) < ptol

            # Find most negative multiplier (if there is one)
            min  = 0.0
            j = 0
            for k = 1:m
                if  [k] < min
                    min  =  [k]
                    j = k
                end
            end
            if j == 0
                return x      # All multipliers non-negative, done!
            else
                W[j] = false  # Release jth constraint
            end

        else

            # Figure out step (and any blocking constraint)
              = 1.0
            r = b-A'*x
            u = A'*p
            blocking_idx = 0
            for k = 1:m
                if !(k in W) && ( *u[k] > r[k])
                      = r[k]/u[k]
                    blocking_idx = k
                end
            end

            # Take step and update list of active constraints
            x[:] +=  *p
            if blocking_idx > 0
                W[blocking_idx] = true
            end
            monitor(x, W)

        end
    end
    error("Did not converge after $nsteps steps")
end
```

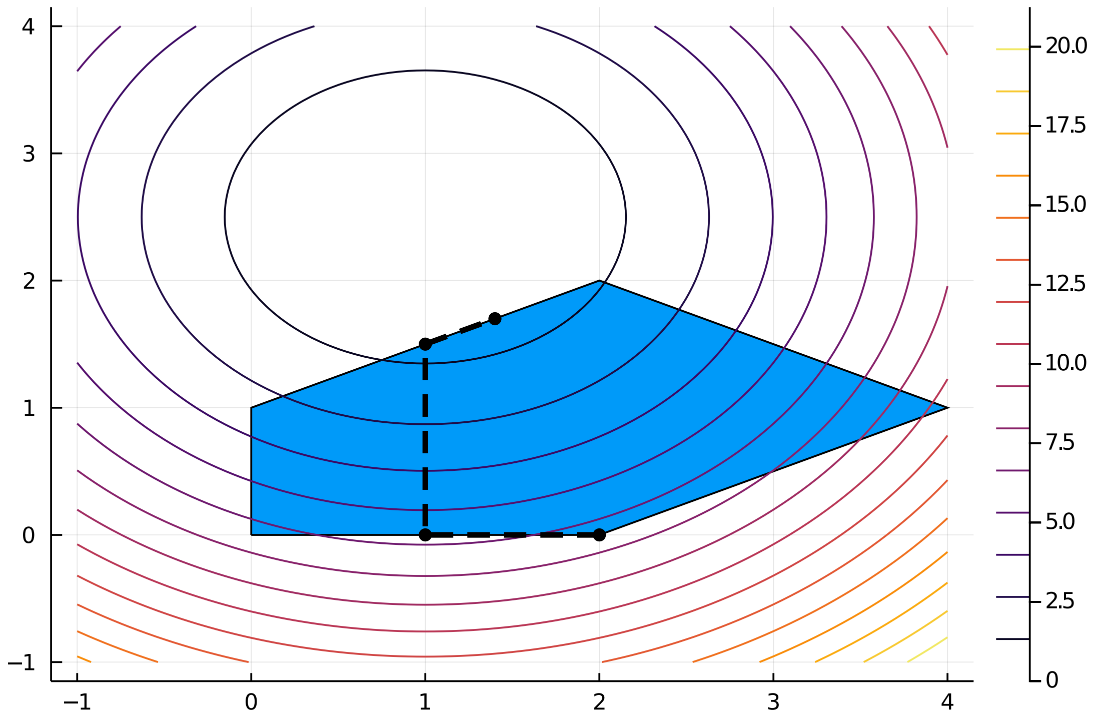
[13]:
```
qp_as (generic function with 1 method)
```

[14]:
```
xhist = []
xsol = qp_as([2.0; 0.0], H, d, A, b, W0=[3 5], monitor=(x, W) -> push!(xhist,␣
 ↪copy(x)))
println("x = $xsol")

p = plot_ex16_3()
plot!([x[1] for x in xhist], [x[2] for x in xhist], marker=true, linewidth=3,␣
 ↪linestyle=:dash, color=:black)
```

```
x = [1.4, 1.7]
```

[14]:



A few remarks about this strategy are in order:

- The strategy is guaranteed not to cycle --- the working set at any given iterate is distinct from the working set at any other iterate. Assuming the steps are computed exactly (via Newton), the iteration converges in a finite number of steps. That said, there are an exponential number of working sets; and, as with the simplex method for linear programming, there are examples where the algorithm may have exponential complexity because of the cost of exploring all the working set. But, as with the simplex method, this is not the common case.
- The strategy only changes the working set by adding or removing one constraint at a time. Hence, if $\mathcal{A}$ is the true active set, the number of steps required is at least $|\mathcal{W}^0| + |\mathcal{A}| - 2|\mathcal{W}^0 \cap \mathcal{A}|$. This is bad news if there are many possible constraints and we lack a good initial guess as to which ones will be active.
- If we compute the steps p^k as described above, the cost per step (after an initial factorization of the Hessian and triangular solves on the constraints) would appear to be $O(n^2 \quad + \quad n|\mathcal{W}^k|^2)$. In practice, though, each linear system differs from the previous system only through the addition or deletion of a constraint. If we are clever with our numerical linear algebra, and re-use the factorization work already invested through updating and downdating, we can reduce the cost per step.

1.8.2 Barriers: hard and soft

Before we proceed, a word is in order about the relationship between Lagrange multipliers and barriers or penalties. To be concrete, let us consider the inequality-constrained problem

$$\text{minimize } \phi(x) \text{ s.t. } c(x) \leq 0,$$

where $c : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with $m < n$, and the inequality should be interpreted elementwise. In a barrier formulation, we approximate the problem by problems of the form

$$\text{minimize } \phi(x) - \mu \sum_{j=1}^{m} \log(-c_j(x)),$$

where the second term shoots to infinity as $c_j(x) \rightarrow 0$; but for any fixed $c_j(x) < 0$ it becomes negligible once μ is small enough. Differentiating this objective gives us the critical point equations

$$\nabla\phi(\hat{x}(\mu)) - \sum_{j=1}^{m} \frac{\mu}{c_j(\hat{x}(\mu))} \nabla c_j(\hat{x}(\mu)) = 0.$$

By way of comparison, if we were to try to exactly optimize this inequality constrained problem, we would want to satisfy the KKT conditions

$$\begin{aligned} \nabla\phi(x) + \nabla c(x)\lambda &= 0 \\ c(x) &\leq 0 \\ \lambda &\geq 0 \\ \lambda_j(x)c_j(x) &= 0. \end{aligned}$$

Comparing the two, we see that the quantities $\hat{\lambda}_j(\mu) \equiv -\mu/c_j(x_*(\mu))$ should approximate the Lagrange multipliers: they play the same role in the equation involving the gradient of ϕ, they are always positive for $\mu > 0$, and $\hat{\lambda}_j(x_*(\mu)) \rightarrow 0$ provided $c_j(x_*(\mu)) \not\rightarrow 0$.

I like to think of barriers and penalties in physical terms as being like slightly flexible walls. In real life, when you push on a wall, however stiff, there is a little bit of give. What we see as an opposing force generated by a rigid wall is really associated with that little bit of elastic give. But a good idealization is that of a perfectly rigid wall, which does not give at all. Instead, it responds to conctact with exactly the amount of force normal to the wall surface that is required to counter any force pushing into the wall. That equal-and-opposite force is exactly what is captured by Lagrange multipliers, where the very stiff elastic response is captured by the barrier or penalty formulation, with the parameter μ representing the compliance of the barrier (inverse stiffness).

The weakness of a penalty or barrier approach is two-fold: if μ is far from zero, we have a thick and spongy barrier (a poor approximation to the infinitely rigid case); whereas if μ is close to zero, we have a nearly-rigid barrier, but the Hessian of the augmented barrier function becomes very ill-conditioned, scaling like μ^{-1}. In contrast, with a Lagrange multiplier formulation, we have a perfect barrier and no problems with ill-conditioning, but at the cost of having to explicitly determine whether the optimum is at one or more of the constraint surfaces, and also what ``contact forces'' are needed to balance the negative gradient of ϕ that pushes into the barrier.

Several modern algorithmic approaches, such as augmented Lagrangian and interior point methods, get the best of both perspectives by combining a penalty or barrier term with a Lagrange multiplier computation.

1.8.3 An interior point strategy

Having touched on the relation between Lagrange multipliers and logarithmic barriers, let us now turn to an interior point method for quadratic programming. We start by rewriting the constraints $A^T x - b \leq 0$ in terms of an extra set of slack variables:

$$y = b - A^T x \geq 0.$$

With this definition, we write the KKT conditions as

$$\begin{aligned} Hx - d + A\lambda &= 0 \\ A^T x - b + y &= 0 \\ \lambda_i y_i &= 0 \\ y_i, \lambda_i &\geq 0. \end{aligned}$$

Interior point methods solve this system by applying Newton-like iterations to the three equations, while at the same time ensuring that the inequalities are enforced strictly at every step (that is, every step is interior to the feasible domain).

Compare this to the critical point conditions for the barrier problem

$$\text{minimize } \frac{1}{2} x^T H x - x^T d - \gamma \sum_{j=1}^{m} \log(y_j)$$

for some small value of the barrier parameter γ, where we note that

$$\nabla_x \left(-\gamma \sum_{j=1}^{n} \log(y_j) \right) = A\hat{\lambda}, \quad \hat{\lambda}_j = \frac{\gamma}{y_j}$$

and we can rewrite this system as

$$\begin{aligned} Hx - d + A\lambda &= 0 \\ A^T x - b + y &= 0 \\ y_i \lambda_i - \gamma &= 0. \end{aligned}$$

Typical interior point methods take guarded Newton steps (or Newton-like steps) on this system of equations, which can be regarded as a relaxation of the KKT conditions or as a critical point of a barrier formulation. The path traced out as μ varies is known as the ``central path.''

[15]:
```
function barrier_qp(x0, H, d, A, b,  ; nsteps=20, ptol=1e-8)

    n = length(d)
    m = length(b)
      = 0.5

    x = copy(x0)
    y = b-A'*x
      =  ./y

    F(x, y,  ) = [H*x - d + A* ;
                  A'*x - b + y;
                  y.*  .-  ]
    J(x, y,  ) = [H           zeros(n,m)  A;
                  A'          I           zeros(m,m);
                  zeros(m,n) diagm( )    diagm(y)]

      = 1.0
    p = -(J(x, y,  ) \ F(x, y,  ))
    for k = 1:nsteps
        xnew = x +  *p[1:n]
        if all(A'*xnew-b .<= 0.0)
            x = xnew
            y +=  *p[n+1:n+m]
              +=  *p[n+m+1:end]
            if   == 1.0 && norm(p) < ptol
                return x,
            end
              = 1.0
            p = -(J(x, y,  ) \ F(x, y,  ))
        else
              /= 2.0
        end
    end

    error("Did not converge in $nsteps steps")
end
```

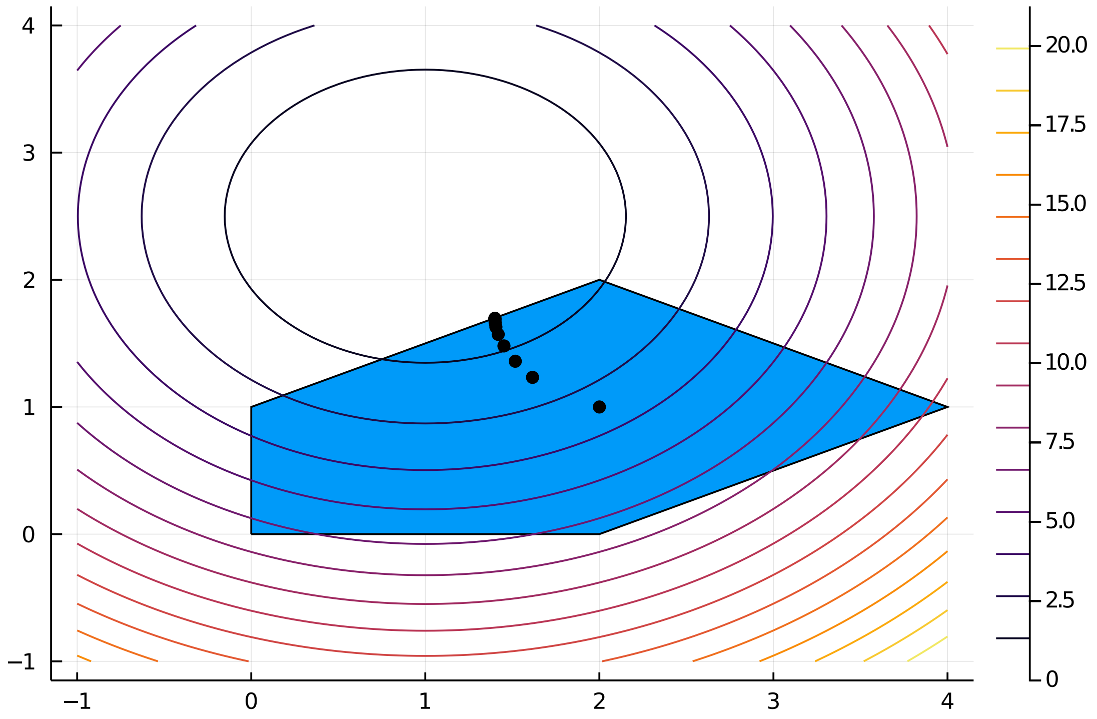
[15]:
```
barrier_qp (generic function with 1 method)
```

[16]:
```
x = [2.0; 1.0]

p = plot_ex16_3()
plot!([x[1]], [x[2]], marker=true, color=:black)
for s = 1:10
    x,   = barrier_qp(x, H, d, A, b, 2.0^(1-s))
    plot!([x[1]], [x[2]], marker=true, color=:black)
```

```
end
p
```

[16]:



The parameter γ is adjusted dynamically during the solve, and is usually written as $\gamma = \sigma\mu$ where $\sigma \in [0,1]$ is the centering parameters and $\mu = y^T\lambda/m$ is the *complimentarity measure*, which should go to zero as we approach a problem solution. Getting all the details right is somewhat complicated, though, and we recommend using a package written by someone with some expertise.

Interior point methods avoid the problem of having to do a combinatorial search to figure out the correct active set. At the same time, active set methods may be more efficient for problems where we have a good initial guess at the active set. Neither approach is universally preferable. Indeed, it is possible to take a hybrid approach where an interior point method (or something similar) is used to estimate which constraints are actually active, and then an active set method serves to ``clean up'' the solution.