Scientific Computing

Jonathan Goodman, Fall, 2025

Constrained Optimization

Formulation

Optimization means finding x_* that minimizes a function $u(x)$. An *equality constraint* is an equation

$$g(x) = 0 \ .$$

If there are $m \geq 1$ equality constraints, then $g(x)$ is a vector with m components

$$g(x) = \begin{pmatrix} g_1(x) \\ \vdots \\ g_m(x) \end{pmatrix} \ .$$

There also may be *inequality constraints*, which are formulated as

$$h(x) \geq 0 \ . \tag{1}$$

If there are $r \geq 1$ inequality constraints, then h has r components and the inequality is supposed to hold for each component:

$$h(x) = \begin{pmatrix} h_1(x) \\ \vdots \\ h_r(x) \end{pmatrix} \ .$$

The vector inequality constraint (1) denotes the r scalar inequalities

$$h_k(x) \geq 0 \ \text{ for } k = 1, \cdots, r \ .$$

The *feasible set* is the set of $x \in \mathbb{R}^d$ that satisfy all the constraints

$$x \in \mathcal{F} \iff g(x) = 0 \ \text{ and } \ h(x) \geq 0 \ .$$

The official goal of constrained optimization is to find

$$x_* = \arg \min_{x \in \mathcal{F}} u(x) \ .$$

Unofficially, the goal may just be to find a point with better "performance". If x_0 is an *initial guess*, this would mean finding feasible *iterates* x_n with $x_n \in \mathcal{F}$ and $u(x_n) \ll u(x_0)$.

As with ordinary differential equations, constrained optimization problems may not be presented in this form precisely. For example, suppose the problem is to find a point on an ellipse closest to a point (a, b) (here, H is a positive definite 2×2 matrix)

$$\min \,(x-a)^2 + (y-b)^2 \ \text{ with } \ \begin{pmatrix} x & y \end{pmatrix} H \begin{pmatrix} x \\ y \end{pmatrix} = r^2 \ .$$

This is put in the general constrained optimization form by taking

$$u(x,y) = (x-a)^2 + (y-b)^2 \ , \ \text{ and } \ g(x,y) = \begin{pmatrix} x & y \end{pmatrix} H \begin{pmatrix} x \\ y \end{pmatrix} - r^2 \ .$$

A *portfolio optimization* problem might be to minimize the variance of an investment subject to a budget constraint and the constraint that all the "positions" must be positive. This may be formulated using *portfolio weights* w_k, for $k = 1, \cdots, d$ representing the "weight" (percentage of the funds) invested in asset k. The budget constraint is $w_1 + \cdots + w_d = 1$. The variance of the portfolio is (here Σ is the covariance matrix of the individual assets)

$$w^T \Sigma \, w \; .$$

The positivity constraints are $w_k \geq 0$ for all k. For this example, suppose there is also the policy that no more than p percentage of the total goes to any particular asset. This optimization problem is put in the generic form above using (here, $\mathbf{1}$ is the column vector of all ones, $\mathbf{1} = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^T$, so $w^T \mathbf{1} = w_1 + \cdots + w_d$)

$$\begin{aligned} u(w) &= w^T \Sigma \, w \\ g(w) &= g_1(w) = w^T \mathbf{1} \\ h_k(w) &= w_k \;\text{ for } k = 1, \cdots, d \\ h_k(w) &= p - w_{k-d} \;\text{ for } k = d+1, \cdots, 2d \; . \end{aligned}$$

There is one equality constraint and there are $2d$ inequality constraints. The first d represent positivity of portfolio weights. The second d represent the balancing condition that no asset gets more than p of the total.

Constraint surfaces

The feasible set defined by equality constraints only will be called the *constraint surface* and denoted $\mathcal{S}$

$$\mathcal{S} = \{ \, x \text{ with } g(x) = 0 \, \} \; .$$

It is helpful to describe the constraint surface using geometric language even though in the end all the algorithms are algebraic and analytic. Normally (at non-degenerate points) the constraint surface is *smooth*. You can think of a constraint surface as being like a surface (sphere, etc.) or a curve in 3D.

We use J to denote the jacobian matrix

$$J = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_d} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_d} \\ \vdots & & & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_d} \end{pmatrix} \; . \tag{2}$$

This is an $m \times d$ matrix whose (j,k) entry is

$$J_{jk} = \frac{\partial g_j}{\partial x_j} \; .$$

Row j of the jacobian matrix describes how g_j changes with a small change in x

$$\Delta g_j(x) = \frac{\partial g_j}{\partial x_1} \, \Delta x_1 + \cdots + \frac{\partial g_j}{\partial x_1} \, \Delta x_d + O(\|\Delta x\|^2) \; .$$

Column k describes the changes in the g_j corresponding to a change Δx_k. In matrix form,

$$\Delta g = \begin{pmatrix} \Delta g_1 \\ \Delta g_2 \\ \vdots \\ \Delta g_m \end{pmatrix} = \begin{pmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_d} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_d} \\ \vdots & & & \vdots \\ \frac{\partial g_m}{\partial x_1} & \frac{\partial g_m}{\partial x_2} & \cdots & \frac{\partial g_m}{\partial x_d} \end{pmatrix} \begin{pmatrix} \Delta x_1 \\ \Delta x_2 \\ \\ \vdots \\ \\ \Delta x_d \end{pmatrix}$$

The jacobian is a "short and fat" matrix because the number of constraints is less than the number of variables.

Let $\overline{x}$ be a point on the constraint surface ($\overline{x} \in \mathbb{S}$, $g(\overline{x}) = 0$). The constraints are *non-degenerate* at $\overline{x}$ if

$$\text{rank}(\, J(\overline{x})\,) = m \ .$$

Non-degenerate means that the rows of J, which are the (transposes of) the gradients of the constraint functions g_j, are linearly independent at $\overline{x}$. We say that J has "full rank" because the rank of a short and fat matrix cannot be more than m. If $\overline{x}$ is a non-degenerate point on $\mathcal{S}$, then there is a *tangent space* to $\mathcal{S}$ defined at $\overline{x}$. We denote it by $T(\overline{x})$, but people use other notations such as $T_{\overline{x}}$ or $T_{*\overline{x}}$. This is the set of vectors $v \in \mathbb{R}^d$ that are in the null space of $J(\overline{x})$.

$$T(\overline{x}) = \{\, v \;\text{ with } Jv = 0\,\} \ .$$

The dimension of $T(\overline{x})$ is $d - m$. This is because the rank of J plus the dimension of its null space is d. The $v \in T(\overline{x})$ are directions you can move from $\overline{x}$ that are tangent to $\mathcal{S}$ in the usual sense.

For example, the unit sphere in $3D$ is described by $m = 1$ constraints $x^2 + y^2 + z^2 = 1$. The jacobian is the 1×3 matrix

$$J = \begin{pmatrix} 2x & 2y & 2z \end{pmatrix} \ .$$

The dimension of $T(\,\overline{(x,,y,z)}\,)$ should be $d - m = 3 - 1 = 2$. You can see that the vectors r_1 and r_2 are a basis for $T(\,\overline{(x,,y,z)}\,)$ at most points

$$r_1 = \begin{pmatrix} -\overline{y} \\ \overline{x} \\ 0 \end{pmatrix} \quad , \quad r_2 = \begin{pmatrix} 0 \\ \overline{z} \\ -\overline{y} \end{pmatrix} \ .$$

The vectors r_1 and r_2 are linearly independent and have $Jr_1 = 0$ and $Jr_2 = 0$.

As another example, consider two unit spheres on $3D$, one centered at the origin and one at $(1, 0, 0)$. The intersection of these spheres is a circle parallel to the (y, z) plane equi-distant from the centers of the spheres. The constraint functions are

$$g(x,y,z) = \begin{pmatrix} g_1(x,y,z) \\ g_2(x,y,z) \end{pmatrix} = \begin{pmatrix} x^2 + y^2 + z^2 - 1 \\ (x-1)^2 + y^2 + z^2 - 1 \end{pmatrix}$$

The jacobian matrix is

$$J = \begin{pmatrix} 2x & 2y & 2z \\ 2(x-1) & 2y & 2z \end{pmatrix} \ . \tag{3}$$

Algebraically, you can describe the constraint "surface" $\mathcal{S}$ by subtracting the two constraint equations

$$\begin{aligned} x^2 \quad + y^2 + z^2 &= 1 \\ (x-1)^2 + y^2 + z^2 &= 1 \\ x^2 \ - \ (x-1)^2 \quad &= 0 \\ 2x \quad &= 1 \\ x \quad &= \frac{1}{2} \\ y^2 + z^2 \quad &= 1 - \frac{1}{4} = \frac{3}{4} \end{aligned}$$

A tangent vector to this circle at the point $\overline{x} = \frac{1}{2}$ and $\overline{y}^2 + \overline{z}^2 = \frac{3}{4}$ is

$$r = \begin{pmatrix} 0 \\ \overline{z} \\ -\overline{y} \end{pmatrix} . \tag{4}$$

You can check that $J(\,(\overline{x}, \overline{y}, \overline{z})\,) = 0$. The dimension of the tangent space in this example is $d - m = 3 - 2 = 1$. The vector r is tangent to the circle in $3D$.

Vectors perpendicular to the tangent "plane" ($T(\overline{x})$ does not have to be two dimensional) are *normal* to $\mathcal{S}$. The gradients ∇g_j are a basis for the vector space of normal vectors if $\overline{x}$ is a non-degenerate point of $\mathcal{S}$. This is "trivial" but possibly confusing. The (transposes of the) gradients ∇g_j are the rows of J, and each row of J is perpendicular to every $v \in T$, which was the definition of T. Thus T is the set of vectors normal to the vector space spanned by the gradients. The normal space to the normal space is the space itself (draw a picture). Thus, the normal space is

$$\mathcal{N}(\overline{x}) = \text{span}\{\, \nabla g_1(\overline{x}), \cdots, \nabla g_m(\overline{x})\,\} \ .$$

Since $\mathcal{N}$ is spanned by m linearly independent vectors, it has dimension m. In the circle example above, the normal space to the circle at a point $(\overline{x}, \overline{y}, \overline{z})$ is spanned by two vectors such as

$$n_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad , \quad n_2 = \begin{pmatrix} 0 \\ \overline{y} \\ \overline{z} \end{pmatrix} \quad .$$

You can check that both of these are normal to r in (4) and that they are linearly independent. You can also check that they span the same space as the (transposes of the) rows of J in (3):

$$\nabla g_1 = \begin{pmatrix} 2\overline{x} \\ 2\overline{y} \\ 2\overline{z} \end{pmatrix} = 2\overline{x}\, n_1 + 2n_2 \ \ , \quad \nabla g_2 = \begin{pmatrix} 2(\overline{x}-1) \\ 2\overline{y} \\ 2\overline{z} \end{pmatrix} = 2(\overline{x}-1)\, n_1 + 2n_2 \ \ .$$

Linear projection

The gradient descent algorithms involve a sequence of iterates $x_n \in \mathcal{S}$. These are found using the gradient ∇u but only to move along $\mathcal{S}$. We view "moving along $\mathcal{S}$ in two steps. First, take the "part of" the gradient in the tangent space at the current iterate, $T(x_n)$. Moving in this direction keeps you close to $\mathcal{S}$ but not exactly on it. Next, you take a point near $\mathcal{S}$ and find a corresponding point in $\mathcal{S}$. These steps are a linear projection (defined below) followed by a nonlinear projection.

A linear *projection* is a linear map that takes any vector y to its "projection" in a subspace T. We call it $P(x_n)$. In principle, we could compute the $d \times d$ matrix that represents P. Instead, we create an algorithm that finds Py for any $y \in \mathbb{R}^d$. To be a projection, P must not move any vector

$v \in T$, which is $Pv = v$ for any $v \in T$. A *perpendicular projection* (also called *normal* projection) is a projection that acts on y by removing the components of y in directions normal to T. We present two approaches to this, one involving finding the components of y in a basis[1] of T. The other involves subtracting away the part of y perpendicular to T. Both of these approaches to projection may be familiar from solving linear least squares problems. Orthogonal projection is essentially equivalent to linear least squares.

In the second projection approach, you have a subspace $\mathcal{N}$ and a vector $y \in \mathbb{R}^d$. The projection "along $\mathcal{N}$" modifies y using vectors in $\mathcal{N}$ to get v perpendicular to $\mathcal{N}$. Suppose $h_1, \cdots, h_m$ is a basis of $\mathcal{N}$. The modification along $\mathcal{N}$ is

$$v = y - \sum_{j=1}^{m} w_j h_j \quad v = y - Hw \; .$$

The matrix form uses the matrix H whose columns are the basis vectors and the vector whose components are the weights w_j:

$$R = \begin{pmatrix} | & & | \\ h_1 & \cdots & h_m \\ | & & | \end{pmatrix} \;, \quad w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} \; .$$

The condition that v is perpendicular to h_j may be written $h_j^T v = 0$ The condition that v is perpendicular to all the vectors r_j may be written

$$H^T v = 0 \; .$$

We can plug in the definition of v to get an equation for w:

$$H^T \left(y - Hw \right) = 0 \; .$$

This leads to

$$w = \left(H^T H \right)^{-1} y \; . \tag{5}$$

The symmetric matrix $M = H^T H$ is the Gram matrix of inner products $h_j^T h_k$. It is positive definite (hence invertible) if R has full rank m. In the "real" computer world, M will be hard to invert if it is poorly conditioned, which happens if the h_j are almost linearly dependent.

The gradients of the constraints form a natural basis of $\mathcal{N}$. The definition of T was that v is tangent to $\mathcal{S}$ if $Jv = 0$. This equation also says that every row of J is perpendicular to every tangent vector $v \in T$. This is geometric/algebraic theorem of linear algebra. If $T \subset \mathbb{R}^d$ and $\mathcal{N}$ is the subspace perpendicular to T, then T is the subspace perpendicular to $\mathcal{N}$. The perpendicular to the perpendicular is the original subspace. This doesn't depend on T and $\mathcal{N}$ being tangent and normal spaces to a surface. These gradients may be thought of as columns of the (tall and thin) gradient matrix

$$G = \begin{pmatrix} | & & | \\ \nabla g_1 & \cdots & \nabla g_m \\ | & & | \end{pmatrix} \; .$$

You can see that the rows of the jacobian matrix J in (2) are the transposes of the gradients, so $J = G^T$. Thus, the projection equation (5) may be written

$$w = \left(GG^T \right)^{-1} \nabla u \;, \quad v = \nabla u - G \left(GG^T \right)^{-1} \nabla u \; .$$

[1]This basis of T has $d - m$ elements. It is not a basis of $\mathbb{R}^d$. If q is a *unit vector*, which means $\|q\|_2 = 1$, then the *component* of y in a direction q is $q^T y$. If this q were one vector in an orthonormal basis of $\mathbb{R}^d$, then $q^T y$ would be the coefficient of y multiplying q. If there are $d - m$ ortho-normal vectors $q_j \in T$, then the part of y in T is the sum of the components in the q_j directions.

The other projection approach uses an ortho-normal basis for T. This may be found using the QR decomposition of $J^T = G$. This factorization is

$$G = QR = \begin{pmatrix} | & | & | \\ Q_1 & | & Q_2 \\ | & | & | \end{pmatrix} \begin{pmatrix} R_1 \\ -- \\ 0 \end{pmatrix} .$$

The matrix blocks in this equation are Q_1 is the first m columns of Q and Q_2 is the remaining $d-m$ columns, R_1 is the $m \times m$ upper triangular matrix and the lower block of R is the $(d-m)\times m$ zero matrix. We assumed that G has full rank m, which implies that R_1 has full rank. Since R_1 is upper triangular, this is equivalent to the diagonal entries of R_1 being non-zero. Transpose this and you get

$$G^T = J = \begin{pmatrix} R_1^T & | & 0 \end{pmatrix} \begin{pmatrix} -Q_1^T- \\ --- \\ -Q_2^T- \end{pmatrix} = R_1^T Q_1^T .$$

If you multiply J with a column q_j from Q, then you get non-zero or zero depending on whether q_j is a column of Q_1 or Q_2. This is because

$$Q^T q_j = e_j \quad , \quad e_j \text{ being the } j^{\text{th}} \text{ standard basis vector.}$$

This, in turn, is because the columns of Q are ortho-normal. $q_k^T q_j = 0$ if $k \neq j$ and $q_j^T q_j = 1$. These are the entries in the basis vector e_j. Now,

$$\begin{pmatrix} R_1^T & | & 0 \end{pmatrix} e_j \neq 0$$

if $j \leq m$ because the diagonals of R_1 are not zero, and

$$\begin{pmatrix} R_1^T & | & 0 \end{pmatrix} e_j = 0$$

if $j > m$, clearly. To summarize, the column space of Q_2 (the space spanned by the columns of Q_2) is the normal space to $\mathcal{N}$, which is the tangent space, T. The vectors q_j, for $j = m+1, \cdots, d$, are a basis of T. Finally, if y is any vector in $\mathbb{R}^d$, then y may be written in terms of the Q basis as

$$y = \sum_{j=1}^{n} a_j \, q_j \quad , \quad a_j = q_j^T y \; . \tag{6}$$

The "part of" y in the tangent space T is the part of this sum involving the basis vectors for T:

$$w = Py = \sum_{j>m} a_j q_j = \sum_{j=m+1}^{n} \left(q_j^T y\right) q_j \; . \tag{7}$$

This is a different way to compute the orthogonal projection of y onto T.

The discussion of convergence below uses the fact that orthogonal projections do not change inner products taken with respect to vectors $v \in T$. That means that if $v \in T$ and $y \in \mathbb{R}^d$ (any vector), then

$$v^T P y = v^T y \; . \tag{8}$$

You can see this in $2D$ or $3D$ by drawing pictures. You can see it in formulas by using a basis q_j where the q_j for $j > m$ are a basis for T and the 4_j for $j \leq m$ are orthogonal to T. In particular, if

$j \leq m$, then $v^T q_j = 0$, because $v \in T$ and q_j is perpendicular to T. Therefore,

$$\begin{aligned} v^T y &= \sum_{j=1}^{d} a_j v^T q_j \\ &= \sum_{j>m} a_j v^T q_j \\ &= v^T \sum_{j>m} a_j q_j \\ &= v^T P y \ . \end{aligned}$$

[Another way to see (8) is to see that the matrix representing P is symmetric. In fact

$$P = \sum_{j>m} q_j q_j^T \ . \tag{9}$$

This is a way to express the projection formula (7), since matrix/vector multiplication is associative, and so

$$\begin{aligned} Py &= \sum_{j>m} q_j q_j^T y \\ &= \sum_{j>m} q_j \left(q_j^T y \right) \end{aligned}$$

This shows that Py is the w of (7). The formula (9) implies that $P^T = P$ because the terms in the sum satisfy

$$\left(q_j q_j^T \right) = \left(q_j^T \right)^T q_j^T = q_j q_j^T \ .$$

The equation (9) also implies that $P^2 = P$, as you can check by writing the formula once with index j and again with index k then treating PP as a double sum, where only the terms $j = k$ are different from zero. We knew $P^2 = P$ already, because $Py \in T$ and $Pv = v$ for any $v \in T$.]

Nonlinear projection

Now suppose $\overline{x}$ satisfies the constraints and v is a tangent vector at $\overline{x}$: $\overline{x} \in \mathcal{S}$, and $v \in T(\overline{x})$. If v is small, then $z = \overline{x} + v$ is a point close to but not on $\mathcal{S}$. Analytically, $g(\overline{x})$ is small but not zero. Nonlinear projection is the problem of projecting z onto $\mathcal{S}$, which means finding $\overline{y} \in \mathcal{S}$ with $g(\overline{y}) = 0$. A convenient way (but not the only way) to find such as $\overline{y}$ is to project perpendicular to $T(\overline{x})$. This is convenient because we have a basis of vectors perpendicular to $T(\overline{x})$. We can take the vectors $\nabla g_j(\overline{x})$ as this basis. Remember that $\mathcal{S}$ is curved, so the gradients ∇g_j at z are different from those at $\overline{x}$ or at the point $\overline{y}$ that we hope to find. Choosing to project perpendicularly to $T(\overline{x})$ means keeping the basis vectors $\nabla g_j(\overline{x})$ fixed during the process of finding y.

We find $\overline{y}$ by formulating a nonlinear system of equations that y satisfies and then solving those equations using Newton's method. To formulate the equations, we express "moving from z perpendicular to $T(\overline{x})$ analytically as

$$y = z + \sum_{j=1}^{m} w_j \nabla g_j(\overline{x}) = G(\overline{x})\, w \ .$$

Here, G is the matrix whose columns are ∇g_j and $w \in \mathbb{R}^m$ is the vector of coefficients. The right $w = \overline{w}$ will give $\overline{y} \in \mathcal{S}$. The equations $g(\overline{y}) = 0$ may be expressed in terms of w as

$$h(w) = g(z + G(\overline{x})\, w) = 0 \ .$$

The jacobian matrix of h is the $m \times m$ matrix

$$H(w) = h'(w) \; .$$

This is found using the chain rule

$$H(w) = \frac{\partial h}{\partial w} = (\partial_y g)\,(\partial_w y) = J(z + G(\overline{x})w)\, G(\overline{x}) \; . \tag{10}$$

To explain the arguments of J and G on the right, the derivatives of g need to be evaluated at the point $z + G(\overline{x})w$, but the jacobian matrix of y with respect to w is the constant matrix $G(\overline{x})$. Note that when $v = 0$ and $w = 0$, then $z = \overline{x} = z + G(\overline{x})w$, so $H_0 = JJ^T$. This is an $m|times m$ symmetric matrix. Assuming J has full rank, this is positive definite. Therefore, if v and w are small, $H(w) \approx H_0$. Thus, H should be non-singular, but the matrix (10) is probably not symmetric.

Newton's method, with the default step size/learning rate $s \equiv 1$, is

$$w_{n+1} = w_n - H(w_n)^{-1} h(w_n) \; .$$

With initial guess $w_0 = 0$, which is $z = x + v$, if v is small enough, the iteration will converge to $\overline{w}$ with $\overline{y} = z + G\overline{w} \in \mathcal{S}$.

Projection gradient descent

The overall gradient descent algorithm for equality constrained optimization has an *outer* iteration and an *inner* iteration. The outer iteration uses finds the new iterate x_{n+1} from the current iterate x_n using $\nabla u(x_n)$. To do this it moves by v_n in the tangent space $T(x_n)$ and then projects (tries to project) $x_n + v_n$ back to the constraint surface $\mathcal{S}$. If the inner Newton iteration does find $y_n = x_n + v_n + G(x_n)w_n \in \mathcal{S}$ then it can do a step size/learning rate check similar to that of unconstrained gradient descent. The predicted decrease is

$$\Delta u_P = -\nabla u(x_n)(y_n - x_n) \; .$$

The actual decrease is[2]

$$\Delta u = -(u(y_n) - u(x_n)) \; .$$

We choose the step size s so that

$$\Delta u \geq \alpha \Delta u_P \qquad \text{(sufficient decrease)} \tag{11}$$

$$\Delta u \leq \beta \Delta u_P \qquad \text{(try a bigger } s\text{).} \tag{12}$$

This is a version of *line search.* If the sufficient decrease condition is violated, replace s with $\frac{1}{2}s$ and try again. If the "try a bigger s" condition is violated, that's because the algorithm thinks the step size is too small, and that a larger s would give more improvement. In that case, replace s with $2s$ and try again. Be careful here. A naive code based on these ideas could lead to an infinite loop in which first $s \to \frac{1}{2}s$ and then $s \to 2s$ because first one then the other condition is violated. The code should detect this and do something else to satisfy (11) and (12) in some other way. You can start this line search process with the s that worked at the previous iterate.

For each s, you first find the tangential step $v = -s\nabla u(x_n)$ and then do a projection Newton iteration to find y_n. The initial guess should be $w = 0$. If the Newton iteration fails (which it will if s is too large), then replace s with $\frac{1}{2}s$ and try again. It is a consequence of the *inverse function theorem* that there is a projection if v is small enough. One proof of this involves showing that Newton's method converges if v is small enough. If you draw a diagram it should be clear why this is true, and why $w = 0$ is a reasonable initial guess.

[2]This clumsy choice of sign is to emphasize that u is supposed to decrease each iteration. We hope $\Delta u = u(y_n) - u(x_n)$ is negative. The *decrease* will be positive in that case.

Convergence and stopping criteria, Lagrange multipliers

There must be some halting criterion. A simple one would be to do a specified number of iterations. This can be restated as using up a given computational budget. More commonly, particularly in casual computing, is to keep iterating until "the algorithm" thinks it is close to the optimal x. Two things that can be checked are the size of the step and the size of the projected gradient. There could be two numbers ϵ_s and ϵ_g and the algorithm would halt if these conditions are satisfied:

$$\|x_{n+1} - x_n\|_2 \leq \epsilon_s \tag{13}$$

$$\|P(x_n)\, \nabla u(x_n)\|_2 \leq \epsilon_g \; . \tag{14}$$

The tolerance numbers ϵ_s and ϵ_g would be supplied to the optimization function by the user.

The user could be expected to have a reasonable idea how close they would like to be to the true optimizer, and ϵ_s is an indicator of this distance. It is not a perfect indicator because, as we saw in the analysis of optimizing a quadratic function, the learning rate might have to be very small relative to the distance to the optimizer if the hessian is ill conditioned. This means that the step taken, as measured in (13) might be much smaller than the distance to the optimizer. Thus, (13) is a useful halting criterion but probably should not be the only one.

The projected gradient, $P(x_n)\,\nabla u(x_n)$, is the direction of steepest ascent in the tangent space of the current iterate $T(x_n)$. This is because if v is any vector, then

$$\frac{d}{ds} u(x+sv)\Big|_{s=0} = v^T \nabla u(x) \; .$$

If $v \in T$, the property (8) allows us to write this as

$$\frac{d}{ds} u(x+sv)\Big|_{s=0} = v^T P\, \nabla u(x) \; .$$

In particular, this shows that if the projected gradient is zero, then x is a stationary point of u in the constraint surface $\mathcal{S}$. In summary, "gradient descent" is a natural way to describe taking the the tangent space step to be $v = -sP\,\nabla u$. Therefore, if the projected gradient is small, which is the condition (14), then you can hope you're near to a stationary point. Stationary points that are not local minima are unstable, so you have to be very unlucky to converge to a stationary point that is not a local min.

The method of *Lagrange multipliers* can be thought of as a way to look for points $x \in \mathcal{S}$ where the projected gradient vanishes. If $P\nabla u(x) = 0$ then $\nabla u \in \mathcal{N}$ (this is the definition of the normal space). Any vector in $\mathcal{N}$ is a linear combination of the basis vectors $\nabla g_j(x)$. Thus, if $\nabla u \in \mathcal{N}$, then we may express $\nabla u(x)$ as

$$\nabla u = \sum_{k=1}^{m} \lambda_k \nabla g_k(x) \; . \tag{15}$$

It is traditional to denote the coefficients of ∇g_k by λ_k and to call them *Lagrange multipliers.* The Lagrange multiplier approach to equality constrained optimization is to formulate $d+m$ equations for the $d+m$ unknowns x_j and λ_k, for $j = 1, \cdots, d$ and $k = 1, \cdots, m$. The Lagrange multiplier formula (15) is d equations (one for each component of ∇u. The other m equations are the constraint equations $g_k(x) = 0$ that define the constraint surface $\mathcal{S}$. There are fancier algorithms for constrained optimization that use Lagrange multipliers in a more fundamental way, but you don't have to think about Lagrange multipliers to understand gradient descent on a constraint surface.