CME 304 Project

Christopher Maes

Abstract

Numerical optimization is used to compute solutions to the Brachistochrone problem. The infinite-dimensional calculus of variations problem is approximated by a finite-dimensional problem using a Chebyshev spectral collocation method. The computed brachistochrone is thus a Chebyshev polynomial of degree N. The resulting numerical optimization problem has linear inequality constraints. This problem is solved by a Quasi-Newton method that computes search directions via a linear inequality constrained convex quadratic subproblem. The subproblem is solved with a infeasible primal-dual interior point method.

The Brachistochrone Problem

In 1696 Johan Bernoulli posed the *brachistochrone* ("shortest time") problem:

> Suppose two points A and B lie in a plane, with A higher but not directly above B. A wire bent in the shape of a curve γ joins A and B (Figure 1). A bead slides along the wire from A to B. There is no force on the bead expect the force of gravity; in particular, there is no friction. Find the shape of γ that minimizes the time required for the bead to fall from A to B. [5, p.1]

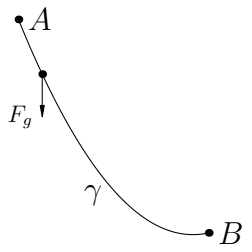

Figure 1: The brachistochrone problem setup.

Following [5] we derive a formula relating a choice of the curve γ to the time required for a bead to fall from A to B (transit time). For reasons that will become clear later, we place the point A at $(-1, 0)$ and the point B at $(1, \beta)$, for some $\beta < 0$ (Figure 2). Then any curve γ will be the graph of a function $f(x)$ on the interval $[-1, 1]$, satisfying $f(x) \leq 0$ for all $x \in [-1, 1]$ and with the end constraints: $f(-1) = 0$, and $f(1) = \beta$.

Let $x = x(t)$ and $y = y(t)$ denote the particles position at time t, $s = s(t)$ denote the distance along γ that the bead has traveled at time t and $v = v(t)$ the velocity of the bead along γ at time t. Assuming the $\gamma = f(x)$ is differentiable,

$$s(t) = \int_{-1}^{x(t)} \sqrt{1 + f'(u)^2}\, du$$

and $v(t) = s'(t)$, so

$$v(t) = x'(t)\sqrt{1 + f'(x(t))} \tag{1}$$

Since there is no friction the total energy at any time t must be the same as the total energy at time zero, which we will take to be zero. Since kinetic energy is $1/2mv^2$ and potential energy is mgh, where h is the height above the x-axis, we have

$$\frac{1}{2}mv^2(t) + mgf(x(t)) \equiv 0$$

so that

$$v(t) = \sqrt{-2gf(x(t))}, \tag{2}$$

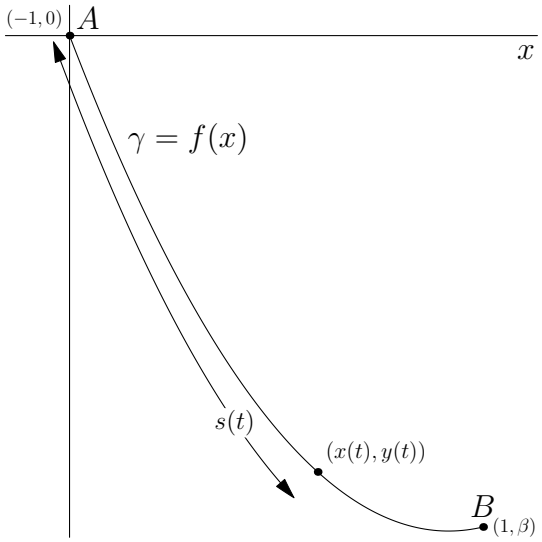

Figure 2: The bead falls from $A = (-1, 0)$ to $B = (1, \beta)$. The position of the bead is given by $(x(t), y(t))$, the distance the bead has traveled along the wire is $s(t)$. The shape of the wire γ is the graph of the function $f(x)$. Taken from [5]

recall that $f(x) \leq 0$ for $x \in [-1, 1]$. Combining (1) and (2) yields the differential equation

$$\sqrt{-2gf(x)} = \sqrt{1 + f'(x)^2}\ \frac{dx}{dt}$$

By separating variables and integrating we arrive at a relation for the transit time T:

$$T = \int_0^T dt = \frac{1}{\sqrt{2g}} \int_{-1}^{1} \sqrt{\frac{1 + f'(x)^2}{-f(x)}}\ dx. \tag{3}$$

With the constraint $f(-1) = 0$ the integrand defined in Equation (3) is singular at the left-endpoint of integration, so this is actually an improper integral. Note that this singularity is not caused by the fact that we choose $A = (-1, 0)$ and therefore we must have $f(-1) = 0$, but rather is inherit in the problem setup and is a result of the bead starting at rest ($v(0) = 0$). There is not much need to worry, however, for if $f(x)$ is a differentiable function the integral will always converge.

Thus, we see that the brachistochrone problem is equivalent to the following minimization problem

$$\begin{array}{ll} \underset{f \in C^1[-1,1]}{\text{minimize}} & T = \dfrac{1}{\sqrt{2g}} \displaystyle\int_{-1}^{1} \sqrt{\frac{1 + f'(x)^2}{-f(x)}}\ dx \\ \text{subject to} & f(x) \leq 0, \quad \forall x \in [-1, 1] \\ & f(-1) = 0, \quad f(1) = \beta \end{array} \tag{4}$$

Problem (4) is a problem of the variational calculus. The optimal, analytical, solution is given by the cycloid, the path in the plane traced out by a point on the circumference of a circle of radius R as the circle rolls along the x-axis. The cycloid is defined by the parametric equations

$$\begin{aligned} x(\theta, R) &= \quad R(\theta - \sin\theta) - 1 \\ y(\theta, R) &= -R(1 - \cos\theta) \end{aligned}$$

for $0 \leq \theta \leq \theta_1$ and θ_1 and R are the unique solutions to the nonlinear equations $x(\theta_1, R) = 1, y(\theta_1, R) = \beta$. Following [5] one can compute the optimal transit time T by rederiving Equation (3) when γ is represented by the parametric equations $(x, y) = (x(v), y(v))$, for $v_0 \leq v \leq v_1$, to obtain

$$T = \frac{1}{\sqrt{2g}} \int_{v_0}^{v_1} \frac{\sqrt{x'(v)^2 + y'(v)^2}}{\sqrt{-y(v)}}\, dv.$$

For the cycloid defined above the integrand simplifies to $\sqrt{2R}$ so $T = \theta_1 \sqrt{R/g}$.

Finite Dimensional Approximation

We seek an approximate solution to Problem (4) via Numerical Optimization. Instead of solving Problem (4) we will solve the slightly modified problem

$$\begin{array}{ll} \underset{f \in \mathcal{T}_N}{\text{minimize}} & \frac{1}{\sqrt{2g}} \int_{-1}^{1} \sqrt{\frac{1 + f'(x)^2}{-f(x)}}\, dx \\ \text{subject to} & f(x) \leq 0, \quad \forall x \in [-1, 1] \\ & f''(x) \geq 0, \quad \forall x \in [-1, 1] \\ & f(-1) = 0, f(1) = \beta \end{array} \tag{5}$$

Instead of searching over the space of differentiable functions $C^1[-1, 1]$, here we restrict ourselves to $\mathcal{T}_N$ the set of Chebyshev polynomials of degree N. We also impose an additional constraint that $f''(x) \geq 0$, or that f is a convex function. Problem (5) looks only slightly different than Problem (4); however by restricting ourselves to the set of Chebyshev polynomials we have moved from an infinite-dimensional space to a finite-dimensional space. We will see that it is possible to exactly transform the constraints in Problem (5) into linear equality, and inequality constraints, and numerically approximate the objective function.

Chebyshev Spectral Methods

The key to this transformation is Chebyshev spectral collocation [6]. The N Chebyshev points (Figure 3) are defined on the interval $[-1, 1]$ as

$$x_j = \cos\left(\frac{j\pi}{N}\right), \quad j = 0, 1, \ldots, N.$$

Let f be the unique polynomial of degree $\leq N$ with $f(x_j) = f_j$, $0 \leq j \leq N$, and let $\bar{f} \in \mathbb{R}^{N+1}$ be a vector containing the values f_j. A critical attribute of spectral methods is that differentiation is as easy as matrix multiplication. Let $D_N \in \mathbb{R}^{(N+1)\times(N+1)}$ be the Chebyshev differentiation matrix, then $u_j = f'(x_j)$ is given by

$$u = D_N \bar{f}$$

where $u \in \mathbb{R}^{N+1}$ is a vector containing the values u_j, $0 \leq j \leq N$. Similarly, it is possible to compute $v_j = f''(x_j)$ via $v = D_N^2 \bar{f}$.

It is also possible to evaluate integrals in the form

$$I = \int_{-1}^{1} g(x)\, dx \tag{6}$$

via spectral methods. We simply find the polynomial, p, of degree $\leq N$ such that $p(x_j) = g_j$, $0 \leq j \leq N$, and then I_N, an approximation of the integral I, is

$$I_n = \int_{-1}^{1} p(x)\, dx \tag{7}$$

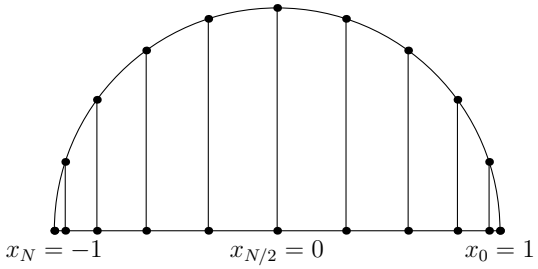

Figure 3: Chebyshev points are projections onto the x-axis of equally spaced points on the unit circle. Note they are numbered right to left. Taken from [6].

Furthermore, using either Clenshaw-Curtis or Fejér quadrature [6], it is possible to write (7) exactly as

$$I_n = w^T \bar{g},$$

where here $\bar{g} \in \mathbb{R}^{N+1}$ is a vector containing the values g_j, $0 \leq j \leq N$. Clenshaw-Curtis and Fejér quadrature each define different weight vectors $w \in \mathbb{R}^{N+1}$. Using Clenshaw-Curtis quadrature we can calculate (6) exactly if $g(x)$ is a polynomial of degree $\leq N$. Fejér quadrature is exact for polynomial integrands of degree $\leq N-2$ [6], but has the advantage of being an open integration rule. That is, it makes no use of the values of the integrand g_0 and g_N at the endpoints of the interval. We will make use of this fact later to define the objective function.

Numerical Optimization

Armed with Chebyshev spectral methods we can finally transform Problem (5) into a problem that we can solve with numerical optimization. Let $f(x)$ be a Chebyshev polynomial of degree $\leq N$ with $f_j = f(x_j)$ for $0 \leq j \leq N$ and $\bar{f} \in \mathbb{R}^{N+1}$ the vector of these values. Then we can transform Problem (5) into the problem

$$\begin{array}{ll} \underset{\bar{f} \in \mathbb{R}^{N+1}}{\text{minimize}} & \bar{w}^T g(\bar{f}) \\ \text{subject to} & D_N^2 \bar{f} \geq 0 \\ & \bar{\ell} \leq \bar{f} \leq \bar{u} \\ & f_0 = \beta, f_N = 0. \end{array} \tag{8}$$

Here $\bar{\ell}, \bar{u} \in \mathbb{R}^{N+1}$ are upper and lower bounds on the function values, $\bar{w} \in \mathbb{R}^{N+1}$ is a vector of weights for Fejér's second quadrature rule, and $g(\bar{f})$ is a function from $\mathbb{R}^{N+1} \to \mathbb{R}^{N+1}$ representing the integrand in Equation (3) sampled at the Cheybshev points with

$$g_j(\bar{f}) = \sqrt{\frac{1 + f_j'^2}{-f_j}}, \text{ with } f_j' = d_j^T \bar{f}$$

where d_j^T is the $j+1$ row of the Chebyshev differentiation matrix D_N. We can also write $g(\bar{f})$ more compactly in vector form as

$$g(\bar{f}) = \text{diag}(e + D_N \bar{f})^{1/2} \text{diag}(-\bar{f})^{-1/2} e$$

where e is the vector of all ones.

Singularity in the Integrand

As we noted before the integrand is singular at the left endpoint of integration. This means that $g_N = \infty$. This is the reason for choosing Fejér quadrature. With Fejér quadrature

$$I_N = \sum_{j=1}^{N-1} w_j g_j,$$

that is $w_0 = w_N = 0$. This means that are estimate of the integral in (3) will be finite. This would not be the case for a closed quadrature rule like Clenshaw-Curtis. However, just because we can approximate the integral doesn't mean the approximation is accurate. In particular we run into the fundamental difficulty of trying to approximate a singular integrand by a polynomial. Since polynomials only have poles at infinity this is necessarily a poor approximation. Perhaps a better (and more complicated) strategy would be to approximate the integrand with a rational function containing a pole at $x = -1$. The author briefly looked into this but could find no results for rational approximations with poles at the endpoints of the interval of integration. We will return to the accuracy of the quadrature rule when discussing the numerical results.

Eliminating the equality constraints

Although, Problem (8) is a problem in $N+1$ variables, the constraints $f_0 = \beta$ and $f_N = 0$ specify values for two of variables. Thus, it is simpler to remove these variables and instead optimize over $n = N-1$ variables. Let $f \in \mathbb{R}^n$ denote the vector with $f_j = f(x_j)$, for $1 \leq j \leq N-1$. The vector f will be the decision variables.

After eliminating these variables we must take care to differentiate correctly by incorporating the constant values of f_0 and f_N. For example the convexity constraint $D_N^2 \bar{f} \geq 0$ corresponds to

$$D_N^2 \bar{f} \equiv \begin{bmatrix} d_{11} & d_1^T & d_{1N} \\ d_{2:N-1,1} & \tilde{D}_N^2 & d_{2:N-1,N} \\ d_{N1} & d_n^T & d_{NN} \end{bmatrix} \begin{bmatrix} f_0 \\ f \\ f_N \end{bmatrix} \geq 0$$

Since $f_0 = \beta$ and $f_N = 0$ are fixed, the above constraint now becomes the three constraints

$$\begin{aligned} d_1^T f &\geq -\beta d_{11} \\ \tilde{D}_N^2 f &\geq -\beta d_{2:N-1,1} \\ d_n^T f &\geq -\beta d_{N1}. \end{aligned}$$

Thus we are left with the final problem

$$\begin{array}{ll} \underset{f \in \mathbb{R}^n}{\text{minimize}} & w^T h(f) \\ \text{subject to} & d_1^T f \geq -\beta d_{11} \\ & \tilde{D}_N^2 f \geq -\beta d_{2:N-1,1} \\ & d_n^T f \geq -\beta d_{N1} \\ & \ell \leq f \leq u \end{array} \tag{9}$$

Here $h : \mathbb{R}^n \to \mathbb{R}^n$ is a function representing the integrand evaluated on the Chebyshev points in the interior of the interval $[-1, 1]$.

Calculating the gradient

It is possible to compute the analytic gradient of the objective function $w^T h(f)$. Let $J \in \mathbb{R}^{n\times n}$ denote the Jacobian of the function $h(f)$. Then $\nabla_f(w^T h(f)) = J^T w$. If $h(f)$ is given by

$$\begin{aligned} f' &= \beta D_N(2:\mathtt{N},1) + \tilde{D}_N f \\ v_{\text{numer}} &= \sqrt{1+f'^2} \\ v_{\text{denom}} &= 1/\sqrt{-f} \\ h(f) &= \text{diag}(v_{\text{numer}})\text{diag}(v_{\text{denom}})e \end{aligned}$$

where $\tilde{D}_N \in \mathbb{R}^{n\times n}$ is the Chebyshev differentiation matrix D_N with the first and last rows and columns deleted (i.e. $\tilde{D}_N = D(2:\mathtt{N}, 2:\mathtt{N})$). Then the Jacobian is given by

$$J = \text{diag}(v_{\text{denom}})\text{diag}(f')\text{diag}(v_{\text{numer}})^{-1}\tilde{D}_N + \frac{1}{2}\text{diag}(v_{\text{denom}})^3\text{diag}(v_{\text{numer}}). \tag{10}$$

The most common mistake made in numerical optimization is incorrectly computing the gradient. The author verified Equation (10) with analytic derivatives computed by MATHEMATICA for small values of N. Numerical verification was also performed for small values of N by using the option `DerivativeCheck` in MATLAB's `fmincon` function. In each case the derivative given in Equation (10) was verified to more than six decimal places. For slightly larger values of N (e.g. $N \geq 20$) a discrepancy of $\simeq$ `1e-3` was noticed in the last component of the gradient (corresponding to f_{N-1}) when compared to a finite-difference estimate. The objective function is very sensitive to changes in f_{N-1}, and the author believes that the discrepancy is caused by a poor choice of finite-difference interval rather than an error in the analytic gradient.

A Quasi-Newton method with linear inequality constraints

Problem (9) is in the form

$$\begin{array}{ll} \underset{x\in\mathbb{R}^n}{\text{minimize}} & F(x) \\ \text{subject to} & Ax \geq \ell \end{array} \tag{11}$$

where here $F : \mathbb{R}^n \to \mathbb{R}$ and $A \in \mathbb{R}^{m_L\times n}$. We were able to calculate an analytic expression for the gradient $\nabla F(x)$ via Equation (10). However, obtaining second derivatives and thus the Hessian $\nabla^2 F(x)$ seems like a daunting task. Furthermore, it is often unclear how to incorporate second derivative information if the Hessian is indefinite. Therefore, a Quasi-Newton method was created to solve Problem (11). Specifically, Algorithm 2.2.4 Quasi-Newton IQP (Inequality constrained Quadratic subProblem) `QNIQP` described in [2, p.146] was implemented. At each iteration of the algorithm the convex quadratic subproblem

$$\begin{array}{ll} \underset{p_k\in\mathbb{R}^n}{\text{minimize}} & g_k^T p_k + \frac{1}{2}p_k^T B_k p_k \\ \text{subject to} & Ap_k \geq -v_k \end{array} \tag{12}$$

is solved to obtain a search direction p_k. Here B_k is a symmetric positive-definite approximation to the Hessian $\nabla^2 F(x_k)$, $g_k = \nabla F(x_k)$, and $v_k = Ax_k - \ell \geq 0$. Problem (12) seeks a step p_k that minimizes the local quadratic model of $F(x_k)$ subject to the constraint that the full step $x_k + p_k$ must be feasible (i.e. $A(x_k + p_k) \geq \ell$).

Solving the convex QP subproblem

The author choose to solve the convex subproblem (12) via an infeasible interior-point predictor-corrector method. The optimality conditions for Problem (12) are given by

$$\begin{aligned} Ap + s &= -v && \text{Primal} \\ g + Bp &= A^T y && \text{Dual} \\ y^T s &= 0 && \text{Complementarity} \\ y, s &\geq 0, \quad p \text{ free} \end{aligned}$$

At each iteration in the subproblem the following system was solved to determine a step $(\Delta p, \Delta s, \Delta y)$

$$\begin{bmatrix} B & & -A^T \\ A & -I & \\ & Y & S \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta s \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_d \\ r_p \\ r_c \end{bmatrix}$$

where $Y = \text{diag}(y)$ and $S = \text{diag}(s)$, and r_p, r_d, and r_c are residuals for the Primal, Dual, and Complementarity equations respectively. This is equivalent to the reduced system

$$\begin{bmatrix} B & -A^T \\ YA & S \end{bmatrix} \begin{bmatrix} \Delta p \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_d \\ r_c + Y r_p \end{bmatrix} \quad \Delta s = A\Delta p - r_p$$

which we can further reduce to the problem

$$\begin{aligned} (B + A^T D^2 A)\Delta p &= r_d + AS^{-1}(r_c + Y r_p) \\ \Delta s &= A\Delta p - r_p \\ \Delta y &= S^{-1}(r_c - Y\Delta s) \end{aligned} \tag{13}$$

where $D^2 = S^{-1}Y$. We can solve for Δp in Equation (13) by computing a Cholesky factorization of the matrix $B + A^T D^2 A$, or if we know the Cholesky factorization $R^T R = B$ we can compute Δp by solving the equivalent least squares problem [4, p.62]

$$\underset{\Delta p \in \mathbb{R}^n}{\text{minimize}} \left\| \begin{bmatrix} DA \\ R \end{bmatrix} \Delta p - \begin{bmatrix} r_p \\ v \end{bmatrix} \right\| \quad \text{where } Rv = r_d + A^T S^{-1} r_c \tag{14}$$

Using Equation (14) would be advantageous if the Hessian approximation B stored (and updated) via its Cholesky factor R [2, p.58], Equation (14) also avoids the loss in precision caused by forming the product $A^T D^2 A$.

Quasi-Newton update

Let $s_k = x_{k+1} - x_k$ and $y_k = \nabla F(x_{k+1}) - \nabla F(x_k)$. If $y_k^T s_k > 0$ and B_k is the current approximation to the Hessian, the updated Hessian approximation B_{k+1} is given by the BFGS (symmetric, positive-definite rank two) update:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{15}$$

Rather than updating B_k via Equation (15) it is possible to update the Cholesky factor R_k, where $R_k^T R_k = B_k$.

Step-length (line search) algorithm

When using a Quasi-Newton method, to update the approximate Hessian B_k we must have that $y_k^T s_k > 0$. This condition holds provided $\alpha_k \in \Gamma(\eta, \mu)$ [2, p.58]. Thus we need a step-length selection algorithm that can ensure such a condition holds. A step-length algorithm based on [1] and the `MINPACK2` library translated into MATLAB [3] was used.

Verification of QNIQP

To verify the solver `QNIQP` accurately computes optimal solutions we generated random convex quadratic programs in the form

$$\begin{array}{ll} \underset{x \in \mathbb{R}^n}{\text{minimize}} & \frac{1}{2} x^T H x + c^T x \\ \text{subject to} & Ax \geq \ell \end{array}$$

The Hessian was not provided to `QNIQP` merely the gradient $c + Hx$. The computed optimal objective values were compared with MATLAB's solver `fmincon`. Since the test problems were strictly convex the optimal objective value is guaranteed to be unique. More than eight digits of accuracy were obtained on small problems $n = 10$ in under 25 iterations. The function `testqniqp` is provided in Appendix **??**. A similar method was used to verify the subproblem solver.

Sample output of the solver `QNIQP` on a test problem is shown overleaf:

```
QNIQP
 iter    f(x)          ||vh||     ||Zgz||     ||p||      g'p        ||x||   miter   stp
   0 +9.4938910e+00   4.7e+00    0.0e+00   4.7e+00  -4.8e+01   1.6e+00      6   4.1e-01
   1 -3.1875422e-01   0.0e+00    9.3e-01   9.3e-01  -8.6e-01   3.5e-01      5   1.0e+00
   2 -9.5737331e-01   1.1e+00    4.1e-01   1.5e+00  -8.6e-01   9.8e-01      7   1.0e+00
   3 -1.6038107e+00   1.4e-01    4.8e-01   3.2e-01  -1.8e-01   2.4e+00      9   1.0e+00
   4 -1.7587361e+00   2.2e-02    2.9e-01   3.7e-01  -1.1e-01   2.5e+00      7   1.0e+00
   5 -1.8302322e+00   2.2e-07    1.2e-01   2.9e-01  -3.4e-02   2.4e+00      6   1.0e+00
   6 -1.8546136e+00   5.1e-02    7.8e-02   7.8e-02  -7.7e-03   2.5e+00      4   1.0e+00
   7 -1.8617517e+00   5.3e-07    5.3e-02   8.5e-02  -4.2e-03   2.5e+00      4   1.0e+00
   8 -1.8643654e+00   4.7e-08    3.2e-02   7.1e-02  -1.6e-03   2.5e+00      4   1.0e+00
   9 -1.8656835e+00   6.4e-09    4.0e-02   2.4e-01  -4.8e-03   2.5e+00      5   1.0e+00
  10 -1.8690301e+00   1.2e-09    5.1e-02   2.3e-01  -3.4e-03   2.5e+00      5   1.0e+00
  11 -1.8714235e+00   1.0e-09    4.1e-02   1.7e-01  -2.2e-03   2.6e+00      6   1.0e+00
  12 -1.8728465e+00   4.7e-11    1.9e-02   3.8e-02  -5.1e-04   2.7e+00      5   1.0e+00
  13 -1.8731591e+00   2.9e-09    6.6e-03   1.5e-02  -7.0e-05   2.7e+00      6   1.0e+00
  14 -1.8732001e+00   7.9e-09    2.6e-03   4.9e-03  -6.1e-06   2.7e+00      4   1.0e+00
  15 -1.8732038e+00   4.1e-07    8.5e-04   1.1e-03  -9.1e-07   2.7e+00      4   1.0e+00
  16 -1.8732043e+00   4.4e-09    2.6e-04   7.3e-04  -1.2e-07   2.7e+00      4   1.0e+00
  17 -1.8732044e+00   3.9e-09    1.2e-04   2.4e-04  -1.8e-08   2.7e+00      4   1.0e+00
  18 -1.8732044e+00   3.9e-09    3.9e-05   5.8e-05  -1.5e-09   2.7e+00      4   1.0e+00
  19 -1.8732044e+00   4.0e-09    5.1e-06   2.5e-05  -7.2e-11   2.7e+00      4   1.0e+00
  20 -1.8732044e+00   4.0e-09    3.5e-07   3.6e-06  -6.4e-12   2.7e+00      4   Done!
```

The output displays the value of the objective function $f(x)$, the residuals for the constraints in the active set (represented by the matrix A_{act} a subset of the columns of the matrix A) $\| \hat{v} \| = \| A_{\text{act}} x - b \|$, the reduced gradient $\| Z g_z \| = \left\| A_{\text{act}}^T y - g \right\|$, the length of the search direction $\| p \|$, the directional derivative $g^T p$, the norm of the decision variables $\| x \|$, the number of minor iterations in the convex quadratic subproblem, and the step-length α.

Numerical Results

Accuracy of the Objective function

To evaluate the accuracy of the objective function, we compare the values computed by the numerical quadrature rule to analytic values for simple curves γ. When γ is a straight-line between A and B, $\gamma \equiv f(x) = (\beta/2)x + \beta/2$ and Equation (3) simplifies to

$$T = \int_{-1}^{1} \sqrt{\frac{1 + (\frac{\beta}{2})^2}{-(\frac{\beta}{2}x + \frac{\beta}{2})}}\, dx = \sqrt{\frac{4+\beta^2}{-2\beta}} \int_{-1}^{1} \frac{dx}{\sqrt{1+x}} = \frac{2\sqrt{4+\beta^2}}{\sqrt{-\beta}}$$

where here we take the force of gravity $g = 1/2$ to avoid the constant multiple. We also know that the transit time for the cycloid is given by $T = \theta_1\sqrt{2R}$. Thus, we can compare these analytical values to the value of the objective function for different values of N and β to assess the accuracy of the objective function. Table 1 shows the results (pairs of absolute errors). Note that the error in the objective function does not seem to be a function of the parameter β. However, Table 1 indicates that for small values of N the best we can hope for is two digits of accuracy in the objective function. And we shouldn't expect more the one!

N/β	$N = 10$	100	400	1000
$\beta = -1/10$	0.2, 0.4	.02,.09	.006,.04	.002,.02
-1/2	0.2, 0.4	.02,.09	.006,.04	.002,.02
-1	0.2, 0.4	.02,.09	.006,.04	.002,.02
-2	0.2, 0.4	.02,.09	.006,.04	.002,.02

Table 1: Absolute error $|T_{\text{obj}}(\gamma) - T_{\text{any}}(\gamma)|$ for $\gamma = \gamma_{\text{str}}$ (the straight line) and $\gamma = \gamma_{\text{cyc}}$ (the cycloid) for different values of β and N.

Bound Constraints

Although we avoid evaluating the integrand at $x = -1$ where it is singular, we expect the integrand to be inaccurate for values of x close to -1. Figure 4 shows a plot of the integrand, note the large values near the left endpoint of the interval. Unfortunately, the solver exploits these inaccuracies producing an "optimal" brachistochrone in which the values of f_{N-1} and f_{N-2} drop extremely rapidly (the blue curve in Figure 5). The transit time for this "optimal" curve is $\simeq 3.22$, less than the true optimal value of 3.6426 obtained via the analytic formula for the cycloid $T = \theta_1\sqrt{2R}$. Clearly, the inaccuracy in the objective function is rearing its ugly head. We can overcome these numerical difficulties by placing a lower bound on the value of f_{N-1}. Problem (9) contains the constraints $\ell \le f \le u$. Thus it simple to add the constraint $f_{N-1} \ge \alpha$, for some parameter α. By choosing $\alpha = f_{N-1}^{\text{cyc}}$ we can make sure the computed brachistochrone does not fall below the cycloid for the first point f_{N-1}. The other values of ℓ were chosen to be $3\beta/2$ to ensure that the f did not get large during intermediate iterates. Values of u were set to $-\epsilon = -1\text{e}{-6}$. This is to avoid producing small values for $-f_j$ in the denominator of the expression $1/\sqrt{-f_j}$. In particular we do not want the solver to evaluate the objective function on vectors f with $f_j \ge 0$ for which it is not defined.

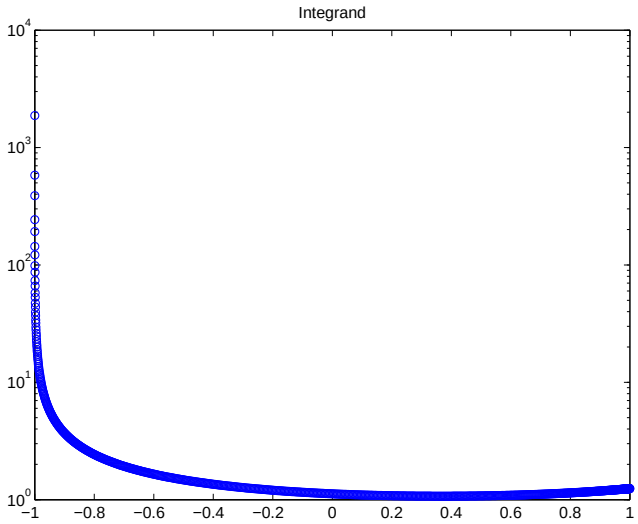

Figure 4: The integrand in Equation (3) evaluated at points in the interior of the interval $[-1, 1]$. Note the large values near the left endpoint and singularity.

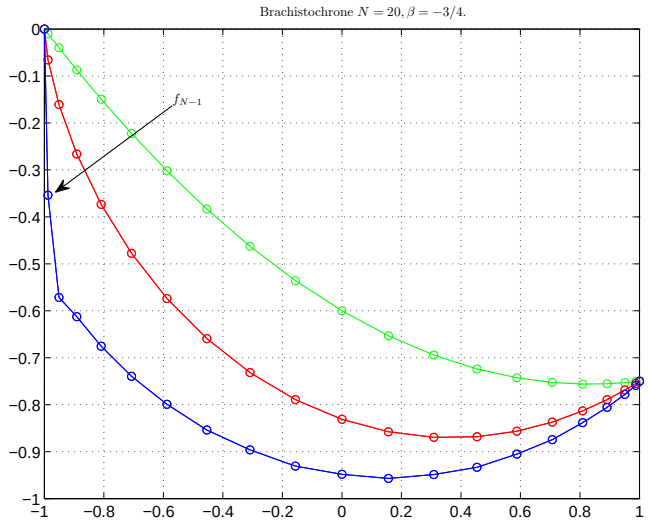

Figure 5: Cycloid (red), computed optimal brachistochrone (blue), starting guess (green). Note the value of f_{N-1} for the computed optimal brachistochrone is much lower than the cycloid value of for f_{N-1}.

Computed Brachistochrone

With all this machinery in place we are finally able to calculate Brachistochrones. Figure 6 shows 7 brachistochrones calculated for values of β between $-3/4$ and -2 for $N = 20$ and $N = 200$. Sample output for $\beta = -3/4$ and $N = 20$ is shown below:

```
QNIQP
 iter    f(x)          ||vh||     ||Zgz||     ||p||     g'p        ||x||    miter   stp
   0 +3.8897586e+00   1.7e+00    2.4e-01    9.5e-01  -2.4e+00    2.5e+00     16   1.0e+00
   1 +3.6560868e+00   4.4e-07    1.6e+00    7.1e-01  -5.6e-01    2.9e+00     13   1.0e+00
   2 +3.6464113e+00   2.9e+01    4.1e-01    5.9e-01  -4.4e-01    3.4e+00     12   1.0e+00
   3 +3.5548934e+00   1.4e+01    3.0e-01    5.4e-01  -2.7e-01    3.5e+00     12   1.0e+00
   4 +3.4793814e+00   1.0e+01    3.9e-01    4.3e-01  -1.6e-01    3.4e+00     11   5.1e-01
   5 +3.4343803e+00   4.7e+00    1.3e+00    3.6e-01  -9.3e-02    3.3e+00     11   5.0e-01
   6 +3.4086068e+00   4.8e+01    1.7e-01    2.0e-01  -2.6e-02    3.2e+00     10   5.2e-01
   7 +3.4007560e+00   9.2e-10    1.4e+00    1.1e-01  -7.9e-03    3.1e+00      7   4.3e-01
   8 +3.3990243e+00   1.4e+00    9.0e-02    1.5e-01  -6.6e-03    3.1e+00      9   2.9e-01
   9 +3.3980477e+00   4.0e-05    4.3e-02    3.9e-02  -7.9e-04    3.0e+00     11   1.8e-01
  10 +3.3979751e+00   3.5e-05    4.9e-02    4.1e-02  -8.6e-04    3.0e+00     11   1.9e-01
  11 +3.3978934e+00   3.0e-05    2.8e-02    2.0e-02  -2.8e-04    3.0e+00     13   7.7e-02
  12 +3.3978826e+00   1.9e-09    1.4e+00    2.7e-03  -1.8e-05    3.0e+00     11   1.0e+00
  13 +3.3978811e+00   5.9e-05    3.0e-01    1.3e-03  -9.2e-06    3.0e+00     11   1.0e+00
  14 +3.3978769e+00   8.7e-06    3.0e-01    3.1e-04  -4.2e-07    3.0e+00     11   1.0e+00
  15 +3.3978768e+00   2.6e-05    5.3e-04    9.0e-05  -5.1e-08    3.0e+00     11   1.0e+00
  16 +3.3978768e+00   5.3e-06    3.0e-01    1.0e-05   5.3e-10    3.0e+00     11   Done!
```

Note here that the last subproblem computes a search direction p for which $\| p \|$ is small and $g^T p > 0$. This is used in a termination condition since we know $p = 0$ is feasible for Problem (12) and yields a objective value of zero, thus the subproblem has computed a suboptimal solution. If $p = 0$ than it follows from the optimality conditions of the subproblem that we are at a first order Kuhn-Tucker point of Problem 11 [2, p.146].

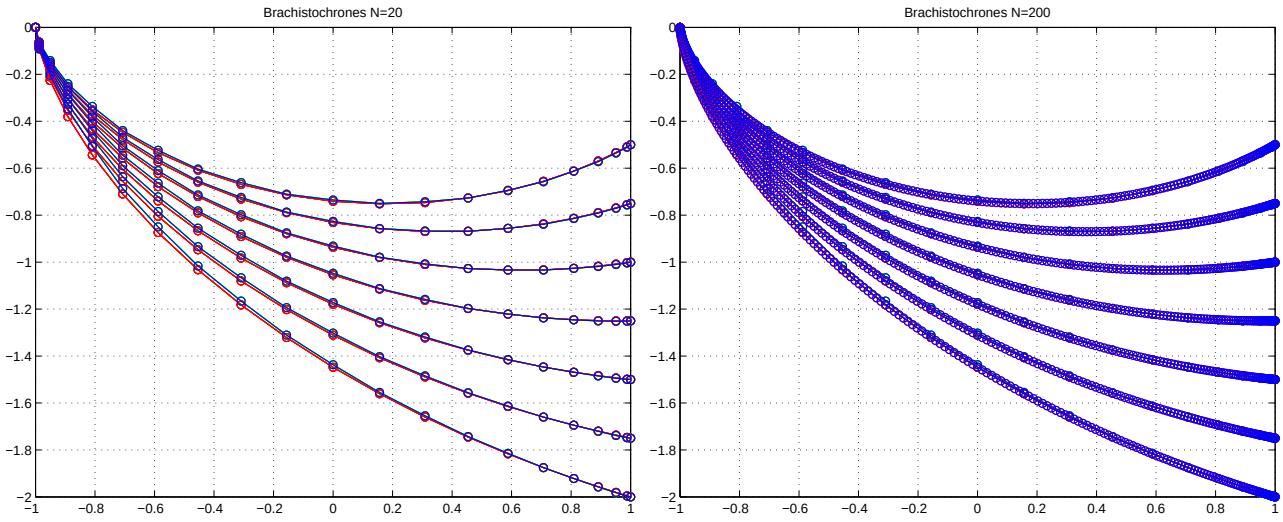

Figure 6: Cycloids (red) and computed optimal brachistochrones (blue) for $N = 20$ and $N = 200$.

The solver `QNIQP` is able to solve medium-sized problems. Thus we can easily compute brachistochrones for values of $N \simeq 500$. These are large polynomials! Figure 7 shows the computed brachistochrone for $N = 450$. The absolute error between the cycloid at the computed optimal solution is shown in Figure 8. Note that the largest error of `3e-2` occurs at f_{N-5} near the left endpoint of integration.

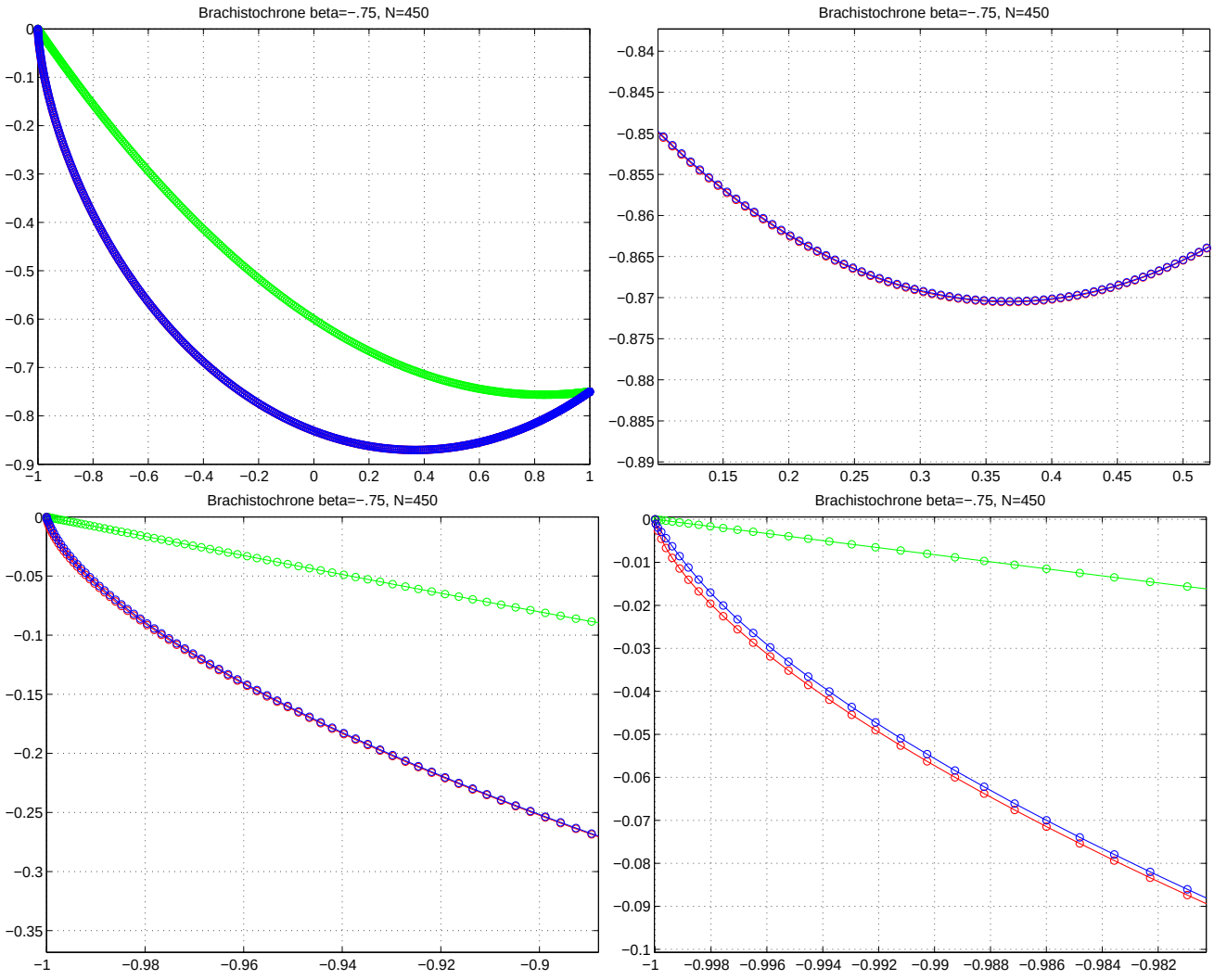

Figure 7: Cycloid (red), computed optimal brachistochrone (blue), initial condition (green) for $N = 450, \beta = -3/4$. Four different zoom levels are presented (note the scales). The largest errors in the computed brachistochrone occur near $x = -1$ near the singularity in the objective function.

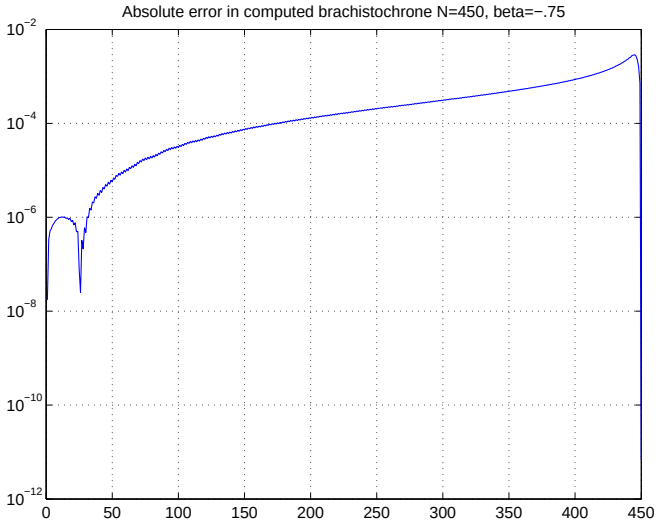

Figure 8: Absolute error in the compute optimal brachistochrone for $N = 450, \beta = -3/4$. Note that the largest error (2.8e−3) occurs in f_{N-5} near the left endpoint of integration.

Discussion

Computing a numerical solution to the brachistochrone problem is difficult. The author choose to use Chebyshev Spectral methods to solve the problem. Chebyshev spectral methods provide exact differentiation and integration of polynomials of degree $\leq N$. As such they are well suited for accurately evaluating the integrand. Unfortunately, because the integrand is singular at the left endpoint, spectral methods do not provided very accurate estimates of the integral objective function (Equation (3)). Nevertheless, by imposing linear and simple bound constraints these methods yield fairly accurate solutions.

A method for minimizing nonconvex functions subject to linear constraints was constructed. This method requires only the gradient of the objective function (no second derivative information is necessary). The method relies on solving a convex quadratic subproblem with linear inequality constraints. The convex subproblem is solved via a infeasible primal-dual interior point method. Although this interior point method requires computing more Newton steps than a active-set method near a minimizer the implementation is straightforward and efficient on medium scale problems.

A Brachistochrone Code

```
%Nice plots!
set(0,'defaultaxesfontsize',12, 'defaultaxeslinewidth',.7,...
      'defaultlinelinewidth',.8,'defaultpatchlinewidth',.7);
for beta = -.5:-.25:-2.
% N is the number of chebyshev points
% x_j = cos(j pi/N)   j = 0, 1, ... , N
N = 20;
str = sprintf('Brachistochrones␣N=%d',N);
title(str);
[D,x] = cheb(N);

% beta is the y-value of the endpoint (i.e. f(1) = beta)
%beta = -1.2;

% Construct a initial guess for the function based on a parabola.
eta = .8;
parabola = @(x) (-eta*beta+beta/2)*x.^2 + beta/2*x + eta*beta;
figure(1)
%clf
plot(x,parabola(x),'go-')
grid on
hold on

% calculate the cycloid
func = @(x) [x(1)*x(2)-x(1)*sin(x(2))-1-1;-x(1)*(1-cos(x(2)))-beta];
[xstar]=fsolve(func, [2*pi - 1; 1]);
r = xstar(1)
t = xstar(2)
thetavals = linspace(0,t,10000);
xcoord = r*(thetavals-sin(thetavals)) - 1;
ycoord = -r*(1-cos(thetavals));
fcyc   = interp1(xcoord,ycoord,x,'spline');
fcycprime = D*fcyc;

plot(x,fcyc,'ro-')
fcyc([1,end]) =[];
[fstar,gradstar] = brach(fcyc,beta);
fstar
norm(gradstar)



% Set up and solve the optimization problem
% Remeber we remove the variables f(-1) = f_0 and f(1) = f_N.
% So we are going to optimize over N-1 variables.
n = N - 1;
e = ones(n,1);
f0 = parabola(x);
% Remove f_0 and f_N
f0(1) = [];  f0(end) = [];
options = optimset('MaxFunEvals',10000,'MaxIter',500,'GradObj','on', ...
                   'DerivativeCheck','on','Display','iter', ...
                   'FunvalCheck', 'on', 'Diagnostics', 'on', ...
                   'LargeScale','off',...
                   'PlotFcns', @(x,y,z) optimplotbrach(x,y,z,beta));

% Create the convexity constraint
D2 = D^2;
delta = D2(1,1);
gamma = D2(end,1);
d1    = D2(1,2:end-1);
dn    = D2(end,2:end-1);
D2([1,end],:) = [];
D1    = D2(:,1);
D2(:,[1,end]) = [];


A = [ d1; D2; dn];
b = [-delta*beta; -beta*D1; -gamma*beta];

% add simple bound constraints l <= x <= u to A and b; we are gonna use the
% form Ax >= b, and then convert when needed to fmincons -Ax <= -b
% Here l = 1.2*beta*e <= x <= -1e-2*e = u
A = [A; eye(n); -eye(n)];
% hack, the solver is exploiting the error in the numerical integration
% we add a constraint to prevent this from happening; the first function
% value can't be below the analytic solution. This prevents the solver from
% exploiting the singularity, but it is probably cheating.

z = 1.5*beta*e;
z(n) = fcyc(end);

b = [b; z; 1e-6*e];



% uncomment below to provide simple bound constraints to fmincon
%[f,objval,exitflag,output,lambda,grad,hessian]= ...
% fmincon(@(f) brach(f,beta),f0,-A,-b,[],[],1.2*beta*e,-1e-2*e,[],options);

% [f,objval,exitflag,output,lambda,grad,hessian]= ...
%     fmincon(@(f) brach(f,beta),f0,-A,-b,[],[],[],[],[],options);
%
% f = [beta; f; 0];

figure(1)
hold on
%plot(x,f,'go-')
```

```
% Alright, here goes nothing baby!

myf = qniqp(@(f) brach(f,beta), A,b, f0,1e-6);
myf = [beta; myf; 0];
plot(x,myf,'bo-')


end
```

```
function [obj,grad] = brach(f,beta)
%        [obj,grad] = brach(f,beta)
%  Computes the objective function and gradient for the brachistochrone
%  problem.
%
%  minimize        T = \int_{-1}^{1} \sqrt{\frac{1 + (f'(x))^2}{-f(x)} } dx
%  f \in C[-1,1]
%  subject  to     f(-1) = 0, f(1) = beta
%                  f(x) <= 0, x \in [0,1]
%
%  We are solving the problem via Chebyshev collocation.
%  The vector f represents the function values at the collocation
%  points
%  x_j = cos(j*pi/N)   j = 0, 1, ..., N
%  f_j = f(x_j)        j = 0, 1, ..., N
%
%  We will approximate the definite integral above with a quadrature rule
%  T   =  w'*g
%  where  w is a set of weights, and g is the integrand sampled at the
%  collocation points.
%
%  g_j  =  sqrt((1+f'(j)^2)/-f(j))   j = 0, 1, ..., N
%
%  Note that the integrand above has a singularity at the left endpoint
%  since we require that f(0)=0.However, this is an integrable singularity.
%
%  We will use Fejer quadrature which doesn't evaulate the integrand
%  at the left and right endpoints (i.e. w_0 = w_N = 0) to avoid this
%  problem.
%
%  In addition since we have that f(-1) = f_0 =  0 and f(1) = f_N = beta.
%  We remove   the variables f_0 and f_N.  So the vector f is a vector of
%  length N-1.

nm1 = length(f);
% Calculate the chebyshev differentiation matrix D
[D,x]    = cheb(nm1+1);                                          %#ok
% Calculate the Fejer weigths
[wf1, w] = fejer(nm1+1);

% To calculate f'_1 and f'_N-1 we need to add back in f_0 and f_N
%fprime   = D*[beta; f; 0];
fprime   = fastdiff([beta; f; 0]);
% Now we remove them again to avoid singularites in the computation.
fprime   = fprime(2:end-1);
numer    = sqrt(1 + fprime.^2);
denom    = 1./sqrt(-f);
g        = numer.*denom;


% Lop off the the w_0 = w_N = 0 entries from w.
w        = w(2:end-1);
obj      = w'*g;

% if ~isreal(obj)
%     obj
% end
% correction = @(c,h) 2*sqrt(-(1+c^2)*h/c);
% if fprime(end) < 0
%     obj = obj+ correction(fprime(end),x(end-1)-x(end));
% end
    %testquad = @(v) interp1(x(2:end-1),g,v);
%obj = quad(testquad,-1+eps,1)


if nargout > 1
    D(1,:) = [];
    D(end,:) = [];
    D(:,1)   = [];
    D(:,end) = [];
    J    = diag((denom.*fprime)./numer)*D;
    J    = J + 1/2*diag(denom.^3.*numer);
    grad = J'*w;
end
```

```
% CHEB  compute D = differentiation matrix, x = Chebyshev grid

  function [D,x] = cheb(N)
  if N==0, D=0; x=1; return, end
  x = cos(pi*(0:N)/N)';
  c = [2; ones(N-1,1); 2].*(-1).^(0:N)';
  X = repmat(x,1,N+1);
  dX = X-X';
  D  = (c*(1./c)')./(dX+(eye(N+1)));      % off-diagonal entries
  D  = D - diag(sum(D'));                 % diagonal entries
```

```
function Dx=fastdiff(x);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% fastdiff.m
%
% Numerically differentiate a function which has been evaluated at the
% Chebyshev-Gauss-Lobatto points and stored in the vector x.
%
% Equivant to D*x where D is the Chebyshev collocation matrix, but much
% faster when x contains many elements. The differentiation is performed
% in the spectral domain, by inverting the integral operator on a subspace.
% Since the integral operator has tridiagonal form, the entire
% differentiation process is O(N*log(N)).
%
% Reference on Chebyshev integration operators:
%
% "Integration preconditioners for differential operators in spectral tau-
% methods" by E. A. Coutsias, T. Hagstrom, J. S. Hesthaven, and D. Torres.
% Houston Journal of Mathematics p.21-38 (1996)
% Available here: http://www.math.unm.edu/~vageli/papers/ico.ps
%
% Written by: Greg von Winckel - 09/14/05
% Contact: gregvw(at)math(dot)unm(dot)edu
% URL: http://math.unm.edu/~gregvw
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


N1=length(x); N=N1-1; k=1:N1; N2=N1+1;
band=[0 0 1./(2*(1:N2))]';
B=spdiags([band(k+2) -band(k)],[-1,1],N1,N1);
B(2,1)=1;
q=real(ifft([x(k);x(N:-1:2)]));
y=([q(1); 2*q(2:N); q(N1)]);
Dy=[B(2:N1,1:N)\y(2:N1);0];
q=fft([Dy(1); [Dy(2:N)/2; Dy(N1); Dy(N:-1:2)/2]]);
Dx=real(q(k));
```

```
function [wf1,wf2,wcc] = fejer(n)
% Weights of the Fejer2, Clenshaw-Curtis and Fejer1 quadratu
% by DFTs. Nodes: x_k = cos(k*pi/n), n>1
N=[1:2:n-1]'; l=length(N); m=n-l; K=[0:m-1]';
% Fejer2 nodes: k=0,1,...,n; weights: wf2, wf2_n=wf2_0=0
v0=[2./N./(N-2); 1/N(end); zeros(m,1)];
v2=-v0(1:end-1)-v0(end:-1:2); wf2=ifft(v2);
wf2 = [wf2; 0.];
%Clenshaw-Curtis nodes: k=0,1,...,n; weights: wcc, wcc_n=wc
g0=-ones(n,1); g0(1+l)=g0(1+l)+n; g0(1+m)=g0(1+m)+n;
g=g0/(n^2-1+mod(n,2)); wcc=ifft(v2+g);
wcc = [wcc; wcc(1)];
% Fejer1 nodes: k=1/2,3/2,...,n-1/2; vector of weights: wf1
v0=[2*exp(i*pi*K/n)./(1-4*K.^2); zeros(l+1,1)];
v1=v0(1:end-1)+conj(v0(end:-1:2)); wf1=ifft(v1);
```

B Solver code

```
function [x,y,info] = qniqp(f,A,b,x0,tol)
%        [x,y,info] = qniqp(f,A,b,x0)
% Quasi-Newton IQP (inequalitiy constrained quadratic subproblem)
% method for Linear Inequality Constrained noncovex optimization
%
% Man that is a mouthful! This program solves the problem
%
% minimize     f(x)
% subject to   Ax >= b
%
% Here f : R^n -> R,   A is a m x n matrix of constraints and b is a m
% vector.
%
% Input:
% f             A Matlab function handle, f, should be in the form
%               [obj,grad] = f(x)
%               That is it also returns the gradient when two output
%               arguments are requested
% A             A dense m x n matrix of the linear constraints
% b             A dense m vector, the rhs
% x0            A inital starting guess for x. Does not necessarily need to
%               be feasible (i.e A*x0 >= b)
% tol           A tolerance for the reduced gradient and the constraint
%               violation



n = size(x0,1);
x = x0;
B = eye(n);

k = 0;
converged = 0;
fprintf('\n␣QNIQP␣\n')
fprintf('␣␣iter␣␣␣␣f(x)␣␣␣␣␣␣␣␣␣||vh||␣␣␣␣␣||Zgz||␣␣␣␣||p||␣␣␣␣␣g''p␣␣␣␣␣␣␣␣||x||␣␣␣miter␣␣␣stp\n')
```

```
while ~converged
    v = A*x - b;
    if any(v < 0)
        error('negative v')
    end
    [obj,g] = f(x);
    [p,y,s,sinfo,iter] = cqp(g, B, A, v);
    if sinfo~=1
        info = sinfo;
        fprintf('\n\nSubproblem error:%d\n',info);
        break
    end

    if g'*p > 0
        norm(p);
    end
    workingset = (s < 10*tol);
    Aa = A(workingset,:);

    [m, nn] = size(Aa);

    Z = null(Aa);
%
%     [mz, nz] = size(Z);
%     if nz ~= nn - m
%         fprintf('Not full rank')
%     end
%
    [obj,g] = f(x);

    vhat = Aa*x - b(workingset);
%
    gz = Z'*g;
%
    Zgz = Aa'*y(workingset) - g;

    str = '%5g %+11.7e %9.1e  %9.1e %9.1e %9.1e %9.1e %5g ';
    fprintf(str,k,obj,norm(vhat),norm(gz),norm(p),g'*p, norm(x),iter);
    if norm(vhat) < tol &&  norm(Zgz) < tol
        converged = 1;
    else
        converged = 0;
    end


    if ~converged
        [alpha,linfo] = linesearch(f,x,p,1,1);
        if linfo == 0 || linfo == 3 || linfo == 4 || linfo == 6
            info = linfo;
            fprintf('\n\nLinesearch error:%d\n',info);
            break
        end
        fprintf('%9.1e\n',alpha);
        [B,binfo] = bfgsupdate(f,B,x,x+alpha*p);
        if binfo~=1
            info = binfo;
            fprintf('\n\nBFGS error:%d\n',info);
            break
        end
        x = x + alpha*p;
        k = k + 1;
    else
        fprintf('Done!\n');
        info = 1;
        break
    end
end

%keyboard
```

```
function [p,y,s,info,k] = cqp(g,B,A,v)
%        [p,y,s,info,k] = cqp(g,B,A,v)
%
% Solves the (strictly) convex qp subproblem
% minimize     g'*p + 1/2*p'*B*p
% subject to   A*p >= -v  (v > 0 data)
%
% via a primal-dual interior point method.
%
% Input:
% g             vector of length n
% B             symmetric positive-definite n x n matrix
% A             m x n matrix of linear inequality constraints
% v             vector of length m with property that v > 0
%
% The dual of the above problem (in p and y)
% maximize      -v'*y - 1/2*p'*B*p
% subject to    A'*y  = B*p + g
%                   y >= 0
%
% The optimality conditions of the QP are given by
% A*p   -s        =  -v      ( Primal )
% B*p       -A'*y =  -g      ( Dual   )
%       y'*s      =   0      ( Complementarity )
%       y,  s    >=   0
%       p   free

n = size(g,1);
m = size(A,1);

k         = 0;
```

```
beta      = 0.99;
tol       = 1e-7;
converged = 0;
maxiter   = 50;
output    = 0;

% As we approach an optimal solution the following becomes the solution.
p  = zeros(n,1);
s  = v;
y  = ones(m,1);

s  = max(s,1e-3);

rp = -A*p - v + s;
rd = -g - B*p + A'*y;
rc = -y.*s;



if norm(rp,Inf) < 10*tol && norm(rd,Inf) < 10*tol && norm(rc,Inf) < 10*tol
    info = 1;
    k    = 1;   % We didn't really iterate but we had to solve linear system
                % so we might as well count that as a single iteration.
    return
end



mu = y'*s/n;

func  = @(p) g'*p + 1/2*p'*B*p;
if output
    fprintf('iter␣␣␣␣obj␣␣␣␣||rp||␣␣␣||rd||␣␣␣||rc||␣␣␣stp␣␣␣␣␣gap\n')
end


while ~converged  && k <= maxiter


% We are going to solve the following problem to determine a step
% (dp,ds,dy).
%     n     m     m
% n [ B          -A'] [ dp ]   [ rd ]
% m [ A    -I       ] [ ds ] = [ rp ]
% m [       Y     S ] [ dy ]   [ rc ]
%
% This is equivalent to the reduced system
%
% [ B  -A'] [ dp ]  = [ rd         ]
% [ YA  S ] [ dy ]    [ rc + Y*rp ]     ds = A*dp  - rp
%
% which we can further reduce to the problem
%
% (B + A'*S^{-1}*Y*A) dp = rd + A'*S^{-1}( rc + Y*rp)
% dy  = S^{-1}*(rc + Y*rp) - S^{-1}Y*A*dp
% ds  = A*dp - rp
%
% Note that if we know a Cholesky factorization of B = R'*R then we
% can solve for dp via the least-squares problem
%
% minimize || [ D*A ] dp -  [ rp ] ||   where w = rd + A'*S^{-1} rc
%   dp     || [   R ]       [ R\w] ||   and   D^2 = S^{-1}*Y

% First time through we compute the affine-step
R      = chol(B + A'*diag(y./s)*A);
rhs    = rd + A'*((rc + y.*rp)./s);
dpaff  = R\(R'\rhs);


dsaff  = A*dpaff - rp;
dyaff  = rc./s - (y./s).*dsaff;


% Now we compute the max step we can take with the affine-step
alphas  = maxstep(s,dsaff);
alphay  = maxstep(y,dyaff);
alpha   = min(alphas,alphay);

% Calculate the reduction in the mu do the affine-step
muaff   = (y + alpha*dyaff)'*(s + alpha*dsaff)/n;
sigma   = (muaff/mu)^3;
sigma   = min([sigma,1]);
% Calculate the center-corrector residual
rcc     = sigma*mu - dyaff.*dsaff;

% Calculate the new rhs, using the Cholesky factorization from the
% affine-step
rhs     = A'*(rcc./s);
dpcc    = R\(R'\rhs);

dscc    = A*dpcc;
dycc    = rcc./s - (y./s).*dscc;

% M = [ B zeros(n,m) -A'; A -eye(m,m) zeros(m,m); zeros(m,n) diag(y) diag(s)];
% bigv = M\[rd; rp; rcc];
%
% norm(bigv);

% The full step is the sum of the affine-step and the center-corrector step
dp      = dpaff + dpcc;
ds      = dsaff + dscc;
```

```
dy      = dyaff + dycc;

% Compute the max step we can take

alphas = maxstep(s,ds);
alphay = maxstep(y,dy);
alpha  = min(alphas,alphay);

alpha  = min(beta*alpha,1);

p      =  p + alpha*dp;
s      =  s + alpha*ds;
y      =  y + alpha*dy;

k = k + 1;

rp = -A*p - v + s;
rd = -g - B*p + A'*y;
rc = -y.*s;

mu = y'*s/n;

%This should be the duality gap
gap = g'*p + v'*y + p'*B*p;

if output
    fprintf('%3.d␣%+5.2e␣%5.2e␣%5.2e␣%5.2e␣%5.2e␣%5.2e\n',...
        k, func(p), norm(rp), norm(rd), norm(rc,Inf),alpha,gap)
end

if norm(rp,Inf) < tol && norm(rd,Inf) < tol && norm(rc,Inf) < tol
    converged = 1;
    if output
    fprintf('iter␣␣␣␣obj␣␣␣␣||rp||␣␣␣||rd||␣␣␣||rc||␣␣␣stp\n')
    fprintf('%3.d␣%+5.2e␣%5.2e␣%5.2e␣%5.2e␣%5.2e\n',...
        k, func(p), norm(rp), norm(rd), norm(rc,Inf),alpha)
    end
end

end

if ~converged
    info = 0;
else
    info = 1;
end



function alpha=maxstep(x,dx)
% calculates the max feasible step alpha such that x + alpha*dx > 0
dxneg = dx < 0;
if any(dxneg)
    alpha = min( -x(dxneg)./dx(dxneg) );
    alpha = min(alpha,1);
else
    alpha = 1;
end
```

```
function [Bp,info] = bfgsupdate(f,B,x,xp)
%        [Bp,info] = bfgsupdate(f,B,x,xp)
% updates the approximate Hessian B when moving from x to xp



s = xp - x;


[obj ,g ] = f(x);
[objp,gp] = f(xp);
y = gp - g;

% ensure y'*s > 0

if y'*s <= 0
    info = 0;
    Bp   = Nan*B;
    return
end

% We want B*y = s, here we use the naive update formula, if we get into
% trouble we will switch to updating Cholesky factors
z  = B*s;
Bp = B - z*z'/(s'*z) + y*y'/(y'*s);
info = 1;
```

```
function [alpha,info] = linesearch(func,x,s,stp,stpmax)
%        [alpha,info] = linesearch(f,x,p,alpha0,alphamax)
% Calls the linesearch routine cvsrch

fcn = @(n,x) func(x);
n   = size(x,1);
[f,g] = func(x);

% here are some default parameter values
stpmin  = 1e-6;
maxfev  = 100;
xtol    = 1e-6;
ftol    = 1e-4;
```

```
gtol    = 9e-1;


[x,f,g,stp,info,nfev] = cvsrch(fcn,n,x,f,g,s,stp,ftol,gtol,xtol,...
                               stpmin,stpmax,maxfev);               %#ok

alpha = stp;
```

```
      function [x,f,g,stp,info,nfev] ...
       = cvsrch(fcn,n,x,f,g,s,stp,ftol,gtol,xtol, ...
                stpmin,stpmax,maxfev)
%   Translation of minpack subroutine cvsrch
%   Dianne O'Leary   July 1991
%     **********
%
%     Subroutine cvsrch
%
%     The purpose of cvsrch is to find a step which satisfies
%     a sufficient decrease condition and a curvature condition.
%     The user must provide a subroutine which calculates the
%     function and the gradient.
%
%     At each stage the subroutine updates an interval of
%     uncertainty with endpoints stx and sty. The interval of
%     uncertainty is initially chosen so that it contains a
%     minimizer of the modified function
%
%          f(x+stp*s) - f(x) - ftol*stp*(gradf(x)'s).
%
%     If a step is obtained for which the modified function
%     has a nonpositive function value and nonnegative derivative,
%     then the interval of uncertainty is chosen so that it
%     contains a minimizer of f(x+stp*s).
%
%     The algorithm is designed to find a step which satisfies
%     the sufficient decrease condition
%
%           f(x+stp*s) <= f(x) + ftol*stp*(gradf(x)'s),
%
%     and the curvature condition
%
%           abs(gradf(x+stp*s)'s)) <= gtol*abs(gradf(x)'s).
%
%     If ftol is less than gtol and if, for example, the function
%     is bounded below, then there is always a step which satisfies
%     both conditions. If no step can be found which satisfies both
%     conditions, then the algorithm usually stops when rounding
%     errors prevent further progress. In this case stp only
%     satisfies the sufficient decrease condition.
%
%     The subroutine statement is
%
%        subroutine cvsrch(fcn,n,x,f,g,s,stp,ftol,gtol,xtol,
%                          stpmin,stpmax,maxfev,info,nfev,wa)
%     where
%
%       fcn is the name of the user-supplied subroutine which
%         calculates the function and the gradient.  fcn must
%         be declared in an external statement in the user
%         calling program, and should be written as follows.
%
%         subroutine fcn(n,x,f,g)
%         integer n
%         f
%         x(n),g(n)
%         ----------
%         Calculate the function at x and
%         return this value in the variable f.
%         Calculate the gradient at x and
%         return this vector in g.
%         ----------
%         return
%         end
%
%       n is a positive integer input variable set to the number
%         of variables.
%
%       x is an array of length n. On input it must contain the
%         base point for the line search. On output it contains
%         x + stp*s.
%
%       f is a variable. On input it must contain the value of f
%         at x. On output it contains the value of f at x + stp*s.
%
%       g is an array of length n. On input it must contain the
%         gradient of f at x. On output it contains the gradient
%         of f at x + stp*s.
%
%       s is an input array of length n which specifies the
%         search direction.
%
%       stp is a nonnegative variable. On input stp contains an
%         initial estimate of a satisfactory step. On output
%         stp contains the final estimate.
%
%       ftol and gtol are nonnegative input variables. Termination
%         occurs when the sufficient decrease condition and the
%         directional derivative condition are satisfied.
%
%       xtol is a nonnegative input variable. Termination occurs
%         when the relative width of the interval of uncertainty
```

```
%          is at most xtol.
%
%        stpmin and stpmax are nonnegative input variables which
%          specify lower and upper bounds for the step.
%
%        maxfev is a positive integer input variable. Termination
%          occurs when the number of calls to fcn is at least
%          maxfev by the end of an iteration.
%
%        info is an integer output variable set as follows:
%
%          info = 0  Improper input parameters.
%
%          info = 1  The sufficient decrease condition and the
%                    directional derivative condition hold.
%
%          info = 2  Relative width of the interval of uncertainty
%                    is at most xtol.
%
%          info = 3  Number of calls to fcn has reached maxfev.
%
%          info = 4  The step is at the lower bound stpmin.
%
%          info = 5  The step is at the upper bound stpmax.
%
%          info = 6  Rounding errors prevent further progress.
%                    There may not be a step which satisfies the
%                    sufficient decrease and curvature conditions.
%                    Tolerances may be too small.
%
%        nfev is an integer output variable set to the number of
%          calls to fcn.
%
%        wa is a work array of length n.
%
%      Subprograms called
%
%        user-supplied......fcn
%
%        MINPACK-supplied...cstep
%
%        FORTRAN-supplied...abs,max,min
%
%      Argonne National Laboratory. MINPACK Project. June 1983
%      Jorge J. More', David J. Thuente
%
%      **********
      p5 = .5;
      p66 = .66;
      xtrapf = 4;
      info = 0;
      infoc = 1;

%
%     Check the input parameters for errors.
%
      if (n <= 0 | stp <= 0.0 | ftol < 0.0 |  ...
          gtol < 0.0 | xtol < 0.0 | stpmin < 0.0  ...
          | stpmax < stpmin | maxfev <= 0)
         info = 0;
         nfev = 0;
         return
      end
%
%     Compute the initial gradient in the search direction
%     and check that s is a descent direction.
%
      dginit = g'*s;
      if (dginit >= 0.0)
          info = 0;
          nfev = 0;
          return
      end
%
%     Initialize local variables.
%
      brackt = 0;
      stage1 = 1;
      nfev = 0;
      finit = f;
      dgtest = ftol*dginit;
      width = stpmax - stpmin;
      width1 = 2*width;
      wa = x;
%
%     The variables stx, fx, dgx contain the values of the step,
%     function, and directional derivative at the best step.
%     The variables sty, fy, dgy contain the value of the step,
%     function, and derivative at the other endpoint of
%     the interval of uncertainty.
%     The variables stp, f, dg contain the values of the step,
%     function, and derivative at the current step.
%
      stx = 0.0;
      fx = finit;
      dgx = dginit;
      sty = 0.0;
      fy = finit;
      dgy = dginit;
%
%     Start of iteration.
%
```

```
   while (1)
%
%        Set the minimum and maximum steps to correspond
%        to the present interval of uncertainty.
%
         if (brackt)
            stmin = min(stx,sty);
            stmax = max(stx,sty);
         else
            stmin = stx;
            stmax = stp + xtrapf*(stp - stx);
         end
%
%        Force the step to be within the bounds stpmax and stpmin.
%
         stp = max(stp,stpmin);
         stp = min(stp,stpmax);
%
%        If an unusual termination is to occur then let
%        stp be the lowest point obtained so far.
%
         if ((brackt & (stp <= stmin | stp >= stmax)) ...
            | nfev >= maxfev-1 | infoc == 0 ...
            | (brackt & stmax-stmin <= xtol*stmax))
            stp = stx;
         end
%
%        Evaluate the function and gradient at stp
%        and compute the directional derivative.
%
         x = wa + stp * s;
         [f,g] = feval(fcn,n,x);
         nfev = nfev + 1;
         dg = g' * s;
         ftest1 = finit + stp*dgtest;
%
%        Test for convergence.
%
         if ((brackt & (stp <= stmin | stp >= stmax)) | infoc == 0)
                  info = 6;
         end
         if (stp == stpmax & f <= ftest1 & dg <= dgtest)
                  info = 5;
         end
         if (stp == stpmin & (f > ftest1 | dg >= dgtest))
                  info = 4;
         end
         if (nfev >= maxfev)
                  info = 3;
         end
         if (brackt & stmax-stmin <= xtol*stmax)
                  info = 2;
         end
         if (f <= ftest1 & abs(dg) <= gtol*(-dginit))
                  info = 1;
         end
%
%        Check for termination.
%
         if (info ~= 0)
                  return
         end
%
%        In the first stage we seek a step for which the modified
%        function has a nonpositive value and nonnegative derivative.
%
         if (stage1 & f <= ftest1 & dg >= min(ftol,gtol)*dginit)
               stage1 = 0;
         end
%
%        A modified function is used to predict the step only if
%        we have not obtained a step for which the modified
%        function has a nonpositive function value and nonnegative
%        derivative, and if a lower function value has been
%        obtained but the decrease is not sufficient.
%
         if (stage1 & f <= fx & f > ftest1)
%
%           Define the modified function and derivative values.
%
            fm = f - stp*dgtest;
            fxm = fx - stx*dgtest;
            fym = fy - sty*dgtest;
            dgm = dg - dgtest;
            dgxm = dgx - dgtest;
            dgym = dgy - dgtest;
%
%           Call cstep to update the interval of uncertainty
%           and to compute the new step.
%
            [stx,fxm,dgxm,sty,fym,dgym,stp,fm,dgm,brackt,infoc] ...
             = cstep(stx,fxm,dgxm,sty,fym,dgym,stp,fm,dgm, ...
                     brackt,stmin,stmax);
%
%           Reset the function and gradient values for f.
%
            fx = fxm + stx*dgtest;
            fy = fym + sty*dgtest;
            dgx = dgxm + dgtest;
            dgy = dgym + dgtest;
         else
%
```

```
%           Call cstep to update the interval of uncertainty
%           and to compute the new step.
%
            [stx,fx,dgx,sty,fy,dgy,stp,f,dg,brackt,infoc] ...
             = cstep(stx,fx,dgx,sty,fy,dgy,stp,f,dg, ...
                     brackt,stmin,stmax);
         end
%
%        Force a sufficient decrease in the size of the
%        interval of uncertainty.
%
         if (brackt)
            if (abs(sty-stx) >= p66*width1)
              stp = stx + p5*(sty - stx);
            end
            width1 = width;
            width = abs(sty-stx);
         end
%
%        End of iteration.
%
      end
%
%     Last card of subroutine cvsrch.
%
```

```
     function  [stx,fx,dx,sty,fy,dy,stp,fp,dp,brackt,info] ...
       = cstep(stx,fx,dx,sty,fy,dy,stp,fp,dp,brackt,stpmin,stpmax)
%    Translation of minpack subroutine cstep
%    Dianne O'Leary   July 1991
%      **********
%
%     Subroutine cstep
%
%     The purpose of cstep is to compute a safeguarded step for
%     a linesearch and to update an interval of uncertainty for
%     a minimizer of the function.
%
%     The parameter stx contains the step with the least function
%     value. The parameter stp contains the current step. It is
%     assumed that the derivative at stx is negative in the
%     direction of the step. If brackt is set true then a
%     minimizer has been bracketed in an interval of uncertainty
%     with endpoints stx and sty.
%
%     The subroutine statement is
%
%       subroutine cstep(stx,fx,dx,sty,fy,dy,stp,fp,dp,brackt,
%                        stpmin,stpmax,info)
%
%     where
%
%       stx, fx, and dx are variables which specify the step,
%         the function, and the derivative at the best step obtained
%         so far. The derivative must be negative in the direction
%         of the step, that is, dx and stp-stx must have opposite
%         signs. On output these parameters are updated appropriately.
%
%       sty, fy, and dy are variables which specify the step,
%         the function, and the derivative at the other endpoint of
%         the interval of uncertainty. On output these parameters are
%         updated appropriately.
%
%       stp, fp, and dp are variables which specify the step,
%         the function, and the derivative at the current step.
%         If brackt is set true then on input stp must be
%         between stx and sty. On output stp is set to the new step.
%
%       brackt is a logical variable which specifies if a minimizer
%         has been bracketed. If the minimizer has not been bracketed
%         then on input brackt must be set false. If the minimizer
%         is bracketed then on output brackt is set true.
%
%       stpmin and stpmax are input variables which specify lower
%         and upper bounds for the step.
%
%       info is an integer output variable set as follows:
%         If info = 1,2,3,4,5, then the step has been computed
%         according to one of the five cases below. Otherwise
%         info = 0, and this indicates improper input parameters.
%
%     Subprograms called
%
%       FORTRAN-supplied ... abs,max,min,sqrt
%                        ... dble
%
%     Argonne National Laboratory. MINPACK Project. June 1983
%     Jorge J. More', David J. Thuente
%
%     **********
      p66 = 0.66;
      info = 0;
%
%     Check the input parameters for errors.
%
      if ((brackt & (stp <= min(stx,sty) | ...
          stp >= max(stx,sty))) | ...
          dx*(stp-stx) >= 0.0 | stpmax < stpmin)
         return
```

```
      end
%
%     Determine if the derivatives have opposite sign.
%
      sgnd = dp*(dx/abs(dx));
%
%     First case. A higher function value.
%     The minimum is bracketed. If the cubic step is closer
%     to stx than the quadratic step, the cubic step is taken,
%     else the average of the cubic and quadratic steps is taken.
%
      if (fp > fx)
         info = 1;
         bound = 1;
         theta = 3*(fx - fp)/(stp - stx) + dx + dp;
         s = norm([theta,dx,dp],inf);
         gamma = s*sqrt((theta/s)^2 - (dx/s)*(dp/s));
         if (stp < stx)
             gamma = -gamma;
         end
         p = (gamma - dx) + theta;
         q = ((gamma - dx) + gamma) + dp;
         r = p/q;
         stpc = stx + r*(stp - stx);
         stpq = stx + ((dx/((fx-fp)/(stp-stx)+dx))/2)*(stp - stx);
         if (abs(stpc-stx) < abs(stpq-stx))
            stpf = stpc;
         else
           stpf = stpc + (stpq - stpc)/2;
         end
         brackt = 1;
%
%     Second case. A lower function value and derivatives of
%     opposite sign. The minimum is bracketed. If the cubic
%     step is closer to stx than the quadratic (secant) step,
%     the cubic step is taken, else the quadratic step is taken.
%
      elseif (sgnd < 0.0)
         info = 2;
         bound = 0;
         theta = 3*(fx - fp)/(stp - stx) + dx + dp;
         s = norm([theta,dx,dp],inf);
         gamma = s*sqrt((theta/s)^2 - (dx/s)*(dp/s));
         if (stp > stx)
            gamma = -gamma;
         end
         p = (gamma - dp) + theta;
         q = ((gamma - dp) + gamma) + dx;
         r = p/q;
         stpc = stp + r*(stx - stp);
         stpq = stp + (dp/(dp-dx))*(stx - stp);
         if (abs(stpc-stp) > abs(stpq-stp))
            stpf = stpc;
         else
            stpf = stpq;
         end
         brackt = 1;
%
%     Third case. A lower function value, derivatives of the
%     same sign, and the magnitude of the derivative decreases.
%     The cubic step is only used if the cubic tends to infinity
%     in the direction of the step or if the minimum of the cubic
%     is beyond stp. Otherwise the cubic step is defined to be
%     either stpmin or stpmax. The quadratic (secant) step is also
%     computed and if the minimum is bracketed then the the step
%     closest to stx is taken, else the step farthest away is taken.
%
      elseif (abs(dp) < abs(dx))
         info = 3;
         bound = 1;
         theta = 3*(fx - fp)/(stp - stx) + dx + dp;
         s = norm([theta,dx,dp],inf);
%
%        The case gamma = 0 only arises if the cubic does not tend
%        to infinity in the direction of the step.
%
         gamma = s*sqrt(max(0.,(theta/s)^2 - (dx/s)*(dp/s)));
         if (stp > stx)
             gamma = -gamma;
         end
         p = (gamma - dp) + theta;
         q = (gamma + (dx - dp)) + gamma;
         r = p/q;
         if (r < 0.0 & gamma ~= 0.0)
            stpc = stp + r*(stx - stp);
         elseif (stp > stx)
            stpc = stpmax;
         else
            stpc = stpmin;
         end
         stpq = stp + (dp/(dp-dx))*(stx - stp);
         if (brackt)
            if (abs(stp-stpc) < abs(stp-stpq))
               stpf = stpc;
            else
               stpf = stpq;
            end
         else
            if (abs(stp-stpc) > abs(stp-stpq))
               stpf = stpc;
            else
               stpf = stpq;
```

```
            end
         end
%
%     Fourth case. A lower function value, derivatives of the
%     same sign, and the magnitude of the derivative does
%     not decrease. If the minimum is not bracketed, the step
%     is either stpmin or stpmax, else the cubic step is taken.
%
      else
         info = 4;
         bound = 0;
         if (brackt)
            theta = 3*(fp - fy)/(sty - stp) + dy + dp;
            s = norm([theta,dy,dp],inf);
            gamma = s*sqrt((theta/s)^2 - (dy/s)*(dp/s));
            if (stp > sty)
                gamma = -gamma;
            end
            p = (gamma - dp) + theta;
            q = ((gamma - dp) + gamma) + dy;
            r = p/q;
            stpc = stp + r*(sty - stp);
            stpf = stpc;
         elseif (stp > stx)
            stpf = stpmax;
         else
            stpf = stpmin;
         end
      end
%
%     Update the interval of uncertainty. This update does not
%     depend on the new step or the case analysis above.
%
      if (fp > fx)
         sty = stp;
         fy = fp;
         dy = dp;
      else
         if (sgnd < 0.0)
            sty = stx;
            fy = fx;
            dy = dx;
         end
         stx = stp;
         fx = fp;
         dx = dp;
      end
%
%     Compute the new step and safeguard it.
%
      stpf = min(stpmax,stpf);
      stpf = max(stpmin,stpf);
      stp = stpf;
      if (brackt & bound)
         if (sty > stx)
            stp = min(stx+p66*(sty-stx),stp);
         else
            stp = max(stx+p66*(sty-stx),stp);
         end
      end
      return
%
%     Last card of subroutine cstep.
%
```

C Verification Code

```
function testqniqp
n = 10;
m = 5;
c = rand(n,1);
H = rand(n,n);
H = (H+H')/2;
[V,D] = eig(H);
d = diag(D);
d = pos(d);
D = diag(d);
H = V*D*V';
A = [eye(n); -eye(n)];
e = ones(n,1);
z = zeros(n,1);
b  = [-e;-e];


% cvx_begin
% variable x(n);
% minimize( c'*x + 1/2*x'*H*x);
% subject to
%          A*x  >= b;
% cvx_end
% fstar = cvx_optval;
% xstar = x;


function [obj,grad] = quad(x,c,H)
```

```
    obj = c'*x + 1/2*x'*H*x;
    if nargout > 1
        grad = c + H*x;
    end
end

x = 1/2*e;

f = @(x) quad(x,c,H);

[xstar,fstar,exitflag]=fmincon(f,x,-A,-b)

[x,y,info]=qniqp(f,A,b,x,1e-6);
obj = f(x)
error = abs(fstar - obj)

end
```

```
n = 10;
m = 20;

g = rand(n,1);

B = rand(n,n);
B = (B + B')/2;
[V,D] = eig(B);
d     = diag(D);
d     = max(d, 1e-1);
B  = V*diag(d)*V';

v = rand(m,1);
A = rand(m,n);

cvx_begin
variable p(n);
dual variable y;
minimize( g'*p + 1/2*p'*B*p);
subject to
        y:   A*p >= - v;
cvx_end

fstar = cvx_optval;
pstar = p;
ystar = y;

[p,y,info] = cqp(g,B,A,v);
```

References

[1] Jorge Moré and David Thuente. Line search algorithms with guaranteed sufficient decrease. `citeseer.ist.psu.edu/more92line.html`, 1994.

[2] Walter Murray. CME 304: Numerical Optimization Notes, 2006.

[3] Dianne O'Leary. CMSC 764 Advanced Numerical Optimization. `http://www.cs.umd.edu/users/oleary/a607/index.html`.

[4] Michael Saunders. CME 338: Large-Scale Numerical Optimization PDCO Primal-Dual Interior Methods, 2006.

[5] Douglas Shafer. The Brachistochrone: Historical Gateway to the Calculus of Variations. `http://mat.uab.es/matmat/PDFv2007/v2007n05.pdf`.

[6] Lloyd N. Trefethen. *Spectral Methods in* MATLAB. SIAM, 2000.