AMS 232 Nonlinear Optimization #1

Harleigh Marsh

May 9, 2015

1. Consider a full-rank matrix $A \in \mathbb{R}^{m\times n}$ where $n \geq m$, let $b \in \mathbb{R}^n$ and $x \in \mathbb{R}^m$. Using necessary conditions, solve the following problem:

$$\underset{x}{\text{minimize}} \quad (Ax-b)^T(Ax-b)$$

Define $f : \mathbb{R}^m \longrightarrow \mathbb{R}$ as $f(x) = (Ax-b)^T(Ax-b)$, then our problem can be written nicely as

$$\begin{aligned} &\underset{x}{\text{minimize}} && f(x) \\ &\text{subject to} && x \in \mathbb{R}^m \end{aligned} \tag{1}$$

As 1 is unconstrained, a first order necessary condition is given as if x is a local minimum then $\dfrac{\partial f}{\partial x} = 0$. Therefore we need but compute the gradient of f and set it to zero.

$$\begin{aligned} 0 &= \frac{\partial f}{\partial x} \\ &= \frac{\partial}{\partial x}\big((Ax-b)^T(Ax-b)\big) \\ &= \frac{\partial}{\partial x}\big(x^TA^TAx - x^TA^Tb - b^TAx + b^Tb\big) \\ &= 2x^TA^TA - b^TA - b^TA + 0 \\ &= 2x^TA^TA - 2b^TA \end{aligned}$$

Therefore, since the inverse and transpose operators commute, we have that $x = (A^TA)^{-1}A^Tb$, which is our necessary condition.

2. Apply necessary conditions on the following optimization problem where $A \in \mathbb{R}^{n\times m}$ is full-rank with $n < m$, and $b \in \mathbb{R}^n$ and $x \in \mathbb{R}^m$

$$\begin{aligned} &\underset{x}{\text{minimize}} && x^Tx \\ &\text{subject to} && Ax = b \end{aligned} \tag{2}$$

As we have an equality constraint $Ax - b = 0$ we apply the necessary condition of Lagrange, which requires that at an optimal point $\dfrac{\partial \mathcal{L}}{\partial x}$ and $\dfrac{\partial \mathcal{L}}{\partial \lambda}$. Building the Lagrangian, we have

$$\begin{aligned} \mathcal{L} &= x^Tx - \lambda^T(Ax-b) && \text{(where } \lambda \in \mathbb{R}^n\text{)} \\ &= x^Tx - \lambda^TAx + \lambda^Tb \end{aligned}$$

Therefore

$$\begin{aligned} 0 &= \frac{\partial \mathcal{L}}{\partial x} = 2x^T - \lambda^TA && \rightarrow && x = \frac{1}{2}A^T\lambda && (\alpha) \\ 0 &= \frac{\partial \mathcal{L}}{\partial \lambda} = Ax - b && \rightarrow && Ax = b && (\beta) \end{aligned}$$

Now with (α) into (β) we solve for λ as $\lambda = 2(AA^T)^{-1}b$, and so putting this back into (α) we get the necessary condition

$$x = A^T(AA^T)^{-1}b$$

3. Here we answer the following out of Nocedal and Wright's Numerical Optimization text:

> **12.18** Consider the problem of finding the point on the parabola $y = \frac{1}{5}(x-1)^2$ that is closest to $(x, y) = (1, 2)$, in the Euclidean norm sense. We can formulate this problem as
>
> $$\min\ f(x, y) = (x-1)^2 + (y-2)^2 \quad \text{subject to } (x-1)^2 = 5y.$$
>
> **(a)** Find all the KKT points for this problem. Is the LICQ satisfied?
>
> **(b)** Which of these points are solutions?
>
> **(c)** By directly substituting the constraint into the objective function and eliminating the variable x, we obtain an unconstrained optimization problem. Show that the solutions of this problem cannot be solutions of the original problem.

With a slight change of notation of $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, we have

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \triangleq (x_1-1)^2 + (x_2-2)^2 \\ \text{subject to} \quad & (x_1-1)^2 - 5x_2 = 0 \end{aligned} \tag{3}$$

(a): In 3 we have only one equality constraint, we have, with $\lambda \in \mathbb{R}$ that the Lagrange function is $\mathcal{L}(x, \lambda) = (x_1-1)^2 + (x_2-2)^2 - \lambda((x_1-1)^2 - 5x_2)$. Applying the KKT conditions gives

$$\frac{\partial \mathcal{L}}{\partial x} = 0$$
$$(x_1-1)^2 - 5x_2 = 0$$

Therefore we have three equations

$$\begin{bmatrix} 2(x_1-1) - 2\lambda(x_1-1) \\ 2(x_2-2) + 5\lambda \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$
$$(x_1-1)^2 - 5x_2 = 0$$

Solving this system, we get only one real solution, $x^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Lastly, we show that LICQ holds at x^*. Computing the gradient of the (only) constraint at x^* gives

$$\frac{\partial}{\partial x}((x_1-1)^2 - 5x_2)\Big|_{x=x^*} = \begin{bmatrix} 2(x_1-1) \\ -5 \end{bmatrix}_{|x=x^*} = \begin{bmatrix} 0 \\ -5 \end{bmatrix}$$

which is clearly linearly independent. Therefore LICQ holds at x^*.

(b): To show that the point we found in (a), $x^* = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, we apply the second order sufficient conditions for KKT (given as theorem 12.6 in the text). With x^* we get $\lambda^* = \frac{4}{5}$. Now, computing the Hessian of $\mathcal{L}$ and evaluating it at $x^*, \lambda*$, we see that it is positive definite:

$$\frac{\partial \mathcal{L}^2}{\partial x^2}_{|x=x^*, \lambda=\lambda^*} = \begin{bmatrix} 2(1-\lambda) & 0 \\ 0 & 2 \end{bmatrix}_{|x=x^*, \lambda=\lambda^*} = \begin{bmatrix} \frac{2}{5} & 0 \\ 0 & 2 \end{bmatrix}$$

Therefore we have by the second order sufficient conditions, that x^* is a strict local minimum.

(c): Substituting the constraint into the objective function gives the unconstrained problem

$$\underset{y \in \mathbb{R}}{\text{minimize}} \quad y^2 + y + 4 \tag{4}$$

Therefore a necessary condition is that the gradient of the objective function must be zero, giving us $2y + 1 = 0$, so all optimal points of this new optimization problem must have that the y-component is $-\frac{1}{2}$, which is clearly different from the results derived in (a). Hence we see that by substituting the constraint into the objective function, we ended up with a different problem; that is, we received a different necessary condition.

4. See comments in the Snopt code to solve the modified Queen Dido's problem. Below is Dido's optimal curve returned from the code attached in the appendix to this paper.

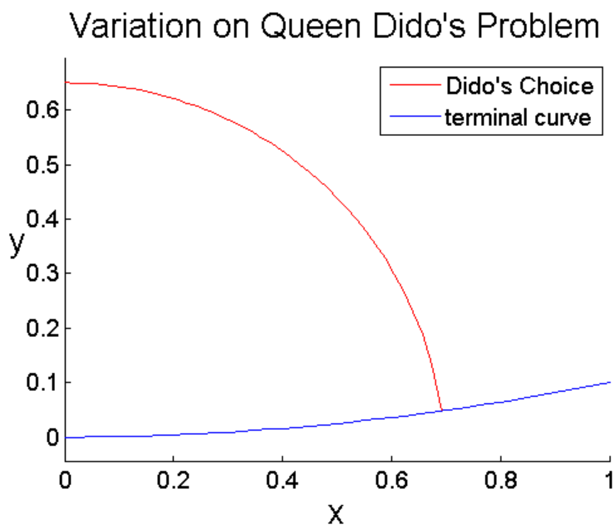

Figure 1: The red curve, of length one, is of the shape to maximize the enclosed region (formed by the red and blue curve).

Matlab Code

queenDidoTerminalCurve.m

```
%{
    What we are doing:  Given a Terminal Curve and a bit of 'string' of length L, we wish
    to optimize the region enclosed by out bit of string.  The string can not stretch:
    It's length L and that's that!


    Choose N uniform intervals (hence N+1 points) and discritize the x-axis into uniform
    {x(1),...,x(N+1)}, then we have y(i) as the point on the curve-to-be-optimized at the
    point x(i).

    DecVars: y(1)...y(N+1), x(N+1)  So N+2 Total
    Constraints:
       * y(i)>=0  i = 1,...,N+1
       * 0<=X(N+1)<=L  That is, the last x-point is upper bounded by the curve length
         this is because the maximum length of the curve y(x) forme by the y(i)'s is L,
         hence the final x-point (relating to the final y point) has a maximim length if L

%}
clear all; close all;

%{
    Global variables nessisary to persist variables into the UserFun that Snopt
    uses to converge to the optimal solution.
        N: Number of (uniform) intervals on x-axis.
        f: Function that describes the terminal curve to the modified DIDO problem.
%}
global N f;

%pointer to the snopt userfun; called in the cmex to snopt.
usrfun = 'queenDidoCurveUserFun';


f = @(t) 0.1*t.^2;    %Terminal curve: The last point of the yCurve must lie on this curve
curveLength = 1;      %Exact length of the yCurve
%Arbirtrary upper bound for the area between the decsion curve and the terminal curve
enclosedRegionUpperBound = 300;

%Number of intervals from [x0, xf] where xf is the last node on the x-axis being x(N+1)
N = 40;

%{
    Given a positive integer N, the decesion variables are N+1 points on the curve to be
    made (decesion curve) y(1),...,y(N+1), as well as the last node on the x-axis: x(N+1)
    Hence there are N+2 decesion variables.
%}

%Build an Initial Guess:  For snopt, the initial guess needs to be as a column vector.
%Our initial guess is just a staight line, of length equal to the curveLength, and whose
%last point lies on the terminal curve
xf = curveLength;             %The largest the last value of x can be is the curveLength
yGuess = f(xf)*ones(N+1,1);  %A straight line starting on the y axis ending on the terminal curve
xInit = [ yGuess; xf ];      %initial guess for the Decision Variables


numDecVar = length(xInit);


%{
   Set the lower and upper bounds on the decision variables. SNOPT wants xLow
   and xUpp as column vectors.  Note that the order in which xInit was pack
   determines the ordering of the decision variables.

    All of the decesion variables must lie in the range of 0<=x<=Upp where Upp is some
    arbitrairly large value.  Although, because the length of the decesion curve is always
    fixed (i.e.: it's always 1) the
   upper bound on the final node on the x-axis
%}
xLow = 0*ones(numDecVar,1);
xUpp =  100*ones(numDecVar,1);
xUpp(end) = curveLength;

%{
   Set the lower and upper bounds for the cost function and the constraints
   Note: the order of Flow and Fupp is determined by the packing of the column
   vector F, which is done in the userFun.  F is packed as:
        1) cost function  --inequality constraint: size 1
        2) The length of the decesion curve remains a fixed constant -- size 1
        3) An endpoint constraint: That the last point of the decesion curve must lie
           on the terminal curve
%}
Flow = [ 0;
         curveLength;
         0];    %endponint condition: final y value must lie on the curve f(x)

%upper bounds on the constraints
Fupp = [ enclosedRegionUpperBound;
         curveLength;
         0];    %endpoint condition: final y value must lie on the curve f(x)

%This little 4-line block of setting variables for Snopt are not we do not use;
%we set them to the defaults; ObjRow for example defines which row the
```

```
%objective function is located within the constraints col-vec ('F' in the
%userFun)
numConstraints  = length(Flow);
xMul   = zeros(numDecVar,1); xState = zeros(numDecVar,1);
Fmul   = zeros(numConstraints,1); Fstate = zeros(numConstraints,1);
ObjAdd = 0; ObjRow = 1;

%Setting the linear and nonlinear sparsity patterns--this version does not
%supply any of the sparsity patterns; it tells Snopt that every entry of the
%Jacobian needs to be calculated
A  = [];  iAfun = [];  jAvar = [];
[iGfun,jGvar] = find(ones(numConstraints, numDecVar));

%Set the Optimal Parameters for SNOPT. See chapter 7, pg 62 'Optimal Parameters'
%Note we first set 'Defaults' to start SNOPT on a clean-slate; very important!
snset('Defaults');              %You NEED this to flush snopt clean before a run!
snseti('Derivative option', 0); %Telling snopt we know nothing about the jacobian
snseti('Verify level', 3);      %Slows performance but very useful
snset('Maximize');


%Sumary and Solution files; see chapter 8 of SNOPT guide (section 8.8, 8.9)
snprint('result.txt');
snsummary('result.sum');

%Call snopt
solveopt = 1;       %Still have no idea what this flag tells snopt (other than 'optimize')
tic                 %We are going to time snopt
[xOpt,F,xMul,Fmul,INFO] = snoptcmex( solveopt, ...
                     xInit,xLow,xUpp,xMul,xState, ...
                     Flow,Fupp,Fmul,Fstate,        ...
                     ObjAdd,ObjRow,A,iAfun,jAvar,  ...
                     iGfun,jGvar,usrfun );
runTime=toc;        %we are timing snopt
%%%%%%%%%%%%%%%%%%%% Plot Code %%%%%%%%%%%%%%%%%%%%

snsummary off; % close the summary .sum file; empties the print buffer
snprint   off; % Closes the print-file .out; empties the print buffer

%Unpack the optimal solution from the column vector Snopt returned
yOpt = xOpt(1:end-1);
xLast = xOpt(end);

%Unpack the initial guess curve, so we can draw it next to the optimal curve
yGuess = xInit(1:end-1);
xGuessDom = linspace(0,curveLength, N+1);   %This is the x-domain used for the guess

%Grab our uniform grid based on the final x-point.  Note that the problem was solved using
%a uniform grid; we just stored the last point durring the computations within the
%userFun.
xGrid = linspace(0, xLast, N+1);

%Plotting the terminal curve along with the optimal curve
hold on;
    plot(xGrid, yOpt, '-r');       %Optimal Curve
    plot(xGuessDom, f(xGuessDom)); %Terminal Curve (last pt on optCurve must lie on this)
%     plot(xGuessDom, yGuess, '-m'); %Initial Guess
    title('Variation on Queen Dido''s Problem', 'FontSize', 20);
    xlabel('x', 'FontSize', 20);
    ylabel('y', 'Rotation', 0, 'FontSize', 20);
    legend('Dido''s Choice', 'terminal curve');
    axis equal;
hold off;

%Just some nicely formated output to the console on the Snopt Run
disp(strcat('Execution time: ', num2str(runTime)));
disp(strcat('SNOPT exited with INFO==', num2str(INFO)));    %see pg 19 snopt manual for INFO

%
% end queenDidoCurve.m
%
```

queenDidoCurveUserFun.m

```
%{
    The UserFun for the Brach Problem. This function is called by SNOPT to
    converge to an optimal solution.  In this function
        1) We pack the nonlinear constraints into F.
        2) We calculate the Jacobian of the constraints with respect to the
           decision variables (variable G).
    In this function, we discritize the dynamics of the Brach problem which were
    set as equality constraints (using Flow and Fupp in the script that calls
    this userFun).


    Inputs:
        decVars: Column vector whose order is determined on how xInit was
                 packed.

    Outputs:
        F: The column vector containing the nonlinear constraints
        G: The Jacobian of the constraints with respect to the decision
           variables; G is a column vector
```

```
%}
function [ F, G ] = queenDidoCurveUserFun( decVars )

    global N f;

    %Unpack 'decVars' into meangful variables
    y = decVars(1:end-1);   %The N+1 values that represent points on the decesion curve
    xLast = decVars(end);   %The last value on the x-axis

    %uniform step-size along the x-axis
    deltaX = xLast/N;

    %Calculation of the arcLength of the yCurve:
    %   Sum(i=1 to i=N) of  sqrt( (y(i+1)-y(i))^2 + (x(i+1)-x(i))^2 )
    curveLength = sum(  ( (y(2:end)-y(1:end-1)).^2 + deltaX^2).^(1/2)  );


    %Calculation of the area made by the yCurve (using trapazoidal approximation)
    topCurve = (deltaX/2)*sum( y(2:end)+y(1:end-1) );
    %Calculation of the area made by the terminal curve; note this compuation can not be
    %moved off-line due to the last x-value always changing.  The terminal curve always
    %starts at x=0 since the first y value lies on the y-axis
    bottomCurve = integral(f, 0, xLast);
    areaEnclosed = topCurve - bottomCurve;  %Remember we are maximizing


    %The way F is packed here determines the order of Fupp Flow and the order of
    %the Jacobian of the constraints wrt the decision variables.  F stores the
    %objective function as well as the constraints.
    F = [ areaEnclosed;         %Objective function to maximize is the enclosed area
          curveLength;          %The curve length must be a fixex constant (set in main file)
          y(end) - f(xLast)];   %the last point of the yCurve must lie on the termCurve


     %The Jacobian of the constraints with respect to the decision variables; in
     %this example, we tell Snopt to do it for us.
     G=[];
end
```