Class 28 - **CONSTRAINED VS. UNCONSTRAINED NONLINEAR PROGRAMMING**

**Lagrangean constrained optimization**

**Figure 1** - Roping off an area

\[
\text{max } f(b, c) = A^2 = s (s - a) (s - b) (s - c) \\
\text{s.t. } g_1(b, c) = a + b + c - 2s = 0
\]

(Notice \(f(b, c)\) is a concave function and our constraint is in the form \(g_i(x) = 0\), rather than \(g_i(x) = b_i\).)

\[
\text{max } L(b, c, \lambda_1) = f(b, c) - \lambda_1 g_1(b, c) \\
\quad = s (s - a) (s - b) (s - c) - \lambda_1 (a + b + c - 2s)
\]

Instead of \(-\lambda_1\), it will be \(+\lambda_1\) for minimizing a convex function.

\[
L'(b) = 0 \text{ yields } -s (s - a) (s - c) - \lambda_1 = 0 \\
L'(c) = 0 \text{ yields } -s (s - a) (s - b) - \lambda_1 = 0 \\
L'(_\lambda_1) = 0 \text{ yields } a + b + c - 2s = 0.
\]

**Figure 2** - Roping off an optimal area

Solving for unknowns \(b, c, \lambda_1\),
\[
\begin{align*}
  b &= s - a/2 \\
  c &= s - a/2 \\
  \lambda_1 &= -a s (s - a) / 2.
\end{align*}
\]

**EXAMPLE:**

Suppose a rope is 100 feet long, \(s = 50\), \(a = 25\), \(b = c = 50 - 25/2 = 37.5\), and \(\lambda_1 = - (37.5) (50) (50 - 25)/2 = -23,437.5\)

Remember \(\partial f / \partial s = \partial A^2 / \partial s\), which explains the large value for \(\lambda_1\). Thus a movement of the pole to the left or the right of the existing position by one foot of
rope length will decrease the enclosed area by \((23,437.5)^{\frac{1}{2}} = 153.09\) sq. ft. / ft. 
Finally, \(A^2 = 50 (50 - 25) (50 - 37.5) (50 - 37.5) = 1.9531 \times 10^5\), or \(A^* = 441.94\) sq. ft.

\(\lambda_1\) is the dual variable, showing the effect of changing the RHS (in this case 0) by \(+\Delta\) or \(-\Delta\). If \(-\Delta\), the rope is lengthened, and the area of the triangle will increase.

On the other hand, if \(+\Delta\), the rope is shortened, and the area will decrease instead. The amount of increase or decrease is \(\lambda_1 \Delta\). Similar interpretation can be made for the distance between the trees \(a\).

**Method of steepest ascent**

\[
\text{max/min } f (x) \\
\mathbf{x} \in \mathbb{R}^n
\]
(i.e., \(\mathbf{x}\) is in an \(n\)-dimensional Euclidean space.)

**Figure 3** - Example search

**General algorithm**

**Step 1.** Select a starting point \(\mathbf{x}^k = \mathbf{x}^0 = (x_1^0, x_2^0, \ldots, x_n^0)^T\) and set \(k = 0\).

**Step 2.** Find a direction to move \(d^k = \nabla f (\mathbf{x}^k)\), which will improve (increase/decrease) the function at iteration \(k\), where \(d^k = (d_1^k, d_2^k, \ldots, d_n^k)^T\).

**Step 3.** Move a distance \(t^k\) in the direction \(d^k\) to a new point \(\mathbf{x}^{k+1} = \mathbf{x}^k + t^k \mathbf{d}^k\), where \(t^k\) is the nonnegative step size at iteration \(k\), to be determined by

a) line search (golden section for example), or

b) analytic technique (parametric in \(t\)).

**Step 4.** Check for local optimality, e.g.,

\[
\frac{\partial f}{\partial x_j} \bigg|_{\mathbf{x} = \mathbf{x}^k} < \epsilon \quad j = 1, 2, \ldots, n. \tag{1}
\]
If stopping criteria are not met, \( k \leq k + 1 \), go to step 2.

**Example**

\[
\text{max } f(x) = 2x_1 x_2 + 2x_2 - x_1^2 - 2x_2^2
\]

\[
d_1 = f'(x_1) = 2x_2 - 2x_1
\]

\[
d_2 = f'(x_2) = 2x_1 + 2 - 4x_2
\]

\[
d^0 = (d_1^0, d_2^0) = \nabla f(x^0) = \nabla f(0, 0) = \left[ f_{x_1}(x_1 = 0, x_2 = 0), f_{x_2}(x_1 = 0, x_2 = 0) \right] = (0, 2).
\]

For \( k = 0 \), set \( x_1^1 = 0 + t(0) = 0, \ x_2^1 = 0 + t(2) = 2t \).

\[
f(x^1) = f[x^0 + t \nabla f(x^0)] = f(0, 2t) = 2(0)(2t) + 2(2t) - (0)^2 - 2(2t)^2 = 4t - 8t^2
\]

\[
t^* = 1/4
\]

\[
x^1 = (0, 0) + 1/4 (0, 2) = (0, \frac{1}{2}).
\]

Since \( d_1 = (2)(\frac{1}{2}) - (2)(0) = 1 \), it is clear that more iterations are necessary.

**Note on “Constrained Optimization:”**

Referring to the “Example search” Figure, should the constraint \( x_1 = x_2 \) be imposed, the global optimum would have been on \( x^* \) instead of \( x^{**} \).

**Another Bonus Pre-test Review (8% Points):**

Write and fully debug the attached C program to implement the “Method of steepest ascent” algorithm. Demonstrate its is working by replicating the problem worked out in Section 12.5 of H&L (2010).

**Due date:** Tuesday 5/10/11, 12:00 noon.
EXERCISE
For roping off an area, repeat the above calculation for a rope of 200 ft.
Roping off an area

Tree 1

Movable pole

Tree 2

a

b

c
Roping off an optimal area

Movable pole

Tree 1

Tree 2

\[
b \quad c
\]

\[
a
\]
Example search
/* Steepest descent */

#include <stdio.h>
#include <math.h>

int COUNT = 0;

float F(float, float);
float GradX (float, float);
float GradY(float, float);

int main()
{
    float a, best_x;
    float best_y, best_f;
    float x;
    float y;
    float d_x;
    float d_y;
    float delta;
    float eps;
    float z;

    int i;
    int j;
    int k;

    //start at 3,2
    best_x = 3.0;
    best_y = 2.0;
    best_f = F(best_x, best_y);

    //step size of 5
    delta = 5.0;

    //our acceptance step (acceptance - step < eps)
    eps = 0.01;
    k = 0;

    //we reduce delta until we reach a value less then our epsilon
    while (delta > eps)
    {
        k++;
        printf ("n---------------------\n");
        printf (" STEP ITERATION %d\n", k);
printf("-------------------\n");

//compute gradient
d_x = GradX(best_x, best_y);
d_y = GradY(best_x, best_y);

printf("Delta is: \nGradient is (%f,%f) from (%f,%f) opt is %f\n", 
delta, d_x, d_y, best_x, best_y, best_f);
a = (delta*delta)/(d_x*d_x + d_y*d_y);
a = (float)sqrt((double)a);
printf("SCALE: %f*2/(%f*2 + %f*2) = %f\n", delta,d_x,d_y,a);

x = best_x - d_x*a;
y = best_y - d_y*a;
z = F(x, y);

printf("mew point (%f%f) has f-%f

", x, y, z);
if (z > best_f)
{
    delta = delta/2.0;
    printf("Delta reduced to %f ...\n", delta);
}
else
{
    best_f = z;
    best_x = x;
    best_y = y;
    printf("Better %f at (%f,%f)\n", z, x, y);
}

printf("Optimal value is %f at (%f,%f)\n", best_f, best_x, best_y);
printf("Function evaluations %d\n", COUNT);

float GradX(float x, float y)
{
    float a, b;

    a = 0.002;
b = F(x+0.001, y) - F(x-0.001,y);
printf("Grad x at (%f,%f) is %f approx is %f\n",x,y,2*x+1, b/a);
    return b/a;
}
float GradY( float x, float y)
{
    float a, b;
    a = 0.002;
    b = F(x, y+0.001) - F(x, y-0.001);
    printf( "Grad y at (%f,%f) is %f approx is %f\n", x, y, 2*y, b/a);
    return b/a;
}

float F(float x, float y)
{
    COUNT++;
    return x*x + y*y + x + 1.0;
}