Scott Greenwald
MAS864.11
Constrained Optimization
May 9, 2011

1 Closest point on a line

Given a point (x_0, y_0) and a line $y = ax + b$, we wish to find the closest point to (x_0, y_0) which lies on the line. This problem may be cast as a constrained optimization problem – minimize $d^2 = (x - x_0)^2 + (y - y_0)^2$ subject to the constraint $y - ax - b = 0$.

The method of Lagrange multipliers uses the observation that at a critical point along a constraint curve, the gradient of the constraint is proportional to the gradient of the object function – that is, these two vectors are aligned. In this case, the method gives rise to the following three equations

$$\begin{aligned} 2x + a\lambda &= 2x_0 \\ 2y - \lambda &= 2y_0 \\ y - ax &= b \end{aligned}$$

Solving these by adding the first two to cancel out λ, we get

$$\begin{aligned} x &\leftarrow \frac{x_0 + ay_0 - ab}{a^2 + 1} \\ y &\leftarrow ax + b \end{aligned}$$

2 Linear programming sort

The idea is to formulate sorting as a constrained optimization by optimizing matrix entries which correspond to position assignments in a sorted vector.

Maximize {1,2,...}. P . {x1,x2,...}
Subject to $\sum_{\text{rows}} P = \sum_{\text{columns}} P = 1$

This is how I'm constructing the objective:

```
{1, 2}.{{p1, p2}, {p3, p4}}.{x1, x2}
```

```
(p1 + 2 p3) x1 + (p2 + 2 p4) x2
```

```
randT[n_, rng_: {-2^16, 2^16}] := Table[Random[Integer, rng], {n}];
```

Here's the program in action:

```
n = 15;
AbsoluteTiming[lpSort[randT[n]]]
```

```
{1.888324, {-64936, -48801, -39631, -36820, -34255, -31101,
  -28411, 11272, 19689, 25544, 27624, 32710, 46667, 48790, 57436}}
```

Not very fast! But the problem is likely the lazy use of symbolics and dense representation in constructing the constraint matrices.

And here's the source:

```
(*aux*)Clear[xName, pName]
pName[i_, j_] := ToExpression[StringJoin["p", ToString[i], ToString[j]]];
xName[i_] := ToExpression[StringJoin["x", ToString[i]]];
Clear[pObjective, pConstraints]
pObjective[mx_, xs_] := Range[Length[xs]].mx.xs;
pConstraints[mx_] := Module[{rowC, colC, n}, n = Length[mx];
    rowC = Flatten /@ (mx.Table[1, {n}]);
    colC = Flatten /@ (Transpose[mx].Table[1, {n}]);
    Join[rowC, colC, -rowC, -colC]];
Clear[progM, progB, progC]
progM[pCon_, mx_] := Outer[Coefficient, pCon, Flatten[mx]];
progB[pCon_, mx_] := Join[Table[1, {Length[pCon] / 2}], Table[-1, {Length[pCon] / 2}]];
progC[pObj_, mx_] := First[progM[{pObj}, mx]];

(*main*)
lpSort[xIn_, givePerm_: False] :=
 Module[{n, mx, xs, pObj, pCon, m, b, c, lpP, sorted}, n = Length[xIn];
  mx = Table[pName[i, j], {i, n}, {j, n}];
  xs = Table[xName[i], {i, n}];
  pObj = pObjective[mx, xs];
  pCon = pConstraints[mx];
  m = progM[pCon, mx];
  b = progB[pCon, mx];
  c = progC[pObj, mx];
  c = (c /. Thread[xs → xIn]);
  lpP = Partition[LinearProgramming[-c, -m, -b], Length[xs]];
  sorted = lpP.xIn;
  If[givePerm, {sorted, lpP}, sorted]]
```