Abraham Hill
October 5th, 2020

Lab 4

Problem 1: Handling Cost

(i) The total handling cost is the unit handling cost times the sum of the number of units moved into and out of the inventory, or roughly: `hand_cost * (into + out_of)`

(ii) Done ☺

Problem 2: Inventory Constraints

(i), (ii) ☺

Problem 3: Conservation of Ice Cream

(i) The first quarter has to be handled separately because there is not a quarter before q1, so calling something like `prev('q1', Quarters)` in AMPL would try to grab something that isn't there. That's why we use `current_inv(p,f)` instead for the first quarter.

(ii) ☺

AMPL output:

```
MINOS 5.51: optimal solution found.
196 iterations, objective 28790635.65
```

```
set Quarters ordered;
    # AMPL is being told to remember the order in which the quarters are listed in the data file.
    # For an ordered set, we can use the functions listed in Table A.5 (page 464) of the AMPL book.
    # – we use “first”, “last” and “prev”.

set Future_Quarters := {q in Quarters: ord(q) > 1};
    # This creates the set of quarters following the first quarter.
    # The first quarter is handled slightly differently,
    #   because the amount in inventory at the beginning of the first quarter
    #   is a known quantity, whereas for future quarters it is a decision variable.

set Flavors;
set Regions;

param demand {Flavors, Regions, Quarters};
    # Demand now depends on the quarter.

set Plants;
set Machines {Plants};

set All_Machines := union {f in Plants} Machines[f];
    # Makes the set All_Machines be the union of the sets Machines[f]

param prod_cost {All_Machines, Flavors};
param days_reqd {All_Machines, Flavors};

param days_avail {All_Machines, Quarters};
    # Machine availibility now depends on the quarter.

param ship_cost {Plants, Regions};

param inv_cap {Plants, Quarters};
param hand_cost;

param current_inv {Plants, Flavors};
    # This is the amount in inventory at the beginning of the planning horizon.


var prod {All_Machines, Flavors, Quarters} >=0;  # amount produced
var ship {Flavors, Plants, Regions, Quarters} >=0;  # amount shipped
var inv {Plants, Flavors, Quarters} >=0;  # amount in inventory at end of quarter
var into {Plants, Flavors, Quarters} >=0; # amount put into inventory
```

```
during the quarter
var out_of {Plants, Flavors, Quarters} >=0; # amount removed from
inventory during the quarter

minimize total_cost:
    sum {m in All_Machines, f in Flavors, q in Quarters}
prod_cost[m,f]*prod[m,f,q]
    + sum {f in Flavors, p in Plants, r in Regions, q in Quarters}
ship_cost[p,r]*ship[f,p,r,q]
    + sum {p in Plants, f in Flavors, q in Quarters}
(into[p,f,q]+out_of[p,f,q])*hand_cost;
# now includes cost of moving units into and out of inventory

subject to  machine_capacity {m in All_Machines, q in Quarters}:
    sum {f in Flavors} days_reqd[m,f]*prod[m,f,q] <= days_avail[m,q];

subject to  satisfy_demand {f in Flavors, r in Regions, q in Quarters}:
    sum {p in Plants} ship[f,p,r,q] = demand[f,r,q];

subject to  determine_amount_handled_in_first_quarter {p in Plants, f in
Flavors}:
    into[p,f,first(Quarters)] - out_of[p,f,first(Quarters)]
        = inv[p,f,first(Quarters)] - current_inv[p,f];
# net units moved into the inventory is the difference between inventory
at the beginning an end of each quarter

subject to  determine_inventory_at_end_of_first_quarter {p in Plants, f in
Flavors}:
    inv[p,f,first(Quarters)]
    = sum {m in Machines[p]} prod[m,f,first(Quarters)] - sum {r in
Regions} ship[f,p,r,first(Quarters)] + current_inv[p,f];
# inventory at the end of the quarter is equal to
# inventory at the beginning, plus units produced at that plant, minus
units shipped away

subject to  determine_amount_handled_in_future_quarters {p in Plants, f in
Flavors, q in Future_Quarters}:
    into[p,f,q] - out_of[p,f,q]
        = inv[p,f,q] - inv[p,f,prev(q, Quarters)];
# net units moved into the inventory is the difference between inventory
at the beginning an end of each quarter

subject to  determine_inventory_at_end_of_future_quarters {p in Plants, f
in Flavors, q in Future_Quarters}:
    inv[p,f,q] = sum {m in Machines[p]} prod[m,f,q] - sum {r in Regions}
ship[f,p,r,q] + inv[p,f,prev(q, Quarters)];
# inventory at the end of the quarter is equal to
# inventory at the beginning, plus units produced at that plant, minus
```

```
76… units shipped away
77
78
79 subject to  do_not_exceed_inventory_capacity {p in Plants, q in Quarters}:
 … sum {f in Flavors} inv[p,f,q] <= inv_cap[p, q];
80
81
82 subject to  end_with_correct_amount_of_inventory {p in Plants, f in
 … Flavors}: inv[p,f,last(Quarters)] = current_inv[p,f];
83
```