



ORF 307: Lecture 1

Linear Programming Chapter 1 Introduction

Robert Vanderbei

February 5, 2019

Slides last edited on February 1, 2019

Course Info

Prereqs: Three semesters of Calculus

Co-reqs: Linear Algebra (MAT 202 or MAT 204)

Textbook: *Linear Programming: Foundations and Extensions, 4th Edition*

Grading:

Homework:	25%
Midterm 1:	25%
Midterm 2:	25%
Final Project:	25%

Homework:

- Due every week at 9pm on Friday.
- All homework must be submitted via Blackboard.
- The lowest homework grade will be dropped.

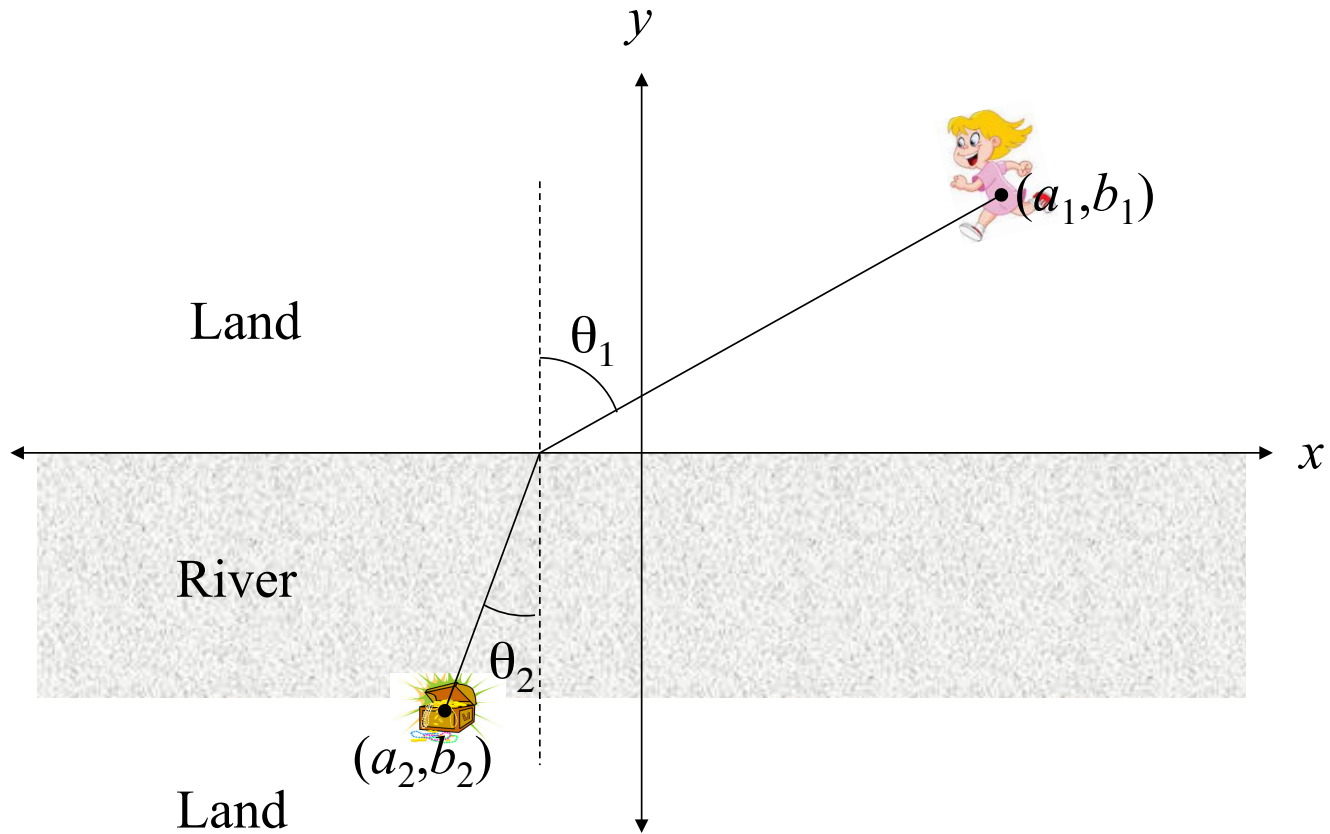
Midterms: In-class on Thursday of the 6th and 11th weeks (Mar 14 and Apr 25).

Lectures: Reading assignments will be posted in advance of each lecture. Please read the material before lecture.

Slides: Will be posted online. But, they are not a replacement for the lecture. They are just my notes to remind me what to say. You must go to lecture to hear what I have to say.

Webpage: <https://vanderbei.princeton.edu/307/lectures.html>

An Example: River Crossing



Solving it on the Computer (using AMPL)

```
# Getting to the treasure fast!
```

```
param a1 := 60;  
param b1 := 40;  
param a2 := -40;  
param b2 := -50;  
param v1 := 10;  
param v2 := 1.5;
```

```
var x;
```

```
minimize time: sqrt((a1-x)^2 + b1^2)/v1 + sqrt((x-a2)^2 + b2^2)/v2;
```

```
solve;
```

```
display x;
```

Solving it on the Computer (using CVXPY)

```
from cvxpy import *

a1 = 60.
b1 = 40.
a2 = -40.
b2 = -50.
v1 = 10.
v2 = 1.5

x = Variable(1)

# simple and elegant but it doesn't work...
#time = Minimize(sqrt((a1-x)**2+b1**2)/v1 + sqrt((x-a2)**2+b2**2)/v2)

# ugly but it works...
time = Minimize(norm(hstack([a1-x, b1]), 2)/v1 \
                 + norm(hstack([x-a2, b2]), 2)/v2 )

prob = Problem(time)
result = prob.solve()
print(x.value)
```

- AMPL is available for free at <http://www.ampl.com/BOOK/download.html>.

I've also made it available here <https://vanderbei.princeton.edu/307/textbook/AMPLbook.pdf>.

These links are also on the course webpage:

<https://vanderbei.princeton.edu/307/lectures.html>.

- Some online tutorials:
 - Google “AMPL tutorial” for examples.

There are three ways to access AMPL:

- Online:** The *Network Enabled Optimization Server (NEOS)*.
- Download Student Version:** Download from AMPL website (<http://ampl.com/try-ampl/download-a-demo-version/>) to your own computer. Never expires. Limited number of variables/constraints (500×500).
- Download Course Version:** Download from *course-specific link* to your own computer. Expires at the end of the semester. Unlimited number of variables/constraints. *Preferred method.*

Details about these three methods are available here:

<http://ampl.com/products/ampl/ampl-for-students/>

Network Enabled Optimization Server (NEOS)

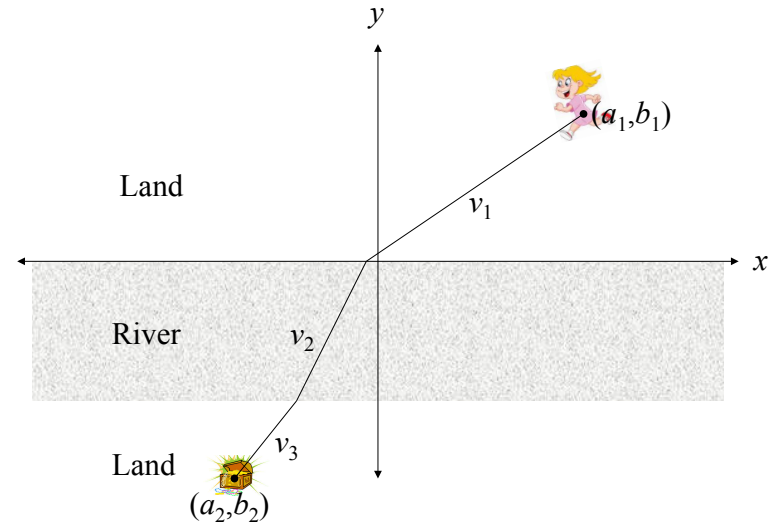
To submit an AMPL model to NEOS...

- visit: <http://www.neos-server.org/neos/>,
- click: on the *Submit a job to NEOS*,
- scroll: to the *Nonlinearly Constrained Optimization* list,
- click: on *LOQO [AMPL input]*,
- scroll: to *Commands File*,
- click: on *Choose File*,
- select: a file from your computer that contains an AMPL model,
- click: on *Short Priority*,
- scroll: to *e-mail address*,
- type: your email address, and
- click: *Submit to NEOS*.

Piece of cake!

First Problem of First Assignment

Suppose the treasure is not exactly at the shore but rather is a certain distance away from the river. As shown, we are assuming the north shore of the river runs along the x -axis of our coordinate system. Assume that the river is $w = 30$ meters wide. For this problem, we need to figure out two things: (i) where you should enter the river and (ii) where you should exit it.



Write an AMPL model for this problem. Solve the problem using

$$(a_1, b_1) = (60, 40)$$

$$(a_2, b_2) = (-50, -50)$$

$$v_1 = 10$$

$$v_2 = 1.5$$

$$v_3 = 7$$

Report the x -coordinate of the location at which you should enter the river and the x -coordinate of the location at which you should exit the river.

The McDonald's Diet Problem

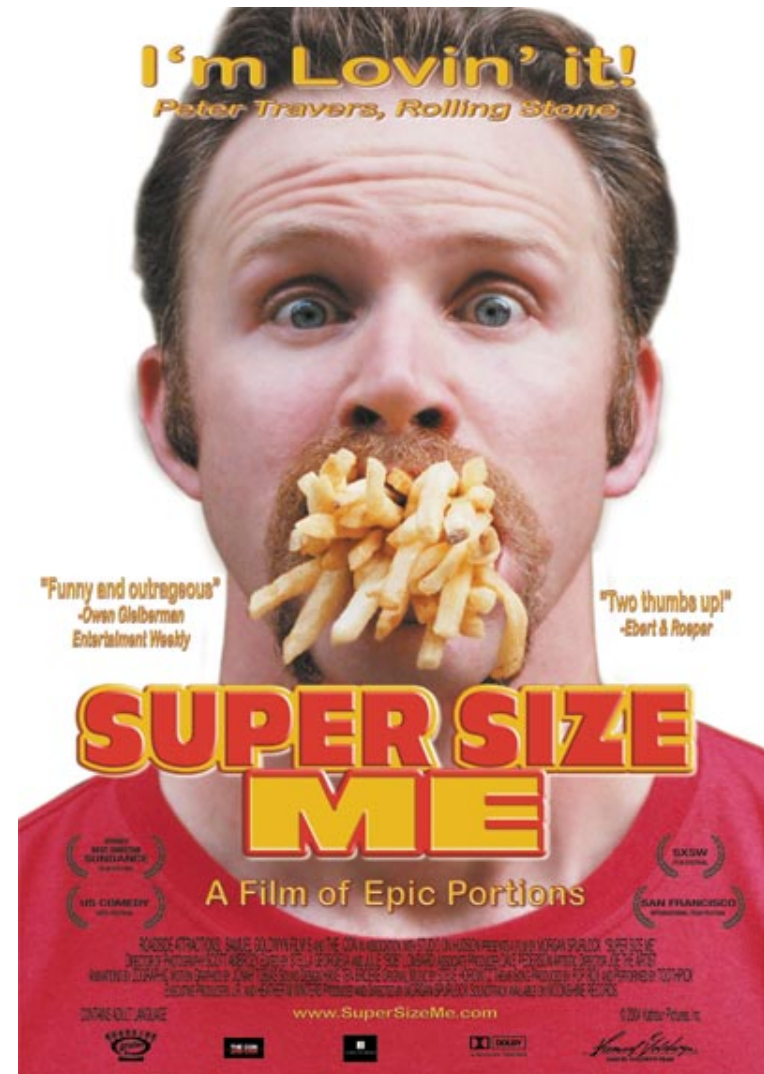
In words:

Minimize:

Calories

Subject to:

Total amounts of nutrients fall between certain minimum and maximum values.



Diet Problem: An AMPL Model

```
# --- Declare the data sets and parameters -----  
  
set NUTR;  
set FOOD;  
  
param f_min {FOOD} >= 0, default 0;  
param f_max {j in FOOD} >= f_min[j], default 200;  
  
param nutr_ideal {NUTR} >= 0;  
  
param amt {NUTR,FOOD} >= 0;  
  
# --- Declare the variables -----  
  
var Buy {j in FOOD} integer, >= f_min[j], <= f_max[j];  
  
# --- State the objective -----  
  
minimize Calories: sum {j in FOOD} amt["Cal",j] * Buy[j];  
  
# --- State the constraints -----  
  
subject to Dietary_bounds {i in NUTR}:  
    0.8*nutr_ideal[i] <= sum {j in FOOD} amt[i,j] * Buy[j] <= 1.2*nutr_ideal[i];
```

Diet Problem: The Data

```
param: NUTR:    nutr_ideal :=
        Cal      2500
        CalFat   600
        Fat      65
        SatFat   20
        Chol     300
        Sodium   2400
        Carbo    300
        Protein  50
        VitA     100
        VitC     100
        Calcium  100
        Iron     100
        ;

set FOOD :=
"Bacon_Buffalo_Ranch_McChicken 5.6_oz_(159_g)"
"Big_Breakfast_(Large_Size_Biscuit) 10_oz_(283_g)"
"Big_Mac 7.6_oz_(215_g)"
"Chicken_McNuggets_(10_piece) 5.7_oz_(162_g)"
"Coca-Cola_Classic_(Medium) 21_fl_oz_cup"
"Diet_Coke_(Medium) 21_fl_oz_cup"
"Double_Quarter_Pounder_with_Cheese++ 10_oz_(283_g)"
"Egg_McMuffin 4.8_oz_(135_g)"
"Frappe_Caramel_(Medium) 16_fl_oz_cup"
"Hamburger 3.5_oz_(100_g)"
"Hash_Brown 2_oz_(56_g)"
"Large_French_Fries 5.4_oz_(154_g)"
"Mac_Snack_Wrap 4.4_oz_(125_g)"
"McFlurry_with_M&M'S_Candies_(12_fl_oz_cup) 10.9_oz_(310_g)"
"McRib_ 7.3_oz_(208_g)"
"Medium_French_Fries 4.1_oz_(117_g)"
        # etc
        ;
```

param amt (tr):

	Cal	CalFat	Fat	SatFat	Chol	Sodium :=
"Bacon_Buffalo_Ranch_McChicken 5.6_oz_(159_g)"	420	180	20	4	50	1250
"Big_Breakfast_(Large_Size_Biscuit) 10_oz_(283_g)"	800	470	52	18	555	1680
"Big_Mac 7.6_oz_(215_g)"	550	260	29	10	75	970
"Chicken_McNuggets_(10_piece) 5.7_oz_(162_g)"	470	270	30	5	65	900
"Coca-Cola_Classic_(Medium) 21_fl_oz_cup"	200	0	0	0	0	5
"Diet_Coke_(Medium) 21_fl_oz_cup"	0	0	0	0	0	20
"Double_Quarter_Pounder_with_Cheese++ 10_oz_(283_g)"	750	380	43	19	160	1280
"Egg_McMuffin 4.8_oz_(135_g)"	290	110	12	5	260	740
"Frappe_Caramel_(Medium) 16_fl_oz_cup"	550	200	23	15	80	160
"Hamburger 3.5_oz_(100_g)"	250	80	9	3	25	480
"Hash_Brown 2_oz_(56_g)"	150	80	9	1	0	310
"Large_French_Fries 5.4_oz_(154_g)"	500	220	25	3	0	350
"Mac_Snack_Wrap 4.4_oz_(125_g)"	330	170	19	7	45	670
"McFlurry_with_M&M'S_Candies_(12_fl_oz_cup) 10.9_oz_(310_g)"	650	210	23	14	50	180
"McRib_ 7.3_oz_(208_g)"	500	240	26	10	70	980
"Medium_French_Fries 4.1_oz_(117_g)"	380	170	19	2	0	270
# etc						

	Carbo	Protein	VitA	VitC	Calcium	Iron :=
"Bacon_Buffalo_Ranch_McChicken 5.6_oz_(159_g)"	41	20	2	10	15	15
"Big_Breakfast_(Large_Size_Biscuit) 10_oz_(283_g)"	56	28	15	2	15	30
"Big_Mac 7.6_oz_(215_g)"	46	25	4	2	25	25
"Chicken_McNuggets_(10_piece) 5.7_oz_(162_g)"	30	22	0	4	2	6
"Coca-Cola_Classic_(Medium) 21_fl_oz_cup"	55	0	0	0	0	0
"Diet_Coke_(Medium) 21_fl_oz_cup"	0	0	0	0	0	0
"Double_Quarter_Pounder_with_Cheese++ 10_oz_(283_g)"	42	48	10	2	30	35
"Egg_McMuffin 4.8_oz_(135_g)"	31	17	10	0	25	15
"Frappe_Caramel_(Medium) 16_fl_oz_cup"	79	9	20	0	30	2
"Hamburger 3.5_oz_(100_g)"	31	12	2	2	10	15
"Hash_Brown 2_oz_(56_g)"	15	1	0	2	0	2
"Large_French_Fries 5.4_oz_(154_g)"	63	6	0	20	2	8
"Mac_Snack_Wrap 4.4_oz_(125_g)"	26	14	2	0	8	15
"McFlurry_with_M&M'S_Candies_(12_fl_oz_cup) 10.9_oz_(310_g)"	96	13	15	0	45	8
"McRib_ 7.3_oz_(208_g)"	44	22	2	2	15	20
"Medium_French_Fries 4.1_oz_(117_g)"	48	4	0	15	2	6
# etc						

;

Invoking the Solver and Displaying Results

```
solve;  
  
printf {f in FOOD: Buy[f] > 0.3}: "%-60s %6.2f %4d \n", f, Buy[f], amt["Cal",f];  
printf {i in NUTR}: "%-60s %7.1f (%4d) \n", i, sum {j in FOOD} amt[i,j] * Buy[j], nutr_ideal[i];
```

Complete AMPL model can be found here:

<https://vanderbei.princeton.edu/307/models/mcdonaldsdiet/idealDiet2014b.txt>

Complete table of nutritional data can be found at:

<http://nutrition.mcdonalds.com/getnutrition/nutritionfacts.pdf>

First Run

```
rvdb@stars $ ampl idealDiet2014b.mod
LOQO 7.00: verbose=0
ignoring integrality of 296 variables
LOQO 7.00: optimal solution (14 QP iterations, 14 evaluations)
primal objective 2000
    dual objective 1999.999986
```

Chocolate_Chip_Cookie 1_cookie_(33_g)	1.63	160
Diet_Coke_(Child) 12_fl_oz_cup	0.38	0
Diet_Coke_(Medium) 21_fl_oz_cup	0.32	0
Diet_Coke_(Small) 16_fl_oz_cup	0.56	0
EQUAL_0_Calorie_Sweetener 1_pkg_(1.0_g)	21.32	0
Hotcakes 5.3_oz_(151_g)	1.83	350
Iced_Tea_(Child) 12_fl_oz_cup	1.45	0
Iced_Tea_(Large) 30_fl_oz_cup	0.38	0
Iced_Tea_(Medium) 21_fl_oz	0.56	0
Iced_Tea_(Small) 16_fl_oz_cup	0.56	0
SPLENDA_No_Calorie_Sweetener 1_pkg_(1.0_g)	21.32	0
Side_Salad 3.1_oz_(87_g)	1.97	20

Cal	2000.0	(2500)
CalFat	609.6	(600)
Fat	68.2	(65)
SatFat	23.9	(20)
Chol	242.3	(300)
Sodium	2870.7	(2400)
Carbo	333.7	(300)
Protein	59.9	(50)
VitA	118.8	(100)
VitC	109.2	(100)
Calcium	92.8	(100)
Iron	80.1	(100)

Second Run: Limit Amount of Artificial Sweetener

```
subject to NotTooMuchSweetener:  
  Buy["EQUAL_0_Calorie_Sweetener 1_pkg_(1.0_g)"]  
  +Buy["SPLENDA_No_Calorie_Sweetener 1_pkg_(1.0_g)"]  
  <= 2* (  
    Buy["Iced_Tea_(Child) 12_fl_oz_cup"]  
    +Buy["Iced_Tea_(Large) 30_fl_oz_cup"]  
    +Buy["Iced_Tea_(Medium) 21_fl_oz"]  
    +Buy["Iced_Tea_(Small) 16_fl_oz_cup"]  
  );
```

Output:

Chocolate_Chip_Cookie 1_cookie_(33_g)	2.19	160
Diet_Coke_(Child) 12_fl_oz_cup	0.63	0
Diet_Coke_(Medium) 21_fl_oz_cup	0.44	0
Diet_Coke_(Small) 16_fl_oz_cup	0.94	0
EQUAL_0_Calorie_Sweetener 1_pkg_(1.0_g)	10.47	0
Fat_Free_Chocolate_Milk_Jug 1_carton_(236_ml)	0.68	130
Hotcakes 5.3_oz_(151_g)	1.53	350
Iced_Tea_(Child) 12_fl_oz_cup	7.24	0
Iced_Tea_(Large) 30_fl_oz_cup	1.02	0
Iced_Tea_(Medium) 21_fl_oz	1.98	0
Iced_Tea_(Small) 16_fl_oz_cup	1.98	0
SPLENDA_No_Calorie_Sweetener 1_pkg_(1.0_g)	10.47	0
Side_Salad 3.1_oz_(87_g)	1.83	20
Cal	2000.0	(2500)
CalFat	601.9	(600)
Fat	67.4	(65)
SatFat	23.9	(20)
Chol	243.5	(300)
Sodium	2742.7	(2400)
Carbo	313.7	(300)
Protein	59.9	(50)
VitA	118.4	(100)
VitC	110.0	(100)
Calcium	101.9	(100)
Iron	80.1	(100)

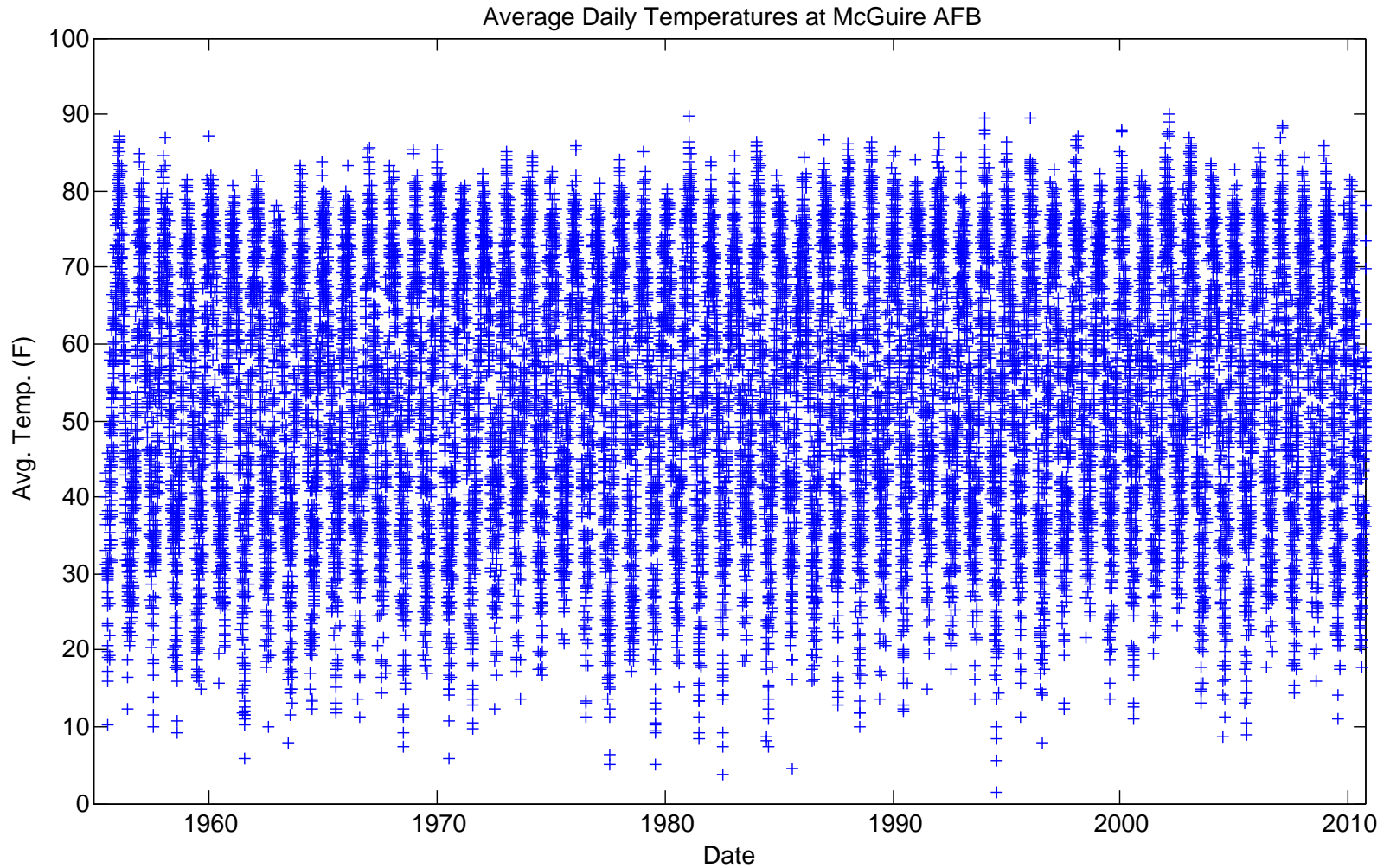
Third Run: Enforce Integrality Constraints

To do that we need a “Mixed Integer Linear Programming” solver such as Gurobi.

Output:

Chocolate_Chip_Cookie 1_cookie_(33_g)	6.00	160
Egg_McMuffin 4.8_oz_(135_g)	1.00	290
Fat_Free_Chocolate_Milk_Jug 1_carton_(236_ml)	2.00	130
Ketchup_Packet 1_pkg_(10_g)	5.00	10
Side_Salad 3.1_oz_(87_g)	1.00	20
Strawberry_Preserves 0.5_oz_(14_g)	12.00	35
Cal	2000.0	(2500)
CalFat	530.0	(600)
Fat	60.0	(65)
SatFat	23.0	(20)
Chol	330.0	(300)
Sodium	2060.0	(2400)
Carbo	330.0	(300)
Protein	48.0	(50)
VitA	97.0	(100)
VitC	83.0	(100)
Calcium	99.0	(100)
Iron	83.0	(100)

Climate Change



Yearly Averages

