Branch: master **JuMP.jl** / examples / **diet.jl** Find file Copy path

joehuchette Rename types and functions 91fee85 on Apr 25

3 contributors

85 lines (70 sloc) 2.49 KB Raw Blame History

```
1  #  Copyright 2016, Iain Dunning, Joey Huchette, Miles Lubin, and contributors
2  #  This Source Code Form is subject to the terms of the Mozilla Public
3  #  License, v. 2.0. If a copy of the MPL was not distributed with this
4  #  file, You can obtain one at http://mozilla.org/MPL/2.0/.
5  ##############################################################################
6  # JuMP
7  # An algebraic modelling langauge for Julia
8  # See http://github.com/JuliaOpt/JuMP.jl
9  ##############################################################################
10 # diet.jl
11 #
12 # Solve the classic "diet problem".
13 # Based on
14 #  http://www.gurobi.com/documentation/5.6/example-tour/diet_cpp_cpp
15 ##############################################################################
16
17 using JuMP
```

```
18
19  function PrintSolution(status, foods, buy)
20      println("RESULTS:")
21      if status == :Optimal
22          for i = 1:length(foods)
23              println("  $(foods[i]) = $(getvalue(buy[i]))")
24          end
25      else
26          println("  No solution")
27      end
28      println("")
29  end
30
31  function SolveDiet()
32
33      # Nutrition guidelines
34      numCategories = 4
35      categories = ["calories", "protein", "fat", "sodium"]
36      minNutrition = [1800, 91, 0, 0]
37      maxNutrition = [2200, Inf, 65, 1779]
38
39      # Foods
40      numFoods = 9
41      foods = ["hamburger", "chicken", "hot dog", "fries",
42                       "macaroni", "pizza", "salad", "milk", "ice cream"]
43      cost = [2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59]
44      nutritionValues = [410 24 26 730;
45                         420 32 10 1190;
46                         560 20 32 1800;
47                         380  4 19 270;
48                         320 12 10 930;
49                         320 15 12 820;
50                         320 31 12 1230;
51                         100  8 2.5 125;
52                         330  8 10 180]
```

```
53
54      # Build model
55      m = Model()
56
57      # Variables for nutrition info
58      @variable(m, minNutrition[i] <= nutrition[i=1:numCategories] <= maxNutrition[i])
59      # Variables for which foods to buy
60      @variable(m, buy[i=1:numFoods] >= 0)
61
62      # Objective - minimize cost
63      @objective(m, Min, dot(cost, buy))
64
65      # Nutrition constraints
66      for j = 1:numCategories
67          @constraint(m, sum{nutritionValues[i,j]*buy[i], i=1:numFoods} == nutrition[j])
68      end
69
70      # Solve
71      println("Solving original problem...")
72      status = solve(m)
73      PrintSolution(status, foods, buy)
74
75      # Limit dairy
76      @constraint(m, buy[8] + buy[9] <= 6)
77      println("Solving dairy-limited problem...")
78      status = solve(m)
79      PrintSolution(status, foods, buy)
80
81  end
82
83  SolveDiet()
84
```