Modeling of systems and processes

About

- What?
  - Computational modeling (statistical, simulation)
- How?
  - We meet three times per week
  - We think together and solve problems
  - You submit your work and I grade it
- ¿Por qué en inglés?
  - Por el planeta… (reciclaje)

Administrative stuff

- Instructor:
  - Juan Malagon
    - malagon@alumni.harvard.edu
    - https://www.linkedin.com/in/juanmalagon/
- GitHub repo:
  - https://github.com/juanmalagon/modeling-of-systems-and-processes.git
  - Create your own branch and name it as the last four digits of your CC
- Software requirements:
  - Python 3.10+

More administrative stuff

- Grades:
  - Problem set 1 (20%)
  - Problem set 2 (20%)
  - Problem set 3 (20%)
  - Final project (40%)
- Game’s rules:
  - Both the psets and the final project must be submitted to the course repo in GitHub (use your own branch)
  - Fell free to work collaboratively but always mention your sources and coworkers

(Some) references

- Hogg, R.V., McKean, J.W. and Craig, A.T. (2020) Introduction to mathematical statistics. Harlow, England: Pearson.
- Kroese, D.P., Taimre, T. and Botev, Z.I. (2011) Handbook of Monte Carlo Methods. Hoboken: John Wiley & Sons.
- Diez, D., Barr, C. and Çetinkaya-Rundel Mine (2014) Introductory statistics with randomization and simulation. Minneapolis, MN: Open Textbook Library.
- Blais, B. (2020) Statistical inference for everyone. Minneapolis, MN: Open Textbook Library.
- Gentle, J.E. (2010) Random number generation and Monte Carlo Methods. New York: Springer.
- Unpingco José H. (2022) Python for probability, statistics and machine learning. Cham: Springer Nature Switzerland.

Let’s go to GitHub

1. Log in GitHub with your account
2. Follow me so I can invite you as collaborators
3. Create your own branch

Let’s set up our Python environment

1. Follow the instructions about how to create a conda environment environment with Python 3.11 (alternatively, use a venv in VisualStudio Code)
2. Use pip to install `requirements.txt`

Computational Models

- Using computation to help understand the world in which we live
- Experimental devices that help us to understand something that has happened or to predict the future

  - Optimization models
  - Statistical models
  - Simulation models

What Is an Optimization Model?

- An objective function that is to be maximized or minimized, e.g.,
  - Minimize time spent traveling from Amsterdam to Eindhoven
- A set of constraints (possibly empty) that must be honored, e.g.,
- Cannot spend more than 100€
- Must be in Eindhoven before 5:00PM

Backpack Problem

- You have limited strength, so there is a maximum weight backpack that you can carry
- You would like to take more stuff than you can carry
- How do you choose which stuff to take and which to leave behind?
- Two variants
  - 0/1 backpack problem
  - Continuous or fractional backpack problem

Backpack Problem, Formalized

- Each item is represented by a pair, *<value, weight>*
- The backpack can accommodate items with a total weight of no more than w
- A vector, L, of length n, represents the set of available items. Each element of the vector is an item
- A vector, V, of length n, is used to indicate whether or not items are taken. If $V[i] = 1$, item $I[i]$ is taken. If $V[i] = 0$, item $I[i]$ is not taken

Backpack Problem, Formalized

Find a V that maximizes

$$\sum_{i=0}^{n-1} V[i] * I[i].value$$

subject to the constraint that

$$\sum_{i=0}^{n-1} V[i] * I[i].weight \leq w$$

Brute Force Algorithm

1. Enumerate all possible combinations of items. That is to say, generate all subsets of the set of items. This is called the power set.
2. Remove all of the combinations whose total units exceeds the allowed weight.
3. From the remaining combinations choose any one whose value is the largest.

Often Not Practical

- How big is power set?
- Recall
  - A vector, *V*, of length *n*, is used to indicate whether or not items are taken. If *V[i] = 1*, item *I[i]* is taken. If *V[i] = 0*, item *I[i]* is not taken
- How many possible different values can *V* have?
  - As many different binary numbers as can be represented in *n* bits
- For example, if there are 100 items to choose from, the power set is of size?
  - 1,267,650,600,228,229,401,496,703,205,376

Are We Just Being Stupid?

- Alas, no
- 0/1 backpack problem is inherently exponential
- But don't despair

Greedy Algorithm a Practical Alternative

```
while backpack not full
      put "best" available item in backpack
```

- But what does best mean?
  - Most valuable
  - Least expensive
  - Highest value/units

An Example (by John Guttag)

- You are about to sit down to a meal
- You know how much you value different foods, e.g., you like donuts more than apples
- But you have a calorie budget, e.g., you don't want to consume more than 750 calories
- Choosing what to eat is a backpack problem

A Menu

Food	wine	beer	pizza	burger	fries	coke	apple	donut
value	89	90	30	50	90	79	90	10
calories	123	154	258	354	365	150	95	195

- Let's look at a program that we can use to decide what to order

Class Food

```
class Food(object):
    def __init__(self, n, v, w):
        self.name = n
        self.value = v
        self.calories = w
    def getValue(self):
        return self.value
    def getCost(self):
        return self.calories
    def density(self):
        return self.getValue()/self.getCost()
    def __str__(self):
        return self.name + ': <' + str(self.value)\
                 + ', ' + str(self.calories) + '>'
```

Build Menu of Foods

```
def buildMenu(names, values, calories):
    """names, values, calories lists of same length.
       name a list of strings
       values and calories lists of numbers
       returns list of Foods"""
    menu = []
    for i in range(len(values)):
        menu.append(Food(names[i], values[i],
                          calories[i]))
    return menu
```

Implementation of Flexible Greedy

```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,
         keyFunction maps elements of items to numbers"""
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

Algorithmic Efficiency

```
def greedy(items, maxCost, keyFunction):
    """Assumes items a list, maxCost >= 0,
         keyFunction maps elements of items to numbers"""
    itemsCopy = sorted(items, key = keyFunction,
                       reverse = True)
    result = []
    totalValue, totalCost = 0.0, 0.0
    for i in range(len(itemsCopy)):
        if (totalCost+itemsCopy[i].getCost()) <= maxCost:
            result.append(itemsCopy[i])
            totalCost += itemsCopy[i].getCost()
            totalValue += itemsCopy[i].getValue()
    return (result, totalValue)
```

Using greedy

```
def testGreedy(items, constraint, keyFunction):
    taken, val = greedy(items, constraint, keyFunction)
    print('Total value of items taken =', val)
    for item in taken:
        print('   ', item)
```

Using greedy

```
def testGreedys(maxUnits):
    print('Use greedy by value to allocate', maxUnits,
    'calories')
  testGreedy(foods, maxUnits, Food.getValue)  print('\nUse
  greedy by cost to allocate', maxUnits,
        'calories')
  testGreedy(foods, maxUnits,
            lambda x: 1/Food.getCost(x))
    print('\nUse greedy by density to allocate', maxUnits,
    'calories')
  testGreedy(foods, maxUnits, Food.density)
```

lambda

- lambda used to create anonymous functions
    - lambda <id1, id2, … idn>: <expression>
    - Returns a function of n arguments
- Can be very handy, as here
- Possible to write amazing complicated lambda expressions
- Don't—use def instead

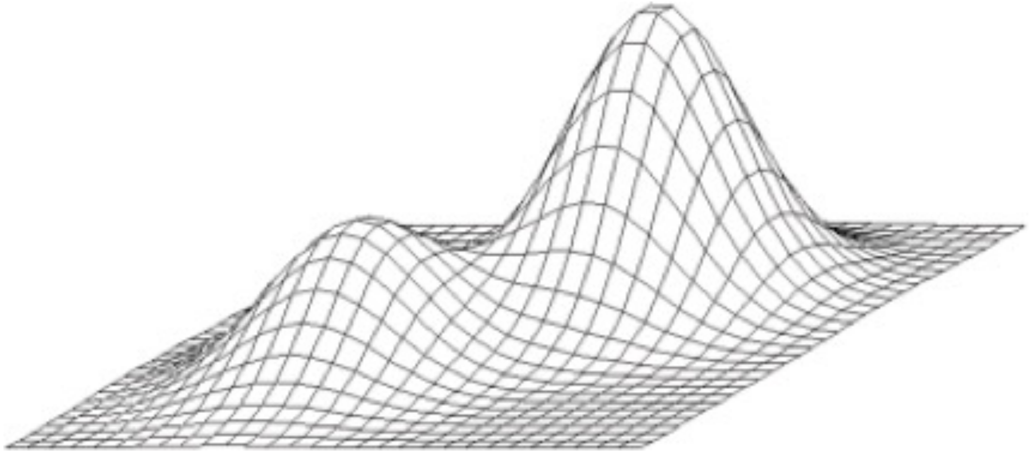
Using greedy

```
names = ['wine', 'beer', 'pizza', 'burger', 'fries',
         'cola', 'apple', 'donut', 'cake']
values = [89,90,95,100,90,79,50,10]
calories = [123,154,258,354,365,150,95,195]


foods = buildMenu(names, values, calories)
testGreedys(foods, 750)
```

Why Different Answers?

- Sequence of locally ”optimal” choices don't always yield a globally optimal solution



- Is greedy by density always a winner?
    - Try `testGreedys(foods, 1000)`

The Pros and Cons of Greedy

- Easy to implement
- Computationally efficient

- But does not always yield the best solution
  - Don't even know how good the approximation is

References

Prof. Eric Grimson, Prof. John Guttag, Dr. Ana Bell. Introduction to Computational Thinking and Data Science. Fall 2016. Massachusetts Institute of Technology: MIT OpenCouseWare, https://ocw.mit.edu/. License: Creative Commons BY-NC-SA.