0311-Ergebnisse-DynamischeProgrammierung

March 12, 2019

0.1 Dynamisches Programm zur Implementierung der Value Function

Wieder möchten wir verschiedene Arten der Implementierung vergleichen. Konkret betrachten wir die Funktion

In [2]:
```
# -*- coding: utf-8 -*-
"""
Created on Mon Mar  11 14:46:31 2019

@author: Stefan
"""

# %% PACKAGES
%matplotlib notebook
import matplotlib.pyplot as plt

# Data
import numpy as np
import pandas as pd

# Calculation and Counting
import itertools
import math

# Distributions
from scipy.stats import bernoulli

# Timing
import time

# Memoization
import functools

# Genaueres inspizieren von Funktionen
# import inspect
# lines = inspect.getsource(value_expected)
# print(lines)
```

```python
# %% OVERALL PARAMETERS
numProducts = 4
products = np.arange(numProducts) + 1  # only real products (starting from 1)
revenues = np.array([1000, 800, 600, 400])  # only real products

numPeriods = 10

customer_segments_num = 1
arrivalProbability = 0.8
preference_weights = np.array([0.4, 0.8, 1.2, 1.6])

varNoPurchasePreferences = np.array([1, 2, 3])
varCapacity = np.arange(40, 120, 20)
# %% LOCAL PARAMETERS


# %% JUST TEMPORARY
preference_no_purchase = 2
capacity = 6
offer_set = np.array([1, 0, 1, 1])


# %% FUNCTIONS
def memoize(func):
    cache = func.cache = {}

    @functools.wraps(func)
    def memoizer(*args, **kwargs):
        key = str(args) + str(kwargs)
        if key not in cache:
            cache[key] = func(*args, **kwargs)
        return cache[key]
    return memoizer


def customer_choice_individual(offer_set_tuple):
    """
    For one customer of one customer segment, determine its purchase probabilities giv

    Tuple needed for memoization.

    :param preference_weights: vector indicating the preference for each product
    :param preference_no_purchase: preference for no purchase
    :param offer_set: vector with offered products indicated by 1=product offered
    :return: vector of purchase probabilities starting with no purchase
    """
    offer_set = np.asarray(offer_set_tuple)
    ret = preference_weights * offer_set
```

```
    ret = np.array(ret / (preference_no_purchase + sum(ret)))
    ret = np.insert(ret, 0, 1 - sum(ret))
    return ret


def sample_path(num_periods, arrival_probability):
    """
    Calculates the sample path.

    :param num_periods:
    :param arrival_probability:
    :return: Vector with arrival of customers.
    """
    return bernoulli.rvs(size=num_periods, p=arrival_probability)


def history(numPeriods, arrivalProbability, capacity, offerSet, revenues):
    """

    Over one complete booking horizon with *numPeriod* periods and a total *capacity*,
    TODO: calculate *offerSet* over time.
    RETURN: data frame with columns (time, capacity (at start), customer arrived, prod
    *customerArrived*: ID of
    *customerPreferences*: for each customer segment stores the preferences to determi

    Helpers
    ----
    customerArrived :
        the ID of the customer segment that has arrived (used for customer preferences

    :param numPeriods:
    :param arrivalProbability:
    :param capacity:
    :param offerSet:
    :param revenues:
    :return:
    """

    index = np.arange(numPeriods + 1)[::-1]  # first row is a dummy (for nice for loop
    columns = ['capacityStart', 'customerArrived', 'productSold', 'revenue', 'capacityE

    df_history = pd.DataFrame(index=index, columns=columns)
    df_history = df_history.fillna(0)

    df_history.loc[numPeriods, 'capacityStart'] = df_history.loc[numPeriods, 'capacityE
    df_history.loc[(numPeriods - 1):0, 'customerArrived'] = sample_path(numPeriods, ar

    revenues_with_no_purchase = np.insert(revenues, 0, 0)
```

```
    products_with_no_purchase = np.arange(len(revenues_with_no_purchase))

    customer_probabilities = customer_choice_individual(offerSet)

    for i in np.delete(index, 0):  # start in second row (without actually deleting ro
        if df_history.loc[i, 'customerArrived'] == 1:
            if df_history.loc[i + 1, 'capacityEnd'] == 0:
                break
            # A customer has arrived and we have capacity.

            df_history.loc[i, 'capacityStart'] = df_history.loc[i + 1, 'capacityEnd']

            df_history.loc[i, 'productSold'] = np.random.choice(products_with_no_purcha
                                                                p=customer_probabiliti

            df_history.loc[i, 'revenue'] = revenues_with_no_purchase[df_history.loc[i,

            if df_history.loc[i, 'productSold'] != 0:
                df_history.loc[i, 'capacityEnd'] = df_history.loc[i, 'capacityStart'] -
            else:
                df_history.loc[i, 'capacityEnd'] = df_history.loc[i, 'capacityStart']
        else:
            # no customer arrived
            df_history.loc[i, 'capacityEnd'] = df_history.loc[i, 'capacityStart'] = df_

    return df_history


def value_expected(capacity, time):
    """
    Recursive implementation of the value function, i.e. dynamic program (DP)

    :param capacity:
    :param time:
    :return: value to be expected
    """
    offer_sets_to_test = list(map(list, itertools.product([0, 1], repeat=len(products)))
    offer_sets_max = 0
    offer_sets_max_val = 0

    if capacity == 0:
        return 0
    if capacity < 0:
        return -math.inf
    if time == 0:
        return 0

    for offer_set_index in range(len(offer_sets_to_test)):
```

```
                offer_set = offer_sets_to_test[offer_set_index]
                probs = customer_choice_individual(tuple(offer_set))

                val = value_expected(capacity, time - 1)
                for j in products:
                    p = float(probs[j])
                    if p > 0.0:
                        value_delta = value_expected(capacity, time - 1) - \
                                      value_expected(capacity - 1, time - 1)
                        val += p * (revenues[j - 1] - value_delta)  # j-1 shifts to right prod

                if val > offer_sets_max_val:
                    offer_sets_max_val = val
                    offer_sets_max = offer_set_index

            return offer_sets_max_val
```

```
In [4]: %timeit value_expected(3,3)
```

```
5.71 s ś 334 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [5]: customer_choice_individual = memoize(customer_choice_individual)
        %timeit value_expected(3,3)
```

```
2.85 s ś 66.9 ms per loop (mean ś std. dev. of 7 runs, 1 loop each)
```

```
In [6]: value_expected = memoize(value_expected)
        %timeit value_expected(3,3)
```

```
875 ns ś 23.5 ns per loop (mean ś std. dev. of 7 runs, 1000000 loops each)
```