Министерство образования и науки Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Лабораторная работа №4 по дисциплине

“Системное программное обеспечение”

Вариант №3

Выполнили:

студенты группы R3335

Кузьминов А. Ю.

Никитенко И. Р.

Гринжола Д. Ю.

Преподаватель:

кандидат технических наук

Капитонов А. А.

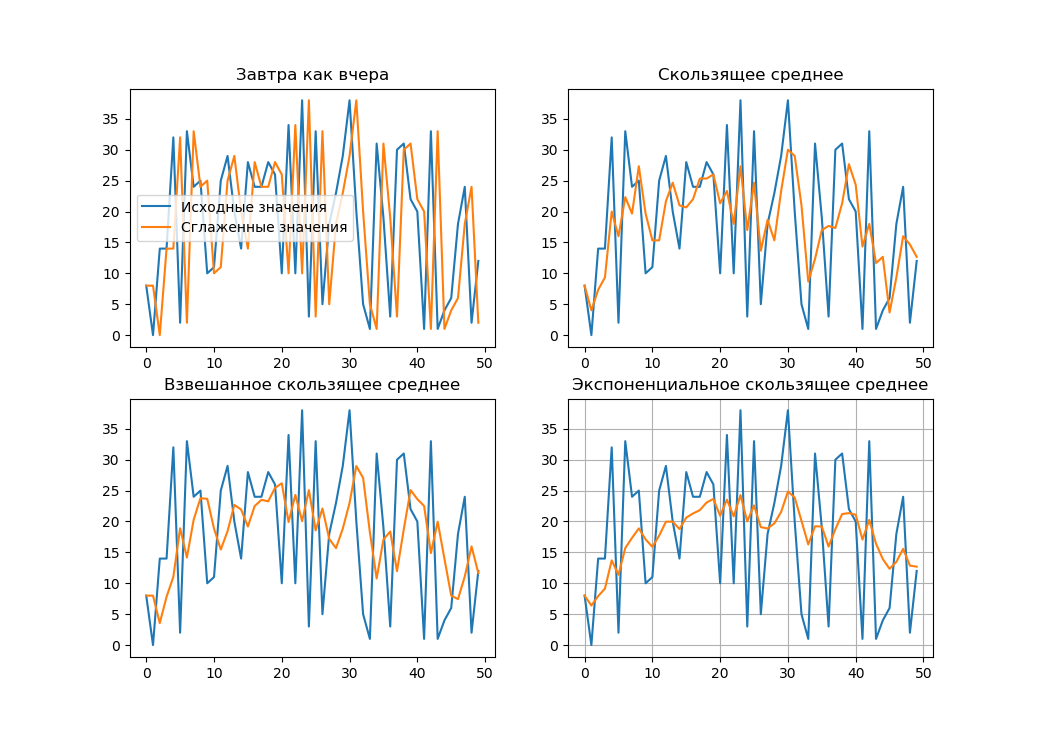
Санкт-Петербург

2018

Задание 1. Сглаживание

1. Сгенерировать случайный список с 50 числами в диапазоне от 0 до 40.
2. Реализовать 4 любых алгоритма сглаживания для заданного списка без использования готовых команд/библиотек. Построить график для исходных данных и каждого из выбранных алгоритмов.
3. Сделать выводы о выбранных методах. Выделить их достоинства и недостатки, область применения.

Вывод графиков:



Код программы:

|  |
| --- |
| **import** random  **import** matplotlib.pyplot **as** plt    num = 50    *#Завтра как вчера*  **def** tomAsYest(nums):  filtered = []  filtered.append(nums[0])  length = len(nums)  **for** i **in** range(1,length):  filtered.append(nums[i-1])  **return** filtered    *#Скользящее среднее*  **def** movAv(nums):  filtered = []  filtered.append(nums[0])  filtered.append((nums[0]+nums[1])/2.0)  summ = nums[0] + nums[1]  n = 3  length = len(nums)  **for** i **in** range(2,length):  summ += nums[i]  filtered.append( 1.0 \* summ / n )  summ -= nums[i-2]  **return** filtered    *#Взвешенное скользящее среднее*  **def** weightedMovAv(nums):  weightNums = [0.06, 0.09, 0.16, 0.29, 0.4]  filtered = [nums[0]]  length = len(nums)  **for** i **in** range(1, length):  y = 0  weightSum = 0  **for** j **in** range(len(weightNums)):  **if** (i - (len(weightNums) - j)) < 0:  **continue**  y += nums[i - (len(weightNums) - j)] \* weightNums[j]  weightSum += weightNums[j]    filtered.append(y / weightSum)  **return** filtered    *#Экспоненциальное скользящее среднее*  **def** expFilt(nums):  filtered = []  alpha = 0.2  filtered.append(nums[0])  length = len(nums)  **for** i **in** range(1,length):  filtered.append(alpha\*nums[i]+(1-alpha)\*filtered[i-1])  **return** filtered    **def** createList(left, right, num):  dat = []  **for** i **in** range(num):  dat.append(random.randint(left, right))  **return** dat    nums = []  nums = createList(0,40,num)  filter1 = tomAsYest(nums)  filter2 = movAv(nums)  filter3 = weightedMovAv(nums)  filter4 = expFilt(nums)      x = []  **for** i **in** range(num):  x.append(i)    plt.subplot(2, 2, 1)  plt.title('Завтра как вчера')  plt.plot(x, nums, label='Исходные значения')  plt.plot(x, filter1, label='Сглаженные значения')  plt.legend()    plt.subplot(2, 2, 2)  plt.title('Скользящее среднее')  plt.plot(x, nums, label='Initial numbers')  plt.plot(x, filter2, label='Moving Avarage')    plt.subplot(2, 2, 3)  plt.title('Взвешанное скользящее среднее')  plt.plot(x, nums, label='Initial numbers')  plt.plot(x, filter3, label='Weighted Moving Avarage')    plt.subplot(2, 2, 4)  plt.title('Экспоненциальное скользящее среднее')  plt.plot(x, nums, label='Initial numbers')  plt.plot(x, filter4, label='Exponential Filter')    plt.grid()  plt.show() |

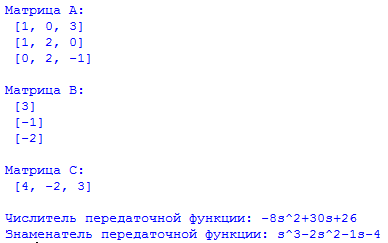
Анализ преимуществ и недостатков методов сглаживания.

|  |  |  |
| --- | --- | --- |
| Метод | Преимущества | Недостатки |
| Завтра как вчера | Простота реализации | Плохо сглаживает значения, просто сдвигает данные относительно первого значения вправо |
| Скользящее среднее | Простота реализации  Сглаживает значения | Веса всех значений одинаковы не зависимо от того, насколько они отдалены от текущего  Двойная реакция на каждое значение: в момент входа в окно вычислений и в момент выхода из него  Запаздывает реакция на изменение данных – максимумы сдвинуты вправо относительно исходных значений |
| Взвешенное скользящее среднее | Реагирует на ближайшие к данному в ряду значения сильнее, чем на более отдаленные | Вес не зависит от значения, поэтому максимумы сглаживаются не сильно  Запаздывает реакция на изменение данных – максимумы сдвинуты вправо относительно исходных значений |
| Экспоненциальное скользящее среднее | Реагирует на ближайшие к данному в ряду значения сильнее, чем на более отдаленные  Вес элемента зависит от его величины  Реакция на скачки запаздывает меньше, чем у других методов | Все еще запаздывает реакция на резкие изменения данных |

Задание 2. «Вход-Состояние-Выход» и «Вход-Выход»

1. Заданы случайные матрицы состояния, управления и выхода A, B, C размерности 3х3, 3х1, 1х3.
2. Разработать программу, которая используя заданные матрицы и не используя готовые команды/библиотеки, рассчитает передаточную функцию системы, используя алгоритм W(s)=C(sI-A)-1B
3. Исходя из полученной передаточной функции системы, сделайте выводы о наличии проблем с управляемостью/наблюдаемостью в системе (см. лекции).

Пример выполнения программы:



Код программы:

|  |
| --- |
| *#импорт библиотеки (только для создания случайных матриц)*  **import** numpy **as** np    *#вывод данных матриц*  A = np.random.randint(-10, 10, (3, 3))  B = np.random.randint(-10, 10, (3, 1))  C = np.random.randint(-10, 10, (1, 3))  **print**("Матрица А:", "**\n**", A[0], "**\n**", A[1], "**\n**", A[2], "**\n**")  **print**("Матрица B:", "**\n**", B[0], "**\n**", B[1], "**\n**", B[2], "**\n**")  **print**("Матрица C:", "**\n**", C[0], "**\n**")    *#нахождение коэффициентов числителя и знаменателя предаточной функции*  d\_k1 = -A[0][0]\*A[1][1]\*A[2][2]+A[0][0]\*A[2][1]\*A[1][2]+A[1][0]\*A[2][2]\*A[0][1]- A[1][0]\*A[0][2]\*A[2][1]-A[2][0]\*A[0][1]\*A[1][2]+A[0][2]\*A[1][1]\*A[2][0]  d\_k2 = A[1][1]\*A[2][2]-A[2][1]\*A[1][2]+A[0][0]\*A[2][2]+A[0][0]\*A[1][1]-A[1][0]\*A[0][1]-A[2][0]\*A[0][2]  d\_k3 = -A[0][0]-A[1][1]-A[2][2]    n\_k1 = (B[0][0]\*C[0][0]\*(A[1][2]\*A[2][1]-A[1][1]\*A[2][2]) + B[0][0]\*C[0][1]\*(A[1][0]\*A[2][2]-A[1][2]\*A[2][1]) + B[0][0]\*C[0][2]\*(A[1][1]\*A[2][0]-A[1][0]\*A[2][1])  + B[1][0]\*C[0][0]\*(A[0][1]\*A[2][2]-A[0][1]\*A[2][2]) + B[1][0]\*C[0][1]\*(A[0][2]\*A[2][0]-A[2][2]\*A[0][0]) + B[2][0]\*C[0][2]\*(A[0][0]\*A[2][1]-A[0][1]\*A[2][0])  + B[2][0]\*C[0][0]\*(A[0][2]\*A[2][2]-A[0][1]\*A[1][2]) + B[2][0]\*C[0][1]\*(A[0][0]\*A[1][2]-A[0][2]\*A[1][0]) + B[2][0]\*C[0][2]\*(A[0][1]\*A[1][0]-A[0][0]\*A[1][1]))  n\_k2 = (B[0][0]\*C[0][0]\*(A[1][1]+A[2][2]) - B[0][0]\*C[0][1]\*A[1][0] - B[0][0]\*C[0][2]\*A[2][0] - B[1][0]\*C[0][0]\*A[0][1] + B[1][0]\*C[0][1]\*(A[0][0]+A[2][2])  - B[1][0]\*C[0][2]\*A[2][1] - B[2][0]\*C[0][0]\*A[0][2] - B[2][0]\*C[0][1]\*A[1][2] + B[2][0]\*C[0][2]\*(A[0][0]+A[1][1]))  n\_k3 = ((-B[0][0]\*C[0][0])+(-B[1][0]\*C[0][1])+(-B[2][0]\*C[0][2]))    *#коэффициенты числителя*  num\_koef = []  num\_koef.append(n\_k1)  num\_koef.append(n\_k2)  num\_koef.append(n\_k3)    *#коэффициенты знаменателя*  den\_koef = []  den\_koef.append(d\_k1)  den\_koef.append(d\_k2)  den\_koef.append(d\_k3)  den\_koef.append(1)    *#определение знаков коэффициентов числителя*  signs = []  **for** i **in** range(3):  **if** num\_koef[i] > 0:  signs.append("+")  **else**:  signs.append("-")  num = signs[2] + str(abs(num\_koef[2])) + 's^2' + signs[1] + str(abs(num\_koef[1]))+'s' + signs[0] + str(abs(num\_koef[0]))    *#определение знаков коэффициентов знаменателя*  signs = []  **for** i **in** range(2):  **if** den\_koef[i] > 0:  signs.append("+")  **else**:  signs.append("-")  den = 's^3' + signs[0] + str(abs(den\_koef[2]))+'s^2' + signs[1] +str(abs(den\_koef[1]))+'s' + signs[0] +str(abs(den\_koef[0]))    *#вывод передаточной функции на экран*  **print** ("Числитель передаточной функции:", num)  **print** ("Знаменатель передаточной функции:", den) |

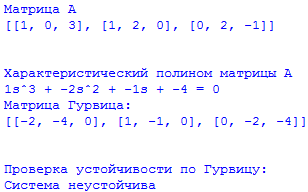
Система управляема и наблюдаема, только если алгебраические уравнения не имеют общих корней, т.е. если порядок передаточной функции системы W(s)=CB совпадает с порядком системы дифференциального уравнения

и равен n. В случае неполной управляемости или наблюдаемости порядок передаточной функции меньше, чем n.

Задание 3. Устойчивость системы.

1. Изучите методы определения устойчивости системы.
2. Используя данные из задания 2, напишите программу, которая будет рассчитывать, является ли система устойчивой, не устойчивой или на границе устойчивости, не используя готовые команды/библиотеки.

Пример выполнения программы:



Код программы:

|  |
| --- |
| **import** numpy **as** np  A = np.random.randint(-10, 10, (3, 3))  **print**("Матрица А")  **print**(A)  **print**("**\n**")  **def** polinom(a):  koefs = []  k0 = 1  koefs.append(k0)  k1 = - a[0][0] - a[1][1] - a[2][2]  koefs.append(k1)  k2 = a[0][0]\*a[1][1] + a[0][0]\*a[2][2] + a[1][1]\*a[2][2] - a[1][2]\*a[2][1] - a[0][1]\*a[1][0] - a[0][2]\*a[2][0]  koefs.append(k2)  k3 = -a[0][0]\*a[1][1]\*a[2][2] + a[0][0]\*a[1][2]\*a[2][1] + a[0][1]\*a[1][0]\*a[2][2] - a[0][1]\*a[1][2]\*a[2][0] - a[0][2]\*a[1][0]\*a[2][1] + a[0][2]\*a[1][1]\*a[2][0]  koefs.append(k3)    **return** koefs    **def** gurv(a):  koefs = polinom(a)  M = [[koefs[1],koefs[3],0],[koefs[0],koefs[2],0],[0,koefs[1],koefs[3]]]  **print** ("Матрица Гурвица:")  **print** (M)  **print** ("**\n**")  **print** ("Проверка устойчивости по Гурвицу:")  **if** (koefs[1] > 0) **and** (koefs[2]\*koefs[1]-koefs[3]\*koefs[0] > 0) **and** (koefs[3]>0):  **return** "Система устойчива"  **if** (koefs[1] < 0) **or** (koefs[2]\*koefs[1]-koefs[3]\*koefs[0] < 0) **or** (koefs[3]<0):  **return** "Система неустойчива"  **if** (koefs[1] > 0) **and** (koefs[2]\*koefs[1]-koefs[3]\*koefs[0] > 0) **and** (koefs[3]==0):  **return** "Система на апериодической гнанице устойчивости"  **if** (koefs[1] > 0) **and** (koefs[2]\*koefs[1]-koefs[3]\*koefs[0] == 0) **and** (koefs[3]>0):  **return** "Система на колебательной гнанице устойчивости"        koefs = polinom(A)    **print**('Характеристический полином матрицы А')  **print**(str(koefs[0])+'sˆ3 + '+str(koefs[1])+'sˆ2 + '+str(koefs[2])+'s + '+str(koefs[3])+' = 0')    **print**(gurv(A)) |

Ссылка на GitHub: <https://github.com/nikotheflow/Python-labs>