

ПЕРВОЕ ВЫСШЕЕ ТЕХНИЧЕСКОЕ УЧЕБНОЕ ЗАВЕДЕНИЕ РОССИИ



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**федеральное государственное бюджетное образовательное учреждение
высшего образования**

«Санкт-Петербургский горный университет»

Кафедра информационных систем и вычислительной техники

КУРСОВАЯ РАБОТА

По дисциплине Технологии обработки информации
(наименование учебной дисциплины согласно учебному плану)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Тема работы: Исследование фильтра Калмана

Выполнил: студент гр. ИСТ-21-1 _____ Темнов А.М.
(шифр группы) (подпись) (Ф.И.О.)

Оценка:

Дата: _____

Проверил
руководитель работы:

<u>ассистент</u>	<u> </u>	<u>Смирнов А.И.</u>
(должность)	(подпись)	(Ф.И.О.)

Санкт-Петербург

2022

РЕФЕРАТ

Пояснительная записка 27 с., 13 рис, 10 источников, 2 приложения.

АНАЛИЗ ДАННЫХ, ФИЛЬТР КАЛМАНА, PYTHON, НОРМАЛЬНОЕ РАСПРЕДЕЛЕНИЕ, GOOGLE COLAB

Объектом исследования является анализ данных.

Предмет исследования – фильтр Калмана.

Цель исследования – исследование фильтра Калмана.

Для достижения цели курсовой работы был проведён анализ предметной области, а также были изучены способы работы с фильтром на языке Python.

В результате проведённого исследования был исследован фильтр Калмана и реализован его код на ЯП Python в среде Google Colab.

СОДЕРЖАНИЕ

РЕФЕРАТ	2
ВВЕДЕНИЕ.....	4
1 ОБЩИЕ СВЕДЕНИЯ О ФИЛЬТРЕ КАЛМАНА	5
1.1 Фильтр Калмана: определение и сфера использования.....	5
1.2 Требования к ДС для корректной фильтрации	5
1.3 Формальная постановка задачи	6
2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ СИСТЕМЫ.....	8
2.1 Математическая модель процесса.....	8
2.2 Математическая модель наблюдения	9
2.3 Ковариационные матрицы	10
2.3.1 Матрица состояния P	10
2.3.2 Матрица шума R	10
2.3.3 Матрица ошибки модели Q	11
3 РЕАЛИЗАЦИЯ ФИЛЬТРА КАЛМАНА НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ ВСТРОЕННОЙ БИБЛИОТЕКИ FILTERPY	12
3.1 Предварительные расчёты и реализация модели	12
3.2 Реализация на Python	13
4 НЕДОСТАТКИ И МОДИФИКАЦИИ ФИЛЬТРА КАЛМАНА	16
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ	21
ПРИЛОЖЕНИЯ.....	22
ПРИЛОЖЕНИЕ А.....	22
ПРИЛОЖЕНИЕ Б.....	27

ВВЕДЕНИЕ

Под анализом данных понимается совокупность методов и приложений, связанных с алгоритмами обработки данных и не имеющих четко зафиксированного ответа на каждый входящий объект [1].

Цель курсовой работы – провести исследование фильтра Калмана: изучить основные принципы работы этого алгоритма, реализовать фильтр Калмана на ЯП Python, сравнить его с другими алгоритмами фильтрации данных.

Для этого необходимо тщательно изучить данные, с которыми фильтрация Калмана работает корректно, исследовать математическую модель фильтра, рассмотреть основные виды и способы реализации алгоритма фильтра Калмана на языке программирования Python, выявить преимущества и недостатки исследуемого алгоритма по сравнению с другими фильтрами, а также оформить пояснительную записку по ГОСТ 7.32-2017 [2].

1 ОБЩИЕ СВЕДЕНИЯ О ФИЛЬТРЕ КАЛМАНА

1.1 Фильтр Калмана: определение и сфера использования

Фильтр Калмана — эффективный рекурсивный фильтр, оценивающий вектор состояния динамической системы, используя ряд неполных и зашумленных измерений. Назван в честь Рудольфа Калмана [3]. Целью фильтрации Калмана является «предсказание» поведения динамической системы (далее — ДС), если само «измерение» состояния ДС не является точным ввиду наличия различных «шумов» и помех.

Этот фильтр эффективен для непрерывно меняющихся систем, так как он является рекурсивным — ему необходимо учитывать лишь предыдущее состояние системы для вычисления текущего (следует из определения). Таким образом, некоторыми преимуществами фильтра Калмана являются то, что ему не нужны большие объёмы памяти, и также он эффективен с точки зрения скорости работы.

1.2 Требования к ДС для корректной фильтрации

Во-первых, модель ДС должна быть линейной [4]. Иначе фильтра Калмана работает не совсем корректно и необходимо вносить различные изменения в принципы его работы. То есть уравнения состояний ДС не должны содержать в себе степенных, тригонометрических, логарифмических и других элементарных функций.

Во-вторых, модель должна быть дискретной. Для более точного рекурсивного обращения к последующим состояниям необходимо измерять состояние системы через равные промежутки времени (например, каждую секунду). Таким образом, на каждом шаге фильтр берёт вектор состояния (переменные,

описывающие состояния) ДС, вычисленный на предыдущем (с учётом шумов и пробелов), и с помощью этих данных оценивает вектор состояния в текущий момент. Обычно используются следующие обозначения:

x – вектор состояния;

P – мера неопределённости вектора состояния.

Содержимое вектора состояния x зависит от конкретной ДС: оно может содержать как одну переменную (например, координату), так и несколько (координату, скорость и ускорение и проч.).

Важнейшим допущением для работы фильтра Калмана является то, что все погрешности и ошибки (при вычислении вектора x или при, непосредственно, измерениях) имеют нормальное (Гауссовское) распределение [5]. А для многомерного нормального распределения его полностью определяют два параметра: математическое ожидание вектора и его ковариационная матрица.

1.3 Формальная постановка задачи

Пусть x – состояние, которое необходимо измерить и оценить в данной ДС. При этом x есть список чисел, задающих некоторые параметры (например, позиция, скорость и ускорение или температура и давление и тд.). Параметры (все или некоторые из них) измеряются с помощью каких-либо приборов и датчиков, которые сами по себе имеют некоторую погрешность. Также существуют различные шумы и неточности, возникающие при внешнем воздействии. Таким образом, измеренный результат не является истинным. Таким образом, с помощью фильтра Калмана необходимо оценить текущее состоя-

ние как можно ближе к истинному значению x , используя знания о предыдущем значении. При этом общая оценка системы является более точной, чем оценка отдельных измерений.

2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ СИСТЕМЫ

2.1 Математическая модель процесса

Рассмотрим математическую модель фильтра Калмана для тела, движущегося равноускорено по следующей известной из курса физики формуле:

$$x_t = x_0 + v_0 t + \frac{at^2}{2} \quad (1)$$

Сама математическая модель процесса имеет вид:

$$x_k = Fx_{k-1} + Bu_k + w_k \quad (2)$$

Здесь:

- Fx_{k-1} – модель эволюции процесса, содержащее саму формулу перемещения (1). F – есть матрица процесса, задающая систему линейных уравнений, описывающих последующие состояния через предыдущие. Учитывая, что скорость (v) – первая производная перемещения по времени, а ускорение (a) – вторая производная этой же величины:

$$F = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix};$$

- Bu_k – произведение матрицы управления на вектор управления. Это слагаемое необходимо, если мы оказываем какое-либо управляющее воздействие на систему. Если же мы являемся наблюдателями, то это слагаемое отсутствует;

- w – вектор ошибки модели, определяющая внешние факторы (неровности на дорогах, сопротивление ветра и т.д.). При этом не факт, что это

слагаемое нам точно известно, но фильтр Калмана предполагает, что оно гарантированно имеет нормальное распределение с нулевым математическим ожиданием и ковариационной матрицей Q .

2.2 Математическая модель наблюдения

Задачей фильтра Калмана является оценка состояния динамической системы с учётом шумов на основании предыдущего значения. Чтобы оценить все «шумы и ошибки», необходимо сопоставить модель процесса («формулу», по которой работает ДС) и модель наблюдения (данные с датчиков, устройств и проч.).

На практике, мы почти никогда не можем непосредственно измерить верное состояние (координату, температуру и т.д.) ДС, так как сами приборы, датчики имеют свою погрешность; к тому же, мы можем, например, измерять текущее положение, зная координаты начальной точки и скорость передвижения без непосредственного измерения перемещения по какому-либо датчику или прибору (например, используя спидометр в автомобиле, а не GPS-технологию). Учитывая всё это, необходимо ввести модель наблюдения:

$$z_k = Hx_k + v_k \quad (3)$$

Здесь:

- z – вектор измерения ДС, то есть те данные, непосредственно полученные нами с датчиков различных приборов;
- Hx_k – слагаемое, связывающее вектор состояния x с вектором измерения z ; Если предположить, что мы имеем прибор, измеряющий координату (какая-то из GPS-технологий) и спидометр, то:

$$H = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix};$$

- v_k – вектор ошибок измерений, как и для модели процесса, имеющее нормальное распределение с нулевым математическим ожиданием, связанный с ковариационной матрицей R .

2.3 Ковариационные матрицы

На каждом шаге фильтр Калмана рекурсивно проводит оценку текущего состояния ДС, используя предыдущие данные моделей измерения и процесса. От неопределённости предыдущего состояния зависит будет ли фильтр приближён к процессу или к измерениям. Таким образом, необходимо введение ковариационных матриц P , Q и R , обозначающие, какая связь между случайными величинами.

2.3.1 Матрица состояния P

P – ковариационная матрица, порядок которой равен размеру вектора x . Эта матрица, самостоятельно обновляющаяся фильтром, отражает «уверенность» фильтра в оценке измерений. Для того, чтобы матрица «работала» и самостоятельно обновлялась, необходимо установить начальные значения.

Так как в текущей ДС нам могут быть неизвестны значения ковариации между переменными, то их можно установить, как нули. P сама обновит эти значения после этого. Дисперсию же можно установить как $3\sigma = x_{max} \Rightarrow \sigma^2 = \left(\frac{x_{max}}{3}\right)^2$.

2.3.2 Матрица шума R .

R – ковариационная матрица, порядок которой равен размеру вектора наблюдения, представляющая собой ковариацию между ошибками и истинным

значением (то есть шумов). Обычно, допускается, что измерения не связаны друг с другом; таким образом, R – диагональная матрица. Ненулевые значения («приборные ошибки») можно найти в справочных материалах к датчикам и приборам, или же заполнить так же, как и для матрицы P .

2.3.3 Матрица ошибки модели Q

Q – ковариационная матрица, порядок которой равен размеру вектора состояния. Она указывает, на какие переменные состояния будут в первую очередь влиять ошибки модели и неучтённые факторы. Например, Q будет показывать, как отреагирует фильтр на расхождение между измерением и моделью во время прохождения «неровности на дороге». Матрица Q выбирается так, чтобы наибольшее значение соответствовало самому высокому порядку производной. Встроенная библиотека `filterpy` может самостоятельно рассчитывать матрицу Q . В простейшем случае, с учётом порядка наивысшей производной (для аналитических расчётов), можно предположить, что:

$$Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}.$$

3 РЕАЛИЗАЦИЯ ФИЛЬТРА КАЛМАНА НА ЯЗЫКЕ PYTHON С ИСПОЛЬЗОВАНИЕМ ВСТРОЕННОЙ БИБЛИОТЕКИ FILTERPY

3.1 Предварительные расчёты и реализация модели

В Python есть встроенная библиотека filterpy [6], позволяющая реализовать фильтр Калмана без необходимости писать код самого фильтра самостоятельно.

Чтобы не нагружать код, в этом примере предлагается использовать фильтр Калмана для одномерного случая. То есть необходимо оценить текущее положение x с помощью, например, GPS, который имеет некоторую погрешность.

Так как для реализации фильтра Калмана важно знать используемые модели наблюдения и состояния, а также значения «погрешностей», для курсовой работы используется генерация значений непосредственно в программе, а не с использованием dataset-а. График сравнения оценки фильтра и истинных значений представлен на рисунке 1.

Модель системы будет включать в себя перемещение x , скорость (первую производную v) и ускорение a . Таким образом, матрица F имеет следующий вид:

$$F = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 1 & \Delta t \\ 0 & 0 & 1 \end{bmatrix};$$

Из «условий» задачи единственный прибор – GPS-датчик. А значит:

$$H = [1 \quad 0 \quad 0];$$

Взяв за начальное состояние вектор $x=[0 \ 0 \ 0]$, необходимо задать ковариационную матрицу P для начального положения, так как неизвестно реальное значение вектора x (для этого на главной диагонали установим большие значения):

$$P = \begin{bmatrix} 20 & 0 & 0 \\ 0 & 20 & 0 \\ 0 & 0 & 20 \end{bmatrix}.$$

3.2 Реализация на Python

Код на языке Python представлен на картинках 2-6 [9]. Реализация фильтра Калмана на языке Python представлена в приложении А.

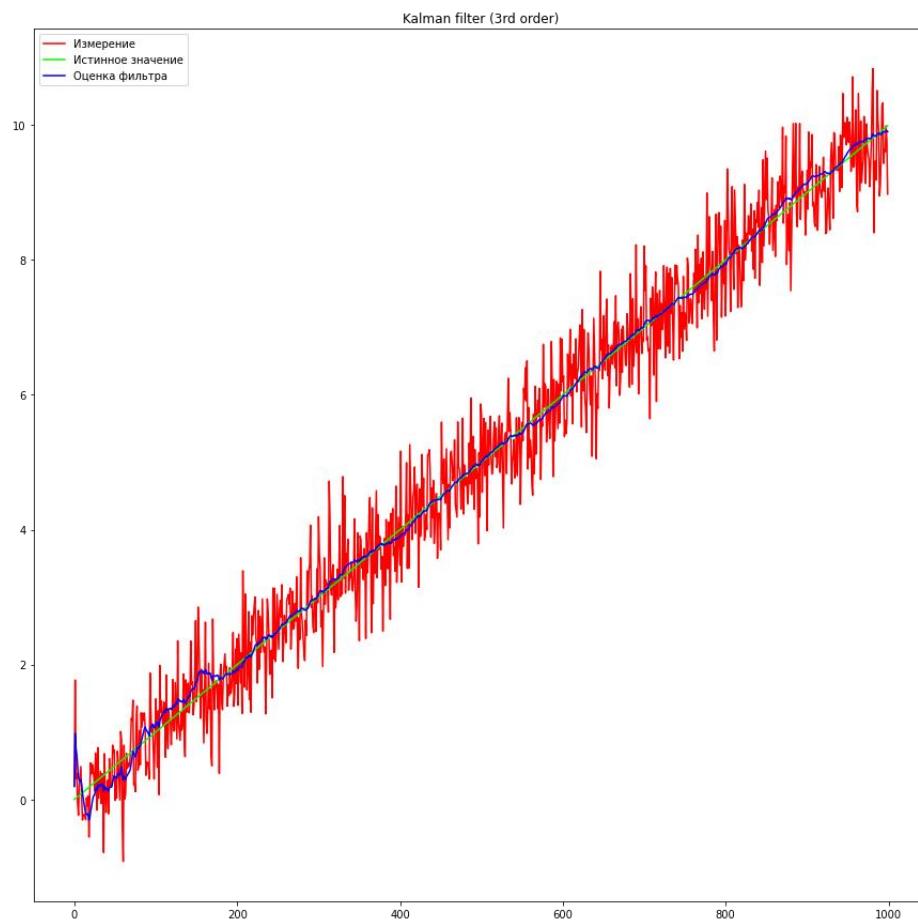


Рис 1. – Результат работы фильтра Калмана

```

import numpy as np
import numpy.random as nr

def data_generation(count, deviation, dt):#dt - время (изменение времени с предыдущего состояния до текущего)
    # Шум с нормальным распределением. мат. ожидание = 0, среднеквадратичное отклонение = deviation, count - форма возвращаемого массива
    noise = nr.normal(loc = 0.0, scale = deviation, size = count)#loc - мат.ожидание
    path = np.zeros((3, count))# Fx
    x = 0
    v = 1.0
    a = 0.0
    for i in range(1, count):
        x += v * dt + (a * dt ** 2) / 2.0 #x=x0+vt+at^2/2
        v += a * dt
        a = a
        path[0][i] = x
        path[1][i] = v
        path[2][i] = a

    data = path[0] + noise

    return path, data # Истинное значение и данные "датчика" с шумом

```

Рис 2. – Код фильтра Калмана, генерация значений «датчика»

```

!pip install filterpy

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting filterpy
  Downloading filterpy-1.4.5.zip (177 kB)
    |#####| 177 kB 5.0 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from filterpy) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from filterpy) (1.7.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (from filterpy) (3.2.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->filterpy) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->filterpy) (1.4.4)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib->filterpy) (3.0.9)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib->filterpy) (0.11.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib->filterpy) (4.1.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib->filterpy) (1.15.0)
Building wheels for collected packages: filterpy
  Building wheel for filterpy (setup.py) ... done
  Created wheel for filterpy: filename=filterpy-1.4.5-py3-none-any.whl size=110474 sha256=8e34af803fb042b904f09afb5695092b604ded537bd8f32217cc95e8de29d01c
  Stored in directory: /root/.cache/pip/wheels/ce/e0/ee/ae/a2b3c5caab3418c1ccd8c4de573d4cbe13315d7e8b0a55fbc2
Successfully built filterpy
Installing collected packages: filterpy
Successfully installed filterpy-1.4.5

```

Рис 3. – Код фильтра Калмана, установка библиотеки filterpy

```

import filterpy.kalman
import filterpy.common
import matplotlib.pyplot as plt
import numpy as np
import numpy.random

dt = 0.01 # дельта времени
disp = 0.5 # среднеквадратичное отклонение
dataNoise = 1e-4 # погрешность ("шум") модели

# Моделирование данных датчиков
coord, measure = data_generation(1000, disp, dt)#пусть будет 1000 измерений

# Создаём объект KalmanFilter
filter = filterpy.kalman.KalmanFilter(dim_x=3, dim_z=1) # Размер вектора состояния x и вектора измерений z

# F - матрица процесса - размер dim_x на dim_x - 3x3
filter.F = np.array([[1, dt, (dt**2)/2],
                    [0, 1.0, dt],
                    [0, 0, 1.0]])

```

Рис 4. – Код фильтра Калмана, сам фильтр непосредственно (часть 1)

```

# Матрица наблюдения H - dim_z на dim_x - 1x3
filter.H = np.array([[1.0, 0.0, 0.0]])

# Ковариационная матрица Q ошибки модели
filter.Q = filterpy.common.Q_discrete_white_noise(dim=3, dt=dt, var=dataNoise)

# Ковариационная матрица R ошибки измерения - 1x1
filter.R = np.array([[disp*disp]])

# Начальное состояние. x0
filter.x = np.array([0.0, 0.0, 0.0])

# Ковариационная матрица P для начального состояния
filter.P = np.array([[20.0, 0.0, 0.0],
                    [0.0, 20.0, 0.0],
                    [0.0, 0.0, 20.0]])

filteredState = [] #вектор оценки состояния
stateCovarianceHistory = [] #вектор обновления ковариации

```

Рис 5. – Код фильтра Калмана, сам фильтр непосредственно (часть 2)

```

filteredState = [] #вектор оценки состояния
stateCovarianceHistory = [] #вектор обновления ковариации
# Обработка данных
for i in range(0, len(measure)):
    z = [ measure[i] ]           # Вектор измерений
    filter.predict()             # Этап предсказания
    filter.update(z)             # Этап коррекции

    filteredState.append(filter.x)
    stateCovarianceHistory.append(filter.P)

filteredState = np.array(filteredState)
stateCovarianceHistory = np.array(stateCovarianceHistory)

```

```

# Визуализация
plt.figure(figsize=(20, 20))
plt.title("Kalman filter (3rd order)")
plt.plot(measure, label="Измерение", color="#FF0000")
plt.plot(coord[0], label="Истинное значение", color="#00FF00")
plt.plot(filteredState[:, 0], label="Оценка фильтра", color="#0000FF")
plt.legend()

```

Рис 6. – Код фильтра Калмана, сам фильтр непосредственно (часть 3) и визуализация

4 НЕДОСТАТКИ И МОДИФИКАЦИИ ФИЛЬТРА КАЛМАНА

Несмотря на высокую эффективность, фильтр Калмана имеет некоторые недостатки:

- Он эффективен лишь для линейных моделей. Для нелинейных моделей трудно представить нормальное распределение шумов. При этом для линейных фильтров фильтр Калмана наилучший (средний квадрат ошибки минимален).

- Ошибки оценки всё ещё актуальны, так как в природе не существует «чисто белого шума», на анализе которого и работает фильтр Калмана.

Несмотря на эти недостатки, фильтр Калмана используется повсеместно [7].

Для использования фильтра для нелинейных моделей используются его изменённые версии:, которые не исследуются в этой курсовой работе:

- Extended Kalman Filter (EKF) — расширенный фильтр Калмана. Этот подход строит линейное приближение модели на каждом шаге. Для этого требуется рассчитать матрицу вторых частных производных функции модели, что бывает весьма непросто, для чего часто используются численные методы.

- Unscented Kalman Filter (UKF). Этот подход строит приближение распределения получающегося после нелинейного преобразования при помощи сигма-точек без расчёта производных вообще. Его реализация на исследуемой ДС представлена на рисунках 7-8, а использование - на рисунке 9. Как видно из рисунка 10, фильтра Калмана «без запаха» даёт лишь небольшие различия по сравнению с истинным фильтром Калмана.

Фильтр Калмана «без запаха» позволяет использовать нелинейные модели, так как преобразует матрицы F и H в соответствующие функции $f(x)$ и $h(x)$, использующие так называемые «сигма-точки» [10].

Несмотря на то, что в текущем исследовании на представленной модели UKF даёт более точный результат, для линейных моделей лучше использовать

обыкновенный фильтр Калмана, так как он предполагает наименьшее отклонение предсказаний от истинного значения в случае более перегруженных линейных моделей, так как не существует рекомендованного и точного алгоритма генерации сигма-точек, которые для каждой задачи подбираются экспериментально. Средние квадратичные ошибки фильтра Калмана и UKF (т.к. выбросы подвержены нормальному распределению) получились соответственно равными 1,22% и 1,2%, при этом самые «крупные» ошибки были при первых прогнозах из-за того, что и вектор x , и матрица P не были достоверно известны изначально (рисунки 11, 12).

```
#Фильтр Калмана без запаха
"""dt = 0.01 # дельта времени
disp = 0.5 # среднеквадратичное отклонение
dataNoise = 1e-4 # погрешность ("шум") модели"""

# функция наблюдения - аналог матрицы наблюдения
# Преобразует вектор состояния x в вектор измерений z
def xToZ(x):
    return np.array([x[0]])

# функция процесса - аналог матрицы процесса
def processFunc(x, dt):
    newState = np.zeros(3)
    newState[0] = x[0] + dt * x[1] + ( (dt**2)/2 ) * x[2]
    newState[1] = x[1] + dt * x[2]
    newState[2] = x[2]

    return newState

# Для unscented kalman filter необходимо выбрать алгоритм выбора сигма-точек
sigma_points = filterpy.kalman.JulierSigmaPoints(3, kappa=0)

# Создаём объект UnscentedKalmanFilter
filter_u = filterpy.kalman.UnscentedKalmanFilter(dim_x = 3,
                                                dim_z = 1,
                                                dt = dt,
                                                hx = xToZ,
                                                fx = processFunc,
                                                points = sigma_points)
```

Рис 7. – Код UKF (часть 1)

```
# Ковариационная матрица ошибки модели
filter_u.Q = filterpy.common.Q_discrete_white_noise(dim=3, dt=dt, var=dataNoise)

# Ковариационная матрица ошибки измерения - 1x1
filter_u.R = np.array([[disp*disp]])

# Начальное состояние.
filter_u.x = np.array([0.0, 0.0, 0.0])

# Ковариационная матрица для начального состояния
filter_u.P = np.array([[10.0, 0.0, 0.0],
                       [0.0, 10.0, 0.0],
                       [0.0, 0.0, 10.0]])

filteredState_u = []
stateCovarianceHistory_u = []

for i in range(0, len(measure)):
    z = [ measure[i] ]
    filter_u.predict()
    filter_u.update(z)

    filteredState_u.append(filter_u.x)
    stateCovarianceHistory_u.append(filter_u.P)

filteredState_u = np.array(filteredState_u)
stateCovarianceHistory_u = np.array(stateCovarianceHistory_u)

# Визуализация UKF
plt.figure(figsize=(15,15))
plt.title("Unscented Kalman filter")
plt.plot(measure, label="Измерение", color="#FF0000")
plt.plot(coord[0], label="Истинное значение", color="#00FF00")
plt.plot(filteredState_u[:, 0], label="Оценка фильтра", color="#0000FF")
plt.legend()
```

Рис 8. – Код UKF (часть 2) и визуализация

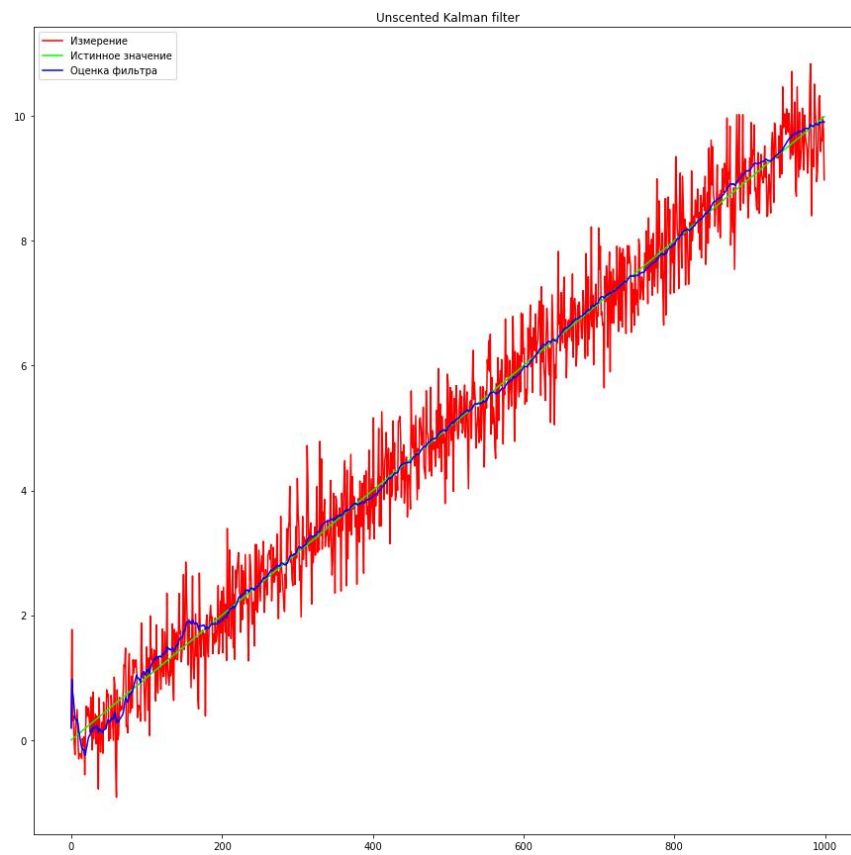


Рис 9. – Результат работы UKF.

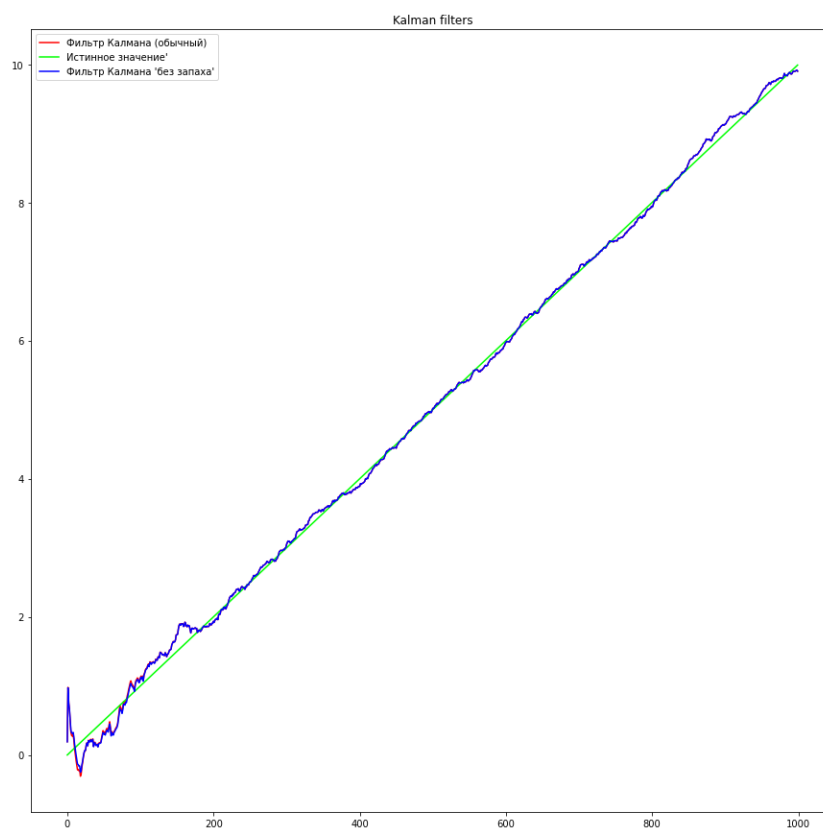


Рис 10. – Сравнение работы фильтров

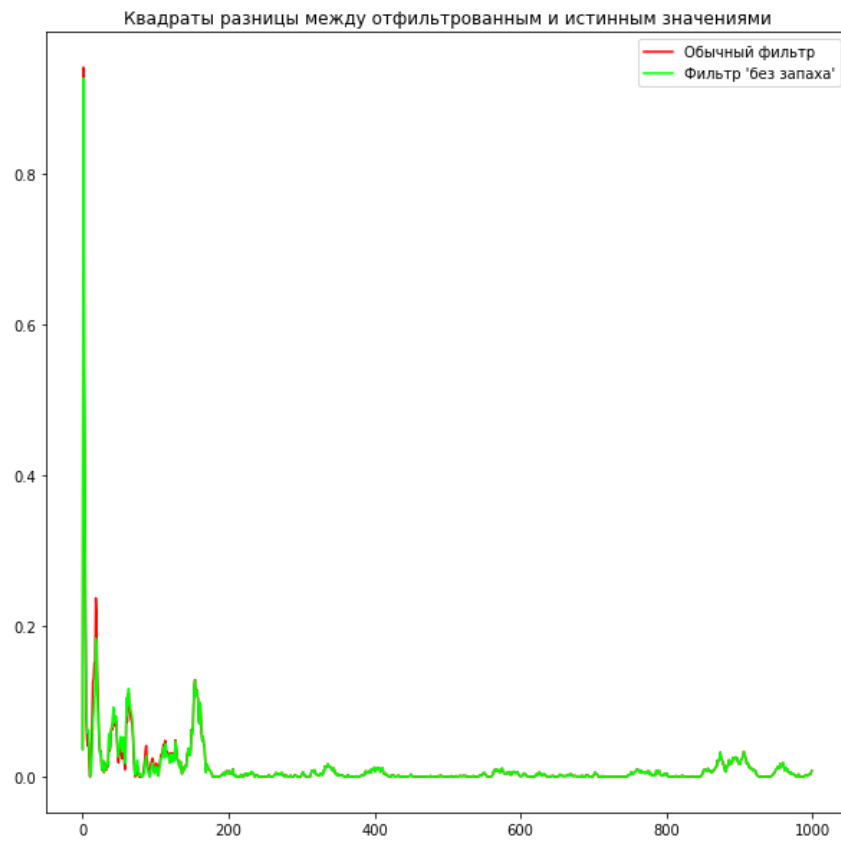


Рис 11. – MSE фильтров (график)

```
#Mean Square Error
def MSE(y_true, y_pred):
    return round((np.power(y_true - y_pred, 2).mean())*100, 2)
```

```
#ошибка фильтра, %
MSE(filteredState[:, 0], coord[0])
```

1.22

```
#ошибка UKF, %
MSE(filteredState_u[:, 0], coord[0])
```

1.2

Рис 12. – MSE фильтров (код и вывод значений в %)

ЗАКЛЮЧЕНИЕ

В курсовой работе был исследован алгоритм работы фильтра Калмана: изучены данные, с которыми он корректно работает, рассмотрены модели состояния и измерения, используемые фильтром. Кроме того, был фильтр Калмана был реализован с помощью библиотеки `filterpy` ЯП Python в среде Google Colaboratory [8].

На данной модели ДС (с измерением лишь координаты x , но при этом содержащей, кроме координаты, ещё и скорость v и ускорение a), «измерения» которой генерируются с помощью `numpy.random` (без использования `dataset-a`) были применены фильтра Калмана (обычный) и «без запаха» (UKF). При этом, средние квадратичные ошибки (MSE) фильтров равны соответственно 1,22% и 1,2%. Несмотря на то, что UKF справился «точнее» с «линейной» задачей, его рекомендуется использовать для нелинейных ДС, а для подобных задач достаточно обычного фильтра, так как при усложнении модели процесса, алгоритм вычисления «сигма-точек», с помощью которого работает UKF, будет усложняться, и, как результат, будет приводить к более «нечёткой» фильтрации. В любом случае, средняя ошибка в диапазоне до 1,5 процента говорит о высокой точности фильтрации (так как большая MSE лишь при первых прогнозах из-за изначальной неопределённости вектора состояний x и матрицы состояния P), а значит, фильтры работают корректно.

Курсовая работа помогла приобрести опыт в анализе и фильтрации данных, дала возможность получить знания о фильтре Калмана и позволила улучшить навыки работы с Python, а также поспособствовала получению умений по оформлению текста работы по ГОСТ 7.32-2017, что упростит дальнейшую работу с отчётами по научно-исследовательским работам.

Таким образом, цель работы была достигнута в установленный срок.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Анализ данных – основы и терминология / Хабр [электронный ресурс]. - URL: <https://habr.com/ru/post/352812/> (дата обращения: 29.10.2022).
2. ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст). / КонсультантПлюс [электронный ресурс]. - URL: http://www.consultant.ru/document/cons_doc_LAW_292293/ (дата обращения: 22.10.2022).
3. Фильтр Калмана / Википедия [электронный ресурс]. – URL: https://ru.wikipedia.org/wiki/Фильтр_Калмана (дата обращения: 29.10.2022)
4. Фильтр Калмана – это легко / Timofeev.ru [электронный ресурс]. – URL: <https://temofeev.ru/info/articles/filtr-kalmana-eto-legko/> (дата обращения: 29.10.2022)
5. Фильтр Калмана / Хабр [электронный ресурс]. – URL: <https://habr.com/ru/post/166693/> (дата обращения: 05.11.2022)
6. FilterPy / FilterPy 1.4.4 documentation [электронный ресурс]. – URL: <https://filterpy.readthedocs.io/en/latest/> (дата обращения: 12.11.2022)
7. Калмоновская фильтрация / BaseGroup Labs [электронный ресурс]. URL: <https://basegroup.ru/community/articles/kalmanfilter> (дата обращения: 12.11.2022)
8. Добро пожаловать в Google Colab! / Google Colaboratory [электронный ресурс]. – URL: https://colab.research.google.com/#scrollTo=5fCEDCU_qrC0 (дата обращения: 12.11.2022)
9. Фильтр.ipynb / Google Colaboratory [электронный ресурс]. – URL: https://colab.research.google.com/drive/1GCojAG5CT_DJkxOhYIC_pKm1uDG3x-ip?authuser=1#scrollTo=r2d1kMAxma1E (дата обращения: 12.11.2022)
10. Фильтр Калмана – Kalman Filter / ВикибриФ [электронный ресурс]. – URL: https://ru.wikibrief.org/wiki/Kalman_filter (дата обращения: 19.11.2022)

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

(обязательное)

```
import numpy as np
import numpy.random as nr

def data_generation(count, deviation, dt):#dt - время (изменение вре-
мени с предыдущего состояния до текущего)
    # Шум с нормальным распределением. мат. ожидание = 0, среднеквадратич-
ное отклонение = deviation, count - форма возвращаемого массива
    noise = nr.normal(loc = 0.0, scale = devia-
tion, size = count)#loc - мат.ожидание
    path = np.zeros((3, count))# Fx
    x = 0
    v = 1.0
    a = 0.0
    for i in range(1, count):
        x += v * dt + (a * dt ** 2) / 2.0 #x=x0+vt+at^2/2
        v += a * dt
        a = a
        path[0][i] = x
        path[1][i] = v
        path[2][i] = a

    data = path[0] + noise

    return path, data # Истинное значение и данные "датчика" с шумом

# Реализация фильтра
import filterpy.kalman
import filterpy.common
import matplotlib.pyplot as plt
import numpy as np
import numpy.random

dt = 0.01 # дельта времени
disp = 0.5 # среднеквадратичное отклонение
dataNoise = 1e-4 # погрешность ("шум") модели

# Моделирование данных датчиков
coord, measure = data_generation(1000, disp, dt)#пусть будет 1000 измерений

# Создаём объект KalmanFilter
```

```

filter = filterpy.kalman.KalmanFilter(dim_x=3, dim_z=1) # Размер век-
тора состояния x и вектора измерений z

# F - матрица процесса - размер dim_x на dim_x - 3x3
filter.F = np.array([ [1, dt, (dt**2)/2],
                      [0, 1.0, dt],
                      [0, 0, 1.0]])

# Матрица наблюдения H - dim_z на dim_x - 1x3
filter.H = np.array([[1.0, 0.0, 0.0]])

# Ковариационная матрица Q ошибки модели
filter.Q = filterpy.common.Q_dis-
crete_white_noise(dim=3, dt=dt, var=dataNoise)

# Ковариационная матрица R ошибки измерения - 1x1
filter.R = np.array([[disp*disp]])

# Начальное состояние. x0
filter.x = np.array([0.0, 0.0, 0.0])

# Ковариационная матрица P для начального состояния
filter.P = np.array([[20.0, 0.0, 0.0],
                    [0.0, 20.0, 0.0],
                    [0.0, 0.0, 20.0]])

filteredState = [] #вектор оценки состояния
stateCovarianceHistory = [] #вектор обновления ковариации
# Обработка данных
for i in range(0, len(measure)):
    z = [ measure[i] ] # Вектор измерений
    filter.predict() # Этап предсказания
    filter.update(z) # Этап коррекции

    filteredState.append(filter.x)
    stateCovarianceHistory.append(filter.P)

filteredState = np.array(filteredState)
stateCovarianceHistory = np.array(stateCovarianceHistory)

# Визуализация
plt.figure(figsize=(20, 20))
plt.title("Kalman filter (3rd order)")
plt.plot(measure, label="Измерение", color="#FF0000")
plt.plot(coord[0], label="Истинное значение", color="#00FF00")

```

```

plt.plot(filteredState[:, 0], label="Оценка фильтра", color="#0000FF")
plt.legend()

#Фильтр Калмана без запаха
"""dt = 0.01                                # дельта времени
disp = 0.5                                # среднеквадратичное отклонение
dataNoise = 1e-4                            # погрешность ("шум") модели"""

# Функция наблюдения - аналог матрицы наблюдения
# Преобразует вектор состояния x в вектор измерений z
def xToZ(x):
    return np.array([x[0]])

# Функция процесса - аналог матрицы процесса
def processFunc(x, dt):
    newState = np.zeros(3)
    newState[0] = x[0] + dt * x[1] + ( (dt**2)/2 ) * x[2]
    newState[1] = x[1] + dt * x[2]
    newState[2] = x[2]

    return newState

# Для unscented kalman filter необходимо выбрать алгоритм выбора сигма-точек
sigma_points = filterpy.kalman.JulierSigmaPoints(3, kappa=0)

# Создаём объект UnscentedKalmanFilter
filter_u = filterpy.kalman.UnscentedKalmanFilter(dim_x = 3,
                                                    dim_z = 1,
                                                    dt = dt,
                                                    hx = xToZ,
                                                    fx = processFunc,
                                                    points = sigma_points)

# Ковариационная матрица ошибки модели
filter_u.Q = filterpy.common.Q_dis-
crete_white_noise(dim=3, dt=dt, var=dataNoise)

# Ковариационная матрица ошибки измерения - 1x1
filter_u.R = np.array([[disp*disp]])

# Начальное состояние.
filter_u.x = np.array([0.0, 0.0, 0.0])

```



```

# Ковариационная матрица для начального состояния
filter_u.P = np.array([[10.0, 0.0, 0.0],
                        [0.0, 10.0, 0.0],
                        [0.0, 0.0, 10.0]])

filteredState_u = []
stateCovarianceHistory_u = []

for i in range(0, len(measure)):
    z = [ measure[i] ]
    filter_u.predict()
    filter_u.update(z)

    filteredState_u.append(filter_u.x)
    stateCovarianceHistory_u.append(filter_u.P)

filteredState_u = np.array(filteredState_u)
stateCovarianceHistory_u = np.array(stateCovarianceHistory_u)

# Визуализация UKF
plt.figure(figsize=(15,15))
plt.title("Unscented Kalman filter")
plt.plot(measure, label="Измерение", color="#FF0000")
plt.plot(coord[0], label="Истинное значение", color="#00FF00")
plt.plot(filteredState_u[:, 0], label="Оценка фильтра", color="#0000FF")
plt.legend()

# Сравнение фильтров
plt.figure(figsize=(15, 15))
plt.title("Kalman filters")
plt.plot(filteredState[:, 0], label="Фильтр Калмана (обычный)", color="#FF0000")
plt.plot(coord[0], label="Истинное значение", color="#00FF00")
plt.plot(filteredState_u[:, 0], label="Фильтр Калмана 'без запаха'", color="#0000FF")
plt.legend()

plt.figure(figsize=(10, 10))
plt.title("Квадраты разницы между отфильтрованным и истинным значениями")
plt.plot(np.power(filteredState[:, 0]-coord[0], 2), label="Обычный фильтр", color="#FF0000")
plt.plot(np.power(filteredState_u[:, 0]-coord[0], 2), label="Фильтр 'без запаха'", color="#00FF00")
plt.legend()

```

```


#Mean Square Error
def MSE(y_true, y_pred):
    return round((np.power(y_true - y_pred, 2).mean())*100, 2)


#ошибка фильтра, %
MSE(filteredState[:, 0], coord[0])
#->1,22

#ошибка UKF, %
MSE(filteredState_u[:, 0], coord[0])
#->1,2


```


ПРИЛОЖЕНИЕ Б


**АНТИПЛАГИАТ**
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ


Реестр
отечественного ПО

Горный университет

СТУДЕНТ 
s210312@stud.spmi.ru

 МЕНЮ

ru 


ГЛАВНАЯ / КАБИНЕТ СТУДЕНТА / РЕЗУЛЬТАТЫ ПРОВЕРКИ

Оригинальность89,77%


Заемствования10,23%



Цитирования0%


Самоцитирования0%


 Проверено: 59,53% текста документа, исключено из проверки: 40,47% текста документа. Некоторые разделы были исключены пользователем при загрузке документа. Более подробная информация содержится на вкладке "Структура документа".


КРАТКИЙ ОТЧЕТ


 РАСПЕЧАТАТЬ

 ВЫГРУЗИТЬ 


 Свойства документа

 Структура документа

 Текстовые метрики **NEW**

 Параметры проверки

Имя исходного файлаИСТ-21-1 Темнов А.М.ТОИ Курсовая работа Исследование фильтра Калмана.pdf

Авторы документа ТемновАлексей Максимович

Название документаИсследование фильтра Калмана

Тип документаКурсовая работа

РЕДАКТИРОВАТЬ СВОЙСТВА

Рис 13. – Проверка на «Антиплагиат»