Homework 5: Matrix and Kalman Filter

Authors: Qifan Zhang zhangqf@shanghaitech.edu.cn.

Supervised, proofread, edited and approved by Prof. Haipeng Zhang zhanghp@shanghaitech.edu.cn.

Proofread and calibrated by Ziqi Gao gaozq@shanghaitech.edu.cn.

Release Date: May 19, 2020

Deadline: 23:59:00 Jun 4 (Thu.), 2020, China Standard Time (UTC+8:00)

Last Modified: May 16, 2020

Get Started

To get started, please simply fork this GitLab repository and follow the structure and submissions guidelines below. **Remember to change your repo into a private one before you commit to it.**

Repository Structure

Readme.md/pdf

Problem description and requirements

KF_np.py

A basic template for Task 1. Some naïve test cases are included for you to test your implementation.

KF_self_impl.py

A basic template for tasks except Task 1, including bonus. Some naïve test cases are included for you to test your implementation.

Submit

You should check in `KF_np.py` and `KF_self_impl.py` to GitLab.

First, make a commit and push your own `KF_np.py`. From the root of this repo, run:

```
git add KF_np.py KF_self_impl.py
git commit -m"{Your commit message}"
git push origin master
```

Then add a tag to request grading on your current submission:

```
git tag {tagname} && git push origin {tagname}
```

Beware that all of your tag names should be distinguished among one homework repo. Therefore, remember to use **a new tag name** `{tagname}` in each submission.

Every submission will create a new GitLab issue, where you can track the progress.

Regulations

- No late submission will be accepted
- In Task 1, you could only import:
  - NumPy (via `import numpy as np`)
    - **Do not modify** `import numpy as np`, or your assignment will be graded as 0 point.
  - All the built-in modules
- You could only import built-in modules in all the tasks except Task 1.
- You will have 30 chanced of grading (i.e. `git tag`) in this homework (Homework 4.1 and Homework 4.2 will be counted respectively). You are able to require at most 10 times every 24 hours. If you hand in more than 30 times, each extra submission will lead to 10% deduction.
- **Hard code is strictly forbidden.** Once found, your score of this homework will be set as 0.
- We enforce academic integrity strictly. If you participate in any from of cheating, you will fail this course immediately. You can view full edition on Piazza.
- If you have any question about this homework, please ask on Piazza first.

Overall Description

Recently, Mr. Sailboat has switched his research interest from Digital Image Processing into Autopilot System for automobile. Multi-Sensor Fusion (MSF) is quite an issue for monitoring status of automobiles and Kalman Filter is one of the common algorithms to complete this task. Therefore, Mr. Sailboat now requires you to implement a self-defined matrix class and then build a Kalman Filter based on it as part of the autopilot system.

Brief Introduction to Kalman Filter

Measurements, a.k.a. observations, are always companied with noises, which are random but may follow a kind of distribution. For one state, we may have several measurements with different (kinds of) noises. Kalman Filter is a way to combine them together and narrow down the noises.

Kalman Filter solves this problem with following assumptions:

- All the noises are Gaussian Noises, i.e. all the noises are under Normal Distribution with zero mean

For example, let us denote an observation $x \in R^{n\times 1}$ with noise parameters, which is called a **covariance matrix**, $P \in R^{n\times n}$. This means that the real current state is $x + w$ where $w \in \mathcal{N}(0, P)$. Kalman Filter will narrow down P in order to make the bias w as small as possible so that the observation becomes precise.

Please attend discussion in Week 14 in order to get a complete view of Kalman Filter.

Structure of Kalman Filter

A Kalman Filter is composed of two parts: prediction and observation.

Prediction

Prediction part is used to predict next state with predefined state transition relation and outside control.

Also, we will update covariance matrix part to indicate *uncertainty* for the new state.

For state transition:

$x_{k+1} = Fx_k + Bu_k$

For covariance transition

$P_{k+1} = FP_kF^T + Q$

where:

- x_k and P_k indicate the current state and its related covariance matrix, x_{k+1} and P_{k+1} indicate next state and its related covariance matrix.
- u_k is the control vector for current state
- F is the transition matrix from the current state to next state
- B is the transition matrix from the current control vector to next state
- Q is the external noise covariance matrix, i.e. the noise covariance matrix of u_k.

Notice that F, B and Q may vary with different states. Of course, it could remain static for every state.

For simplicity, **all F, B and Q remain static in Homework 4.**

Update via Observation

After prediction, we need to do some *calibration* for our current state with some extra *observations*, these observations may be just in the same format with states, or it may need some transition to make sense.

There are 3 steps for update:

$K = P_kH_k^T(H_kP_kH_k^T + R)^{-1}$

$\hat{x_k} = x_k + K(z_k - Hx_k)$

$\hat{P}_k = P_k - KH_kP_k$

where:

- x_k and P_k are what we have gotten from `Prediction` stage
- z_k is observation
- H_k is transition matrix to transit x_k into the same scale with observation z_k
- R_k is the covariance matrix of observation noise

Task 1: Kalman Filter (NumPy Version) (5%)

In Task 1, you are required to implement a Kalman Filter with the help of NumPy.

You should implement your codes in `KF_np.py`.

Task 1.1: Prediction, NumPy Version

In this part, you are required to implement `predict` method of class `KF`.

Input and Output

Input

- x_k: `numpy.array`, size: $n \times 1$
- P_k: `numpy.array`, size: $n \times n$
- u_k: `numpy.array`, size: $m \times 1$
- F: `numpy.array`, size: $n \times n$
- B: `numpy.array`, size: $n \times m$
- Q: `numpy.array`, size: $n \times n$

Output

- x_{k+1}: `numpy.array`, size: $n \times 1$
- P_{k+1}: `numpy.array`, size: $n \times n$

Error Handling

Mr. Sailboat is such a careless guy that there may some errors among inputs he offers.

Possible errors:

dtypeError

In order to ensure precision, Mr. Sailboat requires `dtype` of all the inputs should be `numpy.float64`. All other `dtype`s are unacceptable.

Therefore, please check whether `dtype` of each input is `numpy.float64`. If not, raise exception `dtypeError`.

SizeUnmatchedError

Inputs must follow the provided size strictly. If not, raise exception `SizeUnmatchedError`.

All the given matrices will have **at most one** of the above 2 exceptions.

Reminder

- `dtype` of all the inputs are `numpy.float64`
- `dtype` of your output should be `numpy.float64`. You will fail related test cases if `dtype` of your output is **not** `numpy.float64`.

Task 1.2: Update via Observation, NumPy Version

In this part, you are required to implement `update` method of class `KF`.

Input and Output

Input

- x_k: `numpy.array`, size: $n \times 1$
- P_k: `numpy.array`, size: $n \times n$
- z_k: `numpy.array`, size: $k \times 1$
- H: `numpy.array`, size: $k \times n$
- R: `numpy.array`, size: $k \times k$

Output

- $\hat{x_k}$: `numpy.array`, size: $n \times 1$
- $\hat{P_k}$: `numpy.array`, size: $n \times n$

Error Handling

Mr. Sailboat is such a careless guy that there may some errors among inputs he offers.

Possible errors:

dtypeError

In order to ensure precision, Mr. Sailboat requires `dtype` of all the inputs should be `numpy.float64`. All other `dtype`s are unacceptable.

Therefore, please check whether `dtype` of each input is `numpy.float64`. If not, raise exception `dtypeError`.

SizeUnmatchedError

Inputs must follow the provided size strictly. If not, raise exception `SizeUnmatchedError`.

All the given matrices will have **at most one** of the above 2 exceptions.

Reminder

- `dtype` of all the inputs are `numpy.float64`
- `dtype` of your output should be `numpy.float64`. You will fail related test cases if `dtype` of your output is **not** `numpy.float64`.

Task 2: Matrix Input and Output (20%)

In this task, you need to implement basic inputs and outputs for the self-defined class `Matrix`.

Task 2.1: Load the Matrix in

In order to do matrix operation, you first need to load it in.

Normally, we represent an $m \times n$ matrix with a sequence number in the size of $m \times n$. However, Mr. Sailboat observed that there are many duplicated values in matrices used in Kalman Filter. If we use normal way of representation, many numbers inside the $m \times n$ sequence will be redundant, which is a waste of memory.

In order to solve this waste, Mr. Sailboat proposed a better way: **sparse representation of Matrix**.

Mr. Sailboat will give you an expression in the following format below using EBNF form.

```
ValueKey = {blank}, "[", {blank}, Value, {blank}, ";", {blank}, Keys,
{blank}, "]", {blank} | {blank}, "[", {blank}, "]", {blank};
Keys = {blank}, "[", {blank}, "]", {blank} | {blank}, "[", Key, {blank}, ";",
Keys, {blank}, "]", {blank};
Key = {blank}, "(", {blank}, Value, {blank}, ",", {blank}, Value, {blank},
{blank}, ")", {blank};
Value = int | float | complex;
blank = " " | "\n" | "\r" | "\t";
```

Above is strict definition for hyper-humans to understand without ambiguity.

In human language, the expression is in the format of:

```
[[value1; [key1; key2; ...]; [value2; [key1; key2; ...]]; ...]
```

where for each `key` (i.e. `key1`, `key2` and so on in the expression above), it is in the format:

```
key = (row, col)
```

where `row` and `col` represents row and column number of the value.

For each `[valuei; [key1; key2; ...]]`, every indicated element are assigned with the value `valuei`.

In the other word, suppose `[valuei; [key1; key2; ...]` is part of sparse expression for matrix `A`.

```
A[key1.row][key1.col] = A[key2.row][key2.col] = ... = valuei
```

Keys in the whole expression are all different. In the other word, there will be no overwriting for all the elements in the matrix.

After all sparse value assignment is done and there are some elements are assigned with values, they will be assigned with 0 (**an integer**) by default.

Notice that redundant blanks may appear anywhere inside an expression.

If you are still confused about definition of sparse representation and its expression, please read the appendix, take a view of given test cases and take discussion of Week 12 for more details.

Error Handling

Normally, a Computer Science guy should be much more considerate and careful than others. However, Mr. Sailboat is such a careless guy that lags down the average. As you have observed above, Mr. Sailboat may give an invalid sparse representation expression. If you find a mistake, you need to raise an exception to add a piece of evidence that Mr. Sailboat should quit Computer Science major.

All possible exceptions are listed below:

MatrixKeyError

For a valid key, it should meet the following requirements:

- Values of each key should first be an integer.
- The number of each key should be exactly 2.
- Indexing should be within the range of the matrix.

Mr. Sailboat defines that **both row number and column number starts with zero.**

Also, like `list` indexing in Python, **negative row number (or column number) means count the row (or column) from the last one.**

You should judge whether the values are valid. If not, please raise exception `MatrixKeyError`.

MatrixValueError

Mr. Sailboat requires that each element in any matrix should be either a `float` or an `integer`. `complex` or other strange data types are not accepted by our class `Matrix`.

You should judge whether the given matrix satisfies this requirement. If not, please raise `MatrixValueError` exception.

MatrixSizeError

Of course, matrix size (i.e. `m` and `n`) should be `postive integer`s.

If not, please raise `MatrixSizeError` exception.

All the given matrices will have **at most one** of the above 3 exceptions.

Input and Output

Input

- `m`: row number of the matrix. It should be a **positive** integer.
- `n`: column number of the matrix. It should be a **positive** integer.
- `expr`: sparse representation expression of the matrix. It should be string.

Output

There will be no output for this task.

Task 2.2: Print out the Matrix

In this task, you should implement your class `KF` in order to make it possible to print the matrix out via function `print()`.

The printing format follows the format defined below using EBNF form.

```
Matrix = "[" Rows "]" | "[" "]" ;
Rows = Row | Row ";" Rows ;
Row = element | element "," Row ;
element = int | float;
```

Input and Output

Input

There will be no input

Output

You should output a string following the format defined above.

Notice that **all the float numbers must keep 5 digits**. Please use `'%.5f' % {var}` or function `round()` to keep digits in order to avoid possible errors.

Task 3: Matrix Operations (15%)

In this task, you need to implement basic operations for the self-defined class `Matrix`.

Task 3.1: Matrix Addition and Subtraction

Implement operator `+` and `-` for class `Matrix`.

Error Handling

ObjectError

Two objects for addition and subtraction operation should be both class `Matrix`. Otherwise, raise exception `ObjectError`.

SizeUnmatchedError

If 2 matrices does not have the same size, please raise exception `SizeUnmatchedError`.

Input and Output

Input

`other`: class `Matrix`

Output

`res`: class `Matrix`

Task 3.2: Matrix Multiplication

Implement operator `*` for class `Matrix`.

There are 2 circumstances for matrix multiplication:

- Multiplication between 2 matrices
- Multiplication between one matrix and a scalar (`int` or `float`)

Error Handling

ObjectError

Two objects for addition and subtraction operation should be:

- both classes `Matrix`
- one class `Matrix` and one scalar (`int` or `float`)

Otherwise, raise exception `ObjectError`.

SizeUnmatchedError

If sizes of the 2 matrices do not match (for $A * B$, $A \in R^{m \times n}$ and $A \in R^{n \times k}$), please raise `SizeUnmatchedError` exception.

Input and Output

Input

`other`: class `Matrix` or scalar (`int` or `float`)

Output

`res`: class `Matrix`

Task 3.3: Matrix Scalar Division

Implement operator `/` for class `Matrix`.

Error Handling

ObjectError

The scalar should be an `integer` or a `float`. Otherwise, please raise exception `ObjectError`.

Input and Output

Input

`other`: scalar (`int` or `float`)

Output

`res`: class `Matrix`

Task 3.4: Matrix Equalization

Implement operator `==` for class `Matrix`.

Error Handling

ObjectError

The compared object should also be class `Matrix`. Otherwise, please raise exception `ObjectError`.

Input and Output

Input

`other`: class `Matrix`

Output

`isEqual`: `bool`

Task 3.5: Matrix Transpose

Implement method `transpose` for class `Matrix`.

Input and Output

Input

No input for this task

Output

`res`: class `Matrix`

Task 4: Advanced Matrix Operations (20%)

Task 4.1: Matrix Determinant

Implement `det()` method of class `Matrix`.

Error Handling

NotSquaredError

A matrix will not have determinant unless it is a squared matrix.

Input and Output

Input

No input for this task

Output

`det`: scalar. Answers within 0.1% error rate will be graded as correct.

Task 4.2: Matrix Inverse

Implement `inv()` method of class `Matrix`.

Error Handling

NotSquaredError

A matrix will not have an inverse matrix unless it is a squared matrix (if we do not take pseudo inverse into account).

If not, raise exception `NotSquaredError`.

NotInvertibleError

A matrix will not have an inverse matrix unless its determinant is nonzero (if we do not take pseudo inverse into account).

If not, raise exception `NotInvertibleError`.

Input and Output

Input

No input for this task

Output

`mat_inv`: class `Matrix`

Task 5: Matrix Indexing (35%)

In this task, you need to implement method `__getitem()__` and `__setitem()__` so that part of the class `Matrix` could get accessed or modified, just like what `list` performs.

The following are tasks you need to do (`x` is a given `Matrix` class):

- `x[i]` returns the `i`-th row (starting at `0`) which is also a matrix
    - If `i` is invalid, raise `IndexSyntaxError`
    - `i` could be negative.

- `x[i,j]` returns the element at the `i`-th row and `j`-th column
  - If `i` or `j` or both are invalid, raise `IndexSyntaxError`.
  - Please return the exact number of the element. If the value to be returned is a `float`, answers within 0.1% error rate will be graded as correct.
- `x[i,j] = k` replaces the element `x[i,j]` by `k` in `x`
  - If `i` or `j` or both are invalid, raise `IndexSyntaxError`.
  - The modified value `k` should still be a scalar (`int` or `float`). If `k` is invalid, raise `MatrixValueError`.
- `x[i] = Matrix(..)` replaces the row `x[i]` by the row `Matrix(..)` in `x` if the lengths of the row `x[i]` and the row `Matrix(..)` are identical, otherwise raise the exception `IndexSyntaxError`
  - If `i` is invalid, raise `IndexSyntaxError`
- `x[start1:stop1:step1,start2:stop2:step2]` returns the matrix `y`, such that `y[i,j]` is the element `x[start1+i*step1,start2+j*step2]` if it exists and `start1+i*step1<stop1`, `start2+i*step2<stop2`
  - If `start1:stop1:step1,start2:stop2:step2` is invalid, raise `IndexSyntaxError`
- `x[start1:stop1:step1,start2:stop2:step2] = Matrix(..)` replaces the matrix `x[start1+i*step1,start2+j*step2]` by `Matrix(..)` in `x` if the number of rows (as well as columns) of `x[start1+i*step1,start2+j*step2]` are `Matrix(..)` are identical, otherwise raise the exception `IndexSyntaxError`
  - If `start1:stop1:step1,start2:stop2:step2` is invalid, raise `IndexSyntaxError`

Task 6: Finally, Kalman Filter! (5%)

Task 6.1: Prediction

Implement `predict()` method of class `KF`.

Input

- x_k: class `Matrix`, size: $n \times 1$
- P_k: class `Matrix`, size: $n \times n$
- u_k: class `Matrix`, size: $m \times 1$
- F: class `Matrix`, size: $n \times n$
- B: class `Matrix`, size: $n \times m$
- Q: class `Matrix`, size: $n \times n$

Output

- x_{k+1}: class `Matrix`, size: $n \times 1$
- P_{k+1}: class `Matrix`, size: $n \times n$

Error Handling

SizeUnmatchedError

Just like Task 1&2, inputs must follow the provided size strictly.

If not, raise exception `SizeUnmatchedError`.

Task 6.2: Update

Implement `update()` method of class `KF`.

Input

- x_k: class `Matrix`, size: $n \times 1$
- P_k: class `Matrix`, size: $n \times n$
- z_k: class `Matrix`, size: $m \times 1$
- H_k: class `Matrix`, size: $n \times n$
- R_k: class `Matrix`, size: $n \times m$

Output

- $\hat{x_k}$: class `Matrix`, size: $n \times 1$
- $\hat{P_k}$: class `Matrix`, size: $n \times n$

Error Handling

SizeUnmatchedError

Just like Task 1&2, inputs must follow the provided size strictly.

If not, raise exception `SizeUnmatchedError`.

Task 7 (Bonus Task): Matrix Postfix Evaluation (10%)

You need to implement the function `postfix_eval` to evaluate a postfix expression.

In this evaluation, there are 4 operations:

- Addition (`+`)
- Subtraction (`-`)
- Multiplication (`*`)
- Division (`/`)

The whole expression will be given by a list with elements inside it.

Input and Output

Input

`expr`: a list with tokens inside

Output

You should output a class `Matrix` with evaluation of the matrix.

There will be no exception in this task.