hw7

Coding the Matrix, Summer 2013

Please fill out the stencil file named "`hw7.py`". While we encourage you to complete the Ungraded Problems, they do not require any entry into your stencil file.

Problem 1: Write a procedure `basis(vlist)` with the following spec:

- *input:* a list $vlist$ of Vecs
- *output:* a list of linearly independent Vecs that span the same space as $vlist$

  The Vecs returned should be elements of `orthogonalize(vlist)`.

Your procedure should use the procedure `orthogonalize` defined in the provided module `orthogonalization` but should call no other procedures. Ideally, it should be a one-line procedure.

When given the Vecs corresponding to

$$[2,4,3,5,0],[4,-2,-5,4,0],[-8,14,21,-2,0],$$
$$[-1,-4,-4,0,0],[-2,-18,-19,-6,0],[5,-3,1,-5,2]$$

the procedure might return Vecs that approximately correspond to

$$[2,4,3,5,0],[3.81,-2.37,-5.28,3.54,0],$$
$$[-1.58,-0.73,0.0009,1.21,0],[0.35,-3.16,1.01,-0.99,2]$$

Note: In this problem and the next, to test whether a vector $\boldsymbol{v}$ should be considered a zero vector, you can see if the square of its norm is very small, e.g. less than 10^{-20}.

Problem 2: Write a procedure `subset_basis(vlist)` with the following spec:

- *input:* a list $vlist$ of vectors
- *output:* a list of linearly independent vectors that span the same space as $vlist$ and that are in $vlist$

Your procedure should use `orthogonalize(vlist)` and no other procedure. Ideally, it should be a one-line procedure.

When given the Vecs corresponding to

$$[2,4,3,5,0],[4,-2,-5,4,0],[-8,14,21,-2,0],$$
$$[-1,-4,-4,0,0],[-2,-18,-19,-6,0],[5,-3,1,-5,2]$$

the procedure should return the Vecs corresponding to

$$[2,4,3,5,0],[4,-2,-5,4,0],[-1,-4,-4,0,0],[5,-3,1,-5,2]$$

Projections and representations in different bases

These problems seem to involve giving elaborate algorithms but usually the solution will involve just simple operations: matrix-vector, vector-matrix, and matrix-matrix multiplication; dot-product, maybe transpose. When you need a little more, I will indicate.

Don't start coding until you figure out how to solve the problem. You will find that in each problem the body of the procedure is very short. If the body of your procedure is at all complicated (e.g. involves a loop or even a comprehension), you're doing it wrong!

Just for fun, try making very short solutions. In my solutions the average length of the body of the procedure (not counting `def ...` `return ...` is about five characters! :) I will admit, however, that one of my solutions involves cheating a little: I use an expression that would not be mathematically acceptable if the vectors were translated into row and column vectors.

Use your understanding of linear algebra to give solutions that are as simple and pure as possible.

Problem 3: Write a procedure `orthogonal_vec2rep(Q, b)` for the following:

- *input:* An orthogonal matrix Q, and a vector $\boldsymbol{b}$ whose label set equals the column-label set of Q
- *output:* the coordinate representation of $\boldsymbol{b}$ in terms of the rows of Q.

Your code should use the `mat` module and no other module and no other procedures.

Test case: For $Q = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 10 & 20 & 30 \end{bmatrix}$, you should get $\begin{bmatrix} 21.213 & 11.547 & 28.577 \end{bmatrix}$.

Problem 4: Write a procedure `orthogonal_change_of_basis(A, B, a)` for the following:

- *input:*
  - two orthogonal matrices A and B, such that the row-label set of A equals its column-label set which equals the row and column-label sets of B as well.
  - the coordinate representation a of a vector v in terms of the rows of A.
- *output:* the coordinate representation of v in terms of the columns of B.

Just for fun, try to limit your procedure's body to about five characters (not counting `return`).

Test case: For $A = B = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} \end{bmatrix}$, $\boldsymbol{a} = [\sqrt{2}, \frac{1}{\sqrt{3}}, \frac{2}{\sqrt{6}}]$, you should get $[0.876, 0.538, 1.393]$.

Problem 5: Write a procedure `orthonormal_projection_orthogonal(W, b)` for the following spec.

- *input:* a matrix W whose rows are orthonormal, and a vector $\boldsymbol{b}$ whose label set is the column-label set of W
- *output:* the projection of $\boldsymbol{b}$ orthogonal to the row space of W.

Just for fun, try to limit your procedure's body to about seven characters (not counting `return`). This is cheating, in that the expression would not be considered good mathematical syntax; without cheating, you can do it in nineteen characters.

(Hint: First find the projection of $\boldsymbol{b}$ onto the row space of W.) Test case: For $W = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \end{bmatrix}$, $b = [10, 20, 30]$, you should get $\begin{bmatrix} -11\frac{2}{3} & 11\frac{2}{3} & 23\frac{1}{3} \end{bmatrix}$.

Ungraded Problem: Let $\boldsymbol{v}_1 = [1, 1]$ and let $\boldsymbol{v}_2 = [2, 1]$. Let `vlist` $= [\boldsymbol{v}_1, \boldsymbol{v}_2]$.

(a) Show how `orthogonalize(vlist)` computes $[\boldsymbol{v}_1^*, \boldsymbol{v}_2^*]$.

(b) Show the result of normalizing the vectors of $[\boldsymbol{v}_1^*, \boldsymbol{v}_2^*]$.

Problem 6: Write a module `orthonormalization` that defines a procedure `orthonormalize(L)` with the following spec:

- *input:* a list L of linearly independent Vecs
- *output:* a list L^* of orthonormal Vecs such that, for $i = 1, \ldots, \text{len}(L)$, the first i Vecs of L^* and the first i Vecs of L span the same space.

Your procedure should follow this outline:

1. Call `orthogonalize(L)`,
2. Compute the list of norms of the resulting vectors, and
3. Return the list resulting from normalizing each of the vectors resulting from Step 1.

Be sure to test your procedure.
When the input consists of the list of Vecs corresponding to $[4, 3, 1, 2], [8, 9, -5, -5]$, $[10, 1, -1, 5]$, your procedure should return the list of vecs corresponding approximately to $[0.73, 0.55, 0.18, 0.37], [0.19, 0.40, -0.57, -0.69], [0.53, -0.65, -0.51, 0.18]$.

Problem 7: Write a procedure `aug_orthonormalize(L)` in your `orthonormalization` module with the following spec:

- *input:* a list L of Vecs
- *output:* a pair `Qlist`, `Rlist` of lists of Vecs such that
  - `coldict2mat(L)` equals `coldict2mat(Qlist)` times `coldict2mat(Rlist)`, and
  - `Qlist` $=$ `orthonormalize(L)`

Your procedure should start by calling the procedure `aug_orthogonalize(L)` defined in the module `orthogonalization`. I suggest that your procedure also use a subroutine `adjust(v, multipliers)` with the following spec:

- *input:* a Vec `v` with domain $\{0, 1, 2, \ldots, n-1\}$ and an n-element list `multipliers` of scalars
- *output:* a Vec `w` with the same domain as `v` such that `w[i] = multipliers[i]*v[i]`

Here is an example for testing `aug_orthonormalize(L)`:

```
>>> L = [list2vec(v) for v in [[4,3,1,2],[8,9,-5,-5],[10,1,-1,5]]]
>>> print(coldict2mat(L))

      0  1  2
     ---------
 0  |  4  8 10
 1  |  3  9  1
 2  |  1 -5 -1
 3  |  2 -5  5

>>> Qlist, Rlist = aug_orthonormalize(L)
>>> print(coldict2mat(Qlist))

           0      1      2
     ----------------------
 0  |   0.73  0.187  0.528
 1  |  0.548  0.403 -0.653
 2  |  0.183 -0.566 -0.512
 3  |  0.365 -0.695  0.181

>>> print(coldict2mat(Rlist))

          0    1      2
     ------------------
 0  |  5.48 8.03   9.49
 1  |     0 11.4 -0.636
 2  |     0    0   6.04

>>> print(coldict2mat(Qlist)*coldict2mat(Rlist))

      0  1  2
     ---------
 0  |  4  8 10
 1  |  3  9  1
 2  |  1 -5 -1
 3  |  2 -5  5
```

Keep in mind, however, that numerical calculations are approximate:

```
>>> print(coldict2mat(Qlist)*coldict2mat(Rlist)-coldict2mat(L))

               0 1        2
     ----------------------
 0  |  -4.44E-16 0        0
 1  |          0 0 4.44E-16
 2  |  -1.11E-16 0        0
 3  |  -2.22E-16 0        0
```

Problem 8: In each of the following parts, you are given a matrix A and a vector $\boldsymbol{b}$. You are also given the approximate QR factorization of A. You are to

- find a vector $\hat{\boldsymbol{x}}$ that minimizes $||A\hat{\boldsymbol{x}} - \boldsymbol{b}||^2$,
- prove to yourself that the columns of A are (approximately) orthogonal to the residual $\boldsymbol{b} - A\hat{\boldsymbol{x}}$ by computing the inner products, and
- calculate the value of $\|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|$.

1. $A = \begin{bmatrix} 8 & 1 \\ 6 & 2 \\ 0 & 6 \end{bmatrix}$ and $\boldsymbol{b} = [10, 8, 6]$

$$A = \underbrace{\begin{bmatrix} 0.8 & -0.099 \\ 0.6 & 0.132 \\ 0 & 0.986 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 10 & 2 \\ 0 & 6.08 \end{bmatrix}}_{R}$$

2. $A = \begin{bmatrix} 3 & 1 \\ 4 & 1 \\ 5 & 1 \end{bmatrix}$ and $\boldsymbol{b} = [10, 13, 15]$

$$A = \underbrace{\begin{bmatrix} 0.424 & .808 \\ 0.566 & 0.115 \\ 0.707 & -0.577 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 7.07 & 1.7 \\ 0 & 0.346 \end{bmatrix}}_{R}$$

Problem 9: Write and test a procedure `QR_solve(A, b)`. Assuming the columns of A are linearly independent, this procedure should return the vector $\hat{\boldsymbol{x}}$ that minimizes $\|\boldsymbol{b} - A\hat{\boldsymbol{x}}\|$.

The procedure should use

- `triangular_solve(rowlist, label_list, b)` defined in the module `triangular`, and
- the procedure `factor(A)` defined in the module `QR`, which in turn uses the procedure `aug_orthonormalize(L)` that you wrote in Problem 6

Note that `triangular_solve` requires its matrix to be represented as a list of rows. The row-labels of the matrix R returned by `QR.factor(R)` are 0,1,2,... so it suffices to use the dictionary returned by `mat2rowdict(R)`.

Note also that `triangular_solve` must be supplied with a list `label_list` of column-labels in order that it know how to interpret the vectors in `rowlist` as forming a triangular system. The column-labels of R are, of course, the column-labels of A. The ordering to provide here must match the ordering used in `QR.factor(A)`, which is `sorted(A.D[1], key=repr)`.

You can try your procedure on the examples given in Problem 8 and on the following example:

```
>>> A=Mat(({'a','b','c'},{'A','B'}), {('a','A'):-1, ('a','B'):2,
           ('b','A'):5, ('b','B'):3,('c','A'):1, ('c','B'):-2})
>>> print(A)

       A  B
     -------
 a  |  -1  2
 b  |   5  3
```

```
 c  |   1 -2

>>> Q, R = QR.factor(A)

>>> print(Q)

            0     1
     --------------
 a  |  -0.192  0.68
 b  |   0.962 0.272
 c  |   0.192 -0.68

>>> print(R)

         A    B
     ----------
 0  |  5.2 2.12
 1  |    0 3.54

>>> b = Vec({'a','b','c'}, {'a':1,'b':-1})
>>> x = QR_solve(A,b)
>>> x
Vec({'A', 'B'},{'A': -0.269..., 'B': 0.115...})
```

A good way to test your solution is to verify that the residual is (approximately) orthogonal to the columns of A:

```
>>> A.transpose()*(b-A*x)
Vec({'A', 'B'},{'A': -2.22e-16, 'B': 4.44e-16})
```