GSMST

Applications of Linear Algebra in Programming

Chapter 4 Assignment

Submitted By:
Anish Goyal
4th Period

Submitted To:
Mrs. Denise Stiffler
Educator

April 24, 2023

Infinite Possibilities

Gwinnett School of Mathematics, Science, and Technology

Table of contents

Column-vector and row-vector matrix multiplication 3
Problem 4.17.11 . . . 3

Matrix Class 5
Problem 4.17.12 . . . 5
Testing `mat.py` . . . 13

Matrix-vector and vector-matrix multiplication definitions in Python 14
Problem 4.17.13 . . . 14
Problem 4.17.14 . . . 14
Problem 4.17.15 . . . 15
Problem 4.17.16 . . . 15

Matrix-matrix multiplication in Python 16
Problem 4.17.17 . . . 16
Problem 4.17.18 . . . 16

Dot products via matrix-matrix multiplication 18
Problem 4.17.19 . . . 18

Comprehension practice 20
Problem 4.17.20 . . . 20

The inverse of a 2×2 matrix 21
Problem 4.17.21 . . . 21

Matrix inverse criterion 23
Problem 4.17.22 . . . 23
Problem 4.17.23 . . . 24

Column-vector and row-vector matrix multiplication

Problem 4.17.11

Compute the result of the following matrix multiplications:

(a) $\begin{bmatrix} 2 & 3 & 1 \\ 1 & 3 & 4 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}$

$= \begin{bmatrix} 2 \cdot 2 + 3 \cdot 2 + 1 \cdot 3 \\ 1 \cdot 2 + 3 \cdot 2 + 4 \cdot 3 \end{bmatrix}$

$= \begin{bmatrix} 4 + 6 + 3 \\ 2 + 6 + 12 \end{bmatrix}$

$= \begin{bmatrix} 13 \\ 20 \end{bmatrix}$

(b) $\begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 5 & 1 & 1 \\ 2 & 3 & 0 \end{bmatrix}$

$= \begin{bmatrix} 2 \cdot 1 + 4 \cdot 5 + 1 \cdot 2 & 2 \cdot 2 + 4 \cdot 1 + 1 \cdot 3 & 2 \cdot 0 + 4 \cdot 1 + 1 \cdot 0 \end{bmatrix}$

$= \begin{bmatrix} 2 + 20 + 2 & 4 + 4 + 3 & 0 + 4 + 0 \end{bmatrix}$

$= \begin{bmatrix} 24 & 11 & 4 \end{bmatrix}$

(c) $\begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 3 & 1 & 5 & 2 \\ -2 & 6 & 1 & -1 \end{bmatrix}$

$= \begin{bmatrix} 2 \cdot 3 + 1 \cdot -2 & 2 \cdot 1 + 1 \cdot 6 & 2 \cdot 5 + 1 \cdot 1 + 2 \cdot 2 + 1 \cdot -1 \end{bmatrix}$

$= \begin{bmatrix} 6 + (-2) & 2 + 6 & 10 + 1 & 4 + (-1) \end{bmatrix}$

$= \begin{bmatrix} 4 & 8 & 11 & 3 \end{bmatrix}$

(d) $\begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 1 & 3 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$

$= \begin{bmatrix} 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 4 \\ 1 \cdot 1 + 1 \cdot 2 + 3 \cdot 3 + 1 \cdot 4 \end{bmatrix}$

$= \begin{bmatrix} 1 + 4 + 9 + 16 \\ 1 + 2 + 9 + 4 \end{bmatrix}$

$= \begin{bmatrix} 30 \\ 16 \end{bmatrix}$

(e) $\begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix}^{\intercal} \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix}$

$= \begin{bmatrix} 4 & 1 & -3 \end{bmatrix} \begin{bmatrix} -1 & 1 & 1 \\ 1 & 0 & 2 \\ 0 & 1 & -1 \end{bmatrix}$

$= \begin{bmatrix} 4 \cdot -1 + 1 \cdot 1 + -3 \cdot 0 & 4 \cdot 1 + 1 \cdot 0 + -3 \cdot 1 & 4 \cdot 1 + 1 \cdot 2 + -3 \cdot -1 \end{bmatrix}$

$= \begin{bmatrix} -4 + 1 + 0 & 4 + 0 + (-3) & 4 + 2 + 3 \end{bmatrix}$

$= \begin{bmatrix} -3 & 1 & 9 \end{bmatrix}$

Matrix Class

Problem 4.17.12

You will write a module `mat` implementing a matrix classs `Mat`. The data structure used for instances of `Mat` resembles that used for isntances of `Vec`. The only difference is that the domain `D` will now store a pair (i.e., a 2-tuple) of sets instead of a single set. The keys of the dictionary `f` are pairs of elements of the Cartesian product of the two sets in `D`. The operations defined for `Mat` include entry setters and getters, an equality test, addition and subtraction and negative, multiplication by a scalar, transpose, vector-matrix, and matrix-vector multiplication, and matrix-matrix multiplication. Like `Vec`, the class `Mat` is defined to enable use of operators such as `+` and `*`. The syntax for using instances of `Mat` is as follows, where `A` and `B` are matrices, `v` is a vector, `alpha` is a scalar, `r` is a row label, and `c` is a column label:

operation	syntax
Matrix addition and subtraction	`A+B` and `A-B`
Matrix negative	`-A`
Scalar-matrix multiplication	`alpha*A`
Matrix equality test	`A==B`
Matrix transpose	`A.transpose()`
Getting and setting a matrix entry	`A[r,c]` and `A[r, c] = alpha`
Matrix-vector and vector-matrix multiplication	`v*A` and `A*v`
Matrix-matrix multiplication	`A*B`

You are required to write the procedures `equal, getitem, setitem, mat_add, mat_scalar_mul, transpose, vector_matrix_mul, matrix_vector_mal,` and `matrix_matrix_mul`. You should start by getting `equal` working since `==` is used in the doctests for other procedures.

Note: You are encouraged to use operator (e.g. `M[r, c]`) in your procedures. (Of course, you can't, for example, use the syntax `M[r, c]` in your `getitem` procedure.)

Put the file `mat.py` in your working directory, and, for each procedure, replace the `pass` statement with a working version. Test your implementation using `doctest` as you did with `vec.py` in Problem 2.14.10. Make sure your implementation works with matrices whose row-label sets differ from their column-label sets.

Note: Use the sparse matrix-vector multiplication algorithm described in Section 4.8 (the one based on the "ordinary" definition) for matrix-vector multiplication. Use the analogous algorithm for vector-matrix multiplication. Do not use `transpose` in your multiplication algorithms. Do not use any external proceduers or modules other than `vec`. In particular, do not use procedures from `matutil`. If you do, your `Mat` implementation is likely not to be efficient enough for use with large sparse matrices.

```
# Copyright 2013 Philip N. Klein
from vec import Vec

#Test your Mat class over R and also over GF(2).  The following tests use only R.
def equal(A, B):
    """
    Returns true iff A is equal to B.
    >>> Mat(({'a','b'}, {0,1}), {('a',1):0}) == Mat(({'a','b'}, {0,1}), {('b',1):0})
    True
    >>> A = Mat(({'a','b'}, {0,1}), {('a',1):2, ('b',0):1})
    >>> B = Mat(({'a','b'}, {0,1}), {('a',1):2, ('b',0):1, ('b',1):0})
    >>> C = Mat(({'a','b'}, {0,1}), {('a',1):2, ('b',0):1, ('b',1):5})
    >>> A == B
    True
    >>> A == C
    False
    >>> A == Mat(({'a','b'}, {0,1}), {('a',1):2, ('b',0):1})
    True
    """
    assert A.D == B.D
    for row in A.D[0]:
        for col in A.D[1]:
            if getitem(A,(row, col)) != getitem(B,(row, col)):
                return False
    return True

def getitem(M, k):
    """
    Returns the value of entry k in M, where k is a 2-tuple
    >>> M = Mat(({1,3,5}, {'a'}), {(1,'a'):4, (5,'a'): 2})
    >>> M[1,'a']
    4
    >>> M[3,'a']
    0
    """
    assert k[0] in M.D[0] and k[1] in M.D[1]
    return M.f[k] if k in M.f.keys() else 0
def setitem(M, k, val):
    """
```

```
    Set entry k of Mat M to val, where k is a 2-tuple.
    >>> M = Mat(({'a','b','c'}, {5}), {('a', 5):3, ('b', 5):7})
    >>> M['b', 5] = 9
    >>> M['c', 5] = 13
    >>> M == Mat(({'a','b','c'}, {5}), {('a', 5):3, ('b', 5):9, ('c',5):13})
    True
    >>> N = Mat(({((),), 7}, {True, False}), {})
    >>> N[(7, False)] = 1
    >>> N[(((),), True)] = 2
    >>> N == Mat(({((),), 7}, {True, False}), {(7,False):1, (((),), True):2})
    True
    """
    assert k[0] in M.D[0] and k[1] in M.D[1]
    M.f[k]=val

def add(A, B):
    """
    Return the sum of Mats A and B.
    >>> A1 = Mat(({3, 6}, {'x','y'}), {(3,'x'):-2, (6,'y'):3})
    >>> A2 = Mat(({3, 6}, {'x','y'}), {(3,'y'):4})
    >>> B = Mat(({3, 6}, {'x','y'}), {(3,'x'):-2, (3,'y'):4, (6,'y'):3})
    >>> A1 + A2 == B
    True
    >>> A2 + A1 == B
    True
    >>> A1 == Mat(({3, 6}, {'x','y'}), {(3,'x'):-2, (6,'y'):3})
    True
    >>> zero = Mat(({3,6}, {'x','y'}), {})
    >>> B + zero == B
    True
    >>> C1 = Mat(({1,3}, {2,4}), {(1,2):2, (3,4):3})
    >>> C2 = Mat(({1,3}, {2,4}), {(1,4):1, (1,2):4})
    >>> D = Mat(({1,3}, {2,4}), {(1,2):6, (1,4):1, (3,4):3})
    >>> C1 + C2 == D
    True
    """
    assert A.D == B.D
    C=A.copy()
    for row in A.D[0]:
```

```
        for col in A.D[1]:
            setitem(C, (row,col), getitem(A, (row,col))+getitem(B, (row,col)))
    return C

def scalar_mul(M, x):
    """
    Returns the result of scaling M by x.
    >>> M = Mat(({1,3,5}, {2,4}), {(1,2):4, (5,4):2, (3,4):3})
    >>> 0*M == Mat(({1, 3, 5}, {2, 4}), {})
    True
    >>> 1*M == M
    True
    >>> 0.25*M == Mat(({1,3,5}, {2,4}), {(1,2):1.0, (5,4):0.5, (3,4):0.75})
    True
    """
    C=M.copy()
    for row in M.D[0]:
        for col in M.D[1]:
            setitem(C, (row,col), x*getitem(M, (row,col)))
    return C

def transpose(M):
    """
    Returns the matrix that is the transpose of M.
    >>> M = Mat(({0,1}, {0,1}), {(0,1):3, (1,0):2, (1,1):4})
    >>> M.transpose() == Mat(({0,1}, {0,1}), {(0,1):2, (1,0):3, (1,1):4})
    True
    >>> M = Mat(({'x','y','z'}, {2,4}), {('x',4):3, ('x',2):2, ('y',4):4, ('z',4):5})
    >>> Mt = Mat(({2,4}, {'x','y','z'}), {(4,'x'):3, (2,'x'):2, (4,'y'):4, (4,'z'):5})
    >>> M.transpose() == Mt
    True
    """
    C=Mat((M.D[1], M.D[0]),{} )
    for row in M.D[0]:
        for col in M.D[1]:
            setitem(C, (col,row), getitem(M, (row,col)))
```

```
    return C

def vector_matrix_mul(v, M):
    """
    returns the product of vector v and matrix M
    >>> v1 = Vec({1, 2, 3}, {1: 1, 2: 8})
    >>> M1 = Mat(({1, 2, 3}, {'a', 'b', 'c'}), {(1, 'b'): 2, (2, 'a'):-1, (3, 'a'): 1, (3, 'c'): 7})
    >>> v1*M1 == Vec({'a', 'b', 'c'},{'a': -8, 'b': 2, 'c': 0})
    True
    >>> v1 == Vec({1, 2, 3}, {1: 1, 2: 8})
    True
    >>> M1 == Mat(({1, 2, 3}, {'a', 'b', 'c'}), {(1, 'b'): 2, (2, 'a'):-1, (3, 'a'): 1, (3, 'c'): 7})
    True
    >>> v2 = Vec({'a','b'}, {})
    >>> M2 = Mat(({'a','b'}, {0, 2, 4, 6, 7}), {})
    >>> v2*M2 == Vec({0, 2, 4, 6, 7},{})
    True
    """
    assert M.D[0] == v.D
    v_tmp = Vec(M.D[1], {})
    for col in v_tmp.D:
        for row in M.D[0]:
            v_tmp[col] = v_tmp[col] + getitem(M,(row,col)) * v[row]
    return v_tmp

def matrix_vector_mul(M, v):
    """
    Returns the product of matrix M and vector v.
    >>> N1 = Mat(({1, 3, 5, 7}, {'a', 'b'}), {(1, 'a'): -1, (1, 'b'): 2, (3, 'a'): 1, (3, 'b'):4, (7, 'a'): 3, (5, 'b'):-1})
    >>> u1 = Vec({'a', 'b'}, {'a': 1, 'b': 2})
    >>> N1*u1 == Vec({1, 3, 5, 7},{1: 3, 3: 9, 5: -2, 7: 3})
    True
    >>> N1 == Mat(({1, 3, 5, 7}, {'a', 'b'}), {(1, 'a'): -1, (1, 'b'): 2, (3, 'a'): 1, (3, 'b'):4, (7, 'a'): 3, (5, 'b'):-1})
    True
    >>> u1 == Vec({'a', 'b'}, {'a': 1, 'b': 2})
    True
    >>> N2 = Mat(({('a', 'b'), ('c', 'd')}, {1, 2, 3, 5, 8}), {})
```

```
    >>> u2 = Vec({1, 2, 3, 5, 8}, {})
    >>> N2*u2 == Vec({('a', 'b'), ('c', 'd')},{})
    True
    """
    assert M.D[1] == v.D
    v_tmp = Vec(M.D[0], {})
    for row in v_tmp.D:
        for col in M.D[1]:
            v_tmp[row] = v_tmp[row] + getitem(M,(row,col)) * v[col]
    return v_tmp

def matrix_matrix_mul(A, B):
    """
    Returns the result of the matrix-matrix multiplication, A*B.
    >>> A = Mat(({0,1,2}, {0,1,2}), {(1,1):4, (0,0):0, (1,2):1, (1,0):5, (0,1):3, (0,2):2})
    >>> B = Mat(({0,1,2}, {0,1,2}), {(1,0):5, (2,1):3, (1,1):2, (2,0):0, (0,0):1, (0,1):4})
    >>> A*B == Mat(({0,1,2}, {0,1,2}), {(0,0):15, (0,1):12, (1,0):25, (1,1):31})
    True
    >>> C = Mat(({0,1,2}, {'a','b'}), {(0,'a'):4, (0,'b'):-3, (1,'a'):1, (2,'a'):1, (2,'b'):-2})
    >>> D = Mat(({'a','b'}, {'x','y'}), {('a','x'):3, ('a','y'):-2, ('b','x'):4, ('b','y'):-1})
    >>> C*D == Mat(({0,1,2}, {'x','y'}), {(0,'y'):-5, (1,'x'):3, (1,'y'):-2, (2,'x'):-5})
    True
    >>> M = Mat(({0, 1}, {'a', 'c', 'b'}), {})
    >>> N = Mat(({'a', 'c', 'b'}, {(1, 1), (2, 2)}), {})
    >>> M*N == Mat(({0,1}, {(1,1), (2,2)}), {})
    True
    >>> E = Mat(({'a','b'},{'A','B'}), {('a','A'):1,('a','B'):2,('b','A'):3,('b','B'):4})
    >>> F = Mat(({'A','B'},{'c','d'}),{('A','d'):5})
    >>> E*F == Mat(({'a', 'b'}, {'d', 'c'}), {('b', 'd'): 15, ('a', 'd'): 5})
    True
    >>> F.transpose()*E.transpose() == Mat(({'d', 'c'}, {'a', 'b'}), {('d', 'b'): 15, ('d', 'a'): 5})
    True
```

```
    """
    assert A.D[1] == B.D[0]
    M=Mat((A.D[0], B.D[1]), {})
    for col in B.D[1]:
        for row in A.D[0]:
            v_tmp = Vec(B.D[0], {})
            for row_t in B.D[0]:
                v_tmp[row_t]=getitem(B, (row_t, col))
            v = matrix_vector_mul(A, v_tmp)
            setitem(M,(row, col), v[row])
    return M

################################################################################

class Mat:
    def __init__(self, labels, function):
        assert isinstance(labels, tuple)
        assert isinstance(labels[0], set) and isinstance(labels[1], set)
        assert isinstance(function, dict)
        self.D = labels
        self.f = function

    __getitem__ = getitem
    __setitem__ = setitem
    transpose = transpose

    def __neg__(self):
        return (-1)*self

    def __mul__(self,other):
        if Mat == type(other):
            return matrix_matrix_mul(self,other)
        elif Vec == type(other):
            return matrix_vector_mul(self,other)
        else:
            return scalar_mul(self,other)
            #this will only be used if other is scalar (or not-supported). mat and vec both have __mul__ implemented

    def __rmul__(self, other):
```

```
        if Vec == type(other):
            return vector_matrix_mul(other, self)
        else:  # Assume scalar
            return scalar_mul(self, other)

    __add__ = add

    def __radd__(self, other):
        "Hack to allow sum(...) to work with matrices"
        if other == 0:
            return self

    def __sub__(a,b):
        return a+(-b)

    __eq__ = equal

    def copy(self):
        return Mat(self.D, self.f.copy())

    def __str__(M, rows=None, cols=None):
        "string representation for print()"
        if rows == None: rows = sorted(M.D[0], key=repr)
        if cols == None: cols = sorted(M.D[1], key=repr)
        separator = ' | '
        numdec = 3
        pre = 1+max([len(str(r)) for r in rows])
        colw = {col:(1+max([len(str(col))] + [len('{0:.{1}G}'.format(M[row,col],numdec)) if isinstance(M[row,col], int) or isinstance(M[row,col], float) else len(str(M[row,col])) for row in rows])) for col in cols}
        s1 = ' '*(1+ pre + len(separator))
        s2 = ''.join(['{0:>{1}}'.format(str(c),colw[c]) for c in cols])
        s3 = ' '*(pre+len(separator)) + '-'*(sum(list(colw.values())) + 1)
        s4 = ''.join(['{0:>{1}} {2}'.format(str(r), pre,separator)+''.join(['{0:>{1}.{2}G}'.format(M[r,c],colw[c],numdec) if isinstance(M[r,c], int) or isinstance(M[r,c], float) else '{0:>{1}}'.format(M[r,c], colw[c]) for c in cols])+'\n' for r in rows])
        return '\n' + s1 + s2 + '\n' + s3 + '\n' + s4
```

```python

    def pp(self, rows, cols):
        print(self.__str__(rows, cols))

    def __repr__(self):
        "evaluatable representation"
        return "Mat(" + str(self.D) +", " + str(self.f) + ")"

    def __iter__(self):
        raise TypeError('%r object is not iterable' %
        ↪  self.__class__.__name__)
```

Testing mat.py

```python
import subprocess
subprocess.run(["python", "-m", "doctest", "mat.py"], check=True)
```

```
CompletedProcess(args=['python', '-m', 'doctest', 'mat.py'], returncode=0)
```

Note that a `returncode` of 0 means that all of the testcases executed successfully.

Matrix-vector and vector-matrix multiplication definitions in Python

You will write several procedures, each implementing matrix-vector multiplication using a *specified definition* of matrix-vector multiplication or vector-matrix multiplication.

- These procedures can be written and run after you write `getitem(M, k)` but before you make any other additions to `Mat`.
- These procedures must *not* be designed to exploit sparsity.
- Your code must *not* use the matrix-vector and vector-matrix multiplication operations that are not part of `Mat`.
- Your code should use procedures `mat2rowdict`, `mat2coldict`, `rowdict2mat(rowdict)`, and/or `coldict2mat(coldict)` from the `matutil` module.

Problem 4.17.13

Write the procedure `lin_comb_mat_vec_mult(M, v)`, which multiplies `M` times `v` using the linear-combination definition. For this problem, the only operation on `v` you are allowed is getting the value of an entry using brackets: `v[k]`. The vector returned must be computed as a linear combination.

```
def lin_comb_mat_vec_mult(M, v):
    colDict = mat2coldict(M)
    res = Vec(M.D[0],{})
    for col in v.D:
        res = res + v[col] * colDict[col]
    return res
```

Problem 4.17.14

Write `lin_comb_vec_mat_mult(v, M)`, which multiply `v` times `M` using the linear-combination definition. For this problem, the only operation on `v` you are allowed is getting the value of an entry using brackets: `v[k]`. The vector returned must be computed as a linear combination.

```
def lin_comb_vec_mat_mult(v, M):
  rowDict = mat2rowdict(M)
  res = Vec(M.D[1],{})
  for col in v.D:
      res = res + v[col] * rowDict[col]
```

```
  return res
```

Problem 4.17.15

Write `dot_product_mat_vec_mult(M, v)`, which multiplies `M` times `v` using the dot-product definition. For this problem, the only operation on `v` you are allowed is taking the dot-product of `v` and another vector and `v`: `u*v` or `v*u`. The entries of the vector returned must be computed using dot-product.

```
def dot_product_mat_vec_mult(M, v):
    res = Vec(M.D[0], {})
    rowDict = mat2rowdict(M)
    for row in M.D[0]:
        res[row]= rowDict[row] * v
    return res
```

Problem 4.17.16

Write `dot_product_vec_mat_mult(v, M)`, which multiplies `v` times `M` using the dot-product definition. For this problem, the only operation on `v` you are allowed is taking the dot-product of `v` and another vector and `v`: `u*v` or `v*u`. The entries of the vector returned must be computed using the dot-product.

```
def dot_product_vec_mat_mult(v, M):
  res = Vec(M.D[1], {})
  colDict = mat2coldict(M)
  for col in M.D[1]:
      res[col] = colDict[col] * v
    return res
```

Matrix-matrix multiplication in Python

You will write several procedures, each implementing matrix-matrix multiplication using a *specified definition* of matrix-matrix multiplication.

- These procedures can be written and run only after you have written and tested the procedures in `mat.py` that perform matrix-vector and vector-matrix multiplication.
- These procedures must *not* be designed to exploit sparsity.
- Your code must *not* use the matrix-matrix multiplication that is part of `Mat`. For this reason, you can write these procedures before completing that part of `Mat`.
- Your code should use the procedures `mat2rowdict, mat2coldict, rowdict2mat(rowdict)`, and/or `coldict2mat(coldict)` from the `matutil` module.

Problem 4.17.17

Write `Mv_mat_mat_mult(A, B)` to compute the matrix-matrix product `A*B`, using the matrix-vector multiplication definition of matrix-matrix multiplication. For this procedure, the only operation you are allowed to do on `A` is matrix-vector multiplication, using the `*` operator: `A*v`. Do *not* use the named procedure `matrix_vector_mul` or any of the procedures defined in the previous problem.

```
def Mv_mat_mat_mult(A, B):
  colDict = mat2coldict(B)
  res=dict()
  for col in colDict.keys():
      res[col]=A*colDict[col]
  return coldict2mat(res)
```

Problem 4.17.18

Write `vM_mat_mat_mult(A, B)` to compute the matrix-matrix product `A*B`, using the vector-matrix definition. For this procedure, the only operation you are allowed to do on `B` is vector-matrix multiplication, using the `*` operator: `v*B`. Do *not* use the named procedure `vector_matrix_mul` or any of the procedures defined in the previous problem.

```
def vM_mat_mat_mult(A, B):
  rowDict=mat2rowdict(A)
  res=dict()
```

```
  for row in rowDict.keys():
    res[row]=rowDict[row]*B
  return rowdict2mat(res)
```

Dot products via matrix-matrix multiplication

Problem 4.17.19

Let A be a matrix whose column labels are countries and whose row labels are votes taken in the United Nations (UN), where $A[i, j]$ is +1 or -1 or 0 depending on whether country j votes in favor of or against neither in vote i.
As in the politics lab, we can compare countries by comparing their voting records. Let $M = A^T A$. Then M's row and column labels are countries, and $M[i, j]$ is the dot-product of country i's voting record with country j's voting record. The provided file `UN_voting_data.txt` has one line per country. The line consists of the country's name, followed by +1's, -1's and zeroes, separated by spaces. Read in the data and form the matrix A. Then form the matrix $M = A^T A$. (Note: this will take quite a while—from fifteen minutes to an hour, depending on your computer.)
Use M to answer the following questions.

```
  from matutil import *
  from vecutil import *

file = open('UN_voting_data.txt', 'r')
raw_data = file.readlines()
for i in range(len(raw_data)):
    line = raw_data[i].replace('\n', '')
    raw_data[i] = line

countries_2d = []
for i in range(len(raw_data)):
    curr = raw_data[i].split(' ')
    country = curr[0]
    votes = []
    for j in range(1, len(curr)):
        votes.append(int(curr[j]))
    countries_2d.append([country, votes])

agreement_map = {}
for i in range(0, len(countries_2d) - 1):

    country1 = countries_2d[i][0]
    votes1 = countries_2d[i][1]

    for j in range(i + 1, len(countries_2d)):
```

```
        country2 = countries_2d[j][0]
        votes2 = countries_2d[j][1]

        dot_product = 0
        for k in range(len(votes1)):
            dot_product += votes1[k] * votes2[k]
        agreement_map[tuple([country1, country2])] = dot_product

agreement_map = sorted(agreement_map.items(), key=lambda x:x[1])
```

1. Which pair of countries are most opposed? (They have the most negative dot-product.)

```
print(agreement_map[0])
```

```
(('Belarus', 'United_States_of_America'), -1927)
```

2. What are the ten most opposed pairs of countries?

```
for i in range(10):
  print(agreement_map[i])
```

```
(('Belarus', 'United_States_of_America'), -1927)
(('United_States_of_America', 'Syria'), -1861)
(('United_States_of_America', 'Cuba'), -1807)
(('Algeria', 'United_States_of_America'), -1742)
(('United_States_of_America', 'Viet_Nam'), -1740)
(('Libya', 'United_States_of_America'), -1665)
(('United_States_of_America', 'Guinea'), -1616)
(('United_States_of_America', 'Mongolia'), -1615)
(('United_States_of_America', 'Mali'), -1605)
(('United_States_of_America', 'Sudan'), -1582)
```

3. Which pair of distinct countries are in the greatest agreement (have the most positive dot-product)?

```
print(agreement_map[-1])
```

```
(('Philippines', 'Thailand'), 4229)
```

Comprehension practice

Problem 4.17.20

Write the one-line procedure `dictlist_helper(dlist, k)` with the following spec:

- *input:* a list `dlist` of dictionaries which all have the same keys, and a key `k`
- *output:* the list whose i^{th} element is the value corresponding to the key `k` in the i^{th} dictionary of `dlist`
- *example:* With inputs `dlist=[{'a': 'apple', 'b': 'bear'}, {'a': 1, 'b': 2}]` and `k='a'`, the output is `['apple', 1]`

The procedure should use a comprehension.

```
def dictlist_helper(dlist, k):
  return [d[k] for d in dlist]
dlist=[{'a':'apple', 'b':'bear'}, {'a':1, 'b':2}]
print(dictlist_helper(dlist, 'a'))
```

```
['apple', 1]
```

The inverse of a 2×2 matrix

Problem 4.17.21

1. Use a formula given in the text to solve the linear system $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

$3x_1 + 4x_2 = 1$
$2x_1 + 1x_2 = 0$

$\implies$

$x_2 = -2x_1 \implies 3x_1 + 4(-2x_1) = 1 \implies x_1 = -\frac{1}{5}$, and
$x_2 = \frac{2}{5}$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{5} \\ \frac{2}{5} \end{bmatrix}$$

2. Use the formula to solve the linear system $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

$3y_1 + 4y_2 = 0$
$2y_1 + 1y_2 = 1$

$\implies$

$y_2 = -\frac{3}{4}y_1 \implies 2y_1 - \frac{3}{4}y_1 = 1 \implies y_1 = \frac{4}{5}$, and $y_2 = -\frac{3}{5}$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \frac{4}{5} \\ -\frac{3}{5} \end{bmatrix}$$

3. Use your solutions to find a $\mathbf{2 \times 2}$ matrix M such that $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$ **times M is an identity matrix.**

$$M = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{5} & \frac{4}{5} \\ \frac{2}{5} & -\frac{3}{5} \end{bmatrix}$$

4. Calculate M times $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$ **and calculate** $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$ **times M and use `Corollary 4.13.19` to decide whether M is the inverse of** $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$**. Explain your answer.**

$$\begin{bmatrix} -\frac{1}{5} & \frac{4}{5} \\ \frac{2}{5} & -\frac{3}{5} \end{bmatrix} \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} -\frac{1}{5} & \frac{4}{5} \\ \frac{2}{5} & -\frac{3}{5} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Since AB and BA are both identity matrices, M is the inverse of $\begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}$ according to `Corollary 4.13.19`.

Matrix inverse criterion

Problem 4.17.22

For each of the parts below, use `Corollary 4.13.19` to demonstrate that the pairs of matrices given are or are not inverse of each other.

1. matrices $\begin{bmatrix} 5 & 1 \\ 9 & 2 \end{bmatrix}, \begin{bmatrix} 2 & -1 \\ -9 & 5 \end{bmatrix}$ **over** $\mathbb{R}$

$$\begin{bmatrix} 5 & 1 \\ 9 & 2 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -9 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 2 & -1 \\ -9 & 5 \end{bmatrix} \begin{bmatrix} 5 & 1 \\ 9 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

These matrices are inverses.

2. matrices $\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix}$ **over** $\mathbb{R}$

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

These matrices are inverses.

3. matrices $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}, \begin{bmatrix} 1 & \frac{1}{6} \\ -2 & \frac{1}{2} \end{bmatrix}$ **over** $\boldsymbol{GF(2)}$

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & \frac{1}{6} \\ -2 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ -4 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \frac{1}{6} \\ -2 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 3 & \frac{4}{3} \\ -6 & -1 \end{bmatrix}$$

These matrices are *not* inverses.

4. matrices $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ **over** $\boldsymbol{GF(2)}$

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

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

These matrices are *not* inverses.

Problem 4.17.23

Specify a function f (by domain, co-domain, and rule) that is invertible but such that there is no matrix A such that $f(\boldsymbol{x}) = A\boldsymbol{x}$.

If $f(\boldsymbol{x}) = \{x_i^3, x_i \in \boldsymbol{x}\}$, then
$f'(\boldsymbol{x}) = g(\boldsymbol{x}) = \{x_i^{\frac{1}{3}}, x_i \in \boldsymbol{x}\}$, but there is no matrix A where $f(\boldsymbol{x}) = A\boldsymbol{x}$.