03_07_gaussian_elimination_II

Unknown Author

March 11, 2014

Part I

Gaussian elimination II

1 Importing functions we wrote in the last class : packages.

In python, one can just store a bunch of function definitions in a file. Then the file can be imported like any other package. We have stored our functions in `gauss_elim_old_fn.py`. We import it as

In [1]:
```
from gauss_elim_old_fn import *
```

```
[[1.0, 2.0, 3.0], [2.0, 1.0, 2.0], [3.0, 2.0, 1.0]]
          1           2           3
          2           1           2
          3           2           1
 1  2  3
 2  1  2
 3  2  1
--------------------------
 4  8 12
 2  1  2
 3  2  1
--------------------------
 4  8 12
 2  1  2
 3  2  1
 4  8 12
 2  1  2
 3  2  1
--------------------
 0  6  8
 2  1  2
 3  2  1
----------------------
 0  6  8
 2  1  2
 3  2  1
(1, 1)
```

```
The matrix is the zero matrix.
(-1, -1)
The matrix is the zero matrix.
(-1, -1)
Please input a natural number.
None
1
1
2
3
5
8
13
21
34
55
Please input an positive integer.
None
Please input a natural number.
None
1
1
2
3
5
8
13
21
34
55
Please input a natural number.
None
9.33262154439e+157 Expected a non-negative integer.
None Expected a non-negative integer.
None 933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
(2, 1)
  1   2   3   4
  5   6   7   8
  9  10  11  12

  1   2   3   4
  5   6   7   8
  9  10  11  12
  1   2   3   4
  5   6   7   8
  9  10  11  12
---------------
  1   2   3   4
  5   6   7  -2
  9  10  11  -3
   0.00000    0.50000    1.00000    0.00000
```

```
   2.00000   11.00000    0.00000    0.00000
   1.00000    0.00000    0.00000    9.00000
--------------------------------------------------
The matrix is empty.
Nothing to copy: S is empty
   1.00000    5.50000    0.00000    0.00000
   0.00000    1.00000    2.00000    0.00000
   0.00000    0.00000    1.00000    0.81818
```

In [2]:
```
def gauss_elim_prelim(matrix) :
    row = 0
    row_offset = 0
    col_offset = 0

    (r, c) = first_non_zero(matrix)
    if r == -1 :
        print "Nothing to do: The matrix is zero."
    else :
        matrix = elem1(matrix, 0, r)
        matrix = sweep(matrix, (0, c))
        print_matrix(matrix, "%6.2f")
```

In [3]:
```
gauss_elim_prelim([[0,0,2,2],\
           [1,4,4,4],\
           [0,1,3,3]])
```

```
  1.00   4.00   4.00   4.00
  0.00   0.00   2.00   2.00
  0.00   1.00   3.00   3.00
```

In [4]:
```
def gauss_elim(matrix) :
    row = 0
    row_offset = 0
    col_offset = 0
    no_of_rows = len(matrix)
    if no_of_rows == 0 :
        print "The matrix is empty."
    else :
        no_of_cols = len(matrix[0])
        if no_of_cols == 0 :
            print "The matrix is empty."
        else :
            while row < no_of_rows :
                submatrix = []
                # Copy the submatrix to the right, bottom of (row_offset, col_offset)
                # to submatrix
                for i in range(no_of_rows - row_offset) :
                    row_submatrix = []
                    for j in range(no_of_cols - col_offset) :
                        row_submatrix.append(matrix[i + row_offset][j + col_offset])
                    submatrix.append(row_submatrix)
                    #print_matrix(submatrix, "%6.2f")
                # print row, row_offset, col_offset, submatrix, matrix
                # print "-"*20

                (r, c) = first_non_zero(submatrix)
                if r != -1 :
                    # print "Nothing to do: The (sub)matrix is zero."
                    # else :
                    r += row_offset
                    c += col_offset
                    matrix = elem1(matrix, row_offset, r)
                    matrix = sweep(matrix, (row_offset, c))
                    row_offset += 1
                    col_offset = c+1
                row += 1
    return matrix
```

In [5]:
```
B = read_matrix("files/B.txt")
print_matrix(B, "%6.2f")
print "-"*40
print_matrix(gauss_elim(B), "%6.2f")
print "="*40
C = [[0,0,0,0], [0,1,1,0], [0,0,1,0]]
print_matrix(C, "%3d")
print "-"*40
print_matrix(gauss_elim(C), "%3d")
```

```
  0.00   2.00   3.00   2.00
  1.00   4.00   5.00   5.00
  3.00   0.00   2.00   6.00
----------------------------------------
  1.00   0.00   0.00   1.60
  0.00   1.00   0.00   0.10
  0.00   0.00   1.00   0.60
========================================
  0   0   0   0
  0   1   1   0
  0   0   1   0
----------------------------------------
The matrix is the zero matrix.
  0   1   0   0
  0   0   1   0
  0   0   0   0
```

Part II

Solving a system of linear equations

First suppose we are given an equation of the type $Ax = b$ where A is an $m \times n$ matrix and x is an n-vector and b is an m-vector. We first form the matrix $A : b$ which is obtained by sticking b after A.

In [6]:
```
def join(A, B) :
    """Returns A:B"""
    no_rows_A = len(A)
    no_rows_B = len(B)

    if no_rows_A != no_rows_B :
        print "Matrices should have the same number of rows to be joined."
        retM = None
    else :
        retM = []
        if no_rows_A == 0 :
            retM = A
        else :
            no_cols_A = len(A[0])
            no_cols_B = len(B[0])
            for i in range(no_rows_A) :
                Mrow = []
                for j in range(no_cols_A) :
                    Mrow.append(A[i][j])
                for k in range(no_cols_B) :
                    Mrow.append(B[i][k])
                retM.append(Mrow)
        return retM
```

In [7]:
```
join([[4],[5]], [[4,5],[9,8]])
```

Out [7]:
```
[[4, 4, 5], [5, 9, 8]]
```

To solve $Ax = b$

Gaussian elimination of $A : b$, if A is *invertible* is just finding a matrix P such that $P(A : b)$ is of the form $I_3 : c$. Then P will be the inverse of A and $c = Pb$ will be the solution.

In [8]:
```
# Let A =
A = [[0, 2, 3], [1, 0, 1], [0, 0, 1]]
b = [[5], [2], [1]]
# A:b =
Ab = join(A, b)
print_matrix(gauss_elim(Ab), "%5.2g")
```

```
    1     0     0     1
    0     1     0     1
    0     0     1     1
```

Thus $x = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$.To invert a matrix, one just appends identity on the right and does Gaussian elimination.

In [9]:
```
def idmat(n) :
    """returns the identity matrix of size n"""
    rmat = []
    for i in range(n) :
        row = []
        for j in range(n) :
            if i == j :
                row.append(1)
            else :
                row.append(0)
        rmat.append(row)
    return rmat
```

In [10]:
```
print_matrix(idmat(4), "%2d")
```

```
 1  0  0  0
 0  1  0  0
 0  0  1  0
 0  0  0  1
```

In [11]:
```
def invert(A) :
    size = len(A)
    bigmat = join(A,idmat(size))
    biginv = gauss_elim(bigmat)
    inv = []
    for i in range(size) :
        row = []
        for j in range(size) :
            row.append(biginv[i][j + size])
        inv.append(row)
    return inv
```

In [12]:
```
UT1 = [[1,1,1],[0,1,1],[0,0,1]]
print "Inverse of"
print_matrix(UT1, "%3d")
print "is"
print_matrix(invert(UT1), "%3d")
```

```
Inverse of
  1   1   1
  0   1   1
  0   0   1
is
```

```
1  -1   0
0   1  -1
0   0   1
```