Biweekly quiz 4 direct method for solving linear equation 20191101 ch6

November 4, 2019

[271]:
```
#Gaussian Elims with
#(a)backward substitution
#(b)partial pivoting
#(c)scaled partial pivoting
#(d)complete pivoting

#usage = for biweekly quiz 4 for class PMS.CM Chang @ NCTU AM 11
#Biweekly quiz 4 direct method for solving linear equation 20191101 ch6
#Pactice of Mathematics Software

#author = maxwill lin =  yt lin
#school number = 0712238@NCTU

#last modified 2019.11.4

#show ans with printing out
import numpy as np

eps = 1e-10

#the problem matrix
A0 = np.array([[1,-1,2,-1,-8],
               [2,-2,3,-3,-20],
               [1,1,1,0,-2],
               [1,-1,4,3,4]])
```

[264]:
```
def back_substitution(A):
    A = A.astype(float)
    n = A.shape[0]
    x = np.zeros((n))
    for i in reversed(range(n)):
        x[i] = A[i][n]/A[i][i]
        for j in reversed(range(i)):
            A[j][n] -= A[j][i] * x[i]
    return x
```

```
A = np.array([[1,1,0, 5],[0,1,1, 3],[0,0,1, 1]])
print(A) #sample to varify correctness
back_substitution(A)
#trivial no need to print out
```

```
[[1 1 0 5]
 [0 1 1 3]
 [0 0 1 1]]
```

[264]:
```
array([3., 2., 1.])
```

[265]:
```
def partial_pivoting(A):
    A = A.astype(float)
    n = A.shape[0]
    I = np.eye(n)
    P = np.asarray(list(range(n)))
    print("solve A with partial pivoting with A = \n", A)
    #print(I)
    for i in range(n):
        #find max row
        maxv = abs(A[i][i])
        maxr = i
        for j in range(i+1, n):
            if abs(A[j][i]) > maxv + eps:
                maxv = abs(A[j][i])
                maxr = j
        #no uniq
        if abs(maxv) < eps:
            print("no uniq sol.")
            return None, None, None
        #swap, can use P exchange only, this is naive implementation
        #P version in scaled partial
        print("step {}".format(i))
        if i != maxr:
            print("exchange row {} with row {}".format(i, maxr))
            tmp = A[i].copy()
            A[i] = A[maxr]
            A[maxr] = tmp
            tmp = I[i].copy()
            I[i] = I[maxr]
            I[maxr] = tmp
            P[i], P[maxr] = P[maxr], P[i]
            print(A)

        #elim
        for j in range(i+1, n):
            c = -A[j][i]/A[i][i]
```

```
            for k in range(i, n+1):
                if k == i:
                    A[j][k] = 0
                else:
                    A[j][k] += c * A[i][k]
        print("after forward elims\n", A)
    return A, I, P

#A = np.array([[1,1,1,6],[0,1,1,3],[2,1,0.5,8.5]])
A, _, P = partial_pivoting(A0)
back_substitution(A)
```

```
solve A with partial pivoting with A =
 [[  1.  -1.   2.  -1.  -8.]
 [  2.  -2.   3.  -3. -20.]
 [  1.   1.   1.   0.  -2.]
 [  1.  -1.   4.   3.   4.]]
step 0
exchange row 0 with row 1
[[  2.  -2.   3.  -3. -20.]
 [  1.  -1.   2.  -1.  -8.]
 [  1.   1.   1.   0.  -2.]
 [  1.  -1.   4.   3.   4.]]
after forward elims
 [[  2.   -2.    3.   -3.  -20. ]
 [  0.    0.    0.5   0.5   2. ]
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    2.5   4.5  14. ]]
step 1
exchange row 1 with row 2
[[  2.   -2.    3.   -3.  -20. ]
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    0.5   0.5   2. ]
 [  0.    0.    2.5   4.5  14. ]]
after forward elims
 [[  2.   -2.    3.   -3.  -20. ]
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    0.5   0.5   2. ]
 [  0.    0.    2.5   4.5  14. ]]
step 2
exchange row 2 with row 3
[[  2.   -2.    3.   -3.  -20. ]
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.5   0.5   2. ]]
after forward elims
 [[  2.   -2.    3.   -3.  -20. ]
```

```
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.   -0.4  -0.8]]
step 3
after forward elims
 [[  2.   -2.    3.   -3.  -20. ]
 [  0.    2.   -0.5   1.5   8. ]
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.   -0.4  -0.8]]
```

[265]:
```
array([-7.,  3.,  2.,  2.])
```

[266]:
```
def scaled_partial_pivoting(A):
    A = A.astype(float)
    n = A.shape[0]
    I = np.eye(n)
    P = np.asarray(list(range(n))) #record permutation
    S = [ max(abs(A[i][:-1])) for i in range(n)] #record max val for each row
    print("solve A with scaled partial pivoting with A = \n", A)

    for i in range(n):
        print("step {}".format(i))
        #find max row
        #S = [ max(abs(A[i][:-1])) for i in range(n)] #record max val for each␣
↪row
        #print(S)
        maxv = abs(A[P[i]][i])/S[P[i]]
        #maxr is the original row, for exchange convinience
        maxr = P[i]
        for j in range(i+1, n):
            if abs(A[P[j]][i]/S[P[j]]) > maxv + eps:
                maxv = abs(A[P[j]][i]/S[P[j]])
                maxr = j #notP[j]
        #no uniq
        if abs(maxv) < eps:
            print("no uniq sol.")
            return None, None, None
        #swap using Permutation array to record swap
        if P[i] != maxr:
            print("exchange row {} with row {}".format(P[i], maxr)) #print with␣
↪original index not A[P] index
            tmp = P[i]
            P[i] = P[maxr]
            P[maxr] = tmp
            print(A[P])

        #forward elim
```

```
        for j in range(i+1, n):
            c = -A[P[j]][i]/A[P[i]][i]
            for k in range(i, n+1):
                if k == i:
                    A[P[j]][k] = 0
                else:
                    A[P[j]][k] += c * A[P[i]][k]
        print("after forward elims\n", A[P])
    return A[P], I, P

#A = np.array([[1,1,2,7],[0,1,1,3],[2,1,0.5,8.5]])
A, _, P = scaled_partial_pivoting(A0)
#print(A)
#print(P)
back_substitution(A)
```

```
solve A with scaled partial pivoting with A =
 [[  1.  -1.   2.  -1.  -8.]
 [  2.  -2.   3.  -3. -20.]
 [  1.   1.   1.   0.  -2.]
 [  1.  -1.   4.   3.   4.]]
step 0
exchange row 0 with row 2
[[  1.   1.   1.   0.  -2.]
 [  2.  -2.   3.  -3. -20.]
 [  1.  -1.   2.  -1.  -8.]
 [  1.  -1.   4.   3.   4.]]
after forward elims
 [[  1.   1.   1.   0.  -2.]
 [  0.  -4.   1.  -3. -16.]
 [  0.  -2.   1.  -1.  -6.]
 [  0.  -2.   3.   3.   6.]]
step 1
after forward elims
 [[  1.    1.    1.    0.   -2. ]
 [  0.   -4.    1.   -3.  -16. ]
 [  0.    0.    0.5   0.5   2. ]
 [  0.    0.    2.5   4.5  14. ]]
step 2
exchange row 0 with row 3
[[  1.    1.    1.    0.   -2. ]
 [  0.   -4.    1.   -3.  -16. ]
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.5   0.5   2. ]]
after forward elims
 [[  1.    1.    1.    0.   -2. ]
 [  0.   -4.    1.   -3.  -16. ]
```

```
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.   -0.4  -0.8]]
step 3
after forward elims
 [[  1.    1.    1.    0.   -2. ]
 [  0.   -4.    1.   -3.  -16. ]
 [  0.    0.    2.5   4.5  14. ]
 [  0.    0.    0.   -0.4  -0.8]]
```

[266]:
```
array([-7.,  3.,  2.,  2.])
```

[267]:
```
def complete_pivoting(A):
    A = A.astype(float)
    n = A.shape[0]
    I = np.eye(n) #record permutation as Matrix form
    P = list(range(n)) #record collumn changes

    print("solve A with complete pivoting with A = \n", A)

    for i in range(n):
        #find max elem
        maxv = abs(A[i][i])
        maxid = (i, i)
        for j in range(i, n):
            for k in range(i, n):
                if abs(A[j][k]) > abs(maxv) + eps:
                    maxv = A[j][k]
                    maxid = (j, k)
        #exchange row and col
        if maxid != (i, i):
            print("exchange (row, col) {}, {} with {}, {}".format(i, i,␣
↪maxid[0], maxid[1]))
            for j in range(n+1):
                tmp = A[i][j]
                A[i][j] = A[maxid[0]][j]
                A[maxid[0]][j] = tmp
                if j != n:
                    tmp = I[i][j]
                    I[i][j] = I[maxid[0]][j]
                    I[maxid[0]][j] = tmp

            for j in range(n):
                tmp = A[j][i]
                A[j][i] = A[j][maxid[1]]
                A[j][maxid[1]] = tmp
                tmp = I[j][i]
                I[j][i] = I[j][maxid[1]]
```

```
                I[j][maxid[1]] = tmp
            tmp = P[i]
            P[i] = P[maxid[1]]
            P[maxid[1]] = tmp
            print(A)
        #forward elims
        for j in range(i+1, n):
            c = -A[j][i]/A[i][i]
            for k in range(i, n+1):
                if k == i:
                    A[j][k] = 0
                else:
                    A[j][k] += c * A[i][k]
        print("after forward elims\n", A)

    #reconstruct Pinv
    Pinv = list(range(n))
    for i in range(n):
        for j in range(n):
            if P[j] == i:
                Pinv[i] = j

    return A, I, Pinv
```

[268]:
```
A, I, Pinv = complete_pivoting(A0)
back_substitution(A)[Pinv] #need to recover column
```

```
solve A with complete pivoting with A =
 [[  1.  -1.   2.  -1.  -8.]
 [  2.  -2.   3.  -3. -20.]
 [  1.   1.   1.   0.  -2.]
 [  1.  -1.   4.   3.   4.]]
exchange (row, col) 0, 0 with 3, 2
[[  4.  -1.   1.   3.   4.]
 [  3.  -2.   2.  -3. -20.]
 [  1.   1.   1.   0.  -2.]
 [  2.  -1.   1.  -1.  -8.]]
after forward elims
 [[  4.    -1.     1.     3.     4.  ]
 [  0.    -1.25   1.25  -5.25 -23.  ]
 [  0.     1.25   0.75  -0.75  -3.  ]
 [  0.    -0.5    0.5   -2.5  -10.  ]]
exchange (row, col) 1, 1 with 1, 3
[[  4.     3.     1.    -1.     4.  ]
 [  0.    -5.25   1.25  -1.25 -23.  ]
 [  0.    -0.75   0.75   1.25  -3.  ]
 [  0.    -2.5    0.5   -0.5  -10.  ]]
```

```
after forward elims
 [[  4.           3.           1.          -1.           4.        ]
 [  0.          -5.25         1.25        -1.25       -23.        ]
 [  0.           0.           0.57142857   1.42857143   0.28571429]
 [  0.           0.          -0.0952381    0.0952381    0.95238095]]
exchange (row, col) 2, 2 with 2, 3
[[  4.           3.          -1.           1.           4.        ]
 [  0.          -5.25        -1.25         1.25       -23.        ]
 [  0.           0.           1.42857143   0.57142857   0.28571429]
 [  0.           0.           0.0952381   -0.0952381    0.95238095]]
after forward elims
 [[  4.           3.          -1.           1.           4.        ]
 [  0.          -5.25        -1.25         1.25       -23.        ]
 [  0.           0.           1.42857143   0.57142857   0.28571429]
 [  0.           0.           0.          -0.13333333   0.93333333]]
after forward elims
 [[  4.           3.          -1.           1.           4.        ]
 [  0.          -5.25        -1.25         1.25       -23.        ]
 [  0.           0.           1.42857143   0.57142857   0.28571429]
 [  0.           0.           0.          -0.13333333   0.93333333]]
```

[268]:
```
array([-7.,  3.,  2.,  2.])
```

[ ]:

[ ]: