# Computer exercises week 39, 2011

Jeroen Hofman
10194754

## Exercise 4.2

Code:

```
#Define matrix and the starting vectors
A = np.matrix([[2,3,2],[10,3,4],[3,6,1]])
x = np.array([[0],[0],[1]])
r = np.array([[0],[0],[1]])

#function to compute infinity-norm
def norm(z):
    sum = 0
    for j in range(0,len(z)):
        sum += float(z[[j]])
    return sum

n=1000

#(normed) power iteration
for j in range(0,n):
    y = np.dot(A,x)
    x = y/norm(y)
    if j == n-1:
        print x #eigenvector
        print norm(y) #eigenvalue

#make new matrix with largest eigenvalue removed and
    apply power iteration to that matrix
B = A - np.outer(np.transpose(x),np.array([10,10,13]))

for j in range(0,n):
    y = np.dot(B,r)
    r = y/norm(y)
    if j == n-1:
        print r #eigenvector
        print norm(y) #eigenvalue

#use library routine to calculate eigenvalues and -
    vectors
(w,v) = eig(A)
```

```
print w
print v
```

In this exercise we used power iteration to compute the dominant eigenvalue and corresponding normalized eigenvector for the matrix A. Taking starting vector $x0 = \{0,0,1\}^T$ we obtain the eigenvalue $\lambda = 11.0$ and corresponding eigenvector $x = \{0.2222,0.4444,0.3333\}^T$. Now we deflate out the largest eigenvalue in the matrix A by taking $u = \{10,10,13\}^T$ and computing A' = A - $xu^T$, and then repeating the process, which gives an eigenvalue $\lambda' = -3.0$ and corresponding normalized eigenvector $x' = \{0.1919,0.1111,0.6969\}^T$. If we check this with a library routine which gives, given a matrix A, the eigenvalues and eigenvectors, we see that the eigenvalues and -vectors computed above are indeed correct. Namely the eigenvalues are $\lambda_1 = 11.0, \lambda_2 = -2.0, \lambda_3 = -3.0$ and the corresponding matrix of eigenvectors is:

$$\begin{pmatrix} 3.71390676e^{-01} & 1.82574186e^{-01} & -4.13692033e^{-01} \\ 7.42781353e^{-01} & 3.65148372e^{-01} & -5.54700196e^{-01} \\ 5.57086015e^{-01} & -9.12870929e^{-01} & 8.32050294e^{-01} \end{pmatrix}$$

where the first column is a multiple of $x$ and the third column a multiple of $x'$.

## Exercise 4.3

Code:

```
A = np.matrix([[6,2,1],[2,3,1],[1,1,1]])
x = np.transpose([1,1,1])

#function to compute infinity-norm
def norm(z):
    sum = 0
    for j in range(0,len(z)):
        sum += abs(float(z[[j]]))
    return sum


#function to solve LU x = b, see computer exercise 2.6
def solve(P,L,U,b):
    q = np.array(np.zeros([3]))
    z = np.array(np.zeros([3]))
    b = np.dot(np.transpose(P),b)
    k = 0
    while k < 3:
        q[k] = b[k]
        for l in range(0,k):
            q[k] -= L[k,l]*q[l]
        q[k] = q[k]/L[k,k]
        k += 1

    k = 2
    while k > -1:
```

```
        z[k] = q[k]
        for l in range(k+1,3):
            z[k] -= U[k,l]*z[l]
        z[k] = z[k]/U[k,k]
        k -= 1
    return z


#PLU decomposition of shifted A
(P,L,U) = lu(A-2*np.identity(3))

#inverse iteration
n = 100
for j in range(0,n):
    y = solve(P,L,U,x)
    x = y/norm(y)
    if j==n-1:
        print x #eigenvector
        print 1/norm(y) #eigenvalue

#use library routine to calculate eigenvalues and -
    vectors
(w,v) = eig(A)
print w
print v

#Rayleigh quotient iteration
x = np.transpose([1,1,1])
m = 100

for j in range(0,m):
    sigma = np.dot(np.dot(np.transpose(x),A),x)/np.dot(np
        .transpose(x),x)
    (P,L,U) = lu(A-float(sigma)*np.identity(3))
    y = solve(P,L,U,x)
    x = y/norm(y)
    if j==n-1:
        print x # eigenvector
        print sigma #eigenvalue
```

In this exercise we implement inverse iteration with a shift to compute the eigenvalue nearest to 2. We do this by making an PLU decomposition of A - 2I and then apply inverse iteration, see the code above. In this way we obtain an accurate description for the eigenvalue in less than 10 steps, and we find an eigenvalue of $\lambda = 2.13307447532$ and the corresponding eigenvector of $x = \{-0.3106302, 0.51181407, 0.17755573\}^T$. We then use a library routine to calculate the eigenvalues and -vectors of A, and we obtain the eigenvalues 7.28799214, 2.13307448, 0.57893339 and the corresponding eigenvectors:

$$\begin{pmatrix} 0.86643225 & 0.49742503 & -0.0431682 \\ 0.45305757 & -0.8195891 & -0.35073145 \\ 0.20984279 & -0.28432735 & 0.9354806 \end{pmatrix}$$

so indeed the eigenvalue and -vector obtained by inverse iteration is correct, since the eigenvector $x$ we found is a multiple of the second column of the matrix above.

## Exercise 4.4

We use the code given in the previous exercise to implement Rayleigh quotient iteration. This algorithm gives a correct value for the desired eigenvalue in less than 4 iterations, so the algorithm is very fast and we obtain for the eigenvalue 7.28799214 and for the corresponding normalized eigenvector x = $\{0.56654272,0.29624528,0.137212\}^T$, which is a multiple of the first column of the above matrix.

## Exercise 4.6

Code:

```
A42 = np.matrix([[2,3,2],[10,3,4],[3,6,1]])
A43 = np.matrix([[6,2,1],[2,3,1],[1,1,1]])

n=10
for j in range(0,n):
    sigma = float(A43[2,2])
    (Q,R) = qr(A43-sigma*np.identity(3))
    A43 = np.dot(R,Q) + sigma*np.identity(3)
    print sigma
```

In this exercise we use a simplified version of QR iteration with shifts to quickly find one of the eigenvalues of a matrix. The code is given above: we pick the last element of the matrix as the shift and then compute the QR decomposition of A - $\sigma$I. We then compute our new value of A = RQ + $\sigma$I and we iterate this process. Using the matrix from exercise 4.2 we find that after 8 iterations the eigenvalue of $\lambda = 2.0$ is found and for the matrix from exercise 4.3 we find an eigenvalue of $\lambda = 0.578933385691$ in 4 iterations. Both eigenvalues are the correct eigenvalues also found in exercise 4.2 and 4.3. As a convergence test we can calculate $\frac{\lambda_{k+1}}{\lambda_k}$, where $\lambda_k$ is the eigenvalue obtained after $k$ iteration steps and see if this ratio goes to 1. For the matrix of exercise 4.2 this ratio is 1 after 8 steps, for the matrix of exercise 4.3 the ratio is 1 after 4 steps. This is ofcourse within machine precision, which are just floats in this code.

## Exercise 5.2

Code:

```
#Define functions
def g1(x):
    y = (x*x+2.)/3.
    return y
```

```
def g2(x):
    y = sqrt(3.*x-2.)
    return y
def g3(x):
    y = 3.-2./x
    return y
def g4(x):
    y = (x*x-2.)/(x*2.-3.)
    return y

x1,x2,x3,x4 = 3.,3.,3.,3. #start value of x0 = 3
c2,c3 = 0.75,0.5 #C-values from g2'(2) and g3'(2)
    respectively
n = 20 #number of iterations
x = 2 #root we want to find
for j in range(0,n):
    err2k = abs(x-x2)
    err3k = abs(x-x3)
    x1 = g1(x1)
    x2 = g2(x2)
    x3 = g3(x3)
    x4 = g4(x4)
    err2kp = abs(x-x2)
    err3kp = abs(x-x3)
    if j==n-1:
        r2 = log(err2kp/c2,err2k) #convergence rate for
            g2
        r3 = log(err3kp/c3,err3k) #convergence rate for
            g3
        print "r2 = ",r2," r3 = ",r3
        print "x1 = ",x1," x2 = ",x2," x3 = ",x3," x4 = "
            ,x4
```

We first compute the derivatives of the 4 fixed point iterations corresponding to $f(x) = x^2 - 3x + 2 = 0$ and calculate the derivative in the root x = 2, to find a C-value for these functions:

$$g_1'(x) = \frac{2}{3}x, \quad |g_1'(2)| = \frac{4}{3}$$
$$g_2'(x) = \frac{2}{3\sqrt{3x-2}}, \quad |g_2'(2)| = \frac{3}{4}$$
$$g_3'(x) = \frac{2}{x^2}, \quad |g_3'(2)| = \frac{1}{2}$$
$$g_4'(x) = \frac{2x^2-6x+4}{(2x-3)^2}x, \quad |g_4'(2)| = 0$$

From the above we can conclude that $g_1$ will not converge, but the rest will, and that $g_4$ will converge at least quadratically (so superlinear). We now use the above code to start the fixed point interation at the point $x_0 = 3$, compute the function value at $x_0$, then use that function value for $x_1$, compute the function value for $x_1$, and so on. In this way we obtain after 20 iterations that $x$ diverges

using $g_1$, and the other functions give $x = 2.0000$, so as we argued above, they converge. After 20 iterations we also compute the r-value (the convergence rate) for $g_2$ and $g_3$, which gives $r = 1.00000$ both for $g_2$ and $g_3$, so both these functions give linear convergence.