HW04

November 3, 2015

0.1 ME 500 - Assignment 4 - Brandon Lampe

```
In [3]: from scipy import linalg as LA
        from scipy.sparse import diags as diags
        import numpy as np
        import scipy as sp
        from matplotlib import pyplot as plt

        import sys
        sys.path.append('/Users/Lampe/PyScripts')
        import blfunc as bl

        import ipdb

        np.set_printoptions(precision=3, suppress=False) # precision for numpy operations
        %precision 3
        %matplotlib inline
```

0.2 Problem 3

```
In [4]: A = diags([-1,2,-1],[-1,0,1], shape=(5,5)).toarray()
        A[0,0] = 1
        print A
```

```
[[ 1. -1.  0.  0.  0.]
 [-1.  2. -1.  0.  0.]
 [ 0. -1.  2. -1.  0.]
 [ 0.  0. -1.  2. -1.]
 [ 0.  0.  0. -1.  2.]]
```

0.2.1 3 (a)

Finde the eigenvalues and eigenvectors of $[A]$

```
In [5]: eig, Mo = LA.eig(A)
        eig_min = np.real(min(eig))
        eig_max = np.real(max(eig))
```

```
In [6]: eig_diag = eig * np.eye(5)
```

check to ensure $\Big[[A] - \lambda_1[I]\Big]\{e^1\} = 0$

```
In [7]: print (A - eig[0]*np.eye(5)).dot(Mo[:,0])
```

```
[  9.992e-16+0.j   1.554e-15+0.j   5.551e-16+0.j   1.665e-15+0.j
  -3.331e-16+0.j]
```

(i)

```
In [8]: print np.real(eig_diag)
```

```
[[ 3.683  0.     0.     0.     0.   ]
 [ 0.     2.831  0.     0.     0.   ]
 [ 0.     0.     0.081  0.     0.   ]
 [ 0.     0.     0.     1.715  0.   ]
 [ 0.     0.     0.     0.     0.69 ]]
```

```
In [33]: print Mo
```

```
[[-0.17  -0.326 -0.597  0.456 -0.549]
 [ 0.456  0.597 -0.549 -0.326 -0.17 ]
 [-0.597 -0.17  -0.456 -0.549  0.326]
 [ 0.549 -0.456 -0.326  0.17   0.597]
 [-0.326  0.549 -0.17   0.597  0.456]]
```

(ii)

```
In [34]: MoT = np.transpose(Mo)
         print Mo.dot(MoT)
```

```
[[  1.000e+00   5.013e-16   8.546e-17  -3.753e-16  -2.776e-17]
 [  5.013e-16   1.000e+00   4.905e-16  -2.285e-16   1.943e-16]
 [  8.546e-17   4.905e-16   1.000e+00  -3.905e-16   3.053e-16]
 [ -3.753e-16  -2.285e-16  -3.905e-16   1.000e+00  -5.551e-17]
 [ -2.776e-17   1.943e-16   3.053e-16  -5.551e-17   1.000e+00]]
```

(iii)

```
In [35]: A_star = Mo.dot(eig_diag).dot(MoT)
         print A_star
```

```
[[  1.000e+00+0.j  -1.000e+00+0.j   7.494e-16+0.j  -1.110e-15+0.j
   -8.327e-17+0.j]
 [ -1.000e+00+0.j   2.000e+00+0.j  -1.000e+00+0.j  -8.743e-16+0.j
    1.270e-15+0.j]
 [  7.494e-16+0.j  -1.000e+00+0.j   2.000e+00+0.j  -1.000e+00+0.j
    1.457e-15+0.j]
 [ -1.055e-15+0.j  -8.743e-16+0.j  -1.000e+00+0.j   2.000e+00+0.j
   -1.000e+00+0.j]
 [ -1.943e-16+0.j   1.270e-15+0.j   1.402e-15+0.j  -1.000e+00+0.j
    2.000e+00+0.j]]
```

(iv)

```
In [36]: A_star_inv = Mo.dot(LA.inv(eig_diag)).dot(MoT)
         print A_star_inv
```

```
[[ 5.+0.j  4.+0.j  3.+0.j  2.+0.j  1.+0.j]
 [ 4.+0.j  4.+0.j  3.+0.j  2.+0.j  1.+0.j]
 [ 3.+0.j  3.+0.j  3.+0.j  2.+0.j  1.+0.j]
 [ 2.+0.j  2.+0.j  2.+0.j  2.+0.j  1.+0.j]
 [ 1.+0.j  1.+0.j  1.+0.j  1.+0.j  1.+0.j]]
```

(v)

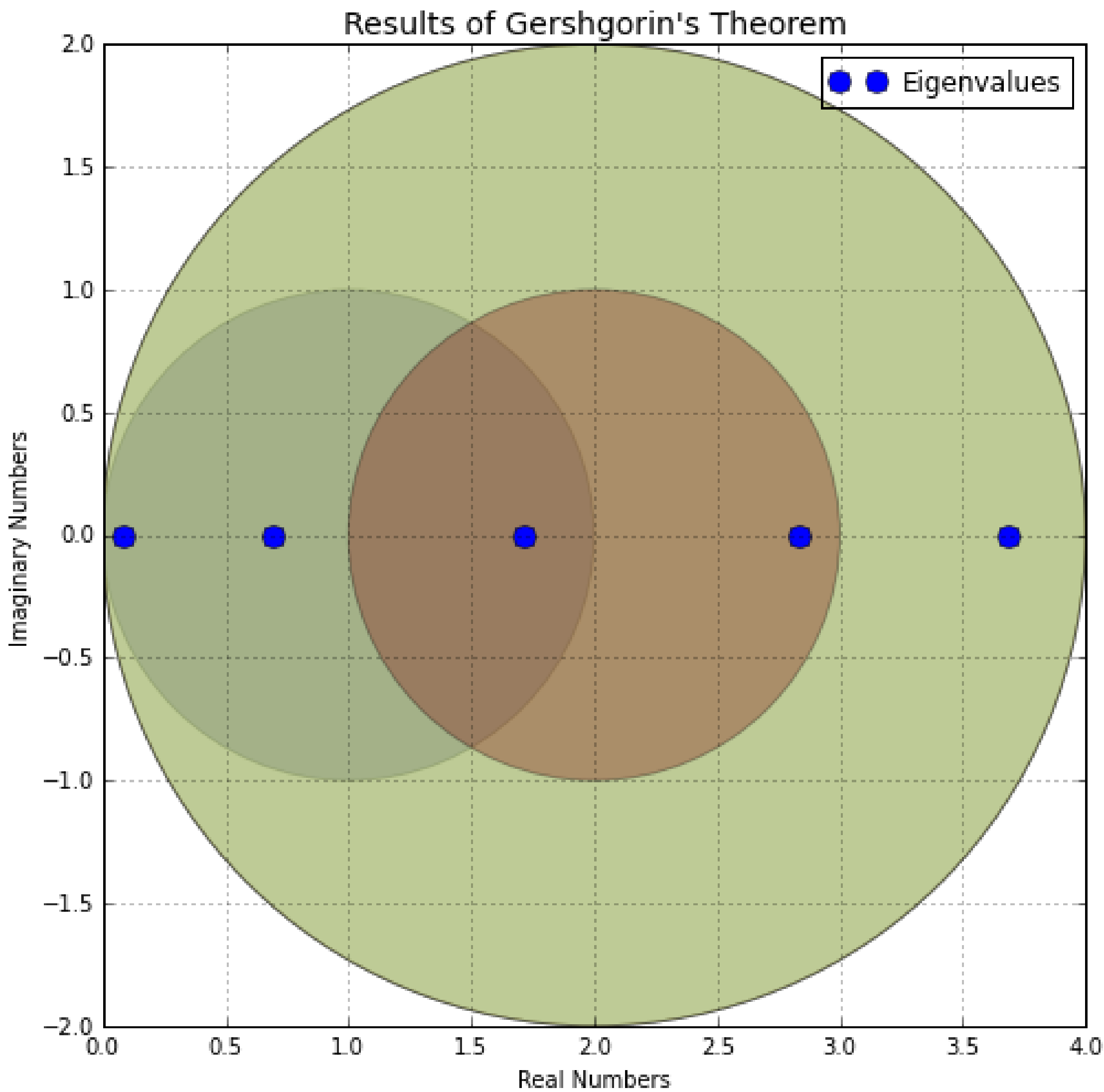
```
In [37]: print A_star_inv.dot(A_star)
```

```
[[  1.000e+00+0.j  -3.469e-17+0.j   6.509e-15+0.j  -1.166e-14+0.j
    7.772e-15+0.j]
 [ -9.437e-16+0.j   1.000e+00+0.j   6.287e-15+0.j  -1.177e-14+0.j
    8.660e-15+0.j]
 [ -1.388e-15+0.j   2.186e-15+0.j   1.000e+00+0.j  -9.437e-15+0.j
    7.772e-15+0.j]
 [ -8.049e-16+0.j  -2.567e-16+0.j   2.734e-15+0.j   1.000e+00+0.j
    4.663e-15+0.j]
 [ -7.216e-16+0.j   9.506e-16+0.j   1.735e-15+0.j  -3.664e-15+0.j
    1.000e+00+0.j]]
```

0.2.2 3 (b)

Gersgorin's theorem to obtain bounds on eigenvalues of $[A]$

```
In [38]: nrow = A.shape[0]
         ncol = A.shape[1]
         center = np.diagonal(A)
         radius = np.zeros(ncol)

         for i in xrange(nrow):
             for j in xrange(ncol):
                 if i != j:
                     radius[i] = radius[i] + np.abs(A[i,j])

         fig_gersh, ax = plt.subplots(figsize = (8,8))
         ax.plot(eig, np.zeros(nrow), 'o',markersize = 10, label="Eigenvalues")

         ax.legend(loc=0); # upper left corner
         ax.set_xlabel('Real Numbers')
         ax.set_ylabel('Imaginary Numbers')
         ax.set_title('Results of Gershgorin\'s Theorem' , fontsize = 14)
         ax.grid(b = True, which = 'minor')
         ax.grid(b = True, which = 'major')
         ax.set_ylim(-2, 2)
         ax.set_xlim(0,4)

         bl.circles(x=center, y=np.zeros(nrow), s=radius, c=np.arange(nrow), ax=ax, alpha=0.3)

         fig_name = 'plot_3b.pdf'
         path = '/Users/Lampe/Documents/UNM_Courses/ME-500/HW04/'
         fig_gersh.savefig(path + fig_name)
         # show()
```



0.2.3 3 (c)

Obtain the Rayleigh quotient for $< v >=< 1, 2, 3, 2, 1 >$

In [39]:
```
v = np.array([1,2,3,2,1])
```

In [40]:
```
num = v.dot(A).dot(v)
den = v.dot(v)
R = num/den
R
```

Out[40]:
```
0.263
```

0.2.4 3 (d)

Compute Norms

```
In [41]: p1 = np.linalg.norm(A, ord = 1)
         p2 = np.linalg.norm(A, ord = 2)
         pInf = np.linalg.norm(A, ord = np.inf)
         print '%.3e' %p1
         print '%.3e' %p2
         print '%.3e' %pInf
```

```
4.000e+00
3.683e+00
4.000e+00
```

```
In [42]: print '%.3e' %np.real(eig_min)
         print '%.3e' %np.real(eig_max)
```

```
8.101e-02
3.683e+00
```

0.2.5 3 (e)

The condition number of [A]

```
In [43]: cond = eig_max / eig_min
         print '%.3e' %cond
```

```
4.546e+01
```

0.2.6 3 (f)

Obtain a 1, 2, and 3-mode solution

```
In [44]: x_ex = np.array([1,2,3,4,5])
         b_ex = A.dot(x_ex)
         print b_ex
```

```
[-1.  0.  0.  0.  6.]
```

approximations for $\{x^{ex}\}$ using all 5 modes of [A]

```
In [45]: nrow = eig.shape[0]
         eig_vect_arr = np.zeros((nrow,nrow))
         eig_val_arr = np.zeros(nrow)
         A_inv_ap = np.zeros((nrow,nrow))
         x_ap = np.zeros((nrow,nrow))
         error = np.zeros(nrow)

         for i in xrange(nrow):
             A_inv_ap = 1./np.real(eig[i]) * np.outer(Mo[:,i],Mo[:,i]) + A_inv_ap
             x_ap[:,i] = A_inv_ap.dot(b_ex)
             error[i] = np.linalg.norm(x_ap[:,i] - x_ex,2)/np.linalg.norm(x_ex,2)

         print x_ap
         print error
```

```
[[ 0.082 -0.334  2.778  3.609  1.   ]
 [-0.221  0.542  3.402  2.808  2.   ]
 [ 0.29   0.072  2.449  1.449  3.   ]
 [-0.266 -0.848  0.852  1.161  4.   ]
 [ 0.158  0.859  1.745  2.833  5.   ]]
[  9.979e-01   9.829e-01   6.867e-01   6.413e-01   7.400e-15]
```

Gram-Schmidt method for obtaining eigenpairs

In [46]:
```
#G-S for eigenpairs
eig_vect_old = np.array([2,2,2,2,5])
eig_val_old = eig_vect_old.dot(A).dot(eig_vect_old)/(eig_vect_old.dot(eig_vect_old))**0.5
A_inv_ap_old = np.zeros((5,5))
x_ap_arr = np.zeros((5,5))

eig_vect_arr = np.zeros((5,5))
eig_val_arr = np.zeros(5)
alpha_arr = np.zeros(5)

error_vect = np.zeros(5)
neg_terms = np.zeros(5)

tol = 0.001
iter_max = 100
error = 10
mode_max = x_ex.shape[0]

for h in xrange(mode_max):
    if h == 0:
        for i in xrange(iter_max): # obtain lowest eigenpair by reverse iteration
            x_star = bl.QR_solve(A,eig_vect_old)
            eig_vect_new = x_star / np.sqrt(x_star.dot(x_star))
            eig_val_new = (eig_vect_new.dot(A).dot(eig_vect_new))/den
            error = np.abs(eig_val_new - eig_val_old)/np.abs(eig_val_new)

            beta = eig_vect_new.dot(b_ex)
            x_ap = beta/eig_val_new * eig_vect_new

            eig_vect_old = eig_vect_new
            eig_val_old = eig_val_new

            if error <= tol:
                eig_vect_arr[:,h] = eig_vect_new
                eig_val_arr[h] = eig_val_new
                break

    else:
        for i in xrange(iter_max):
            x_hat = bl.QR_solve(A,eig_vect_old) # x hat
            alpha_arr[h-1] = eig_vect_arr[:,h-1].dot(x_hat)

            neg_terms = 0
            for j in xrange(mode_max):
                neg_terms = neg_terms + alpha_arr[j]*eig_vect_arr[:,j]
#                 print neg_terms

            x_star = x_hat - neg_terms # apply G-S, corrected for previous eigenvectors
            eig_vect_new = x_star / np.sqrt(x_star.dot(x_star))
            den = eig_vect_new.dot(eig_vect_new) # D
            eig_val_new = (eig_vect_new.dot(A).dot(eig_vect_new))/den
            error = np.abs(eig_val_new - eig_val_old)/np.abs(eig_val_new)
```

```
                eig_vect_old = eig_vect_new
                eig_val_old = eig_val_new

                if error <= tol:
                    eig_vect_arr[:,h] = eig_vect_new
                    eig_val_arr[h] = eig_val_new
                    break

        for k in xrange(h+1):
            beta = eig_vect_arr[:,k].dot(b_ex)
            x_ap = beta/eig_val_arr[k] * eig_vect_arr[:,k]
            x_ap_arr[:,h] = x_ap_arr[:,h] + x_ap

        error_vect[h] = np.linalg.norm(A.dot(x_ap_arr[:,h]) - b_ex,2)/np.linalg.norm(b_ex,2)

    print eig_vect_arr
    print eig_val_arr
    print x_ap_arr
    print error_vect
```

```
[[ 0.596  0.544 -0.596 -0.556  0.593]
 [ 0.548  0.173 -0.548 -0.172  0.547]
 [ 0.456 -0.32  -0.456  0.325  0.458]
 [ 0.327 -0.598 -0.327  0.593  0.33 ]
 [ 0.17  -0.462 -0.17   0.451  0.173]]
[ 0.004  0.69   0.081  0.69   0.081]
[[ 59.545  56.928  60.057  57.43   60.693]
 [ 54.748  53.914  56.792  55.977  58.988]
 [ 45.527  47.064  49.457  50.995  53.515]
 [ 32.605  35.479  37.192  39.996  41.813]
 [ 17.008  19.23   20.123  22.254  23.207]]
[ 1.603  1.508  1.567  1.663  1.724]
```

0.2.7 4

Analysis of Hilbert Matrices

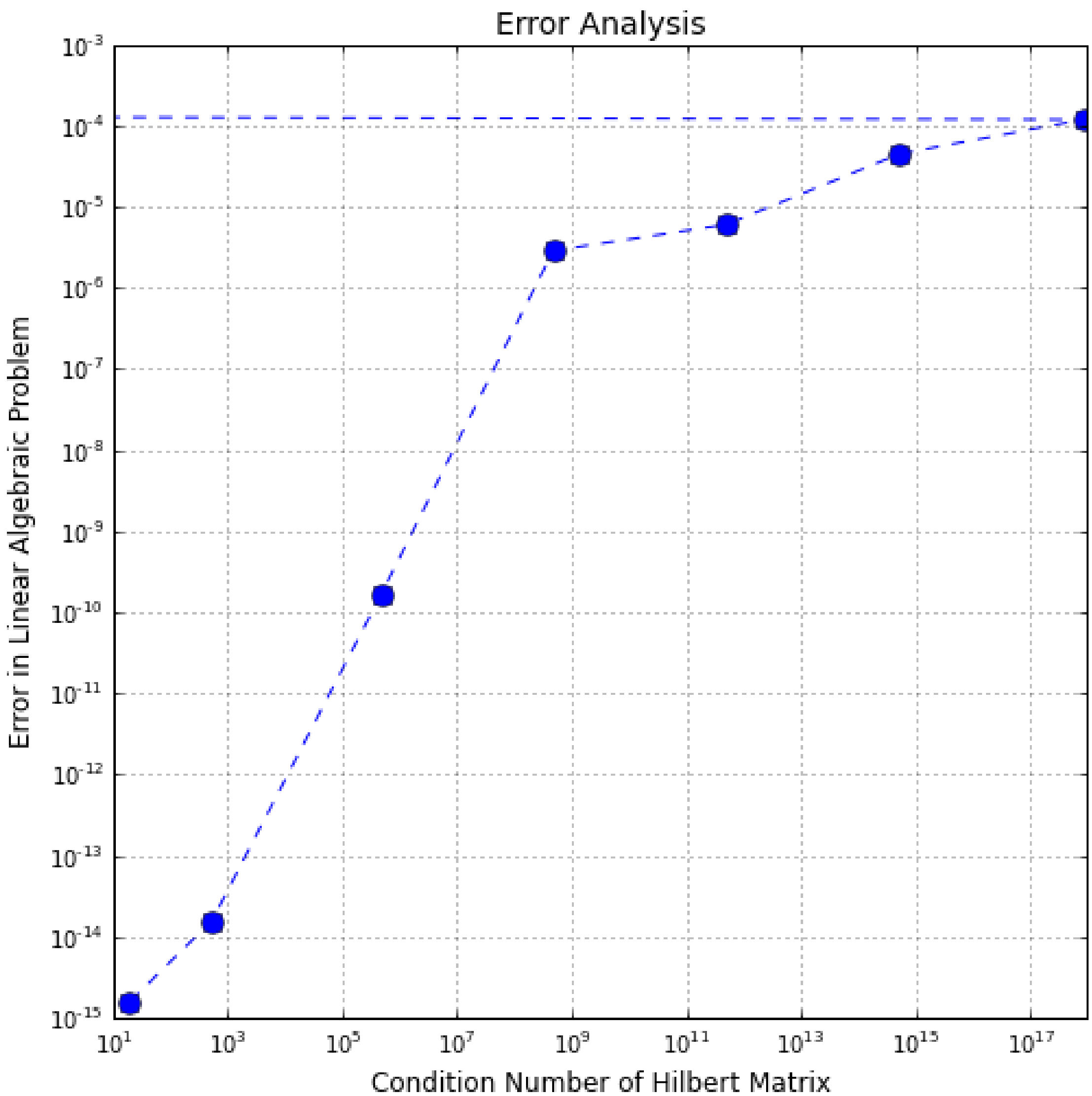
```
In [98]: rng = np.array((2,3,5,7,9,11,13,15))
         analysis = np.zeros((rng.shape[0], 2))
         inc = 0

         for i in rng:
             H = LA.hilbert(i)
             val, Mo = LA.eig(H)
             text = "Size of Hilber Matix: " + str(i)
             print text
             print "Eigenvalues:"
             print np.real(val)
             c = max(np.real(val))/min(np.real(val))
             print 'Condition Number'
             print '%e' %c
             x_ex = np.arange(i)
             b = H.dot(x_ex)
             x_ap = bl.QR_solve(H,b)
```

```
            error = np.linalg.norm(H.dot(x_ap) - b)/np.linalg.norm(b)
            print "approximate solution and error"
            out = str(x_ap) + " error: " + str(error)
            print out
            print
            analysis[inc,0] = c
            analysis[inc,1] = error
            inc = inc + 1
```

```
Size of Hilber Matix: 2
Eigenvalues:
[ 1.268  0.066]
Condition Number
1.928147e+01
approximate solution and error
[ -6.752e-15   1.000e+00] error: 1.57039437051e-15

Size of Hilber Matix: 3
Eigenvalues:
[ 1.408  0.122  0.003]
Condition Number
5.240568e+02
approximate solution and error
[ -1.168e-12   1.000e+00   2.000e+00] error: 1.56539504238e-14

Size of Hilber Matix: 5
Eigenvalues:
[  1.567e+00   2.085e-01   1.141e-02   3.059e-04   3.288e-06]
Condition Number
4.766073e+05
approximate solution and error
[ -1.353e-06   1.000e+00   2.000e+00   3.000e+00   4.000e+00] error: 1.67635118738e-10

Size of Hilber Matix: 7
Eigenvalues:
[  1.661e+00   2.719e-01   2.129e-02   1.009e-03   2.939e-05   4.857e-07
   3.494e-09]
Condition Number
4.753674e+08
approximate solution and error
[  1.096e+00  -4.410e+01   4.460e+02  -1.751e+03   3.258e+03  -2.834e+03
   9.455e+02] error: 2.84920789322e-06

Size of Hilber Matix: 9
Eigenvalues:
[  1.726e+00   3.216e-01   3.104e-02   1.979e-03   8.758e-05   2.673e-06
   5.386e-08   6.461e-10   3.500e-12]
Condition Number
4.931537e+11
approximate solution and error
[  1.602e+00  -6.010e+01   5.751e+02  -2.201e+03   4.080e+03  -3.664e+03
   1.378e+03  -6.298e+01  -9.741e+00] error: 5.95702344812e-06

Size of Hilber Matix: 11
```

```
Eigenvalues:
[  1.775e+00   3.624e-01   4.031e-02   3.114e-03   1.774e-04   7.542e-06
   2.372e-07   5.368e-09   8.283e-11   7.807e-13   3.399e-15]
Condition Number
5.221964e+14
approximate solution and error
[  9.610e+00  -3.522e+02   3.237e+03  -1.227e+04   2.260e+04  -2.046e+04
   7.845e+03  -4.589e+02  -6.220e+01  -1.788e+01  -7.197e+00] error: 4.43459592315e-05

Size of Hilber Matix: 13
Eigenvalues:
[  1.814e+00   3.968e-01   4.903e-02   4.349e-03   2.952e-04   1.562e-05
   6.466e-07   2.076e-08   5.077e-10   9.141e-12   1.144e-13   8.892e-16
   2.042e-18]
Condition Number
8.883830e+17
approximate solution and error
[  3.078e+01  -1.174e+03   1.144e+04  -4.740e+04   9.904e+04  -1.087e+05
   5.863e+04  -1.129e+04  -3.642e+02  -7.949e+01  -2.795e+01  -1.262e+01
  -6.671e+00] error: 0.000118244924399

Size of Hilber Matix: 15
Eigenvalues:
[  1.846e+00   4.266e-01   5.721e-02   5.640e-03   4.365e-04   2.711e-05
   1.362e-06   5.529e-08   1.803e-09   4.658e-11   9.322e-13   1.394e-14
   1.421e-16   8.051e-18  -1.052e-17]
Condition Number
-1.754630e+17
approximate solution and error
[  1.188e+02  -4.647e+03   4.676e+04  -2.020e+05   4.444e+05  -5.208e+05
   3.074e+05  -6.961e+04  -1.102e+03  -2.294e+02  -7.887e+01  -3.508e+01
  -1.834e+01  -1.071e+01  -6.791e+00] error: 0.000364001386973
```

In [99]:
```python
fig_Hilb, ax = plt.subplots(figsize = (8,8))

ax.plot(analysis[:,0],analysis[:,1], 'o--', markersize=10)

# ax.legend(loc=0); # upper left corner
ax.set_xlabel('Condition Number of Hilbert Matrix', fontsize = 12)
ax.set_ylabel('Error in Linear Algebraic Problem', fontsize = 12)
ax.set_title('Error Analysis' , fontsize = 14)
ax.grid(b = True, which = 'major')
ax.grid(b = True, which = 'major')
# ax.set_ylim(-2, 2)
# ax.set_xlim(0,4)
ax.set_xscale('log')
ax.set_yscale('log')

fig_name = 'plot_4.pdf'
path = '/Users/Lampe/Documents/UNM_Courses/ME-500/HW04/'
fig_gersh.savefig(path + fig_name)
# show()
```


Error Analysis
Error in Linear Algebraic Problem
Condition Number of Hilbert Matrix
10^-3
10^-4
10^-5
10^-6
10^-7
10^-8
10^-9
10^-10
10^-11
10^-12
10^-13
10^-14
10^-15
10^1
10^3
10^5
10^7
10^9
10^11
10^13
10^15
10^17

```
In [ ]:
```