

Math 426/CISC 410 08F, All Sections

R.J. Braun

Homework 10 Solutions, Hints and Answers

Problem 4.6.2. (a) Assume that A is n -by- n and nonsingular. The SVD is $A = USV^*$; here S is a diagonal matrix with the singular values on the diagonal, and U and V are orthogonal. $A^{-1} = (USV^*)^{-1} = (V^*)^{-1}S^{-1}U^{-1} = VS^{-1}U^*$. For nonsingular A , then $\sigma_i \neq 0$, $i = 1, 2, \dots, n$ and the diagonal elements of S^{-1} are $1/\sigma_i$. Using (4.18), we know that the norm of a matrix is the largest singular value, and so $\|A^{-1}\| = 1/\sigma_n$. From the definition of the condition number, $\kappa(A) = \|A\|\|A^{-1}\| = \sigma_1/\sigma_n$.

(b) Write A and A_k the same way:

$$A = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^* + \cdots + \sigma_{n-1} \mathbf{u}_{n-1} \mathbf{v}_{n-1}^* + \sigma_n \mathbf{u}_n \mathbf{v}_n^*,$$

$$A_k = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^* + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^* + 0 \mathbf{u}_{k+1} \mathbf{v}_{k+1}^* + \cdots + 0 \mathbf{u}_{n-1} \mathbf{v}_{n-1}^* + 0 \mathbf{u}_n \mathbf{v}_n^*.$$

Here the \mathbf{u}_i and \mathbf{v}_i are columns of U and V respectively. Each product of the two vectors is an n -by- n matrix. Then

$$A - A_k = 0 \mathbf{u}_1 \mathbf{v}_1^* + 0 \mathbf{u}_2 \mathbf{v}_2^* + \cdots + 0 \mathbf{u}_k \mathbf{v}_k^* + \sigma_{k+1} \mathbf{u}_{k+1} \mathbf{v}_{k+1}^* + \cdots + \sigma_n \mathbf{u}_n \mathbf{v}_n^*.$$

This is an SVD that has a bunch of zeros for k singular values. The norm of this matrix is then the biggest singular value, viz., $\|A - A_k\| = \sigma_{k+1}$.

Problem 4.6.3 \mathbf{X} requires $480 \times 500 = 2.4 \times 10^5$ numbers (one number between 0 and 255 inclusive per pixel). To produce the compressed image $\mathbf{X1}$, we needed 40 columns of \mathbf{U} and \mathbf{V} , and there was a 40-by-40 \mathbf{S} matrix. When multiplied together, then we get back the same number of pixels as \mathbf{X} , so that is not a good way to compress. We should keep only \mathbf{U} , \mathbf{V} and the 40 singular values from the diagonal of \mathbf{S} . In that case we have $480 \times 40 + 40 + 500 \times 40 = 3.924 \times 10^4$ numbers. Thus, $\mathbf{X1}$ requires about 6 times less numbers.

Problem 5.1.1. The following will work.

```
d = ones(200,3);
d(:,2) = -2;
A = spdiags(d,-1:1,200,200);
```

For the other matrix we could do

```
d = ones(200,3); d(:,2) = -2;
B = spdiags(d,-1:1,200,200);
B(1,200) = 1; B(200,1) = 1;
```

We could also use the following approach for the second matrix:

```
e = ones(200,1);
B = spdiags([e e -2*e e e],[-199 -1:1 199],200,200);
```

Problem 5.1.2. The table below shows the fill-in that occurs for this simple operation. Following the contents of this diary are the sparsity plots for different powers of B.

```
>> Prob5_1_2
```

Matrix	Fraction of nonzeros				
	B	L	U	Q	R
B	0.0500	0.1150	0.1225	0.6422	0.2789
B ²	0.1167	0.2789	0.2789	0.7731	0.4356
B ⁴	0.3833	0.4356	0.4428	0.9272	0.5081
B ⁸	0.9833	0.5081	0.5081	0.9997	0.5083

```
% There is rapid fill-in for raising B to a power.
% There is also fillin for L and U, but not for Q and R.
```

```
>> type Prob5_1_2.m
```

```
% Script: Prob5_1_2.m
% playing around with the buckyball adjacency matrix

close all; clear all;
format short;

[B,v]=bucky;
gplot(B,v); axis equal;

[m,n] = size(B);
ratio = nnz(B)/m/n;
figure
subplot(2,2,1); spy(B)

disp(' ')
disp('          Fraction of nonzeros          ')
disp(sprintf('Matrix      B      L      U      Q      R      '));
disp('-----');
[L,U,P] = lu(B);
nnl = nnz(L)/m/n;
nnu = nnz(U)/m/n;
[Q,R] = qr(bucky);
nnq = nnz(Q)/m/n;
nnr = nnz(R)/m/n;
disp(sprintf('B          %5.4f  %5.4f  %5.4f  %5.4f  %5.4f  ',...
            ratio,nnl,nnu,nnq,nnr));

for k = 2:4
```

```

B = B*B; ratio = nnz(B)/m/n;
subplot(2,2,k); spy(B);
[L,U,P] = lu(B);
nnl = nnz(L)/m/n;
nnu = nnz(U)/m/n;
[Q,R] = qr(B);
nnq = nnz(Q)/m/n;
nnr = nnz(R)/m/n;
disp(sprintf('B^%1.0f      %5.4f  %5.4f  %5.4f  %5.4f  %5.4f  ',...
            2^(k-1),ratio,nnl,nnu,nnq,nnr));

```

end

```
disp(' ');
```

```
>> diary
```

Problem 5.2.1. >> % Diary for Problem 5.2.1

```
>> % part (a)
```

```
>> A = [1.1 1; 0 2.1]
```

```
A =
```

```

1.1000000000000000  1.0000000000000000
                   0  2.1000000000000000

```

```
>> format short e
```

```
>> [gamma,x] = poweriter(A,20);
```

```
>> x
```

```
x =
```

```

-1.0000e+000
-1.0000e+000

```

```
>> gamma
```

```
gamma =
```

```
Columns 1 through 6
```

```
2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000
```

```
Columns 7 through 12
```

```
2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000
```

```
Columns 13 through 18
```

```
2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000  2.1000e+000
```

```
Columns 19 through 20
```

```
2.1000e+000  2.1000e+000
```

```
>> err = abs(gamma-2.1)
```

```
err =
```

```
Columns 1 through 13
```

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
Columns 14 through 20
```

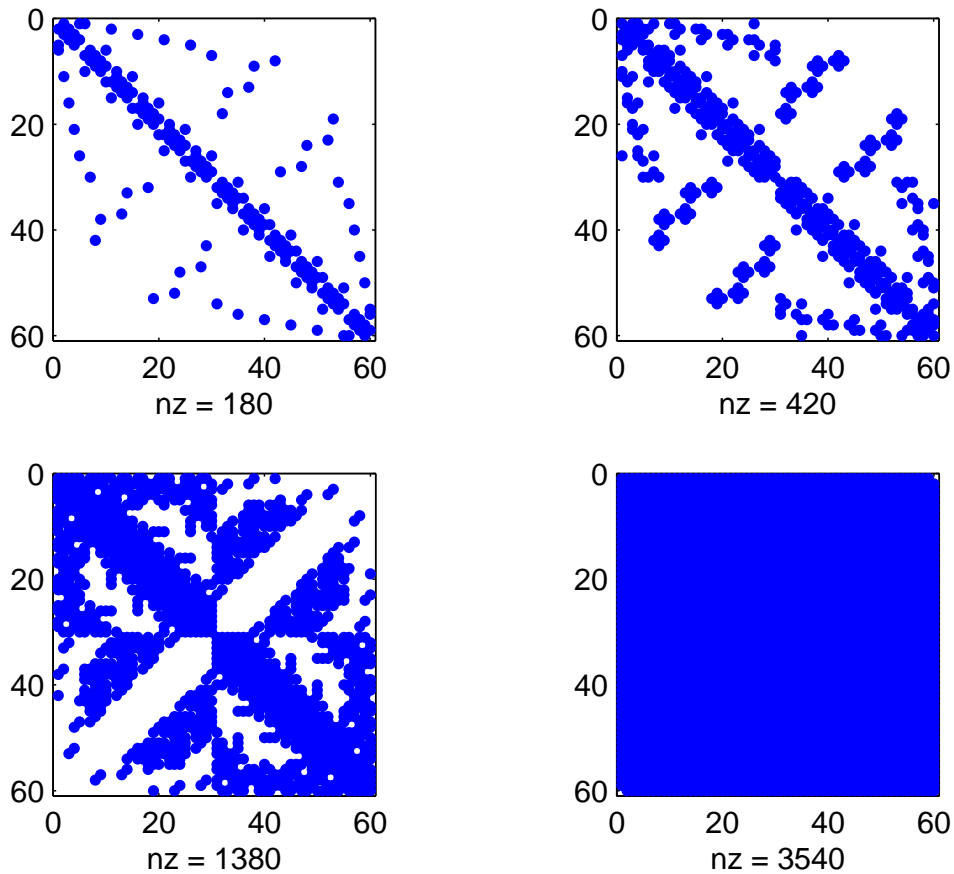


Figure 1: Sparsity plots of powers of B.

```

0 0 0 0 0 0 0
>> % convergence is infinitely faster than linear for this tiny matrix.
>> % part (c)
>> A = [6 5 4; 5 4 3; 4 3 2];
>> [gamma,x] = poweriter(A,20);
>> [V,D] = eig(A)
V =
 5.8268e-001  4.0825e-001  7.0272e-001
-1.1119e-001 -8.1650e-001  5.6654e-001
-8.0506e-001  4.0825e-001  4.3036e-001
D =
-4.8074e-001  0  0
 0  3.2200e-018  0
 0  0  1.2481e+001
>> % Largest eigenvalue is 12.481, with e-vector that is last column of V.

```

```

>> x
x =
-1.0000e+000
-8.0621e-001
-6.1242e-001
>>V(:,3)-(-x/norm(x))
ans =
 1.1102e-016
 4.4409e-016
 0
>> % clearly we have a good approximation to the eigenvector!
>> err = abs(gamma-D(3,3))
err =
Columns 1 through 6
3.1594e+001 1.4611e+000 6.3742e-002 2.4428e-003 9.4111e-005 3.6250e-006
Columns 7 through 12
1.3963e-007 5.3784e-009 2.0717e-010 7.9794e-012 3.0909e-013 1.2434e-014
Columns 13 through 18
0 0 0 0 0 0
Columns 19 through 20
0 0
>> % The answer is reached at the lucky 13th iteration.
>> % Now see if there is a constant factor in error reduction with each iteration.
>> ratios = err(2:12)./err(1:11)
ratios =
Columns 1 through 6
4.6246e-002 4.3626e-002 3.8323e-002 3.8526e-002 3.8518e-002 3.8519e-002
Columns 7 through 11
3.8519e-002 3.8518e-002 3.8517e-002 3.8736e-002 4.0230e-002
>> % typical factor for reduction of the error is about 0.0385;
>> % the last number in particular is likely polluted by roundoff error.
>> % The first few numbers aren't far enough out in the sequence yet.
>> diary

```