CS 421: Fall 1997

Solutions to Assignment 4

1. Using the SVD to compute a least squares fit for planetary orbit:

```
% PROBLEM 1

% Put data into column vectors
x0=[1.02 0.95 0.87 0.77 0.67 0.56 0.44 0.30 0.16 0.01]';
y0=[0.39 0.32 0.27 0.22 0.18 0.15 0.13 0.12 0.13 0.15]';

%%%%%%%%%%%%%%%%%%%% PART (a)

% Compute the SVD of least squares matrix
n=length(x0);
M=[y0.^2 x0.*y0 x0 y0 ones(n,1)];
b=x0.^2;
[U,S,V]=svd(M)
s=diag(S);

%%%%%%%%%%%%%%%%%%%% PART (b)

% Determine parameters by using from 1 to 5 biggest singular values
% and plot the resulting orbit
w=U'*b;
for k=5:-1:1
    p=V(:,1:k)*(w(1:k)./s(1:k));
    a=p(1); b=p(2); c=p(3); d=p(4); e=p(5);

    fprintf('k=%d:\n',k)
    fprintf('    a=%7.4f  b=%7.4f  c=%7.4f  d=%7.4f  e=%7.4f\n',p)

    subplot(3,2,6-k)
    plot(x0,y0,'o')
    hold on
    PlotOrbit(a,b,c,d,e,'-');
    title(['k = ',num2str(k)])
    hold off
end

%%%%%%%%%%%%%%%%%%%% PART (c)

% Random perturbation
x0=x0+0.01*(rand(n,1)-0.5);
y0=y0+0.01*(rand(n,1)-0.5);
fprintf('\n\nPerturbed problem\n\n')

% Repeat part (b) with perturbed data
M=[y0.^2 x0.*y0 x0 y0 ones(n,1)];
b=x0.^2;
[U,S,V]=svd(M);
s=diag(S);
w=U'*b;
```

```
for k=5:-1:1
    p=V(:,1:k)*(w(1:k)./s(1:k));
    a=p(1); b=p(2); c=p(3); d=p(4); e=p(5);

    fprintf('k=%d:\n',k)
    fprintf('    a=%7.4f  b=%7.4f  c=%7.4f  d=%7.4f  e=%7.4f\n',p)

    subplot(3,2,6-k)
    hold on
    PlotOrbit(a,b,c,d,e,'g--');
    title(['k = ',num2str(k)])
    hold off
end
```

The plotting function: (It is of course not necessary to make it this fancy.)

```
function PlotOrbit(a,b,c,d,e,linestyle)

% Plot the orbit given by the equation
%   a*y^2 + b*x*y + c*x + d*y + e = x^2
% First the range of x is determined.  Then the equation
% is solved for y given x in this range.

A=b^2+4*a; B=2*b*d-4*a*c; C=d^2-4*a*e; D=B^2-4*A*C;
if (A<0) & (D>0)  % x is in a bounded interval
    xmin=(-B+sqrt(D))/(2*A);  xmax=(-B-sqrt(D))/(2*A);
    x=linspace(xmin,xmax,100);
    B=b*x+d; C=c*x+e-x.^2; D=B.^2-4*a*C;
    y1=(-B-sqrt(D))/(2*a);  y2=(-B+sqrt(D))/(2*a);
    plot([x fliplr(x)],real([y1 fliplr(y2)]),linestyle)
elseif (A>0) & (D>0)  % x is outside a bounded interval
    xmin=(-B-sqrt(D))/(2*A);  xmax=(-B+sqrt(D))/(2*A);
    x=linspace(xmin-2,xmin,50);   % left branch
    B=b*x+d; C=c*x+e-x.^2; D=B.^2-4*a*C;
    y1=(-B-sqrt(D))/(2*a);  y2=(-B+sqrt(D))/(2*a);
    plot([x fliplr(x)],real([y1 fliplr(y2)]),linestyle)
    hold on
    x=linspace(xmax,xmax+2,50);   % right branch
    B=b*x+d; C=c*x+e-x.^2; D=B.^2-4*a*C;
    y1=(-B-sqrt(D))/(2*a);  y2=(-B+sqrt(D))/(2*a);
    plot([fliplr(x) x],real([fliplr(y1) y2]),linestyle)
    hold off
    axis tight
elseif (A>0) & (D<0)  % defined for all x
    x=linspace(-2,2,100);
    B=b*x+d; C=c*x+e-x.^2; D=B.^2-4*a*C;
    y1=(-B-sqrt(D))/(2*a);  y2=(-B+sqrt(D))/(2*a);
    plot(x,y1,linestyle,x,y2,linestyle);
end
xlabel('x')
ylabel('y')
```

The output for the above program:

```
U =

  Columns 1 through 7

    0.3918   -0.4222    0.5675   -0.3703    0.3960    0.0239    0.0489
    0.3750   -0.3201    0.1514    0.0960   -0.4586   -0.0917   -0.2198
    0.3588   -0.2234   -0.0839    0.3191   -0.3300    0.0381    0.1259
    0.3401   -0.1124   -0.2623    0.3350    0.0140    0.2865    0.4147
    0.3226   -0.0094   -0.3503    0.1603    0.2820   -0.6416   -0.4120
    0.3046    0.0953   -0.3454   -0.0885    0.3529    0.6343   -0.3744
    0.2862    0.2028   -0.2572   -0.3286    0.1500   -0.2776    0.6389
    0.2658    0.3225   -0.0805   -0.4674   -0.2835   -0.0395   -0.2071
    0.2466    0.4367    0.1797   -0.1614   -0.3849    0.1130   -0.0207
    0.2264    0.5583    0.4819    0.5036    0.2621   -0.0455    0.0055

  Columns 8 through 10

    0.1429    0.1661   -0.0362
   -0.4441   -0.4837   -0.1631
    0.1800    0.3197    0.6721
    0.3428   -0.0589   -0.5617
    0.0416    0.2381   -0.1675
   -0.2698   -0.0470    0.1700
   -0.3667   -0.1634    0.1841
    0.6000   -0.3314    0.0760
   -0.2509    0.6117   -0.3028
    0.0241   -0.2513    0.1291


S =

    3.7860         0         0         0         0
         0    0.9449         0         0         0
         0         0    0.2089         0         0
         0         0         0    0.0230         0
         0         0         0         0    0.0055
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0


V =

    0.0464   -0.0940    0.3459   -0.3250    0.8739
    0.1341   -0.3141    0.5899   -0.5524   -0.4799
    0.5188   -0.7484   -0.4027    0.0581    0.0729
    0.1805   -0.1418    0.6085    0.7595    0.0168
    0.8235    0.5589    0.0048   -0.0948   -0.0207
```

```
k=5:
    a=-2.6356  b= 0.1436  c= 0.5514  d= 3.2229  e=-0.4329
k=4:
    a=-0.6528  b=-0.9451  c= 0.7168  d= 3.2610  e=-0.4800
k=3:
    a= 0.4192  b= 0.8769  c= 0.5253  d= 0.7561  e=-0.1674
k=2:
    a= 0.1044  b= 0.3401  c= 0.8918  d= 0.2025  e=-0.1718
k=1:
    a= 0.0190  b= 0.0547  c= 0.2117  d= 0.0737  e= 0.3361


Perturbed problem

k=5:
    a=-0.9068  b=-0.6854  c= 0.6921  d= 3.1125  e=-0.4628
k=4:
    a=-0.5203  b=-0.8891  c= 0.7236  d= 3.1103  e=-0.4709
k=3:
    a= 0.4174  b= 0.8727  c= 0.5279  d= 0.7526  e=-0.1694
k=2:
    a= 0.1051  b= 0.3424  c= 0.8921  d= 0.2016  e=-0.1724
k=1:
    a= 0.0193  b= 0.0553  c= 0.2124  d= 0.0744  e= 0.3366
```

The relative change of the parameter vector caused by the perturbation is:

k	1	2	3	4	5
Change	0.32%	0.27%	0.49%	5.86%	45.59%

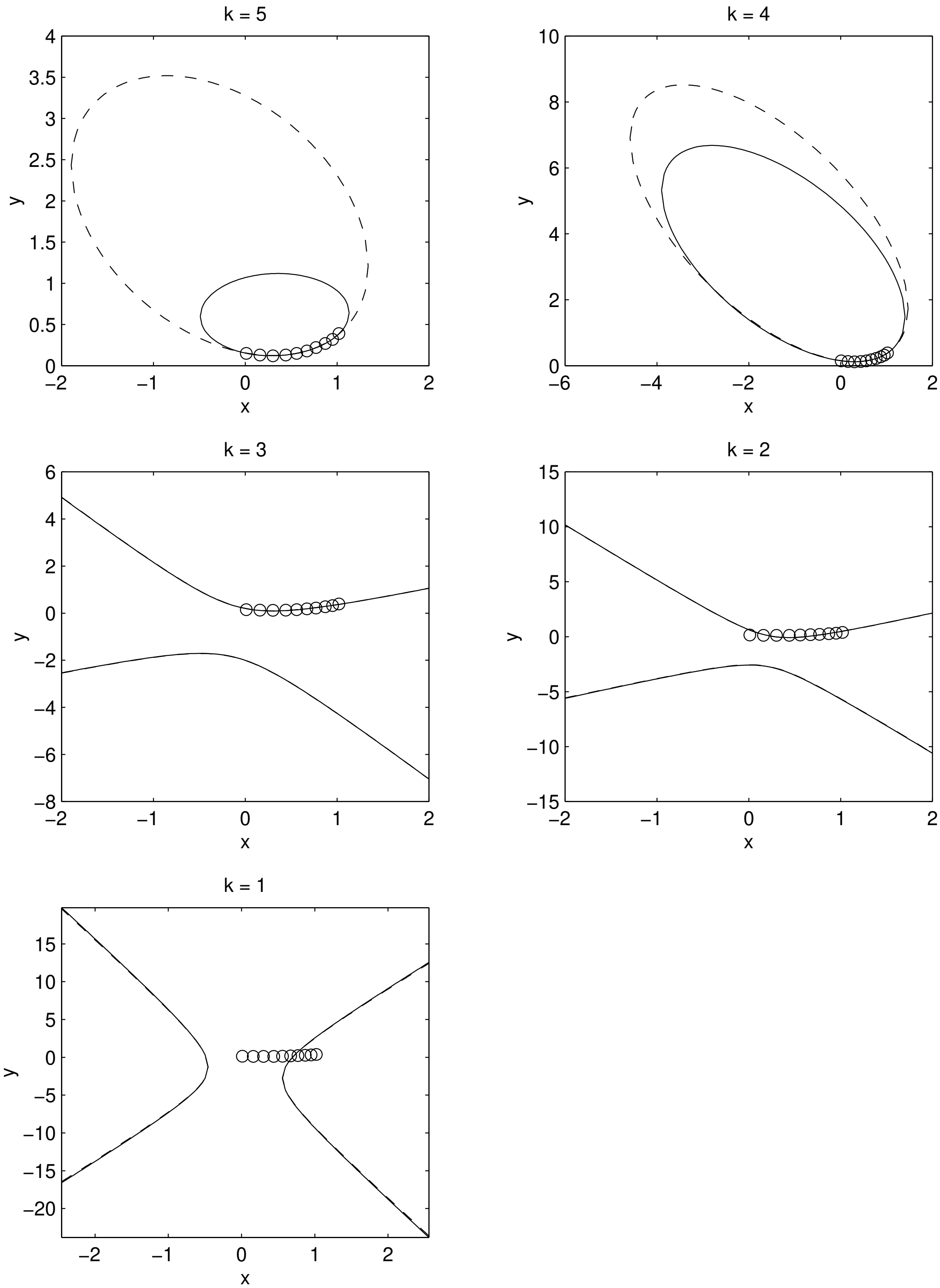
This can also be seen in the plots. The effect of the perturbation is big for $k = 5$, less for $k = 4$ and hardly noticable for $k = 1, 2, 3$.

This can be explained by looking at the condition number, which is equal to the largest singular value divided by the smallest one. When we only use the k biggest singular values, the condtion number for the numerical procedure is σ_1/σ_k. This number decreases as k decreases, making the problem better conditioned.

However, by throwing away singular values, we might be throwing away valuable information. Also, the effect of this is the same as making the smallest singular values equal to zero. Doing this can be reasonable if these values are considerably smaller than the rest, and if they are very sensitive to perturbations. But the difference in magnitude is not that big here.

So which is better, a solution that fits the data more closely, or one that is less sensitvie to perturbation? In general, it depends on the accuracy of the observed data. If it is known to be very accurate, a close fit might be preferrable. Otherwise, closeness can be sacrificed to get less sensitivity. (The solution should though remain reasonably close.)

For this problem, the solutions for $k = 5$ and $k = 4$ are the only plausible ones, since we know the orbit to be an ellipse. But to argue which one to take, we should know more about the accuracy of the data. (Data in planetary science is often fairly accurate.)



Solid line: Original solution. Dashed line: Perturbed solution.

2. (a) The inner product $v^T u = u^T v$ is a scalar, so

$$Au = (uv^T)u = u(v^T u) = (v^T u)u = (u^T v)u$$

which shows $u^T v$ is an eigenvalue of A.

(b) Suppose λ is an eigenvalue of A and x is an corresponding eigenvector. Then

$$Ax = \lambda x \quad \Rightarrow \quad uv^T x = \lambda x \quad \Rightarrow \quad (v^T x)u = \lambda x.$$

If x is not a multiple of u, then we must have $\lambda = 0$ and $v^T x = 0$. So the other eigenvalues are zero and the corresponding eigenvectors are all vectors orthogonal to v.

(c) Let x_0 be an arbitrary nonzero starting vector. The next iterate is

$$x_1 = Ax_0 = uv^T x_0 = (v^T x_0)u$$

which is an eigenvector corresponding to the dominant eigenvalue $u^T v$ (unless $v^T x_0 = 0$, but rounding errors usually guide the algorithm from this special case). Only one iteration of the power method was needed.

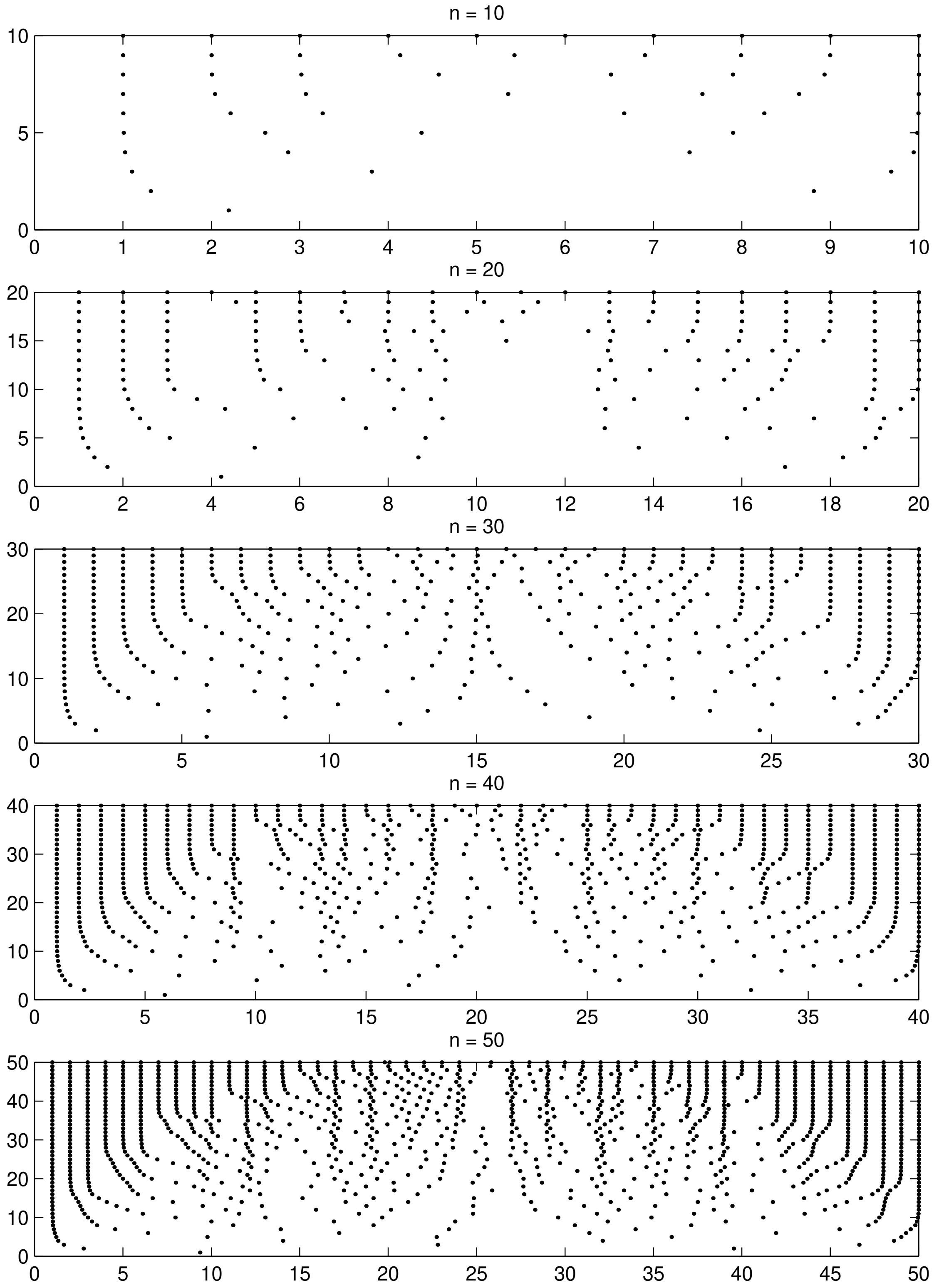
Note: If $u^T v \neq 0$, it is a nonzero eigenvalue with multiplicity one and 0 is an eigenvalue with multiplicty $n - 1$. If $u^T v = 0$, all the eigenvalues are zero, but the gemoetric multiplicity of 0 is only $n - 1$.

3. % PROBLEM 3

```
for n=10:10:50
    % Generate a random symmetric matrix with
    % eigenvalues 1,2,...,n
    subplot(5,1,n/10)
    B=rand(n,n);
    [Q,R]=qr(B);
    A=Q*diag(1:n)*Q';

    % Use Lanczos iteration
    r0=rand(n,1);
    [alpha,beta]=Lanczos(A,r0,n);
    T=diag(alpha)+diag(beta,-1)+diag(beta,1);

    % Plot the Ritz values at each iteration
    for k=1:n
        d=eig(T(1:k,1:k));
        plot(d,k*ones(k,1),'.');
        hold on
    end
    axis([0 n 0 n])
    title(['n = ',num2str(n)])
    hold off
end
```

The following function has the Lanczos method as given in Section 4.3.9 of Heath. Since in Matlab we can't use zero as index, the indices of beta are offset by one in the loop.

```
function [alpha,beta]=Lanczos(A,r,m)
% Apply m steps of Lanczos iteration to A with starting
% vector r.  The function returns the diagonal alpha and
% the sub/super-diagonal beta of the m-by-m tridiagonal matrix.

n=length(r);
alpha=zeros(m,1);
beta=zeros(m+1,1);

beta(1)=norm(r);
q=zeros(n,1);
for k=1:m
    q_old=q;
    q=r/beta(k);
    u=A*q;
    r=u-beta(k)*q_old;
    alpha(k)=q'*r;
    r=r-alpha(k)*q;
    beta(k+1)=norm(r);
end
beta=beta(2:m);
```


n = 10
0 5 10
0 1 2 3 4 5 6 7 8 9 10
n = 20
0 5 10 15 20
0 2 4 6 8 10 12 14 16 18 20
n = 30
0 10 20 30
0 5 10 15 20 25 30
n = 40
0 10 20 30 40
0 5 10 15 20 25 30 35 40
n = 50
0 10 20 30 40 50
0 5 10 15 20 25 30 35 40 45 50