Problem P12: (a) Write a Matlab function for Richardson iteration, with the following signature,

function z = richardson(A, b, x0, omega)

It should return the Nth iterate x_n as z. Confirm that it works by showing you get the same x_3 as on page 4 of the slides.

Solution:
Recall that the formula for Richardson iteration is given by,

$$x_{k+1} = x_k + \omega(b - Ax_k).$$

The following is a Matlab code which performs this iterations, as well as a test run confirming that it gets the same values for x_3 as our class slides.

Code:

```
function z = richardson(A, b, x0, N, omega)
% This function takes in a system Ax = b
% an initial iterate x0, a residual scaling factor omega and a number of
% iterations N and performs richardson iteration
% using x_{k + 1} = x_k + \omega(b - Ax_k).

z = x0;
for i = 1:N
    z = z + (omega.*(b - A*z));
end
end
```

Console:

```
>> A = [2 1 0; 0 2 1; 1 0 3];
>> b = [2 1 4]';
>> z = richardson(A, b, [0 0 0]', 3, .2)

z =
    0.7280
    0.0880
    1.0960
```

(b) How many iteration are needed to get 8 digit accuracy for LS1 with $x_0 = 0$ and using the preferred value of ω. How many iterations for $\omega = .1$ and $\omega = .5$?

Solution:
I am choosing to interpret "8 digits of accuracy" as number of iterations for each term in x_N to achieve 8 digits of accuracy. Recall that the preferred value of ω for LS1 from the slides was .4. For $\omega = .4$ we found that it took $N = 19$ iterations to get 8 digits of accuracy, for $\omega = .1$ it took $N = 83$ iterations and for $\omega = .5$ it took $N = 52$ iterations.

Console:

```
>> zp = [A\b]';
>> [Hist, z] = richardson(A, b, [0 0 0]', 50, .4);
>> Error = abs(zp - Hist);
>> Error(15:20,:)

   1.0e-05 *

   0.197967872006544   0.332726271998483   0.566509568011853
   0.093496934394643   0.293149081602438   0.034114764790871
   0.135959019509357   0.044983910407469   0.030575820786360
   0.045185368058309   0.003233546233274   0.048268443664234
   0.007743655117132   0.019954086714348   0.008420458486036
   0.006432903654208   0.007359000735731   0.001413370354086 <- (N = 19)


>> [Hist, z] = richardson(A, b, [0 0 0]', 100, .1);
>> Error = abs(zp - Hist);
>> Error(80:84,:)

   1.0e-06 *

   0.259224390841695   0.013727141624369   0.183266967379581
   0.206006798508795   0.029308410033207   0.154209316205467
   0.161874597837119   0.038867659647306   0.128547201105889
   0.125612912293960   0.043948847828628   0.106170500613345
   0.096095445045741   0.045776128333313   0.086880641569920 <- (N = 83)


>> [Hist, z] = richardson(A, b, [0 0 0]', 100, .5);
>> Error = abs(zp - Hist);
>> Error(49:53,:)

   1.0e-06 *

   0.158190520238577   0.231839475617335   0.339777265878638
   0.115919737808667   0.169888632939319   0.248983893058607
   0.084944316469659   0.124491946529304   0.182451815433637
   0.062245973264652   0.091225907716819   0.133698065951648
   0.045612953858409   0.066849032975824   0.097972019830195 <- (N = 52)
```

Problem P13: (a) Write a Matlab function which do N iterations of the Jacobi and Gauss-Seidel methods:

function z = jacobi(A, b, x0, N)

function z = gs(A, b, x0, N)

For each of these use the entries of A directly. That is, for jacobi(), implement formula (5) from the slides, and for gs() implement formula (7).

Solution:
Below are matlab implementations of Jacobi and Gauss-Seidel iteration. They were verified using test cases from MAA [1][2]

Code:

```
function [Hist, z] = jacobi(A, b, x0, N)
% This function takes in a system Az = b
% an initial iterate x0 and a number of
% iterations N and performs jacobi iteration,
% using

z = x0';
Hist = z;
[m, n] = size(A);

for i = 1:N
    % Building next iterate with jacobi iteration
    zk = [];
    for j = 1:m
        % There's probably a better way to implement this.
        zk = [zk ((b(j) - ((A(j, :)*z') - (A(j, j)*z(j))))/A(j, j))];
    end
    % Setting and storing next iterate
    z = zk;
    Hist = [Hist; z];
end
end
------------------------------------------------------------------


function [Hist, z] = gs(A, b, x0, N)
% This function takes in a system Az = b
% an initial iterate x0 and a number of
% iterations N and performs Gauss-Seidel iteration,
% using

z = x0;
Hist = z';
[m, n] = size(A);

for i = 1:N
    % Building next iterate with Gauss-Seidel iteration
    for j = 1:n
        % Trick for saving memory is inplace solving of next
        % iterate with A(j,1:j-1)*z(1:j-1)
        z(j) =(b(j) - A(j,1:j-1)*z(1:j-1) - A(j,j+1:n)*z(j+1:n))/A(j,j);
    end
    % Store the updated solution in the history matrix
```

```
        Hist = [Hist; z'];
    end
    end
```

(b) For each method, How many iterations are needed to get 8 digit accuracy for LS1 using $x_0 = 0$?

Solution:
Using the same interpretation of '8 digit accuracy' as before we found that on LS1 Jacobi iteration took $N = 20$ iterations and Gauss-Seidel took $N = 14$ iterations,

Console:

```
>> zp = [A\b]';
>> [Hist, z] = jacobi(A, b, [0 0 0]', 30);
>> Error = abs(zp - Hist);
>> Error(19:21,:)

   1.0e-06 *

   0.334897976683735                   0   0.334897976572712
                   0   0.167448988286356   0.111632658894578
   0.083724494226445   0.055816329558311                   0 <- (N = 20)



>> [Hist, z] = gs(A, b, [0 0 0]', 30);
>> Error = abs(zp - Hist);
>> Error(12:15,:)

   1.0e-05 *

                   0   0.200938786010241                   0
   0.100469393005120                   0   0.033489797657271
                   0   0.016744898828636                   0
   0.008372449411542                   0   0.002790816466813 <- (N = 14)
```

(c) Demonstrate that GS fails on LS2. Now compute an explanatory spectral radius.

Solution:
From the following console output we can see that applying Gauss-Seidel iterations to LS2, we do *NOT* see the error converge.

Console:

```
>> A2 = [1  2  3  0;
         2  1 -2 -3;
        -1  1  1  0;
```

```
        0  1  1  -1];

>> b2 = [7 1 1 3]';
>> z2p = A2\b2

>> [Hist, z] = gs(A2, b2, [0 0 0 0]', 5);
>> Error = abs(z2p' - Hist);

           1           0           2           1
           6          13          19           6
          31         118         149          31
         211         813        1024         211
        1446        5573        7019        1446
        9911       38198       48109        9911
```

Recall from the lecture slides that Gauss-Seidel iteration converges if and only if,

$$\rho((D - L)^{-1}U) < 1$$

Computing the spectral radius of this $(D-L)^{-1}U$ matrix we get that $\rho((D-L)^{-1}U) \approx 6$ as expected.

Console:

```
>> U = diag(diag(A2)) - triu(A2); % Computes U from A
>> M = inv(tril(A2))*U % Computes (D - L)^{-1}U matrix

     0    -2    -3     0
     0     4     8     3
     0    -6   -11    -3
     0    -2    -3     0

>> rho = max(abs(eig(M))) % Find spectral radius

   6.854101966249689
```

Problem P14: Show that Jacobi iteration converges if A is strictly diagonally-dominant.

Proof. Let $Ax = b$ be a linear system with A decomposing as $A = D - L - U$ and D is diagonal, L is strictly lower triangular, and U is strictly upper triangular. Suppose A is strictly diagonally-dominant. Recall that Jacobi iteration converges if and only if $\rho(M) < 1$ where $M = D^{-1}(L + U)$. Since A is strictly diagonally-dominant we know that,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

Now let λ and v such that $D^{-1}(L+U)v = \lambda v$ and consider $v_i \in v$ such that $|v_i| \geq |v_j|$ for all j. Note that since $D^{-1} = 1/D$

$$\left(\frac{\sum_{j\neq i} a_{ij}}{a_{ii}}\right)v_i = \lambda v_i,$$
$$\left|\frac{\sum_{j\neq i} a_{ij}}{a_{ii}}v_i\right| = |\lambda v_i|,$$
$$\frac{\left|\sum_{j\neq i} a_{ij}\right|}{|a_{ii}|}|v_i| = |\lambda v_i|,$$
$$\frac{\sum_{j\neq i} |a_{ij}|}{|a_{ii}|}|v_i| \geq |\lambda v_i|.$$

Since A is strictly diagonally dominant we know that,

$$1 > \frac{\sum_{j\neq i} |a_{ij}|}{|a_{ii}|}$$

and therefore we conclude that $|v_i| > |\lambda v_i|$

$\square$

Problem P15: (a) Consider this boundary value problem from **P10** on Assignement #2:

$$u''(x) + qu(x) = f(x), \qquad u(x_L) = \alpha, \qquad u(x_R) = \beta$$

Implement the centered finite difference method for this problem. Your code should have the signature,

function [x, u] = bvpq(m, xL, xR, q, f, alpha, beta)

where the input f is a function $f(x)$, but the other inputs are integers or real number. The outputs are the grid vector x and the approximate solution vector u. In this initial implementation, your code should use Matlab's backslash command.

Solution:
Consider the following matlab code,
Code:

```
function [x,U] = bvpq(m, xL, xR, q, f, alpha, beta)
% This function uses a centered finite difference scheme to solve
% the following boundary value problem:
%
```

```
%   u''(x) + qu(x) = f(x), u(x_L) = \alpha, u(x_R) = \beta
%
% Using m + 2 point grid and the following finite difference approx.
%
%    D^2U_j = 1/h^2(U_{j - 1} -2U_j +U_{j + 1})
%


x = linspace(xL, xR, m+2); % Generating grid
h = (xR - xL)/(m - 1); % Grid spacing
% Generating A (epic one-liner)
A = (1/h^2).*(
     diag(((q*h^2 - 2)*ones(m, 1)))
            + diag(ones(m - 1, 1), 1)
            + diag(ones(m - 1, 1), -1)
            );

% Generating F
F = f(x(2:m+1))';
F(1) = F(1) - alpha/h^2;
F(m) = F(m) - beta/h^2;

% Solving U
U = A\F;
U = [alpha; U; beta];
```

(b) Check correctness of bvpq() by solving the problem,

$$u''(x) - u(x) = f(x), \qquad u(0) = 1, \qquad u(2) = 0$$

exactly, using the solution $u_{ex}(x) = 1 - \sin(\frac{\pi}{4}x)$. That is, start by finding the $f(x)$ for which $u_{ex}(x)$ is the solution. Show a figure which verifies your code.

Solution:
First we must solve for the desired $f(x)$. Note that

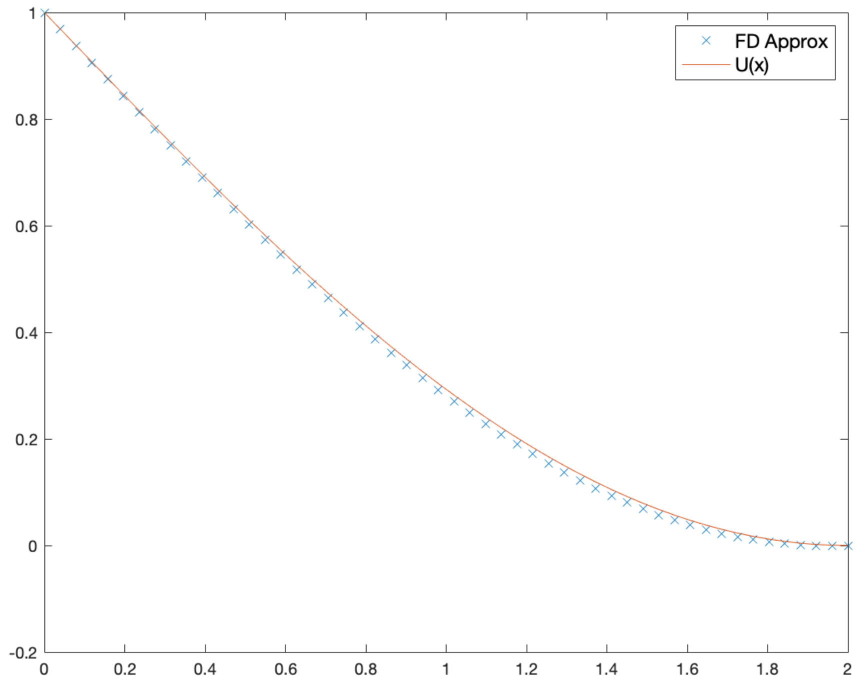
$$u''_{ex}(x) = \frac{\pi^2}{16}\sin\left(\frac{\pi}{4}x\right)$$

With some algebra we find that,

$$\begin{aligned} f(x) &= u''(x) - u(x) \\ &= \frac{\pi^2}{16}\sin\left(\frac{\pi}{4}x\right) - \left(1 - \sin\left(\frac{\pi}{4}x\right)\right), \\ &= \frac{\pi^2}{16}\sin\left(\frac{\pi}{4}x\right) + \sin\left(\frac{\pi}{4}x\right) - 1, \\ &= \left(\frac{\pi^2}{16} + 1\right)\sin\left(\frac{\pi}{4}x\right) - 1. \end{aligned}$$

Plugging in our function $f(x)$, $q = -1$ and our boundary conditions into our bvpq() routine we get the following plot,

Figure 1: Plot of our centered FD Approx. vs. $u(x)$



Console:

```
>> u = @(x) 1 - sin((pi/4)*x);
>> f = @(x) ((pi/4)^2 + 1)*sin((pi/4)*x) - 1;
>> [x,U] = bvpq(50, 0, 2, -1, f, 1, 0);

>> plot(x, U, 'x')
>> plot(x, u(x))
>> legend('FD Approx', 'U(x)', 'Location','northeast','FontSize', 12)
```

(c) Your part (a) code sets up and solves a linear system $AU = F$. For what q values is A strictly diagonally dominant(SDD)?

Proof. Recall that the matrix A has the form,

$$A = \frac{1}{h^2}\begin{bmatrix} (qh^2-2) & 1 & & & & \\ 1 & (qh^2-2) & 1 & & & \\ & 1 & (qh^2-2) & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & (qh^2-2) & 1 \\ & & & & 1 & (qh^2-2) \end{bmatrix}.$$

To be strictly diagonally dominant for all i

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|.$$

Therefore it follows that A is SDD when $|qh^2 - 2| > 2$ and thus either $q > 4/h^2$ or $q < 0$. □

(d) Duplicate the code from part (a), give it a new name bvpqgs(), and implement Gauss-Seidel to solve the linear system, instead of calling the built-in linear solver. Use the problem given in part (b) and for each of $m = 5$ and $m = 50$ find nonzero values q where Gauss-Seidel does converge and does not converge. When convergence happens, how many iterations give 8 digit accuracy?

Solution:
Recall that SDD is a sufficient condition for Gauss-Seidel iteration to converge. Therefore by part (c) for the problem described in part (a) we know that for $m = 5$ GS iteration will converge if,

$$q > 16 \qquad q < 0.$$

and we know that for $m = 50$, GS iteration will converge if,

$$q > 49^2 \qquad q < 0.$$

Now consider the following matlab code bvpqgs() which replaces the backslash operator with a Gauss-Seidel iteration sub-routine to solve the linear system,

Code:

```
function [Hist,x,U] = bvpqgs(m, xL, xR, q, f, alpha, beta, x0, N)
% This function uses a centered finite difference scheme to solve
% the following boundary value problem:
%
%   u''(x) + qu(x) = f(x), u(x_L) = \alpha, u(x_R) = \beta
%
% Using m + 2 point grid and the following finite difference approx.
%
%    D^2U_j = 1/h^2(U_{j - 1} -2U_j +U_{j + 1})
%
% This function also solves the Linear System AU = F via
% Gauss-Seidel iteration, so the user must pass an initial iterate x0
% and a number of iterations.



x = linspace(xL, xR, m+2); % Generating grid
h = (xR - xL)/(m - 1); % Grid spacing
% Generating A
A = (1/h^2).*(
```

```
    diag(((q*h^2 - 2)*ones(m, 1))) +
        diag(ones(m - 1, 1), 1) +
        diag(ones(m - 1, 1), -1)
        );
% Generating F
F = f(x(2:m+1))';
F(1) = F(1) - alpha/h^2;
F(m) = F(m) - beta/h^2;



% Solving U
[Hist, U] = gs(A, F, x0, N);
U = [alpha; U; beta];
Hist = [alpha*ones(N+1, 1) Hist beta*ones(N+1, 1)]

end



function [Hist, z] = gs(A, b, x0, N)
% This function takes in a system Az = b
% an initial iterate x0 and a number of
% iterations N and performs Gauss-Seidel iteration,
% using

z = x0;
Hist = z';
[m, n] = size(A);

for i = 1:N
    % Building next iterate with Gauss-Seidel iteration
    for j = 1:n
        % Trick for saving memory is
        % inplace solving of next iterate with A(j,1:j-1)*z(1:j-1)
        z(j) =(b(j) - A(j,1:j-1)*z(1:j-1) - A(j,j+1:n)*z(j+1:n))/A(j,j);
    end
    % Store the updated solution in the history matrix
    Hist = [Hist; z'];

end
end
```

First we will show that for the indicated values of q, the problem from part (b) does converge. The following console output shows convergence of $m = 5$ with $q = 20$ within $N = 14$ iterations,

Console:

```
>> f = @(x) ((pi/4)^2 + 1)*sin((pi/4)*x) - 1;
>> [Hist, x,U] = bvpqgs(5, 0, 2, 20, f, 1, 0, zeros(1, 5)', 35);
>> Hist

Hist =
```

```
1            0            0            0            0            0   0
1  −0.381794029  0.111300104  −0.025159552  0.041739301  0.032900019   0
1  −0.418894064  0.132053300  −0.045990384  0.037716239  0.034241040   0
1  −0.425811796  0.141302821  −0.047732537  0.037849950  0.034196469   0
1  −0.428894970  0.142911264  −0.048313255  0.038058379  0.034126993   0
1  −0.429431117  0.143283552  −0.048506827  0.038146062  0.034097765   0
1  −0.429555213  0.143389442  −0.048571352  0.038177313  0.034087348   0
1  −0.429590510  0.143422715  −0.048592860  0.038187954  0.034083801   0
1  −0.429601601  0.143433581  −0.048600029  0.038191526  0.034082610   0
1  −0.429605223  0.143437179  −0.048602419  0.038192720  0.034082212   0
1  −0.429606422  0.143438375  −0.048603215  0.038193118  0.034082080   0
1  −0.429606821  0.143438773  −0.048603481  0.038193251  0.034082036   0
1  −0.429606954  0.143438906  −0.048603569  0.038193295  0.034082021   0
1  −0.429606998  0.143438950  −0.048603599  0.038193310  0.034082016   0
1  −0.429607013  0.143438965  −0.048603609  0.038193315  0.034082014   0 <− (N = 14)
```

The following console output shows convergence of $m = 50$ with $q = -1 < 0$, I couldn't figure out the best way to find exactly the number of iterations N and I also played around with measuring absolute or relative error hence the $q = -1$. In doing that I saw the relationship explained by $A\hat{U} = F + \tau$ and how convergence in solving the system (increasing N) does not mean convergence to the true solution (increasing m) because of the LTE. The following shows clear convergence within 5000 iterations.

Console:

```
>> [Hist, x,U] = bvpqgs(50, 0, 2, −1, f, 1, 0, zeros(1, 50)', 5000);
>> Error = abs(Hist(5001, :) − Hist);
>> Error(5000, :)'

ans =

   1.0e−14 *

                    0
   0.011102230246252
   0.033306690738755
   0.055511151231258
   0.066613381477509
   ...
   0.082052420413703
   0.065876124000219
   0.049472145130514
   0.032981430087009
   0.016469031000055
                    0
```

Plugging in $m = 5$ and $q = 10$ results in a system which is not SDD, and also does not converge through GS iteration. With $m = 50$ using $q = 1000$ has the same result.

Console:

```
>> [Hist, x,U] = bvpqgs(5, 0, 2, 10, f, 1, 0, zeros(1, 5)', 5000);
U =
```

```
    1
  NaN
  NaN
  NaN
  NaN
  NaN
    0

>> [Hist, x,U] = bvpqgs(50, 0, 2, 1000, f, 1, 0, zeros(1, 50)', 5000);
U =

    1
  NaN
  NaN
  ...
  NaN
  NaN
    0
```

Problem P16: In calculus you probably learned Newton's method as a memorized formula: $x_{k+1} = x_k - f(x_k)/f'(x_k)$. Rewrite equations Newtons's method equations (8), (9) from the slides, in the one-dimensional case, to derive this memorized formula.

Proof. Recall equations (8) and (9) from class which demonstrate the 'solving a linear system' approach to Newton's method, where $J(x)$ is the evaluation of the jacobian and $f(x)$ is a vector valued function,

$$J(x_k)s = -f(x_k).$$

$$x_{k+1} = x_k + s.$$

In the one-dimensional case $f(x)$ is no longer a vector valued function and it follows that $J(x_k) = f'(x_k)$. By substitution we simply get $f'(x_k)s = -f(x_k)$, since all our elements are scalars we can divide across to get $s = -f(x_k)/f'(x_k)$. By substitution into equation (9) we get the desired result. □

Problem P17: This problem was assigned in Homework #2 of Numerical Linear Algebra last Fall.

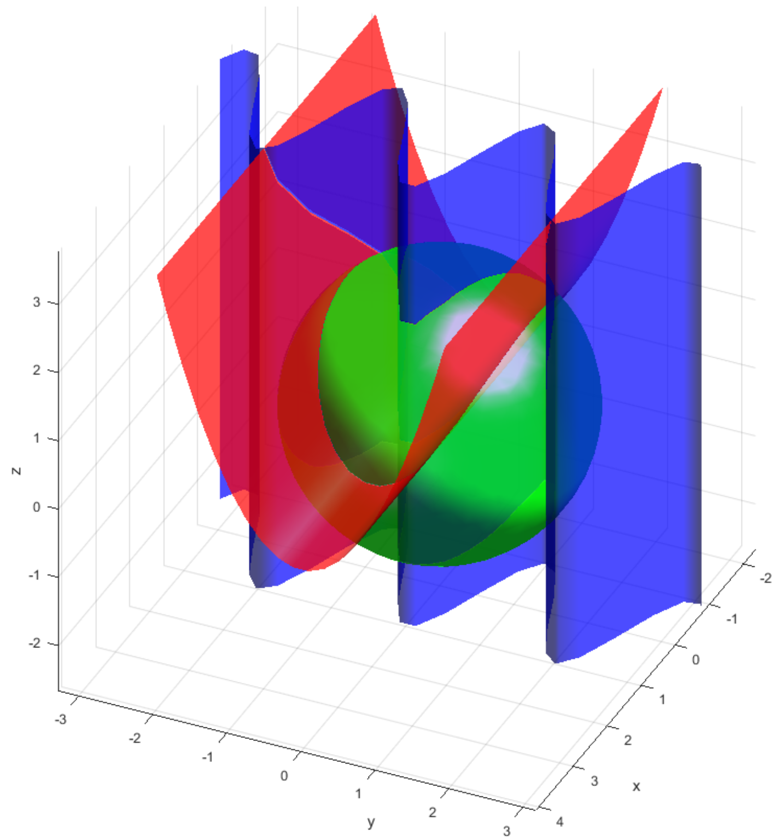
(a) Consider these 3 equations, chosen for visualizability:

$$x^2 + y^2 + z^2 = 4, \qquad x = \cos(\pi x), \qquad z = y^2$$

Sketch the surface of each equation in $\mathbb{R}^3$. Describe informally why there are two solutions of this system of three equations, that is, two points $(x, y, z) \in \mathbb{R}^3$ at which all three equations are satisfied. Explain why both solutions are inside the box $-1 \leq x \leq 1, -2 \leq y \leq 2, 0 \leq z \leq 2$.

Solution:
Here is a sketch of the three proposed surfaces

Figure 2: $x^2 + y^2 + z^2 = 4, \qquad x = \cos(\pi x), \qquad z = y^2$



Code:

```
% Used a package called ezimplot3 for plotting implicit functions
>> f1 = 'x^2 + y^2 + z^2 - 4'
>> f2 = 'cos(\pi*y) - x'
>> f3 = 'y^2 - z'

hold on
>> ezimplot3(f1)
>> ezimplot3(f2)
>> ezimplot3(f3)
```

Describing the solutions we can see that the ellipsoid and the parabolic cylinder intersect at a curve. When this curve is projected to the $x-y$ plane it only intersects the function $x = \cos(\pi y)$ in two places. The solutions lie within $-1 \leq x \leq 1$ because we are bounded by $x = cos(\pi y)$, $0 \leq z$ because we are bounded by $z = y^2$, and $-2 \leq y \leq 2$ and $z \leq 2$ because we are bounded by $x^2 + y^2 + z^2 = 4$.

(b) The slides describe Newton's method for nonlinear systems. Implement it in Matlab to solve the above linear system. Show your script and generate at least five iterations. Use $x_0 = (-1, 1, 1)$ as an initial iterate to find one solution, and also find the other solution using a different initial iterate. Note format long is appropriate here.

Solution:
The following is an implementation of Newton's method for solving this system. It abstracts the Jacobian and solves for the derivatives symbolically so one could plug in any 3 equations in $\mathbb{R}^3$ and attempt to solve the system.

Code:

```
function [Hist, xfinal] = NewtonsMethodP17(f1,f2,f3,xnot,n)
%This function takes the three surfaces from P17 f1,f2,and f3
%an initial iterate xnot, and the number of Newton Method
%iterations n. Symbolic Math Toolbox is required

%Initializing symbolic variables and history
syms x y z
Hist = [xnot];

%Initializing vector functions
NonLinearSystem = [f1;f2;f3];
Jacobian = jacobian(NonLinearSystem,[x y z]);

    for i = 1:n
        %Computing jacobian for current iterate
        J = double(subs(Jacobian,[x y z],xnot));
        %Computing vector function value for current iterate
        f = -1*double(subs(NonLinearSystem,[x y z],xnot));
        %Solving for s
        s = J\f;

        %Newton Step
        xx = xnot + s';
        Hist = [Hist; xx];
        %Computed vector becomes next iterate
        xnot = xx;
    end
%Setting Final
xfinal = xnot;
end
```

The following is the console output for running this routine with an initial iterate x_0 generating 6 Newton iterations and also finding the other solution by using

$x_0 = (-1, -1, 1)$.

Console:

```
>> f1
x^2 + y^2 + z^2 - 4

>> f2
cos(pi*y) - x

>> f3
y^2 - z

>> xnot
    -1        1        1

>> [Hist xfinal]=NewtonsMethodP5(f1,f2,f3,xnot,6)

Hist =

  -1.000000000000000   1.000000000000000   1.000000000000000
  -1.000000000000000   1.166666666666667   1.333333333333333
  -0.850071809562076   1.176823040188952   1.384809315996443
  -0.856411365815418   1.172732213485454   1.375284109683375
  -0.856360744297454   1.172720052382338   1.375272321111742
  -0.856360744261663   1.172720052019146   1.375272320407789
  -0.856360744261663   1.172720052019146   1.375272320407789


xfinal =

  -0.856360744261663   1.172720052019146   1.375272320407789


>> [Hist xfinal]=NewtonsMethodP5(f1,f2,f3,[-1,-1,1], 6)

Hist =

  -1.000000000000000  -1.000000000000000   1.000000000000000
  -1.000000000000000  -1.166666666666667   1.333333333333333
  -0.850071809562076  -1.176823040188952   1.384809315996443
  -0.856411365815418  -1.172732213485454   1.375284109683375
  -0.856360744297454  -1.172720052382338   1.375272321111742
  -0.856360744261663  -1.172720052019146   1.375272320407789
  -0.856360744261663  -1.172720052019146   1.375272320407789


xfinal =

  -0.856360744261663  -1.172720052019146   1.375272320407789
```

References

[1] Iterative Methods for Solving Ax = b—Gauss-Seidel Method — Mathematical Association of America. (n.d.). Retrieved February 19, 2023, from https://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-gauss-seidel-method

[2] Iterative Methods for Solving Ax = b—Jacobi's Method — Mathematical Association of America. (n.d.). Retrieved February 19, 2023, from https://www.maa.org/press/periodicals/loci/joma/iterative-methods-for-solving-iaxi-ibi-jacobis-method