Exercise P5: This question requires nothing but calculus as a prerequisite. It shows a major source of linear systems from applications.

a. Consider these three equation, chosen for visualizability:

$$x^2 + y^2 + z^2 = 4$$

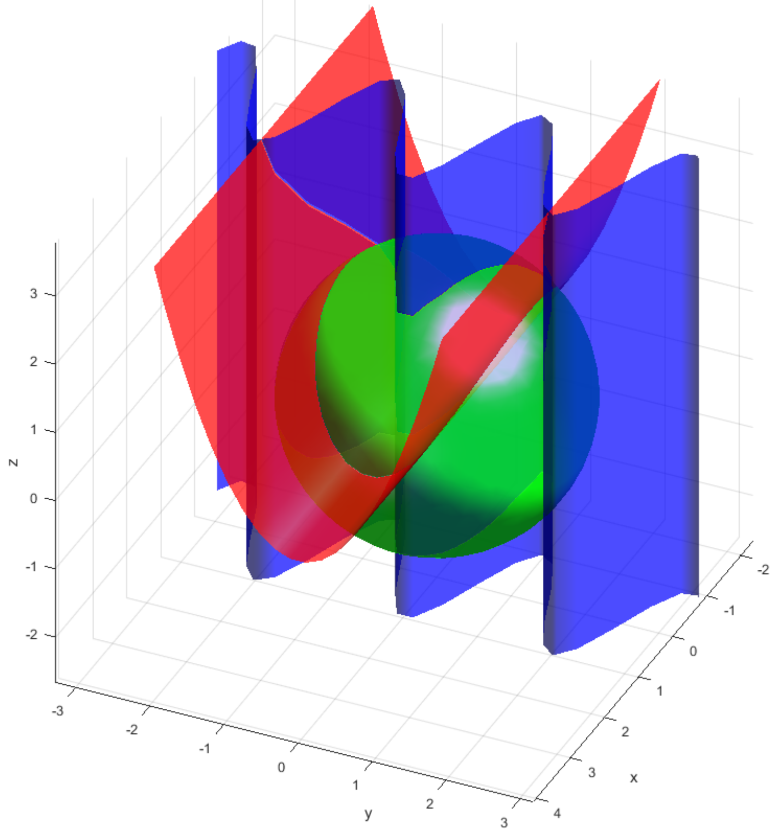
$$x = cos(\pi y)$$

$$z = y^2$$

Provide a sketch of each equation individually as a surface in $\mathbb{R}^3$. Consider where all three surfaces intersect, describe informally why there are two solutions. Explain why both solutions are inside the closed box $-1 \leq x \leq 1$, $-2 \leq y \leq 2$, and $0 \leq z \leq 2$.

Solution:
First let's consider a graph of each of the surfaces described above.

Figure 1: The three proposed surfaces.



Code:

```
% Used a package called ezimplot3 for plotting implicit functions
>> f1 = 'x^2 + y^2 + z^2 - 4'
f1 =
    'x^2 + y^2 + z^2 - 4'

>> f2 = 'cos(\pi*y) - x'
f2 =
    'cos(pi*y) - x'

>> f3 = 'y^2 - z'
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 that looks like a taco shell (Don't know how else to describe the 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. Newton's method for a system of nonlinear equations is an iterative, approximate, and sometimes very fast, method for solving systems like the one above. Let $x = (x_1, x_2, x_3) \in \mathbb{R}^3$. Suppose there are three scalar functions $f_i(x)$ forming a column vector function, and consider the system,

$$f(x) = 0,$$

Also let,

$$J_{ij} = \frac{\delta f_i}{\delta x_j}$$

Be the Jacobian matrix: $J \in \mathbb{R}^{3x3}$. The Jacobian generally depends on location and it generalizes the ordinary scalar derivative.
Newton's method itself is,

$$J(x_n)s = -f(x_n) \tag{1}$$
$$x_{n+1} = x_n + s \tag{2}$$

Where $s = (s_1, s_2, s_3)$ is the step and x_0 is the initial iterate. Using $x_0 = (-1, 1, 1)$ write out equation (1) for the $n = 0$ for the surfaces in part a, as a concrete linear system of three equation for the three unknown components of step $s = (s_1, s_2, s_3)$.

Solution:
First let's form the vector function $f(x)$ with the surfaces in part a,

$$f(x) = \begin{bmatrix} x_1^2 + x_2^2 + x_3^2 - 4 \\ cos(\pi x_2) - x_1 \\ x_2^2 - x_3 \end{bmatrix}$$

Now we can consider the associated Jacobian for this system,

$$J(x) = \begin{bmatrix} 2x_1 & 2x_2 & 2x_3 \\ -1 & -\pi sin(\pi x_2) & 0 \\ 0 & 2x_2 & -1 \end{bmatrix}$$

Solving each for $x_0 = (-1, 1, 1)$,

$$f(x_0) = \begin{bmatrix} -1^2 + 1 + 1 - 4 \\ cos(\pi) - (-1) \\ 1^2 - 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$$

$$J(x_0) = \begin{bmatrix} 2(-1) & 2(1) & 2(1) \\ -1 & -\pi sin(\pi) & 0 \\ 0 & 2(1) & -1 \end{bmatrix} = \begin{bmatrix} -2 & 2 & 2 \\ -1 & 0 & 0 \\ 0 & 2 & -1 \end{bmatrix}$$

Putting everything together to form the system to solve for s,

$$\begin{bmatrix} -2 & 2 & 2 \\ -1 & 0 & 0 \\ 0 & 2 & -1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$$

c. Implement Newton's method in Matlab to solve the part (a) nonlinear 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 that *format long* is appropriate here.

Solution:
Code:

```
function [Hist, xfinal] = NewtonsMethodP5(f1,f2,f3,xnot,n)
%This function takes the three surfaces from P5a f1,f2,and f3
%an initial iterate xnot, and the number of Newton Methed
%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
```

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
```

d. In calculus you likely learned Newton's method as a memorized formula $x_{n+1} = x_n - f(x_n)/f'(x_n)$. rewrite equations (1) and (2) for $\mathbb{R}^1$ to derive this formula.

Solution:
We can see that equation (1) for $\mathbb{R}^1$ is,

$$f'(x_n)s = -f(x_n).$$

We also get that equation (2) for $\mathbb{R}^1$ is,

$$x_{n+1} = x_n + s$$

Solving the first equation for s and substituting into the second equation we get,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Exercise P6: Its likely that you have learned a recursive method for computing determinants called ”expansion by minors”.

a. Compute the following determinant by hand to demonstrate that you can apply expansion in minors:

$$\left\|\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}\right\|$$

Solution:
From what I remember we called this method co-factor expansion,

$$\left\|\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}\right\| = \left\|\begin{bmatrix}5 & 6\\ 8 & 9\end{bmatrix}\right\| - 2\left\|\begin{bmatrix}4 & 6\\ 7 & 9\end{bmatrix}\right\| + 3\left\|\begin{bmatrix}4 & 5\\ 7 & 8\end{bmatrix}\right\|$$

$$\left\|\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}\right\| = 5*9 - 8*6 - 2(4*9 - 7*6) + 3(4*8 - 7*5) = 0$$

b. For any matrix $A \in \mathbb{C}^{mxm}$, count the exact number of multiplication operations needed to compute the determinant of A with expansion in minors.

Solution:
Let’s first recall the first step to compute the determinant with the expansion in minors method for matrix A,

$$det(A) = a_{1,1} * det(A_{m-1,m-1}) - a_{1,2} * det(A_{m-1,m-1}) + \ldots a_{1,m} * det(A_{m-1,m-1})$$

Now consider forming a recurrence $f(m)$ for counting the number of of multiplications and note that we also know the base case for the recurrence $f(2) = 2$.

$$f(m) = mf(m-1) + m$$

Expanding we get the following,

$$\begin{aligned}
f(m) &= mf(m-1) + m \\
&= m((m-1)f(m-2) + (m-1)) + m \\
&= m(m-1)f(m-2) + m(m-1) + m \\
&= m(m-1)((m-2)f(m-3) + (m-2)) + m(m-1) + m \\
&= m(m-1)(m-2)f(m-3) + m(m-1)(m-2) + m(m-1) + m \\
&\dots \\
&= m(m-1)(m-2)\dots(3)f(2) + \sum_{i=2}^{m-1} \frac{m!}{i!} \\
&= \frac{m!}{2!}f(2) + \sum_{i=2}^{m-1} \frac{m!}{i!} \\
&= m! + \sum_{i=2}^{m-1} \frac{m!}{i!}
\end{aligned}$$

c. We know that if $det(A) = 0$ then A is not invertible. However, rounding error makes an exact zero value extremely unlikely. On the other hand, the magnitude of $det(A)$ does not measure invertibility of A. For instance give a formula for $det(A)$ when A is diagonal and give a formula for A^{-1} if it exists. Show by example that $det(A)$ is often very large or very small even for well-conditioned diagonal matrices.

Solution:
Suppose diagonal matrix $A \in \mathbb{C}^{mxm}$. Applying the cofactor expansion(expansion by minor) we get the following,

$$det(A) = a_{1,1} * det(A_{m-1,m-1}) - a_{1,2} * det(A_{m-1,m-1}) + \dots a_{1,m} * det(A_{m-1,m-1})$$

Note that only values $A_{i,i}$ will result in non-zero the equation above simplifies greatly,

$$det(A) = a_{1,1} * det(A_{m-1,m-1})$$

Applying this same simplification to each inner determinant we get,

$$det(A) = \prod_{i=1}^{m} a_{i,i}. \tag{3}$$

Suppose this product is non-zero and therefore A is invertible. We know from the matrix multiplication algorithm thant in order to get $A^{-1}A = I$ we need that,

$$A^{-1} = \begin{bmatrix} \frac{1}{a_{1,1}} & & \\ & \ddots & \\ & & \frac{1}{a_{m,m}} \end{bmatrix}$$

What follows is that,

$$det(A) = \prod_{i=1}^{m} \frac{1}{a_{i,i}}.$$

This outlines precisely why we experience rounding error on even the most well conditioned matrices. Consider a large diagonal matrix with extremely large values in the diagonal. From equation 3 we know the resultant determinant will be orders of magnitude larger. Alternatively we know from 4 that the determinant of the inverse of the matrix will be orders of magnitude smaller. In both cases we experience the limits of IEEE representation.

Exercise P7: Write a Matlab program which draws the unit balls shown in 3.2 on page 18 of Trefethen ann Bau. That is, draw clean pictures of the unit ball of the one, two, infinity, an four norms.

Solution:
To create each plot I initialized several vectors in 3-space using the sphere command(You could do the same thing with meshgrid command). I then coded the norm as an anonymous function (the infinity norm was a little different), passing in each vector to compute the magnitude, and then normalizing. What we are left with are unit vectors with respect to the corresponding norm.

Figure 2: The one-norm unit ball

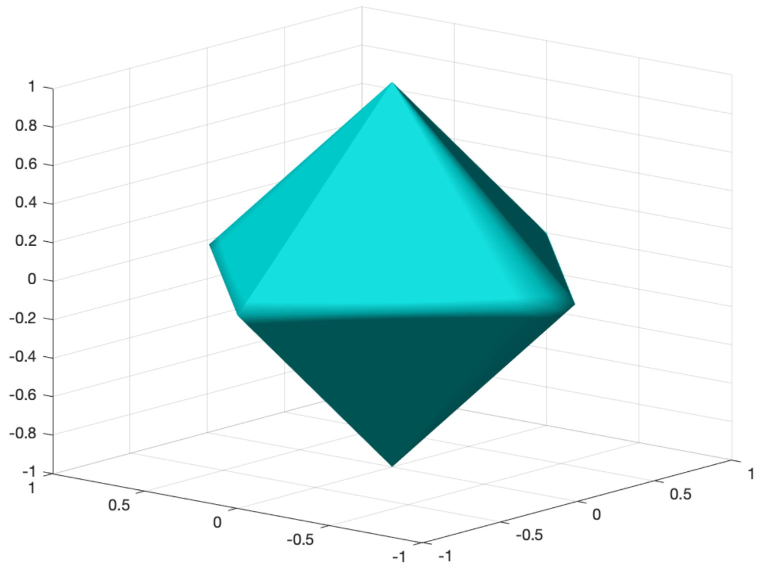


Figure 3: The infinity-norm unit ball

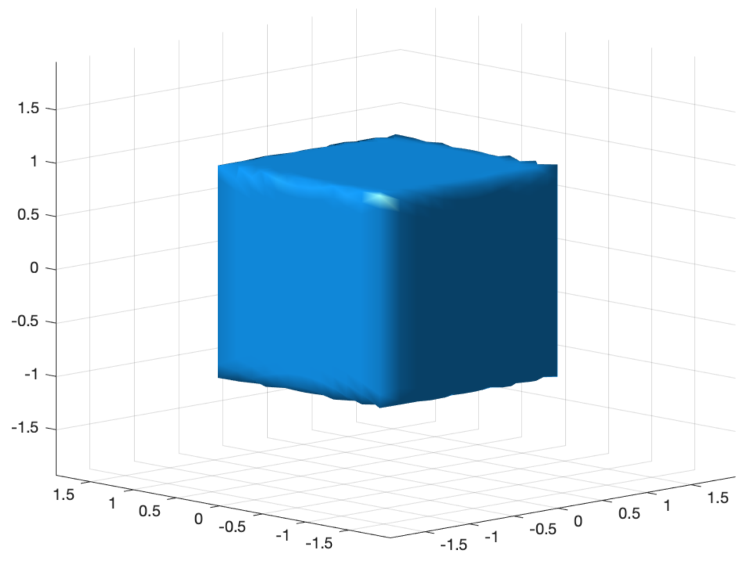


Figure 4: The two-norm unit ball

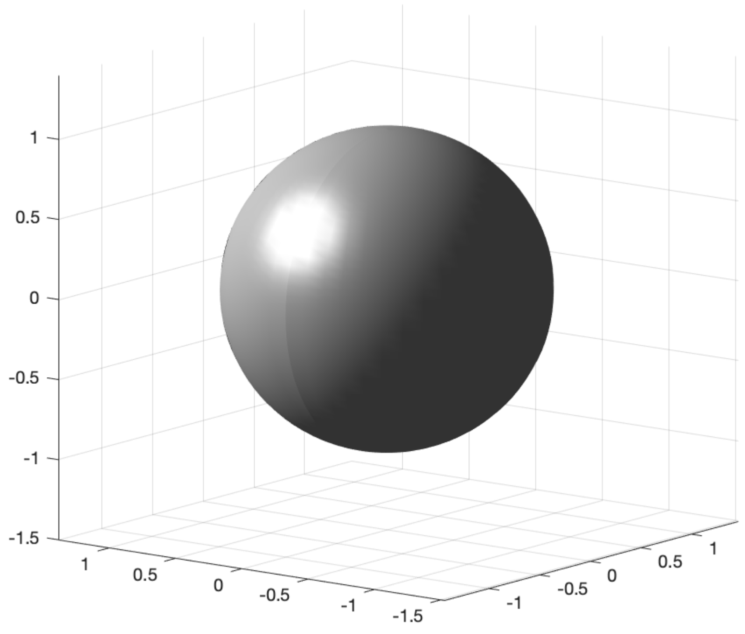
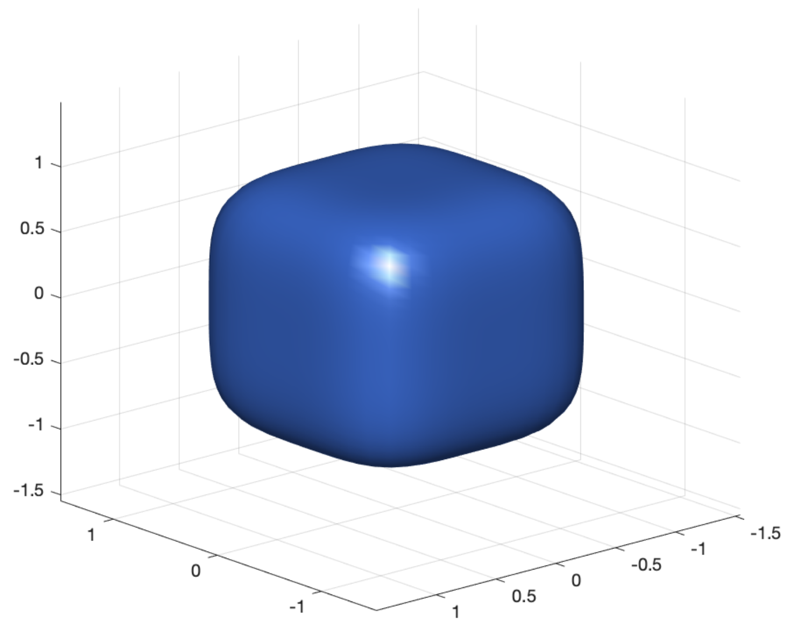


Figure 5: The four-norm unit ball



Code:

```
%Choose which plot to display
 ChoosePlot = 4;
%Choosing the Number of Vectors used to plot each norm
N = 50;
%Quick way of initilizing a mesh
 [X, Y, Z] = sphere(N);

 if ChoosePlot == 1
%%%%%%%%%%%%%%%%%%%%%%% Computing the one−Norm
     f1Norm = @(x, y, z) abs(x) + abs(y) + abs(z);

     Magnitude = f1Norm(X,Y,Z);
     xnorm =  X./Magnitude;
     ynorm =  Y./Magnitude;
     znorm =  Z./Magnitude;

     h = mesh(xnorm, ynorm, znorm);
     light('Position',[−1 0 .9],'Style','infinite')
     h.FaceLighting = 'gouraud';
```

```
    h.EdgeColor = 'none';
    h.FaceColor = 'cyan';
    h.SpecularStrength = 0.9;



elseif ChoosePlot == 2
%%%%%%%%%%%%%%%%%%%%%% Computing the Inf-Norm
    x1 = abs(reshape(X, [1 (N+1)^2]));
    x2 = abs(reshape(Y, [1 (N+1)^2]));
    x3 = abs(reshape(Z, [1 (N+1)^2]));

    Magnitude = max([x1; x2; x3]);
    Magnitude = reshape(Magnitude, [(N+1) (N+1)]);

    xnorm =  X./Magnitude;
    ynorm =  Y./Magnitude;
    znorm =  Z./Magnitude;

    h = mesh(xnorm, ynorm, znorm);
    light('Position',[-1 0 .9],'Style','infinite')
    h.FaceLighting = 'gouraud';
    h.EdgeColor = 'none';
    h.FaceColor = 'cyan';
    h.SpecularStrength = 0.9;
    xlim([-2 2])
    ylim([-2 2])
    zlim([-2 2])




elseif ChoosePlot == 3
%%%%%%%%%%%%%%%%%%%%%% Computing the two-Norm
    f2Norm = @(x, y, z) sqrt(abs(x).^2 + abs(y).^2 + abs(z).^2);

    Magnitude = f2Norm(X,Y,Z);
    xnorm =  X./Magnitude;
    ynorm =  Y./Magnitude;
    znorm =  Z./Magnitude;

    h = mesh(xnorm, ynorm, znorm);
    light('Position',[-1 1 1],'Style','infinite')
    h.FaceLighting = 'gouraud';
    h.EdgeColor = 'none';
    h.FaceColor = 'cyan';
    h.SpecularStrength = 0.9;
    xlim([-2 2])
    ylim([-2 2])
    zlim([-2 2])



else
%%%%%%%%%%%%%%%%%%%%%%%%%% Computing the 4-Norm
    f1Norm = @(x, y, z) sqrt(sqrt(abs(x).^4 + abs(y).^4 + abs(z).^4));

    Magnitude = f1Norm(X,Y,Z);
    xnorm =  X./Magnitude;
    ynorm =  Y./Magnitude;
    znorm =  Z./Magnitude;

    h = mesh(xnorm, ynorm, znorm);
    light('Position',[-1 1 1],'Style','infinite')
    h.FaceLighting = 'gouraud';
    h.EdgeColor = 'none';
    h.FaceColor = 'cyan';
    h.SpecularStrength = 0.9;

    xlim([-2 2])
    ylim([-2 2])
    zlim([-2 2])
end
```

Exercise 2.1: Show that if a matrix is both triangular and unitary then it must be diagonal.

Solution:
Suppose a matrix A is both triangular and unitary. By the definition of unitary we know that

$$A^*A = I.$$

Or equivalently,

$$A^* = A^{-1}.$$

Let's assume without loss of generality that A is upper triangular. Recall that since A is an upper triangular matrix then A^{-1} must also be upper triangular (exercise 1.3). Clearly if A was not diagonal, A^* would be lower triangular and A^{-1} would be upper triangular, therefore it follows that A must be diagonal.

Exercise 2.3: Let $A \in \mathbb{C}^{mxm}$ be hermitian. An eigenvector of A is a nonzero vector $x \in \mathbb{C}^m$ such that $Ax = \lambda x$ for some $\lambda \in \mathbb{C}$ the corresponding eigenvalue.

a. Prove that all eigenvalues of A are real.

Solution:
Suppose that $A \in \mathbb{C}^{mxm}$ is hermitian and consider that for some eigenvector and eigenvalue x, λ we have the following,

$$Ax = \lambda x.$$

Left multiplying both sides by the x^* we get the following,

$$\begin{aligned} x^*Ax &= x^*\lambda x \\ &= \lambda x^*x \\ &= \lambda ||x||^2 \end{aligned}$$

Now consider taking the adjoint of both sides.

$$\begin{aligned} (x^*Ax)^* &= (\lambda ||x||^2)^* \\ x^*A^*x &= \bar{\lambda}||x||^2 \end{aligned}$$

Since A is hermitian we know that $A^* = A$. Therefore by substitution we get,

$$x^*Ax = \bar{\lambda}||x||^2$$

Note that by substituting x^*Ax we can say that,

$$\lambda ||x||^2 = \bar{\lambda}||x||^2$$

We know that $||x||^2$ must always be non-zero since it is defined by a non-zero eigenvector. Dividing both sides by $||x||^2$ we are left with $\lambda = \bar{\lambda}$ which is only possible when λ is a real number.

b. Prove that if x and y are eigenvectors corresponding to distinct eigenvalues then x and y are orthogonal.

Solution:
Suppose A is hermitian with eigenvectors x and y and their corresponding eigenvalues λ_x and λ_y. By definition we know that,

$$Ax = \lambda_x x \tag{4}$$

$$Ay = \lambda_y y \tag{5}$$

Taking equation 4, left multiplying y^*, and simplifying similarly to the last problem we get,

$$\begin{aligned} y^*Ax &= y^*\lambda_x x, \\ &= \lambda_x y^* x, \\ &= \lambda_x < x, y > . \end{aligned}$$

Taking equation 5, left multiplying x^*, and simplifying similarly to the last problem we get,

$$\begin{aligned} x^*Ay &= x^*\lambda_y y, \\ &= \lambda_y y^* x, \\ &= \lambda_y < x, y > . \end{aligned}$$

Taking the adjoint of both sides we get,

$$\begin{aligned} (x^*Ay)^* &= \bar{\lambda}_y < x, y >, \\ (Ay)^* x &= \bar{\lambda}_y < x, y >, \\ y^*A^* x &= \bar{\lambda}_y < x, y > . \end{aligned}$$

Since A is hermitian, by substitution we get that,

$$y^*Ax = \bar{\lambda}_y < x, y > .$$

Substituting y^*Ax and with some algebra we can see that,

$$\begin{aligned} \lambda_x < x, y > &= \bar{\lambda}_y < x, y >, \\ \lambda_x < x, y > - \bar{\lambda}_y < x, y > &= 0, \\ < x, y > (\lambda_x - \bar{\lambda}_y) &= 0. \end{aligned}$$

Since we know that λ_x and $\bar{\lambda}_y$ are distinct real values it must be the case that $< x, y >= 0$ and therefore x and y are orthogonal.