MATLAB 第 1 次作业

第一题

(1)

$x'=\begin{bmatrix}4 & 5 & 1\end{bmatrix}$

(2)

$x+1=\begin{bmatrix}5\\6\\2\end{bmatrix}$

(3)

$x.\hat{}2=\begin{bmatrix}16\\25\\1\end{bmatrix}$

(4)

$x+y'=\begin{bmatrix}7\\5\\3\end{bmatrix}$

(5)

$x*y=\begin{bmatrix}12 & 0 & 8\\15 & 0 & 10\\3 & 0 & 2\end{bmatrix}$

(6)

y*x=14

(7)

$x.*y'=\begin{bmatrix}12\\0\\2\end{bmatrix}$

第二题

(1)

A(2,1)=2

(2)

A(5)=5

(3)

$$A(2:3,:)=\begin{bmatrix} 2 & 6 & 4 \\ 3 & 7 & 0 \end{bmatrix}$$

(4)

A(3,end)=0;

(5)

$$A(:)'=\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 1 & 4 & 0 & 7 \end{bmatrix}$$

(6)

$$A=\begin{bmatrix} 1 & 5 & 1 & 0 \\ 2 & 6 & 4 & 0 \\ 3 & 7 & 0 & 0 \\ 4 & 8 & 7 & 10 \end{bmatrix}$$

第三题

(1)

length(x)=3

(2)

size(A,2)=3

(3)

$$A\text{^}2=\begin{bmatrix} 1 & 0 & 0 \\ 6 & 1 & 4 \\ 0 & 0 & 1 \end{bmatrix}$$

(4)

$$A.\hat{}2=\begin{bmatrix}1 & 0 & 0\\0 & 1 & 4\\9 & 0 & 1\end{bmatrix}$$

(5)

$$A*x'=\begin{bmatrix}0\\5\\2\end{bmatrix}$$

(6)

$$A\backslash x'=\begin{bmatrix}0\\-3\\2\end{bmatrix}$$

(7)

$$A\hat{}{-1}*x'=\begin{bmatrix}0\\-3\\2\end{bmatrix}$$

(8)
区别：
(1)从适用范围上说：
①x = A\B 是对线性方程组 A*x = B 求解。矩阵 A 和 B 必须具有相同的行数。如果 A 未正确缩放或接近奇异值，MATLAB 将会显示警告信息，但还是会执行计算。
如果 A 是标量，那么 A\B 等于 A.\B。
如果 A 是 n×n 方阵，B 是 n 行矩阵，那么 x = A\B 是方程 A*x = B 的解（如果存在解的话）。
如果 A 是矩形 m×n 矩阵，且 m ~= n，B 是 m 行矩阵，那么 A\B 返回方程组 A*x = B 的最小二乘解。
②通过 A^-1*B 来求解线性方程组 A*x = B 时，只能处理 A 是方阵且 A 非奇异的情况，如果 A 不是方阵，MATLAB 会报错；如果 A 是方阵并且是奇异矩阵，MATLAB 会显示警告信息并且给出一个所有元素均为 Inf 的矩阵。

(2)从执行算法上说：
①通过 x = A\B 来求解线性方程组 A*x = B 时，MATLAB 执行的是高斯消元法，高斯消元法是求解线性方程组最高效的方法之一。
②如果通过 A^-1*B 来求解线性方程组 A*x = B，A^-1 与 inv(A)相同，inv 执行输入矩阵的 LU 分解（如果输入矩阵是 Hermitian 矩阵，则执行 LDL 分解），然后使用结果来形成线性方程组，其解为矩阵求逆 inv(A)。这种方法对于求解线性方程组来说，并不是一种高效的算法。
③按照 MATLAB 官方文档的说法，使用 A\B（而非 inv(A)*b）的速度要快两至三倍，而且残差减少了几个数量级。

第四题
(1)
A*B= $\begin{bmatrix} 1 & 1+i \\ 3-i & 3+2i \end{bmatrix}$

（2）
A.*B= $\begin{bmatrix} 1 & 0 \\ 2-2i & 1 \end{bmatrix}$

（3）
A\B= $\begin{bmatrix} 1 & 1+i \\ -1-i & -1-2i \end{bmatrix}$

(4)
B’= $\begin{bmatrix} 1 & 1+i \\ 1-i & 1 \end{bmatrix}$

（5）
Transpose(B)是对矩阵 B 进行转置，B’在转置的同时会对每一个元素的虚部求反,也即共轭转置

第五题
(1)
exp(A)= $\begin{bmatrix} 2.7183+0.0000i & -1.1312+2.4717i \\ 1.0000-0.0000i & 2.7183+0.0000i \end{bmatrix}$

(2)

expm(A)= $\begin{bmatrix} 2.7183+0.0000i & 2.7183+5.4366i \\ 0.0000+0.0000i & 2.7183+0.0000i \end{bmatrix}$

(3)

exp(A)为矩阵 A 中的每个元素 a_{ij} 返回指数运算 $e^{a_{ij}}$ 。对于复数元素 $a_{ij}=z=x+iy$，

它返回以下复指数 $e^{z}=e^{x}\left(\cos y+i\sin y\right)$

expm(A) 计算矩阵 A 的矩阵指数，如果 A 包含一组完整的特征向量 V 和对应特征值 D,则 [V,D] = eig(X) 且 expm(X) = V*diag(exp(diag(D)))/V，但是实际上 matlab 在计算矩阵指数的时候，并不是采用这种方法，根据查阅源代码，使用的是收缩乘方和 Padé 近似来进行计算，

(4)

inv(A)等效于 A^(-1)，即求 A 的逆矩阵。然而就如第四题中所述，只能处理 A 是方阵且 A 非奇异的情况，如果 A 不是方阵，MATLAB 会报错；如果 A 是方阵并且是奇异矩阵，MATLAB 会显示警告信息并且给出一个所有元素均为 Inf 的矩阵。如果 A 未正确缩放或接近奇异矩阵，inv 计算的数值将不准确。

pinv(A)返回矩阵 A 的 Moore-Penrose 伪逆。Moore-Penrose 伪逆是一种矩阵，可在不存在逆矩阵的情况下作为逆矩阵的部分替代。此矩阵常被用于求解没有唯一解或有许多解的线性方程组。对于任何矩阵 A 来说，伪逆 B 都存在，是唯一的，并且具有与 A'相同的维度。如果 A 是方阵且非奇异则 pinv(A)只是一种成本比较高的计算 inv(A) 的方式。但是，如果 A 不是方阵，或者是方阵且奇异，则 inv(A)不存在。在这些情况下，pinv(A)拥有 inv(A)的部分（但非全部）属性。而 pinv(A)的计算是基于 svd(A)（奇异值分解），该计算将小于 tol 的奇异值视为零。

第六题

A=diag([1,2,3]);
student=struct(‘name’,’张三’,’score’,[86,72,93]);
test={‘MATLAB’,student,A};

A 的第二行的引用：test{1,3}(2,:)

第七题

R=@(w,t) eye(3)+w*sin(t)+w^2*(1-cos(t));
定义匿名函数，生成一个函数句柄，此函数输入为标量t和3×3矩阵w（通过后续语句以及矩阵乘法的维数一致性可以得知w的大小）并返回一个3×3矩阵
tic;

启动秒表计时器，记录执行下面语句所用的内部时间

a=[1,0,0];

定义行向量a,根据后续语句可以看出，该行向量用于生成一个与之一一对应的3×3反对称矩阵w

w=[0,-a(3),a(2);a(3),0,-a(1);-a(2),a(1),0]/norm(a);

生成与向量a对应的反对称矩阵并且通过除以norm(a)，a向量的二范数（即向量的模）对该反对称矩阵进行归一化

theta=pi/2;

定义theta并赋值$\pi/2$

T=R(w,theta);

将定义好的theta和反对称矩阵w输入匿名函数，其中theta对应上面标量参数t，反对称矩阵w对应上面3×3输入矩阵，返回赋予变量名T，是一个3×3矩阵

p1=[1,2,3]';

定义3×1向量p1

p2=T*p1;

将返回的T与p1相乘，得到结果p2，是一个3×1向量

t=toc;

停止计时，将从tic开始的程序运行时间返回并赋值给变量t

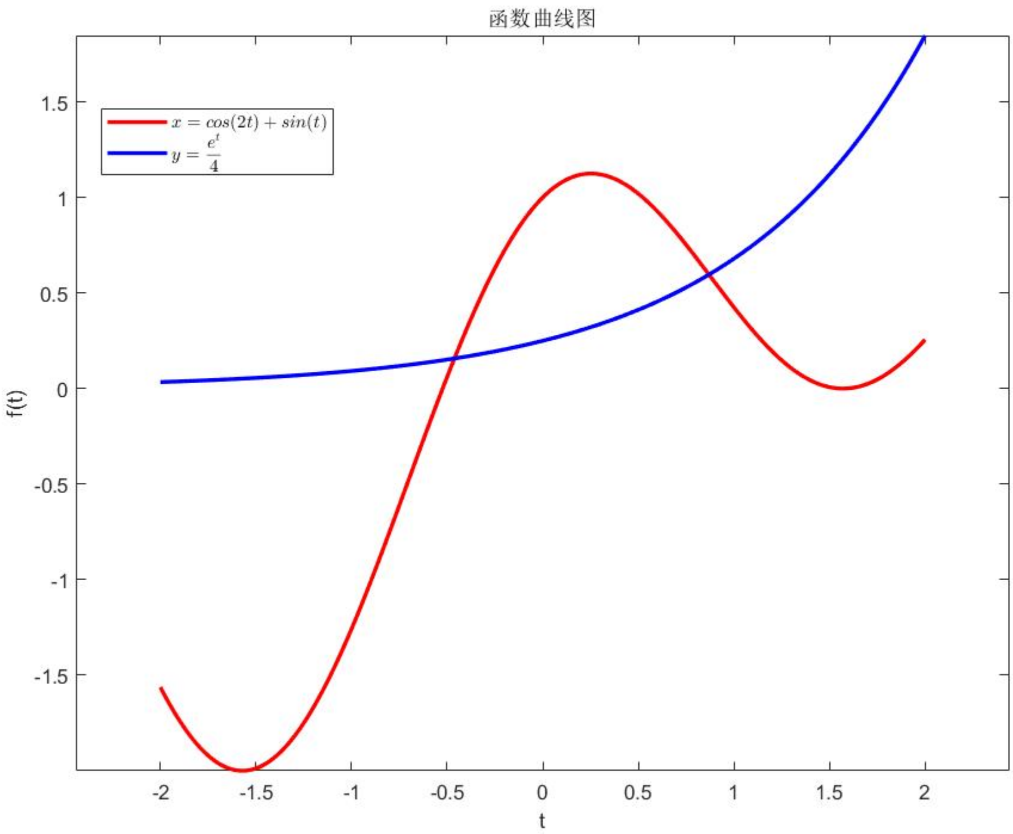
附加题 1

实际上，借用一点机器人学的背景知识，可以知道上述程序主要是在展示罗德里格斯公式，是计算三维空间中，一个向量绕旋转轴旋转给定角度以后得到的新向量的计算公式，在上面的程序中，theta 表示旋转的角度，而 a 则对应旋转轴，整个返回的 T 表示的就是一个旋转矩阵，因此在上述程序中 p2=Tp1，就表示对 p1 做了一次 T 旋转矩阵表示的旋转得到 p2，以 a=[1 0 0], theta= $\pi/2$ 来说的话，就是绕着[1 0 0]方向（即 x 轴正方向）逆时针旋转了 90 度，程序输出原本 p1=[1;2;3]，p2=[1,-3,2]也验证了此说法。

注：罗德里格斯公式形式如下：$R = I + sin(\theta)W + (1 - cos(\theta))W^2$， W 其实是给定旋转向量（轴）对应的反对称矩阵，推导过程中为了简化表达，使用了叉乘来进行向量的正交分解，而向量的叉乘与反对称矩阵有一一对应的关系，因此上面的程序中才会由a生成一个对应的反对称矩阵。

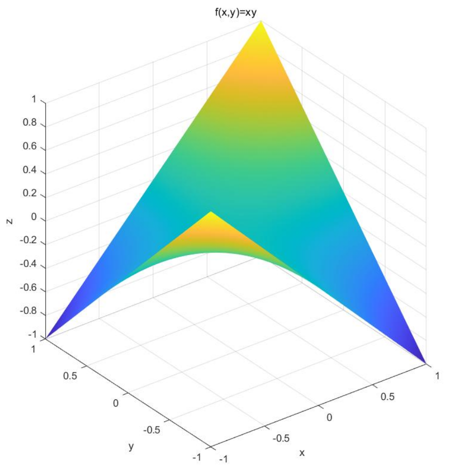
第八题

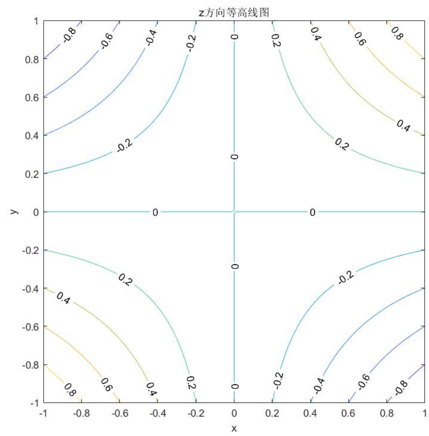
见下图所示：



第九题

见下图所示：





第十题

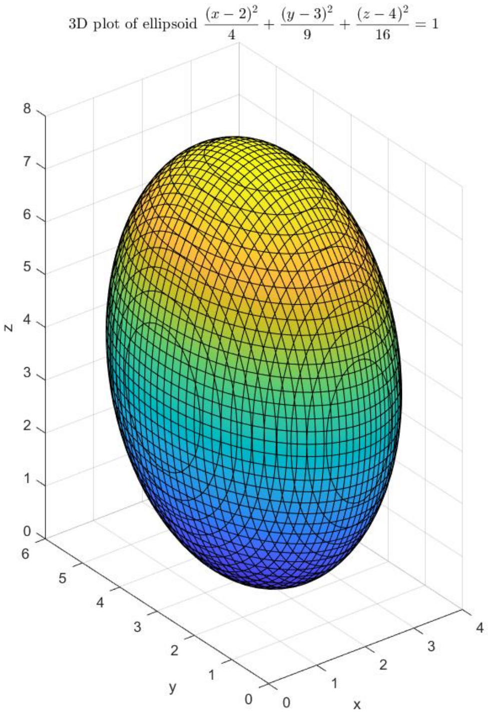
第一种方法，利用MATLAB内置的fimplicit函数绘图,解隐函数并绘图

%法一：隐函数法

f = @(x,y,z) (x-2).^2/4 + (y-3).^2/9 +(z-4).^2/16-1;

```
interval=[0 4 0 6 0 8]; %选取合适范围展示整个椭球
fimplicit3(f,interval)
% 坐标轴命名，取标题
xlabel('x')
ylabel('y')
zlabel('z')
title('3D plot of ellipsoid $$\frac{(x-2)^2}{4}+\frac{(y-3)^2}{9}+\frac{(z-4)^2}{16}=1$$','Interpreter','latex')
axis equal
```

绘图结果如下：

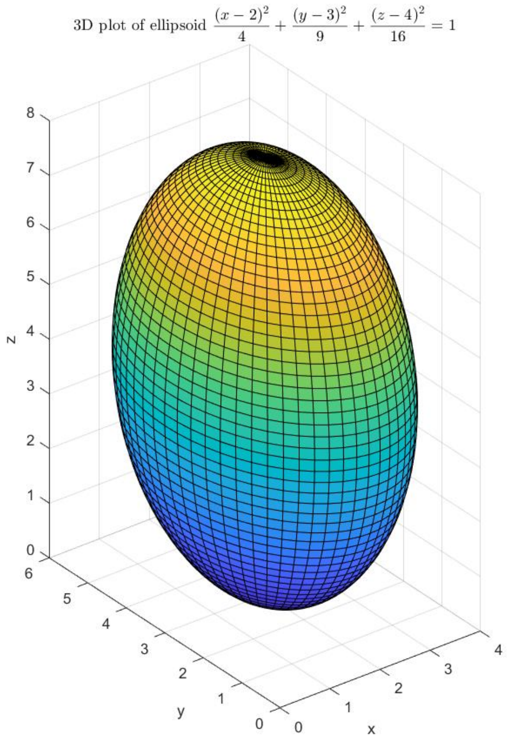


第二种方法，利用MATLAB内置的ellipsoid函数绘制，输入中心和三个半轴长即可得到椭球三维图

```
%法二：内置绘椭球图函数
[x, y, z] = ellipsoid(2,3,4,2,3,4,60);
surf(x, y, z)
% 坐标轴命名，取标题
xlabel('x')
ylabel('y')
```

```
zlabel('z')
title('3D plot of ellipsoid $$\frac{(x-2)^2}{4}+\frac{(y-3)^2}{9}+\frac{(z-4)^2}{16}=1$$','Interpreter','latex')
axis equal
```

绘图结果如下：

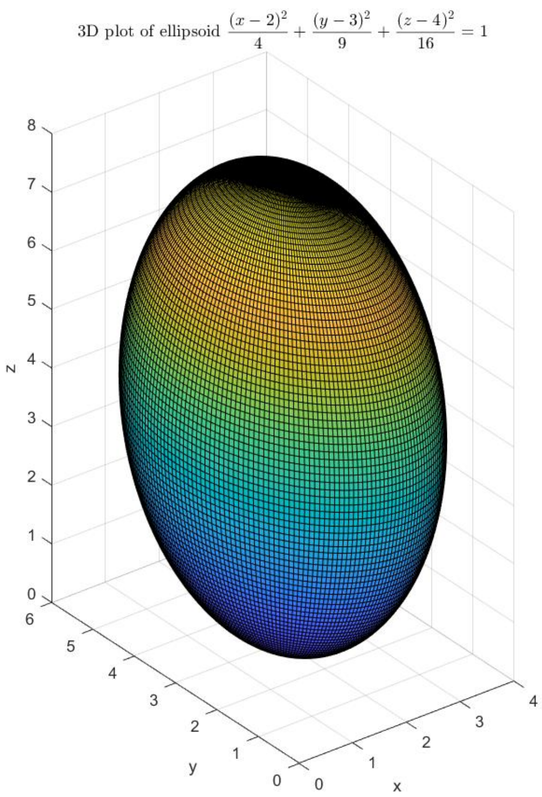


第三种方法，利用surf函数和椭球的参数方程自行编程进行绘制

```
%法三：参数方程法
phi=[0:pi/100:pi];
theta=[0:pi/100:2*pi];
[Phi,Theta]=meshgrid(phi,theta);
x=2*sin(Phi).*cos(Theta)+2;
y=3*sin(Phi).*sin(Theta)+3;
z=4*cos(Phi)+4;
surf(x,y,z);
%选取合适范围展示整个椭球
xlim([0 4])
ylim([0 6])
zlim([0 8])
% 坐标轴命名，取标题
```

```
xlabel('x')
ylabel('y')
zlabel('z')
title('3D plot of ellipsoid $$\frac{(x-2)^2}{4}+\frac{(y-3)^2}{9}+\frac{(z-4)^2}{16}=1$$','Interpreter','latex')
axis equal
```

绘图结果如下：



附加题2

本函数输入为变换次数n和生成子图行数m,将这一系列变换结果输出，排布在m行子图中，

算法思路：使用复数可以表示平面上的一个点（向量），而使用复数序列则可以直接确定平面上的一个图形，两个复数的差则是图形的一条边让$\{z_n\}$序列表示第n次变换的图形，初始 $\{z_0\}=\{z_{0,1} \quad z_{0,2} \quad z_{0,3} \quad z_{0,4}\}=\{1 \quad \frac{1+\sqrt{3}i}{2} \quad 0 \quad 1\}$

算法关键在于如何找到前一次序列到下一次序列的映射T，经过图像生成的原理描述，可以知道映射T可以归纳为简单重复的操作如下：

$T:\{\{z_n\}\mapsto\{z_{n+1}\}\}:$

1.从原始$\{z_n\}$序列，计算开始两个复数的差（确定一条边）$\Delta z = z_{n1} - z_{n2}$

2.找三等分点和外延点，并且顺序插入两点（复数）之间

$$\begin{cases} p_1 = z_{n,1} + \dfrac{\Delta z}{3} \\ p_2 = p_1 + \dfrac{\Delta z}{3} \times \dfrac{1-\sqrt{3}i}{2} \\ p_3 = z_{n,1} + \dfrac{2\Delta z}{3} \end{cases}$$ 并且原始序列 $$\begin{aligned}&\{z_n\} = \{z_{n,1} \quad z_{n,2} \quad \dots \quad z_{n,end}\} \\ &\to \{z_{n,1} \quad p_1 \quad p_2 \quad p_3 \quad z_{n,2} \quad \dots \quad z_{n,end}\}\end{aligned}$$

完成一条边的扩展，其中使用复数的好处在于旋转的外扩可以借由乘上对应复数实现

3.对原始序列中的下一个相邻点$z_{n,2}, z_{n,3}$重复以上操作，完成第二条边的扩展

4.不断重复以上操作，直至遍历原始序列中的所有相邻点组

Matlab函数如下：plt_Tri.m

```
function plt_Tri(n,m)
z=[1 (1+sqrt(3)*i)/2 0 1]; % 初始序列,表示原始图形的四个顶点（初始点重复计算）
iter=n;
for j=1:iter                    % 上述映射变换的迭代次数
    z_0=z;                  % 给定初始序列（图形），后面根据循环更新插入新点
% 计算（插入三等分点操作）次数，也即边的条数例如第一次有4点三条边，第二次有13个点12条边
    div=length(z)-1;
    %需要对每一条边这样处理，用循环
    for k=1:div
        dz=(z_0(k+1)-z_0(k))/3; %计算原图形相邻两点差（即要插的边）
        %开始生成第一条边对应的图形，每一个图形都是4点一个单位
        z(4*k-3)=z_0(k);            %起点
        z(4*k-2)=z_0(k)+dz;         %第一个三等分点
        z(4*k-1)=z(4*k-2)+dz*(1/2-sqrt(3)*1i/2); %第二个点，需要旋转出去的
        z(4*k)=z_0(k)+2*dz;     %第二个三等分点
    end
    z(4*div+1)=z_0(div+1);              %终点
    subplot(m,ceil(n/m),j)              %打印子图
    fill(real(z),imag(z),'k')
    title(['stage',num2str(j)])         %取标题
    axis equal
```

```
end
end
```

在主程序中，调用以上函数plt_Tri(4,2)，产出四次变换生成的图像并打印成2×2的子图如下图所示：