Question 1.

(25 points)

There are three types of coins which have different probabilities of landing heads when tossed:

– Type A coins are fair, with probability 0.5 of heads

– Type B coins are bent and have probability 0.6 of heads

– Type C coins are bent and have probability 0.9 of heads

Suppose I have a drawer containing 5 coins: 2 of type A, 2 of type B, and 1 of type C. I reach into the drawer and pick a coin at random. Without showing you the coin I flip it once and get heads (i.e. event D). What is the probability it is

- type A (i.e. $P(H=A|D)$)?
- Type B (i.e. $P(H=B|D)$)?
- Type C (i.e. $P(H=C|D)$)?

Fill out table below with your answers.

`Step1:prior knowleges`

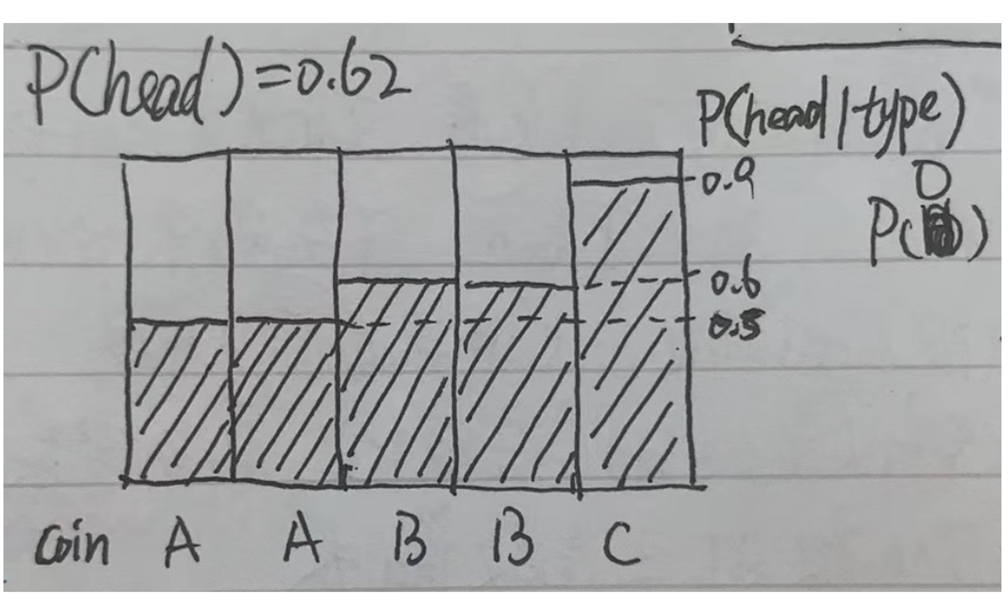
The sample space of B is {a,a,b,b,c}, so the probability to pick up an A Type coin $P(H=A)=2/5$, so as $P(H=B)=2/5$, $P(H=C)=0.2$

hypothesis	prior	likelihood	posterior
H	P(H)	P(D|H)	P(H|D)
A	0.4		
B	0.4		
C	0.2		

`Step2: fill out the specific probabilities when different coin type is selected`

hypothesis	prior	likelihood	posterior
H	P(H)	P(D|H)	P(H|D)
A	0.4	0.5	
B	0.4	0.6	
C	0.2	0.9	

`Step3: get P(D)`



$P(D)=P(head\ of\ observe)$

$= area\ of\ the\ shadow\ of\ the\ probability\ space$

$= \frac{0.5\times2+0.6\times2+0.9}{5}$

`= 0.62`

`Step4: now calculate the shrinked probability of observed a Head given certain coin type.`

$P(H|D)=P(H)\frac{P(D|H)}{P(D)}$

hypothesis	prior	likelihood	posterior
H	P(H)	P(D|H)	P(H|D)
A	0.4	0.5	0.3326
B	0.4	0.6	0.3871
C	0.2	0.9	0.2903

Question 2. (25 points - Written)

Consider the following dataset of four rows and three features (Malicious, Viagara, Meet) with class labels (ham and spam). Suppose we see a message having these features M5 = (Malicious = 'yes' ,Viagara = 'no' ,Meet = 'yes'), What is the probability that it is a spam or ham? Using Naive Bayes algorithm.

S.No	Malicious	Viagara	Meet	class
M1	yes	yes	yes	spam
M2	no	no	yes	ham
M3	yes	no	yes	spam
M4	no	yes	no	ham
M5	yes	no	yes	?%

`first, calculate the prior knowledge of labels spam & ham`

$given\ P(class=spam)=0.5, P(class=ham)=0.5$

$and\ P(Malicious=yes)=0.5, P(Viagara=yes)=0.5, P(Meet=yes)=0.75$

$P(spam|X=M5)$

$=P(spam|yes,no,yes)$

$=\frac{P(spam)P(yes,no,yes|spam)}{P(yes,no,yes)}$

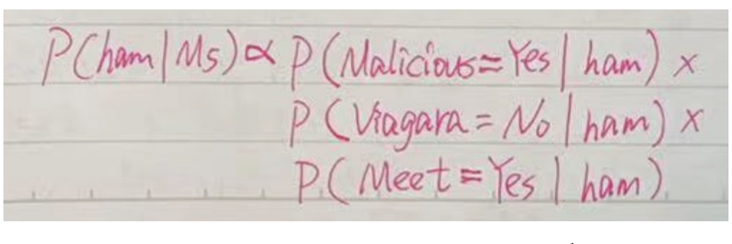
$=\frac{P(spam)\times P(Malicious=yes|spam)\times P(Viagara=no|spam)\times P(Meet=yes|spam)+1}{P(Malicious=yes)\times P(Viagara=no)\times P(Meet=yes)+2}$

$P(ham|X=M5)$

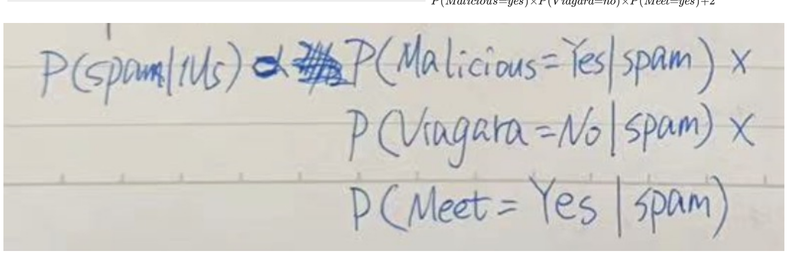
$=P(ham|yes,no,yes)$

$=\frac{P(ham)P(yes,no,yes|ham)}{P(yes,no,yes)}$

$=\frac{P(ham)\times P(Malicious=yes|ham)\times P(Viagara=no|ham)\times P(Meet=yes|ham)+1}{P(Malicious=yes)\times P(Viagara=no)\times P(Meet=yes)+2}$



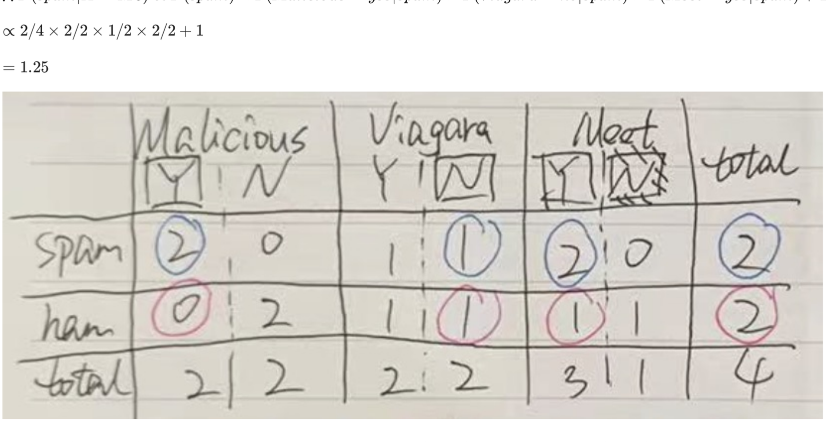
`since the equations have the same part of factor:` $\frac{1}{P(Malicious=yes)\times P(Viagara=no)\times P(Meet=yes)+2}$



$\therefore P(spam|X=M5)\propto P(spam)\times P(Malicious=yes|spam)\times P(Viagara=no|spam)\times P(Meet=yes|spam)+1$

$\propto 2/4\times2/2\times1/2\times2/2+1$

$=1.25$



$\therefore P(ham|X=M5)\propto P(spam)\times P(Malicious=yes|ham)\times P(Viagara=no|ham)\times P(Meet=yes|ham)+1$

$\propto 2/4\times0/2\times1/2\times1/2+1$

$=1$

$\therefore$ M5 has a $\frac{1.25}{1.25+1}=55.6\%$ probability to be spam, and has a 44.4% probability to be ham.

Question 3.(30 points - Written)

Use the below gradient descent algorithm to find the x value that minimizes the function $f(x)=6x_1^2-3x_1x_2+2x_2^2$.

Choose the starting point as $(x_1^{(0)},x_2^{(0)})=(2,3)$ and use the η_k as 0.1 ans ε as 0.0001

Consider the following gradient descent algorithm (which is a variant of the directional search algorithm) that aims to find the x value that minimizes the function f(x).

- Guess $x^{(0)}$, set $k\leftarrow0$
- while $\|\nabla f(x^{(k)})\|\geq\varepsilon$ do

$x^{(k+1)}=x^{(k)}-\eta_k\nabla f(x^{(k)})$

$k\leftarrow k+1$

- end while
- return $x^{(k)}$

Perform only 3 iterations of this algorithm and report the values of (x_1,x_2) after performing gradient descent each time

---

$\because f(x)=6x_1^2-3x_1x_2+2x_2^2$

$\therefore \frac{\partial f(x)}{\partial x_1}=12x_1-3x_2+2x_2^2\ ,\ \frac{\partial f(x)}{\partial x_2}=6x_1^2-3x_1+4x_2$

$\therefore \nabla f(x^{(k)})=[\frac{\partial f(x)}{\partial x_1},\frac{\partial f(x)}{\partial x_2}]$

$\because \nabla f(x^{(k)})>\epsilon=0.0001\ and\ \eta_k=0.1$

$then,\ x^{k+1}=x^k-0.1\times[\frac{\partial f(x)}{\partial x_1},\frac{\partial f(x)}{\partial x_2}]$

`Perform 3 iterations of gradient descent`

k	$x_1^{(k)}$	$x_2^{(k)}$	$x^{(k)}$	$\nabla f(x^{(k)})$		$$	\nabla f(x^{(k)})	$$	$x_1^{(k+1)}$	$x_2^{(k+1)}$
0	2	3	24	$[33,30]$	$\sqrt{33^2+30^2}=40.6>\epsilon$	2-3.3	3-3			
1	-1.3	0	10	$[-15,14]$	$\sqrt{15^2+14^2}=20.5>\epsilon$	-1.3-(-1.5)	0-1.4			
2	0.2	-1.4	5	$[10,-5]$	$\sqrt{10^2+5^2}=11.2>\epsilon$	0.2-1	-1.4-(-0.5)			
3	-0.8	-0.9	3	$[-5,2]$	$\sqrt{5^2+2^2}=5.4>\epsilon$	-0.8-(-0.5)	-0.9-0.2			
4	-0.3	-1.1								

In [1]:

```python
# for self-check
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import math
def f(x1, x2):
    return 6*x1**2 - 3*x1*x2 + 2*x2**2
def fx1(x1, x2):
    return 12*x1-3*x2+2*x2**2
def fx2(x1, x2):
    return 6*x1**2-3*x1+4*x2
def mold(px1, px2):
    return math.sqrt((px1 ** 2 + px2 ** 2))
x1,x2=-0.8, -0.9
print("%d | [%d,%d]" % (f(x1,x2), fx1(x1,x2), fx2(x1,x2)))
T1=np.array([2,-1.3,0.2,-0.8,-0.3])
T2=np.array([3,0,-1.4,-0.9,-1.1])
T3=f(T1,T2)
```

```
3 | [-5,2]
```

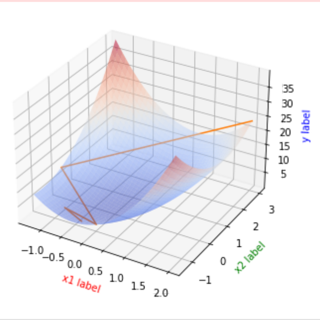
`Report`

k	$x_1^{(k)}$	$x_2^{(k)}$
0	2	3
1	-1.3	0
2	0.2	-1.4
3	-0.8	-0.9

In [2]:

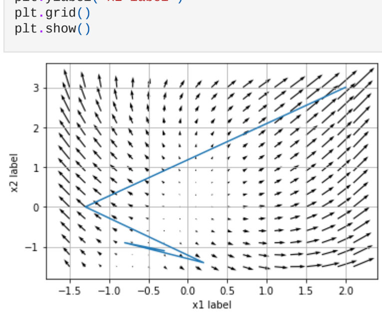
```python
# self-check visualization
fig1 = plt.figure()
ax = Axes3D(fig1)
X1, X2 = np.mgrid[-1.3:2:40j, -1.4:3:40j]
Y = f(X1, X2)
ax.plot_surface(X1, X2, Y, rstride=1, cstride=1, cmap=plt.cm.coolwarm, alpha=0.5)
ax.plot(T1, T2, T3)
ax.set_xlabel('x1 label', color='r')
ax.set_ylabel('x2 label', color='g')
ax.set_zlabel('y label', color='b')
plt.show()
```

```
/var/folders/lm/kt3nd3_d5xbdlqg_f3xz91y40000gn/T/ipykernel_84237/2950451877.py:3: MatplotlibDeprecationWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this warning. The default value of auto_add_to_figure will change to False in mpl3.5 and True values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(fig1)
```



In [3]:

```python
# self-check visualization
import numpy as np
import matplotlib.pyplot as plt
def numerical_gradient(f,P):
    grad = np.zeros_like(P)
    for i in range(P[0].size):
        grad[0][i] = fx1(P[0][i], P[1][i])
        grad[1][i] = fx2(P[0][i], P[1][i])
    return grad
lx1,rx1,r1=-1.5,2,0.2
lx2,rx2,r2=-1.5,3,0.3
x = np.arange(lx1, rx1+r1, r1)
y = np.arange(lx2, rx2+r2, r2)
X, Y = np.meshgrid(x, y)
X = X.flatten()
Y = Y.flatten()
grad = numerical_gradient(f, np.array([X, Y]))
plt.quiver(X, Y, grad[0], grad[1])
T1=np.array([2,-1.3,0.2,-0.8,-0.3])
T2=np.array([3,0,-1.4,-0.9,-1.1])
T3=f(T1,T2)
plt.plot(T1,T2,T3)
plt.xlim([lx1*1.2, rx1*1.2])
plt.ylim([lx2*1.2, rx2*1.2])
plt.xlabel('x1 label')
plt.ylabel('x2 label')
plt.grid()
plt.show()
```



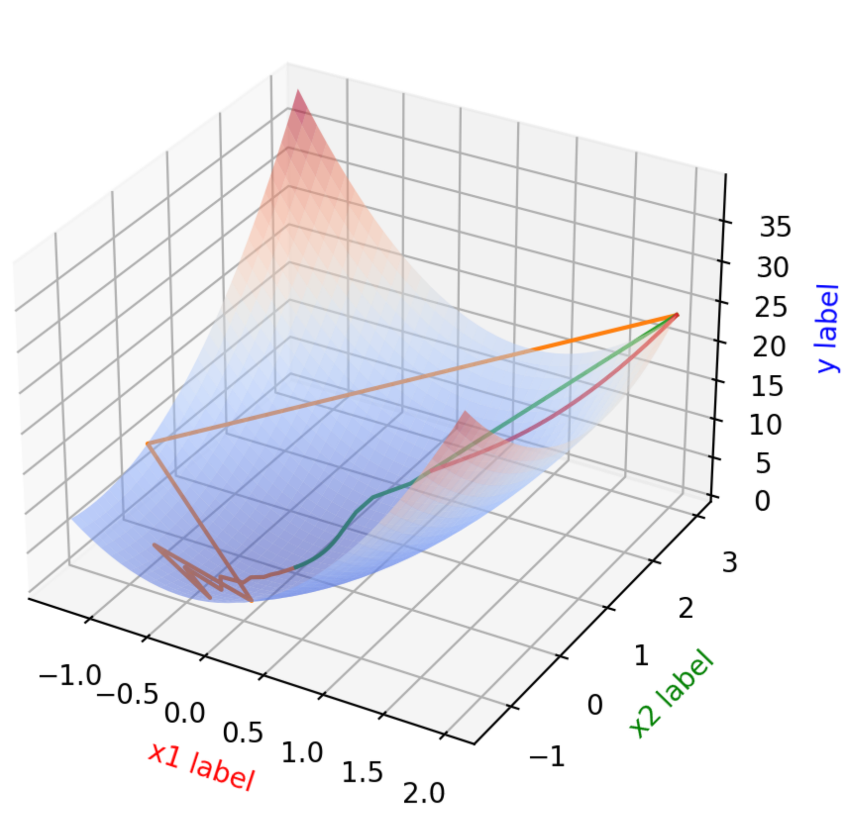
Extra work for a better learning rate visualization

In [4]:

```python
# visualization for a better learning rate
def gradient_decent(x1, x2, alpha=0.05, epsilon=0.0001, max_iter=10000):
    x1, x2 = 2, 3
    px1, px2 = fx1(x1, x2), fx2(x1, x2)
    md = mold(px1, px2)
    T1, T2, T3 = [], [], []
    cnt = 0
    convergence_cnt = 0
    while True:
        if md < epsilon:
            print("break when gradient vanished: %.6f<%.6f after %d times. " % (md, epsilon, cnt))
            break
        if cnt > max_iter:
            print("break when iterate %d times with gradient %.6f. " % (cnt, md))
            break
        if convergence_cnt > 5:
            print("gradient convergence detected. ")
            break
        T1.append(x1)
        T2.append(x2)
        T3.append(f(x1, x2))
        px1 = fx1(x1, x2)
        px2 = fx2(x1, x2)
        tmp = mold(px1, px2)
        if tmp > md:
            convergence_cnt += 1
        md = tmp
        x1 = x1 - alpha * px1
        x2 = x2 - alpha * px2
        cnt += 1
    return T1, T2, T3

# print(T1,"\n",  T2,"\n",  T3)
fig1 = plt.figure(dpi=200)
ax = Axes3D(fig1)
X1, X2 = np.mgrid[-1.3:2:40j, -1.4:3:40j]
Y = f(X1, X2)
ax.plot_surface(X1, X2, Y, rstride=1, cstride=1, cmap=plt.cm.coolwarm, alpha=0.5)
T1,T2,T3=gradient_decent(2,3,alpha=0.1)
ax.plot(T1, T2, T3)
T1,T2,T3=gradient_decent(2,3,alpha=0.05)
ax.plot(T1, T2, T3)
T1,T2,T3=gradient_decent(2,3,alpha=0.00001)
ax.plot(T1, T2, T3)
ax.set_xlabel('x1 label', color='r')
ax.set_ylabel('x2 label', color='g')
ax.set_zlabel('y label', color='b')
plt.show()
```

```
/var/folders/lm/kt3nd3_d5xbdlqg_f3xz91y40000gn/T/ipykernel_84237/2955345944.py:35: MatplotlibDeprecationWarning: Axes3D(fig) adding itself to the figure is deprecated since 3.4. Pass the keyword argument auto_add_to_figure=False and use fig.add_axes(ax) to suppress this warning. The default value of auto_add_to_figure will change to False in mpl3.5 and True values will no longer work in 3.6.  This is consistent with other Axes classes.
  ax = Axes3D(fig1)
break when gradient vanished: 0.000090<0.000100 after 38 times.
break when gradient vanished: 0.000091<0.000100 after 66 times.
break when iterate 10001 times with gradient 9.033484.
```



In [5]:

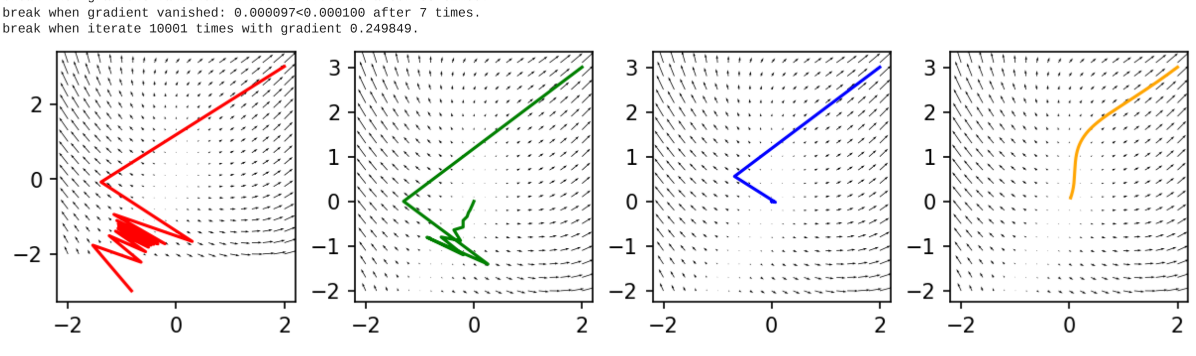
```python
lx1,rx1,r1=-2,2,0.2
lx2,rx2,r2=-2,3,0.3
x = np.arange(lx1, rx1+r1, r1)
y = np.arange(lx2, rx2+r2, r2)
X, Y = np.meshgrid(x, y)
X = X.flatten()
Y = Y.flatten()
grad = numerical_gradient(f, np.array([X, Y]))

fig = plt.figure(figsize=(8, 2), dpi=150, constrained_layout=True)
gs = plt.GridSpec(1, 4, figure=fig, left=0.1, right=0.9, bottom=0.1, top=0.9, wspace=0.05, hspace=0.05)
ax1 = fig.add_subplot(gs[0,0])
ax2 = fig.add_subplot(gs[0,1])
ax3 = fig.add_subplot(gs[0,2])
ax4 = fig.add_subplot(gs[0,3])

def plot_gradient_path(ax, t1,t2, clr):
    ax.quiver(X, Y, grad[0], grad[1])
    ax.plot(T1,T2, c=clr)

T1,T2,T3=gradient_decent(2,3,alpha=0.1027)
plot_gradient_path(ax1, T1,T2, clr='r')
T1,T2,T3=gradient_decent(2,3,alpha=0.1)
plot_gradient_path(ax2, T1,T2, clr='g')
T1,T2,T3=gradient_decent(2,3,alpha=0.081377)
plot_gradient_path(ax3, T1,T2, clr='b')
T1,T2,T3=gradient_decent(2,3,alpha=0.0001)
plot_gradient_path(ax4, T1,T2, clr='orange')
plt.show()
```

```
gradient convergence detected.
break when gradient vanished: 0.000090<0.000100 after 38 times.
break when gradient vanished: 0.000097<0.000100 after 7 times.
break when iterate 10001 times with gradient 0.249849.
```



Question 4. (20 points - Written)

Find the eigen values and eigen vectors for the following matrix.

$$\begin{bmatrix}2&5\\-1&2\end{bmatrix}\tag{1}$$

$A\vec{x}=\lambda\vec{x}$

$\Rightarrow A\vec{x}-\lambda I\vec{x}=0$

$\Rightarrow (A-\lambda I)\vec{x}=0$

$\Rightarrow det(A-\lambda I)=\begin{vmatrix}2-\lambda&5\\-1&2-\lambda\end{vmatrix}=(2-\lambda)^2-(-1\times5)=0$

$\Rightarrow \lambda^2-4\lambda+9=0$

for $ax^2+bx+c=0,\ \begin{cases}x_1=m+ni\\x_2=m-ni\end{cases}$

$then\ \begin{cases}x_1+x_2=-\dfrac{b}{a}=4=2m\\x_1\times x_2=\dfrac{c}{a}=9=m^2+n^2\end{cases}$

$\therefore m=2,\ n=\sqrt{5}$

$$\therefore \lambda_1=2+\sqrt{5}i,\ \lambda_2=2-\sqrt{5}i\tag{2}$$

In [6]:

```python
# for self-check
import numpy as np
C = np.mat('2,5;-1,2')
c1,c2 = np.linalg.eig(C)
print("eigen values: %s"%(c1))
```

```
eigen values: [2.+2.23606798j 2.-2.23606798j]
```

---

remember $A\vec{x}=\lambda\vec{x}$

$\Rightarrow \begin{bmatrix}2&5\\-1&2\end{bmatrix}\begin{bmatrix}x1\\x2\end{bmatrix}=\lambda\begin{bmatrix}1&0\\0&1\end{bmatrix}\begin{bmatrix}x1\\x2\end{bmatrix}$

$\Rightarrow \begin{bmatrix}2-\lambda&5\\-1&2-\lambda\end{bmatrix}\begin{bmatrix}x1\\x2\end{bmatrix}=0$

$\Rightarrow \begin{cases}(2-\lambda)x_1+5x_2=0\\-x_1+(2-\lambda)x_2=0\end{cases}$

$\Rightarrow \frac{x_1}{x_2}=(2-\lambda)$

---

for $\lambda_1=2+\sqrt{5}i$

$\frac{x_1}{x_2}=-\sqrt{5}i$

$$\xi_1=\begin{bmatrix}x1\\x2\end{bmatrix}=\begin{bmatrix}-\sqrt{5}i\\1\end{bmatrix}\tag{3}$$

for $\lambda_2=2-\sqrt{5}i$

$\frac{x_1}{x_2}=\sqrt{5}i$

$$\xi_2=\begin{bmatrix}x1\\x2\end{bmatrix}=\begin{bmatrix}\sqrt{5}i\\1\end{bmatrix}\tag{4}$$

In [7]:

```python
#self-check
from math import sqrt
a = complex(0,sqrt(5))
A = np.mat('2,5;-1,2')

x1 = np.array([-a,1])
λ1 = 2 + a
print("for λ1=%s"%λ1)
print("A dot x1: ",np.dot(A,x1))
print("λ1 *  x1: " , np.dot(λ1,x1))
print()

x2 = np.array([a,1])
λ2 = 2 - a
print("for λ2=%s"%λ2)
print("A dot x2: ",np.dot(A,x2))
print("λ2 *  x2: " , np.dot(λ2,x2))
```

```
for λ1=(2+2.23606797749979j)
A dot x1:  [[5.-4.47213595j 2.+2.23606798j]]
λ1 *  x1:  [5.-4.47213595j 2.+2.23606798j]

for λ2=(2-2.23606797749979j)
A dot x2:  [[5.+4.47213595j 2.-2.23606798j]]
λ2 *  x2:  [5.+4.47213595j 2.-2.23606798j]
```

In [ ]: