SageMath-Lecture6-June08_2020

June 8, 2020

1 Lecture 6: Linear Algebra with Sage

By Ajit Kumar
ICT Mumbai (INDIA)
June 08, 2020

In [ ]:

1.1 Dealing with vectors

In [1]:
```
v = vector(QQ, [1,-2,3])
u = vector(QQ, [3,2,4])
print(u+v)
print(2*u+3*v)
```

```
(4, 0, 7)
(9, -2, 17)
```

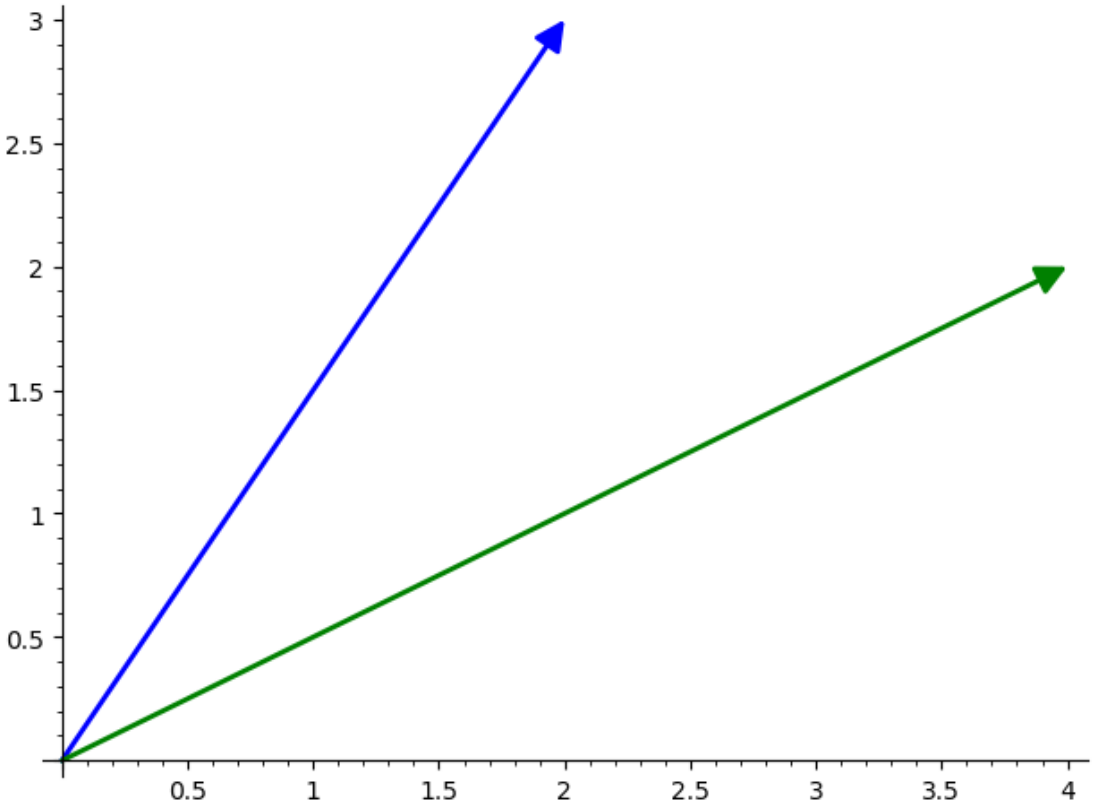
In [2]:
```
## Plotting
O = vector(QQ,[0,0,0])
vec_u = arrow3d(O, u, color='green')
vec_v = arrow3d(O, v, color='blue')
vec_uv = arrow3d(O, u+v, color='blue')
l1 = line3d((u,u+v),color='black',linestyle="--")
l2 = line3d((v,u+v),color='black',linestyle="--")
show(vec_u+vec_v+vec_uv+l1+l2,frame=True)
```

```
Graphics3d Object
```

In [3]:
```
w1 = vector([2,3])
w2 = vector([4,2])
```

In [4]:
```
plot(w1)+plot(w2, color='green')
```

Out[4]:




```
In [5]: ## Norm of vectors
        v.norm(2),v.norm(1),v.norm(oo)
```

```
Out[5]: (sqrt(14), 6, 3)
```

```
In [6]: u,v
```

```
Out[6]: ((3, 2, 4), (1, -2, 3))
```

```
In [7]: ## Dot product of vectors
        u*v
        u.dot_product(v)
        u.inner_product(v)
```

```
Out[7]: 11
```

```
In [8]: ## Cross Product
        w = u.cross_product(v)
        w
```

```
Out[8]: (14, -5, -8)
```

```
In [9]: print(u.dot_product(w))
        print(v.dot_product(w))
```

```
0
0
```

```
In [10]: cross_uv = arrow3d(O, w, color='red')
         show(vec_u+vec_v+vec_uv+cross_uv,frame=True)
```

```
Graphics3d Object
```

```
In [11]: w.dot_product(u), w.dot_product(v)
```

```
Out[11]: (0, 0)
```

1.1.1 Orthogonal Projection

```
In [12]: v.dot_product(u)/norm(u)^2*u
```

```
Out[12]: (33/29, 22/29, 44/29)
```

```
In [13]: def orthogonal_projection(v,u):
             p = v.dot_product(u)/norm(u)^2*u
             return p
```

```
In [14]: w = orthogonal_projection(v,u)
```

```
In [15]: u.dot_product(v-w)
```

```
Out[15]: 0
```

```
In [16]: norm(u+v).n()<=(norm(u)+norm(v)).n()
```

```
Out[16]: True
```

1.2 Defining Matrices

```
In [17]: ## Defining matrices
         M = matrix(QQ, [[1,3,7,-2],[2,-7,1,4],[9,10,16,2]])
         M
```

```
Out[17]: [ 1  3  7 -2]
         [ 2 -7  1  4]
         [ 9 10 16  2]
```

```
In [18]: show(M)
         latex(M)
```

```
[ 1  3  7 -2]
[ 2 -7  1  4]
[ 9 10 16  2]
```

```
Out[18]: \left(\begin{array}{rrrr}
         1 & 3 & 7 & -2 \\
         2 & -7 & 1 & 4 \\
         9 & 10 & 16 & 2
         \end{array}\right)
```

$$\left(\begin{array}{rrrr} 1 & 3 & 7 & -2 \\ 2 & -7 & 1 & 4 \\ 9 & 10 & 16 & 2 \end{array}\right)$$

```
In [19]: M1 = M*M.T
         M1

Out[19]: [ 63 -20 147]
         [-20  70 -28]
         [147 -28 441]

In [20]: M1.det()

Out[20]: 371028

In [21]: M1.inverse()

Out[21]: [   307/3786       8/631  -695/26502]
         [      8/631     21/1262      -2/631]
         [ -695/26502      -2/631 2005/185514]

In [22]: M1.inverse()*M1

Out[22]: [1 0 0]
         [0 1 0]
         [0 0 1]

In [23]: M.pseudoinverse()

Out[23]: [ -1717/13251       11/631   6002/92757]
         [ -2855/26502    -139/1262  9571/185514]
         [  4259/26502      69/1262 -2563/185514]
         [   -724/4417       22/631   1898/30919]

In [24]: ## Matrix from vectors
         u,v,w

Out[24]: ((3, 2, 4), (1, -2, 3), (33/29, 22/29, 44/29))
```

```
In [25]: column_matrix([u,v,w])

Out[25]: [    3     1 33/29]
         [    2    -2 22/29]
         [    4     3 44/29]

In [26]: A = matrix([randint(-10,10) for i in range(16)],nrows=4)

In [27]: A

Out[27]: [ -4  -7  -8  -3]
         [  8  -9 -10   4]
         [-10  -8  -7  -3]
         [  1  10  -9   6]

In [28]: 2*A+4*A

Out[28]: [-24 -42 -48 -18]
         [ 48 -54 -60  24]
         [-60 -48 -42 -18]
         [  6  60 -54  36]

In [29]: ## Standard matrices
         identity_matrix(3)
         zero_matrix(3)
         diagonal_matrix([1,2,3])

Out[29]: [1 0 0]
         [0 2 0]
         [0 0 3]
```

1.3 Elementary row operations

```
In [30]: M = matrix([randint(-10,10) for i in range(16)],nrows=4)
         M

Out[30]: [  5  -6  -6   4]
         [  5   9 -10   8]
         [ -8   3  -9  -3]
         [  2  10   7  -8]

In [31]: M[1,2]

Out[31]: -10

In [32]: M[0]

Out[32]: (5, -6, -6, 4)

In [33]: M[:,1]
```

```
Out[33]: [-6]
         [ 9]
         [ 3]
         [10]

In [34]: M

Out[34]: [  5  -6  -6   4]
         [  5   9 -10   8]
         [ -8   3  -9  -3]
         [  2  10   7  -8]

In [35]: S = copy(M)

In [36]: S.swap_rows(1,3)

In [37]: S

Out[37]: [  5  -6  -6   4]
         [  2  10   7  -8]
         [ -8   3  -9  -3]
         [  5   9 -10   8]

In [38]: S.rescale_row(2,10)

In [39]: S

Out[39]: [  5  -6  -6   4]
         [  2  10   7  -8]
         [-80  30 -90 -30]
         [  5   9 -10   8]

In [40]: S.add_multiple_of_row(1,2,-5)

In [41]: S

Out[41]: [   5   -6   -6    4]
         [ 402 -140  457  142]
         [ -80   30  -90  -30]
         [   5    9  -10    8]

In [42]: M.rref()

Out[42]: [1 0 0 0]
         [0 1 0 0]
         [0 0 1 0]
         [0 0 0 1]
```

In [43]:
```
## Step by step RREF
A=matrix(QQ, [[3,-2,1],[6, -1, 3],[4, 1, 8]])
m=A.nrows()
print ('The original matrix is')
show(A)
####
for k in range(0,m-1):
        if A[k,k] == 0:
            listA = [(A[j,k],j) for j in range(k,m)]
            maxv, pivot = max(listA)
            A[pivot,:],A[k,:]=A[k,:],A[pivot,:]
            print ('We permute rows %d and %d'%(k+1,pivot+1))
            show(A)
        for n in range(k+1,m):
            a=A[k,k]
            if A[n,k] != 0:
                print( "We add %s times row %d to row %d"%(-A[n,k]/a, k+1, n+1))
                A=A.with_added_multiple_of_row(n,k,-A[n,k]/a)
                show(A)
for k in range(m-1,-1,-1):
        for n in range(k-1,-1,-1):
            a=A[k,k]
            if A[n,k]!=0:
                print( "We add %s times row %d to the row %d"%(-A[n,k]/a, k+1, n+1))
                A=A.with_added_multiple_of_row(n,k,-A[n,k]/a)
                show(A)
for k in range(0,m):
        if A[k,k]!=1:
            print ('We divide row %d by %s'%(k+1,A[k,k]))
            A = A.with_rescaled_row(k,1/A[k,k])
            show(A)
####
```

```
The original matrix is


[ 3 -2  1]
[ 6 -1  3]
[ 4  1  8]


We add -2 times row 1 to row 2


[ 3 -2  1]
[ 0  3  1]
[ 4  1  8]
```

We add -4/3 times row 1 to row 3

```
[   3   -2    1]
[   0    3    1]
[   0 11/3 20/3]
```

We add -11/9 times row 2 to row 3

```
[   3   -2    1]
[   0    3    1]
[   0    0 49/9]
```

We add -9/49 times row 3 to the row 2

```
[   3   -2    1]
[   0    3    0]
[   0    0 49/9]
```

We add -9/49 times row 3 to the row 1

```
[   3   -2    0]
[   0    3    0]
[   0    0 49/9]
```

We add 2/3 times row 2 to the row 1

```
[   3    0    0]
[   0    3    0]
[   0    0 49/9]
```

We divide row 1 by 3

```
[   1    0    0]
[   0    3    0]
[   0    0 49/9]
```

We divide row 2 by 3

```
[   1    0    0]
[   0    1    0]
[   0    0 49/9]
```

We divide row 3 by 49/9

```
[1 0 0]
[0 1 0]
[0 0 1]
```

1.4 Solving System of linear equations

```
In [44]: ## Using solve commmand

In [45]: var('x,y,z')
         eq1 = 2*x-3*y+z==7
         eq2 = 3*x+5*y+3*z==17
         eq3 = 5*x+2*y+4*z==24
         solve([eq1,eq2,eq3],[x,y,z])

Out[45]: [[x == -14/19*r1 + 86/19, y == -3/19*r1 + 13/19, z == r1]]

In [46]: ## Solving Ax = b
         A = matrix(QQ,[[2,0,4],[-5,8,7],[9,15,12]])
         b = vector([10,-15,20])

In [47]: show(A, b)

[ 2  0  4]
[-5  8  7]
[ 9 15 12] (10, -15, 20)

In [48]: A.solve_right(b) # A\b

Out[48]: (815/303, -365/303, 350/303)

In [49]:  aug = A.augment(b,subdivide=True)

In [50]: aug.rref()

Out[50]: [       1        0        0| 815/303]
         [       0        1        0|-365/303]
         [       0        0        1| 350/303]

In [ ]:
```

1.5 Defining Vector Spaces

```
In [51]: V = QQ^4
         V

Out[51]: Vector space of dimension 4 over Rational Field

In [52]: v1 = vector(QQ, [1,-5,3,7])

In [53]: v1 in V

Out[53]: True

In [54]: r = V.random_element()
         r

Out[54]: (5/4, -1/2, 0, 1)

In [55]: v1 = vector(QQ, [1,-5,3,7])
         v2 = vector(QQ, [2,-15,13,17])
         v3 = vector(QQ, [13,-20,16,24])
         v4 = vector(QQ, [4,-30,26,34])

In [56]: B = [v1,v2,v3,v4]

In [57]: V.span(B)

Out[57]: Vector space of degree 4 and dimension 3 over Rational Field
         Basis matrix:
         [ 1  0  0  0]
         [ 0  1  0 -2]
         [ 0  0  1 -1]

In [58]: V.linear_dependence(B)

Out[58]: [
         (0, 1, 0, -1/2)
         ]

In [59]: D = [V.random_element() for i in range(4)]
         D

Out[59]: [(0, 0, 1, -63), (-1, -1/15, -1/13, -1), (0, 3/11, -100, 0), (-1, 1, 0, 3)]

In [60]: V.linear_dependence(D)

Out[60]: [

         ]

In [61]: H = column_matrix([v1,v2,v3,v4])
         H
```

```
Out[61]: [  1   2  13   4]
         [ -5 -15 -20 -30]
         [  3  13  16  26]
         [  7  17  24  34]

In [62]: H.rref()

Out[62]: [1 0 0 0]
         [0 1 0 2]
         [0 0 1 0]
         [0 0 0 0]

In [63]: u1 = vector(QQ, [1,2,3,4])
         u2 = vector(QQ, [2,1,-2,0])
         u3 = vector(QQ, [11,12,13,4])
         u4 = vector(QQ, [7,21,31,41])
         B1 = [u1,u2,u3,u4]
         V.linear_dependence(B1)==[]

Out[63]: True

In [64]: W = V.submodule_with_basis(B1)
         W

Out[64]: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1  2  3  4]
         [ 2  1 -2  0]
         [11 12 13  4]
         [ 7 21 31 41]

In [65]: u = vector(QQ,[3,5,7,13])

In [66]: d = W.coordinates(u)
         d

Out[66]: [651/115, 10/23, -21/115, -5/23]

In [67]: d[0]*u1+d[1]*u2+d[2]*u3+d[3]*u4

Out[67]: (3, 5, 7, 13)

In [68]: sum([d[i]*B1[i] for i in range(4)])==u

Out[68]: True

In [69]: M = column_matrix(B1)
         aug = M.augment(u)
         aug.rref()
```

```
Out[69]: [      1       0       0       0 651/115]
         [      0       1       0       0   10/23]
         [      0       0       1       0 -21/115]
         [      0       0       0       1   -5/23]

In [70]: w1 = vector(QQ, [1,-2,0,2])
         w2 = vector(QQ,[-1,1,0,-2])

In [71]: A1 = column_matrix([w1,w2])
         A1

Out[71]: [ 1 -1]
         [-2  1]
         [ 0  0]
         [ 2 -2]

In [72]: A2 =A1.augment(identity_matrix(4))
         A2.rref()

Out[72]: [   1    0    0   -1    0 -1/2]
         [   0    1    0   -1    0   -1]
         [   0    0    1    0    0 -1/2]
         [   0    0    0    0    1    0]

In [73]: A2.pivots()

Out[73]: (0, 1, 2, 4)

In [74]: B1 = [V.random_element() for i in range(3)]
         B2 = [V.random_element() for i in range(3)]
         W1 = V.span(B1)
         W2 = V.span(B2)

In [75]: W1.intersection(W2)

Out[75]: Vector space of degree 4 and dimension 2 over Rational Field
         Basis matrix:
         [           1            0   4449/28489  12324/28489]
         [           0            1 -20141/28489  57780/28489]

In [76]: W1+W2

Out[76]: Vector space of degree 4 and dimension 4 over Rational Field
         Basis matrix:
         [1 0 0 0]
         [0 1 0 0]
         [0 0 1 0]
         [0 0 0 1]

In [77]: dimension(W1+W2)==dimension(W1)+dimension(W2)-dimension(W1.intersection(W2))
```

```
Out[77]: True

In [78]: v = vector(QQ, [1,-2,3])
         u = vector(QQ, [3,2,4])

In [ ]:
```

1.6 Assignment 6

Problem 1. Create user defined function to find the angle between two vectors in degree. Hence use function to find the angle between vectors $(1,-1,3)$ and $(2,1,4)$.

```
In [ ]:
```

Problem 2. Create an user defined function to solve a system of linear equations using the Cramer's rule.

```
In [ ]:
```

Problem 3.

(i) Find the volume of the parallelepiped defined by the vectors $(0,1,-3)$, $(1,2,3)$ and $(-1,0,1)$.

(ii) Find an unit vector perpendicular to both $(2,1,1)$ and $(3,-1,4)$.

```
In [ ]:
```

Problem 4. Consider the matrix $M = \begin{pmatrix} 1 & 3 & -7 & 2 & -9 \\ 7 & 1 & 11 & 3 & 8 \\ 5 & -5 & 25 & 5 & 20 \\ 3 & 1 & 3 & -2 & 5 \\ 7 & 2 & 8 & 11 & -3 \end{pmatrix}$. Find the RREF of M and hence find the pivot columns.

```
In [ ]:
```

Problem 5. Complete the following linearly independent set of vectors $\{(2,0,2,0),(1,0,2,0)\}$ in $\mathbb{Q}^4$ to a basis of $\mathbb{Q}^4$.

```
In [ ]:
```

Problem 6. Show that the following set of vectors in $\mathbb{Q}^4$ is a basis of $\mathbb{Q}^4$.

$$B = \{(2,9,3,1),(3,3,4,10),(1,-2,12,-1),(4,5,5,-11)\}$$

Hence find the coordinates of $w = (2,3,5,9)$ with respect to B.

```
In [ ]:
```

Problem 7. Let $V = \mathbb{Q}^6$. Let

$$B_1 = \{(2, -1, 3, 1, 3, 5), (1, 2, -1, 0, 2, 1), (3, -4, 7, 2, 4, 9), (-1, 3, 5, -3, -7, 3)\}.$$

and

$$B_2 = \{(2, -1, 3, 1, 3, 5), (1, 2, -1, 0, 2, 1), (3, -4, 7, 2, 4, 9), (-1, 3, 5, -3, -7, 3)\}$$

Let $W_1 = L(B_1)$ and $W_2 = L(B_2)$ be linear span of B_1 and B_2 respectively.

Find bases of subspaces $W_1 + W_2$ and $W_1 \cap W_2$.

In [ ]:

Lecture 7 on SageMath

June 10, 2020

1 SageMath Lecture 7 (Linear Algebra with SageMath)

By Ajit Kumar
ICT Mumbai (INDIA)
Topics to be covered

- Change of basis
- Subspaces associated with a matrix
- Matrices over finite fields
- Vector spaces over finite Fields
- Linear Transformations
- Eigenvalues of Linear Transformations

1.1 Change of basis

Problem

Suppose $\mathcal{B}_1$ and $\mathcal{B}_2$ be two bases of finite dimensional vector space V. Let v be any vector in V. Let $\alpha = (\alpha_1, \ldots \alpha_n)$ and $\beta = (\beta_1, \ldots \beta_n)$ be coordinates of v with respect to bases $\mathcal{B}_1$ and $\mathcal{B}_2$.

Then how are α and β related?

```
In [1]: n = 5
        V = QQ^n

In [2]: B1 = [V.random_element() for i in range(n)]
        show(B1)
```

```
[(-1, 8, 0, 1, 0),
 (1, 0, -1, 1, 18),
 (1, 0, -1/2, 1, -1),
 (-4/3, 1, -4, 0, 7/2),
 (2, 3/8, -1, -1/3, 1)]
```

```
In [3]: V.linear_dependence(B1)==[]

Out[3]: True

In [4]: B2 = [V.random_element() for i in range(n)]
        show(B2)
```

```
[(-1, -1, -1/8, -1/9, 1/2),
 (1/39, -1, 2/3, 1/2, 1),
 (4, 1, 0, 1, -1/4),
 (2, 0, 4, 1/2, 0),
 (3/20, -3, -1, 13, -1)]
```

```
In [5]: V.linear_dependence(B2)==[]
```

```
Out[5]: True
```

```
In [6]: v = vector(QQ, [1,2,3,4,5])
```

```
In [7]: W1 = V.subspace_with_basis(B1)
        W2 = V.subspace_with_basis(B2)
```

```
In [8]: alpha = W1.coordinates(v)
        alpha
```

```
Out[8]: [202945/477862, 315685/477862, 583577/238931, -206982/238931, -338496/238931]
```

```
In [9]: beta = W2.coordinates(v)
        beta
```

```
Out[9]: [-75615608/3953469,
         12452762/912339,
         -42147844/11860407,
         -75750862/35581221,
         -33620/1317823]
```

```
In [10]: Mat_B1 = column_matrix(B1)
```

```
In [11]: Mat_B2 = column_matrix(B2)
```

```
In [12]: Mat_B1*vector(alpha)
```

```
Out[12]: (1, 2, 3, 4, 5)
```

```
In [13]: Mat_B2*vector(beta)
```

```
Out[13]: (1, 2, 3, 4, 5)
```

```
In [14]: bool(Mat_B2.inverse()*Mat_B1*vector(alpha))==bool(beta)
```

```
Out[14]: True
```

```
In [15]: M1 = Mat_B2.inverse()*Mat_B1
```

```
In [16]: show(M1)
```

```
[   -90354848/3953469    -49600552/1317823      2854620/1317823  -117811984/11860407     -1440
[      9284900/912339      10507756/304113       -183828/101371     21666433/2737017      3798
[  -57121888/11860407    -22336088/3953469       874836/1317823   -54261122/35581221      4009
[  -86175226/35581221   -87749486/11860407       356938/1317823 -285227603/106743663    -543097
[      -60900/1317823     -1126140/1317823       136660/1317823      -669260/3953469      -295
```

```
In [17]: M1.inverse()*vector(beta)
```

```
Out[17]: (202945/477862, 315685/477862, 583577/238931, -206982/238931, -338496/238931)
```

```
In [18]: aug  = Mat_B2.augment(Mat_B1).rref()
```

```
In [19]: show(aug)
```

```
[                  1                   0                   0                   0
[                  0                   1                   0                   0
[                  0                   0                   1                   0
[                  0                   0                   0                   1
[                  0                   0                   0                   0
```

```
In [20]: show(aug.submatrix(0,5))
```

```
[   -90354848/3953469    -49600552/1317823      2854620/1317823  -117811984/11860407     -1440
[      9284900/912339      10507756/304113       -183828/101371     21666433/2737017      3798
[  -57121888/11860407    -22336088/3953469       874836/1317823   -54261122/35581221      4009
[  -86175226/35581221   -87749486/11860407       356938/1317823 -285227603/106743663    -543097
[      -60900/1317823     -1126140/1317823       136660/1317823      -669260/3953469      -295
```

1.2 Spaces accociated with matrices

The four fundamental subspaces associated to an $m \times n$ matrix A are: * The column space, col(A), spanned by all of the columns of A. This is a subspace of $\mathbb{R}^m$. * The row space, row(A), spanned by all of the rows of A. This is a subspace of $\mathbb{R}^n$. This also happens to be the column space of A^T. * The nullspace (or kernel),null(A), consisting of all those vectors x for which $Ax = 0$. This is a subspace of $\mathbb{R}^n$.

- The left nullspace, which is just the nullspace of A^T. This is a subspace of $\mathbb{R}^m$.

```
In [21]: A =  matrix(QQ, [[1, 3,-2 ,0 ,2 , 0],
                          [2, 6 ,-5, -2, 4,-3],
                          [3, 9, -7, -2, 6, -3],
                          [2, 6, 0, 8, 4, 18],
                         [3, 9, -2, 8, 6, 18]])
```

```
In [22]: A
```

```
Out[22]: [ 1  3 -2  0  2  0]
         [ 2  6 -5 -2  4 -3]
         [ 3  9 -7 -2  6 -3]
         [ 2  6  0  8  4 18]
         [ 3  9 -2  8  6 18]

In [23]: A.column_space()

Out[23]: Vector space of degree 5 and dimension 3 over Rational Field
         Basis matrix:
         [1 0 1 0 1]
         [0 1 1 0 0]
         [0 0 0 1 1]

In [24]: A.row_space()

Out[24]: Vector space of degree 6 and dimension 3 over Rational Field
         Basis matrix:
         [1 3 0 4 2 0]
         [0 0 1 2 0 0]
         [0 0 0 0 0 1]

In [25]: A.right_kernel()

Out[25]: Vector space of degree 6 and dimension 3 over Rational Field
         Basis matrix:
         [   1    0    0    0 -1/2    0]
         [   0    1    0    0 -3/2    0]
         [   0    0    1 -1/2    1    0]

In [26]: A.rank()

Out[26]: 3

In [27]: A.T.right_kernel()

Out[27]: Vector space of degree 5 and dimension 2 over Rational Field
         Basis matrix:
         [ 1  0  0  1 -1]
         [ 0  1 -1 -1  1]

In [28]: A.extended_echelon_form(subdivide=True)

Out[28]: [    1     3     0     4     2     0|    0     0    -6   -22    21]
         [    0     0     1     2     0     0|    0     0    -3 -21/2    10]
         [    0     0     0     0     0     1|    0     0   2/3   5/2  -7/3]
         [------------------------------------+----------------------------]
         [    0     0     0     0     0     0|    1     0     0     1    -1]
         [    0     0     0     0     0     0|    0     1    -1    -1     1]

In [ ]:
```

1.3 Matrices over finite fields

```
In [29]: ## Matrices over Z/5Z
         A = matrix(GF(5),3,3,[[1,2,72],[2,67,4],[3,0,2]])
         show(A)
```

```
[1 2 2]
[2 2 4]
[3 0 2]
```

```
In [30]: A.row_space()
```

```
Out[30]: Vector space of degree 3 and dimension 3 over Finite Field of size 5
         Basis matrix:
         [1 0 0]
         [0 1 0]
         [0 0 1]
```

```
In [31]: A.column_space()
```

```
Out[31]: Vector space of degree 3 and dimension 3 over Finite Field of size 5
         Basis matrix:
         [1 0 0]
         [0 1 0]
         [0 0 1]
```

```
In [32]: A.right_kernel()
```

```
Out[32]: Vector space of degree 3 and dimension 0 over Finite Field of size 5
         Basis matrix:
         []
```

```
In [33]: ## Matrices over polynomial rings
         D.<x> = QQ[]
         A = matrix(D, 3, 3, [[x^2 - x, -x + 5,x^2-2*x],
                              [2*x^2-x-8,7,x^2-x+1],
                              [x^2 - 8, x + 2,x+1]])
         show(A)
```

```
[      x^2 - x        -x + 5     x^2 - 2*x]
[2*x^2 - x - 8             7   x^2 - x + 1]
[      x^2 - 8         x + 2         x + 1]
```

```
In [34]: A.right_kernel()
```

```
Out[34]: Free module of degree 3 and rank 1 over Univariate Polynomial Ring in x over Rational
         Echelon basis matrix:
         [      -x^3 - x^2 + 8*x + 5 x^4 - 3*x^3 - 8*x^2 + 17*x  2*x^3 - 4*x^2 - 10*x + 40]
```

```
In [35]: A.row_space()
```

```
Out[35]: Free module of degree 3 and rank 2 over Univariate Polynomial Ring in x over Rational
         Echelon basis matrix:
         [                                1        -1/28*x^2 - 3/14*x + 13/28 1/56*x^3 + 5/56*x
         [                                0         2*x^3 - 4*x^2 - 10*x + 40       -x^4 + 3*x
```

```
In [36]: A.column_space()
```

```
Out[36]: Free module of degree 3 and rank 2 over Univariate Polynomial Ring in x over Rational
         Echelon basis matrix:
         [                    1 1/10*x^2 + 3/10*x + 1     1/10*x^2 + 3/10*x]
         [                    0                     1                      1]
```

```
In [37]: ## Matrix over finite fields with a generator
         F.<a> = FiniteField(5^2)
         print(F)
         A = matrix(F, 3, 4, [[  1,   a,     1+a,  a^3+a^5],
                              [  a, a^4,   a+a^4,  a^4+a^8],
                              [a^2, a^6, a^2+a^6, a^5+a^10]])
         K = A.right_kernel(); K
```

```
Finite Field in a of size 5^2
```

```
Out[37]: Vector space of degree 4 and dimension 2 over Finite Field in a of size 5^2
         Basis matrix:
         [      1       0 3*a + 4 2*a + 2]
         [      0       1     2*a 3*a + 3]
```

```
In [38]: QQ^5
         VectorSpace(QQ,5)
```

```
Out[38]: Vector space of dimension 5 over Rational Field
```

1.4 Vector Space over Finite Fields

```
In [39]: V = VectorSpace(GF(3),5)
         V.cardinality()
```

```
Out[39]: 243
```

```
In [40]: len(list(V.subspaces(4)))
```

```
Out[40]: 121
```

```
In [41]: V.basis_matrix()
```

```
Out[41]: [1 0 0 0 0]
         [0 1 0 0 0]
         [0 0 1 0 0]
         [0 0 0 1 0]
         [0 0 0 0 1]
```

1.5 Random Matrices

```
In [42]: #random_matrix?

In [43]: random_matrix(ZZ,3,4,x=-100,y=100)

Out[43]: [ 59 -71  78 -25]
         [ 74 -91  48  99]
         [-96 -91  77  39]

In [44]: random_matrix(QQ,3,4,num_bound=10,den_bound=5)

Out[44]: [  -2 -6/5  5/3  2/3]
         [ 4/5 -1/2   -1   -6]
         [   0 -4/5 -5/4 -1/3]

In [45]: C=random_matrix(QQ, 6, 7, algorithm='echelonizable', rank=5); C

Out[45]: [   -5   -35    60   249  1206  2408 -1381]
         [    5    26   -52  -213 -1028 -2046  1169]
         [    3    19   -37  -151  -729 -1453   832]
         [   -2   -12    29   119   561  1112  -637]
         [    1     9   -16   -68  -327  -651   374]
         [    3    20   -35  -145  -702 -1401   803]

In [46]: sage: F.<x>=GF(2^3)
         sage: B = random_matrix(F, 4, 5, algorithm='echelonizable', rank=3); B

Out[46]: [x^2 + x + 1       x + 1         x^2           0     x^2 + x]
         [        x^2           1     x^2 + 1       x + 1           0]
         [    x^2 + x         x^2           x     x^2 + x x^2 + x + 1]
         [      x + 1         x^2       x + 1           1         x^2]

In [47]: A = random_matrix(QQ, 5, algorithm='diagonalizable',
                           eigenvalues=[2,3,-1], dimensions=[1,2,2]); A # random

Out[47]: [ 38 -16   0 -92  16]
         [ 35 -13  -4 -80  16]
         [ 27 -12  -1 -60  12]
         [  9  -4   0 -21   4]
         [ -1   0  -4  16   3]

In [48]: R.<x>=QQ[]
         L2 = R^2
         L2

Out[48]: Ambient free module of rank 2 over the principal ideal domain Univariate Polynomial R

In [49]: R

Out[49]: Univariate Polynomial Ring in x over Rational Field
```

```
In [50]: L2.span([[(x^2+x)/(x^2-3*x+2),1/5],[(x^2+2*x)/(x^2-4*x+3),x]])

Out[50]: Free module of degree 2 and rank 2 over Univariate Polynomial Ring in x over Rational
         Echelon basis matrix:
         [x/(x^3 - 6*x^2 + 11*x - 6)  2/15*x^2 - 17/75*x - 1/75]
         [                         0 x^3 - 11/5*x^2 - 3*x + 4/5]

In [51]: M= span( [[x, x^2+1], [1/x, x^3]], R); M

Out[51]: Free module of degree 2 and rank 2 over Univariate Polynomial Ring in x over Rational
         Echelon basis matrix:
         [          1/x           x^3]
         [            0 x^5 - x^2 - 1]
```

1.6 Linear Transformations

- linear_transformation(A, side='left')
- linear_transformation(D, C, A, side='left')
- linear_transformation(D, C, f)
- linear_transformation(D, C, images)

See **help(linear_transformation)** for more details
Problem
Consider a linear transformation $T\colon \mathbb{Q}^4 \to \mathbb{Q}^4$ given by

$$T\left(x_1, x_2, x_3, x_4\right) = \begin{pmatrix} 8\,x_1 + 8\,x_2 + 24\,x_3 - 26\,x_4 \\ 12\,x_1 + 31\,x_2 + 96\,x_3 - 56\,x_4 \\ -3\,x_1 - 8\,x_2 - 25\,x_3 + 14\,x_4 \\ 3\,x_1 + 4\,x_2 + 12\,x_3 - 11\,x_4 \end{pmatrix}$$

- Find the various concepts related to T.
- Find matrix of T w.r.t. abritrary bases on domain and codomains.
- Find eigenvalues of T and show that it is independent of the basis.

```
In [52]: reset()
         n,m = 4, 4
         V = QQ^n
         W = QQ^m

In [53]: x1, x2, x3, x4 = var('x1, x2, x3, x4')
         f(x1,x2,x3,x4) = [8*x1 + 8*x2 + 24*x3 - 26*x4,\
                           12*x1 + 31*x2 + 96*x3 - 56*x4,\
                           -3*x1 - 8*x2 - 25*x3 + 14*x4,\
                           3*x1 + 4*x2 + 12*x3 - 11*x4]

In [54]: T = linear_transformation(V, W, f)
         T
```

```
Out[54]: Vector space morphism represented by the matrix:
         [  8  12  -3   3]
         [  8  31  -8   4]
         [ 24  96 -25  12]
         [-26 -56  14 -11]
         Domain: Vector space of dimension 4 over Rational Field
         Codomain: Vector space of dimension 4 over Rational Field

In [55]: T.matrix(side='right')

Out[55]: [  8   8  24 -26]
         [ 12  31  96 -56]
         [ -3  -8 -25  14]
         [  3   4  12 -11]

In [56]: E = identity_matrix(4)
         e1 = E[0]
         e2 = E[1]
         e3 = E[2]
         e4 = E[3]

In [57]: T(e1),T(e2)

Out[57]: ((8, 12, -3, 3), (8, 31, -8, 4))

In [58]: v = vector([1,3,5,7])

In [59]: T(v)

Out[59]: (-30, 193, -54, -2)

In [60]: T.domain()
         T.codomain()

Out[60]: Vector space of dimension 4 over Rational Field

In [61]: T.image()
         T.kernel()

Out[61]: Vector space of degree 4 and dimension 0 over Rational Field
         Basis matrix:
         []

In [62]: T.is_injective()

Out[62]: True

In [63]: T.is_surjective()

Out[63]: True
```

```
In [64]: d1 = vector(QQ, [ 1, -3,  2, -1])
         d2 = vector(QQ, [ 0,  1,  0,  1])
         d3 = vector(QQ, [-1,  2, -1, -1])
         d4 = vector(QQ, [ 2, -8,  4, -3])
         D = [d1, d2, d3, d4]
         V.linear_dependence(D)==[]

Out[64]: True

In [65]: V1 = V.subspace_with_basis(D)
         V1

Out[65]: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1 -3  2 -1]
         [ 0  1  0  1]
         [-1  2 -1 -1]
         [ 2 -8  4 -3]

In [66]: c1 = vector(QQ, [1, 0, 7, 4])
         c2 = vector(QQ, [-1, 1, 8, -1])
         c3 = vector(QQ, [2, 0, 1, 2])
         c4 = vector(QQ, [3, 0, 5, -3])
         C = [c1,c2,c3,c4]
         W1 = W.subspace_with_basis(C)
         W1

Out[66]: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1  0  7  4]
         [-1  1  8 -1]
         [ 2  0  1  2]
         [ 3  0  5 -3]

In [67]: W1.coordinates(T(d1))

Out[67]: [-18122/99, 167, 32998/99, -8533/99]

In [68]: M = column_matrix([c1,c2,c3,c4,T(d1),T(d2),T(d3),T(d4)]).rref()
         M.submatrix(0,4)

Out[68]: [-18122/99   2807/99   -382/33   -3247/9]
         [      167       -25        10       328]
         [ 32998/99  -5365/99    758/33    5942/9]
         [ -8533/99   1222/99   -158/33   -1517/9]

In [69]: S = T.restrict_domain(V1).restrict_codomain(W1)
         S
```

```
Out[69]: Vector space morphism represented by the matrix:
         [-18122/99       167  32998/99  -8533/99]
         [  2807/99       -25  -5365/99   1222/99]
         [  -382/33        10    758/33   -158/33]
         [  -3247/9       328    5942/9   -1517/9]
         Domain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1 -3  2 -1]
         [ 0  1  0  1]
         [-1  2 -1 -1]
         [ 2 -8  4 -3]
         Codomain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1  0  7  4]
         [-1  1  8 -1]
         [ 2  0  1  2]
         [ 3  0  5 -3]

In [70]: S.matrix(side='right')

Out[70]: [-18122/99   2807/99   -382/33   -3247/9]
         [      167       -25        10       328]
         [ 32998/99  -5365/99    758/33    5942/9]
         [ -8533/99   1222/99   -158/33   -1517/9]

In [71]: M.submatrix(0,4)

Out[71]: [-18122/99   2807/99   -382/33   -3247/9]
         [      167       -25        10       328]
         [ 32998/99  -5365/99    758/33    5942/9]
         [ -8533/99   1222/99   -158/33   -1517/9]

In [72]: T.eigenvalues()

Out[72]: [3, 2, -1, -1]

In [73]: T.charpoly(x)

Out[73]: x^4 - 3*x^3 - 3*x^2 + 7*x + 6

In [74]: A = T.matrix(side='right')

In [75]: A.eigenvalues()

Out[75]: [3, 2, -1, -1]

In [76]: T1 = T.restrict_domain(V1).restrict_codomain(V1)
         T1.eigenvalues()

Out[76]: [3, 2, -1, -1]
```

```
In [77]: B = T1.matrix(side='right')

In [78]: T2 = T.restrict_domain(W1).restrict_codomain(W1)
         T2.eigenvalues()

Out[78]: [3, 2, -1, -1]

In [79]: A.is_similar(B)

Out[79]: True
```

1.7 Eigenvalues of a linear Transformations

```
In [ ]:
```

Thm: Let $T\colon V \to V$ be a linear transformation and $\mathcal{B}_\infty$ and $\mathcal{B}_\in$ be any two bases of V. Let M_1 and M_2 be matrices of T with respect to the bases $\mathcal{B}_\infty$ and $\mathcal{B}_\in$ respectively. Then M_1 and M_2 have the same eigenvalues.
We illustrate this using a LT over $\mathbb{R}^3$. Note that M_1 and M_2 are similar matrices.

```
In [80]: V = QQ^3
         x, y, z = var('x, y, z')
         outputs = [x+z,x+y+2*z,2*x+y+3*z]
         T_symbolic(x, y, z)=outputs
         T=linear_transformation(QQ^3, QQ^3, T_symbolic)

In [81]: #First FiBasis $B_1$
         u1=vector(QQ,[2,1,-1])
         u2=vector(QQ,[1,3,2])
         u3=vector(QQ,[5,-2,0])
         CC=[u1,u2,u3]
         #CC=[V.random_element() for i in range(3)]
         B1=V.subspace_with_basis(CC);B1

Out[81]: Vector space of degree 3 and dimension 3 over Rational Field
         User basis matrix:
         [ 2  1 -1]
         [ 1  3  2]
         [ 5 -2  0]

In [82]: M1=T.restrict_domain(B1).restrict_codomain(B1)
         M1=M1.matrix(side='right')
         eigen1=M1.eigenvalues()
         show(M1)
         show(eigen1)

[  -4/7  -19/7 -86/35]
[   5/7   29/7  97/35]
[   2/7    6/7   10/7]
```

```
[0, 0.6972243622680054?, 4.302775637731994?]
```

```
In [83]: #Second Basis $B_1$
         v1=vector(QQ,[0,1,1])
         v2=vector(QQ,[1,-1,1])
         v3=vector(QQ,[1,1,-1])
         #CC=[v1,v2,v3]
         CC=[V.random_element() for i in range(3)]
         B2=V.subspace_with_basis(CC);B2
```

```
Out[83]: Vector space of degree 3 and dimension 3 over Rational Field
         User basis matrix:
         [1/73    1    2]
         [  -8   -1    1]
         [   0    4    0]
```

```
In [84]: M2=T.restrict_domain(B2).restrict_codomain(B2)
         M2=M2.matrix(side='right')
         eigen2=M2.eigenvalues()
         show(M2)
         show(eigen2)
```

$$\begin{bmatrix} 4251/1169 & -1241/167 & 2336/1169 \\ -20949/85337 & 144/167 & 4/1169 \\ 48291/170674 & 54/167 & 586/1169 \end{bmatrix}$$

```
[0, 0.6972243622680054?, 4.302775637731994?]
```

```
In [85]: M1.is_similar(M2)
```

```
Out[85]: True
```

Example:

Find the linear transformation (explicitely) $T\colon \mathbb{Q}^3 \to \mathbb{Q}^3$ that takes the basis $\{(1,1,-2),(-1,1,1),(1,-1,1)\}$, to vectors $\{(2,1,1),(1,-1,1),(1,1,-1)\}$ of $\mathbb{Q}^3$.

```
In [86]: x = SR.var('x',4)
         x
```

```
Out[86]: (x0, x1, x2, x3)
```

```
In [87]: n = 3
         V = W = QQ^n
```

```
In [88]: u1=vector(QQ,[1,1,-2])
         u2=vector(QQ,[-1,1,1])
         u3=vector(QQ,[1,-1,1])
         BD=[u1,u2,u3]
         B=V.subspace_with_basis(BD)
```

```
In [89]: v1=vector(QQ,[2,1,1])
         v2=vector(QQ,[1,-1,1])
         v3=vector(QQ,[1,1,-1])
         img=[v1,v2,v3]

In [90]: phi = linear_transformation(V,W,img)

In [91]: phi

Out[91]: Vector space morphism represented by the matrix:
         [ 2  1  1]
         [ 1 -1  1]
         [ 1  1 -1]
         Domain: Vector space of dimension 3 over Rational Field
         Codomain: Vector space of dimension 3 over Rational Field

In [92]: E = identity_matrix(3)
         E

Out[92]: [1 0 0]
         [0 1 0]
         [0 0 1]

In [93]: sum([x[i+1]*phi(E[i]) for i in range(n)])

Out[93]: (2*x1 + x2 + x3, x1 - x2 + x3, x1 + x2 - x3)

In [94]: g(x1,x2,x3) = [2*x1 + x2 + x3, x1 - x2 + x3, x1 + x2 - x3]
         psi = linear_transformation(V,W,g)

In [95]: phi == psi

Out[95]: True

In [96]: T1+T2

Out[96]: Vector space morphism represented by the matrix:
         [  23350/99       -190   71149/99  -65215/99]
         [-107324/99        998  167320/99  -34027/99]
         [   7592/99        -70   -1273/99   -5246/99]
         [       764       -668       1022      -1215]
         Domain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1 -3  2 -1]
         [ 0  1  0  1]
         [-1  2 -1 -1]
         [ 2 -8  4 -3]
         Codomain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1  0  7  4]
         [-1  1  8 -1]
         [ 2  0  1  2]
         [ 3  0  5 -3]
```

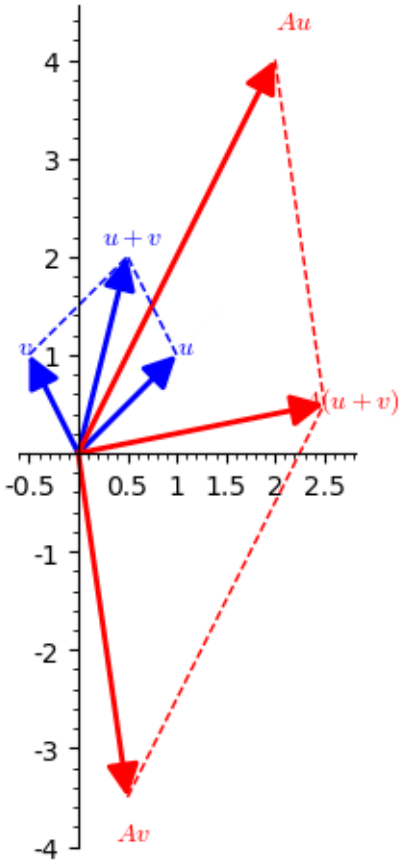
```
In [97]: T1*T2 ## Composition of T1 and T2
```

```
Out[97]: Vector space morphism represented by the matrix:
         [ -72519721/99   21544514/33   15869537/99     3753065/9]
         [-132429892/99   39337637/33   28962074/99     6853469/9]
         [  -1333208/99     396886/33     294454/99       69010/9]
         [     -1082876        964904        236728        616442]
         Domain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1  0  7  4]
         [-1  1  8 -1]
         [ 2  0  1  2]
         [ 3  0  5 -3]
         Codomain: Vector space of degree 4 and dimension 4 over Rational Field
         User basis matrix:
         [ 1 -3  2 -1]
         [ 0  1  0  1]
         [-1  2 -1 -1]
         [ 2 -8  4 -3]
```

```
In [ ]:
```

1.8 Geometric of linear transformations

```
In [98]: A=matrix(QQ,[[1,5],[1,-1]]).transpose()
         u=matrix(QQ,[1,1])
         v=matrix(QQ,[-1/2,1])
         u=vector(QQ,[u[0,0],u[0,1]])
         v=vector(QQ,[v[0,0],v[0,1]])
         r =1.1
         e=vector(QQ,[1/10,1/10])
         u1 = A*u
         v1 = A*v
         p1=plot(u,color='blue')+plot(v,color='blue')
         p1=p1+plot(u+v,color='blue')
         p1=p1+line((u,u+v),linestyle="--",color='blue')
         p1=p1+line((v,u+v),linestyle="--",color='blue')
         p2=plot(A*u,color='red')+plot(A*v,color='red')
         p2=p2+plot(A*(u+v),color='red')
         p2=p2+line((A*u,A*(u+v)),linestyle="--",color='red')
         p2=p2+line((A*v,A*(u+v)),linestyle="--",color='red')
         t=text('$u$',r*u)+text('$v$',r*v)+text('$u+v$',r*(u+v))
         t=t+text('$Au$',r*A*u,color='red')+text('$Av$',r*A*v,color='red')+text('$A(u+v)$',r*A
         show(p1+p2+t,aspect_ratio=1)
```




```
In [ ]: @interact
        def LT(A=matrix(QQ,[[1,1/2],[1,-1]]).transpose(),u=matrix(QQ,[1,1]),v=matrix(QQ,[-1/2,
            u=vector(QQ,[u[0,0],u[0,1]])
            v=vector(QQ,[v[0,0],v[0,1]])
            r =1.1
            e=vector(QQ,[1/10,1/10])
            u1 = A*u
            v1 = A*v
            p1=plot(u,color='blue')+plot(v,color='blue')
            p1=p1+plot(u+v,color='blue')
            p1=p1+line((u,u+v),linestyle="--",color='blue')
            p1=p1+line((v,u+v),linestyle="--",color='blue')
            p2=plot(A*u,color='red')+plot(A*v,color='red')
            p2=p2+plot(A*(u+v),color='red')
            p2=p2+line((A*u,A*(u+v)),linestyle="--",color='red')
            p2=p2+line((A*v,A*(u+v)),linestyle="--",color='red')
            t=text('$u$',r*u)+text('$v$',r*v)+text('$u+v$',r*(u+v))
            t=t+text('$Au$',r*A*u,color='red')
            t=t+text('$Av$',r*A*v,color='red')
```

```
            t=t+text('$A(u+v)$',r*A*(u+v),color='red')
            show(p1+p2+t,aspect_ratio=1)
```

```
In [99]: ## Multivariate polynomials
         R.<x,y> = QQbar[]
         f = 3*x^2 - 2*y + 7*x^2*y^2 + 5*x*y-4*x+5
```

```
In [100]: f.coefficients()
```

```
Out[100]: [7, 3, 5, -4, -2, 5]
```

```
In [101]: f.monomials()
```

```
Out[101]: [x^2*y^2, x^2, x*y, x, y, 1]
```

```
In [102]: f.constant_coefficient()
```

```
Out[102]: 5
```

```
In [103]: f.degree()
```

```
Out[103]: 4
```

```
In [104]: f.dict()
```

```
Out[104]: {(2, 0): 3, (0, 1): -2, (2, 2): 7, (1, 1): 5, (1, 0): -4, (0, 0): 5}
```

```
In [105]: f.derivative(x)
```

```
Out[105]: 14*x*y^2 + 6*x + 5*y - 4
```

```
In [ ]:
```

1.9 Assignments

- Find the fundamental subspaces assiciated with the matrix

$$B = \begin{pmatrix} 1 & 3 & -2 & 0 & 2 & 0 \\ 2 & 6 & -5 & -2 & 4 & -3 \\ 0 & 0 & 5 & 10 & 0 & 15 \\ 2 & 6 & 0 & 8 & 4 & 18 \end{pmatrix}.$$

- Consider the linear tranformations $T\colon \mathbb{R}^4 \to \mathbb{R}^3$ defined by

$$T(x,y,z,w) = (x+z+w, x+y+2z-w, 2x+y+3z-2w)$$

  and $S\colon \mathbb{R}^3 \to \mathbb{R}^4$ definded by

$$S(x,y,z) = (x+z, x+3y+2z, 2x-y+3z, y-z).$$

  Find the composition $S \circ T$ and the matrix associated with composition. Hence show that the matrix of the composition is product of the matrcies associated with T and S.

- Consider a linear tranformation $T\colon \mathbb{R}^3 \to \mathbb{R}^3$ defined by

$$T(x, y, z) = (x + z, x + y + 2z, 2x + y + 3z)$$

(i) Find the matrix associated with this transformation with respect to basis $\{(1, 1, 0), (-1, 1, 1), (0, -1, 1)\}$
of domain and $\{(0, 1, 1), (1, -1, 1), (1, 1, -1)\}$ of the codomain.

- Consider the linear tranformation $T\colon \mathbb{R}^4 \to \mathbb{R}^3$ defined as

$$T(x_1, x_2, x_3, x_4) = (3x_1 + 7x_2 + x_3 - 4x_4, 2x_1 + 5x_2 + x_3 - 3x_4, -x_1 - 2x_2 + x_4)$$

Let us consider a basis

$$\mathcal{D} = \{(1, -3, 2, -1), (0, 1, 0, 1), (-1, 2, -1, -1), (2, -8, 4, -3)\}$$

of domain of T and a basis

$$\mathcal{E} = \{(0, 1, -3), (-1, 2, -1), (2, -4, 3)\}$$

of the codmain of T. Let ρ_B be the linear transformation that takes the standard basis $\mathcal{B}$ of $\mathbb{R}^4$ to the basis $\mathcal{D}$ of $\mathbb{R}^4$. Let ρ_C be the linear transformation that takes standard basis $\mathcal{C}$of $\mathbb{R}^3$ to the basis $\mathcal{E}$ of $\mathbb{R}^3$. Let M_{BC} be the matrix of T with resect to the standard bases $\mathcal{B}$ and $\mathcal{C}$. Let M_{DE} be the matrix of T with resect to standard bases $\mathcal{D}$ and $\mathcal{E}$. Then show that

$$\rho_C M_{BC} \rho_B^{-1} = M_{DE}$$

- Conside two bases $B = \{(1, 3, -4), (1, -2, 3), (1, -1, -3)\}$ and $C = \{(-1, 1, 1), (3, 1, 3), (2, -1, 1)\}$ of $V = \mathbb{Q}^3$. Let us consider the linear transformation $T\colon \mathbb{R}^3 \to \mathbb{R}^3$ such that

  $$T(1, 3, -4) = (2, 1, 1), T(1, -2, 3) = (-1, 1, 3), T(1, -1, -3) = (1, 2, 4).$$

  Find $T(3, -2, 5)$ and also find the matrix $[T]_B^C$

Inner Product with SageMath

Ajit Kumar
ICT Mumbai

June 15, 2020

1 Inner Product Spaces with SageMath

1.1 Standard dot product in $\mathbb{R}^n$

```
In [1]: u = vector(RR,[3,-1,2])
        v = vector(RR,[4,2,-2])

In [2]: u*v

Out[2]: 6.00000000000000

In [3]: u.dot_product(v)

Out[3]: 6.00000000000000

In [4]: u.inner_product(v)

Out[4]: 6.00000000000000

In [5]: norm(v)

Out[5]: 4.89897948556636

In [6]: ## verifying various inequalities and identities.
```

1.2 Inner product defined by a positive definite symmetric matrix

Example: Let $A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$. Define an inner product on $\mathbb{R}^3$ as $< u, v >= v^T Au$.

Use the the Gram-Schmidt orthogonalization process to find an orthonormal basis of from the standard basis vectors e_1, e_2, e_3 with respect to the above inner product.

```
In [7]: A=matrix([[2,-1,0],[-1,2,-1],[0,-1,2]])
        A.is_positive_definite()

Out[7]: True
```

```
In [8]: def inpA(u,v):
            return v.dot_product(A*u)

In [9]: e1=vector([1,0,0])
        e2=vector([0,1,0])
        inpA(e1,e2)

Out[9]: -1

In [10]: sqrt(inpA(e1,e1))

Out[10]: sqrt(2)

In [11]: ## Find a vector perpendicular to (1,0,0) under the above inner product

In [12]: v1 = vector([1,0,0])
         v2 = vector([3,2,1])

In [13]: p = v2- inpA(v2,v1)/inpA(v1,v1)*v1

In [14]: show(p)

(1, 2, 1)


In [15]: inpA(p,v1)

Out[15]: 0
```

1.3 Inner Product in $C[0,1]$

$$< f, g >:= \int_0^1 f(x)g(x)dx$$

```
In [16]: def inpf(f,g):
             return integral(f(x)*g(x),x,0,1)

In [17]: p1(x) = x^3+2*x+2
         p2(x) = x+1

In [18]: inpf(p1,p2)

Out[18]: 307/60

In [19]: def normf(f):
             return sqrt(inpf(f,f))

In [20]: def projf(g,f):
             return inpf(g,f)/inpf(f,f)*f

In [21]: projf(p2,p1)

Out[21]: x |--> 2149/4736*x^3 + 2149/2368*x + 2149/2368
```

1.4 Inner product on $M_n(\mathbb{R})$

$$\langle A, B\rangle := trace(AB^T).$$

```
In [22]: def inpMnR(A,B):
             return (A*B.T).trace()

In [23]: A = matrix(QQ,[[2,-1,0],[1,4,2],[7,9,2]])
         B = matrix(QQ,[[3,2,1],[1,2,0],[5,3,1]])

In [24]: inpMnR(A,B)

Out[24]: 77

In [25]: sqrt(inpMnR(A,A))

Out[25]: 4*sqrt(10)
```

1.5 Inner product on $\mathcal{P}_n(\mathbb{R})$

Let p and q be two polynomials in $\mathcal{P}_n(\mathbb{R})$. Then define

$$\langle p, q\rangle := p(0)q(0) + p(1)q(1) + \cdots + p(n)q(n).$$

```
In [26]: def inpPol(f,g,n):
             a = sum([f(x=i)*g(x=i) for i in range(n+1)])
             return a

In [27]: f(x) = 3*x^3+4*x^2+2*x+1
         g(x) = -2*x^3+x^2+2*x-1

         inpPol(f,g,3)

Out[27]: -5366
```

Example.
Find a vector v in $\mathcal{P}_3(\mathbb{R})$ which is orthogonal to $f(x) = x^2 + x + 2$ in the above inner product space.

```
In [28]: f(x) = x^2+x+2
         g(x) = x^3
         p(x) = inpPol(g,f,3)/inpPol(f,f,3)*f
         h(x) = g(x)-p(x)

In [29]: show(h(x))

x^3 - 223/140*x^2 - 223/140*x - 223/70


In [30]: inpPol(f,h,3)

Out[30]: 0
```

1.6 Gram-Schmidt orthogonalization process

Let $\{v_1, \ldots, v_n\}$ be a basis of a f. d. inner product space V. Then an orthogonal basis $\{q_1, \ldots q_n\}$ can be obtained as

$$q_1 := v_1, q_k = v_k - \sum_{i=1}^{k-1} \frac{\langle v_k, q_i \rangle}{\langle q_i, q_i \rangle} q_i, \quad i = 2, \ldots, n.$$

```
In [31]: def proj(v, u):
             p = v.dot_product(u)/(norm(u)^2)*u
             return(p)

In [32]: def Gram_Schmidt(S):
             n = len(S)
             E=[S[0]/norm(S[0]).n()]
             for k in range(1,n):
                 q = S[k]-sum([proj(S[k],E[i]) for i in range(k)])
                 E.append(q/norm(q).n(digits=5))
             return(E)

In [ ]:
```

Example Find an orthonormal basis from

$$\{(2, -1, 3, 1), (3, 3, 4, 0), (1, -2, 0, -1), (4, 5, 5, -1)\}.$$

```
In [33]: sage: v1 = vector([2, -1, 3, 1])
         sage: v2 = vector([3, 3, 4, 0])
         sage: v3 = vector([1, -2, 0, -1])
         sage: v4 = vector([4, 5, 5, -1])
         sage: L = [v1,v2,v3,v4]

In [34]: Q = column_matrix(Gram_Schmidt(L))

In [35]: Q*Q.T

Out[35]: [      1.0000  0.000037193   -9.0599e-6 -0.000015736]
         [ 0.000037193       1.0000 -0.000030577  0.000021815]
         [  -9.0599e-6 -0.000030577       1.0000  0.000018597]
         [-0.000015736  0.000021815  0.000018597      0.99997]

In [36]: ## Gram-Schmidt using inbuilt function
         A = column_matrix(RDF, L)
         G, M = A.T.gram_schmidt()
         show(G)
         show(M)
```

```
[ -0.5163977794943222  0.25819888974716115  -0.7745966692414835 -0.25819888974716115]
[-0.22941573387056202  -0.9176629354822471  -0.2294157338705617  0.22941573387056183]
[ 0.48914242603205416 -0.28673866353603183  -0.1518028218720169  -0.8096150499840894]
[  0.6644105970267493  0.09491579957524975  -0.5694947974514992  0.47457899787624974]
```

```
[-3.8729833462074166                0.0                 0.0                 0.0]
[ -3.872983346207418  -4.358898943540673                 0.0                 0.0]
[-0.7745966692414834    1.37649440322337  1.8722348030882072                 0.0]
[ -4.389381125701739 -6.8824720161168536  0.5734773270720622 -0.1898315991505009]
```

```
In [37]: Q = G.T
         R = M.T
         show(Q)
         show(R)
```

```
[ -0.5163977794943222 -0.22941573387056202  0.48914242603205416   0.6644105970267493]
[ 0.25819888974716115  -0.9176629354822471 -0.28673866353603183  0.09491579957524975]
[ -0.7745966692414835  -0.2294157338705617  -0.1518028218720169  -0.5694947974514992]
[-0.25819888974716115  0.22941573387056183  -0.8096150499840894  0.47457899787624974]
```

```
[-3.8729833462074166  -3.872983346207418 -0.7745966692414834  -4.389381125701739]
[                0.0  -4.358898943540673    1.37649440322337 -6.8824720161168536]
[                0.0                 0.0  1.8722348030882072  0.5734773270720622]
[                0.0                 0.0                 0.0 -0.1898315991505009]
```

Example: Let $A = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$. Define an inner product on $\mathbb{R}^3$ as $< u, v >= v^T Au$.

Use the the Gram-Schmidt orthogonalization process to find an orthonormal basis of from the standard basis vectors e_1, e_2, e_3 with respect to the above inner product.

```
In [38]: A=matrix([[2,-1,0],[-1,2,-1],[0,-1,2]])
         A.is_positive_definite()
```

```
Out[38]: True
```

```
In [39]: def inpA(u,v):
             return v.dot_product(A*u)
```

```
In [40]: e1=vector([1,0,0])
         e2=vector([0,1,0])
         e3=vector([0,0,1])
```

```
In [41]: u1=e1
         u2=e2-inpA(e2,u1)/inpA(u1,u1)*u1
         u3=e3-inpA(e3,u1)/inpA(u1,u1)*u1-inpA(e3,u2)/inpA(u2,u2)*u2
```

```
In [42]: u1 = u1/sqrt(inpA(u1,u1))
         u2 = u2/sqrt(inpA(u2,u2))
         u3 = u3/sqrt(inpA(u3,u3))
         show([u1,u2,u3])
```

```
[(1/2*sqrt(2), 0, 0),
 (1/3*sqrt(3/2), 2/3*sqrt(3/2), 0),
 (1/2*sqrt(1/3), sqrt(1/3), 3/2*sqrt(1/3))]
```

```
In [43]: ## Check
         inpA(u1,u3)
```

```
Out[43]: 0
```

Example: Orthogonalize the standar basis $\{1, x, x^2, x^3, x^4\}$. respect to the inner product $\langle f, g\rangle := \int_0^1 f(x)g(x)dx$.

```
In [44]: f(x)=1
         g(x)=x
         h(x)=x^2
         k(x) = x^3
         l(x) = x^4
```

```
In [45]: u=f
         v=g-projf(g,u)
         w=h-projf(h,u)-projf(h,v)
         t=k-projf(k,u)-projf(k,v)-projf(k,w)
         s=l-projf(l,u)-projf(l,v)-projf(l,w)-projf(l,t)
```

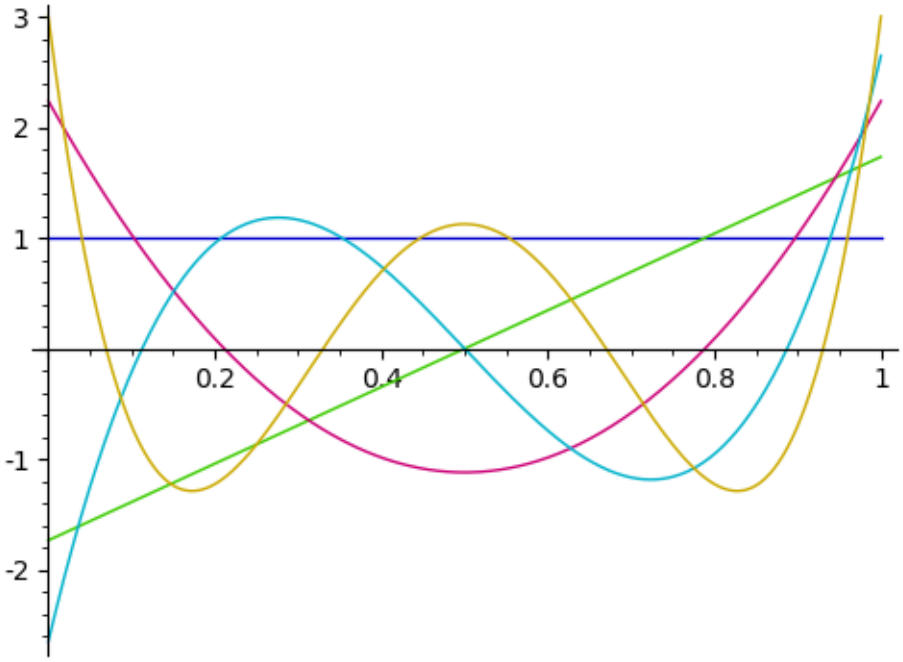
```
In [46]: u = u/normf(u)
         v = v/normf(v)
         w = w/normf(w)
         t = t/normf(t)
         s = s/normf(s)
```

```
In [47]: show([u(x),v(x),w(x),t(x),l(x)])
```

```
[1,
 3*sqrt(1/3)*(2*x - 1),
 5*sqrt(1/5)*(6*x^2 - 6*x + 1),
 7*sqrt(1/7)*(20*x^3 - 30*x^2 + 12*x - 1),
 x^4]
```

```
In [48]: plot([u,v,w,t,s],(x,0,1),figsize=5)
```

```
Out[48]:
```



2 Projections

2.1 Projection to the line $y = mx$

Let $P_m \colon \mathbb{R}^2 \to \mathbb{R}^2$ denote the projection about the line $y = mx$. Let θ denote the angle between the positive x-axis and the line $y = mx$.

The key observation is that the transformation Q_m can be accomplished in three steps:

1. First rotate through $-\theta$ (line coincides with the x-axix)
2. Second take projection about the x-axis
3. Finally rotate back through θ

Thus

$$P_m = R_\theta \circ P_0 \circ R_{-\theta}.$$

Note that $R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, $R_{-\theta} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$ and $P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$.

Note that $\tan\theta = m$, which implies $\cos\theta = \frac{1}{1+m^2}$ and $\sin\theta = \frac{m}{1+m^2}$. Subtituting the values and simplying, it is easy to verifying we get

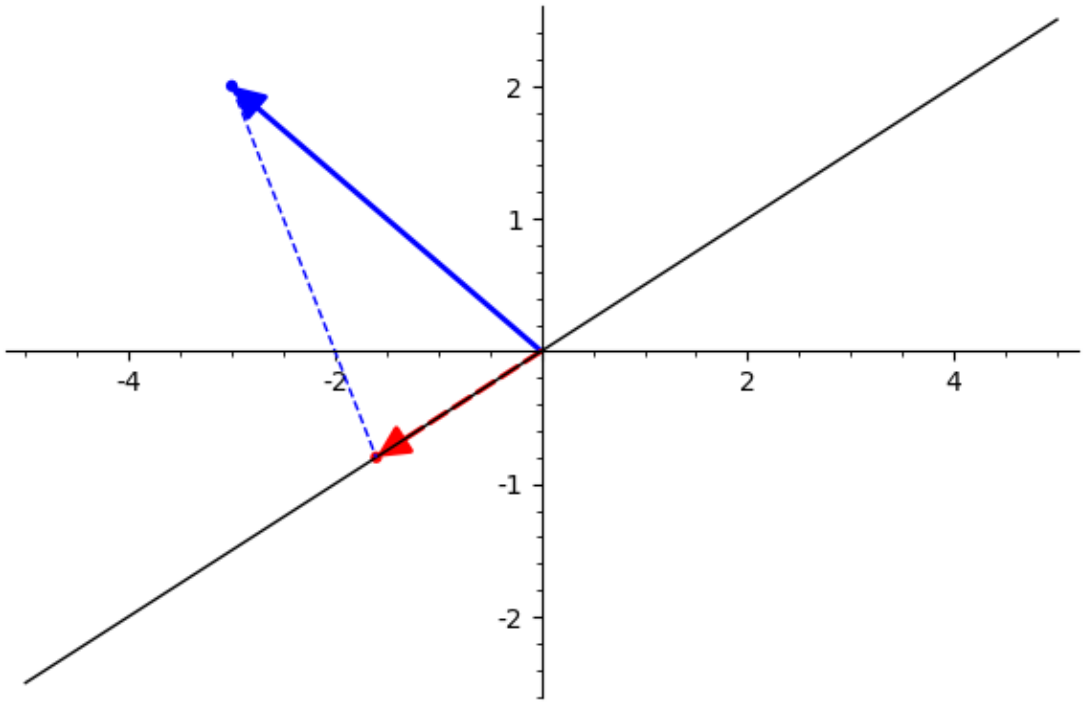
$$P_m = \frac{1}{1+m^2} \begin{bmatrix} 1 & m \\ m & m^2 \end{bmatrix}.$$

2.2 Problem:

Find the projection of $(4,-1)$ about the line $y = \frac{1}{2}x$. Also plot the to show the line and the point along with its reflection.

```
In [49]: m =1/2
         P = 1/(1+m^2)*matrix([[1, m],[m,m^2]])
         v = vector([-3,2])
         Pv = P*v

In [50]: line = plot(m*x,(x,-5,5),color='black')
         point = point2d(v,color='blue',size=20)
         Ppoint = point2d(Pv,color='red',size=20)
         vecP= v.plot(color='blue')
         vecPv= Pv.plot(linestyle="--",color='red')
         ll = ll=line2d([v,Pv],linestyle="--",color='blue')
         line+point+Ppoint+vecP+vecPv+ll
```

Out[50]:



Orthogonal Projection of a vector onto a subspace
Let W be a subspace of $\mathbb{R}^n$ of dimension k and $\{v_1, \ldots, v_k\}$ be a basis of W. Let $v \in \mathbb{R}^n$. We want to find the vector p which the orthogonal projection of v onto W.

Note that $v \in W$, therefore, there exist scalars $x_1, \ldots, x_k$ such that

$$p = x_1v_1 + \cdots + x_kv_k = Ax$$

wheere $A = [v_1 \ \cdots \ v_k]$ and $x = [x_1 \ \cdots \ x_k]^T$.
It is clear that $v - p = v - Ax \in W^T$. Hence $(v_i)^T(v - Ax) = 0$ for $1 \leq i \leq k$. This is same as

$$A^T(v - Ax) = 0 \implies x = (A^TA)^{-1}(A^Tv).$$

Hence

$$p = A(A^TA)^{-1}(A^Tv).$$

The matrix $Q = A(A^TA)^{-1}A^T$ is called the "projection matrix for the subspace W".

```
In [51]: V=QQ^6
         v1 = vector(QQ,[1, -1, 2, 0, 3, 2])
         v2 = vector(QQ,[2, 1, 1, -3, 1, 5])
         v3 = vector(QQ,[1, 5, -1, -9, -3, 11])
         v4 = vector(QQ,[2, -5, 4, 6, 7, -4])
         v = vector(QQ,[1, 2, 3, 4, 5, 6])

In [52]: W=V.span([v1,v2,v3,v4]);W

Out[52]: Vector space of degree 6 and dimension 3 over Rational Field
         Basis matrix:
         [   1    0    0    0    0    0]
         [   0    1    0   -2 -1/3  8/3]
         [   0    0    1   -1  4/3  7/3]

In [53]: # The basis of W are non zero row vectors of rref of the matrix whose rows are v1,...,v
         A=column_matrix([v1,v2,v3,v4]).transpose()
         A.rref()

Out[53]: [   1    0    0    0    0    0]
         [   0    1    0   -2 -1/3  8/3]
         [   0    0    1   -1  4/3  7/3]
         [   0    0    0    0    0    0]

In [54]: B=W.basis_matrix().transpose()
         print( 'The matrix of the basis of W is')
         print(B)

The matrix of the basis of W is
[   1    0    0]
[   0    1    0]
[   0    0    1]
[   0   -2   -1]
[   0 -1/3  4/3]
[   0  8/3  7/3]
```

```
In [55]: pr=(B.transpose()*B).inverse()*B.transpose()*v
         print( 'The projection vector of v onto the space W is')
         print(B*pr)
```

```
The projection vector of v onto the space W is
(1, -137/94, 158/47, -21/47, 467/94, 186/47)
```

```
In [56]: ## Check
         v4.dot_product(v-B*pr)
```

```
Out[56]: 0
```

Example Find the polynomial of degree at most 3 that is closed to $f(x) = \sin(x)$ under the inpper product $\langle f, g\rangle = \int_0^{2\pi} f(x)g(x)dx$.

```
In [57]: def inp(f,g):
             return integral(f(x)*g(x),x,0,2*pi)

         def projf(g,f):
             prgf=inp(g,f)/inp(f,f)*f
             return prgf
```

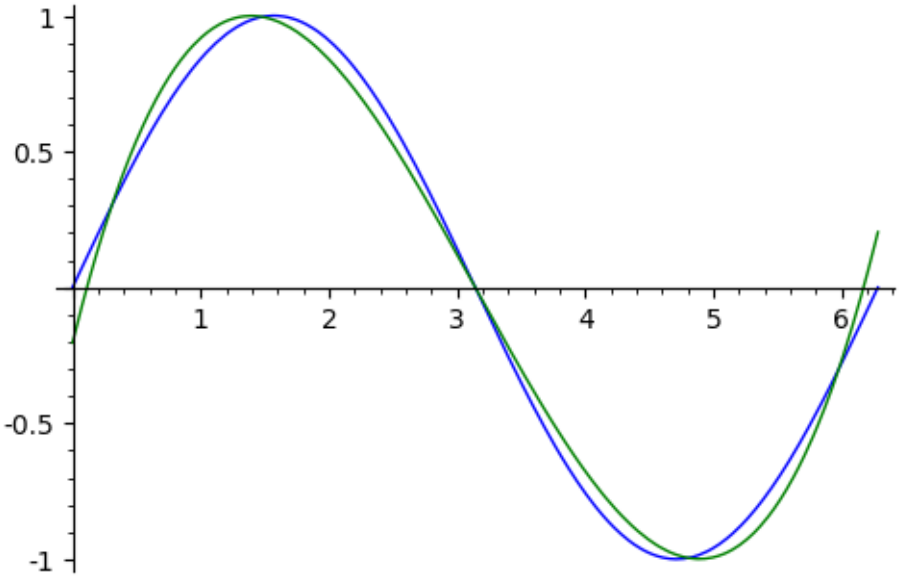
```
In [58]: v1(x) = 1
         v2(x) = x
         v3(x) = x^2
         v4(x) = x^3
         f(x) = sin(x)
```

```
In [59]: u1=v1
         u2=v2-projf(v2,u1)
         u3=v3-projf(v3,u1)-projf(v3,u2)
         u4=v4-projf(v4,u1)-projf(v4,u2)-projf(v4,u3)
```

```
In [60]: p=projf(f,u1)+projf(f,u2)+projf(f,u3)+projf(f,u4)
         show(p(x))
```

```
3*(pi - x)/pi^2 - 7/2*(15*pi - pi^3)*(10*pi^3 - 18*pi^2*(pi - x) - 5*x^3 - 5*pi*(4*pi^2 - 6*pi*(
```

```
In [61]: plt = plot(f,0,2*pi)+plot(p,0,2*pi,color='green')
         show(plt, figsize=5)
```



In [ ]:

3 Reflections

3.1 Reflection about line $y = mx$.

Let $Q_m \colon \mathbb{R}^2 \to \mathbb{R}^2$ denote reflection about the line $y = mx$. Let θ denote the angle between the positive x-axis and the line $y = mx$.

The key observation is that the transformation Q_m can be accomplished in three steps:

1. First rotate through $-\theta$ (line coincides with the x-axix)
2. Second reflect about the x-axis
3. Finally rotate back through θ

Thus

$$Q_m = R_\theta \circ Q_0 \circ R_{-\theta}.$$

Note that $R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, $R_{-\theta} = \begin{bmatrix} \cos\theta & \sin\theta \\ -sin\theta & \cos\theta \end{bmatrix}$ and $Q_0 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

Note that $\tan\theta = m$, which implies $\cos\theta = \frac{1}{1+m^2}$ and $\sin\theta = \frac{m}{1+m^2}$. Subtituting the values and simplying, it is easy to verifying we get

$$Q_m = \frac{1}{1+m^2} \begin{bmatrix} 1-m^2 & 2m \\ 2m & m^2-1 \end{bmatrix}.$$

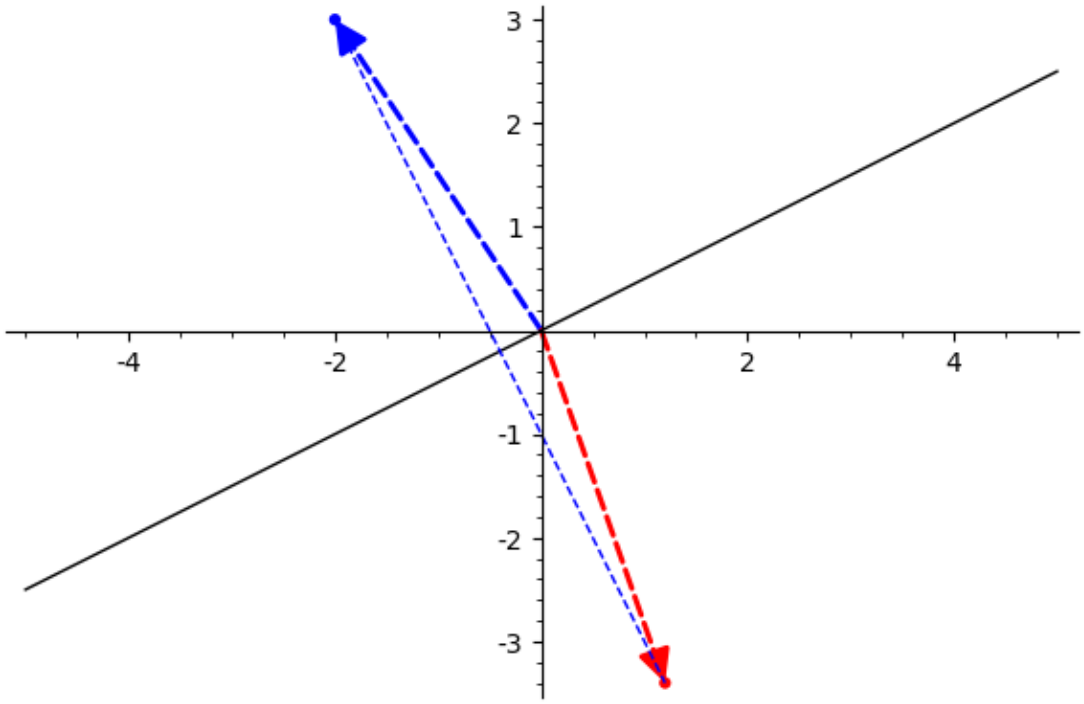
3.2 Problem:

Find the reflection of $(2,4)$ about the line $y = \frac{1}{2}x$. Also plot the to show the line and the point along with its reflection.

```
In [62]: m = 1/2
         Q = 1/(1+m^2)*matrix([[1-m^2, 2*m],[2*m,m^2-1]])
         v = vector([-2,3])
         Rv = Q*v
```

```
In [63]: line = plot(m*x,(x,-5,5),color='black')
         point = point2d(v,color='blue',size=20)
         Rpoint = point2d(Rv,color='red',size=20)
         vecP= v.plot(linestyle="--",color='blue')
         vecRv= Rv.plot(linestyle="--",color='red')
         ll = ll=line2d([v,Rv],linestyle="--",color='blue')
         line+point+Rpoint+vecP+vecRv+ll
```

Out[63]:



3.3 Reflection About a Hypersurface

If W is a hyper surface in $\mathbb{R}^n$ and u is a unit vector perpendicural to W then the reflection of a vector $x \in \mathbb{R}^n$ about W is given by

$$Ref(x) = x - 2(u^T u)x = (I - 2u^T u)x = Rx.$$

Example Find the reflectio of the vector $x = (3,2,5)$ about the plane defined by vectors $v_1 = (-1,2,0)$ and $v_2 = (3,1,0)$.

```
In [64]: v1=vector(QQ,[-1,2,0])
         v2=vector(QQ,[3,1,0])
         v3=vector(QQ,[3,2,5])

In [65]: u=v1.cross_product(v2)/norm(v1.cross_product(v2))

In [66]: R=matrix(RDF,u).transpose()*matrix(RDF,u)

In [67]: H=identity_matrix(RDF,3)-2*R

In [68]: ref=H*v3

In [69]: O = vector([0,0,0])
         vec_v1 = arrow3d(O,v1,color='red')
         vec_v2 = arrow3d(O,v2,color='blue')
         vec_v3 = arrow3d(O,v3,color='green')
         vec_ref = arrow3d(O,ref,color='black')
         vecs = vec_v1+vec_v2+vec_v3 +vec_ref
         show(vecs)

Graphics3d Object


In [70]: var('s,t')
         pl = parametric_plot3d(s*v1+t*v2,(s,-2,2),(t,-2,2),color='red',opacity=.4)
         l1 = line3d((ref,v3),color='black',linestyle="--")

In [71]: show(vecs+pl+l1,frame=False)

Graphics3d Object


In [ ]:
```

Example

Find the reflection of a point $x = (1,1,1,1)$ about the subspace spanned by vectors random 3 vectors in $\mathbb{R}^4$.

```
In [72]: V= QQ^4
         x= vector(QQ, [1,1,1,1])
         w1 = V.random_element()
         w2 = V.random_element()
         w3 = V.random_element()
         S=[w1,w2,w3]
         V.linear_dependence(S)==[]

Out[72]: True
```

```
In [73]: W = V.span(S); W
         A = W.basis_matrix()
         u= A.T.kernel().basis_matrix()
         u = vector(u)
         u = u/norm(u)
         u

Out[73]: (12061/9994320415*sqrt(9994320415), 10962/9994320415*sqrt(9994320415), -98631/999432041

In [74]: # Check if u is orthogonal to W
         [u.dot_product(S[i]) for i in range(3)]

Out[74]: [0, 0, 0]

In [75]: Rx= x-2*x.dot_product(u)*u
         print("The R(x)={}".format(Rx))

The R(x)=(2359849813/1998864083, 2326956743/1998864083, -953161747/1998864083, 2022299273/199886
```

3.4 Least Square Solutions

Problem Let A be a 6×4 matrix whose entries are between -10 and 10 of maximum rank. Then A is a linear map from $\mathbb{R}^4$ to $\mathbb{R}^6$. Suppose $b = (1,2,3,4,5,6)$.

(i) Does b lie in the image space (column space of A?

(ii) Find the point $x^* \in \mathbb{R}^4$ such that Ax^* is at the smmallest distance from b. (Note that Ax^* is notging but the orthogonal projection of b onto the image space (column space of A.)

```
In [76]: m, n = 4, 6
         A = random_matrix(ZZ,n,m,x=-10,y = 10)

In [77]: show(A)

[ 9  1 -8 -4]
[ 4  1 -3  2]
[ 3  1  4 -7]
[-9  4 -9  1]
[-3 -3  2  4]
[-9 -4  8 -4]


In [78]: b = vector([1,2,3,4,5,6])

In [79]: x = SR.var('x',m)
         x

Out[79]: (x0, x1, x2, x3)
```

```
In [80]: X = vector(x)
         X
```

```
Out[80]: (x0, x1, x2, x3)
```

```
In [81]: err = A*X-b
         f = sum([err[i]^2 for i in range(n)])
         show(f)
```

```
(9*x0 + 4*x1 - 8*x2 + 4*x3 + 6)^2 + (9*x0 + x1 - 8*x2 - 4*x3 - 1)^2 + (9*x0 - 4*x1 + 9*x2 - x3 +
```

```
In [82]: gradf = f.gradient()
         show(gradf)
```

```
(554*x0 + 50*x1 - 138*x2 - 68*x3 + 158, 50*x0 + 88*x1 - 162*x2 - 2*x3 + 34, -138*x0 - 162*x1 + 4
```

```
In [83]: s = solve([gradf[i]==0.0 for i in range(m)],x,solution_dict=True )
         show(s[0])
```

```
{x0: -14442012/39943499,
 x1: -38501210/39943499,
 x2: -16751861/39943499,
 x3: -19163316/39943499}
```

```
In [84]: print('The point $x^*$ is:')
         X1 = vector([s[0][d].n(digits=8) for d in x])
         show(X1)
```

```
The point $x^*$ is:
```

```
(-0.36156101, -0.96389177, -0.41938892, -0.47976057)
```

```
In [85]: print('The point closed to b is')
         show(A*X1)
```

```
The point closed to b is
```

```
(1.0562128, -2.1114902, -0.36780648, 2.6932218, 1.2185382, 5.6735471)
```

```
In [86]: ## Solution using the orthogonal projection of b on Im(A)
```

```
In [87]: X2 = (A.T*A).inverse()*A.T*b
         show(X2.n())
```

```
(-0.361561013971260, -0.963891771224148, -0.419388922337525, -0.479760573804513)
```

```
In [88]: print('The point closed to b is')
         show(A*X2.n(digits=8))
```

```
The point closed to b is
```

```
(1.0562128, -2.1114902, -0.36780648, 2.6932218, 1.2185382, 5.6735471)
```

```
In [ ]:
```

```
In [89]: #Solution using the inbuilt function 'lstsq' in numpy.linalg package
         import numpy as np
         ss= np.linalg.lstsq(A,b,rcond=None)
         ss[0]
```

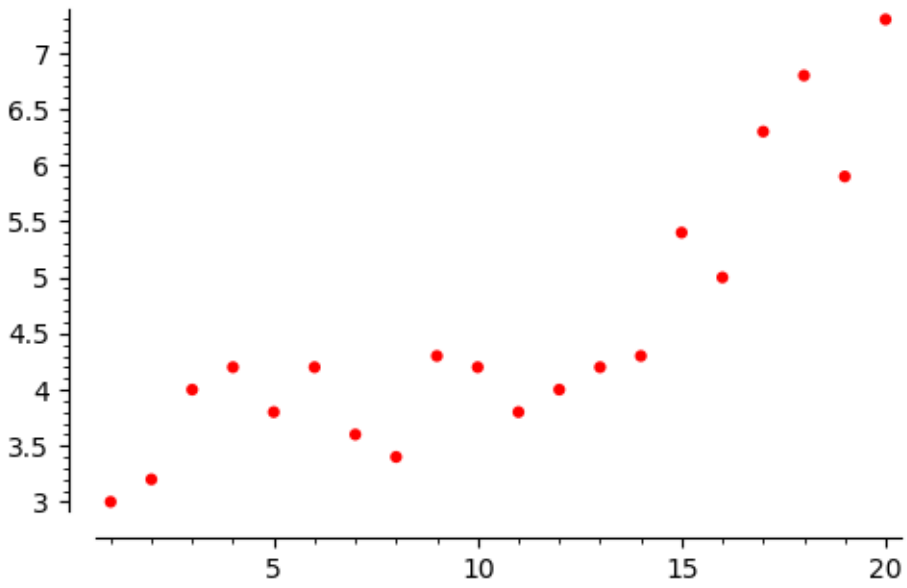
```
Out[89]: array([-0.36156101, -0.96389177, -0.41938892, -0.47976057])
```

```
In [ ]:
```

```
In [90]: ## Fitting a quadratic to a set of 20 points in the plane
```

```
In [91]: x = list(range(1,21))
         y = [3.0,3.2, 4.0, 4.2,3.8,4.2,3.6,3.4,4.3,4.2,3.8,4.0,4.2,4.3,5.4,5.0,6.3,6.8,5.9,7.3]
         data = [(x[i],y[i]) for i in range(20)]
         points(data,size=20,color='red',figsize=5)
```

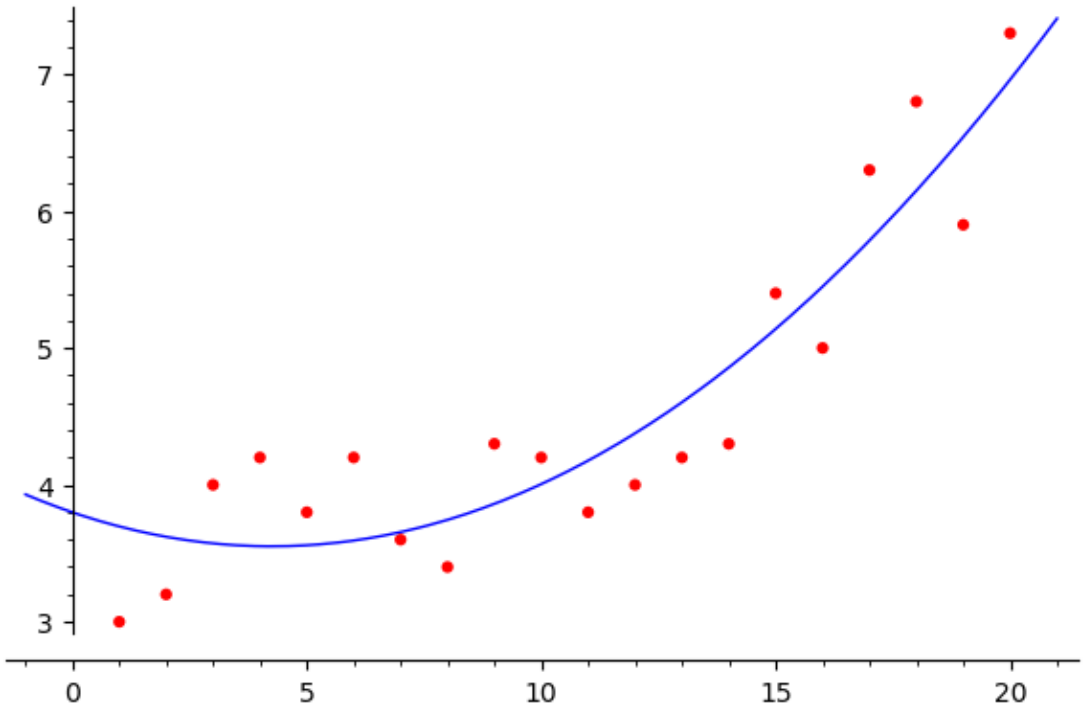
```
Out[91]:
```



In [92]:
```
## Solution using the inbuilt function
var('a,b,c,d,t')
model(t)=a*t^2+b*t+c
fit=find_fit(data,model,solution_dict=True)
model.subs(fit) #
show(fit)
plot(model.subs(fit),(t,-1,21))+points(data,size=20,color='red') # Nice plot
```

```
{a: 0.013744588742438002, b: -0.11683185236061178, c: 3.799385964918422}
```

Out[92]:



In [93]:
```
ones = [1]*20
sqx = [x[i]^2 for i in range(20)]
A = column_matrix([sqx,x,ones])
```

In [94]:
```
print('The solution is orthogonal projection:')
show((A.T*A).inverse()*A.T*vector(y))
```

```
The solution is orthogonal projection:
```

```
(0.0137445887445887, -0.116831852358168, 3.79938596491228)
```

```
In [95]: ## Using Calculus
         err = [(y[i]-(a*x[i]^2+b*x[i]+c)) for i in range(20)]
         f = sum([err[i]^2 for i in range(20)])
         show(f)
```

```
(400*a + 20*b + c - 7.30000000000000)^2 + (361*a + 19*b + c - 5.90000000000000)^2 + (324*a + 18*
```

```
In [96]: fa = f.diff(a)
         fb = f.diff(b)
         fc = f.diff(c)
         s = solve([fa==0.0,fb==0.0,fc==0.0],[a,b,c],solution_dict=True)
         show(s[0])
```

```
{a: 127/9240, b: -6837/58520, c: 43313/11400}
```

```
In [97]: print('The solution is:')
         print('a=',s[0][a].n(), ', b=',s[0][b].n(),', c=',s[0][c].n() )
```

```
The solution is:
a= 0.0137445887445887 , b= -0.116831852358168 , c= 3.79938596491228
```

```
In [ ]:
```

3.5 Assignment 8

1. Take a random 4×4 symmetric positive definite matrix A and define an inner product on $\mathbb{R}^4$ as
$$\langle x, y \rangle := x^T A y.$$
   (i) Verify that it is a norm.
   (ii) Verify the Cauchy-Schwarz and traingle inequalities.
   (iii) Verify the cosine law.
   (iv) Find an orthonormal basis of $\mathbb{R}^4$ starting with an arbitrary basis of $\mathbb{R}^4$.

```
In [ ]:
```

2. Find the polynomial p of degree at most 2 that is closed to $f(x) = |x|$ under the inpper product $\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)dx$. Also plot the graph of f and p.

```
In [ ]:
```

3. Consider $V = \mathbb{R}^6$. Let W be a subspace spanned by vectors $v_1 = (1,2,3,4,5,6)$, $v_2 = (2,-1,3,0,5,2)$, $v_3 = (-2,-2,-3,-4,-5,-6)$ and $v_4 = (-2,2,-3,0,-5,-2)$. Find orthonormal bases of W and $W^{\perp}$. (Use the standard inner product)

In [ ]:

4. Find the orthogonal projection of a vector $v(1,2,3,4,5)$ onto the subspace spanned W by vectors vectors $(1,-1,2,-2,0)$, $(-2,2,-3,3,1)$, $(2,0,1,0,0)$ and $(1,1,-4,3,1)$.

In [ ]:

5. Find the reflection of a vector v(1,2,3,4,5) about subspace spanned W by vectors vectors (1,−1,2,−2,0) , (−2,2,−3,3,1) , (2,0,1,0,0) and (1,0,3,,1).

In [ ]:

6. Find the best fit cubic to set of points

   $\{(-5,-350),(-4,-200),(-3,-100),(-2,-35),(-1,-6),(0,2),(1,3),(2,2),(3,27),(4,65),(5,150)\}$

   Use both inbuit function in SageMath and also notion of orthogonal projection.

In [ ]:

7. Consider the inner product on $\mathcal{P}_3(x)$ defined by

$$\langle p,q\rangle := p(0)q(0)+p(1)q(1)+p(2)q(2)+p(3)q(3).$$

   Find the polynomial closed to the the space $U = \text{span}\{3, 1+x, 2-x^2\}$ from $1+x+x^2+x^3$.

In [ ]:

SageMath Lecture 9

June 17, 2020

1 SageMath Lecture 9

2 Some Applications of Linear Algebra

By Ajit Kumar (ICT Mumbai, INDIA)
June 17, 2020

2.1 Topics to be covered:

- **Forcasting Problems**
- **Message Endoding-Decoding**
- **Google PageRank**
- **Image Processing SVD**
- **Linear Dynamic (Epidemic) Model**

jupyterLab

- JupyterLab is a next-generation web-based user interface for Project Jupyter.
- JupyterLab will eventually replace the classic Jupyter Notebook

Installing JupyterLab

- **sage –pip install jupyterlab**
- **sage -n jupyterlab**

2.2 Taxi Fleet Distribution

A travel company has a fleet of 5000 cars for renting. A car rented at one location can be returned to any of the four locations A, B, C and D. The various fractions of cars returned to the fours locations are given in the table below.

```
In [2]: T= table([["A",1/5, 1/10, 3/10, 3/20],
                  ["B",2/5, 2/5, 1/4, 3/20],
                  ["C",1/10, 1/5, 3/10, 1/10],
                  ["D",3/10, 3/10, 3/20, 3/5]],frame=True,
                header_row=["locations","A","B","C","D"]
               ,align='center')
        show(T)
```

locations	A	B	C	D
A	1/5	1/10	3/10	3/20
B	2/5	2/5	1/4	3/20
C	1/10	1/5	3/10	1/10
D	3/10	3/10	3/20	3/5

- Suppose on Monday there are 1800 cars at location A, 1000 cars at the location B, 500 at C and 1700 cars at the location D.
- What will be the approximate distribution of cars on Thursday?
- How should the company distribute these cars at various locations in order to serve the custoers smoothly?

```
In [2]: ## The matrix
        A=matrix(RR,[[0.20, 0.10, 0.30, 0.15],
                     [0.40, 0.40, 0.25, 0.15],
                     [0.10, 0.20, 0.30, 0.10],
                     [0.30, 0.30, 0.15, 0.60]])
        show(A.n(digits=2))
```

```
[0.20 0.10 0.30 0.15]
[0.40 0.40 0.25 0.15]
[0.10 0.20 0.30 0.10]
[0.30 0.30 0.15 0.60]
```

```
In [3]: v0=vector([1000.,1000.,1300.,1700.])
        print("Distribution on Thursday")
        show(A^3*v0)
```

```
Distribution on Thursday
```

```
(843.500000000000, 1399.62500000000, 803.725000000000, 1953.15000000000)
```

```
In [5]: print("Distribution on after 200 days")
        show(A^200*v0)
```

```
Distribution on after 200 days
```

```
(842.523158359065, 1387.29598588443, 798.411998235553, 1971.76885752095)
```

```
In [4]: print("Distribution on after 2000 days")
        show(A^2000*v0)
```

```
Distribution on after 2000 days
```

```
(842.523158359063, 1387.29598588443, 798.411998235552, 1971.76885752095)
```

2.3 Encripting-Decripting Messages

```
In [26]: message = "Welcome to SageMath session 9!"
         print(message)
```

```
Welcome to SageMath session 9!
```

```
In [27]: message_L = list(message)
         print(message_L)
```

```
['W', 'e', 'l', 'c', 'o', 'm', 'e', ' ', 't', 'o', ' ', 'S', 'a', 'g', 'e', 'M', 'a', 't', 'h'
```

```
In [28]: EncodingMatrix = matrix([[3,2,5],[6,9,2],[9,11,10]])
         show(EncodingMatrix)
```

```
[ 3  2  5]
[ 6  9  2]
[ 9 11 10]
```

```
In [29]: DecodingMatrix = EncodingMatrix.inverse()
         show(DecodingMatrix)
```

```
[ 68/45    7/9 -41/45]
[-14/15   -1/3   8/15]
[  -1/3   -1/3    1/3]
```

```
In [30]: NumericMessage = [ord(s) for s in message_L]
         print(NumericMessage)
```

```
[87, 101, 108, 99, 111, 109, 101, 32, 116, 111, 32, 83, 97, 103, 101, 77, 97, 116, 104, 32, 115
```

```
In [31]: len(NumericMessage)
```

```
Out[31]: 30
```

```
In [32]: m,n = EncodingMatrix.dimensions()
         r = len(NumericMessage)%m
         if (r!=0):
             for i in range(m-r):
                 NumericMessage.append(ord(" "))
         print(NumericMessage)
```

```
[87, 101, 108, 99, 111, 109, 101, 32, 116, 111, 32, 83, 97, 103, 101, 77, 97, 116, 104, 32, 115
```

```
In [33]: matrix_message = matrix(len(NumericMessage)//m,m,NumericMessage).T
         show(matrix_message)
```

```
[ 87  99 101 111  97  77 104 101 105  32]
[101 111  32  32 103  97  32 115 111  57]
[108 109 116  83 101 116 115 115 110  33]
```

```
In [34]: encoded_matrix = EncodingMatrix*matrix_message
         show(encoded_matrix)
```

```
[1003 1064  947  812 1002 1005  951 1108 1087  375]
[1647 1811 1126 1120 1711 1567 1142 1871 1849  771]
[2974 3202 2421 2181 3016 2920 2438 3324 3266 1245]
```

```
In [35]: decoded_matrix = DecodingMatrix*encoded_matrix
         show(decoded_matrix)
```

```
[ 87  99 101 111  97  77 104 101 105  32]
[101 111  32  32 103  97  32 115 111  57]
[108 109 116  83 101 116 115 115 110  33]
```

```
In [36]: decoded_message=decoded_matrix.T.list()
         print(decoded_message)
```

```
[87, 101, 108, 99, 111, 109, 101, 32, 116, 111, 32, 83, 97, 103, 101, 77, 97, 116, 104, 32, 115
```

```
In [37]: print([chr(i) for i in decoded_message])
```

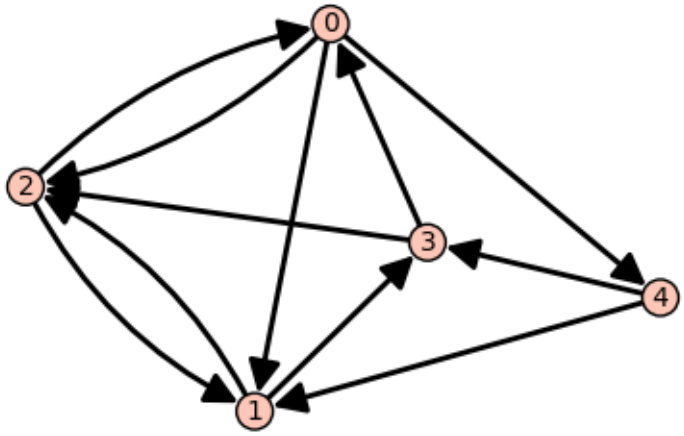
```
['W', 'e', 'l', 'c', 'o', 'm', 'e', ' ', 't', 'o', ' ', 'S', 'a', 'g', 'e', 'M', 'a', 't', 'h'
```

```
In [38]: print(''.join([chr(i) for i in decoded_message]))
```

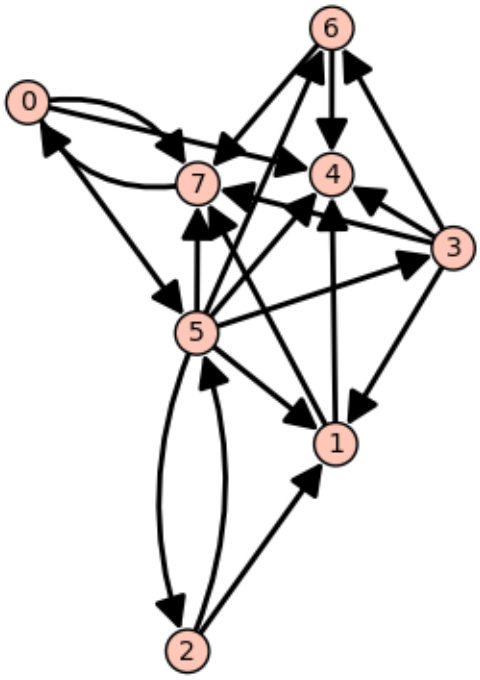
```
Welcome to SageMath session 9!
```

2.4 Google Page Rank

In [39]:
```
## Generating a directed graph (more on 'graphs+DOT+TAB')
G = DiGraph({0: [1,2,4], 1:[2,3], 2:[0,1],3:[0,2],4:[3,1]})
show(G,figsize=5)
```



In [7]:
```
## Random network
n = 8 # Number of webpages/nodes
N  = 20 # Total number of links
D = digraphs.RandomDirectedGNM(n,N)
D.show(figsize=6)
```



```
In [8]: ## Matrix Associated to the network. H=[H_{ij}] where H_{ij}=1 if there is link betwee
        H=D.adjacency_matrix()
        H=H.T
        print( "Ajacency matrix")
        show(H)
```

```
Ajacency matrix


[0 0 0 0 0 0 0 1]
[0 0 1 1 0 1 0 0]
[0 0 0 0 0 1 0 0]
[0 0 0 0 0 1 0 0]
[1 1 0 1 0 1 1 0]
[1 0 1 0 0 0 0 0]
[0 0 0 1 0 1 0 0]
[1 1 0 1 0 1 1 0]
```

```
In [9]: ## Probability matrix P=[p_{ij}] where p_{ij} is the probability of visiting i th webs
        print ("Probability matrix or transition matrix")
        P=column_matrix([H.columns()[i]/max(sum(H.columns()[i]),1) for i in range(n)])
        show(P)
```

```
Probability matrix or transition matrix


[  0   0   0   0   0   0   0   1]
[  0   0 1/2 1/4   0 1/6   0   0]
[  0   0   0   0   0 1/6   0   0]
[  0   0   0   0   0 1/6   0   0]
[1/3 1/2   0 1/4   0 1/6 1/2   0]
[1/3   0 1/2   0   0   0   0   0]
[  0   0   0 1/4   0 1/6   0   0]
[1/3 1/2   0 1/4   0 1/6 1/2   0]
```

```
In [10]: v=vector([0,0,0,0,0,0,0,1])
         P*v
```

```
Out[10]: (1, 0, 0, 0, 0, 0, 0, 0)
```

```
In [11]: S=1/n*matrix([[1]*n]*n)
         show(S)
```

```
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
[1/8 1/8 1/8 1/8 1/8 1/8 1/8 1/8]
```

```
In [12]: print("The Google Matrix of the associated network is given by")
         alpha=0.85
         G = alpha*P+(1-alpha)*S;
         show(G.n(digits=4))
```

```
The Google Matrix of the associated network is given by
```

```
[0.01875 0.01875 0.01875 0.01875 0.01875 0.01875 0.01875  0.8688]
[0.01875 0.01875  0.4437  0.2313 0.01875  0.1604 0.01875 0.01875]
[0.01875 0.01875 0.01875 0.01875 0.01875  0.1604 0.01875 0.01875]
[0.01875 0.01875 0.01875 0.01875 0.01875  0.1604 0.01875 0.01875]
[ 0.3021  0.4437 0.01875  0.2313 0.01875  0.1604  0.4437 0.01875]
[ 0.3021 0.01875  0.4437 0.01875 0.01875 0.01875 0.01875 0.01875]
[0.01875 0.01875 0.01875  0.2313 0.01875  0.1604 0.01875 0.01875]
[ 0.3021  0.4437 0.01875  0.2313 0.01875  0.1604  0.4437 0.01875]
```

```
In [13]: ## Initial  vector
         v=vector([0,0,0,1,0,0,0,0])
         k=10 # No of clicks
         pn = G^k*v.n(digits=6)
         print('After ' +str(k)+' iteration the probability vector is: \n');
         pn
```

```
After 10 iteration the probability vector is:
```

```
Out[13]: (0.0379248, 0.0127956, 0.00715300, 0.00715300, 0.0335378, 0.0206703, 0.00903387, 0.033
```

```
In [14]: k=1000
         pn = G^k*v.n(digits=6)
         print('After ' +str(k)+' iteration the probability vector is: \n');
         pn
```

```
After 1000 iteration the probability vector is:
```

```
Out[14]: (3.33140e-85, 1.11547e-85, 6.29017e-86, 6.29017e-86, 2.91921e-85, 1.79063e-85, 7.9116
```

```
In [15]: k=10000
         pn = G^k*v.n(digits=6)
         print('After ' +str(k)+' iteration the probability vector is: \n');
         pn
```

```
After 10000 iteration the probability vector is:
```

```
Out[15]: (2.76945e-840, 9.27310e-841, 5.22913e-841, 5.22913e-841, 2.42680e-840, 1.48858e-840, 6
```

```
In [16]: PN = flatten(pn)
         sorted_nodes = []
         for i in range(n):
             sorted_nodes.append([PN[i],i])
         sorted_nodes.sort(reverse=True)
         table(sorted_nodes)
```

```
Out[16]:   2.76945e-840   0
           2.42680e-840   7
           2.42680e-840   4
           1.48858e-840   5
           9.27310e-841   1
           6.57712e-841   6
           5.22913e-841   3
           5.22913e-841   2
```

2.5 Singual Value Decomposition (SVD)

Theorem (SVD) Let A be a real $m \times n$ matrix. Then there exist orthogonal matrices U and V such that

$$A = U \sum V^T$$

where U is $m \times m$ orthogonal matrix, V is $n \times n$ orthogonal matrix and $\sum$ is a $m \times n$ diagonal matrix whosediagonal entries are non negative and are arranged in a non increasing order. The number of non zero entries in $\sum$ is the rank of A.

Rank-1 approximation The singular value decomposition of matrix A can also be written as

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \cdots + \sigma_r u_r v_r^T.$$

```
In [17]: M = matrix(CDF, [[1,1],[2,3],[3,4]])
         show(M)
```

```
[1.0 1.0]
[2.0 3.0]
[3.0 4.0]
```

```
In [18]: U, S, V = M.SVD()
         show(U);show(S);show(V)
```

```
[-0.22120462435759758    0.785961310007992  -0.5773502691896275]
[ -0.5700601486798198  -0.5845494791322714  -0.5773502691896245]
[ -0.7912647730374174  0.20141183087572434   0.5773502691896253]


[6.3186120980141895                 0.0]
[                0.0 0.2741188698880916]
[                0.0                 0.0]


[ -0.591129694763722  0.8065765208388779]
[-0.8065765208388779  -0.591129694763722]
```

```
In [19]: U*S*V.T
```

```
Out[19]: [0.9999999999999993 1.0000000000000009]
         [1.9999999999999991 2.9999999999999996]
         [ 2.999999999999999 3.9999999999999996]
```

2.5.1 Application of SVD to image processing

```
In [20]: from matplotlib.pyplot import imread
         import pylab
         import numpy as np
         img=pylab.imread('../Sardar.png')
```

```
In [21]: matrix_plot(img)
```

```
Out[21]:
```

```
In [22]: img.shape
```

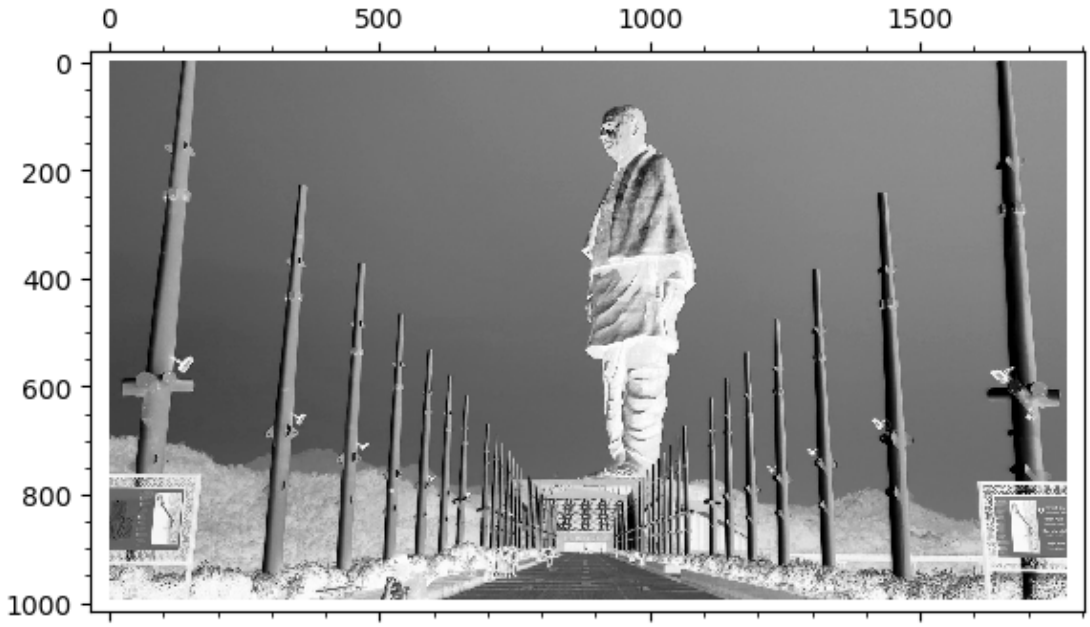
```
Out[22]: (995, 1770, 3)
```

Converting color image to gray scale image

$$0.299 \times R + 0.587 \times G + 0.114 \times B$$

```
In [23]: gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])
         gray_img = gray(img)
         matrix_plot(gray_img)
```

```
Out[23]:
```




```
In [24]: gray_img.shape
```

```
Out[24]: (995, 1770)
```

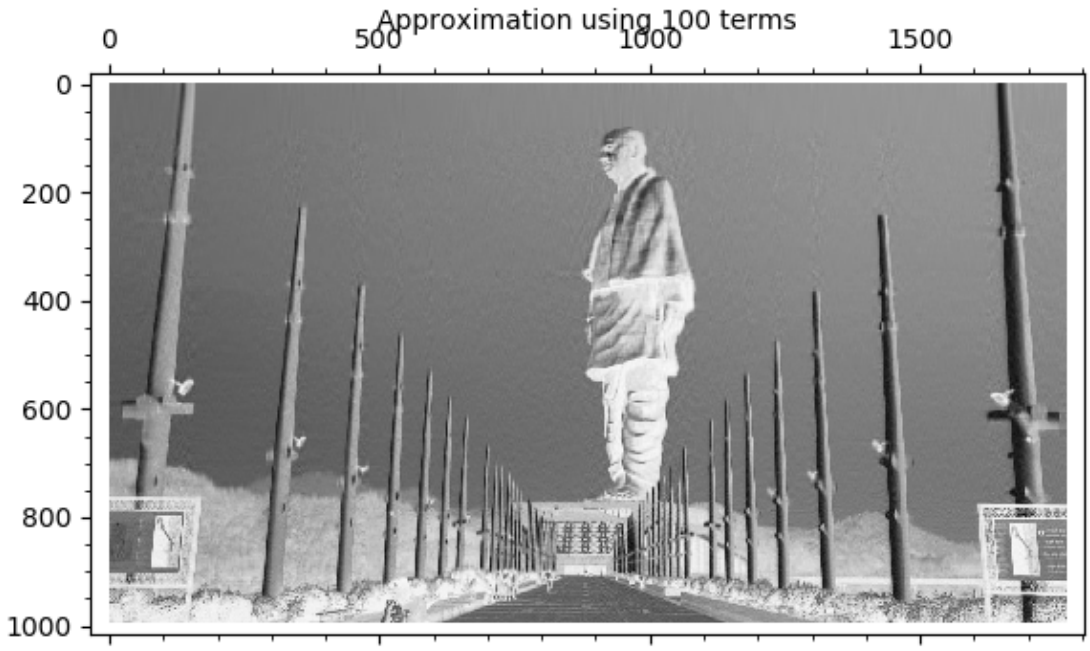
```
In [25]: U,S,V = matrix(gray_img).SVD()
```

```
In [26]: U.dimensions(),S.dimensions(),V.dimensions()
```

```
Out[26]: ((995, 995), (995, 1770), (1770, 1770))
```

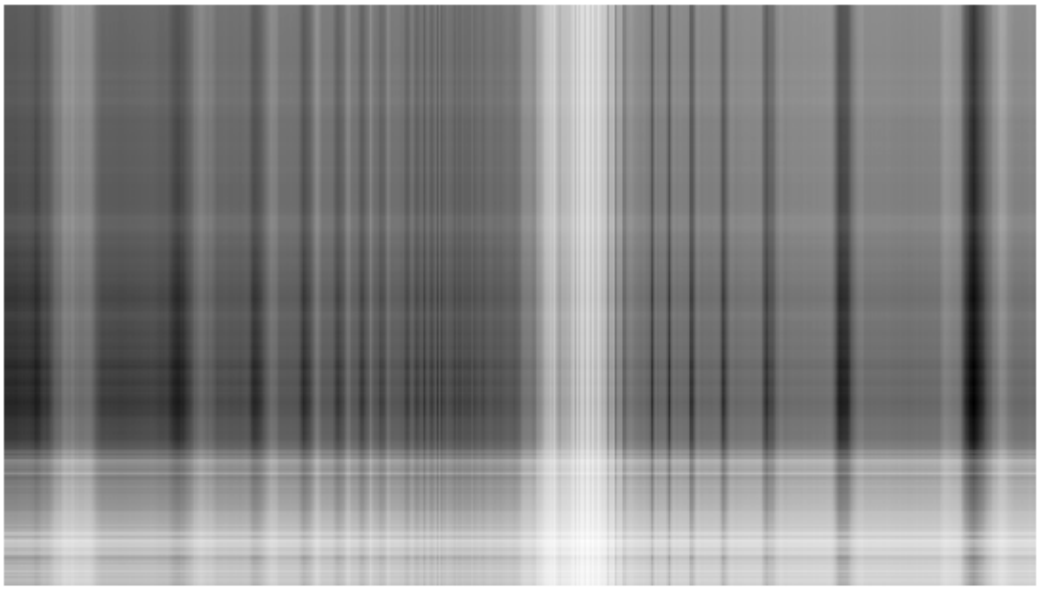
In [27]:
```
n=100
A_approx = U[:,:n]*S[:n,:n]*V.T[:n,:]
#print('Approximation using '+str(n)+ ' terms')
svd_img1=matrix_plot(A_approx,figsize=6,title='Approximation using '+str(n)+' terms')
svd_img1
```

Out[27]:



In [28]:
```
appx =[]
for i in range(1,100,10):
   A_approx = U[:,:i]*S[:i,:i]*V.T[:i,:]
   appx_img=matrix_plot(A_approx,title="Using "+str(i)+' Singular Vaues', frame=False)
   show(appx_img,figsize=6)
```

Using 1 Singular Vaues



Using 11 Singular Vaues

Using 21 Singular Vaues

Using 31 Singular Vaues

Using 41 Singular Vaues

Using 51 Singular Vaues

Using 61 Singular Vaues

Using 71 Singular Vaues

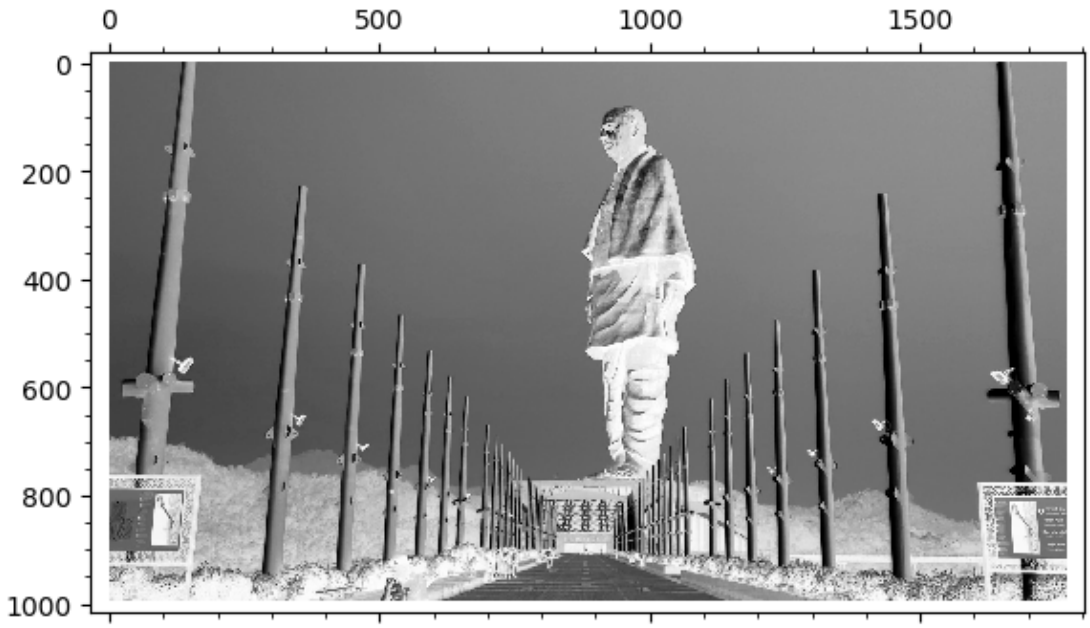
Using 81 Singular Vaues

Using 91 Singular Vaues

2.6 Denoising the image

In [29]:
```
# Remove sigma values below threshold (250)
s_cleaned = diagonal_matrix([si if si > 250 else 0 for si in S.diagonal()])
new_img = U*S*V.T
matrix_plot(new_img,figsize=6)
```

Out[29]:

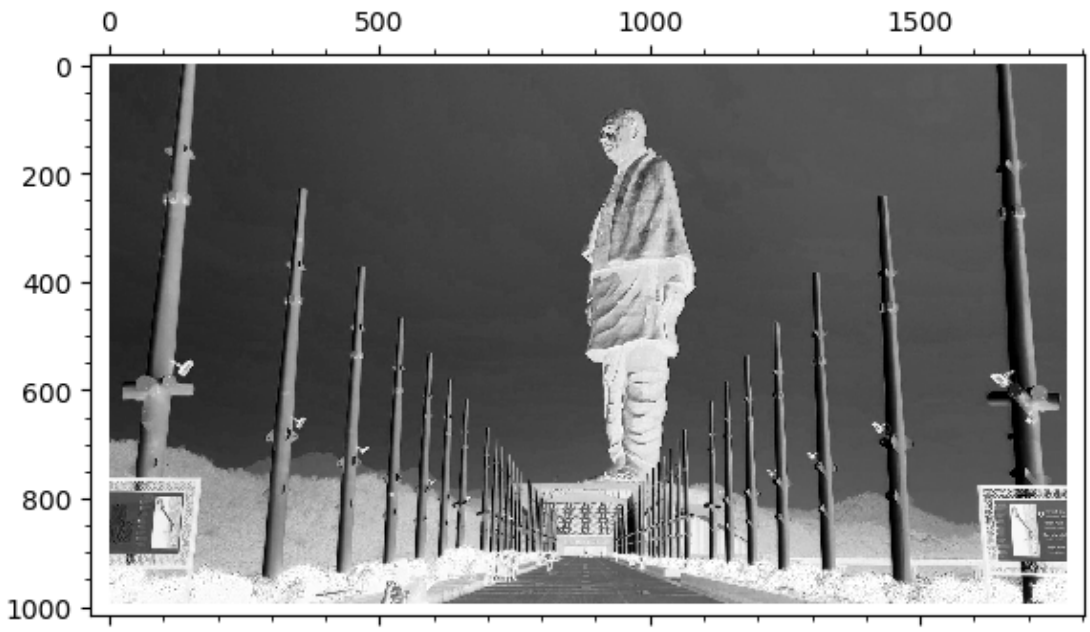


In [30]:
```
## SVD with color image plot
```

In [31]:
```
from matplotlib.pyplot import imread
import pylab
import numpy as np
img=pylab.imread('../Sardar.png')
matrix_plot(img)
```

Out[31]:


```
In [32]: R1 = img[:,:,0] # Red channel
         G1 = img[:,:,1] # Green Channel
         B1 = img[:,:,2] # Blue Channel

In [33]: R1.shape,G1.shape,B1.shape

Out[33]: ((995, 1770), (995, 1770), (995, 1770))

In [34]: matrix_plot(R1)
         matrix_plot(G1)
         matrix_plot(B1)

Out[34]:
```




```
In [35]: img_transposed = np.transpose(img, (2, 0, 1))
         img_transposed.shape

Out[35]: (3, 995, 1770)

In [36]: U, s, Vt = np.linalg.svd(img_transposed)

In [37]: U.shape, s.shape, Vt.shape

Out[37]: ((3, 995, 995), (3, 995), (3, 1770, 1770))

In [38]: Sigma = np.zeros(img_transposed.shape)
         for j in range(3):
             np.fill_diagonal(Sigma[j, :, :], s[j, :])

In [39]: Sigma.shape

Out[39]: (3, 995, 1770)

In [40]: reconstructed = U @ Sigma @ Vt

In [41]: pylab.imshow(np.transpose(reconstructed, (1, 2, 0)))

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]


Out[41]: <matplotlib.image.AxesImage object at 0x7fcda0259748>
```

In [42]:
```
k= 150
approx_img = U @ Sigma[..., :k] @ Vt[..., :k, :]
pylab.imshow(np.transpose(approx_img, (1, 2, 0)))
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

Out[42]: <matplotlib.image.AxesImage object at 0x7fcda6f4ab38>

```
In [43]: matrix_plot(np.transpose(approx_img, (1, 2, 0)))

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255]
```

Out[43]:

2.7 Linear Epodemic Model

The dynamics of infection and the spread of an epidemic can be modeled using a linear dynamical system. Let us look at a simple example.

Suppose a disease is introduced into a population. In each period (say, days) we count the fraction of the population in four different infection states:

- **Susceptible:** These individuals can acquire the disease the next day.
- **Infected:** These individuals have the disease.
- **Recovered (and immune):** These individuals had the disease and survived, and now have immunity.
- **Deceased:** These individuals had the disease, and unfortunately died from it.

We denote the fractions of each of these as a 4-vector

$$x^{(t)} = (x_1^{(t)}, x_2^{(t)}, x_3^{(t)}, x_4^{(t)}).$$

For example, if $x^{(t)} = (0.75, 0.10, 0.10, 0.05)$ means that in day t, 75% of the population is susceptible, 10% is infected, 10% has recovered and immune, and 5% has died from the disease.

There are many mathematical models that predict how the disease state fractions x^t evolve over time. One simple model can be expressed as a linear dynamical system.

Assumptions

We first determine $x_1^{(t+1)}$, the fraction of susceptible individuals in the next day. These include the susceptible individuals from today, who did not become infected, which is $0.95x_1^{(t)}$, plus the infected individuals today who recovered without immunity, which is $0.04x_2^{(t)}$. Thus we have

$$x_1^{(t+1)} = 0.95x_1^{(t)} + 0.04x_2^{(t)}$$

The fraction of infected individuals in the next day

$$x_2^{(t+1)} = 0.85x_2^{(t)} + 0.05x_1^{(t)}$$

where the first term counts those who are infected and remain infected, and the second term counts those who are susceptible and acquire the disease.

Similarly

$$x_3^{(t+1)} = x_3^{(t)} + 0.10x_2^{(t)}$$

$$x_4^{(t+1)} = x_4^{(t)} + 0.01x_2^{(t)}$$

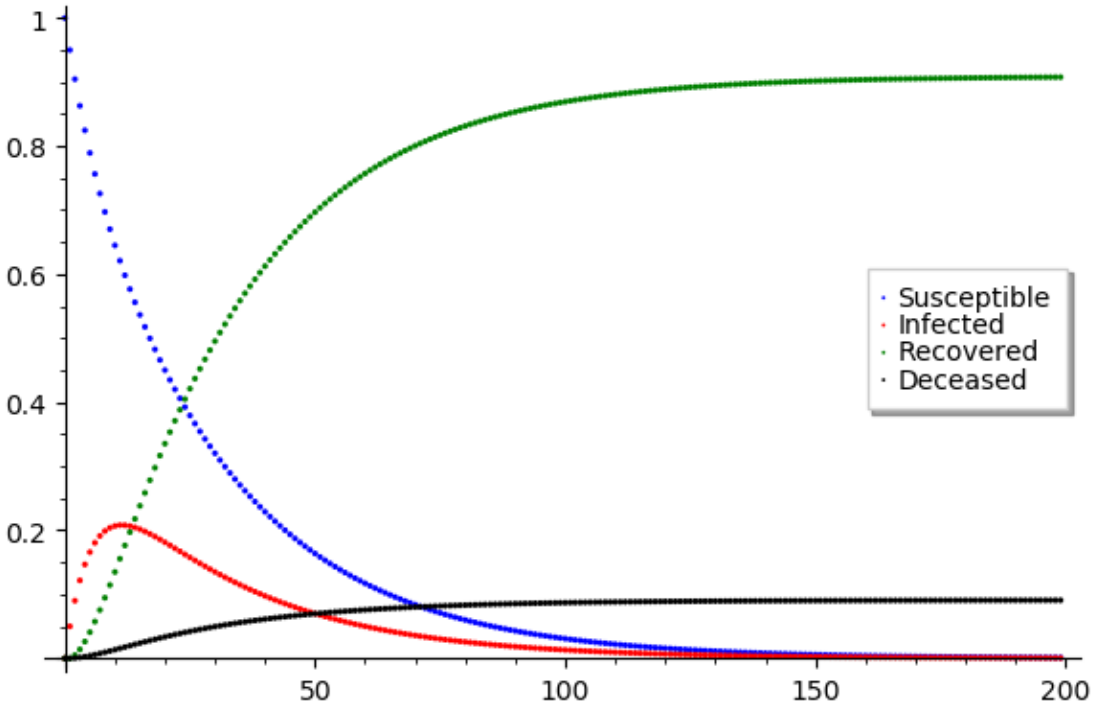
The above equations can be represented by

$$x^{(t+1)} = \begin{pmatrix} 0.95 & 0.04 & 0.00 & 0.00 \\ 0.05 & 0.85 & 0.00 & 0.00 \\ 0.00 & 0.10 & 1.00 & 0.00 \\ 0.00 & 0.01 & 0.00 & 1.00 \end{pmatrix} x^{(t)}.$$

```
In [82]: Epi_Mat = matrix(RR, [[0.95,0.04,0,0],[0.05,0.85,0,0],[0,0.10,1,0],[0,0.01,0,1]])

In [83]: show(Epi_Mat)

[ 0.950000000000000 0.0400000000000000  0.000000000000000  0.000000000000000]
[0.0500000000000000  0.850000000000000  0.000000000000000  0.000000000000000]
[ 0.000000000000000  0.100000000000000   1.00000000000000  0.000000000000000]
[ 0.000000000000000 0.0100000000000000  0.000000000000000   1.00000000000000]


In [85]: x0 = vector([1,0,0,0])

In [90]: Epi_Mat^365*x0

Out[90]: (4.31484264725142e-6, 1.84265779453771e-6, 0.909085311363235, 0.0909085311363235)

In [91]: N = 200 ## No. of days
         Susceptible= point2d([(n,(Epi_Mat^n*x0)[0]) for n in range(N)],color='blue',
                              legend_label='Susceptible',size=5)
         Infected = point2d([(n,(Epi_Mat^n*x0)[1]) for n in range(N)],color='red',
                              legend_label='Infected',size=5)
```

```
Recovered = point2d([(n,(Epi_Mat^n*x0)[2]) for n in range(N)],color='green',
                    legend_label='Recovered',size=5)
Deceased = point2d([(n,(Epi_Mat^n*x0)[3]) for n in range(N)],color='black',
                    legend_label='Deceased',size=5)


Susceptible+Infected+Recovered+Deceased
```

Out[91]:



2.8 Application of Affine Linear Transformations

Affine linear transformation:

Affine linear transformation T on an inner product space V is of the form $T(v) = Av + v_0$ where A is an orthogonal linear transformation on v and v_0 is a vector in V.

Result:

Given two set of three non collinear points $\{v_1, v_2, v_3\}$ and $\{w_1, w_2, w_3\}$ in the plane there exists a unique affine transformation T on the plane such that $T(v_i) = w_i$ for $i = 1, 2, 3$.

The above result is used in sevaral image processing concepts, one of them is image morphing of tranforming any image to another image by a sequence of affine transformations.

2.8.1 Sierpinski Triangle

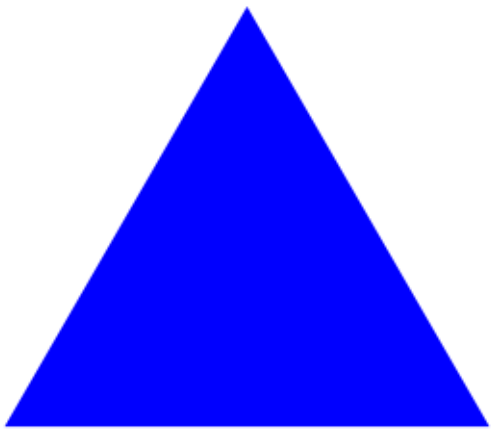
```
In [92]: def affine_transform(T1,T2):
             u,v,w=T1[0],T1[1],T1[2]
```

```
    u1,v1,w1=T2[0],T2[1],T2[2]
    var('a11,a12,a21,a22,a,b')
    M=matrix([[a11,a12],[a21,a22]])
    s=solve([(M*u)[0]+a==u1[0],(M*u)[1]+b==u1[1],(M*v)[0]+a==v1[0],
             (M*v)[1]+b==v1[1],(M*w)[0]+a==w1[0],(M*w)[1]+b==w1[1]],
            [a11,a12,a21,a22,a,b],solution_dict=True)
    MM=matrix([[s[0][a11],s[0][a12]],[s[0][a21],s[0][a22]]])
    vv=vector([s[0][a],s[0][b]])
    return MM,vv
```

In [93]:
```
def Sierpinsky_triangle(S0):
    T0=S0
    T1=[S0[0],(S0[0]+S0[1])/2,(S0[0]+S0[2])/2]
    M1,vv1=affine_transform(T0,T1)
    S1=[M1*S0[0]+vv1,M1*S0[1]+vv1,M1*S0[2]+vv1]
    T2=[(S0[0]+S0[1])/2,S0[1],(S0[1]+S0[2])/2]
    M2,vv2=affine_transform(T1,T2)
    S2=[M2*T1[0]+vv2,M2*T1[1]+vv2,M2*T1[2]+vv2]
    T3=[(S0[0]+S0[2])/2,(S0[1]+S0[2])/2,S0[2]]
    M3,vv3=affine_transform(T1,T3)
    S3=[M3*T1[0]+vv3,M3*T1[1]+vv3,M3*T1[2]+vv3]
    S=[S1,S2,S3]
    return S
```

In [94]:
```
u0,v0,w0=vector([0,0]),vector([1,0]),vector([1/2,sqrt(3)/2])
K=[u0,v0,w0]
polygon(K,axes=False,figsize=4)
```

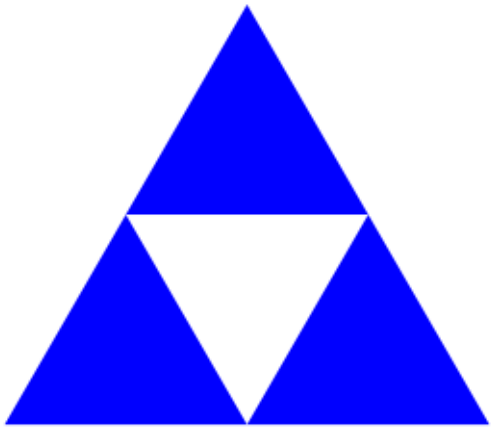
Out[94]:



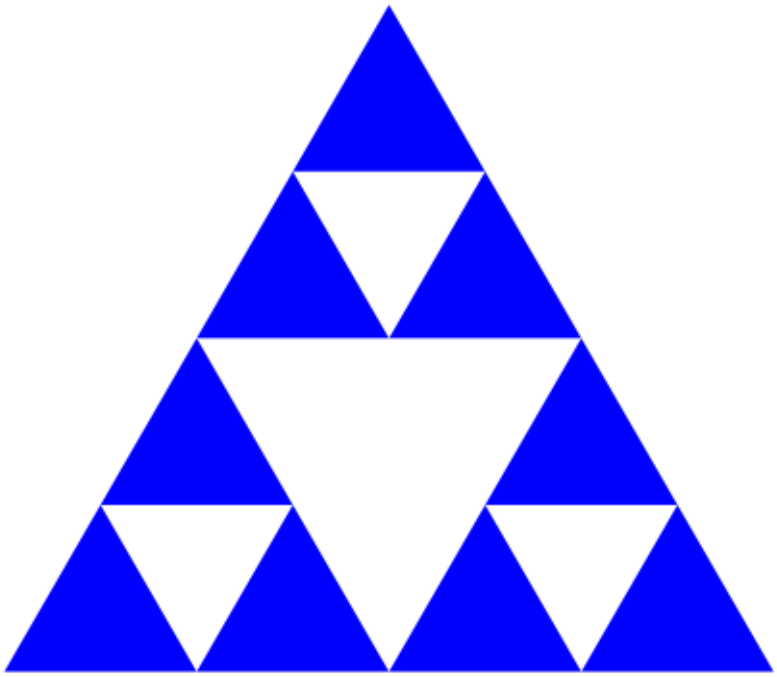
In [95]:
```
K=Sierpinsky_triangle(K)
show(K)
```

```
[[(0, 0), (1/2, 0), (1/4, 1/4*sqrt(3))],
 [(1/2, 0), (1, 0), (3/4, 1/4*sqrt(3))],
 [(1/4, 1/4*sqrt(3)), (3/4, 1/4*sqrt(3)), (1/2, 1/2*sqrt(3))]]
```

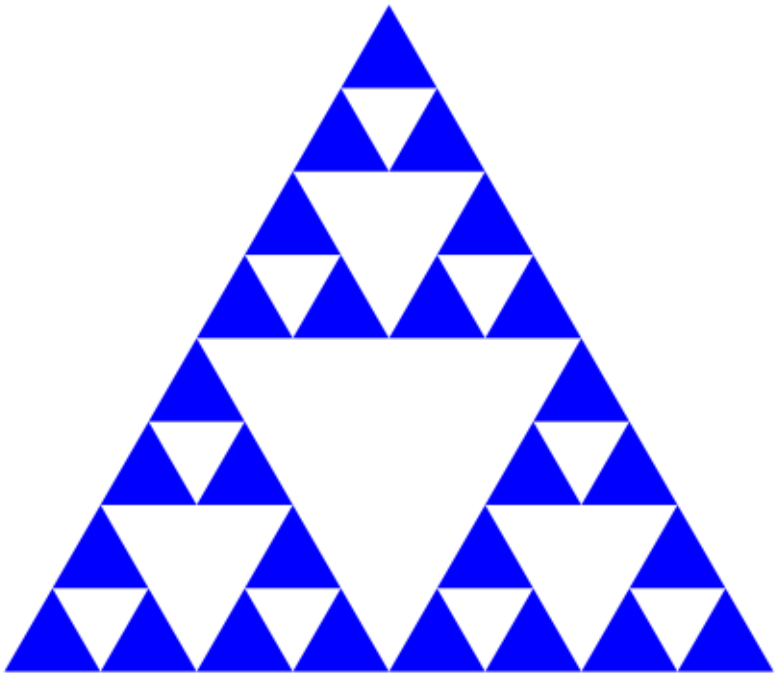
In [96]:
```
P=sum(map(polygon,K))
show(P,figsize=4,axes=False)
```



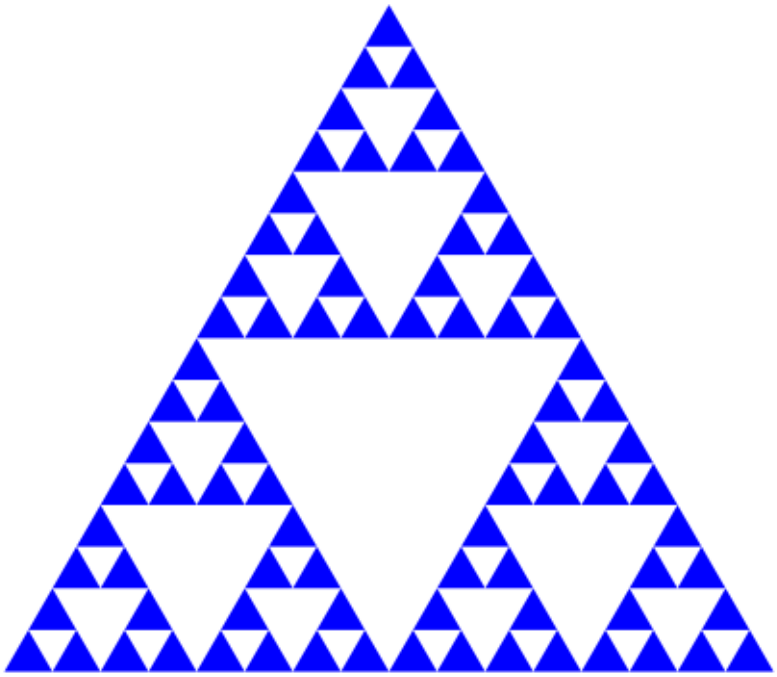
In [97]:
```
K = [Sierpinsky_triangle(S) for S in K]
K = flatten(K, max_level=1)
P=sum([polygon2d(S,axes=False) for S in K])
show(P,axes=False)
```



In [98]:
```
K = [Sierpinsky_triangle(S) for S in K]
K = flatten(K, max_level=1)
P=sum([polygon2d(S,axes=False) for S in K])
show(P,axes=False)
```



```
In [99]: K = [Sierpinsky_triangle(S) for S in K]
         K = flatten(K, max_level=1)
         P=sum([polygon2d(S,axes=False) for S in K])
         show(P,axes=False)
```



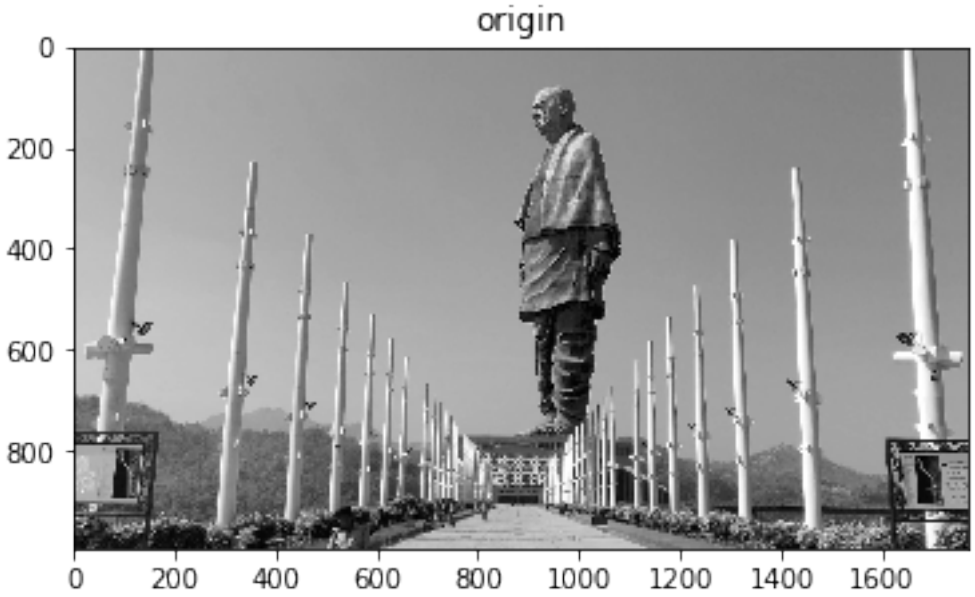
More on affine tranformation on images, you may refer to OpenCV-Python library

```
In [49]: import cv2
         import matplotlib.pyplot as plt
         import numpy as np

In [50]: img = cv2.imread('../Sardar.jpg')
         plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

Out[50]: <matplotlib.image.AxesImage object at 0x7fcda5c428d0>
```

In [ ]:

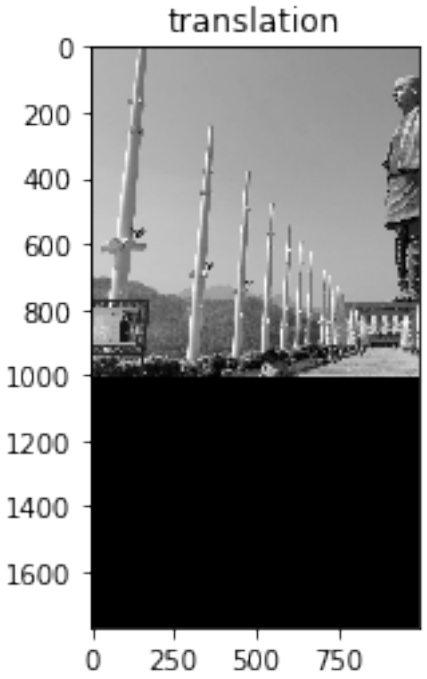
In [51]:
```
image = cv2.imread(r'../Sardar.png', cv2.IMREAD_GRAYSCALE)
plt.imshow(image, 'gray'), plt.title('origin')
```

Out[51]: (<matplotlib.image.AxesImage object at 0x7fcda6f88390>, Text(0.5,1,'origin'))



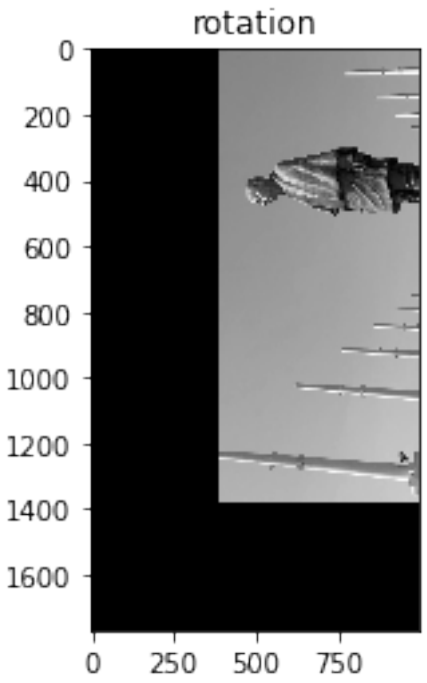
In [52]:
```
# translation
translation_M = np.array([[1, 0, 10], [0, 1, 10]], dtype=np.float32)
translation_image = cv2.warpAffine(image, translation_M, image.shape)
plt.imshow(translation_image, 'gray'), plt.title('translation')
```

Out[52]:
```
(<matplotlib.image.AxesImage object at 0x7fcda011cb70>,
 Text(0.5,1,'translation'))
```



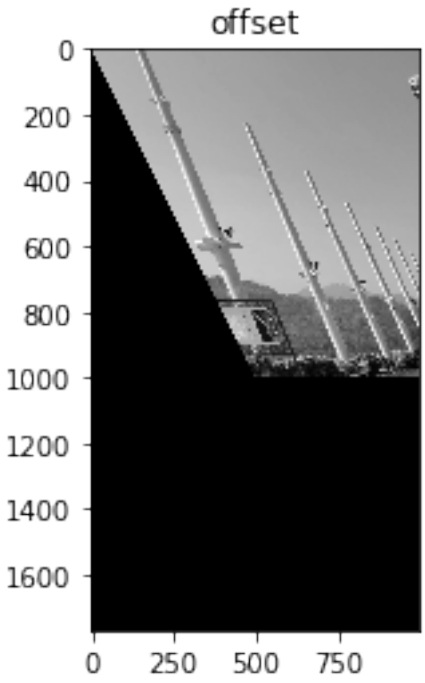
In [53]:
```
# rotation
rows, cols = image.shape
rotation_M = cv2.getRotationMatrix2D((cols/2, rows/2), 90, 1)
rotation_image = cv2.warpAffine(image, rotation_M, image.shape)
plt.imshow(rotation_image, 'gray'), plt.title('rotation')
```

Out[53]:
```
(<matplotlib.image.AxesImage object at 0x7fcda012b128>, Text(0.5,1,'rotation'))
```



In [54]:
```
# offset
offset_M = np.array([[1, 0.5, 0], [0, 1, 0]], dtype=np.float32)
offset_image = cv2.warpAffine(image, offset_M, image.shape)

plt.imshow(offset_image, 'gray'), plt.title('offset')
```

Out[54]:
```
(<matplotlib.image.AxesImage object at 0x7fcda0aeed68>, Text(0.5,1,'offset'))
```



In [ ]: