# SageMatrixFactorizationDemo260831

September 4, 2026

## 1 Matrix generators

[1]:
```
def sgen(i,c,t):
    r"""
    returns the generator s_i(c)
    """
    M=MatrixSpace(QQ,t,t)
    I=identity_matrix(QQ, t)
    E=M.basis()
    return( I-E[i-1,i-1]-E[i,i]+E[i-1,i]+E[i,i-1]+c*E[i-1,i-1] )
```

[2]:
```
def sinv(i,c,t):
    r"""
    returns  s_i(c)^{-1}
    """
    M=MatrixSpace(QQ,t,t)
    E=M.basis()
    I=identity_matrix(QQ, t)
    return( I-E[i-1,i-1]-E[i,i]+E[i-1,i]+E[i,i-1]-c*E[i,i] )
```

[3]:
```
def hgen(i,c,t):
    r"""
    returns the generator h_i(c)
    """
    M=MatrixSpace(QQ,t,t)
    E=M.basis()
    I=identity_matrix(QQ, t)
    return( I+(c-1)*E[i-1,i-1] )
```

[4]:
```
def xgen(i,j,c,t):
    r"""
    returns the generator x_{ij}i(c)
    """
    M=MatrixSpace(QQ,t,t)
    E=M.basis()
    I=identity_matrix(QQ, t)
    if i < j:
```

```
        return( I + c*E[i-1,j-1] )
    else:
        print("must have i not equal to j in x(i,j,c) so returning 1")
        return(I)
```

[5]:
```
sgen(3,2,4)
```

[5]:
```
[1 0 0 0]
[0 1 0 0]
[0 0 2 1]
[0 0 1 0]
```

[6]:
```
sinv(2,5,4)
```

[6]:
```
[ 1  0  0  0]
[ 0  0  1  0]
[ 0  1 -5  0]
[ 0  0  0  1]
```

[7]:
```
hgen(2,5,4)
```

[7]:
```
[1 0 0 0]
[0 5 0 0]
[0 0 1 0]
[0 0 0 1]
```

[8]:
```
xgen(2,3,7,4)
```

[8]:
```
[1 0 0 0]
[0 1 7 0]
[0 0 1 0]
[0 0 0 1]
```

## 2 Some matrices for examples

[9]:
```
M46=MatrixSpace(QQ,4,6)
```

[10]:
```
M55=MatrixSpace(QQ,5,5)
```

[11]:
```
A=M46([-3,-36,-39,-42,-46,-84, -9,-108,-233/2,-114,-247/2,-167,␣
 ↪3,36,39,42,45,48, 0,0,1/4,6,25/4,13/2])
```

[12]:
```
A
```

[12]:
```
[    -3    -36    -39    -42    -46    -84]
[    -9   -108 -233/2   -114 -247/2   -167]
[     3     36     39     42     45     48]
```

```
[     0      0    1/4      6   25/4   13/2]
```

[13]:
```
B0=M46([3,36,39,42,45,48, 0,0,1/4,6,25/4,13/2, 0,0,0,0,-1,-36, 0,0,0,0,0,0])
```

[14]:
```
B0
```

[14]:
```
[   3   36   39   42   45   48]
[   0    0  1/4    6 25/4 13/2]
[   0    0    0    0   -1  -36]
[   0    0    0    0    0    0]
```

[15]:
```
C0=M46([1,12,13,14,15,16, 0,0,1,24,25,26, 0,0,0,0,1,36, 0,0,0,0,0,0])
```

[16]:
```
C0
```

[16]:
```
[ 1 12 13 14 15 16]
[ 0  0  1 24 25 26]
[ 0  0  0  0  1 36]
[ 0  0  0  0  0  0]
```

[17]:
```
V=M55([4,7,9,10,1, 3,6,8,1,0, 2,5,1,0,0, 1,1,0,0,0, 1,0,0,0,0])
```

[18]:
```
V
```

[18]:
```
[ 4  7  9 10  1]
[ 3  6  8  1  0]
[ 2  5  1  0  0]
[ 1  1  0  0  0]
[ 1  0  0  0  0]
```

[19]:
```
w0=M55([0,0,0,0,1, 0,0,0,1,0, 0,0,1,0,0, 0,1,0,0,0, 1,0,0,0,0])
```

[20]:
```
w0
```

[20]:
```
[0 0 0 0 1]
[0 0 0 1 0]
[0 0 1 0 0]
[0 1 0 0 0]
[1 0 0 0 0]
```

[21]:
```
U=M55([1,10,9,7,4, 0,1,8,6,3, 0,0,1,5,2, 0,0,0,1,1, 0,0,0,0,1])
```

[22]:
```
U
```

[22]:
```
[ 1 10  9  7  4]
[ 0  1  8  6  3]
[ 0  0  1  5  2]
[ 0  0  0  1  1]
```

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

## 3 First nonzero sequence, rank and status checks

[23]:
```
def fnz(A):
    r"""
    Constructs the first non-zero sequence of A, i.e.
    the sequence recording the position of the first nonzero
    entry in each row of A
    """
    f = [-1]
    t = A.nrows()
    s = A.ncols()
    for i in range(t):
        nonzero_list = [j for j in range(s) if A[i,j] != 0]
        if len([j for j in range(s) if A[i,j] != 0])>0:
            fi = min([j for j in range(s) if A[i,j] != 0])
        else:
            fi = None
        f.append(fi)
    return(f)
```

[24]:
```
def finccheck(f):
    r"""
    Checks if the sequence f is strictly increasing
    """
    for i in range(1,len(f)):
        if f[i-1] == None and f[i] != None:
            return(False)
        if f[i-1] != None and f[i] != None and f[i] <= f[i-1]:
            return(False)
    else:
        return(True)
```

[25]:
```
def ifionecheck(A):
    r"""
    Checks that A is in echelon form and that the first nonzero entry in each␣
 ↪onzero row is 1
    """
    f = fnz(A)
    if finccheck(f) == True:
        r = erank(A)
        for i in range(1, r+1):
            if A[i-1, f[i]] != 1:
                return(False)
            else:
```

```
                return(True)
```

[26]:
```
def xredcheck(A):
    r"""
    This checks that A is in echelon form and that the first nonzero entry in each nonzero row is 1 and that the entries
    above the first nonzero in each row are all 0
    """
    f = fnz(A)
    if finccheck(f) == False:
        print("the matrix is not in echelon form")
        return(False)
    elif ifionecheck(A) == False:
        print("the matrix is in echelon form but the pivots are not all 1")
        return(False)
    else:
        r = erank(A)
        #print(f"r is {r} and f is {f}")
        for i in range(1,r):
            for j in range(i):
                #print(f"i is {i} and j is {j} and f[i+1] is {f[i+1]} and the [j,f[i+1]] entry is {A[j,f[i+1]]}")
                if A[j,f[i+1]] != 0:
                    return(False)
    return(True)
```

[27]:
```
def erank(A):
    r"""
    This returns the rank of a matrix in echelon form
    """
    f = fnz(A)
    if finccheck(f) == True:
        r = max([i for i in range(len(f)) if f[i] != None])
        return(r)
    else:
        print("A is not in echelon form, first apply step one")
```

[28]:
```
fnz(A)
```

[28]:
```
[-1, 0, 0, 0, 2]
```

[29]:
```
finccheck(fnz(A))
```

[29]:
```
False
```

[30]:
```
finccheck([-1,3,6,9,None,None])
```

[30]: True

[31]:
```
finccheck([-1,3,6,9,9,12])
```

[31]: False

[32]:
```
ifionecheck(B0)
```

[32]: False

[33]:
```
ifionecheck(C0)
```

[33]: True

[34]:
```
erank(A)
```

```
A is not in echelon form, first apply step one
```

[35]:
```
erank(B0)
```

[35]: 3

## 4 Factoring steps

[36]:
```
def stepone(A, slist=[]):
    r"""
    Iterates step one to produce an echelon matrix and an slist
    The slist should default to [] if not specified in the call
    """
    t=A.nrows()
    s=A.ncols()
    f = fnz(A)
    #print(f"the increasing sequence is{f}")
    if finccheck(f):
        #print("the output matrix is now in echelon form, execute steptwo")
        return(A,slist)
    else:
        m = min([f[k] for k in range(1, len(f)) if f[k]!= None and␣
↪(f[k-1]==None or f[k-1]>=f[k])])
        #print(f"the min column with a pivot is {m}")
        i= max([j for j in range(1, len(f)) if f[j]==m])
        #print(f"the max row with a pivot in column {m} is {i}")
        c = A[i-2,f[i]] / A[i-1,f[i]]
        B = sinv(i-1,c,t) * A
        slist.append([i-1,c])
        #print(f"B is {B} and slist is {slist}")
        return(stepone(B, slist))
```

[37]:
```
def steptwo(B):
    r"""
    This takes the matrix B and makes the pivots all 1.
    The output is the new matrix and the hlist
    """
    t=B.nrows()
    s=B.ncols()
    f = fnz(B)
    if finccheck(f):
        hlist=[]
        for i in range(1,len(f)):
            if f[i] == None:
                B=B
                hlist.append(1)
            else:
                hlist.append(B[i-1,f[i]])
                B = hgen(i,1/B[i-1,f[i]],t) * B
        return(B,hlist)
    else:
        print("A is not in echelon form, execute step one")
```

[38]:
```
def stepthree(C):
    t=C.nrows()
    s=C.ncols()
    f = fnz(C)
    if finccheck(f):
        if ifionecheck(C):
            #print(f)
            r = max([k for k in range(1,len(f)) if f[k]!=None])
            xlist = []
            for i in range(r,1,-1):
                #print(f"r and i and f[i] are {r} and {i} and {f[i]}")
                for j in range(i-1,0,-1):
                    #print(f"j and f[i] {j} and {f[i]}")
                    #print(f"calling xgen({j},{i},{C[j-1,f[i]]},{t})")
                    if C[j-1,f[i]] != 0:
                        xlist.append([j,i,C[j-1,f[i]]])
                    C = xgen(j,i,-C[j-1,f[i]],t) * C
            return(C,xlist)
        else:
            print("A is in echelon form but pivots are not all 1, execute␣
↪steptwo")
    else:
        print("A is not in echelon form, execute stepone")
```

[39]:
```
def stepfour(R):
    t=R.nrows()
    s=R.ncols()
    f = fnz(R)
    if finccheck(f):
        if ifionecheck(R):
            if xredcheck(R):
                #print(f)
                r = erank(R)
                rxlist = []
                for i in range(1,r+1):
                    for j in range(f[i]+2,s+1):
                        #print(f"rmult by xgen({f[i]+1},{j},{-R[i-1,j-1]},{s})")
                        if R[i-1,j-1] != 0:
                            #This next line was getting the list in the wrong␣
↪order
                            #rxlist.append([f[i]+1,j,R[i-1,j-1]])
                            #Lets prepend instead
                            rxlist.insert(0,[f[i]+1,j,R[i-1,j-1]])
                            R = R * xgen(f[i]+1,j,-R[i-1,j-1],s)
                            #print(f"new R is {R}")
                return(R,rxlist)
            else:
```

```
                print("A is in echelon form with pivots all 1 but is not␣
↪reduced, execute stepthree")
        else:
            print("A is in echelon form but pivots are not all 1, execute␣
↪steptwo")
    else:
        print("A is not in echelon form, execute stepone")
```

[40]:
```
def stepfive(L):
    t = L.nrows()
    s = L.ncols()
    f = fnz(L)
    if finccheck(f):
        if ifionecheck(L):
            if xredcheck(L):
                r = erank(L)
                #print(f"r is {r}")
                rslist = []
                for i in range(1,r+1):
                    #print(f"i-1 is {i-1} and f[i] is {f[i]} and f[i]>i-1 is␣
↪{f[i]>i-1}")
                    if f[i]>i-1:
                        for j in range(f[i]+1,i,-1):
                            #print(f"j is {j}")
                            L = L * sgen(j-1,0,s)
                            #print(f"the new L is {L}")
                            #This next line was getting the list in the wrong␣
↪order
                            #rslist.append([j-1,0])
                            #Lets prepend instead
                            rslist.insert(0,[j-1,0])
#                     else:
#                         print("Go on please")
                return(L,rslist)
            else:
                print("A is in echelon form with pivots all 1 but is not␣
↪reduced, execute stepthree")
        else:
            print("A is in echelon form but pivots are not all 1, execute␣
↪steptwo")
    else:
        print("A is not in echelon form, execute stepone")
```

[49]:
```
def computePQfact(A):
    r"""
    Combine steponeiter, steptwo and stepthree to produce
    the factorization A = P1_rQ
```

```
    """
    t = A.nrows()
    s = A.ncols()
    B=stepone(A,[])
    C=steptwo(B[(0)])
    D=stepthree(C[0])
    R=stepfour(D[0])
    F=stepfive(R[0])
    slist = B[1]
    slen = len(slist)
    S =identity_matrix(QQ, t)
    for i in range(slen):
        sfactor = slist[i]
        S = S*sgen(sfactor[0],sfactor[1],t)
    hlist = C[1]
    hlen = len(hlist)
    H =identity_matrix(QQ, t)
    for i in range(hlen):
        hfactor = hlist[i]
        H = H*hgen(i+1,hfactor,t)
    xlist = D[1]
    xlen = len(xlist)
    X =identity_matrix(QQ, t)
    for i in range(xlen):
        xfactor = xlist[i]
        X = X*xgen(xfactor[0],xfactor[1],xfactor[2],t)
    P = S*H*X
    rxlist = R[1]
    #print(f"rxlist is {rxlist}")
    rxlen = len(rxlist)
    RX =identity_matrix(QQ, s)
    for i in range(rxlen):
        rxfactor = rxlist[i]
        #print(f"i is {i} and rxfactor is {rxfactor}")
        RX = xgen(rxfactor[0],rxfactor[1],rxfactor[2],s)*RX
    rslist = F[1]
    rslen = len(rslist)
    RS =identity_matrix(QQ, s)
    for i in range(rslen):
        rsfactor = rslist[i]
        RS = RS*sgen(rsfactor[0],rsfactor[1],s)
    Q=RS*RX
    #checkone = S*B[0]
    #print("checkone: S*B[0] is")
    #print(f"{checkone}")
    #checktwo = H*C[0]
    #print("checktwo: H*C[0] is")
```

```
    #print(f"{checktwo}")
    #checkthree = X*D[0]
    #print("checkthree: X*D[0] is")
    #print(f"{checkthree}")
    #checkfour = R[0]*RX
    #print(f"checkfour: R[0]*RX is {checkfour}")
    #P=S*H*X
    #checkfive = P*D[0]
    #print("checkfive: P*D[0] is")
    #print(f"{checkfive}")
    #checksix = F[0]*RS
    #print("checksix: F[0]*RS is")
    #print(f"{checksix}")
    #checkseven = F[0]*Q
    #print("checkseven: F[0]*Q is")
    #print(f"{checkseven}")
    print(f"Computing the factorization of A=")
    print(f"{A}")
    print(f"The s(i,c,{t}) factors are {slist}")
    print(f"The h(i,c,{t}) factors are {hlist}")
    print(f"The x(i,j,c,{t}) factors are {xlist}")
    print(f"The right s(i,0,{s}) factors are {rslist}")
    print(f"The right x(i,j,c.{s}) factors are {rxlist}")
    print(f"The left orbit representative in reduced row echelon form is R=")
    print(f"{D[0]}")
    print(f"The rank of A is r={erank(D[0])}")
    checkeight = P*F[0]*Q
    print("check: P*F[0]*Q is")
    print(f"{checkeight}")
    print("A,P,1_r and Q are")
    return(A,P,F[0],Q)
```

[50]:
```
computePQfact(A)
```

```
Computing the factorization of A=
[    -3    -36    -39    -42    -46    -84]
[    -9   -108 -233/2   -114 -247/2   -167]
[     3     36     39     42     45     48]
[     0      0    1/4      6   25/4   13/2]
The s(i,c,4) factors are [[2, -3], [1, -1], [3, 2], [2, 0], [3, 1]]
The h(i,c,4) factors are [3, 1/4, -1, 1]
The x(i,j,c,4) factors are [[2, 3, 25], [1, 3, 15], [1, 2, 13]]
The right s(i,0,6) factors are [[3, 0], [4, 0], [2, 0]]
The right x(i,j,c.6) factors are [[5, 6, 36], [3, 6, -874], [3, 4, 24], [1, 6,
10838], [1, 4, -298], [1, 2, 12]]
The left orbit representative in reduced row echelon form is R=
[    1    12     0  -298     0 10838]
[    0     0     1    24     0  -874]
```

```
[    0     0     0     0     1    36]
[    0     0     0     0     0     0]
The rank of A is r=3
check: P*F[0]*Q is
[    -3    -36    -39    -42    -46    -84]
[    -9   -108 -233/2   -114 -247/2   -167]
[     3     36     39     42     45     48]
[     0      0    1/4      6   25/4   13/2]
A,P,1_r and Q are
```

[50]:
```
(
[    -3    -36    -39    -42    -46    -84]
[    -9   -108 -233/2   -114 -247/2   -167]
[     3     36     39     42     45     48]
[     0      0    1/4      6   25/4   13/2],

[    -3    -39    -46      1]  [1 0 0 0 0 0]
[    -9 -233/2 -247/2      0]  [0 1 0 0 0 0]
[     3     39     45      0]  [0 0 1 0 0 0]
[     0    1/4   25/4      0], [0 0 0 0 0 0],

[    1    12     0  -298     0 10838]
[    0     0     1    24     0  -874]
[    0     0     0     0     1    36]
[    0     1     0     0     0     0]
[    0     0     0     1     0     0]
[    0     0     0     0     0     1]
)
```

[ ]: