A sage “interact” by Jason Grout (lead developer of the Sage Cell). Find it publicly at http://interact.sagemath.org/node/60. Some notes:

- Slider controls the green unit vector in far right display.
- Red and blue vectors begin far right as columns of $V$.
- Track the transformation of the green vector (and the others) by the matrix $A$, but in the three steps given by the matrices of the SVD (read displays from right to left, as composition of linear transformations).
- Try other matrices, perhaps some special cases, like rank 1, or diagonal.
- What would the demo look like for a $3 \times 3$ matrix? A $3 \times 4$ matrix?

```python
@interact
def _(A=matrix(RDF,2,2,[3,2,-4,3]),
     angle=slider(0,2*pi,default=3*pi/4):
    opts = {'figsize': 3}
    vn = vector(RDF, [cos(angle), sin(angle)])
    e1,e2=identity_matrix(2).columns()
    U,S,V = A.SVD()
    v1,v2 = V.columns()
    s1,s2 = list(S.diagonal())
    u1,u2 = U.columns()
    p1=circle((0,0),1,**opts)
    p1+=plot(vn,color='green')
    p1+=plot(v1,color='red')+plot(v2,color='blue')
    p2 = circle((0,0),1, **opts)
    p2+=plot(V.H*vn,color='green')
    p2+=plot(V.H*v1,color='red')
    p2+=plot(V.H*v2,color='blue')
    p3 = ellipse((0,0), s1,s2, **opts)
    p3 += plot(S*V.H*v1, color='red')
    p3 += plot(S*V.H*v2, color='blue')
    p3+= plot(S*V.H*vn,color='green')

    # we multiply by the sign of the y-coordinate
    # because arccos has a range of 0 to pi
    # we need to handle rotations that go from 0 to 2pi
    rotation = arccos(u1*vector([1,0])/u1.norm())*sign(u1[1])
```

2 The 3 × 3 Case

A routine to apply a matrix to the coordinate axes, and apply also it to a unit sphere (whose image is an ellipsoid). The output will require letting Java run to view properly.

```
def matrix_action(A, dim):
    """Input nonsingular 3x3 matrix A, and maximum bounding dimension
    Plots images of coordinate axes, and image of unit sphere"
    var('x, y, z')
    vect = vector([x, y, z])
    f = vect*(A.transpose())^-1*A^-1*vect
    xpp = point3d(A*vector([1, 0, 0])).list()
    xp = point3d(xpp, color='red', size=10)
    linex = line3d([(0, 0, 0), xpp], color='red', thickness=3)
    ypp = point3d(A*vector([0, 1, 0])).list()
    yp = point3d(ypp, color='green', size=10)
    liney = line3d([(0, 0, 0), ypp], color='green', thickness=3)
    zpp = point3d(A*vector([0, 0, 1])).list()
    zp = point3d(zpp, color='black', size=10)
    linez = line3d([(0, 0, 0), zpp], color='black', thickness=3)
    return implicit_plot3d(f, (x, -dim, dim), (y, -dim, dim),
                           (z, -dim, dim), contour=1, aspect_ratio=[1, 1, 1],
                           opacity=0.6)+xp+yp+zp+linex+liney+linez
```

A 3 × 3 matrix to experiment with. Along with its SVD decomposition.

```
A = matrix(RDF, [[2, 3, 1], [-1, 2, 1], [0, 2, 3]])
show(A)
```

```
U, S, V = A.SVD()
show(U)
show(S)
show(V)
```

Default. The identity matrix.

```
show(matrix_action(identity_matrix(3), 1.2))
```
Inverse of unitary matrix $V$ first, moving the columns of $V$ onto the coordinate axes.

\[
\text{show(matrix\_action(V.conjugate\_transpose(), 1.2))}
\]

Then stretching along the coordinate axes. Size is influenced by maximum singular value.

\[
\text{show(matrix\_action(S*V.conjugate\_transpose(), 5.1))}
\]

Now move coordinate axes to columns of $U$.

\[
\text{show(matrix\_action(U*S*V.conjugate\_transpose(), 5.1))}
\]

This last result should be the same as just applying $A$ directly.

\[
\text{show(matrix\_action(A, 5.1))}
\]