In [1]:
```
%matplotlib notebook
%pylab
```

```
Using matplotlib backend: nbAgg
Populating the interactive namespace from numpy and matplotlib
```

Spectral Theory

Let's take a brief review of what we know about solving linear systems

$$A\mathbf{x} = \mathbf{b}$$

where $A \in \mathcal{R}^{n \times n}$ and $\mathbf{b} \in \mathcal{R}^n$.

When can we solve this?

- When we have a solution? (:-))
- When $\mathbf{b}$ is in the column space of A.
-

Existence of a solution

Write the matrix vector multiply as a linear combination of columns:

$$A\mathbf{x} = \sum_{i=1}^{n} x_i \mathbf{a}_i, \qquad \mathbf{a}_i = \mathrm{Col}_i(A)$$

Then to "solve" the linear system, we need to find the correct linear combination of columns of A so that

$$\mathbf{b} = \sum_{i=1}^{n} x_i \mathbf{a}_i$$

We will be able to find this linear combination as long as $\mathbf{b} \in \mathrm{Col}(A)$.

Solution procedure

How do we find that linear combination?

- Matlab (?) pseudo-inverse ! `pinv(A)`
- Do an eigenvector/eigenvalue decomposition

One approach

- Find eigenvalue/eigenvector pairs $(\lambda_i, \mathbf{v}_i)$ satisfying

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad i = 1, 2, \ldots, m$$

- Write $\mathbf{b}$ as

$$\mathbf{b} = \sum_{i=1}^{m} \alpha_i \mathbf{v}_i \quad \text{OR} \quad V\alpha = \mathbf{b}$$

If υ_i are orthogonal (e.g. $\upsilon_i \cdot \upsilon_j = \delta_{ij}$), then

$$\alpha_i = \mathbf{b} \cdot \mathbf{v}_i, \qquad \text{OR} \qquad \alpha = V^T \mathbf{b}$$

- Assume we can expand $\mathbf{x}$ as

$$\mathbf{x} = \sum_{i=1}^{m} \beta_i \mathbf{v}_i$$

$$A\mathbf{x} = \sum_{i=1}^{m} \beta_i A\mathbf{v}_i = \sum_{i=1}^{m} \beta_i \lambda_i \mathbf{v}_i = \sum_{i=1}^{m} \alpha_i \mathbf{v}_i$$

One approach (cont)

This leads to a possible solution strategy. If we can solve

$$\beta_i \lambda_i = \alpha_i$$

for β_i, then we will have our solution $\mathbf{x}$ as a linear combination of eigenvectors.

Solution :

- If $\lambda_i \neq 0$, then $\beta_i = \alpha_i / \lambda_i$
- If $\lambda_i = 0$ for some i then a solution $\mathbf{x}$ exists if and only if $\alpha_i = 0$.

A solution will exist as long as $\mathbf{b}$ does not have any components in $\mathrm{Col}(A)^{\perp}$. (This is one way of stating the *Fredholm Alternative*.)

But does approach even make sense?

Potential pitfalls :

- (some ideas ...)

What do we need for this to be a reasonable approach?

- We would like A to have a complete set of orthogonal eigenvectors.
-

Spectral Theorem for real matrices

Spectral Theorem: A matrix $A \in \mathcal{R}^{n \times n}$ is orthogonally diagonizable if and only if A is symmmetric.

There are many versions of the above, but for our purposes, we use this idea to mean that we can find a unitary matrix such that

$$Q^T A Q = \Lambda, \qquad Q^T Q = I$$

where $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. This means that we have a *complete* set of eigenvectors, i.e. n linearly independent eigenvectors.

Solution procedure

We can now solve $A\mathbf{x} = \mathbf{b}$ as

- Solve for α_i by *projecting* $\mathbf{b}$ onto each column of Q (e.g. the eigenvectors of A).

$$\alpha_i = \mathbf{b} \cdot \mathbf{q}_i, \qquad \mathbf{q}_i = \mathrm{Col}_i(Q)$$

In matrix form, this is just $\alpha = Q^T\mathbf{b}$.

We then use our assumption that $\mathbf{x} = Q\beta$ for an unknown vector β. Then we have

$$\begin{aligned} A\mathbf{x} &= \mathbf{b} \\ Q\Lambda Q^T\mathbf{x} &= \mathbf{b} \\ \Lambda Q^T\mathbf{x} &= Q^T\mathbf{b} = \alpha \\ \Lambda Q^T(Q\beta) &= \alpha \end{aligned}$$

or that finally, ,

$$\Lambda\beta = \alpha$$

If Λ^{-1} exists, the unique solution to the linear system is given by $\mathbf{x} = Q\Lambda^{-1}\alpha = Q\Lambda^{-1}Q^T\mathbf{b}$.

If Λ^{-1} doesn't exist, we can construct a *pseudo-inverse* from the eigenvectors associated with non-zero eigenvalues (to follow).

For matrices ...

From a computational point of view, this leads to a "direct method" for solving the nonsingular linear algebraic system.

- Compute the eigenvalues and eigenvectors of a given symmetric matrix A to get Λ and Q (normalize eigenvectors, if necessary).
- Project b onto columns of Q to get α
- Solve for $\beta = \alpha/\lambda$ to get solution $\mathbf{x} = Q\beta$.

For matrices, this idea can be used in very special cases. For example, matrices arising from finite difference discretizations of certain elliptic PDEs can make use of the Discrete Sine Transform (DST) and Discrete Cosine Transform (DCT). But in general, this idea is not practical for general matrices.

This general idea of making use of a "spectral decomposition" can also be used in solving certain ODEs. For example, Sturm-Liouville problems are solved in the above manner.

Pseudo-inverse method for singular systems

If Λ^{-1} does not exist, (i.e. $\lambda_i = 0$ for one or more i), we no longer have a unique solution, since β_i is not determined by $\lambda_i \beta_i = \alpha_i$. In this case, it is customary to seek a solution for which $\|\mathbf{x}\|$ is a minimum.

To do this, we partition the indices $1, 2, 3, \ldots, N$ into two sets I and $\bar{I}$. The set I contains the indices for which $\lambda_i \neq 0$ and $\bar{I}$ contains the remaining indices. For $i \in I$, we set $\beta_i = \alpha_i / \lambda_i$. For $i \in \bar{I}$, we define a vector $\bar{\beta} \in \mathcal{R}^p$ where $p = N - \text{rank}(A)$. Then we can express

$$\mathbf{x} = \sum_{i \in I} \beta_i \mathbf{q}_i + \sum_{i \in \bar{I}} \bar{\beta}_i \mathbf{q}_i$$

We write this in matrix form as

$$\mathbf{x} = Q_I \beta_I + Q_{\bar{I}} \bar{\beta}$$

where Q_I is a matrix formed from the columns in I and $Q_{\bar{I}}$ is formed from the remaining columns.

The norm of $\mathbf{x}$ is given by

$$\begin{aligned} \|\mathbf{x}\|^2 &= (Q_I \beta_I + Q_{\bar{I}} \bar{\beta})^T (Q_I \beta_I + Q_{\bar{I}} \bar{\beta}) \\ &= \|Q_I \beta_I\|^2 + 2(Q_I \beta_I)^T (Q_{\bar{I}} \beta_{\bar{I}}) + \|Q_{\bar{I}} \beta_{\bar{I}}\|^2 \end{aligned}$$

Because Q_I and $Q_{\bar{I}}$ have orthonormal columns, and $Q_I^T Q_{\bar{I}} = \mathbf{0}$, this reduces to

$$\|\mathbf{x}\|^2 = \|\beta_I\|^2 + \|\beta_{\bar{I}}\|^2 \geq 0$$

This is *Parseval's Identity* applied to Euclidean space, and generalizes the notion of the Pythagorean Theorem.

From this, we see that setting $\beta_{\bar{I}} = \mathbf{0}$ minimizes this quantity. From this, we have

$$\beta_i = \begin{cases} \alpha_i / \lambda_i & \lambda_i \neq 0 \\ 0 & \lambda_i = 0 \end{cases}$$

Using this β, the solution $\mathbf{x} = Q\beta$ is the "pseudo-inverse" solution.

This minimum norm solution only solves $A\mathbf{x} = \mathbf{b}$ exactly if $\mathbf{b} \in \text{Col}(A)$. Otherwise, the solution is $\mathbf{x}$ is only a solution in an approximate sense.

The pseudo-inverse matrix is given by $P = QDQ^T$, where D is a diagonal matrix with diagonal entries d_i given by

$$d_i = \begin{cases} 1/\lambda_i & \lambda_i \neq 0 \\ 0 & \lambda_i = 0 \end{cases}$$

Example

Consider the matrix system

$$\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

The eigenvalues are given by $\lambda_1 = 0$, $\lambda_2 = 5$, with corresponding normalized eigenvectors given by

$$v_1 = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 \\ 1 \end{bmatrix}, \qquad v_2 = \frac{1}{\sqrt{5}} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

The unitary matrix which diagonalizes A is given by

$$Q = \frac{1}{\sqrt{5}} \begin{bmatrix} -2 & 1 \\ 1 & 2 \end{bmatrix}$$

The pseudo-inverse is then

$$P = QDQ^T = \frac{1}{25} \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$$

and the minimum norm solution to our original equation is given by

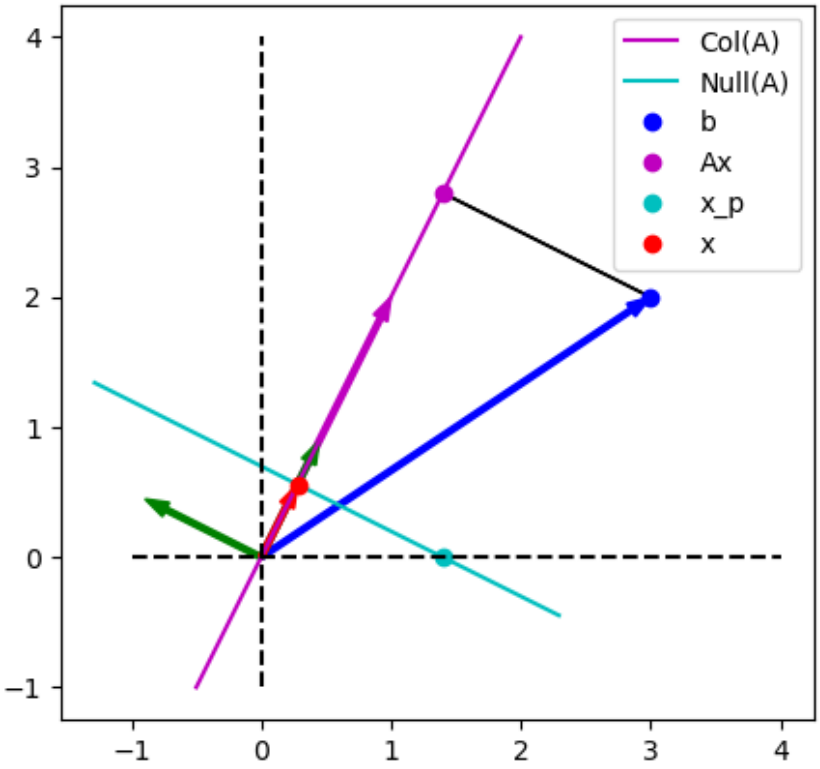
$$\mathbf{x} = P\mathbf{b} = \frac{7}{25} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Graphical interpretation

Graphically, we can interpret the problem above as :

In [117]:
```
figure(1)
clf()

v1 = [-2/sqrt(5), 1/sqrt(5)]
v2 = [1/sqrt(5), 2/sqrt(5)]

colspace = [1,2]
b = [3,2]

xp = [7/5, 0]

x = [7/25, 14/25]


def plot_vector(v,c,width=0.04,origin=[0,0]):
    arrow(*origin,v[0],v[1],color=c,\
```

```
            length_includes_head=True,\
            width=width,\
            label='label')

# plot_vector([7/5,14/5],'k',width=0.005)

#Column space
c1x = -0.5*colspace[0]
c1y = -0.5*colspace[1]

c2x = 2*colspace[0]
c2y = 2*colspace[1]

plot([c1x,c2x],[c1y,c2y],'m',label='Col(A)')
plot_vector(colspace,'m')

# Null space
N = [-2/sqrt(5),1/sqrt(5)]
n1x = xp[0] + -1*N[0]
n1y = xp[1] + -1*N[1]

n2x = xp[0] + 3*N[0]
n2y = xp[1] + 3*N[1]
plot([n1x,n2x],[n1y,n2y],'c',label='Null(A)')

plot_vector(b,'b')
plot_vector([-8/5,4/5],'k',width=0.005,origin=b)
plot(b[0],b[1],'b.',markersize=12,label='b')


plot(7/5,14/5,'m.',markersize=12,label='Ax')

plot(xp[0],xp[1],'c.',markersize=12,label='x_p')


plot_vector(v1,'g')
plot_vector(v2,'g')

# plot_vector(b,'b')

plot_vector(x,'r')
plot(x[0],x[1],'r.',markersize=12,label='x')

plot([-1,4],[0,0],'k--')
plot([0,0],[-1,4],'k--')

legend()

gca().set_aspect('equal')
```


Col(A)
Null(A)
b
Ax
x_p
x
4
3
2
1
0
−1
−1
0
1
2
3
4

What does the length of the red arrow (our $\mathbf{x}$ solution) mean?

We can first look at what $A\mathbf{x}$ gives us.

$$A\mathbf{x} = \frac{7}{5}\begin{bmatrix}1\\2\end{bmatrix}$$

It turn out, this is the projection of $\mathbf{b}$ onto the columns space of A. The solution to this equation is, however, not unique (forgetting for the moment that we solved for $\mathbf{x}$ above).

To find the $\mathbf{x}$ that minimizes $\|\mathbf{x}\|$, we can write

$$\mathbf{x} = \mathbf{x}_p + s\ \mathbf{n}, \qquad \mathbf{n} = \frac{1}{\sqrt{5}}\begin{bmatrix}-2\\1\end{bmatrix}$$

where $\mathbf{n}$ represents the nullspace of A. We can minimize this over s to get $s = 14/25$. Substituing this back into the above, we get

$$\mathbf{x} = \frac{7}{25}\begin{bmatrix}1\\2\end{bmatrix}$$

which is exactly $P\mathbf{b}$ from above.

Code : Pseudo-inverse for symmetric matrices

Below, we compute the pseudo-inverse for a symmetric matrix and compare it to `linalg.pinv` from Numpy.

For general matries, the `pinv` solution uses the SVD. But for symmetric matrices, the eigenvalue decomposition and the SVD are essentially the same.

In [27]:
```
from numpy.linalg import eig,matrix_rank,solve,pinv

N = 10
b = rand(N,1)


# Construct symmetric matrix  A from B.
B = rand(N,N)
A = 0.5*(B + B.transpose())

# Make rank(A) < N
A[3:6,:] = 0
A[:,3:6] = 0
```

```
# ---------------------------
# Solve the linear system
# ---------------------------

# Compute eigenvalues
L,Q = eig(A)

# Vector of eigenvalues
L = reshape(L,(N,1))

# Project b onto columns of Q
alpha = Q.transpose()@b

if matrix_rank(A) == N:
    print('A has full rank')
else:
    print('A has rank {:d}'.format(matrix_rank(A)))

tol = 1e-12
L1 = where(abs(L) >  tol, L, 1)   # Set 0 entries to 1 to avoid divide
by zero in next line
beta = where(L != 0, alpha/L1, 0)

x = Q@beta
x_true = pinv(A)@b   # works even if A has full rank

e = abs(x-x_true)/norm(x)

print('{:>24s} {:>24s} {:>12s}'.format('x (eig)','x (pinv)','Differenc
e'))
print('{:s}'.format('-'*62))
for i in range(N):
    print('{:24.16f} {:24.16f} {:12.4e}'.format(x[i,0],x_true[i,0],e[i
,0]))

# Compute the pseudo-inverse P
L.shape = (N,)
L1.shape = (N,)
D = diag(where(abs(L) > tol, 1/L1, 0))

P = Q@D@Q.transpose()
diff = norm(pinv(A)-P)
print("")
print('|P - linalg.pinv| : {:.4e}'.format(diff))
```

```
A has rank 7
                 x (eig)                 x (pinv)   Difference
------------------------------------------------------------
      0.1563393601561082       0.1563393601561093   4.1891e-16
     -1.1224336515410815      -1.1224336515410791   9.2159e-16
      1.5700230633395533       1.5700230633395502   1.1729e-15
      0.0000000000000000      -0.0000000000000188   7.0797e-15
      0.0000000000000000      -0.0000000000000026   9.9208e-16
      0.0000000000000000      -0.0000000000000016   6.1650e-16
     -0.4491517184308114      -0.4491517184308117   1.0473e-16
      0.5566322873854586       0.5566322873854577   3.3512e-16
      1.0923451153936150       1.0923451153936130   7.5403e-16
     -1.2530007415628439      -1.2530007415628406   1.2567e-15

|P - linalg.pinv| : 7.9025e-14
```

What next?

We will extend these ideas to *infinite dimensional* vector spaces, or *Hilbert* spaces.

- Our matrix will be replaced with a differential operator that is *self-adjoint*.
- This differential operator will have properties in the Hilbert space that are analogous to those we saw for the symmetric matrix in $\mathcal{R}^N$.
- From this, we can then solve a wide variety of problems using eigenvalues and eigenfunctions of this differential operator.

In [ ]: