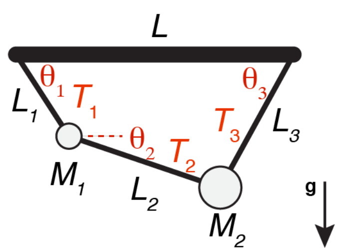
14_Linear_Algebra

April 2, 2020

1 14 Linear Algebra

1.1 Motivating problem: Two masses on three strings

Two masses M_1 and M_2 are hung from a horizontal rod with length L in such a way that a rope of length L_1 connects the left end of the rod to M_1, a rope of length L_2 connects M_1 and M_2, and a rope of length L_3 connects M_2 to the right end of the rod. The system is at rest (in equilibrium under gravity).



Find the angles that the ropes make with the rod and the tension forces in the ropes.

In class we derived the equations that govern this problem – see 14_String_Problem_lecture_notes (PDF).

We can represent the problem as system of nine coupled non-linear equations:

$$\mathbf{f}(\mathbf{x}) = 0$$

1.1.1 Summary of equations to be solved

Treat $\sin\theta_i$ and $\cos\theta_i$ together with T_i, $1 \le i \le 3$, as unknowns that have to simultaneously fulfill the nine equations

$$
\begin{aligned}
-T_1\cos\theta_1 + T_2\cos\theta_2 &= 0 && (1)\\
T_1\sin\theta_1 - T_2\sin\theta_2 - W_1 &= 0 && (2)\\
-T_2\cos\theta_2 + T_3\cos\theta_3 &= 0 && (3)\\
T_2\sin\theta_2 + T_3\sin\theta_3 - W_2 &= 0 && (4)\\
L_1\cos\theta_1 + L_2\cos\theta_2 + L_3\cos\theta_3 - L &= 0 && (5)\\
-L_1\sin\theta_1 - L_2\sin\theta_2 + L_3\sin\theta_3 &= 0 && (6)\\
\sin^2\theta_1 + \cos^2\theta_1 - 1 &= 0 && (7)\\
\sin^2\theta_2 + \cos^2\theta_2 - 1 &= 0 && (8)\\
\sin^2\theta_3 + \cos^2\theta_3 - 1 &= 0 && (9)
\end{aligned}
$$

Consider the nine equations a vector function $\mathbf{f}$ that takes a 9-vector $\mathbf{x}$ of the unknowns as argument:

$$\mathbf{f}(\mathbf{x}) = 0 \tag{10}$$

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} = \begin{pmatrix} \sin\theta_1 \\ \sin\theta_2 \\ \sin\theta_3 \\ \cos\theta_1 \\ \cos\theta_2 \\ \cos\theta_3 \\ T_1 \\ T_2 \\ T_3 \end{pmatrix} \tag{11}$$

$$\mathbf{L} = \begin{pmatrix} L \\ L_1 \\ L_2 \\ L_3 \end{pmatrix}, \quad \mathbf{W} = \begin{pmatrix} W_1 \\ W_2 \end{pmatrix} \tag{12}$$

In more detail:

$$
\begin{aligned}
f_1(\mathbf{x}) &= -x_6x_3 + x_7x_4 && = 0 && (13)\\
f_2(\mathbf{x}) &= x_6x_0 - x_7x_1 - W_1 && = 0 && (14)\\
&\ldots && && (15)\\
f_8(\mathbf{x}) &= x_2^2 + x_5^2 - 1 && = 0 && (16)
\end{aligned}
$$

We generalize the *Newton-Raphson algorithm* from the last lecture to n dimensions:

1.2 General Newton-Raphson algorithm

Given a trial vector $\mathbf{x}$, the correction $\Delta\mathbf{x}$ can be derived from the Taylor expansion

$$f_i(\mathbf{x} + \Delta\mathbf{x}) = f_i(\mathbf{x}) + \sum_{j=1}^{n} \left.\frac{\partial f_i}{\partial x_j}\right|_{\mathbf{x}} \Delta x_j + \dots$$

or in full vector notation

$$\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}) = \mathbf{f}(\mathbf{x}) + \left.\frac{d\mathbf{f}}{d\mathbf{x}}\right|_{\mathbf{x}} \Delta\mathbf{x} + \dots \tag{17}$$

$$= \mathbf{f}(\mathbf{x}) + \mathsf{J}(\mathbf{x})\Delta\mathbf{x} + \dots \tag{18}$$

where $\mathsf{J}(\mathbf{x})$ is the *Jacobian* matrix of $\mathbf{f}$ at $\mathbf{x}$, the generalization of the derivative to multivariate vector functions.

Solve

$$\mathbf{f}(\mathbf{x} + \Delta\mathbf{x}) = 0$$

i.e.,

$$\mathsf{J}(\mathbf{x})\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x})$$

for the correction Δx

$$\Delta\mathbf{x} = -\mathsf{J}(\mathbf{x})^{-1}\mathbf{f}(\mathbf{x})$$

which has the same form as the 1D Newton-Raphson correction $\Delta x = -f'(x)^{-1}f(x)$.

These are *matrix equations* (we linearized the problem). One can either explicitly solve for the unknown vector $\Delta\mathbf{x}$ with the inverse matrix of the Jacobian or use other methods to solve the coupled system of linear equations of the general form

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

1.3 Linear algebra with `numpy.linalg`

[1]:
```python
import numpy as np
```

[2]:
```python
np.linalg?
```

1.3.1 System of coupled linear equations

Solve the coupled system of linear equations of the general form

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

[2]:
```python
A = np.array([
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 2]
    ])
b = np.array([1, 0, 1])
```

What does this system of equations look like?

[3]:
```
for i in range(A.shape[0]):
    terms = []
    for j in range(A.shape[1]):
        terms.append("{1} x[{0}]".format(j, A[i, j]))
    print(" + ".join(terms), "=", b[i])
```

```
1 x[0] + 0 x[1] + 0 x[2] = 1
0 x[0] + 1 x[1] + 0 x[2] = 0
0 x[0] + 0 x[1] + 2 x[2] = 1
```

Now solve it with `numpy.linalg.solve`:

[4]:
```
x = np.linalg.solve(A, b)
print(x)
```

```
[1.  0.  0.5]
```

Test that it satisfies the original equation:

$$\mathsf{A}\mathbf{x} - \mathbf{b} = 0$$

[5]:
```
np.dot(A, x) - b
```

[5]:
```
array([0., 0., 0.])
```

Activity: Solving matrix equations With

$$\mathsf{A}_1 = \begin{pmatrix} +4 & -2 & +1 \\ +3 & +6 & -4 \\ +2 & +1 & +8 \end{pmatrix}$$

and

$$\mathbf{b}_1 = \begin{pmatrix} +12 \\ -25 \\ +32 \end{pmatrix}, \quad \mathbf{b}_2 = \begin{pmatrix} +4 \\ -1 \\ +36 \end{pmatrix},$$

solve for $\mathbf{x}_i$

$$\mathsf{A}_1\mathbf{x}_i = \mathbf{b}_i$$

and *check the correctness of your answer.*

[7]:
```
A1 = np.array([
        [+4, -2, +1],
        [+3, +6, -4],
        [+2, +1, +8]
    ])
b1 = np.array([+12, -25, +32])
b2 = np.array([ 4, -1, 36])
```

[8]:
```
x1 = np.linalg.solve(A1, b1)
print(x1)
print(A1.dot(x1) - b1)
```

```
[ 1. -2.  4.]
[ 0.  0.  0.]
```

[9]:
```
x2 = np.linalg.solve(A1, b2)
print(x2)
print(A1.dot(x2) - b2)
```

```
[ 1.  2.  4.]
[ -8.88178420e-16   0.00000000e+00   7.10542736e-15]
```

1.3.2 Matrix inverse

In order to solve directly we need the inverse of A:

$$\mathsf{A}\mathsf{A}^{-1} = \mathsf{A}^{-1}\mathsf{A} = 1$$

Then

$$\mathbf{x} = \mathsf{A}^{-1}\mathbf{b}$$

If the inverse exists, `numpy.linalg.inv()` can calculate it:

[10]:
```
Ainv = np.linalg.inv(A)
print(Ainv)
```

```
[[ 1.   0.   0. ]
 [ 0.   1.   0. ]
 [ 0.   0.   0.5]]
```

Check that it behaves like an inverse:

[11]:
```
Ainv.dot(A)
```

[11]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

[12]:
```
A.dot(Ainv)
```

[12]:
```
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Now solve the coupled equations directly:

[13]:
```
Ainv.dot(b)
```

[13]:
```
array([ 1. ,  0. ,  0.5])
```

Activity: Solving coupled equations with the inverse matrix

1. Compute the inverse of A_1 and *check the correctness*.
2. Compute $\mathbf{x}_1$ and $\mathbf{x}_2$ with A_1^{-1} and check the correctness of your answers.

[14]:
```
A1_inv = np.linalg.inv(A1)
print(A1_inv)
```

```
[[ 0.19771863  0.06463878  0.00760456]
 [-0.121673    0.11406844  0.07224335]
 [-0.03422053 -0.03041825  0.11406844]]
```

[15]:
```
A1.dot(A1_inv)
```

[15]:
```
array([[  1.00000000e+00,   6.93889390e-18,   0.00000000e+00],
       [ -5.55111512e-17,   1.00000000e+00,   1.11022302e-16],
       [  0.00000000e+00,  -5.55111512e-17,   1.00000000e+00]])
```

[16]:
```
A1_inv.dot(A1)
```

[16]:
```
array([[  1.00000000e+00,  -3.46944695e-18,   5.55111512e-17],
       [  2.77555756e-17,   1.00000000e+00,   2.22044605e-16],
       [  2.77555756e-17,   1.38777878e-17,   1.00000000e+00]])
```

[17]:
```
x1 = A1_inv.dot(b1)
print(x1)
print(A1.dot(x1) - b1)
```

```
[ 1. -2.  4.]
[  0.00000000e+00   0.00000000e+00   7.10542736e-15]
```

[18]:
```
x2 = A1_inv.dot(b2)
print(x2)
print(A1.dot(x2) - b2)
```

```
[ 1.  2.  4.]
[  0.00000000e+00   3.55271368e-15   7.10542736e-15]
```

1.3.3 Eigenvalue problems

The equation

$$\mathsf{A}\mathbf{x}_i = \lambda_i \mathbf{x}_i \tag{19}$$

is the **eigenvalue problem** and a solution provides the eigenvalues λ_i and corresponding eigenvectors x_i that satisfy the equation.

Example 1: Principal axes of a square The principle axes of the moment of inertia tensor are defined through the eigenvalue problem

$$\mathsf{I}\omega_i = \lambda_i \omega_i$$

The principal axes are the ω_i.

[20]:
```
Isquare = np.array([[2/3, -1/4], [-1/4, 2/3]])
```

[22]:
```
lambdas, omegas = np.linalg.eig(Isquare)
```

[23]:
```
lambdas
```

[23]:
```
array([ 0.91666667,  0.41666667])
```

[24]:
```
omegas
```

[24]:
```
array([[ 0.70710678,  0.70710678],
       [-0.70710678,  0.70710678]])
```

Note that the eigenvectors are `omegas[:, i]`! You can transpose so that axis 0 is the eigenvector index:

[32]:
```
omegas.T
```

[32]:
```
array([[ 0.70710678, -0.70710678],
       [ 0.70710678,  0.70710678]])
```

Test:

$$(\mathsf{I} - \lambda_i \mathbf{1})\omega_i = 0$$

(The identity matrix can be generated with `np.identity(2)`.)

[30]:
```
(Isquare - lambdas[0]*np.identity(2)).dot(omegas[:, 0])
```

[30]:
```
array([ 0.,  0.])
```

[36]:
```
(Isquare - lambdas[0]*np.identity(2)).dot(omegas.T[0])
```

[36]:
```
array([ 0.,  0.])
```

[40]:
```
(Isquare - lambdas[1]*np.identity(2)).dot(omegas.T[1])
```

[40]:
```
array([ 0.,  0.])
```

Example 2: Spin in a magnetic field In quantum mechanics, a spin 1/2 particle is represented by a spinor χ, a 2-component vector. The Hamiltonian operator for a stationary spin 1/2 particle in a homogenous magnetic field B_y is

$$\mathsf{H} = -\gamma \mathsf{S}_y B_y = -\gamma B_y \frac{\hbar}{2}\sigma_\mathrm{y} = \hbar\omega \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

Determine the *eigenvalues* and *eigenstates*

$$\mathsf{H}\chi = E\chi$$

of the spin 1/2 particle.

(To make this a purely numerical problem, divide through by $\hbar\omega$, i.e. calculate $E/\hbar\omega$.)

[43]:
```
sigma_y = np.array([[0, -1j], [1j, 0]])
E, chis = np.linalg.eig(sigma_y)
print(E)
print(chis.T)
```

```
[ 1.+0.j -1.+0.j]
[[-0.00000000-0.70710678j  0.70710678+0.j        ]
 [ 0.70710678+0.j          0.00000000-0.70710678j]]
```

Normalize the eigenvectors:

$$\hat{\chi} = \frac{1}{\sqrt{\chi^\dagger \cdot \chi}} \chi$$

[52]:
```
chi1 = chis.T[0]
print(chi1)
norm = np.dot(chi1.conjugate(), chi1)
chi1_hat = chi1/np.sqrt(norm)
print(chi1_hat)
```

```
[-0.00000000-0.70710678j  0.70710678+0.j        ]
[-0.00000000-0.70710678j  0.70710678+0.j        ]
```

[46]:
```
norm
```

[46]:
```
(0.99999999999999989+0j)
```

… they were already normalized.

Activity: eigenvalues Find the eigenvalues and eigenvectors of

$$\mathsf{A}_2 = \begin{pmatrix} -2 & +2 & -3 \\ +2 & +1 & -6 \\ -1 & -2 & +0 \end{pmatrix}$$

Are the eigenvectors normalized?

Check your results.

[41]:
```
A2 = np.array([[-2, +2, -3],
    [+2, +1, -6],
    [-1, -2, +0]])
```

[48]:
```
lambdas, evecsT = np.linalg.eig(A2)
evecs = evecsT.T
print(lambdas)
print(evecs)
```

```
[-3.  5. -3.]
[[-0.95257934  0.27216553 -0.13608276]
 [ 0.40824829  0.81649658 -0.40824829]
 [ 0.05155221  0.82292764  0.5658025 ]]
```

[54]:
```python
np.linalg.norm(evecs, axis=1)
```

[54]:
```
array([ 1.,  1.,  1.])
```

[57]:
```python
Identity = np.identity(A2.shape[0])
for evalue, evec in zip(lambdas, evecs):
    print((A2 - evalue * Identity).dot(evec))
```

```
[ -3.88578059e-16   1.11022302e-16  -1.11022302e-16]
[ -6.66133815e-16   2.66453526e-15   0.00000000e+00]
[ 0.  0.  0.]
```

[ ]:
```python

```