Rayleigh-Ritz method

The usual boilerplate...

```
%matplotlib inline

import numpy as np
from scipy.linalg import eigh, inv
import matplotlib.pyplot as plt
```

A function to solve the reduced eigenvalues problem: we need on input

1. a Ritz base $\boldsymbol{\Phi}$ and
2. the structural matrices $\boldsymbol{K}$ and $\boldsymbol{M}$,

then we compute the reduced matrices and solve the eigenvalue problem using a library function, obtaining the eigenvalues and the eigenvectors *in Ritz coordinates* and finally return the eigenvalues and the eigenvectors *in dynamic coordinates*.

```python
def red_eigh(Phi, K, M):
    red_K = Phi.T@K@Phi
    red_M = Phi.T@M@Phi
    evals, zvec = eigh(red_K, red_M)
    return evals, Phi@zvec
```

`Eigh` computes *mass normalized* eigenvectors but there is still one degree of freedom, i.e., the sign. Being a control freak, I like to have all the top values positive. `change_sign` looks if the top (last) value of a column is negative, in that case it multiplies all the column by `-1`

```python
def change_sign(shapes):
    for n, val in enumerate(shapes[-1]):
        if val<0 : shapes[:,n] *= -1
```

Sometimes we want to plot some modal shape, or some Ritz vector. Here it is a function to do that

```
def plot_shapes(N, shapes):
   for n, shape in enumerate(shapes.T, 1):
       plt.plot(np.arange(N+1), np.hstack((0, shape)), label='%d'%n)
   plt.legend(loc='best') ; plt.grid()
```

Structural Matrices

We start with the number of floors, the masses are all equal ($M = mI$) and the storey stiffnesses go from `37 k` to `8 k`, the upper and lower diagonals are the negative of the `k`'s except the last term, the principal diagonal is the sum of the upper and lower storey stifnesses, except for the last floor.

```
N = 30
M = np.eye(N)
k = 37-np.arange(N)
d = -k[1:]
k[:-1] += k[1:]
K = np.diag(k)+np.diag(d,+1)+np.diag(d,-1)
```

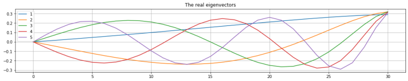
The Real Eigenv*

We want a reference solution, won't we? So we compute the eigenvalues and the eigenvectors, print the lowest eigenvalues and plot the lowest eigenvectors.

```
plt.rcParams["figure.figsize"] = (18,3)

true_evals, true_evecs = eigh(K, M)
change_sign(true_evecs)
print('The real eigenvalues:', true_evals[:5])
plt.title('The real eigenvectors')
plot_shapes(N, true_evecs[:,:5])
```

```
The real eigenvalues: [0.06995811 0.48026479 1.28937631 2.48964376
4.06603788]
```

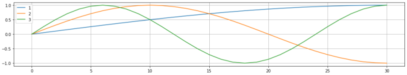


The Rayleigh-Ritz solution

The base

We compute the base according to the instructions in the text of the problem.

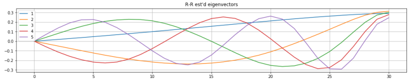
```
L = 10
n, j = np.arange(N)+1, np.arange(L)+1
Phi = np.sin(np.pi*n[:,None]/N*(j-1/2))
plot_shapes(N, Phi[:,:3])
```



The solution

```
est_evals, est_evecs = red_eigh(Phi, K, M) ; change_sign(est_evecs)
print('Eigenvalues:  ', true_evals[:5]) ; print('R-R estimates:', est_evals[:5])
plt.title('R-R est\'d eigenvectors') ; plot_shapes(N, est_evecs[:,:5])
```

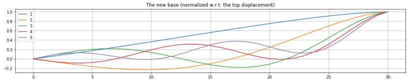
```
Eigenvalues:   [0.06995811 0.48026479 1.28937631 2.48964376 4.066037
88]
R-R estimates: [0.06997053 0.48102404 1.29518935 2.5141276  4.141755
33]
```



Not bad, not really good.

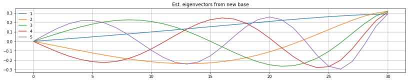
Let's try to see what happens if we derive a better (?) base using the dynamic matrix...

```
Phi2 = inv(K)@M@Phi
plt.title('The new base (normalized w.r.t. the top displacement)')
plot_shapes(N, Phi2[:,:5]/Phi2[-1,:5])
```



```
est2_evals, est2_evecs = red_eigh(Phi2, K, M)
change_sign(est2_evecs)
print('Eigenvalues:  ', true_evals[:5]) ; print('R-R estimates:', est2_evals[:5])
plt.title('Est. eigenvectors from new base')
plot_shapes(N, est2_evecs[:,:5])
```

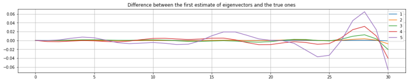
```
Eigenvalues:   [0.06995811 0.48026479 1.28937631 2.48964376 4.066037
88]
R-R estimates: [0.06995811 0.48026587 1.28943808 2.49063294 4.074357
07]
```



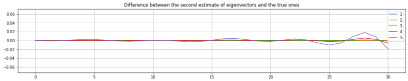
A big improvement in estimation of eigenvalues and also the eigenvectors look better...

In the next slides we will plot the differences between the estimated and the true eigenvectors, using the same vertical scale

```
plt.title('Difference between the first estimate of eigenvectors and the true ones'
)
plot_shapes(N,  est_evecs[:,:5]-true_evecs[:,:5]) ;
yl = plt.ylim()
```



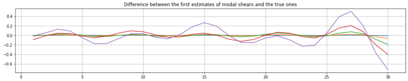
```
plt.title('Difference between the second estimate of eigenvectors and the true one
s')
plt.ylim(yl)
plot_shapes(N, est2_evecs[:,:5]-true_evecs[:,:5])
```



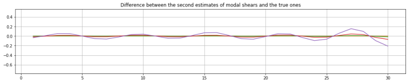
Here, the lowest 3 eigenvectors are OK, the 4th is good and also the last one is correctly approximated. This works because we are looking at displacements, let's look at storey shears...

```
Ns = np.arange(N)+1
k = np.diag(38-Ns)
# first, compute the DeltaX between two floors (including foundation, i.e., x=0),
# next multiply by storey stiffness
V0 = true_evecs[:,:5]-np.vstack((np.zeros(5),true_evecs[:-1,:5])) ; V0 = k@V0
V1 =  est_evecs[:,:5]-np.vstack((np.zeros(5), est_evecs[:-1,:5])) ; V1 = k@V1
V2 = est2_evecs[:,:5]-np.vstack((np.zeros(5),est2_evecs[:-1,:5])) ; V2 = k@V2
```

```
plt.plot(Ns, V1-V0) ; plt.grid()
plt.title('Difference between the first estimates of modal shears and the true one
s')
yl = plt.ylim()
```



```
plt.ylim(yl) ; plt.grid()
plt.title('Difference between the second estimates of modal shears and the true one
s')
plt.plot(Ns, V2-V0);
```



Also the storey shears are OK. Shear type models of buildings are however particular inasmuch the shear is proportional to the first derivative of the lateral displacements, and it is hence *easier* to determine w.r.t. the general case.