Bound on Rel_Error / Recover error

If ask ABS, cannot compute.

rel = old rel * cond

Error of Inverse Iteration

```
lam1, lam2, lam3 = -7, -2, -1
u = np.array([-1, 0, 1])
x0 = np.array([-1, -1, 1])
n = 4
sigma = -0.5

e0 = la.norm(x0 - u, np.inf)
lams = np.array([lam1, lam2, lam3])
def find_convergence(lams, sigma= 0):
    new_lams = list(lams - sigma)
    new_lams.sort(key = abs)
    return abs(new_lams[0] / new_lams[1])
find_convergence(lams) ** (n) * e0
```

$A = XDX^{-1}$

```
# Similarity Transformation of a Matrix
A = np.array([[7, -1],
              [0, 6]])
a = la.eig(A)[1]
ans = np.zeros(a.shape)
for i in range(A.shape[1]):
    vector = a[:, i] / la.norm(a[:, i],1)
    ans[:,i] = vector # x
x = ans
d = np.array([
    [7,0],
    [0,6]
])
print(x,d)
```

$B = A - 6I$ $\quad$ $Eig(B) = 2$

$\frac{1}{x-6} = 2$

There is a risk of overflow (V)

---

webpage rank.

```
def convert(B):
    index = np.where(~B.any(axis=0))[0]
    B[:,index] += 1
    Mb = B / np.sum(B, axis = 0)
    return Mb
A = np.array([[0, 0, 0, 1],
              [0, 0, 0, 1],
              [0, 1, 0, 1],
              [1, 0, 0, 0]
             ])
alpha = 0.9
S = convert(A)
G = alpha * S + (1-alpha)/A.shape[0]
G
```

A为邻接矩阵

第一列表示0能到哪

Matrix Cond Approx.

Lower bound.

$X = [x_1, x_2, x_3]$ $\quad$ $AX = [Ax_1, Ax_2, Ax_3]$

$A = \frac{AX}{X}$. ANS = np.max(A) / np.min(A)

Power iteration. 求次数.

```
lamda1 = -6
lamda2 = 5
fac = 1e-3

x = abs(lamda2 / lamda1)
if x > 1:
    x = 1/x
factor = x
count = 1
while x > fac:
    x *= factor
    count += 1
count
```

Fast coverage.

Power $\quad e_{k+1} = e_k \cdot \left|\frac{\lambda_2}{\lambda_1}\right|$ $\quad$ 系数越小.

Inv $\quad e_{k+1} = e_k \cdot \frac{|\lambda_{close} - \sigma|}{|\lambda_{sec\text{-}close} - \sigma|}$ $\quad$ 越快.

λ_1 IEEE 754 $\quad x_0 = 1\ \Sigma_n$

λ_n Inifinte Iteratio

复杂度 PI: kn^2

else: $n^3 + kn^2$

选出最快收敛σ.

```
lam = np.array([3, 12, 91])
sigmas = [-88.1, 1, -94, 91.1, -7, -4]
temp = np.inf
idx = 0
for i in range(len(sigmas)):
    sigma = sigmas[i]
    lam_new = abs(lam+sigma)
    lam_new = np.sort(lam_new)

    e = lam_new[0]/lam_new[1]

    if e < temp:
        temp = e
        idx = i
    print(temp)
sigmas[idx]
```

算 cond 2.

```
lam = np.array([3, 12, 91])
sigmas = [-88.1, 1, -94, 91.1, -7, -4]
temp = np.inf
idx = 0
for i in range(len(sigmas)):
    sigma = sigmas[i]
    lam_new = abs(lam+sigma)
    lam_new = np.sort(lam_new)

    e = lam_new[0]/lam_new[1]

    if e < temp:
        temp = e
        idx = i
    print(temp)
sigmas[idx]
```

求VT

找出 $\vec{b}$ 中最大(PI) 或最接近 σ (Inv) 一列.

$\vec{x}$ 中对应 除 la.norm(, 2/∞).

---

ill-con $\quad$ A stretch.

The result $\vec{b} = \vec{A}\vec{x}$ changes a lot. $\quad$ True

$\|A\|\|A^{-1}\|$ very large. $\quad$ $\boxed{|\det(A) = 0|}$?

---

well-con $\quad$ A stretches ... $\quad$ Fake

$|\det(A)| = 0$

$\|A\|\|A^{-1}\|$ large The result.

$\|\Delta x\| / \|x\|$ $\quad$ righ-hand rule

$A = 6$, $B = 14$ $\quad$ $fac = 1e^{-4}$

$A * B * fac$

```
lama1 = 3
lama2 = 2
coeff = 1
sigma = 4

lama = np.array([lama1, lama2])
lamb = lama - sigma
print('larger eigen = %s' %np.max(lamb))
print('smaller eigen = %s' %np.min(lamb))
```

Cond and rel_error

```
import numpy as np
import numpy.linalg as la

err_xhat = xtrue - xhat
rel_err_xhat = la.norm(err_xhat, 2) / la.norm(xtrue, 2)
err_Axhat = A @ err_xhat
rel_err_Axhat = la.norm(err_Axhat, 2) / la.norm(A @ xtrue, 2)
cond_A = la.norm(A, 2) * la.norm(la.inv(A), 2)
bound_rel_err_Axhat = cond_A * rel_err_xhat
```

CSR

```
import numpy as np

A = np.zeros(A_csr.shape)

for i in range(1, A_csr.indptr.shape[0]):
    count = A_csr.indptr[i] - A_csr.indptr[i - 1]
    for j in range(count):
        column = A_csr.indices[A_csr.indptr[i - 1] + j]
        value = A_csr.data[A_csr.indptr[i - 1] + j]
        A[i-1][column] = value
```

Train

```
import numpy as np

mat = np.array([[0, 1, 1, 1, 0],
                [1, 0, 0, 1, 0],
                [1, 1, 0, 1, 0],
                [1, 0, 0, 0, 1],
                [1, 0, 1, 0, 0]])
mat = mat / np.sum(mat, axis = 0)
a = np.linalg.eig(mat)[1][:, 0].astype(np.float64)
prob = a/np.sum(a)
```

Magic Pot

```
import numpy as np
import numpy.linalg as la

A_complete = A.copy()
A_complete[missing_row_index] = 1 - np.sum(A, axis = 0)
A_complete[missing_row_index][missing_col_index] = 0
col = (comp2 - A_complete @ comp1) / comp1[missing_col_index]
A_complete[:, missing_col_index] = col



def final_comp(A):
    # complete your function
    # return 1d numpy array with final ingredient composition
    '''
    eig = la.eig(A)
    arr = eig[0]
    idx = np.argmin(abs(arr-1))
    x = eig[1][idx]
    '''
    comp = np.random.rand(A.shape[0])
    comp /= la.norm(comp, 1)
    while not np.array_equal(A @ comp, comp):
        comp = A @ comp
    return comp
```

Estimate correct bits.

```
import numpy as np
import numpy.linalg as la

x = np.linalg.solve(A,b)

n =np.log10(np.linalg.cond(A))
# print(n)
correct_digits = int(-np.log10(2**-52) -n)
```