Prove that for a real nxn matrix, the matrix norm induced by the vector infinity-norm is the maximum row sum of absolute values.

Homework 4.1

Show $\|\cdot\|$ defined by $\|A\| = \sup \frac{\|Ax\|_\infty}{\|x\|_\infty}$ is $\|A\| = \max_i \sum_j |a_{ij}|$.

Well, $\|Ax\|_\infty = \max_i \left|\sum_j a_{ij} x_j\right|$ def of matrix vector product

$\leq \max_i \sum_j |a_{ij}||x_j|$ $\{ |a+b| \leq |a|+|b|$, $|ab| = |a||b|$

$\leq \max_i \sum_j |a_{ij}| \cdot \max_k |x_k|$ each x_j no larger than largest

$= \max_i \sum_j |a_{ij}| \, \|x\|_\infty$ def of vector ∞-norm

So $\frac{\|Ax\|_\infty}{\|x\|_\infty} \leq \max_i \sum_j |a_{ij}|$ divide by $\|x\|_\infty$

and $\|A\|_\infty \leq \max_i \sum_j |a_{ij}|$ $\|A\|_\infty \leq \frac{\|Ax\|_\infty}{\|x\|_\infty}$ $\forall x \neq 0$

It remains to show $\|A\|_\infty \geq \max_i \sum_j |a_{ij}|$ also, so that equality holds.

We try to choose a vector y of norm 1 that is maximally stretched by A.

Let the max row sum of absolute values be attained on row k:

$$\sum_j |a_{kj}| = \max_i \sum_j |a_{ij}|$$

and define y by $y_j = \begin{cases} a_{kj}/|a_{kj}| & \text{if } a_{kj} \neq 0 \\ 0 & \text{if } a_{kj} = 0 \end{cases}$ (conjugate if A complex)

Example

A
2 0 -3 1 7 -9

row k

so that $(Ay)_k = \sum_j |a_{kj}|$ (*)

Then $y = (1\ 0\ -1\ 1\ 1\ -1)^T$ and $\|y\|_\infty = 1$

$\|A\|_\infty \geq \frac{\|Ay\|_\infty}{\|y\|_\infty} \geq |(Ay)_k|/1$ The largest is at least as big as any

$\overset{(*)}{=} \sum_j |a_{kj}| = \max_i \sum_j |a_{ij}|$ by choice of k

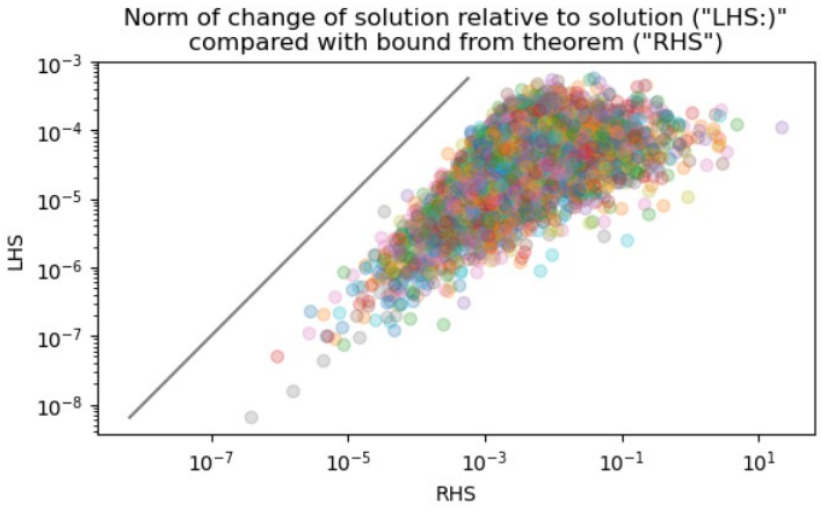
In summary, we've shown $\|A\|_\infty \geq \max_i \sum_j |a_{ij}|$

and $\|A\|_\infty \leq \max_i \sum_j |a_{ij}|$.

so $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ QED ☺

4.2 The following code generates 2000 random linear systems and a random perturbation of each, and makes a picture. Run it and describe/show how the results are consistent or not with Theorem 3.10.

```
import numpy as np
import matplotlib.pyplot as plt

def vnorm(x): return np.abs( x ).max()   # alternatively np.linalg.norm(x,np.inf)
def mnorm(a): return np.abs(a   ).sum(axis=1).max()

n = 5

plt.subplot(111,aspect=1)
smallestl,biggestl = 1.,0.
smallestr,biggestr = 1.,0.
for k in range(10000):

    a0 = np.random.randn(n,n)
    a0inv = np.linalg.inv(a0)
    ka = mnorm(a0)*mnorm(a0inv)
    x0 = np.random.randn(n)
    b0 = a0@x0

    delta = 1.e-4*np.random.rand()
    da = delta*np.random.randn(n,n)
    dx = delta*np.random.randn(n)    # should perturb solution rather than rhs for greater reliability
    a = a0 + da
    b = a@(x0+dx)
    db = b - b0

    lhs = vnorm(dx)/vnorm(x0)  # relative change in solution
    rhs = float( ka/(1 - mnorm(da)*mnorm(a0inv))*( vnorm(db)/vnorm(b) + mnorm(da)/mnorm(a) ))
                              # bound on relative error from Theorem 3.10

    if lhs < smallestl: smallestl = lhs
    if rhs < smallestr: smallestr = rhs
    if lhs > biggestl: biggestl = lhs
    if rhs > biggestr: biggestr = rhs

    plt.loglog(rhs,lhs,'o',alpha=0.25)

plt.loglog([smallestl,biggestl],[smallestl,biggestl],'k',alpha=0.5)

plt.xlabel('RHS')
plt.ylabel('LHS')
plt.title('Norm of change of solution relative to solution ("LHS:)"\ncompared with bound from theorem ("RHS")');
```



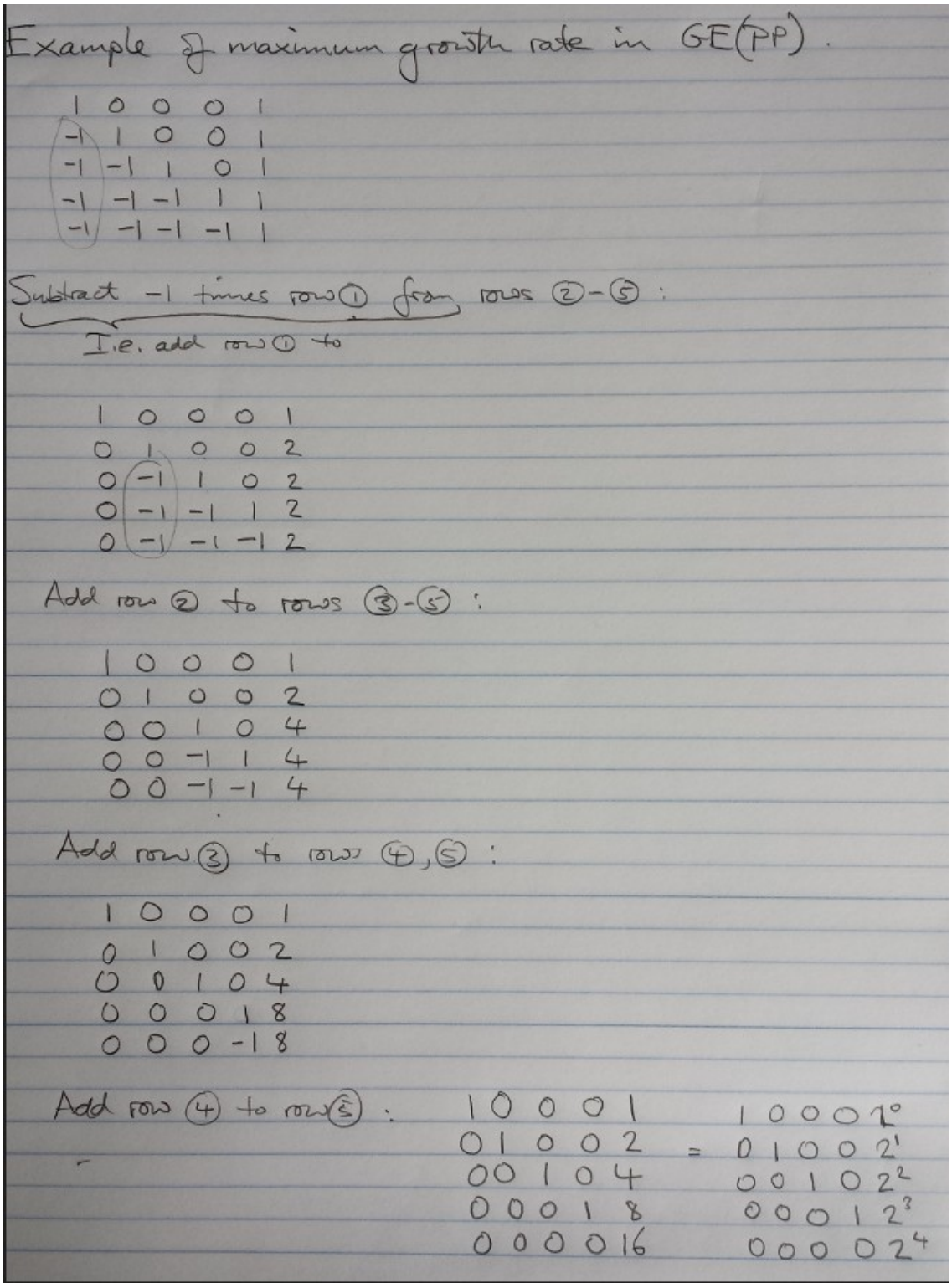
Experimental results are consistent with the theorem in that for every system examined, the bound is greater than the actual relative change in the solution, i.e. all the dots are to the right of the identity line.

The gap between the cloud of dots and the bound makes one wonder if a tighter bound is possible.

4.3 Perform GE(PP) by hand on the matrix

```
[ 1  0  0  0  1 ]
[-1  1  0  0  1 ]
[-1 -1  1  0  1 ]
[-1 -1 -1  1  1 ]
[-1 -1 -1 -1  1 ]
```

demonstrating that the bound 2^{n-1} on the *growth rate* in GEPP can in fact occur. Show the intermediate result for each "k", not for every single row operation.



Homework 4.
4(a) Exact inverse of order-5 Hilbert matrix

```
# sympy for exact inverse of Hilbert matrix
import sympy as sp
# construct the Hilbert matrix of order n
#one = sp.Rational(1,1)
n=5
a = sp.ones(n)
for i in range(n):
    for j in range(n):
        a[i,j] /= (1+i+j)
display(a)
ainv = a.inv()
ainv
```

$$\begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{bmatrix}$$

$$\begin{bmatrix} 25 & -300 & 1050 & -1400 & 630 \\ -300 & 4800 & -18900 & 26880 & -12600 \\ 1050 & -18900 & 79380 & -117600 & 56700 \\ -1400 & 26880 & -117600 & 179200 & -88200 \\ 630 & -12600 & 56700 & -88200 & 44100 \end{bmatrix}$$

Homework 4. 4(b) Condition numbers of Hilbert matrices of order 1 through 10

```
import sympy as sp # sympy for exact inverse of Hilbert matrix
import numpy as np
import pandas as pd

kHn = {}
norms = {}
invnorms = {}

results = []
one = sp.Rational(1,1)
for n in range(1,11):
    # construct the Hilbert matrix of order n
    a = sp.ones(n)
    for i in range(n):
        for j in range(n):
            a[i,j] /= (1+i+j)
    ainv = a.inv()
    norm_a    = np.abs(a   ).sum(axis=1).max()
    norm_ainv = np.abs(ainv).sum(axis=1).max()
    ka = norm_a * norm_ainv
    kHn[n] = ka
    norms[n] = norm_a
    invnorms[n] = norm_ainv
    results.append([norm_a,norm_ainv,ka,'{:.1e}'.format(float(ka))])

# use pandas to make a nice display of results
pd.DataFrame(results,columns=['norm(A)','norm($A^{-1}$)','k(A)','k(A)'])

```

	norm(A)	norm(A^{-1})	k(A)	k(A)
0	1	1	1	1.0e+00
1	3/2	18	27	2.7e+01
2	11/6	408	748	7.5e+02
3	25/12	13620	28375	2.8e+04
4	137/60	413280	943656	9.4e+05
5	49/20	11865420	29070279	2.9e+07
6	363/140	379964970	1970389773/2	9.9e+08
7	761/280	12463050600	33872791095	3.4e+10
8	7129/2520	388712223900	2199309082685/2	1.1e+12
9	7381/2520	12071636216640	35357439251992	3.5e+13

Homework 4.

4(c) For each value of n from 1 to 10, construct and use numpy.linalg.solve() (GEPP) to solve 4000 randomly chosen linear systems $H_n x = b$ and compute the largest relative error in your solutions and the error bound provided by Wilkinson's theorem assuming the worst possible growth rate, ρ. (n on the horizontal axis, log error on vertical; a dot for the worst error in your 4000 trials, and a dot for the Wilkinson error bound.) Comment on the results.

```
# build Hilbert matrices
# numpy
import numpy as np
import matplotlib.pyplot as plt

def vnorm(x): return np.abs( x ).max()   # alternatively np.linalg.norm(x,np.inf)
def mnorm(a): return np.abs(a   ).sum(axis=1).max()

print('max rel error  bound')
for n in range(1,11):
    #n = 10
    mach_eps = 2**(-52)
    rho_max = 2**(n-1)
    rel_error_bound = 4*n**2*kHn[n]*rho_max*mach_eps

    a = np.empty((n,n))
    for i in range(n):
        for j in range(n):
            a[i,j] = 1/(1+i+j)

    max_rel_error = 0
    for k in range(10000):
        x = np.random.randn(n)
        b = a@x
        xgepp = np.linalg.solve(a,b)
        error = xgepp - x

        rel_error = vnorm( xgepp - x) / vnorm( x )
        max_rel_error = max(max_rel_error,rel_error)

    print('{:.2e}       {:.2e}'.format( max_rel_error, rel_error_bound) )
    plt.semilogy(n,max_rel_error  ,'ro')
    plt.semilogy(n,rel_error_bound,'bo')
plt.xlabel('$n$ in $H_n$')
plt.ylabel('max rel error (red) and bound (blue)');
```

```
max rel error  bound
0.00e+00       8.88e-16
2.03e-15       1.92e-13
4.26e-14       2.39e-11
1.25e-12       3.23e-9
3.28e-11       3.35e-7
1.06e-09       2.97e-5
2.61e-08       2.74e-3
7.00e-07       2.46e-1
2.52e-05       2.03e+1
6.75e-04       1.61e+3
```

The error bound is seen to be **valid**, but is a **huge over-estimate** of the largest error seen in these experiments: from about 100 times the largest actual for n=2, to 10 million times the large actual for n=10.