Practical computation of QR factorization using Hausholder reflectors...

from last time

$$v = \frac{a_1 - ce_1}{\|a_1 - ce_1\|} = \frac{a_1 - ce_1}{2v^T a_1}$$

as big as possible

Since v is a unit vector then

$$c = \pm \|a_1\|$$

how to choose + or − ?

Make a random matrix to test the algorithm

rescale number in [0,1] to [-1,1]

since its not possible to subtract a number from a matrix the dot means do it element by element.

```
julia> A=2*rand(5,5).-1
5×5 Matrix{Float64}:
  0.841192  -0.473504   -0.8253    -0.939331   0.019696
  0.834307  -0.887395    0.137436   0.706448  -0.605412
  0.448144   0.0331117  -0.194626  -0.349307   0.608106
 -0.617886  -0.584377   -0.979688  -0.69013   -0.427376
 -0.655803  -0.265738    0.41397    0.512138   0.788342
```

entries between -1 and 1 result in a matrix with better conditioning number and less chance to be singular.

```
julia> using LinearAlgebra
```

we need this for the vector norms

make a copy of A and call the copy R because after the reflections is will be R.

```
julia> R=copy(A)
5×5 Matrix{Float64}:
  0.841192  -0.473504   -0.8253    -0.939331   0.019696
  0.834307  -0.887395    0.137436   0.706448  -0.605412
  0.448144   0.0331117  -0.194626  -0.349307   0.608106
 -0.617886  -0.584377   -0.979688  -0.69013   -0.427376
 -0.655803  -0.265738    0.41397    0.512138   0.788342
```

$a_1$

we first set $c = \|a_1\|$

since this column will turn into the vector v used for the reflection we'll copy it to v

```
julia> v=R[:,1]
5-element Vector{Float64}:
 0.8411920461184637
 0.8343070963949759
 0.4481444930694707
-0.617885852020831
-0.6558025649477148

julia> c=norm(v)
1.5544664057276973
```

positive so

$v+ce_1$

we'll select the ± in the next step...

If $v_1 > 0$ then we want $v + ce_1$

$v_1 < 0$ then we want $v - ce_1$

Everyone will have a different matrix, so use + or − as appropriate. This choice is similar in purpose to the pivoting step in Gaussian elimination as the main point of maximizing $\|v \pm ce_1\|$ is to reduce the rounding error.

```
julia> v[1]+=c
2.395658451846161
```

+= operator works like C/C++ it means add c to v[1] and store the result back in v[1].

```
julia> v
5-element Vector{Float64}:
 2.395658451846161
 0.8343070963949759
 0.4481444930694707
-0.617885852020831
-0.6558025649477148
```

this entry is now larger in magnitude than it was before.

now we need to normalize v so its a unit vector

$$v = \frac{a_1 - ce_1}{2v^T a_1} = \frac{a_1 - ce_1}{\|a_1 - ce_1\|}$$

as big as possible

```
julia> v/=norm(v)
5-element Vector{Float64}:
  0.8778226494521889
  0.3057087145497777
  0.16421013017965586
 -0.22640714717156987
 -0.24030067584814124

julia> norm(v)
0.9999999999999999
```

again /= is like C/C++. It means divide v by ||v|| and store the result back in v.

some rounding error, but close enough to unit vector.

Now we need to compute $HR$

## Some aspects of computational efficiency...

SLOW

slow way to compute

```
julia> (I-2*v*v')*R
5×5 Matrix{Float64}:
 -1.55447       0.378573   0.214183      0.171597   0.301672
  5.01104e-18  -0.590653   0.499445      1.09334   -0.507211
  4.5638e-17    0.192506  -0.000174981  -0.141491   0.660854
 -1.07336e-16  -0.804144  -1.24779      -0.97666   -0.500103
 -5.12747e-17  -0.498991   0.129415      0.208025   0.711152
```

zeros

### Computational efficiency

$H$

mult it out

$$(I - 2vv^T)R$$

$5\times5$ $\quad$ $5\times5$ $\quad$ $5\times5$

matrix-matrix mult ($n^3$ operations)

slow

$v \in \mathbb{R}^5$

$v \in \mathbb{R}^{5\times1}$

$v^T \in \mathbb{R}^{1\times5}$

$vv^T \in \mathbb{R}^{5\times5}$

$5\times1 \quad 1\times5$

FAST

```
julia> R-2*v*(v'*R)
5×5 Matrix{Float64}:
 -1.55447   0.378573   0.214183      0.171597   0.301672
  0.0      -0.590653   0.499445      1.09334   -0.507211
  0.0       0.192506  -0.000174981  -0.141491   0.660854
  0.0      -0.804144  -1.24779      -0.97666   -0.500103
  0.0      -0.498991   0.129415      0.208025   0.711152
```

$$= R - 2v(v^T R)$$

$5\times1 \quad 1\times5 \quad 5\times5$

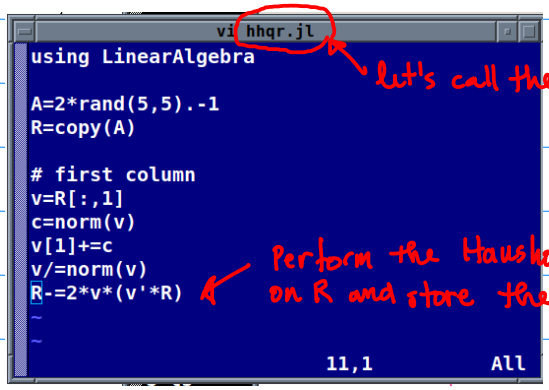
matrix vector mult

vector-vector mult (about $n^2$ operations)

fast

(action of applying $H$)

Now lets copy the code into the editor...




let's call the name of the file hhqr.jl.

Perform the Householder reflection on R and store the result back in R.

Assuming Julia and the editor are running from the same working directory on can type this to run the script.

```
julia> include("hhqr.jl")
ERROR: SystemError: opening file "/x/libb/ejolson/hhqr.jl": No such file or dire
ctory
Stacktrace:
 [1] include(fname::String)
   @ Base.MainInclude ./client.jl:444
 [2] top-level scope
   @ REPL[14]:1
```

oops. it wasn't in the same directory...

fix this...

```
julia> pwd()
"/x/libb/ejolson"

julia> cd("teach/466/2021")

julia> pwd()
"/x/libb/ejolson/teach/466/2021"
```

current directory

change directory

new directory

```
julia> readdir()
30-element Vector{String}:
 "2021-10-02-Note-15-04.xoj"
 "HW1soln.pdf"
 "alpine"
 "aug24"
 "aug26"
 "delmeyer"
 "grades"
 "hw1"
 "ipxe"
 "ipxe.tgz"
 "kahan"
 "mara"
 "numerical_book.pdf"
 "numerical_book.pdf.xoj"
 "oct05"
 "old"
 "prog1"
```

list files in current directory

that's the one

```
julia> cd("oct05")

julia> include("hhqr.jl")
5×5 Matrix{Float64}:
 -1.1303       -0.415771    0.702622    0.386426    0.461906
 -6.93889e-18   0.283918    0.0453595   0.318504   -0.319127
  0.0          -0.257591    0.139863    1.01127    -0.0817933
 -5.55112e-17  -0.0460272   0.434056    0.113025   -0.865745
  1.11022e-16   0.0434425  -0.100318   -0.272793    0.322678
```

change to the correct directory

run the script

zeros

4x4 submatrix

perform the same steps again on the submatrix to make more zeros

get the second column

```
julia> v=R[:,2]
5-element Vector{Float64}:
 -0.41577133161232416
  0.2839178869292594
 -0.2575905424568121
 -0.04602715356150977
  0.043442535748091926

julia> v[1]=0
0

julia> v
5-element Vector{Float64}:
  0.0
  0.2839178869292594
 -0.2575905424568121
 -0.04602715356150977
  0.043442535748091926
```

zero out the first element, because we want to work with the submatrix

zero

positive so we add c
if negative, subtract

The decision whether to add or subtract c can be made with an if statement

Make the reflector and apply it ...

```
julia> c=norm(v)
0.3885460163980188

julia> v[2]+=c
0.6724639033272781

julia> v/=norm(v)
5-element Vector{Float64}:
  0.0
  0.930246962048501
 -0.3563355867686686
 -0.06367125366950158
  0.06009584559615814

julia> R-=2*v*(v'*R)
5×5 Matrix{Float64}:
 -1.1303        -0.415771   0.702622    0.386426   0.461906
 -1.39186e-17   -0.388546   0.122214    0.481583   0.0403326
  2.67363e-18    0.0        0.110424    0.948801  -0.219486
 -5.50334e-17    0.0        0.428796    0.101863  -0.890348
  1.10571e-16    0.0       -0.0953529  -0.262258   0.3459
```

alternative use if

```
julia> if v[2]>0
           v[2]+=c
       else
           v[2]-=c
       end
```

create unit vector

apply reflection

old zeros are still here

new zeros.

Note ... any number about $10^{-16}$ or so is equivalent to zero up to the limits of rounding error in double precision floating point arithmetic (ie. 15 significant digits).

Add these lines to the script file. Note, we will eventually make a loop to process each column automatically. That will be easier after some more experiments within the interactive environment.

New script

```
vi hhqr.jl
using LinearAlgebra

A=2*rand(5,5).-1
R=copy(A)

# first column
v=R[:,1]
c=norm(v)
if v[1]>0
   v[1]+=c
else
   v[1]-=c
end
v/=norm(v)
R-=2*v*(v'*R
)

# second column
v=R[:,2]
v[1]=0
c=norm(v)
if v[2]>0
   v[2]+=c
else
   v[2]-=c
end
v/=norm(v)
                              15,82         Top
```

if statement

if statement

oops. cut & paste error...

→ run script 3 times to check if it is working well...

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
 -1.53198      1.24882    0.276743  -0.391936  -0.835869
 -2.99653e-17  0.548896   0.989889   0.34032    0.304418
  7.44276e-17  0.0        0.47839   -0.81212    0.452227
 -1.46449e-16  0.0       -0.572987   0.900289   0.785544
  9.63619e-17  0.0       -0.396366   0.272941   0.173334

julia> include("hhqr.jl")
5×5 Matrix{Float64}:
 -1.17452      -0.338342  -0.73096    0.614578   0.254891
 -9.05461e-17  -0.889291   0.192138   0.51123   -0.362883
 -1.7664e-16    0.0       -0.531694  -0.198681  -0.583986
 -3.97595e-18   0.0       -0.437382  -0.290928  -0.926389
  1.49687e-16   0.0       -0.585853   0.927334   0.419144

julia> include("hhqr.jl")
5×5 Matrix{Float64}:
  1.15863      -0.331605   0.0180854   0.241246    0.69679
  7.64156e-17  -1.53248    0.866872   -1.27723     0.477329
  9.55314e-17   0.0        0.582135   -0.128258    0.191391
 -2.62863e-17   0.0       -0.150933    0.0772497   0.33839
  7.69016e-17   0.0       -0.199311    0.241939   -0.273387
```

zeros each time ... looks good!

Add lines for 3rd column...

```
# third column
v=R[:,3]
for i=1:2
    v[i]=0
end
c=norm(v)
if v[3]>0
   v[3]+=c
else
   v[3]-=c
end
v/=norm(v)
R-=2*v*(v'*R)
```

zero out the first two elements of v

clever alternative

```
# third column
v=R[:,3]
v[1:2].=0
c=norm(v)
if v[3]>0
   v[3]+=c
else
   v[3]-=c
end
v/=norm(v)
R-=2*v*(v'*R)
```

use a vector operation to zero the first two elements of v.

Advantage of Julia over Python, R or Matlab is that if you can't figure out a vector operation or subroutine call to perform an operation on can use a loop without loosing any performance...

This is because loops are fast in a just-in-time compiled language like Julia, while loops are slow in interpreted languages such as Python, R and Matlab.

note .= means work element by element, so the scalar zero on the right can be assigned to the 2-vector on the left.

Running the new script looks good:

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
  0.46845       -0.0982872      0.245758   -0.83973    -1.13901
  2.38287e-17    1.06862       -0.65371    -0.575581    0.350199
  5.24805e-18    1.45207e-16    0.955817   -0.0975613  -0.0446049
 -2.7289e-18    -1.04514e-17    0.0        -1.1829      0.721533
  5.90746e-17   -6.04158e-17    0.0        -0.147952    0.16539
```

more zeros in the third column

We could finish by adding more code for the column 4. Instead, let's make a loop and delete code...

First save a copy of the working script:

backup script is called hhqrscript.jl

that way if we introduce errors while adding the loop, it's possible to go back and see what went wrong.

```
julia> size(R)
(5, 5)

julia> size(R)[1]
5

julia> size(R)[2]
5
```

The dimensions of a matrix

rows

columns

We loop on the number of columns (minus the last one).

When pasted with the mouse the script works as expected...

New script

```
vi hhqr.jl
using LinearAlgebra

A=2*rand(5,5).-1
R=copy(A)

m=size(R)[1]
n=size(R)[2]

for j=1:n-1
    v=R[:,j]
    v[1:j-1].=0
    c=norm(v)
    if v[j]>0
       v[j]+=c
    else
       v[j]-=c
    end
    v/=norm(v)
    R-=2*v*(v'*R)
    display(R)
end
~
~
~
~
~
-- INSERT --        21,4          All
```

display R after each reflection.

```
5×5 Matrix{Float64}:
 1.55414    0.267671   -0.255659    0.627321    0.612509
 0.0        0.581809    0.660325    0.204566   -0.736143
 0.0       -0.483305   -0.568653   -0.284275   -0.485823
 0.0       -0.406646    0.836453    0.53924    -0.0238721
 0.0        0.584323    0.934285   -0.060292   -0.225137
5×5 Matrix{Float64}:
 1.55414    0.267671     -0.255659    0.627321     0.612509
 0.0       -1.03869      -0.832587   -0.00182922   0.303593
 0.0        5.55112e-17  -0.123401   -0.222718    -0.795918
 0.0        1.11022e-16   1.21108     0.591033    -0.284781
 0.0       -1.11022e-16   0.395968   -0.134714     0.149773
5×5 Matrix{Float64}:
 1.55414    0.267671     -0.255659     0.627321     0.612509
 0.0       -1.03869      -0.832587    -0.00182922   0.303593
 0.0        6.53415e-17   1.28013      0.538952    -0.146369
 0.0        1.0254e-16   -2.22045e-16 -0.0661987   -0.845265
 0.0       -1.13796e-16  -5.55112e-17  -0.349599    -0.0334797
5×5 Matrix{Float64}:
 1.55414    0.267671     -0.255659     0.627321     0.612509
 0.0       -1.03869      -0.832587    -0.00182922   0.303593
 0.0        6.53415e-17   1.28013      0.538952    -0.146369
 0.0        9.27313e-17   9.58534e-17  0.355811     0.190157
 0.0       -1.21921e-16   2.0784e-16   5.55112e-17  0.824278
```

But, weirdly, when run as a script is full of errors... this is one of the many things about Julia which is astonishing and causes confusion...

```
julia> include("hhqr.jl")
┌ Warning: Assignment to `v` in soft scope is ambiguous because a global variable by the same name exists: `v` will be treated as a new local. Disambiguate by using `local v` to suppress this warning or `global v` to assign to the existing global variable.
└ @ ~/teach/466/2021/oct05/hhqr.jl:10
┌ Warning: Assignment to `c` in soft scope is ambiguous because a global variable by the same name exists: `c` will be treated as a new local. Disambiguate by using `local c` to suppress this warning or `global c` to assign to the existing global variable.
└ @ ~/teach/466/2021/oct05/hhqr.jl:12
┌ Warning: Assignment to `R` in soft scope is ambiguous because a global variable by the same name exists: `R` will be treated as a new local. Disambiguate by using `local R` to suppress this warning or `global R` to assign to the existing global variable.
└ @ ~/teach/466/2021/oct05/hhqr.jl:19
ERROR: LoadError: UndefVarError: R not defined
Stacktrace:
 [1] top-level scope
   @ ~/teach/466/2021/oct05/hhqr.jl:10
 [2] include(fname::String)
   @ Base.MainInclude ./client.jl:444
 [3] top-level scope
   @ REPL[49]:1
in expression starting at /x/libb/ejolson/teach/466/2021/oct05/hhqr.jl:9
```

fix these errors using global statement

Revised script

```
vi hhqr.jl
using LinearAlgebra

A=2*rand(5,5).-1
R=copy(A)

m=size(R)[1]
n=size(R)[2]

for j=1:n-1
    global v,c,R
    v=R[:,j]
    v[1:j-1].=0
    c=norm(v)
    if v[j]>0
       v[j]+=c
    else
       v[j]-=c
    end
    v/=norm(v)
    R-=2*v*(v'*R)
    display(R)
end
```

global variables

Now it works as a script as well

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
  1.75185      -1.23869     0.404474  -0.391099  -0.330307
 -2.22045e-16  -0.310436   -0.2798    -1.30203    0.18935
 -2.77556e-17   0.133799   -0.542535   0.274895  -0.415223
 -2.22045e-16   0.227471    0.131951   0.53638    1.09161
  2.22045e-16  -0.269576    0.730269   0.59906   -0.712637
5×5 Matrix{Float64}:
  1.75185      -1.23869       0.404474  -0.391099   -0.330307
 -9.24145e-17   0.488556     -0.312305   0.821799    0.667441
 -4.94634e-17   0.0          -0.537091  -0.0807605  -0.495284
 -2.5895e-16    5.55112e-17   0.141205  -0.0682668   0.955501
  2.65781e-16  -5.55112e-17   0.719302   1.31563    -0.551332
5×5 Matrix{Float64}:
  1.75185      -1.23869       0.404474     -0.391099  -0.330307
 -9.24145e-17   0.488556     -0.312305      0.821799   0.667441
  1.99374e-16  -3.53137e-17   0.908736      1.0785     0.0047986
 -2.83252e-16   5.896e-17     2.77556e-17  -0.181485   0.906661
  1.41984e-16  -3.79425e-17   1.11022e-16   0.738895  -0.800124
5×5 Matrix{Float64}:
  1.75185      -1.23869       0.404474     -0.391099     -0.330307
 -9.24145e-17   0.488556     -0.312305      0.821799      0.667441
  1.99374e-16  -3.53137e-17   0.908736      1.0785        0.0047986
  2.05449e-16  -5.09109e-17   1.01197e-16   0.760857     -0.993292
 -2.4121e-16    4.82079e-17   5.34362e-17   3.33067e-16   0.68964
```

The reason for the difference appears an intention change in standards of how carefully the scope of a variable needs to be specified when running a script compared to interactive...

To finish, we need to store the vectors $v$ as the loop progresses so the reflections $H = I - 2vv^T$ are available later to form the $Q$ matrix. In particular

$j=1$ $\quad H_1 = I - 2vv^T$

$j=2$ $\quad H_2 = I - 2vv^T$

$j=3$ $\quad H_3 = I - 2vv^T$

$j=4$ $\quad H_4 = I - 2vv^T$

$$\underbrace{H_4 H_3 H_2 H_1}_{Q^T} A = R$$

mult by $Q$

$$Q^T A = R$$

$$A = QR$$

$$H = I - 2vv^T$$

$$H^{-1} = H^T = (I - 2vv^T)^T = I^T - 2(vv^T)^T = I - 2v^{TT}v^T = H$$

$$H_4 \quad H_4 H_3 H_2 H_1 A = H_4 R$$

$$A = H_1 H_2 H_3 H_4 R$$

$$Q = H_1 H_2 H_3 H_4$$

# Create an array H to store things in...

```
julia> H=zeros(5,4)
5×4 Matrix{Float64}:
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
 0.0  0.0  0.0  0.0
```

the zeros function makes Matrices full of zeros...

New script

```
vi hhqr.jl
using LinearAlgebra

A=2*rand(5,5).-1
R=copy(A)

m=size(R)[1]
n=size(R)[2]
H=zeros(m,n-1)

for j=1:n-1
    global v,c,R,H
    v=R[:,j]
    v[1:j-1].=0
    c=norm(v)
    if v[j]>0
       v[j]+=c
    else
       v[j]-=c
    end
    v/=norm(v)
    R-=2*v*(v'*R)
    H[:,j]=v
end
display(R)
display(H)
~
                    23,3          All
```

create H

add H to global

save v in H

move the display outside the loop.

Output

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
  1.19855      -0.00192657  -0.492995  -1.0266      0.882446
  2.54091e-16   1.46317     -0.352017  -0.0565976  -0.784219
 -1.58322e-16   0.0         -1.30166    0.697113   -0.0580223
  6.09562e-17   0.0          0.0        1.14036     1.3976
 -2.81546e-16   0.0          0.0        0.0         0.163108
5×4 Matrix{Float64}:
 -0.895023   0.0         0.0        0.0
 -0.175669  -0.88476     0.0        0.0
  0.340017   0.0793833   0.947726   0.0
  0.228867   0.376032   -0.162621  -0.931791
 -0.009051   0.263626    0.274536  -0.362995
```

$H_1$ $H_2$ $H_3$ $H_4$

Vectors for Hausholder transforms...

Finally we check that $A=QR$ by creating a function that applies

$$H_1 H_2 \cdots H_{n-1}$$

← right to left like function composition

to any matrix.

```
vi hhqr.jl
using LinearAlgebra

A=2*rand(5,5).-1
R=copy(A)

m=size(R)[1]
n=size(R)[2]
H=zeros(m,n-1)

function applyQ(M)
    global n,H
    for j=n-1:-1:1
        v=H[:,j]
        M-=2*v*(v'*M)
    end
    return M
end

for j=1:n-1
    global v,c,R,H
    v=R[:,j]
    v[1:j-1].=0
    c=norm(v)
    if v[j]>0
       v[j]+=c
    else
       v[j]-=c
    end
    v/=norm(v)
    R-=2*v*(v'*R)
    H[:,j]=v
end
display(R)
display(H)
println("norm(A-QR)=",
   norm(A-applyQ(R)))
~
                34,10      All
```

new function

display the error in QR.

```
function applyQ(M)
    global n,H
    for j=n-1:-1:1
        v=H[:,j]
        M-=2*v*(v'*M)
    end
    return M
end
```

run the loop backwards.

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
 -0.832088     0.261539      0.340683   -0.464379    0.0625118
 -1.54102e-17  1.24954      -0.624489   -0.547975   -0.621159
 -1.39431e-16  5.20555e-17   0.587335    0.157306    0.293371
  7.4269e-17   1.48102e-16   0.0        -0.806119    0.424095
 -2.05029e-17  2.81377e-18   0.0         0.0         0.211033
5×4 Matrix{Float64}:
  0.713724    0.0          0.0        0.0
 -0.0836586  -0.776493     0.0        0.0
 -0.0877523   0.0770885   -0.72565    0.0
  0.604369   -0.393777     0.2974     0.970782
 -0.332621   -0.485856    -0.620472   0.239964
norm(A-QR)=1.2605556464031413e-15
```

should be exactly zero

← error

The output looks reasonable and the error again good to about 15 significant digits.

Before finishing... we tidy things up by setting the terms in R that were supposed to be zero exactly to zero...

vi hhqr.jl

```
using LinearAlgebra
A=2*rand(5,5).-1
R=copy(A)
m=size(R)[1]
n=size(R)[2]
H=zeros(m,n-1)

function applyQ(M)
    global n,H
    for j=n-1:-1:1
        v=H[:,j]
        M-=2*v*(v'*M)
    end
    return M
end

for j=1:n-1
    global v,c,R,H
    v=R[:,j]
    v[1:j-1].=0
    c=norm(v)
    if v[j]>0
       v[j]+=c
    else
       v[j]-=c
    end
    v/=norm(v)
    R-=2*v*(v'*R)
    R[j+1:m,j].=0
    H[:,j]=v
end
display(R)
display(H)
println("norm(A-QR)=",
   norm(A-applyQ(R)))
~
                6,1           All
```

set R to have exact zeros

Output looks nicer

```
julia> include("hhqr.jl")
5×5 Matrix{Float64}:
 -1.36041   0.378358  0.666071   0.242084  -0.494502
  0.0      -1.05609   0.597951  -0.570969  -0.737176
  0.0       0.0       0.544788   0.158028   0.991906
  0.0       0.0       0.0        1.2126     0.0765811
  0.0       0.0       0.0        0.0       -0.320643
5×4 Matrix{Float64}:
  0.765446   0.0          0.0         0.0
 -0.234632   0.775063     0.0         0.0
 -0.383769   0.628108    -0.758064    0.0
  0.445517  -0.0437555    0.406668   -0.981739
  0.115224   0.0533137    0.509864    0.190232
norm(A-QR)=1.8507475262965 23e-15
```

now exact zero

error about the same

In a practical program one might use a sparse matrix representation that doesn't even store the zeros in computer memory to save space...

In this case the savings would be less than 50% but some other problems may involve matrices with hundred-fold as many zeros as non-zero entries. Then the savings would be more.