Math 578 HW$^{\#}$2

Luke Wukmer

Fall 2016

Problem 1. QR via Givens rotations

(a) An algorithm for QR decomposition of a tridiagonal matrix via Givens rotations is given in `givens.py` and **Appendix A**.

(b) Flop count for forming R within Givens QR, using tridiagonal input A:

(a) 6 flops per computing components c, s of Givens rotation

(b) 1 application of 2×2 Givens matrix by 2×3 subblock. Can calculate as 3 matrix-vector multiplications or $3(2mn - 2)$ where $m, n = 2$. Or direct counting $\rightarrow 6 \cdot 3 = 18$ flops.

(c) (a)-(b) are repeated for each diagonal element except for the last two, so $(n-2)$ times. On the second-to-last diagonal element, the subblock to apply the Givens matrix to is only 2×2, instead of 2×3. Using the logic above, this particular iteration requires $2 \cdot 6$ iterations (plus the 6 in part (a)).

(d) For the very last diagonal element, nothing changes.

Therefore, total flop count is

$$\begin{aligned}(6 \cdot 3 + 6) \cdot (n-2) + (6 \cdot 2 + 6) &= 24(n-2) + 18 \\ &= \mathcal{O}(n)\end{aligned}$$

(c) The matrix begins as tridiagonal. Givens rotations only will interact with nonzero columns of the $2 \times n$ subblock relevant to each diagonal, which is only the first three. Thus, the only element above the first superdiagonal which may be influence by our QR algorithm is the single entry of the 2nd superdiagonal, which appears as the upper-right element of each 2x3 subblock. Of course, our QR decomposition will make R upper triangular, so all subdiagonals will be zeroed. Therefore, R will in general have only nonzero elements on its main diagonal, as well as its first and second superdiagonals.

Problem 2. Solving Least Squares Problems with QR. Relavant code to this problem is located in `lsq_demo.py` and in the **Appendix**.

(a) Backsubsitution to solve a upper triangular system $Rx = b$ is implemented in the function `lsq_demo.back_substitution(...)`.

(b) The above method was used to solve the linear systems described in Problem 2. These systems were generated using `A,b = lsq_demo.demo_system(n)` for $n = 5, 6, \ldots, 25$. Using back-substitution only (since A as described is upper-triangular), we calculate the relative residual error of our solution x via

$$e_n := \frac{\|Ax - b\|_2}{\|b\|_2}$$

(c) We solve the same system by finding the solution x to $A^T Ax = A^T b$. This is found using Givens QR, as the system $A^T A$ is tridiagonal. We then calculate the relative residual error of *this* solution $\widetilde{x}$ as:

$$f_n := \frac{\|A\widetilde{x} - b\|_2}{\|b\|_2}$$

(d) Upon running, `lsq_demo.py` generated the following output:

```
n       e_n (backsub)           f_n (normal eqs)
------------------------------------------------
5       3.43990022796e-16       2.40844199509e-14
6       3.62597321469e-16       9.21161907519e-14
7       3.75323885184e-16       3.44218058155e-13
8       3.84592537277e-16       1.29128889592e-12
9       3.9164986973e-16        4.89224891546e-12
10      3.9720546452e-16        1.85749088174e-11
11      4.01693804323e-16       7.08914229515e-11
12      4.05396129633e-16       2.71605924855e-10
13      4.08502658814e-16       1.04400239547e-09
14      4.11146716571e-16       4.02453325848e-09
15      4.13424555146e-16       1.55530155125e-08
16      4.15407418106e-16       6.02382571746e-08
17      4.17149175925e-16       2.33763354402e-07
18      4.18691322316e-16       9.08711126433e-07
19      4.2006633857e-16        3.53787365819e-06
20      4.21300016229e-16       1.37928389734e-05
21      4.22413096151e-16       5.38360259895e-05
22      4.2342244788e-16        0.000210303218642
23      4.24341933106e-16       0.000821319295305
24      4.25183047773e-16       0.00319431323358
25      4.25955406352e-16       0.0121887295644

esimated slope of line: 1.34122414003
```

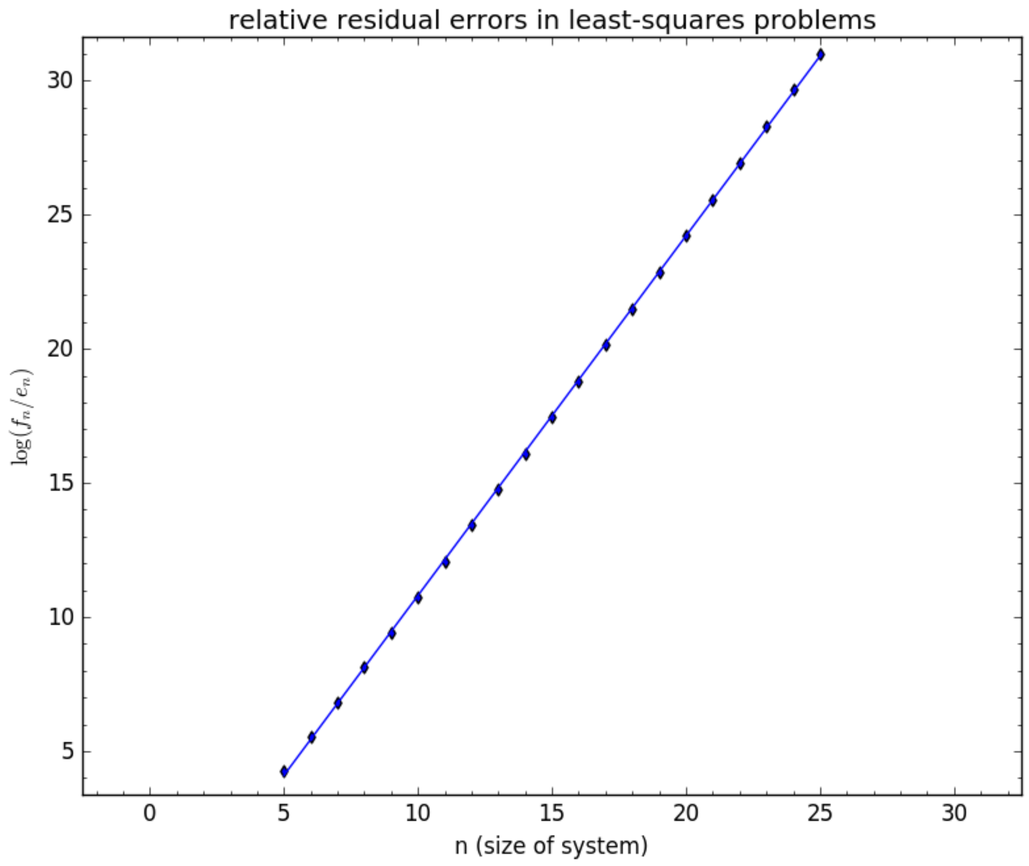

Figure 1: A plot of relative errors of the least squares solutions found via back-substitution (with relative residual e_n and via QR of normal equations (with relative residual f_n.

These points are plotted as the log of the ratio f_n/e_n against n in **Figure 1**.

The slope of our fitted line is approximately 1.3412, and thus we conclude

$$\frac{f_n}{e_n} \propto C^n \text{ with } \log(C) \approx 1.3412 \Rightarrow C \approx 3.823$$

(e) We have the calculation (using results given on the hw spec)

$$\begin{aligned} \|Ax-b\|_2 &= \|Ax-(A+\delta A)x\|_2 \\ &= \|(-\delta A)x\|_2 = \|\delta A\|_2 \approx \left\|(\delta A)A^{-1}b\right\|_2 \end{aligned}$$

and so

$$\begin{aligned} e_n := \frac{\|Ax-b\|_2}{\|b\|_2} \approx \frac{\|(\delta A)A^{-1}b\|_2}{\|b\|_2} \leq \frac{\|\delta A\|_2 \|A^{-1}\|_2 \|b\|}{\|b\|_2} &= \|\delta A\|_2 \left\|A^{-1}\right\|_2 \\ &\approx \epsilon_M \|A\|_2 \left\|A^{-1}\right\|_2 \\ &= \epsilon_M \kappa_2(A) \end{aligned}$$

$$\Longrightarrow \qquad \boxed{e_n \approx \epsilon_M \kappa_2(A)}$$

(f) We have the calculation (all norms are 2-norms)

$$\begin{aligned}\|Ax-b\| &= \left\|A^{-T}A^{T}(Ax-b)\right\| \\ &= \left\|A^{-T}\left(A^{T}Ax-A^{T}b\right)\right\| \\ &= \left\|A^{-T}\left(A^{T}Ax-\left(A^{T}A+\delta A\right)x\right)\right\| \\ &= \left\|A^{-T}(\delta A)\,x\right\|\end{aligned}$$

and so

$$\begin{aligned}f_n = \frac{\|Ax-b\|}{\|b\|} &= \frac{\left\|A^{-T}(\delta A)\,x\right\|}{\|b\|} \\ &\approx \frac{\left\|A^{-T}\left((\delta A)\left(A^{T}A\right)^{-1}A^{T}b\right)\right\|}{\|b\|} \\ &\leq \frac{\left\|A^{-T}\right\|\,\|\delta A\|\,\left\|\left(A^{T}A\right)^{-1}\right\|\,\left\|A^{T}\right\|\,\|b\|}{\|b\|} \\ &= \left\|A^{-T}\right\|\,\left\|A^{T}\right\|\,\|\delta A\|\,\left\|\left(A^{T}A\right)^{-1}\right\| \\ &= \kappa(A^{T})\,\|\delta A\|\,\left\|\left(A^{T}A\right)^{-1}\right\| \\ &\approx \kappa(A^{T})\left(\epsilon_M\left\|A^{T}A\right\|\right)\left\|\left(A^{T}A\right)^{-1}\right\| \\ &= \epsilon_M\cdot\kappa\left(A^{T}\right)\kappa\left(A^{T}A\right) \\ &= \epsilon_M\left[\kappa(A)\right]^{3}\end{aligned}$$

$$\implies \qquad \boxed{f_n \approx \epsilon_M\left[\kappa_2(A)\right]^{3}}$$

(g) Using parts **(e)**,**(f)**, we can estimate

$$\frac{f_n}{e_n} \approx \frac{\epsilon_M\left[\kappa_2(A)\right]^{3}}{\epsilon_M\kappa_2(A)} = \left[\kappa_2(A)\right]^{2}$$

Using the estimate $\kappa_2(A) \approx 2^n$ for the particular system of size n we generated in part **(b)**, this approximation is $2^{(2n)}$. In the language of part **(d)**, this implies $C \approx 4$, as compared to the estimate of 3.8 found by our program. This is absolutely satisfactory given the multiple layers of approximation used to calculate our theoretical result, as well as the fitting of our line.

---

1 Appendix

../givens.py

```
#!/usr/bin/env python3

"""
USED FOR HW 2 PROBLEM 2
MATH 578 Luke Wukmer Fall 2016
blah blah blah
"""

import numpy as np

def tridiagonal(A):
    """
    A must be square. or an integer
    returns a tridiagonal matrix (ones and zeros) if input is an
    integer. if input is a (dense) matrix,
    then zero out what is not on the tridiagonal
    """
    try:
        N = int(A)
    except TypeError:
        N = A.shape[0]
    else:
        A = np.ones((N,N))

    trid = np.eye(N) + np.eye(N,k=1) + np.eye(N,k=-1)

    return trid*A


def givens(x):
    """
    returns the components c,s of the rotation matrix

    G = ( c   -s )
        ( s    c )

    such that the 2x1 input x = [a b]^T would be rotated to align with e_1:

    G x = [ ~, 0]^T for some number ~
    """

    a, b = x

    # this equals 2**(-52) exactly when x is float64
    eps = np.finfo(x.dtype).eps

    if np.abs(a) < eps:
        c, s = 1, 0
    elif np.abs(b) > np.abs(a):
        tau = -a/b

        s = 1 / ( np.sqrt(1+tau*tau))
        c = tau*s
    else:
        tau = -b/a
        c = 1 / ( np.sqrt(1+tau*tau))
        s = tau*c

    return c,s

def givens_qr(A):
    """ applies givens QR"""

    # A needs to be at least float64 dtype
    A = A.astype('float64')
    R = A.copy()
    n = A.shape[0]
    V = np.zeros((n-1,2))

    # for each element on diagonal except for last
    G = lambda c, s : np.array([[c,-s],[s,c]])
    for k in range(n-1):
        x = R[k:k+2,k] # diagonal and entry below

        g  = givens(x) # givens coefficients

        V[k,:] = g
        # apply to whole subblock
        #R[k:k+2,:] = G(*g) @ R[k:k+2,:]

        # apply to 2x3 subblock (assuming tridiagonal)
        R[k:k+2,k:k+3] = G(*g) @ R[k:k+2,k:k+3]

    # now obtain Q by backwards accumulation
    Q = np.eye(n)

    for k in range(n-2,-1,-1):
        c,s = V[k,:]
        qk = np.eye(n)
        qk[k:k+2,k:k+2] = G(c,s)
        Q = qk.T @ Q
    return Q,R

if __name__ == "__main__":

    from numpy.linalg import qr

    np.set_printoptions(precision=3)

    A = 100*np.random.random((10,10))
    A = tridiagonal(A)
    Q,R = givens_qr(A)
```

../lsq_demo.py

```
#!/usr/bin/env python3
import numpy as np
from numpy.linalg import norm

def demo_system(n):
    """
    as described in problem 2(b)
    """
    A = np.eye(n) - 2*np.eye(n,k=1)
    b = np.ones(n) + np.sqrt(6)*(2**-52)

    b = np.expand_dims(b,-1)
    return A, b

def back_substitution(R,b):
    """
    Solves a system Rx=b via backsubstitution where R is an nxn upper-triangular
    matrix
    """
    n = R.shape[0]

    x = np.zeros(b.shape)

    x[-1] = b[-1] / R[-1,-1] # last entry to initialize
    for i in range(n-1,-1,-1):
        x[i] = (b[i] - R[i,i+1:]@x[i+1:]) /R[i,i]

    return x

def relative_residual(A,x,b):

    return norm(A@x-b,2) / norm(b,2)

if __name__ == "__main__":

    from qr import householder_qr
    from givens import givens_qr
    from numpy.linalg import qr

    import matplotlib.pyplot as plt

    print('n', 'e_n (backsub)', 'f_n (normal eqs)', sep='\t')
    print('-'*80)

    Ns = np.arange(5,26)

    errs = list()

    for n in Ns:

        A,b = demo_system(n)

        # backsub only -> e_n
        #Q,R = givens_qr(A)
        x = back_substitution(A,b)
        e = relative_residual(A,x,b)

        # normal_eqs -> f_n
        N = A.T @ A
        c = A.T @ b
        Q,R = givens_qr(N)
        #Q, R = qr(N)
        x = back_substitution(R,Q.T@c)
        f = relative_residual(A,x,b)


        #     # builtin -> g_n
        #     Q,R = qr(A)
        #     x = back_substitution(R,Q.T@b)
        #     g = relative_residual(A,x,b)
        #
        #
        #     # householder -> h_n
        #
        #     Q,R = householder_qr(A)
        #     x = back_substitution(R,Q.T@b)
        #     h = relative_residual(A,x,b)
        #
        #     print(n, e, f, g, h, sep='\t')

        print(n,e,f, sep='\t')
        errs.append((e,f))

    C = np.log([ f / e for e,f in errs])

    plt.scatter(Ns, C, marker='d')

    m, yint = np.polyfit(Ns,C,1)
    xs = np.linspace(5,25)

    plt.plot(xs,m*xs + yint)

    plt.title('relative residual errors in least-squares problems')
    plt.xlabel('n (size of system)')
    plt.minorticks_on()
    plt.ylabel('$\log(f_n/e_n)$')
    plt.axis('equal')
    plt.tight_layout()

    print('esimated slope of line:', m)

    plt.show()
```