Sample Solutions of Programming assignment of Chapter 2

Yu-Rong Wang and Cheng-Hsin Hsu

Note that, the solutions are for your reference only. If you have any doubts about the correctness of the answers, please let the instructor and the TA know. More importantly, like other math questions, the homework questions may be solved in various ways. Do not assume the sample solutions here are the only *correct* answers; discuss with others about alternate solutions.

We will not grade your homework assignment, but you are highly encouraged to discuss with us during the Lab hours. The correlation between the homework assignments and quiz/midterm/final questions is high. So you do want to practice more and sooner.

1 Computer Problem

- **2.11**

(a)

```
function cp02_11(n, times) % pivoting strategies for Gaussian
   elimination
disp('e_norm_np␣␣e_norm_pp␣␣e_norm_cp␣␣r_norm_np␣␣r_norm_pp␣␣
   r_norm_cp␣␣t_norm_np␣␣t_norm_npp␣t_norm_cp')
x_times = [1:times]; e_norm_np = zeros(times, 1); e_norm_pp = zeros
   (times, 1);
e_norm_cp = zeros(times, 1); r_norm_np = zeros(times, 1); r_norm_pp
    = zeros(times, 1);
r_norm_cp = zeros(times, 1); t_norm_np = zeros(times, 1); t_norm_pp
    = zeros(times, 1);
t_norm_cp = zeros(times, 1);
for t = 1:times
A = rand(n,n); x_true = ones(n,1); b = A*x_true;
[x_np, t_norm_np(t)] = GaussElim_np(A, b);
e_norm_np(t) = norm(x_np-x_true,inf); r_norm_np(t) = norm(b-A*x_np,
   inf);
[x_pp, t_norm_pp(t)] = GaussElim_pp(A,b);
e_norm_pp(t) = norm(x_pp-x_true,inf); r_norm_pp(t) = norm(b-A*x_pp,
   inf);
[x_cp, t_norm_cp(t)] = GaussElim_cp(A,b);
e_norm_cp(t) = norm(x_cp-x_true,inf); r_norm_cp(t) = norm(b-A*x_cp,
   inf);
```

```
fprintf('%9.4e␣%9.4e␣%9.4e␣%9.4e␣%9.4e␣%9.4e␣%9.4e␣%9.4e␣%9.4e\n',
    ...
e_norm_np(t), e_norm_pp(t), e_norm_cp(t), r_norm_np(t), r_norm_pp(t
   ), r_norm_cp(t), t_norm_np(t), t_norm_pp(t), t_norm_cp(t))
end
% plot errors
figure(1)
plot(x_times', e_norm_np,'g-', x_times', e_norm_pp, 'b--', x_times
   ', e_norm_cp, 'r:');
xlabel('times'); ylabel('error');
legend('no␣pivoting␣error', 'partial␣pivoting␣error', 'complete␣
   pivoting␣error')
% plot residuals
figure(2)
plot(x_times', r_norm_np,'g-', x_times', r_norm_pp, 'b--', x_times
   ', r_norm_cp, 'r:');
xlabel('times'); ylabel('residual');
legend('no␣pivoting␣residual', 'partial␣pivoting␣residual', '
   complete␣pivoting␣residual')
% plot time
figure(3)
plot(x_times', t_norm_np,'g-', x_times', t_norm_pp, 'b--', x_times
   ', t_norm_cp, 'r:');
xlabel('times'); ylabel('time(s)');
legend('no␣pivoting␣time', 'partial␣pivoting␣time', 'complete␣
   pivoting␣time')
function [x, t] = GaussElim_np(A, b) % Gaussian elimination
   without pivoting
timer = tic;
n = size(A,1); x = zeros(n,1);
for i=1:n
p = A(i,i);
for j=(i+1):n
A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
b(j) = b(j)-b(i)*(A(j,i)/p); % forward substitution
end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1 % back-subsitution
x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
end
t=toc(timer);
function [x, t] = GaussElim_pp(A,b) % Gaussian elimination with
```

```
   partial pivoting
timer = tic;
n = size(A,1); x = zeros(n,1);
for i = 1:n
[p, maxk] = max(abs(A(i:n,i))); % pivot search
maxk = maxk+i-1; p = A(maxk,i);
if i ~= maxk % row interchange
tmp = A(i,i:n); A(i,i:n) = A(maxk, i:n); A(maxk, i:n) = tmp;
z = b(i); b(i) = b(maxk); b(maxk) = z;
end
for j = (i+1):n % elimination
A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
b(j) = b(j)-b(i)*(A(j,i)/p); % forward substitution
end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1 % back-subsitution
x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
end
t=toc(timer);
function [x, t] = GaussElim_cp(A,b) % Gaussian elimination with
   complete pivoting
timer = tic;
n = size(A,1); x = zeros(n,1); index = 1:n;
for i = 1:n
[maxA, maxK] = max(A(i:n,i:n)); % pivot search
[p, maxl] = max(maxA); maxk = maxK(maxl)+i-1;
maxl = maxl+i-1; p = A(maxk,maxl);
if i ~= maxk % row interchange
tmp = A(i,:); A(i,:) = A(maxk,:); A(maxk,:) = tmp;
z = b(i); b(i) = b(maxk); b(maxk) = z;
end
if i ~= maxl % column interchange
tmp = A(:,i); A(:,i) = A(:,maxl); A(:,maxl) = tmp;
ii = index(i); index(i) = index(maxl); index(maxl) = ii;
end;
for j = (i+1):n % elimination
A(j,(i+1):n) = A(j,(i+1):n)-A(i,(i+1):n)*(A(j,i)/p);
b(j) = b(j)-b(i)*(A(j,i)/p); % forward substitution
end;
end; x(n) = b(n)/A(n,n);
for i = n-1:-1:1 % back-subsitution
x(i) = (b(i)-dot(x((i+1):n),A(i,(i+1):n)))/A(i,i);
```

```
end; y = x;
for i = 1:n % reverse permutation
x(i) = y(index(i));
end;
t=toc(timer);
```

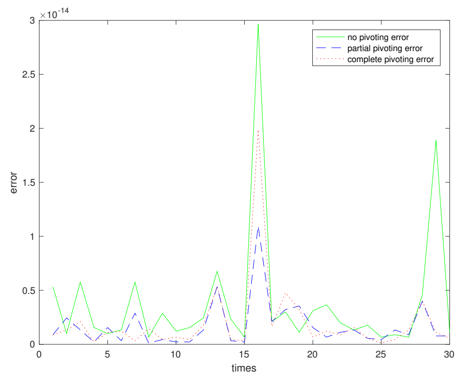

Figure 1: Error

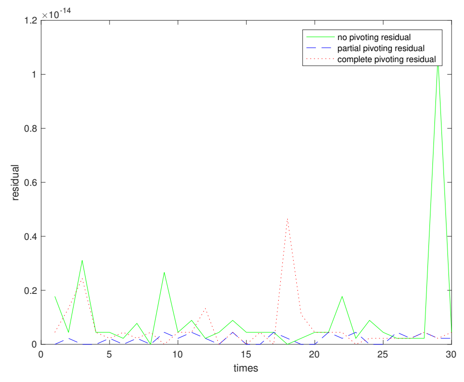

Figure 2: Residual

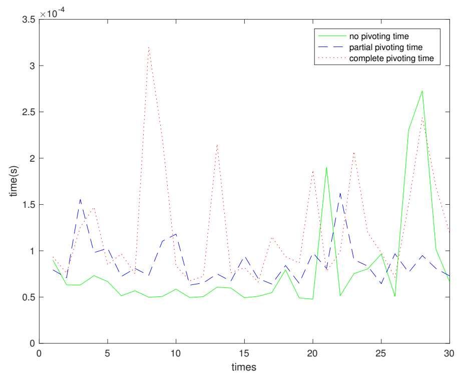

Figure 3: Time

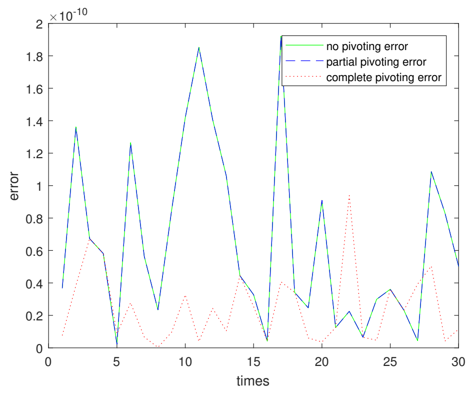

Figure 4: Error of c

(b) Implement Gaussian elimination on thirty 4x4 matrices with random entries(in [0, 1]) and plot these matrices' error, residuals, and time in Figures 1,2, and 3, respectively.

- Accuracy: Sometimes the random matrices may have pivots close to zero, the error of no pivoting is larger than those of partial and complete pivoting. The difference in the error between partial pivoting and complete pivoting.
- Residuals: No pivoting has larger residual than partial and complete pivoting have most of time. Besides, partial pivoting has larger residual than that of complete pivoting most of time.
- Performance:Most of time, no pivoting leads to much better performance than partial and complete pivoting have. In some cases, complete pivoting takes too much time, which leads to the worse performance of complete pivoting compared to no and partial pivoting.

(c) Let $A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 \end{bmatrix}$,

and randomly chose x and b 30 times. We got the results as Figure 4.

Especially in 11th where $x = \begin{bmatrix} 27.4568 \\ 23.8036 \\ 75.9943 \\ 96.0094 \\ 16.6206 \\ 37.8594 \\ 9.6802 \\ 93.4562 \\ 80.2916 \\ 73.0205 \\ 79.4713 \\ 57.3570 \\ 36.1690 \\ 16.1334 \\ 88.6371 \\ 90.1999 \end{bmatrix}$, $b = Ax = \begin{bmatrix} 117.6567 \\ 86.5466 \\ 114.9338 \\ 58.9546 \\ -116.4437 \\ -111.8255 \\ -177.8641 \\ -103.7683 \\ -210.3892 \\ -297.9519 \\ -364.5216 \\ -466.1073 \\ -544.6522 \\ -600.8569 \\ -544.4865 \\ -721.7607 \end{bmatrix}$,

we got $errorPP = 1.8528e-10$, $errorCP = 3.8938e-12$, $\frac{errorPP}{errorCP} = 47.583$.

- **2.16**

```
function cp02_16(n) % performance of algorithms for triangular
   solution
A = rand(n,n); b = rand(n,1); [Q,U] = qr(A); L = U';
tic; row_forward_sub(L, b); time = toc;
fprintf('Time␣using␣row-wise␣forward␣substitution:␣%f\n',time);
```

```
tic; col_forward_sub(L, b); time = toc;
fprintf('Time␣using␣column-wise␣forward␣substitution:␣%f\n',time);
tic; row_back_sub(L, b); time = toc;
fprintf('Time␣using␣row-wise␣back␣substitution:␣%f\n',time);
28 Chapter 2: Systems of Linear Equations
tic; col_back_sub(L, b); time = toc;
fprintf('Time␣using␣column-wise␣back␣substitution:␣%f\n',time);
function [x] = row_forward_sub(L, b)
n = length(b); x = zeros(n,1);
for i = 1:n
        tot = 0;
        for j = 1:i-1
                tot = L(i,j)*x(j);
        end
        if L(i,i) == 0, error('Matrix␣is␣singular'); end
        x(i) = (b(i)-tot)/L(i,i);
end
function [x] = row_back_sub(U, b)
n = length(b); x = zeros(n,1);
for i = n:-1:1
        tot = 0;
        for j = i+1:n
                tot = U(i,j)*x(j);
        end
        if U(i,i) == 0, error('Matrix␣is␣singular'); end
        x(i) = (b(i)-tot)/U(i,i);
end
function [x] = col_back_sub(U, b)
n = length(b); x = zeros(n,1);
for j = n:-1:1
        if U(j,j) == 0, error('Matrix␣is␣singular'); end
        x(j) = b(j)/U(j,j);
        for i = 1:j-1
                b(i) = b(i)-U(i,j)*x(j);
        end
end
function [x] = col_forward_sub(L, b)
n = length(b); x = zeros(n,1);
for j = 1:n
        if L(j,j) == 0, error('Matrix␣is␣singular'); end
        x(j) = b(j)/L(j,j);
        for i = j+1:n
```

```
                b(i) = b(i)-L(i,j)*x(j);
        end
end
```

Implement an 8000x8000 matrix with random entries(in [0, 1]), and we got the result:

```
Time using row-wise forward substitution: 2.025865
Time using column-wise forward substitution: 1.067127
Time using row-wise back substitution: 2.209032
Time using column-wise back substitution: 1.020354
```

We find that the time with column-wise is much shorter than the time using row-wise, because a MATLAB matrix is stored by columns in continuous RAM addresses.