CSC336 A3

Kaiqu Liang

November 22, 2018

Problem 1

(a)
MatLab Program:

```
function question1a
Error1 = zeros(10, 1);
Error2 = zeros(10, 1);
Gamma = zeros(10, 1);
K = zeros(10, 1);
for k = 1:10
    b = [1; 2];
    x = [1; 1];
    gamma = 10^(-2 * k);
    L = [1 0; 1/gamma 1];
    U = [gamma 1 - gamma; 0 2 - 1/gamma];
    y = L\b;
    x_hat = U\y;
    error = x_hat - x;

    K(k) = k;
    Gamma(k) = gamma;
    Error1(k) = error(1);
    Error2(k) = error(2);

end
Table = table(K, Gamma, Error1, Error2, 'VariableNames', ...
        {'k', 'gamma', 'error1', 'error2'});
disp(Table);
```

Program Output:

```
>> question1a
    k      gamma        error1        error2
    --     ------     -----------     ------

     1        0.01      8.8818e-16       0
     2      0.0001     -1.1013e-13       0
     3       1e-06      2.8756e-11       0
     4       1e-08      5.0248e-09       0
```

```
      5       1e-10        8.274e-08        0
      6       1e-12      -2.2122e-05        0
      7       1e-14      -0.00079928        0
      8       1e-16          0.11022        0
      9       1e-18               -1        0
     10       1e-20               -1        0
```

As γ decreases, error(1) increases.
Therefore the computed solution becomes more inaccurate.

(b)
MatLab Program:

```
function question1b
Error1 = zeros(10, 1);
Error2 = zeros(10, 1);
Gamma = zeros(10, 1);
K = zeros(10, 1);
for k = 1:10
    b = [1; 2];
    x = [1; 1];
    P = [0 1; 1 0];
    b_hat = P * b;
    gamma = 10^(-2 * k);
    L = [1 0; gamma 1];
    U = [1 1; 0 1 - 2 * gamma];
    y = L\b_hat;
    x_hat = U\y;
    error = x_hat - x;

    K(k) = k;
    Gamma(k) = gamma;
    Error1(k) = error(1);
    Error2(k) = error(2);

end
Table = table(K, Gamma, Error1, Error2, 'VariableNames', ...
        {'k', 'gamma', 'error1', 'error2'});
disp(Table);
```

Program Output:

```
>> question1b
     k      gamma      error1     error2
     --     ------     ------     ------

      1       0.01        0          0
      2     0.0001        0          0
      3      1e-06        0          0
```

```
    4      1e-08      0       0
    5      1e-10      0       0
    6      1e-12      0       0
    7      1e-14      0       0
    8      1e-16      0       0
    9      1e-18      0       0
   10      1e-20      0       0
```

By the table in program output, we can see that as γ decreases, error(1) and error(2) are always 0, which indicates that the computed solution is very accurate.
Pivoting can avoid loss of information in transformed matrix if the pivot is very small. Therefore, it is very important to reduce the rounding error and compute the accurate solution.

(c)
MatLab Program:

```
function question1c
Error1 = zeros(10, 1);
Error2 = zeros(10, 1);
Gamma = zeros(10, 1);
K = zeros(10, 1);
for k = 1:10
    b = [1; 2];
    x = [1; 1];
    gamma = 10^(-2 * k);
    A = [gamma 1 - gamma; 1 1];
    L = [1 0; 1/gamma 1];
    U = [gamma 1 - gamma; 0 2 - 1/gamma];
    y = L\b;
    x_hat = U\y;

    r = b - A * x_hat;
    z = L\r;
    e = U\z;
    x_tilde = x_hat + e;
    error = x_tilde - x;

    K(k) = k;
    Gamma(k) = gamma;
    Error1(k) = error(1);
    Error2(k) = error(2);

end
Table = table(K, Gamma, Error1, Error2, 'VariableNames', ...
        {'k', 'gamma', 'error1', 'error2'});
disp(Table);
```

Program Output:

```
>> question1c
     k      gamma      error1     error2
    --      ------     ------     ------

     1        0.01       0          0
     2      0.0001       0          0
     3       1e-06       0          0
     4       1e-08       0          0
     5       1e-10       0          0
     6       1e-12       0          0
     7       1e-14       0          0
     8       1e-16       0          0
     9       1e-18       0          0
    10       1e-20       0          0
```

By the table in program output, we can see that as γ decreases, error(1) and error(2) are always 0, which indicates that the "better" approximation solution x is very accurate.
Iterative refinement is very effective in this context. This process potentially produce a solution with a residual as small as possible for the arithmetic precision used and recover full accuracy for systems that are badly scaled.

Problem 2

(c)
MatLab Program:

```
function question2c
n = 60;
A = ones(n, n);
A = A - triu(A);
A = eye(n) - A;
A = A + [ones(n - 1, 1); 0] * [zeros(1, n - 1), 1];

Q = diag(ones(n - 1, 1), 1);
Q(n, 1) = 1;

[L1, U1, P1] = lu(A);
fprintf('U1(n, n) = %d, 2^(n - 1) = %d\n', U1(n, n), 2^(n - 1));
if (U1(n, n) == 2^(n - 1))
    fprintf('Therefore, U1(n, n) = 2^(n - 1)\n');
end

[L2, U2] = lu(A * Q);
fprintf('%s %d.\n\n', 'The value of largest element of U2 is', max(max(abs(U2))));

x = ones(n, 1);
b = A * x;
y1 = L1\b;
x1 = U1\y1;
fprintf('%s %d\n', 'norm(x-x1, inf) =', norm(x-x1, inf));
fprintf(['The relative error of this poor approximation is %d,\nwhich' ...
    ' means that row partial pivoting in this case is very inaccurate.\n\n'],...
    norm(x - x1, inf)/ norm(x, inf));
y2 = L2\b;
z = U2\y2;
x2 = Q * z;
fprintf('%s %d\n', 'norm(x-x2, inf) =', norm(x-x2, inf));
fprintf(['The relative error of this good approximation is %d,\nwhich' ...
    ' means that complete pivoting in this case is very accurate.\n'],...
    norm(x - x2, inf)/ norm(x, inf));
```

Program Output:

```
>> question2c
U1(n, n) = 576460752303423488, 2^(n - 1) = 576460752303423488
Therefore, U1(n, n) = 2^(n - 1)
The value of largest element of U2 is 2.

norm(x-x1, inf) = 1
The relative error of this poor approximation is 1,
which means that row partial pivoting in this case is very inaccurate.

norm(x-x2, inf) = 0
The relative error of this good approximation is 0,
which means that complete pivoting in this case very accurate.
```

Problem 4

(a)
MatLab Program:

```
function y = perm_a(p, x)
n = length(x);
y = x;
for i = 1 : n - 1
    y([p(i) i]) = y([i p(i)]);
end
```

(b)
MatLab Program:

```
function q = perm_b(p)
n = length(p) + 1;
x = (1: n)';
q1 = perm_a(p, x);
q = q1';
```

(c)
MatLab Program:

```
function y = perm_c(q, x)
n = length(x);
y = x;
for i = 1: n
    y(i,:) = x(q(i), :);
end
```

(d)
Test Program:

```
function test
p = [5, 4, 9, 10, 6, 8, 10, 9, 10];
x = (1: 10)';
y1 = perm_a(p, x);
q = perm_b(p);
y2 = perm_c(q, x);
display(y1);
display(q);
display(y2);
```

Program output:

```
>> test

y1 =

     5
     4
     9
    10
     6
     8
     2
     3
     7
     1


q =

     5     4     9    10     6     8     2     3     7     1


y2 =

     5
     4
     9
    10
     6
     8
     2
     3
     7
     1
```

Problem 5

(a)

MatLab Program:

```
function p = perm_d(q)
n = length(q);
index_list = (1: n);
current_list = (1: n);
p = (1: n - 1);
for i = 1: n - 1
    actual_value = q(i);
    actual_index = index_list(actual_value);
    current_value = current_list(i);
    %interchange index_list(actual_number) and index_list(current_value)
    index_list([current_value actual_value]) = index_list([actual_value current_value]);
    %interchange current_list(i) and current_list(actual_index)
    current_list([i actual_index]) = current_list([actual_index i]);
    p(i) = actual_index;
end
```

Test program:

```
function test5
q = [5, 4, 9, 10, 6, 8, 2, 3, 7, 1];
p = perm_d(q);
display(p);
```

Program output:

```
>> test5

p =

     5     4     9    10     6     8    10     9    10
```

(b)

MatLab Program:

```
function detQ = find_det(p)
detQ = 1;
n = length(p);
for i = 1: n
    if i == p(i)
        det_p = 1;
    else
        det_p = -1;
    end
    detQ = detQ * det_p;
end
```