CSC 336 A1

Yichen Ji 1004728967

February 2021

1 Question 1

(a)

Recall the definition of the condition number: $\kappa_f = |\frac{xf'(x)}{f(x)}|$. Given $f(x) = (a+x)^{1/4} - a^{1/4}$, for $x > 0, a > 0$, the first-order derivative is $f'(x) = \frac{1}{4(a+x)^{3/4}}$, so the general formula of κ_f is as follows:

$$\begin{aligned}
\kappa_f &= |\frac{\frac{x}{4(a+x)^{3/4}}}{(a+x)^{1/4} - a^{1/4}}| \\
&= \frac{1}{4}|\frac{x}{(a+x)^{3/4}((a+x)^{1/4} - a^{1/4})}| \\
&= \frac{1}{4}|\frac{x}{(a+x) - a^{1/4}(a+x)^{3/4}}|
\end{aligned}$$

Since large κ_f reflects a relative large sensitivity of the computation of $f(x)$ on relatively small changes in the input x, we access the cases when the numerator in κ_f is large and the denominator is small separately:

- when the numerator is large i.e. $x \to \infty$, by applying de l'Hospital's rule,

$$\lim_{x\to\infty} \kappa_f = \lim_{x\to\infty} \frac{1}{4}\frac{1}{1 - \frac{3a^{1/4}}{4(a+x)^{1/4}}} = \frac{1}{4}$$

- when the denominator is small i.e. $a + x \approx a^{1/4}(a+x)^{3/4} \Leftrightarrow x \approx 0$,

$$\lim_{x\to 0} \kappa_f = \lim_{x\to 0} \frac{1}{4}\frac{1}{1 - \frac{3a^{1/4}}{4(a+x)^{1/4}}} = \frac{1}{4}\frac{1}{1-\frac{3}{4}} = 1$$

In both cases, the condition number is relatively small, so there is no sizable range for which the computation of f is ill-conditioned.

(b)

When x is close to 0, the expression $(a+x)^{1/4} - a^{1/4} \approx a^{1/4} - a^{1/4} = 0$, that is, subtracting nearly equal terms reduces the numerical stability of this computation, which might result in the catastrophic cancellation problem. Therefore, here we propose a mathematically equivalent expression:

$$(a+x)^{1/4} - a^{1/4} = \frac{x}{((a+x)^{1/2} + a^{1/2})((a+x)^{1/4} + a^{1/4})}$$

The expression on RHS avoids subtracting nearly equal numbers, despite the fact that more operations like addition and multiplication are required to get the output.

(c)

Here are my MATLAB codes for computation and outputs:

```
x=10.^(-20:20);
a = 1;
% default setup

f_ori = @(x) (a+x).^0.25 - a.^0.25;
df_ori = @(x) 0.25*(a + x).^(-0.75);
% the original expression and its derivative

f_proposed = @(x) x./(((a+x).^0.5 + a.^0.5)*((a+x).^0.25 + a.^0.25));
df_proposed = @(x) (((a+x).^0.5 + a.^0.5)*((a+x).^0.25 + a.^0.25)- ...
x*(0.75*(a+x).^(-0.25)+a^0.25*0.5*(a+x).^(-0.5)+a^0.5*0.25*(a+x).^(-0.75)))./...
(((a+x).^0.5 + a.^0.5)*((a+x).^0.25 + a.^0.25)).^2;
% the proposed expression and its derivative

condition_ori = @(x_val) abs(x_val .* df_ori(x_val) ./ f_ori(x_val));
condition_proposed = @(x_val) abs(x_val .* df_proposed(x_val)...
./ f_proposed(x_val));
% the condition numbers of two functions

rel_error = @(x) (f_proposed(x) - f_ori(x))./ f_proposed(x);
% the relative error wrt. the proposed expression


for i=1:41
    fprintf('x:%9.2e  f(x):%12.5e  g(x):%12.5e  kf:%12.5e  kg: %12.5e  error:%10
    ,x(i),f_ori(x(i)),f_proposed(x(i)), condition_ori(x(i)),...
    condition_proposed(x(i)),rel_error(x(i)));
end
```

Here is my output from the MATLAB command window. Note that kf denotes the condition numberκ_f, kg denotes the condition number κ_g and g(x) denotes the newly proposed expression i n part (b):

x: 1.00e-20 f(x): 0.00000e+00 g(x): 2.50000e-21 kf: Inf kg: 1.00000e+00 error: 1.00e+000
x: 1.00e-19 f(x): 0.00000e+00 g(x): 2.50000e-20 kf: Inf kg: 1.00000e+00 error: 1.00e+000
x: 1.00e-18 f(x): 0.00000e+00 g(x): 2.50000e-19 kf: Inf kg: 1.00000e+00 error: 1.00e+000
x: 1.00e-17 f(x): 0.00000e+00 g(x): 2.50000e-18 kf: Inf kg: 1.00000e+00 error: 1.00e+000
x: 1.00e-16 f(x): 0.00000e+00 g(x): 2.50000e-17 kf: Inf kg: 1.00000e+00 error: 1.00e+000
x: 1.00e-15 f(x): 2.22045e-16 g(x): 2.50000e-16 kf: 1.12590e+00 kg: 1.00000e+00 error: 1.12e-010
x: 1.00e-14 f(x): 2.44249e-15 g(x): 2.50000e-15 kf: 1.02355e+00 kg: 1.00000e+00 error: 2.30e-020
x: 1.00e-13 f(x): 2.48690e-14 g(x): 2.50000e-14 kf: 1.00527e+00 kg: 1.00000e+00 error: 5.24e-030
x: 1.00e-12 f(x): 2.50022e-13 g(x): 2.50000e-13 kf: 9.99911e-01 kg: 1.00000e+00 error: -8.89e-050
x: 1.00e-11 f(x): 2.50000e-12 g(x): 2.50000e-12 kf: 1.00000e+00 kg: 1.00000e+00 error: -8.27e-080
x: 1.00e-10 f(x): 2.50000e-11 g(x): 2.50000e-11 kf: 1.00000e+00 kg: 1.00000e+00 error: -8.28e-080
x: 1.00e-09 f(x): 2.50000e-10 g(x): 2.50000e-10 kf: 1.00000e+00 kg: 1.00000e+00 error: -8.31e-080
x: 1.00e-08 f(x): 2.50000e-09 g(x): 2.50000e-09 kf: 1.00000e+00 kg: 1.00000e+00 error: 2.33e-090
x: 1.00e-07 f(x): 2.50000e-08 g(x): 2.50000e-08 kf: 1.00000e+00 kg: 1.00000e+00 error: -4.78e-090
x: 1.00e-06 f(x): 2.50000e-07 g(x): 2.50000e-07 kf: 1.00000e+00 kg: 1.00000e+00 error: -3.28e-100
x: 1.00e-05 f(x): 2.49999e-06 g(x): 2.49999e-06 kf: 9.99996e-01 kg: 9.99996e-01 error: 3.78e-110
x: 1.00e-04 f(x): 2.49991e-05 g(x): 2.49991e-05 kf: 9.99963e-01 kg: 9.99963e-01 error: -1.02e-120

x: 1.00e-03 f(x): 2.49906e-04 g(x): 2.49906e-04 kf: 9.99625e-01 kg: 9.99625e-01 error: -3.15e-140
x: 1.00e-02 f(x): 2.49068e-03 g(x): 2.49068e-03 kf: 9.96279e-01 kg: 9.96279e-01 error: -1.74e-160
x: 1.00e-01 f(x): 2.41137e-02 g(x): 2.41137e-02 kf: 9.65232e-01 kg: 9.65232e-01 error: 5.76e-160
x: 1.00e+00 f(x): 1.89207e-01 g(x): 1.89207e-01 kf: 7.85652e-01 kg: 7.85652e-01 error: 2.93e-160
x: 1.00e+01 f(x): 8.21160e-01 g(x): 8.21160e-01 kf: 5.04043e-01 kg: 5.04043e-01 error: -1.35e-160
x: 1.00e+02 f(x): 2.17015e+00 g(x): 2.17015e+00 kf: 3.61583e-01 kg: 3.61583e-01 error: 0.00e+000
x: 1.00e+03 f(x): 4.62482e+00 g(x): 4.62482e+00 kf: 3.03752e-01 kg: 3.03752e-01 error: 1.92e-160
x: 1.00e+04 f(x): 9.00025e+00 g(x): 9.00025e+00 kf: 2.77749e-01 kg: 2.77749e-01 error: 1.97e-160
x: 1.00e+05 f(x): 1.67828e+01 g(x): 1.67828e+01 kf: 2.64894e-01 kg: 2.64894e-01 error: -2.12e-160
x: 1.00e+06 f(x): 3.06228e+01 g(x): 3.06228e+01 kf: 2.58164e-01 kg: 2.58164e-01 error: -1.16e-160
x: 1.00e+07 f(x): 5.52341e+01 g(x): 5.52341e+01 kf: 2.54526e-01 kg: 2.54526e-01 error: -1.29e-160
x: 1.00e+08 f(x): 9.90000e+01 g(x): 9.90000e+01 kf: 2.52525e-01 kg: 2.52525e-01 error: 0.00e+000
x: 1.00e+09 f(x): 1.76828e+02 g(x): 1.76828e+02 kf: 2.51414e-01 kg: 2.51414e-01 error: 0.00e+000
x: 1.00e+10 f(x): 3.15228e+02 g(x): 3.15228e+02 kf: 2.50793e-01 kg: 2.50793e-01 error: 0.00e+000
x: 1.00e+11 f(x): 5.61341e+02 g(x): 5.61341e+02 kf: 2.50445e-01 kg: 2.50445e-01 error: 2.03e-160
x: 1.00e+12 f(x): 9.99000e+02 g(x): 9.99000e+02 kf: 2.50250e-01 kg: 2.50250e-01 error: 0.00e+000
x: 1.00e+13 f(x): 1.77728e+03 g(x): 1.77728e+03 kf: 2.50141e-01 kg: 2.50141e-01 error: -1.28e-160
x: 1.00e+14 f(x): 3.16128e+03 g(x): 3.16128e+03 kf: 2.50079e-01 kg: 2.50079e-01 error: 0.00e+000
x: 1.00e+15 f(x): 5.62241e+03 g(x): 5.62241e+03 kf: 2.50044e-01 kg: 2.50044e-01 error: 0.00e+000
x: 1.00e+16 f(x): 9.99900e+03 g(x): 9.99900e+03 kf: 2.50025e-01 kg: 2.50025e-01 error: 1.82e-160
x: 1.00e+17 f(x): 1.77818e+04 g(x): 1.77818e+04 kf: 2.50014e-01 kg: 2.50014e-01 error: 0.00e+000
x: 1.00e+18 f(x): 3.16218e+04 g(x): 3.16218e+04 kf: 2.50008e-01 kg: 2.50008e-01 error: 0.00e+000
x: 1.00e+19 f(x): 5.62331e+04 g(x): 5.62331e+04 kf: 2.50004e-01 kg: 2.50004e-01 error: -1.29e-160
x: 1.00e+20 f(x): 9.99990e+04 g(x): 9.99990e+04 kf: 2.50003e-01 kg: 2.50003e-01 error: 0.00e+000

Comments:
- When $x \leq 10^{-16}$, the condition number κ_f becomes infinitely large. Notice that $\epsilon_{machine} = 10^{-16}$ for double precision calculations, so we confirm that the original expression is numerically unstable due to the subtraction of two nearly equal numbers in the denominator. In contrast, the condition number of my proposed expression $\kappa_g = 1$ for the same range of x so it happens to be a well-conditioned computation. To conclude, $g(x)$ is more stable than the original $f(x)$ when $x \to 0$.
- When $x > 10^{-16}$, larger than the machine epsilon i.e. the difference between 1 and the next representable number, we would not claim $a + x = a$, so the issue of the denominator being nearly zero can be remedied. As we can see from the output, the values of functions and their condition numbers are nearly the same, and the relative error between these two expressions are relatively small. Therefore, we can assure that $g(x)$ performs as well as $f(x)$ for other values of input.
- Note that as $x \to \infty$, both condition numbers converge to $\frac{1}{4}$, which is a small constant independent of x, which implies that both expressions are well-conditioned and numerically stable.
In summary, $g(x)$ can be deemed as a more stable expression than the original $f(x)$.

2 Question 2

(a)

Applying integration by parts, denote $u = t^n, dv = e^{-t}dt$, then $du = nt^{n-1}dt, v = -e^{-t}$. As a result,

$$\begin{aligned}
y_n &= \int_0^1 t^n e^{-t} dt \\
&= uv\Big|_0^1 - \int_0^1 v du \\
&= -t^n e^{-t}\Big|_0^1 - \int_0^1 nt^{n-1}e^{-t}dt \\
&= -e^{-1} + ny_{n-1}
\end{aligned}$$

That is,

$$y_n = -e^{-1} + ny_{n-1} \tag{A}$$

Rearrange the formula:

$$y_{n-1} = \frac{y_n + e^{-1}}{n} \tag{B}$$

(b)

With repeated applications of (A), the formula of $y_n = f_n(y_0)$ is as follows:

$$\begin{aligned}
y_n &= ny_{n-1} - e^{-1} \\
&= n((n-1)y_{n-2} - e^{-1}) - e^{-1} \\
&= n(n-1)y_{n-2} - (n+1)e^{-1} \\
&= n(n-1)((n-2)y_{n-3} - e^{-1}) - (n+1)e^{-1} \\
&= n(n-1)(n-2)y_{n-3} - ((n-1)n + n + 1)e^{-1} \\
&= ... \\
&= n!y_0 - \sum_{i=1}^{n} \frac{n!}{i!}e^{-1}
\end{aligned}$$

Similarly, with repeated applications of (B), the formula of $y_n = g_{n,m}(y_m)$ is as follows:

$$\begin{aligned}
y_n &= \frac{y_{n+1} + e^{-1}}{n+1} \\
&= \frac{\frac{y_{n+2}+e^{-1}}{n+2} + e^{-1}}{n+1} \\
&= \frac{y_{n+2} + (1 + (n+2))e^{-1}}{(n+2)(n+1)} \\
&= \frac{y_{n+2}}{(n+2)(n+1)} + (\frac{1}{(n+2)(n+1)} + \frac{1}{n+1})e^{-1} \\
&= \frac{\frac{y_{n+3}+e^{-1}}{n+3}}{(n+2)(n+1)} + (\frac{1}{(n+2)(n+1)} + \frac{1}{n+1})e^{-1} \\
&= \frac{y_{n+3}}{(n+3)(n+2)(n+1)} + (\frac{1}{(n+3)(n+2)(n+1)} + \frac{1}{(n+2)(n+1)} + \frac{1}{n+1})e^{-1} \\
&= ... \\
&= \frac{n!}{m!}y_m + \sum_{i=0}^{m-n-1} \frac{n!}{(m-i)!}e^{-1}
\end{aligned}$$

(c)

We treat y_0 as the variable of f_n and y_m as the variable of $g_{n,m}$ respectively, then, to find the condition number of both functions, we first attain their first-order derivatives:

$$f_n'(y_0) = n!$$
$$g_{n,m}'(y_m) = \frac{n!}{m!}$$

Therefore, the condition number κ_f is:

$$\begin{aligned}\kappa_f &= |\frac{y_0 f'(y_0)}{f(y_0)}| \\ &= |\frac{n! y_0}{y_n}| \\ &= |\frac{n! y_0}{n! y_0 - \sum_{i=1}^{n} \frac{n!}{i!} e^{-1}}| \\ &= |\frac{y_0}{y_0 - \sum_{i=1}^{n} \frac{1}{i! e}}| \\ &= |\frac{1}{1 - \frac{1}{y_0 e} \sum_{i=1}^{n} \frac{1}{i!}}|\end{aligned}$$

And the condition number κ_g is:

$$\begin{aligned}\kappa_g &= |\frac{g_{n,m}'(y_m) y_m}{g_{n,m}(y_m)}| \\ &= |\frac{\frac{n!}{m!} y_m}{y_n}| \\ &= |\frac{\frac{n!}{m!} y_m}{\frac{n!}{m!} y_m + \sum_{i=0}^{m-n-1} \frac{n!}{(m-i)!} e^{-1}}| \\ &= |\frac{y_m}{y_m + \sum_{i=0}^{m-n-1} \frac{m!}{(m-i)!} e^{-1}}| \\ &= |\frac{1}{1 + \frac{1}{y_m e} \sum_{i=0}^{m-n-1} \frac{m!}{(m-i)!}}|\end{aligned}$$

(d)

Here are my MATLAB codes for computation and outputs using recursion (A):

```
% Compute and output y0, ..., y20 using recursion (A)

q = 0.018350467697256206326; % the assumed exact value
y = zeros(1,21); % create a container matrix
y(1) = 1 - exp(1)^(-1);

for i = 2:21
    y(i) = (i - 1) * y(i - 1) - exp(-1); % calculation step
    fprintf('n:%3d y_n:%20.16f \n', i-1, y(i));
end
fprintf('n:%3d y_n:%20.16f q:%20.16f error:%10.6e\n', 20, y(21), q, q-y(21));
```

Here is the output:

n	y_n
0	0.6321205588285577
1	0.2642411176571153
2	0.1606027941427883
3	0.1139289412569227
4	0.0878363238562483
5	0.0713021781097991
6	0.0599336274873523
7	0.0516559512400239
8	0.0453681687487486
9	0.0404340775672951
10	0.0364613345015088
11	0.0331952383451544
12	0.0304634189704100
13	0.0281450054438883
14	0.0261506350429941
15	0.0243800844734690
16	0.0222019104040609
17	0.0095530356975937
18	-0.1959247986147559
19	-4.0904506148518038
20	-82.1768917382075159

n: 20 y_n: -82.1768917382075159 q: 0.0183504676972562 error: 8.219524e+01

Explanations:

Treating y_0 as an independent variable, κ_f increases as y_0 decreases. More interestingly, we can get the exact value $y_0 = 1 - \frac{1}{e}$ by integral calculation, and if we apply recursion (A) with $y_0 = 1 - \frac{1}{e}$ to calculate y_n where n is relatively large, given the fact that $\sum_{i=1}^{\infty} \frac{1}{i!} = e - 1$, $\kappa_f \approx \frac{1-e^{-1}}{1-e^{-1}-(e-1)e^{-1}} = \frac{1-e^{-1}}{0} \to \infty$. As a result, the calculation will be ill-conditioned and unstable as n increases.

We can see from the output that as n increases, y_n decreases, and there are even negative values of y_n when $n > 17$ dropping more drastically to $y_20 \approx -82.1769$. Negative values are not reasonable since y_n is defined to be a positive integral irrespective of the choice of n. Moreover, the absolute error between our calculation y_{20} and the assumed value q is sizable($\approx$ 82.2), which corresponds to our previous result that , our computation of $y_{large\ n}$ is unstable, given $y_0 = 1 - \frac{1}{e}$. The finite sum when N=20 can be deemed a good approximation for the infinite sum, so the condition number happens to be extremely large.

(e)

Here are my codes for computation and outputs using recursion (B):

```
q = 0.018350467697256206326; % the assumed exact value
N = 20;
for K = 3:9
    y_N_plus_K = 0.1; % set an appropriate value for y_{N+K}
    y = zeros(1,N+K); % so the first N-1 entries are 0
    y(N+K) = y_N_plus_K;
    i = N + K -1;
    while i >= 20
        y(i) = (y(i+1) + exp(1)^(-1))/(i + 1);
```

```
        fprintf('N+K-1: %3d y_{N+K-1}: %20.16f\n',i, y(i));
        i = i - 1;
    end
    fprintf('K:%3d y_N:%20.16f q:%20.16f error:%10.6e \n',...
    K, y(20), q, q-y(20));
end
```

Here is the output:
Output:

N+K-1: 22 y_{N+K-1}: 0.0203425843987584
N+K-1: 21 y_{N+K-1}: 0.0176464557077364
N+K-1: 20 y_{N+K-1}: 0.0183583760418656
K: 3 y_{20}: 0.0183583760418656 q: 0.0183504676972562 error:-7.908345e-06
N+K-1: 23 y_{N+K-1}: 0.0194949767154768
N+K-1: 22 y_{N+K-1}: 0.0168423659950834
N+K-1: 21 y_{N+K-1}: 0.0174873548712057
N+K-1: 20 y_{N+K-1}: 0.0183507998115547
K: 4 y_{20}: 0.0183507998115547 q: 0.0183504676972562 error:-3.321143e-07
N+K-1: 24 y_{N+K-1}: 0.0187151776468577
N+K-1: 23 y_{N+K-1}: 0.0161081091174292
N+K-1: 22 y_{N+K-1}: 0.0166951108821248
N+K-1: 21 y_{N+K-1}: 0.0174806614569803
N+K-1: 20 y_{N+K-1}: 0.0183504810775439
K: 5 y_{20}: 0.0183504810775439 q: 0.0183504676972562 error:-1.338029e-08
N+K-1: 25 y_{N+K-1}: 0.0179953631219785
N+K-1: 24 y_{N+K-1}: 0.0154349921717368
N+K-1: 23 y_{N+K-1}: 0.0159714347226325
N+K-1: 22 y_{N+K-1}: 0.0166891685171337
N+K-1: 21 y_{N+K-1}: 0.0174803913494807
N+K-1: 20 y_{N+K-1}: 0.0183504682152820
K: 6 y_{20}: 0.0183504682152820 q: 0.0183504676972562 error:-5.180258e-10
N+K-1: 26 y_{N+K-1}: 0.0173288681915349
N+K-1: 25 y_{N+K-1}: 0.0148157042062684
N+K-1: 24 y_{N+K-1}: 0.0153078058151084
N+K-1: 23 y_{N+K-1}: 0.0159661352911063
N+K-1: 22 y_{N+K-1}: 0.0166889381070673
N+K-1: 21 y_{N+K-1}: 0.0174803808762959
N+K-1: 20 y_{N+K-1}: 0.0183504677165590
K: 7 y_{20}: 0.0183504677165590 q: 0.0183504676972562 error:-1.930275e-11
N+K-1: 27 y_{N+K-1}: 0.0167099800418372
N+K-1: 26 y_{N+K-1}: 0.0142440526375289
N+K-1: 25 y_{N+K-1}: 0.0146970574541912
N+K-1: 24 y_{N+K-1}: 0.0153030599450253
N+K-1: 23 y_{N+K-1}: 0.0159659375465195
N+K-1: 22 y_{N+K-1}: 0.0166889295094766
N+K-1: 21 y_{N+K-1}: 0.0174803804854963
N+K-1: 20 y_{N+K-1}: 0.0183504676979495
K: 8 y_{20}: 0.0183504676979495 q: 0.0183504676972562 error:-6.932510e-13

N+K-1: 28 y_{N+K-1}: 0.0161337738334980
N+K-1: 27 y_{N+K-1}: 0.0137147576787479
N+K-1: 26 y_{N+K-1}: 0.0141331184759330
N+K-1: 25 y_{N+K-1}: 0.0146927907556683
N+K-1: 24 y_{N+K-1}: 0.0153028892770844
N+K-1: 23 y_{N+K-1}: 0.0159659304353553
N+K-1: 22 y_{N+K-1}: 0.0166889292002955
N+K-1: 21 y_{N+K-1}: 0.0174803804714426
N+K-1: 20 y_{N+K-1}: 0.0183504676972802
K: 9 y_{20}: 0.0183504676972802 q: 0.0183504676972562 error:-2.402592e-14

Notice from (d) that y_n decreases with n, so I used $y_{N+K} = 0.1 < q$ as an approximate value and the initial value of the backward recursion $g_{n,m}$. From the expression of κ_g, there won't be such cancellation issue as in κ_f, and we confirm this by looking at outputs of y_{N+k-1} and the absolute errors for each choice of K. All errors are quite small and moreover, as K increases i.e. M in the expression of κ_g goes up, y_{20} computation goes closer to the exact value of y_{20}, that is, q.

As a result, $g_{n,m}(y_m)$ should be considered as a more stable and accurate method to compute y_{20} to machine epsilon than $f_n(y_0)$. One can set a relative large K and small y_{N+K} as initial values and apply expression (B) recursively, which is more likely to attain a stable result.

3 Question 3

Our task is to compute $z = B^{-1}(2A + I)(B^{-1} + A)b$ with minimal operation costs where B is non-singular, I is the identity matrix of order n and b is a given $nx1$ vector for some large n. Here are the steps of my proposed computations:

Computations on $z = B^{-1}(2A + I)(B^{-1} + A)b$			
Computation Subject	Operation	Flop Counts	Division Counts
B	LU factorization/GE	$n^3/3$	$n^2/2$
$B^{-1}b$	f/s and b/s	n^2	n
Ab	matrix multiplication	n^2	0
$v_1 = B^{-1}b + Ab$	matrix addition	n	0
Av_1	matrix multiplication	n^2	0
$2Av_1$	scalar multiplication	n	0
Iv_1	matrix multiplication	0	0
$v_2 = 2Av_1 + Iv_1$	matrix addition	n	0
$B^{-1}v_2$	f/s and b/s	n^2	n

So the total operation counts:

$$operation\ counts = flops + division = (\frac{n^3}{3} + 4n^2 + 3n) + (n^2/2 + 2n) = \frac{n^3}{3} + \frac{9n^2}{2} + 5n \approx \frac{n^3}{3}$$

Since n is assumed to be large, we can use the term with highest power as an approximation of operation costs.

Some justifications for my choice of operations:

1. We know from the lecture that we can first apply GE/LU and store the L and U factors, then apply a pair of forward backward substitution(f/s and b/s in abbreviation) for different RHS vectors. Here we utilize this idea and apply twice for b and v_2 respectively and avoid calculating the inverse

matrix B^{-1} directly.

2. The distributive property of matrices is used to reduce operation counts. For example, given v_1 and mathematically equivalent expression $(2A + I)c = 2Ac + Ic$, RHS is more efficient than LHS $(2n^2 + n > n^2 + 2n)$ for large n.