4X03 – Assignment 2

Jeff Suitor 4001386789

1.

a. The accuracy of the solution is reduced as the conditional increases and the residual error increases as the conditional increases. This is because as the condition number increases the matrix becomes more susceptible and becomes more susceptible to errors such as those that occur when computing the inverse.
b. My results were not as accurate as MATLAB's for both the partial pivoting and scaled partial pivoting. It is likely that MATLAB is taking extra steps to ensure result accuracy.
c. Large residuals are likely caused by subtraction of values that are very close numerically causing a loss-of-significance error which gives the large residual.
d. Scaled partial pivoting can sometimes improve the accuracy of results such as when in a 2x2 matrix of the form

$$\begin{matrix} 2 & 200000 \\ 1 & 2 \end{matrix}$$

Scaled partial pivoting would pick row 2 as the pivot row which would be more accurate than row 1 as row one differs by orders of magnitude where row 2 does not

2. To test if MATLAB's lu function does scaled or partial pivoting you would create a 2x2 matrix of the form

$$\begin{matrix} 2 & 200000 \\ 1 & 2 \end{matrix}$$

You would then perform lu decomposition on it and look at the permutation matrix. If the permutation is identical to the identity matrix for a 2x2 than it does not do scaled partial pivoting but if the permutation is of the form

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix}$$

Then MATLAB does scaled partial pivoting.

3.

a. $.1\times(.5\times.9 - .8\times.6) - .2\times(.4\times.9 - .7\times.6) + .3\times(.4\times.8 - .7\times.5)$
$.1 \times (-0.03) - .2 \times (-0.06) + .3 \times (-0.03)$
$-0.12 + 0.12 = 0$
$\det(A) = 0$, therefore singular

b.

$$\left[\begin{array}{ccc|c} 0.1 & 0.2 & 0.3 & 0.1 \\ 0.4 & 0.5 & 0.6 & 0.3 \\ 0.7 & 0.8 & 0.9 & 0.5 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} 1 & 2 & 3 & 1 \\ 4 & 5 & 6 & 3 \\ 7 & 8 & 9 & 5 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} 7 & 8 & 9 & 5 \\ 4 & 5 & 6 & 3 \\ 1 & 2 & 3 & 1 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} 7 & 8 & 9 & 5 \\ 0 & 5-8\cdot\frac{4}{7} & 6-9\cdot\frac{4}{7} & 3-5\cdot\frac{4}{7} \\ 0 & 2-8\cdot\frac{1}{7} & 3-9\cdot 1/7 & 1-5\cdot 1/7 \end{array}\right]$$

$$\downarrow$$

$$\left[\begin{array}{ccc|c} 7 & 8 & 9 & 5 \\ 0 & 3/7 & 6/7 & 1/7 \\ 0 & 6/7 & 12/7 & 2/7 \end{array}\right] \longrightarrow \left[\begin{array}{ccc|c} 7 & 8 & 9 & 5 \\ 0 & 3/7 & 6/7 & 1/7 \\ 0 & 3/7 & 6/7 & 1/7 \end{array}\right]$$

Which do we eliminate?

Simple Gaussian elimination fails at this point because the we will end up with a zero row.

c. MATLAB when using the backslash operator will issue a warning if rcond is between 0 and eps but will proceed with the calculation according to the MATLAB website. This solution is an approximation and can highly vary in accuracy depending on the conditions of the matrix. However, the accuracy can vary wildly as on the MATLAB they give an example with a low residual and a 0 rcond which results in an inf as an output

4.

$$\left[\begin{array}{cccc} 1 & 0 & 0 & 0 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \\ 1 & 4 & 16 & 64 \end{array}\right] \left[\begin{array}{c} c0 \\ c1 \\ c2 \\ c3 \end{array}\right] = \left[\begin{array}{c} 7 \\ 11 \\ 28 \\ 63 \end{array}\right]$$

a. L0(x) = [(x-2)(x-3)(x-4)] / [(0-2)(0-3)(0-4)] = (x-2)(x-3)(x-4)/-24
…
p(x) = 7*L0 + 11 *L2 + 28*L3 + 63*L4
p(x) = $x^3 - 2x + 7$

b. p(x) = 7 + 2(x-0) + 5(x-0)(x-2) + 1(x-0)(x-2)(x-3)
p(x) = $x^3 - 2x + 7$

5.

$$(\sqrt{x})^{n} = (-1)^{n-1} \frac{1}{2^n}(2n-3)!! \cdot x^{-n+1/2}$$

Basis $n=1$

$$\sqrt{x}^{(1)} = (-1)^{1-1}\cdot \frac{1}{2^1}\cdot(2(1)-3)!!\cdot x^{-1+1/2}$$

$$\frac{1}{2\sqrt{x}} = \frac{1}{2}\cdot\frac{1}{\sqrt{x}} \checkmark$$

Basis $n=K$

$$\sqrt{x}^{k} = (-1)^{k-1}\cdot\frac{1}{2^k}\cdot(2k-3)!!\cdot x^{-k+1/2}$$

Prove true for $n=k+1$ (calculated)

$$\sqrt{x}^{(k+1)} = (-1)^{k-1}\cdot\frac{1}{2^k}\cdot(2k-3)!!\cdot(-k+1/2)\,x^{-k-1/2}$$

$$\sqrt{x}^{(k+1)} = (-1)^{k-1}\cdot\frac{1}{2^k}\cdot(2k-3)!!\,\frac{(-2k+1)}{2}\,x^{-(k+1/2)}$$

$$= (-1)^{k-1}\cdot\frac{1}{2^{k+1}}\cdot(2k-3)!!\,(-2k+1)\,x^{-(k+1/2)}$$

$$= (-1)^{k}\cdot\frac{1}{2^{k+1}}\cdot(2k-3)!!\,(2k-1)\,x^{-(k+1/2)}$$

$$= (-1)^{k}\cdot\frac{1}{2^{k+}}\cdot(2k-1)!!\cdot x^{-(k+1/2)}$$

$$(2k-3)!!\cdot(2k-1) = (2k-1)!!$$

$\sqrt{x}^{k+1}$ from Formula

$$= (-1)^{k+1-1}\cdot\frac{1}{2^{k+1}}\cdot(2(k+1)-3)!!\cdot x^{-(k+1)+1/2}$$

$$= (-1)^{k}\cdot\frac{1}{2^{k+1}}\cdot(2k-1)!!\cdot x^{-(k+1/2)}$$

$\therefore$ The Calculated and formula values are identical thus $n=k+1$ holds true and induction is true.

$$|R_n(x)| \le \frac{h^{n+1}}{4(n+1)} \max\left|f^{(n+1)}(\xi)\right|$$

$n=4$ $h=0.25$

* For equally spaced points which is true

$$|R_n(x)| \le \frac{0.25^5}{20} \max\left|f^5(\xi)\right|$$

$$-\sqrt{x}^5 = (-1)^3 \cdot \frac{1}{2^4}(5)!! \cdot x^{-7/2}$$

Question 5

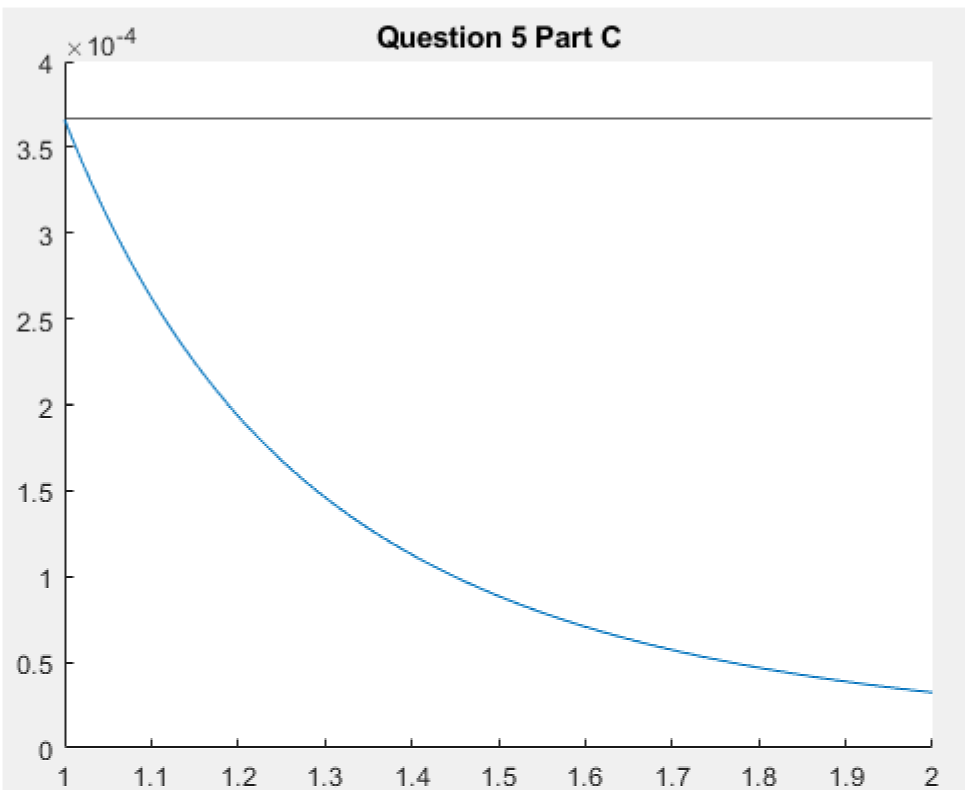
$$\max\left|(-1)^3 \cdot \frac{1}{2^4} \cdot (5)!! \cdot x^{-7/2}\right| \quad x=1$$

$$\max = 7.5$$

$$R_n(x) \le \frac{0.25^5}{20} \cdot 7.5$$

$$\le 3/8192$$

c.



The thin black line is the maximum error bounds.

6.

Question 6

a)

$$\frac{32\sqrt{2}x^3}{5\pi^3} - \frac{16x^3}{3\pi^3} - \frac{64\sqrt{2}x^2}{5\pi^2} + \frac{12x^2}{\pi^2} + \frac{24\sqrt{2}x}{5\pi} - \frac{20x}{3\pi} + 1 = P(x)$$

b) To get bands take $|P(x) - \cos(x)|$ for range $0, \frac{Pi}{2}$ and max is for bands

This error is on top of already existing cos approximation.

Treat as evenly spaced with $h = Pi/4$ $n = 3$ on bounds of $0, 3Pi/4$

*added $3Pi/4$ for additional accuracy

$$\frac{h^{n+1}}{4(n+1)} \max \left| f^{(n+1)}(\xi) \right| = \frac{(Pi/4)^4}{16} \cdot \max |$$

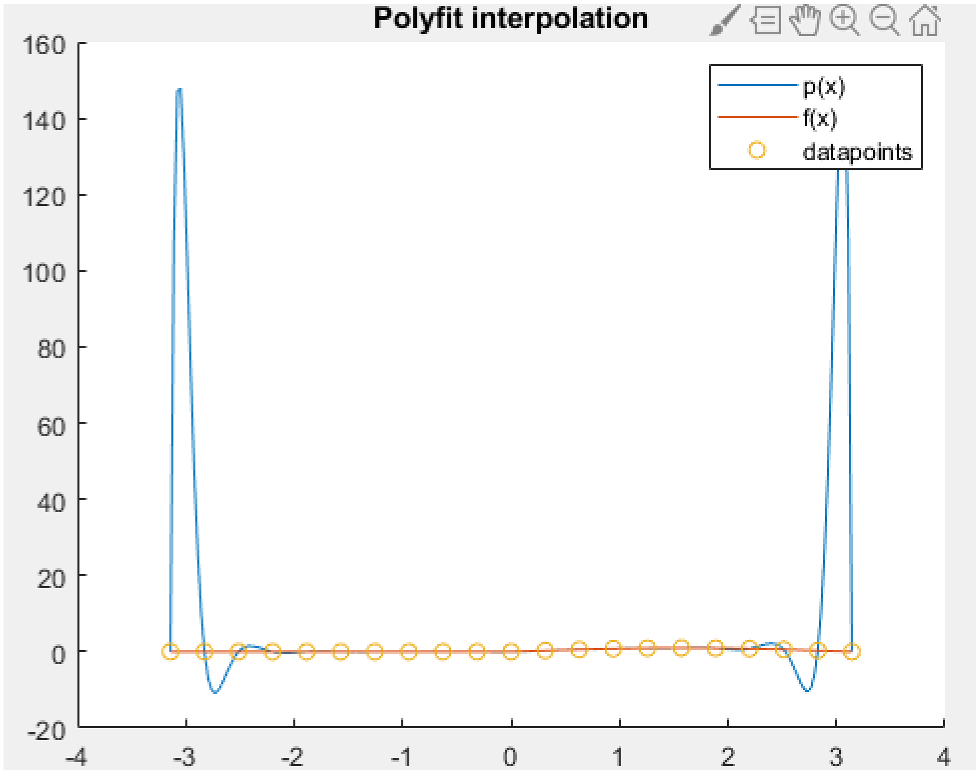
$$\leq Pi^4/4096$$ is error compared to $\cos(x)$

This value is much worse than the actual error because the max value of $abs(P(x) - \cos(x))$ in matlab was < 0.004 which is much less than $Pi^4/4096$

↳ Accuracy is $\frac{Pi^4/4096}{\cos(Pi/6)} \cdot 100\%$ which gives the absolute worse case scenario percentage effect that the approximation could have. In reality it will be much lower.

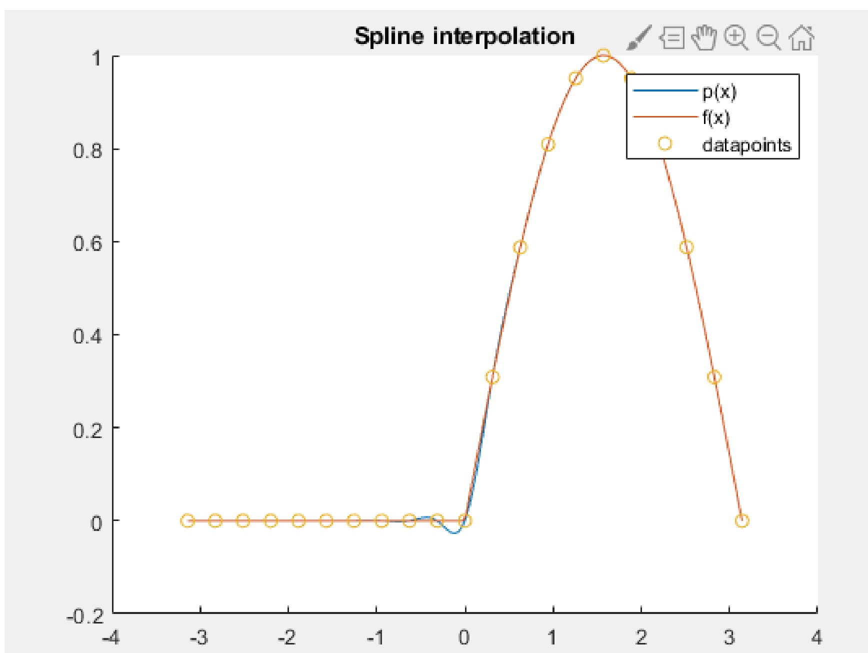
7.

a.



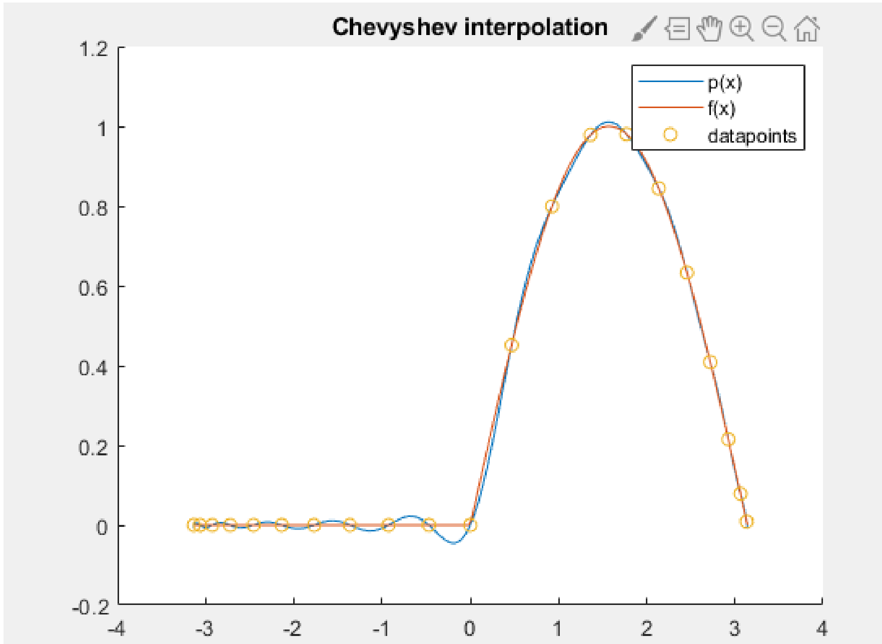
Polyfit interpolation
p(x)
f(x)
datapoints
160
140
120
100
80
60
40
20
0
-20
-4
-3
-2
-1
0
1
2
3
4

b.



Spline interpolation
p(x)
f(x)
datapoints
1
0.8
0.6
0.4
0.2
0
-0.2
-4
-3
-2
-1
0
1
2
3
4

c.



d. The difference between part a and part c is that the Chebyshev points remove the error at the bounds of the function reducing the overall error.

Appendix A: Code for Question 1

solve_all.m

```
function [xm, xpp, xspp] = solve_all(A,b)
xm = A\b;
[n, m] = size(A);
if n ~= m
   disp("ERROR: Matrix not N x N")
   return
end
[L,U,P] = lu_pp(A);
inv_P = inv(P);
B = b;
for i=1:n-1
   max_value=max(abs(inv_P(i:n,i))); % Find the pivot row
   for j=i:n
      if(abs(inv_P(j,i))== max_value)
         swap_index=j;  % Get the swap index
         break;
      end
   end
   inv_P([i,swap_index],:) = inv_P([swap_index,i],:);
   B([i,swap_index],:) = B([swap_index,i],:);


end
   [LS,US,PS] = lu_spp(A);
   inv_PS = inv(PS);
   BS = b;
   for i=1:n-1
```

```
        max_value=max(abs(inv_PS(i:n,i))); % Find the pivot row
        for j=i:n
            if(abs(inv_PS(j,i))== max_value)
                swap_index=j;  % Get the swap index
                break;
            end
        end
        inv_PS([i,swap_index],:) = inv_PS([swap_index,i],:);
        BS([i,swap_index],:) = BS([swap_index,i],:);
    end
    xpp = lu_solve(L,U,B);
    xspp = lu_solve(LS,US,BS);
end
```

lu_spp.m

```
function [L, U, P] = lu_spp(A)

[m, n] = size(A);

if n ~= m

   disp("ERROR: Matrix not N x N")

   return

end

% Create matrices

L=eye(n);

P=eye(n);

U=A;

ratio_vector = max(abs(A'));  % Create the ratio vector

for i=1:n-1

   temp = U;

   div_temp = temp ./ ratio_vector'; % Divide the matrix by the transpose of this vector

   max_value=max(abs(div_temp(i:n,i))); % Find the mmax column value

   for j=i:n

      if(abs(div_temp(j,i))== max_value)

         swap_index=j;  % Get the swap index

         break;

      end

   end

   % Swap rows in matrix

   U([i,swap_index],i:n) = U([swap_index,i],i:n);

   L([i,swap_index],1:i-1) = L([swap_index,i],1:i-1);

   P([i,swap_index],:) = P([swap_index,i],:);

   % Elimination

   for j=i+1:n

      L(j,i)=U(j,i)/U(i,i);

      U(j,i:n)=U(j,i:n)-L(j,i)*U(i,i:n);

   end
```

```
end

end
```

lu_solve.m

```
function x = lu_solve(L,U,b)

y = inv(L) * b;

x = inv(U) * y;

end
```

lu_pp.m

```
function [L, U, P] = lu_pp(A)
[m, n] = size(A);
if n ~= m
    disp("ERROR: Matrix not N x N")
    return
end
% Create matrices
L=eye(n);
P=eye(n);
U=A;
for i=1:n-1
    max_value=max(abs(U(i:n,i))); % Find the pivot row
    for j=i:n
        if(abs(U(j,i))== max_value)
            swap_index=j;  % Get the swap index
            break;
        end
    end
    % Swap rows in matrix
    U([i,swap_index],i:n) = U([swap_index,i],i:n);
    L([i,swap_index],1:i-1) = L([swap_index,i],1:i-1);
    P([i,swap_index],:) = P([swap_index,i],:);
    % Gaussian elimination
    for j=i+1:n
        L(j,i)=U(j,i)/U(i,i);
        U(j,i:n)=U(j,i:n)-L(j,i)*U(i,i:n);
    end
end
end
```

linearsolve.dat

n	cond(A)	Matlab		lu_pp		lu_spp	
		error	residual	error	residual	error	residual
Vandermonde matrices (flipped with fliplr)							
5	2.6e+04	0.0e+00	0.0e+00	7.1e-15	2.6e-13	0.0e+00	0.0e+00
6	7.3e+05	4.1e-11	7.8e-13	4.9e-11	2.4e-12	3.4e-11	1.9e-12
7	2.4e+07	3.7e-11	2.3e-13	8.9e-10	2.0e-10	2.0e-09	2.4e-10
8	9.5e+08	3.5e-09	1.7e-10	6.0e-08	1.1e-08	5.1e-08	1.5e-09
9	4.2e+10	1.1e-07	3.6e-09	1.2e-06	2.0e-08	1.2e-06	3.4e-07
10	2.1e+12	1.7e-05	2.2e-07	1.1e-04	4.6e-05	2.6e-05	5.7e-06
11	1.2e+14	2.2e-03	1.9e-06	3.8e-03	1.4e-03	2.2e-03	1.8e-03
12	7.0e+15	3.8e-01	1.9e-04	3.1e-01	1.6e-02	4.6e-02	5.3e-01
13	1.1e+18	1.5e+00	2.5e-03	2.3e+01	7.2e-01	6.6e+01	2.8e+01
14	2.1e+18	2.1e+02	9.5e-02	1.6e+03	1.2e+02	9.7e+02	5.2e+03
15	9.9e+19	9.4e+04	8.7e+00	1.1e+05	3.2e+04	1.8e+05	1.6e+05
Hilbert matices							
2	1.9e+01	9.0e-16	0.0e+00	6.0e-16	0.0e+00	6.0e-16	0.0e+00
3	5.2e+02	1.8e-14	2.2e-16	1.2e-15	2.2e-16	1.2e-15	2.2e-16
4	1.6e+04	5.4e-13	2.5e-16	1.5e-12	2.2e-16	7.8e-13	5.6e-16
5	4.8e+05	6.3e-12	2.5e-16	8.4e-13	6.0e-16	6.2e-13	7.5e-16
6	1.5e+07	6.8e-10	3.1e-16	9.8e-11	1.6e-16	1.3e-09	1.1e-16
7	4.8e+08	3.0e-08	5.0e-16	1.6e-08	9.6e-16	5.3e-09	9.4e-16
8	1.5e+10	6.7e-07	5.2e-16	9.8e-07	2.4e-15	7.3e-07	4.7e-16
9	4.9e+11	2.8e-05	5.6e-16	2.5e-05	2.8e-15	5.9e-05	7.9e-15
10	1.6e+13	5.2e-04	5.2e-16	1.7e-03	1.8e-15	9.0e-04	2.4e-14
11	5.2e+14	1.5e-02	4.2e-16	3.2e-02	8.6e-15	2.2e-02	6.0e-15
12	1.7e+16	9.0e-01	7.3e-16	4.8e-01	2.9e-14	1.3e+00	4.0e-14
13	6.9e+17	9.4e+00	7.9e-16	7.2e+00	2.6e-14	8.5e+00	9.4e-14
14	4.3e+17	9.5e+00	6.9e-16	1.5e+02	2.7e-13	1.4e+01	4.9e-14
15	4.5e+17	1.5e+01	1.3e-15	1.4e+01	6.5e-13	1.8e+01	3.7e-14
Random matrices							

10	6.1e+01	5.0e-15	2.0e-15	1.3e-14	5.3e-15	1.3e-14	4.8e-15
100	2.8e+03	9.1e-13	1.8e-13	6.6e-13	9.5e-13	1.4e-12	1.5e-12
1000	2.9e+04	3.5e-11	2.2e-11	1.9e-10	6.0e-10	2.3e-10	1.2e-09
2000	3.0e+05	5.9e-10	1.2e-10	8.4e-10	4.0e-09	1.0e-09	4.5e-09