Scientific Computing HW2

Runze Fang

October 4, 2020

1 The Hilber Matrix

1.1 Conditioning Numbers

Fristly I compared direct calculation of condition number with the built-in exact calculation cond.

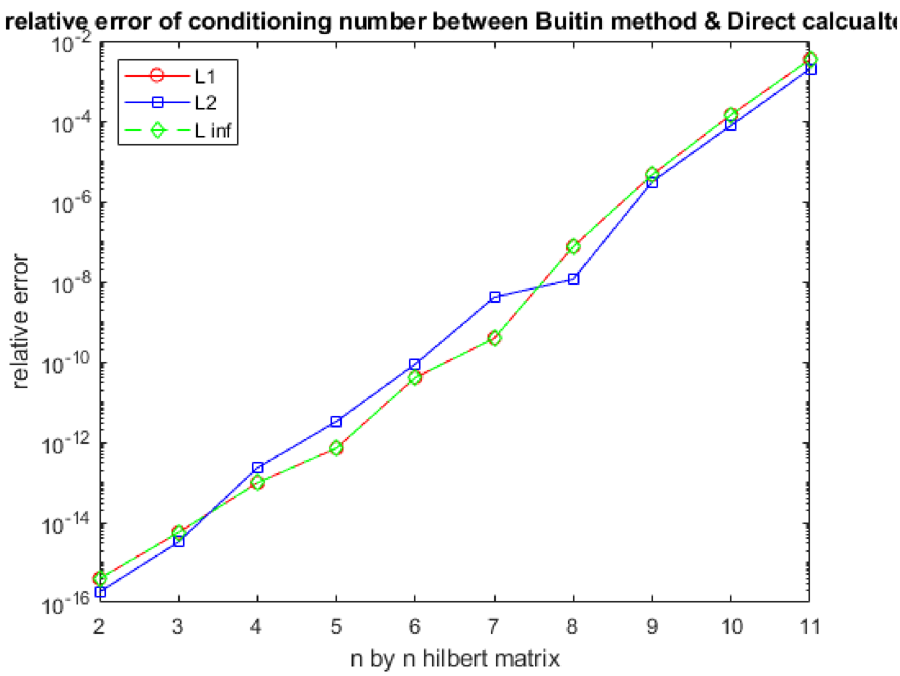

Figure 1: condition number comparation

As shown in Figure 1, the relative error of computing condition number directly increase with the size of hilbert matrix increasing. Each time the hilbert matrix size increases, we lose 2 digits of accuracy. The graph also shows that the built in function has the same result with direct calculating when calculating L1 and L infinite , while it does not when calculating L2. This observation in accordance with Matlab document: when doing L2, cond method use Singular Value Decomposition to do the calculation.

Then I compared direct calculation of reciprocal condition number with the estimate rond number.

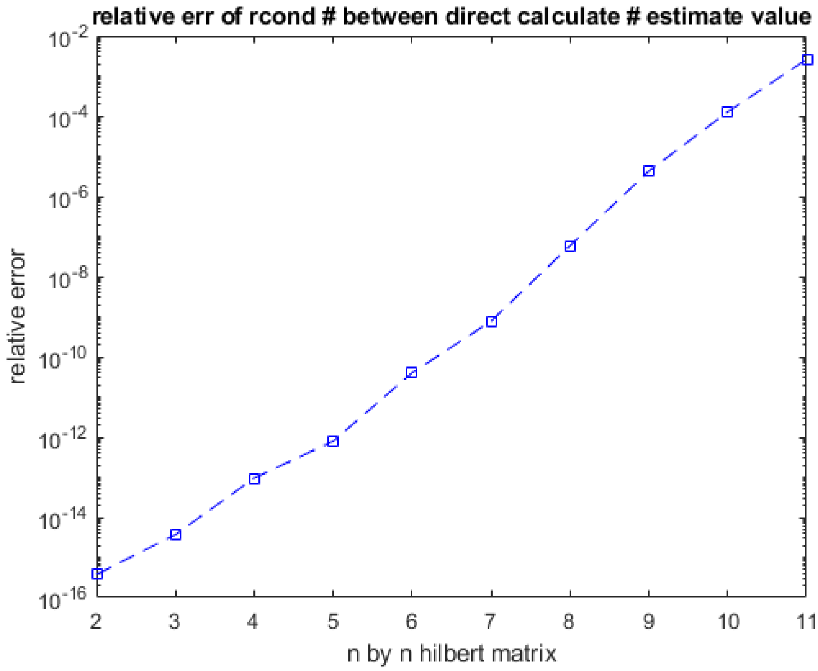

Figure 2: reciprocal condition number comparation

As shown in Figure 2, the relative error of computing reciprocal condition number increase with the size of hilbert matrix increasing. Each time the hilbert matrix size increases, we also lose 2 digits of accuracy.

1.2 Solving ill-conditioned systems

I first solve the linear system by using the built-in solver: *linsolve*. According to the document, it use LU factorization to solve the system. Then I calculate the relative error using the infinite norm. I did the same procedure using Cholesky factorization. Also, I compute the residual under these two senario.

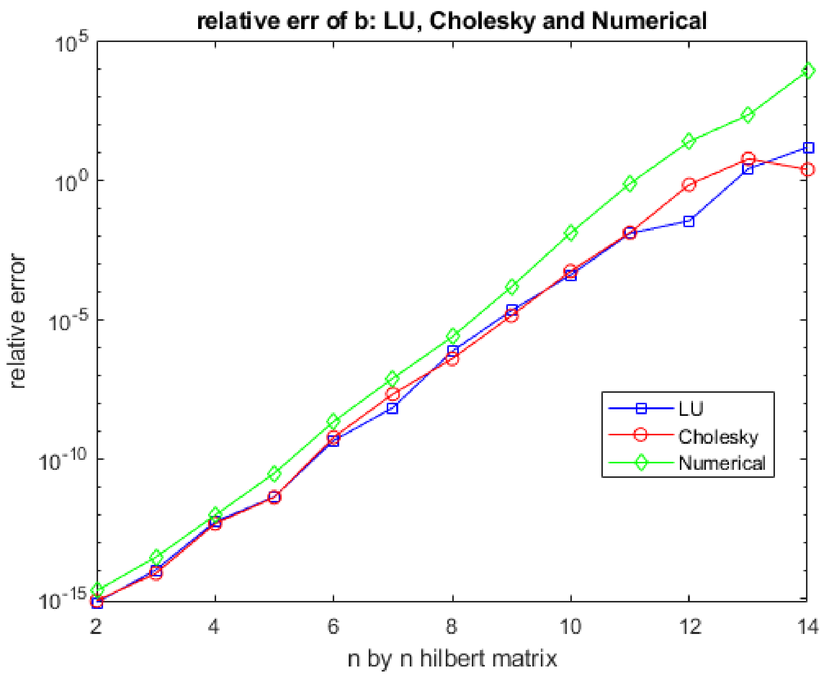

Figure 3: relative error of b by using factorization and numerical computing

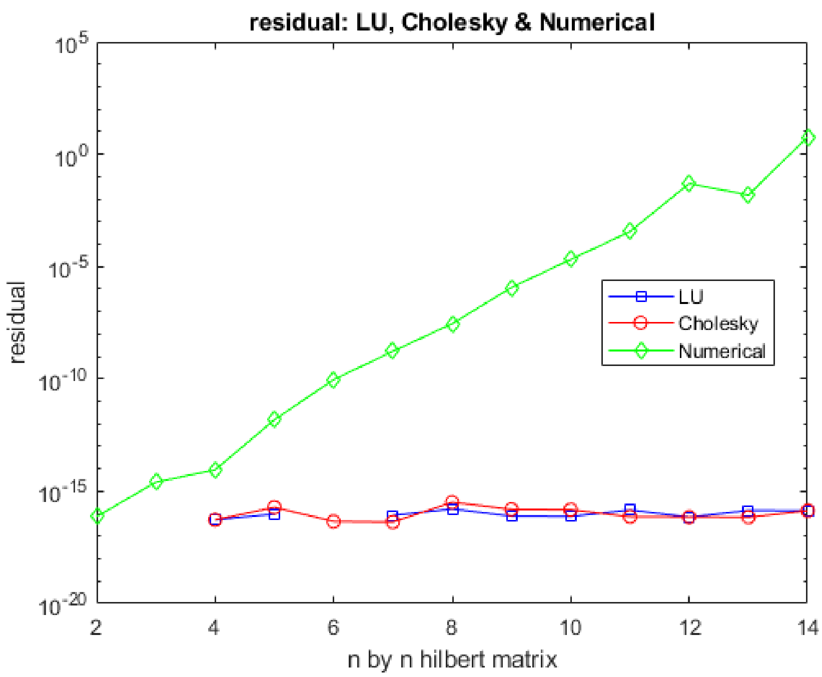

Figure 4: residual by using factorization and numerical computing

As shown in Figure 3, the number of digit accuracy decreases as the size of hilbert matrix increasing. Every time the size of Hilbert matrix increase by 1, we lose 1 to 2 digits of accuracy. When $n > 12$ we canot get any digit of accuracy. The relative error diverges after 6×6 hilbert matrix. This observation comforms with what we can see in Figure 4: the residual calculated by LU factorization of 6×6 hilbert matrix equals 0, which is abnormal. The trend of residual value conformed with the trend of condition number: when $n \leq 3$, the condition number is relatively large, the matrix is reletive stable, therefore, the residuals are 0; when $n \geq 4$, there are residuals. As for conclusion, LU factorization and Cholesky factorization has nearly the same performance: they share the same large of residual, and close relative error. However, after 6 by 6 hilbert matrix, Choleskey factorization seems slightly more stable.

The result confoms to the theoretical expectation discussed in class. After $n = 12$, it no longer make sense to even try solving the system due to the sever ill-condition

Then I compute the solution by using the numerically-computed matrix inverse function *matinv*. Also shown in Figure 3 and 4, I compared numerical method, LU and Cholesky factorization together. Considering the relative error, the numerical method lose at least one more digit of accuracy than both factorizations. As the size of hilbert matrix grows, the numerical method will lose more digits of accuracy. Considering the residual, numerical method has way larger residual then the factorizations have. Therefore, LU and Cholesky factorizations are way better than the numerical method.

Finally, I compared the *matinv* function and *invhilb* function.

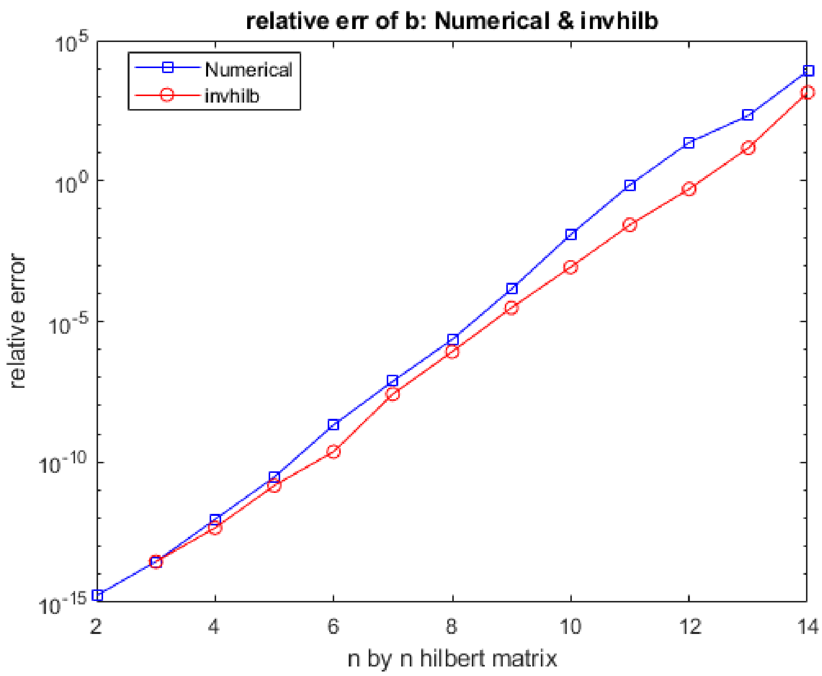

Figure 5: relative error of different numerical method

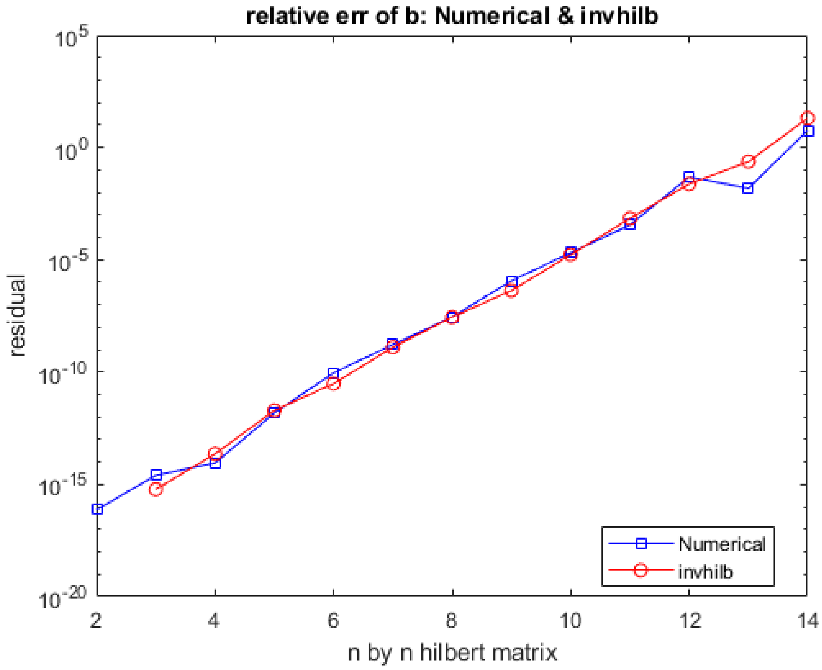

Figure 6: residual by using different numerical method

As shown in Figure 5 and Figure 6, *matinv* and *invhilb* has the same large residuals. Comparing the relative errors, when the size of the matrix is larger than 6, the *invhilb* gets at least 1 digits of accuracy. Therefore, using *invhilb* is more optimal when doing numerical computing. This might be because 3 sets of roundoff errors. Firstly, when doing $inv(hilb(n))$, there will be error representing $hilb(n)$. Secondly, when doing inversion, there will be errors. Finally, there is error when representing $invhilb(n)$. The *invhilb* method may have less error in the first senario.

2 Different Methods

2.1 Three-method fitting

I estimate $\tilde{c}$ and generate $\|c - \tilde{c}\|$ for different ϵ in three ways: using the built-in function $polyfit$, using backslash operator and forming the system of normal equations.

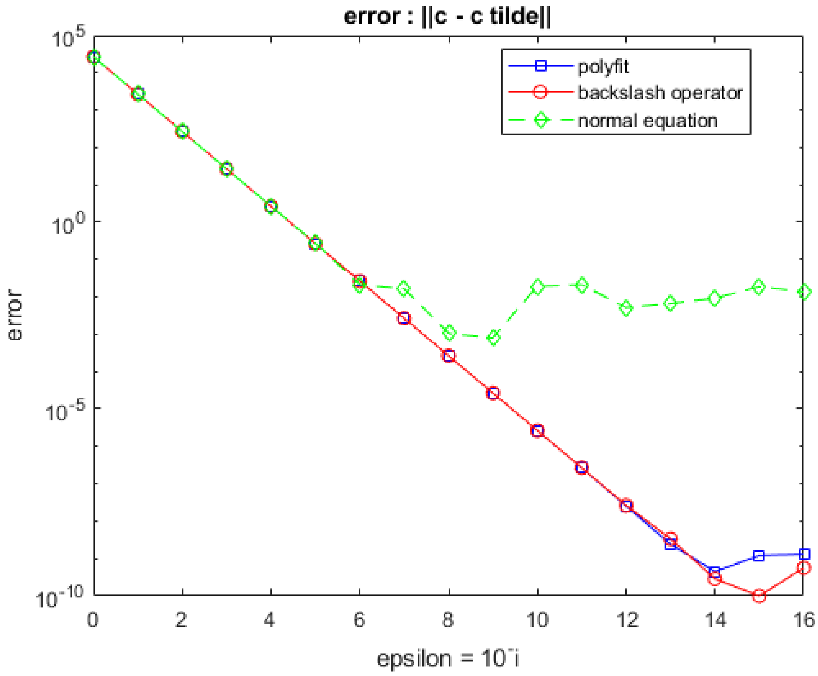

Figure 7: error: norm of (c - c tilda)

c_value_pf | c_tilta_bo | c_value_ne

10x17 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	9.5332e+03	961.4239	104.2424	18.5242	9.9524	9.0952	9.0095	9.0010	9.0001	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000
2	-5.2555e+04	-5.2483e+03	-517.6326	-44.5633	2.7437	7.4744	7.9474	7.9947	7.9995	7.9999	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000
3	1.2020e+05	1.2026e+04	1.2089e+03	127.1943	19.0194	8.2019	7.1202	7.0120	7.0012	7.0001	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000
4	-1.4831e+05	-1.4826e+04	-1.4772e+03	-142.3161	-8.8316	4.5168	5.8517	5.9852	5.9985	5.9999	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000
5	1.0737e+05	1.0742e+04	1.0787e+03	112.3659	15.7366	6.0737	5.1074	5.0107	5.0011	5.0001	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000
6	-4.6181e+04	-4.6145e+03	-457.8455	-42.1846	-0.6185	3.5382	3.9538	3.9954	3.9995	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000
7	1.1310e+04	1.1337e+03	116.0698	14.3070	4.1307	3.1131	3.0113	3.0011	3.0001	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000
8	-1.3870e+03	-136.9012	-11.8901	0.6110	1.8611	1.9861	1.9986	1.9999	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
9	62.5702	7.1570	1.6157	1.0616	1.0062	1.0006	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
10	-0.2263	-0.0226	-0.0023	-2.2632e-04	-2.2632e-05	-2.2632e-06	-2.2632e-07	-2.2632e-08	-2.2632e-09	-2.2631e-10	-2.2627e-11	-2.2495e-12	-2.3667e-13	1.6514e-14	-1.6056e-14	-1.4526e-15	-1.7508e-14

Figure 8: estimate coefficient by polyfit

c_value_pf | c_value_bo | c_value_ne

10x17 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	-0.2263	-0.0226	-0.0023	-2.2632e-04	-2.2632e-05	-2.2632e-06	-2.2632e-07	-2.2632e-08	-2.2632e-09	-2.2631e-10	-2.2633e-11	-2.2469e-12	-2.2960e-13	1.0786e-14	-1.4770e-14	-5.8356e-18	-1.5912e-14
2	62.5702	7.1570	1.6157	1.0616	1.0062	1.0006	1.0001	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	-1.3870e+03	-136.9012	-11.8901	0.6110	1.8611	1.9861	1.9986	1.9999	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
4	1.1310e+04	1.1337e+03	116.0698	14.3070	4.1307	3.1131	3.0113	3.0011	3.0001	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000	3.0000
5	-4.6181e+04	-4.6145e+03	-457.8455	-42.1846	-0.6185	3.5382	3.9538	3.9954	3.9995	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000	4.0000
6	1.0737e+05	1.0742e+04	1.0787e+03	112.3659	15.7366	6.0737	5.1074	5.0107	5.0011	5.0001	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000	5.0000
7	-1.4831e+05	-1.4826e+04	-1.4772e+03	-142.3161	-8.8316	4.5168	5.8517	5.9852	5.9985	5.9999	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000	6.0000
8	1.2020e+05	1.2026e+04	1.2089e+03	127.1943	19.0194	8.2019	7.1202	7.0120	7.0012	7.0001	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000	7.0000
9	-5.2555e+04	-5.2483e+03	-517.6326	-44.5633	2.7437	7.4744	7.9474	7.9947	7.9995	7.9999	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000	8.0000
10	9.5332e+03	961.4239	104.2424	18.5242	9.9524	9.0952	9.0095	9.0010	9.0001	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000	9.0000

Figure 9: estimate coefficient by backslash

c_value_pf | c_value_bo | c_value_ne

10x17 double

	1	2	3	4	5	6	7	8	9
1	-0.2259	-0.0226	-0.0023	-2.2587e-04	-2.2563e-05	-2.3105e-06	-3.0077e-07	2.9205e-08	-5.2104e-08
2	62.5505	7.1551	1.6155	1.0615	1.0062	1.0006	1.0001	1.0000	1.0000
3	-1.3867e+03	-136.8677	-11.8868	0.6114	1.8612	1.9861	1.9985	1.9999	1.9999
4	1.1307e+04	1.1334e+03	116.0427	14.3039	4.1302	3.1134	3.0118	3.0008	3.0004
5	-4.6169e+04	-4.6133e+03	-457.7258	-42.1709	-0.6162	3.5368	3.9516	3.9969	3.9981
6	1.0734e+05	1.0738e+04	1.0783e+03	112.3303	15.7308	6.0772	5.1132	5.0068	5.0047
7	-1.4826e+05	-1.4821e+04	-1.4767e+03	-142.2601	-8.8224	4.5112	5.8425	5.9913	5.9929
8	1.2016e+05	1.2022e+04	1.2085e+03	127.1418	19.0107	8.2072	7.1288	7.0063	7.0064
9	-5.2532e+04	-5.2460e+03	-517.3959	-44.5364	2.7482	7.4717	7.9430	7.9977	7.9968
10	9.5281e+03	960.9155	104.1914	18.5185	9.9515	9.0958	9.0105	9.0003	9.0007

	10	11	12	13	14	15	16	17
1	-1.2333e-08	2.4056e-08	-1.2327e-08	-2.0744e-07	-2.7709e-08	-5.0503e-08	-1.1943e-07	-1.0412e-07
2	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	2.0000	2.0000	2.0000	1.9998	2.0000	2.0000	1.9999	1.9999
4	3.0001	2.9998	3.0001	3.0014	3.0002	3.0003	3.0008	3.0007
5	3.9996	4.0009	3.9996	3.9938	3.9992	3.9986	3.9966	3.9970
6	5.0010	4.9977	5.0010	5.0161	5.0021	5.0036	5.0087	5.0076
7	5.9985	6.0038	5.9984	5.9747	5.9967	5.9944	5.9865	5.9882
8	7.0013	6.9964	7.0015	7.0237	7.0030	7.0052	7.0125	7.0109
9	7.9993	8.0019	7.9992	7.9879	7.9985	7.9974	7.9937	7.9945
10	9.0001	8.9996	9.0002	9.0026	9.0003	9.0006	9.0014	9.0012

Figure 10: estimate coefficient by norm equations

As shown in Figure 7, the error computed by $polyfit$ and backslash operator performs the same error. These two senario diverges from $\epsilon = 10^{-14}$. This may because when ϵ is close to the maximum accuracy of double, the roundoff error occurs. The error of normal equation diverges from the above two methods when $i \geq 6$. This may because of when doing $A^T A$ and $A^T y$, we have a huge round off error. In contrast, we did not perform any multiplication using the backslash operator to solve the linear system. Therefore, the normal equation method has a big error when $i \geq 6$.

2.2 The Best Method

If $\epsilon = 0$, we get the exact result of fitting. I compute the exact result of fitting using 3 method and the result is shown below.

```
when epsion = 0,
the error of polyfit is 3.33151e-10
the error of backslash is 5.53899e-10
the error of normal equation is 2.796758e-03.
```

Figure 11: exact result of fitting

As shown in Figure 11, the highest accuracy I can achieve is 10 digits of accuracy. The $polyfit$ method and the backslash method share the same performance, while the normal equation is obvious inferior to the previos two methods. This result conforms with my discussion in 2.1: the normal equation method has huge roundoff error due to the multiplication of $A^T A$ and $A^T y$.

Empirically, as the degree d increases, the system becomes more ill-conditioned. The more degrees means the $A^T A$ matrix becomes ill-conditioned and as our computation becomes more complex, more roundoff error are introduced. As a result, $(A^T A)^{-1}$ will be estimated with a huge error. Thus, the overdetermined linear system becomes more ill-conditioned.

3 Rank-1 Matrix Updates

3.1 Direct Update

I generate a 5×5 matrix A and a rhs by using $randn$, and then I solve $Ax = b$. Then I generate two vetors u and v to updated the system, then compute the residual $r = b - \tilde{A}\tilde{x}$. The generated matrices and the residual are shown below.

```
generate 5x5 A and 5x1 b using randn
A =
0.288342 1.39185 -1.3455 0.000747406 0.0534948
-2.34191 1.2481 2.80923 -0.232185 0.287032
-0.464604 0.384318 -0.379828 -0.101778 1.61815
-0.84723 -0.575883 -0.0759131 -0.132472 1.43931
2.03126 -0.978202 -0.641024 0.520476 -0.3807
b =
1.99285
 1.63587
 0.559019
 0.563638
 -0.338397

solve Ax = b, x =
-0.224784
 0.58175
 2.23124
 -2.55252
 1.12942

now generate u and v to get A tilde, solve for x tilde.
 the residual = 5.795534e-16
```

Figure 12: generated matrices, x and residual r

As shown in Figure 12, the residual is around $5e - 16$, which is close to the lower bundary of double and it is very small.

3.2 SMW Formula

I generated a 100×100 matrix A and compute $\tilde{x}$ in two ways: direct computing and using SMW formula. Then I compute the norm of these two $\tilde{x}$. The result is shown below.

```
generate a 100 by 100 A, compare the x tilde between two computing method
the norm between them is 4.483405e-14
```

Figure 13: norm of x tilde computed in different ways

The norm between two $\tilde{x}$ computed in two diferent ways is about $4e - 14$, which indicates the gap between them is not big.

If we use the SMW formula to compute $\tilde{x} = \tilde{A}^{-1}b$, we can solve it more efficiently giving the condition that we has already know the LU factorization of A. This will requires $O(n^2)$ work. Using the SMW formula we get $\tilde{x} = \tilde{A}^{-1}b = A^{-1}b - \frac{A^{-1}uv^TA^{-1}b}{1+v^TA^{-1}u}$.

Let $Az = u$ for z, so $z = A^{-1}u$. This will have $FLOPS \approx 2n^2$, since the forward substitution and the backward substitution will have a $FLOPS \approx n^2$ for each. And let $Ay = b$ for y, so $y = A^{-1}b$. This will also have $FLOPS \approx 2n^2$. These two procedure will have a total $FLOPS \approx 4n^2$.

Now we can compute $\tilde{x} = y - \frac{zv^Ty}{1+v^Tz}$. The procedure zv^T will have $FLOPS \approx n^2$ because it is a $n \times 1$ matrix multiply a $1 \times n$ matrix and get an $n \times n$ matrix. Then, the procedure zv^Ty will have a $FLOPS \approx n(2n-1)$. Similarly, to perform v^Tz, we will have $FLOPS \approx 2n - 1$. Adding all flops together with one adding operation, one subtractiong operaton and one dividing operation, we get a total $FLOPS \approx 7n^2 + 2n$.

In comparison, solving $\tilde{A}\tilde{x} = b$ directly will have a $FLOP \approx \frac{2}{3}n^3 + 4n^2$, since it has one LU fratorization and two backslash operations.

In conclusion, computing directly is $O(n^3)$, while the implementation of SMW formula is $O(n^2)$, the later is way faster than the former. This is because if A is already factored, we only do triangular solutions and inner products, while when we compute directly, we will do one more step of explicity inverses.