Foundations of Computational Math
Program 3
Chenchen Zhou

1 Homework description

In this homework, we:

A Write a routine to solve a particular family of linear systems (n × n tridiagonal Toeplitz matrix)via the Jacobi, Gauss Seidel, and Symmetric Gauss Seidel methods.

B We also write a test routine separately to test the correcteness and the convergence rate of specific methods. We also analyze the results and looking for trends and outliers of iteration in behavior with the all kinds of parameters.

2 Alogrithm

1.We do not need to store the whole matrix, for each n × n tridiagonal Toeplitz matrix, we only need to store α. That means we only need to store O(1) number to characterize the specific Toeplitz matrix.

$$T_\alpha = \begin{pmatrix} \alpha & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & \alpha & -1 & 0 & \dots & \dots & 0 \\ 0 & -1 & \alpha & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & \alpha & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & \alpha & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & \alpha \end{pmatrix}$$

2. Computation of the matrix vector product Av → w, as the A here is the tridiagnal Toeplitz matrix, so the complexity of computaion of the matrix vector product must be O(n).

3. storage scheme. we only need a nx1 vector space to to store when updating X in every component. As we need to compute the norm of the zk. We need extra parameter tempx1,tempx2 or just tempx to store the old component of X. caculate the difference between Xk+1-Xk of every component and then sum them up in each step.

4. $\frac{\|x_{final} - x_{true}\|_2}{\|x_{true}\|_2} \leq \epsilon$ use this accurancy to determine the termination of the iteration when we know the true solution of AX=b. Also, we need a parameter which denotes the maxmium of iteration which is used to stop the iteration in case of the iteration method can't reach the required accuracy or it just can't converge to the Xtrue.

When solving Ax=b by using iteration method:

1 Input a,b,X0,epsilon,Xtrue,flag.

a is for α ,X0 is the initial value of X. epsilon the the tolernce error which helps to determine the termination of the iteration. Flag is for the three iteration methods.flag=1 is Jocobi method; flag=2 is Gauss Seidel method; flag=3 is the Symmetric Gauss Seidel method.

2. for i=1:n

compute Xnew(i);

caculate the (Xnew(i)-Xold(i))^2

caculate the (Xnew(i)-Xtrue(i))^2

end

3. get the Xnew via different iterative equation respect to different methods

caculate the norm (b-AXnew)

4 .collect the information of every iterative kth step

Ek $\|e^{(k)}\|_2/\|x_{true}\|_2 = \|x_k - x_{true}\|_2/\|x_{true}\|_2$

Zk $\|z_k\|_2/\|x_k\|_2$

rk $\|r_k\|_2/\|b\|_2$

5.When the epsilon reaches the acceptable range or the number of iteration reached the maxmium iteration number, we stop. And get the final answer X.

3 Instruction for routine test

We have 3 matlab M-files.

Iteration.M

function [X0,Zk,Ek,r,K,p]=Iteration(a,b,X0,epsilon,Xtrue,flag)

output is final computed solution X for AX=b. Here we still use the X0;

Zk is the value of $\|z_k\|_2/\|x_k\|_2$

Ek is the relative error of the computed X to the the actual solution to the AX=b

$\|x_k - x_{true}\|_2/\|x_{true}\|_2$

r is the $\|r_k\|_2/\|b\|_2$

K is the predicted iteration number according to our estimated spectral radium and the required accuracy.

p is estimated spectral radium

Particularly,storage scheme for three methods in loop

Jocobi:

We use a nx1 vector space to store the X. Firstly we have an initial X0, then we have a nx1 vector space. In the later iteration ,we will always use this nx1 space to store the generated X in the next step. When we update the X0(i) from 1 to n, according to the Jocobi method, we need to store the Xold(i) for 2 steps (when 1<i<n). Because Xold(i) will be used to caculate the Xnew(i-1) and Xnew(i+1). So we need two extra parameter to help to store the Xold.

For example, when we will caculate Xnew(i), we first save Xold(i) to tempX2, then update the number in X(i), we get Xnew(i), then we just pass the value of tempX2 to tempX1 which will be used to caculate the Xnew(i+1).

In order to be more efficently, we will caculate the (Xold(i)-Xnew(i))^2 instantly we get the Xnew(i), this will save the space and time.

Guass Seidel:

It is similar to Jocobi, the only difference is that in Gauss Seidel ,we only need one extra paremeter to help store the Xold(i), because when we get the Xnew(i), we don't need it any more in the next caculation. We create a Temp to save it just because we want to caculate the (Xold(i)-Xnew(i))^2.

Symmetric Guass Seidel:

In this case ,it is a little different. Because Symmetric Gauss Seidel is equivalently to apply the forward Gauss Seidel once and then apply the backward Gauss Seidel once. So if we want to compute the Zk, we must keep Xold along until we get every component of Xnew. So we need an extra nx1 vector space which is called temp.

2.test.m

function test(a,n,precision,epsilon,flag,initial)

this is a test for three methods due to different matrix, different size number of matrix ,different initial X, different precision,and different tolerence error bound.

a is for diagnal element of the matrix of A

n is the size of A

precision is for the input datatype.

if precision==1 single

if precision==2 double

epsilon is the torlerence error bound

flag is for the method you choose

if flag==1 Jocabi method

if flag==2 Gauss_Seidel method

if flag==3 Symetric Gauss_Seidel method

initial is the initial guess of the solution X.

if initial ==1 the initial X and the true X is randomly selected

if initail==2 the initial X=0 and the true X is e=e1+e2+…+en

3.tesecases.m

This file is used to test all kinds of cases. Actually, we have precision(2)*a(3)*n(3)*flag(3)*epsilon(3)=162 cases in total needed to be tested.

In order to observe specificly and more easily, I write this file to run the cases that we are interested in and investigate the situations get the outcomes which we need.

You need to do some changes in the code to look for the specific cases.For example, if we want to see all kinds of cases when ‘a’ equals to 3 and 4. We just set a=3, aa=2:3 or aa=1:2, then comes out all cases under a=3 and a=4 with all kinds precision, methods, size, accuracy.

Number in the file is to indicate the number of interation. So you can tell from the number to see the outcome belongs to which case.

This file just for test use, there is no other algorithms.

4 Test result and conclusion

P is the spectral radium, K is the predicted iteration number, count is the actual iteration number. Number is the secenios I have runed

a=2	p	K	count	accuracy	number
	0.9995	1.38E+04	1000	0.154	1
	0.9995	1.98E+04	1000	0.1007	2
	0.9995	3.22E+04	1000	0.032	3
	0.999	6.80E+03	1000	0.0018	4
	0.999	1.03E+04	1000	0.0082	5
	0.999	1.57E+04	1000	0.0171	6
	0.9981	3.07E+03	1000	0.0183	7
	0.9981	5.38E+03	1000	0.0115	8
	0.9981	8.81E+03	1000	0.0113	9
	1	1.32E+06	1000	0.2032	10
	1	1.94E+06	1000	0.1735	11
	1	3.27E+06	1000	0.1572	12
	1	6.66E+05	1000	0.0778	13
	1	1.00E+06	1000	0.082	14
	1	1.71E+06	1000	0.0929	15
	1	331124	1000	0.0925	16
	1	5.14E+05	1000	0.0802	17
	1	8.28E+05	1000	0.0648	18

When a=2, the spectral radius gets really close to 1. So the convergence rate is really slow. The iteration number all reached the maximum 1000. We can see that the K is approximately 10000~100000,so to get the required accuracy for a =2, all of the methods has a low rate. In other word,it will take a very very huge number of iterations to converge to the Xtrue with the required accuracy.

I stop the test at number=18, because when n gets large. It takes too long time for the matlab in my computer to give out the outcome in a time.

In order to be more efficient.I will not list all kinds of scenios and discuss them. Because there are 162 scenios in total. I just list some of them and show that it is actually in the case. You can also run other scenios or just run all of them.

a=3,4.

I didn't test n=100000 which if I test it . There should be 108 scenios in total. But the same reason as before. It took too long time for my computer to run the scenios at the same time. when n=100000. So I first ignore it. And just test several of them later in order to be more efficent. Here I test the 72 scenios. And below is the information I collected.

a=3,4					
	p	K	count	accuracy	number
	0.6663	15.9948	22	7.80E-05	1
	0.6663	23.4318	33	8.23E-07	2
	0.6663	38.4506	1000	8.48E-08	3
	0.444	8.8016	12	8.99E-05	4
	0.444	11.9333	19	5.50E-07	5
	0.444	21.3366	1000	7.54E-08	6
	0.1971	4.0553	6	7.94E-05	7
	0.1971	6.1209	10	4.89E-07	8
	0.1971	9.3873	1000	5.46E-08	9
	0.6667	16.3176	21	9.44E-05	10
	0.6667	24.6516	32	9.30E-07	11
	0.6667	39.9421	1000	8.34E-08	12
	0.4444	7.8629	13	5.21E-05	13
	0.4444	12.2078	19	7.30E-07	14
	0.4444	19.8672	1000	5.65E-08	15
	0.1975	4.0537	7	2.49E-05	16
	0.1975	6.0484	10	3.93E-07	17
	0.1975	10.0889	1000	5.66E-08	18
	0.4998	9.3291	13	6.15E-05	19
	0.4998	14.9416	19	8.51E-07	20
	0.4998	25.773	1000	6.05E-08	21

	0.2498	4.4611	8	9.94E-05	22
	0.2498	7.215	13	3.46E-07	23
	0.2498	12.7942	1000	4.42E-08	24
	0.0624	2.5607	4	8.34E-05	25
	0.0624	3.3812	6	8.63E-07	26
	0.0624	6.0474	1000	3.43E-08	27
	0.5	9.4683	13	6.96E-05	28
	0.5	13.9109	20	4.96E-07	29
	0.5	23.6265	1000	5.56E-08	30
	0.25	4.5937	8	8.84E-05	31
	0.25	7.1844	12	1.00E-06	32
	0.25	11.5895	1000	4.13E-08	33
	0.0625	2.318	4	8.13E-05	34
	0.0625	3.479	6	9.32E-07	35
	0.0625	5.7963	1000	4.06E-08	36
	0.6663	15.4193	21	8.14E-05	37
	0.6663	25.5617	32	7.52E-07	38
	0.6663	43.7681	54	8.78E-11	39
	0.444	8.1881	13	5.42E-05	40
	0.444	11.9869	19	6.28E-07	41
	0.444	20.817	32	5.99E-11	42
	0.1971	3.9117	6	8.90E-05	43
	0.1971	5.8984	10	3.66E-07	44
	0.1971	11.6211	16	4.09E-11	45
	0.6667	16.1688	21	9.49E-05	46
	0.6667	24.5492	33	6.81E-07	47
	0.6667	39.0054	55	8.36E-11	48

	0.4444	8.1206	13	4.95E-05	49
	0.4444	12.0043	19	7.40E-07	50
	0.4444	20.3341	32	7.49E-11	51
	0.1975	3.9847	7	2.67E-05	52
	0.1975	5.9713	10	3.62E-07	53
	0.1975	9.97	16	8.22E-11	54
	0.4998	9.2055	13	6.27E-05	55
	0.4998	13.4392	20	6.03E-07	56
	0.4998	27.1602	32	8.10E-11	57
	0.2498	4.8841	8	6.65E-05	58
	0.2498	6.7409	13	3.76E-07	59
	0.2498	12.6025	21	3.68E-11	60
	0.0624	2.2822	4	8.60E-05	61
	0.0624	3.6553	6	6.69E-07	62
	0.0624	5.5748	11	1.26E-11	63
	0.5	9.6087	13	6.64E-05	64
	0.5	14.1212	20	5.31E-07	65
	0.5	23.636	33	5.20E-11	66
	0.25	4.5609	8	9.07E-05	67
	0.25	7.1432	12	9.17E-07	68
	0.25	11.8252	21	4.15E-11	69
	0.0625	2.3527	4	8.52E-05	70
	0.0625	3.5119	6	9.81E-07	71
	0.0625	5.9618	11	1.37E-11	72

There are a lot information we can tell from this.

The first 36 cases are for single precision number:1~36

The next 36 cases are for double precision number 37~72

The first 18 cases are for a=3 number 1~18

The next 18 cases are for a=4 number 19~36

The first 9 cases are for n=100 number 1~9

The next 9 cases are for n=1000 number 10~18

The first 3 cases are for Jocobi method. number 1~3

The next 3 cases are for Gauss Seidel method number 4~6

The next 3 cases are for Symmetric Gauss Seidel method. number 7~9

The first case is for epsilon equals to 0.0001

The next case is for epsilon equals to 0.000001

The third case is for epsilon equals to 0.0000000001

Discussion and conclusion

1.Of course when a=3,4 all of the methods enjoy a good convergence rate. In single precision, when accuracy is 0.0000000001. We can't reach the required accuracy. It is easy to understand. As under single precision, we only can keep 7digits, we can't keep the update information later.So the generated X will keep stable and never update to more precise digit. It results that, the iteration number reaches to 1000. Because single precision normally can't get 10digits. If you can get the required precision ,it must be in very special cases. That case is to be talked later.

When we update the precision, we can see all of the methods can reach the required accuracy within 60 iterations.

2. .predicted number of iterations versus the number of iterations required.

$$K = \ln \frac{epsilon}{\|e_0\|_2 / \|x_{true}\|_2} / \ln \rho$$

Where epsilon is the tolerence error, $\|e_0\|_2 = \|x^{(0)} - x_{true}\|_2$

Predicted iteration number is always less than the actual iteration number.That is because when we caculate the predicted iteration, we just use the approximate equation. Atcually we omit the terms whose eigenvalues are not the largest.We just consider the term accociated with the largest eigenvalue. Because when the iteration number is big enough, the largest eigenvalue term is the leading role in caculating the error. So if we indeed the run K iteration, we can't actually reach our required accuracy, we need several extra iteration to get the accuracy. And we can't omite that we will produce errors in caculating K. This error may also be the reason for this scenio.

3. No matter we compare the spectral radium or the number of actual iteration, we can draw conclusion that the Symmetric Gauss Seidel is the fastest method, then is Gauss Seidel, the last one is Jacobi

4 spectral radium

. The spectral radium for

Jacobi

$$\rho(G_J) = \frac{2\cos\theta_1}{\alpha}$$

Gauss_seidel

$$\rho(G_{gs}) = \rho^2(G_J)$$

Symetric Guass_Seidel

$$\rho(G_{sgs}) = \rho^4(G_J)$$

$$M = (D - L)D^{-1}(D - L^T)$$

$$G_{sgs} = (D - L^T)^{-1}L(D - L)^{-1}L^T = I - M^{-1}A$$

It follows that M is symmetric positive definite and G_{sgs} is similar to a symmetric positive definite matrix

So we can use 2-norm of Gsgs to estimate the spectral radium of iterative matrix of symmetric Gauss Seidel.

But in practice, it will cost much to caculate it. And as the matrix above are sparse, so it may cost large error when we compute the inverse of matrix.

So in practice, we just use the $\rho(G_{sgs}) = \rho^4(G_J)$ to estimate the spectral radium. I got this not by accurate analysis, actually I have been stuck with how to estimate spectral radium of Symmetric Gauss Seidel for a long time. I just tried some cases and found it not too bad.

Compare a=3 and a=4, we will find that the spectral radium of a=3 is larger than a=4.

We compare number=1 and number=19 or number =4 and number=22, or more generally, we compare number=n1 and number=18+n1, they are different just in a, and their spectral radium's difference is about 0.2

We compare n=100 and n=1000, we will find that the spectral radium of n=100 is smaller than that of n=1000. We can compare number=1 and number=10 or more generally within number 1~36, you just compare number=n1 and number=n1+9. We can find the difference of their spectral radium is about 0.0001.

So we can draw conclusion that a is more important than n n can affect the convergence rate because the spectral radius varys when n varys. More specificly, when n grows up ,the convergence rate will decease.But n will not affects the rate so much as a. a plays more key role affecting the convergence rate.

4.

How did the behavior differ for the extra set of tests with single precision, $\alpha = 4.0$,

$\varrho = 10^{-10}$, $x_{true} = e$ and $x_0 = 0$ compared to those with single precision $\alpha = 4.0$,

$\varrho = 10^{-10}$, but taking normalized random x_{true} and x_0 vectors? Explain the different

behaviors.

>> test(4,1000,1,0.0000000001,1,1)		
x0 xtrue randomly chosed		
	p	0.5
	K	23.4884
	count	1000
	accuracy	5.25E-08
x=0 xtrue=e		
	p	0.5
	K	33.219
	count	34
	accuracy	5.79E-11

When we take Xtrue=e, and X0=0 it will converge to a nice solution. We know the single presion can only assure the accuracy to 7 digits. But the required precision is 10^{-10} . While randomly selected xtrue and x0 can't achieve. It happens for two reasons.

First, every step the new X we produce is nonnegative and will increase after every step. This is easy to see, as the matrix is simple in computation.So it will increace towards e. at some iteration, the X will approach near 1 very closely. And according to the princple of floating number system, if X goes really near e, the computer will take it as 1. Then the iteration will stop. Then we will get the exact answer.

Second, as for the precision. As in the iteration, the computed numbers can be accurately represented with single presion. We don't need to use double precison. So the precision will not let us fail to find the exact solution.

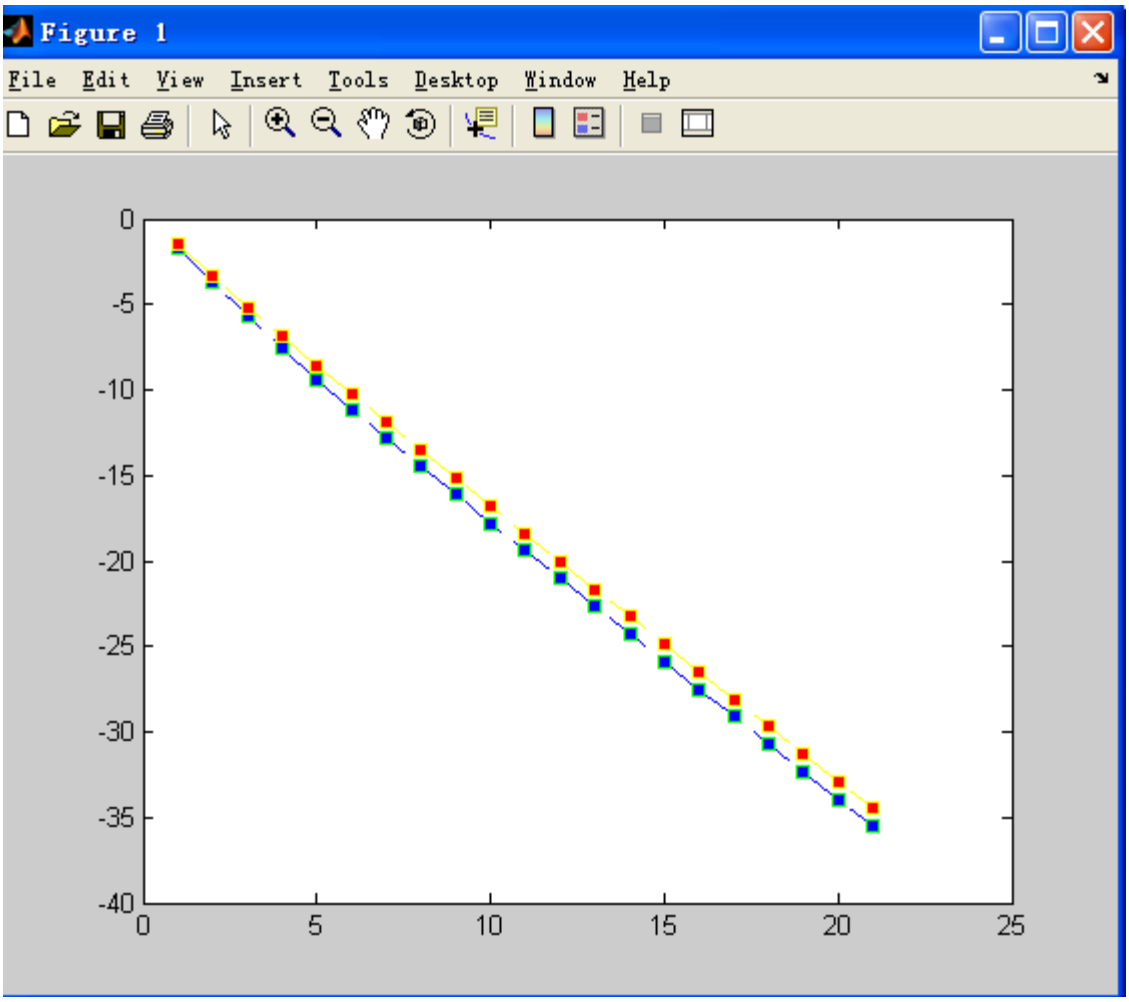
5

Trends involved in the evolution of the values $\|x_k - x_{true}\|_2/\|x_{true}\|_2$ and $\|r_k\|_2/\|b\|_2$

To be more compact, I just draw only one case. The othre scenios are similar.

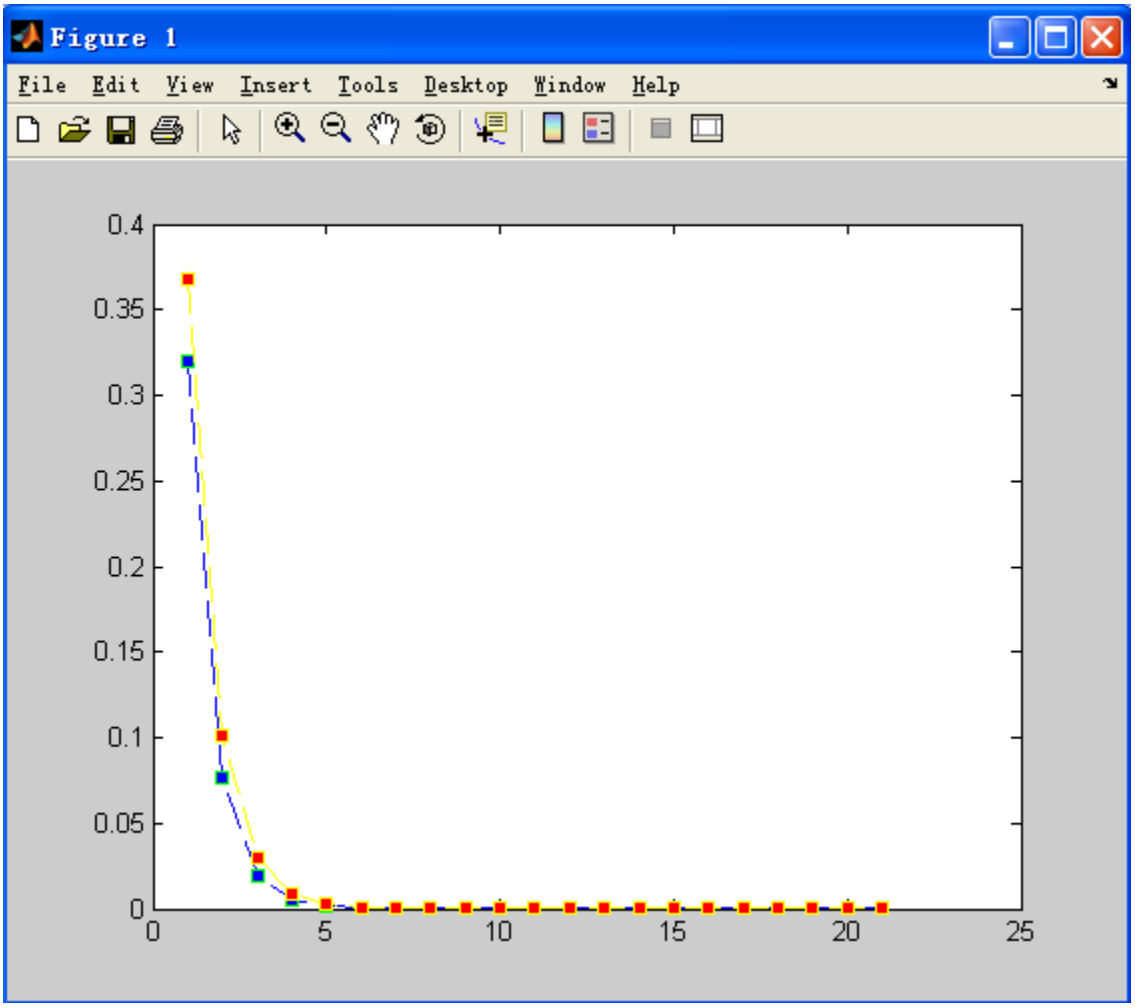
test(4,1000,2,0.0000000001,2,1)

p	0.25
K	12.0839
count	21
accuracy	4.14E-11



the logs vs. the iteration number k

As we can see. linear behavior in the log of the error and residual norms expected



the quantities vs. the iteration number k

6.

The experiments all have a known solution xtrue and therefore we can analyze the iterations' behaviors using information that is not available in practice. Suppose you did not know xtrue. You would only have at your disposal Q, rk, xk, zk, A, and b. How would you decide to terminate the iteration? Is this always effective? For example,

what happens in the cases where we have observed a lack of ability to achieve the

desired accuracy? The text discusses termination tests.

We will caculate use the trueth below to deceide to terminate the iteration namely.

$$\left\|x^{(k)} - x_{true}\right\| \le \frac{\rho}{1-\rho}\left\|x^{(k-1)} - x^{(k)}\right\| \le \frac{\rho^k}{1-\rho}\left\|x^{(0)} - x^{(1)}\right\|$$

Then it will give out the error bound between the final solution to the true solution.So if we don't know the true solution, we can use $\left\|x^{k-1} - x^{k}\right\|$ to determin when to terminate the iteration.

We can comparethe scenarios when we use different control condition.

I just run some cases. And in the code, just make two changes, the first is the argument of 'while', the other is the Zk. We don't need to let Zk to be divided anything.

Results :

	>test(4,1000,2,0.0000000001,2,1)		
	old		new
p	0.25		0.25
K	11.9081		11.6675
count	21		21
accuracy	3.84E-11		4.29E-11

	>> test(4,1000,2,0.0000000001,3,1)		
	old		new
p	0.0625		0.0625
K	5.8255		5.8993
count	11		12
accuracy	1.46E-11		1.39E-12

In our cases, it works nicely. But in practice, if we use this to dertermin the iteration, we will always run much more iteration number than it is really needed . So it has no practical value. I

refer to some books saying that in practice, we usually use the eqution below to terminate the iteration.

$$\left\| x^{(k-1)} - x^{(k)} \right\| < \varepsilon'$$

$$\varepsilon' \le \frac{1-\|G\|}{\|G\|}\varepsilon \qquad \textit{where } \varepsilon \textit{ is the error which we can accept}$$

Besides this, it will not always work for three reasons.

Firstly, because when the spectral radium gets closely to 1. The $\frac{\rho}{1-\rho}$ will blow up, and we can't tell whether the solution we computed is close to the actual solution.

Secondly , if in later iteration, $\left\| x^{k-1} - x^{k} \right\|$ stays at some step and never change any more. There are two possibility to cause such scenario.

Firstly,the iteration number which needed to achieve the accuracy required is too large as the spectral radius is too close to 1. So the rate of convergence is too slow.

Secondly,as we use the limited presicion which can't store and reflect our change in x in the later iteration. In other word, we lose the information when updating the X, as the truncation of the floating number, the X can't be updated in the later iteration. In this case, we can improve the precision to save the information but sometime it will not work.

Thirdly, the inquality just give the bound about the absolute error, we know to measure the error accurately ,we also need the relative error. In case the actual solution Xtrue is really small, then even the absolute error of computed X to the Xtrue looks small, the relative error may be big. As in our cases, we always set the Xtrue to be norm 1, so we don't have to worry about the relative error, as the magnitude of relative error equals to the absolute error.

7 .some cases for n=10000

test(3,100000,2,0.0000000001,1,1)		>> test(4,100000,2,0.0000000001,1,1)	>> test(2,100000,2,0.0000000001,1,1)
p	0.6667	0.5	1
K	3.5381	23.4307	3.30E+10
count	55	33	1000
accuracy	8.08E-11	5.14E-11	0.1882

5 Matlab M-files

Iteraion.m

```
function [X0,Zk,Ek,r,K,p]=Iteration(a,b,X0,epsilon,Xtrue,flag);
Max=1000;
s=size(b);
n=s(1);
count=1;

normx=norm(X0);
if normx<(epsilon/10)
    normx=1;
end
X00=X0;

Zk(1)=1;
Ek(1)=1;
r(1)=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%             Jacobi           %%%%%%%%%%%%%%%%%%
if flag==1
p=2*cos(pi/(n+1))/a;
while count<=Max && Ek(max(count,2)-1)> epsilon
%%  && && Zk(count)>epsilon
Zk(count)=0;
Ek(count)=0;
denom=normx;
normx=0;

    for i=1:n
        temp2=X0(i);

    if i==1
        X0(i)=(X0(i+1)+b(i))/a;
    else if i==n
           X0(i)=(temp1+b(i))/a;
```

```
        else
           X0(i)=(temp1+b(i)+X0(i+1))/a;
        end
    end

    temp1=temp2;

    Zk(count)=Zk(count)+(temp2-X0(i)).^2;
    Ek(count)=Ek(count)+(Xtrue(i)-X0(i)).^2;
    normx=normx+X0(i).^2;
    end
    normx=sqrt(normx);

    r(count)=0;
    for i=1:n
     if i==1
        r(count)=r(count)+(b(i)-a*X0(i)+X0(i+1)).^2;
    else if i==n
          r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)).^2;
        else
         r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)+X0(i+1)).^2;
        end
     end
     end

    r(count)=sqrt(r(count))/norm(b);
    Zk(count)=sqrt(Zk(count))/denom;
    Ek(count)=sqrt(Ek(count))/norm(Xtrue);

    count=count+1;
end
X00=X00-Xtrue;
K=(log(epsilon)/(norm(X00)/norm(Xtrue)))/log(p);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%     Forward      GS       %%%%%%%%%%%%%%%%%%
elseif flag==2
p=(2*cos(pi/(n+1))/a)^2;
while count<=Max && Ek(max(count,2)-1)> epsilon
%%  && && Zk(count)>epsilon
Zk(count)=0;
Ek(count)=0;
denom=normx;
normx=0;

   for i=1:n
        temp=X0(i);
    if i==1
        X0(i)=(b(i)+X0(i+1))/a;
    else if i==n
          X0(i)=(X0(i-1)+b(i))/a;
        else
           X0(i)=(X0(i-1)+b(i)+X0(i+1))/a;
        end
    end
```

```
    Zk(count)=Zk(count)+(temp-X0(i)).^2;
    Ek(count)=Ek(count)+(Xtrue(i)-X0(i)).^2;
    normx=normx+X0(i).^2;
   end
    normx=sqrt(normx);

    r(count)=0;
    for i=1:n
     if i==1
        r(count)=r(count)+(b(i)-a*X0(i)+X0(i+1)).^2;
     else if i==n
          r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)).^2;
        else
         r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)+X0(i+1)).^2;
         end
     end
    end

    r(count)=sqrt(r(count))/norm(b);
    Zk(count)=sqrt(Zk(count))/denom;
    Ek(count)=sqrt(Ek(count))/norm(Xtrue);

    count=count+1;
end

X00=X00-Xtrue;
K=(log(epsilon)/(norm(X00)/norm(Xtrue)))/log(p);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%            SGS               %%%%%%%%%%%%%%%%%%
elseif flag==3
p=(2*cos(pi/(n+1))/a)^4;
while count<=Max && Ek(max(count,2)-1)> epsilon
%%  && && Zk(count)>epsilon
Zk(count)=0;
Ek(count)=0;
denom=normx;
normx=0;

temp=ones(n,1);
    for i=1:n
        temp(i)=X0(i);
    if i==1
        X0(i)=(b(i)+X0(i+1))/a;
    else if i==n
          X0(i)=(X0(i-1)+b(i))/a;
        else
           X0(i)=(X0(i-1)+b(i)+X0(i+1))/a;
        end
    end
    end

     for i=n:-1:1
    if i==1
        X0(i)=(b(i)+X0(i+1))/a;
```

```
    else if i==n
          X0(i)=(X0(i-1)+b(i))/a;
        else
           X0(i)=(X0(i-1)+b(i)+X0(i+1))/a;
        end
    end

    Zk(count)=Zk(count)+(temp(i)-X0(i)).^2;
    Ek(count)=Ek(count)+(Xtrue(i)-X0(i)).^2;
    normx=normx+X0(i).^2;
     end

        normx=sqrt(normx);

    r(count)=0;
    for i=1:n
     if i==1
        r(count)=r(count)+(b(i)-a*X0(i)+X0(i+1)).^2;
     else if i==n
          r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)).^2;
        else
         r(count)=r(count)+(b(i)-a*X0(i)+X0(i-1)+X0(i+1)).^2;
         end
     end
    end
    r(count)=sqrt(r(count))/norm(b);
    Zk(count)=sqrt(Zk(count))/denom;
    Ek(count)=sqrt(Ek(count))/norm(Xtrue);

    count=count+1;
end
X00=X00-Xtrue;
K=(log(epsilon)/(norm(X00)/norm(Xtrue)))/log(p);
end
p
K
count=count-1
```

test.m

```
function test(a,n,precision,epsilon,flag,initial)

if precision==1
a=single(a);
if initial==1
Xtrue=single(random('unif',-1,1,n,1));    % normalize
X0=single(random('unif',-1,1,n,1));
Xtrue=Xtrue/norm(Xtrue);
X0=X0/norm(X0);
else
X0=zeros(n,1);
```

```
Xtrue=ones(n,1);
end
b=single(ones(n,1));
else
    a=double(a);
    if initial==1
Xtrue=double(random('unif',-1,1,n,1));    % normalize
X0=double(random('unif',-1,1,n,1));
Xtrue=Xtrue/norm(Xtrue);
X0=X0/norm(X0);
    else
X0=zeros(n,1);
Xtrue=ones(n,1);
    end
b=double(ones(n,1));
end


b(1)=a*Xtrue(1)-Xtrue(2);
for i=2:n-1
b(i)=-Xtrue(i-1)+a*Xtrue(i)-Xtrue(i+1);
end
b(n)=-Xtrue(n-1)+a*Xtrue(n);

[x,Zk,Ek,r,K,p]=Iteration(a,b,X0,epsilon,Xtrue,flag);
norm(x-Xtrue)/norm(Xtrue)
```

testcases.m

```
number=0;
for precision=1:2
a=3;
for aa=2:3

    n=100;
    for nn=1:2

        for flag=1:3

                epsilon=1/10000;
                for ee=1:3
                    test(a,n,precision,epsilon,flag,1);
                    number=number+1
                   epsilon=epsilon/(100.^ee);
                end

            end

        n=n*(10.^nn);
    end

    a=a+1;
```

```
end
end
```

other codes which are used to draw the graph

```
s=size(r);
for i=1:s(2)
   xxx(i)=i;
end
size(xxx) ;
size(r);
size(Ek);
plot(xxx,r,'bs--
','LineWidth',1,'MarkerEdgeColor','g','MarkerFaceColor','b','MarkerSize
',5);
hold on
plot(xxx,Ek,'ys--
','LineWidth',1,'MarkerEdgeColor','y','MarkerFaceColor','r','MarkerSize
',5);
r;
Ek;
```

```
s=size(r);
for i=1:s(2)
   xxx(i)=i;
   lr(i)=log(r(i))/log(2);
   lEk(i)=log(Ek(i))/log(2);
end
size(xxx) ;
size(lr);
size(lEk);
plot(xxx,lr,'bs--
','LineWidth',1,'MarkerEdgeColor','g','MarkerFaceColor','b','MarkerSize
',5);
hold on
plot(xxx,lEk,'ys--
','LineWidth',1,'MarkerEdgeColor','y','MarkerFaceColor','r','MarkerSize
',5);
lr;
lEk;
```