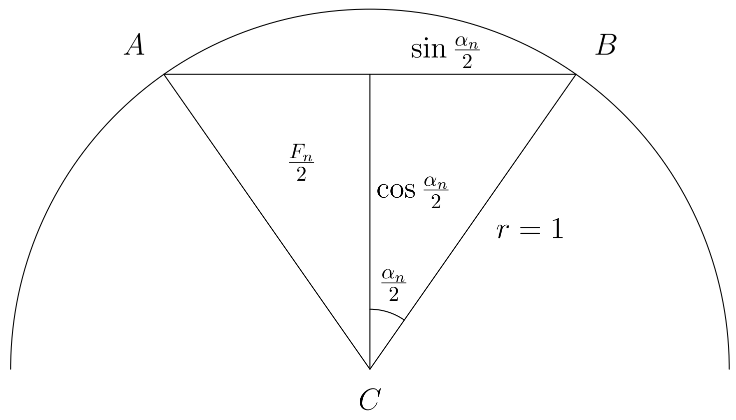
Chapter 1: Finite Precision Arithmetic

1. Some basic operations in MATLAB.

```
>> A=[1,2,3,4;5,6,7,8]
>> b=[1,2,6,8]
>> A(2,3)
>> A(1,1:3)
>> A(1,:)
>> A(:,2)
>> b(2:4)
>> A*b'
%%%%%
>> if 1==1
a=1;
else
a=2;
end
%%%%%
>> a=0;
>> for j=1:1:100
a=a+j;
end
%%%%%
>> j=1;s=0
>> while j<=100
s=s+j;
j=j+1;
end
%%%%%
>> x = -pi:0.01:pi;
plot(x,sin(x)), grid on
>> help exp
>> clc
>> clear
```

2. Use MATLAB to calculate π by unstable and stable algorithms. We consider use polygon



to approach the area of the circle. Without loss of generality, we may assume that $r = 1$. Then the area F_n of the isosceles triangle ABC with center angle $\alpha_n := \frac{2\pi}{n}$ is

$$F_n = \cos\frac{\alpha_n}{2}\sin\frac{\alpha_n}{2},$$

and the area of the associated $n-$sided polygon becomes

$$A_n = nF_n = \frac{n}{2}(2\cos\frac{\alpha_n}{2}\sin\frac{\alpha_n}{2}) = \frac{n}{2}\sin\alpha_n = \frac{n}{2}\sin\frac{2\pi}{n}$$

By expressing $\sin\frac{\alpha_n}{2}$ in terms of $\sin\alpha_n$, we have

$$\sin\frac{\alpha_n}{2} = \sqrt{\frac{1-\cos\alpha_n}{2}} = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}}$$

Since $\sin\alpha_n \to 0$, the numerator on the right hand side is

$$1-\sqrt{1-\epsilon^2}, \quad \text{with small } \epsilon = \sin\alpha_n.$$

and suffers from severe cancellation. It is possible in this case to rearrange the computation and avoid cancellation:

$$\begin{aligned}\sin\frac{\alpha_n}{2} &= \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}} = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}\frac{1+\sqrt{1-\sin^2\alpha_n}}{1+\sqrt{1-\sin^2\alpha_n}}} \\ &= \sqrt{\frac{1-(1-\sin^2\alpha)}{2(1+\sqrt{1-\sin^2\alpha})}} = \frac{\sin\alpha_n}{\sqrt{2(1+\sqrt{1-\sin^2\alpha_n})}}.\end{aligned}$$

Algorithm 1 Computation of π, Naive Version

```
s = sqrt(3)/2; A=3*s; n = 6 ;   %\text{initialization}
z =[ A - pi n A s];             % store the results
while s>1e-10                   %termination if s=sin(alpha) small
  s=sqrt((1-sqrt(1-s*s))/2);    % new sin(alpha/2) value
  n=2*n; A=n/2*s;               % A=new polygon area
  z=[z; A-pi n A s];
  end
m=length(z);
for i=1:m
  fprintf('%10d  %20.15f %20.15f %20.15f n', z(i,2), z(i,3), z(i,1),z(i,4))
end
```

Algorithm 2 Computation of π, Stable Version

```
oldA=0;s=sqrt(3)/2;  newA=3*s;  n=6;     %  initialization
z=[newA-pi n newA  s];                   %  store the results
while newA>oldA                          %  quit if area does not increase
  oldA=newA;
  s=s/sqrt(2*(1+sqrt((1+s)*(1-s))));     %  new  sine  value n=2*n; newA=n/2*s;
  z=[z; newA-pi n newA s];
end m=length(z);
for  i=1:m
  fprintf('%10d  %20.15f %20.15f\n',z(i,2),z(i,3),z(i,1))
end
```

3. The computation of the exponential function using the Taylor series:

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \dots$$

It is well known that the series converges for any x. A naive approach is therefore (in preparation of the better version later, we write the computation in the loop already in a particular form):

Algorithm 3 Compution of e^x, Naive Version

```
Computation of ex, Naive Version
function s=ExpUnstable(x,tol);
% EXPUNSTABLE computation of the exponential function
% s=ExpUnstable(x,tol); computes an approximation s of exp(x)
% up to a given tolerance tol.
% WARNING: cancellation for large negative x.
s=1; term=1; k=1;
while abs(term)>tol*abs(s)
so=s; term=term*x/k;
s=so+term; k=k+1;
end
```

We have seen that computing $f(x) = e^x$ using its Taylor series is not feasible for $x = -20$ because of catastrophic cancellation. However, the series can be used without problems for small $|x| < 1$. Try therefore the following idea:

$$e^x = (\cdots(e^{\frac{x}{2^m}})\cdots)^2$$

This means that we first compute a number m such that

$$z = \frac{x}{2^m}, \quad \text{with } |z| < 1$$

Then we use the series to compute e^z and we get the result by squaring m times.

Write a MATLAB function that computes e^x this way and compare the results with the MATLAB function `exp`.

Chapter 2: Linear Systems of Equations

1. Comparison between Cramer's rule and Gauss-elimination.

   For det(A)$\neq$ 0, the linear system $Ax = b$ has the unique solution

$$x_i = \frac{\det(A_i)}{\det(A)}$$

where A_i is the matrix obtained from A by replacing column $a_{:i}$ by b. Cramer's rule looks simple

Algorithm 4 Cramer's Rule

```
function x=Cramer(A,b);
n=length(b);
detA=DetLaplace(A);
for i=1:n
  AI=[A(:,1:i-1), b, A(:,i+1:n)];
  x(i)=DetLaplace(AI)/detA;
end
x = x(:);
```

and even elegant, but its comlexity is $O(n!)$, while with Gaussian elimination, it is $O(n^3)$.

We will now reduce in $n-1$ elimination steps the given linear system of equations

$$\begin{array}{ccccccc}
a_{11}x_1 & + & a_{12}x_2 & + \ldots + & a_{1n}x_n & = & b_1 \\
\vdots & & \vdots & & \vdots & & \vdots \\
a_{k1}x_1 & + & a_{k2}x_2 & + \ldots + & a_{kn}x_n & = & b_k \\
\vdots & & \vdots & & \vdots & & \vdots \\
a_{n1}x_1 & + & a_{n2}x_2 & + \ldots + & a_{nn}x_n & = & b_n
\end{array}$$

to an equivalent system with an upper triangular matrix. A linear system is transformed into an equivalent one by adding to one equation a multiple of another equation. An elimination step consists of adding a suitable multiple in such a way that one unknown is eliminated in the remaining equations.

To eliminate the unknown x_1 in equations #2 to #n, we perform the operations

```
for k=2:n
{new Eq. # k} = {Eq. # k} - ak1/a11{Eq.# 1}
end
```

We obtain a reduced system with an $(n-1)\times(n-1)$ matrix which contains only the unknowns $x_2,\ldots,x_n$. This remaining system is reduced again by one unknown by freezing the second equation and eliminating x_2 in equations #3 to $\#n$. We continue this way until only one equation with one unknown remains. This way we have reduced the original system to a system with an upper triangular matrix. The whole process is described by two nested loops:

```
for i=1:n-1
for k=i+1:n
{new Eq. # k} = {Eq. # k} - aki/aii{Eq. # i}
end
end
```

The coefficients of the $k-$th new equation are computed as

$$a_{kj} := a_{kj} - \frac{a_{ki}}{a_{ii}} a_{ij} \qquad \text{for} \quad j = i+1,\ldots,n.$$

and the right-hand side also changes,

$$b_k := b_k - \frac{a_{ki}}{a_{ii}} b_i.$$

Note that the $k-$th elimination step is a rank-one change of the remaining matrix. Thus, if we append the right hand side to the matrix A by `A=[A, b]`, then the elimination becomes

```
for i=1:n-1
A(i+1:n,i)=A(i+1:n,i)/A(i,i);
A(i+1:n,i+1:n+1)=A(i+1:n,i+1:n+1)-A(i+1:n,i)*A(i,i+1:n+1);
end
```

where the inner loop over k has been subsumed by Matlab's vector notation. Note that we did not compute the zeros in `A(i+1:n,i)`. Rather we used these matrix elements to store the factors necessary to eliminate the unknown x_i.

Algorithm 5 Gaussian Elimination with Partial Pivoting

```
function x=Elimination(A,b)
n=length(b); norma=norm(A,1);
A=[A,b]; % augmented matrix
for i=1:n
[maximum,kmax]=max(abs(A(i:n,i))); % look for Pivot A(kmax,i)
kmax=kmax+i-1;
if maximum < 1e-14*norma; % only small pivots
error('matrix is singular')                end
if i ~= kmax % interchange rows
h=A(kmax,:); A(kmax,:)=A(i,:); A(i,:)=h;   end
A(i+1:n,i)=A(i+1:n,i)/A(i,i); % elimination step
A(i+1:n,i+1:n+1)=A(i+1:n,i+1:n+1)-A(i+1:n,i)*A(i,i+1:n+1);
end
x=BackSubstitution(A,A(:,n+1));
```

2. For tridiagonal systems, we denote the three diagonals with the vectors $\boldsymbol{c}, \boldsymbol{d}$ and $\boldsymbol{e}$.

$$A = \begin{pmatrix} d_1 & e_1 & & & & \\ c_1 & d_2 & e_2 & & & \\ & c_2 & d_3 & e_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & c_{n-2} & d_{n-1} & e_{n-1} \\ & & & & c_{n-1} & d_n \end{pmatrix}$$

Linear systems with a tridiagonal matrix can be solved in $O(n)$ operations. The LU decomposition with *no pivoting* generates two bidiagonal matrices

$$L = \begin{pmatrix} 1 & & & & \\ l_1 & 1 & & & \\ & l_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & l_{n-1} & 1 \end{pmatrix} \quad U = \begin{pmatrix} u_1 & e_1 & & & \\ & u_2 & e_2 & & \\ & & u_3 & \ddots & \\ & & & \ddots & e_{n-1} \\ & & & & u_n \end{pmatrix}.$$

In order to compute L and U, we consider the elements c_k and d_{k+1} of the matrix A. Multiplying $L \cdot U$ and comparing elements we obtain the relations

$$\begin{aligned} l_k u_k = c_k \quad &\text{therefore} \quad l_k = c_k / u_k, \\ l_k e_k + u_{k+1} = d_{k+1} \quad &\text{therefore} \quad u_{k+1} = d_{k+1} - l_k e_k. \end{aligned}$$

The LU decomposition is thus computed by

```
u(1)=d(1);
for k=1:n-1
  l(k)=c(k)/u(k);
  u(k+1)=d(k+1)-l(k)*e(k);
end
```

Forward and back substitutions with L and U are straightforward. Note that we can overwrite the vectors $\boldsymbol{c}$ and $\boldsymbol{d}$ by $\boldsymbol{l}$ and $\boldsymbol{u}$. Furthermore, the right hand side may also be overwritten with the solution. In the French literature, this algorithm is known as Thomas' Algorithm. We obtain the function

Algorithm 6 Gaussian Elimination for Tridiagonal Systems: Thomas Algorithm

```
function [x,a,c]=Thomas(c,a,b,x);
% THOMAS Solves a tridiagonal linear system
% [x,a,c]=Thomas(c,a,b,x) solves the linear system with a
% tridiagonal matrix A=diag(c,-1)+diag(a)+diag(b,1). The right hand
% side x is overwritten with the solution. The LU-decomposition is
% computed with no pivoting resulting in L=eye+diag(c,-1),
% U=diag(a)+diag(b,1).
n=length(a);
for k=1:n-1 % LU-decomposition with no pivoting
c(k)=c(k)/a(k);
a(k+1)=a(k+1)-c(k)*b(k);
end
for k=2:n % forward substitution
x(k)=x(k)-c(k-1)*x(k-1);
end
x(n)=x(n)/a(n); % backward substitution
for k=n-1:-1:1
x(k)=(x(k)-b(k)*x(k+1))/a(k);
end
```