

MATLAB - POLYNOMIAL TOOLBOX

MATLAB 1

```
% MTIMES : Polynomial matrices multiplication. C = MTIMES(A,B)
%          where the number of the columns A must equal
%          the number of the rows B


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat11.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s \n\n','Mflops','Size','CPU_time');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=2000;st=100;ao=100;


for n=ao:st:fin,
% create two Polynomial Matrices of degree 2 with random coefficients
    A=prand(d,n);B=prand(d,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% multiply the two Polynomial Matrices
    C=mtimes(A,B);
% compute the CPU time needed for the task
    t2=cputime-t1;[ops,mflops]=flops;


% exception handler if the computational time exceeds
% one specific limit of seconds
if t2>200
fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);
for n=n+st:st:fin,
   t2=0; mflops=0;
   fprintf(f,'%12d %12d  %12d \n',mflops,n,t2);
   end

   exit

end
% end of exception loop


    fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);

end
% end of the computational loop
```

```
exit
```

MATLAB 2

```
% PLUS : Polynomial matrices addition. C = PLUS(A,B)
%        where the dimensions of the  A
%        must equal to the dimensions of  B


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat12.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s \n\n','Mflops','Size','CPU_time');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=2000;st=100;ao=100;


for n=ao:st:fin,
% create two Polynomial Matrices of degree 2 with random coefficients
    A=prand(d,n);B=prand(d,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% sum the two Polynomial Matrices
    C=plus(A,B);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;

% exception handler if the computational time
% exceeds one specific limit of seconds
if t2>200
fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);
for n=n+st:st:fin,
   t2=0; mflops=0;
   fprintf(f,'%12d %12d  %12d \n',mflops,n,t2);
   end

   exit

end
% end of exception loop

    fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);

end
% end of the computational loop

exit
```

MATLAB 3

```
% DET  : Polynomial matrix determinant.
%        The matrix should be square in dimensions.


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat13.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s \n\n','Mflops','Size','CPU_time');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=300;ao=10;st=10;


for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    A=prand(d,n);
% start the counter of computational time
    t1=cputime;flops(0);
% compute the determinant of the Polynomial Matrix
    det(A);
% compute the CPU time needed for the task
    t2=cputime-t1;[ops,mflops]=flops;

if t2>200
fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);
for n=n+st:st:fin,
   t2=0; mflops=0;
   fprintf(f,'%12d %12d  %12d \n',mflops,n,t2);
   end

   exit

end
% end of exception loop

    fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);

end
% end of the computational loop

exit
```

MATLAB 4

```
% INV  :  Polynomial matrix inverse.
$         The matrix should be square in dimensions.


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat14.out','w');
```

```
% print the metrics in the output file
fprintf(f,'%12s %12s %12s \n\n','Mflops','Size','CPU_time');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=50;ao=10;st=2;


for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    A=prand(d,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the inverse of the Polynomial Matrix
    inv(A);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;

if t2>200
fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);
for n=n+st:st:fin,
   t2=0; mflops=0;
   fprintf(f,'%12d %12d  %12d \n',mflops,n,t2);
   end

   exit

end
% end of exception loop

    fprintf(f,'%12.2f %12d %12.2f \n',mflops,n,t2);

end
% end of the computational loop


exit
```

MATLAB 5

```
% AXB : Matrix polynomial equation solver finds a particular solution
%       XO of the linear matrix polynomial equation AX=B
%       The degree of the seeking solution is specified to one



% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\\Python23\\mat15.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=200;st=5;ao=5;
```

```
for n=ao:st:fin,
% create Polynomial Matrices of degree 2 with random coefficients
    A=prand(d,n,2*n);
    B=prand(d,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the solution of the linear matrix equation
    XO=axb(A,B,1);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
   ac=norm(A*XO-B)/norm(A);

% exception handler if the computational time exceeds
% one specific limit of 200 seconds
 if t2>200
  fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac=0;
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit

end
% end of exception loop



 fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

end
% end of the computational loop

exit
```

MATLAB 6

```
%AXBYC : Matrix polynomial equation solver finds a particular solution
%        (XO,YO) of the matrix polynomial Diophantine equation AX+BY=C
%        The degree of the seeking solution is specified to one


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat16.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=200;st=5;ao=5;

for n=ao:st:fin,
% create Polynomial Matrices of degree 2 with random coefficients
    A=prand(d,n);
```

```
    B=prand(d,n);
    C=prand(d,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the solution of the matrix diophantine equation
    [XO,YO]=axbyc(A,B,C,1);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
   ac=norm(A*XO+B*YO-C)/norm(A);

% exception handler if the computational time
% exceeds one specific limit of 200 seconds
 if t2>200
   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac=0;
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit

end
% end of exception loop



   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

end
% end of the computational loop

exit
```

MATLAB 7

```
% AXBYC: Polynomial equation solver finds a particular solution
%        (xo,yo) of the scalar polynomial Diophantine equation ax+by=c
%        The degree of the seeking solution is specified to one



% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat17.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=scalar case , fin=max degree of the polynomial ,
% st=forward step of the loop, ao=initial value  of the loop
d=1;fin=2000;st=50;ao=50;


for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    a=prand(n,d);
    b=prand(n,d);
    c=prand(n,d);
% start the counter of computational time and flops
    t1=cputime;flops(0);
```

```
% compute the inverse of the Polynomial Matrix
    [xo,yo]=axbyc(a,b,c,n-1);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
   ac=norm(a*xo+b*yo-c)/norm(a);

% exception handler if the computational time
% exceeds one specific limit of 200 seconds
 if t2>200
   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac=0;
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit

end
% end of exception loop



   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

end
% end of the computational loop

exit
```

MATLAB 8

```
% AXXAB: Symmetric polynomial equation solver that solves
%        the bilateral symmetric matrix polynomial equation
%        AX + XA = B, A is a square polynomial matrix,
%        B is a symmetric polynomial matrix


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat18.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix,
% fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=40;st=2;ao=2;


for n=ao:st:fin,
% create Polynomial Matrices of degree 2 with random coefficients
    A=prand(d,n);
    B=prand(d,n);
% create a symmetric polynomial matrix
    B=B*B';
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the solution without performing the preliminary rank check
    X=axxab(A,B,'fast');
```

```
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
   ac=norm(A'*X+X'*A-B)/norm(A);

% exception handler if the computational time
% exceeds one specific limit of seconds
 if t2>200
   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac=0;
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit

end
% end of exception loop


   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

end
% end of the computational loop

exit
```

MATLAB 9

```
% AXXAB: Symmetric polynomial equation solver that solves
%        the bilateral symmetric scalar polynomial equation ax + xa = b,
%        a is a polynomial , b is a symmetric polynomial


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat19.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=scalar case , fin=max degree of the polynomial ,
% st=forward step of the loop, ao=initial value  of the loop
d=1;fin=2000;st=50;ao=50;



for n=ao:st:fin,
% create polynomials with random coefficients
    a=prand(n,d);
    b=prand(n,d);
    b=b*b' ;
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the sotution
    x=axxab(a,b);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
   ac=norm(a'*x+x'*a-b)/norm(a);
```

```
% exception handler if the computational time
% exceeds one specific limit of 200 seconds
 if t2>200
   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac='-';
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit

end
% end of exception loop



    fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

end
% end of the computational loop

exit
```

MATLAB 10

```
% SPF  : Polynomial spectral factorization. B is a continuous
%        - time para-Hermitian polynomial matrix
%        Thus [A,J]=SPF(B) solves the spectral factorization problem
%        where A is a Hurwitz -stable-
%        matrix and J is the identity matrix.


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat110.out','w');

% print the metrics in the output file
    fprintf(f,'%12s %12s %12s %12s
    \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix, fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=180;st=10;ao=10;



for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    B=prand(d,n);
% create a symmetric polynomial matrix
    B=B*B';
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the spectral factorization
    [A,J]=spf(B);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;
    [ops,mflops]=flops;
% compute the accuracy of the solution
```

```
    ac=norm(A'*J*A-B)/norm(B); % J is the identity matrix

   % exception handler if the computational time
   % exceeds one specific limit of seconds
   if t2>200
   fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
   for n=n+st:st:fin,
      t2=0; mflops=0;ac=0;
      fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
      end

      exit
 end
% end of exception loop


    fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
end
% end of the computational loop

exit
```

MATLAB 11

```
% SPF: Polynomial J spectral factorization. B is a continuous -
%      time para-Hermitian polynomial matrix
%      Thus [A,J]=SPF(B) solves the J spectral factorization problem
%      where A is a Hurwitz -stable- matrix and J is a diagonal
%      matrix of the form J = diag[1,1,..,1,-1,-1,..,-1].


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat111.out','w');


% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix, fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=180;st=10;ao=10;

for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    M1=prand(d,n);
% create the special symmetric polynomial matrix
% that will give a J spectral factorization
    M2=diag(randsrc(1,n,[1,-1]));
    B=M1'*M2*M1;
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the J spectral factorization
    [A,J]=spf(B);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
% compute the accuracy of the solution
    ac=norm(A'*J*A-B)/norm(B);

 % exception handler if the computational time
```

```
 % exceeds one specific limit of seconds
        if t2>200
        fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
        for n=n+st:st:fin,
           t2=0; mflops=0;ac=0;
           fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
           end

           exit

  end
 % end of exception loop


     fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

 end
 % end of the computational loop

 exit
```

MATLAB 12

```
% RDIV : Right polynomial matrix division [Q,R]=RDIV(N,D)
%        computes the polynomial matrix quotient Q
%        and the polynomial matrix remainder R such that N=Q*D + R


% load Polynomial Toolbox in Matlab
pinit

% open the output file
f=fopen('C:\Python23\mat112.out','w');

% print the metrics in the output file
fprintf(f,'%12s %12s %12s %12s \n\n','Mflops','Size','CPU_time','Error');


% d=degree of the Polynomial Matrix, fin=max size of the Polynomial Matrix,
% st=forward step of the loop, ao=initial value  of the loop
d=2;fin=700;st=20;ao=20;


for n=ao:st:fin,
% create one Polynomial Matrix of degree 2 with random coefficients
    N=prand(d,n);
    D=prand(d-1,n);
% start the counter of computational time and flops
    t1=cputime;flops(0);
% compute the solution of the right polynomial matrix division
    [Q,R]=rdiv(N,D);
% compute the CPU time and flops needed for the task
    t2=cputime-t1;[ops,mflops]=flops;
    ac=norm(N-Q*D-R)/norm(N);

 % exception handler if the computational time exceeds
 % one specific limit of seconds
        if t2>200
        fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);
        for n=n+st:st:fin,
           t2=0; mflops=0;ac=0;
           fprintf(f,'%12d %12d  %12d %12d \n',mflops,n,t2,ac);
           end
```

```
            exit

   end
  % end of exception loop


      fprintf(f,'%12.2f %12d %12.2f %12.3f \n',mflops,n,t2,10^10*ac);

  end
  % end of the computational loop

 exit
```


MATHEMATICA - POLYNOMIAL PACKAGE

MATHEMATICA 1

```
(* DOT  :   Multiplication of polynomial matrices *)

<<Polynomial`; (* Load the polynomial package for mathematica *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math11.out"; (* open the output file *)
(* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\n"];

(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop  *)
d = 2; ao=100; b = 2000; st = 100;
Do[
(* create two Polynomial Matrices of degree 2 with random coefficients *)
    a = PMRandom[d, n];
    b = PMRandom[d, n];
    (* multiply the two polynomial matrices *)
    (* count the CPU time needed for the task  *)
    t = Timing[Dot[a,b];] /. Second -> 1;
    (* Exception handler *)
    If[t[[1]]>150,
        Write[strm, d, "\t ", n , "\t ",t[[1]]];
        Do[
        (* t[[1]]=0; *)
        Write[strm, d, "\t ", n , "\t ",0];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
```

```
        Quit[];
        ]; (* end of exception loop  *)
        Write[strm, d, "\t ", n, "\t ", t[[1]]];
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```

MATHEMATICA 2

```
(* PLUS :  Addition of Polynomial matrices *)
<<Polynomial`;(* Load the polynomial package for mathematica *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math12.out"; (* open the output file *)
(* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\n"];

(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop *)
d = 2; ao=100; b = 2000; st = 100;
Do[
 (* create two Polynomial Matrices of degree 2 with random coefficients *)
    a = PMRandom[d, n];
    b = PMRandom[d, n];
    (* sum the two polynomial matrices *)
    (* count the CPU time needed for the task  *)
    t = Timing[Plus[a,b];] /. Second -> 1;
    (* Exception handler *)
    If[t[[1]]>150,
        Write[strm, d, "\t ", n , "\t ",t[[1]]];
        Do[
        (* t[[1]]=0; *)
        Write[strm, d, "\t ", n , "\t ",0];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
        Quit[];
        ]; (* end of exception loop  *)
        Write[strm, d, "\t ", n, "\t ", t[[1]]];
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```

MATHEMATICA 3

```
(* Det :  Determinant of a Polynomial square matrix *)
<<Polynomial`;(* Load the polynomial package for mathematica *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math13.out"; (* open the output file *)
(* set access 'write' in the output file *)
```

```
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\n"];
(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop  *)
d = 2; ao=10; fin = 300; st = 10;
Do[
 (* create one Polynomial Matrix of degree 2 with random coefficients *)
    a1 = PMRandom[d, n];
    (* compute the determinant of the polynomial matrix *)
    (* count the CPU time needed for the task  *)
    t = Timing[Det[a1];] /. Second -> 1;
    (* Exception handler *)
    If[t[[1]]>200,
        Write[strm, d, "\t ", n , "\t ",t[[1]]];
        Do[
        (* t[[1]]=0; *)
        Write[strm, d, "\t ", n , "\t ",0];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
        Quit[];
        ]; (* end of exception loop  *)
        Write[strm, d, "\t ", n, "\t ", t[[1]]];
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```

MATHEMATICA 4

```
(* Inverse of a Polynomial square matrix *)
<<Polynomial`;  (* Load the polynomial package for mathematica *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math14.out"; (* open the output file *)
(* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\n"];

(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop  *)
d = 2; ao=10; fin = 50; st =2;
Do[
(* create one Polynomial Matrix of degree 2 with random coefficients *)
    a = PMRandom[d,n];
    (* compute the inverse of the polynomial matrix *)
    (* count the CPU time needed for the task  *)
    t = Timing[Inverse[a];] /. Second -> 1;
    (* Exception handler *)
    If[t[[1]]>200,
        Write[strm, d, "\t ", n , "\t ",t[[1]]];
        Do[
        (* t[[1]]=0; *)
        Write[strm, d, "\t ", n , "\t ",0];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
        Quit[];
        ]; (* end of exception loop  *)
        Write[strm, d, "\t ", n, "\t ", t[[1]]];
```

```
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```

MATHEMATICA 5

```
(* DESolve : Solve the Polynomial matrix diophantine equation *)
<<Polynomial`; (* load the Polynomial package *)
dir = "C:\\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math15.out"; (* open the output file *)
(* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\tError\n"];

(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop  *)
d = 2; ao=5; fin = 200; st = 5;
Do[
 (* create two Polynomial Matrices of degree 2 with random coefficients *)
    a = PMRandom[d, n,2*n];
    b = PMRandom[d, n];

    (*  find the solution of the equation  *)
    (* count the CPU time needed for the task  *)
    t = Timing[AXBSolve[a,b,1];] /. Second -> 1;
    ac="-";
      (* Exception handler *)
       If[t[[1]]>200,
            Write[strm, deg, "\t ", n , "\t ",t[[1]], "\t \t " , ac];
             Do[
             (* t[[1]]=0; *)
             Write[strm, deg, "\t ", n , "\t ",0 , "\t \t " , ac];
             ,{n,n+st,fin,st}
            ];
           Close[strm];
          Quit[];
           ]; (* end of exception loop  *)
              Write[strm, deg, "\t ", n, "\t ", t[[1]],  "\t \t " , ac];
        , {n, ao, fin, st}
         ]; (* end of the computational loop *)
     Close[strm];
Quit[];
```

MATHEMATICA 6

```
<<Polynomial`; (* Solve the Diohantine matrix equation *)
dir = "C:\Python23"; SetDirectory[dir];
filename = "math16.out";
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
```

```
Write[strm, "Degree \t Size \t CPU time\tError\n"];
(* d=degree of the Polynomial matrix ,
(* fin=max size of the Polynomial matrix *)
(* st=forward step of the loop, ao=initial value  of the loop  *)
d = 2; ao=5; fin = 200; st = 5;
Do[
    (* create 3 Polynomial Matrices of degree 2 with random coefficients *)
    a = PMRandom[d,n];
    b = PMRandom[d,n];
    c = PMRandom[d,n];
        (*  find the solution of the equation  *)
    (* count the CPU time needed for the task  *)
    t = Timing[DESolve[a.x+b.y==c,1];] /. Second -> 1;
     ac="-";
        (* Exception handler *)
        If[t[[1]]>200,
            Write[strm, deg, "\t ", n , "\t ",t[[1]], "\t \t " , ac];
            Do[
            (* t[[1]]=0; *)
             Write[strm, deg, "\t ", n , "\t ",0 , "\t \t " , ac];
            ,{n,n+st,fin,st}
            ];
            Close[strm];
            Quit[];
            ]; (* end of exception loop  *)
             Write[strm, deg, "\t ", n, "\t ", t[[1]],  "\t \t " , ac];
        , {n, ao, fin, st}
        ]; (* end of the computational loop *)
    Close[strm];
    Quit[];
```

MATHEMATICA 7

```
(* AXBYCSolve  : Solve the scalar diophantine polynomial equation *)
<<Polynomial`; (* load the Polynomial package *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math17.out"; (* open the output file *)
(* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\tError\n"];

(* d=scalar case , fin=max degree of the polynomial *)
(* st=forward step of the loop, ao=initial value  of the loop *)
d=1;fin=2000;st=50;ao=50;
Do[
    (* create two polynomials with random coefficients *)
    a = PRandom[n];
    b = PRandom[n];
    c = PRandom[n];

    (* find the solution of the equation *)
    (* count the CPU time needed for the task  *)
    t = Timing[AXBYCSolve[a,b,c];] /. Second -> 1;
    ac="-";
    (* Exception handler *)
    If[t[[1]]>200,
        Write[strm, deg, "\t ", n , "\t ",t[[1]], "\t \t " , ac];
        Do[
        (* t[[1]]=0; *)
```

```
         Write[strm, deg, "\t ", n , "\t ",0 , "\t \t " , ac];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
        Quit[];
        ]; (* end of exception loop  *)
         Write[strm, deg, "\t ", n, "\t ", t[[1]],  "\t \t " , ac];
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```

MATHEMATICA 8

```
(* AXXABSolve  : Solve the scalar symmetric bilateral polynomial equation
*)
<<Polynomial`; (* load the Polynomial package *)
dir = "C:\Python23"; SetDirectory[dir]; (* set the output directory *)
filename = "math18.out";  (* open the output file *)
 (* set access 'write' in the output file *)
strm = OpenWrite[filename, FormatType -> OutputForm];
(* print the metrics in the output file *)
Write[strm, "Degree \t Size \t CPU_time\tError\n"];

(* deg=scalar case , fin=max degree of the polynomial *)
(* st=forward step of the loop, ao=initial value  of the loop *)
deg=1;fin=2000;st=50;ao=50;
Do[
    (* create two polynomials with random coefficients *)
    a = PRandom[n];
    b = PRandom[n];
    b = Chop[Star[b]*b]; (* create a symmetric polynomial  *)
    ac="-";
    (* find the solution of the equation *)
    (* count the CPU time needed for the task  *)
    t = Timing[AXXABSolve[a,b];] /. Second -> 1;
    (* Exception handler *)
    If[t[[1]]>200,
        Write[strm, deg, "\t ", n , "\t ",t[[1]], "\t \t " , ac];
        Do[
        (* t[[1]]=0; *)
        Write[strm, deg, "\t ", n , "\t ",0 , "\t \t " , ac];
        ,{n,n+st,fin,st}
        ];
        Close[strm];
        Quit[];
        ]; (* end of exception loop  *)
        Write[strm, deg, "\t ", n, "\t ", t[[1]],  "\t \t " , ac];
    , {n, ao, fin, st}
    ]; (* end of the computational loop *)
Close[strm];
Quit[];
```


SCILAB - POLYNOMIAL LIBRARY

SCILAB 1

```
//   Multiplication of two Polynomial matrices

//  open the output file with write access
f=mopen('C:\Python23\sci11.out','w');

// print the metrics in the output file
fprintf(f,"Degree  Size     CPU_time\n\n")

// create a polynomial of degree 2 with random coefficients
s=poly([1,2],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
fin=2000;st=100;ao=100;d=2;

// increase the size of the memory
stacksize(100000006)

lines(0);

for n=ao:st:fin,
// create two Polynomial Matrices of degree 2
// with random coefficients
A=s*rand(n,n); B=s*rand(n,n);
// start the counter of computational time
timer();
// multiply 2 Polynomial Matrices
C=A*B;
// count the CPU time needed for the task
t=timer();
// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \n",d,n,t)
   for n=n+st:st:fin,
   t=0;
   fprintf(f,"%i \t %i \t %i \n",d,n,t)
   end
   mclose(f);
   quit
end // end of exception loop
```

```
fprintf(f,"%i \t %i \t %f \n",d,n,t)

end // end of the computational loop
mclose(f);
quit
```

SCILAB 2

```
// Sum of two Polynomial matrices


//  open the output file with write access
f=mopen('C:\Python23\sci12.out','w');

// print the metrics in the output file
fprintf(f,"Degree  Size     CPU_time\n\n")

// create a polynomial of degree 2 with random coefficients
s=poly([1,2],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
fin=2000;st=100;ao=100;d=2;

// increase the size of the memory
stacksize(100000006);

lines(0);


for n=ao:st:fin,
// create two Polynomial Matrices of degree 2 with random coefficients
A=s*rand(n,n); B=s*rand(n,n);
// start the counter of computational time
timer();
// sum 2 Polynomial Matrices
C=A+B;
// count the CPU time needed for the task
t=timer();
// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \n",d,n,t)
   for n=n+st:st:fin,
   t=0;
   fprintf(f,"%i \t %i \t %i \n",d,n,t)
   end
   mclose(f);
   quit
end // end of exception loop
fprintf(f,"%i \t %i \t %f \n",d,n,t)

end // end of the computational loop
mclose(f);
quit
```

SCILAB 3

```
// Determinant of a square Polynomial matrix


//  open the output file with write access
f=mopen('C:\Python23\sci13.out','w');

// print the metrics in the output file
fprintf(f,"Degree  Size     CPU_time\n\n")

// create a polynomial of degree 2 with random coefficients
s=poly([1,2],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
d=2;fin=300;ao=10;st=10;

// increase the size of the memory
stacksize(100000006)

lines(0);

for n=ao:st:fin,
// create one Polynomial Matrix of degree 2 with random coefficients
A=s*rand(n,n);
// start the counter of computational time
timer();
// compute the determinant of the Polynomial Matrix using FFT
determ(A);
// count the CPU time needed for the task
t=timer();
// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \n",d,n,t)
   for n=n+st:st:fin,
   t=0;
   fprintf(f,"%i \t %i \t %i \n",d,n,t)
   end
   mclose(f);
   quit
end // end of exception loop
fprintf(f,"%i \t %i \t %f \n",d,n,t)

end // end of the computational loop
mclose(f);
quit
```

SCILAB 4

```
// Inverse of Polynomial matrix

//  open the output file with write access
f=mopen('C:\Python23\sci14.out','w');

// print the metrics in the output file
fprintf(f,"Degree  Size     CPU_time\n\n")

// create a polynomial of degree 2 with random coefficients
s=poly([1,2],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
d=2;fin=50;ao=10;st=2;

// increase the size of the memory
stacksize(100000006)

lines(0);

for n=ao:st:fin,
// create one Polynomial Matrix of degree 2 with random coefficients
A=s*rand(n,n);
// start the counter of computational time
timer();
// compute the inverse of the Polynomial Matrix
coffg(A);
// count the CPU time needed for the task
t=timer();
// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \n",d,n,t)
   for n=n+st:st:fin,
   t=0;
   fprintf(f,"%i \t %i \t %i \n",d,n,t)
   end
   mclose(f);
   quit
end // end of exception loop
fprintf(f,"%i \t %i \t %f \n",d,n,t)

end // end of the computational loop
mclose(f);
quit
```

SCILAB 5

```
// scalar diophantine equation
```

```
//  open the output file with write access
f=mopen('C:\Python23\sci15.out','w');

// print the metrics in the output file
fprintf(f,"Degree  Size     CPU_time  Error\n\n")

// create a polynomial of degree 2 with random coefficients
s=poly([1,2],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
d=1;fin=2000;ao=50;st=50;

// increase the size of the memory
stacksize(100000006)

lines(0);

for n=ao:st:fin,
// create 3 polynomial Matrices of degree 2 with random coefficients
p1=poly(rand(n,n),'s');
p2=poly(rand(n,n),'s');
b=poly(rand(n,n),'s');
// start the counter of computational time
timer();
// compute the solution of the equation
[x,err]=diophant([p1,p2],b);
// count the CPU time needed for the task
t=timer();
// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \t %f \n",d,n,t,err)
   for n=n+st:st:fin,
   t=0;err=0;
   fprintf(f,"%i \t %i \t %i \t %i \n",d,n,t,err)
   end
   mclose(f);
   quit
end // end of exception loop
fprintf(f,"%i \t %i \t %f \t %f \n",d,n,t,err)

end // end of the computational loop
mclose(f);
quit
```

SCILAB 6

```
// Right Division  of two Polynomial matrices A = Q B + R

//  open the output file with write access
f=mopen('C:\Python23\sci16.out','w');

// print the metrics in the output file
```

```
fprintf(f,"Degree  Size     CPU_time    Error\n\n")

// create a polynomial of degree 1 with random coefficients
s=poly([1],'s');

// d=degree of the Polynomial Matrix,
// fin=max size of the Polynomial Matrix,
// st=forward step of the loop, ao=initial value  of the loop
d=2;fin=700;st=20;ao=20;

// increase the size of the memory
stacksize(100000006)

lines(0);

for n=ao:st:fin,
// create 2 Polynomial Matrices of degree 2 and 1
// each with random coefficients
A=s*rand(n,n)*s;
B=s*rand(n,n);
// start the counter of computational time
timer();
// compute the solution of the right division equation
[R,Q]=pdiv(A,B);
// count the CPU time needed for the task
t=timer();
ac='-';


// exception handler if the computational time
// exceeds one specific limit of seconds
if t>200 then
   fprintf(f,"%i \t %i \t %f \t%s \n",d,n,t,ac)
   for n=n+st:st:fin,
   t=0;
   fprintf(f,"%i \t %i \t %i \t%s \n",d,n,t,ac)
   end
   mclose(f);
   quit
end // end of exception loop
fprintf(f,"%i \t %i \t %f \t%s \n",d,n,t,ac)

end // end of the computational loop
mclose(f);
quit
```