Lab6
Computational Physics I - Phys381

Guilherme Contesini , 10140201
Gerswin Magat , 00325287

April 7, 2014

Abstract

A Case Study: Patriot Missile System

1 Introduction

2 Using LU-decomposition to find the Inverse of a Matrix

2.1 -(i)

2.2 -(ii)

The code for the calculation of the inverse matrix can be found in the appendix **??**

2.3 -(iii)

1.00 0.50 0.33 0.25 0.20
0.50 0.33 0.25 0.20 0.17
0.33 0.25 0.20 0.17 0.14
0.25 0.20 0.17 0.14 0.13
0.20 0.17 0.14 0.13 0.11
(Matrix A)
25.00 -300.00 1050.00 -1400.00 630.00
-300.00 4800.00 -18900.00 26880.00 -12600.00
1050.00 -18900.00 79380.00 -117600.00 56700.00
-1400.00 26880.00 -117600.00 179200.00 -88200.00
630.00 -12600.00 56700.00 -88200.00 44100.00
(Inverse Matrix A)

3 LU Decomposition and solutions of Ax = b systems

3.1 -(i)

3.2 -(ii)

3.3 -(iii)

4 Conclusion

5 Appendix: Codes and Scripts

```
program LU_decomposition
  implicit none
  integer :: size_int , i_int , j_int
  double precision , dimension(5,5) :: L_mtx , U_mtx , a_mtx , inv_a_mtx
  double precision , dimension(5,5) :: aux_vec , sol_vec , b_vec , inv_a_vec
  double precision :: norm_1_dp , norm_inf_dp , norm_2_dp
  size_int = 5
do i_int=1,size_int
  do j_int=1,size_int
    a_mtx(i_int,j_int) = 1./(i_int+j_int-1.)
  end do
end do
  call clean_matrix(L_mtx,size_int)
  call clean_matrix(U_mtx,size_int)
  call clean_matrix(sol_vec,size_int)
  call clean_matrix(aux_vec,size_int)
  call clean_matrix(inv_a_vec,size_int)
  call clean_matrix(inv_a_mtx,size_int)
  call LUdecompCrout(a_mtx,L_mtx,U_mtx,size_int)
  call ProgresSub(L_mtx,b_vec,aux_vec,size_int)
  call RegresSub(U_mtx,aux_vec,sol_vec,size_int)
  call inverse(inv_a_mtx,a_mtx,size_int)
  write(*,*) ""
  write(*,*) "Matrix A"
  write(*,*) ""
  call print_matrix(a_mtx,size_int)
  write(*,*) ""
  write(*,*) ""
  write(*,*) "inverse"
  write(*,*) ""
  call print_matrix(inv_a_mtx,size_int)
  write(*,*) ""
  norm_1_dp = MAXVAL(SUM(ABS(a_mtx),DIM=1))
  norm_inf_dp = MAXVAL(SUM(ABS(a_mtx),DIM=2))
  norm_2_dp = SQRT(SUM(a_mtx**2.0))
  write(*,*) 'norm_1 = ', norm_1_dp
  write(*,*) 'norm_inf = ', norm_inf_dp
  write(*,*) 'norm_2 = ', norm_2_dp
  write(*,*) ""
  norm_inv_1_dp = MAXVAL(SUM(ABS(inv_a_mtx),DIM=1))
  norm_inv_inf_dp = MAXVAL(SUM(ABS(inv_a_mtx),DIM=2))
  norm_inv_2_dp = SQRT(SUM(inv_a_mtx**2.0))
  write(*,*) 'norm_1 = ', norm_inv_1_dp
  write(*,*) 'norm_inf = ', norm_inv_inf_dp
  write(*,*) 'norm_2 = ', norm_inv_2_dp
  write(*,*) ""
  k_a_1_dp = norm_1_dp * norm_inv_1_dp
  k_inf_dp = norm_inf_dp * norm_inv_inf_dp
  write(*,*) 'k(a)1 = ' , k_a_1_dp
  write(*,*) 'k(a)inf = ' , k_inf_dp
```

```
subroutine clean_matrix(a_mtx,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(out) , dimension(size_int,size_int) :: a_mtx
  integer :: i_int , j_int
  do i_int = 1, size_int
    do j_int = 1,size_int
      a_mtx(i_int,j_int) = 0
    end do
  end do
end subroutine
subroutine LUdecompCrout(a_mtx,L_mtx,U_mtx,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(in) , dimension(size_int,size_int) :: a_mtx
  double precision , intent(out) , dimension(size_int,size_int) :: L_mtx , U_mtx
  integer :: i_int , j_int , k_int
  double precision :: sum_float
  !step 1 & 2
  do i_int = 1,size_int,1
    L_mtx(i_int,1) = a_mtx(i_int,1)
    U_mtx(i_int,i_int) = 1.
  end do
  !step 3
  do j_int = 2,size_int,1
    U_mtx(1,j_int) = a_mtx(1,j_int)/L_mtx(1,1)
  end do
  !step 4
  do i_int=2,size_int
    do j_int=2,i_int
      sum_float = 0.
      do k_int=1,j_int-1,1
        sum_float = sum_float + L_mtx(i_int,k_int)*U_mtx(k_int,j_int)
      end do
      L_mtx(i_int,j_int) = a_mtx(i_int,j_int) - sum_float
    end do
    do j_int=i_int+1,size_int,1
      sum_float = 0.
      do k_int = 1,i_int-1,1
        sum_float = sum_float + L_mtx(i_int,k_int)*U_mtx(k_int,j_int)
      end do
      U_mtx(i_int,j_int) = (a_mtx(i_int,j_int)-sum_float)/L_mtx(i_int,i_int)
    end do
  end do
end subroutine
subroutine ProgresSub(a_mtx,b_vec,aux_vec,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(in) , dimension(size_int,size_
  double precision , intent(in) , dimension(size_int,1) ::
  double precision , intent(out) , dimension(size_int,1) :
  integer :: i_int , j_int
  double precision :: sum_float
  aux_vec(1,1) = b_vec(1,1)/a_mtx(1,1)
  do i_int=2,size_int
    sum_float = 0.
    do j_int = 1,i_int-1,1
      sum_float = sum_float + (a_mtx(i_int,j_int)*aux_vec(
    end do
    aux_vec(i_int,1) = (b_vec(i_int,1)-sum_float)/a_mtx(i_
  end do
end subroutine
subroutine RegresSub(a_mtx,b_vec,sol_vec,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(in) , dimension(size_int,size_
  double precision , intent(in) , dimension(size_int,1) ::
  double precision , intent(out) , dimension(size_int,1) :
  integer :: i_int , j_int
  double precision :: sum_float
  sol_vec(size_int,1) = b_vec(size_int,1)/a_mtx(size_int,s
  do i_int = size_int-1,1,-1
    sum_float = 0.
    do j_int =size_int,i_int+1,-1
      sum_float = sum_float + a_mtx(i_int,j_int )*sol_vec(
    end do
    sol_vec(i_int,1) = ( b_vec(i_int,1) - sum_float )/a_mt
  end do
end subroutine
subroutine inverse(inv_a_mtx,a_mtx,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision, intent(in) , dimension(size_int,size_i
  double precision , intent(out) , dimension(size_int, siz
  double precision , dimension(size_int,size_int) :: L_mtx
  double precision , dimension(size_int,size_int) :: aux_v
  integer :: i_int , j_int

  call clean_matrix(L_mtx,size_int)
  call clean_matrix(U_mtx,size_int)
```

```
  call clean_matrix(sol_vec,size_int)
  call clean_matrix(aux_vec,size_int)
  call clean_matrix(i_vec,size_int)
  call identity(i_mtx,size_int)

  do i_int = 1 , size_int
    do j_int = 1, size_int
      i_vec(j_int,1) = i_mtx(j_int,i_int)
    end do
    call LUdecompCrout(a_mtx,L_mtx,U_mtx,size_int)
    call ProgresSub(L_mtx,i_vec,aux_vec,size_int)
    call RegresSub(U_mtx,aux_vec,sol_vec,size_int)
    do j_int = 1, size_int
      inv_a_mtx(j_int,i_int) = sol_vec(j_int,1)
    end do
  end do
end subroutine

subroutine identity(i_mtx,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(out) , dimension(size_int,size_int) :: i_mtx
  integer :: i_int , j_int
  do i_int = 1,size_int
    do j_int = 1, size_int
      i_mtx(i_int,j_int) = 0
    end do
    i_mtx(i_int,i_int) = 1
  end do
end subroutine

subroutine print_matrix(s_mtx,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(in) , dimension(size_int,size_int) :: s_mtx
  integer :: i_int , j_int
  do i_int=1,size_int
    do j_int=1,size_int
      if(j_int==size_int) then
        write(*,"(f10.2)",advance="yes") s_mtx(i_int,j_int)
      else
        write(*,"(f10.2)",advance="no") s_mtx(i_int,j_int)
      end if
    end do
  end do
end subroutine
```

```
subroutine print_vector(s_vec,size_int)
  implicit none
  integer , intent(in) :: size_int
  double precision , intent(in) , dimension(size_int,size_
  integer :: i_int
  do i_int=1,size_int
      write(*,*)s_vec(i_int,1)
  end do
end subroutine

end program
```