Homework 2 & 3

Due date: March 30, 2021, 11:55PM.

Bibliography: Trefethen & Bau, Lectures 12-19. Problems 1-4 = 2 pt each, Problem 5 = 8 points.

1. Exercise 12.2
2. Exercise 12.3
3. Exercise 15.1
4. Exercise 16.2
5. Carry out change of coordinates by solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{I}\boldsymbol{b} = \boldsymbol{b}$, $\boldsymbol{A} \in \mathbb{R}^{m\times m}$, through the steps:
   i. LU-factorization with full pivoting
   ii. Forward substitution
   iii. Backward substitution.

   a) Implement the above as Fortran computational kernels called from within Python using the f2py utility.

   b) Generate within Python random full matrices $\boldsymbol{A}$ with $m = 50$ and condition numbers $\mu(\boldsymbol{A}) \in \{1, 10^3, 10^6, 10^9, 10^{12}, 10^{15}\}$. Generate $n = 20$ random vectors $\boldsymbol{x}_k$, $k = 1, ..., n$. Compute $\boldsymbol{b}_k = \boldsymbol{A}\boldsymbol{x}_k$. Now solve the systems $\boldsymbol{A}\,\tilde{\boldsymbol{x}}_k = \boldsymbol{b}_k$ using your Fortran kernel to obtain errors $\varepsilon_k = \| \boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k \| / \| \boldsymbol{x}_k \|$. Present a scatter plot of $\{(\lg \mu(\boldsymbol{A}), \lg \varepsilon_k), k = 1, ..., n\}$ for matrices with the above-specified condition numbers. Label your plot as for publication. Analyze your results. Is Gaussian elimination with full pivoting accurate?

   c) For each of the $\tilde{\boldsymbol{x}}_k$ and matrices considered in (b) compute the the matrix $\delta\boldsymbol{A}$ that exactly satisfies $(\boldsymbol{A} + \delta\boldsymbol{A})\tilde{\boldsymbol{x}}_k = \boldsymbol{b}_k$. Note that since $\boldsymbol{b}_k = \boldsymbol{A}\boldsymbol{x}_k$, the equality $\delta\boldsymbol{A}\,\tilde{\boldsymbol{x}}_k = \boldsymbol{A}(\boldsymbol{x}_k - \tilde{\boldsymbol{x}}_k) = \boldsymbol{A}\delta\boldsymbol{x}_k = \boldsymbol{c}_k$ is obtained, with $\boldsymbol{A}, \tilde{\boldsymbol{x}}_k, \delta\boldsymbol{x}_k$ known. To mimic exact arithmetic compute relevant quantitites in quad precision in Fortran. The equality $\delta\boldsymbol{A}\,\tilde{\boldsymbol{x}}_k = \boldsymbol{c}_k$ can be satisfied by many choices of $\delta\boldsymbol{A}$. One approach is to choose orthogonal matrices $\boldsymbol{U}, \boldsymbol{V}$, represent $\delta\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ and determine appropriate singular values of $\delta\boldsymbol{A}$. Present a scatter plot of your results $\{(\lg \mu(\boldsymbol{A}), \|\delta\boldsymbol{A}_k\|)\}$. Is Gaussian elimination with full pivoting backward stable?

1 Theory

2 Computational experiments

Recall that to transform coordinates from basis $\boldsymbol{I}$ to basis $\boldsymbol{A}$, solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ through steps:

1. LU factorization $\boldsymbol{L}\boldsymbol{U} = \boldsymbol{A}$, $(\boldsymbol{L}\boldsymbol{U})\boldsymbol{x} = \boldsymbol{b} \Leftrightarrow \boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$, $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$
2. Forward substitution $\boldsymbol{y} = \boldsymbol{L} \backslash \boldsymbol{b}$
3. Backward substitution $\boldsymbol{x} = \boldsymbol{U} \backslash \boldsymbol{y}$

2.1 f2py computational kernels

2.1.1 Forward substitution

Consider the forward substitution algorithm to find solution of $\boldsymbol{L}\boldsymbol{y} = \boldsymbol{b}$, with $\boldsymbol{L}$ lower triangular.

Algorithm fwd

Input: $m \in \mathbb{N}_+, \boldsymbol{b} \in \mathbb{R}^m, \boldsymbol{L} \in \mathbb{R}^{m\times m}$
Output: $\boldsymbol{y} \in \mathbb{R}^m$
$\boldsymbol{y} = \boldsymbol{b}$
for $i = 1$ to m
  $y_i = y_i / L_{ii}$
  for $j = i + 1$ to m
    $y_j = y_j - L_{ji}\, y_i$

This can be transcribed into Fortran 90 code.

```
subroutine fwd(m,b,L,y)
  integer, parameter :: dp = 8
  integer, intent(in) :: m
  real (dp), intent(in),  dimension(m)   :: b
  real (dp), intent(in),  dimension(m,m) :: L
  real (dp), intent(out), dimension(m)   :: y
  ! Forward substitution
  !   0. Initialize
  y = b
  do i=1,m
    y(i) = y(i)/L(i,i)
    do j=i+1,m
      y(j) = y(j) - L(j,i)*y(i)
    enddo
  enddo
end subroutine fwd
```

The above can be compiled into a Python module using `f2py`, through the following shell commands

```
Shell] cd /home/mitran/courses/MATH662/examples/f2py/LUalgorithms; ls
  lualgs.cpython-38-x86_64-linux-gnu.so  lualgs.f90  lualgs.so  Makefile
Shell] export CFLAGS=-w; make
  make: 'lualgs.so' is up to date.
Shell]
```

Test the algorithm within Python

```
Python] import os; os.chdir('/home/mitran/courses/MATH662/examples/f2py/LUalgorithms');
        import sys; sys.path.insert(0,os.getcwd()); print(sys.version_info)
  sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
Python] from lualgs import *
Python] fwd.__doc__
  y = fwd(b,l,[m])

  Wrapper for ``fwd``.

  Parameters
  ----------
  b : input rank-1 array('d') with bounds (m)
  l : input rank-2 array('d') with bounds (m,m)

  Other Parameters
  ----------------
  m : input int, optional
      Default: len(b)

  Returns
```

```
   -------
   y : rank-1 array('d') with bounds (m)
Python] from pylab import *; from scipy.linalg import lu;
Python] m=5; A=rand(m,m); P, L, U = lu(A);
Python] y=rand(m,1); from numpy import matmul; b=matmul(L,y);
Python] y1=fwd(b,L);
Python] y1
    [0.02593487 0.26171791 0.91408395 0.0401856  0.27872207]
Python] y
    [[0.02593487]
    [0.26171791]
    [0.91408395]
    [0.0401856 ]
    [0.27872207]]
Python]
```

2.1.2 Backward substitution

Consider the backward substitution algorithm to find solution of $\boldsymbol{U}\boldsymbol{x} = \boldsymbol{y}$, with $\boldsymbol{U}$ upper triangular.

Algorithm bck

Input: $m \in \mathbb{N}_+, \boldsymbol{y} \in \mathbb{R}^m, \boldsymbol{U} \in \mathbb{R}^{m \times m}$
Output: $\boldsymbol{x} \in \mathbb{R}^m$
$\boldsymbol{x} = \boldsymbol{y}$
for $i = m$ downto 1
  $x_i = x_i / U_{ii}$
  for $j = 1$ to $i - 1$
    $x_j = x_j - U_{ji}\, x_i$

This can be transcribed into Fortran 90 code.

```
subroutine bck(m,y,U,x)
  integer, parameter :: dp = 8
  integer, intent(in) :: m
  real (dp), intent(in),  dimension(m)   :: y
  real (dp), intent(in),  dimension(m,m) :: U
  real (dp), intent(out), dimension(m)   :: x
  ! Forward substitution
  !   0. Initialize
  x = y
  do i=m,1,-1
    x(i) = x(i)/U(i,i)
    do j=1,i-1
      x(j) = x(j) - U(j,i)*x(i)
    enddo
  enddo
end subroutine bck
```

```
Shell] cd /home/mitran/courses/MATH662/examples/f2py/LUalgorithms; ls
   lualgs.cpython-38-x86_64-linux-gnu.so  lualgs.f90  lualgs.so  Makefile
Shell] export CFLAGS=-w; make
   f2py3 -c --quiet -m lualgs lualgs.f90
Shell]
```

Test the algorithm within Python

```
Python] import os; os.chdir('/home/mitran/courses/MATH662/examples/f2py/LUalgorithms');
        import sys; sys.path.insert(0,os.getcwd()); print(sys.version_info)
   sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
Python] from lualgs import *
Python] bck.__doc__
   x = bck(y,u,[m])

   Wrapper for ``bck``.

   Parameters
   ----------
   y : input rank-1 array('d') with bounds (m)
   u : input rank-2 array('d') with bounds (m,m)

   Other Parameters
   ----------------
   m : input int, optional
       Default: len(y)

   Returns
   -------
   x : rank-1 array('d') with bounds (m)
Python] from pylab import *; from scipy.linalg import lu;
Python] m=5; A=rand(m,m); P, L, U = lu(A);
Python] x=rand(m,1); from numpy import matmul; y=matmul(U,x);
Python] x1=bck(y,U);
Python] x1
    [0.86287096 0.12996283 0.81016664 0.71610428 0.1715608 ]
Python] x
    [[0.86287096]
    [0.12996283]
    [0.81016664]
    [0.71610428]
    [0.1715608 ]]
Python]
```