Eigenproblem & Random matrix theory & LU reduction

Umberto Maria Tomasini
Quantum Information Course

13/11/2018

Abstract

In this report we initialize hermitian and diagonal real matrices with random values between 0 and 1, taken uniformely. Then we diagonalize them and we store the eigenvalues in crescent order. We compute the spacings between the eigenvalues, normalizing first globally and then locally. The number of spacings taken into account in the local normalization is controlled via a "Locality parameter" called kk in the report. This is done for different matrix dimension and different kk, which depend on the dimension. We plot the distribution of those normalized spacings, and we fit those distributions with the function $P(s) = as^{\alpha}exp(-be^{\beta})$. For each dimension, we plot those parameters against kk and we compute the average.
In a different section, we perform the LU reduction on hermitian matrices with random values between 0 and 1, taken uniformely. This is done for different matrix dimension. Then we plot the computational time needed for that task, and we fit the graph (log(cpu_time) vs log(dimension)).

Theory - Eigenproblem & Random matrix theory

Given a random matrix, we take into account its eigenvalues, sorted in ascending order: $\{\lambda_1 ... \lambda_N\}$. Then we can obtain the spacings between them, which we can normalize: $s_i = \Delta\lambda_i / \overline{\Delta\lambda_i}$. The normalization can be parametrized via a Locality parameter called kk, which is the number of the closest spacings taken into account in the local normalization of $\Delta\lambda_i$: $\overline{\Delta\lambda_i} = \sum_{j=1,\, closest}^{kk} \Delta\lambda_j / kk$.
Those s_i are distributed like $P(s) = as^{\alpha} exp(-be^{\beta})$. The parameters for Hermitian matrices, according to the literature[1], are:

	Hermitian
a	$32/\pi^2$
α	2
b	$4/\pi$
β	2

Code Development - Eigenproblem & Random matrix theory

Listing 1: Fortran code: Diagonalization, sorting of eigenvalues and computation of normalized spacings between eigenvalues, for Hermitian and Diagonal real matrices.

```
C
C  =========== DOCUMENTATION ===========
C  Definition:
C ===========
C
C*************************************************************
C         MODULE MATRICES
C*************************************************************
C

C        TYPE dcm

C        Arguments:
C
C        INTEGER nr              number of rows
C        INTEGER nr              number of columns
C        double complex  elem    elements of matrix
C        double precision eval   eigenvalues hermitian matrix(
   real)
C        double complex m_trace  trace
C        double complex m_det    determinat
```

[1] Wikipedia-Random matrix-Distribution of level spacings.

```
C          Purpose
C          =======
C
C       \details \b Purpose:
C       \verbatim
C       Can be used to contain double complex matrices and
   their features.
C       The eigenvalues are double precision to treat the
   case of hermitian matrix,
C       but it can be changed.
C       \endverbatim

C
C      SUBROUTINE init_hermmat (aa, aacheck, numrow, numcol,
   debug)
C
C       .. Scalar Arguments ..
C       INTEGER           numrow
C       INTEGER           numcol
C       LOGICAL           debug
C
C      .. Array Arguments ..
C       type(dcm)           aa
C       type(dcm)           aacheck
C
C
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       Initialization of a random hermitian matrix, stored
   in a type(dcm).
C       The random numbers are between 0 and 1, flat
   distribution.
C       Determinat, trace and eigenvalues set to default 0.
C       An identical matrix is initialized (aacheck), in
   order to future
C       debugging.
C       The dimension of matrix should be equal to numrow and
    numcol,
C       if different, a warning message is printed. This
   debug is done
```

```
C        if debug==.true.
C
C        \endverbatim
C
C        Arguments:
C         ==========
C

C        \param[in] aa
C        \verbatim
C           aa is type(dcm)
C                To be initialized
C        \endverbatim
C        \param[in] aacheck
C        \verbatim
C           aacheck is type(dcm)
C                A copy of the initialized matrix
C        \endverbatim
C
C        \param[in] numrow
C        \verbatim
C           numrow is INTEGER
C        \endverbatim
C
C        \param[in] numcol
C        \verbatim
C           numcol is INTEGER
C        \endverbatim
C
C
C        \param[in] debug
C        \verbatim
C           numrow is LOGICAL
C        \endverbatim
C
C=======================================================
C
C
C        SUBROUTINE print_matrices (AA, namefile)
C
C        .. Scalar Arguments ..
C        character(:)            namefile
C        ..
C       .. Array Arguments ..
C        type(dcm)          AA
```

```
C       ..
C
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This subroutine prints a type(dcm) variable on a file
   , called
C       "namefile", at unit 40.
C       Number of rows, columns, matrix elements, determinant
   , trace and eigenvalues
C       are printed. The matrix elements are printed in
   proper order.
C       \endverbatim
C
C       Arguments:
C       ==========
C

C       \param[in] AA
C       \verbatim
C          AA is type(dcm)
C               The variable that will be printed
C       \endverbatim
C
C       \param[in] namefile
C       \verbatim
C               namefile is character(:)
C       \endverbatim
C
C ================================================
C
C
C       SUBROUTINE print_vector (vec, nn, namefile)
C
C       .. Scalar Arguments ..
C       character(:)            namefile
C       integer                 nn
C       ..
C      .. Array Arguments ..
C       double precision         vec(nn)
C       ..
C
```

```
C       Purpose
C       =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This subroutine prints a double precision vector on a
    file, called
C       "namefile", at unit 80.
C       \endverbatim
C
C       Arguments:
C       ==========
C

C       \param[in] vec
C       \verbatim
C          vec is double precision, dimension(:)
C               The vector that will be printed
C       \endverbatim
C       \param[in] nn
C       \verbatim
C          nn is INTEGER
C       \endverbatim
C       \param[in] namefile
C       \verbatim
C               namefile is character(:)
C       \endverbatim
C  =====================================================
C
C*******************************************************
C        MODULE debugging
C*******************************************************
C
C       SUBROUTINE CHECKDIM(nn,nncheck,debug)
C
C       .. Scalar Arguments ..
C       INTEGER*2             nn
C        INTEGER*2             nncheck
C        LOGICAL               debug
C       ..
C       .. Array Arguments ..
C       INTEGER*2          m(nn,nn)
C       INTEGER*2          mcheck(nn,nn)
C       ..
```

```
C       Purpose
C        =======
C
C       \details \b Purpose:
C       \verbatim
C
C       CHECKDIM checks if the dimension of the matrix are
   correct. Firstly checks
C       if the the dimension INTEGER*2 is above 10000. If it
   is so, it prints a WARNING
C       message because it takes too much time. Secondly, it
   checks if nn is inferior
C       to 1, printing a WARNING message. Lastly, it checks
   if nn is actually nncheck
C       as it should be. If it is not true, it prints a
   WARNING message, stating which
C       should be the dimension and the actual one. If
   everything goes well, it prints
C       "okay:␣right␣dimensions␣matrices",
C       and the actual dimension.

C       \endverbatim
C
C       Arguments:
C       ==========
C


C       \param[in] nn
C       \verbatim
C          nn is INTEGER*2
C          The dimension of the squared arrays
C       \endverbatim
C       \param[in] qq
C       \verbatim
C          qq is INTEGER*2
C               the number of the cycle, it should be nn=qq
   *100
C       \endverbatim
C
C       \param[inout] debug
C       \verbatim
C          debug is LOGICAL
C       \endverbatim
C
C=======================================================
```

```
C
C       SUBROUTINE CHECKMAT(nn,m,mcheck,debug)
C
C       .. Scalar Arguments ..
C       INTEGER*2             nn
C        LOGICAL               debug
C       ..
C       .. Array Arguments ..
C       INTEGER*2          m(nn,nn)
C       INTEGER*2          mcheck(nn,nn)
C       ..
C
C
C         Purpose
C         =======
C
C       \details \b Purpose:
C       \verbatim
C
C        CHECKMAT checks if the two imput matrices m and
   mcheck are equal. This is done
C        via a do cycle which checks the equality between the
    two matrices element
C        by element. For each entry which is not equal, the
   internal variable INTEGER*4 accum
C        increseas by 1 (starting from 0). If accum in the
   end is bigger than 0,
C        a WARNING message states how much entries between
   the two matrices are
C        different. Otherwise, it is printed "Same matrices".
C       \endverbatim
C
C        Arguments:
C         ==========
C

C        \param[in] nn
C        \verbatim
C          nn is INTEGER*2
C                 The dimension of the squared arrays
C        \endverbatim
C
C        \param[in]
C        \verbatim
C                 m is INTEGER*2 ARRAY (nn,nn)
```

```
C         \endverbatim
C
C         \param[in]
C         \verbatim
C                  mcheck is INTEGER*2 ARRAY (nn,nn)
C         \endverbatim
C
C         \param[inout] debug
C         \verbatim
C                  debug is LOGICAL
C         \endverbatim
C
C
   ================================================================

C
C*****************************************************
C         MODULE normalized_spacings
C*****************************************************
C=====================================================
C
C        SUBROUTINE normspac(aa,kk,normsp,debug)
C
C        .. Scalar Arguments ..
C        INTEGER          kk
C         LOGICAL         debug
C        ..
C        .. Array Arguments ..
C        type(dcm)             aa
C        doubleprecision       normsp(aa%nr)
C        ..
C
C
C          Purpose
C          =======
C
C        \details \b Purpose:
C        \verbatim
C
C         normspac computes normalized spacings between
   eigenvalues
C         disposed in crescent order of a double complex
   matrix, which
C         is inside type aa. The impute integer kk is a "
   locality_parameter",
```

```
C        i.e the number of spacings plus one considered in
   the computation
C        of the norm for each spacing.
C        The first thing computed are the spacings, then the
   norms.
C        A if cycle secures to have (kk-1) in [1,nn-1] and
   correct if it is not,
C        with a warning message.
C        In a cycle over the spacings, the boundaries over
   which the norm is computed
C        are assigned to each spacing, in a such a way to
   include kk-1 spacings.
C        In case of even kk the interval is symmetric, in
   case of odd kk it is not.
C        In case those boundaries exceed the global
   boundaries, they are translated
C        to consider the interval between the global boundary
    and another point, such
C        that to have kk-1 spacings in between.
C        Lastly, norms and normalized spacing are computed.
C        If debug is .true., debugging is done (printing on
   file and on terminal).
C
C       \endverbatim
C
C        Arguments:
C         ==========
C


C        \param[in] aa
C        \verbatim
C          aa is type(dcm)
C                 The type contain the eigenvalues in eval
C        \endverbatim
C
C        \param[in]
C        \verbatim kk
C                 kk is INTEGERk
C        \endverbatim
C
C        \param[out]
C        \verbatim
C                 normsp is doubleprecision(aa%nr)
C        \endverbatim
C
```

```
C         \param[inout] debug
C         \verbatim
C                  debug is LOGICAL
C         \endverbatim
C
C=============================================================
C*************************************************************
C         PROGRAM eigenproblem
C*************************************************************
C=============================================================
C
C          Purpose
C          =======
C
C       \details \b Purpose:
C       \verbatim
C
C       If hermdiag==.true, this program, given the matrix
   dimension and the locality parameter kk
C       (see subroutine normsp) from file "MatDimension.txt",
    initialize a type(dcm),
C       with inside a random hermitian matrix (in range
   [0,1], flat distribution).
C       Via the lapack routine zheev, the (double precision)
   eigenvalues are stored inside
C       the type, in crescent order. Moreover, normalized
   spacings are computed via the
C       subroutine normsp. Those normalized spacings are
   printed on file
C       "NormSpacingsHerm.txt".
C       If hermdiag==.true, the same is done for a diagonal
   double precision matrix,
C       stored in a type(dcm) variable. zheev is useless now.
    In order to sort in ascending order,
C       the subroutine hpsort_eps_epw is used. The normalized
    spacings are printed on file
C       "NormSpacingsDiag.txt".
C       If debug==.true., debugging is done: checking matrix
   dimension, if matrix is
C       still the same, printing type(dcm) variables on file
   (including eigenvalues).
C       Morever, via the output iinfo, we check if zheev
   worked.
C
C       \endverbatim
```

```
C          Authors:
C          ========
C
C         \author Univ. of Padua
C
C         \date 13 November 2018

C
   ================================================================

         module matrices
implicit none
type dcm !defining the type: doublecomplex matrix
integer :: nr !number of rows
integer :: nc !number of columns
!the actual matrix
double complex, dimension (: , :), allocatable :: elem
!eigenvalues
double precision, dimension (:), allocatable :: eval
!trace
double complex :: m_trace
!determinant
double complex :: m_det
end type dcm

contains

subroutine init_hermmat (aa, aacheck, numrow, numcol,debug)
!this subroutine initializes a type dcm,
!given as imput. The matrix becomes Herminian.
!Also the numbers of rows and columns
!are given as imput. The subroutine creates random matrices
!with values in range [0,1].
!It gives a default value 0 to the determinant, the trace
!and to eigenvalues.
real*8 yy, xx
integer ii,jj
type(dcm) :: aa
type(dcm) :: aacheck
integer numrow, numcol
integer :: my_stat
character (256) :: my_msg
logical debug
```

```
aa%nr= numrow
aa%nc= numcol
aacheck%nr= numrow
aacheck%nc= numcol

!ERROR HANDLING ALLOCATION VECTORS
allocate (aa%eval(aa%nr),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if
allocate (aacheck%eval(aa%nr),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!ERROR HANDLING ALLOCATION MATRICES
allocate(aa%elem(aa%nr, aa%nc), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(aacheck%elem(aa%nr, aa%nc),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if


!WARNING IF NOT SQUARED
if(debug .eqv. .true.) then
if(numrow==numcol) then !checking squareness
print*, " "
print*, "INITIALIZATION"
print*, "Okay, squared"
else
```

```
print*, "WARNING: NOT SQUARED"
print*, "num rows= ", numrow
print*, "num columns= ", numcol
end if
end if

!INITIALIZATION
do ii= 1, aa%nr
do jj= ii, aa%nc
call random_number(xx)
call random_number(yy)
aa%elem(ii,jj)=cmplx(xx,yy)!complex number
aa%elem(jj,ii)=conjg(aa%elem(ii,jj))!hermitianity
!in order to be hermitian, complex part on diagonale
!must be null
if(ii==jj) then
aa%elem(ii,jj)=cmplx(xx,0)
end if
aacheck%elem(ii,jj)= aa%elem(ii,jj)
aacheck%elem(jj,ii)= aa%elem(jj,ii)
end do
end do
aa%m_trace=(0d0,0d0)
aa%m_det=(0d0,0d0)
aacheck%m_trace=(0d0,0d0)
aacheck%m_det=(0d0,0d0)

do ii= 1, numrow
aa%eval(ii)=(0d0,0d0)
aacheck%eval(ii)=(0d0,0d0)
end do

end subroutine init_hermmat


subroutine print_matrices (AA, namefile)
! This subroutine prints a type doublecomplex_matrix:
!the numbers of row and columns, the elements (in the proper
   order)
!, the trace and the determinant. This is all printed on a
   file
type(dcm) :: AA
character(:), allocatable :: namefile
integer :: ii,jj
integer :: my_stat
```

```
character (256) :: my_msg

open(unit = 40, file = namefile, status = "unknown",
$                        iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣",
$                       my_stat, "␣msg␣=␣"//trim(my_msg)
end if

write (40,*) "NUMBER␣OF␣ROWS:", AA%nr
write (40,*) "NUMBER␣OF␣COLUMNS:", AA%nc
write (40,*) "ELEMENTS:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) (AA%elem(ii, jj), jj = 1, AA%nc)
end do
write (40,*) "TRACE:", AA%m_trace
write (40,*) "DET:", AA%m_det
write (40,*) "EIGENVALUES:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) AA%eval(ii)
end do
close(40)

end subroutine print_matrices

subroutine print_vector (vec, nn, namefile)
! This subroutine prints a vector on file,
! given its length nn
character(:), allocatable :: namefile
integer :: ii,nn
integer :: my_stat
character (256) :: my_msg
double precision, dimension(:), allocatable :: vec

open(unit = 80, file = namefile, status = "unknown",
$                        iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣"
$                         , my_stat, "␣msg␣=␣"//trim(my_msg)
end if
```

```
do ii=1, nn !do cycle in order to print
write (80,*) vec(ii)
end do

end subroutine print_vector

end module matrices

c/////////////////////////////////////////////////////////
module normalized_spacings

use matrices

contains

subroutine normspac(aa,kk,normsp,debug)
!computes normalized spacings between eigenvalues
!of a double complex matrix, which is inside type aa
type(dcm) :: aa
integer :: kk
!kk-1 is the number of the spacings to be considered
!in the normalization computation
integer :: ii
integer :: iimax,iimin
double precision, dimension(:), allocatable :: sp
double precision, dimension(:), allocatable :: norm
double precision, dimension(:), allocatable :: normsp
integer :: my_stat
character (256) :: my_msg
character(:), allocatable :: namefile
logical debug


allocate(sp(aa%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate normsp with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(norm(aa%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate normsp with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
```

```
end if

!spacings
do ii=1, aa%nr-2
sp(ii)=aa%eval(ii+1)-aa%eval(ii)
end do

if(debug.eqv..TRUE.) then!check after changing
namefile="SPACINGS.txt"
call print_vector(sp, aa%nr-1,namefile)
end if

!NORMALIZATION and NORMALIZED SPACINGS

!if cycle to be sure to have (kk-1) in [1,nn-1]
if((kk-1)>(aa%nr-1)) then

print*, "␣Too␣many␣spaces␣(kk-1)in␣the␣normalization"
print*, "␣Rescaled␣to", (aa%nr-1)
print*, "␣It␣was", kk
kk=aa%nr
else if((kk-1)<1) then
print*, "␣Not␣enough␣spaces␣(kk-1)␣in␣the␣normalization"
print*, "␣Rescaled␣to␣2"
print*, "␣It␣was", kk
kk=2
end if

!if cycle to distinguish between odd case and even case
do ii=1,aa%nr-1
!setting the borders of the interval of normalization
if(mod(kk,2)==1) then
iimax=ii+(kk-1)/2
iimin=ii-(kk-1)/2
else
iimax=ii-1+(kk)/2
iimin=ii-(kk)/2
end if
!boundary conditions

if(debug.eqv..TRUE.) then!check before changing
print*, "BEFORE␣corrections"
print*, "Spacing␣number", ii
print*, "right␣border:", iimax
print*, "left␣border:", iimin
```

```
print*, "␣"
end if
if(iimax>aa%nr) then!exceeding at right
iimax=aa%nr
iimin=aa%nr-(kk-1)
else if(iimin<1) then!exceeding at left
iimin=1
iimax=kk
end if

if(debug.eqv..TRUE.) then!check after changing
print*, "AFTER␣corrections"
print*, "Spacing␣number:", ii
print*, "right␣border:", iimax
print*, "left␣border:", iimin
print*, "␣"
end if

!normalization of ii-spacing
norm(ii)=(aa%eval(iimax)-aa%eval(iimin))/(kk-1)

if(debug.eqv..TRUE.) then!check after changing
namefile="NORMS.txt"
call print_vector(norm, aa%nr-1,namefile)
end if

!normalized ii-spacing
normsp(ii)=sp(ii)/norm(ii)

end do
end subroutine normspac




end module normalized_spacings



c//////////////////////////////////////////////////////

!MODULE DEBUGGING
module debugging
```

```
use matrices

interface check
module procedure checkdim,checkmat
end interface

contains
subroutine checkdim(nn,nncheck,debug)!checking dimensions nn
implicit none
integer :: nn, nncheck
logical :: debug

if(debug.eqv..TRUE.) then

if(nn>10000) then !is the dim too large?
print*, "WARNING: too large dimension:"
print*, nn
else if(nn<1) then !is the dim minor than 1?
print*, "WARNING: dimension minor than 1"
else if((nn-nncheck)>0.5) then !is the dim wrong?
print*, "the dimension is wrong, it should be:"
$                                , nncheck
print*, "but is", nn
print*, " "
else
print*, "okay: right dimensions matrices",nn
end if
end if
end subroutine checkdim

subroutine checkmat(nn,mm,mmcheck,debug)
implicit none
!checking if the input matrixes are equal
type(dcm) :: mm
type(dcm) :: mmcheck
integer :: tt,ss,nn
integer*4 accum
logical debug

if(debug.eqv..TRUE.) then
accum=0
do tt=1,nn
do ss=1,nn
if(abs(mm%elem(tt,ss)-mmcheck%elem(tt,ss))
```

```
$                          /abs(mmcheck%elem(tt,ss))> 10E-10)
   then
accum=accum+1
end if
end do
end do

if(accum>0) then
print*, "The two matrices have"
$                          , accum, "different entries"
else
print*, "Same matrices"
end if

end if
end subroutine checkmat

end module debugging



!PROGRAM
program eigenproblem

use debugging
use matrices
use normalized_spacings

type(dcm) :: aa
type(dcm) :: aacheck
type(dcm) :: ddcheck
type(dcm) :: dd

integer  ii, jj, nn, nncheck
logical :: debug
character*1 :: choice
integer :: my_stat
character (256) :: my_msg
integer, allocatable :: ipiv(:)
integer :: iinfo

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
integer iinfodiag
```

```
character(:), allocatable :: lufile, diagfile, normfile

double precision, dimension(:), allocatable :: normsp
integer kk
logical diagherm
type(dcm) :: diag
type(dcm) :: diagcheck
real*8 :: temp
double precision, dimension(:), allocatable :: normspdiag


diagherm = .true. !if it is .true. then the procedure is done
!only on Hermitian matrices.
!If .false on Diagonal.

choice="n" !Initialization of choice variable
!if it is "X", it asks the debug
!otherwise the programmer can choose between
!doing the debug or not writing "y" or "n"

!Do you want to debug?
if(choice=="X") then
print *, "Do you want to debug?"
print *, "y for yes, n for no"
read*, choice
end if

if(choice=="y") then
debug=.TRUE.
else if(choice=="n") then
debug=.FALSE.
else
print*, "Not understood"
stop
end if

!HERE THE MATRIX DIMENSION IS TAKEN AS IMPUT FROM
!FILE "MatDimension.txt"
!AND THE PARAMETER REGARDING NORMALIZED SPACINGS
!BETWEEN EIGENVALUES

open(unit = 30, file = "MatDimension.txt",
$       status = "unknown",
$  iostat=my_stat, iomsg=my_msg)
```

```
if(my_stat /= 0) then
print*, 'Open -MatDimension- failed with stat = '
$                          , my_stat, ' msg = '//trim(my_msg)
end if

read(30,*) nn, kk
nncheck=nn

if(diagherm.eqv..true.) then

!INITIALIZATION aa and aacheck:
!random numbers in range [0,1], flat distribution
!hermitian matrices

call init_hermmat (aa, aacheck, nn, nn,debug)

!DEBUG
if(debug.eqv..TRUE.) then
print*, " "
print*, "INITIALIZATION DEBUG"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if

!INITIALIZATION dd:
!random numbers in range [0,1]
!hermitian matrices
!later they will be overwritten
!dd used for diagonalization
!ddcheck useless
call init_hermmat (dd, ddcheck, nn, nn, debug)



!DIAGONALIZATION

!the matrix dd%elem is equal to aa%elem
do ii=1,nn
do jj=1,nn
dd%elem(ii,jj)=aa%elem(ii,jj)
end do
end do

!calling the LAPACK subroutine for eigen values
```

```
!necessary to allocate work and rwork
!and initialize lwork
!for information read relative documentation
lwork= 2*nn

allocate(work(lwork),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate work with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(rwork(lwork),
$                         stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate rwork with stat = '
$                 , my_stat, ' and msg = '//trim(my_msg)
end if


call zheev( "N", "U", nn , dd%elem, nn , aa%eval , work ,
   lwork,
$                         rwork, iinfo1 )
!"N"->only eigenvalues
!"U"->upper triangle of aa is stored, inside dd%elem
! aa%eval will contain the eigen values of aa
!in ascending order(if INFO==0)

!DEBUG
if(debug.eqv..TRUE.) then
print*, " "
print*, "DIAGONALIZATION"
print*, " "
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if

if(debug.eqv..TRUE.) then!checkmat
print*, " "
print*,"aa and aacheck"
call check(nn,aa,aacheck,debug)
end if
```

```
if(debug.eqv..TRUE.) then!check diag
if (iinfo1==0) then
print*, "␣"
print*, "Successful␣DIAGONALIZATION"
else if (iinfo1 < 0) then
print*, "␣"
print*, "the", iinfo1, "-th␣argument
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣of␣ipiv␣had␣an␣illegal␣value"

else
print*, "␣"
print*,  "the␣algorithm␣failed␣to␣converge"
end if
end if

if(debug.eqv..TRUE.) then!print diag
diagfile="Eigenvalues.txt"
call print_matrices (aa, diagfile)
end if

!NORMALIZED SPACINGS BETWEEN EIGENVALUES

!allocations

allocate(normsp(aa%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣normsp␣with␣stat␣=␣'
$                  , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if


!calling the subroutine normspac to compute normalized
   spacings
!IMPUT: the type aa, the parameter kk (kk-1 spacings
!considered in the normalization) taken as imput from file
!OUTPUT: the vector of normalized spaces, nn-1 entries

call normspac(aa,kk,normsp,debug)

normfile="NormSpacingsHerm.txt"

call print_vector(normsp,nn-1,normfile)
```

```
!*********************************************************
else
!*********************************************************

!COMPUTING NORMALIZED SPACING for diagonal real matrices

!INITIALIZATION diag:
!using the same function used for dcm type
!just to use the same subroutines
!then we RE-INTIALIZE to diagonal real matrices
!same dimension as hermitian ones
call init_hermmat (diag, diagcheck, nn, nn, debug)

do ii=1,nn
do jj=1,nn
if(ii==jj) then
call random_number(temp)
diag%elem(ii,ii)=complex(temp, 0d0)
diagcheck%elem(ii,ii)=diag%elem(ii,ii)
diag%eval(ii)= temp
else
diag%elem(ii,jj)=complex(0d0, 0d0)
diagcheck%elem(ii,jj)=diag%elem(ii,jj)
end if
end do
end do

!SORTING OF EIGENVALUES
!via the LAPACK subroutine hpsort_eps_epw
!"I" for incresing order
!"D" for decreasing

call dlasrt ("I",diag%nr, diag%eval, iinfo)

if(debug.eqv..TRUE.) then!check diag
if (iinfo1==0) then
print*, "␣"
print*, "Successful␣SORTING"
else if (iinfo1 < 0) then
print*, "␣"
print*, "the", iinfo1, "-th␣argument
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣had␣an␣illegal␣value"

end if
end if
```

```
if(debug.eqv..TRUE.) then!print diag
diagfile="Eigenvalues.txt"
call print_matrices (diag, diagfile)
end if

!NORMALIZED SPACINGS
!allocations

allocate(normspdiag(diag%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate normspdiag with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if


!calling the subroutine normspac to compute normalized
   spacings
!IMPUT: the type diag, the parameter kk (kk-1 spacings
!considered in the normalization) taken as imput from file
!OUTPUT: the vector of normalized spaces, nn-1 entries

call normspac(diag,kk,normspdiag,debug)

normfile="NormSpacingsDiag.txt"

call print_vector(normspdiag,nn-1,normfile)

end if

stop
end program eigenproblem
```

Listing 2: Scripting: collecting the normalized spacings data. Plotting and analyzing them calling a Gnuplot code.

```
#######################################################
#This script, given a matrix dimension and a locality
#parameter kk, print them on file. The program of ex5
#read them, and then computes normalized spacings of
#a random Hermitian matrix.
#This is done a certain number of times (nrep),
#collecting the data. From the data we obtain an
#histogram.
```

```
#Given a matrix dimension, this can be done for
#different kk, from 2 to the matrix dimension,
#choosing the number of kk within that interval.
#Then a fit with the distribution
#P(s)=a*s**(alpha)*exp(-b*s**(beta)) is performed,
#as well as absolute, relative and log of relative
#residuals. All the results are moved in a dedicated
#folder, called "Results_nn_kk".
#This can be iterated over different matrix dimension.
#The suffix "Herm" in the name of the file is due to
#the fact that this scripting was originally meant for
#Hermitian matrix. Treating different matrices is
#convenient to change the suffix, like "Diag" for
#diagonal matrices (just using "Find and Replace").
#####################################################

import numpy as np
import sys
import os
import subprocess
from matplotlib import pyplot
import matplotlib.pyplot as plt

nmin=1850
#lower bound dimension matrix
nmax=2100+250
#upper bound dimension matrix
stepp=250
#step of the increasing dimension



for ii in range(nmin,nmax,stepp):

numkk=10
#number of different kk
kklist = [0]*numkk
for mm in range(0,numkk,1):
kklist[mm]= int(ii*pow(float(ii),-float(mm)/float(numkk+1)))
#from kk almost 2 to ii (not costant increase)
kklist1 = [0]*(numkk-3)
kklist1[0]=kklist[0]
for ll in range(4,numkk,1):
kklist1[ll-3]=kklist[ll]
```

```
print(kklist1)
for kk in kklist1:
f = open("MatDimension.txt","w")
#opening the file with the dimension of the matrix
f.write(str(ii)+"␣␣␣␣␣␣␣"+str(kk))
#write the dimension on the opened file and kk
f.close()
norm = open("ResultsHerm-%d-%d.txt" %(ii ,kk),"w")
#open file which will collect data per fixed dimension
#and fixed kk

numrip=30
#number ripetitions for each dimension and kk
for jj in range(1,numrip+1,1):
subprocess.call("./a.out")
#execute the fortran program
nsp = open("NormSpacingsHerm.txt","r")
nsplines = nsp.readlines()
#reading norm spacings from file

for qq in nsplines:
norm.write(qq)
#copying data in file "Results-ii"
nsp.close()
#closing file
norm.close()


B=30 #number of bins


normlines = np.loadtxt("ResultsHerm-%d-%d.txt" %(ii ,kk),
   unpack=True)
#reading all norm spacings for fixed dimension and kk



hist, binedges=np.histogram(normlines, B, density=1)#
   normalized to 1
#binedges has len= len(hist)+1
normhist = open("ResultsHerm_hist-%d-%d.txt" %(ii ,kk),"w")

x=np.zeros(B)
for pp in range(0, B):
x[pp]=0.5*(binedges[pp]+binedges[pp+1])
```

```
#middle points of the bins, same lenght of hist
normhist.write(str(x[pp])+ "␣" + str(hist[pp]) + "\n")

normhist.close()

res = open("ResultsHerm_hist-%d-%d.txt" %(ii ,kk),"r")
reslines = res.readlines()
#reading data from result file number ii

temp = open("resultstemp","w")
for jj in reslines:
temp.write(jj)
#copying data in temporary file
res.close()
temp.close()
#closing files

os.system("gnuplot␣fitting")
#fit with law P(s)=a*s**(alpha)*exp(-b*s**(beta))

newnamefit= ("DistributionfitHerm-%d-%d.pdf" %(ii ,kk))
os.rename("tempfit.pdf",newnamefit)
#renaming the pdf file of the fit

newnameabs= ("absresHerm-%d-%d.pdf" %(ii ,kk))
os.rename("tempabsres.pdf",newnameabs)
#renaming the file of the absolute residuals

newnamerel= ("relresHerm-%d-%d.pdf" %(ii ,kk))
os.rename("temprelres.pdf",newnamerel)
#renaming the file of the relative residuals

newnamelog= ("logrelresHerm-%d-%d.pdf" %(ii ,kk))
os.rename("templogrelres.pdf",newnamelog)
#renaming the file of the log of the relative residuals

fitlog = open("fit.log","r")
fitlines = fitlog.readlines()
#reading parameters of the fit from "fit.log"

fitlogd = open("ParametersfitHerm-%d-%d" %(ii ,kk),"w")
for jj in fitlines:
fitlogd.write(jj)
#copying parameters on "parametersfit-ii-kk"
fitlog.close()
```

```
fitlogd.close()
#closing files

os.remove("fit.log")
#removing "fit.log

subprocess.call("mkdir␣RESULTS_"+str(ii)+"_"+ str(kk), shell=
   True)
subprocess.call("mv␣*-"+ str(ii)+"-"+ str(kk)+"*␣␣RESULTS_"+
   str(ii)+"_"+ str(kk), shell=True)
#moving all files -nn-kk in a dedicated folder called "
   Results_nn_kk"
```

Listing 3: Scripting: collecting the parameters value of the distribution's fits. Plotting them calling a Gnuplot code, with the matrix's dimension fixed. Computing the average and th error of the average, with the matrix's dimension fixed.

```
#####################################################
#This script must be placed inside the folder where the
#parameters of the fits of the distribution of the normalized
#spacings of (Hermitian) matrices are saved (file which
   format
#is the one of gnuplot's fit.log). Every file should be named
#with a name which contain the matrix's dimension and the
#value of the locality parameter kk, like "ParametersfitHerm-
   N-%#kk.txt".
#For each on of those file, he read from them only the values
    of #the parameters and their errors. Fixed the matrix
   dimension,
#the script plots the parameters against kk ("
   FitParametersHerm-%d.pdf" ) also computing the
#average and the error on the average for each parameter.
#The values of the paramters (fixed the matrix dimension)
#are printed on a file which name is "FitParHerm-N.txt" . The
    #average and its error are printed on a file which name
   is #MeanVarianceParHerm-N.txt".
#The suffix "Herm" in the name of the file is due to
#the fact that this scripting was originally meant for
#Hermitian matrix. Treating different matrices is
#convenient to change the suffix, like "Diag" for
#diagonal matrices (just using "Find and Replace").
#####################################################

import numpy as np
import sys
import os
```

```
import subprocess
from matplotlib import pyplot
import matplotlib.pyplot as plt
import math

nmin=1100
#lower bound dimension matrix
nmax=2100+250
#upper bound dimension matrix
stepp=250
#step of the increasing dimension
numn=(nmax-nmin)/(stepp)

amean_all=[0]*(numn)
alphamean_all=[0]*(numn)
bmean_all=[0]*(numn)
betamean_all=[0]*(numn)
#vectors of means for different dimensions

accumii=0

for ii in range(nmin,nmax,stepp):

numkk=10
#number of different kk
kklist = [0]*numkk
for mm in range(0,numkk,1):
kklist[mm]= int(ii*pow(float(ii),-float(mm)/float(numkk+1)))
#from kk almost 2 to ii (not costant increase)
kklist1 = [0]*(numkk-3)
kklist1[0]=kklist[0]
for ll in range(4,numkk,1):
kklist1[ll-3]=kklist[ll]

fitpar = open("FitParHerm-%d" %ii, "w")
#open the file which will contain the parameters for
#different kk for each dimension matrix

meansigma = open("MeanVarianceParHerm-%d" %ii, "w")
#open the file which will contain the mean and variance
#of parameters for each dimension matrix

aval=[0]*(numkk-3)
alphaval=[0]*(numkk-3)
bval=[0]*(numkk-3)
```

```
betaval=[0]*(numkk-3)
#list of parameters values for different kk

accumkk=0

for kk in kklist1:
f = open("ParametersfitHerm-%d-%d" %(ii ,kk), "r")
#opening the file with the parameters of the fit
param = f.readlines()
#reading the lines

for pp in param:
#taking value and error of each fit parameter
#valid for files like fit.log from gnuplot
fields=pp.split("=")
if((fields[0] == "a                "):
avalerr=fields[1].split("+/-")
aval[accumkk]=float(avalerr[0])

#taking value and error of each fit parameter

if((fields[0] == "alpha            "):
alphavalerr=fields[1].split("+/-")
alphaval[accumkk]=float(alphavalerr[0])

if((fields[0] == "b                "):
bvalerr=fields[1].split("+/-")
bval[accumkk]=float(bvalerr[0])


if((fields[0] == "beta             "):
betavalerr=fields[1].split("+/-")
betaval[accumkk]=float(betavalerr[0])
f.close()


amean_all[accumii]=amean_all[accumii]+aval[accumkk]

alphamean_all[accumii]=alphamean_all[accumii]+alphaval[
   accumkk]

bmean_all[accumii]=bmean_all[accumii]+bval[accumkk]

betamean_all[accumii]=betamean_all[accumii]+betaval[accumkk]
#vectors of means and variances for dimension ii
```

```
fitpar.write(str(kk)+"␣␣"+str(aval[accumkk])+"␣␣"+str(
   alphaval[accumkk])+"␣␣␣␣␣␣"+str(bval[accumkk])+"␣␣"+str(
   betaval[accumkk])+ "\n")

accumkk=accumkk+1


amean_all[accumii]=amean_all[accumii]/float(numkk-3)

meansigma.write("Dimension:␣"+ str(ii)+ "\n")
meansigma.write("kk:␣"+ str(kk)+ "\n\n")
meansigma.write("a:␣"+ str(amean_all[accumii])+"\n\n")

alphamean_all[accumii]=alphamean_all[accumii]/float(numkk-3)
meansigma.write("alpha:␣"+ str(alphamean_all[accumii])+"\n\n"
   )

bmean_all[accumii]=bmean_all[accumii]/float(numkk-3)
meansigma.write("b:␣"+ str(bmean_all[accumii])+ "\n\n")

betamean_all[accumii]=betamean_all[accumii]/float(numkk-3)
meansigma.write("beta:␣"+ str(betamean_all[accumii])+"\n\n")
#vectors of means and sigmas for different dimensions

fitpar.close()
meansigma.close()

res = open("FitParHerm-%d" %ii, "r")
reslines = res.readlines()
#reading data from result file number ii


temp = open("resultstemp","w")
for jj in reslines:
temp.write(jj)
#copying data in temporary file
res.close()
temp.close()
#closing files

os.system("gnuplot␣parplotting")
#plotting the parameters

newnamefit= ("FitParametersHerm-%d.pdf" %ii)
```

```
os.rename("tempfit.pdf",newnamefit)
#renaming the pdf file of the fit
```

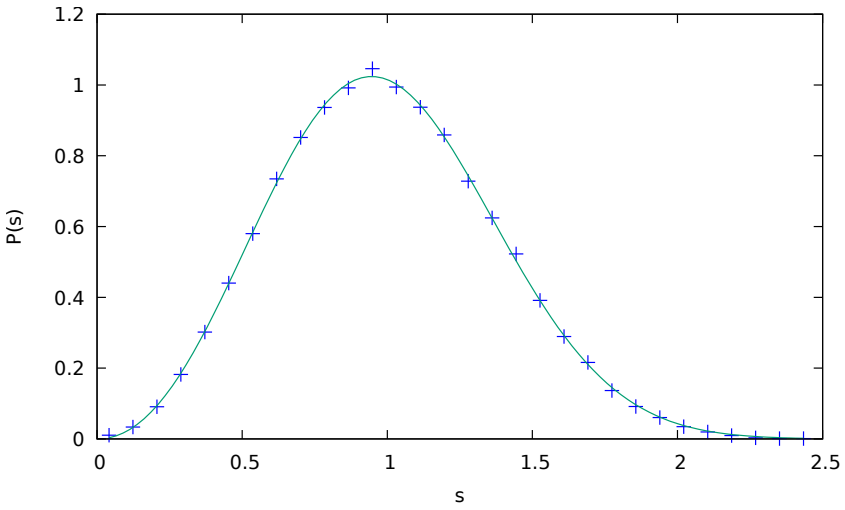
Results-Eigenproblem & Random matrix theory

We have tuned the matrix's dimension between 1100 and 2100. The locality parameter kk(we recall that is number of spacings taken into account in the local normalization) was tuned given the matrix's dimension nn via the following formula: $kk[ii] = nn \cdot (nn)^{-ii/numkk+1}$, where $numkk$ is the number of steps considered from 1 to nn. The $+1$ is due to the fact that we want to avoid the $kk = 1$ case. It was chosen to delete the kk between nn and 200, since the distribution's fit not converged for those values. Given the fact that nn is between 1100 and 2100, for each matrix's dimension there are 7 different kk. This was done both for Hermitian and Diagonal real matrices.

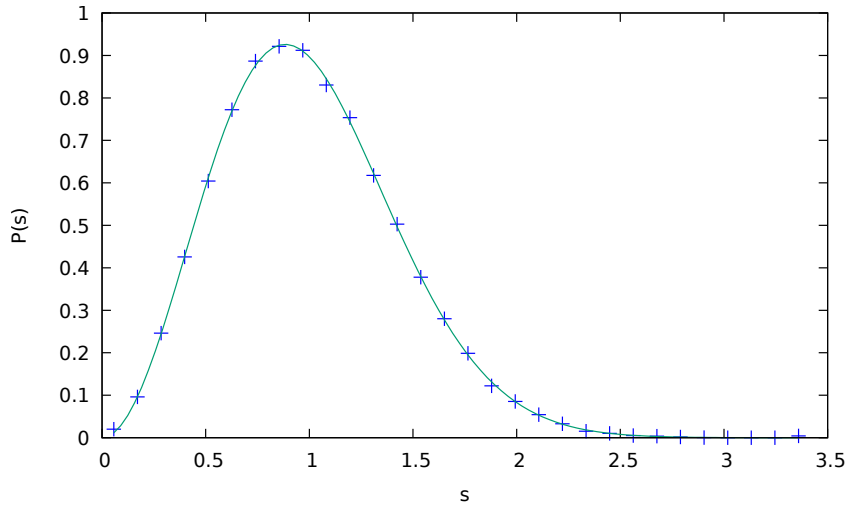
The fits were initialized with the literature values. The parameters of the fit are plotted (with the dimension of the matrix fixed) versus kk. For each dimension the average is computed.

Fit of the distribution- Hermitian Matrices

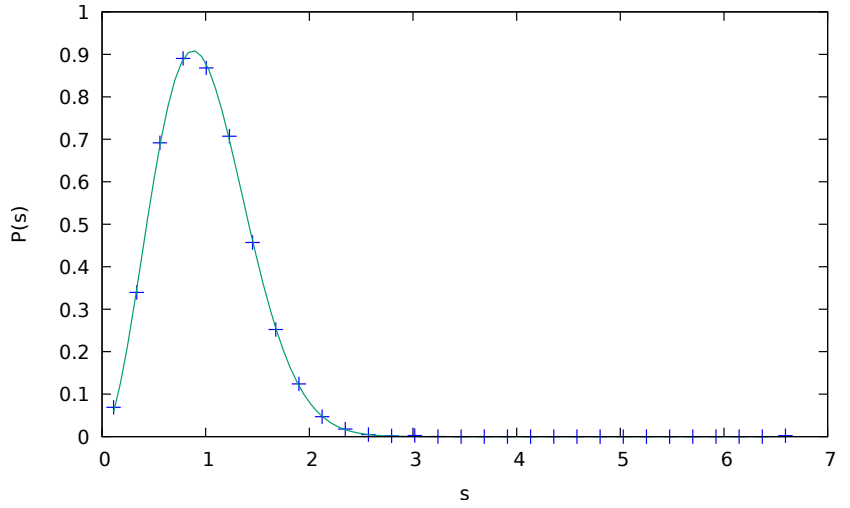
We have chosen to report only the fits of the distribution for one dimension, one for each kk. This choice was motivated by the fact that the shapes of the graphs do not change very much varying the dimension. The chosen dimension is 2100, the highest one. The absolute residuals of those fits, the relative ones and their logarithms were computed but only the last ones are in an appendix (the difference in scale between the distribution's points were such that only the logariths are reasonable to be shown).



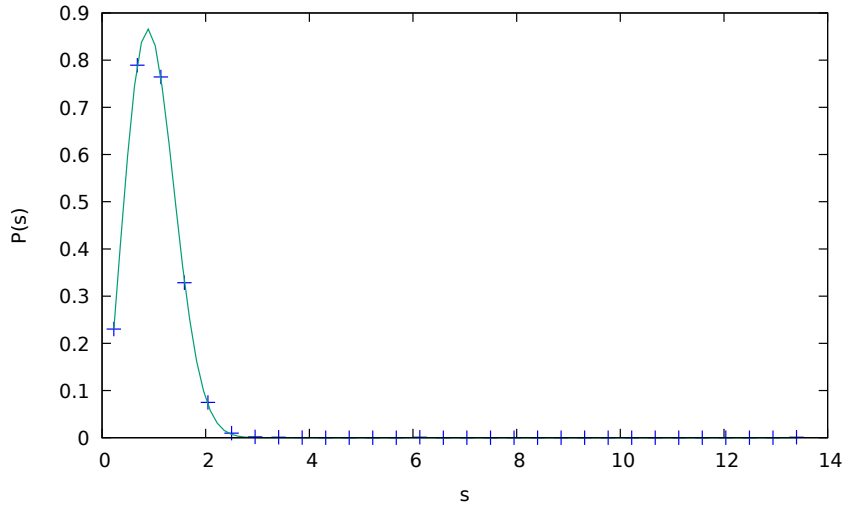
Hermitian: nn= 2100, kk= 4



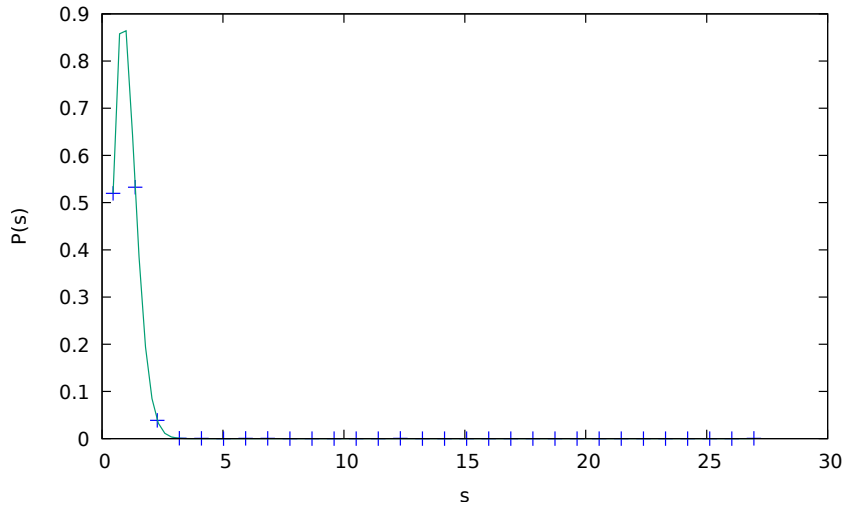
Hermitian: nn= 2100, kk= 8



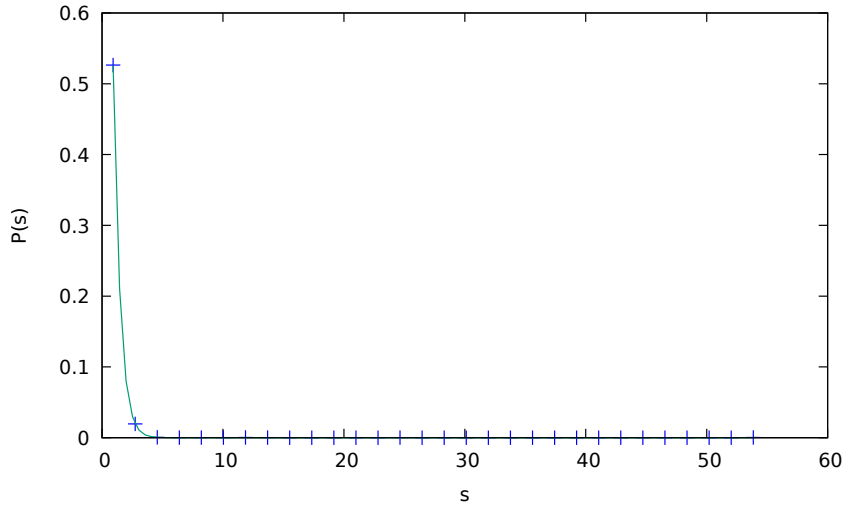
Hermitian: nn= 2100, kk= 16



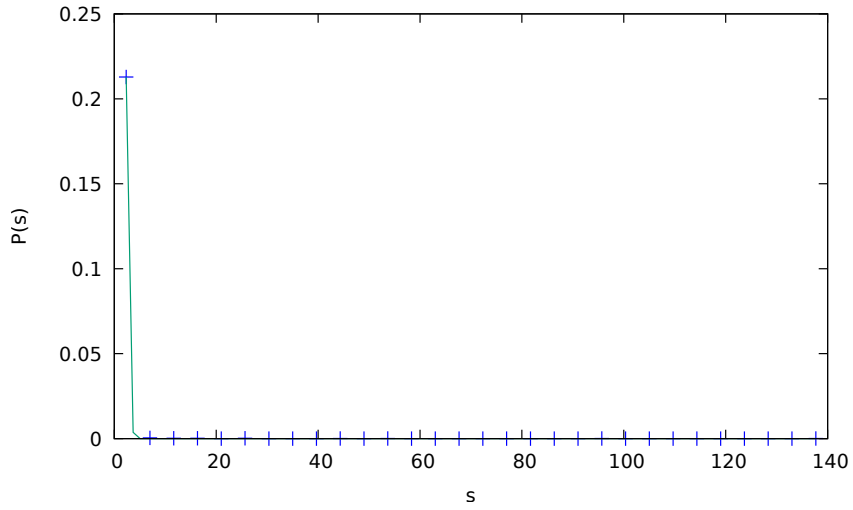
Hermitian: nn= 2100, kk= 32



Hermitian: nn= 2100, kk= 64

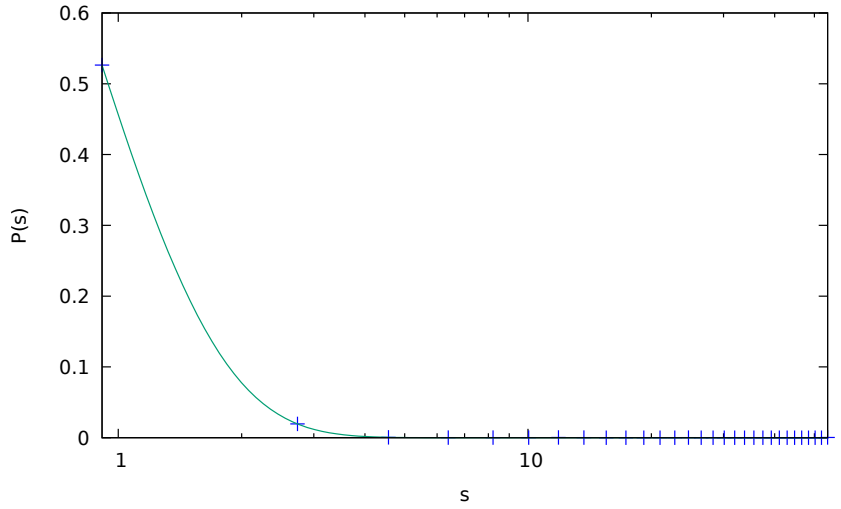


Hermitian: nn= 2100, kk= 130

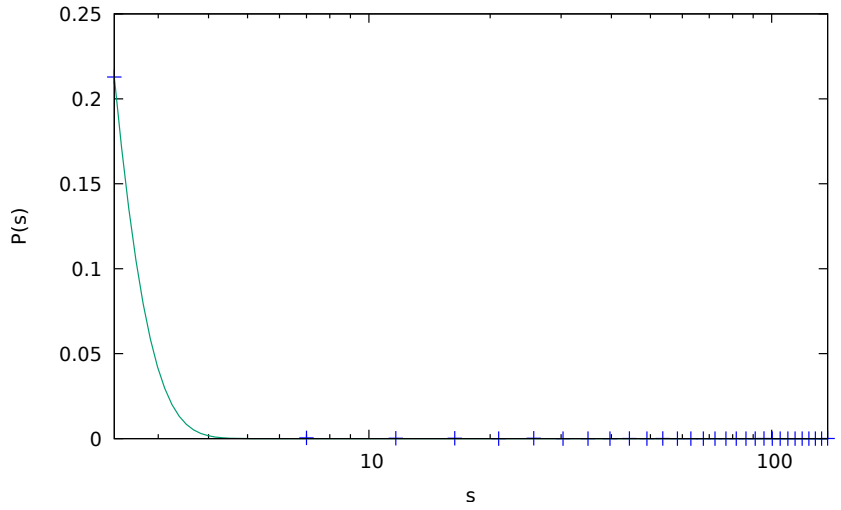


Hermitian: nn= 2100, kk= 2100

Log scale in the x-axis for kk=130 and kk=2100, in order to visualize in a better way.



Hermitian: nn= 2100, kk= 130. Log scale on x



Hermitian: nn= 2100, kk= 2100. Log scale on x

Parameters of the distribution- Hermitian Matrices

The plots are in logscale on the x-axis.

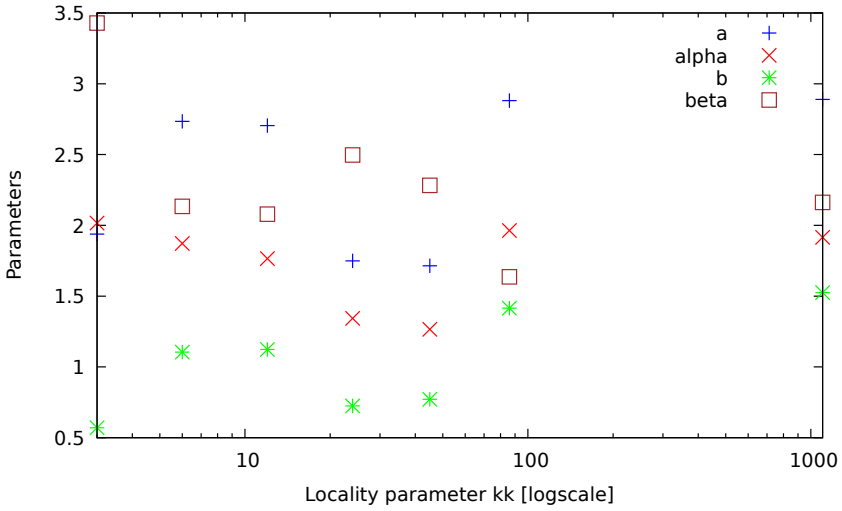


Figure 1: Hermitian: nn= 1100

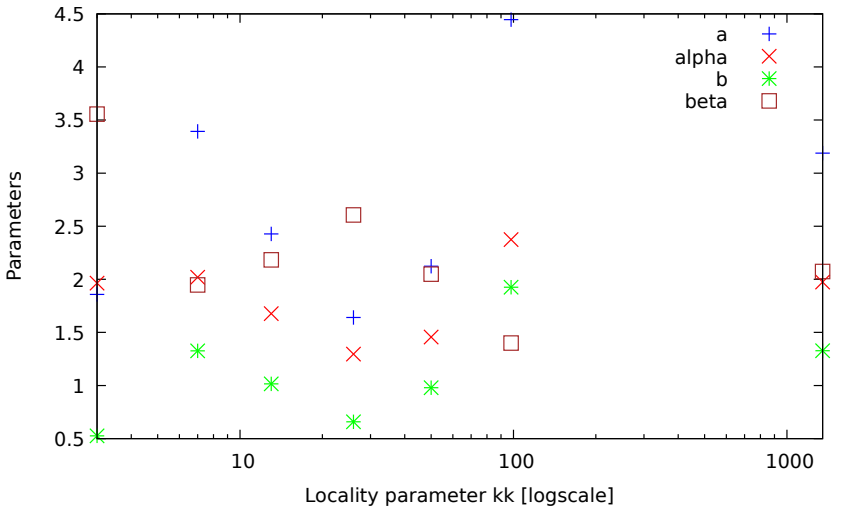


Figure 2: Hermitian: nn= 1350

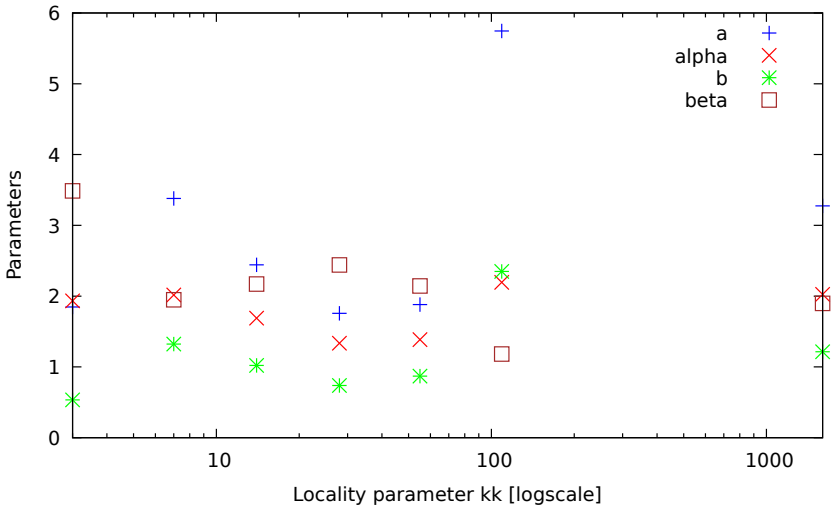


Figure 3: Hermitian: nn= 1600

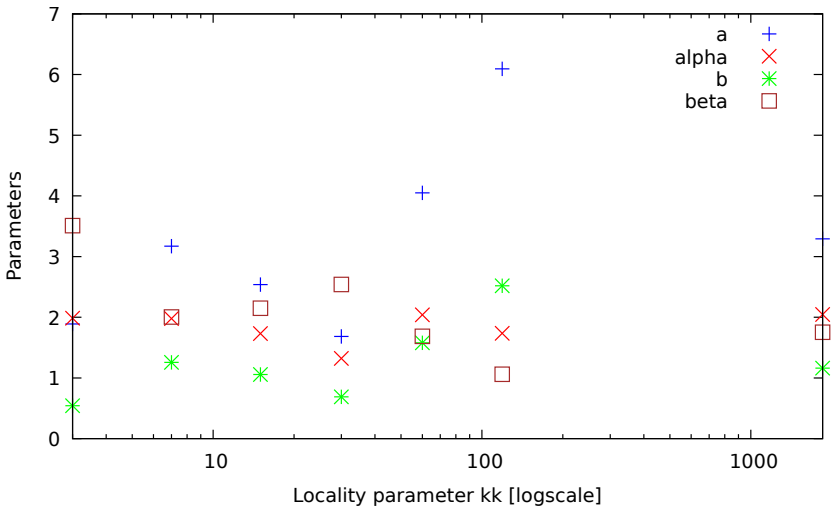


Figure 4: Hermitian: nn= 1850

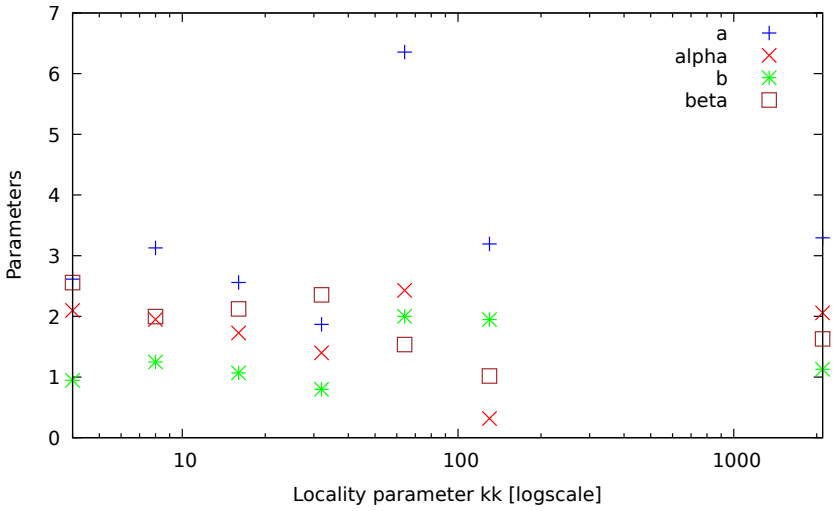
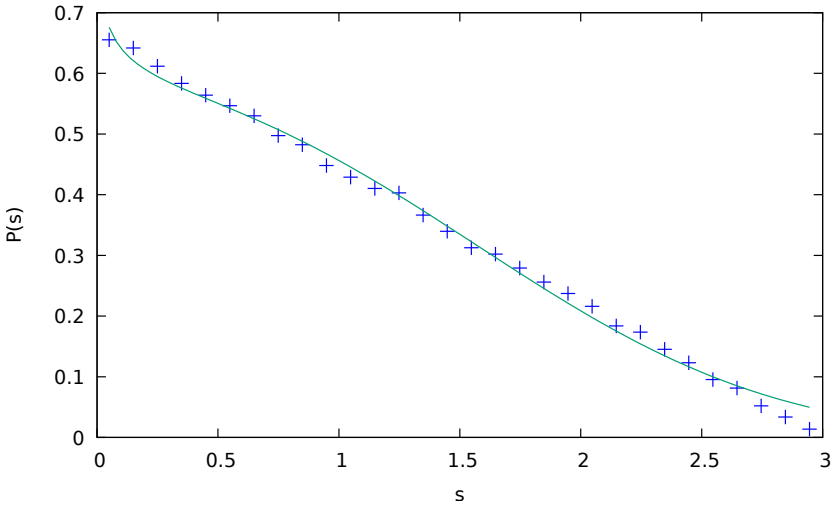


Figure 5: Hermitian: nn= 2100

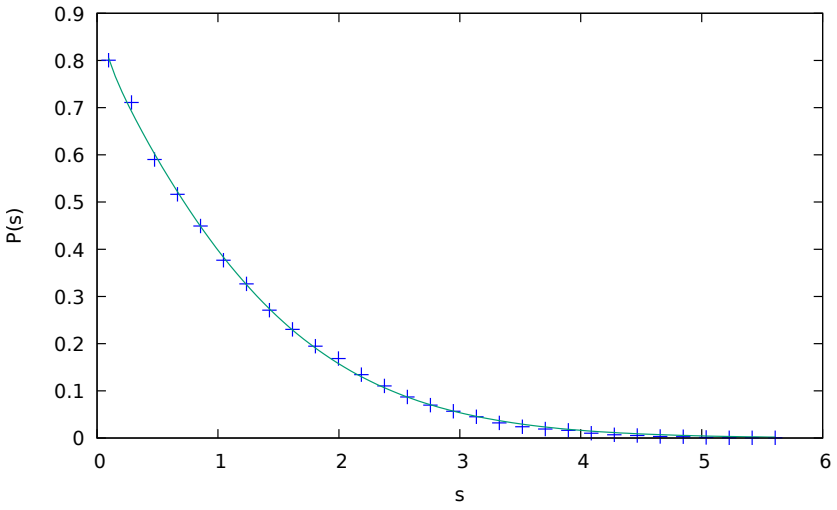
$n =$	1100	1350	1600	1850	2100	Lit.
$< a >$	2.37	3.06	3.34	3.72	3.82	3.24
$< \alpha >$	1.73	2.07	2.09	2.13	2.02	2
$< b >$	1.03	1.25	1.33	1.45	1.51	1.37
$< \beta >$	2.32	2.59	2.55	2.47	2.24	2

Fit of the distribution- Real Diagonal Matrices

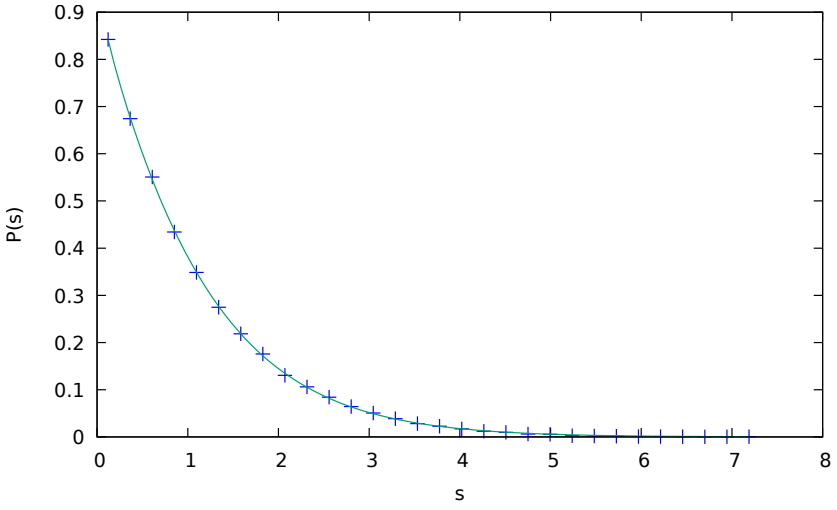
We have chosen to report only the fits of the distribution for one dimension, one for each kk. This choice was motivated by the fact that the shapes of the graphs do not change very much varying the dimension. The chosen dimension is 2100, the highest one. The logarithms of relative residuals are in an appendix.



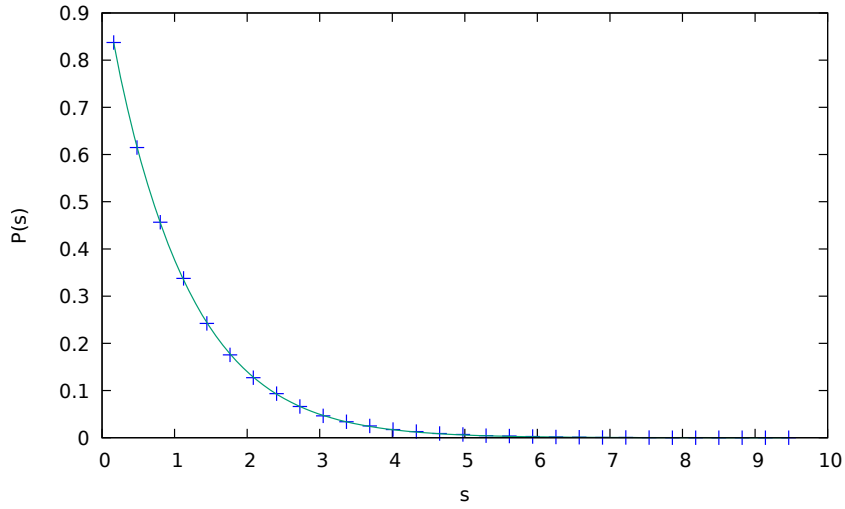
Real Diagonal: nn= 2100, kk= 4



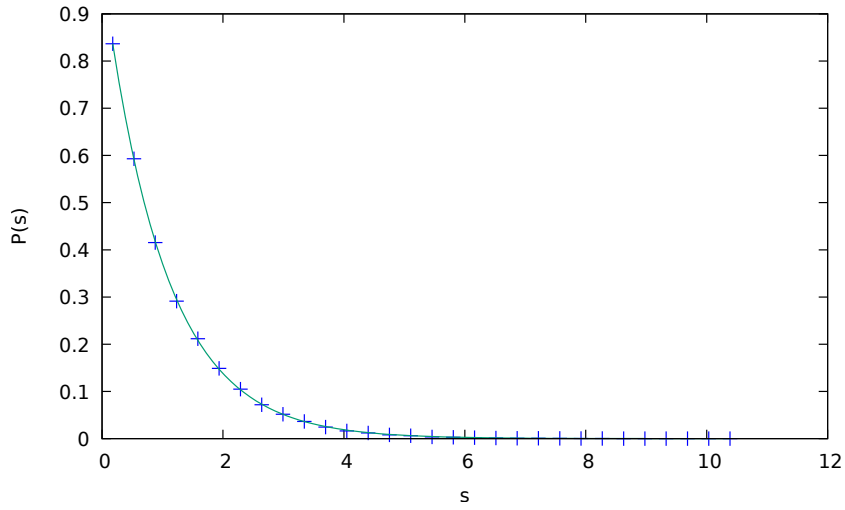
Real Diagonal: nn= 2100, kk= 8



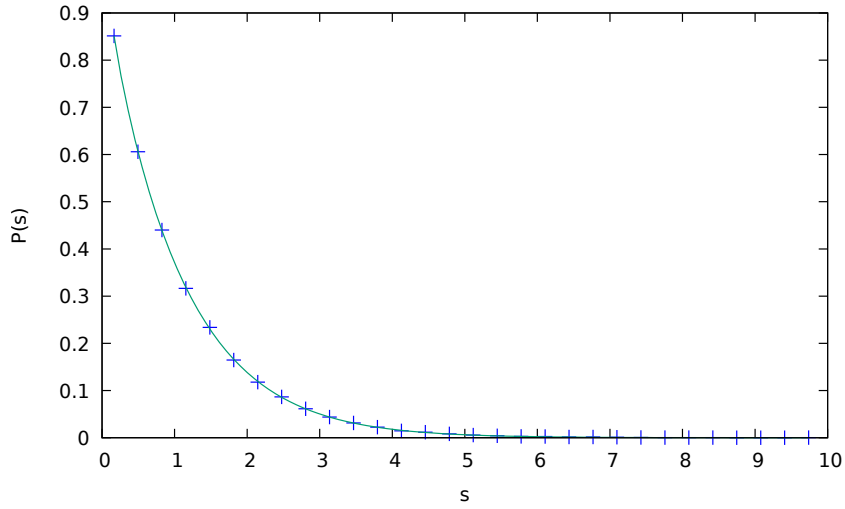
Real Diagonal: nn= 2100, kk= 16



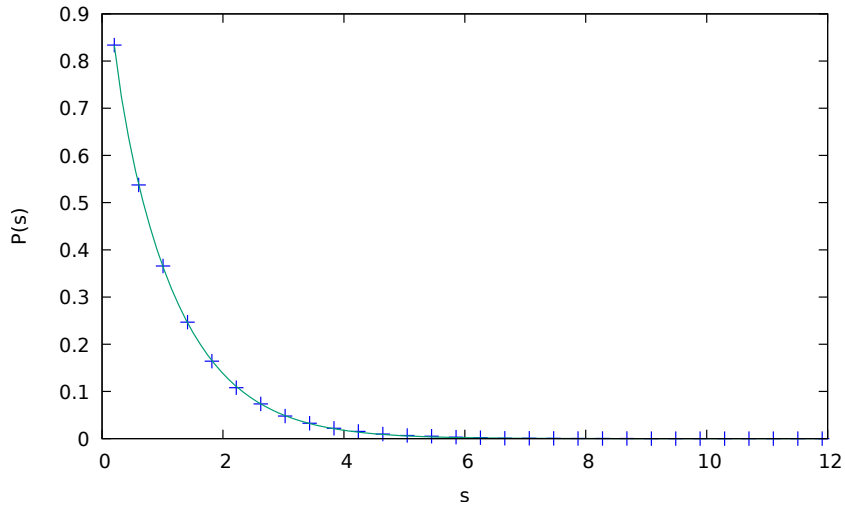
Real Diagonal: nn= 2100, kk= 32



Real Diagonal: nn= 2100, kk= 64

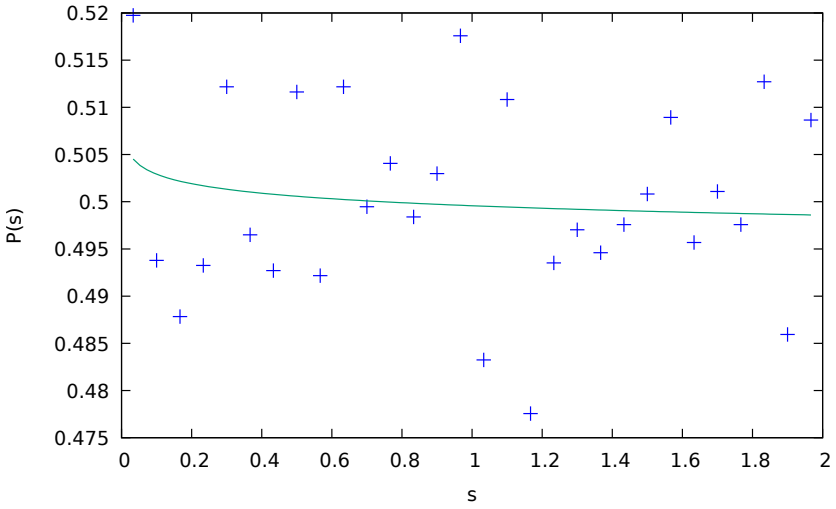


Real Diagonal: nn= 2100, kk= 130



Real Diagonal: nn= 2100, kk= 2100

We want to show also another figure, obtained for nn=1850 and kk=3, in order to show a figure with a lower kk than the ones showed above.



Real Diagonal: nn= 1850, kk= 3

Parameters of the distribution- Real Diagonal Matrices

The plots are in logscale on the x-axis.

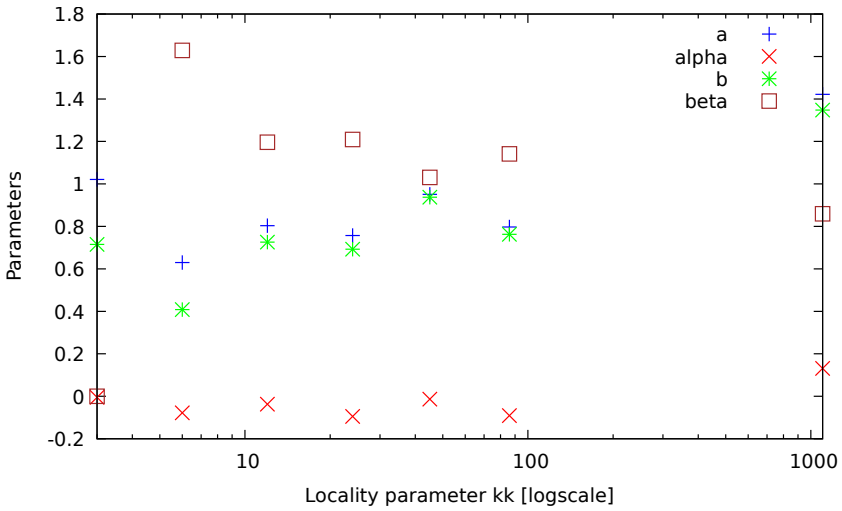


Figure 6: Real Diagonal: nn= 1100

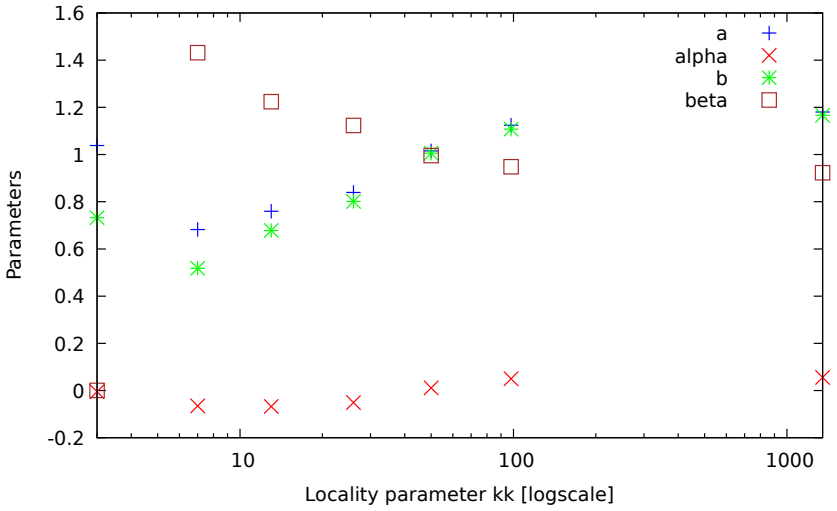


Figure 7: Real Diagonal: nn= 1350

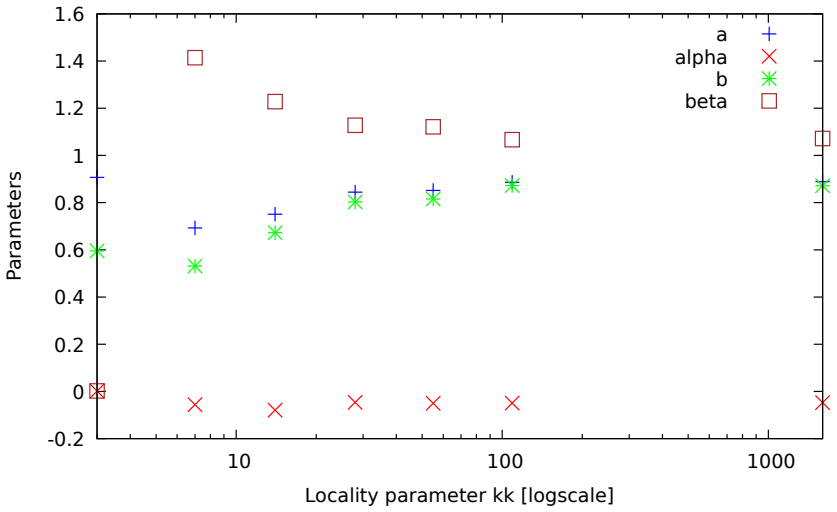


Figure 8: Real Diagonal: nn= 1600

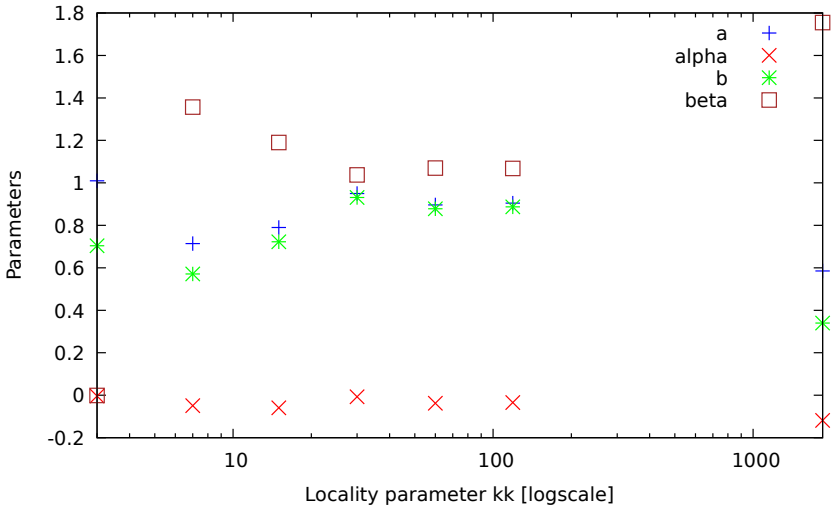


Figure 9: Real Diagonal: nn= 1850

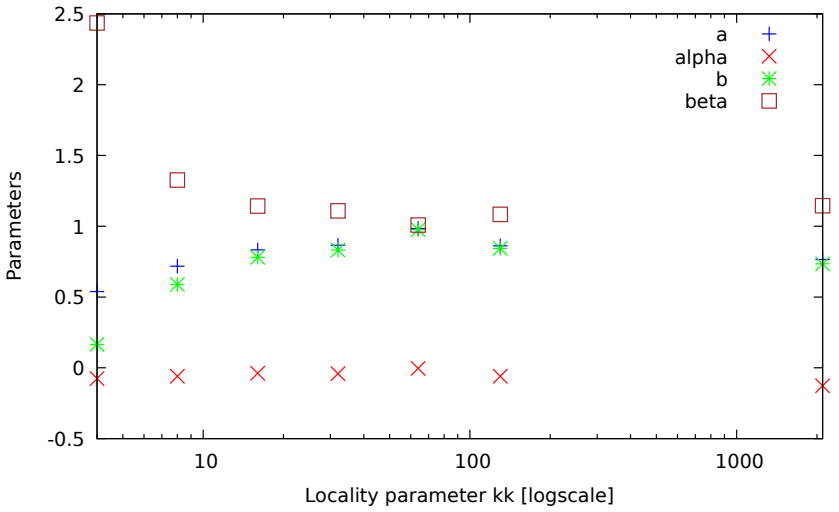


Figure 10: Real Diagonal: nn= 2100

$n =$	1100	1350	1600	1850	2100
$< a >$	0.91	1.08	0.98	0.97	0.93
$< \alpha >$	-0.03	-0.05	-0.05	-0.05	-0.06
$< b >$	0.80	1.25	0.88	0.84	0.82
$< \beta >$	1.01	1.09	1.16	1.23	1.50

Self-Evaluation-Eigenproblem & Random matrix theory

Comments on the Results-Hermitian

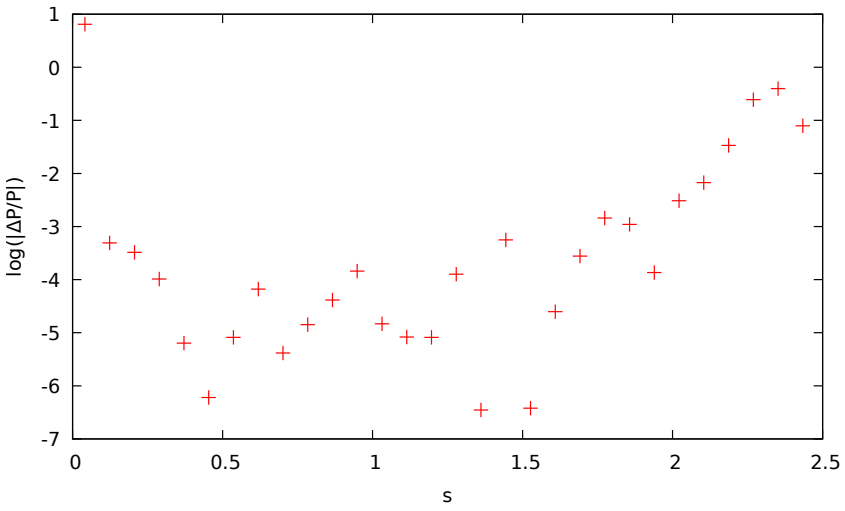
Looking at the fits of the distribution we notice that increasing kk the scale of the normalized spacings tend to enlarge. The peak of the distribution remain around 1, but the right tail of the distribution goes further and further, and the peak lowers. The fact that the peak is for normalized spacing is around 1 (actually less than 1) is reasonable: we are taking into account normalized spacings, which all should be 1 if normalized with itself, or if the spacings are all equal. Since there are spacings which are very far from 1 at right, we have an high probability to have a normalized spacing between 0 and 1: the peak is actually before 1. The fact that the range of distribution at right increase with the locality parameter kk reflects the fact that the big spacings are close to big spacings and small spacings are close to small spacings. In fact the more the normalization is local (the more kk is small), the more symmetric around 1 is the distribution. This means that big spacings are normalized with big spacings and small spacings are normalized with small spacings, hence big are close to big and small are close to small. The limit case is $kk = 1$: a Dirac delta in 1. Increasing the globality (and thus kk) we take into account further spacings. This results in a fall of the norm for big spacings and an increase of the norm for small spacings. This explains why the behaviour is so asymmetric incresing kk.
We can observe that the parameters' average are of the same order of the literature's ones.

Comments on the Results-Real Diagonal

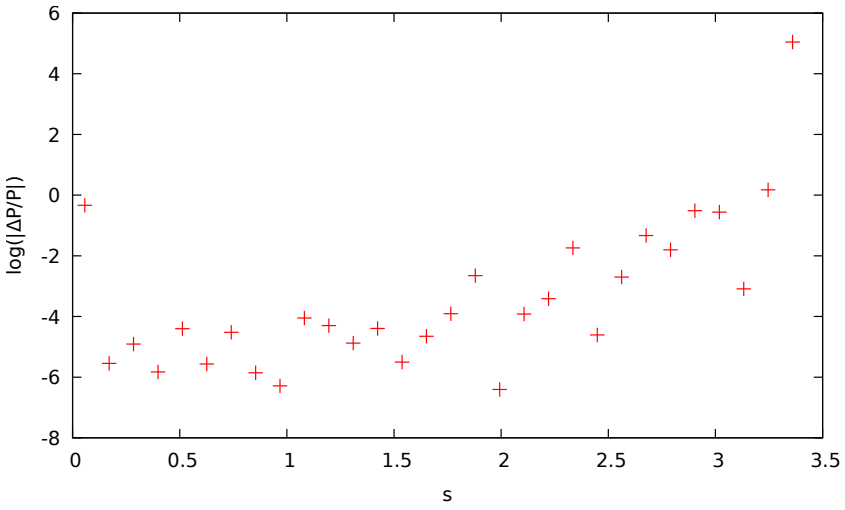
Looking at the fits of the distribution we notice a different behaviour respect the one of the Hermitian's distribution. The peak is not around one and the distribution seems to diverge in 0. Moreover, decreasing the locality parameter kk the distribution tends to become a costant. This is opposite to the Hermitian's case, where small spacings were close to small spacings and big spacings were close to big spacings. This means that there is no such an order in the spacings, which are casually placed around 1.
The cause could be the fact that the Real Diagonal matrices are directly initialized with uniform distribution, hence the spacings result casually placed around 1. On the other hand, the Hermitian matrices, although they have purely real eigenvalues like the Real Diagonal matrices and they are initialized with uniform distribution too, they encounter the process of Diagonalization. This systematic process could be the cause of the correct distribution of the spacings around 1. A new development could be initialize the Real Diagonal Matrices using gaussians and see if the results match with the literature and with the Hermitian's ones.
Another developmente could be understand why the fit do not converge with values of kk between the dimension matrix and 200. Is it merely a cause of numbers?

Logarithm of relative residuals, Distribution fits

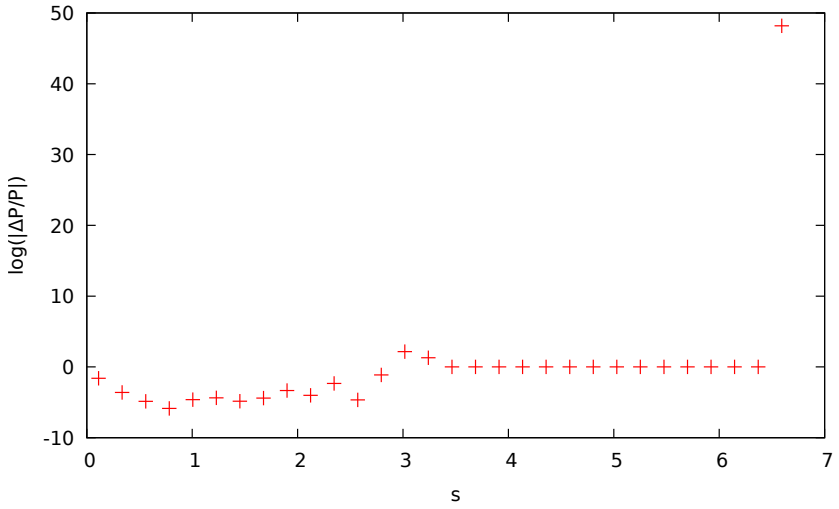
Hermitian matrices



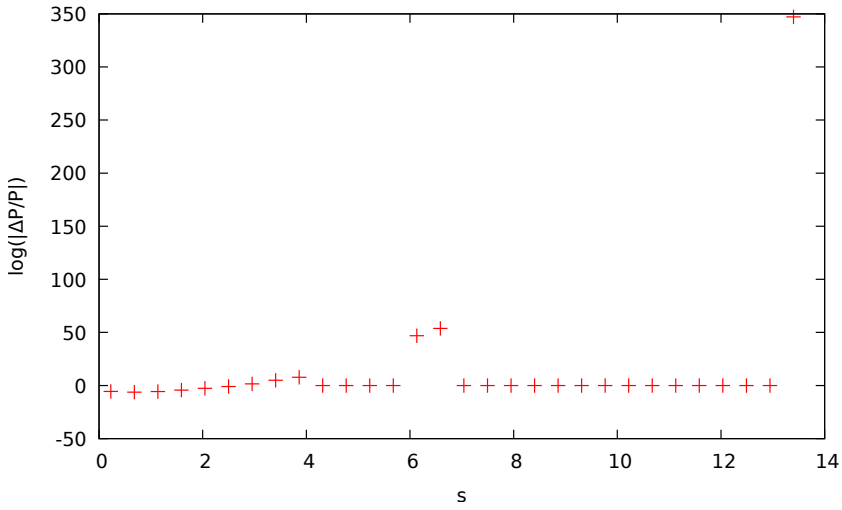
Hermitian: nn= 2100, kk= 4



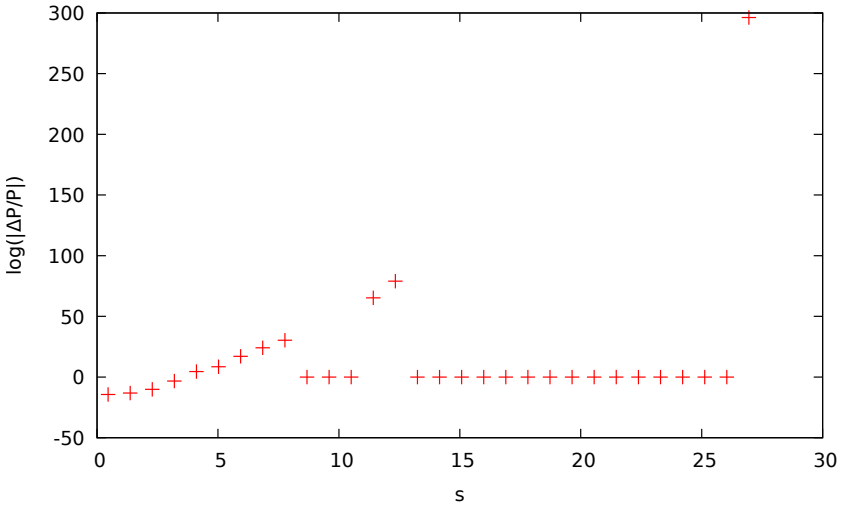
Hermitian: nn= 2100, kk= 8



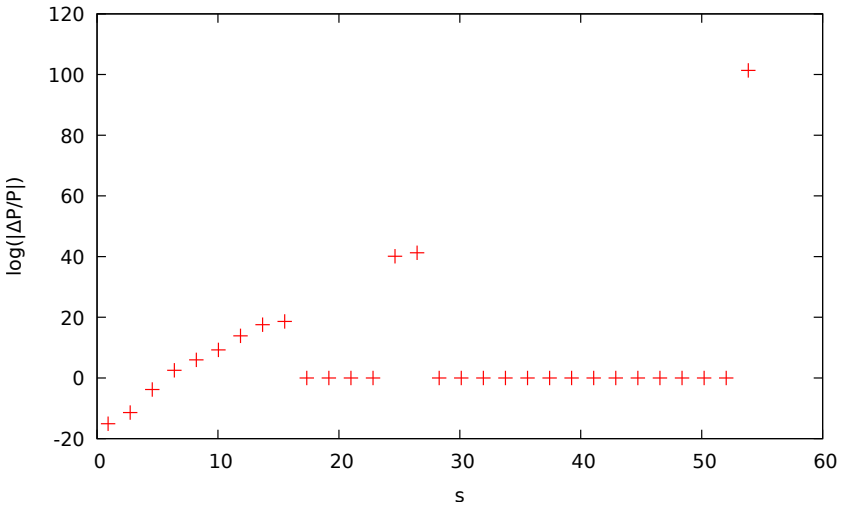
Hermitian: nn= 2100, kk= 16



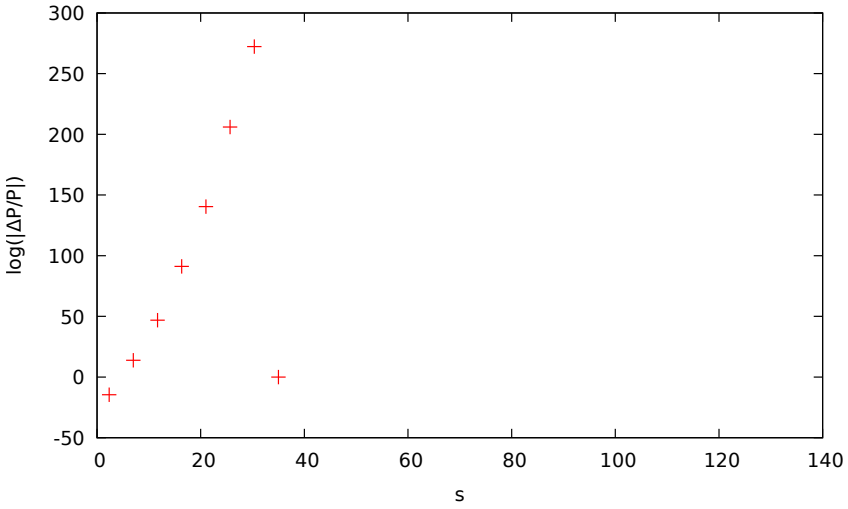
Hermitian: nn= 2100, kk= 32



Hermitian: nn= 2100, kk= 64

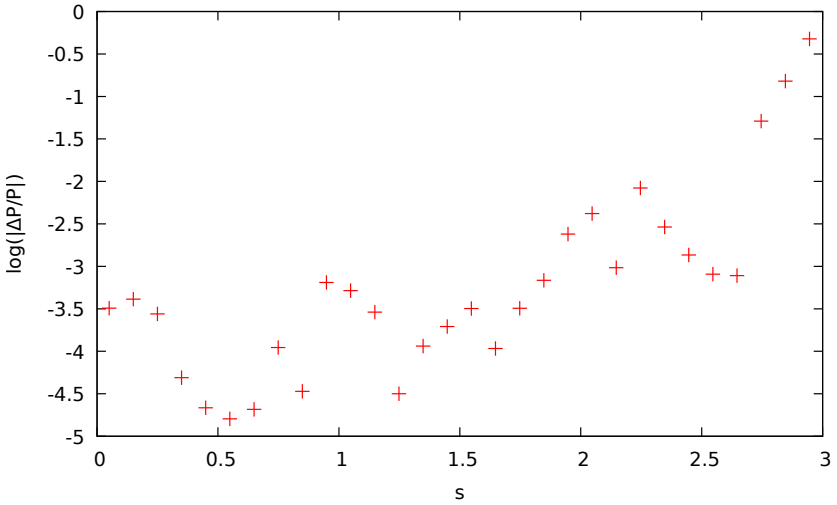


Hermitian: nn= 2100, kk= 130

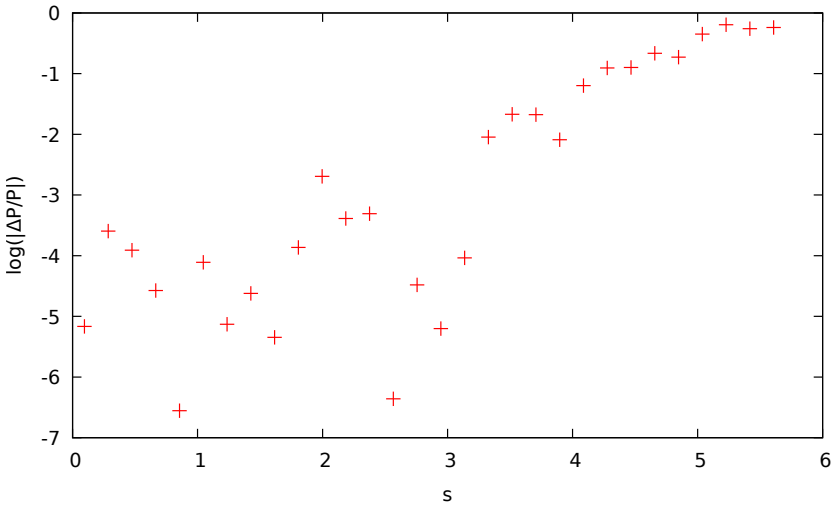


Hermitian: nn= 2100, kk= 2100

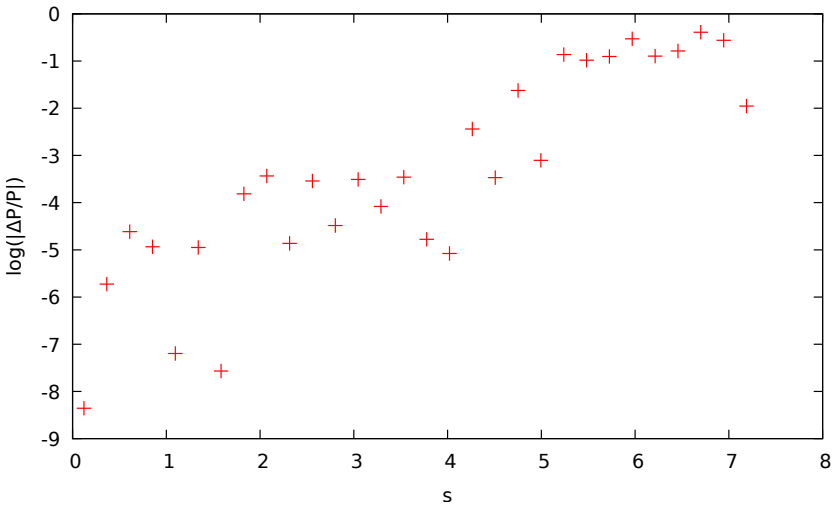
Real Diagonal matrices



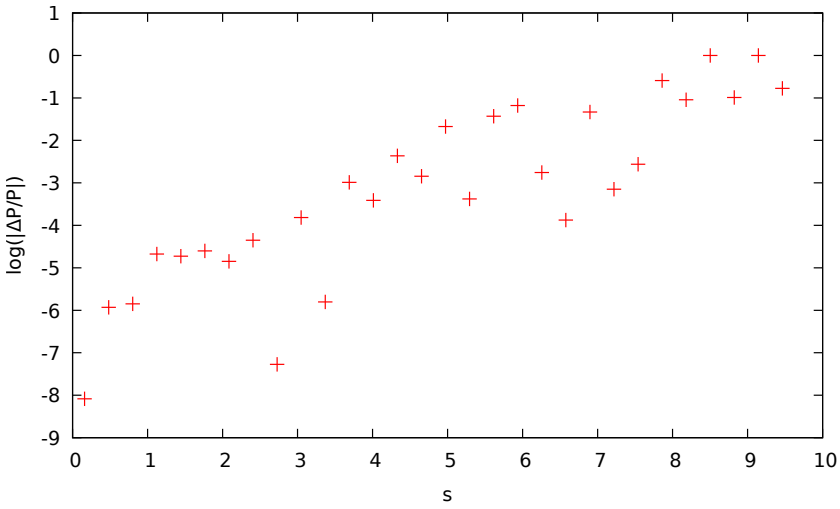
Real Diagonal: nn= 2100, kk= 4



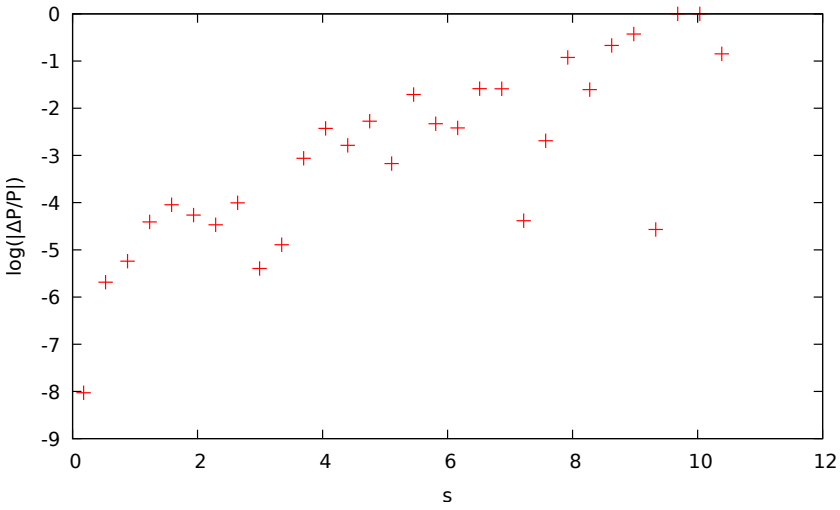
Real Diagonal: nn= 2100, kk= 8



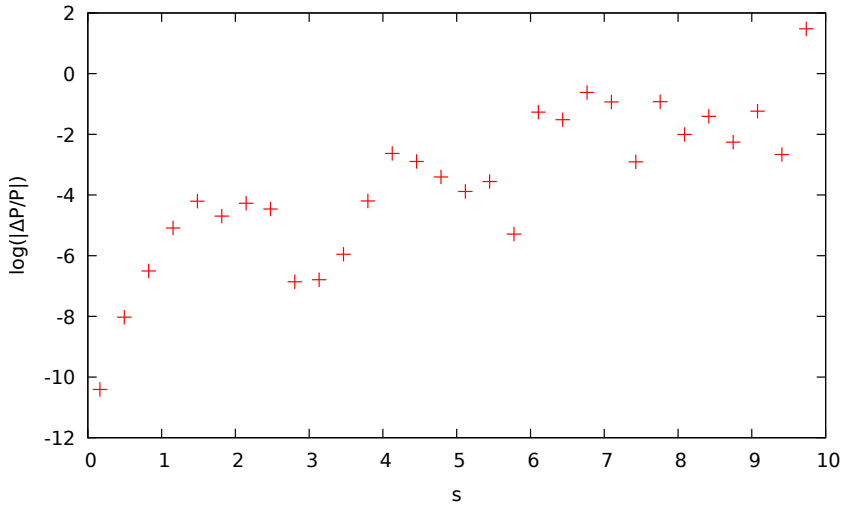
Real Diagonal: nn= 2100, kk= 16



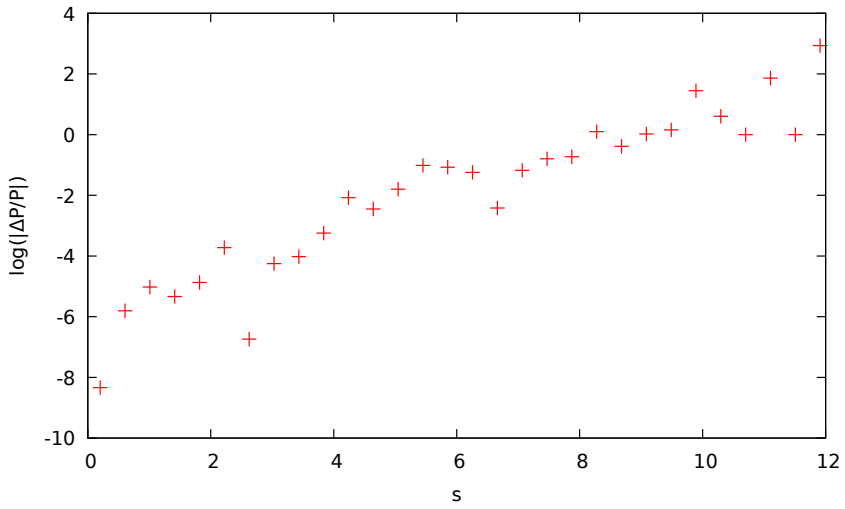
Real Diagonal: nn= 2100, kk= 32



Real Diagonal: nn= 2100, kk= 64



Real Diagonal: nn= 2100, kk= 130



Real Diagonal: nn= 2100, kk= 2100

Gnuplot Codes

Listing 4: Automatic fitting and residuals

```
#FIT-HERMITIAN MATRICES

set term pdf
```

```
set output "templinearfit.pdf"

#name of the output pdf

f(x)= a*x**(alpha)*exp(-b*x**(beta))

a=3.24

alpha= 2.0

b=1.37

beta= 2.0

fit  f(x) "resultstemp" u 1:2 via a,alpha,b,beta

#fitting the data taking the logarithm of both columns
#via a linear function

set xlabel "s"

set ylabel "P(s)"

plot "resultstemp" u 1:2 w p lt rgb "blue" notitle, f(x)
   notitle

#plotting of the data (the log(data)) and
#the interpoling function


#ABSOLUTE RESIDUALS

set output "tempabsres.pdf"

#name of the output pdf

p(x,y)=abs(f(x)-y)

#this function computes absolute residuals

set xlabel "s"

set ylabel "log(|{/Symbol␣D}P/P|)" enhanced
```

```
plot "resultstemp" u 1:(p($1,$2))  w p lt rgb "green" notitle
#the p function accept as imput the two columns of data

#RELATIVE RESIDUALS

set output "temprelres.pdf"

#name of the output pdf

p(x,y)=abs((f(x)-y)/f(x))

#this function computes relative residuals


set xlabel "s"

set ylabel "|{/Symbol␣D}P|" enhanced


plot "resultstemp" u 1:(p($1,$2))  w p lt rgb "brown" notitle
#the p function accept as imput the two columns of data

#LOGARITHM OF RELATIVE RESIDUALS

set output "templogrelres.pdf"

#name of the output pdf

p(x,y)=log(abs((f(x)-y)/f(x)))

#this function is the logarithm of relative residuals

set xlabel "s"

set ylabel "|{/Symbol␣D}P/P|" enhanced


plot "resultstemp" u 1:(p($1,$2)) w p lt rgb "red" notitle
#the p function accept as imput the two columns of data


set term wxt
```

Theory- LU reduction

Since the LU reduction is based on Gauss Eliminations, the computational time scales with n^3 (for each one of n^2 entries, n multplications per a costant are done).

Code Development- LU reduction

Listing 5: Fortran code: LU Reduction.

```
*************************************************************
C         PROGRAM lured
C*************************************************************
C=============================================================
C
C          Purpose
C          =======
C
C       \details \b Purpose:
C       \verbatim
C
C       This program, given the matrix dimension from file "
   MatDimension.txt", initialize a type(dcm),
C       with inside a random hermitian matrix (in range
   [0,1], flat distribution).
C       Via the lapack routine zgetrf, the (double precision)
    the LU decomposition is done.
C       The time needed to do that decomposition is appended
   to a file called "TimesLU.txt".
C       If debug==.true., debugging is done: checking matrix
   dimension, if matrix is
C       still the same, printing type(dcm) variables on file.
    Morever, via the output
C       iinfo, we check if zgetrf worked (and we print the
   output LU matrices on file).
C
C       \endverbatim
C          Authors:
C          ========
C
C         \author Univ. of Padua
C
C         \date 13 November 2018

C
   ============================================================
```

```
module matrices
implicit none
type dcm !defining the type: doublecomplex matrix
integer :: nr !number of rows
integer :: nc !number of columns
!the actual matrix
double complex, dimension (: , :), allocatable :: elem
!eigenvalues
double precision, dimension (:), allocatable :: eval
!trace
double complex :: m_trace
!determinant
double complex :: m_det
end type dcm

contains

subroutine init_hermmat (aa, aacheck, numrow, numcol,debug)
!this subroutine initializes a type dcm,
!given as imput. The matrix becomes Herminian.
!Also the numbers of rows and columns
!are given as imput. The subroutine creates random matrices
!with values in range [0,1].
!It gives a default value 0 to the determinant, the trace
!and to eigenvalues.
real*8 yy, xx
integer ii,jj
type(dcm) :: aa
type(dcm) :: aacheck
integer numrow, numcol
integer :: my_stat
character (256) :: my_msg
logical debug


aa%nr= numrow
aa%nc= numcol
aacheck%nr= numrow
aacheck%nc= numcol

!ERROR HANDLING ALLOCATION VECTORS
allocate (aa%eval(aa%nr),
```

```
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if
allocate (aacheck%eval(aa%nr),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck%eval with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

!ERROR HANDLING ALLOCATION MATRICES
allocate(aa%elem(aa%nr, aa%nc), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aa with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if

allocate(aacheck%elem(aa%nr, aa%nc),
$                        stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed to allocate aacheck with stat = '
$                , my_stat, ' and msg = '//trim(my_msg)
end if


!WARNING IF NOT SQUARED
if(debug .eqv. .true.) then
if(numrow==numcol) then !checking squareness
print*, " "
print*, "INITIALIZATION"
print*, "Okay, squared"
else
print*, "WARNING: NOT SQUARED"
print*, "num rows= ", numrow
print*, "num columns= ", numcol
end if
end if

!INITIALIZATION
```

```
do ii= 1, aa%nr
do jj= ii, aa%nc
call random_number(xx)
call random_number(yy)
aa%elem(ii,jj)=cmplx(xx,yy)!complex number
aa%elem(jj,ii)=conjg(aa%elem(ii,jj))!hermitianity
!in order to be hermitian, complex part on diagonale
!must be null
if(ii==jj) then
aa%elem(ii,jj)=cmplx(xx,0)
end if
aacheck%elem(ii,jj)= aa%elem(ii,jj)
aacheck%elem(jj,ii)= aa%elem(jj,ii)
end do
end do
aa%m_trace=(0d0,0d0)
aa%m_det=(0d0,0d0)
aacheck%m_trace=(0d0,0d0)
aacheck%m_det=(0d0,0d0)

do ii= 1, numrow
aa%eval(ii)=(0d0,0d0)
aacheck%eval(ii)=(0d0,0d0)
end do

end subroutine init_hermmat


subroutine print_matrices (AA, namefile)
! This subroutine prints a type doublecomplex_matrix:
!the numbers of row and columns, the elements (in the proper
   order)
!, the trace and the determinant. This is all printed on a
   file
type(dcm) :: AA
character(:), allocatable :: namefile
integer :: ii,jj
integer :: my_stat
character (256) :: my_msg

open(unit = 40, file = namefile, status = "unknown",
$                       iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣",
```

```
$                        my_stat, "␣msg␣=␣"//trim(my_msg)
end if

write (40,*) "NUMBER␣OF␣ROWS:", AA%nr
write (40,*) "NUMBER␣OF␣COLUMNS:", AA%nc
write (40,*) "ELEMENTS:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) (AA%elem(ii, jj), jj = 1, AA%nc)
end do
write (40,*) "TRACE:", AA%m_trace
write (40,*) "DET:", AA%m_det
write (40,*) "EIGENVALUES:"
do ii=1, AA%nr !do cycle in order to print in the proper
   order
write (40,*) AA%eval(ii)
end do
close(40)

end subroutine print_matrices

subroutine print_vector (vec, nn, namefile)
! This subroutine prints a vector on file,
! given its length nn
character(:), allocatable :: namefile
integer :: ii,nn
integer :: my_stat
character (256) :: my_msg
double precision, dimension(:), allocatable :: vec

open(unit = 80, file = namefile, status = "unknown",
$                        iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open␣failed␣with␣stat␣=␣"
$                          , my_stat, "␣msg␣=␣"//trim(my_msg)
end if

do ii=1, nn !do cycle in order to print
write (80,*) vec(ii)
end do

end subroutine print_vector

end module matrices
```

```
c///////////////////////////////////////////////////////
module normalized_spacings

use matrices

contains

subroutine normspac(aa,kk,normsp,debug)
!computes normalized spacings between eigenvalues
!of a double complex matrix, which is inside type aa
type(dcm) :: aa
integer :: kk
!kk-1 is the number of the spacings to be considered
!in the normalization computation
integer :: ii
integer :: iimax,iimin
double precision, dimension(:), allocatable :: sp
double precision, dimension(:), allocatable :: norm
double precision, dimension(:), allocatable :: normsp
integer :: my_stat
character (256) :: my_msg
character(:), allocatable :: namefile
logical debug
real :: time1,time2,timetot

allocate(sp(aa%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣normsp␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

allocate(norm(aa%nr-1), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, 'Failed␣to␣allocate␣normsp␣with␣stat␣=␣'
$                , my_stat, '␣and␣msg␣=␣'//trim(my_msg)
end if

!spacings
do ii=1, aa%nr-1
sp(ii)=aa%eval(ii+1)-aa%eval(ii)
end do
```

```
if(debug.eqv..TRUE.) then!check after changing
namefile="SPACINGS.txt"
call print_vector(sp, aa%nr-1,namefile)
end if

!NORMALIZATION and NORMALIZED SPACINGS

!if cycle to be sure to have (kk-1) in [1,nn-1]
if((kk-1)>(aa%nr-1)) then

print*, "␣Too␣many␣spaces␣(kk-1)in␣the␣normalization"
print*, "␣Rescaled␣to", (aa%nr-1)
print*, "␣It␣was", kk
kk=aa%nr
else if((kk-1)<1) then
print*, "␣Not␣enough␣spaces␣(kk-1)␣in␣the␣normalization"
print*, "␣Rescaled␣to␣2"
print*, "␣It␣was", kk
kk=2
end if

!if cycle to distinguish between odd case and even case
do ii=1,aa%nr-1
!setting the borders of the interval of normalization
if(mod(kk,2)==1) then
iimax=ii+(kk-1)/2
iimin=ii-(kk-1)/2
else
iimax=ii-1+(kk)/2
iimin=ii-(kk)/2
end if
!boundary conditions

if(debug.eqv..TRUE.) then!check before changing
print*, "BEFORE␣corrections"
print*, "Spacing␣number", ii
print*, "right␣border:", iimax
print*, "left␣border:", iimax
print*, "␣"
end if
if(iimax>aa%nr) then!exceeding at right
iimax=aa%nr
iimin=aa%nr-(kk-1)
else if(iimin<1) then!exceeding at left
iimin=1
```

```
iimax=kk
end if

if(debug.eqv..TRUE.) then!check after changing
print*, "AFTER␣corrections"
print*, "Spacing␣number:", ii
print*, "right␣border:", iimax
print*, "left␣border:", iimax
print*, "␣"
end if

!normalization of ii-spacing
norm(ii)=(aa%eval(iimax)-aa%eval(iimin))/(kk-1)

if(debug.eqv..TRUE.) then!check after changing
namefile="NORMS.txt"
call print_vector(norm, aa%nr-1,namefile)
end if

!normalized ii-spacing
normsp(ii)=sp(ii)/norm(ii)

end do
end subroutine normspac


end module normalized_spacings



c//////////////////////////////////////////////////////

!MODULE DEBUGGING
module debugging

use matrices

interface check
module procedure checkdim,checkmat
end interface

contains
subroutine checkdim(nn,nncheck,debug)!checking dimensions nn
implicit none
integer :: nn, nncheck
```

```
logical :: debug

if(debug.eqv..TRUE.) then

if(nn>10000) then !is the dim too large?
print*, "WARNING: too large dimension:"
print*, nn
else if(nn<1) then !is the dim minor than 1?
print*, "WARNING: dimension minor than 1"
else if((nn-nncheck)>0.5) then !is the dim wrong?
print*, "the dimension is wrong, it should be:"
$                                , nncheck
print*, "but is", nn
print*, " "
else
print*, "okay: right dimensions matrices",nn
end if
end if
end subroutine checkdim

subroutine checkmat(nn,mm,mmcheck,debug)
implicit none
!checking if the input matrixes are equal
type(dcm) :: mm
type(dcm) :: mmcheck
integer :: tt,ss,nn
integer*4 accum
logical debug

if(debug.eqv..TRUE.) then
accum=0
do tt=1,nn
do ss=1,nn
if(abs(mm%elem(tt,ss)-mmcheck%elem(tt,ss))
$                        /abs(mmcheck%elem(tt,ss))> 10E-10)
   then
accum=accum+1
end if
end do
end do

if(accum>0) then
print*, "The two matrices have"
$                        , accum, "different entries"
else
```

```
print*, "Same matrices"
end if

end if
end subroutine checkmat

end module debugging



!PROGRAM
program lured

use debugging
use matrices
use normalized_spacings

type(dcm) :: aa
type(dcm) :: aacheck
type(dcm) :: lu
type(dcm) :: lucheck

integer  ii, jj, nn, nncheck
logical :: debug
character*1 :: choice
integer :: my_stat
character (256) :: my_msg
integer, allocatable :: ipiv(:)
integer :: iinfo

complex*16, allocatable:: work(:)
integer lwork
double precision, allocatable:: rwork(:)
integer iinfodiag
character(:), allocatable :: lufile

choice="n" !Initialization of choice variable
!if it is "X", it asks the debug
!otherwise the programmer can choose between
!doing the debug or not writing "y" or "n"

!Do you want to debug?
if(choice=="X") then
print *, "Do you want to debug?"
print *, "y for yes, n for no"
```

```
read*, choice
end if

if(choice=="y") then
debug=.TRUE.
else if(choice=="n") then
debug=.FALSE.
else
print*, "Not understood"
stop
end if

!HERE THE MATRIX DIMENSION IS TAKEN AS IMPUT FROM
!FILE "MatDimension.txt"
!AND THE PARAMETER REGARDING NORMALIZED SPACINGS
!BETWEEN EIGENVALUES

open(unit = 30, file = "MatDimension.txt",
$       status = "unknown",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open -MatDimension- failed with stat = "
$                         , my_stat, " msg = "//trim(my_msg)
end if

read(30,*) nn
nncheck=nn

!OPEN FILE WHICH WILL CONTAIN THE TIMES NEEDED TO LU

open(unit = 10, file = "TimesLU.txt",
$       status = "unknown", access="append",
$  iostat=my_stat, iomsg=my_msg)

if(my_stat /= 0) then
print*, "Open -TimesLU- failed with stat = "
$                         , my_stat, " msg = "//trim(my_msg)
end if


!INITIALIZATION aa and aacheck:
!random numbers in range [0,1]
!hermitian matrices
```

```
call init_hermmat (aa, aacheck, nn, nn,debug)

!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "INITIALIZATION␣DEBUG"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if

!INITIALIZATION lu:
!random numbers in range [0,1]
!hermitian matrices
!later they will be overwritten
!lu used for lu decomposition
!lucheck useless
call init_hermmat (lu, lucheck, nn, nn, debug)


!LU REDUCTION
allocate(ipiv(nn), stat=my_stat, errmsg=my_msg)
!error handling allocation
if(my_stat /= 0) then
print*, "Failed␣to␣allocate␣ipiv␣with␣stat␣=␣"
$                , my_stat, "␣and␣msg␣=␣"//trim(my_msg)
end if
ipiv = 0

!the matrix lu%elem is equal to aa%elem
do ii=1,nn
do jj=1,nn
lu%elem(ii,jj)=aa%elem(ii,jj)
end do
end do


call cpu_time(time1)

!call lapack function for LU reduction
call zgetrf(nn, nn, lu%elem, nn , ipiv, iinfo)
!lu%elem now contains the matrices L and U
!(unit diagonal entries of L are discarded)

call cpu_time(time2)
```

```
timetot=time2-time1

write(10,"(I4,4X,E16.9)",iostat=my_stat, iomsg=my_msg)
$               nn, timetot

!DEBUG
if(debug.eqv..TRUE.) then
print*, "␣"
print*, "LU␣REDUCTION"
print*, "␣"
end if
if(debug.eqv..TRUE.) then!checkdim
call check(nn,nncheck,debug)
end if

if(debug.eqv..TRUE.) then!checkmat
print*, "␣"
print*,"aa␣and␣aacheck"
call check(nn,aa,aacheck,debug)
end if

if(debug.eqv..TRUE.) then!check LU
if (iinfo==0) then
print*, "␣"
print*, "Successful␣LU"
else if (iinfo < 0) then
print*, "␣"
print*, "the", iinfo, "-th␣argument
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣of␣ipiv␣had␣an␣illegal␣value"

else
print*, "␣"
print*,  "the", iinfo, "-th␣diagonal␣entry
$␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣of␣U␣had␣an␣illegal␣value"
end if
end if

if(debug.eqv..TRUE.) then!print LU
lufile="LUMatrix.txt"
call print_matrices (lu, lufile)
end if

stop
end program lured
```

Listing 6: Scripting: fitting cpu_time vs nn

```
import numpy as np
import sys
import os
import subprocess



nmin=1000
#lower bound dimension matrix
nmax=4000+250
#upper bound dimension matrix
stepp=250
#step of the increasing dimension

################################################
#cycle to re-iterate the launch of the program
#that performs the LU reduction of random matrix
#with values from 0 to 1 uniformely distributed,
#computing the cpu_time requested. The cycle is from
#nmin to nmax with step
#The results are saved in the file:
#"TimesLU.txt"

################################################

for ii in range(nmin,nmax,stepp):
f = open("MatDimension.txt","w")
#opening the file with the dimension of the matrix
f.write(str(ii))
f.close()
#write the dimension on the opened file
subprocess.call("./a.out")
#execute the fortran program



################################################
#AUTOMATIC FITTING of the set of data.

#For every "Results-%d" file, the list of commands within "
   fitting" executes:
#linear fit (taking the log of the cubic law), inside "
   linearfit-%d.pdf"
#absolute residuals (\Delta f), inside "absres-%d.pdf"
```

```
#relative residuals (\Delta f/f), inside "relres-%d.pdf"
#logarithm of relative residuals (log(\Delta f/f(), inside "
   logrelres-%d.pdf"
#the x range is from nmin to nmax (to be inserted manually
   inside "rawplot")
#On the x-axis there is log(N), with N the dimension of the
   matrix
#The data of the fit are contained inside "parametersfit-%d."

#Logic of the script: the data contained in the considered "
   Results-%d" file
#are copied inside a temporary file "resultstemp". On that
   file are done the
#plot and the fit listed above. The results are stored in
   file with a
#temporary name, which are changed in the names listed above,
    which contain
#the number of the "Results-%d" file. This applies also for
   the data on the fit.


#This procedure is valid for a general number of "Results-%d"
    file,
#contained in the same directory of the script.
#This number is saved in the file called "numresults.txt",
#from which it is read.
################################################

res = open("TimesLU.txt","r")
reslines = res.readlines()
#reading data from result file number ii

temp = open("resultstemp","w")
for jj in reslines:
temp.write(jj)
#copying data in temporary file
res.close()
temp.close()
#closing files

os.system("gnuplot␣fitting")
#linear fit of data, absolute residuals,
#relative residuals, log of realtive residuals

newnamelinear= "LUfit.pdf"
```

```
os.rename("templinearfit.pdf",newnamelinear)
#renaming the file of the linear fit

newnameabs= "LUabsres.pdf"
os.rename("tempabsres.pdf",newnameabs)
#renaming the file of the absolute residuals

newnamerel= "LUrelres.pdf"
os.rename("temprelres.pdf",newnamerel)
#renaming the file of the relative residuals

newnamelog= "LUlogrelres.pdf"
os.rename("templogrelres.pdf",newnamelog)
#renaming the file of the log of the relative residuals

os.remove("resultstemp")
#removing the temporary file of results

fitlog = open("fit.log","r")
fitlines = fitlog.readlines()
#reading parameters of the fit from "fit.log"

fitlogd = open("LUparametersfit.txt","w")
for jj in fitlines:
fitlogd.write(jj)
#copying parameters on "parametersfit-%d"
fitlog.close()
fitlogd.close()
#closing files

os.remove("fit.log")
#removing "fit.log"


exit()
```

Results -LU reduction

Fit and residuals

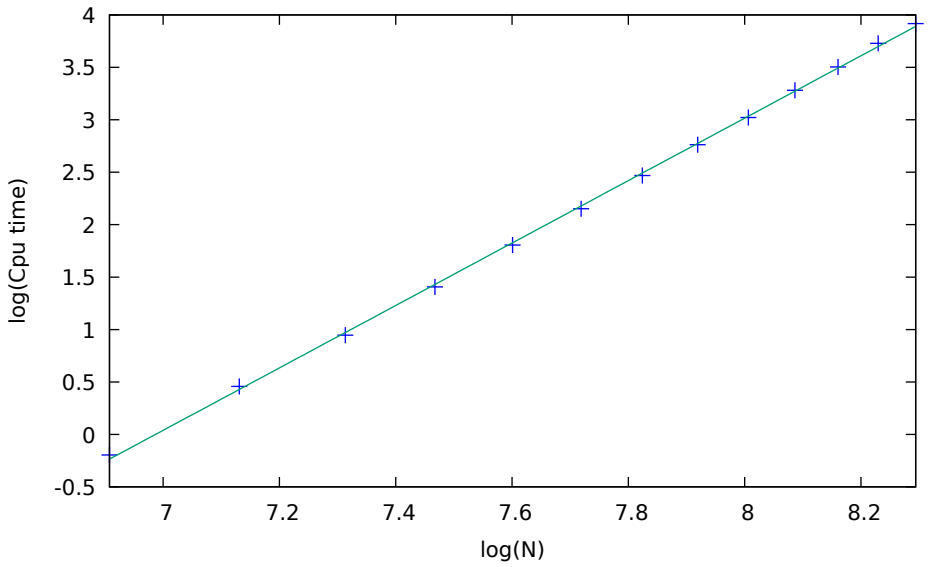


Figure 11: Fit of log(t) vs log(dimension)

Obtained parameters:

a	-20.8 ± 0.1
b	2.97 ± 0.02

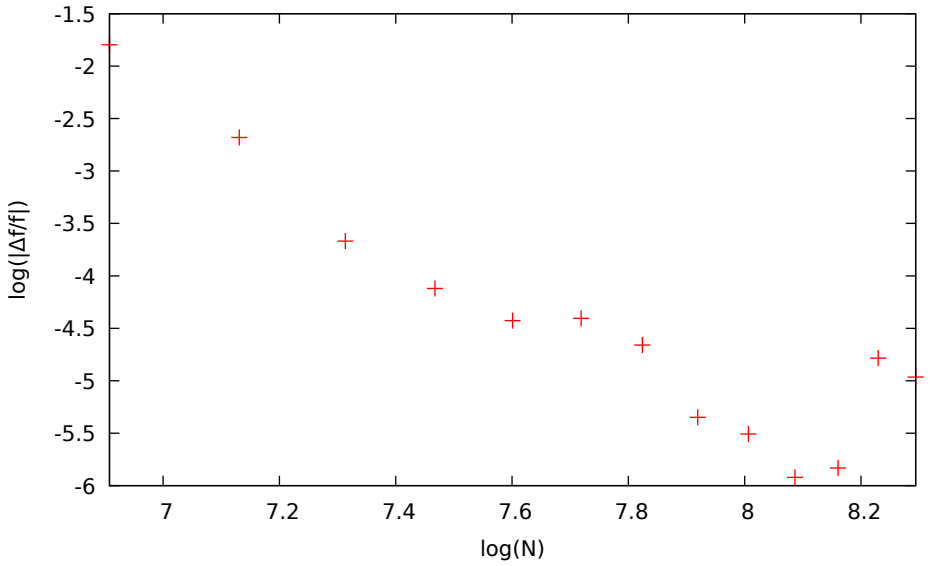


Figure 12: Log of relative residuals

Self-Evaluation- LU reduction

The obtained slope is very close to 3, as expected from the theory. The logarithms of the residuals are very low. Hence the LAPACK subroutine $zgetrf$ requests a computational time $\sim n^3$ and the fit is well done.