FVN Documentation

William Daniau

September 8, 2011

Contents

1 Whatis fvn,license

1.1 Whatis fvn

fvn is a Fortran95 mathematical library with several modules. It provides various usefull subroutine covering linear algebra, numerical integration, least square polynomial, spline interpolation, zero finding, special functions etc.

Most of the work for linear algebra is done by interfacing Lapack http://www.netlib.org/lapack which means that Lapack and Blas http://www.netlib.org/blas must be available on your system for linking fvn. If you use an AMD microprocessor, the good idea is to use ACML (AMD Core Math Library http://developer.amd.com/acml.jsp which contains an optimized Blas/Lapack.

fvn include some integrated libraries : integration tasks uses a slightly modified version of Quadpack http://www.netlib.org/quadpack, the fnlib library http://www.netlib.org/fn is used for special functions and sparse system resolution uses SuiteSparse http://www.cise.ufl.edu/research/sparse/SuiteSparse/.

This library has been initially written for the use of the "Acoustic and microsonic" group leaded by Sylvain Ballandras in the Time and Frequency Department of institute Femto-ST http://www.femto-st.fr/.

1.2 License

Your use or distribution of fvn or any modified version of fvn implies that you agree to this License.

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Permission is hereby granted to use or copy this program under the terms of the GNU LGPL, provided that the Copyright, this License, and the Availability of the original version is retained on all copies. User documentation of any code that uses this code or any modified version of this code must cite the Copyright, this License, the Availability note, and "Used by permission." Permission to modify the code and to distribute modified code is granted, provided the Copyright, this License, and the Availability note are retained, and a notice that the code was modified is included.

Authors and Contributors

- William Daniau (william.daniau@femto-st.fr)
- Sylvain Ballandras (sylvain.ballandras@femto-st.fr)
- Christian Waterkeyn (christian.waterkeyn@epcos.com)

2 Naming scheme and convention

The naming scheme of the routines is as follow :

```
fvn_*_name()
```

where * can be s,d,c or z.

- s is for single precision real (real,real*4,real(4),real(kind=4))
- d for double precision real (double precision,real*8,real(8),real(kind=8))
- c for single precision complex (complex,complex*8,complex(4),complex(kind=4))
- z for double precision complex (double complex,complex*16,complex(8),complex(kind=8))

In the following description of subroutines parameters, input parameters are followed by (in), output parameters by (out) and parameters which are used as input and modified by the subroutine are followed by (inout).

For each routine, there is a generic interface (simply remove `_*` in the name), so using the specific routine is not mandatory.

There's a general module called "`fvn`" that include all fvn submodules. So whatever part of the library is used in the program a "`use fvn`" will be sufficient instead of specifying the specific module.

3 kind specification

In `fvn` the following definitions are made (this is done in module `fvn_common`) to ease portability:

```
integer, parameter :: ip_kind = kind(1)
integer, parameter :: sp_kind = kind(1.0E0)
integer, parameter :: dp_kind = kind(1.0D0)
```

Although not mandatory, it is a good idea to use these definitions when programming with fvn, that is for example :

```
real(kind=sp_kind) :: x
real(kind=dp_kind) :: y
complex(kind=dp_kind) :: z
```

instead of

```
real :: x
double precision :: y
complex(8) :: z
```

4 Linear algebra

The linear algebra routines of fvn are an interface to lapack, which make it easier to use.

4.1 Matrix inversion

```
Module : use fvn_linear
call fvn_matinv(d,a,inva,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a real or complex matrix. It will remain untouched.
- inva (out) is a real or complex matrix which contain the inverse of a at the end of the routine
- status (out) is an optional integer equal to zero if something went wrong

Example

```
program inv
 use fvn_linear
 implicit none

 real(8),dimension(3,3) :: a,inva

 call random_number(a)
 a=a*100

 call fvn_matinv(3,a,inva)
 write (*,*) a
 write (*,*)
```

```
 write (*,*) inva
 write (*,*)
 write (*,*) matmul(a,inva)
end program
```

4.2 Matrix determinants

```
Module : use fvn_linear
det=fvn_det(d,a,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a real or complex matrix. It will remain untouched.
- status (out) is an optional integer equal to zero if something went wrong

Example

```
program det
 use fvn_linear
 implicit none

 real(8),dimension(3,3) :: a
 real(8) :: deta
 integer :: status

 call random_number(a)
 a=a*100

 deta=fvn_det(3,a,status)
 write (*,*) a
 write (*,*)
 write (*,*) "Det = ",deta
end program
```

4.3 Matrix condition

```
Module : use fvn_linear
call fvn_matcon(d,a,rcond,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a real or complex matrix. It will remain untouched.
- rcond (out) is a real of same kind as matrix a, it will contain the reciprocal condition number of the matrix
- status (out) is an optional integer equal to zero if something went wrong

The reciprocal condition number is evaluated using the 1-norm and is define as in equation 1

$$R = \frac{1}{norm(A) * norm(invA)} \tag{1}$$

The 1-norm itself is defined as the maximum value of the columns absolute values (modulus for complex) sum as in equation 2

$$L1 = max_j(\sum_i \mid A(i,j) \mid) \tag{2}$$

Example

```
program cond
 use fvn_linear
 implicit none

 real(8),dimension(3,3) :: a
 real(8) :: rcond
 integer :: status

 call random_number(a)
 a=a*100

 call fvn_d_matcon(3,a,rcond,status)
 write (*,*) a
 write (*,*)
 write (*,*) "Cond = ",rcond
end program
```

4.4 Eigenvalues/Eigenvectors

```
Module : use fvn_linear
call fvn_matev(d,a,evala,eveca,status,sortval)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a real or complex matrix. It will remain untouched.
- evala (out) is a complex array of same kind as a. It contains the eigenvalues of matrix a
- eveca (out) is a complex matrix of same kind as a. Its columns are the eigenvectors of matrix a : eveca(:,j)=jth eigenvector associated with eigenvalue evala(j).
- status (out) is an optional integer equal to zero if something went wrong
- sortval (in) is an optional logical, if it is true the eigenvalues (and eigenvectors) are sorted in decreasing order of eigenvalues's modulus.

Example

```
program eigen
 use fvn_linear
 implicit none

 real(8),dimension(3,3) :: a
 complex(8),dimension(3) :: evala
 complex(8),dimension(3,3) :: eveca
 integer :: status,i,j

 call random_number(a)
 a=a*100

 call fvn_matev(3,a,evala,eveca,status)
 write (*,*) a
 write (*,*)
 do i=1,3
```

```
    write(*,*) "Eigenvalue ",i,evala(i)
    write(*,*) "Associated Eigenvector :"
    do j=1,3
        write(*,*) eveca(j,i)
    end do
    write(*,*)
 end do

end program
```

4.5 Sparse matrix

By interfacing Tim Davis's SuiteSparse from university of Florida http://www.cise.ufl.edu/research/sparse/SuiteSparse/ which is a reference for this kind of problems, fvn provides simple subroutines for solving linear sparse systems and calculating determinants.

4.5.1 Sparse solving

The provided routines solves the equation $Ax = B$ where A is sparse and given in its triplet form.

Note that interface has changed from previous versions, splitting complex inputs into real and imaginary part and using 0-based indices, to fit umfpack routines inputs and avoid duplicating datas inside the routines.

```
Module : fvn_sparse
call fvn_sparse_solve(n,nz,T,Ti,Tj,B,x,status,det)
call fvn_sparse_solve(n,nz,Tx,Tz,Ti,Tj,Bx,Bz,x,status,det)
```

- For this family of subroutines the two letters (zl,zi,dl,di) of the specific interface name decribe the arguments's type. z is for complex(8), d for real(8), l for integer(8) and i for integer(4)
- `n` (in) is an integer equal to the matrix rank
- `nz` (in) is an integer equal to the number of non-zero elements
- `T(nz) or Tx(nz),Tz(nz)` (in) is a real array (or two real arrays for real/imaginary in case of a complex system) containing the non-zero elements
- `Ti(nz),Tj(nz)` (in) are the indexes of the corresponding element of `T` in the original matrix, this has to be 0-based as in C.
- `B(n) or Bx(n),Bz(n)` (in) is a real array or two real arrays for real/imaginary in case of a complex system) containing the second member of the equation.
- `x(n)` (out) is a complex/real array containing the solution
- `status` (out) is an integer which contain non-zero is something went wrong
- `det` (out), is an optional real(8) array of dimension 2 for dl and di specific interface (real systems) and dimension 3 for zl and zi interface (complex systems)

Example

```
program test_sparse

 use fvn_sparse
 implicit none
```

```
 integer(8), parameter :: nz=12
 integer(8), parameter :: n=5
 complex(8),dimension(nz) :: A
 real(8), dimension(nz) :: Ax,Az
 integer(8),dimension(nz) :: Ti,Tj
 complex(8),dimension(n) :: B,x
 real(8), dimension(n) :: Bx,Bz
 integer(8) :: status

 A = (/ (2.,0.),(3.,0.),(3.,0.),(-1.,0.),(4.,0.),(4.,0.),(-3.,0.),&
          (1.,0.),(2.,0.),(2.,0.),(6.,0.),(1.,0.) /)
 B = (/ (8.,0.), (45.,0.), (-3.,0.), (3.,0.), (19.,0.)/)
 Ti = (/ 1,2,1,3,5,2,3,4,5,3,2,5 /)
 Tj = (/ 1,1,2,2,2,3,3,3,3,4,5,5 /)
 Ax=real(A)
 Az=aimag(A)
 Bx=real(B)
 Bz=aimag(B)

 ! 1-based to 0-based translation
 Ti=Ti-1
 Tj=Tj-1

 !specific routine that will be used here
 !call fvn_zl_sparse_solve(n,nz,Ax,Az,Ti,Tj,Bx,Bz,x,status)
 call fvn_sparse_solve(n,nz,Ax,Az,Ti,Tj,Bx,Bz,x,status)
 write(*,*) x

end program


program test_sparse

use fvn_sparse
implicit none

integer(4), parameter :: nz=12
integer(4), parameter :: n=5
real(8),dimension(nz) :: A
integer(4),dimension(nz) :: Ti,Tj
real(8),dimension(n) :: B,x
integer(4) :: status

A = (/ 2.,3.,3.,-1.,4.,4.,-3.,1.,2.,2.,6.,1. /)
B = (/ 8., 45., -3., 3., 19./)
Ti = (/ 1,2,1,3,5,2,3,4,5,3,2,5 /)
Tj = (/ 1,1,2,2,2,3,3,3,3,4,5,5 /)

! 1-based to 0-based translation
Ti=Ti-1
Tj=Tj-1

!specific routine that will be used here
!call fvn_di_sparse_solve(n,nz,A,Ti,Tj,B,x,status)
```

```
call fvn_sparse_solve(n,nz,A,Ti,Tj,B,x,status)
write(*,*) x

end program
```

If optional parameter `det` is given, the routine will also calculates the matrix determinant and returns it on a mantissa + exponent form, that is the actual determinant will be $det(1).10^{det(2)}$ for real problems and $(det(1) + i.det(2)).10^{det(3)}$ for complex problems. This is given in this form as the determinant can be considerably higher/lower than the biggest/lowest usable double precision real. There's an example of how to use this in following paragraph.

4.5.2 Sparse determinant

The provided subroutines calculates the determinant of a matrix given in its triplet form.

```
Module : fvn_sparse
call fvn_sparse_det(n,nz,T,Ti,Tj,det,status)
call fvn_sparse_det(n,nz,Tx,Tz,Ti,Tj,det,status)
```

- For this family of subroutines the two letters (zl,zi,dl,di) of the specific interface name decribe the arguments's type. z is for complex(8), d for real(8), l for integer(8) and i for integer(4)
- `n` (in) is an integer equal to the matrix rank
- `nz` (in) is an integer equal to the number of non-zero elements
- `T(nz) or Tx(nz),Tz(nz)` (in) is a real array (or two real arrays for real/imaginary in case of a complex system) containing the non-zero elements
- `Ti(nz),Tj(nz)` (in) are the indexes of the corresponding element of `T` in the original matrix, it has to be 0-based as in C.
- `det` (out), a real(8) array of dimension 2 for dl and di specific interface (real systems) and dimension 3 for zl and zi interface (complex systems)
- `status` (out) is an integer which contain non-zero is something went wrong

The matrix determinant is returned on a mantissa + exponent form, that is the actual determinant will be $det(1).10^{det(2)}$ for real problems and $(det(1) + i.det(2)).10^{det(3)}$ for complex problems. This is given in this form as the determinant can be considerably higher/lower than the biggest/lowest usable double precision real.

Here are the possibly returned errors in `status` parameter :

- `0` : no errors
- `-1`: out of memory
- `1` : singular matrix
- `2` : determinant underflow, the "natural" form of the determinant $det(1).10^{det(2)}$ or $(det(1) + i.det(2)).10^{det(3)}$ will underflow.
- `3` : determinant overflow, the "natural" form of the determinant (as above) will overflow

And here's an example using this

```
program test_sparse
use fvn
implicit none
integer(kind=sp_kind), parameter :: nz=12
integer(kind=sp_kind), parameter :: n=5
complex(kind=dp_kind),dimension(nz) :: A
real(kind=dp_kind), dimension(nz) :: Ax,Az
complex(kind=dp_kind),dimension(n,n) :: As
integer(kind=sp_kind),dimension(nz) :: Ti,Tj
complex(kind=dp_kind),dimension(n) :: B,x
real(kind=dp_kind), dimension(n) :: Bx,Bz
integer(kind=sp_kind) :: status,i
real(kind=dp_kind),dimension(3) :: det
character(len=80) :: fmcmplx

fmcmplx='(5("(",f8.5,",",f8.5,")  "))'

! Description of the matrix in triplet form
A = (/ (2.,-1.),(3.,2.),(3.,1.),(-1.,5.),(4.,-7.),(4.,0.),(-3.,-4.),(1.,3.),(2.,0.),(2.,-2.),(6.,
B = (/ (8.,3.), (45.,1.), (-3.,-2.), (3.,0.), (19.,2.) /)
Ti = (/ 1,2,1,3,5,2,3,4,5,3,2,5 /)
Tj = (/ 1,1,2,2,2,3,3,3,3,4,5,5 /)

Ax=real(A)
Az=aimag(A)
Bx=real(B)
Bz=aimag(B)

! Reconstruction of the matrix in standard form
As=0.
do i=1,nz
    As(Ti(i),Tj(i))=A(i)
end do

write(*,*) "Matrix in standard representation :"
do i=1,5
    write(*,fmcmplx) As(i,:)
end do
write(*,*)
write(*,*) "Standard determinant : ",fvn_det(5,As)
write(*,*)
write(*,*) "Right hand side :"
write(*,fmcmplx) B

! can use either specific interface, fvn_zi_sparse_det
! either generic one fvn_sparse_det
call fvn_zi_sparse_det(n,nz,Ax,Az,Ti,Tj,det,status)
write(*,*)
write(*,*) "Sparse Det = ",cmplx(det(1),det(2),kind=dp_kind)*10**det(3)
! can use either specific interface fvn_zi_sparse_solve
! either generic one fvn_sparse_solve
! parameter det is optional
call fvn_zi_sparse_solve(n,nz,Ax,Az,Ti,Tj,Bx,Bz,x,status,det)
write(*,*)
```

```
write(*,*) "Sparse Det as solve option= ",cmplx(det(1),det(2),kind=dp_kind)*10**det(3)
write(*,*)
write(*,*) "Solution :"
write(*,fmcmplx) x
write(*,*)
write(*,*) "Product matrix Solution :"
write(*,fmcmplx) matmul(As,x)
end program
```

4.6 Identity matrix

```
Module : use fvn_linear
I=fvn_*_ident(n)     (*=s,d,c,z)
```

- n (in) is an integer equal to the matrix rank

This function return the identity matrix of rank n, in the specified type. No generic interface for this one.

4.7 Operators

fvn defines some linear operators similar to those defined in IMSL®(http://www.vni.com/products/imsl/), that can be used for matrix operations.

```
Module : use fvn_linear
```

4.7.1 Unary operators

.i. This operator gives the inverse matrix of the argument which must be a square matrix. The status of the operation can be found in the module variable fvn_status (fourth parameter fvn_matinv).

$$\text{b=.i.a} \iff b = a^{-1}$$

.t. This operator gives the transpose matrix of the argument.

$$\text{b=.t.a} \iff b = {}^t a$$

.h. This operator gives the conjugate transpose matrix of the argument (also called Hermitian transpose or adjoint matrix).

$$\text{b=.h.a} \iff b = a^* = \overline{({}^t a)} = {}^t \overline{a}$$

4.7.2 Binary operators

.x. This operator gives the matrix product of the two operands.

$$\text{c=a.x.b} \iff c = ab$$

.ix. This operator gives the matrix product of the inverse of the first operand and the second one. The status of the inversion can be found in the module variable fvn_status (fourth parameter fvn_matinv).

$$\text{c=a.ix.b} \iff c = a^{-1} b$$

.xi. This operator gives the matrix product of the first operand and the inverse of the second one. The status of the inversion can be found in the module variable fvn_status (fourth parameter fvn_matinv).

$$\text{c=a.xi.b} \iff c = ab^{-1}$$

.tx. This operator gives the matrix product of the transpose matrix of the first operand and the second one.

$$\text{c=a.tx.b} \iff c = {}^t ab$$

.xt. This operator gives the matrix product of the first operand and the transpose matrix of the second one.

$$\text{c=a.xt.b} \iff c = a {}^t b$$

.hx. This operator gives the matrix product of the conjugate transpose matrix of the first operand and the second one.

$$\text{c=a.hx.b} \iff c = a^* b$$

.xh. This operator gives the matrix product of thee first operand and the conjugate transpose matrix of the second one.

$$\text{c=a.xh.b} \iff c = ab^*$$

5 Interpolation

5.1 Quadratic Interpolation

fvn provide function for interpolating values of a tabulated function of 1, 2 or 3 variables, for both single and double precision.

5.1.1 One variable function

```
Module : use fvn_interpol
value=fvn_quad_interpol(x,n,xdata,ydata)
```

- x is the real where we want to evaluate the function
- n is the number of tabulated values
- xdata(n) contains the tabulated coordinates
- ydata(n) contains the tabulated function values ydata(i)=y(xdata(i))

xdata must be strictly increasingly ordered. x must be within the range of xdata to actually perform an interpolation, otherwise the resulting value is an extrapolation

Example

```
program inter1d

use fvn_interpol
implicit none

integer(kind=4),parameter :: ndata=33
integer(kind=4) :: i,nout
real(kind=8) :: f,fdata(ndata),h,pi,q,sin,x,xdata(ndata)
real(kind=8) ::tv
```

```
intrinsic sin

f(x)=sin(x)

xdata(1)=0.
fdata(1)=f(xdata(1))
h=1./32.
do i=2,ndata
      xdata(i)=xdata(i-1)+h
      fdata(i)=f(xdata(i))
end do
call random_seed()
call random_number(x)

q=fvn_d_quad_interpol(x,ndata,xdata,fdata)

tv=f(x)
write(*,*) "x ",x
write(*,*) "Calculated (real) value :",tv
write(*,*) "fvn interpolation :",q
write(*,*) "Relative fvn error :",abs((q-tv)/tv)

end program
```

5.1.2 Two variables function

```
Module : use fvn_interpol
value=fvn_quad_2d_interpol(x,y,nx,xdata,ny,ydata,zdata)
```

- x,y are the real coordinates where we want to evaluate the function
- nx is the number of tabulated values along x axis
- xdata(nx) contains the tabulated x
- ny is the number of tabulated values along y axis
- ydata(ny) contains the tabulated y
- zdata(nx,ny) contains the tabulated function values zdata(i,j)=z(xdata(i),ydata(j))

xdata and ydata must be strictly increasingly ordered. (x,y) must be within the range of xdata and ydata to actually perform an interpolation, otherwise the resulting value is an extrapolation

Example

```
program inter2d
use fvn_interpol
implicit none

integer(kind=4),parameter  :: nx=21,ny=42
integer(kind=4) :: i,j
real(kind=8) :: f,fdata(nx,ny),dble,pi,q,sin,x,xdata(nx),y,ydata(ny)
real(kind=8) :: tv
```

```
intrinsic dble,sin

f(x,y)=sin(x+2.*y)
do i=1,nx
      xdata(i)=dble(i-1)/dble(nx-1)
end do
do i=1,ny
      ydata(i)=dble(i-1)/dble(ny-1)
end do
do i=1,nx
      do j=1,ny
            fdata(i,j)=f(xdata(i),ydata(j))
      end do
end do
call random_seed()
call random_number(x)
call random_number(y)

q=fvn_d_quad_2d_interpol(x,y,nx,xdata,ny,ydata,fdata)
tv=f(x,y)

write(*,*) "x y",x,y
write(*,*) "Calculated (real) value :",tv
write(*,*) "fvn interpolation :",q
write(*,*) "Relative fvn error :",abs((q-tv)/tv)

end program
```

5.1.3 Three variables function

```
 Module : use fvn_interpol
value=fvn_quad_3d_interpol(x,y,z,nx,xdata,ny,ydata,nz,zdata,tdata)
```

- x,y,z are the real coordinates where we want to evaluate the function
- nx is the number of tabulated values along x axis
- xdata(nx) contains the tabulated x
- ny is the number of tabulated values along y axis
- ydata(ny) contains the tabulated y
- nz is the number of tabulated values along z axis
- zdata(ny) contains the tabulated z
- tdata(nx,ny,nz) contains the tabulated function values tdata(i,j,k)=t(xdata(i),ydata(j),zdata(k))

xdata, ydata and zdata must be strictly increasingly ordered. (x,y,z) must be within the range of xdata and ydata to actually perform an interpolation, otherwise the resulting value is an extrapolation

Example

```
program inter3d
use fvn_interpol

implicit none

integer(kind=4),parameter  :: nx=21,ny=42,nz=18
integer(kind=4) :: i,j,k
real(kind=8) :: f,fdata(nx,ny,nz),dble,pi,q,sin,x,xdata(nx),y,ydata(ny),z,zdata(nz)
real(kind=8) :: tv

intrinsic dble,sin

f(x,y,z)=sin(x+2.*y+3.*z)
do i=1,nx
      xdata(i)=2.*(dble(i-1)/dble(nx-1))
end do
do i=1,ny
      ydata(i)=2.*(dble(i-1)/dble(ny-1))
end do
do i=1,nz
      zdata(i)=2.*(dble(i-1)/dble(nz-1))
end do
do i=1,nx
      do j=1,ny
            do k=1,nz
                  fdata(i,j,k)=f(xdata(i),ydata(j),zdata(k))
            end do
      end do
end do
call random_seed()
call random_number(x)
call random_number(y)
call random_number(z)

q=fvn_d_quad_3d_interpol(x,y,z,nx,xdata,ny,ydata,nz,zdata,fdata)
tv=f(x,y,z)

write(*,*) "x y z",x,y,z
write(*,*) "Calculated (real) value :",tv
write(*,*) "fvn interpolation :",q
write(*,*) "Relative fvn error :",abs((q-tv)/tv)

end program
```

5.1.4 Utility procedure

fvn provides a simple utility procedure to locate the interval in which a value is located in an increasingly ordered array.

```
Module : use fvn_interpol
call fvn_find_interval(x,i,xdata,n)
```

- x (in) the real value to locate
- i (out) the resulting indice
- xdata(n) (in) increasingly ordered array
- n (in) size of the array

The resulting integer i is as : $xdata(i) <= x < xdata(i+1)$. If $x < xdata(1)$ then $i = 0$ is returned. If $x > xdata(n)$ then $i = n$ is returned. Finally if $x = xdata(n)$ then $i = n - 1$ is returned.

5.2 Akima spline

fvn provides Akima spline interpolation and evaluation for both single and double precision real.

5.2.1 Interpolation

```
Module : use fvn_interpol
call fvn_akima(n,x,y,br,co)
```

- n (in) is an integer equal to the number of points
- x(n) (in) ,y(n) (in) are the known couples of coordinates
- br (out) on output contains a copy of x
- co(4,n) (out) is a real matrix containing the 4 coefficients of the Akima interpolation spline for a given interval.

5.2.2 Evaluation

```
Module : use fvn_interpol
y=fvn_spline_eval(x,n,br,co)
```

- x (in) is the point where we want to evaluate
- n (in) is the number of known points and br(n) (in), co(4,n) (in) are the outputs of fvn_x_akima(n,x,y,br,co)

5.2.3 Example

In the following example we will use Akima splines to interpolate a sinus function with 30 points between -10 and 10. We then use the evaluation function to calculate the coordinates of 1000 points between -11 and 11, and write a 3 columns file containing : x, calculated sin(x), interpolation evaluation of sin(x).

One can see that the interpolation is very efficient even with only 30 points. Of course as soon as we leave the -10 to 10 interval, the values are extrapolated and thus can lead to very inacurrate values.

```
program akima
 use fvn_interpol
 implicit none

 integer :: nbpoints,nppoints,i
 real(8),dimension(:),allocatable :: x_d,y_d,breakpoints_d
 real(8),dimension(:,:),allocatable :: coeff_fvn_d
 real(8) :: xstep_d,xp_d,ty_d,fvn_y_d
```

```
 open(2,file='fvn_akima_double.dat')
 open(3,file='fvn_akima_breakpoints_double.dat')
 nbpoints=30
 allocate(x_d(nbpoints))
 allocate(y_d(nbpoints))
 allocate(breakpoints_d(nbpoints))
 allocate(coeff_fvn_d(4,nbpoints))

 xstep_d=20./dfloat(nbpoints)
 do i=1,nbpoints
    x_d(i)=-10.+dfloat(i)*xstep_d
    y_d(i)=dsin(x_d(i))
    write(3,44) (x_d(i),y_d(i))
 end do
 close(3)

 call fvn_d_akima(nbpoints,x_d,y_d,breakpoints_d,coeff_fvn_d)

 nppoints=1000
 xstep_d=22./dfloat(nppoints)
 do i=1,nppoints
    xp_d=-11.+dfloat(i)*xstep_d
    ty_d=dsin(xp_d)
    fvn_y_d=fvn_d_spline_eval(xp_d,nbpoints-1,breakpoints_d,coeff_fvn_d)
    write(2,44) (xp_d,ty_d,fvn_y_d)
 end do

 close(2)

44      FORMAT(4(1X,1PE22.14))

end program
```

Results are plotted on figure 1

6 Least square polynomial

fvn provide a function to find a least square polynomial of a given degree, for real in single or double precision. It is performed using Lapack subroutine sgels (dgels), which solve this problem.

```
Module : use fvn_linear
call fvn_lspoly(np,x,y,deg,coeff,status)
```

- np (in) is an integer equal to the number of points
- x(np) (in),y(np) (in) are the known coordinates
- deg (in) is an integer equal to the degree of the desired polynomial, it must be lower than np.
- coeff(deg+1) (out) on output contains the polynomial coefficients
- status (out) is an integer containing 0 if a problem occured.

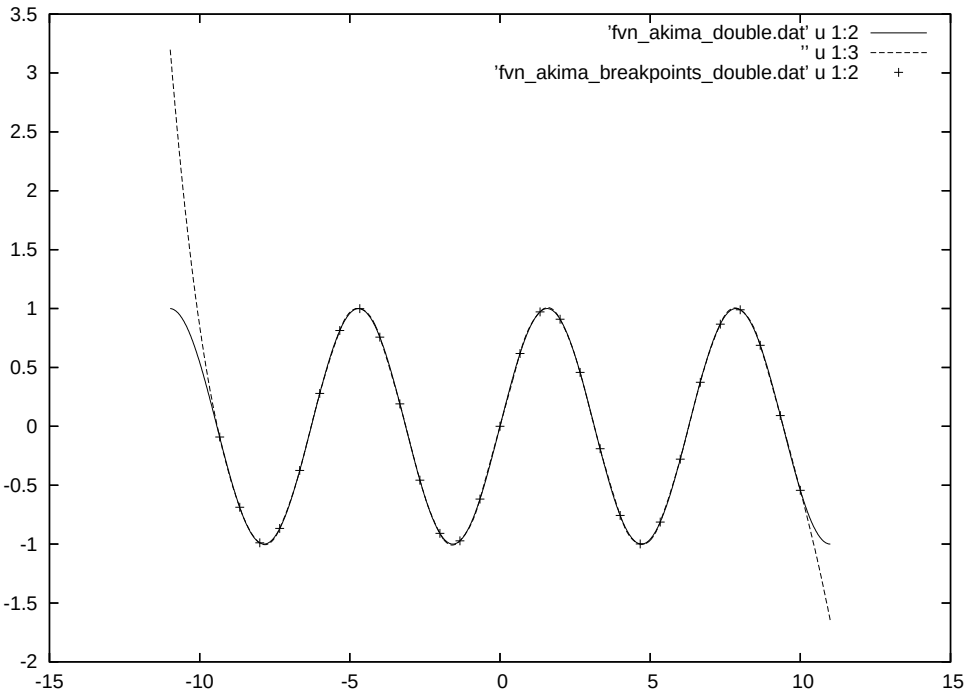

Figure 1: Akima Spline Interpolation

Example

Here's a simple example : we've got 13 measurement points and we want to find the least square degree 3 polynomial for these points :

```
program lsp
use fvn_linear
implicit none

integer,parameter :: npoints=13,deg=3
integer :: status,i
real(kind=8) :: xm(npoints),ym(npoints),xstep,xc,yc
real(kind=8) :: coeff(deg+1)

xm = (/ -3.8,-2.7,-2.2,-1.9,-1.1,-0.7,0.5,1.7,2.,2.8,3.2,3.8,4. /)
ym = (/ -3.1,-2.,-0.9,0.8,1.8,0.4,2.1,1.8,3.2,2.8,3.9,5.2,7.5  /)

open(2,file='fvn_lsp_double_mesure.dat')
open(3,file='fvn_lsp_double_poly.dat')

do i=1,npoints
   write(2,44) xm(i),ym(i)
end do
close(2)


call fvn_d_lspoly(npoints,xm,ym,deg,coeff,status)
```

```
 xstep=(xm(npoints)-xm(1))/1000.
 do i=1,1000
    xc=xm(1)+(i-1)*xstep
    yc=poly(xc,coeff)
    write(3,44) xc,yc
 end do
 close(3)

44      FORMAT(4(1X,1PE22.14))

contains
function poly(x,coeff)
    implicit none
    real(8) :: x
    real(8) :: coeff(deg+1)
    real(8) :: poly
    integer :: i

    poly=0.

    do i=1,deg+1
        poly=poly+coeff(i)*x**(i-1)
    end do

end function
end program
```

The results are plotted on figure 2 .

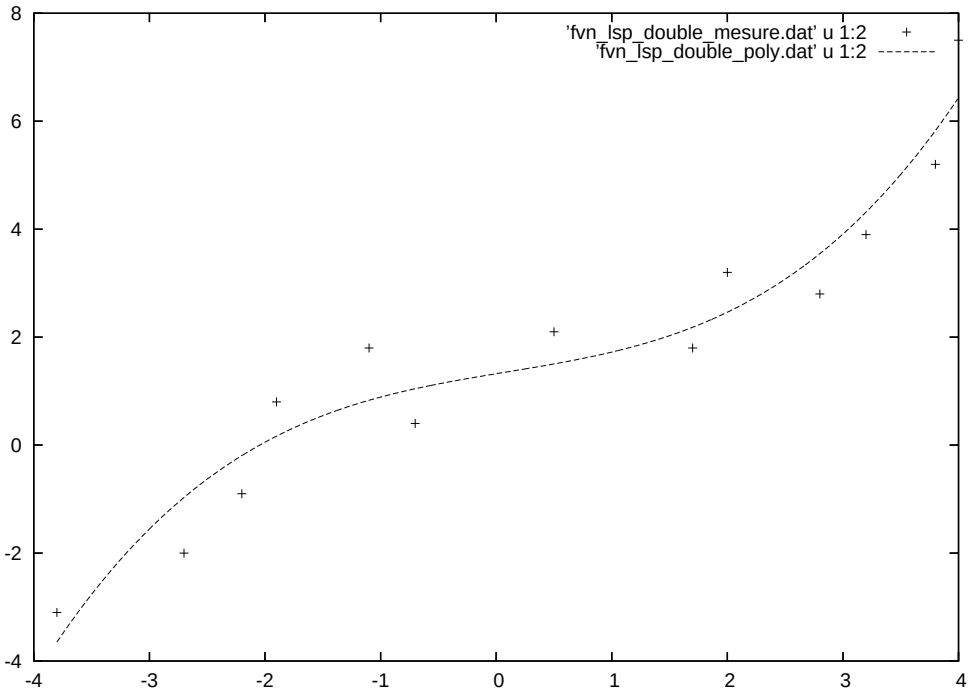

Figure 2: Least Square Polynomial

7 General least square fitting

fvn provides a routine performing a general least square fitting using Levenberg-Marquardt algorithm. It uses routine `lmdif` from `MINPACK` (http://www.netlib.org/minpack). The used version has been converted to fortran90 by Alan Miller `amiller @ bigpond.net.au`.
The purpose of `fvn_lm` is to minimize the sum of the squares of m nonlinear functions in n variables by a modification of the Levenberg-Marquardt algorithm. This is done by using the more general least-squares solver lmdif. The user must provide a subroutine which calculates the functions. The jacobian is then calculated by a forward-difference approximation.

```
Module : use fvn_misc
call fvn_lm(fcn,m,n,a,info,tol)
```

- fcn is the user-supplied subroutine which calculates the functions. fcn must follow the following interface that must be declared in the calling subroutine :

```
interface
    subroutine fcn(m,n,a,fvec,iflag)
        use fvn_common
        integer(ip_kind), intent(in) :: m
        integer(ip_kind), intent(in) :: n
        real(dp_kind), dimension(:), intent(in) :: a
        real(dp_kind), dimension(:), intent(inout) :: fvec
        integer(ip_kind), intent(inout) :: iflag
    end subroutine
end interface
```

  This is the function which calculate the differences for which which square sum will be minimized outputing this difference in vector fvec. Parameters of fcn are as follows :

  - m : positive integer input variable set to the number of functions (number of measurement points)
  - n : positive integer input variable set to the number of variables (number of parameters in the function to fit)
  - a : vector of length n containing parameters for which fcn should perform the calculation
  - fvec : vector of length m containing the resulting evaluation
  - iflag : integer normally not used, can be used to exit the the algorithm by setting it to a negative value

- m : positive integer input variable set to the number of functions (number of measurement points)
- n : positive integer input variable set to the number of variables (number of parameters in the function to fit)
- a(n) : vector of length n, on input must contains an initial guess (or unity vector) and on output the solution vector
- info : is an output positive integer
  - info = 0 improper input parameters.
  - info = 1 algorithm estimates that the relative error in the sum of squares is at most tol.
  - info = 2 algorithm estimates that the relative error between x and the solution is at most tol.

  - info = 3 conditions for info = 1 and info = 2 both hold.
  - info = 4 fvec is orthogonal to the columns of the jacobian to machine precision.
  - info = 5 number of calls to fcn has reached or exceeded 200*(n+1).
  - info = 6 tol is too small. no further reduction in the sum of squares is possible.
  - info = 7 tol is too small. No further improvement in the approximate solution x is possible.

- tol : is an optional positive value. Termination occurs when the algorithm estimates either that the relative error in the sum of squares is at most tol or that the relative error between x and the solution is at most tol. If not provided default value is : sqrt(epsilon(0.0d0))

Example

Here's a simple example solving the same problem as for the least square polynomial example but using `fvn_lm` instead :

```
module excursion
real(8), dimension(:), pointer :: xm => NULL()
real(8), dimension(:), pointer :: ym => NULL()
end module


program gls
use fvn_misc
use excursion
implicit none

    interface
        subroutine zef(m, n, x, fvec, iflag)
            integer(4), intent(in) :: m,n
            real(8), dimension(:), intent(in) :: x
            real(8), dimension(:), intent(inout) :: fvec
            integer(4), intent(inout) :: iflag
        end subroutine
    end interface


 integer,parameter :: npoints=13,deg=3
 integer :: status,i
 real(kind=dp_kind) :: xstep,xc,yc
 real(kind=dp_kind) :: coeff(deg+1)
 real(kind=dp_kind) :: fvec(npoints)
 integer(4) :: iwa(deg+1)
 integer(4) :: info
 real(8) :: tol

 allocate(xm(npoints),ym(npoints))

 xm = (/ -3.8,-2.7,-2.2,-1.9,-1.1,-0.7,0.5,1.7,2.,2.8,3.2,3.8,4. /)
 ym = (/ -3.1,-2.,-0.9,0.8,1.8,0.4,2.1,1.8,3.2,2.8,3.9,5.2,7.5 /)
 open(2,file='fvn_lsp_double_mesure.dat')
 open(3,file='fvn_lm.dat')
 do i=1,npoints
```

```
    write(2,44) xm(i),ym(i)
 end do
 close(2)

 coeff=1.
 call fvn_lm(zef,npoints,deg+1,coeff,info)
 write(*,*) "info : ",info

 xstep=(xm(npoints)-xm(1))/1000.
 do i=1,1000
    xc=xm(1)+(i-1)*xstep
    yc=poly(xc,coeff)
    write(3,44) xc,yc
 end do
 close(3)
write(*,*) "All done, plot results with gnuplot using command :"
write(*,*) "pl 'fvn_lsp_double_mesure.dat' u 1:2 w p,'fvn_lm.dat' u 1:2 w l"
44      FORMAT(4(1X,1PE22.14))
contains
function poly(x,coeff)
    use fvn_common
    implicit none
    real(kind=dp_kind) :: x
    real(kind=dp_kind) :: coeff(deg+1)
    real(kind=dp_kind) :: poly
    integer :: i
    poly=0.
    do i=1,deg+1
         poly=poly+coeff(i)*x**(i-1)
    end do
end function
end program

subroutine zef(m, n, x, fvec, iflag)
    use excursion
    implicit none
    integer(4), intent(in) :: m,n
    real(8), dimension(:), intent(in) :: x
    real(8), dimension(:), intent(inout) :: fvec
    integer(4), intent(inout) :: iflag

    integer(4) :: i

    do i=1,m
        fvec(i)=ym(i)-(x(1)+x(2)*xm(i)+x(3)*xm(i)**2+x(4)*xm(i)**3)
    enddo
end subroutine
```

Note the need to use a supplementary module `excursion`, to allow the subroutine `zef` to have access to the measurement points.

8 Zero finding

fvn provides a routine for finding zeros of a complex function using Muller algorithm (only for double complex type). It is based on a version provided on the web by Hans D Mittelmann http://plato.asu.edu/ftp/other_software/muller.f.

```
Module : use fvn_misc
call fvn_muller(f,eps,eps1,kn,nguess,n,x,itmax,infer,ier)
```

- f (in) is the complex function (kind=8) for which we search zeros
- eps (in) is a real(8) corresponding to the first stopping criterion : let fp(z)=f(z)/p where p = (z-z(1))*(z-z(2))*,,,*(z-z(k-1)) and z(1),...,z(k-1) are previously found roots. if ((cdabs(f(z)).le.eps) .and. (cdabs(fp(z)).le.eps)), then z is accepted as a root.
- eps1 (in) is a real(8) corresponding to the second stopping criterion : a root is accepted if two successive approximations to a given root agree within eps1. Note that if either or both of the stopping criteria are fulfilled, the root is accepted.
- kn (in) is an integer equal to the number of known roots, which must be stored in x(1),...,x(kn), prior to entry in the subroutine.
- nguess (in) is the number of initial guesses provided. These guesses must be stored in x(kn+1),...,x(kn+nguess). nguess must be set equal to zero if no guesses are provided.
- n (in) is an integer equal to the number of new roots to be found.
- x (inout) is a complex(8) vector of length kn+n. x(1),...,x(kn) on input must contain any known roots. x(kn+1),..., x(kn+n) on input may, on user option, contain initial guesses for the n new roots which are to be computed. If the user does not provide an initial guess, zero is used. On output, x(kn+1),...,x(kn+n) contain the approximate roots found by the subroutine.
- itmax (in) is an integer equal to the maximum allowable number of iterations per root.
- infer (out) is an integer vector of size kn+n. On output infer(j) contains the number of iterations used in finding the j-th root when convergence was achieved. If convergence was not obtained in itmax iterations, infer(j) will be greater than itmax
- ier (out) is an integer used as an error parameter. ier = 33 indicates failure to converge within itmax iterations for at least one of the (n) new roots.

This subroutine always returns the last approximation for root j in x(j). if the convergence criterion is satisfied, then infer(j) is less than or equal to itmax. if the convergence criterion is not satisified, then infer(j) is set to either itmax+1 or itmax+k, with k greater than 1. infer(j) = itmax+1 indicates that muller did not obtain convergence in the allowed number of iterations. in this case, the user may wish to set itmax to a larger value. infer(j) = itmax+k means that convergence was obtained (on iteration k) for the deflated function fp(z) = f(z)/((z-z(1)...(z-z(j-1))) but failed for f(z). in this case, better initial guesses might help or, it might be necessary to relax the convergence criterion.

Example

Example to find the ten roots of $x^{10} - 1$

```
program muller
use fvn_misc
implicit none
```

```
integer :: i,info
complex(8),dimension(10) :: roots
integer,dimension(10) :: infer
complex(8), external :: f

call fvn_z_muller(f,1.d-12,1.d-10,0,0,10,roots,200,infer,info)

write(*,*) "Error code :",info
do i=1,10
   write(*,*) roots(i),infer(i)
enddo
end program

function f(x)
   complex(8) :: x,f
   f=x**10-1
end function
```

9 Numerical integration

Using an integrated slightly modified version of quadpack http://www.netlib.org/quadpack, fvn provide adaptative numerical integration (Gauss Kronrod) of real functions of 1 and 2 variables. fvn also provide a function to calculate Gauss-Legendre abscissas and weight, and a simple non adaptative integration subroutine. All routines exists only in fvn for double precision real.

9.1 Gauss Legendre Abscissas and Weigth

This subroutine was inspired by Numerical Recipes routine gauleg.

```
Module : use fvn_integ
call fvn_gauss_legendre(n,qx,qw)
```

- n (in) is an integer equal to the number of Gauss Legendre points
- qx (out) is a real(8) vector of length n containing the abscissas.
- qw (out) is a real(8) vector of length n containing the weigths.

This subroutine computes n Gauss-Legendre abscissas and weigths

9.2 Gauss Legendre Numerical Integration

```
Module : use fvn_integ
call fvn_gl_integ(f,a,b,n,res)
```

- f (in) is a real(8) function to integrate
- a (in) and b (in) are real(8) respectively lower and higher bound of integration
- n (in) is an integer equal to the number of Gauss Legendre points to use
- res (out) is a real(8) containing the result

This function is a simple Gauss Legendre integration subroutine, which evaluate the integral of function f as in equation 3 using n Gauss-Legendre pairs.

9.3 Gauss Kronrod Adaptative Integration

This kind of numerical integration is an iterative procedure which try to achieve a given precision.

9.3.1 Numerical integration of a one variable function

```
Module : use fvn_integ
call fvn_integ_1_gk(f,a,b,epsabs,epsrel,key,res,abserr,ier,limit)
```

This routine evaluate the integral of function f as in equation 3

- f (in) is an external real(8) function of one variable
- a (in) and b (in) are real(8) respectively lower an higher bound of integration
- epsabs (in) and epsrel (in) are real(8) respectively desired absolute and relative error
- key (in) is an integer between 1 and 6 correspondind to the Gauss-Kronrod rule to use :
  - 1 : 7 - 15 points
  - 2 : 10 - 21 points
  - 3 : 15 - 31 points
  - 4 : 20 - 41 points
  - 5 : 25 - 51 points
  - 6 : 30 - 61 points
- res (out) is a real(8) containing the estimation of the integration.
- abserr (out) is a real(8) equal to the estimated absolute error
- ier (out) is an integer used as an error flag
  - 0 : no error
  - 1 : maximum number of subdivisions allowed has been achieved. one can allow more subdivisions by increasing the value of limit (and taking the according dimension adjustments into account). however, if this yield no improvement it is advised to analyze the integrand in order to determine the integration difficulaties. If the position of a local difficulty can be determined (i.e.singularity, discontinuity within the interval) one will probably gain from splitting up the interval at this point and calling the integrator on the subranges. If possible, an appropriate special-purpose integrator should be used which is designed for handling the type of difficulty involved.
  - 2 : the occurrence of roundoff error is detected, which prevents the requested tolerance from being achieved.
  - 3 : extremely bad integrand behaviour occurs at some points of the integration interval.
  - 6 : the input is invalid, because (epsabs.le.0 and epsrel.lt.max(50*rel.mach.acc.,0.5d-28)) or limit.lt.1 or lenw.lt.limit*4. result, abserr, neval, last are set to zero. Except when lenw is invalid, iwork(1), work(limit*2+1) and work(limit*3+1) are set to zero, work(1) is set to a and work(limit+1) to b.
- limit (in) is an optional integer equal to maximum number of subintervals in the partition of the given integration interval (a,b). If the parameter is not present a default value of 500 will be used.

$$\int_a^b f(x)\ dx \tag{3}$$

9.3.2 Numerical integration of a two variable function

```
Module : use fvn_integ
call fvn_integ_2_gk(f,a,b,g,h,epsabs,epsrel,key,res,abserr,ier,limit)
```

This function evaluate the integral of a function f(x,y) as defined in equation 4. The parameters of same name as in the previous paragraph have exactly the same function and behaviour thus only what differs is decribed here

- a (in) and b (in) are real(8) corresponding respectively to lower and higher bound of integration for the x variable.
- g(x) (in) and h(x) (in) are external functions describing the lower and higher bound of integration for the y variable as a function of x.

$$\int_a^b \int_{g(x)}^{h(x)} f(x,y)\ dy\ dx \tag{4}$$

Example

```
program integ
 use fvn_integ
 implicit none

 real(8), external :: f1,f2,g,h
 real(8) :: a,b,epsabs,epsrel,abserr,res
 integer :: key,ier

 a=0.
 b=1.
 epsabs=1d-8
 epsrel=1d-8
 key=2
 call fvn_d_integ_1_gk(f1,a,b,epsabs,epsrel,key,res,abserr,ier,500)
 write(*,*) "Integration of x*x between 0 and 1 : "
 write(*,*) res

 call fvn_d_integ_2_gk(f2,a,b,g,h,epsabs,epsrel,key,res,abserr,ier,500)
 write(*,*) "Integration of x*y between 0 and 1 on both x and y : "
 write(*,*) res


end program

function f1(x)
 implicit none
    real(8) :: x,f1
    f1=x*x
end function

function f2(x,y)
 implicit none
    real(8) :: x,y,f2
    f2=x*y
end function
```

```
function g(x)
 implicit none
    real(8) :: x,g
    g=0.
end function

function h(x)
 implicit none
    real(8) :: x,h
    h=1.
end function
```

10 Special functions

Specials functions are available in fvn by using an implementation of fnlib http://www.netlib.org/fn with some additions. This can be used separatly from the rest of fvn by using the module `fvn_fnlib` and linking the library `libfvn_fnlib.a` . The module provides a generic interfaces to all the routines. Specific names of the routines are given in the description.

```
Module : use fvn_fnlib
```

Important Note Due to the addition of fnlib to fvn, some functions that were in fvn and are redondant are now removed from fvn, so update your code now and replace them with the fnlib version. These are listed here after :

- `fvn_z_acos` replaced by `acos`
- `fvn_z_asin` replaced by `asin`
- `fvn_d_asinh` replaced by `asinh`
- `fvn_d_acosh` replaced by `acosh`
- `fvn_s_csevl` replaced by `csevl`
- `fvn_d_csevl` replaced by `csevl`
- `fvn_d_factorial` replaced by `fac`
- `fvn_d_lngamma` replaced by `alngam`

10.1 Elementary functions

10.1.1 carg

```
carg(z)
```

- z (in) is a complex

This function evaluates the argument of the complex z. That is θ for $z = \rho e^{i\theta}$.
Specific interfaces : `carg,zarg`

10.1.2 cbrt

```
cbrt(x)
```

- x is a real or complex

This function evaluates the cubic root of the argument x.
Specific interfaces : `cbrt,dcbrt,ccbrt,zcbrt`

10.1.3 exprl

`exprl(x)`

- x is a real or complex

This function evaluates $\frac{e^x-1}{x}$.
Specific interfaces : `exprel,dexprl,cexprl,zexprl`

10.1.4 log10

`log10(x)`

- x is a real or complex

This function is an extension of the intrinsic function log10 to complex arguments.
Specific interfaces : `clog10,zlog10`

10.1.5 alnrel

`alnrel(x)`

- x is a real or complex

This function evaluates $ln(1+x)$.
Specific interfaces : `alnrel,dlnrel,clnrel,zlnrel`

10.2 Trigonometry

10.2.1 tan

`tan(x)`

- x is a real or complex

This function evaluates the tangent of the argument. It is an extension of the intrinsic function tan to complex arguments.
Specific interfaces : `ctan,ztan`

10.2.2 cot

`cot(x)`

- x is a real or complex

This function evaluate the cotangent of the argument.
Specific interfaces : `cot,dcot,ccot,zcot`

10.2.3 sindg

`sindg(x)`

- x is a real

This function evaluate the sinus of the argument expressed in degrees.
Specific interfaces : `sindg,dsindg`

10.2.4 cosdg

```
cosdg(x)
```

- x is a real

This function evaluate the cosinus of the argument expressed in degrees.
Specific interfaces : `cosdg,dcosdg`

10.2.5 asin

```
asin(x)
```

- x is a real or complex

This function evaluates the arc sine of the argument. It is an extension of the intrinsic function asin to complex arguments.
Specific interfaces : `casin,zasin`

10.2.6 acos

```
acos(x)
```

- x is a real or complex

This function evaluates the arc cosine of the argument. It is an extension of the intrinsic function acos to complex arguments.
Specific interfaces : `cacos,zacos`

10.2.7 atan

```
atan(x)
```

- x is a real or complex

This function evaluates the arc tangent of the argument. It is an extension of the intrinsic function atan to complex arguments.
Specific interfaces : `catan,zatan`

10.2.8 atan2

```
atan2(x,y)
```

- x,y are real or complex

This function evaluates the arc tangent of $\frac{x}{y}$. It is an extension of the intrinsic function atan2 to complex arguments.
Specific interfaces : `catan2,zatan2`

10.2.9 sinh

```
sinh(x)
```

- x is a real or complex

This function evaluates the hyperbolic sine of the argument. It is an extension of the intrinsic function sinh to complex arguments.
Specific interfaces : `csinh,zsinh`

10.2.10 cosh

`cosh(x)`

- x is a real or complex

This function evaluates the hyperbolic cosine of the argument. It is an extension of the intrinsic function cosh to complex arguments.

Specific interfaces : `ccosh,zcosh`

10.2.11 tanh

`tanh(x)`

This function evaluates the hyperbolic tangent of the argument. It is an extension of the intrinsic function tanh to complex arguments.

Specific interfaces : `ctanh,ztanh`

10.2.12 asinh

`asinh(x)`

- x is a real or complex

This function evaluates the arc hyperbolic sine of the argument.

Specific interfaces : `asinh,dasinh,casinh,zasinh`

10.2.13 acosh

`acosh(x)`

- x is a real or complex

This function evaluates the arc hyperbolic cosine of the argument.

Specific interfaces : `acosh,dacosh,cacosh,zacosh`

10.2.14 atanh

`atanh(x)`

- x is a real or complex

This function evaluates the arc hyperbolic tangent of the argument.

Specific interfaces : `atanh,datanh,catanh,zatanh`

10.3 Exponential Integral and related

10.3.1 ei

`ei(x)`

- x is a real

This function evaluates the exponential integral for argument greater then 0 and the Cauchy principal value for argument less than 0. It is define by equation 5 for $x \neq 0$.

$$ei(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t} dt \tag{5}$$

Specific interfaces : `ei,dei`

10.3.2 e1

`e1(x)`

- x is a real or complex

For a real argument, this function evaluates the exponential integral for argument greater than 0 and the Cauchy principal value for argument less than 0. It is define by equation 6 for $x \neq 0$.

$$e1(x) = \int_x^\infty \frac{e^{-t}}{t} dt \tag{6}$$

For a complex argument, the notation in equation 7 is used (Abramowitz and Stegun, p.228 http://www.math.ucla.edu/~cbm/aands/page_228.htm):

$$e1(z) = \int_z^\infty \frac{e^{-t}}{t} dt \text{ with } |arg(z)| < \pi \tag{7}$$

For positive values of real part of z, this can be written as in equation 8 :

$$e1(z) = \int_1^\infty \frac{e^{-tz}}{t} dt \text{ with } Re(z) > 0 \tag{8}$$

Specific interfaces : `e1,de1,ze1`

10.3.3 ali

`ali(x)`

- x is a real

This function evaluates the logarithm integral. it is define by equation 9 for $x > 0$ and $x \neq 1$.

$$ali(x) = -\int_0^x \frac{dt}{ln(x)} \tag{9}$$

Specific interfaces : `ali,dli`

10.3.4 si

`si(x)`

- x is a real

This function evaluates the sine integral defined by equation 10.

$$si(x) = \int_0^x \frac{sin(t)}{t} dt \tag{10}$$

Specific interfaces : `si,dsi`

10.3.5 ci

`ci(x)`

- x is a real

This function evaluates the cosine integral defined by equation 11 where $\gamma \approx 0.57721566$ represent Euler's constant.

$$ci(x) = \gamma + ln(x) + \int_0^x \frac{1 - cos(t)}{t} dt \tag{11}$$

Specific interfaces : `ci,dci`

10.3.6 cin

`cin(x)`

- x is a real

This function evaluates the cosine integral alternate definition given by equation 12.

$$cin(x) = \int_0^x \frac{1 - cos(t)}{t} dt \tag{12}$$

Specific interface : `cin,dcin`

10.3.7 shi

$$shi(x) \tag{13}$$

- x is a real

This function evaluates the hyperbolic sine integral defined by equation 14.

$$shi(x) = \int_0^x \frac{sinh(t)}{t} dt \tag{14}$$

Specific interfaces : `shi,dshi`

10.3.8 chi

`chi(x)`

- x is a real

This function evaluates the hyperbolic cosine integral defined by equation 15 where $\gamma \approx 0.57721566$ represent Euler's constant.

$$chi(x) = \gamma + ln(x) + \int_0^x \frac{cosh(t) - 1}{t} dt \tag{15}$$

Specific interfaces : chi,dchi

10.3.9 cinh

`cinh(x)`

- x is a real

This function evaluates the hyperbolic cosine integral alternate definition given by equation 16.

$$cinh(x) = \int_0^x \frac{cosh(t) - 1}{t} dt \tag{16}$$

Specific interfaces : cinh,dcinh

10.4 Gamma function and related

10.4.1 fac

`fac(n)`
`dfac(n)`

- n is an integer

This function return $n!$ as a real(4) or real(8) for dfac. There's no generic interface for this one.
Specific interfaces : `fac,dfac`

10.4.2 binom

```
binom(n,m)
dbinom(n,m)
```

- n,m are integers

This function return the binomial coefficient defined by equation 17 with $n \geq m \geq 0$. binom returns a real(4), dbinom a real(8). There's no generic interface for this one.

$$binom(n,m) = C_n^m = \frac{n!}{m!(n-m)!} \tag{17}$$

Specific interfaces : `binom,dbinom`

10.4.3 gamma

```
gamma(x)
```

- x is a real or complex

This function evaluates $\Gamma(x)$ defined by equation 18.

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t} dt \tag{18}$$

Note that $n! = \Gamma(n+1)$.
Specific interfaces :`gamma,dgamma,cgamma,zgamm`

10.4.4 gamr

```
gamr(x)
```

- x is a real or complex

This function evaluates the reciprocal gamma function $gamr(x) = \frac{1}{\Gamma(x)}$

10.4.5 alngam

```
alngam(x)
```

- x is a real or complex

This function evaluates $ln(|\Gamma(x)|)$
Specific interfaces : `alngam,dlngam,clngam,zlngam`

10.4.6 algams

```
call algams(x,algam,sgngam)
```

- x (in) is a real
- algam (out) is a real
- sgngam (out) is a real

This subroutine evaluates the logarithm of the absolute value of gamma and the sign of gamma. $algam = ln(|\Gamma(x)|)$ and $sgngam = 1.0$ or -1.0 according to the sign of $\Gamma(x)$.
Specific interfaces : `algams,dlgams`

10.4.7 gami

`gami(a,x)`

- x is a positive real
- a is a strictly positive real

This function evaluates the incomplete gamma function defined by equation 19.

$$gami(a,x) = \gamma(a,x) = \int_0^x t^{a-1}e^{-t}dt \tag{19}$$

Specific interfaces : `gami,dgami`

10.4.8 gamic

`gamic(a,x)`

- x is a positive real
- a is a real

This function evaluates the complementary incomplete gamma function defined by equation 20.

$$gamic(a,x) = \Gamma(a,x) = \int_x^\infty t^{a-1}e^{-t}dt \tag{20}$$

Specific interfaces : `gamic,dgamic`

10.4.9 gamit

`gamit(a,x)`

- x is a positive real
- a is a real

This function evaluates the Tricomi's incomplete gamma function defined by equation 21.

$$gamit(a,x) = \gamma^*(a,x) = \frac{x^{-a}\gamma(a,x)}{\Gamma(a)} \tag{21}$$

Specific interfaces : `gamit,dgamit`

10.4.10 psi

`psi(x)`

- x is a real or complex

This function evaluates the psi function which is the logarithm derivative of the gamma function as defined in equation 22.

$$psi(x) = \psi(x) = \frac{d}{dx}ln(\Gamma(x)) \tag{22}$$

x must not be zero or a negative integer.

Specific interfaces : `psi,dpsi,cpsi,zpsi`

10.4.11 poch

`poch(a,x)`

- x is a real
- a is a real

This function evaluates a generalization of Pochhammer's symbol.
Pochhammer's symbol for n a positive integer is given by equation 23

$$(a)_n = a(a-1)(a-2)...(a-n+1) \tag{23}$$

The generalization of Pochhammer's symbol is given by equation 24

$$poch(a,x) = (a)_x = \frac{\Gamma(a+x)}{\Gamma(a)} \tag{24}$$

Specific interfaces : `poch,dpoch`

10.4.12 poch1

`poch1(a,x)`

- x is a real
- a is a real

This function is defined by equation 25. It is usefull for certains situations, especially when x is small.

$$poch1(a,x) = \frac{(a)_x - 1}{x} \tag{25}$$

Specific interfaces : `poch1,dpoch1`

10.4.13 beta

`beta(a,b)`

- a,b are real positive or complex

This function evaluates β function defined by equation 26.

$$beta(a,b) = \beta(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)} \tag{26}$$

Specific interfaces : `beta,dbeta,cbeta,zbeta`

10.4.14 albeta

`albeta(a,b)`

- a,b are real positive or complex

This function evaluates the natural logarithm of beta function : $ln(\beta(a,b))$
Specific interfaces : `albeta,dlbeta,clbeta,zlbeta`

10.4.15 betai

`betai(x,pin,qin)`

- x is a real in [0,1]
- pin and qin are strictly positive real

This function evaluates the incomplete beta function ratio, that is the probability that a random variable from a beta distribution having parameters pin and qin will be less than or equal to x. It is defined by equation 27.

$$betai(x, pin, qin) = I_x(pin, qin) = \frac{1}{\beta(pin, qin)} \int_0^x t^{pin-1}(1-t)^{qin-1} dt \tag{27}$$

Specific interfaces : `betai,dbetai`

10.5 Error function and related

10.5.1 erf

`erf(x)`

- x is a real

This function evaluates the error function defined by equation 28.

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{28}$$

Specific interfaces : `erf,derf`

10.5.2 erfc

`erfc(x)`

- x is a real

This function evaluates the complimentary error function defined by equation 29.

$$erfc(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt \tag{29}$$

Specific interfaces : `erfc,derfc`

10.5.3 daws

`daws(x)`

- x is a real

This function evaluates Dawson's function defined by equation 30.

$$daws(x) = e^{-x^2} \int_0^x e^{t^2} dt \tag{30}$$

Specific interfaces : `daws,ddaws`

10.6 Bessel functions and related

10.6.1 bsj0

`bsj0(x)`

- x is a real

This function evaluates Bessel function of the first kind of order 0 defined by equation 31.

$$bsj0(x) = J_0(x) = \frac{1}{\pi}\int_0^{\pi} cos(xsin(\theta))d\theta \tag{31}$$

Specific interfaces : `besj0,dbesj0`

10.6.2 bsj1

`bsj1(x)`

- x is a real

This function evaluates Bessel function of the first kind of order 1 defined by equation 32.

$$bsj1(x) = J_1(x) = \frac{1}{\pi}\int_0^{\pi} cos(xsin(\theta) - \theta)d\theta \tag{32}$$

Specific interfaces : `besj1,dbesj1`

10.6.3 bsjn

`bsjn(n,x,factor,big)`

- n is an integer
- x is a real
- factor is an optional integer
- big is an optional real

This function evaluates Bessel function of the first kind of order n (plotted in figure 3). These functions satisfy the recurrent relation 33.

$$J_{n+1}(x) = \frac{2n}{x}J_n(x) - J_{n-1}(x) \tag{33}$$

This relation is directly used in upward direction to compute $J_n(x)$ for $x > n$. However it is unstable for $x < n$, therefore a Miller's Algorithm is used. The principle of this method is to use the reccurent relation downward from an arbitrary higher than n order with an arbitrary seed and then normalize the solution with 34

$$1 = J_0 + 2J_2 + 2J_4 + 2J_6 + ... \tag{34}$$

The optional parameters `factor` and `big` can be used to modify the behaviour of the algorithm. `factor` is used in determining the arbitrary starting order (an even integer near $n + \sqrt{factor\ n}$), the default $factor$ value is 40 for single precision and 150 for double precision. `big` is a real determining the threshold for which anti-overflow counter measures has to be taken, default value is 1.10^{10}

By convenience, the routine accept $n = 0$ and $n = 1$, in that cases a call to $bsj0(x)$ or $bsj1(x)$ is actually performed.

Specific interfaces : `besjn,dbesjn`

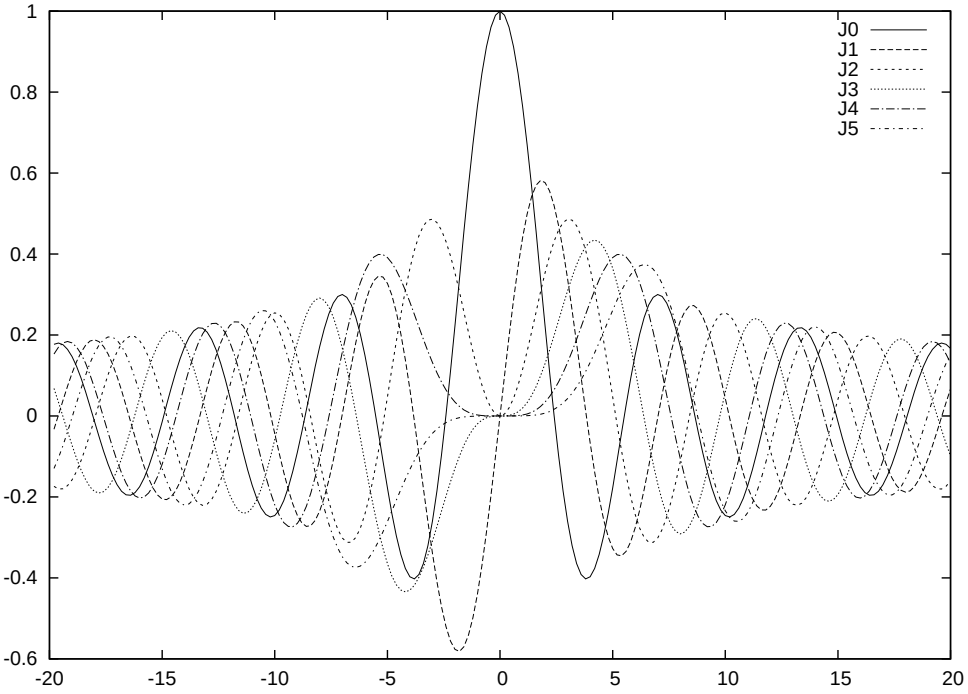

Figure 3: Bessel J_n functions family

10.6.4 besrj

```
call besrj(x,n,b)
```

- x (in) is a real
- n (in) is an integer
- b (out) is a real array of dimension n

This subroutine evaluates Bessel function of the first kind of order `0` to `n-1` for argument x and return the result in array b, which then contain `b(1)`=$J_0(x)$,`b(2)`=$J_1(x)$,...,`b(n)`=$J_{n-1}(x)$.

The algorithm is different from the one used in `bsjn`, the choice between the two depends on accuracy and timing considerations, there are test programs (`test_besrj` and `test_bestime`) in the `fvn_test` directory which can help choosing the good one.

Specific interfaces : `besrj,dbesrj`

10.6.5 bsy0

```
bsy0(x)
```

- x is a strictly positive real

This function evaluates the Bessel function of the second kind of order 0 defined by equation 35

$$bsy0(x) = Y_0(x) = \frac{1}{\pi}\int_0^{\pi} sin(xsin(\theta))d\theta - \frac{2}{\pi}\int_0^{\infty} e^{-xsinh(t)}dt \tag{35}$$

Specific interfaces : `besy0,dbesy0`

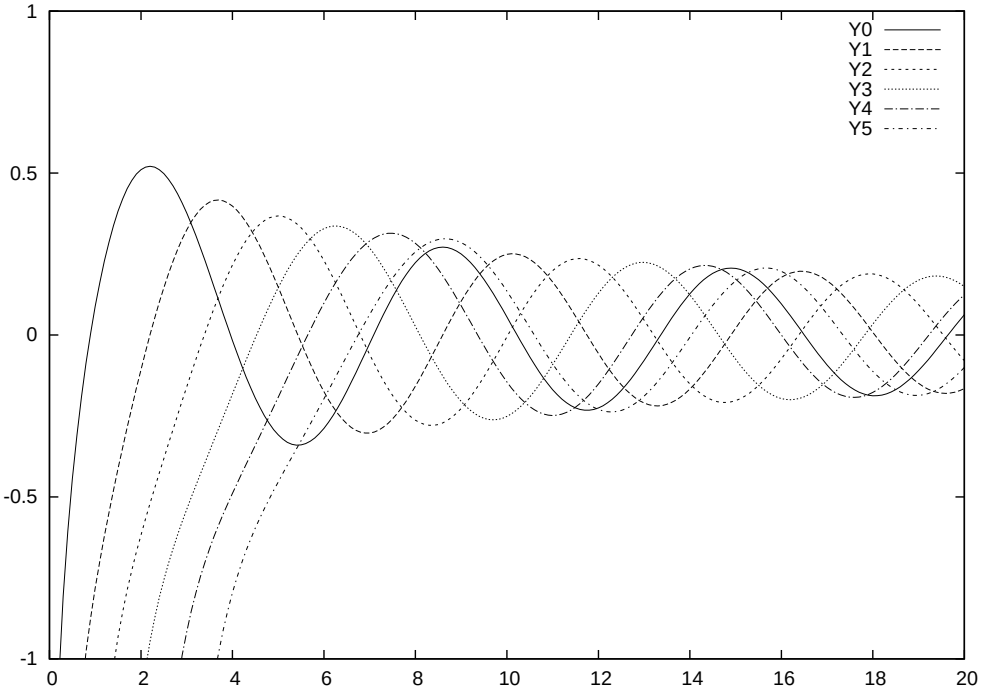

Figure 4: Bessel Y_n functions family

10.6.6 bsy1

`bsy1(x)`

- x is a strictly positive real

This function evaluates the Bessel function of the second kind of order 1 defined by equation 36.

$$bsy1(x) = Y_1(x) = -\frac{1}{\pi}\int_0^{\pi} sin(\theta - xsin(\theta))d\theta - \frac{1}{\pi}\int_0^{\infty}(e^t - e^{-t})e^{-xsinh(t)}dt \tag{36}$$

Specific interfaces : `besy1,dbesy1`

10.6.7 bsyn

`bsyn(n,x)`

- n is an integer
- x is a strictly positive real

This function evaluates the Bessel function of the second kind of order n (plotted in figure 4). These functions satisfy the recurrent relation 37.

$$Y_{n+1}(x) = \frac{2n}{x}Y_n(x) - Y_{n-1}(x) \tag{37}$$

This recurrent relation is directly used in the upward direction to compute $Y_n(x)$.

By convenience, the routine accept $n = 0$ and $n = 1$, in that cases a call to $bsy0(x)$ or $bsy1(x)$ is actually performed.

Specific interfaces : `besyn,dbesyn`

10.6.8 bsi0

`bsi0(x)`

- x is a real

This function evaluates the Bessel function of the third kind of order 0 defined by equation 38.

$$bsi0(x) = I_0(x) = \frac{1}{\pi}\int_0^{\pi} cosh(xcos(\theta))d\theta \tag{38}$$

Specific interfaces : `besi0,dbesi0`

10.6.9 bsi1

`bsi1(x)`

- x is a real

This function evaluates the Bessel function of the third kind of order 1 defined by equation 39.

$$bsi1(x) = I_1(x) = \frac{1}{\pi}\int_0^{\pi} e^{xcos(\theta)}cos(\theta)d\theta \tag{39}$$

Specific interfaces : `besi1,dbesi1`

10.6.10 bsin

`bsin(n,x,factor,big)`

- n is an integer
- x is a real
- factor is an optional integer
- big is an optional real

This function evaluates Bessel function of the third kind of order n (plotted in figure 5). These functions satisfy the recuurent relation 40

$$I_{n+1}(x) = -\frac{2n}{x}I_n(x) + I_{n-1}(x) \tag{40}$$

This relation is unstable in the upward direction, therefore a Miller's Algorithm is used to evaluate the function. Even if there's a usable normalization relation 41, it is not used in the routine, instead normalization is done by a simple call to `bsi0(x)`.

$$1 = I_0 - 2I_2 + 2I_4 - 2I_6 + ... \tag{41}$$

The optional parameters `factor` and `big` can be used to modify the behaviour of the algorithm. `factor` is used in determining the arbitrary starting order (an even integer near $n + \sqrt{factor\ n}$), the default $factor$ value is 40 for single precision and 150 for double precision. `big` is a real determining the threshold for which anti-overflow counter measures has to be taken, default value is 1.10^{10}

By convenience, the routine accept $n = 0$ and $n = 1$, in that cases a call to $bsi0(x)$ or $bsi1(x)$ is actually performed.

Specific interfaces : `besin,dbesin`

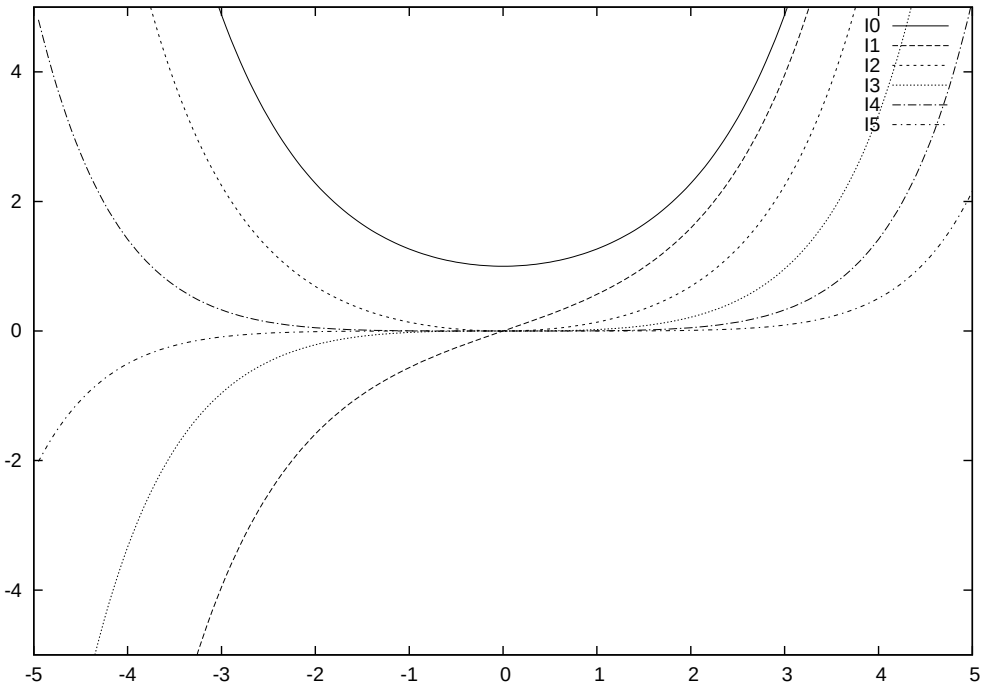

Figure 5: Bessel I_n functions family

10.6.11 besri

```
call besri(x,n,b)
```

- x (in) is a real
- n (in) is an integer
- b (out) is a real array of dimension n

This subroutine evaluates Bessel function of the third kind of order `0` to `n-1` for argument x and return the result in array b, which then contain `b(1)`=$I_0(x)$,`b(2)`=$I_1(x)$,...,`b(n)`=$I_{n-1}(x)$.

The algorithm is different from the one used in `bsin` and tends to be more accurate, the choice between the two depends on accuracy and timing considerations, there are test programs (`test_besri` and `test_bestime`) in the `fvn_test` directory which can help choosing the good one.

Specific interfaces : `besri,dbesri`

10.6.12 bsk0

```
bsk0(x)
```

- x is a strictly positive real

This function evaluates the modified Bessel function of the second kind of order 0 defined by equation 42

$$bsk0(x) = K_0(x) = \int_0^\infty cos(xsinh(t))dt \tag{42}$$

Specific interfaces : `besk0,dbesk0`

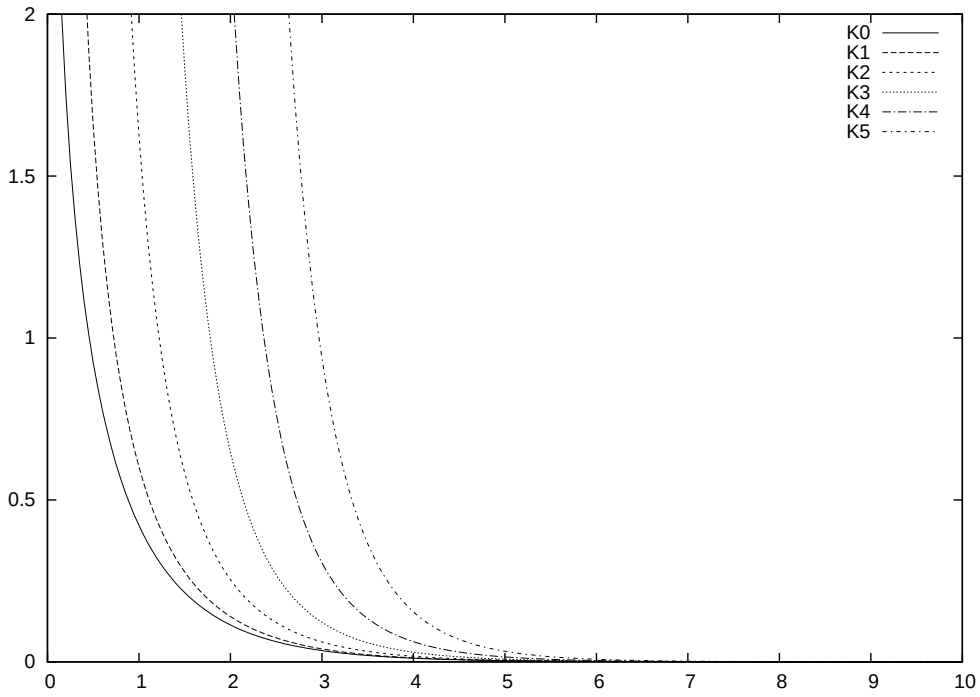

Figure 6: Bessel K_n functions family

10.6.13 bsk1

`bsk1(x)`

- x is a strictly positive real

This function evaluates the modified Bessel function of the second kind of order 1 defined by equation 43

$$bsk1(x) = K_1(x) = \int_0^\infty sin(xsinh(t))sinh(t)dt \tag{43}$$

Specific interfaces : `besk1,dbesk1`

10.6.14 bskn

`bskn(n,x)`

- n is an integer
- x is strictly positive real

This function evaluates the modified Bessel function of the second kind of order n (plotted in figure 6). These functions satisfy the recurrent relation 44

$$K_{n+1}(x) = \frac{2n}{x}K_n(x) + K_{n-1}(x) \tag{44}$$

This recurrent relation is directly used in the upward direction to compute $K_n(x)$.

By convenience, the routine accept $n = 0$ and $n = 1$, in that cases a call to $bsk0(x)$ or $bsk1(x)$ is actually performed.

Specific interface : `beskn,dbeskn`

10.6.15 bsi0e

```
bsi0e(x)
```

- x is a real

This function evaluates $e^{-|x|}I_0(x)$
Specific interfaces : `besi0e,dbsi0e`

10.6.16 bsi1e

```
bsi1e(x)
```

- x is a real

This function evaluates $e^{-|x|}I_1(x)$
Specific interfaces : `besi1e,dbsi1e`

10.6.17 bsk0e

```
bsk0e(x)
```

- x is a strictly positive real

This function evaluates $e^{x}K_0(x)$
Specific interfaces : `besk0e,dbsk0e`

10.6.18 bsk1e

```
bsk1e(x)
```

- x is a strictly positive real

This function evaluates $e^{x}K_1(x)$
Specific interfaces : `besk1e,dbsk1e`

10.6.19 bsks

```
call bsks(xnu,x,nin,bk)
```

- xnu (in) is a real with $|xnu| < 1$. It's the fractional order
- x (in) is a real. The value for which the sequence of Bessel functions is to be evaluated.
- nin (in) is an integer.
- bk (out) is a real vector of length abs(nin), containing the values of the function.

This subroutine evaluates a sequence of modified Bessel function of the second kind of fractional order.

If nin is positive, on completion $bk(1) = K_\nu(x)$,$bk(2) = K_{\nu+1}(x)$,...,$bk(nin) = K_{\nu+nin-1}(x)$. If nin is negative, on completion $bk(1) = K_\nu(x)$,$bk(2) = K_{\nu-1}(x)$,...,$bk(|nin|) = K_{\nu+nin+1}(x)$.

Specific interfaces : `besks,dbesks`

10.6.20 bskes

```
call bskes(xnu,x,nin,bke)
```

- xnu (in) is a real with $|xnu| < 1$. It's the fractional order
- x (in) is a real. The value for which the sequence of exponentialy scaled Bessel functions is to be evaluated.
- nin (in) is an integer. Number of elements in the sequence.
- bke (out) is a real vector of length abs(nin), containing the values of the function.

This subroutine evaluates a sequence of exponentially scaled modified Bessel function of the second kind of fractional order.

If nin is positive, on completion $bk(1) = e^x K_\nu(x)$,$bk(2) = e^x K_{\nu+1}(x)$,...,$bk(nin) = e^x K_{\nu+nin-1}(x)$. If nin is negative, on completion $bk(1) = e^x K_\nu(x)$,$bk(2) = e^x K_{\nu-1}(x)$,...,$bk(|nin|) = e^x K_{\nu+nin+1}(x)$.

Specific interfaces : `beskes,dbskes`

10.7 Airy function and related

10.7.1 ai

```
ai(x)
```

- x is a real

This function evaluates the airy function defined by equation 45

$$Ai(x) = \frac{1}{\pi}\int_0^\infty cos(xt + \frac{1}{3}t^3)dt \tag{45}$$

Specific interfaces : `ai,dai`

10.7.2 bi

```
bi(x)
```

- x is a real

This function evaluates the Airy function of the second kind defined by equation 46

$$Bi(x) = \frac{1}{\pi}\int_0^\infty e^{xt-\frac{1}{3}t^3}dt + \frac{1}{\pi}\int_0^\infty sin(xt + \frac{1}{3}t^3)dt \tag{46}$$

Specific interfaces : `bi,dbi`

10.7.3 aid

```
aid(x)
```

- x is a real

This function evaluates the derivative of the Airy function, $aid(x) = \frac{d}{dx}Ai(x)$.

Specific interface : `aid,daid`

10.7.4 bid

`bid(x)`

- x is a real

This function evaluates the derivative of the Airy function of the second kind, $bid(x) = \frac{d}{dx}Bi(x)$.
Specific interfaces : `bid,dbid`

10.7.5 aie

`aie(x)`

- x is a real

This function evaluates the exponentially scaled Airy function defined in equation 47.

$$aie(x) = \begin{cases} Ai(x) & \text{if } x \leq 0 \\ e^{\frac{2}{3}x^{\frac{3}{2}}} Ai(x) & \text{if } x > 0 \end{cases} \tag{47}$$

Specific interfaces : `aie,daie`

10.7.6 bie

`bie(x)`

- x is a real

This function evaluates the exponentially scaled Airy function of the second kind defined in equation 48.

$$bie(x) = \begin{cases} Bi(x) & \text{if } x \leq 0 \\ e^{-\frac{2}{3}x^{\frac{3}{2}}} Bi(x) & \text{if } x > 0 \end{cases} \tag{48}$$

Specific interfaces : `bie,dbie`

10.7.7 aide

`aide(x)`

- x is a real

This function evaluates the exponentially scaled derivative of the Airy function as defined in equation 49.

$$aide(x) = \begin{cases} Ai'(x) & \text{if } x \leq 0 \\ e^{\frac{2}{3}x^{\frac{3}{2}}} Ai'(x) & \text{if } x > 0 \end{cases} \tag{49}$$

Specific interfaces : `aide,daide`

10.7.8 bide

`bide(x)`

- x is a real

This function evaluates the exponentially scaled derivative of the Airy function of the second kind as defined in equation 50.

$$bie(x) = \begin{cases} Bi'(x) & \text{if } x \leq 0 \\ e^{-\frac{2}{3}x^{\frac{3}{2}}} Bi'(x) & \text{if } x > 0 \end{cases} \tag{50}$$

Specific interfaces : `bide,dbide`

10.8 Miscellanous functions

10.8.1 spenc

`spenc(x)`

- x is a real

This function evaluates Spence function defined in equation 51.

$$spenc(x) = s(x) = -\int_0^x \frac{ln(|1-t|)}{t} dt \tag{51}$$

Specific interfaces : `spenc,dspenc`

10.8.2 inits

`inits(os,nos,eta)`

- os is a real vector of length nos, containing the coefficients in an orthogonal series.
- nos is an integer
- eta is a real (Warning eta is a real(4) even with the double precision version) representing the requested accuracy.

This function initialize the orthogonal series so that inits is the number of terms needed to insure the error is no larger than eta.
Specific interfaces : `inits,initds`

10.8.3 csevl

`csevl(x,cs,n)`

- x is a real in [-1,1]
- cs is a real vector of length n containing the coefficients of the Chebyshev serie.
- n is an integer

This function evaluates the Chebyshev series whose coefficients are stored in cs.
Specific interfaces : `csevl,dcsevl`