

# FVN Documentation

William Daniau

January 25, 2008

## Contents

<b>1</b>	<b>Whatis fvn,licence,disclaimer etc</b>	<b>2</b>
1.1	Whatis fvn . . . . .	2
1.2	Licence . . . . .	2
1.3	Disclaimer . . . . .	2
<b>2</b>	<b>Naming scheme and convention</b>	<b>2</b>
<b>3</b>	<b>Linear algebra</b>	<b>2</b>
3.1	Matrix inversion . . . . .	3
3.2	Matrix determinants . . . . .	3
3.3	Matrix condition . . . . .	4
3.4	Eigenvalues/Eigenvectors . . . . .	4
3.5	Sparse solving . . . . .	5
<b>4</b>	<b>Interpolation</b>	<b>7</b>
4.1	Quadratic Interpolation . . . . .	7
4.1.1	One variable function . . . . .	7
4.1.2	Two variables function . . . . .	8
4.1.3	Three variables function . . . . .	9
4.1.4	Utility procedure . . . . .	10
4.2	Akima spline . . . . .	10
4.2.1	Interpolation . . . . .	10
4.2.2	Evaluation . . . . .	10
4.2.3	Example . . . . .	11
<b>5</b>	<b>Least square polynomial</b>	<b>12</b>
<b>6</b>	<b>Zero finding</b>	<b>13</b>
<b>7</b>	<b>Trigonometry</b>	<b>15</b>
7.1	Complex Sine Arc . . . . .	15
7.2	Complex Cosine Arc . . . . .	15
7.3	Real Sine Hyperbolic Arc . . . . .	16
7.4	Real Cosine Hyperbolic Arc . . . . .	16
<b>8</b>	<b>Numerical integration</b>	<b>16</b>
8.1	Gauss Legendre Abscissas and Weigth . . . . .	16
8.2	Gauss Legendre Numerical Integration . . . . .	16
8.3	Gauss Kronrod Adaptative Integration . . . . .	16
8.3.1	Numerical integration of a one variable function . . . . .	17
8.3.2	Numerical integration of a two variable function . . . . .	18

# 1 Whatis fvn,licence,disclaimer etc

## 1.1 Whatis fvn

fvn is a Fortran95 mathematical module. It provides various usefull subroutine covering linear algebra, numerical integration, least square polynomial, spline interpolation, zero finding, complex trigonometry etc.

Most of the work 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 also contains a slightly modified version of Quadpack <http://www.netlib.org/quadpack> for performing the numerical integration tasks.

This module has been initially written for the use of the “Acoustic and microsonic” group leaded by Sylvain Ballandras in institute Femto-ST <http://www.femto-st.fr/>.

## 1.2 Licence

The licence of fvn is free. You can do whatever you want with this code as far as you credit the authors.

### Authors

As of the day this manuel is written there’s only one author of fvn :  
William Daniau  
[william.daniau@femto-st.fr](mailto:william.daniau@femto-st.fr)

## 1.3 Disclaimer

The usual disclaimer applied : This software is provided AS IS in the hope it will be usefull. Use it at your own risks. The authors should not be taken responsible of anything that may result by the use of this software.

# 2 Naming scheme and convention

The naming scheme of the routines is as follow :

`fvn_x_name()`

where x 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).

# 3 Linear algebra

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

### 3.1 Matrix inversion

```
call fvn_x_matinv(d,a,inva,status)
```

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

#### Example

```
program inv
  use fvn
  implicit none

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

  call random_number(a)
  a=a*100

  call fvn_d_matinv(3,a,inva,status)
  write (*,*) a
  write (*,*)
  write (*,*) inva
  write (*,*)
  write (*,*) matmul(a,inva)
end program
```

### 3.2 Matrix determinants

```
det=fvn_x_det(d,a,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a matrix of type x. It will remain untouched.
- status (out) is an integer equal to zero if something went wrong

#### Example

```
program det
  use fvn
  implicit none

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

  call random_number(a)
  a=a*100

  deta=fvn_d_det(3,a,status)
  write (*,*) a
```

```

write (*,*)
write (*,*) "Det = ",deta
end program

```

### 3.3 Matrix condition

```
call fvn_x_matcon(d,a,rcond,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a matrix of type x. 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 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

$$R = \frac{1}{\text{norm}(A) * \text{norm}(\text{inv}A)} \quad (1)$$

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

$$L1 = \max_j \left( \sum_i |A(i,j)| \right) \quad (2)$$

#### Example

```

program cond
use fvn
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

```

### 3.4 Eigenvalues/Eigenvectors

```
call fvn_x_matev(d,a,evala,eveca,status)
```

- d (in) is an integer equal to the matrix rank
- a (in) is a matrix of type x. 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 integer equal to zero if something went wrong

### Example

```

program eigen
  use fvn
  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_d_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

```

## 3.5 Sparse solving

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.

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

`call fvn*_sparse_solve(n,nz,T,Ti,Tj,B,x,status)` where `*` is either `zl`, `zi`, `dl` or `di`

- For this family of subroutine the two letters (`zl,zi,dl,di`) describe 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)` (in) is a complex/real array containing the non-zero elements
- `Ti(nz),Tj(nz)` (in) are the indexes of the corresponding element of `T` in the original matrix.
- `B(n)` (in) is a complex/real array 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

### Example

```
program test_sparse
```

```
use fvn
implicit none
```

```
integer(8), parameter :: nz=12
integer(8), parameter :: n=5
complex(8),dimension(nz) :: A
integer(8),dimension(nz) :: Ti,Tj
complex(8),dimension(n) :: B,x
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 /)
```

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

```
end program
```

```
program test_sparse
```

```
use fvn
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 /)
```

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

```
end program
```

## 4 Interpolation

### 4.1 Quadratic Interpolation

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

#### 4.1.1 One variable function

```
value=fvn_x_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
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
```

### 4.1.2 Two variables function

```
value=fvn_x_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
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
```



### 4.1.3 Three variables function

value=fvn\_x\_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(nz) 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

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
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

```

#### 4.1.4 Utility procedure

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

```
call fvn_x_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 :  $xdata(i) \leq x < xdata(i+1)$ . 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.

## 4.2 Akima spline

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

### 4.2.1 Interpolation

```
call fvn_x_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.

### 4.2.2 Evaluation

```
y=fvn_x_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)

### 4.2.3 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 inaccurate values.

```
program akima
  use fvn
  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,yp_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

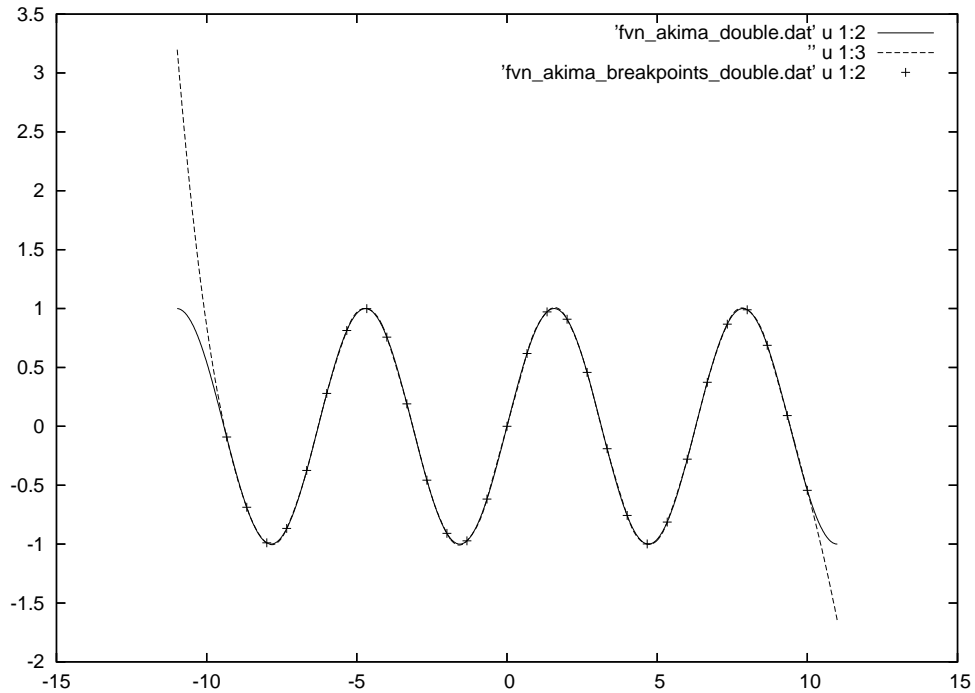


Figure 1: Akima Spline Interpolation

## 5 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 sgelss (dgelss), which solve this problem using singular value decomposition.

```
call fvn_x_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 occurred.

### 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
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 .

## 6 Zero finding

fvn provide 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](http://plato.asu.edu/ftp/other/_software/muller.f).

```
call fvn_z_muller(f,eps,eps1,kn,nguess,n,x,itmax,infer,ier)
```

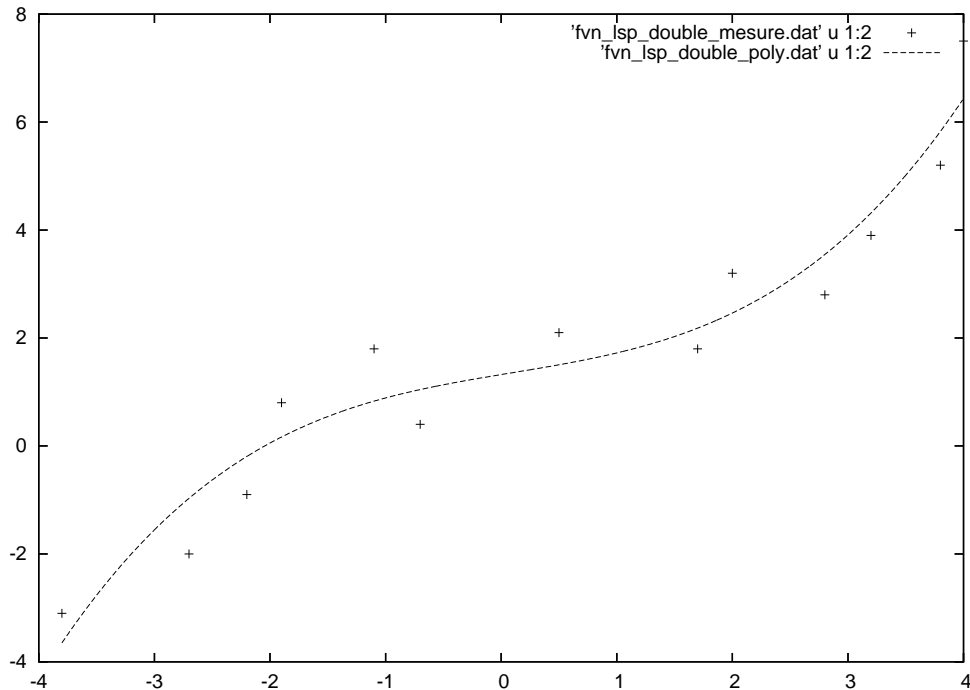


Figure 2: Least Square Polynomial

- $f$  (in) is the complex function (kind=8) for which we search zeros
- $\text{eps}$  (in) is a real(8) corresponding to the first stopping criterion : let  $f_p(z)=f(z)/p$  where  $p = (z-z(1))*(z-z(2))*\dots*(z-z(k-1))$  and  $z(1),\dots,z(k-1)$  are previously found roots. if  $((\text{cdabs}(f(z)).\text{le.eps}) .\text{and.} (\text{cdabs}(f_p(z)).\text{le.eps}))$ , then  $z$  is accepted as a root.
- $\text{eps1}$  (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  $\text{eps1}$ . Note that if either or both of the stopping criteria are fulfilled, the root is accepted.
- $\text{kn}$  (in) is an integer equal to the number of known roots, which must be stored in  $x(1),\dots,x(\text{kn})$ , prior to entry in the subroutine.
- $\text{nguess}$  (in) is the number of initial guesses provided. These guesses must be stored in  $x(\text{kn}+1),\dots,x(\text{kn}+\text{nguess})$ .  $\text{nguess}$  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  $\text{kn}+n$ .  $x(1),\dots,x(\text{kn})$  on input must contain any known roots.  $x(\text{kn}+1),\dots, x(\text{kn}+n)$  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,  $x(\text{kn}+1),\dots,x(\text{kn}+n)$  contain the approximate roots found by the subroutine.
- $\text{itmax}$  (in) is an integer equal to the maximum allowable number of iterations per root.
- $\text{infer}$  (out) is an integer vector of size  $\text{kn}+n$ . On output  $\text{infer}(j)$  contains the number of iterations used in finding the  $j$ -th root when convergence was achieved. If convergence was not obtained in  $\text{itmax}$  iterations,  $\text{infer}(j)$  will be greater than  $\text{itmax}$

- `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 satisfied, 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  $fp(z) = f(z)/((z-z(1))\dots(z-z(j-1)))$  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
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

```

## 7 Trigonometry

### 7.1 Complex Sine Arc

( only `complex(kind=8)` version )

```
y=fvn_z_asin(z)
```

This function return the complex arc sine of `z`. It is adapted from the c gsl library <http://www.gnu.org/software/gsl/>.

### 7.2 Complex Cosine Arc

( only `complex(kind=8)` version )

```
y=fvn_z_acos(z)
```

This function return the complex arc cosine of `z`. It is adapted from the c gsl library <http://www.gnu.org/software/gsl/>.

### 7.3 Real Sine Hyperbolic Arc

( only real(kind=8) version )

```
y=fvn_d_asinh(x)
```

This function return the arc hyperbolic sine of x.

### 7.4 Real Cosine Hyperbolic Arc

( only real(kind=8) version )

```
y=fvn_d_acosh(x)
```

This function return the arc hyperbolic cosine of x. In the current implementation error handling is ugly... it will stop program execution if argument is lesser than one.

## 8 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.

### 8.1 Gauss Legendre Abscissas and Weigth

This subroutine was inspired by Numerical Recipes routine gauleg.

```
call fvn_d_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 weigth

### 8.2 Gauss Legendre Numerical Integration

```
call fvn_d_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.

### 8.3 Gauss Kronrod Adaptative Integration

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



### 8.3.1 Numerical integration of a one variable function

call `fvn_d_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 integer equal to maximum number of subintervals in the partition of the given integration interval (`a,b`). A value of 500 will usually give good results.

$$\int_a^b f(x) dx \quad (3)$$

### 8.3.2 Numerical integration of a two variable function

```
call fvn_d_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 described 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  $x$ .

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

#### Example

```
program integ
  use fvn
  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
```