VOFTools

A package of routines with analytical and geometrical tools for 2D/3D VOF methods in general grids

User Manual

(Version 3.2, July 2019)

by

J. López and J. Hernández

Contents

1 Introduction

VOFTools is a package of routines with analytical and geometrical tools for 2D/3D VOF (volume of fluid) methods in general grids. The VOFTools library includes simple and efficient analytical and geometrical tools for area/volume computation, truncation operations that typically arise in VOF methods, area/volume conservation enforcement (VCE) in PLIC (piecewise linear interface calculation) reconstruction, liquid area/volume fraction initialization and computation of the distance from a given point to the reconstructed interface. Earlier versions of this library were presented in [1–4]. The present version incorporates an accurate procedure, based on recursive local grid refinement, to compute the liquid area/volume bounded by a convex polygonal/polyhedral cell and a given implicitly-defined liquid interface. The implementation details and a performance analysis can be found in [5].

The VOFTools routines are implemented in FORTRAN. To enable the routines to be used with C codes, the declarations in C of all the implemented routines are also included in the distributed software. The relevant arrays are pre-allocated for a user-specified number of faces (ns) and vertices (nv). By default, all the variables whose name starts with a letter between 'a' and 'h' or between 'o' and 'z' are considered as double precision reals and the rest are considered as regular integers. The files included in the supplied package of routines are the following:

- voftools.f: source code of the analytical and geometrical tools.
- uservoftools.f: source code of user-defined functions.
- mesh.f: definitions of different cell geometries.
- dim.h: array dimensions for FORTRAN codes.
- dimc.h: array dimensions for C codes.
- cvoftools.h: declaration of the subroutines of the VOFTools library to be used in C codes.
- cuservoftools.h: declaration of the user-defined functions included in the uservoftools.f file to be used in C codes.
- cmesh.h: declaration of the subroutines used to define the cell geometries considered in the test programs, to be used in C codes.

- `test2d.f`: 2D test program in FORTRAN.
- `test2d.c`: 2D test program in C.
- `test3d.f`: 3D test program in FORTRAN.
- `test3d.c`: 3D test program in C.
- `Makefile`: constructs the VOFTools library and makes executable files for test programs in C and FORTRAN.
- `change.log`: list of notable changes made to the VOFTools package.
- COPYNG: copy of the GNU General Public License, version 3.
- `user-manual.pdf`: this user manual in pdf format.
- `readme.txt`: names and a brief description of all the files that make up the package and instructions on the installation and execution of the test programs.

2 Installation

To install the VOFTools library and execute the test programs supplied in FORTRAN and C, perform the following steps:

1. Decompress the downloaded package in the working directory:

   ```
   tar -zxvf voftools-3.2.tgz
   ```

2. Go to the VOFTools directory:

   ```
   cd voftools-3.2
   ```

3. Edit the `Makefile` file and choose the compilers (COMPILER = 1 for GNU compilers and 2 for Intel compilers). The user can introduce other compilers by setting variables CC, F77 and LIBS.

4. Build the VOFTools library (`libvoftools.a`):

   ```
   make
   ```

5. Type

```
make all
```

to compile the four versions of the test program (`test2d_c`, `test2d_f`, `test3d_c` and `test3d_f`).

Optionally, the built `VOFTools` library can be moved to a search path (for example, `/usr/lib/`) and the test program (`testprogram.f`) can be compiled as follows:

```
sudo cp libvoftools.a /usr/lib
sudo chmod 777 /usr/lib/libvoftools.a
ifort -o testprogram testprogram.f -lvoftools
```

6. Execute the test program(s).

If the user needs to modify the values of parameters `ns` or `nv` in the `dim.h` file for the `FORTRAN` version of the code or in the `dimc.h` file for the `C` version of the code, for example in order to use cells with a number of faces or vertices higher than that initially specified in the above files, the `VOFTools` library must be recompiled. To recompile the `VOFTools` library in the same directory, type first

```
make clean
```

and then proceed as indicated from step 4.

3 Routines description and usage

In the routines described below, the structure of a given polyhedron is arranged using the following parameters:

`ipv`:	array of dimensions `(ns,nv)`, which stores the global indices of the polyhedron vertices
`nipv`:	array of dimension `ns`, which stores the number of vertices of each face
`ntp`:	last global vertex index
`nts`:	total number of faces
`ntv`:	total number of vertices (note that, if the polyhedron has not been truncated previously, then `ntp=ntv`)
`vertp`:	array of dimensions `(nv,3)`, which stores the coordinates of the polyhedron vertices
`xns, yns, zns`:	arrays of dimension `ns`, which store the components of the unit-length vectors normal to the faces of the polyhedron

In 2D, parameters `ipv(nv)`, `ntp`, `ntv` and `vertp(nv,2)` define the structure of a given polygon in a similar way.

In the supplied `mesh.f` file, different polyhedra and polygons (Table 1 summarizes the available geometries) can be set by calling the indicated routines, which return the parameters that define the structure of polyhedra or polygons, as described above. As an example, the calling conventions for a cubic cell in `FORTRAN` and `C` are, respectively,

```
call cubicmesh(ipv,nipv,ntp,nts,ntv,vertp,xns,yns,zns)
```

```
cubicmesh_(ipv,nipv,&ntp,&nts,&ntv,vertp,xns,yns,zns);
```

and, for a square cell,

```
call squaremesh(ipv,ntp,ntv,vertp)
```

```
squaremesh_(ipv,&ntp,&ntv,vertp);
```

In the following, the input and output arguments and the calling convention of each routine implemented in the `VOFTools` package are described in detail. In some of the routines, an additional numerical character has been added to the name of some of the parameters defined at the beginning of this section (for example, `ipv1`) to denote a value previous to, or obtained from, a certain operation.

Table 1: Cell geometries included in the `mesh.f` file

Routine name	Cell geometry
	3D
`cubicmesh`	Cubic
`hexahemesh`	Irregular hexahedron
`tetramesh`	Tetrahedron
`dodecamesh`	Dodecahedron
`icosamesh`	Icosahedron
`complexmesh`	Complex polyhedron with 32 vertices and 18 faces
	2D
`squaremesh`	Square
`hexagomesh`	Regular hexagon
`trianglemesh`	Irregular triangle
`quadranglemesh`	Irregular quadrangle
`pentagonmesh`	Irregular pentagon
`hexagonmesh`	Irregular hexagon

3.1 enforv3d

Solves the local volume conservation enforcement problem in 3D, and is invoked in FORTRAN and C as follows:

```
      call enforv3d(c,ipv,nipv,ntp,nts,ntv,v,vt,vertp,xnc,
     -              xns,ync,yns,znc,zns)
```

```
enforv3d_(&c,ipv,nipv,&ntp,&nts,&ntv,&v,&vt,vertp,&xnc,xns,
&ync,yns,&znc,zns);
```

where the arguments are

- On entry:

ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns:	parameters that define the structure of the polyhedron
v:	liquid volume
vt:	total volume of the polyhedron
xnc, ync, znc:	components of the unit-length vector normal to the interfacial plane

- On return:

c:	solution of the VCE problem

3.2 enforv3dsz

Solves the local volume conservation enforcement problem for rectangular parallelepipedic cells, such as that defined in the cubicmesh routine, using the efficient analytical method of Scardovelli and Zaleski [6], which was proposed to be used specifically with this type of cell:

```
      call enforv3dsz(c,dx,dy,dz,v,vertp,xnc,ync,znc)
```

```
enforv3dsz_(&c,&dx,&dy,&dz,&v,vertp,&xnc,&ync,&znc);
```

Arguments:

- On entry:

`dx, dy, dz`:	cell dimensions
`vertp`:	array of vertex coordinates
`v`:	liquid volume
`xnc, ync, znc`:	components of the unit-length vector normal to the interfacial plane

- On return:

`c`:	solution of the VCE problem

3.3 newpol3d

Rearranges the vertices of a truncated polyhedron:

```
      call newpol3d(ia,ipia0,ipia1,ipv,iscut,nipv,ntp,nts,
     -              ntv,xnc,xns,ync,yns,znc,zns)
```

```
newpol3d_(ia,ipia0,ipia1,ipv,iscut,nipv,&ntp,&nts,&ntv,
&xnc,xns,&ync,yns,&znc,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters that define the structure of the original polyhedron
`xnc, ync, znc`:	components of the unit-length vector normal to the interface plane
`ia`:	array of dimension `nv` that, for each original polyhedron vertex, stores a value of 0 if the normal to the interface plane points out from the vertex and 1 otherwise

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters that define the structure of the truncated polyhedron (note that the parameters of the original polyhedron are replaced by those of the truncated polyhedron, as also occurs in the routine `inte3d` described below)
`ipia0`, `ipia1`:	arrays of dimension `nv` that store the global indices of the original polyhedron vertices, for which `ia`=0 and `ia`=1, respectively, that are located on the edge containing the corresponding intersection point
`iscut`:	array of dimension `ns`, whose elements are equal to 1 if the face is truncated and 0 otherwise

3.4 `inte3d`

Performs the intersection between a generic convex polyhedron and a plane:

```
      call inte3d(c,icontn,icontp,ipv,nipv,ntp,nts,ntv,vertp,
     - xnc,xns,ync,yns,znc,zns)
```

```
inte3d_(&c,&icontn,&icontp,ipv,nipv,&ntp,&nts,&ntv,vertp,
&xnc,xns,&ync,yns,&znc,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the original polyhedron
`xnc, ync, znc`:	components of the unit-length vector normal to the interface plane
`c`:	constant of the truncating plane

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the truncated polyhedron
`icontn`:	number of vertices of the original polyhedron outside the truncated region
`icontp`:	number of vertices of the original polyhedron that remain in the truncated region

3.5 toolv3d

Computes the volume of a polyhedron:

```
call toolv3d(ipv,nipv,nts,vertp,vol,xns,yns,zns)
```

```
toolv3d_(ipv,nipv,&nts,vertp,&vol,xns,yns,zns);
```

Arguments:

- On entry:

`ipv, nipv, nts, vertp, xns, yns, zns`:	polyhedron parameters

- On return:

`vol`:	polyhedron volume

3.6 cppol3d

Makes a copy of the structure of a polyhedron:

```
      call cppol3d(cs,cs0,ipv,ipv0,nipv,nipv0,ntp,ntp0,nts,
     - nts0,ntv,ntv0,vertp,vertp0,xns,xns0,yns,yns0,zns,zns0)
```

```
cppol3d_(cs,cs0,ipv,ipv0,nipv,nipv0,&ntp,&ntp0,&nts,&nts0,
&ntv,&ntv0,vertp,vertp0,xns,xns0,yns,yns0,zns,zns0);
```

Arguments:

- On entry:

`ipv0, nipv0,` `ntp0, nts0,` `ntv0,` `vertp0,` `xns0, yns0,` `zns0`:	parameters of the original polyhedron
`cs0`:	array of dimension `ns` that stores the positions of the planes containing the faces of the original polyhedron

- On return:

`ipv, nipv,` `ntp, nts,` `ntv, vertp,` `xns, yns,` `zns`:	parameters of the copied polyhedron
`cs`:	copy of `cs0`

3.7 restore3d

Restores the structure of a polyhedron by renumbering consecutively the faces and values of the global vertex index of the polyhedron:

```
      call restore3d(cs,ipv,nipv,ntp,nts,ntv,vertp,xns,yns,
     - zns)
```

```
restore3d_(cs,ipv,nipv,&ntp,&nts,&ntv,vertp,xns,yns,zns);
```

Arguments:

- On entry:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the original polyhedron
`cs`:	array of dimension `ns` that stores the constants of the planes containing the faces of the original polyhedron

- On return:

`ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns`:	parameters of the restored polyhedron
`cs`:	array of dimension `ns` that stores the constants of the planes containing the faces of the restored polyhedron

Note that the parameters of the original polyhedron are replaced by those of the restored polyhedron.

3.8 dist3d

Computes the distance from a point P to a general convex polygon in 3D:

```
call dist3d(d,n,x,y,z,xp,yp,zp)
```

```
dist3d_(&d,&n,x,y,z,&xp,&yp,&zp);
```

Arguments:

- On entry:

`n`:	number of vertices of the polygon
`x, y, z`:	arrays of dimension `nv` that store the coordinates of the polygon vertices
`xp, yp, zp`:	coordinates of point P

- On return:

  d: distance from point P to the polygon

3.9 initf3d

Computes the fraction of the liquid volume contained in a cell:

```
      call initf3d(func3d,ipv,nc,nipv,ntp,nts,ntv,tol,vertp,vf,
     - xns,yns,zns)
```

```
initf3d_(func3d_,ipv,&nc,nipv,&ntp,&nts,&ntv,&tol,vertp,&vf,
xns,yns,zns);
```

Arguments:

- On entry:

  func3d: user-defined implicit function, included in the uservoftools.f file, that defines the liquid body shape

  ipv, nipv, ntp, nts, ntv, vertp, xns, yns, zns: parameters of the original polyhedron

  nc: number of divisions along each coordinate axis of the superimposed cell box

  tol: prescribed positive tolerance for the distance to the interface of the liquid body (a sufficiently high value must be picked to ensure the application of the recursive refinement procedure to any interfacial cell ($0 < \texttt{vf} < 1$) with all its vertices on one side of the liquid body interface)

- On return:

  vf: liquid volume fraction

3.10 enforv2d

Solves the local volume conservation enforcement problem in 2D:

```
call enforv2d(c,ipv,ntp,ntv,v,vt,vertp,xnc,ync)
```

```
enforv2d_(&c,ipv,&ntp,&ntv,&v,&vt,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the polygonal cell
`v`:	liquid area
`vt`:	total area of the polygonal cell
`xnc, ync`:	components of the unit-length vector normal to the interfacial line

- On return:

`c`:	solution of the VCE problem

3.11 enforv2dsz

Solves the local volume conservation enforcement problem for rectangular cells, such as that defined in the `squaremesh` routine, using the efficient analytical method of Scardovelli and Zaleski [6]:

```
call enforv2dsz(c,dx,dy,v,vertp,xnc,ync)
```

```
enforv2dsz_(&c,&dx,&dy,&v,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`dx, dy`:	cell dimensions
`vertp`:	array of vertex coordinates of the cell
`v`:	liquid area
`xnc, ync`:	components of the unit-length vector normal to the interfacial line

- On return:

  `c`: solution of the VCE problem

3.12 newpol2d

Rearranges the vertices of a truncated polygon:

```
      call newpol2d(ia,ipia0,ipia1,ipv,ntp,ntv,vertp,xncut,
     - yncut)
```

```
newpol2d_(ia,ipia0,ipia1,ipv,&ntp,&ntv,vertp,xncut,yncut);
```

Arguments:

- On entry:

  `ipv, ntp, ntv, vertp`: parameters of the original polygon

  `xnc, ync`: components of the unit-length vector normal to the interfacial line

  `ia`: array of dimensions `nv` that stores a value of 0 if the normal to the interface plane points out from the vertex and 1 otherwise

- On return:

  `ipv, ntp, ntv, vertp`: parameters of the truncated polygon (note that the parameters of the original polygon are replaced by those of the truncated polygon, as also occurs in the routine `inte2d` described below)

  `xncut, yncut`: arrays of dimensions 2 that store the components of the unit-length vectors normal to the two edges cut by the interface line

  `ipia0`, `ipia1`: arrays of dimension `nv` that store the global indices of the original polygon vertices, for which `ia`=0 and `ia`=1, respectively, that are located on the edge containing the corresponding intersection point

3.13 inte2d

Performs the intersection between a generic convex polygon and a line:

```
call inte2d(c,icontn,icontp,ipv,ntp,ntv,vertp,xnc,ync)
```

```
inte2d_(&c,&icontn,&icontp,ipv,&ntp,&ntv,vertp,&xnc,&ync);
```

Arguments:

- On entry:

`ipv, ntp, ntv, vertp`:	parameters of the original polygon
`xnc, ync`:	components of the unit-length vector normal to the interfacial line
`c`:	constant of the truncating line

- On return:

`ipv, ntp, ntv, vertp`:	parameters corresponding of the truncated polygon
`icontn, icontp`:	number of vertices of the original polygon that are outside and inside the truncated region, respectively

3.14 toolv2d

Computes the area of a generic polygon:

```
call toolv2d(ipv,ntv,vertp,vol)
```

```
toolv2d_(ipv,&ntv,vertp,&vol);
```

Arguments:

- On entry:

`ipv, ntv, vertp`:	parameters of the polygon

- On return:

`vol`:	area of the polygon

3.15 cppol2d

Copies the structure of a polygon:

```
call cppol2d(ipv0,ipv,ntp0,ntp,ntv0,ntv,vertp0,vertp)
```

```
cppol2d_(ipv0,ipv,&ntp0,&ntp,&ntv0,&ntv,vertp0,vertp);
```

Arguments:

- On entry:
  `ipv0, ntp0, ntv0, vertp0`: parameters of the original polygon
- On return:
  `ipv, ntp, ntv, vertp`: parameters of the copied polygon

3.16 restore2d

Restores the structure of a polygon by renumbering consecutively the values of the global vertex index of the polygon:

```
call restore2d(ipv,ntp,ntv,vertp)
```

```
restore2d_(ipv,&ntp,&ntv,vertp);
```

Arguments:

- On entry:
  `ipv, ntp, ntv, vertp`: parameters of the original polygon
- On return:
  `ipv, ntp, ntv, vertp`: parameters of the restored polygon

Note that the parameters of the original polygon are replaced by those of the restored polygon.

3.17 dist2d

Computes the distance from point P to a segment in 2D:

```
call dist2d(d,x,y,xp,yp)
```

```
dist2d_(&d,x,y,&xp,&yp);
```

Arguments:

- On entry:

x, y:	two-element arrays that store the coordinates of the segment vertices
xp, yp:	coordinates of point P

- On return:

d:	distance from point P to the segment

3.18 initf2d

Computes the fraction of the liquid area contained in a cell:

```
call initf2d(func2d,ipv,nc,ntp,ntv,tol,vertp,vf)
```

```
initf2d_(func2d_,ipv,&nc,&ntp,&ntv,&tol,vertp,&vf);
```

Arguments:

- On entry:

func2d:	user-defined implicit function, included in the uservoftools.f file, that defines the liquid body shape
ipv, ntp, ntv, vertp:	parameters of the original polygon
nc:	number of divisions along each coordinate axis of the superimposed cell box
tol:	prescribed positive tolerance for the distance to the interface of the liquid body (for its value selection, follow the same consideration that it was recommended for the initf3d routine)

- On return:

  `vf`: liquid area fraction

4 Test program

The `VOFTools` package includes 2D and 3D versions of a test program implemented in `FORTRAN` (`test2d.f` and `test3d.f`) and `C` (`test2d.c` and `test3d.c`). In this program, all the arrays are first dimensioned, and the shape of cell to be used is assigned by calling one of the routines listed in Table 1. The shape of the liquid body, whose volume contained in the selected cell is computed by the initialization procedure, is set through the variable `ishape`, which can be 1 for circular or spherical shapes or 2 for elliptical or toroidal shapes. Then, the following operations are performed:

- Volume computation of the selected cell. The routine `toolv3d` in 3D or `toolv2d` in 2D is called, which gives the volume (area in 2D) of the cell, `vt`.
- Volume conservation enforcement (Fig. 1). An interface with orientation given by a vector normal to the interface with each component equal to $1/\sqrt{n_{dim}}$, where $n_{dim} = 2$ for 2D and 3 for 3D, is positioned in the cell to obtain a liquid volume fraction equal to 0.5. The position, given by parameter `c`, is obtained by calling the routine `enforv3d` in 3D or `enforv2d` in 2D.

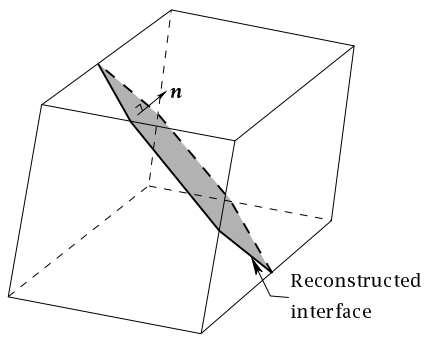

Figure 1: Positioning the interface by solving the VCE problem on a hexahedral cell.

- Volume truncation (Fig. 2). The intersection between the original cell and a plane $\mathcal{P}$ that contains the reconstructed interface is performed by calling the routine `inte3d` in 3D or `inte2d` in 2D.

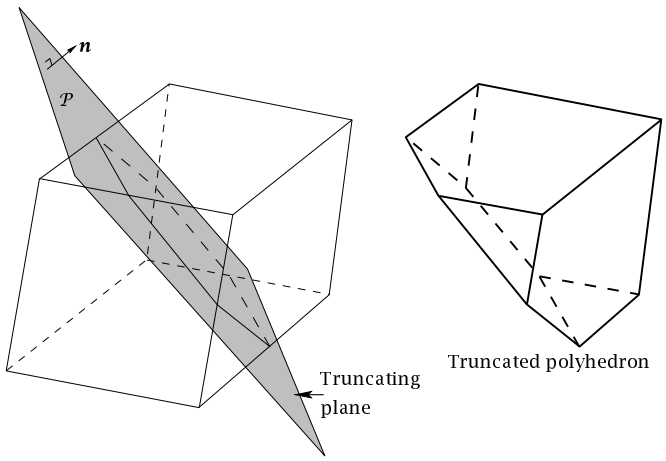

Figure 2: Truncation of the hexahedral cell by a plane containing the interface.

- Distance computation (Fig. 3). The distance, d, between a point located on the origin of the coordinate system and the new face (edge in 2D) resulting from the truncation operation is obtained by calling the routine `dist3d` in 3D or `dist2d` in 2D.

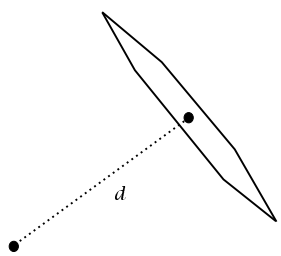

Figure 3: Computation of the distance from the origin of the coordinate system to the reconstructed interface.

- Volume/area fraction initialization (Fig. 4). The fraction of the volume/area

of a liquid body contained inside the cell is computed by calling the routine `initf3d` in 3D or `initf2d` in 2D.

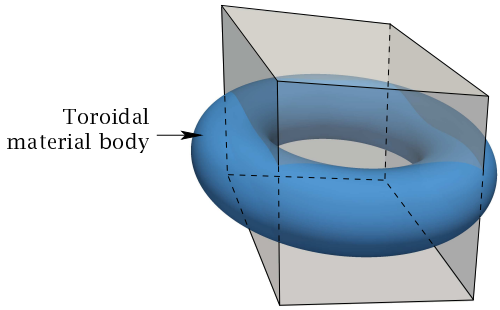

Figure 4: Initialization of the volume contained inside the hexahedral cell of a toroidal liquid body.

References

[1] J. López, J. Hernández, Analytical and geometrical tools for 3D volume of fluid methods in general grids, *J. Comput. Phys.* 227 (2008) 5939-5948.

[2] J. López, P. Gómez, J. Hernández, F. Faura, A two-grid adaptive volume of fluid approach for dendritic solidification, *Comput. Fluids* 86 (2013) 326-342. Open access,

[3] J. López, J. Hernández, P. Gómez, F. Faura, A new volume conservation enforcement method for PLIC reconstruction in general convex grids, *J. Comput. Phys.* 316 (2016) 338-359. Open access,

[4] J. López, J. Hernández, P. Gómez, F. Faura, `VOFTools` – A software package of calculation tools for volume of fluid methods using general convex grids, *Comput. Phys. Commun.* 223 (2018) 45-54 Open access,

[5] J. López, J. Hernández, P. Gómez, C. Zanzi, R. Zamora, `VOFTools 3.2`: Added VOF functionality to initialize the liquid volume fraction in general convex cells, submitted to *Comput. Phys. Commun.*

[6] R. Scardovelli, S. Zaleski, Analytical relations connecting linear interfaces and volume fractions in rectangular grids, *J. Comput. Phys.* 164 (2000) 228-237.