S O L I D G R A P H I C S
T O O L B O X

-

Popis funkcí

Petr Hora, Olga Červená

2004

Petr Hora
Olga Červená

SOLID GRAPHICS TOOLBOX

Obsah

Uživatelská část

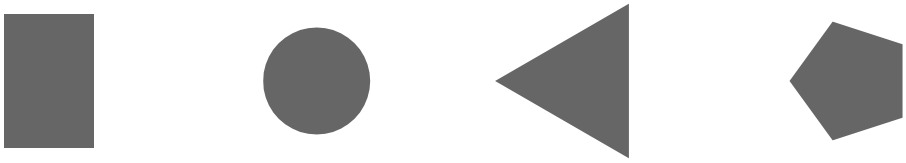
Solid graphics je toolbox MATLABu, který je určen pro snažší tvorbu a zobrazení trojrozměrných grafických objektů. Podnětem pro jeho vytvoření byla vizualizace výsledků získaných ze simulací v molekulární dynamice.

Funkce **Solid graphics**u (sol-funkce) lze rozdělit do tří skupin. Do první z nich lze zařadit funkce, které generují objekty v základní poloze a velikosti, tzv. sol-objekty. Druhá skupina funkcí objekty slučuje, mění jejich velikost, otáčí a posouvá do požadované polohy. Poslední skupinu tvoří dvě funkce, kterými je možné objektům přiřadit dodatečné vlastnosti (např.: barvu, průhlednost, ...) a zobrazit je.

Tvorba sol-objektů

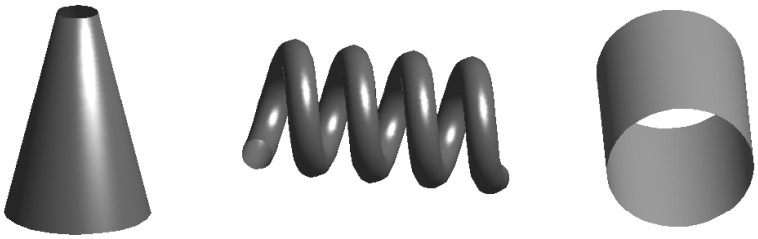
Funkce generující objekty lze rozdělit do několika skupin:

- Funkce generující dvojrozměrné sol-objekty:
  `sol_rectangle` a `sol_circle`.
  Volbou malé hodnoty vstupního parametru `n` lze funkci `sol_circle` použít též ke generování n-úhelníků.



- Funkce generující trojrozměrné sol-objekty:
  `sol_cuboid`, `sol_ellipsoid`, `sol_polyhedra`, `sol_ring`, `sol_ringext` a `sol_sphere`.

- Funkce generující "pláště" těles: `sol_conus`, `sol_cylinder` a `sol_spiral`.



- Funkce převádějící "patch" nebo "surface" grafické objekty MATLABu na sol-objekt: `sol_surface2obj` a `sol_patch2obj`.

Sol-objekt je v MATLABu reprezentován buňkovým polem struktur. Obsah dané stuktury závisí na typu generovaného sol-objektu, viz tabulka. Pokud je sol-objekt modifikován příkazem `sol_set`, bude struktura obsahovat navíc záznam 'flag'.

	rectangle	circle	cuboid	ellipsoid	polyhedra	ring	ringext	sphere	conus	cylinder	spiral	patch2obj	surface2obj
label	√	√	√	√	√	√	√	√	√	√	√	√	√
xdata				√		√	√	√	√	√	√		√
ydata				√		√	√	√	√	√	√		√
zdata				√		√	√	√	√	√	√		√
vertices	√	√	√		√							√	
faces	√	√	√		√							√	
normals	√	√	√	√	√	√	√	√	√	√	√	√	√
lx	√		√										
ly	√		√										
lz			√										
rx				√		√							
ry				√		√							
rz				√									
radius		√			√	√	√	√		√			
radius1									√				
radius2									√				
radius_tube											√		
radius_spiral											√		
height									√	√	√		
base1									√	√	√		
base2									√	√	√		
origin	√	√	√	√	√	√	√	√	√	√	√	√	√

Modifikace sol-objektů

Objekty se generují v základní poloze, tj. se středem v počátku kartézského souřadného systému. Modifikaci jejich polohy a velikosti lze dosáhnout následujícími sol-funkcemi:

- `sol_translate`,
- `sol_rotateX`,
- `sol_rotateY`,
- `sol_rotateZ`,
- `sol_rotate`,
- `sol_scale`,
- `sol_combine`.

Zobrazení sol-objektů

Pro dodatečnou úpravu sol-objektů lze použít funkci

- `sol_set`

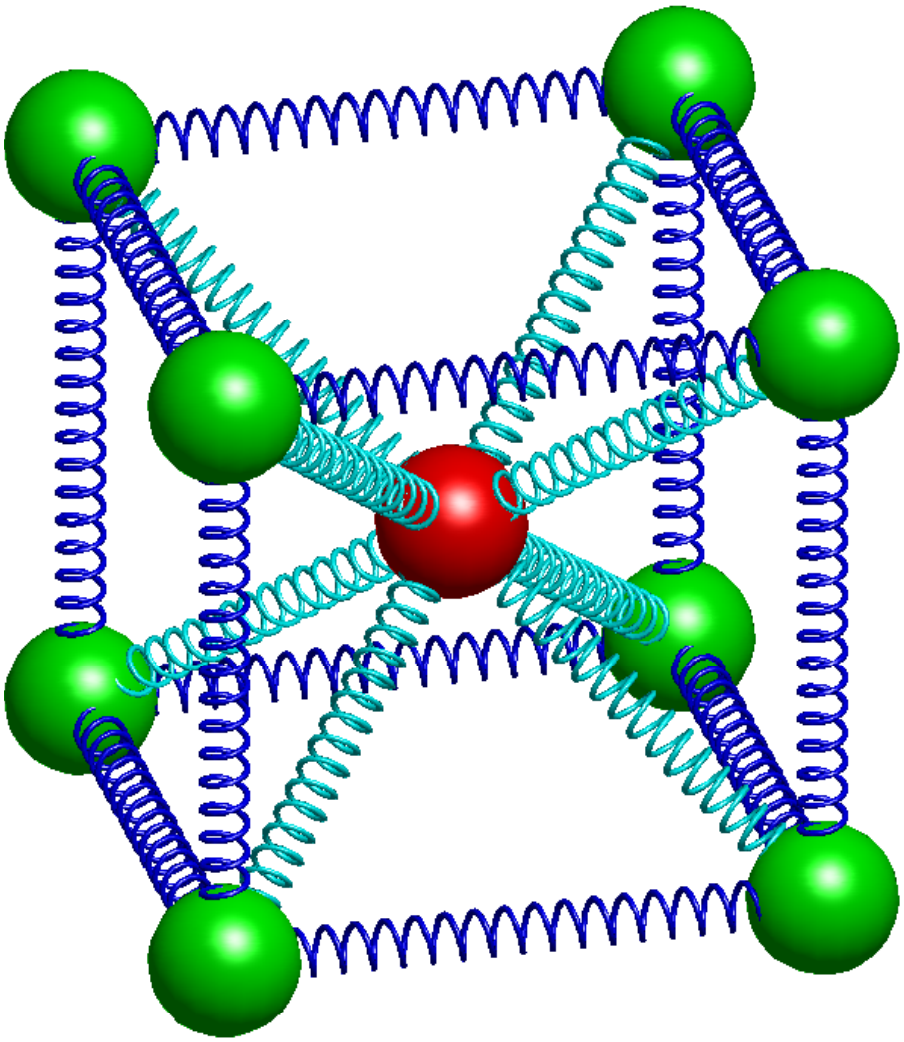
a scénu na závěr vykreslíme funkcí

- `sol_render`.

Obsah toolboxu

Soubor	Poznámka
Contents.m	
sol_circle.m	Generování kruhu.
sol_rectangle.m	Generování obdélníku.
sol_cuboid.m	Generování kvádru.
sol_ellipsoid.m	Generování elipsoidu.
sol_polyhedra.m	Generování jednoho z pěti pravidelných mnohostěnů.
sol_ring.m	Generování prstence.
sol_ringext.m	Generování prstence s různými profily.
sol_sphere.m	Generování koule.
sol_conus.m	Generování komolého kužele.
sol_cylinder.m	Generování válce.
sol_spiral.m	Generování spirály.
sol_patch2obj.m	Změna grafického objektu MATLABu patch na sol-objekt.
sol_surface2obj.m	Změna grafického objektu MATLABu surface na sol-objekt.
sol_combine.m	Slučování sol-objektů.
sol_rotate.m	Otáčení sol-objektů.
sol_rotateX.m	Otáčení sol-objektů okolo osy X.
sol_rotateY.m	Otáčení sol-objektů okolo osy Y.
sol_rotateZ.m	Otáčení sol-objektů okolo osy Z
sol_scale.m	Změna velikosti sol-objektů.
sol_translate.m	Posunutí sol-objetů.
sol_set.m	Nastavení dodatečných vlastností sol-objektů.
sol_render.m	Zobrazování sol-objektů.
demo/demo1.m	Hlava s kloboukem.
demo/demo2.m	Prostorový kříž.
demo/demo3.m	Krychle s pozadím.
demo/demo4.m	Věnec.
demo/demo5.m	Dvě poloprůhledné koule spojené pružinou.
demo/demo6.m	Model bcc-krystalu, atomy spojené válcem.
demo/demo7.m	Model bcc-krystalu, atomy spojené pružinou.
demo/demo8.m	Dvanáctistěn (vrcholy koule, hrany pružiny).
demo/demo9.m	Animace, koule na pružině.
demo/demo10.m	Všech pět pravidelných mnohostěnů.
demo/demo11.m	Animace, rotace Země kolem své osy.
demo/earth_2000.mat	Obrázek Země pro demo11.
private/xy_circle.m	Funkce řídicí křivky pro sol_ringext.
private/xy_ellipse.m	Funkce řídicí křivky pro sol_ringext.

Referenční část



sol_circle

Funkce Generování kruhu.

Syntaxe
```
circle=sol_circle
circle=sol_circle(radius)
circle=sol_circle(radius,n)
```

Popis `sol_circle(radius,n)` generuje kruh o poloměru `radius`. Počet bodů po obvodu kruhu je dán parametrem `n`.

`sol_circle(radius)` použije implicitní hodnotu `n=20`,
`sol_circle` bez vstupních hodnot použije implicitní hodnoty `n=20` a `radius=1`, generuje jednotkový kruh.

`sol_circle` vrací sol-objekt `circle`.

Příklady
```
sol_circle(2,40)
```

Viz též `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_combine

Funkce Slučování sol-objektů.

Syntaxe `objOut=sol_combine(varargin);`

Popis `sol_combine(varargin)` slučuje sol-objekty. Vstupem je seznam sol-objektů.

`sol_combine` vrací sol-objekt `objOut`.

Příklady
```
cuboid1=sol_cuboid(1,1,7);
cuboid2=sol_cuboid(7,1,1);
cross=sol_combine(cuboid1,cuboid2);
```

Viz též `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_set`, `sol_translate`.

sol_conus

Funkce Generování komolého kužele.

Syntaxe
```
conus=sol_conus
conus=sol_conus(radius1)
conus=sol_conus(radius1,radius2)
conus=sol_conus(radius1,radius2,height)
conus=sol_conus(radius1,radius2,height,n)
```

Popis `sol_conus(radius1,radius2,height,n)` generuje komolý kužel o poloměrech podstav `radius1`, `radius2` a výšce `height` (výšku kužele lze zadat i pomocí dvou bodů ve tvaru matice `[2x3]: [x1 y1 z1;x2 y2 z2]`). Počet bodů podél kruhových podstav kužele je dán parametrem `n`.

`sol_conus(radius1,radius2,height)` použije implicitní hodnotu `n=20`.
`sol_conus(radius1,radius2)` použije implicitní hodnoty `n=20` a `height=1`.
`sol_conus(radius1)` použije implicitní hodnoty `n=20`, `height=1` a `radius2=0`.
`sol_conus` bez vstupních parametrů použije implicitní hodnoty `n=20`, `height=1`, `radius2=0` a `radius1=1`.

`sol_conus` vrací sol-objekt `conus`.

Příklady
```
conus1=sol_conus(3,2,10,40);
conus2=sol_conus(3,2,[0 0 0;2 11 5],40);
```

Viz též `sol_circle`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_cuboid

Funkce Generování kvádru.

Syntaxe
```
cuboid=sol_cuboid
cuboid=sol_cuboid(lx)
cuboid=sol_cuboid(lx,ly,lz)
```

Popis `sol_cuboid(lx,ly,lz)` generuje kvádr o rozměrech `lx`, `ly` a `lz`.
`sol_cuboid(lx)` generuje krychli o velikosti hrany `lx`.
`sol_cuboid` bez vstupních parametrů generuje jednotkovou krychli.

`sol_cuboid` vrací sol-objekt `cuboid`.

Příklady
```
cuboid=sol_cuboid(2,3,4);
cuboid=sol_cuboid(3);
```

Viz též `sol_circle`, `sol_conus`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_cylinder

Funkce Generování válce.

Syntaxe
```
cylinder=sol_cylinder
cylinder=sol_cylinder(radius)
cylinder=sol_cylinder(radius,height)
cylinder=sol_cylinder(radius,height,n)
```

Popis `sol_cylinder(radius,height,n)` generuje válec o poloměru podstavy `radius` a výšce `height` (výšku válce lze zadat i pomocí dvou bodů ve tvaru matice `[2x3]: [x1 y1 z1;x2 y2 z2]`). Počet bodů podél podstavy válce je dán parametrem `n`.

`sol_cylinder(raduis,height)` použije implicitní hodnotu `n=20`.
`sol_cylinder(raduis)` použije implicitní hodnoty `n=20` a `height=1`.
`sol_cylinder` bez vstupních parametrů použije implicitní hodnoty `n=20`, `height=1` a `radius=1`.

`sol_cylinder` vrací sol-objekt `cylinder`.

Příklady
```
cylinder1=sol_cylinder(2,10,30);
cylinder2=sol_cylinder(2,[0 0 0;12 3 5],30);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_ellipsoid

Funkce Generování elipsoidu.

Syntaxe
```
ellipsoid=sol_ellipsoid(rx,ry,rz)
ellipsoid=sol_ellipsoid(rx,ry,rz,n)
```

Popis `sol_ellipsoid(rx,ry,rz,n)` generuje elipsoid o poloosách `rx`, `ry` a `rz`. Dělení (počet poledníků a rovnoběžek) elipsoidu je určeno parametrem `n`.

`sol_ellipsoid(rx,ry,rz)` užije implicitní hodnotu `n=20`.

`sol_ellipsoid` vrací sol-objekt `ellipsoid`.

Příklady
```
ellipsoid=sol_ellipsoid(3,4,5,40);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_patch2obj

Funkce Změna grafického objektu MATLABu patch na sol-objekt.

Syntaxe
```
objOut=sol_patch2obj(v,f,n);
objOut=sol_patch2obj(v,f,n,origin);
```

Popis `sol_patch2obj(v,f,n,origin)` mění grafický objekt MATLABu patch na sol-objekt.

Není-li zadán vstupní parametr `origin`, počátek se vypočte jako průměr hodnot `v`.

`sol_patch2obj` vrací sol-objekt `objOut`.

Příklady
```
[x,y,z,v]=flow;
p=isosurface(x,y,z,v,-3);
h=patch(p);
isonormals(x,y,z,v,h)
n=get(h,'vertexnormals');
a=sol_patch2obj(p.vertices,p.faces,n);
a{1}
```

Viz též `sol_surface2obj`.

sol_polyhedra

Funkce Generování jednoho z pěti pravidelných mnohostěnů.

Syntaxe

```
polyhedron=sol_polyhedra(ptype)
[polyhedron,vertices]=sol_polyhedra(ptype)
[polyhedron,vertices,faces]=sol_polyhedra(ptype)
polyhedron=sol_polyhedra(ptype,radius)
[polyhedron,vertices]=sol_polyhedra(ptype,radius)
[polyhedron,vertices,faces]=sol_polyhedra(ptype,radius)
```

Popis `sol_polyhedra(ptype,radius)` generuje pravidelný mnohostěn určený parametrem `ptype` (`tetrahedron` - čtyřstěn, `hexahedron` - šestistěn, `octahedron` - osmistěn, `dodecahedron` - dvanáctistěn, `icosahedron` - dvacetistěn) vepsaný do kole o poloměru `radius`.

`sol_polyhedra(ptype)` generuje mnohostěn vepsaný do jednotkové koule.

`sol_polyhedra` vrací sol-objekt `polyhedron`, matici vrcholů `vertices` a matici stěn `faces`.

Příklady

```
d=sol_polyhedra('dodecahedron',3);
[h,vertices]=sol_polyhedra('hexahedron');
[t,v,f]=sol_polyhedra('tetrahedron');
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_rectangle

Funkce Generování obdélníku.

Syntaxe
```
rectangle=sol_rectangle
rectangle=sol_rectangle(lx)
rectangle=sol_rectangle(lx,ly)
```

Popis `sol_rectangle(lx,ly)` generuje obdélník o velikostech stran `lx` a `ly`.

`sol_rectangle(lx)` generuje čtverec o straně `lx`,
`sol_rectangle` generuje jednotkový čtverec.

`sol_rectangle` vrací sol-objekt `rectangle`.

Příklady
```
rectangle=sol_rectangle(2,3);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_ring`, `sol_ringext`, `sol_sphere`, `sol_spiral`.

sol_render

Funkce Zobrazování sol-objektů.

Syntaxe
```
h=sol_render(objIn)
h=sol_render(objIn,mode)
h=sol_render(objIn,mode,options)
```

Popis `sol_render(objIn,mode,options)` zobrazuje sol-objekt `objIn`. Parametr `mode` určuje, způsob zobrazení (`'solid'` nebo `'wire'`). Je-li parametr `options` různý od nuly, pak se objekt zobrazí v 3D náhledu, osy budou mít stejná měřítka a nebudou zobrazeny `(view(3); axis equal off)`.

`sol_render(objInobjIn,mode)` použije implicitní hodnotu `options=1`, `sol_render(objIn)` použije implicitních hodnot `mode='solid'` a `options=1`.

`sol_render` vrací identifikátory grafických objektů MATLABu.

Příklady
```
hCyl=sol_render(sol_cylinder(1.5,3),'wire');
hCube=sol_render(sol_cuboid);
sol_render(sol_spiral(0.2,3,4,4,[30,120]),'solid',0);
```

Viz též `sol_combine`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_set`, `sol_translate`.

sol_ring

Funkce Generování prstence.

Syntaxe `ring=sol_ring(rx,ry,radius,[p q])`

Popis `sol_ring(rx,ry,radius,[p q])` generuje prstenec vzniklý rotací elipsy o poloosách `rx` a `ry` kolem osy ležící v rovině elipsy ve vzdálenosti `radius` od středu elipsy. Počet bodů podél řídicí elipsy je dán parametrem `p` a počet bodů podél prstence parametrem `q`.

`sol_ring` vrací sol-objekt `ring`.

Příklady
```
ring=sol_ring(1,2,8,[40 40]);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ringext`, `sol_sphere`, `sol_spiral`, `matlab\demos\cruller.m`.

sol_ringext

Funkce Generování prstence s různými profily.

Syntaxe `ringext=sol_ringext(xy,[a b],[radius twist revs],[p q],varargin)`

Popis `sol_ringext(xy,[a b],[radius twist revs],[p q],varargin)` generuje prstenec vzniklý rotací řídicí křivky popsané funkcí `xy` kolem osy ležící v rovině řídicí křivky ve vzdálenosti `radius` od středu řídicí křivky, která může rotovat kolem svého středu.

Funkce `xy` musí mít následující tvar: `[xt,yt]=xy(t,P1,P2,...)`. Parametry `P1,P2...` jsou dány `varargin`. Definiční interval parametru `t` řídicí křivky (parametr funkce `xy`) leží v intervalu `[a b]`. Parametr `twist` určuje počet rotací řídicí křivky kolem svého středu, `revs` určuje počet rotací řídicí křivky kolem osy prstence. Počet bodů podél řídicí křivky je dán parametrem `p` a počet bodů podél prstence parametrem `q`.

`sol_ringext` vrací sol-objekt `ringext`.

Příklady

```
t1=sol_ringext('ellipse',[0 2*pi],[8 4 1],[80 80],1,2,1,2);
t2=sol_ringext('circle',[0 2*pi],[8 4 1],[80 80],1,-2,2);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_sphere`, `sol_spiral`, `matlab\demos\cruller.m`.

sol_rotate

Funkce Otáčení objektu.

Syntaxe
```
objOut=sol_rotate(objIn,azel,alpha)
objOut=sol_rotate(objIn,azel,alpha,origin)
```

Popis `sol_rotate(objIn,azel,alpha,origin)` otáčí sol-objekt `objIn` o úhel `alpha` (proti směru otáčení hodinových ručiček) okolo zadaného bodu `origin` vzhledem ke směrovému vektoru `azel`. Směrový vektor může být určen ve sférických souřadnicích `[theta,phi]` nebo v kartézských souřadnicích `[x y z]`. `Theta` je úhel, který svírá průmět směrového vektoru na plochu `xy` s kladnou částí osy `X`. Měřeno proti směru otáčení hodinových ručiček od kladné části osy `X`. `Phi` je elevace směrového vektoru od plochy `xy`. (Viz též `SPH2CART`).
Všechny úhly jsou ve stupních.

`sol_rotate(objIn,azel,alpha)` použije `origin=[0 0 0]`.

`sol_rotate` vrací sol-objekt `objOut`.

Příklady
```
a=sol_rotate(sol_cuboid(1,1,2),[45 60],90,[0 0 3]);
b=sol_rotate(sol_cylinder(1,2),[1 1 1],90);
```

Viz též `sol_combine`, `sol_render`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_set`, `sol_translate`.

sol_rotateX

Funkce Otáčení objektu okolo osy `X`.

Syntaxe
```
objOut=sol_rotateX(objIn,alpha)
objOut=sol_rotateX(objIn,alpha,origin)
```

Popis `sol_rotateX(objIn,alpha,origin)` otáčí sol-objekt `objIn` o úhel `alpha` (proti směru otáčení hodinových ručiček) okolo daného bodu `origin` a osy `X`.
Úhel otáčení je ve stupních.

`sol_rotateX(objIn,alpha)` použije `origin=objOut{i}.origin`.

`sol_rotateX` vrací sol-objekt `objOut`.

Příklady
```
a=sol_rotateX(sol_cuboid(1,1,2),60,[0 0 3]);
b=sol_rotateX(sol_cylinder(1,3),90);
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_set`, `sol_translate`.

sol_rotateY

Funkce Otáčení objektu okolo osy `Y`.

Syntaxe
```
objOut=sol_rotateY(objIn,alpha)
objOut=sol_rotateY(objIn,alpha,origin)
```

Popis `sol_rotateY(objIn,alpha,origin)` otáčí sol-objekt `objIn` o úhel `alpha` (proti směru otáčení hodinových ručiček) okolo daného bodu `origin` a osy `Y`.
Úhel otáčení je ve stupních.

`sol_rotateY(objIn,alpha)` použije `origin=objOut{i}.origin`.

`sol_rotateY` vrací sol-objekt `objOut`.

Příklady
```
a=sol_rotateY(sol_cuboid(1,1,2),60,[0 0 3]);
b=sol_rotateY(sol_cylinder(1,3),90);
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateZ`, `sol_scale`, `sol_set`, `sol_translate`.

sol_rotateZ

Funkce Otáčení objektu okolo osy `Z`.

Syntaxe
```
objOut=sol_rotateZ(objIn,alpha)
objOut=sol_rotateZ(objIn,alpha,origin)
```

Popis `sol_rotateZ(objIn,alpha,origin)` otáčí sol-objekt `objIn` o úhel `alpha` (proti směru otáčení hodinových ručiček) okolo daného bodu `origin` a osy `Z`.
Úhel otáčení je ve stupních.

`sol_rotateZ(objIn,alpha)` použije `origin=objOut{i}.origin`.

`sol_rotateZ` vrací sol-objekt `objOut`.

Příklady
```
a=sol_rotateZ(sol_cuboid(1,1,2),60,[0 0 3]);
b=sol_rotateZ(sol_cylinder(1,3),90);
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_scale`, `sol_set`, `sol_translate`.

sol_scale

Funkce Změna velikosti sol-objektu.

Syntaxe `objOut=sol_scale(objIn,x)`

Popis `sol_scale(objIn,x)` mění velikost sol-objektu `objIn` v poměru `x` vzhledem k původní velikosti.

`sol_scale` vrací sol-objekt `objOut`.

Příklady
```
triple_cube=sol_scale(sol_cuboid,3);
half_sphere=sol_scale(sol_sphere,0.5);
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_set`, `sol_translate`.

sol_set

Funkce Nastavení dodatečných vlastností sol-objektu.

Syntaxe `objOut=sol_set(objIn,varargin)`

Popis `sol_set(objIn,varargin)` nastavuje dodatečné vlastnosti sol-objektu `objIn` zadané parametrem `varargin`. Parametr `varargin` je tvořen páry "název vlastnosti - hodnota vlastnosti", které je možno použít při modifikování grafického objektu MATLABu `patch` (`sol_patch2obj`, `sol_circle`, `sol_rectangle`, `sol_cuboid` a `sol_polyherda`) nebo `surface` (zbývající sol-objekty).

`sol_set` vrací sol-objekt `objOut`.

Příklady
```
ball=sol_sphere; cube=sol_cuboid(3);
red_ball=sol_set(ball,'FaceColor',[1 0 0],'EdgeColor',[1 1 0]);
transparent_cube=sol_set(cube,'FaceAlpha',0.2);
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_translate`.

sol_sphere

Funkce Generování koule.

Syntaxe
```
sphere=sol_sphere
sphere=sol_sphere(radius)
sphere=sol_sphere(radius,n)
```

Popis `sol_sphere(radius,n)` generuje kouli o poloměru `radius`. Dělení (počet poledníků a rovnoběžek) koule je určeno parametrem `n`.

`sol_sphere(radius)` použije implicitní hodnotu `n=20`.
`sol_sphere` bez vstupních hodnot použije implicitní hodnoty `n=20` a `radius=1`, generuje jednotkovou kouli.

`sol_sphere` vrací sol-objekt `sphere`.

Příklady
```
sphere=sol_sphere(3,40)
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_spiral`.

sol_spiral

Funkce Generování spirály.

Syntaxe `spiral=sol_spiral(Rt,Rs,Hs,Ns,[m n])`

Popis `sol_spiral(Rt,Rs,Hs,Ns,[m n])` generuje spirálu o poloměru `Rs`, výšce `Hs` (výšku spirály lze zadat i pomocí dvou bodů ve tvaru matice `[2x3]`: `[x1 y1 z1;x2 y2 z2]`) a počtu závitů `Ns`. Spirála je tvořena kruhovou trubicí o poloměru `Rt`. Počet bodů podél kruhového průřezu trubice je dán parametrem `m` a počet bodů podél spirály parametrem `n`.

`sol_spiral` vrací sol-objekt `spiral`.

Příklady
```
spiral1=sol_spiral(0.4,1,4,4,[20 60]);
spiral2=sol_spiral(0.2,1,[0 0 0; 3 3 3],4,[40 40]);
```

Viz též `sol_circle`, `sol_conus`, `sol_cuboid`, `sol_cylinder`, `sol_ellipsoid`, `sol_polyhedra`, `sol_rectangle`, `sol_ring`, `sol_ringext`, `sol_sphere`, `matlab\demos\knot.m`

sol_surface2obj

Funkce Změna grafického objektu MATLABu surface na sol-objekt.

Syntaxe
```
objOut=sol_surface2obj(x,y,z,n);
objOut=sol_surface2obj(x,y,z,n,origin);
```

Popis `sol_surface2obj(x,y,z,n,origin)` mění grafický objekt MATLABu surface na sol-objekt.

Není-li zadán vstupní parametr `origin`, počátek se vypočte jako průměr hodnot `x`, `y` a `z`.

`sol_surface2obj` vrací sol-objekt `objOut`.

Příklady
```
[x,y,z]=peaks;
h=surf(x,y,z);
n=get(h,'vertexnormals');
a=sol_surface2obj(x,y,z,n);
a{1}
```

Viz též `sol_patch2obj`.

sol_translate

Funkce Posunutí objektů.

Syntaxe
```
objOut=sol_translate(objIn,dx,dy,dz)
objOut=sol_translate(objIn,dx,dy,dz,options)
```

Popis `sol_translate(objIn,dx,dy,dz,options)` posunuje sol-objekt `objIn` o vzdálenosti `dx`, `dy` a `dz` (`options='rel'`) nebo na pozici `dx`, `dy` a `dz` (`options='abs'`).

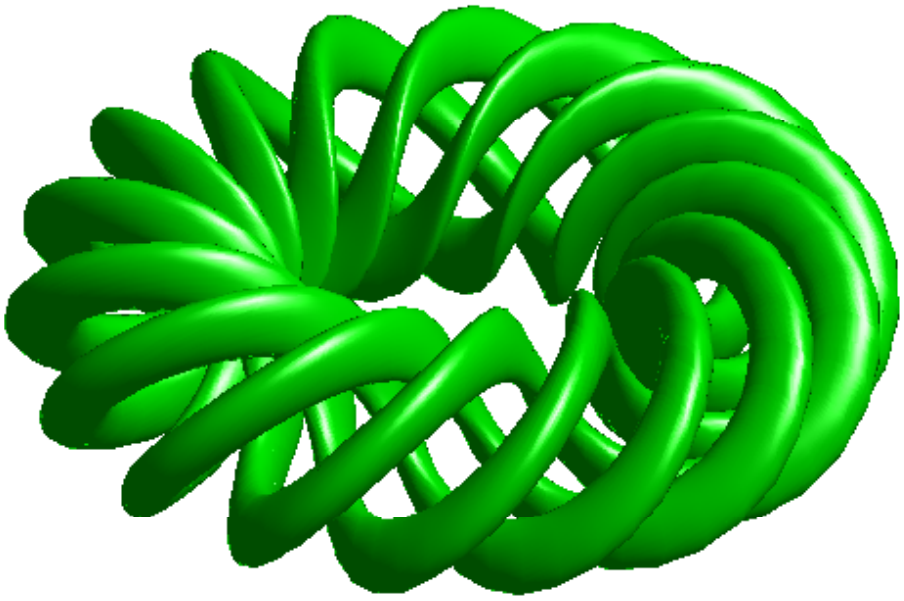
`sol_translate(objIn,dx,dy,dz)` použije implicitní hodnotu `options='rel'`.

`sol_translate` vrací sol-objekt `objOut`.

Příklady
```
conus=sol_conus(2,3,10);
conus1=sol_translate(conus,3,-3,5);
conus2=sol_translate(conus,5,5,5,'abs');
```

Viz též `sol_combine`, `sol_render`, `sol_rotate`, `sol_rotateX`, `sol_rotateY`, `sol_rotateZ`, `sol_scale`, `sol_set`.

Výpis programu



sol_circle.m

```
function circle=sol_circle(radius,n)
% SOL_CIRCLE     generates circle.
%
% Syntax: circle = sol_circle(radius,n)
%    The input paramenter n is related to the circle resolution.
%    Higher number is better approximation.
%
%    sol_circle(radius) uses n = 20.
%    sol_circle uses  radius = 1, n = 20  generates unit circle.
%
% Ex: circle = sol_circle(2,40);
%
% See also  SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID, SOL_POLYHEDRA,
%           SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%           SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2


if nargin < 1, radius = 1; end
if nargin < 2, n = 20; end

% -pi <= theta <= pi is a row vector.
theta = (-n:2:n)/n*pi;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;

x = radius*cos(theta);
y = radius*sintheta;
z = zeros(size(x));

circle.label = 'circle';

circle.vertices = [x; y; z]';
circle.faces = [1:length(x)] ;
circle.normals = [[0; 0; -1]*ones(1,length(x))]';

circle.radius = radius;

circle.origin = [0 0 0];

circle = {circle};
```

sol_combine.m

```
function objOut = sol_combine(varargin);
% SOL_COMBINE combines objects.
% Takes a list of objects (cell arrays) and returns a cell array.
%
% Syntax: objOut = sol_combine(varargin);
%
% Ex: cuboid1=sol_cuboid(1,1,7);
%     cuboid2=sol_cuboid(7,1,1);
%     cross=sol_combine(cuboid1,cuboid2);
%
% See also SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY,
%          SOL_ROTATEZ, SOL_SCALE, SOL_SET, SOL_TRANSLATE.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

num=length(varargin);

if (num==0)
   error('must have at least one input object');
end

objOut={};

for i=1:num
   if (iscell(varargin{i})) %a list of structs
      objOut=[objOut, varargin{i}];
   else
      error('input must be cell array')
   end %if (iscell(varargin(i)))
end %for
```

sol_conus.m

```
function conus = sol_conus(radius1, radius2, height, n)
% SOL_CONUS  generates the conus.
%
% Syntax: conus = sol_conus(radius1, radius2, height, n).
%    The conus has n points around the circumference.
%
%    sol_conus(radius1, radius2, height) uses n = 20.
%    sol_conus(radius1, radius2) uses height = 1, n=20.
%    sol_conus(radius1) uses radius2 = 0, height = 1,n = 20.
%    sol_conus uses radius1=1, radius2 = 0, height = 1,n = 20.
%
%    Height could be two points in form [2x3]-matrix, [x1 y1 z1;x2 y2 z2].
%
% Ex1: conus = sol_conus(3, 2, 10, 40);
% Ex2: conus = sol_conus(3, 2,  [0 0 0;2 11 5], 40);
%
% See also SOL_CIRCLE, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID, SOL_POLYHEDRA,
%          SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 1, radius1 = 1; end
if nargin < 2, radius2 = 0; end
if nargin < 3, height = 1; end
if nargin < 4, n = 20; end

rotflag = 0;
if all(size(height)==[2 3]),
    rotflag = 1;
    first_point = height(1,:);
    second_point = height(2,:);
    sc=[0 0 1];                                % directional vector of sample conus
    sb=second_point - first_point;             % directional vector of conus
    height=norm(sb);                           % height of conus
    fi=acos(dot(sc,sb)/(norm(sc)*norm(sb)));   % angle of rotation in rad
                                               % fi=acos(sb(3)/norm(sb));
    if fi==0,
        u=[1 0 0];
    else
        u=cross(sc,sb);                        % directional vector of axis of rotation,
```

```
                                        % [-sb(2) sb(1) 0]
    end
    u=u/norm(u);
end

r = [radius1 radius2]';
r = r(:);                               % Make sure r is a vector.
theta = (0:n)/n*2*pi;
sintheta = sin(theta); sintheta(n+1) = 0;

x = r * cos(theta);
y = r * sintheta;
z = [-height/2 height/2]' * ones(1,n+1);
normals = cat(3,x,y,radius1*(radius1-radius2)/height*ones(size(z)));
origin = [0 0 0];
base1 = [0 0 -height/2];
base2 = [0 0  height/2];

if rotflag,
    cosa = cos(fi);
    sina = sin(fi);
    vera = 1 - cosa;
    rot = [cosa+u(1)^2*vera u(1)*u(2)*vera-u(3)*sina u(1)*u(3)*vera+u(2)*sina; ...
           u(1)*u(2)*vera+u(3)*sina cosa+u(2)^2*vera u(2)*u(3)*vera-u(1)*sina; ...
           u(1)*u(3)*vera-u(2)*sina u(2)*u(3)*vera+u(1)*sina cosa+u(3)^2*vera]';
    X=x;
    Y=y;
    Z=z+height/2;
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    x = first_point(1)+reshape(newxyz(:,1),m,n);
    y = first_point(2)+reshape(newxyz(:,2),m,n);
    z = first_point(3)+reshape(newxyz(:,3),m,n);

    X=normals(:,:,1);
    Y=normals(:,:,2);
    Z=normals(:,:,3);
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    normals=cat(3,reshape(newxyz(:,1),m,n),...
                  reshape(newxyz(:,2),m,n),...
                  reshape(newxyz(:,3),m,n));
    base1= first_point;
```

```
    base2=second_point;
    origin = mean([base1; base2]);
end

conus.label = 'conus';

conus.xdata = x;
conus.ydata = y;
conus.zdata = z;
conus.normals=normals;

conus.radius1 = radius1;
conus.radius2 = radius2;
conus.height = height;

conus.base1 = base1;
conus.base2 = base2;
conus.origin = origin;

conus = {conus};
```

sol_cuboid.m

```
function cuboid=sol_cuboid(lx,ly,lz)
% SOL_CUBOID generates cuboid.
%
% Syntax: cuboid = sol_cuboid(lx,ly,lz).
%
%    sol_cuboid uses lx=1, ly=1, lz=1 generates unit cube.
%    sol_cuboid(lx) uses  lx=ly=lz    generates cube.
%
% Ex: cuboid = sol_cuboid(2,3,4);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CYLINDER, SOL_ELLIPSOID, SOL_POLYHEDRA,
%          SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 1, lx=1;ly=1;lz=1;end
if nargin < 2, ly=lx;lz=lx;end

%Define a cuboid

cuboid.label = 'cuboid';

a = [ 0    0    0;
     lx    0    0;
     lx   ly    0;
      0   ly    0;
      0    0   lz;
     lx    0   lz;
     lx   ly   lz;
      0   ly   lz;
     lx    0    0;
     lx   ly    0;
     lx   ly   lz;
     lx    0   lz;
      0    0    0;
      0   ly    0;
      0   ly   lz;
      0    0   lz;
      0    0    0;
     lx    0    0;
```

```
     lx    0   lz;
      0    0   lz;
      0   ly    0;
     lx   ly    0;
     lx   ly   lz;
      0   ly   lz];

cuboid.vertices = [a(:,1)-lx/2, a(:,2)-ly/2, a(:,3)-lz/2];

cuboid.faces = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16; 17 18 19 20; 21 22 23 24] ;

cuboid.normals = [ 0  0 -1;
                   0  0 -1;
                   0  0 -1;
                   0  0 -1;
                   0  0  1;
                   0  0  1;
                   0  0  1;
                   0  0  1;
                  -1  0  0;
                  -1  0  0;
                  -1  0  0;
                  -1  0  0;
                   1  0  0;
                   1  0  0;
                   1  0  0;
                   1  0  0;
                   0  1  0;
                   0  1  0;
                   0  1  0;
                   0  1  0;
                   0 -1  0;
                   0 -1  0;
                   0 -1  0;
                   0 -1  0];

cuboid.lx = lx;
cuboid.ly = ly;
cuboid.lz = lz;

cuboid.origin = [0 0 0];

cuboid = {cuboid};
```

sol_cylinder.m

```
function cylinder = sol_cylinder(radius,height,n)
% SOL_CYLINDER generates the cylinder.
%
% Syntax: cylinder = sol_cylinder(radius, height, n)
%    The cylinder has n points around the circumference.
%
%    sol_cylinder(raduis, height) uses n = 20.
%    sol_cylinder(raduis) uses height = 1, n = 20.
%    sol_cylinder uses radius = 1, height = 1,n = 20.
%
%    Height could be two points in form [2x3]-matrix, [x1 y1 z1;x2 y2 z2].
%
% Ex1: cylinder = sol_cylinder(2, 10, 30);
% Ex2: cylinder = sol_cylinder(2, [0 0 0;12 3 5], 30);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_ELLIPSOID, SOL_POLYHEDRA,
%          SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 1, radius = 1; end
if nargin < 2, height = 1; end
if nargin < 3, n = 20; end

rotflag = 0;
if all(size(height)==[2 3]),
    rotflag = 1;
    first_point = height(1,:);
    second_point = height(2,:);
    sc=[0 0 1];                                % directional vector of sample cylinder
    sb=second_point - first_point;             % directional vector of cylinder
    height=norm(sb);                           % height of cylinder
    fi=acos(dot(sc,sb)/(norm(sc)*norm(sb))); % angle of rotation in rad
                                               % fi=acos(sb(3)/norm(sb));
    if fi==0,
        u=[1 0 0];
    else
        u=cross(sc,sb);                        % directional vector of axis of rotation,
                                               % [-sb(2) sb(1) 0]
    end
```

```
    u=u/norm(u);
end

r = [radius radius]';
r = r(:);                                   % Make sure r is a vector.
theta = (-n:2:n)/n*pi;
sintheta = sin(theta); sintheta(n+1) = 0;

x = r * cos(theta);
y = r * sintheta;
z = [-height/2 height/2]' * ones(1,n+1);
normals=cat(3,x,y,zeros(size(z)));
origin = [0 0 0];
base1 = [0 0 -height/2];
base2 = [0 0  height/2];

if rotflag,
    cosa = cos(fi);
    sina = sin(fi);
    vera = 1 - cosa;
    rot = [cosa+u(1)^2*vera u(1)*u(2)*vera-u(3)*sina u(1)*u(3)*vera+u(2)*sina; ...
           u(1)*u(2)*vera+u(3)*sina cosa+u(2)^2*vera u(2)*u(3)*vera-u(1)*sina; ...
           u(1)*u(3)*vera-u(2)*sina u(2)*u(3)*vera+u(1)*sina cosa+u(3)^2*vera]';
    X=x;
    Y=y;
    Z=z+height/2;
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    x = first_point(1)+reshape(newxyz(:,1),m,n);
    y = first_point(2)+reshape(newxyz(:,2),m,n);
    z = first_point(3)+reshape(newxyz(:,3),m,n);

    X=normals(:,:,1);
    Y=normals(:,:,2);
    Z=normals(:,:,3);
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    normals=cat(3,reshape(newxyz(:,1),m,n),...
                  reshape(newxyz(:,2),m,n),...
                  reshape(newxyz(:,3),m,n));
    base1= first_point;
    base2=second_point;
    origin = mean([base1; base2]);
```

```
end

cylinder.label = 'cylinder';

cylinder.xdata = x;
cylinder.ydata = y;
cylinder.zdata = z;
cylinder.normals = normals;

cylinder.radius = radius;
cylinder.height = height;

cylinder.base1 = base1;
cylinder.base2 = base2;
cylinder.origin = origin;

cylinder = {cylinder};
```

sol_ellipsoid.m

```
function ellipsoid = sol_ellipsoid(rx,ry,rz,n)
% SOL_ELLIPSOID generates the ellipsoid.
%
% Syntax: ellipsoid = sol_ellipsoid(rx,ry,rz,n)
%    The ellipsoid has n points around the circumference.
%
%    sol_ellipsoid(rx,ry,rz) uses n = 20.
%
% Ex: ellipsoid = sol_ellipsoid(3,4,5,40);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_POLYHEDRA,
%          SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 4, n = 20; end

% -pi <= theta <= pi is a row vector.
% -pi/2 <= phi <= pi/2 is a column vector.
theta = (-n:2:n)/n*pi;
phi = (-n:2:n)'/n*pi/2;
cosphi = cos(phi); cosphi(1) = 0; cosphi(n+1) = 0;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;

x = rx*cosphi*cos(theta);
y = ry*cosphi*sintheta;
z = rz*sin(phi)*ones(1,n+1);

% egf = sqrt((rx*ry*sin(phi)*ones(1,n+1)).^2+(rx*rz*cosphi*sintheta).^2+...
%       (rz*ry*cosphi*cos(theta)).^2);
% xn  = ry*rz*cosphi*cos(theta)./egf;
% yn  = rx*rz*cosphi*sintheta./egf;
% zn  = rx*ry*sin(phi)*ones(1,n+1)./egf;

egf = sqrt((x/rx/rx).^2+(y/ry/ry).^2+(z/rz/rz).^2);
xn  = x/rx/rx./egf;
yn  = y/ry/ry./egf;
zn  = z/rz/rz./egf;

ellipsoid.label = 'ellipsoid';
```

```
ellipsoid.xdata = x;
ellipsoid.ydata = y;
ellipsoid.zdata = z;
ellipsoid.normals = cat(3,xn,yn,zn);

ellipsoid.rx = rx;
ellipsoid.ry = ry;
ellipsoid.rz = rz;

ellipsoid.origin = [0 0 0];

ellipsoid = {ellipsoid};
```

sol_patch2obj.m

```
function objOut = sol_patch2obj(v,f,n,origin);
% SOL_PATCH2OBJ changes patch data to solid object.
%
% Syntax: objOut = sol_patch2obj(v,f,n,origin);
%
%    sol_patch2obj(v,f,n) calculates origin as a mean of v.
%
% Ex: [x,y,z,v] = flow;
%     p=isosurface(x,y,z,v,-3);
%     h = patch(p);
%     isonormals(x,y,z,v,h)
%
%     n=get(h,'vertexnormals');
%     a=sol_patch2obj(p.vertices, p.faces, n);
%     a{1}
%
% See also SOL_SURFACE2OBJ,
%          SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER,
%          SOL_ELLIPSOID, SOL_POLYHEDRA, SOL_RECTANGLE,
%          SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 4, origin = mean(v); end

objOut.label = 'any_patch';

objOut.vertices = v;
objOut.faces = f;
objOut.normals = n;

objOut.origin = origin;

objOut = {objOut};
```

sol_polyhedra.m

```
function [polyhedron,vertices,faces]=sol_polyhedra(ptype,radius);
% SOL_POLYHEDRA generates one of five regular polyhedra
%
% Syntax:[polyhedron,vertices,faces]=sol_polyhedra(ptype,radius);
%
%    Output parametrs are:
%        polyhedron .... solid object (cell array),
%           vertices   .... matrix of vertices,
%           faces      .... matrix of faces.
%    Input parametrs are:
%           ptype  .... can be: tetrahedron, hexahedron, octahedron,dodecahedron or icosahedron,
%           radius .... number. The polyherdon is scaled to fit inside a sphere with this radius.
%
%    sol_polyhedra(ptype) ................... the polyherdon is scaled to fit inside a unit sphere.
%    p=sol_polyhedra(ptype,radius) .......... output is only solid object.
%    [p,v]= sol_polyhedra(ptype,radius) ..... outputs are solid object and matrix of vertices.
%    [p,v,f]= sol_polyhedra(ptype,radius) ... outputs are solid object, matrix of vertices and
%                                                                    matrix of faces.
%
% Ex: d = sol_polyhedra('dodecahedron',3);
%     [h,vertices]=sol_polyhedra('hexahedron');
%     [t,v,f]=sol_polyhedra('tetrahedron');
%
% See also  SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%           SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%           SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin<2,radius=1;end
if (strcmp(ptype,'tetrahedron'))
    vertices=radius*[0.453608  0.890430  0.037037
                     0.544331 -0.628540 -0.555555
                    -0.090722 -0.366647  0.925925
                    -0.907218  0.104757 -0.407407];

   faces=[1     2     3
          1     2     4
          1     3     4
          2     3     4];
```

```
elseif (strcmp(ptype,'hexahedron'))
   vertices=0.5774*[0 0 0
                    1 0 0
                    1 1 0
                    0 1 0
                    0 0 1
                    1 0 1
                    1 1 1
                    0 1 1] ;

   vertices=radius*(vertices *2 - 0.5774);

   faces=[1 2 6 5
          2 3 7 6
          3 4 8 7
          4 1 5 8
          1 2 3 4
          5 6 7 8] ;

elseif (strcmp(ptype,'octahedron'))
   vertices=radius*[0.408248  0.707107 -0.577350
                    0.408248 -0.707107 -0.577350
                    0.816496  0.000000  0.577350
                   -0.408248  0.707107  0.577350
                   -0.408248 -0.707107  0.577350
                   -0.816496  0.000000 -0.577350];

   faces=[4     3     1
          4     1     6
          6     1     2
          1     3     2
          4     6     5
          6     2     5
          5     2     3
          5     3     4];

  elseif (strcmp(ptype,'dodecahedron'))
   vertices=radius*[0.175988  0.745447 -0.642912
                    0.230032  0.972675  0.031428
                    0.276500 -0.379561 -0.882882
                    0.392664 -0.847627 -0.356851
                    0.417990  0.215329  0.882563
                    0.480109 -0.479965  0.734253
                    0.643598  0.214231 -0.734769
                    0.731044  0.581893  0.356335
```

```
                    0.831555 -0.543116  0.116366
                    0.986641  0.113149 -0.117200
                   -0.175988 -0.745447  0.642912
                   -0.230033 -0.972675 -0.031428
                   -0.276499  0.379562  0.882882
                   -0.392664  0.847627  0.356851
                   -0.417989 -0.215329 -0.882563
                   -0.480109  0.479965 -0.734254
                   -0.643598 -0.214231  0.734769
                   -0.731044 -0.581893 -0.356336
                   -0.831555  0.543115 -0.116366
                   -0.986641 -0.113149  0.117200];

   faces=[ 1  7 10  8  2
           1  7  3 15 16
           7 10  9  4  3
          10  8  5  6  9
           2  8  5 13 14
           2  1 16 19 14
          17 11 12 18 20
          17 11  6  5 13
          11 12  4  9  6
          12 18 15  3  4
          18 20 19 16 15
          20 17 13 14 19];

elseif (strcmp(ptype,'icosahedron'))
    vertices=radius*[0.250254 -0.916162 -0.313083
                     0.320072 -0.078054 -0.944171
                     0.437594 -0.598061  0.671442
                     0.550563  0.758025 -0.349682
                     0.623196  0.436642  0.648821
                     0.975676 -0.177816 -0.128204
                    -0.250253  0.916161  0.313082
                    -0.320073  0.078054  0.944172
                    -0.437593  0.598061 -0.671442
                    -0.550563 -0.758024  0.349683
                    -0.623195 -0.436643 -0.648822
                    -0.975676  0.177817  0.128204];

   faces=[4     5     6
          5     8     3
          4     6     2
          6     3     1
          4     2     9
```

```
          2     1    11
          4     9     7
          9    11    12
          4     7     5
          7    12     8
          3     6     5
         10     3     8
          1     2     6
         10     1     3
         11     9     2
         10    11     1
         12     7     9
         10    12    11
          8     5     7
         10     8    12];

else
   error('type must be:tetrahedron, hexahedron, octahedron,dodecahedron,icosahedron');
end

vn=[];
normals=[];
for face=faces',
   v=vertices(face,:);
   vn=[vn; v];
   normal=mean(v);
   normals=[normals; normal(ones(1,size(v,1)),:)];
end

m=size(faces,1);
n=size(faces,2);
fn=reshape(1:m*n,n,m)';

%Define a polyhedron
polyhedron.label = ptype;

polyhedron.vertices=vn;
polyhedron.faces=fn;
polyhedron.normals=normals;

polyhedron.radius=radius;

polyhedron.origin = [0 0 0];

polyhedron = {polyhedron};
```

sol_rectangle.m

```
function rectangle=sol_rectangle(lx,ly)
% SOL_RECTANGLE generates rectangle
%
% Syntax: rectangle = sol_rectangle(lx,ly)
%
%    sol_rectangle  uses lx=1, ly=1 generates unit square.
%    sol_rectangle(lx)  uses  lx = ly   generates square.
%
% Ex: rectangle = sol_rectangle(2,3);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%          SOL_POLYHEDRA, SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 1, lx = 1;ly=1; end
if nargin < 2, ly=lx; end

%Define a rectangle
a = [ 0    0    0;
     lx    0    0;
     lx   ly    0;
      0   ly    0];

rectangle.label = 'rectangle';

rectangle.vertices = [a(:,1)-lx/2, a(:,2)-ly/2, a(:,3)];

rectangle.faces = [1 2 3 4] ;

rectangle.normals = [ 0  0 -1;
                      0  0 -1;
                      0  0 -1;
                      0  0 -1];
rectangle.lx = lx;
rectangle.ly = ly;

rectangle.origin = [0 0 0];

rectangle = {rectangle};
```

sol_render.m

```
function handles = sol_render(objIn,mode,options)
% SOL_RENDER render function for cell array structure.
%
% Takes a cell array as input.
%
% Syntax: count = sol_render(objIn,mode,options)
%
% Input parametrs are:
%    mode    .... 'solid' or 'wire'
%    options ..... 0 or 1,
%       if options=1 then will be used view(3); axis off, axis equal.
%
%    sol_render(objIn) uses  mode = 'solid', obtions = 1.
%
% Ex1: cage=sol_render(sol_cylinder(1.5,3),'wire');
%      solid_cube=sol_render(sol_cuboid);
% Ex2: spiral=sol_render(sol_spiral(0.2,3,4,4,[30,120]),'solid',0);
%
% See also SOL_COMBINE, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY, SOL_ROTATEZ,
%          SOL_SCALE, SOL_SET, SOL_TRANSLATE.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 3, options = 1; end
if nargin < 2, mode = 'solid'; end

handles=zeros([length(objIn) 1]);
for i=1:length(objIn)
    if (isfield(objIn{i},'vertices'))
        h = patch('Vertices', objIn{i}.vertices,...
                  'Faces', objIn{i}.faces,...
                  'VertexNormals', objIn{i}.normals,'FaceColor','green');
    else
        h = surface('XData', objIn{i}.xdata,...
                    'YData', objIn{i}.ydata,...
                    'ZData', objIn{i}.zdata,...
                    'VertexNormals', objIn{i}.normals,'FaceColor','green');
    end
    handles(i)=h;
%                   'CData', objIn{i}.zdata,...
    if (isfield(objIn{i},'flag'))
```

```
        set(h, objIn{i}.flag(1:2:end),objIn{i}.flag(2:2:end));
    end

    if strcmpi(mode, 'wire')
        set(h, 'FaceColor', 'none', ...
               'EdgeColor', [0 0 0]);
    else
        fc=get(h,'FaceColor');
        if strcmp(fc,'flat') & strcmp(fc,'interp'), set(h, 'FaceColor', [0 1 0]), end
        set(h,'EdgeColor', 'none','FaceLighting', 'phong');
    end
end

if options,
    view(3);
    axis off
    axis equal
end
```

sol_ring.m

```
function ring = sol_ring(rx,ry,radius,pq)
% SOL_RING generates the ring.
%
% Syntax:ring = sol_ring(rx,ry,radius,pq)
%
% The input arguments are following:
%
%   rx     .... radius of the tube in x direction.
%   ry     .... radius of the tube in y direction.
%   radius .... radius of the ring.
%   pq = [p q]
%           q .... number of grid points in each (circular) section of the tube.
%           p .... number of sections along the ring.
%
% Ex: ring = sol_ring(1, 2, 8,[40 40]);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%          SOL_POLYHEDRA, SOL_RECTANGLE, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%          matlab\demos\cruller.m

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

a = 0;   b = 2*pi;
p = pq(1);   q = pq(2);
h = (b-a)/p;
t = a : h : b;

k = 2*pi/q;
u = 0 : k : 2*pi;

[tt,uu] = meshgrid(t,u);

xt = rx*cos(tt);
yt = ry*sin(tt);
egf = sqrt((xt.*ry./rx).^2+(yt.*rx./ry).^2);
xnt = xt.*ry./rx./egf;
ynt = yt.*rx./ry./egf;

xx = (radius + xt).*cos(uu);
yy = (radius + xt).*sin(uu);
zz = yt;
```

```
xxn = xnt.*cos(uu);
yyn = xnt.*sin(uu);
zzn = ynt;

ring.label = 'ring';

ring.xdata = xx;
ring.ydata = yy;
ring.zdata = zz;
ring.normals = cat(3,xxn,yyn,zzn);

ring.radius = radius;
ring.rx = rx;
ring.ry = ry;

ring.origin = [0 0 0];

ring= {ring};
```

sol_ringext.m

```
function ringext = sol_ringext(xy,ab,rtr,pq,varargin)
% SOL_RINGEXT generates ring with various profiles. This ring can twists.
%
% Syntax: ringext = sol_ringext(xy,ab,rtr,pq,varargin)
%
% The input arguments are following :
%
%    xy .... string name of function  [xt,yt] = xy(t, P1, P2, ...)
%            defining parametric curve to be revolved
%    ab =[a b] .... interval of definition of parametric curve
%    rtr=[radius twist revs] for revolution of curve
%    pq =[p q] .... numbers of t- and u-subintervals
%    varagin   .... parameters of function xy
%
% Ex: t1=sol_ringext('xy_ellipse',[0 2*pi],[8 4 1], [80 80],1,2,1,2);
%     t2=sol_ringext('xy_circle', [0 2*pi],[8 4 1], [80 80],1,-2,2);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%          SOL_POLYHEDRA, SOL_RECTANGLE, SOL_RING, SOL_SPHERE, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%          matlab\demos\cruller.m

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

a = ab(1);   b = ab(2);
p = pq(1);   q = pq(2);
h = (b-a)/p;
t = a : h : b;

radius = rtr(1);
twist  = rtr(2);
revs   = rtr(3);

k = 2*revs*pi/q;
u = 0 : k : 2*revs*pi;

[tt,uu] = meshgrid(t,u);

[xt,yt] = feval(xy,tt,varargin{:});
w = xt.*cos(twist*uu) - yt.*sin(twist*uu);
xx = (radius + w).*cos(uu);
yy = (radius + w).*sin(uu);
```

```
w = xt.*sin(twist*uu) + yt.*cos(twist*uu);
zz = w;

[xn,yn,zn] = surfnorm(xx,yy,zz);

ringext.label = 'ringext';

ringext.xdata = xx;
ringext.ydata = yy;
ringext.zdata = zz;
ringext.normals = cat(3,xn,yn,zn);

ringext.radius = radius;

ringext.origin = [0 0 0];

ringext = {ringext};
```

sol_rotate.m

```
function objOut = sol_rotate(objIn,azel,alpha,origin)
% SOL_ROTATE rotates objects about specified origin and direction.
%
%    Direction could be determined by the 2-element direction vector [theta, phi] (spherical
%    coordinates) or  the direction vector [x y z] (Cartesian coordinates).
%    The direction vector is the vector from the center of the plot box to (x,y,z).
%
% Syntax: objOut = sol_rotate(objIn,azel,alpha,origin)
%
% The input arguments are following:
%          objIn  .... object (cell array),
%          azel   .... direction vector = [theta,phi] (spherical coordinates) or
%                                       = [x,y,z] (Cartesian coordinates),
%          alpha  .... angle of rotation,
%          origin .... [x0 y0 z0] the center of rotation instead of
%                                 the center of the coordinates.
%
% All the angles are in degrees.
%
%    Theta is the angle in the xy plane counterclockwise from the
%    positive x axis.  Phi is the elevation of the direction vector
%    from the xy plane (see also SPH2CART).  Positive alpha is defined
%    as the righthand-rule angle about the direction vector as it
%    extends from the origin.
%
%    sol_rotate(objIn, azel, alpha) uses origin = [0 0 0]
%
%
% Ex: a = sol_rotate(sol_cuboid(1,1,2),[45 60],90,[0 0 3]);
%     b = sol_rotate(sol_cylinder(1,2),[1 1 1],90);
%
%
% See also  SOL_COMBINE, SOL_RENDER, SOL_ROTATEX, SOL_ROTATEY,
%           SOL_ROTATEZ, SOL_SCALE, SOL_SET, SOL_TRANSLATE.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2


% Determine the default origin (center of coordinates).
if nargin < 4
  origin = [0 0 0];
```

```
end

% find unit vector for axis of rotation
if prod(size(azel)) == 2 % theta, phi
    theta = pi*azel(1)/180;
    phi = pi*azel(2)/180;
    u = [cos(phi)*cos(theta); cos(phi)*sin(theta); sin(phi)];
elseif prod(size(azel)) == 3 % direction vector
    u = azel(:)/norm(azel);
end

alph = alpha*pi/180;
cosa = cos(alph);
sina = sin(alph);
vera = 1 - cosa;
x = u(1);
y = u(2);
z = u(3);
rot = [cosa+x^2*vera x*y*vera-z*sina x*z*vera+y*sina; ...
       x*y*vera+z*sina cosa+y^2*vera y*z*vera-x*sina; ...
       x*z*vera-y*sina y*z*vera+x*sina cosa+z^2*vera]';

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        if (isfield(objIn{i},'vertices'))
            V=objOut{i}.vertices;
            V=[V(:,1)-origin(1), V(:,2)-origin(2), V(:,3)-origin(3)];
            V = V*rot;
            V=[V(:,1)+origin(1), V(:,2)+origin(2), V(:,3)+origin(3)];
            objOut{i}.vertices=V;
            N=objOut{i}.normals;
            N = N*rot;
            objOut{i}.normals=N;
        else
            X=objOut{i}.xdata-origin(1);
            Y=objOut{i}.ydata-origin(2);
            Z=objOut{i}.zdata-origin(3);
            [m,n] = size(X);
            newxyz = [X(:), Y(:), Z(:)];
            newxyz = newxyz*rot;
            objOut{i}.xdata = origin(1) + reshape(newxyz(:,1),m,n);
            objOut{i}.ydata = origin(2) + reshape(newxyz(:,2),m,n);
            objOut{i}.zdata = origin(3) + reshape(newxyz(:,3),m,n);
```

```
            X=objOut{i}.normals(:,:,1);
            Y=objOut{i}.normals(:,:,2);
            Z=objOut{i}.normals(:,:,3);
            [m,n] = size(X);
            newxyz = [X(:), Y(:), Z(:)];
            newxyz = newxyz*rot;
            objOut{i}.normals=cat(3,reshape(newxyz(:,1),m,n),...
                                    reshape(newxyz(:,2),m,n),...
                                    reshape(newxyz(:,3),m,n));
        end
        % 'origin'
        O=objOut{i}.origin;
        O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
        O = O*rot;
        O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            O=objOut{i}.base1;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O = O*rot;
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base1=O;
        end
        if (isfield(objIn{i},'base2'))
            O=objOut{i}.base2;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O = O*rot;
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base2=O;
        end
    end
else
  error('input must be cell array')
end %if
```

sol_rotateX.m

```
function objOut = sol_rotateX(objIn,alpha,origin)
% SOL_ROTATEX rotates objects about specified origin and direction of axis X.
%
% Syntax: objOut = sol_rotateX(objIn,alpha,origin)
%    The angle is in degrees.
%
%    sol_rotateX(objIn, alpha) uses origin = objOut{i}.origin;
%
% Ex: a = sol_rotateX(sol_cuboid(1,1,2),60,[0 0 3]);
%     b = sol_rotateX(sol_cylinder(1,3),90);
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEY, SOL_ROTATEZ,
%          SOL_SCALE, SOL_SET, SOL_TRANSLATE
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

alpha=alpha/180*pi;  %degrees to radians

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        if nargin < 3, origin = objOut{i}.origin; end
        if (isfield(objIn{i},'vertices'))
            V=objOut{i}.vertices;
            V=[V(:,1)-origin(1), V(:,2)-origin(2), V(:,3)-origin(3)];
            V=[V(:,1), ...
               cos(alpha)*V(:,2)-sin(alpha)*V(:,3), ...
               sin(alpha)*V(:,2)+cos(alpha)*V(:,3)];
            V=[V(:,1)+origin(1), V(:,2)+origin(2), V(:,3)+origin(3)];
            objOut{i}.vertices=V;
            N=objOut{i}.normals;
            N=[N(:,1), ...
               cos(alpha)*N(:,2)-sin(alpha)*N(:,3), ...
               sin(alpha)*N(:,2)+cos(alpha)*N(:,3)];
            objOut{i}.normals=N;
        else
            Y=objOut{i}.ydata-origin(2);
            Z=objOut{i}.zdata-origin(3);
            objOut{i}.ydata=(cos(alpha)*Y-sin(alpha)*Z)+origin(2);
            objOut{i}.zdata=(sin(alpha)*Y+cos(alpha)*Z)+origin(3);
            X=objOut{i}.normals(:,:,1);
```

```
            Y=objOut{i}.normals(:,:,2);
            Z=objOut{i}.normals(:,:,3);
            Yrot=cos(alpha)*Y-sin(alpha)*Z;
            Zrot=sin(alpha)*Y+cos(alpha)*Z;
            objOut{i}.normals=cat(3,X,Yrot,Zrot);
        end
        % 'origin'
        O=objOut{i}.origin;
        O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
        O=[O(1), ...
           cos(alpha)*O(2)-sin(alpha)*O(3), ...
           sin(alpha)*O(2)+cos(alpha)*O(3)];
        O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            O=objOut{i}.base1;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[O(1), ...
               cos(alpha)*O(2)-sin(alpha)*O(3), ...
               sin(alpha)*O(2)+cos(alpha)*O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base1=O;
        end
        if (isfield(objIn{i},'base2'))
            O=objOut{i}.base2;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[O(1), ...
               cos(alpha)*O(2)-sin(alpha)*O(3), ...
               sin(alpha)*O(2)+cos(alpha)*O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base2=O;
        end
    end
else
    error('input must be cell array')
end %if
```

sol_rotateY.m

```
function objOut = sol_rotateY(objIn,alpha,origin)
% SOL_ROTATEY rotates objects about specified origin and direction of axis Y.
%
% Syntax: objOut = sol_rotateY(objIn,alpha,origin)
%    The angle is in degrees.
%
%    sol_rotateY(objIn,alpha) uses origin = objOut{i}.origin;
%
% Ex: a = sol_rotateY(sol_cuboid(1,1,2),60,[0 0 3]);
%     b = sol_rotateY(sol_cylinder(1,3),90);
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEZ,
%          SOL_SCALE, SOL_SET, SOL_TRANSLATE
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

alpha=alpha/180*pi;  %degrees to radians

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        if nargin < 3, origin = objOut{i}.origin; end
        if (isfield(objIn{i},'vertices'))
            V=objOut{i}.vertices;
            V=[V(:,1)-origin(1), V(:,2)-origin(2), V(:,3)-origin(3)];
            V=[cos(alpha)*V(:,1)+sin(alpha)*V(:,3), ...
               V(:,2), ...
               -sin(alpha)*V(:,1)+cos(alpha)*V(:,3)];
            V=[V(:,1)+origin(1), V(:,2)+origin(2), V(:,3)+origin(3)];
            objOut{i}.vertices=V;
            N=objOut{i}.normals;
            N=[cos(alpha)*N(:,1)+sin(alpha)*N(:,3), ...
               N(:,2), ...
               -sin(alpha)*N(:,1)+cos(alpha)*N(:,3)];
            objOut{i}.normals=N;
        else
            X=objOut{i}.xdata-origin(1);
            Z=objOut{i}.zdata-origin(3);
            objOut{i}.xdata=(cos(alpha)*X+sin(alpha)*Z)+origin(1);
            objOut{i}.zdata=(-sin(alpha)*X+cos(alpha)*Z)+origin(3);
            X=objOut{i}.normals(:,:,1);
```

```
            Y=objOut{i}.normals(:,:,2);
            Z=objOut{i}.normals(:,:,3);
            Xrot=cos(alpha)*X+sin(alpha)*Z;
            Zrot=-sin(alpha)*X+cos(alpha)*Z;
            objOut{i}.normals=cat(3,Xrot,Y,Zrot);
        end
        % 'origin'
        O=objOut{i}.origin;
        O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
        O=[cos(alpha)*O(1)+sin(alpha)*O(3), ...
           O(2), ...
           -sin(alpha)*O(1)+cos(alpha)*O(3)];
        O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            O=objOut{i}.base1;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[cos(alpha)*O(1)+sin(alpha)*O(3), ...
               O(2), ...
               -sin(alpha)*O(1)+cos(alpha)*O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base1=O;
        end
        if (isfield(objIn{i},'base2'))
            O=objOut{i}.base2;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[cos(alpha)*O(1)+sin(alpha)*O(3), ...
               O(2), ...
               -sin(alpha)*O(1)+cos(alpha)*O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base2=O;
        end
    end
else
    error('input must be cell array')
end %if
```

sol_rotateZ.m

```
function objOut = sol_rotateZ(objIn,alpha,origin)
% SOL_ROTATEZ rotates objects about specified origin and direction of axis Z.
%
% Syntax: objOut = sol_rotateZ(objIn,alpha,origin)
%    The angle is in degrees.
%
%    sol_rotateZ(objIn, alpha) uses origin = objOut{i}.origin;
%
% Ex:a = sol_rotateZ(sol_cuboid(1,1,2),60,[0 -2 0]);
%    b = sol_rotateZ(sol_cuboid(0.5,1,3),20);
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY,
%          SOL_SCALE, SOL_SET, SOL_TRANSLATE
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2


alpha=alpha/180*pi;  %degrees to radians

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        if nargin < 3, origin = objOut{i}.origin; end
        if (isfield(objIn{i},'vertices'))
            V=objOut{i}.vertices;
            V=[V(:,1)-origin(1), V(:,2)-origin(2), V(:,3)-origin(3)];
            V=[cos(alpha)*V(:,1)-sin(alpha)*V(:,2), ...
               sin(alpha)*V(:,1)+cos(alpha)*V(:,2), ...
               V(:,3)];
            V=[V(:,1)+origin(1), V(:,2)+origin(2), V(:,3)+origin(3)];
            objOut{i}.vertices=V;
            N=objOut{i}.normals;
            N=[cos(alpha)*N(:,1)-sin(alpha)*N(:,2), ...
               sin(alpha)*N(:,1)+cos(alpha)*N(:,2), ...
               N(:,3)];
            objOut{i}.normals=N;
        else
            X=objOut{i}.xdata-origin(1);
            Y=objOut{i}.ydata-origin(2);
            objOut{i}.xdata=(cos(alpha)*X-sin(alpha)*Y)+origin(1);
            objOut{i}.ydata=(sin(alpha)*X+cos(alpha)*Y)+origin(2);
```

```
            X=objOut{i}.normals(:,:,1);
            Y=objOut{i}.normals(:,:,2);
            Z=objOut{i}.normals(:,:,3);
            Xrot=cos(alpha)*X-sin(alpha)*Y;
            Yrot=sin(alpha)*X+cos(alpha)*Y;
            objOut{i}.normals=cat(3,Xrot,Yrot,Z);
        end
        % 'origin'
        O=objOut{i}.origin;
        O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
        O=[cos(alpha)*O(1)-sin(alpha)*O(2), ...
           sin(alpha)*O(1)+cos(alpha)*O(2), ...
           O(3)];
        O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            O=objOut{i}.base1;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[cos(alpha)*O(1)-sin(alpha)*O(2), ...
               sin(alpha)*O(1)+cos(alpha)*O(2), ...
               O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base1=O;
        end
        if (isfield(objIn{i},'base2'))
            O=objOut{i}.base2;
            O=[O(1)-origin(1), O(2)-origin(2), O(3)-origin(3)];
            O=[cos(alpha)*O(1)-sin(alpha)*O(2), ...
               sin(alpha)*O(1)+cos(alpha)*O(2), ...
               O(3)];
            O=[O(1)+origin(1), O(2)+origin(2), O(3)+origin(3)];
            objOut{i}.base2=O;
        end
    end
else
    error('input must be cell array')
end %if
```

sol_scale.m

```
function objOut = sol_scale(objIn,x)
% SOL_SCALE scale function for cell array structure.
%
% Syntax: objOut = sol_scale(objIn,x)
%
% Ex: triple_cube = sol_scale(sol_cuboid,3);
%     half_sphere = sol_scale(sol_sphere,0.5);
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY,
%          SOL_ROTATEZ, SOL_SET, SOL_TRANSLATE.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
        origin = objOut{i}.origin;
        if (isfield(objIn{i},'vertices'))
            V=objIn{i}.vertices;
            V=[(V(:,1)-origin(1))*x+origin(1),...
               (V(:,2)-origin(2))*x+origin(2),...
               (V(:,3)-origin(3))*x+origin(3)];
            objOut{i}.vertices=V;
        else
            objOut{i}.xdata=(objOut{i}.xdata-origin(1))*x+origin(1);
            objOut{i}.ydata=(objOut{i}.ydata-origin(2))*x+origin(2);
            objOut{i}.zdata=(objOut{i}.zdata-origin(3))*x+origin(3);
        end
        % 'origin'
        O=objOut{i}.origin;
        O=(O-origin)*x+origin;
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            B=objOut{i}.base1;
            B=(B-origin)*x+origin;
            objOut{i}.base1=B;
        end
        if (isfield(objIn{i},'base2'))
            B=objOut{i}.base2;
            B=(B-origin)*x+origin;
```

```
    objOut{i}.base2=B;
end
if (isfield(objIn{i},'length'))
    L=objOut{i}.length;
    L=L*x;
    objOut{i}.length=L;
end
if (isfield(objIn{i},'radius'))
    R=objOut{i}.radius;
    R=R*x;
    objOut{i}.radius=R;
end
if (isfield(objIn{i},'radius1'))
    R=objOut{i}.radius1;
    R=R*x;
    objOut{i}.radius1=R;
end
if (isfield(objIn{i},'radius2'))
    R=objOut{i}.radius2;
    R=R*x;
    objOut{i}.radius2=R;
end
if (isfield(objIn{i},'lx'))
    L=objOut{i}.lx;
    L=L*x;
    objOut{i}.lx=L;
end
if (isfield(objIn{i},'ly'))
    L=objOut{i}.ly;
    L=L*x;
    objOut{i}.ly=L;
end
if (isfield(objIn{i},'lz'))
    L=objOut{i}.lz;
    L=L*x;
    objOut{i}.lz=L;
end
if (isfield(objIn{i},'rx'))
    L=objOut{i}.rx;
    L=L*x;
    objOut{i}.rx=L;
end
if (isfield(objIn{i},'ry'))
    L=objOut{i}.ry;
    L=L*x;
```

```
            objOut{i}.ry=L;
        end
        if (isfield(objIn{i},'rz'))
            L=objOut{i}.rz;
            L=L*x;
            objOut{i}.rz=L;
        end
    end
else
    error('input must be cell array')
end %if
```

sol_set.m

```
function objOut = sol_set(objIn,varargin)
% SOL_SET sets the value of the specified property
%         for the cell array structure.
%
% Syntax: objOut = sol_set(objIn,varargin)
%
% Ex: ball=sol_sphere; cube= sol_cuboid(3);
%     red_ball= sol_set(ball,'FaceColor',[1 0 0],'EdgeColor',[1 1 0]);
%     transparent_cube= sol_set(cube,'FaceAlpha',0.2);
%
% Input is:
%  a cell array of structure consisting of a vertex and face array
%  or a x,y,z  and 'PropertyName',PropertyValue
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY,
%          SOL_ROTATEZ, SOL_SCALE, SOL_TRANSLATE.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};
         if (isfield(objOut{i},'flag'))
             objOut{i}.flag=[objOut{i}.flag varargin];
         else
             objOut{i}.flag=varargin;
         end
    end
else
    error('input must be cell array')
end
```

sol_sphere.m

```
function sphere = sol_sphere(radius,n)
% SOL_SPHERE generates sphere.
%
% Syntax: sphere = sol_sphere(radius,n)
%
%   sol_sphere(radius) uses n=20.
%   sol_sphere uses radius=1, n=20 generates unit sphere.
%
% Ex: sphere = sol_sphere(3,40)
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%          SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPIRAL,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 1, radius = 1; end
if nargin < 2, n = 20; end

theta = (-n:2:n)/n*pi;          % -pi <= theta <= pi is a row vector.
phi = (-n:2:n)'/n*pi/2;         % -pi/2 <= phi <= pi/2 is a column vector.
cosphi = cos(phi); cosphi(1) = 0; cosphi(n+1) = 0;
sintheta = sin(theta); sintheta(1) = 0; sintheta(n+1) = 0;

x = radius*cosphi*cos(theta);
y = radius*cosphi*sintheta;
z = radius*sin(phi)*ones(1,n+1);

sphere.label = 'sphere';

sphere.xdata = x;
sphere.ydata = y;
sphere.zdata = z;
sphere.normals = cat(3,x,y,z);

sphere.radius = radius;

sphere.origin = [0 0 0];

sphere = {sphere};
```

sol_spiral.m

```
function spiral = sol_spiral(Rt,Rs,Hs,Ns,mn)
% SOL_SPIRAL generates spiral.
%
% Syntax: spiral = sol_spiral(Rt,Rs,Hs,Ns,mn)
%
% The input arguments are following :
%
%   Rt ... radius of the tube.
%   Rs ... radius of the spiral.
%   Hs ... height of the spiral. Height could be two points in form [2x3]-matrix,
%                                [x1 y1 z1;x2 y2 z2].
%   Ns ... number of revolution per Hs.
%
%   mn = [m n]
%           m ... number of grid points in each (circular) section of the tube.
%           n ... number of sections along the tube.
%
% Ex1: spiral1 = sol_spiral(0.4, 1, 4, 4, [20 60]);
% Ex2: spiral2 = sol_spiral(0.2,1,[0 0 0; 3 3 3],4,[40 40]);
%
% See also SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER, SOL_ELLIPSOID,
%          SOL_POLYHEDRA, SOL_RECTANGLE, SOL_RING, SOL_RINGEXT, SOL_SPHERE,
%          SOL_SURFACE2OBJ, SOL_PATCH2OBJ.
%          matlab\demos\knot.m

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

rotflag = 0;
if all(size(Hs)==[2 3]),
    rotflag = 1;
    first_point = Hs(1,:);
    second_point = Hs(2,:);
    sc=[0 0 1];                                 % directional vector of sample spiral
    sb=second_point - first_point;              % directional vector of spiral
    Hs=norm(sb);                                % height of spiral
    fi=acos(dot(sc,sb)/(norm(sc)*norm(sb)));    % angle of rotation in rad
                                                % fi=acos(sb(3)/norm(sb));
    if fi==0,
        u=[1 0 0];
    else
        u=cross(sc,sb);                         % directional vector of axis of rotation,
                                                % [-sb(2) sb(1) 0]
```

```
    end
    u=u/norm(u);
end

Fs = Ns/Hs;

m = mn(1);              % Number of grid points in each (circular) section of the tube.
n = mn(2);              % Number of sections along the tube.
A = Rs;                 % Radius of the spiral
R = Rt;                 % Radius of the tube.

% Do not change this!
t = Hs*(0:n)/n;

% [x10;x20;x30] is the parametric representation of
% the center-line of the tube:
x10 = A*cos(2*pi*Fs*t);
x20 = A*sin(2*pi*Fs*t);
x30 = t;

% [x11;x21;x31] is velocity (same as tangent) vector:
x11 = -A*2*pi*Fs*sin(2*pi*Fs*t);
x21 =  A*2*pi*Fs*cos(2*pi*Fs*t);
x31 = ones(size(x30));

% [x12;x22;x32] is acceleration vector:
x12 = -A*(2*pi*Fs)^2*cos(2*pi*Fs*t);
x22 = -A*(2*pi*Fs)^2*sin(2*pi*Fs*t);
x32 = zeros(size(x30));

speed = sqrt(x11.^2 + x21.^2 + x31.^2);

% This is the dot-product of the velocity and acceleration vectors:
velacc = x11.*x12 + x21.*x22 + x31.*x32;

% Here is the normal vector:
nrml1 = speed.^2 .* x12 - velacc.*x11;
nrml2 = speed.^2 .* x22 - velacc.*x21;
nrml3 = speed.^2 .* x32 - velacc.*x31;
normallength = sqrt(nrml1.^2 + nrml2.^2 + nrml3.^2);

% And here is the normalized normal vector:
unitnormal1 = nrml1 ./ normallength;
unitnormal2 = nrml2 ./ normallength;
unitnormal3 = nrml3 ./ normallength;
```

```
% And the binormal vector ( B = T x N )
binormal1 = (x21.*unitnormal3 - x31.*unitnormal2) ./ speed;
binormal2 = (x31.*unitnormal1 - x11.*unitnormal3) ./ speed;
binormal3 = (x11.*unitnormal2 - x21.*unitnormal1) ./ speed;

% s is the coordinate along the circular cross-sections of the tube:
s = (0:m)';
s = (2*pi/m)*s;

% Finally, the parametric surface.
% Each of x1, x2, x3 is an (m+1)x(n+1) matrix.
% Rows represent coordinates along the tube.  Columns represent coordinates
% in each (circular) cross-section of the tube.

xa1 = ones(m+1,1)*x10;
xb1 = (cos(s)*unitnormal1 + sin(s)*binormal1);
xa2 = ones(m+1,1)*x20;
xb2 = (cos(s)*unitnormal2 + sin(s)*binormal2);
xa3 = ones(m+1,1)*x30;
xb3 = (cos(s)*unitnormal3 + sin(s)*binormal3);
color = ones(m+1,1)*((0:n)*2/n-1);

x1 = xa1 + R*xb1;
x2 = xa2 + R*xb2;
x3 = xa3 + R*xb3;

x1n=x1-xa1;
x2n=x2-xa2;
x3n=x3-xa3;

normals = cat(3,x1n,x2n,x3n);
base1=[0 0 0];
base2=[0 0 Hs];

if rotflag,
    cosa = cos(fi);
    sina = sin(fi);
    vera = 1 - cosa;
    rot = [cosa+u(1)^2*vera u(1)*u(2)*vera-u(3)*sina u(1)*u(3)*vera+u(2)*sina; ...
           u(1)*u(2)*vera+u(3)*sina cosa+u(2)^2*vera u(2)*u(3)*vera-u(1)*sina; ...
           u(1)*u(3)*vera-u(2)*sina u(2)*u(3)*vera+u(1)*sina cosa+u(3)^2*vera]';
    X=x1;
    Y=x2;
    Z=x3;
```

```
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    x1 = first_point(1)+reshape(newxyz(:,1),m,n);
    x2 = first_point(2)+reshape(newxyz(:,2),m,n);
    x3 = first_point(3)+reshape(newxyz(:,3),m,n);

    X=normals(:,:,1);
    Y=normals(:,:,2);
    Z=normals(:,:,3);
    [m,n] = size(X);
    newxyz = [X(:), Y(:), Z(:)];
    newxyz = newxyz*rot;
    normals=cat(3,reshape(newxyz(:,1),m,n),...
                          reshape(newxyz(:,2),m,n),...
                          reshape(newxyz(:,3),m,n));
     base1= first_point;
     base2=second_point;
     origin = mean([base1; base2]);
end

spiral.label = 'spiral';

spiral.xdata = x1;
spiral.ydata = x2;
spiral.zdata = x3;
spiral.normals=normals;

spiral.radius_tube = Rt;
spiral.radius_spiral = Rs;
spiral.height = Hs;

spiral.base1 = base1;
spiral.base2 = base2;

spiral.origin = mean([base1; base2]);

spiral= {spiral};
```

sol_surface2obj.m

```
function objOut = sol_surface2obj(x,y,z,n,origin);
% SOL_SURFACE2OBJ changes surface to solid object.
%
% Syntax: objOut = sol_surface2obj(x,y,z,n,origin);
%
%    sol_surface2obj(v,f,n) calculates origin as a mean of  x, y, z.
%
% Ex:[x,y,z]=peaks;
%    h=surf(x,y,z);
%    n=get(h,'vertexnormals');
%    a = sol_surface2obj(x,y,z,n);
%    a{1}
%
% See also SOL_PATCH2OBJ,
%          SOL_CIRCLE, SOL_CONUS, SOL_CUBOID, SOL_CYLINDER,
%          SOL_ELLIPSOID, SOL_POLYHEDRA, SOL_RECTANGLE,
%          SOL_RING, SOL_RINGEXT, SOL_SPHERE, SOL_SPIRAL.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin < 5, origin = mean([x(:) y(:) z(:)]); end

objOut.label = 'any_surface';

objOut.xdata = x;
objOut.ydata = y;
objOut.zdata = z;
objOut.normals = n;

objOut.origin = origin;

objOut = {objOut};
```

sol_translate.m

```
function objOut = sol_translate(objIn,dx,dy,dz,options)
% SOL_TRANSLATE translate function for cell array structure.
%
% Syntax:objOut = sol_translate(objIn,dx,dy,dz,options)
%
% Input argument options can be 'rel' or 'abs'
% (insted of  relatively or absolutely translation).
%
% Relatively translation means taht the object will be translate
%                          about distances dx,dy,dz.
% Absolutely translation means taht the origin of the object will be placed
%                          at position dx,dy,dz.
%
%    sol_translate(objIn,dx,dy,dz) uses options='rel'
%
% Ex:conus=sol_conus(2,3,10);
%    conus1=sol_translate(conus,3,-3,5);
%    conus2=sol_translate(conus,5,5,5,'abs');
%
% See also SOL_COMBINE, SOL_RENDER, SOL_ROTATE, SOL_ROTATEX, SOL_ROTATEY,
%          SOL_ROTATEZ, SOL_SCALE, SOL_SET.
%

% (c) Petr HORA & Olga CERVENA, 2004, Solid Graphics Toolbox 0.2

if nargin<5, options='rel'; end

if (iscell(objIn)) %a list of structs
    for i=1:length(objIn)
        objOut{i}=objIn{i};

        O=objOut{i}.origin;
        if strcmp(options, 'abs')
            x=dx-O(1);
            y=dy-O(2);
            z=dz-O(3);
        else
            x=dx;
            y=dy;
            z=dz;
        end
```

```
        if (isfield(objIn{i},'vertices'))
            V=objOut{i}.vertices;
            V=[V(:,1)+x, V(:,2)+y, V(:,3)+z];
            objOut{i}.vertices=V;
        else
            objOut{i}.xdata=objOut{i}.xdata+x;
            objOut{i}.ydata=objOut{i}.ydata+y;
            objOut{i}.zdata=objOut{i}.zdata+z;
        end
        % 'origin'
        O=objOut{i}.origin;
        O=[O(1)+x, O(2)+y, O(3)+z];
        objOut{i}.origin=O;

        if (isfield(objIn{i},'base1'))
            B=objOut{i}.base1;
            B=[B(1)+x, B(2)+y, B(3)+z];
            objOut{i}.base1=B;
        end
        if (isfield(objIn{i},'base2'))
            B=objOut{i}.base2;
            B=[B(1)+x, B(2)+y, B(3)+z];
            objOut{i}.base2=B;
        end
    end
else
    error('input must be cell array')
end %if
```