1.积分的计算

方案一：符号计算(int)

```
In [1]: pkg load symbolic  % 需安装，由octave-forge提供，octave默认安装没有
```

```
In [ ]: pkg list   % 查看已经安装的octave包
        % 如果没有，用 pkg install -forge symbolic 安装(在octave环境下)
        % win10环境下有可能会失败，可以安装带sympy的版本：  https://github.com/cbm755/octsympy/re
        leases
```

```
In [ ]: pkg install -forge symbolic
```

```
In [ ]: % pkg uninstall symbolic
```

```
In [3]: pkg load symbolic   % remember to load it, as well as import in python and java
```

符号的定义

```
In [6]: syms x y z t;    % 或用 x = sym('x');
        % 可以用  syms x y z f g  创建多个符号
```

```
In [9]: z = sin(x)*exp(t)
```

```
z = (sym)

   t
  e *sin(x)
```

```
In [17]: diff(z,x,2)
```

```
ans = (sym)

    t
  -e *sin(x)
```

```
In [8]: y = 1/(1+cos(x))^2
```

```
y = (sym)

        1
  -------------
              2
  (cos(x) + 1)
```

In [18]:

```
int(y)    % 输出格式有点难看的
```

```
ans = (sym)

     3/x\      /x\
  tan |-|   tan|-|
      \2/      \2/
  ------- + ------
     6         2
```

In [14]:

```
diff(y)
```

```
ans = (sym)

     2*sin(x)
  -------------
              3
  (cos(x) + 1)
```

In [11]:

```
val = int(y,1,2)    % 定积分
```

```
val = (sym)

                  3           3
    tan(1/2)   tan (1/2)   tan (1)   tan(1)
  - -------- - --------- + ------- + ------
       2           6          6        2
```

In [13]:

```
vpa(val,3500)   % 输出其实还是一个符号
```

```
ans = (sym)

  1.10796590586240441417050738189862399898056041000767931847167389920435277241815
  906422149139739326975819198732397738142296529606602716548809420550769528719624
  27489941379135006446055455969452788690704198675644421551737055492141163741474
  204146322463490781139288624739509159252482330216335264268499064506851529997085
  249316465171826414291045033193936521301583594950045422799051091039712584258750
  351315958819182282390600986056973811847991876134669547117228357571044602294943
  131201057684805485839093493903382400167901829179060159751206374303081066452063
  558669672598030434197013192017072369690468223498517991619545991915222186500550
  928515629461586335390700462111241626203651478582368758366780167939700407077940
  506742881660927096570470826215168977150606350732798464004849305096505458083421
  504090547278620388038816462095259559338274996629182386847486204020997384735494
  453242638544555372414129551497381986138273835790366465995904993460649154302189
  138489672981014184126880879334999999832970949235415326467885272720249289236251
  544097294965056736421619055091694514100847188657672053179521769774672580421600
  752429239960273938482919908887295763286733192852036478900239060496974880770556
  958002230183579155200481136147998076470441940447650626853732531778426488689093
  625040543008581812047693439305835625437254745788923147297063434690079847023537
  766749308927815705561258715172846425989740917202503243244141979170699386538660
  540721267586001339379384258390223296206226900798808655887402172642915270729448
  613571356118884286376905884641767181268438449075990665370742162152085536921042
  982876161584869705966623610372145055768933671164347933805839113830940242121287
  629140505832494350479959669589285088071301444440259208996022075725312769639257
  748494644771956983918968898065523492786415459954051410384025763217814044530209
  825880472315487484783254965788899372992900556558915456369567762761404148878145
  980108686785910455908317845055806230380174496019082996094672780383172510329500
  901337659866754389691088184331945407648349897374876959196316822610958029652072
  693608790570724623212133313779205665151525189383307661632914688016249511407139
  477096972000817453497644890034160458718996197266639351249278667915661448064929
  374698346528767918426994477052762105766604095733314550260832967369934623114785
  499183889247647006883290581271442758019700643598181916356549726894576976645615
  979369065444600974077015207142776562355314340468240965294003627025873204065952
  136185221615288610343987200387100229489722337724278294489135974122480096435704
  121513464580624900531986093878851990694708236821270111619169257746596916515588
  162679815318224473958919431332560987154909452815736984487941596898091179673343
  417167324201315078923876960349066073505520827495180126499275379803247560426573
  441676962608024190992341879780736772377048123112381548658292328815925750761767
  074416466364599094797434041659415358019227726058372821292261847134385534073114
  264882668725972061927311374581942154617456828403791252735972710202100783232234
  109502125893765685005764321193819718790893419548851185249058609663438549185141
  463073472897979481716194677721389192574051842643935780809807941040318291179130
  517477724739409551797197923047854277474253042102659088987840328982718173226953
  540392697779103650497847063766327879601698995659956528206636820806868344784570
  337030282942188754873191966122290099913600211192865150078591779073787337831925
  742954543437960249186251483605270635867538260842363888152159747759310348695471
  363603635382348286367289737896737302864831255202230472866634273992916
```

In [10]:

```
vpa(pi, 2180)   % 只要内存和计算时间允许，可以任意多位！
```

```
ans = (sym)

  3.1415926535897932384626433832795028841971693993751058209749445923078164062862
  0899862803482534211706798214808651328230664709384460955058223172535940812848111
  7450284102701938521105559644622948954930381964428810975665933446128475648233
  7867831652712019091456485669234603486104543266482133936072602491412737245870066
  0631558817488152092096282925409171536436789259036001133053054882046652138414695
  1941511609433057270365759591953092186117381932611793105118548074462379962749567
  3518857527248912279381830119491298336733624406566430860213949463952247371907021
  7986094370277053921717629317675238467481846766940513200056812714526356082778577
  1342757789609173637178721468440901224953430146549585371050792279689258923542019
  9561121290219608640344181598136297747713099605187072113499999983729780499510597
  3173281609631859502445945534690830264252230825334468503526193118817101000313783
  8752886587533208381420617177669147303598253490428755468731159562863882353787593
  7519577818577805321712268066130019278766111959092164201989380952572010654858632
  7886593615338182796823030195203530185296899577362259941389124972177528347913151
  5574857242454150695950829533116861727855889075098381754637464939319255060400927
  7016711390098488240128583616035637076601047101819429555961989467678374494482553
  7977472684710404753464620804668425906949129331367702898915210475216205696602405
  8038150193511253382430035587640247496473263914199272604269922796782354781636009
  3417216412199245863150302861829745557067498385054945885869269956909272107975093
  0295532116534498720275596023648066549911988183479775356636980742654252786255181
  8417574672890977772793800081647060016145249192173217214772350141441973568548161
  3611573525521334757418494684385233239073941433345477624168625189835694855620992
  1922218427255025425688767179049460165346680498862723279178608578438382796797668
  1454100953883786360950680064225125205117392984896084128488626945604241965285022
  2106611863067442786220391949450471237137869609563643719172874677646575739624138
  9086583264599581339047802759009946576407895126946839835259570982582262052248940
  7726719478268482601476990902640136394437455305068203496252451749399651431429809
  1906592509372216964615157098583874105978859597729755
```

方案二：　数值积分法（quad)

求Gauss函数的积分：

$$I_G(x) = \int_{-\infty}^{x} e^{-x^2} dx$$

我们知道准确值为$I_G(\infty) = \sqrt{\pi}$

In [23]:

```
format long
sqrt(pi)
```

```
ans =  1.772453850905516
```

In [20]:

```
function y = func_gauss(t)
y = exp(-t*t); %1/(1+t*t)^2;
endfunction
```

In [27]:

```
[val, ierr, nfun, err] = quad('func_gauss', -10, +10)    % quadrature
```

```
val =  1.772453850905516
ierr = 0
nfun =  231
err =    3.695852112137264e-13
```

In [ ]:

符号计算的更多用法

In [13]:
```
f = sym('log(x)')
```

```
f = (sym) log(x)
```

In [14]:
```
int(sym('log(x)'),1,10)
```

```
ans = (sym) -9 + 10·log(10)
```

In [16]:
```
int('log(x)',1,10)    % this is wrong => int(f,1,10)
```

```
ans = (sym) -9 + 10·log(10)
```

In [ ]:

函数的运算

In [28]:
```
sym x;
f = x^2 + 3*x + 54;
g = 2*x^3 + 1;
```

In [30]:
```
compose(f,g)   %  在matlab和mathematica中是标配
```

```
'perl' ڲшX̨ǿeïj
ļ
error: 'compose' undefined near line 1 column 1
```

In [18]:
```
help compose
```

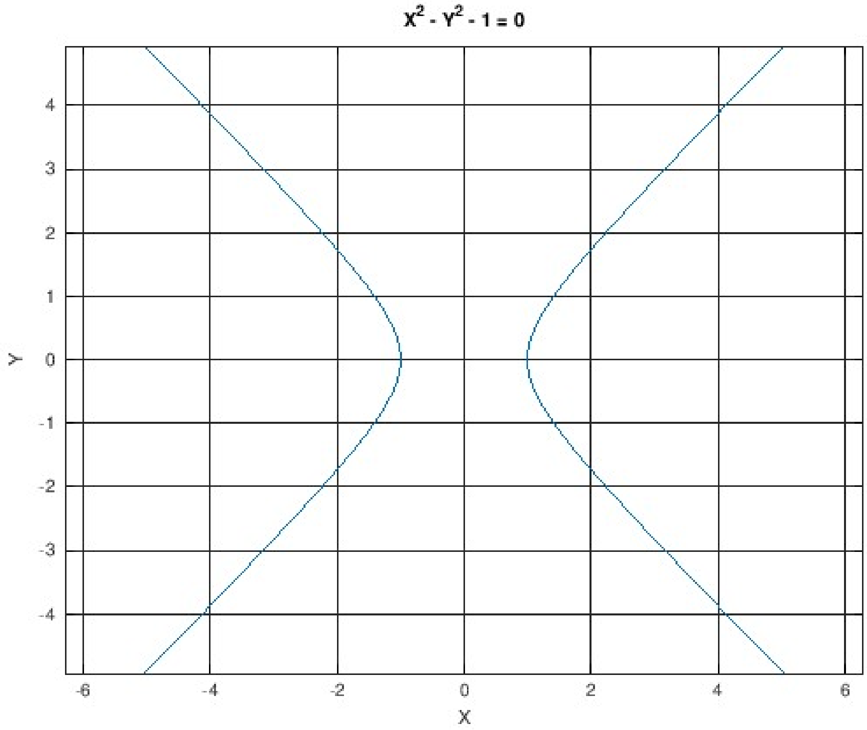
```
error: help: 'compose' not found
```

In [19]:
```
finverse(f)   % 待开发更新，开源软件！！ 详阅warning信息
```

```
warning: the 'finverse' function belongs to the symbolic package from Octave
Forge but has not yet been implemented.

Please read <http://www.octave.org/missing.html> to learn how you can
contribute missing functionality.
error: 'finverse' undefined near line 1 column 1
```

In [20]:

```
syms X Y
 fun = X.^2 - Y.^2 - 1;    % 双曲线 fun = 0, 即 x^2 - y^2 - 1 = 0
 ezplot(fun); grid on;  axis equal; %axis tight
```



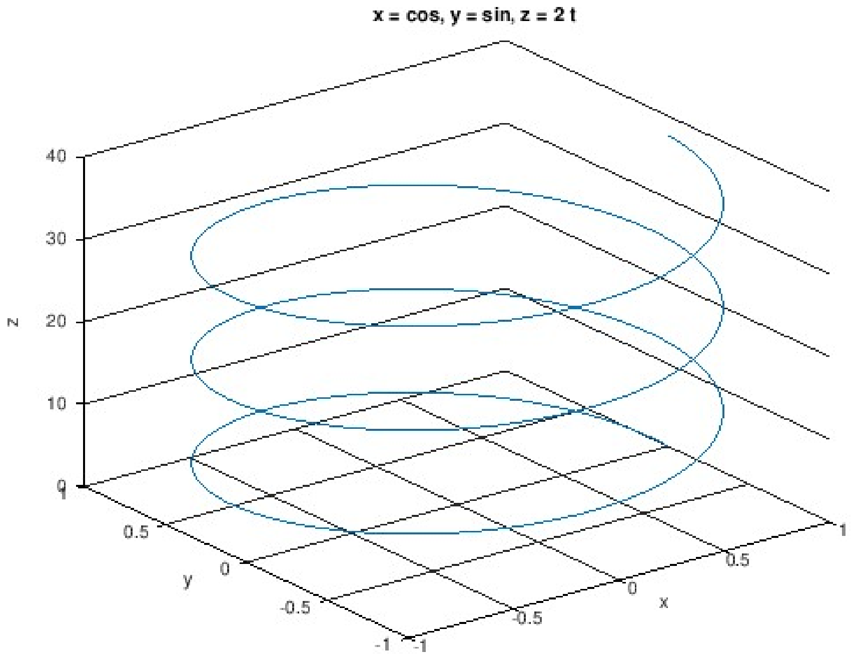
In [ ]:

```
fun
```

其他函数绘图展示（非符号计算工具包）

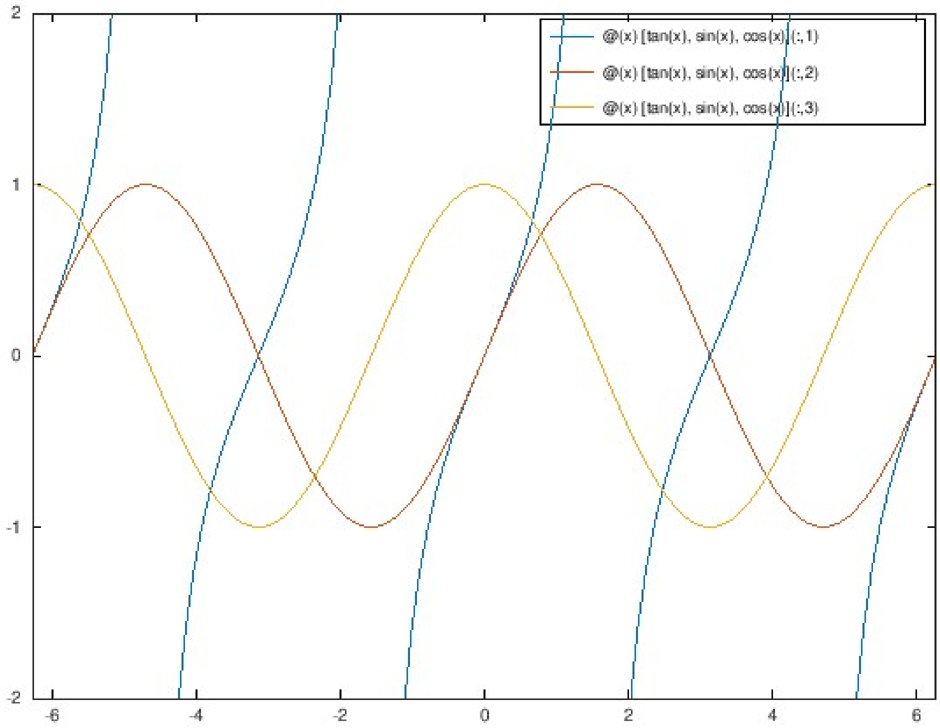
In [32]:

```
ezplot3(@cos, @sin, @(t) 2*t, [0,6*pi])      % 注意："@函数名"可理解为函数指针，注意与符号的区别
```



In [33]:

```
fplot(@(x)[tan(x),sin(x),cos(x)], [-2*pi 2*pi -2 2])
```



四则运算

In [ ]:

```
f*g
```

In [34]:

```
clear x y
syms x y
%s = ((x^3 + 3*y + 1)^2 + (x^2 - y^2)^3)
s = (x^2+y^2)^2 + (x^2-y^2)^2
```

```
s = (sym)

            2             2
  / 2    2\     / 2    2\
  \x  - y /   + \x  + y /
```

In [35]:

```
simplify(s)    % simple在octave中没有的
```

```
ans = (sym)

     4      4
  2*x  + 2*y
```

In [36]:

```
simplify(x^2 + 3*x + 1 + 2*x^2 - 3)  % 合并同类项等，不同的数学软件、不同软件版本输出结果可能会不一样
```

```
ans = (sym)

     2
  3*x  + 3*x - 2
```

In [41]:
```
factor(x^3 - 2*x^2 + x)
```

```
ans = (sym)

              2
  x*(x - 1)
```

In [46]:
```
factor(sym('93603961275884'))
```

```
ans = (sym)

     1   2          1       1
  151 *2 *2607181 *59441
```

In [ ]:
```
expand(s)
```

In [47]:
```
expand((x+1)^3)
```

```
ans = (sym)

    3       2
  x   + 3*x   + 3*x + 1
```

数值转换

In [ ]:
```
sym('3.14')    % 数值表达式3.14转成符号表达式
```

In [ ]:
```
sym(3.14) % 暂时也能工作
```

In [ ]:
```
sym(314)/100     % 整数就没警告，浮点数直接转换符号有危险！
```

In [ ]:
```
sym(3.14, 'r') %这样也不会有警告
```

In [50]:
```
gold = '(1+sqrt(5))/2 - 1';
eval(gold)  % evaluation
```

```
ans =    6.180339887498949e-01
```

In [ ]:
```

```

In [51]:
```
vpa(gold,18)
```

```
ans = (sym) 0.618033988749894848
```

In [ ]:
```

```

符号矩阵

求行列式的值det，秩rank，迹trace，上下三角矩阵tril,triu等与数值计算中的矩阵类似

In [ ]:
```
a = sym([1/x, sin(x), cos(x)+1; 9, exp(x), log(tanh(x))])
```

In [ ]:
```
A = [sin(x) cos(x); acos(x) asin(x)]
```

In [ ]:
```
det(A)
```

In [ ]:
```
mat_numer = [2/3, sqrt(2), 0.3323; ...
             1.4, -0.3,    exp(3.4); ...
             log(3), 1/0.243, sin(209.3)]
```

In [ ]:
```
mat_from_numer = sym(mat_numer,'r')
```

In [ ]:
```
mat_numer   % 转换过程有误差，所以警告！
```

In [ ]:
```
diff((1+3*x)/(x^2 + 3*x) + cos(x^3)*exp(-x^2))
```

矩阵分析其他功能举例

In [52]:
```
mat_sym = sym([1 2 3; 0 1 3; 0 0 2])
```

```
mat_sym = (sym 3x3 matrix)

  [1  2  3]
  [       ]
  [0  1  3]
  [       ]
  [0  0  2]
```

In [53]:
```
eig(mat_sym)
```

```
ans = (sym 3x1 matrix)

  [1]
  [ ]
  [1]
  [ ]
  [2]
```

In [54]:
```
diag(mat_sym)
```

```
ans = (sym 3x1 matrix)

  [1]
  [ ]
  [1]
  [ ]
  [2]
```

In [ ]:
```
triu(mat_sym)  % and tril(mat_sym)
```

```
In [55]: jordan(mat_sym)

         ans = (sym 3x3 matrix)

           [1  1  0]
           [       ]
           [0  1  0]
           [       ]
           [0  0  2]
```

```
In [ ]:
```

符号运算的运算效率远不如数值运算，但优点在于可以执行足够精度的计算。可以辅助人类执行较为繁琐的“简单计算”，从而解放人们对于数学计算的高强度劳动。下面我们再看一个方法较为简单，但计算过程略为繁琐的问题：

例：求λ使得如下齐次线性方程组有非零解

$$\begin{cases} (1-\lambda)x_1 - 2x_2 + 4x_3 & = & 0 \\ 2x_1 + (3-\lambda)x_2 + x_3 & = & 0 \\ x_1 + x_2 + (1-\lambda)x_3 & = & 0 \end{cases}$$

```
In [57]: syms lambda
```

```
In [59]: mat_coeff = [1-lambda     -2        4; ...
                          2     3-lambda     1; ...
                          1         1     1-lambda]

         mat_coeff = (sym 3x3 matrix)

           [1 - lambda      -2          4     ]
           [                                  ]
           [    2       3 - lambda      1     ]
           [                                  ]
           [    1            1      1 - lambda]
```

```
In [60]: det(mat_coeff)

         ans = (sym)

                                   2
           lambda + (1 - lambda) *(3 - lambda) - 3
```

```
In [61]: factor(det(mat_coeff))

         ans = (sym) -lambda*(lambda - 3)*(lambda - 2)
```

故当$\lambda = 0, 2, 3$中任意一值时，原方程组具有非零解。

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

In [ ]:

```

```

In [ ]:

```

```

2. 求解（非线性）代数方程

方案一：符号计算

In [74]:

```
syms p x r;
solve(p*sin(x)==r)
```

```
ans = (sym)

    r
  ------
  sin(x)

```

In [76]:

```
solve(x^2 + 3*x*r + r == 3, x^2 -4*x + 3 == 0) %
```

```
ans =
{
  [1,1] =

    scalar structure containing the fields:

      x =

        <class sym>

      r =

        <class sym>


  [1,2] =

    scalar structure containing the fields:

      x =

        <class sym>

      r =

        <class sym>


}
```

In [73]:

```
solve(x^2 + x*r + r - 3, x^2 -4*x + 3 )   %% 与前一种方式等价
```

```
error: Python exception: NonSquareMatrixError
    occurred at line 2 of the Python code block:
    return x**y
error: called from
    pycall_sympy__ at line 178 column 7
    mpower at line 76 column 5
```

方案二：　数值求根

fsolve和fzero

In [77]:

```
%function f = equation_1(x)
%   f = [sin(x(1)) + x(2) + x(3)^2*exp(x(1)) - 4;  x(1) + x(2)*x(3); x(1)*x(2)*x(3)
+ 2]
%endfunction
equation1 = @(x) [sin(x(1)) + x(2) + x(3)^2*exp(x(1)) - 4;  ...
                  x(1) + x(2)*x(3); ...
                  x(1)*x(2)*x(3) + 2];
```

In [78]:

```
[x,fval] = fsolve(equation1, [1., 1., 1.])
```

```
warning: matrix singular to machine precision, rcond = 3.80012e-18
warning: called from
    fsolve>__dogleg__ at line 532 column 5
    fsolve at line 351 column 11
x =

   1.414213563042528  -1.370106997584169   1.032192059769083

fval =

   1.377543945579873e-08
  -8.979097465555697e-10
  -3.163278083917476e-09
```

In [ ]:

In [12]:

```
fzero(equation1, [1., 1., 1.])   % 只能求解一元方程  help fzero
```

```
error: x(2): out of bound 1
error: called from
    @<anonymous>
    fzero at line 145 column 6
```

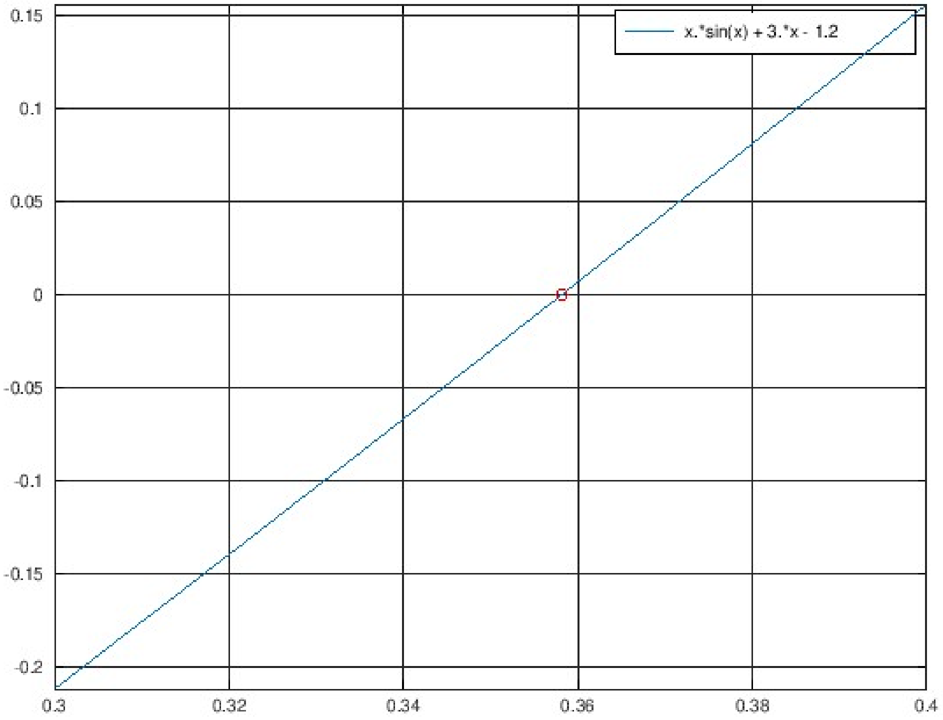
In [20]:

```
[x,fval] = fzero(@(x) x*sin(x) + 3*x - 1.2, 1.0)
```

```
x =  0.35815
fval =   -1.9984e-15
```

In [35]:

```
fplot('x*sin(x) + 3*x - 1.2', [0.3,0.4]); grid on; axis tight;
hold on; plot(0.35815,0,'ro'); hold off;
```



其他数值方法： 牛顿法、二分法（略）

In [ ]:

In [ ]:

In [ ]:

In [ ]:

3. 求解微分方程

In [79]:

```
syms y(x) w
DE = diff(y,2) + w^2 *y == 0     % a harmonic equation
```

```
DE = (sym)

                 2
   2            d
  w *y(x) + ---(y(x)) = 0
                 2
              dx
```

```
In [80]: dsolve(DE)
```

```
ans = (sym)

                -I*w*x        I*w*x
  y(x) = C1*e       + C2*e
```

参考octave-forge官方文档网页更方便: https://octave.sourceforge.io/symbolic/function/@sym/dsolve.html (https://octave.sourceforge.io/symbolic/function/@sym/dsolve.html)

```
In [ ]:
```

```
In [ ]: help @sym/dsolve
```

```
In [ ]:
```

```
In [ ]:
```

初值问题

```
In [ ]:
```

```
In [86]: syms y(x);
```

方案1-符号计算

```
In [87]: [sol, classify] = dsolve(diff(y,1)==-2*y+2*x*(x+1), y(0)==1.0)
```

```
sol = (sym)

           / 2  2*x     \  -2*x
  y(x) = \x *e    + 1/*e

classify = 1st_linear
```

```
In [88]: ezplot(sol)
```

```
error: function_handle: python codegen failed: y(x) == (x.^2.*exp(2*x) + 1).*exp
(-2*x)
error: called from
    function_handle at line 166 column 9
    ezplot at line 117 column 17
```

遗憾的是，这里的sym是一个表达式，不是函数！故无法用ezplot(sol)画图， 有没有其他办法？PS: 在matlab的符号工具箱中，下面这段是可以工作的（Octave不行）

In [81]:

```
syms x(t) y(t)
A=diag([-1,2]); % creates diagonal matrix
Y = [x; y];
odes = diff(Y) == A*Y;
[xSol(t), ySol(t)] = dsolve(odes, x(0)=1, y(0)=1);
xSol(t) = simplify(xSol(t))
ySol(t) = simplify(ySol(t))
ezplot(xSol(t),ySol(t))
```

```
warning: Classification of systems of ODEs is currently not supported
warning: called from
    dsolve at line 176 column 5
error: Python exception: AttributeError: 'float' object has no attribute 'rhs'
    occurred at line 5 of the Python code block:
    ics2[s.lhs] = s.rhs
error: called from
    pycall_sympy__ at line 178 column 7
    dsolve at line 187 column 8
error: 'xSol' undefined near line 1 column 20
error: 'ySol' undefined near line 1 column 20
error: 'xSol' undefined near line 1 column 8
```

In [ ]:

```

```

方案2-数值解法

RungeKutta方法是求解微分方程的有效方法

例1：一个简单的初值问题

$$\frac{dy}{dx} = -2y + 2x(x+1), \qquad x \in [0, 0.5], y(0) = 1.$$

In [89]:

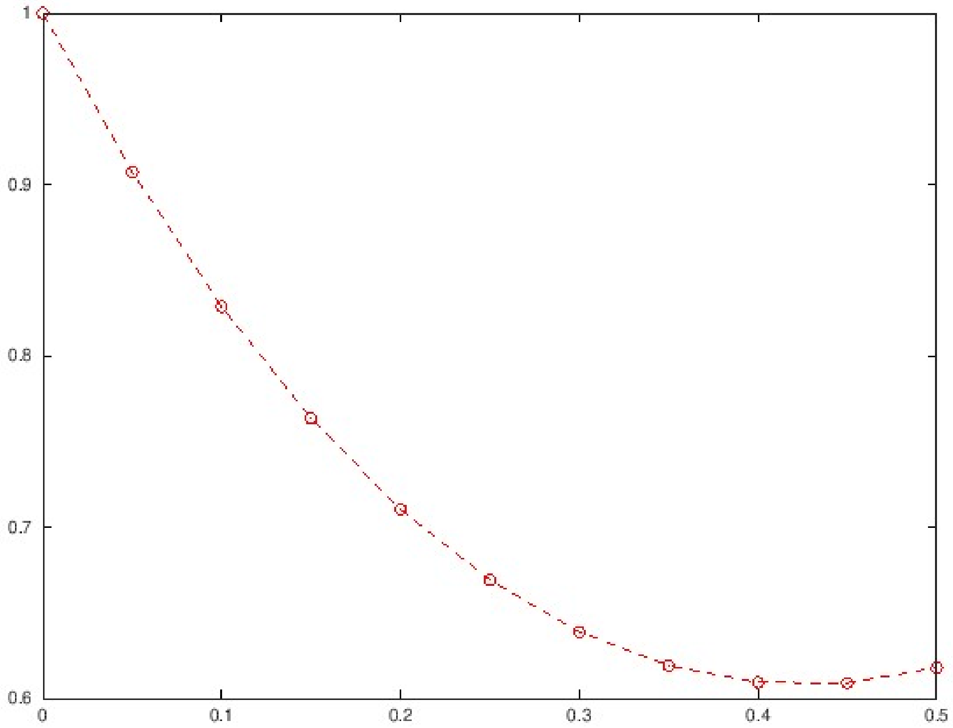
```
function f = rhs(x,y)
 f = -2*y+2*x*(x+1);
endfunction
```

In [90]:

```
%[x2,y2] = ode45(inline('-2*y+2*x*(x+1)'),[0,0.5],1);
[x2,y2] = ode45('rhs',[0,0.5],1);
```

In [92]:

```
plot(x2,y2,'ro--')
```



In [87]:

```
% this is the same with above
% [x2,y2] = ode45(@(x,y) -2*y+2*x*(x+1),[0,0.5],1);
```

In [81]:

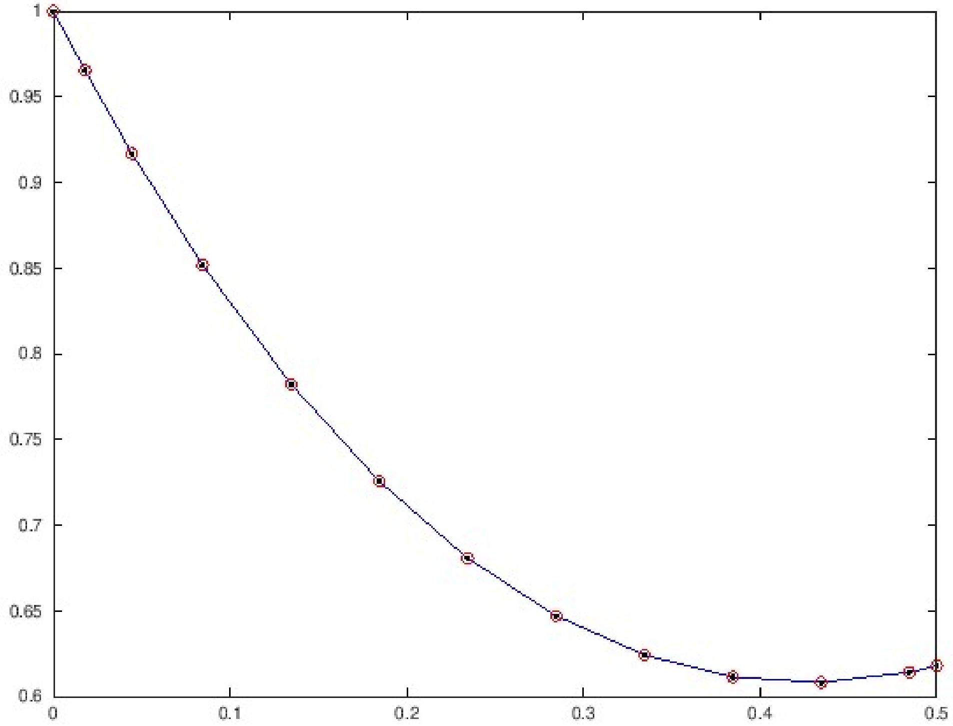
```
% evaluate the analytic solution
% yy2 = (x2.*x2.*exp(2*x2) + 1.0).*exp(-2*x2);
```

一般来说，ode45比ode23的积分段少，从而运算速度更快些。

In [121]:

```
[x23,y23] = ode23(@(x,y) -2*y+2*x*(x+1),[0,0.5],1);
```

```
In [88]: plot(x2,y2,'b-',x2,y2,'ro') % ,x2, yy2, 'g*'  % ,x23,y23, 'k.'
```



```
In [ ]:
```

例：考虑刚性问题

$$\begin{pmatrix} u' \\ v' \end{pmatrix} = \begin{pmatrix} -2 & 1 \\ 998 & -999 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} + \begin{pmatrix} 2\sin x \\ 999(\cos x - \sin x) \end{pmatrix}$$

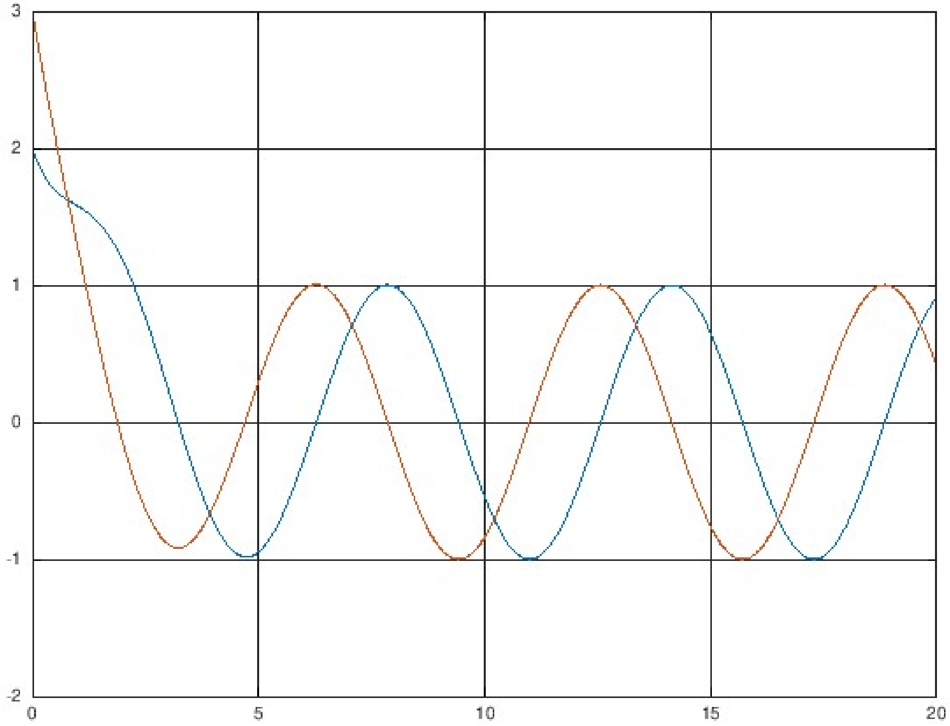
```
In [93]: syms x y u v;
         func_rhs = @(x,y) [-2, 1; 998,-999]*y + [2*sin(x); 999*(cos(x) - sin(x)) ];
```

```
In [94]: [x23,y23] = ode23(func_rhs, [0,20], [2,3]);
```

```
In [96]: size(y23)
```

```
ans =

   7956      2
```

In [95]:
```
plot(x23,y23);grid on
```

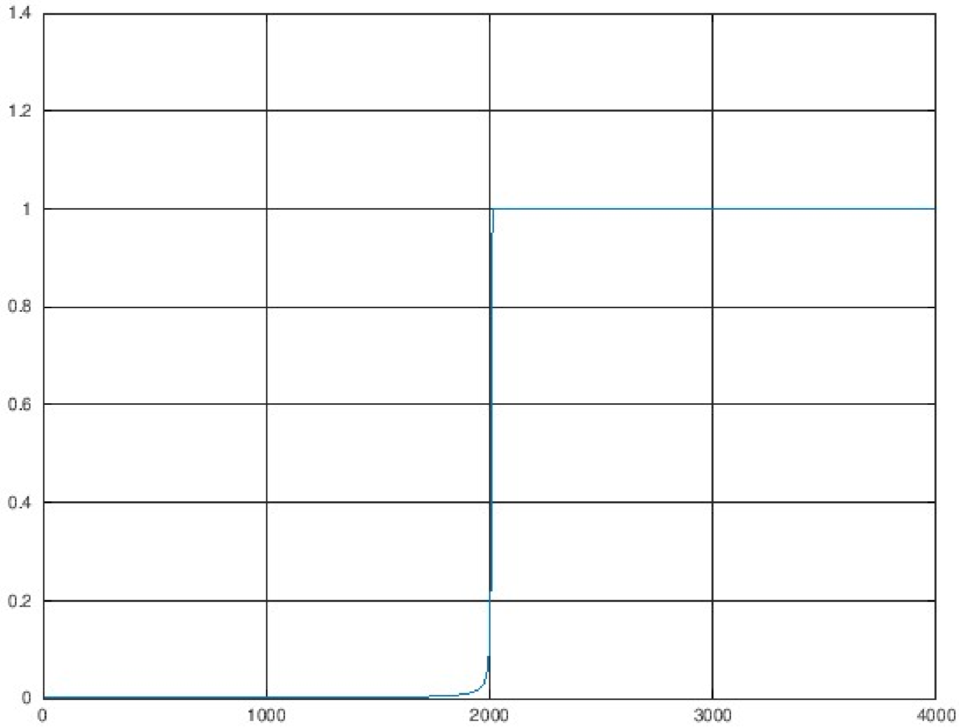


一般来说："A problem is stiff if the solution being sought is varying slowly, but there are nearby solutions that vary rapidly, so the numerical method must take small steps to obtain satisfactory results." 研究如下的火焰蔓延问题(δ越小在靠近稳定解时刚性越大,真解出现blowup(爆炸，奇点))：

$$y' = y^2 - y^3, y(0) = \delta, x \in [0, 2/\delta]$$

In [101]:
```
func_fire = @(x,y) y*y*(1-y);
delta = 0.0005;    % try delta = 0.5, 0.2, 0.1, 0.05, 0.01, 0.005, 0.001
[xfire,yfire] = ode23(func_fire,[0,2/delta],delta);
```

In [102]:
```
plot(xfire,yfire);grid on;
```



例1：Lorenz吸引子（典型初值问题非线性）

$$\begin{pmatrix} y_1' \\ y_2' \\ y_3' \end{pmatrix} = \begin{pmatrix} -\beta & 0 & y_2 \\ 0 & -\sigma & \sigma \\ -y_2 & \rho & -1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

参数选取为$\sigma = 10, \rho = 28, \beta = \frac{8}{3}$.

In [133]:
```

```

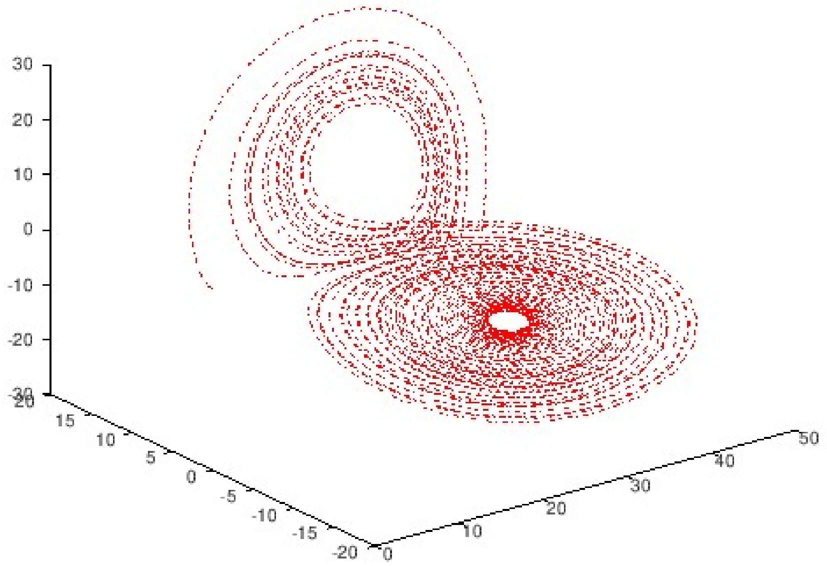
In [138]:
```
function yp = yprime_lorenz(t,y)
rho = 28;  sigma = 10;    beta = 8/3;
A = [ -beta     0      y(2);...
         0  -sigma   sigma ;...
      -y(2)    rho     -1   ];
yp = A*y;
endfunction
```

In [145]:
```
[t_l,y_l] = ode23(@yprime_lorenz,[0,50],[0;0;3]);
```

In [141]:
```
size(y_l)
```

```
ans =

   59    3
```

In [146]:
```
plot3(y_l(:,1),y_l(:,2),y_l(:,3), 'r-.') % think about how to animate it?
```



练习：采用不同的参数，有没有更多的吸引子？

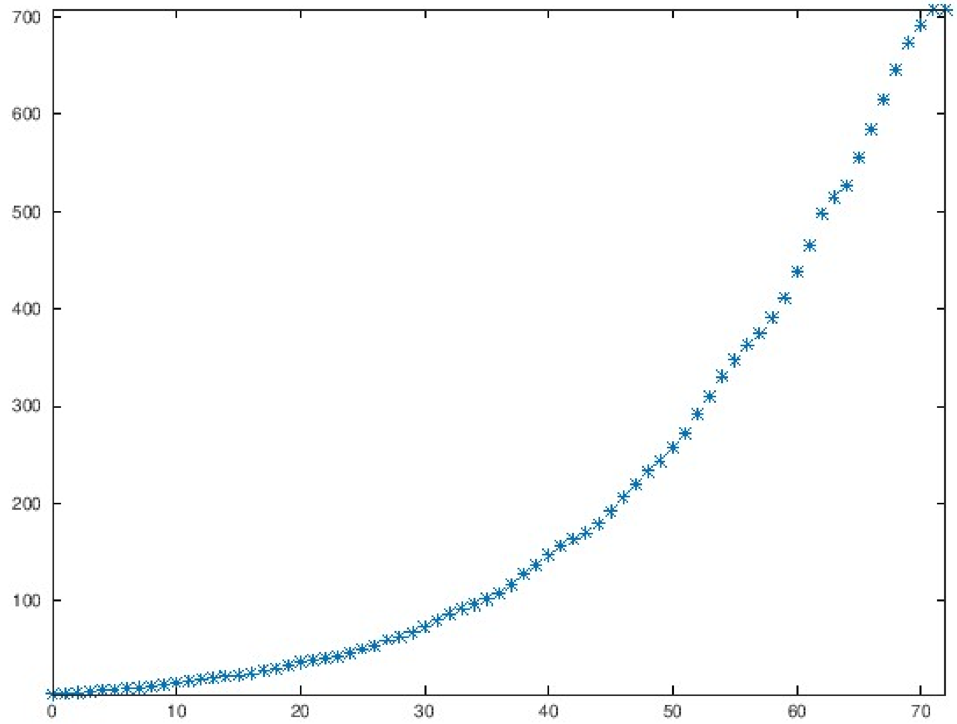
In [ ]:
```

```

例2：Simulation of COVID-19

bla... bla...

In [1]:
```
%% COVID-19 data in Brazil
xx = linspace(0,72,73)';
yy = [4256;4681;5861;7011;8165;9216;10431;11298; 12341;14152;16238;18176;19943;21042;22625;23955; ...
     25758;29015;30891;34485;36925;39144;40814;43592; 46348;50512;54043;59479;63328;67446;73235;80246; ...
     87187;92630;96559;101826;108620;116299;127389;137309; 147003;156604;163510;170021;179457;192081;206507;220291; ...
    233511;244052;257396;271885;291579;310087;330890;347398; 363211;374898;391222;411821;438238;465166;498440;514849; ...
    526447;555383;584016;614941;646006;673587;691758;707412;707412]/1000;
```

```
In [2]: plot(xx,yy,'*'); axis([0 72 4.2 708])
```



SIR模型是在传染病学研究中较为经典的模型，其将人群分为易感人群（Susceptible）、感染人群（Infected）以及恢复（Recovered）人群，数量分别记为S(t); I(t)以及R(t)。 经过一些假设和数学推导，可以获得如下常微分方程组

$$\begin{cases} S'(t) &=& -\beta\frac{I(t)}{N(t)}S(t), \\ I'(t) &=& \beta\frac{I(t)}{N(t)}S(t)-\gamma I(t), \\ R'(t) &=& \gamma I(t) \end{cases}$$

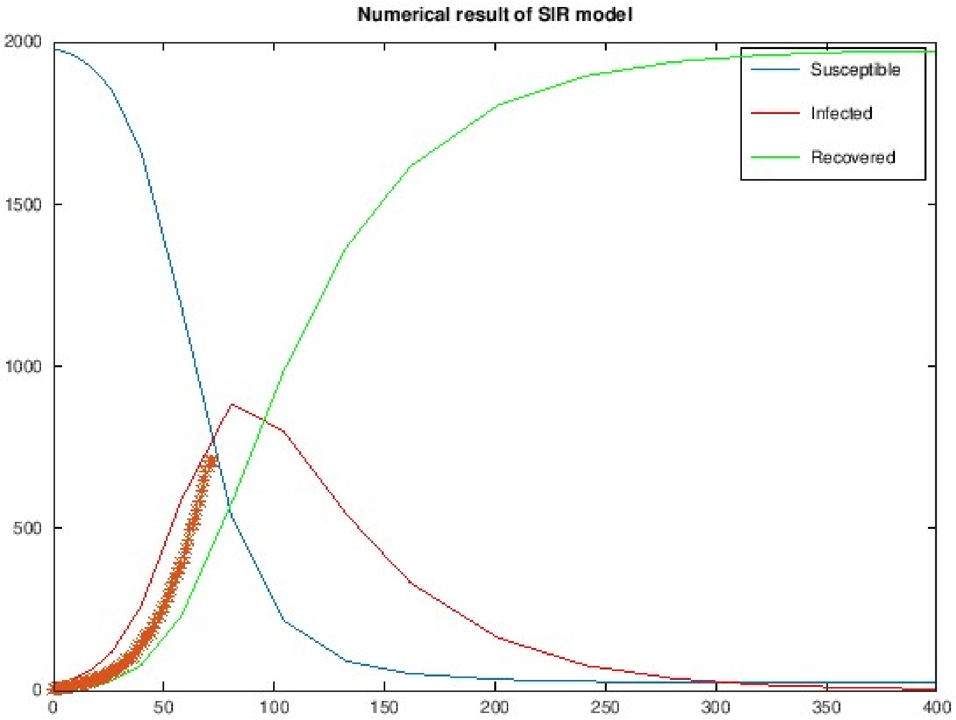
初始条件为$S(0)=S_0, I(0)=I_0, R(0)=0$事先给出，其中应满足病患数量守恒的相容性条件$S_0+I_0=N$。这里我们假设总人口数不变，即

$$N(t)=S(t)+I(t)+R(t)\equiv N.$$

下面是一个参数取为$(N=2000;\beta=0.08;\gamma=0.04;I_0=20;S_0=N-I0=1980)$的算例

In [8]:

```
N = 2000;     % total population
beta = 0.09; gamma = 0.02;       % grow rate !!
SIRfunc = @(t, y)  [-beta*y(2)/N*y(1); beta*y(2)/N*y(1)-gamma*y(2);   gamma*y(2)];
t0 = 0; tfinal = 400;
% initial conditions
I0 = 20; S0 = N-I0;  R0 = 0;     %  RO is also important
y0 = [S0; I0; R0];
%% solve ODE
[t, y] = ode45(SIRfunc,[t0,tfinal],y0);
%% visualization
plot(t,y(:,1),'-',t,y(:,2),'r-',t,y(:,3),'g-','LineWidth',3);
legend('Susceptible','Infected','Recovered')
title('Numerical result of SIR model')
%% append the observed data, how far it is?
hold on;  plot(xx,yy,'*');
```



1.练习: How to adjust β, γ and R_0 to match the simulation and real data?

2.找一找相关技术文章，解释你观察到的现象

3.如果将观测数据换成英国、法国、西班牙、意大利、美国等公布的数据，又有怎样的现象？

4.是否有很好的方式可以用于参数的自动选取/学习？

In [ ]:

此外，对COVID-19的放射学标记物的研究是一个活跃的研究领域,从X射线图像检测由COVID-19引起的冠状肺炎是目前最流行的做法之一。 COVID-19肺部扫描数据集目前有限，但是用于该项目的最佳数据集来自COVID-19开源数据集：

- 相关数据集：
    - [1] (https://github.com/ieee8023/covid-chestxray-dataset) https://github.com/ieee8023/covid-chestxray-dataset (https://github.com/ieee8023/covid-chestxray-dataset)
    - [2] (https://github.com/ajsanjoaquin/Pneumothorax) https://github.com/ajsanjoaquin/Pneumothorax (https://github.com/ajsanjoaquin/Pneumothorax)
- 开源代码：
    - [1] (https://github.com/ajsanjoaquin/COVID-19-Scanner) https://github.com/ajsanjoaquin/COVID-19-Scanner (https://github.com/ajsanjoaquin/COVID-19-Scanner)
    - [2] (https://github.com/ilmimris/ct-covid19-model) https://github.com/ilmimris/ct-covid19-model (https://github.com/ilmimris/ct-covid19-model)

感兴趣的读者可下载相关数据集开展学习研究，但不在本次作业要求范围内。

In [ ]: