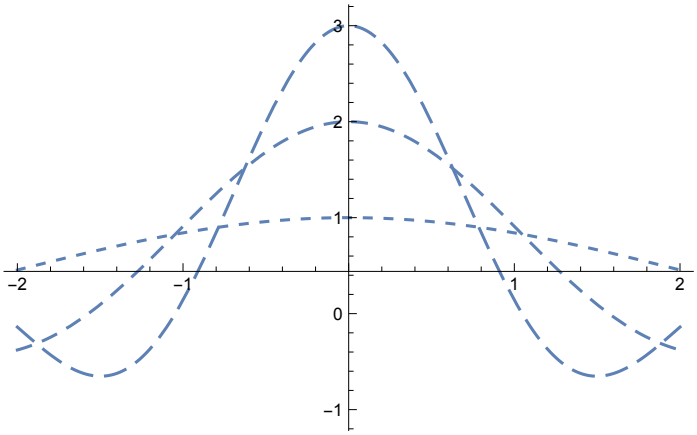
In[1]:=
```
(*III.GRAPHICS:TOPICS:
  Two-dimensional Three-dimensional and color Two-dimensional:Main functions:Plot,
PlotRange,Show Plot[f,{x,xmin,xmax}] generates a plot of f as a
  function of x from xmin to xmax.Plot[{f1,f2,...},{x,xmin,xmax}]
  plots several functions fi.PlotRange is an option for
  graphics functions that specifies what points to include in a
  plot.Show[graphics,options] displays two-and three-dimensional graphics,
using the options specified.Show[g1,g2,...] shows several plots
  combined.Use with Evaluate for complicated functions Dashing of
  lines Dashing[{r1,r2,...}] is a two-dimensional graphics directive
  which specifies that lines which follow are to be drawn dashed,
with successive segments of lengths r1,r2,... (repeated cyclically).The
 ri is given as a fraction of the total width of the graph.*)
```

In[2]:=
```
plo[n_] := Plot[Sin[n x] / x, {x, -2, 2}, PlotStyle → Dashing[{0.01 * n, 0.02}]]
```

In[3]:=
```
Show[Table[plo[n], {n, 1, 3}], PlotRange → {-1, 3}]
(*Note:to plot Tables of more complex functions,use Plot[Release[Table[...]]]*)
```

Out[3]=

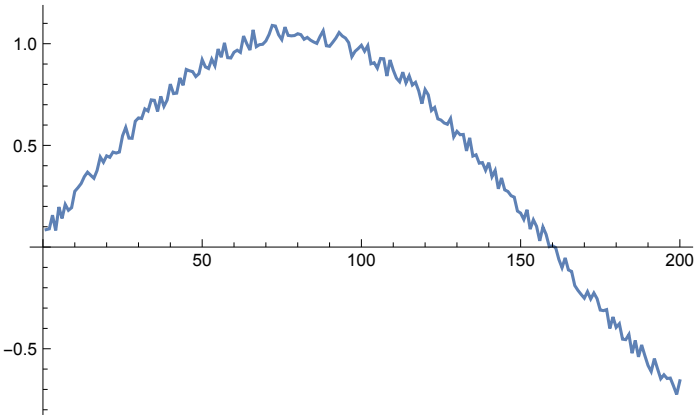


In[4]:=
```
(*AxesLabel→{"x","y"} will label each axes.Greek letters:α,
etc.(in Notebook can copy from pallet)
*)
```

In[5]:=
```
(*Plotting discrete data points:ListPlot[{y1,y2,...}] plots a
   list of values.The x coordinates for each point are taken to be 1,
2,....ListPlot[{{x1,y1},{x2,y2},...}] plots a list of values
 with specified x and y coordinates. *)
```

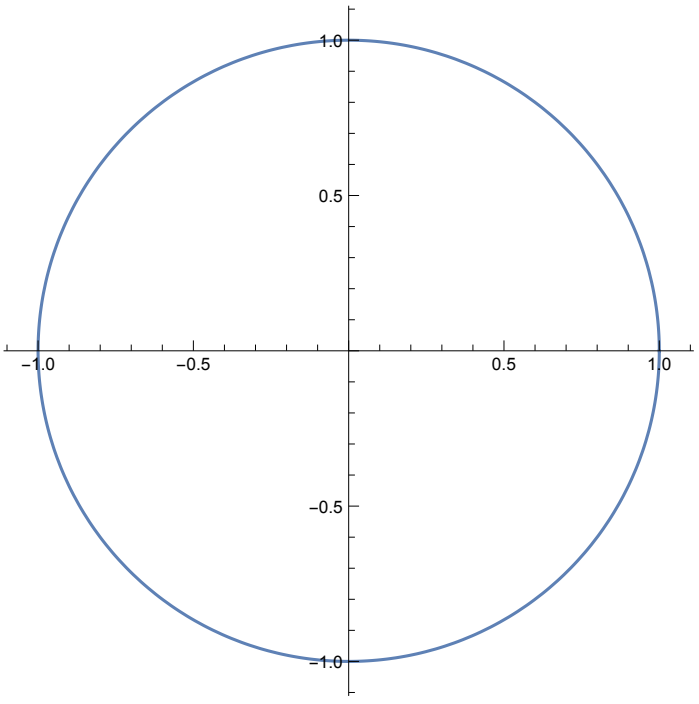
In[6]:=
```
list = Table[Sin[i / 50.] + .1 * Random[], {i, 200}];
ListPlot[list, Joined → True]
```

Out[6]=



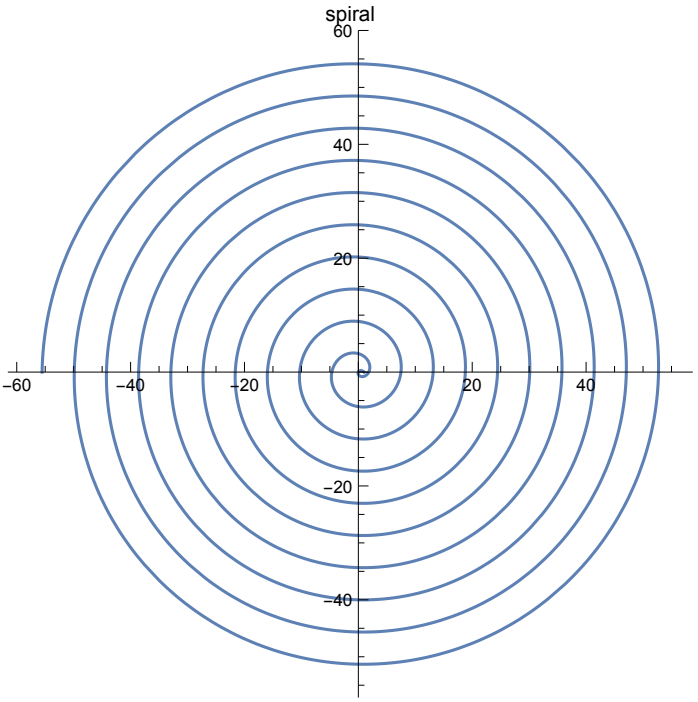
In[7]:=
```
(*Parametric plot:ParametricPlot[{fx,fy},{t,tmin,tmax}] produces a
   parametric plot with x and y coordinates fx and fy generated as a
   function of t.ParametricPlot[{{fx,fy},{gx,gy},...},{t,tmin,tmax}]
   plots several parametric curves.*)
x[phi_] := Cos[phi]; y[phi_] := Sin[phi];
ParametricPlot[{x[phi], y[phi]}, {phi, 0, 2 Pi}]
```

Out[8]=



In[9]:=
```
Clear[r]; r = 1 - .9 phi; x[phi_] := r Cos[phi]; y[phi_] := r Sin[phi];
spir = ParametricPlot[{x[phi], y[phi]}, {phi, 0, 20 Pi}, PlotLabel → "spiral"]
```

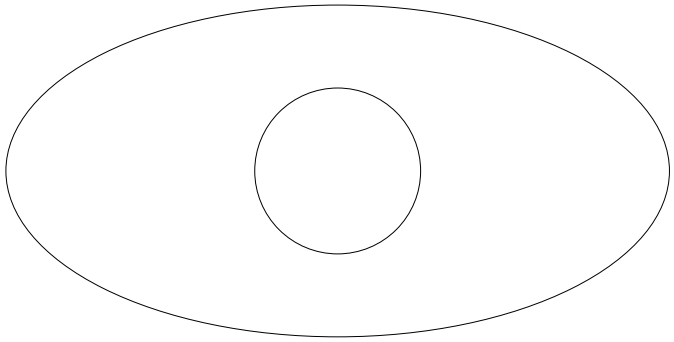
Out[10]=




```
(*if we like what we see and want to create,
e.g.a postscript file,t.ps*) disp := Export["t.ps", #, "EPS"] &
(*now disp[spir] will create t.ps,as a GOOD postscript*)
(*Graphics
 Primitives:Line[{pt1,pt2,...}] is a graphics primitive which represents a line
    joining a sequence of points.Point[coords] is a graphics primitive
    that represents a point.Circle[{x,y},r] is a two-dimensional graphics
    primitive that represents a circle of radius r centered at the point x,
y.Circle[{x,y},{rx,ry}] yields an ellipse with semi-
 axes rx and ry.Circle[{x,y},r,{theta1,theta2}] represents a circular arc.Polygon,
etc. -use with Graphics,similar to Arrow;
HW:give graphic examples*).
```

In[12]:=
```
Graphics[{Circle[{0, 0}, 1], Circle[{0, 0}, {4, 2}]}]
```

Out[12]=

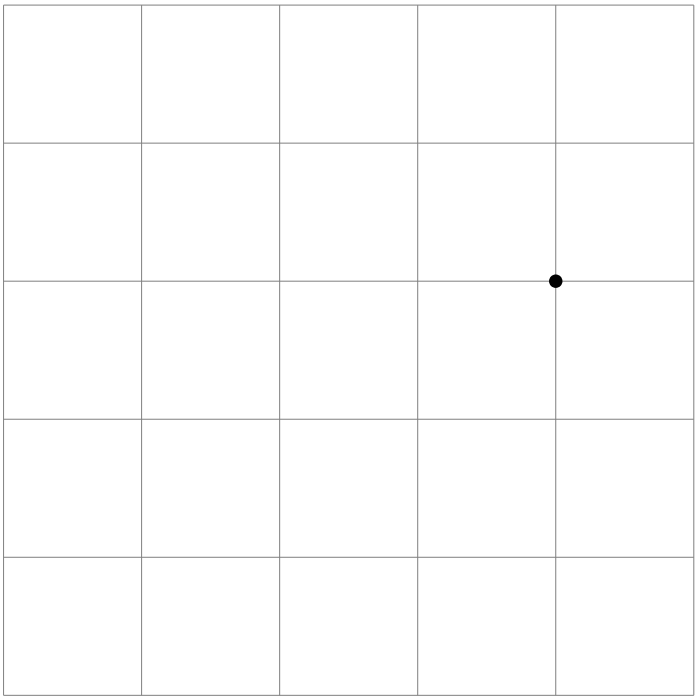


In[13]:=
```
(*COMPLEX NUMBERS*)
```

In[14]:=
```
i = I;
```

In[15]:=
```
z = 2 + i; Show[Graphics[{PointSize[Large], Point[{Re[z], Im[z]}]}],
 PlotRange → {{-2, 3}, {-2, 3}}, GridLines -> Automatic]
```

Out[15]=



In[16]:=
```
sol = Solve[w^2 == -1, w]
```

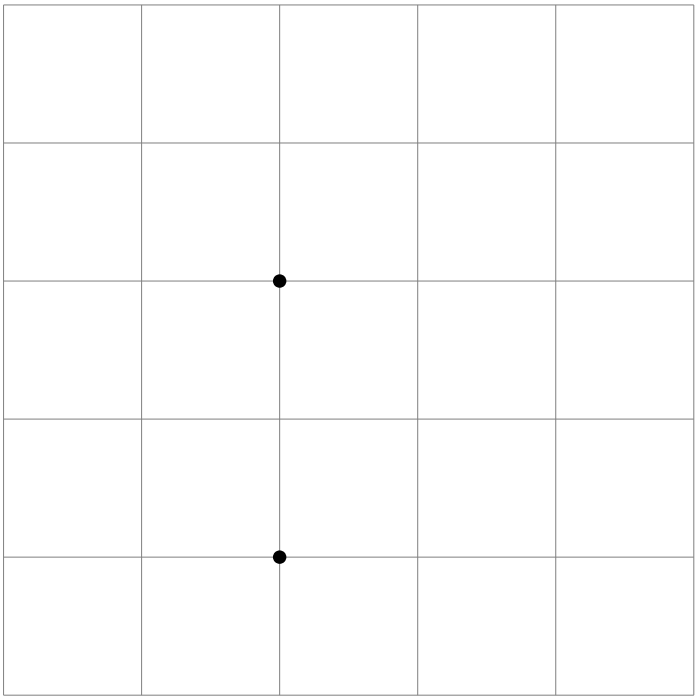
Out[16]=
```
{{w → -i}, {w → i}}
```

In[17]:=
```
roots = {Re[w], Im[w]} /. sol
```

Out[17]=
```
{{0, -1}, {0, 1}}
```

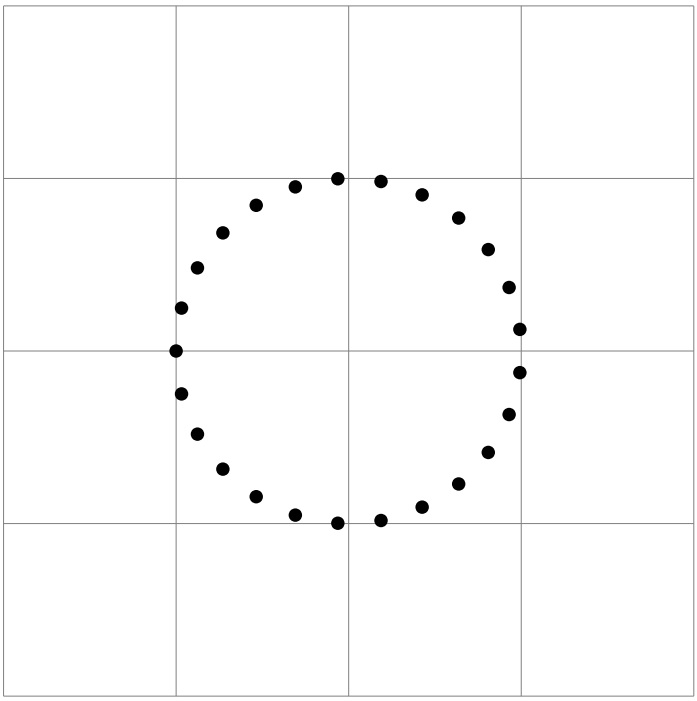
In[18]:=
```
Show[Graphics[{PointSize[Large], Point /@ roots}],
 PlotRange → {{-2, 3}, {-2, 3}}, GridLines -> Automatic]
```

Out[18]=



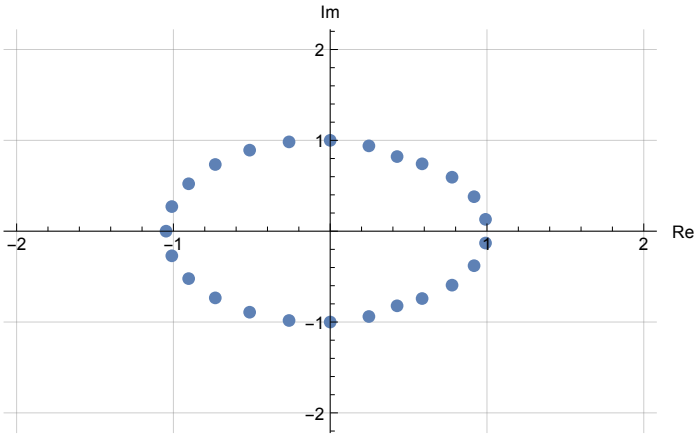
In[19]:=
```
sol = Solve[w^25 == -1, w]; roots = {Re[w], Im[w]} /. sol;
Show[Graphics[{PointSize[Large], Point /@ roots}],
 PlotRange → {{-2, 2}, {-2, 2}}, GridLines -> Automatic]
```

Out[19]=



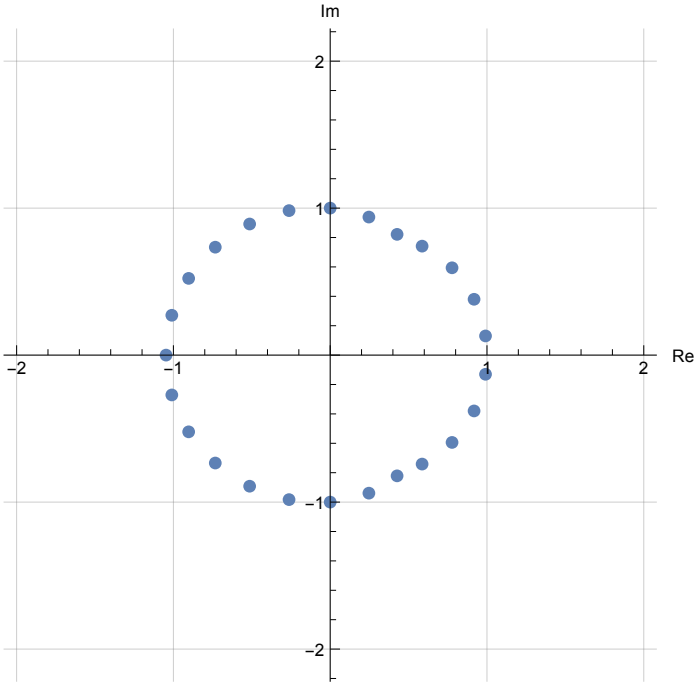
In[20]:=
```
sol = NSolve[w^25 - w + w^2 == -1, w]; roots = {Re[w], Im[w]} /. sol;
Show[ListPlot[roots], PlotRange → {{-2, 2}, {-2, 2}},
 GridLines -> Automatic, AxesLabel → {"Re", "Im"}]
```

Out[20]=



In[21]:=
```
Show[%, AspectRatio → 1]
```

Out[21]=



In[22]:=

```
(*PLOTTING SOLUTIONS OF ODE*)
```

In[23]:=
```
Clear[x, y]; DSolve[{D[y[x], x] == x, y[0] == 1}, y[x], x]
```

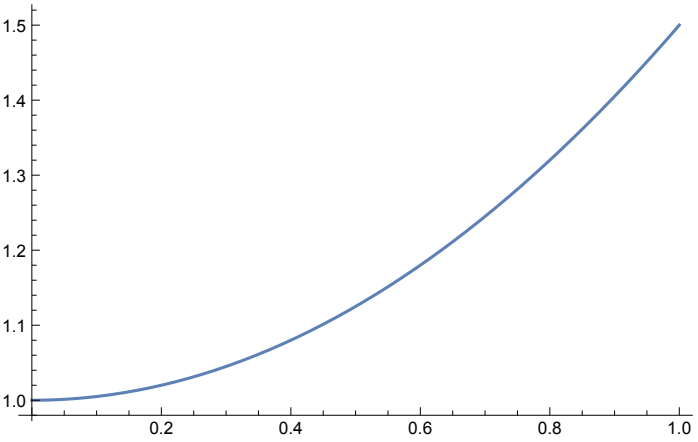
Out[23]= $\left\{\left\{y[x] \to \frac{1}{2}\left(2 + x^2\right)\right\}\right\}$

In[24]:=
```
sol = %[[1]]
```

Out[24]= $\left\{y[x] \to \frac{1}{2}\left(2 + x^2\right)\right\}$

In[25]:=
```
Plot[y[x] /. sol, {x, 0, 1}]
```

Out[25]=



In[26]:=
```
(*2nd order Newtons eq-n with Hooks force;m=1=k*)
```

In[27]:=
```
Clear[x, p, t]; eqs = {D[x[t], t] == p[t], D[p[t], t] == -x[t] - .4 p[t]}
```

Out[27]=
```
{x'[t] == p[t], p'[t] == -0.4 p[t] - x[t]}
```

In[28]:=
```
ic = {x[0] == 1, p[0] == 0}
```

Out[28]=
```
{x[0] == 1, p[0] == 0}
```

In[29]:=
```
alleqs = Join[eqs, ic]
```

Out[29]=
```
{x'[t] == p[t], p'[t] == -0.4 p[t] - x[t], x[0] == 1, p[0] == 0}
```

In[30]:=
```
sol = DSolve[alleqs, {x[t], p[t]}, t]
```

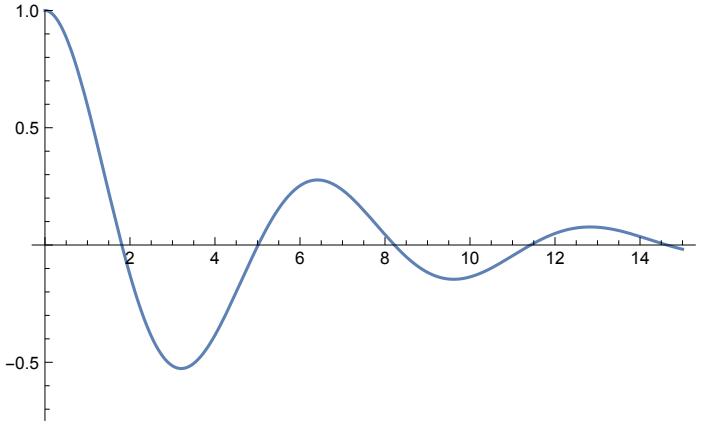
Out[30]=
```
{{p[t] → e^(-0.2 t) ((0. - 2.13821 × 10^-50 i) Cos[0.979796 t] -
        (1.02062 - 2.80417 × 10^-17 i) Sin[0.979796 t]), x[t] → (1. - 3.42114 × 10^-49 i) e^(-0.2 t)
     ((1. + 0. i) Cos[0.979796 t] + (0.204124 - 2.83279 × 10^-18 i) Sin[0.979796 t])}}
```

In[31]:=
```
sol = %[[1]] // Chop
```

Out[31]=
```
{p[t] → -1.02062 e^(-0.2 t) Sin[0.979796 t],
 x[t] → 1. e^(-0.2 t) (1. Cos[0.979796 t] + 0.204124 Sin[0.979796 t])}
```

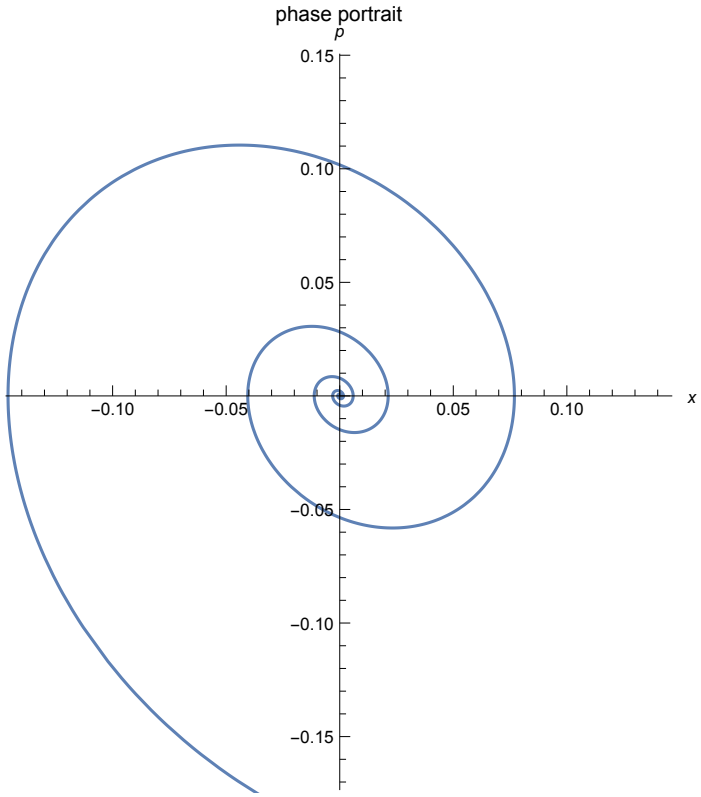
In[32]:=
```
Plot[x[t] /. sol, {t, 0, 15}, PlotRange → {-.75, 1}]
```

Out[32]=



In[33]:=
```
ParametricPlot[{x[t], p[t]} /. sol, {t, 0, 40},
 AxesLabel → {x, p}, PlotLabel → "phase portrait"]
```

Out[33]=

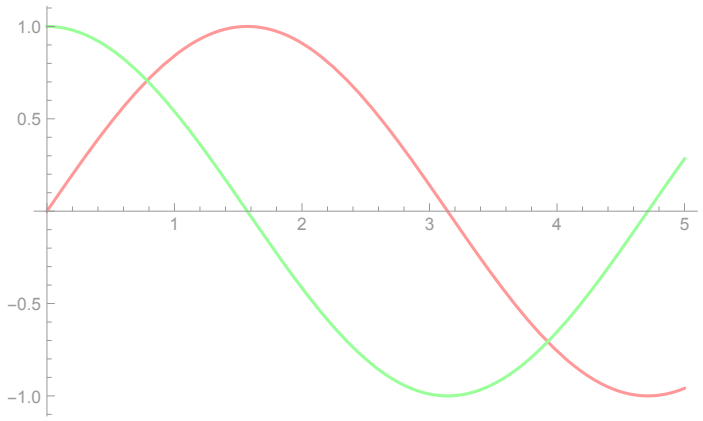


```
(*Advanced:
 3-Dimensional graphics and colors"Plot3D[f, {x, xmin, xmax}, {y, ymin, ymax}]
     generates a three-dimensional plot of f as a function of x and y
"

   Color:"RGBColor[red, green, blue] is a graphics directive which specifies
    that graphical objects which follow are to be displayed, if
    possible, in the color given." ("knows" main colors,Red,Blue,etc.*)

(*in 2-D*)
Plot[{Sin[x], Cos[x]}, {x, 0, 5}, PlotStyle → {Red, Green}]
(*HW:try this*)
```

Out[34]=



In[35]:=

```
? Plot3D
```

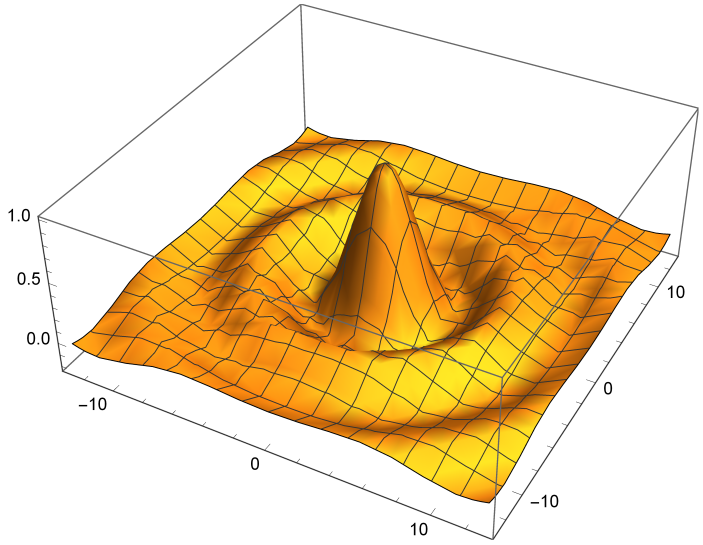
Plot3D[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a three-dimensional plot of f as a function of x and y.

Plot3D[{f_1, f_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several functions.

Plot3D[…, {x, y} ∈ reg] takes variables {x, y} to be in the geometric region reg. ≫

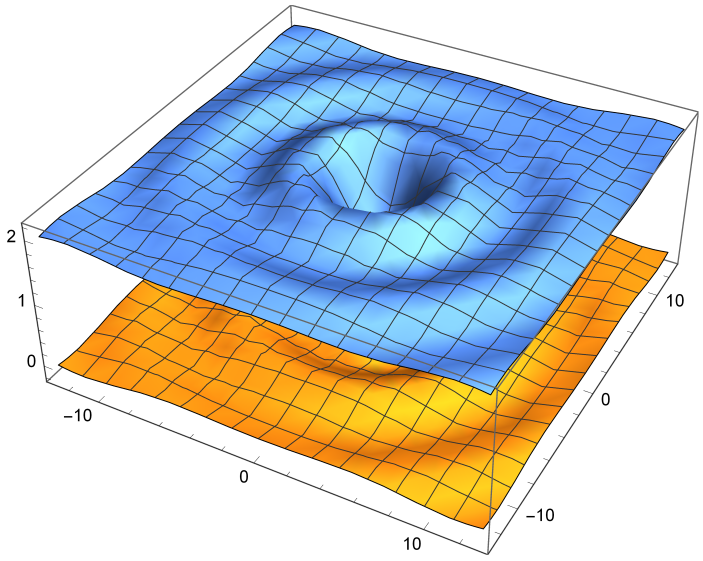
In[36]:=
```
Clear[r]; r = Sqrt[x^2 + y^2];
Plot3D[Sin[r] / r, {x, -13, 13}, {y, -13, 13}, PlotRange → All]
```

Out[37]=



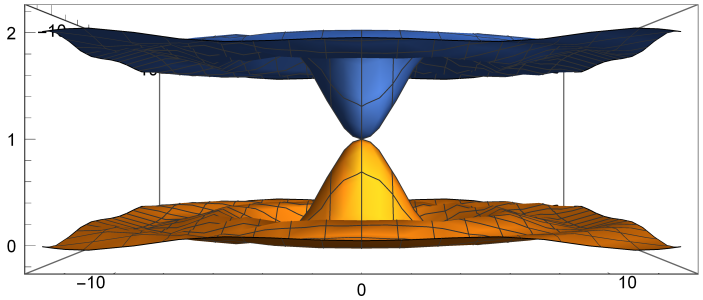
In[38]:=
```
Plot3D[{Sin[r] / r, 2 - Sin[r] / r}, {x, -13, 13}, {y, -13, 13}, PlotRange → All]
```

Out[38]=



In[39]:=
```
Show[%, ViewPoint -> Front]
```

Out[39]=



In[40]:=
```
? ParametricPlot3D
```

ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}] produces a three–dimensional space curve parametrized by a variable u which runs from u_{min} to u_{max}.

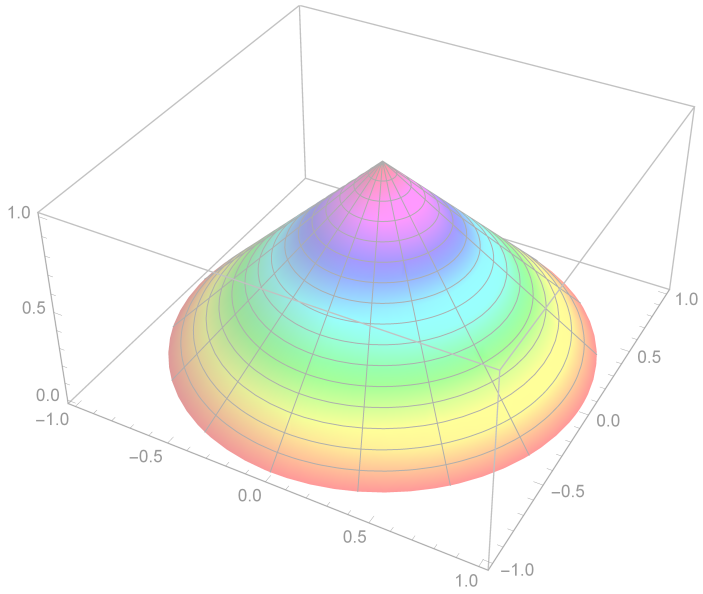
ParametricPlot3D[{f_x, f_y, f_z}, {u, u_{min}, u_{max}}, {v, v_{min}, v_{max}}] produces a three–dimensional surface parametrized by u and v.

ParametricPlot3D[{{f_x, f_y, f_z}, {g_x, g_y, g_z} …} …] plots several objects together.

ParametricPlot3D[…, {u, v} ∈ reg] takes parameters {u, v} to be in the geometric region reg. ≫

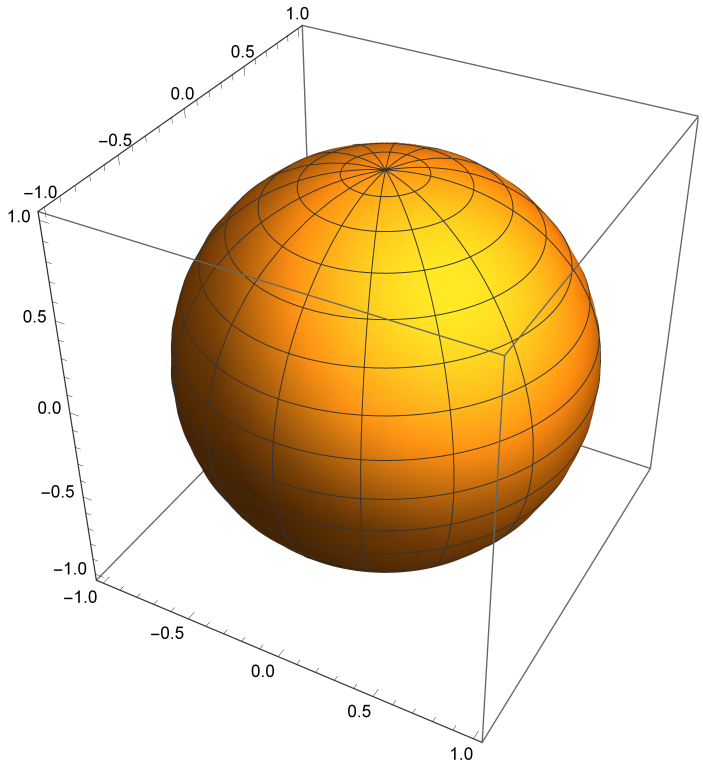
```
cone = ParametricPlot3D[{Cos[u] * v, Sin[u] * v, 1 - v},
  {u, 0, 2 * Pi}, {v, 0, 1}, PlotPoints → {60, 12}, ColorFunction → Hue]
```

Out[41]=



In[42]:=
```
sphere = ParametricPlot3D[{Cos[u] * Sin[v], Sin[u] * Sin[v], Cos[v]},
  {u, 0, 2 * Pi}, {v, 0, Pi}, PlotPoints → {60, 12}]
```

Out[42]=



In[43]:=
```
(*ALTERNATIVES TO 3D*)
```

In[44]:=
```
? DensityPlot
```

DensityPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] makes a density plot of f as a function of x and y.

DensityPlot[f, {x, y} ∈ reg] takes the variables {x, y} to be in the geometric region reg. ≫

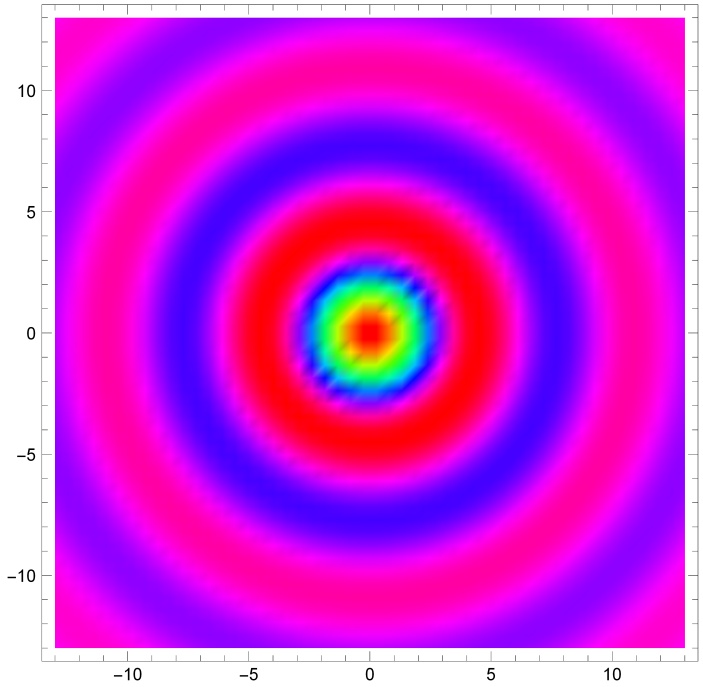
In[45]:=
```
? r
```

```
Global`r
```

$r = \sqrt{x^2 + y^2}$

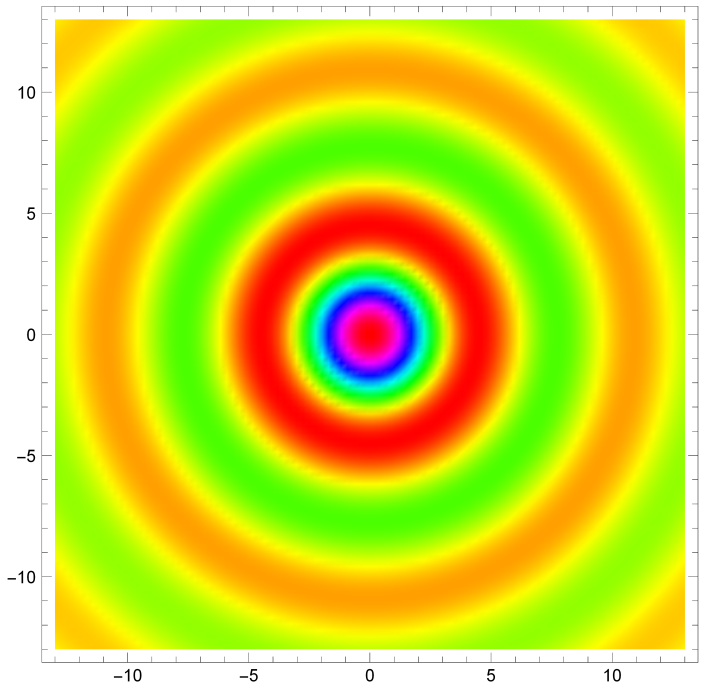
In[46]:=
```
DensityPlot[-Sin[r] / r, {x, -13, 13}, {y, -13, 13},
 ColorFunction → Hue, PlotPoints → 50, PlotRange → All]
```

Out[46]=



In[47]:=
```
DensityPlot[Sin[r] / r, {x, -13, 13}, {y, -13, 13},
 ColorFunction → Hue, PlotPoints → 100, PlotRange → All]
```

Out[47]=



In[48]:=
```
? ContourPlot
```

ContourPlot[f, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] generates a contour plot of f as a function of x and y.

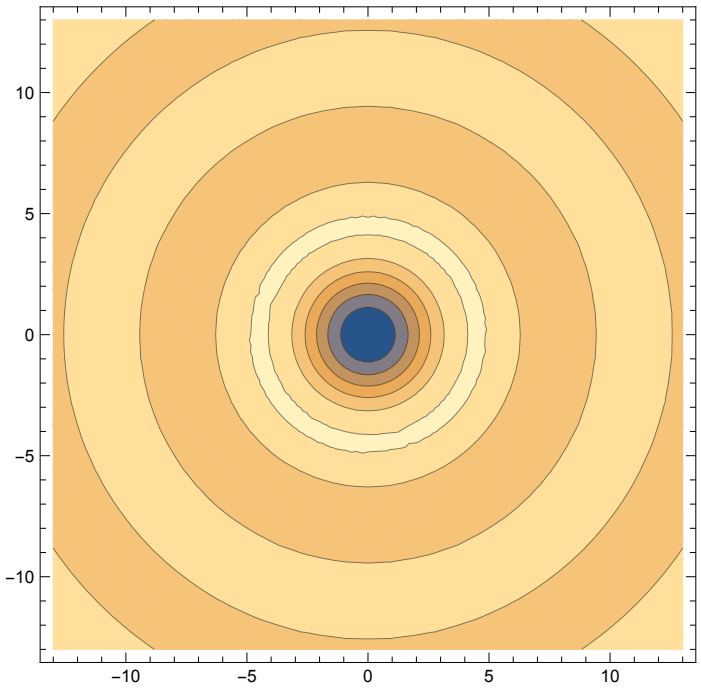
ContourPlot[f == g, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots contour lines for which f = g.

ContourPlot[{f_1 == g_1, f_2 == g_2, …}, {x, x_{min}, x_{max}}, {y, y_{min}, y_{max}}] plots several contour lines.

ContourPlot[…, {x, y} ∈ reg] takes the variables {x, y} to be in the geometric region reg. ≫

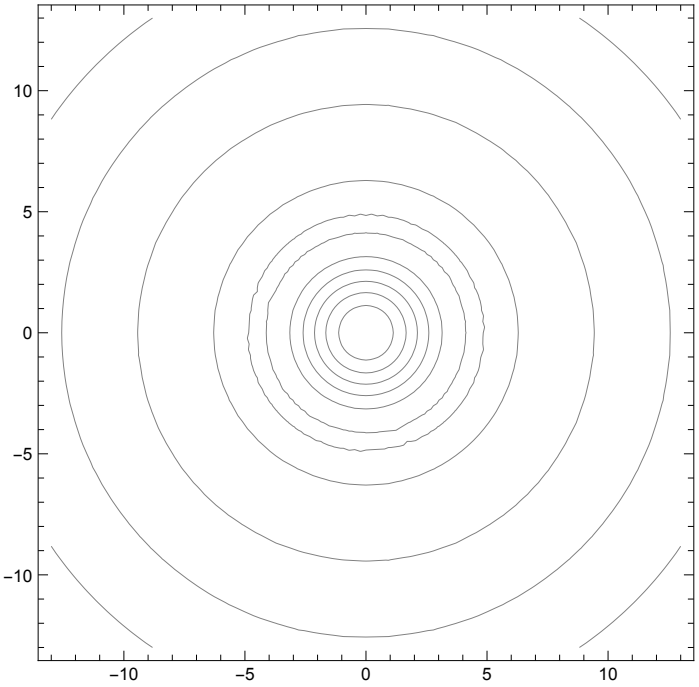
In[49]:=
```
ContourPlot[-Sin[r] / r, {x, -13, 13}, {y, -13, 13}, PlotRange → All]
```

Out[49]=



In[50]:=
```
ContourPlot[-Sin[r] / r, {x, -13, 13},
 {y, -13, 13}, PlotRange → All, ContourShading → False]
```

Out[50]=

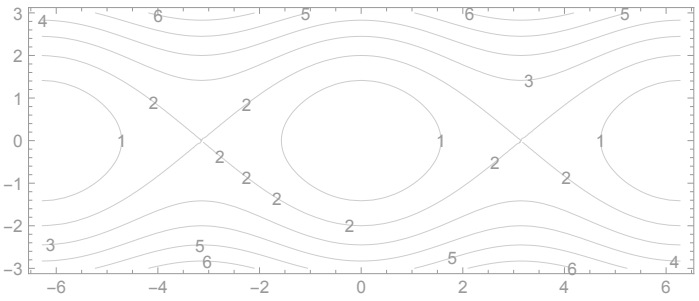



```
(*pendulum; x angle, p -momentum*)
Clear[u, e, p];
u = 1 - Cos[x](*pot. energy*);
e = u + p^2 / 2 (*full energy*)
```

Out[53]=
$$1 + \frac{p^2}{2} - \text{Cos}[x]$$

```
ContourPlot[e, {x, -2 Pi, 2 Pi}, {p, -3, 3},
 ContourShading → False, ContourLabels → All, AspectRatio → .4
]
```

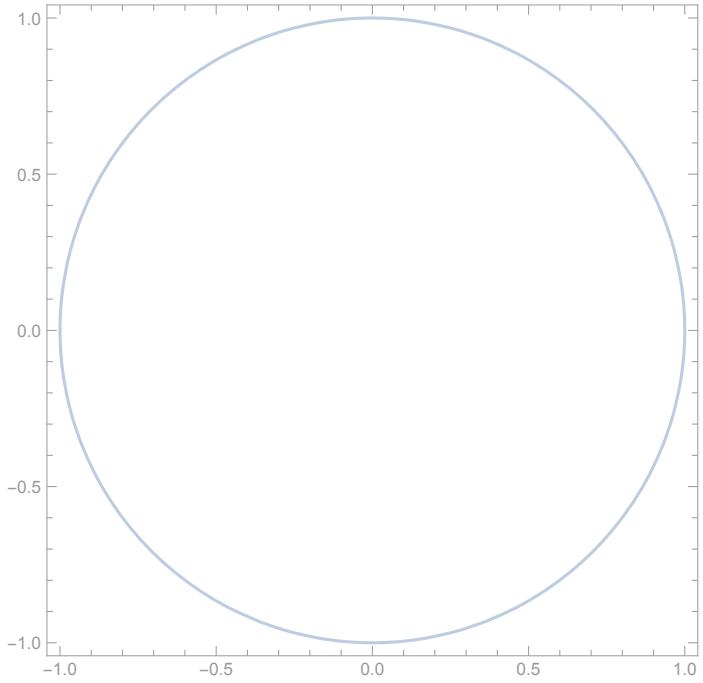
Out[54]=



In[55]:=

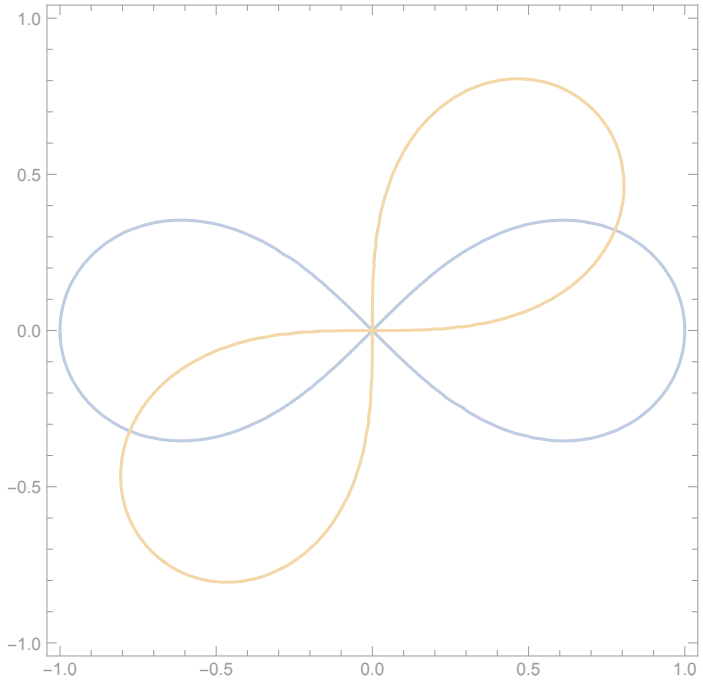
```
(*IMPLICIT PLOT-from ContourPlot;*)
ContourPlot[x^2 + y^2 == 1, {x, -1, 1}, {y, -1, 1}] (*circle*)
```

Out[56]=




```
ContourPlot[{(x^2 + y^2) ^2 == x^2 - y^2, (x^2 + y^2) ^2 == 2 x y}, {x, -1, 1}, {y, -1, 1}]
(*2plots with different styles-see output below.ContourPlot3D-similar*)
```

Out[57]=



In[58]:=

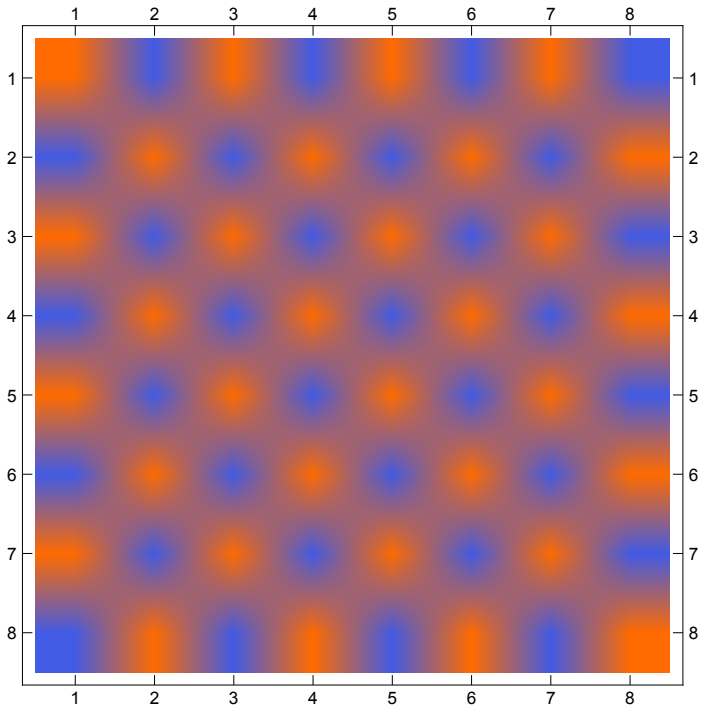
```
(*DISCRETE PLOTS*)
```

In[59]:=
```
chesslist = Table[If[EvenQ[x + y], -1, 1], {x, 8}, {y, 8}]
```

Out[59]=
```
{{-1, 1, -1, 1, -1, 1, -1, 1}, {1, -1, 1, -1, 1, -1, 1, -1},
 {-1, 1, -1, 1, -1, 1, -1, 1}, {1, -1, 1, -1, 1, -1, 1, -1}, {-1, 1, -1, 1, -1, 1, -1, 1},
 {1, -1, 1, -1, 1, -1, 1, -1}, {-1, 1, -1, 1, -1, 1, -1, 1}, {1, -1, 1, -1, 1, -1, 1, -1}}
```

In[60]:=
```
MatrixPlot[-chesslist]
```

Out[60]=



In[61]:=
```
(*IMAGE as ARRAY of NUMBERS*)
```

In[62]:=
```
lena = Import["ExampleData/lena.tif"]
```

Out[62]=

In[63]:=
```
Binarize[%]
```

Out[63]=

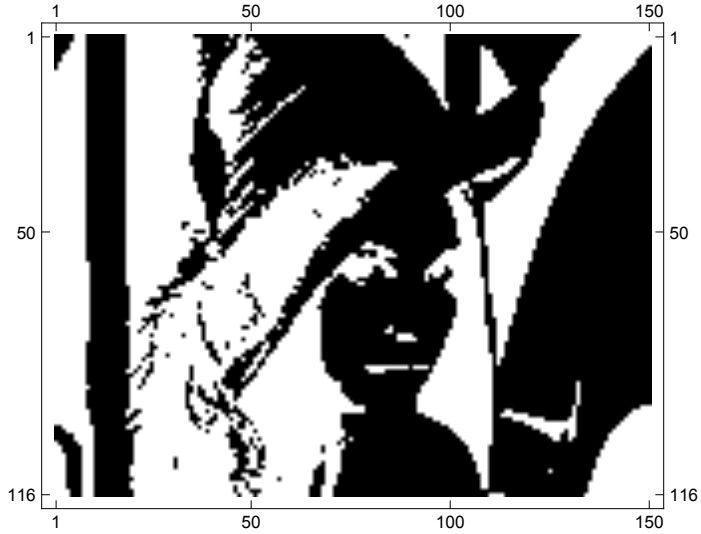
In[64]:=
```
im = ImageData[%];
```

In[65]:=
```
im[[50]] // Short
```

Out[65]//Short=
```
{0, 0, 0, 0, 0, 0, <<139>>, 1, 1, 1, 1, 1}
```

In[66]:=
```
MatrixPlot[im, ColorFunction → "Monochrome"]
```

Out[66]=

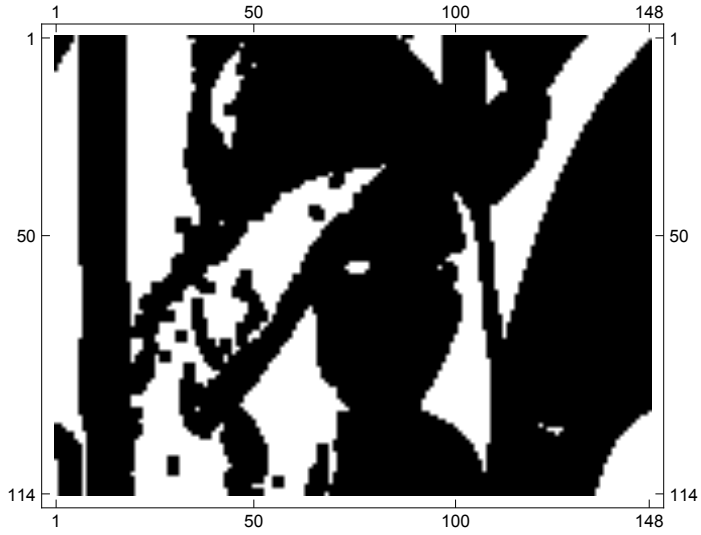


In[67]:=
```
A = Table[1, {3}, {3}]
```

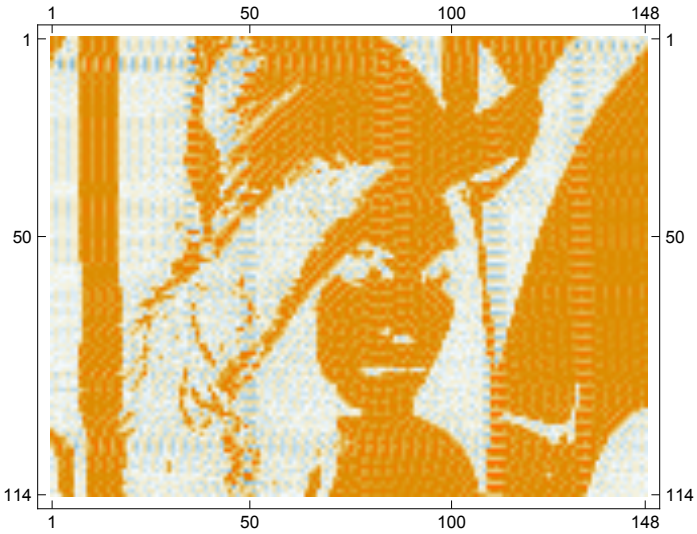
Out[67]=
```
{{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}
```

In[68]:=
```
imc = ListConvolve[A, im]; MatrixPlot[imc, ColorFunction → "Monochrome"]
```

Out[68]=



```
imb = ListDeconvolve[A, imc]; MatrixPlot[imb]
```



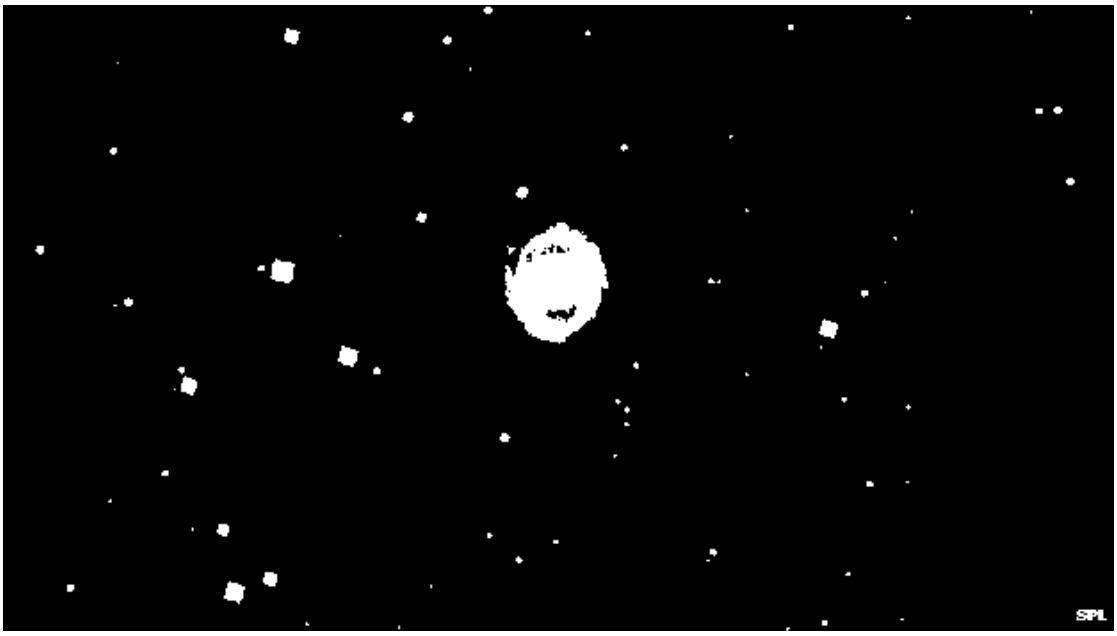
In[69]:=
```
(*Real Image from BBC 4/25/2014*)
```

In[70]:=
```
nova = Import["http://web.njit.edu/~vitaly/114/bbc-supernova.jpg"]
```

Out[70]=

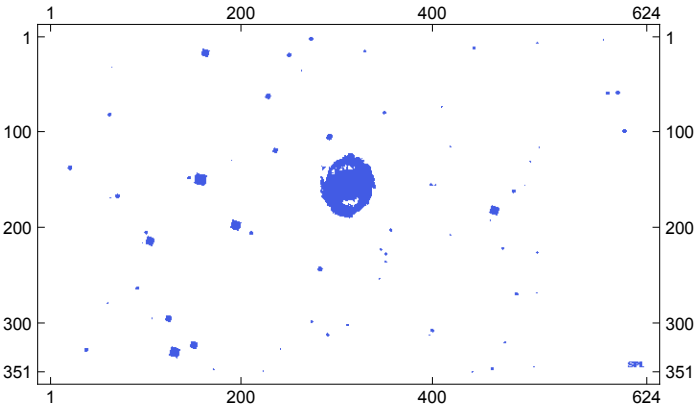
In[71]:=
```
bw = Binarize[nova]
```

Out[71]=



In[72]:=
```
im = ImageData[bw]; MatrixPlot[-im]
```

Out[72]=



In[73]:=