バージョン6での変更点

2008/8/29

はじめに

「工学系のための *Mathematica* 入門第 2 版」は *Mathematica* バージョン 5 に合わせて書かれました。。*Mathematica* は 2007 年からバージョン 6 になりました。バージョン 6 ではグラフィックやアニメーション関係が強化されました。またこれまでパッケージで提供されてきた関数がカーネルに組み込まれ、パッケージを読まなくても使えるようになりました。

ここではバージョン 6 になって「工学系のための *Mathematica* 入門第 2 版」で変った主な点を紹介することにします。

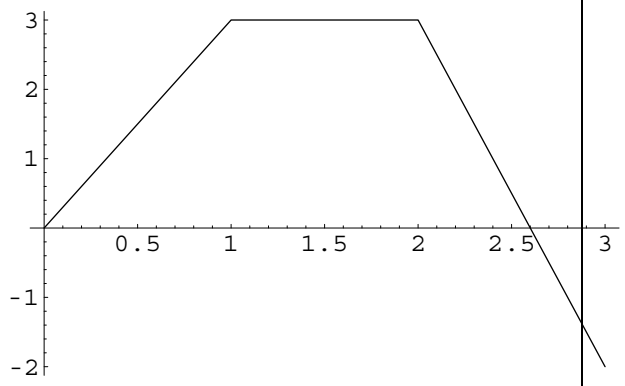
第1章

Mathematica を使おう

1.3 グラフィックス

ver.5 では `PlotJoined` で線を結んでいましたが、ver. 6 では `Joined` に変更されています。

```
In[8]:= ListPlot[{{0, 0}, {1, 3}, {2, 3}, {3, -2}},
          PlotJoined -> True]
```

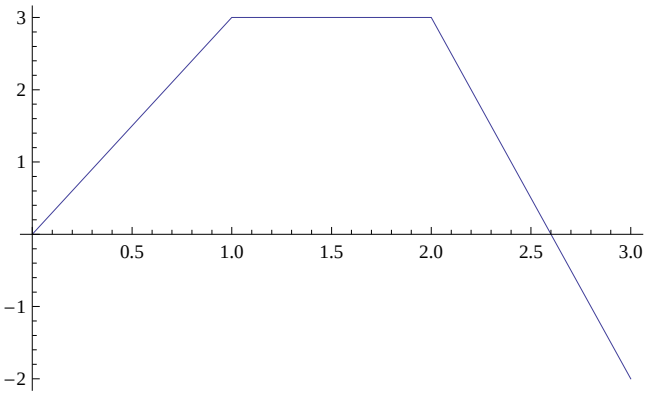



```
Out[8]= - Graphics -
```

二次元のデータのグラフ表示 ver.5

```
In[8]:= ListPlot[{{0, 0}, {1, 3}, {2, 3}, {3, -2}}, Joined → True]
```

Out[8]=



二次元のデータのグラフ表示 ver.6

第2章

ノートブックの使い方

2.2 ヘルプブラウザ

ver.6 ではヘルプブラウザは大幅に変更されました。

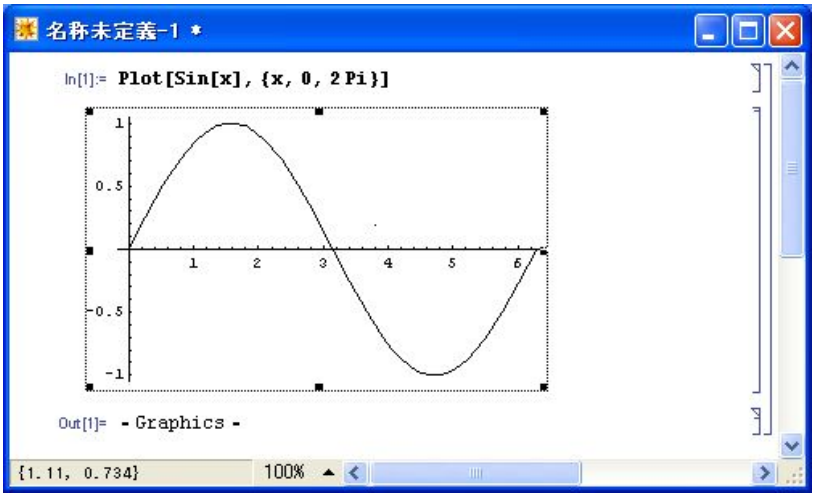
ヘルプブラウザは大きく分けて 3 つのカテゴリに分けて情報を得ることができるようになっています。「ドキュメントセンター」、「関数ナビゲータ」、「バーチャルブック」です。入口は違うのですが、内容はどれも同じになっております。それぞれを使いやすい方法で使うといいです。

さらに「デモンストレーション」は「デモンストレーションプロジェクト」というインターネット上のウェブページにまとめられています。そこではアニメーションや `Manipulate[ ]` を利用したデモが数多く紹介されているので、ぜひご覧下さい。

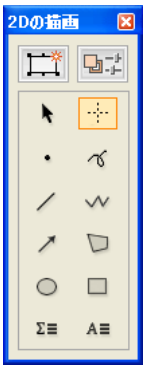
2.5.1 グラフの座標の読みとり方, Ctrl-d

ver.6 では次のようにしてグラフの座標を読み取ります。

まずマウスをこのグラフの上に載せてクリックします。するとオレンジ色の線でグラフが囲まれます。このことによりこのグラフが選択されました。さらにコントロールキー (Ctrl) を押したまま d を押す Ctrl + d により「2D の描画」というパレットが表示されます。このパレットの中の点線の十字を選択して、マウスをグラフ上で移動させると、マウスカーソルが十字になります。この時マウス付近に座標が $\{x, y\}$ のように表示されるのがわかると思います。この機能を使ってグラフの座標をおおざっぱに読むことができます。



グラフから座標を読みとる ver.6



2D の描画 ver.6

第3章

計算処理

3.3.4 機械精度

次の例では ver.5 と ver.6 で違う結果が得られています。どうも計算の順番を考慮しなくなり、機械精度のみを考慮しているようです。つまり ver.6 では機械精度の範囲内では別に間違えた答えになっているわけではないです。ver.5 では機械精度を越えた計算が一部できていたと考えることもできます。

```
In[13]:= $MachinePrecision

Out[13]= 15.9546

In[14]:= 3.14 + 1.23 10^-30 - 3.14

Out[14]= 0.

In[15]:= 3.14 - 3.14 + 1.23 10^-30

Out[15]= 1.23 × 10^-30

In[16]:= Precision[3.14]

Out[16]= MachinePrecision
```

機械精度による誤差で順番により結果が違う ver.5

```
In[13]:= $MachinePrecision

Out[13]= 15.9546

In[14]:= 3.14 + 1.23×10^-30 - 3.14

Out[14]= 0.

In[15]:= 3.14 - 3.14 + 1.23×10^-30

Out[15]= 0.

In[16]:= Precision[3.14]

Out[16]= MachinePrecision
```

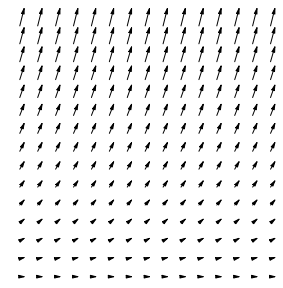
機械精度による誤差を考慮しないといけない ver.6

3.6.5 微分方程式, DSolve

ベクトル図を表示させるパッケージ名が Graphics‘PlotFiled‘から VectorFieldPlots‘に変更されています。また関数名も PlotVectorField[ ] から VectorFieldPlot[ ] になっています。

```
In[26]:= Needs["Graphics`PlotField` "]

In[27]:= vec = PlotVectorField [
           {1, 1 / 2 x}, {t, -4, 4}, {x, 0, 8}]
```



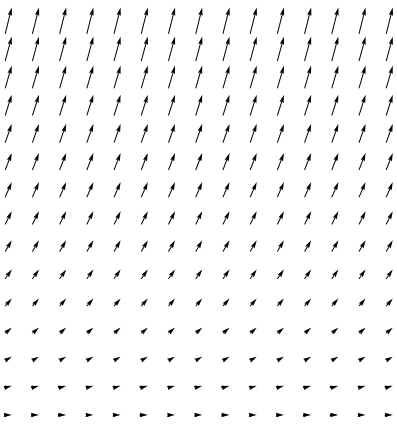
```
Out[27]= - Graphics -
```

ベクトル図のプロット ver.5

```
In[28]:= Needs["VectorFieldPlots`"]

In[29]:= vec = VectorFieldPlot[{1, 1 / 2 x}, {t, -4, 4}, {x, 0, 8}]

Out[29]=
```



ベクトル図のプロット ver.6

3.8.2 少し高度な近似, FindFit

ver.6 では FindFit[] で統一されています。

第4章

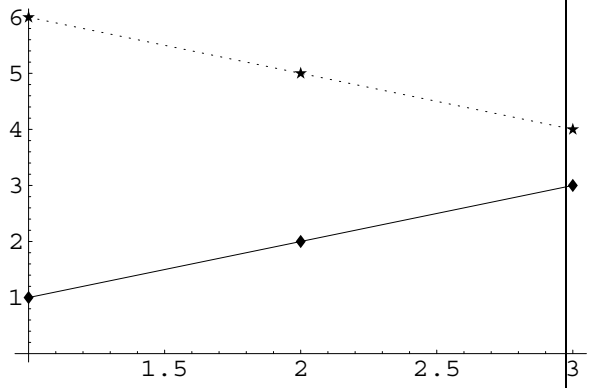
グラフィックスとアニメーション

4.1.6 データの表示, ListPlot

MultipleListPlot[ ] は ListLinePlot[ ] に統合されました。

```
In[16]:= Needs["Graphics`MultipleListPlot` "]

In[17]:= MultipleListPlot[{1, 2, 3}, {6, 5, 4},
          PlotJoined -> True]
```

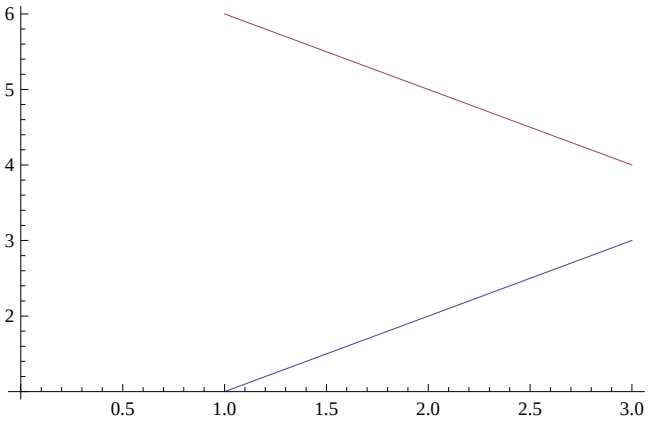



```
Out[17]= - Graphics -
```

MultipleListPlot を使ったグラフ ver.5

```
In[14]:= ListLinePlot[{{1, 2, 3}, {6, 5, 4}},
          Joined -> True]
```

Out[14]=



ListLinePlot を使ったグラフ ver.6

4.1.7 対数グラフ, LogPlot

対数グラフは ver.5 ではパッケージを読み込んでいましたが、ver.6 ではカーネルに組み込まれたので読み込む必要が無いです。片対数も充実しました。次のようになります。

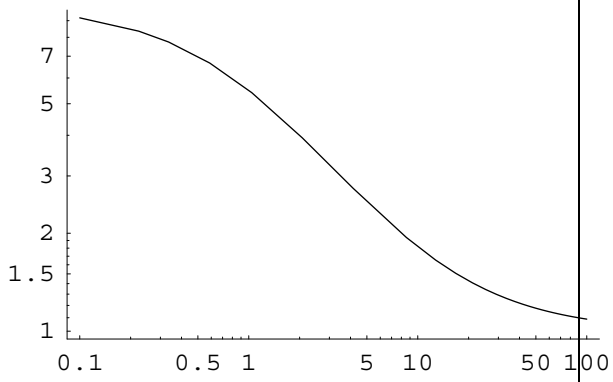
- `LogPlot[ ]` が y 軸を対数とした片対数グラフ
- `LogLinearPlot[ ]` が x 軸を対数とした片対数グラフ
- `LogLogPlot[ ]` が両対数グラフ

またリストデータを引数にしてグラフにする関数も作られております。

- `ListLogPlot[ ]`
- `ListLogLinearPlot[ ]`
- `ListLogLogPlot[ ]`

```
In[18]:= Needs["Graphics`Graphics` "]

In[19]:= LogLogPlot[(x + 10) / ((x + 1)), {x, 0.1, 100}]
```

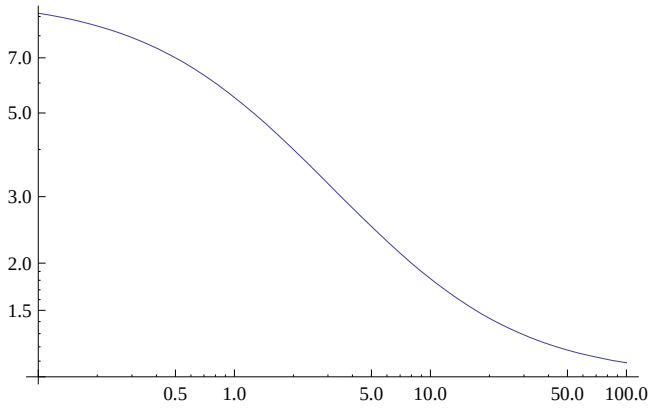



```
Out[19]= - Graphics -
```

パッケージにある LogLogPlot ver.5

ver.6 では Needs[ ] によるパッケージの読み込みは必要なくそのまま実行できる。

```
In[15]:= LogLogPlot[(x + 10) / ((x + 1)), {x, 0.1, 100}]
```




```
Out[15]=
```

LogLogPlot ver.6

4.1.10 3 次元のグラフ表示, Plot3D

3D 表示の視点を変えるには ver.5 では次の「ビューポイントの設定」パネルを利用しておりました。

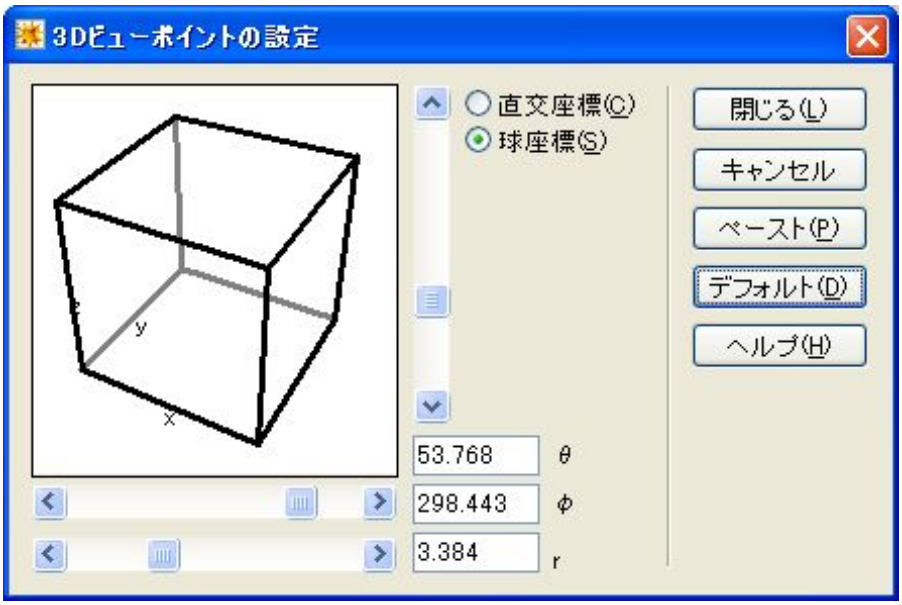


図 4.1 三次元の視点を変えるためのツール。これで ViewPoint を指定する。ver.5

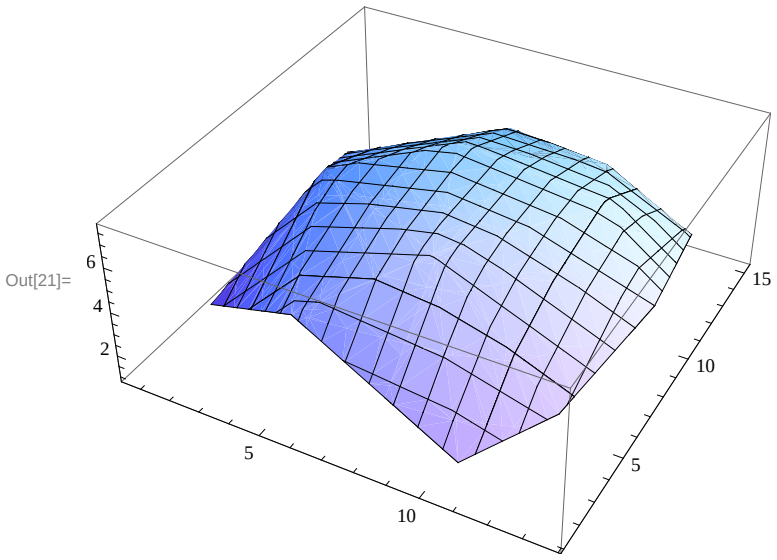
ver.6 からは直接作画された 3D 表示の図をマウスでクリックして回転させることができるようになりました。

4.1.11 数値データの 3 次元表示, TriangularSurfacePlot

ver.6 から ListPlot3D[ ] が拡張されて、ver.5 の TriangularSurfacePlot[ ] と同じデータについてプロットすることができるようになりました。

In[21]:= **ListPlot3D[data3D]**

Out[21]=

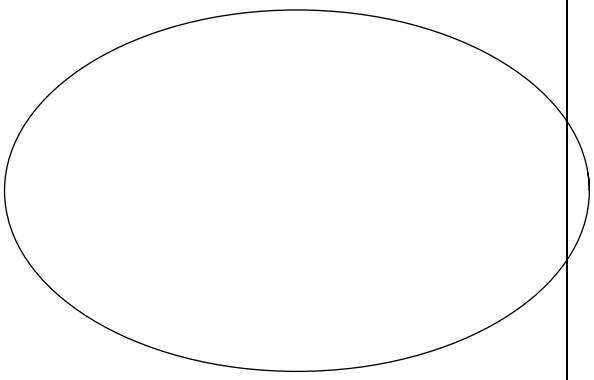


三次元データのグラフ化の例 ver.6

4.2.1 グラフィックス要素, Line, Circle,..

ver.5 では Show[Graphics[...]] として始めて表示されましたが、ver.6 では Show[ ] は必要でなく Graphics[...] で表示されます。また次の例を見れば分かりますが、AspectRatio -> 1 のように 1 にしなくても、自動的に必要なときには AspectRatio が 1 になるようです。

```
In[3]:= Show[Graphics[Circle[{0, 0}, 1]]]
```

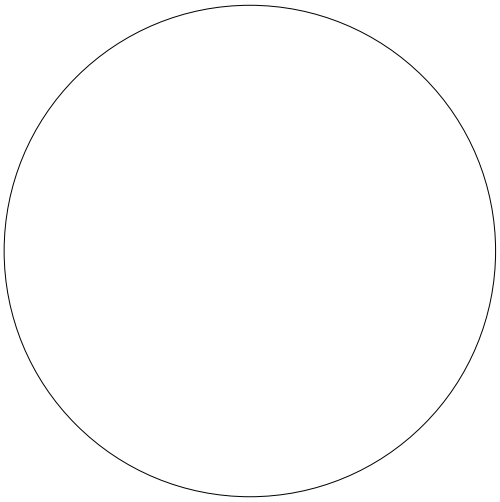


```
Out[3]= -Graphics-
```

グラフィックスオブジェクトの表示 ver.5

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

Out[2]=



グラフィックスオブジェクト ver.6

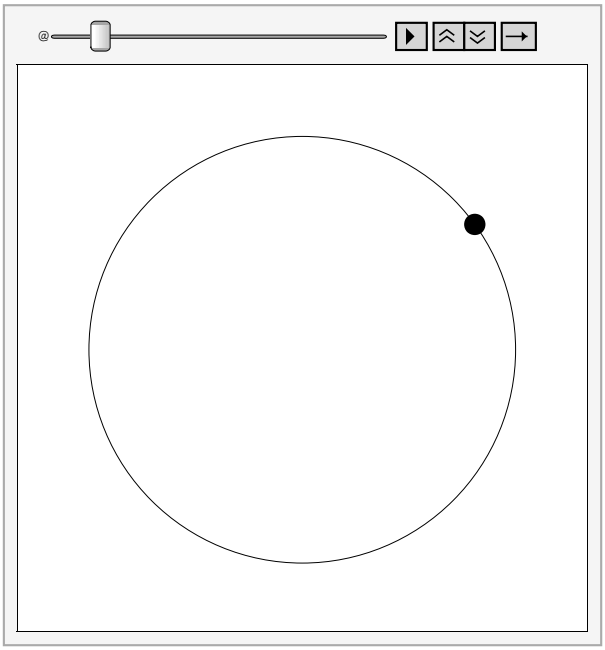
4.2.4 円運動

ver.6 では Animate[ ] が強化されました。今までのように Do[] でループさせて Show[Graphics[...]] で表示するのではなく、Animate[ ] がループを持ち、Graphics[...] で表示します。ver.6 による円運動のプログラムを示します。

In[1]:=
```
Animate[
 locX[t_] := Cos[t];
 locY[t_] := Sin[t];
 circ = Circle[{0, 0}, 1];
 point =
  Disk[{locX[t], locY[t]}, 0.05];
 options =
  {AspectRatio -> 1, PlotRange ->
    {{-1.2, 1.2}, {-1.2, 1.2}}};

 Graphics[{circ, point}, options]
     , {t, 0, 2 Pi, Pi / 20}
]
```

Out[1]=

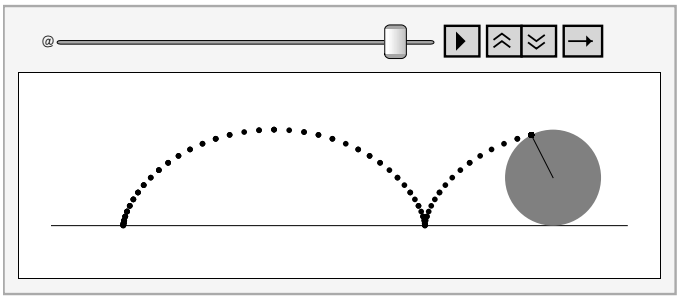


円運動のアニメーション ver.6

4.2.5 サイクロイド

ver.6 でのサイクロイドのプログラムを示します。最初の If[ ] では $t = 0$ の時に、リスト変数 dots の初期化をしています。

```
Animate[
 If[t == 0,
  dots = {PointSize[0.006]}];
 cycloid[t_] :=
  {t - Sin[t], 1 - Cos[t]};
 baseline = Line[
   {{-2, 0}, {10.5, 0}}];
 disk = {GrayLevel[.5],
   Disk[{t, 1}, 1] };  (* disk *)
 spoke = Line[{{t, 1}, cycloid[t]}];
 options =  {AspectRatio -> 3 / 12,
       PlotRange ->
    {{-1.5, 10.5}, {-.5, 2.5}} };
 AppendTo[dots, Point[cycloid[t]]];
 Graphics[{baseline,
   disk, spoke, dots}, options
   ],
 {t, 0, 3 Pi, Pi / 20}
]
```



サイクロイドのアニメーション ver.6

ver.6 では Manipulate[ ] という関数が新設され、いくつかのパラメータをスライダを使って変化させたときに、図などの結果に反映させる方法が提供されています。これらのアニメーションの例で Animate[ ] を Manipulate[ ] にして試してみてください。

アニメーション GIF の製作

折角できたアニメーションはこのままでは *Mathematica* の上でしか利用することができません。アニメーションを**パワーポイント**や**ウェブページ**上で利用することを考えてみます。

考え方としては、すこしづつ違う図を描かせて、それが格納されているリストを Export[ ] を利用して**アニメーション GIF** ファイルとして書き出すという手順をとります。

Do[ ] 文を利用して円運動のアニメーションの一つ一つの図を描き、それをリストに格納してみましょう。プログラムは次のようになります。

```
In[7]:= locX[t_] := Cos[t];
        locY[t_] := Sin[t];

        circ = Circle[{0, 0}, 1];
        point = Disk[{locX[t], locY[t]}, 0.05];
        options =     {AspectRatio -> 1,
           PlotRange -> {{-1.2, 1.2}, {-1.2, 1.2}}};
        gifs = {}; (* gifsを初期化  *)
        Do[
        gif =     Graphics[{circ, point}, options];
         AppendTo[gifs, gif]
             , {t, 0, 2 Pi, Pi / 20}
        ]
```

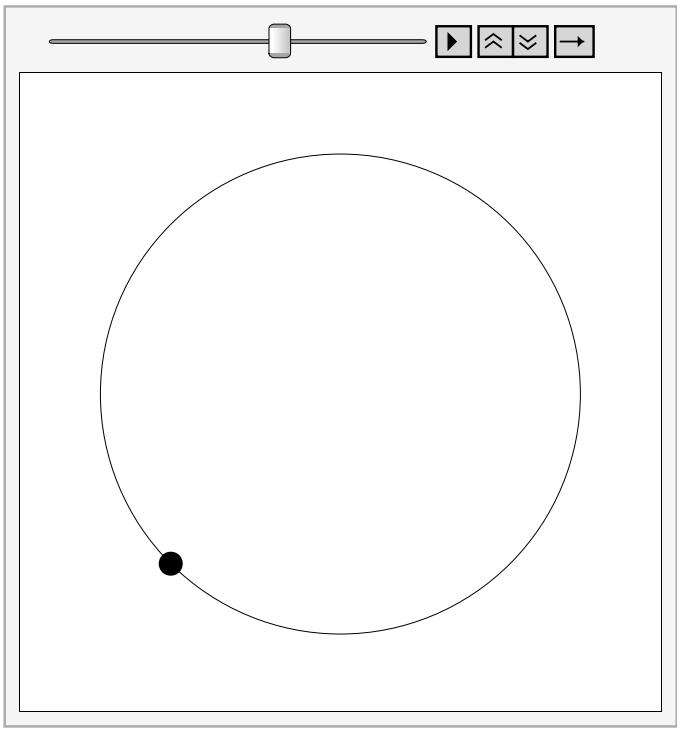
円運動のアニメーション: gifs の準備 ver.6

おおよその構造はさきほどの例と一緒ですが、最初にいろいろと準備して、Do[ ] 文でループさせています。途中に gifs={}; というのがありますが、これは gifs という変数をリストとして初期化しています。そして Do[ ] 文の中で、まず gif という変数にグラフィックオブジェクトを記憶します。これを AppendTo[ ] を用いて gifs リストに追加していきます。このようにして、グラフィックオブジェクト (図) を多数格納したリストができます。

できたリストはアニメーションとして再生することができます。再生には ListAnimate[ ] を利用します。

In[14]:= **ListAnimate[gifs]**

Out[14]=



円運動のアニメーション: 再生 ver.6

最後に、この gifs を Export[ ] を用いてアニメーション GIF のファイル形式でファイルに出力します。ファイルの拡張子を.gif にすることにより *Mathematica* は自動的にファイル形式を変更します。

```
In[15]:= Directory[]
Out[15]= D:\Documents and Settings\otabe\My Documents
In[16]:= Export["test.gif", gifs]
Out[16]= test.gif
```

円運動のアニメーション: ファイル保管 ver.6

出力されたファイルは GIF89 形式になっているので、パワーポイントやウェブページでアニメーションとして再生することができます。

参考までに、サイクロイドのアニメーションについても図のリスト (gifs) を作るプログラムを載せておきます。

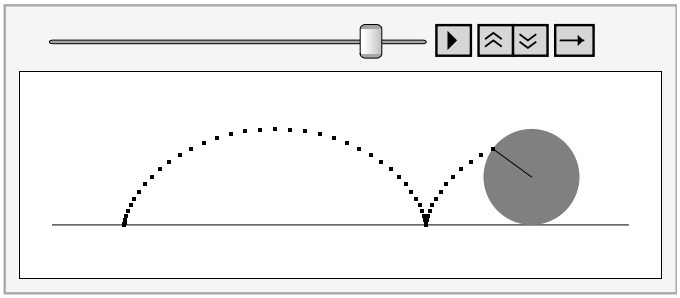
In[15]:=
```
cycloid[t_] := {t - Sin[t], 1 - Cos[t]};
baseline = Line[{{-2, 0}, {10.5, 0}}];
disk = {GrayLevel[.5], Disk[{t, 1}, 1] };
(* disk *)
spoke = Line[{{t, 1}, cycloid[t]}];
dots = {PointSize[0.006]};
options = {AspectRatio -> 3 / 12,

   PlotRange -> {{-1.5, 10.5}, {-.5, 2.5}} };
gifs = {};

Do[
 AppendTo[dots, Point[cycloid[t]]];
 AppendTo[gifs,
   Graphics[
   {baseline, disk, spoke, dots}, options
   ]
    ], {t, 0, 3 Pi, Pi / 20}
    ]

ListAnimate[gifs]
```

Out[23]=



サイクロイドのアニメーション ver.6

第5章

物理現象の可視化

5.2.2 点電荷における電界と等電位面

ver.5 では下記の 4 つのパッケージを読む必要がありましたが、ver.6 では 2 つになります。

```
In[1]:= << Graphics`PlotField`
        << Graphics`PlotField3D`
        << Graphics`ContourPlot3D`
        << Calculus`VectorAnalysis`
```

必要なパッケージの読み込み ver.5

```
In[1]:= << VectorFieldPlots`;
        << VectorAnalysis`;
```

必要なパッケージの読み込み ver.6

また関数名も下記のように異なります。考え方としては `Plot[]` や `Plot3D[]` という基本的な名前があって、それに `VectorField` や `Log` が前に付いたということです。

ver.5	ver.6
PlotVectorField3D	VectorFieldPlot3D
PlotVectorField	VectorFieldPlot