```
In[2]:= Needs["Graphics`Graphics` "]

        LogPlot::shdw :
         複数のコンテキスト {Graphics`Graphics`, Global`}においてシンボル
           LogPlotが使われています. コンテキスト Graphics`Graphics`
           の定義は他の定義で隠れてしまう可能性があります.    詳細

In[3]:= LogPlot[x^2 + 3 x, {x, 1, 10}]

Out[3]= LogPlot[3 x + x^2, {x, 1, 10}]
```

一度定義してしまうとエラーが出る

ここで**?**を使って確認すると定義されてしまっていることがわかります。

```
In[4]:= ?LogPlot

        Global`LogPlot
```

定義されていることがわかる

このようなときには **Remove[ ]** を使って Global から消して、パッケージを読み込んだ定義がある領域から定義をちゃんと参照できるようにします。そうすると期待通りに動くようになります。

```
In[18]:= FindRoot[Exp[x] - x^4 == 0, {x, 1000}]

         FindRoot::cvmit :
          100回以内の反復では要求された精度または確度に収束するこ ⋱
              とができませんでした.  >>

Out[18]= {x → 900.}
```

初期値の悪い例

FintRoot では連立した場合にも初期値を与えて数値的に求めることができます。この例では複素数が解になっています。

```
In[19]:= FindRoot[{x == Log[y], y == Log[x]}, {x, I}, {y, I}]

Out[19]= {x → 0.318132 + 1.33724 i, y → 0.318132 + 1.33724 i}
```

連立した方程式の数値解

3.6 微積分

この節では微積分について *Mathematica* で解いてみることにしましょう。方程式の解と同様に解析的に解けるものと数値的にしか求められないものがあります。

3.6.1 微分, D

微分は組み込み関数 **D[ ]** により計算します。詳しく書くと **D[ ]** は**偏微分**であり、**全微分**を使うときには組み込み関数 **Dt[ ]** を使います。

```
In[1]:= D[x^n, x]

Out[1]= n x^(-1+n)

In[2]:= D[Sin[x] Cos[y], x]

Out[2]= Cos[x] Cos[y]
```

微分を求める

多重積分の際には、積分の範囲を続けて書いていきます。積分は外側の変数、この場合には y から行われます。

```
In[11]:= Integrate[x^2 + y^2, {x, 0, 1}, {y, 0, x}]

Out[11]= 1/3
```

多重定積分を求める

さて次のような**特異点**を持っている式の積分について考えてみます。

$$x^2 \sin \frac{1}{x}$$

上式は $x = 0$ で極限をとると 0 ですが、そのまま数字を代入しただけだと**不定**です。このような関数をそのまま数値積分すると次のようなエラーを出して計算してくれません。

```
In[12]:= NIntegrate[x^2 Sin[1 / x], {x, -1, 2}]

      NIntegrate::slwcon :
       数値積分の収束が遅すぎます．特異性がある，積分の値がゼロ
           ，被積分関数が激しく振動する，あるいは WorkingPreci
           sionが小さすぎるのいずれかが原因である可能性があり
           ます．  >>

Out[12]= 1.38755
```

特異点を含む定積分

このようなときには特異点をさけて積分するように積分範囲を決めることができます。このような方法で数値積分ができる可能性があります。

```
In[13]:= NIntegrate[x^2 Sin[1 / x], {x, -1, 0, 2}]

Out[13]= 1.38755
```

特異点をさけて定積分をする

3.6.3 極限, Limit

次の式は x = 0 で極限をとると 1 ですが、そのまま数字を代入しただけでは**発散**します。

$$\frac{\sin x}{x}$$

の x に 0 を代入してみます。次にようにルール (**->**) と置換 (**/.**) を使って一時的に代入を試みます。

```
In[14]:= Sin[x] / x /. x -> 0

         Power::infy : 無限式 1/0 が見付かりました.    >>

         ∞::indet :
          不定式0 ComplexInfinityが見付かりました.    >>

Out[14]= Indeterminate
```

ルールを用いた代入

Indeterminate はヘルプブラウザで見れば分かりますが、「その大きさが定まらない数値的な量を示す」とあります。これは**不定形**ということで、この場合には 0/0 の形になっています。途中のメッセージにに **ComplexInfinity** がありますが、これは「偏角が不定で絶対値が無限な複素量を表す」とあり、この場合には無限大のことで不定形とは異なります。

しかしこの式は $x \to 0$ の**極限**でちゃんと値を持っています。0.01 のように小さい値を与えると、この式は 1 に近い値を返してきます。

```
In[15]:= Sin[x] / x /. x -> 0.01
Out[15]= 0.999983
```

$x = 0$ での値があるように思える

Mathematica では極限を組み込み関数 **Limit[ ]** を使って計算することができます。

```
In[5]:= eq[0]

        InterpolatingFunction::dmval : 入力値{0}
           は補間関数のデータ範囲外です. 外挿が使用されます.
           >>

Out[5]= 76.4375
```

補完データの外挿

さらに微分積分すら可能です。これらは数値的に調べることのみできます。

```
In[6]:= Integrate[eq[x], {x, 0.2, 6}]

Out[6]= 40.3135
```

補完関数の積分

この **InterpolatingFuntion[ ]** 自身を外に連れ出すことはできませんので、数値的にのみ外に取り出すことができます。

Interpolation をドキュメントセンターで調べるとさらに関連するパッケージとして

ListInterpolation,
FunctionInterpolation,
InterpolatingPolynomial,
Fit, ...

という情報を見ることができます。多項式で近似するための便利な関数がパッケージに準備されています。是非読まれてみてください。

3.9　簡単なプログラミング

ここの節では簡単な関数の定義のしかたのみを説明しましょう。**プログラミング**については本書では詳しくは述べませんが、ここでも簡単に触れておきます。なぜならば、自分の関数を作ることで格段に *Mathematica* が使いやすくなるからです。これまでのところを読み進められた方は *Mathematica* ですべての

```
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

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。