In order to solve this differential equation you look at it till a solution occurs to you.

George Pólya (1887 - 1985)

10

Differential Equations and Recurrences

10.1 Differential Equations

10.1.1 Introduction

If George Pólya's method does not seem very effective, one can appeal to Sage even if the domain of formal resolution of differential equations remains a weakness of many symbolic computation systems. Sage is evolving however by expanding its spectrum of resolution.

One can, if desired, invoke Sage in order to obtain a qualitative study: indeed, its numerical and graphical tools will guide the intuition. This is the subject of Section 14.2 from the chapter on numerical integration. Tools for the graphical study of the solutions are given in Section 4.1.6. Solving methods using series can be found in Section 7.5.2.

One may prefer to solve differential equations exactly. Sage can sometimes help by directly giving a formal answer as we will see in this chapter.

In most cases, it will be necessary to go through a tricky manipulation of these equations to help Sage. It should be kept in mind that the expected solution of a differential equation is a *function* differentiable over a certain interval, but that Sage manipulates *expressions* without a definition domain. The machine will therefore require human intervention to move towards a rigorous solution.

We shall first study generalities on ordinary differential equations of order 1 and some special cases such as linear equations, separable equations, homogeneous equations, a parameter dependent equation (§10.1.2); then more briefly the equations of order 2 and an example of a partial differential equation (§10.1.3). We will end with the use of the Laplace transform (§10.1.4) and finally the resolution of some differential systems (§10.1.5).

An *ordinary differential equation* (ODE) is an equation involving an (unknown) function of a single variable, as well as one or more derivatives, successive or not, of the function.

In the equation $y'(x) + x \cdot y(x) = \sin(x)$ the unknown function y is called the *dependent variable* and the variable x (relative to which y varies) is called the *independent variable*.

A *partial differential equation* (referred to as PDE) involves several independent variables as well as the partial derivatives of the dependent variable with respect to these independent variables.

Unless otherwise stated, we shall consider in this chapter functions of a real variable.

10.1.2 First-Order Ordinary Differential Equations

Basic Commands. We would like to solve a first-order ODE:

$$F(x, y(x), y'(x)) = 0.$$

We start by defining a variable x and a function y depending on this variable:

```
sage: x = var('x')
sage: y = function('y')(x)
```

Then:

```
sage: desolve(equation, variable, ics = ..., ivar = ...,
....:         show_method = ..., contrib_ode = ...)
```

where:

- `equation` is the differential equation. Equality is designated by `==`, for instance, the equation $y' = 2y + x$ is written `diff(y,x) == 2*y+x`;
- `variable` is the dependent variable, i.e., y in $y' = 2y + x$;
- `ics` is optional and stands for initial conditions. For a first-order equation, write $[x_0,y_0]$ and for a second-order equation write $[x_0,y_0,x_1,y_1]$ or $[x_0,y_0,y_0']$;
- `ivar` is optional and stands for the independent variable, i.e., x in $y' = 2y+x$. It must be specified if there is more than one independent variable or parameters as in $y' = ay + bx + c$;
- `show_method` is an optional boolean set to false. If true, then Sage returns a pair "[solution, method]", where method is the string describing the method which has been used to get a solution. The method can be one of the following: `linear`, `separable`, `exact`, `homogeneous`, `bernoulli`, `generalized homogeneous`.
- `contrib_ode` is an optional boolean set to false. If true, `desolve` allows to solve Clairaut, Lagrange, Riccati and some other equations. This may take a long time and is thus turned off by default.

First-Order Equations Directly Solved by Sage. We will study in this section how to solve with Sage linear, separable, Bernoulli, Riccati, Lagrange, Clairaut, homogeneous and exact equations.

Linear equations. These are equations of the form

$$y' + P(x)y = Q(x),$$

where P and Q are continuous functions on given intervals.

Example: $y' + 3y = \mathrm{e}^x$.

```
sage: x = var('x'); y = function('y')(x)
```

```
sage: desolve(diff(y,x) + 3*y == exp(x), y, show_method=True)
[1/4*(4*_C + e^(4*x))*e^(-3*x), 'linear']
```

Separable equations. These are equations of the form

$$P(x) = y'Q(x),$$

where P and Q are continuous functions on given intervals.

Example: $yy' = x$.

```
sage: desolve(y*diff(y,x) == x, y, show_method=True)
[1/2*y(x)^2 == 1/2*x^2 + _C, 'separable']
```

Caution! Sometimes Sage solves separable equations as exact. Example: $y' = \mathrm{e}^{x+y}$.

```
sage: desolve(diff(y,x) == exp(x+y), y, show_method=True)
[-(e^(x + y(x)) + 1)*e^(-y(x)) == _C, 'exact']
```

Bernoulli equations. These are equations of the form

$$y' + P(x)y = Q(x)y^\alpha,$$

where P and Q are continuous functions on given intervals and $\alpha \notin \{0, 1\}$. Example: $y' - y = xy^4$.

```
sage: desolve(diff(y,x)-y == x*y^4, y, show_method=True)
[e^x/(-1/3*(3*x - 1)*e^(3*x) + _C)^(1/3), 'bernoulli']
```

Homogeneous equations. These are equations of the form

$$y' = \frac{P(x,y)}{Q(x,y)},$$

where P and Q are homogeneous functions of same degree on given intervals.

Example: $x^2y' = y^2 + xy + x^2$.

```
sage: desolve(x^2*diff(y,x) == y^2+x*y+x^2, y, show_method=True)
```

```
[_C*x == e^(arctan(y(x)/x)), 'homogeneous']
```

Solutions are not given explicitly. We will see further on how to deal with these equations in some situations.

EXACT EQUATIONS. These are equations of the form

$$\frac{\partial f}{\partial x}\mathrm{d}x + \frac{\partial f}{\partial y}\mathrm{d}y,$$

where f is a differentiable function of two variables.

Example: $y' = \frac{\cos(y)-2x}{y+x\sin(y)}$ with $f = x^2 - x\cos y + y^2/2$.

```
sage: desolve(diff(y,x) == (cos(y)-2*x)/(y+x*sin(y)), y,
....:         show_method=True)
[x^2 - x*cos(y(x)) + 1/2*y(x)^2 == _C, 'exact']
```

Once again, solutions are not given explicitly.

RICCATI EQUATIONS. These are equations of the form

$$y' = P(x)y^2 + Q(x)y + R(x),$$

where P, Q and R are continuous functions on given intervals.

Example: $y' = xy^2 + \frac{1}{x}y - \frac{1}{x^2}$.

In this case, we set `contrib_ode` to `True` to make Sage use more complex methods.

```
sage: desolve(diff(y,x) == x*y^2+y/x-1/x^2, y,
....:         contrib_ode=True, show_method=True)[1]
'riccati'
```

LAGRANGE AND CLAIRAUT EQUATIONS. When the equation is of the form $y = xP(y') + Q(y')$ where P and Q are $\mathcal{C}^1$ on a given interval, it is a Lagrange equation. When P is the identity function, it is a Clairaut equation. Example: $y = xy' - y'^2$.

```
sage: desolve(y == x*diff(y,x)-diff(y,x)^2, y,
....:         contrib_ode=True, show_method=True)
[[y(x) == -_C^2 + _C*x, y(x) == 1/4*x^2], 'clairault']
```

Linear Equations. Let us solve $y' + 2y = x^2 - 2x + 3$:

```
sage: x = var('x'); y = function('y')(x)
```

```
sage: DE = diff(y,x)+2*y == x**2-2*x+3
sage: desolve(DE, y)
1/4*((2*x^2 - 2*x + 1)*e^(2*x) - 2*(2*x - 1)*e^(2*x) + 4*_C
+ 6*e^(2*x))*e^(-2*x)
```

We can rearrange the output with `expand`:

```
sage: desolve(DE, y).expand()
1/2*x^2 + _C*e^(-2*x) - 3/2*x + 9/4
```

It is thus convenient to use the form `desolve(...).expand()`. Let us check which method has been used:

```
sage: desolve(DE, y, show_method=True)[1]
'linear'
```

Let us add an initial condition, for instance $x(0) = 1$:

```
sage: desolve(DE, y, ics=[0,1]).expand()
1/2*x^2 - 3/2*x - 5/4*e^(-2*x) + 9/4
```

Separable Equations. Let us solve $y' \log(y) = y \sin(x)$:

```
sage: x = var('x'); y = function('y')(x)
sage: desolve(diff(y,x)*log(y) == y*sin(x), y, show_method=True)
[1/2*log(y(x))^2 == _C - cos(x), 'separable']
```

Sage agrees with us: it is a separable equation.

We should assign the solutions in order to use them later on:

```
sage: ed = desolve(diff(y,x)*log(y) == y*sin(x), y); ed
1/2*log(y(x))^2 == _C - cos(x)
```

Here, $y(x)$ is not explicitly given: $\frac{1}{2}\log^2\big(y(x)\big) = \_C - \cos(x)$.

We can get $y(x)$ explicitly using `solve`. Be aware that `ed` is an equation where `y` is a variable:

```
sage: solve(ed, y)
[y(x) == e^(-sqrt(2*_C - 2*cos(x))), y(x) == e^(sqrt(2*_C - 2*cos(x)))]
```

We should take care that `sqrt(2*_C - 2*cos(x))` may cause some problems even if Sage does not warn us. We will then assume that $\_C \geqslant 1$.

To draw the graph of solutions, we need their right-hand side. For instance, in order to get the first solution's right-hand side with $\_C = 5$, we could type:

```
sage: solve(ed, y)[0].substitute(_C==5).rhs()
Traceback (most recent call last):
...
NameError: name '_C' is not defined
```

$\_C$ has not been defined but only introduced by Sage. We can get it through the `variables()` command which gives the variables list:

```
sage: ed.variables()
(_C, x)
```

Only $\_C$ and x are variables, y having been defined as function of the variable x.

```
sage: c = ed.variables()[0]
sage: solve(ed, y)[0].substitute(c == 5).rhs()
e^(-sqrt(-2*cos(x) + 10))
```

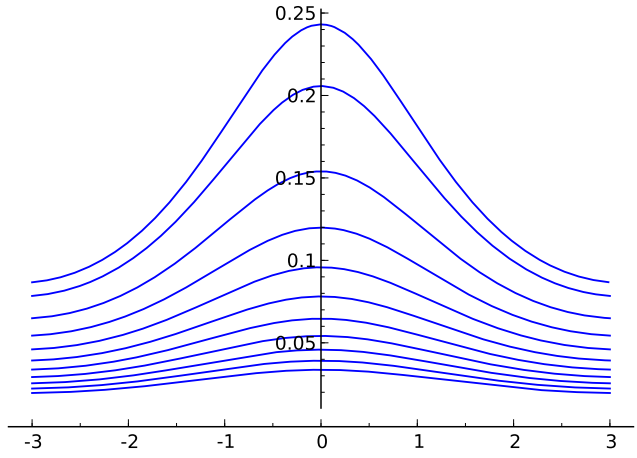
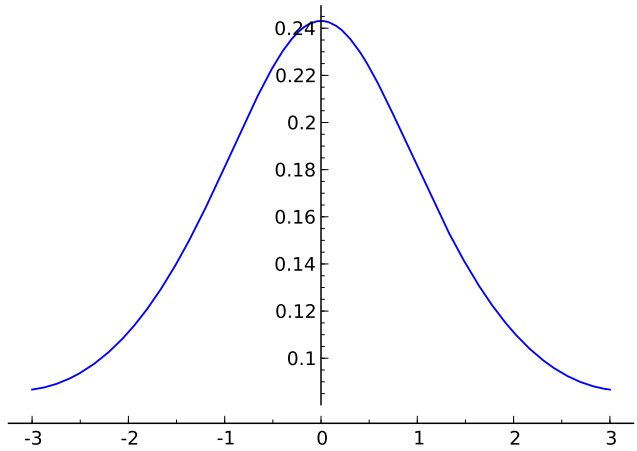


FIGURE 10.1 – Some solutions of $y' \log(y) = y \sin(x)$.

Another example with $\_C = 2$:

```
sage: plot(solve(ed, y)[0].substitute(c == 2).rhs(), x, -3, 3)
```

which gives:



To get several curves, (see Figure 10.1), we use a loop:

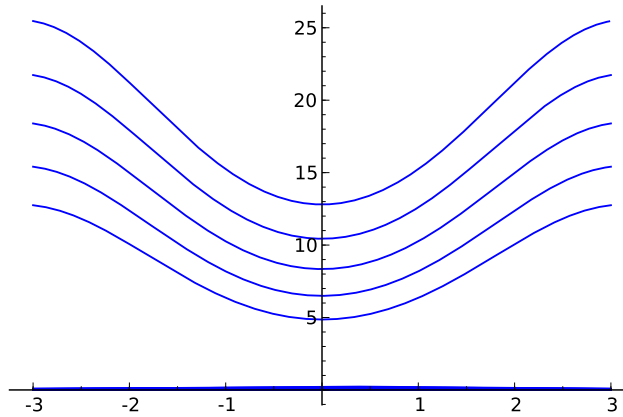
```
sage: P = Graphics()
sage: for k in range(1,20,2):
....:     P += plot(solve(ed, y)[0].substitute(c==1+k/4).rhs(), x, -3, 3)
```

We could have used a double loop in order to get the two solutions:

```
sage: P = Graphics()
sage: for j in [0,1]:
....:     for k in range(1,10,2):
```

```
....:         f = solve(ed,y)[j].substitute(c==2+0.25*k).rhs()
....:         P += plot(f, x, -3, 3)
sage: P
```

but the scales are too different to see both sets of curves:



Exercise 39 (Separable equations)**.** Find the solutions in $\mathbb{R}$ of these separable equations:

1. (E_1) : $\frac{yy'}{\sqrt{1+y^2}} = \sin(x)$;
2. (E_2) : $y' = \frac{\sin(x)}{\cos(y)}$.

Homogeneous Equations. We want to solve the differential equation $xy' = y + \sqrt{y^2 + x^2}$ which is homogeneous since

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{y + \sqrt{y^2 + x^2}}{x} = \frac{N(y,x)}{M(y,x)},$$

with $N(ky, kx) = kN(y, x)$ and $M(ky, kx) = kM(y, x)$.

We just need to introduce the change of variables $y(x) = x \cdot u(x)$ for all real x in order to get a separable equation.

```
sage: u = function('u')(x)
sage: y = x*u
sage: DE = x*diff(y,x) == y + sqrt(x**2 + y**2)
```

Let us change variables in the initial differential equation. The equation being undefined at 0, we solve it first on $]0, +\infty[$ and then on $]-\infty, 0[$.

```
sage: assume(x>0)
sage: desolve(DE, u)
x == _C*e^arcsinh(u(x))
```

We do not get u explicitly. We therefore use Maxima's `ev` command (as in *ev*aluate) with `logarc=True` in order to use the inverse hyperbolic functions as logarithms; u will then be expressed thanks to the `solve` command:

```
sage: S = desolve(DE,u)._maxima_().ev(logarc=True).sage().solve(u); S
[u(x) == -(sqrt(u(x)^2 + 1)*_C - x)/_C]
```

Here, `S` is a list containing a single equation; `S[0]` is therefore the equation itself.

Here we can observe that the equation is still implicitly solved thus we will ask Sage to solve the equivalent equation:

$$c^2(u^2+1) = (x-uc)^2,$$

via

```
sage: solu = (x-S[0]*c)^2; solu
(_C*u(x) - x)^2 == (u(x)^2 + 1)*_C^2
sage: sol = solu.solve(u); sol
[u(x) == -1/2*(_C^2 - x^2)/(_C*x)]
```

We then just need to go back to y:

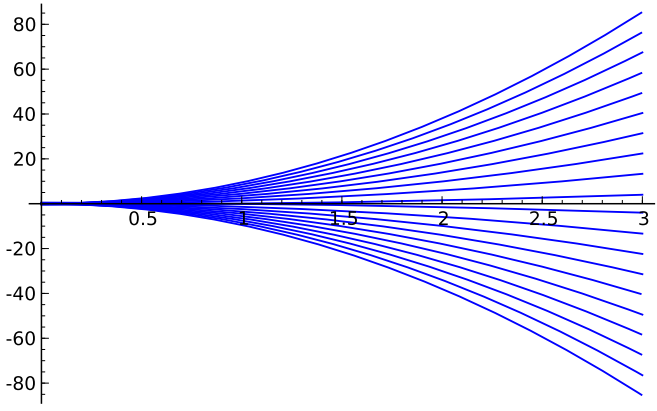
```
sage: y(x) = x*sol[0].rhs(); y(x)
-1/2*(_C^2 - x^2)/_C
```

And here are the explicit solutions!

$$y(x) = \frac{x^2-c^2}{2c}.$$

We then draw the solutions on $]0,+\infty[$, keeping in mind that $\_C$ must be a non-zero constant.

```
sage: c = y(x).variables()[0]
sage: P = Graphics()
sage: for k in range(-19,19,2):
....:     P += plot(y(x).substitute(c == 1/k), x, 0, 3)
sage: P
```



Exercise 40 (Homogeneous differential equations)**.** Solve the following homogeneous equation over $\mathbb{R}$: (E_5) : $xyy' = x^2+y^2$.

A Parametric Equation: the Verhulst Equation. The relative rate of growth of a population is a linearly decreasing function of the population. In order to study this, one can attempt to solve an equation of the form:

$$y' = ay - by^2,$$

with a and b being positive real parameters.

```
sage: x = var('x'); y = function('y')(x); a, b = var('a, b')
sage: DE = diff(y,x) - a*y == -b*y**2
sage: sol = desolve(DE,[y,x]); sol
-(log(b*y(x) - a) - log(y(x)))/a == _C + x
```

As usual, we do not get y explicitly. Let us try to isolate it with `solve`:

```
sage: Sol = solve(sol, y)[0]; Sol
log(y(x)) == _C*a + a*x + log(b*y(x) - a)
```

We still do not have an explicit solution. We group together the terms on the left-hand side and simplify this expression using `simplify_log()`:

```
sage: Sol(x) = Sol.lhs()-Sol.rhs(); Sol(x)
-_C*a - a*x - log(b*y(x) - a) + log(y(x))
sage: Sol = Sol.simplify_log(); Sol(x)
-_C*a - a*x + log(y(x)/(b*y(x) - a))
sage: solve(Sol, y)[0].simplify()
y(x) == a*e^(_C*a + a*x)/(b*e^(_C*a + a*x) - 1)
```

10.1.3 Second-Order Equations

Linear Ordinary Differential Equations with Constant Coefficients. Let us solve now a second-order linear ordinary differential equation with constant coefficients, for instance:

$$y'' + 3y = x^2 - 7x + 31.$$

Here we use the same syntax as for the first-order equations, the second derivative of y with respect to x is obtained with `diff(y, x, 2)`.

```
sage: x = var('x'); y = function('y')(x)
sage: DE = diff(y,x,2)+3*y == x^2-7*x+31
sage: desolve(DE, y).expand()
1/3*x^2 + _K2*cos(sqrt(3)*x) + _K1*sin(sqrt(3)*x) - 7/3*x + 91/9
```

Let us add initial conditions, for instance $y(0) = 1$ and $y'(0) = 2$:

```
sage: desolve(DE, y, ics=[0,1,2]).expand()
1/3*x^2 + 13/9*sqrt(3)*sin(sqrt(3)*x) - 7/3*x - 82/9*cos(sqrt(3)*x) +
    91/9
```

or $y(0) = 1$ and $y(-1) = 0$:

```
sage: desolve(DE, y, ics=[0,1,-1,0]).expand()
```

```
1/3*x^2 - 7/3*x - 82/9*cos(sqrt(3))*sin(sqrt(3)*x)/sin(sqrt(3))
+ 115/9*sin(sqrt(3)*x)/sin(sqrt(3)) - 82/9*cos(sqrt(3)*x) + 91/9
```

that is

$$\frac{1}{3}\,x^2 - \frac{7}{3}\,x - \frac{82\,\sin(\sqrt{3}x)\cos(\sqrt{3})}{9\,\sin(\sqrt{3})} + \frac{115\,\sin(\sqrt{3}x)}{9\,\sin(\sqrt{3})} - \frac{82}{9}\,\cos(\sqrt{3}x) + \frac{91}{9}.$$

How to Solve a PDE: the Heat Equation. Next we will study the famous heat equation. The temperature z is distributed in a homogeneous rectilinear rod of length ℓ according to the equation (where x is the abscissa along the rod, and t the time):

$$\frac{\partial^2 z}{\partial x^2}(x,t) = C\frac{\partial z}{\partial t}(x,t).$$

This equation will be studied against the following initial conditions:

$$\forall t \in \mathbb{R}^+, \quad z(0,t) = 0 \quad z(\ell,t) = 0 \quad \forall x \in]0;\ell[, \quad z(x,0) = 1.$$

We will seek non-zero solutions of the form:

$$z(x,t) = f(x)g(t).$$

This is the method of separation of variables.

```
sage: x, t = var('x, t'); f = function('f')(x); g = function('g')(t)
sage: z = f*g
sage: eq(x,t) = diff(z,x,2) == diff(z,t); eq(x,t)
g(t)*diff(f(x), x, x) == f(x)*diff(g(t), t)
```

The equation thus becomes:

$$g(t)\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = f(x)\frac{\mathrm{d}g(t)}{\mathrm{d}t}.$$

Let us divide by $f(x)g(t)$, assumed not to be zero:

```
sage: eqn = eq/z; eqn(x,t)
diff(f(x), x, x)/f(x) == diff(g(t), t)/g(t)
```

We then obtain an equation where each side depends only on one variable:

$$\frac{1}{f(x)}\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2} = \frac{1}{g(t)}\frac{\mathrm{d}g(t)}{\mathrm{d}t}.$$

Each side can therefore only be constant. Let us separate equations and introduce a constant k:

```
sage: k = var('k')
sage: eq1(x,t) = eqn(x,t).lhs() == k; eq2(x,t) = eqn(x,t).rhs() == k
```

We solve the equations separately, beginning with the second one:

```
sage: g(t) = desolve(eq2(x,t),[g,t]); g(t)
```

```
_C*e^(k*t)
```

therefore $g(t) = ce^{kt}$ with c a constant. For the first one, we cannot do it directly:

```
sage: desolve(eq1,[f,x])
Traceback (most recent call last):
  ...
TypeError: ECL says: Maxima asks:
Is  k  positive, negative, or zero?
```

Let us use the `assume` mechanism:

```
sage: assume(k>0); desolve(eq1,[f,x])
_K1*e^(sqrt(k)*x) + _K2*e^(-sqrt(k)*x)
```

that is $f(x) = k_1 e^{x\sqrt{k}} + k_2 e^{-x\sqrt{k}}$.

10.1.4 The Laplace Transform

The Laplace transform converts a differential equation with initial conditions into an algebraic equation and the inverse transform then makes it possible to get back to the solution of the differential equation.

If f is a function defined on $\mathbb{R}$ and is identically zero on $]-\infty, 0[$, we call Laplace transform of f the function F defined, under certain conditions, by:

$$\mathcal{L}\big(f(x)\big) = F(s) = \int_0^{+\infty} e^{-sx} f(x)\, dx.$$

Laplace transforms are easily obtained from polynomial, trigonometric, exponential functions, and so on. These transforms have very interesting properties, especially concerning the transform of a derivative: if f' is a piecewise continuous function on $\mathbb{R}_+$ then

$$\mathcal{L}\big(f'(x)\big) = s\mathcal{L}\big(f(x)\big) - f(0),$$

and if f' satisfies the conditions imposed on f:

$$\mathcal{L}\big(f''(x)\big) = s^2\mathcal{L}\big(f(x)\big) - sf(0) - f'(0).$$

Example. We want to solve the differential equation $y'' - 3y' - 4y = \sin(x)$ using the Laplace transform with the initial conditions: $y(0) = 1$ and $y'(0) = -1$. Thus:

$$\mathcal{L}\left(y'' - 3y' - 4y\right) = \mathcal{L}\left(\sin(x)\right),$$

that is:

$$(s^2 - 3s - 4)\mathcal{L}(y) - sy(0) - y'(0) + 3y(0) = \mathcal{L}(\sin(x)).$$

In case we forgot the Laplace transforms of the most common functions, we can use Sage:

```
sage: x, s = var('x, s'); f = function('f')(x)
sage: f(x) = sin(x); f.laplace(x,s)
```

```
x |--> 1/(s^2 + 1)
```

Thus we get an expression of the Laplace transform of y:

$$\mathcal{L}(y) = \frac{1}{(s^2-3s-4)(s^2+1)} + \frac{s-4}{s^2-3s-4}.$$

Let us use Sage to get the inverse transform:

```
sage: X(s) = 1/(s^2-3*s-4)/(s^2+1) + (s-4)/(s^2-3*s-4)
sage: X(s).inverse_laplace(s, x)
3/34*cos(x) + 1/85*e^(4*x) + 9/10*e^(-x) - 5/34*sin(x)
```

If one wants to "cheat", one can decompose $X(s)$ into partial fractions first:

```
sage: X(s).partial_fraction()
1/34*(3*s - 5)/(s^2 + 1) + 9/10/(s + 1) + 1/85/(s - 4)
```

And all that remains is to read an inversion table. We can however use the black box `desolve_laplace` which will give the solution directly:

```
sage: x = var('x'); y = function('y')(x)
sage: eq = diff(y,x,x) - 3*diff(y,x) - 4*y - sin(x) == 0
sage: desolve_laplace(eq, y)
1/85*(17*y(0) + 17*D[0](y)(0) + 1)*e^(4*x) + 1/10*(8*y(0)
- 2*D[0](y)(0) - 1)*e^(-x) + 3/34*cos(x) - 5/34*sin(x)
sage: desolve_laplace(eq, y, ics=[0,1,-1])
3/34*cos(x) + 1/85*e^(4*x) + 9/10*e^(-x) - 5/34*sin(x)
```

10.1.5 Systems of Linear Differential Equations

A Simple Example of System of First-Order Linear Differential Equations. We want to solve the following system of linear differential equations

$$\begin{cases} y'(x) = A \cdot y(x) \\ y(0) = c \end{cases}$$

knowing that

$$A = \begin{bmatrix} 2 & -2 & 0 \\ -2 & 0 & 2 \\ 0 & 2 & 2 \end{bmatrix}, \qquad y(x) = \begin{bmatrix} y_1(x) \\ y_2(x) \\ y_3(x) \end{bmatrix}, \qquad c = \begin{bmatrix} 2 \\ 1 \\ -2 \end{bmatrix}.$$

We write:

```
sage: x = var('x'); y1 = function('y1')(x)
sage: y2 = function('y2')(x); y3 = function('y3')(x)
sage: y = vector([y1, y2, y3])
sage: A = matrix([[2,-2,0],[-2,0,2],[0,2,2]])
sage: system = [diff(y[i], x) - (A * y)[i] for i in range(3)]
sage: desolve_system(system, [y1, y2, y3], ics=[0,2,1,-2])
```

```
[y1(x) == e^(4*x) + e^(-2*x),
 y2(x) == -e^(4*x) + 2*e^(-2*x),
 y3(x) == -e^(4*x) - e^(-2*x)]
```

Here the syntax for the initial conditions is: `ics = [x0,y1(x0),y2(x0),y3(x0)]`.

A Matrix with Complex Eigenvalues. Let us consider now

$$A = \begin{bmatrix} 3 & -4 \\ 1 & 3 \end{bmatrix}, \qquad\qquad c = \begin{bmatrix} 2 \\ 0 \end{bmatrix}.$$

With Sage:

```
sage: x = var('x'); y1 = function('y1')(x); y2 = function('y2')(x)
sage: y = vector([y1,y2])
sage: A = matrix([[3,-4],[1,3]])
sage: system = [diff(y[i], x) - (A * y)[i] for i in range(2)]
sage: desolve_system(system, [y1, y2], ics=[0,2,0])
[y1(x) == 2*cos(2*x)*e^(3*x), y2(x) == e^(3*x)*sin(2*x)]
```

that is:

$$\begin{cases} y_1(x) = 2\cos(2x)\mathrm{e}^{3x} \\ y_2(x) = \sin(2x)\mathrm{e}^{3x}. \end{cases}$$

A Second-Order System. We want to solve the following system

$$\begin{cases} y_1''(x) - 2y_1(x) + 6y_2(x) - y_1'(x) - 3y_2'(x) = 0 \\ y_2''(x) + 2y_1(x) - 6y_2(x) - y_1'(x) + y_2'(x) = 0. \end{cases}$$

We reduce to a first-order system by setting

$$u = (u_1, u_2, u_3, u_4) = (y_1, y_2, y_1', y_2').$$

Thus we get:

$$\begin{cases} u_1' = u_3 \\ u_2' = u_4 \\ u_3' = 2u_1 - 6u_2 + u_3 + 3u_4 \\ u_4' = -2u_1 + 6u_2 + u_3 - u_4, \end{cases}$$

That is $u'(x) = A \cdot u(x)$ with

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 2 & -6 & 1 & 3 \\ -2 & 6 & 1 & -1 \end{bmatrix}.$$

With Sage:

```
sage: x = var('x'); u1 = function('u1')(x); u2 = function('u2')(x)
sage: u3 = function('u3')(x); u4 = function('u4')(x)
sage: u = vector([u1,u2,u3,u4])
sage: A = matrix([[0,0,1,0],[0,0,0,1],[2,-6,1,3],[-2,6,1,-1]])
sage: system = [diff(u[i], x) - (A*u)[i] for i in range(4)]
sage: sol = desolve_system(system, [u1, u2, u3, u4])
```

We will only consider the first two coordinates because we need y_1 and y_2, that is u_1 and u_2:

```
sage: sol[0]
u1(x) == 1/12*(2*u1(0) - 6*u2(0) + 5*u3(0) + 3*u4(0))*e^(2*x)
+ 1/24*(2*u1(0) - 6*u2(0) - u3(0) + 3*u4(0))*e^(-4*x) + 3/4*u1(0)
+ 3/4*u2(0) - 3/8*u3(0) - 3/8*u4(0)
sage: sol[1]
u2(x) == -1/12*(2*u1(0) - 6*u2(0) - u3(0) - 3*u4(0))*e^(2*x)
- 1/24*(2*u1(0) - 6*u2(0) - u3(0) + 3*u4(0))*e^(-4*x) + 1/4*u1(0)
+ 1/4*u2(0) - 1/8*u3(0) - 1/8*u4(0)
```

which can be summarised more concisely as:

$$\begin{cases} y_1(x) = k_1 \mathrm{e}^{2x} + k_2 \mathrm{e}^{-4x} + 3k_3 \\ y_2(x) = k_4 \mathrm{e}^{2x} - k_2 \mathrm{e}^{-4x} + k_3 \end{cases}$$

with k_1, k_2, k_3 and k_4 parameters depending on the initial conditions.

10.2 Recurrence Relations

10.2.1 Recurrences $u_{n+1} = f(u_n)$

Definition. Let $u_{n+1} = f(u_n)$ be a recurrence relation, with $u_0 = a$. We can define the relation naturally by means of a recursive algorithm. Let us consider for instance the logistic map (defined by $x_{n+1} = rx_n(1 - x_n)$):

$$f : x \mapsto 3.83 \cdot x \left(1 - \frac{x}{100\,000}\right) \quad \text{and} \quad u_0 = 20\,000.$$

With Sage:

```
sage: x = var('x'); f = function('f')(x)
sage: f(x) = 3.83*x*(1 - x/100000)
sage: def u(n):
....:     if n==0: return(20000)
....:     else: return f(u(n-1))
```

An iterative definition may be preferred:

```
sage: def v(n):
....:     V = 20000;
....:     for k in [1..n]:
....:         V = f(V)
....:     return V
```

Differential equations	
Variable declaration	`x=var('x')`
Function declaration	`y=function('y')(x)`
Solving an equation	`desolve(equation, y, <options>)`
Solving a system	`desolve_system([eq1, ...], [y1, ...], <options>)`
First-order initial conditions	$[x_0, y(x_0)]$
Second-order initial conditions	$[x_0, y(x_0), x_1, y(x_1)]$
	$[x_0, y(x_0), y'(x_0)]$
System initial conditions	$[x_0, y_1(x_0), y_2(x_0), ...]$
Independent variable	`ivar=x`
Resolution method	`show_method=True`
Call for special methods	`contrib_ode=True`

Laplace transform	
Laplace transform of $f : x \mapsto f(x)$	`f.laplace(x,s)`
Inverse transform of $X(s)$	`X(s).inverse_laplace(s,x)`
Solving an ODE with the Laplace transform	`desolve_laplace(equation,function)`

Miscellaneous commands	
First-order derivative	`diff(y,x)`
Expanding an expression	`expr.expand()`
Getting the variables	`expr.variables()`
Variable substitution	`expr.substitute(var==val)`

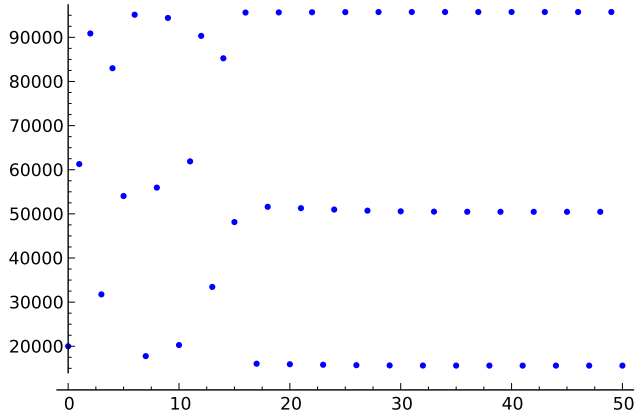
TABLE 10.1 – Useful commands for solving differential equations.

Graphical Representation. Let us plot the coordinates of (k, u_k):

```
sage: def cloud(u,n):
....:     L = [[0,u(0)]];
....:     for k in [1..n]:
....:         L += [[k,u(k)]]
....:     points(L).show()
```

From the following graph, we can assume the existence of three limit points:

```
sage: cloud(u,50)
```



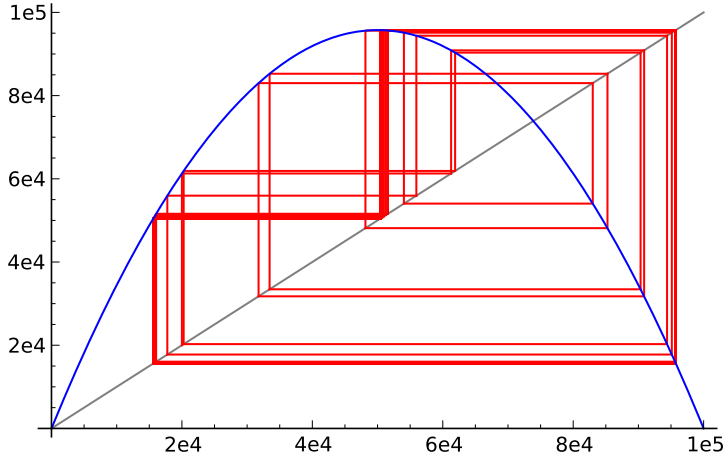
The representation involving the first bisector and the representative curve of f could have been preferred. Since this does not exist natively in Sage, we will build a small procedure that will do the job:

```
sage: def snail(f,x,u0,n,xmin,xmax):
....:     u = u0
....:     P = plot(x, x, xmin, xmax, color='gray')
....:     for i in range(n):
....:         P += line([[u,u],[u,f(u)],[f(u),f(u)]], color = 'red')
....:         u = f(u)
....:     P += f.plot(x, xmin, xmax, color='blue')
....:     P.show()
```

For instance, with the same relation:

```
sage: f(x) = 3.83*x*(1 - x/100000)
sage: snail(f,x,20000,100,0,100000)
```

The three limit points are highlighted:



10.2.2 Linear Recurrences with Rational Coefficients

Sage deals with relations of the following kind:

$$a_k u_{n+k} + a_{k-1} u_{n+k-1} + \cdots + a_1 u_{n+1} + a_0 u_n = 0,$$

the $(a_i)_{0 \leqslant i \leqslant k}$ being an indexed family of rational scalars.

For instance, consider the following relation:

$$u_0 = -1,\ u_1 = 1,\ u_{n+2} = \frac{3}{2} u_{n+1} - \frac{1}{2} u_n.$$

The well known function `rsolve` is not directly accessible. It must be picked up in SymPy, which causes some inconvenience such as syntax changes to declare variables. Here, for example, a preamble is necessary:

```
sage: from sympy import Function
sage: from sympy.abc import n
sage: u = Function('u')
```

The recurrence relation must then be defined as: $a_k u_{n+k} + \cdots + a_0 u_n = 0$. Here $u_{n+2} - \frac{3}{2} u_{n+1} + \frac{1}{2} u_n = 0$:

```
sage: f = u(n+2)-(3/2)*u(n+1)+(1/2)*u(n)
```

Finally, we use `rsolve`, observing how the initial conditions are declared (`u(0):value`, `u(1):value`, etc.):

```
sage: from sympy import rsolve
sage: rsolve(f, u(n), {u(0):-1,u(1):1})
3 - 4*2**(-n)
```

that is $u_n = 3 - \frac{1}{2^{n-2}}$.

10.2.3 Non-Homogeneous Linear Recurrence Relations

Sage also deals with relations of the following kind:

$$a_k(n) u_{n+k} + a_{k-1}(n) u_{n+k-1} + \cdots + a_1(n) u_{n+1} + a_0(n) u_n = f(n),$$

the $(a_i)_{0 \leqslant i \leqslant k}$ being an indexed family of polynomial, rational or hypergeometric functions of n.

The command will depend on the nature of $f(n)$:

- `rsolve_poly` if f is polynomial;
- `rsolve_ratio` if f is rational;
- `rsolve_hyper` if f is hypergeometric.

The coefficients $a_i(n)$ are given as a list $[a_0(n), \ldots, a_{k-1}(n), a_k(n)]$. For example, in order to study the complexity of merge sort, one has to study the following relation:

$$u_{n+1} = 2u_n + 2^{n+2}, \quad u_0 = 0.$$

The computation yields:

```
sage: from sympy import rsolve_hyper
sage: from sympy.abc import n
sage: rsolve_hyper([-2,1],2**(n+2),n)
2**n*C0 + 2**(n + 2)*(C0 + n/2)
```

and since $u_0 = 0$ gives `C0=0`, we obtain $u_n = n \cdot 2^{n+1}$.

Part III

Numerical Computation