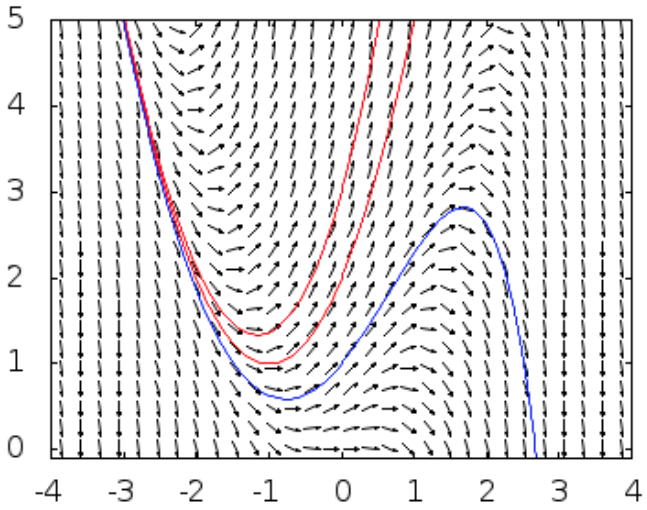
Exploring Math with Maxima

# Introductory Differential Equations

**with** Maxima




**Dr. Wolfgang Lindner**
dr.w.g.Lindner@gmail.com
Leichlingen, Germany
2023

# Contents

## Preface

In the 90ties of last century the CAS Derive was very popular in Austria and Germany and was profusely used in Mathematics courses in college and university. I was particularly impressed be the versatile and omnipresent Derive function `iterate/s`, which was programmed by Albert RICH and David STOUTEMYER. So I used a function `iterate` for MAXIMA just to see, how useful it is for downsizing the programming bureaucracy w.r.t. the use of loop and recurrence constructs.
It turned out that essential functions in the field of elementary Differential Equations shrank to 1-5 liners when using `iterate` as control structure - which is in my opinion a great benefit for the beginner in order to concentrate on the discussed methods and to provide a unified treatment (no index juggling necessary).
In this booklet there are only sketches of the mathematic reasoning given, because there is a vast set of specialized books and online informations available. For the mathematical treatment of Differential Equations (DE) parallel with this script, I recommend the books by BRONSON [9], BULIRSCH–STOER [46], BURDEN-FAIRES [10], LOWE/BERRY [29] and on the internet BAZETT [5], DAWKINS [13], HELLEVIK [21], LEBL [28], HAIRER/LUBICH [20], SÜLI [47] or AMMARI & AL. [1] – all cited in the bibliography.

The content of this script is limited to the treatment of some rudimentary analytical solution methods for Ordinary Differential Equations (ODE), e.g. from direct solution techniques to exact ODE's. The numerical solution methods for initial value problems (IVP) range from the classical EULER method via HEUN and Midpoint method to RUNGE–KUTTA `RK4` methods – all these methods are easily coded in MAXIMA using `iterate` and friends. Systems of ODEs are numerically solved using RK variants, likewise ODEs of $2^{nd}$ order. Boundary value problems (BVP) are only approximately solved with two methods: the *shooting method* and the *Finite Difference Method* (FDM).
Examples and exercises are borrowed from the above (quoted) sources, so the reader may control own solutions therein.

The collection of the MAXIMA examples and exercise scripts in this booklet not only want to help the reader to dive into the praxis of solving ordinary Differential equations and observing phenomena, but also to become comfortable with the use of the CAS MAXIMA in this field. Therefore, to program the necessary concepts as simply and directly as possible, I limit myself to the use of short self-made or build-in functions.
I hope that this collection of MAXIMA programs and the corresponding selection of examples from diverse sources will encourage the beginner to go further into the theory.

To run a MAXIMA script of this booklet no installation is necessary, *everything runs directly online*: a click on a link like ▷ Click here to Run. in this text is enough to invoke MAXIMA$^{on\ line}$. Such a link is at the same time a silent invitation to become active.
Of course, it is far better and more convenient to run MAXIMA in the wxMAXIMA variant on the PC. If you own a Mac or Linux PC, there is the option to install the app MAXIMA

free of charge and run the scripts of this booklet by *mark–copy–paste* into the wxMaxima window.

The author studied Numerical Mathematics from his academic teacher Prof. Dr. Roland Z. Bulirsch at the university of Cologne, Germany, in the 70ies of the last century. There I heard for the first time of the so-called shooting method to solve BVP's and I was impressed by the idea to transform a BVP to an IVP in order to use RK methods. At that time, we had to program in Fortran, coding our scripts on punch cards, which had to be delivered to the computer center. This has now all changed for the better, this booklet try to demonstrate this with Maxima for calculations and plots in the field of ODEs.

Sadly, Prof. Roland Bulirsch passed away in 2022.

Roland Z. Bulirsch, 1934–2022

User comments are very welcome.

Wolfgang Lindner
Leichlingen, Germany
April 2023
dr.w.g.Lindner@gmail.com

We use the following abbreviations.

| LEXICON *shorthand* | meaning |
|---|---|
| ODE | ordinary differential equation |
| IVP | initial value problem |
| BVP | boundary value problem |
| IC | initial condition(s) |
| BC | boundary condition(s) |

# 1 What is an Ordinary Differential Equation?

*Differential* equations make heavy use of *(partial) differentiating* and *integrating* functions. We start with a definition what a *(ordinary) differential equation* is. Then we code a procedure `iterate`, which we will use very often. A first use of `iterate` will be the construction of the so called successive PICARD-LINDELÖF approximations, which establish a constructive proof of the existence of solutions of ODE's under special conditions.

## 1.1 Definition of ODE

Let $U$ be subset of $\mathbb{R}^2$.
Let $f\colon U \to \mathbb{R}$ be a function on $U$.
Then we call

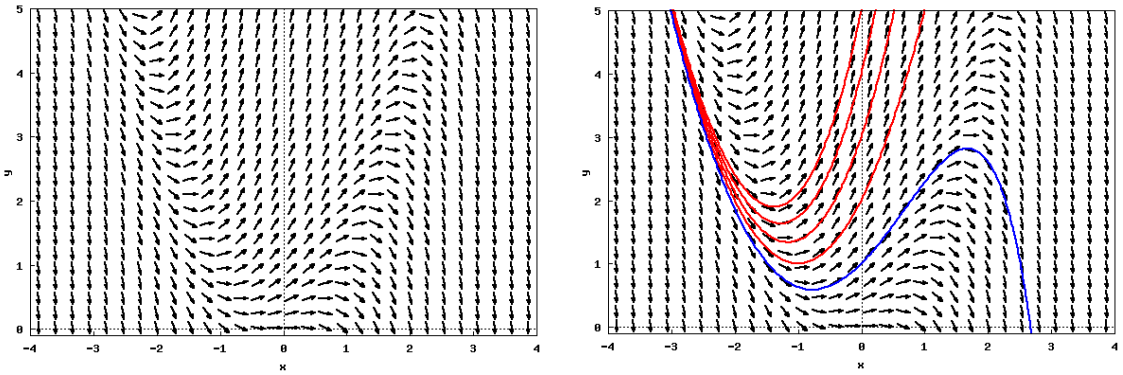
$$y' = f(x, y) \tag{1.1}$$

an *ordinary differential equation* (short: ODE), i.e.
we look for an interval $J \subset \mathbb{R}$ and a differentiable function $y\colon J \to \mathbb{R}$ with

$$(x, y(x)) \in U \qquad \text{for all} \quad x \in J \tag{1.2}$$

$$f(x, y(x)) = y'(x) \qquad \text{for all} \quad x \in J \tag{1.3}$$

• A given ODE's should be transformed in the "standard" form (1.1), because most facts and methods relay on this representation.
• As a first preview on what follows have a look at the following two figures:



*Basic perception/mental image of an ODE and its solutions:*
left: the direction field of ODE $y' = f(x, y) = y - x^2$.
right: the direction field of $y' = y - x^2$ including ...
Figure 1: ...: the special solution $y(x)$ with initial condition $y(0) = 1$.
...: the special solutions $y(x)$ going through the points $(0, n)$, $n = 2, 3, 4, 5$, i.e. $y(0) = n$,
i.e. $n$ is the initial value of $y(x)$ at $x = 0$.

**Comment.** The so called *direction field* gives a rough picture of an ordinary differential equation $y' = f(x, y)$ in a region $U$ of $\mathbb{R}^2$.
1. The left figure shows the *direction field* of the ordinary differential equation (ODE) $y' = f(x, y) := y - x^2$. At every point $(x, y)$ of the grid we plot a tiny directed line segment ('vector') whose slope is as great as the value $f(x, y)$ prescribe, e.g. at point $(1, 2)$ the slope is $f(1, 2) = 2 - 1^2 = 1$.
2. The right figure shows the same *direction field* of the given ODE, but with 5 special solution functions $y(x)$ inscribed. The blue solution function $y(x)$ goes across the point $(0, 1)$, which means that this solution fulfills the supplementary condition $y(0) = 1$.
We call this situation an *initial value problem* (IVP).

**Example 1.** (Plotting the direction field of an ODE)
Here is the code to reproduce the plot Fig.1.RHS using wxMaxima:

```
/* wxMaxima */
kill(functions,values,arrays)$
load(draw)$
wxdrawdf(y-x^2, [x,y],                        /* the given ODE  y'=y-x^2  */
    xaxis = true, yaxis = true, xlabel="x", ylabel="y",
    [x, -4, 4], [y, -0.1, 5],              /* chose interval  J=[-4, +4] */
    solns_at( [0,2],[0,3],[0,4],[0,5] ), /* the solution with y(0)=2 .. */
    color=blue,
    solns_at([0,1])         /* the solution with y(0)=1 painted in blue */
    )$
```

If you use e.g. Maxima$^{on\ line}$ instead of wxMaxima, then you must use a bit other syntax – the suffix `wx..` has to be dropped.

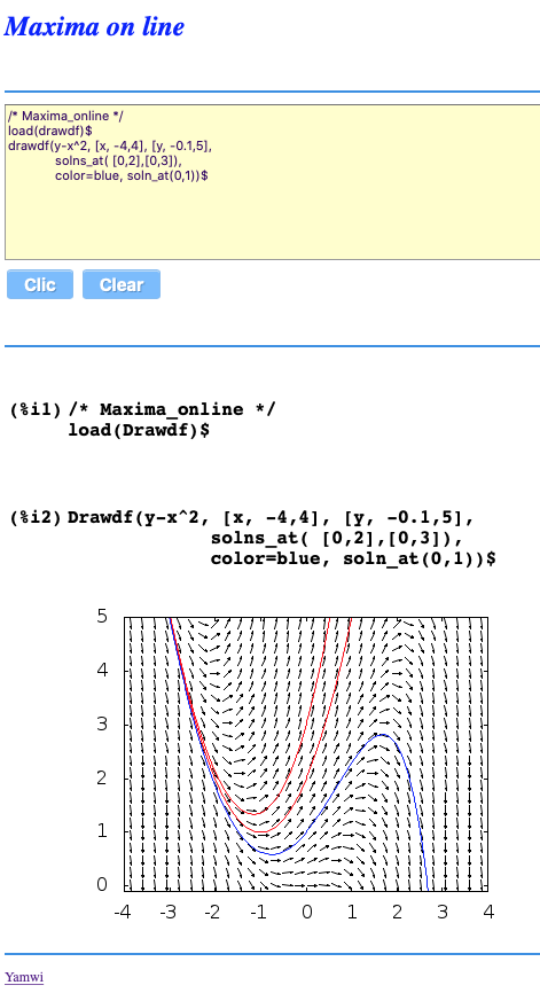
| LEXICON | Maxima$^{on\ line}$ | wxMaxima |
|---|---|---|
| *direction field* | `load(drawdf);` | load(draw); |
| | `drawdf(`$y - x^2$`);` | wxdrawdf($y - x^2$); |

Here is the above code prepared to copy & paste into the online frame of Maxima$^{on\ line}$:

```
/* Maxima_online */
load(drawdf)$
drawdf(y-x^2, [x, -4,4], [y, -0.1,5],
             solns_at( [0,2],[0,3]),
             color=blue, soln_at(0,1))$
```

▷ Mark-Copy-Paste the magenta code lines and RUN it with Maxima$^{on\ line}$.

The MAXIMA$^{on\ line}$ output:




**Remark.** (A hint for the use of wxMAXIMA resp. MAXIMA$^{on\ line}$)
If you see a link like

- ▷ Mark & Copy & Paste code and ☐RUN it.

you will be linked to MAXIMA$^{on\ line}$[1]to run, work or alter this (maybe a little adjusted) code script. You have the focus in MAXIMA$^{on\ line}$, if you click into the frame ☐ and it changed to blue ☐.

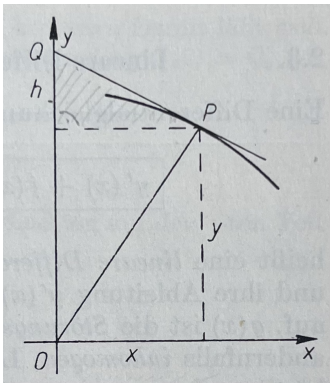
• Anyway: You should interpret such a link to *become active* and really run the blue code lines from an input cell [_] of wxMAXIMA or in the MAXIMA$^{on\ line}$ box ☐.

[1]Thanks to Mario RIOTORTO for his work on it.

**Example 2.** (Examples of ODE's)
**a.** ODE: $y' = -y + t + 1$.
Here $f(t, y) := -y + t + 1$.
An initial value problem would be $y' = -y + t + 1$ with $y(t = 0) = 2$.
**b.** ODE: $y' = y$.
Here $f(x, y) := y$. An initial value problem would be $y' = y$ and $y(1) = 0$.
**c.** ODE: $y' = 1 + y^2$.
Here $f(x, y) := 1 + y^2$. An initial value problem (IVP) would be $y' = 1 + y^2$ with $y(3) = 2$.
**d.** ODE: $y' = 2xy$.
Here $f(x, y) := 2 * x * y$. An IVP would be $y' = 2xy$ with $y(x = 4) = -2$.
**e.** ODE: $y' = t/y$.
Here $f(t, y) := ....$ An IVP would be $y' = -t/y$ with $y(0) = 2$.
**f.** ODE: $y' = y^2 \sin 2x$ on $U = ]0, 6[\times]0, 5[$.
Here $f(x, y) := ....$ An IVP would be $y' = f(x, y) = ..$ with $y(1.5) = 2$.

**Exercise 1.** Plot the direction fields of the ODE's of example 2 and the corresponding solution function of the IVP.

**Exercise 2.** (a geometric problem leading to an ODE)



Bräuning [12, p.45]

Determine those curves $y = f(x)$, where the cut point of the tangent with the ordinate axis from the origin of the coordinate system has the same distance as the touching point of the tangent with the curve.

*Hint*: Determine a corresponding ODE.
Restrict your solution to positive values of $x$ and $y$.

*Verify*: the ODE is $y - xy' = \sqrt{x^2 + y^2}$.
Plot the direction field of the ODE and some corresponding solution functions $y$.
What do you observe?

**Exercise 3.** Review on IVP: ▷ Hellevik: ivp.

## 1.2 Existence-Uniqueness Theorem - PICARD iteration

The Existence-Uniqueness Theorem for the ODE $y' = f(x,y) \wedge y(a) = b$ guaranties the existence of a locally unique solution $y$ under certain conditions on $f$, cf. e.g. [47, p.1 ff], [1, p.20], [10, p.238, P7] or [12, p.149], ▷ HELLEVIK: E and U. .

This solution $y$ is constructed with the *method of successive approximations* after PICARD-LINDELÖF with this recipe:
(1) start with the initial solution

$$y_0(x) := b. \tag{1.4}$$

(2) construct the *new approximating solution* function $y_{n+1}(x)$ through the following recursion formula for $n \in \{0, 1, 2 \dots\}$

$$y_{n+1}(x) := b + \int_a^x f(x, y_n(x))dx \tag{1.5}$$

Then $y_n(x) \overset{n\to\infty}{\longrightarrow} y(x) :=$ solution.

**Remark.** The RHS of (1.5) is only dependent of the variable $x$, because the process starts with the constant function $b$ via (1.4). Therefore the integral is direct calculable.

### 1.2.1 PICARD by hand

We construct the solution of the IVP $y' = 2xy,\ y(0) = 1$ along the recipe (1.4) and (1.5). We have: $f(x,y) := 2xy;\ a = 0;\ b = 1$.

$$y_0(x) := 1 \tag{1.6}$$

$$y_1(x) := 1 + \int_0^x 2x \cdot y_0(x)dx = 1 + x^2 \Big|_0^x = 1 + x^2 \tag{1.7}$$

$$\begin{aligned} y_2(x) &:= 1 + \int_0^x 2x \cdot y_1(x)dx = 1 + \int_0^x (2x + 2x^3)dx \\ &= (x^2 + x^4/4) \Big|_0^x = 1 + x^2 + \frac{x^4}{2} \end{aligned} \tag{1.8}$$

$$\begin{aligned} y_3(x) &:= 1 + \int_0^x 2x \cdot y_2(x)dx = 1 + \int_0^x (2x + 2x^3 + x^5)dx \\ &= (x^2 + x^4/4 + x^6/6) \Big|_0^x = 1 + x^2 + \frac{x^4}{2} + \frac{x^6}{6} \end{aligned} \tag{1.9}$$

$$\dots$$

$$y_n(x) := \frac{x^0}{0!} + \frac{x^2}{1!} + \frac{x^4}{2!} + \frac{x^6}{3!} + \frac{x^8}{4!} + \dots + \frac{x^{2n}}{n!} \tag{1.10}$$

Please watch, how the result $y_k$ of a previous step is substituted into the term $f(x, \square)$ for the next iterate $y_{k+1}$. This is made visible by using different colors for the $y_k$.

**Exercise 4.** Verify (1.10) via induction.

Following (1.10) we conclude:

$$y_n(x) := \sum_{k=0}^{n} \frac{(x^2)^k}{k!} \tag{1.11}$$

$$y(x) := \sum_{k=0}^{\infty} \frac{(x^2)^k}{k!} \equiv \exp(x^2) \tag{1.12}$$

We check with MAXIMA that $y(x) := e^{x^2}$ is indeed the solution of IVP $y' = 2xy$, $y(0) = 1$:

```
/* wxMaxima */
y(x):= exp(x^2);
y(0);
diff(y(x),x);
2*x*y(x);
```

▷ Mark-Copy-Paste the blue code lines and RUN it.

The wxMAXIMA output:

```
(%i4)   y(x):= exp(x^2);
        y(0);
        diff(y(x),x);
        2*x*y(x);
(%o1)   y(x):=exp(x^2)
(%o2)   1
(%o3)   2 x %e^x^2
(%o4)   2 x %e^x^2
```

**Comment.** Eq. (%o3) = (%o3) verifies, that $y = e^{x^2}$ is a solution of the ODE and (%o2) fulfills the IVP $y(0) = 1$. With the method of successive approximation along lines (1.6)...(1.10) we have constructed the solution of the IVP $y' = 2xy$, $y(0) = 1$ and therefore established the existence and uniqueness of the solution function $y$.

We will now implement this method in MAXIMA.

### 1.2.2 PICARD by MAXIMA

```
/* MAXIMA --- PICARD iteration --- */
fpprintprec: 5$
ratprint: false$
kill(functions, values, arrays)$        /* kill arrays !*/

f(x,y):= 2*x*y;
[a,b]: [0,1];                                        /* (0) */
y[0](x) := b;                                        /* (1) */
y[n](x) := b + integrate(f(t,y[n-1](t)),t,a,x);      /* (n) */

y[3](x), expand;                                     /* (3) */
makelist(y[n](x),n,0,4);                             /* (4) */
```

▷ Mark-Copy-Paste and RUN the code lines.

The wxMAXIMA output:

(%o4) $f(x,y):=2xy$

(%o5) $[0,1]$

(%o6) $y_0(x):=b$

(%o7) $$y_n(x):=b+\int_0^x f(t,y_{n-1}(t))\,dt$$

(%o8) $$\frac{x^6}{6}+\frac{x^4}{2}+x^2+1$$

(%o9) $$[1,x^2+1,\frac{x^4}{2}+x^2+1,\frac{x^6}{6}+\frac{x^4}{2}+x^2+1,\frac{x^8+4x^6+12x^4+24x^2}{24}+1]$$

**Comment.** In (0) we set the IV's to $a := 0$ and $b := 1$. Line (2) corresponds to (1.4) and line (n) is the translation of (1.5). Because $y[.]$ is an indexed variable (without creating a list first), an *undeclared* array (also called 'hashed' array) is created that grows dynamically with its indices: so a recursive process is established which allows to call a arbitrary function `y[k](x)` or a particular value `y[9](1.2)`, e.g. see (3). In (4) we display the first 5 approximate solution functions. We see in (%o9) this list shows the results (1.6) to (1.9).
• Warning: you should always start the use of a hashed array with kill(arrays); in order to have an empty array for a fresh use. Otherwise the old values in the array slots could irritate the computation!

Let's visualize the approximation process.

```
/* wxMaxima */
ratprint:false$
Pic: makelist([1,y[n](1)],n,0,4), numer;       /* (5) */

wxdraw2d(xaxis = true,
```

```
/* user_preamble="set size ratio -1", */   /* (6) */
xrange = [0,2], yrange = [-0.1,4],
xlabel="x", ylabel="y",
color=cyan,   explicit(y[0](x),x,0,2 ),    /* (7) */
color=blue,   explicit(y[1](x),x,0,2 ),
color=green,  explicit(y[2](x),x,0,2 ),
color=yellow, explicit(y[3](x),x,0,2 ),
color=red,    explicit(exp(x^2),x,0,2 ),   /* (8) */
point_size=2,  points_joined=false,
color=magenta, points( Pic ),
title="PICARD iteration")$
```

▷ Mark-Copy-Paste and RUN the code in wxMaxima.

The wxMaxima output:

```
(Pic) [[1,1],[1,2],[1,2.5],[1,2.6667],[1,2.7083]]
```

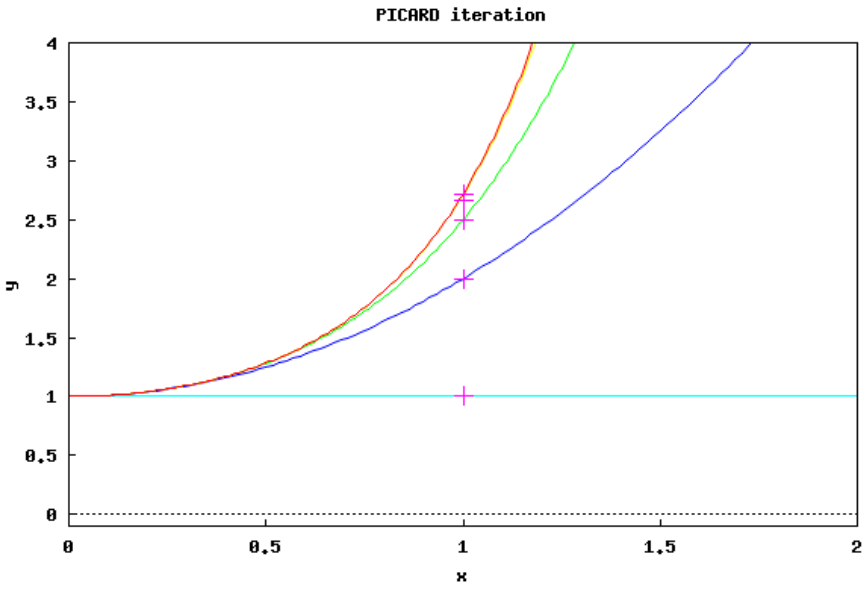



Figure 2: The plot shows the approximating functions $y[0](x)$, $y[1](x)$, $y[2](x), y[3](x)$, which tend to always better approach to the exact solution $y(x) = \exp(x^2)$.
The points list (5), which is $(1, y_n(1))_{n=0,1,2,3}$, tends to approach the limit value $y(1) = e^1 = e \approx 2.71828$. It is visualized as magenta crosses $+$.

**Comment.** In (5) we pick a point on each approximating function at the same position at $x = 1$ and collect them in the list `Pic`(ard). If you uncomment line (6) you get a an equal scale on each axis. In (7) we plot the start function $y_0$ as a constant line above the interval $[0, 2]$ in color cyan. (8) plots the exact solution function $y$.

**Exercises.**

**Exercise 5.** Construct the solution $y$ of the IVP $y' = x - y^2,\ y(0) = -0.5$ using PICARD iterates.
[Control: $y_6(x) = -\frac{1}{2} + \frac{1}{4}x + \frac{3}{8}x^2 + \frac{5}{48}x^3 + \dots$.

**Exercise 6.** Construct the solution $y$ of the IVP $y' = x^2 + y^2,\ y(0) = 0$ via successive approximation. Plot its direction field and show the solutions, which fulfill the IVP's $y(0) = -1, y(0) = 0$ and $y(0) = 1$.

**Exercise 7.** Given the IVP $dy/dx = x - y \wedge y = 1$ *at* $x = 0$, use PICARD's method to approximate $y$ when $x = 0.2$. [Control: $y_5(0.2) \approx 0.83746$. - Exact solution: $y = x - 1 + 2e^{-x}$.

**Exercise 8.** Calculate the solution $y$ of $y' = y^2 - xy$, which has value $y = 1$ for $x = 0$. [Control: $y_5(0.5) \approx 1.6987$. - Exact solution: $y(x) = (1 - \int_0^x e^{-1/2 \cdot t^2} dt)^{-1} \cdot e^{-1/2 \cdot x^2}$, cf. [24, p.308].

**Exercise 9.** Find the solution to the equation $y'(t) = 1 + y(t)^2$ with initial condition $y(t_0) = y_0 = 0, t_0 = 0$. Plot the first 4 PICARD iteration steps.
[Result: $y(x) = \tan(x)$
url: ▷ WIKI: Picard-Lindelöf.

**Exercise 10.** An ODE is given through $x' = \sin(t) - x$ with IC $x(0) = 1$.
Do two steps of the PICARD iteration.
See ▷: Wiki: Picard-Iteration

**Exercise 11.** Construct the PICARD iterates for the IVP $y' = 2t(y + 1),\ y(0) = 0$ and show that they converge to the exact solution $y(t) = e^{t^2} - 1$. Cf. [11, p.110].

**Exercise 12.** Construct the PICARD iterates for the initial value problem in [1, p.20, Example 2.7].

**Exercise 13.** Do example 2.3.2 at ▷ HELLEVIK: Taylor's method.

# 2 Analytical Solution Methods

In this chapter we speak about some types of ODEs, which can be solved without a special theory, but only with elementary knowledge of integration. We want to accompany these analytical procedures with support by MAXIMA.
These ODE types are:

| ODE: | *type* | solution method |
|---|---|---|
| I | $y' = f(x)$ | direct integration |
| II | $y' = f(y)$ | homogen ODE |
| III | $y' = f(y) \cdot g(x)$ | separation of variables |
| IV | $y' = f(x) \cdot y + g(x)$ | linear ODE |
| V | $y' = f(x/y)$ | variable transformation |
| VI | $h_x + h_y \cdot y' = 0$ | exact ODE |

To use one of the corresponding solution procedures one has to decide, in which of these shapes a given ODE falls.

## 2.1 ... the case $y' = f(x)$ : direct integration of ODE

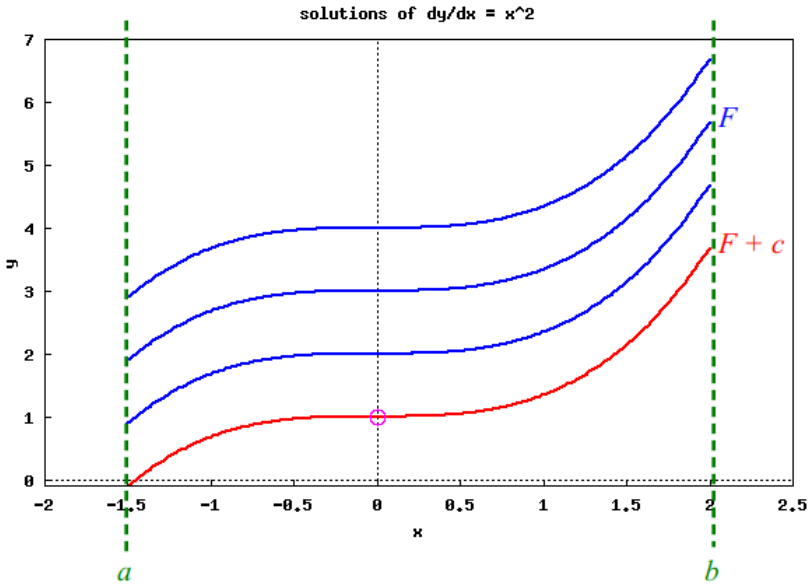



Figure 3: The plot shows four of the infinity many solution functions $y(x) = F(x) = \frac{1}{3}x^3 + c$ of the ODE $y' = x^2$ with initial value $y(0) = 1$. The special solution function $F + c$ with $y(0) = F(0) = 1$ is plotted in *red* and has $c = 1$.

- The ODE type I is: $y' = f(x)$, i.e. the right-hand side of the ODE depends only on $x$.

The theoretical solution method for an ODE of shape I: $y' = f(x)$ is by *direct integration*, i.e. if we know the *indefinite integral* [2] $F(x)$ for $f(x)$ on an interval $[a, b] \subset \mathbb{R}$, then *every other* solutions of $y' = f(x)$ is of the form $y(x) = F(x) + c,\ c \in \mathbb{R}$, see Fig. 3:

$$y(x) = F(x) := \int_a^x f(t)dt + c,\ c \in \mathbb{R} \tag{2.1}$$

The solution method for the IVP $y' = f(x),\ y(y_o) = x_o$ is in 2 steps:

I1: $\int_{x_o}^{x} f(t)dt + C = y(x)$

I2: solve $y(x_o) = y_o$ for $C$

I3: Solution: $y(x) + C$.

**Example 3.** The ODE $y' = x^2,\ y(0) = 1$ is of type I, because$f(x, y) = x = f(x\ only)$.
**a.** The solution function $y(x)$ is constructed in 3 steps:
I1: $\int_{x_o}^{x} t^2 dt + C = \frac{x^3}{3} + C = y(x)$.
I2: $y(x_o = 0) = \frac{0^3}{3} + C = 1 \rightsquigarrow C = 1$.
I3: Solution: $y(x) = \frac{x^3}{3} + 1$.
**b.** We write the solution process as a user defined MAXIMA function `odefx`:

```
/* MAXIMA --- ODE type I --- */
odefx(u,x,xo,yo):= block(
      I1: integrate(u,x,xo,x),                        /* (1) */
      I2: I1+C,                                       /* (2)  */
      I3: rhs( solve( at(I2,x=xo) = yo, C)[1]),       /* (3) */
      I4: I1+I3 )$                                    /* (4) */
```

The invoke arguments of `odefx(u,x,xo,yo)` are as follows:

1. `u`: the RHS of the ODE $u = f(x)$, which depends only on $x$
2. `x`: the integrating variable
3. `xo`: the initial value $x_o$ on the x-axis
4. `yo`: the initial value $y_o$ on the y-axis

Let's test our function `odefx`:

```
odefx(x^2, x,0,1);    /* (5) */
odefx(x^2, x,1,1);    /* (6) */
```

[2] (i.e. per definition $F' = f$)

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output for the test cases (5) and (6) are:

$$(\%o2) \quad \frac{x^3}{3}+1$$

$$(\%o3) \quad \frac{x^3}{3}+\frac{2}{3}$$

**Comment.** In (1) we calculate the indefinite integral $F = \int f$ and add the integration constant $C$ to it in order to build all solutions. In (3) we put $x_o$ into I2 and solve this equation for $C$. With this $C$ we pick the particular IVP solution I4.

**c.** We now solve the ODE $y' = x^2$, $y(0) = 1$ with the build-in MAXIMA function `ode2`:

```
/* Maxima -- IVP example with build-in function ode2*/
'diff(y,x) = x^2;       /* the given ODE */
ode2(%,y,x);            /* solve ODE */
ic1(%,x=0,y=1);         /* solve ODE respecting the IVP y(0)=1 */
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output for the test cases (5) and (6) are:

$$(\%o5) \quad \frac{d}{dx}y = x^2$$

$$(\%o6) \quad y = \frac{x^3}{3}+\%c$$

$$(\%o7) \quad y = \frac{x^3+3}{3}$$

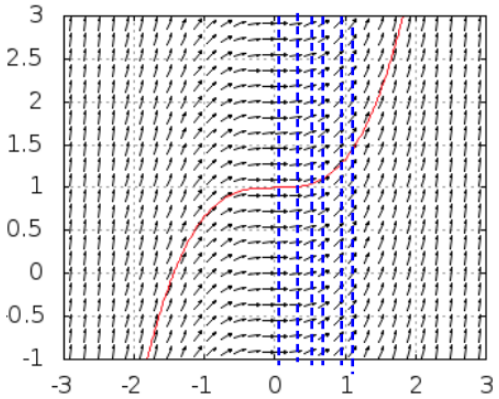



Figure 4: The direction field of the ODE $y' = x^2$ shows, that every point with the same x-coordinate has the same slope: this is visualized through the blue vertical lines, see [24, p.6].

**Exercises.**

Solve the following ODEs by hand, then using our function `odefx` and MAXIMA's `ode2`. Check the plausibility of the solution by an slope plot.

**Exercise 14.** Do some of the exercises of [28, p.21-26].
You will find the solutions at ▷ BAZETT: Integral solutions.

**Exercise 15.** Solve $x^2 \cdot y''(x) = 3x^4 - 1$ given $y = 0$ when $x = 1$ and $y'(1) = 2$.
Cf. LOWE [29, p. 35].

**Exercise 16.** Solve $y'(x) = \frac{1}{x} - \sin(x)$.

**Exercise 17.** Solve $\sqrt{1 - x^2} \cdot y'(x) = 1$.

**Exercise 18.** (LOWE [29, p.38]) The height $h$ of a ball thrown vertically upwards from ground level satisfies the equation $h''(t) = -10$.
The initial speed of the ball $h'(t)$ is 20 m/s. .
**a.** Find the time to reach the highest point.
**b.** How high will the ball go?
**c.** What is the total time of the flight?
**d.** Find the spees with which the ball hits the ground.

**Exercise 19.** Do the problems of Case 1 of ▷ FAN/CIFARELLI: ODEs.

## 2.2 ... the case $y' = f(y)$

• The ODE type II is: $y' = f(y)$, i.e. the right-hand side of the ODE depends only on $y$.

A rough theoretical derivation should motivate the MAXIMA code for this kind of ODE, see [22, p.140]. We have:

$$y' = f(y) \overset{1)}{\rightsquigarrow} \frac{y'}{f(y)} = 1 \overset{2)}{\rightsquigarrow} (F(y))' \overset{3)}{=} \frac{1}{f(y)} \cdot y' = 1 \rightsquigarrow F(y) = x + c \overset{4)}{\rightsquigarrow} y = F^{-1}(x + c)$$

under some assumptions[3].

**Example 4.** The ODE $y' = \cos^2(y),\ y(1) = 1$ is of type II,
because $f(x, y) = \cos^2(y) = f(y\ only)$.
**a.** $y' = \cos^2(y) \overset{1)}{\rightsquigarrow} \frac{y'}{\cos^2(y)} = 1 \rightsquigarrow (\tan(y)))' = 1 \rightsquigarrow \tan(y) = x + c \rightsquigarrow y = \arctan(x + c)$
for functions with values in $[-\frac{\pi}{2}, +\frac{\pi}{2}]$. Solution: $y(x) = \arctan(x + c)$.
**b.** We write the theoretical solution process as a user defined MAXIMA function `odefy`, which allows to follow a semi-automatic solution process and gives insight into the black-box `ode2(.)`:

```
/* MAXIMA --- ODE type II --- */
odefy(u,y,xo,yo):= block(
      F: integrate(u,y),
      /* print(F), */
    Sol: solve(F=x+C,y)[1],
     eq: at(Sol,[x=xo,y=yo]),
      c: rhs(solve(eq,C)[1]),
     I4: [Sol, c])$
```

The invoke arguments of `odefy(u,y,xo,yo)` are as follows:

1. `u`: the RHS of the ODE $u = f(y)$, which depends only on $y$
2. `y`: the integrating variable

Let's test our function `odefy` (watch the term `u`, which is a function in `y` !):

```
odefy( 1/cos(y)^2, y, 1,1);
```

▷ Mark-Copy-Paste and RUN the code lines.

The output is (watch the intermediate variables `F, Sol, eq, c`) ..

[3] 1) $f(x) \neq 0$ 2) $F(y)$ is indefinit integral of $\frac{1}{f(y)}$ 3) $(F \circ y)' = F'(y) \cdot y' = \frac{1}{f(y)} y'$ 3) $F(y) = x + c$ is solvable for $x$. – We note $\rightsquigarrow$ ('leads to') instead of $\Leftrightarrow$ ('is equivalent') as a warning for not being 'fully' mathematical precise. So you should always check your solution by a derivative and/or a direction field: is the proposed solution plausible?

```
(F)     tan(y)
        solve: using arc-trig functions to get a solution.
        Some solutions will be lost.
(Sol)   y=atan(x+C)
(eq)    1=atan(C+1)
(c)     tan(1)-1
(I4)    [y=atan(x+C),tan(1)-1]
```

.. and allows to read off the result in `I4` as `y=atan(x+tan(1)-1)`.

**c.** We now solve the ODE $y' = \cos^2(y),\ y(0) = 1$ by invoking the build-in function `ode2`:

```
/* Maxima --- ODE type II using ode2 --- */
ode2( 'diff(y,x) = cos(y)^2,  y, x);
ic1(%,x=1,y=1);
solve(%,y);
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

```
(%o177)  tan(y)=x+%c
(%o178)  tan(y)=x+tan(1)-1
         solve: using arc-trig functions to get a solution.
         Some solutions will be lost.
(%o179)  [y=atan(x+tan(1)-1)]
```

Please observe, that MAXIMA's full-automatic solver `ode2` also presents its solution *not-inverted*: therefore we have to solve the `ic1` result for `y`. We get the same result as our semi-automatic solution with `odefy`.

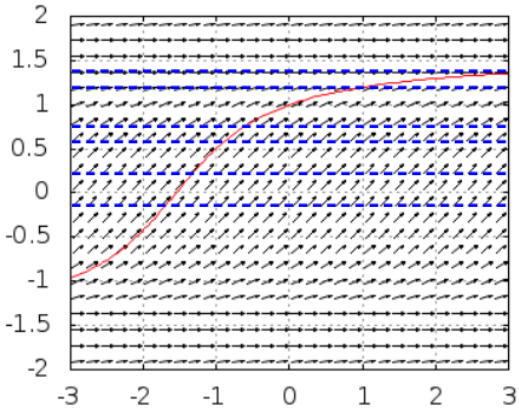



Figure 5: The direction field of the ODE $y' = 1/cos(y)^2$ shows, that every point on the same parallel w.r.t. the x-axis has the same slope: observe the blue horizontal lines, see [24, p.7].

**Exercises.**

Solve the following ODEs by hand, then using our function `odefy` and MAXIMA's `ode2`. Check the plausibility of the solution by an slope plot.

**Exercise 20.** Solve the ODE $y' = ky,\ k > 0$.

**Exercise 21.** Solve the ODE $y' = y^2,\ y(0) = 1$.

**Exercise 22.** Solve the ODE $y' = \sqrt{1 - y^2}$.

**Exercise 23.** Solve the corresponding IVP $y(1) = 2$ for the exercises 20–22.

**Exercise 24.** Do exercises 1.1.5, 1.1.6, 1.1.7 froml ▷ BAZETT: Integrals as solutions.

**Exercise 25.** Do the problems of Case 2 of ▷ FAN/CIFARELLI: ODEs.
If you like: do some of the Review problems.

## 2.3 ... the case $y' = f(x) \cdot g(y)$ : Separation of Variables

• The ODE type III is: $y' = f(x) \cdot g(y)$, i.e. the right-hand side of the ODE depends on $x, y$ – but is a product of two functions, which each depend on one of both variables alone.

This method is a generalization of the two special cases before: setting $f(x) := 1$ we get ODE shape I and setting $g(y) := 1$ we get ODE shape II.

A rough theoretical motivation is: write the given ODE in the old-fashioned 'differential form' as $y' \equiv \frac{dy}{dx} = f(x) \cdot g(y)$. Then it can be solved by the *method of separation of the variables*: collect terms with $y$ at LHS and terms with $x$ on RHS.

$$\begin{aligned} y' = \frac{dy}{dx} = f(x) \cdot g(y) \quad &\rightsquigarrow \quad \frac{1}{g(y)} dy = f(x)\, dx \\ &\rightsquigarrow \quad G(y) := \int_{yo}^{y} \frac{1}{g(y)} dy = (\int_{xo}^{x} f(x)\, dx) =: F(x) \\ &\rightsquigarrow \quad G(y) = F(x) + c \\ &\rightsquigarrow \quad y = .. \text{ hopefully solvable to } y \end{aligned}$$

**Example 5.** The ODE $y' + xy^2 = 0,\ y(0) = 2$ is of type III,
because $y' = f(x, y) = -xy^2 = h(x) \cdot g(y)$.

**a.** We argue: $y' = -xy^2 \overset{1)}{\rightsquigarrow} \frac{y'}{y^2} = -x \overset{2)}{\rightsquigarrow} (-\frac{1}{y})' = -x \rightsquigarrow \frac{1}{y} = \frac{1}{2}x^2 + C \rightsquigarrow y = \frac{2}{x^2+C}$
with the remark $^{1)}$: separate! $^{2)}$: $(-\frac{1}{y})' = (-\frac{1}{y(x)})' \overset{\text{Chain rule}}{=} (-y(x)^{-1})' = y(x)^{-2} \cdot y'(x)$.

**b.** We write the theoretical solution process as a user defined MAXIMA function `odefxgy`, which allows to follow a semi-automatic 'pedagogical' solution process :

```
/* MAXIMA --- ODE type III --- */
odefxgy(f,g, x,y, xo,yo):= block(
  F: integrate(f,x,xo,x),
  G: integrate(1/g,y,yo,y),
  print("DGL:", 'diff(y,x) = f*g ),
  print(".. initial condition:", y(xo) = yo),
  print("1. step - identify f(x) and g(y): ", f, " , ", g),
  print("2. step - calculate F(x)=", 'integrate(f,x,xo,x)= F),
  print("3. step - calculate G(y)=", 'integrate(1/g,y,yo,y) = G),
  print("4. step - set up Eq. F(x)=G(y):", F=G),
  print("5. step - solve  Eq. F(x)=G(y(x)) for y: ", Sol: solve(F=G,y)),
  print("6. step - choose solution with ", y(xo)=yo),
  print("solution: Sol ="),
Sol)$
```

The invoke arguments of `odefxgy(f,g, x,y, xo,yo)` should be self-explaining.

Let's test our function `odefxgy` for the example above $y' = f(x,y) = -xy^2 = h(x) \cdot g(y)$:

```
odefxgy(-x,   y^2, x,y, 0,1);
/*     f(x)   g(y) */
```

▷ Mark-Copy-Paste and RUN the code lines.

The output of the dialog is ...

*Is y-1 positive, negative or zero?*p;

*DGL:* $\frac{d}{dx}y = -x\,y^2$

*.. initial condition:* y(0)=1

*1. step - identify f(x) and g(y):* $-x$ , $y^2$

*2. step - calculate F(x)=* $-\int_0^x x\,dx = -\frac{x^2}{2}$

*3. step - calculate G(y)=* $\int_1^y \frac{1}{y^2}dy = 1-\frac{1}{y}$

*4. step - set up Eq. F(x)=G(y):* $-\frac{x^2}{2} = 1-\frac{1}{y}$

*5. step - solve Eq. F(x)=G(y(x)) for y:* $[y=\frac{2}{x^2+2}]$

*6. step - choose solution with* y(0)=1

*solution: Sol =*

(%o35) $[y=\frac{2}{x^2+2}]$

.. and allows to read off the result in (%o35) $y=\frac{2}{x^2+2}$.

**c.** We now solve this IVP by invoking the build-in function `ode2`:

```
/* Maxima -- ODE type III by ode2 -- */
ode2( 'diff(y,x) = -x*y^2,  y, x);
ic1(%,x=0,y=1);
solve(%,y);
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

(%o54) $\frac{1}{y} = \frac{x^2}{2} + \%c$

(%o55) $\frac{1}{y} = \frac{x^2+2}{2}$

(%o56) $[y=\frac{2}{x^2+2}]$

Please observe, that MAXIMA's full-automatic solver `ode2` also needs help by *inverting* the `ic1` result for `y`. We get the same result as our semi-automatic solution with `odefxgy`.

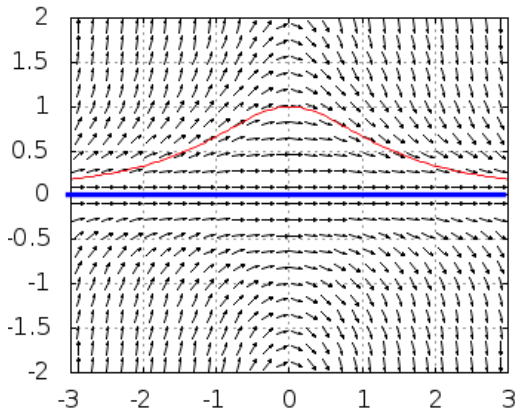



Figure 6: The direction field of the ODE $y' = xy^2$ shows a kind of symmetry. But it also shows that this ODE has the function $y(x) \equiv 0$ (blue line) as an – maybe overseen – solution.

**d.** If we prefer to have a compact function for solving separable ODE's without commenting hints we can do:

```
/* MAXIMA --- ODE type III : SEPARATION OF VARIABLES  --- */
separable(f,g, x,y, xo,yo):=
                  solve( integrate(1/g,y,yo,y)=integrate(f,x,xo,x), y);

separable(-x,y^2, x,y, 0,1);
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

```
        Is y-1 positive, negative or zero?p;

(%o63)  [y=2/(x^2+2)]
```

**Exercises.**

Solve the following ODEs by hand, then using our functions `odefxfy, separable` and MAXIMA's `ode2`. Check the plausibility of the solution by an slope plot.

**Exercise 26.** Solve the ODE $(1+x^2)xy\frac{dy}{dx} = 1-y^2$, cf. [27, p.239].

**Exercise 27.** Solve the IVP $\sin^2 x + (\frac{dy}{dx})^2 = 1$ with $y(x_0) = y_0$, cf. [27, p.229].

**Exercise 28.** Why is the ODE $y' = x - y^2$ not solvable by separation of the variables?

**Exercise 29.** Solve the IVP $y' = -\frac{y^2}{2x+1}$, at $x = 0$ is $y = 1$, cf. [12, p.38].

**Exercise 30.** Solve the IVP $y' = x^2 - x^2 y$, at $x = 1$ is $y = 5$. Is $y(x) \equiv 1$ a solution?

**Exercise 31.** Solve the IVP $xy' = y \ln y,\ y(1) = 2$.

**Exercise 32.** Solve $t\varphi' + \varphi^2 - 1 = 0$.

**Exercise 33.** Solve $yy' = x,\ y(1) = 1$.

**Exercise 34.** A sky diver falls so that his speed changes according to $\frac{dv}{dt} = 10 - 0.3v$. Solve this differential equation given that $v = 0$ when $t = 0$. Plot the solution and state the terminal speed of the sky diver. See [6, p.281].

**Exercise 35.** The population P of a colony of insects grows by $\frac{dP}{dt} = \frac{P}{4}$.
Find the time taken for the population to double in size, cf: [6, p.281].

**Exercise 36.** Do some of the exercises of [28, p.33 ff].
You will find the solutions at ▷ BAZETT: Separable equations.

**Exercise 37.** Do some of the examples and exercises of ▷ DAWKINS, P.: Separable Equations.

**Exercise 38.** Do exercises 1.1.5, 1.1.6, 1.1.7 froml ▷ BAZETT: Integrals as solutions.

**Exercise 39.** Do example 1.2 from [1, p.7].

**Exercise 40.** Do some examples and review problems of ▷ FAN/CIFARELLI: Separable ODEs.

## 2.4 ... the case $y' = f(x) \cdot y + g(x)$ : linear ODE

• The ODE type IV is: $y' = f(x) \cdot y + g(x)$, i.e. the shape of the right-hand side of the ODE is in analogy to a straight line '$y = ax + b$'. The *linear part* is $f(x) \cdot y$ and the so called *inhomogeneity* is $g(x)$.

The solution method is called the *variation of the constant* and is a generalization of the two special cases before: setting $f(x) := 1$ we get ODE shape I and setting $g(y) := 1$ we get ODE shape II.

We first solve the linear part of the ODE and then the whole ODE.
**1**$^{st}$ solve linear part, ie. solve $y' = f(x) \cdot y,\ y(a) = b$. But this is case III which can be solved by the *method of separation of the variables* and leads to an explicit solution:

$$\begin{aligned} y' = \frac{dy}{dx} = f(x) \cdot y \quad &\rightsquigarrow \quad \frac{1}{y} dy = f(x)\, dx \rightsquigarrow \int_b^y \frac{1}{y} dy = \int_a^x f(t)\, dt \\ &\rightsquigarrow \quad \ln(y) - \ln(b) = \int_a^x f(t)\, dt \\ &\rightsquigarrow \quad y = b \cdot \exp(\int_a^x f(t)\, dt) \end{aligned}$$

**2**$^{st}$ solve the given ODE, ie. solve $y' = f(x)y + g(x),\ y(a) = b$. We make the ansatz (guess) to substitute the constant $b$ in $y(x) = b \cdot \int_a^x f(t)\, dt$ through an suitable function $\phi(x)$ of $x$, which fulfills the initial condition $\phi(a) = b$ and then assume the following function $h$ be a solution of the ode:

$$h(x) := \phi(x) \cdot \exp(\int_a^x f(t)\, dt) \tag{2.2}$$

This method is therefore called the *variation of the constant*. For the moment we also compactify the 2nd factor of $h$ to be $F(x) := \exp(\int_a^x f(t)\, dt)$, so we have:

$$\begin{aligned} y' \overset{ODE}{=} f(x) \cdot y + g(x) \quad &\rightsquigarrow \quad h'(x) = f(x) \cdot h(x) + g(x) \qquad \text{(because } h \text{ solves the ode)} \\ &\overset{(2.2)}{\rightsquigarrow} \quad \phi'(x) \cdot F(x) + \phi(x) \cdot f(x) \cdot F(x) = f(x) \cdot \phi(x) \cdot F(x) + g(x) \\ &\overset{:F(x)\neq 0}{\rightsquigarrow} \quad \phi'(x) \cdot F(x) = g(x) \rightsquigarrow \phi'(x) = \frac{g(x)}{F(x)} \\ &\rightsquigarrow \quad \phi(x) = b + \int_a^x \frac{g(t)}{F(t)}\, dt \end{aligned}$$

So we arrive at the explicit solution formula $y(x)$ of the linear ODE:

$$y(x) : \overset{(2.2)}{=} (b + \int_a^x \frac{g(u)}{\exp(\int_a^u f(t)\, dt)}\, du) \cdot \exp(\int_a^x f(t)\, dt) \tag{2.3}$$

The formula (2.3) is easily transformed to MAXIMA:

```
/* MAXIMA --- ODE type IV : linear ODE and VARIATION OF CONSTANT  --- */
linear1(f,g, x,y, xo,yo) :=
                (yo+integrate(g/exp(integrate(f,x,xo,x)), x,xo,x))
                  * exp(integrate(f,x,xo,x));
```

**Example 6.** The ODE $y' + y = 1,\ y(0) = 1$ is of type III,
because $y' = f(x,y) = -y + 1$ with $f = -y,\ g = 1,\ a = 0, b = 1$, cf. [29, p.65].
**a.** To solve by hand, we put the ODE in another form: $y'(x) + p(x)y(x) = q(x)$, i.e. $y' + y = 1, p = 1 = q$ and choose $M(x) := e^{\int_0^x p(x)dx} = e^{\int_0^x 1dx} = e^x$ as 'integrating factor':

$$y' + y = 1 \overset{\bullet M(x)}{\rightsquigarrow} e^x y' + e^x y = e^x \overset{\text{prod.rule}}{\rightsquigarrow} (ye^x)' = e^x \rightsquigarrow ye^x = e^x + C \overset{:e^x}{\rightsquigarrow} y = 1 + Ce^{-x}$$

Determine C: $1 = y(0) = 1 + Ce^{-0} \rightsquigarrow 1 = 1 + C \rightsquigarrow C = 0$

**b.** In a. we solved the ODE in a slightly modified method, called the *integrating factor method* alias the *variation of the constant*. We code their steps in the following MAXIMA function `VoC`, which allows to follow the steps of the semi-automatic 'pedagogical' solution process :

```
/* Maxima --ODE type IV - INTEGRATING FACTOR METHOD */
VoC(p,q, x,y, xo,yo):= block(
  M: exp(integrate(p,x)),  /* integrating factor */
  My: integrate(P*y,x),
  print("Step 1: check correct shape y'+py=q :  ", 'diff(y,x)+p*y=q),
  print("    ... with initial condition:", y(xo) = yo),
  print("Step 2 - choose integrating factor M(x) = exp(integral p) =", M),
  print("Step 3 - multiply ODE by M*(y'+py=q), integrate : ",
                                  M*y=integrate(M*q,x)),
  print("Step 4 - divide through M :", y=expand(integrate(M*q,x)/M)+C/M),
  c:   at(expand(integrate(M*q,x)/M+C/M), x=xo),
  print("Step 5 - calculate C for yo=y(xo) :", solve(yo=c,C) ));
```

The invoke arguments of `VoC(p,q, x,y, xo,yo)` should be self-explaining.
Let's test our function `VoC` for the example above $y' + y = 1,\ y(0) = 1$:

```
VoC(1, 1,x,y,0,1);
/*  p  q         */
```

▷ Mark-Copy-Paste and RUN the code lines.

The output of the dialog is ...

```
        Step 1: check correct shape y'+py=q :   d/dx y+y=1
            ... with initial condition: y(0)=1
        Step 2 - choose integrating factor M(x) = exp(∫p) = %e^x
        Step 3 - multiply ODE by M*(y'+py=q), integrate :  %e^x y=%e^x
        Step 4 - divide through M :  y=C %e^-x+1
        Step 5 - calculate C for yo=y(xo) :  [C=0]
(%o28)  [C=0]
```

.. and allows to read off the result in `(%o28) y≡ 1`.

**c.** We now solve this IVP by invoking the build-in function `ode2`:

```
/* Maxima -- ODE type IV -- */
ode2( 'diff(y,x) = -y+1,  y, x);
ic1(%,x=0,y=1);
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

```
(%o7)   y=%e^-x (%e^x+%c)
(%o8)   y=1
```

**c.** We now solve this IVP by invoking our user-bulid function `linear1`:

```
linear1(-1,0,x,y,0,1);
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

```
    (%o6) %exp(-x)
```

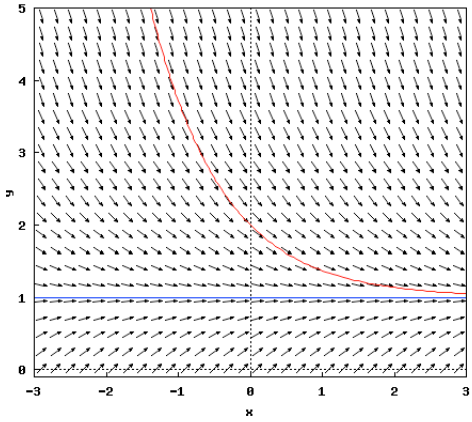



Figure 7: The direction field of the ODE $y' = 1 - y$ shows the special solution $y(x) = 1$ and the solution with IVP $y(0) = 2$.

**Exercises.**

Solve the following ODEs by hand, then using our functions `linear1`, `VoC` and MAXIMA's `ode2`. Check the plausibility of the solution by an slope plot.

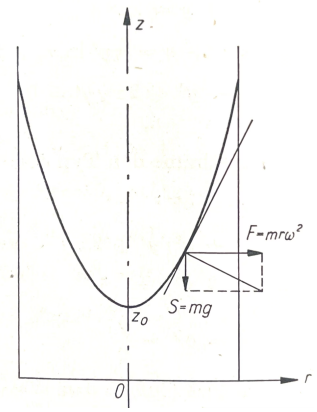
**Exercise 41.** Solve the ODE $y' = x + y$.

**Exercise 42.** Solve the ODE $y' - \frac{y}{x} = \frac{x-1}{x}$.

**Exercise 43.** Solve the IVP $4s\psi' - 2\psi = 3s^2,\ \psi(1) = 0$.

**Exercise 44.** Solve the IVP $\dot{z}\cos t + z\sin t = 1\ z(0) = 1$.

**Exercise 45.** A cylindrical vessel containing a liquid, rotates with the constant angular velocity $\omega$ around its vertical axis. Which is the shape of the liquid surface after entering the stationary state?




[Result: $z = \frac{\omega^2}{2g}r^2 + z_0$ (paraboloid of revolution) cf. [12, p.248].

**Exercise 46.** Do some of the exercises of [28, p.40 ff].
You will find the solutions at url ▷ BAZETT: Linear equations and the integrating factor.

**Exercise 47.** Do some of the examples and exercises of
▷ DAWKINS, P.: Linear Differential Equations.

**Exercise 48.** Do exercise 1.4 on p.9 of ▷ AMMARI: integrating factors.

## 2.5 ... the case $f_x + f_y \cdot y' = 0$ : exact ODE

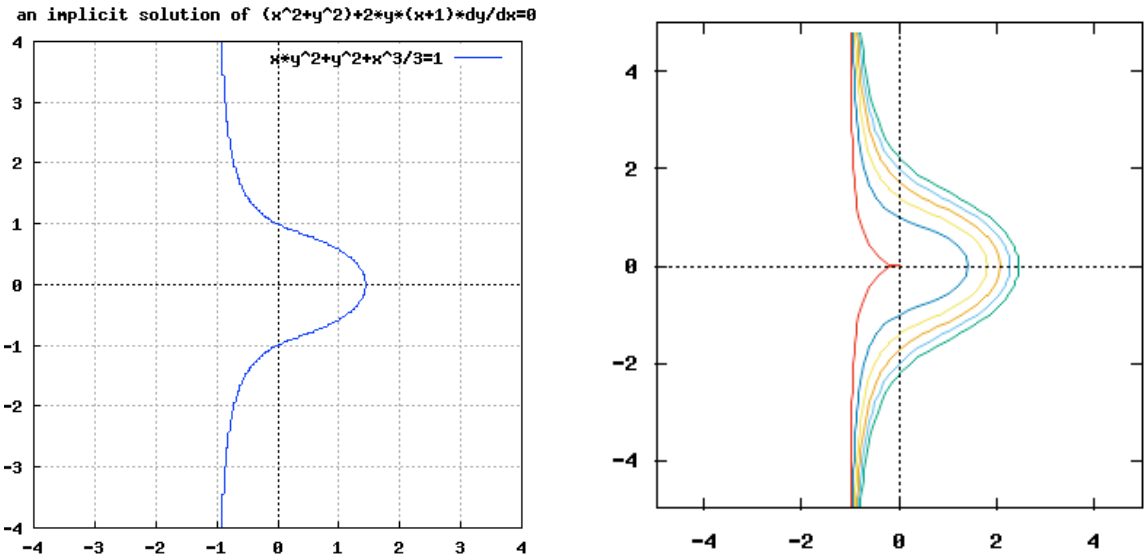



Figure 8: Left: The exact ODE $(x^2 + y^2) + 2 \cdot y \cdot (x + 1) \cdot y' = 0$ has an implicit solution $f(x, y) = C$, which is part of ..
Right: .. one of the contour lines of the contour plot of $f(x, y)$, i.e. $y(x)$ is solution, if its graph runs on an 'isocline' of same high.

• The ODE type V is: $P(x, y) + Q(x, y) \cdot y'(x) = 0$.[4] A function $y(x)$ is solution of such an ODE, if there exists a $C^1$-function $f(x, y)$ on a region in $\mathbb{R}^2$ with $f_x = P, f_y = Q$ and $f(x, y(x)) = const$ – because we then have $\frac{d}{dx} f(x, y(x)) = \frac{\partial}{\partial x} f(x, y) + \frac{\partial}{\partial y} f(x, y) \cdot y'(x) = P + Q \cdot y' = 0$, cf. [22, p.148]. ODEs of this comfortable type V are called *exact*.

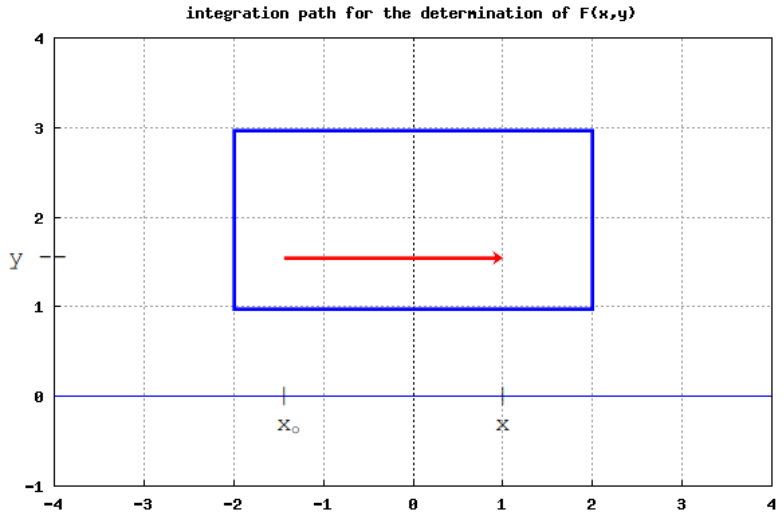
**Definition.** (*exact* ODE)
The ODE $P(x, y) + Q(x, y) \cdot y'(x) = 0$ is called *exact*,
iff there is a $C^1$-function $f(x, y)$ with $\frac{\partial f}{\partial x} \equiv f_x = P$ and $\frac{\partial f}{\partial y} \equiv f_y = Q$.

**Fact.** (*Test for exactness* )
IF we are given two $C^1$-functions $P(x, y)$ and $Q(x, y)$ on a rectangle $R \subset \mathbb{R}^2$,
THEN the ODE $P(x, y) + Q(x, y) \cdot y'(x) = 0$ is exact iff $P_y = Q_x$.

A rough theoretical argument and a method how to find such an function $f$ is as follows, cf. [22, p.149]: first determine the indefinite integral $F(x, y)$ of $P$ by integrating along the $x$-axis (i.e. $F_x = P$), e.g.

$$F(x, y) := \int_{xo}^{x} f(t, y) dt \tag{2.4}$$

[4] Often denoted as $P(x, y) \cdot dx + Q(x, y) \cdot dy = 0$.




Then choose an anonymous function $A(y)$, which only depends on $y$. It follows:

$$\begin{aligned} f(x,y) := F(x,y) + A(y) \quad &\Rightarrow \quad f_x = F_x + A_x(y) = P + 0 = P \\ &\rightsquigarrow \quad f_y = F_y + A'(y) = 0 + Q = Q \\ &\rightsquigarrow \quad A'(y) = Q - F_y \\ &\rightsquigarrow \quad A(y) = \int (Q - F_y)dy \\ &\rightsquigarrow \quad f(x,y) = \int f(x,y)dx + \int (Q - F_y)dy \end{aligned}$$

and because of $Q_x - F_{xy} = Q_x - P_y = 0$ the integrand $(Q - F_y)$ only depends on $y$. □

**Example 7.** The ODE $(x^2 + y^2) + 2 \cdot y \cdot (x + 1) \cdot y' = 0$ is of type V,
with $P(x,y) = (x^2 + y^2)$ and $Q(x,y) = 2 \cdot y \cdot (x + 1)$.

**a.** The ODE is exact: $P_y = 2y = Q_x$, ok.

**b.** We follow now the theoretical solution process to construct the solution:

$$\begin{aligned} F(x,y) \quad &:= \quad \int P(x,y)dx = \int (x^2 + y^2)dx = \frac{x^3}{3} + xy^2 \\ f(x,y) := F(x,y) + A(y) \quad &\Rightarrow \quad f_y = F_y + A'(y) = 0 + Q = Q \\ &\rightsquigarrow \quad A'(y) = Q - F_y = 2y \\ &\rightsquigarrow \quad A(y) := \int (Q - F_y)dy = y^2 + C \\ f(x,y) = F(x,y) + A(y) \quad &= \quad xy^2 + \frac{x^3}{3} + y^2 + C. \end{aligned}$$

**Maxima code.** Here is an implementation of a MAXIMA function `exact(.)`, which constructs an implicit solution $f(x,y) = C$ of an exact ODE $P(x,y) + Q(x,y) \cdot y'(x) = 0$ along this theoretical method. It goes along the steps $F \rightarrow ... \rightarrow f$.

```
/* MAXIMA --- ODE type V: EXACT ODE  --- */
exact(P,Q, x,y):=
  if is( diff(P,y) = diff(Q,x))              /* (1) */
  then
     block( [A,B,C,D],
             F: integrate(P,x),
             B: F + funmake(A, [y]),         /* (2) */
             C: diff(B,y) = Q,
             D: rhs((solve(C,A(y))[1])),     /* (3) */
             E: integrate(D,y),
             f: F+E  )
  else ("ODE is not exact.")$
```

*Comment*: In (1) we check, if the ODE is indeed exact. It it is, we work through the steps. In (2) we construct the helper function $A$ using MAXIMAs `funmake`[5] concept. In (3) we solve the equation C for $A$ and pick their right-hand side. This RHS is integrated along $y$ and gives the solution function $f(x, y)$.

The invoke arguments of `exact(P,Q, x,y)` should be self-explaining.

**Example 8.** Let's test our function `exact` for the ODE $(x^2 + y^2) + 2 \cdot y \cdot (x + 1) \cdot y' = 0$: **a.**

```
P: x^2+y^2;
Q: 2*y*(x+1);
exact(P,Q, x,y);

solve(f=1, y);          /* (4) */
wxplot2d (sqrt(3/(x+1)-x^3/(x+1))/sqrt(3), [x, -2, 2], [y, -1, 5])$
```

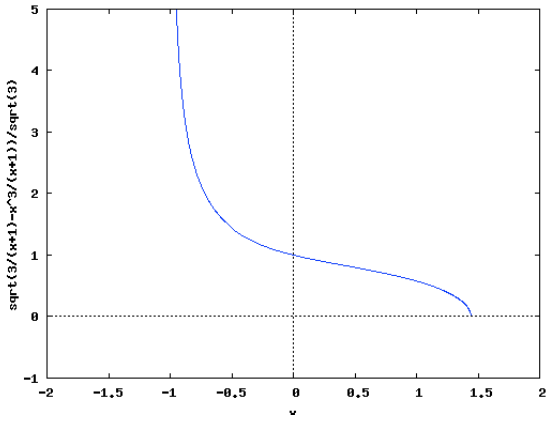
In (4) we solve the implicit solution $f(x, y(x)) = 1$ for an explicit solution $y(x) = ...$ The output of this dialog is[6]

(P) $y^2+x^2$

(Q) $2\,(x+1)\,y$

(%o50) $x\,y^2+y^2+\frac{x^3}{3}$

(%o51) $[y=-\frac{\sqrt{\frac{3}{x+1}-\frac{x^3}{x+1}}}{\sqrt{3}},y=\frac{\sqrt{\frac{3}{x+1}-\frac{x^3}{x+1}}}{\sqrt{3}}]$




[5]=`make` an anonymous `function` named $A$ dependent only on $y$.

[6]Using MAXIMA*on line* the last line of the code snippet reads

```
draw2d( explicit( sqrt(3/(x+1)-x*x*x/(x+1))/sqrt(3), x,-2,2));
```

*Comment*: Left: The implicit solution $f(x,y) = xy^2 + y^2 + x^3/3$ of the exact ODE $(x^2+y^2) + 2 \cdot y \cdot (x+1) \cdot y' = 0$ is displayed in (%o50). In (%o51 RHS) the explicit solution for $f = 1$ is displayed. Right: The graph of $y$ is plotted and corresponds to the $y > 0$ part of Fig.8 LHS.

▷ Mark-Copy-Paste and RUN the code lines.

**b.** We now solve this ODE by invoking the build-in function `ode2`:

```
/* Maxima -- ODE type V EXACT ODE -- */
(x^2+y^2)+2*y*(x+1)*'diff(y,x)=0;
ode2(%,y,x);
solve(%,y);
```

The MAXIMA output is:

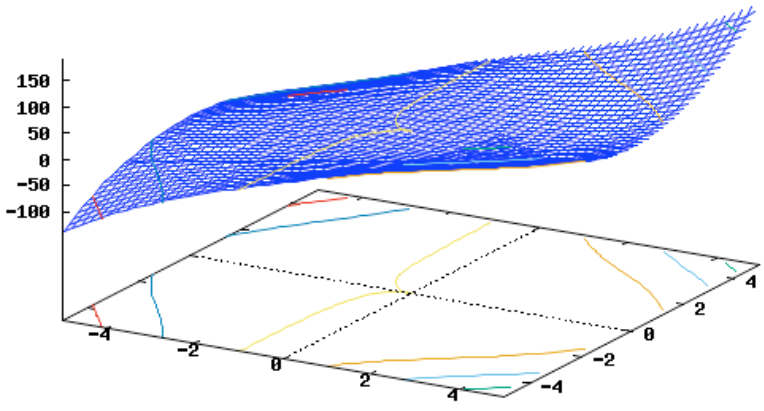
$$(\%o64) \quad 2\,(x+1)\,y\left(\frac{d}{d\,x}\,y\right)+y^2+x^2=0$$

$$(\%o65) \quad \frac{(3\,x+3)\,y^2+x^3}{3}=\%c$$

Please observe, that MAXIMA's full-automatic solver `ode2` also needs help by calculating the *explicit* solution `y`. We get the same result as our semi-automatic solution with `exact(.)`.

• Looking back, we see that an explicit solution $y(x)$ of an exact ode is always part of an niveau line of the implicit solution $f(x,y)$:

```
wxdraw3d(  user_preamble="set size ratio -1",
           xaxis=true, yaxis=true,
         explicit( x*y^2+y^2+x^3/3, x,-5,5, y,-5,5),
           contour_levels = 10,
           /* contour_levels = {0,1,2,3,4,5}, */
           contour        = both ) $
```



**Exercises.**

Solve the following ODEs by hand, using our functions `exact` and MAXIMA's `ode2`. Check the plausibility of the solution by an slope plot.

**Exercise 49.** Solve the ODE $2xy\,dx + x^2\,dy = 0$. Cf. [29, p.89].

**Exercise 50.** Solve the ODE $y' = -\frac{3x^2+4xy}{2x^2+2y}$.

**Exercise 51.** Solve $(xe^y + \cos(y))y' = -e^y$. Cf. [16, p.61].

Here are some exercises from KRYSICKI [27, p.299 ff]:

**Exercise 52.** Solve the ODE $2x - y + (4y - x)\frac{dy}{dx} = 0$.

**Exercise 53.** Solve the ODE $e^x(1 + e^y) = -e^y(1 + e^x)y'$.

**Exercise 54.** Solve the ODE $(\frac{x}{y} - 2y)y' = 2x - \ln y$.

**Exercise 55.** Do some of the examples and exercises of [28, p.63 ff].
You will find the solutions at ▷ BAZETT: Exact equations.

**Exercise 56.** Do some of the examples of ▷ DAWKINS: Exact Equations.

**Exercise 57.** Read §1.3.2 and do examples 1.12 – 1.14 on p.11 of ▷ AMMARI: Exact equations.

**Exercise 58.** Read about isoclines and do some of the problems of ▷ FAN/CIFARELLI: Isoclines.

## 2.6 ... the case $y' = f(\frac{y}{x})$ : substitutions

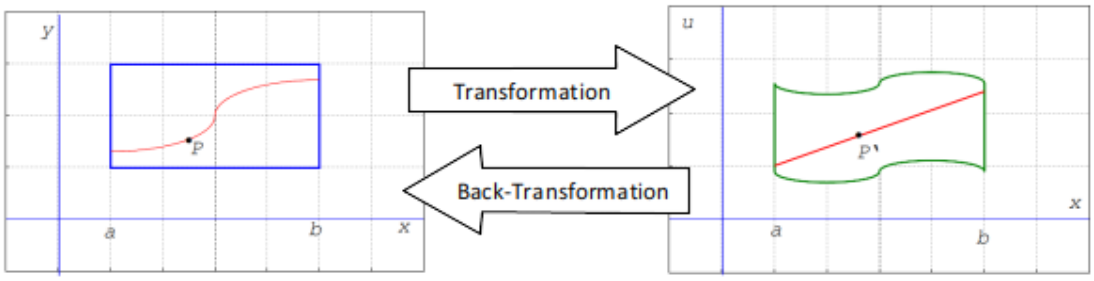



Figure 9: Sometimes the introduction of new coordinates may transform a given ODE in an equivalent, but better hand-able form. Often one transforms ('substitutes') the dependent variable $y$ into a new one $u$, then solves the transformed ODE and transforms this $u$-solution back to get a solution $y$ of the original ODE: $y(x) \to u(x) \to y(x)$.

• The ODE type VI is: $y' = f(\frac{y}{x})$, i.e. the right-hand side of an ODE of this type depends only on the quotient $\frac{y}{x}$. Such an ODE is called a *homogenous* ode and is solved after the substitution $u := \frac{y}{x}$ by the method of separation of variables.

**Definition.** (*homogenous* ODE)
The ODE $y'(x) = f(x, y)$ is called *homogenous*[7], iff $f(t^1x, t^1y) = f(x, y)$ for all $t \in \mathbb{R}$.

A rough theoretical motivation of the solution method is:

| LEXICON | *Transformation* | Back-Transformation |
|---|---|---|
| | $u(x) := \frac{y(x)}{x}$ | $y(x) = x \cdot u(x)$ |

It follows for the back-transformed $y(x)$ of a function $u(x)$

$$y(x) = x \cdot u(x) \quad \rightsquigarrow \quad y' = u + x \cdot u' \qquad (\rightsquigarrow y'' = 2u' + x \cdot u'' \dots) \tag{2.5}$$

$$\begin{aligned} y'(x) = f(y/x) \quad &= \quad u(x) + x \cdot u'(x) = f(u(x)) \\ &\rightsquigarrow \quad u'(x) = \frac{f(u) - u}{x} = \frac{du}{dx} \\ &\rightsquigarrow \quad \frac{f(u) - u}{du} = \frac{x}{dx} \\ &\rightsquigarrow \quad \text{now solve this ODE via separation of the variables} \\ &\rightsquigarrow \quad \text{and solve the original ODE via back-substitution.} \end{aligned} \tag{2.6}$$

[7] of grade 1

**Example 9.** Solve the IVP $y' = \frac{y+x}{x},\ y(0) = 2$.
**a.** The ODE is of type VI, because $y' = \frac{y+x}{x} = \frac{y}{x} + 1$.
**b.** We argue:

$$\begin{aligned} y' = \frac{y+x}{x} &\overset{(2.5)}{\rightsquigarrow} & u + x\frac{du}{dx} &= \frac{xu+x}{x} \\ &\rightsquigarrow & x\frac{du}{dx} &= 1 \\ &\overset{\text{separarte var's}}{\rightsquigarrow} & \frac{dx}{x} &= du \\ &\rightsquigarrow & u &= \ln|x| + C \\ &\overset{\text{back-subst}}{\rightsquigarrow} & \frac{y}{x} &= \ln|x| + C \\ &\rightsquigarrow & y(x) &= x\ln|x| + x\cdot C \end{aligned}$$

**Maxima code.** Here is a small script to run this solution method in MAXIMA.
It goes along the steps (2.5) to (2.6).

```
/* MAXIMA --- ODE type VI : HOMOGENOUS ODE --- */
kill(values,arrays)$
f(x,y) := (y+x)/x;                                          /* (1) */
eq1: u+x*du/dx = subst(u*x,y,f(x,y));                       /* (2) */
solve(eq1,du);                                              /* (3) */
SEPARABLE(g,h, x,y) := integrate(g,x)=integrate(1/h,y)$
eq2: SEPARABLE(1/x, 1, x,u) ;                               /* (4) */
eq3: subst(y/x, u, eq2);                                    /* (5) */
solve(eq3,y);                                               /* (6) */
```

*Comment*: In (1) we write down the RHL of the given ODE. (2) translates equation (2.5) into MAXIMA language and substitutes $u * x$ for $y$ in the RHS $f(x, y)$ of the ODE. This equation (eq1) is rearranged in (3) to give $du = ...$ Now this new ODE in variables $x, u$ is solved via the user-defined function SEPARABLE and the solution $u$ is saved in (eq2). In (4), we now back-substitute $y/x$ for $u$ in the $u$-solution (eq2) and resolve this for $y$. ok.

▷ Mark-Copy-Paste and RUN the code lines.

The output of the dialog is ...

```
(%o61)  f(x,y):=(y+x)/x
(eq1)   du x/dx+u=(u x+x)/x
(%o63)  [du=dx/x]
(eq2)   log(x)=u
(eq3)   log(x)=y/x
(%o67)  [y=x log(x)]
```

.. and displays the solution in (%o67) to be $y \overset{\text{Maxima}}{=} x\cdot\log(x) + C \overset{\text{Math}}{=} x\cdot\ln(x) + C$.

**c.** We solve this ODE by invoking the build-in function `ode2`:

```
'diff(y,x)  =  (y+x)/x ;
ode2(%,y,x);
method;
```

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output is:

$$\frac{d}{dx}y=\frac{y+x}{x} \qquad (\%o71)$$

$$y=x(\log(x)+\%c) \qquad (\%o72)$$

$$linear \qquad (\%o73)$$

Please observe, that MAXIMA's full-automatic solver `ode2` interprets this ODE as a linear one and therefore uses another solution method. Please do this as an exercise.

**Exercises.**

In the exercises we point also to other substitutions.

**Exercise 59.** Solve the ODE $x^2y' = y^2 + xy,\ y(1) = 1$.
Choose the substitution $u = y/x$. Cf. [28, p.46].

**Exercise 60.** Solve the ODE $y' = (x - y + 1)^2$.
Choose the substitution $u = x - y + 1$. Cf. [28, p.49].

**Exercise 61.** Do some of the exercises of [28, p.46 ff].
You will find the solutions at ▷ BAZETT: Substitutions.

**Exercise 62.** Do some of the examples and exercises of ▷ DAWKINS: Substitutions.

**Exercise 63.** Solve the IVP $y' = \sin(\frac{y}{x+1}) + \frac{y}{x+1}$, where for $x_0 = 1$ we have $y_0 = \pi/2$.
**a.** First use MAXIMAs `ode2` to solve the IVP.
Result by *Mathematica*, cf. [16, p.51]: $y(x) = 2(1 + x)\arctan((1 + x)/2 \cdot \tan(\pi/8))$.
**b.** Transform the IVP by means of the substitution $u(x) := \frac{y(x)-y_0}{x-x_0}$ in a separable ode.
**c.** Transform the IVP by means of the substitution $v(x) := \tan(u(x)/2)$ in a separable ode.
Compare with b.
PS: look for the MAXIMA function `isolate(expr,x)`. Maybe it is of some help ..

**Exercise 64.** (BERNOULLI equation) Study the type $y' + f(x) \cdot y + h(x) \cdot y^\alpha = 0,\ \alpha \neq 1$.
**a.** Transform the ODE by means of the substitution $u := y^{1-\alpha}$ in a separable ODE.
**b.** Solve the corresponding IVP in general for $y(a) = b$.
**c.** Solve the IVP $y' + \frac{y}{1+x} + (1 + x)y^4 = 0,\ y(0) = 1$. Cf. [26, p.91].

**Exercise 65.** Read §1.2.2 and do example 1.3 on p.8 of ▷ AMMARI: Change of variables.

# 3 Intermezzo: iterate

As prototype example for an iteration process we chose the well-known HERON algorithm to approximately calculate the square root of a reell number, see ▷ WIKI: HERON method. We first study this process by a hand calculation to get a feeling for the procedure and then introduce the corresponding MAXIMA function `iterate` to fully automate it.

## 3.1 iterate by hand

Let's calculate approximately $\sqrt{2}$ starting with $x = 3$, i.e. $\sqrt{2} \approx 3$. Watch the process:

| Step | *variable* | recurrence term |
|---|---|---|
| $n$ | $x$ | $f(x) = 0.5 \cdot (x + 2/x)$ |
| 1 | 3 | |
| 2 | 1.8333 | $f(\mathbf{3}) = 0.5 * (\mathbf{3} + 2/\mathbf{3}) \approx 1.8333$ |
| 3 | 1.4621 | $f(1.8333) = 0.5 * (1.8333 + 2/1.8333) \approx 1.4621$ |
| 4 | 1.415 | $f(1.4621) = 0.5 * (1.4621 + 2/1.4621) \approx 1.415$ |
| 5 | 1.4142 | $f(1.415) = 0.5 * (1.415 + 2/1.415) \approx 1.4142$ |

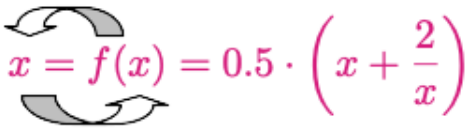
So we have the following recurrence (sequence) process, which after 5 iteration steps gives $\sqrt{2} \approx 1.4142$:

$$3 \overset{f}{\rightsquigarrow} 1.8333 \overset{f}{\rightsquigarrow} 1.4621 \overset{f}{\rightsquigarrow} 1.415 \overset{f}{\rightsquigarrow} 1.4142 \overset{f}{\rightsquigarrow} \dots$$

Or a little bit abstracted:

$$x_1 \overset{f}{\rightsquigarrow} x_2 \overset{f}{\rightsquigarrow} x_3 \overset{f}{\rightsquigarrow} x_4 \overset{f}{\rightsquigarrow} \dots \overset{f}{\rightsquigarrow} x_n \overset{f}{\rightsquigarrow} x_{n+1} = f(x_n)$$

Let's visualize this approximation process.



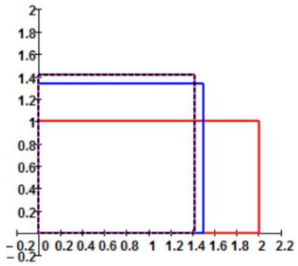




Figure 10: *Left:* *Recurrence*: The result $f(x)$ for its 'old' input $x$ is substituted in $f$ again and therefore recurs as new input $x$, i.e. in sequence language: $f(x_n) = x_{n+1}$.
*Right:* *Visualization*: the starting red rectangle with length $x = 2$ and width $y = 1$ is transformed in 3 steps to a square $\square$ with equal sides $\sqrt{2} \approx 1.4$.

• Following the pattern of the PICARD iterates we may code the HERON algorithm using a hashed array:

```
/* Maxima */
fpprintprec:5$ ratprint:false$ kill(functions, values, arrays)$

f(x):= 0.5*(x+2/x);              /* recurrence formula */
y[0](x) := 3;                    /* start value */
y[n](x) := f(y[n-1](x));        /* recursion:  y(n)  --f-->  y(n+1)  */

y[5](x);                        /*  calculated value */
```

▷ Mark-Copy-Paste and RUN the code lines.

The wxMaxima output:

```
(%o1) 1.4142
```

## 3.2 iterate by `iterate`

We now abstract the recurrence pattern from Fig.10 $x_0 \overset{u}{\rightsquigarrow} x_{new} \overset{u}{\rightsquigarrow} x_{newnew} \ldots$ to a Maxima function `iterate(u,x,x0,n)`[8], which automates this iterated substitution process. We implement two versions, a 1-dimensional version and a 2-dimensional version. Both are heavily used as a working horse for recurrences in the following chapters - despite its innocent looking code.

### 3.2.1 the one-dimensional function `iterate`

```
/* MAXIMA --- ITERATE --- */
iterate(u,x,xo,n) := block(
                     [l:[xo], numer:true, val:xo],              /* (1) */
                     for i thru n do
                        (val: subst(val,x,u), l: cons(val,l)),  /* (2) */
                     reverse(l));                               /* (3) */
```

The invoke arguments of `iterate(u,x,x0,n)` are as follows:

1. `u`: the recurrence formula $u(x)$, which depends on the variable $x$
2. `x`: the changing recurrence variable
3. `xo`: the initial value of $x$
4. `n`: the number of repeated substitutions

*Comment.* We elaborate on the code of `iterate`. (1) prepares two local variables inside the block structure: The list `l` is filled with the starting value `xo` of the iteration process

[8] This function mimics the omnipresent function iterate of CAS Derive, which was implemented by Albert Rich and David Stoutemyer. In this way we may connect Maxima to the Derive literature.

and will collect all following outputs $x_i$, the variable `val` gets the initial value `xo`. The `for` loop repeats the process (2) `n` times. In (2) variable `val` is actualized via `subst` by inserting the old `val` for `x` in the term `u(x)`. Afterwards this updated `val` is pushed in front of the current list `l` via `cons`. Because the newest value is now in front of the list, we invert it in (3) using `reverse` to see the latest value of $x$ at the end of the list.

**Example 10.** Let's test our function on the HERON algorithm to calculate $\sqrt{2}$:

```
fpprintprec: 5$                          /* (4) */
iterate( 0.5*(x+2/x),  x,3, 4);          /* (5) */
```

Here `iterate(..)` produces the first 5 iterations (i.e. repeating 4 times) of the recurrence relation $x_{n+1} = 0.5 \cdot (x_n + \frac{2}{x_n})$ starting with the given value $x_o = 3$ for `n=1`.

▷ Mark-Copy-Paste and RUN the code lines.

The MAXIMA output reproduce in `(%o3)` the results in §3.1:

```
(%i3)  fpprintprec:5$
       iterate(0.5*(x+2/x), x,3,4);  /* sqrt 2 */
(%o3)  [3,1.8333,1.4621,1.415,1.4142]
```

**Exercise 66.** Let $x > 0$. Take the HERON iteration $x_{n+1} = 0.5 \cdot (x_n + \frac{2}{x_n})$.
**a.** Verify: $x = 0.5 \cdot (x + \frac{2}{x}) \Leftrightarrow \cdots \Leftrightarrow x^2 - 2 = 0$.
**b.** Calculate `iterate( x*x - 2, x,3, 4);` – What do you observe? Consequence?

**Exercise 67.** Write a function `power(x,n):= iterate(...)` to compute the $n$-th power of a number $x$.

**Exercise 68.** Write a function `factorial(n):= iterate(.)` to compute $n!$ of a number $n$.

### 3.2.2 the two-dimensional function `iterate2`

```
/* MAXIMA --- ITERATE2 --- */
iterate2(u,x,xo,n) := block(
        [ l: [[xo[1],xo[2]]], numer:true,  val: [xo[1],xo[2]]],
        for i thru n do
           (val: subst([x[1]=val[1], x[2]=val[2]], u), l:cons(val,l)),
           reverse(l));
```

*Comment.* Function `iterate2` is very similar to `iterate`. But the local variables list `l` and the variable `val` are now filled with 'points' and `subst` awaits a list of defining equations[9].

**Example 11.** Let's test our function `iterate2` on the HERON algorithm to calculate $\sqrt{2}$.

[9] I thank Michel TALON for pointing me to this syntax difference.

```
fpprintprec: 5$
Sqrt2 :    iterate2([n+1, 0.5*(x+2/x)], [n,x], [1,3],4);      /* (6) */
transpose( iterate2([n+1, 0.5*(x+2/x)], [n,x], [1,3],4) );   /* (7) */
```

The MAXIMA output displays in `(Sqrt2)` the results in §3.2.1, supplemented by the counter in the first slot:

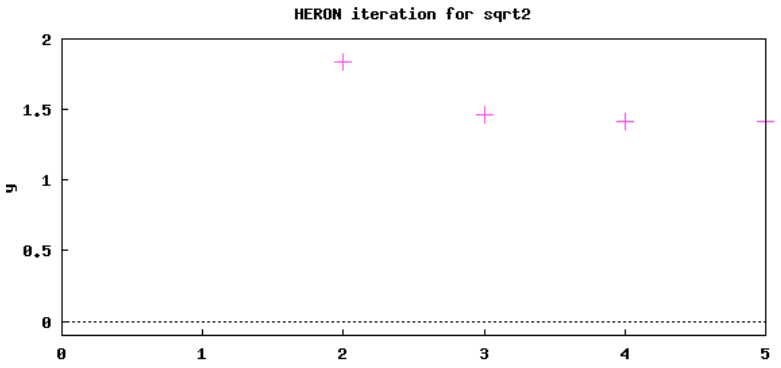
```
(%i16)  fpprintprec:5$
        Sqrt2 : iterate2([n+1, 0.5*(x+2/x)], [n,x], [1,3],4);
(Sqrt2) [[1,3],[2,1.8333],[3,1.4621],[4,1.415],[5,1.4142]]
```

Code line (7) allows to give the output of the list `Sqrt2` in a rough table shape by `transpose`ing the matrix `Sqrt2`:

```
(%i17)  transpose( iterate2([n+1, 0.5*(x+2/x)], [n,x], [1,3],4) );
                 [ [1,3]      ]
                 [ [2,1.8333] ]
(%o17)           [ [3,1.4621] ]
                 [ [4,1.415]  ]
                 [ [5,1.4142] ]
```

Code line (7) also allows to give the output of the original list `Sqrt2` as a graphical point list:

```
/* wxMaxima */
wxdraw2d(xaxis = true,
    user_preamble="set size ratio -1", /* axis equal scaled */
    xrange = [0,5], yrange = [-0.1,2],
    xlabel ="x",ylabel ="y",
    point_size=2, points_joined=false,
    color = magenta,
    points( Sqrt2 ),
  title="HERON iteration for Sqrt2");
```




▷ Mark-Copy-Paste and RUN the code lines.

**Exercise 69.** Write a function `fib(n):= iterate2(..., [0,1], n)` to compute the $n$-th Fibonocci number of an integer $n$.

## 3.3 First Applications of `iterate`

To convince us of the power of `iterate/2` we demonstrate its use for the approximation of the root of a function with the help of the so called NEWTON-RAPHSON method and the Secant method. We also introduce an alternative for the PICARD iterates: the TAYLOR method for the approximate solution of an ODE.

### 3.3.1 simplified NEWTON-RAPHSON method

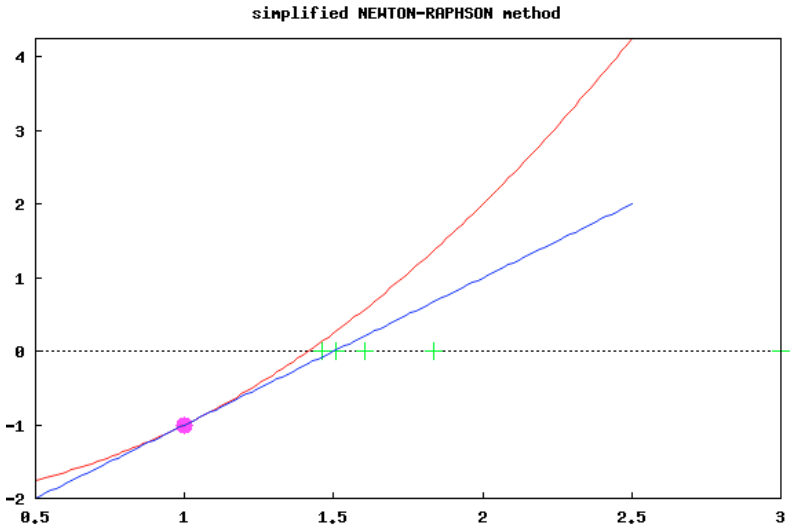



Figure 11: *Red:* The graph of $f(x) = x^2 - 2$. Where is the root of $f$?
*Blue:* The tangent on Graph($f$) at point $P = (1, -1)$.
Graphically the root $\xi$ of $f(x) = 0$ is located where the graph of $f$ crosses the $x$-axis, i.e. $\xi \approx 1.4$.
*Idea* of NEWTON-RAPHSON method: If we know - e.g. by looking at the graph of $f$ on $[a, b]$ - that a root $\xi$ of $f$ exists between $a$ and $b$, then we take the intersection $x_1$ of the tangent of $f$ going through point $(a, f(a))$ with the $x$-axis as the initial guess for $\xi \approx x_1$.
*Green:* The sequence of approximate solutions towards the root $\xi$ using the *simplified* NEWTON *method* for $x^2 - 2 = 0$ tending to $\sqrt{2} \approx 1.41$.

• How to get a recurrence formula for this idea?

The equation of the tangent line at point $(x_0, f(x_0))$ is $y = f(x_0) + f'(x_0)(x - x_0)$. Crossing the x-axis means $y = 0$, so we get $0 = f(x_0) + f'(x_0)(x - x_0)$ and solving for $x$ and setting $c := f'(x_0)$, we get using MAXIMA

```
solve(0 = f(x_0) + c*(x-x_0), x), expand;
```

$$(\%o14) \quad \left[x = x_0 - \frac{f(x_0)}{c}\right]$$

for the next guess $x_1 := x_0 - f(x_0)/c$. So we arrive at the recurrence formula

$$x_{n+1} = x_n - \frac{f(x_n)}{c} \tag{3.1}$$

with $c := f'(x_0)$. We implement this formula using iterate:[10]

```
/* MAXIMA --- NEWTON-RAPHSON iteration -- */
/*           include our function iterate(..) here */

newton1(u,x,a,n) := block([c: float(at(diff(u,x),x=a))],       /* (1) */
                          iterate( x-u/c, x,a, n)     );      /* (2) */
```

**Comment.** In (1) we translate $c := f'(x_0)$ into MAXIMA language. (2) takes the recurrence as RHS of (3.1) with an arbitrary term $u$ instead of $f(x)$. Updating $x_n \rightsquigarrow x_{n+1}$ is done automatically by `iterates`.

**Example 12.** (3) and (4) demonstrate two invocations of `newton1`.

```
newton1( x^2-2 ,x,1, 9);        /* (3) */

f(x):= x^2-cos(x);
newton1(f(x),x,1,5);            /* (4) */
```

▷ Mark-Copy-Paste and RUN the code lines.

Output:

```
(%o11)  [1,1.5,1.375,1.4297,1.4077,1.4169,1.4131,1.4147,1.414,1.4143]
(%o12)  f(x):=x^2-cos(x)
(%o13)  [1,0.83822,0.82632,0.82448,0.82419,0.82414]
```

**Exercise 70.** The recurrence formula (3.1) is called the *simplified* NEWTON-RAPHSON method, because it uses the same constant slope $c := f'(x_0)$ of the first tangent at the touching point unaltered in every step of the iteration. If the slope $f'(x)$ is actualized in each step, the recurrence formula is changed to the *original* NEWTON-RAPHSON method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{3.2}$$

Then we get the MAXIMA function

```
newton(u,x,a,n) := iterate( x - u/diff(u,x), x,a, n);
```

**a.** Run `newton(x*x-2,x,1,5)` and `newton(x*x-cos(x)),x,1,5)` using floating point precision `fpprintprec:5`. Compare the results with (3) and (4).
**b.** Look at ▷ WIKI: NEWTON method. and watch the animation of the changing tangents.

[10] Please observe: using `iterate` it is not necessary to use indices as in (3.1) – the RHS is sufficient!

**Exercise 71.** Reproduce Fig.11 with this code:

```
/* wxMaxima */
f(x):= x*x-2;  a: 1;  b: 2;
m: at(diff(f(x),x),x=a);
X: makelist(0,i,1,6);
Y: newton1(f(x),x,3,4);  /* you must run newton1 before its use ! */
XY: makelist([Y[i],X[i]], i,1,5);
wxdraw2d(xaxis = true,
    point_size = 2, color = green, points(XY),
    point_size = 2, point_type = filled_circle,
    color =magenta, points([[a,f(a)]]),
    color =red,     explicit( f(x), x,0.5,2.5),
    color =blue,    explicit( m*(x-a)+f(a), x,0.5,2.5),
    title="simplified NEWTON-RAPHSON method")$
```

Think about every line of the code and its purpose.

### 3.3.2 simplified Secant method

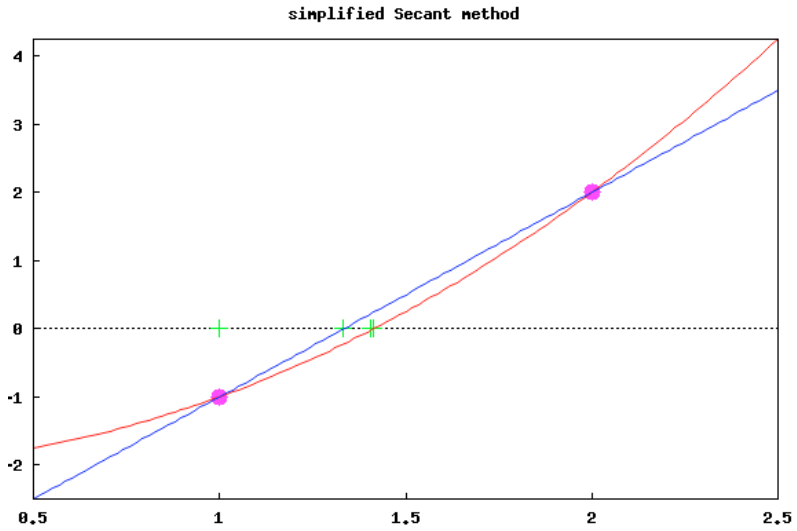



Figure 12: *Red:* The graph of $f(x) = x^2 - 2$. Where is the root of $f$? Graphically the root $\xi$ of $f(x) = 0$ is located where the graph of $f$ crosses the $x$-axis, i.e. at $x \approx 1.4$.
*Idea* of Secant method: If we know - eg by looking at the graph of $f$ on $[a, b]$ - that a zero $\xi$ of $f$ exists between $a$ and $b$, then we take the intersection $x_1$ of the secant (blue line) of $f$ through the two points $(a, f(a))$ and $(b, f(b))$ as a first approximate value for $\xi \approx x_1$.
*Green:* The sequence of approximate solutions for the root.

• How to get a recurrence formula for this idea?

If we consider the tangent in §3.3.1 approximately replaced by the secant, we should replace the constant slope $c := f'(a)$ in the recurrence formula (3.1) with the constant slope $c := \frac{f(b)-f(a)}{b-a}$ of the secant. This gives us the new simplified recurrence formula

$$x_{n+1} = x_n - \frac{f(x_n)}{c} = x_n - C \cdot f(x_n) \tag{3.3}$$

with $C := \frac{b-a}{f(b)-f(a)}$. We implement this formula using `iterate`:

```
/* MAXIMA --- SECANT METHOD   --- */
/*           include function iterate(..) here */

secant1(u,x, a,b, n) := block(
                          [C: (a-b)/(subst(a,x,u)-subst(b,x,u))],  /* (1) */
                          iterate( x - C*u, x,a,n)) ;              /* (2) */
```

**Comment.** In (1) we translate $u(a)$ into MAXIMA language as `subst(a,x,u)`.
(2) takes the recurrence as RHS of (3.3) with an arbitrary term $u$ instead of $f(x)$.

**Example 13.** Let's do the two tests from above:

```
secant1(x^2-2,x, 1,2, 5);
```

Output:

```
(%o7)   [1,1.3333,1.4074,1.4138,1.4142,1.4142]
```

```
f(x):= x^2-cos(x);
secant1(f(x),x, 1,2, 5), numer;
```

Output:

```
(%o28)  f(x):=x^2-cos(x)
(%o29)  [1,0.88381,0.84668,0.83293,0.82761,0.82551]
```

▷ Mark-Copy-Paste and RUN the code lines.

**Exercise 72.** (sign change test) In Fig.12 the function values of $f$ at points $(a, f(a))$ and $(b, f(b))$ have different signs, so we can argue that (our continuous) function $f$ must cross the $x$-axis and therefore must have a root $\xi$ between $a$ and $b$.
Here is function `signChange`, that checks, if there is a sign change on the interval $[a, b]$:

```
signChange(a,b) := if  f(a)*f(b) < 0  then true else false;
```

Test function `signChange` by running

```
f(x):= x*x-2;
signChange(-2,2);
signChange(0,2);
```

**Exercise 73.** (the general secant method) The recurrence formula (3.3) is called the *simplified Secant* method, because it uses the same constant slope $C$ of the first secant unaltered in every step of the iteration. If the slope $C$ is actualized in each step, we get the *original Secant* method. We will derive it in this exercise. We follow [17, p.39, p.56 ff].

**a.** Verify, that the straight line (secant) through the points $(a, f(a))$ and $(b, f(b))$ is given by

$$y = \frac{f(b) - f(a)}{b - a} \cdot x + \frac{b \cdot f(b) - a \cdot f(a)}{b - a} \tag{3.4}$$

**b.** The straight line **a.** crosses the $x$-axis at $y = 0$, so we have $0 = \frac{f(b)-f(a)}{b-a} \cdot x + \frac{b \cdot f(b)-a \cdot f(a)}{b-a}$. Use MAXIMA to show: $x = \frac{a \cdot f(b)-b \cdot f(a)}{b-a}$.

**c.** Implement MAXIMA function `iterate3(u,x,xo,n) := block( [l:[[xo[1],xo[2],xo[3]]], ...)`, which operates on lists of 3 elements.

**d.** Then define for a global defined user function $f(x)$ :

```
iterate3(u,x,xo,n) := block( [l:[[xo[1],xo[2],xo[3]]], ... );  /* by you */
slope(x,y):= (x*f(y)-y*f(x))/(f(y)-f(x));
secants(a,b,n) := iterate3([x,y,slope(y,z)],[x,y,z],[a,b,slope(a,b)],n) ;
```

**d.** Test your functions with

```
f(x):= x^2-2;
transpose(secants(0,2, 5));
```

The output should be:

```
(%o50)  [ [0,2,1]      ]
        [ [0,2,1.3333] ]
        [ [0,2,1.4]    ]
        [ [0,2,1.4118] ]
        [ [0,2,1.4138] ]
        [ [0,2,1.4141] ]
```

**e.** Functions `slope` and `secants` work for a global defined function $f$. Alter both functions to work on an universal local function $u$ a la `slope(u,x,y)` resp. `secants(u,a,b,n)`, so that a call `secants(x*x-2, 0,2, 5)` is possible.

**Remark.** The theme of determine the roots of a function is part of general Numerical Analysis/Methods. For our purpose to solve boundary value problems (BVP) of an ODE our function `secant1` will be stable enough to serve as a possible helper function for the Shooting method.

**Exercise 74.** Reproduce Fig.12 with this code:

```
/* wxMaxima */
f(x):= x^2-2;
a:1; b:2;
```

```
m:  (f(a)-f(b))/(a-b), numer;
X:  makelist(0,i,1,5)$
Y:  secant1(f(x),x, 1,2, 5)$
XY: makelist([Y[i],X[i]], i,1,5);
wxdraw2d(xaxis = true,
    point_size=2,
    color=green, points(XY),
    point_size = 2,
    point_type = filled_circle,
    color=magenta, points([[a,f(a)],[b,f(b)]]),
    color=red,     explicit(f(x),x,0.5,2.5),
    color=blue,    explicit(m*(x-a)+f(a),x,0.5,2.5),
    title="simplified Secant method")$
```

Think about every line of the code and its purpose.

## 3.4 Implict differentation and the TAYLOR method for ODE's

To show the usefulness of the function `iterate`, we have shown some examples that are not directly related to ODE's. We now return to dealing with the solution of an ODE. For this we implement the TAYLOR method as an alternative to the PICARD iteration. We start by implementing a helper function `idiff`, which shows another application of the Swiss army knife `iterate`.
We need two requisites: the 2-dimensional Chain rule and the TAYLOR expansion.

### 3.4.1 implicit differentiation.

Let's remember the advanced calculus course. Quoting MARSDEN [33, p.780] we have

**Theorem (2-dimensional Chain Rule; implicit differentiation)**
*IF $f(x,y)$ (e.g. the RHS of an ODE) has continuous partial derivatives and the utility $U(x) := f(g(x),h(x))$ is a function of $x$ alone with $g(x)$ and $h(x)$ differentiable, THEN with $f_x := \frac{\partial f}{\partial x}$*

$$U'(x) = f_x(g(x),h(x)) \cdot g'(x) + f_y(g(x),h(x)) \cdot h'(x) \tag{3.5}$$

i.e. to get $U'(x)$ you have to multiply the partial derivatives of $f(x,y)$ w.r.t each variable $x$ and $y$ with the derivatives of $g$ and $h$ and add the products.

- You may prefer the other notation of (3.5): $\frac{\partial U}{\partial x} = \frac{\partial f}{\partial x} \cdot \frac{dg}{dx} + \frac{\partial f}{\partial y} \cdot \frac{dh}{dx}$.
- Because of the dependence of $U$ on two interior intermediate functions $g$ aand $h$ this differentiation process is called *implicit differentiation with respect to $x$*.

**Example 14.** Consider the ODE $y' = f(x,y) := xy$, i.e. more precise: $y'(x) = f(x,y(x)) \equiv f(g(x),h(x))$ with $g(x) := x$ and $h(x) := y(x)$.
Calculate the first three derivatives of $U(x) := f(x,y(x))$.

*Solution.* $U$ depends only on $x$. We want to use equation (3.5).
We have $g'(x) = 1$ and $h'(x) = y'(x) \overset{ODE}{=} xy$. Therefore

$$\begin{aligned} y''(x) = U'(x) &= f_x(g(x), h(x)) \cdot g'(x) + f_y(g(x), h(x)) \cdot h'(x) \\ &= (xy)_x \cdot x' + (xy)_y \cdot y'(x) = y \cdot 1 + x \cdot y'(x) \overset{y'=xy}{=} y + x \cdot xy \\ &= y(x) + x^2 \cdot y(x) \end{aligned}$$

Again $U'$ depends only on $x$. So we have a new function $U_2(x) := U'(x) = y(x) + x^2 \cdot y(x)$ and we can repeat the implicit differentiation via the chain rule (3.5). Therefore

$$\begin{aligned} y'''(x) = U_2'(x) &= U''(x) = \frac{\partial}{\partial x}(y + x^2 y) \cdot x' + \frac{\partial}{\partial y}(y + x^2 y) \cdot y' \\ &\overset{y'=xy}{=} (2xy) \cdot 1 + (1 + x^2) \cdot xy = 2xy + xy + x^3 y \\ &= 3xy + x^2 \cdot y \\ &=: U_3(x) \end{aligned}$$

We see an emerging pattern for the sequence of new functions $U_n$:

$$U_n'(x) = \frac{\partial}{\partial x} U_{n-1} \cdot x' + \frac{\partial}{\partial y} U_{n-1} \cdot f(x, y) \tag{3.6}$$

**Exercise 75.** Determine $y''''(x)$ for the ODE $y' = xy$.

### 3.4.2 idiff.

Exercise 75 shows that it becomes annoying and uncomfortable to iterate the implicit differentiation of $f(x, y)$ to get more derivatives of $y'(x)$. So let our function `iterate` do this work for us by defining a function `idiff` ('implicit' diff)[11]. We use the recursion (3.6) for the recurrence term in `idiff`, cf. [17, p.206 ff]:

```
/* MAXIMA  --- IMPLICIT DIFFERENTATION --- */
idiff(f,x,y,m):=block(
                IMP1: iterate('diff(u,x) + f*'diff(u,y), u,f,m-1), /*(1)*/
                factor(ev(IMP1, diff)) )$                          /*(2)*/
```

The invoke arguments of `idiff(f,x,y,m)` are as follows:

1. `f`: the function $f(x, y)$ ( i.e. the right-hand side of an ODE $y' = f(x, y)$) ...
2. `x,y` : ... depending on the variables $x$ and $y$.
3. `m`: the number of repeated substitutions of new functions $u$ according to (3.6).

[11] In 1999 Dan Stanger wrote an implicit differentiation function in Maxima. The files are in share/contrib/`impdiff.mac` and `makeOrders.mac`. In contrast to our `idiff` it is a very professional code.

*Comment.* In (1) the recurrence term `diff(u,x,1)+f*diff(u,y,1)` is $\frac{\partial u}{\partial x} \cdot 1 + \frac{\partial u}{\partial y} \cdot f(x,y)$, i.e. the translation of (3.6) to MAXIMA. The iteration variable here is $u$, i.e. the each newly created function $u$ in the process (3.6)! That is why the initial value for $u$ in the process is the *function* $f(x,y)$. In (1) the build-in function `diff` is not evaluated because of the quote `'`; therefore we have explicitly to `ev`(aluate) the `diff` function in the `IMP1` term in (2).[12]

**Example 15.** Check example 14 and exercise 75 with `idiff`.

*Solution.*

```
/* Maxima */
iterate(u,x,xo,n) := block( [l:[xo], numer:true, val:xo],
     for i thru n do (val: subst(val,x,u), l: cons(val,l)), reverse(l))$
idiff(f,x,y,m):=( IMP1: iterate('diff(u,x) + f*'diff(u,y), u, f, m-1),
                  factor(ev(IMP1, diff)))$

expand( idiff(x*y,x,y,4) );
```

▷ Click to RUN the code.

The output displays all implicit derivatives starting with $f$ until $y'''$, i.e. the list $[y, y', y'', y''']$:

```
(%o6)   [x y, x^2 y+y, x^3 y+3 x y, x^4 y+6 x^2 y+3 y]
```

In case you only want to look at the result of exercise 75, do

```
(%i7)   expand( idiff(x*y,x,y,4)[4] );
(%o7)   x^4 y+6 x^2 y+3 y
```

### 3.4.3 TAYLOR method for ODEs.

Let $y' = f(x,y),\ y(x_0) = x_0$ be an ODE. If we assume the solution $y(x)$ to be an analytic function, we may approximate its values near $x$ by a TAYLOR polynomial of degree $m$, i.e.:

$$y(x+h) \approx y(x) + \frac{y'(x)}{1!} \cdot h + \frac{y''(x)}{2!} \cdot h^2 + \frac{y'''(x)}{3!} \cdot h^3 + \cdots + + \frac{y^{(m)}(x)}{m!} \cdot h^m \qquad (3.7)$$

Since we want to get a recurrence formula for $y_{k+1} \approx y(x_{k+1}) = y(x_k + h)$, we define

$$y_{k+1} := y_k + \frac{y'(x_k)}{1!} \cdot h + \frac{y''(x_k)}{2!} \cdot h^2 + \frac{y'''(x_k)}{3!} \cdot h^3 + \cdots + + \frac{y^m(x_k)}{m!} \cdot h^m \qquad (3.8)$$

In (3.8) we need the derivatives $y'(x_k), y''(x_k), y'''(x_k), \ldots$ *But these derivatives are interpreted and therefore determined as implicit derivatives of* $f(x,y)$, which we can easily calculate via our self-defined function `idiff`: so equation (3.8) gives the recurrence term of the 'implicit' Taylor method of oder $m$. Please observe: In difference to the 'normal' Taylor expansion we have to use `idiff` instead of `diff`.

[12] I thank Michel `Talon` who friendly helped in coding (2).

We may also look at equation (3.8) as a scalar product of the two lists (ok: vectors)

$$y_{k+1} = [y, y', y'', y''', ..] \bullet [h, \frac{h^2}{2}, \frac{h^3}{6}, \frac{h^4}{24}, ...] \tag{3.9}$$

This motivates the function `iTaylor`, cf. [17, p.208]:

```
/* MAXIMA --- IMPLICIT TAYLOR METHOD of order m --- */
/* include our functions iterate(u,x,xo,n) and idiff(f,x,y,m) here */

iTaylor(f,x,y, h, m) := y + idiff(f,x,y,m).makelist(h^r/r!, r,1,m);
```

**Example 16.** We test `iTaylor` on the IVP $y' = xy$ with $y(0) = 1$.

```
makelist(h^r/r!, r,1,4);
iTaylor(x*y, x,y, h,3);
iTaylor(x*y, x,y, 1/5, 3),expand;
```

The output displays all `Taylor` method order 3 terms with its implicit derivatives

```
(%i9)  iTaylor(x*y,x,y,h,3);
(%o9)  h^3 x (x^2+3) y / 6 + h^2 (x^2+1) y / 2 + h x y + y
```

```
(%i10) iTaylor(x*y, x,y, 1/5, 3),expand;
(%o10) x^3 y/750 + x^2 y/50 + 51 x y/250 + 51 y/50
```

For the IVP the command `at(%, [x=0, y=1]),float;` gives `(%o12) 1.02`.

**Exercise 76.** Look at the following MAXIMA script:

```
iterate2(u,x,xo,n) := block(
         [ l: [[xo[1],xo[2]]], numer:true, val:[xo[1],xo[2]] ],
         for i thru n do
            (val: subst([x[1]=val[1],x[2]=val[2]],u), l: cons(val,l)),
         reverse(l))$

fpprintprec: 5$
iterate2([x+0.2, iTaylor(x*y, x,y, 0.2, 3)], [x,y], [0,1], 5);   /*(1)*/
```

▷ Click to RUN the code.
The output of (1) is

```
(%o7) [[0,1],[0.2,1.02],[0.4,1.0828],[0.6,1.1964],[0.8,1.3757],[1.0,1.6463]].
```

Explain: what does command (1) do?

**Exercise 77.** **a.** Use a TAYLOR method of order 2 to approximate the solution of the IVP $y' = \frac{3x^2}{2y}$ with $y(0) = 1$ for $0 \leq x \leq 2$ using 4 steps of length 0.5, see [43, p.237]. – Code:

```
fpprintprec:5; ratprint : false
iterate2([x+0.5, iTaylor(3*x*x/(2*y),x,y, 0.5,2)], [x,y], [0,1], 4);
```

Result: `(%o8) [[0,1],[0.5,1],[1.0,1.3574],[1.5,2.0738],[2.0,2.9991]]`

**b.** Verify: the exact solution of the IVP is $y(x) = \sqrt{1 + x^2}$.

**c.** Calculate the 'global absolute error', i.e the summed distances of all $y$-approximate values from their corresponding exact values of the solution function $y(x)$. [Result: 0.0606

**d.** Plot the exact solution $y$ and the points of the approximate solution values $y_k$ to visualize the global absolute error.

**e.** Do part **a.** by hand ..

### 3.4.4 `iTaylorsol` (TAYLOR solution method for ODE)

Example 16 shows a TAYLOR method of order 3 to approximate the solution of the IVP $y' = xy$ with $y(0) = 1$ for $0 \leq x \leq 1$ using 5 steps of length 0.2. We will now abstract this procedure to a function `iTaylorsol`, which will iterate the TAYLOR recurrence relation for given data to establish a full automatic TAYLOR *solution method* of a prescribed order[13]:

```
/* MAXIMA ---  TAYLOR SOLUTION METHOD of ODE y'=f(x,y) --- */
/* include functions iterate2(u,x,xo,n), idiff(f,x,y,m)
   and     iTaylor(f,x,y,h,m) here */

iTaylorsol(f,x,y, xo,yo, h, m, n) :=
                  iterate2([x+h, iTaylor(f,x,y, h, m)], [x,y], [xo,yo], n);
```

The set of invoke arguments of `iTaylorsol(f, x,y, xo,yo, h, m, n)` is as follows:

1. `f`: the right-hand side of the ODE $y' = f(x, y))$
2. `x,y`: the names for the variables $x$ and $y$ of the function $f$
3. `xo,yo`: the IVP initial values for $x$ and $y$, i.e. $y(x_o) = y_o$
4. `h`: the positive constant step size
5. `m`: the order of the method, i.e. the highest exponent of the Taylor expansion
6. `n`: the number of iterations, i.e. the number of iterated invocations of `iTaylor`

**Example 17.** Let's check `iTaylorsol` for the IVP $\{ \begin{smallmatrix} y'=xy \\ y'(0)=1 \end{smallmatrix}$:

```
iTaylorsol(x*y, x,y, 0,1, 0.2, 3, 5) ;
```

MAXIMA output:

```
(%o8) [[0,1],[0.2,1.02],[0.4,1.0828],[0.6,1.1964],[0.8,1.3757],[1.0,1.6463]]
```

[13] cf. [17, p.208 ff]

**Exercises.**

**Exercise 78.** Someone wrote this piece of MAXIMA code.:

```
Taylor3(x,y, xo,yo, h,n):=
        iterate2([x+h, y+h*x*y + h^2/2*(x^2*y+y) + h^3/6*(x^3*y+3*x*y)],
                 [x,y],[xo,yo],n)$

Taylor3(x,y, 0,1, 0.2, 5);
```

▷ Click to RUN the code.
**a.** What does `Taylor3` do? What looks a corresponding call to `iTaylorsol` like?
**b.** Write `Taylor2` and `Taylor4`.
**c.** Discuss pros and cons of `Taylor3` w.r.t. `iTaylorsol`.

**Exercise 79.** Use a Taylor method of order 3 to solve the IVP $y' = x - y^2,\ y(0) = -0.5$ . Compare with exercise 4.

**Exercise 80.** Construct the solution $y$ of the IVP $y' = x^2 + y^2,\ y(0) = 0$ via a Taylor method of order 2. Compare with exercise 5.

**Exercise 81.** Given the IVP $dy/dx = x - y,\ y = 1\ at\ x = 0$, use a Taylor method of order 4 to approximate $y$ when $x = 0.2$, cf. [3, p. 189].
[Control: $y_5(0.2) \approx 0.83746$. - Exact solution: $y = x - 1 + 2e^{-x}$.

**Exercise 82.** Calculate the solution $y$ of the ODE $y' = y^2 - xy$, which has value $y = 1$ for $x = 0$, via a Taylor method of order 3, cf. [24, p.79, p.308]. [Control: $y_5(0.5) \approx 1.6987$.

**Exercise 83.** Find the approximate solution to the equation $y'(t) = 1 + y(t)^2$ with initial condition $y(t_0) = y_0 = 0, t_0 = 0$ via a Taylor method of order 4. Compare with exercise 8.
[Result: $y(x) = \tan(x)$

**Exercise 84.** An ODE is given through $x' = \sin(t) - x$ with IC $x(0) = 1$.
Do a Taylor method of order 3 to approximate the solution $x$. Compare with exercise 9.

**Exercise 85.** Find an approximate solution to the initial value problem
$y' = 2t(y + 1),\ y(0) = 0$ using a Taylor method.
Compare with the exact solution $y(t) = e^{t^2} - 1$, cf. [11, p.110].

**Exercise 86.** Discuss: what are the similarities and differences between `iTaylor(.)` and `iTaylorsol(.)`.

# 4 Numerical Solution Methods for IVP

We discussed several methods of approximating solutions to an initial value problem (IVP). We may think of the solution $y(x)$ to an IVP as describing a physical system that changes in time, for example the displacement of a swinging pendulum from its rest position. The initial conditions $y(x_o) = a$ and $y'(x_o) = b$ correspond to specifying the displacement and the velocity of the Pendulum at time $t = x_o$. We set the Pendulum in motion and watch it for some interval of time $x_o$ to $x_e$ by means of the ODE. – Read ▷ Ammari: Examples.

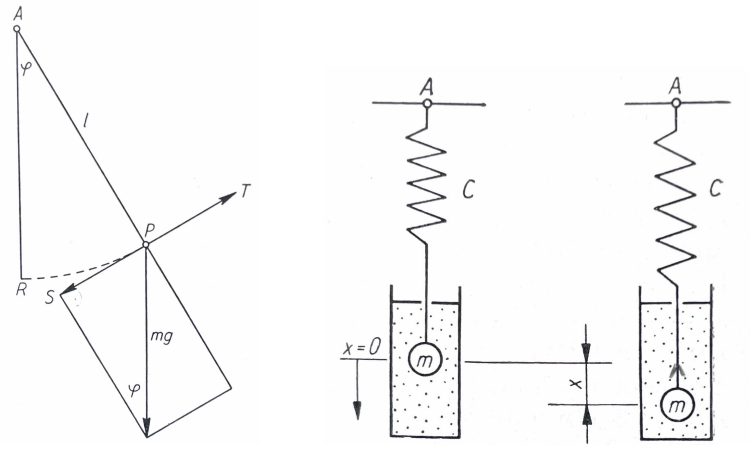



Figure 13: Bräuning [12, p.11] motivates the dealing with differential equations through their use in the natural sciences and presents examples like:
*Left:*(*pendulum*) "A small body of mass $m$ is attached to point $A$ with a thread of length $\ell$. Let's deflect the body out of its resting position, keeping the thread taut and then leaving the body to itself, it will oscillate around this resting position. We describe this swing of a pendulum by means of this ODE: $-\mathbf{m}\ell\ddot{\varphi} - \mathbf{mg}\sin\varphi = \mathbf{0}$."
*Right:* (*swinging spring*) "An elastic spring is attached to a suspension point $A$, to which a mesh $m$ is attached at the other end, which is located in a liquid for damping. $c$ is the spring constant. A coordinate system is directed in its x-direction downward so, that its origin is the rest position of the center of gravity. If the mass is deflected from its rest position in the x-direction and then released, it performs vertical oscillations around the rest position.
The associated differential equation for describing this movement is $-m\ddot{x} - k\dot{x} - cx = 0$, where $k \in \mathbb{R}$ a constant w.r.t damping."

We now study numerical ('approximative') solution methods for ODE's and make extensively use of our user-defined functions `iterate/2/3`, which allows to program compact code for methods like `EULER`, `HEUN` or `RK4` in 2-4 lines, to focus on the underlying recurrence formulas and relieve of programming technical bureaucracy. Therefore this chapter can profit from some ideas of the books [17, p.175 ff], [29, p.101 ff] and [43, p.228 ff]. Error analysis for the methods are left to the literature, e.g. [29], [1], [47], [20].

## 4.1 RK1 alias the EULER Method

We start by looking at the idea of the EULER method for approximately solving ODE's.

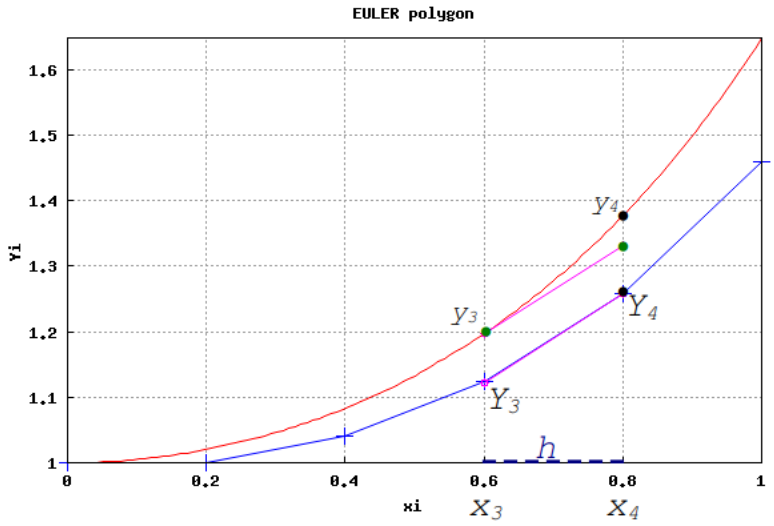



Figure 14: *Red:* $y(x) = exp(\frac{x^2}{2})$ is the exact solution of ODE $y' = x \cdot y,\ y(0) = 1$.
*Blue:* We try to follow the graph of $y$ approximately by a polygon.
*Idea* of EULER method: If we are at a point $(x_3, Y_3)$ on the approximating polygon near the exact point $(x_3, y_3)$ on the solution, then we follow the tangent at $(x_3, y_3)$ with slope $y'(x_3) = f(x_3, y_3) = x_3 \cdot y_3$ a bit - but starting at $(x_3, Y_3)$ on the polygon. This way we arrive at the next approximation point $(x_4, Y_4) \approx (x_4, y_4) \in graph(y)$ .
This process is then repeated starting at $(x4, Y4)$.

- How to get a recurrence formula for this idea? I.e. how to calculate $Y_4$? $Y_n$?

We start with the observation, that the slope of the tangent at the point $(x_3, y_3)$ on the graph of the solution $y(x)$ is the same as the slope of the approximating polygon line between $x_3$ and $x_4$:

$$
\begin{aligned}
y'(x_3) = f(x_3, y_3) \quad &\approx \quad f(x_3, Y_3) = \frac{Y_4 - y_3}{x_4 - x_3} = \frac{Y_4 - y_3}{h} && (4.1)\\
&\rightsquigarrow \quad f(x_3, Y_3) \approx (Y_4 - y_3)/h \\
&\rightsquigarrow \quad y_3 + f(x_3, Y_3) \cdot h \approx Y_4. \\
&\overset{y_3 \approx Y_3}{\rightsquigarrow} \quad Y_4 = Y_3 + h * f(x_3, Y_3) && (4.2)\\
&\overset{\text{by analogy}}{\rightsquigarrow} \quad Y_{n+1} = Y_n + h * f(x_n, Y_n) && (4.3)
\end{aligned}
$$

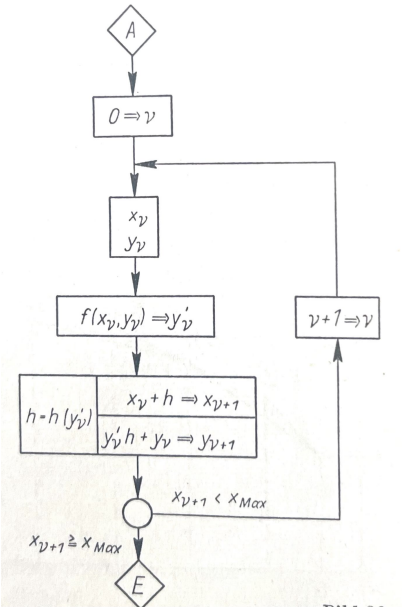
Formula (4.3) is called the EULER *recurrence formula*.

**Remark.**

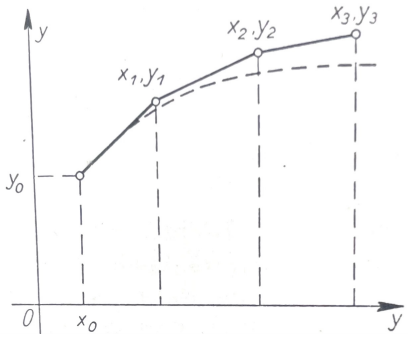
1. In equation (4.3) we denote the approximative values on the polygon with uppercase letters $Y..$ and the exact values of the solution function $y(x)$ with lowercase letters $x.., y..$ in order to distinguish between them.

2. For more information about the EULER method and derivations of (4.3) see e.g. ▷ Wiki: EULER method - Derivation or ▷ Cheever: EULER method, [47, p.4], [20, p.1], [1, p.44].

3. To organize the steps of the calculation, program flow charts were used in the past, especially when using programmable pocket calculators, cf. BRÄUNING [12, p.145]:




flow chart




4. The EULER method is an algorithm with *only one evaluation* of the function $f(.,.)$ in (4.3). We will see that the other numeric algorithms need more $f$-evaluations.

**Example 18.** We calculate the data for Fig.14 by studying the ODE $y' = f(x,y) = x \cdot y$, $y(0) = 1$. We know: the exact solution $y_e$ of this ODE is $y_e(x) = \exp(\frac{x^2}{2})$, because the derivative of $y_e$ is again $y_e$ with a factor $x$ in front, i.e the equation $y'_e = x \cdot y_e$ is fulfilled and verified by MAXIMA:

```
ye: exp(x^2/2);      /* exact solution  ye */
yp: diff(ye, x);     /* their slope ye' */
```

▷ Click to RUN the code.

```
(ye)   %e^(x^2/2)
(yp)   x %e^(x^2/2)
```

First we do a semi-automatically calculation by hand and then a fully automatic iteration.

**1.** (EULER method by hand) We use formula (4.3) with $f(x,y) := x \cdot y$ and a step-size $h := x_4 - x_3 = 0.2$. We do 5 steps from $x_0 = 0$ until $x_5 = 1$.

If we have e.g. $x_3 = 0.6 = 3 \cdot h$ and $y_3 = 1.1232$, it follows $y_4 = y_3 + h \cdot f(x_3, y_3) = 1.1232 + 0.2 \cdot 0.6 \cdot 1.1232 = 1.257984$.
To make such steps semi-automatic, we first define and invoke a helper function $E(x, y)$, starting with $x = 0$ and $y = 1$, catch the corresponding result by eye and use it as input for the next call $E(0.2, 1)$ and so on. So we step through the points of the polygon in Fig.13:

```
E(x,y) := [x + 0.2, y +0.2* x*y];
       /* xn + h    Yn+  h*f(xn,Yn)  */
ratprint:false$
fpprintprec:5$
E(0.0, 1);
E(0.2, 1);
E(0.4, 1.04);
E(0.6, 1.1232);
E(0.8, 1.257984);
```

▷ Click to RUN the code.

```
(%o7)   E(x,y):=[x+0.2,y+0.2 x y]
(%o10)  [0.2,1.0]
(%o11)  [0.4,1.04]
(%o12)  [0.6,1.1232]
(%o13)  [0.8,1.258]
(%o14)  [1.0,1.4593]
```

We may now reproduce the plot of Fig.13:

```
XY: [E(0,1),E(0.2,1),E(0.4,1.04),E(0.6,1.1232),E(0.8,1.257984)];
wxdraw2d(xaxis = true,
    point_size=2, points_joined=true,
    xlabel="xi", ylabel="Yi",
    points(XY),
    color=red,
    explicit(exp(x^2/2),x,0,1),  /* exact solution ye */
    title="EULER polygon")$
```

▷ Click to RUN the code.

**2.** (EULER method by iterate) We use formula (4.3) with $f(x, y) := x \cdot y$ and step through the points of the polygon in Fig.13 automatically via **`iterate2`** (because we construct points with 2 coordinates), where the helper function $E(x, y)$ is now incorporated as the recurrence term $[x + h, y + h * f(x, y)]$:[14]

```
f(x,y):= x*y;
xo: 0;
yo: 1;
h: 0.2;
```

[14] ♡ Don't forget to call `iterate2` beforehand !

```
n: 5;   /* recurrence__term    vars.   initial  steps */
iterate2( [x+h,  y+h*f(x,y)],  [x,y],  [xo,yo], n     );
```

▷ Click to RUN the code. The results are the same as above.

### 4.1.1 The EULER Method for IVP

We now abstract the whole approximation process by constructing an MAXIMA function EULER to fully-automate the calculation process. We check the correctness by repeating the handish calculation in 1. in (1).

```
/* MAXIMA : --- EULER method  --- */
EULER(x,y, xo,yo, h, n) := iterate2([x+h, y+h*f(x,y)],[x,y],[xo,yo],n);

/* Test ODE  y'= xy with y(0)=1 */
fpprintprec: 5$
f(x,y):=x*y;
EULER( x,y, 0, 1, 0.2, 5);      /* (1) */
```

▷ Click to RUN the code. The results are the same as above. ♡ Don't forget to call `iterate2` !

- The invoke arguments of `EULER(x,y, xo,yo, h, n)` are as follows:
  1. `f` : the RHS of the ODE $y'(x) = f(x, y(x)))$ given global before the call
  2. `x,y` : the variables used in the iteration (coordinates) i.e. by $f$
  3. `xo,yo` : the initial values with $y(x_o) = y_o$
  4. `h` : the step size of the EULER method
  5. `n` : the number of repeated iterations

*Comment.* We elaborate on the code of EULER. The recurrence formula $[x+h, y+h*f(x,y)]$ in EULER updates permanently both the $x$ value and the $y$ value of $[x, y]$, starting with $[x, y] := [x_o, y_o]$. To the $x$ value of $[x, y]$ is added the stepsize $h$ at each step. The corresponding $y$ slot of $[x, y]$ is replaced by $y + h * f(x, y)$. This updating procedure runs $n$ times.

**Remark.** If we want to emphasize that only 1 function evaluation $k_1$ is required in each step of the iteration process, we may write our function EULER as follows and then call this variant the RUNGE–KUTTA *method of 1st kind*, denoted RK1:

```
/* MAXIMA : --- EULER method denoted as RK1  --- * /
RK1(x,y, xo,yo, h, n) := block(
    k1: f(x,y),
    iterate2([x+h, y+h*k1],[x,y],[xo,yo],n);
```

**Exercises.**

**Exercise 87.** Use Euler's method with step size $h = 0.1$ to find an approximate solution of $\frac{dy}{dx} = x - y^2,\ y(0) = -0.5$, see [12, p.145]. [Intermediate result: $y_3 = -0.5520$]

**Exercise 88.** (*the local error of the EULER method*)
Study the following code snippet:

```
fpprintprec:5$
second(z):= z[2];
ye(x):= exp(x^2/2);
f(x,y):=x*y;
Pe: EULER( x,y, 0,1, 0.2, 5);              /* Euler method points */
L1: map(second,Pe);
L2: makelist(ye(x),x,0,1,0.2);
abs(L2-L1);
localError: lmax(abs(L2-L1));
globalError: sum(abs(L2-L1)[i],i,1,length(L1));
```

▷ Click to RUN the code.
**a.** Explain each code line in your own words.
**b.** Write a MAXIMA function `localError(.)`. – See e.g. [47, p.7], [20, p.3].
**c.** Write a MAXIMA function *truncation error* `truncError(.)`, which relates the global error to the mesh count $n$, i.e. the number of steps. See e.g. [47, p.7].

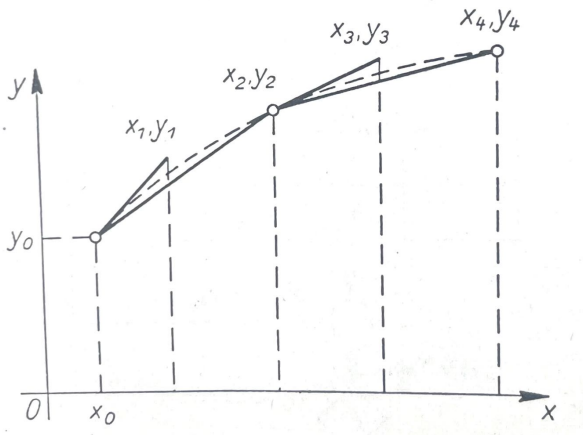
**Exercise 89.** **a.** Use Euler's method to approximate the solution of $y' = \frac{3t^2}{2y},\ y(0) = 1$ on the interval $0 \le t \le 2$ using $n = 2, 4, 8$ steps, see [43, p.230].
**b.** Find the local error between your approximations and the exact solution $y = \sqrt{1 + t^3}$.

**Exercise 90.** **a.** Use Euler's method to approximate the solution of the IVP
$y' = y^2 \sin(2x),\ y(0.5) = 1$ on the interval $0.5 \le x \le 3.5$ using $n = 6, 12, 24, 48$ steps,
**b.** Find the local error between your approximations and the exact solution $y = \frac{2}{\cos(2x)+2-\cos(1)}$.

**Exercise 91.** **a.** Use Euler's method with step size $h = 0.1$ to find the solution of the IVP $\frac{dy}{dx} = yx,\ y(0) = 1$ at the points with $x = 0.1, 0.2, 0.3, ...1.0$. Cf. [17, p.179].
**b.** Use Euler's method with step size $h = 0.01$ to find the solution of $\frac{dy}{dx} = yx,\ y(0) = 1$ at the points with $x = 0.01, 0.02, 0.03, ...1.0$.
**c.** Plot analogous figures to Fig.14 for a. and b. What do you observe?

**Exercise 92.** Do the examples in ▷ HELLEVIK: Euler's method. and the example in ▷ 2.6.4.

### 4.1.2 Modified EULER method alias 'RK2e of order 2'




[12, p.146]

Idea: we want to change the EULER procedure a bit to achieve a noticeable improvement with the same increment as before. We again calculate the slope in $x_1, y_1$, but start drawing the corresponding line element *not* at $x_1, y_1$ – we start the new line element at $x_0, y_0$ with the double argument length ending at point $x_2, y_2$. Here the process begins again ...

• Here is the *modified* EULER *recurrence formula*, see [17, p.186], [10, p.258], [1, p.47] :

$$Y_{n+1} = Y_n + \frac{h}{2} \cdot (f(x_n, Y_n) + f(x_{n+1}, Y_n + h \cdot f(x_n, Y_n))) \tag{4.4}$$

• A derivation of the formula (4.4) is given at [17, p.185ff] or at ▷ BAZETT: Integral solutions.

**a.** The following 'modified' EULER method[15] solves example 18 semi-automatic:

```
fpprintprec:5$
f(x,y) := x*y;
k1 : f(x,y);
k2 : f(x+0.2, y + 0.2*k1);
iterate2([x+0.2, y + 0.2*(k1+k2)/2],[x,y],[0,1],5);
```

▷ Click to RUN the code.

**b.** Here is the MAXIMA function `modiEULER`, which abstracts the above method.

```
/* MAXIMA : --- modified EULER method --- */
modiEULER(x,y, xo,yo, h,n):= block(
                k1 : f(x,y),
                k2 : f(x+h, y + h*k1),
                iterate2([x+h, y+h/2*(k1+k2)],[x,y],[xo,yo],n) );

/* Test using ODE  y'= xy with y(0)=1 */
f(x,y) := x*y;
modiEULER( x,y, 0,1, 0.2, 5);
```

[15] sometimes also called the RUNGE-KUTTA method of order 2

▷ Click to RUN the code.
Verify: recurrence (4.4) can be written as `y+h/2*(k1+k2)` using $k_1$ and $k_2$.
Solve exercises 85..89 using `modiEULER`.

**c.** Calculate the local/global/trunc errors of the modified EULER method for the ODE of Exercise 86. Compare with the errors of the simple EULER method.

**d.** Here is the equivalent code of the RUNGE-KUTTA *method of order 2*, which is a 1-1 translation of the recurrence formula (4.4) to MAXIMA:

```
RK2e( x,y, xo,yo, h,n) :=
       iterate2([x+h, y+h/2*( f(x,y)  + f(x+h,y+h*f(x,y)) )],
                [x,   y], [xo,yo], n) ;

f(x,y) := x*y;
RK2e( x,y, 0,1, 0.2, 5);
```

▷ Click to RUN the code.
Discuss pros and cons of `modiEULER` vs. `RK2e`!

**e.** Plot analogous figures to Fig.14 for a. or c. What do you observe?

**f.** Plot a figure analog to Fig.14, which shows simultaneous the exact solution, the `EULER` method approximation and the `RK2e` approximation for a step size of $h = 0.1$ for the ODE of exercise 87.a. What do you observe?

**Exercise 93.** Solve $y'(x) = \frac{1}{x} - \sin(x)$ and $\sqrt{1-x^2} \cdot y'(x) = 1$ approximately.

**Exercise 94.** Look at this code:

```
f(x,y):= sqrt(x)+sqrt(y);
modiEULER( x,y, 1,0.5, 0.05, 20);

wxdraw2d(xaxis = true,
    point_size=2,
    points_joined=true,
    xlabel="x",ylabel="y",
    points( modiEULER(f, x,y,1,0.5,0.05,20) ),
    color=red,
    explicit( ? ,x,0,1),       /* (?)  */
    title="modified EULER polygon")$
```

**a.** What is the discussed IVP?
**b.** Find the exact solution $y_e$ of the ODE.
**c.** Complete code line (?) with the term of $y_e$ and plot the scene.
**d.** What about the trunc error of the method in this example?

**Exercise 95.** Do some of the examples/exercises in ▷ Wiki: EULER method - Derivation. or in ▷ Cheever: EULER method. or ▷ Bazett: EULER method. or ▷ Fan: numerical methods.

## 4.2 RK2h alias HEUN's Method

We start by looking at the idea of the HEUN method for approximately solving ODE's.

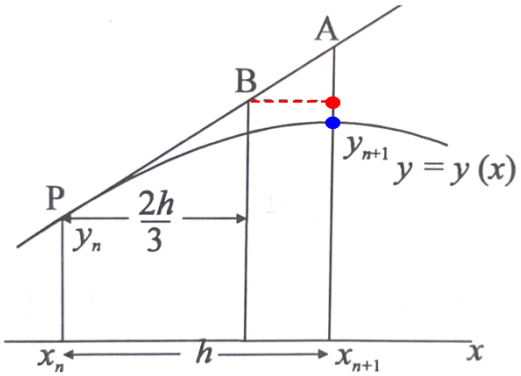



Figure 15: The modified EULER recurrence was written as $Y_{n+1} = Y_n + \frac{h}{2}(k_1 + k_2)$. Idea: maybe we get a better approximation, if we weight the average of $k_1$ and $k_2$ towards $k_2$ in the ratio of 3:1 instead of 1:1, see Fig.14. $A$ is the approximative value $Y^E_{n+1}$ for $y_{n+1}$ using EULER's method. $B$ and it's approximative value $Y^H_{n+1}$ (red dot) for $y_{n+1}$ using HEUN's method is closer to $y_{n+1}$ (blue dot), cf. [17, p.198].

- Here is the HEUN *recurrence formula*, also known as RALSTON's method:

$$Y_{n+1} = Y_n + \frac{h}{4} \cdot \left[ f(x_n, Y_n) + 3 \cdot f(x_n + \frac{2}{3}h, Y_n + \frac{2}{3}h \cdot f(x_n, Y_n)) \right] \tag{4.5}$$

- A derivation of the formula (4.5) is in [10, p.258] or at ▷ WIKI: HEUN's method.
- A MAXIMA implementation of the HEUN *recurrence formula* is:

```
/* MAXIMA --- HEUN's method --- */
HEUN( x,y, xo,yo, h,n) := block(
           k1 : f(x,y),
           k2 : f( x+2*h/3,  y+2*h/3*k1 ),
           iterate2([x+h, y+h/4*(k1+3*k2)],[x,y],[xo,yo],n) );

/* Test with ODE  y'= xy with y(0)=1 */
f(x,y):= x*y;
HEUN( x,y, 0,1, 0.2, 5);
```

▷ Click to RUN the code.

```
(%o11)  [[0,1],[0.2,1.02],[0.4,1.0826],[0.6,1.1954],[0.8,1.3733],[1.0,1.641]]
```

**Exercise 96.** Verify: the explicit recurrence formula (4.5) for the HEUN algorithm can be written as `y+h/4*(1*k1+3*k2)` using the values of $k_1$ and $k_2$.

**Exercises.**

**Exercise 97.** Use HEUN's method with step size $h = 0.1$ to find an approximate solution of $\frac{dy}{dx} = x - y^2$, $y(0) = -0.5$, see [12, p.145]. [Intermediate result: $y_3 = -0.5520$]

**Exercise 98.** (*the global error of the* HEUN *method*)
Study the following code snippet:

```
fpprintprec:5$
k(z):= z[2];
ye(x):=exp(x^2/2);
f(x,y):= x*y;
P : HEUN(x,y, 0,1, 0.2, 5);
L1: map(k,P);
L2: makelist(ye(x),x,0,1,0.2);
abs(L2-L1);
globalErrorHEUN: sum(abs(L2-L1)[i],i,1,length(L1));
```

▷ Click to RUN the code.
Explain each code line in your own words.
What is the corresponding truncation error?

**Exercise 99.** Use the HEUN method to solve the initial-value problem

$$\frac{dy}{dt} = \tan(y) + 1, \quad y_0 = 1, \ t \in [1, 1.1]$$

with step size $h = 0.025$. Control your results by looking at ▷ WIKI: Runge-Kutta>Use.

**Exercise 100.** Investigate the 'critical' IVP example in fig.1. of ▷ HAIRER/LUBICH: p.2.
Use `EULER` and `modiEULER` and `HEUN`.

**Exercise 101.** Do some of the examples and exercises of w.r.t. the HEUN ('improved' EULER) method in ▷ FAN: Numerical Methods.
If you like: do some of the Review problems.

**Exercise 102.** Review ▷ HELLEVIK: Heun's method.
and solve the examples there at ▷ : Newton's equation., ▷ : Falling sphere.
Maybe study ▷ : Generic second order Runge-Kutta method.

## 4.3 RK2m alias the Midpoint Method

We start by looking at the idea of the *Midpoint method* ('half step method') for approximately solving ODE's.

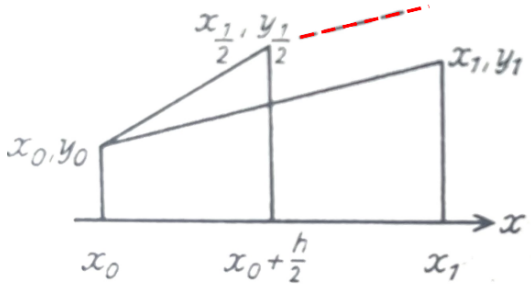



Figure 16: For the ODE $y' = f(x,y),\ y(x_0) = y_0$ one calculates $y'_0 = f(x_0, y_0)$ and walks in this direction to point $(x_{\frac{1}{2}}, y_{\frac{1}{2}}) := (x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}hy'_0)$. For this point the new slope $y'_{\frac{1}{2}}$ (dashed) is calculated with $f(x,y)$ and one goes again - starting at $P_0$ in this direction along a straight line to $P_1(x_1, y_1) := (x_0 + h, y_0 + hy'_{\frac{1}{2}})$. – Repeat this process starting at $P_1(x_1, y_1)$ instead of $P_0(x_0, y_0)$, cf. KAMKE [24, p.90].

- Here is the *Midpoint recurrence formula*:

$$Y_{n+1} = Y_n + h \cdot f\left(x_n + \frac{h}{2},\ Y_n + \frac{h}{2} \cdot f(x_n, Y_n)\right) \tag{4.6}$$

- A derivation of the formula (4.6) is in [10, p.257] or at ▷ WIKI: Midpoint method.
- A MAXIMA implementation of the MIDPOINT *recurrence formula* is:

```
/* MAXIMA : --- MIDPOINT METHOD --- */
midpoint(x,y, xo,yo, h,n) :=
         iterate2([x+h, y+h*f( x+h/2, y+h/2*f(x,y) ) ],[x,y],[xo,yo],n);

/* Test with ODE  y'= xy with y(0)=1 */
f(x,y):=x*y;
midpoint( x,y, 0,1, 0.2, 5);
```

▷ Click to RUN the code.

```
(%o11)  [[0,1],[0.2,1.02],[0.4,1.0826],[0.6,1.1954],[0.8,1.3733],[1.0,1.641]]
```

**Exercises.**

**Exercise 103.** Use the Midpoint method with step size $h = 0.1$ to find an approximate solution of $\frac{dy}{dx} = x - y^2$, $y(0) = -0.5$, see [12, p.145]. [Intermediate result: $y_3 = -0.5520$]

**Exercise 104.** (*the global error of the Midpoint method*)
Study the following code snippet:

```
fpprintprec:5$
k(z):= z[2];
ye(x):=exp(x^2/2);
f(x,y):=x*y;
P : midpoint(x,y, 0,1, 0.2, 5);
L1: map(k,P);
L2: makelist(ye(x),x,0,1,0.2);
abs(L2-L1);
globalErrorMidpoint: sum(abs(L2-L1)[i],i,1,length(L1));
```

▷ Click to RUN the code.
Compare this error with that of the EULER and HEUN methods.

**Exercise 105.** A RUNGE–KUTTA (RK) like formulation of the midpoint method using slope constants to structure the calculation is

```
/* MAXIMA : --- MIDPOINT METHOD as RK2 method --- */
midpoint1( x,y, xo,yo, h,n) :=  block(
               k1 : f(x ,y),
               k2 : f(x, y + h/2*k1),
               iterate2([x+h, y+h*k2], [x,y], [xo,yo], n));

/* Test with ODE  y'= xy with y(0)=1 */
f(x,y):=x*y;
midpoint1(x,y, 0,1, 0.2, 5);
```

A small variant is the following: (compare the 3 variants! Global errors?)

```
/* MAXIMA --- Midpoint method variant--- */
midpoint2(x,y, xo,yo, h,n) :=
         iterate2([x+h,
                   y+1/2*h*( f(x,y)+f(x+h,y+h)) ], [x,y], [xo,yo], n);

f(x,y):=2*x*y;
midpoint2( x,y, 0,1, 0.1, 15);
```

**a.** Do some of the $\begin{matrix}\text{examples}\\ \text{exercises}\end{matrix}$ in ▷ FAN: Numerical Methods. with the Midpoint method.
**b.** If you like: do also some of the Review problems.

## 4.4 RK4 alias the classic RUNGE-KUTTA method

We start by looking at the idea of the *classic* RUNGE–KUTTA *method of order 4* for approximately solving ODE's, cf. ▷ WIKI: Runge-Kutta, flow chart by [12, p.170].

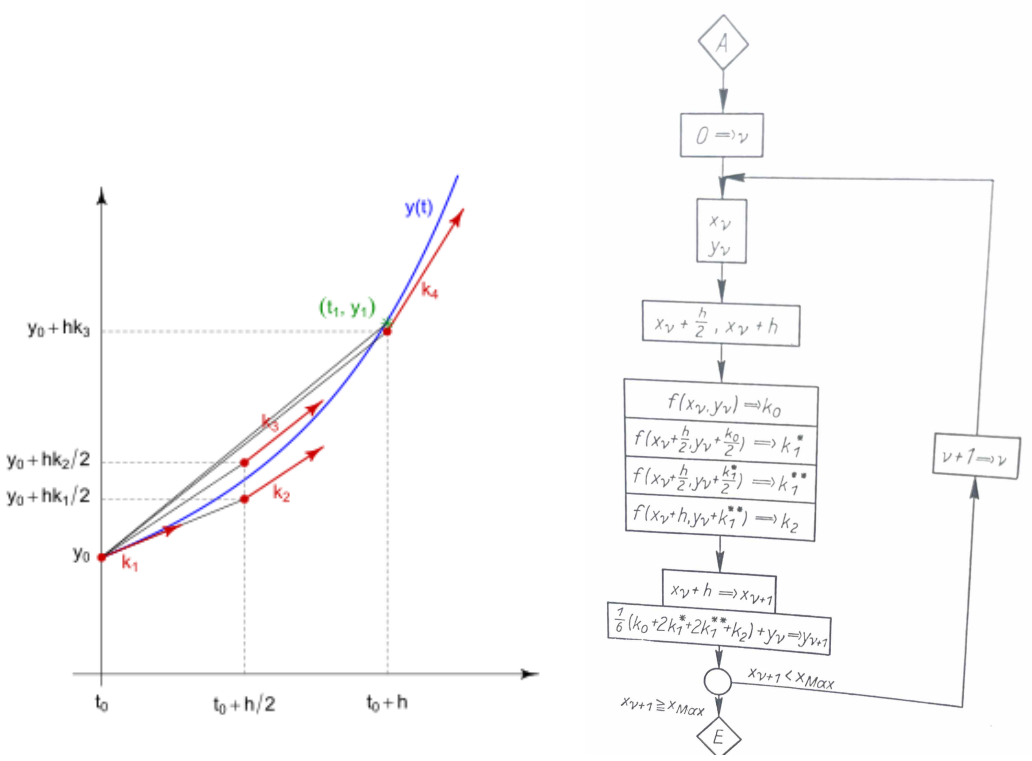



Figure 17: A RUNGE–KUTTA step for the ODE $y' = f(t, y)$ starting at $(t_0, y_0)$.
*Red:* Use four different slopes at different points near the solution.
*Green:* New (green) slope as weighted mean of the 4 red slopes.
This process is then repeated, now starting at $(t_1, y_1^R)$, a R(unge)-point.

The classic RUNGE–KUTTA method (RK4) gets the slope at 4 points and calculates the slope for the next step by an 'appropriate weighted' average of these 4 slopes $k_1, k_2, k_3, k_4$: RUNGE-KUTTA use a weigthed average in a ratio of $k_1 : k_2 : k_3 : k_4 = 1 : 2 : 2 : 1$.

### 4.4.1 the RUNGE–KUTTA RK4 recurrence

$$
\begin{aligned}
Y_{n+1} = Y_n \quad &+ \quad \frac{1}{6} \cdot [1 \cdot f(x_n, Y_n) \\
&+ \quad 2 \cdot f(x_n + \frac{h}{2}, Y_n + \frac{1}{2} \cdot f(x_n, Y_n)) \\
&+ \quad 1 \cdot f(x_n + \frac{h}{2}, Y_n + \frac{h}{2} \cdot f(x_n + \frac{h}{2}, Y_n + \frac{h}{2} \cdot f(x_n, Y_n))) \\
&+ \quad 1 \cdot f(x_n + \frac{h}{2}, Y_n + \frac{h}{2} \cdot f(x_n + \frac{h}{2}, Y_n + \frac{h}{2} \cdot f(x_n + \frac{h}{2}, Y_n + \frac{h}{2} \cdot f(x_n, Y_n))))]
\end{aligned}
\tag{4.7}
$$

• (4.7) displays the explicit RUNGE–KUTTA *recurrence formula* for the above idea.

• A derivation of the formula (4.7) is in ▷ WIKI: Runge-Kutta > Derivation or in [24, p.92-93]. Formula (4.7) is incomprehensible in the presented explicit form, not memorizable and shadows the simple structure of the RUNGE–KUTTA method.
So forget about it and let's program it structured and shortened using the four slopes $k_1, .., k_4$: Then this classic method becomes a captivating simplicity and focus on the nice property that the successive calculation of the slopes $k_i$ only need the previous value $k_{i-1}$, see [35, p.98] and [17, p.201] or ▷ SÜLI: p.13–19 or ▷HELLEVIK: RK 4th.

• The *classic* RUNGE–KUTTA *method* (RK4) in MAXIMA using the 4 slopes $k_1, k_2, k_3, k_4$:

```
/* MAXIMA : --- classic RUNGE-KUTTA  method of order 4 --- */

RK4(x,y,xo,yo,h,n):= block(
                      k1 : f(x,y),
                      k2 : f(x+h/2, y + h/2*k1),
                      k3 : f(x+h/2, y + h/2*k2),
                      k4 : f(x+h  , y + h*k3),
                      iterate2([x+h, y+h/6*(1*k1+2*k2+2*k3+1*k4)],
                               [x  , y], [xo,yo], n) );

/* Test ODE: y'= xy with y(0)=1 */
f(x,y):= x*y;
RK4( x,y, 1,1, 0.2, 10);
```

▷ Click to RUN the code.

```
(%o17)  [[0,1],[0.2,1.0202],[0.4,1.0833],[0.6,1.1972],[0.8,1.3771],[1.0,1.6487]]
```

### 4.4.2 the build-in `rk(.)` method

`RK4` is build-in in MAXIMA as function

```
rk( f(x,y), y,   yo, [x, xo,b, h])
      ODE    var init   domain
```

cf. https://maxima.sourceforge.io/docs/manual/maxima_112.html#index-rk

▷ "The independent variable $y$ is specified with *domain*, which must be a list of four elements as, for instance: $[x,\ 0,8,\ 0.1]$ – the first element of the list identifies the independent variable $x$, the second and third elements are the initial $x_0$ and final values $b$ for that variable, and the last element sets the increments $h$ that should be used within that interval.
*Example*: To solve numerically the differential equation $\frac{dy}{dx} = x - y^2$ with initial value $y(x = 0) = 1$, in the interval of $x$ from 0 to 8 and with increments of 0.1 for $x$, use:

```
results: rk(x-y^2, y,1, [x, 0,8, 0.1])$
plot2d ([discrete, results])$
```

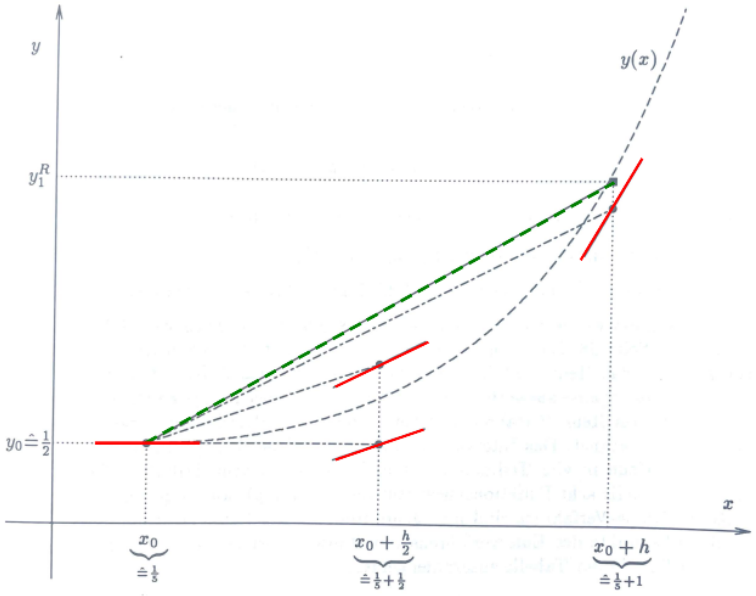
the results will be saved in the list `results` and the plot will show the solution obtained, with $x$ on the horizontal axis and $y$ on the vertical axis."

**Exercises.**

**Exercise 106.** Solve the test ODE: $y' = xy$ with $y(0) = 1$ using the build-in method `rk()`.

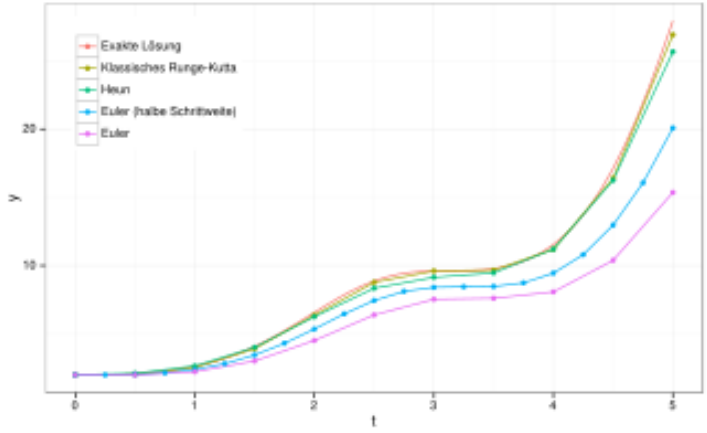
**Exercise 107.** **a.** Use `RK4` function with step size $h = 0.1$ to find an approximate solution of ODE $\frac{dy}{dx} = x - y^2$, $y(0) = -0.5$, see [12, p.145]. [Intermediate result: $y_3 = -0.5520$]
**b.** Solve **a.** with the built-in function `rk`, see the remark above.
**c.** Calculate the global error for the methods in a. and b. Compare.

**Exercise 108.** Solve $y'(x) = 2 \cdot (x - \frac{1}{5})(y + \frac{1}{2})$, $y(\frac{1}{5}) = \frac{1}{2}$ approximately using `RK4` only for the first RK-step, i.e. for $h = 1$, cf. [35, p.98].




– Calculate the the four slopes $k_1, .., k_4$ at the $x$-positions seen in the figure.
– Calculate now 'by hand' the 'green' slope $k$ as weighted mean of the red slopes $k_i$.
– Check the result for plausibility.
– Plot the whole scene of the figure.
– Redo this exercise using built-in function `rk`.

**Exercise 109.** Look at ▷ WIKI: Runge-Kutta and the ODE $y' = \sin(t)^2 \cdot y$:



**a.** Solve the ODE $y' = \sin(t)^2 \cdot y$ approximately with `RK4/rk` on the domain of the figure.
**b.** Determine the exact solution $y_e(x)$ (red in the figure).
**c.** The figure also shows the methods HEUN (green), modified EULER (cyan) and EULER (magenta). Use these methods to solve the given ODE.

**Exercise 110.** Look at ▷ WIKI NL: Runge-Kutta and the ODE $\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{-t}{x}$.
**a.** Solve the ODE $\frac{\mathrm{d}x}{\mathrm{d}t} = \frac{-t}{x}$ approximately with `RK4/rk` on the domain $t \in [0, 1]$ with $x(0) = 1$ and $h = 0.1$.
**b.** Determine the exact solution $y(x)$.

**Exercise 111.** (RK3) This is `RK3` method formulated in MAXIMA language:

```
/* MAXIMA --- RUNGE-KUTTA order 3 --- */
RK3(x,y,xo,yo,h,n) := block(
                k1 :  f(x,y),
                k2 : f(x+h/2, y + h/2*k1),
                k3 : f(x+h, y + 2*h*k2-h*k1),
                iterate2([x+h, y+h/6*(1*k1+4*k2+1*k3)],[x,y],[xo,yo],n));

/* Test ODE: y'= xy with y(0)=1 */
f(x,y):= x*y;
RK3( 0,1, 0.2, 5);
```

▷ Click to RUN the code.
**a.** Do exercise 105 using method `RK3`. Compare. Truncation error(s)?
**b.** Do exercise 107 using `RK3` and reproduce the plot incl. method `RK3`.

**Exercise 112.** Solve the OVP $\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{2y-x}{x}$, cf. [42, p.26]
**a.** ... approximately with `RK4/rk/RK3` on the domain $x \in [0, 2]$ with the IV $y(0) = 0.5$ using $h = 0.2$.
**b.** ... exact with solution $y(x) =$? [Result: $y(x) = x^2 + x$

**Exercise 113.** Solve the OVP $y' = y - t^2 + 1$, cf. [18, p.260], ...
**a.** approximately with `RK4/rk` on the domain $t \in [1, 3]$ with $y(1) = 2$ and $h = 0.1$.
**b.** Determine the exact solution $y(t)$. [Result: $= y(t) = (t + 1)^2 - \frac{1}{2}e^t$
**c.** Compare the methods EULER (with h=0.025), Midpoint (h=0.05), RK4/rk (h=0.1) on the mesh points $0.1, 0.2, 0.3, 0.4, 0.5$ working on this ODE, where each of the techniques requires 20 evaluations. Make an EXCEL like table with MAXIMA.
**d.** Calculate the absolute errors for the methods in c. Compare.

**Exercise 114.** Read ▷ CHEEVER: Runge-Kutta Method and do the example with our `RK4`.

**Exercise 115.** Read ▷ WIKI: Runge-Kutta>Use and do the example with our `RK4`.

**Exercise 116.** Read ▷HELLEVIK: Falling sphere using RK4. and do the example with our `RK4`.

**Exercise 117.** Read ▷ ESE: Runge-Kutta and do the example ODE $y' = x^2$ and the exercise with our `RK4`.

**Exercise 118.** Do the examples and problems in ▷ Fan: Numerical Methods. w.r.t. the RK method. If you like: do some of the Review problems.

**Exercise 119.** Re-do the four examples in ▷Hellevik: EXERCISES.

**Exercise 120.** * Study ▷Hellevik: Basic notions on numerical methods for IVPs. and adapt these notations for 'our' error handling. Look also at ▷ ..: On errors.

**Exercise 121.** ** Study ▷Hellevik: Absolute stability of numerical methods Euler, Heun, RK.

**Exercise 122.** Do some Further Reading, e.g.
▷ stackexchange: Lehmann: On history
▷ scholarPedia: Butcher: Runge-Kutta
▷ scholarPedia: Shampine & Thompson: IVPs
▷ Hairer/Lubich

♡

Let's sum up our numerical solution methods for IVPs:

| METHOD | *Math* | Maxima |
|---|---|---|
| | IVP $y' = f(x,y),\ y(x_o) = y_o$ | `f(x,y):= ...;` |
| | Euler | `EULER(x,y, xo,yo, h, n);` or `RK1(..)` |
| | modified Euler | `modiEULER(x,y, xo,yo, h,n); RK2e(..)` |
| | Heun | `HEUN(x,y, xo,yo, h, n); RK2h(..)` |
| | Midpoint | `midpoint(x,y, xo,yo, h,n)); RK2(..)` |
| | classic 4th Runge–Kutta | `RK4(x,y,xo,yo,h,n)` |
| where | | |
| | `x,y` | the variables used in the iteration by $f$ |
| | `xo,yo` | the initial values with $y(x_o) = y_o$ |
| | `h` | the step size used in the method |
| | `n` | the number of repeated iterations |

♡

# 5 Systems of IVP's

A physical process is often not described by a *single* differential equation, but by a system of *two* or more differential equations which are related to one another, i.e. are coupled. Such a system of two simultaneously working and coupled $1^{st}$-order ODEs has the form

$$ODEsys: \left\{ \begin{array}{l} x' = f(t,x,y),\ x(t_o) = x_o \\ y' = g(t,x,y),\ y(t_o) = y_o \end{array} \right\} \tag{5.1}$$

For a system of two ODEs, two initial conditions are now required. We seek for two solution functions $x(t)$ und $y(t)$ that fulfills both equations $f$ and $g$ and both initial conditions. *Moreover*, a $2^{nd}$-order differential equation of the form $y'' = f(x,y,y')$ can be replaced by two coupled $1^{st}$-order differential equations. It is therefore sufficient to know a solution method for two coupled $1^{st}$-order differential equations.

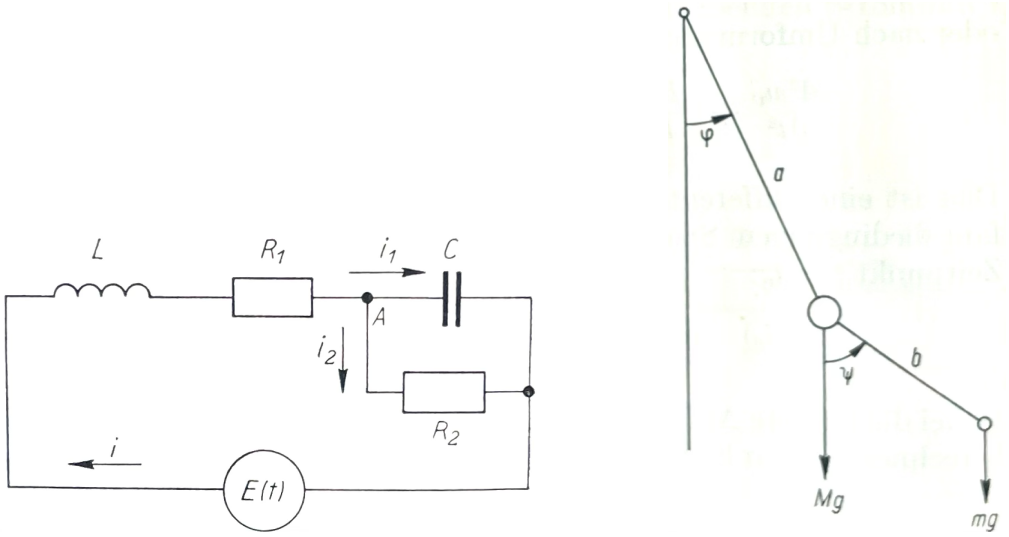



Figure 18: BRÄUNING [12, p.11] motivates the dealing with systems of ordinary differential equations through their use in the natural sciences like:
*Left:*(*circuit*) "An electrical circuit contains a generator with the time-dependent voltage E(t), a spool with the inductance L an ohmic resistor R1 as well as in series the parallel connection of a second ohmic resistor R 2 and 1 capacitor with the capacity C. If we want a differential equation for the time of the voltage uC on the capacitor, the result is system of 2 first-order differential equations: $\mathbf{L\frac{di}{dt} + R_1 i + u_C = E(t)}$ & $\mathbf{i = C\frac{du_C}{dt} + \frac{u_C}{R_2}}$."
*Right:* (*coupled springs*) "When examining the plane movement of a double pendulum, one is led to a complicated 4th order differential equation system, where the highest derivatives therein are $\ddot{\varphi}$ and $\ddot{\psi}$. ..."

In this chapter we first study *systems of two ODEs* and corresponding MAXIMA solution methods. Then we deal with *$2^{nd}$-order differential equations* of the form $y'' = f(x,y,y')$ and specialized numerical solution methods.

## 5.1 Solving IVP systems

**Example 19.** (*solving an IVP system of 2 equations in* MAXIMA)
Solve numerically the system

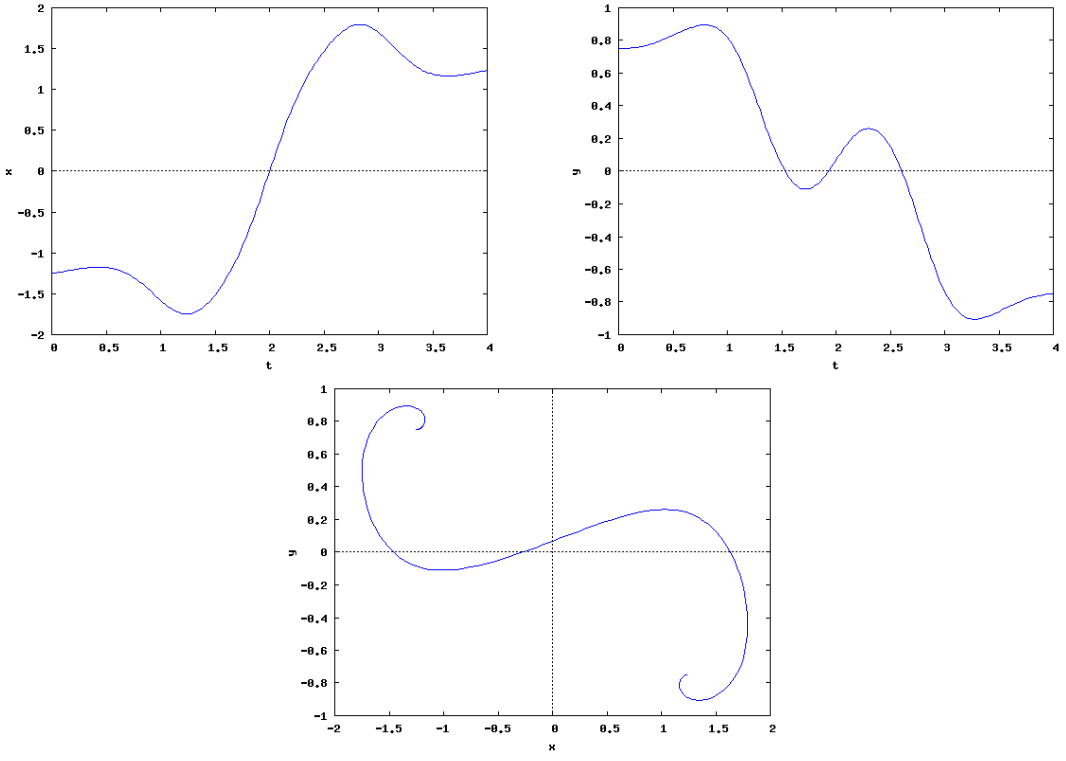
$$\frac{dx}{dt} = 4 - x^2 - 4y^2 \;\&\; \frac{dy}{dt} = y^2 - x^2 + 1$$

for t between 0 and 4, and with values of -1.25 and 0.75 for $x$ and $y$ at $t = 0$.
Cf. https://maxima.sourceforge.io/docs/manual/maxima_112.html#index-rk

*Solution*: We have the system $\left\{\begin{smallmatrix} x'=f(t,x,y)=4-x^2-4y^2,\ x(0)=-1.25 \\ y'=g(t,x,y)=y^2-x^2+1\ \ ,\ y(0)=0.75 \end{smallmatrix}\right\}$.
To get the solutions $x(t)$ and $y(t)$, we call the build-in MAXIMA function `rk(.)`:

```
/* MAXIMA -- ODE system of 2 coupled equations */
sol: rk([4-x^2-4*y^2, y^2-x^2+1], [x, y], [-1.25, 0.75],  [t, 0,4, 0.02])$
       /* eq1        eq2          vars     x0    y0       region  h   */

wxplot2d([discrete, makelist([p[1], p[2]], p, sol)],
         [xlabel, "t"], [ylabel, "x"])$
wxplot2d([discrete, makelist([p[1], p[3]], p, sol)],
         [xlabel, "t"], [ylabel, "y"]),
wxplot2d([discrete, makelist([p[2], p[3]], p, sol)],
         [xlabel, "x"], [ylabel, "y"])$
```

The plots show the solutions for variables x and y as a function of t and the plot $(x_t, y_t)$.



### 5.1.1 iterate3

In order to program our own numerical solution methods for systems of ODEs using `iterate`, we need a version `iterate3`, which deals with 3 recurrent entries.

```
/* MAXIMA --- iterate3 for triples --- */
iterate3(u,x,xo,n) := block(
            [    l: [ [xo[1],xo[2],xo[3]]], numer: true,
               val: [  xo[1],xo[2],xo[3]]],
    for i thru n do
        (val: subst([x[1]=val[1], x[2]=val[2], x[3]=val[3]], u),
           l: cons(val,l)),
        reverse(l));
```

We show two examples of a simple approximative solutions of ODE pairs using `iterates3`.

**Example 20.** (a *simple Euler method for a system of 2 coupled ODEs*, cf. [9, p.258])
Solve the system $\left\{\begin{smallmatrix} x'=f(t,x,y)=y, & ,x(0)=4 \\ y'=g(t,x,y)=-ty-x-6t^2, & y(0)=0 \end{smallmatrix}\right\}$.

```
/* Do not forget to invoke iterate3 before .. */
fpprintprec: 5$
f(t,x,y) := y;                              /* the system (f,g) */
g(t,x,y) := -t*y-x-6*t^2;
a:0; b:4; c:0;                              /* ICs */
h: 1/8;                                     /* chose step size */
n: 16;                                      /* chose number of iterations */
iterate3([t+h, x+h*f(t,x,y),                /* updating the ODE system */
              y+h*g(t,x,y)], [t,x,y], [a,b,c], n);
```

▷ Click to RUN the code.

```
(%o21)  [[0,4,0],[0.125,4,-0.5],[0.25,3.9375,-1.0039],
        3.6231,-2.0227],[0.625,3.3702,-2.5367],[0.75,3.0
        [1.0,2.2253,-4.0878],[1.125,1.7143,-4.6049],[1.2
        5.635],[1.5,-0.20578,-6.1467],[1.625,-0.97412,-(
        -2.7015,-7.667],[2.0,-3.6598,-8.1691]]
```

The display shows in the last triple entry, that for $t = 2.0$ we get $x(2) \approx -3.6598$ and $y(2) \approx -8.1691$.

**Example 21.** Solve the system $\left\{\begin{smallmatrix} y'=f(x,y,z)=2x-3z, & y(0)=1 \\ z'=g(x,y,z)=y-2z, & z(0)=0 \end{smallmatrix}\right\}$.
This system has the exact solution $(y_x, z_x) = (\cosh(x) + 2\sinh(x), \sinh(x))$.

```
fpprintprec:5$
h: 0.1;
f(x,y,z) := 2*y-3*z;
g(x,y,z) := y-2*z;
iterate3([x+h, f(x,y,z)*h + y, g(x,y,z)*h + z], [x,y,z], [0,1,0], 10);
```

```
/* test: */
y5: cosh(1.0)+2*sinh(1.0);
z5: sinh(1.0);
```

▷ Click to RUN the code.

```
(%o6)   [[0,1,0],[0.1,1.2,0.1],[0.2,1.41,0.2],[0.3,1.632,0.3
        ,0.51001],[0.6,2.3916,0.62006],[0.7,2.6839,0.73521],[0
        1.0,3.7163,1.1225]]
(y5)    3.8935
(z5)    1.1752
```

♡ *We now transfer our knowledge about solution methods for a single ordinary differential equation in* MAXIMA *in rapid succession to systems of 2 coupled IVPs. We demonstrate the respective procedure with a typical example directly after the implementation.*

### 5.1.2 Euler's method for systems of ODEs

We interpret the EULER method for a single ODE now for a system of 2 IVP's, cf. BRONSON [9, p.258], ▷HELLEVIK: Systems., ▷: Free fall. We have:

```
/* MAXIMA --- EULER method for systems of IVP --- */
EULERsys(t,x,y, to,xo,yo, h,n) :=
     iterate3([t+h, x+h*f(t,x,y), y+h*g(t,x,y)], [t,x,y], [to,xo,yo], n)$

/* Test: */
f(t,x,y) := y;
g(t,x,y) := x+t;
EULERsys(t,x,y, 0,0,1, 0.2, 5);
```

▷ Click to RUN the code.

```
(%o22)  [[0,0,1],[0.2,0.2,1],[0.4,0.4,1.08],[0.6,0.616,1.24],[0.8,0.864,1.4832],[
        1.0,1.1606,1.816]]
```

### 5.1.3 PICARD–LINDELÖF method of successive approximation for systems

BRÄUNING [12, p.160] formulate the PICARD–LINDELÖF method for a system of 2 IVP's, which I quote here:

```
fpprintprec:5$  ratprint:false$  kill(arrays)$

f(x,y,z) := z;      /* =y' */
g(x,y,z) := y + x; /* =z' */

[a,b] : [0,0];    /* y(a) = b */
[c,d] : [0,1];    /* z(c) = d = y'(c) */
```

```
y[0](x) := b;
y[n](x) := b + integrate( f(t, y[n-1](t), z[n-1](t)), t,a,x);

z[0](x) := d;
z[n](x) := d + integrate( g(t, y[n](t), z[n-1](t)), t,c,x);

y[10](x), expand;
y[10](1), numer;
z[10](1), numer;
```

▷ Click to RUN the code.

$$(\%o18)\quad \frac{3x^5}{20}-\frac{x^4}{8}+\frac{x^3}{3}+\frac{x^2}{2}+2x+1$$

```
(%o19)  3.8583
```

```
(%i23)  y[10](1), numer;
(%o23)  3.8934

(%i20)  z[10](1), numer;
(%o20)  1.1752
```

### 5.1.4 RK4sys alias RUNGE–KUTTA method for Systems of IVP's

We define the analogon of RK4 for a system of IVP's, cf. *Bronson* [9, p.258] or BRÄUNING[12, p.258].

```
/* MAXIMA --- RUNGE-KUTTA method RK4sys for systems of IVP --- */

RK4sys( x,y,z, xo,yo,zo, h,n):= block(
    k1 : h*f(x,y,z),
    l1 : h*g(x,y,z),
    k2 : h*f(x+1/2*h, y + 1/2*k1, z + 1/2*l1),
    l2 : h*g(x+1/2*h, y + 1/2*k1, z + 1/2*l1),
    k3 : h*f(x+1/2*h, y + 1/2*k2, z + 1/2*l2),
    l3 : h*g(x+1/2*h, y + 1/2*k2, z + 1/2*l2),
    k4 : h*f(x+    h, y +     k3, z + l3),
    l4 : h*g(x+    h, y +     k3, z + l3),
    iterate3([x+h, y + 1/6*(1*k1+2*k2+2*k3+1*k4),
                   z + 1/6*(1*l1+2*l2+2*l3+1*l4)],
             [x,y,z],[xo,yo,zo],n) )$

/* Test: ODE system {f=z, g=y+x} with y(0)=0 & z(0)=1. Wanted: y=? & z=? */
fpprintprec:5$
f(x,y,z) := z;
g(x,y,z) := y+x;
RK4sys(x,y,z, 0,0,1, 0.2, 5);
```

▷ Click to RUN the code.

```
(%o50)  [[0,0,1],[0.2,0.20267,1.0401],[0.4,0.42149,1.1621],[0.6,0.67329,1.3709],
        ,0.97619,1.6749],[1.0,1.3504,2.0861]]
```

## 5.2 IVP of $2^{nd}$ Order

One can replace a second-order differential equation $y'' = f(x, y, y')$ by two coupled first-order differential equations and then use the methods in 5.1 for systems of differential equations to solve such an equation: we make the simple substitution $y' = z$, and the second-order differential equation $y'' = f(x, y, y')$ transforms into the two 1st order ODEs $y' = z \;\&\; z' = f(x, y, y')$. Ergo, it is in principe sufficient to know a solution method for two coupled 1st-order differential equations to solve a a 2nd-order ODE.

We demonstarte this in the next example.

**Example 22.** (*one $2^{nd}$ order ODE $\rightsquigarrow$ two $1^{st}$ order ODEs*, cf. BRONSON, [9, p.258])
**a.** Reduce the IVP $y'' - y = x;\ \begin{smallmatrix} y(0)=0 \\ y'(0)=1 \end{smallmatrix}$ of order 2 to a system of two $1^{st}$ order IVP's.
**b.** Find $y(1)$ using Euler's method `EULERsys` for systems with $h = 0.1$.
**c.** Find $y(1)$ using `RK4sys` method for systems with $h = 0.1$.
**d.** Find $y(1)$ using PICARD–LINDELÖF method with a 'good' index $i$ for $(y[i](1), z[i](1))$.
**e.** Find $y(1)$ using `rk(.)` for systems with $h = 0.1$.

*Solution*:

**a.** Defining $z := y'$, we have $z(0) = y'(0) = 1$ and $z' = y''$.
The given ODE is therefore $y'' = y + x = z'$. To sum up:

| TRANSFORMATION | *one $2^{nd}$ order ODE* | two $1^{st}$ order ODEs |
|---|---|---|
| | $y'' = f(x, y, y')$ | $y' := z \;\wedge\; z' = f(x, y, y')$ |
| | $y(x_o) = a$ | $y(x_o) = a$ |
| | $y'(x_o) = b$ | $z(x_o) = b$ |
| here | | |
| | $y'' = f(x, y, y') = y + x$ | $y' := z \;\wedge\; z' = f(x, y, y') = y + x$ |
| | $y(0) = 0$ | $y(0) = 0$ |
| | $y'(0) = 1$ | $z(0) = 1$ |

We obtain the $1^{st}$ order ODE system $\{y' = z,\ z' = y + x,\ y(0) = 0,\ z(0) = 1\}$, which we have used in the tests 5.1.2 to 5.1.4 before.
We now calculate the particular solution point $y(1)$ of the solution function $y(x)$ using numeric methods from 5.1

**b.** Euler's method

```
f(x,y,z) := z;
g(x,y,z) := y+x;
EULERsys(x,y,z, 0,0,1, 0.1, 10);
```

▷ Click to RUN the code.

```
(%o54)  [[0,0,0],[0.1,0,0],[0.2,0,0.01],[0.3,0.001,0.03],[
        0.01001,0.1005],[0.6,0.02006,0.1515],[0.7,0.03521,0.2
        0.9,0.085264,0.37268],[1.0,0.12253,0.47121]]
```

**c.** `RK4sys` method

```
fpprintprec:5$
f(t,x,y):=y;
g(t,x,y):=x+t;
RK4sys(t,x,y, 0,0,1, 0.1, 10);
```

▷ Click to RUN the code.

```
(%o66)  [[0,0,1],[0.1,0.10033,1.01],[0.2,0.20267,1.0401],[0.3,0.30904,1
        0.4215,1.1621],[0.5,0.54219,1.2553],[0.6,0.67331,1.3709],[0.7,0.8
        0.97621,1.6749],[0.9,1.153,1.8662],[1.0,1.3504,2.0862]]
```

**d.** Picard–Lindelöf method

```
fpprintprec:5$   ratprint:false$   kill(arrays)$  /* because of y[n] */

f(x,y,z):= z;
g(x,y,z):= y+x;

[a,b] : [0,0];   /* x0=a, b=y0=y(x0) */
[c,d] : [0,1];   /* x0=c, d=y'(x0)=z(x0) */

y[0](x) := b;
y[n](x) := b + integrate( f(t, y[n-1](t), z[n-1](t)), t,a,x);

z[0](x) := d;
z[n](x) := d + integrate( g(t, y[n](t), z[n-1](t)), t,c,x);

y[9](x), expand;
y[9](1), numer;
```

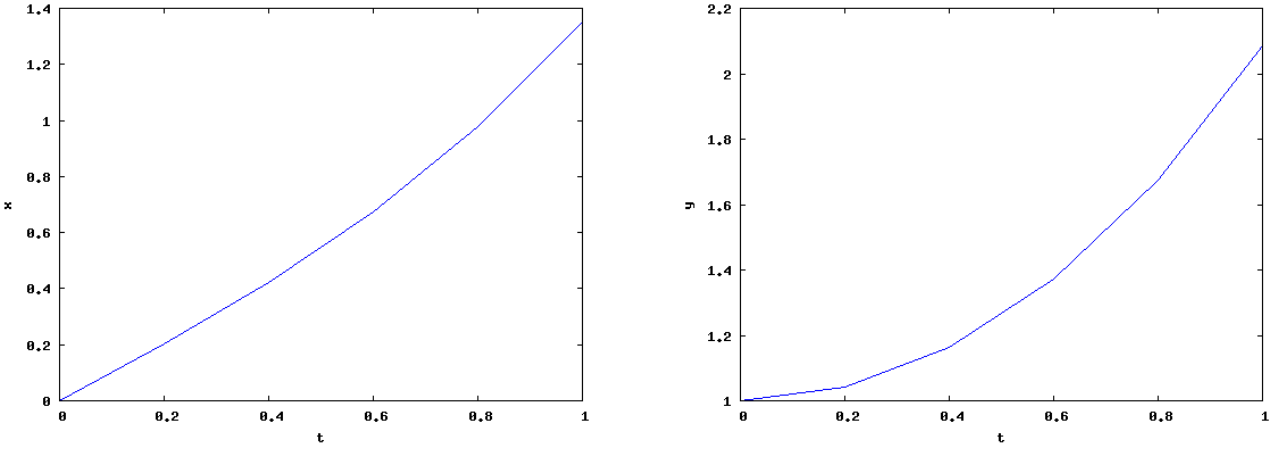
▷ Click to RUN the code.

```
(%i80)  z[10](1), numer;
(%o80)  2.0862

(%i81)  y[10](1), numer;
(%o81)  1.3504
```

**e.** `rk(.)` build-in for systems

```
fpprintprec:5$
sol: rk([y, x+t], [x, y], [0, 1],  [t, 0, 1, 0.2])$

wxplot2d([discrete, makelist([p[1], p[2]], p, sol)],
         [xlabel, "t"], [ylabel, "x"])$
wxplot2d([discrete, makelist([p[1], p[3]], p, sol)],
         [xlabel, "t"], [ylabel, "y"])$
```

▷ Click to RUN the code.




We check on the left graph of $x(t)$, that $x(t=1) \approx 1.3$ and on the right graph of $y(t)$, that $y(t=1) \approx 2.1$, which looks OK.

♡

We now present two specialized functions called `RK4ode2` and `RK4romer`, which allows to give up the transformation $2^{nd}\,orderODE \rightsquigarrow 1^{st}\,orderSystem$ and direct use the given 2nd order ODE $y'' = f(x, y, y')$ as input for these methods.

### 5.2.1 Braeuning's method for $2^{nd}$ order IVP

`Braeuning` [12, p.181 ff] discuss the following variant of `RK4sys` and give recurrence formulas and an example. We translate this algorithm to MAXIMA, `yp` ('*y prime*') is $y'$.

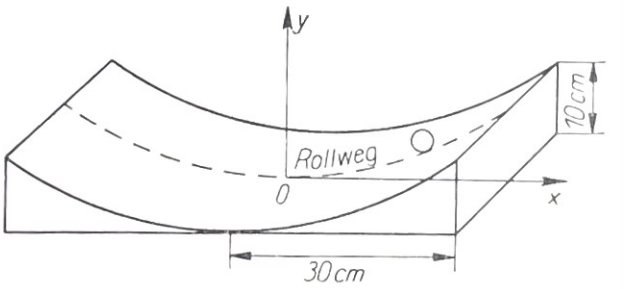
```
/* MAXIMA -- BRAEUNING's method for 2nd order IVP --- */
RK4ode2( x,y,yp, xo,yo,ypo, h, n):= block(
        k0 : 1/2*h^2* f(x,y,yp),
        k1 : 1/2*h^2* f(x+h/2, y + 1/2*h*yp + 1/4*k0, yp + k0/h),
       k1p : 1/2*h^2* f(x+h/2, y + 1/2*h*yp + 1/4*k0, yp + k1/h),
        k2 : 1/2*h^2* f(x+h,   y +     h*yp + k1p,    yp + 2*k1p/h),
         k : 1/3*(k0+  k1+  k1p   ),
         l : 1/6*(k0+2*k1+2*k1p+k2),
      iterate3([x+h, y+h*yp+k, yp+2*l/h],[x,y,yp],[xo,yo,ypo],n) );

/* Test 2nd order IVP y''=x+y+y' with y(0)=1 and y'(0)=1  */
f(x,y,yp):=x+y+yp;
RK4ode2(x,y,yp, 0,1,1, 0.1, 10);
```

▷ Click to RUN the code.

```
(%o69)  [[0,1,1],[0.1,1.1107,1.221],[0.2,1.2458,1.4887],[0.3,1.4102,1.8107],[0.4,
        1.61,2.1962],[0.5,1.8519,2.6558],[0.6,2.1441,3.2023],[0.7,2.4958,3.8505],[0.8,2
        ,4.6179],[0.9,3.424,5.5251],[1.0,4.0286,6.5965]]
```

**Remark.** If the IVP of $2^{nd}$ order does not depend on $y'$ (yp), there are more simplifications possible. Example: $y'' = -\sin(y), y(0) = 0, y'(0) = 2$. Solve this $2^{nd}$ order IVP.

**Exercise 123.** (BRÄUNING, [12, p.27]) We let a ball roll on a fall line of a parabolic cylinder, where the dimensions can be taken from the picture. We measure $x$ and $y$ in centimeters, so the path of the ball has the equation $y = \frac{1}{90}x^2$. We use the value $g = 981$ $[c/s^2]$ for the gravitational acceleration.




LEXICON:

| *German* | English |
|---|---|
| Rollweg | path of contact point |

Then one gets – for physical reasons[16] – the differential equation of the motion

$$\ddot{x} = -\frac{x}{2025 + x^2} \cdot (31532 + \dot{x}^2)$$

Calculate the motion $x(t)$ of the ball for $x(0) = 30$ cm and $x'(0) = 0$ cm.
What is the approximative value of $x(20)$?

### 5.2.2 Romer's method for $2^{nd}$ order IVP

Romer [42, p.28 ff] discuss the following variant of RK4sys, which we translate to MAXIMA.

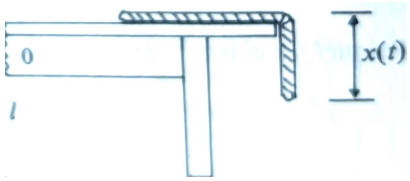
```
/* MAXIMA --- ROMER's method for 2nd order ODE --- */
RK4romer( x,y,yp, xo,yo,ypo, h, n):= block(
         k1 : h* f(x,y,yp),
         k2 : h* f(x+h/2, y + 1/2*h*yp + h/8*k1, yp + k1/2),
         k3 : h* f(x+h/2, y + 1/2*h*yp + h/8*k1, yp + k2/2),
         k4 : h* f(x+h,   y +     h*yp + h/2*k3, yp + k3),
         iterate3([x+h, y+h*(yp+1/6*(k1+k2+k3)), yp+1/6*(k1+2*k2+2*k3+k4)],
                  [x,y,yp],[xo,yo,ypo],n) );

/* Test: 2nd order ODE y''=x+y+y' with y(0)=1 and y'(0)=1; wanted: y(1)=? */
fpprintprec:5$
f(x,y,yp):=x+y+yp;
RK4romer(x,y,yp, 0,1,1, 0.1, 10);
```

▷ Click to RUN the code.

```
(%o76)  [[0,1,1],[0.1,1.1107,1.221],[0.2,1.2458,1.4887],[0.3,1.4102,1.8107],[0.4,
        1.61,2.1962],[0.5,1.8519,2.6558],[0.6,2.1441,3.2023],[0.7,2.4958,3.8505],[0.8,2.9
        ,4.6179],[0.9,3.424,5.5251],[1.0,4.0286,6.5965]]
```

[16] The interested reader can get the derivation (in German) of the ODE on request from the author.

**Example 23.** (a classic problem, ROMER [42, p.30 ff])
A perfectly flexible rope of length $\ell$ and mass $m$ glide smoothly over a table edge.




We have the initial conditions: $x(t=0) = 0.1$ [m], $\dot{x}(t) = 0$, $g = 9.81$ [m/s$^{-2}$], $\ell = 1$ [m].

**a.** Determine the ropes position $x(t = 0.5)$ [sec].

**b.** At which moment $t$ the rope leave completely the table?

**c.** The exact solution is $x(t) = \frac{x_o}{2} \cdot (e^{-\sqrt{g/\ell}\cdot t} + e^{\sqrt{g/\ell}\cdot t})$.
Compare with the approximate solution.

*Solution*:
ad **a.** : We have $M\ddot{x} = \frac{M}{\ell} \cdot x \cdot g \rightsquigarrow \ddot{x} = \frac{g}{\ell} \cdot x$

```
fpprintprec:5$
g: 9.81$ L: 1$
f(x,y,yp) := g/L*y;      /* = y'' */

                /* x0 y0   yp0      h    n */
RK4romer(x,y,yp,  0, 0.1, 0,     0.05, 20);
```

▷ Click to RUN the code.

```
(%o18)  f(x,y,yp):=g/L*y
(%o19)  [[0,0.1,0],[0.05,0.10123,0.049251],[0.1,0.10495,0.099712],[0.15,0.11124,
        0.15262],[0.2,0.12027,0.20929],[0.25,0.13225,0.27109],[0.3,0.14749,0.33956],[
        0.16635,0.41637],[0.4,0.1893,0.50342],[0.45,0.2169,0.60283],[0.5,0.24983,0.717
        0.55,0.2889,0.84892],[0.6,0.33507,1.0016],[0.65,0.38947,1.179],[0.7,0.45344,1
        [0.75,0.52856,1.6256],[0.8,0.61667,1.9059],[0.85,0.71993,2.233],[0.9,0.84089,
        [0.95,0.9825,3.0613],[1.0,1.1483,3.5828]]
```

The returned values coincide with the tabulated values of the Basic program by ROMER.
The ropes position at $t = 0.5$ [sec] is ca. $x = 0.25$ [m].

ad **b.** : left as exercise.

ad **c.** : left as exercise.

**Exercises.**

**Exercise 124.** Reduction of Higher order Equations ▷ HELLEVIK: ivp.

**Exercise 125.** Solve the IVP system $y' - 2y = -3z;\ z' = y - 2z,\ \begin{smallmatrix} y(0)=0 \\ y'(0)=1 \end{smallmatrix}$.
Use different numerical solving methods.

**Exercise 126.** Solve $y'' + \frac{5}{x}y' = -y$ of $2^{nd}$ order with the initial conditions $\begin{smallmatrix} y(0)=0 \\ y'(1)=0 \end{smallmatrix}$.

**Exercise 127.** Solve $y'' + 6y' + 8y = 4x + 3e^{-x}$ given $\begin{smallmatrix} y(0)=0 \\ y'(0)=4 \end{smallmatrix}$ to find $y(1)$ correct to six decimal places, cf. [29, p.284]. [Result: $y(1) = 0.598451$ to 6D.
Determine the exact analytical solution $y_e(x)$ and compare with the approximative solution.

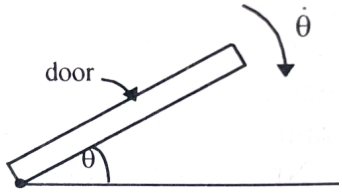
**Exercise 128.** Solve the system of $1^{st}$ order IVPs $\frac{dx}{dt} - x - 5y = 18t;\ \frac{dy}{dt} + 2x + y = 9$, given $x(0) = 4, y(0) = -1$, cf. [29, p.287].

**Exercise 129.** Solve the system $\{\frac{dx}{dt} - 2x - 3y = 3\cos t;\ \frac{dy}{dt} + x + 2y = \sin t\}$,
given $x(0) = -1, y(0) = 1$, cf. [29, p.287].

**Exercise 130.** Express the equation $x^2y'' + 7xy' - 7y = 14\ln x + 2$ as a pair of simultaneous $1^{st}$ order equations. Given $x(0) = -1, y(0) = 1$, find $y(2)$ correct to six decimal places, cf. [29, p.287]. Determine the exact analytical solution $y_e(x)$ and compare with the approximative solution. [Result: $y(2) \approx 5.617612$ to 6D.

**Exercise 131.** (swinging door, cf. [29, p.328])

> LOWE/BERRY *present many solved applications of second order and simultaneous first order differential equations on about 60 pages. There is also a modeling approach to differential equations. We quote here a sample:*




A swinging door has a damping device so that the mathematical model for the angular displacement $\theta(t)$ is

$$I\ddot{\theta} = -a\theta - b\dot{\theta}$$

where $I, a$ and $b$ are constants. Consider a system for which $I = 1.5$ kg s$^2$ m$^{-1}$, $a = 6$kg m$^{-1}$, $b = 7.5$ kg s m$^{-1}$ and for which the initial conditions are $\Theta(0) = \pi/3$ and $\dot{\Theta}(0) = 0$.

**a.** Formulate and solve an initial value problem that models this system.

**b.** Draw a graph of the displacement $\Theta$ with time.

**c.** What does your model predict as $t$ becomes large?

**Exercise 132.** In fig.13*Left* we got the ODE of the pendulum through the equation

$$-m\ell\ddot{\varphi} - mg\sin\varphi = 0$$

We measure the length of the pendulum so, that $\frac{g}{\ell} = 1$. For small deviations of $\varphi$ we may use the first 2 terms of the Taylor series of $\sin\varphi \approx \varphi - \frac{\varphi^3}{6}$ and arrive at the simplified ODE

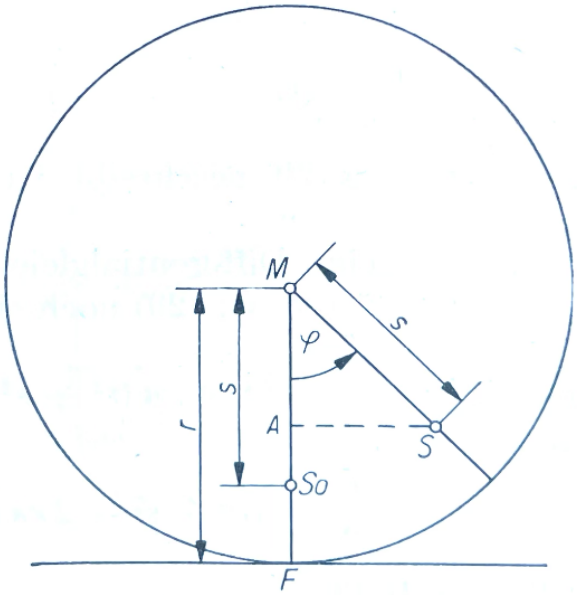
$$\ddot{\varphi} = -\tfrac{g}{\ell}(\varphi - \tfrac{1}{6}\varphi^3) \qquad (\ddagger)$$

**a.** Solve eq. ‡ for the IC $\varphi(0) = 0, \dot{\varphi}(0) = 2$ with the methods
– successive approximation
– `RK4sys`, `RK4ode2`, `RK4romer` and `rk`$^{Maxima}$.
and compare the methods. Plot the solutions.

**b.** Solve the original not simplified $2^{nd}$-order ODE: $\ddot{\varphi} = -\sin\varphi$ with $\varphi(0) = 0$ and $\dot{\varphi}(0) = 2$. (use: $h = 0.2$)

**c.** Solve the linearized and therefore simplified eq.‡: $\ddot{\varphi} = -\varphi$ with $\varphi(0) = 0$ and $\dot{\varphi}(0) = 2$.

**Exercise 133.** (rolling pendulum, cf. Bräuning, [12, p.27])




A circular cylinder with mass $m$ and radius $r$, whose center of gravity $S$ is a small distance $s(0 < s < r)$ from the central axis, lies on a horizontal plane. The cylinder is deflected from its rest position by an angle $\varphi$ and then left to its own movement. It oscillates back and forth (neglecting damping, closed physical system). – The physical investigation of the oscillation process of this roll pendulum leads to a second-order differential equation:

$$[\frac{\Theta_s}{m} + (r^2 + s^2 - 2rs\cos\varphi)]\ddot{\varphi} + (r\dot{\varphi}^2 + g)s\sin\varphi = 0 \qquad (\dagger)$$

where $\Theta_s$ is the moment of inertia.

**a.** Using special values for $m, \Theta_s, r, s$ we get the specialized ODE:

$$(8 - 2\cos\varphi)\ddot{\varphi} + (\dot{\varphi}^2 + 2)\sin\varphi = 0 \qquad (\dagger\dagger)$$

Solve (††) for the IC $\varphi(0) = \frac{\pi}{2},\ \dot{\varphi}(0) = 0$, i.e. the pendulum is deflected just far enough, that its center of gravity is at the height of the central axis and then left to its own motion.
[Result: we get for small values e.g. $\varphi(0.5) \approx 1.5395$ and $\dot{\varphi}(0.5) \approx -1.2564$.

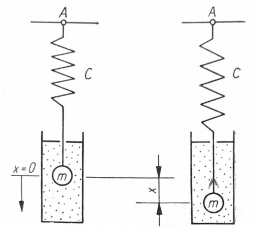
**b.** Verify, that the corresponding system of simultaneous IVP's for ‡ is

$$\begin{aligned} y' &= z \\ z' &= -\frac{z^2+2}{8-2\cos y} \cdot \sin y \\ & y(0) = \frac{\pi}{2},\ z(0) = 0 \end{aligned}$$

Use $h = 0.2$ and calculate the motion over the time interval $-0.2 \le x \le 2$.
[Control: $(x, Y, Z)\,|_{2.0} \approx$ (2.0, 1.0530, −0.5314)

**c.** Redo this exercise using built-in function `rk`.

**Exercise 134.** (swinging spring, cf. Bräuning, [12, p.21 ff])

Look again at fig.13right: 

.

The physical investigation of the oscillation process of this spring leads to a second-order differential equation:

$$-m\ddot{x} - k\dot{x} - cx = 0$$

where $k \in \mathbb{R}$ a constant w.r.t damping. Using the shortcuts $\frac{k}{m} =: 2d$ and $\frac{c}{m} =: \omega_0^2$, we arrive at the 2nd order IVP

$$\ddot{x} + 2d\dot{x} + \omega_0^2 \cdot x = 0$$

Discuss the motion of the swinging spring for $d := 1/2$ and $\omega_0^2 := 1$.
Chose different IC's. Interpret your choice.
Use different methods and also the built-in function `rk`.

**Remark.**
1. Woollett discuss in his publication *Maxima by Example: Ch. 3, Ordinary Differential Equation Tools*, see p.24 ff in §3.4.4 in great detail the linear oscillator with damping and in §3.4.6 on p.30 ff. the motion of a driven damped planar pendulum. He make primarily use of build-in analytic tools like `desolve` etc. I will recommend the study of his work especially for the physicist.

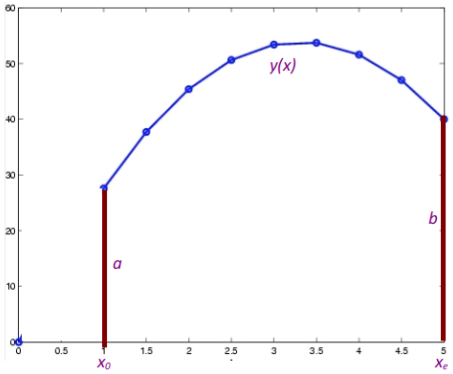
2. Timberlake [48, p.97 ff] discuss the damped harmonic oscillator and the pendulum on p.115 ff. He also make use of analytical methods like Maxima's `desolve`.

# 6 Boundary Value Problems - Numerical Methods

We presented several methods of approximating solutions to an initial value problem (IVP). Now we study approximative solutions to a boundary value problem (BVP). Such a problem consists of a differential equation together with two boundary conditions imposed at each end of the interval of definition.
We devote to methods of approximating the solution to a 2nd order boundary value problem of the form

$$y'' = f(x, y, y') \quad \text{with} \quad y(x_o) = a,\ y(x_e) = b \tag{6.1}$$

The solution $y(x)$ to a BVP (6.1) may describe a physical situation, such as the height of a construction at different points and supported by posts at $x = x_o$ and $x = x_e$. The boundary conditions (BC) $y(x_o) = a$ and $y(x_e) = b$ give the height of the construction at each post at the boundary. We are only interested in the solution *between* the posts, because there is no construction outside this interval:




## 6.1 Shooting Method

The theoretical idea of the shooting method of approximating the solution $y$ of a second order boundary value problem of the form $y'' = f(x, y, y')$ with boundary conditions (BC) $y(x_o) = a,\ y(x_e) = b$ is:

**1**: *transform* the 2nd order BVP into one system of two 1st order IVP's with initial conditions (IC) using a *unknown* $w$ by

| ODE | *BVP* | IVP |
|---|---|---|
| *differential eq.* | $y'' = f(x, y, y')$ | $y' = z \ \wedge \ z' = f(x, y, z)$ |
| BC: | $y(x_o) = a,\ y(x_e) = b$ | |
| IC: | | $y(x_o) = a,\ z(x_0) = w = ?$ |
| | | with $w$ so, that $y(x_e) \overset{!}{=} b$ |

**2**: *define* helper function $F(w) :=$ (approx. solution to IVP at $x = x_e$) - b
**3**: *solve* $F(w) = 0$, i.e. search for $w^*$ with $F(w^*) = 0$.
**4**: *put* $w^*$ back in IVP to solve the BVP.

Let's look at the idea in a figure, modified from ▷ Heckbert: bvp:

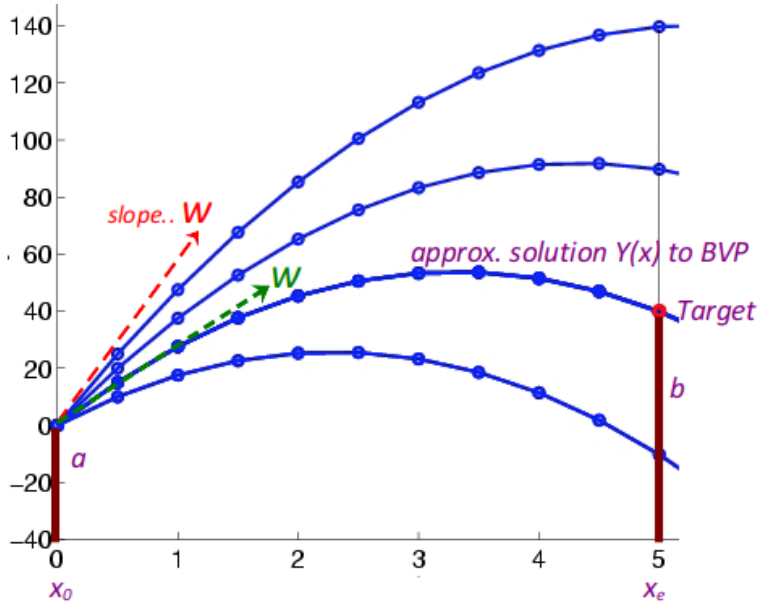



Figure 19: *Idea*: a projectile is shooted from start position $(0,0)$ at the target • $(5,40)$ with different slopes $w_k$. The winning slope $w$ and its corresponding trajectory is sandwiched between three trial shoots • – •. The BVP $y'' = f(x,y,y')$ with BC $y(x_o) = a,\ y(x_e) = b$ is approximately solved with `RK4` and a bisection of the trials $w_k$.

We now follow the shooting method in an example, see [43, p.403].

### 6.1.1 Solve BVP $y'' = y + \sin(x + y'),\ y(0) = 1.2,\ y(3) = 2.4$ by shooting.

We follow the 4-step plan of the shooting method. We use MAXIMA.
**1**: *transform* $2^{nd}$ order BVP $\rightsquigarrow$ system of two $1^{st}$ order IVP's
**2**: *define* function $F(w) :=$ (approx. solution to IVP at $x = x_e$) - b
**3**: *solve* $F(w) = 0$
**4**: *put* $w$ back in IVP (solves original BVP).

*Solution*:
ad **1**: transform the 2nd order BVP into two 1st order IVP's:

| Translate | *BVP* | IVP |
|---|---|---|
| *differential eq.* | $y'' = y + \sin(x + y')$ | $y' = z \ \wedge\ z' = y + \sin(x + y')$ |
| BC1: | $y(0) = 1.2$ | |
| BC2: | $y(3) = 2.4$ | |
| IC1: | | $y(0) = 1.2$ |
| IC2: | | $z(w) = 2.4$ with $w$ so, that $y(3) \stackrel{!}{=} 2.4$ ! |

We fire two shots with guess $w = 1$ and $w = -1$:

```
fpprintprec:5$
ratprint:false$
f(x,y,z):=z;
g(x,y,z):=y+sin(x+z);
RK4sys(x,y,z, 0,1.2, 1,   3/16, 16);
```

▷ Click to RUN the code.

```
(%o13)  f(x,y,z):=z
(%o14)  g(x,y,z):=y+sin(x+z)
(%o15)  [[0,1.2,1],[0.1875,1.426,1.4227],[0.375,1.7363,1.8898],[0.5625,2.134,2.3479]
        ,[0.75,2.6149,2.7772],[0.9375,3.1751,3.2013],[1.125,3.8185,3.6771],[1.3125,4.563,
        4.299],[1.5,5.4485,5.2073],[1.6875,6.5396,6.4817],[1.875,7.8881,7.9035],[2.0625,9
        ,9.3759],[2.25,11.434,11.312],[2.4375,13.782,13.773],[2.625,16.609,16.487],[2.812
        20.023,20.006],[3.0,24.141,24.102]]
```

```
last(RK4sys(x,y,z, 0,1.2, -1,   3/16, 16))[2];
```

▷ Click to RUN the code.

```
(%o50)  1.4358
```

ad **2**: define helper function $F(w) :=$ (approx. solution to IVP at $x = x_e$) - b

```
fpprintprec:5$
ratprint:false$
f(x,y,z):=z;
g(x,y,z):=y+sin(x+z);
F(w):= last(RK4sys(x,y,z, 0,1.2,w,   3/16, 16))[2] - 2.4;
```

▷ Click to RUN the code.

There is no output - don't be disappointed: MAXIMA has only learned and accepted this definition.

ad **3**: solve $F(w) = 0$, i.e. search for $w^*$ with $F(w^*) = 0$.

• First: Let's do the built-in function `find_root` the search for us:

```
find_root(F,w,-1,0);
```

▷ Click to RUN the code.

```
find_root(F,w,-1,0);
(%o51)  -0.9369
```

Result: the optimal slope for the best shot is $w^* = -0.9369$.

• Second: Let's write a homemade *bisection search method*, named `bisec`, and use it for searching the root $w$, cf. [48, p.222]:

```
bisec(F,xL,xR, tol,n) := block(
   if    (F(xL)*F(xR) > 0) then
         print("Sign does not change within interval.")
   else  for i:1 while ((i<100) and (abs(xR-xL)>tol)) do
             (xM:(xL+xR)/2,
              if   (F(xM)*F(xR) > 0)
              then xR:xM
              else xL:xM, n:n+1 , print(n, float(xM)) ,
    xM  ))$

bisec(F, xL:-1, xR:0, tol:0.00001, n:0);
```

▷ Click to RUN the code.

```
1  -0.5
2  -0.75
3  -0.875
4  -0.9375
5  -0.90625
6  -0.92188
7  -0.92969
8  -0.93359
9  -0.93555
10  -0.93652
11  -0.93701
12  -0.93677
13  -0.93689
14  -0.93695
15  -0.93692
16  -0.9369
17  -0.9369
(%o52)  done
```

ad **4**: put $w^*$ back in IVP to solve the BVP.

```
RK4sys(x,y,z, 0,1.2, -0.9369,   3/16, 16);
```

▷ Click to RUN the code.

```
(%o53)  [[0,1.2,-0.9369],[0.1875,1.0312,-0.86288],[0.375,0.87694,-0.78061],[0.5625,
        0.73978,-0.67787],[0.75,0.62478,-0.54234],[0.9375,0.53913,-0.3634],[1.125,0.4915,
        0.1369],[1.3125,0.49025,0.12841],[1.5,0.54063,0.40899],[1.6875,0.64275,0.67587],[
        1.875,0.79196,0.90926],[2.0625,0.98136,1.1051],[2.25,1.2045,1.2716],[2.4375,1.4573
        1.4231],[2.625,1.7384,1.5774],[2.8125,2.0503,1.7557],[3.0,2.4,1.9864]]
```

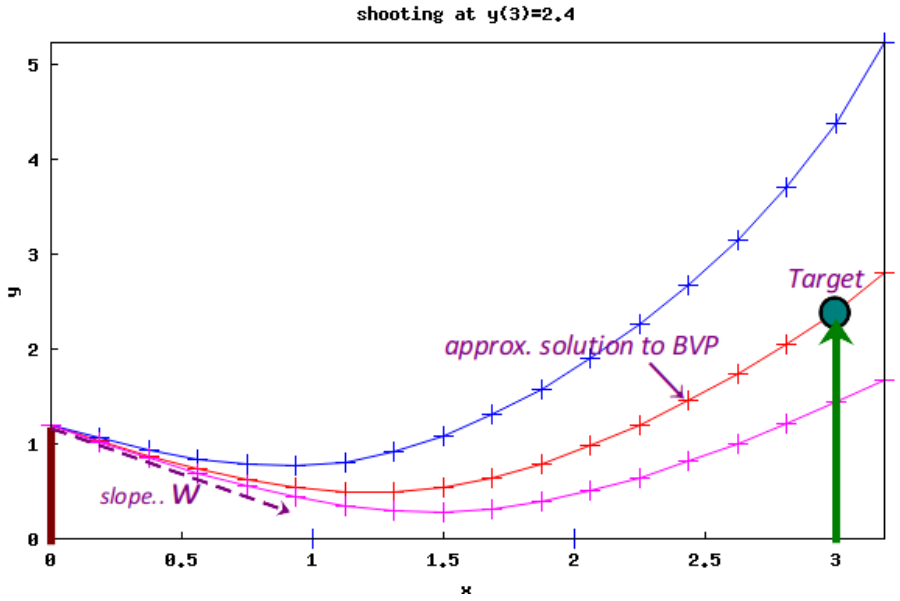
Alternatively we may also use our specialized function RK4romer for this 2nd order IVP:

```
f(x,y,yp):=y+sin(x+yp);
RK4romer(x,y,yp, 0,1.2,-0.9369, 3/16, 16)
```

▷ Click to RUN the code.

```
(%o40) [[0,1.2,-0.9369],[0.1875,1.0312,-0.86288],[0.375,0.87694,-0.78061],[0.5625,
       0.73979,-0.67786],[0.75,0.62479,-0.54232],[0.9375,0.53914,-0.36337],[1.125,0.4915
       0.13684],[1.3125,0.49028,0.12848],[1.5,0.54068,0.40909],[1.6875,0.64281,0.67599],
       1.875,0.79204,0.90938],[2.0625,0.98146,1.1052],[2.25,1.2047,1.2717],[2.4375,1.4574
       1.4233],[2.625,1.7386,1.5776],[2.8125,2.0505,1.7559],[3.0,2.4002,1.9866]]
```

Result: in both cases the RKsys functions produce the correct BC `[3.0,2.4]`.
We check the plausibility of the solution by a plot of the situation:




```
f(x,y,yp):=y+sin(x+yp);
wxdraw2d(xaxis = true,
    point_size=2,
    points_joined=true,
    points([0,0]),
    xlabel="x",ylabel="y",
    color=red,
    points( RK4romer(x,y,yp, 0,1.2,-0.93689, 3/16, 16+1) ),
    color=blue,
    points( RK4romer(x,y,yp, 0,1.2, -0.8,       3/16, 16+1) ),
    color=magenta,
    points( RK4romer(x,y,yp, 0,1.2, -1,       3/16, 16+1) ),
  title="shooting at y(3)=2.4")$
```

▷ Click to RUN the code.

**Exercise 135.** **a.** What is the (approximate) value of $y(0.75)$?
**b.** What is the (approximate) value of $y'(0.8)$?
**c.** Verify the results of a. and b. with your ruler on the figure.

### 6.1.2 Example: *solving a BVP via shooting*

Solve the BVP $y'' = \frac{y}{1+x^2} + \frac{y'}{10}$ with BC: $y(0) = 1,\ y(2) = 3$. cf. [43, p.398].

*Solution.*
This time we do the solution in one step: search for $w$ using $F$ and `find_root`.
Therefore we use the equivalent system $(f, g)$ of 2 IVPs $\equiv$ 1 BVP.

```
/* MAXIMA --- SHOOTING METHOD for BVP --- */
fpprintprec:5$
ratprint:false$
f(x,y,z) := z;  /* = y'  */
g(x,y,z) := y/(1+x^2)+z/10;
F(w):= last( RK4sys(x,y,z, 0,1,w,   1/8, 16))[2] - 3;
find_root(F,w,0,1);
```

▷ Click to RUN the code.

```
(%o80)  f(x,y,z):=z
(%o81)  g(x,y,z):=y/(1+x^2)+z/10
(%o82)  F(w):=last(RK4sys(x,y,z,0,1,w,1/8,16))[2]-3
(%o83)  0.058068
```

We get $w = 0.58068$. Check:

```
f(x,y,yp):=y/(1+x^2)+yp/10;
RK4romer(x,y,yp, 0,1, 0.058068, 0.1, 20);
```

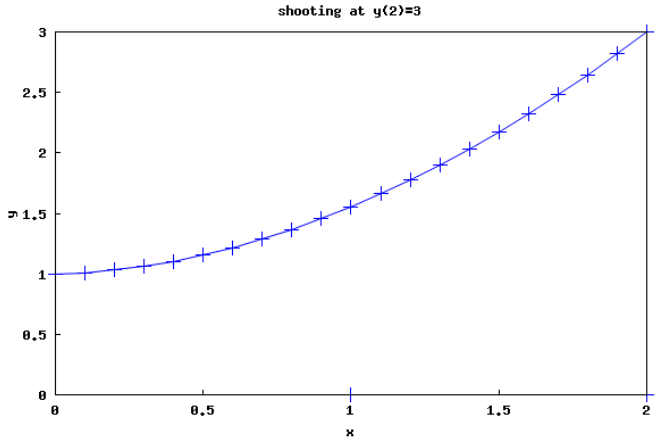
▷ Click to RUN the code.

```
(%o76)  f(x,y,yp):=y/(1+x^2)+yp/10
(%o77)  [[0,1,0.058068],[0.1,1.0109,0.15928],[0.2,1.0319,0.26111],[0.3,1.0631,
        0.36266],[0.4,1.1044,0.4632],[0.5,1.1557,0.56219],[0.6,1.2167,0.65927],[0.7,1.287
        0.75428],[0.8,1.3675,0.84717],[0.9,1.4568,0.93803],[1.0,1.5551,1.027],[1.1,1.6621
        1.1141],[1.2,1.7778,1.1997],[1.3,1.902,1.2839],[1.4,2.0346,1.3669],[1.5,2.1754,
        1.4488],[1.6,2.3243,1.5298],[1.7,2.4813,1.61],[1.8,2.6463,1.6896],[1.9,2.8192,1.7
        ],[2.0,3.0,1.8473]]
```

Result: we see in the last slot $y(2) = 3$ and $y'(2) = 1.8473$ for the given BVP. Ok.
Let us plot the approximate solution $y(x)$:

```
f(x,y,yp):=y/(1+x^2)+yp/10;
wxdraw2d(xaxis = true,
    point_size=2,
    points_joined=true,
    points([0,0]),
    xlabel="x",ylabel="y",
    points( RK4romer(x,y,yp, 0,1, 0.058068, 0.1, 20) ),
    color=red,
    title="shooting at y(2)=3")$
```

▷ Click to RUN the code.




**Exercise 136.** **a.** What is the (approximate) value of $y(0.75)$?
**b.** What is the (approximate) value of $y'(1)$?
**c.** Verify the results of a. and b. with your ruler on the figure.

**Exercises.**

**Exercise 137.** HECKBERT ▷ bvp gives the following procedural Python code for the secant method to determine roots of a function. Write a MAXIMA function `secant` and use it in examples 6.1.1 and 6.1.2 instead of `find_root` and `bisec`.

```
%% PYTHON code -- rocket = y
function x = secant(x1,x2,tol)
% secant method for one-dimensional root finding
global ye;
y1 = rocket(x1)-ye;
y2 = rocket(x2)-ye;
while abs(x2-x1)>tol
      disp(sprintf('(%g,%g) (%g,%g)', x1, y1, x2, y2));
      x3 = x2-y2*(x2-x1)/(y2-y1);
      y3 = rocket(x3)-ye;
      x1 = x2;
      y1 = y2;
      x2 = x3;
      y2 = y3;
end
x = x2;
return;
```

▷*For each of the following BVP, use* `RKsys` *with* $n = 8$ *steps in conjunction with* `find_root` *or* `bisec` *to approximate the solution function* $y(x)$ *and answer the required questions. Check for plausibility wit* `RKromer` *and do a plot of the scene.*

**Exercise 138.** Solve the BVP: $y'' = 2y^3$ with BC: $y(1) = 1/4$, $y(3) = 1/6$, cf. [43, p.403].
**a.** What is the (approximate) value of $y(1.5)$?
**b.** What is the (approximate) value of $y'(1.25)$?
**c.** The exact solution to the BVP is $y_e(x) = \frac{1}{x+3}$ on the interval $1 \le x \le 3$.
How do your approximate values in a. and b. compare with the exact values?
**d.** Plot the approximate solution function $y$ and the exact solution $y_e$ and check the results on the graph.

**Exercise 139.** Solve $y'' = y^2 + y' + \frac{2}{x^3} - 3 - x^2$ with $y(1) = 2$, $y(2) = \frac{5}{2}$, cf. [43, p.403].
**a.** What is the (approximate) value of $y(1.75)$?
**b.** What is the (approximate) value of $y'(1.75)$?
**c.** The exact solution to the BVP is $y_e(x) = x + \frac{1}{x}$ on the interval $1 \le x \le 2$.
How do your approximate values in a. and b. compare with the exact values?
**d.** Plot the approximate solution function $y$ and the exact solution $y_e$ and check the results on the graph.

**Exercise 140.** (Bulirsch–Stoer, [46, p.156], §7.3.1) Solve the BVP $y''(x) = \frac{3}{2}y^2$ with $y(0) = 4,\ y(1) = 1$ approximately using `RK4sys` or `RK4romer`.
The graph of $F(w) := ... - 1$ has two roots, the first one is $w_1 = -8$ and has the exact solution $y(x) = \frac{4}{1-x^2}$.
Do a survey analog to exercise 138. Cf. ▷ wiki: Example: Standard boundary value problem.

**Exercise 141.** (Bulirsch–Stoer, [46, p.167], §7.3.4) Solve the

$$\text{BVP}: \begin{cases} y'' = \lambda \cdot \sinh(\lambda y) \quad (\lambda \in \mathbb{R}) \\ y(0) = 0 \\ y(0) = 0 \end{cases}$$

This BVP makes problems. Try to get the solution $4.5750 \cdot 10^{-2}$ for $\lambda = 5$.

**Exercise 142.** (Burden–Faires, [10, p.582]) **a.** Solve the

$$\text{BVP}: \begin{cases} y'' = -\frac{2}{x}y' + \frac{2}{x^2}y + \frac{\sin(\ln x)}{x^2} \\ y(1) = 1 \\ y(2) = -2 \end{cases} \qquad \text{on the interval} \quad 1 \le x \le 2.$$

**b.** Do also example 1 ▷ Barannyk

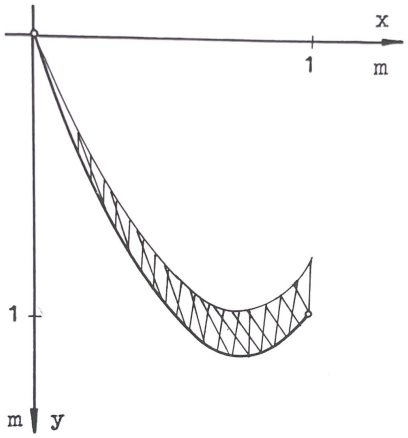
**Exercise 143.** (Borse, [7, p.459]) Solve the

$$\text{BVP}: \begin{cases} (y-1)^2 y'' + 2(y-1)(y')^2 = 0 \\ y(0) = 2 \\ y(1) = 3 \end{cases}$$

by the shooting method and compare with the exact solution $y_e(x) = 1 + (7x+1)^{1/3}$.

**Exercise 144.** (BORSE, [7, p.459]) Solve the BVP $y'' + y = 2e^{-x}$, $y(0) = 0$, $y(\frac{\pi}{2}) = 0$ approximately using the shooting method and compare with the exact solution $y_e(x) = -e^{-\pi/2}\sin(x) - \cos(x) + e^{-x}$.

**Exercise 145.** (VENZ, [49, p.49]) The physical theory of a sagly rope leads to the differential equation $S \cdot y'' = -q(x)$, where $S$ is the traction in [N] and $q$ the load per unit of length [N/m]. On the rope lie a snow load of variable size:

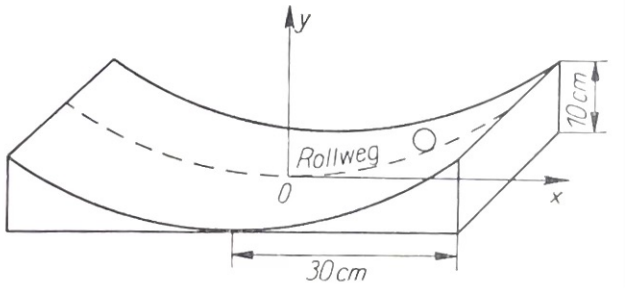



We approximate the load by the function $q = q_0 \cdot \sin(\frac{\pi}{2} \cdot \frac{x}{x_b})$.
Then the differential equation is

$$y'' = \frac{q_0}{S} \cdot \sin(\frac{\pi}{2} \cdot \frac{x}{x_b})$$

Calculate the rope's sag function $y(x)$, where we have $x_a = 0$ m, $y(x_a) = 0$ m and $x_b = 1$ m, $y(x_b) = 1$ m and $\frac{q_0}{S} = 6$ [1/m]. Use a step size of $h = 0.125$ m.

**Exercise 146.** (BRÄUNING, [12, p.27]) We let a ball roll on a fall line of a parabolic cylinder, where the dimensions can be taken from the picture. We measure $x$ and $y$ in centimeters, so the path has the equation $y = \frac{1}{90}x^2$. We use the value $g = 981$ $[cm/s^2]$ for the gravitational acceleration.




LEXICON: *German* | English
Rollweg | path of contact point

Then one gets – for physical reasons[17] – the differential equation of the motion

$$\ddot{x} = -\frac{x}{2025 + x^2} \cdot (31532 + \dot{x})$$

[17] The interested reader can get the derivation (in German) of the ODE on request from the author.

Calculate the motion $x(t)$ of the ball for $x(0) = 30$ cm and $x'(0) = 0$ cm.
Use a shooting method with a step size of $h = 0.05$.

**Exercise 147.** Do the examples from ▷ Grothmann: Shooting ..
>Documentation>Examples>All Examples>Shooting method for boundary Problems
and at ...>Singular Boundary Value Problem.

**Exercise 148.** Do the examples from ▷ iron: bvp 1, i.e. solve the BVP $y'' = -\frac{(y')^2}{y}$ with BC $y(0) = 1,\ y(1) = 2$.

**Exercise 149.** Do the ▷ Heckbert: rocket problem. using our Shooting method.

**Exercise 150.** Do the BVP's in ▷ Niemeyer: Shooting method. using our Shooting method.

**Exercise 151.** Do the BVP's ▷ Dorine: Shooting method. on the deflection of a supported beam with a constant distributed load.

**Exercise 152.** Do the 3 BVP's ▷ Arefin et al.: Shooting methods. in: Analysis of Reliable Solutions to the Boundary Value Problems by Using Shooting Method

**Exercise 153.** Do the BVP examples on Shooting by ▷ Uni Muenster, DE: Shooting methods.

**Exercise 154.** Do the BVP's ▷ Berkeley Python Numerical Methods: Shooting methods.

**Exercise 155.** Have a look at the BVP examples in ▷ Verschelde 2022: Shooting.

**Exercise 156.** Look at the BVP's ▷ Hellevik: NM 4 Engineers: Shooting methods. and do a few of the examples as you like

**a.** ▷ : Couette-Poiseuille flow.
**b.** ▷ : Simply supported beam.
**c.** ▷ : boundary value problems with nonlinear ODEs.
**d.** ▷ : Large deflection of a cantilever.
**e.** ▷ : Stokes first problem.

Here are more infos and examples:

**Exercise 157.** ▷ Kumar: Shooting methods.

**Exercise 158.** ▷ Vesely: Shooting methods.

**Exercise 159.** ▷ Pardyjak: Shooting methods.

**Exercise 160.** ▷ Numerical Methods for Engineers: Shooting methods.

## 6.2 Finite Difference Method

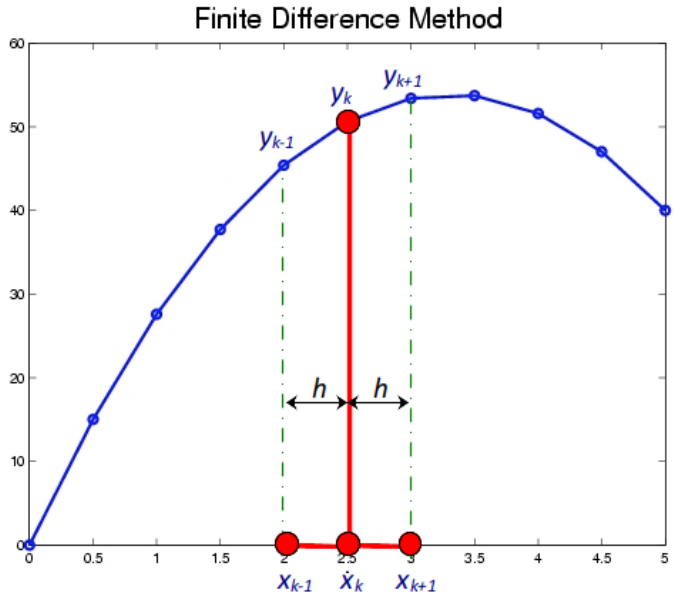



Figure 20: *Idea*: The approximative values $y_k$ of the solution $y$ to the BVP $y'' = f(x, y, y')$ of 2nd order are calculated by replacing the derivatives $y''$ and $y'$ by so-called *central difference approximations*. The 4-point '*stencil*' $x_{k-1}$ $\overset{y_k}{x_k}$ $x_{k+1}$ traverse from the first BC1: $x_0 = 0, y_0 = 0$ on the left to the second BC2: $x_n = 5, y_n = 40$ on the right in steps of length $h$, producing a new equation in each step.

The theoretical idea of the *Finite Difference Method* (FDM) of approximating the solution $y$ of a second order boundary value problem of the form $y'' = f(x, y, y')$ with boundary conditions BC: $\begin{smallmatrix} y(x_o)=a \\ y(x_e)=b \end{smallmatrix}$ is:

**1**: *replace* the first and second derivatives $y', y''$ in the BVP equation $y'' = f(x, y, y')$ by approximating values ('central differences') at each point of the constructed grid $(x_0, x_1, x_2, ..., x_e)$, where $x_k := x_0 + k \cdot h$ and $h := \frac{x_0 - x_e}{n}$ for a user chosen $n \in \mathbb{N}$, i.e.:

| BVP: | *equation* | substitute by for $k = 1, 2, , ..., n-1$ |
|---|---|---|
| | $y'' = f(x, y, y')$ | $y_k$ : approximate value of the BVP solution at $x_k$ |
| | $y'(x_k) \approx$ | $yp_k := \frac{y_{k+1} - y_{k-1}}{2h}$ |
| | $y''(x_k) \approx$ | $ypp_k := \frac{y_{k+1} - 2 \cdot y_k + y_{k+1}}{h^2}$ |

**2**: *write down* the corresponding $\text{BVP}_k$ equation for each interior point (stencil) $(x_k, y_k)$
**3**: *include* the BC equations $y_0 := a, y_n := b$ to get now a linear system of $n+1$ equations for the $n+1$ unknowns $y_0, y_1, ..., y_n$
**4**: *solve* this *tridiagonal* system of linear equations using i.e. `rref` or `linsolve` or ... . STOP.

We now follow the Finite Difference Method (FDM) in an example, see [43, p.407]. We do one step after the other in this procedure to *semi*-automate the solution process.

### 6.2.1 Solve BVP $y'' + 0.2y' + 4y = 3x + 1,\ y(0) = 0.1,\ y(1) = 0.7$ for $n = 4$.

We fix $n = 4$.
We follow the 4-step plan of the FDM. We use MAXIMA.
**1**: *replace* the first and second derivatives in the BVP equation by their approximations
**2**: *write down* the corresponding $\text{BVP}_k$ equations
**3**: *include* the BC equations
**4**: *solve* this *tridiagonal* system.

*Solution*:
ad **1**: we define the set of unknowns $y_k$ and the substitutions[18] for $y', y''$ in MAXIMA:

```
kill(y)$ fpprintprec:5$ ratprint:false$

y        :  makelist (concat (y,i), i,0, 4);              /* (0) */
yp(k,h)  := (y[k+2]-y[k])/(2*h);                          /* (1) */
ypp(k,h) := (y[k]-2*y[k+1]+y[k+2])/h^2;
x(k)     := xo+k*h;
FDM(k,h) := ypp(k,h) + 0.2*yp(k,h) + 4*y[k+1] = 3*x(k)-1;  /* (2) */
/*          y''      + 0.2 y'      + 4 y      = 3x-1   */
```

▷ Click to RUN the code.
Herein `FDM(k,h)` represents the corresponding $\text{BVP}_k$ equation $y'' + 0.2y' + 4y = 3x + 1$ (see 2:) and `yp` and `ypp` the derivatives (see 1:).
ad **2**: *write down* the corresponding $\text{BVP}_{k=1,2,3}$ equations, named `eq[i]` in MAXIMA :

```
n:4$ xo:0$ xe:1$
h: (xe-xo)/n;
eq[1]: FDM(1,h), expand, numer;
eq[2]: FDM(2,h), expand, numer;
eq[3]: FDM(3,h), expand, numer;
```

▷ Click to RUN the code.

```
(h)    1/4
(%o13) 16.4 y2-28.0 y1+15.6 y0=-0.25
(%o14) 16.4 y3-28.0 y2+15.6 y1=0.5
(%o15) 16.4 y4-28.0 y3+15.6 y2=1.25
```

[18] *yp* means yprime i.e. $y'$ and $ypp \equiv y''$.

ad **3**: *include* the BC equations `y0`=$y(0) = 0.1$, `y4` =$y(1) = 0.7$ into the eq. list $M$:

```
M: flatten([y0=0.1, makelist(eq[i], i,1,3), y4=0.7]);
```

*Comment*: MAXIMA's `makelist` returns a list of the equations $eq_k$ in the form [[..], [..], ..], therefore we use the MAXIMA function `flatten(.)` to kill the interior brackets [..] and to get a comma separated list of the $\mathrm{BVP}_k$ equations:

▷ Click to RUN the code.

```
(m)  [y0=0.1,16.4*y2-28.0*y1+15.6*y0=-0.25,16.4*y3-28.0*y2+15.6*y1=0.5,16.4*y4-28.0
     *y3+15.6*y2=1.25,y4=0.7]
```

ad **4**: *solve* this *tridiagonal* system. We use a homemade routine named `rref`[19] to solve the system.

```
rref(a):=block([p,q,k], [p,q]:matrix_size(a), a:echelon(a), k:min(p,q),
    for i thru min(p,q) do (if a[i,i]=0 then (k:i-1, return())),
    for i:k thru 2 step -1 do
            (for j from i-1 thru 1 step -1 do a: rowop(a,j,i,a[j,i])),
    a)$

Am: augcoefmatrix(M, [y0,y1,y2,y3,y4]);      /* (1) */
rref(Am), numer;                             /* (2) */
Y: -col(rref(Am), 6), numer;                 /* (3) */
```

▷ Click to RUN the code.

```
(%i24)  rref(Am), numer;

        [ 1  0    0    0    0    -0.1     ]
        [ 0  1    0.0  0.0  0.0  -0.45611 ]
(%o24)  [ 0  0    1    0.0  0.0  -0.66836 ]
        [ 0  0    0    1    0.0  -0.73773 ]
        [ 0  0    0    0    1    -0.7     ]
```

$$(Am)\quad \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & -0.1 \\ \frac{78}{5} & -28 & \frac{82}{5} & 0 & 0 & 0.25 \\ 0 & \frac{78}{5} & -28 & \frac{82}{5} & 0 & -0.5 \\ 0 & 0 & \frac{78}{5} & -28 & \frac{82}{5} & -1.25 \\ 0 & 0 & 0 & 0 & 1 & -0.7 \end{pmatrix}$$

```
(%i26)  -col(rref(Am), 6), numer;

        [ 0.1     ]
        [ 0.45611 ]
(%o26)  [ 0.66836 ]
        [ 0.73773 ]
        [ 0.7     ]
```

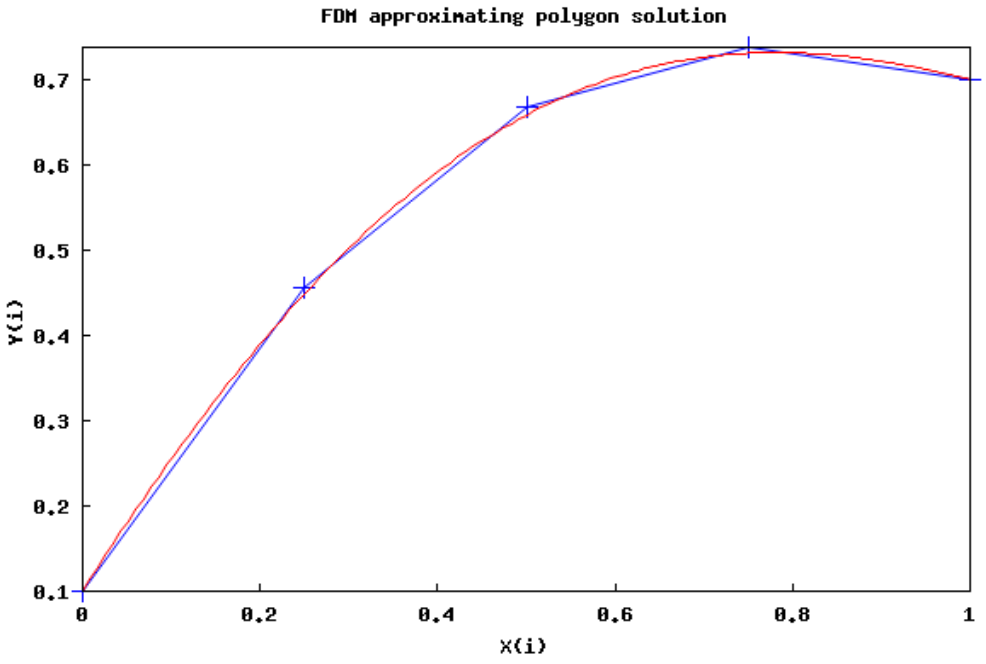
[19] `rref` i.e. Row Reduced Echelon Form; this script is found at ▷ stackoverflow: `rref`

*Comment*: Looking at the augmented coefficient matrix `(Am)` of our system of the equations $eq_k$ in (1), we observe the *tridiagonal* shape of the system. The MAXIMA function `augcoefmatrix` returns the augmented coefficient matrix for the variables $x_0, .., x_n$ of the system of linear equations $eq_0, .., eq_4$. This is the coefficient matrix of the system with a column adjoined for the constant terms on the RHS in each equation.
*Warning*: `Am` produces wrong signs on the RHS of the system `m`! Therefore the same is true for the result of `rref` in (2) and we have to invert the signs to get the correct solution vector `Y` in (3) resp. in `(%o26)`.

**4**: *plot* the result. The complicated exact solution $y_e(x)$ of the BVP is given in [43, p.409]:

```
ye(x):= exp(-x/10)*( 31/80*cos(sqrt(399)*x/10)
                    + 0.464502*sin(sqrt(399)*x/10)) + 3*x/4 - 23/80;

X: makelist(x(k), k,0,4);
Y: flatten (makelist (makelist (Y[i], i, 5)));
XY: makelist([ X[i],Y[i]] ,i,1,n+1);
wxdraw2d(xaxis = true,
    point_size=2,
    points_joined=true,
    xlabel="X(i)",ylabel="Y(i)",
    points(XY),
    color=red,
    explicit(ye(x),x,0,1),  /* Vegleich mit exakter LÃ¶sung */
  title="FDM approximating polygon solution")$
```

▷ Click to RUN the code.



**Exercise 161.** Repeat example 6.2.1 for $n = 8$.

**Exercise 162.** Solve 6.2.1 using `linsolve` or `solve` instead of `rref`.

**Exercise 163.** Solve 6.2.1 using `invert` for the matrix inverse instead of `rref`.
You may use `Cm:coefmatrix(m,[y0,y1,y2,y3,y4]);` and then do `Cmm1.(-col(Am,6));`

**Exercise 164.** (augcoefmatrix)

> Using MAXIMA's `augcoefmatrix` allows to go from the system of linear equations $eq_k$ directly and comfortable to the matrix of coefficients and also to append the RHS's as a column vector. This allows very compact and short code lines to formulate the FDM.

- SCHONEFELD [43, p.409] has to construct the 'augcoefmatrix' 'by hand' - complicated.
- The same is true for the Python code for the FDM by HECKBERT at ▷ BVP: FDM.

```
% PYTHON code for finite differences method to solve fireworks problem
function findif(n)
..
h = te/n;

A(1,1) = 1;
b(1) = 0;     % y0 = 0

for i = 2:n
  A(i, i-1:i+1) = [1 -2 1];
  b(i) = ...;
end

A(n+1,n+1) = 1;
b(n+1) = ye;
...
```

**a.** Try to explicitly construct a MAXIMA function augcoefmatrix1 along the Python code lines of HECKBERT.
**b.** Solve 6.2.1 using your homemade MAXIMA function augcoefmatrix1.

**Exercise 165.** Redo the FDM *fireworks example* by HECKBERT at ▷ BVP: FDM. using our 4-step-procedure to solve an BVP.

### 6.2.2 the general procedure FDM for BVP $y'' = f(x, y, y'), BC : \begin{smallmatrix} y(x_o)=a \\ y(x_e)=b \end{smallmatrix}$.

```
/* MAXIMA --- FINITE DIFFERENCE METHOD (FDM) for BVP of 2nd order --- */
FDM(xo,xe,a,b,n):= block( kill(y),
    h    : (xe-xo)/n,
    x(k) := xo + k*h,

    y     :  makelist (concat (y,i), i,0, n),                  /* (1) */
    yn    :  y[n+1],
    yp(k) := (y[k+2]-y[k])/(2*h),
    ypp(k):= (y[k]-2*y[k+1]+y[k+2])/h^2,

    Eqs: flatten([y0=a, makelist( BVP(i),i,1,n-1), yn=b]),      /* (2) */
    Y  : map(rhs, linsolve(Eqs, y))                             /* (3) */
    )$


/* Test BVP:   y''+0.2y'+4y=3x+1   BC: y(0)=0.1=a, y(1)= 0.7=b */
fpprintprec:5$  ratprint:false$

BVP(k):=ypp(k) + 0.2*yp(k) + 4*y[k+1]=3*x(k)-1;                 /* (4) */
FDM(xo:0, xe:1, a:0.1, b:0.7, n:8), numer;                      /* (5) */
```

The arguments of `FDM(xo,xn, a,b,n)` are as follows:

1. `xo`: the left corner of the interval $x_o \le x \le x_e$ of interest
2. `xe`: the right corner
3. `a`: the left BC, i.e $y(x_o) = a$
4. `b`: the right BC, i.e $y(x_e) = b$
5. `n`: the number of sub-intervals of equal length $h$
6. `BVP(k)`: the BVP equation in a standard form with lexicon $\begin{smallmatrix} y \\ y[k] \end{smallmatrix}$ $\begin{smallmatrix} y' \\ yp(k) \end{smallmatrix}$ $\begin{smallmatrix} y'' \\ ypp[k] \end{smallmatrix}$, see (4). Has to be defined before the call to `FDM`.

▷ Click to RUN the code.

```
(%o4)   BVP(k):=ypp(k)+0.2 yp(k)+4 y[k+1]=3 x(k)-1
(%o5)   [0.1,0.29143,0.45051,0.57398,0.66091,0.71261,0.73255,0.72607,0.7]
```

*Comment*: In (1) ff. we define the apparatus for the approximation, i.e. the $y_i$ variables and the discretized derivatives $yp_i \equiv y'$, $ypp_i \equiv y''$. (2) constructs the flattend list of all discretized BVP equations, including the BC's, put at the first and last position. (3) solves the system `Eqs` of equations using MAXIMA's build-in routine `linsolve`, picks all RHS values via `map(rhs,..)` and saves all approximate results for the solution function $y$ in the container variable `Y`.

**Exercises.**

**Exercise 166.** The test run (4) solves exercise 161.
Run the test for $n = 16$ and do a plot a la 6.2.1.4:.
Solve exercise 138 using our `FDM` function.

**Exercise 167.** Use the finite difference method with $n = 4$, $n = 8$ and $n = 16$ to approximate the solution to the BVP: $t^2y'' = -4xy' - 2y$ with $y(1) = 12$ and $y(3) = 2$ and compare with the exact solution $ye = \frac{3}{x} + \frac{9}{x^2}$, cf. [43, p.417].

**Exercise 168.** Solve exercise 139 using our `FDM` function.

**Exercise 169.** Use the finite difference method to
**a.** approximate the solution to the BVP: $y'' + \frac{y}{9} = 5\sin(\frac{x}{2})$ with $y(0) = 0$ and $y(\pi) = 0$ with $h = \frac{\pi}{10}$ and compare with the exact solution $ye = 24\sqrt{3}\sin(x/3) - 36\sin(x/2)$,
**b.** solve the BVP again with $h = \frac{\pi}{20}$ and compare the the absolute errors of the two calculations. Cf. [7, p.458].

**Exercise 170.** Do the example from ▷ IRON: bvp 3, i.e.
solve the BVP $y'' = -\frac{(y')^2}{y}$ with BC $y(0) = 1,\ y(1) = 2$.

**Exercise 171.** Do the difficult example from ▷ IRON: bvp 4, i.e.
solve the BVP $y'' - y + y^2 = 0$ with BC $y'(0) = 0,\ y \to 0$ as $x \to \inf, y > 0$.

**Exercise 172.** Do the ▷ HECKBERT rocket problem. using our function `FDM(.)`.

**Exercise 173.** Solve the BESSEL IVP $y'' + \frac{y'}{x} + y = 0,\ y(0) = 1,\ y'(0) = 0$ approximately using our `FDM` function, cf. [12, p.197].
Compare with the exact solution, the BESSEL function $J_0(x)$.

**Exercise 174.** Do the ▷ WIKI: Standard boundary value problem. using our `FDM(.)` function.

**Exercise 175.** Do the BVP's examples 4.2.3, 4.2.5 and 4.2.6 of ▷ NIEMEYER: FDM.

**Exercise 176.** Do the 2 BVP's ▷ BERKELEY PYTHON NUMERICAL METHODS: FDM.

**Exercise 177.** Do the BVP's examples ▷ UNI MUENSTER, DE: FDM methods. in §3.2.

**Exercise 178.** Do the example in ▷ VESELY: Relaxation Method.

**Exercise 179.** Study ▷ HELLEVIK: NM 4 Engineers: FDM. and ▷ : Differences.

**Exercise 180.** Do some of the exercises 140 ff using our `FDM(.)` function.

# 7 Appendix: Collection of Source Code

This is a collection of relevant definitions from the booklet.

```
/*===== Dr. W.G. Lindner, Leichlingen, Germany, 2023 =====*/

/* PICARD approximation */
fpprintprec: 5$
ratprint: false$
kill(functions, values, arrays)$

f(x,y):= 2*x*y;
[a,b]: [0,1];
y[0](x) := b;
y[n](x) := b + integrate(f(t,y[n-1](t)),t,a,x);


/* Maxima --ODE type I */
odefx(u,x,xo,yo):= block(
      I1: integrate(u,x,xo,x),
      I2: I1+C,
      I3: rhs( solve( at(I2,x=xo) = yo, C)[1]),
      I4: I1+I3 )$


/* Maxima --ODE type II */
odefy(u,y,xo,yo):= block(
      F: integrate(u,y),
    Sol: solve(F=x+C,y)[1],
     eq: at(Sol,[x=xo,y=yo]),
      c: rhs(solve(eq,C)[1]),
     I4: [Sol, c])$


/* Maxima --ODE type III */
odefxgy(f,g, x,y, xo,yo):= block(
F: integrate(f,x,xo,x),
G: integrate(1/g,y,yo,y),
print("DGL:", 'diff(y,x) = f*g ),
print(".. initial condition:", y(xo) = yo),
print("1. step - identify f(x) and g(y): ", f, " , ", g),
print("2. step - calculate F(x)=", 'integrate(f,x,xo,x)= F),
print("3. step - calculate G(y)=", 'integrate(1/g,y,yo,y) = G),
print("4. step - set up Eq. F(x)=G(y):", F=G),
```

```
print("5. step - solve  Eq. F(x)=G(y(x)) for y: ", Sol: solve(F=G,y)),
print("6. step - choose solution with ", y(xo)=yo),
print("solution: Sol ="),
Sol)$



/* Maxima -- ODE type III : SEPARATION OF VARIABLES */
separable(f,g,x,y,xo,yo):=
                      solve( integrate(1/g,y,yo,y)=integrate(f,x,xo,x), y);



/* Maxima --ODE type VI : HOMOGENOUS ODE */
kill(values,arrays)$
f(x,y) := (y+x)/x;
eq1: u+x*du/dx = subst(u*x,y,f(x,y));
solve(eq1,du);
SEPARABLE(g,h, x,y) := integrate(g,x)=integrate(1/h,y)$
eq2: SEPARABLE(1/x, 1, x,u) ;
eq3: subst(y/x, u, eq2);
solve(eq3,y);


/* Maxima --ODE type IV - INTEGRATING FACTOR METHOD */
VoC(p,q, x,y, xo,yo):= block(
M: exp(integrate(p,x)),  /* integrating factor */
My: integrate(P*y,x),
print("Step 1: check correct shape y'+py=q :  ", 'diff(y,x)+p*y=q),
print("    ... with initial condition:", y(xo) = yo),
print("Step 2 - choose integrating factor M(x) = exp(integral p) =", M),
print("Step 3 - multiply ODE by M*(y'+py=q), integrate : ",
                                M*y=integrate(M*q,x)),
print("Step 4 - divide through M :", y=expand(integrate(M*q,x)/M)+C/M),
c:   at(expand(integrate(M*q,x)/M+C/M), x=xo),
print("Step 5 - calculate C for yo=y(xo) :", solve(yo=c,C) ));

/* Maxima -- ODE type IV : linear ODE and VARIATION OF CONSTANT  */
linear1(f,g, x,y, xo,yo) :=
                     (yo+integrate(g/exp(integrate(f,x,xo,x)), x,xo,x))
                    * exp(integrate(f,x,xo,x));
```

```
/* Maxima --ODE type V: EXACT ODE */
exact(P,Q, x,y):=
if is( diff(P,y) = diff(Q,x))
then
   block( [A,B,C,D],
           F: integrate(P,x),
           B: F + funmake(A, [y]),
           C: diff(B,y) = Q,
           D: rhs((solve(C,A(y))[1])),
           E: integrate(D,y),
           f: F+E  )
else ("ODE is not exact.")$


/* Maxima -- ITERATE */
iterate(u,x,xo,n) := block(
                     [l:[xo], numer:true, val:xo],
                     for i thru n do
                        (val: subst(val,x,u), l: cons(val,l)),
                     reverse(l));


/* Maxima -- ITERATE2 */
iterate2(u,x,xo,n) := block(
        [ l: [[xo[1],xo[2]]], numer:true,  val: [xo[1],xo[2]]],
        for i thru n do
           (val: subst([x[1]=val[1], x[2]=val[2]], u), l:cons(val,l)),
           reverse(l));


/* Maxima -- SECANT METHOD   */
/*           include function iterate(..) here */
secant1(u,x, a,b, n) := block(
                         [C: (a-b)/(subst(a,x,u)-subst(b,x,u))],
                         iterate( x - C*u, x,a,n)) ;


/* Maxima -- NEWTON-RAPHSON iteration */
/*           include our function iterate(..) here */
newton1(u,x,a,n) := block([c: float(at(diff(u,x),x=a))],
                           iterate( x-u/c, x,a, n)     );
```

```
/* Maxima  -- IMPLICIT DIFFERENTATION */
idiff(f,x,y,m):=block(
                IMP1: iterate('diff(u,x) + f*'diff(u,y), u,f,m-1),
                factor(ev(IMP1, diff)) )$


/* Maxima -- IMPLICIT TAYLOR METHOD of order m */
/* include our functions iterate(u,x,xo,n) and idiff(f,x,y,m) here */
iTaylor(f,x,y, h, m) := y + idiff(f,x,y,m).makelist(h^r/r!, r,1,m);


/* Maxima --  TAYLOR SOLUTION METHOD of ODE y'=f(x,y) */
/* include functions iterate2(u,x,xo,n), idiff(f,x,y,m)
   and     iTaylor(f,x,y,h,m) here */
iTaylorsol(f,x,y, xo,yo, h, m, n) :=
                iterate2([x+h, iTaylor(f,x,y, h, m)], [x,y], [xo,yo], n);


/* MAXIMA : --- EULER method  ---* /
EULER(x,y, xo,yo, h, n) := iterate2([x+h, y+h*f(x,y)],[x,y],[xo,yo],n);


/* MAXIMA : --- modified EULER method --- */
modiEULER(x,y, xo,yo, h,n):= block(
               k1 : f(x,y),
               k2 : f(x+h, y + h*k1),
               iterate2([x+h, y+h/2*(k1+k2)],[x,y],[xo,yo],n) );


/* MAXIMA --- HEUN's method --- */
HEUN( x,y, xo,yo, h,n) := block(
           k1 : f(x,y),
           k2 : f( x+2*h/3,  y+2*h/3*k1 ),
           iterate2([x+h, y+h/4*(k1+3*k2)],[x,y],[xo,yo],n) );


/* MAXIMA : --- MIDPOINT METHOD as RK2 method --- */
midpoint( x,y, xo,yo, h,n) :=  block(
               k1 : f(x ,y),
               k2 : f(x, y + h/2*k1),
               iterate2([x+h, y+h*k2], [x,y], [xo,yo], n));
```

```
/* MAXIMA --- RUNGE-KUTTA order 3 --- */
RK3(x,y,xo,yo,h,n) := block(
                k1 :  f(x,y),
                k2 : f(x+h/2, y + h/2*k1),
                k3 : f(x+h, y + 2*h*k2-h*k1),
                iterate2([x+h, y+h/6*(1*k1+4*k2+1*k3)],[x,y],[xo,yo],n));


/* MAXIMA : --- classic RUNGE-KUTTA  method of order 4 --- */
RK4(x,y,xo,yo,h,n):= block(
                     k1 : f(x,y),
                     k2 : f(x+h/2, y + h/2*k1),
                     k3 : f(x+h/2, y + h/2*k2),
                     k4 : f(x+h  , y + h*k3),
                     iterate2([x+h, y+h/6*(1*k1+2*k2+2*k3+1*k4)],
                              [x  , y], [xo,yo], n) );


/* MAXIMA --- iterate3 for triples --- */
iterate3(u,x,xo,n) := block(
            [    l: [ [xo[1],xo[2],xo[3]]], numer: true,
               val: [  xo[1],xo[2],xo[3]]],
    for i thru n do
        (val: subst([x[1]=val[1], x[2]=val[2], x[3]=val[3]], u),
           l: cons(val,l)),
        reverse(l));


/* MAXIMA --- EULER method for systems of IVP --- */
EULERsys(t,x,y, to,xo,yo, h,n) :=
     iterate3([t+h, x+h*f(t,x,y), y+h*g(t,x,y)], [t,x,y], [to,xo,yo], n)$


/* MAXIMA --- RUNGE-KUTTA method RK4sys for systems of IVP --- */
RK4sys( x,y,z, xo,yo,zo, h,n):= block(
    k1 : h*f(x,y,z),
    l1 : h*g(x,y,z),
    k2 : h*f(x+1/2*h, y + 1/2*k1, z + 1/2*l1),
    l2 : h*g(x+1/2*h, y + 1/2*k1, z + 1/2*l1),
    k3 : h*f(x+1/2*h, y + 1/2*k2, z + 1/2*l2),
    l3 : h*g(x+1/2*h, y + 1/2*k2, z + 1/2*l2),
    k4 : h*f(x+    h, y +     k3, z + l3),
    l4 : h*g(x+    h, y +     k3, z + l3),
```

```
    iterate3([x+h, y + 1/6*(1*k1+2*k2+2*k3+1*k4),
                   z + 1/6*(1*l1+2*l2+2*l3+1*l4)],
             [x,y,z],[xo,yo,zo],n) )$


/* MAXIMA -- BRAEUNING's method for 2nd order IVP --- */
RK4ode2( x,y,yp, xo,yo,ypo, h, n):= block(
        k0 : 1/2*h^2* f(x,y,yp),
        k1 : 1/2*h^2* f(x+h/2, y + 1/2*h*yp + 1/4*k0, yp + k0/h),
       k1p : 1/2*h^2* f(x+h/2, y + 1/2*h*yp + 1/4*k0, yp + k1/h),
        k2 : 1/2*h^2* f(x+h, y + h*yp + k1p, yp + 2*k1p/h),
         k : 1/3*(k0+k1+k1p),
         l : 1/6*(k0+2*k1+2*k1p+k2),
      iterate3([x+h, y+h*yp+k, yp+2*l/h],[x,y,yp],[xo,yo,ypo],n) );


/* MAXIMA --- ROMER's method for 2nd order IVP --- */
RK4romer( x,y,yp, xo,yo,ypo, h, n):= block(
         k1 : h* f(x,y,yp),
         k2 : h* f(x+h/2, y + 1/2*h*yp + h/8*k1, yp + k1/2),
         k3 : h* f(x+h/2, y + 1/2*h*yp + h/8*k1, yp + k2/2),
         k4 : h* f(x+h, y + h*yp + h/2*k3, yp + k3),
         iterate3([x+h, y+h*(yp+1/6*(k1+k2+k3)), yp+1/6*(k1+2*k2+2*k3+k4)],
                  [x,y,yp],[xo,yo,ypo],n) );


/* MAXIMA --- SHOOTING METHOD for BVP --- */
shoot(xo,a,b ,h,n, p,q) := block(
      F(w):= last(RK4sys(x,y,z, xo,a,w, h, n))[2] - b,
      W: find_root(F,w, p,q))$

/* MAXIMA --- FINITE DIFFERENCE METHOD (FDM) for BVP of 2nd order --- */
FDM(xo,xe,a,b,n):= block( kill(y),
    h    : (xe-xo)/n,
    x(k) := xo + k*h,
    y     :  makelist (concat (y,i), i,0, n),
    yn    :  y[n+1],
    yp(k) := (y[k+2]-y[k])/(2*h),
    ypp(k):= (y[k]-2*y[k+1]+y[k+2])/h^2,
    Eqs: flatten([y0=a, makelist( BVP(i),i,1,n-1), yn=b]),
    Y  : map(rhs, linsolve(Eqs, y))  )$
```

# 8 Bibliography

# References

[1] Ammari & al. (20??): *Numerical Methods for Ordinary Differential Equations.* url: https://people.math.ethz.ch/~grsam/SS19/NAII/resources/lecture1.pdf

[2] Andrecut, M. (2000): *Introductory Numerical Analysis. Lecture Notes.* Parkland: Universal Publishers.

[3] Ayres, F. (1952): *Theory and Problems of Differential Equations.* New York: McGraw-Hill (Schaum's Outline Series)

[4] Barth, E.J. (2023): *The MaximaList (Blog).* url: https://themaximalist.org

[5] Bazett, T. (2023): *Introduction to Differential Equations.* url: https://web.uvic.ca/~tbazett/diffyqs/diffyqs.html

[6] Berry, J.S. et al. (1993): *Learning Mathematics through* Derive. Chichester: Ellis Horwood.

[7] Borse, G.J. (1997): *Numerical Methods with MATLAB.* Boston: PWS Publishing.

[8] Borse, G.J. (1991): *FORTRAN 77 and Numerical Methods for Engineers.* Boston: PWS Publishing.

[9] Bronson, R. (1973): *Modern Introductory Differential Equations.* New York: McGraw-Hill. Schaum's Outline Series.

[10] Burden, R.L. & Faires, J.D. ($^2$1993): *Numerical Analysis.* Boston: PWS Publishing.

[11] Braun, M. (1975): *Differential Equations and Their Applications.* New York, Heidelberg: Springer.

[12] Bräuning, G. ($^2$1965): *Gewöhnliche Differentialgleichungen.* Leipzig: VEB Fachbuch Verlag.

[13] Dawkins, P. (2023): *Pauls's Online Notes - Differential Equations.* url: https://tutorial.math.lamar.edu/Classes/DE/DE.aspx

[14] Dorner, G.C. et al. (2000): *Visual Mathematics, Illustrated by the TI-92 and the TI-89.* France: Springer.

[15] Dowling, E.T. ($^2$1980): *Introduction to Mathematical Economics.* New York: McGraw-Hill (Schaum's Outline Series)

[16] Forst, W. & Hoffmann, D. (2005): *Gewöhnliche Differentialgleichungen. - Theorie und Praxis - vertieft und visualisiert mit* Maple. Berlin: Springer.

[17] Etchells, T.A. & Berry, J.S. (1997): *Learning Numerical Analysis through* Derive. Lund: Studentlitteratur. Chartwell-Bratt.

[18] Faires, J.D. & Burden, R.L. (1993): *Numerical Methods.* Boston: PWS Publishing.

[19] Günter, N.M. & Kusmin, R.O. ($^5$1966): *Aufgabensammlung zur Höheren Mathematik I.* Berlin: Deutscher Verlag der Wissenschaften.

[20] Hairer, E. & Lubich, C. (?): *Numerical solution of ordinary differential equations.* url: `https://na.uni-tuebingen.de/~lubich/pcam-ode.pdf`

[21] Hellevik, L.E. (2020): *Numerical Methods for Engineers.* url: `https://folk.ntnu.no/leifh/teaching/tkt4140/._main000.html`

[22] Jänich, K. (1983): *Analysis fur Physiker und Ingenieure.* Berlin: Springer.

[23] Jordan-Engeln, G. & Reutter, F. ($^2$1976): *Formelsammlung zur Numerischen Mathematik mit* Fortran IV*-Programmen.* Mannheim: Bibliographisches Institut.

[24] Kamke, E. ($^5$1964): *Differentialgleichungen I.* Leipzig: Akademische Verlagsgesellschaft Geest & Portig.

[25] Koepf, W. (1996): Derive *für den Mathematikunterricht.* Braunschweig: Vieweg.

[26] Koepf, W. (1994): *Höhere Analysis mit* Derive. Braunschweig: Vieweg.

[27] Krysicki, W. & Wlodaski, L. (1971): *Höhere Mathematik in Aufgaben. Teil 2.* Leipzig: BSB B. G. Teubner.

[28] Lebl, J. (2022): *Notes on DiffyQs.* url: `https://www.jirka.org/diffyqs/diffyqs.pdf`

[29] Lowe, B. & Berry, J.S. (1998): *Learning Differential Equations through* Derive. Lund: Studentlitteratur.

[30] MathSoft (1995): *Exploring Numerical Recipes.* Cambridge: MathSoft Inc. (MathCAD)

[31] Marsden, J. & Weinstein, A. ($^2$1985): *Calculus I.* New York: Springer.

[32] Marsden, J. & Weinstein, A. ($^2$1985): *Calculus II.* New York: Springer.

[33] Marsden, J. & Weinstein, A. ($^2$1985): *Calculus III.* New York: Springer.

[34] Minorski, W. P. ($^2$1965): *Aufgabensammlung der Höheren Mathematik.* Leipzig: VEB Fachbuchverlag.

[35] Mohr, R. (1998): *Numerische Methoden in der Technik* (MatLab). Braunschweig: Vieweg.

[36] Nakamura, S. (1993): *Applied Numerical Methods in C.* Eaglewood Cliffs: Prentice-Hall.

[37] Opfer, G. (2008): *Numerische Mathematik für Anfänger.* Wiesbaden: Vieweg+Teubner.

[38] Press, W.H. & al. (1988): *Numerical Recipes in C.* Cambridge: Cambridge University Press.

[39] Quarteroni, A. et al. (2000): *Numerical Mathematics.* London, New York: Springer.

[40] Rich, A. & Stoutemyer, D. et al. (1988 ff): Derive *User Manual.* Honolulu: Soft Warehouse..

[41] Roberts, L.F. (1995): *Introduction to Ordinary Differential Equations.* Cambridge: MathSoft Inc. (MathCAD)

[42] Romer, A. (1983): *50* BASIC*-Programme.* Mannheim: Bibliographisches Institut.

[43] Schonefeld, S. (1994): *Numerical Analysis via* Derive. Urbana: MathWare.

[44] Spiegel, M. R. (1963): *Advanced Calculus.* New York: McGraw-Hill.

[45] Steeb, W.-H & Lewien, D. (1991): *Algorithms and Computation with* Reduce. Mannheim: Bibliographisches Institut.

[46] Stoer, J. & Bulirsch, R.Z. (1973): *Einführung in die Numerische Mathematik II.* Berlin: Springer.

[47] Süli, E. (2022): *Numerical Solution of Ordinary Differential Equations.* url: `https://people.maths.ox.ac.uk/suli/nsodes.pdf`

[48] Timberlake, T.K. & Mixon, J.W. (2016): *Classical Mechanics with* Maxima. New York: Springer.

[49] Venz, G. (1978): *Lösung von Differentialgleichungen mit programmierbaren Taschenrechnern.* München, Wien: Oldenburg.

[50] Werner, W. ($^2$1998): *Mathematik lernen mit* Maple. *Band 2.* Heidelberg: dpunkt.

[51] Wolfram mathworld: Picards Existence Theorem. url: `https://mathworld.wolfram.com/PicardsExistenceTheorem.html`

[52] Woollett, E.L. (2023): *Maxima by Example, Ch. 3: Ordinary Differential Equation Tools.* url: `https://home.csulb.edu/~woollett/mbe03.html`