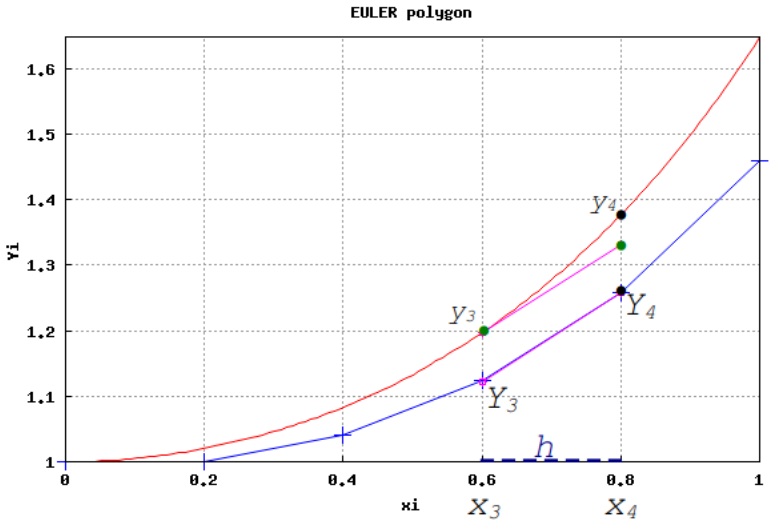
Exploring Math Σ *math* with EIGENMATH

Differential Equations

An Introduction with EIGENMATH and MAXIMA*online*



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 wrote a function `iterate` for CAS EIGENMATH 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 [47], BURDEN-FAIRES [10], LOWE/BERRY [30] and on the internet BAZETT [5], DAWKINS [13], HELLEVIK [21], LEBL [28], HAIRER/LUBICH [20], SÜLI [48] 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 EIGENMATH 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 EIGENMATH 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 EIGENMATH 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 EIGENMATH programs and the corresponding selection of examples from diverse sources will encourage the beginner to go further into the theory.

This text uses the free CAS EIGENMATH, enhanced and reinforced by MAXIMA^{online}. It is a small, but well designed and powerful computer algebra system (CAS), that can be used to solve problems in mathematics and the natural and engineering sciences. It is a personal resource for students, teachers and scientists.
For running an EIGENMATH script of this booklet no installation is necessary, *everything*

runs directly online: a click on a link like Click here .. in this text is enough to call the corresponding script - and by a click on the `Run` button the calculation is made, allowing further free inputs by the user. Such a link is therefore at the same time a silent invitation to become active. If you own an iMac, there is the option to install the *app*[52] EIGENMATH free of charge and run the scripts by *mark–copy–paste* into EIGENMATH's script window. We also use the CAS MAXIMA^{online} to draw slope fields of ODE's or to do special calculations which are beyond the current possibilities of EIGENMATH.

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 EIGENMATH and MAXIMA^{online} in the field of ODEs.

Sadly, Prof. Roland BULIRSCH passed away in 2022.

Roland Z. BULIRSCH, 1934–2022

User comments are very welcome.
I want to thank George WEIGT for his friendly support while writing these notes.

Wolfgang Lindner
Leichlingen, Germany
July 2023
dr.w.g.Lindner@gmail.com
https://lindnerdrwg.github.io

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

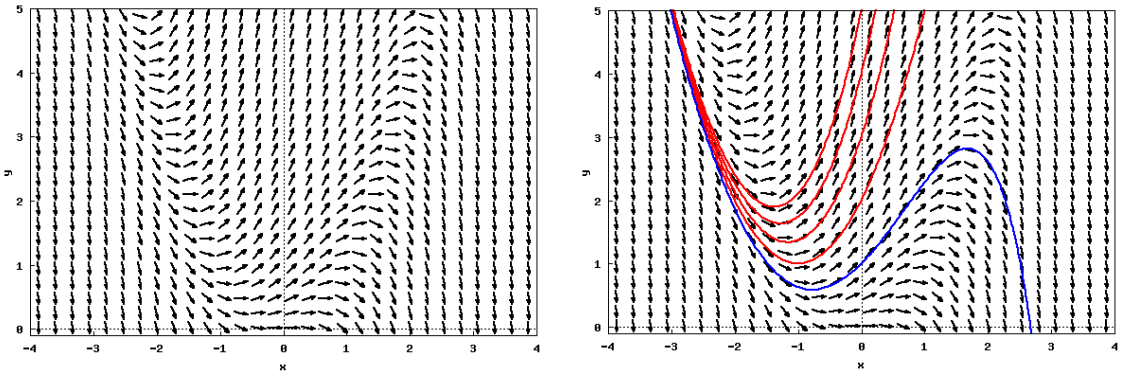
$$y' = f(x, y) \tag{1.1}$$

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

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

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

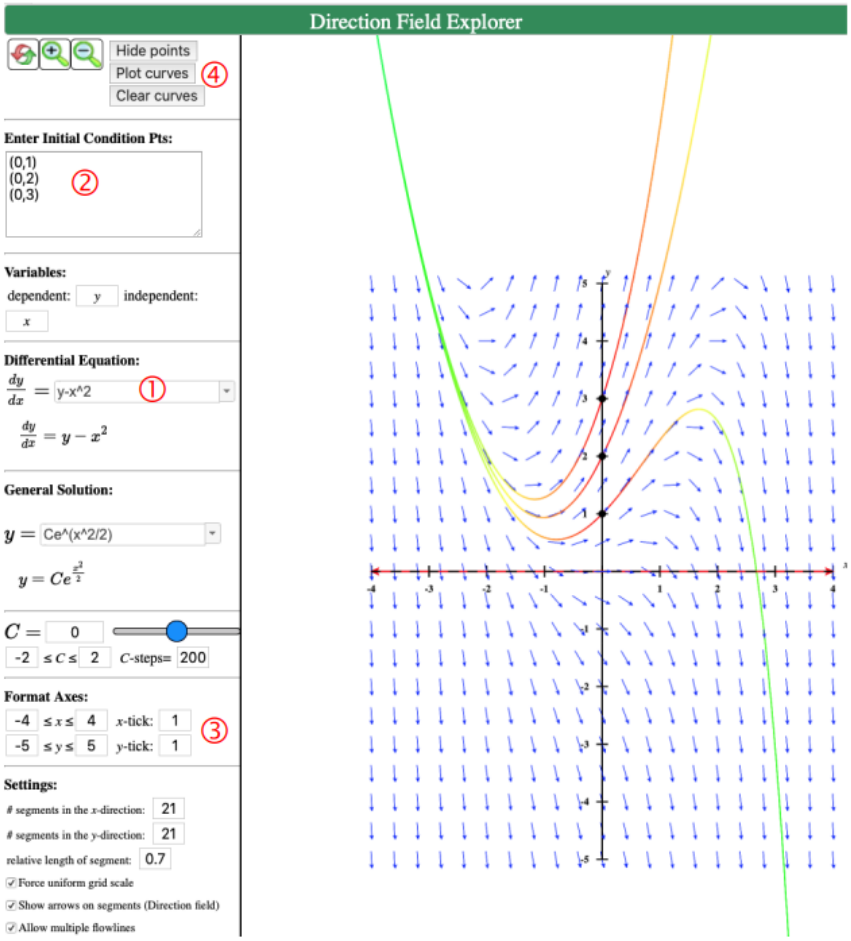
• A given ODE's should be transformed in the "standard" form (1.1), because most facts and methods relay on this representation.
• As a first preview on what follows have a look at the following two figures:



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)
EIGENMATH can't plot direction fields. So we have to use extern utilities.
A. We may use the interactive CALCPLOT3D to reproduce the plot Fig.1.RHS:



• ▷ Click here to OPEN CALCPLOT3D. Now fill in the relevant data:
Step **1** : fill in the RHS of the ODE: $y - x^2$.
Step **2** : fill in the initial conditions, e.g. $y(0) = 1$ as the point $(0, 1)$.
Step **3** : fill in the grid dimensions, e.g. $[-4, 4] \times [-5, 5]$
Step **4** : press the button Plot curves.

B. If you prefer a statement driven plotting utility, then use e.g. MAXIMA^{online} and the following function:

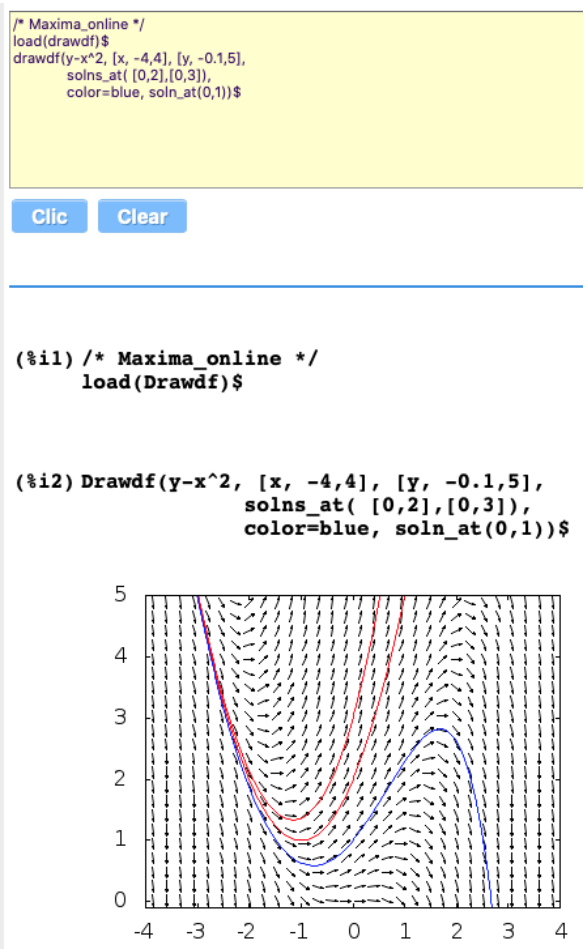
LEXICON	*concept*	MAXIMA^{online}
	direction field ..df	load(drawdf);
	call plot	drawdf($y - x^2$);

Here is the code[1] that plots the direction field of the given ODE $y' = y - x^2$ and a solution function by MAXIMA^{online}:

```
/* 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 into □ of MAXIMA^{online} and RUN it with CLICK

The MAXIMA^{online} output:



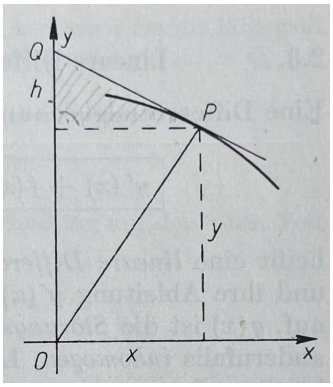
[1] MAXIMA^{online} code lines in this booklet are written in magenta.

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 $y(1.5) = 2$.

Exercises.

Exercise 1. Plot the direction fields of the ODE's of example 2 and the corresponding solution function of the IVP by means of CalcPlot3D or Maxima^{online}.

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. [48, 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.

• We calculate eq. (1.6) until (1.9) with EIGENMATH:

```
--- EIGENMATH : PICARD 1---
y0(x) = 1

y1(x) = 1 + defint(2*x*y0(x),x,0,x)
y1

y2(x) = 1 + defint(2*x*y1(x),x,0,x)
y2

y3(x) = 1 + defint(2*x*y2(x),x,0,x)
y3
```

▷ Click here to RUN the code.[2]

The EIGENMATH output is:

$$y_1 = x^2 + 1$$

$$y_2 = \frac{1}{2}x^4 + x^2 + 1$$

$$y_3 = \frac{1}{6}x^6 + \frac{1}{2}x^4 + x^2 + 1$$

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 EIGENMATH that $y(x) := e^{x^2}$ is the solution of IVP $y' = 2xy$, $y(0) = 1$:

```
--- EIGENMATH ---
y(x)=exp(x^2)
y(0)              -- (1)
d(y(x),x)         -- (2)
2 x y(x)
```

▷ Click here to RUN the code.

The EIGENMATH output:

[2] The first invocation could take some seconds.

$$1$$

$$2\,x\,\exp\left[x^2\right]$$

$$2\,x\,\exp\left[x^2\right]$$

Comment. Eq. –(2)– verifies, that $y = e^{x^2}$ is a solution of the ODE and (1) 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 the PICARD method in EIGENMATH. We use (1.5) and this lexicon:

LEXICON	*Math*	EigenMath
	$\int_a^b f(x)dx$	`defint(f(x),x,a,b))`

1.2.2 PICARD by EIGENMATH

```
--- EIGENMATH : PICARD iteration ---
f(x,y) = 2*x*y                                          -- (1) --
do(a=0, b=1)                                            -- (2) --

y(n,x)= test(n = 0, b, b+defint(f(x,y(n-1,x)),x,a,x))       -- (3) --

y(4,x)                                                  -- (4) --
for(i,1,5, print(y(i,x)))                               -- (5) --
```

▷ Click here to RUN the code.

The EIGENMATH output:

$$\frac{1}{24}x^8 + \frac{1}{6}x^6 + \frac{1}{2}x^4 + x^2 + 1$$

$$x^2 + 1$$

$$\frac{1}{2}x^4 + x^2 + 1$$

$$\frac{1}{6}x^6 + \frac{1}{2}x^4 + x^2 + 1$$

$$\frac{1}{24}x^8 + \frac{1}{6}x^6 + \frac{1}{2}x^4 + x^2 + 1$$

$$\frac{1}{120}x^{10} + \frac{1}{24}x^8 + \frac{1}{6}x^6 + \frac{1}{2}x^4 + x^2 + 1$$

Comment. Line (1) defines the RHS of the ODE. In (2) we set the IV's to $a := 0$ and $b := 1$. Line (3) is the direct translation of (1.5): Because $y(.,x)$ is an indexed function where the index n is 'recursively' reduced by 1, the next function $y(n,x)$ is calculated as

the definite integral of the previous function $y(n-1,x)$ with the limits $x=a$ and $x=x$. So a recursive process is established which allows to call an arbitrary function `y(n,x)` or a particular value `y(9,1.2)`, e.g. see (4). In (5) we display the first 5 approximate solution functions. We see that this list shows the results (1.6) to (1.9). Ok.

• Let's visualize the approximation process.

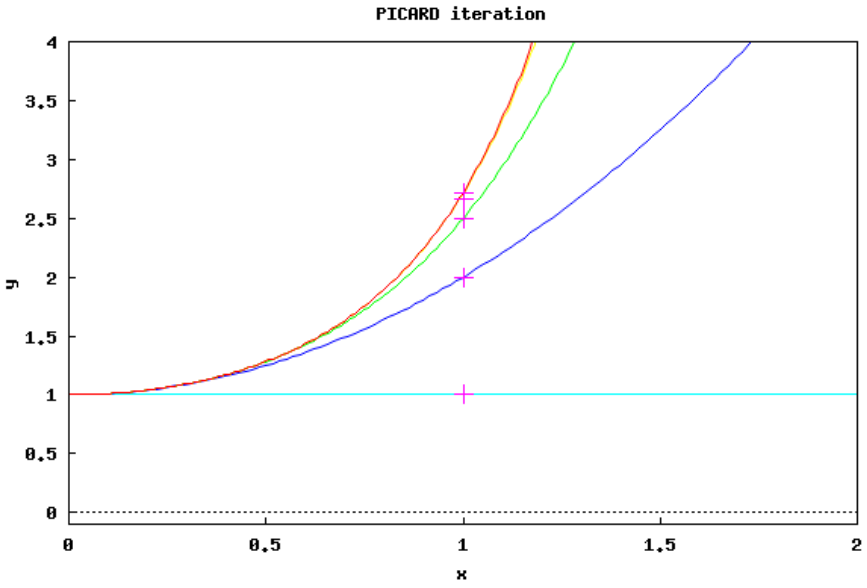


Figure 2: The plot shows the approximating functions $y(0,x)$, $y(1,x)$, $y(2,x), y(3,x)$ tend to always better approach to the exact solution $y(x)=\exp(x^2)$.
The points list $(1,y(n,1))_{n=0,1,2,3}$ tends to approach the limit value $y(1)=e^1=e\approx 2.71828$. It is visualized as magenta crosses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

Let's calculate the points list $(1,y(n,1))_{n=0,1,2,3} \rightsquigarrow e\approx 2.71828$:

```
--- EIGENMATH : PICARD iteration at x=1 ---
f(x,y) = 2*x*y
do(a=0, b=1)
y(n,x)= test(n = 0, b, b+defint(f(x,y(n-1,x)),x,a,x))

Table    = zero(5,2)
Table[1] = (1,1)
for(i,1,4, Table[i+1]=(1, float(eval(y(i,x),x,1) )))
Table
```

Comment. We define a value table `Table` as a 5-by-2-matrix, whose 1st value pair `Table[1]` is set to $(1,1)$. Then the table is filled successively with the float values $y(i,1)$: `(eval(y(i,x),x,1)` substitutes 1 for x in function $y(i,x)$.

▷ Click here to RUN the code.

The EIGENMATH output:

$$T_{able} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 2.5 \\ 1 & 2.66667 \\ 1 & 2.70833 \end{bmatrix}$$

Exercise 5. Reproduce fig.2 using CALCPLOT3D or MAXIMA^{online}, see example 2.

Exercises.

Exercise 6. 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 7. 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 8. 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 9. 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 10. 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 11. 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 12. 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 13. Construct the PICARD iterates for the initial value problem in [1, p.20, Example 2.7].

Exercise 14. 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 EIGENMATH.
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

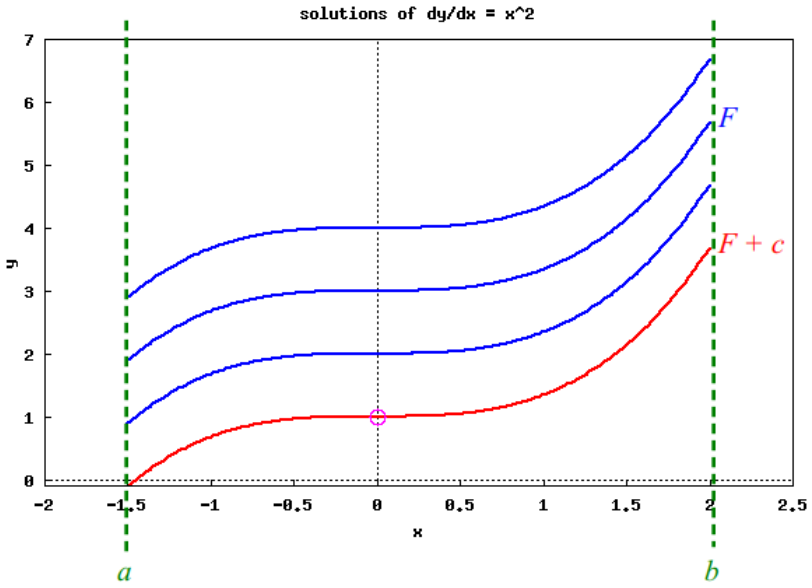


Figure 3: The plot shows four of the infinity many solution functions $y(x) = F(x) = \frac{1}{3}x^3 + c$ of the ODE $y' = x^2$ with initial value $y(0) = 1$. The special solution function $F + c$ with $y(0) = F(0) = 1$ is plotted in *red* and has $c = 1$.

- 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* [3] $F(x)$ 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 3 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$f(x, y) = x^2 = f(x\ only)$.
a. The solution function $y(x)$ is constructed in 3 steps:
I1: $\int_{x_o}^{x} t^2dt + C = \frac{x^3}{3} + C = y(x)$.
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 part of the solution process as a user defined EIGENMATH function `odefx`:

```
--- EIGENMATH : ODE typ I ---
odefx(u,x,xo,yo)= do(
      I1 = defint(u,x,xo,x),                          -- (1)
      I2 = I1+C,                                      -- (2)
      I2 )

odefx(x^2, x,0,1)                                     -- (3)
-- C+1/3*0^3=1     ==> C=1 ==> y(x) = 1/3x^3+1

odefx(x^2, x,1,1)                                     -- (4)
-- C+1/3*1^3-1/3=1 ==> C=1 ==> y(x) = 1/3x^3+2/3
```

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`: ▷ Click here to RUN the code.

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

The EIGENMATH output for the test cases (1) and (2) are:

$$C + \frac{1}{3}x^3$$

$$C + \frac{1}{3}x^3 - \frac{1}{3}$$

Comment. In (1) we calculate the definite integral $F = \int_{x0}^{x} f$ and add in (2) the integration constant C to it in order to build all solutions. (3) and (4) are tests. We put by hand x_o into I2 and solve this equation for C. With this C we pick the particular IVP solution.

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

```
/* MAXIMA-online -- IVP example with build-in function ode2 */
depends(y, x) $         /* y(x) depends on x */
e: diff(y,x) = x^2;     /* the given ODE */
r: ode2(e,y,x);         /* solve ODE equation e, result is r */
ic1(r, x=1, y=1);       /* solve ODE respecting the IVP y(1)=1 */
```

▷ Click here, then mark code, paste into ☐ and CLICK to RUN the code.

The MAXIMA^{online} output for the test case (4) is:

```
(%i2) e: diff(y,x)=x^2;
```

(%o2) $\frac{d}{dx}y = x^2$

```
(%i3) r: ode2(e,y,x);
```

(%o3) $y = \frac{x^3}{3} + \%c$

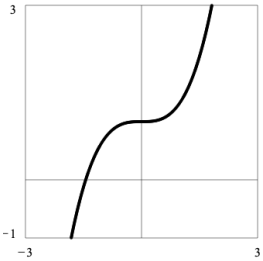
```
(%i4) ic1(r, x=1,y=1);
```

(%o4) $y = \frac{x^3+2}{3}$

Let's plot the solution function with EIGENMATH:

```
--- EIGENMATH-online : IVP example y'=x^2 solution curve ---
xrange = (-3,3)
yrange = (-1,3)
draw(x^3/3+1,x)
```

▷ Click here to RUN the code.



Let's sum up:

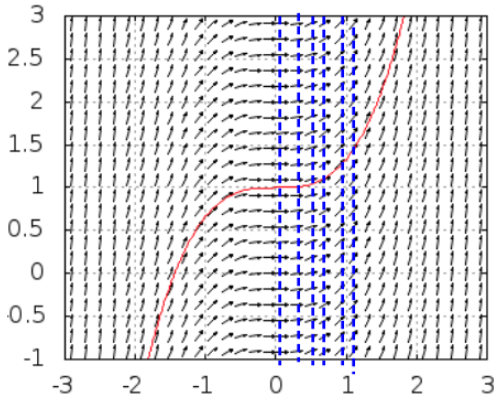


Figure 4: The direction field of the ODE $y' = x^2$ shows, that every point with the same x-coordinate has the same slope: this is visualized through the blue vertical lines, see [24, p.6].

Exercise 15. Reproduce fig.4 by MAXIMA^{online}, see example 2.

Exercises.

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

Exercise 16. Do some of the exercises of [28, p.21-26].
You will find the solutions at ▷ BAZETT: Integral solutions.

Exercise 17. Solve $x^2 \cdot y''(x) = 3x^4 - 1$ given $y = 0$ when $x = 1$ and $y'(1) = 2$.
Cf. LOWE [30, p. 35].

Exercise 18. Solve $y'(x) = \frac{1}{x} - \sin(x)$.

Exercise 19. Solve $\sqrt{1 - x^2} \cdot y'(x) = 1$.

Exercise 20. (LOWE [30, 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 21. 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 EIGENMATH 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[4].

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^{online} function `odefy`, which allows to follow a semi-automatic solution process and gives insight into the black-box `ode2(.)`:

```
/* --- MAXIMA-online : 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])$

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

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` !):

---

[4] 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?

▷ Click here, the mark, paste and Click.

The output is ..

```
(%i2) odefy( 1/cos(y)^2, y, 1,1);

solve: using arc-trig functions to get a solution.
Some solutions will be lost.

(%o2) [y = arctan(C + x), tan 1 − 1]
```

.. and allows to read off the result in `(%o2)` 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-online: solve IVP with build-in function ode2 --- */
depends(y, x) $             /* y(x) depends on x */
e: diff(y,x) = cos(y)^2;    /* the given ODE */
r: ode2(e,y,x);             /* solve ODE equation e, result is r */
ic: ic1(r, x=1, y=1);       /* solve ODE respecting the IVP y(1)=1 */
solve(ic, y);               /* solve eq. ic for y(x) */
```

▷ Click here to RUN the code.

The MAXIMA^{online} output is:

```
(%i4) /* solve ODE equation e, result is r */
      ic: ic1(r, x=1, y=1);

(%o4) tan y = x + tan 1 − 1

(%i5) /* solve ODE respecting the IVP y(1)=1 */
      solve(ic, y);

solve: using arc-trig functions to get a solution.
Some solutions will be lost.

(%o5) [y = arctan(x + tan 1 − 1)]
```

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

Exercise 22. Reproduce the following fig.5 by MAXIMA^{online}, see example 2.

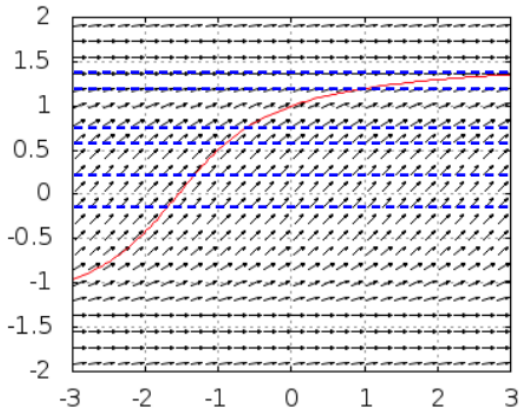


Figure 5: The direction field of the ODE $y' = \cos(y)^2$ shows, that every point on the same parallel w.r.t. the x-axis has the same slope: observe the blue horizontal lines, see [24, p.7].

Exercises.

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

Exercise 23. Solve the ODE $y' = ky,\ k > 0$.

Exercise 24. Solve the ODE $y' = y^2,\ y(0) = 1$.

Exercise 25. Solve the ODE $y' = \sqrt{1 - y^2}$.

Exercise 26. Solve the corresponding IVP $y(1) = 2$ for the exercises 20–22.

Exercise 27. Do exercises 1.1.5, 1.1.6, 1.1.7 froml ▷ BAZETT: Integrals as solutions.

Exercise 28. 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 EIGENMATH function `odefxgy`, which allows to follow a semi-automatic 'pedagogical' solution process :

```
--- EIGENMATH : ODE type III ---
odefxgy(f,g, x,y, xo,yo) = do(
   print("1. step - identify f(x) and g(y): ", eval(f), eval(g)),
   F = defint(f,x,xo,x),
   G = defint(1/g,y,yo,y),
   print("2. step - calculate F(x)", F),
   print("3. step - calculate G(y)", G)
)

odefxgy(-x,   y^2, x,y, 0,1)   -- test
--     f(x)   g(y)
```

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

▷ Click here to RUN the code.

• The test of our function `odefxgy` for the example above $y' = f(x, y) = -xy^2 = h(x) \cdot g(y)$ gives the

EIGENMATH output:

1. step - identify f(x) and g(y):

$-x$

y^2

2. step - calculate F(x)

$F = -\frac{1}{2}x^2$

3. step - calculate G(y)

$G = -\frac{1}{y} + 1$

We now have to solve equation $F = G$ for y by hand or use MAXIMA^{online}:

```
F(x):= -1/2* x^2;
G(y):= -1/y +1;
Sol: solve(F(x)=G(y),y);
```

▷ Click here to go. MAXIMA^{online} output:

```
(%i3) Sol: solve(F(x)=G(y),y);
```

(%o3) $\left[y = \frac{2}{x^2+2}\right]$

Read off the result in `(%o3):` `y=`$\frac{2}{x^2+2}$.

c. We now solve this IVP by invoking the MAXIMA^{online} build-in function `ode2`:

```
/* Maxima -- ODE type III by ode2 -- */
depends(y, x) $             /* y(x) depends on x */
e: diff(y,x) =  -x*y^2;    /* the given ODE */
r: ode2(e,y,x);             /* solve ODE equation e, result is r */
ic: ic1(r, x=0, y=1);       /* solve ODE respecting the IVP y(1)=1 */
solve(ic, y);               /* solve eq. ic for y(x) */
```

▷ Click here to RUN the code.

The MAXIMA^{online} output is the same as in b.
Please observe, that MAXIMA^{online}'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`.

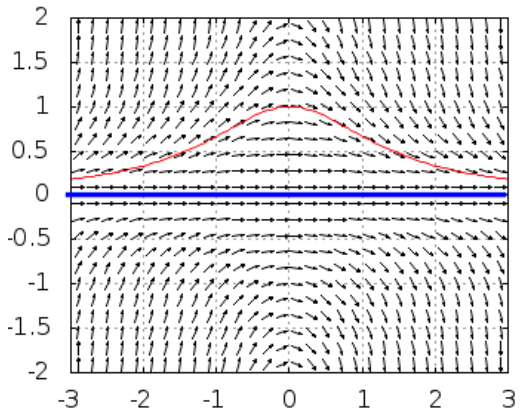


Figure 6: The direction field of the ODE $y' = xy^2$ shows a kind of symmetry. But it also shows that this ODE has the function $y(x) \equiv 0$ (blue line) as an – maybe overseen – solution.

d. Here is a compact function for solving separable ODE's without commenting hints:

```
/* --- MAXIMA-online : ODE type III : SEPARATION OF VARIABLES  --- */
separable(f,g, x,y, xo,yo):= block( assume(y-1>0),
       solve( [integrate(1/g,y,yo,y)=integrate(f,x,xo,x)], [y]));

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

▷ Click here to RUN the code.

The MAXIMA^{online} output is the same as above.

Exercises.

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

Exercise 29. Solve the ODE $(1+x^2)xy\frac{dy}{dx} = 1-y^2$, cf. [27, p.239].

Exercise 30. Solve the IVP $\sin^2 x + (\frac{dy}{dx})^2 = 1$ with $y(x_0) = y_0$, cf. [27, p.229].

Exercise 31. Why is the ODE $y' = x - y^2$ not solvable by separation of the variables?

Exercise 32. Solve the IVP $y' = -\frac{y^2}{2x+1}$, at $x = 0$ is $y = 1$, cf. [12, p.38].

Exercise 33. Solve the IVP $y' = x^2 - x^2y$, at $x = 1$ is $y = 5$. Is $y(x) \equiv 1$ a solution?

Exercise 34. Solve the IVP $xy' = y \ln y,\ y(1) = 2$.

Exercise 35. Solve $t\varphi' + \varphi^2 - 1 = 0$.

Exercise 36. Solve $yy' = x,\ y(1) = 1$.

Exercise 37. 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 38. 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 39. Do some of the exercises of [28, p.33 ff].
You will find the solutions at ▷ BAZETT: Separable equations.

Exercise 40. Do some of the examples and exercises of ▷ DAWKINS, P.: Separable Equations.

Exercise 41. Do exercises 1.1.5, 1.1.6, 1.1.7 froml ▷ BAZETT: Integrals as solutions.

Exercise 42. Do example 1.2 from [1, p.7].

Exercise 43. 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.
1st 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:

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

2st 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 EIGENMATH:

```
--- EIGENMATH : ODE type IV : linear ODE  ---
linear1(f,g, x,y, xo,yo) =
             (yo+defint(g/exp(defint(f,x,xo,x)), x,xo,x))*
                          exp(defint(f,x,xo,x))

linear1(-1,0,x,y,0,1)     -- Test with ODE y'+y=1, y(0)=1
```

▷ Click here to RUN the code.

The output is ...

```
| exp(-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. [30, 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 EIGENMATH function `VoC`, which allows to follow the steps of the semi-automatic 'pedagogical' solution process :

```
--- EIGENMATH : ODE type IV : linear ODE  ---
VoC(p,q, x,y, xo,yo)= do(
print("INTEGRATING FACTOR METHOD:"),
M = exp(integral(p,x)), -- integrating factor */
print("Step 1 - choose integrating factor M(x):", M),
My = integral(M*q,x),
print("Step 2 - multiply ODE by M and integrate: ", My),
y = (integral(M*q,x)/M + C/M),
print("Step 3 - divide result through M :", eval(y)),
c = eval(y, x, xo),
print("Step 4 - calculate C for yo=y(xo), i.e. solve yo=c for C:", c)
)

VoC(1, 1,x,y,0,1)    -- Test with ODE y'+y=1, y(0)=1
```

▷ Click here to RUN the code.

The invoke arguments of `VoC(p,q, x,y, xo,yo)` should be self-explaining.

The output of the dialog is ...

INTEGRATING FACTOR METHOD:

Step 1 - choose integrating factor M(x):

$M = \exp(x)$

Step 2 - multiply ODE by M and integrate:

$M_y = \exp(x)$

Step 3 - divide result through M :

$C \exp(-x) + 1$

Step 4 - calculate C for yo=y(xo), i.e. solve yo=c for C:

$c = C + 1$

.. and allows to calculate by hand the result $y \equiv 1$.

c. We now solve this IVP by invoking the build-in MAXIMA^{online} function `ode2`:

```
/* MAXIMA-online : linear ODE type IV by ode2 */
depends(y, x) $              /* y(x) depends on x */
e: diff(y,x) =  -y+1;        /* the given ODE */
r: ode2(e,y,x);              /* solve ODE equation e, result is r */
ic: ic1(r, x=0, y=1);        /* solve ODE respecting the IVP y(0)=1 */
solve(ic, y);                /* solve eq. ic for y(x) */
```

▷ Click here to RUN the code.

The MAXIMA^{online} output is the same as in b: `[y=1]`.

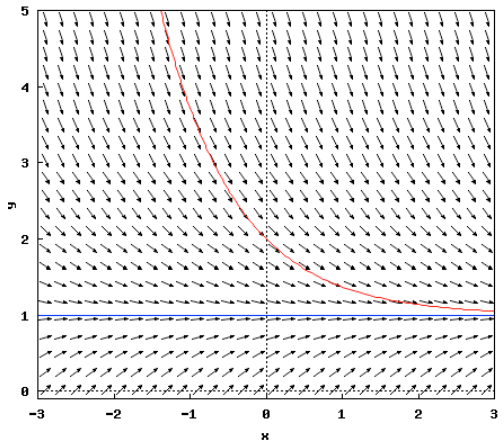


Figure 7: The direction field of the ODE $y' = 1 - y$ shows the special solution $y(x) = 1$ and the solution with IVP $y(0) = 2$.

Exercise 44. Reproduce fig. 7 with MAXIMA^{online} or CALCPLOT3D.

Exercises.

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

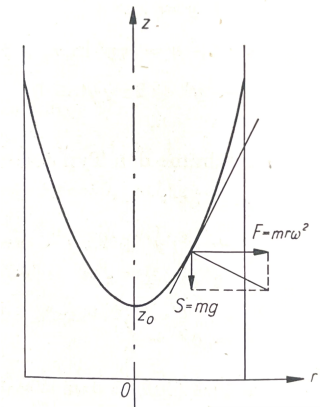
Exercise 45. Solve the ODE $y' = x + y$.

Exercise 46. Solve the ODE $y' - \frac{y}{x} = \frac{x-1}{x}$.

Exercise 47. Solve the IVP $4s\psi' - 2\psi = 3s^2,\ \psi(1) = 0$.

Exercise 48. Solve the IVP $\dot{z}\cos t + z\sin t = 1\ z(0) = 1$.

Exercise 49. A cylindrical vessel containing a liquid, rotates with the constant angular velocity ω 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 50. 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 51. Do some of the examples and exercises of
▷ DAWKINS, P.: Linear Differential Equations.

Exercise 52. Do exercise 1.4 on p.9 of ▷ AMMARI: integrating factors.

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

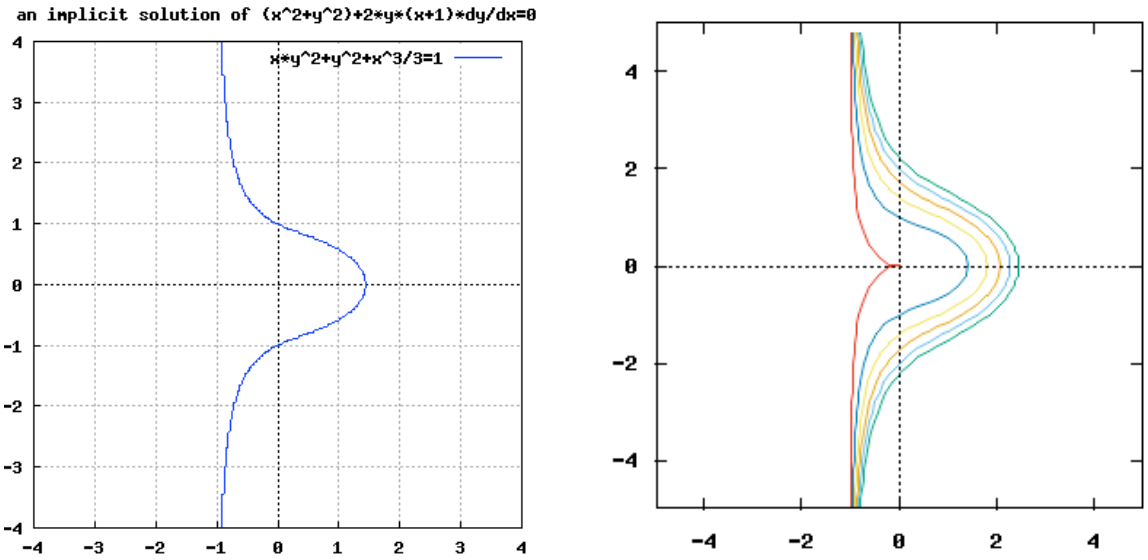


Figure 8: Left: The exact ODE $(x^2 + y^2) + 2 \cdot y \cdot (x + 1) \cdot y' = 0$ has an implicit solution $f(x, y) = C$, which is part of ..
Right: .. one of the contour lines of the contour plot of $f(x, y)$, i.e. $y(x)$ is solution, if its graph runs on an 'isocline' of same high.

• The ODE type V is: $P(x, y) + Q(x, y) \cdot y'(x) = 0$.[5] 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*.

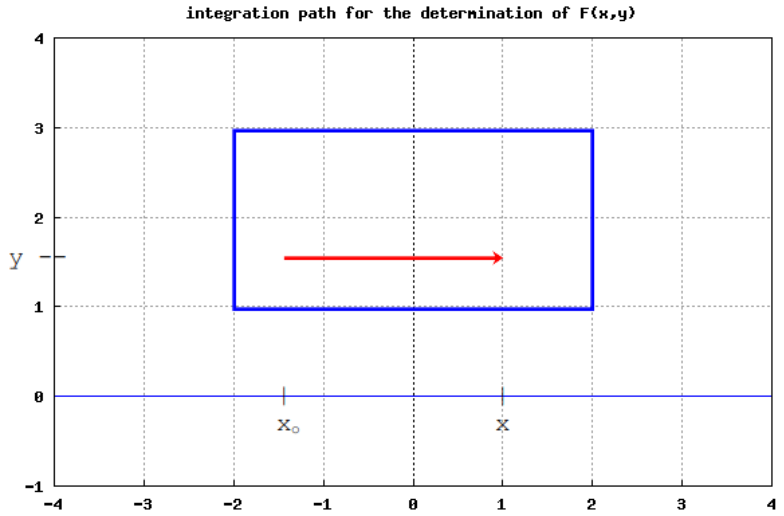
Definition. (*exact* ODE)
The ODE $P(x, y) + Q(x, y) \cdot y'(x) = 0$ is called *exact*,
iff there is a C^1-function $f(x, y)$ with $\frac{\partial f}{\partial x} \equiv f_x = P$ and $\frac{\partial f}{\partial y} \equiv f_y = Q$.

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.

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

[5] 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:

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

c. Here is an implementation of a MAXIMA^{online} 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 $F \to ... \to f$.

```
/* 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.")$

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

solve(f=1, y);          /* (4) */
draw2d( color = red,
        implicit( y=sqrt(3/(x+1)-x^3/(x+1))/sqrt(3), x,-1,1.5, y,-1,5) )$
```

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

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 MAXIMA^{online}s `funmake`[6] 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 $f(x,y)$.

▷ We test our function `exact` for the ODE $(x^2+y^2) + 2 \cdot y \cdot (x+1) \cdot y' = 0$.

▷ Click here to RUN the code.

a. In (4) we solve the implicit solution $f(x,y(x)) = 1$ for an explicit solution $y(x) = ...$
The output of this dialog is shown in the next figure on the
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 (%o5). 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.

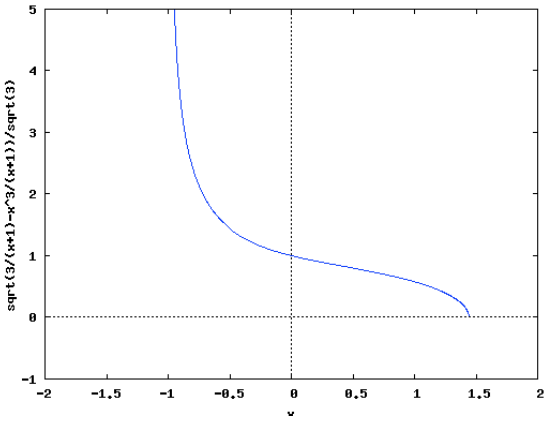
[6]=`make` an anonymous `fun`ction named A dependent only on y.

```
(%i4) exact(P,Q, x,y);
```

(%o4) $xy^2 + y^2 + \frac{x^3}{3}$

```
(%i5) solve(f=1, y);
```

(%o5) $\left[y = -\frac{\sqrt{\frac{3}{x+1} - \frac{x^3}{x+1}}}{\sqrt{3}}, y = \frac{\sqrt{\frac{3}{x+1} - \frac{x^3}{x+1}}}{\sqrt{3}}\right]$



b. We now solve this ODE by invoking the build-in function `ode2` of MAXIMA^{online}:

```
/* Maxima -- ODE type V:  EXACT ODE -- */
depends(y, x) $             /* y(x) depends on x */
e: (x^2+y^2)+2*y*(x+1) * diff(y,x)=0;       /* the given ODE */
r: ode2(e,y,x);             /* solve ODE equation e, result is r */
ic: ic1(r, x=0, y=1);       /* solve ODE respecting the IVP y(0)=1 */
solve(ic, y);               /* solve eq. ic for y(x) */
```

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

The MAXIMA^{online} output is:

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

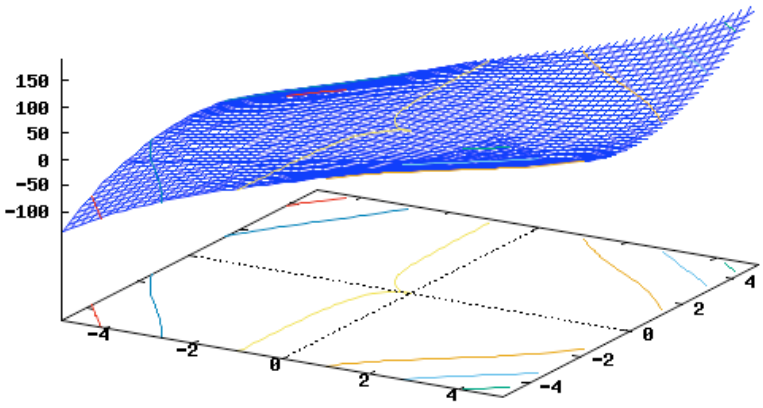
(%o65) $\frac{(3x+3)y^2+x^3}{3}=\%c$

Please observe, that MAXIMA^{online}'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 $f(x, y)$:

```
draw3d(  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 ) $
```

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



Exercise 53. Plot the implicit solution $f(x,y) = xy^2 + y^2 + x^3/3 = 1$ using EIGENMATH:
`do( xrange=(-2,2), yrange=(-1,5), draw(sqrt(3/(x+1)-x^3/(x+1))/sqrt(3),x))`

Exercises.

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

Exercise 54. Solve the ODE $2xy\,dx + x^2\,dy = 0$. Cf. [30, p.89].

Exercise 55. Solve the ODE $y' = -\frac{3x^2+4xy}{2x^2+2y}$.

Exercise 56. Solve $(xe^y + \cos(y))y' = -e^y$. Cf. [16, p.61].

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

Exercise 57. Solve the ODE $2x - y + (4y - x)\frac{dy}{dx} = 0$.

Exercise 58. Solve the ODE $e^x(1 + e^y) = -e^y(1 + e^x)y'$.

Exercise 59. Solve the ODE $(\frac{x}{y} - 2y)y' = 2x - \ln y$.

Exercise 60. Do some of the examples and exercises of [28, p.63 ff].
You will find the solutions at ▷ BAZETT: Exact equations.

Exercise 61. Do some of the examples of ▷ DAWKINS: Exact Equations.

Exercise 62. Read §1.3.2 and do examples 1.12 – 1.14 on p.11 of ▷ AMMARI: Exact equations.

Exercise 63. Read about isoclines and do some of the problems of ▷ FAN/CIFARELLI: Isoclines.

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

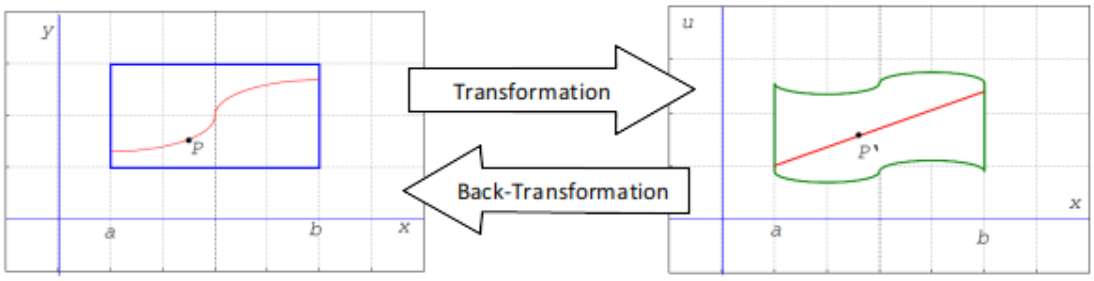


Figure 9: Sometimes the introduction of new coordinates may transform a given ODE in an equivalent, but better hand-able form. Often one transforms ('substitutes') the dependent variable y into a new one u, then solves the transformed ODE and transforms this u-solution back to get a solution y of the 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 8. 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:

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

c. (MAXIMA^{online}) Here is a small script to run this solution method in MAXIMA^{online}. It goes along the steps (2.5) to (2.6).

```
/* --- MAXIMA-online  : 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^{online} 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 $du = ...$ 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.

▷ Click here to RUN the code.

The output of the dialog is ...

(%o61) $f(x,y) := \frac{y+x}{x}$

(eq1) $\frac{du\,x}{dx} + u = \frac{u\,x+x}{x}$

(%o63) $[du = \frac{dx}{x}]$

(eq2) $\log(x) = u$

(eq3) $\log(x) = \frac{y}{x}$

(%o67) $[y = x\log(x)]$

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

d. We solve this ODE by invoking the build-in MAXIMA^{online} function `ode2`:

```
/* --- Maxima : ODE type VI  HOMOGENOUS ODE ---*/
depends(y, x) $              /* y(x) depends on x */
e: diff(y,x)=(y+x)/x;        /* the given ODE */
r: ode2(e,y,x);              /* solve ODE equation e, result is r */
method;
```

▷ Click here to RUN the code.

The MAXIMA^{online} output is:

```
(%o71)  d/dx y=(y+x)/x
(%o72)  y=x(log(x)+%c)
(%o73)  linear
```

Please observe, that MAXIMA^{online}'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 64. Solve the ODE $x^2y' = y^2 + xy,\ y(1) = 1$.
Choose the substitution $u = y/x$. Cf. [28, p.46].

Exercise 65. Solve the ODE $y' = (x - y + 1)^2$.
Choose the substitution $u = x - y + 1$. Cf. [28, p.49].

Exercise 66. Do some of the exercises of [28, p.46 ff].
You will find the solutions at ▷ BAZETT: Substitutions.

Exercise 67. Do some of the examples and exercises of ▷ DAWKINS: Substitutions.

Exercise 68. 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 MAXIMA^{online}s `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^{online} function `isolate(expr,x)`. Maybe it is of some help ..

Exercise 69. (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. [25, p.91].

Exercise 70. 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 EIGENMATH 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	$f(\mathbf{1.415}) = 0.5 * (\mathbf{1.415} + 2/\mathbf{1.415}) \approx 1.4142$

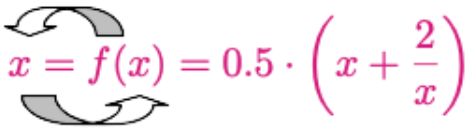
So we have the following recurrence (sequence) process, which after 5 iteration steps gives $\sqrt{2} \approx 1.4142$:

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

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.



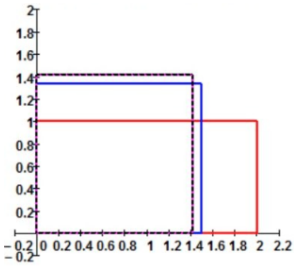


Figure 10: *Left:* *Recurrence*: The result $f(x)$ for its 'old' input x is substituted in f again and therefore recurs as new input x, i.e. in sequence language: $f(x_n) = x_{n+1}$.
Right: *Visualization*: the starting red rectangle with length $x = 2$ and width $y = 1$ is transformed in 3 steps to 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 an '2-dimensional indexed' function and recur on the index::

```
--- EIGENMATH : recurrence by recursion ---
f(x) = 0.5*(x+2/x)                        -- recurrence formula
y(n,x)= test(n = 0, 3, eval( f( y(n-1,x) )))
--                  |              |
--                start value      |                    f
--                                 recursion:  y(n-1)  ----->  y(n)

y(5,x)       -- calculate test value
```

▷ Click here to RUN the code.

The EIGENMATH output:

```
|    1.4142
```

3.2 iterate by `iterate`

We now abstract the recurrence pattern from Fig.10 $x_0 \overset{u}{\rightsquigarrow} x_{new} \overset{u}{\rightsquigarrow} x_{newnew} \dots$ to a EIGENMATH 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`

```
--- EIGENMATH : ITERATE ---
iterate(u,X,Xo,n) = do( M = zero(2,n+1),                              --(1)
                        M[1,1] = Xo,                                  --(2)
                        for(i,2,n+1, M[1,i] = eval(u, X,M[1,i-1])),   --(3)
                        M)                                            --(4)

iterate( 0.5*(x+2/x), x, 3, 4)            -- (5)  test: calculate sqrt(2)
```

▷ Click here to RUN the code.

The EIGENMATH output reproduce in (7) the results in §3.1:

$$\begin{bmatrix} 3 & 1.83333 & 1.46212 & 1.415 & 1.41421 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

[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 EIGENMATH to the DERIVE literature.

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) defines a 2-by-(n+1) matrix M as a container to hold all calculated values. M consists originally of zeros. In (2) the 1st entry of M is feed with the value `Xo`. In (3) `eval(u,X,M[1,i-1])` means 'substitute the previous entry $M[1, i-1]$ for x in $u(x)$'. The `for`-loop starts at $i = 2$, because the first place is already occupied. In (3) we also fill the slot `M[1,i]` with the updated $u(..)$ and put this new value in the i^{st} slot $(1, i)$. The whole result (matrix) is returned (4) in M. The iteration process itself is done by the for loop in (3) and repeats $n + 1 - 2 = n - 1$ times.

In (5) the test `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`.

Remark. In our function `iterate` the identifiers M is *global* and therefore accessible outside of the function. If you want to have it *local* in the function `iterate` you may write `iterate(u,X,Xo,n, M)` and let the rest of the definition as it is.
The call is the same: `iterate(u,X,Xo,n)`.
There are maximal 9 local variables allowed.

Exercise 71. 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 72. Write a function `power(x,n)= iterate(...)` to compute the n-th power of a number x.

Exercise 73. Write a function `factorial(n)= iterate(.)` to compute $n!$ of a number n.

3.2.2 the two-dimensional function `iterate2`

```
--- EIGENMATH : ITERATE2  ---
iterate2(u,X,Xo,n) = do( M = zero(n+1,2), M[1] = Xo,
           for(i,2,n+1, M[i] = eval(u, X[1],M[i-1,1], X[2],M[i-1,2])),
           transpose(M))

-- test on HERON algorithm for sqrt(2)
 iterate2( (n+1, 0.5*(x+2/x)), (n,x), (1,3), 4)
 --            u                 X      Xo   n

 transpose(M)                                          -- (1)
```

▷ Click here to RUN the code.

• The EIGENMATH output displays the results of §3.2.1, supplemented by the counter in the first slot:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 1.83333 & 1.46212 & 1.415 & 1.41421 \end{bmatrix}$$

• Code line (1) allows to give the output of the list M in table shape by `transpose`'ing the matrix `M`, which is global defined and therefore accessible outside of `iterate2` :

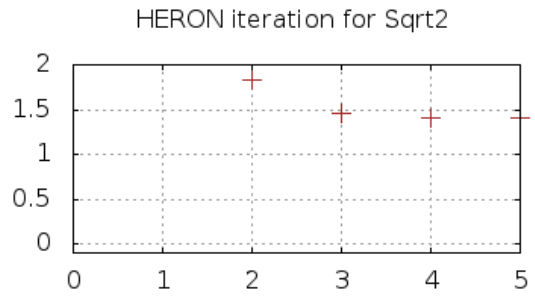
$$\begin{bmatrix} 1 & 3 \\ 2 & 1.83333 \\ 3 & 1.46212 \\ 4 & 1.415 \\ 5 & 1.41421 \end{bmatrix}$$

You may also write `transpose( iterate2( (n+1, 0.5*(x+2/x)), (n,x), (1,3), 4) )`

Comment. Function `iterate2` is very similar to `iterate`. But the variables `u,X,Xo` are now lists of each 2 elements ('pairs, points') in the updating process, which has to be updated both in the variable $X = (x, y)$, i.e. the variables of u.

• The coordinates in M also allow to plot the list `M=Sqrt2` as a graphical point list using MAXIMA^{online}:

```
/* --- MAXIMA-online --- */
draw2d ( title="HERON iteration for Sqrt2",
         user_preamble = "set size ratio -1",  /* both axis equal scale */
         grid  = true,
         xrange = [0,5], yrange = [-0.1,2],
         /* point_type    = circle, */
         point_size    = 2,
         color         = brown,
         /* points_joined = true, */
         points( [1,2,3,4,5], [3,1.833,1.462,1.415,1.414] ) ) $
```



▷ Click here to RUN the code.

Exercise 74. 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

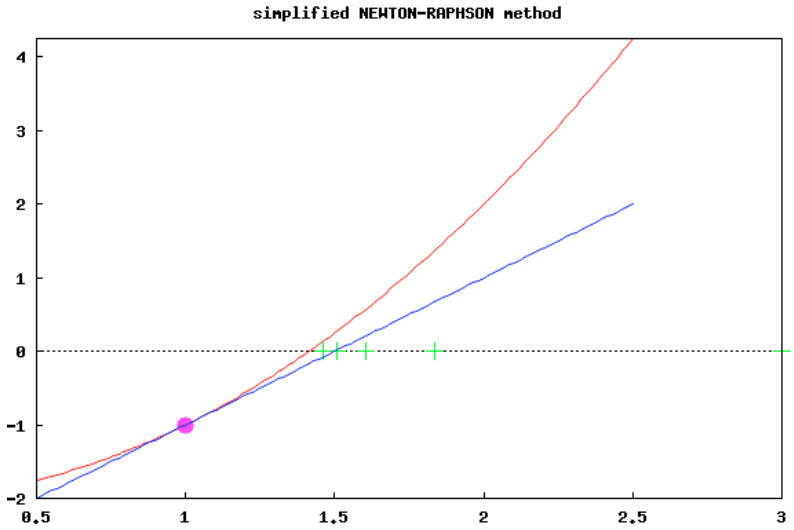


Figure 11: *Red:* The graph of $f(x) = x^2 - 2$. Where is the root of f?
Blue: The tangent on Graph(f) at point $P = (1, -1)$.
Graphically the root ξ of $f(x) = 0$ is located where the graph of f crosses the x-axis, i.e. $\xi \approx 1.4$.
Idea of NEWTON-RAPHSON method: If we know - e.g. by looking at the graph of f on $[a, b]$ - that a root ξ 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 ξ 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 handish or by using MAXIMA[online]

```
solve(0 = f(x0) + c*(x-x0), x), expand;
```

(%o1) $\left[x = x_0 - \frac{f(x_0)}{c}\right]$

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:[9]

```
--- EIGENMATH : simplified NEWTON-RAPHSON iteration  ---
---             include our function iterate(..) here

newton1(u,x,a,n) = do( c = float(eval(d(u,x),x,a)),        -- (1)
                       iterate( x-u/c, x, a, n) )          -- (2)

newton1( x^2-2 , x,1, 9)[1,9]   -- test: calculate sqrt(2)  --(3)
```

Output:

| 1.41402

▷ Click here to RUN the code.

Comment. In (1) we translate $c := f'(a)$ into EIGENMATH 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 `iterate`. The test in (3) returns only the last entry, because of the access via $[1, 9]$.

Example 9. Here we demonstrate a invocation of `newton1`.

```
f(x) = x^2-cos(x)
newton1(f,x, 1.0, 5)          --(4)
```

▷ Click here to RUN the code.

Output:

$$\begin{bmatrix} 1 & 0.838218 & 0.826316 & 0.824483 & 0.824189 & 0.824141 \end{bmatrix}$$

Remark. We have not shown the 0-line of the result, because it has no info for us.
Be warned: if you use $a = 1$ instead of $a = 1.0$ in (4) the output is not simplified: $cos(1)$ is a symbolic result and the iteration does not stop - you get a system overflow error.

Exercise 75. 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 EIGENMATH function

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

```
--- EIGENMATH : NEWTON method ---
---             include our function iterate(..) here

newton(u,x,a,n) = iterate( x - u/d(u,x), x,a, n)
```

a. Run `newton(x*x-2,x, 1.0, 5)` and `newton(`x^2`-cos(x)),x, 1.0, 5)`.
Compare the results with (3) and (4).

b. Look at ▷ WIKI: NEWTON method. and watch the animation of the changing tangents.

Exercise 76. Reproduce Fig.11 with this code:

```
/*** MAXIMA-online  ***/
f(x):= x*x-2;  a: 1;  b: 2;
m: at(diff(f(x),x),x=a);
X: makelist(0, i,1,9);
Y: [1, 1.5, 1.375, 1.429, 1.4077, 1.4169, 1.4131, 1.4147, 1.414];
XY: makelist([Y[i],X[i]], i,1,9);
print(XY);
draw2d( title="simplified NEWTON-RAPHSON method",
    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)
   )$
```

▷ Click here to RUN the code.
Think about every line of the code and its purpose.

Exercise 77. Here is an direct implementation of the NEWTON method without `iterate`:

```
newton0(u,x,xo,n,  M,VAL) = do( M  = zero(2,n),
                        VAL=eval(u,x,xo),
                        for(i,1,n,   VAL   = eval(x-u/d(u,x),x,VAL),
                                     M[1,i]= VAL),
                        M[1,n])                          -- (1)

newton0( x^2-2 ,x,2, 9)
float(newton0( x^2-2 ,x,2, 9))

f = x^2-cos(x)
newton0(f,x, 1.0, 5)
```

▷ Click here to RUN the code.
a. Discuss pros and cons of this implementation w.r.t. `newton`.
b. What, if you write `M[1]` or `M` instead of `M[1,n]` in (1). Test it.

3.3.2 simplified Secant method

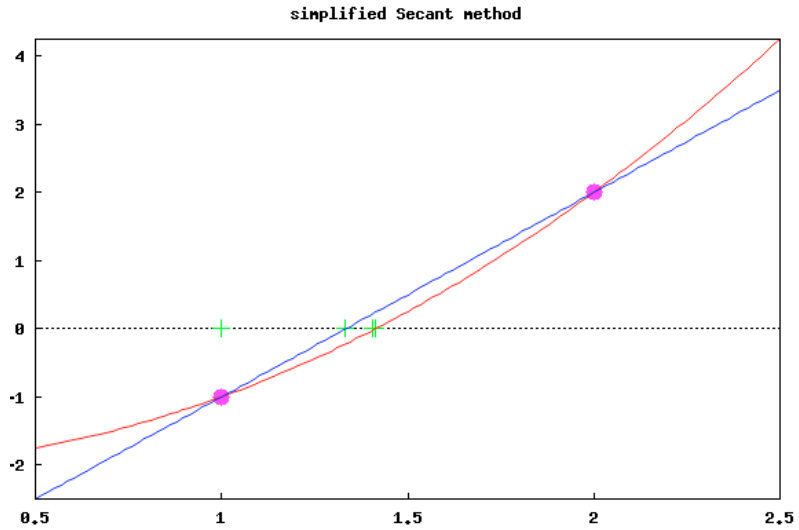


Figure 12: *Red:* The graph of $f(x) = x^2 - 2$. Where is the root of f? Graphically the root ξ of $f(x) = 0$ is located where the graph of f crosses the x-axis, i.e. at $x \approx 1.4$.
Idea of Secant method: If we know - eg by looking at the graph of f on $[a, b]$ - that a zero ξ of f exists between 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`:

```
--- EIGENMATH : simplified SECANT METHOD   ---
--           include function iterate(..) here
secant1(u,x, a,b, n) = do( C= (a-b)/(eval(u,x,a)-eval(u,x,b)),  -- (1)
                           iterate( x - C*u, x,a,n))            -- (2)

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

```
f = x^2-cos(x)
secant1( f, x, 1.0,2.0, 5)
```

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

▷ Click here to RUN the code.
Output:

$$\begin{bmatrix} 1 & 1.33333 & 1.40741 & 1.41381 & 1.41419 & 1.41421 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.883811 & 0.846678 & 0.832931 & 0.827607 & 0.825511 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Exercise 78. (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 ξ between a and b.
Here is function `signChange`, that checks, if there is a sign change on the interval $[a, b]$:

```
signChange(a,b) = test( f(a)*f(b) < 0, true, false)

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

Test this function.

Exercise 79. (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 paper'n pencil or MAXIMA^{online} to show: $x = \frac{a \cdot f(b)-b \cdot f(a)}{b-a}$.

c. Implement an EIGENMATH function `iterate3(u,X,Xo,n) = do( ...)`, which operates on lists of 3 elements.

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

```
--- EIGENMATH : general SECANT METHOD   ---
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)
```

Test your functions with

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

The output should be:

```
[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 80. Reproduce Fig.12 with this MAXIMA^{online} code:

```
/* Maxima-online : SECANT method  */
f(x):= x*x-2;  a: 1;  b: 2;
m:(f(a)-f(b))/(a-b);
X: makelist(0, i,1,6);
Y: [1, 1.333, 1.4, 1.411, 1.4138, 1.4141];
XY: makelist([Y[i],X[i]], i,1,6);
print(XY);
draw2d(xaxis = true, grid=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")$
```

▷ Click here to RUN the code.
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 [34, 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 10. 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 81. Determine $y''''(x)$ for the ODE $y' = xy$.

3.4.2 idiff.

Exercise 81 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 us define a function `idiff` ('implicit' diff) to do this work for us. We use the recursion (3.6) for the recurrence term.

Step **1** We recalculate example 10 with EIGENMATH to get a feeling how to abstract the calculation.

```
--- EIGENMATH  : stepwise implicit differentiation ---
f(x,y)=x*y
F1 = d(f,x)+d(f,y)*f
F1
F2 = d(F1,x)+d(F1,y)*f
F2
F3 = d(F2,x)+d(F2,y)*f
F3

f(x,y)=x*y
u = d(f,x)+d(f,y)*f          --(1)
u = d(u,x)+d(u,y)*f          --(2)
u = d(u,x)+d(u,y)*f
u
```

▷ Mark-Copy and Click here to paste and RUN the code.

In (1) we start by using the RHS f for u and get a new u. In (2) we begin iterating by substituting the previous u in `differentiate(.)` getting again a new u. etc.

We get the results of example 10, ok.

$$F_1 = x^2 y + y$$

$$F_2 = x^3 y + 3\,x\,y$$

$$F_3 = x^4 y + 6\,x^2 y + 3\,y$$

$$F_3 = x^4 y + 6\,x^2 y + 3\,y$$

$$u = x^4 y + 6\,x^2 y + 3\,y$$

Step **2** We now abstract the procedure (1)&(2) to a general function `idiff`:

```
--- EIGENMATH : idiff = implicit differentiation --- */
idiff(f,x,y,n,  u) = do(
                     M = zero(2,n),                                  --(1)
                     u = d(f,x)+d(f,y)*f,                            --(2)
                     M[1,1]=f,                                       --(3)
                     M[1,2]=u,                                       --(4)
                     for( i,3,n, u = d(u,x)+d(u,y)*f, M[1,i]=u),     --(5)
                     M[1])                                           --(6)

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

▷ Mark-Copy and Click here to paste and RUN the code.

We get again the results of example 10, ok:

$$\begin{bmatrix} x\,y \\ x^2\,y + y \\ x^3\,y + 3\,x\,y \\ x^4\,y + 6\,x^2\,y + 3\,y \end{bmatrix}$$

- 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. `n`: the number of repeated substitutions of new functions u according to (3.6).
  4. `u`: a *local* variable for the recurrence process, not needed in a function call. To indicate it, we have separated u from the other variables with an extra whitespace in the function's variables list.

Comment. In (1) we define a container (matrix) M to catch the intermediate resulting functions u. (2) is the initial start of the recurrence, giving the first u. These f and u are saved in the first slots of M, see (3) and (4). The for-loop uses the recurrence term `d(u,x)+f*d(u,y)`, which corresponds to $\frac{\partial u}{\partial x} \cdot 1 + \frac{\partial u}{\partial y} \cdot f(x,y)$, i.e. is the translation of (3.6) to EIGENMATH. Each new calculated u is saved at slot position i by $M[1,i] = u$, see (5). In (6) only the 1st row of M is returned (the other entries are 0).
The output displays all implicit derivatives starting with f until y''', i.e. the list $[y, y', y'', y''']$.

- In case you only want to look at the result of exercise 75, do `idiff(x*y,x,y,4)[4]`

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 $y'(x_k), y''(x_k), y'''(x_k), ...$ *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 `d(iff)`.

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

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

The LHS list is the list of `idiff`'s of y and the RHS list is $\{h^r/r!|r = 1..\}$.
To construct the RHS list we write a helper function `makelist`. Then we code (3.9).

- (3.9) motivates to write the following function `iTaylor`, cf. [17, p.208]:

```
--- EIGENMATH : IMPLICIT TAYLOR METHOD of order m ---
--  include our function idiff(f,x,y,m) here

makelist(u,i,m,n) = do( M   = zero(2,n),
                        VAL = eval(u,i,m), M[1,m] = VAL,
                        for( j,m+1,n, VAL = eval(u,i,j), M[1,j]= VAL ),
                        M[1])

iTaylor(f,x,y, h, m) = y + dot( idiff(f,x,y,m), makelist(h^r/r!, r,1,m) )

-- Test: iTaylor on the IVP y'=xy with y(0)=1
makelist(h^r/r!, r,1,3)
iTaylor(x*y, x,y, h,3)
iTaylor(x*y, x,y, 1/5, 3)
```

▷ Mark-Copy and Click here to paste and RUN the code.
The output displays all `Taylor` method order 3 terms with its implicit derivatives

$$\frac{1}{6}h^3x^3y + \frac{1}{2}h^3xy + \frac{1}{2}h^2x^2y + \frac{1}{2}h^2y + hxy + y$$

$$\frac{1}{750}x^3y + \frac{1}{50}x^2y + \frac{51}{250}xy + \frac{51}{50}y$$

For the IVP the command `eval(last, x,0, y,1)` gives `1.02`.

Exercise 82. Look at the following EIGENMATH script:

```
--- EIGENMATH : IMPLICIT TAYLOR METHOD of order m ---
--  include our functions iterate2, idiff, makelist, iTaylor here

iterate2((x+0.2, iTaylor(x*y, x,y, 0.2, 3)), (x,y), (0,1), 5)       --(1)
```

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

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 1 & 1.02 & 1.08284 & 1.19642 & 1.37575 & 1.64633 \end{bmatrix}$$

Explain: what does command (1) do?

Exercise 83.
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 [44, 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)

Exercise 82 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 now abstract this procedure to a function `iTaylorsol`, which iterates the TAYLOR recurrence relation for given data to establish a fully automatic TAYLOR *solution method* of a prescribed order[10]:

```
-- EIGENMATH : TAYLOR SOLUTION METHOD for ODE y'=f(x,y) ---
-- include iterate2(u,x,xo,n), idiff(f,x,y,m), iTaylor(f,x,y,h,m) here

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

The arguments of `iTaylorsol(f, x,y, xo,yo, h, m, n)` are:

1. `f`: the right-hand side of the ODE $y' = f(x, y))$, this time given global outside
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 11. Let's check `iTaylorsol` for the IVP $\begin{cases} y'=xy \\ y'(0)=1 \end{cases}$:

```
-- Test on ODE y'=xy withy(0)=1
f(x,y) = x*y
iTaylorsol( x,y, 0,1, 0.2, 3, 5)
```

EIGENMATH output:

▷ Click here to RUN the code.

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 1 & 1.02 & 1.08284 & 1.19642 & 1.37575 & 1.64633 \end{bmatrix}$$

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

Exercises.

Exercise 84. Someone wrote this piece of EIGENMATH 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)

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

▷ Click here 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 85. Use a Taylor method of order 3 to solve the IVP $y' = x - y^2,\ y(0) = -0.5$. Compare with exercise 4.

Exercise 86. 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 87. 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 88. 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 89. 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: $y(x) = \tan(x)$

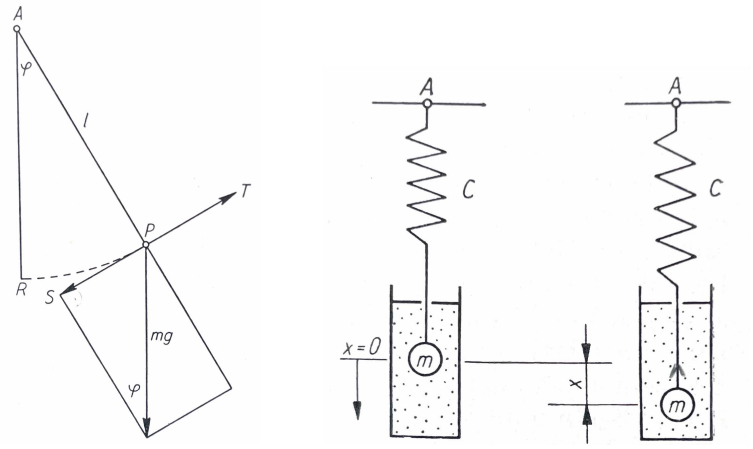
Exercise 90. An ODE is given through $x' = \sin(t) - x$ with IC $x(0) = 1$.
Do a Taylor method of order 3 to approximate the solution x. Compare with exercise 9.

Exercise 91. 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 92. 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.



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 ℓ. 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: $-\mathbf{m}\ell\ddot{\varphi} - \mathbf{mg}\sin\varphi = \mathbf{0}$."

Figure 13: *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`, 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], [30, p.101 ff] and [44, p.228 ff].
Error analysis for the methods are left to the literature, e.g. [30], [1], [48], [20].

4.1 RK1 alias the EULER Method

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

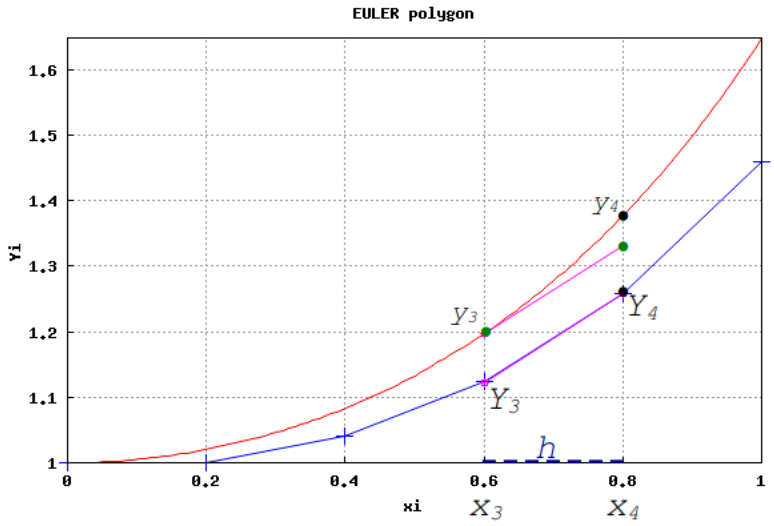


Figure 14: *Red:* $y(x) = exp(\frac{x^2}{2})$ is the exact solution of ODE $y' = x \cdot y,\ y(0) = 1$.
Blue: We try to follow the graph of y approximately by a polygon.
Idea of EULER method: If we are at a point (x_3, Y_3) on the approximating polygon near the exact point (x_3, y_3) on the solution, 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:

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

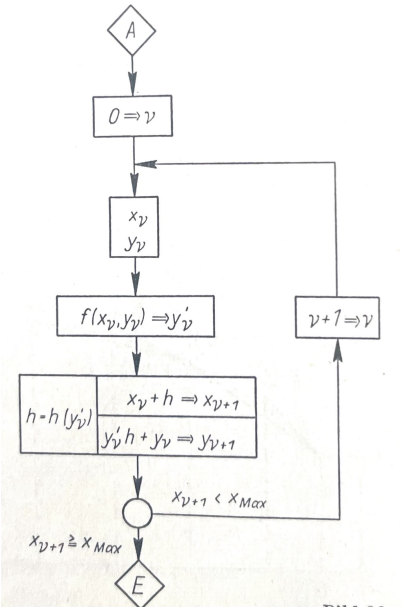
Formula (4.3) is called the EULER *recurrence formula*.

Remark.

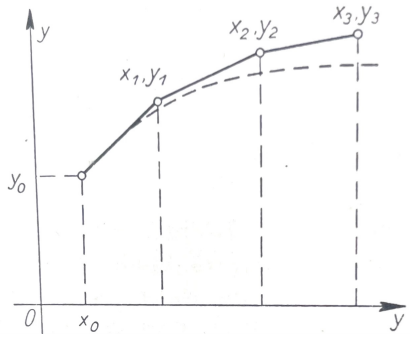
1. In equation (4.3) we denote the approximative values on the polygon with uppercase letters $Y..$ and the exact values of the solution function $y(x)$ with lowercase letters $x.., y..$ in order to distinguish between them.

2. For more information about the EULER method and derivations of (4.3) see e.g. ▷ Wiki: EULER method - Derivation or ▷ Cheever: EULER method, [48, 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 12. 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 EIGENMATH:

```
ye= exp(x^2/2)      -- exact solution  ye
ye
yp= d(ye, x)        -- their slope ye
yp
```

▷ Click here to RUN the code.

Tho output of EIGENMATH is

$$y_e = \exp\left[\frac{1}{2}x^2\right]$$

$$y_p = x \exp\left[\frac{1}{2}x^2\right]$$

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:

```
--- EIGENMATH ---
E(x,y) = (x + 0.2, y +0.2*x*y)

E(0.0, 1)
E(0.2, 1)
E(0.4, 1.04)
print(0.4+0.2, 1.04+0.2*0.4*1.04)  -- example calc by hand
E(0.6, 1.1232)
E(0.8, 1.257984)
```

▷ Click here to RUN the code.

```
[0.2,1.0]
[0.4,1.04]
[0.6,1.1232]
[0.8,1.258]
[1.0,1.4593]
```

We may now reproduce the plot of Fig.13 e.g. with MAXIMA^{online}:

```
/* ** MAXIMA online ** */
XY: [[0,1],[0.2,1],[0.4,1.04],[0.6,1.1232],[0.8,1.257984],[1,1.4593]];
draw2d(xaxis = true, grid=true,
    xrange = [0,1], yrange = [1,1.6],
    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 here 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 $(x + h, y + h * f(x,y))$:[11]

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

```
f(x,y) = x*y
xo = 0
yo = 1
h = 0.2
n = 5
iterate2( (x+h,  y+h*f(x,y)),  (x,y),  (xo,yo), n)
```

▷ Click here 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 EIGENMATH function `EULER` to fully-automate the calculation process. We check the correctness by repeating the handish calculation in 1. in (1).

```
--- EIGENMATH : EULER method  ---
EULER( x,y, xo,yo, h, n) =  iterate2((x+h, y+h*f(x,y)), (x,y), (xo,yo), n)

--   Test of ODE  y'= xy with y(0)=1
f(x,y) = x*y
EULER(x,y, 0.0, 1.0, 0.2, 5)           --(1)
--   ( x,y, xo,  yo,  h,   n)
```

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

- 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 and outside
  2. `x,y` : the variables used in the iteration (coordinates) 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 a little bit 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) the step-size h is added at each step. The corresponding y slot of (x, y) is replaced by $y + h \cdot f(x, y)$. 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:`

```
---  EULER method denoted as RK1  ---
RK1(x,y, xo,yo, h, n) = do( k1= f(x,y),
                            iterate2((x+h, y+h*k1),(x,y),(xo,yo),n))
-- Test:
f(x,y) = x*y
RK1(x,y, 0.0, 1.0, 0.2, 5)      -- (1)
```

Exercises.

Exercise 93. 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 94. (*the global resp. local error of the EULER method*)
Study the following code snippet:

```
--- EIGENMATH : local resp. global error ---
--  include iterate2 and EULER here
-- function max contributed by G. Weigt:
max(L, MAX,i) = do( MAX = L[1],
                    for(i,2,dim(L), test(L[i]>MAX, MAX=L[i])), MAX)

second(z) = z[2]
ye(x)  = exp(x^2/2.0)                    -- explicit solution for ode
f(x,y) = x*y
Pe = EULER( x,y, 0,1, 0.2, 5)            -- Euler method points
L1 = second(Pe)
L2 = (ye(0.0), ye(0.2), ye(0.4), ye(0.6), ye(0.8), ye(1.0) )   --(1)

globalError = abs(L1-L2)     -- global error i.e. sum of absolute values
globalError

localError = max( max(L2-L1), max(L1-L2) )    --- (2)
localError
```

▷ Click here to RUN the code.

a. Explain each code line in your own words.

b. Write a EIGENMATH function `localError(.)`. – See e.g. [48, p.7], [20, p.3].

c. Write a EIGENMATH function *truncation error* `truncError(.)`, which relates the global error to the mesh count n, i.e. the number of steps. See e.g. [48, p.7].

d. The exact function values of *ye* are listed in (1) in a "hard-coded" way. Here is the start for an automatic calculation of a function values table. Enhance it:

```
Table = zero(6,2)
for(i,0,5, Table[i+1]=(i, float(ye(0+i*0.2)) ))
Table = transpose(Table)
Table[2]
```

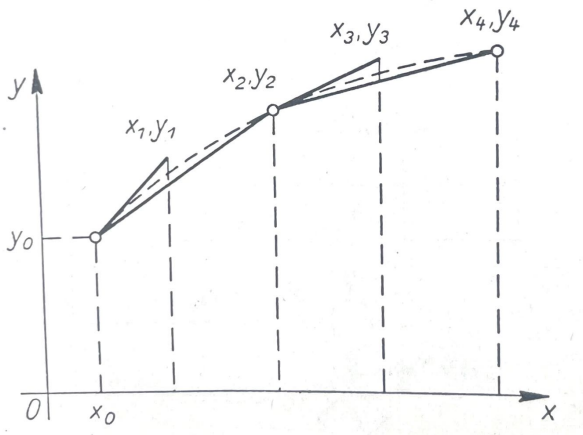
Exercise 95. **a.** Use Euler's method to approximate the solution of $y' = \frac{3t^2}{2y},\ y(0) = 1$ on the interval $0 \leq t \leq 2$ using $n = 2, 4, 8$ steps, see [44, p.230].
b. Find the local error between your approximations and the exact solution $y = \sqrt{1 + t^3}$.

Exercise 96. **a.** Use Euler's method to approximate the solution of the IVP
$y' = y^2 \sin(2x),\ y(0.5) = 1$ on the interval $0.5 \leq x \leq 3.5$ 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 97. **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 98. 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[12] solves example 18 semi-automatic:

```
f(x,y) = x*y
k1 = f(x,y)
k2 = f(x+0.2,y+0.2*k1)
iterate2( (x+0.2,  y+0.2/2*(k1+k2) ),  (x,y),  (0,1), 5)
```

▷ Click here to RUN the code.

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

```
--- EIGENMATH :  modified EULER method ---
modiEULER(x,y, xo,yo, h,n,   k1,k2) = do(
                  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)
```

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

▷ Click here 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 93..98 using `modiEULER`.

c. Calculate the local/global/trunc errors of the modified EULER method for the ODE of Exercise 93. 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 EIGENMATH:

```
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)     -- testing ..
```

▷ Click here 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 93. What do you observe?

Exercise 99. Solve $y'(x) = \frac{1}{x} - \sin(x)$ and $\sqrt{1-x^2} \cdot y'(x) = 1$ approximately.

Exercise 100. Look at this code:

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

a. What is the discussed IVP?
b. Find the exact solution y_e of the ODE.
c. Plot the scene.
d. What about the truncation error of the method in this example?

Exercise 101. 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.

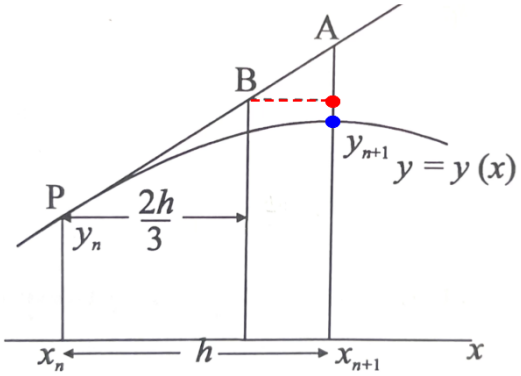


Figure 15: The modified EULER recurrence was written as $Y_{n+1} = Y_n + \frac{h}{2}(k_1 + k_2)$. Idea: maybe we get a better approximation, if we weight the average of k_1 and k_2 towards k_2 in the ratio of 3:1 instead of 1:1, see Fig.14. A is the approximative value Y^E_{n+1} for y_{n+1} using EULER's 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 EIGENMATH implementation of the HEUN *recurrence formula* is:

```
--- EIGENMATH :  HEUN's method ---
HEUN(x,y, xo,yo, h,n,  k1,k2) = do(
        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 here to RUN the code.
The output is

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 1 & 1.02 & 1.08242 & 1.195 & 1.37233 & 1.63912 \end{bmatrix}$$

Exercise 102. 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 103. 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 104. Calculate the local resp. global error of the HEUN method for example 103. Look at exercise 92.
What is the corresponding truncation error?

Exercise 105. 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 106. Investigate the 'critical' IVP example in fig.1. of ▷ HAIRER/LUBICH: p.2.
Use `EULER` and `modiEULER` and `HEUN`.

Exercise 107. 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 108. 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.

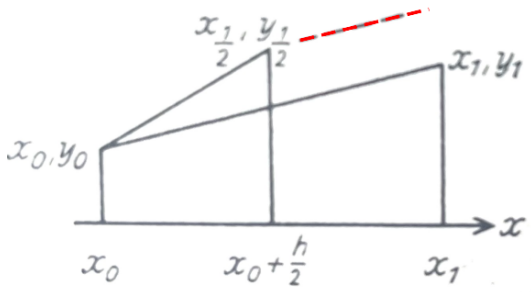


Figure 16: For the ODE $y' = f(x,y),\ y(x_0) = y_0$ one calculates $y'_0 = f(x_0, y_0)$ and walks in this direction to point $(x_{\frac{1}{2}}, y_{\frac{1}{2}}) := (x_0 + \frac{1}{2}h, y_0 + \frac{1}{2}hy'_0)$. For this point the new slope $y'_{\frac{1}{2}}$ (dashed) is calculated with $f(x,y)$ and one 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 EIGENMATH implementation of the MIDPOINT *recurrence formula* is:

```
MIDPOINT(x,y, xo,yo, h,n) =
              iterate2( (x+h,  y+h*f(x+h/2, y+h/2*f)),
                        (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 here to RUN the code.
The output is

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 1 & 1.02 & 1.08242 & 1.195 & 1.37233 & 1.63912 \end{bmatrix}$$

Exercises.

Exercise 109. 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 110. Calculate the local resp. global error of the MIDPOINT method for example 109. Look at exercise 92.
What is the corresponding truncation error?
Compare this error with that of the EULER and HEUN methods.

Exercise 111. A RUNGE–KUTTA (RK) like formulation of the midpoint method using slope constants to structure the calculation is

```
--- EIGENMATH : MIDPOINT2 method alias RK2 ---
MIDPOINT1( x,y, xo,yo, h,n,  k1,k2) = do(
                k1 = f(x,y),
                k2 = f(x+h/2, 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?)

```
--- EIGENMATH : MIDPOINT method ---
MIDPOINT2(x,y, xo,yo, h,n) =
       iterate2( (x+h,
                  y+h/2*(f(x,y)+f(x+h, y+h))),
                  (x,y), (xo,yo), n)

-- Test with ODE  y'= xy with y(0)=1
f(x,y)=x*y
MIDPOINT2(x,y, 0,1, 0.2, 5)
```

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].

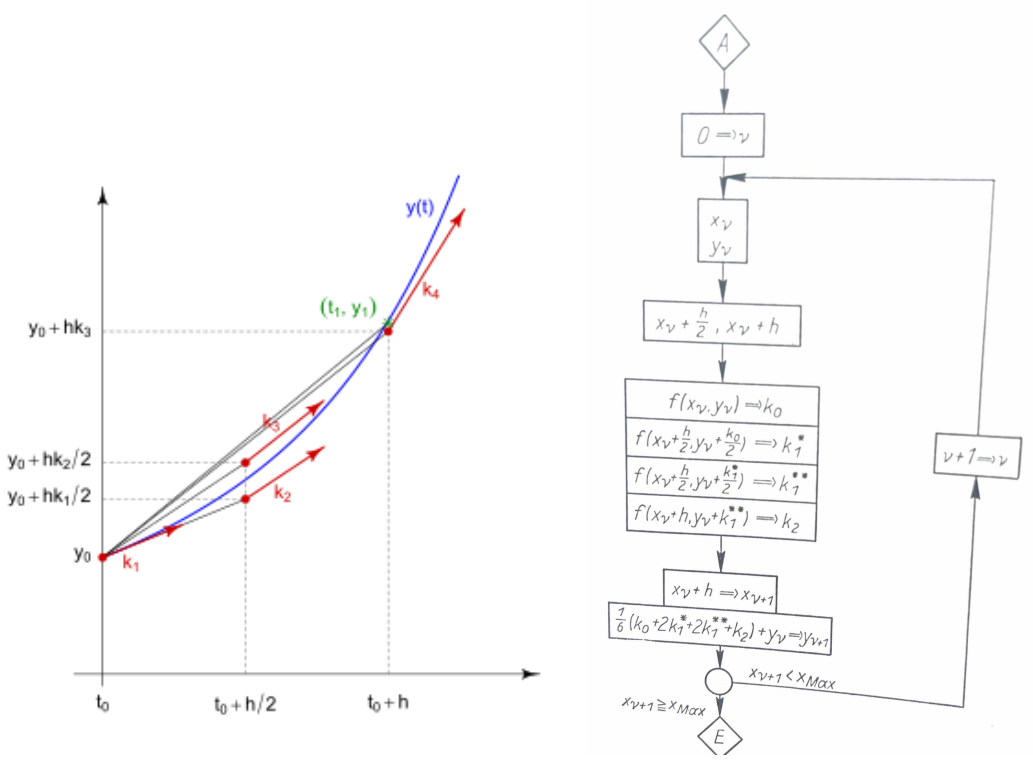


Figure 17: A RUNGE–KUTTA step for the ODE $y' = f(t, y)$ starting at (t_0, y_0).
Red: Use four different slopes at different points near the solution.
Green: New (green) slope as weighted mean of the 4 red slopes.
This process is then repeated, now starting at (t_1, y_1^R), a R(unge)-point.

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 [36, p.98] and [17, p.201] or ▷ SÜLI: p.13–19 or ▷HELLEVIK: RK 4th.

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

```
--- EIGENMATH : classic RUNGE-KUTTA  method of order 4 ---

RK4(x,y,xo,yo,h,n) = do(
                     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/6.0*(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, 0,1, 0.2, 5)
```

▷ Click here to RUN the code.
The output is

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 1 & 1.0202 & 1.08329 & 1.19722 & 1.37713 & 1.64872 \end{bmatrix}$$

4.4.2 * the build-in `rk(.)` method in MAXIMA^{online}

`RK4` is build-in in MAXIMA^{online} as function

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

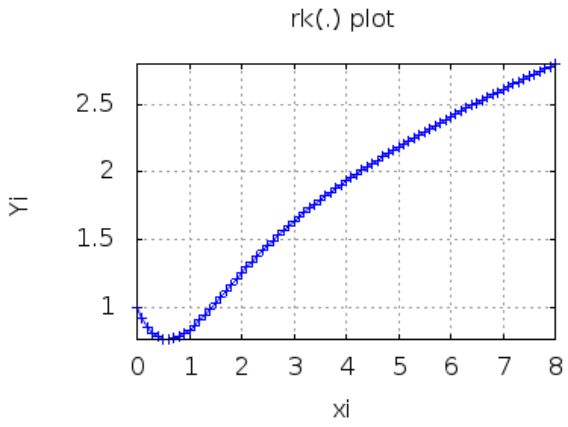
cf. https://maxima.sourceforge.io/docs/manual/maxima_112.html#index-rk

▷ "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])$
draw2d( xaxis = true, grid=true,
        points_joined=true, xlabel="xi", ylabel="Yi",
        points(results),
        title="rk(.) plot")$
```

Click to invoke MAXIMA^{online}
This is the plot

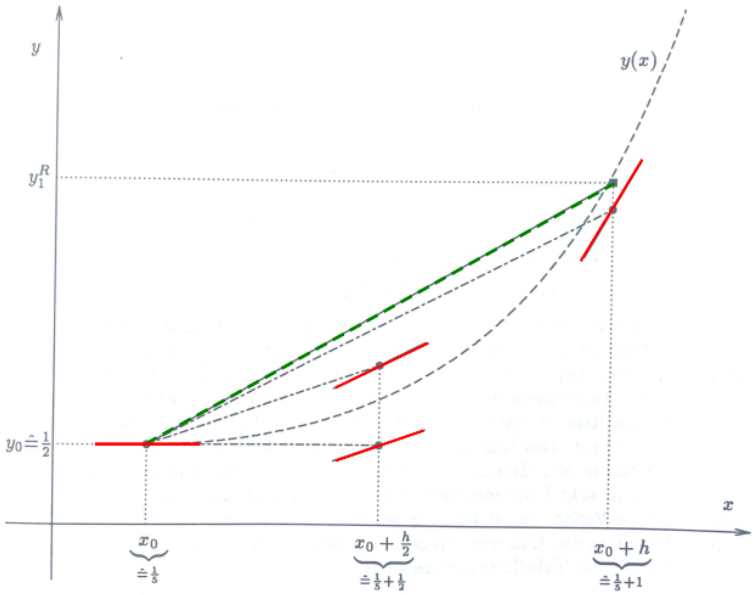


Exercises.

Exercise 112. Solve the test ODE: $y' = xy$ with $y(0) = 1$ using the MAXIMA^{online} build-in method `rk()`.

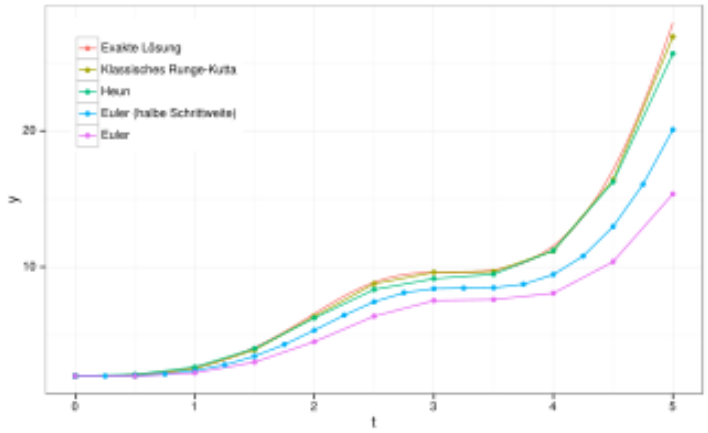
Exercise 113.
a. Use EIGENMATH's `RK4` function with step size $h = 0.1$ to find an approximate solution of ODE $\frac{dy}{dx} = x - y^2$, $y(0) = -0.5$, see [12, p.145]. [Intermediate result: $y_3 = -0.5520$]
b. Solve **a.** with the MAXIMA^{online} built-in function `rk`, see the remark above.
c. Calculate the global error for the methods in a. and b. Compare.

Exercise 114. 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. [36, 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 MAXIMA^{online}'s built-in function `rk`.

Exercise 115. 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), `modiEULER` (cyan) and `EULER` (magenta). Use these methods to solve the given ODE.

Exercise 116. 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 117. (RK3) This is RK3 method formulated in MAXIMA^{online} 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(x,y, 0,1, 0.2, 5);
```

Translate this MAXIMA^{online} code to EIGENMATH.
Use this playground: ▷ Click here to work on the code.
Then do
a. exercise 113 using method RK3. Compare. Truncation error(s)?
b. exercise 115 using RK3 and reproduce the plot incl. method RK3.

Exercise 118. Solve the OVP $\frac{dy}{dx} = \frac{2y-x}{x}$, cf. [43, 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 119. 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 EIGENMATH.
d. Calculate the absolute errors for the methods in c. Compare.

Exercise 120. Read ▷ CHEEVER: Runge-Kutta Method and do the example with our RK4.

Exercise 121. Read ▷ WIKI: Runge-Kutta>Use and do the example with our RK4.

Exercise 122. Read ▷HELLEVIK: Falling sphere using RK4. and do the example with our RK4.

Exercise 123. Read ▷ ESE: Runge-Kutta and do the example ODE $y' = x^2$ and the exercise with our RK4.

Exercise 124. 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 125. Re-do the four examples in ▷HELLEVIK: EXERCISES.

Exercise 126. * Study ▷HELLEVIK: Basic notions on numerical methods for IVPs. and adapt these notations for 'our' error handling. Look also at ▷ ..: On errors.

Exercise 127. ** Study ▷Hellevik: Absolute stability of numerical methods Euler, Heun, RK.

Exercise 128. 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*	Eigenmath
	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) or RK2e(..)`
	Heun	`HEUN(x,y, xo,yo, h,n) or RK2h(..)`
	Midpoint	`MIDPOINT(x,y, xo,yo, h,n)) or 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

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

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.

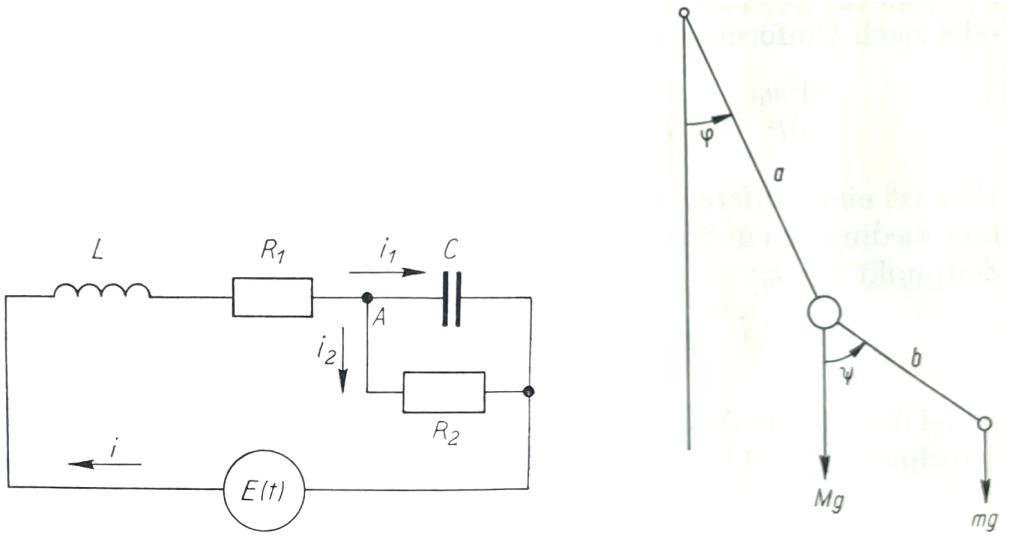


Figure 18: BRÄUNING [12, p.11] motivates the dealing with systems of ordinary differential equations through their use in the natural sciences like:
Left:(*circuit*) "An electrical circuit contains a generator with the time-dependent voltage E(t), a spool with the inductance L an ohmic resistor R1 as 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 EIGENMATH 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 13. (*solving an IVP system of 2 equations in* EIGENMATH)
Solve numerically the system

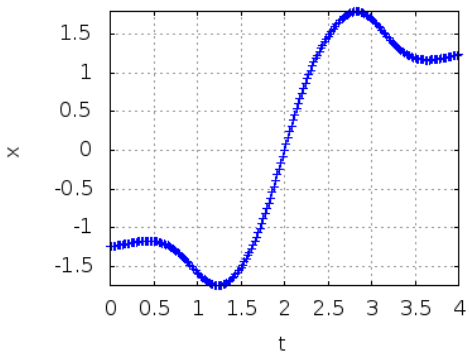
$$\frac{dx}{dt} = 4 - x^2 - 4y^2 \;\&\; \frac{dy}{dt} = y^2 - x^2 + 1$$

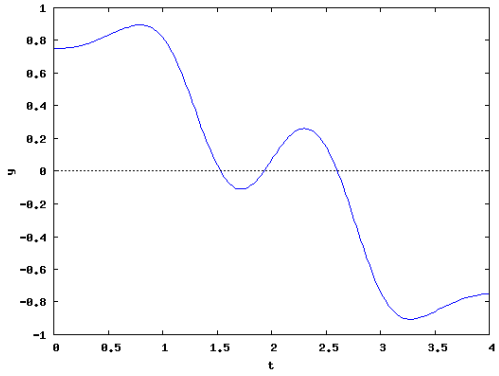
for t between 0 and 4, and with values of -1.25 and 0.75 for x and y at $t = 0$.
Cf. https://maxima.sourceforge.io/docs/manual/maxima_112.html#index-rk

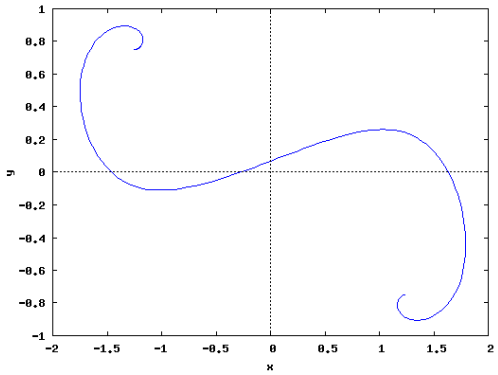
Solution: We have the system $\left\{\begin{matrix} x'=f(t,x,y)=4-x^2-4y^2,\, x(0)=-1.25 \\ y'=g(t,x,y)=y^2-x^2+1 \;\;,\, y(0)=0.75 \end{matrix}\right\}$.
To get the solutions $x(t)$ and $y(t)$, we call the build-in MAXIMA^{online} function `rk(.)`:

```
/* MAXIMA-online : 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   */
draw2d( grid=true,
        xlabel= "t", ylabel= "x",
        points_joined = true,
        points(sol)) $
```

Click to invoke MAXIMA^{online}
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`[13], which deals with 3 recurrent entries, e.g. time t and position in plane (x, y).
This is a straight forward implementation of **iterate3** in analogy to `iterate2`:

```
-- EIGENMATH : iterate3 for triples ---

iterate3(u,X,Xo,n) = do( M = zero(n+1,3),
                         M[1] = Xo,
                         for(i,2,n+1, M[i] = eval(u, X[1],M[i-1,1],
                                                     X[2],M[i-1,2],
                                                     X[3],M[i-1,3])),
                          transpose(M) )
```

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

Example 14. (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 ..
f(t,x,y) = y                 -- 1st part of system (f,g)
g(t,x,y) = -t*y-x-6*t^2      -- 2nd part of system (f,g)
do(a=0, b=4, c=0)            -- initial conditions "ICs"
h = 1/8                      -- chose step size
n = 16                       -- chose number of iterations
float( iterate3((t+h, x+h*f(t,x,y), y+h*g(t,x,y)),  -- updating the system
                (t   , x           , y ), (a,b,c), n))
```

▷ Click here to RUN the code.
The output is

0	0.125	0.25	0.375	0.5	0.625	0.75	0.8
4	4	3.9375	3.81201	3.62306	3.37022	3.05314	2.67
0	−0.5	−1.00391	−1.5116	−2.02271	−2.53667	−3.05274	−3.5 ...

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 15. 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 $(y_x, z_x) = (\cosh(x) + 2\sinh(x), \sinh(x))$.

[13] I owe this elegant version to a private discussion with George WEIGHT. My original function was much more cumbersome.

```
h = 0.1
f(x,y,z) = 2*y-3*z          -- variables are: (x,y,z)
g(x,y,z) = y-2*z            -- to be seen here   |
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)
y5
z5= sinh(1.0)
z5
```

▷ Click here to RUN the code.
The output is

$$\begin{bmatrix} 0 & 0.1 & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 \\ 1 & 1.2 & 1.41 & 1.632 & 1.8681 & 2.12052 & 2.39162 & 2.68393 & 3.00015 \\ 0 & 0.1 & 0.2 & 0.301 & 0.404 & 0.51001 & 0.62006 & 0.73521 & 0.856561 \end{bmatrix}$$

$y_5 = 3.89348$

$z_5 = 1.1752$

Exercise 129. Use this simple EULER method to solve numerically example 13.

♡ *We now transfer our knowledge about solution methods for a single ordinary differential equation in* EIGENMATH *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 IVPs

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:

```
--- EIGENMATH : EULER method for systems of IVP ---
--              f and g are to be defined global
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 here to RUN the code.
The output is

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 0 & 0.2 & 0.4 & 0.616 & 0.864 & 1.16064 \\ 1 & 1 & 1.08 & 1.24 & 1.4832 & 1.816 \end{bmatrix}$$

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:

```
--- EIGENMATH : PICARD iteration for systems ---
f(x,y,z) = z
g(x,y,z) = y + x      -- = z'
do(a=0,b=1)           -- y(a) = b
do(c=0,d=1)           -- z(c) = d = y'(c)

y(n,x) = test(n = 0, b, defint(f(x, y(n-1,x), z(n-1,x)),x,a,x))
z(n,x) = test(n = 0, d, defint(g(x, y(n,x)  , z(n-1,x)),x,c,x))

y(5,x)
eval(y(5,x),x,1)
float

z(5,x)
eval(z(5,x),x,1)
float
```

▷ Click here to RUN the code.

a. The code should produce this result:

(%o75) $z_n(x):=d+\int_c^x g(t,y_n(t),z_{n-1}(t))dt$

(%o76) $\frac{x^9}{181440}+\frac{x^7}{2520}+\frac{x^5}{60}+\frac{x^3}{3}+x$

(%o77) 1.3504

(%o78) 2.0862

Fix it.

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].

```
-- EIGENMATH : RUNGE-KUTTA method RK4sys for systems of IVP ---

RK4sys(x,y,z, xo,yo,zo, h,n)= do(
              k1 = h*f(x,y,z),
              l1 = h*g(x,y,z),
              k2 = h*f(x+h/2, y+k1/2, z+l1/2),
              l2 = h*g(x+h/2, y+k1/2, z+l1/2),
              k3 = h*f(x+h/2, y+k2/2, z+l2/2),
              l3 = h*g(x+h/2, y+k2/2, z+l2/2),
              k4 = h*f(x+h,   y+k3,   z+l3  ),
              l4 = h*g(x+h,   y+k3,   z+l3  ),
              iterate3((x+h,
                        y+1/6.0*(1*k1+2*k2+2*k3+1*k4),
                        z+1/6.0*(1*l1+2*l2+2*l3+1*l4)),
                       (x,y,z), (xo,yo,zo), n)  )

-- Test ODE: system f(x,y,z)=z, g(x,y,z)=y+x with y(0)=0, z(0)=1
f(x,y,z) = z
g(x,y,z) = y+x
RK4sys(x,y,z, 0,0,1, 0.2, 5)
```

▷ Click here to RUN the code.

The output is

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 0 & 0.202667 & 0.421494 & 0.67329 & 0.976186 & 1.35037 \\ 1 & 1.04013 & 1.16214 & 1.37092 & 1.67486 & 2.08614 \end{bmatrix}$$

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 16. (*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 MAXIMA^{online}'s `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(t,x,y) = y
g(t,x,y) = x+t
EULERsys(t,x,y, 0,0,1, 0.1, 10)
```

▷ Click here to RUN the code.

$$\begin{bmatrix} 0 & 0.2 & 0.4 & 0.6 & 0.8 & 1 \\ 0 & 0.2 & 0.4 & 0.616 & 0.864 & 1.16064 \\ 1 & 1 & 1.08 & 1.24 & 1.4832 & 1.816 \end{bmatrix}$$

Approximate value for $y(1) \approx 1.94242$.

c. `RK4sys` method

```
f(t,x,y) = y
g(t,x,y) = x+t
RK4sys(t,x,y, 0,0,1, 0.1, 10)
```

▷ Click here to RUN the code.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0	0.100333	0.202672	0.30904	0.421504	0.54219	0.673306	0.817166	0.97621	1.15303	1.3504
1	1.01001	1.04013	1.09068	1.16214	1.25525	1.37093	1.51034	1.67487	1.86617	2.08616

Approximate value for $y(1) \approx 2.08616$.

d. PICARD–LINDELÖF method using MAXIMA^{online}:

```
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[5](x), expand;   y[5](1), numer;
z[5](x), expand;   z[5](1), numer;
```

▷ Click to RUN the code.

```
(%i12) y[5](x), expand;
```

(%o12) $\frac{x^9}{181440} + \frac{x^7}{2520} + \frac{x^5}{60} + \frac{x^3}{3} + x$

```
(%i13) y[5](1), numer;
```

(%o13) 1.3504

```
(%i14) z[5](x), expand;
```

(%o14) $\frac{x^{10}}{1814400} + \frac{x^8}{20160} + \frac{x^6}{360} + \frac{x^4}{12} + x^2 + 1$

```
(%i15) z[5](1), numer;
```

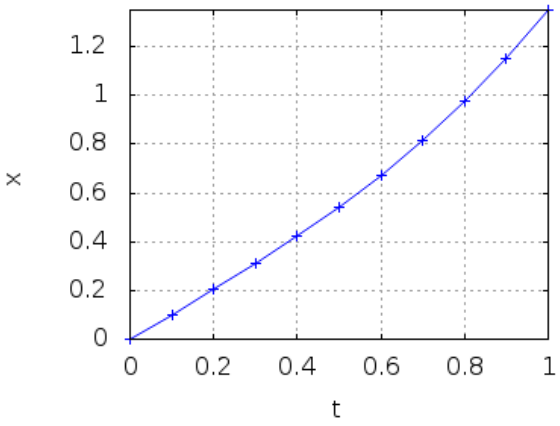
(%o15) 2.0862

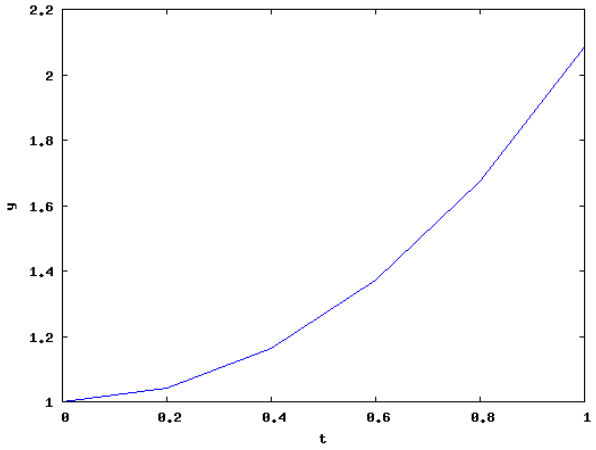
Yamwi

e. `rk(.)` build-in MAXIMA^{online} function for systems of ODEs

```
fpprintprec:5$
sol: rk([y, x+t], [x, y], [0, 1],  [t, 0, 1, 0.1])$

draw2d( grid=true,
        xlabel= "t", ylabel= "x",
        points_joined = true,
        points(sol)) $
```

▷ Click here 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 use the given 2nd order ODE* $y'' = f(x, y, y')$ *directly 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 a recurrence formula and an example. We translate the algorithm to EIGENMATH, `yp` $('_{y\ prime}') \equiv y'$.

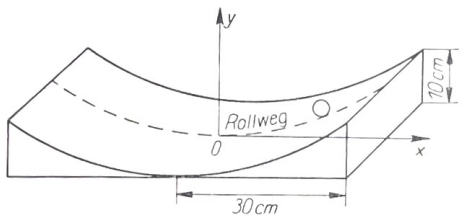
```
RK4ode2( x,y,yp, xo,yo,ypo, h, n) = do(
         k0 = 1/2*h^2* f(x,y,yp),
         k1 = 1/2*h^2* f(x+h/2, y+1/2*h*yp+1/4*k0, yp+k0/h),
        k1p = 1/2*h^2* f(x+h/2, y+1/2*h*yp+1/4*k0, yp + k1/h),
         k2 = 1/2*h^2* f(x+h,   y +h*yp + k1p,  yp + 2*k1p/h),
          k = 1/3*(k0+  k1+  k1p   ),
          l = 1/6.0*(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 here to RUN the code.
The output is

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1.11069	1.24576	1.41024	1.61002	1.85195	2.14407	2.4958	2.91814	3.42403	4.02863
1	1.22104	1.48871	1.81073	2.19619	2.65584	3.20232	3.85049	4.6179	5.52514	6.59648

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 130. (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[14] – 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 $x(20)$?

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

5.2.2 ROMER's method for 2^{nd} order IVP

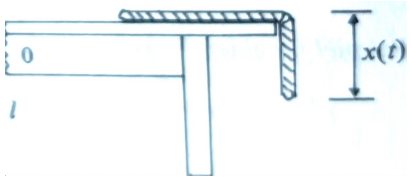
ROMER [43, p.28 ff] discuss the following variant of `RK4sys`, which we translate to EIGENMATH.

```
--- EIGENMATH --- ROMER's method for 2nd order ODE ---
RK4romer( x,y,yp, xo,yo,ypo, h, n) = do(
                  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+1/6.0*(k1+k2+k3)),
                  yp+1/6.0*(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
f(x,y,yp) = x+y+yp
RK4romer(x,y,yp, 0,1,1, 0.1, 10)
```

▷ Click here to RUN the code.

0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	1.11069	1.24576	1.41024	1.61002	1.85195	2.14407	2.4958	2.91814	3.42403	4.02863
1	1.22104	1.48871	1.81073	2.19619	2.65584	3.20232	3.85049	4.6179	5.52514	6.59648

Example 17. (a classic problem, ROMER [43, p.30 ff])
A perfectly flexible rope of length ℓ 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 $M\ddot{x} = \frac{M}{\ell} \cdot x \cdot g \rightsquigarrow \ddot{x} = \frac{g}{\ell} \cdot x$

```
g = 9.81
L = 1
f(x,y,yp) = g/L*y     -- ..= y''
```

```
                              -- x0 y0   yp0      h    n --
transpose( RK4romer(x,y,yp,  0, 0.1, 0,     0.05, 20))
```

▷ Click here to RUN the code.

Output: (only last 5 entries)

0.8	0.616669	1.9059
0.85	0.719931	2.23304
0.9	0.840885	2.61505
0.95	0.982505	3.06132
1	1.14827	3.58283

The returned values coincide with the tabulated values of the Basic program by ROMER. The ropes position at $t = 0.9$ [sec] is ca. $x = 0.26$ [m].

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

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

Exercises.

Exercise 131. Reduction of Higher order Equations ▷ HELLEVIK: ivp.

Exercise 132. 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 133. 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 134. 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. [30, p.284]. [Result: $y(1) = 0.598451$ to 6D.
Determine the exact analytical solution $y_e(x)$ and compare with the approximative solution.

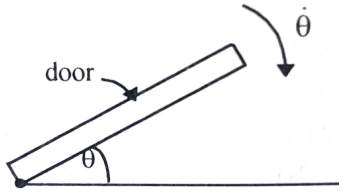
Exercise 135. Solve the system of 1^{st} order IVPs $\frac{dx}{dt} - x - 5y = 18t;\ \frac{dy}{dt} + 2x + y = 9$, given $x(0) = 4, y(0) = -1$, cf. [30, p.287].

Exercise 136. 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. [30, p.287].

Exercise 137. 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. [30, 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 138. (swinging door, cf. [30, 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 Θ with time.

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

Exercise 139. 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 φ 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

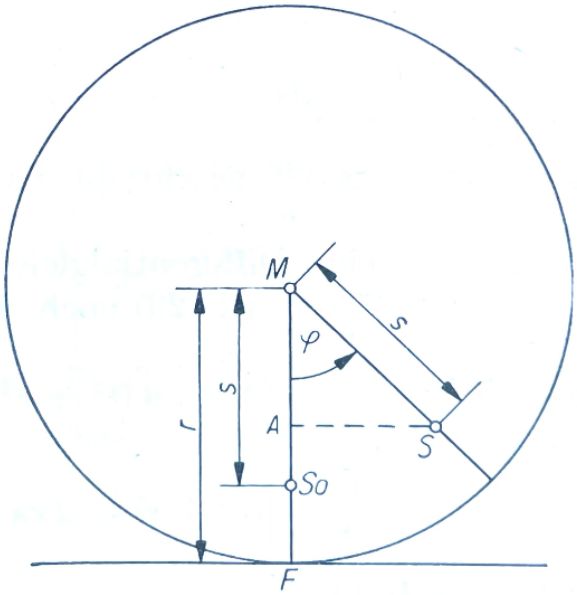
$$\ddot{\varphi} = -\tfrac{g}{\ell}(\varphi - \tfrac{1}{6}\varphi^3) \qquad (\ddagger)$$

a. Solve eq. ‡ for the IC $\varphi(0) = 0, \dot{\varphi}(0) = 2$ with the methods
– successive approximation
– `RK4sys`, `RK4ode2`, `RK4romer` and `rk`^{Eigenmath}.
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 140. (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 φ 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 Θ_s is the moment of inertia.

a. Using special values for m, Θ_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 $\dot{\varphi}(0.5) \approx -1.2564$.

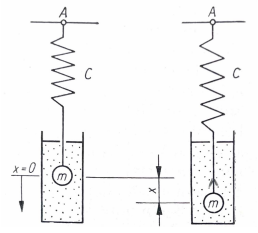
b. Verify, that the corresponding system of simultaneous IVP's for ‡ is

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

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 141. (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.

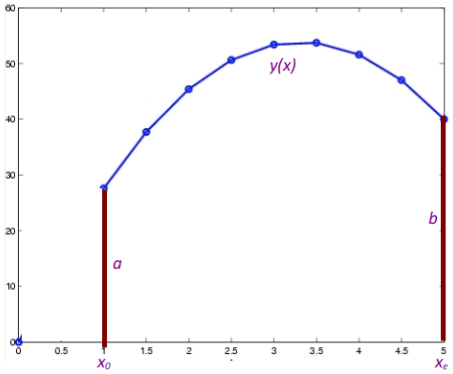
2. TIMBERLAKE [49, p.97 ff] discuss the damped harmonic oscillator and the pendulum on p.115 ff. He also make use of analytical methods like EIGENMATH's `desolve`.

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:

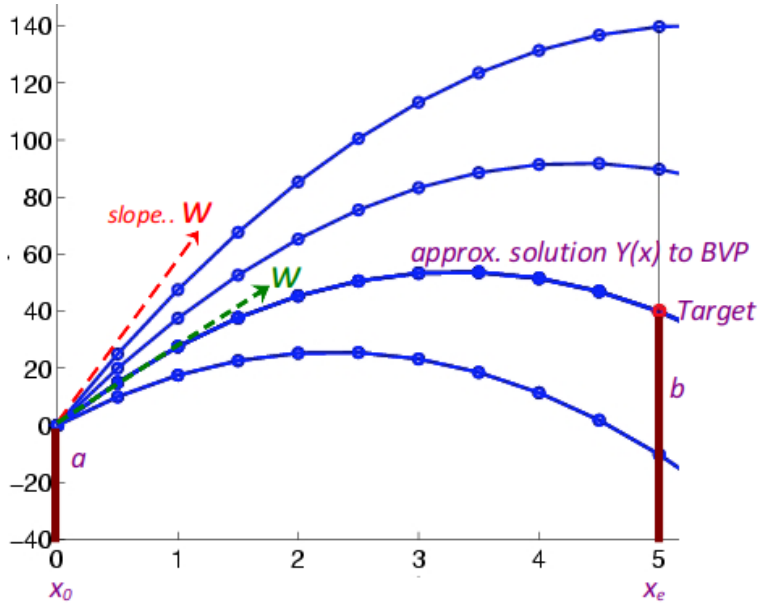


Figure 19: *Idea*: a projectile is shooted from start position $(0,0)$ at the target • $(5,40)$ with different slopes w_k. The winning slope w and its corresponding trajectory is sandwiched between three trial shoots • – •. The BVP $y'' = f(x,y,y')$ with BC $y(x_o) = a,\ y(x_e) = b$ is approximately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

We now follow the shooting method in an example, see [44, 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 EIGENMATH.
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 $w = -1$:

```
f(x,y,z) = z
g(x,y,z) = y+sin(x+z)
RK4sys(x,y,z, 0.0,1.2, 1.0,  3/16, 16)
```

▷ Click here to RUN the code.

Output:

$$\begin{bmatrix} \frac{9}{4} & 11.4337 & 11.3121 \\ \frac{39}{16} & 13.7825 & 13.7731 \\ \frac{21}{8} & 16.609 & 16.4873 \\ \frac{45}{16} & 20.0233 & 20.0056 \\ 3 & 24.1414 & 24.1019 \end{bmatrix}$$

We determine the length of the `RK4sys`–list and pick its last element:

```
dim( RK4sys(x,y,z, 0.0,1.2, 1.0,  3/16, 16), 2)
RK4sys(x,y,z, 0.0,1.2, 1.0,  3/16, 16)[2,17]
```

▷ Click here to RUN the code.

| 17
| 24.1414

ad **2**: define helper function $F(w) :=$ (approx. solution to IVP at $x = x_e$) - b
and repeat the two shoots from 1.:

```
f(x,y,z)=z
g(x,y,z)=y+sin(x+z)
F(w)= RK4sys(x,y,z, 0.0,1.2, w,  3/16, 16)[2,17] - 2.4

F(1.0)
F(-1.0)
```

▷ Click here to RUN the code.

The output is

| 24.1414
| -0.9641

We observe, that the last value is approaching 0.

ad **3**: solve $F(w) = 0$, i.e. search for w^* with $F(w^*) = 0$.

• We calculate the first 4 decimals of w by watching how $F(w)$ approaches 0:

```
for(n,1,9, print(n, F(    0-n*0.1)) )
for(n,1,9, print(n, F(-0.9-n*0.01)) )
for(n,1,9, print(n, F(-0.93-n*0.001)) )
for(n,1,9, print(n, F(-0.936-n*0.0001)) )
```

▷ Click here to RUN the code.

$n = 7$

0.00294856

$n = 8$

0.00147308

$n = 9$

-2.52329×10^{-6}

The output shows that the 4st decimal has to be choosen $n = 9$, i.e. $w = -0.xxx9$. Result: the optimal slope for the best shot is $w^* = -0.9369$.

ad **4**: put w^* back in IVP to solve the BVP.

```
RK4sys(x,y,z, 0.0,1.2, -0.9369,  3/16, 16)[2,17]
```

▷ Click here to RUN the code.

| 2.4

Alternatively we may also use our specialized function `RK4romer` for this 2nd order IVP:

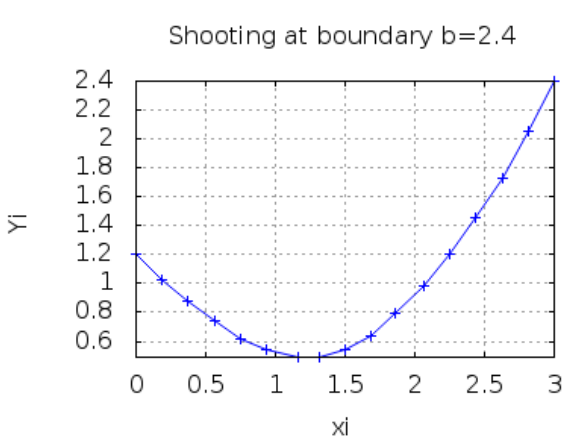
```
f(x,y,yp) = y+sin(x+yp)
RK4romer(x,y,yp, 0,1.2,-0.9369, 3/16, 16)
```

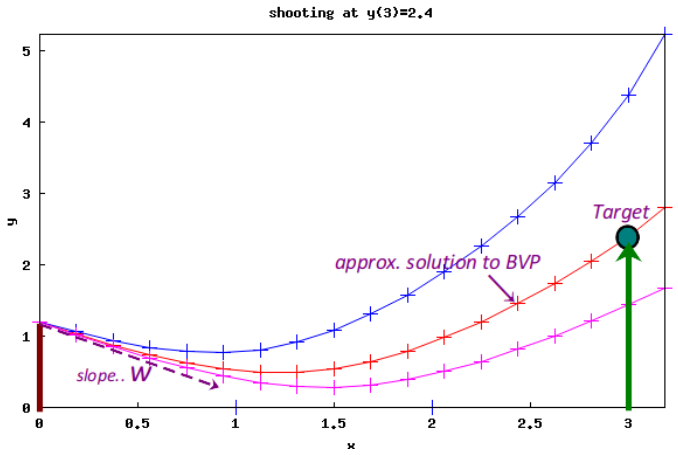
▷ Click here to RUN the code.

Output:

0	1.2	−0.9369
0.1875	1.03124	−0.862881
0.375	0.876935	−0.780611
0.5625	0.739784	−0.677872
0.75	0.624781	−0.542339
0.9375	0.539127	−0.363402
1.125	0.491503	−0.136902
1.3125	0.490249	0.128405
1.5	0.540626	0.408992
1.6875	0.642747	0.675873
1.875	0.791956	0.909258
2.0625	0.981356	1.10511
2.25	1.20453	1.27159
2.4375	1.45727	1.42314
2.625	1.73837	1.57742
2.8125	2.05028	1.75569
3	2.4	1.98642

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:






```
X: [0, 0.18, 0.37, 0.56, 0.75, 0.94, 1.16, 1.31, 1.5,
     1.69, 1.86, 2.06, 2.25, 2.44, 2.63, 2.81,3];
Y: [1.2, 1.03, 0.88, 0.74, 0.62, 0.54, 0.49, 0.49, 0.54,
     0.64, 0.79, 0.98, 1.20,1.46, 1.73, 2.05, 2.4];

draw2d( xaxis = true, grid=true,
points_joined=true, xlabel="xi", ylabel="Yi", points(X,Y),
title="Shooting at boundary b=2.4")$
```

▷ Click here to RUN the code.

Exercise 142. **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 using* MAXIMA^{online}.

Solve the BVP $y'' = \frac{y}{1+x^2} + \frac{y'}{10}$ with BC: $y(0) = 1,\ y(2) = 3$. cf. [44, 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-online  :  SHOOTING METHOD for BVP --- */
/* Write function RK4sys. Inspiration by ex. 117 and RK4sys/Eigenmath
   and insert it here: */
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 here to RUN the code.

$$f(x,y,z) := z$$

$$g(x,y,z) := \frac{y}{1+x^2} + \frac{z}{10}$$

$$F(w) := \left(\text{last}\left(\text{RK4sys}\left(x,y,z,0,1,w,\frac{1}{8},16 \right) \right) \right)_2 - 3$$

$$0.058068$$

We get $w = 0.58068$. Check:

```
f(x,y,z) := z;
g(x,y,z) := y/(1+x^2)+z/10;
RK4sys(x,y,z, 0,1,  0.58068,   1/8, 16);
```

▷ Click here to RUN the code.

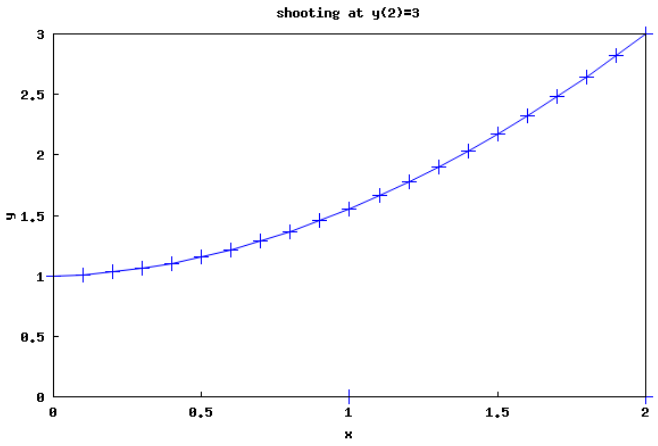
The output is:

```
[[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 $y(x)$:

```
X: [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
       1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0];
Y: [1, 1.01, 1.03, 1.06, 1.10, 1.16, 1.22, 1.29, 1.37, 1.46,
       1.56, 1.66, 1.78, 1.90, 2.03, 2.18, 2.32, 2.48, 2.65, 2.82, 3];
draw2d( xaxis = true, grid=true,
points([[0,0]]),
points_joined=true, xlabel="xi", ylabel="Yi", points(X,Y),
title="Shooting at boundary b=3 for 2nd order ODE ypp=y/(1+x**2)+yp/10"$
```

▷ Click here to RUN the code.



Exercise 143. **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 144. HECKBERT ▷ bvp gives the following procedural Python code for the secant method to determine roots of a function. Write a EIGENMATH resp. MAXIMA^{online} function `secant` and use it in examples 6.1.1 and 6.1.2 instead of `find_root`.

```
%% 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* `RK4sys` *with* $n = 8$ *steps to approximate the solution function* $y(x)$ *and answer the required questions. Check for plausibility with* `RK4romer` *and do a plot of the scene.*

Exercise 145. Solve the BVP: $y'' = 2y^3$ with BC: $y(1) = 1/4$, $y(3) = 1/6$, cf. [44, 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 146. Solve $y'' = y^2 + y' + \frac{2}{x^3} - 3 - x^2$ with $y(1) = 2$, $y(2) = \frac{5}{2}$, cf. [44, 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 147. (BULIRSCH–STOER, [47, 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 148. (BULIRSCH–STOER, [47, 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 149. (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

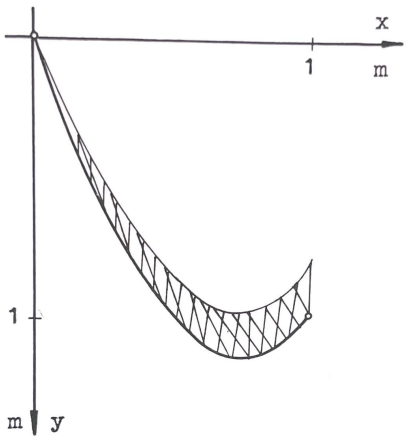
Exercise 150. (BORSE, [8, p.459]) Solve the

$$\text{BVP}: \begin{cases} (y-1)^2 y'' + 2(y-1)(y')^2 = 0 \\ y(0) = 2 \\ y(1) = 3 \end{cases}$$

by the shooting method and compare with the exact solution $y_e(x) = 1 + (7x+1)^{1/3}$.

Exercise 151. (BORSE, [8, 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 152. (VENZ, [50, 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:

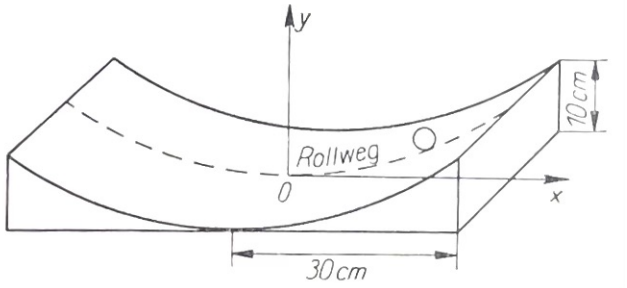


We approximate the load by the function $q = q_0 \cdot \sin(\frac{\pi}{2} \cdot \frac{x}{x_b})$.
Then the differential equation is

$$y'' = \frac{q_0}{S} \cdot \sin(\frac{\pi}{2} \cdot \frac{x}{x_b})$$

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 153. (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[15] – the differential equation of the motion

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

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 154. Do the examples from ▷ GROTHMANN: Shooting ..
`>Documentation>Examples>All Examples>Shooting method for boundary Problems`
and at `...>Singular Boundary Value Problem`.

Exercise 155. 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 156. Do the ▷ HECKBERT: rocket problem. using our Shooting method.

Exercise 157. Do the BVP's in ▷ NIEMEYER: Shooting method. using our Shooting method.

Exercise 158. Do the BVP's ▷ DORINE: Shooting method. on the deflection of a supported beam with a constant distributed load.

Exercise 159. Do the 3 BVP's ▷ AREFIN et al.: Shooting methods. in: Analysis of Reliable Solutions to the Boundary Value Problems by Using Shooting Method

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

Exercise 160. Do the BVP examples on Shooting by ▷ Uni Muenster, DE: Shooting methods.

Exercise 161. Do the BVP's ▷ Berkeley Python Numerical Methods: Shooting methods.

Exercise 162. Have a look at the BVP examples in ▷ Verschelde 2022: Shooting.

Exercise 163. 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 164. ▷ Kumar: Shooting methods.

Exercise 165. ▷ Vesely: Shooting methods.

Exercise 166. ▷ Pardyjak: Shooting methods.

Exercise 167. ▷ Numerical Methods for Engineers: Shooting methods.

6.2 Finite Difference Method

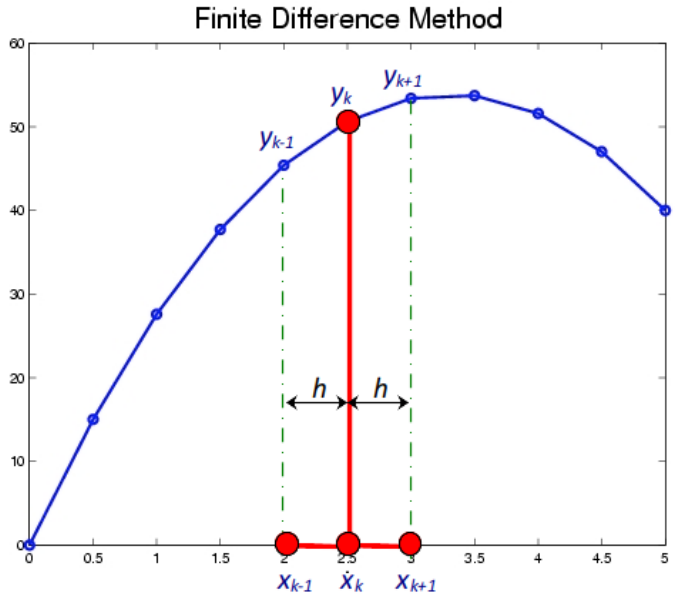


Figure 20: *Idea*: The approximative values y_k of the solution y to the BVP $y'' = f(x, y, y')$ of 2nd order are calculated by replacing the derivatives y'' and y' by so-called *central difference approximations*. The 4-point '*stencil*' x_{k-1} $\overset{y_k}{x_k}$ x_{k+1} traverse from the first BC1: $x_0 = 0, y_0 = 0$ 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 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 [44, 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 EIGENMATH.
1: *replace* the first and second derivatives in the BVP equation by their approximations
2: *write down* the corresponding 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[16] for y', y'' in EIGENMATH:

```
y        = (y0, y1, y2, y3, y4)
yp(k,h)  = (y[k+2]-y[k])/(2*h)
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]   -- RHS = 3*x(k)-1   /*(2)*/
--          y''      + 0.2 y'      + 4 y              = 3x-1
```

▷ Click here to RUN the code. Observe: NO output.

Herein `FDM(k,h)` represents the RHS of the corresponding 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 and the separated RHS in EIGENMATH :

```
do(n=4, xo=0, xe=1, h=(xe-xo)/n)
for(i,1,3, print(FDM(i,h), 3*x(i)-1))    -- show LHS and RHS separated
```

▷ Click here to RUN the code.

$$15.6\,y_0 - 28\,y_1 + 16.4\,y_2$$

$$-\frac{1}{4}$$

$$15.6\,y_1 - 28\,y_2 + 16.4\,y_3$$

$$\frac{1}{2}$$

$$15.6\,y_2 - 28\,y_3 + 16.4\,y_4$$

$$\frac{5}{4}$$

[16] 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 list of equations to arrive at the augmented matrix of the system $augM$:

```
-- y[1]=0.1     -- 1st BV as 1st equation
-- y[5]=0.7     -- 2nd BV as 2nd equation

augM =((   1,     0,     0,     0,     0,    0.1 ),
       (15.6, -28.0,  16.4,     0,     0,  -0.25 ),
       (   0,  15.6, -28.0,  16.4,     0,    0.5 ),
       (   0,     0,  15.6, -28.0,  16.4,   1.25 ),
       (   0,     0,     0,     0,   1.0,    0.7 ))

augM
```

▷ Click here to RUN the code.

EIGENMATH output:

$$a_{ugM} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.1 \\ 15.6 & -28 & 16.4 & 0 & 0 & -0.25 \\ 0 & 15.6 & -28 & 16.4 & 0 & 0.5 \\ 0 & 0 & 15.6 & -28 & 16.4 & 1.25 \\ 0 & 0 & 0 & 0 & 1 & 0.7 \end{bmatrix}$$

Comment: Looking at the augmented coefficient matrix `augM` of our system of the equations `BVP`$_k = 3x_k - 1$ in (1), we observe the *tridiagonal* shape of the system. `augM` is the coefficient matrix of the system with a column adjoined for the constant terms on the RHS in each equation.

ad **4**: *solve* this *tridiagonal* system.
We use our homemade routine named `RREF`[17] to solve the system.

```
run("gjBox.txt")
RREF(augM)
```

▷ Click here to RUN the code.

EIGENMATH output:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 & 0 & 0.456111 \\ 0 & -1.82645\times10^{-16} & 1 & 0 & 0 & 0.668361 \\ 0 & -1.0176\times10^{-16} & 0 & 1 & 0 & 0.73773 \\ 0 & 0 & 0 & 0 & 1 & 0.7 \end{bmatrix}$$

[17] `RREF` i.e. Row Reduced Echelon Form

Remark. Alternatively we can solve the system $A \bullet X = R$ using the inverse of the coefficient matrix A of the system and the isolated RHS R, i.e. $X = A^{-1} \bullet R$:

```
A =   ((   1,     0,     0,     0,     0 ),
       (15.6, -28.0,  16.4,     0,     0 ),
       (   0,  15.6, -28.0,  16.4,     0 ),
       (   0,     0,  15.6, -28.0,  16.4 ),
       (   0,     0,     0,     0,   1.0 ))

R=(0.1,-0.25,0.5,0.75,0.7)

dot(inv(A),R)
```

▷ Click here to RUN the code.

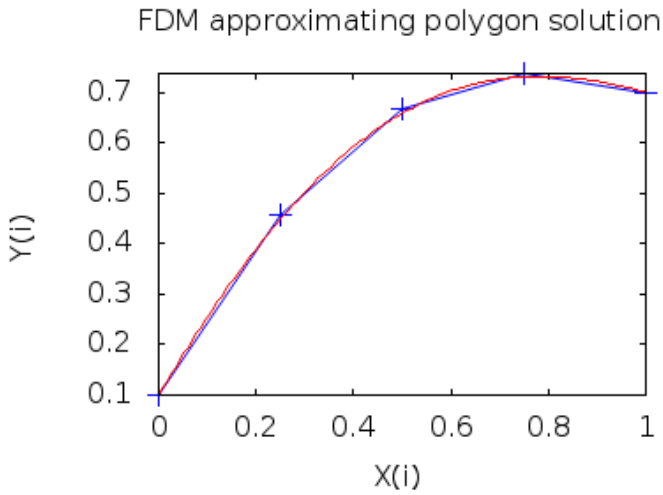
EIGENMATH output:

$$\begin{bmatrix} 0.1 \\ 0.473748 \\ 0.698472 \\ 0.772363 \\ 0.7 \end{bmatrix}$$

4: *plot* the result.
The complicated exact solution $y_e(x)$ of the BVP is given in [44, 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;
n:4; xo:0; xe:1; h:(xe-xo)/n;
x(k)     := xo+k*h;
X: makelist(x(k), k,0,4);
Y: [0.1, 0.456, 0.668, 0.737, 0.7];
draw2d(xaxis = true,
    point_size=2,
    points_joined=true,
    xlabel="X(i)",ylabel="Y(i)",
    points(X,Y),
    color=red,
    explicit(ye(x),x,0,1),
    title="FDM approximating polygon solution")$
```

▷ Click here to RUN the code.



Exercise 168. Repeat example 6.2.1 for $n = 8$.

Exercise 169. Redo the FDM *fireworks example* by HECKBERT at ▷ BVP: FDM. using our 4-step-procedure to solve an BVP.

6.2.2 a generalized procedure FDM for BVP $y'' = f(x, y, y'), BC : \begin{smallmatrix} y(x_o)=a \\ y(x_e)=b \end{smallmatrix}$.

We give this procedure for MAXIMA^{online}. Extend the variable list y in (1), if necessary.

```
/* MAXIMA-ONLINE --- FINITE DIFFERENCE METHOD (FDM) for BVP  --- */
FDM(xo,xe,a,b,n):= block(
    h     : (xe-xo)/n,
    x(k)  := xo + k*h,
    y     :  [y0,y1,y2,y3,y4,y5,y6,y7,y8,y9],                  /* (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 here to RUN the code.

$$BVP(k) := ypp(k) + 0.2\, yp(k) + 4\, y_{k+1} = 3\, x(k) - 1$$

$$[0.1, 0.29143, 0.45051, 0.57398, 0.66091, 0.71261, 0.73255, 0.72607, 0.7]$$

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^{online}'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`.

6.2.3 a general procedure FDMeq for BVP $y'' = f(x, y, y'), BC : \begin{smallmatrix} y(x_o)=a \\ y(x_e)=b \end{smallmatrix}$.

Robert DODIER from the MAXIMA team suggests[18] the following variant for MAXIMA^{online}, which makes use of subscripted variables and avoids the inflexible hard-coded variable names as in 6.2.2(1). He arranged the main function `FDMeqs` so that it only constructs the list of equations, but does not solve them. So one may review the equations before solving. He also use *centered* differences to keep the indexing more straight. By the way, the functions `x`, `yp`, and `ypp` are global, because they are defined by " `:=` ".

```
/* MAXIMA-ONLINE --- FINITE DIFFERENCE METHOD (FDMeq) for BVP  --- */
FDMeqs(x_0, x_n, y_0, y_n, n) :=
   block([h      : (x_n - x_0)/n],
         x(k)    := x_0 + k*h,
         yp(k)   := (y[k + 1] - y[k - 1]) / (2*h),
         ypp(k)  := (y[k + 1] - 2*y[k] + y[k - 1])/h^2,
         append ([y[0]=y_0], makelist(BVP(i), i,1,n-1), [y[n]=y_n]));

BVP(k) := ypp(k) + (2/10)*yp(k) + 4*y[k] = 3*x(k) - 1;

/* Test BVP:   y''+0.2y'+4y=3x+1   BC: y(0)=0.1=a, y(1)= 0.7=b */
fpprintprec:5$  ratprint:false$
n   : 4;
eqs : FDMeqs(0, 1, 1/10, 7/10, n);       /* construct list of eqs */
yy  : makelist(y[i],i,0,n);              /* construct list of variables */
linsolve(eqs, yy). numer;                /* solve eq(uation)s */
```

The arguments of `FDMeqs` are the same as in `FDM`.

▷ Click here to RUN the code.

Here is the MAXIMA^{online} output:

```
(%i6) eqs : FDMeqs(0, 1, 1/10, 7/10, n);

(%o6) [y0 = 1/10, 16 (y2 − 2 y1 + y0) + 2 (y2−y0)/5 + 4 y1 = −1/4, 16 (y3 -

(%i7) /* construct list of eqs */
      yy  : makelist(y[i],i,0,n);

(%o7) [y0, y1, y2, y3, y4]

(%i8) /* construct list of variables */
      linsolve(eqs, yy), numer;

(%o8) [y0 = 0.1, y1 = 0.456, y2 = 0.668, y3 = 0.738, y4 = 0.7]
```

[18] on [maxima-discuss] with my question and Roberts helpful answer

Exercises.

Exercise 170. 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 171. 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. [44, p.417].

Exercise 172. Solve exercise 139 using our `FDM` function.

Exercise 173. 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. [8, p.458].

Exercise 174. 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 175. 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 176. Do the ▷ HECKBERT rocket problem. using our function `FDM(.)`.

Exercise 177. 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 178. Do the ▷ WIKI: Standard boundary value problem. using our `FDM(.)` function.

Exercise 179. Do the BVP's examples 4.2.3, 4.2.5 and 4.2.6 of ▷ NIEMEYER: FDM.

Exercise 180. Do the 2 BVP's ▷ BERKELEY PYTHON NUMERICAL METHODS: FDM.

Exercise 181. Do the BVP's examples ▷ UNI MUENSTER, DE: FDM methods. in §3.2.

Exercise 182. Do the example in ▷ VESELY: Relaxation Method.

Exercise 183. Study ▷ HELLEVIK: NM 4 Engineers: FDM. and ▷ : Differences.

Exercise 184. Do some of the exercises 140 ff using our `FDM(.)` function.

7 Appendix: Collection of EIGENMATH Source Code

This is a collection of relevant definitions from the booklet.
These functions are also collected in the file `odeBox.txt`.
Invoke them with the command `run("odeBox.txt")`. This way you do not have to include the predefined user functions e.g. `iterate/2/3` etc. in your EIGENMATH script file:

```
run("odeBox.txt")
f(t,x,y) = y
g(t,x,y) = x+t
transpose( EULERsys(t,x,y, 0,0,1, 0.1, 10) )
```

Run

$$\begin{bmatrix} 0 & 0 & 1 \\ 0.1 & 0.1 & 1 \\ 0.2 & 0.2 & 1.02 \\ 0.3 & 0.302 & 1.06 \\ 0.4 & 0.408 & 1.1202 \\ 0.5 & 0.52002 & 1.201 \\ 0.6 & 0.64012 & 1.303 \\ 0.7 & 0.77042 & 1.42701 \\ 0.8 & 0.913122 & 1.57406 \\ 0.9 & 1.07053 & 1.74537 \\ 1 & 1.24506 & 1.94242 \end{bmatrix}$$

```
###################################
## ORDINARY DIFFERENTIAL EQUATIONS  (2023) Dr. W.Lindner, Leichlingen GE
###################################

--- ODE type III ---

odefxgy(f,g, x,y, xo,yo) = do(
   print("1. step - identify f(x) and g(y): ", eval(f), eval(g)),
   F = defint(f,x,xo,x),
   G = defint(1/g,y,yo,y),
   print("2. step - calculate F(x)", F),
   print("3. step - calculate G(y)", G) )
```

```
--- ODE type IV : linear ODE  ---

linear1(f,g, x,y, xo,yo) =
             (yo+defint(g/exp(defint(f,x,xo,x)), x,xo,x))*
                         exp(defint(f,x,xo,x))

##### ITERATE

iterate(u,X,Xo,n) = do( M = zero(2,n+1), M[1,1] = Xo,
                        for(i,2,n+1, M[1,i] = eval(u, X,M[1,i-1])), M)

newton0(u,x,xo,n,  M,VAL) = do( M  = zero(2,n),
                             VAL=eval(u,x,xo),
                             for(i,1,n,   VAL   = eval(x-u/d(u,x),x,VAL),
                                          M[1,i]= VAL), M[1,n])

newton1(u,x,a,n) = do( c = float(eval(d(u,x),x,a)),
                       iterate( x-u/c, x, a, n) )

newton(u,x,a,n) = iterate( x - u/d(u,x), x,a, n)

secant1(u,x, a,b, n) = do( C= (a-b)/(eval(u,x,a)-eval(u,x,b)),
                           iterate( x - C*u, x,a,n))


iterate2(u,X,Xo,n) = do( M = zero(n+1,2), M[1] = Xo,
           for(i,2,n+1, M[i] = eval(u, X[1],M[i-1,1], X[2],M[i-1,2])),
           transpose(M))

iterate3(u,X,Xo,n) = do( M = zero(n+1,3), M[1] = Xo,
           for(i,2,n+1,
              M[i] = eval(u, X[1],M[i-1,1], X[2],M[i-1,2], X[3],M[i-1,3])),
           transpose(M) )

idiff(f,x,y,n,  u) = do( M = zero(2,n),
                     u = d(f,x)+d(f,y)*f, M[1,1]=f, M[1,2]=u,
                     for( i,3,n, u = d(u,x)+d(u,y)*f, M[1,i]=u), M[1])

makelist(u,i,m,n) = do( M   = zero(2,n),
                        VAL = eval(u,i,m), M[1,m] = VAL,
                        for( j,m+1,n, VAL = eval(u,i,j), M[1,j]= VAL ), M[1])
```

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


iTaylorsol(x,y, xo,yo, h, m, n) =
           iterate2((x+h, iTaylor(f,x,y, h, m)), (x,y), (xo,yo), n)

##### NUMERICAL METHODS -- f and g are to be defined global

--- EULER method  ---
EULER( x,y, xo,yo, h, n) =  iterate2((x+h, y+h*f(x,y)), (x,y), (xo,yo), n)

--- modified EULER method ---
modiEULER(x,y, xo,yo, h,n,   k1,k2) = do(
                  k1 = f(x,y),
                  k2 = f(x+h, y+h*k1),
                  iterate2((x+h, y+h/2*(k1+k2)),(x,y),(xo,yo),n) )

---  HEUN's method ---
HEUN(x,y, xo,yo, h,n,  k1,k2) = do(
        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) )

MIDPOINT(x,y, xo,yo, h,n) =
              iterate2( (x+h,  y+h*f(x+h/2, y+h/2*f)),
                        (x,    y), (xo,yo), n)

--- classic RUNGE-KUTTA  method of order 4 ---

RK4(x,y,xo,yo,h,n) = do(
                     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/6.0*(1*k1+2*k2+2*k3+1*k4)),
                              (x  , y), (xo,yo), n) )


##### NUMERICAL METHODS for systems of ODEs

---  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)
```

```
---  RUNGE-KUTTA method for systems of IVP ---
RK4sys(x,y,z, xo,yo,zo, h,n)= do(
              k1 = h*f(x,y,z),
              l1 = h*g(x,y,z),
              k2 = h*f(x+h/2, y+k1/2, z+l1/2),
              l2 = h*g(x+h/2, y+k1/2, z+l1/2),
              k3 = h*f(x+h/2, y+k2/2, z+l2/2),
              l3 = h*g(x+h/2, y+k2/2, z+l2/2),
              k4 = h*f(x+h,   y+k3,   z+l3  ),
              l4 = h*g(x+h,   y+k3,   z+l3  ),
      iterate3((x+h,
                y+1/6.0*(1*k1+2*k2+2*k3+1*k4),
                z+1/6.0*(1*l1+2*l2+2*l3+1*l4)),
               (x,y,z), (xo,yo,zo), n)  )

---  RUNGE-KUTTA like method for 2nd order ODE ---
RK4ode2( x,y,yp, xo,yo,ypo, h, n) = do(
         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 = 1/6.0*(k0+2*k1+2*k1p+k2),
        iterate3((x+h, y+h*yp+k, yp+2*l/h),(x,y,yp),(xo,yo,ypo),n) )

---  ROMER's method for 2nd order ODE ---
RK4romer( x,y,yp, xo,yo,ypo, h, n) = do(
                  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+1/6.0*(k1+k2+k3)),
                  yp+1/6.0*(k1+2*k2+2*k3+k4)),
                 (x,y,yp), (xo,yo,ypo), n) )


-- MAXimum of a list of numbers - contributed by G. Weigt
max(L, MAX,i) = do( MAX = L[1],
                    for(i,2,dim(L), test(L[i]>MAX, MAX=L[i])), MAX)
```

8 Appendix: iterate3, RK4sys, RREF for MAXIMA^{online}

Here are three user-defined function for the use in MAXIMA^{online}. We may also use a homemade routine named `rref`[19] to alternatively solve a linear system system.

```
/* MAXIMA on line : iterate3 and RK4sys */
iterate3(u,x,x0,n) := block([numer:true],
 cons(x0, makelist(x0: subst([x[1]=x0[1],x[2]=x0[2],x[3]=x0[3]],u) ,i,1,n)));
```

```
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) )$
```

```
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)$
```

[19] `rref` i.e. Row Reduced Echelon Form; this script is found at ▷ stackoverflow: `rref`

9 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. (1991): *FORTRAN 77 and Numerical Methods for Engineers.* Boston: PWS Publishing.

[8] Borse, G.J. (1997): *Numerical Methods with MATLAB.* 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. (1994): *Höhere Analysis mit* Derive. Braunschweig: Vieweg.

[26] Koepf, W. (1996): Derive *für den Mathematikunterricht.* 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] Lindner, W. (2023): *Introductory Differential Equations with* Maxima. url: `https://lindnerdrwg.github.io/DifferentialEquationsMaxima.pdf`

[30] Lowe, B. & Berry, J.S. (1998): *Learning Differential Equations through* Derive. Lund: Studentlitteratur.

[31] MathSoft (1995): *Exploring Numerical Recipes.* Cambridge: MathSoft Inc. (MathCAD)

[32] Marsden, J. & Weinstein, A. (^{2}1985): *Calculus I.* New York: Springer.

[33] Marsden, J. & Weinstein, A. (^{2}1985): *Calculus II.* New York: Springer.

[34] Marsden, J. & Weinstein, A. (^{2}1985): *Calculus III.* New York: Springer.

[35] Minorski, W. P. ([2]1965): *Aufgabensammlung der Höheren Mathematik.* Leipzig: VEB Fachbuchverlag.

[36] Mohr, R. (1998): *Numerische Methoden in der Technik* (MatLab). Braunschweig: Vieweg.

[37] Nakamura, S. (1993): *Applied Numerical Methods in C.* Eaglewood Cliffs: Prentice-Hall.

[38] Opfer, G. (2008): *Numerische Mathematik für Anfänger.* Wiesbaden: Vieweg+Teubner.

[39] Press, W.H. & al. (1988): *Numerical Recipes in C.* Cambridge: Cambridge University Press.

[40] Quarteroni, A. et al. (2000): *Numerical Mathematics.* London, New York: Springer.

[41] Rich, A. & Stoutemyer, D. et al. (1988 ff): Derive *User Manual.* Honolulu: Soft Warehouse..

[42] Roberts, L.F. (1995): *Introduction to Ordinary Differential Equations.* Cambridge: MathSoft Inc. (MathCAD)

[43] Romer, A. (1983): *50* BASIC*-Programme.* Mannheim: Bibliographisches Institut.

[44] Schonefeld, S. (1994): *Numerical Analysis via* Derive. Urbana: MathWare.

[45] Spiegel, M. R. (1963): *Advanced Calculus.* New York: McGraw-Hill.

[46] Steeb, W.-H & Lewien, D. (1991): *Algorithms and Computation with* Reduce. Mannheim: Bibliographisches Institut.

[47] Stoer, J. & Bulirsch, R.Z. (1973): *Einführung in die Numerische Mathematik II.* Berlin: Springer.

[48] Süli, E. (2022): *Numerical Solution of Ordinary Differential Equations.* url: `https://people.maths.ox.ac.uk/suli/nsodes.pdf`

[49] Timberlake, T.K. & Mixon, J.W. (2016): *Classical Mechanics with* Maxima. New York: Springer.

[50] Venz, G. (1978): *Lösung von Differentialgleichungen mit programmierbaren Taschenrechnern.* München, Wien: Oldenburg.

[51] Weigt, G. (2021): Eigenmath *Homepage.* url: `https://georgeweigt.github.io`

[52] Weigt, G. (2022): Eigenmath *Apple app for* iMac.
url: https://apps.apple.com/de/app/eigenmath/id584939279?mt=12

[53] Weigt, G. (2021): Eigenmath *online.*
url: https://georgeweigt.github.io/eigenmath-demo.html

[54] Werner, W. ([2]1998): *Mathematik lernen mit* Maple. *Band 2.* Heidelberg: dpunkt.

[55] Wolfram mathworld: Picards Existence Theorem.
url: https://mathworld.wolfram.com/PicardsExistenceTheorem.html

[56] Woollett, E.L. (2023): *Maxima by Example, Ch. 3: Ordinary Differential Equation Tools.* url: https://home.csulb.edu/~woollett/mbe03.html