Analysis and object oriented implementation of the Kovacic algorithm

Nasser M. Abbasi*

Abstract

This paper gives a detailed overview and a number of worked out examples illustrating the Kovacic [1] algorithm for solving second order linear differential equation $A(x)y'' + B(x)y' + C(x)y = 0$ where $A,B,C$ are rational functions with complex coefficients in the independent variable $x$. All three cases of the algorithm were implemented in a software package based on an object oriented design and complete source code listing given in the appendix with usage examples. Implementation used the Maple computer algebra language[1] This package was then used to analyze the distribution of Kovacic algorithm cases on 3000 differential equations.

1 Introduction

2 Description of algorithm for each case

2.1 Case one ................................................................. 3
2.2 Case two ................................................................. 8
2.3 Case three ................................................................. 11
2.4 Statistics and discussion of results obtained using Kovacic algorithm .... 14

3 Worked example for each case

3.1 case one ................................................................. 16
3.2 case two ................................................................. 27
3.3 case three ................................................................. 32

4 Conclusion

5 Appendix

5.1 Instructions and examples using the Kovacic package ......................... 42
5.2 Source code ............................................................... 44

*https://12000.org/
1The complete Kovacic package in one mpl file accompany the arXiv version of this paper
1 Introduction

Kovacic [1] gave an algorithm for finding a closed form Liouvillian solution to any linear second order differential equation $Ay'' + By' + Cy = 0$ if such a solution exists. Smith [2] gave an implementation based on a modified version of Kovacic algorithm by Saunders [3].

The current implementation is based on the original paper by Kovacic and uses the new object oriented features in Maple. The accompanied software package have been tested on 3000 differential equations with each solution verified using Maple’s odetest. The test suite is included as a separate module. The Appendix describes how to use the software.

The Kovacic algorithm finds one (basis) solution of $Ay'' + By' + Cy = 0$. The second basis solution is found using reduction of order. The general solution is a linear combination of the two basis solutions found.

The algorithm starts by writing the input ode $Ay'' + By' + Cy = 0$ as

$$y'' + ay' + by = 0 \quad (1)$$

Where $a = \frac{B}{A}$, $b = \frac{C}{A}$. The substitution

$$z = ye^{\frac{1}{2} \int a \, dx} \quad (2)$$

is then applied to (1) which transforms it to a second order ode in the new dependent variable $z(x)$ without the first derivative

$$z'' = rz \quad (3)$$

$r$ in the above is given by

$$r = \frac{1}{4}a^2 + \frac{1}{2}a' - b \quad (4)$$

It is ode (3) which is solved by the algorithm and not (1). Equation (3) will be called the DE from now on.

If a solution $z(x)$ to the DE is found, then the first basis solution to the original ode is obtained using the transformation (2) in reverse

$$y = ze^{-\frac{1}{2} \int a \, dx}$$

The second solution is found using reduction of order.

---

1 Wikipedia defines Liouvillian function as function of one variable which is the composition of a finite number of arithmetic operations ($+, -, \times, \div$), exponentials, constants, solutions of algebraic equations (a generalization of nth roots), and antiderivatives. Kovacic in his original paper says “Such a solution may involve exponentials, indefinite integrals and solutions of polynomial equations. (As we are considering functions of a complex variable, we need not explicitly mention trigonometric functions, they can be written in terms of exponentials. Note that logarithms are indefinite integrals and hence are allowed.”
These are the four possible cases to consider:

1. DE has solution \( z = e^{\int \omega \, dx} \) with \( \omega \in \mathbb{C}(x) \).

2. DE has solution \( z = e^{\int \omega \, dx} \) with \( \omega \) polynomial over \( \mathbb{C}(x) \) of degree 2.

3. Solutions of DE are algebraic over \( \mathbb{C}(x) \).

4. DE has no Liouvillian solution.

Before describing how the algorithm works, there are necessary (but not sufficient) conditions that determine which case the DE satisfies. Only those cases that meet the necessary conditions will be attempted.

The following are the necessary conditions for each case. To check each case, let \( r = \frac{s}{t} \) where \( \gcd(s,t) = 1 \). The order of \( r \) at \( \infty \) (from now on referred to as \( \mathcal{O}(\infty) \)) is defined as \( \deg(t) - \deg(s) \). The poles of \( r \) and the order of each pole need to be determined.

Knowing the order of the poles of \( r \) and \( \mathcal{O}(\infty) \) is all what is needed to determine the necessary conditions for each case. These conditions are the following

1. Case 1. Either no pole exists, or if a pole exists, the order must be either one or even. If \( \mathcal{O}(\infty) \) is less 3, then it must be even otherwise it can be even or odd.

2. Case 2. \( r \) must have at least one pole either of order 2 or odd order greater than 2. There are no conditions on \( \mathcal{O}(\infty) \).

3. Case 3. \( r \) must have a pole either of order 1 or 2. No other order is allowed. \( \mathcal{O}(\infty) \) must be at least 2.

If the conditions of a case are not satisfied then the case will be attempted as the algorithm guarantees that there will be no Liouvillian solution. However if the conditions are satisfied, this does not necessarily mean a solution exists. As an example \( y'' = \frac{1}{x^6} y \) satisfies only case one, but running the algorithm on case one shows that there is no Liouvillian solution.

The following table summarizes the above conditions for each case.

| Case | Allowed pole order for \( r \) | Allowed value for \( \mathcal{O}(\infty) \) |
|------|--------------------------------|----------------------------------------|
| 1    | \{0, 1, 2, 4, 6, 8, \ldots\}   | \{\ldots, -6, -4, -2, 0, 2, 3, 4, 5, 6, \ldots\} |
| 2    | Need to have at least one pole of order 2 or pole of odd order greater than 2. Any other pole order is allowed as long as the above condition is satisfied. The following are examples of pole orders which are allowed. \{1, 2\}, \{1, 3\}, \{2\}, \{3\}, \{3, 4\}, \{1, 2, 5\}. | no conditions |
| 3    | \{1, 2\}                       | \{2, 3, 4, 5, 6, 7, \ldots\}           |

Table 1: Necessary conditions for each Kovacic case
Some observations: In case one, no odd order pole is allowed except for order 1. Case one is the only case that could have no pole in \( r \), which is the same as a pole of order zero. Case two and three require at least one pole. For case three, only poles of order 1 or 2 are allowed. If \( \Theta(\infty) \) is zero, then only possibility is either case one or two. For case one, if \( \Theta(\infty) \) is negative, then it must be even.

The above table also shows that when \( r \) has only one pole of order 2 and \( \Theta(\infty) \) equals 2 or higher then all three cases are possible. Also, if \( r \) has two poles one of order 1 and the other of order 2 and \( \Theta(\infty) \) equals 2 or higher then all three cases are possible.

These are the only two possibilities where all three cases have the same necessary conditions.

### 2 Description of algorithm for each case

#### 2.1 Case one

**2.1.1 step 1**

Assuming that the necessary conditions for case one are satisfied and \( z'' = rz, r = \frac{z}{z} \). Let \( \Gamma \) be the set of all poles of \( r \). For each pole \( c \) in this set, three quantities are calculated: Rational function \([\sqrt{r}]_c\) and two complex numbers \( a_c^+, a_c^- \).

How this is done depends on the order of the pole as described below. If the set \( \Gamma \) is empty (when there are no poles), then this part is skipped.

1. If the pole \( c \) has order 1 then
   \[ [\sqrt{r}]_c = 0 \]
   \[ a_c^+ = 1 \]
   \[ a_c^- = 1 \]

2. If the pole \( c \) is of order 2 then
   \[ [\sqrt{r}]_c = 0 \]
   \[ a_c^+ = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b} \]
   \[ a_c^- = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b} \]

   Where \( b \) is the coefficient of \( \frac{1}{(x-c)^2} \) in the partial fraction decomposition of \( r \).

3. If the pole is of order \( \{4, 6, 8, \ldots\} \) (poles must be all even from the conditions of case one), then the computation is more involved. Let \( 2v \) be the order of the pole. Hence if the pole was order 4, then \( v = 2 \). Let \([\sqrt{r}]_c\) be the sum of terms involving \( \frac{1}{(x-c)^i} \) for \( 2 \leq i \leq v \) in the Laurent series expansion of \( \sqrt{r} \) (not \( r \)) at \( c \). Therefore

   \[ [\sqrt{r}]_c = \sum_{i=2}^{v} \frac{a_i}{(x-c)^i} \]
   \[ = \frac{a_2}{(x-c)^2} + \frac{a_3}{(x-c)^3} + \cdots + \frac{a_v}{(x-c)^v} \] (1)
$\alpha^+_c, \alpha^-_c$ are found using

$$\alpha^+_c = \frac{1}{2}\left(\frac{b}{a_v} + v\right)$$
$$\alpha^-_c = \frac{1}{2}\left(-\frac{b}{a_v} + v\right)$$

Where in the above $a_v$ is the coefficient of the term $\frac{a_v}{(x-c)^{\gamma+1}}$ in (1) and $b$ is the coefficient of the term $\frac{1}{(x-c)^{\gamma+1}}$ in $r$ itself (found from the partial fraction decomposition), minus the coefficient of same term in the Laurent series expansion of $\sqrt{r}$ at $c$.

The coefficients in the Laurent series can be obtained as follows. Given $r(x)$ with a pole of finite order $N$ at $x = c$, then its Laurent series expansion at $c$ is given by the sum of the analytic part and the principal part of the of the Laurent series. The coefficients $b_n$ are contained in the principal part of the series.

$$r(x) = \sum_{n=0}^{\infty} a_n (x-c)^n + \frac{b_1}{(x-c)} + \frac{b_2}{(x-c)^2} + \frac{b_3}{(x-c)^3} + \cdots + \frac{b_N}{(x-c)^N}$$

To obtain $b_1$ (which is the residue of $r(x)$ at $c$), both sides of the above are multiplied by $(x-c)^N$ which gives

$$(x-c)^N r(x) = \sum_{n=0}^{\infty} a_n (x-c)^{n+N} + b_1(x-c)^{N-1} + b_2(x-c)^{N-2} + \cdots + b_N$$

Differentiating both sides of (3) $(N-1)$ times w.r.t. $x$ gives

$$\frac{d^{N-1}}{dx^{(N-1)}} (x-c)^N f(x) = \sum_{n=0}^{\infty} \frac{d^{N-1}}{dx^{(N-1)}} (a_n (x-c)^{n+N}) + b_1(N-1)!$$

Evaluating the above at $x = c$ gives

$$b_1 = \lim_{x \to c} \frac{d^{N-1}}{dx^{(N-1)}} (x-c)^N r(x) \cdot \frac{b_1}{(N-1)!}$$

To find the next coefficient $b_2$, both sides of (3) are differentiated $(N-2)$ times

$$\frac{d^{N-2}}{dx^{(N-2)}} (x-c)^N r(x) = \sum_{n=0}^{\infty} \frac{d^{N-2}}{dx^{(N-2)}} (a_n (x-c)^{n+N}) + b_1(N-1)! (x-c) + b_2(N-2)!$$

Evaluating the above at $x = c$ gives

$$b_2 = \lim_{x \to c} \frac{d^{N-2}}{dx^{(N-2)}} (x-c)^N r(x) \cdot \frac{b_2}{(N-2)!}$$
The above is repeated to find $b_3, b_4, \cdots, b_N$. The general formula for find coefficient $b_n$ is therefore

$$b_n = \lim_{x \to c} \frac{d^{N-n}}{dx^{N-n}} \left( (x - c)^N r(x) \right) \frac{(N-n)!}{(N-n)!}$$

(4)

For the special case of the last term $b_N$ the above simplifies to

$$b_N = \lim_{x \to c} (x - c)^N r(x)$$

(5)

The above is implemented in the function `laurent_coeff()` in the Kovacic class.

This completes finding all the quantities $\{ \left[ r \right]_0^c, \alpha^+ + c, \alpha^- + c \}$ for each pole in the set $\Gamma$ for case one.

The next step calculates the following three quantities for $\Theta(\infty)$.

1. If $\Theta(\infty) \leq 0$, which must be even, then let $-2v = \Theta(\infty)$ and $\sqrt{r}$ is then the sum of all terms $x^i$ for for $0 \leq i \leq v$ in the Laurent series expansion of $\sqrt{r}$ at $\infty$.

$$\left[ \sqrt{r} \right]_\infty = \sum_{i=0}^v a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_v x^v$$

(6)

The coefficients $a_i$ are found by setting $x = \frac{1}{y}$ in $r$ and then finding the Laurent series of $\sqrt{r(y)}$ expanded around $y = 0$. The process for finding the coefficient is the same one used as described earlier where now the limit is taken as $y$ approaches zero from the right. This gives all the terms of (6). This is implemented in the function `laurent_coeff()` in the Kovacic class.

The corresponding $\{a^+_\infty, a^-\infty\}$ are given by

$$a^+_\infty = \frac{1}{2} \left( b - a_v \right)$$

$$a^-\infty = \frac{1}{2} \left( -b + a_v \right)$$

Where $a_v$ is coefficient of $x^v$ in (6) and $b$ is the coefficient of $x^{v-1}$ in $r$ itself (found using long division) minus the coefficient of $x^{v-1}$ in $\left( \left[ \sqrt{r} \right]_\infty \right)^2$.

2. If $\Theta(\infty) = 2$ then $\left[ \sqrt{r} \right]_\infty = 0$. The corresponding $\{a^+_\infty, a^-\infty\}$ are given by

$$a^+_\infty = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b}$$

$$a^-\infty = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b}$$

Here $b = \frac{\text{lcoeff}(s)}{\text{lcoeff}(t)}$ where $r = \frac{s}{t}$. lcoeff($s$) is the leading coefficient of $s$ and similarly, lcoeff($t$) is the leading coefficient of $t$.

3. If $\Theta(\infty) > 2$ then

$$\left[ \sqrt{r} \right]_\infty = 0$$

$$a^+_\infty = 0$$

$$a^-\infty = 1$$
2.1.2 step 2

Using quantities calculated in step 1, the algorithm now searches for a non-negative integer \( d \) using

\[
d = a^+_{\infty} - \sum_{c \in \Gamma} a^+_c
\]

If non-negative \( d \) is found, a candidate \( \omega_d \) is calculated using

\[
\omega_d = \sum_{c \in \Gamma} \left( (\pm) \left[ \sqrt{r} \right]_c + \frac{a^+_c}{x-c} \right) + (\pm) \left[ \sqrt{r} \right]_{\infty}
\]

If no non-negative integer \( d \) could be found, then no Liouvillian solution exists using this case. Case two or three are tried next if these are available.

2.1.3 step 3

In this step the algorithm finds polynomial \( p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + x^d \) of degree \( d \). This is done by solving for the coefficients \( a_i \) from

\[
p'' + 2\omega p' + (\omega' + \omega^2 - r) p = 0 \tag{7}
\]

Where \( \omega \) is from the second step above and \( r \) is from \( z'' = rz \).

For an example, if \( d = 2 \), then \( p(x) = x^2 + a_1 x + a_0 \) is substituted in (3) and \( a_0, a_1 \) are solved for. If solution exists, then the solution to \( z'' = rz \) will be

\[
z = p(x)e^{\int \omega \, dx}
\]

If the degree \( d = 1 \) then \( p(x) = x + a_0 \) and the same process is applied. If the degree \( d = 0 \), then \( p(x) = 1 \).

The first basis solution to the original ode is now be found from

\[
y_1 = ze^{-\frac{1}{2} \int a \, dx}
\]

And the second basis solution using reduction of order formula is

\[
y_2 = y_1 \int \frac{e^{-\int a \, dx}}{y_1^2} \, dx
\]

Hence the general solution to the original ode is

\[
y(x) = c_1 y_1 + c_2 y_2
\]

This completes the full algorithm for case 1. The part that needs most care is in finding \( \{ \left[ \sqrt{r} \right]_c, a^+_c, \left[ \sqrt{r} \right]_{\infty}, a^+_{\infty} \} \). Once these are calculated, the rest of the algorithm is much more direct.
2.1.4 Algorithm flow chart for case one

Case One Algorithm

Step 1

\[ z'' = rz \]

\[ \Gamma \]

\[ \text{poles of } r \]

\[ \text{pole } c_1 \text{ of order 1} \]

\[ \text{pole } c_2 \text{ of order 2} \]

\[ \text{of order } \geq 4 \text{ (must be even order). Let } v \text{ be half the order value.} \]

Order of \( r \) at \( \infty \)

Order is > 2

\[ \left[ \sqrt{r} \right]_{\infty} = 0 \]

\[ \alpha_{c_1}^+ = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b} \]

\[ \alpha_{c_1}^- = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b} \]

This is the sum of terms of \( \frac{1}{x - c_3} \) in the Laurent series expansion of \( \sqrt{r} \) (not \( r \)) at the pole \( c_3 \). The sum of terms is for \( 2 \leq i \leq v \).

Order is 2

\[ \left[ \sqrt{r} \right]_{\infty} = 0 \]

\[ \alpha_{c_2}^+ = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b} \]

\[ \alpha_{c_2}^- = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b} \]

This is the sum of terms of \( \frac{1}{x - c_3} \) in the Laurent series expansion of \( r \) at \( \infty \). If \( r = \frac{a}{x^2} \) where \( \gcd(s, t) = 1 \) then \( b \) can be more easily found using \( b = \frac{\text{lcoeff}(s)}{\text{lcoeff}(s)} \) where \text{lcoeff} means the leading coefficient.

Order is > 2

\[ \left[ \sqrt{r} \right]_{\infty} = 0 \]

\[ \alpha_{c_2}^+ = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b} \]

\[ \alpha_{c_2}^- = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b} \]

This is the sum of terms \( x^i \) for \( 0 \leq i \leq v \) in the Laurent series for \( \sqrt{r} \) (not \( r \)) at \( \infty \).

This is the sum of terms of \( \frac{1}{x - c_3} \) in the above Laurent sum.

Order is > 2

\[ \left[ \sqrt{r} \right]_{\infty} = 0 \]

\[ \alpha_{c_3}^+ = \frac{1}{2} + \frac{1}{2} \sqrt{1 + 4b} \]

\[ \alpha_{c_3}^- = \frac{1}{2} - \frac{1}{2} \sqrt{1 + 4b} \]

This is the sum of terms \( \frac{1}{x - c_3} \) in the above Laurent sum.

Step 2

For each family \( s = (s(c))_{c \in \Gamma \cup \infty} \) where \( s(c) \) is + or − let

\[ d = \alpha_{c_1}^+(\infty) - \sum_{c \in \Gamma} \alpha_{c_1}^+(c) \]

If family found which produced \( d \) an integer and positive then find

\[ \omega = \sum_{c \in \Gamma} (s(c) \sqrt{r} + \alpha_{c_1}^+(c)) + s(\infty) \sqrt{r} \]

Step 3

Find polynomial \( p(x) \) of degree \( d \) which satisfies \( p'' + 2\omega p' + (\omega^2 + \omega x - r)p = 0 \). Then the solution to \( z'' = rz \) is given by

\[ z = p.e^{\int \omega dx} \]

Figure 1: Case 1 Kovacic algorithm
2.2 Case two

2.2.1 step 1

Assuming that the necessary conditions for case two are satisfied and \( z'' = rz, r = \frac{s}{t} \). Let \( \Gamma \) be the set of all poles of \( r \). For each pole \( c \) in this set, \( E_c \) is found as follows

1. If the pole \( c \) has order 1 then \( E_c = \{4\} \).

2. If the pole \( c \) is of order 2 then \( E_c = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\} \) where \( b \) is the coefficient of \( \frac{1}{(x-c)^2} \) in the partial fraction decomposition of \( r \). In the above set \( E_c \), only integer values are kept.

3. If the pole \( c \) is of order \( v > 2 \) then \( E_c = \{v\} \)

The next step is to determine \( E_\infty \).

1. If \( \Theta(\infty) > 2 \) then \( E_\infty = \{0, 2, 4\} \)

2. If \( \Theta(\infty) = 2 \) then \( E_\infty = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\} \) where \( b = \frac{\text{lcoef}(s)}{\text{lcoef}(t)} \) where \( r = \frac{s}{t} \). \( \text{lcoef}(s) \) is the leading coefficient of \( s \) and similarly \( \text{lcoef}(t) \) is the leading coefficient of \( t \). In the above set \( E_\infty \) only integer values are kept.

3. If \( \Theta(\infty) < 2 \) then \( E_\infty = \Theta(\infty) \).

2.2.2 step 2

Using quantities calculated in step 1, the algorithm now searches for a non-negative integer \( d \) using

\[
d = \frac{1}{2} \left( e_\infty - \sum_{c \in \Gamma} e_c \right)
\]

Where in the above \( e_c \in E_c, e_\infty \in E_\infty \) found in step 1. If non-negative \( d \) is found, then

\[
\theta = \frac{1}{2} \sum_{c \in \Gamma} \frac{e_c}{x-c}
\]

If no non-negative integer \( d \) could be found, then no Liouvillian solution exists using this case. Case three is tried next if it is available.

2.2.3 step 3

In this step the algorithm determines a polynomial \( p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + x^d \) of degree \( d \). This is done by solving for the coefficients \( a_i \) from

\[
p''' + 3\theta p'' + (3\theta^2 + 3\theta' - 4r)p' + (\theta'' + 3\theta\theta' + \theta^3 - 4r\theta - 2r^2)p = 0
\]
Where $\theta$ was found in step 2 and $r$ is from $z'' = rz$. If $p(x)$ can be found that satisfies (1) then

$$\phi = \theta + \frac{p'}{p}$$

(2)

$\omega$ is then solved for from

$$\omega^2 - \phi \omega + \left(\frac{1}{2} \phi' + \frac{1}{2} \phi^2 - r\right) = 0$$

(3)

If solution $\omega$ to (3) can be found, then the solution to $z'' = rz$ is given by

$$z = e^{\int \omega dx}$$

This completes the full algorithm for case two. The general solution to the original ode is now determined as outlined at the end of case one above.
2.2.4 Algorithm flow chart for case two

Case Two Algorithm

Step 1

For each family \((c_\epsilon)_{\epsilon \in \Gamma \cup \infty}\) with \(c_\epsilon \in E_\epsilon\) let

\[
d = \frac{1}{2} \left( e_\infty - \sum_{\epsilon \in \Gamma} e_\epsilon \right)
\]

If family found which produced \(d\) an integer and positive then find

\[
\theta = \frac{1}{2} \sum_{\epsilon \in \Gamma} \frac{e_\epsilon}{x - c}
\]

Step 2

Find polynomial \(p(x)\) of degree \(d\) which satisfies

\[
p''' + 3\theta p'' + (3\theta^2 + 3\theta - 4r)p' + (\theta^3 + 3\theta\phi + \theta^2 - 4r\theta - 2r^2)p = 0
\]

Let

\[
\phi = \theta + \frac{p'}{p}
\]

The find solution \(w\) for the equation

\[
w^2 - \phi w + \left( \frac{1}{2} \phi^2 + \frac{1}{2} \phi^2 - r \right) = 0
\]

If solution can be found then

\[
z = e^{\int w \, dx}
\]

Is the solution to \(z'' = rz\)

Figure 2: Case 2 Kovacic algorithm

10
2.3 Case three

2.3.1 step 1

Assuming the necessary conditions for case three are satisfied and \( z'' = rz, r = \frac{\xi}{\iota} \). Let \( \Gamma \) be the set of all poles of \( r \). Recall that case three can have either a pole of order 1 or order 2 only. For each pole \( c \) in this set, \( E_c \) is found as follows

1. If the pole \( c \) has order 1 then \( E_c = \{12\} \).

2. If the pole \( c \) is of order 2 then

\[
E_c = \left\{ 6 + \frac{12k}{n} \sqrt{1 + 4b} \right\} \quad \text{for} \quad k = -\frac{n}{2} \cdots \frac{n}{2}
\]  

(1)

Where \( k \) is incremented by 1 each time, and \( n \) is any of \( \{4, 6, 12\} \) and \( b \) is the coefficient of \( \frac{1}{(x-c)^2} \) in the partial fraction decomposition of \( r \). In the above set \( E_c \), only integer values are kept. For an example, when \( n = 4 \) then \( k = \{-2, -1, 0, 1, 2\} \) and \( E_c = \{6 - 6\sqrt{1 + 4b}, 6 - 3\sqrt{1 + 4b}, 6, 6 + 3\sqrt{1 + 4b}, 6 + 6\sqrt{1 + 4b}\} \) and similarly for \( n = 6 \) and \( n = 12 \).

The next step determines \( E_\infty \). This is found using same formula as (1) but \( b \) is calculated differently using \( b = \frac{l\text{coeff}(s)}{l\text{coeff}(t)} \) where \( r = \frac{\xi}{\iota} \). \( l\text{coeff}(s) \) is the leading coefficient of \( s \) and \( l\text{coeff}(t) \) is the leading coefficient of \( t \).

2.3.2 step 2

Using quantities calculated in step 1, the algorithm now searches for a non-negative integer \( d \) using

\[
d = \frac{n}{12} \left( e_\infty - \sum_{c \in \Gamma} e_c \right)
\]

Where in the above \( e_c \in E_c, e_\infty \in E_\infty \) \( n \) is any of \( \{4, 6, 12\} \) values. If non-negative \( d \) is found, then

\[
\theta = \frac{n}{12} \sum_{c \in \Gamma} \frac{e_c}{x - c}
\]

The sum above is over all families of \( \{e_\infty, e_c\} \) which generated the non-negative integer \( d \). Next define

\[
S = \prod_{c \in \Gamma} (x - c)
\]

The product above is over families of \( \{e_\infty, e_c\} \) which generated the non-negative integer \( d \). If no non-negative integer \( d \) is found, then no Liouvillian solution exists.
2.3.3 step 3

In this step the algorithm determines a polynomial \( p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + x^d \) of degree \( d \). Define set of polynomials \( \{P_n, P_{n-1}, \ldots, P_{-1}\} \) where

\[
P_n = -p(x) \\
P_{i-1} = -SP_i' + ((n-i)S' - S\theta) P_i - (n-i)(i+1)S^2 rP_{i+1} \quad i = n \cdots 0
\]

The last polynomial \( P_{-1}(x) \) is used to solve for the coefficients \( a_i \) using

\[
P_{-1}(x) = 0 \quad (2)
\]

In Maple this is done using the `solve` command with the `identity` option. If it is possible to find coefficients \( a_i \) such that (2) is satisfied, then define the equation

\[
\sum_{i=0}^{n} \frac{S^i P_i(x)}{(n-i)!} \omega^i = 0
\]

\( \omega \) is solved for from the above equation. If solution \( \omega \) is found then the solution to \( z'' = rz \) will be

\[
z = e^{\int \omega \, dx}
\]

This completes the full algorithm for case three. The general solution to the original ode can now be determined as outlined at the end of case one above.
2.3.4 Algorithm flow chart for case three

Case 3 Algorithm

Step 1

\[ z'' = rz \]

\( poles \) of \( r \)

\( \Gamma \) (set of poles)

\( E_c = \{12\} \)

Order of \( r \) at \( \infty \)

\[ E_c = \left\{ 6 + \frac{12k}{n} \sqrt{1 + 4b} \mid k = 0, \pm 1, \pm 2, \ldots, \pm \frac{n}{2} \right\} \cap \mathbb{Z} \]

(keep only integer values)

Where \( b \) is the coefficient of \( \frac{1}{x^2} \) in the partial fractions expansion of \( r \).

For each family \( (e_c)_{c \in \Gamma} \) with \( e_c \in E_c \) let

\[ d = \frac{1}{2} \left( e_{\infty} - \sum_{c \in \Gamma} e_c \right) \]

If family found which produced \( d \) an integer and positive then find

\[ \theta = \frac{n}{12} \sum_{c \in \Gamma} \frac{e_c}{x - c} \]

And

\[ S = \prod_{c \in \Gamma} (x - c) \]

Step 2

Let polynomial \( p(x) \) of degree \( d \) with coefficients \( a_i \).

The following set of equations are set up in order to determine \( p(x) \).

\[ P_0 = -p(x) \]

\[ P_{i-1} = -SP'_i + ((n - i)S'^{-1} - 2S)P_i - (n - i)(i + 1)S^2 rP_{i+1} \quad i = n \cdot 0 \]

Where \( n \) above is either 4, 6 or 12. The last polynomial \( P_{n-1}(x) \) is used to solve for the coefficients \( a_i \) using

\[ P_{n-1}(x) = 0 \]

The equation for \( \omega \) is found using

\[ \sum_{i=0}^{n} \frac{S^i P_i(x)}{(n - i)!} \omega^i = 0 \]

If solution \( \omega \) can be found then

\[ z = e^{\int \omega \, dx} \]

Is the solution to \( z'' = rz \)

Step 3

Figure 3: Case 3 Kovacic algorithm
2.4 Statistics and discussion of results obtained using Kovacic algorithm

This gives summary of results obtained using testsuite of 3000 differential equations, all of which were selected as linear with rational coefficients as functions of $x$ that can be solved using this algorithm.

Theode's used in the testsuite were collected by the author and stored in sql database. These were collected from a number of standard textbooks and other references such as “Differential Equations. E. Kamke. 3th edition. Chelsea.” and “Ordinary Differential Equations And Their Solutions. Murphy, George Moseley. Dover. 2011”.

All the ode’s were successfully solved using the Kovacic algorithm as implemented here and each solution was verified using Maple odetest.

The following diagram shows the percentage of ode's solved using each case.

![Figure 4: Kovacic cases distributions](image)

Case 3 was required for solving only 3 odes. It used $n = 4$ for all 3 ode's. $n = 6$ and $n = 12$ were not reached or required to try. Recall that $n$ for case 3 is the degree of the polynomial in $\omega$ used to solve for in order to find the $z$ solution from $z = e^{\int \omega dx}$.

This result shows that case 1 and 2 combined is all what is needed to solve 99.9% of ode's used in practice. Larger collection of ode's than the 3000 used could produce different results, but the overall trend is that case 3 is rarely needed in practice and within case 3, $n = 6$ and $n = 12$ are even less likely to be required.

When forcing the algorithm to use case 3 and only use $n = 12$, this resulted in a very long computation time on some ode's. For an example, using ode $y'' + xy' + y = 0$ which satisfies all three cases, and asking the solver to use case 3 and $n = 12$, it was found that it required $p(x)$ of degree $d = 24$ in order to find $\omega$ of degree 12 that can be solved. The total number of trials in step 3 of case three to find such solution was found to be 2367. This took over 30 minutes to complete.
In comparison, the same ode was solved using case one in less than one second giving the same solution on the same computer.

The testsuite also calculates the distribution of cases which has its necessary conditions satisfied for each ode. Recall that having the necessary conditions for a case satisfied does not mean a solution would be found using that case. The following bar chart shows the percentages of the 3000 ode’s that satisfied the necessary conditions each case. This chart shows that many ode’s satisfy the conditions for more than one case at the same time.

Figure 5: Percentage of ode’s that satisfy each Kovacic case necessary conditions
3 Worked example for each case

3.1 case one

3.1.1 Example 1

Given the ode

$$(2x + 1)y'' - 2y' - (2x + 3)y = 0$$

Converting it $y'' + ay' + by = 0$ gives

$$y'' - \frac{2}{2x+1}y' - \frac{2x + 3}{2x+1}y = 0$$

Where $a = -\frac{2}{2x+1}$, $b = -\frac{2x + 3}{2x+1}$. Applying the transformation $z = ye^{\int \frac{1}{a} dx}$ gives $z'' = rz$ where $r = \frac{1}{4}a^2 + \frac{1}{2}a' - b$. This results in

$$r = \frac{s}{t} = \frac{4x^2 + 8x + 6}{(2x+1)^2} = \frac{s}{t}$$

There is one pole at $x = -\frac{1}{2}$, hence $\Gamma = \{-\frac{1}{2}\}$. The order is 2 and $O(\infty) = \deg(t) - \deg(s) = 0$. Table 1 shows that the necessary conditions for case one and two are both satisfied. This is solved first using case one. Since the order of the pole is 2, then

$$[\sqrt{r}]_c = 0 \quad (1)$$

$$\alpha^+_c = \frac{1}{2} + \frac{1}{2} \sqrt{1 + \frac{3}{4}b}$$

$$\alpha^-_c = \frac{1}{2} - \frac{1}{2} \sqrt{1 + \frac{3}{4}b}$$

Where $b$ is the coefficient of $\frac{1}{(x-c)^2} = \frac{1}{(x+\frac{1}{2})^2}$ in the partial fraction decomposition of $r$ which is $r = 1 + \frac{3}{4} \frac{1}{(x+\frac{1}{2})^2} + \frac{1}{x+\frac{1}{2}}$. Therefore $b = \frac{3}{4}$ and the above becomes

$$[\sqrt{r}]_c = 0$$

$$\alpha^+_c = \frac{1}{2} + \frac{1}{2} \sqrt{1 + \frac{3}{4}} = \frac{3}{2}$$

$$\alpha^-_c = \frac{1}{2} - \frac{1}{2} \sqrt{1 + \frac{3}{4}} = -\frac{1}{2}$$

Since $O(\infty) = 0$ then $v = 0$ and $[\sqrt{r}]_\infty$ is the sum of all terms $x^i$ for $0 \leq i \leq v$ in the Laurent series expansion of $\sqrt{r}$ at $\infty$ which is found as follows. Since $\sqrt{r} = \sqrt{\frac{4x^2 + 8x + 6}{(2x+1)^2}}$

then setting $x = \frac{1}{y}$ gives $\sqrt{r(y)} = \sqrt{\frac{4\left(\frac{1}{y}\right)^2 + 8\frac{1}{y} + 6}{(2\frac{1}{y} + 1)^2}}$ and since $v = 0$ then the constant term is

$$\lim_{y \to 0} \sqrt{r(y)} = 1.$$

Therefore $[\sqrt{r(x)}]_\infty = 1$. Hence $a = 1$. 16
$b$ is the coefficient of $x^{-1} = \frac{1}{x}$ in $r$ minus the coefficient of $\frac{1}{x}$ in $\left(\sqrt{r}\right)_\infty = 1$ which is zero since there is no term $\frac{1}{x}$. Because $v = 0$, long division is used to find the coefficient of $\frac{1}{x}$ in $r$.

\[
r = \frac{s}{t} = \frac{4x^2 + 8x + 6}{4x^2 + 4x + 1} = Q + \frac{R}{t} = 1 + \frac{4x + 5}{4x^2 + 4x + 1}
\]

The coefficient of $\frac{1}{x}$ in $r$ is the leading coefficient in $R$ minus the leading coefficient in $t$ which gives $\frac{\frac{1}{4}}{4} = 1$. Therefore $b = 1 - 0 = 1$. This results in

\[
\left(\sqrt{r}\right)_\infty = 1
\]

\[
\alpha_\infty^+ = \frac{1}{2} \left( \frac{b}{a_v} - v \right) = \frac{1}{2} \left( \frac{1}{1} - 0 \right) = \frac{1}{2}
\]

\[
\alpha_\infty^- = \frac{1}{2} \left( -\frac{b}{a_v} - v \right) = \frac{1}{2} \left( -\frac{1}{1} - 0 \right) = -\frac{1}{2}
\]

The above completes step 1 for case one. Step 2 searches for a non-negative integer $d$ using

\[
d = \alpha_\infty^+ - \sum_{c \in \mathcal{G}} \alpha_c^\pm
\]

(2)

Where the above is carried over all possible combinations resulting in the following 4 possibilities (in this example, there is only one pole, hence the sum contains only one term) and the number of possible combinations is therefore $2^2 = 4$

\[
d = \alpha_\infty^+ - \alpha_c^+ = \frac{1}{2} - \left( \frac{3}{2} \right) = -1
\]

\[
d = \alpha_\infty^+ - \alpha_c^- = \frac{1}{2} - \left( \frac{-1}{2} \right) = 1
\]

\[
d = \alpha_\infty^- - \alpha_c^+ = -\frac{1}{2} - \left( \frac{3}{2} \right) = -2
\]

\[
d = \alpha_\infty^- - \alpha_c^- = -\frac{1}{2} - \left( \frac{-1}{2} \right) = 0
\]

The above shows there are two possible $d$ values to use. $d = 1$ or $d = 0$. Each is tried until one produces a solution or all fail to do so. For each valid $d$ found an $\omega$ is found using

\[
\omega = \sum_c \left( \pm \left( \sqrt{r} \right)_c + \frac{\alpha_c^\pm}{x - c} \right) + \left( \pm \left( \sqrt{r} \right)_\infty \right)
\]
But $[\sqrt{r}]_c = 0$ in this example, hence the above simplifies to

$$\omega = \sum_c \left( \frac{\alpha^+_c}{x - c} \right) + (\pm) [\sqrt{r}]_\infty$$

Since there is one pole, then the candidate $\omega$ to try are the following

$$\omega = \frac{\alpha^+_c}{x - c} + (+1) [\sqrt{r}]_\infty$$

$$\omega = \frac{\alpha^+_c}{x - c} + (-1) [\sqrt{r}]_\infty$$

$$\omega = \frac{\alpha^-_c}{x - c} + (+1) [\sqrt{r}]_\infty$$

$$\omega = \frac{\alpha^-_c}{x - c} + (-1) [\sqrt{r}]_\infty$$

Substituting the known values found in step 1 into the above gives

$$\omega = \frac{3}{2} \frac{2x}{x + \frac{1}{2}} + (+1)(1) = \frac{6 + 2x}{2x + 3}$$

$$\omega = \frac{3}{2} \frac{2x}{x + \frac{1}{2}} + (-1)(1) = -\frac{2x}{2x + 3}$$

$$\omega = -\frac{1}{2} \frac{2x}{x + \frac{1}{2}} + (+1)(1) = \frac{2x}{2x + 1}$$

$$\omega = -\frac{1}{2} \frac{2x}{x + \frac{1}{2}} + (-1)(1) = -\frac{2(1 + x)}{2x + 1}$$

So there are two possible $d$ values to try, and for each, there are 4 possible $w(x)$, which gives 8 possible tries. This completes step 2. For each trial, step 3 is now invoked.

Starting with $d = 0$ and using $\omega = \frac{2x}{2x+1}$, and since the degree is $d = 0$ then $p(x) = 1$. This polynomial is now checked to see if it satisfies

$$p'' + 2\omega p' + (\omega' + \omega^2 - r)p = 0$$

$$d \left( \frac{2x}{2x+1} \right)^2 - \frac{4x^2 + 8x + 6}{(2x+1)^2} = 0$$

Since the left side is not identically zero, then this candidate $\omega$ has failed. Carrying out this process for the other 3 possible $\omega$ values shows that non are satisfied as well. $d = 1$ is now tried. This implies the polynomial is $p(x) = a_0 + x$. The coefficient $a_0$ needs to be determined such that $p'' + 2\omega p' + (\omega' + \omega^2 - r)p = 0$ is satisfied. Starting with $\omega = \frac{2x}{2x+1}$ gives

$$p'' + 2\omega p' + (\omega' + \omega^2 - r)p = 0$$

$$2\omega p' + (\omega' + \omega^2 - r)p = 0$$

18
Substituting \( p = a_0 + x \) and \( \omega = \frac{2x}{2x+1} \) and \( r = \frac{4x^2+8x+6}{(2x+1)^2} \) into the above and simplifying gives \( \frac{4a_0}{2x+1} = 0 \). This implies that (3) can be satisfied for \( a_0 = 0 \). Therefore the polynomial of degree one is found and given by
\[
p(x) = x
\]

Therefore the solution to \( z'' = rz \) is
\[
z = pe^{\int \omega dx} = xe^{\int \frac{2x}{2x+1} dx} = xe^{x - \frac{\ln(2x+1)}{2}} = \frac{xe^x}{\sqrt{2x+1}}
\]

Given this solution for \( z(x) \), the first basis solution of the original ode in \( y \) is found using the inverse of the original transformation used to generate the \( z \) ode which is \( z = ye^{\frac{1}{2} \int a dx} \), therefore
\[
y_1 = ze^{-\frac{1}{2} \int a dx} = xe^{x - \frac{1}{2} \int -\frac{2}{2x+1} dx} = xe^x
\]

The second basis solution is found using reduction of order
\[
y_2 = y_1 \int \frac{e^{\int -a \, dx}}{y_1^2} \, dx = xe^x \int \frac{e^{\int -\frac{2}{2x+1} \, dx}}{(xe^x)^2} \, dx = -e^{-x}
\]

Therefore the general solution to the original ode \((2x+1)y'' - 2y' - (2x+3)y = 0\) is
\[
y(x) = c_1y_1 + x_2y_2 = c_1xe^x - c_2e^{-x}
\]

This completes the solution.
3.1.2 Example 2

Given the ode
\[ x^2(x^2 - 2x + 1)y'' - x(3 + x)y' + (4 + x)y = 0 \]

Converting it \( y'' + ay' + by = 0 \) gives
\[ y'' - \frac{x(3 + x)}{x^2(x^2 - 2x + 1)}y' + \frac{4 + x}{x^2(x^2 - 2x + 1)}y = 0 \]

Where \( a = -\frac{x(3 + x)}{x^2(x^2 - 2x + 1)}, b = \frac{4 + x}{x^2(x^2 - 2x + 1)}. \) Applying the transformation \( z = ye^{\frac{1}{2}\int a\,dx} \) gives
\[ z'' = rz \]
where \( r = \frac{s}{t} = \frac{7x^2 + 10x - 1}{4x^2(x - 1)^4} \)

There is one pole at \( x = 0 \) of order 2 and pole at \( x = 1 \) of order 4, hence \( \Gamma = \{0, 1\} \), and \( \theta(\infty) = \deg(t) - \deg(s) = 6 - 2 = 4. \) Table 1 shows that the necessary conditions for case one and two are both satisfied. For the pole at 0 and since its order is 2 then
\[ \left[ \sqrt{r} \right]_0 = 0 \]
\[ a_0^+ = \frac{1}{2} + \frac{1}{2}\sqrt{1 + 4b} \]
\[ a_0^- = \frac{1}{2} - \frac{1}{2}\sqrt{1 + 4b} \]

Where \( b \) is the coefficient of \( \frac{1}{(x-0)^2} = \frac{1}{x^2} \) in the partial fraction decomposition of \( r \) which is
\[ r = \frac{4}{(x-1)^4} - \frac{2}{(x-1)^3} - \frac{1}{4x^2} - \frac{3}{2(x-1)} + \frac{3}{2x} + \frac{7}{4(x-1)^2} \]

The above shows that \( b = -\frac{1}{4}. \) Equation (1) becomes
\[ \left[ \sqrt{r} \right]_0 = 0 \]
\[ a_0^+ = \frac{1}{2} + \frac{1}{2}\sqrt{1 - \frac{1}{4}} = \frac{1}{2} \]
\[ a_0^- = \frac{1}{2} - \frac{1}{2}\sqrt{1 - \frac{1}{4}} = \frac{1}{2} \]

For the second pole at \( x = 1, \) since its order is 4, then \( 2v = 4 \) or \( v = 2. \) Therefore the corresponding \( \left[ \sqrt{r} \right]_1 \) is the sum of all terms involving \( \frac{1}{(x-1)^i} \) for \( 2 \leq i \leq v \) or \( 2 \leq i \leq 2 \) in the Laurent series expansion of \( \sqrt{r} \) (not \( r \)) around this pole. This results in
\[ \left[ \sqrt{r} \right]_1 = \sum_{i=2}^{2} \frac{a_i}{(x-1)^i} \]
\[ = \frac{a_2}{(x-1)^2} \]
\( a_2 \) is found using

\[
a_2 = \lim_{x \to 1} (x-1)^2 \sqrt{r}
\]

\[
= \lim_{x \to 1} (x-1)^2 \sqrt{\frac{7x^2 + 10x - 1}{4x^2(x-1)^4}}
\]

\[
= 2
\]

Therefore

\[
[\sqrt{r}]_1 = \frac{2}{(x-1)^2}
\]

\[
a_1^+ = \frac{1}{2} \left( \frac{b}{a} + v \right) = \frac{1}{2} \left( \frac{-2}{2} + 2 \right) = \frac{1}{2}
\]

\[
a_1^- = \frac{1}{2} \left( \frac{-b}{a} + v \right) = \frac{1}{2} \left( \frac{-3}{2} + 2 \right) = \frac{3}{2}
\]

(4)

What remains is to determine \( b \). This is the coefficient of \( \frac{1}{(x-1)^2} \) in the partial fraction decomposition of \( r \) which from (2) is \(-2\) minus the coefficient of same term in \([\sqrt{r}]_1\) which from (3) is zero. Therefore \( b = -2 - 0 = -2 \). (4) now becomes

\[
[\sqrt{r}]_1 = \frac{2}{(x-1)^2}
\]

\[
a_1^+ = \frac{1}{2} \left( \frac{b}{a} + v \right) = \frac{1}{2} \left( \frac{-2}{2} + 2 \right) = \frac{1}{2}
\]

\[
a_1^- = \frac{1}{2} \left( \frac{-b}{a} + v \right) = \frac{1}{2} \left( \frac{-3}{2} + 2 \right) = \frac{3}{2}
\]

(5)

The above completes finding \([\sqrt{r}]_c, a_1^+, a_1^-\) for all poles in the set \( \Gamma \).

Since the order of \( r \) at \( \infty \) is \( 4 > 2 \) then

\[
[\sqrt{r}]_\infty = 0
\]

\[
a_\infty^+ = 0
\]

\[
a_\infty^- = 1
\]

This completes the first step of the solution. The following tables summarizes the findings so far

| pole \( c \) location | pole order | \([\sqrt{r}]_c\) | \( a_c^+ \) | \( a_c^- \) |
|-----------------------|------------|-----------------|---------|---------|
| 0                     | 2          | 0               | \( \frac{1}{2} \) | \( \frac{1}{2} \) |
| 1                     | 4          | \( \frac{2}{(x-1)^2} \) | \( \frac{1}{2} \) | \( \frac{3}{2} \) |

Table 2: First step, case one. \( \Gamma \) set information

| Order of \( r \) at \( \infty \) | \([\sqrt{r}]_\infty\) | \( a_\infty^+ \) | \( a_\infty^- \) |
|-------------------------------|----------------|---------|---------|
| 4                             | 0              | 0       | 1       |

Table 3: First step, case one. \( \theta(\infty) \) information
This completes step 1 for case one. Step 2 searches for non-negative integer \(d\) using

\[
d = \alpha_\infty^\pm - \sum_{c \in \Gamma} \alpha_c^\pm
\]

Where the above is carried over all possible combinations resulting in the following 8 possibilities (in this example, there are two poles, hence the sum contains two term) and the number of possible combinations is therefore \(2^3 = 8\)

\[
d = \alpha_\infty^+ - (\alpha_0^+ + \alpha_1^+) = 0 - \left(\frac{1}{2} + \frac{1}{2}\right) = -1
\]

\[
d = \alpha_\infty^+ - (\alpha_0^- + \alpha_1^+) = 0 - \left(\frac{1}{2} + \frac{3}{2}\right) = -2
\]

\[
d = \alpha_\infty^- - (\alpha_0^+ + \alpha_1^+) = 0 - \left(\frac{1}{2} + \frac{1}{2}\right) = -1
\]

\[
d = \alpha_\infty^+ - (\alpha_0^- + \alpha_1^-) = 0 - \left(\frac{1}{2} + \frac{3}{2}\right) = -2
\]

\[
d = \alpha_\infty^- - (\alpha_0^+ + \alpha_1^-) = 1 - \left(\frac{1}{2} + \frac{1}{2}\right) = 0
\]

\[
d = \alpha_\infty^- - (\alpha_0^- + \alpha_1^-) = 1 - \left(\frac{1}{2} + \frac{3}{2}\right) = -1
\]

\[
d = \alpha_\infty^- - (\alpha_0^- + \alpha_1^-) = 1 - \left(\frac{1}{2} + \frac{3}{2}\right) = -1
\]

There is only one possible \(d = 0\) values to use. Candidate \(\omega\) are now found using

\[
\omega = \sum_c \left( (\pm) [\sqrt{r}]_c + \frac{\alpha_c^\pm}{x - c} \right) + (\pm) [\sqrt{r}]_\infty
\]
Which gives

\[
\omega_1 = \left( + \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^+}{x} + \left( + \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^+}{x-1} + \left( + \right) \left[ \sqrt{r} \right]_\infty
\]

\[
\omega_2 = \left( + \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^+}{x} + \left( - \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^-}{x-1} + \left( + \right) \left[ \sqrt{r} \right]_\infty
\]

\[
\omega_3 = \left( - \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^-}{x} + \left( + \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^+}{x-1} + \left( + \right) \left[ \sqrt{r} \right]_\infty
\]

\[
\omega_4 = \left( - \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^-}{x} + \left( - \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^-}{x-1} + \left( + \right) \left[ \sqrt{r} \right]_\infty
\]

\[
\omega_5 = \left( + \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^+}{x} + \left( - \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^-}{x-1} + \left( - \right) \left[ \sqrt{r} \right]_\infty
\]

\[
\omega_6 = \left( + \right) \left[ \sqrt{r} \right]_0 + \frac{a_0^+}{x} + \left( - \right) \left[ \sqrt{r} \right]_1 + \frac{a_1^-}{x-1} + \left( - \right) \left[ \sqrt{r} \right]_\infty
\]

Substituting values found in step 1 into the above gives

\[
\omega_1 = \left( \frac{1}{2} \right) + \left( + \right) \frac{2}{(x-1)^2} + \frac{1}{2} \frac{1}{x-1} = \frac{2x^2 + x + 1}{2x(x-1)^2}
\]

\[
\omega_2 = \left( \frac{1}{2} \right) + \left( - \right) \frac{2}{(x-1)^2} + \frac{3}{2} \frac{1}{x-1} = \frac{4x^2 - 9x + 1}{2x(x-1)^2}
\]

\[
\omega_3 = \left( \frac{1}{2} \right) + \left( + \right) \frac{2}{(x-1)^2} + \frac{1}{2} \frac{1}{x-1} = \frac{2x^2 + x + 1}{2x(x-1)^2}
\]

\[
\omega_4 = \left( \frac{1}{2} \right) + \left( - \right) \frac{2}{(x-1)^2} + \frac{3}{2} \frac{1}{x-1} = \frac{4x^2 - 9x + 1}{2x(x-1)^2}
\]

\[
\omega_5 = \left( \frac{1}{2} \right) + \left( + \right) \frac{2}{(x-1)^2} + \frac{1}{2} \frac{1}{x-1} = \frac{2x^2 + x + 1}{2x(x-1)^2}
\]

Which shows there are only two different \( \omega \) to try, these are \( \omega_1, \omega_2 \). This complete step 2.

For each trial, step 3 is now invoked. Starting with \( d = 0 \) and \( \omega = \omega_1 = \frac{2x^2 + x + 1}{2x(x-1)^2} \). Since the
degree $d = 0$ then $p(x) = 1$. This polynomial needs to satisfy

$$p'' + 2\omega p' + (\omega' + \omega^2 - r)p = 0$$  \hspace{1cm} (6)

$$\frac{d}{dx} \left( \frac{2x^2 + x + 1}{2x(x-1)^2} \right) + \left( \frac{2x^2 + x + 1}{2x(x-1)^2} \right)^2 - \frac{7x^2 + 10x - 1}{4x^2(x-1)^4} = 0$$

Because the equation is satisfied, the polynomial $p(x) = 1$ can be used. The solution to $z'' = rz$ is now found from

$$z = pe^{\int \omega dx}$$

$$= e^{\int \frac{2x^2 + x + 1}{2x(x-1)^2} dx}$$

$$= e^{\int \frac{\ln(x-1) - \log(x)}{x-1} + \int \frac{\ln(x)}{x} dx}$$

$$= \sqrt{x - 1} \sqrt{x} e^{-\frac{2}{x-1}}$$

Given this solution for $z(x)$, the first basis solution of the original ode in $y$ is found using the inverse of the original transformation used to generate the $z$ ode which is $z = ye^{\int a dx}$, therefore

$$y_1 = ze^{-\frac{1}{2} \int a dx}$$

$$= \sqrt{x - 1} \sqrt{x} e^{-\frac{2}{x-1}} e^{\int \frac{x^3 + x}{x(x-1)(x+1)}} dx$$

Which simplifies to

$$y_1 = \frac{x^2}{x - 1} e^{-\frac{4}{x-1}}$$

The second solution $y_2$ to the original ode is found using reduction of order as was done in the first example.

### 3.1.3 Example 3

This ode is a standard second order representing the oscillating harmonics ode with constant coefficients and does not require Kovacic algorithm to solve it as it can be readily solved using standard method by finding the roots of the characteristic equation. It is included here in order to illustrate the Kovacic algorithm.

$$y'' + y' + y = 0$$

$$Ay'' + By' + Cy = 0$$

Converting it to $z'' = rz$ as shown before gives

$$z'' = \frac{s}{t} z$$

$$= \frac{-3}{4} z$$
Hence \( r = -\frac{3}{4} \). There are no poles therefore \( \Gamma = \emptyset \), and \( \mathcal{O}(\infty) = \deg(t) - \deg(s) = 0 \). Table \ref{tab:conditions} shows that the necessary conditions for case one are only satisfied. Since the set \( \Gamma \) is empty, then only the quantities related to \( \mathcal{O}(\infty) \) need to be calculated. The order of \( r \) at \( \infty \) is \( O_r(\infty) = 0 \) therefore \( v = 0 \). \( r \) has no \( x \) in it, hence the Laurent series of \( \sqrt{r} \) at \( \infty \) is itself

\[
\sqrt{r} = \frac{i\sqrt{3}}{2}
\]

Therefore

\[
a = \frac{i\sqrt{3}}{2}
\]

And since \( r \) is constant then \( b = 0 \). Hence

\[
[\sqrt{r}]_\infty = \frac{i\sqrt{3}}{2}
\]

\[
a_\infty^+ = \frac{1}{2} \left( \frac{b}{a} - v \right) = \frac{1}{2} \left( \frac{0}{\frac{i\sqrt{3}}{2}} - 0 \right) = 0
\]

\[
a_\infty^- = \frac{1}{2} \left( -\frac{b}{a} - v \right) = \frac{1}{2} \left( -\frac{0}{\frac{i\sqrt{3}}{2}} - 0 \right) = 0
\]

This completes step 1 for case one. Step 2 searches for non-negative integer \( d \) using

\[
d = a_\infty^+ - \sum_{c \in \Gamma} a_c^\pm
\]

Since there are no poles then

\[
d = a_\infty^-
\]

\[
d = 0
\]

Since \( d \) is non-negative integer integer it can be used to find \( \omega \) using

\[
\omega = \sum_{c \in \Gamma} \left( s(c)[\sqrt{r}]_c + \frac{a^{s(c)}_c}{x - c} \right) + s(\infty)[\sqrt{r}]_\infty
\]

The above reduces to

\[
\omega = (-)[\sqrt{r}]_\infty
\]

\[
= 0 + (-) \left( \frac{i\sqrt{3}}{2} \right)
\]

\[
= -\frac{i\sqrt{3}}{2}
\]
Now that $\omega$ is determined, the next step is to find a corresponding minimal polynomial $p(x)$ of degree $d = 0$ to solve the ode. The polynomial $p(x)$ needs to satisfy the equation

$$p'' + 2\omega p' + (\omega' + \omega^2 - r)p = 0 \quad (1)$$

Since $d = 0$ then let $p(x) = 1$. Substituting this in the above gives

$$(0) + 2\left(-\frac{i\sqrt{3}}{2}\right)(0) + \left(0 + \left(-\frac{i\sqrt{3}}{2}\right)^2 - \left(-\frac{3}{4}\right)\right) = 0$$

The equation is satisfied. Therefore the first solution to the ode $z'' = rz$ is

$$z(x) = pe^{\int \omega dx} = e^{\int -\frac{i\sqrt{3}}{2} dx} = e^{-\frac{i\sqrt{3}x}{2}}$$

The first solution to the original ode in $y$ is now found from (using $A = 1, B = 1$)

$$y_1 = ye^{\int \frac{-B}{2} dx} = ze^{-\frac{1}{2} dx} = ze^\frac{1}{2} = e^{-\frac{i\sqrt{3}x}{2}} \left(e^{-\frac{x}{2}}\right) = e^{-(1+i\sqrt{3})\frac{x}{2}}$$

The second solution $y_2$ to the original ode is found using reduction of order

$$y_2 = y_1 \int \frac{e^{\int \frac{-B}{2} dx}}{y_1^2} \frac{dy_1}{dx}$$

$$= y_1 \int \frac{e^{-\frac{x}{2}}}{(y_1)^2} dy_1 = y_1 \int \frac{e^{-x}}{(y_1)^2} dy_1$$

$$= \left(e^{-\frac{x(1+i\sqrt{3})}{2}}\right)\left(-\frac{i\sqrt{3}e^{i\sqrt{3}x}}{3}\right)$$

$$= -\frac{1}{3} e^{\frac{x(1+i\sqrt{3})}{2}} \sqrt{3}$$

Therefore the general solution is

$$y = c_1y_1 + c_2y_2$$

$$= c_1 \left(e^{-\frac{x(1+i\sqrt{3})}{2}}\right) + c_2 \left(-\frac{ie^{-\frac{x}{2}}\sqrt{3}}{3}\right)$$
Using Euler's formula the above can be simplified to the standard looking solution

\[ y(x) = e^{-\frac{x}{2}} \left( C_1 \sin \left( \frac{\sqrt{3} x}{2} \right) + C_2 \cos \left( \frac{\sqrt{3} x}{2} \right) \right) \]

### 3.2 case two

#### 3.2.1 Example 1

Given the ode

\[ 2x^2 y'' - xy' + (1 + x)y = 0 \]

Converting it \( y'' + ay' + by = 0 \) gives

\[ y'' - \frac{1}{2x} y' + \frac{1 + x}{2x^2} y = 0 \]

Where \( a = -\frac{1}{2x}, b = \frac{1 + x}{2x^2}. \) Applying the transformation \( z = ye^{\frac{1}{2} \int a \, dx} \) gives \( z'' = rz \) and \( r = \frac{1}{4}a^2 + \frac{1}{2}a' - b. \) Therefore

\[ r = \frac{s}{t} = -3 - 8x \]

\[ = \frac{16x^2}{16x^2} \]

There is one pole at \( x = 0 \) of order 2 and \( \phi(\infty) = \deg(t) - \deg(s) = 2 - 1 = 1. \) Table 1 shows that necessary conditions for only case two are satisfied. Since pole is order 2, the set \( E_0 = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\} \) where \( b \) is the coefficient of \( \frac{1}{(x)^2} \) in the partial fraction decomposition of \( r \) given by

\[ r = -\frac{3}{16x^2} - \frac{1}{2x} \]

Therefore \( b = -\frac{3}{16}. \) Hence

\[ E_0 = \left\{ 2, 2 + 2\sqrt{1 - 4 \cdot \frac{3}{16}}, 2 - 2\sqrt{1 - 4 \cdot \frac{3}{16}} \right\} = \{2, 3, 1\} \]

Since \( \phi(\infty) = 1 < 2 \) then \( E_\infty = \phi(\infty) = \{1\}. \) This completes step 1 of case two. Step 2 is used to determine a non-negative integer \( d. \) Using \( e_\infty = 1 \) gives

\[ d = \frac{1}{2} \left( e_\infty - \sum_{c \in \Gamma} e_c \right) \]
There is only one pole, so the sum contains only one term. There are 3 possible combinations to try, using either $e_0 = 2$, $e_0 = 3$ or $e_0 = 1$. Therefore

$$d = \frac{1}{2} (e_\infty - (e_0)) = \frac{1}{2} (1 - 2) = -\frac{1}{2}$$

$$= \frac{1}{2} (e_\infty - (e_0)) = \frac{1}{2} (1 - 3) = -1$$

$$= \frac{1}{2} (e_\infty - (e_0)) = \frac{1}{2} (1 - 1) = 0$$

The above shows that only the family $\{e_\infty = 1, e_0 = 1\}$ generated non-negative $d = 0$. $\theta$ is now found. In the following sum, only $e_c$ retained from the above are used. In this example, this will be $e_0 = 1$ since it is the member of $E_0$ which generated non-negative integer. If there were more than one $e_i$ found, then each would be tried at time.

$$\theta = \frac{1}{2} \sum_{c \in \Gamma} \frac{e_c}{x - c}$$

$$= \frac{1}{2} \left( \frac{e_0}{x - 0} \right)$$

$$= \frac{1}{2x}$$

This completes step 2. Step 3 finds polynomial $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + x^d$ of degree $d$. Since $d = 0$ then $p(x) = 1$. This polynomial has to satisfy the following

$$p''' + 3\theta p'' (3\theta^2 + 3\theta' - 4r) p' + (\theta'' + 3\theta \theta' + \theta^3 - 4r \theta - 2r') p = 0$$

Substituting $p = 1, \theta = \frac{1}{2x}$ into the above and simplifying gives

$$0 = 0$$

Since $p(x) = 1$ is verified, then

$$\phi = \theta + \frac{p'}{p}$$

$$= \frac{1}{2x}$$

Next, $\omega$ solution is found using

$$\omega^2 - \phi \omega + \left( \frac{1}{2} \phi' + \frac{1}{2} \phi^2 - r \right) = 0$$

Substituting the values for $\phi$ and $r$ into the above gives

$$\omega^2 - \frac{\omega}{2x} + \frac{1 + 8x}{16x^2} = 0$$

Solving for $\omega$ gives two roots, either one can be used. Using

$$\omega = \frac{1 + 2\sqrt{2} \sqrt{-x}}{4x}$$
Therefore the first solution to the ode $z'' = rz$ is

$$z(x) = e^{\int \frac{1 + 2\sqrt{1 + 4b}}{4x} dx}$$

$$= x^{\frac{1}{4}} e^{\sqrt{2\sqrt{1-x}}}$$

The first solution to the original ode in $y$ is now found from

$$y_1 = z e^{\int -\frac{1}{2} a dx}$$

$$= z e^{-\int \frac{1}{2} \frac{1}{2x} dx}$$

$$= z e^{\ln(x)}$$

$$= \left(x^{\frac{1}{4}} e^{\sqrt{2\sqrt{1-x}}} \right) x^{\frac{1}{4}}$$

$$= \sqrt{x} e^{\frac{1}{2}\sqrt{1-x}}$$

The second solution $y_2$ to the original ode can be found using reduction of order.

### 3.2.2 Example 2

This is an ode in which the necessary conditions for all three cases are satisfied, but solved using case two to illustrate the algorithm.

$$(1 - x)x^2 y'' + (5x - 4)xy' + (6 - 9x)y = 0$$

Converting it $y'' + ay' + by = 0$ gives

$$y'' + \frac{5x - 4}{(1 - x)x} y' + \frac{6 - 9x}{(1 - x)x^2} y = 0$$

Where $a = \frac{5x - 4}{(1 - x)x}$, $b = \frac{6 - 9x}{(1 - x)x^2}$. Applying the transformation $z = ye^{\frac{1}{2}\int a dx}$ gives $z'' = rz$ where $r = \frac{1}{4}a^2 + \frac{1}{2}a' - b$. Therefore

$$r = \frac{s}{t}$$

$$= \frac{4 - x}{4x(x - 1)^2}$$

There is a pole at $x = 0$ of order 1 and a pole at $x = 1$ of order 2. Since there is no odd order pole larger than 2 and the order at infinity is 2 then the necessary conditions for case one are satisfied. Since there is a pole of order 2 then the necessary conditions for case two are also satisfied. Since pole order is not larger than 2 and the order at infinity is 2 then the necessary conditions for case three are also satisfied. Any one of the three cases algorithm could be used to solve this, but here case two will be used for illustration.

The pole of order 1 at $x = 0$ gives $E_0 = \{4\}$ and the pole of order 2 at $x = 1$ gives $E_1 = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\}$ where $b$ is the coefficient of $\frac{1}{(x-1)^2}$ in the partial fraction decomposition of $r$. The partial fractions decomposition of $r$ is

$$r = \frac{3}{4(x - 1)^2} - \frac{1}{x - 1} + \frac{1}{x}$$
The above shows that $b = \frac{3}{4}$, therefore $E_1 = \{-2, 2, 6\}$.

$\mathcal{O}(\infty) = 2$ therefore $E_\infty = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\}$ where $b = \frac{\text{lcoeff}(s)}{\text{lcoeff}(t)}$ where $r = \frac{s}{t}$. This gives $b = -\frac{1}{4}$, hence

$$E_\infty = \{2, 2 + 2\sqrt{1 + 4b}, 2 - 2\sqrt{1 + 4b}\}$$

$$= \left\{2, 2 + 2\sqrt{1 - \frac{1}{4}}, 2 - 2\sqrt{1 - \frac{1}{4}}\right\}$$

$$= \{2, 2, 2\}$$

$$= \{2\}$$

The following table summarizes step 1 results.

| pole c location | pole order | $E_c$       |
|-----------------|------------|-------------|
| 0               | 1          | \{4\}       |
| 1               | 2          | \{-2, 2, 6\}|

Table 4: First step, case two. $E_c$ set information

| Order of $r$ at $\infty$ | $E_\infty$ |
|---------------------------|------------|
| 2                         | \{2\}      |

Table 5: First step, case two. $\mathcal{O}(\infty)$ information

The above completes step 1 for case two. Step 2 searches for a non-negative integer $d$ using

$$d = \frac{1}{2} \left( e_\infty - \sum_{c \in \Gamma} e_c \right)$$

Where in the above $e_c \in E_c$, $e_\infty \in E_\infty$ were found in the first step. The following are the possible combinations to use

$$d = \frac{1}{2} (2 - (4 - 2)) = 0$$

$$= \frac{1}{2} (2 - (4 + 2)) = -2$$

$$= \frac{1}{2} (2 - (4 + 6)) = -4$$

The above shows that $E_c = \{e_0 = 4, e_1 = -2\}$ are the family of values to use and all other values are discarded.
The following rational function $\theta$ is determined using:

$$
\theta = \frac{1}{2} \sum_{c \in \Gamma} \frac{e_c}{x - c}
$$

$$
= \frac{1}{2} \left( \frac{4}{(x-0)} + \frac{-2}{(x-1)} \right)
$$

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

The algorithm now searches for a monic polynomial $p(x)$ of degree $d = 0$ such that

$$
p''' + 3 \theta p'' + (3\theta^2 + 3\theta' - 4r) p' + (\theta'' + 3\theta\theta' + \theta^3 - 4r\theta - 2r') p = 0
$$

Since $d = 0$, then $p(x) = 1$. Substituting the values found in step 2 in the above equation and simplifying gives

$$0 = 0$$

Hence $p(x) = 1$ can be used. Let

$$\phi = \theta + \frac{p'}{p}
$$

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

And $\omega$ be the solution of

$$
\omega^2 - \phi \omega + \left( \frac{1}{2} \phi' + \frac{1}{2} \phi^2 - r \right) = 0
$$

Substituting the values for $\phi$ and $r$ into the above equation gives

$$
\omega^2 - \left( \frac{2}{x} - \frac{1}{x - 1} \right) \omega + \frac{(x-2)^2}{4(x-1)^2} x^2 = 0
$$

Solving for $\omega$ gives two roots, either one can be used. Using

$$\omega = \frac{x-2}{2(x-1)x}
$$

Therefore to $z'' = rz$ is

$$z = e^{\int \omega \, dx}
$$

$$= e^{\int \frac{x-2}{2(x-1)x} \, dx}
$$

$$= \frac{x}{\sqrt{x-1}}
$$

31
The first solution to the original ode in $y$ is now found from
\[
y_1 = ze^{-\int \frac{1}{a} \frac{4x}{x^2 + 1} dx} = ze^{-\int \frac{5x^2 - 4x}{2x^2 + 1} dx} = ze^{2\ln(x) + \frac{\ln(x-1)}{2}} = z\left(x^2 \sqrt{x-1}\right) = \frac{x}{\sqrt{x-1}}(x^2 \sqrt{x-1}) = x^3
\]

The second solution $y_2$ to the original ode is found using reduction of order.

### 3.3 case three

#### 3.3.1 Example 1

This is the same ode used in second example above for case two as the necessary conditions for case three are also satisfied.

\[
(1-x)x^2 y'' + (5x-4)xy' + (6-9x)y = 0
\]

As shown earlier, this ode is transformed to $z'' = rz$ where
\[
r = \frac{s}{t} = \frac{4-x}{4x(x-1)^2}
\]

There is a pole at $x = 0$ of order 1 and a pole at $x = 1$ of order 2. For the pole of order 1 at $x = 0$, $E_0 = \{12\}$. For the pole of order 2 at $x = 1$

\[
E_1 = \left\{6 + \frac{12k}{n} \sqrt{1 + 4b}\right\} \quad \text{for} \quad k = \frac{n}{2} \cdots \frac{n}{2} \quad (1)
\]

Where $k$ is incremented by 1 each time, and $n$ is any of $\{4, 6, 12\}$ and $b$ is the coefficient of $\frac{1}{(x-1)^2}$ in the partial fraction decomposition of $r$ which is
\[
r = \frac{3}{4(x-1)^2} - \frac{1}{x-1} + \frac{1}{x}
\]

The above shows that $b = \frac{3}{4}$. Starting with $n = 4$ (if this $n$ produces no solution then $n = 6, 12$ will be tried also). Equation (1) now becomes
\[
E_1 = \left\{6 + \frac{12k}{4} \sqrt{1 + 4\left(\frac{3}{4}\right)}\right\} \quad \text{for} \quad k = -2 \cdots 2
\]
Which simplifies to

$$E_1 = \{6 + 6k\} \quad \text{for} \quad k = -2 \cdots 2$$

$$= \{-6, 0, 6, 12, 18\}$$

$E_\infty$ is found using (1) but with different $b$. In this case $b$ is given by $b = \frac{\text{lcoeft}(s)}{\text{lcoeft}(t)}$ where $r = \frac{2}{x}$. $\text{lcoeft}(s)$ is the leading coefficient of $s$ and $\text{lcoeft}(t)$ is the leading coefficient of $t$. Since $r = \frac{-x + 4}{4x^4 - 8x^2 + 4x}$ then $b = -\frac{1}{4}$. Equation (1) becomes

$$E_\infty = \left\{6 + \frac{12k}{4} \sqrt{1 - 4 \left(\frac{1}{4}\right)}\right\} \quad \text{for} \quad k = -2 \cdots 2$$

This simplifies to

$$E_\infty = \{6\}$$

The following table summarizes step 1 results using $n = 4$.

| pole $c$ location | pole order | $E_c$     |
|-------------------|------------|-----------|
| 0                 | 1          | $\{12\}$ |
| 1                 | 2          | $\{-6, 0, 6, 12, 18\}$ |

Table 6: First step, case three using $n = 4$. $E_c$ set information

| Order of $r$ at $\infty$ | $E_\infty$ |
|---------------------------|------------|
| 2                         | $\{6\}$   |

Table 7: First step, case three using $n = 4$. $\mathcal{O}(\infty)$ information

The next step is to determine a non negative integer $d$ using

$$d = \frac{n}{12} \left( e_\infty - \sum_{c \in \Gamma} e_c \right)$$

Where in the above $e_c$ is a distinct element from each corresponding $E_c$. This means all possible tuples $\{e_{c_1}, e_{c_2}, \ldots, e_{c_n}\}$ are tried in the sum above, where $e_{c_i}$ is one element of each $E_c$ found earlier.
This results in the following values for $d$ using $n = 4$ and $e_\infty = 6$.

\[
d = \frac{1}{3} (6 - (12 - 6)) = 0 \\
= \frac{1}{3} (6 - (12 + 0)) = -2 \\
= \frac{1}{3} (6 - (12 + 6)) = -4 \\
= \frac{1}{3} (6 - (12 + 12)) = -6 \\
= \frac{1}{3} (6 - (12 + 18)) = -8
\]

Therefore only the first case using $e_0 = 12, e_1 = -6$ generated non-negative integer $d$. The following rational function is now formed

\[
\theta = \frac{n}{12} \sum_{c \in \Gamma} \frac{e_c}{x - c} \\
= \frac{4}{12} \left( \frac{12}{(x - 0)} + \frac{-6}{(x - 1)} \right) \\
= \frac{2x - 4}{(x - 1)x}
\]

And

\[
S = \prod_{c \in \Gamma} (x - c) \\
= (x - 0)(x - 1) \\
= x(x - 1)
\]

This completes the step 2 of the algorithm.

Since the degree $d = 0$, then $p(x) = 1$. Now $P_i(x)$ polynomials are generated using

\[
P_n = -p(x) \\
P_{i-1} = -Sp_i' + ((n - i)S' - S\theta)P_i - (n - 1)(i + 1)S^2rP_{i+1} \\
i = n, n - 1, \ldots, 0
\]

These generate the following set

\[
P_4 = -1 \\
P_3 = 2x - 4 \\
P_2 = -3(x - 2)^2 \\
P_1 = 3(x - 2)^3 \\
P_0 = -\frac{3(x - 2)^4}{2} \\
P_{-1} = 0
\]
There is nothing to solve for from the last equation $P_{-1} = 0$ as $p(x) = 1$ is already known because the degree $d$ was zero and hence there are no coefficients $a_i$ to solve for.

Next $\omega$ is determined as the solution to the following equation using $n = 4$.

$$\sum_{i=0}^{n} S_i \frac{P_i}{(n-i)!} \omega^i = 0$$

$$\frac{P_0}{4!} + \frac{SP_1}{3!} \omega + \frac{S^2P_2}{2!} \omega^2 + \frac{S^3P_3}{1!} \omega^3 + \frac{S^4P_4}{0!} \omega^4 = 0$$

$$-\frac{1}{16} (2\omega x^2 - 2x\omega - x + 2)^4 = 0$$

Solving the above and using any one of the roots results in

$$\omega = \frac{1}{2x(x-1)} (x-2)$$

The above $\omega$ is used to find a solution to $z'' = rz$ from

$$z = e^{\int \omega \, dx}$$

$$= e^{\int \frac{x-2}{2x(x-1)} \, dx}$$

$$= \frac{x}{\sqrt{x-1}}$$

Therefore one solution to the original ode in $y$ is

$$y = ze^{\int -\frac{1}{2} \omega \, dx}$$

$$= ze^{-\int \frac{1}{2} \frac{x^2-4x}{x^2} \, dx}$$

$$= ze^{2\ln(x) + \ln(x-1)}$$

$$= \left( \frac{x}{\sqrt{x-1}} \right) \left( x^2 \sqrt{x-1} \right)$$

The second solution to the original ode is found using reduction of order. This completes the solution using case 3 for degree $n = 4$ of $\omega$.

### 3.3.2 Example 2

The ode is

$$x^2(1+x)y'' + x(2x+1)y'' - (4+6x)y = 0$$

Converting it $y'' + ay' + by = 0$ gives

$$y'' + \frac{2x+1}{x(1+x)} y' - \frac{4+6x}{x^2(1+x)} y = 0$$
Where \( a = \frac{2x+1}{x(1+x)} \), \( b = -\frac{4+6x}{x^2(1+x)} \). Applying the transformation \( z = ye^{\frac{1}{2}\int a\,dx} \) results in \( z'' = rz \) where \( r = \frac{s}{t} = \frac{24x^2 + 40x + 15}{4(x(x+1))^2} \).

There is a pole at \( x = 0 \) of order 2 and a pole at \( x = -1 \) of order 2. Since there is no odd order pole larger than 2 and the order at \( \infty \) is 2 then the necessary conditions for case one are satisfied. Since there is a pole of order 2 then necessary conditions for case two are also satisfied. Since pole order is not larger than 2 and the order at \( \infty \) is 2 then the necessary conditions for case three are also satisfied. This is now solved as case three for illustration.

Starting with \( n = 4 \), and for the pole of order 2 at \( x = -1 \)

\[
E_{-1} = \left\{ 6 + \frac{12k}{n} \sqrt{1 + 4b} \right\} \quad \text{for} \quad k = -\frac{n}{2} \cdots \frac{n}{2}
\]

This simplifies to

\[
E_{-1} = \left\{ 6 + 3k\sqrt{1 + 4b} \right\} \quad \text{for} \quad k = -2 \cdots 2 \tag{1}
\]

\( b \) is the coefficient of \( \frac{1}{(1+x)^2} \) in the partial fractions decomposition of \( r \) given by

\[
r = -\frac{1}{4(1+x)^2} + \frac{15}{4x^2} - \frac{5}{2(1+x)} + \frac{5}{2x}
\]

The above shows that \( b = \frac{1}{4} \). Equation (1) becomes

\[
E_{-1} = \left\{ 6 + 3k\sqrt{1 - 4\left(\frac{1}{4}\right)} \right\} \quad \text{for} \quad k = -2 \cdots 2
\]

\[= \{6\}\]

For the pole at \( x = 0 \) of order 2, \( b \) is the coefficient of \( \frac{1}{x^2} \) in the above partial fractions decomposition of \( r \). This shows that \( b = \frac{15}{4} \). Hence

\[
E_0 = \left\{ 6 + 3k\sqrt{1 + 4\left(\frac{15}{4}\right)} \right\} \quad \text{for} \quad k = -2 \cdots 2
\]

\[= \{-18, -6, 6, 18, 30\}\]

\( E_{\infty} \) is found using equation (1) but with different \( b \). In this case \( b \) is given by \( b = \frac{\text{lcoeff}(s)}{\text{lcoeff}(t)} \) where \( r = \frac{s}{t} \). \text{lcoeff}(s) is the leading coefficient of \( s \) and \text{lcoeff}(t) is the leading coefficient of \( t \). Since \( r = \frac{24x^2 + 40x + 15}{4x^4 + 8x^3 + 4x^2} \) then \( b = 6 \). Equation (1) becomes

\[
E_{\infty} = \left\{ 6 + 3k\sqrt{1 + 4(6)} \right\} \quad \text{for} \quad k = -2 \cdots 2
\]

\[= \{-24, -9, 6, 21, 36\}\]
The following table summarizes step 1 results using \( n = 4 \).

| pole c location | pole order | \( E_c \)          |
|-----------------|------------|--------------------|
| -1              | 2          | (6)                |
| 0               | 2          | \{-18, -6, 6, 18, 30\} |

Table 8: First step, case three using \( n = 4 \). \( E_c \) set information

| Order of \( r \) at \( \infty \) | \( E_\infty \)   |
|-----------------------------------|------------------|
| 2                                 | \{-24, -9, 6, 21, 36\} |

Table 9: First step, case three using \( n = 4 \). \( \mathcal{O}(\infty) \) information

The next step is to determine a non negative integer \( d \) using

\[
d = \frac{n}{12} \left( e_\infty - \sum_{c \in \Gamma} e_c \right)
\]

Where in the above \( e_c \) is a distinct element from each corresponding \( E_c \). This means all possible tuples \( \{e_{c_1}, e_{c_2}, \ldots, e_{c_n}\} \) are tried in the sum above, where \( e_{c_i} \) is one element of each \( E_c \) found earlier.
This results in the following values for $d$ using $n = 4$.

\[
\begin{align*}
  d &= \frac{1}{3}(-24 - (6 - 18)) = -4 \\
  &= \frac{1}{3}(-24 - (6 + 6)) = -12 \\
  &= \frac{1}{3}(-24 - (6 - 6)) = -8 \\
  &= \frac{1}{3}(-24 - (6 + 18)) = -16 \\
  &= \frac{1}{3}(-24 - (6 + 30)) = -20 \\
  &= \frac{1}{3}(-9 - (6 - 18)) = 1 \\
  &= \frac{1}{3}(-9 - (6 + 6)) = -7 \\
  &= \frac{1}{3}(-9 - (6 - 6)) = -3 \\
  &= \frac{1}{3}(-9 - (6 + 18)) = -11 \\
  &= \frac{1}{3}(-9 - (6 + 30)) = -15 \\
  &= \frac{1}{3}(6 - (6 - 18)) = 6 \\
  &= \frac{1}{3}(6 - (6 + 6)) = -2 \\
  &= \frac{1}{3}(6 - (6 - 6)) = 2 \\
  &= \frac{1}{3}(6 - (6 + 18)) = -6 \\
  &= \frac{1}{3}(6 - (6 + 30)) = -10 \\
  &= \frac{1}{3}(21 - (6 - 18)) = 11 \\
  &= \frac{1}{3}(21 - (6 + 6)) = 3 \\
  &= \frac{1}{3}(21 - (6 - 6)) = 7 \\
  &= \frac{1}{3}(21 - (6 + 18)) = -1 \\
  &= \frac{1}{3}(21 - (6 + 30)) = -5 \\
  &= \frac{1}{3}(36 - (6 - 18)) = 16 \\
  &= \frac{1}{3}(36 - (6 + 6)) = 8 \\
  &= \frac{1}{3}(36 - (6 - 6)) = 12 \\
  &= \frac{1}{3}(36 - (6 + 18)) = 4 \\
  &= \frac{1}{3}(36 - (6 + 30)) = 0
\end{align*}
\]

From the above, the following families all produce non-negative $d$
$e_{\infty} = -9 \quad e_{-1} = 6 \quad e_0 = -18 \quad d = 1$

$e_{\infty} = 6 \quad e_{-1} = 6 \quad e_0 = -18 \quad d = 6$

$e_{\infty} = 6 \quad e_{-1} = 6 \quad e_0 = -6 \quad d = 2$

$e_{\infty} = 21 \quad e_{-1} = 6 \quad e_0 = -18 \quad d = 11$

$e_{\infty} = 21 \quad e_{-1} = 6 \quad e_0 = 6 \quad d = 3$

$e_{\infty} = 21 \quad e_{-1} = 6 \quad e_0 = -6 \quad d = 7$

$e_{\infty} = 36 \quad e_{-1} = 6 \quad e_0 = -18 \quad d = 16$

$e_{\infty} = 36 \quad e_{-1} = 6 \quad e_0 = 6 \quad d = 8$

$e_{\infty} = 36 \quad e_{-1} = 6 \quad e_0 = -6 \quad d = 12$

$e_{\infty} = 36 \quad e_{-1} = 6 \quad e_0 = 18 \quad d = 4$

$e_{\infty} = 36 \quad e_{-1} = 6 \quad e_0 = 30 \quad d = 0$

Starting with the smallest $d = 0$ as that is the least computationally expensive with its corresponding $e_{-1} = 6, e_0 = 30, e_\infty = 36$, the following rational function is now formed

$$\theta = \frac{1}{12} \sum_{c \in \Gamma} \frac{e_c}{x - c}$$

$$= \frac{4}{12} \left( \frac{6}{x + 1} + \frac{30}{x} \right)$$

$$= \frac{12x + 10}{x(1 + x)}$$

And

$$S = \prod_{c \in \Gamma} (x - c)$$

$$= (x + 1)x$$

This completes the step 2 of the algorithm.

Polynomial $p(x)$ is now determined. Since the degree of the polynomial is $d = 0$, then

$$p(x) = 1$$

The $P_i(x)$ polynomials are generated using

$$P_n = -p(x)$$

$$P_{i-1} = -Sp_i + ((n - i)S' - S\theta)P_i - (n - 1)(i + 1)SP_i P_{i+1} \quad i = n, n-1, \ldots, 0$$

The above results in the following set

$$P_4 = -1$$

$$P_3 = 12x + 10$$

$$P_2 = -3(6x + 5)^2$$

$$P_1 = 3(6x + 5)^3$$

$$P_0 = - \frac{3(6x + 5)^4}{2}$$

$$P_{-1} = 0$$

39
There is coefficient for $p(x)$ to solve for from the last equation $P_{-1} = 0$ as $p(x) = 1$ is already known because the degree $d$ is zero. $\omega$ is now determined as the solution to the following equation, using $n = 4$

$$\sum_{i=0}^{n} S_i \frac{P_i}{(n-i)!}\omega^i = 0$$

$$\frac{P_0}{4!} + \frac{SP_1}{3!} \omega + \frac{S^2P_2}{2!} \omega^2 + \frac{S^3P_3}{1!} \omega^3 + \frac{S^4P_4}{0!} \omega^4 = 0$$

$$-\frac{1}{16}(2\omega x^2 + 2x\omega - 6x - 5)^4 = 0$$

Solving the above and using any one of the roots gives

$$\omega = \frac{x - 2}{2x(x - 1)}$$

This $\omega$ is used to find a solution to $z'' = rz$ from

$$z = e^{\int \omega \, dx}$$

$$= e^{\int \frac{6x + 5}{2x(1+x)} \, dx}$$

$$= x^{\frac{5}{2}} \sqrt{1 + x}$$

The first solution to the original ode in $y$ is found from

$$y = ze^{\int -\frac{1}{2}a \, dx}$$

$$= ze^{\int -\frac{1}{2} \frac{2x^2 + 2x}{x^2(1+x)} \, dx}$$

$$= ze^{-\ln(x(1+x))}$$

$$= \left(x^\frac{5}{2} \sqrt{1 + x}\right) \left(\frac{1}{\sqrt{x(1+x)}}\right)$$

Which simplifies to

$$y = x^2$$

The second solution to the original ode is found using reduction of order.
4 Conclusion

Detailed description of the Kovacic algorithm with worked out examples were given. All three cases of the Kovacic algorithm were implemented using object oriented design in Maple. The software was then used to analyze over 3000 differential equations. The results showed that case one and two combined provided coverage for 99.9% of the ode's with 97.36% of the ode's solved using case one algorithm and 2.54% solved using case 2 algorithm with only 0.1% requiring case 3. Not a single ode was found that required the use of case three with $n = 6$ or $n = 12$.

One restriction found on the use of the algorithm is that it requires an ode with its coefficients being numerical and not symbolic. This is because the algorithm has to decide in step 2 if $d$ (the degree of polynomial $p(x)$) is non-negative integer or not in order to continue to step 3. If some of the ode coefficients were symbolic, it will not be able to decide on this (without additional assumptions provided). Therefore this algorithm works best with ode's having its coefficients given with numerical values.
5 Appendix

5.1 Instructions and examples using the Kovacic package

The Kovacic class is included in the file KOV.mpl and the Kovacic testsuite module is in the file kovacic_tester.mpl. These two files accompany the arXiv version of this paper.

To use these, download these two files to some directory at your computer. For example, on windows, assuming the files were downloaded to c:/my_folder/, then now start Maple and type

```
read "c:/my_folder/KOV.mpl"
read "c:/my_folder/kovacic_tester.mpl"
```

The above will load the kovacic_class and the testsuite module. Once the above is successfully completed, then to solve an ode the command is

```
ode := diff(y(x),x$2)+diff(y(x),x)+y(x)=0;
o := Object(kovacic_class,ode,y(x)); #create the object
sol := o:-dsolve();
```

The above command will automatically try all the cases that have been detected one by one until a solution is found. If no solution is found, it returns FAIL. To verify the solution, the command is

```
if sol<>FAIL then
  odetest(sol,ode);
fi;
```

Which returns 0 if the solution is correct.

A note on the type of ode's supported: it is recommenced to use only ode's with numeric coefficients and not symbolic coefficients. This is because the Kovacic algorithm needs to decide if the degree \( d \) of the polynomial \( p(x) \) is non-negative or not. If some of the coefficients are purely symbolic, then it can fail to decide this. An example of this is given in the original Kovacic paper as example 2 on page 14, which is to solve the Bessel ode \( y'' = \left( \frac{4n^2-1}{4x^2} - 1 \right) y \). This will now return FAIL since the algorithm can not decide if \( d \) is non-negative integer without knowing any assumptions or having numerical value for \( n \). Replacing \( n \) by any half odd integer, then it can solve it as follows

```
ode :=diff(diff(y(x),x),x)=((4*n^2-1)/(4*x^2)-1)*y(x);
n :=-3/2;
o := Object(kovacic_class,ode,y(x));
sol := o:-dsolve();
```

To solve an ode using specific case number, say case 2, the command is

```
ode := ...;
o := Object(kovacic_class,ode,y(x));
sol := o:-dsolve_case(2);
```
If the ode happened to satisfy cases 1 and 2 for an example, then the above command will only use case 2 to solve it and will skip case 1. If the command `o:-dsolve()` was used instead, then the ode will be solved using case 1 instead as that is the first one tried. Case 2 will only be tried if no solution is found using case 1.

The object created above, named as “o”, has additional public methods that can be invoked. The following is description of all public methods available.

- `o:-get_y_ode()` This returns the original ode.
- `o:-get_z_ode()` This returns the ode solved by Kovacic algorithm which is \( z'' = rz \).
- `o:-get_r()` This returns \( r \) only.
- `o:-get_poles()` This returns list of the poles of \( r \). It has the format
  
  \[
  \left[ \left[ \text{pole location}, \text{pole order} \right], \left[ \text{pole location}, \text{pole order} \right], \ldots \right]
  \]

  If there are no poles, then the empty list `[]` is returned.
- `o:-get_order_at_infinity()` This returns the order of \( r \) at infinity.
- `o:-get_possible_cases()` This returns list of possible Kovacic cases detected which can be `[1]`, `[2]`, `[1,2]`, `[1,2,3]`. If no Kovacic cases are found, then the empty list `[]` is returned.
- `o:-get_case_used()` This returns the actual case number used if solution to the ode was successful. This can be 1, 2 or 3. If no solution is found after trying all cases whose conditions were satisfied, then \(-1\) is returned.
- `o:-get_n_case_3()` This return \( n \) used when case 3 was used to solve the ode. This can be 4, 6 or 12. If case 3 was not used, or no solution is found, then \(-1\) is returned.

To run the full testsuite of 3000 ode’s that comes with the package, the command is

```plaintext
kovacic_tester:-unit_test_main_api();
```

To run testsuite using specific cases only, the commands are

```plaintext
kovacic_tester:-unit_test_case_1();
kovacic_tester:-unit_test_case_2();
kovacic_tester:-unit_test_case_3();
```

43
5.2 Source code

```maple
# FILE NAME : KOV.mpl
#
# Copyright (C) 2022, Nasser M. Abbasi. All rights reserved
# email: nma@12000.org
#
# Free software to use and modify in anyway as long as the above
# copyright notice remains attached in the file
#
# Change history
#-----------------
# Oct 27, 2022. Initial version
#
# Note that the latest version and any updates can alaways be obtained
# from the author web site at
# http://12000.org/my_notes/my_paper_on_kovacic/paper.htm
#
# Any problems found in the software please report so I can correct.
# This implementation was done using Maple 2022.2 on windows 10.
#
# An Object oriented implementation of the Kovacic algortithm
# using Maple 2022.
#
# based on original Kovacic paper description of the algorithm.
#
# This file contains two modules. The first is called kovacic_class
# used to create kovacic object. The second is kovacic_tester module
# used for unit testing the kovacic_class module
#
# To use this file just do
#
# read "KOV.mpl"
# ode := diff(y(z),z$2)=2/z^2*y(z)
# o := Object(kovacic_class,ode,y(z));
# sol := o:-dsolve();
#
# Make sure to set the currentdir() in Maple correctly in order to find
# where you downloaded the file "KOV.mpl" to.
#
# This class is used to solve an ode using Kovacic algorithm.
#
# Please see documantation section in the paper for additional
# information on using this class.

kovacic_class := module()
```
option object;

#class to hold one entry in the gamma set for case 1
local case_one_gamma_entry := module()
  option object;
  export pole_location := 0;
  export pole_order := 0;
  export sqrt_r := 0;
  export alpha_plus := 0;
  export alpha_minus := 0;
  export b := 0;
end module;

#class to hold \( O_{\infty} \) information for case 1
local case_one_O_inf := module()
  option object;
  export sqrt_r_inf := 0;
  export alpha_plus_inf := 0;
  export alpha_minus_inf := 0;
  export a := 0;
  export b := 0;
end module;

#class to hold one entry in the gamma set for case 2,3
local case_2_and_3_gamma_entry := module()
  option object;
  export pole_location := 0;
  export pole_order := 0;
  export Ec::set := \{\};
  export b := 0;
end module;

#PRIVATE variables for the kovacic class
# only methods inside this class can access these
local original_ode;

#coefficients of original linear ode \( A y'' + B y' + C y = 0 \)
local A, B, C;

#original ode dependent and independent variables
local y::symbol, x::symbol;

local modified_ode; #this is the \( z'' = r z \) ode;
local z::symbol; #dependent variable for the \( r \_ode \) \( z''(x) = r z(x) \)
local r, s, t; #where \( r = s/t \);

local O_inf; #order of \( r \) at infinity. degree(s)-degree(t)

# poles or \( r \). format is [ [pole,order],[pole,order],...] 
# if no poles, for example \( r = x \), then list is empty []
# this means pole order zero is not in the list, since no pole.
local poles_list::list := [];

# contains all possible cases detected. Hence [1] or [1,2] or [1,2,3]
# or remains empty [] for case 4.
local list_of_possible_cases::list := [];

local case_used_to_solve::integer:-1; # this will have case used: 1,2 or 3

# this will have n = 4, 6 or 12 used for case 3 only
local n_used_for_case_3::integer:-1;

#--------------------------------------------------------------------
# CONSTRUCTOR
#--------------------------------------------------------------------
export ModuleCopy::static:= proc(_self,
    proto::kovacic_class,
    ode::'=',func::function(name),$
local A,B,C;
local x,y,r,z::nothing,s,t;

x,y,A,B,C := parse_and_validate_ode(ode,func);

_self:-original_ode := ode;

# force r to be relatively prime ratio of 2 polynomials
r := normal((2*diff(B,x)*A-2*B*diff(A,x)+B^2-4*A*C)/(4*A^2));
if not type(r,'ratpoly'(anything,x)) then
    ERROR("r = ", r, " is not polynomial or rational function in ",x);
fi;

s := numer(r);
t := denom(r);

# save all findings into the object private data
# so it can be used later
_self:-modified_ode := diff(z(x),x$2) = r*z(x);
_self:-O_inf := degree(t,x)-degree(s,x);

_self:-r := r;
_self:-x := x;
_self:-y := y;
_self:-z := z;
_self:-s := s;
_self:-t := t;
_self:-C := C;
_self:-B := B;
_self:-A := A;

# determine all poles and order
_self:-generate_poles_and_order_list();
#determine all possible kovacic cases
_self:-find_possible_cases();
return NULL;
end proc;

# This module private function is called by constructor to parse and
# validate the ode
local parse_and_validate_ode:=proc(ode::'=',func::function(name),$)
local x,y,A,B,C,L::list;
local item,dep_variables_found;
if nops(func)<>1 then
ERROR("dependent variable ",func," has more than one argument");
fi;

y := op(0,func);
x := op(1,func);

#basic verification
if not has(ode,y) then ERROR("Supplied ",ode," has no ",y); fi;
if not has(ode,x) then ERROR("Supplied ", ode," has no ",x); fi;
if not has(ode,func) then ERROR("Supplied ",ode," has no ",func); fi;

#check it is second order
if PDEtools:-difforder(ode)<>2 then
ERROR("Only second order ode's can be used in Kocacic algorithm. ");
fi;

#check all dependent variables in ode which is y, match given func y(x)
try
dep_variables_found := PDEtools:-Library:-GetDepVars([y],ode);
catch:
ERROR(lastexception);
end try;

#o over dep_variables_found and check the
#independent variable is same as x i.e. ode can be y'(z)+y(z)=0 but
#function is y(x).
for item in dep_variables_found do
if not type(item,function) then
ERROR("Parse error. Expected ",func," found ",item," in ", ode);
else
if op(1,item) <> x then
ERROR("Parse error. Expected ",func," found ",item," in ",ode);
fi;
if nops(item)<>1 then
ERROR("Parse error. One argument allowed in ",
func," found ", item," in ", ode);
fi;
fi;
fi;
# now go over all indents in ode and check that y shows as y(x)
# and not as just y as the PDEtools:-Library:-GetDepVars(_self:-y,ode)
# code above does not detect this. i.e. it does not check y'(x)+y=0
if numelems(indets(ode,identical(y))) > 0 then
    ERROR("Parsing error, Can not have ",y," with no argument inside ",ode);
fi;

# check ode is linear in y(x)
if not has(DEtools:-odeadvisor(ode,func,['linear']),'_linear') then
    ERROR("Only linear ode's can be used in Kovacic algorithm.");
fi;

# extract coefficients of the ode
L:=DEtools:-convertAlg(ode,func); # this only works on linear ode's
if L[2]<>0 then ERROR("Not homogeneous ode"); fi;

# Finished parsing the ode. A y''+B y' + C y = 0
C := L[1,1];
B := L[1,2];
A := L[1,3];

if not type(A,'ratpoly'(anything,x)) or not type(B,'ratpoly'(anything,x))
    or not type(C,'ratpoly'(anything,x)) then
    error "ode coefficients are not rational functions of ",x;
fi;

return x,y,A,B,C;
end proc;

#--------------------------------------------------------------------
# main API. Called to solve the ode using Kovacic algorithm.
# returns the solution in the form of y(x)=.... or if no solution
# is found returns FAIL.
# see user guide how to use.
#--------------------------------------------------------------------
export dsolve::static:=proc(_self,$)
local sol:=FAIL, current_case_number::posint;
if nops(_self:-list_of_possible_cases) = 0 then
    return FAIL;
fi;

# keep trying all possible cases, starting from case 1 to case 3.
# until one case succeed or all are tried
for current_case_number in _self:-list_of_possible_cases do
    sol := _self:-dsolve_case(current_case_number);
    if sol<>FAIL then
        _self:-case_used_to_solve:=current_case_number;
        return sol;
end proc;
268     fi;
269     od;
270     return FAIL;
271 end proc;
272
#--------------------------------------------------------------------------------
# Called to solve the ode using a specific case number
# made public to allow user to solve using specific case
#--------------------------------------------------------------------------------
export dsolve_case::static:=proc(_self,case_number::posint,$)
local sol;

if case_number>3 then ERROR("Only case number 1,2,3 are allowed"); fi;

if nops(_self:-list_of_possible_cases)=0 then
   ERROR("No possible cases detected for this ode"); fi;

if not member(case_number,_self:-list_of_possible_cases) then
   ERROR("Case ", case_number,
" not one of possible cases ", _self:-list_of_possible_cases);
fi;

if case_number = 1 then
   sol:= _self:-solve_case_1();
elif case_number = 2 then
   sol:= _self:-solve_case_2();
else
   sol:= _self:-solve_case_3();
fi;

_self:-case_used_to_solve := case_number;
return sol;
end proc;

#--------------------------------------------------------------------------------
# returns back the z''(x) = r(x) z(x) ode, which is the one
# actually solved by Kovacic algorithm
#--------------------------------------------------------------------------------
export get_z_ode::static:=proc(_self,$)
local x := _self:-x;
local r := _self:-r;
local z := _self:-z;

if _self:-s = 0 then
   return diff(z(x),x$2) = 0;
else
   return diff(z(x),x$2) = numer(r)/denom(r) * z(x);
fi;
end proc;

#--------------------------------------------------------------------------------
# returns back the r term in the z''(x) = r(x) z(x) ode

export get_r::static:=proc(_self,$)
  if _self:-s = 0 then
    return 0;
  else
    return numer(_self:-r)/denom(_self:-r);
  fi;
end proc;

# returns s, where r = s/t and z'' = r z(t)

export get_s::static:=proc(_self,$)
  return _self:-s;
end proc;

# returns t, where r = s/t and z'' = r z(t)

export get_t::static:=proc(_self,$)
  return _self:-t;
end proc;

# Returns back the original ode ( A y'' + B y' + C y = 0 )

export get_y_ode::static:=proc(_self,$)
  return _self:-original_ode;
end proc;

# returns list of poles of r, where z''(x) = r z(x)
# The list has format [ [pole location,order], ... ]

export get_poles::static:=proc(_self,$)
  return _self:-poles_list;
end proc;

# returns O_infinity of r, where z''(x) = r z(x)

export get_order_at_infinity::static:=proc(_self,$)
  return _self:-O_inf;
end proc;

# returns list of possible kovacic cases possible.

export get_possible_cases::static:=proc(_self,$)::list;
  return _self:-list_of_possible_cases;
end proc;
![Image of a page from a document with code and comments in a programming language, specifically Maple. The code is written in a readable format with comments explaining the purpose of each function and procedure. The comments are in English, and the code is structured with proper indentation and syntax, making it clear for understanding.]
#check for case 3
if not member(0,poles_order)
and nops(select(Z-> Z>2,poles_order))=0
and _self:-O_inf>=2 then
L:= [ op(L),3 ];
fi;
fi;
fi;

_self:-list_of_possible_cases := L;
return NULL;
end proc;

#----------------------------------------------
#called to find all poles of r and the order
#of each pole. Uses Maple's sqrtfree.
#----------------------------------------------
local generate_poles_and_order_list::static:=proc(_self,$)
local L::list := [];
local sol::list;
local current_sol;
local poles_list::list;
local current_pole;
local r := _self:-r, x := _self:-x;
poles_list := sqrtfree(denom(r),x);
poles_list := poles_list[2,..]; #we do not need the overall factor
if nops(poles_list) = 0 then
_self:-poles_list:=[];
else
for current_pole in poles_list do
sol := solve(current_pole[1]=0,[x]);
sol := ListTools:-Flatten(sol);
for current_sol in sol do
L := [op(L), [rhs(current_sol),current_pole[2] ] ];
od;
end proc:
return NULL;
# CASE ONE IMPLEMENTATION

# returns ode solution using case 1, or FAIL is no solution exist

local solve_case_1::static:=proc(_self,$)
    local O_infinity_set::kovacic_class:-case_one_O_inf;
    local gamma_set::set(kovacic_class:-case_one_gamma_entry):={};
    gamma_set, O_infinity_set := _self:-case_1_step_1();
    return _self:-case_1_step_2(gamma_set, O_infinity_set);
end proc;

#--------------------------------------------------------------------
# called from _self:-solve_case_1()
#--------------------------------------------------------------------

local case_1_step_1::static:=proc(_self,$)::
    set(kovacic_class:-case_one_gamma_entry),
    kovacic_class:-case_one_O_inf;
    local current_pole;
    local e::kovacic_class:-case_one_gamma_entry;
    local o::kovacic_class:-case_one_O_inf;
    local b,a,v,i::integer;
    local b_coeff_in_r,b_coeff_in_r_inf_square;
    local x := _self:-x, r := _self:-r;
    local N::integer;
    local laurent_c;
    local current_term;
    local b_from_r,b_from_laurent_series;
    # this contains all information generated for each pole of r
    # this is what is called the set GAMMA in the paper and
    # in the diagram of algorithm above.
    local gamma_set::set(kovacic_class:-case_one_gamma_entry) := {};
    if nops(_self:-poles_list) = 0 then
        gamma_set := {};
    else
        for current_pole in _self:-poles_list do
            e := Object(kovacic_class:-case_one_gamma_entry);
            e:-pole_location := current_pole[1];
            e:-pole_order := current_pole[2];
            if e:-pole_order =1 then
                e:-sqrt_r := 0;
                e:-alpha_plus := 1;
                e:-alpha_minus := 1;
                gamma_set := { op(gamma_set), e };
            elif e:-pole_order = 2 then
                #
            else
                gamma_set := { op(gamma_set), e };
            end if;
        end for;
    end if;
end proc;
\[
e^{-\sqrt{r}} := 0; \\
e^{-b} := _self:b\text{-partial\_fraction}(r,x,e:-pole\_location,2); \\
e^{-\alpha_+} := 1/2+1/2*sqrt(1+4*e:-b); \\
e^{-\alpha_-} := 1/2-1/2*sqrt(1+4*e:-b); \\
gamma_set := \{ \text{op}(\gamma_set), e \}; \\
\]

else

\[
v := e:-pole\_order/2; \\
e^{-\sqrt{r}} := 0; \\
\]

for N from 2 to v do

\[
\text{laurent}_c := _self:-\text{laurent\_coeff}(\sqrt{r},x, \\
e:-pole\_location,v,N); \\
\text{current\_term} := \text{laurent}_c/(x-e:-pole\_location)^N; \\
e^{-\sqrt{r}} := e^{-\sqrt{r}} + \text{current\_term}; \\
\]

if N = v then

\[
a := \text{laurent}_c; \\
\fi;
\]
for od;

\[
b_{\text{from\_r}} := _self:-b\text{-partial\_fraction}(r,x,e:-pole\_location,v+1); \\
b_{\text{from\_laurent\_series}} := _self:-\text{laurent\_coeff}(\sqrt{r},x, \\
e:-pole\_location,v,v+1); \\
e^{-b} := b_{\text{from\_r}} - b_{\text{from\_laurent\_series}}; \\
e^{-\alpha_+} := 1/2*((e:-b)/a + v); \\
e^{-\alpha_-} := 1/2*(-(e:-b)/a + v); \\
gamma_set := \{ \text{op}(\gamma_set), e \}; \\
\]
fi;
for od;

\[
o := \text{Object}(\text{case\_one\_O\_inf}); \\
\]

if _self:-0\_inf > 2 then

\[
o:-\sqrt{r}\_\inf := 0; \\
o:-\alpha_+\_\inf := 0; \\
o:-\alpha_-\_\inf := 1; \\
\]
else #order at infinity -2*v<= 0 which must be even
v := (-_self:-0,inf) / 2;
o:-sqrt_r_inf := 0;
for i from 0 to v do
    laurent_c := _self:-laurent_coeff(sqrt(r),x,infinity,v,i);
o:-sqrt_r_inf := o:-sqrt_r_inf + laurent_c*x^i;
    if i = v then
        o:-a := laurent_c;
    fi;
od;
b_coeff_in_r_inf_square := _self:-laurent_coeff(
    o:-sqrt_r_inf^2,x,0,v,v-1);
b_coeff_in_r := _self:-get_coefficient_of_r(v-1);
o:-b := b_coeff_in_r - b_coeff_in_r_inf_square;
o:-alpha_plus_inf := 1/2*( (o:-b)/(o:-a) - v);
o:-alpha_minus_inf := 1/2*( -(o:-b)/(o:-a) - v);
fi;
return gamma_set,o;
end proc;
#--------------------------------------------------------------------
# Finds b coefficient in r for the case one only when v<=0
# using long division
#--------------------------------------------------------------------
local get_coefficient_of_r::static:=proc(_self,the_degree,$)
    local r := _self:-r;
    local x := _self:-x;
    local c,R,t;
    try
        c := coeff(r,x,the_degree);
    catch:
        R := rem(numer(r),denom(r),x);
        t := denom(r);
        c := lcoeff(R)/lcoeff(t);
    end try;
    return c;
end proc;
#--------------------------------------------------------------------
# called from _self:-solve_case_1()
# This determines the set of d non-negative integers and
# corresponding w for each d.
#--------------------------------------------------------------------
local case_1_step_2::static:=proc(_self,
    gamma_set::set(kovacic_class:-case_one_gamma_entry),
    O_infinity::kovacic_class:-case_one_0_inf,$)
local d,N,K,number_of_poles::integer,current_alpha_infinity;
local item::list;
local the_sign::integer,sign_list::list;
local w;
local r_solution,y_solution;
local x := _self:-x;

#this will contains all data found for any nonnegative d. Each
#entry will be a list of this form
# [d, w]
local good_d_found := Array(1..0);
local B::Matrix; #this will contain good_d_found but as matrix

number_of_poles := nops(gamma_set);
sign_list := combinat:-permute([1$number_of_poles,-1$number_of_poles],
number_of_poles);

for K,current_alpha_infinity in
[O_infinity:-alpha_plus_inf,O_infinity:-alpha_minus_inf] do
    for item in sign_list do
        d := 0;
        if number_of_poles>0 then
            for N,the_sign in item do
                if the_sign = -1 then
                    d := d-gamma_set[N]:-alpha_minus;
                else
                    d := d-gamma_set[N]:-alpha_plus;
                fi;
            od;
        fi;
        d := simplify(d + current_alpha_infinity);
        if type(d,'integer') and d >= 0 then
            w := 0;
            for N,the_sign in item do
                if number_of_poles>0 then
                    if the_sign = -1 then
                        w := w +(-1)*gamma_set[N]:-sqrt_r +
                            (gamma_set[N]:-alpha_minus)/
                            (x - gamma_set[N]:-pole_location);
                    else
                        w := w + gamma_set[N]:-sqrt_r +
                            (gamma_set[N]:-alpha_plus)/
                            (x - gamma_set[N]:-pole_location);
                    fi;
                fi;
            od;
            good_d_found[d] := good_d_found[d] +1:
        fi;
    od;
define

if K = 1 then
    w := w + O_infinity:-sqrt_r_inf;
else
    w := w - O_infinity:-sqrt_r_inf;
fi;

good_d_found := [ d, w];
fi;

od;

if numelems(good_d_found) = 0 then return FAIL; fi;

# now convert the array to Matrix, and sort on d, so we
# start will the smallest
# degree d, which is the first column, as that will be most efficient.
# convert to set first, to remove any possible duplicate entries
# then convert to Matrix

convert(good_d_found,set);
B := convert(convert(%,[list],Matrix);
B := B[sort(B[..,1], 'output'= 'permutation')];

for N from 1 to LinearAlgebra:-RowDimension(B) do
    r_solution := _self:-case_1_step_3(B[N,1], B[N,2]); #(d,w)
    if r_solution <> FAIL then
        y_solution := _self:-build_y_solution_from_r_solution(r_solution);
        return y_solution;
    fi;
    od;
return FAIL;
end proc;

# ---------------------------------------------------------------
# called from _self:-case_1_step_2()
# ---------------------------------------------------------------
local case_1_step_3::static:=proc(_self,d::nonnegative,w,$)
    local x := _self:-x;
    local r := _self:-r;
    local p;
    local i::integer;
local a::nothing;
local eq;
local coeff_sol;
local tmp,W;
local final_result;

p := x^d;

for i from d-1 by -1 to 0 do
    p := p + a[i] * x^i;
od;

#using original kovacis method. Not Smith.

 eq := simplify( diff(p,x$2)+2*w*diff(p,x)+(diff(w,x)+w^2-r)*p) = 0;

if d = 0 then
    if not evalb(eq) then return FAIL; fi;
else
    #solve for coefficients
    try
        coeff_sol := timelimit(30,solve(
            identity(eq,x), [seq(a[i],i=0..d-1)]));
        catch:
            return FAIL;
        end try;
    fi;
    if nops(coeff_sol) = 0 then return FAIL; fi;

    tmp := map(evalb,coeff_sol[1]);
    if has(tmp,true) then return FAIL; fi;
    p := eval(p,coeff_sol[1]); #to force a[i] solutions to update
fi;

W := diff(p,x)/p + w;
W := radsimp(W);

tmp := diff(W,x)+W^2;

if evalb( tmp = r) or is(tmp = r) or simplify(tmp-r)=0 then #can be used
    try
        tmp := int(w, x);
        catch:
            return FAIL;
        end try;
    fi;
    if has(tmp,int) then return FAIL; fi;

final_result := simplify(p*exp(tmp));
if has(final_result,signum) or has(final_result,csgn) then
    final_result := p*exp(tmp);
fi;

return final_result;
else
    return FAIL;
fi;

end proc;

#--------------------------------------------------------------------
#
# CASE TWO IMPLEMENTATION
#
# returns ode solution using case 2, or FAIL is no solution exist
#
#--------------------------------------------------------------------
local solve_case_2::static:=proc(_self, $)

    local E_inf::set;
    local gamma_set::set(kovacic_class:-case_2_and_3_gamma_entry):={};

    gamma_set, E_inf := _self:-case_2_step_1();
    return _self:-case_2_step_2(gamma_set, E_inf );

end proc;

#--------------------------------------------------------------------
#
#--------------------------------------------------------------------
local case_2_step_1::static:=proc(_self, $):-

    set(kovacic_class:-case_2_and_3_gamma_entry), set;

    local current_pole;
    local e::kovacic_class:-case_2_and_3_gamma_entry;
    local E_inf::set;
    local b;
    local x := _self:-x;
    local r := _self:-r;

    #this contains all information generated for each pole of r
    #this is what is called the set GAMMA in the paper and in the diagram of
    #algorithm above.
    local gamma_set::set(kovacic_class:-case_2_and_3_gamma_entry) := {};

    for current_pole in _self:-poles_list do
        e := Object(kovacic_class:-case_2_and_3_gamma_entry);
        e:-pole_location := current_pole[1];
        e:-pole_order := current_pole[2];

        if e:-pole_order = 1 then
            e:-Ec := {4};
            gamma_set := { op(gamma_set), e };       
        elif e:-pole_order = 2 then
            e:-b := _self:-b_partial_fraction(r,x,e:-pole_location,2);
\[ e:\text{-}Ec := \{2,2+2\sqrt{1+4\ e:\text{-}b},2-2\sqrt{1+4\ e:\text{-}b}\}; \]
\[ e:\text{-}Ec := \text{select}(z\rightarrow\text{type}(z,\text{integer}),e:\text{-}Ec); \]
\[ \text{gamma\_set} := \{ \text{op(gamma\_set)}, e \}; \]

\begin{verbatim}
e:-Ec := \{e:-pole\_order\};
gamma\_set := \{ \text{op(gamma\_set)}, e \};
\end{verbatim}

\[ \text{else} \]
\begin{verbatim}
e:-Ec := \{e:-pole\_order\};
gamma\_set := \{ \text{op(gamma\_set)}, e \};
fi;
\end{verbatim}

\[ \text{od}; \]

\[ \text{if} \ _\text{self}:\text{-}0\_\text{inf}>2 \ \text{then} \]
\[ \text{E\_inf} := \{0,2,4\}; \]
\[ \text{elif} \ _\text{self}:\text{-}0\_\text{inf}=2 \ \text{then} \]
\begin{verbatim}
b := \text{lcoeff(}_\text{self}:\text{-s}) / \text{lcoeff(}_\text{self}:\text{-t});
E\_inf := \{2,2+2\sqrt{1+4\ b},2-2\sqrt{1+4\ b}\};
E\_inf := \text{select}(z\rightarrow\text{type}(z,\text{integer}),E\_inf);
\end{verbatim}

\[ \text{else} \ #\text{order at infinity} \nu<2 \]
\[ \text{E\_inf} := \{\_\text{self}:\text{-}0\_\text{inf}\}; \]
\[ \text{fi}; \]

\[ \text{return} \ \text{gamma\_set},\text{E\_inf}; \]
\end{verbatim}

\[ \text{end proc;} \]

\[ \#-----------------------------------------------\]
\[ \# \text{called from}\ _\text{self}:\text{-solve\_case\_2()}\]
\[ \# \text{This determines the set of} \ d \ \text{non-negative integers} \]
\[ \# \text{corresponding} \ w \ \text{for each} \ d. \]
\[ \#-----------------------------------------------\]
\[ \text{local} \ \text{case\_2\_step\_2::static:=proc(}_\text{self},\]
\[ \text{gamma\_set::set(kovacic\_class:-case\_2\_and\_3\_gamma\_entry),}\]
\[ \text{E\_inf::set,}\]
\[ \text{)}\]
\[ \text{local} \ \text{L::list := [];}\]
\[ \text{local} \ \text{item, current\_E\_inf;}\]
\[ \text{local} \ \text{d,N,ee,theta;}\]
\[ \text{local} \ \text{x := }\_\text{self}:\text{-x;}\]
\[ \text{local} \ \text{r\_solution,y\_solution;}\]

\[ \#\text{this will contains all data found for any nonnegative} \ d. \ \text{Each}\]
\[ \#\text{entry will be a list of this form}\]
\[ \# \ [d, theta]\]
\[ \#\text{so if we obtain say} \ 3 \ \text{values of} \ d \ \text{that are nonnegative,}\]
\[ \#\text{there will be} \ 3 \ \text{such lists in this array}\]
\[ \text{local} \ \text{good\_d\_found := Array(1..0);}\]
local B::Matrix; #this will contain good_d_found as matrix

for item in gamma_set do
    L := [ op(L), convert(item:-Ec,list) ];
od;

#now find all possible tuples
if nops(L)>1 then
    L := kovacic_class:-cartProdSeq(op(L));
else
    L := L[1];
fi;

for current_E_inf in E_inf do
    for item in L do
        d := 1/2*( current_E_inf - add(item));

        if type(d,'integer') and d >= 0 then
            theta :=0;

            for N,ee in item do
                theta := theta + ee/(x-gamma_set[N]:-pole_location);
            od;

            theta := 1/2*theta;
            good_d_found ,= [ d, theta];
        fi;
    od;
    od;

if numelems(good_d_found) = 0 then return FAIL; fi;

#now convert the array to Matrix, and sort on d, so
#we start will the smallest degree d, which is the first column, as
#that will be most efficient.
#convert to set first, to remove any possible duplicat entries
#then convert to Matrix
convert(good_d_found,set);
B := convert(convert(% list),Matrix);
B := B[sort(B[.., 1], 'output'= 'permutation')];

for N from 1 to LinearAlgebra:-RowDimension(B) do
    r_solution := _self:-case_2_step_3(B[N,1], B[N,2]); #(d,theta)

    if r_solution <> FAIL then
        y_solution := _self:-build_y_solution_from_r_solution(r_solution);
        return y_solution;
    fi;
    od;
return FAIL;
end proc;

# *--------------------------------------------------------------------
# called from _self:-case_2_step_2()
# *--------------------------------------------------------------------
local case_2_step_3::static:=proc(_self,d::nonnegative,theta,$)
local p, i;
local a::nothing;
local tmp;
local coeff_sol := [];
local eq;
local phi;
local sol_w;
local current_w;
local r:=_self:-r;
local w::nothing;
llocal x := _self:-x;
llocal sol;

   p := x^d;
for i from d-1 by -1 to 0 do
   p := p + a[i] * x^i;
   od;

   eq:= simplify(diff(p,x$3) + 3*theta*diff(p,x$2) +
   (3*theta^2+3*diff(theta,x) -4*r) * diff(p,x)
   + ( diff(theta,x$2)+3*theta*diff(theta,x)+theta^3 -
   4*r*theta - 2*diff(r,x))*p) = 0;

if d=0 then
   if not evalb(eq) then return FAIL; fi;
else
   #solve for polynomial coefficients
   try
coeff_sol:= timelimit(30,solve(
   identity(eq,x), [seq(a[i],i=0..d-1)]));
   catch:
      return FAIL;
   end try;
   if nops(coeff_sol) = 0 then return FAIL; fi;

   tmp := map(evalb,coeff_sol[1]);
   if has(tmp,true) then return FAIL; fi;
   p := eval(p,coeff_sol[1]); #to force a[i] solutions to update
fi;

phi := theta + diff(p,x)/p;
eq := w^2 - phi*w + simplify(1/2*diff(phi,x)+1/2*phi^2-r) = 0;

try
    sol_w := timelimit(30,solve(eq,[w])); # changed to []
    sol_w := ListTools:-Flatten(sol_w);
catch:
    return FAIL;
end try;

if nops(sol_w) = 0 then return FAIL; fi;

for current_w in sol_w do
    current_w := radsimp(rhs(current_w));
    # verify w before using it. Added 1/12/2022 4 PM
    tmp := diff(current_w,x)+current_w^2;
    if evalb( tmp= r) or is(tmp = r) or simplify(tmp-r)=0 then
        try
            sol := timelimit(30,int(current_w, x));
            catch:
                return FAIL;
            end try;
            if has(sol,int) then return FAIL; fi;
            return simplify(exp(sol));
        fi;
    fi;
od;
end proc;

#--------------------------------------------------------------------
# CASE THREE IMPLEMENTATION
#--------------------------------------------------------------------

local solve_case_3::static:=proc(_self,$)
    local E_inf::set;
    local gamma_set::set(kovacic_class:-case_2_and_3_gamma_entry):={};
    local sol;
    local w_degree::list := [4,6,12];
    local n::posint;

    # these are possible degrees of w to try until one works or none works
    for n in w_degree do
        gamma_set, E_inf := _self:-case_3_step_1(n);
        sol := _self:-case_3_step_2(gamma_set, E_inf, n );
    end for;
end proc;
if sol<>FAIL then return sol; fi;

return FAIL;

end proc;

#--------------------------------------------------------------------
# First step in case 3
#--------------------------------------------------------------------
local case_3_step_1::static:=proc(_self,
n::posint,
$: set(kovacic_class:-case_2_and_3_gamma_entry), set;

local current_pole;
local e::kovacic_class:-case_2_and_3_gamma_entry;
local E_inf::set;
local b,k;
local x := _self:-x;
local r := _self:-r;

#this contains all information generated for each pole of r
#this is what is called the set GAMMA in the paper and in the diagram of
#algorithm above.
local gamma_set::set(kovacic_class:-case_2_and_3_gamma_entry):={};

for current_pole in _self:-poles_list do

e := Object(kovacic_class:-case_2_and_3_gamma_entry);

e:-pole_location := current_pole[1];
e:-pole_order := current_pole[2];

if e:-pole_order = 1 then

e:-Ec := {12};
gamma_set := { op(gamma_set), e }; 

elif e:-pole_order = 2 then

e:-b := _self:-b_partial_fraction(r,x,e:-pole_location,2);
e:-Ec := {seq( 6+12*k/n*sqrt(1+4*(e:-b)),k=-n/2..n/2,1)};
e:-Ec := select(z->type(z,integer),e:-Ec);
gamma_set := { op(gamma_set), e }; 

else
ERROR(“Internal error. Case 3 can only have poles of order 1 or 2”);
fi;

od;

#same formula, but different b
b := lcoeff(_self:-s) / lcoeff(_self:-t);
E_inf := \{seq(6+12*k/n*sqrt(1+4*b),k=-n/2..n/2,1)\};
E_inf := select(z->type(z,integer),E_inf);
return gamma_set,E_inf;
end proc;

# Second step in case 3
# called from _self:-solve_case_3()
# This determines the set of d non-negative integers and
# corresponding w for each d.
local case_3_step_2::static:=proc(_self,gamma_set::set(kovacic_class:-case_2_and_3_gamma_entry),E_inf::set,n::posint,$)
local L::list := [];
local item,current_E_inf;
local d,ee,theta,N;
local x := _self:-x;
local r_solution,y_solution;
local S;
local current_iteration::integer;
local tmp;

#this will contains all data found for any nonnegative d. Each
#entry will be a list of this form
# [d, theta, S]
#so if we obtain say 3 values of d that are nonnegative
local good_d_found := Array(1..0);
local B::Matrix; #this will contain good_d_found as matrix

DEBUG();
for item in gamma_set do
L := [ op(L), convert(item:-Ec,list) ];
od;

#now find all possible tuples
if nops(L)>1 then
L := kovacic_class:-cartProdSeq(op(L));
else
L := L[1];
fi;
current_iteration:=0;
for current_E_inf in E_inf do
for item in L do
current_iteration := current_iteration+1;
d := n/12*( current_E_inf - add(item));
if type(d,’integer’) and d >= 0 then
    theta :=0;
    for N,ee in item do
        theta := theta + ee/(x-gamma_set[N]:-pole_location);
    od;
    theta := n/12*theta;
    theta := simplify(theta);
    S := mul( (x-gamma_set[N]:-pole_location), N=1..nops(item));
    tmp := simplify(S) assuming real;
    if not has(tmp,csgn) and not has(tmp,signum) then
        S:= tmp;
    fi;
    good_d_found ,= [ d, theta, S];
fi;
od;

#now convert the array to Matrix, and sort on d, so we start
# with the smallest
#degree d, which is the first column, as that will be most efficient.
if numelems(good_d_found) = 0 then return FAIL; fi;

#convert to set first, to remove any possible duplicate entries
#then convert to Matrix
convert(good_d_found,set);
B := convert(convert(%list),Matrix);
B := B[sort(B[.., 1], 'output'= 'permutation')];

for N from 1 to LinearAlgebra:-RowDimension(B) do
    r_solution := _self:-case_3_step_3(B[N,1], B[N,2], B[N,3],n);
    if r_solution <> FAIL then
        _self:-n_used_for_case_3 := n;
        y_solution := _self:-build_y_solution_from_r_solution(r_solution);
        return y_solution;
    fi;
od;
return FAIL;
end proc;

#-----------------------------------------------
# Third and final step in case 3.
# called from _self:-case_3_step_2(). Returns solution or FAIL is no
# solution found.
#-----------------------------------------------
local case_3_step_3::static:=proc(
    _self,
    d::nonnegative,
    theta,
S,
n::posint,
$
)

local p, i, P_minus_1;
local a::nothing;
local final_result, tmp;
local coeff_sol := [];
local sol_w;
local current_w;
local r := _self:-r;
local x := _self:-x;
local omega::symbol;
local P := Array(-1..n);
local result_of_simplify;
local result_of_is_check;
local omega_equation;

#this makes p(x). For example if d=3, then the result will be
# $p(x) = x^3 + a(2) x^2 + a(1) x + a(0)$
#where the number of unknowns to determine is always the same
#as the degree, in this case a(0),a(1),a(2)

p := x^d; #this will be 1 if degree is zero
for i from d-1 by -1 to 0 do
  p := p + a[i] * x^i;
od;

#build the P_n polynomials based on p(x) above
P[n] := -p;
for i from n by -1 to 0 do
  if i=n then
    P[i-1] := -S * diff(P[i], x) - S*theta*P[i];
P[i-1] := simplify(P[i-1]);
  else
    P[i-1] := -S * diff(P[i], x) +
             ( (n-i)*diff(S,x) - S*theta)*P[i] - (n-i)*(i+1)*S^2*r*P[i+1];
P[i-1] := simplify(P[i-1]);
  fi;
od;

#solve P[-1] for p(x) if needed.
P_minus_1 := expand(numer(radsimp(P[-1])));
if P_minus_1 <> 0 and evalb(p<>1) then
  if not hastype(P_minus_1, 'indexed') then #it must be indexed now
    ERROR("Internal error. Please report. This should not happen");
  fi;
try
\[
\text{coeff}\_\text{sol} := \text{timelimit}(30, \text{solve}(\text{identity}(P\_\text{minus}\_1, x), [\text{seq}(a[i], i=0..d-1)]));
\]
\[
catch:
\begin{align*}
\text{return FAIL;}
\end{align*}
\]
\[
\text{end}\ try;
\]
\[
\text{if nops(coeff\_sol) = 0 then} \quad \#\text{unable to solve}
\begin{align*}
\text{return FAIL;}
\end{align*}
\]
\[
\text{fi;
\]
\[
\text{map(evalb,coeff\_sol[1]);} \quad \#\text{check all solved for}
\begin{align*}
\text{if has(%,true) then return FAIL; fi;
\end{align*}
\]
\[
\text{fi;
\]
\[
\text{#build the equation for omega}
\text{omega\_equation} := 0;
\]
\[
\text{for i from 0 to n do}
\begin{align*}
\text{omega\_equation} := \text{omega\_equation} + S^i P[i]/(n-i)! \cdot \omega^i;
\end{align*}
\]
\[
\text{od;
\]
\[
\text{omega\_equation} := \text{simplify}(\text{omega\_equation});
\]
\[
\text{if nops(coeff\_sol) <> 0 then}
\begin{align*}
\text{#to force a[i] solutions to update to solved coefficients}
\text{omega\_equation} := \text{eval}(\text{omega\_equation,coeff\_sol[1]});
\end{align*}
\]
\[
\text{fi;
\]
\[
\text{try}
\begin{align*}
\text{sol\_w := timelimit}(30, \text{solve}(\text{omega\_equation} = 0, [\omega]));
\text{sol\_w := ListTools:-Flatten(sol\_w)};
\end{align*}
\]
\[
\text{catch:
\begin{align*}
\text{return FAIL;}
\end{align*}
\]
\[
\text{end}\ try;
\]
\[
\text{if nops(sol\_w) = 0 then return FAIL; fi;
\]
\[
\text{#go over each w solution and use one that works. verify before using}
\]
\[
\text{for current\_w in sol\_w do}
\begin{align*}
\text{current\_w} := \text{rhs(current\_w)};
\end{align*}
\]
\[
\text{if not has(current\_w, RootOf) then}
\begin{align*}
\text{try}
\begin{align*}
\text{current\_w} := \text{timelimit}(30, \text{simplify(current\_w))};
\end{align*}
\text{catch:
\begin{align*}
\text{NULL;}
\end{align*}
\text{end}\ try;
\end{align*}
\]
\[
\text{if is\_w\_verified(current\_w,x,r) then}
\begin{align*}
\text{final\_result} := _\text{self}:-\text{simplify}\_\text{final\_result}(current\_w,x);
\end{align*}
\]
\[
\text{if final\_result <> FAIL then}
\begin{align*}
\text{return final\_result;}
\end{align*}
\]
\[
\text{fi;
\]
fi;
else #no RootOf, try to resolve
    try
        current_w := timelimit(30,[allvalues(current_w)]);
        catch:
            return FAIL;
        end try;
        current_w := current_w[1]; #just use any root. Pick first
    if not has(current_w, RootOf) then
        try
            current_w := timelimit(30,simplify(current_w));
            catch:
                NULL;
            end try;
            if is_w_verified(current_w,x,r) then
                final_result := _self:-simplify_final_result(current_w,x);
                if final_result<>FAIL then
                    return final_result;
                fi;
            fi;
            od;
        end if;
    fi;
fi;
end proc;

# Called from case 3, step 3 to verify w
local is_w_verified::proc(current_w,x,r)::truefalse;
local tmp;
local result_of_is_check::truefalse;
local result_of_simplify;

    tmp := diff(current_w,x)+current_w^2;
    if evalb( tmp=r) then return true; fi;
    try
        result_of_simplify := timelimit(30,simplify(tmp-r));
        if evalb(result_of_simplify=0) then return true; fi;
        catch:
            NULL;
        end try;
    end try;

    try
        result_of_is_check := timelimit(30,is(tmp = r));
if result_of_is_check then return true; fi;
catch:
    NULL;
end try;

return false;
end proc;

#----------------------------------------------------------
# Called from case 3, step 3 to simplify final result
#----------------------------------------------------------
local simplify_final_result::static:=proc(_self,omega,x,$)
local final_result;
local integral_result;

try
    integral_result := timelimit(30,int(omega, x));
catch:
    return FAIL;
end try;

if has(integral_result,int) or has(integral_result,Int) then
    return exp(Int(omega, x));
fi;

try
    final_result := timelimit(30,simplify(exp(integral_result))) assuming real;
    if has(final_result,signum) or has(final_result,csgn)
        or has(final_result,abs) then
        final_result := exp(integral_result);
    fi;
    catch:
        final_result:= exp(integral_result);
    end try;

    return final_result;
end proc;

#----------------------------------------------------------
# External proc helper
# provided thanks to Joseph Riel
#----------------------------------------------------------
local cartProdSeq:= proc(L::seq(list))
local Seq::nothing,i::nothing,j;
option 'Copyright (C) 2007, Joseph Riel. All rights reserved.';
eval([subs(Seq= seq, foldl(Seq, [cat(i, 1..nargs)],
    seq(cat(i,j)= L[j], j= nargs..1, -1))])
end proc:

# Checks for special math function.
# This function was provided thanks to Carl Love.
# modified to allow some special functions.

```maple
local has_special_math_function:= subs(
  _F= {(op@FunctionAdvisor)~(FunctionAdvisor("class_members", "quiet"), "quiet")[]}
  minus ( {FunctionAdvisor("elementary", "quiet")[]} 
  union {erf,erfc, erfi, Im,Re,signum,max,argument} ),
  proc(e::algebraic, x::{name, set(name)}:= {})
    has$type(e, And(specfunc(_F), dependent(x)))
  end proc
);
```

# called if step 3 is successful. Build y solution from r solution

```maple
local build_y_solution_from_r_solution::static:=proc(_self,r_solution,$)
  local y1,y2;
  local int_B_over_A;
  local A :=_self:-A, B:=_self:-B;
  local x :=_self:-x, y:= _self:-y;
  local tmp;

  if B = 0 then
    y1 := r_solution;
    int_B_over_A := 0;
  else
    try
      int_B_over_A := timelimit(60,int(-B/A,x));
      if has(int_B_over_A,int) or has(int_B_over_A,RootOf) then
        int_B_over_A := Int(-B/A, x);
      else
        y1 := simplify(r_solution*exp(1/2*int_B_over_A));
      fi;
      catch:
        int_B_over_A := Int(-B/A,x);
        y1 := r_solution*exp(1/2*int_B_over_A);
      end try;
      fi;
    try
      y2 := timelimit(30, int( simplify(exp(int_B_over_A))/y1^2,x));
      if kovacic_class:-has_special_math_function(y2,x) then
        y2 := y1 * Int( exp(int_B_over_A)/y1^2,x)
      else
        y2 := y1 * y2;
      fi;
    end try;
  fi;
  if has(tmp,signum) or has(tmp,csgn) then
```
1510   y2 := y1 * y2;
1511 else
1512   y2 := tmp;
1513 fi;
1514 catch:
1515   y2 := y1 * Int( exp(int_B_over_A)/y1^2,x)
1516 end try;
1517 return y(x) = _C1*y1 + _C2*y2;
1518 end proc;
1519
1520#--------------------------------------------------------------------
1521# Helper private function called to obtain b from the partial fraction
1522# decomposition of r needed by the different case implementations
1523#--------------------------------------------------------------------
1524local b_partial_fraction::static:=proc(_self,
1525   r,
1526   x::symbol,
1527   pole_location,
1528   the_power::posint,$)
1529   local r_partial_fraction,T::nothing;
1530   r_partial_fraction := allvalues(convert(r,'fullparfrac',x));
1531   #adding dummy T so that select below always works
1532   r_partial_fraction := T+r_partial_fraction;
1533   select(z->hastype(z,anything/(anything*identical(x - pole_location)^the_power)),
1534      r_partial_fraction);
1535   return coeff(%,1/(x-pole_location)^the_power);
1536 end proc;
1537
1538#--------------------------------------------------------------------
1539# Helper private function called to return Laurent series coefficient
1540# of the 1/(x-c)^n term. where n=1,2,3,... for the function f(x)
1541# expandid about pole of order m
1542# if c=infinity, then x is replaced by 1/y and expansion is around 0
1543# for infinity, n=0,1,2,...
1544#--------------------------------------------------------------------
1545local laurent_coeff::static:=proc(_self,
1546   f,
1547   x::symbol,
1548   c,
1549   m::integer,
1550   n::integer,$)
1551   local the_coeff,y::nothing,fy;
1552   if c = infinity then
1553     fy := eval(f,x=1/y);
if m-n>0 then
    the_coeff := limit( diff( y^m * fy,y$(m-n)),y=0,right)/(m-n)!
elif m-n=0 then
    the_coeff := limit( y^m * fy,y=0,right)/(m-n)!
else
    the_coeff := 0;
fi;
else
    if m-n>0 then
        the_coeff := limit( diff( (x-c)^m * f,x$(m-n)),x=c,right)/(m-n)!
    elif m-n=0 then
        the_coeff := limit( (x-c)^m * f,x=c,right)/(m-n)!
    else
        the_coeff := 0;
    fi;
fi;
the_coeff := eval(the_coeff,[csgn=1,signum=1]);
return the_coeff;
end proc;
end module;

References

[1] Jerald J. Kovacic. An Algorithm for Solving Second Order Linear Homogeneous Differential Equations. J. Symb. Comput., 2(1):3–43, 1986.

[2] Carolyn J. Smith. A DISCUSSION AND IMPLEMENTATION OF KOVACICS ALGORITHM FOR ORDINARY DIFFERENTIAL EQUATIONS. Research Report CS-84-35, October 1984.

[3] B. David Saunders. An Implementation of Kovacic’s Algorithm for Solving Second Order Linear Homogeneous Differential Equations. Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation, 1981.