Computer algebra independent integration tests

Summer 2022 edition. Special build

Nasser M. Abbasi

June 7, 2022 Compiled on June 7, 2022 at 3:54pm

Contents

Chapter 1

Introduction

This report gives the result of running the computer algebra independent integration problems. The current number of problems in this test suite is [3809].

1.1 Listing of CAS systems tested

The following are the CAS systems tested:

1. Mathematica 13.0.1 (February 17, 2022) on windows 10.
2. Rubi 4.16.1 (Dec 19, 2018) on Mathematica 13.0.1 on windows 10.
3. Maple 2022.1 (June 1, 2022) on windows 10.
4. Maxima 5.46 (April 13, 2022) using Lisp SBCL 2.0.1.debian on Ubuntu 20.04 Linux under window 10 WSL 2.0 subsystem via sagemath 9.6.
5. Fricas 1.3.7 (June 30, 2021) based on based on ecl 21.2.1 on Ubuntu 20.04 Linux under window 10 WSL 2.0 subsystem via sagemath 9.6.
6. Giac/Xcas 1.9.0-7 (April 2022) on on Ubuntu 20.04 Linux under window 10 WSL 2.0 subsystem. Direct testing using C++ API.
7. Sympy 1.10.1 (March 20, 2022) Using Python 3.10.4 Ubuntu 20.04 Linux under window 10 WSL 2.0 subsystem via sagemath 9.6.
8. Mupad using Matlab 2021a with Symbolic Math Toolbox Version 8.7 on windows 10.
9. Mathics 4.0 via sagemath 9.6.

Maxima, Fricas, Mathics are called using Sagemath. This was done using Sagemath `integrate` command by changing the name of the algorithm to use the different CAS systems. Mathics was called using its own interface in Sagemath as in this example

```
from sage.interfaces.mathics import mathics
res = mathics('Integrate[Sin[x]/(3 + Cos[x])^2,x]')
```

Sympy was called directly from Python. Giac was also called directly via its C++ interface.

1.2 Results

Important note: A number of problems in this test suite have no antiderivative in closed form. This means the antiderivative of these integrals can not be expressed in terms of elementary, special functions or `Hypergeometric2F1` functions. `RootSum` and `RootOf` are not allowed.

If a CAS returns the above integral unevaluated within the time limit, then the result is counted as passed and assigned an A grade.

However, if CAS times out, then it is assigned an F grade even if the integral is not integrable, as this implies CAS could not determine that the integral is not integrable in the time limit.

If a CAS returns an antiderivative to such an integral, it is assigned an A grade automatically and this special result is listed in the introduction section of each individual test report to make it easy to identify as this can be important result to investigate.

The results given in in the table below reflects the above.

System	solved	Failed
Mathematica	% 99.61 (3794)	% 0.39 (15)
Rubi	% 99.5 (3790)	% 0.5 (19)
Fricas	% 89.16 (3396)	% 10.84 (413)
Maple	% 87.63 (3338)	% 12.37 (471)
Giac	% 79.42 (3025)	% 20.58 (784)
Maxima	% 75.58 (2879)	% 24.42 (930)
Mupad	% 73.33 (2793)	% 26.67 (1016)
Sympy	% 67.68 (2578)	% 32.32 (1231)
Mathics	% 67.08 (2555)	% 32.92 (1254)

Table 1.1: Percentage solved for each CAS

The table below gives additional break down of the grading of quality of the antiderivatives generated by each CAS. The grading is given using the letters A,B,C and F with A being the best quality. The grading is accomplished by comparing the antiderivative generated with the optimal antiderivatives included in the test suite. The following table describes the meaning of these grades.

grade	description
A	Integral was solved and antiderivative is optimal in quality and leaf size.
B	Integral was solved and antiderivative is optimal in quality but leaf size is larger than twice the optimal antiderivatives leaf size.
C	Integral was solved and antiderivative is non-optimal in quality. This can be due to one or more of the following reasons 1. antiderivative contains a hypergeometric function and the optimal antiderivative does not. 2. antiderivative contains a special function and the optimal antiderivative does not. 3. antiderivative contains the imaginary unit and the optimal antiderivative does not.
F	Integral was not solved. Either the integral was returned unevaluated within the time limit, or it timed out, or CAS hanged or crashed or an exception was raised.

Table 1.2: Description of grading applied to integration result

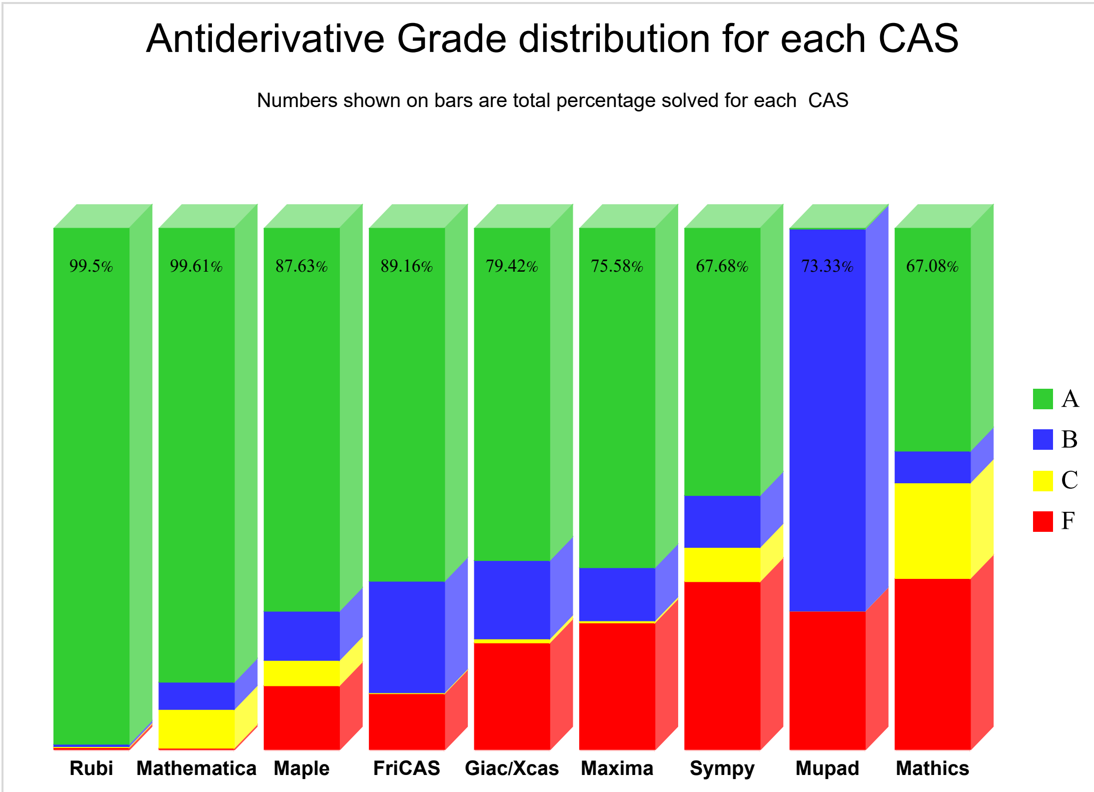
Grading is implemented for all CAS systems in this version except for CAS Mupad where a grade of B is automatically assigned as a place holder for all integrals it completes on time.

The following table summarizes the grading results.

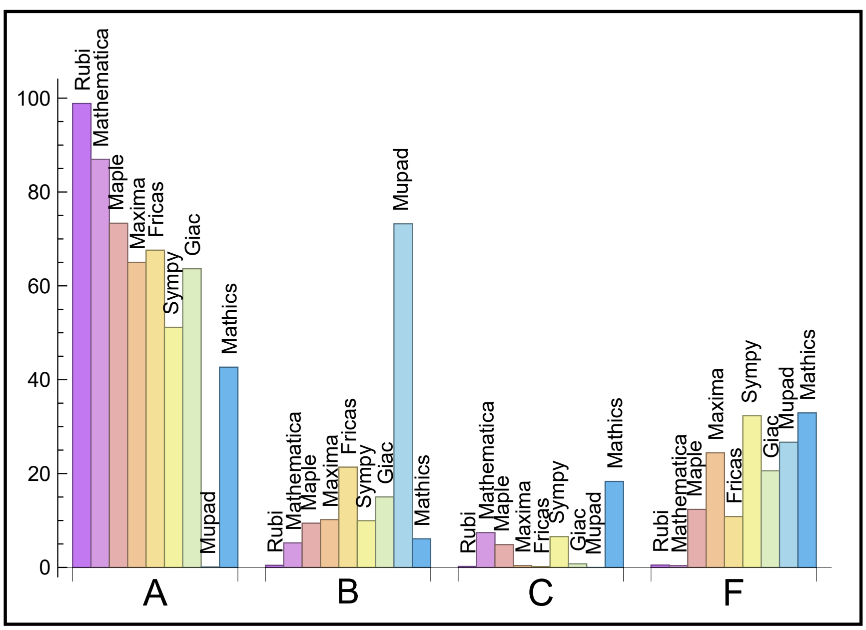
System	% A grade	% B grade	% C grade	% F grade
Rubi	98.84	0.45	0.21	0.5
Mathematica	86.95	5.22	7.43	0.39
Maple	73.35	9.43	4.86	12.37
Fricas	67.6	21.37	0.18	10.84
Maxima	65.	10.19	0.39	24.42
Giac	63.64	15.02	0.76	20.58
Sympy	51.17	9.95	6.56	32.32
Mathics	42.66	6.09	18.33	32.92
Mupad	0.11	73.22	0.	26.67

Table 1.3: Antiderivative Grade distribution for each CAS

The following is a Bar chart illustration of the data in the above table.



The figure below compares the CAS systems for each grade level.



1.2.1 Time and leaf size Performance

The table below summarizes the performance of each CAS system in terms of time used and leaf size of results.

System	Mean time (sec)	Mean size	Normalized mean	Median size	Normalized median
Rubi	0.05	78.61	1.2	42.	1.
Giac	0.13	129.69	1.95	43.	1.2
Maple	0.2	557.56	6.4	35.	0.93
Mupad	0.25	76.16	1.35	28.	0.87
Maxima	0.3	70.76	1.32	32.	0.9
Fricas	0.47	145.22	1.78	41.	1.13
Mathematica	0.85	81.31	1.2	39.	1.
Sympy	3.48	156.92	2.82	37.	1.07
Mathics	7.02	110.26	2.04	33.	1.

Table 1.4: Time and leaf size performance for each CAS

1.3 Performance per integrand type

The following are the different integrand types the test suite contains.

1. Independent tests.
2. Algebraic Binomial problems (products involving powers of binomials and monomials).
3. Algebraic Trinomial problems (products involving powers of trinomials, binomials and monomials).
4. Miscellaneous Algebraic functions.
5. Exponentials.
6. Logarithms.
7. Trigonometric.
8. Inverse Trigonometric.
9. Hyperbolic functions.
10. Inverse Hyperbolic functions.
11. Special functions.
12. Sam Blake input file.
13. Waldek Hebisch input file.

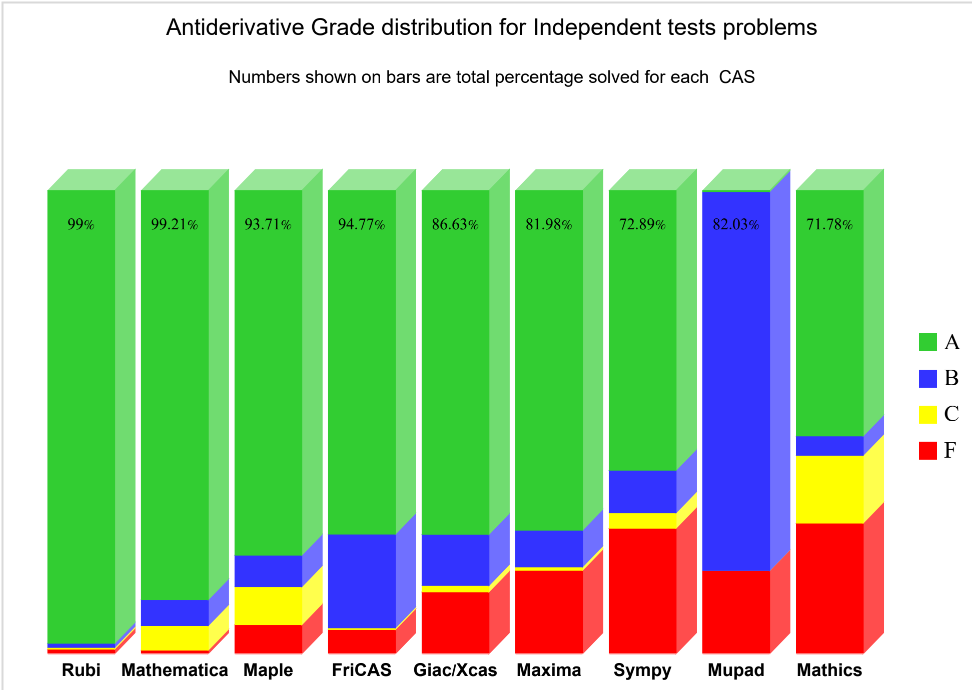
The following table gives percentage solved of each CAS per integrand type.

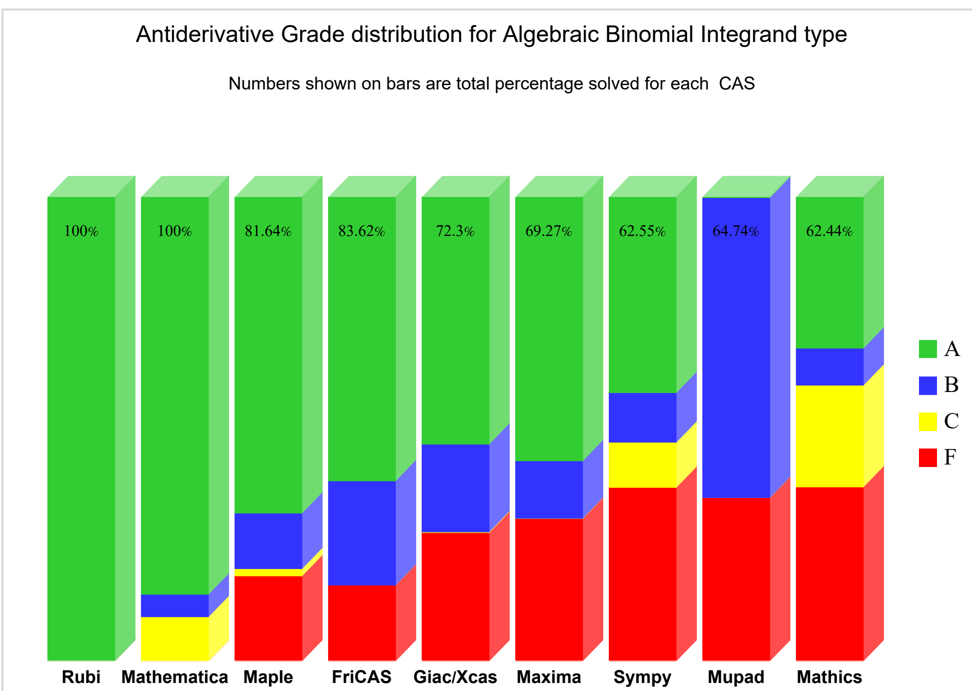
Integrand type	problems	Rubi	Mathematica	Maple	Maxima	Fricas	Sympy	Giac	Mupad	Mathics
Independent tests	1892	99.	99.21	93.71	81.98	94.77	72.89	86.63	82.03	71.78
Algebraic Binomial	1917	100.	100.	81.64	69.27	83.62	62.55	72.3	64.74	62.44
Algebraic Trinomial	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Algebraic Miscellaneous	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Exponentials	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Logarithms	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Trigonometric	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Inverse Trigonometric	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Hyperbolic	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Inverse Hyperbolic	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Special functions	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Sam Blake file	0	0.	0.	0.	0.	0.	0.	0.	0.	0.
Waldek Hebisch file	0	0.	0.	0.	0.	0.	0.	0.	0.	0.

Table 1.5: Percentage solved per integrand type

In addition to the above table, for each type of integrand listed above, 3D chart is made which shows how each CAS performed on that specific integrand type.

These charts and the table above can be used to show where each CAS relative strength or weakness in the area of integration.


Antiderivative Grade distribution for Independent tests problems
Numbers shown on bars are total percentage solved for each CAS
99%
99.21%
93.71%
94.77%
86.63%
81.98%
72.89%
82.03%
71.78%
A
B
C
F
Rubi
Mathematica
Maple
FriCAS
Giac/Xcas
Maxima
Sympy
Mupad
Mathics


Antiderivative Grade distribution for Algebraic Binomial Integrand type
Numbers shown on bars are total percentage solved for each CAS
100%
100%
81.64%
83.62%
72.3%
69.27%
62.55%
64.74%
62.44%
A
B
C
F
Rubi
Mathematica
Maple
FriCAS
Giac/Xcas
Maxima
Sympy
Mupad
Mathics

1.4 Maximum leaf size ratio for each CAS against the optimal result

The following table gives the largest ratio found in each test file, between each CAS antiderivative and the optimal antiderivative.

For each test input file, the problem with the largest ratio $\frac{\text{CAS leaf size}}{\text{Optimal leaf size}}$ is recorded with the corresponding problem number.

In each column in the table below, the first number is the maximum leaf size ratio, and the number that follows inside the parentheses is the problem number in that specific file where this maximum ratio was found. This ratio is determined only when CAS solved the the problem and also when an optimal antiderivative is known.

If it happens that a CAS was not able to solve all the integrals in the input test file, or if it was not possible to obtain leaf size for the CAS result for all the problems in the file, then a zero is used for the ratio and -1 is used for the problem number.

This makes it easy to locate the problem. In the future, a direct link will be added as well.

Table 1.6: Maximum leaf size ratio for each CAS against the optimal result

#	Rubi	Mathematica	Maple	Maxima	FriCAS	Sympy	Giac	Mupad	Mathi
1	1. (1)	3.9 (50)	16.9 (114)	3.8 (169)	4. (45)	7.5 (169)	4.3 (45)	42.4 (169)	8.9 (15
2	7.3 (21)	5. (20)	3.6 (17)	1.9 (4)	14.3 (13)	16.8 (5)	6.5 (2)	3.3 (26)	2.4 (19
3	1. (1)	1.1 (14)	17. (6)	11.1 (7)	2. (8)	1.9 (5)	2.5 (3)	11.3 (5)	1.4 (5)
4	6.4 (5)	14.3 (13)	14.7 (46)	16.6 (43)	5.5 (43)	4.8 (40)	7.1 (8)	6.9 (4)	3.2 (16
5	1. (1)	54.7 (278)	12737.8 (278)	8.1 (280)	7.7 (280)	39.8 (123)	26.9 (141)	14.1 (204)	17.9 (
6	1. (1)	1.4 (3)	2.2 (4)	1.9 (1)	1.4 (7)	0.8 (4)	2.2 (5)	1.3 (3)	1.3 (6)
7	2.2 (3)	5.6 (7)	1.8 (3)	2.8 (3)	6.7 (9)	45.4 (9)	2.6 (3)	1.7 (3)	1.7 (7)
8	1.6 (50)	5.3 (31)	5.1 (40)	6.5 (11)	5. (42)	26.4 (71)	7. (40)	22.5 (70)	7.3 (56
9	1.2 (365)	7.2 (80)	3.7 (296)	12.1 (328)	4.2 (341)	8.1 (75)	16.9 (328)	6. (9)	13.1 (
10	3.2 (335)	93. (452)	3343.5 (327)	36.9 (399)	32.1 (595)	76.3 (215)	24.5 (537)	12.8 (253)	41.9 (
11	529. (82)	127. (82)	317. (82)	2.7 (2)	70. (82)	41.3 (17)	49.1 (50)	207. (82)	15.8 (
12	1.8 (6)	2.3 (4)	1.2 (8)	1.5 (2)	3.3 (3)	3.4 (3)	2.1 (2)	0.9 (8)	5.4 (3)
13	7.1 (369)	23.8 (1323)	30.9 (1323)	32.9 (1323)	32.9 (1323)	136.1 (671)	49.1 (1444)	38.1 (1323)	61.3 (

1.5 Pass/Fail per test file for each CAS system

The following table gives the number of passed integrals and number of failed integrals per test number. There are 210 tests. Each tests corresponds to one input file.

Table 1.7: Pass/Fail per test file for each CAS

#	Rubi		MMA		Maple		Maxima		FriCAS		Sympy		Giac		Mupad		Mathics	
	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail	Pass	Fail
1	175	0	175	0	173	2	166	9	172	3	158	17	170	5	169	6	156	19
2	33	2	35	0	28	7	15	20	24	11	7	28	17	18	9	26	6	29
3	13	1	14	0	12	2	8	6	12	2	9	5	10	4	11	3	10	4
4	48	2	50	0	33	17	24	26	48	2	19	31	42	8	12	38	15	35
5	279	5	284	0	282	2	252	32	281	3	253	31	268	16	270	14	248	36
6	3	4	7	0	5	2	3	4	7	0	5	2	5	2	7	0	5	2
7	7	2	9	0	9	0	7	2	9	0	5	4	9	0	9	0	3	6
8	113	0	113	0	113	0	111	2	112	1	105	8	111	2	106	7	104	9
9	376	0	376	0	376	0	374	2	376	0	345	31	375	1	372	4	349	27
10	705	0	705	0	655	50	564	141	653	52	436	269	590	115	542	163	426	279
11	113	3	101	15	79	37	20	96	91	25	29	87	34	82	37	79	28	88
12	8	0	8	0	8	0	7	1	8	0	8	0	8	0	8	0	8	0
13	1917	0	1917	0	1565	352	1328	589	1603	314	1199	718	1386	531	1241	676	1197	720

1.6 Timing

The command `AbsoluteTiming[]` was used in Mathematica to obtain the elapsed time for each integrate call. In Maple, the command `Usage` was used as in the following example

```
cpu_time := Usage(assign ('result_of_int',int(expr,x)),output='realtime'
```

For all other CAS systems, the elapsed time to complete each integral was found by taking the difference between the time after the call completed from the time before the call was made. This was done using Python's `time.time()` call.

All elapsed times shown are in seconds. A time limit of 3 CPU minutes was used for each integral. If the integrate command did not complete within this time limit, the integral was aborted and considered to have failed and assigned an F grade. The time used by failed integrals due to time out was not counted in the final statistics.

1.7 Verification

A verification phase was applied on the result of integration for `Rubi`, `Mathematica` and `Mathics`. Future version of this report will implement verification for the other CAS systems. For the integrals whose result was not run through a verification phase, it is assumed that the antiderivative was correct.

Verification phase also had 3 minutes time out. An integral whose result was not verified could still be correct, but further investigation is needed on those integrals. These integrals were marked in the summary table below and also in each integral separate section so they are easy to identify and locate.

1.8 Important notes about some of the results

1.8.1 Important note about Maxima results

Since tests were run in a batch mode, and using an automated script, then any integral where Maxima needed an interactive response from the user to answer a question during the evaluation of the integral will fail.

The exception raised is `ValueError`. Therefore Maxima results is lower than what would result if Maxima was run directly and each question was answered correctly.

The percentage of such failures were not counted for each test file, but for an example, for the **Timofeev** test file, there were about 14 such integrals out of total 705, or about 2 percent. This percentage can be higher or lower depending on the specific input test file.

Such integrals can be identified by looking at the output of the integration in each section for Maxima. The exception message will indicate the cause of error.

Maxima integrate was run using SageMath with the following settings set by default

```
'besselexpand : true'
'display2d : false'
'domain : complex'
'keepfloat : true'
'load(to_poly_solve)'
'load(simplify_sum)'
'load(abs_integrate)' 'load(diag)'
```

SageMath automatic loading of Maxima `abs_integrate` was found to cause some problems. So the following code was added to disable this effect.

```
from sage.interfaces.maxima_lib import maxima_lib
maxima_lib.set('extra_definite_integration_methods', '[]')
maxima_lib.set('extra_integration_methods', '[]')
```

See `https://ask.sagemath.org/question/43088/integrate-results-that-are-different-from-using-maxima/` for reference.

1.8.2 Important note about FriCAS result

There were few integrals which failed due to SageMath interface and not because FriCAS system could not do the integration.

These will fail With error `Exception raised: NotImplementedError`.

The number of such cases seems to be very small. About 1 or 2 percent of all integrals. These can be identified by looking at the exception message given in the result.

1.8.3 Important note about finding leaf size of antiderivative

For Mathematica, Rubi, Mathics and Maple, the builtin system function `LeafSize` was used to find the leaf size of each antiderivative.

The other CAS systems (SageMath and Sympy) do not have special builtin function for this purpose at this time. Therefore the leaf size for Fricas and Sympy antiderivative was determined using the following function, thanks to user `slelievre` at `https://ask.sagemath.org/question/57123/could-we-have-a-leaf_count-function-in-base-sagemath/`

```
def tree_size(expr):
    r"""
    Return the tree size of this expression.
    """
    if expr not in SR:
        # deal with lists, tuples, vectors
        return 1 + sum(tree_size(a) for a in expr)
    expr = SR(expr)
    x, aa = expr.operator(), expr.operands()
    if x is None:
        return 1
    else:
        return 1 + sum(tree_size(a) for a in aa)
```

For Sympy, which was called directly from Python, the following code was used to obtain the leafsize of its result

```
try:
  # 1.7 is a fudge factor since it is low side from actual leaf count
  leafCount = round(1.7*count_ops(anti))

  except Exception as ee:
       leafCount =1
```

For Giac, the call `taille(anti_derivative,RAND_MAX);` is used to find leaf size.

1.8.4 Important note about Mupad results

Matlab's symbolic toolbox does not have a leaf count function to measure the size of the antiderivative. Maple was used to determine the leaf size of Mupad output by post processing Mupad result.

Currently no grading of the antiderivative for Mupad is implemented. If it can integrate the problem, it was assigned a B grade automatically as a placeholder. In the future, when grading function is implemented for Mupad, the tests will be rerun again.

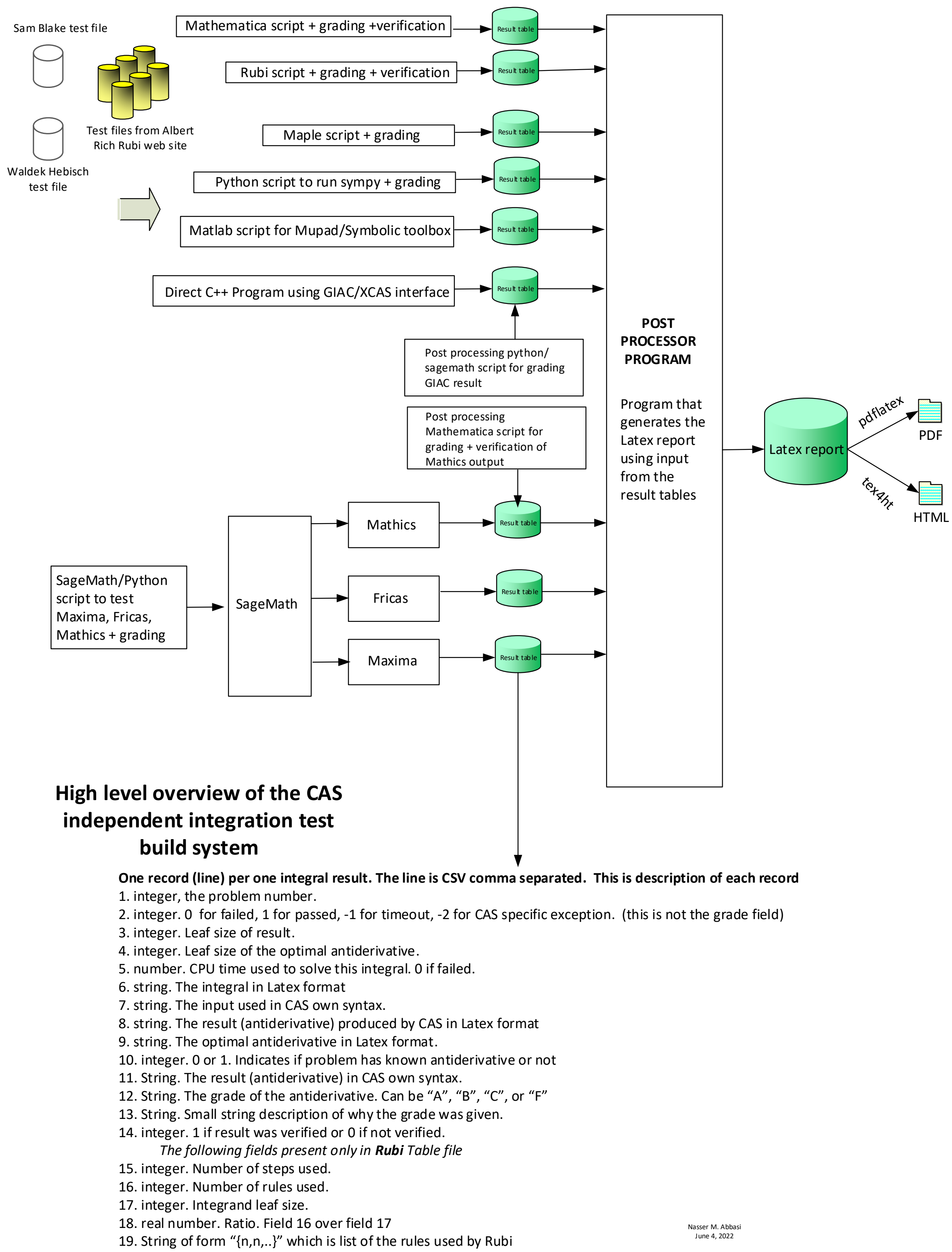
The following is an example of using Matlab's symbolic toolbox (Mupad) to solve an integral

```
integrand = evalin(symengine,'cos(x)*sin(x)')
the_variable = evalin(symengine,'x')
anti = int(integrand,the_variable)
```

Which gives `sin(x)^2/2`

1.9 Design of the test system

The following diagram gives a high level view of the current test build system.



Chapter 2

links to individual test reports

These are links to each test report. The number in square brackets to right of the link is the number of integrals in the test. The list of numbers in the curly brackets after that (if any) is the list of the integrals in that specific test which were solved by any CAS which are known not to have antiderivative. This makes it easier to find these integrals and do more investigation into them.

2.1 Tests completed

1. test_cases/0_Independent_test_suites/1_Apostol_Problems/report.htm [175]
2. test_cases/0_Independent_test_suites/2_Bondarenko_Problems/report.htm [35]
3. test_cases/0_Independent_test_suites/3_Bronstein_Problems/report.htm [14]
4. test_cases/0_Independent_test_suites/4_Charlwood_Problems/report.htm [50]
5. test_cases/0_Independent_test_suites/5_Hearn_Problems/report.htm [284] { **Maxima: 145.** }
6. test_cases/0_Independent_test_suites/6_Hebisch_Problems/report.htm [7]
7. test_cases/0_Independent_test_suites/7_Jeffrey_Problems/report.htm [9]
8. test_cases/0_Independent_test_suites/8_Moses_Problems/report.htm [113]
9. test_cases/0_Independent_test_suites/9_Stewart_Problems/report.htm [376]
10. test_cases/0_Independent_test_suites/10_Timofeev_Problems/report.htm [705]
11. test_cases/0_Independent_test_suites/11_Welz_Problems/report.htm [116]
12. test_cases/0_Independent_test_suites/12_Wester_Problems/report.htm [8]
13. test_cases/1_Algebraic_functions/1.1_Binomial_products/1.1.1_Linear/13_1.1.1.2-a+b_x-^{m}-c+d_x-^{n}/report.htm [1917]

Chapter 3

Listing of integrals solved by CAS which has no known antiderivatives

3.1 Test file Number [5]

3.1.1 Maxima

Integral number [145]

$$\int x \cos(k \csc(x)) \cot(x) \csc(x)\, dx$$

[B] time = 0.258088 (sec), size = 240 ,normalized size = 17.14

$$-\frac{\left(xe^{\left(\frac{4\,k\cos(2\,x)\cos(x)}{\cos(2\,x)^2+\sin(2\,x)^2-2\,\cos(2\,x)+1}+\frac{4\,k\sin(2\,x)\sin(x)}{\cos(2\,x)^2+\sin(2\,x)^2-2\,\cos(2\,x)+1}\right)}+xe^{\left(\frac{4\,k\cos(x)}{\cos(2\,x)^2+\sin(2\,x)^2-2\,\cos(2\,x)+1}\right)}\right)e^{\left(-\frac{2\,k\cos(2\,x)\cos(x)}{\cos(2\,x)^2+\sin(2\,x)^2-2\,\cos(2\cdots}\right.}}{2\,k}$$

```
[In] integrate(x*cos(x)*cos(k/sin(x))/sin(x)^2,x, algorithm=""maxima"")
```

```
[Out] -1/2*(x*e^(4*k*cos(2*x)*cos(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1) +
 4*k*sin(2*x)*sin(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1)) + x*e^(4*k
*cos(x)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1)))*e^(-2*k*cos(2*x)*cos(x
)/(cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1) - 2*k*sin(2*x)*sin(x)/(cos(2*x
)^2 + sin(2*x)^2 - 2*cos(2*x) + 1) - 2*k*cos(x)/(cos(2*x)^2 + sin(2*x)^2 -
2*cos(2*x) + 1))*sin(2*(k*cos(x)*sin(2*x) - k*cos(2*x)*sin(x) + k*sin(x))/(
cos(2*x)^2 + sin(2*x)^2 - 2*cos(2*x) + 1))/k
```

Chapter 4

Appendix

Local contents

4.1 Listing of grading functions

The following are the current version of the grading functions used for grading the quality of the antiderivative with reference to the optimal antiderivative included in the test suite.

There is a version for Maple and for Mathematica/Rubi. There is a version for grading Sympy and version for use with Sagemath.

The following are links to the current source code.

The following are the listings of source code of the grading functions.

4.1.1 Mathematica and Rubi grading function

```
(* Original version thanks to Albert Rich emailed on 03/21/2017 *)
(* ::Package:: *)

(* Nasser: April 7,2022. add second output which gives reason for the grade *)
(*                        Small rewrite of logic in main function to make it*)
(*                        match Maple's logic. No change in functionality otherwise*)

(* ::Subsection:: *)
(*GradeAntiderivative[result,optimal]*)


(* ::Text:: *)
(*If result and optimal are mathematical expressions, *)
(*          GradeAntiderivative[result,optimal] returns*)
(*  "F" if the result fails to integrate an expression that*)
(*      is integrable*)
(*  "C" if result involves higher level functions than necessary*)
(*  "B" if result is more than twice the size of the optimal*)
(*      antiderivative*)
(*  "A" if result can be considered optimal*)


GradeAntiderivative[result_,optimal_] := Module[{expnResult,expnOptimal,leafCountResult,leafCountOptimal
    expnResult  = ExpnType[result];
    expnOptimal = ExpnType[optimal];
    leafCountResult = LeafCount[result];
    leafCountOptimal = LeafCount[optimal];

    (*Print["expnResult=",expnResult," expnOptimal=",expnOptimal];*)
    If[expnResult<=expnOptimal,
        If[Not[FreeQ[result,Complex]], (*result contains complex*)
            If[Not[FreeQ[optimal,Complex]], (*optimal contains complex*)
                If[leafCountResult<=2*leafCountOptimal,
                    finalresult={"A","none"}
                    ,(*ELSE*)
                    finalresult={"B","Both result and optimal contain complex but leaf count is larger t
                ]
            ,(*ELSE*)
                finalresult={"C","Result contains complex when optimal does not."}
            ]
```

```
        ,(*ELSE*)(*result does not contains complex*)
            If[leafCountResult<=2*leafCountOptimal,
               finalresult={"A","none"}
            ,(*ELSE*)
               finalresult={"B","Leaf count is larger than twice the leaf count of optimal. $"<>ToStri
              ]
        ]
    ,(*ELSE*) (*expnResult>expnOptimal*)
        If[FreeQ[result,Integrate] && FreeQ[result,Int],
            finalresult={"C","Result contains higher order function than in optimal. Order "<>ToString
            ,
            finalresult={"F","Contains unresolved integral."}
        ]
    ];

    finalresult
]

(* ::Text:: *)
(*The following summarizes the type number assigned an *)
(*expression based on the functions it involves*)
(*1 = rational function*)
(*2 = algebraic function*)
(*3 = elementary function*)
(*4 = special function*)
(*5 = hyperpergeometric function*)
(*6 = appell function*)
(*7 = rootsum function*)
(*8 = integrate function*)
(*9 = unknown function*)


ExpnType[expn_] :=
  If[AtomQ[expn],
    1,
  If[ListQ[expn],
    Max[Map[ExpnType,expn]],
  If[Head[expn]===Power,
    If[IntegerQ[expn[[2]]],
      ExpnType[expn[[1]]],
    If[Head[expn[[2]]]===Rational,
      If[IntegerQ[expn[[1]]] || Head[expn[[1]]]===Rational,
        1,
      Max[ExpnType[expn[[1]]],2]],
    Max[ExpnType[expn[[1]]],ExpnType[expn[[2]]],3]]],
  If[Head[expn]===Plus || Head[expn]===Times,
    Max[ExpnType[First[expn]],ExpnType[Rest[expn]]],
  If[ElementaryFunctionQ[Head[expn]],
    Max[3,ExpnType[expn[[1]]]],
  If[SpecialFunctionQ[Head[expn]],
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],4]],
  If[HypergeometricFunctionQ[Head[expn]],
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],5]],
  If[AppellFunctionQ[Head[expn]],
```

```
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],6]],
  If[Head[expn]===RootSum,
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],7]],
  If[Head[expn]===Integrate || Head[expn]===Int,
    Apply[Max,Append[Map[ExpnType,Apply[List,expn]],8]],
  9]]]]]]]]]


ElementaryFunctionQ[func_] :=
  MemberQ[{
  Exp,Log,
  Sin,Cos,Tan,Cot,Sec,Csc,
  ArcSin,ArcCos,ArcTan,ArcCot,ArcSec,ArcCsc,
  Sinh,Cosh,Tanh,Coth,Sech,Csch,
  ArcSinh,ArcCosh,ArcTanh,ArcCoth,ArcSech,ArcCsch
},func]


SpecialFunctionQ[func_] :=
  MemberQ[{
  Erf, Erfc, Erfi,
  FresnelS, FresnelC,
  ExpIntegralE, ExpIntegralEi, LogIntegral,
  SinIntegral, CosIntegral, SinhIntegral, CoshIntegral,
  Gamma, LogGamma, PolyGamma,
  Zeta, PolyLog, ProductLog,
  EllipticF, EllipticE, EllipticPi
},func]


HypergeometricFunctionQ[func_] :=
  MemberQ[{Hypergeometric1F1,Hypergeometric2F1,HypergeometricPFQ},func]


AppellFunctionQ[func_] :=
  MemberQ[{AppellF1},func]
```

4.1.2 Maple grading function

```
# File: GradeAntiderivative.mpl
# Original version thanks to Albert Rich emailed on 03/21/2017

#Nasser 03/22/2017  Use Maple leaf count instead since buildin
#Nasser 03/23/2017  missing 'ln' for ElementaryFunctionQ added
#Nasser 03/24/2017  corrected the check for complex result
#Nasser 10/27/2017  check for leafsize and do not call ExpnType()
#                   if leaf size is "too large". Set at 500,000
#Nasser 12/22/2019  Added debug flag, added 'dilog' to special functions
#                   see problem 156, file Apostol_Problems
#Nasser 4/07/2022   add second output which gives reason for the grade

GradeAntiderivative := proc(result,optimal)
```

```
local leaf_count_result,
        leaf_count_optimal,
        ExpnType_result,
        ExpnType_optimal,
        debug:=false;

        leaf_count_result:=leafcount(result);
        #do NOT call ExpnType() if leaf size is too large. Recursion problem
        if leaf_count_result > 500000 then
            return "B","result has leaf size over 500,000. Avoiding possible recursion issues.";
        fi;

        leaf_count_optimal := leafcount(optimal);
        ExpnType_result  := ExpnType(result);
        ExpnType_optimal := ExpnType(optimal);

        if debug then
            print("ExpnType_result",ExpnType_result," ExpnType_optimal=",ExpnType_optimal);
        fi;

# If result and optimal are mathematical expressions,
#  GradeAntiderivative[result,optimal] returns
#   "F" if the result fails to integrate an expression that
#       is integrable
#   "C" if result involves higher level functions than necessary
#   "B" if result is more than twice the size of the optimal
#       antiderivative
#   "A" if result can be considered optimal

    #This check below actually is not needed, since I only
    #call this grading only for passed integrals. i.e. I check
    #for "F" before calling this. But no harm of keeping it here.
    #just in case.


    if not type(result,freeof('int')) then
        return "F","Result contains unresolved integral";
    fi;


    if ExpnType_result<=ExpnType_optimal then
        if debug then
             print("ExpnType_result<=ExpnType_optimal");
        fi;
        if is_contains_complex(result) then
            if is_contains_complex(optimal) then
                if debug then
                       print("both result and optimal complex");
                fi;
                if leaf_count_result<=2*leaf_count_optimal then
                    return "A","";
                else
                   return "B",cat("Both result and optimal contain complex but leaf count of result is
                                convert(leaf_count_result,string)," vs. $2 (",
```

```
                                   convert(leaf_count_optimal,string)," ) = ",convert(2*leaf_count_opti
                end if
            else #result contains complex but optimal is not
                if debug then
                        print("result contains complex but optimal is not");
                fi;
                return "C","Result contains complex when optimal does not.";
            fi;
        else # result do not contain complex
             # this assumes optimal do not as well. No check is needed here.
            if debug then
                   print("result do not contain complex, this assumes optimal do not as well");
            fi;
            if leaf_count_result<=2*leaf_count_optimal then
                if debug then
                    print("leaf_count_result<=2*leaf_count_optimal");
                fi;
                return "A","";
            else
                if debug then
                    print("leaf_count_result>2*leaf_count_optimal");
                fi;
                return "B",cat("Leaf count of result is larger than twice the leaf count of optimal. $"
                                   convert(leaf_count_result,string),"$ vs. $2(",
                                   convert(leaf_count_optimal,string),")=",convert(2*leaf_count_optimal
            fi;
        fi;
    else #ExpnType(result) > ExpnType(optimal)
        if debug then
            print("ExpnType(result) > ExpnType(optimal)");
        fi;
        return "C",cat("Result contains higher order function than in optimal. Order ",
                       convert(ExpnType_result,string)," vs. order ",
                       convert(ExpnType_optimal,string),".");
    fi;

end proc:

#
# is_contains_complex(result)
# takes expressions and returns true if it contains "I" else false
#
#Nasser 032417
is_contains_complex:= proc(expression)
  return (has(expression,I));
end proc:

# The following summarizes the type number assigned an expression
# based on the functions it involves
# 1 = rational function
# 2 = algebraic function
# 3 = elementary function
# 4 = special function
# 5 = hyperpergeometric function
```

```
# 6 = appell function
# 7 = rootsum function
# 8 = integrate function
# 9 = unknown function

ExpnType := proc(expn)
  if type(expn,'atomic') then
    1
  elif type(expn,'list') then
    apply(max,map(ExpnType,expn))
  elif type(expn,'sqrt') then
    if type(op(1,expn),'rational') then
       1
    else
       max(2,ExpnType(op(1,expn)))
    end if
  elif type(expn,'`^`') then
    if type(op(2,expn),'integer') then
      ExpnType(op(1,expn))
    elif type(op(2,expn),'rational') then
      if type(op(1,expn),'rational') then
         1
      else
         max(2,ExpnType(op(1,expn)))
      end if
    else
         max(3,ExpnType(op(1,expn)),ExpnType(op(2,expn)))
    end if
  elif type(expn,'`+`') or type(expn,'`*`') then
    max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
  elif ElementaryFunctionQ(op(0,expn)) then
    max(3,ExpnType(op(1,expn)))
  elif SpecialFunctionQ(op(0,expn)) then
    max(4,apply(max,map(ExpnType,[op(expn)])))
  elif HypergeometricFunctionQ(op(0,expn)) then
    max(5,apply(max,map(ExpnType,[op(expn)])))
  elif AppellFunctionQ(op(0,expn)) then
    max(6,apply(max,map(ExpnType,[op(expn)])))
  elif op(0,expn)='int' then
    max(8,apply(max,map(ExpnType,[op(expn)]))) else
  9
  end if
end proc:


ElementaryFunctionQ := proc(func)
  member(func,[
        exp,log,ln,
        sin,cos,tan,cot,sec,csc,
        arcsin,arccos,arctan,arccot,arcsec,arccsc,
        sinh,cosh,tanh,coth,sech,csch,
        arcsinh,arccosh,arctanh,arccoth,arcsech,arccsch])
end proc:
```

```
SpecialFunctionQ := proc(func)
  member(func,[
        erf,erfc,erfi,
        FresnelS,FresnelC,
        Ei,Ei,Li,Si,Ci,Shi,Chi,
        GAMMA,lnGAMMA,Psi,Zeta,polylog,dilog,LambertW,
        EllipticF,EllipticE,EllipticPi])
end proc:

HypergeometricFunctionQ := proc(func)
  member(func,[Hypergeometric1F1,hypergeom,HypergeometricPFQ])
end proc:

AppellFunctionQ := proc(func)
  member(func,[AppellF1])
end proc:

# u is a sum or product.  rest(u) returns all but the
# first term or factor of u.
rest := proc(u) local v;
  if nops(u)=2 then
     op(2,u)
  else
     apply(op(0,u),op(2..nops(u),u))
  end if
end proc:

#leafcount(u) returns the number of nodes in u.
#Nasser 3/23/17  Replaced by build-in leafCount from package in Maple
leafcount := proc(u)
   MmaTranslator[Mma][LeafCount](u);
end proc:
```

4.1.3 Sympy grading function

```
#Dec 24, 2019. Nasser M. Abbasi:
#              Port of original Maple grading function by
#              Albert Rich to use with Sympy/Python
#Dec 27, 2019  Nasser. Added `RootSum`. See problem 177, Timofeev file
#              added 'exp_polar'
from sympy import *

def leaf_count(expr):
   #sympy do not have leaf count function. This is approximation
   return round(1.7*count_ops(expr))

def is_sqrt(expr):
   if isinstance(expr,Pow):
      if expr.args[1] == Rational(1,2):
         return True
      else:
         return False
```

```
  else:
    return False

def is_elementary_function(func):
  return func in [exp,log,ln,sin,cos,tan,cot,sec,csc,
          asin,acos,atan,acot,asec,acsc,sinh,cosh,tanh,coth,sech,csch,
          asinh,acosh,atanh,acoth,asech,acsch
      ]

def is_special_function(func):
  return func in [ erf,erfc,erfi,
           fresnels,fresnelc,Ei,Ei,Li,Si,Ci,Shi,Chi,
           gamma,loggamma,digamma,zeta,polylog,LambertW,
           elliptic_f,elliptic_e,elliptic_pi,exp_polar
       ]

def is_hypergeometric_function(func):
  return func in [hyper]

def is_appell_function(func):
  return func in [appellf1]

def is_atom(expn):
  try:
    if expn.isAtom or isinstance(expn,int) or isinstance(expn,float):
      return True
    else:
      return False

  except AttributeError as error:
    return False

def expnType(expn):
  debug=False
  if debug:
    print("expn=",expn,"type(expn)=",type(expn))

  if is_atom(expn):
    return 1
  elif isinstance(expn,list):
    return max(map(expnType, expn))   #apply(max,map(ExpnType,expn))
  elif  is_sqrt(expn):
    if isinstance(expn.args[0],Rational): #type(op(1,expn),'rational')
      return 1
    else:
      return max(2,expnType(expn.args[0]))  #max(2,ExpnType(op(1,expn)))
  elif isinstance(expn,Pow):   #type(expn,'`^`')
    if isinstance(expn.args[1],Integer):  #type(op(2,expn),'integer')
      return expnType(expn.args[0])   #ExpnType(op(1,expn))
    elif isinstance(expn.args[1],Rational):  #type(op(2,expn),'rational')
      if isinstance(expn.args[0],Rational): #type(op(1,expn),'rational')
        return 1
      else:
        return max(2,expnType(expn.args[0]))  #max(2,ExpnType(op(1,expn)))
```

```
        else:
            return max(3,expnType(expn.args[0]),expnType(expn.args[1])) #max(3,ExpnType(op(1,expn)),ExpnType(op(2,e
    elif isinstance(expn,Add) or isinstance(expn,Mul): #type(expn,''+'') or type(expn,''*'')
        m1 = expnType(expn.args[0])
        m2 = expnType(list(expn.args[1:]))
        return max(m1,m2)  #max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
    elif is_elementary_function(expn.func): #ElementaryFunctionQ(op(0,expn))
        return max(3,expnType(expn.args[0]))  #max(3,ExpnType(op(1,expn)))
    elif is_special_function(expn.func): #SpecialFunctionQ(op(0,expn))
        m1 = max(map(expnType, list(expn.args)))
        return max(4,m1)  #max(4,apply(max,map(ExpnType,[op(expn)])))
    elif is_hypergeometric_function(expn.func): #HypergeometricFunctionQ(op(0,expn))
        m1 = max(map(expnType, list(expn.args)))
        return max(5,m1)  #max(5,apply(max,map(ExpnType,[op(expn)])))
    elif is_appell_function(expn.func):
        m1 = max(map(expnType, list(expn.args)))
        return max(6,m1)  #max(5,apply(max,map(ExpnType,[op(expn)])))
    elif isinstance(expn,RootSum):
        m1 = max(map(expnType, list(expn.args))) #Apply[Max,Append[Map[ExpnType,Apply[List,expn]],7]],
        return max(7,m1)
    elif str(expn).find("Integral") != -1:
        m1 = max(map(expnType, list(expn.args)))
        return max(8,m1)  #max(5,apply(max,map(ExpnType,[op(expn)])))
    else:
        return 9

#main function
def grade_antiderivative(result,optimal):

    #print ("Enter grade_antiderivative for sagemath")
    #print("Enter grade_antiderivative, result=",result," optimal=",optimal)

    leaf_count_result  = leaf_count(result)
    leaf_count_optimal = leaf_count(optimal)

    #print("leaf_count_result=",leaf_count_result)
    #print("leaf_count_optimal=",leaf_count_optimal)

    expnType_result  = expnType(result)
    expnType_optimal = expnType(optimal)

    if str(result).find("Integral") != -1:
        grade = "F"
        grade_annotation =""
    else:
        if expnType_result <= expnType_optimal:
            if result.has(I):
                if optimal.has(I): #both result and optimal complex
                    if leaf_count_result <= 2*leaf_count_optimal:
                        grade = "A"
                        grade_annotation =""
                    else:
                        grade = "B"
                        grade_annotation ="Both result and optimal contain complex but leaf count of result is larger than twic
```

```
            else: #result contains complex but optimal is not
                grade = "C"
                grade_annotation ="Result contains complex when optimal does not."
        else: # result do not contain complex, this assumes optimal do not as well
            if leaf_count_result <= 2*leaf_count_optimal:
                grade = "A"
                grade_annotation =""
            else:
                grade = "B"
                grade_annotation ="Leaf count of result is larger than twice the leaf count of optimal. "+str(leaf_count_
    else:
        grade = "C"
        grade_annotation ="Result contains higher order function than in optimal. Order "+str(ExpnType_result)+" 


    #print("Before returning. grade=",grade, " grade_annotation=",grade_annotation)

    return grade, grade_annotation
```

4.1.4 SageMath grading function

```
#Dec 24, 2019. Nasser: Ported original Maple grading function by
#              Albert Rich to use with Sagemath. This is used to
#              grade Fricas, Giac and Maxima results.
#Dec 24, 2019. Nasser: Added 'exp_integral_e' and 'sng', 'sin_integral'
#              'arctan2','floor','abs','log_integral'
#June 4, 2022  Made default grade_annotation "none" instead of "" due
#              issue later when reading the file.

from sage.all import *
from sage.symbolic.operators import add_vararg, mul_vararg

debug=False;

def tree_size(expr):
    r"""
    Return the tree size of this expression.
    """
    #print("Enter tree_size, expr is ",expr)

    if expr not in SR:
        # deal with lists, tuples, vectors
        return 1 + sum(tree_size(a) for a in expr)
    expr = SR(expr)
    x, aa = expr.operator(), expr.operands()
    if x is None:
        return 1
    else:
        return 1 + sum(tree_size(a) for a in aa)

def is_sqrt(expr):
    if expr.operator() == operator.pow:   #isinstance(expr,Pow):
```

```
        if expr.operands()[1]==1/2: #expr.args[1] == Rational(1,2):
            if debug: print ("expr is sqrt")
            return True
        else:
            return False
    else:
        return False

def is_elementary_function(func):
    debug=False
    m = func.name() in ['exp','log','ln',
            'sin','cos','tan','cot','sec','csc',
            'arcsin','arccos','arctan','arccot','arcsec','arccsc',
            'sinh','cosh','tanh','coth','sech','csch',
            'arcsinh','arccosh','arctanh','arccoth','arcsech','arccsch','sgn',
        'arctan2','floor','abs'
        ]
    if debug:
        if m:
            print ("func ", func , " is elementary_function")
        else:
            print ("func ", func , " is NOT elementary_function")


    return m

def is_special_function(func):
    debug=False
    if debug: print ("type(func)=", type(func))

    m= func.name() in ['erf','erfc','erfi','fresnel_sin','fresnel_cos','Ei',
           'Ei','Li','Si','sin_integral','Ci','cos_integral','Shi','sinh_integral'
           'Chi','cosh_integral','gamma','log_gamma','psi,zeta',
           'polylog','lambert_w','elliptic_f','elliptic_e',
           'elliptic_pi','exp_integral_e','log_integral']

    if debug:
        print ("m=",m)
        if m:
            print ("func ", func ," is special_function")
        else:
            print ("func ", func ," is NOT special_function")


    return m


def is_hypergeometric_function(func):
    return func.name() in ['hypergeometric','hypergeometric_M','hypergeometric_U']

def is_appell_function(func):
    return func.name() in ['hypergeometric']   #[appellf1] can't find this in sagemath

def is_atom(expn):
```

```
    debug=False
    if debug: print ("Enter is_atom")


    #thanks to answer at https://ask.sagemath.org/question/49179/what-is-sagemath-equivalent-to-atomic-type-in-m
    try:
       if expn.parent() is SR:
          return expn.operator() is None
       if expn.parent() in (ZZ, QQ, AA, QQbar):
          return expn in expn.parent() # Should always return True
       if hasattr(expn.parent(),"base_ring") and hasattr(expn.parent(),"gens"):
          return expn in expn.parent().base_ring() or expn in expn.parent().gens()
       return False

    except AttributeError as error:
       return False


def expnType(expn):

    if debug:
       print (">>>>>Enter expnType, expn=", expn)
       print (">>>>>is_atom(expn)=", is_atom(expn))

    if is_atom(expn):
       return 1
    elif type(expn)==list:   #isinstance(expn,list):
       return max(map(expnType, expn))   #apply(max,map(ExpnType,expn))
    elif  is_sqrt(expn):
       if  type(expn.operands()[0])==Rational: #type(isinstance(expn.args[0],Rational):
          return 1
       else:
          return max(2,expnType(expn.operands()[0]))  #max(2,expnType(expn.args[0]))
    elif expn.operator() == operator.pow:   #isinstance(expn,Pow)
       if type(expn.operands()[1])==Integer:   #isinstance(expn.args[1],Integer)
          return expnType(expn.operands()[0])   #expnType(expn.args[0])
       elif type(expn.operands()[1])==Rational:   #isinstance(expn.args[1],Rational)
          if type(expn.operands()[0])==Rational: #isinstance(expn.args[0],Rational)
             return 1
          else:
             return max(2,expnType(expn.operands()[0]))  #max(2,expnType(expn.args[0]))
       else:
          return max(3,expnType(expn.operands()[0]),expnType(expn.operands()[1])) #max(3,expnType(expn.operands(
    elif expn.operator() == add_vararg or expn.operator() == mul_vararg: #isinstance(expn,Add) or isinstance(expn,
       m1 = expnType(expn.operands()[0]) #expnType(expn.args[0])
       m2 = expnType(expn.operands()[1:]) #expnType(list(expn.args[1:]))
       return max(m1,m2)  #max(ExpnType(op(1,expn)),max(ExpnType(rest(expn))))
    elif is_elementary_function(expn.operator()):  #is_elementary_function(expn.func)
       return max(3,expnType(expn.operands()[0]))
    elif is_special_function(expn.operator()): #is_special_function(expn.func)
       m1 = max(map(expnType, expn.operands()))      #max(map(expnType, list(expn.args)))
       return max(4,m1)  #max(4,m1)
    elif is_hypergeometric_function(expn.operator()): #is_hypergeometric_function(expn.func)
```

```
        m1 = max(map(expnType, expn.operands()))        #max(map(expnType, list(expn.args)))
        return max(5,m1)    #max(5,m1)
    elif is_appell_function(expn.operator()):
        m1 = max(map(expnType, expn.operands()))        #max(map(expnType, list(expn.args)))
        return max(6,m1)    #max(6,m1)
    elif str(expn).find("Integral") != -1: #this will never happen, since it
                #is checked before calling the grading function that is passed.
                #but kept it here.
        m1 = max(map(expnType, expn.operands()))        #max(map(expnType, list(expn.args)))
        return max(8,m1)    #max(5,apply(max,map(ExpnType,[op(expn)])))
    else:
        return 9

#main function
def grade_antiderivative(result,optimal):

    #print ("Enter grade_antiderivative for sagemath")
    #print("Enter grade_antiderivative, result=",result," optimal=",optimal)

    leaf_count_result  = tree_size(result) #leaf_count(result)
    leaf_count_optimal = tree_size(optimal) #leaf_count(optimal)

    #if debug: print ("leaf_count_result=", leaf_count_result, "leaf_count_optimal=",leaf_count_optimal)


    expnType_result  = expnType(result)
    expnType_optimal = expnType(optimal)

    if debug: print ("expnType_result=", expnType_result, "expnType_optimal=",expnType_optimal)

    if expnType_result <= expnType_optimal:
        if result.has(I):
            if optimal.has(I): #both result and optimal complex
                if leaf_count_result <= 2*leaf_count_optimal:
                    grade = "A"
                    grade_annotation ="none"
                else:
                    grade = "B"
                    grade_annotation ="Both result and optimal contain complex but leaf count of result is larger than twice th
            else: #result contains complex but optimal is not
                grade = "C"
                grade_annotation ="Result contains complex when optimal does not."
        else: # result do not contain complex, this assumes optimal do not as well
            if leaf_count_result <= 2*leaf_count_optimal:
                grade = "A"
                grade_annotation ="none"
            else:
                grade = "B"
                grade_annotation ="Leaf count of result is larger than twice the leaf count of optimal. "+str(leaf_count_resu
    else:
        grade = "C"
        grade_annotation ="Result contains higher order function than in optimal. Order "+str(expnType_result)+" vs. or
```

```
#print("Before returning. grade=",grade, " grade_annotation=",grade_annotation)

return grade, grade_annotation
```