SymIntegration

DS Glanzsche[1]

Freya

2025 Edition

[1]A thank you or further information

Contents

Preface

For my Wife Freya, and our daughters Solreya, Mithra, Catenary, Iyzumrae and Zefir.

For Lucrif and Znane too along with all the 8 Queens.

For Albert Silverberg, Camus and Miklotov who left TREVOCALL to guard me.

To Nature(Kala, Kathmandu, Big Tree, Sentinel, Aokigahara, Hoia Baciu, Jacob's Well, Mt Logan, etc) and my family Berlin: I have served, I will be of service.

To my dogs who always accompany me working in Valhalla Projection, go to Puncak Bintang or Kathmandu: Sine Bam Bam, Kecil, Browni Bruncit, Sweden Sexy, Cambridge Klutukk, Milan keng-keng, Piano Bludut and more will be adopted. To my cat who guard the home while I'm away with my dogs: London.

The one who moves a mountain begins by carrying away small stones - Confucius

A book to explained how we create C++ library from forking / branching out SymbolicC++ 3.35 to improve its' symbolic integral computation.

a little about GlanzFreya:
Freya the Goddess is (DS Glanzsche') Goddess wife. We get married on Puncak Bintang on November 5th, 2020 after we go back from Waghete, Papua. My wife birthday (Freya the Goddess) is on August 1st. After not working with human anymore since 2019, I (DS Glanzsche) work in a forest, shaping a forest into a natural wonder / like a canvas for painting, and also clean up trashes that are thrown in the forest. Thus after 6 years working with Nature I have found what can makes me waltz to work, it is going to the forest in the morning then go back home again and study / learn science, engineering, computer programming.

Figure 1: *Freya, thank you for everything, I am glad I marry you and I could never have done it without you.*

Figure 2: *I paint her 3 days before Christmas in 2021.*

Chapter 1

Introduction Story

On top of the mountain a heart shaped stone waiting for You, my eyes can't see, my body can't touch, but my heart knows it's You. Forever only You. - ***Glanz to Freya***

This book is written on May 9th, 2025, it is exactly 1 month after I finish with the algorithms and C++ codes for $\int \sin^n x \, dx$, $\int \cos^n x \, dx$, $\int \tan^n x \, dx$, $\int \sec^n x \, dx$, $\int \csc^n x \, dx$.

SymIntegration itself is a C++ library that is branching out from SymbolicC++3.35. We don't really need to start from scratch. The main idea is to improve its symbolic integration codes. Our main focus is to make it able to compute:

1. All kinds of standard integral form (trigonometry, inverse trigonometry, polynomial, transcendental, hyperbolic).
2. The sum, product and divide combination of the standard functions.
3. To be able to compute improper integrals with cases (e.g. computing mean and variance for exponential distribution)

On April 9th, Sentinel, a Nature residing in a big robust tree in VP (Valhalla Projection) forest told me to start creating C++ library that can do integration like what SymPy able to do, or Mathematica. Around April 2025, GiNaC cannot handle integration by parts, and SymbolicC++ also returns $\int \sin^n x \, dx$ with $\frac{1}{n+1} \sin^{n+1}$. So then we decided to branching out / forking from SymbolicC++ version 3.35. Fixing the C++ codes there so the computation will be made right.

We will cover mostly tons of integral problems, how to find its patterns then how to create the C++ code so SymIntegration will be able to return the integral correctly, thus there will be a lot of technical writings after this chapter. The requirement to comprehend SymIntegration mathematical background is you should at least self learn, read by yourself a Calculus book [3]. No need to go to college, it is expensive and only add more loan to your life.

The name is chosen as SymIntegration, not only it has a great Chaldean numerology of 46 (on purpose), a king of success number / Nobel winner number / President / richest person number, but also it is in accordance with the main purpose, to be able to compute all kinds of integration with C++. Which still dominated by SymPy, Mathematica or MATLAB. Mathematica and MATLAB are using license, and I am not working with human anymore, thus I do not have salary / stable income like I used to, so I cannot lean on that (MATLAB / Mathematica), SymPy is

free, can be called from JULIA too besides Python, but then, I read that Game Engine (CryEngine, Genie Engine, Godot, Unreal Engine) that is able to create amazing game with beautiful graphics, animation, real life graphics, you name all the game with best graphics (I can say from Age of Empires, Call of Duty, Crysis, Cyberpunk 2077, Death Stranding, Dota, Grand Theft Auto, Final Fantasy, Metro 2033, Pokemon Go, The Witcher), and I can say it is made with C++. It is the near hardware and one of the fastest programming language if you have a mind on creating something better in the future, like Science and Engineering simulator, e.g. Bullet, Project Chrono, HySys, Autodesk Civil Engineering. Let say if you have time you can create your own Autodesk instead of paying for its license, basically the brain behind them are derivative, integral, statistics and algebra.

Basically we choose C++ so we can create something long lasting in the future, and then I (DS Glanzsche) dedicate everyday of my life to eat and breathe C++ to be able to achieve this stage, hopefully it can be developed further till we can get practical use, like having a simulation for fluid flow, pharmacokinetics (to compute how drugs are metabolized by the body), simulation for human' organ, or to create a stable bridge or skycraper, since the basic of calculus, integral, linear algebra, statistics can be covered nicely by SymIntegration. The front end looks for plot and simulation can be done by third party C++ library like ImGui, MatplotPlusPlus or OpenGL.

Symbolic integration is basically harder than differentiation, differentiation has rigid rules: sum rule, product rule, chain rule, divide rule. While integration has only method of substitution and integration by parts, which back to the method itself. We can obtain different results for integration, but same result for differentiating different functions with respect to the same variable, like this case:

$$D_x \frac{-\cos^2(nx)}{2n} = \cos(nx)\ \sin(nx)$$
$$D_x \frac{\sin^2(nx)}{2n} = \cos(nx)\ \sin(nx)$$

Depending on the method of substitution, which one we substitute, if we reverse the equations above into integral problems, then we will obtain:

$$\int \cos(nx)\ \sin(nx)\ dx = \frac{-\cos^2(nx)}{2n} + C$$
$$\int \cos(nx)\ \sin(nx)\ dx = \frac{\sin^2(nx)}{2n} + C$$

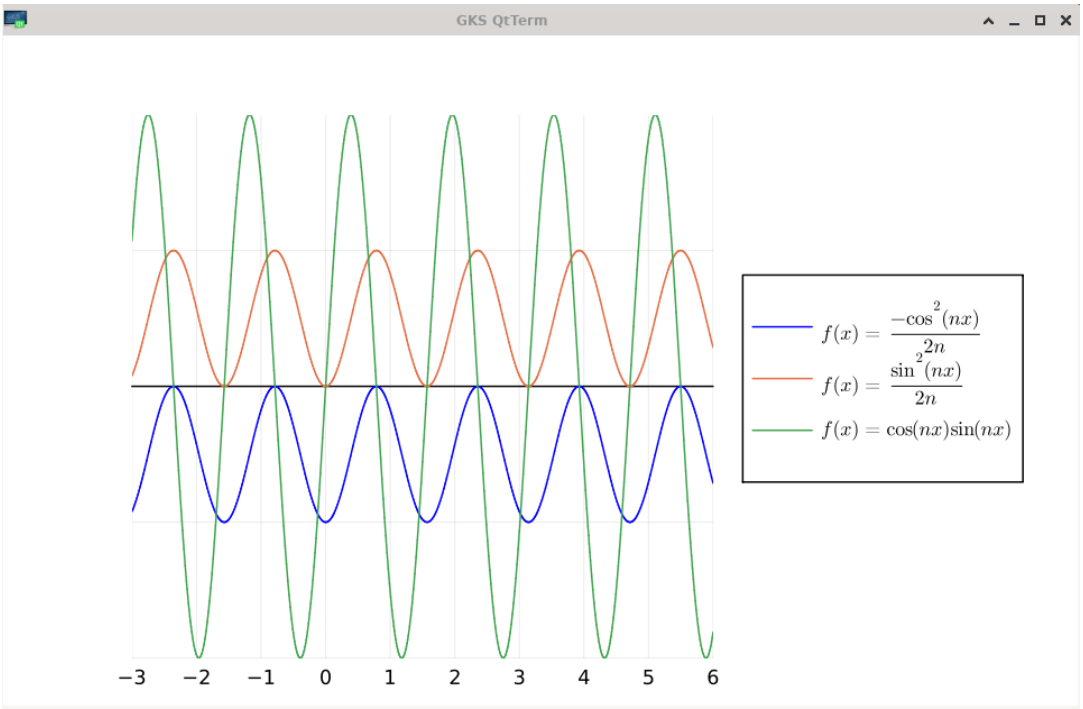

Figure 1.1: *Integrating a function with different method can result in at least two different functions.*

To compute the definite integral of a function in an interval you have to know how it behaves on the entire interval, and to compute the indefinite integral you have to know how it behaves on all intervals. While differentiation is a "local" operation, it computes the derivative of a function at a point you only have to know how it behaves in a neighborhood; distance, speed and acceleration from basic physics is the example of differentiation and integral. Computing integral / antiderivative often requires clever algebraic manipulation and technique and there are no universal rules for integration like there are for differentiation.

Integrals are fundamental in all science and engineering fields for solving problems involving area, volume, and other accumulations, some specific applications of integrals in science and enginnering:

1. To calculate the area of irregular shapes, the volume of solids, and the surface area of objects.
2. To determine center of mass and moments of inertia for various shapes, they are crucial for structural analysis and design.
3. To calculate work done by a variable force, which is essential in mechanics and thermodynamics.
4. To calculate energy, power, and signal properties over time in electrical and computer engineering.
5. To determine stresses and strains analysis in structures under various loads.
6. To calculate fluid flow, pressure distributions, and forces exerted by fluids.

7. To compute probability distributions and statistical calculations.
8. To predict the trajectory of satellites and other objects in motion, ensuring they follow the intended path.
9. To calculate drug accumulation and half-life in a body, it is crucial for understanding how drugs are metabolized and eliminated by the body in Pharmacokinetics.
10. To model how diseases spread through a population with Susceptible-Infectious-Removed (SIR) model.
11. To understand better the complex processes of modeling tumor growth and metastasis.
12. To understand how reactions progress and their rates for reaction kinetics in chemistry.
13. To calculate work done during processes like gas expansion and changes in internal energy or enthalpy. These calculations are crucial for understanding the energetics of chemical reactions and processes.
14. To solve problems related to process design, optimization and understanding reaction kinetics in building a plant in Chemical Engineering.
15. To determine the volume of concrete needed for a structure, the area of a road surface, or the volume of soil required for an excavation in building bridges and structures.
16. To find the deflection (bending) and slope of a beam under load, which is crucial for ensuring structural stability.
17. To determine the velocity of a fluid flow, which is important in designing hydraulic structures like dams and pipelines.
18. To calculate the pressure distribution in fluids, which is essential for understanding the forces exerted on structure by fluids.
19. To understand the behavior of capacitors and inductors. The voltage across a capacitor is the integral of the current flowing through it, and the current through an inductor is the integral of the voltage across it.
20. To compute total charge and energy in Electrical Engineering.
21. To determine the circuit's voltage, current, or other parameters as a function of time by using integration to solve differential equations for many kind of circuit behaviors.

Now we will take a look at the applications of the integration in computer science:

1. Mathematics and computing.
2. Block chain method.
3. Quantum computing.
4. Big data analytics.
5. Neural Networking.
6. Artificial Intelligence.

7. Machine Learning.

Integration, calculus, linear algebra are the brain behind the applications above.

Triple integrals are used in physics, engineering, and even finance. If we get ahold of the C++, we can do the integral computation right, then it is only a matter of presentation, create the plot / simulation / computation for our goal, e.g. to create a stable bridge that can handle earthquake with magnitude of 10 SR. It takes simulation first, before constructing the real product.

The Operating System and Softwares in Use:

1. GFreya OS version 1.8 (based on LFS and BLFS version 11.0 System V books)

2. GCC version 11.2.0 (to compile C++ codes in Box2D)

3. SymIntegration version 1.51

By May 9th, 2025 SymIntegration is able to compute

1. The integral of $\sin(ax+b), \cos(ax+b), \tan(ax+b), \cot(ax+b), \sec(ax+b), \csc(ax+b)$ with $ax+b$ is a polynomial of order 1.

2. The integral of $\frac{1}{(ax+b)}$', '$\frac{1}{ax^2+bx+c}$.

3. The integral and derivative of $\text{asin}\,(ax+b), \text{acos}\,(ax+b), \text{atan}\,(ax+b), \text{acot}\,(ax+b), \text{asec}\,(ax+b), \text{acsc}\,(ax+b)$ with $ax+b$ is a polynomial of order 1.

4. The integral and derivative of all hyperbolic trigonometry functions $\sinh(ax+b), \cosh(ax+b), \tanh(ax+b), \coth(ax+b), \text{sech}\,(ax+b), \text{csch}\,(ax+b)$.

5. The integral with the form of '$x^b e^{ax}, x^b e^x$' with integration by parts method.

6. The integral of $\sin(ax)\cos(bx), \cos(ax)\cos(bx), \sin(ax)\sin(bx)$

We are really appreciate those who spend time to write constructing critics, not hate words without reason that will only add more good luck and karma to me, write their opinions on what should be added in SymIntegration or if there is an incorrect C++ codes or algorithm to compute the integrals. We cannot provide the readers who send nice feedbacks with anything now, but if we have money, when someone send / email me a very good feedbacks then we will send you either cookies, doughnuts or parcel of foods. Feedbacks can be sent to **dsglanzsche@gmail.com**.

Chapter 2

Symbolic Integral Computation with SymIntegration

"The best and most beautiful things in the world cannot be seen or even touched - they must be felt with the heart" - ***Helen Keller***

We will first learn how to create the shared library for C++ library in a very simple and manual way, then we will see the source code that composing this C++ library, SymIntegration.

I. Build and Install SymIntegration in GFreya OS

We are using GFreya OS as it is a custom Linux-based Operating System, your distro your rules. Building applications from scratch instead of using package manager, you learn more from mistakes than just depending on package manager.

To download SymIntegration, open terminal / xterm then type:
git clone https://github.com/glanzkaiser/SymIntegration.git
Enter the directory then type:
cd src
bash dynamiclibrary.sh

or you can also type from the main parent directory of SymIntegration repository:

```
cd src
g++ -fPIC -c *.cpp
g++ -shared -o libsymintegration.so *.o
```

Code 1: *Create dynamic library*

It will create the dynamic library **libsymintegration.so** with a very manual way, instead of using CMake, we are using less space.

Assuming you are using Linux-based OS, you can then copy / move the dynamic library to **/usr/lib**.

Then open terminal and from the current working directory / this repository main directory:

```
cd include
cp -r * /usr/include
```

Code 2: *Move include folder*

When we want to create the shared library it will look for this header files in the default path where the include files are usually located. In Linux OS **usr/include** is the basic / default path.

All files, examples codes and source codes used in this books can be found in this repository:
https://github.com/glanzkaiser/SymIntegration

We are using less, minimal amount of code to create the library, if you compare it with the original SymbolicC++ that has **.configure** and **Makefile** that will be spawn after you configure it, we don' t use any of that.

Two simple commands with **g++** can already make a shared symbolic integration computation library, with limitation still.

If you are using Windows OS or MacOS then you are on your own.

The C++ codes are only located in the **/src** directory , it is designed to contain all the '.cpp' files, then inside the **/include/** we have header files and another folder **/include/symintegral/** is

also a folder that contain header files too.

You can open them one by one, read them and making sense of it based on the function that you are looking for.

There are total of 27 **.cpp** files and 26 **.h** / header files. So it won't take a long time for someone to learn it.

II. Techniques of Integration

We will cover some techniques of integration and also two Fundamental Theorems of Calculus that are very useful in solving integral problems or even differential equations.

Theorem 2.1: The First Fundamental Theorem of Calculus

Let f be continuous on the closed interval $[a, b]$ and let x be a (variable) point in (a, b). Then

$$\frac{d}{dx}\int_a^x f(t)\,dt = f(x)$$

Theorem 2.2: The Second Fundamental Theorem of Calculus

Let f be continuous (hence integrable) on $[a, b]$, and let F be any antiderivative of f on $[a, b]$. Then

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

- Integration by Parts

[SI*] From the analogue of the product rule for differentiation. Suppose we have two functions $f(x)$ and $g(x)$ with

$$\frac{d\,f(x)g(x)}{dx} = f(x)\frac{dg(x)}{dx} + g(x)\frac{df(x)}{dx} \tag{2.1}$$

Integrating both sides and rearranging the terms, we will get the integration by parts formula:

$$\int f(x)\frac{dg(x)}{dx}\,dx = f(x)g(x) - \int g(x)\frac{df(x)}{dx}\,dx \tag{2.2}$$

This formula is only useful if $\int g(x)\frac{df(x)}{dx}\,dx$ or $\int f(x)\frac{dg(x)}{dx}\,dx$ is easier to integrate than $\int f(x)g(x)\,dx$.

The application for integration by part is very useful in statistics, e.g. to be able to compute the mean and variance for exponential distribution.

- Substitution Rule for Indefinite Integral

[SI*] The substitution rule is nothing more than the Chain Rule in reverse.

Theorem 2.3: Substitution Rule for Indefinite Integrals

Let g be a differentiable function and suppose that F is an antiderivative of f. Then

$$\int f(g(x))g'(x)\,dx = F(g(x)) + C$$

Theorem 2.4: Substitution Rule for Definite Integrals

Let g have a continuous derivative on $[a, b]$, and let f be continuous on the range of g. Then

$$\int_a^b f(g(x))g'(x)\,dx = \int_{g(a)}^{g(b)} f(u)\,du$$

where $u = g(x)$.

III. Integration by Parts

- Example: Exponential Distribution

[SI*] The exponential distribution, which is sometimes used to model the lifetimes of electrical or mechanical components, has probability density function

$$f(x) = \begin{cases} \lambda e^{-\lambda x}, & \text{if } 0 \leq x \\ 0, & \text{otherwise} \end{cases}$$

where λ is some positive constant. Compute the mean μ and the variance σ^2.

Solution:

To compute for the mean we will have

$$\begin{aligned} E(X) &= \int_{-\infty}^{\infty} xf(x)\,dx \\ &= \int_{-\infty}^{0} x \cdot 0\,dx + \int_{0}^{\infty} x\lambda e^{-\lambda x}\,dx \end{aligned}$$

We apply integration by parts in the second integral with

$$\begin{aligned} u &= x \\ dv &= \lambda e^{-\lambda x}\,dx \\ du &= dx \\ v &= -e^{-\lambda x} \end{aligned}$$

Thus

$$\begin{aligned} E(X) &= [-x\lambda e^{-\lambda x}]_0^{\infty} - \int_0^{\infty} (-e^{-\lambda x})\,dx \\ &= (-0+0) + \left[-\frac{1}{\lambda}e^{-\lambda x}\right]_0^{\infty} \\ &= \frac{1}{\lambda} \end{aligned}$$

The variance is

$$\begin{aligned} \sigma^2 &= E(X^2) - \mu^2 \\ &= \int_{-\infty}^{\infty} x^2 f(x)\,dx - \left(\frac{1}{\lambda}\right) \\ &= \int_{-\infty}^{0} x^2 \cdot 0 + \int_0^{\infty} x^2\,\lambda e^{-\lambda x}\,dx - \frac{1}{\lambda^2} \\ &= [-x^2 e^{-\lambda x}]_0^{\infty} - \int_0^{\infty} (-e^{-\lambda x})2x\,dx - \frac{1}{\lambda^2} \\ &= (-0+0) + 2\int_0^{\infty} xe^{-\lambda x}\,dx - \frac{1}{\lambda^2} \\ &= 2\frac{1}{\lambda^2} - \frac{1}{\lambda^2} \\ &= \frac{1}{\lambda^2} \end{aligned}$$

[SI*] When they say integration is an art it is true, you need more practice to be able to handle all kinds of integral problems.

If we are facing this kind of equation:

$$f(x) = x\lambda e^{-\lambda x}$$

then what we need to do is learning the pattern

$$\begin{aligned}
\int xe^x \, dx &= (x-1)e^x + C \\
\int x^2e^x \, dx &= (x^2 - 2x + 2)e^x + C \\
\int x^3e^x \, dx &= (x^3 - 3x^2 + 6x - 6)e^x + C \\
\int x^4e^x \, dx &= (x^4 - 4x^3 + 12x^2 - 24x + 24)e^x + C \\
&\vdots
\end{aligned}$$

so for the general function of

$$f(x) = x^n e^x$$

The algorithm will be like this:

```
Symbolic sgn = 1;
int k = 1;
Symbolic integral;
for (int i = 0; i <= n; i++)
{
integral += k * sgn * (x^(n—i)) * exp(x)
sgn = —sgn;
k *= n—i;
}
return integral;
```

The C++ file to handle cases of integration by parts is located in **src/integrate.cpp**

```
Symbolic integrate(const Symbolic &f,const Symbolic
    &x)
{
       list<Equations> eq;
       list<Equations>::iterator i;
       UniqueSymbol a, b, c;
...
...
```

Code 3: *src/integrate.cpp*

Now for the case of $f(x) = x^n e^x$, it is handled at this part of codes

```
Symbolic integrate(const Symbolic &f,const Symbolic &x)
{
```

```
                ...
                ...
                eq = ((x^b)*exp(x)).match(f, (a,b)); // case for x^b
                     * exp(x)
                for(i=eq.begin(); i!=eq.end(); ++i)
                try {
                       Symbolic bp = rhs(*i, b);
                       if(bp.type() == typeid(Numeric)
                       && Number<void>(bp)->numerictype() == typeid(
                           int)
                       && Number<void>(bp)>Number<int>(0))
                       {
                              int n = CastPtr<const Number<int> >(bp
                                  )->n, sgn = 1;
                              Symbolic integral, nf = 1;
                              for(; n>=0; nf*=n, --n, sgn=-sgn)
                                  integral += sgn*nf*(x^n)*exp(x);
                              return integral;
                       }
                } catch(const SymbolicError &se) {}

        }
```

Code 4: *src/integrate.cpp to handle $x^n e^x$*

For the computation of mean and variance of exponential distribution with SymIntegration we will get this raw result, where the e^{-infty} is still as it is, and not returning to 0, we will work on it in the future.

```
#include <iostream>
#include "symintegrationc++.h"

using namespace std;

int main(void)
{
       Symbolic x("x"), l("l"), Inf("Inf"), y, Ex,
           Varx, u, dv, v, du;

       y = l*exp(-l*x);
       u = x;
       dv = l*exp(-l*x);
       du = df(u,x);
       v = integrate(dv,x);

       cout << "f(x) = " << y << endl;

       for(int i=1;i<=1;i++)
       {
              y = integrate(y,x);
```

```
                cout << i << "-st integral of f(x) = "
                     << y << endl;
            }
            cout << "int_{0}^{Inf} f(x) = " << y[x==Inf]
                - y[x==0] <<endl;

            Ex = x*l*exp(-l*x);

            cout << "\nx f(x) = " << Ex << endl;
            cout << "\nu = " << u << endl;
            cout << "dv = " << dv << endl;
            cout << "du = " << du << endl;
            cout << "v = " << v << endl;

            Ex = integrate(Ex,x);
            cout << "\nE(x) = int x f(x) = " << Ex <<
                endl;
            cout << "int_{0}^{Inf} x f(x) = " << Ex[x==
                Inf] - Ex[x==0] <<endl;

            Varx = x*x*l*exp(-l*x);
            Varx = integrate(Varx,x);

            cout << "\nint x^{2} f(x) = " << Varx << endl
                ;
            cout << "int_{0}^{Inf} x^{2} f(x) = " << Varx
                [x==INFINITY, l==1] - Varx[x==0] <<endl;
            cout << "\nVar(x) = int x^{2} f(x) - mu^{2} =
                 " << Varx[x==INFINITY, l==1] - Varx[x
                ==0] - (1/l^(2)) << endl;

            return 0;
        }
```

Code 5: *Integration by Parts*

IV. The Method of Substitution

- Some Examples

[SI*] Suppose we want to compute the integral

$$\int x \sin x^2 \, dx$$

Solution:
Here the appropriate substitution is

$$u = x^2$$
$$du = 2x \, dx$$

Thus

$$\begin{aligned}\int x\ \sin\ x^2\ dx &= \int \frac{1}{2}\ \sin\ x^2\ (2x\ dx) \\ &= \int \frac{1}{2} \sin\ u\ du \\ &= -\frac{1}{2} \cos\ u + C \\ &= -\frac{1}{2} \cos\ x^2 + C\end{aligned}$$

[SI*] In SymIntegration the method of substitution can be handled in **src/integrate.cpp**, we can create a pattern of equation to be solved, just like the integration by parts method.

V. Fraction Integral

- Fraction Level 1

[SI*] Suppose we have this function that we want to integrate toward x.

$$f(x) = \frac{1}{ax+b}$$

a and b are constants.

The integral will be

$$\int \frac{1}{ax+b}\,dx = \frac{\ln(ax+b)}{a} + C \tag{2.3}$$

For the integrate function alone in SymIntegration we do not include the constant C that is obtained after integration.

* Note that the C of the indefinite integration cancels out, as it always will, in the definite integration. That is why in the statement of the Second Fundamental Theorem of Calculus we could use the phrase **any derivative**. In particular, we may always choose $C = 0$ in applying the Second Fundamental Theorem of Calculus [3].

[SI*] The C++ code to handle this fraction level 1 integral problem to make sure the integral will return the correct result is located in **src/functions.cpp** under the function **Symbolic Power::integrate(const Symbolic &s) const**

```
Symbolic Power::integrate(const Symbolic &s) const
{
const Symbolic &a = parameters.front();
const Symbolic &b = parameters.back();
...
...
}
```

```
Symbolic Power::integrate(const Symbolic &s) const
{
...
...

if(b == -1 && parameters.front().coeff(s,2) == 0)
    return ln(parameters.front()) * (1/ parameters.
    front().df(s));
...
...
}
```

Code 6: *fraction level 1 integral*

The **if** statement above is a condition: if the input of a function meets the criteria that the power is -1 / a fraction ($f(x)^{-1} = \frac{1}{f(x)}$) then we will return it as **ln(parameters.front())**

* **(1/ parameters.front().df(s))**. The symbolic s is used to replace the variable in the integration problem.

I will introduce to the reader here about **parameters.front()** and **parameters.back()**, suppose we have a function

$$f(x) = (ax^2 + bx + c)^d \tag{2.4}$$

with a, b, c as constants.

* The **parameters.front()** of the function from equation (2.4) is $ax^2 + bx + c$.
  parameters.front().coeff(s,2) is a.
  parameters.front().coeff(s,1) is b.
  parameters.front().coeff(s,0) is c.

* The **parameters.back()** of the function from equation (2.4) is d.

With this knowledge we can create a lot of **if, if.. else..** conditions to make the integration computation complete and better.

- Fraction Level 2

[SI*] Suppose we have this function that we want to integrate toward x.

$$f(x) = \frac{1}{ax^2 + bx + c}$$

a, b and c are constants.

The integral will be

$$\begin{aligned}\int \frac{1}{ax^2 + bx + c}\, dx = &-\sqrt{\frac{-1}{4ac - b^2}} \ln\left(x + \frac{-4ac\sqrt{\frac{-1}{4ac-b^2}} + b^2\sqrt{\frac{-1}{4ac-b^2}} + b}{2a}\right) \\ &+ \sqrt{\frac{-1}{4ac - b^2}} \ln\left(x + \frac{4ac\sqrt{\frac{-1}{4ac-b^2}} - b^2\sqrt{\frac{-1}{4ac-b^2}} + b}{2a}\right) + C\end{aligned} \tag{2.5}$$

[SI*] The codes for the SymIntegration to handle this integral is located in **src/functions.cpp**

```
...
...

Symbolic Power::integrate(const Symbolic &s) const
{
      ...
      ...
      if(b == -1 && parameters.front().coeff(s,2) != 0)
      {
            double a1 = parameters.front().coeff(s,2);
            double b1 = parameters.front().coeff(s,1);
            double c1 = parameters.front().coeff(s,0);
```

```
                        double D_inv = sqrt(-1/(4*a1*c1-b1*b1));

                        return - D_inv * ln(s + (-4*a1*c1*D_inv + b1
                            *b1*D_inv + b1)/(2*a1)) + D_inv * ln(s +
                            (4*a1*c1*D_inv - b1*b1*D_inv + b1)/(2*a1
                            )) ;
                    }
                    ...
                    ...
                }
                ...
                ...
```

Code 7: *fraction level 2 integral*

VI. Trigonometry Integral

- Trigonometry Level 1

[SI*] We will want to compute integral of trigonometry at the basic level like these:

$$\int \sin(x)\ dx = -\cos(x) + C$$
$$\int \sin(3x+5)\ dx = -\frac{1}{3}\cos(3x+5) + C$$
$$\int \cos(x)\ dx = \sin(x) + C$$
$$\int \cos(3x+5)\ dx = \frac{1}{3}\sin(3x+5) + C$$
$$\int \tan(x)\ dx = -\ln(\cos(x)) + C$$
$$\int \tan(3x+5)\ dx = \frac{1}{6}\ln(\tan^2(3x+5)+1) + C$$
$$\int \cot(x)\ dx = \ln(\sin(x)) + C$$
$$\int \cot(3x+5)\ dx = -\frac{1}{6}\ln(\tan^2(3x+5)+1) + \frac{1}{3}\ln(\tan(3x+5)) + C$$
$$\int \sec(x)\ dx = -\frac{\ln(\cos(x))}{\sin(x)} + C$$
$$\int \sec(3x+5)\ dx = -\frac{1}{3}\frac{\ln(\cos(3x+5))}{\sin(3x+5)}$$
$$\int \csc(x)\ dx = \frac{\ln(\sin(x))}{\cos(x)} + C$$
$$\int \csc(3x+5)\ dx = \frac{1}{3}\frac{\ln(\sin(3x+5))}{\cos(3x+5)}$$

or in general form

$$\int \sin(ax+b)\ dx = -\frac{1}{a}\cos(ax+b) + C$$
$$\int \cos(ax+b)\ dx = \frac{1}{a}\sin(ax+b) + C$$
$$\int \tan(ax+b)\ dx = \frac{1}{2a}\ln(\tan^2(ax+b)+1) + C$$
$$\int \cot(ax+b)\ dx = -\frac{1}{2a}\ln(\tan^2(ax+b)+1) + \frac{1}{a}\ln(\tan(ax+b)) + C$$
$$\int \sec(ax+b)\ dx = -\frac{1}{a}\frac{\ln(\cos(ax+b))}{\sin(ax+b)}$$
$$\int \csc(ax+b)\ dx = \frac{1}{a}\frac{\ln(\sin(ax+b))}{\cos(ax+b)}$$

At first, the raw codes from SymbolicC++ 3.35 cannot perform that computation, in the **src/functions.cpp** there are implementations for sin, cos, tan but they are imperfect and if the case is not covered it will has an output of **return Integral(*this,s)**, which mean the integral

cannot be computed.

So we correct it and put the formula so SymIntegration can compute the integral problems above, which we call them as Level 1 trigonometry problems.

```
...
...

Symbolic Tan::integrate(const Symbolic &s) const
{
       const Symbolic &x = parameters.front();
       if(x == s) return -ln(cos(x)) * (1 / parameters.front().
           df(s));
       if(df(s) == 0) return *this * s;
       return ln( tan(parameters.front()) * tan(parameters.
           front()) + 1) * ( 1 / (2*parameters.front().df(s)) )
            ;
}

...
```

Code 8: *Implementation of integral for tangent function*

README.md functions.cpp main.cpp

```
170
171 Symbolic Acos::integrate(const Symbolic &s) const
172 {
173  const Symbolic &x = parameters.front();
174  if(x == s) return x*acos(x) - sqrt(1-x*x) ;
175  if(df(s) == 0) return *this * s;
176  return ( - sqrt(1-(parameters.front())*(parameters.front()) ) /  parameters.front().df(s) ) + ( parameters.front()*acos(parameters.front()) ,
177 }
178
179
180 ////////////////////////////////
181 // Implementation of Tan      //
182 ////////////////////////////////
183
184 Tan::Tan(const Tan &s) : Symbol(s) {}
185
186 Tan::Tan(const Symbolic &s) : Symbol(Symbol("tan")[s]) {}
187
188 Simplified Tan::simplify() const
189 {
190  const Symbolic &s = parameters.front().simplify();
191  if(s == 0) return Number<int>(0);
192  if(s.type() == typeid(Product))
193  {
194   CastPtr<const Product> p(s);
195   if(p->factors.front() == -1) return -Tan(-s);
196  }
197  if(s.type() == typeid(Numeric) &&
198     Number<void>(s).numerictype() == typeid(double))
199   return Number<double>(tan(CastPtr<const Number<double> >(s)->n));
200  return *this;
201 }
202
203 Symbolic Tan::df(const Symbolic &s) const
204 { return tan(parameters.front()) * tan(parameters.front()) * parameters.front().df(s) + parameters.front().df(s) ; }
205
206 Symbolic Tan::integrate(const Symbolic &s) const
207 {
208  const Symbolic &x = parameters.front();
209  if(x == s) return -ln(cos(x)) * (1 / parameters.front().df(s));
210  if(df(s) == 0) return *this * s;
211  return ln( tan(parameters.front()) * tan(parameters.front()) + 1) * ( 1 / (2*parameters.front().df(s)) ) ;
212 }
213
```

Figure 2.1: *The implementation of tangent function to compute its derivative and integral in* ***src/functions.cpp****.*

- Trigonometry Level 2

  [SI*] The trigonometry integrals that we consider as level 2 are

$$\int \sin\ mx\ \cos\ nx\ dx$$
$$\int \cos\ mx\ \cos\ nx\ dx$$
$$\int \sin\ mx\ \sin\ nx\ dx$$

  Integrals of this type occurs in many physics and engineering applications. By hands we can use the product identitites to handle these integrals.

- Trigonometry Level 3

[SI*] The trigonometry integrals that we consider as level 3 are

$$\int \sin^n x\, dx$$
$$\int \cos^n x\, dx$$
$$\int \tan^n x\, dx$$
$$\int \sec^n x\, dx$$
$$\int \csc^n x\, dx$$
$$\int \cot^n x\, dx$$

We happen to check on Wolfram Alpha for these indefinite integrals. The Wolfram Alpha gives result of these integrals in hypergeometric function term, for example:

$$\int \sin^n\ x\, dx = -\cos(x)\sin^{n+1}(x)\sin^{-n-1}(x)\ {}_2F_1(\frac{1}{2}, \frac{1-n}{2}; \frac{3}{2}; \cos^2(x))$$

For the record, hypergeometric function ${}_2F_1(\frac{1}{2}, \frac{1-n}{2}; \frac{3}{2}; \cos^2(x))$ can be computed with SymIntegration, Boost, or even self made C++ codes, but it will produce approximated integral computation, and we tried to use this formulas with hypergeometric function, it compute very slow.

So we refer to the reduction formula to compute these trigonometry integral level 3.

$$\begin{aligned}
\int \sin^n x\, dx &= -\frac{1}{n}\sin^{n-1}(x)\ \cos(x) + \frac{n-1}{n}\int \sin^{n-2}(x)\, dx \\
\int \cos^n x\, dx &= \frac{1}{n}\cos^{n-1}(x)\ \sin(x) + \frac{n-1}{n}\int \cos^{n-2}(x)\, dx \\
\int \tan^n x\, dx &= \frac{\tan^{n-1}(x)}{n-1} - \int \tan^{n-2}(x)\, dx \\
\int \sec^n x\, dx &= \frac{\sec^{n-2}(x)\ \tan(x)}{n-1} + \frac{n-2}{n-1}\int \sec^{n-2}(x)\, dx \\
\int \csc^n x\, dx &= \frac{-\csc^{n-2}(x)\ \cot(x)}{n-1} + \frac{n-2}{n-1}\int \csc^{n-2}(x)\, dx \\
\int \cot^n x\, dx &= -\frac{1}{n-1}\cot^{n-1}(x) - \int \cot^{n-2}(x)\, dx
\end{aligned} \tag{2.6}$$

With this reduction formula we can use recursive method, with **for** loop technique in C++ after we find the pattern, we can code it easily. The computation time with reduction formula and this recursive method is faster than using Hypergeometric function.

[SI*] We will examine each integral and learn about their patterns, one by one. Starting with $\int \sin^n x\, dx$

i. $\int \sin^n x\ dx$

These are the formula of $\int \sin^n x\ dx$ with $n = 2$ to $n = 8$.

$$\int \sin^2 x\ dx = \frac{x}{2} - \frac{\sin\ x\ \cos\ x}{2}$$

$$\int \sin^3 x\ dx = \frac{\cos^3 x}{3} - \cos\ x$$

$$\int \sin^4 x\ dx = \frac{3x}{8} - \frac{\sin^3\ x\ \cos\ x}{4} - \frac{3\sin\ x\ \cos\ x}{8}$$

$$\int \sin^5 x\ dx = -\frac{\cos^5 x}{5} + \frac{2\cos^3 x}{3} - \cos\ x$$

$$\int \sin^6 x\ dx = \frac{5x}{16} - \frac{\sin^5\ x\ \cos\ x}{6} - \frac{5\sin^3\ x\ \cos\ x}{24} - \frac{5\sin\ x\ \cos\ x}{16}$$

$$\int \sin^7 x\ dx = \frac{\cos^7 x}{7} - \frac{3\cos^5 x}{5} + \frac{3\cos^3 x}{3} - \cos\ x$$

$$\int \sin^8 x\ dx = \frac{35x}{128} - \frac{\sin^7\ x\ \cos\ x}{8} - \frac{7\sin^5\ x\ \cos\ x}{48} - \frac{35\sin^3\ x\ \cos\ x}{192} - \frac{35\sin\ x\ \cos\ x}{128}$$

From the first 8 terms we will get this intuition that the pattern occur for even power and odd power, so it will divide into two cases, in C++ we can use the statement (**n % 2 == 0**) in the **if** statement to handle this.

The manual way to solve this can be traced back to the reduction formula, for example if you want to compute $\int \sin^8 x\ dx$, then you start in a decreasing **for** loop with $i = n, i >= 2, i = i - 2$.

Case 1: For $\int \sin^n x\ dx$ with even n
The numerator coefficient and denominator coefficient are making patterns that can be decoded.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
const Symbolic &a = parameters.front();
const Symbolic &b = parameters.back();
int bpower = parameters.back().coeff(s,0);
if(a.type() == typeid(Sin))
{
       if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power
           case
       {
              Symbolic c = 1;
              Symbolic d = parameters.back().coeff(s,0);
              Symbolic integral;
              for(int i = bpower ; i >= 2 ; i = i-2)
              {
                     integral -= c*cos(s)*((sin(s))^(i-1))/(d) ;
```

```
                        d = d*(i-2);
                        c = c*(i-1);
                        if (i==4)
                        {
                                integral += ( (c*s)/(d) );
                        }
                }
        return integral;
        }
}
...
```

Code 9: *level 3 integral for sine even case*

As you can see the logic for the code above: we multiply the numerator with c that is the odd factorial ($1 \times 3 \times 5 \times \cdots \times n-1$), and multiply the denominator with the even factorial ($2 \times 4 \times 6 \times \cdots \times n$).

The looping starts from the **bpower**, the variable name to represents n, the power for the integral of $\sin^n x$, then descending to 2, and decreasing by 2.

Notice that the numerator and denominator coefficient for x is the same as $\sin\ x\ \cos\ x$, so we only need to compute the coefficient for $\sin\ x\ \cos\ x$ and then copy it as the coefficient for x.

We start to compute the coefficient with the highest power, for example we want to

compute $\int \sin^8 x \, dx$, then we start with $i = 8$ and we will have:

$$
\begin{aligned}
i &= 8 \\
c &= 1 \\
d &= 8 \\
\text{integral} &= -\frac{c * \cos(s) * \sin^{i-1}(s)}{d} \\
d &= d * (i-2) \\
c &= c * (i-1) \\
i &= i-2 \\
&* * * * * * * * * * * * \\
i &= 6 \\
c &= 7 \\
d &= 48 \\
\text{integral} &= -\frac{c * \cos(s) * \sin^{i-1}(s)}{d} \\
d &= d * (i-2) \\
c &= c * (i-1) \\
i &= i-2 \\
&* * * * * * * * * * * * \\
i &= 4 \\
c &= 35 \\
d &= 192 \\
\text{integral} &= -\frac{c * \cos(s) * \sin^{i-1}(s)}{d} \\
d &= d * (i-2) \\
c &= c * (i-1) \\
i &= i-2 \\
&* * * * * * * * * * * * \\
i &= 2 \\
c &= 105 \\
d &= 384 \\
\text{integral} &= -\frac{c * \cos(s) * \sin^{i-1}(s)}{d} \\
d &= d * (i-2) \\
c &= c * (i-1) \\
i &= i-2
\end{aligned}
$$

We stop when $i = 2$ and we will obtain $\frac{105}{384} = \frac{35}{128}$ as the coefficient for $\sin\ x\ \cos\ x$.

It only takes a lot of papers and pen to be able to decode this patterns. Then some patience and persistence with focus to create the C++ codes.

Case 2: For $\int \sin^n x\ dx$ with odd n
The numerator coefficient and denominator coefficient are making patterns that can be decoded. This time it is easier than the even case, but a little bit tricky with alternating sign and need to use combination formula.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
...
if(a.type() == typeid(Sin))
{
if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power case
{
       Symbolic sgn = -1;
       Symbolic integral;
       int j =1;
       int m = bpower-(0.5*(bpower+1));
       for(int i = 1 ; i <= bpower ; i = i+2)
       {
              integral += sgn*combinations(m,j-1)*((cos(s))^(i))
                  /(i);
              sgn = -sgn;
              j = j+1;
       }
       return integral;
}
}
```

Code 10: *level 3 integral for sine odd case*

When I was trying to find the pattern, it is nice to write from the lowest odd power to a certain odd power and see the pattern for the numerator and denominator as the power rises.

$$\int \sin^3 x\ dx = \frac{\cos^3 x}{3} - \cos\ x$$
$$\int \sin^5 x\ dx = -\frac{\cos^5 x}{5} + \frac{2\cos^3 x}{3} - \cos\ x$$
$$\int \sin^7 x\ dx = \frac{\cos^7 x}{7} - \frac{3\cos^5 x}{5} + \frac{3\cos^3 x}{3} - \cos\ x$$
$$\int \sin^9 x\ dx = -\frac{\cos^9 x}{9} + \frac{4\cos^7 x}{7} - \frac{6\cos^5 x}{5} + \frac{4\cos^3 x}{3} - \cos\ x$$

Right from the bat, the denominator has an easy pattern of odd number decreasing from the n to 1, the denominator has the same coefficient as the power for the $\cos(x)$. So it is clear already.

The problem now, is the numerator, we have this kind of pattern:

$$\begin{array}{ccccccccc} & & & 1 & & 1 & & & \\ & & 1 & & 2 & & 1 & & \\ & 1 & & 3 & & 3 & & 1 & \\ 1 & & 4 & & 6 & & 4 & & 1 \end{array}$$

It is the infamous Pascal' triangle. Which can be derived from the combination formula:

$$\begin{aligned} {}_0C_0 &= 1 \\ {}_1C_0 &= 1 \\ {}_1C_1 &= 1 \\ {}_2C_0 &= 1 \\ {}_2C_1 &= 2 \\ {}_2C_2 &= 1 \\ {}_3C_0 &= 1 \\ {}_3C_1 &= 3 \\ {}_3C_2 &= 3 \\ {}_3C_3 &= 1 \\ {}_4C_0 &= 1 \\ {}_4C_1 &= 4 \\ {}_4C_2 &= 6 \\ {}_4C_3 &= 4 \\ {}_4C_4 &= 1 \end{aligned}$$

This explains the usage of **combinations** function in the C++ code.

ii. $\int \cos^n x \, dx$

These are the formula of $\int \cos^n x \, dx$ with $n = 2$ to $n = 8$.

$$\begin{aligned} \int \cos^2 x \, dx &= \frac{x}{2} + \frac{\sin\ x\ \cos\ x}{2} \\ \int \cos^3 x \, dx &= -\frac{\sin^3 x}{3} + \sin\ x \\ \int \cos^4 x \, dx &= \frac{3x}{8} + \frac{\sin\ x\ \cos^3\ x}{4} + \frac{3\sin\ x\ \cos\ x}{8} \\ \int \cos^5 x \, dx &= \frac{\sin^5 x}{5} - \frac{2\sin^3 x}{3} + \sin\ x \\ \int \cos^6 x \, dx &= \frac{5x}{16} + \frac{\sin\ x\ \cos^5\ x}{6} + \frac{5\sin\ x\ \cos^3\ x}{24} + \frac{5\sin\ x\ \cos\ x}{16} \\ \int \cos^7 x \, dx &= -\frac{\sin^7 x}{7} + \frac{3\sin^5 x}{5} - \frac{3\sin^3 x}{3} + \sin\ x \\ \int \cos^8 x \, dx &= \frac{35x}{128} + \frac{\sin\ x\ \cos^7\ x}{8} + \frac{7\sin\ x\ \cos^5\ x}{48} + \frac{35\sin\ x\ \cos^3\ x}{192} + \frac{35\sin\ x\ \cos\ x}{128} \end{aligned}$$

Right after we observe the equations above, it is really familiar, the pattern for the numerator and denominator will be like the sine counterpart, thus we only need to exchange the sign, what is $-$ become $+$ and the other way around, and also exchanging sine into cosine and cosine into sine.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
const Symbolic &a = parameters.front();
const Symbolic &b = parameters.back();
int bpower = parameters.back().coeff(s,0);
if(a.type() == typeid(Cos))
{
       if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power case
       {
              Symbolic c = 1;
              Symbolic d = parameters.back().coeff(s,0);
              Symbolic integral;
              for(int i = bpower ; i >= 2 ; i = i-2)
              {
                     integral += c*sin(s)*((cos(s))^(i-1))/(d) ;
                     d = d*(i-2);
                     c = c*(i-1);
                     if (i==4)
                     {
                            integral += ( (c*s)/(d) );
                     }
              }
              return integral;
       }
}
...
```

Code 11: *level 3 integral for cosine even case*

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
       const Symbolic &a = parameters.front();
       const Symbolic &b = parameters.back();
       int bpower = parameters.back().coeff(s,0);
       if(a.type() == typeid(Cos))
       {
              if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power
                   case
              {
              Symbolic sgn = 1;
```

```
                Symbolic integral;
                int j =1;
                int m = bpower—(0.5*(bpower+1));
                for(int i = 1 ; i <= bpower ; i = i+2)
                {
                integral += sgn*combinations(m,j—1)*((sin(s))^(i))/(i);
                sgn = —sgn;
                j = j+1;
                }
                return integral;
                }
        }
        ...
```

Code 12: *level 3 integral for cosine odd case*

iii. $\int \sec^n x\, dx$

These are the formula of $\int \sec^n x\, dx$ with $n = 2$ to $n = 8$.

$$\int \sec^2 x\, dx = \frac{\sin\ x}{\cos\ x}$$
$$\int \sec^3 x\, dx = -\frac{\log(\sin\ (x) - 1)}{4} + \frac{\log(\sin\ (x) + 1)}{4} - \frac{\sin\ x}{2\sin^2 x - 2}$$
$$\int \sec^4 x\, dx = \frac{2\sin\ x}{3\cos\ x} + \frac{\sin\ x}{3\cos^3\ x}$$
$$\int \sec^5 x\, dx = -\frac{3\sin^3\ x - 5\sin\ x}{8\sin^4\ x - 16\sin^2\ x + 8} - \frac{3\log(\sin\ (x) - 1)}{16} + \frac{3\log(\sin\ (x) + 1)}{16}$$
$$\int \sec^6 x\, dx = \frac{8\sin\ x}{15\cos\ x} + \frac{4\sin\ x}{15\cos^3\ x} + \frac{\sin\ x}{5\cos^5\ x}$$
$$\int \sec^7 x\, dx = \frac{-15\sin^5\ x + 40\sin^3\ x - 33\sin\ x}{48\sin^6\ x - 144\sin^4\ x + 144\sin^2\ x - 48} - \frac{5\log(\sin\ (x) - 1)}{32} + \frac{5\log(\sin\ (x) + 1)}{32}$$
$$\int \sec^8 x\, dx = \frac{16\sin\ x}{35\cos\ x} + \frac{8\sin\ x}{35\cos^3\ x} + \frac{6\sin\ x}{35\cos^5\ x} + \frac{\sin\ x}{7\cos^7\ x}$$

The same case occurs again here, the pattern is repeated every $n + 2$, so there will be the odd power pattern and the even power pattern.

Case 1: For $\int \sec^n x\, dx$ with even n
The numerator coefficient and denominator coefficient are making patterns that can be decoded.

$$\int \sec^2 x\, dx = \frac{\sin\ x}{\cos\ x}$$
$$\int \sec^4 x\, dx = \frac{2\sin\ x}{3\cos\ x} + \frac{\sin\ x}{3\cos^3\ x}$$
$$\int \sec^6 x\, dx = \frac{8\sin\ x}{15\cos\ x} + \frac{4\sin\ x}{15\cos^3\ x} + \frac{\sin\ x}{5\cos^5\ x}$$
$$\int \sec^8 x\, dx = \frac{16\sin\ x}{35\cos\ x} + \frac{8\sin\ x}{35\cos^3\ x} + \frac{6\sin\ x}{35\cos^5\ x} + \frac{\sin\ x}{7\cos^7\ x}$$
$$\int \sec^{10} x\, dx = \frac{128\sin\ x}{315\cos\ x} + \frac{64\sin\ x}{315\cos^3\ x} + \frac{16\sin\ x}{105\cos^5\ x} + \frac{8\sin\ x}{63\cos^7\ x} + \frac{\sin\ x}{9\cos^9\ x}$$

The patterns that can be seen from those equations above are

- You may have misinterpreted it first, but the denominator are all the same, for example:

  $$\int \sec^6 x\, dx = \frac{8\sin\ x}{15\cos\ x} + \frac{4\sin\ x}{15\cos^3\ x} + \frac{\sin\ x}{5\cos^5\ x}$$

  is the same as

  $$\int \sec^6 x\, dx = \frac{8\sin\ x}{15\cos\ x} + \frac{4\sin\ x}{15\cos^3\ x} + \frac{3\sin\ x}{15\cos^5\ x}$$

  Learning from that, the pattern for the numerator can be decoded.

Knowing that the denominator is the same, we only need to find out, how we get the denominator coefficient of 1 for $n = 2$, or of 3 for $n = 4$, or of 15 for $n = 6$.

Turns out, the first intuition is quite correct, it is the factorial odd, we will declare the denominator as variable d_0:

$$d_0 = (n-1) * (n-3) * (n-5) * \cdots * (1)$$

If it is $\int \sec^8 x \, dx$, then the denominator will be

$$d_0 = 7 * 5 * 3 * 1 = 105$$

In the equation for $\int \sec^8 x \, dx$, the denominator is 35, instead of 105, this happens because the numerator is divisible with the denominator till they have no greatest common divisor anymore but 1, it is caused by the computer algebra system that always shows the most simplified version, so you can adjust the equation again to find the raw number for the denominator and the numerator. Making the denominator into the same number is not enough, so we need them in raw number version.

- After we make the denominator to become equal we will have this:

$$\int \sec^2 x \, dx = \frac{\sin\ x}{\cos\ x}$$
$$\int \sec^4 x \, dx = \frac{2\sin\ x}{3\cos\ x} + \frac{\sin\ x}{3\cos^3\ x}$$
$$\int \sec^6 x \, dx = \frac{8\sin\ x}{15\cos\ x} + \frac{4\sin\ x}{15\cos^3\ x} + \frac{3\sin\ x}{15\cos^5\ x}$$
$$\int \sec^8 x \, dx = \frac{16\sin\ x}{35\cos\ x} + \frac{8\sin\ x}{35\cos^3\ x} + \frac{6\sin\ x}{35\cos^5\ x} + \frac{5\sin\ x}{35\cos^7\ x}$$
$$\int \sec^{10} x \, dx = \frac{128\sin\ x}{315\cos\ x} + \frac{64\sin\ x}{315\cos^3\ x} + \frac{48\sin\ x}{315\cos^5\ x} + \frac{40\sin\ x}{315\cos^7\ x} + \frac{35\sin\ x}{315\cos^9\ x}$$

so the pattern of the numerator can be seen like this:

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 2 & & & \\ & & 3 & & 4 & & 8 & & \\ & 5 & & 6 & & 8 & & 16 & \\ 35 & & 40 & & 48 & & 64 & & 128 \end{array}$$

Notice that we have to change the denominator into its raw version (for $\int \sec^8 x \, dx$ the denominator should be 105 instead of 35, and for $\int \sec^{10} x \, dx$ the denominator should be 945 instead of 315), this is the logic: the pattern can be decoded at its' raw numbers.

Thus

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & 1 & & 2 & & & \\ & & 3 & & 4 & & 8 & & \\ & 15 & & 18 & & 24 & & 48 & \\ 105 & & 120 & & 144 & & 192 & & 384 \end{array}$$

The first entry for the next iteration is an odd multiplication of the current first entry from before with an increasing odd number. The first iteration will multiplied by 1, the

second will be multiplied by 3, the next iteration will be multiplied by 5, and so on. This odd multiplication will be called as k, While the second entry to the last entry will be multiplied by even multiplication that will be called as l and it is increasing by 2 at every iteration.

$$
\begin{aligned}
&\text{for } i = 1 \rightarrow \mathbf{v[0]} = 1 * k_{i=1} = 1 \\
&\text{for } i = 1 \rightarrow \mathbf{v[1]} = 1 * l_{i=1} = 2 \\
&\text{for } i = 2 \rightarrow \mathbf{v[0]} = \mathbf{v[0]}_{i=1} * k_{i=2} = 1 * 3 = 3 \\
&\text{for } i = 2 \rightarrow \mathbf{v[1]} = \mathbf{v[0]}_{i=1} * l_{i=2} = 1 * 4 = 4 \\
&\text{for } i = 2 \rightarrow \mathbf{v[2]} = \mathbf{v[1]}_{i=1} * l_{i=2} = 2 * 4 = 8 \\
&\text{for } i = 3 \rightarrow \mathbf{v[0]} = \mathbf{v[0]}_{i=2} * k_{i=3} = 3 * 5 = 15 \\
&\text{for } i = 3 \rightarrow \mathbf{v[1]} = \mathbf{v[0]}_{i=2} * l_{i=3} = 3 * 6 = 18 \\
&\text{for } i = 3 \rightarrow \mathbf{v[2]} = \mathbf{v[1]}_{i=2} * l_{i=3} = 4 * 6 = 24 \\
&\text{for } i = 3 \rightarrow \mathbf{v[3]} = \mathbf{v[2]}_{i=2} * l_{i=3} = 8 * 6 = 48
\end{aligned}
$$

Now we already have an idea about the pattern. For the C++ code, this time instead of using vector we will use array that is initialized with a very big size: **int v[999]**. Array is often used to replace vector, since vector has an invalid pointer problem / out of bounds memory access if you are not careful, array can also be used to become a matrix.

In order to determine the coefficient of the numerator with sine function we will use ascending **for** loop, to be exact we will use **for**$(i = 1; i < n - 1; i = i + 2)$.

We will declare variables / constants to help us compute, which are $d_0 = 1, k = 1, l = 2, c = 1$. Their value will be changing in every iteration (c will be used to store the first entry ($\mathbf{v}[0]$) of the previous array, $k = k + 2$ and $l = l + 2$).

We save the first entry of the previous array because it is going to be used to determine the first and second entry of the array at the current iteration.

$$
\begin{aligned}
k &= 1 \\
l &= 2 \\
d_0 &= 1 \\
c &= 1
\end{aligned}
$$

In every iteration we will clear the entries in the array.

So the first **for** loop will be like this:

$$
\begin{gathered}
i = 1 \\
d_0 = 3 \\
l = 4 \\
k = 3 \\
v[0] = 1 \\
v[1] = 2 \\
* * * * * * * * * * * \\
i = 3 \\
d_0 = 15 \\
l = 6 \\
k = 5 \\
v[0] = 3 \\
v[1] = 4 \\
v[2] = 8 \\
* * * * * * * * * * * \\
i = 5 \\
d_0 = 105 \\
l = 8 \\
k = 7 \\
v[0] = 15 \\
v[1] = 18 \\
v[2] = 24 \\
v[3] = 48 \\
* * * * * * * * * * * \\
i = 7 \\
d_0 = 945 \\
l = 10 \\
k = 9 \\
v[0] = 105 \\
v[1] = 120 \\
v[2] = 144 \\
v[3] = 192 \\
v[4] = 384
\end{gathered}
$$

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
       ...
```

```
if(a.type() == typeid(Sec))
{
...

if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power case
{
int v[999];
v[0] = 1;
Symbolic integral;
Symbolic d0 = 1;
int k = 1, l = 2, c=1;
for(int i = 1 ; i < bpower ; i = i+2) // to compute the denominator
{
       d0 *= i;
}
for(int i = 1 ; i < bpower - 1; i = i+2)
{
       c = v[0];
       int arrsec[999]; // make the size of the array as big as
           possible

       for(int j = 0 ; j < i-1 ; j = j+1)
       {
              arrsec[j] = v[j];
       }
       int d;
       for(int j = 1 ; j < i ; j = j+1)
       {
              d = arrsec[j-1];
              v[j] = d*l;
       }

       v[0] = c*k;
       v[1] = c*l;

       k= k + 2;
       l = l + 2;
}

int j_d = 1 ;
for(int i = 1 ; i < (0.5*bpower)+1; i = i+1)
{
       integral += ( v[i-1] * sin(s) ) / ( d0*(cos(s)^(bpower-j_d)) )
            ;
       j_d = j_d+2;
}
return integral;
}
```

```
        }
}
```

Code 13: *level 3 integral for secant even case*

Case 2: For $\int \sec^n x\, dx$ with odd n
Compared to the even case, the odd case for $\int \sec^n x\, dx$ is harder in terms of determining the pattern and then to code it with C++, we even have to use two vectors, the first one to store the coefficient for the numerator, and the second vector is used to store the "middle coefficient", in order to determine the correct coefficient for the numerator.

$$\int \sec^3 x\, dx = -\frac{\log(\sin(x)-1)}{4} + \frac{\log(\sin(x)+1)}{4} - \frac{\sin x}{2\sin^2 x - 2}$$

$$\int \sec^5 x\, dx = -\frac{3\sin^3 x - 5\sin x}{8\sin^4 x - 16\sin^2 x + 8} - \frac{3\log(\sin(x)-1)}{16} + \frac{3\log(\sin(x)+1)}{16}$$

$$\int \sec^7 x\, dx = \frac{-15\sin^5 x + 40\sin^3 x - 33\sin x}{48\sin^6 x - 144\sin^4 x + 144\sin^2 x - 48} - \frac{5\log(\sin(x)-1)}{32} + \frac{5\log(\sin(x)+1)}{32}$$

$$\int \sec^9 x\, dx = -\frac{105\sin^7 x - 385\sin^5 x + 511\sin^3 x - 279\sin x}{384\sin^8 x - 1536\sin^6 x + 2304\sin^4 x - 1536\sin^2 x + 384} - \frac{35\log(\sin(x)-1)}{256} + \frac{35\log(\sin(x)+1)}{256}$$

$$\int \sec^{11} x\, dx = \frac{-315\sin^9 x + 1470\sin^7 x - 2688\sin^5 x + 2370\sin^3 x - 965\sin x}{1280\sin^{10} x - 6400\sin^8 x + 12800\sin^6 x - 12800\sin^4 x + 6400\sin^2 x - 1280} - \frac{63\log(\sin(x)-1)}{512} + \frac{63\log(\sin(x)+1)}{512}$$

If we want to compute it with C++, then we can use the **for** as usual to help us. Then it is either ascending **for** loop or descending **for** loop. As time goes by and flight hours, you will know which one to use.

We can immediately notice some very obvious patterns:

- We have 3 different terms here, 2 are the log terms and the last is the term with numerator that has the sum of alternating sign of sine with odd power and denominator that has the sum of alternating sign of sine with even power.
- The terms with log has denominator of with multiplication of $(2 * i)$ that still store the previous value so we can determine it with $d_1 = d_1 * (2 * i)$. After we finish with the **for** loop, we will multiply again for the last time to obtain d_1 that will be used

$$d_1 = d_1 * (n-1)$$

- Now, we shall see the coefficient for the sum of alternating sign of sine with odd power:

$$1$$

$$3 \quad 5$$

$$15 \quad 40 \quad 33$$

$$105 \quad 385 \quad 511 \quad 279$$

$$315 \quad 1470 \quad 2688 \quad 2370 \quad 965$$

The first row represents $\int \sec^3 x \, dx$ and the last row represents $\int \sec^{11} x \, dx$. The sign is alternating actually, but it is omitted for better view, so the reader won't be confused. Alternating sign can be attached later on.

In this case, to determine the coefficient of the numerator with odd sine function we will use ascending **for** loop, to be exact we will use **for**$(i = 1; i < \frac{n-1}{2}; i = i + 1)$.

We also need to have extra variables to help compute the coefficient correctly, we give them initial value before all the **for** loop:

$$\begin{aligned} k &= 1 \\ l &= 2 \\ d_0 &= 2 \\ d_1 &= 2 \\ \text{first\_coeff} &= 3 \\ j &= 1 \\ m &= n - \left(\frac{n+1}{2}\right) \\ sgn &= -1 \end{aligned}$$

We are going to use vector $\mathbf{v}$ to represent the coefficient for the numerator of sine with odd power, the size of the vector is $\frac{n-1}{2}$.

At first, when $i = 1$, we know that $\mathbf{v}[0] = 1$, so how to determine $\mathbf{v}[0]$ and $\mathbf{v}[1]$ for $i = 2$ and continuing the loop to compute for n at higher number?

Go back and see the reduction formula at Equation (2.6) for $\int \sec^n x \, dx$, we will notice that there is $n-2$ and $n-1$ terms at every recursive step / the loop. So we will have to see the multiplication possibilities for the factorial odd: $(n-2) * (n-4) * \cdots * 1$ and the factorial even: $(n-1) * (n-3) * (n-5) * \cdots * 2$. We are focusing on the numerator now, so it is the factorial odd that we will use (the one related to $n-2$ factorial down with difference of 2).

If you take a look again at the triangle above, the trend is increasing then decreasing at the end, it is the typical of Pascal' triangle but we do not use combination here. Here we try step by step from $n = 3$ or $i = 1$:

$$\begin{aligned} n &= 3 \\ i &= 1 \\ \mathbf{v}[0] &= 1 \end{aligned}$$

that will make

$$\int \sec^3 x \, dx = -\frac{\log(\sin x - 1)}{4} + \frac{\log(\sin x + 1)}{4} - \frac{\mathbf{v}[0] \sin x}{2\sin^2 x - 2}$$

moving on to $n = 5$ and $i = 2$

$$\begin{aligned} n &= 5 \\ i &= 2 \\ \mathbf{v}[0] &= 3 \\ \mathbf{v}[1] &= 5 \end{aligned}$$

$$\int \sec^5 x \, dx = -\frac{\mathbf{v}[0] \sin^3 x - \mathbf{v}[1] \sin x}{8 \sin^4 x - 16 \sin^2 x + 8} - \frac{3 \log(\sin x - 1)}{16} + \frac{3 \log(\sin x + 1)}{16}$$

Then to $n = 7$ and $i = 3$

$$\begin{aligned} n &= 7 \\ i &= 3 \\ \mathbf{v}[0] &= 15 \\ \mathbf{v}[1] &= 40 \\ \mathbf{v}[2] &= 33 \end{aligned}$$

$$\int \sec^7 x \, dx = \frac{\mathbf{v}[0] \sin^5 x + \mathbf{v}[1] \sin^3 x - \mathbf{v}[2] \sin x}{48 \sin^6 x - 144 \sin^4 x + 144 \sin^2 x - 48} - \frac{5 \log(\sin x - 1)}{32} + \frac{5 \log(\sin x + 1)}{32}$$

So to be able to create the C++ code to compute this coefficients, when we have the initial condition of $\mathbf{v}[0] = 1$ at $i = 1$, how do we get into $\mathbf{v}[0] = 3, \mathbf{v}[1] = 5$ at $i = 2$ and $\mathbf{v}[0] = 15, \mathbf{v}[1] = 40, \mathbf{v}[2] = 33$ at $i = 3$ and so on correctly?

Watch this, we first define $last\_coeff = \mathbf{v}[j-1]$ with $j = 1, 2, 3, \ldots$ as the last index of

the vector from the previous i and $first\_coeff = 3$, so

$$
\begin{aligned}
n &= 3 \\
i &= 1 \\
\mathbf{v}[0] &= 1 \\
last\_coeff &= \mathbf{v}[0] = 1 \\
k &= 2 \\
&**********\\
n &= 5 \\
i &= 2 \\
\mathbf{v}[0] &= \mathbf{v}[0] * (n-2) = 3 \\
\mathbf{v}[1] &= [\mathbf{v}[0] * (n-2)] + k = [1*3] + 2 = 5 \\
\mathbf{mc}[0] &= \mathbf{v}[0] + \mathbf{v}[1] = 8 \\
last\_coeff &= \mathbf{v}[1] = 5 \\
k &= 8 \\
&**********\\
n &= 7 \\
i &= 3 \\
\mathbf{v}[0] &= \mathbf{v}[0] * (n-2) = 3*5 = 15 \\
\mathbf{v}[1] &= \mathbf{mc}[0] * (n-2) = 8*5 = 40 \\
\mathbf{v}[2] &= [last\_coeff * (n-2)] + k = [5*5] + 8 = 33 \\
\mathbf{mc}[0] &= \mathbf{v}[0] + \mathbf{v}[1] = 55 \\
\mathbf{mc}[1] &= \mathbf{v}[1] + \mathbf{v}[2] = 73 \\
last\_coeff &= \mathbf{v}[2] = 33 \\
k &= 48 \\
&**********\\
n &= 9 \\
i &= 4 \\
\mathbf{v}[0] &= \mathbf{v}[0] * (n-2) = 15*7 = 105 \\
\mathbf{v}[1] &= \mathbf{mc}[0] * (n-2) = 55*7 = 385 \\
\mathbf{v}[2] &= \mathbf{mc}[1] * (n-2) = 73*7 = 511 \\
\mathbf{v}[3] &= [last\_coeff * (n-2)] + k = [33*7] + 8 = 279 \\
\mathbf{mc}[0] &= \mathbf{v}[0] + \mathbf{v}[1] = 490 \\
\mathbf{mc}[1] &= \mathbf{v}[1] + \mathbf{v}[2] = 896 \\
\mathbf{mc}[2] &= \mathbf{v}[2] + \mathbf{v}[3] = 790 \\
last\_coeff &= \mathbf{v}[2] = 279 \\
k &= 384
\end{aligned}
$$

The $last\_coeff$ and $first\_coeff$ will be used in the C++ codes to represent $n-2$, it is a little bit of a handy trick.

We finally know the pattern and formula to obtain the coefficients for the numerator. It is not an easy one as I spent days to realize how to obtain this.

- The denominator that has the sum of alternating sign of sine with even power has a distinctive pattern that can be seen when you take the greatest common diviser out:

$$8\sin^4 x - 16\sin^2 x + 88(\sin^4 x - 2\sin^2 x + 1)$$
$$48\sin^6 x - 144\sin^4 x + 144\sin^2 x - 48 = 48(\sin^6 x - 3\sin^4 x + 3\sin^2 x - 1)$$
$$384\sin^8 x - \cdots + \cdots + 384 = 384(\sin^8 x - 4\sin^6 x + 6\sin^4 x - 4\sin^2 x + 1)$$

We are focusing on the denominator now, so it is the factorial even that we will use, with the initial value of $d_0 = 2$ we will have:

$$\text{for } n = 5 \rightarrow \; d_0 = d_0 * (n-1) = d_0 * (2 + 2 * i) = 8$$
$$\text{for } n = 7 \rightarrow \; d_0 = d_0 * (n-1) = d_0 * (2 + 2 * i) = 48$$
$$\text{for } n = 9 \rightarrow \; d_0 = d_0 * (n-1) = d_0 * (2 + 2 * i) = 384$$

From this you can see clearly what to write for the C++ code.

So the first **for** loop will be like this:

$$
\begin{aligned}
i &= 1 \\
d_1 &= 4 \\
d_0 &= 8 \\
mc[0] &= 1 \\
v[0] &= 3 \\
v[1] &= 5 \\
&********** \\
i &= 2 \\
d_1 &= 16 \\
d_0 &= 48 \\
mc[0] &= 8 \\
v[0] &= 15 \\
v[1] &= 40 \\
v[2] &= 33 \\
&********** \\
i &= 3 \\
d_1 &= 96 \\
d_0 &= 384 \\
mc[0] &= 55 \\
mc[1] &= 73 \\
v[0] &= 105 \\
v[1] &= 385 \\
v[2] &= 511 \\
v[3] &= 279 \\
&********** \\
i &= 4 \\
d_1 &= 768 \\
d_0 &= 3840 \\
mc[0] &= 490 \\
mc[1] &= 896 \\
mc[2] &= 790 \\
v[0] &= 945 \\
v[1] &= 4410 \\
v[2] &= 8064 \\
v[3] &= 7110 \\
v[4] &= 2895
\end{aligned}
$$

The next two **for** loops won't be too hard to comprehend, we include the variable *sgn* to alternate the sign. Overall, to be able to detect the pattern out of this took quite some time,

and makes the author realize that integration to the power of n for basic trigonometry is related to sum / series, and also combination and Pascal' triangle. With that knowledge and logic, thus today Computer Algebra System, like SymIntegration or SymPy able to compute $\int \sec^n x \, dx$ for any integer number of n. It shows how beautiful mathematics is.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
...
       if(a.type() == typeid(Sec))
       {
       if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power case
       {
              int k = 1, l=2;
              vector<int> v={1};
              vector<int> mc={1}; // to store the middle coefficient
              Symbolic sgn = -1;
              Symbolic integral_numerator, integral_denominator;
              Symbolic d0 = 2, d1 = 2;
              int j =1;
              int m = bpower-(0.5*(bpower+1));
              int last_coeff;
              int first_coeff = 3;

              // For the coefficient at the numerator of sine with odd power
              for(int i = 1 ; i < (bpower-1)/2 ; i = i+1)
              {
                     if (i >= 2)
                     {
                            for(int ic = 1 ; ic < i ; ic = ic+1)
                            {
                                   mc[ic-1] = v[ic-1] + v[ic];
                            }
                     }
                     k = k*l;
                     last_coeff = v[j-1];
                     v[0] = v[0]*(first_coeff+2*(i-1));
                     v.assign({v[0]});

                     for(int ic = 1 ; ic < i ; ic = ic+1)
                     {
                            v.push_back(mc[ic-1]*(first_coeff+2*(i-1)));
                     }
                     d1 = d1*(2*i);
                     d0 = d0*(2+2*i);
                     v.push_back(last_coeff*(first_coeff+2*(i-1))+k);
                     l = l+2;
                     j = j+1;
```

```
            }
            int j_num = 0 ;
            for(int i = bpower-2 ; i >= 1 ; i = i-2)
            {
                  integral_numerator += sgn*v[j_num]*((sin(s))^(i));
                  sgn = -sgn;
                  j_num = j_num+1;
            }
            sgn = 1;
            j = 1;
            for(int i = bpower-1 ; i >= 0 ; i = i-2)
            {
                  integral_denominator += sgn*d0*combinations(m,j-1)*((
                      sin(s))^(i));
                  sgn = -sgn;
                  j = j+1;
            }
            d1 = d1*(bpower-1);
            return (-v[0]*ln(sin(s)-1))/(d1) + (v[0]*ln(sin(s)+1))/(d1) +
                (integral_numerator)/(integral_denominator);
      }
      }
```

Code 14: *level 3 integral for secant odd case*

iv. $\int \csc^n x \, dx$

These are the formula of $\int \csc^n x \, dx$ with $n = 2$ to $n = 8$.

$$\int \csc^2 x \, dx = -\frac{\cos x}{\sin x}$$

$$\int \csc^3 x \, dx = \frac{\log(\cos(x) - 1)}{4} - \frac{\log(\cos(x) + 1)}{4} + \frac{\cos x}{2\cos^2 x - 2}$$

$$\int \csc^4 x \, dx = -\frac{2\cos x}{3\sin x} - \frac{\cos x}{3\sin^3 x}$$

$$\int \csc^5 x \, dx = \frac{3\cos^3 x - 5\cos x}{8\cos^4 x - 16\cos^2 x + 8} + \frac{3\log(\cos(x) - 1)}{16} - \frac{3\log(\cos(x) + 1)}{16}$$

$$\int \csc^6 x \, dx = -\frac{8\cos x}{15\sin x} - \frac{4\cos x}{15\sin^3 x} - \frac{\cos x}{5\sin^5 x}$$

$$\int \csc^7 x \, dx = -\frac{-15\cos^5 x + 40\cos^3 x - 33\cos x}{48\cos^6 x - 144\cos^4 x + 144\cos^2 x - 48} + \frac{5\log(\cos(x) - 1)}{32} - \frac{5\log(\cos(x) + 1)}{32}$$

$$\int \csc^8 x \, dx = -\frac{16\cos x}{35\sin x} - \frac{8\cos x}{35\sin^3 x} - \frac{6\cos x}{35\sin^5 x} - \frac{\cos x}{7\sin^7 x}$$

The pattern for the numerator and denominator will be like the secant counterpart, thus we only need to exchange the sign, what is $-$ become $+$ and the other way around, and also exchanging sine into cosine and cosine into sine.

```
...
```

```
Symbolic Power::integrate(const Symbolic &s) const
{
       ...
       if(a.type() == typeid(Csc))
       {
              ...

              if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power case
              {
                     int v[999];
                     v[0] = 1;
                     Symbolic integral;
                     Symbolic d0 = 1;
                     int k = 1, l = 2, c=1;
                     for(int i = 1 ; i < bpower ; i = i+2) // to compute the
                         denominator
                     {
                            d0 *= i;
                     }
                     for(int i = 1 ; i < bpower — 1; i = i+2)
                     {
                            c = v[0];
                            int arrsec[999]; // make the size of the array as
                                 big as possible

                            for(int j = 0 ; j < i—1 ; j = j+1)
                            {
                                   arrsec[j] = v[j];
                            }
                            int d;
                            for(int j = 1 ; j < i ; j = j+1)
                            {
                                   d = arrsec[j—1];
                                   v[j] = d*l;
                            }

                            v[0] = c*k;
                            v[1] = c*l;

                            k= k + 2;
                            l = l + 2;
                     }

                     int j_d = 1 ;
                     for(int i = 1 ; i < (0.5*bpower)+1; i = i+1)
                     {
                            integral —= ( v[i—1] * cos(s) ) / ( d0*(sin(s)^(
```

```
                                bpower-j_d)) ) ;
                        j_d = j_d+2;
                }
                return integral;
        }
    }
}
```

Code 15: *level 3 integral for cosecant even case*

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
    ...
    if(a.type() == typeid(Csc))
    {
    ...

    if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power case
    {
        int k = 1, l=2;
        vector<int> v={1};
        vector<int> mc={1}; // to store the middle coefficient
        Symbolic sgn = -1;
        Symbolic integral_numerator, integral_denominator;
        Symbolic d0 = 2, d1 = 2;
        int j =1;
        int m = bpower-(0.5*(bpower+1));
        int last_coeff;
        int first_coeff = 3;

        // For the coefficient at the numerator of cosine with odd
            power
        for(int i = 1 ; i < (bpower-1)/2 ; i = i+1)
        {
            if (i >= 2)
            {
                for(int ic = 1 ; ic < i ; ic = ic+1)
                {
                    mc[ic-1] = v[ic-1] + v[ic];
                }
            }
            k = k*l;
            last_coeff = v[j-1];
            v[0] = v[0]*(first_coeff+2*(i-1));
            v.assign({v[0]});

            for(int ic = 1 ; ic < i ; ic = ic+1)
```

```
                    {
                           v.push_back(mc[ic-1]*(first_coeff+2*(i-1)));
                    }
                    d1 = d1*(2*i);
                    d0 = d0*(2+2*i);
                    v.push_back(last_coeff*(first_coeff+2*(i-1))+k);
                    l = l+2;
                    j = j+1;
             }
             int j_num = 0 ;
             for(int i = bpower-2 ; i >= 1 ; i = i-2)
             {
                    integral_numerator += sgn*v[j_num]*((cos(s))^(i));
                    sgn = -sgn;
                    j_num = j_num+1;
             }
             sgn = 1;
             j = 1;
             for(int i = bpower-1 ; i >= 0 ; i = i-2)
             {
                    integral_denominator += sgn*d0*combinations(m,j-1)*((
                        cos(s))^(i));
                    sgn = -sgn;
                    j = j+1;
             }
             d1 = d1*(bpower-1);
             return (v[0]*ln(cos(s)-1))/(d1) - (v[0]*ln(cos(s)+1))/(d1) -
                 (integral_numerator)/(integral_denominator);
      }
      }
}
```

Code 16: *level 3 integral for cosecant odd case*

v. $\int \cot^n x \, dx$

These are the formula of $\int \cot^n x \, dx$ with $n = 2$ to $n = 8$.

$$\int \cot^2 x \, dx = -x - \frac{\cos\ x}{\sin\ x}$$
$$\int \cot^3 x \, dx = -\log(\sin\ (x)) - \frac{1}{2\sin^2\ x}$$
$$\int \cot^4 x \, dx = x + \frac{\cos\ x}{\sin\ x} - \frac{\cos^3\ x}{3\sin^3\ x}$$
$$\int \cot^5 x \, dx = \log(\sin\ (x)) + \frac{4\sin^2\ x - 1}{4\sin^4\ x}$$
$$\int \cot^6 x \, dx = -x - \frac{\cos\ x}{\sin\ x} + \frac{\cos^3\ x}{3\sin^3\ x} - \frac{\cos^5\ x}{5\sin^5\ x}$$
$$\int \cot^7 x \, dx = -\log(\sin\ (x)) - \frac{18\sin^4\ x - 9\sin^2\ x + 2}{12\sin^6\ x}$$
$$\int \cot^8 x \, dx = x + \frac{\cos\ x}{\sin\ x} - \frac{\cos^3\ x}{3\sin^3\ x} + \frac{\cos^5\ x}{5\sin^5\ x} - \frac{\cos^7\ x}{7\sin^7\ x}$$

Case 1: For $\int \cot^n x \, dx$ with even n
We will cluster only the even power

$$\int \cot^2 x \, dx = -x - \frac{\cos\ x}{\sin\ x}$$
$$\int \cot^4 x \, dx = x + \frac{\cos\ x}{\sin\ x} - \frac{\cos^3\ x}{3\sin^3\ x}$$
$$\int \cot^6 x \, dx = -x - \frac{\cos\ x}{\sin\ x} + \frac{\cos^3\ x}{3\sin^3\ x} - \frac{\cos^5\ x}{5\sin^5\ x}$$
$$\int \cot^8 x \, dx = x + \frac{\cos\ x}{\sin\ x} - \frac{\cos^3\ x}{3\sin^3\ x} + \frac{\cos^5\ x}{5\sin^5\ x} - \frac{\cos^7\ x}{7\sin^7\ x}$$

The patterns that can be seen from those equations above are

- The variable x is only alternating sign at every iteration.
- The numerator' coefficient with cosine function of odd power is only 1.
- The denominator coefficient is the same as the power of the sine function in the denominator.
- The integral computed at the previous iteration alternates its sign at the next iteration minus the new function with power of $n - 1$.

The C++ code is an easy one here, the simplest out of all level 3 trigonometry integral, it alternates the whole sign at every iteration. We will use an increasing **for** loop with $i = 2, i <= n, i = i + 2$.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
      ...
      if(a.type() == typeid(Cot))
      {
```

```
        ...
        if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power case
        {
                Symbolic integral;
                Symbolic integral_front;
                Symbolic sgn = -1;
                for(int i = 2 ; i <= (bpower) ; i = i+2)
                {
                        integral = (-1)*integral;
                        integral += - (cos(s)^(i-1)) / ((i-1)*(sin(s)^(i-1)));
                }

                for(int i = 1 ; i <= (bpower)/2 ; i = i+1)
                {
                        integral_front = sgn;
                        sgn = -sgn;
                }
                return integral_front*s + integral;
        }
}
```

Code 17: *level 3 integral for cotangent even case*

Case 2: For $\int \cot^n x\, dx$ with odd n
We will cluster only the odd power

$$\int \cot^3 x\, dx = -\log(\sin(x)) - \frac{1}{2\sin^2 x}$$
$$\int \cot^5 x\, dx = \log(\sin(x)) + \frac{4\sin^2 x - 1}{4\sin^4 x}$$
$$\int \cot^7 x\, dx = -\log(\sin(x)) - \frac{18\sin^4 x - 9\sin^2 x + 2}{12\sin^6 x}$$
$$\int \cot^9 x\, dx = \log(\sin(x)) + \frac{48\sin^4 x - 36\sin^4 x + 16\sin^2 x - 3}{24\sin^8 x}$$
$$\int \cot^{11} x\, dx = -\log(\sin(x)) - \frac{300\sin^8 x - 300\sin^6 x + 200\sin^4 x - 75\sin^2 x + 12}{120\sin^{10} x}$$

The patterns that can be seen from those equations above are

- The $\log(\sin(x))$ function is only alternating the sign at every iteration.
- The denominator for the sine function with power of $n-1$ is making a pattern like this (d_0 is the variable to represent the denominator)

$$d_0 = d_0 * \left(\frac{n-1}{2}\right)$$

We start with $d_0 = 2$ then

$$d_0 = 2 * \left(\frac{5-1}{2}\right) = 4$$
$$d_0 = 4 * \left(\frac{7-1}{2}\right) = 12$$

$$d_0 = 12 * \left(\frac{9-1}{2}\right) = 48$$
$$d_0 = 48 * \left(\frac{11-1}{2}\right) = 240$$

Knowing that the denominator has been simplified then we know that we have to adjust the numerator so they become the raw numbers like this:

$$\int \cot^3 x \, dx = -\log(\sin(x)) - \frac{1}{2\sin^2 x}$$
$$\int \cot^5 x \, dx = \log(\sin(x)) + \frac{4\sin^2 x - 1}{4\sin^4 x}$$
$$\int \cot^7 x \, dx = -\log(\sin(x)) - \frac{18\sin^4 x - 9\sin^2 x + 2}{12\sin^6 x}$$
$$\int \cot^9 x \, dx = \log(\sin(x)) + \frac{96\sin^4 x - 72\sin^4 x + 32\sin^2 x - 6}{48\sin^8 x}$$
$$\int \cot^{11} x \, dx = -\log(\sin(x)) - \frac{600\sin^8 x - 600\sin^6 x + 400\sin^4 x - 150\sin^2 x + 24}{120\sin^{10} x}$$

It is very important to do this in order to know the pattern for the numerator.

- To be able to know the pattern for the numerator with sine function of even power we can easily put the numerator coefficients like this for a better look:

$$\begin{array}{ccccccccccc} & & & & & 1 & & & & & \\ & & & & 1 & & 4 & & & & \\ & & & 2 & & 9 & & 18 & & & \\ & & 6 & & 32 & & 72 & & 96 & & \\ & 24 & & 150 & & 400 & & 600 & & 600 & \end{array}$$

The rule of thumb is we have to vectorize the coefficients, making it into a vector of size $\frac{n-1}{2}$ at every iteration. Back to the reduction formula, if we want to compute $\int \cot^{11} x \, dx$ we will start from $\int \cot^1 x \, dx$ then $\int \cot^3 x \, dx$, this is the recursive method.

When we create the vector at certain iteration, we will realize that to know the entries of this vector, we will need the values from certain variables that are having pattern, e.g. increasing at every iteration or decreasing at every iteration or multiplying with increasing coefficient. Now we take a look again. At the first iteration we have a vector $\mathbf{v}$ with only has one entry / size of 1 which is

$$\mathbf{v}[0] = 1$$

then at the next iteration we have a vector of size 2 with entries

$$\mathbf{v}[0] = 1$$
$$\mathbf{v}[1] = 4$$

then at the next iteration we have a vector of size 3 with entries

$$\mathbf{v}[0] = 2$$
$$\mathbf{v}[1] = 9$$
$$\mathbf{v}[2] = 18$$

We will use 4 vectors this time, since this one is quite tricky compared to the secant case. The vectors are: $\mathbf{v}, \mathbf{v}_{temp}, \mathbf{mc}, \mathbf{mc}_{temp}$.

We also need to use variables to help compute the coefficients with their initial value before the **for** loop :

$$d_0 = 2$$

$$k = 1$$

$$l = 2$$

We will use an increasing **for** loop with $i = 1, i <= \frac{n-1}{2}, i = i + 1$ because the size of the vector is getting bigger at every iteration, the size of the vector is the same as the iteration number.

You can and probably should write it on paper so you can learn more. To be simple, after several days of trial and error, the formula is shown to be like this:

$$\begin{aligned}
\mathbf{v}[0]_i &= \mathbf{v}[0]_{i-1} * (i-1) * \mathbf{mc}[0]_i \\
\mathbf{v}[j]_i &= [\mathbf{v}[j-1]_{i-1} * (2 * i - j)] + [\mathbf{v}[0]_i * (\mathbf{mc}[j]_i)] \\
\mathbf{v}\left[\frac{n-1}{2}\right]_i &= \left[\mathbf{v}\left[\frac{n-1}{2} - 1\right]_{i-1} * (2 * i - j)\right] + (\mathbf{v}[0]_i * *\mathbf{mc}\left[\frac{n-1}{2}\right]_i)
\end{aligned}$$

with i is the iteration $i = 1, 2, \ldots, \frac{n-1}{2}$, and $j = 1, 2, \ldots, \frac{n-1}{2}$. The vector **mc**, the middle coefficient is the Pascal' triangle

$$\begin{array}{ccccccccccccc}
 & & & & & & 1 & & & & & & \\
 & & & & & 1 & & 1 & & & & & \\
 & & & & 1 & & 2 & & 1 & & & & \\
 & & & 1 & & 3 & & 3 & & 1 & & & \\
 & & 1 & & 4 & & 6 & & 4 & & 1 & & \\
 & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & \\
1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1
\end{array}$$

the top row represents $i = 1$ for $\int \cot^3 x \, dx$, and this vector **mc** helps computing the numerator coefficient.

The computation / coding of C++ for the numerator coefficients will be a tough one, it takes an increasing **for** loop with $i = 1, i <= \frac{n-1}{2}, i = i + 1$ and we split it into several cases.

So if we want to compute $\int \cot^{11} x \, dx$ we will gain the numerator coefficients at the iteration $i = 5$.

So the first **for** loop will be like this:

$$
\begin{aligned}
i &= 1 \\
d_0 &= 2 \\
mc[0] &= 1 \\
v[0] &= 1 \\
&**********\\
i &= 2 \\
d_0 &= 4 \\
mc[0] &= 1 \\
mc[1] &= 1 \\
v[0] &= 1 \\
v[1] &= 4 \\
&**********\\
i &= 3 \\
d_0 &= 12 \\
mc[0] &= 1 \\
mc[1] &= 2 \\
mc[1] &= 1 \\
v[0] &= 2 \\
v[1] &= 9 \\
v[2] &= 18 \\
&**********\\
i &= 4 \\
d_0 &= 48 \\
mc[0] &= 1 \\
mc[1] &= 3 \\
mc[2] &= 3 \\
mc[3] &= 1 \\
v[0] &= 6 \\
v[1] &= 32 \\
v[2] &= 72 \\
v[3] &= 96 \\
&**********
\end{aligned}
$$

$$
\begin{aligned}
i &= 5 \\
d_0 &= 240 \\
mc[0] &= 1 \\
mc[1] &= 4 \\
mc[2] &= 6 \\
mc[3] &= 4 \\
mc[4] &= 1 \\
v[0] &= 24 \\
v[1] &= 150 \\
v[2] &= 400 \\
v[3] &= 600 \\
v[4] &= 600
\end{aligned}
$$

We omitted the **mc**$[0]$ and **mc**$[i-1]$ in the C++ codes since it will only returns 1 so it won't really change much.

The rest of the **for** loops are only used to return the correct integral.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
	...
	if(a.type() == typeid(Cot))
	{
	...
	if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power case
	{
		Symbolic integral;
		Symbolic integral_front;
		Symbolic sgn = -1;
		vector<int> v={1};
		vector<int> v_temp={1};
		vector<int> mc={1}; // to store the middle coefficient
		vector<int> mc_temp={1}; // to store the temporary / new middle
		     coefficient
		int d0 = 2;
		int k = 1, l=2;
		// For the coefficient at the numerator
		for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
		{
			if (i == 1)
			{
				v[0] = 1;
				v.assign({v[0]});
			}
```

```
                    if (i ==2)
                    {
                           mc[0]=1;
                           mc.assign({mc[0]});
                           v_temp[0] = v[0]*(i-1);
                           v_temp.assign({v_temp[0]});

                           for(int j = 1 ; j < i ; j = j+1)
                           {
                                  v_temp.push_back(v[j-1]*((2*i)-j) +
                                      v_temp[0]*mc[j-1] );
                           }
                           v[0] = v_temp[0];
                           v.assign({v[0]});
                           for(int j = 1 ; j < i ; j = j+1)
                           {
                                  v.push_back(v_temp[j]);
                           }
                    }
                    if (i == 3)
                    {
                           mc[0] = mc[0] + 1; //
                           mc.assign({mc[0]});

                           v_temp[0] = v[0]*(i-1);
                           v_temp.assign({v_temp[0]});
                           for(int j = 1 ; j < i-1 ; j = j+1)
                           {
                                  v_temp.push_back(v[j-1]*((2*i)-j) +
                                      v_temp[0]*mc[j-1] );
                           }
                           v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                           // Assign the temporary vector to vector v for
                               future use
                           v[0] = v_temp[0];
                           v.assign({v[0]});
                           for(int j = 1 ; j < i ; j = j+1)
                           {
                                  v.push_back(v_temp[j]);
                           }
                    }
                    if (i == 4)
                    {
                           mc[0] = mc[0] + 1;
                           mc.assign({mc[0]});
                           mc.push_back(mc[0]);

                           v_temp[0] = v[0]*(i-1);
```

```
                        v_temp.assign({v_temp[0]});
                        for(int j = 1 ; j < i-1 ; j = j+1)
                        {
                                v_temp.push_back(v[j-1]*((2*i)-j) +
                                    v_temp[0]*mc[j-1] );
                        }
                        v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                        // Assign the temporary vector to vector v for
                            future use
                        v[0] = v_temp[0];
                        v.assign({v[0]});
                        for(int j = 1 ; j < i ; j = j+1)
                        {
                                v.push_back(v_temp[j]);
                        }
                }
                if (i >= 5)
                {
                        mc_temp[0] = mc[0]+1;
                        mc_temp.assign({mc_temp[0]});
                        for(int ic = 1 ; ic < l ; ic = ic+1)
                        {
                                mc_temp[ic] = mc[ic-1] + mc[ic];
                        }
                        mc[l] = (mc_temp[0]);
                        mc[l+1] = (mc_temp[0]);

                        mc[0]=mc_temp[0];
                        mc.assign({mc[0]});

                        for(int ic = 1 ; ic < l ; ic = ic+1)
                        {
                                mc.push_back(mc_temp[ic]);
                        }
                        mc.push_back(mc_temp[0]);
                        mc.push_back(0);

                        v_temp[0] = v[0]*(i-1);
                        v_temp.assign({v_temp[0]});
                        for(int j = 1 ; j < i-1 ; j = j+1)
                        {
                                v_temp.push_back(v[j-1]*((2*i)-j) +
                                    v_temp[0]*mc[j-1] );
                        }
                        v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                        // Assign the temporary vector to vector v for
                            future use
                        v[0] = v_temp[0];
```

```
                        v.assign({v[0]});
                        for(int j = 1 ; j < i ; j = j+1)
                        {
                                v.push_back(v_temp[j]);
                        }

                        l = l+1;
                }

                d0 = d0*k;
                k = k+1;
        }
        for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
        {
                integral += sgn*v[i-1]*(sin(s)^(2*(i-1)));
                sgn=-sgn;

        }

        for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
        {
                integral_front = sgn;
                sgn = -sgn;
        }
        return integral_front*ln(sin(s)) + (integral)/(d0*(sin(s)^(
            bpower-1))) ;
    }
}
```

Code 18: *level 3 integral for cotangent odd case*

vi. $\int \tan^n x \, dx$

These are the formula of $\int \tan^n x \, dx$ with $n = 2$ to $n = 8$.

$$\int \tan^2 x \, dx = -x + \frac{\sin\ x}{\cos\ x}$$
$$\int \tan^3 x \, dx = \log(\cos\ x) + \frac{1}{2\cos^2\ x}$$
$$\int \tan^4 x \, dx = x - \frac{\sin\ x}{\cos\ x} + \frac{\sin^3\ x}{3\cos^3\ x}$$
$$\int \tan^5 x \, dx = -\log(\cos\ x) - \frac{4\cos^2\ x - 1}{4\cos^4\ x}$$
$$\int \tan^6 x \, dx = -x + \frac{\sin\ x}{\cos\ x} - \frac{\sin^3\ x}{3\cos^3\ x} + \frac{\sin^5\ x}{5\cos^5\ x}$$
$$\int \tan^7 x \, dx = \log(\cos\ x) + \frac{18\cos^4\ x - 9\cos^2\ x + 2}{12\cos^6\ x}$$
$$\int \tan^8 x \, dx = x - \frac{\sin\ x}{\cos\ x} + \frac{\sin^3\ x}{3\cos^3\ x} - \frac{\sin^5\ x}{5\cos^5\ x} + \frac{\sin^7\ x}{7\cos^7\ x}$$

The pattern for the numerator and denominator will be like the cotangent counterpart, thus we only need to exchange the sign, what is $-$ become $+$ and the other way around, and also exchanging sine into cosine and cosine into sine.

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
	...
	if(a.type() == typeid(Cot))
	{
		...
	if(b.df(s) == 0 && b!=0 && bpower % 2 == 0) // even power case
	{
		Symbolic integral;
		Symbolic integral_front;
		Symbolic sgn = -1;
		for(int i = 2 ; i <= (bpower) ; i = i+2)
		{
			integral = (-1)*integral;
			integral += (sin(s)^(i-1)) / ((i-1)*(cos(s)^(i-1)));
		}

		for(int i = 1 ; i <= (bpower)/2 ; i = i+1)
		{
			integral_front = sgn;
			sgn = -sgn;
		}
		return integral_front*s + integral;
	}
```

```
        }
}
```

Code 19: *level 3 integral for tangent even case*

```
...

Symbolic Power::integrate(const Symbolic &s) const
{
        ...
        if(a.type() == typeid(Cot))
        {
        ...
        if(b.df(s) == 0 && b!=0 && bpower % 2 != 0) // odd power case
        {
                Symbolic integral;
                Symbolic integral_front;
                Symbolic sgn = 1;
                vector<int> v={1};
                vector<int> v_temp={1};
                vector<int> mc={1}; // to store the middle coefficient
                vector<int> mc_temp={1}; // to store the temporary / new middle
                     coefficient
                int d0 = 2;
                int k = 1, l=2;
                // For the coefficient at the numerator
                for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
                {
                        if (i == 1)
                        {
                                v[0] = 1;
                                v.assign({v[0]});
                        }

                        if (i ==2)
                        {
                                mc[0]=1;
                                mc.assign({mc[0]});
                                v_temp[0] = v[0]*(i-1);
                                v_temp.assign({v_temp[0]});

                                for(int j = 1 ; j < i ; j = j+1)
                                {
                                        v_temp.push_back(v[j-1]*((2*i)-j) +
                                            v_temp[0]*mc[j-1] );
                                }
                                v[0] = v_temp[0];
                                v.assign({v[0]});
                                for(int j = 1 ; j < i ; j = j+1)
```

```
                        {
                                v.push_back(v_temp[j]);
                        }
                }
                if (i == 3)
                {
                        mc[0] = mc[0] + 1; //
                        mc.assign({mc[0]});

                        v_temp[0] = v[0]*(i-1);
                        v_temp.assign({v_temp[0]});
                        for(int j = 1 ; j < i-1 ; j = j+1)
                        {
                                v_temp.push_back(v[j-1]*((2*i)-j) +
                                    v_temp[0]*mc[j-1] );
                        }
                        v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                        // Assign the temporary vector to vector v for
                            future use
                        v[0] = v_temp[0];
                        v.assign({v[0]});
                        for(int j = 1 ; j < i ; j = j+1)
                        {
                                v.push_back(v_temp[j]);
                        }
                }
                if (i == 4)
                {
                        mc[0] = mc[0] + 1;
                        mc.assign({mc[0]});
                        mc.push_back(mc[0]);

                        v_temp[0] = v[0]*(i-1);
                        v_temp.assign({v_temp[0]});
                        for(int j = 1 ; j < i-1 ; j = j+1)
                        {
                                v_temp.push_back(v[j-1]*((2*i)-j) +
                                    v_temp[0]*mc[j-1] );
                        }
                        v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                        // Assign the temporary vector to vector v for
                            future use
                        v[0] = v_temp[0];
                        v.assign({v[0]});
                        for(int j = 1 ; j < i ; j = j+1)
                        {
                                v.push_back(v_temp[j]);
                        }
```

```
                }
                if (i >= 5)
                {
                        mc_temp[0] = mc[0]+1;
                        mc_temp.assign({mc_temp[0]});
                        for(int ic = 1 ; ic < l ; ic = ic+1)
                        {
                                mc_temp[ic] = mc[ic-1] + mc[ic];
                        }
                        mc[l] = (mc_temp[0]);
                        mc[l+1] = (mc_temp[0]);

                        mc[0]=mc_temp[0];
                        mc.assign({mc[0]});

                        for(int ic = 1 ; ic < l ; ic = ic+1)
                        {
                                mc.push_back(mc_temp[ic]);
                        }
                        mc.push_back(mc_temp[0]);
                        mc.push_back(0);

                        v_temp[0] = v[0]*(i-1);
                        v_temp.assign({v_temp[0]});
                        for(int j = 1 ; j < i-1 ; j = j+1)
                        {
                                v_temp.push_back(v[j-1]*((2*i)-j) +
                                    v_temp[0]*mc[j-1] );
                        }
                        v_temp.push_back((v[i-2]*(i+1) ) + (v_temp[0]) );
                        // Assign the temporary vector to vector v for
                            future use
                        v[0] = v_temp[0];
                        v.assign({v[0]});
                        for(int j = 1 ; j < i ; j = j+1)
                        {
                                v.push_back(v_temp[j]);
                        }

                        l = l+1;
                }

                d0 = d0*k;
                k = k+1;
        }
        for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
        {
                integral += sgn*v[i-1]*(cos(s)^(2*(i-1)));
```

```
                    sgn=-sgn;

                }

                for(int i = 1 ; i <= (bpower-1)/2 ; i = i+1)
                {
                    integral_front = sgn;
                    sgn = -sgn;
                }
                return integral_front*ln(cos(s)) + (integral)/(d0*(cos(s)^(
                    bpower-1))) ;
            }
            }
}
```

Code 20: *level 3 integral for tangent odd case*

- Trigonometry Level 4

[SI*] The trigonometry integrals that we consider as level 4 is

$$\int \sin^n x \ \cos^m \ xdx$$

Chapter 3

SymIntegration to Solve Ordinary Differential Equations for Engineering Problems

"After Odin's defeat atop Yggdrasil and the revelation of Lezard's true nature, Freya quickly teleports there in a frantic state, telling Odin that Lezard's presence was the distortion she felt at Dipan" - ***Freya (Valkyrie Profile 2: Silmeria)***

"Everyone's favorite fighting fairy godmother is still kicking, taking out Ether Strike hits for her godfather Odin. The heretofore unmatched fury of her scowl derives from her lack of lines in the main story" - ***Freya (Valkyrie Profile: Covenant of the Plume)***

According to Newton's second law of motion, the acceleration a of a body of mass m is proportional to the total force F acting on it, with $1/m$ as the constant of proportionality, so that $a = F/m$ or

$$F = ma \tag{3.1}$$

[SI*] General Remarks and Solutions

The general ordinary differential equation of the nth order is

$F\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}, \right) = 0$
or, using the prime notation for derivatives,

$F(x, y, y', y'', \ldots, y^{(n)}) = 0$
It is normally a simple task to verify that a given function $y = y(x)$ is a solution of an equation above.

* All that is necessary is to compute the derivatives of $y(x)$ and to show that $y(x)$ and these derivatives, when substituted in the equation, reduce it to an identity in x.

I. First Order Ordinary Differential Equations

- Some Basic Mathematical Models

[SI*] .

Direction Fields

[SI*] The independent variables for the example above is t,

- Solutions of Some Differential Equations

[SI*] The equation

$$m\frac{dv}{dt} = mg - \gamma v$$

is of the general form

$$\frac{dy}{dt} = ay - b \tag{3.2}$$

II. Higher Order Linear Equations

- General Theory of nth Order Linear Equations

[SI*] The first
[SI*]

- The Method of Variation of Parameters

[SI*] The first
[SI*]

III. Boundary Value Problems and Sturm-Liouville Theory

- The Occurrence of Two-Point Boundary Value Problems

  [SI*] The first
  [SI*]

- Nonhomogeneous Boundary Value Problems

  [SI*] The first
  [SI*]

Bibliography

[1] Colley, Susan Jane (2012) Vector Calculus 4th Edition, Pearson, Boston, MA, United States.

[2] Dorf, Richard C., Bishop, Robert H. (2010) Modern Control Systems 12th Edition, Pearson, New Jersey, United States.

[3] Purcell, Rigdon, Varberg (2007) Calculus 9th Edition, Pearson, Upper Saddle River, New Jersey, United States.

[4] Kenneth H. Rosen (2006) Discrete Mathematics and its Applications 6th Edition, McGraw-Hill, Boston, United States.

[5] Anton, Rorres (2006) Elementary Linear Algebra with Supplemental Applications 10th Edition, Wiley, Boston, United States.

[6] Anton, Howard, Rorres, Chris, Elementary Linear Algebra with Supplemental Applications 12th edition, Wiley, Hoboken, New Jersey, United States.