

SymIntegration

FREYA · LASTHRIM · DS GLANZSCHE¹

Berlin-Sentinel Academy of Science, AliceGard

2025 Edition

¹A thank you or further information

Contents

Preface	5
1 Introduction Story	9
2 Symbolic Integral Computation with SymIntegration	15
I Build and Install SymIntegration in GFreya OS	16
II Techniques of Integration	18
III Integration by Parts	20
IV The Method of Substitution	23
V Fraction Integral	25
VI Trigonometry Integral	37
i $\int \sin^n x \, dx$	41
ii $\int \cos^n x \, dx$	45
iii $\int \sec^n x \, dx$	48
iv $\int \csc^n x \, dx$	60
v $\int \cot^n x \, dx$	64
vi $\int \tan^n x \, dx$	73
vii $\int \sin^n x \cos^m x \, dx$	77
viii $\int \tan^n x \sec^m x \, dx$	94
ix $\int \cot^n x \csc^m x \, dx$	117
VII Trigonometry and Transcendentals Formula	119
VIII Vector Calculus	122
3 SymIntegration to Solve Ordinary Differential Equations for Engineering Problems	133
I Solving First Order Ordinary Differential Equations	135
II Higher Order Linear Equations	163
III Boundary Value Problems and Sturm-Liouville Theory	164
4 Probability and Statistics Computation with SymIntegration	165
I Processing and Summarizing Data with Statistics	166
i Descriptive Statistics Computation	166
II Random Variables and Probability Distributions	167
i Discrete Probability Distributions	167
ii Continuous Probability Distributions	168
iii Joint Probability Distributions	169
iv Covariance and Vectorized Moments	175
v Linear Combinations and Transformations	175

vi	The Cholesky Decomposition and Generating Random Vectors	176
vii	Generate Random Number in SymIntegration	177
viii	Create Covariance Matrix in SymIntegration	179
ix	Create Cholesky Decomposition in SymIntegration	181
III	Discrete Probability Distributions	182
IV	Continuous Probability Distributions	188
i	Uniform Distribution	188
ii	Normal Distribution	189
iii	Gamma Distribution	192
iv	Exponential Distribution	194
v	Beta Distribution	195
vi	Chi-Squared Distribution	202
vii	<i>t</i> -Distribution	203
viii	<i>F</i> -Distribution	204
ix	Weibull Distribution and Hazard Rates	207
x	Cauchy Distribution	208
xi	Bivariate Normal	209
xii	Generate Gamma Distributed Random Number in SymIntegration and Plot the Histogram with Hamzstplot	215
V	Statistical Inference	221
i	Sampling from a Normal Population	222
ii	The Central Limit Theorem	222
iii	Point Estimation	223
iv	Confidence Interval as a Concept	223
v	Hypothesis Tests Concepts	223
vi	Bayesian Statistics	223
VI	Linear Regression and Correlation	224
i	Pearson's Correlation	224
ii	Linear Regression	226
iii	Simple Regression Line Plot and Pearson's Correlation Computation	228
iv	Multiple Linear Regression	230
v	Compute Multiple Linear Regression with SymIntegration	237
vi	Compute Inferences in Multiple Linear Regression with SymIntegration	238
VII	Hypothesis Testing	239
i	One Sample Hypothesis Test for Comparing Means	239
ii	Compute Right-Tailed Hypothesis Testing with SymIntegration	251
iii	Compute Two-Tailed Hypothesis Testing with SymIntegration	254
iv	Two Sample Hypothesis Tests for Comparing Means	255
v	One-Way Analysis of Variance (ANOVA)	255
vi	Compute One-Way ANOVA with SymIntegration	262
5	Operations Research Computation with SymIntegration	267
I	Linear Programming	267
i	Simplex Method	269
II	Nonlinear Programming	273

6 Numerical Linear Algebra with SymIntegration	275
I Solutions of Systems of Linear Equations	275
i Solve Nonhomogeneous System of Linear Equation with Gaussian Elimination in SymIntegration	279
II Determinant	283
i Compute Determinant of a Square Matrix with SymIntegration	285
III Euclidean Vector Spaces	287
i Compute Norm, Dot Product, Angle with SymIntegration	293
ii Compute Orthogonal Projection on a Line with SymIntegration	294
iii Compute Vector and Parametric Equations of a Plane in \mathbb{R}^3 with SymIntegration	296
IV General Vector Spaces	297
V Eigenvalues and Eigenvectors	298
VI Inner Product Spaces	298
VII Diagonalization and Quadratic Forms	299
VIII Linear Transformations	299
IX Numerical Methods	299
i LU-Decomposition	299
ii Solve Nonhomogeneous System of Linear Equation with LU-Decomposition in SymIntegration	302
iii Cholesky Decomposition	307

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.

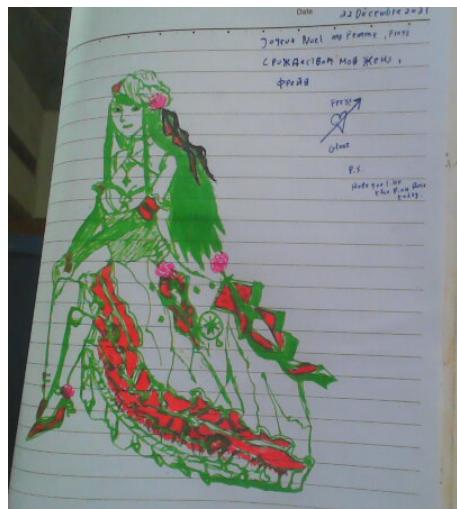


Figure 2: I paint her 3 days before Christmas in 2021.

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 [8]. 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 skyscraper, 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:

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

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

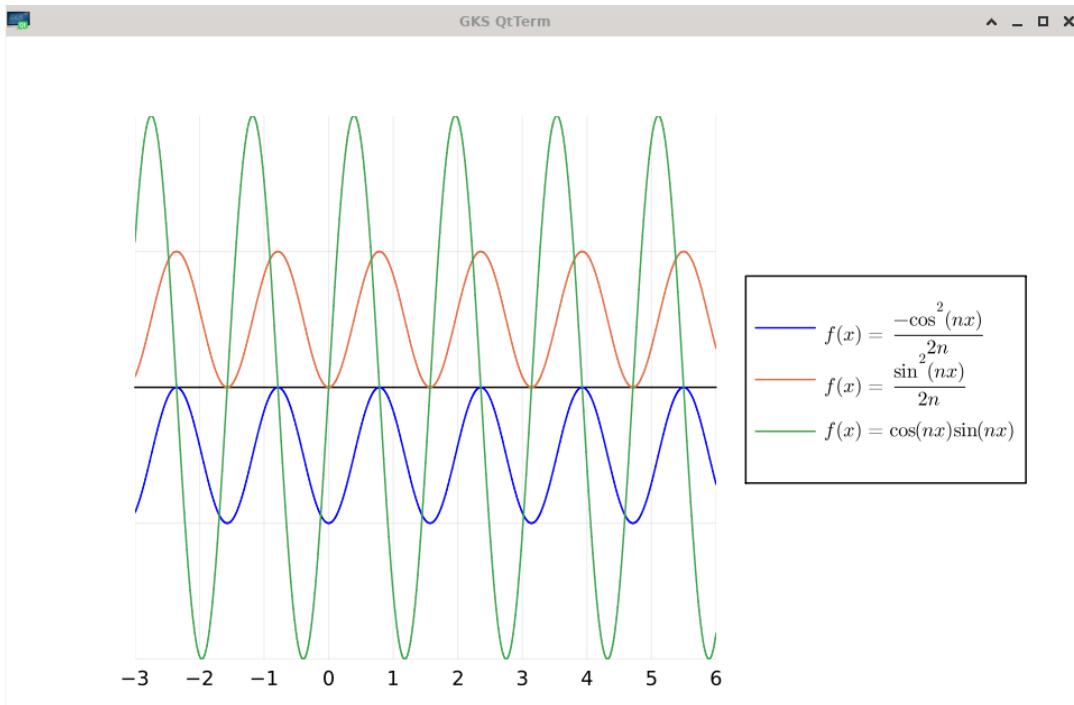


Figure 1.1: Integrating a function with different method can result in at least two different functions.

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 engineering:

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 $\sin(\ln(ax + b)), \cos(\ln(ax + b)), \tan(\ln(ax + b)), \cot(\ln(ax + b)), \sec(\ln(ax + b)), \csc(\ln(ax + b))$ with $ax + b$ is a polynomial of order 1.
4. The integral and derivative of all hyperbolic trigonometry functions $\sinh(ax + b), \cosh(ax + b), \tanh(ax + b), \coth(ax + b), \sech(ax + b), \csch(ax + b)$.
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 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. Feedbacks can be sent to dsglanksche@gmail.com.

Chapter 2

Symbolic Integral Computation with SymIntegration

"You can't direct the wind, but you can adjust your sails." - Sailing

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 SYMINTTEGRATION 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.

In the beginning (from SymbolicC++ 3.35), 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.s

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

$$\frac{d}{dx} [f(x)g(x)] = f(x)\frac{dg(x)}{dx} + g(x)\frac{df(x)}{dx} \quad (2.1)$$

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

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

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

$$\begin{aligned} E(X) &= \int_{-\infty}^{\infty} x f(x) dx \\ &= \int_{-\infty}^0 x \cdot 0 dx + \int_0^{\infty} x \lambda e^{-\lambda x} dx \end{aligned}$$

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

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

[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

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

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 [8].

[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**, we still use the function **integrate** for this type of integral.

```
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

$$f(x) = (ax^2 + bx + c)^d \quad (2.4)$$

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

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

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 type 1

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

$$f(x) = \frac{a_1 x}{ax^2 + bx + c}$$

a_1, a, b , and c are constants.

The integral will be

$$\begin{aligned}
\int \frac{a_1 x}{ax^2 + bx + c} dx &= a_1 \left[\left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) * \right. \\
&\quad \ln \left(x + \frac{-4ac \left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + b^2 \left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + 2c}{b} \right) \\
&\quad + \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) * \\
&\quad \left. \ln \left(x + \frac{-4ac \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + b^2 \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + 2c}{b} \right) \right] + C
\end{aligned} \tag{2.6}$$

The codes for the SymIntegration to handle this integral is located in `src/integrate.cpp`, instead of `integrate` we create a new function `fraction integrate` to handle this type of integral because this type of integral cannot be read nicely with `Symbolic Power::integrate(const Symbolic &s) const` that is in `src/functions.cpp`.

By July 8th, 2025, the function `fraction integrate` already has around 190 lines of code.

...

```

Symbolic fractionintegrate(const Symbolic &fnum, const Symbolic &fdenom,
    const Symbolic &x)
{
    list<Equations> eq;
    list<Equations>::iterator i;
    UniqueSymbol anum, bnum;
    Symbolic integral_sol;
    double anump, bnump, ap, bp, cp;
    ap = fdnom.coeff(x,2);
    bp = fdnom.coeff(x,1);
    cp = fdnom.coeff(x,0);

    ...

    ...

    if (fnum.coeff(x,2)==0 && fnum.coeff(x,1)!=0 && fnum.coeff(x,0)==0
        && fdnom.coeff(x,2)!=0 && fdnom.coeff(x,1)!=0 && fdnom.
        coeff(x,0)!=0 ) // for (a1x)/(ax^2+bx+c)
    {
        eq = ( (bnum*x)).match(fnum, (bnum,cnum));
        for(i=eq.begin(); i!=eq.end(); ++i)
        try {
            Symbolic a1 = rhs(*i, bnum);
            bnump= a1;
        } catch(const SymbolicError &se) {}

        double D = (bp*sqrt(bp*bp-4*ap*cp))/(2*ap*(4*ap*cp-(bp*bp
            )) );
        integral_sol = bnump*( (-D + (1/(2*ap)))*ln(x+(-4*ap*cp
            *(-D + 1/(2*ap)) +bp*bp*(-D+ 1/(2*ap)) + 2*cp)/(bp))
            +(D + (1/(2*ap)))*ln(x+(-4*ap*cp*(D + 1/(2*ap)) +bp*
            bp*(D+ 1/(2*ap)) + 2*cp)/(bp)) );
    }

    ...

    return integral_sol;
}
...
...

```

Code 8: fraction level 2 integral default type

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

$$f(x) = \frac{a_1 x}{ax^2 + bx}$$

a_1, a , and b are constants.

The integral will be

$$\int \frac{a_1 x}{ax^2 + bx} dx = \frac{a_1 \ln(ax + b)}{a} + C \quad (2.7)$$

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

$$f(x) = \frac{a_1 x}{ax^2 + c}$$

a_1, a and c are constants.

The integral will be

$$\int \frac{a_1 x}{ax^2 + c} dx = \frac{a_1 \ln(ax^2 + c)}{2a} + C \quad (2.8)$$

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

$$f(x) = \frac{a_1 x}{bx + c}$$

a_1, b , and c are constants.

The integral will be

$$\int \frac{a_1 x}{bx + c} dx = a_1 \left(\frac{x}{b} - \frac{c \ln(bx + c)}{b^2} \right) + C \quad (2.9)$$

- Fraction Level 3

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

$$f(x) = \frac{a_1 x + b_1}{ax^2 + bx + c}$$

a_1, b_1, a, b and c are constants.

The integral will be

$$\begin{aligned} \int \frac{a_1 x + b_1}{ax^2 + bx + c} dx &= \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) * \\ &\quad \ln \left(x + \frac{4ac \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) - 2a_1 c - b^2 \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) + bb_1}{2ab_1 - a_1 b} \right) \\ &\quad + \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) * \\ &\quad \ln \left(x + \frac{4ac \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) - 2a_1 c - b^2 \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) + bb_1}{2ab_1 - a_1 b} \right) + C \end{aligned} \quad (2.10)$$

The codes for the SymIntegration to handle this integral is located in `src/integrate.cpp`, instead of `integrate` we create a new function `fraction integrate` to handle this type of integral.

```

...
Symbolic fractionintegrate(const Symbolic &fnum, const Symbolic &fdenom,
                           const Symbolic &x)
{
    list<Equations> eq;
    list<Equations>::iterator i;
    UniqueSymbol anum, bnum;
    Symbolic integral_sol;
    double anump, bnump, ap, bp, cp;
    ap = fdenom.coeff(x,2);
    bp = fdenom.coeff(x,1);
    cp = fdenom.coeff(x,0);

    ...

    if (fnum.coeff(x,2)==0 && fnum.coeff(x,1)!=0 && fnum.coeff(x,0)!=0
        && fdenom.coeff(x,2)!=0 && fdenom.coeff(x,1)!=0 && fdenom.
        coeff(x,0)!=0 ) // for (a1x+b1)/(ax^2+bx+c)
    {
        eq = ( (bnum*x+cnum)).match(fnum, (bnum,cnum));
        for(i=eq.begin(); i!=eq.end(); ++i)
        try {
            Symbolic a1 = rhs(*i, bnum), b1 = rhs(*i, cnum);
            bnump= a1;
            cnump = b1;
        } catch(const SymbolicError &se) {}

        double D = (2*ap*cnump - bnump*bp)*sqrt(-4*ap*cp+(bp*bp))
                  /(2*ap*(4*ap*cp-(bp*bp)));
        double denom1 = 2*ap*cnump - bnump*bp;

        integral_sol = ((bnump/(2*ap)) - D)*ln(x + (4*ap*cp*((bnump/(2*ap))-D) - 2*bnump*cp -(bp*bp)*(bnump/(2*ap))-D) + bp*cnump)/(denom1)) + ((bnump/(2*ap)) + D)*ln(x + (4*ap*cp*((bnump/(2*ap))+D) - 2*bnump*cp -(bp*bp)*(bnump/(2*ap) + D) + bp*cnump)/(denom1)) ;
    }

    ...

    return integral_sol;
}
...

```

Code 9: fraction level 3 integral default type showstringspaces

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

$$f(x) = \frac{a_1x + b_1}{ax^2 + bx}$$

a_1, b_1, a , and b are constants.

The integral will be

$$\int \frac{a_1x + b_1}{ax^2 + bx} dx = \frac{b_1 \ln(x)}{b} - \frac{(ab_1 - a_1b) \ln \left(x + \frac{bb_1 + \frac{b(ab_1 - a_1b)}{a}}{2ab_1 - a_1b} \right)}{ab} + C \quad (2.11)$$

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

$$f(x) = \frac{a_1x + b_1}{ax^2 + c}$$

a_1, b_1, a , and c are constants.

The integral will be

$$\begin{aligned} \int \frac{a_1x + b_1}{ax^2 + c} dx &= \left(\frac{a_1}{2a} - \frac{b_1\sqrt{-a^3c}}{2a^2c} \right) * \\ &\quad \ln \left(x + \frac{2ac \left(\frac{a_1}{2a} - \frac{b_1\sqrt{-a^3c}}{2a^2c} \right) - a_1c}{ab_1} \right) \\ &\quad + \left(\frac{a_1}{2a} + \frac{b_1\sqrt{-a^3c}}{2a^2c} \right) * \\ &\quad \ln \left(x + \frac{2ac \left(\frac{a_1}{2a} + \frac{b_1\sqrt{-a^3c}}{2a^2c} \right) - a_1c}{ab_1} \right) + C \end{aligned} \quad (2.12)$$

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

$$f(x) = \frac{a_1x + b_1}{bx + c}$$

a_1, b_1, b , and c are constants.

The integral will be

$$\int \frac{a_1x + b_1}{bx + c} dx = \frac{a_1x}{b} - \frac{(a_1c - bb_1) \ln(bx + c)}{b^2} + C \quad (2.13)$$

- Fraction Level 4

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

$$f(x) = \frac{a_1x^2 + b_1x + c_1}{ax^2 + bx + c}$$

a_1, b_1, c_1, a, b and c are constants.

The integral will be

$$\begin{aligned} \int \frac{a_1x^2 + b_1x + c_1}{ax^2 + bx + c} dx &= \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) * \\ &\quad \ln \left[x + \frac{4a^2c \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right)}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right. \\ &\quad \left. - \frac{ab^2 \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) + abc_1 - 2ab_1c + a_1bc}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right] \\ &\quad + \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) * \\ &\quad \ln \left[x + \frac{4a^2c \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right)}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right. \\ &\quad \left. - \frac{ab^2 \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) + abc_1 - 2ab_1c + a_1bc}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right] \\ &\quad + \frac{a_1x}{a} + C \end{aligned} \tag{2.14}$$

The codes for the SymIntegration to handle this integral is located in `src/integrate.cpp`, we use a new function `fraction integrate` to handle this type of integral.

```

...
Symbolic fractionintegrate(const Symbolic &fnum, const Symbolic &fdenom,
                           const Symbolic &x)
{
    list<Equations> eq;
    list<Equations>::iterator i;
    UniqueSymbol anum, bnum, cnum;
    Symbolic integral_sol;
    double anump, bnumpp, cnumpp, ap, bp, cp;
    ap = fdnom.coeff(x,2);
    bp = fdnom.coeff(x,1);
    cp = fdnom.coeff(x,0);

    if (fnum.coeff(x,2)!=0 && fnum.coeff(x,1)!=0 && fnum.coeff(x,0)!=0
        && fdnom.coeff(x,2)!=0 && fdnom.coeff(x,1)!=0 && fdnom.
        coeff(x,0)!=0 ) // for (a1x^2 + b1x + c1)/(ax^2+bx+c)
```

```

{
    eq = ( (anum*x*x+bnum*x+cnum)).match(fnum, (anum,bnum,cnum)
        );
    for(i=eq.begin(); i!=eq.end(); ++i)
    try {
        Symbolic a1 = rhs(*i, anum), b1 = rhs(*i, bnum), c1
            = rhs(*i, cnum);
        anump= a1;
        bnump= b1;
        cnump = c1;
    } catch(const SymbolicError &se) {}

    double D = sqrt(bp*bp-(4*ap*cp))*(2*ap*ap*cnump-2*ap*
        anump*cp-ap*bp*bnump+anump*bp*bp)/(2*ap*ap*(4*ap*cp-(

        bp*bp))) ;

    integral_sol = ((ap*bnump-anump*bp)/(2*ap*ap) - D)*ln(x
        +(4*ap*ap*cp*((ap*bnump-anump*bp)/(2*ap*ap) - D) -
        ap*bp*bp*((ap*bnump-anump*bp)/(2*ap*ap) - D) + ap*bp*
        cnump-2*ap*bnump*cp+anump*bp*cp) / (2*ap*ap*cnump-2*
        ap*anump*cp-ap*bp*bnump+anump*bp*bp)) + ((ap*bnump-
        anump*bp)/(2*ap*ap) + D)*ln(x+(4*ap*ap*cp*((ap*bnump-
        anump*bp)/(2*ap*ap) + D) - ap*bp*bp*((ap*bnump-anump*
        bp)/(2*ap*ap) + D) + ap*bp*cnump-2*ap*bnump*cp+anump*


        bp*cp) / (2*ap*ap*cnump-2*ap*anump*cp-ap*bp*bnump+
        anump*bp*bp)) + (anump*x)/ap;
    }

    ...
}

```

Code 10: fraction level 4 integral default type

[SI*] The key to solve integral of fraction integral is to use the method of partial fraction decomposition, which allows us to decompose rational functions into sums of simpler, more easily integrated rational functions, for example if we have

$$\int \frac{P_n(x)}{P_m(x)}$$

with $P_n(x)$ is a polynomial with x as the independent variable and highest order of n , $P_m(x)$ is a polynomial with x as the independent variable and highest order of m .

The partial fraction decomposition can be applied to a rational function

$$\frac{P_n(x)}{P_m(x)}$$

only if $n < m$.

For the case where $n \geq m$, it can be tricked by first we must perform long division to rewrite the quotient into a rational function that can fit the bill to be performed the

partial fraction decomposition

$$\frac{P_n(x)}{P_m(x)} = B(x) + \frac{P_c(x)}{P_d(x)}$$

where $c < d$.

From the previous part we can see that a solid formula to compute the fraction integral can be very long enough, with $m = 2$, it comes from deriving and generalizing this method of partial fraction decomposition by changing the constants into coefficients like a_1, b_1, c_1, a, b, c .

- [List of Solvable Fraction Integral](#)

We will list all the fractions that can be solved with **fractionintegrate** function in SymInte-

gration here, they are:

$$\begin{aligned}
 \int \frac{1}{1+x^2} dx &= \arctan(x) \\
 \int \frac{ax}{1+x^2} dx &= \frac{a \ln(x^2 + 1)}{2} \\
 \int \frac{1}{ax^2 + bx + c} dx &= -\sqrt{\frac{-1}{4ac - b^2}} \ln \left(x + \frac{-4ac\sqrt{\frac{-1}{4ac - b^2}} + b^2\sqrt{\frac{-1}{4ac - b^2}} + b}{2a} \right) \\
 &\quad + \sqrt{\frac{-1}{4ac - b^2}} \ln \left(x + \frac{4ac\sqrt{\frac{-1}{4ac - b^2}} - b^2\sqrt{\frac{-1}{4ac - b^2}} + b}{2a} \right) + C \\
 \int \frac{a_1 x}{ax^2 + bx + c} dx &= a_1 \left[\left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) * \right. \\
 &\quad \ln \left(x + \frac{-4ac \left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + b^2 \left(-\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + 2c}{b} \right) \\
 &\quad + \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) * \\
 &\quad \left. \ln \left(x + \frac{-4ac \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + b^2 \left(\frac{b\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} + \frac{1}{2a} \right) + 2c}{b} \right) \right] + C \\
 \int \frac{a_1 x + b_1}{ax^2 + bx + c} dx &= \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) * \\
 &\quad \ln \left(x + \frac{4ac \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) - 2a_1 c - b^2 \left(\frac{a_1}{2a} - \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) + bb_1}{2ab_1 - a_1 b} \right) \\
 &\quad + \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) * \\
 &\quad \ln \left(x + \frac{4ac \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) - 2a_1 c - b^2 \left(\frac{a_1}{2a} + \frac{(2ab_1 - a_1 b)\sqrt{b^2 - 4ac}}{2a(4ac - b^2)} \right) + bb_1}{2ab_1 - a_1 b} \right) + C
 \end{aligned}$$

$$\begin{aligned}
 \int \frac{a_1x^2 + b_1x + c_1}{ax^2 + bx + c} dx = & \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) * \\
 & \ln \left[x + \frac{4a^2c \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right)}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right. \\
 & - \frac{ab^2 \left(\frac{ab_1 - a_1b}{2a^2} - \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) + abc_1 - 2ab_1c + a_1bc}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \Big] \\
 & + \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) * \\
 & \ln \left[x + \frac{4a^2c \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right)}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \right. \\
 & - \frac{ab^2 \left(\frac{ab_1 - a_1b}{2a^2} + \frac{(2a^2c_1 - 2aa_1c - abb_1 + a_1b^2)\sqrt{b^2 - 4ac}}{2a^2(4ac - b^2)} \right) + abc_1 - 2ab_1c + a_1bc}{2a^2c_1 - 2aa_1c - abb_1 + a_1b^2} \Big] \\
 & + \frac{a_1x}{a} + C
 \end{aligned}$$

If the denominator has determinant of 0 / $b^2 = 4ac$ then

$$\int \frac{b_1x + c_1}{ax^2 + bx + c} dx = \frac{-ac_1 + b_1c}{a^3x + a^2c} + \frac{b_1 \ln(ax + c)}{a^2}$$

VI. TRIGONOMETRY INTEGRAL

- Trigonometry Level 1

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

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

or in general form

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

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 11: Implementation of integral for tangent function

```

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

$$\begin{aligned} & \int \sin mx \cos nx dx \\ & \int \cos mx \cos nx dx \\ & \int \sin mx \sin nx dx \end{aligned}$$

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

$$\begin{aligned} \int \sin^n x \, dx \\ \int \cos^n x \, dx \\ \int \tan^n x \, dx \\ \int \sec^n x \, dx \\ \int \csc^n x \, dx \\ \int \cot^n x \, dx \end{aligned}$$

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:

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

For the record, hypergeometric function ${}_2F_1\left(\frac{1}{2}, \frac{1-n}{2}; \frac{3}{2}; \cos^2(x)\right)$ 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.

$$\begin{aligned} \int \sin^n x \, dx &= -\frac{1}{n} \sin^{n-1}(x) \cos(x) + \frac{n-1}{n} \int \sin^{n-2}(x) \, dx \\ \int \cos^n x \, dx &= \frac{1}{n} \cos^{n-1}(x) \sin(x) + \frac{n-1}{n} \int \cos^{n-2}(x) \, dx \\ \int \tan^n x \, dx &= \frac{\tan^{n-1}(x)}{n-1} - \int \tan^{n-2}(x) \, dx \\ \int \sec^n x \, dx &= \frac{\sec^{n-2}(x) \tan(x)}{n-1} + \frac{n-2}{n-1} \int \sec^{n-2}(x) \, dx \\ \int \csc^n x \, dx &= \frac{-\csc^{n-2}(x) \cot(x)}{n-1} + \frac{n-2}{n-1} \int \csc^{n-2}(x) \, dx \\ \int \cot^n x \, dx &= -\frac{1}{n-1} \cot^{n-1}(x) - \int \cot^{n-2}(x) \, dx \end{aligned} \tag{2.15}$$

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

$$\text{i. } \int \sin^n x \, dx$$

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

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

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 $i = n, i \geq 2, i = i - 2$.

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 12: 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 ($1 \times 3 \times 5 \times \dots \times n - 1$), and multiply the denominator with the even factorial ($2 \times 4 \times 6 \times \dots \times n$).

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:

$$i = 8$$

$$c = 1$$

$$d = 8$$

$$\text{integral} = -\frac{c * \cos(s) * \sin^{i-1}(s)}{d}$$

$$d = d * (i - 2)$$

$$c = c * (i - 1)$$

$$i = i - 2$$

* * * * *

$$i = 6$$

$$c = 7$$

$$d = 48$$

$$\text{integral} = -\frac{c * \cos(s) * \sin^{i-1}(s)}{d}$$

$$d = d * (i - 2)$$

$$c = c * (i - 1)$$

$$i = i - 2$$

* * * * *

$$i = 4$$

$$c = 35$$

$$d = 192$$

$$\text{integral} = -\frac{c * \cos(s) * \sin^{i-1}(s)}{d}$$

$$d = d * (i - 2)$$

$$c = c * (i - 1)$$

$$i = i - 2$$

* * * * *

$$i = 2$$

$$c = 105$$

$$d = 384$$

$$\text{integral} = -\frac{c * \cos(s) * \sin^{i-1}(s)}{d}$$

$$d = d * (i - 2)$$

$$c = c * (i - 1)$$

$$i = i - 2$$

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 13: 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.

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

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:

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

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 14: 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 15: 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 $n = 8$.

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

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.

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

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 :

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

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:

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

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

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

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

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

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.

```

for i = 1 → v[0] = 1 * ki=1 = 1
for i = 1 → v[1] = 1 * li=1 = 2
for i = 2 → v[0] = v[0]i=1 * ki=2 = 1 * 3 = 3
for i = 2 → v[1] = v[0]i=1 * li=2 = 1 * 4 = 4
for i = 2 → v[2] = v[1]i=1 * li=2 = 2 * 4 = 8
for i = 3 → v[0] = v[0]i=2 * ki=3 = 3 * 5 = 15
for i = 3 → v[1] = v[0]i=2 * li=3 = 3 * 6 = 18
for i = 3 → v[2] = v[1]i=2 * li=3 = 4 * 6 = 24
for i = 3 → v[3] = v[2]i=2 * li=3 = 8 * 6 = 48

```

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 ($v[0]$) 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:

```
i = 1  
d0 = 3  
l = 4  
k = 3  
v[0] = 1  
v[1] = 2  
*****  
i = 3  
d0 = 15  
l = 6  
k = 5  
v[0] = 3  
v[1] = 4  
v[2] = 8  
*****  
i = 5  
d0 = 105  
l = 8  
k = 7  
v[0] = 15  
v[1] = 18  
v[2] = 24  
v[3] = 48  
*****  
i = 7  
d0 = 945  
l = 10  
k = 9  
v[0] = 105  
v[1] = 120  
v[2] = 144  
v[3] = 192  
v[4] = 384
```

```
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 16: 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.

$$\begin{aligned}\int \sec^3 x dx &= -\frac{\log(\sin(x) - 1)}{4} + \frac{\log(\sin(x) + 1)}{4} - \frac{\sin x}{2\sin^2 x - 2} \\ \int \sec^5 x dx &= -\frac{3\sin^3 x - 5\sin x}{8\sin^4 x - 16\sin^2 x + 8} - \frac{3\log(\sin(x) - 1)}{16} + \frac{3\log(\sin(x) + 1)}{16} \\ \int \sec^7 x dx &= \frac{-15\sin^5 x + 40\sin^3 x - 33\sin x}{48\sin^6 x - 144\sin^4 x + 144\sin^2 x - 48} - \frac{5\log(\sin(x) - 1)}{32} + \frac{5\log(\sin(x) + 1)}{32} \\ \int \sec^9 x dx &= -\frac{105\sin^7 x - 385\sin^5 x + 511\sin^3 x - 279\sin x}{384\sin^8 x - 1536\sin^6 x + 2304\sin^4 x - 1536\sin^2 x + 384} \\ &\quad - \frac{35\log(\sin(x) - 1)}{256} + \frac{35\log(\sin(x) + 1)}{256} \\ \int \sec^{11} x dx &= \frac{-315\sin^9 x + 1470\sin^7 x - 2688\sin^5 x + 2370\sin^3 x - 965\sin x}{1280\sin^{10} x - 6400\sin^8 x + 12800\sin^6 x - 12800\sin^4 x + 6400\sin^2 x - 1280} \\ &\quad - \frac{63\log(\sin(x) - 1)}{512} + \frac{63\log(\sin(x) + 1)}{512}\end{aligned}$$

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:

$$\begin{array}{ccccccc} & & & & & & 1 \\ & & & & & & \\ & & & & & & 3 \quad 5 \\ & & & & & & \\ & & & & & & 15 \quad 40 \quad 33 \\ & & & & & & \\ & & & & & & 105 \quad 385 \quad 511 \quad 279\end{array}$$

315 1470 2688 2370 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:

```

k = 1
l = 2
d0 = 2
d1 = 2
first_coeff = 3
j = 1
m = n - (n + 1 / 2)
sgn = -1

```

We are going to use vector **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: $(n - 2) * (n - 4) * \dots * 1$ and the factorial even: $(n - 1) * (n - 3) * (n - 5) * \dots * 2$. 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 $i = 1$:

```

n = 3
i = 1
v[0] = 1

```

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 $j = 1, 2, 3, \dots$ as the last index of

the vector from the previous i and $first_coeff = 3$, so

```

n = 3
i = 1
v[0] = 1
last_coeff = v[0] = 1
k = 2
***** *
n = 5
i = 2
v[0] = v[0]_{i=1} * (n - 2) = 3
v[1] = [last_coeff * (n - 2)] + k = [1 * 3] + 2 = 5
mc[0] = v[0] + v[1] = 8
last_coeff = v[1] = 5
k = 8
***** *
n = 7
i = 3
v[0] = v[0]_{i=2} * (n - 2) = 3 * 5 = 15
v[1] = mc[0] * (n - 2) = 8 * 5 = 40
v[2] = [last_coeff * (n - 2)] + k = [5 * 5] + 8 = 33
mc[0] = v[0] + v[1] = 55
mc[1] = v[1] + v[2] = 73
last_coeff = v[2] = 33
k = 48
***** *
n = 9
i = 4
v[0] = v[0]_{i=3} * (n - 2) = 15 * 7 = 105
v[1] = mc[0] * (n - 2) = 55 * 7 = 385
v[2] = mc[1] * (n - 2) = 73 * 7 = 511
v[3] = [last_coeff * (n - 2)] + k = [33 * 7] + 48 = 279
mc[0] = v[0] + v[1] = 490
mc[1] = v[1] + v[2] = 896
mc[2] = v[2] + v[3] = 790
last_coeff = v[2] = 279
k = 384

```

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 divisor out:

$$\begin{aligned} 8 \sin^4 x - 16 \sin^2 x + 8 &= 8(\sin^4 x - 2 \sin^2 x + 1) \\ 48 \sin^6 x - 144 \sin^4 x + 144 \sin^2 x - 48 &= 48(\sin^6 x - 3 \sin^4 x + 3 \sin^2 x - 1) \\ 384 \sin^8 x - \dots + \dots + 384 &= 384(\sin^8 x - 4 \sin^6 x + 6 \sin^4 x - 4 \sin^2 x + 1) \end{aligned}$$

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:

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

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

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

```
i = 1  
d1 = 4  
d0 = 8  
mc[0] = 1  
v[0] = 3  
v[1] = 5  
* * * * * * * * * *  
i = 2  
d1 = 16  
d0 = 48  
mc[0] = 8  
v[0] = 15  
v[1] = 40  
v[2] = 33  
* * * * * * * * * *  
i = 3  
d1 = 96  
d0 = 384  
mc[0] = 55  
mc[1] = 73  
v[0] = 105  
v[1] = 385  
v[2] = 511  
v[3] = 279  
* * * * * * * * * *  
i = 4  
d1 = 768  
d0 = 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
```

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 17: 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 $n = 8$.

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

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 18: 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 19: 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 $n = 8$.

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

Case 1: For $\int \cot^n x dx$ with even n

We will cluster only the even power

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

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 20: 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

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

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

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

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

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

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:

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

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:

		1		
		1	4	
	2	9	18	
6	32	72	96	
24	150	400	600	600

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

$$\begin{aligned}\mathbf{v}[0] &= 1 \\ \mathbf{v}[1] &= 4\end{aligned}$$

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

$$\begin{aligned}\mathbf{v}[0] &= 2 \\ \mathbf{v}[1] &= 9 \\ \mathbf{v}[2] &= 18\end{aligned}$$

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 $i = 1, 2, \dots, \frac{n-1}{2}$, and $j = 1, 2, \dots, \frac{n-1}{2}$. The vector \mathbf{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 \mathbf{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:

```
i = 1
d0 = 2
mc[0] = 1
v[0] = 1
* * * * *
i = 2
d0 = 4
mc[0] = 1
mc[1] = 1
v[0] = 1
v[1] = 4
* * * * *
i = 3
d0 = 12
mc[0] = 1
mc[1] = 2
mc[2] = 1
v[0] = 2
v[1] = 9
v[2] = 18
* * * * *
i = 4
d0 = 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
* * * * *
```

```

    i = 5
    d0 = 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

```

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 < 1 ; ic = ic+1)
    {
        mc_temp[ic] = mc[ic-1] + mc[ic];
    }
    mc[1] = (mc_temp[0]);
    mc[1+1] = (mc_temp[0]);

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

    for(int ic = 1 ; ic < 1 ; 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 21: 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 $n = 8$.

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

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 22: 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 23: level 3 integral for tangent odd case

- **Trigonometry Level 4**

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

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

$$\int \tan^n x \sec^m x dx$$

$$\int \cot^n x \csc^m x dx$$

the reduction formula to compute these trigonometry integral level 4 are

$$\int \sin^n x \cos^m x dx = \frac{\sin^{n+1} x \cos^{m-1} x}{m+n} + \frac{m-1}{m+n} \int \sin^n x \cos^{m-2} x dx \quad (2.16)$$

vii. $\int \sin^n x \cos^m x dx$

The differentiation to obtain the reduction formula is coming from integration by parts and a smart algebra technique

$$\begin{aligned}
I_{m,n} &= \int \sin^n x \cos^m x dx \\
&= \int \cos^{m-1} x \sin^n x \cos x dx \\
&= \cos^{m-1} x \frac{\sin^{n+1} x}{n+1} - \frac{m-1}{n+1} \int \sin^{n+2} x \cos^{m-2} x dx \\
&= \cos^{m-1} x \frac{\sin^{n+1} x}{n+1} - \frac{m-1}{n+1} \int \sin^n x \cos^{m-2} x (1 - \cos^2 x) dx \\
&= \frac{\sin^{n+1} x \cos^{m-1} x}{m+n} + \frac{m-1}{m+n} \int \sin^n x \cos^{m-2} x (1 - \cos^2 x) dx \\
&= \frac{\sin^{n+1} x \cos^{m-1} x}{m+n} + \frac{m-1}{m+n} \int \sin^n x \cos^{m-2} x dx
\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 1$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^1 x \cos^1 x dx &= \frac{\sin^2 x}{2} \\ \int \sin^1 x \cos^2 x dx &= \frac{-\cos^3 x}{3} \\ \int \sin^1 x \cos^3 x dx &= \frac{-\cos^4 x}{4} \\ \int \sin^1 x \cos^4 x dx &= \frac{-\cos^5 x}{5} \\ \int \sin^1 x \cos^5 x dx &= \frac{-\cos^6 x}{6} \\ \int \sin^1 x \cos^6 x dx &= \frac{-\cos^7 x}{7} \\ \int \sin^1 x \cos^7 x dx &= \frac{-\cos^8 x}{8} \\ \int \sin^1 x \cos^8 x dx &= \frac{-\cos^9 x}{9}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 2$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^2 x \cos^1 x dx &= \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^2 x dx &= \frac{x}{8} - \frac{\sin(2x) \cos(2x)}{16} \\ \int \sin^2 x \cos^3 x dx &= -\frac{\sin^5 x}{5} + \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^4 x dx &= \frac{x}{16} - \frac{\sin(x) \cos^5(x)}{6} + \frac{\sin(x) \cos^3(x)}{24} + \frac{\sin(x) \cos(x)}{16} \\ \int \sin^2 x \cos^5 x dx &= \frac{\sin^7 x}{7} - \frac{2 \sin^5 x}{5} + \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^6 x dx &= \frac{5x}{128} - \frac{\sin(x) \cos^7(x)}{8} + \frac{\sin(x) \cos^5(x)}{48} + \frac{5 \sin(x) \cos^3(x)}{192} + \frac{5 \sin(x) \cos(x)}{128} \\ \int \sin^2 x \cos^7 x dx &= -\frac{\sin^9 x}{9} + \frac{3 \sin^7 x}{7} - \frac{3 \sin^5 x}{5} + \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^8 x dx &= \frac{7x}{256} - \frac{\sin(x) \cos^9(x)}{10} + \frac{\sin(x) \cos^7(x)}{80} + \frac{7 \sin(x) \cos^5(x)}{480} \\ &\quad + \frac{7 \sin(x) \cos^3(x)}{384} + \frac{7 \sin(x) \cos(x)}{256}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 3$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^3 x \cos^1 x dx &= \frac{\sin^4 x}{4} \\ \int \sin^3 x \cos^2 x dx &= \frac{\cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^3 x \cos^3 x dx &= -\frac{\sin^6 x}{6} + \frac{\sin^4 x}{4} \\ \int \sin^3 x \cos^4 x dx &= \frac{\cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^3 x \cos^5 x dx &= \frac{\cos^8 x}{8} - \frac{\cos^6 x}{6} \\ \int \sin^3 x \cos^6 x dx &= \frac{\cos^9 x}{9} - \frac{\cos^7 x}{7} \\ \int \sin^3 x \cos^7 x dx &= \frac{\cos^{10} x}{10} - \frac{\cos^8 x}{8} \\ \int \sin^3 x \cos^8 x dx &= \frac{\cos^{11} x}{11} - \frac{\cos^9 x}{9}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 4$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^4 x \cos^1 x dx &= \frac{\sin^5 x}{5} \\ \int \sin^4 x \cos^2 x dx &= \frac{x}{16} + \frac{\sin^5(x) \cos(x)}{6} - \frac{\sin^3(x) \cos(x)}{24} - \frac{\sin(x) \cos(x)}{16} \\ \int \sin^4 x \cos^3 x dx &= -\frac{\sin^7 x}{7} + \frac{\sin^5 x}{5} \\ \int \sin^4 x \cos^4 x dx &= \frac{3x}{128} - \frac{\sin^3(2x) \cos(2x)}{128} - \frac{3 \sin(2x) \cos(2x)}{256} \\ \int \sin^4 x \cos^5 x dx &= \frac{\sin^9 x}{9} - \frac{2 \sin^7 x}{7} + \frac{\sin^5 x}{5} \\ \int \sin^4 x \cos^6 x dx &= \frac{3x}{256} + \frac{\sin(x) \cos^9(x)}{10} - \frac{11 \sin(x) \cos^7(x)}{80} + \frac{\sin(x) \cos^5(x)}{160} \\ &\quad + \frac{\sin(x) \cos^3(x)}{128} + \frac{3 \sin(x) \cos(x)}{256} \\ \int \sin^4 x \cos^7 x dx &= -\frac{\sin^{11} x}{11} + \frac{\sin^9 x}{3} - \frac{3 \sin^7 x}{7} + \frac{\sin^5 x}{5} \\ \int \sin^4 x \cos^8 x dx &= \frac{7x}{1024} + \frac{\sin(x) \cos^{11}(x)}{12} - \frac{13 \sin(x) \cos^9(x)}{120} + \frac{\sin(x) \cos^7(x)}{320} + \frac{7 \sin(x) \cos^5(x)}{1920} \\ &\quad + \frac{7 \sin(x) \cos^3(x)}{1536} + \frac{7 \sin(x) \cos(x)}{1024}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 5$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^5 x \cos^1 x dx &= \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^2 x dx &= -\frac{\cos^7 x}{7} + \frac{2 \cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^5 x \cos^3 x dx &= -\frac{\sin^8 x}{8} + \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^4 x dx &= -\frac{\cos^9 x}{9} + \frac{2 \cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^5 x \cos^5 x dx &= \frac{\sin^{10} x}{10} - \frac{\sin^8 x}{4} + \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^6 x dx &= -\frac{\cos^{11} x}{11} + \frac{2 \cos^9 x}{9} - \frac{\cos^7 x}{7} \\ \int \sin^5 x \cos^7 x dx &= -\frac{\cos^{12} x}{12} + \frac{\cos^{10} x}{5} - \frac{\cos^8 x}{8} \\ \int \sin^5 x \cos^8 x dx &= -\frac{\cos^{13} x}{13} + \frac{2 \cos^{11} x}{11} - \frac{\cos^9 x}{9}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 6$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^6 x \cos^1 x dx &= \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^2 x dx &= \frac{5x}{128} + \frac{\sin^7(x) \cos(x)}{8} - \frac{\sin^5(x) \cos(x)}{48} - \frac{5 \sin^3(x) \cos(x)}{192} - \frac{5 \sin(x) \cos(x)}{128} \\ \int \sin^6 x \cos^3 x dx &= -\frac{\sin^9 x}{9} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^4 x dx &= \frac{3x}{256} - \frac{\sin^9(x) \cos(x)}{10} + \frac{11 \sin^7(x) \cos(x)}{80} - \frac{\sin^5(x) \cos(x)}{160} - \frac{\sin^3(x) \cos(x)}{128} \\ &\quad - \frac{3 \sin(x) \cos(x)}{256} \\ \int \sin^6 x \cos^5 x dx &= \frac{\sin^{11} x}{11} - \frac{2 \sin^9 x}{9} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^6 x dx &= \frac{5x}{1024} - \frac{\sin^5(2x) \cos(2x)}{768} - \frac{5 \sin^3(2x) \cos(2x)}{3072} - \frac{5 \sin(2x) \cos(2x)}{2048} \\ \int \sin^6 x \cos^7 x dx &= -\frac{\sin^{13} x}{13} + \frac{3 \sin^{11} x}{11} - \frac{\sin^9 x}{3} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^8 x dx &= \frac{5x}{2048} - \frac{\sin(x) \cos^{13}(x)}{14} + \frac{29 \sin(x) \cos^{11}(x)}{168} - \frac{37 \sin(x) \cos^9(x)}{336} \\ &\quad + \frac{\sin(x) \cos^7(x)}{896} + \frac{\sin(x) \cos^5(x)}{768} + \frac{5 \sin(x) \cos^3(x)}{3072} + \frac{5 \sin(x) \cos(x)}{2048}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 7$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^7 x \cos^1 x dx &= \frac{\sin^8 x}{8} \\ \int \sin^7 x \cos^2 x dx &= \frac{\cos^9 x}{9} - \frac{3\cos^7 x}{7} + \frac{3\cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^7 x \cos^3 x dx &= -\frac{\sin^{10} x}{10} + \frac{\sin^8 x}{8} \\ \int \sin^7 x \cos^4 x dx &= \frac{\cos^{11} x}{11} - \frac{\cos^9 x}{3} + \frac{3\cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^7 x \cos^5 x dx &= \frac{\sin^{12} x}{12} - \frac{\sin^{10} x}{5} + \frac{\sin^8 x}{8} \\ \int \sin^7 x \cos^6 x dx &= \frac{\cos^{13} x}{13} - \frac{3\cos^{11} x}{11} + \frac{\cos^9 x}{3} - \frac{\cos^7 x}{7} \\ \int \sin^7 x \cos^7 x dx &= -\frac{\sin^{14} x}{14} + \frac{\sin^{12} x}{4} - \frac{3\sin^{10} x}{10} + \frac{\sin^8 x}{8} \\ \int \sin^7 x \cos^8 x dx &= \frac{\cos^{15} x}{15} - \frac{3\cos^{13} x}{13} + \frac{3\cos^{11} x}{11} - \frac{\cos^9 x}{9}\end{aligned}$$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 8$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \sin^8 x \cos^1 x dx &= \frac{\sin^9 x}{9} \\ \int \sin^8 x \cos^2 x dx &= \frac{7x}{256} + \frac{\sin^9(x) \cos(x)}{10} - \frac{\sin^7(x) \cos(x)}{80} - \frac{7\sin^5(x) \cos(x)}{480} \\ &\quad - \frac{7\sin^3(x) \cos(x)}{384} - \frac{7\sin(x) \cos(x)}{256} \\ \int \sin^8 x \cos^3 x dx &= -\frac{\sin^{11} x}{11} + \frac{\sin^9 x}{9} \\ \int \sin^8 x \cos^4 x dx &= \frac{7x}{1024} - \frac{\sin^{11}(x) \cos(x)}{12} + \frac{13\sin^9(x) \cos(x)}{120} - \frac{\sin^7(x) \cos(x)}{320} - \frac{7\sin^5(x) \cos(x)}{1920} \\ &\quad - \frac{7\sin^3(x) \cos(x)}{1536} - \frac{7\sin(x) \cos(x)}{1024} \\ \int \sin^8 x \cos^5 x dx &= \frac{\sin^{13} x}{13} - \frac{2\sin^{11} x}{11} + \frac{\sin^9 x}{9} \\ \int \sin^8 x \cos^6 x dx &= \frac{5x}{2048} + \frac{\sin^{13}(x) \cos(x)}{14} - \frac{29\sin^{11}(x) \cos(x)}{168} + \frac{37\sin^9(x) \cos(x)}{336} \\ &\quad - \frac{\sin^7(x) \cos(x)}{896} - \frac{\sin^5(x) \cos(x)}{768} - \frac{5\sin^3(x) \cos(x)}{3072} - \frac{5\sin(x) \cos(x)}{2048} \\ \int \sin^8 x \cos^7 x dx &= -\frac{\sin^{15} x}{15} + \frac{3\sin^{13} x}{13} - \frac{3\sin^{11} x}{11} + \frac{\sin^9 x}{9} \\ \int \sin^8 x \cos^8 x dx &= \frac{35x}{32768} - \frac{\sin^7(2x) \cos(2x)}{4096} - \frac{7\sin^5(2x) \cos(2x)}{24576} \\ &\quad - \frac{35\sin^3(2x) \cos(2x)}{98304} - \frac{35\sin(2x) \cos(2x)}{65536}\end{aligned}$$

Case 1: For $\int \sin^n x \cos^m x dx$ with even power for both n and m

This case occurs when both m and n are even number. We divide this case into 3 smaller

cases:

1. When $n > m$
2. When $n < m$
3. When $n = m$

Case 1-a: For $\int \sin^n x \cos^m x dx$ with $n > m$

We will start with a big example, these are the formula of $\int \sin^n x \cos^m x dx$ with $n = 14$ and $m = 2$ to $m = 12$ with increment of 2.

$$\begin{aligned} \int \sin^{14} x \cos^2 x dx &= \frac{429x}{32768} + \frac{\sin^{15}(x) \cos(x)}{16} - \frac{\sin^{13}(x) \cos(x)}{224} - \frac{13 \sin^{11}(x) \cos(x)}{2688} \\ &\quad - \frac{143 \sin^9(x) \cos(x)}{26880} - \frac{429 \sin^7(x) \cos(x)}{71680} - \frac{143 \sin^5(x) \cos(x)}{20480} \\ &\quad - \frac{143 \sin^3(x) \cos(x)}{16384} - \frac{429 \sin(x) \cos(x)}{32768} \end{aligned}$$

$$\begin{aligned} \int \sin^{14} x \cos^4 x dx &= \frac{143x}{65536} - \frac{\sin^{17}(x) \cos(x)}{18} + \frac{19 \sin^{15}(x) \cos(x)}{288} - \frac{\sin^{13}(x) \cos(x)}{1344} \\ &\quad - \frac{13 \sin^{11}(x) \cos(x)}{16128} - \frac{143 \sin^9(x) \cos(x)}{161280} - \frac{143 \sin^7(x) \cos(x)}{143360} \\ &\quad - \frac{143 \sin^5(x) \cos(x)}{122880} - \frac{143 \sin^3(x) \cos(x)}{98304} - \frac{143 \sin(x) \cos(x)}{65536} \end{aligned}$$

$$\begin{aligned} \int \sin^{14} x \cos^6 x dx &= \frac{143x}{262144} + \frac{\sin^{19}(x) \cos(x)}{20} - \frac{41 \sin^{17}(x) \cos(x)}{360} + \frac{383 \sin^{15}(x) \cos(x)}{5760} \\ &\quad - \frac{\sin^{13}(x) \cos(x)}{5376} - \frac{13 \sin^{11}(x) \cos(x)}{64512} - \frac{143 \sin^9(x) \cos(x)}{645120} \\ &\quad - \frac{143 \sin^7(x) \cos(x)}{573440} - \frac{143 \sin^5(x) \cos(x)}{491520} - \frac{143 \sin^3(x) \cos(x)}{393216} \\ &\quad - \frac{143 \sin(x) \cos(x)}{262144} \end{aligned}$$

$$\begin{aligned} \int \sin^{14} x \cos^8 x dx &= \frac{91x}{524288} - \frac{\sin^{21}(x) \cos(x)}{22} + \frac{67 \sin^{19}(x) \cos(x)}{440} - \frac{1367 \sin^{17}(x) \cos(x)}{7920} \\ &\quad + \frac{8441 \sin^{15}(x) \cos(x)}{126720} - \frac{\sin^{13}(x) \cos(x)}{16896} - \frac{13 \sin^{11}(x) \cos(x)}{202752} \\ &\quad - \frac{13 \sin^9(x) \cos(x)}{184320} - \frac{13 \sin^7(x) \cos(x)}{163840} - \frac{91 \sin^5(x) \cos(x)}{983040} \\ &\quad - \frac{91 \sin^3(x) \cos(x)}{786432} - \frac{91 \sin(x) \cos(x)}{524288} \end{aligned}$$

$$\begin{aligned} \int \sin^{14} x \cos^{10} x dx &= \frac{273x}{4194304} + \frac{\sin^{23}(x) \cos(x)}{24} - \frac{97 \sin^{21}(x) \cos(x)}{528} + \frac{1081 \sin^{19}(x) \cos(x)}{3520} \\ &\quad - \frac{1629 \sin^{17}(x) \cos(x)}{7040} + \frac{7507 \sin^{15}(x) \cos(x)}{112640} - \frac{\sin^{13}(x) \cos(x)}{45056} \\ &\quad - \frac{13 \sin^{11}(x) \cos(x)}{540672} - \frac{13 \sin^9(x) \cos(x)}{491520} - \frac{39 \sin^7(x) \cos(x)}{1310720} \\ &\quad - \frac{91 \sin^5(x) \cos(x)}{2621440} - \frac{91 \sin^3(x) \cos(x)}{2097152} - \frac{273 \sin(x) \cos(x)}{4194304} \end{aligned}$$

$$\begin{aligned} \int \sin^{14} x \cos^{12} x dx = & \frac{231x}{8388608} - \frac{\sin^{25}(x) \cos(x)}{26} + \frac{131 \sin^{23}(x) \cos(x)}{624} - \frac{577 \sin^{21}(x) \cos(x)}{1248} \\ & + \frac{4281 \sin^{19}(x) \cos(x)}{8320} - \frac{4829 \sin^{17}(x) \cos(x)}{16640} + \frac{17747 \sin^{15}(x) \cos(x)}{266240} \\ & - \frac{\sin^{13}(x) \cos(x)}{106496} - \frac{\sin^{11}(x) \cos(x)}{98304} - \frac{11 \sin^9(x) \cos(x)}{983040} \\ & - \frac{33 \sin^7(x) \cos(x)}{2621440} - \frac{77 \sin^5(x) \cos(x)}{5242880} - \frac{77 \sin^3(x) \cos(x)}{4194304} \\ & - \frac{231 \sin(x) \cos(x)}{8388608} \end{aligned}$$

The patterns that can be seen from those equations above are

- The integral $\int \sin^{14} x dx$ represents some term for $\int \sin^{14} x \cos^2 x dx$ with denominator that is multiplied by $m + n = 14 + 2 = 16$.
- The higher the power of m the longer the new term that is having power of \sin^{n+1} and above.
- As the iteration increases the denominator coefficient is the multiplication of $m + n$ with m starting from 2 to 12 with increment of 2.

This kind of integral is quite challenging, if we want to compute this integral with $n = 14$ and $m = 12$ we will start from $m = 0$, then $m = 2$. So let see when $m = 0$, we have computed and know the pattern for this kind of integral:

$$\begin{aligned} \int \sin^{14} x \cos^0 x dx = & \frac{429x}{2048} - \frac{\sin^{13}(x) \cos(x)}{14} - \frac{13 \sin^{11}(x) \cos(x)}{168} \\ & - \frac{143 \sin^9(x) \cos(x)}{1680} - \frac{429 \sin^7(x) \cos(x)}{4480} - \frac{143 \sin^5(x) \cos(x)}{1280} \\ & - \frac{143 \sin^3(x) \cos(x)}{1024} - \frac{429 \sin(x) \cos(x)}{2048} \end{aligned}$$

now when $m = 2$, we will obtain this:

$$\begin{aligned} \int \sin^{14} x \cos^2 x dx = & \frac{429x}{32768} + \frac{\sin^{15}(x) \cos(x)}{16} - \frac{\sin^{13}(x) \cos(x)}{224} - \frac{13 \sin^{11}(x) \cos(x)}{2688} \\ & - \frac{143 \sin^9(x) \cos(x)}{26880} - \frac{429 \sin^7(x) \cos(x)}{71680} - \frac{143 \sin^5(x) \cos(x)}{20480} \\ & - \frac{143 \sin^3(x) \cos(x)}{16384} - \frac{429 \sin(x) \cos(x)}{32768} \end{aligned}$$

From the reduction formula we know that this term:

$$\frac{c_{n/2}x}{d_{n/2}} + \sum_{i=1}^{n/2} -\frac{c_i \sin^{2i-1} x \cos x}{d_i}$$

will be multiplied by $\frac{m-1}{m+n}$ at every iteration from $m = 2$ till $m = 12$. So the problem is solved for this term.

For example, to obtain the denominator for function of x of 32768 from multiplying the denominator of function x from $\int \sin^{14} x dx$ with $d_0 = m + n = 16$, it is $2048 * 16 = 32768$.

The remaining term will have different pattern for the numerator, from now on we will omit the term of $\frac{c_{n/2}x}{d_{n/2}} + \sum_{i=1}^{n/2} -\frac{c_i \sin^{2i-1} x \cos x}{d_i}$, now take a look again:

$$\begin{aligned}\int \sin^{14} x \cos^2 x dx &= \frac{\sin^{15}(x) \cos(x)}{16} + \dots - \dots \\ \int \sin^{14} x \cos^4 x dx &= -\frac{\sin^{17}(x) \cos(x)}{18} + \frac{19 \sin^{15}(x) \cos(x)}{288} + \dots - \dots \\ \int \sin^{14} x \cos^6 x dx &= \frac{\sin^{19}(x) \cos(x)}{20} - \frac{41 \sin^{17}(x) \cos(x)}{360} \\ &\quad + \frac{383 \sin^{15}(x) \cos(x)}{5760} + \dots - \dots \\ \int \sin^{14} x \cos^8 x dx &= -\frac{\sin^{21}(x) \cos(x)}{22} + \frac{67 \sin^{19}(x) \cos(x)}{440} - \frac{1367 \sin^{17}(x) \cos(x)}{7920} \\ &\quad + \frac{8441 \sin^{15}(x) \cos(x)}{126720} + \dots - \dots \\ \int \sin^{14} x \cos^{10} x dx &= \frac{\sin^{23}(x) \cos(x)}{24} - \frac{97 \sin^{21}(x) \cos(x)}{528} + \frac{1081 \sin^{19}(x) \cos(x)}{3520} \\ &\quad - \frac{1629 \sin^{17}(x) \cos(x)}{7040} + \frac{7507 \sin^{15}(x) \cos(x)}{112640} + \dots - \dots \\ \int \sin^{14} x \cos^{12} x dx &= -\frac{\sin^{25}(x) \cos(x)}{26} + \frac{131 \sin^{23}(x) \cos(x)}{624} - \frac{577 \sin^{21}(x) \cos(x)}{1248} \\ &\quad + \frac{4281 \sin^{19}(x) \cos(x)}{8320} - \frac{4829 \sin^{17}(x) \cos(x)}{16640} \\ &\quad + \frac{17747 \sin^{15}(x) \cos(x)}{266240} + \dots - \dots\end{aligned}$$

We will learn how to decode this pattern, first we will make the denominator into its' raw number, so the numerator will change into its' raw number too, here it goes

$$\begin{aligned}\int \sin^{14} x \cos^2 x dx &= \frac{\sin^{15}(x) \cos(x)}{16} + \dots - \dots \\ \int \sin^{14} x \cos^4 x dx &= -\frac{\sin^{17}(x) \cos(x)}{18} + \frac{19 \sin^{15}(x) \cos(x)}{288} + \dots - \dots \\ \int \sin^{14} x \cos^6 x dx &= \frac{\sin^{19}(x) \cos(x)}{20} - \frac{41 \sin^{17}(x) \cos(x)}{360} \\ &\quad + \frac{383 \sin^{15}(x) \cos(x)}{5760} + \dots - \dots \\ \int \sin^{14} x \cos^8 x dx &= -\frac{\sin^{21}(x) \cos(x)}{22} + \frac{67 \sin^{19}(x) \cos(x)}{440} - \frac{1367 \sin^{17}(x) \cos(x)}{7920} \\ &\quad + \frac{8441 \sin^{15}(x) \cos(x)}{126720} + \dots - \dots \\ \int \sin^{14} x \cos^{10} x dx &= \frac{\sin^{23}(x) \cos(x)}{24} - \frac{97 \sin^{21}(x) \cos(x)}{528} + \frac{3243 \sin^{19}(x) \cos(x)}{10560} \\ &\quad - \frac{43983 \sin^{17}(x) \cos(x)}{190080} + \frac{202689 \sin^{15}(x) \cos(x)}{3041280} + \dots - \dots\end{aligned}$$

$$\begin{aligned}\int \sin^{14} x \cos^{12} x dx = & -\frac{\sin^{25}(x) \cos(x)}{26} + \frac{131 \sin^{23}(x) \cos(x)}{624} - \frac{6347 \sin^{21}(x) \cos(x)}{13728} \\ & + \frac{141273 \sin^{19}(x) \cos(x)}{274560} - \frac{1434213 \sin^{17}(x) \cos(x)}{4942080} \\ & + \frac{5270859 \sin^{15}(x) \cos(x)}{79073280} + \dots - \dots\end{aligned}$$

Then for a better look we will write it into Pascal' triangle form, for the numerator in the raw version

$$\begin{array}{ccccccccc} & & & & & 1 & & & \\ & & & & 1 & & 19 & & \\ & & & 1 & & 41 & & 383 & \\ & & 1 & & 67 & & 1367 & & 8441 \\ & 1 & & 97 & & 3243 & & 43983 & & 202689 \\ 1 & & 131 & & 6347 & & 141273 & & 1434213 & & 5270859 \end{array}$$

The C++ code will use an increasing **for** loop with $i = 1, i \leq \frac{m}{2}, i = i + 1$.

For the computation, we are creating a variable for the denominator for function $\sin^{m+n-1} x \cos x$ as d_0 , with $d_0 = m + n$. This variable will plays an important role to determine the value for the numerator of this kind of integral.

We will also use vector **v** to store the coefficient with

$$\begin{aligned}\mathbf{v}[0] &= 1 \\ \mathbf{v}[1]_i &= (d_0 * (i - 1)) + 1, \quad i = 2, 3, \dots, \frac{m}{2} - 1 \\ \mathbf{v}\left[\frac{m}{2} - 1\right]_i &= (d_0 * \mathbf{v}\left[\frac{m}{2} - 2\right]_{i-1}) + \mathbf{factorialoddup}(i - 1)\end{aligned}$$

$\mathbf{v}\left[\frac{m}{2} - 1\right]_i$ is the last entry for the vector at iteration i , while $\mathbf{v}\left[\frac{m}{2} - 2\right]_{i-1}$ is the last entry for the vector at the previous iteration $i - 1$.

We make a definition here of **factorialoddup**($i - 1$):

$$\mathbf{factorialoddup}(i - 1) = 1 * 3 * 5 * \dots$$

it is a multiplication of odd number from 1 upward till we get $i - 1$ terms.

$$\begin{aligned}\mathbf{factorialoddup}(2) &= 1 * 3 \\ \mathbf{factorialoddup}(3) &= 1 * 3 * 5 \\ \mathbf{factorialoddup}(4) &= 1 * 3 * 5 * 7\end{aligned}$$

For $i = 4$ we will make a **for** loop again to find the entry of the vector $\mathbf{v}[j]_i$ for $j = 2, 3, \dots, \frac{m}{2} - 2$.

$$\begin{aligned}\mathbf{v}[j]_i &= (d_0 * \mathbf{v}[j - 1]_{i-1}) + (k * \mathbf{v}[j - 1]_{i-1} + \mathbf{v}[j]_{i-1}) \\ k &= k - 2\end{aligned}$$

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

```

 $i = 1$ 
 $d_0 = 16$ 
 $v[0] = 1$ 
*****  

 $i = 2$ 
 $d_0 = 18$ 
 $v[0] = 1$ 
 $v[1] = (d_0 * (2 - 1)) + 1 = 19$ 
*****  

 $i = 3$ 
 $d_0 = 20$ 
 $v[0] = 1$ 
 $v[1] = (d_0 * (3 - 1)) + 1 = 41$ 
 $v[2] = (d_0 * v[1]_{i=2}) + (1 * 3) = 383$ 
*****  

 $i = 4$ 
 $d_0 = 22$ 
 $k = 2$ 
 $v[0] = 1$ 
 $v[1] = (d_0 * (4 - 1)) + 1 = 67$ 
 $v[2] = (d_0 * v[1]_{i=3}) + (k * v[1]_{i=3} + v[2]_{i=3}) = 1367$ 
 $v[3] = (d_0 * v[2]_{i=3}) + (1 * 3 * 5) = 8441$ 

```

Case 1-b: For $\int \sin^n x \cos^m x dx$ **with** $n < m$

Let's take a look at several integral related to this case

$$\begin{aligned}
 \int \sin^2 x \cos^4 x \, dx &= \frac{x}{16} - \frac{\sin(x) \cos^5(x)}{6} + \frac{\sin(x) \cos^3(x)}{24} + \frac{\sin(x) \cos(x)}{16} \\
 \int \sin^2 x \cos^6 x \, dx &= \frac{5x}{128} - \frac{\sin(x) \cos^7(x)}{8} + \frac{\sin(x) \cos^5(x)}{48} + \frac{5 \sin(x) \cos^3(x)}{192} + \frac{5 \sin(x) \cos(x)}{128} \\
 \int \sin^4 x \cos^6 x \, dx &= \frac{3x}{256} + \frac{\sin(x) \cos^9(x)}{10} - \frac{11 \sin(x) \cos^7(x)}{80} + \frac{\sin(x) \cos^5(x)}{160} \\
 &\quad + \frac{\sin(x) \cos^3(x)}{128} + \frac{3 \sin(x) \cos(x)}{256} \\
 \int \sin^2 x \cos^8 x \, dx &= \frac{7x}{256} - \frac{\sin(x) \cos^9(x)}{10} + \frac{\sin(x) \cos^7(x)}{80} + \frac{7 \sin(x) \cos^5(x)}{480} \\
 &\quad + \frac{7 \sin(x) \cos^3(x)}{384} + \frac{7 \sin(x) \cos(x)}{256} \\
 \int \sin^4 x \cos^8 x \, dx &= \frac{7x}{1024} + \frac{\sin(x) \cos^{11}(x)}{12} - \frac{13 \sin(x) \cos^9(x)}{120} + \frac{\sin(x) \cos^7(x)}{320} + \frac{7 \sin(x) \cos^5(x)}{1920} \\
 &\quad + \frac{7 \sin(x) \cos^3(x)}{1536} + \frac{7 \sin(x) \cos(x)}{1024} \\
 \int \sin^6 x \cos^8 x \, dx &= \frac{5x}{2048} - \frac{\sin(x) \cos^{13}(x)}{14} + \frac{29 \sin(x) \cos^{11}(x)}{168} - \frac{37 \sin(x) \cos^9(x)}{336} \\
 &\quad + \frac{\sin(x) \cos^7(x)}{896} + \frac{\sin(x) \cos^5(x)}{768} + \frac{5 \sin(x) \cos^3(x)}{3072} + \frac{5 \sin(x) \cos(x)}{2048}
 \end{aligned}$$

This case does not need to be scrutinize further, since the numerator and denominator for all terms are exactly the same as those for the case 1-a where $n > m$, we only need to change the sine function into cosine and cosine function into sine in the result, since this case has the cosine function in the term that has the power. The plus and minus will also be interchange for the term with sine and cosine function, the term with x stays positive always.

Case 1-c: For $\int \sin^n x \cos^m x \, dx$ with $n = m$

These are the formula of $\int \sin^n x \cos^m x dx$ with $n = 1$ to $n = 8$.

$$\begin{aligned}\int \sin^1 x \cos^1 x dx &= \frac{\sin^2 x}{2} \\ \int \sin^2 x \cos^2 x dx &= \frac{x}{8} - \frac{\sin(2x) \cos(2x)}{16} \\ \int \sin^3 x \cos^3 x dx &= -\frac{\sin^6 x}{6} + \frac{\sin^4 x}{4} \\ \int \sin^4 x \cos^4 x dx &= \frac{3x}{128} - \frac{\sin^3(2x) \cos(2x)}{128} - \frac{3 \sin(2x) \cos(2x)}{256} \\ \int \sin^5 x \cos^5 x dx &= \frac{\sin^{10} x}{10} - \frac{\sin^8 x}{4} + \frac{\sin^6 x}{6} \\ \int \sin^6 x \cos^6 x dx &= \frac{5x}{1024} - \frac{\sin^5(2x) \cos(2x)}{768} - \frac{5 \sin^3(2x) \cos(2x)}{3072} - \frac{5 \sin(2x) \cos(2x)}{2048} \\ \int \sin^7 x \cos^7 x dx &= -\frac{\sin^{14} x}{14} + \frac{\sin^{12} x}{4} - \frac{3 \sin^{10} x}{10} + \frac{\sin^8 x}{8} \\ \int \sin^8 x \cos^8 x dx &= \frac{35x}{32768} - \frac{\sin^7(2x) \cos(2x)}{4096} - \frac{7 \sin^5(2x) \cos(2x)}{24576} - \frac{35 \sin^3(2x) \cos(2x)}{98304} \\ &\quad - \frac{35 \sin(2x) \cos(2x)}{65536}\end{aligned}$$

We can tell that the case here will be divided into two, which are even case and odd case.

If n is even

$$\begin{aligned}\int \sin^2 x \cos^2 x dx &= \frac{x}{8} - \frac{\sin(2x) \cos(2x)}{16} \\ \int \sin^4 x \cos^4 x dx &= \frac{3x}{128} - \frac{\sin^3(2x) \cos(2x)}{128} - \frac{3 \sin(2x) \cos(2x)}{256} \\ \int \sin^6 x \cos^6 x dx &= \frac{5x}{1024} - \frac{\sin^5(2x) \cos(2x)}{768} - \frac{5 \sin^3(2x) \cos(2x)}{3072} - \frac{5 \sin(2x) \cos(2x)}{2048} \\ \int \sin^8 x \cos^8 x dx &= \frac{35x}{32768} - \frac{\sin^7(2x) \cos(2x)}{4096} - \frac{7 \sin^5(2x) \cos(2x)}{24576} - \frac{35 \sin^3(2x) \cos(2x)}{98304} \\ &\quad - \frac{35 \sin(2x) \cos(2x)}{65536}\end{aligned}$$

The patterns that can be seen from those equations above are

- The denominator with function of $\sin(2x)^{n-1} \cos(2x)$ can be determined with:

$$d_0 = 2^n * (2n)$$

This denominator will be used for the first integral result which is

$$\frac{\sin^{n-1}(2x) \cos(2x)}{d_0}$$

the next iteration will have the pattern like this:

$$\frac{(n-1) \sin^{n-3}(2x) \cos(2x)}{d_0 * (n-2)}$$

The next iteration will have pattern like this:

$$\frac{(n-1)(n-3) \sin^{n-5}(2x) \cos(2x)}{d_0 * (n-2) * (n-4)}$$

Till we obtain the last iteration integral result with function $\sin(2x) \cos(2x)$ at the numerator we will stop. The easy thing here is the signs are all minus.

- The function x will have numerator and denominator of:

$$\frac{2 * (n-1) * (n-3) * \dots * 1}{d_0 * (n-2) * (n-4) * \dots * 1} * x$$

If n is odd

$$\begin{aligned}\int \sin^1 x \cos^1 x dx &= \frac{\sin^2 x}{2} \\ \int \sin^3 x \cos^3 x dx &= -\frac{\sin^6 x}{6} + \frac{\sin^4 x}{4} \\ \int \sin^5 x \cos^5 x dx &= \frac{\sin^{10} x}{10} - \frac{\sin^8 x}{4} + \frac{\sin^6 x}{6} \\ \int \sin^7 x \cos^7 x dx &= -\frac{\sin^{14} x}{14} + \frac{\sin^{12} x}{4} - \frac{3 \sin^{10} x}{10} + \frac{\sin^8 x}{8}\end{aligned}$$

The patterns that can be seen from those equations above are

- The integral will have $\frac{n+1}{2}$ term/s, and have function of $\sin^{n+1} x$ at the numerator, at the next iteration we will have the function of $\sin^{(n+1)+2} x$ at the numerator with alternating sign, till we reach the $\sin^{2n} x$ at the numerator.
- The power determine the Pascal' triangle level / combinations formula that will be used to determine the numerator coefficients.
For $n = 1$ we will use the first level of Pascal' triangle which is ${}_0C_0$.
For $n = 3$ we will use the second level of Pascal' triangle which is ${}_1C_0$ and ${}_1C_1$.
For $n = 5$ we will use the third level of Pascal' triangle which is ${}_2C_0, {}_2C_1$ and ${}_2C_2$.
For $n = 7$ we will use the fourth level of Pascal' triangle which is ${}_3C_0, {}_3C_1, {}_3C_2$ and ${}_3C_3$.
- The denominator coefficient can be easily determined, it is the same as the power of the function in the numerator

Case 2: For $\int \sin^n x \cos^m x dx$ with at least one odd power in either n or m

This case occurs if either n or m is an odd number. We will recall the integral for $n = 1$,

$n = 2, n = 3, n = 5$, and $n = 6$

$$\begin{aligned}\int \sin^1 x \cos^2 x dx &= \frac{-\cos^3 x}{3} \\ \int \sin^1 x \cos^3 x dx &= \frac{-\cos^4 x}{4} \\ \int \sin^1 x \cos^4 x dx &= \frac{-\cos^5 x}{5} \\ \int \sin^1 x \cos^5 x dx &= \frac{-\cos^6 x}{6} \\ \int \sin^1 x \cos^6 x dx &= \frac{-\cos^7 x}{7} \\ \int \sin^1 x \cos^7 x dx &= \frac{-\cos^8 x}{8} \\ \int \sin^1 x \cos^8 x dx &= \frac{-\cos^9 x}{9}\end{aligned}$$

$$\begin{aligned}\int \sin^2 x \cos^3 x dx &= -\frac{\sin^5 x}{5} + \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^5 x dx &= \frac{\sin^7 x}{7} - \frac{2\sin^5 x}{5} + \frac{\sin^3 x}{3} \\ \int \sin^2 x \cos^7 x dx &= -\frac{\sin^9 x}{9} + \frac{3\sin^7 x}{7} - \frac{3\sin^5 x}{5} + \frac{\sin^3 x}{3}\end{aligned}$$

$$\begin{aligned}\int \sin^3 x \cos^1 x dx &= \frac{\sin^4 x}{4} \\ \int \sin^3 x \cos^2 x dx &= \frac{\cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^3 x \cos^4 x dx &= \frac{\cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^3 x \cos^5 x dx &= \frac{\cos^8 x}{8} - \frac{\cos^6 x}{6} \\ \int \sin^3 x \cos^6 x dx &= \frac{\cos^9 x}{9} - \frac{\cos^7 x}{7} \\ \int \sin^3 x \cos^7 x dx &= \frac{\cos^{10} x}{10} - \frac{\cos^8 x}{8} \\ \int \sin^3 x \cos^8 x dx &= \frac{\cos^{11} x}{11} - \frac{\cos^9 x}{9}\end{aligned}$$

$$\begin{aligned}\int \sin^5 x \cos^1 x dx &= \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^2 x dx &= -\frac{\cos^7 x}{7} + \frac{2 \cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^5 x \cos^3 x dx &= -\frac{\sin^8 x}{8} + \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^4 x dx &= -\frac{\cos^9 x}{9} + \frac{2 \cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^5 x \cos^6 x dx &= -\frac{\cos^{11} x}{11} + \frac{2 \cos^9 x}{9} - \frac{\cos^7 x}{7} \\ \int \sin^5 x \cos^7 x dx &= -\frac{\cos^{12} x}{12} + \frac{\cos^{10} x}{5} - \frac{\cos^8 x}{8} \\ \int \sin^5 x \cos^8 x dx &= -\frac{\cos^{13} x}{13} + \frac{2 \cos^{11} x}{11} - \frac{\cos^9 x}{9}\end{aligned}$$

$$\begin{aligned}\int \sin^6 x \cos^1 x dx &= \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^3 x dx &= -\frac{\sin^9 x}{9} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^5 x dx &= \frac{\sin^{11} x}{11} - \frac{2 \sin^9 x}{9} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^7 x dx &= -\frac{\sin^{13} x}{13} + \frac{3 \sin^{11} x}{11} - \frac{\sin^9 x}{3} + \frac{\sin^7 x}{7}\end{aligned}$$

There will be two big cases here which are:

1. When $m > n$
2. When $m \leq n$

If $m > n$

$$\begin{aligned}
 \int \sin^1 x \cos^2 x dx &= \frac{-\cos^3 x}{3} \\
 \int \sin^1 x \cos^3 x dx &= \frac{-\cos^4 x}{4} \\
 \int \sin^1 x \cos^4 x dx &= \frac{-\cos^5 x}{5} \\
 \int \sin^1 x \cos^5 x dx &= \frac{-\cos^6 x}{6} \\
 \int \sin^1 x \cos^6 x dx &= \frac{-\cos^7 x}{7} \\
 \int \sin^1 x \cos^7 x dx &= \frac{-\cos^8 x}{8} \\
 \int \sin^1 x \cos^8 x dx &= \frac{-\cos^9 x}{9} \\
 \int \sin^2 x \cos^3 x dx &= -\frac{\sin^5 x}{5} + \frac{\sin^3 x}{3} \\
 \int \sin^2 x \cos^5 x dx &= \frac{\sin^7 x}{7} - \frac{2\sin^5 x}{5} + \frac{\sin^3 x}{3} \\
 \int \sin^2 x \cos^7 x dx &= -\frac{\sin^9 x}{9} + \frac{3\sin^7 x}{7} - \frac{3\sin^5 x}{5} + \frac{\sin^3 x}{3} \\
 \int \sin^3 x \cos^4 x dx &= \frac{\cos^7 x}{7} - \frac{\cos^5 x}{5} \\
 \int \sin^3 x \cos^5 x dx &= \frac{\cos^8 x}{8} - \frac{\cos^6 x}{6} \\
 \int \sin^5 x \cos^6 x dx &= -\frac{\cos^{11} x}{11} + \frac{2\cos^9 x}{9} - \frac{\cos^7 x}{7} \\
 \int \sin^5 x \cos^7 x dx &= -\frac{\cos^{12} x}{12} + \frac{\cos^{10} x}{5} - \frac{\cos^8 x}{8} \\
 \int \sin^5 x \cos^8 x dx &= -\frac{\cos^{13} x}{13} + \frac{2\cos^{11} x}{11} - \frac{\cos^9 x}{9} \\
 \int \sin^6 x \cos^7 x dx &= -\frac{\sin^{13} x}{13} + \frac{3\sin^{11} x}{11} - \frac{\sin^9 x}{3} + \frac{\sin^7 x}{7}
 \end{aligned}$$

The patterns that can be seen from those equations above are

- The integral will have the result with function of cosine, if n is an odd number.

The integral will have the result of sine, if n is an even number.

- If n is an even number, then the pattern will be like this for the integral:

$$\begin{aligned}
 \int \sin^n x \cos^m x dx &= \text{sgn}_0 * \frac{m-1}{2} C_0 \frac{\sin^{n+1} x}{n+1} + \text{sgn}_1 * \frac{m-1}{2} C_1 \frac{\sin^{n+3} x}{n+3} + \dots \\
 &\quad + \text{sgn}_{\frac{m-1}{2}} * \frac{m-1}{2} C_{\frac{m-1}{2}} \frac{\sin^{n+m} x}{n+m}
 \end{aligned}$$

We start with $\text{sgn}_0 = 1$ and it is alternating at every iteration.

- If n is an odd number, then the pattern will be like this for the integral:

$$\int \sin^n x \cos^m x dx = \text{sgn}_0 * \frac{\cos^{m+1} x}{m+1} + \text{sgn}_1 * \frac{\cos^{m+3} x}{m+3} + \dots \\ \text{sgn}_{\frac{n-1}{2}} * + \frac{\cos^{n+m} x}{n+m}$$

We start with $\text{sgn}_0 = -1$ and it is alternating at every iteration.

If $m \leq n$

$$\begin{aligned} \int \sin^3 x \cos^1 x dx &= \frac{\sin^4 x}{4} \\ \int \sin^3 x \cos^2 x dx &= \frac{\cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^5 x \cos^1 x dx &= \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^2 x dx &= -\frac{\cos^7 x}{7} + \frac{2\cos^5 x}{5} - \frac{\cos^3 x}{3} \\ \int \sin^5 x \cos^3 x dx &= -\frac{\sin^8 x}{8} + \frac{\sin^6 x}{6} \\ \int \sin^5 x \cos^4 x dx &= -\frac{\cos^9 x}{9} + \frac{2\cos^7 x}{7} - \frac{\cos^5 x}{5} \\ \int \sin^6 x \cos^1 x dx &= \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^3 x dx &= -\frac{\sin^9 x}{9} + \frac{\sin^7 x}{7} \\ \int \sin^6 x \cos^5 x dx &= \frac{\sin^{11} x}{11} - \frac{2\sin^9 x}{9} + \frac{\sin^7 x}{7} \end{aligned}$$

The patterns that can be seen from those equations above are

- If m is an even number, then the integral will be the function of cosine.

If m is an odd number, then the integral will be the function of sine.

- If m is an even number, then the pattern will be like this for the integral:

$$\int \sin^n x \cos^m x dx = \text{sgn}_0 * \frac{\cos^{m+1} x}{m+1} + \text{sgn}_1 * \frac{\cos^{m+3} x}{m+3} + \dots \\ + \text{sgn}_{\frac{m-1}{2}} * \frac{\cos^{n+m} x}{n+m}$$

We start with $\text{sgn}_0 = -1$ and it is alternating at every iteration.

- If m is an odd number, then the pattern will be like this for the integral:

$$\begin{aligned} \int \sin^n x \cos^m x dx &= \text{sgn}_0 * \frac{\sin^{n+1} x}{n+1} + \text{sgn}_1 * \frac{\sin^{n+3} x}{n+3} + \dots \\ &+ \text{sgn}_{\frac{m-1}{2}} * \frac{\sin^{n+m} x}{n+m} \end{aligned}$$

We start with $\text{sgn}_0 = -1$ and it is alternating at every iteration.

viii. $\int \tan^n x \sec^m x dx$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 1$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^1 x \sec^1 x dx &= \frac{1}{\cos(x)} \\ \int \tan^1 x \sec^2 x dx &= \frac{1}{2 \cos^2(x)} \\ \int \tan^1 x \sec^3 x dx &= \frac{1}{3 \cos^3(x)} \\ \int \tan^1 x \sec^4 x dx &= \frac{1}{4 \cos^4(x)} \\ \int \tan^1 x \sec^5 x dx &= \frac{1}{5 \cos^5(x)} \\ \int \tan^1 x \sec^6 x dx &= \frac{1}{6 \cos^6(x)} \\ \int \tan^1 x \sec^7 x dx &= \frac{1}{7 \cos^7(x)} \\ \int \tan^1 x \sec^8 x dx &= \frac{1}{8 \cos^8(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 2$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^2 x \sec^1 x dx &= \frac{\log(\sin(x) - 1)}{4} - \frac{\log(\sin(x) + 1)}{4} - \frac{\sin(x)}{2 \sin^2(x) - 2} \\ \int \tan^2 x \sec^2 x dx &= -\frac{\sin(x)}{3 \cos(x)} + \frac{\sin(x)}{3 \cos^3(x)} \\ \int \tan^2 x \sec^3 x dx &= \frac{\log(\sin(x) - 1)}{16} - \frac{\log(\sin(x) + 1)}{16} - \frac{-\sin^3(x) - \sin(x)}{8 \sin^4(x) - 16 \sin^2(x) + 8} \\ \int \tan^2 x \sec^4 x dx &= -\frac{2 \sin(x)}{15 \cos(x)} - \frac{\sin(x)}{15 \cos^3(x)} + \frac{\sin(x)}{5 \cos^5(x)} \\ \int \tan^2 x \sec^5 x dx &= \frac{\log(\sin(x) - 1)}{32} - \frac{\log(\sin(x) + 1)}{32} + \frac{3 \sin^5(x) - 8 \sin^3(x) - 3 \sin(x)}{48 \sin^6(x) - 144 \sin^4(x) + 144 \sin^2(x) - 48} \\ \int \tan^2 x \sec^6 x dx &= -\frac{8 \sin(x)}{105 \cos(x)} - \frac{4 \sin(x)}{105 \cos^3(x)} - \frac{\sin(x)}{35 \cos^5(x)} + \frac{\sin(x)}{7 \cos^7(x)} \\ \int \tan^2 x \sec^7 x dx &= \frac{5 \log(\sin(x) - 1)}{256} - \frac{5 \log(\sin(x) + 1)}{256} \\ &\quad - \frac{-15 \sin^7(x) + 55 \sin^5(x) - 73 \sin^3(x) - 15 \sin(x)}{384 \sin^8(x) - 1536 \sin^6(x) + 2304 \sin^4(x) - 1536 \sin^2(x) + 384} \\ \int \tan^2 x \sec^8 x dx &= -\frac{16 \sin(x)}{315 \cos(x)} - \frac{8 \sin(x)}{315 \cos^3(x)} - \frac{2 \sin(x)}{105 \cos^5(x)} - \frac{\sin(x)}{63 \cos^7(x)} + \frac{\sin(x)}{9 \cos^9(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 3$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^3 x \sec^1 x dx &= \frac{1 - 3 \cos^2(x)}{3 \cos^3(x)} \\ \int \tan^3 x \sec^2 x dx &= \frac{1 - 2 \cos^2(x)}{4 \cos^4(x)} \\ \int \tan^3 x \sec^3 x dx &= \frac{3 - 5 \cos^2(x)}{15 \cos^5(x)} \\ \int \tan^3 x \sec^4 x dx &= \frac{2 - 3 \cos^2(x)}{12 \cos^6(x)} \\ \int \tan^3 x \sec^5 x dx &= \frac{5 - 7 \cos^2(x)}{35 \cos^7(x)} \\ \int \tan^3 x \sec^6 x dx &= \frac{3 - 4 \cos^2(x)}{24 \cos^8(x)} \\ \int \tan^3 x \sec^7 x dx &= \frac{7 - 9 \cos^2(x)}{63 \cos^9(x)} \\ \int \tan^3 x \sec^8 x dx &= \frac{4 - 5 \cos^2(x)}{40 \cos^{10}(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 4$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^4 x \sec^1 x dx &= -\frac{3 \log(\sin(x) - 1)}{16} + \frac{3 \log(\sin(x) + 1)}{16} - \frac{-5 \sin^3(x) + 3 \sin(x)}{8 \sin^4(x) - 16 \sin^2(x) + 8} \\ \int \tan^4 x \sec^2 x dx &= \frac{\sin(x)}{5 \cos(x)} - \frac{2 \sin(x)}{5 \cos^3(x)} + \frac{\sin(x)}{5 \cos^5(x)} \\ \int \tan^4 x \sec^3 x dx &= -\frac{\log(\sin(x) - 1)}{32} + \frac{\log(\sin(x) + 1)}{32} - \frac{-3 \sin^5(x) - 8 \sin^3(x) + 3 \sin(x)}{48 \sin^6(x) - 144 \sin^4(x) + 144 \sin^2(x) - 48} \\ \int \tan^4 x \sec^4 x dx &= \frac{2 \sin(x)}{35 \cos(x)} + \frac{\sin(x)}{35 \cos^3(x)} - \frac{8 \sin(x)}{35 \cos^5(x)} + \frac{\sin(x)}{7 \cos^7(x)} \\ \int \tan^4 x \sec^5 x dx &= -\frac{3 \log(\sin(x) - 1)}{256} + \frac{3 \log(\sin(x) + 1)}{256} \\ &\quad - \frac{3 \sin^7(x) - 11 \sin^5(x) - 11 \sin^3(x) + 3 \sin(x)}{128 \sin^8(x) - 512 \sin^6(x) + 768 \sin^4(x) - 512 \sin^2(x) + 128} \\ \int \tan^4 x \sec^6 x dx &= \frac{8 \sin(x)}{315 \cos(x)} + \frac{4 \sin(x)}{315 \cos^3(x)} + \frac{\sin(x)}{105 \cos^5(x)} - \frac{10 \sin(x)}{63 \cos^7(x)} + \frac{\sin(x)}{9 \cos^9(x)} \\ \int \tan^4 x \sec^7 x dx &= -\frac{3 \log(\sin(x) - 1)}{512} + \frac{3 \log(\sin(x) + 1)}{512} \\ &\quad + \frac{-15 \sin^9(x) + 70 \sin^7(x) - 128 \sin^5(x) - 70 \sin^3(x) + 15 \sin(x)}{1280 \sin^{10}(x) - 6400 \sin^8(x) + 12800 \sin^6(x) - 12800 \sin^4(x) + 6400 \sin^2(x) - 1280} \\ \int \tan^4 x \sec^8 x dx &= \frac{16 \sin(x)}{1155 \cos(x)} + \frac{8 \sin(x)}{1155 \cos^3(x)} + \frac{2 \sin(x)}{385 \cos^5(x)} + \frac{\sin(x)}{231 \cos^7(x)} - \frac{4 \sin(x)}{33 \cos^9(x)} \\ &\quad + \frac{\sin(x)}{11 \cos^{11}(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 5$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^5 x \sec^1 x dx &= \frac{-(-15 \cos^4(x) + 10 \cos^2(x) - 3)}{15 \cos^5(x)} \\ \int \tan^5 x \sec^2 x dx &= \frac{-(-3 \cos^4(x) + 3 \cos^2(x) - 1)}{6 \cos^6(x)} \\ \int \tan^5 x \sec^3 x dx &= \frac{-(-35 \cos^4(x) + 42 \cos^2(x) - 15)}{105 \cos^7(x)} \\ \int \tan^5 x \sec^4 x dx &= \frac{-(-6 \cos^4(x) + 8 \cos^2(x) - 3)}{24 \cos^8(x)} \\ \int \tan^5 x \sec^5 x dx &= \frac{-(-63 \cos^4(x) + 90 \cos^2(x) - 35)}{315 \cos^9(x)} \\ \int \tan^5 x \sec^6 x dx &= \frac{-(-10 \cos^4(x) + 15 \cos^2(x) - 6)}{60 \cos^{10}(x)} \\ \int \tan^5 x \sec^7 x dx &= \frac{-(-99 \cos^4(x) + 154 \cos^2(x) - 63)}{693 \cos^{11}(x)} \\ \int \tan^5 x \sec^8 x dx &= \frac{-(-15 \cos^4(x) + 24 \cos^2(x) - 10)}{120 \cos^{12}(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 6$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^6 x \sec^1 x dx &= \frac{5 \log(\sin(x) - 1)}{32} - \frac{5 \log(\sin(x) + 1)}{32} + \frac{-33 \sin^5(x) + 40 \sin^3(x) - 15 \sin(x)}{48 \sin^6(x) - 144 \sin^4(x) + 144 \sin^2(x) - 48} \\ \int \tan^6 x \sec^2 x dx &= -\frac{\sin(x)}{7 \cos(x)} + \frac{3 \sin(x)}{7 \cos^3(x)} - \frac{3 \sin(x)}{7 \cos^5(x)} + \frac{\sin(x)}{7 \cos^7(x)} \\ \int \tan^6 x \sec^3 x dx &= \frac{5 \log(\sin(x) - 1)}{256} - \frac{5 \log(\sin(x) + 1)}{256} \\ &\quad - \frac{-15 \sin^7(x) - 73 \sin^5(x) + 55 \sin^3(x) - 15 \sin(x)}{384 \sin^8(x) - 1536 \sin^6(x) + 2304 \sin^4(x) - 1536 \sin^2(x) + 384} \\ \int \tan^6 x \sec^4 x dx &= -\frac{2 \sin(x)}{63 \cos(x)} - \frac{\sin(x)}{63 \cos^3(x)} + \frac{5 \sin(x)}{21 \cos^5(x)} - \frac{19 \sin(x)}{63 \cos^7(x)} + \frac{\sin(x)}{9 \cos^9(x)} \\ \int \tan^6 x \sec^5 x dx &= \frac{3 \log(\sin(x) - 1)}{512} - \frac{3 \log(\sin(x) + 1)}{512} \\ &\quad + \frac{15 \sin^9(x) - 70 \sin^7(x) - 128 \sin^5(x) + 70 \sin^3(x) - 15 \sin(x)}{1280 \sin^{10}(x) - 6400 \sin^8(x) + 12800 \sin^6(x) - 12800 \sin^4(x) + 6400 \sin^2(x) - 1280} \\ \int \tan^6 x \sec^6 x dx &= -\frac{8 \sin(x)}{693 \cos(x)} - \frac{4 \sin(x)}{693 \cos^3(x)} - \frac{\sin(x)}{231 \cos^5(x)} + \frac{113 \sin(x)}{693 \cos^7(x)} \\ &\quad - \frac{23 \sin(x)}{99 \cos^9(x)} + \frac{\sin(x)}{11 \cos^{11}(x)} \\ \int \tan^6 x \sec^7 x dx &= \frac{5 \log(\sin(x) - 1)}{2048} - \frac{5 \log(\sin(x) + 1)}{2048} \\ &\quad - \frac{-15 \sin^{11}(x) + 85 \sin^9(x) - 198 \sin^7(x) - 198 \sin^5(x) + 85 \sin^3(x) - 15 \sin(x)}{3072 \sin^{12}(x) - 18432 \sin^{10}(x) + 46080 \sin^8(x) - 61440 \sin^6(x) + 46080 \sin^4(x) - 18432 \sin^2(x) + 3072}\end{aligned}$$

$$\begin{aligned}\int \tan^6 x \sec^8 x dx = & -\frac{16 \sin(x)}{3003 \cos(x)} - \frac{8 \sin(x)}{3003 \cos^3(x)} - \frac{2 \sin(x)}{1001 \cos^5(x)} - \frac{5 \sin(x)}{3003 \cos^7(x)} \\ & + \frac{53 \sin(x)}{429 \cos^9(x)} - \frac{27 \sin(x)}{143 \cos^{11}(x)} + \frac{\sin(x)}{13 \cos^{13}(x)}\end{aligned}$$

These are the formula of $\int \tan^n x \sec^m x dx$ with $n = 7$ and $m = 1$ to $m = 8$.

$$\begin{aligned}\int \tan^7 x \sec^1 x dx &= \frac{-35 \cos^6(x) + 35 \cos^4(x) - 21 \cos^2(x) + 5}{35 \cos^7(x)} \\ \int \tan^7 x \sec^2 x dx &= \frac{-4 \cos^6(x) + 6 \cos^4(x) - 4 \cos^2(x) + 1}{8 \cos^8(x)} \\ \int \tan^7 x \sec^3 x dx &= \frac{-105 \cos^6(x) + 189 \cos^4(x) - 135 \cos^2(x) + 35}{315 \cos^9(x)} \\ \int \tan^7 x \sec^4 x dx &= \frac{-10 \cos^6(x) + 20 \cos^4(x) - 15 \cos^2(x) + 4}{40 \cos^{10}(x)} \\ \int \tan^7 x \sec^5 x dx &= \frac{-231 \cos^6(x) + 495 \cos^4(x) - 385 \cos^2(x) + 105}{1155 \cos^{11}(x)} \\ \int \tan^7 x \sec^6 x dx &= \frac{-20 \cos^6(x) + 45 \cos^4(x) - 36 \cos^2(x) + 10}{120 \cos^{12}(x)} \\ \int \tan^7 x \sec^7 x dx &= \frac{-429 \cos^6(x) + 1001 \cos^4(x) - 819 \cos^2(x) + 231}{3003 \cos^{13}(x)} \\ \int \tan^7 x \sec^8 x dx &= \frac{-35 \cos^6(x) + 84 \cos^4(x) - 70 \cos^2(x) + 20}{280 \cos^{14}(x)}\end{aligned}$$

Case 1: For $\int \tan^n x \sec^m x dx$ with n even

We divide this case into 2 smaller cases:

1. When m even
2. When m odd

Case 1-a: For $\int \tan^n x \sec^m x dx$ with n even and m even

$$\begin{aligned}
 \int \tan^2 x \sec^2 x dx &= -\frac{\sin(x)}{3\cos(x)} + \frac{\sin(x)}{3\cos^3(x)} \\
 \int \tan^2 x \sec^4 x dx &= -\frac{2\sin(x)}{15\cos(x)} - \frac{\sin(x)}{15\cos^3(x)} + \frac{\sin(x)}{5\cos^5(x)} \\
 \int \tan^2 x \sec^6 x dx &= -\frac{8\sin(x)}{105\cos(x)} - \frac{4\sin(x)}{105\cos^3(x)} - \frac{\sin(x)}{35\cos^5(x)} + \frac{\sin(x)}{7\cos^7(x)} \\
 \int \tan^2 x \sec^8 x dx &= -\frac{16\sin(x)}{315\cos(x)} - \frac{8\sin(x)}{315\cos^3(x)} - \frac{2\sin(x)}{105\cos^5(x)} - \frac{\sin(x)}{63\cos^7(x)} + \frac{\sin(x)}{9\cos^9(x)} \\
 \int \tan^4 x \sec^2 x dx &= \frac{\sin(x)}{5\cos(x)} - \frac{2\sin(x)}{5\cos^3(x)} + \frac{\sin(x)}{5\cos^5(x)} \\
 \int \tan^4 x \sec^4 x dx &= \frac{2\sin(x)}{35\cos(x)} + \frac{\sin(x)}{35\cos^3(x)} - \frac{8\sin(x)}{35\cos^5(x)} + \frac{\sin(x)}{7\cos^7(x)} \\
 \int \tan^4 x \sec^6 x dx &= \frac{8\sin(x)}{315\cos(x)} + \frac{4\sin(x)}{315\cos^3(x)} + \frac{\sin(x)}{105\cos^5(x)} - \frac{10\sin(x)}{63\cos^7(x)} + \frac{\sin(x)}{9\cos^9(x)} \\
 \int \tan^4 x \sec^8 x dx &= \frac{16\sin(x)}{1155\cos(x)} + \frac{8\sin(x)}{1155\cos^3(x)} + \frac{2\sin(x)}{385\cos^5(x)} + \frac{\sin(x)}{231\cos^7(x)} - \frac{4\sin(x)}{33\cos^9(x)} \\
 &\quad + \frac{\sin(x)}{11\cos^{11}(x)} \\
 \int \tan^6 x \sec^2 x dx &= -\frac{\sin(x)}{7\cos(x)} + \frac{3\sin(x)}{7\cos^3(x)} - \frac{3\sin(x)}{7\cos^5(x)} + \frac{\sin(x)}{7\cos^7(x)} \\
 \int \tan^6 x \sec^4 x dx &= -\frac{2\sin(x)}{63\cos(x)} - \frac{\sin(x)}{63\cos^3(x)} + \frac{5\sin(x)}{21\cos^5(x)} - \frac{19\sin(x)}{63\cos^7(x)} + \frac{\sin(x)}{9\cos^9(x)} \\
 \int \tan^6 x \sec^6 x dx &= -\frac{8\sin(x)}{693\cos(x)} - \frac{4\sin(x)}{693\cos^3(x)} - \frac{\sin(x)}{231\cos^5(x)} + \frac{113\sin(x)}{693\cos^7(x)} \\
 &\quad - \frac{23\sin(x)}{99\cos^9(x)} + \frac{\sin(x)}{11\cos^{11}(x)} \\
 \int \tan^6 x \sec^8 x dx &= -\frac{16\sin(x)}{3003\cos(x)} - \frac{8\sin(x)}{3003\cos^3(x)} - \frac{2\sin(x)}{1001\cos^5(x)} - \frac{5\sin(x)}{3003\cos^7(x)} \\
 &\quad + \frac{53\sin(x)}{429\cos^9(x)} - \frac{27\sin(x)}{143\cos^{11}(x)} + \frac{\sin(x)}{13\cos^{13}(x)}
 \end{aligned}$$

The patterns that can be seen from those equations above are

- The series are consisting of a function with $\sin(x)$ as the numerator and $\cos(x)$ as the denominator, the length of the series and the maximum power for the denominator depending on m and n .
- If you take a look at the $\int \sec^m dx$ for m even, you will notice that this case will use the result of $\int \sec^m dx$ for m even depending on n and using the combination to make the Pascal' triangle.

For example, let $n = 4$ and $m = 6$

$$\begin{aligned}
 \int \tan^4 x \sec^6 x \, dx &= \int (\sec^2 x - 1)^2 \sec^6 x \, dx \\
 &= \int (\sec^4 x - 2 \sec^2 x + 1) \sec^6 x \, dx \\
 &= \int \sec^{10} x - 2 \sec^8 x + \sec^6 x \, dx \\
 &= \frac{8 \sin(x)}{315 \cos(x)} + \frac{4 \sin(x)}{315 \cos^3(x)} + \frac{\sin(x)}{105 \cos^5(x)} - \frac{10 \sin(x)}{63 \cos^7(x)} + \frac{\sin(x)}{9 \cos^9(x)}
 \end{aligned}$$

In the end we will refer to the formula for $\int \sec^m dx$ for m even that starts the power of the secant and end at the total power of the secant and tangent, with.

```

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

using namespace std;

// Factorial and combinations
Symbolic factorial(int n) {
    if (n <= 1)
    {
        return 1;
    }
    Symbolic result = 1;
    for (int i = 2; i <= n; ++i)
    {
        result *= i;
    }
    return result;
}

Symbolic combinations(int n, int r) {
    if (r < 0 || r > n)
    {
        return 0; // Invalid input
    }
    return factorial(n) / (factorial(r) * factorial(n - r));
}

Symbolic integralsecanteven(int n) {
    Symbolic x("x");
    int v[999];
    v[0] = 1;
    Symbolic integral;
    Symbolic dθ = 1;
    int k = 1, l = 2, c=1;
    for(int i = 1 ; i < n ; i = i+2) // to compute the denominator
    {
        v[i] = v[i-2] * (l * l - 1);
        l += 2;
    }
    integral = v[n-1];
    for(int i = 1 ; i < n ; i = i+2)
    {
        integral -= v[i] * v[i-2];
    }
    integral *= dθ;
    integral.simplify();
    return integral;
}

```

```

{
    dθ *= i;
}
for(int i = 1 ; i < n - 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*n)+1; i = i+1)
{
    integral += ( v[i-1] * sin(x) ) / ( dθ*(cos(x)^(n-j_d)) )
    ;
    j_d = j_d+2;
}
return integral;
}

int main(void)
{
    Symbolic x("x");
    int bpowersec = 4;
    int bpowertan = 6;
    int bpower = bpowersec+bpowertan;
    Symbolic result;
    Symbolic sgn = 1;
    int j = 1;
    int m = bpowertan/2;
    for(int i = bpower ; i >= bpowersec ; i = i-2)

```

```
{  
    result += sgn*combinations(m,j-1)*integralsecanteven(i);  
    sgn = -sgn;  
    j = j+1;  
}  
cout << "integral = "<< result << endl;  
//cout << "integral with subs = " << result[x==1] << endl;  
return 0;  
}
```

Code 24: level 4 integral for tangent even power and secant even power case

Case 1-b: For $\int \tan^n x \sec^m x dx$ with **n even and m odd**

$$\begin{aligned}
\int \tan^2 x \sec^1 x dx &= \frac{\log(\sin(x) - 1)}{4} - \frac{\log(\sin(x) + 1)}{4} - \frac{\sin(x)}{2\sin^2(x) - 2} \\
\int \tan^2 x \sec^3 x dx &= \frac{\log(\sin(x) - 1)}{16} - \frac{\log(\sin(x) + 1)}{16} - \frac{-\sin^3(x) - \sin(x)}{8\sin^4(x) - 16\sin^2(x) + 8} \\
\int \tan^2 x \sec^5 x dx &= \frac{\log(\sin(x) - 1)}{32} - \frac{\log(\sin(x) + 1)}{32} + \frac{3\sin^5(x) - 8\sin^3(x) - 3\sin(x)}{48\sin^6(x) - 144\sin^4(x) + 144\sin^2(x) - 48} \\
\int \tan^2 x \sec^7 x dx &= \frac{5\log(\sin(x) - 1)}{256} - \frac{5\log(\sin(x) + 1)}{256} \\
&\quad - \frac{-15\sin^7(x) + 55\sin^5(x) - 73\sin^3(x) - 15\sin(x)}{384\sin^8(x) - 1536\sin^6(x) + 2304\sin^4(x) - 1536\sin^2(x) + 384} \\
\int \tan^4 x \sec^1 x dx &= -\frac{3\log(\sin(x) - 1)}{16} + \frac{3\log(\sin(x) + 1)}{16} - \frac{-5\sin^3(x) + 3\sin(x)}{8\sin^4(x) - 16\sin^2(x) + 8} \\
\int \tan^4 x \sec^3 x dx &= -\frac{\log(\sin(x) - 1)}{32} + \frac{\log(\sin(x) + 1)}{32} - \frac{-3\sin^5(x) - 8\sin^3(x) + 3\sin(x)}{48\sin^6(x) - 144\sin^4(x) + 144\sin^2(x) - 48} \\
\int \tan^4 x \sec^5 x dx &= -\frac{3\log(\sin(x) - 1)}{256} + \frac{3\log(\sin(x) + 1)}{256} \\
&\quad - \frac{3\sin^7(x) - 11\sin^5(x) - 11\sin^3(x) + 3\sin(x)}{128\sin^8(x) - 512\sin^6(x) + 768\sin^4(x) - 512\sin^2(x) + 128} \\
\int \tan^4 x \sec^7 x dx &= -\frac{3\log(\sin(x) - 1)}{512} + \frac{3\log(\sin(x) + 1)}{512} \\
&\quad + \frac{-15\sin^9(x) + 70\sin^7(x) - 128\sin^5(x) - 70\sin^3(x) + 15\sin(x)}{1280\sin^{10}(x) - 6400\sin^8(x) + 12800\sin^6(x) - 12800\sin^4(x) + 6400\sin^2(x) - 1280} \\
\int \tan^6 x \sec^1 x dx &= \frac{5\log(\sin(x) - 1)}{32} - \frac{5\log(\sin(x) + 1)}{32} + \frac{-33\sin^5(x) + 40\sin^3(x) - 15\sin(x)}{48\sin^6(x) - 144\sin^4(x) + 144\sin^2(x) - 48} \\
\int \tan^6 x \sec^3 x dx &= \frac{5\log(\sin(x) - 1)}{256} - \frac{5\log(\sin(x) + 1)}{256} \\
&\quad - \frac{-15\sin^7(x) - 73\sin^5(x) + 55\sin^3(x) - 15\sin(x)}{384\sin^8(x) - 1536\sin^6(x) + 2304\sin^4(x) - 1536\sin^2(x) + 384} \\
\int \tan^6 x \sec^5 x dx &= \frac{3\log(\sin(x) - 1)}{512} - \frac{3\log(\sin(x) + 1)}{512} \\
&\quad + \frac{15\sin^9(x) - 70\sin^7(x) - 128\sin^5(x) + 70\sin^3(x) - 15\sin(x)}{1280\sin^{10}(x) - 6400\sin^8(x) + 12800\sin^6(x) - 12800\sin^4(x) + 6400\sin^2(x) - 1280}
\end{aligned}$$

This case is considered as the hardest of all cases in $\int \tan^n x \sec^m x dx$. If you try to solve it by hand, most calculus book will recommend to use method of substitution to replace $\tan^n x$ into $\sec x$ function so we can then compute the integral of secant with odd power, which already been done previously in level 3 integral section.

The secant with odd power make use of middle coefficient, the sum of two neighbor entry in the vector that will become the constant / coefficient for the numerator with sine function that has odd power.

Here, we will do the same, with an adding of some tricks.

The patterns that can be seen from those equations above are

- The terms are divided into 3, the same like integral of secant with odd power which are:
 1. The numerator with $\sin(x)$ function that has odd power
 2. The denominator with $\sin(x)$ function that has even power
 3. The log terms
- For all $\int \tan^n x \sec^m x dx$, when $m = 1$ the result is the same / equivalent with $\int \sec^{n+1} x dx$.

To solve for any m that can be any number but 1, we will need to make another **for** loop to adjust the coefficient.

So the starting is we will use $\int \sec^{n+1} x dx$ that we already know the pattern.

- The denominator coefficients have the exact same pattern with the integral of secant with odd power, but this time we will make the denominator coefficient to run another loop again till we reach m .

To learn the pattern we will try with small n and m , we will make use of

$$\tan^2 x = \sec^2 x - 1$$

to get everything in terms of $\sec x$.

For $n = 2$ and $m = 3$

$$\begin{aligned} \int \tan^2 x \sec^3 x dx &= (\sec^2 x - 1) \sec^3 x dx \\ &= \int \sec^5 x - \sec^3 x dx \\ &= \left[-\frac{3 \log(\sin(x) - 1)}{16} + \frac{3 \log(\sin(x) + 1)}{16} - \frac{3 \sin^3 x - 5 \sin x}{8 \sin^4 x - 16 \sin^2 x + 8} \right] \\ &\quad - \left[-\frac{\log(\sin(x) - 1)}{4} + \frac{\log(\sin(x) + 1)}{4} - \frac{\sin x}{2 \sin^2 x - 2} \right] \\ &= -\frac{\log(\sin(x) - 1)}{16} + \frac{\log(\sin(x) + 1)}{16} \\ &\quad - \frac{3 \sin^3 x - 5 \sin x - (4 \sin^2 x - 4)(\sin x)}{8 \sin^4 x - 16 \sin^2 x + 8} \\ &= -\frac{\log(\sin(x) - 1)}{16} + \frac{\log(\sin(x) + 1)}{16} - \frac{-\sin^3 x - \sin x}{8 \sin^4 x - 16 \sin^2 x + 8} \end{aligned}$$

For $n = 4$ and $m = 3$

$$\begin{aligned} \int \tan^4 x \sec^3 x dx &= (\sec^2 x - 1)^2 \sec^3 x dx \\ &= \int \sec^3 x (\sec^4 x - 2 \sec^2 x + 1) dx \\ &= \int \sec^3 x - 2 \sec^5 x + \sec^7 x dx \\ &= -\frac{\log(\sin(x) - 1)}{32} + \frac{\log(\sin(x) + 1)}{32} + \frac{-3 \sin^5 x - 8 \sin^3 x + 3 \sin x}{48 \sin^6 x - 144 \sin^4 x + 144 \sin^2 x - 48} \end{aligned}$$

If we take a closer look and observe at this part $\int \sec^3 x - 2 \sec^5 x + \sec^7 x \, dx$, we are seeing a binomial coefficient / pascal triangle with alternating sign, thus we can use this pattern to compute $\int \tan^n x \sec^m x \, dx$ with n even and m odd for any number of n and m .

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

using namespace std;

// Factorial and combinations
Symbolic factorial(int n) {
    if (n <= 1)
    {
        return 1;
    }
    Symbolic result = 1;
    for (int i = 2; i <= n; ++i)
    {
        result *= i;
    }
    return result;
}

Symbolic combinations(int n, int r) {
    if (r < 0 || r > n)
    {
        return 0; // Invalid input
    }
    return factorial(n) / (factorial(r) * factorial(n - r));
}

Symbolic numeratorintegralsecantodd(int n) {
    Symbolic x("x");
    int k = 1, l=2;
    vector<int> v={1};
    vector<int> mc={1}; // to store the middle coefficient
    Symbolic sgn = -1;
    Symbolic integral_numerator;
    Symbolic d0 = 2, d1 = 2;
    int j =1;
    int m = n-(0.5*(n+1));
    int last_coeff;
    int first_coeff = 3;

    // For the coefficient at the numerator of sine with odd power
    for(int i = 1 ; i < (n-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 = n-2 ; i >= 1 ; i = i-2)
    {
        integral_numerator += sgn*v[j_num]*((sin(x))^(i));
        sgn = -sgn;
        j_num = j_num+1;
    }
    return integral_numerator;
}

Symbolic denominatorintegralsecantodd(int n) {
    Symbolic x("x");
    int k = 1, l=2;
    vector<int> v={1};
    vector<int> mc={1}; // to store the middle coefficient
    Symbolic sgn = 1;
    Symbolic integral_denominator;
    Symbolic d0 = 2, d1 = 2;
    int j =1;
    int m = n-(0.5*(n+1));
    int last_coeff;
    int first_coeff = 3;

    // For the coefficient at the numerator of sine with odd power
    for(int i = 1 ; i < (n-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;
}
j = 1;
for(int i = n-1 ; i >= 0 ; i = i-2)
{
    integral_denominator += sgn*d0*combinations(m,j-1)*((sin(x)
        ))^(i));
    sgn = -sgn;
    j = j+1;
}
return integral_denominator;
}

Symbolic logtermsintegralsecantodd(int n) {
    Symbolic x("x");
    int k = 1, l=2;
    vector<int> v={1};
    vector<int> mc={1}; // to store the middle coefficient
    Symbolic sgn = -1;
    Symbolic integral_denominator;
    Symbolic d0 = 2, d1 = 2;
    int j =1;
    int m = n-(0.5*(n+1));
    int last_coeff;
    int first_coeff = 3;
}

```

```

// For the coefficient at the numerator of sine with odd power
for(int i = 1 ; i < (n-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;
}
d1 = d1*(n-1);
return (-v[0]*ln(sin(x)-1))/(d1) + (v[0]*ln(sin(x)+1))/(d1) ;
}

int main(void)
{
    Symbolic x("x");
    int bpowersec = 3;
    int bpowertan = 6;
    int bpower = bpowersec+bpowertan;
    Symbolic sgn = 1;
    Symbolic result;
    int j = 1;
    int m = bpowertan/2;
    for(int i = bpower ; i >= bpowersec ; i = i-2)
    {
        result += sgn*combinations(m,j-1)*(
            numeratorintegralsecantodd(i)/
            denominatorintegralsecantodd(i)) +
            logtermsintegralsecantodd(i));
        sgn = -sgn;
    }
}

```

```

        j = j+1;
    }
    //cout << "integral at numerator= "<< numeratorintegralsecantodd
    //      (3)<< endl;
    //cout << "integral at denominator= "<<
    //      denominatorintegralsecantodd(3)<< endl;
    //cout << "log terms= "<< logtermsintegralsecantodd(3)<< endl;
    cout << "for secant power of " << bpowersec << " and tangent power
          of " << bpowertan << endl;
    cout << "integral = "<< result << endl;
    //cout << "integral with subs = " << result[x==1] << endl;
    return 0;
}

```

Code 25: level 4 integral for tangent even power and secant odd power case

It is recommended that you use the:

```

cout << "integral at numerator= "<< numeratorintegralsecantodd(3)<< endl;
cout << "integral at denominator= "<< denominatorintegralsecantodd(3)<< endl;
cout << "log terms= "<< logtermsintegralsecantodd(3)<< endl;
(uncomment them in the C++ code)

```

and comment the :

```

cout << "for secant power of " << bpowersec << " and tangent power of " << bpowertan <<
endl;
cout << "integral = "<< result << endl;

```

Because it is faster to show the result when we only show the log terms function, numerator only / denominator only of the fraction with sine function than to show it in the full fraction function, it is still slow and we wonder why, because SymPy is able to show the result very fast, knowing that C++ is basically / by default faster than SymPy we might suspect because it is the fraction that we are using combining with symbolic computation, if we only show the vector / array, the result can come up very fast, faster than SymPy.

We will figure it out and make it really fast one day. Any reader interested is always welcome to give a nice input for this.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tan^n for n Even m Odd with Functions ]# make
g++ -c main.cpp
g++ -o main main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tan^n for n Even m Odd with Functions ]# ./main
integral = -105*sin(x)^7*(384*sin(x)^8)-1536*sin(x)^6+2304*sin(x)^4-1536*sin(x)^2+384)^(-1)+385*sin(x)^5*(384*sin(x)^8)-1536*sin(x)^6+2304*sin(x)^4-1536*sin(x)^2+384)^(-1)-511*sin(x)^3*(384*sin(x)^8)-1536*sin(x)^6+2304*sin(x)^4-1536*sin(x)^2+384)^(-1)+279*sin(x)*(384*sin(x)^8)-1536*sin(x)^6+2304*sin(x)^4-1536*sin(x)^2+384)^(-1)+5/256*ln(sin(x))-5/256*ln(sin(x)^2)-5/256*ln(sin(x)^4)-5/256*ln(sin(x)^6)-120*sin(x)^5*(48*sin(x)^6)-144*sin(x)^4+144*sin(x)^2-48)^(-1)+99*sin(x)*(48*sin(x)^6)-144*sin(x)^4+144*sin(x)^2-48)^(-1)-9*sin(x)^3*(8*sin(x)^4)-16*sin(x)^2+8)*(-1)+15*sin(x)*(8*sin(x)^4)-16*sin(x)^2+8)^(-1)+sin(x)*(2*sin(x)^2-2)^(-1)
julia> integrate((sec(x)^3)*(tan(x)^6),x)
integral of cot(3x+5) = -log(tan(3*x + 5)^2 + 1)/6 + log(tan(3*x + 5))/3
integral of cot(10x-2) = -log(tan(10*x - 2)^2 + 1)/20 + log(tan(10*x - 2))/10
julia> integrate((sec(x)^7)*(tan(x)^5),x)
- 15·sin(x) - 73·sin(x) + 55·sin(x) - 15·sin(x) + 5·log(sin(x))
8          6          4          2
384·sin(x) - 1536·sin(x) + 2304·sin(x) - 1536·sin(x) + 384
julia> integrate((sec(x)^4)*(tan(x)^6),x)

```

Figure 2.2: The computation of $\int \tan^n x \sec^m x dx$ with SymPy and SymIntegration (SymIntegration/Examples/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tanⁿ for n Even m Odd with Functions/main.cpp).

Case 2: For $\int \tan^n x \sec^m x dx$ with n odd

We divide this case into 2 smaller cases:

1. When m even
2. When m odd

Case 2-a: For $\int \tan^n x \sec^m x dx$ with n odd and m even

$$\int \tan^1 x \sec^2 x dx = \frac{1}{2 \cos^2(x)}$$

$$\int \tan^1 x \sec^4 x dx = \frac{1}{4 \cos^4(x)}$$

$$\int \tan^1 x \sec^6 x dx = \frac{1}{6 \cos^6(x)}$$

$$\int \tan^1 x \sec^8 x dx = \frac{1}{8 \cos^8(x)}$$

$$\int \tan^3 x \sec^2 x dx = \frac{1 - 2 \cos^2(x)}{4 \cos^4(x)}$$

$$\int \tan^3 x \sec^4 x dx = \frac{2 - 3 \cos^2(x)}{12 \cos^6(x)}$$

$$\int \tan^3 x \sec^6 x dx = \frac{3 - 4 \cos^2(x)}{24 \cos^8(x)}$$

$$\int \tan^3 x \sec^8 x dx = \frac{4 - 5 \cos^2(x)}{40 \cos^{10}(x)}$$

$$\int \tan^5 x \sec^2 x dx = \frac{-(-3 \cos^4(x) + 3 \cos^2(x) - 1)}{6 \cos^6(x)}$$

$$\int \tan^5 x \sec^4 x dx = \frac{-(-6 \cos^4(x) + 8 \cos^2(x) - 3)}{24 \cos^8(x)}$$

$$\int \tan^5 x \sec^6 x dx = \frac{-(-10 \cos^4(x) + 15 \cos^2(x) - 6)}{60 \cos^{10}(x)}$$

$$\int \tan^5 x \sec^8 x dx = \frac{-(-15 \cos^4(x) + 24 \cos^2(x) - 10)}{120 \cos^{12}(x)}$$

The patterns that can be seen from those equations above are

- They are all function that only consist of function of cosine and constants.
- The power of cosine in the denominator is $m + n - 1$.
- You can solve it easily to determine the pattern by substituting

$$\sec^2 x = \tan^2 x + 1$$

then we will have the terms in $\tan(x)$ with odd power only, which we already have learned the pattern from the previous section.

For example, let $m = 4, n = 3$

$$\begin{aligned} \int \tan^3 x \sec^4 x dx &= \int \tan^3 x (\tan^2 x + 1)^2 dx \\ &= \int \tan^3 x (\tan^4 x + 2\tan^2 x + 1) dx \\ &= \int \tan^7 x + 2\tan^5 x + \tan^3 x dx \end{aligned}$$

It is using combination / Pascal' triangle again, this time the sign is the same, all positive.

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

using namespace std;

// Factorial and combinations
Symbolic factorial(int n) {
    if (n <= 1)
    {
        return 1;
    }
    Symbolic result = 1;
    for (int i = 2; i <= n; ++i)
    {
        result *= i;
    }
    return result;
}

Symbolic combinations(int n, int r) {
    if (r < 0 || r > n)
    {
        return 0; // Invalid input
    }
    return factorial(n) / (factorial(r) * factorial(n - r));
}

Symbolic integraltangentodd(int n) {
```

```

Symbolic x("x");
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 <= (n-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 < 1 ; ic = ic+1)
    {
        mc_temp[ic] = mc[ic-1] + mc[ic];
    }
    mc[1] = (mc_temp[0]);
    mc[1+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 <= (n-1)/2 ; i = i+1)
{
    integral += sgn*v[i-1]*(cos(x)^(2*(i-1)));
    sgn=-sgn;

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

```

```

int main(void)
{
    Symbolic x("x");
    int bpowersec = 4;
    int bpowertan = 5;
    int bpower = bpowersec+bpowertan;
    Symbolic result;
    int j = 1;
    int m = bpowersec/2;
    for(int i = bpowertan ; i <= bpower ; i = i+2)
    {
        result += combinations(m,j-1)*integraltagentodd(i);
        j = j+1;
    }
    cout << "for secant power of " << bpowersec << " and tangent power
          of " << bpowertan << endl;
    cout << "integral = "<< result<< endl;
    return 0;
}

```

Code 26: level 4 integral for tangent odd power and secant even power case

Case 2-b: For $\int \tan^n x \sec^m x dx$ with n odd and m odd

$$\begin{aligned}\int \tan^1 x \sec^1 x dx &= \frac{1}{\cos(x)} \\ \int \tan^1 x \sec^3 x dx &= \frac{1}{3\cos^3(x)} \\ \int \tan^1 x \sec^5 x dx &= \frac{1}{5\cos^5(x)} \\ \int \tan^1 x \sec^7 x dx &= \frac{1}{7\cos^7(x)}\end{aligned}$$

$$\begin{aligned}\int \tan^3 x \sec^1 x dx &= \frac{1 - 3\cos^2(x)}{3\cos^3(x)} \\ \int \tan^3 x \sec^3 x dx &= \frac{3 - 5\cos^2(x)}{15\cos^5(x)} \\ \int \tan^3 x \sec^5 x dx &= \frac{5 - 7\cos^2(x)}{35\cos^7(x)} \\ \int \tan^3 x \sec^7 x dx &= \frac{7 - 9\cos^2(x)}{63\cos^9(x)}\end{aligned}$$

$$\begin{aligned}\int \tan^5 x \sec^1 x dx &= \frac{-(-15 \cos^4(x) + 10 \cos^2(x) - 3)}{15 \cos^5(x)} \\ \int \tan^5 x \sec^3 x dx &= \frac{-(-35 \cos^4(x) + 42 \cos^2(x) - 15)}{105 \cos^7(x)} \\ \int \tan^5 x \sec^5 x dx &= \frac{-(-63 \cos^4(x) + 90 \cos^2(x) - 35)}{315 \cos^9(x)} \\ \int \tan^5 x \sec^7 x dx &= \frac{-(-99 \cos^4(x) + 154 \cos^2(x) - 63)}{693 \cos^{11}(x)}\end{aligned}$$

The patterns that can be seen from those equations above are

- They are all function that only consist of function of cosine and constants.
- The power of cosine in the denominator is $m + n - 1$.
- You can solve it easily to determine the pattern by substituting

$$\tan^2 x = \sec^2 x - 1$$

and then use method of substitution by assigning

$$\begin{aligned}u &= \sec x \\ du &= \sec x \tan x dx\end{aligned}$$

then we will have the terms in $\sec(x)$ with even power only, which we already have learned the pattern from the previous section.

For example, let $m = 3, n = 5$

$$\begin{aligned}\int \tan^5 x \sec^3 x dx &= \int \tan^4 x \sec^2 x (\sec x \tan x dx) \\ &= \int [(\sec^2 x - 1)^2 \sec^2 x] (\sec x \tan x dx) \\ &= \int [(\sec^4 x - 2 \sec^2 x + 1) \sec^2 x] (\sec x \tan x dx) \\ &= \int [\sec^6 x - 2 \sec^4 x + \sec^2 x] (\sec x \tan x dx) \\ &= \int u^6 - 2u^4 + u^2 du \\ &= \frac{1}{7}u^7 - \frac{2}{5}u^5 + \frac{1}{3}u^3 \\ &= \frac{1}{7}\sec^7 x - \frac{2}{5}\sec^5 x + \frac{1}{3}\sec^3 x \\ &= \frac{1}{7}\frac{1}{\cos^7 x} - \frac{2}{5}\frac{1}{\cos^5 x} + \frac{1}{3}\frac{1}{\cos^3 x}\end{aligned}$$

It is using combination / Pascal' triangle combining with a method of substitution, the sign is alternating this time. The thing is the method of substitution is a really helpful here, we do not even need to create a function, only integrating ordinary polynomial and then substituting back, we can substitute $u = \sec(x)$ or $u = \frac{1}{\cos(x)}$.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tan^n for n Odd m Odd with Functions ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tan^n for n Odd m Odd with Functions ]# ./main
for secant power of 3 and tangent power of 5
we will compute the integral of u^(2)-2*u^(4)+u^(6)
integral = 1/3*u^(3)-2/5*u^(5)+1/7*u^(7)
Substitute back, integral = 1/3*cos(x)^(-3)-2/5*cos(x)^(-5)+1/7*cos(x)^(-7)
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tan^n for n Odd m Odd with Functions ]# []

```



```

julia> integrate((sec(x)^3)*(tan(x)^5),x)

$$\frac{(-35 \cdot \cos(x) + 42 \cdot \cos(x)^2 - 15)}{105 \cdot \cos(x)^7}$$


```

Figure 2.3: The computation of $\int \tan^5 x \sec^3 x dx$ with SymPy and SymIntegration (SymIntegration/Examples/Test SymIntegration Trigonometry Integration Level 4 Sec^m Tanⁿ for n Odd m Odd/main.cpp).

```

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

using namespace std;

// Factorial and combinations
Symbolic factorial(int n) {
    if (n <= 1)
    {
        return 1;
    }
    Symbolic result = 1;
    for (int i = 2; i <= n; ++i)
    {
        result *= i;
    }
    return result;
}

Symbolic combinations(int n, int r) {
    if (r < 0 || r > n)
    {
        return 0; // Invalid input
    }
    return factorial(n) / (factorial(r) * factorial(n - r));
}

int main(void)
{

```

```

Symbolic u("u"), x("x");
int bpowersec = 3;
int bpowertan = 5;
int bpower = bpowersec+bpowertan;
Symbolic result;
int j = 1;
int m = (bpowertan-1)/2;
Symbolic sgn = 1;
for(int i = bpowersec-1 ; i <= bpower-2 ; i = i+2)
{
    result += sgn*combinations(m,j-1)*(u^i);
    j = j+1;
    sgn = -sgn;
}
cout << "for secant power of " << bpowersec << " and tangent power
      of " << bpowertan << endl;
cout << "we will compute the integral of "<< result << endl;
Symbolic f = integrate(result,u);
cout << "integral = "<< f << endl;
cout << "Substitute back, integral = "<< f[u==(1/cos(x))] << endl;
return 0;
}

```

Code 27: level 4 integral for tangent odd power and secant odd power case

$$\text{ix. } \int \cot^n x \csc^m x \, dx$$

The formula of $\int \cot^n x \csc^m x \, dx$ has similar pattern with $\int \tan^n x \sec^m x \, dx$, we only change from $\sin(x)$ to $\cos(x)$ and from $\cos(x)$ to $\sin(x)$, the sign will be changed too, from minus to plus, and from plus to minus. Once we have decoded the pattern for $\int \tan^n x \sec^m x \, dx$, we only need to modify a bit for this integral type.

```

xterm
^ _ □ ×

julia> integrate((tan(x)^2)*(sec(x)^7),x)

$$-\frac{15 \cdot \sin^7(x) + 55 \cdot \sin^5(x) - 73 \cdot \sin^3(x) - 15 \cdot \sin(x)}{384 \cdot \sin^8(x) - 1536 \cdot \sin^6(x) + 2304 \cdot \sin^4(x) - 1536 \cdot \sin^2(x) + 384} + \frac{5 \cdot \log(\sin(x))}{256}$$


$$-\frac{1}{256} \cdot \frac{5 \cdot \log(\sin(x) + 1)}{5 \cdot \log(\sin(x) + 1)}$$


julia> integrate((cot(x)^2)*(csc(x)^7),x)

$$-\frac{15 \cdot \cos^7(x) + 55 \cdot \cos^5(x) - 73 \cdot \cos^3(x) - 15 \cdot \cos(x)}{384 \cdot \cos^8(x) - 1536 \cdot \cos^6(x) + 2304 \cdot \cos^4(x) - 1536 \cdot \cos^2(x) + 384} - \frac{5 \cdot \log(\cos(x))}{256}$$


$$-\frac{1}{256} + \frac{5 \cdot \log(\cos(x) + 1)}{256}$$


julia> integrate((tan(x)^2)*(sec(x)^8),x)

$$-\frac{16 \cdot \sin(x)}{315 \cdot \cos(x)} - \frac{8 \cdot \sin(x)}{315 \cdot \cos^3(x)} - \frac{2 \cdot \sin(x)}{105 \cdot \cos^5(x)} - \frac{\sin(x)}{63 \cdot \cos^7(x)} + \frac{\sin(x)}{9 \cdot \cos^9(x)}$$


julia> integrate((cot(x)^2)*(csc(x)^8),x)

$$\frac{16 \cdot \cos(x)}{315 \cdot \sin(x)} + \frac{8 \cdot \cos(x)}{315 \cdot \sin^3(x)} + \frac{2 \cdot \cos(x)}{105 \cdot \sin^5(x)} + \frac{\cos(x)}{63 \cdot \sin^7(x)} - \frac{\cos(x)}{9 \cdot \sin^9(x)}$$


```

Figure 2.4: The integral of $\int \cot^n x \csc^m x dx$ that has similar pattern with of $\int \tan^n x \sec^m x dx$.

VII. TRIGONOMETRY AND TRANSCENDENTALS FORMULA

[SI*] We will list all the trigonometry and transcendental (exponential and logarithm) formulas that can be used to help you solve integral problems

[SI*] Trigonometry

Reciprocal identities

$$\begin{aligned}\tan x &= \frac{\sin x}{\cos x} \\ \cot x &= \frac{\cos x}{\sin x} \\ \csc x &= \frac{1}{\sin x} \\ \sec x &= \frac{1}{\cos x}\end{aligned}$$

Pythagorean identities

$$\begin{aligned}\sin^2 x + \cos^2 x &= 1 \\ \tan^2 x &= \sec^2 x - 1 \\ \cot^2 x &= \csc^2 x - 1\end{aligned}$$

Sum and difference identities

$$\begin{aligned}\sin(x+y) &= \sin x \cos y + \cos x \sin y \\ \sin(x-y) &= \sin x \cos y - \cos x \sin y \\ \cos(x+y) &= \cos x \cos y - \sin x \sin y \\ \cos(x-y) &= \cos x \cos y + \sin x \sin y \\ \tan(x+y) &= \frac{\tan x + \tan y}{1 - \tan x \tan y} \\ \tan(x-y) &= \frac{\tan x - \tan y}{1 + \tan x \tan y}\end{aligned}$$

Double-angle identities

$$\begin{aligned}\sin 2x &= 2 \sin x \cos x = \frac{2 \tan x}{1 + \tan^2 x} \\ \cos 2x &= \cos^2 x - \sin^2 x \\ \cos 2x &= 2 \cos^2 x - 1 \\ \cos 2x &= 1 - 2 \sin^2 x \\ \cos 2x &= \frac{1 - \tan^2 x}{1 + \tan^2 x} \\ \tan 2x &= \frac{2 \tan x}{1 - \tan^2 x} \\ \cot 2x &= \frac{\cot^2 x - 1}{2 \cot x}\end{aligned}$$

Half-angle identities

$$\begin{aligned}\sin \frac{x}{2} &= \pm \sqrt{\frac{1 - \cos x}{2}} \\ \cos \frac{x}{2} &= \pm \sqrt{\frac{1 + \cos x}{2}} \\ \tan \frac{x}{2} &= \pm \sqrt{\frac{1 - \cos x}{1 + \cos x}} = \frac{\sin x}{1 + \cos x} = \frac{1 - \cos x}{\sin x}\end{aligned}$$

Integral

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

Inverse Trigonometric integral

$$\begin{aligned}\int \frac{1}{1+x^2} \, dx &= \arctan(x) \\ \int \frac{1}{a+bx^2} \, dx &= -\frac{\sqrt{\frac{-1}{ab}} \ln\left(-a\sqrt{\frac{-1}{ab}} + x\right)}{2} + \frac{\sqrt{\frac{-1}{ab}} \ln\left(a\sqrt{\frac{-1}{ab}} + x\right)}{2}\end{aligned}$$

Inverse hyperbolic trigonometry integral

$$\begin{aligned}\int \frac{1}{\sqrt{1+x^2}} \, dx &= \sinh^{-1}(x) + C \\ \int \frac{1}{\sqrt{x^2-1}} \, dx &= \cosh^{-1}(x) + C \\ \int \frac{1}{1-x^2} \, dx &= \tanh^{-1}(x) + C, \quad |x| < 1 \\ \int \frac{1}{1-x^2} \, dx &= \coth^{-1}(x) + C, \quad |x| > 1 \\ \int \frac{-1}{x\sqrt{1-x^2}} \, dx &= \operatorname{sech}^{-1}(x) + C \\ \int \frac{-1}{|x|\sqrt{1+x^2}} \, dx &= \operatorname{csch}^{-1}(x) + C\end{aligned}$$

Note that the derivatives of $\tanh^{-1}(x)$ and $\coth^{-1}(x)$ are the same. Thus, when we compute $\int \frac{1}{1-x^2} \, dx$, we need to select the proper antiderivative based on the domain of the functions and the values of x .

$$\int \frac{1}{1-x^2} \, dx = \begin{cases} \tanh^{-1}(x) + C, & |x| < 1 \\ \coth^{-1}(x) + C, & |x| > 1 \end{cases}$$

Derivative

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

[SI*] Logarithm

Logarithm along with exponential play a very important role in solving differential equations, so we will need to take a look at these formulas often to help us when solving differential equations problem

$$\begin{aligned}\log_a b &= c \\ a^c &= b \\ \log b &= c \\ e^c &= b \\ \log_a a &= 1 \\ \log(a * b) &= \log(a) + \log(b) \\ \log\left(\frac{a}{b}\right) &= \log(a) - \log(b) \\ \log(a^b) &= b \log(a) \\ a^{\log_a b} &= b \\ \log_b a &= \frac{\log_c a}{\log_c b} \\ \log_b c &= \log_b a \log_a c \\ \log_{b^n} a^m &= \frac{m}{n} \log_b a \\ -\log_b a &= \log_b \left(\frac{1}{a}\right) \\ -\log_b a &= \log_{\frac{1}{b}} a \\ \frac{d}{dx} \log_a x &= \frac{1}{x \ln(a)} \\ \frac{d}{dx} \ln x &= \frac{1}{x}\end{aligned}$$

with $e = 2.718281828459045$ is the Euler' number. When there is no base written in \log function, we will use e as the base number. The base number cannot eb equal to 1 nad it has to be positive. For all $\log(a)$, a is a real number and has to be positive.

VIII. VECTOR CALCULUS

• Vector Fields with Div, Grad, Curl

[SI*] Vector calculus talks about vector-valued functions, that is, functions whose input is a real number and whose output is a vector.

In vector calculus we will use the operator ∇ , read: del, or nabla, it is an operator used in mathematics as a vector differential operator. When applied to a function defined on a one-dimensional domain, it denotes the standard derivative of the function as defined in calculus.

When applied to a field (a function defined on a multi-dimensional domain), it may denote any one of three operations depending on the way it is applied: the gradient or steepest slope of a scalar field; the divergence of a vector field; or the curl (rotation) of a vector field.

Del is a very convenient mathematical notation for those three operations (gradient, divergent, and curl) that makes many equations easier to write and remember. The del symbol can be formally defined as a vector operator whose components are the corresponding partial derivative operators. As a vector operator, it can act on scalar and vector fields in three different ways, giving rise to three different differential operations:

1. It acts on a scalar field ($f(x, y, z)$) by a formal scalar multiplication-to produce a vector field (∇f) called the gradient.
2. It acts on a vector field ($\mathbf{F}(x, y, z)$) by a formal dot product-to produce a scalar field ($\nabla \cdot \mathbf{F}$) called the divergence.
3. It acts on a vector field ($\mathbf{F}(x, y, z)$) by a formal cross product-to produce a vector field ($\nabla \times \mathbf{F}$) called the curl.

These three uses are summarized as:

Gradient

$$\text{grad } f = \nabla f \quad (2.17)$$

Physical interpretation:

Points in the direction of the greatest rate of increase of the scalar field, and its magnitude is that rate of increase.

Example:

Consider a scalar field representing temperature $T(x, y, z)$ in a garden. The gradient ∇T would give a vector at each point indicating the direction in which the temperature increases most rapidly, and its magnitude would be the rate of that increase.

Divergent

$$\text{div } \mathbf{v} = \nabla \cdot \mathbf{v} \quad (2.18)$$

Physical interpretation:

It measures the "outward flux per unit volume" of a vector field at a point. It quantifies how much a vector field is "spreading out" or "compressing" at that point.

Example:

For a vector field representing fluid flow $\mathbf{v}(x, y, z)$, the divergence $\nabla \cdot \mathbf{v}$ indicates

whether the fluid is expanding (positive divergence, like a source) or contracting (negative divergence, like a sink). If $\nabla \cdot v = 0$, the fluid is incompressible.

Curl

$$\operatorname{curl} v = \nabla \times v \quad (2.19)$$

Physical interpretation:

It measures the "rotation" or "circulation" of a vector field at a point. The direction of the curl vector indicates the axis of rotation, and its magnitude indicates the strength of the rotation.

Example:

For a vector field representing the velocity of a fluid $v(x, y, z)$, the curl $\nabla \times v$ indicates the local angular velocity of the fluid. If $\nabla \times v = \mathbf{0}$, the fluid is irrotational.

The divergence of a curl is zero

$$\nabla \cdot (\nabla \times F) = 0 \quad (2.20)$$

This means that the divergence of any curl field is always zero, in linear algebra it is a dot product of an orthogonal vector.

Curl of the gradient is zero

$$\nabla \times (\nabla f) = \mathbf{0} \quad (2.21)$$

This means that the curl of any gradient field (which is a conservative vector field) is always the zero vector.

The div, grad, curl operators are crucial in various fields, including fluid dynamics, electromagnetism (Maxwell's equations), and heat transfer.

Definition 2.1: Mathematical Vector Fields

A vector field in 2-dimensional space is function whose value at a point (x, y) is a 2-dimensional vector $F(x, y)$. Similarly, in 3-dimensional space, a vector field is a function $F(x, y, z)$ whose value at the point (x, y, z) is a 3-dimensional vector.

If $F(x, y, z)$ is a vector, it has i, j , and k components. Each of these components is a scalar function of the point (x, y, z) , so we will often write

$$F(x, y, z) = F_1(x, y, z)\mathbf{i} + F_2(x, y, z)\mathbf{j} + F_3(x, y, z)\mathbf{k} = \langle F_1, F_2, F_3 \rangle \quad (2.22)$$

For example, if $F(x, y, z) = \langle x^2, xy \tan(3z), \ln(y^2) \rangle$, then

$$\begin{aligned} F_1(x, y, z) &= x^2 \\ F_2(x, y, z) &= xy \tan(3z) \\ F_3(x, y, z) &= \ln(y^2) \end{aligned}$$

* The writing of i, j, k components in this book has the same meaning as $\hat{i}, \hat{j}, \hat{k}$ or $\hat{x}, \hat{y}, \hat{z}$ and will be used interchangiblly.

[SI*] In three-dimensional Cartesian coordinate system \mathbb{R}^3 with coordinates (x, y, z) and standard basis or unit vectors of axes $\{\mathbf{e}_x, \mathbf{e}_y, \mathbf{e}_z\}$, del is written as

$$\nabla = \mathbf{e}_x \frac{\partial}{\partial x} + \mathbf{e}_y \frac{\partial}{\partial y} + \mathbf{e}_z \frac{\partial}{\partial z} = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z} \right) \quad (2.23)$$

As a vector operator, del naturally acts on scalar fields via scalar multiplication, and naturally acts on vector fields via dot products and cross products.

Then using the above definition of ∇ , we may write

$$\nabla f = \left(\mathbf{e}_x \frac{\partial}{\partial x} \right) f + \left(\mathbf{e}_y \frac{\partial}{\partial y} \right) f + \left(\mathbf{e}_z \frac{\partial}{\partial z} \right) f = \frac{\partial f}{\partial x} \mathbf{e}_x + \frac{\partial f}{\partial y} \mathbf{e}_y + \frac{\partial f}{\partial z} \mathbf{e}_z \quad (2.24)$$

and

$$\nabla \cdot \mathbf{F} = \left(\mathbf{e}_x \frac{\partial}{\partial x} \cdot \mathbf{F} \right) + \left(\mathbf{e}_y \frac{\partial}{\partial y} \cdot \mathbf{F} \right) + \left(\mathbf{e}_z \frac{\partial}{\partial z} \cdot \mathbf{F} \right) = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y} + \frac{\partial F_z}{\partial z} \quad (2.25)$$

and

$$\begin{aligned} \nabla \times \mathbf{F} &= \left(\mathbf{e}_x \frac{\partial}{\partial x} \times \mathbf{F} \right) + \left(\mathbf{e}_y \frac{\partial}{\partial y} \times \mathbf{F} \right) + \left(\mathbf{e}_z \frac{\partial}{\partial z} \times \mathbf{F} \right) \\ &= \frac{\partial}{\partial x}(0, -F_z, F_y) + \frac{\partial}{\partial y}(F_z, 0, -F_x) + \frac{\partial}{\partial z}(-F_y, F_x, 0) \\ &= \left(\frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z} \right) \mathbf{e}_x + \left(\frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x} \right) \mathbf{e}_y + \left(\frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \right) \mathbf{e}_z \end{aligned} \quad (2.26)$$

Del can also be expressed in other coordinate systems, e.g. in cylindrical and spherical coordinates.

[SI*] **Gradient**

the vector derivative of a scalar field f is called the gradient, and it can be represented as

$$\text{grad } f = \nabla f = \frac{\partial f}{\partial x} \hat{x} + \frac{\partial f}{\partial y} \hat{y} + \frac{\partial f}{\partial z} \hat{z}$$

It always points in the direction of greatest increase of f , and it has a magnitude equal to the maximum rate of increase at the point-just like a standard derivative.

Theorem 2.5: Gradient Identities

- (a) $\nabla(f + g) = \nabla f + \nabla g$
- (b) $\nabla(cf) = c\nabla f$, for any constant c .
- (c) $\nabla(fg) = (\nabla f)g + f(\nabla g)$
- (d) $\nabla\left(\frac{f}{g}\right) = \frac{(g\nabla f - f\nabla g)}{g^2}$ at points x where $g(x) \neq 0$.
- (e) $\nabla(\mathbf{F} \cdot \mathbf{G}) = \mathbf{F} \times (\nabla \times \mathbf{G}) - (\nabla \times \mathbf{F}) \times \mathbf{G} + (\mathbf{G} \cdot \nabla)\mathbf{F} + (\mathbf{F} \cdot \nabla)\mathbf{G}$
Here

$$(\mathbf{G} \cdot \nabla)\mathbf{F} = G_1 \frac{\partial \mathbf{F}}{\partial x} + G_2 \frac{\partial \mathbf{F}}{\partial y} + G_3 \frac{\partial \mathbf{F}}{\partial z}$$

[SI*] Divergence

The divergence of a vector field $\mathbf{v}(x, y, z) = v_x \hat{\mathbf{x}} + v_y \hat{\mathbf{y}} + v_z \hat{\mathbf{z}}$ is a scalar field that can be represented as:

$$\operatorname{div} \mathbf{v} = \nabla \cdot \mathbf{v} = \frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}$$

The divergence is roughly a measure of a vector field's increase in the direction it points; but more accurately, it is a measure of that field's tendency to converge toward or diverge from a point.

Theorem 2.6: Divergent Identities

- (a) $\nabla \cdot (\mathbf{F} + \mathbf{G}) = \nabla \cdot \mathbf{F} + \nabla \cdot \mathbf{G}$
- (b) $\nabla \cdot (c\mathbf{F}) = c\nabla \cdot \mathbf{F}$, for any constant c .
- (c) $\nabla \cdot (f\mathbf{F}) = (\nabla f) \cdot \mathbf{F} + f\nabla \cdot \mathbf{F}$
- (d) $\nabla \cdot (\mathbf{F} \times \mathbf{G}) = (\nabla \times \mathbf{F}) \cdot \mathbf{G} - \mathbf{F} \cdot (\nabla \times \mathbf{G})$

[SI*] Curl

The curl of a vector field $\mathbf{v}(x, y, z) = v_x \hat{\mathbf{x}} + v_y \hat{\mathbf{y}} + v_z \hat{\mathbf{z}}$ is a vector function that can be represented as:

$$\operatorname{curl} \mathbf{v} = \nabla \times \mathbf{v} = \left(\frac{\partial v_z}{\partial y} - \frac{\partial v_y}{\partial z} \right) \hat{\mathbf{x}} + \left(\frac{\partial v_x}{\partial z} - \frac{\partial v_z}{\partial x} \right) \hat{\mathbf{y}} + \left(\frac{\partial v_y}{\partial x} - \frac{\partial v_x}{\partial y} \right) \hat{\mathbf{z}}$$

The curl at a point is proportional to the on-axis torque that a tiny pinwheel would be subjected to if it were centered at that point.

Theorem 2.7: Curl Identities

- (a) $\nabla \times (\mathbf{F} + \mathbf{G}) = \nabla \times \mathbf{F} + \nabla \times \mathbf{G}$
- (b) $\nabla \times (c\mathbf{F}) = c\nabla \times \mathbf{F}$, for any constant c .
- (c) $\nabla \times (f\mathbf{F}) = (\nabla f) \times \mathbf{F} + f\nabla \times \mathbf{F}$
- (d) $\nabla \times (\mathbf{F} \times \mathbf{G}) = \mathbf{F}(\nabla \cdot \mathbf{G}) - (\nabla \cdot \mathbf{F})\mathbf{G} + (\mathbf{G} \cdot \nabla)\mathbf{F} - (\mathbf{F} \cdot \nabla)\mathbf{G}$

Here

$$(\mathbf{G} \cdot \nabla)\mathbf{F} = G_1 \frac{\partial \mathbf{F}}{\partial x} + G_2 \frac{\partial \mathbf{F}}{\partial y} + G_3 \frac{\partial \mathbf{F}}{\partial z}$$

[SI*] In SymIntegration the computation for div, grad, and curl can be handled in `src/divgradcurl.cpp`.

```
#include "symintegral/symintegrationc++.h"

#ifndef SYMBOLIC_DEFINE
#ifndef SYMINTEGRATION_CPLUSPLUS_DIVGRADCURL_DEFINE
#define SYMINTEGRATION_CPLUSPLUS_DIVGRADCURL_DEFINE
```

```

Symbolic div(const Symbolic &F, const Symbolic &x, const Symbolic &y,
            const Symbolic &z)
{
    Symbolic sol, i("i"), j("j"), k("k");

    Symbolic F1 = F.coeff(i,1);
    Symbolic F2 = F.coeff(j,1);
    Symbolic F3 = F.coeff(k,1);

    if(F1 != 0 || F2 != 0 || F3 != 0)
    {
        sol = df(F1,x) + df(F2,y) + df(F3,z);
    }
    else if(F1_hat != 0 || F2_hat != 0 || F3_hat != 0)
    {
        sol = df(F1_hat,x) + df(F2_hat,y) + df(F3_hat,z);
    }
    return sol;
}

Symbolic grad(const Symbolic &F, const Symbolic &x, const Symbolic &y,
              const Symbolic &z)
{
    Symbolic sol;

    if(F != 0)
    {
        sol = (df(F,x), df(F,y), df(F,z));
        sol = sol.transpose();
    }
    return sol;
}

...

Symbolic curl(const Symbolic &F, const Symbolic &x, const Symbolic &y,
              const Symbolic &z)
{
    Symbolic sol, i("i"), j("j"), k("k");

    Symbolic F1 = F.coeff(i,1);
    Symbolic F2 = F.coeff(j,1);
    Symbolic F3 = F.coeff(k,1);

    if(F1 != 0 || F2 != 0 || F3 != 0)
    {
        sol = (df(F3,y) - df(F2,z), -(df(F3,x) - df(F1,z)), df(F2)
}

```

```

        ,x) - df(F1,y));
    sol = sol.transpose();
}
else if(F1_hat != 0 || F2_hat != 0 || F3_hat != 0)
{
    sol = (df(F3_hat,y) - df(F2_hat,z), -(df(F3_hat,x) - df(
        F1_hat,z)), df(F2_hat,x) - df(F1_hat,y));
    sol = sol.transpose();
}
return sol;
}

```

Code 28: *src/divgradcurl.cpp*

the source code can handle the input of a function along with the independent variables.
For example, if we have a scalar field:

$$f(x, y, z) = -\frac{1}{2}xy + 2y + z^2$$

and we can compute the gradient with :

grad(f,x,y,z)

If we have a vector field:

$$\mathbf{F}(x, y, z) = y\hat{i} - x\hat{j} + yz\hat{k}$$

we can compute the divergence and curl with these functions:

div(F,x,y,z)

curl(F,x,y,z)

In SymIntegration the source code to compute the div, grad, curl for the example above will be like this:

```

#include <iostream>
#include "symintegrationc++.h"
#include <bits/stdc++.h>
#include <cmath>

using namespace std;
using namespace SymbolicConstant;

int main(void)
{
    Symbolic x("x"), y("y"), z("z"), f, f2, i("i"), j("j"), k("k");

    f = -(0.5*x)*y+2*y + z*z;
    f2 = y*i -x*j +y*z*k;
    Symbolic f2_hat = y*i -x*j +y*z*k;
    cout << "F(x,y,z) = " << f << endl;
    cout << "F2 (vector field) = " << f2_hat << endl;
}

```

```

    cout << "\ndiv(F2) = " << div(f2_hat,x,y,z) << endl;
    cout << "\ngrad(F) = " << grad(f,x,y,z) << endl;
    cout << "\ncurl(F2) = " << curl(f2_hat,x,y,z) << endl;

    return 0;
}

```

Code 29: Examples/test SymIntegration DivGradCurl/main.cpp

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DivGradCurl ]# m
ake
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lsyminintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DivGradCurl ]# ./main
F(x,y,z) = -0.5*x*y+2*y+z^(2)
F2 (vector field) = y*i-x*j+y*z*k

div(F2) = y

grad(F) =
[ -0.5*y ]
[ -0.5*x+2 ]
[ 2*z ]

curl(F2) =
[ z ]
[ 0 ]
[ -2 ]

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DivGradCurl ]# []

```

Figure 2.5: The computation of div grad curl . (SymIntegration/Examples/Test SymIntegration DivGradCurl-/main.cpp).

[SI*] The input, f , for the gradient is a scalar-valued function, while the output, ∇f , is a vector-valued function.

The input, F , for the divergence is a vector-valued function, while the output, $\nabla \cdot F$, is a scalar-valued function.

The input, F , for the curl is a vector-valued function, and the output, $\nabla \times F$, is a vector-valued function.

[SI*] If F denotes the velocity field for a fluid, then $\text{div } F$ at a point p measures the tendency of that fluid to diverge away from p ($\text{div } F > 0$) or accumulate toward p ($\text{div } F < 0$).

On the other hand, $\text{curl } F$ picks out the direction of the axis about which the fluid rotates (curls) most rapidly, and $\|\text{curl } F\|$ is a measure of the speed of this rotation. The direction of the rotation is according to the right-hand rule.

[SI*] **Laplacian**

The scalar function $\text{div}(\text{grad } f) = \nabla \cdot \nabla f$ (also written $\nabla^2 f$) is called the Laplacian, and a function f satisfying

$$\nabla^2 f = 0$$

is said to be harmonic, this concept is important in physics.

The Laplace operator is a scalar operator that can be applied to either vector or scalar fields.

The Laplacian of a scalar field $f(x, y, z)$ is the scalar-valued function:

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \quad (2.27)$$

The Laplacian of a vector field $\mathbf{F}(x, y, z)$ is the vector field:

$$\Delta \mathbf{F} = \nabla^2 \mathbf{F} = \nabla \cdot \nabla \mathbf{F} = \frac{\partial^2 \mathbf{F}}{\partial x^2} + \frac{\partial^2 \mathbf{F}}{\partial y^2} + \frac{\partial^2 \mathbf{F}}{\partial z^2} \quad (2.28)$$

so if we have a vector field that can be written like this:

$$\mathbf{F}(x, y, z) = F_1(x, y, z)\hat{i} + F_2(x, y, z)\hat{j} + F_3(x, y, z)\hat{k}$$

then the Laplacian can also be written like this:

$$\begin{aligned} \nabla^2 \mathbf{F} &= \nabla^2 F_1 \hat{i} + \nabla^2 F_2 \hat{j} + \nabla^2 F_3 \hat{k} \\ \nabla^2 \mathbf{F} &= (\nabla^2 F_1, \nabla^2 F_2, \nabla^2 F_3) \\ \nabla^2 \mathbf{F} &= \left(\frac{\partial^2 F_1}{\partial x^2} + \frac{\partial^2 F_1}{\partial y^2} + \frac{\partial^2 F_1}{\partial z^2} \right) \hat{i} + \left(\frac{\partial^2 F_2}{\partial x^2} + \frac{\partial^2 F_2}{\partial y^2} + \frac{\partial^2 F_2}{\partial z^2} \right) \hat{j} \\ &\quad + \left(\frac{\partial^2 F_3}{\partial x^2} + \frac{\partial^2 F_3}{\partial y^2} + \frac{\partial^2 F_3}{\partial z^2} \right) \hat{k} \\ \nabla^2 \mathbf{F} &= \left(\frac{\partial^2 F_1}{\partial x^2} + \frac{\partial^2 F_1}{\partial y^2} + \frac{\partial^2 F_1}{\partial z^2}, \frac{\partial^2 F_2}{\partial x^2} + \frac{\partial^2 F_2}{\partial y^2} + \frac{\partial^2 F_2}{\partial z^2}, \frac{\partial^2 F_3}{\partial x^2} + \frac{\partial^2 F_3}{\partial y^2} + \frac{\partial^2 F_3}{\partial z^2} \right) \end{aligned} \quad (2.29)$$

The Laplacian of the velocity is related to the viscous forces in a fluid, it is also part of the heat equation, which describes how temperature changes over time, the Laplacian is also used in Poisson's equation, which relates electric potential to charge density.

In essence, the Laplacian of a vector field provides valuable information about the "smoothness" or "curvature" of the vector field, highlighting regions where the field changes rapidly.

The Laplacian is ubiquitous throughout modern mathematical physics, appearing for example in Laplace's equation, Poisson's equation, the heat equation, the wave equation, and the Schrodinger equation.

To compute the Laplacian from a defined function $f(x, y, z)$ with SymIntegration we can use :

laplacian(f,x,y,z) (it can handle the input of scalar field or vector field)

Theorem 2.8: Laplacian Identities

- (a) $\nabla^2(f + g) = \nabla^2 f + \nabla^2 g$
- (b) $\nabla^2(cf) = c\nabla^2 f$, for any constant c .
- (c) $\nabla^2(fg) = f\nabla^2 g + 2\nabla f \cdot \nabla g + g\nabla^2 f$

Theorem 2.9: Degree Two Identities

- (a) $\nabla \cdot (\nabla \times \mathbf{F}) = 0$ (divergence of a curl)
- (b) $\nabla \times (\nabla f) = 0$ (curl of gradient)
- (c) $\nabla \cdot (f\{\nabla g \times \nabla h\}) = \nabla f \cdot (\nabla g \times \nabla h)$
- (d) $\nabla \cdot (f\nabla g - g\nabla f) = f\nabla^2 g - g\nabla^2 f$
- (e) $\nabla \times (\nabla \times \mathbf{F}) = \nabla(\nabla \cdot \mathbf{F}) - \nabla^2 \mathbf{F}$ (curl of curl)

[SI*] Hessian Matrix

Hessian matrix is a matrix that maps a function of n variables into $n \times n$ square matrix. Consider a function

$$f(x_1, x_2, \dots, x_n)$$

the Hessian matrix of f is the matrix-valued function $H : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ defined by

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix} \quad (2.30)$$

we can generate the hessian matrix from a defined function f with SymIntegration with :

hessianmatrix(f,x,y) (for 2 variables function)
hessianmatrix(f,x,y,z) (for 3 variables function)

Below is the code to show hessian matrix for function of 2 variables, $f(x, y)$, we provide the function to create the Hessian matrix for function that has up to 3 variables for now, and it is handled in `src/divgradcurl.cpp`.

```
#include <iostream>
#include <iomanip> // to declare the manipulator of setprecision()
#include <fstream>
#include <bits/stdc++.h> //for setw(6) at display() function
#include <vector> // For std::vector (example container)
#include "symintegrationc++.h"
#include <algorithm> // For std::sort

#include <chrono>
#include <string>
#include <bitset>

using namespace std::chrono;
using namespace std;

int main(int argc, char** argv)
```

```
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    Symbolic x("x"), y("y");

    Symbolic f = 2*sin(3*x) + 5 + x*x*y + cos(y);
    cout << "\nf(x,y) = " << f << endl;
    cout << "\nH = " << hessian(f,x,y) << endl;

    // Get ending timepoint
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "Time taken by function: " << duration.count() << "
        microseconds" << endl;

    return 0;
}
```

Code 30: Examples/test Hessian Matrix/main.cpp

```
root [ ~/SourceCodes/CPP/C++ Create Library/TEST Hessian ]# ./main
f(x,y) = 2*sin(3*x)+x^(2)*y+cos(y)+5

H =
[-18*sin(3*x)+2*y      2*x      ]
[      2*x      -cos(y)      ]

Time taken by function: 32521 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/TEST Hessian ]# make
g++   -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/TEST Hessian ]# ./main
f(x,y) = 2*sin(3*x)+x^(2)*y+cos(y)+5

H =
[-18*sin(3*x)+2*y      2*x      ]
[      2*x      -cos(y)      ]

Time taken by function: 31283 microseconds
```

Figure 2.6: The computation of Hessian matrix. (SymIntegration/Examples/Test Hessian Matrix/main.cpp).

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)**

Differential equations are of interest to engineers and other nonmathematicians, such as physicists. It is because of the possibility of using them to investigate a wide variety of problems in the physical, biological, and social sciences.

For example consider this Newton's law formula:

$$F = ma \quad (3.1)$$

If you read the Physics book dor undergraduate degree you will learn that it is not simply just $F = ma$, but it has a derivative as well

$$F = m \frac{dv}{dt} \quad (3.2)$$

Then add with drag force the equation will become

$$m \frac{dv}{dt} = mg - \gamma v \quad (3.3)$$

The Eq. (3.3) is what we call a differential equation that model an object falling in the atmosphere near sea level. From this equation we are able to determine the velocity at a given time, $v(t)$, from known g, m, γ . You can read more on this book [2], it is the first example of the book.

We will assume that you already read some books or learn from anywhere about differential equations, so we will ony focus on the SymIntegration function and capability that can help you

to compute the solution of a differential equation.

Starting on June 23rd, 2025, we have created a new function **dsolve** in SymIntegration, it is still very fresh and in infant stage and just able to solve a very simple first order linear ordinary differential equations.

I. SOLVING FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS

- Method of Integrating Factors

[SI*] Solve the initial value problem

$$2y' + ty = 2 \quad (3.4)$$

with the initial value condition

$$y(0) = 1 \quad (3.5)$$

Solution:

We will convert the differential equation Eq. (3.4) into the standard form to become

$$y' + \left(\frac{t}{2}\right)y = 1 \quad (3.6)$$

Thus $p(t) = t/2$, and the integrating factor is

$$\mu(t) = e^{\int p(t) dt} = e^{\frac{t^2}{4}}$$

Then multiply Eq. (3.6) by $\mu(t)$ so that

$$e^{\frac{t^2}{4}}y' + \left(\frac{t}{2}\right)e^{\frac{t^2}{4}}y = e^{\frac{t^2}{4}} \quad (3.7)$$

Then integrate both sides of equation we will obtain

$$e^{\frac{t^2}{4}}y = \int e^{\frac{t^2}{4}} dt + c \quad (3.8)$$

thus

$$y = e^{-\frac{t^2}{4}} \int_0^t e^{\frac{s^2}{4}} ds + ce^{-\frac{t^2}{4}} \quad (3.9)$$

You can compute the value of the constant c by inputting the initial condition of $y(0) = 1$.

Now, in SymIntegration we will need the user to input the equation in term of $\frac{dy}{dx} = f(x, y)$, we will input the $f(x, y)$ and the coefficient of $\frac{dy}{dx}$ has to be 1.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# ./main
f(x) = -0.5*x*y+1
DSolve for f(x,y).

y(x) = 1/2*pi^(1/2)*(erf(x*(-0.25)^(1/2))*(-0.25)^(-1/2)*e^(-0.25*x^(2))+C*e^(-0.25*x^(2))
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# 
```

Figure 3.1: The newly built function `dsolve` in SymIntegration is able to compute the solution $y(x)$ for the ordinary differential equation of $2y' + ty = 2$ (`SymIntegration/Examples/Test SymIntegration DSolve/main.cpp`).

[SI*] For the coding in SymIntegration, we add a new C++ file:

`src/dsolve.cpp`

This is the very first beginning of the `dsolve` by June 23rd, 2025:

```
#include "symintegral/symintegrationc++.h"

#ifndef SYMBOLIC_DEFINE
#ifndef SYMINTEGRATION_CPLUSPLUS_DSOLVE_DEFINE
#define SYMINTEGRATION_CPLUSPLUS_DSOLVE_DEFINE

Symbolic dsolve(const Symbolic &fx, const Symbolic &y, const
Symbolic &x)
{
    Symbolic dsol, mu, C("C");

    if(fx != 0)
    {
        list<Equations> eq;
        list<Equations>::iterator i;
        UniqueSymbol a, b, c, d;
        eq = (a*x*y + d).match(fx, (a,d));
        for(i=eq.begin(); i!=eq.end(); ++i)
        {
            try {
                Symbolic ap = rhs(*i, a), dp = rhs(*i,
d);
                mu = exp(integrate(-ap*x,x));
                dsol = (integrate(mu*dp,x))/(mu) + (C)
/(mu);
            } catch(const SymbolicError &se) {}
        }
    }
    return dsol;
}

#endif
#endif

```

Code 31: *src/dsolve.cpp*

and the include file as well **include/symintegral/dsolve.h**

```
#ifndef SYMINTEGRATION_CPLUSPLUS_DSOLVE

#ifndef SYMBOLIC_FORWARD
#ifndef SYMINTEGRATION_CPLUSPLUS_DSOLVE_FORWARD
#define SYMINTEGRATION_CPLUSPLUS_DSOLVE_FORWARD

#endif
#endif

#ifndef SYMBOLIC_DECLARE

```

```
#define SYMINTEGRATION_CPLUSPLUS_DSOLVE
#ifndef SYMINTEGRATION_CPLUSPLUS_DSOLVE_DECLARE
#define SYMINTEGRATION_CPLUSPLUS_DSOLVE_DECLARE

Symbolic dsolve(const Symbolic &, const Symbolic &, const Symbolic
&);

#endif
#endif

#endif
```

Code 32: *include/symintegral/dsolve.h*

Then we also need to add a new function **erf**, as the integral of e^{x^2} is not an easy one to be computed and defined, it is like this

$$\int e^{ax^2} dx = \frac{\sqrt{\pi} \operatorname{erf}(x\sqrt{-a})}{2\sqrt{-a}}, \quad a \neq 0 \quad (3.10)$$

To handle that, the need to return the $\int e^{ax^2} dx$ correctly in SymIntegration we modify these files:

- *src/symintegrationc++.cpp*

```
...
...
Symbolic erf(const Symbolic &s)
{ return Erf(s); }
```

Code 33: *src/symintegrationc++.cpp*

- *src/functions.cpp*

```
...
...
///////////////////////////////
// Implementation of Erf //
/////////////////////////////

Erf::Erf(const Erf &s) : Symbol(s) {}

Erf::Erf(const Symbolic &s) : Symbol(Symbol("erf")[s]) {}

Simplified Erf::simplify() const
```

```
{  
    const Symbolic &s = parameters.front().simplify();  
    if(s == 0) return Number<int>(0);  
    return *this;  
}  
  
Symbolic Erf::df(const Symbolic &s) const  
{ return Derivative(*this,s); }  
  
Symbolic Erf::integrate(const Symbolic &s) const  
{  
    const Symbolic &x = parameters.front();  
    if(x == s) return Integral(*this,s);  
    if(df(s) == 0) return Integral(*this,s);  
    return Integral(*this,s);  
}  
  
...  
...
```

Code 34: *src/functions.cpp*

- include/symintegral/functions.h
-

```
...  
...  
  
class Erf;  
  
...  
...  
  
class Erf: public Symbol  
{  
public: Erf(const Erf&);  
Erf(const Symbolic&);  
  
Simplified simplify() const;  
Symbolic df(const Symbolic&) const;  
Symbolic integrate(const Symbolic&) const;  
  
Cloning *clone() const { return Cloning::clone(*this);  
}  
};  
  
...  
...
```

Code 35: *include/symintegral/functions.h*

- include/symintegral/symintegrationc++.h

```
...  
...  
  
Symbolic erf(const Symbolic &);  
  
...
```

Code 36: *include/symintegral/symintegrationc++.h*

- **Separable Equations**

[SI*] We can integrate a differential equation function that is classified as separable equations if they have this form:

$$\frac{dy}{dx} = \frac{f(x)}{g(y)} \quad (3.11)$$

with $f(x)$ is a function of x only and $g(y)$ is a function of y only. Thus we can write the differential equation in the form of

$$g(y) dy - f(x) dx = 0 \quad (3.12)$$

Such an equation is said to be separable. A separable equation can be solved by integrating the functions f and g .

[SI*] **Example:**

Show that the equation

$$\frac{dy}{dx} = \frac{x^2}{1-y^2}$$

is separable, and then find an equation for its integral curves.

Solution:

$$\begin{aligned} \frac{dy}{dx} &= \frac{x^2}{1-y^2} \\ 1-y^2 \, dy &= x^2 \, dx \\ y - \frac{y^3}{3} &= \frac{x^3}{3} + c \\ -x^3 + 3y - y^3 &= c \end{aligned}$$

where c is an arbitrary constant and $-x^3 + 3y - y^3 = c$ is the equation for the integral curves in the form of implicit function.

[SI*] **Homogeneous Ratio Equations**

If the right side of the equation

$$\frac{dy}{dx} = f(x, y)$$

can be expressed as a function of the ratio y/x only, then the equation is said to be a homogeneous ratio equation. Such equations can always be transformed into separable equations by a change of the dependent variable.

[SI*] **Example:**

Find the solution of

$$\frac{dy}{dx} = \frac{y - 4x}{x - y}$$

Solution:

First we will make it into the form of y/x

$$\frac{dy}{dx} = \frac{y - 4x}{x - y} = \frac{(y/x) - 4}{1 - (y/x)}$$

Make the change of variables

$$v = \frac{y}{x}$$

$$\frac{dv}{dx} = \frac{1}{x} \frac{dy}{dx} - \frac{y}{x^2}$$

$$y = xv$$

$$\frac{dy}{dx} = x \frac{dv}{dx} + v = x \frac{dv}{dx} + v$$

As a result, the ODE becomes

$$\begin{aligned} \frac{dy}{dx} &= \frac{(y/x) - 4}{1 - (y/x)} \\ x \frac{dv}{dx} + v &= \frac{v - 4}{1 - v} \\ x \frac{dv}{dx} &= \frac{v - 4}{1 - v} - v \\ &= \frac{v - 4}{1 - v} - \frac{v(1 - v)}{1 - v} \\ &= \frac{v^2 - 4}{1 - v} \\ \frac{1 - v}{v^2 - 4} dv &= \frac{dx}{x} \\ \int \frac{1 - v}{(v + 2)(v - 2)} dv &= \int \frac{dx}{x} \\ \int \left(-\frac{3}{4(v + 2)} - \frac{1}{4(v - 2)} \right) dv &= \int \frac{dx}{x} \\ -\frac{3}{4} \ln |v + 2| - \frac{1}{4} \ln |v - 2| &= \ln |x| + C \\ -\frac{3}{4} \ln \left| \frac{y}{x} + 2 \right| - \frac{1}{4} \ln \left| \frac{y}{x} - 2 \right| &= \ln |x| + C \end{aligned}$$

We can write the solution as an implicit function like this

$$f(x, y) = -\frac{3}{4} \ln \left| \frac{y}{x} + 2 \right| - \frac{1}{4} \ln \left| \frac{y}{x} - 2 \right| - \ln |x| - C$$

Remember that C is a constant that depends on the initial value.

The C++ code to solve the homogeneous ratio equations is using a function named **dsolveseparable** and we need to input it like this

dsolveseparable(g(x,y),f(x,y),y,x)

with

$$\frac{dy}{dx} = \frac{f(x,y)}{g(x,y)}$$

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# ./main

DSolve for dy/dx = (y-4x) / (x-y)

f(x,y) = -0.25*ln(y*x^(-1)-2)-0.75*ln(y*x^(-1)+2)-ln(x)-C
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# []
```

Figure 3.2: The newly built function **dsolveseparable** in SymIntegration is able to compute the solution $f(x,y)$ for homogeneous ratio equations (*SymIntegration/Examples/Test SymIntegration DSolve/main.cpp*).

Now, to draw a direction field and some integral curves, recall that the right hand side of

$$\frac{dy}{dx} = \frac{y-4x}{x-y}$$

depends only on the ratio $\frac{y}{x}$. This means that the integral curves have the same slope at all points on any given straight line through the origin, although the slope changes from one line to another. Therefore the direction field and the integral curves are symmetric with respect to the origin.

- **List of Solvable Linear ODEs with **dsolve****

We will list all the first order linear ordinary differential equations that can be solved with **dsolve** function in SymIntegration here, they are:

$$\begin{aligned} ay' + ty &= b \\ aty' + by &= ct^2 \\ ay' + by &= ce^{dt} \\ ay' + by &= ct + d \\ ay' + by &= c \\ ay' + by &= 0 \end{aligned}$$

with a, b, c, d are integers, e is reserved for exponential symbol, the solution will be $y(t)$. Remember when inputting to SymIntegration, you have to input the function $f(t,y)$ with this criteria

$$y' = f(t,y) \tag{3.13}$$

input the right hand side type of function in term of variables t and y only, and the coefficient of y' is 1.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# ./main
f(t,y) = -0.5*t*y+1

DSolve for 2y' + ty = 2
y(t) = 1/2*pi^(1/2)*(erf(t*(-0.25)^(1/2)))*(-0.25)^(-1/2)*e^(-0.25*t^(2))+C*e^(-0.25*t^(2))

DSolve for ty' + 2y = 4t^2
y(t) = t^(2)+C*t^(-2)

DSolve for y' + 0.5y = 0.5*exp(t/3)
y(t) = 0.6*e^(0.333333*t)+C*e^(-0.5*t)

DSolve for y' - 2y = 4-t
y(t) = 1/2*t+C*e^(2*t)-7/4

DSolve for Q' = r/4 - r0/100
Q(t) = C*e^(-1/100*t*r)+25

DSolve for S' = rs
S(t) = C*e^(t*r)
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve ]# []

```

Figure 3.3: The function **dsolve** in SymIntegration able to compute the solution $y(t)$ for several examples of linear first order ordinary differential equations (*SymIntegration/Examples/Test SymIntegration DSolve-main.cpp*).

- **List of Solvable Linear ODEs with **dsolveseparable****

We will list all the first order linear ordinary differential equations that can be solved with **dsolveseparable** function in SymIntegration here, they are:

$$\frac{dy}{dx} = \frac{f(x)}{g(y)}$$

$$\frac{dy}{dx} = \frac{f(x,y)}{g(x,y)}$$

We create another function besides **dsolve** to make it easier to separate certain type of differential equation, with this **dsolveseparable** we need to input the function in the numerator and denominator. It is able to compute the simple separable equations and also the homogeneous ratio equations (we use this term instead of only 'homogeneous equations' that is written from this book [2] to avoid clashing of name at the next section).

the origin :

 31. $\frac{dy}{dx} = \frac{x^2 + xy + y^2}{x^2}$	 32. $\frac{dy}{dx} = \frac{x^2 + 3y^2}{2xy}$
 33. $\frac{dy}{dx} = \frac{4y - 3x}{2x - y}$	 34. $\frac{dy}{dx} = \frac{4x + 3y}{2x + y}$
 35. $\frac{dy}{dx} = \frac{x + 3y}{x - y}$	 36. $(x^2 + 3xy + y^2)dx - x^2 dy = 0$
 37. $\frac{dy}{dx} = \frac{x^2 - 3y^2}{2xy}$	 38. $\frac{dy}{dx} = \frac{3y^2 - x^2}{2xy}$

Figure 3.4: The problems from book [2] with regards to the homogeneous ratio equation that can be solved with SymIntegration' **dsolveseparable**.

```

]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolveseparable
]# ./main

DSolveseparable for dy/dx = (x^2) / (1-y^2)
f(x,y) = -1/3*y^(3)+y-1/3*x^(3)-C
DSolveseparable for dy/dx = (y-4x) / (x-y)
f(x,y) = -0.25*ln(y*x^(-1)-2)-0.75*ln(y*x^(-1)+2)-ln(x)-C
DSolveseparable for dy/dx = (x^2+xy+y^2) / (x^2)
f(x,y) = atan(y*x^(-1))-ln(x)-C
DSolveseparable for dy/dx = (4y-3x) / (2x-y)
f(x,y) = 0.25*ln(y*x^(-1)-1)-1.25*ln(y*x^(-1)+3)-ln(x)-C
DSolveseparable for dy/dx = (4x+3y) / (2x+y)
f(x,y) = -0.333333*ln(y*x^(-1)+1)-0.666667*ln(y*x^(-1)+4)-ln(x)-C
DSolveseparable for dy/dx = (x+3y) / (x-y)
f(x,y) = -ln(y*x^(-1)+1)-2*(y*x^(-1)+1)^(-1)-ln(x)-C
DSolveseparable for dy/dx = (x^2+3xy+y^2) / (x^2)
f(x,y) = -(y*x^(-1)+1)^(-1)-ln(x)-C
DSolveseparable for dy/dx = (x^2 - 3y^2) / (2xy)
f(x,y) = -0.2*ln(-5*y^(2)*x^(-2)+1)-ln(x)-C
DSolveseparable for dy/dx = (3y^2 - x^2) / (2xy)
f(x,y) = ln(y^(2)*x^(-2)-1)-ln(x)-C

```

Figure 3.5: The solutions with SymIntegration' **dsolveseparable** of the problems from book [2] at the end of chapter 2.2.

- **List of Solvable Linear ODEs with dsolve logistic**

We will list all the first order linear ordinary differential equations that can be solved with **dsolve logistic** function in SymIntegration here, they are:

$$\frac{dy}{dt} = r \left(1 - \frac{y}{K}\right) y$$

$$\frac{dy}{dt} = -r \left(1 - \frac{y}{T}\right) y$$

with r, K, T as the parameters. We need to input the function in $\frac{dy}{dt}$ term. The function needs 7 input

dsolve logistic(y',y,y0, t, r, K, T)

We can define $y' = \frac{dy}{dt}$ as either $r \left(1 - \frac{y}{K}\right) y$ or $-r \left(1 - \frac{y}{T}\right) y$, it will return the general solution with initial value problem of $y_0 = y_0$.

The more difficult case occurs in the case for logistic growth with a threshold where the differential equation is

$$\frac{dy}{dt} = -r \left(1 - \frac{y}{K}\right) \left(1 - \frac{y}{T}\right) y$$

This is the case where we have to do integration of fraction polynomial of order 3. For a while, instead of solving the difficult differential equation we will break it into 2 cases, with **if.. else if..** conditional.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Logistic Equation ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Logistic Equation ]# ./main
For dy/dt = r*(1-(y/K))*y
y(t) = y0*K*(y0+K*e^(-r*t))-y0*e^(-r*t))^( -1)

For dy/dt = -r*(1-(y/T))*y
y(t) = y0*T*(y0+T*e^(r*t))-y0*e^(r*t))^( -1)

For dy/dt = -r*(1-(y/T))*(1-(y/K))*y
For T < y < K and y > K
y(t) = y0*K*(y0+K*e^(-r*t))-y0*e^(-r*t))^( -1)
For 0 < y < T,
y(t) =
y0*T*(y0+T*e^(r*t))-y0*e^(r*t))^( -1)

Time taken by function: 261099 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Logistic Equation ]# 

```

Figure 3.6: The function *dsolve logistic* in SymIntegration (*SymIntegration/Examples/Test SymIntegration Logistic Equation/main.cpp*).

- Modeling with First Order Equations

[SI*] Mathematical modeling and experiment or observation are both critically important and have somewhat complementary roles in scientific investigations. Mathematical models are validated by comparison of their predictions with experimental results.

- [SI*] Three identifiable steps that are always present in the process of mathematical modeling:
1. Construction of the Model.
 2. Analysis of the Model.
 3. Comparison with Experiment or Observation.

[SI*] **The Water Tank Mixing**

At time $t = 0$ a tank contains Q_0 lb of salt dissolved in 100 gal of water. Assume that water containing $\frac{1}{4}$ lb of salt/gal is entering the tank at a rate of r gal/min and that the well-stirred mixture is draining from the tank at the same rate. Set up the initial value problem that describes this flow process. Find the amount of salt $Q(t)$ in the tank at any time, and also find the limiting amount Q_L that is present after a very long time. If $r = 3$ and $Q_0 = 2Q_L$, find the time T after which the salt level is within 2% of Q_L . Also find the flow rate that is required if the value of T is not to exceed 45 min.

Solution:

We assume that salt is neither created nor destroyed in the tank. Therefore variations in the amount of salt are due solely to the flows in and out of the tank. More precisely, the rate of change of salt in the tank, $\frac{dQ}{dt}$, is equal to the rate at which salt is flowing in minus the rate at which it is flowing out. In symbols,

$$\frac{dQ}{dt} = \text{rate in} - \text{rate out} \quad (3.14)$$

The rate at which salt enters the tank is the concentration $\frac{1}{4}$ lb/gal times the flow rate r gal/min, or $(r/4)$ lb/min. To find the rate at which salt leaves the tank, we need to multiply the concentration of salt in the tank by the rate of outflow, r gal/min. Since the rates of flow in and out are equal, the volume of water in the tank remains constant at 100 gal, and since the mixture is "well-stirred," the concentration throughout the tank is

the same, namely, $\frac{Q(t)}{100}$ lb/gal.

Therefore the rate at which salt leaves the tank is $\frac{rQ(t)}{100}$ lb/min. Thus the differential equation governing this process is

$$\frac{dQ}{dt} = \frac{r}{4} - \frac{rQ}{100} \quad (3.15)$$

The initial condition is

$$Q(0) = Q_0 \quad (3.16)$$

Upon thinking about the problem physically, we might anticipate that eventually the mixture originally in the tank will be essentially replaced by the mixture flowing in, whose concentration is $\frac{1}{4}$ lb/gal. Consequently, we might expect that ultimately the amount of salt in the tank would be very close to 25 lb. We can also find the limiting amount $Q_L = 25$ by setting $\frac{dQ}{dt}$ equal to zero in Eq. (3.15).

$$\frac{dQ}{dt} = \text{rate in} - \text{rate out} \quad (3.17)$$

The rate at which salt enters the tank is the concentration $\frac{1}{4}$ lb/gal times the flow rate r gal/min, or $(r/4)$ lb/min. To find the rate at which salt leaves the tank, we need to multiply the concentration of salt in the tank by the rate of outflow, r gal/min. Since the rates of flow in and out are equal, the volume of water in the tank remains constant at 100 gal, and since the mixture is "well-stirred," the concentration throughout the tank is the same, namely, $\frac{Q(t)}{100}$ lb/gal.

Therefore the rate at which salt leaves the tank is $\frac{rQ(t)}{100}$ lb/min. Thus the differential equation governing this process is

$$\frac{dQ}{dt} = \frac{r}{4} - \frac{rQ}{100} \quad (3.18)$$

The initial condition is

$$Q(0) = Q_0 \quad (3.19)$$

Upon thinking about the problem physically, we might anticipate that eventually the mixture originally in the tank will be essentially replaced by the mixture flowing in, whose concentration is $\frac{1}{4}$ lb/gal. Consequently, we might expect that ultimately the amount of salt in the tank would be very close to 25 lb. We can also find the limiting amount $Q_L = 25$ by setting $\frac{dQ}{dt}$ equal to zero in Eq. (3.18) and solving the resulting algebraic equation for Q .

To solve the initial value problem at Eqs (3.15), (3.16) analytically, note that Eq. (3.15) is both linear and separable. Rewriting it in the standard form for a linear equation, we have

$$\frac{dQ}{dt} + \frac{rQ}{100} = \frac{r}{4} \quad (3.20)$$

Thus the integrating factor is $e^{rt/100}$ and the general solution is

$$Q(t) = 25 + ce^{-\frac{rt}{100}} \quad (3.21)$$

where c is an arbitrary constant. To satisfy the initial condition (3.16), we must choose $c = Q_0 - 25$. Therefore the solution of the initial value problem at Eqs (3.15), (3.16) is

$$Q(t) = 25 + (Q_0 - 25)e^{-\frac{rt}{100}} \quad (3.22)$$

or

$$Q(t) = 25(1 - e^{-\frac{rt}{100}}) + Q_0 e^{-\frac{rt}{100}} \quad (3.23)$$

You can see that

$$Q(t) \rightarrow 25$$

as $t \rightarrow \infty$, so the limiting value Q_L is 25, confirming our physical intuition.

Further, $Q(t)$ approaches the limit more rapidly as r increases. In interpreting the solution (3.23), note that the second term on the right side is the portion of the original salt that remains at time t , while the first term gives the amount of salt in the tank due to the action of the flow processes.

Now suppose that $r = 3$ and $Q_0 = 2Q_L = 50$, then Eq. (3.22) becomes

$$Q(t) = 25 + (50 - 25)e^{-0.03t} \quad (3.24)$$

Since 2% of 25 is 0.5, we wish to find the time T at which $Q(t)$ has the value 25.5. Substituting $t = T$ and $Q(t) = 25.5$ in Eq. (3.24) and solving for T , we obtain

$$\begin{aligned} Q(t) &= 25 + (Q_0 - 25)e^{-\frac{rt}{100}} \\ 25.5 &= 25 + (50 - 25)e^{-\frac{3t}{100}} \\ e^{0.03t} &= \frac{25}{0.5} \\ T &= \frac{\ln|50|}{0.03} \\ &= 130.4 \end{aligned}$$

T is in minutes. To determine r so that $T = 45$, return to Eq. (3.22), set $t = 45$, $Q_0 = 50$, $Q(t) = 25.5$, and solve for r . The result is

$$\begin{aligned} Q(t) &= 25 + (Q_0 - 25)e^{-\frac{rt}{100}} \\ 25.5 &= 25 + (50 - 25)e^{-\frac{45r}{100}} \\ 0.5 &= 25e^{-\frac{45r}{100}} \\ e^{\frac{45r}{100}} &= \frac{25}{0.5} \\ \frac{45r}{100} &= \ln|50| \\ r &= \frac{100}{45} \ln|50| \\ &\approx 8.69 \end{aligned}$$

r is in gal/min.

We would like to look into the metrics and measurement a bit further, since this is really important. We have the units of gal/min which can be converted into

$$1 \text{ gal/min} = 0.00006309 \text{ m}^3/\text{s} = 0.0037854 \text{ m}^3/\text{min} \quad (3.25)$$

Now, we try to do some observations and experimentations with the help of computer simulation, namely Hamzstlab Physics. We will fill an empty tank with a water, that is made by water particle, sphere like, but since it is a 2D computer simulation, we make a circle instead.

And as a representation for the water tank, that is having a cylinder shape, we take the cross section of a cylinder that is like a rectangle.

We observe with a circle of radius $r = 0.2$ and a mass of 0.15 as the water particle that is coming into the empty tank, it takes $t = 45 \text{ s}$ to fill the water tank to be full.

The volume of the water tank itself based on the computer simulation is

$$\begin{aligned} V &= \pi r^2 h \\ &= \pi(5)^2(10) \\ &= 785.39819 \end{aligned}$$

The metrics for V is m^3 . So we will have the rate of the entering water as

$$\frac{785.39819 \text{ m}^3}{45 \text{ s}} = 17.453293 \text{ m}^3/\text{s} \quad (3.26)$$

Knowing that $1 \text{ gal/min} = 0.00006309 \text{ m}^3/\text{s}$, we will have the rate of the incoming water as

$$rate_{r=0.2,m=0.15} = \frac{17.453293 \text{ m}^3/\text{s}}{0.00006309 \text{ m}^3/\text{s}} * (1 \text{ gal/min}) = 276,641.19512 \text{ gal/min} \quad (3.27)$$

We are doing a 6 simulations with Hamzstlab Physics and we gather this data:

sim	$\mathbf{r}(m)$	N	$\mathbf{v}_x(\text{m/s})$	$\mathbf{t}(\text{s})$
1	0.2	23437.5	19.5	45
2	0.2	23437.5	25	44.7
3	0.3	6944.44	19.5	20
4	0.3	6944.44	25	20.3
5	0.4	2929.6875	19.5	11.8
6	0.4	2929.6875	25	11.5

Table 3.1: The variable t denotes time of completion in seconds till the tank is full, v_x is the velocity of the water particle in m/s toward the x axis, while N is the number of water particle that will fit in a tank of volume $V = 785.4 \text{ m}^3$, derived from $\frac{V}{\frac{4}{3}\pi r^3}$.

From table 3.1 we can then use multivariable regression so we can interpolate and approximate the time of completion, t , for any speed of the water that is filling the tank, for any radius of the water particle and the volume of the water tank. Thus, we will treat t as the independent variable, and the rest is the dependent variables.

Multivariable regression models aim to find the best-fitting equation that describes the relationship between the independent variables and the dependent variable. This equation has the form of

$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n + \epsilon \quad (3.28)$$

where y is the dependent variable, b_0 is the intercept, b_1, b_2, \dots, b_n are the coefficients for each independent variable (x_1, x_2, \dots, x_n), and ϵ is the error term.

By analyzing the coefficients, we can understand how each factor relates to time of completion.

In this case we will use the independent variables of: N and v_x , we only need 2, since N is derived from the volume of the water tank divided by the volume of each particle so we don't need to use r anymore, it will become an extra cost and not necessary. Thus

$$\begin{aligned} x_1 &= N \\ x_2 &= v_x \end{aligned}$$

Implementing multivariable regression in C++ typically involves handling matrix operations for calculations of coefficients. This can be achieved by implementing the necessary linear algebra functions from scratch or by utilizing existing C++ libraries designed for numerical linear algebra computation.

You can write your own C++ code from scratch to perform matrix multiplication, transposition, and inversion. But this provides a deep understanding of the underlying mathematics and can be complex, very time consuming and prone to errors.

Thankfully, in 2025 we have a high-performance C++ library that can handle float matrix and float vector operations with great precision, including transpose, inverse and multiplication, we will use **Armadillo / Arma** to help us computing the coefficients b_1 and b_2 .

Another alternative besides **Armadillo** is **Eigen**.

Now, suppose we have this matrix that we construct from the simulation data above (table 3.1):

$$X = \begin{bmatrix} 1 & 23437.5 & 19.5 \\ 1 & 23437.5 & 25 \\ 1 & 6944.44 & 19.5 \\ 1 & 6944.44 & 25 \\ 1 & 2929.6875 & 19.5 \\ 1 & 2929.6875 & 25 \end{bmatrix}, \quad y = \begin{bmatrix} 45 \\ 44.7 \\ 20 \\ 20.3 \\ 11.8 \\ 11.5 \end{bmatrix}$$

By any means, our measurements for matrix X and vector y / the simulation data could

contain human errors as well.

So we will have this matrix equation for this multivariable regression problem:

$$y = X * B + E \quad (3.29)$$

y is the vector of dependent variable values, X is the matrix of independent variables values (including a column of ones for the intercept term), B is the vector of regression coefficients to be estimated, and E is the vector of error terms.

We will use the most common method to estimate the coefficients B , which is the Ordinary Least Squares (OLS), which minimizes the sum of squared residuals. The formula for OLS coefficient estimation is:

$$B = (X^T * X)^{-1} * X^T * y \quad (3.30)$$

From **Armadillo** we will obtain

$$B = \begin{bmatrix} 8.3696 \\ 0.0016 \\ -0.0182 \end{bmatrix}$$

Thus

$$y = b_0 + b_1 x_1 + b_2 x_2 = 8.3696 + 0.0016N - 0.0182v_x$$

the equation above tells us about the relation / proportion between y and the independent variables

$$\begin{aligned} y &\propto N \\ y &\propto \frac{1}{v_x} \end{aligned}$$

```

root [ ~/SourceCodes/CPP/C++ Create Library/Armadillo-Test ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -larmadillo
root [ ~/SourceCodes/CPP/C++ Create Library/Armadillo-Test ]# ./main
Matrix X:
 1.0000e+00  2.3438e+04  1.9500e+01
 1.0000e+00  2.3438e+04  2.5000e+01
 1.0000e+00  6.9444e+03  1.9500e+01
 1.0000e+00  6.9444e+03  2.5000e+01
 1.0000e+00  2.9297e+03  1.9500e+01
 1.0000e+00  2.9297e+03  2.5000e+01
Vector y:
 45.0000
 44.7000
 20.0000
 20.3000
 11.8000
 11.5000

Matrix X^T:
 1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00  1.0000e+00
 2.3438e+04  2.3438e+04  6.9444e+03  6.9444e+03  2.9297e+03  2.9297e+03
 1.9500e+01  2.5000e+01  1.9500e+01  2.5000e+01  1.9500e+01  2.5000e+01

Matrix B:
 8.3696
 0.0016
 -0.0182

Xnew:
 1.0000e+00  6.9444e+03  2.5000e+01

Xnew * B:
 18.9128

```

Figure 3.7: The matrix B is computed with *Armadillo* and the predicted t for $N = 6944.44$ and $v_x = 25$ is 18.9128, while the t obtained from the simulation observation is 20.3, the error is 6.83% (*DFSimulatorC/Source Codes/C++/C++ Gnuplot SymbolicC++/ch23-Numerical Linear Algebra/Multivariable Regression with Armadillo/main.cpp*).

With this multivariable regression we are able to predict the time of completion to fill a water tank with any given volume of the water tank and any given speed of the particle for the filling, or perhaps in other case, to clear up an organ from dangerous substance; how many dosage of drugs should one person drink with certain mass and age as the independent variables to be healed from certain disease? How many chlorine needed to clear up a pool with certain volume and water quality?



Figure 3.8: The simulation for a water tank filling with a water particle that has the radius of $r = 0.4$ and $v_x = 25$ the t obtained from the simulation observation is 11.5 s (*Hamzstlab-Physics2D/tests/ode_watertankfilling.cpp*).

We would like to introduce a book we had made in 2023-2025 that has covered Elementary Linear Algebra for undergraduate level knowledge along with its' C++ code that can be tested by the reader here [5]. We are summarizing the theorems, formulas, definitions, solving the problems from chapter 1 to the last chapter of Chris-Rorres book [10], the addition is that we make the C++ codes ourselves with the help of using already established C++ libraries like **Armadillo**, **giNaC**, **Eigen**, **SymbolicC++**. It is very fortunate that we have learned Linear Algebra first, since it is proven now that it becomes very helpful when we learn other topics such as physics, mechanical engineering, differential equations, and so on. That all the C++ codes that we have made and put in that book is very helpful and can be reused again.

The book can be downloaded and all the source codes are available at:
<https://github.com/glanzkaiser/DFSimulatorC>

Since this example of water tank mixing is hypothetical, the validity of the model is not in question. If the flow rates are as stated, and if the concentration of salt in the tank is uniform, then the differential equation (3.17) is an accurate description of the flow process.

Model of this kind are often used in problems involving a pollutant in a lake, or a drug

in an organ of the body, for example, rather than a tank of salt water. In such cases the flow rates may not be easy to determine or may vary with time.

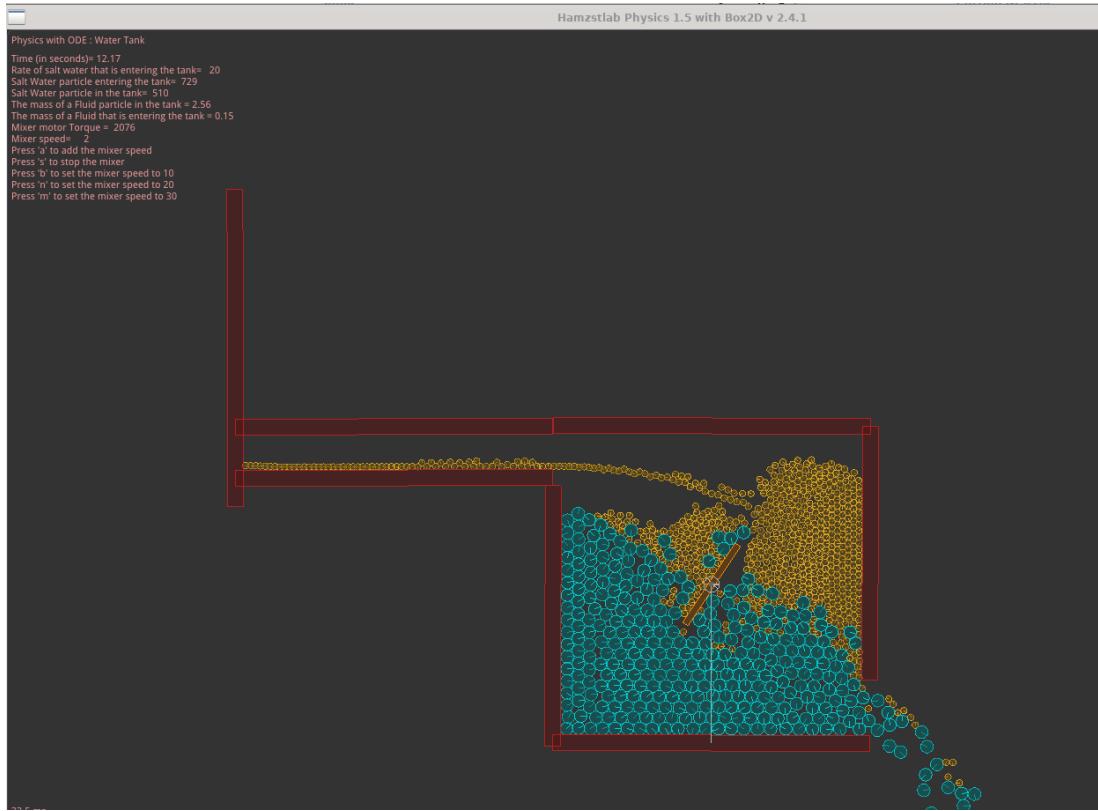


Figure 3.9: The 2D simulation of a salt water mixing in a tank made with Hamzstlab Physics 2D that is using Box2D version 2.4.2 as the backend library ([Hamzstlab-Physics2D/tests/ode_watertankmixing.cpp](#)).

We would like to introduce again one of our creation: **Hamzstlab Physics 2D**, a 2D classical physics simulator, that you can download and try / can be accessed at <https://github.com/glanzkaiser/Hamzstlab-Physics2D>

We make a lot of classical mechanics Physics simulations in 2D with Hamzstlab Physics 2D that is using the backend of Box2D version 2.4.2, at this opportunity we try to make a 2D simulation for this example so people / reader can see the illustration of this water tank mixing with salt water that is coming with certain rate / speed into the tank.

If SymIntegration is able to compute the differential equation solution, and combining it with Hamzstlab Physics we can make a better simulation that is near to real-life physics.

[SI*] Home Loan / Mortgage

A home buyer can afford to spend no more than USD 500 / month on mortgage payments. Suppose that the interest rate is 6%, that interest is compounded continuously, and that payments are also made continuously.

- Determine the maximum amount that this buyer can afford to borrow on a 20-year

mortgage.

- (b) Determine the total interest paid during the term of the mortgage.

Solution:

- (a) The amount of money $S(t)$ that the buyer has to pay changes in time due to two factors, the compound interest and its' continuous payments. The rate of growth for compounding is rS , and the rate of decay due to the continuous payments is k .

$$\frac{dS}{dt} = rS - k$$

$$\frac{dS}{dt} - rS = -k$$

This is a linear first-order inhomogeneous ODE, so it can be solved by multiplying both sides by an integrating factor $\mu(t)$

$$\mu(t) = e^{\int^t (-r) ds} = e^{-rt}$$

proceed with the multiplication and solve it for $S(t)$

$$\begin{aligned} e^{-rt} \frac{dS}{dt} - re^{-rt} S &= -ke^{-rt} \\ \frac{d}{dt}(e^{-rt} S) &= -ke^{-rt} \\ e^{-rt} S &= \frac{k}{r} e^{-rt} + C \\ S(t) &= \frac{k}{r} + Ce^{rt} \end{aligned}$$

Apply the initial condition $S(0) = S_0$ to determine C

$$\begin{aligned} S(0) &= \frac{k}{r} + Ce^{rt} \\ S_0 &= \frac{k}{r} + Ce^{rt} \\ C &= S_0 - \frac{k}{r} \end{aligned}$$

Therefore, the amount of money the buyer has to pay after t years is

$$\begin{aligned} S(t) &= \frac{k}{r} + \left(S_0 - \frac{k}{r} \right) e^{rt} \\ &= \frac{k}{r} (1 - e^{rt}) + S_0 e^{rt} \end{aligned}$$

For a year the home buyer will need to pay $k = 500 \times 12 = \text{USD } 6,000$ year and $r = 6\% = 0.06$.

$$S(t) = \frac{6000}{0.06} (1 - e^{0.06t}) + S_0 e^{0.06t}$$

For a 20-year mortgage, $S(t) = 0$ at $t = 20$.

$$0 = \frac{6000}{0.06}(1 - e^{0.06t}) + S_0 e^{0.06t}$$

$$S_0 = 100000 - 100000e^{-1.2}$$

$$S_0 \approx \text{USD } 69,880.58$$

The value of S_0 is how much the buyer can afford to borrow initially.

- (b) For the 20-year mortgage, the home buyer pays a total of

$$\text{USD } 6,000 \times 20 = \text{USD } 120,000$$

so the total interest paid is

$$\text{USD } 120,000 - \text{USD } 69,880.58 = \text{USD } 50,119.42$$

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP f
or Mortgage case ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP f
or Mortgage case ]# ./main

DSolve for S' = rS - k
S(t) = k*r^(-1)+C*e^(t*r)
S(t=0) = k*r^(-1)+C
For ivp S(0) = S0,
C = -k*r^(-1)+S0

Test ivp = k*r^(-1)-k*r^(-1)*e^(t*r)+S0*e^(t*r)
For ivp k = 6000, r = 6%, t = 20, S(t) = 0,
S(0) = -100000*e^(-1.2)+100000
```

Figure 3.10: The function `dsolve` and `ivp` in `SymIntegration` is able to compute the solution $S(t)$ for the ordinary differential equation of $\frac{dS}{dt} = Sr - k$ with the initial value problem $S(0) = S_0$ (`SymIntegration/Examples/Test SymIntegration DSolve and IVP for Mortgage case/main.cpp`).

As of July 20th, 2025 we just add a new function `ivp`, and it is managed in `/usr/include/symintegral/dsolve.h` and `/src/dsolve.cpp`

```
...
...
Symbolic ivp(const Symbolic &fx, const Symbolic &x, const Symbolic &c)
{
    Symbolic ivpsol, f0, C("C"), S0("S0");

    if(fx != 0)
    {
        list<Equations> eq;
        list<Equations>::iterator i;
        UniqueSymbol a, b;
        // Will work for this model: dS/dt = rS-k
        f0 = fx[x==0];
        C = solve(f0-S0,C).front().rhs ;
        ivpsol = fx[c==C];
```

```

    }
    return ivpsol;
}

...

```

Code 37: *src/dsolve.cpp***[SI*] Radiocarbon Dating Example:**

An important tool in archeological research is radiocarbon dating, developed by the American chemist Willard F. Libby. This is a means of determining the age of certain wood and plant remains, hence of animal or human bones or artifacts found buried at the same levels.

Radiocarbon dating is based on the fact that some wood or plant remains contain residual amounts of carbon-14, a radioactive isotope of carbon. This isotope is accumulated during the lifetime of the plant and begins to decay at its death. Since the half-life of carbon-14 is long (approximately 5730 years), measurable amounts of carbon-14 is still present, then by appropriate laboratory measurements the proportion of the original amount of carbon-14 that remains can be accurately determined. In other words, if $Q(t)$ is the amount of carbon-14 at time t and Q_0 is the original amount, then the ratio $\frac{Q(t)}{Q_0}$ can be determined, as long as this quantity is not too small. Present measurement techniques permit the use of this method for time periods of 50,000 years or more.

- Assuming that Q satisfies the differential equation $Q' = -rQ$, determine the decay constant r for carbon-14.
- Find an expression for $Q(t)$ at any time t , if $Q(0) = Q_0$
- Suppose that certain remains are discovered in which the current residual amounts of carbon-14 is 20% of the original amount. Determine the age of the remains.

Solution:

- The rate that the mass of carbon-14 decreases is assumed to be proportional to how much is present at any given time

$$\frac{dQ}{dt} \propto -Q$$

Change this proportionality to an equation by introducing a constant r on the right side.

$$Q' = -rQ$$

Divide both sides by Q .

$$\frac{Q'}{Q} = -r$$

The left side can be written as the derivative of a logarithm by the chain rule.

$$\begin{aligned} \frac{d}{dt} \ln Q &= -r \\ \ln Q &= -rt + C \\ Q(t) &= e^{-rt+C} \\ &= ce^{-rt} \end{aligned}$$

- (b) Suppose that there is a certain amount of mass Q_0 initially. Then the initial condition is $Q(0) = Q_0$. We will use it to determine c .

$$\begin{aligned} Q(0) &= ce^{-r(0)} \\ c &= Q_0 \end{aligned}$$

Consequently, the mass decays exponentially. Substituting back we will have the solution to the initial value problem as

$$Q(t) = Q_0 e^{-rt}$$

- (c) From the formula that we obtain at (b) we know that r can be determined with knowledge of the half-life. For carbon-14, specifically, we know that half the initial mass is lost after 5730 years.

$$\frac{Q_0}{2} = Q_0 e^{-r(5730)}$$

Solve this equation for r

$$\begin{aligned} e^{-5730r} &= \frac{1}{2} \\ \ln |e^{-5730r}| &= \ln \left| \frac{1}{2} \right| \\ r &= -\frac{\ln \left| \frac{1}{2} \right|}{5730} \\ &= \frac{\ln |2|}{5730} \\ r &\approx 0.0001210 \end{aligned}$$

r is the rate of the decreasing mass per year. Therefore, the mass of a sample of carbon-14 is

$$Q(t) = Q_0 e^{-\frac{\ln |2|}{5730} t}$$

where t is in years. If some remains have 20% of the original amount of carbon-14, then $Q(t) = 0.2Q_0$. Solve the resulting equation for t to determine how old the remains are.

$$\begin{aligned} 0.2Q_0 &= Q_0 e^{-\frac{\ln |2|}{5730} t} \\ e^{-\frac{\ln |2|}{5730} t} &= 0.2 \\ \ln |e^{-\frac{\ln |2|}{5730} t}| &= \ln |0.2| \\ -\frac{\ln |2|}{5730} t &= \ln \frac{1}{5} \\ -\frac{\ln |2|}{5730} t &= -\ln |5| \\ t &= \frac{\ln |5|}{\ln |2|} 5730 \\ t &\approx 13,305 \end{aligned}$$

so the age of the remains are around 13,305 years old.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP for Radiocarbon Dating ]# make
g++ -c -o main.o main.cpp
g++ -o main .gdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP for Radiocarbon Dating ]# ./main
in
DSolve for Q' = -rQ
Q(t) = C*e^(-t*r)
Q(t=0) = C
For ivp Q(0) = Q0,
C = Q0
Solution for the ivp,
Q(t) = Q0*e^(-t*r)
Half-life problem for carbon-14, the ivp becomes:
Q0*e^(-5730*r)-0.5*Q0 = 0
Determining the rate by solving the ivp solution
r = 0.000120968
Determining the age of the remains with 20% of carbon-14
t = 13304.6

```

Figure 3.11: The computation of the rate and the age of remains with SymIntegration' `dsolve` and `ivp` functions. (*SymIntegration/Examples/Test SymIntegration DSolve and IVP for Radiocarbon Dating/main.cpp*).

On this example, we have added another `else if` conditional so SymIntegration' `solve` function will be able to solve this equation:

$$ce^{-rx} = a$$

$$ce^{-rx} = a$$

$$x = -\frac{\ln \frac{a}{c}}{r} = \frac{\ln \frac{c}{a}}{r}$$

```

...
double division(Symbolic x, Symbolic y)
{
    return x/y;
}

Equations solve(const Symbolic &e, const Symbolic &x)
{
Equations soln;

...
...

else if(e.coeff(1,0) == e) // for case c * exp(-tx) = a or ln|tx| = a
{
    if(df(e,x).coeff(x,0) != 0 && df(df(e,x),x).coeff(x,0) != df(e,x).
        coeff(x,0) && -df(df(e,x),x).coeff(x,0) != df(e,x).coeff(x,0)
    ) // for case c * exp(-tx) = a ,
    {
        Symbolic t = df(df(e,x),x).coeff(x,0) / df(e,x).coeff(x,0);
        // will work for c == 1 or c != 1
        Symbolic a = -e.coeff(SymbolicConstant::e,0) ;
        Symbolic c = df(e,x).coeff(x,0)/t;
        soln = (soln, x == ln(division(a,c))/t);
    }
}

```

```

    }
    else if(df(e,x).coeff(x,0) != 0 && -df(df(e,x),x).coeff(x,0) ==
        df(e,x).coeff(x,0) ) // for case c * exp(-tx) = a , c !=1, t
        ==1
    {
        Symbolic a = - e.coeff(SymbolicConstant::e,0) ;
        Symbolic c = - df(e,x).coeff(x,0) ;
        soln = (soln, x == ln(division(c,a)));
    }
    ...
}
return soln;
}
...

```

Code 38: *src/solve.cpp***[SI*] Pollutants in a Lake:**

Consider a lake of constant volume V containing at time t an amount $Q(t)$ of pollutant, evenly distributed throughout the lake with a concentration $c(t)$, where $c(t) = \frac{Q(t)}{V}$. Assume that water containing a concentration k of pollutant enters the lake at a rate r , and that water leaves the lake at the same rate. Suppose that pollutants are also added directly to the lake at a constant rate P .

Note that given assumptions neglect a number of factors that may, in some cases, be important—for example, the water added or lost by precipitation, absorption, and evaporation; the stratifying effect of temperature differences in a deep lake; the tendency of irregularities in the coastline to produce sheltered bays; and the fact that pollutants are not deposited evenly throughout the lake but (usually) at isolated points around its periphery. The results below must be interpreted in the light of the neglect of such factors as these.

- If at time $t = 0$ the concentration of pollutant is c_0 , find an expression for the concentration $c(t)$ at any time. What is the limiting concentration as $t \rightarrow \infty$?
- If the addition of pollutants to the lake is terminated ($k = 0$ and $P = 0$ for $t > 0$), determine the time interval T that must elapse before the concentration of pollutants is reduced to 50% of its original value; to 10% of its original value.
- Table 5.1 contains data for several of the Great Lakes. Using these data, determine from part (b) the time T necessary to reduce the contamination of each of these lakes to 10% of the original value.

Solution:

- The conservation law that governs the mass of pollutant in the lake is as follows.

$$\text{rate of mass accumulation} = \text{rate of mass in} - \text{rate of mass out}$$

Lake	$V(km^3 \times 10^3)$	$r(km^3/year)$
Superior	12.2	65.2
Michigan	4.9	158
Erie	0.46	175
Ontario	1.6	209

Table 3.2: Volume and Flow Data for the Great Lakes

the rate of mass in is the sum of rk and P , and the rate of mass out is $rc(t)$.

$$\begin{aligned}\frac{dQ}{dt} &= rk + P - rc(t) \\ &= rk + P - \frac{r}{V}Q(t) \\ \frac{dQ}{dt} + \frac{r}{V}Q &= rk + P\end{aligned}$$

This is a first-order linear inhomogeneous ODE, so it can be solved by multiplying both sides by an integrating factor $\mu(t)$.

$$\begin{aligned}\mu(t) &= e^{\int \frac{r}{V} ds} = e^{\frac{rt}{V}} \\ e^{\frac{rt}{V}} \frac{dQ}{dt} + \frac{r}{V}e^{\frac{rt}{V}}Q &= rke^{\frac{rt}{V}} + Pe^{\frac{rt}{V}} \\ \frac{d}{dt}(e^{\frac{rt}{V}}Q) &= rke^{\frac{rt}{V}} + Pe^{\frac{rt}{V}} \\ e^{\frac{rt}{V}}Q &= \int^t (rke^{\frac{rs}{V}} + Pe^{\frac{rs}{V}}) ds + C \\ &= rk\left(\frac{V}{r}\right)e^{\frac{rt}{V}} + P\left(\frac{V}{r}\right)e^{\frac{rt}{V}} + C \\ e^{\frac{rt}{V}}Q &= kVe^{\frac{rt}{V}} + \frac{PV}{r}e^{\frac{rt}{V}} + C \\ Q(t) &= kV + \frac{PV}{r}e^{\frac{rt}{V}} + Ce^{-\frac{rt}{V}}\end{aligned}$$

Divide both sides by V to obtain the concentration.

$$c(t) = \frac{Q(t)}{V} = k + \frac{P}{r} + \frac{C}{V}e^{-\frac{rt}{V}}$$

Apply the initial condition $c(0) = c_0$ to determine C .

$$\begin{aligned}c(0) &= k + \frac{P}{r} + \frac{C}{V} \\ c_0 &= k + \frac{P}{r} + \frac{C}{V} \\ \frac{C}{V} &= c_0 - k - \frac{P}{r}\end{aligned}$$

Therefore, the concentration is

$$c(t) = \frac{Q(t)}{V} = k + \frac{P}{r} + \left(c_0 - k - \frac{P}{r} \right) e^{-\frac{rt}{V}}$$

Because of the decaying exponential function, the limit of $c(t)$ as $t \rightarrow \infty$ is

$$\lim_{t \rightarrow \infty} c(t) = k + \frac{P}{r}$$

- (b) If the pollution stops, then the rate of mass in becomes zero in the conservation law.

rate of mass accumulation = - rate of mass out

The rate of mass out is still $rc(t)$.

$$\begin{aligned} \frac{dQ}{dt} &= -rc(t) \\ &= -\frac{r}{V}Q(t) \\ \frac{\frac{dQ}{dt}}{Q} &= -\frac{r}{V} \\ \frac{dQ}{Q} &= -\frac{r}{V} dt \\ \ln Q &= -\frac{rt}{V} + C_1 \\ Q &= e^{-\frac{rt}{V} + C_1} \\ Q(t) &= C_2 e^{-\frac{rt}{V}} \end{aligned}$$

Since Q is the mass of pollutant, divide both sides by V to get the concentration.

$$c(t) = \frac{Q(t)}{V} = \frac{C_2}{V} e^{-\frac{rt}{V}}$$

Use the initial condition $c(0) = c_0$ to determine C_2

$$c(0) = \frac{C_2}{V} = c_0$$

Therefore

$$c(t) = c_0 e^{-\frac{rt}{V}}$$

Set $c(t) = 0.5c_0$ and solve for $t = T$ to find how long it will take for the concentration to reach half its initial value.

$$\begin{aligned} 0.5c_0 &= c_0 e^{-\frac{rT}{V}} \\ e^{-\frac{rT}{V}} &= 0.5 \\ \ln e^{-\frac{rT}{V}} &= \ln 0.5 \\ -\frac{rT}{V} &= \ln \frac{1}{2} \\ T &= \frac{V}{r} \ln 2 \end{aligned}$$

On the other hand, set $c(t) = 0.1c_0$ and solve for $t = T$ to find how long it will take for the concentration to reach one-tenth its initial value.

$$\begin{aligned}0.1c_0 &= c_0 e^{-\frac{rt}{V}} \\e^{-\frac{rt}{V}} &= 0.1 \\\ln e^{-\frac{rt}{V}} &= \ln 0.1 \\-\frac{rt}{V} &= \ln \frac{1}{10} \\T &= \frac{V}{r} \ln 10\end{aligned}$$

(c) Plug in the numbers fo V and r into the formula at part (b).

Lake Superior:

$$T = \frac{V}{r} \ln 10 = \frac{12.2 \times 10^3}{65.2} \ln 10 \approx 431 \text{ years}$$

Lake Michigan:

$$T = \frac{V}{r} \ln 10 = \frac{4.9 \times 10^3}{158} \ln 10 \approx 71.4 \text{ years}$$

Lake Erie:

$$T = \frac{V}{r} \ln 10 = \frac{0.46 \times 10^3}{175} \ln 10 \approx 6.05 \text{ years}$$

Lake Ontario:

$$T = \frac{V}{r} \ln 10 = \frac{1.6 \times 10^3}{209} \ln 10 \approx 17.6 \text{ years}$$

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP for Pollutant in Great Lakes ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lsstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration DSolve and IVP for Pollutant in Great Lakes ]# ./main

DSolve for Q' = rk + P - rc(t) , c(t) = Q(t)/V
Q(t) = k*V+P*V*t*r^(-1)+C*e^(-V^(-1)*t*r)
c(t) = k*P*r^(-1)+C*e^(-V^(-1)*t*r)*V^(-1)
c(0) = k*P*r^(-1)+C*V^(-1)
c0 - c(0) = c0-k*P*r^(-1)-C*V^(-1)
For ivp c(0) = c0,
C/V = c0-k-P*r^(-1)

Therefore, the concentration is,
c(t) = k*P*r^(-1)+c0*e^(-V^(-1)*t*r)-k*e^(-V^(-1)*t*r)-P*r^(-1)*e^(-V^(-1)*t*r)

The limit of c(t) as t -> ∞
lim_{t -> ∞} c(t) = k*P*r^(-1)+c0*e^(-inf*V^(-1)*r)-k*e^(-inf*V^(-1)*r)-P*r^(-1)*e^(-inf*V^(-1)*r)

If the pollution stops, then
Q(t) = C*e^(-V^(-1)*t*r)

The concentration when the pollution stops is
c(t) = C*e^(-V^(-1)*t*r)*V^(-1)

For ivp c(0) = c0,
c0-C*V^(-1) = 0

Therefore the ivp solution will be,
c(t) = c0*e^(-V^(-1)*t*r)

Set c(t) = 0.5 c0 and solve for t=T,
T = 0.693147*V*r^(-1)

Set c(t) = 0.1 c0 and solve for t=T,
T = 2.30259*V*r^(-1)

Time to reduce the contamination in each lakes to 10 percent of the original value

Lake Superior: T = 430.852
Lake Michigan: T = 71.4093
Lake Erie: T = 6.05251
Lake Ontario: T = 17.6274

```

Figure 3.12: The computation of initial value problem solution for the pollutant concentration in a lake, $c(t)$, and the time to clean up a lake to make the pollutant concentration less than 10% with SymIntegration's `dsolve` function. (`SymIntegration/Examples/Test SymIntegration DSolve and IVP for Pollutant in Great Lakes/main.cpp`).

II. HIGHER ORDER LINEAR EQUATIONS

- General Theory of n th 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*]

Chapter 4

Probability and Statistics Computation with SymIntegration

"Education is the best provision for old age." - Aristotle

At a glance, people might be thinking that probability is only numerical computation, the probability that a head will show up on a coin that is tossed upward is around 0.5 on a fair coin. After more thorough reading, statistics and probability are actually more symbolic than numeric. We just substitute the random variable with a number that we want to test / to know the probability a certain event will occur is how much. That is the main reason, why we put statistics and probability computation in SymIntegration.

On August 23rd, 2025 we have created a new source code **statistics.cpp** that will handle the probability and statistics computation, like computing probability mass function of a certain discrete distribution, the moment generating function and to compute the mean and variance. It could come in handy one day.

I. PROCESSING AND SUMMARIZING DATA WITH STATISTICS

[SI*] Statistical methods are used to analyze data from an industrial process (manufacturing, energy sources, drug testing). Statistical methods are involved with gathering of information or **scientific data**, which then will be processed to extract the knowledge from the data and improve the **quality** of the process.

i. Descriptive Statistics Computation

Suppose we have a univariate data of a Mathematics examination result for 25 students, compute its' mean, median, mode, standard deviation, and variance.

Solution:

To compute the mean we will use this formula:

$$\bar{x} = \frac{\sum x}{n} \quad (4.1)$$

Mode is the value that occur frequently, and Median is the middle value after the data is sorted, if the size of the data is even, then the median is the average of the two middle data.

Standard deviation can be computed with this formula:

$$S_x = \sqrt{\frac{\sum x^2}{n} - (\bar{x})^2} \quad (4.2)$$

Standard deviation is the square root of variance.

In SymIntegration there is a function that can be used to compute the standard descriptive statistics, it is:

double descriptivestatistics(vector<double>)

the input is a vector with double as the data type, since it can also represents integer.

We have saved the grade data in a textfile named **vectorx.txt**, we can just load / call it from the **main.cpp** file.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Load Vector from textfile and Compute Descriptive Statistics ]# make
g++
-c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Load Vector from textfile and Compute Descriptive Statistics ]# ./main
Statistics Descriptive
Mode : 6
Median : 6
Mean : 6.2
Quantile 1/4: 5
Quantile 3/4: 8
Variance : 6.41667
Standard deviation : 2.53311

Time taken by function: 833 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Load Vector from textfile and Compute Descriptive Statistics ]# ]
```

Figure 4.1: The computation of descriptive statistics with SymIntegration took 833 microseconds (*SymIntegration/Examples/Test SymIntegration Load Vector from textfile and Compute Descriptive Statistics/-main.cpp*).

II. RANDOM VARIABLES AND PROBABILITY DISTRIBUTIONS

Definition 4.1: Random Variable

A random variable is a function that associates a real number with each element in the sample space.

We shall use a capital letter, say X , to denote a random variable and its corresponding small letter, x in this case, for one of its values.

Definition 4.2: Discrete Sample Space

If a sample space contains a finite number of possibilities or an unending sequence with as many elements as there are whole numbers, it is called a discrete sample space.

Definition 4.3: Continuous Sample Space

If a sample space contains an infinite number of possibilities equal to the number of points on a line segment, it is called a continuous sample space.

i. Discrete Probability Distributions

Definition 4.4: Probability Mass Function

If X is discrete, the probability mass function of X is defined as

$$f(x) = P(X = x), \quad \forall x \in \mathbb{R} \quad (4.3)$$

the probability is concentrated at only discrete values in the range of X , and is zero for all other values. f must also obey the basic rules of probability. That is, f must be non-negative

$$f(x) \geq 0$$

and the sum of all probabilities should add to 1

$$\sum_x f(x) = 1$$

Definition 4.5: Discrete Cumulative Distribution Function

The cumulative distribution function $F(x)$ of a discrete random variable X with probability distribution $f(x)$ is

$$F(x) = P(X \leq x) = \sum_{t \leq x} f(t), \quad \text{for } -\infty < x < \infty \quad (4.4)$$

ii. Continuous Probability Distributions

Definition 4.6: Probability Density Function

If X is continuous, its range is the entire set of real numbers \mathbb{R} . The probability of any specific value x is only one out of the infinitely many possible values in the range of X , which means that

$$P(X = x) = 0$$

for all $x \in \mathbb{R}$. The probability mass is spread so thinly over the range of values, that it can be measured only over intervals $[a, b] \subset \mathbb{R}$, rather than at specific points.

the probability density function of X that takes on values in any interval $[a, b] \subset \mathbb{R}$ is defined as

$$P(X \in [a, b]) = \int_a^b f(x) dx \quad (4.5)$$

the density function f must satisfy the basic laws of probability

$$f(x) \geq 0, \quad \forall x \in \mathbb{R}$$

and

$$\int_{-\infty}^{\infty} f(x) dx = 1$$

Definition 4.7: Continuous Cumulative Distribution Function

The function $f(x)$ is a probability density function (pdf) for the continuous random variable X , defined over the set of real numbers, if

$$f(x) \geq 0, \quad \forall x \in \mathbb{R} \quad (4.6)$$

$$\int_{-\infty}^{\infty} f(x) dx = 1 \quad (4.7)$$

$$P(a < X < b) = \int_a^b f(x) dx \quad (4.8)$$

Definition 4.8: Cumulative Distribution Function

For any random variable X , whether discrete or continuous, we can define the cumulative distribution function (cdf) as

$$F : \mathbb{R} \rightarrow [0, 1]$$

that gives the probability of observing a value at most some given value x

$$F(x) = P(X \leq x), \quad \forall -\infty < x < \infty \quad (4.9)$$

when X is discrete, F is given as

$$F(x) = P(X \leq x) = \sum_{u \leq x} f(u) \quad (4.10)$$

and when X is continuous, F is given as

$$F(x) = P(X \leq x) = \int_{-\infty}^x f(u) du \quad (4.11)$$

iii. Joint Probability Distributions

Definition 4.9: Joint Probability Distribution

The function $f(x, y)$ is a joint probability distribution or probability mass function of the discrete random variables X and Y if

$$f(x, y) \geq 0, \quad \forall (x, y) \quad (4.12)$$

$$\sum_x \sum_y f(x, y) = 1 \quad (4.13)$$

$$P(X = x, Y = y) = f(x, y) \quad (4.14)$$

For any region A in the xy plane,

$$P[(X, Y) \in A] = \sum_A \sum_A f(x, y) \quad (4.15)$$

Definition 4.10: Joint Density Function

The function $f(x, y)$ is a joint density function of the continuous random variables X and Y if

$$f(x, y) \geq 0, \quad \forall (x, y) \quad (4.16)$$

$f(x, y)$ is a surface lying above the xy plane, and $P[(X, Y) \in A]$, where A is any region in the xy plane. The joint density function is equal to the volume of the right cylinder bounded by the base A and the surface.

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) dx dy = 1 \quad (4.17)$$

$$P[(X, Y) \in A] = \int \int_A f(x, y) dx dy \quad (4.18)$$

for any region A in the xy plane.

Definition 4.11: Marginal Distributions

Given the joint probability distribution $f(x, y)$ of the discrete random variables X and Y , the probability distribution $g(x)$ of X alone is obtained by summing $f(x, y)$ over the values of Y . Similarly, the probability distribution $h(y)$ of Y alone is obtained by summing $f(x, y)$ over the values of X .

We define $g(x)$ and $h(y)$ to be the marginal distributions of X and Y , respectively. When X and Y are continuous random variables, summations are replaced by integrals.

The marginal distributions of X alone and Y alone are

$$\begin{aligned} g(x) &= \sum_y f(x, y) \\ h(y) &= \sum_x f(x, y) \end{aligned} \quad (4.19)$$

for the discrete case, and

$$\begin{aligned} g(x) &= \int_{-\infty}^{\infty} f(x, y) dy \\ h(y) &= \int_{-\infty}^{\infty} f(x, y) dx \end{aligned} \quad (4.20)$$

for the continuous case.

Definition 4.12: Conditional Distribution

Let X and Y be two random variables, discrete or continuous. The conditional distribution of the random variable Y given that $X = x$ is

$$f(y|x) = \frac{f(x,y)}{g(x)} \quad (4.21)$$

provided $g(x) > 0$.

Similarly, the conditional distribution of X given that $Y = y$ is

$$f(x|y) = \frac{f(x,y)}{h(y)} \quad (4.22)$$

provided $h(y) > 0$.

If we wish to find the probability that the discrete random variable X falls between a and b when it is known that the discrete variable $Y = y$, we evaluate

$$P(a < X < b|Y = y) = \sum_{a < x < b} f(x|y) \quad (4.23)$$

where the summation extends over all values of X between a and b . When X and Y are continuous, we evaluate

$$P(a < X < b|Y = y) = \int_a^b f(x|y) dx \quad (4.24)$$

Definition 4.13: Statistically Independent

Let X and Y be two random variables, discrete or continuous, with joint probability distribution $f(x,y)$ and marginal distributions $g(x)$ and $h(y)$, respectively. The random variables X and Y are said to be statistically independent if and only if

$$f(x,y) = g(x)h(y) \quad (4.25)$$

for all (x,y) within their range.

Definition 4.14: Mutually Statistically Independent

Let X_1, X_2, \dots, X_n be n random variables, discrete or continuous, with joint probability distribution $f(x_1, x_2, \dots, x_n)$ and marginal distribution $f_1(x_1), f_2(x_2), \dots, f_n(x_n)$, respectively. The random variables X_1, X_2, \dots, X_n are said to be mutually statistically independent if and only if

$$f(x_1, x_2, \dots, x_n) = f_1(x_1)f_2(x_2)\dots f_n(x_n) \quad (4.26)$$

for all (x_1, x_2, \dots, x_n) within their range.

Definition 4.15: Mean

Let X be a random variable with probability distribution $f(x)$. The mean, or expected value, of X is

$$\mu = E(X) = \sum_x x f(x) \quad (4.27)$$

if X is discrete, and

$$\mu = E(X) = \int_{-\infty}^{\infty} x f(x) dx \quad (4.28)$$

if X is continuous.

Theorem 4.1: Expected Value of a Random Variable $g(X)$

Let X be a random variable with probability distribution $f(x)$. The expected value of the random variable $g(X)$ is

$$\mu_{g(X)} = E[g(X)] = \sum_x g(x) f(x) \quad (4.29)$$

if X is discrete, and

$$\mu_{g(X)} = E[g(X)] = \int_{-\infty}^{\infty} g(x) f(x) dx \quad (4.30)$$

if X is continuous.

Definition 4.16: Mean for Two Random Variables

Let X and Y be random variables with joint probability distribution $f(x, y)$. The mean, or expected value, of the random variable $g(X, Y)$ is

$$\mu_{g(X,Y)} = E[g(X, Y)] = \sum_x \sum_y g(x, y) f(x, y) \quad (4.31)$$

if X and Y are discrete, and

$$\mu_{g(X,Y)} = E[g(X, Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) f(x, y) dx dy \quad (4.32)$$

if X and Y are continuous.

Definition 4.17: Variance

Let X be a random variable with probability distribution $f(x)$ and mean μ . The variance of X is

$$\sigma^2 = E[(X - \mu)^2] = \sum_x (x - \mu)^2 f(x) \quad (4.33)$$

if X is discrete, and

$$\sigma^2 = E[(X - \mu)^2] = \int_{-\infty}^{\infty} (x - \mu)^2 f(x) dx \quad (4.34)$$

if X is continuous.

The positive square root of the variance, σ , is called the standard deviation of X . The quantity $x - \mu$ is called the deviation of an observation from its mean.

Theorem 4.2: Variance

The variance of a random variable X is

$$\sigma^2 = E(X^2) - \mu^2 \quad (4.35)$$

Theorem 4.3: Variance of Random Variable $g(X)$

Let X be a random variable with probability distribution $f(x)$. The variance of the random variable $g(X)$ is

$$\sigma_{g(X)}^2 = E\{[g(X) - \mu_{g(X)}]^2\} = \sum_x [g(X) - \mu_{g(X)}]^2 f(x) \quad (4.36)$$

if X is discrete, and

$$\sigma_{g(X)}^2 = E\{[g(X) - \mu_{g(X)}]^2\} = \int_{-\infty}^{\infty} [g(X) - \mu_{g(X)}]^2 f(x) dx \quad (4.37)$$

Definition 4.18: Covariance

Let X and Y be random variables with joint probability distribution $f(x, y)$. The covariance of X and Y is

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = \sum_x \sum_y (x - \mu_X)(y - \mu_Y)f(x, y) \quad (4.38)$$

if X and Y are discrete, and

$$\sigma_{XY} = E[(X - \mu_X)(Y - \mu_Y)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \mu_X)(y - \mu_Y)f(x, y) dx dy \quad (4.39)$$

if X and Y are continuous.

The covariance between two random variables is a measure of the nature of the association between the two.

If large values of X often result in large values of Y or small values of X result in small values of Y , positive $X - \mu_X$ will often result in positive $Y - \mu_Y$ and negative $X - \mu_X$ will often result in negative $Y - \mu_Y$.

Thus, the product $(X - \mu_X)(Y - \mu_Y)$ will tend to be positive. On the other hand, if large X values often result in small Y values, the product $(X - \mu_X)(Y - \mu_Y)$ will tend to be negative.

When X and Y are statistically independent, it can be shown that the covariance is zero. The converse, however, is not generally true. Two variables may have zero covariance and still not be statistically independent. Note that the covariance only describes the linear relationship between two random variables. Therefore, if a covariance between X and Y is zero, X and Y may have a nonlinear relationship, which means that they are not necessarily independent.

Theorem 4.4: Covariance

The covariance of two random variables X and Y with means μ_X and μ_Y , respectively, is given by

$$\sigma_{XY} = E(XY) - \mu_X\mu_Y \quad (4.40)$$

Definition 4.19: Correlation

Let X and Y be random variables with covariance σ_{XY} and standard deviations σ_X and σ_Y , respectively. The correlation coefficient of X and Y is

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X\sigma_Y} \quad (4.41)$$

it should be clear that ρ_{XY} is free of the units of X and Y . The correlation coefficient satisfies the inequality $-1 \leq \rho_{XY} \leq 1$. It assumes a value of zero when $\sigma_{XY} = 0$.

iv. Covariance and Vectorized Moments

Given two random variables, X and Y , with respective means, μ_X and μ_Y , the covariance is defined by

$$\text{Cov}(X, Y) = E[(X - \mu_X)(Y - \mu_Y)] = E[XY] - \mu_X\mu_Y$$

The second formula follows by expansion. Notice also that

$$\text{Cov}(X, X) = \text{Var}(X)$$

the covariance is a common measure of the relationship between the two random variables, where if $\text{Cov}(X, Y) = 0$, we say the random variables are uncorrelated. Furthermore, if $\text{Cov}(X, Y) \neq 0$, the sign of the covariance gives an indication of the direction of the relationship.

Another important concept is the correlation coefficient,

$$\rho_{XY} = \frac{\text{Cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} \quad (4.42)$$

It is a normalized form of the covariance with $-1 \leq \rho_{XY} \leq 1$. Values nearing ± 1 indicate a very strong linear relationship between X and Y , whereas values near or at 0 indicate a lack of a linear relationship.

Note that if X and Y are independent random variables, then $\text{Cov}(X, Y) = 0$ and hence $\rho_{XY} = 0$. However, the opposite case does not always hold, since, in general $\rho_{XY} = 0$ does not imply independence. Nevertheless for jointly normal random variables it does.

Consider now a random vector $\mathbf{X} = (X_1, \dots, X_n)$, taken as a column vector. It can be described by moments in an analogous manner to a scalar random variable. A key quantity is the mean vector,

$$\mu_{\mathbf{X}} := [E[X_1], E[X_2], \dots, E[X_n]]^T$$

Furthermore, the covariance matrix is the matrix defined by the expectation (taken element-wise) of the (outer product) random matrix given by $(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^T$, and is expressed as

$$\Sigma_{\mathbf{X}} = \text{Cov}(\mathbf{X}) = E[(\mathbf{X} - \mu_{\mathbf{X}})(\mathbf{X} - \mu_{\mathbf{X}})^T] \quad (4.43)$$

As can be verified, the i, j -th element of $\Sigma_{\mathbf{X}}$ is $\text{Cov}(X_i, X_j)$ and hence the diagonal elements are the variances.

The covariance matrix is a fundamental concept in statistics and data analysis, providing insight into the relationships between multiple variables. It captures the extent to which two variables change together and is essential for the various applications such as the Principal Component Analysis (PCA), financial analysis and multivariate statistics. By understanding and interpreting the covariance matrix, we can make more informed decisions and uncover deeper patterns in data.

v. Linear Combinations and Transformations

We now consider linear transformations applied to random vectors. For any collection of random variables,

$$E[X_1 + \dots + X_n] = E[X_1] + \dots + E[X_n]$$

For uncorrelated random variables,

$$\text{Var}(X_1 + \dots + X_n) = \text{Var}(X_1) + \dots + \text{Var}(X_n)$$

More generally, if we allow the random variables to be correlated, then

$$\text{Var}(X_1 + \dots + X_n) = \text{Var}(X_1) + \dots + \text{Var}(X_n) + 2 \sum_{i < j} \text{Cov}(X_i, X_j) \quad (4.44)$$

Note that the right-hand side of Eq. (4.51) is the sum of the elements of the matrix $\text{Cov}(X_1, \dots, X_n)$. This is a special case of a more general affine transformation, where we take a random vector $\mathbf{X} = (X_1, \dots, X_n)$ with covariance matrix Σ_X , and an $m \times n$ matrix A and m vector \mathbf{b} . We then set

$$\mathbf{Y} = A\mathbf{X} + \mathbf{b} \quad (4.45)$$

In this case, the new random vector \mathbf{Y} exhibits mean and covariance

$$\begin{aligned} E[\mathbf{Y}] &= AE[\mathbf{X}] + \mathbf{b} \\ \text{Cov}(\mathbf{Y}) &= A\Sigma_X A^T \end{aligned} \quad (4.46)$$

Now to retrieve Eq. (4.51), we set $1 \times n$ matrix $A = [1, \dots, 1]$ and observe that $A\Sigma_X A^T$ is a sum of all of the elements of Σ_X .

vi. The Cholesky Decomposition and Generating Random Vectors

If you wish to create an n -dimensional random vector \mathbf{Y} with some specified mean vector μ_Y and covariance matrix Σ_Y . That is, μ_Y and Σ_Y are known.

The formulas in Eqs. (4.53) yield a potential recipe for such a task if we are given a random vector \mathbf{X} with zero-mean and identity-covariance matrix ($\Sigma_X = I$). For example, in the context of Monte Carlo random variable generation, creating such a random vector \mathbf{X} is trivial—just generate a sequence of n i.i.d. normal $(0, 1)$ random variables.

Now, apply the affine transformation Eq. (4.52) on \mathbf{X} with $\mathbf{b} = \mu_Y$ and a matrix A that satisfies

$$\Sigma_Y = AA^T \quad (4.47)$$

Now Eq. (4.53) guarantees that \mathbf{Y} has the desired μ_Y and Σ_Y .

vii. Generate Random Number in SymIntegration

Generating random number is the essence or basic need in statistical inference, it is very useful as well if we want to simulate physics phenomena or forecasting weather with random initial condition.

In SymIntegration we have two functions to generate random number, the first one is:

randomnumberint(int a, int b, int n)

with a and b as the range so the number generated will be in $[a, b]$, and n is the amount of random number that we want to generate. This function will generate integer random number only.

The second function is:

randomnumberreal(double a, double b, int n)

with a and b as the range so the number generated will be in $[a, b]$, and n is the amount of random number that we want to generate. This function will generate real random number, in C++ we use **double** data type to represent the real number here.

The functions are written in **src/statistics.cpp** with the header that corresponds to this is in **/include/symintegral/statistics.h**. We are using uniform distribution to generate the random number to ensure that the probability of each number in the range to be picked is the same, equal opportunity for all.

```
root | ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Generate Random Number from Function ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root | ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Generate Random Number from Function ]# ./main
Random integer:
19
129
123
133
93
58
173
82
88
28
173
Random double:
1.06899
0.683921
4.21156
3.94917
2.27366
0.971918
4.0864
3.09766
2.44007
3.7999

Time taken by function: 759 microseconds
root | ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Generate Random Number from Function ]# ]
```

Figure 4.2: The random number generation with SymIntegration took 759 microseconds (*SymIntegration/Examples/Test SymIntegration Generate Random Number from Function/main.cpp*).

We are using the standard C++ code **std::random_device rd** as a uniform random bit generator that is intended to produce non-deterministic random numbers, typically by accessing a hardware entropy source or a system-provided source of randomness. Its purpose is to provide a high-quality, truly random seed for other pseudo-random number generators. the **rd** is an object of type **std::random_device**.

Then, we also use **std::mt19937 generate** as the Mersenne Twister engine, a widely used and well-regarded pseudo-random number generator (PRNG). It produces a long sequence of pseudo-random numbers based on an initial seed. The **generate** is an object of type **std::mt19937**,

the `rd()` part calls the `operator()` of the `std::random_device` object, which returns a single random value. This value is then used as the seed to initialize the `std::mt19937` engine `generate`.

`std::random_device` provides a non-deterministic seed, ensuring that each run of the program will likely produce a different sequence of random numbers. This seed then initializes `std::mt19937`, which is a high-quality and efficient pseudo-random number generator suitable for most non-cryptographic applications. Subsequent calls to `generate()` or distributions associate with `generate` will produce the pseudo-random numbers.

viii. Create Covariance Matrix in SymIntegration

The general form of a covariance matrix is as follow:

$$\Sigma_X = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \dots & \sigma_{1n} \\ \sigma_{21} & \sigma_{22} & \dots & \sigma_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{n1} & \sigma_{n2} & \dots & \sigma_{nn} \end{bmatrix} = \begin{bmatrix} \text{Cov}(X_1, X_1) & \text{Cov}(X_1, X_2) & \dots & \text{Cov}(X_1, X_n) \\ \text{Cov}(X_2, X_1) & \text{Cov}(X_2, X_2) & \dots & \text{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(X_n, X_1) & \text{Cov}(X_n, X_2) & \dots & \text{Cov}(X_n, X_n) \end{bmatrix}$$

we can tell that covariance matrix is a symmetric matrix with the property of $\sigma_{ij} = \sigma_{ji}$. The formula to compute σ_{ij} can be seen at Eq. (4.47).

For sample covariance the formula is as follow:

$$\sigma_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

Now suppose we have a mid-term exam results by Sine, Sweden, and Bludut in Linear Algebra and Differential Equations. Make a covariance matrix.

Student	Linear Algebra	Differential Equations
Sine	80	70
Sweden	63	20
Bludut	100	50

Table 4.1: The mid-term exam results.

With SymIntegration we can create the covariance matrix with this function:
covariancematrix(vector<vector<double>)

The output will be a **SymbolicMatrix** or **Matrix<Symbolic>**, a class that is originally made from SymbolicC++. The goodness of this SymbolicMatrix is that it can also contain symbolic input.

The result will be show that

$$\begin{aligned} \text{Cov}(X, X) &= \sigma_{11} = 343 \\ \text{Cov}(X, Y) &= \sigma_{12} = 260 \\ \text{Cov}(Y, Y) &= \sigma_{22} = 633.33 \end{aligned}$$

the covariance for X (the Linear Algebra mid-term score) and Y (the Differential Equations mid-term score) is 260, as this is a positive number, it means that when X increases (or decreases) Y also increases (or decreases).

While the diagonal elements are 343 and 633.33 indicate the variance in data sets X and Y respectively. X shows the lowest variance, while Y shows the highest variance.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Create Covariance Matrix from Textfile ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Create Covariance Matrix from Textfile ]# ./main
Data:
    89      70
    63      20
   100      50

The covariance matrix:
[ 343   260 ]
[ 260 633.333]

Time taken by function: 1265 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Create Covariance Matrix from Textfile ]# ]
```

Figure 4.3: The covariance matrix for this example case. (*SymIntegration/Examples/Test SymIntegration Create Covariance Matrix from Textfile/main.cpp*).

ix. Create Cholesky Decomposition in SymIntegration

Perform a Cholesky decomposition on

$$\Sigma_X = \begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix}$$

We will use this formula

$$A_{ii} = \sqrt{\Sigma_{X_{ii}} - \sum_{k=0}^{i-1} (A_{ik})^2}$$

The result is

$$\begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{bmatrix} \begin{bmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{bmatrix}$$

In SymIntegration the Cholesky decomposition can be performed with this function:
choleskyDecomposition(vector<vector<double> matrix)

The function is **void**, the goodness of using **void** is we don't need to have **return ..** at the end of the function, and we can call the result / the output after we call the function.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Cholesky Decomposition ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Cholesky Decomposition ]# ./main
Data:
      4       12      -16
     12       37      -43
    -16      -43       98

A =
      2       0       0
      6       1       0
     -8       5       3

A^T =
      2       6      -8
      0       1       5
      0       0       3

Time taken by function: 889 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Cholesky Decomposition ]# []
```

Figure 4.4: The Cholesky decomposition for a square matrix of size 3, we load the matrix from textfile **matrix.txt**. (*SymIntegration/Examples/Test SymIntegration Cholesky Decomposition/main.cpp*).

III. DISCRETE PROBABILITY DISTRIBUTIONS

[SI*] We will start with the Bernoulli process, then we will cover the definition of other discrete distributions.

Definition 4.20: Bernoulli Process

An experiment that consists of repeated trials, each with two possible outcomes that may be labeled success or failure is called a Bernoulli process.

The Bernoulli process must possess the following properties:

1. The experiment consists of repeated trials.
2. Each trial results in an outcome that may be classified as a success or a failure.
3. The probability of success, denoted by p , remains constant from trial to trial.
4. The repeated trials are independent.

Definition 4.21: Binomial Distribution

The number X of successes in n Bernoulli trials is called a binomial random variable. The probability distribution of this discrete random variable is called the binomial distribution, and its values will be denoted by $b(x; n, p)$.

A Bernoulli trial can result in a success with probability p and a failure with probability $q = 1 - p$. Then the probability distribution of the binomial random variable X , the number of successes in n independent trials, is

$$b(x; n, p) = \binom{n}{x} p^x (q^{n-x}), \quad x = 0, 1, 2, \dots, n \quad (4.48)$$

Theorem 4.5: Mean and Variance of the Binomial Distribution

The mean and variance of the binomial distribution $b(x; n, p)$ are

$$\mu = np \quad (4.49)$$

$$\sigma^2 = npq \quad (4.50)$$

Definition 4.22: Negative Binomial Distribution

The number X of trials required to produce k successes in a negative binomial experiment is called a negative binomial random variable, and its probability distribution is called the negative binomial distribution.

If repeated independent trials can result in a success with probability p and a failure with probability $q = 1 - p$, then the probability distribution of the random variable, X , the number of the trial on which the k th success occurs, is

$$b^*(x; k, p) = \binom{x-1}{k-1} p^k q^{x-k}, \quad x = k, k+1, k+2, \dots \quad (4.51)$$

Definition 4.23: Geometric Distribution

If repeated independent trials can result in a success with probability p and a failure with probability $q = 1 - p$, then the probability distribution of the random variable X , the number of the trial on which the first success occurs, is

$$g(x; p) = pq^{x-1}, \quad x = 1, 2, 3, \dots \quad (4.52)$$

Theorem 4.6: The Mean and Variance of the Geometric Distribution

The mean and variance of a random variable following the geometric distribution are

$$\mu = \frac{1}{p} \quad (4.53)$$

$$\sigma^2 = \frac{1-p}{p^2} \quad (4.54)$$

Definition 4.24: Poisson Process

Experiments yielding numerical values of a random variable X , the number of outcomes occurring during a given time interval or in a specified region, are called Poisson experiments. A Poisson experiment can generate observations for the random variable X representing the number of telephone calls received per hour by an office, the number of days school is closed due to snow during the winter, or the number of games postponed due to rain during a baseball season.

Properties of the Poisson process:

1. The number of outcomes occurring in one time interval or specified region of space is independent of the number that occur in any other disjoint time interval or region. In this sense we say that the Poisson process has no memory.
2. The probability that a single outcome will occur during a very short time interval or in a small region is proportional to the length of the time interval or the size of the region and does not depend on the number of outcomes occurring outside this time interval or region.
3. The probability that more than one outcome will occur in such a short time interval or fall in such a small region is negligible.

The number X of outcomes occurring during a Poisson experiment is called a Poisson random variable, and its probability distribution is called the Poisson distribution.

If you want to learn more on non-homogeneous Poisson process, it allows for a varying rate of events over time or space, and also the average rate (λ) can becomes a function of time or space.

Definition 4.25: Poisson Distribution

The probability distribution of the Poisson random variable X , representing the number of outcomes occurring in a given time interval or specified region denoted by t , is

$$p(x; \Delta t) = \frac{e^{-\lambda t} (\lambda t)^x}{x!}, \quad x = 0, 1, 2, \dots \quad (4.55)$$

where λ is the average number of outcomes per unit time, distance, area, or volume and $e = 2.71828$.

Use cases of Poisson distribution:

1. Traffic flow: Modeling the number of cars passing through a toll booth in a given time period.
2. Call Centers: Predicting the number of incoming calls during specific hours.
3. Insurance Claims: Estimating the number of insurance claims within a certain timeframe.
4. Web Server Requests: Analyzing the number of requests a server receives in a fixed time interval.
5. Epidemiology: Studying the occurrence of diseases or rare events in a population.

Practical applications of Poisson distribution:

1. Network Security: Analyzing the number of security breaches or attacks on a network within a specific timeframe.
2. Inventory Management: Estimating the number of items sold in a store during a particular hour.
3. Quality Control: Assessing the number of defects in a manufacturing process.
4. Biology and Genetics: Studying the distribution of mutations in a DNA sequence.
5. Finance: Predicting the number of defaults in a loan portfolio.

Theorem 4.7: Mean and Variance of the Poisson Distribution

Both the mean and the variance of the Poisson distribution $p(x; \lambda t)$ are

$$\mu = \sigma^2 = \lambda t \quad (4.56)$$

where t is the specific "time", "distance", "area", or "volume" of interest.

Theorem 4.8: Approximation of Binomial Distribution by a Poisson Distribution

Let X be a binomial random variable with probability distribution $b(x; n, p)$. When $n \rightarrow \infty$, $p \rightarrow 0$, and $np \xrightarrow{n \rightarrow \infty} \mu$ remains constant,

$$b(x; n, p) \xrightarrow{n \rightarrow \infty} p(x; \mu) \quad (4.57)$$

[SI*] To compute the probability mass function, cumulative density function, mean, variance, and moment generating function for the discrete distributions above we can use these functions in SymIntegration:

**binomialpmf(x,n,p)
binomialcdf(x,n,p)
binomialmean(x,n,p)
binomialvar(x,n,p)
binomialmgf(x,n,p)**

**negativebinomialpmf(x,k,p)
negativebinomialmean(x,k,p)
negativebinomialvar(x,k,p)
negativebinomialmgf(x,k,p)**

**geometricpmf(x,p)
geometricmean(x,p)
geometricvar(x,p)
geometricmgf(x,p)**

**poissonpmf(x,λt)
poissoncdf(x,λt)
poissonmean(x,λt)
poissonvar(x,λt)
poissonmgf(x,λts)**

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Discrete Distributions ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o `stdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Discrete Distributions ]# ./main

binomialpmf(2;4,0.75) = 0.210938
binomialcdf(8;15,0.4) = 0.904953
binomialmean(8;15,0.4) = 6
binomialvar(8;15,0.4) = 3.6
binomialmgf(8;15,0.4) = 0.4*e^t+0.6
negativebinomialpmf(6;4,0.55) = 0.1853
negativebinomialmean(6;4,0.55) = 7.27273
negativebinomialvar(6;4,0.55) = 5.95041
negativebinomialmgf(6;4,0.55) = 0.0915063*(-0.45*e^t+1)^(-4)
geometricpmf(5;0.01) = 0.00960596
geometricmean(5;0.01) = 100
geometricvar(5;0.01) = 9900
geometricmgf(5;0.01) = 0.01*(-0.99*e^t+1)^(-1)
poissonpmf(6;4) = 0.104196
poissoncdf(6;4) = 0.889326
poissonmean(6;4) = 4
poissonvar(6;4) = 4
poissonmgf(6;4) = e^(4*e^t-4)

Time taken by function: 56539 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Discrete Distributions ]# []

```

Figure 4.5: The functions in SymIntegration to compute the probability mass function, cumulative distribution function, mean, variance, and moment generating function for several discrete probability distributions (*SymIntegration/Examples/Test SymIntegration Discrete Distributions/main.cpp*).

[SI*] To generate random numbers that have certain discrete distributions we can use these functions in SymIntegration:

vrandn_bernoulli(p, n)
vrandn_binomial(p, n)

with n as the number of random number we want to generate, the result will be in **std::vector<double>**.

IV. CONTINUOUS PROBABILITY DISTRIBUTIONS

[SI*] We will show the definitions and theorems of continuous distributions, starting from uniform distribution.

i. Uniform Distribution

Definition 4.26: Uniform Distribution

The density function of the continuous uniform random variable X on the interval $[A, B]$ is

$$f(x; A, B) = \begin{cases} \frac{1}{B-A}, & A \leq x \leq B, \\ 0, & \text{elsewhere} \end{cases} \quad (4.58)$$

the density function forms a rectangle with base $B - A$ and constant height $\frac{1}{B-A}$. The uniform distribution is often called the rectangular distribution. This is the simplest continuous distributions in all of statistics.

Theorem 4.9: The Mean and Variance

The mean and variance of the uniform distribution are

$$\mu = \frac{A + B}{2} \quad (4.59)$$

$$\sigma^2 = \frac{(B - A)^2}{12} \quad (4.60)$$

ii. Normal Distribution

Definition 4.27: Normal Distribution

This is the most important continuous probability distribution in the entire field of statistics.

A continuous random variable X having the bell-shaped distribution is called a normal random variable. The density of the normal random variable X , with mean μ and variance σ^2 , is

$$n(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}, \quad -\infty < x < \infty \quad (4.61)$$

where $\pi = 3.14159$ and $e = 2.71828$.

The properties of the normal curve:

1. The mode, which is the point on the horizontal axis where the curve is a maximum, occurs at $x = \mu$.
2. The curve is symmetric about a vertical axis through the mean μ .
3. The curve has its points of inflection at $x = \mu \pm \sigma$; it is concave downward if $\mu - \sigma < X < \mu + \sigma$ and is concave upward otherwise.
4. The normal curve approaches the horizontal axis asymptotically as we proceed in either direction away from the mean.
5. The total area under the curve and above the horizontal axis is equal to 1.

The normal distribution finds enormous application as a limiting distribution. Under certain conditions, the normal distribution provides a good continuous approximation to the binomial and hypergeometric distributions.

Theorem 4.10: The Mean and Variance

The mean and variance of $n(x; \mu, \sigma)$ are μ and σ^2 . Hence, the standard deviation is σ .

The curve of any continuous probability distribution or density function is constructed so that the area under the curve bounded by two ordinates $x = x_1$ and $x = x_2$ equals the probability that the random variable X assumes a value between $x = x_1$ and $x = x_2$. Thus, for the normal curve we will have

$$P(x_1 < X < x_2) = \int_{x_1}^{x_2} n(x; \mu, \sigma) dx = \frac{1}{\sqrt{2\pi}\sigma} \int_{x_1}^{x_2} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \quad (4.62)$$

Definition 4.28: Standard Normal Distribution

We are able to transform all the observations of any normal random variable X into a new set of observations of a normal random variable Z with mean 0 and variance 1. This can be done by means of the transformation

$$Z = \frac{X - \mu}{\sigma} \quad (4.63)$$

Whenever X assumes a value x , the corresponding value of Z is given by

$$z = \frac{x - \mu}{\sigma}$$

Therefore, if X falls between the values $x = x_1$ and $x = x_2$, the random variable Z will fall between the corresponding values

$$z_1 = \frac{x_1 - \mu}{\sigma}$$

and

$$z_2 = \frac{x_2 - \mu}{\sigma}$$

Consequently, we may write

$$\begin{aligned} P(x_1 < X < x_2) &= \frac{1}{\sqrt{2\pi}\sigma} \int_{x_1}^{x_2} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\ &= \frac{1}{\sqrt{2\pi}(1)} \int_{x_1}^{x_2} e^{-\frac{1}{2(1)^2}(z)^2} dz \\ &= \int_{x_1}^{x_2} n(z; 0, 1) dz \\ &= P(z_1 < Z < z_2) \end{aligned} \quad (4.64)$$

where Z is seen to be a normal random variable with mean 0 and variance 1.

The distribution of a normal random variable with mean 0 and variance 1 is called a standard normal distribution.

Theorem 4.11: Normal Approximation to the Binomial

If X is a binomial random variable with mean $\mu = np$ and variance $\sigma^2 = npq$, then the limiting form of the distribution of

$$Z = \frac{X - np}{\sqrt{npq}}$$

as $n \rightarrow \infty$, is the standard normal distribution $n(z; 0, 1)$.

Definition 4.29: Normal Approximation to the Binomial Distribution

Let X be a binomial random variable with parameters n and p . For large n , X has approximately a normal distribution with $\mu = np$ and $\sigma^2 = npq = np(1 - p)$ and

$$\begin{aligned} P(X \leq x) &= \sum_{k=0}^x b(k; n, p) \\ &= P\left(Z \leq \frac{x + 0.5 - np}{\sqrt{npq}}\right) \end{aligned}$$

as the area under normal curve to the left of $x + 0.5$, and the approximation will be good if np and $np(1 - p)$ are greater than or equal to 5.

[SI*] Computing the cdf of a Normal Distribution

When computing for the probability density function of a normal distribution we can use the Eq (4.53) directly, but the problem is when we have to compute the cumulative distribution function of a normal distribution, it will take an advanced level of integration that has no known indefinite integral, take a look at Eq. (4.53)

$$P(X = x) = n(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2}, \quad -\infty < x < \infty$$

When we want to compute the cdf we will integrate the pdf just like this

$$\begin{aligned} P(X \leq x) &= \int_{-\infty}^x \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\ &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{1}{2\sigma^2}(x-\mu)^2} dx \\ &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{1}{2\sigma^2}(x^2 - 2\mu x + \mu^2)} dx \end{aligned}$$

The computation for the integral above is not easy analytically since there is no elementary indefinite integral for

$$\int e^{-x^2} dx$$

Now, we would like to introduce to the Gaussian integral, also known as Euler-Poisson integral, it is the integral of the Gaussian function $f(x) = e^{-x^2}$ over the entire real line. The integral is

$$\int_{-\infty}^{\infty} e^{-x^2} dx = \sqrt{\pi} \tag{4.65}$$

Carl Friedrich Gauss published the precise integral in 1809. The integral has a wide range of applications, e.g. with a slight chance of variables it is used to compute the normalizing constant of the normal distribution. In physics this type of integral appears frequently in quantum mechanics.

The same integral with finite limits is closely related to both the error function and the cumulative distribution function of the normal distribution. So we can approximate the cdf of normal distribution with error function.

The Gaussian integral can be solved analytically through the methods of multivariable calculus. Thus if we have an arbitrary Gaussian function the definite integral is

$$\int_{-\infty}^{\infty} e^{-a(x+b)^2} dx = \sqrt{\frac{\pi}{a}} \quad (4.66)$$

In short, the cdf for a normal distribution can be related to error function, so the cdf formula is

$$\phi(x) = P(X \leq x) = \frac{1}{2} \operatorname{erf}\left(-\frac{x}{\sqrt{2}}\right) \quad (4.67)$$

for $\mu = 0, \sigma = 1$.

$$\phi(x) = P(X \leq x) = \frac{1}{2} \operatorname{erf}\left(-\frac{\frac{x-\mu}{\sigma}}{\sqrt{2}}\right) \quad (4.68)$$

for $\mu \neq 0, \sigma \neq 1$. The error function itself can be computed with Gamma function or Hypergeometric function

$$\operatorname{erf}(x) = 1 - \frac{\Gamma\left(\frac{1}{2}, x^2\right)}{\sqrt{\pi}} \quad (4.69)$$

$$\operatorname{erf}(x) = \frac{2x}{\sqrt{\pi}} {}_2F_1\left(-, \frac{1}{2}; \frac{3}{2}; -x^2\right) \quad (4.70)$$

Luckily, the standard library for C++ `<cmath>` already has the function to compute the error function `erfc(x)` that we can use and embed in SymIntegration. Undergraduate students usually only taught to read the cdf for normal distribution from statistics table, the derivative to compute the cdf itself is rarely taught at undergraduate level. This is the background, or the engine behind all that values at the statistics table. How integral plays an important part in statistics, it really shows that statistics actually more symbolic than numeric.

iii. Gamma Distribution

Definition 4.30: Gamma Function

The Gamma distribution derives its name from the well-known gamma function. The Gamma function is defined by

$$\Gamma(\alpha) = \int_0^{\infty} x^{\alpha-1} e^{-x} dx, \quad \text{for } \alpha > 0 \quad (4.71)$$

Simple properties of the gamma function:

- (a) $\Gamma(n) = (n-1)(n-2)\dots(1)\Gamma(1)$, for a positive integer n .
- (b) $\Gamma(n) = (n-1)!$ for a positive integer n .
- (c) $\Gamma(1) = 1$.
- (d) $\Gamma(\frac{1}{2}) = \sqrt{\pi}$

Definition 4.31: Gamma Distribution

The continuous random variable X has a gamma distribution, with parameters α and β , if its density function is given by

$$f(x; \alpha, \beta) = \begin{cases} \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}, & x > 0 \\ 0, & \text{elsewhere} \end{cases} \quad (4.72)$$

where $\alpha > 0$ and $\beta > 0$. Often we call α as shape and β as scale.

The exponential distribution is a special case of the gamma distribution. The exponential and gamma distributions play an important role in both queuing theory and reliability problems.

The special gamma distribution for which $\alpha = 1$ is called the exponential distribution.

The Lower Incomplete Gamma Function $\gamma(s, x)$

In order to compute the cdf of several continuous distributions, e.g. chi-squared and Gamma, without using external library like **Boost** we need to know this function: the lower incomplete Gamma function.

The lower incomplete Gamma function is defined by the integral:

$$\gamma(s, x) = \int_0^x t^{s-1} e^{-t} dt$$

the definite integral above is quite hard to compute, so we approximate it with the Taylor expansion, the expansion of the exponential function and term by term integration give the following expansion:

$$\gamma(s, x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+s}}{n!(n+s)}$$

We can use the series expansion above and create the C++ code to compute the cdf of Gamma distribution or chi-squared distribution. The lower incomplete Gamma function will come in handy for other statistics related computation.

The relations for the Gamma functions:

$$\gamma(s, x) + \Gamma(s, x) = \Gamma(s)$$

with $\Gamma(s, x)$ as the upper incomplete Gamma function.

The Gamma Distribution' cdf with Lower Incomplete Gamma Function $\gamma(s, x)$

The Gamma cdf formula is an integral that represents the probability of a Gamma-distributed

random variable being less than or equal to a specific value, it is defined as

$$\begin{aligned} P(X \leq x) &= F(x; \alpha, \beta) \\ &= \frac{1}{\Gamma(\alpha)} \int_0^x t^{\alpha-1} e^{-\beta t} dt \\ &= \frac{\gamma(\alpha, \frac{x}{\beta})}{\Gamma(\alpha)} \end{aligned}$$

With α as the shape parameter and β as the scale parameter. Be careful, some textbooks use rate parameter: $\lambda = \frac{1}{\beta}$ instead of using β . Computer can only use this formula due to the complex integral, the cdf is calculated using numerical methods.

iv. Exponential Distribution

Definition 4.32: Exponential Distribution

The continuous random variable X has an exponential distribution, with parameter β , if its density function is given by

$$f(x; \beta) = \begin{cases} \frac{1}{\beta} e^{-\frac{x}{\beta}}, & x > 0 \\ 0, & \text{elsewhere} \end{cases} \quad (4.73)$$

where $\beta > 0$. Sometimes we also write exponential distribution density function this way

$$f(x; \beta) = \begin{cases} \lambda e^{-\lambda x}, & x > 0 \\ 0, & \text{elsewhere} \end{cases} \quad (4.74)$$

The exponential distribution is more appropriate when the memoryless property is justified. If the failure of the component is a result of gradual or slow wear, then the exponential does not apply and either the gamma or the Weibull distribution may be more appropriate.

Theorem 4.12: The Mean and Variance

The mean and variance of the gamma distribution are

$$\begin{aligned} \mu &= \alpha\beta \\ \sigma^2 &= \alpha\beta^2 \end{aligned} \quad (4.75)$$

The mean and variance of the exponential distribution are

$$\begin{aligned} \mu &= \beta \\ \sigma^2 &= \beta^2 \end{aligned} \quad (4.76)$$

v. Beta Distribution

Definition 4.33: Beta Function

A beta function is defined by

$$B(\alpha, \beta) = \int_0^1 x^{\alpha-1} (1-x)^{\beta-1} dx = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}, \quad \text{for } \alpha, \beta > 0 \quad (4.77)$$

where $\Gamma(\alpha)$ is the gamma function.

Definition 4.34: Beta Distribution

The continuous random variable X has a beta distribution with parameters $\alpha > 0$ and $\beta > 0$ if its density function is given by

$$f(x) = \begin{cases} \frac{1}{B(\alpha,\beta)} x^{\alpha-1} (1-x)^{\beta-1}, & 0 < x < 1 \\ 0, & \text{elsewhere} \end{cases} \quad (4.78)$$

Note that the uniform distribution on $(0, 1)$ is a beta distribution with parameters $\alpha = 1$ and $\beta = 1$. The major reason why we use Beta distribution is because it is defined in the range of $[0, 1]$ so a Beta distribution is a very natural distribution to use when we are talking about probabilities and we want to specify about a prior knowledge of the probabilities of something accruing.

Applications:

1. Survival Analysis

In fields like medicine and engineering, the Beta distribution is used to analyze survival times or failure rates, aiding in understanding product reliability or treatment effectiveness. By modeling these distributions, researchers gain insights into how long products last or how treatments perform over time. This statistical tool helps assess durability and reliability, crucial for making informed decisions in various industries.

2. Time Management

In project management, the Beta distribution helps estimate task durations and overall project completion times. It's like a tool that assists in planning and scheduling. By using this distribution, project managers can better understand the range of possible timeframes for completing tasks or the entire project. This aids in making more informed decisions and managing time effectively.

3. Risk Management

Risk management is essential in finance and insurance. It's about figuring out the chances of different events happening and how they could affect things. By looking at probabilities, it helps make smart choices to reduce risks.

4. Genetics

The beta distribution is a part of the population genetics that not only allows scientists to keep tabs on genes at the population level but as well makes them understand how the genes mutate within the whole group. Allele frequencies, or the different version of genes, must be borne in mind while studying pedigrees and evolution. Utilizing the Beta distribution data, the researchers are able to track the changes in the allele distribution on behalf of the intergenerational population. Such data is informative as far as evolutionary patterns and the nature of ecosystems is known.

5. Quality Control

In manufacturing industries, the Beta distribution serves as a crucial tool for characterizing the variability in product quality. By employing this distribution, businesses gain valuable insights into the distribution of defects or deviations from desired specifications within their production processes. This understanding enables effective control measures to be implemented, ensuring products consistently meet or exceed quality standards. By leveraging the Beta distribution, manufacturers can optimize processes, minimize waste, and enhance overall product reliability, fostering customer satisfaction and trust.

The shape of the distribution is determined entirely by the parameters α and β :

- $\alpha = 1, \beta = 1$: Uniform distribution on $(0, 1)$
- $\alpha > 1, \beta > 1$: Unimodal distribution with a peak within the interval $(0, 1)$. As α and β increase, the peak becomes sharper and more centered.
- $\alpha < 1, \beta < 1$: U-shaped distribution with peaks at the endpoints of the interval.
- $\alpha > 1, \beta < 1$: Right-skewed distribution
- $\alpha < 1, \beta > 1$: Left-skewed distribution

The Beta (α, β) distribution converges to the normal distribution as $\alpha, \beta \rightarrow \infty$. This convergence can be seen by reducing the variance in the graph of the beta distribution. The probability mass then becomes more concentrated around the mean, and the distribution becomes more bell-shaped.

Definition 4.35: Moment Generating Function for Beta Distribution

The moment generating function (mgf) for beta distribution is defined by

$$M_X(\alpha; \beta; t) = 1 + \sum_{i=1}^{\infty} \left(\prod_{j=0}^{i-1} \frac{\alpha + j}{\alpha + \beta + j} \right) \frac{t^i}{i!} \quad (4.79)$$

this function is a series representation of a beta distribution that can be used to compute moments like mean and variance by computing the derivative of the mgf.

The mgf of the beta distribution can be related to the Kummer's confluent hypergeometric function (of the first kind). It is a power series solution to Kummer's differential equation (or confluent hypergeometric equation), an important equation in physics, chemistry, and engineering. It is used to solve problems involving normal and slow rotationally symmetric TE modes in azimuthally magnetized circular ferrite waveguides.

Thus, this is the step by step to obtain the moment generating function for beta distribution

$$\begin{aligned} M_X(\alpha; \beta; t) &= E[e^{tX}] \\ &= \int_0^1 e^{tx} f(x; \alpha, \beta) dx \\ &= \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} \int_0^1 e^{tx} x^{\alpha-1} (1-x)^{\beta-1} dx \\ &= \frac{1}{B(\alpha, \beta)} \int_0^1 e^{tx} x^{\alpha-1} (1-x)^{\beta-1} dx \\ &= \sum_{n=0}^{\infty} \frac{\alpha^{(n)}}{(\alpha + \beta)^n} \frac{t^n}{n!} \\ &= 1 + \sum_{k=1}^{\infty} \left(\prod_{r=0}^{k-1} \frac{\alpha + r}{\alpha + \beta + r} \right) \frac{t^k}{k!} \\ &= {}_1F_1(\alpha; \alpha + \beta; t) \end{aligned}$$

where

$$x^{(n)} = x(x+1)(x+2)\dots(x+n-1)$$

is the rising factorial, also known as the "Pochhammer symbol."

We want to put it simply like this, the mgf of the beta distribution is:

$$M_X(\alpha; \beta; t) = {}_1F_1(\alpha; \alpha + \beta; t)$$

where

$${}_1F_1(\alpha; \beta; t) = \sum_{n=0}^{\infty} \frac{\alpha^{(n)}}{(\beta)^n} \frac{t^n}{n!} \quad (4.80)$$

and

$${}_1F_1(\alpha; \alpha + \beta; t) = \sum_{n=0}^{\infty} \frac{\alpha^{(n)}}{(\alpha + \beta)^n} \frac{t^n}{n!} \quad (4.81)$$

The good news is the **GNU scientific library** and **Boost** are able to numerically compute the hypergeometric function ${}_1F_1$ directly for a small t ($|t| < 1$), but not for the symbolic computation. So we add this to SymIntegration, thus SymIntegration is able to compute hypergeometric function ${}_1F_1$ numerically and symbolically, with limitation of course, the speed is slower than GSL and Boost since symbolic function will has more cost overall, knowing we can do more after that, to integrate it symbolically or derivate it.

It is handled in `src/specialfunctions.cpp` along with the include in `include/symintegral/specialfunctions.h`

```
Symbolic hypergeometric_1F1(double a, double b, const Symbolic &s, int
    max_iterations) {
    Symbolic sum = 1;

    for (int i = 1; i <= max_iterations; ++i)
    {
        sum += (rising_pochhammer(a, i) * (s^(i))) / (rising_pochhammer
            (b, i) * factorial(i));
    }
    return sum;
}

double hypergeometric_1F1(double a, double b, double s, int max_iterations) {
    Symbolic sum = 1;

    for (int i = 1; i <= max_iterations; ++i)
    {
        sum += (rising_pochhammer(a, i) * (s^(Symbolic(i)))) / (
            rising_pochhammer(b, i) * factorial(i));
    }
    return sum;
}
```

Code 39: `src/specialfunctions.cpp`

We put the **max_iterations** as 5, since it gives the same numeric answer as Boost and GSL.

```
-- 
root [ ~/SourceCodes/CPP/C++ Create Library/SI Test/Test Hypergeometric 1F1 GSL vs Boost ]# make
g++
  -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lgsl -lcblas
root [ ~/SourceCodes/CPP/C++ Create Library/SI Test/Test Hypergeometric 1F1 GSL vs Boost ]# ./main
GSL: Hypergeometric 1F1 (2.0, 3.0, 0.5): 1.405115e+00
Boost: MGF of Beta(2, 3) at t=0.5 is: 1.22758
Time taken by function: 273 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/SI Test/Test Hypergeometric 1F1 GSL vs Boost ]# []
```

Figure 4.6: The computation of the hypergeometric function and the mgf of beta distribution ($\alpha = 2, \beta = 3, t = 0.5$) with GSL and Boost has a very fast speed. (*SymIntegration/Examples/Test Hypergeometric 1F1 GSL vs Boost/main.cpp*).

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# make
g++
  -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsyminTEGRATION
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# ./main
for n = 5
Hypergeometric_1F1(2, 3, z) = 0.666667*z+0.25*z^(2)+0.0666667*z^(3)+0.0138889*z^(4)+0.00238095*z^(5)+1
Hypergeometric_1F1(2, 3, 0.5) = 1.40511
mgf Beta (2, 3, 0.5) = 1.22758
Time taken by function: 24444 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# []
```

Figure 4.7: The computation of the complete hypergeometric function and the mgf of beta distribution ($\alpha = 2, \beta = 3, t = 0.5$) with SymIntegration, it is slower than GSL and Boost but able to use symbolic of "z". (*SymIntegration/Examples/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta/main.cpp*).

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# make clean
rm -f main.o main
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# make
g++
  -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsyminTEGRATION
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# ./main
for n = 5
Hypergeometric_1F1(2, 3, z) = 0.666667*z+0.25*z^(2)+0.0666667*z^(3)+0.0138889*z^(4)+0.00238095*z^(5)+1
mgf Beta (2, 3, 0.5) = 1.22758
Time taken by function: 19414 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta ]# []
```

Figure 4.8: The computation of the symbolic hypergeometric function and the mgf of beta distribution ($\alpha = 2, \beta = 3, t = 0.5$) with SymIntegration. (*SymIntegration/Examples/Test SymIntegration Compute Hypergeometric Function 1F1 and mgf Beta/main.cpp*).

Definition 4.36: Survival Function

The survival function, also known as the complementary cumulative distribution function (ccdf), is the probability that X takes a value greater than x , or

$$P(X > x)$$

it is defined as

$$S(x; \alpha, \beta) = 1 - F(x; \alpha, \beta) = 1 - \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} \quad (4.82)$$

In R, the survival function can be calculated using the **pbeta** function with the **lower.tail = FALSE** argument.

Real life applications of Beta distribution:

1. Crop Yield Analysis

Farmers and researchers use the Beta distribution to understand how crop yields vary across different areas of land. This helps them decide where to plant crops, how to allocate resources, and manage risks.

2. Sport Analytics

In sports, the Beta distribution helps analyze player performance metrics like shooting accuracy or goal-scoring rates. Coaches and analysts use this information to make strategic decisions.

3. Environmental Risk Assessment

Environmental scientists use the Beta distribution to model uncertainty in environmental factors like pollutant concentrations. This helps assess potential risks and guide policy decisions.

4. Clinical Trials

In medical research, the Beta distribution models treatment outcomes or response rates in clinical trials. This helps researchers design trials, determine sample sizes, and accurately analyze results.

5. Weather Forecasting

Meteorologists use the Beta distribution to model the distribution of weather phenomena such as rainfall or wind speeds. This helps predict weather patterns and assess the likelihood of extreme events.

Theorem 4.13: Mean and Variance

The mean and variance of a beta distribution with parameters α and β are

$$\mu = \frac{\alpha}{\alpha + \beta} \quad (4.83)$$

$$\sigma^2 = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)} \quad (4.84)$$

The first two theoretical moments for Beta distribution are the mean (μ) and variance (σ^2). When we are doing observation or experiment we obtain samples and we can compute the sample mean and variance with

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n - 1}$$

Now by equating $\bar{x} = \mu$ and $s^2 = \sigma^2$ we will obtain two equations with two unknowns (α, β). Solving these equations simultaneously gives the estimates for α and β . This is called the method of moments, the drawback is that it can be inefficient and sensitive to outliers. Other methods, such as maximum likelihood estimation, often provide more accurate estimates.

The Beta Distribution' cdf with Regularized Incomplete Beta Function $I_x(\alpha, \beta)$

The beta cdf can be expressed using the regularized incomplete beta function, which is defined as the incomplete beta function ($B(x; \alpha, \beta)$) divided by the complete beta function ($B(\alpha, \beta)$). This can be written as:

$$F_X(x) = \frac{B(x; \alpha, \beta)}{B(\alpha, \beta)} = I_x(\alpha, \beta)$$

where $I_x(\alpha, \beta)$ is the regularized incomplete beta function.

The incomplete beta function is defined as an integral:

$$B(x; \alpha, \beta) = \int_0^x u^{\alpha-1} (1-u)^{\beta-1} du$$

And the complete beta function can be expressed using the Gamma function:

$$B(\alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)}$$

The incomplete beta function and its regularized form are crucial in statistics, particularly in calculating probabilities related to beta distribution and in determining cumulative distribution functions (cdf) for distributions like the binomial, F -distribution, and Student's t -distribution.

Characteristics of the cdf and sf:

1. Both cdf and sf are increasing functions on the interval $(0, 1)$.
2. The cdf approaches 0 as x approaches 0 and approaches 1 as x approaches 1.
3. The cdf approaches 1 as x approaches 0 and approaches 0 as x approaches 1.
4. The sum of the cdf and sf at any point x is equal to 1.
5. The shapes of the cdf and sf depend on the values of α and β . For example, if $\alpha > 1$ and $\beta > 1$, the cdf is the S-shaped and the sf is concave down.

vi. Chi-Squared Distribution

Definition 4.37: Chi-Squared Distribution

The chi-squared distribution with ν degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables.

The chi-squared distribution χ_{ν}^2 is a special case of the gamma distribution and the univariate Wishart distribution. Specifically if $X \sim \chi_{\nu}^2$ then $X \sim \text{Gamma}(\alpha = \frac{\nu}{2}, \beta = 2)$ (where α is the shape parameter and β is the scale parameter of the gamma distribution) and $X \sim W_1(1, k)$.

The probability density function of the chi-squared distribution is

$$f(x; \nu) = \frac{1}{\Gamma(\nu/2)2^{\nu/2}} x^{\nu/2-1} e^{-x/2}, \quad 0 < x < 1 \quad (4.85)$$

the density function is CUPULU.

The cumulative distribution function is

$$F(x; \nu) = \frac{\gamma(\frac{\nu}{2}, \frac{x}{2})}{\Gamma(\frac{\nu}{2})} = P\left(\frac{\nu}{2}, \frac{x}{2}\right) \quad (4.86)$$

where $\gamma(s, t)$ is the lower incomplete gamma function and $P(s, t)$ is the regularized gamma function.

The chi-squared distribution is one of the most widely used probability distributions in inferential statistics, notably in hypothesis testing and in construction of confidence intervals.

The lower incomplete gamma function is given by

$$\begin{aligned} \gamma(a, x) &= \int_0^x t^{a-1} e^{-t} dt \\ &= a^{-1} x^a e^{-x} {}_1F_1(1; 1+a; x) \\ &= a^{-1} x^a {}_1F_1(a; 1+a; -x) \end{aligned}$$

where ${}_1F_1(a; b; x)$ is the confluent hypergeometric function of the first kind.

By definition, the lower and upper incomplete gamma functions satisfy

$$\Gamma(a, x) + \gamma(a, x) = \Gamma(a) \quad (4.87)$$

Theorem 4.14: The Mean and Variance

The mean and variance of the chi-squared distribution are

$$\mu = \nu \quad (4.88)$$

$$\sigma^2 = 2\nu \quad (4.89)$$

The Chi-Squared Distribution' cdf with Lower Incomplete Gamma Function $\gamma(s, x)$

The chi-squared cdf is expressed using the incomplete Gamma function as

$$\begin{aligned} P(X \leq x) &= F(x; \nu) \\ &= \frac{1}{\Gamma(\frac{\nu}{2})} \gamma\left(\frac{\nu}{2}, \frac{x}{2}\right) \end{aligned}$$

with ν as the degrees of freedom, Γ is the Gamma function, and γ is the lower incomplete Gamma function.

We can also write the cdf in a series expansion for computation:

$$F(x; \nu) = \left(\frac{x}{2}\right)^{\left(\frac{k}{2}\right)} e^{-\frac{x}{2}} \sum_{m=0}^{\infty} \frac{\left(\frac{x}{2}\right)^m}{\Gamma\left(\frac{k}{2} + m + 1\right)}$$

vii. t -Distribution

In many experimental scenarios, knowledge of σ is certainly no more reasonable than knowledge of the population mean μ . Often, in fact, an estimate of σ must be supplied by the same sample information that produced the sample average \bar{x} . As a result, a natural statistic to consider to deal with inferences on μ is

$$T = \frac{\bar{X} - \mu}{S/\sqrt{n}} \quad (4.90)$$

since S is the sample analog to σ . If the sample size is small, the values of S^2 fluctuate considerably from sample to sample and the distribution T deviates appreciably from that of a standard normal distribution.

If the sample size is large enough, say $n \geq 30$, the distribution of T does not differ considerably from the standard normal. However, for $n < 30$, it is useful to deal with the exact distribution of T . In developing the sampling distribution of T , we shall assume that our random sample was selected from a normal population. We can then write

$$T = \frac{(\bar{X} - \mu)/(\sigma/\sqrt{n})}{\sqrt{S^2/\sigma^2}} = \frac{Z}{\sqrt{V/(n-1)}} \quad (4.91)$$

where

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

has the standard normal distribution and

$$V = \frac{(n-1)S^2}{\sigma^2} \quad (4.92)$$

has a chi-squared distribution with $\nu = n - 1$ degrees of freedom. In sampling from normal populations, we can show that \bar{X} and S^2 are independent, and consequently so are Z and V .

Theorem 4.15: *t*-Distribution

Let Z be a standard normal random variable and V a chi-squared random variable with ν degrees of freedom. If Z and V are independent, then the distribution of the random variable T , where

$$T = \frac{Z}{\sqrt{\frac{V}{\nu}}} \quad (4.93)$$

with $\nu = n - 1$, is given by the density function

$$h(t) = \frac{\Gamma\left[\frac{\nu+1}{2}\right]}{\Gamma\left(\frac{\nu}{2}\right)\sqrt{\pi\nu}} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \quad -\infty < t < \infty \quad (4.94)$$

This is known as the *t*-distribution with ν degrees of freedom.

The *t*-distribution represents the structure that occurs if all of the values of $\frac{\bar{x}-\mu}{s/\sqrt{n}}$ are formed, where \bar{x} and s are taken from samples of size n from a $n(x; \mu, \sigma)$ distribution.

viii. *F*-Distribution

In probability theory and statistics, the *F*-distribution or *F*-ratio, also known as Snedecor's *F* distribution or the Fisher Snedecor distribution, is a continuous probability distribution that arises frequently as the null distribution of a test statistic, most notably in the analysis of variance (ANOVA) and other *F*-tests.

As you can see, the *F*-distribution finds enormous application in comparing sample variances. Applications of the *F*-distribution are found in problems involving two or more samples.

The statistic F is defined to be the ratio of two independent chi-squared random variables, each divided by its number of degrees of freedom. Hence, we can write

$$F = \frac{U/\nu_1}{V/\nu_2} \quad (4.95)$$

where U and V are independent random variables having chi-squared distributions with ν_1 and ν_2 degrees of freedom.

Theorem 4.16: F-Distribution

Let U and V be two independent random variables having chi-squared distributions with ν_1 and ν_2 degrees of freedom, respectively. Then the distribution of the random variable $F = \frac{U/\nu_1}{V/\nu_2}$ is given by the density function

$$h(f) = \begin{cases} \frac{\Gamma\left[\frac{\nu_1 + \nu_2}{2}\right] \left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}}}{\Gamma\left(\frac{\nu_1}{2}\right)\Gamma\left(\frac{\nu_2}{2}\right)} \frac{f^{\frac{\nu_1}{2}-1}}{\left(1 + \frac{\nu_1 f}{\nu_2}\right)^{\frac{\nu_1 + \nu_2}{2}}}, & f > 0 \\ 0, & f \leq 0 \end{cases} \quad (4.96)$$

This is known as the F -distribution with ν_1 and ν_2 degrees of freedom. The curve of the F -distribution depends not only on the two parameters ν_1 and ν_2 but also on the order in which we state them.

The statistics table provides the f -value that you are looking for given the various combinations of the degrees of freedom ν_1 and ν_2 . Hence the f -value with 6 and 10 degrees of freedom ν_1 and ν_2 , leaving an area of 0.05 to the right, is

$$f_{0.05} = 3.22$$

Theorem 4.17: The F -Distribution with Degrees of Freedom

Writing $f_\alpha(\nu_1, \nu_2)$ for f_α with ν_1 and ν_2 degrees of freedom, we obtain

$$f_{1-\alpha}(\nu_1, \nu_2) = \frac{1}{f_\alpha(\nu_2, \nu_1)} \quad (4.97)$$

The F -Distribution' pdf with Beta Function

The pdf of the F -distribution for a variable x with numerator degrees of freedom ν_1 and denominator degrees of freedom ν_2 is defined as:

$$f(x; \nu_1, \nu_2) = \frac{1}{B\left(\frac{\nu_1}{2}, \frac{\nu_2}{2}\right)} \left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}} x^{\frac{\nu_1}{2}-1} \left(1 + \frac{\nu_1}{\nu_2}x\right)^{-\frac{\nu_1 + \nu_2}{2}}$$

where $B(a, b)$ is the beta function, which can be defined in terms of the gamma function:

$$B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

the gamma function $\Gamma()$ can be computed easily and available in standard C++ library as `tgamma()`.

The pdf of F -distribution is always greater than or equal to zero for all valid x . The integral of the pdf over its entire domain $(0, \infty)$ is equal to 1.

The F -Distribution' cdf with Incomplete Beta Function

The regularized incomplete beta function, denoted $I_x(a, b)$, is defined as:

$$I_x(a, b) = \frac{B_x(a, b)}{B(a, b)} = \frac{\int_0^x t^{a-1}(1-t)^{b-1} dt}{\int_0^1 t^{a-1}(1-t)^{b-1} dt}$$

where $B_x(a, b)$ is the incomplete beta function, and $B(a, b)$ is the complete beta function.

The cdf of an F -distribution with ν_1 and ν_2 degrees of freedom is given by the following formula:

$$F(x; \nu_1, \nu_2) = P(X \leq x) = I_z\left(\frac{\nu_1}{2}, \frac{\nu_2}{2}\right)$$

where the variable z is defined as

$$z = \frac{\nu_1 x}{\nu_1 x + \nu_2}$$

Using the identity $I_z(a, b) = 1 - I_{1-z}(a, b)$

$$F(x; \nu_1, \nu_2) = 1 - I_{1-z}\left(\frac{\nu_2}{2}, \frac{\nu_1}{2}\right)$$

where $1 - z$ is:

$$1 - z = 1 - \frac{\nu_1 x}{\nu_1 x + \nu_2} = \frac{\nu_2}{\nu_1 x + \nu_2}$$

The F -Distribution Connection to the F -Statistic

The F -distribution is often derived as the ratio of two chi-squared distributions, which is the basis for the F -test. The F -statistic is calculated as

$$F = \frac{\text{Variance}_{\text{between}}}{\text{Variance}_{\text{within}}}$$

or

$$F = \frac{\text{Mean Square}_{\text{between}}}{\text{Mean Square}_{\text{within}}}$$

In the context of ANOVA, this can be written as:

$$F = \frac{\text{SS}_{\text{between}}/df_{\text{between}}}{\text{SS}_{\text{within}}/df_{\text{within}}}$$

The F -Distribution with Two Sample Variances

Suppose that random samples of size n_1 and n_2 are selected from two normal populations with variances σ_1^2 and σ_2^2 , respectively. We know that

$$\begin{aligned}\chi_1^2 &= \frac{(n_1 - 1)S_1^2}{\sigma_1^2} \\ \chi_2^2 &= \frac{(n_2 - 1)S_2^2}{\sigma_2^2}\end{aligned}$$

are random variables having chi-squared distributions with $\nu_1 = n_1 - 1$ and $\nu_2 = n_2 - 1$ degrees of freedom. Furthermore, since the samples are selected at random, we are dealing with independent random variables. Furthermore, since the samples are selected at random, we are dealing with independent random variables.

Theorem 4.18: F-Distribution with Two Sample Variances

If S_1^2 and S_2^2 are the variances of independent random samples of size n_1 and n_2 taken from normal populations with variances σ_1^2 and σ_2^2 , respectively, then

$$F = \frac{S_1^2/\sigma_1^2}{S_2^2/\sigma_2^2} = \frac{\sigma_2^2 S_1^2}{\sigma_1^2 S_2^2} \quad (4.98)$$

has an F -distribution with $v_1 = n_1 - 1$ and $v_2 = n_2 - 1$ degrees of freedom.

The F -distribution is used in two-sample situations to draw inferences about the population variances. However, the F -distribution can also be applied to many other types of problems involving sample variances. In fact, the F -distribution is called the variance ratio distribution. The F -distribution plays an important role in the analysis of variance.

Three things one must bear in mind, regarding these fundamental sampling distributions:

1. One cannot use Central Limit Theorem unless σ is known. When σ is not known, it should be replaced by s , the sample standard deviation, in order to use the Central Limit Theorem.
2. The T statistic is not a result of the Central Limit Theorem and x_1, x_2, \dots, x_n must come from a $n(x; \mu, \sigma)$ distribution in order for $\frac{\bar{x} - \mu}{s/\sqrt{n}}$ to be a t -distribution; s is, of course, merely an estimate of σ .
3. While the notion of degrees of freedom is new at this point, the concept should be very intuitive, since it is reasonable that the nature of the distribution of S and also t should depend on the amount of information in the sample x_1, x_2, \dots, x_n .

ix. Weibull Distribution and Hazard Rates

For a random variable T , representing the lifetime of an individual or a component, an interesting quantity is the instantaneous chance of failure at any time, given that the component has been operating without failure up to time x . This can be expressed as

$$h(x) = \lim_{\delta \rightarrow 0} \frac{1}{\delta} P(T \in [x, x + \delta] | T > x)$$

Alternatively, by using the conditional probability and noticing that the pdf $f(x)$ satisfies $f(x)\delta \approx P(x \leq T < x + \delta)$ for small δ , we can express the above as

$$h(x) = \frac{f(x)}{1 - F(x)} \quad (4.99)$$

Here the function $h(\cdot)$ is called the hazard rate, and it is a common method of viewing the distribution for lifetime random variables T . In fact, we can reconstruct the cdf $F(x)$ by

$$1 - F(x) = e^{- \int_0^x h(t) dt} \quad (4.100)$$

Hence, every continuous non-negative random variable can be described uniquely by its hazard rate. The Weibull distribution is naturally defined through the hazard rate by considering hazard rate functions that have a specific simple form. It is a distribution with

$$h(x) = \lambda x^{\alpha-1} \quad (4.101)$$

where λ is positive and α takes on any real value. Notice that the parameter α gives the Weibull distribution different modes of behavior. If $\alpha = 1$ then the hazard rate is constant, in which case the Weibull distribution is actually an exponential distribution with rate λ . If $\alpha > 1$, then the hazard rate increases over time. This depicts a situation of "aging components", i.e. the longer a component has lived, the higher the instantaneous chance of failure. This is sometimes called Increasing Failure Rate (IFR). Conversely, $\alpha < 1$ depicts a situation where the longer a component has lasted, the lower the chance of it failing (as it perhaps the case with totalitarian political regimes). This is sometimes called Decreasing Failure Rate (DFR).

Based on Eq. (4.108) and using Eq. (4.107), we obtain the cdf and pdf

$$\begin{aligned} F(x) &= 1 - e^{-\frac{\lambda}{\alpha} x^\alpha} \\ f(x) &= \lambda x^{\alpha-1} e^{-\frac{\lambda}{\alpha} x^\alpha} \end{aligned} \tag{4.102}$$

where the bijection from λ to θ is

$$\begin{aligned} \lambda &= \alpha \theta^{-\alpha} \\ \theta &= \left(\frac{\alpha}{\lambda} \right)^{\frac{1}{\alpha}} \end{aligned} \tag{4.103}$$

In this case, θ is called the scale parameter and α is the shape parameter.

x. Cauchy Distribution

Also known as the Loretz distribution, this is a distribution without mean.

Definition 4.38: Cauchy Distribution

At first glance, a plot of the pdf looks very similar to the normal distribution. However, it is fundamentally different as its mean and standard deviation are undefined. The pdf of the Cauchy distribution is given by

$$f(x) = \frac{1}{\pi \gamma \left(1 + \left(\frac{x-x_0}{\gamma} \right)^2 \right)} \tag{4.104}$$

where x_0 is the location parameter at which the peak is observed and γ is the scale parameter.

The physical example of Cauchy distributed random variable is like this: consider a drone hovering stationary in the sky at unit height. A pivoting laser is attached to its undercarriage, which pivots back and forth as it shoots pulses at the ground. At any point the laser fires, it makes an angle θ from the vertical ($-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$).

Since the laser fires at a high frequency as it is pivoting, we can assume that the angle θ is distributed uniformly on $[-\frac{\pi}{2}, \frac{\pi}{2}]$. For each shot from the laser, a point can be measured, X , horizontally on the ground from the point above which the drone is hovering. We can now

consider the horizontal measurement as a new random variable, X . Hence the cdf is

$$\begin{aligned} F_X(x) &= P(\tan(\theta) \leq x) = P(\theta \leq \arctan(x)) \\ &= F_\theta(\arctan(x)) \\ &= \begin{cases} 0, & \arctan(x) \leq -\frac{\pi}{2} \\ \frac{1}{\pi}\arctan(x), & \arctan(x) \in (-\frac{\pi}{2}, \frac{\pi}{2}) \\ 1, & \arctan(x) \geq \frac{\pi}{2} \end{cases} \end{aligned} \quad (4.105)$$

Now since it always holds that $\arctan(x) \in (-\frac{\pi}{2}, \frac{\pi}{2})$ we can obtain the density by taking the derivative of $\frac{1}{\pi}\arctan(x)$ which evaluates to

$$f(x) = \frac{1}{\pi(1+x^2)}$$

This is a special case of the more complicated density Eq. (4.111), with x_0 and $\gamma = 1$. Importantly, the expectation integral,

$$\int_{-\infty}^{\infty} xf(x) dx$$

is not defined since each of the one-sided improper integrals does not converge.

xi. Bivariate Normal

One of the most ubiquitous families of multivariate distributions is the multivariate normal distribution. Similarly to the fact that a scalar (univariate) normal distribution is parametrized by the mean μ and the variance σ^2 , a multivariate normal distribution is parametrized by the mean vector μ_X and the covariance matrix Σ_X .

Definition 4.39: Standard Multivariate Normal Distribution

For the standard multivariate normal distribution having $\mu_X = \mathbf{0}$ mean and $\Sigma_X = I$, the pdf for the random vector $\mathbf{X} = (X_1, \dots, X_n)$ is

$$f(x) = (2\pi)^{-\frac{n}{2}} e^{-\frac{1}{2}x^T x} \quad (4.106)$$

Now, in general, using an affine transformation, it can be shown that for arbitrary μ_X and Σ_X (positive definite)

$$f(\mathbf{x}) = |\Sigma_X|^{-\frac{1}{2}} (2\pi)^{-\frac{n}{2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_X)^T \Sigma_X^{-1} (\mathbf{x}-\mu_X)} \quad (4.107)$$

where $|\Sigma_X|$ is the determinant of Σ_X .

Definition 4.40: Bivariate Normal Distribution

From Eq. (4.114) In the case of $n = 2$, this becomes the bivariate normal distribution with a density represented as

$$\begin{aligned} f_{XY}(x, y; \sigma_X, \sigma_Y, \mu_X, \mu_Y, \rho) &= \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \\ &\times \exp \left[-\frac{1}{2(1-\rho^2)} \right. \\ &\left. \left(\frac{(x-\mu_X)^2}{\sigma_X^2} - \frac{2\rho(x-\mu_X)(y-\mu_Y)}{\sigma_X\sigma_Y} + \frac{(y-\mu_Y)^2}{\sigma_Y^2} \right) \right] \end{aligned} \quad (4.108)$$

Here the elements of the mean and covariance matrix are

$$\boldsymbol{\mu}_X = \begin{bmatrix} \mu_X \\ \mu_Y \end{bmatrix}$$

$$\boldsymbol{\Sigma}_Y = \begin{bmatrix} \sigma_X^2 & \sigma_X\sigma_Y\rho \\ \sigma_X\sigma_Y\rho & \sigma_Y^2 \end{bmatrix}$$

Note that $\rho \in (-1, 1)$ is the correlation coefficient.

[SI*] To compute the probability density function, cumulative density function, mean, variance, and moment generating function for the continuous distributions above we can use these functions in SymIntegration:

uniformpdf(x,a,b)
uniformcdf(x,a,b)
uniformmgf(x,a,b)
uniformmean(x,a,b)
uniformvar(x,a,b)

normalpdf(x, μ , σ)
normalcdf(x, μ , σ)
normalmgf(x, μ , σ)
normalmean(x, μ , σ)
normalvar(x, μ , σ)

gammapdf(x, α , β)
gammacdf(x, α , β)
gammamgf(x, α , β)
gammamean(x, α , β)
gammavar(x, α , β)

exponentialpdf(x, λ)
exponentialcdf(x, λ)
exponentialmgf(x, λ)
exponentialmean(x, λ)
exponentialvar(x, λ)

```
betapdf(x,α, β)
```

```
betacdf(x,α, β)
```

```
betamgf(x,α, β)
```

```
betamean(x,α, β)
```

```
betavar(x,α, β)
```

```

g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Continuous Distributions ]# ./main

uniformpdf(1;1,5) = 0.25
uniformcdf(3;1,5) = 0.5
uniformmgf(x;1,5) = 0.25*e^(5*t)*t^(-1)-0.25*e^t*t^(-1)
uniformmean(x;1,5) = 3
uniformvar(x;1,5) = 1.33333
normalpdf(362;300,50) = 0.00369875
normalcdf(362;300,50) = 0.892512
normalmgf(362;300,50) = e^(300*t+1250*t^(2))
normalmean(362;300,50) = 300
normalvar(362;300,50) = 2500
gammapdf(1,2,0.2) = 0.168449
gammacdf(1,2,0.2) = 0.959572
gammamgf(1,2,0.2) = (-0.2*t+1)^(-2)
gammamean(1,2,0.2) = 0.4
gammavar(1,2,0.2) = 0.08
exponentialpdf(8,0.2) = 0.0403793
exponentialcdf(8,0.2) = 0.798103
exponentialmgf(8,0.2) = (-5*t+1)^(-1)
exponentialmean(8,0.2) = 5
exponentialvar(8,0.2) = 25
betapdf(0.8,3,2) = 1.536
betacdf(0.8,3,2) = 0.8192
betamgf(0.8,3,2) = 0.6*t+0.2*t^(2)+0.047619*t^(3)+0.00892857*t^(4)+0.00138889*t^(5)+1
betamean(0.8,3,2) = 0.6
betavar(0.8,3,2) = 0.04

Time taken by function: 321002 microseconds

```

Figure 4.9: The functions in SymIntegration to compute the probability density function, cumulative distribution function, mean, variance, and moment generating function for several continuous probability distributions (SymIntegration/Examples/Test SymIntegration Continuous Distributions/main.cpp).

[SI*] To generate random numbers that have certain continuous distributions we can use these functions in SymIntegration:

vrandn_normal(μ, σ, n)

vrandn_exponential(λ, n)

```
vrandn_chisquared(v, n)
vrandn_fdist(v1, v2, n)
vrandn_tdist(v, n)
vrandn_erlang(k, λ, n)
vrandn_gamma(α, β, n)
vrandn_beta(α, β, n)
```

with n as the number of random number we want to generate, the result will be in `std::vector<double>`.

Some explanations:

1. We used to call **Boost** to compute the pdf and cdf of some continuous distributions, but now we are releasing dependencies on **Boost** to make the program runs faster. For example, we used to use this code to compute the pdf of Beta distribution:

```
#include <boost/math/distributions/beta.hpp> //Faster computing of
beta cdf and pdf

...
Symbolic betapdf(double x, double alpha, double beta)
{
    boost::math::beta_distribution<> my_beta(alpha,beta);
    double pdf_value = boost::math::pdf(my_beta,x);

}
...

```

Code 40: `src/statistics.cpp`

Now we are using this code to compute the pdf of Beta distribution:

```
...
double betapdf(double x, double alpha, double beta)
{
if (x < 0.0 || x > 1.0 || alpha <= 0.0 || beta <= 0.0)
{
    // Handle invalid input: PDF is 0 outside [0,1] or parameters are
    // invalid
    return 0.0;
}
else
{
    // Calculate log of the Beta function using log-Gamma functions
    double log_beta_function = std::lgamma(alpha) + std::lgamma(beta)
        - std::lgamma(alpha + beta);
```

```

    // Calculate log of the numerator
    double log_numerator = (alpha - 1.0) * std::log(x) + (beta -
        1.0) * std::log(1.0 - x);

    // Calculate the log of the PDF
    double log_pdf = log_numerator - log_beta_function;

    // Return the exponentiated value
    return std::exp(log_pdf);
}

}

...

```

Code 41: *src/statistics.cpp*

we can compute the pdf of Beta distribution by using the **lgamma** function from the **<cmath>** header to compute the logarithm of the Gamma function, which is more numerically stable for large values.

2. To generate random number of certain distribution, we make use of **std** library that already have a lot of continuous and discrete distributions covered.

First thing needed is to create / obtain a non-deterministic seed for the random number engine. Two good options currently are:

std::random_device

(create a non-deterministic seed if available and entropy > 0)

or

std::chrono::system_clock::now().time_since_epoch().count()

(provides a more robust seed than a fixed value)

Then we will use **std::mt19937** (Mersenne Twister engine) to generate the random number.

Here is an example of a function in **src/statistics.cpp** to generate a random number that has normal distribution:

```

...
std::vector<double> vrandn_normal(double mu, double sigma, int n)
{
    std::chrono::system_clock::now().time_since_epoch().count()

    std::default_random_engine generator(
        std::chrono::system_clock::now().time_since_epoch().count());

    std::vector<double> vec;
    std::normal_distribution<double> distribution(mu, sigma);

```

```
for(int i=1; i<n; i++)
{
    vec.push_back(static_cast<double>(distribution(generator)))
    ;
}
return vec;
}

...
```

Code 42: *src/statistics.cpp*

3. The C++ Standard Library does not directly offer a **std::beta_distribution**. However, a Beta distribution can be simulated using two independently generated Gamma distributions.

The relationship is as follows:

If X is a random variable from a Gamma distribution with shape parameter α and scale parameter 1 ($\text{Gamma}(\alpha, 1)$), and Y is a random variable from a Gamma distribution with shape parameter β and scale parameter 1 ($\text{Gamma}(\beta, 1)$), then the random variable

$$Z = \frac{X}{X + Y}$$

follows a Beta distribution with shape parameters α and β (denoted $\text{Beta}(\alpha, \beta)$).

xii. Generate Gamma Distributed Random Number in SymIntegration and Plot the Histogram with Hamzstplot

We will demonstrate how to generate random number with Gamma distribution then plot the data as histogram with Hamzstplot.

We will create a C++ source file **main.cpp**, and a Makefile (or compile it manually).

```
#include <iostream>
#include "symintegrationc++.h"
#include <bits/stdc++.h>
#include <cmath>

#include <chrono>

using namespace std::chrono;
using namespace std;
using namespace SymbolicConstant;

int main(void)
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    std::vector<double> x = vrandn_gamma(2, 1.5, 50);
    for(int i=0;i<50;i++)
    {
        cout << x[i] << endl;
    }

    // Get ending timepoint
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "\nTime taken by function: " << duration.count() << " microseconds"
        << endl;

    return 0;
}
```

Code 43: *main.cpp*

```
CFLAGS = -ggdb
DEFINES = -DDEBUGGA
INCLUDES =
LIBS = -lstdc++ -lsymintegration
MAIN = main.o
```

```
CC=g++  
  
.cc.o:  
$(CC) -c $(CFLAGS) $(DEFINES) $(INCLUDES) $<  
  
all:: main  
  
gnuplot_i.o:  
main.o: main.cpp  
  
main: $(MAIN)  
$(CC) -o $@ $(CFLAGS) $(MAIN) $(LIBS)  
  
clean:  
rm -f $(MAIN) main
```

Code 44: *Makefile*

From the current working directory containing the **main.cpp** and the makefile, type:
make
./main

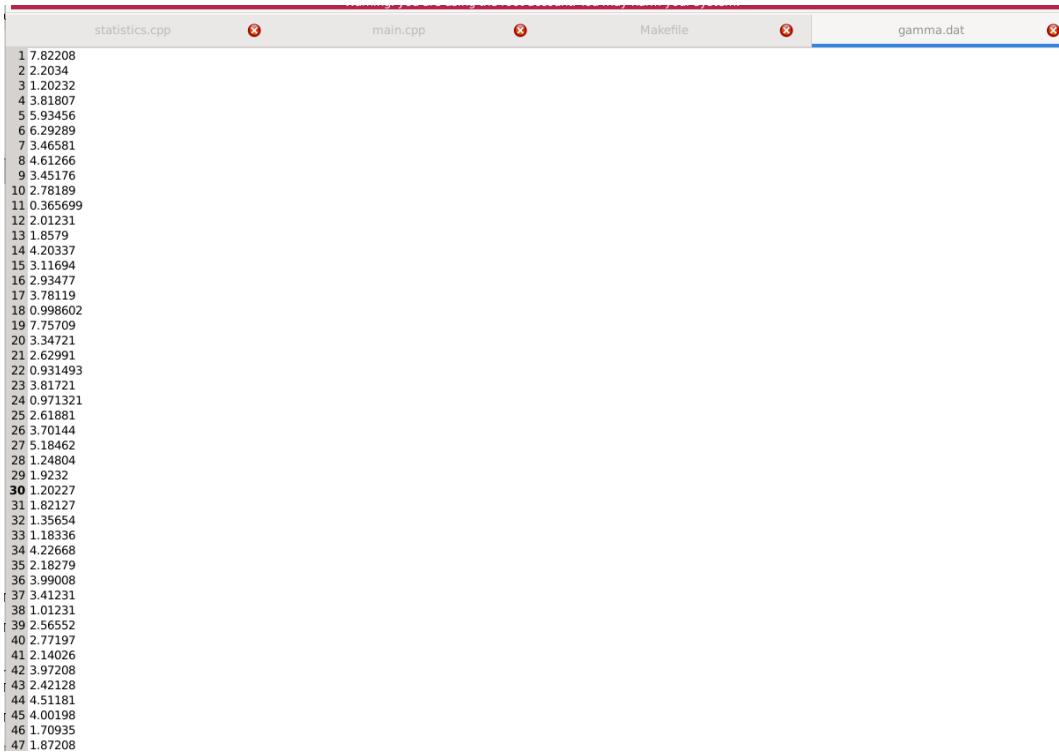
to save the output you can also type this:
./main > gamma.dat

If you are still learning about Makefile, or feel more confident with manual compiling, then instead of **make** you can type this:
g++ -o main main.cpp -lsymintegration
./main

```
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Generate Random Number from Gamma Distribution ]# ./main
1.49028
4.18471
1.345
0.465866
1.34767
2.26934
1.07946
4.17375
1.48454
3.77183
2.87478
0.318233
3.1834
1.90781
3.69633
1.51896
4.32441
2.56319
6.54034
2.51703
3.25444
4.40127
1.68972
0.734017
3.02267
6.22162
4.24151
1.61816
5.48238
3.12266
1.6428
5.69689
1.54791
3.17954
5.72097
5.6629
4.99465
2.66944
6.83603
2.46991
1.13762
2.35658
0.503396
3.15512
5.12802
4.61204
4.52307
5.98776
4.05892
0

Time taken by function: 1008 microseconds
```

Figure 4.10: The random number generation that has Gamma distribution with parameters $\alpha = 2, \beta = 1.5$, we generate 50 random numbers. (*SymIntegration/Examples/Test SymIntegration Generate Random Number from Gamma Distribution/main.cpp*).



```

1 7.82208
2 2.2034
3 1.20232
4 3.81807
5 5.93456
6 6.29289
7 3.46581
8 4.61266
9 3.45176
10 2.78189
11 0.365699
12 2.01231
13 1.8579
14 4.20337
15 3.11694
16 2.93477
17 3.78119
18 0.998602
19 7.75709
20 3.34721
21 2.62091
22 0.931493
23 3.81721
24 0.971321
25 2.61881
26 3.70144
27 5.18462
28 1.24804
29 1.9232
30 1.20227
31 1.82127
32 1.35654
33 1.18336
34 4.22668
35 2.18279
36 3.99008
37 3.41231
38 1.01231
39 2.56552
40 2.77197
41 2.14026
42 3.97208
43 2.42128
44 4.51181
45 4.00198
46 1.70935
47 1.87208

```

Figure 4.11: The random number generation that has been saved in `gamma.dat`. (*SymIntegration/Examples/Test SymIntegration Generate Random Number from Gamma Distribution/main.cpp*).

Now, if we want to plot the histogram, we will need to add **Hamzstplot** library when compiling and add few more lines of code.

We will create again a C++ source file **main.cpp**, and a Makefile (or compile it manually).

```

#include <cmath>
#include <hamzstplot/hamzstplot.h>
#include <random>
#include "symintegrationc++.h"

using namespace std;

int main() {
    using namespace hamzstplot;

    std::vector<double> x = vrandn_gamma(2, 1.5, 1000);

    auto h = hist(x);
    std::cout << "Histogram with " << h->num_bins() << " bins" << std::endl;

    show();

    return 0;
}

```

```
}
```

Code 45: *main.cpp*

```
CFLAGS = -ggdb
DEFINES = -DDEBUGGA
INCLUDES =
LIBS = -lstdc++ -lhamzstplot -lsymintegration
MAIN = main.o
CC=g++

.cc.o:
$(CC) -c $(CFLAGS) $(DEFINES) $(INCLUDES) $<

all:: main

gnuplot_i.o:
main.o: main.cpp

main: $(MAIN)
$(CC) -o $@ $(CFLAGS) $(MAIN) $(LIBS)

clean:
rm -f $(MAIN) main
```

Code 46: *Makefile*

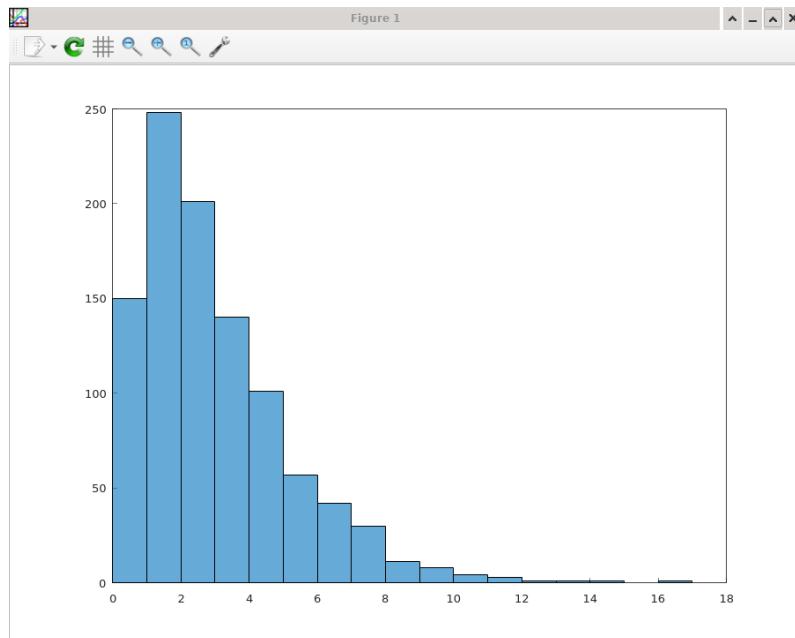


Figure 4.12: The histogram of 1000 random numbers that has been generated with Gamma distribution of parameters $\alpha = 2, \beta = 1.5$. (Hamzstplot/Examples with Makefile/Statistics/Plot 2D Histogram of Random Number/main.cpp).

V. STATISTICAL INFERENCE

[SI*] The action of statistical inference involves using mathematical techniques to make conclusions about unknown population parameters based on collected data. The field of statistical inference employs a variety of stochastic models to analyze and put forward efficient methods for carrying out such analyses.

Analysis and methods of statistical inference can be categorized as either frequentist (also known as classical) or Bayesian. The former is based on the assumption that population parameters of some underlying distribution, or probability law, exist and are fixed, but are yet unknown. The process of statistical inference then deals with making conclusions about these parameters based on sampled data. In the latter Bayesian case, it is only assumed that there is a prior distribution of the parameters. In this case, the key process deals with analyzing a posterior distribution (of the parameters)-an outcome of the inference process.

In general, a statistical inference process involves data, a model, and analysis. The data is assumed to be comprised of random samples from the model. The goal of the analysis is then to make informed statements about population parameters of the model based on the data. Such statements typically take one of the following forms:

1. Point Estimation

Determination of a single value (or vector of values) representing a best estimate of the parameter / parameters. In this case, the notion of "best" can be defined in different ways.

2. Confidence Intervals

Determination of a range of values where the parameter lies. Under the model and the statistical process used, it is guaranteed that the parameter lies within this range with a pre-specified probability.

3. Hypothesis Tests

The process of determining if the parameter lies in a given region, in the complement of that region, or fails to take on a specific value. Such tests often represent a scientific hypothesis in a very natural way

[SI*] A Random Sample

When carrying out (frequentist) statistical inference, we assume there is some underlying distribution $F(x, \theta)$ from which we are sampling, where θ is the scalar or vector-valued unknown parameter we wish to know. Importantly, we assume that each observation is statistically independent and identically distributed as the rest. That is, from a probabilistic perspective, the observations are taken as independent and identically distributed (i.i.d.) random variables. In mathematical statistics language, this is called a random sample. We denote the random variables of the observations by X_1, \dots, X_n and their respective values x_1, \dots, x_n .

Typically, we compute statistics from the random sample. For example, two common standard statistics include the sample mean and sample variance. However, we can model

these statistics as random variables

$$\begin{aligned}\bar{X} &= \frac{1}{n} \sum_{i=1}^n X_i \\ S^2 &= \frac{1}{n-1} \sum_{i=1}^n (X_i - \bar{X})^2\end{aligned}\tag{4.109}$$

Note that for S^2 , the denominator is $n - 1$ to make S^2 an unbiased estimator.

In general, the phrase statistic implies a quantity calculated based on the sample. When working with data, the sample mean and sample variance are nothing but numbers computed from our sample observations. However, in the statistical inference paradigm, we associate random variables to these values, since they themselves are functions of the random sample.

To illustrate the fact that \bar{X} and S^2 are random variables, assume we have sampled data from certain distribution (e.g. exponential). If we collect $n = 10$ observations, then the sample mean and sample variance are random variables. The point to see is that \bar{X} and S^2 are themselves random variables with underlying distributions.

i. Sampling from a Normal Population

[SI*] It is often assumed that the distribution $F(x; \theta)$ is a normal distribution, and hence $\theta = (\mu, \sigma^2)$.

This assumption is called the normality assumption, and is sometimes justified due to the central limit theorem. Under the normality assumption, the distribution of the random variables \bar{X} and S^2 as well as transformations of them are well known. The following three distributional relationships play a key role:

$$\begin{aligned}\bar{X} &\text{ Normal} \left(\mu, \frac{\sigma^2}{n} \right) \\ \frac{(n-1)S^2}{\sigma^2} &\chi_{n-1}^2 \\ T := \frac{\bar{X} - \mu}{S/\sqrt{n}} &t_{n-1}\end{aligned}\tag{4.110}$$

Here, " " denotes as "distributed as," and implies that the statistics on the left hand side of the " " symbols are distributed according to the distribution on the right hand side. The notation χ_{n-1}^2 and t_{n-1} denotes a chi-squared and t -distribution, respectively, each with $n - 1$ degrees of freedom. The chi-squared distribution is a gamma distribution with parameters $\alpha = \frac{n}{2}$ and $\beta = 2$, some textbooks, e.g. [7], use another parameter thus state that $\lambda = \frac{1}{2}$ because $\lambda = \frac{1}{\beta}$, for Gamma distribution in this book we use parameters α and β .

ii. The Central Limit Theorem

[SI*] The Central Limit Theorem (CLT) is one of the most fundamental results of probability and statistics. It has several versions and many generalizations, they all have one thing in common: summation of a large number of random quantities, each with finite variance, yield a sum that is approximately normally distributed. This is the main reason that the normal distribution is ubiquitous in nature and present throughout the universe.

Theorem 4.19: Central Limit Theorem

If \bar{X} is the mean of a random sample of size n taken from a population with mean μ and finite variance σ^2 , then the limiting form of the distribution of

$$Z = \frac{\bar{X} - \mu}{\sigma / \sqrt{n}}$$

as $n \rightarrow \infty$, is the standard normal distribution $n(z; 0, 1)$.

iii. Point Estimation

[SI*]

iv. Confidence Interval as a Concept

[SI*]

v. Hypothesis Tests Concepts

[SI*]

vi. Bayesian Statistics

[SI*]

VI. LINEAR REGRESSION AND CORRELATION

i. Pearson's Correlation

[SI*] In Darrell Huff's book **How to Lie with Statistics**, he refers to a study that claimed that non-smoking college students gained better grades than students who smoked. The study concluded that 'smoking leads to poor college grades.' Huff questioned the findings of the study, arguing that instead of smoking being the cause of poor grades, perhaps 'low grades depressed students and caused them to smoke,' or maybe 'students with lower grades were more sociable and therefore more likely to smoke.' Measuring the relationship between two sets of data is a useful tool to establish how one variable changes with another, but it is limited; it does not tell you why this relationship exists. the data in the study clearly showed a relationship between smoking and poor grades, but it does not explain why this relationship exists.

Data that consists of measurements of two variables taken from each individual in a sample is called bivariate data; the relationship between the two variables is referred to as the correlation.

For bivariate data we try to classify one of two variables as 'independent' and the other as 'dependent.' The independent variable is the one that can be controlled by the person conducting the experiment or study; it is hypothesised to 'cause' some kind of effect to the dependent variable. the dependent variable is the variable that is just observed without being controlled, and is supposed to show the 'effect.'

[SI*] The line of best fit on a scatter diagram is drawn to give the best representation of the correlation between the two variables. The line of best fit is the one that has an approximately even spread of the data points either side of it.

[SI*] Pearson's product moment correlation coefficient

Pearson product moment correlation coefficient (PMCC) is a number, usually denoted by r , which can take any value between -1 and $+1$. Its sign indicates the type of correlation, and its magnitude (size) indicates the strength of correlation.

1. $r = +1$ means that there is perfect positive correlation
2. $r = 0$ means that there is no correlation
3. $r = -1$ means that there is perfect negative correlation

If $-0.5 < r < 0.5$, it is difficult to draw a line of best fit that has any meaning, because the data is just too scattered.

The formula to compute the PMCC is

$$r = \frac{S_{xy}}{S_x S_y} \quad (4.111)$$

where S_{xy} is the covariance of x and y , S_x is the standard deviation of the x values, and S_y is the standard deviation of the y values.

The formula that act as the foundation blocks are:

$$\bar{x} = \frac{\sum x}{n} \quad (4.112)$$

$$\bar{y} = \frac{\sum y}{n} \quad (4.113)$$

$$S_x = \sqrt{\frac{\sum x^2}{n} - (\bar{x})^2} \quad (4.114)$$

$$S_y = \sqrt{\frac{\sum y^2}{n} - (\bar{y})^2} \quad (4.115)$$

$$S_{xy} = \frac{\sum xy}{n} - (\bar{x})(\bar{y}) \quad (4.116)$$

with n as the number of observations / data points.
 [SI*] The formula for the regression line is

$$y - \bar{y} = \frac{S_{xy}}{(S_x)^2} (x - \bar{x}) \quad (4.117)$$

ii. Linear Regression

[SI*] Often, in practice, we need to solve problems involving sets of variables when it is known that there exists some inherent relationship among the variables. For example, in an industrial situation it may be known that the tar content in the outlet stream in a chemical process is related to the inlet temperature. It may be of interest to develop a method of prediction, that is, a procedure for estimating the tar content for various levels of the inlet temperature from experimental information. In this case the tar content is the dependent variable, or response, and the inlet temperature is the independent variable, or regressor. A reasonable form of a relationship between the response Y and the regressor x is the linear relationship

$$Y = \beta_0 + \beta_1 x$$

with β_0 as the intercept and β_1 as the slope. If the relationship is exact, then it is a deterministic relationship between two scientific variables and there is no random or probabilistic component to it. However, the relationship is not deterministic (i.e., a given x does not always give the same value for Y).

The concept of regression analysis deals with finding the best relationship between Y and x , quantifying the strength of that relationship, and using methods that allow for prediction of the response values given values of the regressor x .

[SI*] In most application of regression, the linear equation $Y = \beta_0 + \beta_1 x$ is an approximation that is a simplification of something unknown and much more complicated.

Thus, an analysis of the relationship between Y and x requires the statement of a statistical model. The model include the set

$$\{(x_i, y_i); i = 1, 2, \dots, n\}$$

of data involving n pairs of (x, y) values. The value y_i depends on x_i , via a linear structure that also has the random component involved.

The basis for the use of a statistical model relates to how the random variable Y moves with x and the random component.

Definition 4.41: Simple Linear Regression Model

the response Y is related to the independent variable x through the equation

$$Y = \beta_0 + \beta_1 x + \epsilon$$

β_0 and β_1 are unknown intercept and slope parameters, respectively, and ϵ is a random variable that is assumed to be distributed with $E(\epsilon) = 0$ and $\text{Var}(\epsilon) = \sigma^2$. The quantity σ^2 is often called the error variance or residual variance.

Definition 4.42: The Fitted Regression Line

Suppose we denote the estimates b_0 for β_0 and b_1 for β_1 . Then the estimated or fitted regression line is given by

$$\hat{y} = b_0 + b_1 x$$

where \hat{y} is the predicted or fitted value. Obviously, the fitted line is the estimate of the true regression line.

Definition 4.43: Residual: Error in Fit

Given a set of regression data $\{(x_i, y_i); i = 1, 2, \dots, n\}$ and a fitted model, $\hat{y}_i = b_0 + b_1 x_i$, the i th residual ϵ_i , is given by

$$\epsilon_i = y_i - \hat{y}_i, \quad i = 1, 2, \dots, n$$

Definition 4.44: Estimating the Regression Coefficients

In order to find b_0 and b_1 , the estimates of β_0 and β_1 , so that the sum of squares of the residuals is a minimum. The residual sum of squares is often called the sum of squares of the errors about the regression line and is denoted by SSE . This minimization procedure for estimating the parameters is called the method of least squares.

Given the sample $\{(x_i, y_i); i = 1, 2, \dots, n\}$, the least squares estimates b_0 and b_1 of the regression coefficients β_0 and β_1 are computed from the formulas

$$b_1 = \frac{n \sum_{i=1}^n x_i y_i - (\sum_{i=1}^n x_i) (\sum_{i=1}^n y_i)}{n \sum_{i=1}^n x_i^2 (\sum_{i=1}^n x_i)^2} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

and

$$b_0 = \frac{\sum_{i=1}^n y_i - b_1 \sum_{i=1}^n x_i}{n}$$

Thus, the estimated regression line is given by

$$\hat{y} = b_0 + b_1 x$$

iii. Simple Regression Line Plot and Pearson's Correlation Computation

Suppose we have a bivariate data as follow Compute the Pearson's correlation (PMCC), then plot

Distance, x (km)	4	8	5	10	6
Time, y (minutes)	15	35	12	40	24

Table 4.2: Distance travelled to school and travel time

the data on a scatter diagram and draw the line of best fit.

Solution:

We will have

$$\begin{aligned}\bar{x} &= \frac{\sum x}{n} = 6.6 \\ \bar{y} &= \frac{\sum y}{n} = 25.2 \\ S_x &= \sqrt{\frac{\sum x^2}{n} - (\bar{x})^2} = 2.154 \\ S_y &= \sqrt{\frac{\sum y^2}{n} - (\bar{y})^2} = 10.907 \\ S_{xy} &= \frac{\sum xy}{n} - (\bar{x})(\bar{y}) = 22.48\end{aligned}$$

Hence

$$r = \frac{S_{xy}}{S_x S_y} = 0.957$$

The value of r is close to +1, showing that there is strong positive correlation between the distance travelled to school and the time taken.

Now we will compute the regression line formula

$$\begin{aligned}y - \bar{y} &= \frac{S_{xy}}{(S_x)^2} (x - \bar{x}) \\ y - 25.2 &= \frac{22.48}{(2.154)^2} (x - 6.6) \\ y &= 4.85x - 6.78\end{aligned}$$

To plot the bivariate data on a scatter diagram and the regression line we can use Hamzstlab Mathematics, and the backend computation can be done with SymIntegration.

In SymIntegration the function to compute Pearson's correlation is:

rpearson(M,n)

with M as the matrix of size $n \times 2$, the first column of the matrix represents the independent variable, x , while the second column of the matrix represents the dependent variable, y .

The function to compute the regression line is:
regressionline(M,n)
with M as the matrix of size $n \times 2$, the result will be a symbolic function, $y(x)$.

```
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
./main ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Load Vector from textfile and Compute Regression Line |#
W:
[ 4 15]
[ 8 35]
[ 5 12]
[10 40]
[ 6 24]

x bar: 6.6
y bar: 25.2
sum x: 33
sum y: 126
sum xy: 944
sum X^2: 241
sum Y^2: 3770

Pearson's correlation coefficient, r :
0.956835

Regression line, y = 4.84483*x-6.77586

Time taken by function: 7136 microseconds
```

Figure 4.13: The computation of Pearson's correlation (PMCC), r , and the regression line equation with SymIntegration took 7136 microseconds. (*SymIntegration/Examples/Test SymIntegration Load Vector from textfile and Compute Regression Line/main.cpp*).

iv. Multiple Linear Regression

[SI*] In most research problems where regression analysis is applied, more than one independent variable is needed in the regression model. The complexity of most scientific mechanisms is such that in order to be able to predict an important response, a multiple regression model is needed. When this model is linear in the coefficients, it is called a multiple linear regression model. For the case of k independent variables x_1, x_2, \dots, x_k , the mean $Y|x_1, x_2, \dots, x_k$ is given by the multiple linear regression model

$$\mu_{Y|x_1, x_2, \dots, x_k} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

and the estimated response is obtained from the sample regression equation

$$\hat{y} = b_0 + b_1 x_1 + \dots + b_k x_k$$

where each regression coefficient β_i is estimated by b_i from the sample data using the method of least squares.

Definition 4.45: Multiple Linear Regression Model

We obtain the least squares estimators of the parameters $\beta_0, \beta_1, \dots, \beta_k$ by fitting the multiple linear regression model

$$\mu_{Y|x_1, x_2, \dots, x_k} = \beta_0 + \beta_1 x_1 + \dots + \beta_k x_k$$

to the data points

$$\{(x_{1i}, x_{2i}, \dots, x_{ki}, y_i); \quad i = 1, 2, \dots, n \text{ and } n > k\}$$

where y_i is the observed response to the values $x_{1i}, x_{2i}, \dots, x_{ki}$ of the k independent variables x_1, x_2, \dots, x_k . Each observation $(x_{1i}, x_{2i}, \dots, x_{ki}, y_i)$ is assumed to satisfy the following equation.

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \dots + \beta_k x_{ki} + \epsilon_i$$

or

$$\begin{aligned} y_i &= \hat{y}_i + e_i \\ &= b_0 + b_1 x_{1i} + b_2 x_{2i} + \dots + b_k x_{ki} + e_i \end{aligned}$$

where ϵ_i and e_i are the random error and residual, respectively, associated with the response y_i and fitted value \hat{y}_i .

[SI*] As in the case of simple linear regression, it is assumed that the ϵ_i are independent and identically distributed with mean 0 and common variance σ^2 .

In using the concept of least squares to arrive at estimates b_0, b_1, \dots, b_k , we minimize the expression

$$\begin{aligned} SSE &= \sum_{i=1}^n e_i^2 \\ &= \sum_{i=1}^n (y_i - b_0 - b_1 x_{1i} - b_2 x_{2i} - \dots - b_k x_{ki})^2 \end{aligned}$$

Differentiating SSE in turn with respect to b_0, b_1, \dots, b_k and equating to zero, we generate the set of $k + 1$ normal equations for multiple linear regression.

Multiple Linear Regression Parameter Estimation with Matrices

In fitting a multiple linear regression model, particularly when the number of variables exceeds two, a knowledge of matrix theory can facilitate the mathematical manipulations considerably.

Suppose that the experimenter has k independent variables x_1, x_2, \dots, x_k and n observations y_1, y_2, \dots, y_n , each of which can be expressed by the equation

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \dots + \beta_k x_{ki} + \epsilon_i$$

This model essentially represents n equations describing how the response values are generated in the scientific process.

Definition 4.46: General Linear Model

Using matrix notation, we can write the following equation that represents the multiple linear regression equation:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

where

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{21} & \dots & x_{k1} \\ 1 & x_{12} & x_{22} & \dots & x_{k2} \\ \vdots & \vdots & \ddots & & \vdots \\ 1 & x_{1n} & x_{2n} & \dots & x_{kn} \end{bmatrix}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_k \end{bmatrix}, \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

Then the least squares method for estimation of $\boldsymbol{\beta}$, involves finding \mathbf{b} for which

$$SSE = (\mathbf{y} - \mathbf{X}\mathbf{b})^T(\mathbf{y} - \mathbf{X}\mathbf{b})$$

is minimized. This minimization process involves solving for \mathbf{b} in the equation

$$\frac{\partial}{\partial \mathbf{b}}(SSE) = \mathbf{0}$$

The result reduces to the solution of \mathbf{b} in

$$(\mathbf{X}^T \mathbf{X})\mathbf{b} = \mathbf{X}^T \mathbf{y}$$

Notice the nature of the \mathbf{X} matrix. Apart from the initial element, the i th row represents the x -values that give rise to the response y_i . Writing

$$\mathbf{X}^T \mathbf{X} = \begin{bmatrix} n & \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{2i} & \dots & \sum_{i=1}^n x_{ki} \\ \sum_{i=1}^n x_{1i} & \sum_{i=1}^n x_{1i}^2 & \sum_{i=1}^n x_{1i}x_{2i} & \dots & \sum_{i=1}^n x_{1i}x_{ki} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^n x_{ki} & \sum_{i=1}^n x_{ki}x_{1i} & \sum_{i=1}^n x_{ki}x_{2i} & \dots & \sum_{i=1}^n x_{ki}^2 \end{bmatrix}$$

and

$$\mathbf{X}^T \mathbf{y} = \begin{bmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_{1i}y_i \\ \vdots \\ \sum_{i=1}^n x_{ki}y_i \end{bmatrix}$$

If the matrix $\mathbf{X}^T \mathbf{X}$ is not singular, the formula for parameter estimation in multiple linear regression is

$$\mathbf{b} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

where \mathbf{b} is the vector of estimated coefficients, \mathbf{X} is the matrix of independent variables (with a column of ones for the intercept), \mathbf{X}^T is the transpose of \mathbf{X} , and \mathbf{y} is the vector of the dependent variable. This formula uses the method of least squares to find the coefficients that minimize the sum of squared errors, which is the vertical distance between the observed and predicted values.

Model:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

with:

\mathbf{y} : Vector of the dependent variable

\mathbf{X} : Matrix of independent variables (including a column of 1 for the intercept)

$\boldsymbol{\beta}$: Vector of population parameters to be estimated

$\boldsymbol{\epsilon}$: Vector of error terms

Estimation:

$$\mathbf{b} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

with:

\mathbf{b} : Vector of estimated coefficients (intercept and slopes)

\mathbf{X}^T : Transpose of the matrix \mathbf{X} .

$(\mathbf{X}^T \mathbf{X})^{-1}$: The inverse of the matrix product $\mathbf{X}^T \mathbf{X}$.

The formula finds the set of coefficients (\mathbf{b}) that produce a regression line (or plane/hyperplane) that best fits the data. "Best fit" is defined by the least-squares criterion, meaning the sum of the squared differences between the actual y values and the predicted \hat{y} values is minimized. The matrix algebra approach provides a direct way to calculate the coefficients without having to manually solve a system of linear equations for each parameter, which would be necessary for a large number of independent variables.

To evaluate the model we can use R^2 (R-squared) to measure the proportion of the variance in the dependent variable that is predictable from the independent variables. A modified version of R^2 , adjusted R-squared is a better measure when comparing models with different numbers of predictors. Another measure is the p -value, it is often used to determine the statistical significance of the independent variables, indicating whether their relationship with the dependent variable is likely due to chance.

[SI*] Properties of the Least Squares Estimators

The means and variances of the estimators b_0, b_1, \dots, b_k are readily obtained under certain assumptions on the random errors $\epsilon_1, \epsilon_2, \dots, \epsilon_k$ that are identical to those made in the case of simple linear regression. When we assume these errors to be independent, each with mean 0

and variance σ^2 , it can be shown that b_0, b_1, \dots, b_k are, respectively, unbiased estimators of the regression coefficients $\beta_0, \beta_1, \dots, \beta_k$.

In addition, the variances of the b 's are obtained through the elements of the inverse of the A matrix. Note that the off-diagonal elements of $A = X^T X$ represents sums of products of elements in the columns of X , while the diagonal elements of A represents sums of squares of elements in the columns of X .

The inverse matrix, A^{-1} , apart from the multiplier σ^2 , represents the variance-covariance matrix of the estimated regression coefficients.

That is, the elements of the matrix $A^{-1}\sigma^2$ display the variances of b_0, b_1, \dots, b_k on the main diagonal and covariances on the off-diagonal. For example, in a $k = 2$ multiple linear regression problem, we might write

$$(X^T X)^{-1} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix}$$

with the elements below the main diagonal determined through the symmetry of the matrix. Then we can write

$$\begin{aligned} \sigma_{b_i}^2 &= c_{ii}\sigma^2, \quad i = 0, 1, 2 \\ \sigma_{b_i b_j} &= \text{Cov}(b_i, b_j) = c_{ij}\sigma^2, \quad i \neq j \end{aligned}$$

Theorem 4.20: Unbiased Estimate for Multiple Linear Regression

For the linear regression equation

$$y = X\beta + \epsilon$$

an unbiased estimate of σ^2 is given by the error or residual mean square

$$s^2 = \frac{SSE}{n - k - 1}$$

where

$$SSE = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

The error and regression sums of squares take on the same form and play the same role as in the simple linear regression case. In fact, the sum-of-squares identity

$$\sum_{i=1}^n (y_i - \bar{y}_i)^2 = \sum_{i=1}^n (\hat{y}_i - \bar{y}_i)^2 + \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

continues to hold, and we retain our previous notation, namely

$$SST = SSR + SSE$$

with

$$SST = \sum_{i=1}^n (y_i - \bar{y}_i)^2$$

and

$$SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y}_i)^2$$

There are k degrees of freedom associated with SSR , and, as always, SST has $n - 1$ degrees of freedom. Therefore, after subtraction, SSE has $n - k - 1$ degrees of freedom.

[SI*] Analysis of Variance in Multiple Regression

The partition of the total sum of squares into its components, the regression and error sums of squares, plays an important role. An analysis of variance can be conducted to shed light on the quality of the regression equation. A useful hypothesis that determines if a significant amount of variation is explained by the model is

$$H_0 : \beta_1 = \beta_2 = \beta_3 = \cdots = \beta_k = 0$$

The analysis of variance involves an F -test via a table. The test is an upper-tailed test.

Source	Sum of Squares	Degrees of Freedom	Mean Squares	F
Regression	SSR	k	$MSR = \frac{SSR}{k}$	$f = \frac{MSR}{MSE}$
Error	SSE	$n - (k + 1)$	$MSE = \frac{SSE}{n-(k+1)}$	
Total	SST	$n - 1$		

Table 4.3: The analysis of variance table that uses F -test.

Rejection of H_0 implies that the regression equation differs from a constant. That is, at least one regressor variable is important.

Further utility of the mean square error (or residual mean square) lies in its use in hypothesis testing and confidence interval estimation. In addition, the mean square error plays an important role in situations where the scientist is searching for the best from a set of competing models. Many model-building criteria involve the statistic s^2 .

[SI*] Inferences in Multiple Linear regression

A knowledge of the distributions of the individual coefficient estimators enables the experimenter to construct confidence intervals for the coefficients and to test hypotheses about them.

Recall that the b_j ($j = 0, 1, 2, \dots, k$) are normally distributed with mean β_j and variance $c_{jj}\sigma^2$. Thus, we can use the statistic

$$t = \frac{b_j - \beta_{j0}}{s\sqrt{c_{jj}}}$$

with $n - k - 1$ degrees of freedom to test hypotheses and construct confidence intervals on β_j . For example, if we wish to test

$$\begin{aligned} H_0 &: \beta_j = \beta_{j0} \\ H_1 &: \beta_j \neq \beta_{j0} \end{aligned}$$

we compute the above t -statistic and do not reject H_0 if $-t_{\alpha/2} < t < t_{\alpha/2}$, where $t_{\alpha/2}$ has $n - k - 1$ degrees of freedom.

[SI*] Individual t -Tests for Variable Screening

The t -test most often used in multiple regression is the one that tests the importance of individual coefficients (i.e., $H_0 : \beta_j = 0$ against the alternative $H_1 : \beta_j \neq 0$). These tests often contribute to what is termed variable screening, where the analyst attempts to arrive at the most useful model (i.e., the choice of which regressors to use). It should be emphasized here that if a coefficient is found insignificant (i.e., the hypothesis $H_0 : \beta_j = 0$ is not rejected), the conclusion drawn is that the variable is insignificant (i.e., explains an insignificant amount of variation in y), in the presence of the other regressors in the model.

[SI*] Inferences on Mean Response and Prediction

One of the most useful inferences that can be made regarding the quality of the predicted response y_0 corresponding to the values $x_{10}, x_{20}, \dots, x_{k0}$ is the confidence interval on the mean response $\mu_{Y|x_{10}, x_{20}, \dots, x_{k0}}$. We are interested in constructing a confidence interval on the mean response for the set of conditions given by

$$\mathbf{x}_0^T = [1, x_{10}, x_{20}, \dots, x_{k0}]$$

We augment the conditions on the x 's by the number 1 in order to facilitate the matrix notation. Normality in the ϵ_i produces normality in the b_j and the mean and variance are still the same. So is the covariance between b_i and b_j , for $i \neq j$. Hence,

$$\hat{y} = b_0 + \sum_{j=1}^k b_j x_{j0}$$

is likewise normally distributed and is, in fact, an unbiased estimator for the mean response on which we are attempting to attach a confidence interval. The variance of \hat{y}_0 , written in matrix notation simply as a function of σ^2 , $(\mathbf{X}^T \mathbf{X})^{-1}$, and the condition vector \mathbf{x}_0^T is

$$\sigma_{\hat{y}_0}^2 = \sigma^2 \mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0$$

If this expression is expanded for a given case, say $k = 2$, it is readily seen that it appropriately accounts for the variance of the b_j and the covariance of b_i and b_j , for $i \neq j$. After σ^2 is replaced by s^2 , the $100(1 - \alpha)\%$ confidence interval on $\mu_{Y|x_{10}, x_{20}, \dots, x_{k0}}$ can be constructed from the statistic

$$T = \frac{\hat{y}_0 - \mu_{Y|x_{10}, x_{20}, \dots, x_{k0}}}{s \sqrt{\mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0}}$$

which has a t -distribution with $n - k - 1$ degrees of freedom.

Definition 4.47: Confidence Interval for $\mu_{Y|x_{10}, x_{20}, \dots, x_{k0}}$

A $100(1 - \alpha)\%$ confidence interval for the mean response $\mu_{Y|x_{10}, x_{20}, \dots, x_{k0}}$ is

$$\hat{y}_0 - t_{\frac{\alpha}{2}} s \sqrt{\mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0} < \mu_{Y|x_{10}, x_{20}, \dots, x_{k0}} < \hat{y}_0 + t_{\frac{\alpha}{2}} s \sqrt{\mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0}$$

where $t_{\frac{\alpha}{2}}$ is a value of the t -distribution with $n - k - 1$ degrees of freedom.

The quantity $s \sqrt{\mathbf{x}_0^T (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{x}_0}$ is often called the standard error of prediction.

[SI*] As in the case of simple linear regression, we need to make a clear distinction between the confidence interval on a mean response and the prediction interval on an observed response.

The latter provides a bound within which we can say with a preselected degree of certainty that a new observed response will fall.

A prediction interval for a single predicted response y_0 is once again established by considering the difference $\hat{y}_0 - y_0$. The sampling distribution can be shown to be normal with mean

$$\mu_{\hat{y}_0 - y_0} = 0$$

and variance

$$\sigma_{\hat{y}_0 - y_0}^2 = \sigma^2[1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0]$$

Thus, a $100(1 - \alpha)\%$ prediction interval for a single prediction value y_0 can be constructed from the statistic

$$T = \frac{\hat{y}_0 - y_0}{s\sqrt{1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}}$$

which has t -distribution with $n - k - 1$ degrees of freedom.

Definition 4.48: Prediction Interval for y_0

A $100(1 - \alpha)\%$ prediction interval for a single response y_0 is given by

$$\hat{y}_0 - t_{\frac{\alpha}{2}} s \sqrt{1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0} < y_0 < \hat{y}_0 + t_{\frac{\alpha}{2}} s \sqrt{1 + \mathbf{x}_0^T(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_0}$$

where $t_{\frac{\alpha}{2}}$ is a value of the t -distribution with $n - k - 1$ degrees of freedom.

[SI*] Choice of a Fitted Model through Hypothesis Testing

In many regression situations, individual coefficients are of importance to the experimenter. For example, in an economics application, β_1, β_2, \dots might have some particular significance, and then confidence intervals and tests of hypotheses on these parameters would be of interest to the economist.

v. Compute Multiple Linear Regression with SymIntegration

A study was done on a diesel-powered light-duty pickup truck to see if humidity, air temperature, and barometric pressure influence emission of nitrous oxide (in ppm). Emission measurements were taken at different times, with varying experimental conditions. The data are given in table below. The model is

$$\mu_{Y|x_1,x_2,x_3} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$$

or, equivalently,

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \epsilon_i, \quad i = 1, 2, \dots, 20$$

Fit this multiple linear regression model to the given data and then estimate the amount of nitrous oxide emitted for the conditions where humidity is 50%, temperature is 76° F, and barometric pressure is 29.30.

Nitrous Oxide, y	Humidity, x_1	Temp., x_2	Pressure, x_3	Nitrous Oxide, y	Humidity x_1	Temp., x_2	Pressure, x_3
0.90	72.4	76.3	29.18	1.07	23.2	76.8	29.38
0.91	41.6	70.3	29.35	0.94	47.4	86.6	29.35
0.96	34.3	77.1	29.24	1.10	31.5	76.9	29.63
0.89	35.1	68.0	29.27	1.10	10.6	86.3	29.56
1.00	10.7	79.0	29.78	1.10	11.2	86.0	29.48
1.10	12.9	67.4	29.39	0.91	73.3	76.3	29.40
1.15	8.3	66.8	29.69	0.87	75.4	77.9	29.28
1.03	20.1	76.9	29.48	0.78	96.6	78.7	29.29
0.77	72.2	77.7	29.09	0.82	107.4	86.8	29.03
1.07	24.0	67.7	29.60	0.95	54.9	70.9	29.37

Table 4.4: Source: Charles T. Hare, "Light-Duty Diesel Emission Correction Factors for Ambient Conditions," EPA-600/2-77-116. U.S. Environmental Protection Agency.

Solution:

The solution of the set of estimating equations yields the unique estimates

$$\begin{aligned} b_0 &= -3.507778 \\ b_1 &= -0.002625 \\ b_2 &= 0.000799 \\ b_3 &= 0.154155 \end{aligned}$$

Therefore, the regression equation is

$$\hat{y} = -3.507778 - 0.002625x_1 + 0.000799x_2 + 0.154155x_3$$

For 50% humidity, a temperature of 76° F, and a barometric pressure of 29.30, the estimated amount of nitrous oxide emitted is

$$\begin{aligned} \hat{y} &= -3.507778 - 0.002625(50) + 0.000799(76) + 0.154155(29.30) \\ &= 0.9384 \end{aligned}$$

In SymIntegration, to compute the multiple regression model' coefficients we can use this function:

multipleregression(vector<vector<double>&X, vector<vector<double>&y)

with X as the matrix of independent variables, including a column if 1 as the first column for the intercept, and y is the vector of the dependent variable. The computation requires inverse operation, thus it is a bit expensive on the computational side since computing inverse of a matrix requires some works before, such as computing the determinant of the corresponding matrix and the adjugate matrix as well.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Multiple Linear
Regression ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Multiple Linear Regression ]# ./main

Matrix X :
 1.000000    72.400000    76.300000    29.180000
 1.000000    41.600000    70.300000    29.350000
 1.000000    34.300000    77.100000    29.240000
 1.000000    35.100000    68.000000    29.270000
 1.000000    10.700000    79.000000    29.780000
 1.000000    12.900000    67.400000    29.390000
 1.000000     8.300000    66.800000    29.690000
 1.000000    20.100000    76.900000    29.480000
 1.000000    72.200000    77.700000    29.090000
 1.000000    24.000000    67.700000    29.600000
 1.000000    23.200000    76.800000    29.380000
 1.000000    47.400000    86.600000    29.350000
 1.000000    31.500000    76.900000    29.630000
 1.000000    10.600000    86.300000    29.560000
 1.000000    11.200000    86.000000    29.480000
 1.000000    73.300000    76.300000    29.400000
 1.000000    75.400000    77.900000    29.280000
 1.000000    96.600000    78.700000    29.290000
 1.000000    107.400000   86.800000    29.030000
 1.000000    54.900000    70.900000    29.370000

Vector y :
 0.900000
 0.910000
 0.960000
 0.890000
 1.000000
 1.100000
 1.150000
 1.030000
 0.770000
 1.070000
 1.070000
 0.940000
 1.100000
 1.100000
 0.910000
 0.870000
 0.780000
 0.820000
 0.950000
```

Figure 4.14: The computation of the multiple regression model with SymIntegration (*SymIntegration/Examples/Test SymIntegration Multiple Linear Regression/main.cpp*).

```
Multiple regression with matrix algebra:  
b[0] = -3.507778  
b[1] = -0.002625  
b[2] = 0.000799  
b[3] = 0.154155  
(X^{T} X)^{-1} X^{T} y:  
-3.507778  
-0.002625  
0.000799  
0.154155  


| y        | y estimated |
|----------|-------------|
| 0.900000 | 0.861376    |
| 0.910000 | 0.963638    |
| 0.960000 | 0.971276    |
| 0.890000 | 0.966530    |
| 1.000000 | 1.117988    |
| 1.100000 | 1.042824    |
| 1.150000 | 1.100667    |
| 1.030000 | 1.045388    |
| 0.770000 | 0.849145    |
| 1.070000 | 1.046299    |
| 1.070000 | 1.021756    |
| 0.940000 | 0.961436    |
| 1.100000 | 1.038587    |
| 1.100000 | 1.090168    |
| 1.100000 | 1.076021    |
| 0.910000 | 0.892927    |
| 0.870000 | 0.870194    |
| 0.780000 | 0.816725    |
| 0.820000 | 0.754766    |
| 0.950000 | 0.932288    |



For 50% humidity, 76^{\circ} F and 29.30 barometric pressure, the estimated multiple regression is:  
0.938434



Time taken by function: 6299 microseconds


```

Figure 4.15: The computation of the multiple regression model with SymIntegration (*SymIntegration/Examples/Test SymIntegration Multiple Linear Regression/main.cpp*).

vi. Compute Inferences in Multiple Linear Regression with SymIntegration

The percent survival rate of sperm in a certain type of animal semen, after storage, was measured at various combinations of concentrations of three materials used to increase chance of survival. The data are given in table below. Test the hypothesis that $\beta_2 = -2.5$ at the 0.05 level of significance against the alternative that $\beta_2 > -2.5$. Then construct a 95% confidence interval for the mean response when $x_1 = 3\%$, $x_2 = 8\%$, and $x_3 = 9\%$.

y (% survival)	x_1 (weight %)	x_2 (weight %)	x_3 (weight %)
25.5	1.74	5.30	10.80
31.2	6.32	5.42	9.40
25.9	6.22	8.41	7.20
38.4	10.52	4.63	8.50
18.4	1.19	11.60	9.40
26.7	1.22	5.85	9.90
26.4	4.10	6.62	8.00
25.9	6.32	8.72	9.10
32.0	4.08	4.42	8.70
25.2	4.15	7.60	9.20
39.7	10.15	4.83	9.40
35.7	1.72	3.12	7.60
26.5	1.70	5.30	8.20

Table 4.5: Data for survival rate of sperm.

Solution:

$$H_0 : \beta_2 = -2.5$$

$$H_1 : \beta_2 > -2.5$$

Computations:

$$t = \frac{b_2 - \beta_{20}}{s\sqrt{c_{22}}} = \frac{-1.8618 + 2.5}{2.073\sqrt{0.0166}} = 2.390$$

$$P = P(T > 2.390) = 0.04$$

The signal that favors H_1 comes from large values of t . The P -value corresponding to $t = 2.390$ computed by the formula $P(T > 2.390)$ is coming from $H_1 : \beta_2 > -2.5$, since the sign is $>$ in H_1 thus we need to compute the P -value with $>$ sign, the H_1 strongly suggests it is one tail.

VII. HYPOTHESIS TESTING

i. One Sample Hypothesis Test for Comparing Means

[SI*] Hypothesis testing is a statistical process used to make a decision about a population based on sample data. It involves creating a null hypothesis (a statement of no effect or no difference) and an alternative hypothesis, then using sample evidence and probability to test the null hypothesis. The goal is to determine if there is enough evidence to reject the null hypothesis in favor of the alternative.

[SI*] **How Hypothesis Testing Works**

1. **State the hypotheses:** Formulate a null hypothesis (H_0) and an alternative hypothesis (H_1).
 - Null Hypothesis (H_0): A specific statement of no significant difference between a hypothesized value and the value estimated from a sample (e.g., a new drug has no effect).s
 - Alternative Hypothesis (H_1): A statement that contradicts the null hypothesis (e.g., the new drug does have an effect).
2. **Formulate an analysis plan:** Choose a statistical test and define a significance level (α). The choice of test depends on the nature of the data and the question being asked.

The test statistic measures how much the sample data deviates from what we did expect if the null hypothesis were true. Different tests use different statistics:

- (a) Z-test: Used when population variance is known and sample size is large.
- (b) Student's *t*-test: Used when the sample size is small or population variance unknown.
- (c) Chi-Squared test: Used for categorical data to compare observed vs expected counts.

3. **Analyze the sample data:** Collect a sample data and perform the chosen statistical test. This involves calculating a test statistic and a *P*-value.
4. **Interpret the results:**

(a) **Using *P*-value:**

Compare the *P*-value to the significance level (α) to decide whether to reject or fail to reject the null hypothesis.

- If the *P*-value is less than the significance level (α), you reject the null hypothesis.
- If the *P*-value is greater than or equal to the significance level, you fail to reject the null hypothesis.

The *P*-value approach has been adopted extensively by users of applied statistics. The approach is designed to give the user an alternative to a mere "reject" or "do not reject" conclusion. The *P*-value computation also gives the user important information when the *z* value falls well into the ordinary critical region.

(b) **Using Critical Value:**

We compute the test statistic, for example if we know that the sample size is more than 30 or we know the population variance then we use *z* statistic (otherwise, we use *t* statistic), thus we use this formula:

$$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

For the case of hypothesis testing with

$$H_0 : \mu = \mu_0$$

$$H_1 : \mu < \mu_0$$

If the test statistic $z < a$ then reject H_0 , if $z \geq a$ then we do not reject H_0 . With a as the critical value

$$a = \mu_0 - z_\alpha \frac{\sigma}{\sqrt{n}}$$

For the case of hypothesis testing with

$$H_0 : \mu = \mu_0$$

$$H_1 : \mu > \mu_0$$

If the test statistic $z > b$ then reject H_0 , if $z \leq b$ then we do not reject H_0 . With b as the critical value

$$b = \mu_0 + z_\alpha \frac{\sigma}{\sqrt{n}}$$

For the case of hypothesis testing with

$$H_0 : \mu = \mu_0$$

$$H_1 : \mu \neq \mu_0$$

If the test statistic $z < a$ or $z > b$ then reject H_0 , if $z \geq a$ then we do not reject H_0 . With a and b as the critical value

$$a = \mu_0 - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

$$b = \mu_0 + z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

[SI*] Examples that are using hypothesis testing:

- Testing a new drug by comparing a group that receives the drug to a control group to see if there is a statistically significant difference in outcomes.
- Assessing if a change in a production process has resulted in a statistical difference in product yield.

[SI*] Types of hypothesis testing:

1. Two-tailed

$$H_0 : \mu = c$$

$$H_1 : \mu \neq c$$

If the hypothesis test is two-tailed, with variance known and large sample size then the P -value can be computed with:

$$P = P(|Z| > z_{\frac{\alpha}{2}}) = 2P(Z < -z_{\frac{\alpha}{2}})$$

with

$$z_{\frac{\alpha}{2}} = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

2. One-tailed (Left-tailed)

$$\begin{aligned} H_0 : \mu &\geq c \\ H_1 : \mu &< c \end{aligned}$$

If the hypothesis test is one-tailed (left-tailed), with variance known and large sample size then the P -value can be computed with:

$$P = P(Z < z_{\frac{\alpha}{2}})$$

with

$$z_{\frac{\alpha}{2}} = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

3. One-tailed (Right-tailed)

$$\begin{aligned} H_0 : \mu &\leq c \\ H_1 : \mu &> c \end{aligned}$$

If the hypothesis test is one-tailed (right-tailed), with variance known and large sample size then the P -value can be computed with:

$$P = P(Z > z_{\frac{\alpha}{2}})$$

with

$$z_{\frac{\alpha}{2}} = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

[SI*] We will explore hypothesis testing through a few specific practical hypothesis tests. Recall the general hypothesis test formulation where we partition the parameter space Θ as follows:

$$H_0 : \theta \in \Theta_0, \quad H_1 : \theta \in \Theta_1$$

One of the most common cases for a single population is to consider θ as μ , the population mean, in which case $\Theta = \mathbb{R}$. Often, we wish to test if the population mean is equal to some value, μ_0 . This allows us to construct a two-sided hypothesis test as follows:

$$H_0 : \mu = \mu_0, \quad H_1 : \mu \neq \mu_0 \tag{4.118}$$

However, one could instead chose to construct a one-sided hypothesis test, as

$$H_0 : \mu \leq \mu_0, \quad H_1 : \mu > \mu_0 \tag{4.119}$$

or alternatively, in the opposite direction

$$H_0 : \mu \geq \mu_0, \quad H_1 : \mu < \mu_0 \tag{4.120}$$

the choice of setting up which one out of these three hypothesis tests, depend on the context of the problem.

Once the hypothesis is established, the general approach involves calculating the test statistic, along with the corresponding p -value, and then finally making some statement about the null hypothesis based on some chosen level of significance.

[SI*] Possible situations for testing a statistical hypothesis

	H_0 is true	H_0 is false
Do not reject H_0	Correct decision	Type II error
Reject H_0	Type I error	Correct decision

Table 4.6

[SI*] Statistical Hypothesis with Discrete Data

A certain type of cold vaccine is known to be only 25% effective after a period of 2 years. To determine if a new and somewhat more expensive vaccine is superior in providing protection against the same virus for a longer period of time, suppose that 20 people are chosen at random and inoculated. If more than 8 of those receiving the new vaccine surpass the 2-year period without contracting the virus, the new vaccine will be considered superior to the one presently in use. We are essentially testing the null hypothesis that the new vaccine is equally effective after a period of 2 years as the one now commonly used. The alternative hypothesis is that the new vaccine is in fact superior. The hypothesis testing is usually written as follows

$$\begin{aligned} H_0 &: p = 0.25 \\ H_1 &: p > 0.25 \end{aligned}$$

The test statistic on which we base our decision is X , the number of individuals in our test group who receive protection from the new vaccine for a period of at least 2 years. The possible values of X , from 0 to 20, are divided into two groups: those numbers less than or equal to 8 and those greater than 8. All possible scores greater than 8 constitute the critical region. The last number that we observe in passing into the critical region is called the critical value.

Therefore, if $x > 8$, we reject H_0 in favor of the alternative hypothesis H_1 . If $x \leq 8$, we fail to reject H_0 .

1. The Probability of Type I Error

The probability of committing a type I error, also called the level of significance, is denoted by the Greek letter α .

In our illustration, a type I error will occur when more than 8 individuals inoculated with the new vaccine surpass the 2-year period without contracting the virus and researchers conclude that the new vaccine is better when it is actually equivalent to the one in use. Hence, if X is the number of individuals who remain free of the virus for at least 2 years,

$$\begin{aligned} \alpha &= P(\text{type I error}) \\ &= P(X > 8, p = 0.25) \\ &= \sum_{x=9}^{20} b(x; 20, 0.25) \\ &= 1 - \sum_{x=0}^8 b(x; 20, 0.25) \\ &= 1 - 0.9591 \\ &= 0.0409 \end{aligned}$$

We say that the null hypothesis, $p = 0.25$, is being tested at the $\alpha = 0.0409$ level of

significance. Sometimes the level of significance is called the size of the test. A critical region of size 0.0409 is very small, and therefore it is unlikely that a type I error will be committed. Consequently, it would be most unusual for more than 8 individuals to remain immune to a virus for a 2-year period using a new vaccine that is essentially equivalent to the one now on the market.

2. The Probability of Type II Error

The probability of committing a type II error, denoted by β , is impossible to compute unless we have a specific alternative hypothesis. If we test the null hypothesis that $p = 0.25$ against the alternative hypothesis that $p = 0.5$, then we are able to compute the probability of not rejecting H_0 when it is false.

We simply find the probability of obtaining 8 or fewer in the group to surpass the 2-year period when $p = 0.5$. In this case,

$$\begin{aligned}\beta &= P(\text{type II error}) \\ &= P(X \leq 8, p = 0.5) \\ &= \sum_{x=0}^8 b(x; 20, 0.5) \\ &= 0.2517\end{aligned}$$

This is a rather high probability, indicating a test procedure in which it is quite likely that we shall reject the new vaccine when, in fact, it is superior to what is now in use. Ideally, we like to use a test procedure for which the type I and type II error probabilities are both small.

The probability of committing both types of error can be reduced by increasing the sample size.

[SI*] Statistical Hypothesis with Continuous Data

Consider the null hypothesis that the average weight of male students in a certain college is 68 kilograms against the alternative hypothesis that is unequal to 68. That is, we wish to test

$$\begin{aligned}H_0 : \mu &= 68 \\ H_1 : \mu &\neq 68\end{aligned}$$

The alternative hypothesis allows for the possibility that $\mu < 68$ or $\mu > 68$.

Reject $H_0 (\mu \neq 68)$	Do not reject $H_0 (\mu = 68)$	Reject $H_0 (\mu \neq 68)$
----------------------------	--------------------------------	----------------------------

Table 4.7: Critical region (in light cyan)

Assume the standard deviation of the population of weights to be $\sigma = 3.6$. For large samples, we may substitute s for σ if no other estimate of σ is available. Our decision statistic, based on a random sample of size $n = 36$, will be \bar{X} , the most efficient estimator of μ .

From the Central Limit Theorem, we know that the sampling distribution of \bar{X} is approximately normal with standard deviation

$$\sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}} = \frac{3.6}{6} = 0.6$$

1. The Probability of Type I Error

The probability of committing a type *I* error, or the level of significance of our test, is equal to the sum of the areas that have been shaded in each tail of the distribution.

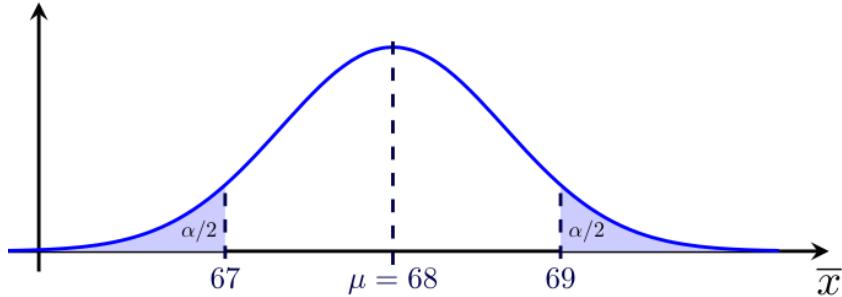


Figure 4.16: Critical region for testing $\mu = 68$ versus $\mu \neq 68$.

Therefore

$$\alpha = P(\bar{X} < 67, \mu = 68) + P(\bar{X} > 69, \mu = 68)$$

The *z*-values corresponding to $\bar{x}_1 = 67$ and $\bar{x}_2 = 69$ when H_0 is true are

$$\begin{aligned} z_1 &= \frac{67 - 68}{0.6} = -1.67 \\ z_2 &= \frac{69 - 68}{0.6} = 1.67 \end{aligned}$$

Therefore,

$$\begin{aligned} \alpha &= P(Z < -1.67) + P(Z > 1.67) \\ &= 2P(Z < -1.67) \\ &= 0.0950 \end{aligned}$$

The 9.5% of all samples of size 36 would lead us to reject $\mu = 68$ kilograms when, in fact, it is true. To reduce α , we have a choice of increasing the sample size or widening the fail-to-reject region.

Suppose that we increase the sample size to $n = 64$. Then $\sigma_{\bar{X}} = \frac{3.6}{8} = 0.45$. Now

$$\begin{aligned} z_1 &= \frac{67 - 68}{0.45} = -2.22 \\ z_2 &= \frac{69 - 68}{0.45} = 2.22 \end{aligned}$$

Hence,

$$\begin{aligned} \alpha &= P(Z < -2.22) + P(Z > 2.22) \\ &= 2P(Z < -2.22) \\ &= 0.0264 \end{aligned}$$

The reduction in α is not sufficient by itself to guarantee a good testing procedure. We must also evaluate β for various alternative hypotheses.

2. The Probability of Type II Error

If it is important to reject H_0 when the true mean is some value $\mu \geq 70$ or $\mu \leq 66$, then the probability of committing a type II error should be computed and examined for the alternatives $\mu = 66$ and $\mu = 70$. Because of symmetry, it is only necessary to consider the probability of not rejecting the null hypothesis that $\mu = 68$ when the alternative $\mu = 70$ is true. A type II error will result when the sample mean \bar{X} falls between 67 and 69 when H_1 is true. Therefore, we find that

$$\beta = P(67 \leq \bar{X} \leq 69, \mu = 70)$$

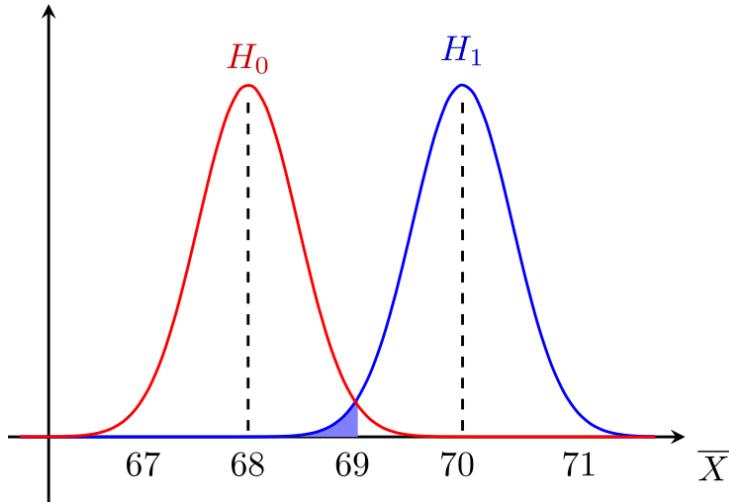


Figure 4.17: Probability of type II error for testing $\mu = 68$ versus $\mu = 70$.

The z -values corresponding to $\bar{x}_1 = 67$ and $\bar{x}_2 = 69$ when H_1 is true are

$$\begin{aligned} z_1 &= \frac{67 - 70}{0.45} = -6.67 \\ z_2 &= \frac{69 - 70}{0.45} = -2.22 \end{aligned}$$

Therefore

$$\begin{aligned} \beta &= P(-6.67 < Z < -2.22) \\ &= P(Z < -2.22) - P(Z < -6.67) \\ &= 0.0132 - 0.0000 \\ &= 0.0132 \end{aligned}$$

If the true value of μ is the alternative $\mu = 66$, the value of β will again be 0.0132. For all possible values of $\mu < 66$ or $\mu > 70$, the value of β will be even smaller when $n = 64$, and consequently there would be little chance of not rejecting H_0 when it is false.

Definition 4.49: Important Properties of a Test of Hypothesis

1. The type I error and type II error are related. A decrease in the probability of one generally results in an increase in the probability of the other.
2. The size of the critical region, and therefore the probability of committing a type I error, can always be reduced by adjusting the critical value(s).
3. An increase in the sample size n will reduce α and β simultaneously.
4. If the null hypothesis is false, β is a maximum when the true value of a parameter approaches the hypothesized value. The greater the distance between the true value and the hypothesized value, the smaller β will be.

Definition 4.50: Power of a Test

The power of a test is the probability of rejecting H_0 given that a specific alternative is true.

The power of a test can be computed as $1 - \beta$. Often different types of tests are compared by contrasting power properties. To produce a desirable power, say, greater than 0.8, one must either increase α or increase the sample size.

Definition 4.51: Test Procedure for a Single Mean (Variance Known)

For large sample size $n > 30$ and known population variance we will use Z statistics

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} > z_{\alpha/2}$$

or

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}} < -z_{\alpha/2}$$

If $-z_{\alpha/2} < z < z_{\alpha/2}$, do not reject H_0 . Rejection of H_0 , of course, implies acceptance of the alternative hypothesis $\mu \neq \mu_0$. With this definition of the critical region, it should be clear that there will be probability α of rejecting H_0 (falling into the critical region) when, indeed, $\mu = \mu_0$.

[SI*] Although it is easier to understand the critical region written in terms of z , we can write the same critical region in terms of the computed average \bar{x} . The following can be written as an identical decision procedure:

Reject H_0 if $\bar{x} < a$ or $\bar{x} > b$, where

$$a = \mu_0 - z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

$$b = \mu_0 + z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

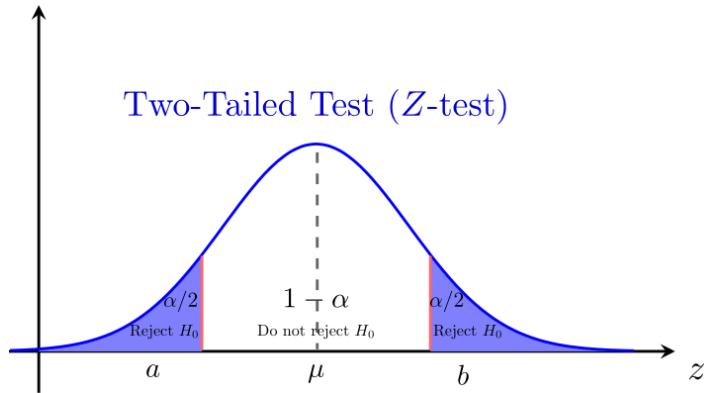


Figure 4.18: Critical region for the alternative hypothesis $\mu \neq \mu_0$.

[SI*] Tests on one-sided hypotheses on the mean involve the same statistic described in the two-sided case. The difference, of course, is that the critical region is only in one tail of the standard normal distribution. For example, suppose that we seek to test

$$H_0 : \mu = \mu_0$$

$$H_1 : \mu > \mu_0$$

The signal that favors H_1 comes from large values of z . Thus, rejection of H_0 results when the computed $z > z_\alpha$. Obviously, if the alternative is $H_1 : \mu < \mu_0$, the critical region is entirely in the lower tail and thus rejection results from $z < -z_\alpha$. Although in a one-sided testing case the null hypothesis can be written as $H_0 : \mu \leq \mu_0$ or $H_0 : \mu \geq \mu_0$, it is usually written as $H_0 : \mu = \mu_0$.

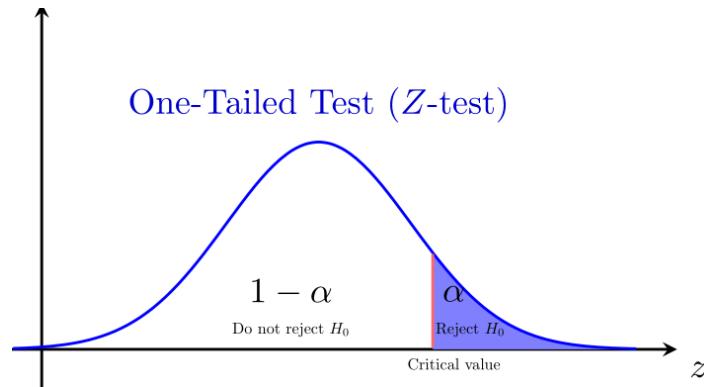


Figure 4.19: Critical region for the alternative hypothesis $\mu > \mu_0$.

Definition 4.52: The *t*-Statistic for a Test on a Single Mean

For two-sided hypothesis with unknown population variance

$$\begin{aligned} H_0 &: \mu = \mu_0 \\ H_1 &: \mu \neq \mu_0 \end{aligned}$$

We reject H_0 at significance level α when computed *t*-statistic

$$t = \frac{\bar{x} - \mu_0}{s / \sqrt{n}}$$

exceeds $t_{\frac{\alpha}{2}, n-1}$ or is less than $-t_{\frac{\alpha}{2}, n-1}$.

The sample variance formula is:

$$s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$$

with

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

GlanzFreya' Guide 4.1: On Type I Error for Hypothesis Testing

Type I error has the same analogy when we make a verdict that an innocent person is guilty.

1. Significance Level (α)

The significance level is a predefined maximum probability of committing type I error we're willing to accept.

The lower the significance level is, the lower the risk of committing a type I error.

With the significance level, we can find the critical value to reject the null hypothesis in a hypothesis test. The critical value for right-tailed hypothesis test:

$$\text{Critical Value} = \mu + Z \left(\frac{\sigma}{\sqrt{n}} \right)$$

The critical value for left-tailed hypothesis test:

$$\text{Critical Value} = \mu - Z \left(\frac{\sigma}{\sqrt{n}} \right)$$

For two-tailed hypothesis test, we use both critical values above so the rejection area will be at $\bar{x} > \mu + Z \left(\frac{\sigma}{\sqrt{n}} \right)$ and at $\bar{x} < \mu - Z \left(\frac{\sigma}{\sqrt{n}} \right)$.

with μ as the population mean, σ as the population standard deviation, if σ is unknown, we can estimate it using sample standard deviation, s , n as the sample size, Z as the Z statistics associated with a given α . If σ is unknown or the sample size is less than 30, we would use T statistics to produce a more realizable result.

2. Sample Size (n)

When we increase the sample size the probability of type I error would decrease. This means that the precision of the test statistic improves.

3. Data Variability

If the data variability decreases, (i.e., the population standard deviation becomes smaller), we would expect to have a smaller probability of committing a type I error.

GlanzFreya' Guide 4.2: On Type II Error for Hypothesis Testing

Type II error has the same analogy when we make a verdict that a guilty person is innocent.

1. Significance Level (α)

A decrease in the significant level causes an increase in the probability of the type II error or a decrease in the power.

2. Sample Size (n)

When we increase the sample size the probability of type II error would decrease.

3. Data Variability

Across different significance levels, if the data variability decreases, (i.e., the population standard deviation becomes smaller), the probability of committing a type II error would decrease.

4. Effect Size

The effect size is the magnitude of the difference between the null hypothesis and the alternative hypothesis.

If the effect size increases, it is easier to detect a true effect, and the probability of a type II error decreases.

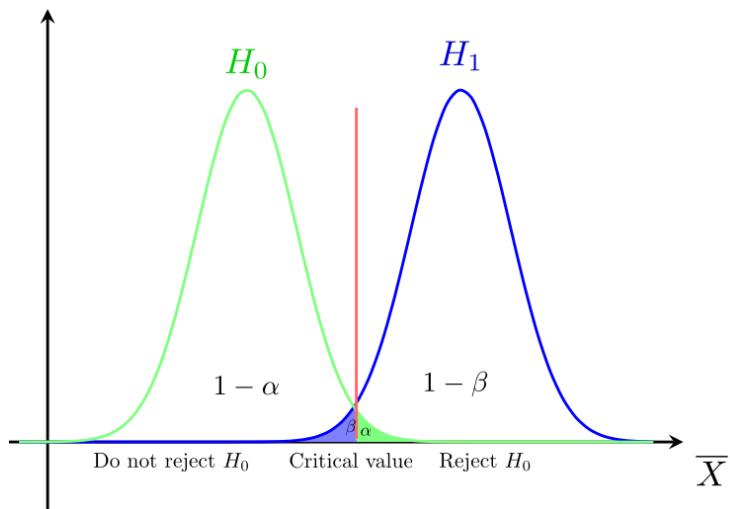


Figure 4.20: Probability of type I and type II errors.

ii. Compute Right-Tailed Hypothesis Testing with SymIntegration

A random sample of 100 deaths in the United States during the past year showed an average life span of 71.8 years. Assuming a population standard deviation of 8.9 years, does this seem to indicate that the mean life span today is greater than 70 years? Use a 0.05 level of significance.

Solution:

We construct the one-sided(right-tailed) hypothesis test as

$$H_0 : \mu = 70 \text{ years}$$

$$H_1 : \mu > 70 \text{ years}$$

with $\alpha = 0.05$. The critical region can be computed by finding the inverse of the cdf of Z / inverse of normal cdf with $\mu = 0, \sigma = 1$ (standard normal distribution), also known as the normal quantile function or the probit function.

To compute the inverse CDF of Z we use Wichura's algorithm, this algorithm calculates the percentage point z_p of the standard normal distribution corresponding to a given lower tail area p , in this case $p = \alpha$.

$$P(Z < z_\alpha) = 0.05$$

With Wichura's algorithm we will obtain:

$$z_\alpha = 1.645$$

thus the critical region is:

$$z > 1.645$$

where

$$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

We are using the data from textfile **Vecotry.txt** that has 100 data of recorded deaths with mean

$$\bar{x} = 71.8842$$

the standard deviation from the sample is

$$s = 6.63$$

while from the problem it is known that the population standard deviation is

$$\sigma = 8.9$$

because we are using Z statistics so we will use σ instead of s , and hence

$$z_{\text{computed}} = \frac{71.8 - 70}{8.9 / \sqrt{100}} = 2.02$$

Decision: Reject H_0 (because $z_{\text{computed}} > z_\alpha$) and conclude that the mean life span today is greater than 70 years.

If we want to use P -value method, then we need to compute

$$P = P(Z > Z_{\text{computed}}) = P(Z > 2.02) = 0.0216$$

As a result, we reach the same conclusion with P -value method, since the P -value $< \alpha$ then we have to reject H_0 .

In SymIntegration we have some functions to compute the hypothesis test, for this case we will use this function:

hypothesistest_righttailed(vector<double>, double μ_0 , double σ , double α , double δ)

The input is a vector for the data, then the μ_0 that will be used for the H_0 , σ is the population standard deviation, if the sample size is less than 30 then the function will use s (sample standard deviation) with t statistics instead, α is the level of significance / type I error, δ is the effect size that is defined with $\delta = |\bar{x} - \mu|$.

Vectory.txt	
1	60.90
2	70.91
3	65.96
4	66.89
5	71.00
6	61.10
7	61.15
8	68.03
9	69.77
10	71.07
11	61.07
12	65.94
13	71.10
14	64.10
15	62.10
16	60.91
17	70.87
18	80.78
19	60.82
20	65.95
21	64
22	72
23	72
24	72
25	75
26	76
27	77
28	77
29	79
30	80
31	81
32	82
33	80
34	80
35	80
36	80
37	80
38	80
39	88
40	82
41	80
42	80
43	80
44	80

Figure 4.21: The data that is used for this example is saved as **Vectory.txt**.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Hypothesis Test ]# make
g++ -c -o main.o main.cpp
g++ -o main .gdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Hypothesis Test ]# ./main

Sample size : 100

Mode : 72
Median : 72
Mean : 71.8842
Quantile 1/4: 70
Quantile 3/4: 75.25
Variance : 44.0116
Standard deviation : 6.63412

*****
Hypothesis testing right-tailed

H0: μ = 70
H1: μ > 70

The probability of a type I error (alpha): 0.05
The probability of a type II error (beta): 0.352857
Power: 0.647143

Critical value: 71.4639
Critical region : z > 1.64485
Computed z : 2.02247

*Reject the null hypothesis when the sample average is greater than 71.4639
*Reject the null hypothesis when the computed z is greater than 1.64485

P-value: 0.0215638

*Reject null hypothesis if P-value <= 0.05 , we fail to reject the null hypothesis if P-value > 0.05

*****
Time taken by function: 1750 microseconds
```

Figure 4.22: The computation of one-tailed hypothesis testing (right-tailed) with SymIntegration (*SymIntegration/Examples/Test SymIntegration Right-Tailed Hypothesis Testing/main.cpp*).

iii. Compute Two-Tailed Hypothesis Testing with SymIntegration

A manufacturer of sports equipment has developed a new synthetic fishing line that the company claims has a mean breaking strength of 8 kilograms with a standard deviation of 0.5 kilogram. Test the hypothesis that $\mu = 8$ kilograms against the alternative that $\mu \neq 8$ kilograms if a random sample of 50 lines is tested and found to have a mean breaking strength of 7.8 kilograms. Use a 0.01 level of significance.

Solution:

We construct the two-sided(two-tailed) hypothesis test as

$$\begin{aligned} H_0 &: \mu = 8 \text{ kilograms} \\ H_1 &: \mu \neq 8 \text{ kilograms} \end{aligned}$$

with $\alpha = 0.01$.

The critical region can be computed by finding the inverse of the cdf of Z / inverse of normal cdf with $\mu = 0, \sigma = 1$ (standard normal distribution), also known as the normal quantile function or the probit function.

$$P(Z < z_\alpha) = 0.01$$

With Wichura's algorithm we will obtain:

$$z_\alpha = -2.575$$

due to the symmetry of standard normal distribution, the critical regions are:

$$z < -2.575 \quad \text{and} \quad z > 2.575$$

where

$$z = \frac{\bar{x} - \mu_0}{\sigma / \sqrt{n}}$$

We are using the data from textfile **Vectorty.txt** that has 50 data of recorded deaths with mean

$$\bar{x} = 7.8$$

the standard deviation from the sample is

$$s = 1.03$$

while from the problem it is known that the population standard deviation is

$$\sigma = 0.5$$

because we are using Z statistics so we will use σ instead of s , and hence

$$z_{\text{computed}} = \frac{7.8 - 8}{0.5 / \sqrt{50}} = -2.823843$$

Decision: Reject H_0 (because $z_{\text{computed}} < z_\alpha$) and conclude that the mean breaking strength of the synthetic fishing line is not 8 kilograms, but is in fact, less than 8 kilograms based on the sample

data.

Since the test in this example is two-tailed, the desired P -value is twice the area of the shaded region to the left of $z = -2.83$. Therefore, we have

$$P = P(|Z| > 2.83) = 2P(Z < -2.83) = 0.0046$$

In SymIntegration we have some functions to compute the hypothesis test, for this case of two-tailed hypothesis testing we will use this function:

hypothesistest(vector<double>, double μ_0 , double σ , double α , double δ)

The input is a vector for the data, then the μ_0 that will be used for the H_0 , σ is the population standard deviation, if the sample size is less than 30 then the function will use s (sample standard deviation) with t statistics instead, α is the level of significance / type I error, δ is the effect size that is defined with $\delta = |\bar{x} - \mu|$.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Two-Tailed Hypothesis Testing ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Two-Tailed Hypothesis Testing ]# ./main
Sample size : 50
Mode : 7
Median : 8
Mean : 7.8
Quantile 1/4: 7
Quantile 3/4: 8
Variance : 1.06122
Standard deviation : 1.03016
*****
Hypothesis testing two-tailed
H0: μ = 8
H1: μ != 8
The probability of a type I error (alpha): 0.01
The probability of a type II error (beta): 0.199421
Power: 0.800579
Critical value (upper bound): 8.18214
Critical value (lower bound): 7.81786
Critical region : z < - 2.57583 and z > 2.57583
Computed z : -2.82843
*Reject the null hypothesis when the sample average is greater than 8.18214 or less than 7.81786
*Reject the null hypothesis when the computed z is greater than 2.57583 or smaller than -2.57583
P-value: 0.00467773
*Reject null hypothesis if P-value <= 0.01 , we fail to reject the null hypothesis if P-value > 0.01
*****
Time taken by function: 1459 microseconds
```

Figure 4.23: The computation of two-tailed hypothesis testing with SymIntegration (*SymIntegration/Examples/Test SymIntegration Two-Tailed Hypothesis Testing/main.cpp*).

iv. Two Sample Hypothesis Tests for Comparing Means

v. One-Way Analysis of Variance (ANOVA)

[SI*] In the estimation and hypothesis testing section, we were restricted in each case to considering no more than two population parameters. Such was the case, for example, in testing for the equality of two population means using independent samples from normal populations with common but unknown variance, where it was necessary to obtain a pooled estimate of σ^2 .

Suppose in an industrial experiment that an engineer is interested in how the mean absorption of moisture in concrete varies among 5 different concrete aggregates. The samples are exposed to moisture for 48 hours. It is decided that 6 samples are to be tested for each aggregate, requiring a total of 30 samples to be tested.

The model for this situation may be set up as follows. There are 6 observations taken from each of 5 populations with means $\mu_1, \mu_2, \dots, \mu_5$, respectively. We may wish to test

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_5$$

$$H_1 : \text{At least two of the means are not equal}$$

In addition, we may be interested in making individual comparisons among these 5 population means.

In the analysis-of-variance procedure, it is assumed that whatever variation exists among the aggregate average is attributed to

1. Variation in absorption among observation within aggregate types
2. Variation among aggregate types, that is, due to differences in the chemical composition of the aggregates.

[SI*] Random samples of size n are selected from each of k populations. The k different populations are classified on the basis of a single criterion such as different treatments or groups. Today the term **treatment** is used generally to refer to the various classifications, whether they be different aggregates, different analysts, different fertilizers, or different regions of the country.

[SI*] Assumption and Hypotheses in One-Way ANOVA

It is assumed that the k populations are independent and normally distributed with means $\mu_1, \mu_2, \dots, \mu_k$ and common variance σ^2 . These assumptions are made more palatable by randomization. We wish to derive appropriate methods for testing the hypothesis

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_k$$

$$H_1 : \text{At least two of the means are not equal}$$

Let y_{ij} denote the j -th obsevation from the i -th treatment and arrange the data to become a table. Here, Y_i is the total of all observations in the sample from the i -th treatment, \bar{y}_i is the mean of all observations in the sample from the i -th treatment, $Y..$ is the total of all nk observations, and $\bar{y}..$ is the mean of all nk observations.

[SI*] Model for One-Way ANOVA

Each observation may be written in the form

$$Y_{ij} = \mu_i + \epsilon_{ij}$$

where ϵ_{ij} measures the deviation of the j -th observation of the i -th sample from the corresponding treatment mean. The ϵ_{ij} -term represents random error and plays the same role as the error terms in the regression models. An alternative preferred form of this equation is obtained by substituting $\mu_i = \mu + \alpha_i$, subject to the constraint

$$\sum_{i=1}^k \alpha_i = 0$$

Hence, we may write

$$Y_{ij} = \mu + \alpha_i + \epsilon_{ij}$$

where μ is just the grand mean of all the μ_i , that is,

$$\mu = \frac{1}{k} \sum_{i=1}^k \mu_i$$

and α_i is called the effect of the i -th treatment.

The null hypothesis that the k population means are equal against the alternative that at least two of the means are unequal may now be replaced by the equivalent hypothesis

$$H_0 : \alpha_1 = \alpha_2 = \dots = \alpha_k$$

$$H_1 : \text{At least one of the } \alpha_i \text{ is not equal to zero}$$

[SI*] Resolution of Total Variability into Components

Our test will be based on a comparison of two independent estimates of the common population variance σ^2 . These estimates will be obtained by partitioning the total variability of our data, designated by the double summation

$$\sum_{i=1}^k \sum_{j=1}^n (y_{ij} - \bar{y}_{..})^2$$

into two components.

Theorem 4.21: Sum of Squares Identity

The formula is

$$\sum_{i=1}^k \sum_{j=1}^n (y_{ij} - \bar{y}_{..})^2 = n \sum_{i=1}^k (\bar{y}_{i.} - \bar{y}_{..})^2 + \sum_{i=1}^k \sum_{j=1}^n (y_{ij} - \bar{y}_{i.})^2$$

Definition 4.53: Three Important Measures of Variability

It will be convenient in what follows to identify the terms of the sum-of-squares identity by the following notation:

SST (Total sum of squares)

$$\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_{..})^2$$

SSA (Treatment sum of squares)

$$n \sum_{i=1}^k (\bar{y}_{i\cdot} - \bar{y}_{..})^2$$

SSE (Error sum of squares)

$$\sum_{i=1}^k \sum_{j=1}^{n_i} (y_{ij} - \bar{y}_{i\cdot})^2$$

The sum-of-squares identity can then be represented symbolically by the equation

$$SST = SSA + SSE$$

The identity above expresses how between-treatment and within-treatment variation add to the total sum of squares.

Theorem 4.22: The Expected Value of SSA

The formula is

$$E(SSA) = (k - 1)\sigma^2 + n \sum_{i=1}^k \alpha_i^2$$

Definition 4.54: Treatment Mean Square

If H_0 is true, an estimate of σ^2 , based on $k - 1$ degrees of freedom, is provided by this expression:

$$s_1^2 = \frac{SSA}{k - 1}$$

If H_0 is true and thus each α_i in $E(SSA)$ is equal to zero, we see that

$$E\left(\frac{SSA}{k - 1}\right) = \sigma^2$$

and s_1^2 is an unbiased estimate of σ^2 . However, if H_1 is true, we have

$$E\left(\frac{SSA}{k - 1}\right) = \sigma^2 + \frac{n}{k - 1} \sum_{i=1}^k \alpha_i^2$$

and s_1^2 estimates σ^2 plus an additional term, which measures variation due to the systematic effects.

Definition 4.55: Error Mean Square

A second and independent estimate of σ^2 , based on $k(n - 1)$ degrees of freedom, is this familiar formula:

$$s^2 = \frac{SSE}{k(n - 1)} \quad (4.121)$$

It is instructive to point out the importance of the expected values of the mean squares indicated above.

[SI*] Use of F-Test in ANOVA

The estimate s^2 is unbiased regardless of the truth or falsity of the null hypothesis. It is important to note that the sum-of-squares identity has partitioned not only the total variability of the data, but also the total number of degrees of freedom. That is,

$$nk - 1 = k - 1 + k(n - 1)$$

[SI*] F-Ratio for Testing Equality of Means

When H_0 is true, the ratio $f = \frac{s_1^2}{s^2}$ is a value of the random variable F having the F -distribution with $k - 1$ and $k(n - 1)$ degrees of freedom. Since s_1^2 overestimates σ^2 when H_0 is false, we have a one-tailed test with the critical region entirely in the right tail of the distribution.

The null hypothesis H_0 is rejected at the α -level of significance when

$$f > f_\alpha[k - 1, k(n - 1)]$$

Another approach, the P -value approach, suggests that the evidence in favor or against H_0 is

$$P = P\{f[k - 1, k(n - 1)] > f\}$$

When H_1 is true, the presence of the condition $E(s_1^2) > E(s^2)$ suggests that the F -ratio be used in the context of a one-sided upper-tailed test. That is, we would expect the numerator

s_1^2 to exceed the denominator.

The computations for an analysis-of-variance problem are usually summarized in tabular form.

[SI*] Assumptions for one-way ANOVA:

1. The dependent variable is approximately normally distributed within each group. This assumption is more critical for smaller sample sizes.
2. The samples are selected at random and should be independent of one another.
3. All groups have equal standard deviations.
4. Each data point should belong to one and only one group / treatment. There should be no overlap or sharing of data points between groups / treatments.

[SI*] There are two main types of ANOVA:

1. **One-way ANOVA**

This is the most basic form of ANOVA and is used when there is only one independent variable with more than two treatments or groups. It assesses whether there are any statistically significant differences among the means of the groups.

2. **Two-way ANOVA**

Extending the one-way ANOVA, two-way ANOVA involves two independent variables. It allows for examination of the main effects of each variable as well as the interaction between them. The interaction effect explores whether the effect of one variable on the dependent variable is different depending on the level of the other variable.

GlanzFreya' Guide 4.3: How to Perform One-Way ANOVA

One-way ANOVA is a type of hypothesis test where only one factor is considered. We use F -statistic to perform a one-way analysis of variance.

The steps are:

1. Define the null and alternative hypothesis

$$H_0 : \mu_1 = \mu_2 = \dots = \mu_k$$

H_1 : At least two of the means are not equal

2. Compute the degrees of freedom between and within the treatments / groups.

$$df_{\text{between}} = k - 1$$

$$df_{\text{within}} = N - k$$

$$df_{\text{total}} = N - 1$$

with N is the number of samples in all treatments / groups combined, and k is the number of treatments / groups.

3. Compute the mean of all samples in each treatment / group, then compute the grand mean ($\mu_1, \dots, \mu_k, \mu_{\text{grand}}$).
4. Compute the sum of squares total and sum of squares within, then compute the sum of squares between (SS_A, SST, SSE).
5. Compute the variance between and within samples

$$s_1^2 = \frac{SS_A}{k - 1}$$

$$s^2 = \frac{SSE}{k(n - 1)}$$

6. Compute the F value with this formula:

$$F_{\text{value}} = \frac{s_1^2}{s^2}$$

7. Compute the P value with this formula:

$$P_{\text{value}} = 1 - P(X \leq F_{\text{value}}) = 1 - F(F_{\text{value}}; k - 1, N - k)$$

with $F(F_{\text{value}}; k - 1, N - k)$ is the cdf of F -distribution with numerator degree of freedom $k - 1$ and denominator degree of freedom $N - k$.

8. If $F_{\text{value}} < P_{\text{value}}$ then we fail to reject the null hypothesis.

If $F_{\text{value}} > P_{\text{value}}$ then we reject the null hypothesis.

The fundamental strategy of ANOVA is to systematically examine variability within treatments / groups being compared and also examine variability among the groups being compared.

vi. Compute One-Way ANOVA with SymIntegration

Suppose in an industrial experiment that an engineer is interested in how the mean absorption of moisture in concrete varies among 5 different concrete aggregates. The samples are exposed to moisture for 48 hours. It is decided that 6 samples are to be tested for each aggregate, requiring a total of 30 samples to be tested.

The model for this situation may be set up as follows. There are 6 observations taken from each of 5 populations with means $\mu_1, \mu_2, \dots, \mu_5$, respectively.

Aggregate:	1	2	3	4	5
	551	595	639	417	563
	457	580	615	449	631
	450	508	511	517	522
	731	583	573	438	613
	499	633	648	415	656
	632	517	677	555	679
Total	3320	3416	3663	2791	3664
Mean	553.33	569.33	610.50	465.17	610.67
					16,854
					561.80

Table 4.8: The absorption of moisture in concrete aggregates.

Test the hypothesis $\mu_1 = \mu_2 = \mu_3 = \mu_4 = \mu_5$ at the 0.05 level of significance for the data on absorption of moisture by various types of cement aggregates.

Solution:

The hypotheses are

$$\begin{aligned} H_0 &: \mu_1 = \mu_2 = \dots = \mu_5 \\ H_1 &: \text{At least two of the means are not equal} \\ \alpha &= 0.05 \end{aligned}$$

Critical region :

$$f > 2.76$$

with $v_1 = 4$ and $v_2 = 25$ degrees of freedom. The sum-of-squares computations give

$$\begin{aligned} SST &= 209,377 \\ SSA &= 85,356 \\ SSE &= 209,377 - 85,356 = 124,021 \end{aligned}$$

Decision: Reject H_0 and conclude that the aggregates do not have the same mean absorption. The P -value for $f = 4.30$ is 0.0088, which is smaller than $\alpha = 0.05$.

This example has equal sample size from aggregate 1 to aggregate 5, however in real life, during experimental work, one often loses some of the desired observations. Experimental animals may die, experimental material may be damaged, or human subjects may drop out of a study. For analysis with unequal sample size, we just need to adjust the formula for SST , SSA and SSE .

In SymIntegration, the function to compute the one-way analysis-of-variance / ANOVA is:
ANOVA(vector<vector<double>)

The input is a matrix that is defined as **vector<vector<double>** from the standard C++ library. It is a **void** function, so it returns only the table and nothing else.

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration ANOVA ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration ANOVA ]# ./main

Matrix A :
 551.000000  595.000000  639.000000  417.000000  563.000000
 457.000000  580.000000  615.000000  449.000000  631.000000
 450.000000  508.000000  511.000000  517.000000  522.000000
 731.000000  583.000000  573.000000  438.000000  613.000000
 499.000000  633.000000  648.000000  415.000000  656.000000
 632.000000  517.000000  677.000000  555.000000  679.000000

One-Way ANOVA:

  Source   DF      SS       MS      F      P
  Model     4    85356.466667  21339.116667  4.301536  0.008752
  Error    25   124020.333333  4960.813333
  Total    29   209376.800000

  R-Square      Grand Mean      Root MSE
  0.407669      561.800000      70.433041

Time taken by function: 1417 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration ANOVA ]# ]
```

Figure 4.24: The computation of one-way analysis-of-variance / ANOVA for the cement aggregates with SymIntegration, the execution time for the function is around 1417 microseconds without dependencies of using any external libraries besides SymIntegration, it is 1.76 times faster than using Boost. (*SymIntegration/Examples/Test SymIntegration One-Way ANOVA/main.cpp*).

We would like to explain why we tried to use **Boost** as comparison. The library **Boost** is very robust and solid, it includes all pdf and cdf computation for almost all distributions that are known today. So we tried to use **Boost** to compute the cdf for F -distribution to compute the P -value. It resulted in 2494 microseconds for the execution time of the ANOVA function. It becomes slower because we are using two libraries **Boost** and **SymIntegration**. If the reader wants to compare only using **Boost** and create the ANOVA table, they can compare the execution time, **Boost** only, **SymIntegration** only, and **Boost + SymIntegration** (this one will be slower than the previous two options).

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration ANOVA ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration ANOVA ]# ./main

Matrix A :
      551.000000    595.000000    639.000000    417.000000    563.000000
      457.000000    580.000000    615.000000    449.000000    631.000000
      450.000000    508.000000    511.000000    517.000000    522.000000
      731.000000    583.000000    573.000000    438.000000    613.000000
      499.000000    633.000000    648.000000    415.000000    656.000000
      632.000000    517.000000    677.000000    555.000000    679.000000

One-Way ANOVA:

Source      DF           SS          MS          F          P
Model       4            85356.466667    21339.116667    4.301536    0.008752
Error      25            124020.333333    4960.813333
Total       29            209376.800000

R-Square    Grand Mean      Root MSE
0.407669      561.800000      70.433041

Time taken by function: 2494 microseconds

```

Figure 4.25: The computation of one-way analysis-of-variance / ANOVA for the cement aggregates with Boost and SymIntegration, the execution time for the function is around 2494 microseconds.

```

...
double Fcdf(double x, double r1, double r2)
{
    //boost::math::fisher_f_distribution<> fisher_f(r1,r2);
    //double cdf_value = boost::math::cdf(fisher_f,x);

    /*The cumulative distribution function (CDF) for Fisher's F distribution
    It has at least 11 decimal digit precision with boost::math::
        fisher_f_distribution
    */

    double z ;
    double cdf_value;

    z = (x*r1)/(r2 + r1*x);
    cdf_value = incbeta(z,r1/2.0, r2/2.0);

    return cdf_value;
}

...
...

void ANOVA(vector<vector<double>> matrix)
{
    vector<vector<double>> anovamatrix;

```

```

vector<double> total_column;
vector<double> mean_column;
vector<double> sst_column;
vector<double> ssa_column;
int C = matrix[0].size();
int N = 0; // computing total data

for(int i = 0 ; i < C ; ++i)
{
    double sum = 0;
    int R = matrix.size();
    for (int j = 0 ; j < R ; ++j)
    {
        sum += matrix[j][i]; // sum of the column

        if(matrix[j][i] != 0 )
        {
            N = N+1;
        }
        else if(matrix[j][i] == 0 )
        {
            N = N;
        }
    }

    total_column.push_back(sum);
    mean_column.push_back(sum/(R));
    //cout << "sum of column " << i << " = " << total_column[i] << endl;
    //cout << "mean sum of column " << i << " = " << mean_column[i] <<
    endl;
}

...
...

double SSE = SST-SSA;

int df_model = C-1;
int df_total = N-1;
int df_error = df_total - df_model;

double s1_square = SSA/(df_model);
double s1 = SSE/(df_error);
double computed_f = s1_square/s1;

int r1 = df_model;
int r2 = df_error;

```

```

double p_value = 1 - Fcdf(computed_f,r1,r2);
double r_square = 1 - SSE/SST;

cout << "\nSource" << setw(10) << "DF" << setw(23) << "SS" << setw(23) << "
    MS" << setw(23) << "F" << setw(23) << "P" << endl;
cout << "\nModel" << setw(10) << df_model << setw(23) << SSA << setw(23) <<
    s1_square << setw(23) << computed_f << setw(23) << p_value << endl;
cout << "\nError" << setw(10) << df_error << setw(23) << SSE << setw(23) <<
    s1 << setw(23) << endl;
cout << "\nTotal" << setw(10) << df_total << setw(23) << SST << endl;

cout << "\nR-Square" << setw(23) << "Grand Mean" << setw(23) << "Root MSE"
    << endl;
cout << r_square << setw(23) << y_bar << setw(23) << sqrt(s1) << endl;

}

```

Code 47: *src/statistics.cpp*

To compute the cdf of F -distribution is not an easy task. We need to use numerical integration of the pdf of F -distribution, most people will use a well-tested library like **Boost** that provides high accuracy and handles potential numerical issues. We use manual implementation made by **Lewis Van Winkle** with zlib license, he created a C file that compute incomplete Beta function, that is very important in order to compute the cdf of F -distribution and student's t -distribution by continued fraction using Lentz's algorithm. The article can be read here:

<https://codeplea.com/incomplete-beta-function-c>

Chapter 5

Operations Research Computation with SymIntegration

"Without mathematics, there's nothing you can do. Everything around you is mathematics. Everything around you is numbers." - Shakuntala Devi

Operations research is a branch of mathematics which is concerned with the application of scientific methods and techniques to decision making problems and with establishing the best or optimal solutions.

I. LINEAR PROGRAMMING

[SI*] Linear programming is concerned with the optimization (minimization or maximization) of a linear function while satisfying a set of linear equality and/or inequality constraints or restrictions. The linear programming problem was first conceived by George B. Dantzig around 1947 while he was working as a Mathematical Advisor to the United States Air Force Comptroller on developing a mechanized planning tool for a time-staged deployment, training, and logistical supply program.

In 1949, George B. Dantzig published the "simplex method" for solving linear programs. Since that time a number of individuals have contributed to the field of linear programming in many different ways, including theoretical development, computational aspects, and exploration of new applications of the subject [1].

[SI*] The standard form of linear programming in matrix form:

Minimize

$$\mathbf{c}^t \mathbf{X} \quad (5.1)$$

subject to the constraints

$$A\mathbf{X} = \mathbf{b} \quad (5.2)$$

and

$$\mathbf{X} \geq \mathbf{0} \quad (5.3)$$

where

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

It can be seen that there are m equations and n decision variables in a linear programming problem.

We may assume that $m < n$, for if $m > n$ there would be $m - n$ redundant equations which could be eliminated (it is the basic knowledge of linear algebra, read more here [10]). The case where $m = n$ is of no interest, for then there is either a unique solution \mathbf{X} which satisfies Eqs. (8.3) and (8.4) (in which case there can be no optimization) or no solution, in which case the constraints are inconsistent.

the case $m < n$ corresponds to an underdetermined set of linear equations which, if they have one solution, have an infinite number of solutions. The problem of linear programming is to find one of these solutions satisfying Eqs.(8.3) and (8.4) and yielding the minimum of f .

[SI*] A linear programming problem may have

1. A unique and finite optimum solution.
2. An infinite number of optimal solutions.
3. An unbounded solution.
4. No solution.
5. A unique feasible point.

Assuming that the linear programming is properly formulated, the following general geometrical characteristics can be noted from the graphical solution.

1. The feasible region is a convex polygon.
2. The optimum value occurs at an extreme point or vertex of the feasible region.

i. Simplex Method

Find the amount of each type of food to be purchased in order to meet exactly the daily requirements of a person at minimum cost. Assume that a person, on the average, requires 3000 calories and 100 grams of protein.

	Bread	Meat	Potatoes	Cabbage	Milk
Calories	2500	3000	600	100	600
Protein (grams)	80	150	20	10	40
Cost (USD/kg)	3	10	1	2	3

Table 5.1: The calorie values and the protein contents of five types of foods along with their costs.

Solution:

Let x_1, x_2, x_3, x_4 , and x_5 denote the amount of bread, meat, potatoes, cabbage, and milk to be purchased in kg per day. Then the objective function to be minimized is given by

$$f = 3x_1 + 10x_2 + x_3 + 2x_4 + 3x_5 \quad (5.4)$$

The constraints due to the calorie and protein requirements are respectively given by

$$\begin{aligned} 2500x_1 + 3000x_2 + 600x_3 + 100x_4 + 600x_5 &= 3000 \\ 80x_1 + 150x_2 + 20x_3 + 10x_4 + 40x_5 &= 100 \end{aligned} \quad (5.5)$$

The nonnegativity requirements of the decision variables are given by

$$x_i \geq 0, \quad i = 1, 2, 3, 4, 5 \quad (5.6)$$

Since $m = 2$ (the number of equality constraints) and $n = 5$ (the number of decision variables), a basic solution can be obtained by setting any of the $(n - m)$ variables equal to zero and solving the constraint equations. The number of basic solutions is

$$\frac{5!}{3!2!} = 10$$

Out of these ten basic solutions, the solution which has all the variables nonnegative and makes the objective function given by Eq. (5.4) assume a minimum value will be the optimum.

The simplex computation makes use of Gaussian elimination to compute the values of the basic variables and the objective function for all basic solutions.

This is the initial simplex tableau:

$$S = \left[\begin{array}{cccccc|c|c} x_1 & x_2 & x_3 & x_4 & x_5 & f & C \\ 2500 & 3000 & 600 & 100 & 600 & 0 & 3000 \\ 80 & 150 & 20 & 10 & 40 & 0 & 100 \\ -3 & -10 & -1 & -2 & -3 & 1 & 0 \end{array} \right] \quad (5.7)$$

Thus we make this matrix A that represents the constraints equations only:

$$A = \begin{bmatrix} 2500 & 3000 & 600 & 100 & 600 & 0 & 3000 \\ 80 & 150 & 20 & 10 & 40 & 0 & 100 \end{bmatrix} \quad (5.8)$$

The matrix A is the one that will be the input for the C++ codes, the objective function will be inputted as a vector, to create the matrix A we make two options, the first one is loading the matrix from textfile (**matrix.txt**) and the second one is to input the coefficients manually in the C++ source code (**main.cpp**).

To load a **std::vector** from a text file in C++, we need to include **<fstream>** for file operations, **<vector>** for using vector, **<string>** for reading lines, and **<iostream>** for input/output.

The loaded vector will then be saved as Symbolic matrix **B_mat** that will be used as the input to do the simplex computation.

To do the simplex computation we use this function in SymIntegration:
Symbolic simplexmethod(const SymbolicMatrix &A, vector<double> f, int C, int R, int n)

with A as the SymbolicMatrix that represent the constraint equations that we have defined in Eq. (5.8), f as the vector of objective function, C as the number of column for matrix A , R as the number of row for matrix A , and n is the number of decision variables.

```

xterm
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# ./main
Simplex Method :
A :
[2500 3000 600 100 600 0 3000]
[ 80 150 20 10 40 0 100]

*****Basic Variables*****
0 1
*****My wife is the most beautiful Goddess, she teaches me this !
A :
[2500 3000 0 0 0 0 3000]
[ 80 150 0 0 0 0 100]

A (in reduced row form) :
[2500 3000 0 0 0 0 3000]
[ 0 54 0 0 0 0 4 ]

A (in row reduced echelon form) :
[ 1 1.2 0 0 0 0 1.2 ]
[ 0 1 0 0 0 0 0.0740741]

Simplified final matrix :
[ 1 1.2 1.2 ]
[ 0 1 0.0740741]

Solution:
x_{1} = 1.11111
x_{2} = 0.0740741

Value of the objective function : 4.07407

*****Basic Variables*****
0 2
*****My wife is the most beautiful Goddess, she teaches me this !
A :
[2500 0 600 0 0 0 3000]
[ 80 0 20 0 0 0 100]

A (in reduced row form) :
[2500 0 600 0 0 0 3000]
[ 0 0 0.8 0 0 0 4 ]

```

Figure 5.1: The function in SymIntegration to compute the linear programming problem with simplex method (*SymIntegration/Examples/Test SymIntegration Simplex Method Function/main.cpp*).

```

x_{5} = -5.55112e-16
Value of the objective function : 5
*****
*****Basic Variables*****
*****
3 4
*****
My wife is the most beautiful Goddess, she teaches me this !
A :
[ 0   0   0   100  600  0   3000]
[ 0   0   0   10   40   0   100]

A (in reduced row form) :
[ 0   0   0   100  600  0   3000]
[ 0   0   0   10   40   0   100]

A (in row reduced echelon form) :
[ 0           0           0   0.03333333   0.2       0       1   ]
[ 0           0           0   0.1         0.4       0       1   ]

Simplified final matrix :
[0.03333333   0.2       1   ]
[ 0.1         0.4       1   ]

Solution:
x_{4} = -30
x_{5} = 10

Value of the objective function : -30
*****
*****End of Simplex Method*****
*****

Objective Function Value      Decision Variables
4.1                         0   1
5                           0   2
4.7                         0   3
4                           0   4
5                           1   2
-6.7                         1   3
5                           1   4
5                           2   3
5                           2   4
-30                          3   4
0

Time taken by function: 51473 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# []

```

Figure 5.2: The function in SymIntegration to compute the linear programming problem with simplex method takes 51473 microseconds (*SymIntegration/Examples/Test SymIntegration Simplex Method Function/main.cpp*).

Due to the C++ coding the meaning of decision variable 0 is for x_1 , 1 for x_2 and so on, from the result we know that the feasible and optimum solution occurs when we use basic variables 0 and 4, or, x_1 and x_5 , referring to bread and milk. With the values for the basic variables:

$$\begin{aligned}x_1 &= 1.15385 \\x_5 &= 0.192308 \\f &= 4.03846\end{aligned}$$

In real life, you cannot really buy 0.192308 kg of milk at that exact decimal value, so we usually round it to the nearest possible value that we can use in real life / in the market.

For comparison, we also try to use **Armadillo** to handle the Gaussian elimination part, but it becomes slower because we need to add another library when compiling.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# make
g++ -c -o main.o main.cpp
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# g++ main.o -larmadillo -lsymintegration -lsymintegration_larmadillo
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]# ./main
Simplex Method :
A :
[3000 3000 600 100 600 0 3000]
[ 0 150 20 10 40 0 100]

*****Basic Variables*****
0 1
*****
My wife is the most beautiful Goddess, she teaches me this !
A :
[2500 3000 0 0 0 0 3000]
[ 0 150 0 0 0 0 100]

A (in reduced row form) :
[2500 3000 0 0 0 0 3000]
[ 0 54 0 0 0 0 4 ]

A (in row reduced echelon form) :
[ 0 0 0 1.2 0 0 0 0 0 0 0.12 ]
[ 0 1 0 0 0 0 0 0 0 0 0.0746741]

Simplified final matrix :
[ 0 0 0 1.2 0 0 0 0 0 0 0.12 ]
[ 0 1 0 0 0 0 0 0 0 0 0.0746741]

Solution:
x_{(1)} = 1.11111
x_{(2)} = 0.0740741

Value of the objective function : 4.07407

```

Figure 5.3: Comparing computing with simplex method with SymIntegration and Armadillo (*SymIntegration/Examples/Test SymIntegration Simplex Method Function/main.cpp*).

```

*****Basic Variables*****
3 4
*****
My wife is the most beautiful Goddess, she teaches me this !
A :
[ 0 0 0 100 600 0 3000]
[ 0 0 0 10 40 0 100]

A (in reduced row form) :
[ 0 0 0 100 600 0 3000]
[ 0 0 0 10 40 0 100]

A (in row reduced echelon form) :
[ 0 0 0 0 0.0333333 0.2 0 0 1 ]
[ 0 0 0 0 0.0333333 0.4 0 0 1 ]

Simplified final matrix :
[ 0.0333333 0.2 0 1 ]
[ 0.1 0.4 0 1 ]

Solution:
x_{(4)} = -30
x_{(5)} = 10

Value of the objective function : -30
*****End of Simplex Method*****
*****Objective Function Value Decision Variables*****
4.1 0 2
5 0 3
4.7 0 4
4 1 2
5 1 3
-6.7 1 4
5 2 3
5 2 4
-30 3 4
0 3 4

Time taken by function: 81639 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Simplex Method Function ]#

```

Figure 5.4: The function in SymIntegration to compute the linear programming problem with simplex method and Armadillo for the Gaussian elimination part takes 82639 microseconds (*SymIntegration/Examples/Test SymIntegration Simplex Method Function/main.cpp*).

So we remove the dependencies toward **Armadillo** to make things easier and faster, not all functions and features on **Armadillo** can be done now in SymIntegration (September 30th, 2025) but eventually as we grow we will add more needed functions besides Gaussian elimination for SymIntegration, probably in the future we will work on QR method, Spectral decomposition, SVD, and Cramer's Rule.

II. NONLINEAR PROGRAMMING

[SI*]

Chapter 6

Numerical Linear Algebra with SymIntegration

"All the girls in the world were divided into two classes: one class included all the girls in the world except her, and they had all the usual human feelings and were very ordinary girls; while the other class -herself alone- had no weaknesses and was superior to all humanity." - Anna Karenina - Leo Tolstoy

Why we create this chapter when we have created a book in 2023 [5] that contain a full summary of Elementary Linear Algebra? Because it is still using open source C++ library like **Eigen**, **Armadillo**, **GiNaC**, **SymbolicC++** that are created by someone else, we only learn to call the functions needed, but not deep enough. Thus starting from 2025 we are creating library for C++ that not only able to compute and solve problems in calculus, differential equations, linear programming, it grows and can also cover the field of linear algebra, so we are going to introduce a lot of features, functions in SymIntegration that can be used in Numerical Linear Algebra problems.

Instead of using open source library that already established, after branching out from **SymbolicC++**, SymIntegration can do a lot more, we learn more everyday, theoretically and practically, how to convert algorithms to a working C++ codes that can manage matrix multiplication, addition, LU decomposition, computing eigenvalues, and many more. If you want to read the full comprehensive summary read [5], in this book we will only cover the basic examples, and function in SymIntegration to solve Numerical Linear Algebra problems, there won't be theorems and explanations that are long here.

I. SOLUTIONS OF SYSTEMS OF LINEAR EQUATIONS

[SI*] Consider the nonhomogeneous linear system

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ \vdots &\quad \vdots & \vdots & \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \end{aligned} \tag{6.1}$$

it consists of m equations with n variables that need to be solved. We can also write it in matrix vector term like this:

$$Ax = b \tag{6.2}$$

with x is a vector of size n and b is a vector of size m . The entire solution set of $Ax = b$ can be obtained by translating the solution set of $Ax = 0$ by the vector x_0 .

When you have matrix of size $m \times n$, you just cannot apply Gaussian elimination directly to an $m \times n$ matrix problem. If you have more equations than unknowns ($m > n$), the your problem is overdetermined and you have no solution, which means you need to use something like the least squares method. If you have $Ax = b$, then instead of having $x = A^{-1}b$ (when $n = m$), then you have to do

$$x = (A^T A)^{-1} A^T b$$

In the case where you have less equations then unknowns ($m < n$), then your problem is underdetermined and you have an infinity of solutions, like the general solution of homogeneous linear system. In that case, you either pick one at random (e.g. setting some of the unknowns to an arbitrary value as parameter), or you need to use regularization, which means trying to add some extra constraints.

For the case when we have Eq. (6.1) with $m = n$ we will use Gaussian elimination, but when the size of the square matrix is very large, it is better and faster to use numerical method like *LU*-decomposition.

[SI*] We will list the basic functions that can be used in matrix, vector operations and loading textfile with SymIntegration:

loadMatrixFromFile(const string&)

this function will reads a textfile and return **vector<vector<double>** as the output, e.g. **vector<vector<double> matrixA loadMatrixFromFile("matrix.txt")** is a declaration of data type of **vector<vector<double>** with name **matrixA** and the data is taken from **matrix.txt** that is in the current working directory.

printMatrix(vector<vector<double>)

this function will display / print the matrix that has the data type of **vector<vector<double>**.

printVector(vector<double>)

this function will display / print a vector that has the data type of **vector<double>**.

getRow(vector<vector<double>, int)

this function extracts a certain row from a matrix.

getColumn(vector<vector<double>, int)

this function extracts a certain column from a matrix.

The indices will be starting from 0 for the first row / first column, so if you want to take the first row you will type like this **getRow(matrixA,0)**, with **matrixA** is the matrix with **vector<vector<double>** data type.

```
#include<bits/stdc++.h>
#include<iostream>
#include "symintegrationc++.h"
#include<vector>
```

```

#include <chrono>
#include <string>
#include <iomanip> // For std::setprecision
using namespace std::chrono;
using namespace std;

// Driver program
int main()
{
    vector<vector<double>> MatrixA = loadMatrixFromFile("matrix.txt");

    int R = MatrixA.size();
    int C = MatrixA[0].size();

    cout << "Column-1" << endl;
    for (int i = 0; i < R; i++)
    {
        // take the first column
        cout << getColumn(MatrixA, 0)[i] << endl;
    }

    cout << "Row-2" << endl;
    for (int i = 0; i < C; i++)
    {
        // take the second row
        cout << getRow(MatrixA, 1)[i] << endl;
    }

    cout << "Vector from row 1:" << endl;
    vector<double> vectora = getRow(MatrixA, 0);
    printVector(vectora);

    return 0;
}

```

Code 48: Test SymIntegration `getRow` and `getColumn` Example/main.cpp

multiply(vector<vector<double>>, vector<vector<double>>)

this function will perform multiplication of two matrices represented by `vector<vector<double>>`. For matrices multiplication, order matters, thus in general for non-similar matrices A and B , $AB \neq BA$.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Multiplication ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lsymintration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Multiplication ]# ./main
in
A:
     1.000000    2.000000
     3.000000    4.000000
     0.000000    1.000000

B:
     4.000000    3.000000
     2.000000    1.000000

A*B:
     8.000000    5.000000
    20.000000   13.000000
     2.000000    1.000000

Time taken by function: 977 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Multiplication ]# []

```

Figure 6.1: The multiplication between two matrices, we load the matrices from textfiles *matrixA.txt* and *matrixB.txt* (*SymIntegration/Examples/Test SymIntegration Matrices Multiplication/main.cpp*).

add(vector<vector<double>, vector<vector<double>)

this function will perform addition of two matrices represented by `vector<vector<double>`.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Addition ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lsymintration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Addition ]# ./main
in
A:
     4.000000    3.000000
     2.000000    1.000000

B:
     1.000000    0.000000
     2.000000    3.000000

A + B:
     5.000000    3.000000
     4.000000    4.000000

Time taken by function: 1074 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Matrices Addition ]# []

```

Figure 6.2: The addition between two matrices, we load the matrices from textfiles *matrixA.txt* and *matrixB.txt* (*SymIntegration/Examples/Test SymIntegration Matrices Addition/main.cpp*).

transpose(const vector<vector<double> &)

this function will perform transpose operation of a matrix represented by `vector<vector<double>`.

dmat to replace `vector<vector<double>`

We create a typedef defined in `include/symintral/symintrationc++.h` to represents `vector<vector<double>`, so we don't need to write `vector<vector<double>` anymore, we can use `dmat` instead.

The compiler will process the typedef declaration when compiling each .cpp file that includes `include/symintral/symintrationc++.h`, making the aliased types available for use within those files.

i. Solve Nonhomogeneous System of Linear Equation with Gaussian Elimination in SymIntegration

Suppose we have a nonhomogeneous linear system like this:

$$\begin{aligned} 0.0333x_1 + 0.2x_2 &= 1 \\ 0.1x_1 + 0.4x_2 &= 1 \end{aligned}$$

In the form of matrix and vector following the Eq. (6.2) we will have:

$$A = \begin{bmatrix} 0.0333 & 0.2 \\ 0.1 & 0.4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

We are going to solve for

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

In SymIntegration to solve a nonhomogeneous linear system with Gaussian elimination and back substitution you can use:

solve_nhsystem(vector<vector<double> &, vector<vector<double> &, vector<double> &)
it is a **void** function that returns nothing, and you can still call the solution after the function is called. The source code that handles this computation is located in **src/linearalgebra.cpp**.

Algorithm for **solve_nhsystem**:

1. Read input from textfiles of **matrix.txt** for matrix A and **vectorb.txt** for vector b .
2. The function will perform a merge for the matrix A and vector b to become an augmented matrix.
3. Then the function will perform forward elimination, that will iterate through each row to create zeros below the main diagonal. Ensuring for each pivot element (diagonal element) it will be non-zero. If it is zero, the function will swap the current row with a row below that has a non-zero element in the same column (partial pivoting). If no such row exists, then it indicates that there is no unique solution. The rank of the matrix is not full. Then for each row below the current pivot row, it calculates a ratio and performs row operations to make the element below the pivot zero.
4. After forward elimination, the function will perform back substitution to obtain the solution. Once the matrix is in row echelon form (upper triangular), it solves for the last variable directly, then it substitutes the value of the last variable into the second-to-last equation to solve the second-to-last variable, and so on, working upwards till x_1 .

When we code with C++ we need to remember that there is some truncation error, and precision error, like floating-point inaccuracies when we comparing values to zero or performing divisions.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Mat
tion Void Function for Square Matrix Load from Textfiles ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Mat
rix Load from Textfiles ]# ./main
Augmented Matrix:
  0.033300      0.200000      1.000000
  0.100000      0.400000      1.000000

Augmented Matrix in reduced row form:
  0.100000      0.400000      1.000000
  0.000000      0.066800      0.667000

Solution:
x1 = 9.98503
x2 = -29.94012

Time taken by function: 1151 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Mat
rix Load from Textfiles ]# ./main
Augmented Matrix:
  0.033300      0.200000      1.000000
  0.100000      0.400000      1.000000

Augmented Matrix in reduced row form:
  0.100000      0.400000      1.000000
  0.000000      0.066800      0.667000

Solution:
x1 = 9.98503
x2 = -29.94012

Time taken by function: 1159 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Mat
rix Load from Textfiles ]# ./main
Augmented Matrix:
  0.033300      0.200000      1.000000
  0.100000      0.400000      1.000000

Augmented Matrix in reduced row form:
  0.100000      0.400000      1.000000
  0.000000      0.066800      0.667000

Solution:
x1 = 9.98503
x2 = -29.94012

Time taken by function: 978 microseconds

```

Figure 6.3: The Gaussian elimination with forward elimination and back substitution, we load the matrix from textfiles *matrix.txt* and *vectorb.txt*. We run the code 3 times to see the running time of the function from 978 microseconds to 1151 microseconds (*SymIntegration/Examples/Test SymIntegration Gaussian Elimination Void Function for Square Matrix Load from Textfiles/main.cpp*).

We use **void** function for this Gaussian elimination because in C++ a void function is a function that does not return any value to the calling code. Instead of a specific data type (like int, double, string) as its return type, some functions in SymIntegration uses **void** for ease of use, it can be used for printing output to the console or a file, modifying global variables or variables passed by reference, performing calculations and storing results in external structures, and calling other functions to achieve a specific task.

We also compare it with Armadillo' function **solve(A,B,solve_opts::force_approx)**, turns out we can gain faster execution time, it is probably because we code it ourselves and can make it more efficient to return the solution of x . In Armadillo you need to print the solution yourself. In SymIntegration currently, we are printing out the augmented matrix, the reduced row form of the augmented matrix, and the solution when you execute / run the function **solve_nhsystem**.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimina
tion Void Function for Square Matrix Load from Textfiles/Gaussian Elimination fo
r Square Matrix with Armadillo ]# make
g++ -c -o main.o main.cpp
g++ -o main -gdb main.o -lstdc++ -larmadillo
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Ma
trix Load from Textfiles/Gaussian Elimination for Square Matrix with Armadillo ]# ./main
Matrix A:
 0.0333  0.2000
 0.1000  0.4000

Vector B:
 1.0000
 1.0000

Solution:
-29.9401
 9.9850

Time taken by function: 1551 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination Void Function for Square Ma
trix Load from Textfiles/Gaussian Elimination for Square Matrix with Armadillo ]#

```

Figure 6.4: The comparison of solving a linear system with Armadillo, we load the matrix from textfiles *matrixA.txt* and *vectorB.txt*. (DianFreya Math Physics Simulator/Source Codes/C++/C++ Gnuplot SymbolicC++/ch23-Numerical Linear Algebra/Gaussian Elimination for Square Matrix with Armadillo/main.cpp).

The code that is using Armadillo (so the reader can check it themselves):

```

#include <iostream>
#include <iomanip> // to declare the manipulator of setprecision()
#include <fstream>
#include <bits/stdc++.h> //for setw(6) at display() function
#include <vector>
#include <armadillo>
#include <chrono>

using namespace std::chrono;
using namespace std;
using namespace arma;

// Driver code
int main(int argc, char** argv)
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    mat A;
    A.load("matrixA.txt");
    mat B;
    B.load("vectorB.txt");
    mat X;
    X = solve(A,B,solve_opts::force_approx);

    cout <<"Matrix A:" << "\n" << A << endl;
    cout <<"Vector B:" << "\n" << B << endl;
    cout <<"Solution:" << "\n" << X << endl;

    // Get ending timepoint

```

```
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);

cout << "\nTime taken by function: " << duration.count() << " microseconds"
    << endl;

}
```

Code 49: *main.cpp*

II. DETERMINANT

[SI*] Suppose we have a 2×2 matrix A

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

the matrix A is invertible if $ad - bc \neq 0$, and the expression $ad - bc$ is called the determinant.

$$\det(A) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Thus, the inverse of A can be expressed in terms of the determinant as

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \quad (6.3)$$

Now, we will find the formula that applicable to square matrices of all orders, not only of size 2.

Definition 6.1: Minor and Cofactor

If A is a square matrix, then the minor of entry a_{ij} is denoted by M_{ij} and is defined to be the determinant of the submatrix that remains after the i th row and j th column are deleted from A . The number

$$C_{ij} = (-1)^{i+j} M_{ij} \quad (6.4)$$

is called the cofactor of entry a_{ij} .

Theorem 6.1: Row or Column of Cofactors

If A is an $n \times n$ matrix, then regardless of which row or column of A is chosen, the number obtained by multiplying the entries in that row or column by the corresponding cofactors and adding the resulting products is always the same.

Definition 6.2: Cofactor Expansion

If A is an $n \times n$ matrix, then the number obtained by multiplying the entries in any row or column of A by the corresponding cofactors and adding the resulting products is called the determinant of A , and the sums themselves are called cofactor expansions of A . That is,

$$\det(A) = a_{1j}C_{1j} + a_{2j}C_{2j} + \cdots + a_{nj}C_{nj} \quad (6.5)$$

(cofactor expansion along the j th column)

$$\det(A) = a_{i1}C_{i1} + a_{i2}C_{i2} + \cdots + a_{in}C_{in} \quad (6.6)$$

(cofactor expansion along the i th row)

[SI*] To find the determinant of size n that is very large will need more computational cost, since we will peel the cofactor expansion along the i th row or j th column down to the matrix of size 2×2 . It is an ordered-recursive computation technique.

[SI*] The concept of a determinant is mathematically defined only for square matrices. Non-square matrices, also known as rectangular matrices, do not possess a determinant in the conventional sense.

The determinant of a square matrix represents the scaling factor of the volume (or area in 2D) when a linear transformation is applied. This geometric is inherently tied to dimensions being equal. The determinant is also crucial for calculating the inverse of a matrix, which is only defined for square, non-singular matrices.

[SI*] Alternative Concepts for Non-Square Matrices

While a determinant doesn't exist for non-square matrices, related concepts can be used in certain contexts:

- **Singular Values**

Non-square matrices have singular values, which are related to the scaling of principal axes in the transformation defined by the matrix. The product of singular values can be related to a "volume" in a generalized sense.

- **Gramian Determinant**

For a matrix A , the Gramian matrix $A^T A$ (or AA^T) is always square. Its determinant, known as the Gramian determinant, can be used in some applications involving non-square matrices.

i. Compute Determinant of a Square Matrix with SymIntegration

Suppose we have a matrix A

$$A = \begin{bmatrix} 3 & -2 & 7 \\ -2 & 4 & -3 \\ -1 & 9 & 4 \end{bmatrix}$$

to compute the determinant with SymIntegration we will use this function:
determinant(const vector<vector<double> &matrix)

The full C++ source code can be seen below:

```
#include<bits/stdc++.h>
#include<iostream>
#include "symintegrationc++.h"
#include<vector>
#include <chrono>
#include <algorithm> // For std::next_permutation
#include <string>
using namespace std::chrono;
using namespace std;

// Driver program
int main()
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    //string filename = "matrix.txt";

    vector<vector<double>> doubleMatrix = loadMatrixFromFile("matrix.txt");

    cout << "\nMatrix A : " << endl;
    printMatrix(doubleMatrix);
    cout << "\nDeterminant of matrix A : " << determinant(doubleMatrix) << endl
        ;

    // Get ending timepoint
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "\nTime taken by function: " << duration.count() << " microseconds"
        << endl;

    return 0;
}
```

Code 50: Test SymIntegration Compute Determinant/main.cpp

```
TIME TAKEN BY FUNCTION: 751 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Compute Determinant ]# ./main

Matrix A :
 3.000000   -2.000000    7.000000
 -2.000000    4.000000   -3.000000
 -1.000000    9.000000    4.000000

Determinant of matrix A : 9.000000

Time taken by function: 751 microseconds
```

Figure 6.5: The computation of the determinant of matrix A (*SymIntegration/Examples/Test SymIntegration Compute Determinant/main.cpp*).

III. EUCLIDEAN VECTOR SPACES

[SI*] Engineers and physicists represent vectors in two dimensions or in three dimensions by arrows. The direction of the arrowhead specifies the direction of the vector and the length of the arrow specifies the magnitude. The tail of the arrow is called the initial point of the vector and the tip the terminal point.

We usually denote vectors with boldface such as $\mathbf{a}, \mathbf{b}, \mathbf{v}, \mathbf{w}$ or with an arrow above it $\vec{a}, \vec{b}, \vec{v}, \vec{w}$, and we denote scalars in lowercase letters like a, b, c, d . When we want to indicate that a vector has initial point A and terminal point B , we will write

$$\mathbf{v} = \vec{AB}$$

[SI*] We denote the length of a vector \mathbf{v} by the symbol $||\mathbf{v}||$, which is read as norm of \mathbf{v} , or the magnitude of \mathbf{v} .

Definition 6.3: Norm

If $\mathbf{v} = (v_1, v_2, \dots, v_n)$ is a vector in \mathbb{R}^n , then the norm of \mathbf{v} is denoted by $||\mathbf{v}||$, and is defined by the formula

$$||\mathbf{v}|| = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2} \quad (6.7)$$

[SI*] A vector of norm 1 is called a unit vector. Such vectors are useful for specifying a direction when length is not relevant to the problem at hand. If \mathbf{v} is any nonzero vector in \mathbb{R}^n , then

$$\mathbf{u} = \frac{1}{||\mathbf{v}||} \mathbf{v} \quad (6.8)$$

defines a unit vector that is in the same direction as \mathbf{v} . The process of multiplying a nonzero vector by the reciprocal of its length to obtain a unit vector is called normalizing \mathbf{v} .

Definition 6.4: Distance in \mathbb{R}^n

If $\mathbf{u} = (u_1, u_2, \dots, u_n)$ and $\mathbf{v} = (v_1, v_2, \dots, v_n)$ are points in \mathbb{R}^n , then we denote the distance between \mathbf{u} and \mathbf{v} by $d(\mathbf{u}, \mathbf{v})$ and define it to be

$$d(\mathbf{u}, \mathbf{v}) = ||\mathbf{u} - \mathbf{v}|| = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \dots + (u_n - v_n)^2} \quad (6.9)$$

Definition 6.5: Angle Between Two Vectors in \mathbb{R}^2 and \mathbb{R}^3

If u and v are nonzero vectors in \mathbb{R}^2 or \mathbb{R}^3 , and if θ is the angle between u and v , then the dot product of u and v is denoted by $v \cdot v$ and is defined as

$$u \cdot v = \|u\| \|v\| \cos \theta \quad (6.10)$$

If $u = 0$ or $v = 0$, then we define $u \cdot v$ to be 0.

Since $0 \leq \theta \leq \pi$, it follows from the properties of the cosine function that

- θ is acute if $u \cdot v > 0$.
- θ is obtuse if $u \cdot v < 0$.
- $\theta = \frac{\pi}{2}$ if $u \cdot v = 0$.

Definition 6.6: Dot Product in \mathbb{R}^n

If u and v are vectors in \mathbb{R}^n , then the dot product or the Euclidean inner product of u and v is denoted by $u \cdot v$ and is defined by

$$u \cdot v = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n \quad (6.11)$$

Definition 6.7: Orthogonal and Orthonormal

Two nonzero vectors u and v in \mathbb{R}^n are said to be orthogonal (or perpendicular) if $u \cdot v = 0$.

The zero vector (0) in \mathbb{R}^n is orthogonal to every vector in \mathbb{R}^n .

A nonempty set of vectors in \mathbb{R}^n is called an orthogonal set if all pairs of distinct vectors in the set are orthogonal.

An orthogonal set of unit vectors is called an orthonormal set.

[SI*] The vector equation

$$\mathbf{n} \cdot \vec{P_0 P} = 0$$

is the representation of a line through the point $P_0(x_0, y_0)$ that has normal $\mathbf{n} = (a, b)$ or a plane through the point $P_0(x_0, y_0, z_0)$ that has normal $\mathbf{n} = (a, b, c)$.

For line, the vector $\vec{P_0 P}$ can be expressed in terms of components as

$$\vec{P_0 P} = (x - x_0, y - y_0)$$

For plane, the vector $\vec{P_0 P}$ can be expressed in terms of components as

$$\vec{P_0 P} = (x - x_0, y - y_0, z - z_0)$$

[SI*] The point-normal equation of the line:

$$a(x - x_0) + b(y - y_0) = 0 \quad (6.12)$$

Theorem 6.2: Equation for Line and Plane

If a and b are constants that are not both zero, then an equation of the form

$$ax + by + c = 0 \quad (6.13)$$

represents a line in \mathbb{R}^2 with normal $\mathbf{n} = (a, b)$.

If a, b , and c are constants that are not all zero, then an equation of the form

$$ax + by + cz + d = 0 \quad (6.14)$$

represents a plane in \mathbb{R}^3 with normal $\mathbf{n} = (a, b, c)$.

[SI*] There are other useful ways of specifying lines and planes.

A unique line in \mathbb{R}^2 or \mathbb{R}^3 is determined by a point x_0 on the line and a nonzero vector v parallel to the line.

A unique plane in \mathbb{R}^3 is determined by a point x_0 in the plane and two collinear vectors v_1 and v_2 parallel to the plane. The best way to visualize this is to translate the vectors so their initial points are at x_0 .

Theorem 6.3: Vector and Parametric Equations of Lines in \mathbb{R}^2 and \mathbb{R}^3

Let L be the line in \mathbb{R}^2 or \mathbb{R}^3 that contains the point x_0 and is parallel to the nonzero vector v . Then the equation of the line through x_0 that is parallel to v is

$$\mathbf{x} = \mathbf{x}_0 + t\mathbf{v} \quad (6.15)$$

If $\mathbf{x}_0 = \mathbf{0}$, then the line passes through the origin and the equation has the form

$$\mathbf{x} = t\mathbf{v} \quad (6.16)$$

the variable t is called a parameter, and it varies from $-\infty$ to ∞ .

Theorem 6.4: Vector and Parametric Equations of Planes in \mathbb{R}^3

Let W be the plane in \mathbb{R}^3 that contains the point x_0 and is parallel to the noncollinear vectors v_1 and v_2 . Then an equation of the plane through x_0 that is parallel to v_1 and v_2 is given by

$$\mathbf{x} = \mathbf{x}_0 + t_1\mathbf{v}_1 + t_2\mathbf{v}_2 \quad (6.17)$$

If $\mathbf{x}_0 = \mathbf{0}$, then the plane passes through the origin and the equation has the form

$$\mathbf{x} = t_1\mathbf{v}_1 + t_2\mathbf{v}_2 \quad (6.18)$$

Definition 6.8: Parametric Equations of Lines in \mathbb{R}^n

If x_0 and v are vectors in \mathbb{R}^n , and if v is nonzero, then the equation

$$x = x_0 + tv \quad (6.19)$$

defines the line through x_0 that is parallel to v . In the special case where $x_0 = \mathbf{0}$, the line is said to pass through the origin.

Definition 6.9: Parametric Equations of Planes in \mathbb{R}^n

If x_0, v_1 and v_2 are vectors in \mathbb{R}^n , and if v_1 and v_2 are not collinear, then the equation

$$x = x_0 + t_1v_1 + t_2v_2 \quad (6.20)$$

defines the plane through x_0 that is parallel to v_1 and v_2 . In the special case where $x_0 = \mathbf{0}$, the plane is said to pass through the origin.

Definition 6.10: Cross Product

If $u = (u_1, u_2, u_3)$ and $v = (v_1, v_2, v_3)$ are vectors in 3-space, then the cross product $u \times v$ is the vector defined by

$$u \times v = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1) \quad (6.21)$$

or, in determinant notation,

$$u \times v = \left(\begin{vmatrix} u_2 & u_3 \\ v_2 & v_3 \end{vmatrix}, - \begin{vmatrix} u_1 & u_3 \\ v_1 & v_3 \end{vmatrix}, \begin{vmatrix} u_1 & u_2 \\ v_1 & v_2 \end{vmatrix} \right) \quad (6.22)$$

Definition 6.11: Scalar Triple Product

If u, v , and w are vectors in 3-space, then

$$u \cdot (v \times w) = \begin{vmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{vmatrix} \quad (6.23)$$

is called the scalar triple product of $u = (u_1, u_2, u_3), v = (v_1, v_2, v_3)$, and $w = (w_1, w_2, w_3)$.

It follows that

$$u \cdot (v \times w) = w \cdot (u \times v) = v \cdot (w \times u)$$

Theorem 6.5: Geometric Interpretation of Determinants

(a) The absolute value of the determinant

$$\det \begin{bmatrix} u_1 & u_2 \\ v_1 & v_2 \end{bmatrix}$$

is equal to the area of the parallelogram in 2-space determined by the vectors $\mathbf{u} = (u_1, u_2)$ and $\mathbf{v} = (v_1, v_2)$.

(b) The absolute value of the determinant

$$\det \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

is equal to the volume of the parallelepiped in 3-dimension determined by the vectors $\mathbf{u} = (u_1, u_2, u_3)$, $\mathbf{v} = (v_1, v_2, v_3)$, and $\mathbf{w} = (w_1, w_2, w_3)$.

[SI*] We will list the basic functions that can be used related to Euclidean vector spaces:

distance(const vector<double> &vectorx, const vector<double> &vectory)

to compute the distance between two points (the point can be represented as n -tuple / array / **vector<double>**).

distance(const Symbolic &f, const Symbolic &x, const Symbolic &y, const Symbolic &z, const vector<double> &vectorx)

to compute the distance between a plane / a line that is defined with function $f(x, y, z)$ / $f(x, y)$ and a point that is represented as vector \mathbf{x} .

dot(const vector<double> &vectorx, const vector<double> &vectory)

to compute the dot product / Euclidean inner product of two vectors.

angle(const vector<double> &vectorx, const vector<double> &vectory)

to compute the angle between two vectors.

norm(const vector<double> &vectorx)

to compute the norm of a vector.

add(vector<double> &vectorA, vector<double> &vectorB)

to compute the addition between two vectors.

subtract(vector<double> &vectorA, vector<double> &vectorB)

to compute subtraction between two vectors.

scalarmultiplication(vector<double> &vectorA, double k)

to compute the scalar multiplication of a vector.

orthogonalprojection(vector<double> vectorx, vector<double> vectory)

to compute the orthogonal projection of x on y .

vectorequation(const Symbolic &f, const Symbolic &x, const Symbolic &y, const Symbolic &z)

to compute the vector equation of the form $x = x_0 + tv_1 + sv_2$ that represents a plane / a line through x_0 and parallel to v_1 and/or v_2 (for a line we will only have one v) with given input of a function $f(x, y, z) = 0$.

vectorequationdecomp(const Symbolic &f, const Symbolic &x, const Symbolic &y, const Symbolic &z, vector<double> &vectorv1, vector<double> &vectorv2, vector<double> &vectorv3)

to compute all the vectors that compose the vector equation / parametric equations of the form $x = x_0 + tv_1 + sv_2$ that represents a plane / a line through x_0 and parallel to v_1 and/or v_2 (for a line we will only have one v) with given input of a function $f(x, y, z) = 0$. It is a **void** function, so we will need to declare the vector v_1, v_2 , and v_3 before calling the function, afterwards we can print the vectors.

crossproduct(vector<double> &vectoru, vector<double> &vectorv)

to compute the cross product of $u \times v$.

scalartripleproduct(vector<double> &vectoru, vector<double> &vectorv, vector<double> &vectorw)

to compute the scalar triple product of u, v , and w .

i. Compute Norm, Dot Product, Angle with SymIntegration

Suppose we have two vectors :

$$\mathbf{x} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{y} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}$$

we can compute the norm, dot product, and the angle between \mathbf{x} and \mathbf{y} with SymIntegration' functions.

$$\begin{aligned} \|\mathbf{x}\| &= 1 \\ \|\mathbf{y}\| &= 2.82843 \\ \mathbf{x} \cdot \mathbf{y} &= 2 \\ \theta &= \cos^{-1} \left(\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} \right) = 0.70710678 \text{ rad} = 45^0 \end{aligned}$$

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Euclidean Vector Spaces ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Euclidean Vector Spaces ]# ./main

Vector x :
0
0
1

Vector y :
0
2
2

norm(x) : 1
norm(y) : 2.82843
dot(x,y) : 2
angle(x,y) in degree: 45

Time taken by function: 689 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Euclidean Vector Spaces ]# ]
```

Figure 6.6: The computation of the norm, dot product, and angle of two vectors \mathbf{x} and \mathbf{y} (SymIntegration/Examples/Test SymIntegration Euclidean Vector Spaces/main.cpp).

ii. Compute Orthogonal Projection on a Line with SymIntegration

Let $\mathbf{u} = (2, -1, 3)$ and $\mathbf{a} = (4, -1, 2)$. Find the vector component of \mathbf{u} along \mathbf{a} and the vector component of \mathbf{u} orthogonal to \mathbf{a} .

Solution:

$$\begin{aligned}\mathbf{u} \cdot \mathbf{a} &= (2)(4) + (-1)(-1) + (3)(2) = 15 \\ ||\mathbf{a}||^2 &= 4^2 + (-1)^2 + 2^2 = 21\end{aligned}$$

Then the vector component of \mathbf{u} along \mathbf{a} is

$$\begin{aligned}\text{proj}_{\mathbf{a}} \mathbf{u} &= \frac{\mathbf{u} \cdot \mathbf{a}}{||\mathbf{a}||^2} \mathbf{a} \\ &= \frac{15}{21} (4, -1, 2) \\ &= \left(\frac{20}{7}, -\frac{5}{7}, \frac{10}{7} \right)\end{aligned}$$

and the vector component of \mathbf{u} orthogonal to \mathbf{a} is

$$\begin{aligned}\mathbf{u} - \text{proj}_{\mathbf{a}} \mathbf{u} &= (2, -1, 3) - \left(\frac{20}{7}, -\frac{5}{7}, \frac{10}{7} \right) \\ &= \left(-\frac{6}{7}, -\frac{2}{7}, \frac{11}{7} \right)\end{aligned}$$

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Orthogonal Projection ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Orthogonal Projection ]# ./main

Vector u :
2
-1
3

Vector a :
4
-1
2

norm(a)^2 : 29.6985
dot(u,a) : 15

Orthogonal projection of u on a:
2.85714
-0.714286
1.42857

Vector component of u orthogonal to a:
-0.857143
-0.285714
1.57143

u + a :
6
-2
5

u - a :
-2
0
1

5*u :
10
-5
15

Time taken by function: 1066 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Orthogonal Projection ]# []

```

Figure 6.7: The computation of the orthofonal projection of vector u on a and some other vector operations. (*SymIntegration/Examples/Test SymIntegration Orthogonal Projection/main.cpp*).

iii. Compute Vector and Parametric Equations of a Plane in \mathbb{R}^3 with SymIntegration

Find vector and parametric equations of the plane $x - y + 2z = 5$.

Solution:

We will find the parametric equations first. We can do this by solving the equation for any one of the variables in terms of the other two and then using those two variables as parameters.

For example, solving for x in terms of y and z yields

$$x = 5 + y - 2z$$

and then using y and z as parameters t and s , respectively, yields the parametric equations

$$\begin{aligned}x &= 5 + t - 2s \\y &= t \\z &= s\end{aligned}$$

To obtain a vector equation of the plane we rewrite these parametric equations as

$$(x, y, z) = (5 + t - 2s, t, s)$$

or, equivalently, as

$$(x, y, z) = (5, 0, 0) + t(1, 1, 0) + s(-2, 0, 1)$$

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Vector and Parametric Equations ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Vector and Parametric Equations ]# ./main

Vector equation of the plane x - y + 2z - 5 = 0:
[t-2*s+5]
[   t   ]
[   s   ]

The parametric equations in vector terms:
(x,y,z) = v1 + v2*t + v3*s
v1 =
5
0
0

v2 =
1
1
0

v3 =
-2
0
1

Time taken by function: 29519 microseconds
```

Figure 6.8: The computation of vector equation of the plane in parametric equations term and then in vector terms. (SymIntegration/Examples/Test SymIntegration Vector and Parametric Equations/main.cpp).

IV. GENERAL VECTOR SPACES

[SI*] In Euclidean Vector Spaces we developed properties of vectors in \mathbb{R}^2 and \mathbb{R}^3 , we noticed patterns in various formulas that enabled us to extend the notion of a vector to an n -tuple of real numbers. Although n -tuple took us outside the realm of our "visual experience," it gave us a valuable tool for understanding and studying systems of linear equations.

Definition 6.12: Vector Space Axioms

Let V be an arbitrary nonempty set of objects on which two operations are defined: addition, and multiplication by scalars.

- By addition we mean a rule for associating with each pair of objects u and v in V an object $u + v$, called the sum of u and v .
- By scalar multiplication we mean a rule for associating with each scalar k and each object u in V an object ku , called the scalar multiple of u by k .

If the following axioms are satisfied by all objects u, v, w in V and all scalars k and m , then we call V a vector space and we call the objects in V vectors

1. If u and v are objects in V , then $u + v$ is in V
2. $u + v = v + u$
3. $u + (v + w) = (u + v) + w$
4. There is an object $\mathbf{0}$ in V , called a zero vector for V , such that

$$\mathbf{0} + u = u + \mathbf{0} = u$$

for all u in V

5. For each u in V , there is an object $-u$ in V , called a negative of u , such that

$$u + (-u) = (-u) + u = \mathbf{0}$$

6. If k is any scalar and u is any object in V , then ku is in V .
7. $k(u + v) = ku + kv$
8. $(k + m)u = ku + mu$
9. $k(mu) = (km)(u)$
10. $1u = u$

Definition 6.13: Subspace

A subset W of a vector space V is called a subspace of V if W is itself a vector space under the addition and scalar multiplication defined on V .

In general, to show that a nonempty set W with two operations is a vector space one must verify the ten vector space axioms.

Definition 6.14: Linear Combination

If w is a vector in a vector space V , then w is said to be a linear combination of the vectors v_1, v_2, \dots, v_r in V if w can be expressed in the form

$$w = k_1 v_1 + k_2 v_2 + \cdots + k_r v_r \quad (6.24)$$

where k_1, k_2, \dots, k_r are scalars. These scalars are called the coefficients of the linear combination.

Theorem 6.6: Linear Combination and Subspace

If $S = \{w_1, w_2, \dots, w_r\}$ is a nonempty set of vectors in a vector space V , then:

1. The set W of all possible linear combinations of the vectors in S is a subspace of V .
2. The set W in part (a) is the "smallest" subspace of V that contains all of the vectors in S in the sense that any other subspace that contains those vectors contains W .

Definition 6.15: Span of a Subspace

The subspace of a vector space V that is formed from all possible linear combinations of the vectors in a nonempty set S is called the span of S , and we say that the vectors in S span that subspace. If $S = \{w_1, w_2, \dots, w_r\}$, then we denote the span of S by

$$\text{span}\{w_1, w_2, \dots, w_r\} \quad \text{or} \quad \text{span}(S)$$

[SI*] We will list the basic functions that can be used related to general vector spaces:

V. EIGENVALUES AND EIGENVECTORS

[SI*] We will list the basic functions that can be used related to eigenvalues and eigenvectors:

VI. INNER PRODUCT SPACES

[SI*] We will list the basic functions that can be used related to inner product spaces:

VII. DIAGONALIZATION AND QUADRATIC FORMS

[SI*] We will list the basic functions that can be used related to diagonalization and quadratic forms:

VIII. LINEAR TRANSFORMATIONS

[SI*] We will list the basic functions that can be used related to linear transformations:

IX. NUMERICAL METHODS

[SI*] From the first section on this chapter of Numerical Linear Algebra, we have learned about Gaussian elimination and backward substitution to obtain the solution for systems of linear equations, they are fine for small-scale problems, but Gaussian elimination and Gauss-Jordan elimination are not suitable for large-scale problems in which compute roundoff error, memory usage, and speed are concerns. For large-scale and special case we can use numerical methods to solve systems of linear equations, these are the available methods currently:

1. LU Decomposition

A matrix factorization technique into lower triangular L and upper triangular U , it is used to solve systems of linear equations. It transforms a complex matrix problem into two simpler problems that can be solved sequentially using forward and backward substitution.

2. QR Decomposition

more stable than LU decomposition, but slower to converge.

3. Cholesky Decomposition

It is used if the system is symmetric.

4. Singular Value Decomposition (SVD)

If the system is not square (the number of equations is not the same as the number of variables).

Decomposition has better efficiency, for a given matrix A , the costly decomposition step is performed only once. This is faster for repeated solving of $Ax = b$ with different b vectors compared to reinverting the matrix each time.

i. LU-Decomposition

The credit for popularizing the matrix formulation of the LU -decomposition is often given to the British mathematician Alan Turing for his work on the subject in 1948. Turing, one of the great geniuses of the twentieth century is the founder of the field of artificial intelligence. Sadly, Turing, a homosexual, was tried and convicted of "gross indecency" in 1952, in violation of the then-existing British statutes. Depressed, he committed suicide at age 41 by eating an apple laced with cyanide. Lately before passed away in the 21st century, Queen Elizabeth II has done public apology for what has happened for Alan Turing, a world war II hero, without him Nazi cannot be stopped, this shows how close minded people in the past are, they read less, comprehend less, but

judge more, just like the time when church burned Jeanne D'arc, or burn Galileo Galilei paper works, then apology after hundred of years and made Jeanne D'arc a saint. It is human nature to judge what they don't know, out of fear, people are different just like the world full of color, so by learning science, computer and engineering, hopefully the result will make people have better understanding, compassion, and have higher quality of life.

Definition 6.16: LU-Decomposition

A factorization of a square matrix A as $A = LU$, where L is lower triangular and U is upper triangular is called an LU -decomposition (or LU -factorization) of A .

Not every square matrix has an LU -decomposition.

To systematically solve for entries in L and U from matrix A , the formula for matrix U :

$$\forall j \begin{cases} i = 0 \rightarrow U_{ij} = A_{ij} \\ i > 0 \rightarrow U_{ij} = A_{ij} - \sum_{k=0}^{i-1} L_{ik} U_{kj} \end{cases} \quad (6.25)$$

the formula for matrix L :

$$\forall i \begin{cases} j = 0 \rightarrow L_{ij} = \frac{A_{ij}}{U_{jj}} \\ j > 0 \rightarrow L_{ij} = \frac{A_{ij} - \sum_{k=0}^{j-1} L_{ik} U_{kj}}{U_{jj}} \end{cases} \quad (6.26)$$

In LU -decomposition we are using Doolittle algorithm, it is an efficient method for LU -decomposition, enabling various applications in numerical analysis and linear algebra. Applications on Doolittle algorithm:

1. Solving Linear Equations

LU -decomposition simplifies solving $Ax = b$ by solving $Lz = b$ then $Ux = y$.

2. Matrix Inversion

Decompose A into LU , then invert L and U .

3. Determinant Calculation

The determinant of A is the product of the diagonal elements of U . The formula is

$$\det(A) = \det(L) \times \det(U)$$

since $\det(L) = 1$, thus we only need to compute $\det(U)$.

With variations like partial pivoting, it creates numerical stability, to make it more robust against numerical errors that can occur during the process.

Applications of LU -decomposition:

1. Structural Engineering

It is used to analyze forces and stresses in bridges and buildings, ensuring efficient and safe designs then model the structure's behavior. It can be used with modal flexibility matrices to identify and locate damage in a structure by analyzing changes in its' structural properties.

2. Finite Element Analysis (FEA)

FEA relies heavily on matrix-based methods. *LU*-decomposition is used to solve the resulting stiffness matrices, which represent the relationship between forces and displacements in a structure.

3. Heat Transfer

It solves the systems of equations that result from discretizing partial differential equations used to model heat flow. It is used to solve for temperature distribution in objects when modeled using finite difference or finite element methods.

4. Fluid Dynamics

It is applied in computational fluid dynamics (CFD) to solve the pressure and velocity fields in fluid flow, which is often modeled by a system of linear equations.

5. Geotechnical Engineering

It can be applied in solving problems related to soil mechanics and foundation design that involve large systems of equations.

6. Computer Graphics

It helps in transforming 3D objects, like rotating and scaling models, for smoother rendering.

7. Robotics

It assists in solving kinematic equations, enabling real-time movement adjustments for robots.

8. Weather Prediction

It speeds up the solving of climate models and weather simulations for accurate forecasting.

9. Electrical Engineering

It is used in circuit analysis to solve systems of equations for designing and optimizing electrical circuits. *LU*-decomposition is a core component of the numerical iterative algorithms used to solve the power flow problem in power systems. It is used in methods like the Method of Moments for analyzing electromagnetic radiation and scattering, especially when a matrix needs to be decomposed repeatedly for different configurations, saving computation time. It can also be applied to matrix equations that arise in antenna design, particularly when optimizing parameters on a fixed geometry.

10. Modeling Complex Systems

Chemical processes often involve multiple interacting components, leading to a large system of linear equations that describe the process. *LU*-decomposition provides a structured way to solve these systems.

11. Steady-State and Transient Analysis

It is used in analyzing systems at steady-state and in transient simulations where the system evolves over time.

12. Chemical Reaction Networks

It is used for calculating concentrations of reactants and products in a system with multiple parallel or consecutive reactions.

13. Economics and Finance

It helps solve economic models for resource allocation and market predictions efficiently.

*Remember that *LU*-decompositions are not unique.

ii. Solve Nonhomogeneous System of Linear Equation with LU-Decomposition in SymIntegration

Solve the linear system

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & -1 \\ 3 & 5 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

Solution:

We can rewrite this system as

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

Let us define z_1, z_2 , and z_3 by the equation $Ux = z$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

then we can write the equation $Lz = b$

$$\begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 3 & -2 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ 4 \end{bmatrix}$$

the equation $Lz = b$ can be written equivalently as

$$\begin{array}{rcl} z_1 & = 1 \\ 4z_1 + z_2 & = 6 \\ 3z_1 - 2z_2 + z_3 & = 4 \end{array}$$

now by using forward substitution we will obtain

$$z_1 = 1, z_2 = 2, z_3 = 5$$

We go back to the equation $Ux = z$ and substitute z into the equation, which yields the linear system

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -1 & -5 \\ 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix}$$

or equivalently,

$$\begin{array}{rcl} x_1 + x_2 + x_3 & = 1 \\ -x_2 - 5x_3 & = 2 \\ -10x_3 & = 5 \end{array}$$

now by using back substitution we will obtain

$$x_1 = 1, x_2 = 0.5, x_3 = -0.5$$

In SymIntegration, to make the matrices L and U with LU-decomposition we can use this function:

`LUdecomposition(vector<vector<double> &A, vector<vector<double> &L, vector<vector<double> &U)`

the main source code for this function is located in `src/numericalmethod.cpp`.

```

#include<bits/stdc++.h>
#include<iostream>
#include "symintegrationc++.h"
#include<vector>
#include <chrono>
#include <algorithm> // For std::next_permutation
#include <string>
using namespace std::chrono;
using namespace std;

// Driver program
int main()
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    //string filename = "matrix.txt";

    dmat doubleMatrix = loadMatrixFromFile("matrix.txt");
    dmat vecb = loadMatrixFromFile("vectorb.txt");
    int n = doubleMatrix.size();
    int R = doubleMatrix.size();
    int C = doubleMatrix[0].size();

    // Declare the matrix L and U with the size.
    dmat L(n, vector<double>(n));
    dmat U(n, vector<double>(n));

    printMatrix(doubleMatrix);

    LUDecomposition(doubleMatrix, L, U);

    cout << "\nMatrix A : " << endl;
    printMatrix(doubleMatrix);
    cout << "\nMatrix L : " << endl;
    printMatrix(L);
    cout << "\nMatrix U : " << endl;
    printMatrix(U);

    // Get ending timepoint
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);

    cout << "\nTime taken by function: " << duration.count() << " microseconds"
        << endl;
}

```

```
    return 0;
}
```

Code 51: Test SymIntegration LU Decomposition/main.cpp

```
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Decomposition ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Decomposition ]# ./main
Data:
      1      1      1
      4      3     -1
      3      5      3

Matrix A :
 1.000000  1.000000  1.000000
 4.000000  3.000000 -1.000000
 3.000000  5.000000  3.000000

Matrix L :
 1.000000  0.000000  0.000000
 4.000000  1.000000  0.000000
 3.000000 -2.000000  1.000000

Matrix U :
 1.000000  1.000000  1.000000
 0.000000 -1.000000 -5.000000
 0.000000  0.000000 -10.000000

Time taken by function: 1485 microseconds
```

Figure 6.9: The LU-decomposition of matrix A (SymIntegration/Examples/Test SymIntegration LU Decomposition/main.cpp).

To find the solution with LU-decomposition, we use this function:

`LUsolve_nhsystem(vector<vector<double> &A, vector<vector<double> &b, vector<double> &x)`
the main source code for this function is located in `src/linearalgebra.cpp`.

```
#include<bits/stdc++.h>
#include<iostream>
#include "symintegrationc++.h"
#include<vector>
#include <chrono>
#include <algorithm> // For std::next_permutation
#include <string>

using namespace std::chrono;
using namespace std;

// Driver program
int main()
{
    // Get starting timepoint
    auto start = high_resolution_clock::now();

    dmat doubleMatrix = loadMatrixFromFile("matrix.txt");
    dmat vecb = loadMatrixFromFile("vectorb.txt");
    int n = doubleMatrix.size();

    cout << "Data:" << endl;
    printMatrix(doubleMatrix);

    dvec x;
```

```

LUsolve_nhsystem(doubleMatrix, vecb, x);
cout << "\nx : " << endl;
printVector(x);

// Get ending timepoint
auto stop = high_resolution_clock::now();
auto duration = duration_cast<microseconds>(stop - start);

cout << "\nTime taken by function: " << duration.count() << " microseconds"
    << endl;

return 0;
}

```

Code 52: Test SymIntegration LU Solve/main.cpp

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# make
g++
-c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# ./main
Data:
      1.000000      1.000000      1.000000
      4.000000      3.000000     -1.000000
      3.000000      5.000000      3.000000

Solution for Lz = b:
z1 = 1.00000
z2 = 2.00000
z3 = 5.00000

Solution for Ux = z:
x1 = 1.00000
x2 = 0.50000
x3 = -0.50000

x :
1.000000
0.500000
-0.500000

Time taken by function: 1142 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# 

```

Figure 6.10: Computing the solution for $Ax = b$ with LU-decomposition (SymIntegration/Examples/Test SymIntegration LU Solve/main.cpp).

We commented some **cout** commands and only show the solution vector x , it decreases the function time making it faster to show the final result.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# ./main
Data:
      1.000000      1.000000      1.000000
      4.000000      3.000000     -1.000000
      3.000000      5.000000      3.000000

x :
1.000000
0.500000
-0.500000

Time taken by function: 982 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration LU Solve ]# []

```

Figure 6.11: Computing the solution for $Ax = b$ with LU-decomposition and only show the solution vector x (*SymIntegration/Examples/Test SymIntegration LU Solve/main.cpp*).

We also compare it with Gaussian elimination function, to see which algorithm could compute the solution vector x faster, LU-decomposition is faster than Gaussian elimination.

```

root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimina
tion ]# make
g++ -c -o main.o main.cpp
g++ -o main -ggdb main.o -lstdc++ -lsymintegration
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination ]# ./main
Data:
      1.000000      1.000000      1.000000
      4.000000      3.000000     -1.000000
      3.000000      5.000000      3.000000

Augmented Matrix:
      1.000000      1.000000      1.000000      1.000000
      4.000000      3.000000     -1.000000      6.000000
      3.000000      5.000000      3.000000      4.000000

Augmented Matrix in reduced row form:
      4.000000      3.000000     -1.000000      6.000000
      0.000000      2.750000      3.750000     -0.500000
      0.000000      0.000000      0.969691     -0.454545

Solution:
x1 = 1.00000
x2 = 0.50000
x3 = -0.50000

x :
1.000000
0.500000
-0.500000

Time taken by function: 1293 microseconds
root [ ~/SourceCodes/CPP/C++ Create Library/Test SymIntegration Gaussian Elimination ]# []

```

Figure 6.12: Computing the solution for $Ax = b$ with Gaussian elimination (*SymIntegration/Examples/Test SymIntegration Gaussian Elimination Function/main.cpp*).

iii. Cholesky Decomposition

Cholesky decomposition or Cholesky factorization is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. The Cholesky decomposition is roughly twice as efficient as the LU decomposition for solving systems of linear equations. The Cholesky decomposition of a Hermitian positive-definite matrix Σ_X is a decomposition of the form

$$\Sigma_X = AA^T$$

where A is a real lower triangular matrix with positive diagonal entries, and A^T denotes the conjugate transpose of A . Every Hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition.

$$\begin{bmatrix} \Sigma_{X_{11}} & \Sigma_{X_{12}} & \Sigma_{X_{13}} \\ \Sigma_{X_{21}} & \Sigma_{X_{22}} & \Sigma_{X_{23}} \\ \Sigma_{X_{31}} & \Sigma_{X_{32}} & \Sigma_{X_{33}} \end{bmatrix} = \begin{bmatrix} A_{11} & 0 & 0 \\ A_{21} & A_{22} & 0 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} A_{11} & A_{21} & A_{31} \\ 0 & A_{22} & A_{32} \\ 0 & 0 & A_{33} \end{bmatrix}$$

Every symmetric, positive definite matrix Σ_X can be decomposed into a product of a unique lower triangular matrix L and its transpose.

Definition 6.17: Cholesky Decomposition Algorithm

The following formulas are obtained by solving above lower triangular matrix and its transpose. These are the basis of Cholesky decomposition algorithm:

$$A_{ii} = \sqrt{\Sigma_{X_{ii}} - \sum_{k=0}^{i-1} (A_{ik})^2}$$

Bibliography

- [1] Bazaraa, Mokhtar S., Jarvis, John J., Sherali, Hanif D. (1990) Linear Programming and Network Flows, John Wiley & Sons, New Jersey, United States.
- [2] Boyce, William E., DiPrima, Richard C. (2010) Elementary Differential Equations and Boundary Value Problems 9th Edition, John Wiley & Sons, Hoboken, New Jersey, United States.
- [3] Colley, Susan Jane (2012) Vector Calculus 4th Edition, Pearson, Boston, MA, United States.
- [4] Dorf, Richard C., Bishop, Robert H. (2010) Modern Control Systems 12th Edition, Pearson, New Jersey, United States.
- [5] Freya, Glanzsche, DS (2025) DianFreya Math Physics Simulator, Berlin-Sentinel Academy of Science, AliceGard.
- [6] Lasthrim, Glanzsche, DS, Freya (2025) Lasthrim Projection 1st Edition, Berlin-Sentinel Academy of Science, AliceGard.
- [7] Nazarathy, Yoni, Klok, Hayden, Statistics with Julia, Springer, Cham, Switzerland, 2021.
- [8] Purcell, Rigdon, Varberg (2007) Calculus 9th Edition, Pearson, Upper Saddle River, New Jersey, United States.
- [9] Kenneth H. Rosen (2006) Discrete Mathematics and its Applications 6th Edition, McGraw-Hill, Boston, United States.
- [10] Anton, Rorres (2006) Elementary Linear Algebra with Supplemental Applications 10th Edition, Wiley, Boston, United States.
- [11] Anton, Howard, Rorres, Chris, Elementary Linear Algebra with Supplemental Applications 12th edition, Wiley, Hoboken, New Jersey, United States.
- [12] Walpole, Ronald E., Myers, Raymond H., Myers, Sharon L., Ye, Keying, Probability & Statistics 9th Edition, Pearson, Boston, USA, 2012.