StatAnaly

A Statistical Analysis Package

Ansel Blumers

February 15, 2022

Contents

Chapter 1

README

Please see README.md from repository.

1.1 To-Do Features

1. Python API.

Chapter 2

Probability Distribution

2.1 List of supported probability distributions

StatAnaly supports the following probability distributions, in alphabetical order:

1. Cauchy distribution
2. Chi distribution
3. Chi Squared distribution
4. Erlang distribution
5. Exponential distribution
6. Gamma distribution
7. Irwin-Hall distribution
8. Non-central Chi distribution
9. Non-central Chi Squared distribution
10. Normal distribution
11. Rayleigh distribution
12. Standard Normal distribution
13. Uniform distribution (Continuous)

2.1.1 Cauchy distribution

PDF

$$P(x) = \frac{1}{\pi} \frac{b}{(x-m)^2 + b^2}$$

where b is the scale parameter which specifies the half-width at half-maximum; m is the location parameter which specifies the location of the peak of the distribution.

CDF

$$D(x) = \frac{1}{\pi} \arctan\left(\frac{x-m}{b}\right) + \frac{1}{2}$$

Mean

Undefined

Variance

Undefined

Skewness

Undefined

Reference: WolframMathWorld

2.1.2 Chi distribution

PDF

$$P(x) = \frac{2^{1-k/2} x^{k-1} \exp\left(-x^2/2\right)}{\Gamma(k/2)}$$

where k is the degrees of freedom.

CDF

$$D(x) = P(k/2, x^2/2)$$

where $P(a, b)$ is the regularized gamma function.

Mean

$$\mu = \sqrt{2}\frac{\Gamma((k+1)/2)}{\Gamma(k/2)}$$

Variance

$$\sigma^2 = k - \mu^2$$

Skewness

$$\frac{\mu}{\sigma^3}(1 - 2\sigma^2)$$

Reference: Wikipedia

2.1.3 Chi Squared distribution

A Chi Squared distribution with k degrees of freedom is the distribution of a sum of the squares of k independent standard normal random variables.

PDF

$$P(x) = \frac{x^{k/2-1}\exp(-x/2)}{2^{k/2}\Gamma(k/2)}$$

where k is the degrees of freedom; $\Gamma(a)$ is the gamma function.

CDF

$$D(x) = P(k/2, x/2)$$

where $P(a, b)$ is the regularized gamma function.

Mean

$$\mu = k$$

Variance

$$2k$$

Skewness

$$\sqrt{8/k}$$

Reference: Wikipedia

2.1.4 Erlang distribution

PDF

$$P(x) = \frac{\lambda^k x^{k-1}\exp(\lambda x)}{(k-1)!}$$

where k is the shape parameter; λ is the rate parameter.

CDF

$$D(x) = \frac{\gamma(k, \lambda x)}{(k-1)!}$$

where $\gamma(a, b)$ is the lower gamma function.

Mean

$$\frac{k}{\lambda}$$

Variance

$$\frac{k}{\lambda^2}$$

Skewness

$$\frac{2}{\sqrt{k}}$$

Reference: Wikipedia

2.1.5 Exponential distribution

PDF

$$P(x) = \begin{cases} \lambda \exp\left(-\lambda x\right) & x \geq 0, \\ 0 & x < 0. \end{cases}$$

CDF

$$D(x) = \begin{cases} 1 - \exp\left(-\lambda x\right) & x \geq 0, \\ 0 & x < 0. \end{cases}$$

Mean

$$\frac{1}{\lambda}$$

Variance

$$\frac{1}{\lambda^2}$$

Skewness

$$2$$

Reference: Wikipedia

2.1.6 Gamma distribution

PDF

$$P(x) = \frac{1}{\Gamma(\alpha)\theta^{\alpha}} x^{\alpha-1} \exp\left(-\frac{x}{\theta}\right)$$

where α is the shape parameter; θ is the scale parameter.

CDF

$$D(x) = \frac{1}{\Gamma(\alpha)} \gamma(\alpha, \frac{x}{\theta})$$

Mean

$$k\theta$$

Variance

$$k\theta^2$$

Skewness

$$\frac{2}{\sqrt(\alpha)}$$

Reference: WolframMathWorld

2.1.7 Irwin-Hall distribution

PDF

$$P(x) = \frac{1}{(n-1)!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k \binom{n}{k}(x-k)^{n-1}$$

where n is number of IDD of uniform distributions.

CDF

$$D(x) = \frac{1}{n!}\sum_{k=0}^{\lfloor x \rfloor}(-1)^k \binom{n}{k}(x-k)^{n}$$

Mean

$$\frac{n}{2}$$

Variance

$$\frac{n}{12}$$

Skewness

$$0$$

Reference: WolframMathWorld

2.1.8 Non-central Chi distribution

PDF

$$P(x) = \frac{\exp\left(-(x^2+\lambda^2)/2\right)x^k\lambda}{(\lambda x)^{k/2}} I_{k/2-1}(\lambda x)$$

where k is the degrees of freedom; λ is the distance parameter; $I_M(a)$ is a modified cylindrical Bessel function of the first kind.

CDF

$$D(x) = 1 - Q_{\frac{k}{2}}\left(\lambda, x\right)$$

where $Q_M(a, b)$ is Marcum Q-function.

Mean

To be implemented.

Variance

To be implemented.

Reference: Wikipedia

2.1.9 Non-central Chi Squared distribution

PDF

$$P(x) = \frac{1}{2}e^{-(x+\lambda)/2}\left(\frac{x}{\lambda}\right)^{k/4-1/2} I_{k/2-1}(\sqrt{\lambda x})$$

where k is the degrees of freedom; λ is the distance parameter;

CDF

$$D(x) = 1 - Q_{\frac{k}{2}}\left(\sqrt{\lambda}, \sqrt{x}\right)$$

where $Q_M(a, b)$ is Marcum Q-function.

Mean

$$k + \lambda$$

Variance

$$2(k + 2\lambda)$$

Skewness

$$\frac{2^{3/2}(k + 3\lambda)}{(k + 2\lambda)^{3/2}}$$

Reference: Wikipedia

2.1.10 Normal distribution

PDF

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

where μ is the expection of the distribution; σ is the standard deviation.

CDF

$$D(x) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right]$$

where $erf(a)$ is the error function.

Mean

$$\mu$$

Variance

$$\sigma^2$$

Skewness

$$0$$

Reference: Wikipedia

2.1.11 Rayleigh distribution

PDF

$$P(x) = \frac{x}{\sigma^2} e^{-x^2/\left(2\sigma^2\right)}$$

CDF

$$D(x) = 1 - e^{-x^2/\left(2\sigma^2\right)}$$

Mean

$$\sigma\sqrt{\frac{\pi}{2}}$$

Variance

$$\frac{4-\pi}{2}\sigma^2$$

Skewness

$$\frac{2\sqrt{\pi}(\pi-3)}{(4-\pi)^{3/2}}$$

Reference: Wikipedia

2.1.12 Standard Normal distribution

PDF

$$P(x) = \frac{1}{\sqrt{2\pi}} \exp{(-\frac{1}{2}x^2)}$$

where μ is the expection of the distribution; σ is the standard deviation.

CDF

$$D(x) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

where $\operatorname{erf}(a)$ is the error function.

Mean

$$0$$

Variance

$$1$$

Skewness

$$0$$

2.1.13 Uniform distribution (Continuous)

PDF

$$P(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$$

where a is the lower bound; b is the upper bound.

CDF

$$D(x) = \begin{cases} 0 & \text{for } x < a \\ \frac{x-a}{b-a} & \text{for } x \in [a, b] \\ 1 & \text{for } x > b \end{cases}$$

Mean

$$\frac{1}{2}(a + b)$$

Variance

$$\frac{1}{12}(b - a)^2$$

Skewness

$$0$$

Reference: Wikipedia

2.2 Mixture distribution

A **mixture distribution** is the probability distribution of a random variable that is derived from a collection of other random varibales. The probability density function (and the cumulative distribution function) can be expressed as the a convex combination of other distribution functions. The individual distributions that are combined to form the mixture distribution are called the **mixture components**. The weights associated with each component are called the **mixture weights**. (Wikipedia)

PDF

Given the mixture components' PDF $P_1(x), ..., P_n(x)$ and weights $w_1, ..., w_n$, the mixture's PDF is a convex combination:

$$P_{\text{mixture}}(x) = \sum_{i=1}^{n} w_i P_i(x)$$

CDF

Given the mixture components' CDF $D_1(x), ..., D_n(x)$ and weights $w_1, ..., w_n$, the mixture's CDF is a convex combination:

$$D_{\text{mixture}}(x) = \sum_{i=1}^{n} w_i D_i(x)$$

The moments of a mixture distribution are not as mathematically simple as PDF or CDF. To acquire the mathematically expressions for Mean, Variance, and Skewness, we need to clarify the three types of moments:

1. k^{th} non-central moment: $\mu^{(k)} = \mathbb{E}[x^k]$
2. k^{th} central moment: $\mu_c^{(k)} = \mathbb{E}[(x - \mu^{(1)})^k]$
3. k^{th} standardized moment: $\mu_s^{(k)} = \mathbb{E}[(\frac{x-\mu^{(1)}}{\sigma})^k]$

, where $^{(k)}$ denotes the k^{th} moment.

Mean

Mean is also known as the first non-central moment. The k^{th} non-central moment of a random variable can be rewritten in terms of integrals as:

$$\begin{aligned}
\mu^{(k)} &= \mathbb{E}[x^k] && (2.1)\\
&= \int_{-\infty}^{\infty} x^k f(x)dx && (2.2)\\
&= \int_{-\infty}^{\infty} x^k \sum_{i=1}^{n} w_i f_i(x)dx && (2.3)\\
&= \sum_{i=1}^{n} w_i \int_{-\infty}^{\infty} x^k f_i(x)dx && (2.4)\\
&= \sum_{i=1}^{n} w_i \mathbb{E}_{f_i}[x^k] && (2.5)\\
&= \sum_{i=1}^{n} w_i \mu_i^{(k)} && (2.6)
\end{aligned}$$

, where $\mu_i^{(k)}$ is the k^{th} non-central moment of distribution function f_i.

Given the mixture components' mean $\mu_1, ..., \mu_n$ and weights $w_1, ..., w_n$, it is easy to see the mixture's mean is just a convex combination:

$$\mu_{\text{mixture}}^{(1)} = \sum_{i=1}^{n} w_i \mu_i^{(1)}$$

Variance

Variance is also known as the second central moment. The k^{th} central moment follows a deriavation similar to the non-central moment:

$$\begin{aligned}
\sigma^2 := \mu_c^{(2)} &= \mathbb{E}[(x - \mu^{(1)})^2] && (2.7)\\
&= \mathbb{E}[x^2] - (\mathbb{E}[x])^2 && (2.8)\\
&= \mu^{(2)} - (\mu^{(1)})^2 && (2.9)\\
&= \sum_{i=1}^{n} w_i \mu_i^{(2)} - \left(\sum_{i=1}^{n} w_i \mu_i^{(1)}\right)^2 && (2.10)\\
&= \sum_{i=1}^{n} w_i \left(\sigma_i^2 + (\mu_i^{(1)})^2\right) - \left(\sum_{i=1}^{n} w_i \mu_i^{(1)}\right)^2 && (2.11)\\
&= \sum_{i=1}^{n} w_i \sigma_i^2 + \sum_{i=1}^{n} w_i (\mu_i^{(1)})^2 - \left(\sum_{i=1}^{n} w_i \mu_i^{(1)}\right)^2 && (2.12)
\end{aligned}$$

, where σ_i^2 is i^{th} component's variance, and $\mu_i^{(1)}$ is i^{th} component's mean.

With simplified the notations, we get a familiar expression for the mixture's variance.

$$\sigma_{\text{mixture}}^2 = \sum_{i=1}^{n} w_i(\sigma_i^2 + \mu_i^2) - \mu_{\text{mixture}}^2$$

Skewness

Skewness is also known as the third standardized moment.

$$\mu_s^{(3)} = \mathbb{E}[\left(\frac{x-\mu^{(1)}}{\sigma}\right)^3] \tag{2.13}$$

$$= \frac{\mathbb{E}[x^3] - 3\mu^{(1)}\mathbb{E}[x^2] + 3(\mu^{(1)})^2\mathbb{E}[x] - (\mu^{(1)})^3}{\sigma^3} \tag{2.14}$$

$$= \frac{\mathbb{E}[x^3] - 3\mu^{(1)}\sigma^2 - (\mu^{(1)})^3}{\sigma^3} \tag{2.15}$$

To rewrite it in terms of the skewness of each component $(\mu_s^{(3)})_i$, let's start by rewriting the third non-central moment:

$$\mathbb{E}[x^3] = \sum_{i=1}^{n} w_i \mu_i^{(3)} = \sum_{i=1}^{n} w_i \left(\sigma_i^3 (\mu_s^{(3)})_i + 3\mu_i^{(1)}\sigma_i^2 + (\mu_i^{(1)})^3\right)$$

Now the Skewness of the mixture can be rewritten to:

$$\mu_s^{(3)} = \frac{\sum_{i=1}^{n} w_i \left(\sigma_i^3 (\mu_s^{(3)})_i + 3\mu_i^{(1)}\sigma_i^2 + (\mu_i^{(1)})^3\right) - 3\mu^{(1)}\sigma^2 - (\mu^{(1)})^3}{\sigma^3}$$

where $\mu^{(1)}$ is the mean of the mixture, and σ^2 is the variance of the mixture.

2.3 Usage in C++

A good place to see example usages is the gtest files in the **tests** directory.

Because all distribution classes are derived *probDistr* class in *probDistr.h*, their interfaces are the same. That is, you can call the following methods in all probability distribution classes:

1. pdf
2. cdf
3. mean
4. stddev
5. variance
6. skewness
7. hash
8. print
9. isEqual_tol
10. isEqual_ulp
11. getID

In addition to the interface methods above, the Mixture distribution has some generic container methods. Please see $tests/unit_test/tst_disMixture.cpp$ for reference.

2.3.1 How to add a new probability distribution?

Since probability distribution classes inherit from *probDistr* class in *probDistr.h*, a new probability classes should implement:

1. the pure virtual functions in *probDistr*.
2. a hasher.
3. a cloner which duplicates an object that is raw or is managed by smart-pointer.
4. a serializer for human-friendly text to STDOUT.
5. two comparers with different type of thresholds – floating-point values and ULPs.
6. a unique ID that identifies the distribution.

```
class probDistr {
public:

    virtual double pdf(const double=0) const    = 0;
    virtual double cdf(const double=0) const    = 0;
```

```
    virtual double mean() const                  = 0;
    virtual double stddev() const                = 0;
    virtual double variance() const              = 0;
    virtual double skewness() const              = 0;

    virtual std::size_t hash() const noexcept {
        std::size_t seed = 0;
        combine_hash(seed, id);
        return seed;
    }

    virtual std::unique_ptr<probDistr> cloneUnique() const = 0;
    virtual probDistr* clone() const = 0;    // Return type can be Covariant.

    friend std::ostream& operator << (std::ostream&, const probDistr&);
    virtual void print(std::ostream&) const = 0;

    virtual bool isEqual_tol(const probDistr&, const double) const = 0;
    virtual bool isEqual_ulp(const probDistr&, const unsigned) const = 0;

    virtual dFuncID getID() const {return id;};
    const dFuncID id = dFuncID::BASE_DISTR;

    ...
}
```

In addition, it is recommanded to implement those to align the feature set with existing distributions:

1. getters for distribution parameters.
2. re-direct std::hash to call custome hasher.

For example, below is my implementation of Rician distribution.

```
class disRician : public probDistr {
private:

    double nu;      // distance
    double sigma;   // scale

public:
    template<class T, class U>
    requires std::is_arithmetic_v<T> && std::is_arithmetic_v<U>
    disRician(const T distance, const U scale) {
        nu = std::abs(distance);
        sigma = scale;
    }
    disRician() = delete;
    ~disRician() = default;
```

```cpp

    double pdf(const double x) const override {
        const double s2inv = 1/(sigma*sigma);
        const double x_e = exp(-(x*x+nu*nu)*0.5*s2inv) * std::cyl_bessel_i(0,x*nu*s2inv);
        return x * s2inv * x_e;
    }

    double cdf(const double x) const override {
        const double s_inv = 1/sigma;
        return 1 - marcumQ(1, nu*s_inv, x*s_inv);
    }

    double mean() const override {
        const double x = -0.5*nu*nu/(sigma*sigma);
        const double lague = exp(x/2) *
            ((1-x)*std::cyl_bessel_i(0,-0.5*x) - x*std::cyl_bessel_i(1,-0.5*x));
        return sigma * SQRT_PI_2 * lague;
    }

    double stddev() const override {
        return std::sqrt(variance());
    }

    double variance() const override {
        const double x = -0.5*nu*nu/(sigma*sigma);
        const double lague = exp(x/2) *
            ((1-x)*std::cyl_bessel_i(0,-0.5*x) - x*std::cyl_bessel_i(1,-0.5*x));
        return 2*sigma*sigma + nu*nu - M_PI_2*sigma*sigma*lague*lague;
    }

    double skewness() const override {
        throw std::runtime_error("Rician distribution's skewness is too complicated.");
        return 0;
    }

    inline std::size_t hash() const noexcept {
        std::size_t seed = 0;
        combine_hash(seed, char(id));
        combine_hash(seed, nu);
        combine_hash(seed, sigma);
        return seed;
    }

    std::unique_ptr<probDistr> cloneUnique() const override {
        return std::make_unique<disRician>(static_cast<disRician const&>(*this));
    };

    disRician* clone() const override {
        return new disRician(*this);
    }
```

```cpp

    void print(std::ostream& output) const override {
        output << "Rician distribution -- nu = " << nu << " sigma = " << sigma;
    }

    bool isEqual_tol(const probDistr& o, const double tol) const override {
        const disRician& oo = dynamic_cast<const disRician&>(o);
        bool r = true;
        r &= isEqual_fl_tol(nu, oo.nu, tol);
        r &= isEqual_fl_tol(sigma, oo.sigma, tol);
        return r;
    }

    bool isEqual_ulp(const probDistr& o, const unsigned ulp) const override {
        const disRician& oo = dynamic_cast<const disRician&>(o);
        bool r = true;
        r &= isEqual_fl_ulp(nu, oo.nu, ulp);
        r &= isEqual_fl_ulp(sigma, oo.sigma, ulp);
        return r;
    }

    auto p_distance() const noexcept {
        return nu;
    }

    auto p_scale() const noexcept {
        return sigma;
    }
};

template<>
class std::hash<statanaly::disRician> {
public:
    std::size_t operator() (const statanaly::disRician& d) const {
        return d.hash();
    }
};
```

Chapter 3

Sum of Random Variables of Probability Distribution

3.1 Sum $R = X + Y$

StatAnaly supports the sum of random variable in the format of $R = X + Y$, where X and Y are random variables of probability distributions. The supported combinations of the distributions are listed in the section.

3.1.1 Standard Uniform & Standard Uniform (continuous)

$$R = \sum_{i=1}^{2} X_i \qquad X_i \sim \text{StdUniform}, \quad R \sim \text{Irwin-Hall}(2) \tag{3.1}$$

3.1.2 Normal & Normal

$$R = \sum_{i=1}^{2} X_i \qquad X_i \sim \text{Normal}(\mu_i, \sigma_i), \quad R \sim \text{Normal}\left(\sum_{i=1}^{2} \mu_i, \sum_{i=1}^{2} \sigma_i^2\right) \tag{3.2}$$

3.1.3 Cauchy & Cauchy

$$R = \sum_{i=1}^{2} X_i \qquad X_i \sim \text{Cauchy}(m_i, b_i), \quad R \sim \text{Cauchy}\left(\sum_{i=1}^{2} m_i, \sum_{i=1}^{2} b_i\right) \tag{3.3}$$

3.1.4 Gamma & Gamma

$$R = \sum_{i=1}^{2} X_i \qquad X_i \sim \text{Gamma}(\alpha_i, \theta), \quad R \sim \text{Gamma}\left(\sum_{i=1}^{2} \alpha_i, \theta\right) \tag{3.4}$$

3.1.5 Exponential & Exponential

$$R = \sum_{i=1}^{2} X_i \qquad X_i \sim \text{Exponential}(\theta), \quad R \sim \text{Erlang}\left(2, \theta\right) \tag{3.5}$$

3.2 List of supported sum $R = X^2 + Y^2$

StatAnaly supports the sum of random variable in the format of $R = X^2 + Y^2$, where X and Y are random variables of probability distributions. The supported combinations of the distributions are listed in the section.

3.2.1 Normal & Normal

$$R = \sum_{i=1}^{2} X_i^2 \qquad X_i \sim \text{Normal}(0, 1), \quad R \sim \text{Chi-Squared}\,(2) \tag{3.6}$$

$$R = \sum_{i=1}^{2} X_i^2 \qquad X_i \sim \text{Normal}(\mu_i, 1), \quad R \sim \text{Non-central Chi-Squared}\left(2, \sum_{i=1}^{2} \mu_i^2\right) \tag{3.7}$$

3.3 List of supported sum $R = \sqrt{X^2 + Y^2}$

StatAnaly supports the sum of random variable in the format of $R = \sqrt{X^2 + Y^2}$, where X and Y are random variables of probability distributions. The supported combinations of the distributions are listed in the section.

3.3.1 Normal & Normal

$$R = \sqrt{\sum_{i=1}^{2} X_i^2} \qquad X_i \sim \text{Normal}(0, \sigma^2), \quad R \sim \text{Rayleigh}\,(\sigma) \tag{3.8}$$

$$R = \sqrt{\sum_{i=1}^{2} X_i^2} \qquad X_i \sim \text{Normal}(\mu_i, \sigma^2), \quad R \sim \text{Rician}\left(\sqrt{\sum_{i=1}^{2} \mu_i^2}, \sigma\right) \tag{3.9}$$

3.4 List of supported sum $R = \sum_i X_i$

StatAnaly supports the sum of random variable in the format of $R = \sum_i X_i$, where X is random variable of a probability distribution. The supported combinations of the distributions are listed in the section.

3.4.1 Array of Standard Uniform

$$R = \sum_{i=1}^{n} X_i \qquad X_i \sim \text{StdUniform}, \quad R \sim \text{Irwin-Hall}(n) \tag{3.10}$$

3.4.2 Array of Normal

$$R = \sum_{i=1}^{n} X_i \qquad X_i \sim \text{Normal}(\mu_i, \sigma_i), \quad R \sim \text{Normal}\left(\sum_{i=1}^{n} \mu_i, \sum_{i=1}^{n} \sigma_i^2\right) \tag{3.11}$$

3.4.3 Array of Cauchy

$$R = \sum_{i=1}^{n} X_i \qquad X_i \sim \text{Cauchy}(m_i, b_i), \quad R \sim \text{Cauchy}\left(\sum_{i=1}^{n} m_i, \sum_{i=1}^{n} b_i\right) \tag{3.12}$$

3.4.4 Array of Gamma

$$R = \sum_{i=1}^{n} X_i \qquad X_i \sim \text{Gamma}(\alpha_i, \theta), \quad R \sim \text{Gamma}\left(\sum_{i=1}^{n} \alpha_i, \theta\right) \tag{3.13}$$

3.4.5 Array of Exponential

$$R = \sum_{i=1}^{n} X_i \qquad X_i \sim \text{Exponential}(\theta), \quad R \sim \text{Erlang}\left(n, \theta\right) \tag{3.14}$$

3.5 List of supported sum $R = \sum_i X_i^2$

StatAnaly supports the sum of random variable in the format of $R = \sum_i X_i^2$, where X is random variable of a probability distribution. The supported combinations of the distributions are listed in the section.

3.5.1 Array of Normal

$$R = \sum_{i=1}^{n} X_i^2 \qquad X_i \sim \text{Normal}(0, 1), \quad R \sim \text{Chi-Squared}\left(n\right) \tag{3.15}$$

$$R = \sum_{i=1}^{n} X_i^2 \qquad X_i \sim \text{Normal}(\mu_i, 1), \quad R \sim \text{Non-central Chi-Squared}\left(2, \sum_{i=1}^{n} \mu_i^2\right) \tag{3.16}$$

3.6 List of supported sum $R = \sqrt{\sum_i X_i^2}$

StatAnaly supports the sum of random variable in the format of $R = \sqrt{\sum_i X_i^2}$, where X is random variable of a probability distribution. The supported combinations of the distributions are listed in the section.

3.6.1 Array of Normal

$$R = \sqrt{\sum_{i=1}^{n} X_i^2} \qquad X_i \sim \text{Normal}(0, 1), \quad R \sim \text{Chi}\left(n\right) \tag{3.17}$$

$$R = \sqrt{\sum_{i=1}^{n} X_i^2} \qquad X_i \sim \text{Normal}(\mu_i, 1), \quad R \sim \text{Non-central Chi}\left(2, \sqrt{\sum_{i=1}^{n} \mu_i^2}\right) \tag{3.18}$$

3.7 Usage in C++

A good place to see example usages is the gtest files *tests/tst_dConvolution.cpp* and *tests/tst_dConvolution_squares.cpp*.

There are three global static variables representing different sums:

1. *cnvl* represents the simple sum, ie $R = X + Y$.
2. *cnvlSq* represents the sum of the squares, ie $R = X^2 + Y^2$.
3. *cnvlSSqrt* represents the sum of the squares and then take square root, ie $R = \sqrt{X^2 + Y^2}$.

3.7.1 Simple Sum

Example 1

The sum of two RVs of standard uniform distributions is a RV of Irwin-Hall distribution.

```
disStdUniform su1{}, su2{};
probDistr* rsu = cnvl.go(su1, su2);
disIrwinHall* rih = dynamic_cast<disIrwinHall*>(rsu);
```

Example 2

The sum of an array of RVs of standard uniform distributions is a RV of Irwin-Hall distribution.

```
disStdUniform su1{}, su2{}, su3{}, su4{};
probDistr* rsu = convolve<disStdUniform>({su1, su2, su3, su4});
disIrwinHall* rih = dynamic_cast<disIrwinHall*>(rsu);
```

3.7.2 Sum of the Squares

Example

The sum of two squares of RVs of normal distributions is a RV of Chi Square distribution.

```
disNormal a(0,1);
disNormal b(0,1);
disChiSq s = *static_cast<disChiSq*>(cnvlSq.go(a,b));
```

3.7.3 Sum of the Squares, and Then Take Square Root

Example

The sum of two squares of RVs of normal distribution (with same variance) is a RV of Rician distribution.

```
disNormal a(2,9);
disNormal b(3,9);
disRician s = *static_cast<disRician*>(cnvlSSqrt.go(a,b));
```

3.8 How to add a new sum of random variables?

If you do not find your distributions in currenctly supported list, you can easily add a new sum for your random variables.

... is implemented in the double dispatcher pattern. (See *Modern C++ Design by Andrei Alexandrescu* for pattern detail.)

TBW

1. Register the pair in *dConvolution.cpp*.
2. Write a function prototype in *dConvolution.h*.
3. Implement the sum in *dConvolution.cpp*.

```
/* dConvolution.cpp */

auto ConvolutionSqDoubleDispatcherInitialization = [](){
    cnvlSq.add<disNormal,disNormal,convolveSq>();
    cnvlSq.add<disApple,disBananna,convolveSq>();  // new distribution pair.
    return true;
}();

probDistr* convolve(disApple& l, disBananna& r) {
    // Implement the sum
    // Blah, blah, blah
    probDistr* res = xxxxx;
    return res;
};
```

```
/* dConvolution.h */

probDistr* convolve(disCauchy& lhs, disCauchy& rhs);
```