Special Functions

Bessel Functions

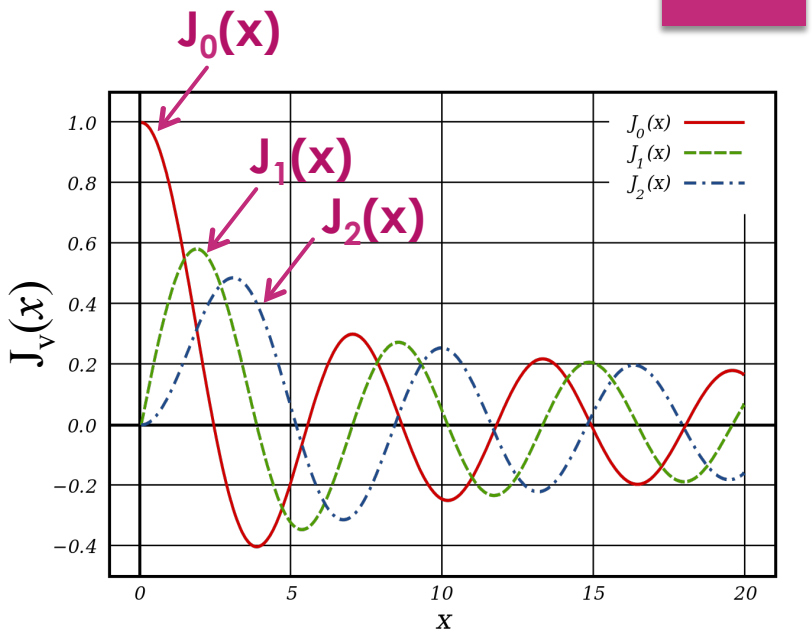
- Solutions to the Bessel ordinary differential equation
  - Two independent solutions: $J_v(x)$, $Y_v(x)$
  - Special case: v is an integer number → $J_n(x)$, $Y_n(x)$
- **Bessel differential equation (ODE)**

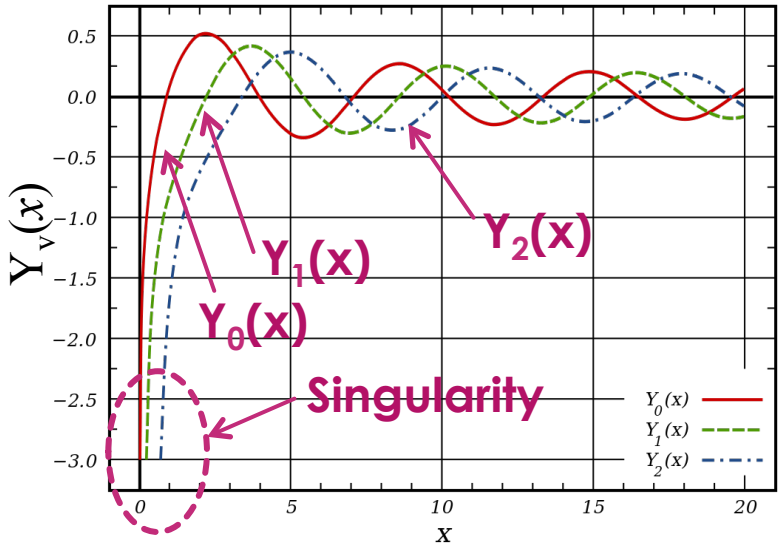
$$x^2\,y'' + x\,y' + (x^2 - \nu^2)y = 0$$

parameter

- Singular point: x = 0
  - $J_v(x)$ has no singularity at x = 0
  - $Y_v(x)$ has singularity at x = 0
- How find the numerical value?
  - By solving the differential equation
    - *Need initial conditions*
  - Also provides the derivative of $J_v(x)$ and $Y_v(x)$
  - Use asymptotic expressions when possible

→ based on elementary functions





Bessel Functions

- Solutions to the Bessel ordinary differential equation
- **Bessel differential equation (ODE)**

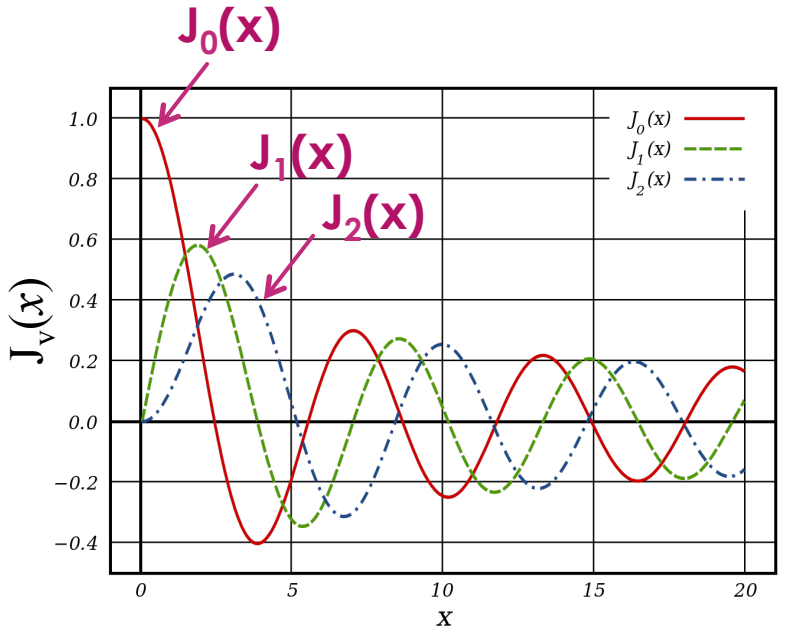
$$x^2\,y'' + x\,y' + (x^2 - n^2)y = 0$$

Integer

- $J_n(x)$ has no singularity at x = 0
- Initial condition at x = 0:
  - $J_0(0) = 1$, $J_0'(0) = 0$
  - For n > 0: $J_n(0) = 0$

  } **$J_n(0) = \delta[n]$** discrete delta function

  - For n > 0: $J_n'(0) = (J_{n-1}(0) - J_{n+1}(0))/2$
- **For n > 0: $J_{-n}(x) = (-1)^n J_n(x)$**
- **For x < 0: $J_n(-x) = (-1)^n J_n(x)$**
- $J_0(x)$, $J_2(x)$, $J_4(x)$, … → Even functions
- $J_1(x)$, $J_3(x)$, $J_5(x)$, … → Odd functions



Integer

Numerical evaluation

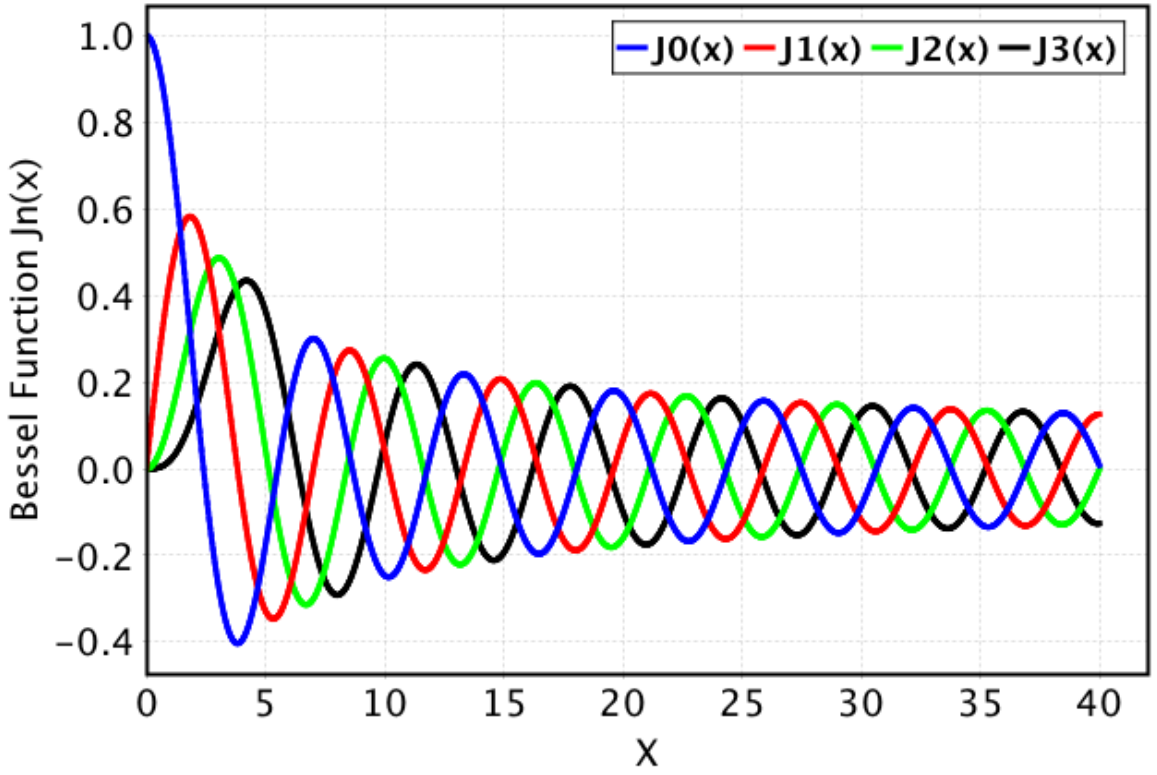
$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(n\theta - x\sin\theta)\,d\theta = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - n\theta)\,d\theta$$

$$Y_n(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin\theta - n\theta)\,d\theta - \frac{1}{\pi}\int_0^{\infty} (e^{nt} + (-1)^n e^{-nt})\,e^{-x\sinh t}\,dt$$

Bessel Function of First Kind: $J_n(x)$

- Solutions to the Bessel ordinary differential equation
- **Java implementation of $J_n(x)$ → integer index**

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(n\theta - x\sin\theta)\,d\theta = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - n\theta)\,d\theta$$



Romberg on Simpson for more accuracy

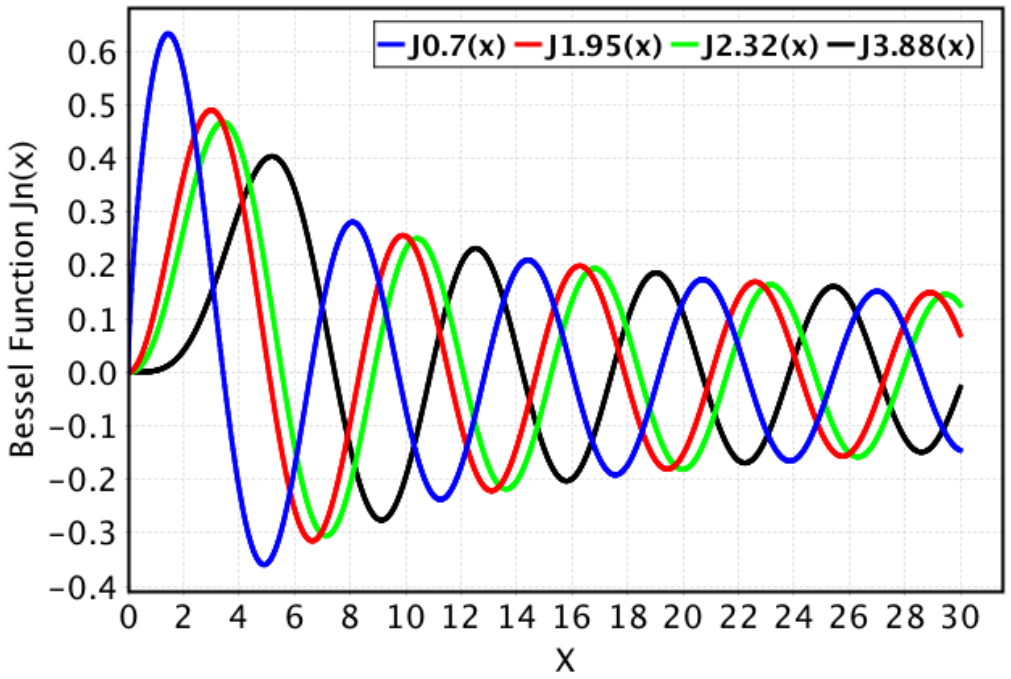
```
public double jn(int n, double x) {
      funcJn = t -> 1.0/PI * cos(x*sin(t)-n*t) ;
      integralJn.setIntegralFunction(funcJn);
      return integralJn.rombergOnSimpson(0.0, PI, order) ;
}
```

Default order = 300

```
BesselFunction bessel = new BesselFunction() ;

double[] x = MathUtils.linspace(1e-5, 40.0, 1000) ;
double[] f0 = ArrayFunc.apply(t -> bessel.jn(0, t), x) ;
double[] f1 = ArrayFunc.apply(t -> bessel.jn(1, t), x) ;
double[] f2 = ArrayFunc.apply(t -> bessel.jn(2, t), x) ;
double[] f3 = ArrayFunc.apply(t -> bessel.jn(3, t), x) ;

MatlabChart fig = new MatlabChart() ;
fig.plot(x, f0, "b", 2f, "J0(x)");
fig.plot(x, f1, "r", 2f, "J1(x)");
fig.plot(x, f2, "g", 2f, "J2(x)");
fig.plot(x, f3, "k", 2f, "J3(x)");
fig.renderPlot();
fig.xlabel("X");
fig.ylabel("Bessel Function Jn(x)");
fig.legendON();
fig.show(true);
```

Bessel Function of First Kind: $J_\alpha(x)$

- Solutions to the Bessel ordinary differential equation
- **Java implementation of $J_\alpha(x)$** → non-integer index
- An infinite integral (improper integral) is added
  - $\sin(n\pi) = 0$ for integer n



only for x>0

$$J_\alpha(x) = \frac{1}{\pi}\int_0^\pi \cos(x\sin\theta - \alpha\theta)\,d\theta - \frac{\sin\alpha\pi}{\pi}\int_0^\infty e^{-x\sinh t - \alpha t}\,dt$$

Romberg on Simpson for more accuracy

```
public double jn(double alpha, double x) {
      if(abs(alpha-(int)alpha)<1e-5)
            return jn((int)alpha, x) ;
      funcJn = t -> 1.0/PI * cos(x*sin(t)-alpha*t) ;
      funcJnInf = t -> -1.0/PI * sin(alpha*PI) * exp(-x*sinh(t)-alpha*t) ;
      integralJn.setIntegralFunction(funcJn);
      double term1 = integralJn.rombergOnSimpson(0.0, PI, order)  ;
      integralJn.setIntegralFunction(funcJnInf);
      double term2 = integralJn.rombergOnSimpson(0.0, 1e2, order)  ;
      return term1 + term2 ;
}
```

Some large number

Bessel Function of First Kind: $J_n(x)$

- Solutions to the Bessel ordinary differential equation
- **Accuracy comparison with implementation in PYTHON (scipy.special package)**

Jupyter Notebook

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams.update({"font.size":15})
import scipy.special as sp
```

Import alias for "scipy.special"

Data from java plots

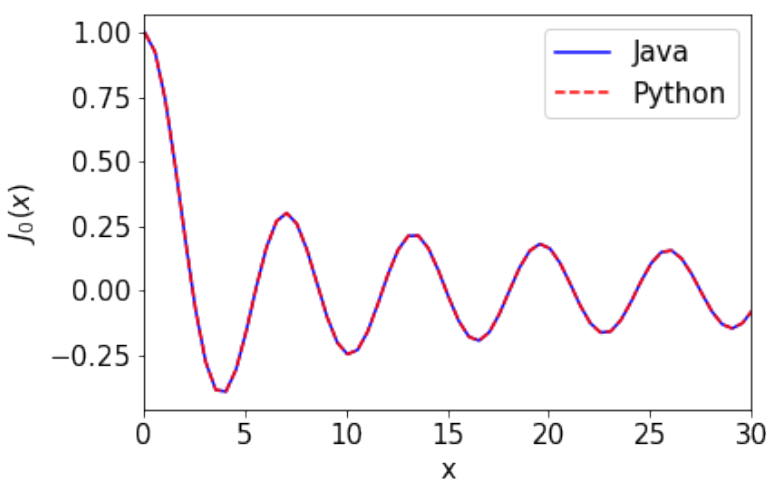
```
x_0 = [0.05, 0.5504504504504505, 1.050900900900901, 1.5513513513513513, 2.0518018018018016, 2.552252252252252, 3.052702
y_0 = [0.9993747075398763, 0.9256734450658163, 0.7423847516377767, 0.4830007354418437, 0.19411122798919678, -0.07401366
x_1 = [0.05, 0.5504504504504505, 1.050900900900901, 1.5513513513513513, 2.0518018018018016, 2.552252252252252, 3.052702
y_1 = [0.024992273499568302, 0.2649308939774831, 0.45617425441741943, 0.5645853281021118, 0.5728489955266317, 0.4837465
x_2 = [0.05, 0.5504504504504505, 1.050900900900901, 1.5513513513513513, 2.0518018018018016, 2.552252252252252, 3.052702
y_2 = [3.1276472145691514E-4, 0.0369273896018664, 0.12577460209528604, 0.24486187100410461, 0.36427611112594604, 0.4530
x_3 = [0.05, 0.5504504504504505, 1.050900900900901, 1.5513513513513513, 2.0518018018018016, 2.552252252252252, 3.052702
y_3 = [2.814023749427482E-6, 0.0034097107127308846, 0.022555979589621227, 0.06676456828912099, 0.13730852802594504, 0.2
```

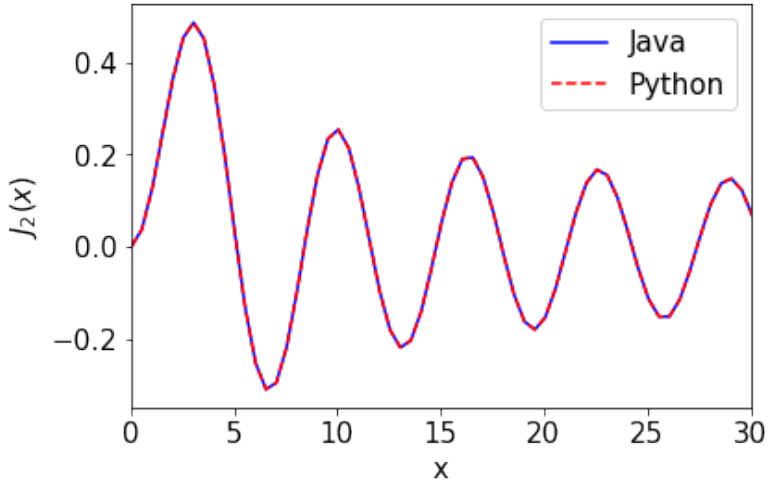
```
plt.plot(x_0, y_0, color='b', label='Java')
plt.plot(x_0, sp.jn(0, x_0), color='r', linestyle='--', label='Python')
plt.xlabel('x')
plt.ylabel('$J_0(x)$')
plt.legend()
plt.xlim([0, 30])
```

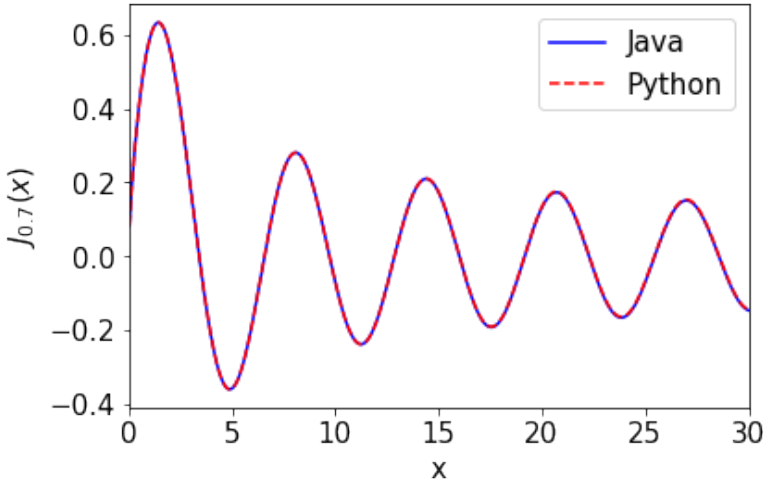
$J_n(n, x)$

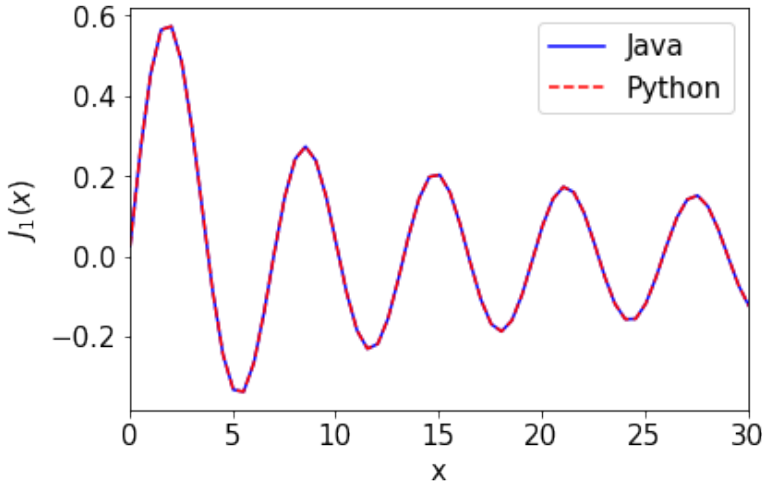
Bessel Function of First Kind: $J_n(x)$

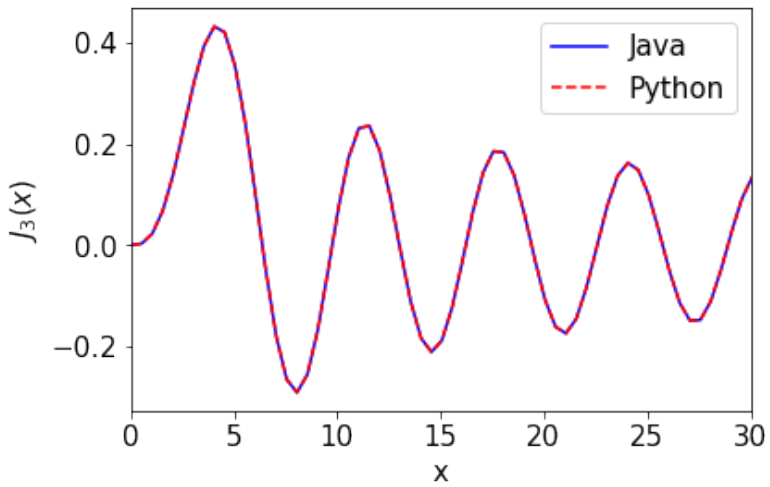
- Solutions to the Bessel ordinary differential equation
- **Comparison with implementation in PYTHON (scipy.special package)**

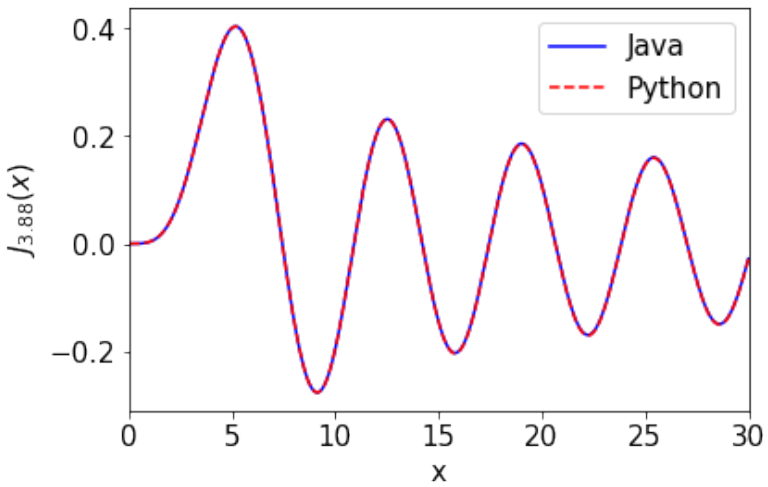












Bessel Function of First Kind: $J_n(x)$

- Expansion to complex numbers (x is complex)
- For integer index, this equation is valid for **all** complex numbers x:

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(n\theta - x\sin\theta)\,d\theta = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - n\theta)\,d\theta$$

- For non-integer index, this equation is valid for **Re(x)>0**:

Must convergence

$$J_\alpha(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - \alpha\theta)\,d\theta - \frac{\sin\alpha\pi}{\pi}\int_0^{\infty} e^{-x\sinh t - \alpha t}\,dt$$

- Need to use complex **cos(z)** function
  - z = a + j b
- Calculate the integrals twice
  - Real and Imaginary parts
- Use ComplexMath class

Real part

```
public Complex jn(int n, Complex x) {
      funcJn = t -> 1.0/PI * ComplexMath.cos(x*sin(t)-n*t).re() ;
      integralJn.setIntegralFunction(funcJn) ;
      double realPart = integralJn.rombergOnSimpson(0.0, PI, order) ;
      funcJn = t -> 1.0/PI * ComplexMath.cos(x*sin(t)-n*t).im() ;
      integralJn.setIntegralFunction(funcJn) ;
      double imagPart = integralJn.rombergOnSimpson(0.0, PI, order) ;
      return realPart + j*imagPart ;
}
```

Complex number

Imaginary part

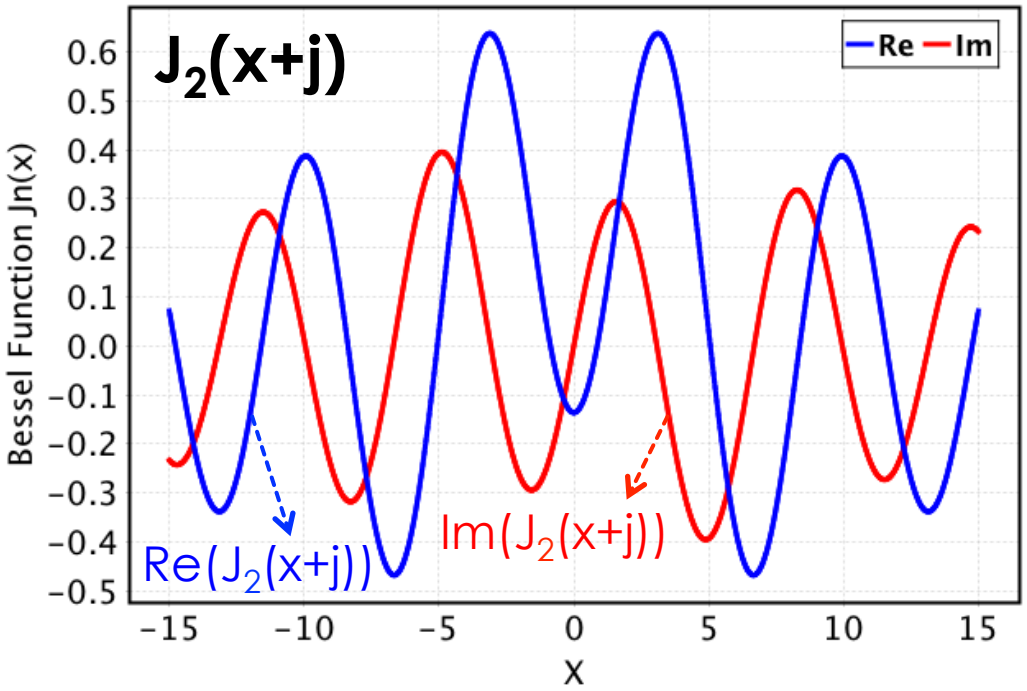
$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - n\theta)\,d\theta$$

Bessel Function of First Kind: $J_n(x)$

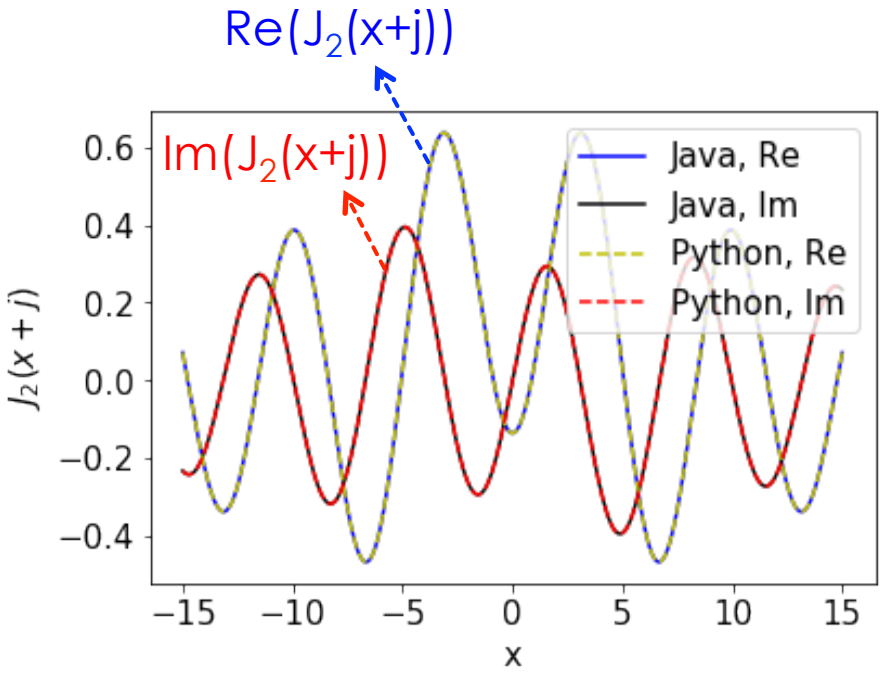
- Expansion to complex numbers

$$J_n(x) = \frac{1}{\pi}\int_0^{\pi} \cos(x\sin\theta - n\theta)\, d\theta$$

- **Comparison with implementation in PYTHON (scipy.special package)**
- x = linspace(-15, 15, 1000) → z = x + j → $J_2(z)$ = **J_2(x+j)**



Compare with scipy



Bessel Function of First Kind: $J_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- **Evaluation using power series for Small Arguments $|x| < 30$**
  - $\Gamma(x)$ is the gamma function → for integer ν, $\Gamma(1+\nu) = \nu!$

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k!\Gamma(\nu+k+1)}$$

$$= \underbrace{\frac{1}{\Gamma(1+\nu)}\left(\frac{x}{2}\right)^\nu}_{\text{Leading term}} \underbrace{\left\{1 - \frac{(x/2)^2}{1(1+\nu)}\left(1 - \frac{(x/2)^2}{2(2+\nu)}\left(1 - \frac{(x/2)^2}{3(3+\nu)}(1-\cdots)\right)\right)\right\}}_{\text{Polynomial → remember Horner's method?}}$$

  - When x → 0, the asymptotic behavior of $\mathbf{J_\nu(x)}$ is:

$$J_\nu(x) \approx \frac{1}{\Gamma(\nu+1)}\left(\frac{x}{2}\right)^\nu$$ → Small-argument expansion

  - Power series also works for complex numbers
  - Define a sequence to calculate the sum
  - Power series exhibits fast convergence (100 terms are enough)

Bessel Function of First Kind: $J_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- **Evaluation using power series for Small Arguments $|x| < 30$**
  - $\Gamma(x)$ is the gamma function → for integer ν, $\Gamma(1+\nu) = \nu!$

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k (x/2)^{\nu+2k}}{k!\Gamma(\nu+k+1)}$$

$$= \underbrace{\frac{1}{\Gamma(1+\nu)}\left(\frac{x}{2}\right)^\nu}_{\text{Leading term}} \underbrace{\left\{1 - \frac{(x/2)^2}{1(1+\nu)}\left(1 - \frac{(x/2)^2}{2(2+\nu)}\left(1 - \frac{(x/2)^2}{3(3+\nu)}(1-\cdots)\right)\right)\right\}}_{J_n \text{ sequence}} \longrightarrow \textbf{Expansion about x = 0}$$

clever calculation!!!

- Good convergence
- Faster performance than integration (**much faster**)
- Works for real and complex numbers

Power series is the most efficient way of evaluating Bessel functions

```
public double jv(double v, double x) {
      leadTerm = Math.pow(x/2.0, v)/gammaFunc.gamma(v+1.0) ;
      val = (0.5*x)*(0.5*x) ;
      jnSeq = n -> {
            double result = 1.0 ;
            for(int i=(int)n; i>0; i--) {
                  result = 1.0 - val/(i*(i+v))*result ;
            }
            return result ;
      } ;
      return jnSeq.evaluate(100)*leadTerm ;
}
```

Bessel Function of First Kind: $J_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- **Evaluation using power series for Large Arguments |x| > 2**
- Change of variable: t = 1/x → as x grows, t goes to zero
- Amplitude and phase representation
  - It is always possible to write $J_\nu(x)$ and $Y_\nu(x)$ in terms of amplitude and phases

$$J_\nu(x) = A_\nu(x)\cos(\phi_\nu(x))$$
$$Y_\nu(x) = A_\nu(x)\sin(\phi_\nu(x))$$

Amplitude varies slowly

Rapid oscillations are due to the phase

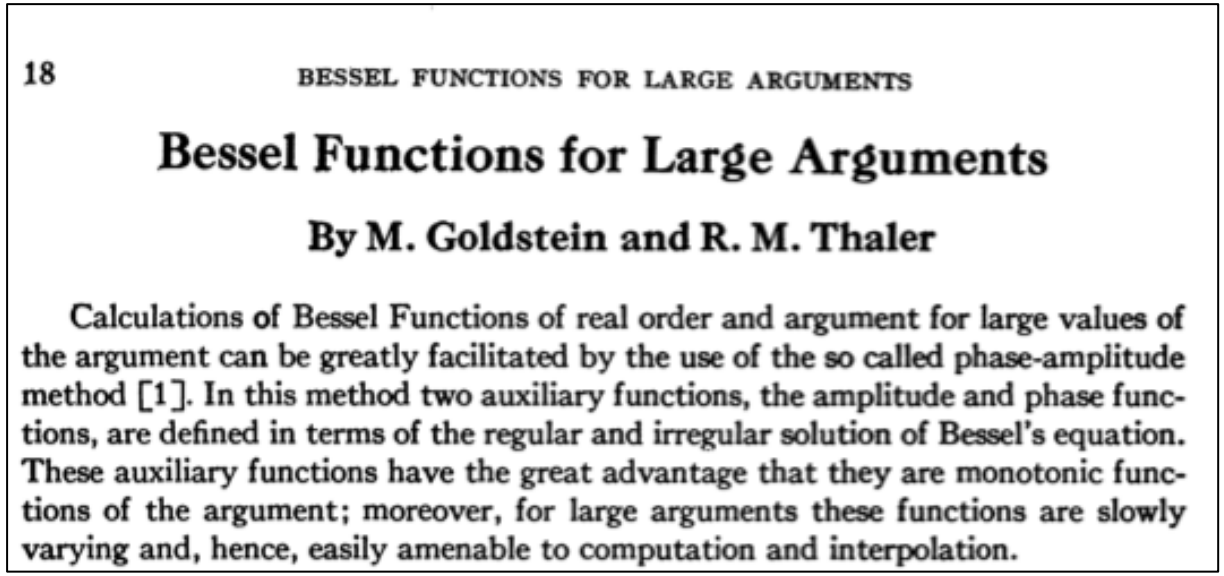
- Define in terms of "t" instead of x:

$$J_\nu(x) = \left[\frac{B_\nu(t)}{t}\right]\sqrt{\frac{2t}{\pi}}\cos(\phi_\nu(t))$$

$$Y_\nu(x) = \left[\frac{B_\nu(t)}{t}\right]\sqrt{\frac{2t}{\pi}}\sin(\phi_\nu(t))$$

- Now find a Taylor series for $B_\nu(t)/t$ and $\phi_\nu(t)$
  - In terms of "t" because t → 0

Paper published in 1957



18 BESSEL FUNCTIONS FOR LARGE ARGUMENTS

Bessel Functions for Large Arguments

By M. Goldstein and R. M. Thaler

Calculations of Bessel Functions of real order and argument for large values of the argument can be greatly facilitated by the use of the so called phase-amplitude method [1]. In this method two auxiliary functions, the amplitude and phase functions, are defined in terms of the regular and irregular solution of Bessel's equation. These auxiliary functions have the great advantage that they are monotonic functions of the argument; moreover, for large arguments these functions are slowly varying and, hence, easily amenable to computation and interpolation.

Bessel Function of First Kind: $J_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- **Evaluation using power series for Large Arguments x > 2**
- General formula with $t = 1/x > 0$:

Notice what happens when ν is an integer number

Polynomial for amplitude

Polynomial for phase

$$J_\nu(x) = \mathrm{Re}\left\{ \sqrt{\frac{2t}{\pi}}\, P_A(t)\, \exp(j\frac{P_\phi(t)}{t})\, \exp(-j\nu\frac{\pi}{2})\, \exp(-j\frac{\pi}{4}) \right\}$$

Real Part

$$A_\nu(x)\exp(j\phi_\nu(x))$$

$$J_\nu(x) = \sqrt{\frac{2t}{\pi}} P_A(t) \cos\left(\frac{P_\phi(t)}{t} - (\nu+\frac{1}{2})\frac{\pi}{2}\right)$$

Re(x)>0, Im(x)>0
Re(x)>0, Im(x)<0
Re(x)<0, Im(x)>0
Re(x)<0, Im(x)<0

Four cases for complex numbers

$$A_\nu(x) = \sqrt{\frac{2t}{\pi}} P_A(t) \quad \text{and} \quad \phi_\nu(x) = \frac{P_\phi(t)}{t} - \left(\nu+\frac{1}{2}\right)\frac{\pi}{2}$$

Bessel Function of First Kind: $J_n(x)$

- Solutions to the Bessel ordinary differential equation
- **Evaluation using power series for Large Arguments |x| > 2**
- General formula with t = 1/x:

$$A_\nu(x) = \sqrt{\frac{2t}{\pi}} P_A(t) \qquad \phi_\nu(x) = \frac{P_\phi(t)}{t} - \left(\nu + \frac{1}{2}\right)\frac{\pi}{2}$$

- Polynomials in terms of "t = 1/x":

$$P_A(t) = 1 + a_2 t^2 + a_4 t^4 + a_6 t^6 + a_8 t^8 + a_{10} t^{10} + \cdots$$

$$P_\phi(t) = 1 + b_2 t^2 + b_4 t^4 + b_6 t^6 + b_8 t^8 + b_{10} t^{10} + \cdots$$

- Coefficients:

$$p = \nu^2 - \frac{1}{4}$$

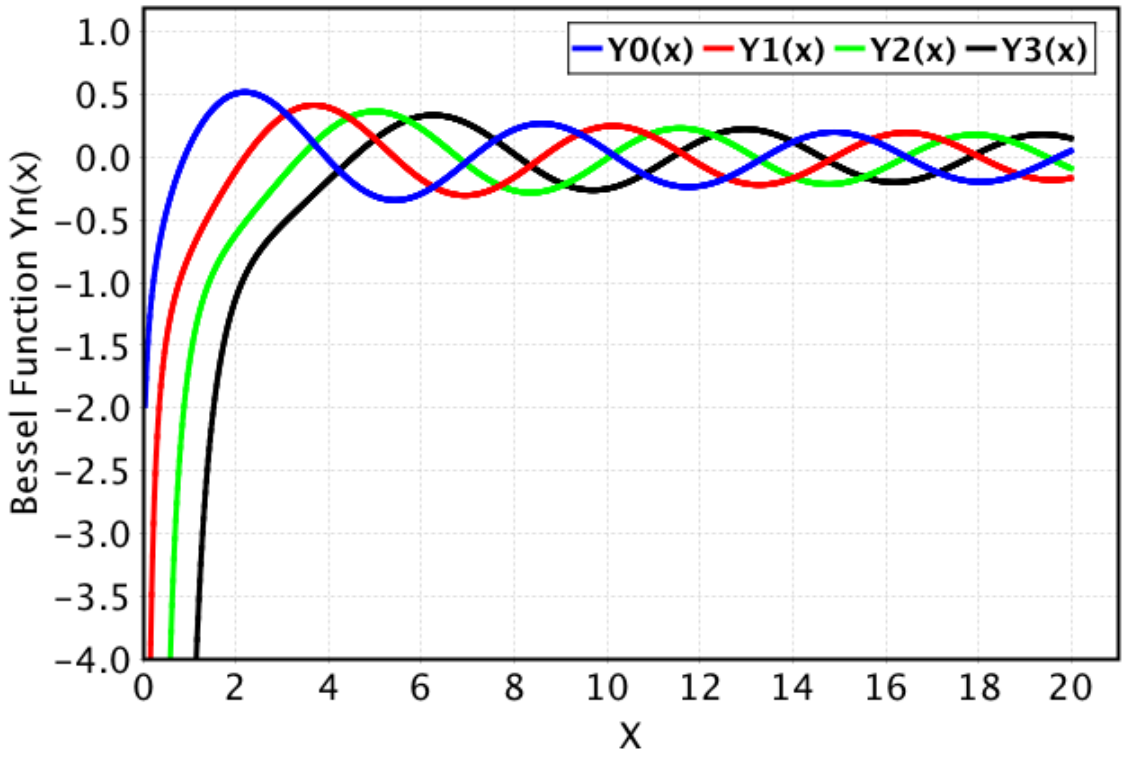
$$a_2 = \frac{1}{4}p \quad a_4 = \frac{5}{32}p^2 - \frac{3}{8}p \quad a_6 = \frac{15}{128}p^3 - \frac{37}{32}p^2 + \frac{15}{8}p \quad a_8 = \frac{195}{2048}p^4 - \frac{611}{256}p^3 + \frac{1821}{128}p^2 - \frac{315}{16}p$$

$$a_{10} = \frac{663}{8192}p^5 - \frac{4199}{1024}p^4 + \frac{29811}{512}p^3 - \frac{2223}{8}p^2 + \frac{2835}{8}p \qquad b_2 = \frac{1}{2}p \quad b_4 = \frac{1}{24}p^2 - \frac{1}{4}p \quad b_6 = \frac{1}{80}p^3 - \frac{7}{20}p^2 + \frac{3}{4}p$$

$$b_8 = \frac{1}{1792}p^4 - \frac{95}{224}p^3 + \frac{807}{224}p^2 - \frac{315}{56}p \qquad b_{10} = \frac{7}{2304}p^5 - \frac{35}{72}p^4 + \frac{1975}{192}p^3 - 58p^2 + \frac{315}{4}p$$

Bessel Function of Second Kind: $Y_n(x)$

- Solutions to the Bessel ordinary differential equation
- $Y_n(x)$ has other names: Neumann function, Weber function
- Java implementation of $Y_n(x)$:
  - Approach 1: calculate **by numerical integration**



inefficient approach

Re(x)>0

$$Y_n(x) = \frac{1}{\pi}\int_0^{\pi} \sin(x\sin\theta - n\theta)\,d\theta - \frac{1}{\pi}\int_0^{\infty} \left(e^{nt} + (-1)^n e^{-nt}\right) e^{-x\sinh t}\,dt$$

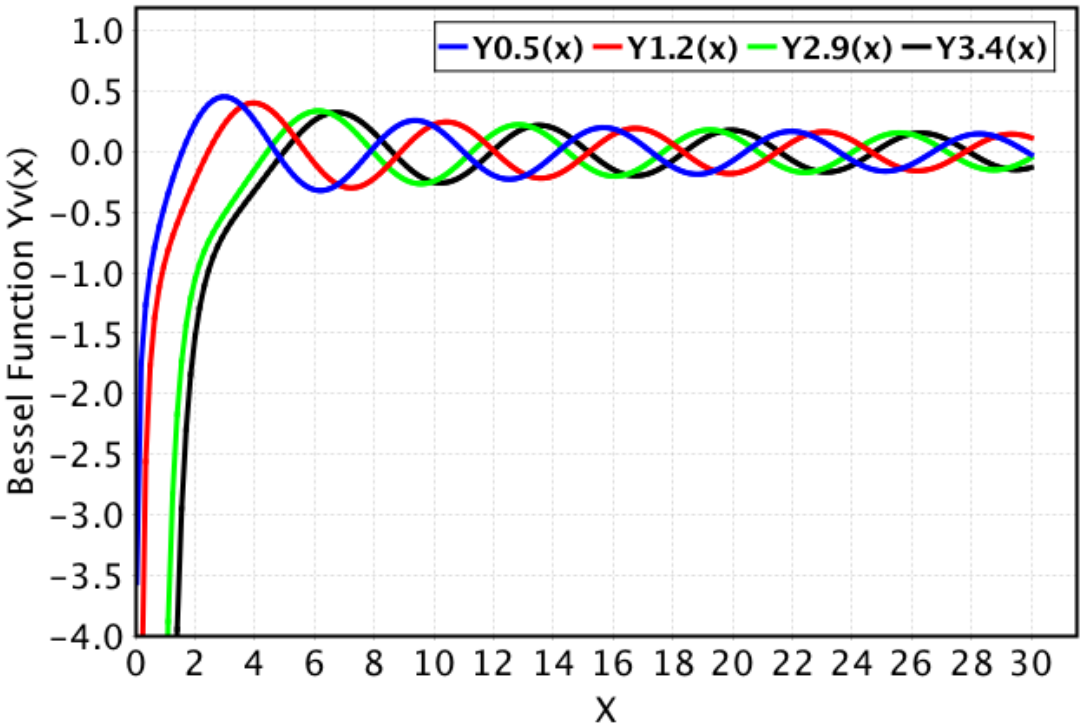
```
public double yn(int n, double x) {
      factor =  n%2==0 ? 1.0 : -1.0 ;
      funcYn = t -> 1.0/PI * sin(x*sin(t)-n*t) ;
      funcYnInf = t -> -1.0/PI * (exp(n*t)+exp(-n*t)*factor) * exp(-
x*sinh(t)) ;
      integralYn.setIntegralFunction(funcYn);
      double term1 =  integralYn.simpson(0.0, PI, order) ;
      integralYn.setIntegralFunction(funcYnInf);
      double term2 = integralYn.simpson(0.0, 10.0, order) ;
      return term1 + term2 ;
}
```

Simpson integration has good accuracy

Default order = 300

Bessel Function of Second Kind: $Y_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- $Y_n(x)$ has other names: Neumann function, Weber function
- Java implementation of $Y_\nu(x)$:
  - Calculate **by relation to $J_\nu(x)$**



Non-integer

$$Y_\nu(x) = \frac{J_\nu(x)\cos\nu\pi - J_{-\nu}(x)}{\sin\nu\pi} \quad \text{for} \quad \nu \neq 0, 1, 2, 3, \cdots$$

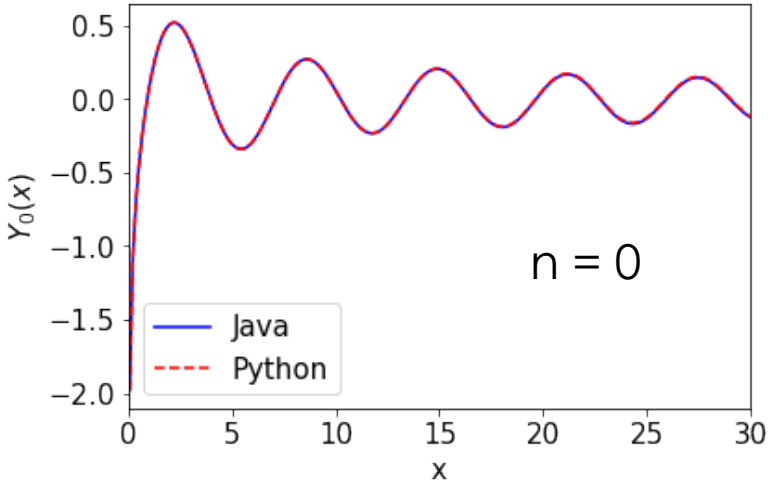
```
public double yv(double v, double x) {
      if(abs(v-(int)v)>=1e-2)
            return (jv(v, x)*cos(PI*v)-jv(-v, x))/sin(PI*v) ;
      else
            return yn((int)v, x) ;
}
```

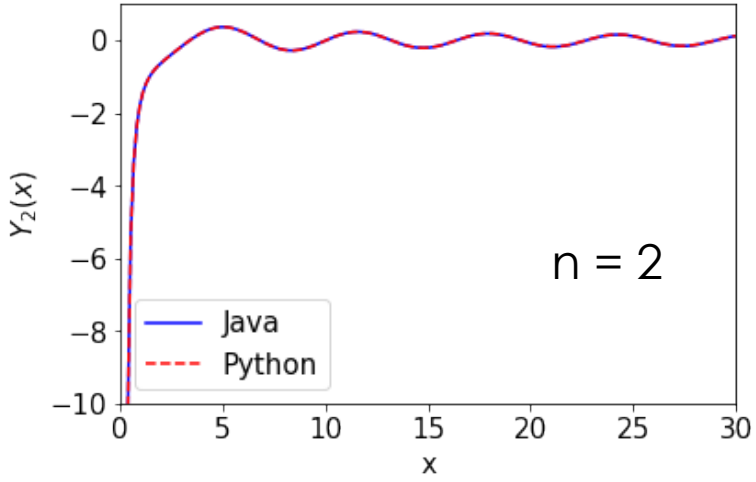
- For integer v=n, **take the limit as v → n**:
  - v = n – epsilon
  - Set epsilon to 0.01 for example

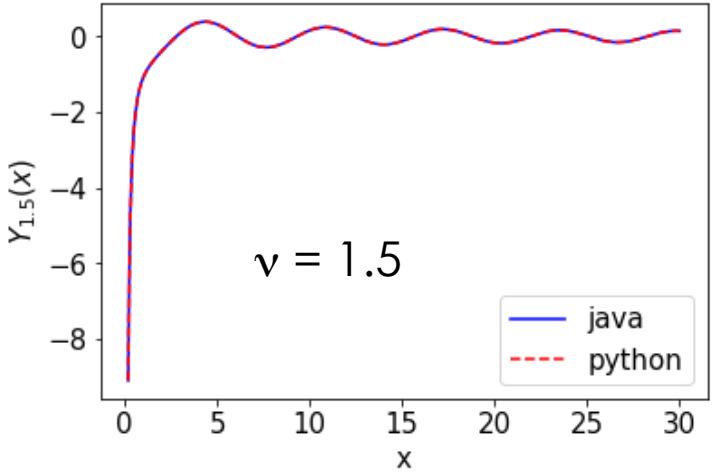
$$Y_n(x) = \lim_{\nu\to n} \frac{J_\nu(x)\cos\nu\pi - J_{-\nu}(x)}{\sin\nu\pi}$$

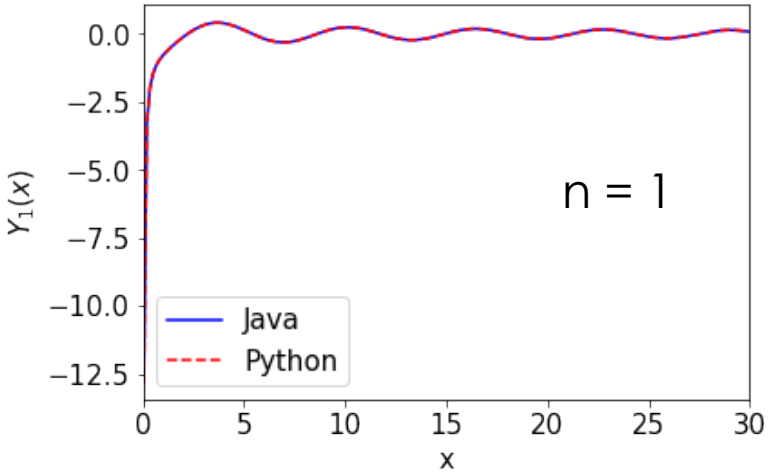
Bessel Function of Second Kind: $Y_\nu(x)$

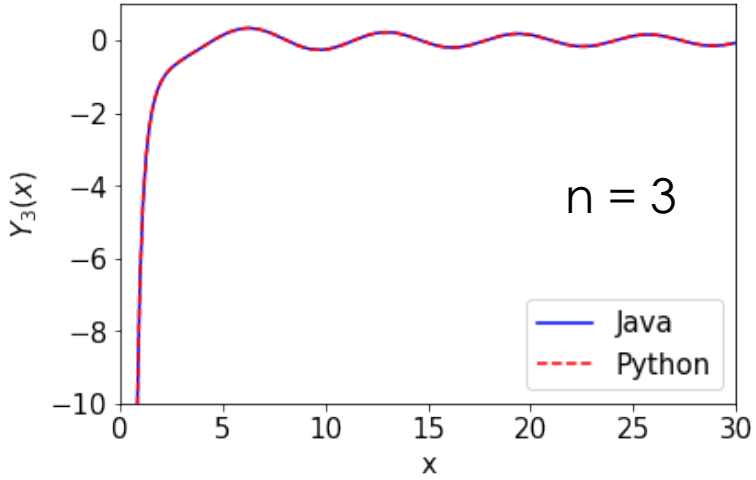
- Solutions to the Bessel ordinary differential equation
- **Comparison with implementation in PYTHON (scipy.special package)**

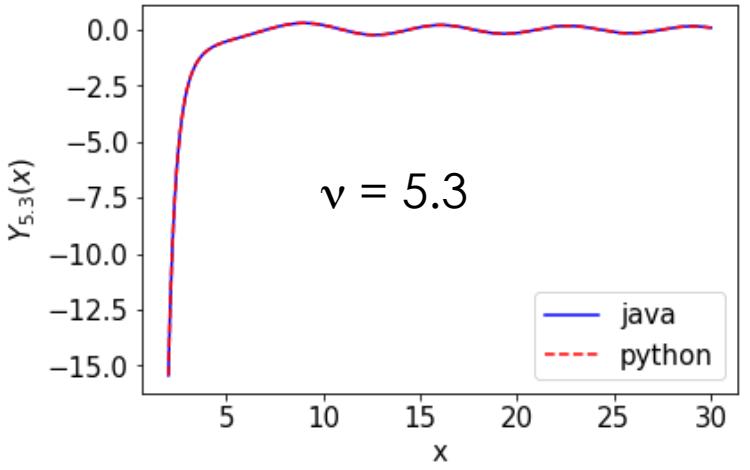












Bessel Function of Second Kind: $Y_\nu(x)$

- Solutions to the Bessel ordinary differential equation
- Asymptotic equation for **Small Arguments**
  - Based on the small argument equation for $J_\nu(x)$
  - Works for complex numbers (look at the phase)

take the limit v → n for integer values

General definition: $$Y_\nu(x) = \frac{J_\nu(x)\cos\nu\pi - J_{-\nu}(x)}{\sin\nu\pi} \quad \text{for} \quad \nu \neq 0, 1, 2, 3, \cdots$$

- Asymptotic equation for **Large Arguments**
  - t = 1/x > 0
  - Can work for complex numbers

$$Y_\nu(x) = \sqrt{\frac{2t}{\pi}} P_A(t) \sin\left(\frac{P_\phi(t)}{t} - (\nu + \frac{1}{2})\frac{\pi}{2}\right)$$

polynomials

eps→0

```
public double yv(double v, double x) {
     if(abs(v-(int)v)>1e-2) {
          return (jv(v, x)*cos(PI*v)-jv(-v, x))/
sin(PI*v) ;
     }
     else {
          double eps = 1e-2 ;
          return (jv(v-eps, x)*cos(PI*(v-eps))-jv(-(v-
eps), x))/sin(PI*(v-eps)) ;
     }
}
```

Hankel Functions: $H_\nu^{(1)}$, $H_\nu^{(2)}$

- Solutions to the Bessel ordinary differential equation
- Represent radially propagating (inward & outward) cylindrical waves
- Definitions:

$$H_\nu^{(1)}(x) = J_\nu(x) + j\,Y_\nu(x) = \frac{J_{-\nu}(x) - e^{-j\pi\nu}J_\nu(x)}{j\sin\pi\nu}$$

Hankel of the first kind

j goes to -j

$$H_\nu^{(2)}(x) = J_\nu(x) - j\,Y_\nu(x) = \frac{e^{j\pi\nu}J_\nu(x) - J_{-\nu}(x)}{j\sin\pi\nu}$$

Hankel of the second kind

- Two $J_\nu(x)$ evaluations are used for $Y_\nu(x)$
- Use the definition based on $J_\nu(x)$ and $J_{-\nu}(x)$
- For integer $\nu = n$, take the limit $\nu \rightarrow n$

```
public Complex hankel1(double v, Complex x) {
     if(abs(v-(int)v)>1e-2) {
          return (-j)*(jv(-v, x)-ComplexMath.exp(-j*PI*v)*jv(v,
x))/sin(PI*v) ;
     }
     else {
          double eps = 1e-2 ;
          return (-j)*(jv(-(v-eps), x)-ComplexMath.exp(-j*PI*(v-
eps))*jv(v-eps, x))/sin(PI*(v-eps)) ;
     }
}
```

Derivative of Bessel Functions

- Solutions to the Bessel ordinary differential equation
- Useful identities (**recurrence** formula) → Exact relations

Main

$$\begin{cases} J_{\nu-1}(x) + J_{\nu+1}(x) = \dfrac{2\nu}{x} J_\nu(x) \\ J_{\nu-1}(x) - J_{\nu+1}(x) = 2J'_\nu(x) \end{cases} \Rightarrow \begin{cases} xJ'_\nu(x) + \nu J_\nu(x) = xJ_{\nu-1}(x) \\ xJ'_\nu(x) - \nu J_\nu(x) = -xJ_{\nu+1}(x) \end{cases}$$

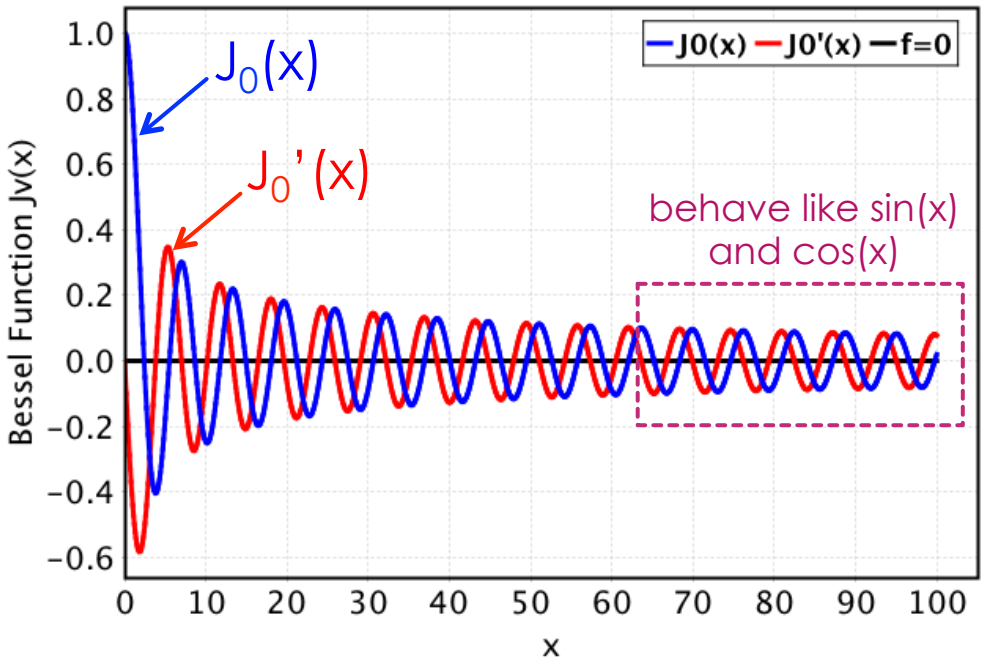
We can **exactly** calculate the derivatives

Neumann

$$\begin{cases} Y_{\nu-1}(x) + Y_{\nu+1}(x) = \dfrac{2\nu}{x} Y_\nu(x) \\ Y_{\nu-1}(x) - Y_{\nu+1}(x) = 2Y'_\nu(x) \end{cases}$$

Hankel

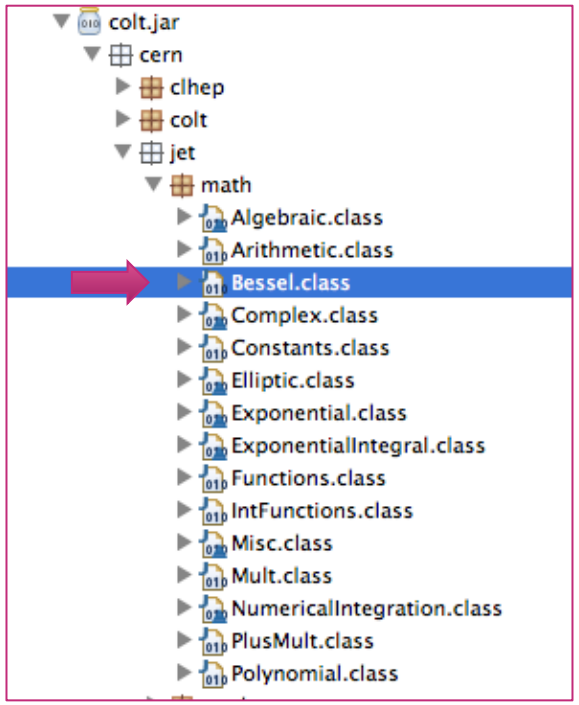
$$\begin{cases} H^{(1,2)}_{\nu-1}(x) + H^{(1,2)}_{\nu+1}(x) = \dfrac{2\nu}{x} H^{(1,2)}_\nu(x) \\ H^{(1,2)}_{\nu-1}(x) - H^{(1,2)}_{\nu+1}(x) = 2H'^{(1,2)}_\nu(x) \end{cases}$$



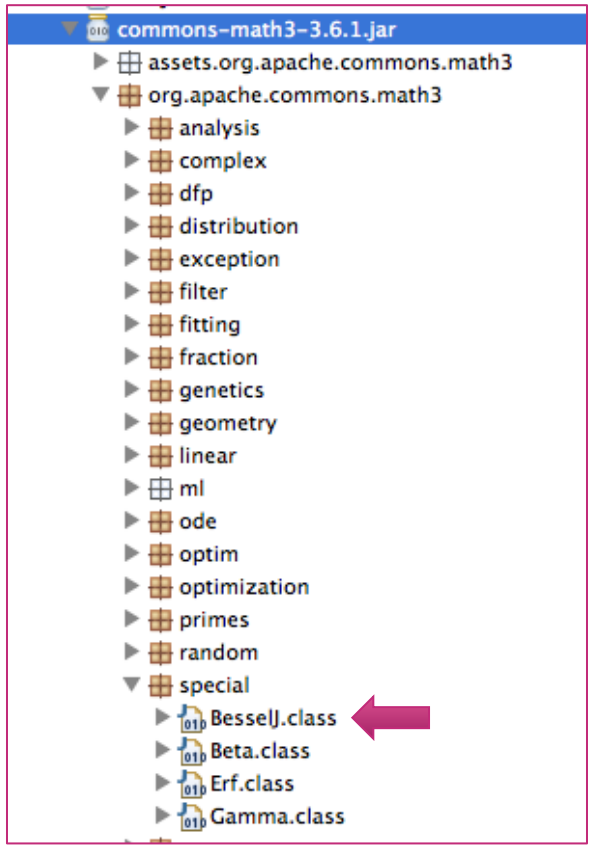
Other Bessel Function Libraries

- Colt library (cern.jet.math.Bessel)
- Apache commons math (org.apache.commons.math3.special.BesselJ)
- Using static methods for function calls

Colt



Apache Commons Math



Problems:

- Slower implementation
- No support for complex numbers
- Mostly implemented for integer n
- Generally don't support double ν
- No support for Hankel functions

Comparison with our implementation

```
public static void main(String[] args) {
      // from my library --> 15 msec
      test1() ;
      // from Colt library (cern.jet.math) --> 52 msec
      test2() ;
      // from apache commons math --> 77 msec
      test3() ;
}
```

Modified Bessel Functions

- Solutions to the Modified Bessel ordinary differential equation
  - Two independent solutions: $I_v(x)$, $K_v(x)$
  - Special case: v is an integer number → $I_n(x)$, $K_n(x)$
- **Modified Bessel differential equation (ODE)**

$$x^2y'' + xy' - (x^2 + \nu^2)y = 0$$

- Integral equations

$$I_\nu(x) = \frac{1}{\pi}\int_0^\pi e^{x\cos\theta}\cos\nu\theta\,d\theta - \frac{\sin\nu\pi}{\pi}\int_0^\infty e^{-x\cosh t-\nu t}\,dt$$

Re(x)>0

$$K_\nu(x) = \int_0^\infty e^{-x\cosh t}\cosh\nu t\,dt$$

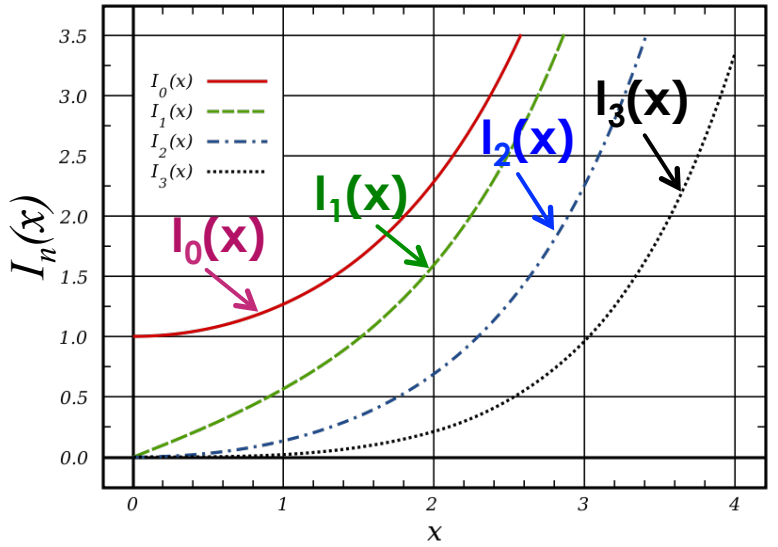
- Relation to Bessel Functions

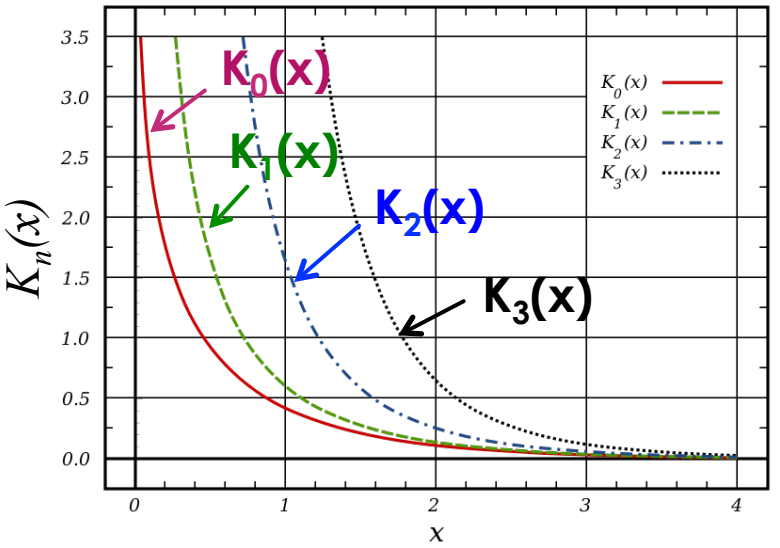
$$I_\nu(x) = j^{-\nu}J_\nu(jx) = \sum_{k=0}^{\infty}\frac{1}{k!\,\Gamma(\nu+k+1)}\left(\frac{x}{2}\right)^{\nu+2k}$$

Imaginary number

$$K_\nu(x) = \frac{\pi}{2}\frac{I_{-\nu}(x) - I_\nu(x)}{\sin\nu\pi}$$

If v → n: take the limit





Modified Bessel of First Kind: $I_\nu(x)$

- Solutions to the Modified Bessel ordinary differential equation
- **Java implementation of $I_n(x)$**

$$I_\nu(x) = \frac{1}{\pi}\int_0^\pi e^{x\cos\theta}\cos\nu\theta\,d\theta - \frac{\sin\nu\pi}{\pi}\int_0^\infty e^{-x\cosh t-\nu t}\,dt$$

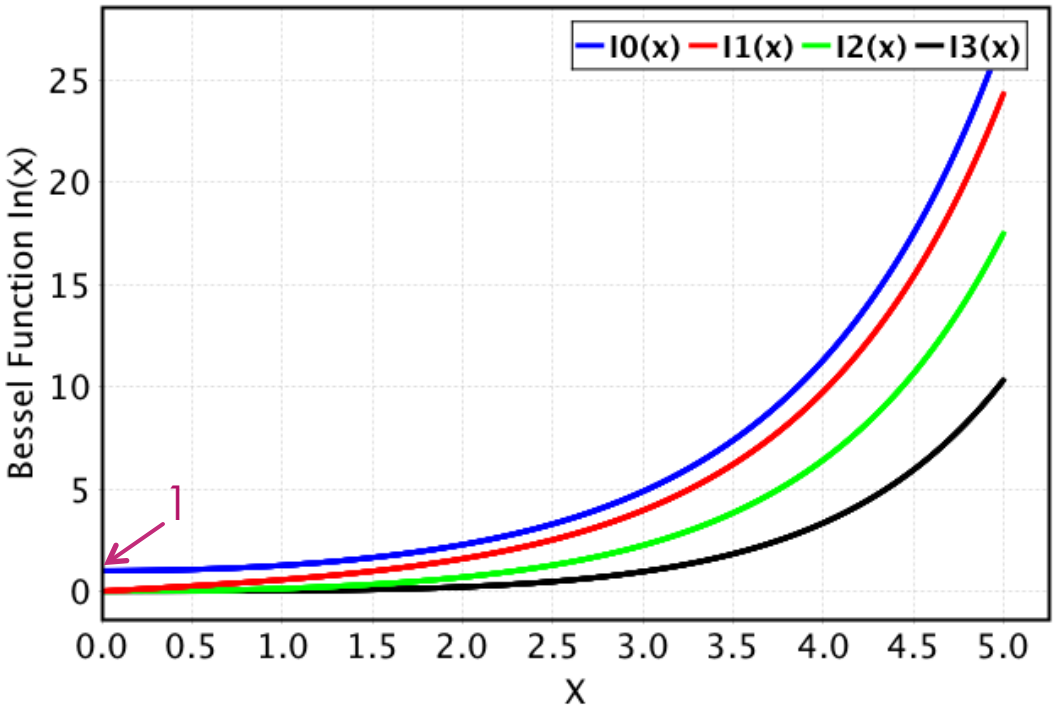
For ν = n, second term vanishes

$I_{-n}(x) = I_n(x)$

Romberg on Simpson for more accuracy

```
public double in(int n, double x) {
      funcIn = t -> 1.0/PI * exp(x*cos(t))*cos(n*t) ;
      integralIn.setIntegralFunction(funcIn);
      return integralIn.rombergOnSimpson(0.0, PI, order) ;
}
```

Default order = 300

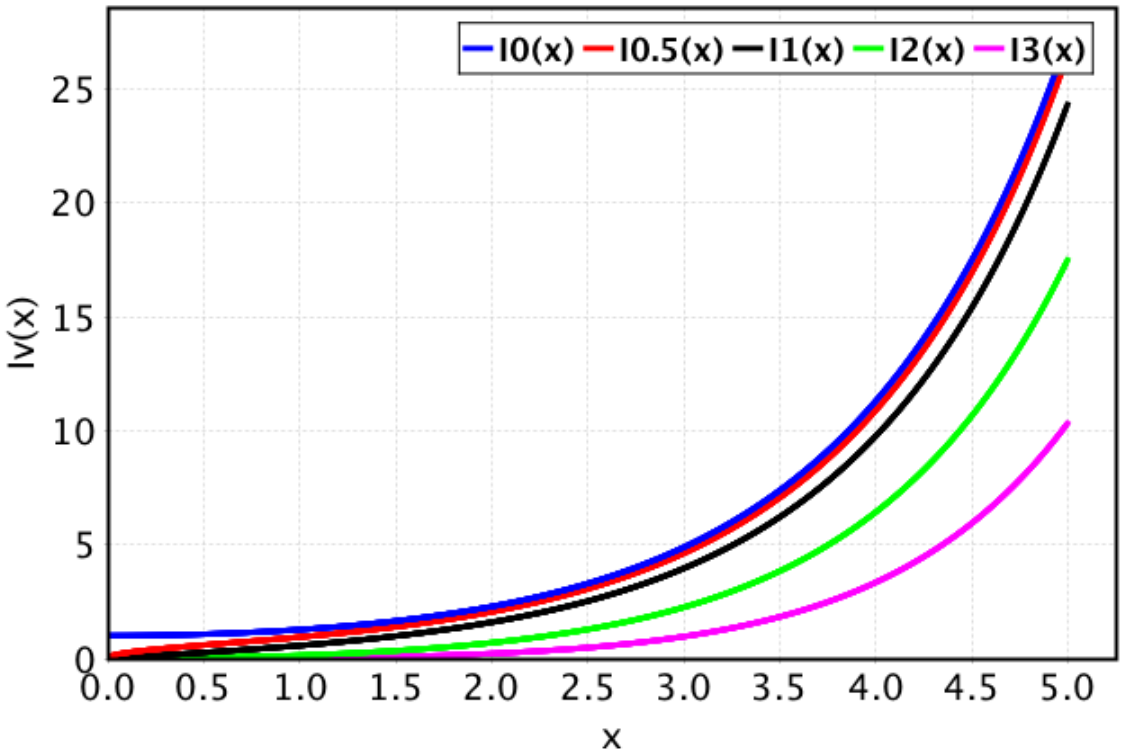



```
ModifiedBesselFunction bessel = new ModifiedBesselFunction() ;

double[] x = MathUtils.linspace(0.0, 5.0, 1000) ;
double[] f0 = ArrayFunc.apply(t -> bessel.in(0, t), x) ;
double[] f1 = ArrayFunc.apply(t -> bessel.in(1, t), x) ;
double[] f2 = ArrayFunc.apply(t -> bessel.in(2, t), x) ;
double[] f3 = ArrayFunc.apply(t -> bessel.in(3, t), x) ;

MatlabChart fig = new MatlabChart() ;
fig.plot(x, f0, "b", 2f, "I0(x)");
fig.plot(x, f1, "r", 2f, "I1(x)");
fig.plot(x, f2, "g", 2f, "I2(x)");
fig.plot(x, f3, "k", 2f, "I3(x)");
fig.renderPlot();
fig.xlabel("X");
fig.ylabel("Modified Bessel Function In(x)");
fig.legendON();
fig.show(true);
```

Modified Bessel of First Kind: $I_\nu(x)$

- Solutions to the Modified Bessel ordinary differential equation
- **Efficient java implementation of $I_\nu(x)$**

$$I_\nu(x) = j^{-\nu} J_\nu(jx) = \sum_{k=0}^{\infty} \frac{1}{k!\,\Gamma(\nu + k + 1)} \left(\frac{x}{2}\right)^{\nu+2k}$$



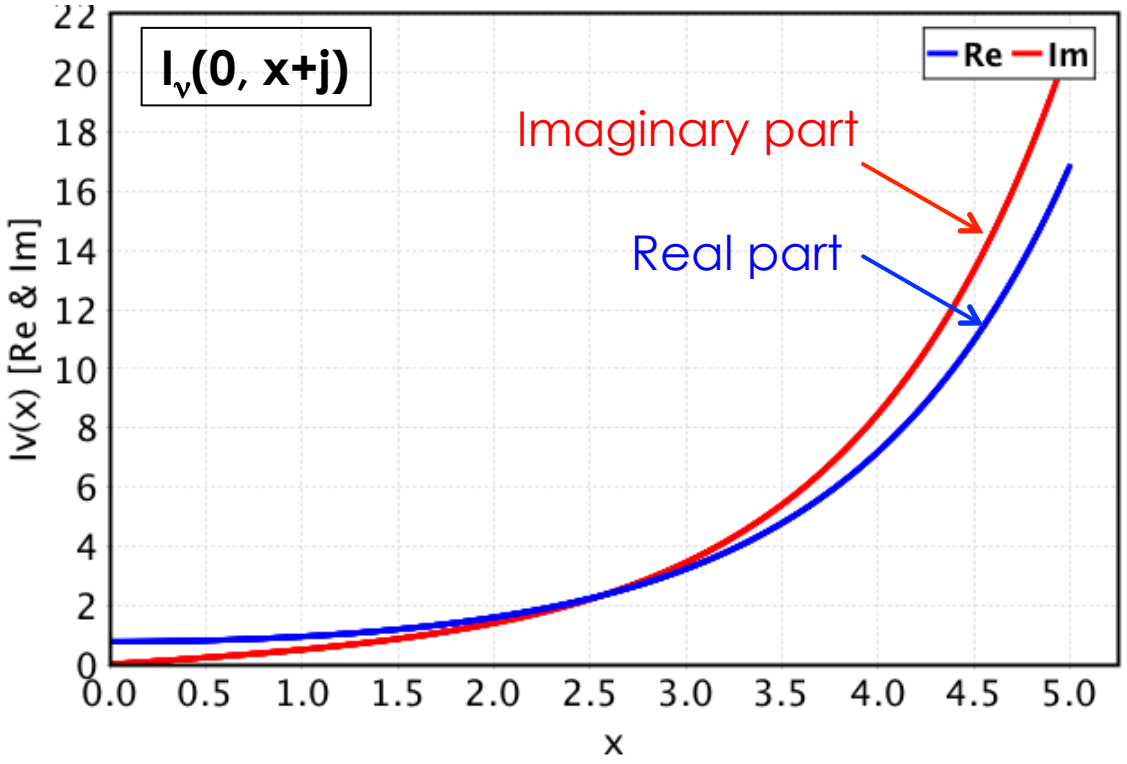
Most efficient implementation

```
private double ivSeries(double v, double x) {
      if(abs(v-lastv)<1e-2) {
            leadTerm = pow(x/2.0, v) / lastGamma ;
      }
      else {
            lastv = v ;
            lastGamma = gammaFunc.gamma(v+1.0) ;
            leadTerm = pow(x/2.0, v) / lastGamma ;
      }
      val = (0.5*x)*(0.5*x) ;
      ivSeq = n -> {
            double result = 1.0 ;
            for(int i=(int)n; i>0; i--) {
                  result = 1.0 + val/(i*(i+v))*result ;
            }
            return result ;
      } ;
      return ivSeq.evaluate(200)*leadTerm ;
}
```

- Cache the value of $\Gamma(v+1)$ when only x is changing

Modified Bessel of First Kind: $I_\nu(x)$

- Solutions to the Modified Bessel ordinary differential equation
- **Efficient java implementation of $I_\nu(x)$ for complex arguments**
- Fast implementation
- No need to use large arguments asymptotic

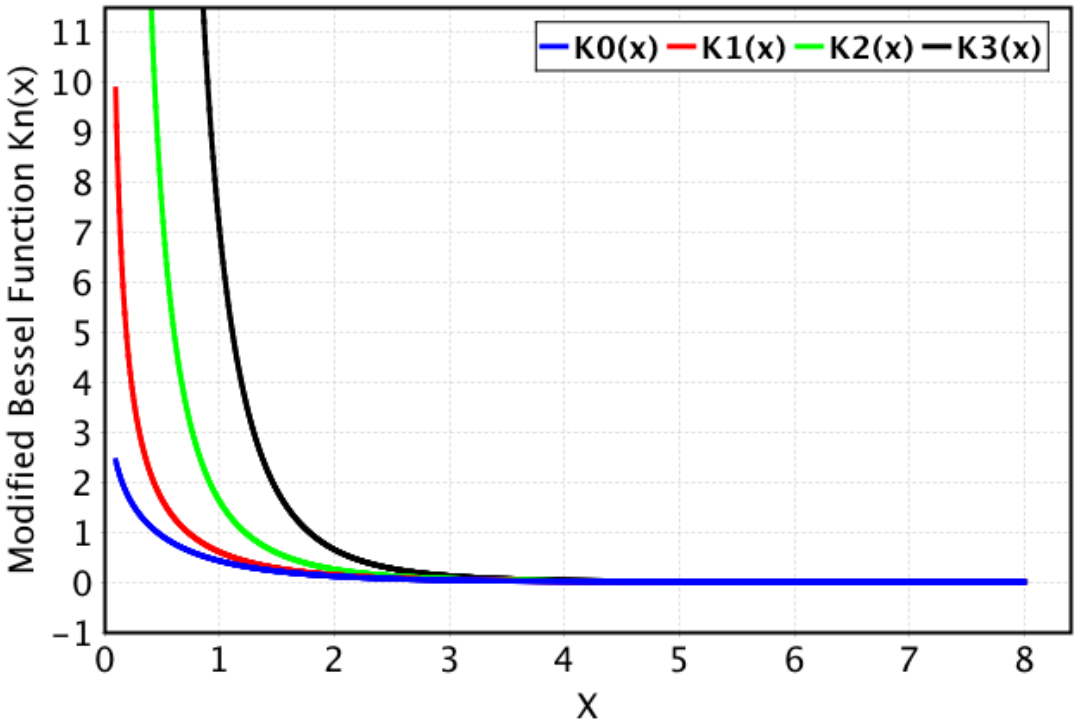



```
private Complex ivSeries(double v, Complex x) {
      if(x.re()<=0.0 && x Im()<0.0)
            return ivSeries(v, x.conjugate()).conjugate() ;
      else if(x.re()>=0.0 && x.im()<0.0)
            return ivSeries(v, x.conjugate()).conjugate() ;
      if(abs(v-lastv)<1e-2) {
            leadTermComplex = ComplexMath.pow(x/2.0, v) / lastGamma ;
      }
      else {
            lastv = v ;
            lastGamma = gammaFunc.gamma(v+1.0) ;
            leadTermComplex = ComplexMath.pow(x/2.0, v) / lastGamma ;
      }
      valComplex = (0.5*x)*(0.5*x) ;
      ivSeqComplex = n -> {
            Complex result = 1.0+0.0*j ;
            for(int i=(int)n; i>0; i--) {
                  result = 1.0 + valComplex/(i*(i+v))*result ;
            }
            return result ;
      } ;
      return ivSeqComplex.evaluate(200)*leadTermComplex ;
}
```

Modified Bessel of Second Kind

- Solutions to the Modified Bessel ordinary differential equation
- **Java implementation of $K_n(x)$**

$$K_\nu(x) = \int_0^\infty e^{-x\cosh t}\cosh \nu t\, dt$$



Romberg on Simpson for more accuracy

```
public double kn(int n, double x) {
      funcKnInf = t -> exp(-x*cosh(t))*cosh(n*t) ;
      integralKn.setIntegralFunction(funcKnInf);
      return integralKn.rombergOnSimpson(0.0, 100.0, order) ;
}
```

Default order = 300

```
ModifiedBesselFunction bessel = new ModifiedBesselFunction() ;

double[] x = MathUtils.linspace(0.0, 5.0, 1000) ;
double[] f0 = ArrayFunc.apply(t -> bessel.kn(0, t), x) ;
double[] f1 = ArrayFunc.apply(t -> bessel.kn(1, t), x) ;
double[] f2 = ArrayFunc.apply(t -> bessel.kn(2, t), x) ;
double[] f3 = ArrayFunc.apply(t -> bessel.kn(3, t), x) ;

MatlabChart fig = new MatlabChart() ;
fig.plot(x, f0, "b", 2f, "K0(x)");
fig.plot(x, f1, "r", 2f, "K1(x)");
fig.plot(x, f2, "g", 2f, "K2(x)");
fig.plot(x, f3, "k", 2f, "K3(x)");
fig.renderPlot();
fig.xlabel("X");
fig.ylabel("Modified Bessel Function Kn(x)");
fig.legendON();
fig.show(true);
```

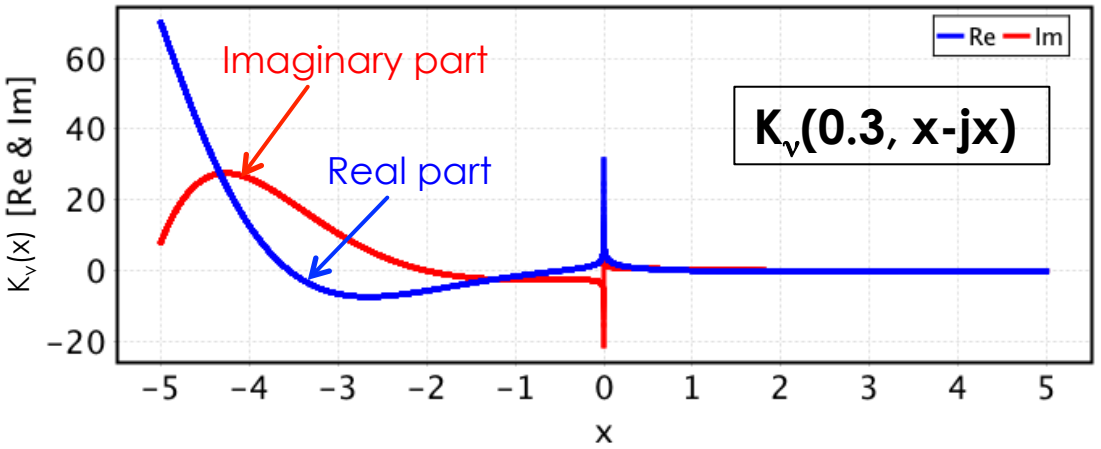
Modified Bessel of Second Kind

- Solutions to the Modified Bessel ordinary differential equation
- **Efficient Java implementation of $K_v(x)$**
- For small x:

$$K_\nu(x) = \frac{\pi}{2}\frac{I_{-\nu}(x) - I_\nu(x)}{\sin \nu\pi}$$

- For large x:

$$K_\nu \approx \sqrt{\frac{0.5\pi}{x}} e^{-x}\left(1 + \frac{4v^2-1}{8x} + \frac{(4v^2-1)(4v^2-9)}{128x^2} + \frac{(4v^2-1)(4v^2-9)(4v^2-25)}{3072x^3}\right)$$



Most efficient implementation

```
public Complex kvSmallArgument(double v, Complex x) {
      if(abs(v-(int)v)>1e-2) {
            return PI/2.0*(iv(-v, x)-iv(v, x))/sin(PI*v) ;
      }
      else {
            double eps = 1e-2 ;
            return PI/2.0*(iv(-(v-eps), x)-iv(v-eps, x))/
sin(PI*(v-eps)) ;
      }
}
```

```
public Complex kvLargeArgument(double v, Complex x) {
      Complex lead = ComplexMath.sqrt(0.5*PI/
x)*ComplexMath.exp(-x) ;
      double a0 = 1.0 ;
      double a1 = 4.0*v*v-1.0 ;
      double a2 = 4.0*v*v-9.0 ;
      double a3 = 4.0*v*v-25.0 ;
      Complex terms = a0 + a0*a1/(8.0*x) + a0*a1*a2/
(2.0*64*x*x) + a0*a1*a2*a3/(6.0*512.0*x*x*x) ;
      return lead*terms ;
}
```

Gamma Function: $\Gamma(z)$

- Gamma function generalizes **the concept of factorial** 4! = 4*3*2*1=24, 0! = 1
- For integer numbers (n>0) we get factorial: $\Gamma(n) = (n-1)!$
- For non-integer & complex numbers: $\Gamma(z+1) = z\,\Gamma(z)$ with $\Gamma(1) = 1$ → **Definition based on a property**
- **Integral definition for Re(z) > 0**
  - Relatively fast convergence for small z > 0
  - We can choose [0, 100] as the integration interval

$$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x}\, dx$$

```
public double gamma(double x) {
      funcGamma = t -> pow(t, x-1) * exp(-t) ;
      integralGamma.setIntegralFunction(funcGamma) ;
      return integralGamma.simpson(0.0, 100.0, order) ;
}
```

- **Product Series definition**
  - Painfully slow convergence
  - Use richardson acceleration

$$\Gamma(z) = \frac{1}{z} \prod_{n=1}^{\infty} \frac{\left(1 + \frac{1}{n}\right)^z}{1 + \frac{z}{n}}$$

```
public double gamma2(double x) {
      gammaSeq = n -> pow(1.0+1.0/n, x)/(1.0+x/n) ;
      gammaSeqProd = n -> {
            double result = 1.0 ;
            for(int i=1; i<n+1; i++)
                  result *= gammaSeq.evaluate(i) ;
            return result ;
      } ;
      return gammaSeqProd.richardson4().evaluate(300)/x ;
}
```

Richardson acceleration

- Implement in java using both definitions

Gamma Function: Γ(x)

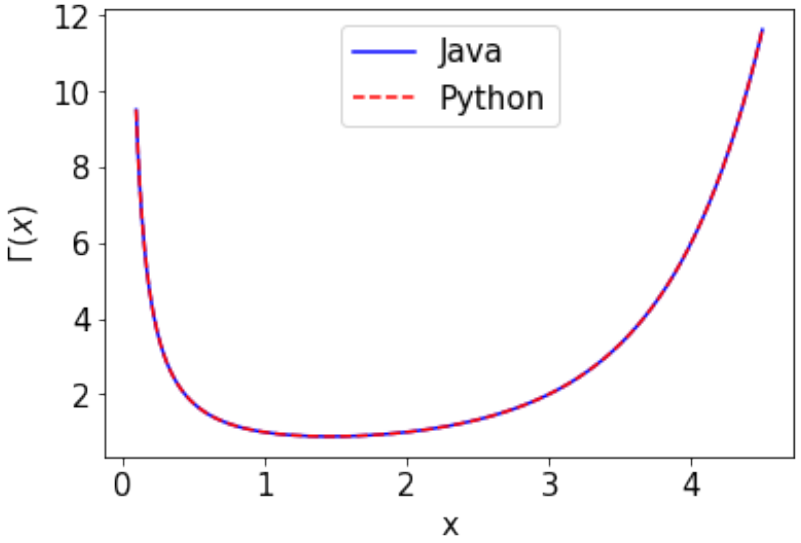
- Product Series definition
  - Singularity at **z = -n**
  - z = 0, -1, -2, -3, …

$$\Gamma(z) = \frac{1}{z}\prod_{n=1}^{\infty}\frac{\left(1+\frac{1}{n}\right)^{z}}{1+\frac{z}{n}}$$

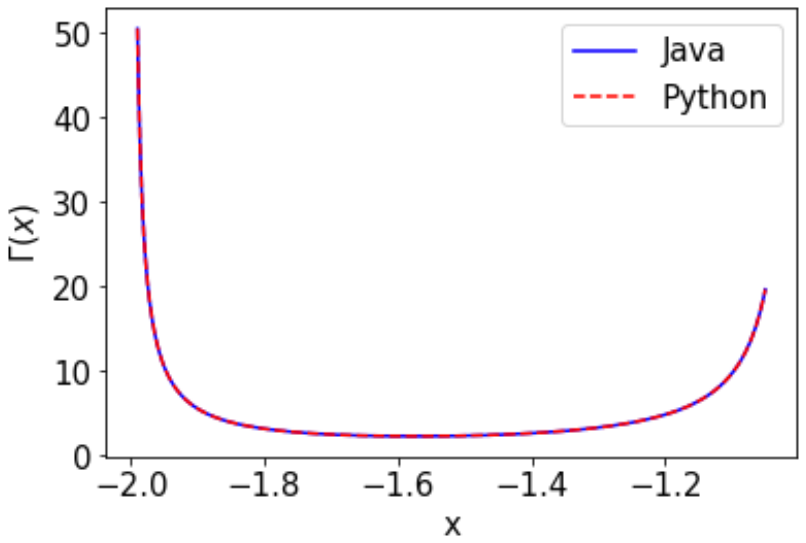
- Comparison with implementation in PYTHON
  - **scipy.special** package

```
public double gamma2(double x) {
      gammaSeq = n -> pow(1.0+1.0/n, x)/(1.0+x/n) ;
      gammaSeqProd = n -> {
            double result = 1.0 ;
            for(int i=1; i<n+1; i++)
                  result *= gammaSeq.evaluate(i) ;
            return result ;
      } ;
      return gammaSeqProd.richardson4().evaluate(300)/x ;
}
```

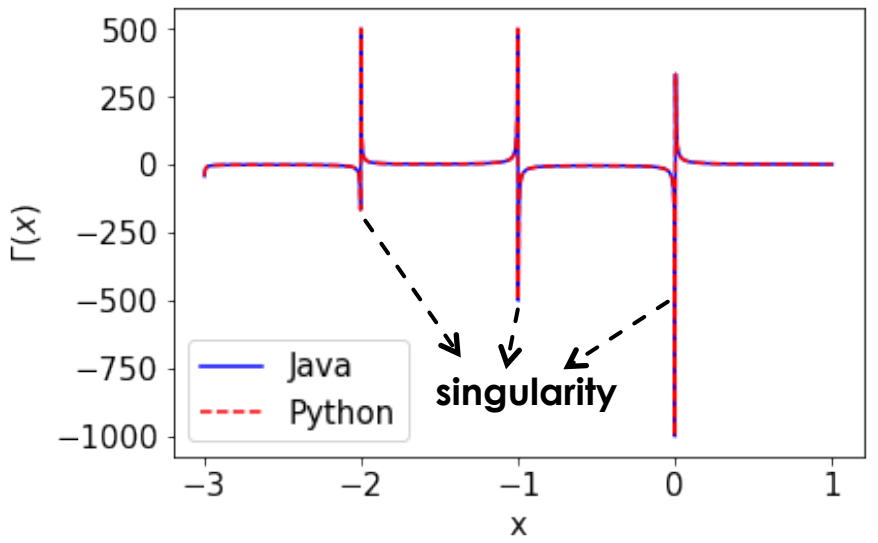
0 < x < 6



-2 < x < -1



-3 < x < 1



Gamma Function: Γ(x)

- **Euler**'s definition

$$\Gamma(z) = \frac{1}{z}\prod_{n=1}^{\infty}\frac{\left(1+\frac{1}{n}\right)^{z}}{1+\frac{z}{n}}$$

➡ Γ(1) = 1, Γ(z+1) = z Γ(z)

Any definition that satisfies these two properties is a valid representation of Gamma function

- **Weierstrass**'s definition
  - γ = 0.577216

Euler–Mascheroni constant

$$\Gamma(z) = \frac{e^{-\gamma z}}{z}\prod_{n=1}^{\infty}\frac{e^{z/n}}{\left(1+\frac{z}{n}\right)}$$

➡ Γ(1) = 1, Γ(z+1) = z Γ(z)

- Which one is faster for evaluation?
  - Weierstrass
  - Both work for complex numbers
- **Euler's reflection formula**

$$\Gamma(1-z)\,\Gamma(z) = \frac{\pi}{\sin(\pi z)} \quad \Rightarrow \quad \Gamma(z)\Gamma(-z) = \frac{-\pi}{z\sin(\pi z)}$$

```
public double gamma3(double x) {
      gammaSeqProd = n -> {
            double result = 1.0 ;
            for(int i=1; i<n+1; i++) {
                  result *= 1.0+x/i ;
            }
            return result ;
      } ;
      Sequence sum = n -> {
            double result = 0.0 ;
            for(int i=1; i<n+1; i++)
                  result += 1.0/i ;
            return result-g ;
      } ;
      Sequence tot = n -> exp(x*sum.evaluate(n))/
gammaSeqProd.evaluate(n) ;
      return tot.richardson4().evaluate(300)/x ;
}
```

Gamma Function: Γ(x)

- **Euler**'s definition

$$\Gamma(z) = \frac{1}{z}\prod_{n=1}^{\infty}\frac{\left(1+\frac{1}{n}\right)^z}{1+\frac{z}{n}} \rightarrow \Gamma(z) = \frac{1}{z}\frac{\left(\prod_{n=1}^{\infty}\left(1+\frac{1}{n}\right)\right)^z}{\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)}$$

1 Calculate this first; 2 $(\ldots)^z$

→ Efficient way of evaluating the product

- **Weierstrass**'s definition
  - γ = 0.577216

$$\Gamma(z) = \frac{e^{-\gamma z}}{z}\prod_{n=1}^{\infty}\frac{e^{z/n}}{\left(1+\frac{z}{n}\right)} \rightarrow \Gamma(z) = \frac{1}{z}\frac{e^{\left(-\gamma+\sum_{n=1}^{\infty}\frac{1}{n}\right)z}}{\prod_{n=1}^{\infty}\left(1+\frac{z}{n}\right)}$$

1 Calculate this first; 2 $e^{(\ldots)z}$

→ Efficient way of evaluating the product

- Which one is faster for evaluation?
- **It all depends on how we calculate!!!**
  - Weierstrass is slightly faster

The key is to reduce the number of times elementary functions such as exp(…) and pow(…) are used.

Most efficient implementation of Euler's definition

```
public double gamma4(double x) {
        gammaSeq = n -> {
                double result = 1.0 ;
                for(int i=1; i<n+1; i++)
                        result *= 1.0 + 1.0/i ;
                return result ;
        } ;
        gammaSeqProd = n -> {
                double result = 1.0 ;
                for(int i=1; i<n+1; i++) {
                        result *= 1.0+x/i ;
                }
                return result ;
        } ;
        Sequence tot = n -> pow(gammaSeq.evaluate(n), x)/gammaSeqProd.evaluate(n) ;
        return tot.richardson4().evaluate(300)/x ;
}
```

Digamma Function: φ(x)

- Relates to the derivative of gamma function
- Formal definition (also known as polygamma)

$$\psi(x) = \frac{d}{dx}\ln(\Gamma(x)) = \frac{\Gamma'(x)}{\Gamma(x)}$$

Do not calculate φ(x) using Γ(x)

Singularities at x = 0, -1, -2, …

- Series representation (**small x**)
  - for x ≠ 0,-1,-2,…

$$\psi(x) = -\gamma + \sum_{k=0}^{\infty} \frac{x-1}{(k+1)(k+x)}$$

Euler–Mascheroni constant

- Series representation (**large x**)
  - Fast evaluation
  - Very good accuracy for x>5

$$\psi(x) \approx \log x - \frac{1}{2x} - \frac{1}{12x^2} + \frac{1}{120x^4} - \frac{1}{252x^6} + \frac{1}{240x^8} - \frac{5}{660x^{10}} + \frac{691}{32760x^{12}} - \frac{1}{12x^{14}} + \cdots$$

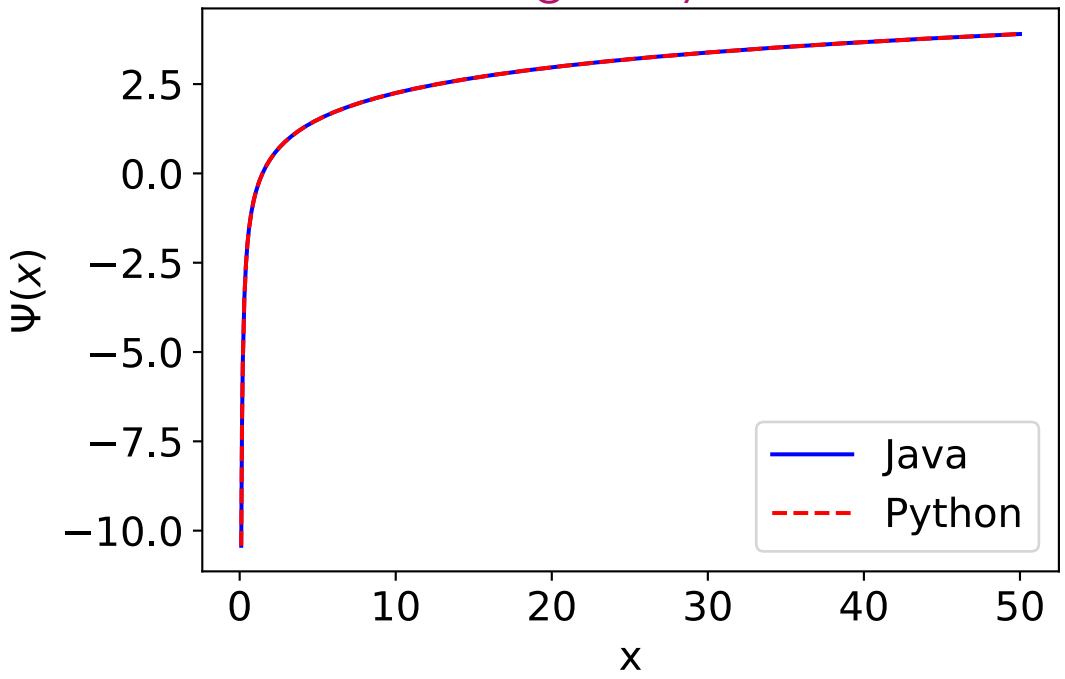
```
public double digamma(double x) {
      seqSum = n -> {
            double result = 0.0 ;
            for(int i=0; i<n; i++)
                  result += 1.0/((i+1.0)*(i+x)) ;
            return result ;
      } ;

      if(abs(x)<=5.0)
            return -g + (x-1)*seqSum.richardson4().evaluate(100) ;
      else if(x>5.0) {
            return log(x)-1.0/(2.0*x)-1.0/(12.0*x*x)+1.0/(120.0*x*x*x*x)
                        -1.0/(252.0*pow(x,6))+1.0/(240.0*pow(x,8))-5.0/
(660.0*pow(x,10))+691.0/(32760.0*pow(x,12))-1.0/(12.0*pow(x,14)) ;
      }
      else {
            return -g + (x-1)*seqSum.richardson4().evaluate(300) ;
      }
}
```

4th-order Richardson acceleration

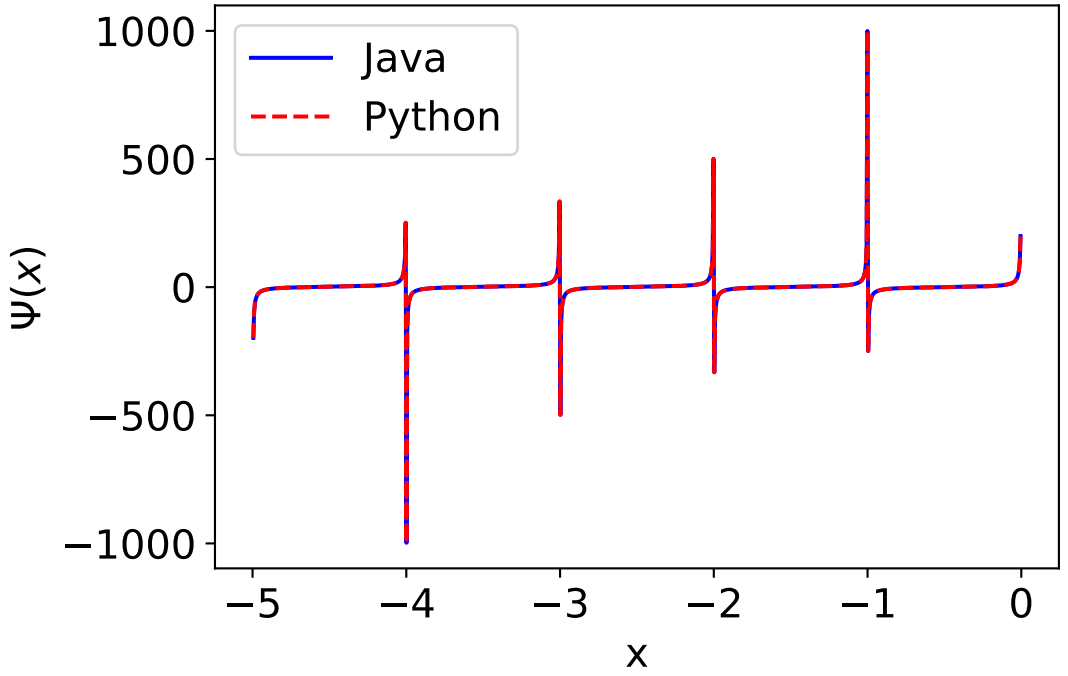
Digamma Function: φ(x)

- Relates to the derivative of gamma function
- Comparison with implementation in PYTHON
  - **scipy.special** package (scipy.special.digamma)

No singularity for x >0



Singularities at x = -n



Euler-Mascheroni Number: γ

- γ is the limit of a simple sequence

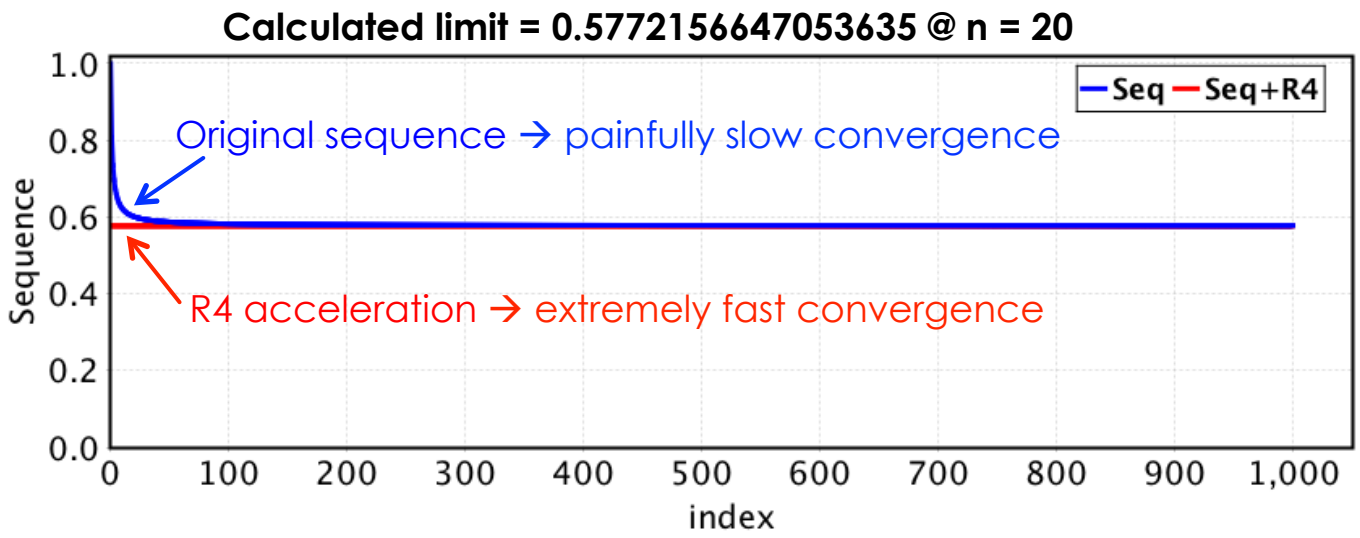
Natural log

Harmonic number

Euler–Mascheroni constant
0.57721566490153286060

$$\gamma = \lim_{n\to\infty}\left(-\ln n + \sum_{k=1}^{n}\frac{1}{k}\right)$$

- Note that the sum of 1/k diverges by itself
- However when subtracting log(n), it converges
- Error ~ $1/n^2$ → Richardson acceleration (4^{th} order)




```
public static void eulerMacheroni() {
      Sequence seq = n -> {
            double result = 0.0 ;
            for(int k=1; k<=n; k++)
                  result += 1.0/k ;
            return result - Math.log(n) ;
      } ;

      Sequence seqR4 = seq.richardson4() ;
      System.out.println("limit = " + seqR4.evaluate(20));

      double[] index = new double[1000] ;
      double[] seqVals = new double[index.length] ;
      double[] seqValsR4 = new double[index.length] ;

      for(int i=0, len=index.length; i<len; i++) {
            index[i] = i+1 ;
            seqVals[i] = seq.evaluate(i+1) ;
            seqValsR4[i] = seqR4.evaluate(i+1) ;
      }

      MatlabChart fig = new MatlabChart() ;
      fig.plot(index, seqVals, "b", 2f, "Seq");
      fig.plot(index, seqValsR4, "r", 2f, "Seq+R4");
      fig.renderPlot();
      fig.legendON();
      fig.xlabel("index");
      fig.ylabel("Sequence");
      fig.show(true);
}
```

Beta Function: B(x, y)

- A two-dimensional representation of combinatory function 1/C(n,m+n) = n!m!/(n+m)!
- Formal definition

for all x, y: $B(x,y) = \dfrac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$

for all Re(x)>0, Re(y)>0: $B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,dt$

```
public double beta(double x, double y) {
      return gammaFunc.gamma(x)*gammaFunc.gamma(y)/gammaFunc.gamma(x+y) ;
}

public Complex beta(Complex x, Complex y) {
      return gammaFunc.gamma(x)*gammaFunc.gamma(y)/gammaFunc.gamma(x+y) ;
}
```

- Some properties

$$B(x,y) = B(y,x)$$

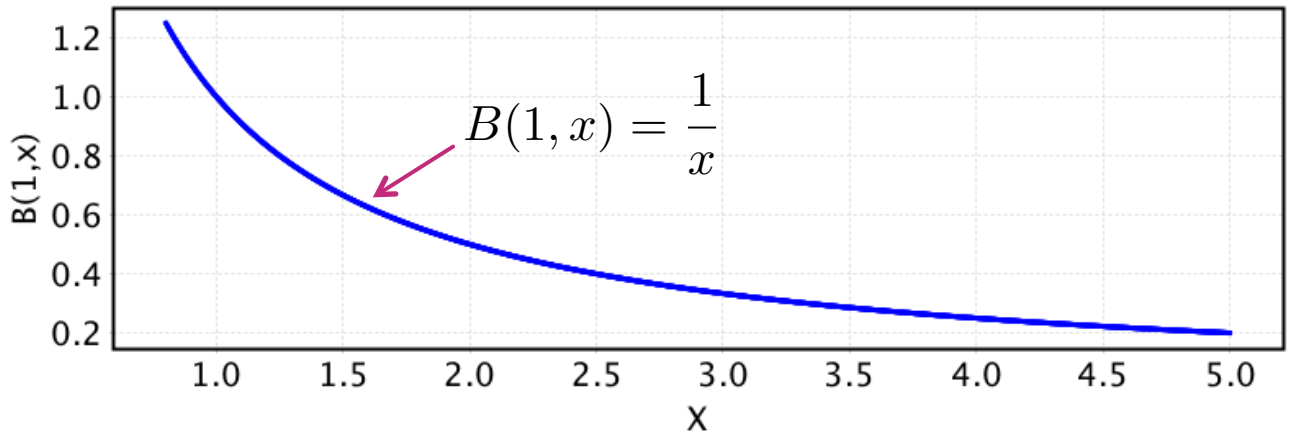
$$B(x,y) = B(x,y+1) + B(x+1,y)$$

$$B(x+1,y) = B(x,y)\frac{x}{x+y}$$

$$B(x,y+1) = B(x,y)\frac{y}{x+y}$$

$$B(x,y)B(x+y,1-y) = \frac{\pi}{x\sin(\pi y)}$$

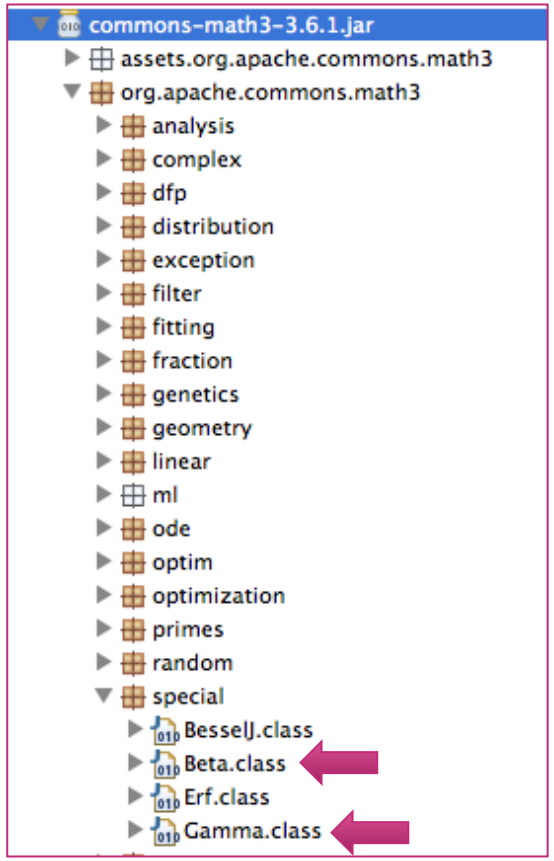
$$B(x,1-x) = \frac{\pi}{\sin(\pi x)}$$ → Reflection formula



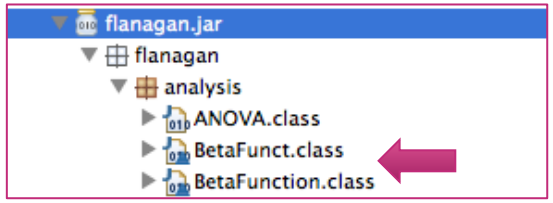
Other Gamma Function Libraries

- Gamma function and its related functions
- Apache commons (org.apache.commons.math3.special.*)

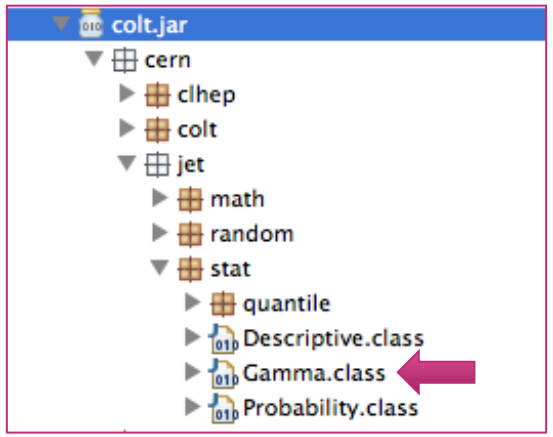
Apache Commons Math



Flanagan library



Colt library



Problems:

- Slower implementation
- No support for complex numbers

Speed comparison

```
public static void main(String[] args) {
     // from my library --> 37 msec
     test1() ;
     // from Colt library (cern.jet.stat) --> 46 msec
     test2() ;
     // from apache commons math --> 69 msec
     test3() ;
     // from flanagan --> 78 msec
     test4() ;
}
```

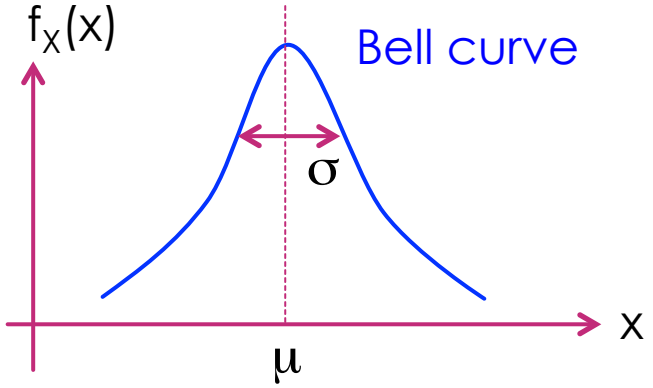
Error Function: erf(x)

- Most natural phenomena follow a normal distribution of probabilities
- Gaussian distribution

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

μ : statistical expected value (mean)

σ : statistical standard deviation



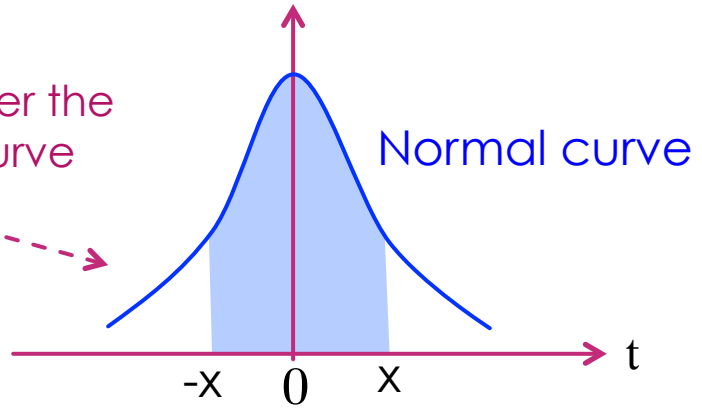
- Normal distribution
  - $\mu = 0, \sigma = 1$ ➡ $f_{X_n}(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ ➡ $\int_{-\infty}^{\infty} f_X(x)dx = 1$
- Error function

$$\text{erf}(x) = \int_{-x}^{x} f_{X_n}(\sqrt{2}t)d(\sqrt{2}t) \Rightarrow \text{erf}(x) = \int_{-x}^{x} \frac{1}{\sqrt{\pi}} e^{-t^2} dt$$

Area under the normal curve

Erf(∞) = 1

Erf(-∞) = -1



Erf(-x) = -Erf(x)
(odd symmetry)

- Complementary error function
  - Prob(|t| > x) for x>0

$$\text{erfc}(x) = 1 - \text{erf}(x)$$

Prob(|t| < x) = Erf(x)

Prob(|t| > x) = Erfc(x)

Error Function

- Small-argument expansion (|**x**|**<4**)

$$\operatorname{erf}(x) = \frac{2x}{\sqrt{\pi}} \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{k!(2k+1)}$$

- Clever summation
  - Calculate x^2 only once

$$\sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{k!(2k+1)} = 1 - \frac{x^2}{1\times 3}\left(1 - \frac{x^2\times 3}{2\times 5}\left(1 - \frac{x^2\times 5}{3\times 7}(1-\cdots)\right)\right)$$

k=0 k=1 k=2 k=3

Very efficient implementation of Erf(x)

```
private double erfSmallArgument(double x) {
      leadTerm = 2.0/sqrt(PI)*x ;
      val = x*x ;
      erfSeq = n -> {
            double result = 1.0 ;
            for(int i=(int)n; i>0; i--) {
                  result = 1.0 - val*(2.0*i-1.0)/(i*(2.0*i+1.0))*result ;
            }
            return result ;
      } ;
      return erfSeq.evaluate(200)*leadTerm ;
}
```

- Large-argument expansion (|**x**|**>4**)

$$\operatorname{erf}(x) \sim 1 - \frac{e^{-x^2}}{x\sqrt{\pi}} \sum_{m=0}^{\infty} (-1)^m \frac{1\cdot 3\cdot 5\cdots(2m-1)}{(2x^2)^m}$$

$$\operatorname{erfc}(x) \sim \frac{e^{-x^2}}{x\sqrt{\pi}} \sum_{m=0}^{\infty} (-1)^m \frac{1\cdot 3\cdot 5\cdots(2m-1)}{(2x^2)^m}$$

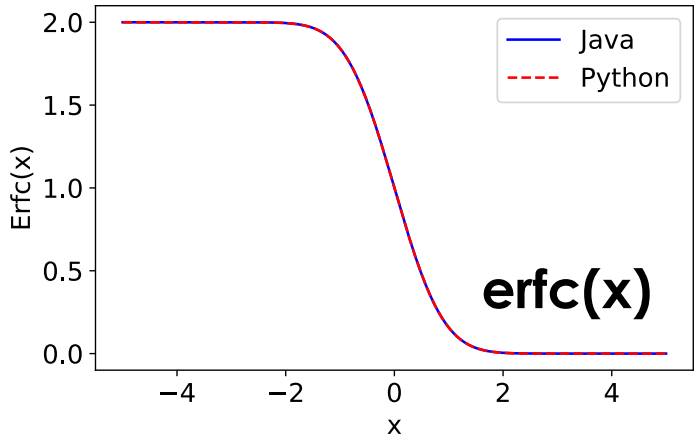
x is positive

Erf(-x) = -Erf(x)

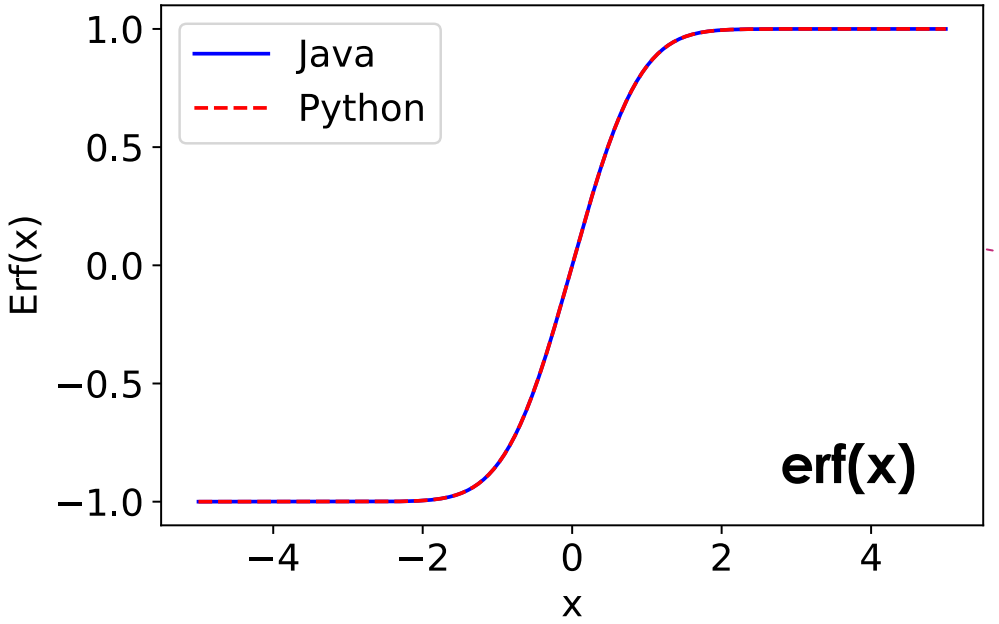
```
private double erfLargeArgument(double x) {
      if(x<0.0)
            return -erfLargeArgument(-x) ;
      leadTerm = exp(-x*x)/(abs(x)*sqrt(PI)) ;
      val = x*x ;
      double a0 = 1.0
      double a1 = 0.5/val ;
      double a2 = 0.75/(val*val) ;
      double a3 = 1.875/(val*val*val) ;
      return 1.0-leadTerm*(a0-a1+a2-a3) ;
}
```

Error Function

- Comparison with **Scipy.Special** package in **PYTHON**
- Java implementation has good accuracy and is fast
- erfc(x) = 1 – erf(x)



-3 < x < 3 is the important interval




```
public static void test1() {
      ErrorFunction func = new ErrorFunction() ;

      double[] x = MathUtils.linspace(-5.0, 5.0, 1000) ;
      Timer timer = new Timer() ;
      timer.start();
      double[] erf = ArrayFunc.apply(t -> func.erf(t), x) ;
      timer.stop();
      timer.show();

      MatlabChart fig = new MatlabChart() ;
      fig.plot(x, erf, "r");
      fig.renderPlot();
      fig.show(true);
}
```

Complex Error Function

- Series expansion based on real and imaginary parts
- z = x + jy → erf(z) = term1 + term2 + term3

implement as sequences

$$\operatorname{erf}(z) = \operatorname{erf}(x+jy) = \operatorname{erf}(x) + \frac{e^{-x^2}}{2\pi x}\left[(1-\cos 2xy) + j\sin 2xy\right] + \frac{2}{\pi}e^{-x^2}\sum_{k=1}^{\infty}\frac{e^{-k^2/4}}{k^2+4x^2}\left[f_k(x,y) + jg_k(x,y)\right]$$

reSeq $f_k(x,y) = 2x(1-\cos 2xy\cosh ky) + k\sin 2xy\sinh ky$

imSeq $g_k(x,y) = 2x\sin 2xy\cosh ky + k\cos 2xy\sinh ky$

Handling x = 0 singularity

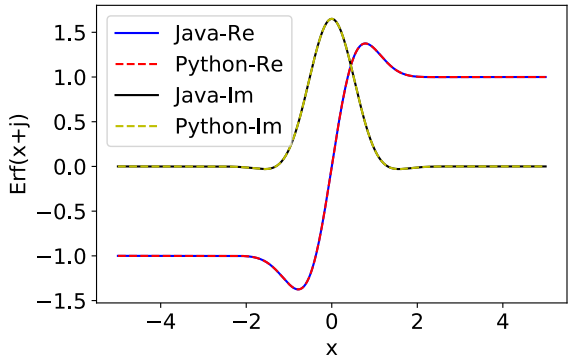
sinc(t) = sin(t)/t

- Has good accuracy even for large |z|
- Make sure to handle x=0 singularity in the second term

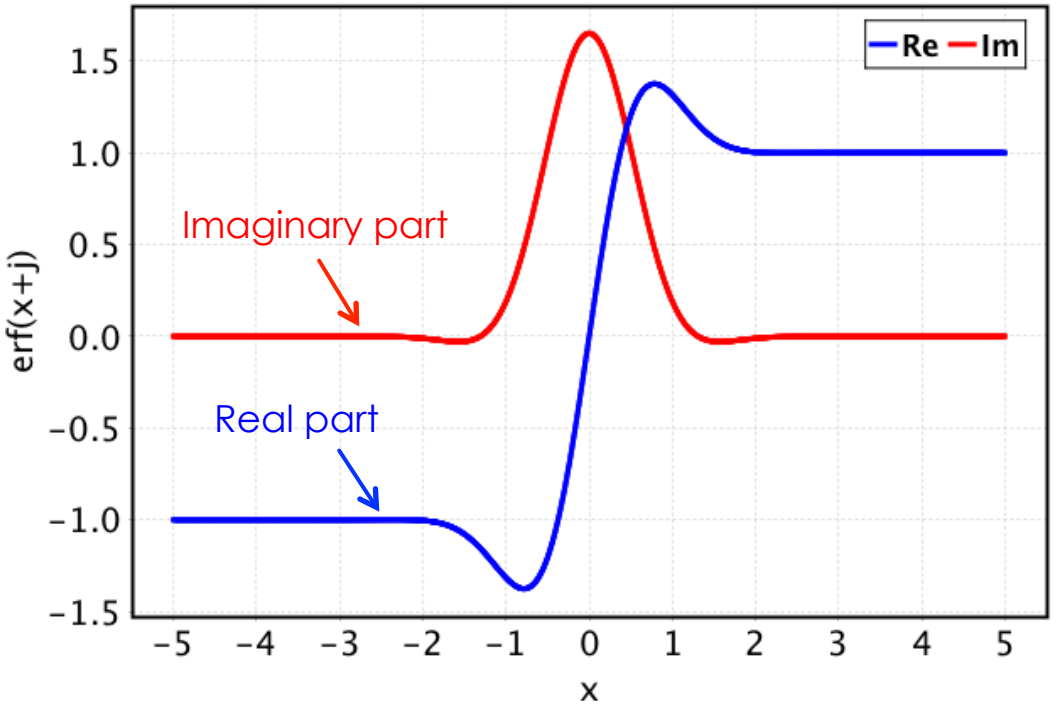
```
public Complex erf(Complex z) {
      double x = z.re() ;
      double y = z.im() ;
      double coeff = exp(-x*x)/PI ;
      double term1 = erf(x) ;
      Complex term2 = coeff*y*sinc(x*y)*(sin(x*y)+j*cos(x*y)) ;
      reSeq = n -> 2.0*x-2.0*x*cosh(n*y)*cos(2.0*x*y)+n*sinh(n*y)*sin(2.0*x*y) ;
      imSeq = n -> 2.0*x*cosh(n*y)*sin(2.0*x*y)+n*sinh(n*y)*cos(2.0*x*y) ;
      reSum = n -> Series.sum(m -> exp(-0.25*m*m)/(m*m+4.0*x*x)*reSeq.evaluate(m), 1, n) ;
      imSum = n -> Series.sum(m -> exp(-0.25*m*m)/(m*m+4.0*x*x)*imSeq.evaluate(m), 1, n) ;
      Complex term3 = 2.0*coeff*(reSum.evaluate(50)+j*imSum.evaluate(50)) ;
      return term1 + term2 + term3 ;
}
```

Complex Error Function

- Series expansion based on real and imaginary parts
- z = x + jy → erf(z) = term1 + term2 + term3
- Plot erf(x+j) with -5 < x < 5
- High computation efficiency and good accuracy



Comparison with Python




```
public static void test10() {
      ErrorFunction func = new ErrorFunction() ;

      double[] x = MathUtils.linspace(-5.0, 5.0, 1000) ;
      Timer timer = new Timer() ;
      timer.start();
      Complex[] erf = new Complex[x.length] ;
      for(int i=0, len=x.length; i<len; i++)
            erf[i] = func.erf(x[i]+j) ;
      timer.stop();
      timer.show();

      MatlabChart fig = new MatlabChart() ;
      fig.plot(x, erf);
      fig.renderPlot();
      fig.xlabel("x");
      fig.ylabel("erf(x+j)");
      fig.legendON();
      fig.show(true);
}
```

Family of Error Functions

- Inverse error function: **erfinv(z)**

$$y = \mathrm{erfinv}(z) = \mathrm{erf}^{-1}(z) \rightarrow \mathrm{erf}(\mathrm{y}) = z$$

- Imaginary error function: **erfi(z)**

$$\mathrm{erfi}(z) = -j\,\mathrm{erf}(jz)$$

- Scaled complementary error function: **erfcx(z)**

$$\mathrm{erfcx}(z) = e^{z^2}\,\mathrm{erfc}(z)$$

- Faddeeva function: **w(z)**

$$\mathrm{w}(z) = e^{-z^2}\,\mathrm{erfc}(-jz) = \mathrm{erfcx}(-jz)$$

- Dawson function: **D(z)**

$$\mathrm{D}(z) = \frac{\sqrt{\pi}}{2} e^{-z^2}\,\mathrm{erfi}(z)$$

- Derivative and integral of error function

$$\frac{d}{dz}\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}} e^{-z^2} \qquad \int \mathrm{erf}(z)\,dz = z\,\mathrm{erf}(z) + \frac{1}{\sqrt{\pi}} e^{-z^2}$$

Using root finding for erfinv(x)

```
public double erfinv(double x) {
      if(abs(x)>=1.0)
            return Double.NaN ;
      funcErfInv = t -> erf(t)-x ;
      rootFinderErfInv = new RealRoot(funcErfInv) ;
      return rootFinderErfInv.newton(0.0, 100) ;
}
```

```
public Complex faddeeva(Complex z) {
      return ComplexMath.exp(-z*z)*erfc(-j*z) ;
}

// another name for faddeeva: w(z)
public Complex wofz(Complex z) {
      return ComplexMath.exp(-z*z)*erfc(-j*z) ;
}
```

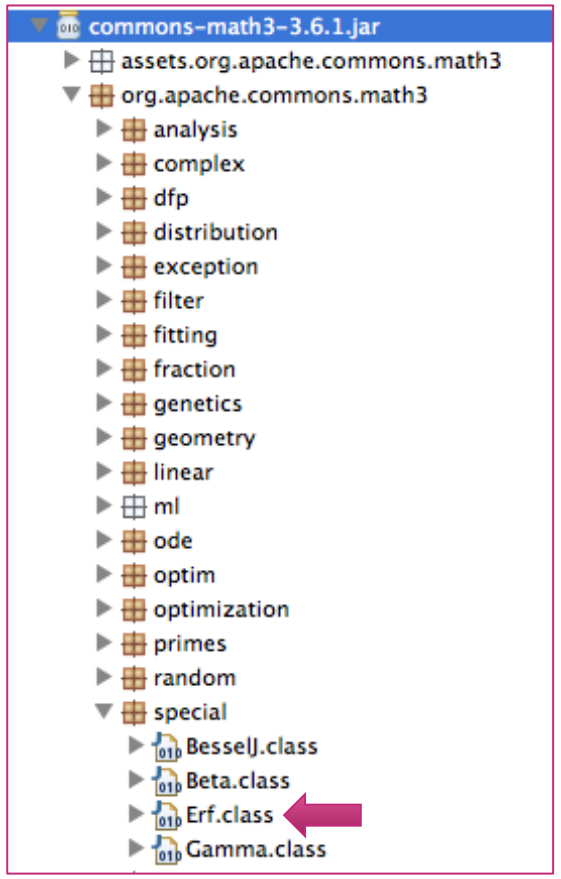
Using built-in function interface

```
public Function<Double, Double> erfDeriv() {
      return x -> 2.0/sqrt(PI)*exp(-x*x) ;
}


public Function<Double, Double> erfIntegral() {
      return x -> x*erf(x)+exp(-x*x)/sqrt(PI) ;
}
```

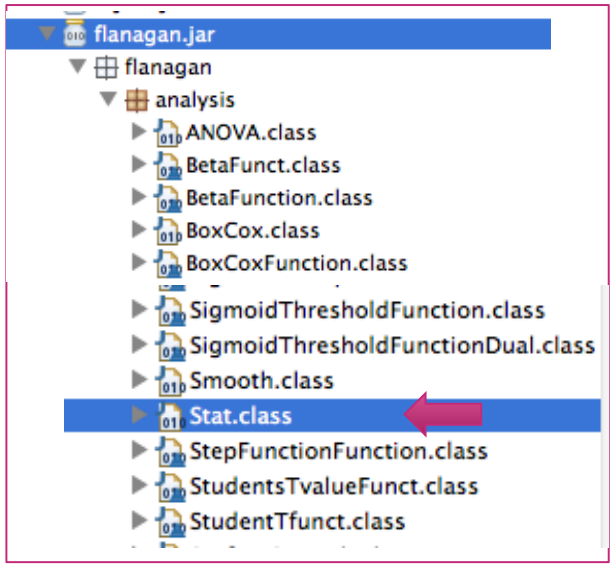
Other Error Function Libraries

- Apache commons math (org.apache.commons.math3.special.Erf)
- Flanagan library (flanagan.analysis.Stat.erf)

Apache Commons Math



Flanagan library



Problems:

- Slower implementation
- No support for complex numbers

Speed comparison

```
public static void main(String[] args) {
      // from my library --> 44 msec
      test1() ;
      // from Flanagan library --> 78 msec
      test2() ;
      // from apache commons math --> 79 msec
      test3() ;
}
```

Handling Large Arrays

- Assume we want to evaluate an array of some numbers
  - 1,000 elements
  - 1,000,000 elements
  - 100,000,000 elements

One thread

Sequential calculation ⟶ Iterate and calculate one by one

vs.

Parallel calculation ⟶ Divide in pieces and calculate in parallel

Multi-thread

spliterator

- A simple "**for**" loop is only good for small-sized arrays
  - Typically, **size < 10,000**
- For larger sizes of arrays, use built-in **Stream API** in java 8
  - **Arrays.stream(x)** → returns a sequential stream
  - **Arrays.stream(x).parallel()** → returns a parallel stream
- Significant difference when handling double arrays of size > 1,000,000
- Example
  - Calculate erf(x) for 1,000,000

Handling Large Arrays

- Significant difference when handling arrays of size > 1,000,000
- **Example**: double numbers
  - Calculate *erf(x)* for 1,000,000
  - Time: 990 msec vs. 560 msec
- How to use Stream API with arrays
  - .map(functional interface)
    - Intermediate operation
  - toArray() → closes the stream
    - Terminal operation
- **Example**: Complex numbers
  - toArray() collects as objects
  - Time: 33 sec vs. 6.8 sec

Example of stream API for large arrays

```
public static void test12() {
      ErrorFunction func = new ErrorFunction() ;
      double[] x = MathUtils.linspace(-5.0, 5.0, 1000000) ;
      Timer timer = new Timer() ;
      timer.start();
      double[] erf = ArrayFunc.apply(t -> func.erf(t), x) ;
      double[] erf = Arrays.stream(x).map(t -> func.erf(t)).toArray() ;
      double[] erf = Arrays.stream(x).parallel().map(t -> func.erf(t)).toArray() ;
      timer.stop();
      timer.show();
}
```

```
public static void test13() {
      ErrorFunction func = new ErrorFunction() ;
      double[] x = MathUtils.linspace(-5.0, 5.0, 1000000) ;
      List<Complex> xc = new ArrayList<>(x.length) ;
      for(int i=0, len=x.length; i<len; i++)
            xc.add(x[i]+1.0*j) ;

      Timer timer = new Timer() ;
      timer.start();
//    List<Complex> erf = ArrayFunc.apply(t -> func.erf(t), xc) ;
      Object[] erf = xc.stream().map(t -> func.erf(t)).toArray() ;
      Object[] erf = xc.parallelStream().map(t -> func.erf(t)).toArray() ;
      timer.stop();
      timer.show();
}
```

Do not use parallel stream for small arrays

Spherical Bessel Functions

- Solutions of wave equation in the spherical coordinate system: $j_n(x)$, $y_n(x)$
- Differential equation:

$$x^2\,y'' + 2x\,y' + (x^2 - n(n+1))\,y = 0$$

- Relation to Bessel functions

$$j_n(x) = \sqrt{\frac{0.5\pi}{x}}\,J_{n+0.5}(x)$$

$$y_n(x) = (-1)^{n+1}\sqrt{\frac{0.5\pi}{x}}\,J_{-n-0.5}(x)$$

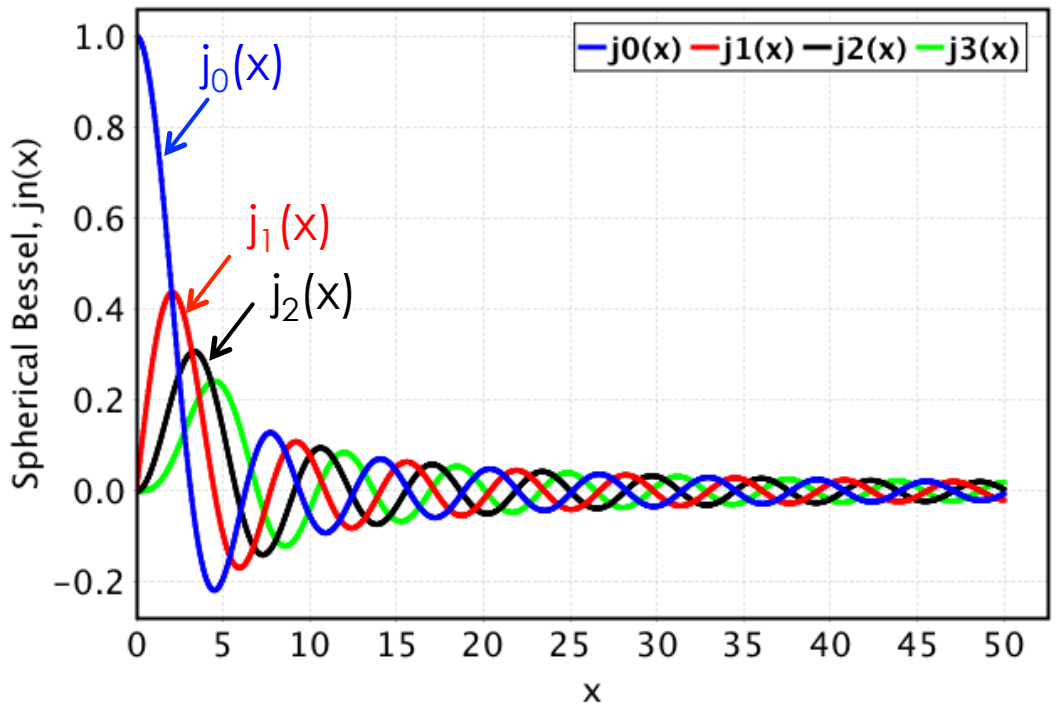
$J_{-v}(x)$ and $J_v(x)$ are independent

n can only be an integer (n>=0)

$$\begin{cases} h_n^{(1)}(x) = j_n(x) + j\,y_n(x) \\ h_n^{(2)}(x) = j_n(x) - j\,y_n(x) \end{cases}$$

Spherical Hankel Functions

We take advantage of our efficient implementation of Bessel functions

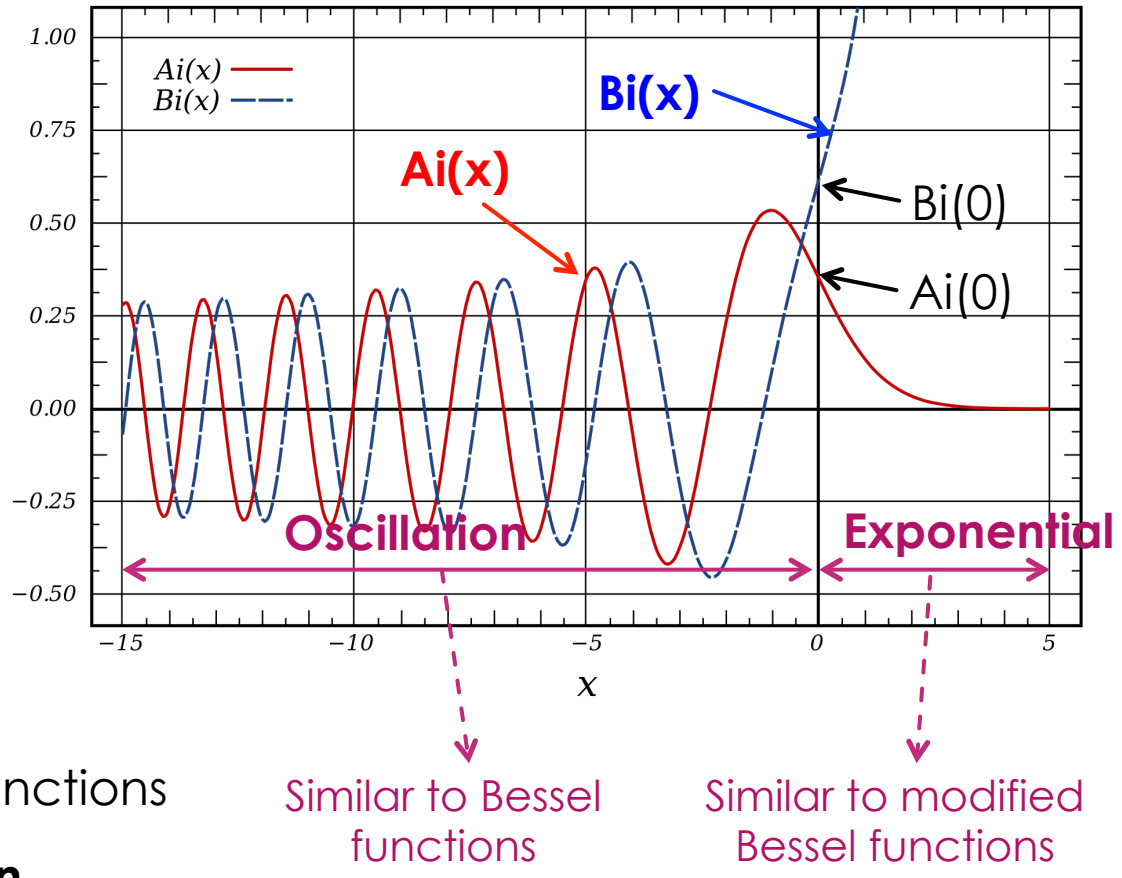


Airy Functions

- Solutions to the Airy differential equation: **Ai(x)**, **Bi(x)**

$$y'' - xy = 0$$

- **Exact** values at x = 0 :
  - $Ai(0) = 1/(3^{2/3}\,\Gamma(2/3)) = 0.3550280538$
  - $Ai'(0) = -1/(3^{1/3}\,\Gamma(1/3)) = -0.2588194037$
  - $Bi(0) = 1/(3^{1/6}\,\Gamma(2/3)) = 0.6149266274$
  - $Bi'(0) = 3^{1/6}/\Gamma(1/3) = 0.4482883573$
- How to numerically calculate Airy functions?
  - Direct approach: solve differential equation
    - Need initial conditions at x=0: y(0) & y'(0)
  - Indirect approach: use its connection to other functions
    - For **x>0** → relation to **modified Bessel function**
    - For **x<0** → relation to **Bessel functions**



Airy Functions

- Solutions to the Airy differential equation: Ai(x), Bi(x)

$$y'' - xy = 0$$

$$\begin{cases} y' = u \\ u' = xy \end{cases}$$

Converting to a system of ODEs

State variable z = (y, u)

- How to numerically calculate Airy functions?
  - Direct approach: solve differential equation
    - Need initial conditions at x=0: y(0) & y'(0)
- Use the efficient ODE solver that was developed
- Second-order ODE → Use system solver
  - Convert the problem into a system of ODEs
- Use 4th-order Runge-Kutta for high accuracy
- Ai(x) drops rapidly for x>0
- Set the threshold to x = 6
  - Ai(x) = 0 for x>6
- **Benefit:** both Ai(x) & Ai'(x) are found together.

Constructor

```
public AiryFunctionIntegration() {
      func = (t, z) -> new double[]{z[1], t*z[0]} ;
      odeSolverAi = new OdeSystemSolver(func, x0, ai0) ;
      odeSolverBi = new OdeSystemSolver(func, x0, bi0) ;
}

public double ai(double x) {
      if(x > 6.19635)
            return 0.0 ;
      else
            return
odeSolverAi.rungeKuttaSequence(x).evaluate(1000).array()[0] ;
}

public double aiDeriv(double x) {
      if(x > 6.19635)
            return 0.0 ;
      else
            return
odeSolverAi.rungeKuttaSequence(x).evaluate(1000).array()[1] ;
}
```

Smaller number can be used

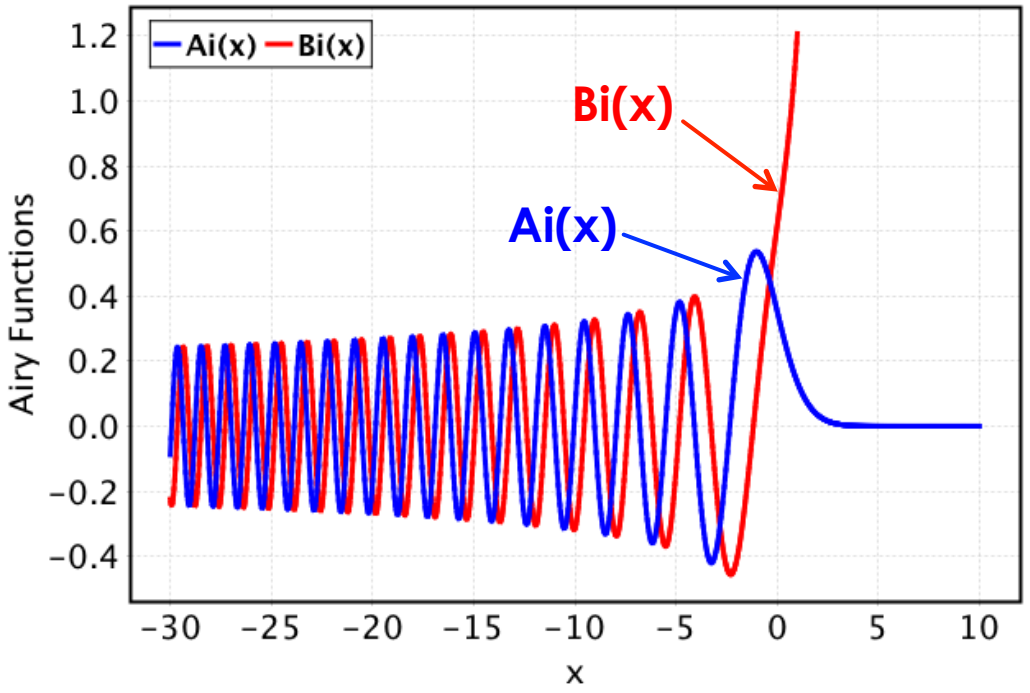
Airy Functions

- Solutions to the Airy differential equation: Ai(x), Bi(x)

$$y'' - xy = 0 \quad \Rightarrow \quad \begin{cases} y' = u \\ u' = xy \end{cases}$$

Converting to a system of ODEs

- How to numerically calculate Airy functions?
  - Direct approach: solve differential equation ⟶ Not efficient
    - Need initial conditions at x=0: y(0) & y'(0)




```
public static void test1() {
      AiryFunctionIntegration airy =
                              new AiryFunctionIntegration() ;

      Timer timer = new Timer() ;
      timer.start();
      double[] xa = MathUtils.linspace(-30.0, 10.0, 1000) ;
      double[] ai = ArrayFunc.apply(t -> airy.ai(t), xa) ;
      double[] xb = MathUtils.linspace(-30.0, 1.0, 1000) ;
      double[] bi = ArrayFunc.apply(t -> airy.bi(t), xb) ;
      timer.stop();
      timer.show();

      MatlabChart fig = new MatlabChart() ;
      fig.plot(xa, ai, "b", 2f, "Ai(x)");
      fig.plot(xb, bi, "r", 2f, "Bi(x)");
      fig.renderPlot();
      fig.legendON();
      fig.xlabel("x");
      fig.ylabel("Airy Functions");
      fig.show(true);
}
```

Airy Functions

- Solutions to the Airy differential equation: Ai(x), Bi(x)

$$y'' - xy = 0$$

- How to numerically calculate Airy functions?
  - Indirect approach: use its connection to other functions ⟶ Very efficient
- For **x>0** → relation to **modified Bessel function**

$$\zeta = \frac{2}{3}x^{3/2} \Rightarrow \mathrm{Ai}(x) = \frac{1}{\pi}\sqrt{x/3}\,K_{1/3}(\zeta)$$

$$\zeta = \frac{2}{3}x^{3/2} \Rightarrow \mathrm{Ai}'(x) = -\frac{1}{\pi}\frac{x}{\sqrt{3}}\,K_{2/3}(\zeta)$$

- For **x<0** → relation to **Bessel functions**

$$\zeta = \frac{2}{3}(-x)^{3/2} \Rightarrow \mathrm{Ai}(x) = \frac{\sqrt{-x}}{3}\left(J_{1/3}(\zeta) + J_{-1/3}(\zeta)\right)$$

$$\zeta = \frac{2}{3}(-x)^{3/2} \Rightarrow \mathrm{Ai}'(x) = \frac{-x}{3}\left(J_{2/3}(\zeta) - J_{-2/3}(\zeta)\right)$$

```
private double aiNegativeDouble(double x) {
      x = -x ;
      double zeta = 2.0/3.0 * pow(x, 1.5) ;
      return sqrt(x)/3.0 * (bessel.jv(1.0/3.0, zeta)
+bessel.jv(-1.0/3.0, zeta)) ;
}

private double aiPositiveDouble(double x) {
      double zeta = 2.0/3.0 * pow(x, 1.5) ;
      return sqrt(x/3.0)/PI * modifiedBessel.kv(1.0/3.0, zeta) ;
}

public double ai(double x) {
      if(x>0.0) {
            return aiPositiveDouble(x) ;
      }
      else if(x<0.0)
            return aiNegativeDouble(x) ;
      else
            return ai0[0] ;
}
```

Airy Functions

- Solutions to the Airy differential equation: Ai(x), Bi(x)

$$y'' - xy = 0$$

- How to numerically calculate Airy functions?
  - Indirect approach: use its connection to other functions ⟶ Very efficient
- For **x>0** → relation to **modified Bessel function**

$$\zeta = \frac{2}{3}x^{3/2} \Rightarrow \mathrm{Bi}(x) = \sqrt{\frac{x}{3}}\left(I_{1/3}(\zeta) + I_{-1/3}(\zeta)\right)$$

$$\zeta = \frac{2}{3}x^{3/2} \Rightarrow \mathrm{Bi}'(x) = \frac{x}{\sqrt{3}}\left(I_{2/3}(\zeta) + I_{-2/3}(\zeta)\right)$$

- For **x<0** → relation to **Bessel functions**

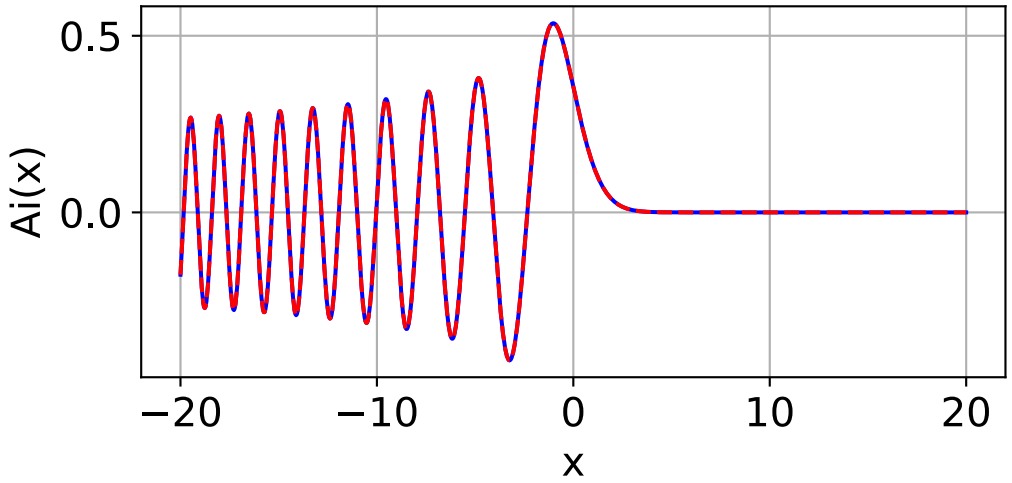
$$\zeta = \frac{2}{3}(-x)^{3/2} \Rightarrow \mathrm{Bi}(x) = \sqrt{\frac{x}{3}}\left(J_{-1/3}(\zeta) - J_{1/3}(\zeta)\right)$$

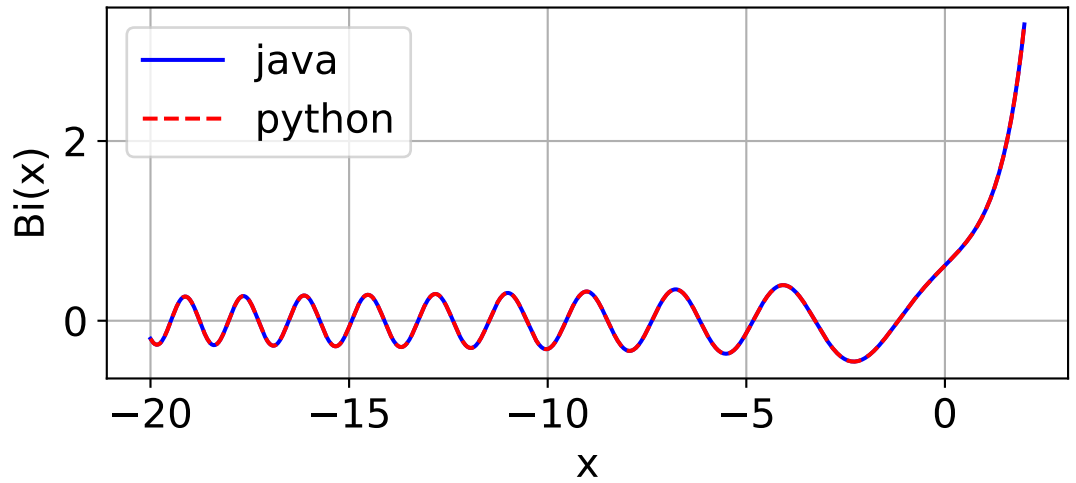
$$\zeta = \frac{2}{3}(-x)^{3/2} \Rightarrow \mathrm{Bi}'(x) = \frac{x}{\sqrt{3}}\left(J_{-2/3}(\zeta) + J_{2/3}(\zeta)\right)$$

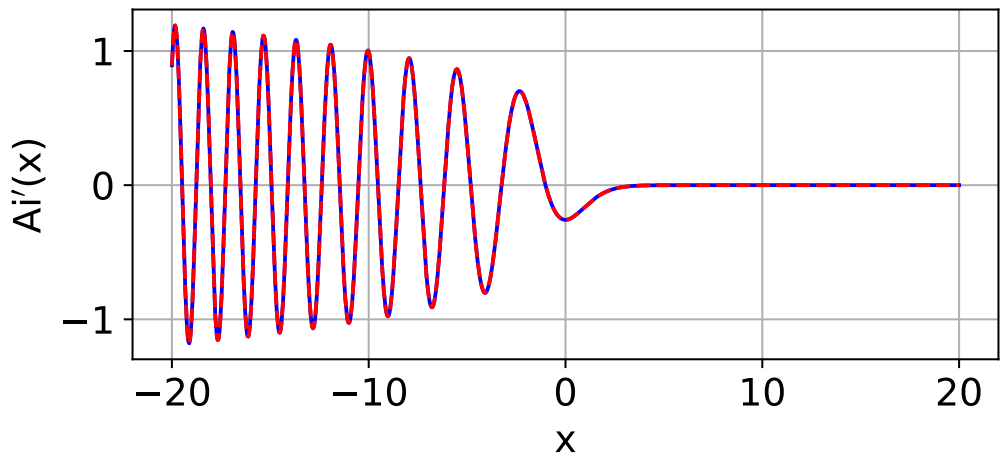
```
private double biPositiveDouble(double x) {
      double zeta = 2.0/3.0 * pow(x, 1.5) ;
      return sqrt(x/3.0) * (modifiedBessel.iv(1.0/3.0, zeta) +
modifiedBessel.iv(-1.0/3.0, zeta)) ;
}

private double biNegativeDouble(double x) {
      x = -x ;
      double zeta = 2.0/3.0 * pow(x, 1.5) ;
      return sqrt(x/3.0) * (bessel.jv(-1.0/3.0, zeta) -
bessel.jv(1.0/3.0, zeta)) ;
}

public double bi(double x) {
      if(x>0.0) {
            return biPositiveDouble(x) ;
      }
      else if(x<0.0)
            return biNegativeDouble(x) ;
      else
            return bi0[0] ;
}
```

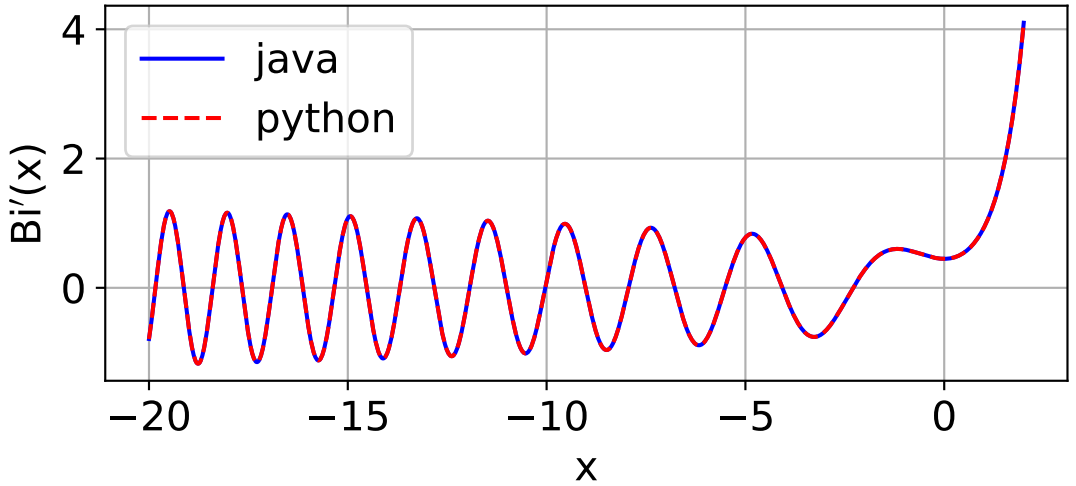
Airy Functions

- Comparison with **scipy.special** package









Struve Functions

- Solutions to the Struve differential equation: $\mathbf{H_\nu(x)}$, $\mathbf{K_\nu(x)}$ = $H_\nu(x)$-$Y_\nu(x)$

$$x^2y'' + xy' + (x^2 - \nu^2)y = \frac{4(x/2)^{\nu+1}}{\sqrt{\pi}\,\Gamma(\nu + 0.5)} \Rightarrow H_\nu(x) = \left(\frac{x}{2}\right)^{\nu+1} \sum_{k=0}^{\infty} \frac{(-1)^k\,(x/2)^{2k}}{\Gamma(k + 1.5)\,\Gamma(\nu + k + 1.5)}$$

integer

- How to calculate the series?
- Factor out $\Gamma(1.5)$ and $\Gamma(v+1.5)$

$$\Gamma(1.5) = \sqrt{\pi}/2 \qquad \Gamma(0.5) = \sqrt{\pi}$$

- For large argument
  - $H_\nu(x) = K_\nu(x) + Y_\nu(x)$
- Asymptotic behavior of $\mathbf{K_\nu(z)}$

$$\mathbf{K}_\nu(z) \sim \frac{1}{\pi} \sum_{k=0}^{\infty} \frac{\Gamma\left(k + \frac{1}{2}\right)\left(\frac{1}{2}z\right)^{\nu-2k-1}}{\Gamma\left(\nu + \frac{1}{2} - k\right)}, \quad |\operatorname{ph} z| \le \pi - \delta,$$

Factor out $\Gamma(0.5)$ and $\Gamma(v+0.5)$

Excluding x < 0 of real numbers

$\Gamma(1.5) = \sqrt{\pi}/2$

```
private double hvSmallArgument(double v, double x) {
     if(abs(v-lastv)<1e-2) {
          leadTerm = pow(x/2.0, v+1) / (coeff*lastGammaH) ;
     }
     else {
          lastv = v ;
          lastGammaH = gammaFunc.gamma(v+1.5) ;
          leadTerm = pow(x/2.0, v+1) / (coeff*lastGammaH) ;
     }
     val = (0.5*x)*(0.5*x) ;
     hvSeq = n -> {
          double result = 1.0 ;
          for(int i=(int)n; i>0; i--) {
               result = 1.0 - val/((i+0.5)*(i+0.5+v))*result ;
          }
          return result ;
     } ;
     return hvSeq.evaluate(200)*leadTerm ;
}
```

Struve Functions

- Solutions to the Struve differential equation: **$H_\nu(x)$**, **$K_\nu(x)$** = $H_\nu(x)$-$Y_\nu(x)$

$$x^2y'' + xy' + (x^2 - \nu^2)y = \frac{4(x/2)^{\nu+1}}{\sqrt{\pi}\,\Gamma(\nu + 0.5)} \Rightarrow H_\nu(x) = \left(\frac{x}{2}\right)^{\nu+1} \sum_{k=0}^{\infty} \frac{(-1)^k\,(x/2)^{2k}}{\Gamma(k + 1.5)\,\Gamma(\nu + k + 1.5)}$$

- What do I mean by factorizing gamma function?
  - Remember: **$\Gamma(z) = (z-1)\,\Gamma(z-1)$** with **$\Gamma(1) = 1$**
- Therefore:

$$\Gamma(k + 1.5) = (k - 1 + 1.5)\Gamma(k - 1 + 1.5) = (k - 1 + 1.5)(k - 2 + 1.5)\Gamma(k - 2 + 1.5) = \cdots = (k - 1 + 1.5)(k - 2 + 1.5)\cdots(1.5)\Gamma(1.5)$$

integer

- In general,

$$\Gamma(k + A) = (k - 1 + A)\Gamma(k - 1 + A) = (k - 1 + A)(k - 2 + A)\Gamma(k - 2 + A) = \cdots = (k - 1 + A)(k - 2 + A)\cdots(A)\Gamma(A)$$

integer Any number

Struve Functions

- Solutions to the Struve differential equation: **$H_\nu(x)$**, **$K_\nu(x)$** = $H_\nu(x)$-$Y_\nu(x)$

$$x^2y'' + xy' + (x^2 - \nu^2)y = \frac{4(x/2)^{\nu+1}}{\sqrt{\pi}\,\Gamma(\nu + 0.5)} \quad \Rightarrow \quad H_\nu(x) = \left(\frac{x}{2}\right)^{\nu+1} \sum_{k=0}^{\infty} \frac{(-1)^k\,(x/2)^{2k}}{\Gamma(k + 1.5)\,\Gamma(\nu + k + 1.5)}$$

- Comparison with **scipy.special**