Powersum User Manual

Mariappan Asokan

June 15, 2025

1 Introduction

This package contains an efficient arbitrary precision implementation of all known power sum formulas namely Faulhaber, Bernoulli, Stirling, Euler, and Central Factorial. These formulas are used to compute

$$S_m(n) = \sum_{i=1}^{n} i^m = 1^m + 2^m + \ldots + n^m$$

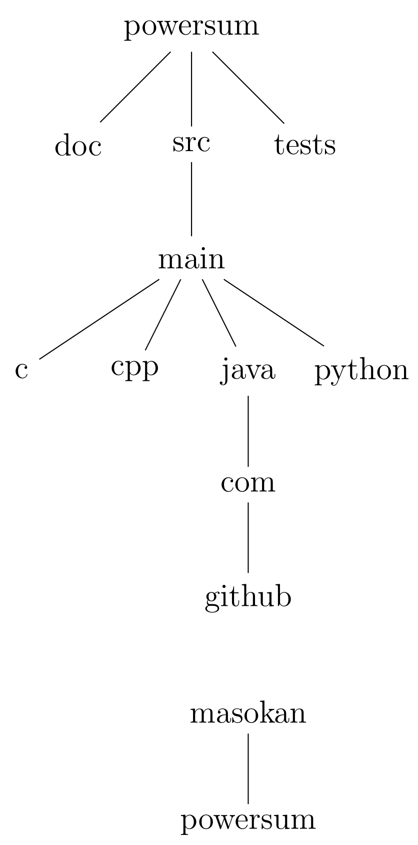
The implementation provides callable APIs in C, C++, Java, and Python. They are thread-safe and can be embedded in a larger program easily. Currently, the implementation was built and tested under Linux. The Java implementation should work under Windows as well but it has not been tested. If you have Cygwin under Windows, the other language implementations should work as well. Again they have not been tested.

2 Installation

First clone the source archive from powersum by typing the following command in UNIX shell:

```
$ git clone https://github.com/masokan/powersum.git
```

This assumes that you have `git` installed on your Linux system. It should create a directory structure as shown below:



2.1 C and C++

2.1.1 Requirements

The C code conforms to C99 (ISO/IEC 9899:1999) standard. So at the minimum `gcc` version 4.5 is required. The C++ code conforms to C++11 (ISO/IEC 14882:2011) standard. So at the minimum `g++` version 4.8.1 is required. You need to install GNU multiple precision arithmetic library. Prebuilt binary packages are available for Linux. Consult your Linux distribution documentation on how to install them. Specifically, you need the `libgmp` development packages for C and C++.

2.1.2 Building the binaries

To build the C binaries, type the following commands in UNIX shell:

```
$ cd powersum/src/main/c
$ make clean
$ make
```

This will create an archive library file `libpowersum.a` and an executable file `powersum` in the C source directory. The archive library can be linked with any larger program that uses the C APIs. The executable file can be run as a UNIX command.
To build the C++ binaries, type the following commands in UNIX shell:

```
$ cd powersum/src/main/cpp
$ make clean
$ make
```

This will create an archive library file `libpowersum.a` and an executable file `PowerSum` in the C++ source directory. The archive library can be linked with any larger program that uses the C++ APIs. The executable file can be run as a UNIX command.

2.2 Java

2.2.1 Requirements

The Java implementation should work on any JDK version.

2.2.2 Building the `jar` file

To build the `jar` file, type the following commands in UNIX shell:

```
$ cd powersum/src/main/java
$ ./build.sh
```

This should create `PowerSum.jar` file in the same directory.

2.3 Python

2.3.1 Requirements

Python 3.x should be available as `python3` command.
The file `powersum/src/main/python/requirements.txt` contains the required Python modules. The best way to run the Python implementation is to create a virtual environment and install the required Python modules. You can do this by typing the following commands in UNIX shell:

```
$ python3 -m venv powersumenv
$ . ./powersumenv/bin/activate
$ pip install -r powersum/src/main/python/requirements.txt
```

The above will create a directory `powersumenv` under the current directory (where the source archive was cloned) and install the required modules.

3 Command line invocation

It is possible to run command lines for different languages to check the functionalities available. We will show the user interaction when using the C implementation. The command options are identical for all languages. Only the command invocation is different.
To get help on usage:

```
$ ./powersum
Usage: ./powersum (-c|-f|-h|-s|-sv) [<power>] [<numTerms>]

<power> and <numTerms> should be greater than or equal to 0

Examples:
To print the help on usage:
./powersum -h or just ./powersum

To print coefficients in the formula for power 10:
./powersum -c 10

To print formula for sum for power 5:
./powersum -f 5

To print the sum of series for power 6 for the first 20 terms:
The sum will be computed using the formula
./powersum -s 6 20

To compute the sum in two ways one using the formula and the
other with actual series expansion and verify the results for correctness
./powersum -sv 6 20

If <numTerms> is missing, a default of 20 is assumed
```

To print the coefficients in different formulas for power 10:

```
$ ./powersum -c 10
Computing coefficients for power 10
Faulhaber: -------------------------------------
 1/11 -10/33 17/33 -5/11 5/33
Time taken = 97046
Bernoulli: -------------------------------------
 1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66
Time taken = 59496
Stirling: --------------------------------------
 0 1 511 9330 34105 42525 22827 5880 750 45 1
Time taken = 23634
Euler: -----------------------------------------
 1 1013 47840 455192 1310354 1310354 455192 47840 1013 1 0
Time taken = 24347
Central Factorial: -----------------------------
 0 1 85 147 30 1
Time taken = 12841
```

To print sum formulas for power 4:

```
$ ./powersum -f 4
Faulhaber: -------------------------------------
(2n + 1){(1/5)N^2 + (-1/15)N}/2
where N = n(n + 1)
Bernoulli: -------------------------------------
{ (n + 1)^5 + (-1/2)5(n + 1)^4 + (1/6)10(n + 1)^3 + (-1/30)5(n + 1) }/5
Stirling: --------------------------------------
   (n + 1)n/2 + 7(n + 1)n(n - 1)/3 + 6(n + 1)n(n - 1)(n - 2)/4 + (n + 1)n(n - 1)(n - 2)(n - 3)/5
Euler: -----------------------------------------
{ (n + 1)n(n - 1)(n - 2)(n - 3) + 11(n + 2)(n + 1)n(n - 1)(n - 2) + 11(n + 3)(n + 2)(n + 1)n(n - 1)
 + (n + 4)(n + 3)(n + 2)(n + 1)n + (n + 5)(n + 4)(n + 3)(n + 2)(n + 1) }/120
Central Factorial: -----------------------------
(2n + 1)(n + 1)n/6 + (2n + 1)(n + 2)(n + 1)n(n - 1)/10
```

To compute sum for 21^{st} power for 100 terms using the formulas:

```
$ ./powersum -s 21 100
Computing S(21, 100)
Faulhaber: -------------------------------------
Sum computed = 5062943470191369347210738338827565614896500
Time taken = 65520:46989:18531
Bernoulli: -------------------------------------
Sum computed = 5062943470191369347210738338827565614896500
Time taken = 60069:45994:14075
Stirling: --------------------------------------
Sum computed = 5062943470191369347210738338827565614896500
Time taken = 15135:10664:4471
Euler: -----------------------------------------
Sum computed = 5062943470191369347210738338827565614896500
Time taken = 17568:12749:4819
Central Factorial: -----------------------------
Sum computed = 5062943470191369347210738338827565614896500
Time taken = 11376:7572:3804
```

The time taken is a triplet of numbers with the first one being total execution time, the second one the time taken to initialize the coefficients in the formulas, and the third one the time taken to compute the sum once the coefficients are known. The unit of time is nanoseconds.

To compute sum for 16^{th} power for 10000 terms using the formulas

and verify for correctness:

```
$ ./powersum -sv 16 10000
Computing S(16, 10000)
Faulhaber: -------------------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 64763:53635:11128
Bernoulli: -------------------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 47793:34061:13732
Stirling: --------------------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 12043:7478:4565
Euler: -----------------------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 13683:9049:4634
Central Factorial: -----------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 9208:5488:3720
Series addition:--------------------------------
Sum computed = 58873542745097572549036941175993921577294117554925490662745097330000
Time taken = 1057122
The sum matches with Faulhaber formula :-)
The sum matches with Bernoulli formula :-)
The sum matches with Stirling formula :-)
The sum matches with Euler formula :-)
The sum matches with Central Factorial formula :-)
```

Series addition time is the time taken for the simple implementation.
In the following, we show how the command can be invoked from other languages: For C++:

```
$ ./PowerSum
```

For Java:

```
$ java -jar PowerSum.jar
```

For Python:

```
$ python3 ./power_sum_main.py
```

Here we have assumed that you are attached to the source directory for respective languages before running the commands. For Python, it is assumed that the virtual environment created before is activated.

4 API usage

For the same functionalities, we will show source code examples for different languages and how to run the examples. The code invokes the APIs to get the coefficients for some formulas, print sum formula for Central Factorial, and compute and verify sum calculated using different formulas.

4.1 C

Create a file named `CExample.c` in your favorite text editor and paste the code below:

```c
#include <stdio.h>
#include <stdlib.h>
#include "bernoulli_power_sum.h"
#include "central_factorial_power_sum.h"
#include "euler_power_sum.h"
#include "faulhaber_power_sum.h"
#include "power_sum.h"
#include "stirling_power_sum.h"
static void printCoefficients(PowerSum * psPtr, long power) {
  mpq_t * coeffs;
  long numCoeffs;
  psPtr->getCoefficients(power, &coeffs, &numCoeffs);
  for (long t = 0; t < numCoeffs; t++) {
    gmp_printf(" %Qd", coeffs[t]);
  }
  printf("\n");
  psPtr->freeCoefficients(coeffs, numCoeffs);
}
static void computeAndPrintSum(PowerSum * psPtr, long power, long numTerms,
                               mpz_t sum) {
  psPtr->computeSum(power, numTerms, sum);
  gmp_printf("Sum computed = %Zd\n", sum);
}
int main(int argc, char ** argv) {
  /* Initialize function pointers */
  PowerSum fps, bps, sps, eps, cps;
  initFaulhaberPowerSum(&fps);
  initBernoulliPowerSum(&bps);
  initStirlingPowerSum(&sps);
  initEulerPowerSum(&eps);
  initCentralFactorialPowerSum(&cps);
  long power, numTerms;
  power = 11;
  printf("Printing Bernoulli coefficients for power %ld\n", power);
  printCoefficients(&bps, power);
  power = 5;
  printf("Printing Central Factorial formula for power %ld\n", power);
  cps.printSumFormula(power, stdout);
  mpz_t sumFaulhaber, sumBernoulli, sumStirling, sumEuler, sumCentral, sumSeries;
  mpz_init(sumFaulhaber);
  mpz_init(sumBernoulli);
  mpz_init(sumStirling);
  mpz_init(sumEuler);
  mpz_init(sumCentral);
  mpz_init(sumSeries);
  power = 19;
  numTerms = 1000;
  printf("Computing S(%ld, %ld) using Faulhaber formula\n", power, numTerms);
  computeAndPrintSum(&fps, power, numTerms, sumFaulhaber);
  printf("Computing S(%ld, %ld) using Bernoulli formula\n", power, numTerms);
  computeAndPrintSum(&bps, power, numTerms, sumBernoulli);
  printf("Computing S(%ld, %ld) using Stirling formula\n", power, numTerms);
  computeAndPrintSum(&sps, power, numTerms, sumStirling);
  printf("Computing S(%ld, %ld) using Euler formula\n", power, numTerms);
  computeAndPrintSum(&eps, power, numTerms, sumEuler);
  printf("Computing S(%ld, %ld) using Central Factorial formula\n", power,
         numTerms);
  computeAndPrintSum(&cps, power, numTerms, sumCentral);
  printf("Computing S(%ld, %ld) using series addition\n", power, numTerms);
  sps.computeSumUsingSeries(power, numTerms, sumSeries);
  gmp_printf("Sum computed = %Zd\n", sumSeries);
  if (mpz_cmp(sumFaulhaber, sumSeries) != 0
      || mpz_cmp(sumBernoulli, sumSeries) != 0
      || mpz_cmp(sumStirling, sumSeries) != 0
      || mpz_cmp(sumEuler, sumSeries) != 0
      || mpz_cmp(sumCentral, sumSeries) != 0) {
    printf("Sums do not match!\n");
  }
  mpz_clear(sumSeries);
  mpz_clear(sumCentral);
  mpz_clear(sumEuler);
  mpz_clear(sumStirling);
  mpz_clear(sumBernoulli);
  mpz_clear(sumFaulhaber);
  return EXIT_SUCCESS;
}
```

4.1.1 Running the example

Following commands show how to compile, link, and run the C example. The output of the program is also shown:

```
$ gcc -Wall -O3 -I./powersum/src/main/c -o CExample CExample.c \
      ./powersum/src/main/c/libpowersum.a -lgmp
$ ./CExample
Printing Bernoulli coefficients for power 11
 1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66 0
Printing Central Factorial formula for power 5
(n + 1)n/2 + 5(n + 2)(n + 1)n(n - 1)/4 + (n + 3)(n + 2)(n + 1)n(n - 1)(n - 2)/6
Computing S(19, 1000) using Faulhaber formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Bernoulli formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Stirling formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Euler formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Central Factorial formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using series addition
Sum computed = 50501583325258379475980526890307929576070522091449331750000
```

4.2 C++

Create a file named `CppExample.cc` in your favorite text editor and paste the code below:

```
#include <stdlib.h>

#include <iomanip>
#include <iostream>

#include "BernoulliPowerSum.h"
#include "CentralFactorialPowerSum.h"
#include "EulerPowerSum.h"
#include "FaulhaberPowerSum.h"
#include "PowerSum.h"
#include "StirlingPowerSum.h"

using std::cout;
using std::endl;

static void printCoefficients(PowerSum & ps, long power) {
  vector<mpq_class> coeffs = ps.getCoefficients(power);
  for (size_t t = 0; t < coeffs.size(); t++) {
    cout << " " << coeffs[t];
  }
  cout << endl;
}
static mpz_class computeAndPrintSum(PowerSum & ps, long power, long numTerms) {
  mpz_class sum = ps.computeSum(power, numTerms);
  cout << "Sum computed = " << sum << endl;
  return sum;
}
int main(int argc, char ** argv) {
  FaulhaberPowerSum fps;
  BernoulliPowerSum bps;
  StirlingPowerSum sps;
  EulerPowerSum eps;
  CentralFactorialPowerSum cps;
  mpz_class sumFaulhaber, sumBernoulli, sumStirling, sumEuler, sumCentral, sumSeries;

  long power, numTerms;
  power = 11;
  cout << "Printing Bernoulli coefficients for power " <<  power << endl;
  printCoefficients(bps, power);
  power = 5;
  cout << "Printing Central Factorial formula for power " <<  power << endl;
  cps.printSumFormula(power, cout);

  power = 19;
  numTerms = 1000;

  cout << "Computing S(" << power << ", " << numTerms
       << ") using Faulhaber formula" << endl;
  sumFaulhaber = computeAndPrintSum(fps, power, numTerms);
  cout << "Computing S(" << power << ", " << numTerms
       << ") using Bernoulli formula" << endl;
  sumBernoulli = computeAndPrintSum(bps, power, numTerms);
  cout << "Computing S(" << power << ", " << numTerms
       << ") using Stirling formula" << endl;
  sumStirling = computeAndPrintSum(sps, power, numTerms);
  cout << "Computing S(" << power << ", " << numTerms
       << ") using Euler formula" << endl;
  sumEuler = computeAndPrintSum(eps, power, numTerms);
  cout << "Computing S(" << power << ", " << numTerms
       << ") using Central Factorial formula" << endl;
  sumCentral = computeAndPrintSum(cps, power, numTerms);
  cout << "Computing S(" << power << ", " << numTerms
       << ") using series addition" << endl;
  sumSeries = sps.computeSumUsingSeries(power, numTerms);
  cout << "Sum computed = " << sumSeries << endl;
  if (sumFaulhaber != sumSeries || sumBernoulli != sumSeries
      || sumStirling != sumSeries || sumEuler != sumSeries
      || sumCentral != sumSeries) {
    cout << "Sums do not match!" << endl;
  }
  return EXIT_SUCCESS;
}
```

4.2.1 Running the example

Following commands show how to compile, link, and run the C++ example. The output of the program is also shown:

```
$ g++ -Wall -O3 -I./powersum/src/main/cpp -o CppExample CppExample.cc \
      ./powersum/src/main/cpp/libpowersum.a -lgmp -lgmpxx
$ ./CppExample
Printing Bernoulli coefficients for power 11
 1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66 0
Printing Central Factorial formula for power 5
(n + 1)n/2 + 5(n + 2)(n + 1)n(n - 1)/4 + (n + 3)(n + 2)(n + 1)n(n - 1)(n - 2)/6
Computing S(19, 1000) using Faulhaber formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Bernoulli formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Stirling formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Euler formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Central Factorial formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using series addition
Sum computed = 50501583325258379475980526890307929576070522091449331750000
```

4.3 Java

Create a file named `JavaExample.java` in your favorite text editor and paste the code below:

```
import com.github.masokan.powersum.*;

import java.math.BigInteger;
import java.util.List;

public class JavaExample {
  private static void printCoefficients(PowerSum ps, long power) {
    List<RationalNumber> coeffs = ps.getCoefficients(power);
    for (int t = 0; t < coeffs.size(); t++) {
      System.out.print(" " + coeffs.get(t));
    }
    System.out.println();
  }

  private static BigInteger computeAndPrintSum(PowerSum ps, long power,
                                               long numTerms) {
    BigInteger sum = ps.computeSum(power, numTerms);
    System.out.println("Sum computed = " + sum);
    return sum;
  }

  public static void main(String[] args) {
    PowerSum fps = new FaulhaberPowerSum();
    PowerSum bps = new BernoulliPowerSum();
    PowerSum sps = new StirlingPowerSum();
    PowerSum eps = new EulerPowerSum();
    PowerSum cps = new CentralFactorialPowerSum();
    BigInteger sumFaulhaber, sumBernoulli, sumStirling, sumEuler, sumCentral, sumSeries;
    long power, numTerms;
    power = 11;
    System.out.println("Printing Bernoulli coefficients for power " + power);
    printCoefficients(bps, power);
    power = 5;
    System.out.println("Printing Central Factorial formula for power " + power);
    cps.printSumFormula(power, System.out);
    power = 19;
    numTerms = 1000;
    System.out.printf("Computing S(%d, %d) using Faulhaber formula", power,
                      numTerms);
    System.out.println();
    sumFaulhaber = computeAndPrintSum(fps, power, numTerms);
    System.out.printf("Computing S(%d, %d) using Bernoulli formula", power,
                      numTerms);
    System.out.println();
    sumBernoulli = computeAndPrintSum(bps, power, numTerms);
    System.out.printf("Computing S(%d, %d) using Stirling formula", power,
                      numTerms);
    System.out.println();
    sumStirling = computeAndPrintSum(sps, power, numTerms);
    System.out.printf("Computing S(%d, %d) using Euler formula", power,
                      numTerms);
    System.out.println();
    sumEuler = computeAndPrintSum(eps, power, numTerms);
    System.out.printf("Computing S(%d, %d) using Central Factorial formula",
                      power, numTerms);
    System.out.println();
    sumCentral = computeAndPrintSum(cps, power, numTerms);
    System.out.printf("Computing S(%d, %d) using series addition", power,
                      numTerms);
    System.out.println();
    sumSeries = sps.computeSumUsingSeries(power, numTerms);
    System.out.println("Sum computed = " + sumSeries);
    if (!sumFaulhaber.equals(sumSeries) || !sumBernoulli.equals(sumSeries)
        || !sumStirling.equals(sumSeries) || !sumEuler.equals(sumSeries)
        || !sumCentral.equals(sumSeries)) {
      System.out.println("Sums do not match!");
    }
    System.exit(0);
  }
}
```

4.3.1 Running the example

Following commands show how to compile and run the Java example. The output of the program is also shown:

```
$ javac -cp powersum/src/main/java/PowerSum.jar JavaExample.java
$ java -cp powersum/src/main/java/PowerSum.jar:. JavaExample
Printing Bernoulli coefficients for power 11
 1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66 0
Printing Central Factorial formula for power 5
(n + 1)n/2 + 5(n + 2)(n + 1)n(n - 1)/4 + (n + 3)(n + 2)(n + 1)n(n - 1)(n - 2)/6
Computing S(19, 1000) using Faulhaber formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Bernoulli formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Stirling formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Euler formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Central Factorial formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using series addition
Sum computed = 50501583325258379475980526890307929576070522091449331750000
```

4.4 Python

Create a file named `pythonExample.py` in your favorite text editor and paste the code below:

```
from gmpy2 import mpz, mpq
from central_factorial_power_sum import CentralFactorialPowerSum
from power_sum import PowerSum
from stirling_power_sum import StirlingPowerSum
from euler_power_sum import EulerPowerSum
from bernoulli_power_sum import BernoulliPowerSum
from faulhaber_power_sum import FaulhaberPowerSum

def print_coefficients(ps: PowerSum, power: int):
  coeffs: mpq = ps.get_coefficients(power)
  for i in range(len(coeffs)):
    print(f' {coeffs[i]}', end='')
  print('')

def compute_and_print_sum(ps: PowerSum, power: int, num_terms: int) -> mpz:
  sum: mpz = ps.compute_sum(power, num_terms)
  print(f'Sum computed = {sum}')
  return sum

fps = FaulhaberPowerSum()
bps = BernoulliPowerSum()
sps = StirlingPowerSum()
eps = EulerPowerSum()
cps = CentralFactorialPowerSum()
power = 11
print(f'Printing Bernoulli coefficients for power {power}')
print_coefficients(bps, power)
power = 5
print(f'Printing Central Factorial coefficients for power {power}')
print_coefficients(cps, power)
power = 19
num_terms = 1000
print(f'Computing S({power}, {num_terms}) using Faulhaber formula')
sum_faulhaber = compute_and_print_sum(fps, power, num_terms)
print(f'Computing S({power}, {num_terms}) using Bernoulli formula')
sum_bernoulli = compute_and_print_sum(bps, power, num_terms)
print(f'Computing S({power}, {num_terms}) using Stirling formula')
sum_stirling = compute_and_print_sum(sps, power, num_terms)
print(f'Computing S({power}, {num_terms}) using Euler formula')
sum_euler = compute_and_print_sum(eps, power, num_terms)
print(f'Computing S({power}, {num_terms}) using Central Factorial formula')
sum_central = compute_and_print_sum(cps, power, num_terms)
print(f'Computing S({power}, {num_terms}) using series addition')
sum_series = sps.compute_sum_using_series(power, num_terms)
print(f'Sum computed = {sum_series}');
if (sum_faulhaber != sum_series or sum_bernoulli != sum_series
    or sum_stirling != sum_series or sum_euler != sum_series
    or sum_central != sum_series):
  print('Sums do not match!')
```

4.4.1 Running the example

Following commands show how to run the Python example. The output of the program is also shown:

```
$ python3 -m venv powersumenv
$ pip install -r powersum/src/main/python/requirements.txt
$ PYTHONPATH=powersum/src/main/python/ python3 ./PythonExample.py
Printing Bernoulli coefficients for power 11
 1 -1/2 1/6 0 -1/30 0 1/42 0 -1/30 0 5/66 0
Printing Central Factorial formula for power 5
(n + 1)n/2 + 5(n + 2)(n + 1)n(n - 1)/4 + (n + 3)(n + 2)(n + 1)n(n - 1)(n - 2)/6
Computing S(19, 1000) using Faulhaber formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Bernoulli formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Stirling formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Euler formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using Central Factorial formula
Sum computed = 50501583325258379475980526890307929576070522091449331750000
Computing S(19, 1000) using series addition
Sum computed = 50501583325258379475980526890307929576070522091449331750000
```

5 Regression testing

To test the functionalities of the implementation for all languages, type the following in a terminal window:

```
$ powersum/tests/runFunctionalTests.sh
```

This will build the binaries, run a core set of functional tests, and verify for correctness.

6 Performance testing

Performance testing requires `awk` and gnuplot installed on your system. To test the performance and generate graphic plots of the results, type the following from a terminal window:

```
$ mkdir PerformanceResults
$ powersum/tests/runPerformanceTests.sh ./PerformanceResults
```

This will use the C implementation to run the performance tests. It will take several minutes for the tests to finish. The time depends on the CPU speed on your system. Tests are run for powers ranging from 10 to 4000 and number of terms ranging from 10 to 1000000. At the end, you should see the following directories populated with files:

```
performanceResults/perfTestResults-c/plotData
performanceResults/perfTestResults-c/images
```

The `plotData` directory contains the test results as CSV files. The `images` directory contains PNG files which are graphical plots generated from the CSV files.
Optionally, you can run the performance tests against the C++ version by typing:

```
$ mkdir PerformanceResults
$ powersum/tests/runPerformanceTests.sh ./PerformanceResults cpp
```

It will create the following directories for output:

```
performanceResults/perfTestResults-cpp/plotData
performanceResults/perfTestResults-cpp/images
```

Except for some noise, there should be very little difference in the final results between C and C++ implementations.