scMSI Testing Documentation

Version: 1.0
Date Compiled: September 2024

Overview

This testing documentation outlines rigorous testing procedures for scMSI, a bioinformatics tool for deconvoluting microsatellite length distributions. These procedures are intended for bioinformatics professionals experienced in large-scale data processing and high-performance computing environments.

Testing Prerequisites

- Python 3.7 environment with HPC-optimized libraries (Intel MKL, OpenBLAS)
- Minimum of 512 GB RAM and 64 CPU cores
- Availability of bwa, samtools, bedtools, gurobi, and the scMSI codebase
- Access to example genomic data and test files in the scMSI repository

Test Data Specification

Test data is available within the scMSI repository under the `Examples/` directory. The following commands can be used to download additional high-complexity test datasets:

```
wget https://github.com/SeqAnalysis/scMSI.git
unzip scMSI-main.zip
```

Test Command Execution

Run the following commands to test the parallelized execution of scMSI using multiple threads and high-performance I/O configurations:

```
python main.py -d Examples/tumor_data.txt -c Examples/control_data.txt -o results/ --threads 64 --log-level DEBUG
```

This command utilizes 64 threads for parallel execution and includes debug-level logging for in-depth error tracking.

python Z_test.py

Output File Format and Validation

The output will include:
- `means `: Mean estimates for each sub-clone.
- `covariances `: Covariance matrices for the sub-clonal distributions.
- `fractions `: Sub-clonal fractions.

Microsatellite status:mss

or

Microsatellite status:msi

Validate the output by comparing it to reference results in the `expectedResults_large/` directory, using checksum validation:
```
sha256sum results/* expectedResults_large/*
```

Performance Profiling and Optimization

Use performance profiling tools such as `gprof` or Intel VTune to analyze CPU-bound or memory-bound performance bottlenecks:
```
gprof python main.py gmon.out > profiling_results.txt
```
Alternatively, run VTune to profile parallel performance across multiple cores:
```
vtune -collect hotspots -- python main.py -d Examples/tumor_data.txt -c Examples/control_data.txt -o results/ --threads 64
```

Troubleshooting

- If execution fails due to memory exhaustion, reduce the number of parallel threads and monitor memory usage with:
```
htop
```
- For numerical stability issues (e.g., NaN values in the covariance matrix), ensure that input data is properly normalized. Use the `numpy` function:
```
X = (X - X.mean(axis=0)) / X.std(axis=0)
```