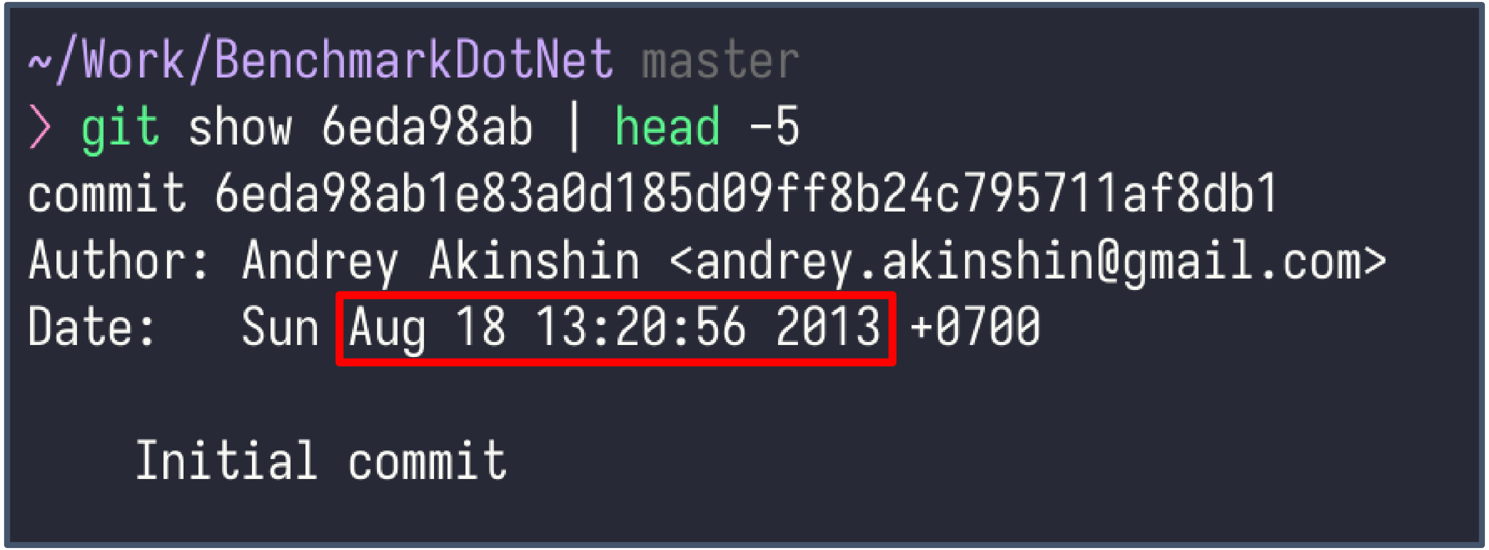
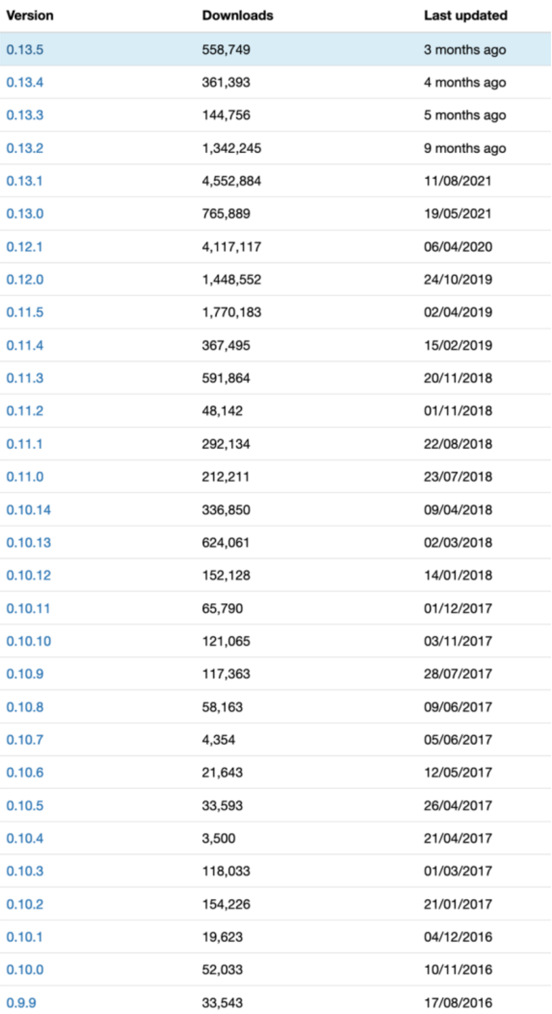
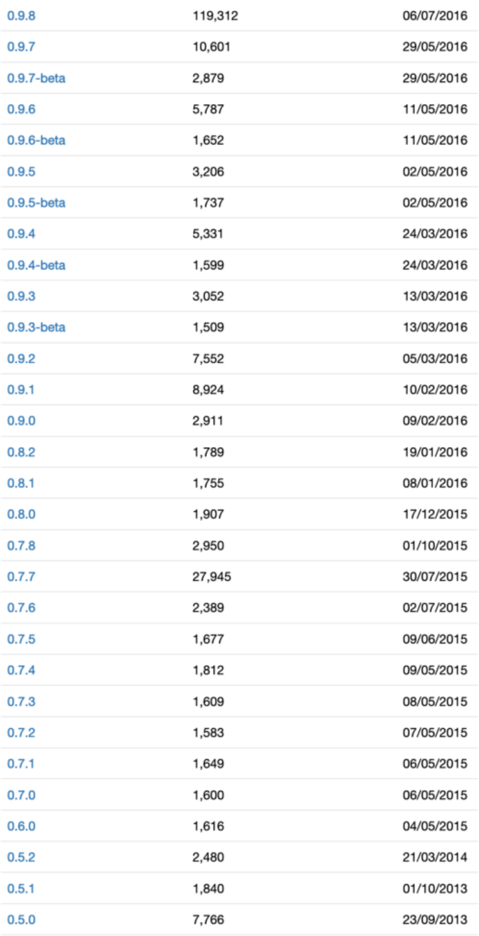
BenchmarkDotNet State of the art

Andrey Akinshin, Adam Sitnik



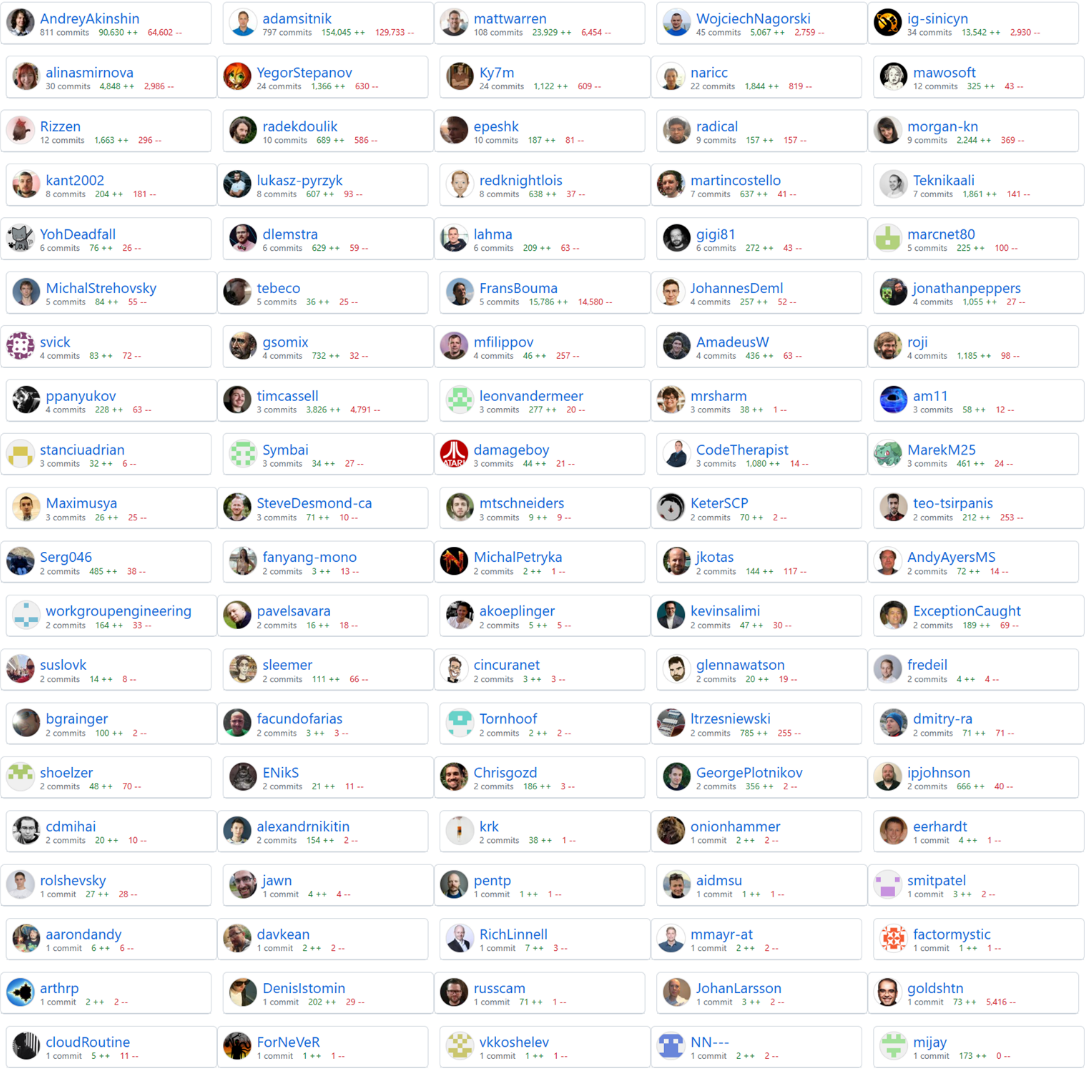
```
~/Work/BenchmarkDotNet master
> git show 6eda98ab | head -5
commit 6eda98ab1e83a0d185d09ff8b24c795711af8db1
Author: Andrey Akinshin <andrey.akinshin@gmail.com>
Date:   Sun Aug 18 13:20:56 2013 +0700

    Initial commit
```





Version	Downloads	Last updated
0.13.5	558,749	3 months ago
0.13.4	361,393	4 months ago
0.13.3	144,756	5 months ago
0.13.2	1,342,245	9 months ago
0.13.1	4,552,884	11/08/2021
0.13.0	765,889	19/05/2021
0.12.1	4,117,117	06/04/2020
0.12.0	1,448,552	24/10/2019
0.11.5	1,770,183	02/04/2019
0.11.4	367,495	15/02/2019
0.11.3	591,864	20/11/2018
0.11.2	48,142	01/11/2018
0.11.1	292,134	22/08/2018
0.11.0	212,211	23/07/2018
0.10.14	336,850	09/04/2018
0.10.13	624,061	02/03/2018
0.10.12	152,128	14/01/2018
0.10.11	65,790	01/12/2017
0.10.10	121,065	03/11/2017
0.10.9	117,363	28/07/2017
0.10.8	58,163	09/06/2017
0.10.7	4,354	05/06/2017
0.10.6	21,643	12/05/2017
0.10.5	33,593	26/04/2017
0.10.4	3,500	21/04/2017
0.10.3	118,033	01/03/2017
0.10.2	154,226	21/01/2017
0.10.1	19,623	04/12/2016
0.10.0	52,033	10/11/2016
0.9.9	33,543	17/08/2016
0.9.8	119,312	06/07/2016
0.9.7	10,601	29/05/2016
0.9.7-beta	2,879	29/05/2016
0.9.6	5,787	11/05/2016
0.9.6-beta	1,652	11/05/2016
0.9.5	3,206	02/05/2016
0.9.5-beta	1,737	02/05/2016
0.9.4	5,331	24/03/2016
0.9.4-beta	1,599	24/03/2016
0.9.3	3,052	13/03/2016
0.9.3-beta	1,509	13/03/2016
0.9.2	7,552	05/03/2016
0.9.1	8,924	10/02/2016
0.9.0	2,911	09/02/2016
0.8.2	1,789	19/01/2016
0.8.1	1,755	08/01/2016
0.8.0	1,907	17/12/2015
0.7.8	2,950	01/10/2015
0.7.7	27,945	30/07/2015
0.7.6	2,389	02/07/2015
0.7.5	1,677	09/06/2015
0.7.4	1,812	09/05/2015
0.7.3	1,609	08/05/2015
0.7.2	1,583	07/05/2015
0.7.1	1,649	06/05/2015
0.7.0	1,600	06/05/2015
0.6.0	1,616	04/05/2015
0.5.2	2,480	21/03/2014
0.5.1	1,840	01/10/2013
0.5.0	7,766	23/09/2013

The Contributors



AndreyAkinshin 811 commits 90,630 ++ 64,602 --	adamsitnik 797 commits 154,045 ++ 129,733 --	mattwarren 108 commits 23,929 ++ 6,454 --	WojciechNagorski 45 commits 5,067 ++ 2,759 --	ig-sinicyn 34 commits 13,542 ++ 2,930 --
alinasmirnova 30 commits 4,848 ++ 2,986 --	YegorStepanov 24 commits 1,366 ++ 630 --	Ky7m 24 commits 1,122 ++ 609 --	naricc 22 commits 1,844 ++ 819 --	mawosoft 12 commits 325 ++ 43 --
Rizzen 12 commits 1,663 ++ 296 --	radekdoulik 10 commits 689 ++ 586 --	epeshk 10 commits 187 ++ 81 --	radical 9 commits 157 ++ 157 --	morgan-kn 9 commits 2,244 ++ 369 --
kant2002 8 commits 204 ++ 181 --	lukasz-pyrzyk 8 commits 607 ++ 93 --	redknightlois 8 commits 638 ++ 37 --	martincostello 7 commits 637 ++ 41 --	Teknikaali 7 commits 1,861 ++ 141 --
YohDeadfall 6 commits 76 ++ 26 --	dlemstra 6 commits 629 ++ 59 --	lahma 6 commits 209 ++ 63 --	gigi81 6 commits 272 ++ 43 --	marcnet80 5 commits 225 ++ 100 --
MichalStrehovsky 5 commits 84 ++ 55 --	tebeco 5 commits 36 ++ 25 --	FransBouma 5 commits 15,786 ++ 14,580 --	JohannesDeml 4 commits 257 ++ 52 --	jonathanpeppers 4 commits 1,055 ++ 27 --
svick 4 commits 83 ++ 72 --	gsomix 4 commits 732 ++ 32 --	mfilippov 4 commits 46 ++ 257 --	AmadeusW 4 commits 436 ++ 63 --	roji 4 commits 1,185 ++ 98 --
ppanyukov 4 commits 228 ++ 63 --	timcassell 3 commits 3,826 ++ 4,791 --	leonvandermeer 3 commits 277 ++ 20 --	mrsharm 3 commits 38 ++ 1 --	am11 3 commits 58 ++ 12 --
stanciuadrian 3 commits 32 ++ 6 --	Symbai 3 commits 34 ++ 27 --	damageboy 3 commits 44 ++ 21 --	CodeTherapist 3 commits 1,080 ++ 14 --	MarekM25 3 commits 461 ++ 24 --
Maximusya 3 commits 26 ++ 25 --	SteveDesmond-ca 3 commits 71 ++ 10 --	mtschneiders 3 commits 9 ++ 9 --	KeterSCP 2 commits 70 ++ 2 --	teo-tsirpanis 2 commits 212 ++ 253 --
Serg046 2 commits 485 ++ 38 --	fanyang-mono 2 commits 3 ++ 13 --	MichalPetryka 2 commits 2 ++ 1 --	jkotas 2 commits 144 ++ 117 --	AndyAyersMS 2 commits 72 ++ 14 --
workgroupengineering 2 commits 164 ++ 33 --	pavelsavara 2 commits 16 ++ 18 --	akoeplinger 2 commits 5 ++ 5 --	kevinsalimi 2 commits 47 ++ 30 --	ExceptionCaught 2 commits 189 ++ 69 --
suslovk 2 commits 14 ++ 8 --	sleemer 2 commits 111 ++ 66 --	cincuranet 2 commits 3 ++ 3 --	glennawatson 2 commits 20 ++ 19 --	fredeil 2 commits 4 ++ 4 --
bgrainger 2 commits 100 ++ 2 --	facundofarias 2 commits 3 ++ 3 --	Tornhoof 2 commits 2 ++ 2 --	ltrzesniewski 2 commits 785 ++ 255 --	dmitry-ra 2 commits 71 ++ 71 --
shoelzer 2 commits 48 ++ 70 --	ENikS 2 commits 21 ++ 11 --	Chrisgozd 2 commits 186 ++ 3 --	GeorgePlotnikov 2 commits 356 ++ 2 --	ipjohnson 2 commits 666 ++ 40 --
cdmihai 2 commits 20 ++ 10 --	alexandrnikitin 2 commits 154 ++ 2 --	krk 2 commits 38 ++ 1 --	onionhammer 1 commit 2 ++ 2 --	eerhardt 1 commit 4 ++ 1 --
rolshevsky 1 commit 27 ++ 28 --	jawn 1 commit 4 ++ 4 --	pentp 1 commit 1 ++ 1 --	aidmsu 1 commit 1 ++ 1 --	smitpatel 1 commit 3 ++ 2 --
aarondandy 1 commit 6 ++ 6 --	davkean 1 commit 2 ++ 2 --	RichLinnell 1 commit 7 ++ 3 --	mmayr-at 1 commit 2 ++ 2 --	factormystic 1 commit 1 ++ 1 --
arthrp 1 commit 2 ++ 2 --	DenisIstomin 1 commit 202 ++ 29 --	russcam 1 commit 71 ++ 1 --	JohanLarsson 1 commit 3 ++ 2 --	goldshtn 1 commit 73 ++ 5,416 --
cloudRoutine 1 commit 5 ++ 11 --	ForNeVeR 1 commit 1 ++ 1 --	vkkoshelev 1 commit 1 ++ 1 --	NN--- 1 commit 2 ++ 2 --	mijay 1 commit 173 ++ 0 --

BenchmarkDotNet

Powerful .NET library for benchmarking

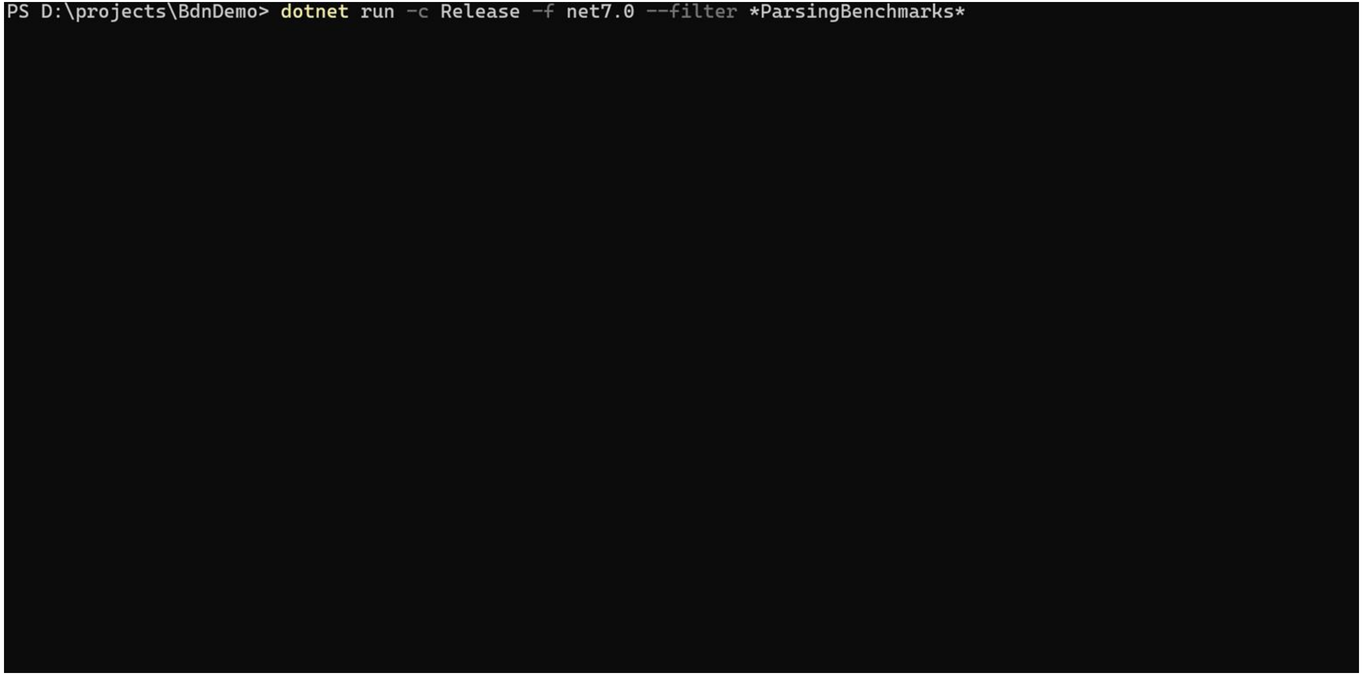
stars 8964 | used by 15745 | downloads 18624834

Introduction

Sample

```
public class ParsingBenchmarks
{
    [Benchmark]
    public int ParseInt() => int.Parse("123456789");
}

void Main(string[] args)
    => BenchmarkRunner.Run<ParsingBenchmarks>();
```

Demo

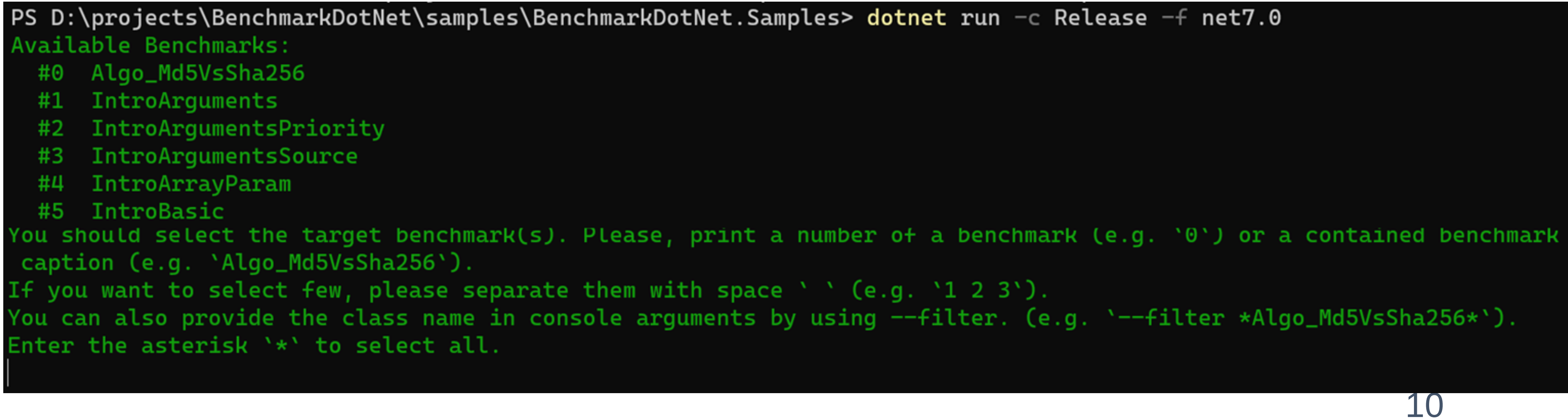


```
PS D:\projects\BdnDemo> dotnet run -c Release -f net7.0 --filter *ParsingBenchmarks*
```

How does it work?

BenchmarkSwitcher

```
void Main(string[] args)
    => BenchmarkSwitcher
            .FromAssembly(typeof(Program).Assembly)
            .Run(args, config: null);
```



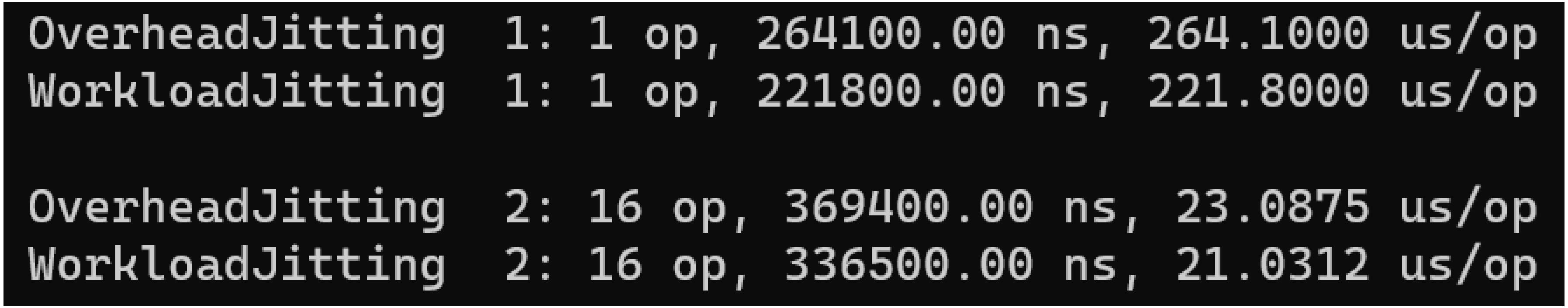
```
PS D:\projects\BenchmarkDotNet\samples\BenchmarkDotNet.Samples> dotnet run -c Release -f net7.0
Available Benchmarks:
  #0  Algo_Md5VsSha256
  #1  IntroArguments
  #2  IntroArgumentsPriority
  #3  IntroArgumentsSource
  #4  IntroArrayParam
  #5  IntroBasic
You should select the target benchmark(s). Please, print a number of a benchmark (e.g. `0`) or a contained benchmark
 caption (e.g. `Algo_Md5VsSha256`).
If you want to select few, please separate them with space ` ` (e.g. `1 2 3`).
You can also provide the class name in console arguments by using --filter. (e.g. `--filter *Algo_Md5VsSha256*`).
Enter the asterisk `*` to select all.
```

Configuration

- [Attributes]
- Fluent API
- Command line arguments
- Every filtered benchmark is executed for every Job defined in the Config.

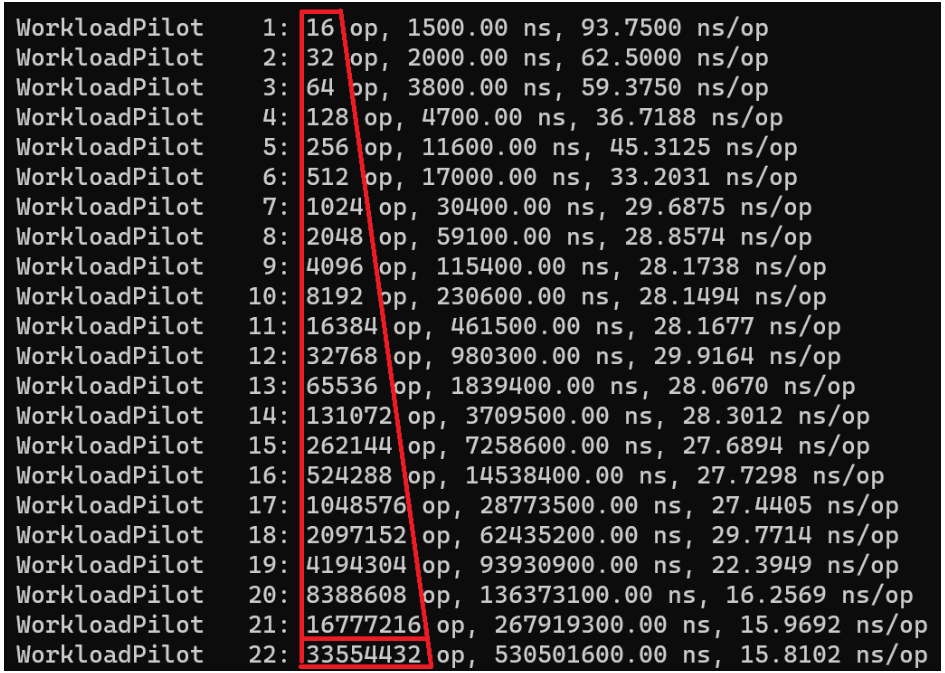
```
var config = DefaultConfig.Instance
     .AddJob(Job.Default.WithAbc())
     .AddJob(Job.Default.WithXyz());
```

Jitting



```
OverheadJitting  1: 1 op, 264100.00 ns, 264.1000 us/op
WorkloadJitting  1: 1 op, 221800.00 ns, 221.8000 us/op

OverheadJitting  2: 16 op, 369400.00 ns, 23.0875 us/op
WorkloadJitting  2: 16 op, 336500.00 ns, 21.0312 us/op
```

Pilot stage – perfect invocation count

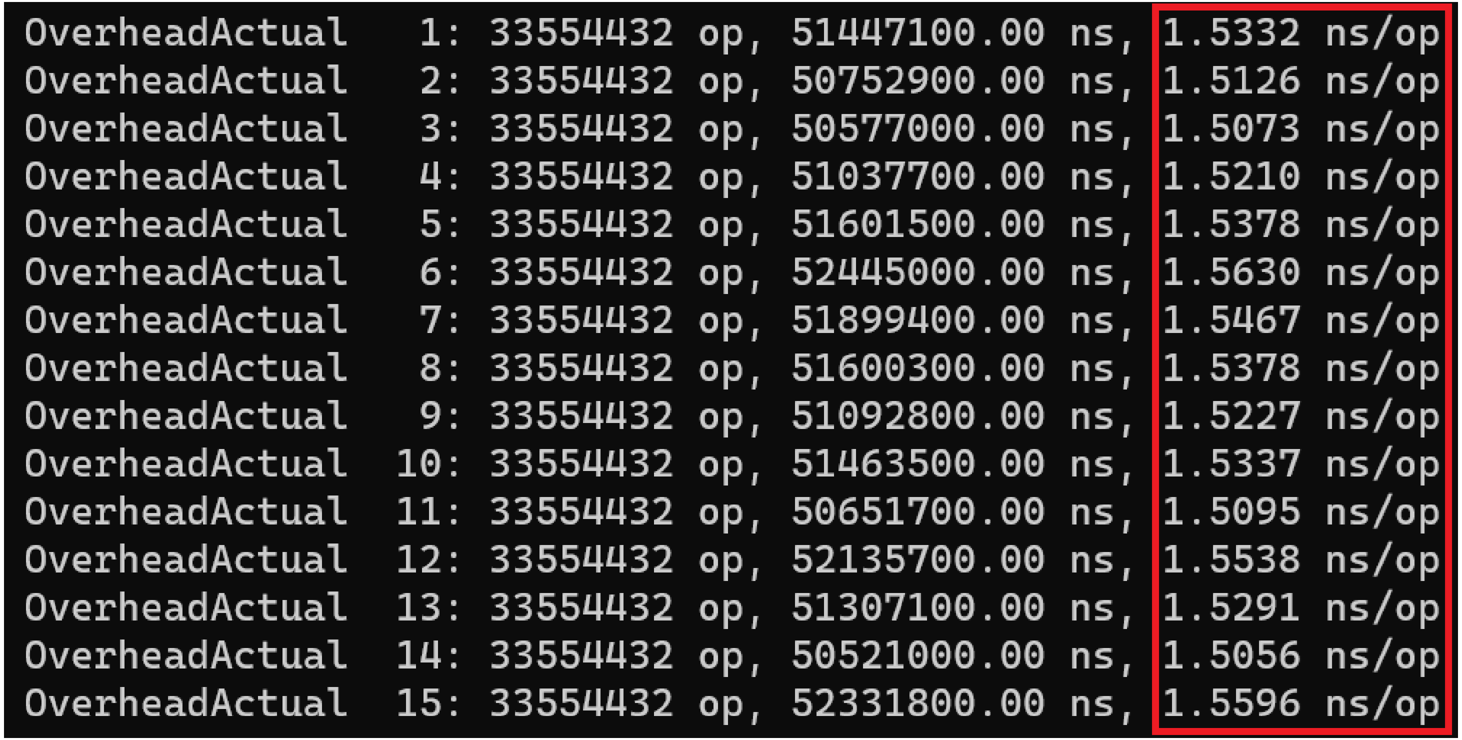


```
WorkloadPilot    1: 16 op, 1500.00 ns, 93.7500 ns/op
WorkloadPilot    2: 32 op, 2000.00 ns, 62.5000 ns/op
WorkloadPilot    3: 64 op, 3800.00 ns, 59.3750 ns/op
WorkloadPilot    4: 128 op, 4700.00 ns, 36.7188 ns/op
WorkloadPilot    5: 256 op, 11600.00 ns, 45.3125 ns/op
WorkloadPilot    6: 512 op, 17000.00 ns, 33.2031 ns/op
WorkloadPilot    7: 1024 op, 30400.00 ns, 29.6875 ns/op
WorkloadPilot    8: 2048 op, 59100.00 ns, 28.8574 ns/op
WorkloadPilot    9: 4096 op, 115400.00 ns, 28.1738 ns/op
WorkloadPilot   10: 8192 op, 230600.00 ns, 28.1494 ns/op
WorkloadPilot   11: 16384 op, 461500.00 ns, 28.1677 ns/op
WorkloadPilot   12: 32768 op, 980300.00 ns, 29.9164 ns/op
WorkloadPilot   13: 65536 op, 1839400.00 ns, 28.0670 ns/op
WorkloadPilot   14: 131072 op, 3709500.00 ns, 28.3012 ns/op
WorkloadPilot   15: 262144 op, 7258600.00 ns, 27.6894 ns/op
WorkloadPilot   16: 524288 op, 14538400.00 ns, 27.7298 ns/op
WorkloadPilot   17: 1048576 op, 28773500.00 ns, 27.4405 ns/op
WorkloadPilot   18: 2097152 op, 62435200.00 ns, 29.7714 ns/op
WorkloadPilot   19: 4194304 op, 93930900.00 ns, 22.3949 ns/op
WorkloadPilot   20: 8388608 op, 136373100.00 ns, 16.2569 ns/op
WorkloadPilot   21: 16777216 op, 267919300.00 ns, 15.9692 ns/op
WorkloadPilot   22: 33554432 op, 530501600.00 ns, 15.8102 ns/op
```

Heuristic

`job.WithInvocationCount(count) or --invocationCount`

Result = (Result + Overhead) - Overhead

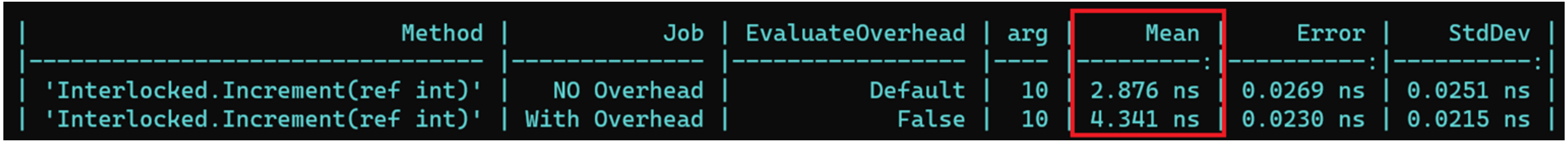


```
OverheadActual   1: 33554432 op, 51447100.00 ns, 1.5332 ns/op
OverheadActual   2: 33554432 op, 50752900.00 ns, 1.5126 ns/op
OverheadActual   3: 33554432 op, 50577000.00 ns, 1.5073 ns/op
OverheadActual   4: 33554432 op, 51037700.00 ns, 1.5210 ns/op
OverheadActual   5: 33554432 op, 51601500.00 ns, 1.5378 ns/op
OverheadActual   6: 33554432 op, 52445000.00 ns, 1.5630 ns/op
OverheadActual   7: 33554432 op, 51899400.00 ns, 1.5467 ns/op
OverheadActual   8: 33554432 op, 51600300.00 ns, 1.5378 ns/op
OverheadActual   9: 33554432 op, 51092800.00 ns, 1.5227 ns/op
OverheadActual  10: 33554432 op, 51463500.00 ns, 1.5337 ns/op
OverheadActual  11: 33554432 op, 50651700.00 ns, 1.5095 ns/op
OverheadActual  12: 33554432 op, 52135700.00 ns, 1.5538 ns/op
OverheadActual  13: 33554432 op, 51307100.00 ns, 1.5291 ns/op
OverheadActual  14: 33554432 op, 50521000.00 ns, 1.5056 ns/op
OverheadActual  15: 33554432 op, 52331800.00 ns, 1.5596 ns/op
```

The Overhead

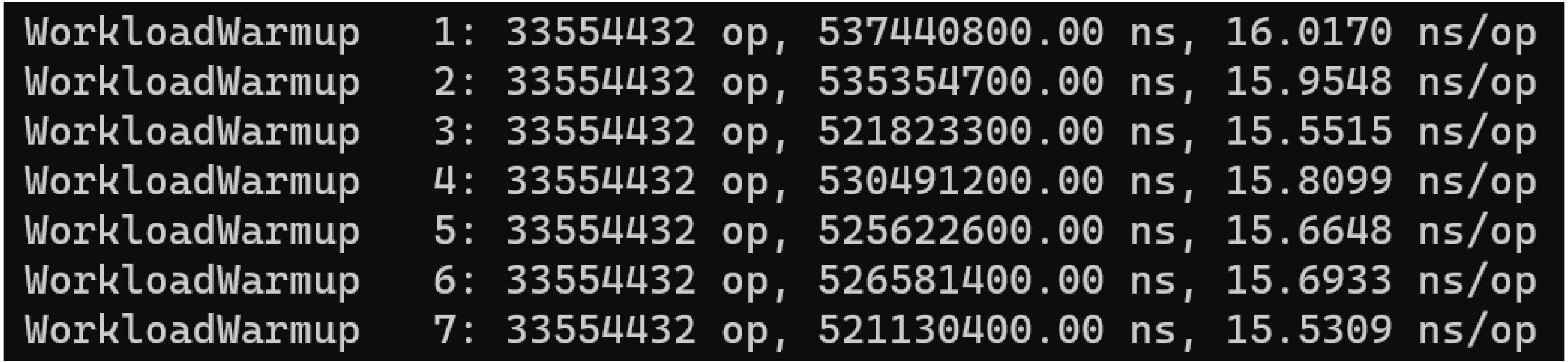
```
[Benchmark(Description = "Interlocked.Increment(ref int)")]
[Arguments(10)]
public int Increment(ref int arg) => Interlocked.Increment(ref arg);

[Benchmark]
[Arguments(10)]
public int Overhead(ref int arg) => 0;

DefaultConfig.Instance
    .AddJob(Job.Default.WithId("NO Overhead"))
    .AddJob(Job.Default.WithEvaluateOverhead(false).WithId("With Overhead"))
```

The difference



```
|                           Method |           Job | EvaluateOverhead | arg |     Mean |     Error |    StdDev |
|--------------------------------- |-------------- |----------------- |---- |---------:|----------:|----------:|
| 'Interlocked.Increment(ref int)' |   NO Overhead |          Default |  10 | 2.876 ns | 0.0269 ns | 0.0251 ns |
| 'Interlocked.Increment(ref int)' | With Overhead |            False |  10 | 4.341 ns | 0.0230 ns | 0.0215 ns |
```

Warmup stage

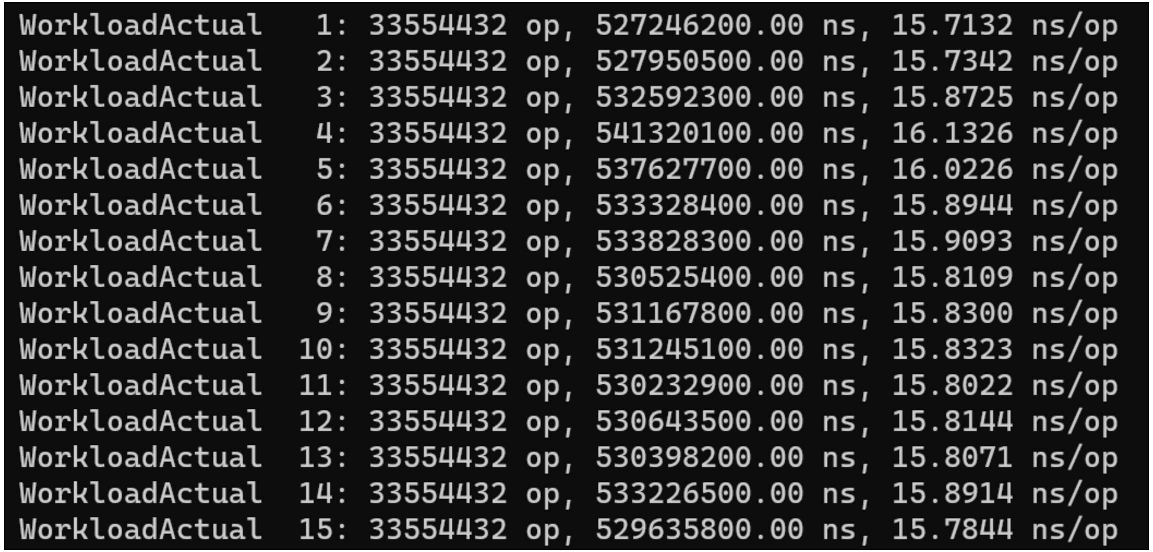


```
WorkloadWarmup   1: 33554432 op, 537440800.00 ns, 16.0170 ns/op
WorkloadWarmup   2: 33554432 op, 535354700.00 ns, 15.9548 ns/op
WorkloadWarmup   3: 33554432 op, 521823300.00 ns, 15.5515 ns/op
WorkloadWarmup   4: 33554432 op, 530491200.00 ns, 15.8099 ns/op
WorkloadWarmup   5: 33554432 op, 525622600.00 ns, 15.6648 ns/op
WorkloadWarmup   6: 33554432 op, 526581400.00 ns, 15.6933 ns/op
WorkloadWarmup   7: 33554432 op, 521130400.00 ns, 15.5309 ns/op
```

```
job.WithWarmupCount(count)
or
--warmupCount
--minWarmupCount
--maxWarmupCount
```

Heuristic

Actual Workload

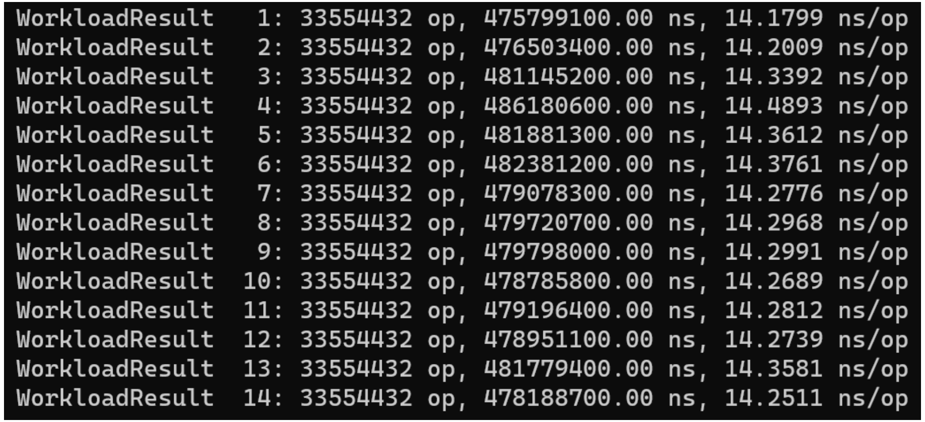


```
WorkloadActual   1: 33554432 op, 527246200.00 ns, 15.7132 ns/op
WorkloadActual   2: 33554432 op, 527950500.00 ns, 15.7342 ns/op
WorkloadActual   3: 33554432 op, 532592300.00 ns, 15.8725 ns/op
WorkloadActual   4: 33554432 op, 541320100.00 ns, 16.1326 ns/op
WorkloadActual   5: 33554432 op, 537627700.00 ns, 16.0226 ns/op
WorkloadActual   6: 33554432 op, 533328400.00 ns, 15.8944 ns/op
WorkloadActual   7: 33554432 op, 533828300.00 ns, 15.9093 ns/op
WorkloadActual   8: 33554432 op, 530525400.00 ns, 15.8109 ns/op
WorkloadActual   9: 33554432 op, 531167800.00 ns, 15.8300 ns/op
WorkloadActual  10: 33554432 op, 531245100.00 ns, 15.8323 ns/op
WorkloadActual  11: 33554432 op, 530232900.00 ns, 15.8022 ns/op
WorkloadActual  12: 33554432 op, 530643500.00 ns, 15.8144 ns/op
WorkloadActual  13: 33554432 op, 530398200.00 ns, 15.8071 ns/op
WorkloadActual  14: 33554432 op, 533226500.00 ns, 15.8914 ns/op
WorkloadActual  15: 33554432 op, 529635800.00 ns, 15.7844 ns/op
```

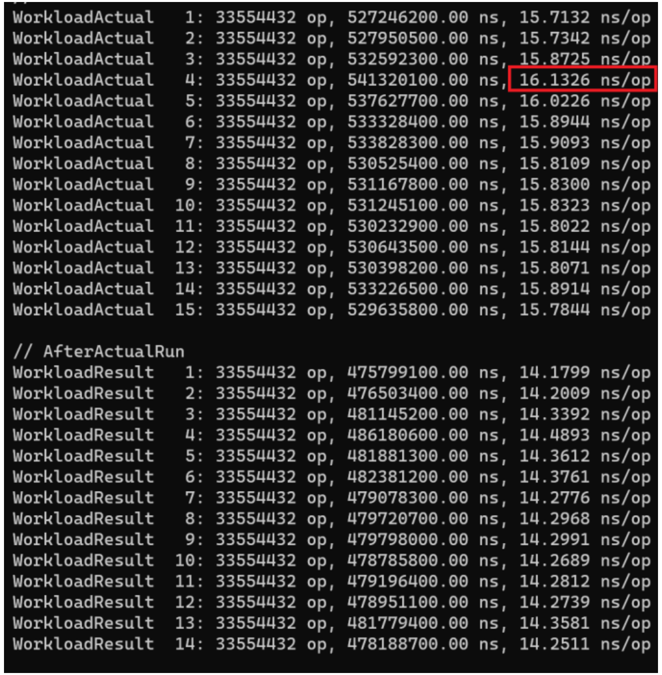
```
job.WithIterationCount(count)
```

Heuristic

Results



```
WorkloadResult   1: 33554432 op, 475799100.00 ns, 14.1799 ns/op
WorkloadResult   2: 33554432 op, 476503400.00 ns, 14.2009 ns/op
WorkloadResult   3: 33554432 op, 481145200.00 ns, 14.3392 ns/op
WorkloadResult   4: 33554432 op, 486180600.00 ns, 14.4893 ns/op
WorkloadResult   5: 33554432 op, 481881300.00 ns, 14.3612 ns/op
WorkloadResult   6: 33554432 op, 482381200.00 ns, 14.3761 ns/op
WorkloadResult   7: 33554432 op, 479078300.00 ns, 14.2776 ns/op
WorkloadResult   8: 33554432 op, 479720700.00 ns, 14.2968 ns/op
WorkloadResult   9: 33554432 op, 479798000.00 ns, 14.2991 ns/op
WorkloadResult  10: 33554432 op, 478785800.00 ns, 14.2689 ns/op
WorkloadResult  11: 33554432 op, 479196400.00 ns, 14.2812 ns/op
WorkloadResult  12: 33554432 op, 478951100.00 ns, 14.2739 ns/op
WorkloadResult  13: 33554432 op, 481779400.00 ns, 14.3581 ns/op
WorkloadResult  14: 33554432 op, 478188700.00 ns, 14.2511 ns/op
```



```
WorkloadActual   1: 33554432 op, 527246200.00 ns, 15.7132 ns/op
WorkloadActual   2: 33554432 op, 527950500.00 ns, 15.7342 ns/op
WorkloadActual   3: 33554432 op, 532592300.00 ns, 15.8725 ns/op
WorkloadActual   4: 33554432 op, 541320100.00 ns, 16.1326 ns/op
WorkloadActual   5: 33554432 op, 537627700.00 ns, 16.0226 ns/op
WorkloadActual   6: 33554432 op, 533328400.00 ns, 15.8944 ns/op
WorkloadActual   7: 33554432 op, 533828300.00 ns, 15.9093 ns/op
WorkloadActual   8: 33554432 op, 530525400.00 ns, 15.8109 ns/op
WorkloadActual   9: 33554432 op, 531167800.00 ns, 15.8300 ns/op
WorkloadActual  10: 33554432 op, 531245100.00 ns, 15.8323 ns/op
WorkloadActual  11: 33554432 op, 530232900.00 ns, 15.8022 ns/op
WorkloadActual  12: 33554432 op, 530643500.00 ns, 15.8144 ns/op
WorkloadActual  13: 33554432 op, 530398200.00 ns, 15.8071 ns/op
WorkloadActual  14: 33554432 op, 533226500.00 ns, 15.8914 ns/op
WorkloadActual  15: 33554432 op, 529635800.00 ns, 15.7844 ns/op

// AfterActualRun
WorkloadResult   1: 33554432 op, 475799100.00 ns, 14.1799 ns/op
WorkloadResult   2: 33554432 op, 476503400.00 ns, 14.2009 ns/op
WorkloadResult   3: 33554432 op, 481145200.00 ns, 14.3392 ns/op
WorkloadResult   4: 33554432 op, 486180600.00 ns, 14.4893 ns/op
WorkloadResult   5: 33554432 op, 481881300.00 ns, 14.3612 ns/op
WorkloadResult   6: 33554432 op, 482381200.00 ns, 14.3761 ns/op
WorkloadResult   7: 33554432 op, 479078300.00 ns, 14.2776 ns/op
WorkloadResult   8: 33554432 op, 479720700.00 ns, 14.2968 ns/op
WorkloadResult   9: 33554432 op, 479798000.00 ns, 14.2991 ns/op
WorkloadResult  10: 33554432 op, 478785800.00 ns, 14.2689 ns/op
WorkloadResult  11: 33554432 op, 479196400.00 ns, 14.2812 ns/op
WorkloadResult  12: 33554432 op, 478951100.00 ns, 14.2739 ns/op
WorkloadResult  13: 33554432 op, 481779400.00 ns, 14.3581 ns/op
WorkloadResult  14: 33554432 op, 478188700.00 ns, 14.2511 ns/op
```

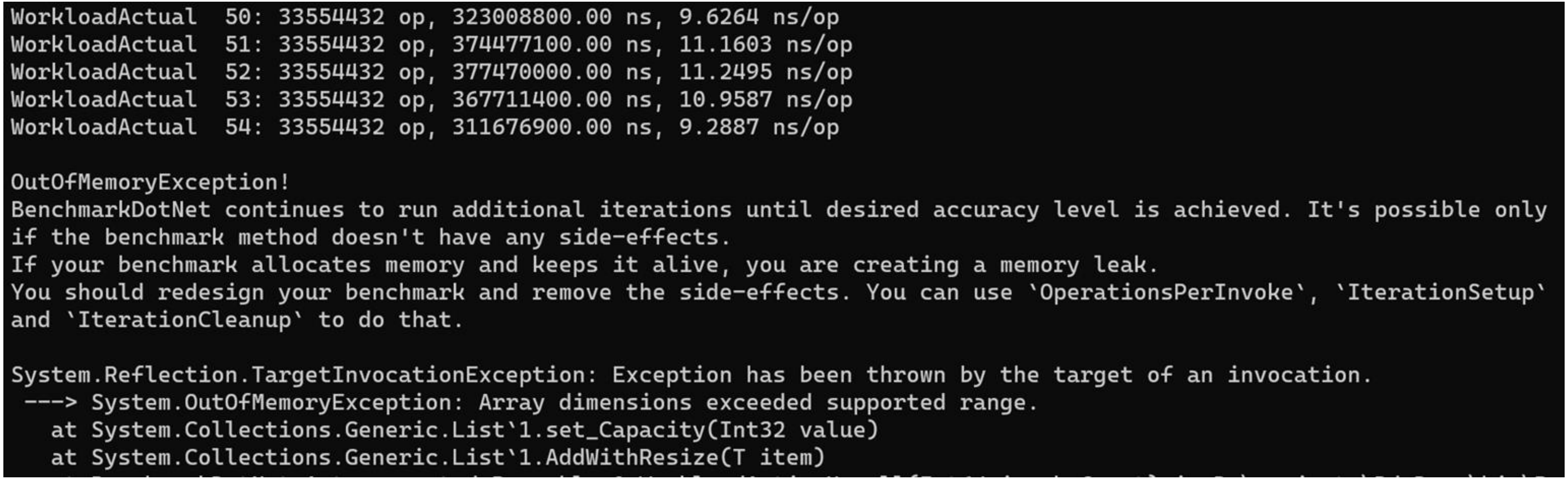
```
job.WithOutlierMode(mode)
or --outliers
```

IsOutlier

The trap

```
public class ListBenchmarks
{
    private List<int> list = new List<int>();

    [Benchmark]
    public void Add() => list.Add(1234);

    [Benchmark]
    public void AddLoop()
    {
        list.Clear();

        for (int i = 0; i < 1000; i++)
            list.Add(1234);
    }
}
```

OOM



```
WorkloadActual  50: 33554432 op, 323008800.00 ns, 9.6264 ns/op
WorkloadActual  51: 33554432 op, 374477100.00 ns, 11.1603 ns/op
WorkloadActual  52: 33554432 op, 377470000.00 ns, 11.2495 ns/op
WorkloadActual  53: 33554432 op, 367711400.00 ns, 10.9587 ns/op
WorkloadActual  54: 33554432 op, 311676900.00 ns, 9.2887 ns/op

OutOfMemoryException!
BenchmarkDotNet continues to run additional iterations until desired accuracy level is achieved. It's possible only
if the benchmark method doesn't have any side-effects.
If your benchmark allocates memory and keeps it alive, you are creating a memory leak.
You should redesign your benchmark and remove the side-effects. You can use `OperationsPerInvoke`, `IterationSetup`
and `IterationCleanup` to do that.

System.Reflection.TargetInvocationException: Exception has been thrown by the target of an invocation.
 ---> System.OutOfMemoryException: Array dimensions exceeded supported range.
   at System.Collections.Generic.List`1.set_Capacity(Int32 value)
   at System.Collections.Generic.List`1.AddWithResize(T item)
```

Strategies

- Throughput – default, perfect for microbenchmarks with a steady state
- Monitoring
  - no Pilot stage
  - no Overhead evaluation
  - Outliers remain untouched
  - 1 iteration = 1 benchmark invocation
- ColdStart – no warmup, no pilot stage

How it works: Summary

- Using statistics to get stable results
- Users don’t need to worry about specifying invocation count
- Results don’t contain overhead
- **It takes time to do all of that**
- User can specify invocation/iteration/warmup/target count
- User can customize the heuristic
- Benchmarks should not have side-effects

Architecture

Architecture

- Host Process (console app)
    - Generates a project file with C# source file (Console App)
    - Builds (Roslyn/dotnet cli)
    - Executes Child Process
- Child Process (console app)
    - **Executes benchmark**
    - Signals events to Host
    - Reports results to Host

See: IToolchain, IGenerator, IBuilder and IExecutor

Why Process-level Isolation?

- We want to have stable and repeatable results
- Order of executing benchmarks should not affect the results
  - Benchmarks can have side effects
  - GC is self-tuning (generation size can change over time)
  - JIT is also self-tuning (Tiered JIT + PGO)
  - .NET Runtime can apply some optimizations
- [InProcessToolchain] does **not** spawn new process

How to prevent inlining?

```
[Benchmark(Baseline = true)]
public void OneWay() { /* one way to solve the problem */ }
[Benchmark]
public void AnotherWay() { /* another way to solve the problem */ }
```

What if one of the methods get inlined?

How to prevent inlining without modifying the code?

```
public delegate Span<byte> TargetDelegate();

private TargetDelegate targetDelegate = BenchmarkedMethod;
```

How to minimize loop overhead?

```
private void MainMultiAction(long invokeCount)
{
    for (long i = 0; i < invokeCount; i++)
        targetDelegate();
}
```

```
private void MainMultiAction(long invokeCount)
{
    for (long i = 0; i < invokeCount / unrollFactor; i++)
    {
        targetDelegate(); targetDelegate(); targetDelegate(); targetDelegate();
        targetDelegate(); targetDelegate(); targetDelegate(); targetDelegate();
        targetDelegate(); targetDelegate(); targetDelegate(); targetDelegate();
        targetDelegate(); targetDelegate(); targetDelegate(); targetDelegate();
    }
}
```

`job.WithUnrollFactor(count) or --unrollFactor`

More info

How to prevent from Out-of-order execution?

```
private void MainMultiAction(long invokeCount)
{
    for (long i = 0; i < invokeCount / unrollFactor; i++)
    {
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
        consumer.Consume(targetDelegate()); consumer.Consume(targetDelegate());
    }
}
```

Consumer

```
public class Consumer
{
    private volatile byte byteHolder;
    // (more types skipped for brevity)
    private string stringHolder;
    private object objectHolder;

    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    public void Consume(ulong ulongValue)
        => Volatile.Write(ref ulongHolder, ulongValue);
}
```

Generating new project

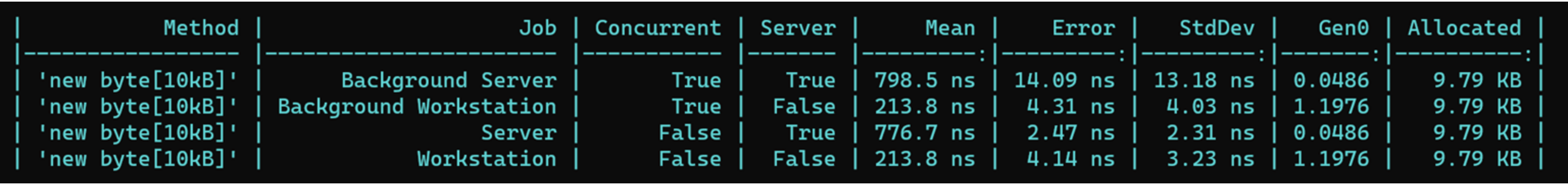
- Benchmark.not**cs** (customized for every benchmark)
- Benchmark.**csproj**
  - Architecture (Job.Env.Platform)
  - Optimizations: ALWAYS on
- Benchmark**.config -** derives from Host.config file, except of:
  - GC Mode (Job.Env.Gc)
  - JIT: Legacy/RyuJIT/LLVm (Job.Env.Jit, *LLVM only for Mono)
  - & more: GCCpuGroup, gcAllowVeryLargeObjects
- Use [KeepBenchmarkFiles] to see what is generated

See: AppConfigGenerator, Generator, EnvMode, GcMode

Different GC modes

```
[Config(typeof(GcConfig))]
public class GcBenchmarks
{
    private class GcConfig : ManualConfig
    {
        public GcConfig()
        {
            AddJob(Job.Default.WithGcMode(new GcMode { Server = true, Concurrent = true }).WithId("Background Server"));
            AddJob(Job.Default.WithGcMode(new GcMode { Server = true, Concurrent = false }).WithId("Server"));
            AddJob(Job.Default.WithGcMode(new GcMode { Server = false, Concurrent = true }).WithId("Background
Workstation"));
            AddJob(Job.Default.WithGcMode(new GcMode { Server = false, Concurrent = false }).WithId("Workstation"));

            AddDiagnoser(MemoryDiagnoser.Default);
        }
    }

    [Benchmark(Description = "new byte[10kB]")]
    public byte[] Allocate() => new byte[10000];
}
```

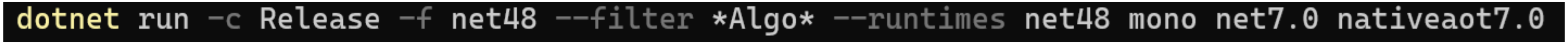
Different GC modes: sample results



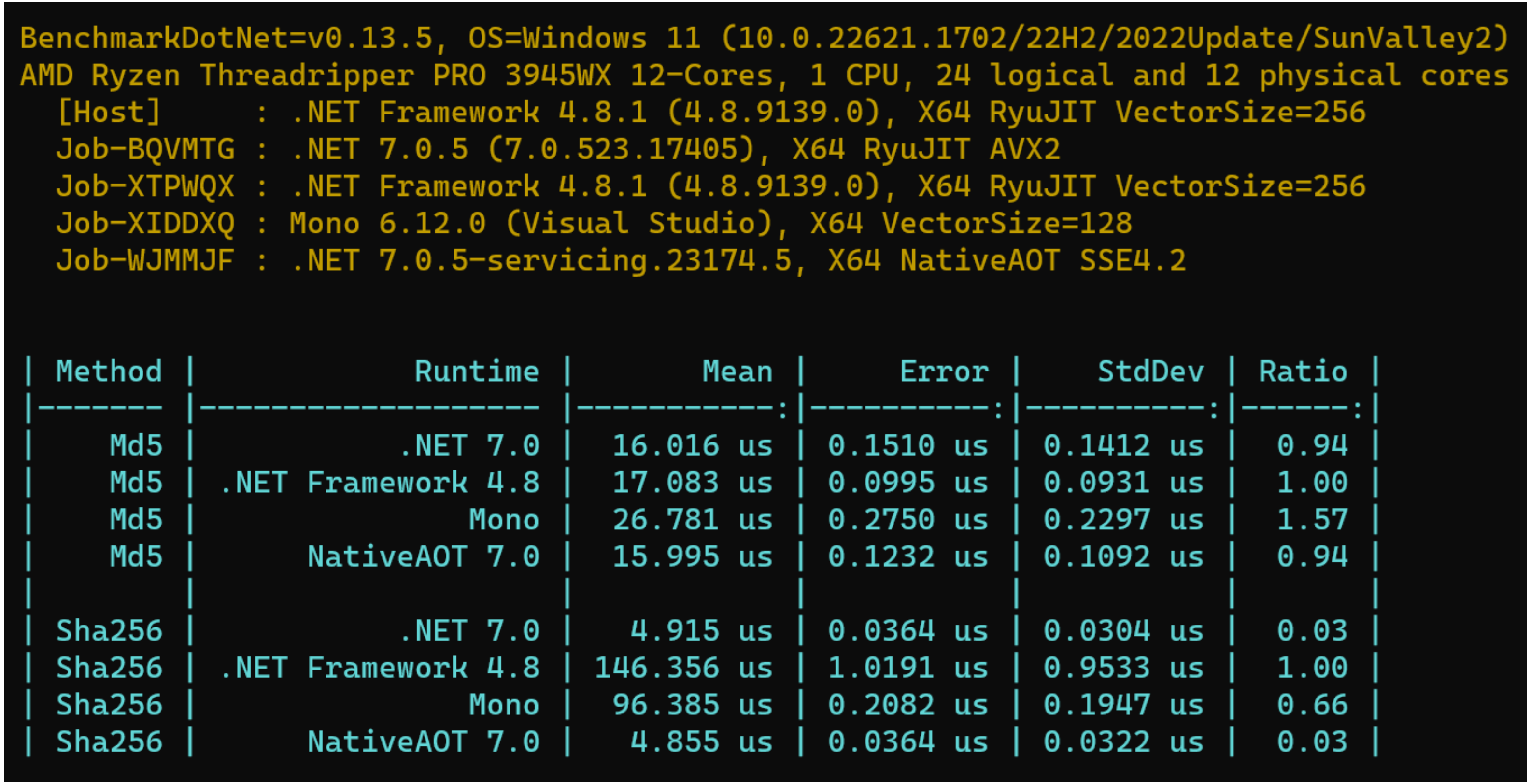
```
|           Method |                    Job | Concurrent | Server |     Mean |    Error |   StdDev |   Gen0 | Allocated |
|----------------- |----------------------- |----------- |------- |---------:|---------:|---------:|-------:|----------:|
| 'new byte[10kB]' |      Background Server |       True |   True | 798.5 ns | 14.09 ns | 13.18 ns | 0.0486 |   9.79 KB |
| 'new byte[10kB]' | Background Workstation |       True |  False | 213.8 ns |  4.31 ns |  4.03 ns | 1.1976 |   9.79 KB |
| 'new byte[10kB]' |                 Server |      False |   True | 776.7 ns |  2.47 ns |  2.31 ns | 0.0486 |   9.79 KB |
| 'new byte[10kB]' |            Workstation |      False |  False | 213.8 ns |  4.14 ns |  3.23 ns | 1.1976 |   9.79 KB |
```

Compare frameworks

```
public class Algo_Md5VsSha256
{
    private readonly byte[] data;
    private readonly MD5 md5 = MD5.Create();
    private readonly SHA256 sha256 = SHA256.Create();

    public Algo_Md5VsSha256()
    {
        data = new byte[10000];
        new Random(42).NextBytes(data);
    }

    [Benchmark]
    public byte[] Md5() => md5.ComputeHash(data);

    [Benchmark]
    public byte[] Sha256() => sha256.ComputeHash(data);
}
```



```
dotnet run -c Release -f net48 --filter *Algo* --runtimes net48 mono net7.0 nativeaot7.0
```



```
BenchmarkDotNet=v0.13.5, OS=Windows 11 (10.0.22621.1702/22H2/2022Update/SunValley2)
AMD Ryzen Threadripper PRO 3945WX 12-Cores, 1 CPU, 24 logical and 12 physical cores
  [Host]     : .NET Framework 4.8.1 (4.8.9139.0), X64 RyuJIT VectorSize=256
  Job-BQVMTG : .NET 7.0.5 (7.0.523.17405), X64 RyuJIT AVX2
  Job-XTPWQX : .NET Framework 4.8.1 (4.8.9139.0), X64 RyuJIT VectorSize=256
  Job-XIDDXQ : Mono 6.12.0 (Visual Studio), X64 VectorSize=128
  Job-WJMMJF : .NET 7.0.5-servicing.23174.5, X64 NativeAOT SSE4.2


| Method |            Runtime |       Mean |     Error |    StdDev | Ratio |
|------- |------------------- |-----------:|----------:|----------:|------:|
|    Md5 |           .NET 7.0 |  16.016 us | 0.1510 us | 0.1412 us |  0.94 |
|    Md5 | .NET Framework 4.8 |  17.083 us | 0.0995 us | 0.0931 us |  1.00 |
|    Md5 |               Mono |  26.781 us | 0.2750 us | 0.2297 us |  1.57 |
|    Md5 |      NativeAOT 7.0 |  15.995 us | 0.1232 us | 0.1092 us |  0.94 |
|        |                    |            |           |           |       |
| Sha256 |           .NET 7.0 |   4.915 us | 0.0364 us | 0.0304 us |  0.03 |
| Sha256 | .NET Framework 4.8 | 146.356 us | 1.0191 us | 0.9533 us |  1.00 |
| Sha256 |               Mono |  96.385 us | 0.2082 us | 0.1947 us |  0.66 |
| Sha256 |      NativeAOT 7.0 |   4.855 us | 0.0364 us | 0.0322 us |  0.03 |
```

Architecture: Summary

- Host process generates, builds and runs .exe per benchmark
- It helps us to get repeatable results
- It allows the users to compare different settings:
  - GC Workstation vs GC Server
  - .NET Framework vs Mono vs .NET vs NativeAOT
- It limits us to only known frameworks
- InProcessToolchain runs in process (-i)

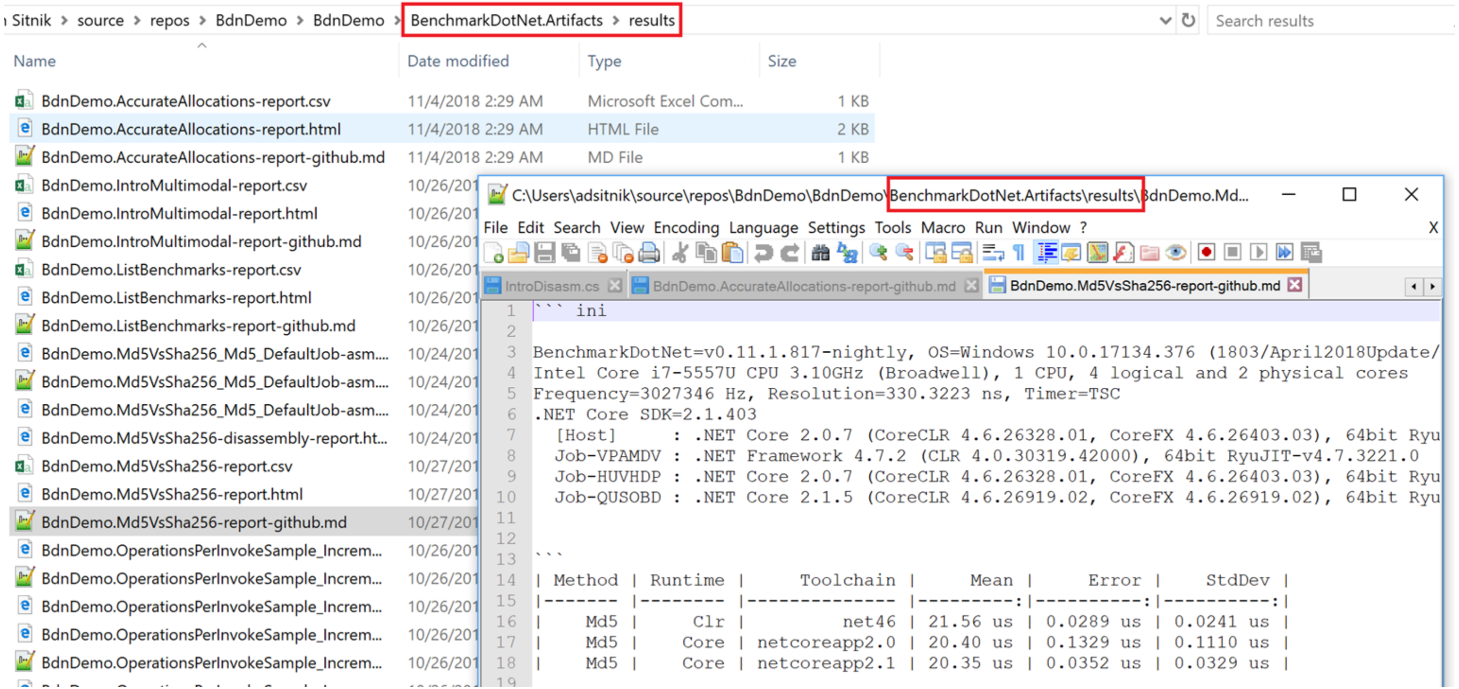
Features

Exporters

- HTML
- Markdown: GitHub, StackOverflow
- CSV
- RPlot (requires R)
- XML
- JSON

```
[AsciiDocExporter]
[CsvExporter]
[CsvMeasurementsExporter]
[HtmlExporter]
[PlainExporter]
[RPlotExporter]
[JsonExporterAttribute.Brief]
[JsonExporterAttribute.BriefCompressed]
[JsonExporterAttribute.Full]
[JsonExporterAttribute.FullCompressed]
[MarkdownExporterAttribute.Default]
[MarkdownExporterAttribute.GitHub]
[MarkdownExporterAttribute.StackOverflow]
[MarkdownExporterAttribute.Atlassian]
[XmlExporterAttribute.Brief]
[XmlExporterAttribute.BriefCompressed]
[XmlExporterAttribute.Full]
[XmlExporterAttribute.FullCompressed]
public class IntroExporters
```

.\BenchmarkDotNet.Artifacts\results



ı Sitnik › source › repos › BdnDemo › BdnDemo › BenchmarkDotNet.Artifacts › results
Search results
C:\Users\adsitnik\source\repos\BdnDemo\BdnDemo\BenchmarkDotNet.Artifacts\results\BdnDemo.Md...
File Edit Search View Encoding Language Settings Tools Macro Run Window ?
IntroDisasm.cs
BdnDemo.AccurateAllocations-report-github.md
BdnDemo.Md5VsSha256-report-github.md

Name	Date modified	Type	Size
BdnDemo.AccurateAllocations-report.csv	11/4/2018 2:29 AM	Microsoft Excel Com...	1 KB
BdnDemo.AccurateAllocations-report.html	11/4/2018 2:29 AM	HTML File	2 KB
BdnDemo.AccurateAllocations-report-github.md	11/4/2018 2:29 AM	MD File	1 KB
BdnDemo.IntroMultimodal-report.csv	10/26/201		
BdnDemo.IntroMultimodal-report.html	10/26/201		
BdnDemo.IntroMultimodal-report-github.md	10/26/201		
BdnDemo.ListBenchmarks-report.csv	10/26/201		
BdnDemo.ListBenchmarks-report.html	10/26/201		
BdnDemo.ListBenchmarks-report-github.md	10/26/201		
BdnDemo.Md5VsSha256_Md5_DefaultJob-asm....	10/24/201		
BdnDemo.Md5VsSha256_Md5_DefaultJob-asm....	10/24/201		
BdnDemo.Md5VsSha256_Md5_DefaultJob-asm....	10/24/201		
BdnDemo.Md5VsSha256-disassembly-report.ht...	10/24/201		
BdnDemo.Md5VsSha256-report.csv	10/27/201		
BdnDemo.Md5VsSha256-report.html	10/27/201		
BdnDemo.Md5VsSha256-report-github.md	10/27/201		
BdnDemo.OperationsPerInvokeSample_Increm...	10/26/201		
BdnDemo.OperationsPerInvokeSample_Increm...	10/26/201		
BdnDemo.OperationsPerInvokeSample_Increm...	10/26/201		
BdnDemo.OperationsPerInvokeSample_Increm...	10/26/201		
BdnDemo.OperationsPerInvokeSample_Increm...	10/26/201		

~~~
``` ini

BenchmarkDotNet=v0.11.1.817-nightly, OS=Windows 10.0.17134.376 (1803/April2018Update/
Intel Core i7-5557U CPU 3.10GHz (Broadwell), 1 CPU, 4 logical and 2 physical cores
Frequency=3027346 Hz, Resolution=330.3223 ns, Timer=TSC
.NET Core SDK=2.1.403
  [Host]     : .NET Core 2.0.7 (CoreCLR 4.6.26328.01, CoreFX 4.6.26403.03), 64bit Ryu
  Job-VPAMDV : .NET Framework 4.7.2 (CLR 4.0.30319.42000), 64bit RyuJIT-v4.7.3221.0
  Job-HUVHDP : .NET Core 2.0.7 (CoreCLR 4.6.26328.01, CoreFX 4.6.26403.03), 64bit Ryu
  Job-QUSOBD : .NET Core 2.1.5 (CoreCLR 4.6.26919.02, CoreFX 4.6.26919.02), 64bit Ryu


```
| Method | Runtime |      Toolchain |     Mean |      Error |     StdDev |
|------- |-------- |--------------- |---------:|-----------:|-----------:|
|    Md5 |     Clr |          net46 | 21.56 us | 0.0289 us | 0.0241 us |
|    Md5 |    Core | netcoreapp2.0 | 20.40 us | 0.1329 us | 0.1110 us |
|    Md5 |    Core | netcoreapp2.1 | 20.35 us | 0.0352 us | 0.0329 us |
~~~

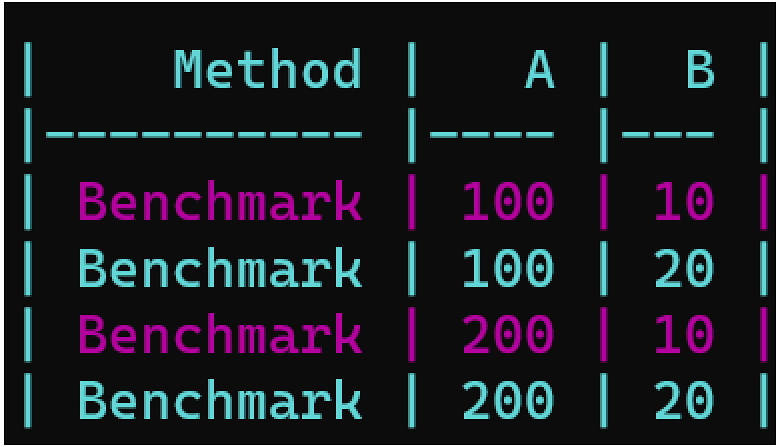
Setup & Cleanup

```
public class SetupAndCleanupExample
{
    [GlobalSetup]
    public void GlobalSetup() { }

    [IterationSetup] // sets 1 iteration = 1 invocation
    public void IterationSetup() { }

    [Benchmark]
    public void Benchmark() { }

    [IterationCleanup]
    public void IterationCleanup() { }

    [GlobalCleanup]
    public void GlobalCleanup() { }
}
```

More info

Params

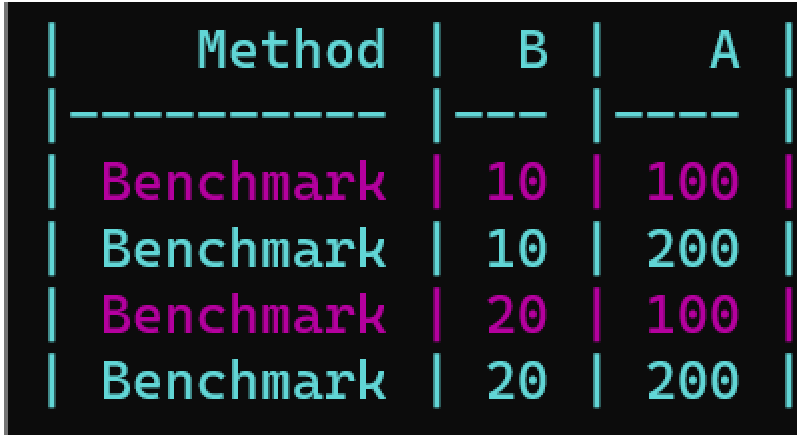
```
public class IntroParams
{
    [Params(100, 200)]
    public int A { get; set; }

    [Params(10, 20)]
    public int B { get; set; }

    [Benchmark]
    public void Benchmark()
          => Thread.Sleep(A + B + 5);
}
```



```
|    Method |   A |  B |
|---------- |---- |--- |
| Benchmark | 100 | 10 |
| Benchmark | 100 | 20 |
| Benchmark | 200 | 10 |
| Benchmark | 200 | 20 |
```

ParamsSource

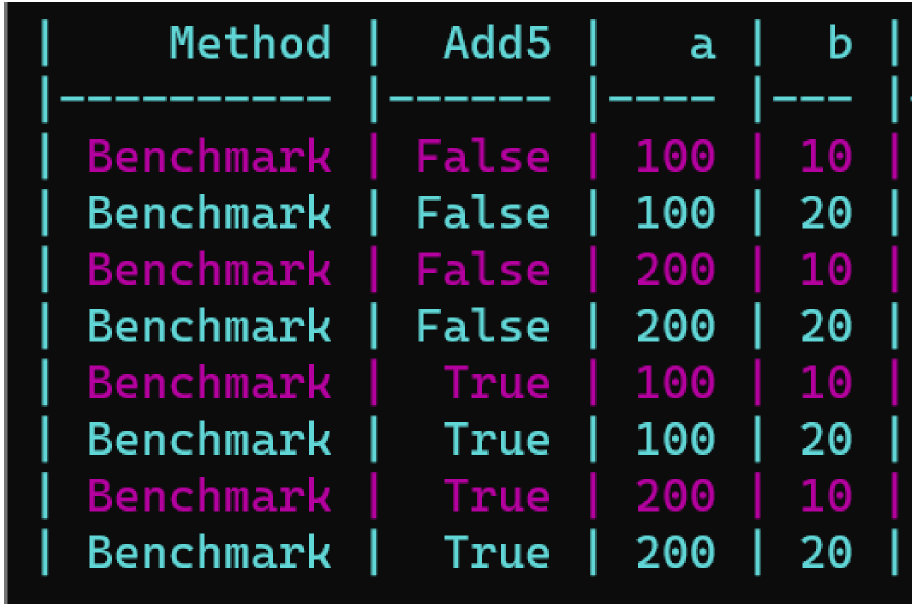
```
public class IntroParamsSource
{
    [ParamsSource(nameof(ValuesForA))]
    public int A { get; set; }

    [ParamsSource(nameof(ValuesForB))]
    public int B;

    public IEnumerable<int> ValuesForA
            => new[] { 100, 200 };

    public static IEnumerable<int> ValuesForB()
            => new[] { 10, 20 };

    [Benchmark]
    public void Benchmark()
            => Thread.Sleep(A + B + 5);
}
```



```
|    Method |  B |   A |
|---------- |--- |---- |
| Benchmark | 10 | 100 |
| Benchmark | 10 | 200 |
| Benchmark | 20 | 100 |
| Benchmark | 20 | 200 |
```

Arguments

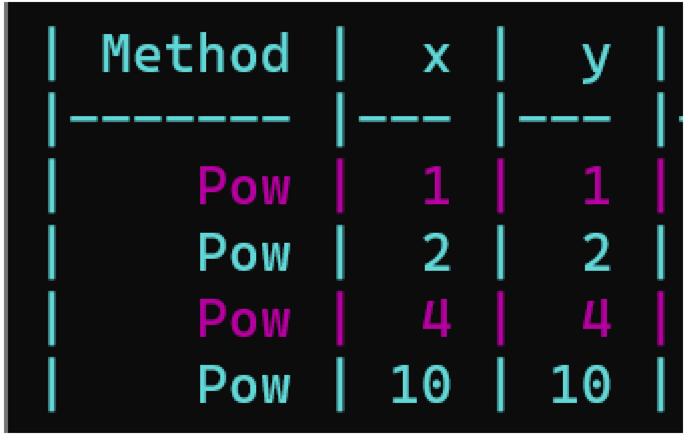
```
public class IntroArguments
{
    [Params(true, false)]
    public bool Add5;

    [Benchmark]
    [Arguments(100, 10)]
    [Arguments(100, 20)]
    [Arguments(200, 10)]
    [Arguments(200, 20)]
    public void Benchmark(int a, int b)
    {
        if (Add5)
            Thread.Sleep(a + b + 5);
        else
            Thread.Sleep(a + b);
    }
}
```



```
|    Method | Add5 |   a |  b |
|---------- |----- |---- |--- |
| Benchmark | False | 100 | 10 |
| Benchmark | False | 100 | 20 |
| Benchmark | False | 200 | 10 |
| Benchmark | False | 200 | 20 |
| Benchmark |  True | 100 | 10 |
| Benchmark |  True | 100 | 20 |
| Benchmark |  True | 200 | 10 |
| Benchmark |  True | 200 | 20 |
```

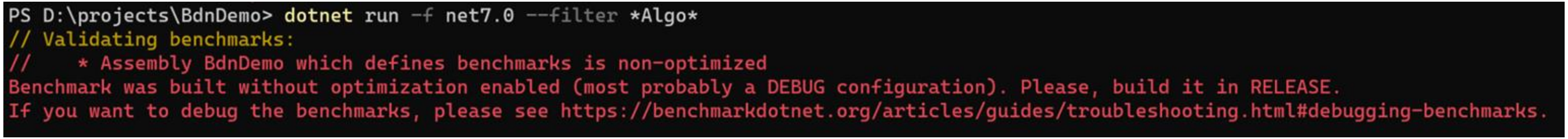
ArgumentsSource

```
public class IntroArgumentsSource
{
    [Benchmark]
    [ArgumentsSource(nameof(Numbers))]
    public double Pow(double x, double y)
            => Math.Pow(x, y);

    public IEnumerable<object[]> Numbers()
    {
        yield return new object[] { 1.0, 1.0 };
        yield return new object[] { 2.0, 2.0 };
        yield return new object[] { 4.0, 4.0 };
        yield return new object[] { 10.0, 10.0 };
    }
}
```



```
| Method |  x |  y |
|------- |--- |--- |-
|    Pow |  1 |  1 |
|    Pow |  2 |  2 |
|    Pow |  4 |  4 |
|    Pow | 10 | 10 |
```

Validators



```
PS D:\projects\BdnDemo> dotnet run -f net7.0 --filter *Algo*
// Validating benchmarks:
//    * Assembly BdnDemo which defines benchmarks is non-optimized
Benchmark was built without optimization enabled (most probably a DEBUG configuration). Please, build it in RELEASE.
If you want to debug the benchmarks, please see https://benchmarkdotnet.org/articles/guides/troubleshooting.html#debugging-benchmarks.
```

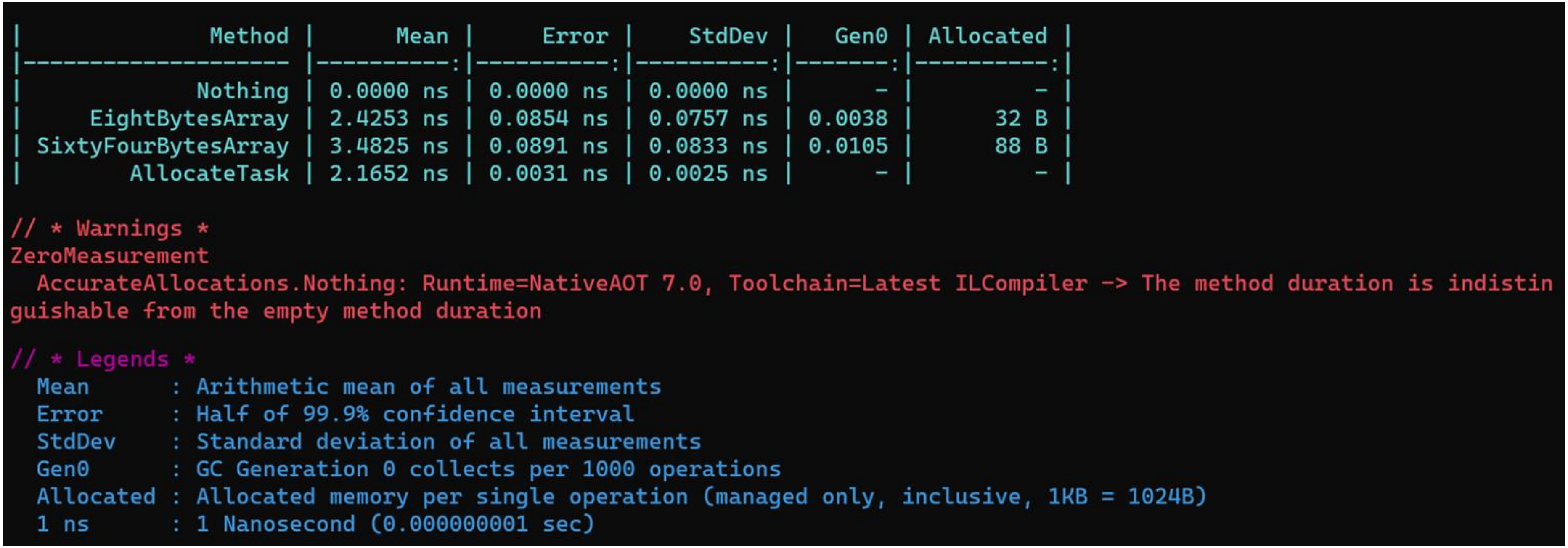
Diagnosers

- Plugins that allow to get some extra diagnostic information
- Can attach to the child process on specific events
- Few types: extra run / no overhead / separate logic

Memory Diagnoser: sample

```
[MemoryDiagnoser]
public class AccurateAllocations
{
    [Benchmark] public void Nothing() { }
    [Benchmark] public byte[] EightBytesArray() => new byte[8];
    [Benchmark] public byte[] SixtyFourBytesArray() => new byte[64];

    [Benchmark] public Task<int> AllocateTask()
                            => Task.FromResult(default(int));
}
```

Memory Diagnoser: results

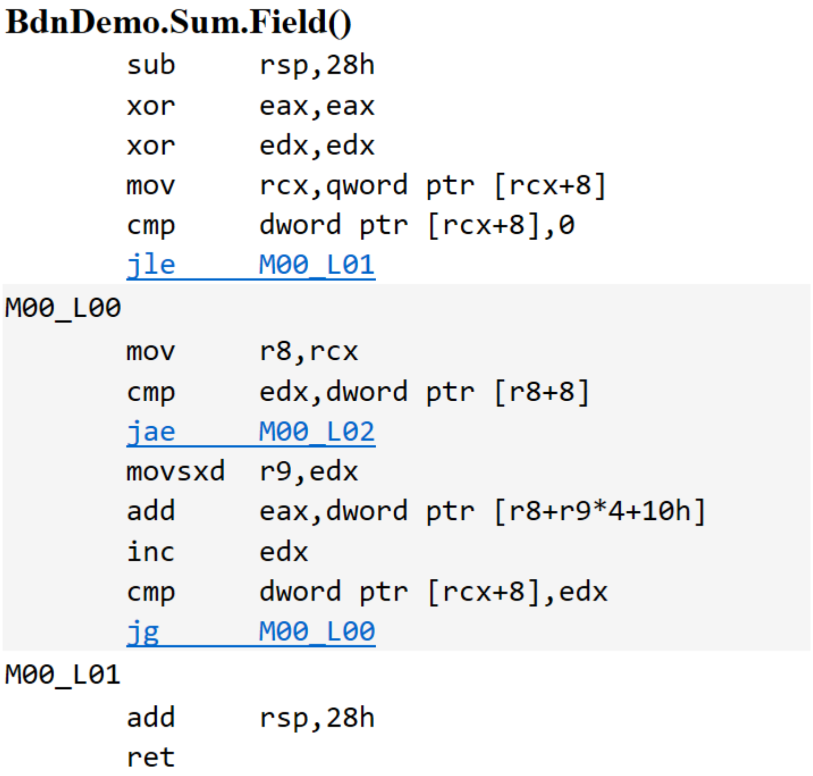


```
|             Method |      Mean |     Error |    StdDev |   Gen0 | Allocated |
|------------------- |----------:|----------:|----------:|-------:|----------:|
|            Nothing | 0.0000 ns | 0.0000 ns | 0.0000 ns |      - |         - |
|    EightBytesArray | 2.4253 ns | 0.0854 ns | 0.0757 ns | 0.0038 |      32 B |
| SixtyFourBytesArray | 3.4825 ns | 0.0891 ns | 0.0833 ns | 0.0105 |      88 B |
|       AllocateTask | 2.1652 ns | 0.0031 ns | 0.0025 ns |      - |         - |

// * Warnings *
ZeroMeasurement
  AccurateAllocations.Nothing: Runtime=NativeAOT 7.0, Toolchain=Latest ILCompiler -> The method duration is indistin
guishable from the empty method duration

// * Legends *
  Mean      : Arithmetic mean of all measurements
  Error     : Half of 99.9% confidence interval
  StdDev    : Standard deviation of all measurements
  Gen0      : GC Generation 0 collects per 1000 operations
  Allocated : Allocated memory per single operation (managed only, inclusive, 1KB = 1024B)
  1 ns      : 1 Nanosecond (0.000000001 sec)
```

Disassembly Diagnoser

- Attaches at the end (no ovehread)
- Uses ClrMD to get the ASM
- Works on Windows and Linux
- Supports x64, x86 and arm64
- Command line arguments:
  - -d/--disasm
  - --disasmDepth $recursiveDepth
  - --disasmFilter $glob

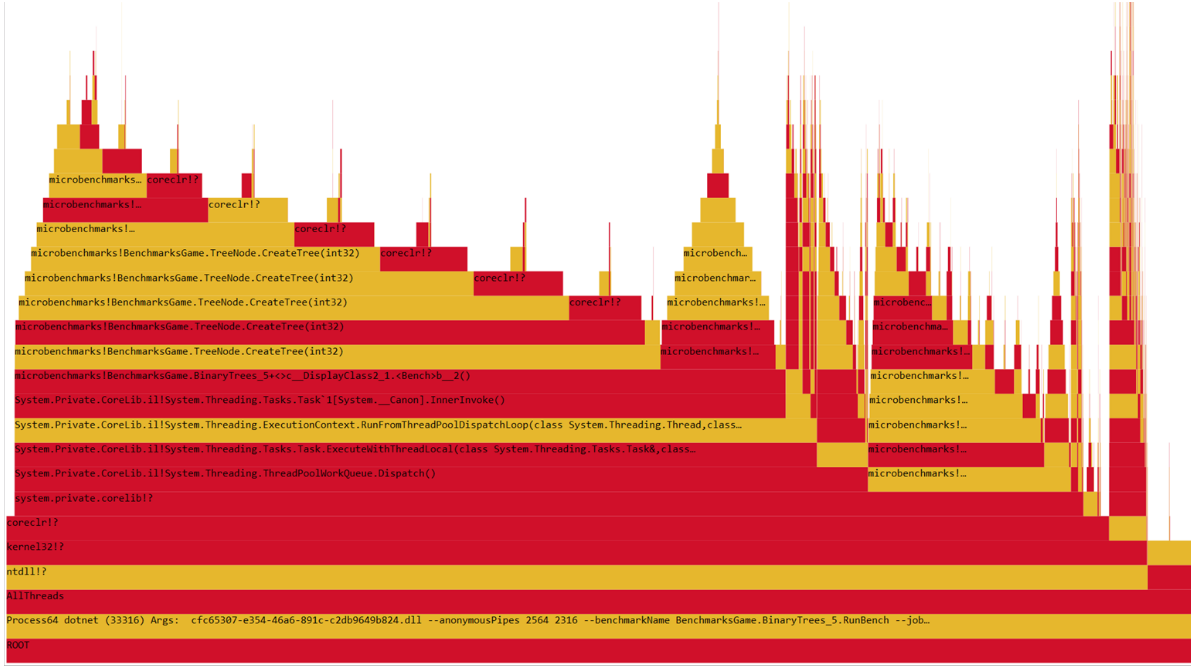
Sample HTML report



BdnDemo.Sum.Field()

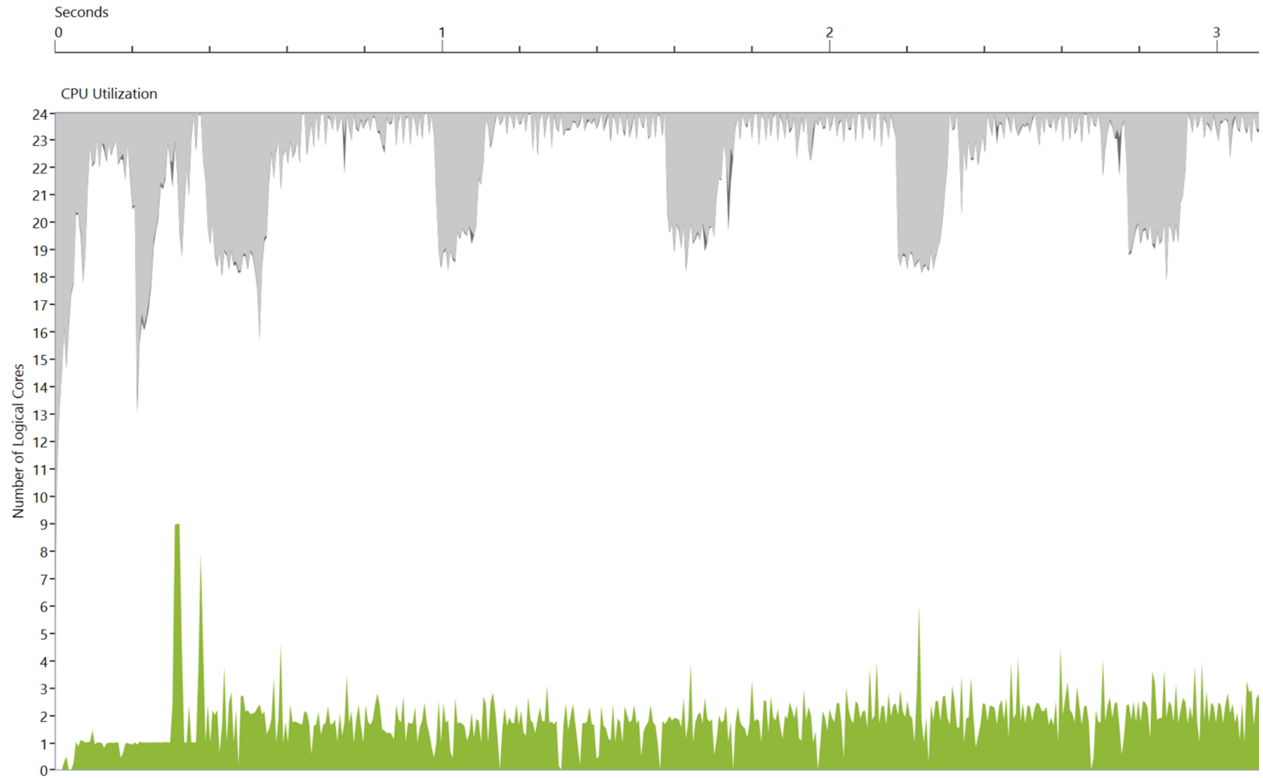
```
       sub     rsp,28h
       xor     eax,eax
       xor     edx,edx
       mov     rcx,qword ptr [rcx+8]
       cmp     dword ptr [rcx+8],0
       jle     M00_L01
M00_L00
       mov     r8,rcx
       cmp     edx,dword ptr [r8+8]
       jae     M00_L02
       movsxd  r9,edx
       add     eax,dword ptr [r8+r9*4+10h]
       inc     edx
       cmp     dword ptr [rcx+8],edx
       jg      M00_L00
M00_L01
       add     rsp,28h
       ret
```

ETW Profiler (--profiler ETW)



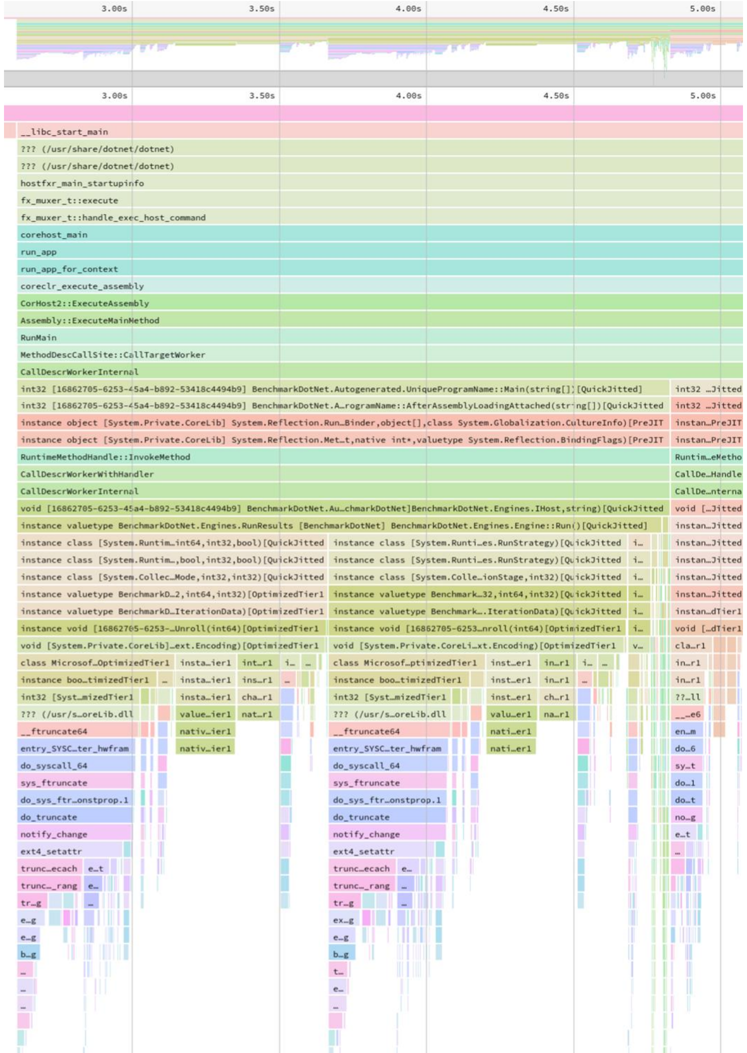
https://adamsitnik.com/ETW-Profiler/

Concurrency Visualiser (--profiler CV)



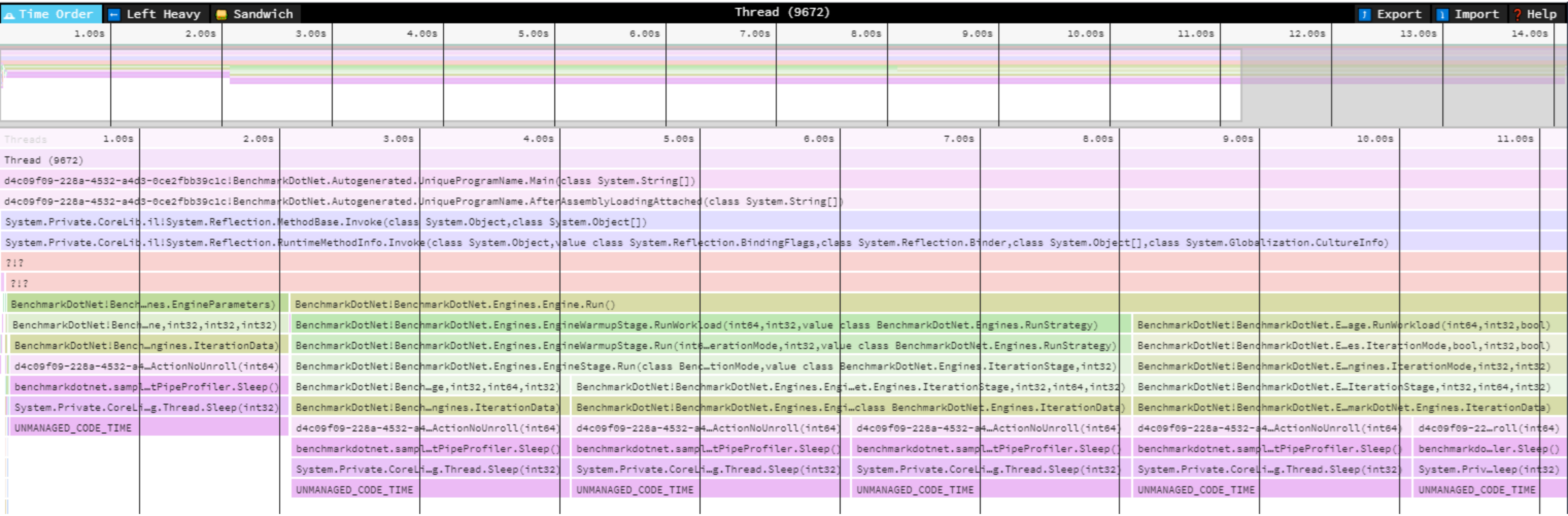
https://adamsitnik.com/ConcurrencyVisualizer-Profiler/

PerfCollect
(--profiler perf)



https://adamsitnik.com/PerfCollectProfiler/

EventPipe (--profiler EP)


Time Order
Left Heavy
Sandwich
Thread (9672)
Export
Import
Help
1.00s 2.00s 3.00s 4.00s 5.00s 6.00s 7.00s 8.00s 9.00s 10.00s 11.00s 12.00s 13.00s 14.00s
Threads 1.00s 2.00s 3.00s 4.00s 5.00s 6.00s 7.00s 8.00s 9.00s 10.00s 11.00s
Thread (9672)
d4c09f09-228a-4532-a4d3-0ce2fbb39c1c!BenchmarkDotNet.Autogenerated.UniqueProgramName.Main(class System.String[])
d4c09f09-228a-4532-a4d3-0ce2fbb39c1c!BenchmarkDotNet.Autogenerated.UniqueProgramName.AfterAssemblyLoadingAttached(class System.String[])
System.Private.CoreLib.il!System.Reflection.MethodBase.Invoke(class System.Object,class System.Object[])
System.Private.CoreLib.il!System.Reflection.RuntimeMethodInfo.Invoke(class System.Object,value class System.Reflection.BindingFlags,class System.Reflection.Binder,class System.Object[],class System.Globalization.CultureInfo)
?!?
?!?
BenchmarkDotNet!Bench…nes.EngineParameters)
BenchmarkDotNet!Bench…ne,int32,int32,int32)
BenchmarkDotNet!Bench…ngines.IterationData)
d4c09f09-228a-4532-a4…ActionNoUnroll(int64)
benchmarkdotnet.sampl…tPipeProfiler.Sleep()
System.Private.CoreLi…g.Thread.Sleep(int32)
UNMANAGED_CODE_TIME
BenchmarkDotNet!BenchmarkDotNet.Engines.Engine.Run()
BenchmarkDotNet!BenchmarkDotNet.Engines.EngineWarmupStage.RunWorkload(int64,int32,value class BenchmarkDotNet.Engines.RunStrategy)
BenchmarkDotNet!BenchmarkDotNet.Engines.EngineWarmupStage.Run(int…erationMode,int32,value class BenchmarkDotNet.Engines.RunStrategy)
BenchmarkDotNet!BenchmarkDotNet.Engines.EngineStage.Run(class Benc…tionMode,value class BenchmarkDotNet.Engines.IterationStage,int32)
BenchmarkDotNet!Bench…ge,int32,int64,int32)
BenchmarkDotNet!Bench…ngines.IterationData)
d4c09f09-228a-4532-a4…ActionNoUnroll(int64)
benchmarkdotnet.sampl…tPipeProfiler.Sleep()
System.Private.CoreLi…g.Thread.Sleep(int32)
UNMANAGED_CODE_TIME
BenchmarkDotNet!BenchmarkDotNet.Engines.Engi…et.Engines.IterationStage,int32,int64,int32)
BenchmarkDotNet!BenchmarkDotNet.Engines.Engi…class BenchmarkDotNet.Engines.IterationData)
d4c09f09-228a-4532-a4…ActionNoUnroll(int64)
benchmarkdotnet.sampl…tPipeProfiler.Sleep()
System.Private.CoreLi…g.Thread.Sleep(int32)
UNMANAGED_CODE_TIME
d4c09f09-228a-4532-a4…ActionNoUnroll(int64)
benchmarkdotnet.sampl…tPipeProfiler.Sleep()
System.Private.CoreLi…g.Thread.Sleep(int32)
UNMANAGED_CODE_TIME
BenchmarkDotNet!BenchmarkDotNet.E…age.RunWorkload(int64,int32,bool)
BenchmarkDotNet!BenchmarkDotNet.E…es.IterationMode,bool,int32,bool)
BenchmarkDotNet!BenchmarkDotNet.E…ngines.IterationMode,int32,int32)
BenchmarkDotNet!BenchmarkDotNet.E…IterationStage,int32,int64,int32)
BenchmarkDotNet!BenchmarkDotNet.E…markDotNet.Engines.IterationData)
d4c09f09-228a-4532-a4…ActionNoUnroll(int64)
benchmarkdotnet.sampl…tPipeProfiler.Sleep()
System.Private.CoreLi…g.Thread.Sleep(int32)
UNMANAGED_CODE_TIME
d4c09f09-22…roll(int64)
benchmarkdo…ler.Sleep()
System.Priv…leep(int32)
UNMANAGED_CODE_TIME

https://wojciechnagorski.com/2020/04/cross-platform-profiling-.net-code-with-benchmarkdotnet/

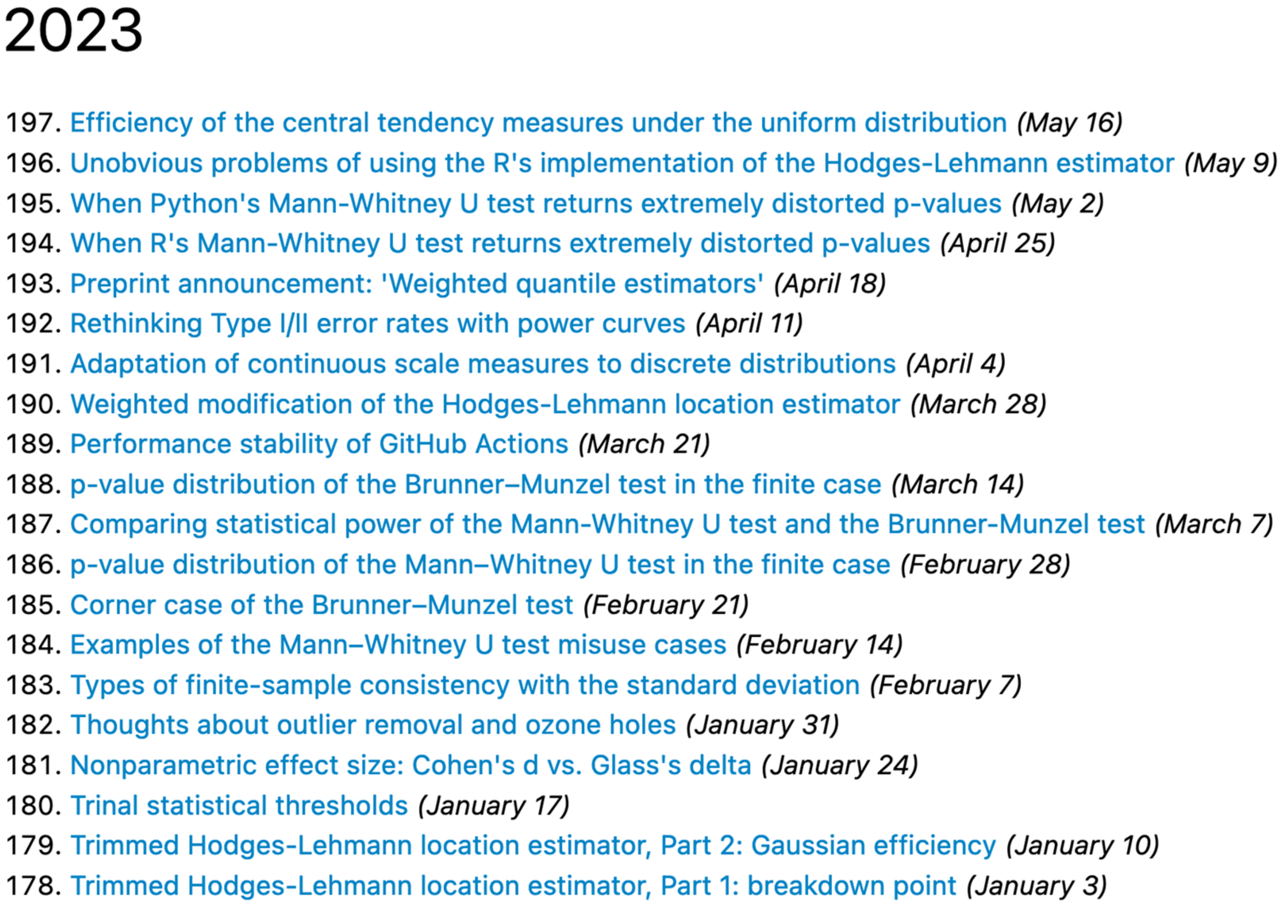
Other Diagnosers

- PmcDiagnoser – Hardware Counters on Windows
- NativeMemoryProfiler - allocated and leaking native memory (Windows)
- InliningDiagnoser uses ETW to get info about inlining
- TailCallDiagnoser uses ETW to get info about Tail Call opt
- ExceptionsDiagnoser - reports the frequency of exceptions
- ThreadingDiagnoser - threading statistics
- Architecture allows for more (like integration with profilers)

Statistics

Statistical challenges

- Summary tables
  - Should be reproducible
  - Should not be misleading
- Benchmark comparison
- Execution strategy

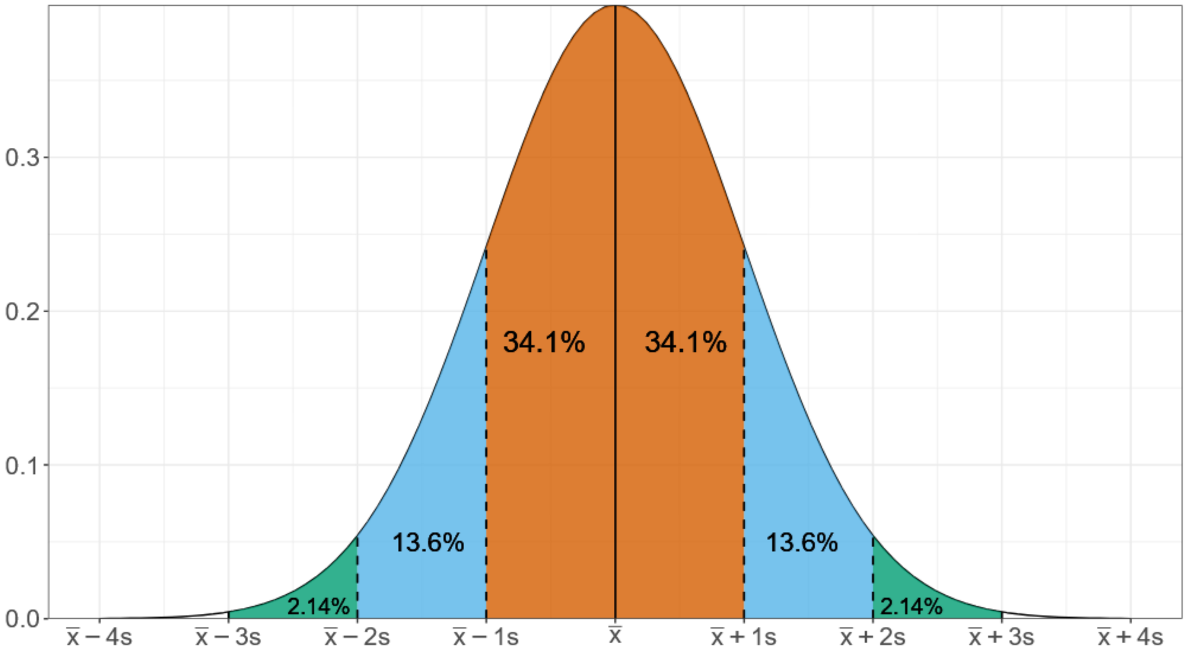


2023

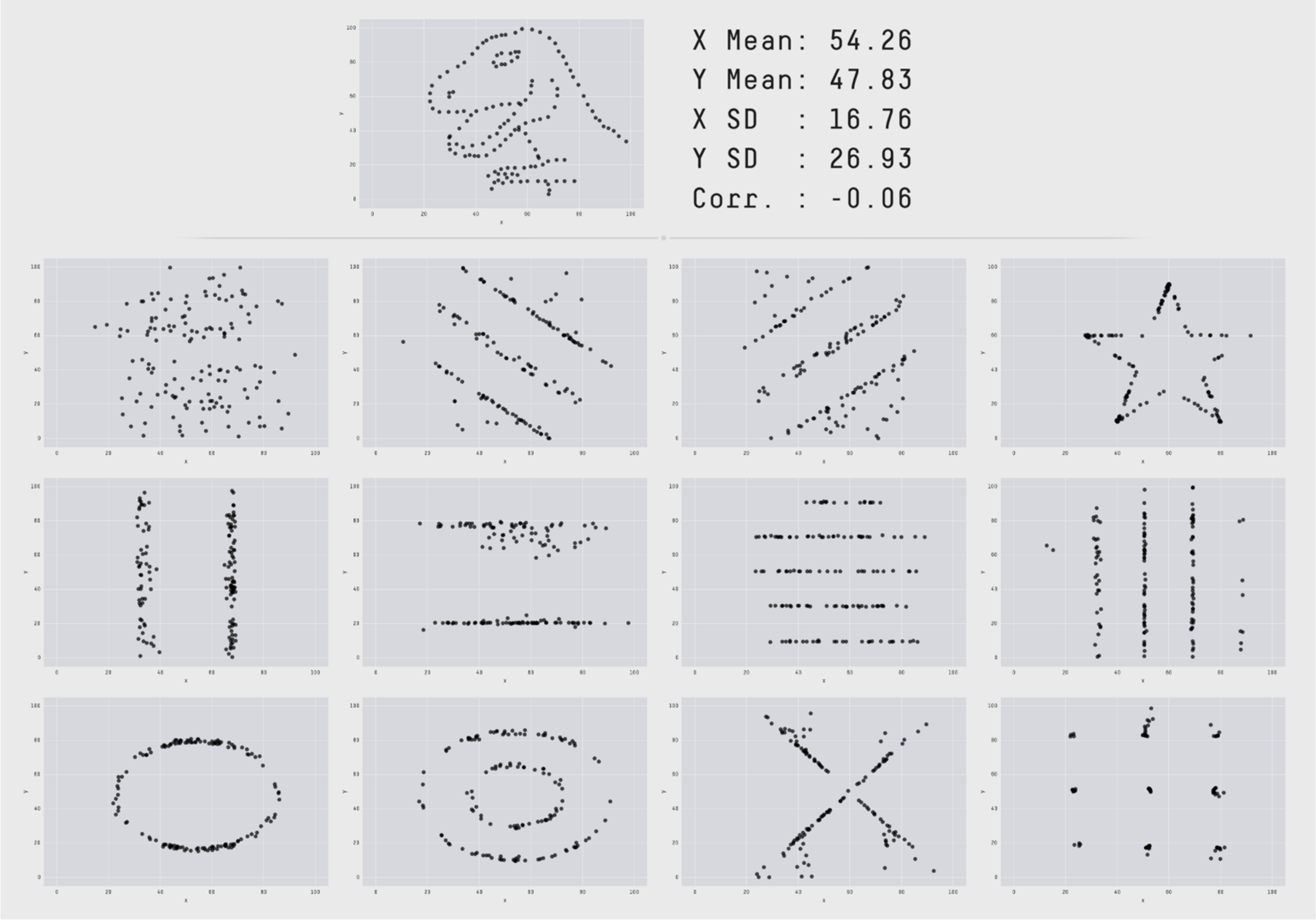
197. Efficiency of the central tendency measures under the uniform distribution *(May 16)*
196. Unobvious problems of using the R's implementation of the Hodges-Lehmann estimator *(May 9)*
195. When Python's Mann-Whitney U test returns extremely distorted p-values *(May 2)*
194. When R's Mann-Whitney U test returns extremely distorted p-values *(April 25)*
193. Preprint announcement: 'Weighted quantile estimators' *(April 18)*
192. Rethinking Type I/II error rates with power curves *(April 11)*
191. Adaptation of continuous scale measures to discrete distributions *(April 4)*
190. Weighted modification of the Hodges-Lehmann location estimator *(March 28)*
189. Performance stability of GitHub Actions *(March 21)*
188. p-value distribution of the Brunner–Munzel test in the finite case *(March 14)*
187. Comparing statistical power of the Mann-Whitney U test and the Brunner-Munzel test *(March 7)*
186. p-value distribution of the Mann–Whitney U test in the finite case *(February 28)*
185. Corner case of the Brunner–Munzel test *(February 21)*
184. Examples of the Mann–Whitney U test misuse cases *(February 14)*
183. Types of finite-sample consistency with the standard deviation *(February 7)*
182. Thoughts about outlier removal and ozone holes *(January 31)*
181. Nonparametric effect size: Cohen's d vs. Glass's delta *(January 24)*
180. Trinal statistical thresholds *(January 17)*
179. Trimmed Hodges-Lehmann location estimator, Part 2: Gaussian efficiency *(January 10)*
178. Trimmed Hodges-Lehmann location estimator, Part 1: breakdown point *(January 3)*

https://aakinshin.net/tags/statistics/

The Normal distribution



> *Normality is a myth; there never was, and never will be, a normal distribution.*
> — "Testing for normality", R.C. Geary, 1947



X Mean: 54.26

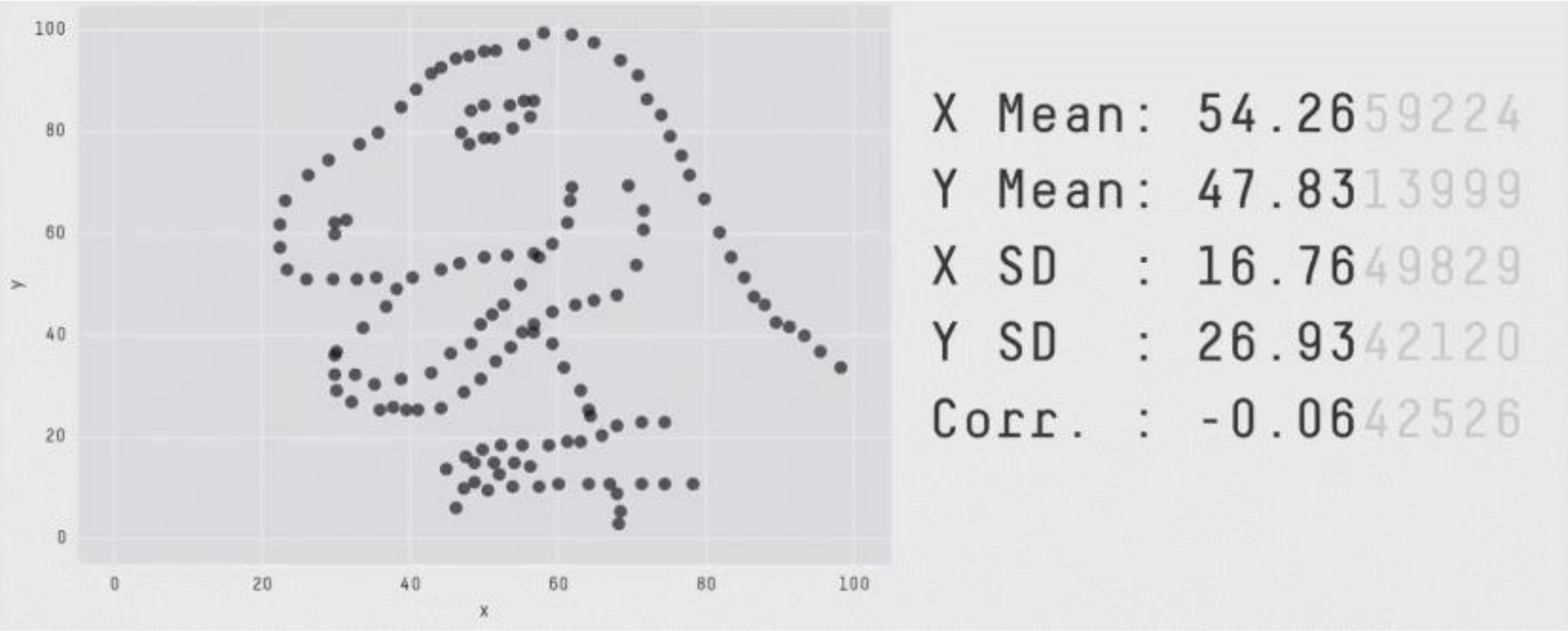
Y Mean: 47.83

X SD : 16.76

Y SD : 26.93

Corr. : -0.06

Justin Matejka, George Fitzmaurice (2017), "Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing", CHI 2017 Conference proceedings: ACM SIGCHI Conference on Human Factors in Computing Systems



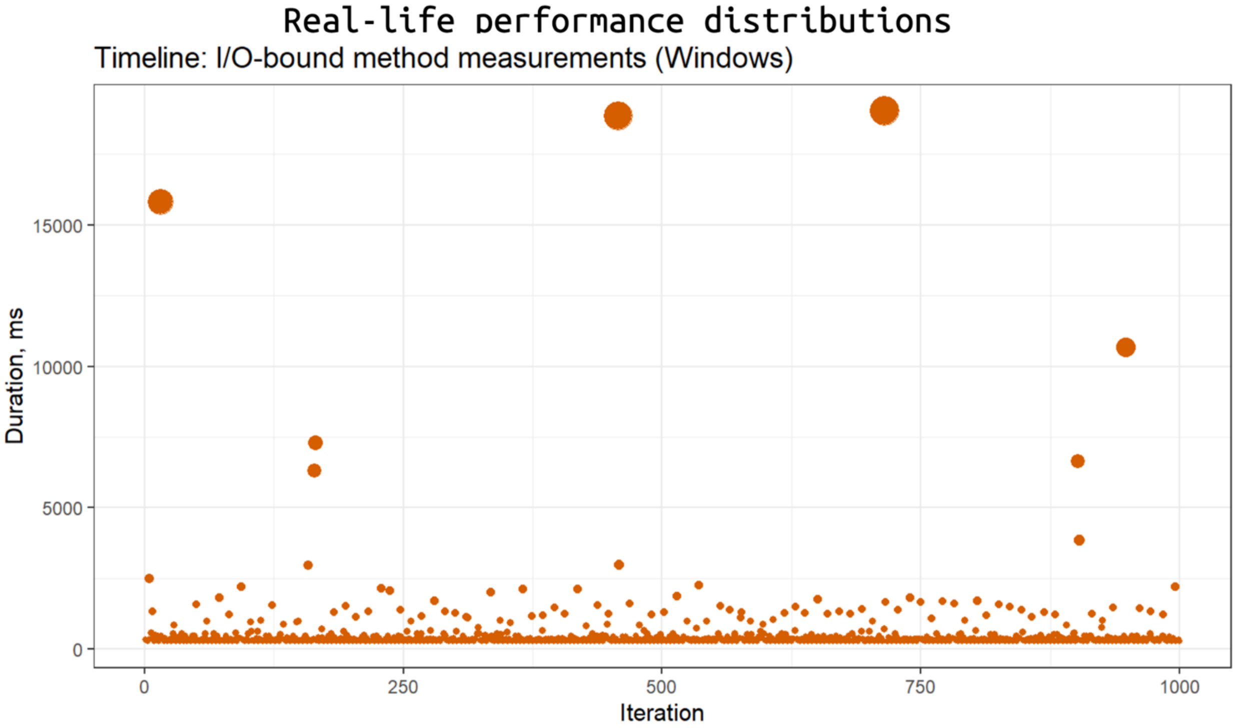
```
X Mean: 54.2659224
Y Mean: 47.8313999
X SD  : 16.7649829
Y SD  : 26.9342120
Corr. : -0.0642526
```

Justin Matejka, George Fitzmaurice (2017), "Same Stats, Different Graphs: Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing", CHI 2017 Conference proceedings: ACM SIGCHI Conference on Human Factors in Computing Systems

IO-Benchmark example

```
int N = 1000; // The number of iterations
var measurements = new long[N];
byte[] data = new byte[64 * 1024 * 1024]; // 64MB

for (int i = 0; i < N; i++)
{
    var stopwatch = Stopwatch.StartNew();
    var fileName = Path.GetTempFileName();
    File.WriteAllBytes(fileName, data);
    File.Delete(fileName);
    stopwatch.Stop();
    measurements[i] = stopwatch.ElapsedMilliseconds;
}
```

IO-Benchmark example: results



Distribution features to support

- Huge dispersion
- Extreme outliers
- Multimodality
- Asymmetry
- Discretization

Central tendency / The mean

$$x = \{x_1, x_2, \ldots, x_n\}; \quad \overline{x} = \frac{x_1 + x_2 + \ldots + x_n}{n}$$

$$x = \{1, 2, 3, 4, 5, 6, 7\}; \quad \overline{x} = 4$$

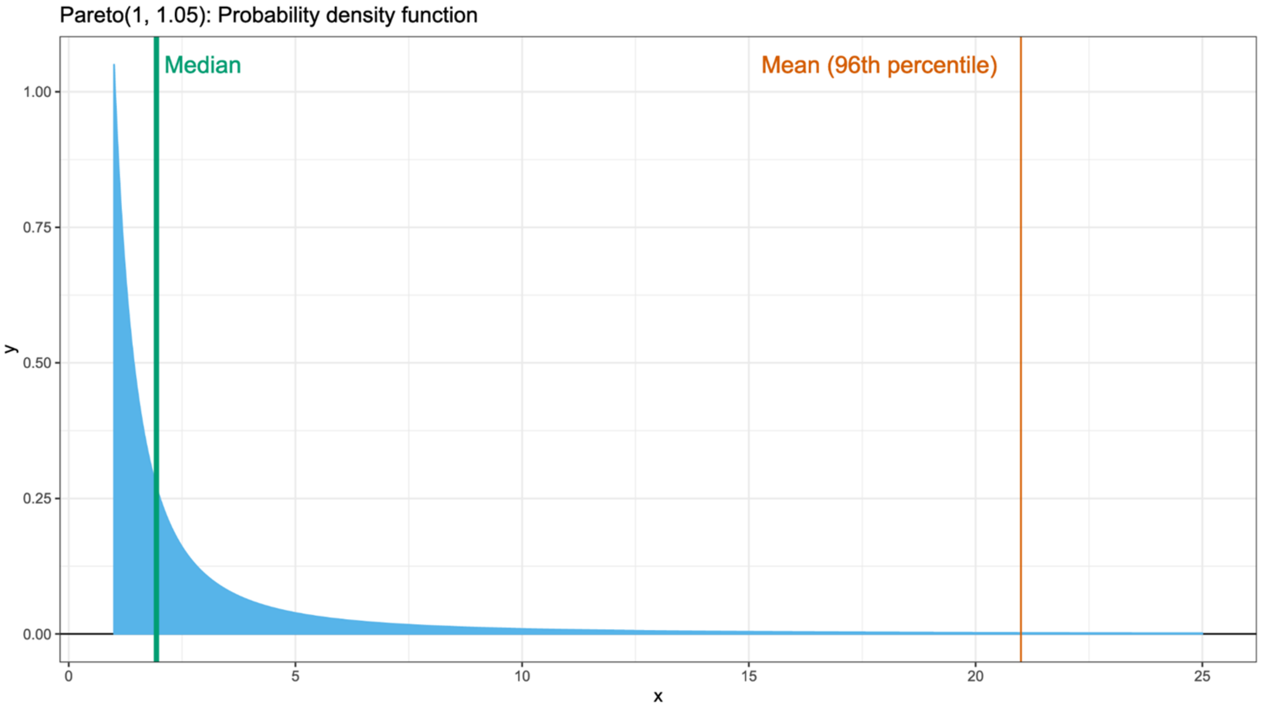
$$x = \{1, 2, 3, 4, 5, 6, 273\}; \quad \overline{x} = 42$$

Central tendency / The median

$$x = \{1, 2, 3, 4, 5, 6, 7\}; \quad \tilde{x} = 4$$

$$x = \{1, 2, 3, 4, 5, 6, 273\}; \quad \tilde{x} = 4$$

Central tendency



Central tendency

	Mean	Median
Gaussian efficiency	100%	≈64%

Central tendency

	Mean	Median
Gaussian efficiency	100%	≈64%
Cost	100€	**157€**

Central tendency / Hodges-Lehmann

$$\mathrm{HL} = \underset{i \le j}{\mathrm{median}} \left(\frac{x_i + x_j}{2} \right)$$

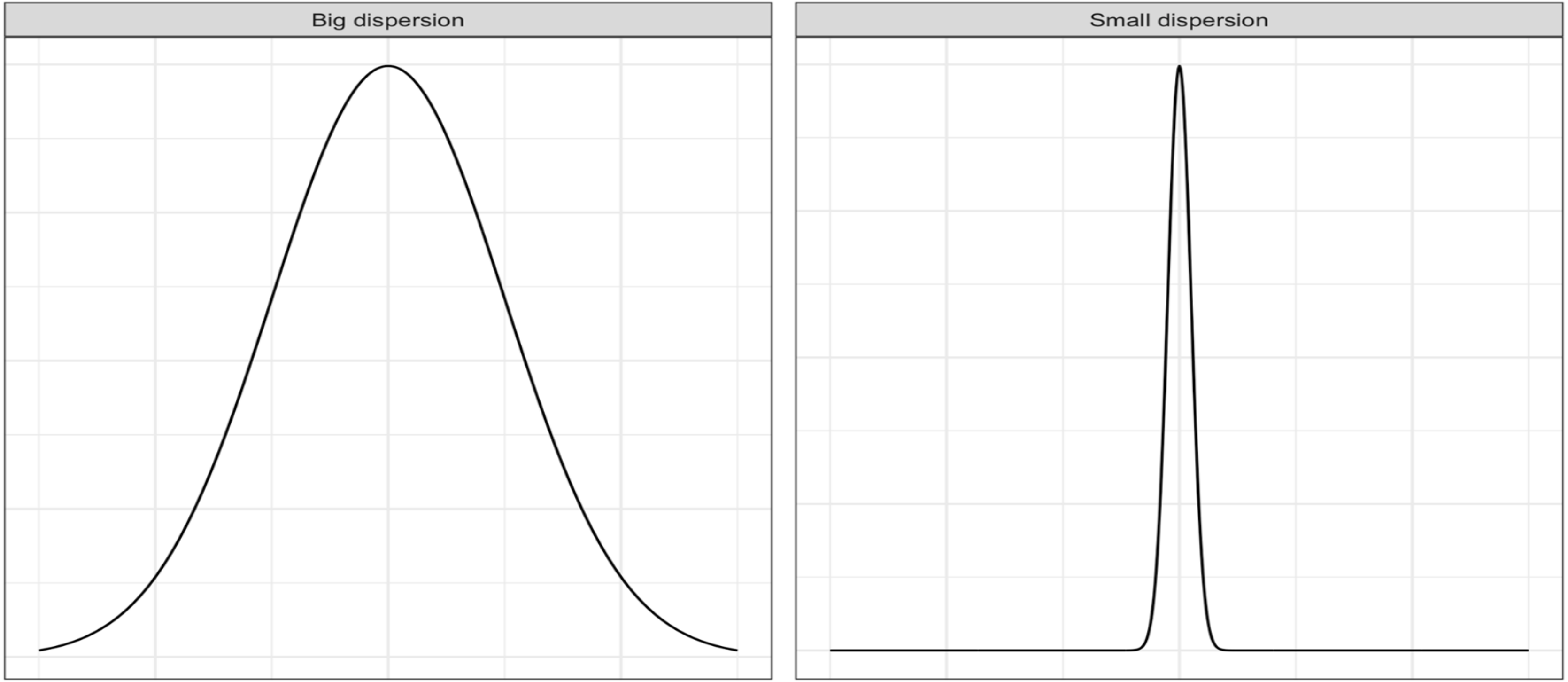
Central tendency

	Mean	Median	HL
Gaussian efficiency	100%	≈64%	≈96%
Cost	100€	157€	**104€**

Central tendency / Other estimators

- Trimmed mean
- Winsorized mean
- Mode
- Geometric mean
- Harmonic mean
- Interquartile mean
- Midrange
- Midhinge
- Trimean
- Huber M-estimator
- Andrews M-estimator
- Hampel M-estimator
- Biweight M-estimator
- …

Dispersion



Dispersion / Standard deviation

$$\boldsymbol{s} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Dispersion / Median absolute deviation

$$\mathrm{MAD} = C_n \cdot \mathbf{median}\Big(\left|x_i - \mathbf{median}(x)\right|\Big)$$

Dispersion / Other estimators

Shamos estimator

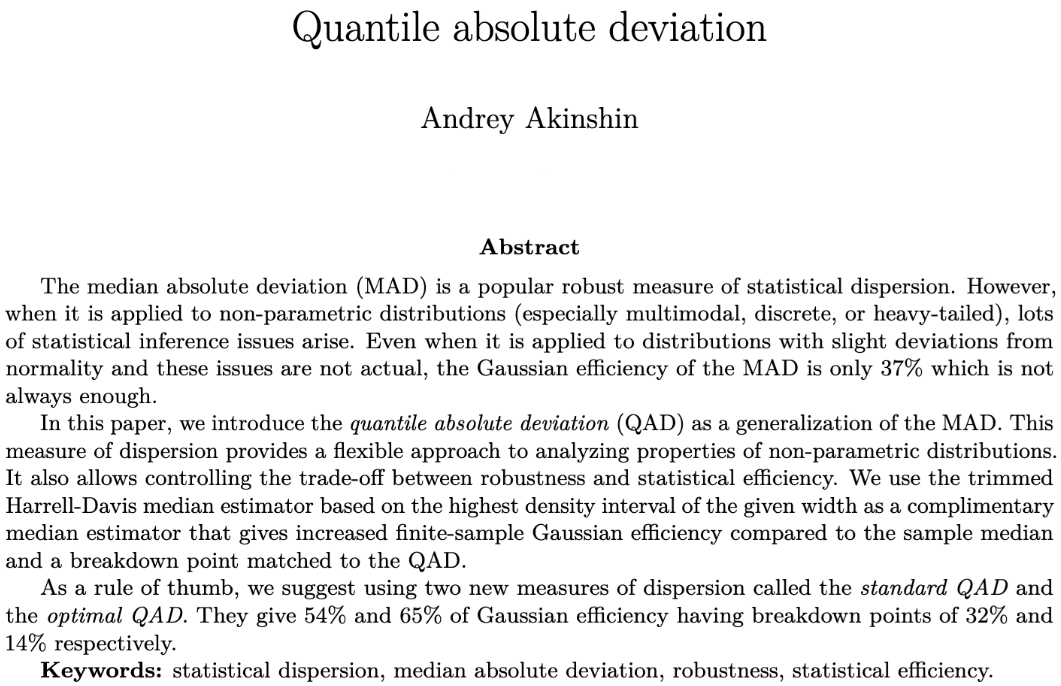
$$\text{Shamos} = C_n \cdot \text{median}(|x_i - x_j|_{i<j}); \quad C_\infty \approx 1.048358$$

Rousseeuw-Croux estimator

$$S_n = C_n \cdot \text{median}_i\Big(\text{median}_j(|x_i - x_j|)\Big); \quad C_\infty \approx 1.1926$$

$$Q_n = C_n \cdot Q(|x_i - x_j|_{i<j}, 0.25); \quad C_\infty \approx 2.2191$$

Dispersion / Quantile absolute deviation



Quantile absolute deviation

Andrey Akinshin

Abstract

The median absolute deviation (MAD) is a popular robust measure of statistical dispersion. However, when it is applied to non-parametric distributions (especially multimodal, discrete, or heavy-tailed), lots of statistical inference issues arise. Even when it is applied to distributions with slight deviations from normality and these issues are not actual, the Gaussian efficiency of the MAD is only 37% which is not always enough.

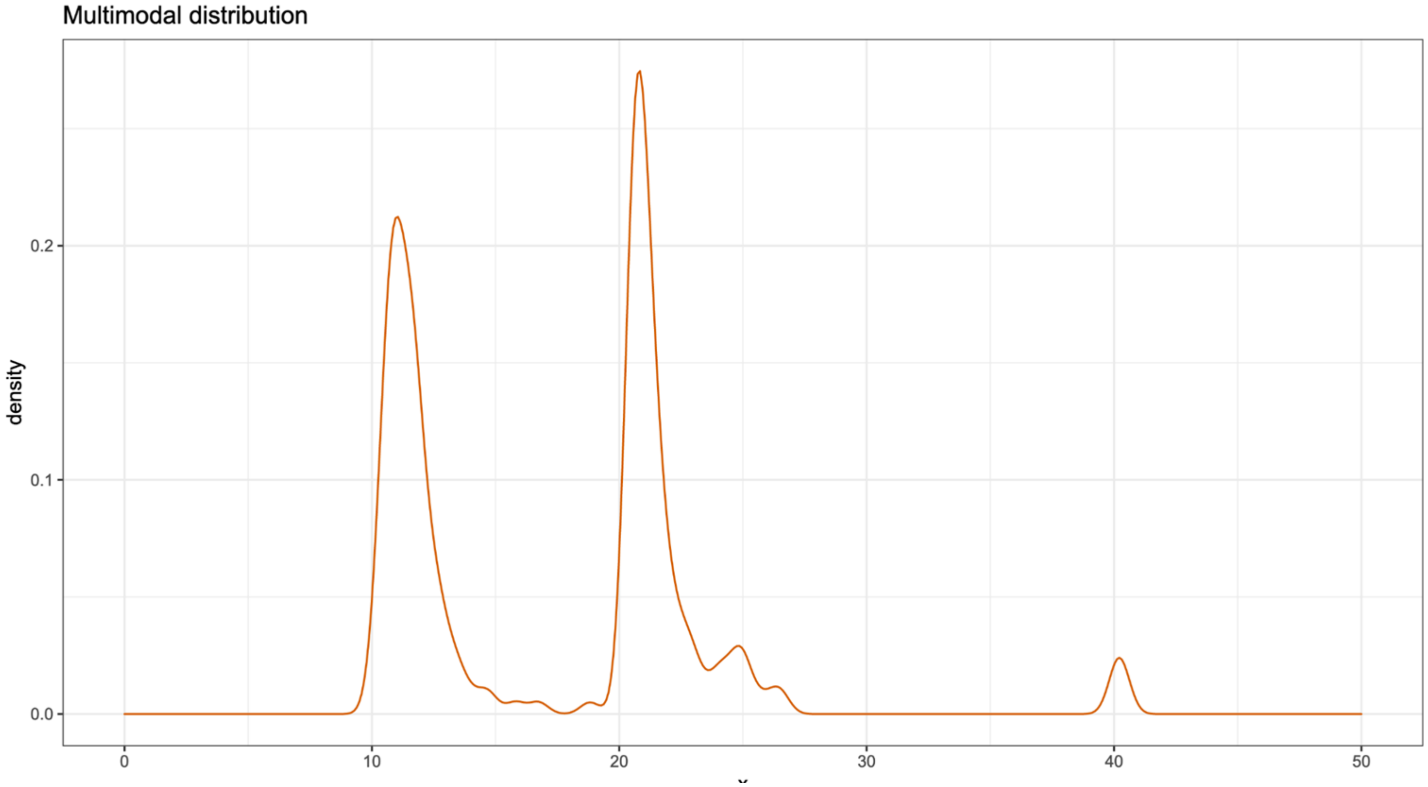
In this paper, we introduce the *quantile absolute deviation* (QAD) as a generalization of the MAD. This measure of dispersion provides a flexible approach to analyzing properties of non-parametric distributions. It also allows controlling the trade-off between robustness and statistical efficiency. We use the trimmed Harrell-Davis median estimator based on the highest density interval of the given width as a complimentary median estimator that gives increased finite-sample Gaussian efficiency compared to the sample median and a breakdown point matched to the QAD.

As a rule of thumb, we suggest using two new measures of dispersion called the *standard QAD* and the *optimal QAD*. They give 54% and 65% of Gaussian efficiency having breakdown points of 32% and 14% respectively.

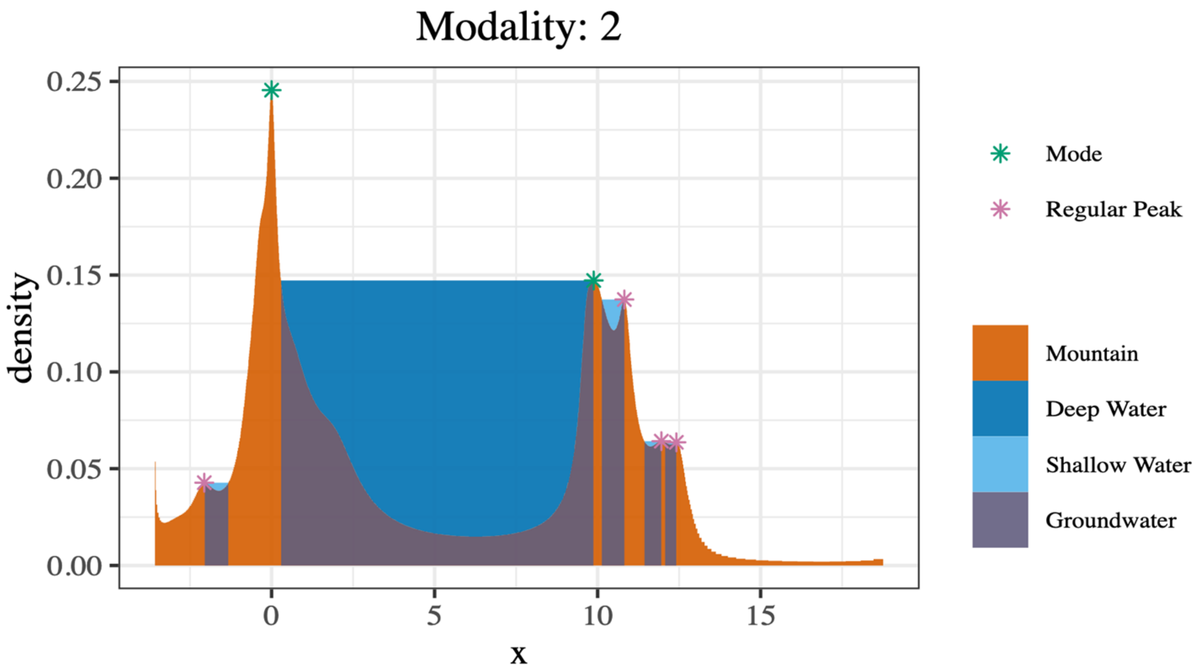
Keywords: statistical dispersion, median absolute deviation, robustness, statistical efficiency.

https://arxiv.org/abs/2208.13459

Real-life distributions

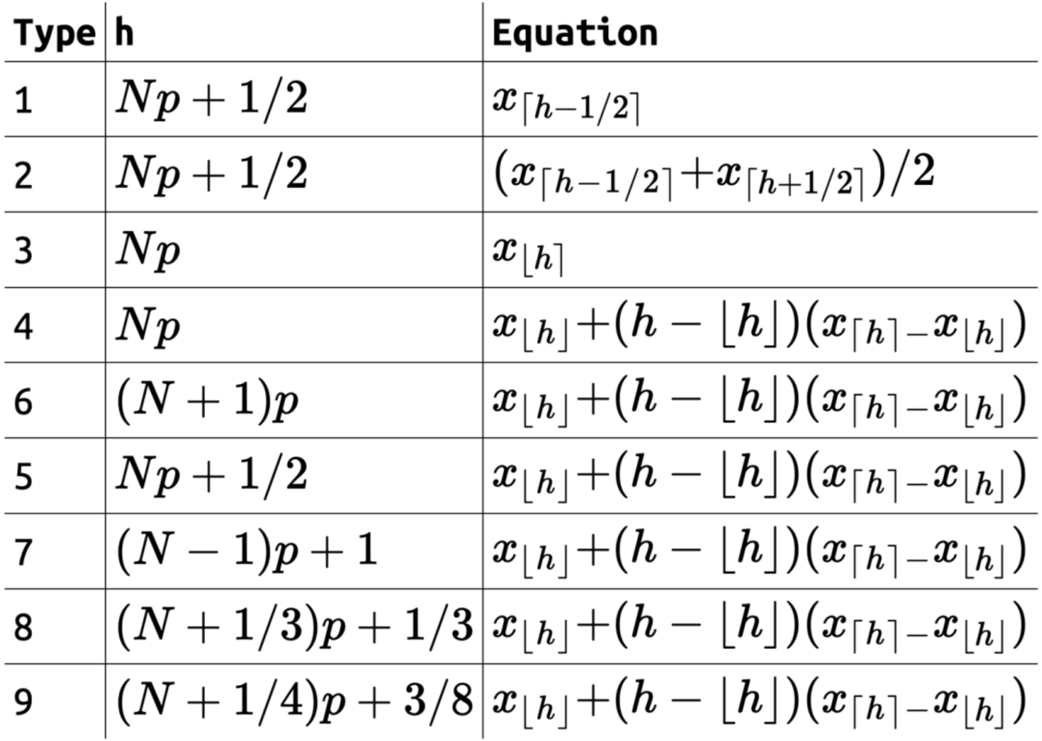


Multimodality



https://aakinshin.net/posts/lowland-multimodality-detection/

Quantile estimations / “Traditional”



Type	h	Equation
1	$Np+1/2$	$x_{\lceil h-1/2\rceil}$
2	$Np+1/2$	$(x_{\lceil h-1/2\rceil}+x_{\lceil h+1/2\rceil})/2$
3	Np	$x_{\lfloor h\rceil}$
4	Np	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$
6	$(N+1)p$	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$
5	$Np+1/2$	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$
7	$(N-1)p+1$	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$
8	$(N+1/3)p+1/3$	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$
9	$(N+1/4)p+3/8$	$x_{\lfloor h\rfloor}+(h-\lfloor h\rfloor)(x_{\lceil h\rceil}-x_{\lfloor h\rfloor})$

https://doi.org/10.2307/2684934

Quantile estimations / Harrell-Davis

$$Q_{HD}(p) = \sum_{i=1}^{n} W_i \cdot x_{(i)}$$

$$W_i = I_{i/n}(a, b) - I_{(i-1)/n}(a, b)$$

$$a = p(n+1),\ b = (1-p)(n+1)$$

https://dx.doi.org/10.1093/biomet/69.3.635

Quantile estimations / Sfakianakis-Verginis

$$
\begin{aligned}
\mathrm{SV1}_p = {} & \frac{B_0}{2}\left(X_{(1)} + X_{(2)} - X_{(3)}\right) + \\
& \sum\nolimits_{i=1}^{n} \frac{B_i + B_{i-1}}{2} X_{(i)} + \\
& \frac{B_n}{2}\left(-X_{(n-2)} + X_{(n-1)} - X_{(n)}\right), \\
\mathrm{SV2}_p = {} & \sum\nolimits_{i=1}^{n} B_{i-1} X_{(i)} + B_n \cdot \left(2X_{(n)} - X_{(n-1)}\right), \\
\mathrm{SV3}_p = {} & \sum\nolimits_{i=1}^{n} B_i X_{(i)} + B_0 \cdot \left(2X_{(1)} - X_{(2)}\right).
\end{aligned}
$$

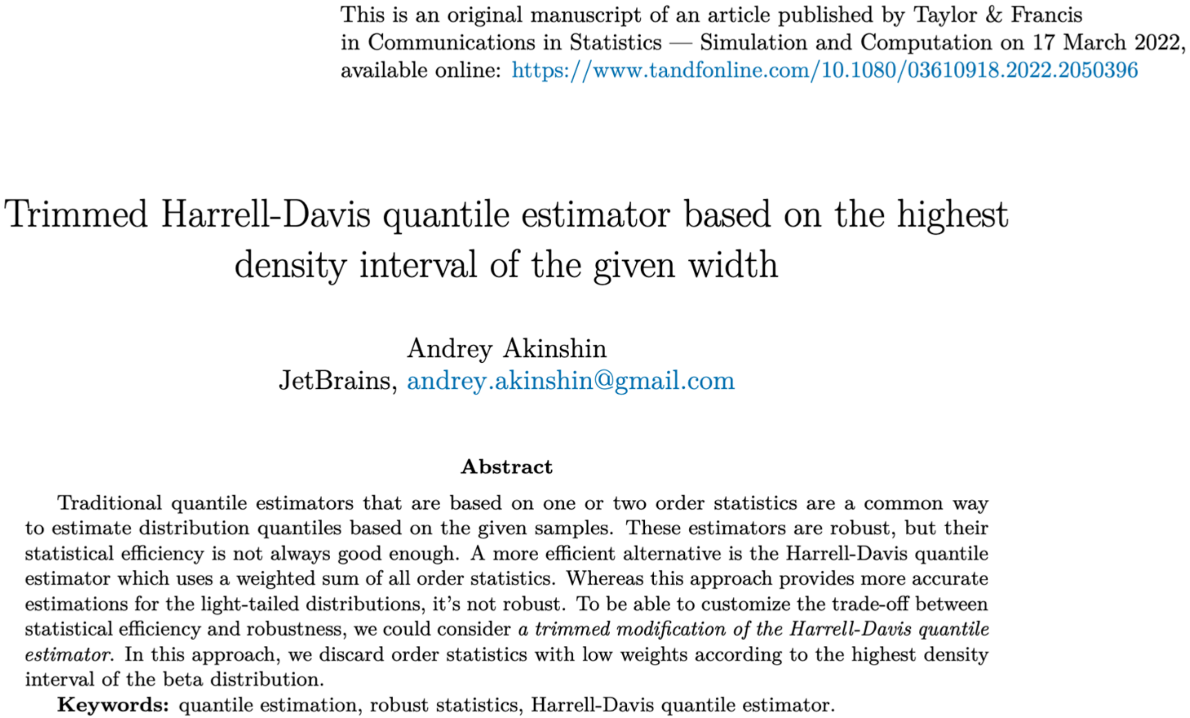
https://doi.org/10.1080/03610910701790491

Quantile estimations / Navruz-Özdemir

$$\begin{aligned}\mathrm{NO}_p = &\Big((3p-1)X_{(1)} + (2-3p)X_{(2)} - (1-p)X_{(3)}\Big)B_0 + \\ &+ \sum\nolimits_{i=1}^{n}\Big((1-p)B_{i-1} + pB_i\Big)X_{(i)} + \\ &+ \Big(-pX_{(n-2)} + (3p-1)X_{(n-1)} + (2-3p)X_{(n)}\Big)B_n\end{aligned}$$

https://doi.org/10.1111/bmsp.12198

Quantile estimations / Trimmed Harrell-Davis



This is an original manuscript of an article published by Taylor & Francis
in Communications in Statistics — Simulation and Computation on 17 March 2022,
available online: https://www.tandfonline.com/10.1080/03610918.2022.2050396

Trimmed Harrell-Davis quantile estimator based on the highest density interval of the given width

Andrey Akinshin
JetBrains, andrey.akinshin@gmail.com

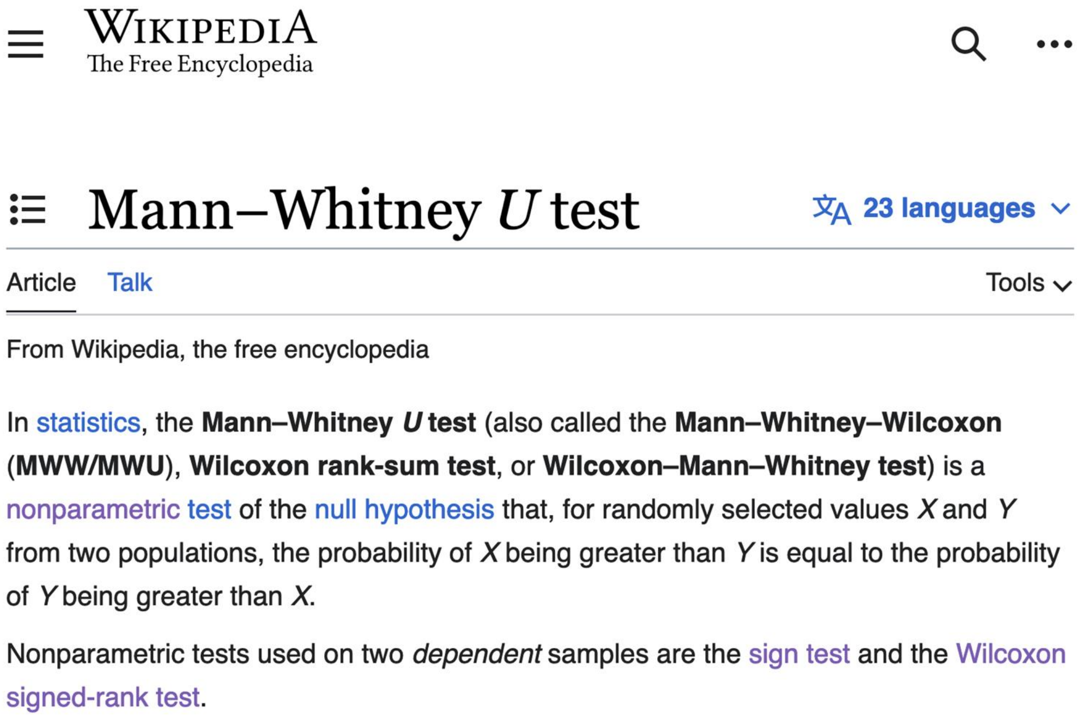
Abstract

Traditional quantile estimators that are based on one or two order statistics are a common way to estimate distribution quantiles based on the given samples. These estimators are robust, but their statistical efficiency is not always good enough. A more efficient alternative is the Harrell-Davis quantile estimator which uses a weighted sum of all order statistics. Whereas this approach provides more accurate estimations for the light-tailed distributions, it's not robust. To be able to customize the trade-off between statistical efficiency and robustness, we could consider *a trimmed modification of the Harrell-Davis quantile estimator*. In this approach, we discard order statistics with low weights according to the highest density interval of the beta distribution.

Keywords: quantile estimation, robust statistics, Harrell-Davis quantile estimator.

https://arxiv.org/abs/2111.11776

Mann-Whitney U-test



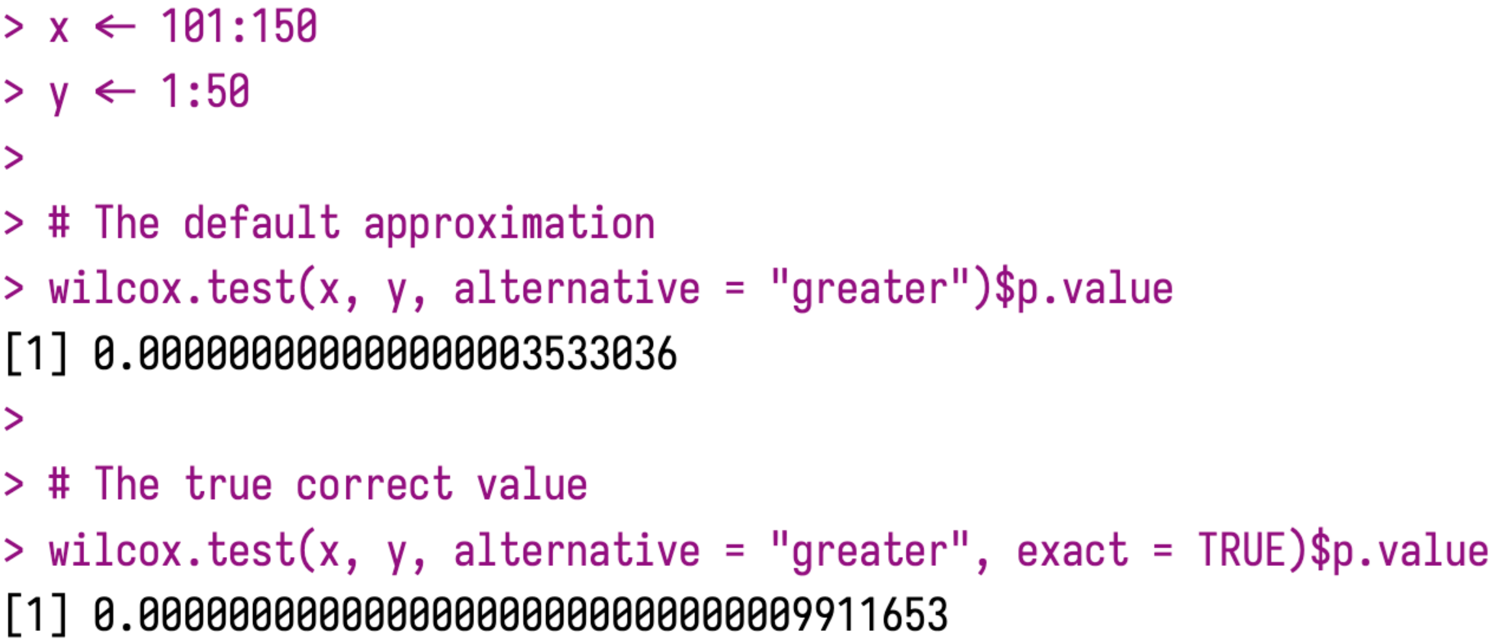
Mann–Whitney *U* test

From Wikipedia, the free encyclopedia

In statistics, the **Mann–Whitney *U* test** (also called the **Mann–Whitney–Wilcoxon** (**MWW/MWU**), **Wilcoxon rank-sum test**, or **Wilcoxon–Mann–Whitney test**) is a nonparametric test of the null hypothesis that, for randomly selected values *X* and *Y* from two populations, the probability of *X* being greater than *Y* is equal to the probability of *Y* being greater than *X*.

Nonparametric tests used on two *dependent* samples are the sign test and the Wilcoxon signed-rank test.

Mann-Whitney U-test

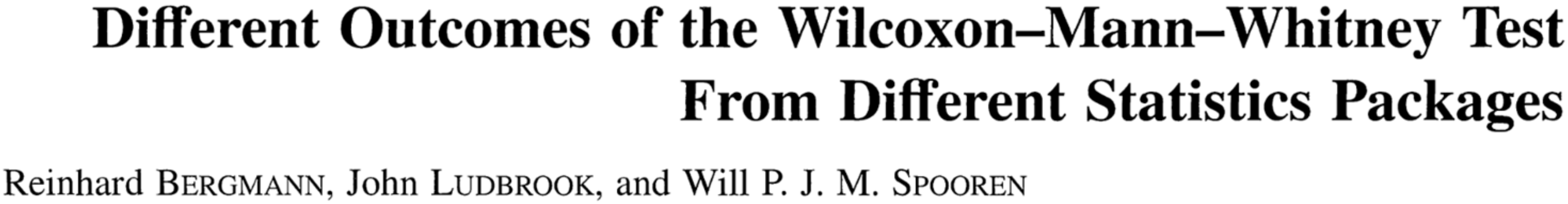


```
> x ← 101:150
> y ← 1:50
>
> # The default approximation
> wilcox.test(x, y, alternative = "greater")$p.value
[1] 0.000000000000000003533036
>
> # The true correct value
> wilcox.test(x, y, alternative = "greater", exact = TRUE)$p.value
[1] 0.000000000000000000000000000009911653
```

Difference: 356’452’748’856 times!!!

https://aakinshin.net/posts/r-mann-whitney-incorrect-p-value/

Mann-Whitney U-test

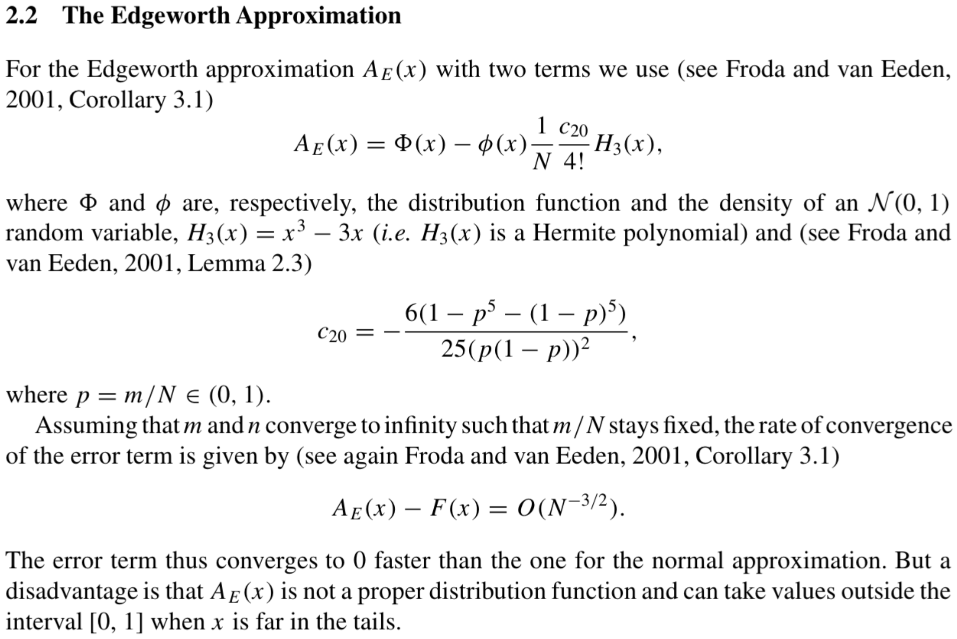


Different Outcomes of the Wilcoxon–Mann–Whitney Test From Different Statistics Packages

Reinhard BERGMANN, John LUDBROOK, and Will P. J. M. SPOOREN

https://www.jstor.org/stable/2685616

Mann-Whitney U-test



2.2 The Edgeworth Approximation

For the Edgeworth approximation $A_E(x)$ with two terms we use (see Froda and van Eeden, 2001, Corollary 3.1)

$$A_E(x) = \Phi(x) - \phi(x)\frac{1}{N}\frac{c_{20}}{4!}H_3(x),$$

where Φ and ϕ are, respectively, the distribution function and the density of an $\mathcal{N}(0,1)$ random variable, $H_3(x) = x^3 - 3x$ (*i.e.* $H_3(x)$ is a Hermite polynomial) and (see Froda and van Eeden, 2001, Lemma 2.3)

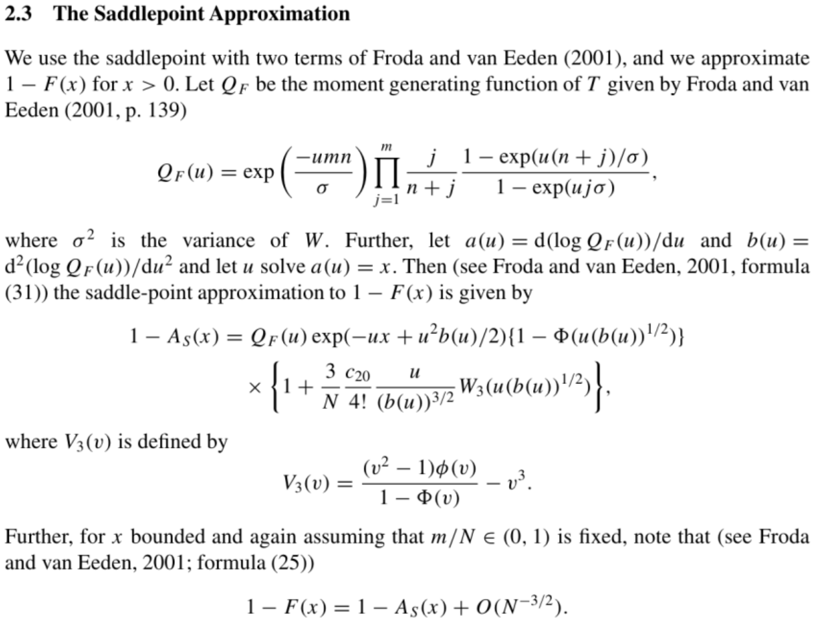
$$c_{20} = -\frac{6(1 - p^5 - (1-p)^5)}{25(p(1-p))^2},$$

where $p = m/N \in (0, 1)$.

Assuming that m and n converge to infinity such that m/N stays fixed, the rate of convergence of the error term is given by (see again Froda and van Eeden, 2001, Corollary 3.1)

$$A_E(x) - F(x) = O(N^{-3/2}).$$

The error term thus converges to 0 faster than the one for the normal approximation. But a disadvantage is that $A_E(x)$ is not a proper distribution function and can take values outside the interval [0, 1] when x is far in the tails.



2.3 The Saddlepoint Approximation

We use the saddlepoint with two terms of Froda and van Eeden (2001), and we approximate $1 - F(x)$ for $x > 0$. Let Q_F be the moment generating function of T given by Froda and van Eeden (2001, p. 139)

$$Q_F(u) = \exp\left(\frac{-umn}{\sigma}\right)\prod_{j=1}^{m}\frac{j}{n+j}\,\frac{1 - \exp(u(n+j)/\sigma)}{1 - \exp(uj\sigma)},$$

where σ^2 is the variance of W. Further, let $a(u) = \mathrm{d}(\log Q_F(u))/\mathrm{d}u$ and $b(u) = \mathrm{d}^2(\log Q_F(u))/\mathrm{d}u^2$ and let u solve $a(u) = x$. Then (see Froda and van Eeden, 2001, formula (31)) the saddle-point approximation to $1 - F(x)$ is given by

$$\begin{aligned}1 - A_S(x) &= Q_F(u)\exp(-ux + u^2b(u)/2)\{1 - \Phi(u(b(u))^{1/2})\}\\ &\quad\times\left\{1 + \frac{3}{N}\frac{c_{20}}{4!}\frac{u}{(b(u))^{3/2}}W_3(u(b(u))^{1/2})\right\},\end{aligned}$$

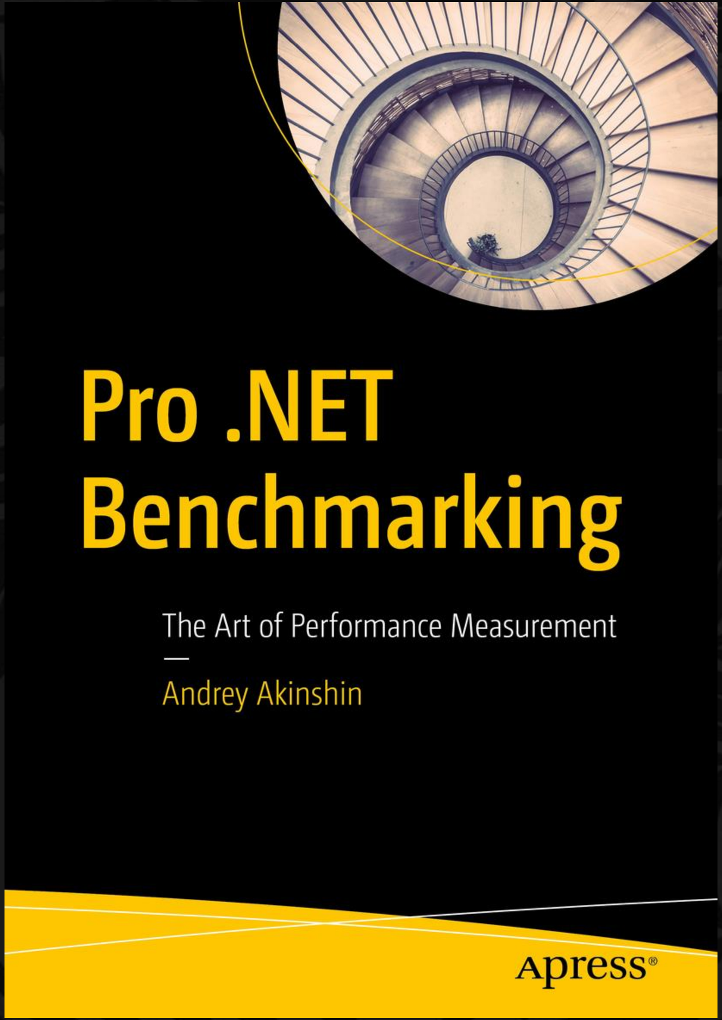
where $V_3(v)$ is defined by

$$V_3(v) = \frac{(v^2 - 1)\phi(v)}{1 - \Phi(v)} - v^3.$$

Further, for x bounded and again assuming that $m/N \in (0, 1)$ is fixed, note that (see Froda and van Eeden, 2001; formula (25))

$$1 - F(x) = 1 - A_S(x) + O(N^{-3/2}).$$

https://dx.doi.org/10.1080/10485250310001622677



Pro .NET Benchmarking

The Art of Performance Measurement

Andrey Akinshin

Apress®

Roadmap to BenchmarkDotNet 1.0.0

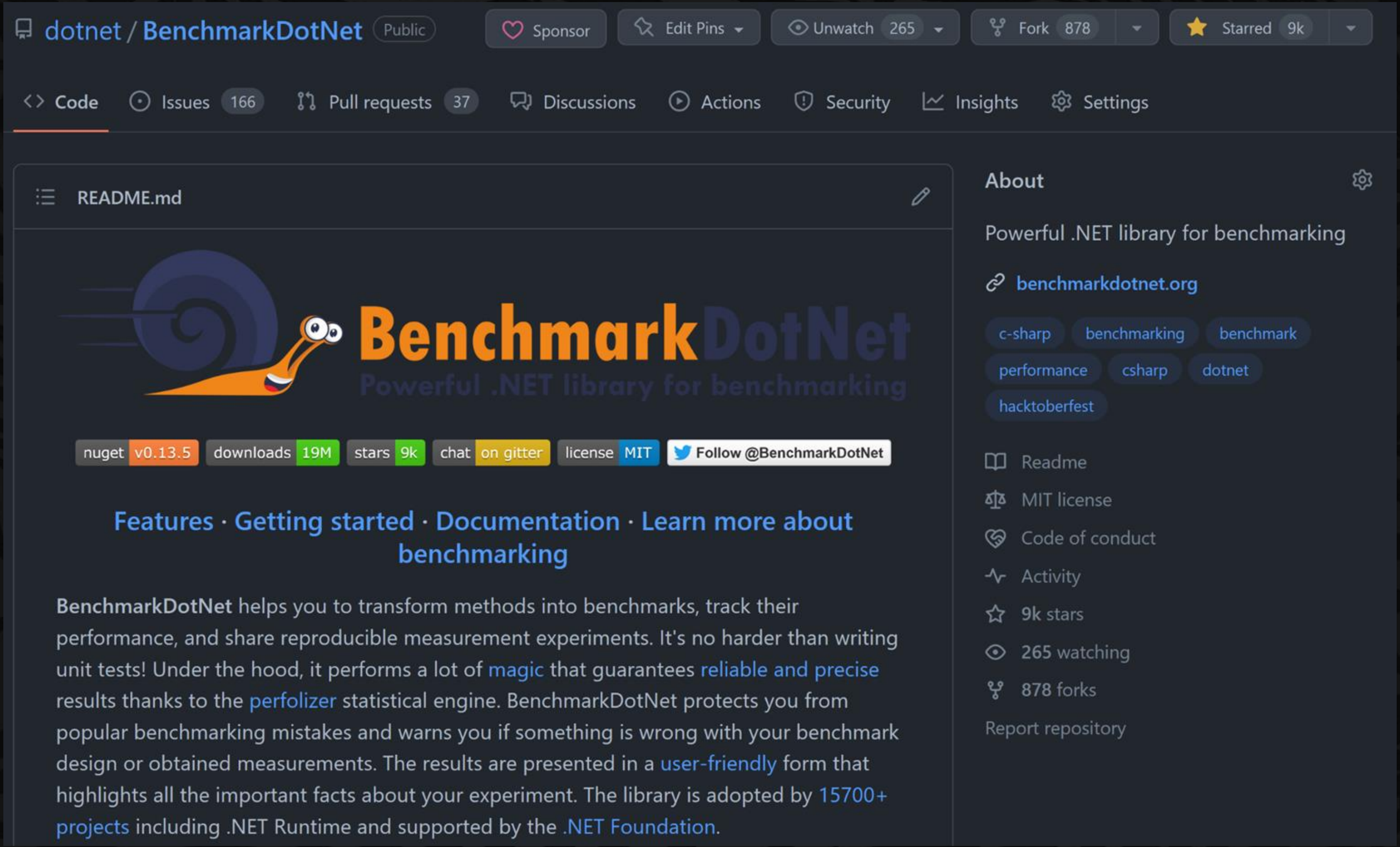
1. New statistical engine
2. Fix some bugs
3. Improve API

Conclusion

BenchmarkDotNet: Summary

- Accurate, Repeatable and Stable Results
- **Powerful** Statistics
- Rich support:
  - C#, F#, VB
  - .NET 4.6+, .NET Core 2.0+, Mono, NativeAOT, WASM
  - Windows, Linux, macOS
  - x64, x86, arm64, arm, S390x, LoongArch64, armv6, ppc64le
- Easy benchmark design (no boilerplate code and nice API)
- Great User Experience
- Strong community
- Very good test coverage

Do you still want to write your own harness using Stopwatch?



Features · Getting started · Documentation · Learn more about benchmarking

BenchmarkDotNet helps you to transform methods into benchmarks, track their performance, and share reproducible measurement experiments. It's no harder than writing unit tests! Under the hood, it performs a lot of magic that guarantees reliable and precise results thanks to the perfolizer statistical engine. BenchmarkDotNet protects you from popular benchmarking mistakes and warns you if something is wrong with your benchmark design or obtained measurements. The results are presented in a user-friendly form that highlights all the important facts about your experiment. The library is adopted by 15700+ projects including .NET Runtime and supported by the .NET Foundation.

About

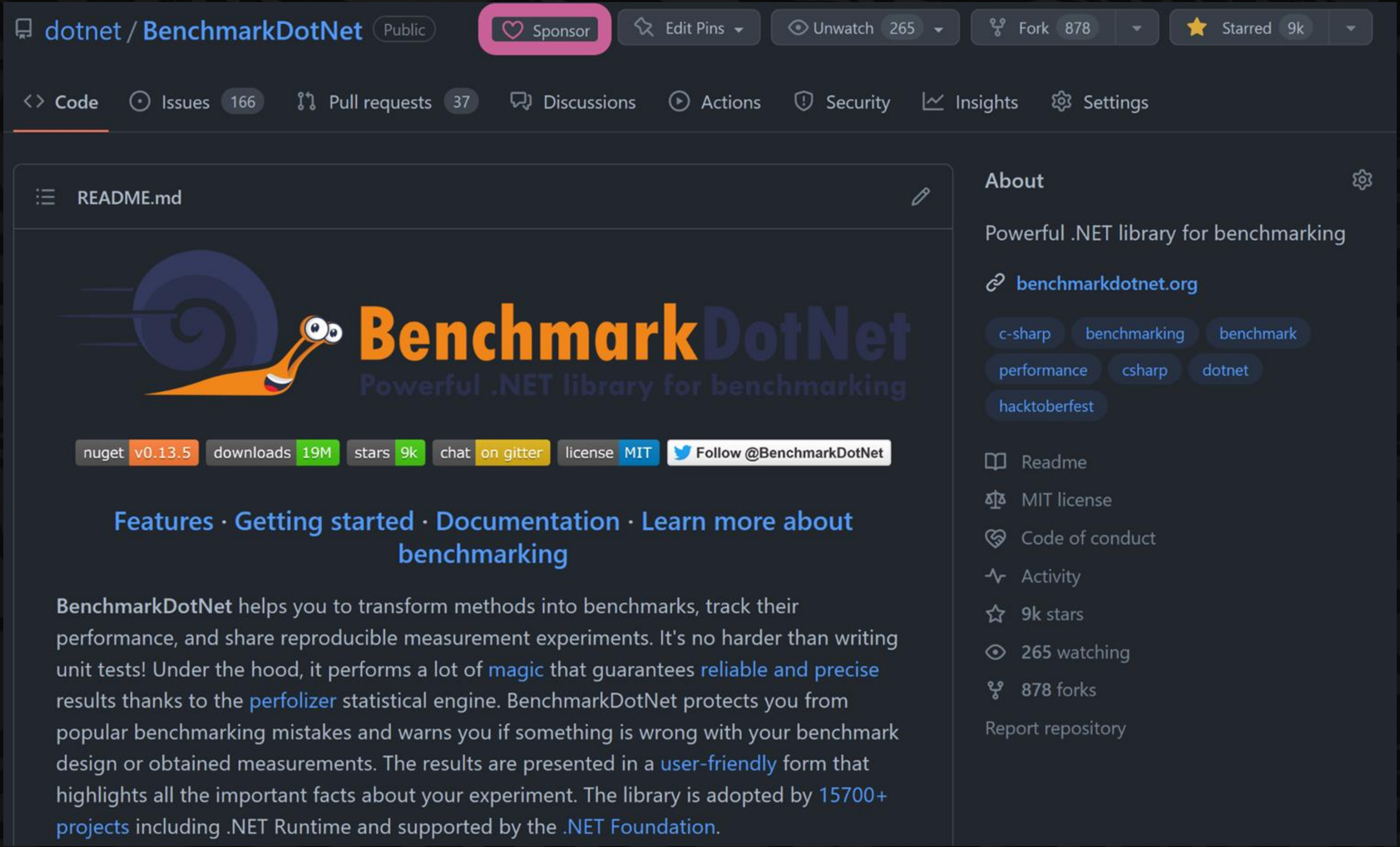
Powerful .NET library for benchmarking

benchmarkdotnet.org

c-sharp benchmarking benchmark performance csharp dotnet hacktoberfest

- Readme
- MIT license
- Code of conduct
- Activity
- 9k stars
- 265 watching
- 878 forks

Report repository



Powerful .NET library for benchmarking

Features · Getting started · Documentation · Learn more about benchmarking

BenchmarkDotNet helps you to transform methods into benchmarks, track their performance, and share reproducible measurement experiments. It's no harder than writing unit tests! Under the hood, it performs a lot of magic that guarantees reliable and precise results thanks to the perfolizer statistical engine. BenchmarkDotNet protects you from popular benchmarking mistakes and warns you if something is wrong with your benchmark design or obtained measurements. The results are presented in a user-friendly form that highlights all the important facts about your experiment. The library is adopted by 15700+ projects including .NET Runtime and supported by the .NET Foundation.

https://github.com/dotnet/BenchmarkDotNe

@BenchmarkDotNet

https://aakinshin.net

andrey.akinshin@gmail.com

@andrey_akinshin

https://adamsitnik.com

adam.sitnik@gmail.com

@SitnikAdam