# Object-Oriented Incremental Dynamic Analysis (ooida)

Version 0.1.1

Alex Baker

https://github.com/ambaker1/ooida

September 9, 2023

# Contents

# General

Incremental Dynamic Analysis (IDA) is a methodology for collapse assessment of buildings, and is essentially a parametric study, investigating the nonlinear response of a structure exposed to different earthquakes at increasing intensity. Of primary interest is to determine, at a high resolution, the collapse capacity of the structure for the given ground motion. Of secondary interest is to trace the curve below the capacity point, to ensure that the capacity point is not part of a "structural resurrection". The main hunt-fill tracing algorithm in this package is based on the original paper by Vamvatsikos and Cornell 2002 [5] and the parallel processing implementation is based on that described by Vamvatsikos and Cornell 2011 [4].

In similar fashion to the traditional hunt-fill tracing algorithm, the "ooida" package hunts for the capacity point and fills in the curve using a multi-staged approach:

Stage 0: Initialization stage, or simply the starting point.

Stage 1: "Hunt-up" stage, where it will increase the intensity until collapse is reached.

Stage 2: "Bracketing" stage, where it bisects to achieve desired precision on collapse capacity.

Stage 3: "Fill-in" stage, where the gaps below collapse are refined to achieve desired curve granularity.

Stage 4: Completion stage: collapse is reached and the curve is filled, but jobs may still be active.

## *Package Dependencies*

Due to the multi-stage nature of an IDA algorithm, and to facilitate running multiple IDA curves simultaneously, an Object-Oriented approach was adopted, using the standard "TclOO" package.

Besides being able to run multiple IDA curves in parallel, intra-record parallelization (running multiple runs for a single curve simultaneously) is supported by the IDA algorithm for the "fill-in" stage, and implemented for OpenSeesMP using the "mpjobs" framework: https://github.com/ambaker1/mpjobs

As with the "mpjobs" framework, the "ooida" package utilizes the tabular data structure provided by the "Tda" package: https://github.com/ambaker1/Tda.

All package dependencies are automatically installed if "ooida" is installed with the "Tin" package: https://github.com/ambaker1/Tin.

# Creating IDA Objects

IDA objects are created from the *ida* class using the standard methods *new* or *create*. Both class methods require definition of IDA settings `-collapse` and `-precision`. Once created, *ida* objects act as commands with an ensemble of subcommands, or methods. These objects can be copied with the method *copy* and deleted with the method *destroy*.

```
ida new $option $value ...
ida create $objectName $option $value ...
```

| | |
|---|---|
| **`$objectName`** | Explicit name for object. |
| **`$option $value ...`** | Paired list of IDA options and corresponding values, see *$idaObj configure*. |

Example 1: Creating an IDA object

*Code:*

```
set settings {}
dict set settings -huntup {Geometric 1.0 2.0}
dict set settings -collapse {SC {code != 0} NSC {drift > 0.10}}
dict set settings -precision {0.01 0.1}
set idaObj [ida new {*}$settings]
```

## *Copying IDA Objects*

The method *copy* copies all the data from an IDA object to a new object.

```
$idaObj copy <$objectName>
```

| | |
|---|---|
| **`$objectName`** | Explicit name for object. By default, returns an auto-generated name. |

## *Removing IDA Objects*

The standard method *destroy* removes an IDA object from the interpreter.

```
$idaObj destroy
```

# IDA Configuration

The settings of an IDA object can be queried and modified with the method *configure*. If this method is called with no arguments, it returns a dictionary of all configuration settings. If called with a single argument, it will return the current configuration for the specified setting. If called with more than one argument, the input will be parsed as a paired list of settings and configuration values to apply.

```
$idaObj configure
$idaObj configure $option
$idaObj configure $option $value ...
```

| | |
|---|---|
| `$option ...` | Setting options to set or query. |
| `$value ...` | Configuration values for settings. |

## *Collapse Criteria*

The collapse criteria must be defined with the *-collapse* configuration setting.

```
-collapse "$name1 $cc1 $name2 $cc2 ...  "
```

| | |
|---|---|
| `$name1 $name2 ...` | Unique collapse names. |
| `$cc1 $cc2 ...` | List of collapse criteria (`$dm $op $value`), as described below: |
| `$dm` | Damage measure name. |
| `$op` | Comparison operator (`== != > < >= <= in ni eq` or `ne`). |
| `$value` | Value to compare to. |

There are two types of collapse conditions: damage measure based and intensity measure based [5]. A damage measure based rule states that the first IDA point that exceeds the damage measure limit is the collapse point. An intensity measure based rule, on the other-hand, defines the collapse point as the last point with a slope greater than the limit. For example, FEMA report 350 defines the last point at which the tangent slope is greater than 20% of the elastic IDA slope as the collapse point [3]. Currently, the "ooida" package only supports damage-measure based rules.

## *Hunt-up Method*

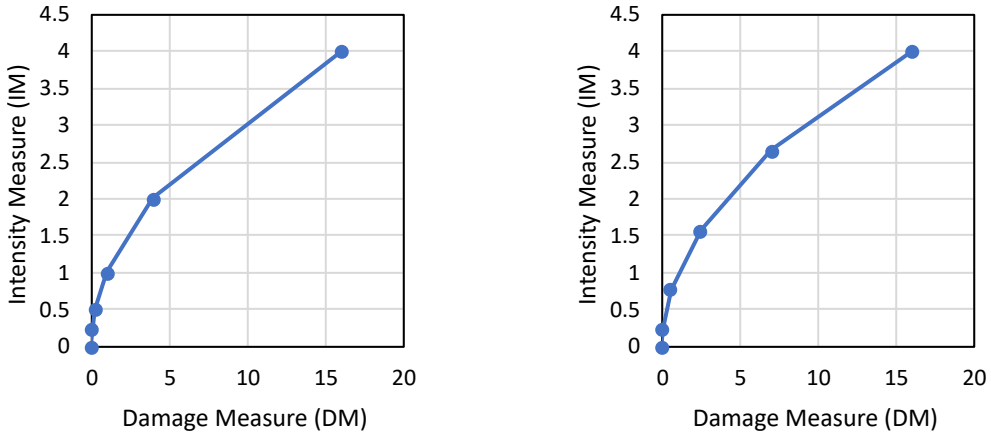
The algorithm used to determine the next intensity measure prior to collapse is defined with the *-huntup* configuration setting. If not specified, it will default to "Geometric 1.0 2.0".

```
-huntup "$type $start $step"
```

| | |
|---|---|
| **$type** | Hunt-up method (*Geometric*, *Quadratic*, or *Linear*, default *Geometric*) |
| **$start** | Starting intensity measure. Also starting step for *Quadratic*. Default 1.0. |
| **$step** | For *Geometric*, the geometric ratio. For *Quadratic*, the step increase. For *Linear*, the step size. Default 2.0. |

The *Geometric* method ensures that the ratio between gaps is constant, while the *Quadratic* method ensures that the difference between gaps is constant. To illustrate their differences, the hunt-up of a simple IDA curve ($DM = IM^2$) is performed with both methods with the same number of steps and the same start and end point, as shown in Figure 1.

As can be seen in the figure, the *Geometric* method can increase gap size more quickly, while the *Quadratic* method produces more consistent gap sizes. In fact, if the *Quadratic* "step" is set to 0.0, the gap sizes will be constant.




(a) Geometric Hunt-up:
Start = 0.25, Step = 2.0

(b) Quadratic Hunt-up:
Start = 0.25, Step = 0.275

Figure 1: Geometric vs Quadratic Hunt-up

## *Hunt and Fill Precision*

Once collapse is reached in the IDA algorithm, the algorithm will focus first on meeting precision on the capacity point, then on meeting precision on the rest of the IDA curve. For the hunt precision, the gap between the estimated capacity and the minimum collapse intensity is bisected until the gap is less than the tolerance. For the fill precision, the gaps below the estimated capacity are bisected or trisected, going in order of largest gaps first, until no gap exists that is greater than the precision. When both precision tolerances are met and all jobs are complete, the IDA curve is considered to be complete.

```
-precision "$hunt $fill"
```

| | |
|---|---|
| **$hunt** | Intensity measure hunt precision. |
| **$fill** | Intensity measure fill precision. |

## *IDA Curve Refining Limits*

Sometimes, the only information needed from an IDA is whether the capacity is above or below a certain value. In this case, the option *-limits* can be used. The -limits option define upper and lower intensity measure limits for collapse refining. By default, they are set to 0 and Inf, respectively.

```
-limits "$lower $upper"
```

| | |
|---|---|
| **$lower** | Lower limit for refining collapse point. |
| **$upper** | Upper limit for refining collapse point. |

The limits define three intensity measure zones as shown in Figure 2. In the figure, if collapse has not been reached, the minimum collapse can be considered to be in zone 3.

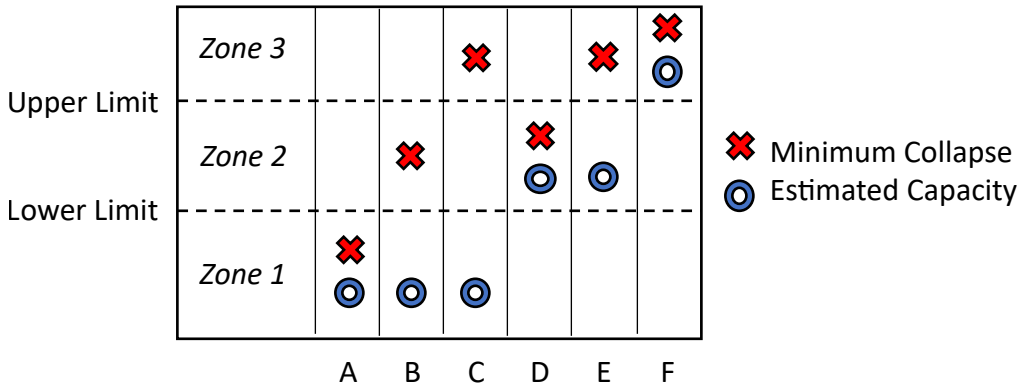



Figure 2: Partial IDA Zones

Depending on which zones the the capacity and minimum collapse points are, it will modify the behavior of the IDA as described below:

A Both estimated capacity and minimum collapse are in zone 1. No more runs are required.

B Capacity is below lower limit, but minimum collapse is above. Fill-in stage is not needed.

C Same as B.

D Both estimated capacity and minimum collapse are in zone 2. Full refinement is necessary to capture any structural resurrections.

E Same as D.

F Both estimated capacity and minimum collapse are in zone 3. Hunting and bracketing are not required, but curve must be filled in below upper limit to capture any structural resurrections.

# Running an IDA

The "ooida" package does not run the individual dynamic analyses of an IDA. Rather, it provides a guiding framework for tracing IDA curves. Individual runs are run either directly in the current instance of OpenSees, or with a job using the job board framework (recommended due to its save-state functionality).

## *Adding IDA Points*

IDA data points are inputted via the method *add*. If the input is a single item, it is interpreted as a job name, using the job board framework. If the job is posted but not running, it will run the job. If the input is a key-value paired list, it is interpreted as damage measure names and values. The only reserved keys are "collapsed", "intensity", and "jobTag".

```
$idaObj add $im <$jobTag> <$dmName $dmValue ...>
```

| | |
|---|---|
| **$im** | Intensity measure to add. |
| **$jobTag** | Integer job tag, using the job board framework, that corresponds to the intensity measure. |
| **$dmName $dmValue ...** | Paired list of damage measure names and values for the intensity measure. Mutually exclusive with $jobTag. |

## *Removing IDA Points*

The method *remove* removes an intensity measure from an IDA curve. If the intensity measure does not exist in the IDA object, it will do nothing.

```
$idaObj remove $im
```

| | |
|---|---|
| **$im** | Intensity measure to remove. |

## *Wiping the IDA*

An IDA can be wiped with the method *wipe*. This removes all existing points and resets collapse information, but maintains any IDA configuration settings.

```
$idaObj wipe
```

## *Getting the Next Intensity Measure*

The next intensity measure to run according to the hunt-up settings, collapse criteria, and precision values, can be queried with *$idaObj next*. Returns -1 if waiting on active jobs, and -2 if all criteria are met.

```
$idaObj next
```

## *Check if IDA is Complete*

The completion status of an IDA curve can be queried with *$idaObj complete*. The IDA is considered complete when collapse is reached, hunt and fill precisions are met, and no active jobs remain. This is equivalent to checking if the next intensity is -2.

```
$idaObj complete
```

## *Updating the IDA*

If using the job board framework for running an IDA, results from jobs will only be added when *update* is called. This method is automatically called during *$idaObj run*.

```
$idaObj update
```

## *Run an IDA*

For convenience, the flow of an IDA can be controlled entirely with *$idaObj run*.

```
$idaObj run $imVar $body
```

| | |
|---|---|
| **$imVar** | Variable to pass next intensity measure to. |
| **$body** | Body to evaluate that runs each IDA point. Must evaluate to a result dictionary or a job tag, in the same fashion as the Tcl command "lmap". |

Example 2: Running a single IDA in series

*Code:*

```
$ida run Sa {
    wipe
    source DoDynamic.tcl
}
```

Example 3: Running a single IDA with "mpjobs" framework (series or parallel)

*Code:*

```
jobBoard IDA {
    $ida run Sa {
        makeJob DoDynamic.tcl Sa $Sa
    }
}
```

# Getting IDA State Information

Various methods are provided for accessing the state of the IDA.

## *IDA Stage*

The stage of the IDA can be queried with the method *stage*.

```
$idaObj stage
```

## *Estimated Capacity*

The estimated capacity, as in the largest intensity measure value below the minimum collapse, can be queried with the method *capacity*.

```
$idaObj capacity
```

## *Whether Collapse Was Reached*

The method *collapsed* returns 1 if collapse was reached, and zero otherwise.

```
$idaObj collapsed
```

## *Minimum Collapse*

The minimum collapse intensity, if collapse was reached, can be queried with the method *minCollapse*. If collapse was not reached, it will return blank.

```
$idaObj minCollapse <$name>
```

| | |
|---|---|
| **$name** | Collapse name associated with collapse criteria. Default returns minimum for all collapse criteria. |

## *List of All Intensities*

A sorted list of all intensity measure values can be queried with *$idaObj getIMs*.

```
$idaObj getIMs
```

## *Damage Measure Curve*

A damage measure curve, in matrix form, can be queried with *$idaObj curve*. The first column of the matrix will be the damage measure values, and the second column will be the intensity values, in increasing order.

```
$idaObj curve $dm
```

| | |
|---|---|
| **$dm** | Damage measure name. |

## *Estimated Empirical CDF*

If the IDA object is a part of a suite object, the method *cdf* will return the estimated CDF for the corresponding ground motion. Otherwise, the method does not exist.

```
$idaObj cdf
```

## *Getting Table of IDA Data*

The method *table* returns a "Tda" table object containing IDA curve data, where keys correspond to intensity measure values and fields correspond to damage measures. Additionally, the table includes a field called "jobTag", a field named "collapsed", which is 1 if the intensity is above capacity, and 0 otherwise, and fields for each collapse criteria name.

```
$idaObj table
```

The IDA curve can be easily exported to CSV with the "Tda" command *writeTable*, as shown below.

Example 4: Exporting IDA table to CSV

*Code:*

```
writeTable curve.csv [$idaObj table]
```

# IDA Suite Objects

Since IDAs are typically done with a suite of ground motions, an wrapper class is provided: *suite*. This wrapper class can be used to easily run multiple IDAs simultaneously. Suite objects, like IDA objects, are created from the *suite* class using the standard methods *new* or *create*. Once created, *suite* objects act as commands with an ensemble of subcommands, or methods, and can be deleted with the method *destroy*.

```
suite new <-medianSearch $medianSearch>
suite create $objectName <-medianSearch $medianSearch>
```

| | |
|---|---|
| **$objectName** | Explicit name for object. |
| **$medianSearch** | Whether to do partial IDA, targeting median CDF. Default false. |

The *-medianSearch* option toggles the partial IDA refinement algorithm, which minimizes the number of analyses required to obtain the median collapse intensity measure required for FEMA P-695 analyses [2]. Further explanation of the partial IDA refinement algorithm can be found in [1].

## *Removing Suite Objects*

The standard method *destroy* removes an IDA suite object from the interpreter, and also destroys any linked IDA objects.

```
$suiteObj destroy
```

## *Adding IDA Objects to Suite*

The method *add* adds an IDA curve to the suite. Adding an IDA to a suite sets up a "write" trace on the capacity variable of the IDA object, such that whenever it changes, the estimated fragility curve is updated accordingly. Additionally, if the *-medianSearch* option is set, it will then update the *-limits* configuration of each IDA object to reduce the number of extraneous runs. It also adds the *cdf* method to the IDA object.

```
$suiteObj add $gm $idaObj
```

| | |
|---|---|
| `$gm` | Ground motion name to identify IDA by. |
| `$idaObj` | IDA object. |

## *Removing IDA Objects from Suite*

The method *remove* removes an IDA curve from the suite. It also removes the associated variable trace and *cdf* method from the IDA object.

```
$suiteObj remove $gm
```

| | |
|---|---|
| `$gm` | Ground motion name to identify IDA by. |

## *Getting the Next Ground Motion and Intensity*

Similar to *$idaObj next*, the next ground motion and intensity measure can be queried with *$suiteObj next*. If waiting on active jobs, it will return "`{} -1`", and if all ground motions are complete, it will return "`{} -2`". Otherwise, it will return a list of the ground motion and an intensity measure to run. If called multiple times, it will cycle through the available IDA curves.

```
$suiteObj next
```

## *Checking if all IDA are Complete*

The method *complete* returns 1 if all IDA curves are complete, and 0 otherwise.

```
$suiteObj complete
```

## *Updating an IDA Suite*

If using the job board framework for running an IDA suite, results from jobs will only be added when *$idaObj update* is called for each IDA. The suite method *update* simply calls the corresponding IDA method for each IDA object in the suite. This method is automatically called during *$suiteObj run*.

```
$suiteObj update
```

## *Running an IDA Suite*

For convenience, the flow of running an entire IDA suite can be controlled entirely with *$suiteObj run*.

```
$suiteObj run $gmVar $imVar $body
```

| | |
|---|---|
| **$gmVar** | Variable to pass next ground motion to. |
| **$imVar** | Variable to pass next intensity measure to. |
| **$body** | Body to evaluate that runs each IDA point. Must evaluate to a result dictionary or a job tag, in the same fashion as the Tcl command "lmap". |

Example 5: Running an IDA suite in series

*Code:*

```
$suite run gmFile Sa {
    wipe
    source DoDynamic.tcl
}
```

Example 6: Running an IDA suite with "mpjobs" framework (series or parallel)

*Code:*

```
$suite run gmFile Sa {
    makeJob DoDynamic.tcl gmFile $gmFile Sa $Sa
}
```

## Accessing Suite IDA Objects

The IDA objects in a suite can be queried or called directly with *$suiteObj ida*.

```
$suiteObj ida $gm $arg1 $arg2 ...
```

| | |
|---|---|
| `$gm` | Ground motion associated with IDA object. |
| `$arg1 $arg2 ...` | Arguments to pass to IDA object. If none, it will return the IDA object. |

## Getting all Ground Motions in Suite

The method *getGMs* returns a list of all ground motions defined for the suite.

```
$suiteObj getGMs
```

## Getting Summary Table of Fragility Data

The method *table* returns a "Tda" table, where the ground motions are the keys and sorted in increasing collapse capacity. The fields in the fragility data table are as follows:

| Field | Description |
|---|---|
| capacity | Collapse intensity (result of *$idaObj capacity*). |
| cdf | Cumulative distribution function (CDF) value of fragility curve. |
| collapsed | Whether the ground motion reached collapse. |
| ... | Collapse criteria names defined with -collapse option of *$idaObj configure*. |

```
$suiteObj table
```

The fragility data can be easily exported to CSV with the "Tda" command *writeTable*, as shown below.

Example 7: Using "Tda" table methods to access fragility data

*Code:*

```
set fragilityData [$suiteObj table]
set gms [$fragilityData keys]; # list of ground motions, in order of increasing collapse
    capacity
set cdf [$fragilityData cget cdf]; # List of cdf values for each ground motion
writeTable fragility.csv $fragilityData
```

# Example Applications

The "ooida" package is a flexible framework for running incremental dynamic analyses. It can be used to process existing data, trace IDA curves in series, or trace IDA curves in parallel using the job board framework.

## *Process Existing Data*

Since the IDA framework does not actually run the analyses, existing data can be added directly to an IDA object.

Example 8: Add existing data from CSV file with intensity as first column

*Code:*

```
package require tin
tin import readTable from tda
set idaTable [readTable idadata.csv]; # from ida table output
dict for {im results} [$idaTable data] {
    $idaObj add $im {*}$results
}
```

## *Tracing an IDA Curve in Series*

The job board framework is not needed for running an IDA in series, as shown in the example below.

Example 9: IDA of a nonlinear frame (RunDynamic.tcl) in series

*Code:*

```
package require tin
tin import writeTable from tda
# Series example
set idaObj [ida new -huntup {Geometric 0.1 2.0} -collapse {NSC {drift > 0.10}} \
        -precision {0.005 0.01}]
$idaObj run Sa {
    wipe
    source RunDynamic.tcl
}
# Save data to file
writeTable idadata.csv [$idaObj table]
```

## *Tracing Multiple IDA Curves in Parallel*

Using the "ooida" package and the "mpjobs" package, it is easy to run multiple IDA curves in parallel, with load sharing between curves, as shown in the example below.

Example 10: IDA for suite of ground motions in parallel

*Code:*

```
# Main loop (load shares between IDA curves!)
package require tin
tin import writeTable from tda
tin import mpjobs
tin import ooida
jobBoard -wipe IDA {
    # Create suite object
    set settings ""
    dict set settings -huntup {Geometric 0.1 2.0}
    dict set settings -collapse {NSC {drift > 0.10}}
    dict set settings -precision {0.005 0.01}
    set suite [suite new]
    foreach gmFile [glob -directory GMs *.AT2] {
        $suite add $gmFile [ida new {*}$settings]
    }

    $suite run gmFile Sa {
        makeJob RunDynamic.tcl gmFile $gmFile Sa $Sa
    }

    # Export data
    writeTable fragility.csv [$suite table]
    foreach gmFile [$suite getGMs] {
        set gmName [file rootname [file tail $gmFile]]
        writeTable $gmName.csv [$suite ida $gmFile table]
    }
}
```

# References

[1] Alexander M Baker. "Kinematics and Preliminary FEMA P-695 Evaluation of the NewZ-BREAKSS Steel Boundary Frame with Buckling Restrained Brace Structural Fuses". Publisher: Michigan Technological University. PhD thesis. 2022.

[2] FEMA. *Quantification of Building Seismic Performance Factors, FEMA P-695*. Washington, D. C.: Federal Emergency Management Agency, 2009.

[3] FEMA. *Recommended Seismic Design Criteria for New Steel Moment-Frame Buildings, FEMA 350*. Washington, D.C.: Federal Emergency Management Agency, 2000.

[4] Dimitrios Vamvatsikos. "Performing Incremental Dynamic Analysis in Parallel". In: *Computers and Structures* 89.1 (2011), pp. 170–180. ISSN: 0045-7949.

[5] Dimitrios Vamvatsikos and C. Allin Cornell. "Incremental Dynamic Analysis". In: *Earthquake Engineering and Structural Dynamics* 31.3 (2002), pp. 491–514. ISSN: 0098-8847.

# Command Index