Object-Oriented Incremental Dynamic Analysis (ooida)

Version 0.2

Alex Baker

https://github.com/ambaker1/ooida

June 1, 2024

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 tracing algorithm used is based on the original "hunt-fill" algorithm by Vamvatsikos and Cornell [4, 3], modified as described in [1].

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 all tolerances are met.

Installation and Use

The "ooida" package and its dependencies can be installed with Tin, a package manager for Tcl. The code below handles installation and importing of all commands.

Example 1: Installing and Importing "ooida"

Code:

```
package require tin
tin add -auto ooida https://github.com/ambaker1/ooida install.tcl
tin import ooida
```

Creating IDA Objects

The command *ida* is a TclOO class based on the superclass *::vutil::ValueContainer*, from the package vutil. It is an object-oriented approach to running Incremental Dynamic Analysis.

```
ida new $varName <$value> <$settings>
ida create $name $varName <$value> <$settings>
```

`$varName`	Variable to tie to IDA object.
`$value`	Dictionary of intensity measures and collapse codes. Default blank.
`$settings`	Settings dictionary. Default blank. See *$idaObj configure* for options.

Copying IDA Objects

The copy operator, "`-->`", copies the IDA to a new variable, and returns the new object.

```
$idaObj --> $varName
```

`$varName`	Variable to store object name for access and garbage collection.

Destroying IDA Objects

Objects can either be destroyed with the method *destroy*, or by simply unsetting or overwriting the variable that it is tied to.

```
$idaObj destroy
```

Below is a simple example of how you create, copy, and delete IDA objects

Example 2: Creating IDA objects

Code:

```
ida new x {1.0 0 2.0 1}; # create new IDA object
$x --> y; # copy IDA object to new variable
unset x; # destroys object stored in x
```

Access/Modification

The IDA data can be accessed by calling the IDA object without any methods, and the value can be overwritten with the assignment operator "=". The assignment operator returns the IDA object.

```
$idaObj = $value
```

`$value`	Dictionary of intensity measures and collapse codes.

The methods *set* and *unset* add or remove points from the IDA curve. Both methods return the object.

```
$idaObj set $im $code
```

`$im`	Intensity measure.
`$code`	Boolean collapse code. 0 for no collapse.

```
$idaObj unset $im
```

`$im`	Intensity measure.

Example 3: Creating and Modifying IDA objects

Code:

```
ida new x; # create new IDA object
$x = {1.0 0 2.0 1}
$x set 1.5 1
$x unset 2.0
puts [$x]
```

Output:

```
1.0 0 1.5 1
```

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, and it will return the object.

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

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

Initialization

The initial intensity measure used for a blank IDA can be configured with the *-start* configuration setting. If not specified, it will default to “1.0”.

```
$idaObj configure -start $init
```

`$init`	Starting intensity measure. Default 1.0.

Hunt-up Method

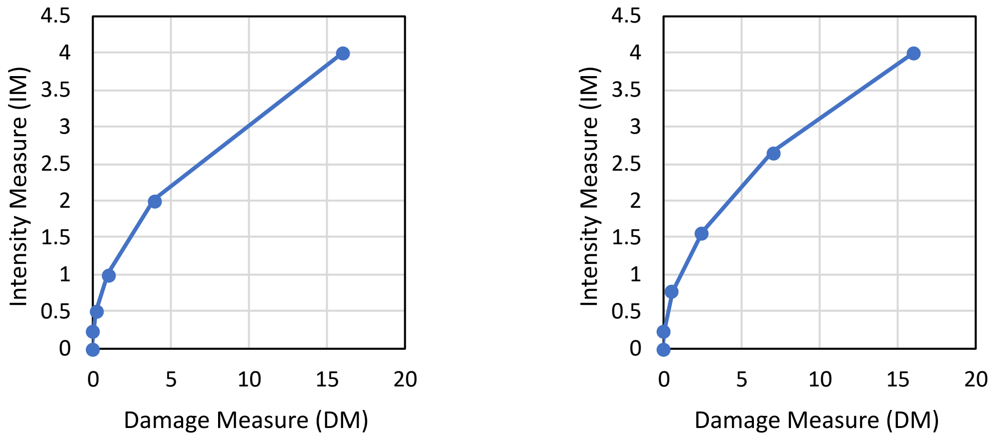
The algorithm used to determine the next intensity measure prior to collapse is defined with the *-huntup* configuration setting. If not specified, it will default to “Geometric 2.0”.

```
$idaObj configure -huntup "$type $step"
```

`$type`	Hunt-up method (*Geometric*, *Quadratic*, or *Linear*. Default *Geometric*)
`$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.



(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

Tolerances for Bracketing and Fill-In

The tolerances for the Bracketing and Fill-In stages of the algorithm are defined with the *-precision* configuration setting. If not specified, it will default to “0.25 0.5”.

```
$idaObj configure -precision "$eps1 $eps2"
```

$eps1	Bracketing stage tolerance. Default 0.25
$eps2	Fill-in stage tolerance. Default 0.5

Once collapse is reached in the IDA algorithm, the algorithm will focus first on meeting tolerance on the collapse point. This is the Bracketing stage, where the gap between the capacity bounds is bisected until the gap is less than the tolerance. When the bracketing tolerance is met, the algorithm enters the Fill-In stage, which focuses on meeting tolerance between points below the collapse point. For the Fill-In stage, the gaps below the collapse point are bisected or trisected, going in order of largest gaps first, until no gap exists that is greater than the specified tolerance. When both precision tolerances are met, the IDA curve is considered to be complete.

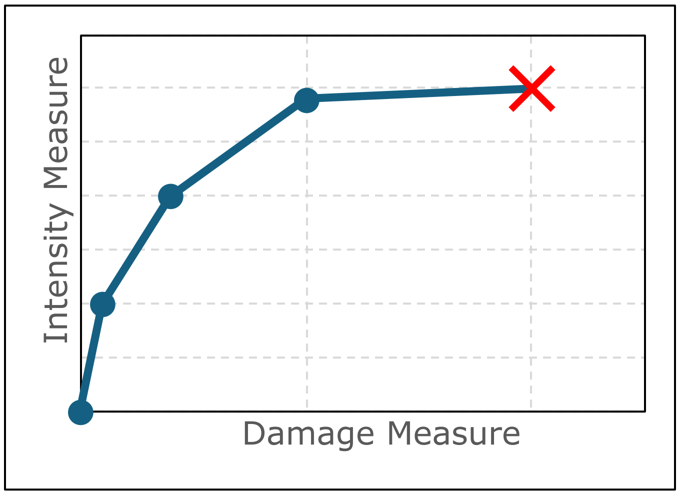


Figure 2: Representative IDA Curve

IDA Curve Refining Limits

The lower and upper limits for refining the IDA collapse point are defined with the *-limits* configuration setting. This can be used to do partial IDAs as described in the FEMA P-695 methodology [2]. If not specified, it will default to "0.0 Inf".

```
-limits "$min $max"
```

`$min`	Lower limit for refining collapse point.
`$max`	Upper limit for refining collapse point.

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

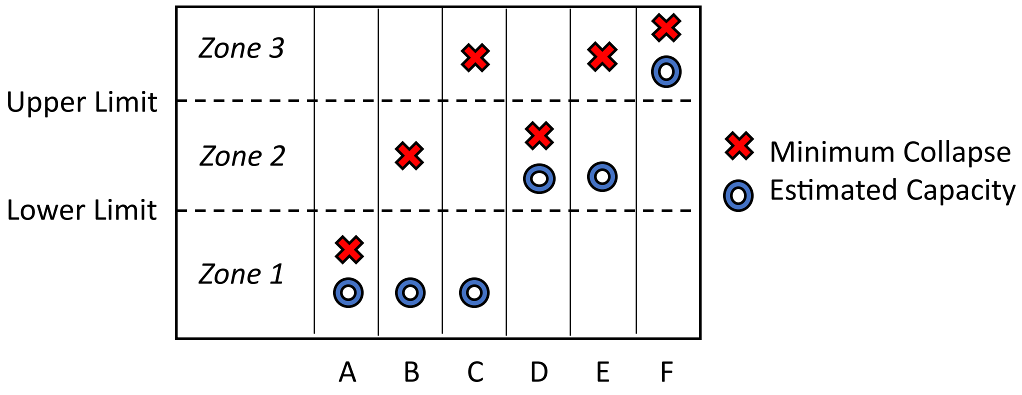


Figure 3: Partial IDA Zones

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.

IDA Results

The IDA dictionary and the configuration settings are used to analyze and determine the stage of the analysis, the next intensity measure to run, and what the estimated bounds are on collapse capacity.

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

```
$idaObj stage
```

The next intensity measure can be queried with *$idaObj next*. If the IDA is complete, it will return blank.

```
$idaObj next
```

The lower and upper limits on collapse capacity can be queried with the method *capacity*. The upper limit is the minimum collapse if collapse is reached.

```
$idaObj capacity
```

Below is an example of how to use the IDA framework to run an analysis.

Example 4: Example Application

Code:

```
# Define place-holder collapse equation (with structural resurrection)
proc Collapsed {x} {
    expr {$x >= 7.5 || ($x > 3.0 && $x <= 3.5)}
}
# Create IDA object
ida new ida {} {
    -huntup {Geometric 2.0}
    -precision {0.25 0.5}
}
# Run IDA
while {[$ida stage] < 4} {
    set im [$ida next]
    $ida set $im [Collapsed $im]
}
# Print out capacity, and history of analysis
puts [$ida capacity]
puts [dict keys [$ida]]
```

Output:

```
3.0 3.25
1.0 2.0 4.0 8.0 6.0 7.0 7.5 7.25 3.0 5.0 0.5 1.5 2.5 3.5 3.25
```

References

[1] Alex Baker. “Kinematics and Preliminary FEMA P-695 Evaluation of the NewZ-BREAKSS Steel Boundary Frame with Buckling Restrained Brace Structural Fuses”. MA thesis. Michigan Technological University, 2022.

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

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

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