

UniDyn--Demo-01.nb

John A. Marohn
jam99@cornell.edu
Cornell University

Abstract: This demonstration notebook loads the **UniDyn** package and executes the package's unit tests.

Set the path to the package

Check the Mathematica version number .

```
In[1]:= $VersionNumber
```

```
Out[1]:= 12.3
```

Tell *Mathematica* the path to the directory containing the packages.

EDIT THE FOLLOWING PATH STRING:

```
In[2]:= $UniDynPath =  
        "/Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/  
        unidyn";
```

YOU SHOULD NOT NEED TO EDIT ANYTHING FROM HERE ONWARDS.

Load the package

Append the package path to the system path. Before trying to load the package, ask *Mathematica* to find it. This is a test that we directed *Mathematica* to the correct directory. The output of this command should be the full system path to the Uni-Dyn.m file.

```
In[3]:= $Path = AppendTo[$Path, $UniDynPath];  
        FindFile["UniDyn`"]
```

```
Out[4]:= /Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/unidyn/UniDyn.m
```

Now that we are confident that the path is set correctly, load the package. Setting the global \$VerboseLoad variable to True will print out the help strings for key commands

in the package.

```
In[5]:= $VerboseLoad = True;
Needs["UniDyn`"]
```

- ... **CreateOperator**: CreateOperator[] is used to batch-define a bunch of operators. Example: CreateOperator[{{lx, ly, lz},{Sx,Sy,Sz}}] will create six operators, where each of the operators in the first list will commute with each of the operators of the second list.
- ... **CreateScalar**: CreateScalar[list] is used to batch-define a bunch of scalars. The parameter list can be a single scalar or a list of scalars. Example: CreateScalar[{w1,w2}].
- ... **NCSort**: NCSort[list] sorts the operators in list into canonical order.
- ... **SortedMult**: SortedMult[list] returns Mult[list\$ordered], where list\$ordered are the elements of list sorted into canonical order.
- ... **MultSort**: MultSort[NonCommutativeMultiply[list]] returns NonCommutativeMultiply[list\$ordered], where list\$ordered are the elements of list sorted into canonical order.
- ... **Comm**: Comm[a,b] calculates the commutator of two operators.
- ... **Inv**: Inv[a] returns the inverse of the expression.
- ... **SpinSingle\$CreateOperators**: SpinSingle\$CreateOperators[lx,ly,lz,L] creates lx, ly, and lz angular momentum operators and defines their commutation relations. When the total angular momentum $L = 1/2$, additional rules are defined to simplify products of the angular momentum operators. When the total angular momentum L is unspecified, no such simplification rules are defined.
- ... **OscSingle\$CreateOperators**: OscSingle\$CreateOperators[aL,aR] creates a raising operator aR and a lowering operator aL for single harmonic oscillator and defines the operator commutation relations.
- ... **Evolve**: Evolve[H, t, $\rho(0)$] calculates $\rho(t) = \text{Exp}[-i H t] \rho(0) \text{Exp}[+i H t]$, assuming that H is time independent, according to the commutation rules followed by $\rho(0)$ and H.
- ... **AllCommutingQ**: A test to see if all the terms in a sum commute with each other.
- ... **VisualComplexity**: A cost function to coax Mathematica into writing simpler-looking answers.
- ... **Evolver1**: Evolver1[H, t, $\rho(0)$] calculates $\rho(t) = \text{Exp}[-i H t] \rho(0) \text{Exp}[+i H t]$, assuming that H is time independent, according to the commutation rules followed by $\rho(0)$ and H.
- ... **Evolver2**: Evolver2[H, t, $\rho(0)$] calculates $\rho(t) = \text{Exp}[-i H t] \rho(0) \text{Exp}[+i H t]$, assuming that H is time independent, according to the commutation rules followed by $\rho(0)$ and H.
- ... **SpinBoson\$CreateOperators**: SpinBoson\$CreateOperators[lx,ly,lz,lp,lm,aR,aL] creates lx, ly, lz spin one half angular-momentum operators; the associated spin raising and lowering operators lp, lm; and harmonic-oscillator raising and lowering operators aR, aL.

Execute the units tests in batch

Included with the package are a number of files, ending in “-tests.m”, that contain tests of the package’s functions -- so-called unit tests. Set the working directory to

the package directory and pretty-print the directory name.

```
In[7]:= SetDirectory[$UniDynPath];
TableForm[{{$UniDynPath}}, TableHeadings → {None, {"Directory"}}]
```

Out[8]//TableForm=

```
Directory
-----
/Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/unidyn
```

Get the names of all the unit-testing files included with the package (following my convention that the unit testing file end in “-tests.m”). Pretty-print the names of the unit-test files included with the package.

```
In[9]:= fn = FileNames["*-tests.m"];
TableForm[{{$fn}}, TableHeadings → {None, {"Test files found"}}]
```

Out[10]//TableForm=

```
Test files found
-----
Comm-tests.m
Evolver1-tests.m
Evolver2-tests.m
Evolve-tests.m
Inv-tests.m
Mult-tests.m
OpQ-tests.m
Osc-tests.m
SpinBoson-tests.m
Spins-tests.m
```

Finally, carry out the unit tests.

```
In[11]:= test$report = TestReport /@ fn;
```

```
TableForm[Table[test$report [[k]], {k, 1, Length[test$report]}]]
```

Out[12]//TableForm=

TestReportObject	 	Title: Test Report: Comm-tests.m Success rate: 100% Tests run: 23
TestReportObject	 	Title: Test Report: Evolver1-tests.m Success rate: 100% Tests run: 14
TestReportObject	 	Title: Test Report: Evolver2-tests.m Success rate: 100% Tests run: 13
TestReportObject	 	Title: Test Report: Evolve-tests.m Success rate: 100% Tests run: 9
TestReportObject	 	Title: Test Report: Inv-tests.m Success rate: 100% Tests run: 24
TestReportObject	 	Title: Test Report: Mult-tests.m Success rate: 100% Tests run: 18
TestReportObject	 	Title: Test Report: OpQ-tests.m Success rate: 100% Tests run: 21
TestReportObject	 	Title: Test Report: Osc-tests.m Success rate: 100% Tests run: 22
TestReportObject	 	Title: Test Report: SpinBoson-tests.m Success rate: 100% Tests run: 2
TestReportObject	 	Title: Test Report: Spins-tests.m Success rate: 100% Tests run: 14

Make a report.

```
In[13]:= tests$passed$total = Plus @@ (test$report[[#]]["TestsSucceededCount"] & /@
List @@ Table[k, {k, 1, Length[test$report]}]);
tests$failed$total = Plus @@ (test$report[[#]]["TestsFailedCount"] & /@
List @@ Table[k, {k, 1, Length[test$report]}]);
```

```
Print[Style[ToString[tests$passed$total] <> " tests passed",
FontWeight → Bold, FontSize → 18, FontColor → Blue]]
Print[Style[ToString[tests$failed$total] <> " tests failed",
FontWeight → Bold, FontSize → 18, FontColor → Red]]
```

160 tests passed**0 tests failed**

Execute the units tests one-by-one

Re-execute the tests in an order determined by us. This is useful for debugging. Running the *Evolve-test.m* file takes a minute.

```
In[17]:= SetDirectory[$UniDynPath];
TableForm[{{$UniDynPath}}, TableHeadings -> {None, {"Directory"}}]
```

```
Out[18]/TableForm=
Directory
/Users/jam99/Dropbox/MarohnGroup__Software_Library/UniDyn/unidyn
```

```
In[19]:= $VerboseLoad = False;
Needs["UniDyn`"]
```


```
In[21]:= TestReport[FileNames["OpQ-tests.m"]][[1]]
```

```
Out[21]= TestReportObject[  Title: Test Report: OpQ-tests.m
Success rate: 100% Tests run: 21
```

```
In[22]:= TestReport[FileNames["Mult-tests.m"]][[1]]
```

```
Out[22]= TestReportObject[  Title: Test Report: Mult-tests.m
Success rate: 100% Tests run: 18
```

```
In[23]:= TestReport[FileNames["Comm-tests.m"]][[1]]
```

```
Out[23]= TestReportObject[  Title: Test Report: Comm-tests.m
Success rate: 100% Tests run: 23
```



```
In[24]:= TestReport[FileNames["Inv-tests.m"]][[1]]
```

```
Out[24]= TestReportObject[  Title: Test Report: Inv-tests.m
Success rate: 100% Tests run: 24
```



```
In[25]:= TestReport[FileNames["Evolve-tests.m"]][[1]]
```

```
Out[25]= TestReportObject[  Title: Test Report: Evolve-tests.m
Success rate: 100% Tests run: 9
```



```
In[26]:= TestReport[FileNames["Evolver1-tests.m"]][1]
```

```
Out[26]= TestReportObject[  Title: Test Report: Evolver1-tests.m  
Success rate: 100% Tests run: 14]
```



```
In[27]:= TestReport[FileNames["Evolver2-tests.m"]][1]
```

```
Out[27]= TestReportObject[  Title: Test Report: Evolver2-tests.m  
Success rate: 100% Tests run: 13]
```



```
In[28]:= TestReport[FileNames["Spins-tests.m"]][1]
```

```
Out[28]= TestReportObject[  Title: Test Report: Spins-tests.m  
Success rate: 100% Tests run: 14]
```

```
In[29]:= TestReport[FileNames["Osc-tests.m"]][1]
```

```
Out[29]= TestReportObject[  Title: Test Report: Osc-tests.m  
Success rate: 100% Tests run: 22]
```

```
In[30]:= TestReport[FileNames["SpinBoson-tests.m"]][1]
```

```
Out[30]= TestReportObject[  Title: Test Report: SpinBoson-tests.m  
Success rate: 100% Tests run: 2]
```

Congratulations

At this point you should have

- (1) loaded the UniDyn package and
- (2) run the UniDyn units tests demonstrating that UniDyn is working as expected.