Comparison Of Differential Equation Solver Software

Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Hairer	ODEPACK/Netlib /NAG	JitCODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Fair	Excellent	Good	Fair	Good	Poor	Poor	Fair	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor****	Poor***	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	None	Good	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	None	Fair	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	None	Good	None
Constant-Lag Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	None	Good	Excellent
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	None	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	None	Good	Fair
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DASPK)	None	Poor	Good	None	None	Excellent	None

* Efficiency takes into account not only the efficiency of the implementation, but the features of the implemented methods (advanced timestepping controls, existence of methods which are known to be more efficient, Jacobian handling)

** Event handling needs to be implemented yourself using basic rootfinding functionality

*** There is a way to write your own C/Fortran code for the derivative, in which case it nearly matches Julia's speed

**** This timing includes JIT compilation with Numba, see https://github.com/JuliaDiffEq/SciPyDiffEq.jl for timing details

For more detailed explainations and comparisons, see the following blog post:

http://www.stochasticlifestyle.com/a-comparison-between-differential-equation-solver-suites-in-matlab-r-julia-python-c-and-fortran

Scale	None	Poor	Fair	Good	Excellent
Explanation	Functionality does not exist	Functionality exists, but is feature-incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.	Has all of the basic features and more. Extra features for flexibility and efficiency.