Progress with **M**ATLAB **S**ource transformation **AD**

MSAD

Rahul Kharche

Cranfield University, Shrivenham

`R.V.Kharche@Cranfield.ac.uk`

AD Fest 2005, Nice

14th - 15th April 2005

Agenda

- Project Goals
- Previous work on MSAD
- Further developments
- Test results from
  - MATLAB ODE examples
  - MINPACK optimisation problems
  - `bvp4cAD`
- Summary
- Future Directions
- References

Project Goals

- Enhance performance by eliminating overheads introduced by operator overloading in MAD [For04]
- Explore MATLAB[a] source analysis and transformation techniques to aid AD
- Create a portable tool that easily integrates with MATLAB based solvers
- Provide control over readability of generated code
- Provide an array of selectable AD specific optimisations

---

[a] MATLAB is a trademark of The MathWorks, Inc.

Previous work on MSAD

- Was shown to successfully compute the *gradient/Jacobian* of MATLAB programs involving vector valued functions using the *forward* mode of AD and source transformation [Kha04]
- Augmented code generated by inlining the `fmad` class operations from MAD
  - the `derivvec` class continued to hold the derivatives and perform derivative combinations
  - resulted in a *hybrid approach* analogous to [Veh01]
- Simple *forward* dependence based *activity analysis*
- Active independent variables and supplementary shape size information can be provided through *user directives* inserted in the code

Previous work on MSAD (contd.)

- Rudimentary *size*(shape) and *type*(constant, real, imaginary) inference
- Thus removed one level of overheads encountered in MAD giving
  - discernible savings over MAD for small problem sizes
  - but these savings grew insignificant as the problem size was increased

Further developments

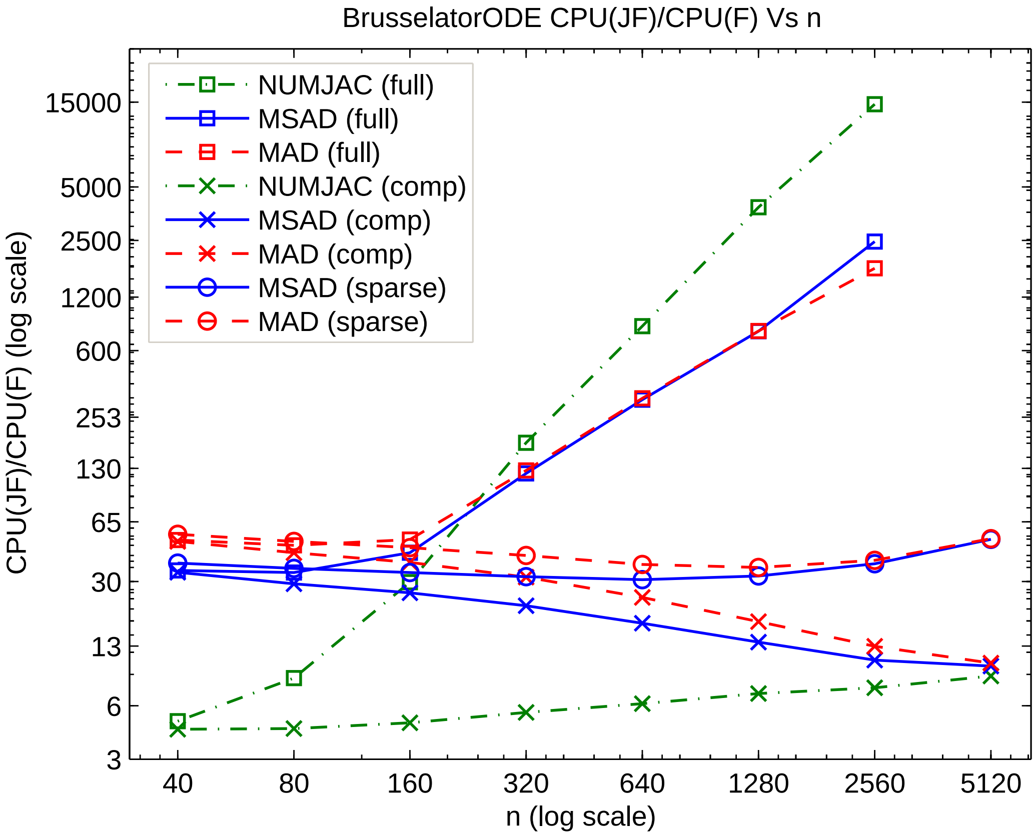
- Now uses size, type inference to *specialise* and further *inline* `derivvec` class operations
- Optionally generates code for holding and propagating sparse derivatives
- Incorporated *sparsity inference* (propagating MATLAB sparse types for derivative variables)
  - if S implies a sparse operand and F full, then rules such as
    - $S + F \to F$, $S * F \to F$
    - $S. * F \to S$, $S\&F \to S$
    - $T = S(i,j) \to T$ is sparse, if i, j are vectors
    - $T(i,j) = S \to$ T retains its full or sparse storage type

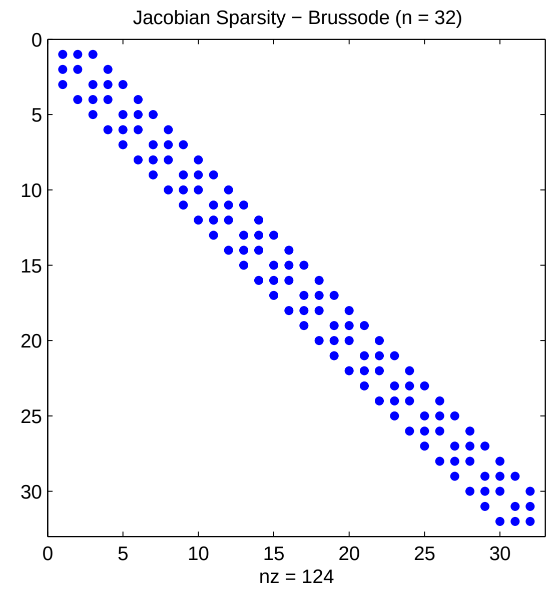
    are applied

Further developments (contd.)

- Run-times are obtained using MATLAB 7.0 on a Linux machine with a 2.8GHz Pentium-4 processor and 512MB of RAM.

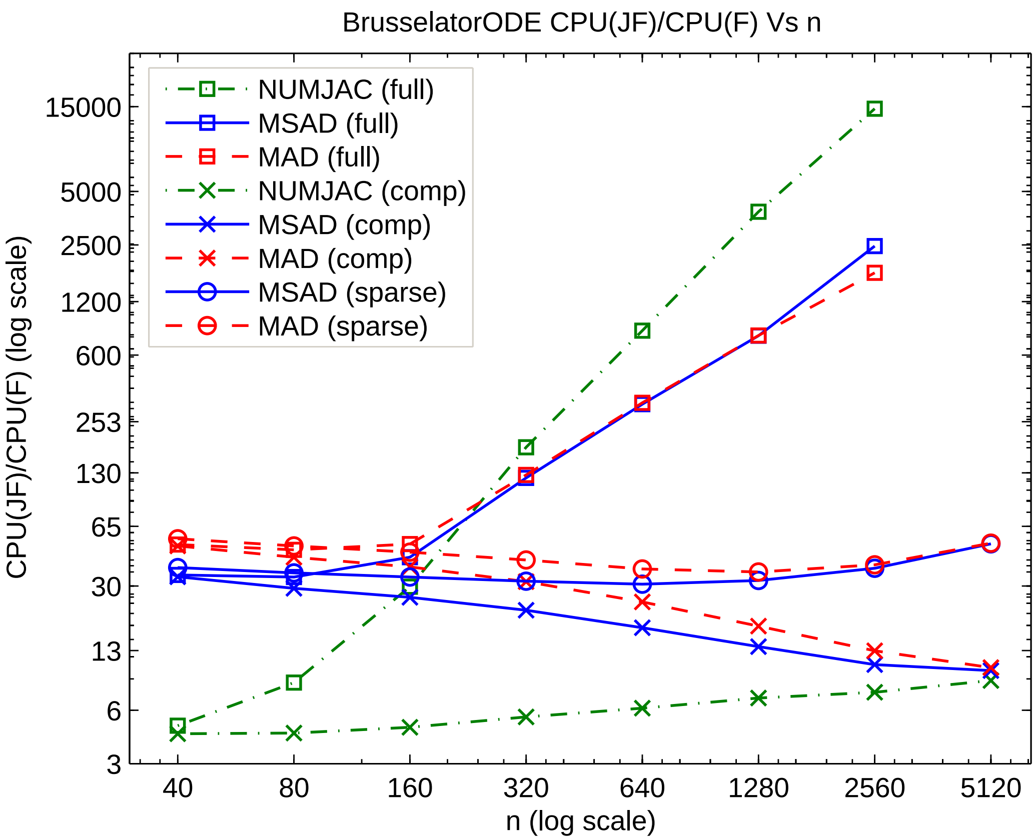
previous Results - Brusselator ODE

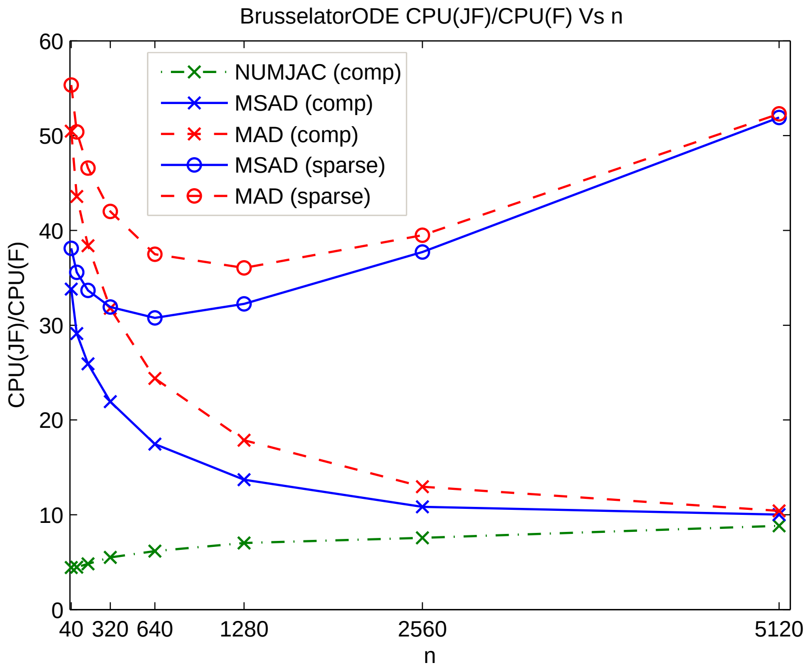




- 30% improvement over MAD for small n down to 4% for large n, with *compression*
- performance matches that of finite-differencing, `numjac(vec)`, only asymptotically

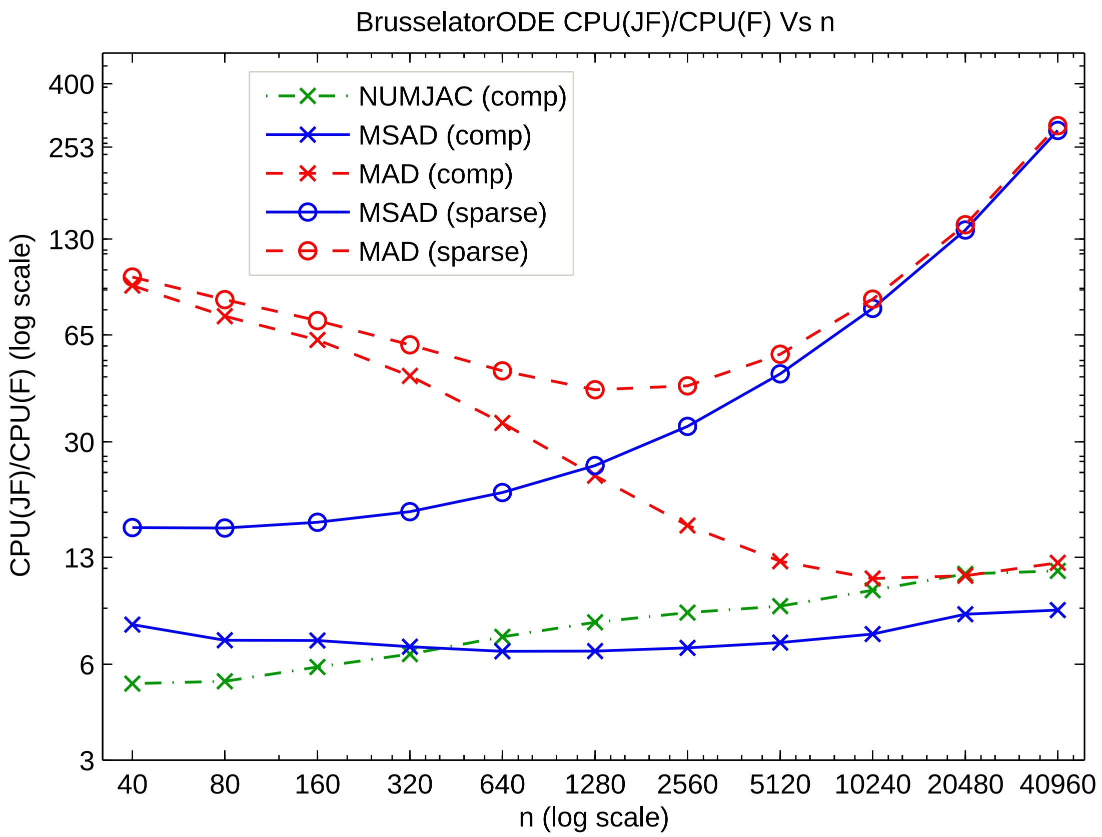
previous Results - Brusselator ODE

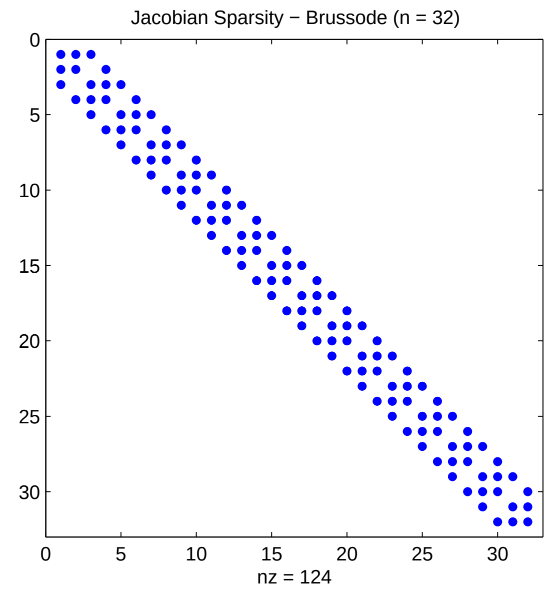




- 30% improvement over MAD for small n down to 4% for large n, with *compression*
- performance matches that of finite-differencing, `numjac(vec)`, only asymptotically
- using *sparse* derivatives, performance converges asymptotically to that of MAD
- almost exponentially increasing savings over *full* evaluation with increasing n

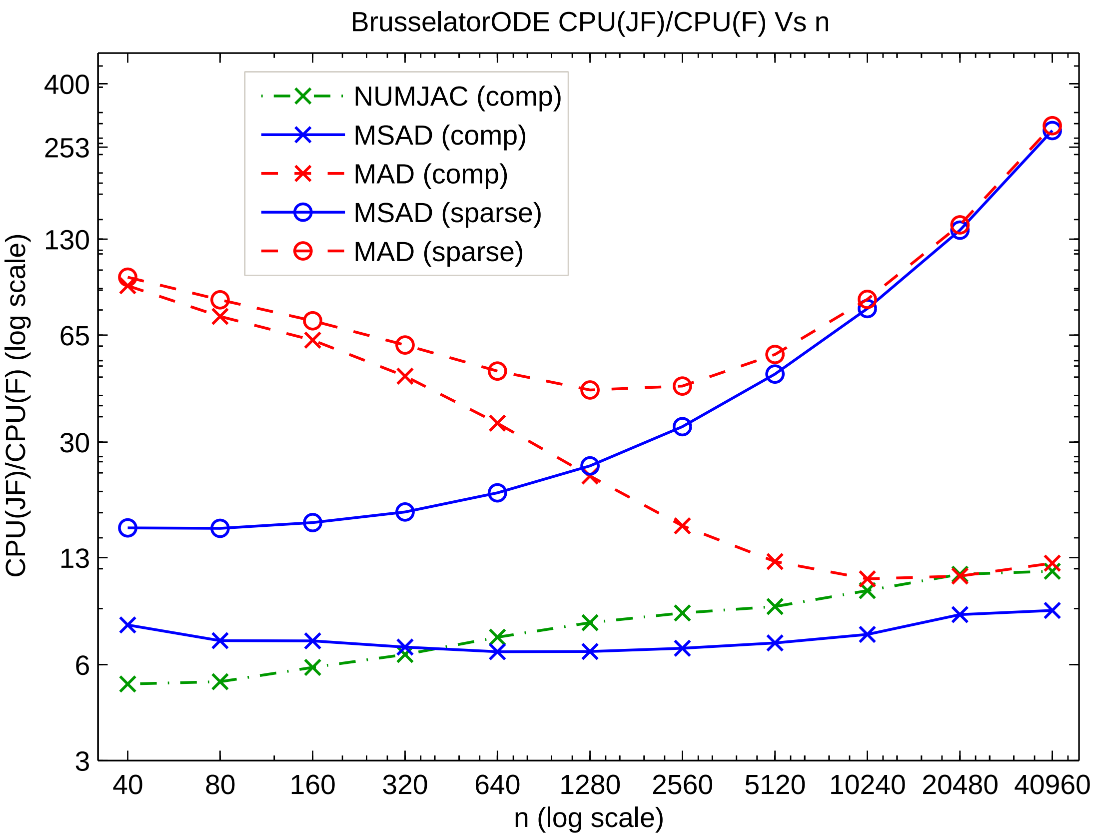
Results - Brusselator ODE

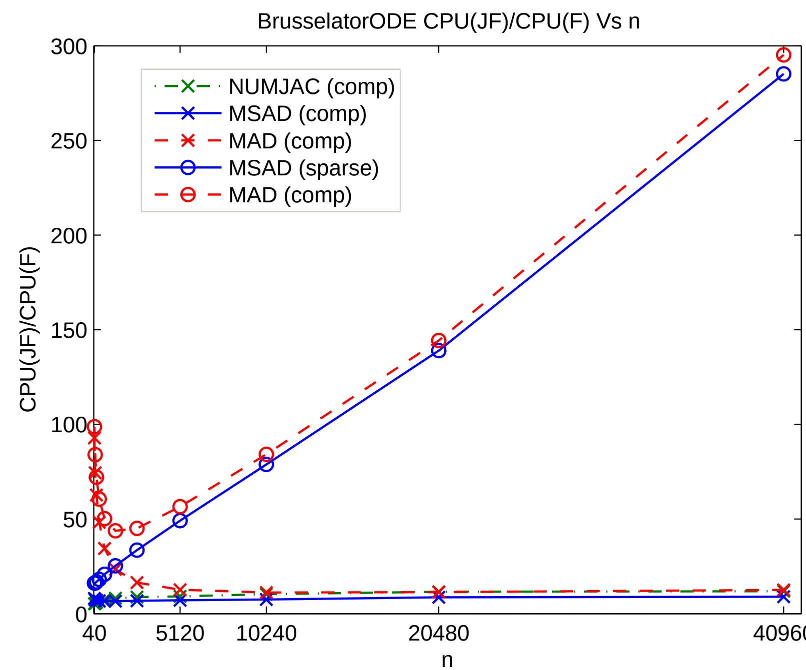




- $91\% \rightarrow 30\%$ speedup over MAD with increasing n using *compression*
- outperforms `numjac(vec)` $n = 640$ onward, with gains upto 25%

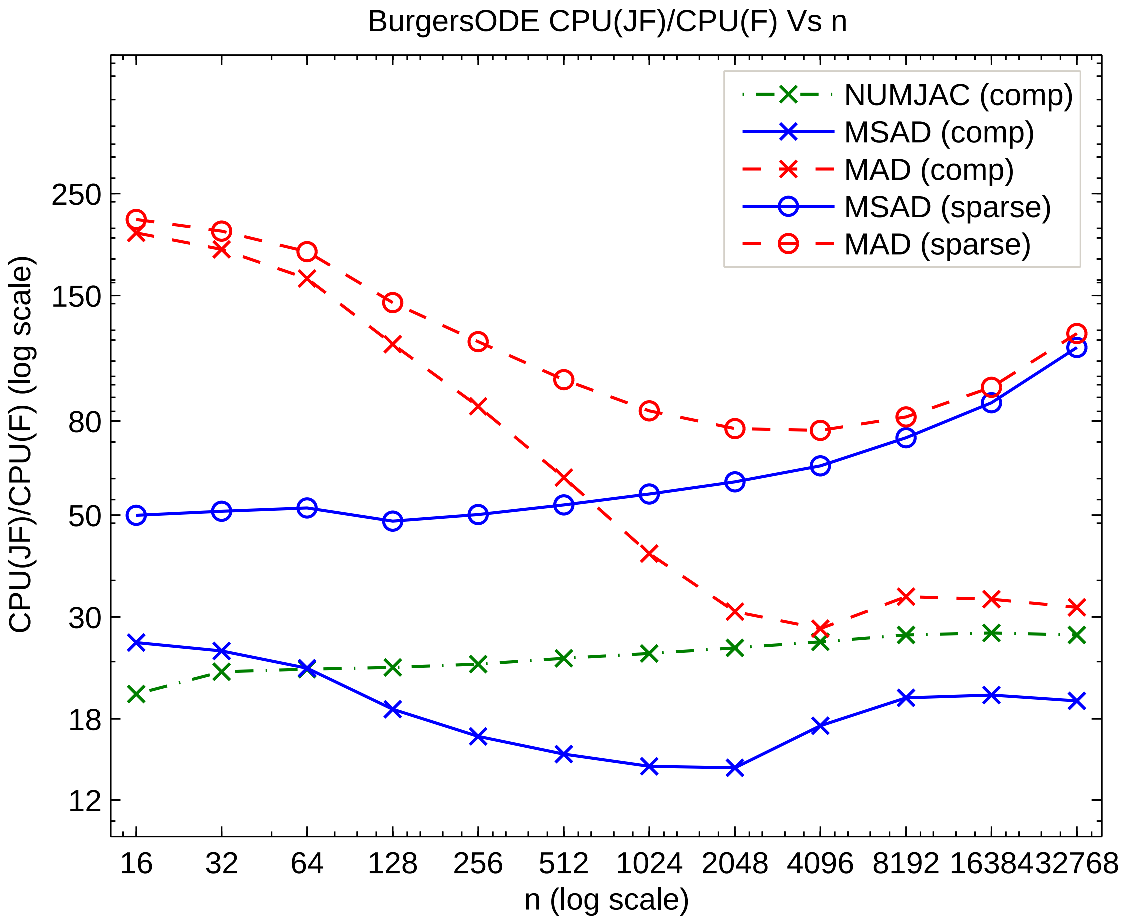
Results - Brusselator ODE

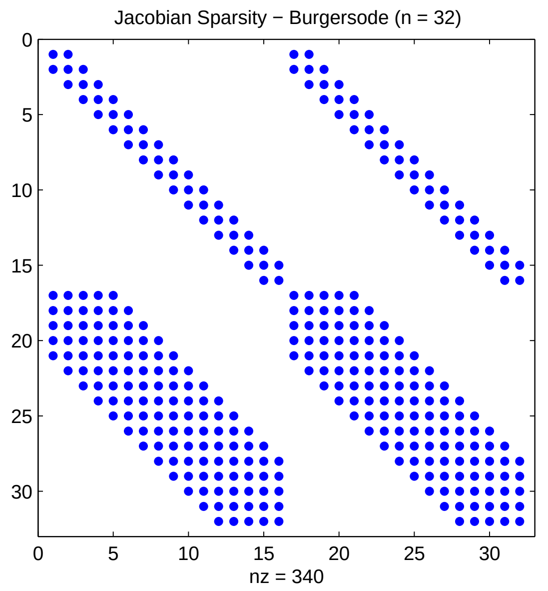




- $91\% \rightarrow 30\%$ speedup over MAD with increasing n using *compression*
- outperforms `numjac(vec)` $n = 640$ onward, with gains upto 25%
- decreasing *relative* speedup, but a small constant saving, over MAD using *sparse* derivatives

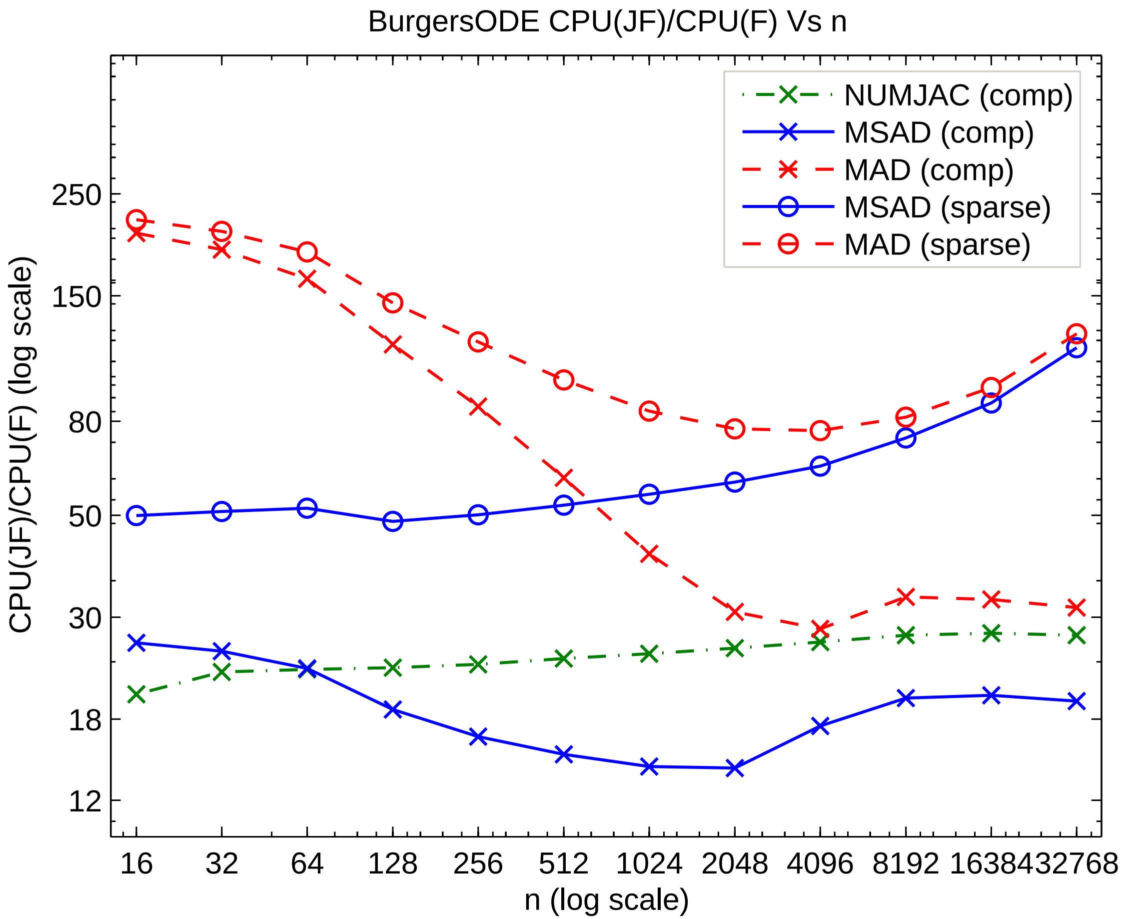
Results - Burgersode ODE

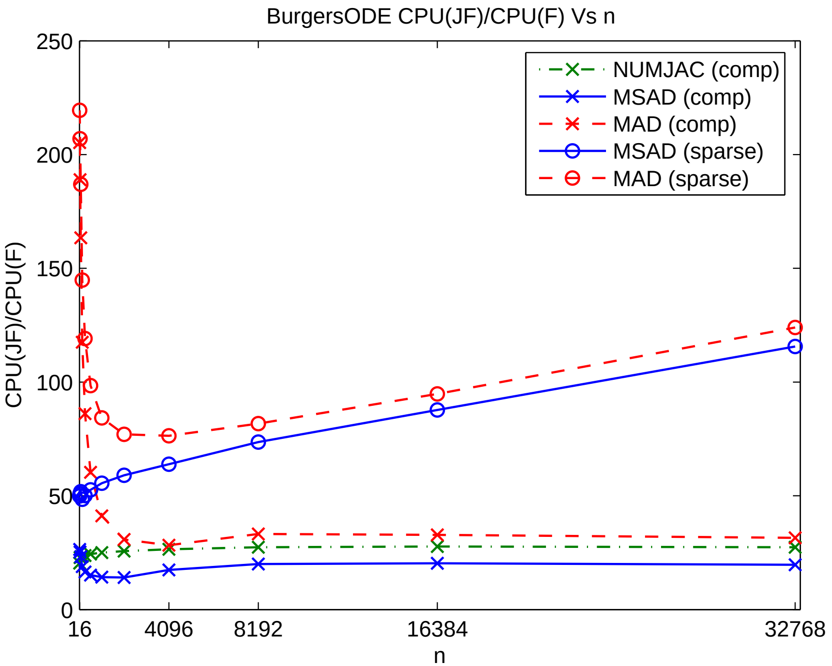




- $87\% \rightarrow 37\%$ speedup over MAD with increasing n, using *compression*
- outperforms `numjac` $n = 64$ onward, with gains between 28% and 45%

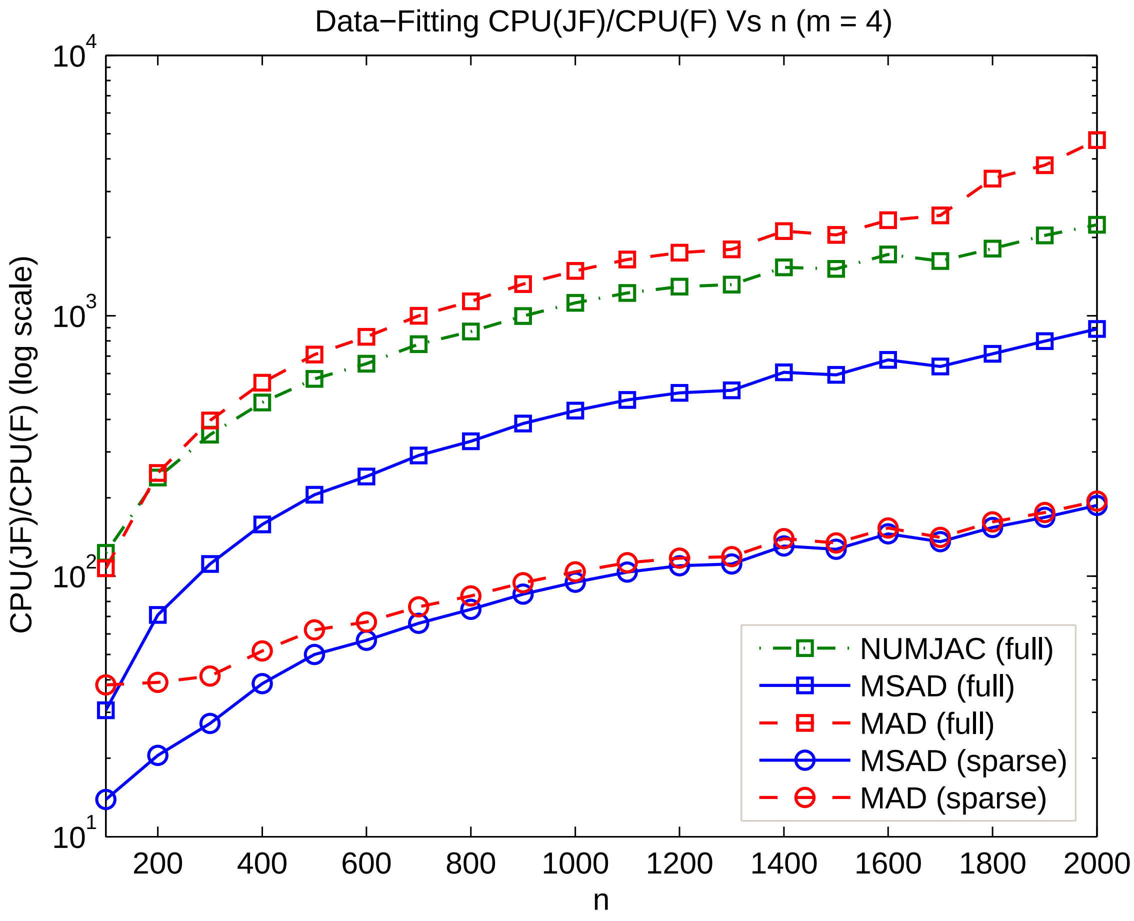
Results - Burgersode ODE





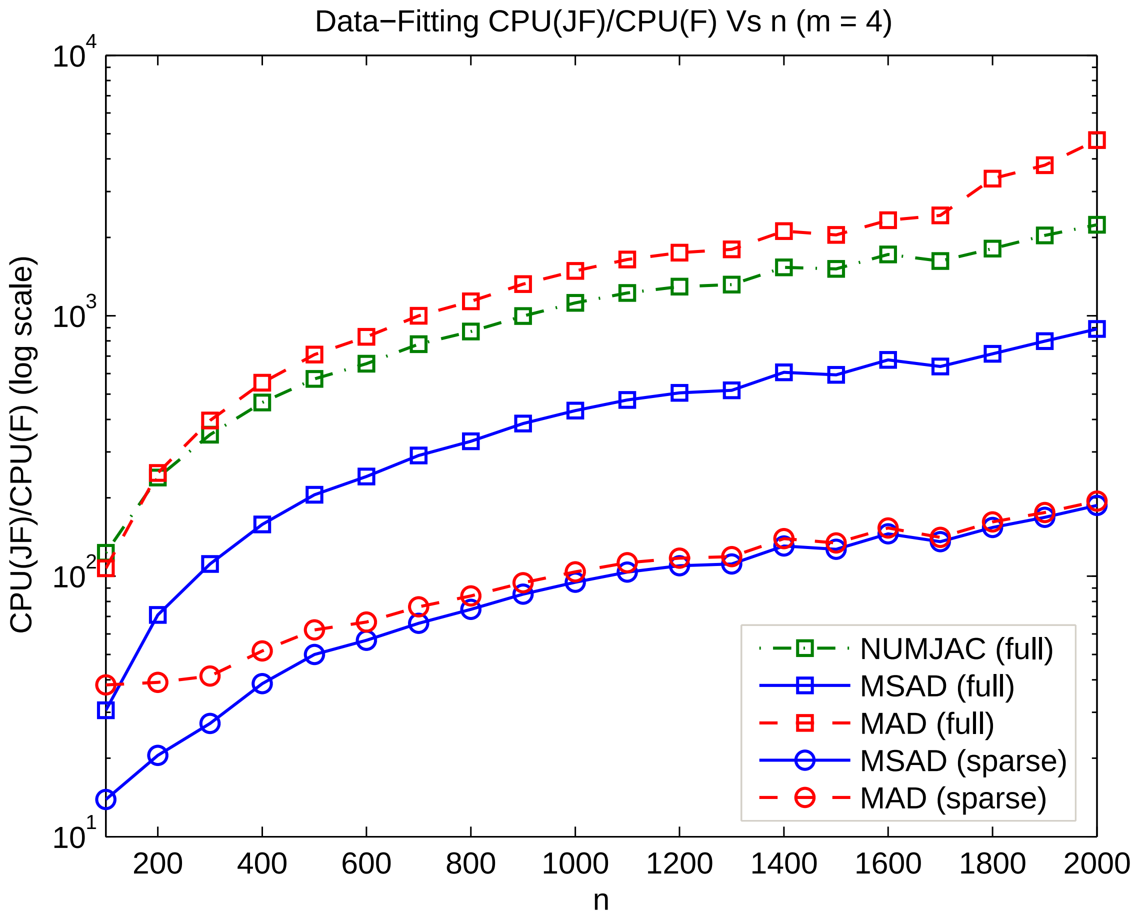
- $87\% \rightarrow 37\%$ speedup over MAD with increasing n, using *compression*
- outperforms `numjac` $n = 64$ onward, with gains between 28% and 45%
- decreasing *relative* speedup, but a small constant saving, over MAD using *sparse* derivatives

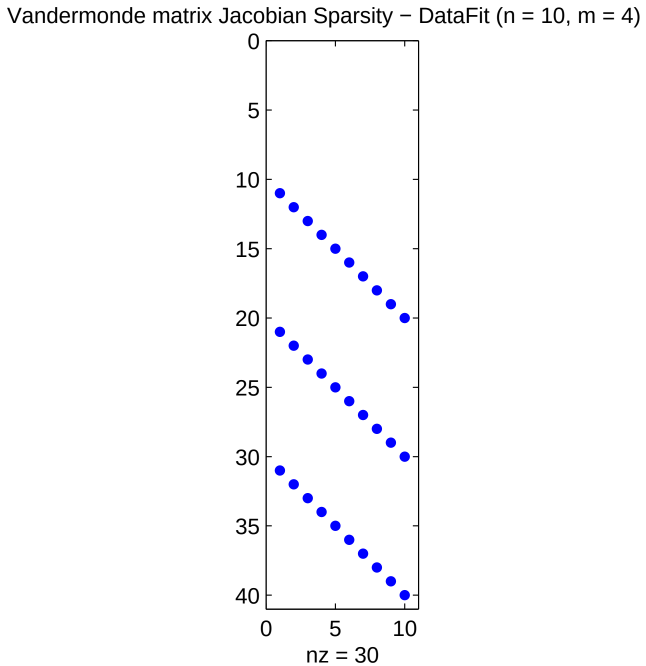
Results - Data Fitting problem



- outperforms both MAD and `numjac` in direct evaluation of the Jacobian by $> 60\%$

Results - Data Fitting problem





- outperforms both MAD and `numjac` in direct evaluation of the Jacobian by $> 60\%$
- if we take note of the sparsity in the Jacobian of the intermediate Vandermonde matrix [For04] and use *sparse* derivatives, we get an order of magnitude improvement over `numjac`, but a decreasing *relative* improvement over MAD

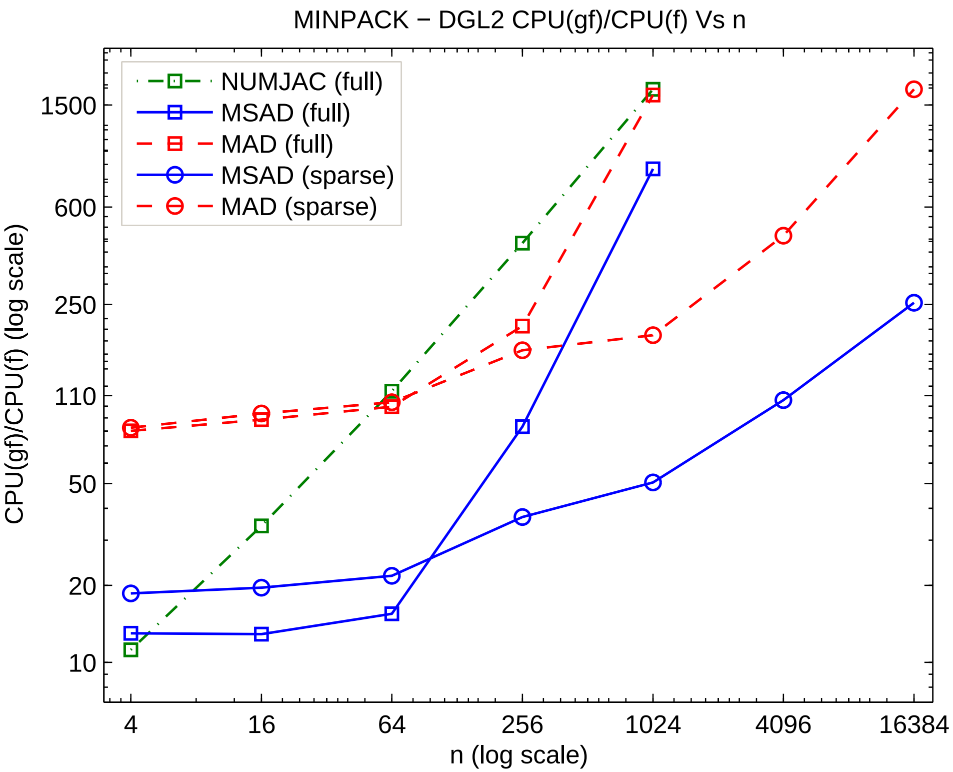
Observations

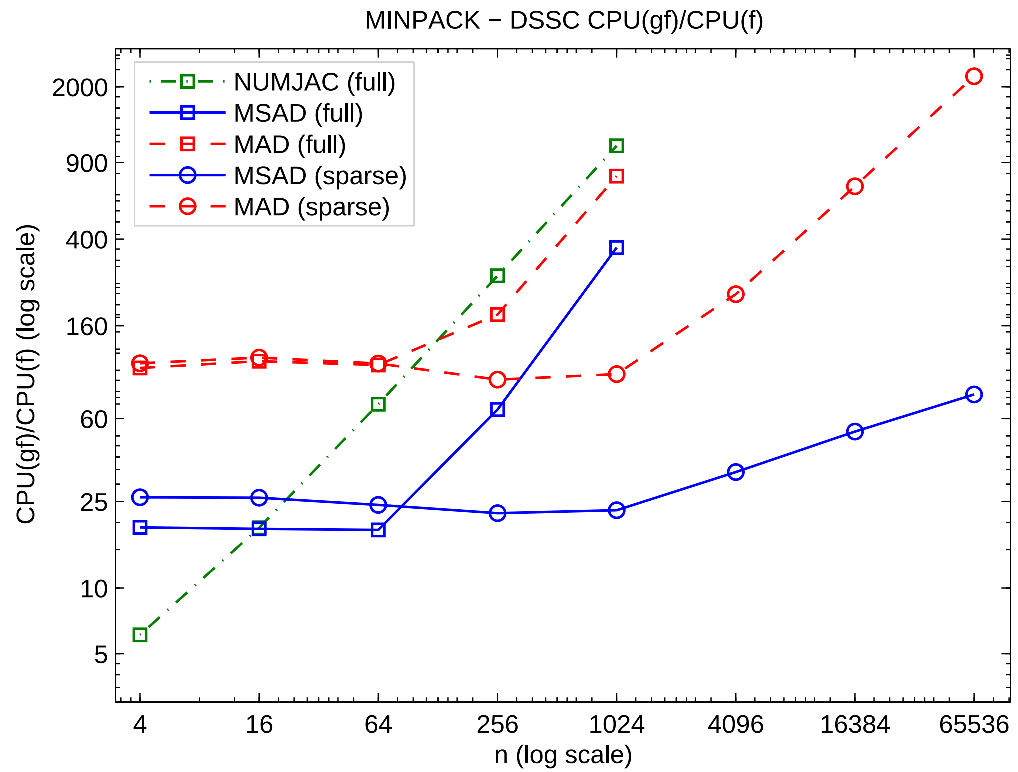
- Significantly better performance using Jacobian *compression* compared to other methods and to `numjac`, MAD and the previous approach using *compression*, even for large n
- MSAD using *full* evaluation of the Jacobian performs well compared to MAD and `numjac` using *full*
- Decrease in relative performance with increasing n, when using *sparse* derivatives.

Observations

- Significantly better performance using Jacobian *compression* compared to other methods and to `numjac`, MAD and the previous approach using *compression*, even for large n
- MSAD using *full* evaluation of the Jacobian performs well compared to MAD and `numjac` using *full*
  - When using the *full* or the *compressed* mode, the generated code contains only native data-types qualifying it for any *MATLAB JIT-Acceleration*
- Decrease in relative performance with increasing n, when using *sparse* derivatives.
  - This can be attributed to the larger *overheads in* manipulating the *internal sparse representation* of a matrix, making any savings relatively small

Results - MINPACK problems

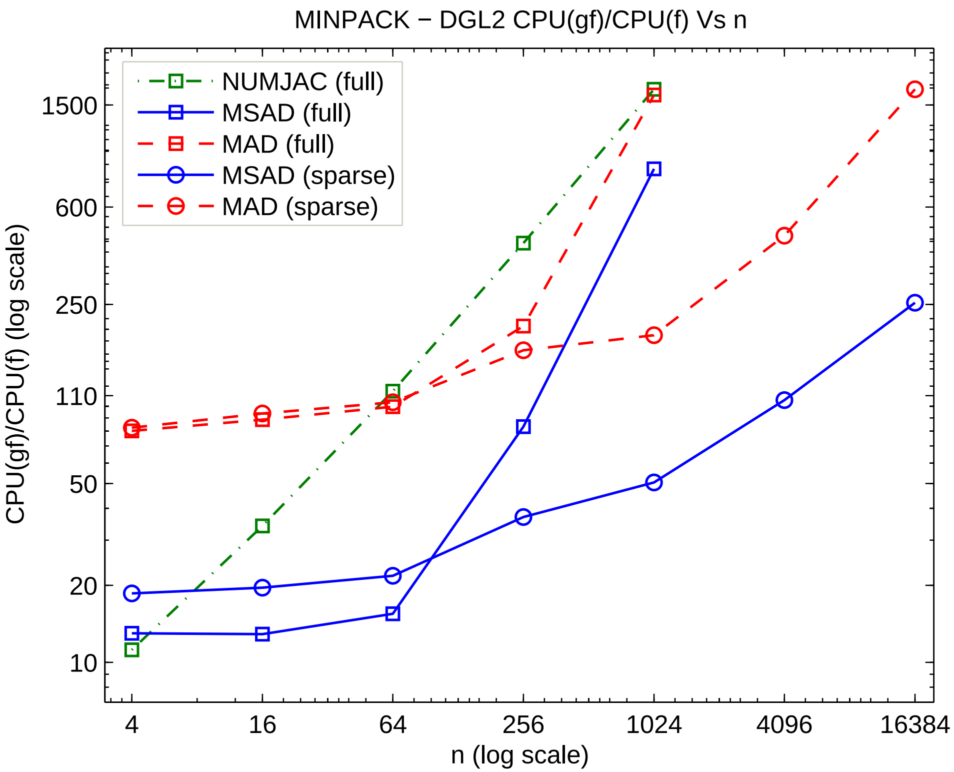


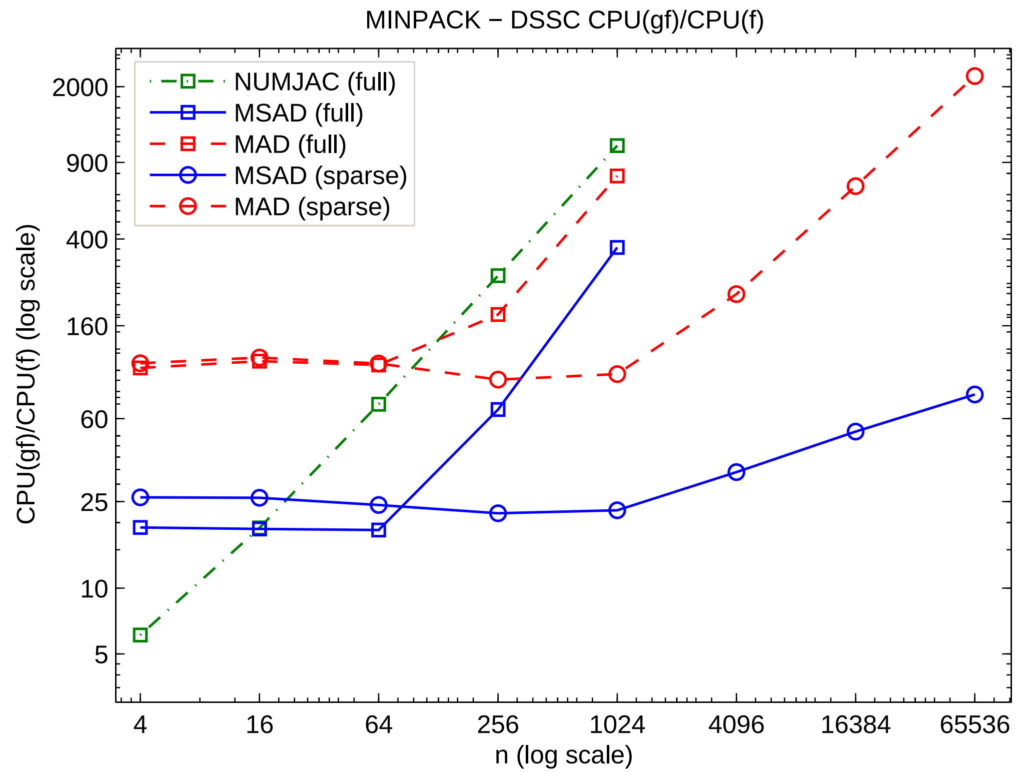


Results from *2-D Ginzburg-Landau* and *Steady-state combustion* problems

- using *full* derivatives to evaluate the gradient shows $80\% \rightarrow 50\%$ improvement over MAD, and outperforms `numjac` by a similar margin over medium and large n

Results - MINPACK problems

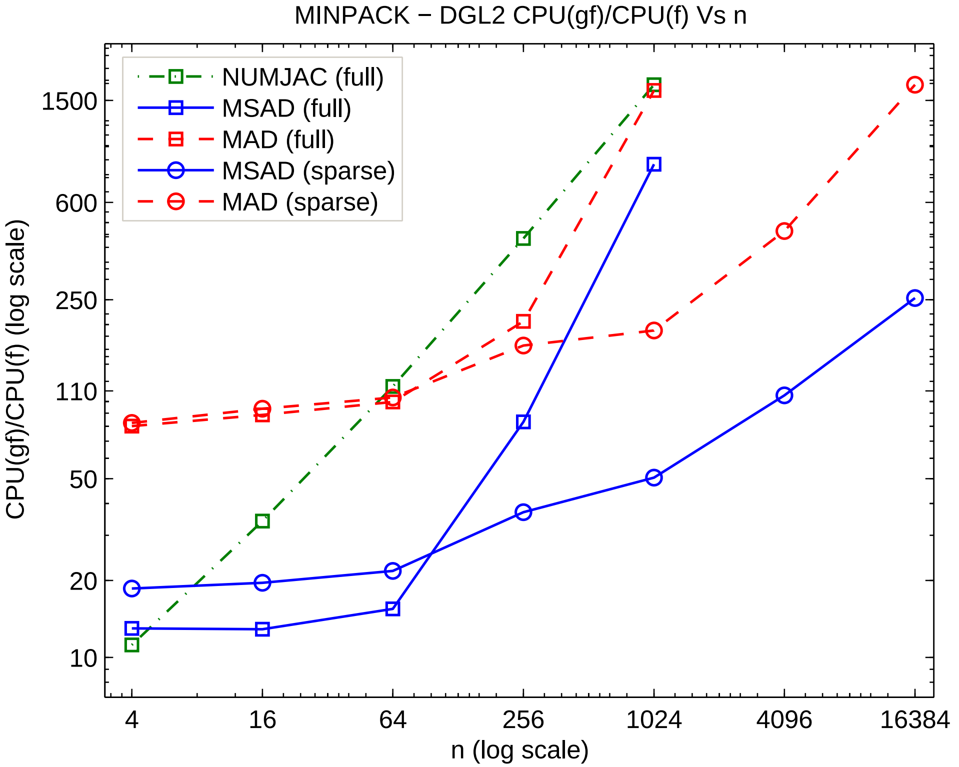


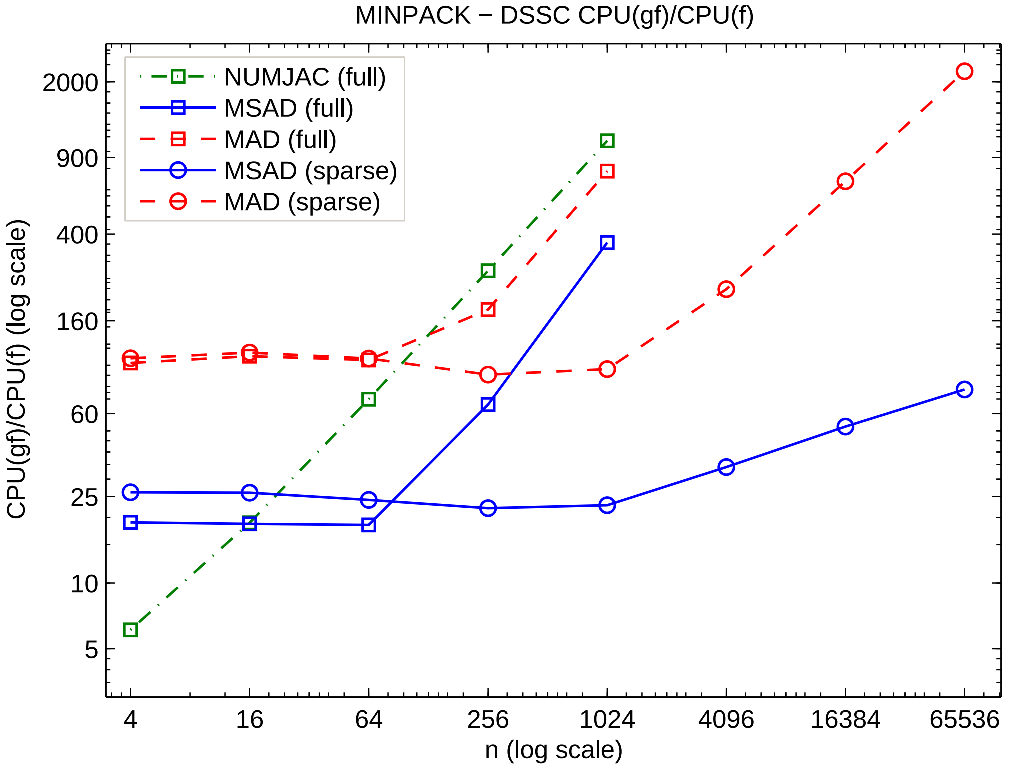


Results from *2-D Ginzburg-Landau* and *Steady-state combustion* problems

- using *full* derivatives to evaluate the gradient shows $80\% \rightarrow 50\%$ improvement over MAD, and outperforms `numjac` by a similar margin over medium and large n
- using *sparse* derivatives shows a vast improvement over MAD here, $75\% \rightarrow 85\%$

Results - MINPACK problems





Results from *2-D Ginzburg-Landau* and *Steady-state combustion* problems

- using *full* derivatives to evaluate the gradient shows $80\% \rightarrow 50\%$ improvement over MAD, and outperforms `numjac` by a similar margin over medium and large n
- using *sparse* derivatives shows a vast improvement over MAD here, $75\% \rightarrow 85\%$
  - caused by *redundant computations* involving some inactive intermediates treated as active in MAD [For04, 7-8]

Results - Smaller problems

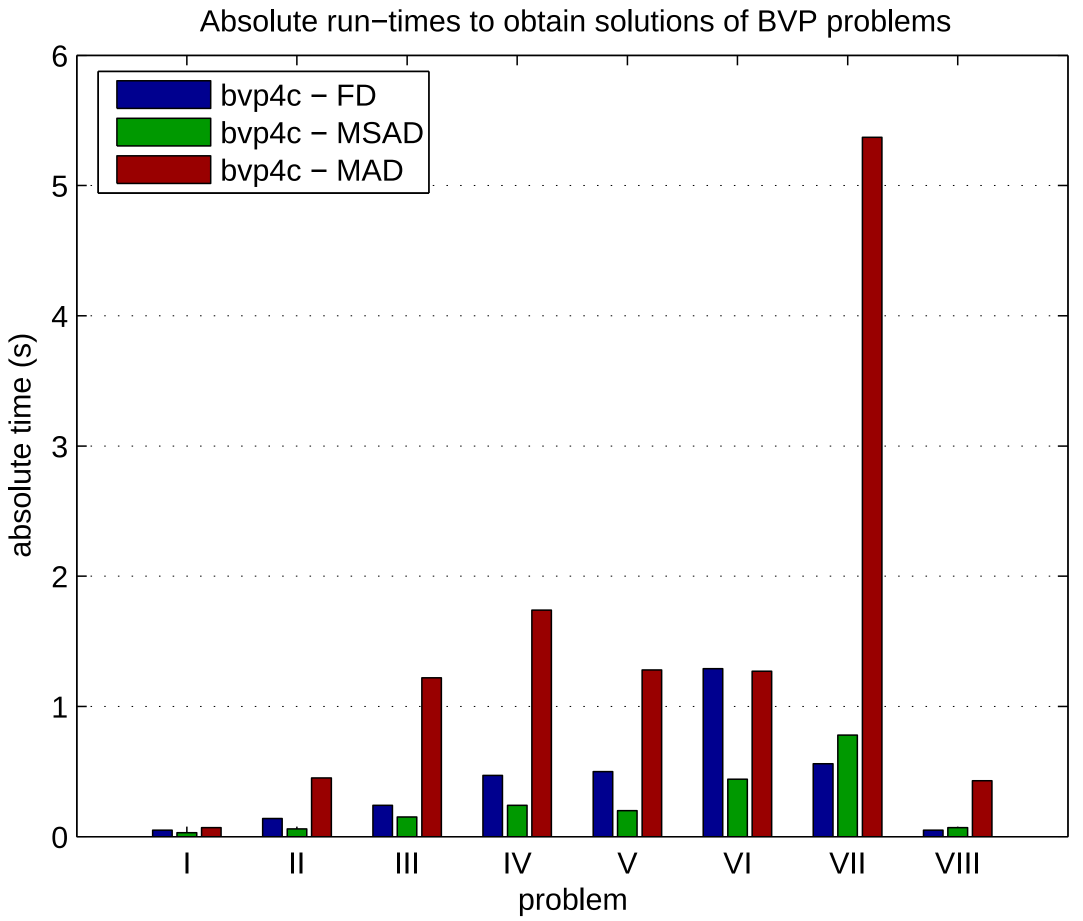
Problem	n	Ratio CPU($\mathbf{Jf}$)/CPU($\mathbf{f}$) numjac	MSAD	MAD
Coating Thickness Standardization	134	256.28	49.65	107.87
Pollution ODE	20	10.86[a]	9.84	113.37
Combustion of Propane - Full	11	22.22	35.03	394.29
Human Heart Dipole	8	23.12	53.08	737.16
Chemical AzkoNobel	6	16.24	17.22	252.71
Combustion of Propane - Reduced	5	20.67	64.94	921.80
Amplifier DAE	5	13.86	15.08	170.11
Enzyme Reaction	4	18.69	9.51	111.89
Robertson ODE	3	11.05	10.48	124.22

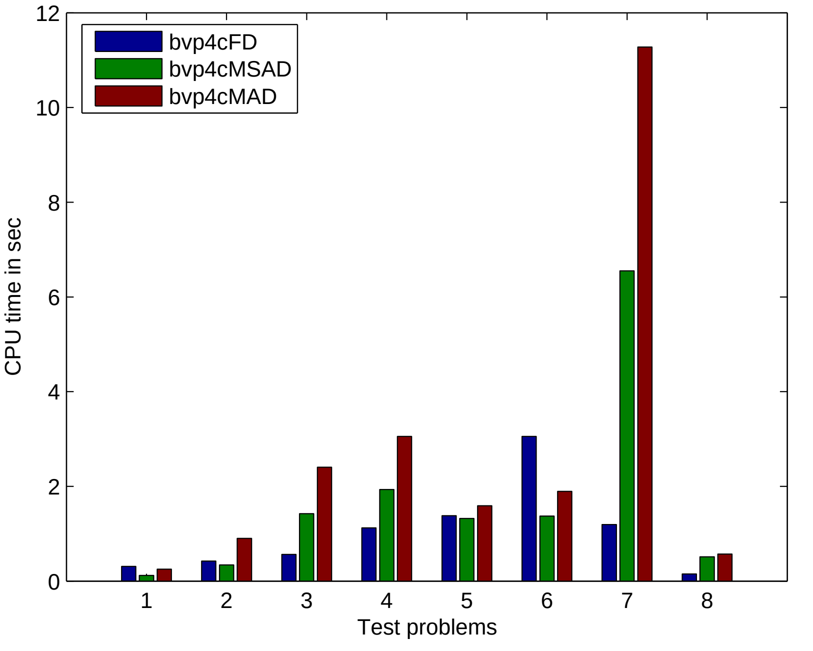
Smaller sized problems from `MINPACK`, `Test set for IVPs` and `MATLAB ODE examples`

- almost all cases show an order of magnitude speedup over MAD
- performance is fairly close to that of finite-differencing(`numjac`), in four cases better

[a] function vectorised to the advantage of numjac

Results - bvp4cAD





Results from `bvp4c` using MSAD (previous results on the right)

- significant speedup over previously adopted hybrid approach in MSAD
- performance better than using `numjac`[a] in six of eight cases, and comparable otherwise

[a] note the improved speed using `numjac` compared to earlier results (previously MATLAB 6.5 was used)

Summary

- MSAD shows *definite improvement* in *full* and *compressed* Jacobian evaluation over MAD and `numjac`
  - order of magnitude speedup in small and medium sized test cases
- In problems with sparsity in the derivatives of results or intermediates, using sparse derivatives in MAD and MSAD shows a *large saving* over the *full* evaluation of gradients/Jacobian
- In general, MSAD shows only a *constant saving* over MAD using *sparse* derivatives. In certain cases larger gains may be obtained
- Use of only native data types in the output code allows *MATLAB JIT* to perform some *run-time optimisations*

Future Directions

- Feature enhancement
  - Support for branching constructs involving active variables
  - Handle cells and structures
  - Incorporate exception handling to trap non-differentiability and syntactic errors
- Improving performance
  - Optimising generated code using dependency analysis (CFG, call-graphs)
  - Use more refined shape inference techniques
  - Apply constant folding
- Testing
  - Include a mechanism for systematic testing
  - Construct a comprehensive test suite

References

[ASU86] A.V. Aho, R. Sethi, and J.D. Ullman. *Compilers, Principles, Techniques, and Tools*. Computer Science. Addison-Wesley, Reading, Massachusetts, 1986.

[Eat02] J.W. Eaton. GNU Octave – a high-level language for numerical computations. *http://www.octave.org*, 2002.

[For04] S.A. Forth. An efficient overloaded implementation of forward mode automatic differentiation in MATLAB. Technical report, Cranfield University (RMCS Shrivenham), Swindon, UK, June 2004.

[Gri00] A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 2000.

[Kha04] R. Kharche. Source transformation for AD in MATLAB. Masters thesis, AMOR, Cranfield University, Shrivenham, UK, 2004.

[SKF03] L.F. Shampine, R. Ketzscher, and S.A. Forth. Using AD to solve BVPs in MATLAB. Technical report, Cranfield University (RMCS Shrivenham), Swindon, UK, 2003.

[Veh01] A. Vehreschild. Semantic augmentation of MATLAB programs to compute derivatives. Diploma thesis, Institute for Scientific Computing, Aachen University, Germany, 2001.

[Ver98] A. Verma. *Structured Automatic Differentiation*. PhD thesis, Cornell University, May 1998.