

DSJM: A Software Toolkit for Direct Determination of Sparse Jacobian Matrices*

Mahmudul Hasan¹, Shahadat Hossain¹, Trond Steihaug²

¹ University of Lethbridge, Department of Mathematics and Computer Science
Lethbridge, AB T1K 3M4, Canada

shahadat.hossain@uleth.ca, hasan@uleth.ca

² University of Bergen, Department of Informatics
Bergen, N-5020 Norway
Trond.Steihaug@ii.uib.no

Abstract. We describe DSJM, a software toolkit written in portable C++ that enables the direct determination of sparse Jacobian matrices whose sparsity pattern is a priori known. Using the seed matrix $S \in \mathbb{R}^{n \times p}$ defined by the partitioning information the nonzero unknowns of $A \in \mathbb{R}^{m \times n}$ can be obtained in $AS = B$ where $B \in \mathbb{R}^{m \times p}$ has been obtained via finite difference approximation or forward automatic differentiation. Our preliminary implementation includes well-known as well as new column ordering heuristics. Early numerical testing is highly promising both in terms of running time and the number of matrix-vector products needed to determine A .

Keywords. Structural Orthogonality, Direct Determination, Software Implementation.

1 Efficient Software Implementation for Direct Determination of Sparse Jacobian Matrices

The determination of a sparse Jacobian matrix with a priori known sparsity pattern of at least once continuously differentiable mapping $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ can be viewed as a computation of p matrix-vector products $AS \equiv B$ where A is an approximation of the Jacobian matrix $F'(x)$ but has the same sparsity pattern and $S \in \mathbb{R}^{n \times p}$ is a seed matrix of p directions. The nonzero entries of A are obtained using finite difference approximation

$$\left. \frac{\partial F(x + ts)}{\partial t} \right|_{t=0} = F'(x)s \approx As = \frac{1}{\varepsilon} [F(x + \varepsilon s) - F(x)] \equiv b, \quad (1)$$

or via the forward mode of automatic differentiation at a computational effort equal to a small multiple of the computational effort for evaluating F at x . If the ρ_i rows of S defined by the ρ_i nonzero unknowns of $A(i, :)$ are linearly independent then $A(i, :)$ is uniquely determined from $B(i, :)$. We have *direct determination* if the nonzero unknowns can be recovered from B without any extra arithmetic operations. The Curtis, Powell, and Reid method, henceforth the CPR [2] method, exploits A 's sparsity to define $S(:, k) = \sum_{j \in \mathcal{C}_k} e_j$, $k = 1, \dots, p$ where $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_p\}$ is a partition of column indices such that for each pair of indices $j, l \in \mathcal{C}_k$, the product $a_{ij}a_{il}$, $i = 1, \dots, m$ is identically 0, thereby yielding direct determination of the unknowns. The DSJM currently implements the well-known column ordering algorithms “largest first order (LFO)”, “smallest last order (SLO)”, and “incidence degree order (IDO)”, as well as a new column partitioning algorithm M(atrix)R(LF) based on “recursive largest first (RLF)” heuristic [8] for direct determination of sparse Jacobian matrices. Inspired by the highly acclaimed software DSM [1] our implementation uses general purpose data structures compressed row storage (CRS) and compressed column storage (CCS) for input matrices emphasizing efficiency and simplicity of use. Results from preliminary numerical experiments are given in Tables 1 and 2.

Table 1 displays timing results for selected large problems [4] (m rows, n columns, nnz nonzero entries) in seconds obtained on an IBM PC with 2.8 GHz Intel Pentium CPU, 1 GB RAM, and 512 KB L2 cache running Linux. The timing shown is that of IDO column ordering (ot) and greedy partitioning (pt) [5]. For both CoPack and DSJM the column ordering step is computationally more expensive than the greedy partitioning for almost all the test problems. This is expected since

* This research was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Research Council of Norway (NFR).

Table 1. Timing Results for IDO Partitioning.

<i>Matrix</i>	<i>m</i>	<i>n</i>	<i>nnz</i>	ColPack [5]		DSJM	
				ot	pt	ot	pt
lpcreb	9648	77137	260785	13.9	1.49	2.46	1
lpcred	8926	73948	246614	14.79	1.48	2.46	1.01
lpfit2d	25	10524	129042	64.19	24.94	16.32	17.2
lpken18	105127	154699	358171	15.92	0.63	1.47	0.48
lposa07	1118	25067	144812	161.76	28.34	40.84	18.35

Table 2. Partitioning Results.

<i>Matrix</i>	<i>m</i>	<i>n</i>	<i>nnz</i>	ρ	ColPack[5]	DSJM
af23560	23560	23560	484256	21	41 (SLO)	37 (MRLF)
cage11	39082	39082	559722	31	62 (SLO)	54 (MRLF)
cage12	130228	130228	2032536	33	68 (SLO)	56 (MRLF)

the ordering step requires the calculation and updating of column “degrees”. A similar performance profile is exhibited by the ordering algorithms LFO and SLO.

In Table 2 we provide partitioning results for a selection of problems [4] where there is a large gap between the maximum number of nonzero entries in any row (ρ) and the number of groups in the partition (p). In the parentheses we provide the ordering that gives the smallest partition. As evident from Table 2, SLO consistently produces the best ordering for ColPack while MRLF is the winner for DSJM. Also for other problems (not reported here) MRLF, on average, yields the best partition among the ordering heuristics considered.

We note that both the DSJM and the bipartite implementation ColPack require $\Theta(nnz)$ storage – thus achieving an important design objective identified in [7] for solving large and sparse problems.

2 Concluding Remarks

DSJM is being developed to address the need for efficient software toolkits that are simple and intuitive to use. The current implementation provides a collection of stand-alone column ordering and partitioning algorithms and driver routines for efficient direct determination of sparse Jacobian matrices. Future extension would include a user-friendly interface to automatic differentiation software and MATLAB [3]. Further, we envisage incorporation of substitution and elimination techniques [6] in addition to direct determination methods currently available.

References

1. T. F. Coleman, B. S. Garbow, and J. J. Moré. Software for estimating sparse Jacobian matrices. *ACM Trans. Math. Software*, 10(3):329–345, 1984.
2. A. R. Curtis, M. J. D. Powell, and J. K. Reid. On the estimation of sparse Jacobian matrices. *J. Inst. Math. Appl.*, 13:117–119, 1974.
3. Shaun Forth. The MAD (MATLAB Automatic Differentiation) Project, <http://www.amorg.co.uk/AD/MAD/> (accessed May 2009).
4. A. H. Gebremedhin, F. Manne, and A. Pothen. What color is your Jacobian? Graph coloring for computing derivatives. *SIAM Rev.*, 47(4):629–705, 2005.
5. A. H. Gebremedhin, A. Tarafdar, D. Nguyen, and A. Pothen. ColPack. <http://www.cs.odu.edu/~dnguyen/dox/colpack/html/> (accessed May 2009)
6. S. Hossain and T. Steihaug. Sparsity Issues in the Computation of Jacobian Matrices. In T. Mora, editor, *Proceedings of the International Symposium on Symbolic and Algebraic Computing (ISSAC)*, pages 123–130, New York, 2002. ACM.
7. J. R. Gilbert, S. Reinhardt, and V. Shah. High-performance graph algorithms from parallel sparse matrices. In B. Kågström, E. Elmroth, J. Dongarra, and J. Wasniewski, editors, *PARA*, volume 4699 of *Lecture Notes in Computer Science*, pages 260–269. Springer, 2006.
8. F. T. Leighton. A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–506, 1979.