NHH

INSTITUTT FOR SAMFUNNSØKONOMI

DEPARTMENT OF ECONOMICS

SAM 20 2010

ISSN: 0804-6824

AUGUST 2010

Discussion paper

Computing the Jacobian in spatial models: an applied survey

BY

ROGER BIVAND

Norges Handelshøyskole

NORWEGIAN SCHOOL OF ECONOMICS AND BUSINESS ADMINISTRATION

Computing the Jacobian in spatial models: an applied survey*

Roger Bivand
Norwegian School of Economics and Business Administration†

August 17, 2010

Abstract

Despite attempts to get around the Jacobian in fitting spatial econometric models by using GMM and other approximations, it remains a central problem for maximum likelihood estimation. In principle, and for smaller data sets, the use of the eigenvalues of the spatial weights matrix provides a very rapid and satisfactory resolution. For somewhat larger problems, including those induced in spatial panel and dyadic (network) problems, solving the eigenproblem is not as attractive, and a number of alternatives have been proposed. This paper will survey chosen alternatives, and comment on their relative usefulness.

1 Introduction

Spatial regession models are fitted in a wide range of disciplines, from political and regional science to epidemiology and ecology. In many cases, maximum likelihood methods are chosen for fitting, but problems can arise when data sets become large. The ways in which the fitted models are conceptualised also impact their interpretation, as misspecification is a reccurring problem.

*Thanks to particpants at sessions at: II World Conference of the Spatial Econometrics Association, New York; Eighth Spatial Econometrics and Statistics Workshop, Besançon, France; 49th European Congress of the Regional Science Association International, Łódź, Poland; 5th Nordic Econometric Meeting, Lund, Sweden, and a research seminar at the Institute for Economic Geography and GIScience, Vienna University of Economics and Business.

†Department of Economics, Norwegian School of Economics and Business Administration, Helleveien 30, N-5045 Bergen, Norway; E-mail: <Roger.Bivand@nhh.no>

Having defined the spatial regression models to be treated here, we will start by considering the efficient computation of the log determinant of a possibly sparse real symmetric positive definite matrix is necessary.[1] One of these is in finding the values of the log likelihood function for various spatial regression models, where the underlying sparse matrix of spatial weights represents a graph of relationships between observations. For small numbers of observations, there are no difficulties in treating the spatial weights matrix as dense, and computing the log determinant using its eigenvalues.

The initial intention in this paper was to discuss in detail developments in the computation of such log determinants using Cholesky factorization in the **Matrix** package for R. However, since Walde et al. (2008) have undertaken a broader comparison of different methods and approximations for computing the Jacobian as part of a "contest" between maximum likelihood and generalized method of moments model fitting methods, we will, in addition, take up a number of their conclusions, and try to qualify them in the light of further analysis.[2]

After starting by describing the log likelihood function for a standard spatial regression model, and explaining the eigenvalue approach, the paper turns to a discussion of sparse matrix methods. This is followed by a short review of some approximations to the log determinant. The review will also be set in the context of the detailed presentation of log determinant computation by LeSage and Pace (2009).

In the study of the Jacobian, we will use six data sets of neighbour relationships (spatial weights): Queen contiguities between 3111 US counties (sharing at least one common boundary point); a complete 4900 observation grid as defined by Walde et al. (2008, p. 157); Queen contiguities between cells on a 1° grid for world land areas omitting Antarctica with 15260 observations; sphere-of-influence neighbours for 25357 houses sold in Lucas county, Ohio[3]; Queen contiguities between 32698 US 2000 Census Zip Code Tabulation Areas (ZCTA, omitting Alaska and Hawaii); and Queen contiguities between 64878 US Census tracts in 2000 (omitting Alaska and Hawaii).

Only the 4900 observation grid has a single connected graph; the other data sets have multiple disjoint connected subgraphs, including islands with no neighbours, as shown in Table 1, which also shows the distribution of neighbours. The large number of subgraphs in the Lucas county housing data is caused by the frequent

[1]This paper is based on extended discussions with Douglas Bates and Martin Mächler, and made possible by helpful additions to the R **Matrix** package; they bear no resposibility for any remaining misunderstandings on the part of the author.

[2]Following helpful collaboration with Janette Walde in throwing light on a number of counter-intuitive conclusions in that paper, we understand that a correction to their paper will be submitted.

[3]Dataset included in the Spatial Econometrics toolbox for Matlab, `http://www.spatial-econometrics.com/html/jplv7.zip`, temporal ordering disregarded here, coordinates transformed to Ohio North State Plane.

Table 1: Distributions of numbers of neighbours and numbers of subgraphs for data sets.

	US Counties	Grid 70x70	World grid	Lucas OH	US ZCTA	US Tracts
0	4	0	7	0	229	17
1	31	0	80	3098	634	418
2	40	0	147	7228	1778	1098
3	96	0	274	7280	2712	2940
4	289	0	507	4587	4144	7080
5	621	0	907	2130	5593	12343
6	1047	0	752	809	5506	14311
7	694	0	1013	175	4478	12158
8	222	4	11573	48	2937	7552
9	53	0	0	1	1689	3644
10	10	0	0	1	1015	1711
11	2	8	0	0	612	767
12	0	0	0	0	356	377
13	1	0	0	0	202	178
14	1	264	0	0	107	124
15	0	4	0	0	74	55
16	0	0	0	0	50	39
17	0	0	0	0	41	19
18	0	0	0	0	30	12
19	0	264	0	0	32	12
20	0	0	0	0	27	8
21	0	0	0	0	30	3
22	0	0	0	0	25	1
23	0	0	0	0	21	1
24	0	4356	0	0	23	4
25	0	0	0	0	25	2
26	0	0	0	0	13	1
27	0	0	0	0	15	1
28	0	0	0	0	20	0
29	0	0	0	0	20	1
30	0	0	0	0	12	1
subgraphs	6	1	49	1481	243	30

occurrence of pairs of houses that are much closer to each other than to any other house; the dataset does not include repeat sales. The very large numbers of contiguous numbers of neighbours for some observations in the two latter data sets is caused by the way these observations are structured, for example central entities with radiating neighbours, such as Central Park in New York (the entity in the Census tract data set with 30 neighbours[4]). In the ZCTA data set, all the entities with over 26 neighbours are three-digit codes with either XX or HH suffixes, indicating parks, forest lands etc., or water bodies. The selection of data sets is similar to those chosen by Smirnov and Anselin (2001).

2 Spatial regression models

Assuming that the variance of the disturbance term is constant, we start from the standard linear regression model:

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon, \qquad \varepsilon \sim N(0, \sigma^2)$$

There are a number of alternative forms of spatial regression models; here we will consider the simultaneous autoregressive (SAR) form, because the computation of the Jacobian presents similar challenges for the conditional autoregressive and spatial moving average representations. The SAR model may be written as (Cliff and Ord, 1973; Ord, 1975; Ripley, 1981):

$$\mathbf{y} = \mathbf{X}\beta + \mathbf{u}, \qquad \mathbf{u} = \lambda\mathbf{W}\mathbf{u} + \varepsilon,$$

where $\mathbf{y}$ is an $(N \times 1)$ vector of observations on a dependent variable taken at each of N locations, $\mathbf{X}$ is an $(N \times k)$ matrix of exogenous variables, β is an $(k \times 1)$ vector of parameters, ε is an $(N \times 1)$ vector of disturbances and λ is a scalar spatial error parameter, and $\mathbf{u}$ is a spatially autocorrelated disturbance vector with constant variance and covariance terms specified by a fixed spatial weights matrix and a single coefficient λ:

$$\mathbf{u} \sim N(0, \sigma^2(\mathbf{I} - \lambda\mathbf{W})^{-1}(\mathbf{I} - \lambda\mathbf{W}')^{-1})$$

It is usual in the literature to define the contiguity relation in terms of sets $N_{(i)}$ of neighbours of zone or site i. These are coded in the form of a weights matrix $\mathbf{W}$, with a zero diagonal, and the off-diagonal non-zero elements often scaled to sum to unity in each row (termed row standardized weights matrices), with typical elements:

[4] See `http://factfinder.census.gov/jsp/saff/SAFFInfo.jsp?_pageId=gn7_maps`

$$w_{ij} = \frac{c_{ij}}{\sum_{j=1}^{N} c_{ij}}$$

where $c_{ij} = 1$ if i is linked to j and $c_{ij} = 0$ otherwise. This implies no use of other information than that of neighbourhood set membership. Set membership may be defined on the basis of shared boundaries, of centroids lying within distance bands, or other a priori grounds. In general, the number of neighbours for each observation will be small compared to N, so that $\mathbf{W}$ is usually sparse. It may be reasonable, based on knowledge of the underlying spatial interaction processes, to specify c_{ij} in other way, for example trade or migration flows, or in other ways that introduce asymmetry. Indeed, the spatial weights defined here by row-standardisation are asymmetric, but if $c_{ij} = c_{ji}$, the matrix is similar to a symmetric matrix.

Ord (1975) gives a maximum likelihood method for estimating the spatial error SAR model. Unlike the time series case, the logarithm of the determinant of the $(N \times N)$ matrix $(\mathbf{I} - \lambda\mathbf{W})$ does not tend to zero with increasing sample size; it constrains the parameter values to their feasible range between the inverses of the smallest and largest eigenvalues of $\mathbf{W}$. For positive autocorrelation, as $\lambda \rightarrow 1/\max_i(\zeta_i)$ — ζ_i are the eigenvalues of $\mathbf{W}$, $\ln|\mathbf{I} - \lambda\mathbf{W}| \rightarrow -\infty$. The log-likelihood function for the spatial error model:

$$\ell(\beta, \lambda, \sigma^2) = -\frac{N}{2}\ln 2\pi - \frac{N}{2}\ln\sigma^2 + \ln|\mathbf{I} - \lambda\mathbf{W}|$$
$$-\frac{1}{2\sigma^2}\left[(\mathbf{y} - \mathbf{X}\beta)'(\mathbf{I} - \lambda\mathbf{W})'(\mathbf{I} - \lambda\mathbf{W})(\mathbf{y} - \mathbf{X}\beta)\right]$$

β may be concentrated out of the sum of squared errors term, for example as:

$$\ell(\lambda, \sigma^2) = -\frac{N}{2}\ln 2\pi - \frac{N}{2}\ln\sigma^2 + \ln|\mathbf{I} - \lambda\mathbf{W}|$$
$$-\frac{1}{2\sigma^2}\left[\mathbf{y}'(\mathbf{I} - \lambda\mathbf{W})'(\mathbf{I} - \mathbf{Q}_\lambda\mathbf{Q}_\lambda')(\mathbf{I} - \lambda\mathbf{W})\mathbf{y}\right]$$

where $\mathbf{Q}_\lambda$ is obtained by decomposing $(\mathbf{X} - \lambda\mathbf{W}\mathbf{X}) = \mathbf{Q}_\lambda\mathbf{R}_\lambda$.

As we can see, the problem is one of balancing the log determinant term against the sum of squares term. When λ approaches the ends of its feasible range, the log determinant term may swamp the sum of squares term (Bivand, 1984).

With moderate to large N, the calculation of the analytical variance-covariance matrix of the model coefficients is impeded by the need to handle dense $N \times N$

matrices. The variance-covariance matrix may be approximated by a numerical Hessian, the computation of which also involves the Jacobian. Typically, the number of calls of the Jacobian function in this step is larger than in the line search to find λ at the optimum.

3 Computing the Jacobian

The first published versions of the eigenvalue method for finding the Jacobian (Ord, 1975, p. 121) present it in product form:

$$\log(|\mathbf{I} - \lambda\mathbf{W}|) = \log(\prod_{i=1}^{N}(1 - \lambda\zeta_i))$$

instead of the equivalent summation form:

$$\log(|\mathbf{I} - \lambda\mathbf{W}|) = \sum_{i=1}^{N}\log(1 - \lambda\zeta_i)$$

where ζ_i are the eigenvalues of $\mathbf{W}$. In the product form, it may become difficult to compute, since the value of the determinant may underflow (become indistinguishable from zero before taking the logarithm) if care is not shown.

One specific problem addressed by Ord (1975, p. 125) is that of the eigenvalues of the asymmetric row-standardised matrix W with underlying symmetric neighbour relations $c_{ij} = c_{ji}$. If we write $\mathbf{w} = \mathbf{C1}$, where $\mathbf{1}$ is a vector of ones, we can get: $\mathbf{W} = \mathbf{CD}$, where $\mathbf{D} = \mathrm{diag}(1/\mathbf{w})$. By similarity, the eigenvalues of $\mathbf{W}$ are equal to those of: $\mathbf{D}^{\frac{1}{2}}\mathbf{C}\mathbf{D}^{\frac{1}{2}}$. Of course, if the underlying neighbour relations are not symmetric, the eigenvalues of $\mathbf{W}$ will not necessarily be real; the consequences of using such asymmetric weights matrices are not known (Smirnov and Anselin, 2001, p. 303–304). The handling of intrinsically asymmetric weights matrices is also discussed by LeSage and Pace (2009, pp. 88–89).

In addition to choices with regard to the underlying neighbour relations used to structure covariance between observations, by no means all applications use row standardisation of spatial weights matrices. Row standardisation has the convenient consequence that the largest eigenvalue of $\mathbf{W}$ is known to be equal to one by design; the value of the smallest eigenvalue is unknown, but in line search for λ, the relevant interval is often taken as $[0, 1)$. However, row standardisation upweights neighbour relations for observations with few neighbours, and downweights relations for those with many neighbours. Tiefelsdorf et al. (1999) propose a variance-stabilising scheme instead of row standardisation, which for underlying symmetric neighbour

Table 2: Lower and upper interval bounds on λ, 3111 US counties Queen contiguity data set, for binary weights (B), binary weights scaled to sum to N (C), variance-stabilising weights (S) — real part only, row standardised weights (W) — real part only, and variance-stabilising weights (S (sym)) and row standardised weights (W (sym)) transformed to symmetry by similarity.

	B	C	S	W	S (sym)	W (sym)
lower	-0.2932	-1.7179	-1.8026	-1.0000	-1.8026	-1.0000
upper	0.1489	0.8724	0.9374	1.0000	0.9374	1.0000

relations also yields an asymmetric spatial weights matrix that is similar to symmetric. Further discussion of these issues may be found in Bivand et al. (2008); Ward and Gleditsch (2008).

Many disciplines using spatial regression methods simply use unstandardised neighbour relations matrices which may or may not be symmetric. Table 2 shows the lower and upper bounds for λ for the same set of symmetric contiguous neighbours for 3111 US counties under different weights representations. The underlying eigenvalues have been calculated using the R `eigen` function, using the standard LAPACK functions and with symmetry of the input matrix determined by the internal code. As can be seen, the intervals vary greatly, depending on choices of specification.

One point that needs to be taken forward from this discussion is that although, for SAR models, neither $\mathbf{W}$ nor $(\mathbf{I} - \lambda\mathbf{W})$ are required to be symmetric postive definite matrices, such an assumption makes computing the Jacobian more feasible. A second point is that a line search for λ without knowledge of the extreme eigenvalues of $\mathbf{W}$ should be able to recover from leaving the feasible range of λ (Smirnov and Anselin, 2001). There are obvious limits on N, because in general dense matrices have to be used to find the eigenvalues of $\mathbf{W}$, which impact both the use of eigenvalues in computing the Jacobian and in setting the search interval for λ.[5]

A consequence of this discussion is that implementation is of the essence, something that we feel is demonstrated by Walde et al. (2008)[6]. Using MATLAB Release 7 version 14 on a 70 slave processor Linux cluster, they undertake 3000 Monte Carlo runs pitting different fitting methods against each other. In fact, all the fitting methods except one are maximum likelihood with differing methods for computing the Jacobian. The simulation scenario is for a regular 4900 observation grid, a

[5]Since the line search interval for λ can be manipulated, so far little attention has been given to finding the extreme eigenvalues of sparse $\mathbf{W}$ for large N computationally; for some regular spatial observation designs, analytical eigenvalues are known (Griffith, 2000).

[6]We are grateful to Janette Walde for her willingness to clarify questions arising during our study, and for sharing code exerpts with us.

SAR process with a λ coefficient value of 0.5, an intercept of one and a uniform random x variable within zero and one and a coefficient of one; the remainder error is assumed to be normal with zero mean and a standard deviation of one. The final fitting method is generalised method of moments, which we will not be addressing here. By setting the contest between methods for computing the Jacobian inside model fitting simulation runs, it is not possible to see how well or poorly the actual Jacobian values are computed for varying values of λ, rather how well the optimisation technique performs in providing the Jacobian method in the function returning the log likelihood with proposed λ values — this raises concerns about whether the optimisation technique is not giving some methods of computing the Jacobian a fair chance.

With regard to the Ord eigenvalue method for computing the Jacobian, Walde et al. (2008) conclude that it fails dramatically for $N = 4900$, when in principle for a fixed $\mathbf{W}$, the eigenvalues are also fixed, and consequently any variation in their Monte Carlo runs cannot be coming from this source. Their numerical results suggest that the line search for λ often halted at its lower bound, naturally leading to poor performance; this diagnosis has been confirmed in correspondence with the authors. In the next section, we present comparative results for computing the Jacobian using the eigenvalue method, and do not find that they diverge from the values from sparse matrix methods for moderate N.

4 Sparse Matrix methods

When spatial regression models began to be taken up in applied research, hardware constraints on computing eigenvalues for moderate N prompted work on alternative methods for computing the Jacobian. In a series of contributions, Pace and Barry (1997b,c,a) show that sparse matrix methods can be used to find the log determinant directly.[7] The method of choice is the Cholesky decomposition of a sparse, symmetric, positive-definite matrix, but can be extended to the LU decomposition if requirements on the matrix need to be relaxed (Smirnov and Anselin, 2001). Naturally, for the same sparse, symmetric, positive-definite matrix, one would expect the log determinants based on the Cholesky decomposition and the LU decomposition to be identical within machine precision.

Walde et al. (2008) find, by implication, that the Jacobian values from Cholesky decomposition and the LU decomposition for the same $(\mathbf{I} - \lambda\mathbf{W})$ matrix differ.[8] Both Cholesky and LU decomposition implementations are provided in the **Matrix** package in R, so we will use these to match the Jacobian values based on the

[7] The S-PLUS SpatialStats module also uses sparse matrix methods (Kaluzny et al., 1998).

[8] Correspondence with Janette Walde, who made code extracts available, indicates that the Cholesky Jacobian was erroneously divided by 2, explaining the discrepancy.

eigenvalue method for two sets of neighbours, and against each other for the four larger sets in the range $[-0.9, 0.99]$, in steps of 0.01, extending a little the values used by Walde et al. (2008) for their grid search.[9] The analysis has been carried out on an Intel Core-2 Duo 64-bit system with 4GB RAM running R 2.11.1, **Matrix** 0.999375-43, **spam** 0.22-0, and **spdep** 0.5-19, under Red Hat Enterprise Linux 5; a threaded GotoBLAS 1.26 library optimised for the hardware was used, with gfortran 4.1.2 for Fortran compilation.

From the 0.5 release of the **spdep** package, Jacobian computation has been modularised to use two functions for each method, one to set up the objects needed for calculating the Jacobian, and a second called through the `do_ldet` function in each log likelihood objective function call. The objects prepared once only in the set up function are passed between the functions in an environment, avoiding the need to pass the objects themselves separately. The method being used is also assigned to the environment in the set up function.

The code used here is given in the appendix, and for each data set calculates Jacobian values for selected methods. The set up timings are recorded, as are timings for computing 190 Jacobian values. The timings for the set up are not an average of multiple runs, and are simply those observed on the platform used. The set up code for some methods is very simple, while for others it is more complex. For the eigenvalue method, the set up timings reflect the computation of the eigenvalues from the symmetric, or transformed to symmetric, weights matrix. In the latter case, transformation to the similar symmetric matrix is an extra time cost. Similar steps are required for set up for the other methods; the objects involved are described in the help page for `do_ldet` in **spdep** (reproduced at the end of this paper).

Our reasoning in this analysis is that the Jacobian value for the same λ may vary a little between methods and implementations of these methods, but should not differ so much that they lead the optimisation procedure to choose an inappropriate solution. As noted above, the discrepancy noted by Walde et al. (2008) was caused by a coding error. Table 3 shows timings for the methods we have presented so far: computing the eigenvalues of a symmetric representation of spatial weights and then using them to calculate the required Jacobian values; computing the Jacobian directly using Cholesky decomposition; and computing the Jacobian directly using LU decomposition. As LeSage and Pace (2009, p. 83) point out, one would expect the LU decomposition to take about twice as long as the Cholesky decomposition on the same matrix. The LU decomposition also seems to be seriously affected by the less sparse nature of the 70 by 70 grid data set.

The test setting is somewhat artificial, because candidate values of λ are proposed by the numerical optimisation function, typically a line search function, and

[9] Smirnov and Anselin (2001, p. 313) remark that a line search for λ is unlikely to need more than 50 evaluations of the Jacobian.

few model fitting runs need as many as 190 values. Indeed, the overhead of computing the sum of squared errors component of the log likelihood for each λ may be quite substantial. LeSage and Pace (2009, p. 48, and in numerous implementations in the Spatial Econometrics toolbox for Matlab) prefer to vectorise the computation of the Jacobian and log likelihood over a fine grid of values of λ.

In the implementations in the **Matrix** and **spam** packages, the `determinant` method for symmetric sparse matrix objects computes a Cholesky factorization of $(\mathbf{I} - \lambda\mathbf{W})$, and then extracts the determinant on the logarithm scale. The LU decomposition is undertaken in an analogous way. In the Cholesky case, if $\mathbf{W}$ can be transformed to symmetry by similarity, this is done during setup. In both of these approaches, the Cholesky factorization or LU decomostition and the extraction of the determinant is done for each value of λ.

As can be seen in Table 3, the Jacobian values for all methods agree within machine precision for the two smaller data sets.[10] Timings for the eigenvalue method are divided between setup and run times for 190 Jacobian values; for the remaining alternatives, total set up and run times are reported. Eigenvalue-based Jacobian values are not available for the four larger data sets, but the comparison of the Cholesky and LU decomposition Jacobian values shows that they are equal within machine precision. The LU decomposition is substantially more time-consuming, which, with the equality of the values of the Jacobian for given λ, appears to support the use of the Cholesky decomposition where possible.

The implementations of sparse Cholesky decomposition in the **Matrix** and **spam** packages are independent of each other, with the former using approximate minimal degree ordering, and the latter multiple minimum degree (MMD, default) or reverse Cuthill-McKee (RCM) pivoting. The implementation in the **Matrix** package provides simplicial or supernodal decomposition, which can be specified directly. In addition, a heuristic is provided in the CHOLMOD code used by **Matrix**, which chooses the prefered decomposition method automatically (here termed CHOLMOD). With the exception of the very sparse Lucas, OH data set, the simplicial decomposition is faster than the supernodal. For the denser 70×70 grid, the simplicial decomposition is twice as fast as using eigenvalues, while the supernodal is slower. The CHOLMOD-heuristic — called at each Jacobian calculation — appears to increase execution times, and is at best comparable to supernodal decomposition.

If the **spam** implementation is chosen, it seems important to avoid the reverse Cuthill-McKee (RCM) pivoting scheme, which only performs acceptably for the Lucas, OH data set. Using the default multiple minimum degree (MMD) pivoting scheme, this implementation is at best no better than the **Matrix** supernodal decomposition. Consequently, in model fitting using sparse Cholesky decomposition dur-

[10] Machine precision is taken as 1.49012e-08, as in `all.equal()` in R.

Table 3: Timings for computing 190 Jacobian values for $\lambda[-0.9, 0.99]$ using dense matrix eigenvalues, five sparse matrix Cholesky versions (**Matrix** simplicial and supernodal decompositions and decomposition chosen by a CHOLMOD-iternal heuristic, and **spam** using pivoting schemes MMD and RCM) and LU decomposition (**Matrix**); output Jacobian values are tested for equality within machine precision between methods.

	US Counties	Grid 70x70	World grid	Lucas OH	US ZCTA	US Tracts
Eigen setup	12.514s	44.035s				
Eigen	0.052s	0.066s				
Matrix simplicial	3.307s	21.434s	33.927s	29.108s	24.405s	96.071s
Eigen == **Matrix** simplicial	TRUE	TRUE				
Matrix supernodal	3.982s	50.612s	38.296s	18.817s	52.155s	103.821s
Matrix simplicial == **Matrix** supernodal	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Matrix CHOLMOD	3.249s	50.369s	31.085s	27.933s	49.723s	101.833s
Matrix simplicial == **Matrix** CHOLMOD	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
spam MMD	5.13s	49.305s	43.797s	28.857s	61.72s	111.462s
Matrix simplicial == **spam** MMD	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
spam RCM	22.939s	151.749s	110.747s	33.115s	482.055s	629.077s
Matrix simplicial == **spam** RCM	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
LU	8.525s	137.855s	50.16s	16.319s	110.717s	148.576s
Matrix simplicial == LU	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
N	3111	4900	15260	25357	32698	64878
Sparseness %	0.188070	0.472453	0.048073	0.011645	0.019177	0.009563

ing each Jacobian computation, the prefered choice would be **Matrix** with simplicial decomposition (`method="Matrix_J"`, default control option `super=FALSE`).

5 Updating Cholesky decompositions

A promising innovation for reducing the computational burden of computing the Jacobian in spatial regression models was introduced in the **Matrix** package in March 2008. A comparable facility was introduced into the **spam** package in June 2008. Since the pattern of sparseness in the matrix for which the log determinant is to be found does not change, it is possible to carry out the Cholesky decomposition once, and then update the values respecting the fill-reducing permutation found when the decomposition was first undertaken. This incurs a moderate setup cost, but speeds up the finding of each Jacobian value for successive λ proposed by the optimiser.

In **Matrix**, the `Cholesky` method computes the Cholesky decomposition of a sparse, symmetric, positive-definite matrix, permitting the user to choose among different kinds of sparse Cholesky decompositions (Davis, 2006). It returns an object extending `CHMfactor`, so that the `determinant` method used will be for the appropriate class. The `super=` argument is `FALSE` by default, leading to the use of a simplicial decomposition; when `TRUE`, a supernodal decomposition is created, or if set to `as.logical(NA)`, the CHOLMOD-heuristic is used to choose the decomposition method. The `Imult=` argument defaults to zero, but for our purposes needs to be larger than the maximum row sum of $\mathbf{W}$, and is here taken as 2. The matrix that is decomposed is $\mathbf{W}+m\mathbf{I}$ where m is the value of `Imult`, and for positive λ, we reverse the sign of $\mathbf{W}$.

Depending on the value of λ, we either return zero for λ within machine precision of zero, or switch on the sign of λ. The `update` method for `CHMfactor` objects takes as additional arguments the original `parent=` `dsCMatrix` matrix, and argument `mult=` taking values $1/\lambda$ for positive λ, or $1/(-\lambda)$ for negative λ. This value updates the numerical values of the decomposition as λ changes. The output value from `determinant` is multiplied by $N\log(\lambda)$ (or $-\lambda$ for $\lambda < 0$) yielding the Jacobian in logarithm form (see for a similar development, in the context of a characteristic polynomial approach Smirnov and Anselin, 2001, p. 307).

In the **spam** package, updating still requires the provision of $(\mathbf{I}-\lambda\mathbf{W})$ for each value of λ, but avoids repeated decompositions, using an initial decomposition made with the chosen pivoting method. Table 4 shows that the Jacobian values returned by the updating Cholesky decomposition methods are equal to those of the direct Cholesky method. The setup timings differ between the representations, with setup for the less sparse 70 by 70 grid data set taking markedly longer for simplicial than supernodal decomposition, and marginally for the **spam** MMD pivoting scheme.

The same seems to apply to the run times for finding 190 Jacobian values, with

Table 4: Comparison of updated Cholesky decompositions, using simplicial and supernodal decomposition and CHOLMOD-heuristic decomposition method choice from **Matrix** and updating from **spam** with two different pivoting schemes.

	US Counties	Grid 70x70	World grid	Lucas OH	US ZCTA	US Tracts
Simplicial (**Matrix**) setup	0.067s	0.211s	0.152s	0.105s	0.323s	0.571s
Simplicial (**Matrix**)	0.776s	12.244s	5.636s	1.596s	7.635s	14.024s
Simplicial (**Matrix**) == Cholesky (**Matrix**)	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Supernodal (**Matrix**) setup	0.044s	0.12s	0.151s	0.124s	0.317s	0.628s
Supernodal (**Matrix**)	1.037s	4.797s	5.392s	3.751s	9.864s	19.066s
Supernodal (**Matrix**) == Cholesky (**Matrix**)	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
CHOLMOD (**Matrix**) setup	0.042s	0.122s	0.149s	0.11s	0.294s	0.583s
CHOLMOD (**Matrix**)	0.762s	4.571s	5.597s	1.804s	7.386s	14.275s
CHOLMOD (**Matrix**) == Cholesky (**Matrix**)	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
MMD update (**spam**) setup	0.039s	0.199s	0.187s	0.261s	0.509s	0.697s
MMD update (**spam**)	1.472s	10.872s	9.081s	10.309s	11.405s	21.973s
MMD update (**spam**) == Cholesky (**Matrix**)	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
RCM update (**spam**) setup	0.054s	1.542s	1.323s	0.264s	3.881s	5.331s
RCM update (**spam**)	4.182s	59.8s	16.947s	7.099s	238.936s	316.11s
RCM update (**spam**) == Cholesky (**Matrix**)	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
N	3111	4900	15260	25357	32698	64878
Sparseness %	0.188070	0.472453	0.048073	0.011645	0.019177	0.009563
Cholesky (**Matrix**)	3.307s	21.434s	33.927s	29.108s	24.405s	96.071s

the supernodal representation clearly out-performing simplicial for the 70 by 70 grid. The run time of the simplicial decomposition is also close to that of the **spam** MMD pivoting scheme. Once again the **spam** reverse Cuthill-McKee (RCM) pivoting scheme proves much worse than the default **spam** multiple minimum degree (MMD) pivoting scheme for all data sets.

The supernodal representation is a little faster for the world grid, but for the US counties, the Lucas county houses, the US ZCTAs, and the US Census tracts, the simplicial representation is faster. Note that the supernodal representation does better on regular grids with little variation in the number of nonzero elements per row, while the simplicial representation seems to handle greater variation better (see Table 1). While the **spam** multiple minimum degree (MMD) pivoting scheme does quite creditably, it is always inferior to the **Matrix** methods on run times. The overall best choice is to permit the CHOLMOD heuristic decide between the simplicial and supernodal decomposition methods, since the heuristic procedure is only run once when the initial fill-in pattern is determined.

6 Approximations

Walde et al. (2008) also try out a number of approximations to the Jacobian, two of which will be presented here. The timings here reflect the proof-of-concept nature of the coding, which has not been optimised. Despite this, the results are of some interest, and use sparse matrix operations from the **Matrix** package throughout.

Barry and Pace (1999) propose the use of a Monte Carlo estimator with two tuning parameters, p and m (see also LeSage and Pace, 2009, pp. 96–105). The outcome is minus the mean of p random variates V_i, calculated from an $N \times p$ matrix of drawings from the Normal distribution with zero mean and unit variance, and m products of this matrix and the spatial weights matrix $\mathbf{W}$. The setup function prepares a list of these expansion products, storing powers of $\mathbf{W}$ in m $N \times p$ matrices. Zhang and Leithead (2007) suggest that the p candidate draws could be subject to selection to eliminate inappropriately generated seeds. Here the original description due to Barry and Pace (1999) is followed, using $p = 16$ and $m = 30$ as in Walde et al. (2008).

The implementation here re-uses the same random seeds for each λ value by calculating a list of expansion products, but pre-calculates m matrix operations on $N \times p$ matrices to save time. The method is as follows:

$$V_i = -N \sum_{k=1}^{m} \frac{\mathbf{x}_i' \mathbf{W}^k \mathbf{x}_i}{\mathbf{x}_i' \mathbf{x}_i} \frac{\lambda^k}{k}$$

for $i = 1, \ldots, p$, $\mathbf{W}$ with real eigenvalues in $[-1, 1]$, and $\mathbf{x}_i \sim N(0, I)$.

Table 5: Timings for Monte Carlo and Chebyshev approximations to the Jacobian.

	US Counties	Grid 70x70	World grid	Lucas OH	US ZCTA	US Tracts
Monte Carlo setup	0.143s	0.257s	1.467s	3.044s	5.005s	7.445s
Monte Carlo	0.053s	0.043s	0.043s	0.043s	0.043s	0.044s
Chebyshev setup q=2	0.039s	0.093s	0.54s	0.052s	1.548s	1.598s
Chebyshev q=2	0.029s	0.029s	0.029s	0.029s	0.029s	0.029s
Chebyshev setup q=5	0.172s	1.101s	3.394s	1.937s	12.254s	9.652s
Chebyshev q=5	0.092s	0.086s	0.09s	0.09s	0.092s	0.094s
N	3111	4900	15260	25357	32698	64878
Sparseness %	0.188070	0.472453	0.048073	0.011645	0.019177	0.009563

A second approximation is proposed by Pace and LeSage (2004), who elaborate a Chebyshev decomposition, where:

$$\log|\mathbf{I}-\lambda\mathbf{W}| \approx \sum_{j=1}^{q+1} c_j(\lambda) tr(\mathbf{T}_{j-1}(\mathbf{W})) - \frac{n}{2}c_1(\lambda)$$

where $\mathbf{T}_0(\mathbf{W}) = \mathbf{I}$, $\mathbf{T}_1(\mathbf{W}) = \mathbf{W}$, $\mathbf{T}_2(\mathbf{W}) = 2\mathbf{W}^2 - \mathbf{I}$, $\mathbf{T}_{k+1}(\mathbf{W}) = 2\mathbf{W}\mathbf{T}_k(\mathbf{W}) - \mathbf{T}_{k-1}(\mathbf{W})$, and q represents the highest power of the approximating polynomial. The matrix traces can be set up without knowledge of the λ values entering into the Jacobian, and may be constructed more efficiently as shown by Pace and LeSage (2004, p. 188); the maximum value of q is taken here as 5 (see also LeSage and Pace, 2009, pp. 105–108).

The $q+1$ coefficients $c_j(\lambda)$ are given by:

$$c_j(\lambda) = (\frac{2}{q+1}) \sum_{k=1} q+1 \log[1-\lambda\cos(\frac{\pi(k-0.5)}{q+1})]\cos(\frac{\pi(j-1)(k-0.5)}{q+1})$$

No matrix operations are involved in calculating the approximations to the Jacobian for successive values of λ, yielding very fast look-up times.

Table 5 shows that the setup times for the approximations are not large, but that the Monte Carlo execution is laborious in this implementation. The Chebyshev approximation method performs very fast, and is constant in q, although setup increases in q as one would expect. Walde et al. (2008) advise against using the Chebyshev approximation, but only tried $q = 2$, refering to results in the initial paper (Pace and LeSage, 2004). Since the setup time for larger q is not great, $q = 5$ has also been used here. Figure 1 and Table 6 show that while the Chebyshev approximation with $q = 2$ has obvious problems, both the Monte Carlo approximation and the Chebyshev approximation with $q = 5$ perform quite well in terms of accuracy. Performance falls off as λ moves to the extremes of its feasible range.

Other approximation methods are discussed by Smirnov and Anselin (2001) and Griffith (2004), and work in this area is continuing (Smirnov and Anselin, 2009).

7 Conclusions

We have reviewed implementation details of sparse matrix and approximate approaches to fitting spatial regression models and to interpreting their results. Many of the implementation details are not obvious to users, but do affect their ability to get work done. More studies are required to compare alternative implementation choices; more will certainly be undertaken following the publication of LeSage and Pace (2009). The currently released version of the R package **spdep** provides some opportunities for experimentation.

Table 6: Summaries of differences between Jacobian values computed from Cholesky decompositions and Monte Carlo and Chebyshev $q = 5$ approximations.

	US Counties	Grid 70x70	World grid	Lucas OH	US ZCTA	US Tracts
Monte Carlo Min.	-16.5700	-1.7010	-102.5000	-1742.0000	-124.2000	-336.7000
Chebyshev q=5 Min.	-5.6480	0.0000	0.0000	-1645.0000	-41.1100	-77.0200
Monte Carlo 1st Qu.	-0.2615	-0.9372	-2.7830	-20.2500	-3.8970	-7.0890
Chebyshev q=5 1st Qu.	-0.0719	0.0037	0.0025	-7.2640	-0.0698	-0.1283
Monte Carlo Median	-0.0154	0.1230	-0.0358	-1.6140	-1.5240	-0.5285
Chebyshev q=5 Median	-0.0051	0.2769	0.1818	-0.3515	-0.0025	-0.0003
Monte Carlo Mean	0.0056	0.7608	0.1394	-23.4800	-4.7280	-5.8940
Chebyshev q=5 Mean	-0.2135	7.6270	5.0260	-31.6700	-0.2113	1.8600
Monte Carlo 3rd Qu.	0.5734	1.9430	4.1670	15.6200	-0.3082	4.7840
Chebyshev q=5 3rd Qu.	-0.0000	4.7880	3.0530	-0.0043	-0.0000	0.0429
Monte Carlo Max.	1.1770	6.1630	10.4900	34.5900	0.0792	10.4900
Chebyshev q=5 Max.	1.6670	135.8000	73.7600	0.0000	42.7400	127.8000

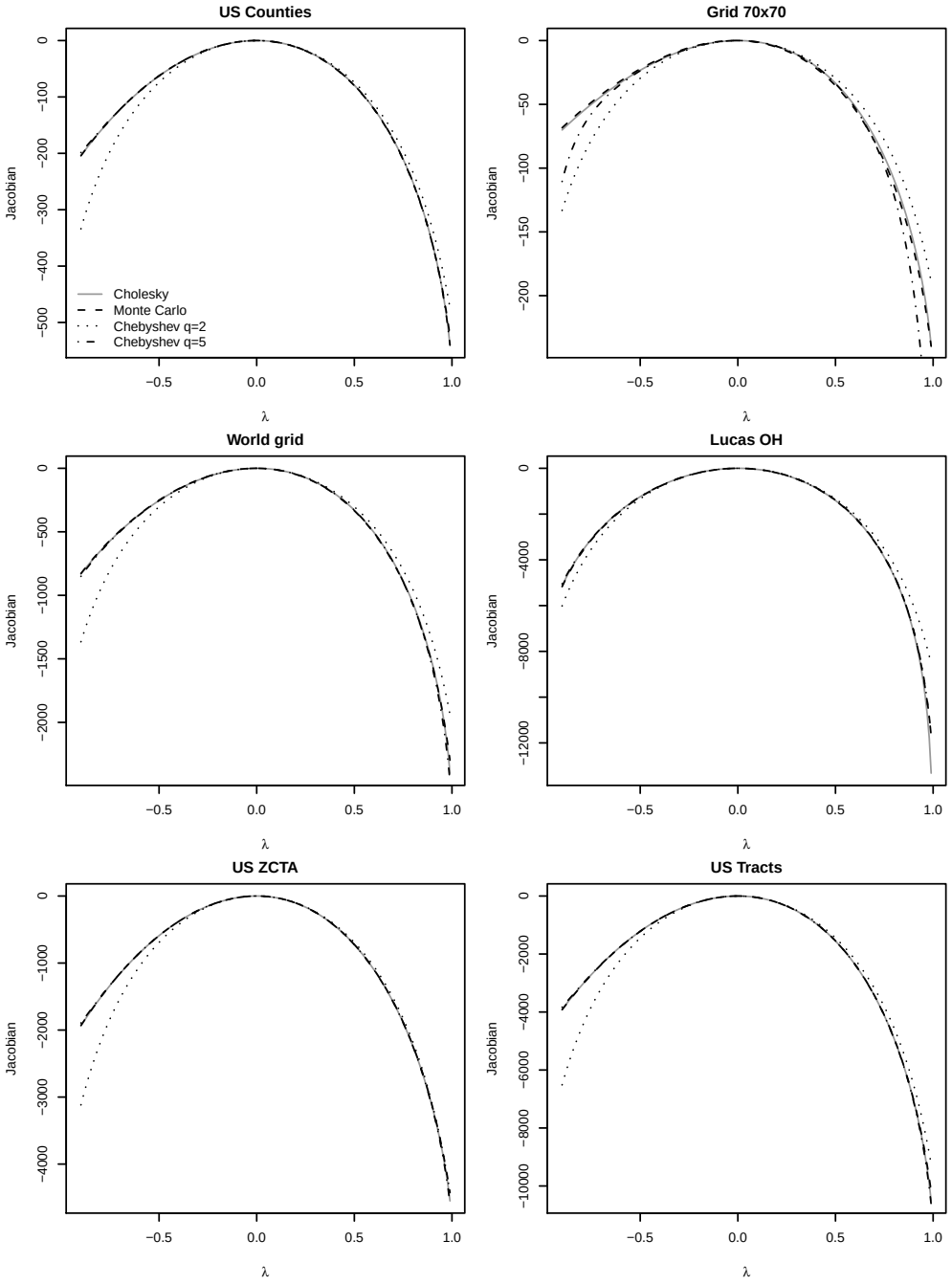

Figure 1: Jacobian approximations compared with Cholesky decomposition Jacobian values.

While many of the points made in the discussion of the computation of the Jacobian are critical of the conclusions drawn by Walde et al. (2008), it is only through work like theirs that these issues can be given the attention that they deserve. Our conclusions are that the use of eigenvalues and sparse matrix decompositions are equivalent, and will only yield different results in maximum likelihood estimation if other aspects of the implementation differ — the Jacobian values for given λ will be the same within machine precision. There is an open question concerning the finding of the interval for conducting the line search for λ when its largest and smallest eigenvalues are not available either because the eigenproblem cannot be solved for large N, or where the interval is not imposed by design. Another open question concerns the symmetry of the underlying spatial weights, which is often imposed by design, but which may not represent the underlying data generation process adequately.

Further, we would argue that updating a Cholesky decomposition of the spatial weights matrix is an alternative that deserves broader use. The rejection by Walde et al. (2008) of the Chebyshev approximation appears to be driven by their choice of $q = 2$ — we find that a value of $q = 5$ gives acceptable results when λ is not strongly negative, and that higher values of q deserve study. The implementation of the Monte Carlo approximation used here is not inefficient, but could be improved to reduce timings even more — its accuracy is acceptable.

Appendix

Code used to generate results:

```
> library(spdep)
> tab_out <- list()
> dsets <- c("USC", "Walde4900", "wrld", "LO", "USZC", "UST")
> for (dset in dsets) {
+     load(paste(dset, "_lw.RData", sep = ""))
+     nb <- get(paste(dset, "_lw", sep = ""))$neighbours
+     tab_out[[dset]] <- vector(mode = "list", length = 4)
+     cnb <- card(nb)
+     tab_out[[dset]][[1]] <- c(table(cnb))
+     tab_out[[dset]][[2]] <- length(nb)
+     tab_out[[dset]][[3]] <- n.comp.nb(nb)$nc
+     tab_out[[dset]][[4]] <- sum(cnb)
+ }
> save(tab_out, file = "tab_out.RData")

> library(spdep)
> load("USC_lw.RData")
> USC_nb <- USC_lw$neighbours
> set.ZeroPolicyOption(TRUE)
```

```
> eigs_out <- vector(mode = "list", length = 6)
> eigs_out[[1]] <- eigenw(nb2listw(USC_nb, style = "B"))
> eigs_out[[2]] <- eigenw(nb2listw(USC_nb, style = "C"))
> eigs_out[[3]] <- eigenw(nb2listw(USC_nb, style = "S"))
> eigs_out[[4]] <- eigenw(nb2listw(USC_nb, style = "W"))
> eigs_out[[5]] <- eigenw(similar.listw(nb2listw(USC_nb, style = "S")))
> eigs_out[[6]] <- eigenw(similar.listw(nb2listw(USC_nb, style = "W")))
> eig_res <- sapply(eigs_out, function(x) 1/range(Re(x)))
> save(eig_res, file = "eigs_out_res.RData")
```

```
> library(spdep)
> library(spam)
> set.ZeroPolicyOption(TRUE)
> lambda <- seq(-0.9, 0.99, 0.01)
> dsets <- c("USC", "Walde4900", "wrld", "LO", "USZC", "UST")
> output <- list()
> for (dset in dsets) {
+     load(paste(dset, "_lw.RData", sep = ""))
+     listw <- get(paste(dset, "_lw", sep = ""))
+     can.sim <- FALSE
+     if (listw$style %in% c("W", "S"))
+         can.sim <- spdep:::can.be.simmed(listw)
+     res <- list()
+     if (length(listw$neighbours) < 5000) {
+         env <- new.env(parent = globalenv())
+         assign("listw", listw, envir = env)
+         assign("verbose", FALSE, envir = env)
+         assign("can.sim", can.sim, envir = env)
+         assign("family", "SAR", envir = env)
+         setTime <- system.time(eigen_setup(env))
+         type <- get("method", envir = env)
+         res[[type]] <- vector(mode = "list", length = 3)
+         res[[type]][[1]] <- setTime
+         res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+             env)))
+         res[[type]][[3]] <- out
+         rm(env)
+     }
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("family", "SAR", envir = env)
+     set.seed(length(listw$neighbours))
+     setTime <- system.time(mcdet_setup(env))
+     type <- get("method", envir = env)
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
```

```
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(cheb_setup(env, q = 2))
+     type <- paste(get("method", envir = env), "2", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(cheb_setup(env, q = 5))
+     type <- paste(get("method", envir = env), "5", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_setup(env, Imult = 2, super = FALSE))
+     type <- paste(get("method", envir = env), "simp", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_setup(env, Imult = 2, super = TRUE))
+     type <- paste(get("method", envir = env), "sup", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
```

```
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_setup(env, Imult = 2, super = as.logical(NA)))
+     type <- paste(get("method", envir = env), "NA", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_J_setup(env, super = FALSE))
+     type <- paste(get("method", envir = env), "simp", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_J_setup(env, super = TRUE))
+     type <- paste(get("method", envir = env), "sup", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(Matrix_J_setup(env, super = as.logical(NA)))
+     type <- paste(get("method", envir = env), "NA", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
```

```
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(spam_setup(env))
+     type <- get("method", envir = env)
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(spam_setup(env, pivot = "RCM"))
+     type <- paste(get("method", envir = env), "RCM", sep = "_")
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(spam_update_setup(env))
+     type <- get("method", envir = env)
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("can.sim", can.sim, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(spam_update_setup(env, pivot = "RCM"))
+     type <- paste(get("method", envir = env), "RCM", sep = "_")
```

```
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     env <- new.env(parent = globalenv())
+     assign("listw", listw, envir = env)
+     assign("n", length(listw$neighbours), envir = env)
+     assign("family", "SAR", envir = env)
+     setTime <- system.time(LU_setup(env))
+     type <- get("method", envir = env)
+     res[[type]] <- vector(mode = "list", length = 3)
+     res[[type]][[1]] <- setTime
+     res[[type]][[2]] <- system.time(out <- sapply(lambda, function(x) do_ldet(x,
+         env)))
+     res[[type]][[3]] <- out
+     rm(env)
+     output[[dset]] <- res
+ }
> save(output, file = "output_Jacobian.RData")
```

Functions help page

`do_ldet` *Spatial regression model Jacobian computations*

Description

These functions are made available in the package namespace for other developers, and are not intended for users. They provide a shared infrastructure for setting up data for Jacobian computation, and then for caclulating the Jacobian, either exactly or approximately, in maximum likelihood fitting of spatial regression models. The techniques used are the exact eigenvalue, Cholesky decompositions (Matrix, spam), and LU ones, with Chebyshev and Monte Carlo approximations.

Usage

```
do_ldet(coef, env, which=1)
cheb_setup(env, q=5, which=1)
mcdet_setup(env, p=16, m=30, which=1)
```

```
eigen_setup(env, which=1)
spam_setup(env, pivot="MMD", which=1)
spam_update_setup(env, in_coef=0.1, pivot="MMD", which=1)
Matrix_setup(env, Imult, super=as.logical(NA), which=1)
Matrix_J_setup(env, super=FALSE, which=1)
LU_setup(env, which=1)
```

Arguments

`coef`	spatial coefficient value
`env`	environment containing pre-computed objects, fixed after assignment in setup functions
`which`	default 1; if 2, use second listw object
`q`	Chebyshev approximation order; default in calling spdep functions is 5, here it cannot be missing and does not have a default
`p`	Monte Carlo approximation number of random normal variables; default calling spdep functions is 16, here it cannot be missing and does not have a default
`m`	Monte Carlo approximation number of series terms; default in calling spdep functions is 30, here it cannot be missing and does not have a default
`pivot`	default "MMD", may also be "RCM" for Cholesky decompisition using spam
`in_coef`	fill-in initiation coefficient value, default 0.1
`Imult`	see `Cholesky`; numeric scalar which defaults to zero. The matrix that is decomposed is A+m*I where m is the value of Imult and I is the identity matrix of order ncol(A). Default in calling spdep functions is 2, here it cannot be missing and does not have a default, but is rescaled for binary weights matrices in proportion to the maximim row sum in those calling functions
`super`	see `Cholesky`; logical scalar indicating is a supernodal decomposition should be created. The alternative is a simplicial decomposition. Default in calling spdep functions is FALSE for "Matrix_J" and `as.logical(NA)` for "Matrix". Setting it to NA leaves the choice to a CHOLMOD-internal heuristic

Details

Since environments are containers in the R workspace passed by reference rather than by value, they are useful for passing objects to functions called in numerical optimisation, here for the maximum likelihood estimation of spatial regression models. This technique can save a little time on each function call, balanced against the need to access the objects in the environment inside the function. The environment should contain a `family` string object either "SAR", "CAR" or "SMA" (used in `do_ldet` to choose spatial moving average in `spautolm`, and these specific objects before calling the set-up functions:

eigen Classical Ord eigenvalue computations:

listw A listw spatial weights object

can.sim logical scalar: can the spatial weights be made symmetric by similarity

verbose logical scalar: legacy report print control, for historical reasons only

and assigns to the environment:

eig a vector of eigenvalues

eig.range the search interval for the spatial coefficient

method string: "eigen"

Matrix Sparse matrix pre-computed Cholesky decomposition with fast updating:

listw A listw spatial weights object

can.sim logical scalar: can the spatial weights be made symmetric by similarity

and assigns to the environment:

csrw sparse spatial weights matrix

nW negative sparse spatial weights matrix

pChol a "CHMfactor" from factorising `csrw` with `Cholesky`

nChol a "CHMfactor" from factorising `nW` with `Cholesky`

method string: "Matrix"

Matrix_J Standard Cholesky decomposition without updating:

listw A listw spatial weights object

can.sim logical scalar: can the spatial weights be made symmetric by similarity

n number of spatial objects

and assigns to the environment:

csrw sparse spatial weights matrix

I sparse identity matrix

super the value of the `super` argument

method string: "Matrix_J"

spam Standard Cholesky decomposition without updating:

listw A listw spatial weights object

can.sim logical scalar: can the spatial weights be made symmetric by similarity

n number of spatial objects

and assigns to the environment:

csrw sparse spatial weights matrix

I sparse identity matrix

pivot string — pivot method

method string: "spam"

spam_update Pre-computed Cholesky decomposition with updating:

listw A listw spatial weights object

can.sim logical scalar: can the spatial weights be made symmetric by similarity

n number of spatial objects

and assigns to the environment:

csrw sparse spatial weights matrix

I sparse identity matrix

csrwchol A Cholesky decomposition for updating

method string: "spam"

LU Standard LU decomposition without updating:

listw A listw spatial weights object

n number of spatial objects

and assigns to the environment:

W sparse spatial weights matrix

I sparse identity matrix

method string: "LU"

MC Monte Carlo approximation:

listw A listw spatial weights object

and assigns to the environment:

clx list of Monte Carlo approximation terms

W sparse spatial weights matrix

method string: "MC"

cheb Chebyshev approximation:

listw A listw spatial weights object

and assigns to the environment:

trT vector of Chebyshev approximation terms

W sparse spatial weights matrix

method string: "Chebyshev"

Some set-up functions may also assign `similar` to the environment if the weights were made symmetric by similarity.

Value

`do_ldet` returns the value of the Jacobian for the calculation method recorded in the environment argument; the remaining functions modify the environment in place as a side effect and return nothing.

Author(s)

Roger Bivand `<Roger.Bivand@nhh.no>`

References

LeSage J and RK Pace (2009) Introduction to Spatial Econometrics. CRC Press, Boca Raton, pp. 77–110

References

Barry, R. and Pace, R. (1999). Monte Carlo estimates of the log determinant of large sparse matrices. *Linear Algebra and its Applications*, 289(1-3):41–54.

Bivand, R. S. (1984). Regression modeling with spatial dependence: an application of some class selection and estimation methods. *Geographical Analysis*, 16:25–37.

Bivand, R. S., Pebesma, E. J., and Gómez-Rubio, V. (2008). *Applied Spatial Data Analysis with R*. Springer, New York.

Cliff, A. D. and Ord, J. K. (1973). *Spatial Autocorrelation*. Pion, London.

Davis, T. A. (2006). *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia.

Griffith, D. (2000). Eigenfunction properties and approximations of selected incidence matrices employed in spatial analyses. *Linear Algebra and its Applications*, 321(1-3):95–112.

Griffith, D. (2004). Faster maximum likelihood estimation of very large spatial autoregressive models: An extension of the Smirnov-Anselin result. *Journal of Statistical Computation and Simulation*, 74(12):855–866.

Kaluzny, S. P., Vega, S. C., Cardoso, T. P., and Shelly, A. A. (1998). *S+SpatialStats, User Manual for Windows and UNIX*. Springer-Verlag, Berlin.

LeSage, J. and Pace, R. (2009). *Introduction to Spatial Econometrics*. CRC Press, Boca Raton, FL.

Ord, J. (1975). Estimation methods for models of spatial interaction. *Journal of the American Statistical Association*, 70(349):120–126.

Pace, R. and Barry, R. (1997a). Fast spatial estimation. *Applied Economics Letters*, 4(5):337–341.

Pace, R. and Barry, R. (1997b). Quick computation of spatial autoregressive estimators. *Geographics Analysis*, 29(3):232–247.

Pace, R. and Barry, R. (1997c). Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297.

Pace, R. and LeSage, J. (2004). Chebyshev approximation of log-determinants of spatial weight matrices. *Computational Statistics & Data Analysis*, 45(2):179–196.

Ripley, B. D. (1981). *Spatial Statistics*. Wiley, New York.

Smirnov, O. and Anselin, L. (2001). Fast maximum likelihood estimation of very large spatial autoregressive models: a characteristic polynomial approach. *Computational Statistics & Data Analysis*, 35(3):301–319.

Smirnov, O. and Anselin, L. (2009). An O(N) parallel method of computing the Log-Jacobian of the variable transformation for models with spatial interaction on a lattice. *Computational Statistics & Data Analysis*, 53(8):2980 – 2988.

Tiefelsdorf, M., Griffith, D. A., and Boots, B. (1999). A variance-stabilizing coding scheme for spatial link matrices. *Environment and Planning A*, 31:165–180.

Walde, J., Larch, M., and Tappeiner, G. (2008). Performance contest between MLE and GMM for huge spatial autoregressive models. *Journal of Statistical Computation and Simulation*, 78(2):151–166.

Ward, M. D. and Gleditsch, K. S. (2008). *Spatial Regression Models*. Sage, Thousand Oaks, CA.

Zhang, Y. and Leithead, W. (2007). Approximate implementation of the logarithm of the matrix determinant in gaussian process regression. *Journal of Statistical Computation and Simulation*, 77(4):329–348.

NHH

Norges Handelshøyskole

Norwegian School of Economics
and Business Administration

NHH
Helleveien 30
NO-5045 Bergen
Norway

Tlf/Tel: +47 55 95 90 00
Faks/Fax: +47 55 95 91 00
nhh.postmottak@nhh.no
www.nhh.no