Lab 1: Sparse matrix methods in R and Matlab

Gaussian Markov random fields

David Bolin
Chalmers University of Technology
February 3, 2015

R or Matlab?

Matlab is probably easier to use if you have not used any of the programs for sparse matrix computations before, it will probably also result in faster computations.

R or Matlab?

Matlab is probably easier to use if you have not used any of the programs for sparse matrix computations before, it will probably also result in faster computations.

The advantage with using R is that parts of the next project has to be done in R. There are also more functions available for doing GMRF related computations in R.

R or Matlab?

Matlab is probably easier to use if you have not used any of the programs for sparse matrix computations before, it will probably also result in faster computations.

The advantage with using R is that parts of the next project has to be done in R. There are also more functions available for doing GMRF related computations in R.

We will now take a look at some sparse matrix basics for the two programs.

Sparse matrices in Matlab

See

`www.mathworks.com/help/matlab/sparse-matrices.html`

Sparse matrices in Matlab

See

`www.mathworks.com/help/matlab/sparse-matrices.html`

- The basic way to create an m-by-n sparse matrix is

  `S = sparse(i,j,s,m,n,nzmax)`

  where i, j, and s are vectors such that `S(i(k),j(k)) = s(k)`, with space allocated for nzmax nonzeros.

Sparse matrices in Matlab

See

`www.mathworks.com/help/matlab/sparse-matrices.html`

- The basic way to create an m-by-n sparse matrix is

  ```
  S = sparse(i,j,s,m,n,nzmax)
  ```

  where i, j, and s are vectors such that `S(i(k),j(k)) = s(k)`, with space allocated for nzmax nonzeros.
- For band matrices, use

  ```
  A = spdiags(B,d,m,n)
  ```

  which creates an m-by-n sparse matrix by taking the columns of B and placing them along the diagonals specified by d.

Sparse matrices in Matlab

See

`www.mathworks.com/help/matlab/sparse-matrices.html`

- The basic way to create an m-by-n sparse matrix is

  `S = sparse(i,j,s,m,n,nzmax)`

  where i, j, and s are vectors such that `S(i(k),j(k)) = s(k)`, with space allocated for nzmax nonzeros.
- For band matrices, use

  `A = spdiags(B,d,m,n)`

  which creates an m-by-n sparse matrix by taking the columns of B and placing them along the diagonals specified by d.
- An n-by-n sparse identity matrix is obtained by

  `S = speye(n)`

Cholesky factoization and solves

- The upper triangular cholesky factor of a SPD matrix:

```
R = chol(Q);
```

Cholesky factoization and solves

- The upper triangular cholesky factor of a SPD matrix:

```
R = chol(Q);
```

- The lower triangular cholesky factor of a SPD matrix:

```
L = chol(Q,'lower');
```

Cholesky factoization and solves

- The upper triangular cholesky factor of a SPD matrix:

  `R = chol(Q);`

- The lower triangular cholesky factor of a SPD matrix:

  `L = chol(Q,'lower');`

- Solve `Ax = B`:

  `X = A\B`

Cholesky factoization and solves

- The upper triangular cholesky factor of a SPD matrix:

  `R = chol(Q);`

- The lower triangular cholesky factor of a SPD matrix:

  `L = chol(Q,'lower');`

- Solve `Ax = B`:

  `X = A\B`

- Solve `xA = B`:

  `X = B/A`

Reorderings

- An approximate minumum degree ordering of a matrix $\mathbf{Q}$ is given by

```
reo = amd(Q);
```

Reorderings

- An approximate minumum degree ordering of a matrix $\mathbf{Q}$ is given by

```
reo = amd(Q);
```

- A profile reduction ordering is given by

```
reo = symrcm(Q);
```

Reorderings

- An approximate minumum degree ordering of a matrix $\mathbf{Q}$ is given by

```
reo = amd(Q);
```

- A profile reduction ordering is given by

```
reo = symrcm(Q);
```

- Inverse reordering

```
n = size(Q,1);
ireo(reo) = 1:n;
```

Reorderings

- An approximate minumum degree ordering of a matrix $\mathbf{Q}$ is given by

```
reo = amd(Q);
```

- A profile reduction ordering is given by

```
reo = symrcm(Q);
```

- Inverse reordering

```
n = size(Q,1);
ireo(reo) = 1:n;
```

- So, for example, sample a mean-zero GMRF using

```
reo = amd(Q);
ireo(reo) = 1:n;
R = chol(Q(reo,reo));
Xreo = R\randn(n,1);
X = Xreo(ireo);
```

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  ```
  dist = squareform(pdist(s));
  ```

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  ```
  dist = squareform(pdist(s));
  ```

- Compute distances between two sets of points:

  ```
  dist = pdist2(s1,s2);
  ```

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  `dist = squareform(pdist(s));`

- Compute distances between two sets of points:

  `dist = pdist2(s1,s2);`

- Kronecker product of two matrices

  `kron(A,B)`

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  ```
  dist = squareform(pdist(s));
  ```

- Compute distances between two sets of points:

  ```
  dist = pdist2(s1,s2);
  ```

- Kronecker product of two matrices

  ```
  kron(A,B)
  ```

- Simulate gamma distributed random numbers

  ```
  x  = gamrnd(alpha, 1/beta);
  ```

  Warning: Note the inverse, Matlab has a slightly non-standard parameterisation.

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  ```
  dist = squareform(pdist(s));
  ```
- Compute distances between two sets of points:

  ```
  dist = pdist2(s1,s2);
  ```
- Kronecker product of two matrices

  ```
  kron(A,B)
  ```
- Simulate gamma distributed random numbers

  ```
  x   = gamrnd(alpha, 1/beta);
  ```

  Warning: Note the inverse, Matlab has a slightly non-standard parameterisation.
- `besselk` is used to compute modified Bessel functions.

Other Matlab commands you might need

- Compute all pairwise distances of a set of points `s = [x y]`

  ```
  dist = squareform(pdist(s));
  ```
- Compute distances between two sets of points:

  ```
  dist = pdist2(s1,s2);
  ```
- Kronecker product of two matrices

  ```
  kron(A,B)
  ```
- Simulate gamma distributed random numbers

  ```
  x  = gamrnd(alpha, 1/beta);
  ```

  Warning: Note the inverse, Matlab has a slightly non-standard parameterisation.
- `besselk` is used to compute modified Bessel functions.
- `M(:)` column stacks the matrix, use this to convert images to column vectors which are used in calculations.

Example of sparse matrix construction

```
% Construct a precision matrix with stencil q for a m-by-n grid
II = []; KK = []; JJ_I = []; JJ_J = [];
[I,J] = ndgrid(1:m,1:n);
I = I(:); J = J(:);
for i=1:size(q,1)
  for j=1:size(q,2)
    if (q(i,j) ~= 0)
      II = [II;I+m*(J-1)];
      JJ_I = [JJ_I;I+i-(size(q,1)+1)/2];
      JJ_J = [JJ_J;J+j-(size(q,2)+1)/2];
      KK = [KK; q(i,j)*ones(m*n,1)];
    end
  end
end
JJ = JJ_I+m*(JJ_J-1);
ok = (JJ_I>=1) & (JJ_I<=m) & (JJ_J>=1) & (JJ_J<=n);
II(~ok) = []; JJ(~ok) = []; KK(~ok) = [];
Q = sparse(II,JJ,KK,m*n,m*n);
```

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
  - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
  - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.
  - `SPAM`: Often faster than `Matrix`, especially useful for MCMC methods, where one can take advantage of symbolic factorisations.

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
  - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.
  - `SPAM`: Often faster than `Matrix`, especially useful for MCMC methods, where one can take advantage of symbolic factorisations.
- Although not a sparse matrix package, `INLA` has several useful methods for sparse matrices and GMRFs.

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
  - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.
  - `SPAM`: Often faster than `Matrix`, especially useful for MCMC methods, where one can take advantage of symbolic factorisations.
- Although not a sparse matrix package, `INLA` has several useful methods for sparse matrices and GMRFs.
  - It is based on `GMRFlib`, which is used in the course book.

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
    - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.
    - `SPAM`: Often faster than `Matrix`, especially useful for MCMC methods, where one can take advantage of symbolic factorisations.
- Although not a sparse matrix package, `INLA` has several useful methods for sparse matrices and GMRFs.
    - It is based on `GMRFlib`, which is used in the course book.
    - This package is not on CRAN, so see `www.r-inla.org/download` for easy installation instructions.

Sparse matrices in R

- There are several sparse matrix packages in R, I typically use:
  - `Matrix`: Standard in R. Based on Suitesparse, so the internals are quite similar to the Matlab system. We will focus on this package.
  - `SPAM`: Often faster than `Matrix`, especially useful for MCMC methods, where one can take advantage of symbolic factorisations.
- Although not a sparse matrix package, `INLA` has several useful methods for sparse matrices and GMRFs.
  - It is based on `GMRFlib`, which is used in the course book.
  - This package is not on CRAN, so see `www.r-inla.org/download` for easy installation instructions.
  - We will look at this in more detail later.

Sparse matrices using the `Matrix` package

- Here we have more control over what internal structure one should use to store the matrix: Row-oriented or column-oriented storage? Use symmetry? Compressed storage? Triplet form? etc etc.

Sparse matrices using the `Matrix` package

- Here we have more control over what internal structure one should use to store the matrix: Row-oriented or column-oriented storage? Use symmetry? Compressed storage? Triplet form? etc etc.
- However, a basic column-oriented sparse matrix can be constructed just as in Matlab

```
Q = sparseMatrix(i=row.index,
                 j=col.index,
                 x=values,
                 dims=c(m, n),
                 symmetric=FALSE)
```

Sparse matrices using the Matrix package

- Here we have more control over what internal structure one should use to store the matrix: Row-oriented or column-oriented storage? Use symmetry? Compressed storage? Triplet form? etc etc.
- However, a basic column-oriented sparse matrix can be constructed just as in Matlab

```
Q = sparseMatrix(i=row.index,
                 j=col.index,
                 x=values,
                 dims=c(m, n),
                 symmetric=FALSE)
```

- Use `Diagonal(n,data)` to construct a diagonal matrix

Sparse matrices using the Matrix package

- Here we have more control over what internal structure one should use to store the matrix: Row-oriented or column-oriented storage? Use symmetry? Compressed storage? Triplet form? etc etc.
- However, a basic column-oriented sparse matrix can be constructed just as in Matlab

```
Q = sparseMatrix(i=row.index,
                 j=col.index,
                 x=values,
                 dims=c(m, n),
                 symmetric=FALSE)
```

- Use `Diagonal(n,data)` to construct a diagonal matrix
- To construct a sparse Toeplitz matrix you can use

```
M <- toeplitz(sparseVector(values,indices,n))
```

Cholesky factors and solves using `Matrix`

- `chol(M)` gives you a sparse supernodal Cholesky factorization if the matrix has a sparse matrix class.

Cholesky factors and solves using `Matrix`

- `chol(M)` gives you a sparse supernodal Cholesky factorization if the matrix has a sparse matrix class.
- The `pivot` argument sets if the method also should do a fill-in reducing reordering.

Cholesky factors and solves using `Matrix`

- `chol(M)` gives you a sparse supernodal Cholesky factorization if the matrix has a sparse matrix class.
- The `pivot` argument sets if the method also should do a fill-in reducing reordering.
- If we set `pivot=TRUE`, an AMD ordering is used. For more flexibility, we can set `pivot=FALSE` and reorder the matrix manually before calling the function.

Cholesky factors and solves using `Matrix`

- `chol(M)` gives you a sparse supernodal Cholesky factorization if the matrix has a sparse matrix class.
- The `pivot` argument sets if the method also should do a fill-in reducing reordering.
- If we set `pivot=TRUE`, an AMD ordering is used. For more flexibility, we can set `pivot=FALSE` and reorder the matrix manually before calling the function.
- `X <- solve(A,B)` solves the system `AX = B`

INLA methods

- `INLA` implements several orderings: the function `inla.qreordering` can be used to reorder a matrix.

INLA methods

- `INLA` implements several orderings: the function `inla.qreordering` can be used to reorder a matrix.
- The default is to test several orderings and return the best, you can also fix a specific ordering, type `inla.reorderings()` to see which are available.

INLA methods

- INLA implements several orderings: the function inla.qreordering can be used to reorder a matrix.
- The default is to test several orderings and return the best, you can also fix a specific ordering, type inla.reorderings() to see which are available.
- Other useful INLA functions, based on GMRFlib are

INLA methods

- INLA implements several orderings: the function inla.qreordering can be used to reorder a matrix.
- The default is to test several orderings and return the best, you can also fix a specific ordering, type inla.reorderings() to see which are available.
- Other useful INLA functions, based on GMRFlib are
  - inla.qsolve: Solves systems AX = B

INLA methods

- INLA implements several orderings: the function `inla.qreordering` can be used to reorder a matrix.
- The default is to test several orderings and return the best, you can also fix a specific ordering, type `inla.reorderings()` to see which are available.
- Other useful INLA functions, based on GMRFlib are
    - `inla.qsolve`: Solves systems `AX = B`
    - `inla.qsample`, for sampling GMRFs

INLA methods

- `INLA` implements several orderings: the function `inla.qreordering` can be used to reorder a matrix.
- The default is to test several orderings and return the best, you can also fix a specific ordering, type `inla.reorderings()` to see which are available.
- Other useful `INLA` functions, based on `GMRFlib` are
  - `inla.qsolve`: Solves systems `AX = B`
  - `inla.qsample`, for sampling GMRFs
  - `inla.qinv`, for computing marginal variances based on precision matrices (see Lecture 4)

Some comments about the project

- The Rosetta images are stored in RGB format, extract, for example, the first “color” to get univariate data.

Some comments about the project

- The Rosetta images are stored in RGB format, extract, for example, the first "color" to get univariate data.
- Convert to double precision, several methods will complain if they get `uint8` numbers.

Some comments about the project

- The Rosetta images are stored in RGB format, extract, for example, the first "color" to get univariate data.
- Convert to double precision, several methods will complain if they get `uint8` numbers.
- Also standardise the data to $(0, 1)$ (assumed in Part 2)

Some comments about the project

- The Rosetta images are stored in RGB format, extract, for example, the first “color” to get univariate data.
- Convert to double precision, several methods will complain if they get `uint8` numbers.
- Also standardise the data to $(0, 1)$ (assumed in Part 2)
- In Matlab:

```
x = imread('rosetta_small.jpg');
x = double(squeeze(x(:,:,1)));
x = x/max(x(:));
```

Regarding the first problem

- Assume, for example, unit distance between the pixels.

Regarding the first problem

- Assume, for example, unit distance between the pixels.
- There is a typo in the slides for Lecture 1. The expression for the Matérn covariance should read

$$C(\mathbf{h}) = \frac{2^{1-\nu}\sigma^2}{(4\pi)^{\frac{d}{2}}\Gamma(\nu + \frac{d}{2})\kappa^{2\nu}}(\kappa\|\mathbf{h}\|)^{\nu}K_{\nu}(\kappa\|\mathbf{h}\|), \quad \mathbf{h} \in \mathbb{R}^d, \nu > 0,$$

Regarding the first problem

- Assume, for example, unit distance between the pixels.
- There is a typo in the slides for Lecture 1. The expression for the Matérn covariance should read

$$C(\mathbf{h}) = \frac{2^{1-\nu}\sigma^2}{(4\pi)^{\frac{d}{2}}\Gamma(\nu + \frac{d}{2})\kappa^{2\nu}}(\kappa\|\mathbf{h}\|)^{\nu}K_{\nu}(\kappa\|\mathbf{h}\|), \quad \mathbf{h} \in \mathbb{R}^d, \nu > 0,$$

- The weird scaling will make sense later.

Regarding the first problem

- Assume, for example, unit distance between the pixels.
- There is a typo in the slides for Lecture 1. The expression for the Matérn covariance should read

$$C(\mathbf{h}) = \frac{2^{1-\nu}\sigma^2}{(4\pi)^{\frac{d}{2}}\Gamma(\nu + \frac{d}{2})\kappa^{2\nu}}(\kappa\|\mathbf{h}\|)^{\nu}K_{\nu}(\kappa\|\mathbf{h}\|), \quad \mathbf{h} \in \mathbb{R}^d, \nu > 0,$$

- The weird scaling will make sense later.
- For $d = 2$, the variance of the field is given by

$$\frac{\sigma^2\Gamma(\nu)}{4\pi\Gamma(\nu+1)\kappa^{2\nu}}.$$

Regarding the first problem

- Assume, for example, unit distance between the pixels.
- There is a typo in the slides for Lecture 1. The expression for the Matérn covariance should read

$$C(\mathbf{h}) = \frac{2^{1-\nu}\sigma^2}{(4\pi)^{\frac{d}{2}}\Gamma(\nu+\frac{d}{2})\kappa^{2\nu}}(\kappa\|\mathbf{h}\|)^{\nu}K_{\nu}(\kappa\|\mathbf{h}\|), \quad \mathbf{h}\in\mathbb{R}^d, \nu>0,$$

- The weird scaling will make sense later.
- For $d=2$, the variance of the field is given by

$$\frac{\sigma^2\Gamma(\nu)}{4\pi\Gamma(\nu+1)\kappa^{2\nu}}.$$

- The practical range (i.e. the distance where the correlation is approximately 0.1) of the field is given by $\rho = \sqrt{8\nu}\kappa^{-1}$.