# *sparseHessianFD*: An R Package for Estimating Sparse Hessian Matrices

Michael Braun
Cox School of Business
Southern Methodist University

April 20, 2015

The Hessian matrix of a log likelihood function or log posterior density function plays an important role in statistics. From a frequentist point of view, the inverse of the negative Hessian is the asymptotic covariance of the sampling distribution of a maximum likelihood estimator. In Bayesian analysis, when evaluated at the posterior mode, it is the covariance of a Gaussian approximation to the posterior distribution. More broadly, many numerical optimization algorithms require repeated computation, estimation or approximation of the Hessian or its inverse; see Nocedal et al. (2006).

The Hessian of an objective function with $M$ variables has $M^2$ elements, of which $M(M+1)/2$ are unique. Thus, the storage requirements of the Hessian, and computational cost of many linear algebra operations on it, grow quadratically with the number of decision variables. For functions with hundreds of thousands of variables, computing the Hessian even once might not be practical for applications constrained by time, storage or processor availability.

For many problems, the Hessian is *sparse*, meaning that the proportion of non-zero elements in the Hessian is small. Consider a log posterior density in a Bayesian hierarchical model. If the outcomes across heterogeneous units are conditionally independent, the cross-partial derivatives with respect to those parameters are zero. As the number of units increases, the size of the Hessian still grows quadratically, but the number of *non-zero* elements grows only linearly, and the Hessian becomes increasingly sparse. The row and column indices of the non-zero elements comprise the *sparsity pattern* of the Hessian, and are typically known in advance, before computing the values of those elements.

The *sparseHessianFD* package is a tool for estimating sparse Hessians using *finite differencing*. Section 4 will cover the specifics, but the basic idea is as follows. Consider a function $f(x)$ and its gradient $Df(x)^\top$ (the transpose of the derivative). Let $e_m$ be the $m$th coordinate vector, and let $\delta$ be a sufficiently small scalar constant. The vector $\mathsf{H}_m f(x) = (\mathsf{D}f(x+\delta e_m) - \mathsf{D}f(x))/\delta$ is a linear approximation of the $m$th column of the Hessian matrix $\mathsf{H}f(x)$. Estimating a dense Hessian involves $M+1$ calculations of the derivative: one for the derivative at $x$, and one after perturbing each of the $M$ elements of $x$ one at a time. However, if the Hessian has a sparsity pattern that allows it, we could perturb more than one element of $x$ at a time, evaluate the gradient fewer than $M+1$ times, and still recover the non-zero Hessian values. For some sparsity patterns, estimating a Hessian in this way can be profoundly efficient. In fact, for the hierarchical models that we consider in this paper, the number of gradient evaluations is *constant*, even as additional heterogeneous units are added to the model. How to decide which variables can be permuted together is actually a graph coloring problem, which we discuss in Section 4.2.

At the outset, we want to mention that there may be some applications for which *sparseHessianFD* is not an appropriate package to use. To extract the maximum benefit from using *sparseHessianFD*, we need to accept a few conditions or assumptions.

1. Preferred alternatives to computing the Hessian are not available. Finite differencing is not generally a "first choice" method. Deriving a gradient or Hessian symbolically, and writing a subroutine to compute it, will give an exact answer, but might be tedious or difficult to implement. Algorithmic differentiation (AD) is probably the most efficient method, but requires specialized libraries that, at this moment, are not yet available in R. *sparseHessianFD* makes the most sense when the gradient is easy to get, but the Hessian is not.

2. The application can tolerate the approximation error in the Hessian that comes with finite differencing methods.

3. The objective function $f(x)$ is twice differentiable, and can be computed "quickly and easily," even for a large number of variables. We leave the definition of "quickly and easily" intentionally murky, since no method of differentiation can overcome pathologies in a function that itself is hard to compute.

4. The gradient can be computed quickly, easily and *exactly* (within machine precision). We do not recommend using finite differenced gradients when computing finite differenced Hessians, for two reasons. First, the approximation errors will be compounded. Second, the time complexity of computing a gradient grows with the number of variables when using finite differencing, but can be just a constant multiple of the time to compute the objective function using other methods like AD (Griewank et al., 2008, p. xii).

5. The sparsity pattern is known in advance, and does not depend on the values of the variables.

Some users may find the requirement to provide the gradient burdensome. We take the position that deriving a vector of first derivatives, and writing R functions to compute them, is a lot easier than doing the same for a matrix of second derivatives. Also, we have found in practice that even when we have derived and coded the Hessian matrix symbolically, sometimes it can still be faster to estimate the Hessian using *sparseHessianFD* than running coding it directly (maybe because of our weak coding skills, or the computation time to compute intermediate values). These are the situations in which *sparseHessianFD* adds the most value. If AD software is available to compute the gradient, then it is probably available for sparse Hessians as well, and *sparseHessianFD* would not be needed.

The rest of this article proceeds as follows. In Section 1, we discuss matrix sparsity in the context of a hierarchical models. In Section 2, we demonstrate how to use the package. Section 3 includes some time and accuracy tests. We save the discussion of the underlying theory, and the specific algorithms that we use, for Section 4. If they wish, end users can skip that last section.

## 1 Sparsity patterns

Before going into the details of how to use the package, let's consider the following example of an objective function with a sparse Hessian. Suppose we have a dataset of $N$ households, each with $T$ opportunities to purchase a particular product. Let $y_i$ be the number of times household $i$ purchases the product, out of the $T$ purchase opportunities, and let $p_i$ be the probability of purchase. The heterogeneous parameter $p_i$ is the same for all $T$ opportunities, so $y_i$ is a binomial random variable. Define each $p_i$ such that it depends on both $k$ continuous covariates $x_i$, and a heterogeneous coefficient vector $\beta_i$.

$$p_i = \frac{\exp(x_i'\beta_i)}{1+\exp(x_i'\beta_i)},\ i = 1...N \tag{1}$$

The coefficients are distributed across the population of households following a multivariate normal distribution with mean $\mu$ and covariance $\Sigma$. Assume that we know $\Sigma$, but not $\mu$. Instead, place a multivariate normal prior on $\mu$, with mean 0 and covariance $\Omega_0$. Thus, each $\beta_i$, and $\mu$ are $k-$dimensional vectors, and the total number of unknown variables in the model is $(N+1)k$.

The log posterior density, ignoring any normalization constants, is

$$\log \pi(\beta_{1:N}, \mu | Y, X, \Sigma_0, \Omega_0) = \sum_{i=1}^{N} p_i^{y_i}(1-p_i)^{T-y_i} - \frac{1}{2}(\beta_i - \mu)' \Sigma^{-1} (\beta_i - \mu) - \frac{1}{2}\mu' \Omega_0^{-1} \mu \tag{2}$$

We will return to this example throughout the article.

The log posterior density in Equation 2 has a sparse Hessian. Since the $\beta_i$ are drawn iid from a multivariate normal, and the $y_i$ are conditionally independent, $\mathsf{H}_{ij} = \frac{\partial^2 \log \pi}{\partial \beta_i \beta_j} = 0$ for all $i \neq j$. However, all of the $\beta_i$ are correlated with $\mu$.

The sparsity pattern depends on how the variables are ordered within the vector. One such ordering is to group all of the coefficients for each unit together.

$$\beta_{11}, ..., \beta_{1k}, \beta_{21}, ..., \beta_{2k}, ... , ... , \beta_{N1} , ... , \beta_{Nk}, \mu_1, ..., \mu_k \tag{3}$$

In this case, the Hessian has a "block-arrow" structure. For example, if $N = 6$ and $k = 2$, then there are ‘14‘ total variables, and the Hessian will have the following pattern.

```
M <- as(kronecker(diag(N),matrix(1,k,k)),"lMatrix")
M <- rBind(M, Matrix(TRUE,k,N*k))
M <- cBind(M, Matrix(TRUE, k*(N+1), k))
print(M)
```

```
## 14 x 14 sparse Matrix of class "lgCMatrix"
##
##  [1,] | | . . . . . . . . . . | |
##  [2,] | | . . . . . . . . . . | |
##  [3,] . . | | . . . . . . . . | |
##  [4,] . . | | . . . . . . . . | |
##  [5,] . . . . | | . . . . . . | |
##  [6,] . . . . | | . . . . . . | |
##  [7,] . . . . . . | | . . . . | |
##  [8,] . . . . . . | | . . . . | |
##  [9,] . . . . . . . . | | . . | |
## [10,] . . . . . . . . | | . . | |
## [11,] . . . . . . . . . . | | | |
## [12,] . . . . . . . . . . | | | |
## [13,] | | | | | | | | | | | | | |
## [14,] | | | | | | | | | | | | | |
```

Another possibility is to group coefficients for each covariate together.

$$\beta_{11}, ..., \beta_{1N}, \beta_{21}, ..., \beta_{2N}, ..., ..., \beta_{k1}, ..., \beta_{kN}, \mu_1, ..., \mu_k \tag{4}$$

Now the Hessian has an "off-diagonal" sparsity pattern.

```
M <- as(kronecker(matrix(1,k,k), diag(N)),"lMatrix")
M <- rBind(M, Matrix(TRUE,k,N*k))
M <- cBind(M, Matrix(TRUE, k*(N+1), k))
print(M)
```

```
## 14 x 14 sparse Matrix of class "lgCMatrix"
##
##  [1,] | . . . . . | . . . . . | |
##  [2,] . | . . . . . | . . . . | |
##  [3,] . . | . . . . . | . . . | |
##  [4,] . . . | . . . . . | . . | |
##  [5,] . . . . | . . . . . | . | |
##  [6,] . . . . . | . . . . . | | |
##  [7,] | . . . . . | . . . . . | |
##  [8,] . | . . . . . | . . . . | |
##  [9,] . . | . . . . . | . . . | |
## [10,] . . . | . . . . . | . . | |
## [11,] . . . . | . . . . . | . | |
## [12,] . . . . . | . . . . . | | |
## [13,] | | | | | | | | | | | | | |
## [14,] | | | | | | | | | | | | | |
```

In both cases, the number of non-zeros is the same. There are 196 elements in this symmetric matrix, but only 76 are non-zero, and only 45 values are unique. Although the reduction in RAM from using a sparse matrix structure for the Hessian may be modest, consider what would happen if $N = 1000$ instead. In that case, there are 2002 variables in the problem, and more than 4 million elements in the Hessian. However, only $1.2004 \times 10^4$ of those elements are non-zero. If we work with only the lower triangle of the Hessian we only need to work with only 7003 values.

## 2 Using the package

### 2.1 The sparseHessianFD class

The package functionality is implemented as a reference class `sparseHessianFD`. The initializer takes the following arguments.

x.init A numeric vector of variables at which the object will be initialized and tested. It is not stored in the object, so it can really be any value, as long as the objective function, gradient and Hessian are all finite.

fn,gr R functions that return the value of the objective function, and its gradient. The first argument is the numeric variable vector. Other named arguments can be passed to `fn` and gr as well (as elements of the ... argument).

rows, cols Integer vectors of the row and column indices of the non-zero elements in the *lower triangle* of the Hessian.

direct This argument is deprecated, and is included only for backwards compatibility with earlier versions.

eps The perturbation amount for finite differencing of the gradient to compute the Hessian. Defaults to `sqrt(.Machine$double.eps)`.

index1 If `TRUE` (the default), `row` and `col` use one-based indexing. If `FALSE`, zero-based indexing is used (which is the internal storage format for matrix classes in the *Matrix* package).

... Additional arguments to be passed to `fn` and `gr`.

To create a `sparseHessianFD` object, just call `sparseHessianFD`. If you are accepting all of the default arguments, the call will look like:

```
obj <- sparseHessianFD(x.init, fn, gr, rows, cols, ...)
```

where `...` represents all other named arguments that are passed to `fn` and `gr`.

The class defines a number of different fields, none of which should be accessed directly. The initializer automatically calls the graph coloring subroutine, evaluates the Hessian at $x$, and performs some other tests, so it may take some time to create the object.

## 2.2 Providing the sparsity pattern

The sparsity pattern of the Hessian is defined as the row and column indices of the non-zero elements in the *lower triangle* the Hessian. Internally, this pattern is stored in a compressed format, but the `sparseHessianFD` initializer requires rows and columns, to keep things simple. It is the responsibility of the user to ensure that the sparsity pattern is correct. Any elements in the upper triangle will be automatically removed, but there is no check that a corresponding element in the lower triangle exists.

The `Matrix.to.Coord` function extracts row and column indices from a sparse matrix. The input matrix to `Matrix.to.Coord` does not have to include the values (if the full Hessian were known, that would possibly defeat the purpose of this package). It is sufficient to supply a logical or pattern matrix, such as `lgCMatrix` or `ngCMatrix`. Rather than trying to keep track of the row and column indices directly, it might be easier to construct a pattern matrix first, check visually that the matrix has the right pattern, and then extract the indices.

The following code constructs a block diagonal matrix, and extracts the sparsity pattern from its lower triangle.

```
M <- as(kronecker(Diagonal(3), Matrix(T,2,2)),"nMatrix")
M

## 6 x 6 sparse Matrix of class "ngTMatrix"
##
## [1,] | | . . . .
## [2,] | | . . . .
## [3,] . . | | . .
## [4,] . . | | . .
## [5,] . . . . | |
## [6,] . . . . | |

tril(M)

## 6 x 6 sparse Matrix of class "ntTMatrix"
##
## [1,] | . . . . .
## [2,] | | . . . .
## [3,] . . | . . .
## [4,] . . | | . .
## [5,] . . . . | .
## [6,] . . . . | |
mc <- Matrix.to.Coord(tril(M))
mc

## $rows
## [1] 1 2 2 3 4 4 5 6 6
##
## $cols
## [1] 1 1 2 3 3 4 5 5 6
```

The list elements `mc$row` and `mc$col` can be passed to `sparseHessianFD` as the `row` and `col` arguments, respectively.

To visually check that a proposed sparsity pattern represents the intended matrix, use the `Coord.to.Pattern.Matrix` function, which is just a wrapper to *Matrix*'s `sparseMatrix` constructor.

```
M2 <- Coord.to.Pattern.Matrix(mc$rows, mc$cols, dims=dim(M))
M2

## 6 x 6 sparse Matrix of class "ngCMatrix"
##
## [1,] | . . . . .
## [2,] | | . . . .
## [3,] . . | . . .
## [4,] . . | | . .
## [5,] . . . . | .
## [6,] . . . . | |
```

### 2.3 Evaluating the Hessian

The `fn`, `gr` and `hessian` methods respectively evaluate the function, gradient and Hessian at a variable vector $x$.

```
f <- obj$fn(x)
df <- obj$gr(x)
hess <- obj$hessian(x)
```

The `fn` and `gr` methods call the same functions that were provided to the class initializer. Since the additional arguments were already supplied as `...`, they do not need to be supplied again. This feature makes subsequent calling of `fn` and `gr` simpler, because only the variable is included in the call.

Similarly, the `hessian` method takes the single argument $x$. The return value is always a `dgCMatrix` object (defined in the *Matrix* package). `dgCMatrix` objects are sparse matrices, stored in a compressed, column-oriented format, and includes all non-zero elements in both the upper and lower triangles.

The `fngr` method returns the function and gradient as a list. The `fngrhs` includes the Hessian as well.

### 2.4 The example

Now we can use *sparseHessianFD* to estimate the Hessian for the log posterior density of the model from Section 1. The package includes functions that compute the value (`binary.f`), the gradient (`binary.grad`) and the Hessian `binary.hess`. The result from `binary.hess` is a "true" value against which we will compare the estimates from `sparseHessianFD`.

The package also includes sample datasets of different sizes.

To start, we load the data, set some dimension parameters, set prior values for $\Sigma^{-1}$ and $\Omega^{-1}$, and simulate a vector of variables at which to evaluate the function. The `binary.f` and `binary.grad` functions take the data and priors as lists. The `data()` call adds the appropriate data list to the environment, but we need to construct the prior list ourselves.

```
set.seed(123)
data(binary_small)
binary <- binary_small
str(binary)
```

```
## List of 3
##  $ Y: int [1:4] 135 127 114 90
##  $ X: num [1:2, 1:4] -0.06369 0.33905 0.00157 0.23102 -0.20344 ...
##  $ T: num 200
```

```
N <- length(binary[["Y"]])
k <- NROW(binary[["X"]])
nvars <- as.integer(N*k + k)
P <- rnorm(nvars) ## random starting values
priors <- list(inv.Sigma = rWishart(1,k+5,diag(k))[,,1],
               inv.Omega = diag(k))
```

This dataset represents the simulated choices for $N = 4$ customers over $T = TRUE$ purchase opportunties, where the probability of purchase is influenced by $k = 2$ covariates.

The following code chunk evaluates the "true" value, gradient and Hessian. The `order.row` argument tells the function whether the variables are ordered by household (`TRUE`) or by covariate (`FALSE`). If 'order.row=TRUE', then the Hessian will have an off-diagonal pattern. If 'order.row=FALSE', then the Hessian will have a block-arrow pattern.

```
true.f <- binary.f(P, binary, priors, order.row=FALSE)
true.grad <- binary.grad(P, binary, priors, order.row=FALSE)
true.hess <- binary.hess(P, binary, priors, order.row=FALSE)
```

The sparsity pattern of the Hessian is specified by two integer vectors: one each for the row and column indices of the non-zero elements of the lower triangule of the Hessian. If you happen have have an example of a matrix with the same sparsity pattern of the Hessian you are trying to compute, you can use the `Matrix.to.Coord` function to extract the appropriate index vectors.

```
pattern <- Matrix.to.Coord(tril(true.hess))
str(pattern)
```

```
## List of 2
##  $ rows: int [1:31] 1 2 9 10 2 9 10 3 4 9 ...
##  $ cols: int [1:31] 1 1 1 1 2 2 2 3 3 3 ...
```

If not, you need to determine the row and column indices manually. If the model is hierarchical, you could use the method of constructing the matrices in 1.

Now we can create a new instance of a `sparseHessianFD` object.

```
obj <- sparseHessianFD(P, fn=binary.f, gr=binary.grad,
       rows=pattern[["rows"]], cols=pattern[["cols"]],
       data=binary, priors=priors, order.row=FALSE)
```

Now we can evaluate the function value, gradient and Hessian through 'obj'.

```
f <- obj$fn(P) #$
gr <- obj$gr(P) #$
hs <- obj$hessian(P) #$
```

Do we get the same results that we would get after calling `binary.f`, 'binary.grad' and 'binary.hess' directly? Let's see.

```
all.equal(f, true.f)

## [1] TRUE

all.equal(gr, true.grad)

## [1] TRUE

all.equal(hs, true.hess)

## [1] "class(target) is dgCMatrix, current is dgeMatrix"
```

If there is any difference, keep in mind that `hs` is a numerical estimate that is not always exact. I certainly wouldn't worry about mean relative differences smaller than, say, $10^{-6}$.

## 3 Speed comparison

The `hessian` function in the *numDeriv* package also estimates Hessians with finite differences, but treats all Hessians a dense. The advantage of using *numDeriv* over *sparseHessianFD* is that *numDeriv* does not require the gradient. However, it does takes some time to run. As with everything in life, there are trade-offs.

```
hess.time <- system.time(H1 <- obj$hessian(P)) #$
print(hess.time)

##    user  system elapsed
##   0.002   0.000   0.002

fd.time <- system.time(H2 <- hessian(obj$fn, P)) #$
print(fd.time)

##    user  system elapsed
##   0.056   0.000   0.056
```

## 4 Underlying theory and algorithms

### 4.1 Numerical differentiation

In this section, we review the theoretical basis for approximating derivatives using finite differences. To facilitate this discussion, we use the notation of Magnus et al. (2007), and their notion of the differential, but we will not be as formal in our proofs.

#### 4.1.1 Definitions of derivatives

Let $f(x)$ be a scalar-valued function, and let $x$ and $u$ be $M$-dimensional vectors. Let $u$ be a sufficiently small positive real value. If $k = 1$, then the definition of the first *derivative* of $f(x)$ is

$$f'(x) = \lim_{u \to 0} \frac{f(x+u) - f(x)}{u} \tag{5}$$

which is equivalent to

$$f(x+u) = f(x) + uf'(x) + o(u) \tag{6}$$

The *partial* derivative of $f(x)$ with respect to $x_j$ (the $j$th component of $x$) is defined as

$$\mathsf{D}_j f(x) = \lim_{\delta \to 0} \frac{f(x+\delta e_j) - f(x)}{\delta} \tag{7}$$

with $\mathsf{D}f(x) = \big(\mathsf{D}_1 f(x), \ldots, \mathsf{D}_M f(x)\big)$ as the vector of all partial derivatives. Thus, we can compute a linear approximation to $\mathsf{D}_j f(x)$ by computing

$$\mathsf{D}_j f(x) \approx \frac{f(x+\delta e_j) - f(x)}{\delta} \tag{8}$$

for a sufficiently small $t$.

The *gradient* is defined as $\nabla f(x) = (\mathsf{D}f(x))^\top$.

The second-order partial derivative is defined as

$$\mathsf{D}^2_{jk} f(x) = \lim_{\delta \to 0} \frac{\mathsf{D}_j f(x+\delta e_k) - \mathsf{D}_j f(x)}{\delta} \tag{9}$$

and the Hessian matrix is defined as

$$\mathsf{H}f(x) = \begin{pmatrix} \mathsf{D}^2_{11} f(x) & \mathsf{D}^2_{12} f(x) & \ldots & \mathsf{D}^2_{1M} f(x) \\ \mathsf{D}^2_{21} f(x) & \mathsf{D}^2_{22} f(x) & \ldots & \mathsf{D}^2_{2M} f(x) \\ \vdots & \vdots & & \vdots \\ \mathsf{D}^2_{K1} f(x) & \mathsf{D}^2_{K2} f(x) & \ldots & \mathsf{D}^2_{MM} f(x) \end{pmatrix} \tag{10}$$

To estimate the $k$th column of $\mathsf{H}f(x)$, we again choose a sufficiently small $t$, and compute

$$\mathsf{H}_k f(x) \approx \frac{\mathsf{D}f(x+\delta e_k) - \mathsf{D}f(x)}{t} \tag{11}$$

$$H = \begin{pmatrix} f_1'(x_1+\delta, x_2+h) - f_1'(x_1,x_2) & 0 \\ 0 & f_2'(x_1+\delta, x_2+\delta) - f_2'(x_1,x_2) \end{pmatrix} / \delta \tag{12}$$

For $M = 2$, our estimate of a general $\mathsf{H}f(x)$ would be

$$\mathsf{H}f(x)\delta = \begin{pmatrix} \mathsf{D}_1 f(x_1+\delta, x_2) - \mathsf{D}_1 f(x_1, x_2) & \mathsf{D}_1 f(x_1, x_2+\delta) - \mathsf{D}_1 f(x_1, x_2) \\ \mathsf{D}_2 f(x_1+\delta, x_2) - \mathsf{D}_2 f(x_1, x_2) & \mathsf{D}_2 f(x_1, x_2+\delta) - \mathsf{D}_2 f(x_1, x_2) \end{pmatrix} \tag{13}$$

This estimate requires three evaluations of the first derivative vector: $\mathsf{D}f(x_1, x_2)$, $\mathsf{D}f(x_1+\delta, x_2)$, and $\mathsf{D}f(x_1, x_2+\delta)$. Now suppose that the Hessian is sparse, and that the off-diagonal elements of it are zero. Not only are

$$\mathsf{D}_1 f(x_1, x_2+\delta) - \mathsf{D}_1 f(x_1, x_2) = 0 \tag{14}$$

$$\mathsf{D}_2 f(x_1+\delta, x_2) - \mathsf{D}_2 f(x_1, x_2) = 0 \tag{15}$$

, but also,

$$\mathsf{D}_1 f(x_1+\delta, x_2+\delta) - \mathsf{D}_1 f(x_1+\delta, x_2) = 0 \tag{16}$$

$$\mathsf{D}_2 f(x_1+\delta, x_2+\delta) - \mathsf{D}_2 f(x_1, x_2+\delta) = 0 \tag{17}$$

Therefore,

$$\mathsf{H}f(x)\delta = \begin{pmatrix} \mathsf{D}_1 f(x_1+\delta, x_2+\delta) - \mathsf{D}_1 f(x_1, x_2) & 0 \\ 0 & \mathsf{D}_2 f(x_1+\delta, x_2+\delta) - \mathsf{D}_2 f(x_1, x_2) \end{pmatrix} \tag{18}$$

This estimate requires only two evaluations of the derivative: $\mathsf{D}f(x_1, x_2)$ and $\mathsf{D}f(x_1+\delta, x_2+\delta)$. Reducing the number of gradient evaluations from 3 to 2 depends on knowing that the cross-partial derivatives are zero.

Now let's consider a general case, starting with a "direct" method first proposed in Curtis et al. (1974) for Jacobian matrices, and described in Powell et al. (1979). To begin, partition the variables into $C$ mutually exclusive groups, or "colors." so $c_k$ indexes the color of variable $m$. Let $G$ and $Y$ be $M \times C$ matrices, where $G_{kc} = \delta$ if variable $m$ belongs to group $c$, and zero otherwise, and let $G_c$ be the $c$th column of $G$. Each column in $Y$ is defined as

$$Y_c = \mathsf{D}f(x + G_c) - \mathsf{D}f(x) \tag{19}$$

If $C = K$ and $c_m = m$, then $G$ is a diagonal matrix with $\delta$ in each diagonal element. The matrix equation $\mathsf{H}f(x)G = Y$ represents the Taylor series approximation $\mathsf{H}_{im}f(x)\delta \approx y_{im}$, and we can solve for all elements of $\mathsf{H}f(x)$ just by computing $Y$. But if $C < M$, there must be at least one column $G_c$ with $\delta$ in at least two rows. Column $Y_c$ would have been computed by perturbing two variables at once, and we would not have been able to solve for any $\mathsf{H}_{im}f(x)$ without further constraints.

The necessary restrictions come from the sparsity pattern. For example, consider a function with the following Hessian.

$$\mathsf{H}f(x) = \begin{pmatrix} h_{11} & 0 & h_{31} & 0 & 0 \\ 0 & h_{22} & 0 & h_{42} & 0 \\ h_{31} & 0 & h_{33} & 0 & h_{53} \\ 0 & h_{42} & 0 & h_{44} & 0 \\ 0 & 0 & h_{53} & 0 & h_{55} \end{pmatrix} \tag{20}$$

Note the subscripts on the elements take the symmetry of the Hessian into account.

Suppose $M = 2$, and define the colors through the following $G$ matrix.

$$G = \begin{pmatrix} \delta & 0 \\ \delta & 0 \\ 0 & \delta \\ 0 & \delta \\ \delta & 0 \end{pmatrix} \tag{21}$$

Variables 1 and 2 have color 1, and variables 3, 4 and 5 have color 2. For the moment, we will postpone the discussion of how to choose $C$ and how to color the variables.

Next, compute the columns of $Y$ using Equation 19. We now have the following system of linear equations from $\mathsf{H}f(x)G = Y$.

$$\begin{aligned} h_{11} &= y_{11} & h_{31} &= y_{12} \\ h_{22} &= y_{21} & h_{42} &= y_{22} \\ h_{31} + h_{53} &= y_{31} & h_{33} &= y_{32} \\ h_{42} &= y_{41} & h_{44} &= y_{42} \\ h_{55} &= y_{51} & h_{53} &= y_{52} \end{aligned} \tag{22}$$

Note that this system is overdetermined; Curtis et al. (1974) did not assume that their Jacobian is symmetric. Both $h_{31} = y_{12}$ and $h_{53} = y_{52}$ can be determined directly, but $h_{31} + h_{53} = y_{31}$, and $h_{42}$ could be either $y_{41}$ or $y_{22}$. Powell et al. (1979) prove that it is sufficient to solve $LD = Y$ via a substitution method, where $L$ is the lower triangular part of $H$. This has the effect of removing the equations $h_{42} = y_{22}$ and $h_{31} = y_{12}$ from the system, but retaining $h_{53} = y_{52}$. We can then solve for $h_{31} = y_{31} - h_{53} = y_{31} - y_{52}$. Thus, we have determined a $5 \times 5$ Hessian matrix with only three gradient evaluations, in contrast with the six that would have been needed had $H$ been treated as dense. Solving $LG = Y$ is known as either an "indirect," or "triangular substitution" method. *sparseHessianFD* implements only substitution methods, and we will consider only those methods in this paper.

## 4.2 Partitioning the variables

Powell et al. (1979) showed that the following rule for partitioning the variables is consistent with a triangular substitution method: two variables cannot be in the same group if their respective columns have a non-zero element in the same row. Certainly the grouping in the last example meets that criterion, but it is not the only possible grouping. In fact, a valid, yet trivial grouping would have been to define five groups, each with one variable. An objective of sparse Hessian estimation is to minimize the number of required gradient evaluations. And that means to minimize the number of groups.

Coleman et al. (1984) were the first to characterize the partitioning of the variables as a graph coloring problem. If each variable is a vertex in an undirected graph, then the sparsity pattern is an adjacency matrix. Each non-zero element in the lower triangle of the Hessian represents an edge between two vertices.

For example, look at Figure 1. Figure 1a presents a sparsity pattern in the form of an adjacency matrix, and Figure 1b represents the pattern as a graph. To estimate the Hessian associated with the sparsity pattern, we can partition the variables using three colors. In the adjacency matrix, we can see that there are no columns of the same color with a non-zero element in the same row. In the graph, we see that the coloring is valid, in that there are no two connected vertices with the same color.

$$\begin{array}{c|ccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
1 & 1 & & & & & & \\
2 & 1 & 1 & & & & & \\
3 & 1 & 1 & 1 & & & & \\
4 & 1 & 0 & 0 & 1 & & & \\
5 & 1 & 0 & 0 & 1 & 1 & & \\
6 & 1 & 0 & 0 & 0 & 0 & 1 & \\
7 & 1 & 0 & 0 & 0 & 0 & 1 & 1
\end{array}$$

(a)

(b)

Figure 1

However, not all valid colorings are consistent with a substitution method. Coleman et al. (1986) proved that a coloring is consistent if and only if the coloring is *cyclic*: any cycle in the graph must use at least three colors. This is readily apparent in Figure 1b; there is no way to get from any vertex back to itself without passing through a red, a blue and a green vertex.

For this particular sparsity pattern, a "greedy" coloring algorithm will work. We start with variable 1, and color it green. Since variable 1 is connected to every other variable, no other variable can be green, so we color variable 2 blue. Variable 3 can be neither green nor blue, so we color it red. Coloring variables 4 and 6 blue, and variables 5 and 7 red, completes a cyclic coloring of the graph.

Powell et al. (1979), Coleman et al. (1984) and Coleman et al. (1986) all note that the ordering of the variables matters when determining the a valid coloring. Figure 2 presents the sparsity pattern from Figure 1a, but with the first and last variables switched. Because we cannot assign the same color to two columns with a non-zero in the same row, coloring the variables in sequence would lead to each variable having a unique color. Such a coloring is indeed cyclic, and would be consistent with a substitution method under the Coleman et al. (1986) criterion. But it is hardly the best cyclic coloring available.

The *sparseHessianFD* package implements the "smallest-last ordering," first described in Powell et al. (1979). Other coloring algorithms have been proposed since (Coleman et al., 1986; Gebremedhin et al., 2007; Gebremedhin et al., 2005), but this algorithm is conceptually simple and easy to implement using standard data structures. We include a detailed presentation of the algorithm as Algorithm 1, but the basic idea is as follows.

1. For Color $i$, start with the graph of the sparsity structure for the Hessian after removing the rows and columns of previously colored variables. All remaining vertices in the graph are "candidates" for the current color.
2. Let variable $r$ be the candidate vertex with the highest degree. Assign $r$ to color $i$, and remove it from both the candidate list and the graph of uncolored variables.

$$\begin{array}{c|ccccccc}
 & 7 & 2 & 3 & 4 & 5 & 6 & 1 \\
\hline
7 & 1 & & & & & & \\
2 & 0 & 1 & & & & & \\
3 & 0 & 1 & 1 & & & & \\
4 & 0 & 0 & 0 & 1 & & & \\
5 & 0 & 0 & 0 & 1 & 1 & & \\
6 & 1 & 0 & 0 & 0 & 0 & 1 & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}$$

Figure 2

3. Remove from the candidate list all of $r$'s neighbors, and their neighbors.
4. If some candidates remain, return to Step 2 to assign more vertices to color $i$.
5. If the candidate list is empty, move to the next color and return to Step 1.

For the sparsity patterns in Figures 1 and 2, the algorithm would proceed as follows:

1. Variable 1 has the highest degree, so color that vertex green. Since 1 is connected to all other variable, no other variable can be colored green. Variable 1 is now removed from the graph of uncolored vertices.
2. Variables 2, 4 and 6 all have the same degree, so we color vertex 2 blue. It's only distance-1 or distance-2 neighbor is vertex 3, so we remove it from the blue candidate set. S

**Algorithm 1** Algorithm to generate a substitution-consistent coloring of variables

```
Require: Rows and columns are labeled as consecutive integers $1, \ldots, M$
Require: $P, Q$, are lists of $M$ sets (initially identical), where each $P[i]$ or $Q[i]$ is the set of column indices of
  the non-zero values in row $i$
Require: $W$, a stack of sets
Require: $D$, a vector of size $M$
Require: $S$, $A$ and $N$ are ordered sets
  for $i = 1$ to $M$ do
    $D[i] \leftarrow size(P[i])$
  end for
  Copy $P \rightarrow Q$
  $c \leftarrow 0$
  Insert $\{1, \ldots, M\} \rightarrow S${indices of all variables}
  Initialize set $W[c]$
  while $S$ is not empty do
    Copy $S \rightarrow A$
    while $A$ is not empty do
      $r \leftarrow$ index of max$(D)$.
      Insert $r \rightarrow W[c]$
      Remove $r$ from $S$
      for each $j$ in $A$ do
        Clear $N$
        $N \leftarrow P[r] \cap Q[j]${Remaining 2-degree neighbors}
        if $N$ is not empty then
          $D[j] \leftarrow 0$
          Remove $j$ from $A$
        end if
      end for
      Clear $P[r]$
    end while
    $D[1, \ldots, K] \leftarrow 0$
    for each $i$ in $S$ do
      for each $j$ in $W[c]$ do
        Remove $j$ from $P[i]$
        $D[i] \leftarrow size(P[i])$
      end for
    end for
    $c \leftarrow c + 1$
  end while
```

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 |  |  |  |  |  |  |
| 2 | 1 | 1 |  |  |  |  |  |
| 3 | 0 | 0 | 1 |  |  |  |  |
| 4 | 0 | 0 | 1 | 1 |  |  |  |
| 5 | 0 | 0 | 0 | 0 | 1 |  |  |
| 6 | 0 | 0 | 0 | 0 | 1 | 1 |  |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 |  |  |  |  |  |  |
| 2 | 1 | 1 |  |  |  |  |  |
| 3 | 0 | 0 | 1 |  |  |  |  |
| 4 | 0 | 0 | 1 | 1 |  |  |  |
| 5 | 0 | 0 | 0 | 0 | 1 |  |  |
| 6 | 0 | 0 | 0 | 0 | 1 | 1 |  |
| 7 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

## 4.3 Partitioning the variables

This rule tells us how to exclude certain groupings, but not how to find an optimal one, with the fewest possible number of groups. Coleman et al. (1984) were the first to recognize that the partitioning task is actually a graph coloring problem. The sparsity pattern of $L$ is an adjacency matrix in an undirected graph.

If $h_{i,j} \neq 0$ , then variables $i$ and $j$ are "neighbors" in the graph, and they cannot have the same color. Now, let's introduce a new variable $k$, where $j$ and $k$ are neighbors, but $i$ and $k$ are not. The colors of $i$ and $k$ still have to be different, because they have a common neighbor in $j$.

For a partition to be compatible with estimating a Hessian via substitution, it is sufficient, but not necessary, that no variable have the same color as any other variable with which it shares a common neighbor. That is, no variable within two "steps" on the undirected graph may have the same color. But in the example above, variables 1 and 5 have the same color, even though 1 is connected to 3, and 3 is connected to 5. Coleman et al. (1986) prove that the rule that no same-colored columns may have non-zero elements in the same row is equivalent to an "acyclic" graph coloring scheme, and that coloring the variables in this way is also sufficient for estimating a sparse Hessian. Thus, as Gebremedhin et al. (2009) point out, minimizing the number of partitions is equivalent to minimizing the number of colors in an acyclic graph.

Define acyclic.

## 4.4 Note: Who proved what

Consistent partitioning of $L$: No column in the same group has a non-zero element in the same row.

Powell et al. (1979) showed that a consistent partitioning of $L$ allows for a substitution method (mentioned by Coleman et al. (1984). This is a substitutable partition.

Powell et al. (1979) show that the order in which variables are assigned to partitions affects the partition.

Coleman et al. (1984) characterize Powell et al., 1979 as a graph coloring problem. Also justify using the smallest-last ordering on the lower triangle to color the variables

Coleman et al. (1986): there are other substitutable partitions than lower triangular. Theorem 2.2. A mapping induces a substitution method if and only if the mapping is a cyclic coloring.

Cyclic coloring: At least 3 colors in every cycle.

From Coleman et al. (1986), general result of substitutable (beyond Powell et al. (1979). Order nonzero elements $1 \ldots M$. If, for nonzero element $(i_m, j_m)$,

1. columns $j_m$ and another other $j_{m'}$ are in the same group, and both $j_m$ and $j_{m'}$ have a nonzero in row $i_m$, then $i_{m'}, j_{m'}$ must be ordered before $i_m, j_m$; *or*
2. columns $i_m$ and another other $i_{m'}$ are in the same group, and both $i_m$ and $i_{m'}$ have a nonzero in row $j_m$, then $j_{m'}, i_{m'}$ must be ordered before $j_m, i_m$

What this means is that we can ignore column intersections that occur in lower rows (what?).

The point is that lower triangular substitution qualifies, which means that we just need a cyclic coloring of the graph.

Note: My coloring algorithm is a smallest last ordering of the full symmetric Hessian This is *slpt* in Coleman et al. (1984). What I should really do is a smallest-last on the lower triangle (*slsl* in citetColemanMore1984. So I need to change that.

In any event, it's still finding a cyclic coloring of the adjacency graph. Just using a different heuristic for the coloring.