Title: On Diagonally Structured Problems in Unconstrained Optimization using an Inexact Super Halley Method

Article Type: Special Issue: Proceedings of ICCAM 2010

Section/Category: Special Issue: ICCAM 2010

Keywords: Halley's method; Chebyshev's method; Inexact Newton methods; Large Scale unconstrained Optimization

Corresponding Author: Dr. Trond Steihaug, Ph.D.

Corresponding Author's Institution: University of Bergen

First Author: Trond Steihaug, PhD

Order of Authors: Trond Steihaug, PhD; Geir Gundersen, PhD
On Diagonally Structured Problems in Unconstrained Optimization using an Inexact Super Halley Method

Geir Gundersen\textsuperscript{b}, Trond Steihaug\textsuperscript{a}

\textsuperscript{a}Department of Informatics, University of Bergen, Bergen, Norway
\textsuperscript{b}Department of Informatics, University of Bergen, Bergen, Norway

Abstract

We consider solving the unconstrained minimization problem using an iterative method derived from the third order Super Halley method. The Super Halley method requires solution of two linear systems of equations. We show a practical implementation using an iterative method to solve the linear systems. This paper introduces an array of arrays (jagged) data structure for storing the second and third derivative of a multivariate function and suitable termination criteria for the (inner) iterative method to achieve a cubic rate of convergence. Using a jagged compressed diagonal storage of the Hessian matrices and for the tensor, numerical results show that storing the diagonals are more efficient than the row or column oriented approach when we use an iterative method for solving the linear systems of equations.

\textit{Keywords:} Halley’s method, Chebyshev’s method, Systems of equations

\textit{2000 MSC:} 65K05, 90C06, 65H10

1. Introduction

In this paper we consider the unconstrained optimization problem

\[
\min_{x \in \mathbb{R}^n} f(x)
\]  

where \( f : \mathbb{R}^n \to \mathbb{R} \) is three times continuously differentiable in a neighborhood of the strict local minimizer \( x^* \). To solve (1) we will consider methods which utilizes the third derivative of the multivariate function \( f \).

Gutiérrez and Hernández [1] defined a class of methods for solving the nonlinear system of equations \( F = 0 \), where \( F : \mathbb{R}^n \to \mathbb{R}^n \). We apply this class on the unconstrained optimization problem by setting \( F = \nabla f \) and we introduce the notation that for a given \( x \in \mathbb{R}^n \) \( g = \nabla f(x) \), \( H = \nabla^2 f(x) \), and \( T = \nabla^3 f(x) \). For a starting point \( x_0 \) the iterates in the Halley Class are defined by

\[
x_{k+1} = x_k - \left( I + \frac{1}{2} L_k [I - \alpha L_k]^{-1} \right) H_k^{-1} g_k, \quad k \geq 0
\]  

where \( I \) is the identity matrix and \( L_k = H_k^{-1} (H_k^{-1} g_k T^{(k)}) \).
For $\alpha = 0$ the method is the called Chebyshev’s method [2], Halley’s method is $\alpha = \frac{1}{2}$ [3, 4, 2], and for $\alpha = 1$ the method is referred to as Super Halley’s method [5, 1, 6]. Members of the Halley class have a third order rate of convergence [7].

These methods are known as single steps methods since all the derivatives used in one iteration are computed at the same point. Gundersen and Steihaug [8] show that the Halley class of iterations can be rewritten as

\[
\text{Given } x_0 \\
\text{for } k = 0, 1, 2, \ldots \text{ until convergence do} \\
\quad \text{Solve } H_k s_k^{(1)} = -g_k \\
\quad \text{Solve } \left( H_k + \alpha(s_k^{(1)} T^{(k)}) \right) s_k^{(2)} = -\frac{1}{2}(s_k^{(1)} T^{(k)}) s_k^{(1)} \\
\quad \text{Set } x_{k+1} = x_k + s_k^{(1)} + s_k^{(2)}. \\
\text{end for}
\]

In the two formulations we have introduced the notation $(sT)_{ij} = \sum_{k=1}^{n} T_{ijk}s_k$ and $(sT)$ is a symmetric $n \times n$ matrix. Two linear systems of equations for solving $s_k^{(1)}$ and $s_k^{(2)}$ and one tensor operations to compute $(sT)$ have to be performed. To compute $(sT)s$ we need a matrix vector multiplication. The two coefficient matrices are symmetric and will be positive definite close to a strict minimizer.

Gundersen and Steihaug [9] used direct methods for solving these two systems of equations. but experienced fill-ins when performing the $LDL^T$ factorization. For large problems the amount of fill-ins was substantial and the number of floating point arithmetic operations in the $LDL^T$ factorization totally dominated the tensor operations. Iterative methods can for those problems be very attractive.

Deng et al [10], Zhang [11] and Gui-Feng et al [12] considers the special case where $\alpha = 0$ and propose to solve the linear systems with a preconditioned conjugate gradient method. If an iterative method for the linear systems of equations method is terminated with an approximate solution $s_k^{(j)}$ there will be an error $r_k^{(j)}$ in the equation.

\[
\text{Given } x_0 \\
\text{for } k = 0, 1, 2, \ldots \text{ until convergence do} \\
\quad \text{Solve approximately: } H_k s_k^{(1)} = -g_k \text{ with residual } r_k^{(1)} = H_k s_k^{(1)} + g_k \\
\quad \text{Solve approximately } H_k s_k^{(2)} = -\frac{1}{2}(s_k^{(1)} T^{(k)}) s_k^{(1)}; r_k^{(2)} = H_k s_k^{(2)} + \frac{1}{2}(s_k^{(1)} T^{(k)}) s_k^{(1)} \\
\quad \text{Set } x_{k+1} = x_k + s_k^{(1)} + s_k^{(2)}. \\
\text{end for}
\]

Deng [10] suggests using the termination

$$
\|r_k^{(j)}\| \leq \|g_k\|^{3+\epsilon}
$$

where $\epsilon > 0$ is small.

In this paper we present an inexact Super Halley method using an iterative method for solving the systems of linear equations. We will derive the inexact Halley method from the observation in [8] that one step of super Halley is two step of Newton’s method on a cubic scalar function.
The most time consuming operation in iterative methods for solving linear systems of equations is the matrix-vector operation. The efficiency of the matrix vector product depends heavily on the data structure. In this paper we will use a variation of Compressed Diagonal Storage scheme which is a preferred datastructure for matrix vector product when the matrix has a diagonal structure.

We will show that a large class of partially separable functions gives a diagonally structured second derivative. When a matrix has a diagonal structure, we usually store such a matrix by diagonals. For a tri-diagonal matrix we are traversing arrays of length $O(n)$ instead of row or column based arrays or length $3$ in a matrix vector product. It is well known that traversing large arrays instead of small arrays can lead to more efficient algorithms on parallel and vector processors [13, page 41-46].

We will show how to efficiently implement tensor operations and matrix vector multiplications using compressed diagonal storage using programming languages that allow the construction of array of arrays. Since the induced tensor inherits its structure from the matrix, it also has a diagonal structure.

2. The Inexact Super Halley Method

Consider the second order approximation to the function $\nabla f$ at $x_k + s$

$$T(s) = g + Hs + \frac{1}{2}(sT)s$$

where we have eliminated the iteration index. To solve the quadratic equation $T(s) = 0$ we use two iterations of the Inexact Newton’s method. For a given initial value $s^{(0)}$ two steps of an inexact Newton method [14] are given by

$$T'(s^{(0)})s^{(1)} = -T(s^{(0)}) + r^{(1)}, \quad \|r^{(1)}\| \leq \eta^{(1)}\|T(s^{(0)})\|$$
$$T'(s^{(0)} + s^{(1)})s^{(2)} = -T(s^{(0)} + s^{(1)}), \quad \|r^{(2)}\| \leq \eta^{(2)}\|T(s^{(0)} + s^{(1)})\|$$

where $s^{(1)} = s^{(0)} + s^{(1)} + s^{(2)}$.

The new iteration will be $x_{k+1} = x_k + s$. By using the fact that $T'(s) = H + (sT)$ and $s^{(0)} = 0$ we see that $s^{(1)}$ will satisfy $Hs^{(1)} + g = r^{(1)}$. Further, $T(s^{(0)} + s^{(1)}) = T(s^{(1)}) = g + Hs^{(1)} + \frac{1}{2}(s^{(1)}T)s^{(1)} = r^{(1)} + \frac{1}{2}(s^{(1)}T)s^{(1)}$. We can now formulate a step of an Inexact Super Halley.

$$Hks^{(1)} = -g_k + r_k^{(1)}, \quad \|r_k^{(1)}\| \leq \eta_k^{(1)}\|g_k\|$$
$$b_k = x_k^{(1)} + \frac{1}{2}(s_k^{(1)}T(s_k^{(1)}))s_k^{(1)}$$
$$\left(H_k + (s_k^{(1)}T^{(k)})\right)s_k^{(2)} = -b_k + r_k^{(2)}, \quad \|r_k^{(2)}\| \leq \eta_k^{(2)}\|b_k\|$$
$$x_{k+1} = x_k^{(1)} + s_k^{(1)} + s_k^{(2)}.$$ (3)

Using an exact solver the residual $r^{(1)} = r^{(2)} = 0$ and we see that the Inexact Super Halley reduces to the Super Halley method with $\alpha = 1$.

We propose the following tolerances for terminating the iterative method

$$\eta_k^{(j)} \leq \min\{\eta, \|g_k\|\}, \quad j = 1, 2.$$ (4)
where $\eta < 1$ is something to be adjusted. For more details on inexact Newton methods and the choice of $\eta$ see for example [14]. Assume that $x^*$ is a strict local minimizer of (1) and that $f$ is three times continuously differentiable in a neighborhood of $x^*$ and the tensor $\nabla^2 f$ is Lipschitz continuous. The next theorem shows that the Inexact Super Halley is locally convergent with a q-order 3.

**Theorem 2.1.** Assume that $\eta^{(j)}_{k} \leq \eta < 1$, $j = 1, 2$. There exist an $\varepsilon > 0$ and $1 > \rho > \eta$ so that for $\|x_0 - x^*\| \leq \varepsilon$ all iterates $x_k$ in (3) will satisfy $\|x_k - x^*\| \leq \varepsilon$ and $\|\nabla f(x_{k+1})\| \leq \rho\|\nabla f(x_k)\|$.

If $\eta^{(j)}_{k}$, $j = 1, 2$ given in (4) hold then there exists $C$ so that

$$\|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^3$$

for $k$ sufficiently large.

**Proof.** Consider

$$T(s^{(1)} + s^{(2)}) = g + H(s^{(1)} + s^{(2)}) + \frac{1}{2}((s^{(1)} + s^{(2)})T)((s^{(1)} + s^{(2)})$$

$$= r^{(1)} + (H + (s^{(1)}T))s^{(2)} + \frac{1}{2}(s^{(2)}T)s^{(2)}$$

$$= r^{(2)} + \frac{1}{2}(s^{(2)}T)s^{(2)}$$

Let $\eta < \bar{\eta}$. Choose $\varepsilon > 0$ and $1 > \delta > 0$ sufficient small and let $K$ be so that

$$\|(\nabla^2 f(x))^{-1}\| \leq K,$$

$$\|(\nabla^2 f(x) + (\nabla^2 f(x))^{-1}\nabla f(x)T(x))^{-1}\| \leq K,$$

$$\|T(x)\| \leq 2K,$$

$$\eta + K^3\|\nabla f(x)\| \leq \bar{\eta},$$

$$\|\nabla f(z) - T(z - x)\| \leq K\|z - x\|^3,$$

$$\bar{\eta}^2 + K^4 (1 + \bar{\eta})^3\|\nabla f(x)\|^2 \leq \rho < 1$$

and

$$(1 - \delta)\|\nabla^2 f(x^*)(x - x^*)\| \leq \|\nabla f(x)\| \leq (1 + \delta)\|\nabla^2 f(x^*)(x - x^*)\|$$

for all $x$ and $z$ in a neighborhood of $x^*$. It follows that

$$\|s^{(1)}\| \leq K\|g\|,$$

$$\|b\| = \|r^{(1)} + \frac{1}{2}(s^{(1)}T)s^{(1)}\| \leq \eta\|g\| + K\|s^{(1)}\|^2 \leq (\eta + K^3\|g\|)\|g\| \leq \bar{\eta}\|g\|$$

and

$$\|s^{(2)}\| \leq K\|b\| \leq K\bar{\eta}\|g\|.$$

We now have

$$\|T(s^{(1)} + s^{(2)})\| \leq \|r^{(2)}\| + K\|s^{(2)}\|^2$$

$$\leq \eta\|b\| + K^2\|s^{(2)}\|\|b\|$$

$$\leq (\eta + K^3\|g\|)\|b\| \leq \bar{\eta}^2\|g\|$$
and finally
\[ \| \nabla f(x + s^{(1)} + s^{(2)}) \| \leq \| T(s^{(1)} + s^{(2)}) \| + L \| s^{(1)} + s^{(2)} \|^3 \]
\[ \leq (\eta^2 + K^4(1 + \bar{\eta})^3g^2)\| \nabla f(x) \| \leq \rho \| \nabla f(x) \| \]

We now need to show that the next iterate will be in the neighborhood
\[ (1 - \delta)\| x + s^{(1)} + s^{(2)} - x^* \|, \leq \| \nabla f(x + s^{(1)} + s^{(2)}) \| \leq \rho \| \nabla f(x) \| \leq \rho(1 + \delta)\| x - x^* \|. \]

Let \( \delta \) be chosen so that \( \rho(1 + \delta)/(1 - \delta) < 1 \) and the new point also is in the neighborhood.

To show the third order rate of convergence, choose iterations sufficiently close to \( x^* \) so that
\[ \| r^{(1)}_k \| \leq \eta^{(1)}_k \| g_k \| \leq \| g_k \|^2 \text{ and } \| r^{(2)}_k \| \leq \eta^{(2)}_k \| b_k \| \leq \| g_k \| \| b_k \|. \]

We also have that \( \| b_k \| = \| r^{(1)}_k + \frac{1}{2}(s^{(1)}_k T_k) s^{(1)}_k \| \leq (1 + K^3)\| g_k \|^2 \) and we have \( \| s^{(2)}_k \| \leq K\| b_k \| \leq K(1 + K^3)\| g_k \|^2 \). Then
\[ \| T_k(s^{(1)}_k + s^{(2)}_k) \| \leq \| r^{(2)}_k \| + K\| s^{(2)}_k \|^2 \leq (1 + K^3)\| g_k \|^3 + K^3(1 + K^3)^2\| g_k \|^4 \]
and we can conclude that
\[ \| \nabla f(x_{k+1}) \| \leq ((1 + K^3) + K^3(1 + K^3)^2 + 8K^3) \| \nabla f(x_k) \|^3 \]
using that \( \| s^{(1)}_k + s^{(2)}_k \| \leq 2K\| g_k \| \), and \( \| g_k \| \leq 1 \).

\( \square \)

3. Induced Sparsity for Diagonally structured Hessians

Since the function \( f \) is three times continuously differentiable we can utilize the symmetries in the second and third derivatives. Super-symmetry arises when two mixed 3rd order partial derivatives of a function \( f \) involve the same differentiation but in different orders. If those mixed 3rd order partial derivatives are continuous in a neighborhood of the given point, then the two mixed partial derivatives are equal. The tensor element
\[ T_{ijk} = \frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f(x) \]
and
\[ T_{ijk} = T_{ikj} = T_{jki} = T_{kij} = T_{kji}. \]

The only nonzero elements of the Hessian matrix we need to consider are for which \( j \leq i \) and \( k \leq j \leq i \) for the tensor.

The Hessian matrix is called sparse if most of the elements \( H_{pq} \) are zero for all \( x \in \mathbb{R}^n \)
[15]
\[ \frac{\partial^2}{\partial x_p \partial x_q} f(x) = 0, \ \forall x \in \mathbb{R}^n. \]  \hspace{1cm} (5)

Let \( Z \) be the set of pairs of indices \( (p, q) \) for which (5) hold and let \( N \) be the corresponding nonzero elements
\[ N = \{ (p, q)|\forall p, q \} \backslash Z. \]  \hspace{1cm} (6)

\( N \) will be the set of indices for which the elements of the Hessian matrix at \( x \) in general will be nonzero. Without loss of generality we assume that \( (p, p) \in N, \ \forall p \). It follows
immediately that the elements in the matrix \((sT)\) will be zero where the elements \(\nabla^2 f\) are zero.

Consider element \(T_{pqr}\) in the tensor with indices \((p, q, r)\). Using the super symmetry the element is non-zero when

\[(p, q) \in \mathcal{N}, (p, r) \in \mathcal{N}, \text{ and } (q, r) \in \mathcal{N}.
\]

We say that the sparsity of the tensor is induced by the sparsity of the Hessian.

3.1. Diagonal Structured Hessian Matrices

A general matrix \(A\) of order \(n\) has \(2^n - 1\) diagonals. Since \(k = i - j\) is constant for the elements \(A_{ij}\) along each diagonal, each diagonal can be assigned a diagonal number \(k\). For a symmetric matrix \(H\) we only consider elements \(H_{ij}\), where \(j \leq i\) and each element \(H_{ij}\) is associated with a non-negative diagonal number \(i - j\). To be consistent with the chosen language we let the first index in a vector be 0 and similarly \((0,0)\) is the pair of indices of the upper left hand element in the matrix. The elements of \(H\) in diagonal \(k\) are then

\[H_{k+j,j} = 0, \ldots, n - 1 - k.\]

The diagonal number is also the row number of the first element in the diagonal.

Let \(i_0 = 0 < i_1 < i_2 < \cdots < i_\beta \leq n - 1\) be the diagonals and let \(I = \{i_0, ..., i_\beta\}\).

The illustrate the use of diagonals in the method for matrix vector product \(y = Hs\) where \(H\) is a symmetric matic with diagonals in the set \(I\) and \(s \in \mathbb{R}^n\) is shown in Algorithm 1 in Figure 2.

3.2. The Induced Tensor

Consider element \(T_{pqr}\) in the tensor with indices \((p, q, r)\). The element is non-zero provided (7) holds. Rewrite the indices as \((i + k, j + k, k) = (p, q, r)\) and we have the equivalent formulation to (7)

\[(i + k, j + k) \in \mathcal{N}, (j + k, k) \in \mathcal{N}, (i + k, k) \in \mathcal{N}.
\]

We note that the element with indices \((i + k, j + k)\) in diagonal \(i - j\), \((j + k, k)\) is in diagonal \(j\), and \((i + k, k)\) is in diagonal \(i\) of the Hessian matrix.

We define a diagonal of the super-symmetric tensor \(T\) to be the a pair \((i, j), \ j \leq i\) with the elements of the tensor with indices

\[(i + k, j + k, k), \ k = 0, \ldots, n - 1 - i.
\]

Let \(\mathbb{I} \subseteq \{(i, j) : i = 0, 1, ..., n - 1, j \leq i\}\) be the diagonals of the tensor. Element \(T_{pqr}\) of the tensor will be in the diagonal \((p - r, q - r)\).

We can then make the following observations about a tensor induced by a banded Hessian matrix with diagonals \(I\).

- \(\mathbb{I} \subseteq I \times I\)
- \((i, j) \in \mathbb{I}\) if and only if \(i \in I, j \in I\) and \(i - j \in I\).
- If \(I = \{0, \ldots, \beta\}\) then \(\mathbb{I} = \{(i, j) | i = 0, \ldots, \beta, j = 0, \ldots, i\}\).

The first observation implies that the induced tensor has at most \(|I|(|I| + 1)/2\) diagonals for \(j \leq i\) which is the numbers of diagonals in the case when the Hessian is a bandmatrix with (half) bandwidth \(\beta = |I| - 1\). (\(|\cdot|\) is the number of elements in the set).
4. Jagged Compressed Diagonal Storage (JCDS)

The compressed diagonal storage scheme is for matrices that are banded with bandwidth that is fairly constant from row to row. It is worthwhile to take advantage of this structure in the storage scheme by storing subdiagonals of the matrix in consecutive memory locations. The compressed diagonal storage (CDS), is a standard storage format and described extensively in [16, 17].

CDS storage mode requires entire diagonals to be stored \((n)\) elements in a two dimensional array of size the numbers of diagonals \((\beta + 1)\) and number of rows \((n)\). If the nonzero elements in the matrix are not concentrated along the main diagonal, this storage mode requires extra storage. Jagged compressed diagonal storage is similar to compressed diagonal storage but using array of arrays where each diagonal is stored as an one dimensional array of length \(n - k\) where \(k\) is the diagonal number. This is the storage scheme used in the Matrix Toolkits for Java\(^1\).

We say that the Hessian is stored by jagged compressed diagonal storage (JCDS) when we store:

- A two-dimensional floating point array \(H\) for storing the numerical elements in the Hessian matrix. The number of elements of the Hessian diagonals are the size \(n - i\), where \(i\) is the diagonal number.
- A one-dimensional integer array \(Iptr\) of size the number of nonzero Hessian diagonals containing the diagonal number for each nonzero diagonal in the Hessian matrix.

Consider the matrix and JCDS storage\(^2\) in Figure 1. For \(j \leq i\) the first digit in \(H_{i,j}\) is \(i + 1\) and the second digit is \(j + 1\).

The array \(Iptr\) stores the indices of the diagonals \(I\) and \(numdia = \beta + 1\). If \(i \in I\) let \(k\) be so that \(Iptr[k] = i\) then the elements in diagonal \((8)\) will be

\[
Hval[k][j] \text{ for } j = 0, \ldots, n - 1 - i.
\]

Using array of arrays this will for \(diag = Hval[k]\)

\[
diag[j] \text{ for } j = 0, \ldots, diag.length - 1,
\]

\(^1\)http://code.google.com/p/matrix-toolkits-java/

\(^2\)In Java, ANSI C, and C# all arrays are zero based as opposed to MATLAB and Fortran in which arrays are one based.
Algorithm 1 Computing $y \leftarrow Hs$.

\begin{algorithm}
for $j = 0$ to $n - 1$ do
  $y_j = H_{j,j}s_j$
end for

for $i \in I \land i > 0$ do
  for $j = 0$ to $n - 1 - i$ do
    $y_{i+j} = y_{i+j} + H_{i+j,j}s_j$
    $y_j = y_j + H_{i+j,j}s_{i+j}$
  end for
end for
\end{algorithm}

Figure 2: Matrix vector product using diagonals. $\text{Iptr.length=}|I|$.

where $\text{diag}$ is a one dimensional array.

Traversing only the diagonals (arrays of arrays) of length $O(n)$ in the innermost loop we avoid loop overhead compared to traversing row sized arrays. Therefore it can be highly efficient on parallel and vector processors.

JCDs is very useful for iterative methods for solving the system of linear equations. A generalization of the compressed diagonal storage (CDS) format more suitable for manipulating general sparse matrices on vector supercomputers is discussed in Melhem [18].

To illustrate the use of JCDs in Java consider the matrix vector product given in Algorithm 1. In Figure 2 we also show a implementation in Java.

5. The $(pT)$ Tensor Operation

The algorithm for the operation $(pT)$ is shown in Algorithm 2.
Algorithm 2 Computing $H \leftarrow H + (pT)$.

Let $T \in \mathbb{R}^{n \times n \times n}$ be a super symmetric tensor.
Let $H \in \mathbb{R}^{n \times n}$ be a symmetric matrix with diagonal numbers in $I$.
Let $p \in \mathbb{R}^n$ be a vector.

for $i = 0$ to $n - 1$ do
    $H_{i,i} = H_{i,i} + p_i T_{i,i,i}$
end for

for $i \in I$ do
    for $k = 0$ to $n - 1 - i$ do
        $H_{i+k,k} = H_{i+k,k} + p_k T_{i+k,k,k}$
        $H_{k,k} = H_{k,k} + p_{i+k} T_{i+k,k,k}$
    end for
    for $j \in I$ and $i - j \in I$ and $0 < j < i$ do
        for $k = 0$ to $n - 1 - i$ do
            $H_{i+k,j+k} = H_{i+k,j+k} + p_k T_{i+k,j+k}$
            $H_{i+k,k} = H_{i+k,k} + p_{j+k} T_{i+k,j+k,k}$
            $H_{j+k,k} = H_{j+k,k} + p_{i+k} T_{i+k,j+k,k}$
        end for
    end for
end for

for $k = 0$ to $n - 1 - i$ do
    $H_{i+k,i+k} = H_{i+k,i+k} + p_k T_{i+k,i+k,k}$
end for

The total number of arithmetic operations in Algorithm 2 is $6 \text{nnz}(T) - 4 \text{nnz}(H)$. We note that if $I = \{0, \ldots, \beta\}$, then the computation in the for-loop can be simplified.

6. Data Structure for Storing the Induced Tensor

We say that the tensor is stored by jagged compressed diagonal storage for induced tensor (JCDST) when we store:

- A one-dimensional integer array $\text{Iptr}$, of size the number of nonzero Hessian diagonals, containing the (first element) row indices for each nonzero diagonal in the Hessian matrix.
- A two-dimensional integer sparse matrix $\text{IIptr}$, of size the number of nonzero tensor diagonals, that store the $j$ indices for each nonzero diagonal in the tensor, where $T_{i,j,k}$ is a tensor element $(i, j) \in I$. Note that the $k$ indices for each given $(i, j)$ starts with the index 0.
- A Three-dimensional floating point array $\text{Tval}$ of size $\text{nnz}(T)$ for storing the numerical values in the tensor.

The jagged compressed diagonal storage for induced tensors (JCDST) induced from the matrix in Figure 1 is given in Figure 3. The set $I$ of diagonals of $H$ and the set $\mathbb{I}$ of
double[][] Tval = {{{111,222,333,444,555,666,777}},
{{311,422,533,644,755},{331,442,553,664,775}},
{{411,522,633,744},{441,552,663,774}},
{{611,721},{631,742},{641,752},{661,772}}};

int[][] IIptr = {{0},{0,2},{0,3},{0,2,3,5}};

int[] Iptr = {0,2,3,5};

Figure 3: The datastructure for induced Hessian in Figure 1

diagonals of the tensor are

I = \{0, 2, 3, 5\} and I = \{(0,0), (2,0), (2,2), (3,0), (3,3), (5,0), (5,2), (5,3), (5,5)\}.

Note (3, 2) is not a diagonal in the induced tensor since 1 \notin I

7. A Numerical Comparison Between Row and Diagonal oriented storage schemes

We follow the Compressed Row Storage (CRS) approach for storing general sparse matrices to create a storage scheme for tensors. CRS stores \( C_i \) for \( i = 0, \ldots, n-1 \) which is the set of column indices of the nonzero elements in row \( i \) below or on the diagonal. The indices of the nonzero elements in the induced tensor (7) will be

\[ T = \{(i,j,k)|i = 0, \ldots, n-1, j \in C_i, k \in C_i \cap C_j\}. \quad (9) \]

The tensor operation \((pT)\) and is implemented with three nested for loops where index \( i \) runs in the outermost loop, index \( j \) runs in the middle loop, and index \( k \) runs in the inner most loop. The elements are stored in sequential memory locations as a linear array with a lexicographic order of the indices. Two of the three storage schemes follow this order of the loops while the third storage scheme JCDS stores the tensor diagonals instead.

Compressed tube storage scheme for sparse supersymmetric induced tensors stores a pointer instead of the actual \( k \)-index and was introduced as a datastructure for induced tensors from general sparse matrices combined with a direct solver [19, 9]. The storage scheme can be extended to arrays of arrays with a th ree dimensional array containing the numerical values and a three dimensional array containing the pointers to the index structure of the matrix [20].

The skyline storage scheme approach for storing banded matrices or skyline matrices stores \( f_i \), the index of first nonzero element in row \( i \). Then for \( j = f_i, \ldots, i \leftrightarrow (i,j) \in \mathcal{N} \) for \( i = 0, \ldots, n - 1 \). The elements are stored in sequential memory locations as a linear array with a lexicographic order of the indices. It follows from (7) that the set of indices of the nonzero elements in the induced tensor is

\[ T = \{(i,j,k)|i = 0, \ldots, n-1, j = f_i, \ldots, i, k = \max\{f_i, f_j, \ldots, j\}\}. \quad (10) \]

This storage scheme can also be extended to arrays of arrays [20].

In Table 1 we compare the operation \( H \leftarrow H + (pT) \) performed on compressed tube storage (CTS) [19, 9] and on jagged compressed diagonal storage for induced tensors.
(JCDST) and skyline storage of the tensor (SST) [8]. The input is a tensor induced from a \(n \times n\) band matrix with half band width \(\beta\), the tensor is stored in the three different formats. In the first column we have the problem size \(n\), the dimension of the symmetric matrix, and in the second column \(\beta\) which is the number of subdiagonals below the main diagonal of the matrix.

Using CTS the number of times the innermost loop is executed is the length of each inner array which is the intersection between two rows of the Hessian matrix. The size of these intersections are small and vary between 1 and \(\beta\) in size. For the skyline format the intersection size between the sets of the indices of two rows in the Hessian matrix will be the same as for CTS. These are significantly smaller in size compared to the \(O(n)\) sized diagonals of the JCDST storage. This difference in size of tubes and tensor diagonals have a significant influence on the computing time.

Table 1: Measuring the tensor computations in microseconds on band matrices.

<table>
<thead>
<tr>
<th>Matrix ((pT))</th>
<th>(n)</th>
<th>(\beta)</th>
<th>SST</th>
<th>CTS</th>
<th>JCDST</th>
</tr>
</thead>
<tbody>
<tr>
<td>1200</td>
<td>1</td>
<td>96</td>
<td>145</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td>1</td>
<td>180</td>
<td>305</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>4800</td>
<td>1</td>
<td>406</td>
<td>545</td>
<td>170</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>3</td>
<td>267</td>
<td>388</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td>3</td>
<td>571</td>
<td>718</td>
<td>242</td>
<td></td>
</tr>
<tr>
<td>4800</td>
<td>3</td>
<td>819</td>
<td>844</td>
<td>348</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>10</td>
<td>804</td>
<td>1341</td>
<td>505</td>
<td></td>
</tr>
<tr>
<td>2400</td>
<td>10</td>
<td>1855</td>
<td>2313</td>
<td>1015</td>
<td></td>
</tr>
<tr>
<td>4800</td>
<td>10</td>
<td>5011</td>
<td>5288</td>
<td>2557</td>
<td></td>
</tr>
</tbody>
</table>

Columns 3-5 shows the timings for computing \(H \leftarrow H + (pT)\) with the SST, CTS and JCDST formats. We see that for almost all problems SST was more efficient than CTS due to the extra look-up of the indices and the indirect addressing used in CTS. Further for all problems JCDST was more efficient than both SST and CTS due to the effect of traversing a array length of essentially the same size as the problem. This effect would probably be more prominent on parallel and vector processors.

In Table 2 we compare the same tensor operation performed using CTS and JCDST. The input is modified matrices from Matrix Market [21]. For each nonzero element in the matrix corresponding subdiagonal is extended to its full length. For all problems there diagonal numbers \(i_k < i_{k+1}\).
Table 2: Measuring the tensor computations in microseconds on diagonally structured matrices.

<table>
<thead>
<tr>
<th>Matrix based on</th>
<th>((pT))</th>
<th>CTS</th>
<th>JCDST</th>
</tr>
</thead>
<tbody>
<tr>
<td>nos1</td>
<td></td>
<td>38</td>
<td>21</td>
</tr>
<tr>
<td>nos2</td>
<td></td>
<td>226</td>
<td>66</td>
</tr>
<tr>
<td>nos4</td>
<td></td>
<td>56</td>
<td>45</td>
</tr>
<tr>
<td>nos6</td>
<td></td>
<td>185</td>
<td>44</td>
</tr>
<tr>
<td>nos7</td>
<td></td>
<td>158</td>
<td>41</td>
</tr>
<tr>
<td>gr3030</td>
<td></td>
<td>188</td>
<td>97</td>
</tr>
</tbody>
</table>

In the first column we have the name of the Matrix Market matrix that our input is based on. Columns 2-3 shows the timings for computing \(H \leftarrow H + (pT)\) with the CTS and JCDST formats. For all problems JCDST was more efficient than CTS due to the effect of traversing a array length of essentially the same size as the problem. Again this effect would probably be more prominent on parallel and vector processors.

The timing results, which are very preliminary, shown in Table 1 and 2 shows that the operations performed on a JCDST format is more efficient than the same operations performed on the CTS and SST. This can partially be explained by the effect of two things one is that traversing arrays of size \(O(n)\) instead of arrays of size \(\beta\) in the innermost for loop decreases loop overhead, two we have also the arrays of arrays effect in that we do not access a subset of a large array with JCDST as we do with CTS and SST when accessing the numerical and index values.

The test machine for all the numerical experiments done in this paper was on Intel(R) Pentium(R) 4 CPU 2.80GHz running Red Hat 4.1.2-33. The implementation was done using ANSI C with gcc (GCC) 4.1.2.

8. An Example

Consider a partially separable function on the form

\[
    f(x) = \sum_{k \in \mathcal{K}} \phi_k(x_k, x_{k+d_1}, x_{k+d_2}, \ldots, x_{k+d_\beta})
\]  

where for \(d_i \in \mathcal{I} \) and \(k \in \mathcal{K} \) then \(0 \leq k + d_i \leq n - 1\).

**Theorem 8.1.** If each element function \(\phi_k : \mathbb{R}^{\beta+1} \to \mathbb{R}\) is three times continuously differentiable then the Hessian matrix has the diagonals

\[
    I = \{|i-j| \mid i \in \mathcal{I}, j \in \mathcal{I}\}
\]  

and the third derivative has the diagonals

\[
    \mathcal{I} = \{(i,j)|i \in \mathcal{I}, j \in \mathcal{I}, i-j \in \mathcal{I}\}
\]

**Proof.** We only need to show that \(I\), given in (12), contains the diagonals of the Hessian matrices of the diagonal elements. We show this by considering \(\phi_k\) and

\[
    \frac{\partial^2}{\partial x_{k+i} \partial x_{k+j}} \phi_k(x_k, x_{k+d_1}, \ldots, x_{k+d_\beta}), \quad i \in \mathcal{I}, j \in \mathcal{I}
\]
are the only nonzero elements of the element function $\phi_k$.

Element $(k + i, k + j)$ in the Hessian matrix is in diagonal $k + i - k - j = i - j$. The
diagonals of the Hessian matrix are thus given by (12).

The second part of the theorem follows from the observation that

$$(i, j) \in I$$

if and only if $i \in I$, $j \in I$, $i - j \in I$.

Most of the generalizations of chained functions based on classical test functions will
have this form. We have modified the Chained Rosenbrock function from [22]:

$$f(x) = \sum_{k=1}^{\beta} \sum_{i=d_k+1}^{n} \left[ 4\alpha(x_{i-d_k} - x_i^2)^2 + (1 - x_i)^2 \right].$$

where $\beta + 1$ is the number of diagonals, $d_0 = 0$ and $d_k$, for $k = 1, ..., \beta$, contains the
start index $i$ for each nonzero diagonals. As shown in Figure 4 this function will have a
diagonal structure for the Hessian with $I = \{0, 1, 5, 8, 9\}$

![Figure 4: The structure of the Hessian of the modified Chained Rosenbrock function for $n = 25$ and $I = \{0, 1, 5, 8, 9\}$](image)

We present some preliminary numerical results, using the inexact Super Halley’s
method presented in the paper for solving the unconstrained optimization problem (1).
The outer iteration is terminated when $\|\nabla f(x_k)\| \leq 10^{-8}\|\nabla f(x_0)\|$. We set the tolerances a priori to be as shown in (4) for $\eta_k^{(1)}$ and $\eta_k^{(2)} \leq \min\{\eta, \|b_k\|\}$ in obtaining $s_k^{(1)}$ and $s_k^{(2)}$.

![Figure 5: CG steps in an inexact Super Halley method.](image)

In Figure 5 we show the the total number of conjugate gradient (inner) iterations that reach the tolerances and the norm of the gradient at every inexact super Halley iteration (outer) (marked with a black dot). Super Halley-CG with $\eta = \frac{1}{2}$ uses four iterations to reach the final accuracy, while for the other choices the method uses three outer iterations. The method with loose a tolerance in the iterative method is much more efficient than the one with strict tolerance.

References


