A More Robust Multigrid Algorithm for Diffusion Type Registration Models

Tony Thompson[*] and Ke Chen[*†]

Abstract

Registration refers to the useful process of aligning two similar but different intensity image functions in order to either track changes or combine information. Variational models are capable of finding transform maps containing large and non-uniform deformations between such a pair of images. Since finding a transform map is an inverse problem, as with all models, suitable regularisation is necessary to overcome the non-uniqueness of the problem. In the case of diffusion type models regularisation terms impose smoothness on the transformation by minimising the gradient of the flow field. The diffusion model also coincides with the basic model for optical flow frameworks of Horn-Schunck (1981, AI). The biggest drawback with variational models is the large computational cost required to solve the highly non-linear system of PDEs; Chumchob-Chen (2011, JCAM) developed a non-linear multigrid (NMG) method to address this cost problem. However, a closer look at the analysis of the NMG scheme highlighted omissions which affected the convergence of the NMG scheme. Moreover, the NMG method proposed by Chumchob-Chen did not impose any control of non-physical folding which invalidates a map. This paper has proposed several key ideas. First we re-evaluate the analysis of the NMG method to show how the omissions in [16] have a noticeable impact on the convergence of the NMG method. In addition, we also provide a way of estimating the convergence rate of a solver on the coarsest grid in order to estimate the number of iterations that will be required to obtain a solution with appropriate accuracy. Secondly we propose an extension to the Chumchob-Chen NMG method which controls any folding within the deformation. Experimental results on the proposed multigrid framework demonstrate improvements in convergence and the accuracy of registrations compared with previous methods.

Keywords. Variational model, Image registration, Fast Multigrid, Mesh folding control

1 Introduction

Image registration is the process of aligning pairs, or sequences, of similar images. This alignment is achieved by fixing one image, called the reference image, and then applying geometric transformations on the remaining images, called the template images, such that the template images become similar to the reference image. This technique is a very powerful tool in many real world applications spanning diverse areas such as computer imaging, weather satellite imaging [19] and especially medical imaging which is of interest to us [4,12–14,23,24]. However, image registration is also one of the most difficult tasks of image processing with many challenges to be overcome. Generally image registration models can be classified into two main categories; parametric and non-parametric models. In parametric models, the transformations are global and can be described by matching a finite number of features in the images, leading to so-called landmark based registration [31,33], or the transformations are governed by a small number of parameters such as in the case of affine image registration [3,15] (with 6 parameters in 2D and 12 parameters in 3D). However, the focus of this paper will be on the latter category, namely non-parametric models.

Denote respectively a reference and a template image (both given as grey-scale images) $R,\ T \in \Omega \subset \mathbb{R}^d$. The aim of image registration is to transform this T to R such that they become similar to one another,

[*]Centre for Mathematical Imaging Techniques and Department of Mathematical Sciences, University of Liverpool, United Kingdom. Emails: [anthony.thompson, k.chen]@liv.ac.uk

[†]Corresponding author. Web: http://www.liv.ac.uk/∼cmchenke. The second author's work is supported by the UK EPSRC grant EP/N014499/1.

or in other words we look to find the transformation $\varphi(\boldsymbol{x}) : \mathbb{R}^d \to \mathbb{R}^d$ such that

$$T \circ \varphi(\boldsymbol{x}) = T\left(\varphi(\boldsymbol{x})\right) \approx R(\boldsymbol{x}) \text{ for } \boldsymbol{x} = (x_1, \ldots, x_d)^T \in \Omega \subset \mathbb{R}^d. \tag{1.1}$$

In variational image registration the transformation $\varphi(\boldsymbol{x})$ is equivalent to finding the displacement of every pixel $\boldsymbol{x}$ in T to their corresponding pixel in R, and so we can define $\varphi(\boldsymbol{x})$ by the following

$$\varphi \equiv \varphi(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{u}(\boldsymbol{x}) \tag{1.2}$$

where $\boldsymbol{u} \equiv \boldsymbol{u}(\boldsymbol{x}) = (u_1(\boldsymbol{x}), \ldots, u_d(\boldsymbol{x}))^T$ denotes the displacement field. Then the problem of determining φ is the same as finding $\boldsymbol{u}$. From this point onward we shall consider only the 2D case, that is $d = 2$, however all ideas presented in this paper are readily extendible to the 3D case $d = 3$. Furthermore we will also assume that the image domain Ω given by the unit square, that is $\Omega = [0,1]^2 \subset \mathbb{R}^2$. In order to determine $\boldsymbol{u}$, the variational minimisation problem will take the following form

$$\min_{\boldsymbol{u}} E(\boldsymbol{u}) = \mathscr{D}(R,T,\boldsymbol{u}) + \alpha \mathscr{R}(\boldsymbol{u}) \tag{1.3}$$

where in the energy functional $\mathscr{D}(R,T,\boldsymbol{u})$ is a distance measure, $\mathscr{R}(\boldsymbol{u})$ is the regularisation term and $\alpha \in \mathbb{R}^+$ is a weighting parameter. Note that inclusion of the regularisation term is a necessity as without it the minimisation would be ill-posed in the sense of Hadamard. For the purpose of this paper we shall consider only mono-modal images, that is images taken using the same imaging modality (e.g. CT), this means that image intensities are comparable. In the mono-modal case, the typical choice of similarity measure is the sum of squared distances (SSD) measure given by

$$\mathscr{D}(R,T,\boldsymbol{u}) = \frac{1}{2} \int_\Omega \Big(T(\boldsymbol{x}+\boldsymbol{u}) - R(\boldsymbol{x}) \Big)^2 d\Omega\ . \tag{1.4}$$

Here SSD is only one of many choices of similarity measure [34]. Moreover, the choice of regularisation term is less straightforward as there is a large selection to choose from [1, 6, 17, 18, 20–22, 34–36] and no one is yet the best. In this paper we will only consider one regularisation term, namely the diffusion regulariser and focus on optimal solution. As for numerical implementation, the common approach is to use an optimise-discretise approach, and indeed this is the approach we will adopt throughout this paper.

Solutions of variational models can be computationally intensive, but such non-parametric models are worth the effort as they can produce very accurate results and are able to deal with local deformations effectively; the high computational expense is due to the need of determining the displacement of every pixel in the image. Multigrid techniques as known fast solvers have been used in previous works [20, 21, 25, 27–29, 32, 37, 40] to greatly reduce the computational cost and produce more accurate results, however few of these directly deal with the non-linearity resulting from the similarity measure (1.4). The reason for this is that, while multigrid techniques and theories have been established for linear equations for a long time, achieving optimal convergence in a non-linear multigrid framework is never automatic and still poses a great challenge. However, the work done by Chumchob-Chen [16] introduced a robust multigrid framework for diffusion type variational models that treats the non-linearity directly. We propose to improve the convergence problems of the NMG method from [16] through a more in-depth and accurate analysis of the multigrid framework as well as using an alternate coarsest solver to obtain a more efficient solution, thus resulting in a better method. Next we address how to overcome mesh folding by incorporating an additional constraint into the diffusion model presented in [16], this idea can be thought of as a simplification of the hyper-elastic model introduced in the work by Burger et al. [11]. The addition of this constraint imposes that the transformation produced is regular and diffeomorphic i.e. there is no folding. The production of diffeomorphic transformations lead to more physically meaningful results, which is particularly useful in medical imaging. In this paper, we consider one specific (yet widely used) model, namely the diffusion model to focus on our main aims: (i) improving the convergence of the NMG method from [16] ; (ii) development of a fast NMG method for a refined diffusion model which controls folding.

There are, however, many other choices for the regularisation term [1,6,17,18,20–22,34–36], each offering a different model and with their own distinct benefits and drawbacks. In particular, we mention

Total Variation (TV) [20, 21, 35, 36]: $\mathscr{R}^{TV}(\boldsymbol{u}) = \displaystyle\sum_{s=1}^{2} \int_\Omega |\nabla u_s|\, d\Omega$ where $|\cdot|$ denotes the Euclidean norm;

Linear Elastic (LE) [1] [6] [22] [34]: $\mathscr{R}^{LE}(\boldsymbol{u}) = \int_\Omega \frac{\mu}{4} \sum_{s,t=1}^{2} \left(\partial_{x_s} u_t + \partial_{x_t} u_s\right)^2 + \frac{\lambda}{2} \left(\nabla \cdot \boldsymbol{u}\right)^2 \, d\Omega$ where μ, λ are Lamé constants;

Mean Curvature (MC) [17,18]: $\mathscr{R}^{MC}(\boldsymbol{u}) = \frac{1}{2} \int_\Omega \sum_{s=1}^{2} \nabla \cdot \left(\frac{\nabla u_s}{\sqrt{\left|\nabla u_s\right|^2 + \beta}} \right)^2 d\Omega$ where β is some small positive quantity.

While each such models might be solved by a NMG framework, achieving optimal efficiency would require further work and development.

The remainder of this paper will be set out as followed. In §2 we will introduce the formulation of the registration model focusing specifically on the diffusion model. Next in §3 we will discuss the non-linear multigrid (NMG) framework applied to the diffusion model, along with a detailed analysis to highlight how we can improve the convergence of the Chumchob-Chen NMG method. Then in §4 we will formulate our non-folding constraint model, and also present an optimisation for the implementation of the constraint. §5 will comprise of tests and comparisons with our proposed work, and finally in §6 we will present our conclusions.

2 Review of the registration model and its algorithm of [16]

The model. The diffusion regulariser is a popular choice among variational models [7–10,30], it imposes a simple smoothness constraint upon the displacement field and is given by the following

$$\mathscr{R}^{\text{Diff}}(\boldsymbol{u}) = \frac{1}{2} \int_\Omega \sum_{s=1}^{2} \left|\nabla u_s\right|^2 d\Omega \,. \tag{2.1}$$

In fact, the diffusion model is one of the few models that coincides with models from optical flow frameworks (see [8,9,30] as examples), which is particularly useful when registering sequences of images. The diffusion model is given by the following minimisation problem

$$\min_{\boldsymbol{u}} E^{\text{Diff}}(\boldsymbol{u}) = \mathscr{D}(R, T_{\boldsymbol{u}}) + \alpha \mathscr{R}^{\text{Diff}}(\boldsymbol{u}) = \frac{1}{2} \int_\Omega (T_{\boldsymbol{u}} - R)^2 + \alpha \sum_{s=1}^{2} \left|\nabla u_s\right|^2 d\Omega \tag{2.2}$$

where $T_{\boldsymbol{u}} \equiv T(\boldsymbol{x} + \boldsymbol{u})$ and $R \equiv R(\boldsymbol{x})$. The corresponding Euler-Lagrange (EL) equations are derived from the following limits

$$\lim_{\varepsilon_1 \to 0} \frac{E^{\text{Diff}}(u_1 + \varepsilon_1 \phi_1, u_2) - E^{\text{Diff}}(u_1, u_2)}{\varepsilon_1} = 0, \; \lim_{\varepsilon_2 \to 0} \frac{E^{\text{Diff}}(u_1, u_2 + \varepsilon_2 \phi_2) - E^{\text{Diff}}(u_1, u_2)}{\varepsilon_2} = 0 \tag{2.3}$$

which eventually result in the following integrals

$$\int_\Omega \phi_m \left[\partial_{u_m} T_{\boldsymbol{u}} \left(T_{\boldsymbol{u}} - R\right) - \alpha \Delta u_m \right] d\Omega + \alpha \int_{\partial\Omega} \phi_m \left(\nabla u_m \cdot \boldsymbol{n}\right) dS = 0 \tag{2.4}$$

and thus, after the use of the fundamental lemma of calculus of variations, yield the EL equations

$$-\alpha \Delta u_m + F_m(\boldsymbol{u}) = 0 \tag{2.5}$$

with Neumann boundary conditions $\nabla u_m \cdot \boldsymbol{n} = 0$ where $\boldsymbol{n}$ denotes the outward unit normal and

$$F_m(\boldsymbol{u}) = \partial_{u_m} T_{\boldsymbol{u}} \left(T_{\boldsymbol{u}} - R\right) \tag{2.6}$$

denote the force terms, for $m = 1, 2$.

2.1 Optimise-discretise approach for diffusion model

We consider a numerical approximation to the EL equations (2.5) by discretising the image domain Ω into a uniform $n \times n$ mesh with interval width h, using a finite difference (FD) method. The size of the mesh is chosen to be equal to the dimension of the image (e.g. 512×512 to coincide with resolution of given images) and in general need not be square, however in this paper we consider square images as this is common for medical image slices. Using the following central FD approximations

$$(\partial_{u_1} T_{\boldsymbol{u}})_{i,j} \approx \frac{1}{2h}\Big((T_{\boldsymbol{u}})_{i+1,j} - (T_{\boldsymbol{u}})_{i-1,j}\Big),\ (\partial_{u_2} T_{\boldsymbol{u}})_{i,j} \approx \frac{1}{2h}\Big((T_{\boldsymbol{u}})_{i,j+1} - (T_{\boldsymbol{u}})_{i,j-1}\Big)$$

$$(\Delta u_m)_{i,j} \approx \frac{1}{h^2}\Big((u_m)_{i,j-1} + (u_m)_{i-1,j} - 4(u_m)_{i,j} + (u_m)_{i+1,j} + (u_m)_{i,j+1}\Big) \tag{2.7}$$

at a general discrete point (i,j), leads to the following discrete versions of the EL equations (2.5)

$$-\alpha\,(\Delta u_m)_{i,j} + (F_m(\boldsymbol{u}))_{i,j} = 0 \tag{2.8}$$

with

$$(F_m(\boldsymbol{u}))_{i,j} = (\partial_{u_m} T_{\boldsymbol{u}})_{i,j}\Big((T_{\boldsymbol{u}})_{i,j} - (R)_{i,j}\Big) \tag{2.9}$$

for $m = 1,2$ and $i,\ j = 2,\ldots,n-1$.

2.2 The collective pointwise smoother

The term smoother, which stems from multigrid theory, is nothing but an iterative solver. In [16] the lexicographic Gauss-Seidel (GS-LEX) method was employed to solve the linear part of the system (2.8) through an inner iteration loop, and a fixed point iteration scheme to solve the non-linear part through an outer iteration loop. In a lexicographical ordering system, a general discrete point (i,j) as in (2.9) is linked to the global index $k = (j-2)(n-1) + (i-1)$, with n the size of the discrete image dimensions; then for $m = 1,2$, we get

$$-\alpha\,(\Delta u_m)_k + (F_m(\boldsymbol{u}))_k = 0 \tag{2.10}$$

as illustrated in Figure 1. Now to solve the non-linear part of this system, we employ the following semi-implicit fixed point iteration scheme

$$-\alpha\,(\Delta u_m)_k^{(l+1)} + (F_m(\boldsymbol{u}))_k^{(l+1)} = 0 \tag{2.11}$$

where

$$(F_1(\boldsymbol{u}))_k^{(l+1)} = \Big(\partial_{u_1} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k \Big((T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)}))_k - (R(x_1,x_2))_k\Big)$$

$$(F_2(\boldsymbol{u}))_k^{(l+1)} = \Big(\partial_{u_2} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k \Big((T(x_1 + u_1^{(l)}, x_2 + u_2^{(l+1)}))_k - (R(x_1,x_2))_k\Big). \tag{2.12}$$

The key question addressed in [16] was how to treat the non-linear terms $(T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)}))_k$, $(T(x_1 + u_1^{(l)}, x_2 + u_2^{(l+1)}))_k$ in a GS-LEX scheme. It proposed to use the first order approximations:

$$\begin{aligned}
\Big(T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)})\Big)_k &\approx \Big(T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k \\
&\quad + \Big((u_1)_k^{(l+1)} - (u_1)_k^{(l)}\Big)\Big(\partial_{u_1} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k \\
\Big(T(x_1 + u_1^{(l)}, x_2 + u_2^{(l+1)})\Big)_k &\approx \Big(T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k \\
&\quad + \Big((u_2)_k^{(l+1)} - (u_2)_k^{(l)}\Big)\Big(\partial_{u_2} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)})\Big)_k
\end{aligned}$$

which are substituted back into the discrete force terms (2.10) leading to the following discrete system

$$-\alpha\left(\Delta u_m\right)_k^{(l+1)} + \left(\partial_{u_m} T_{\boldsymbol{u}}\right)_k^{(l)}\left(\left(T_{\boldsymbol{u}}\right)_k^{(l)} + \left(\left(u_m\right)_k^{(l+1)} - \left(u_m\right)_k^{(l)}\right)\left(\partial_{u_m} T_{\boldsymbol{u}}\right)_k^{(l)} - \left(R\right)_{\boldsymbol{k}}\right) = 0 \qquad (2.13)$$

with $(T_{\boldsymbol{u}})_k^{(l)} \equiv \left(T(\boldsymbol{x}+\boldsymbol{u}^{(l)})\right)_{\boldsymbol{k}}$ *etc.* for $m = 1, 2$. Using the FD approximations (2.7), we can write (2.13) in the following way

$$\begin{aligned}
&-\frac{\alpha}{h^2}\left(\left(u_m\right)_{k-n}^{(l+1)} + \left(u_m\right)_{k-1}^{(l+1)}\right) + \left(\left(\left(\partial_{u_m} T_{\boldsymbol{u}}\right)^2\right)_k^{(l)} + \frac{4\alpha}{h^2}\right)\left(u_m\right)_k^{(l+1)} \\
&-\frac{\alpha}{h^2}\left(\left(u_m\right)_{k+1}^{(l+1)} + \left(u_m\right)_{k+n}^{(l+1)}\right) = \left(\left(\partial_{u_m} T_{\boldsymbol{u}}\right)^2\right)_k^{(l)}\left(u_m\right)_k^{(l)} - \left(\partial_{u_m} T_{\boldsymbol{u}}\right)_k^{(l)}\left(\left(T_{\boldsymbol{u}}\right)_k^{(l)} - \left(R\right)_{\boldsymbol{k}}\right)
\end{aligned} \qquad (2.14)$$

for $m = 1, 2$. Then to compute the $(l+1)$ updates in (2.14), we use a GS-LEX based method.

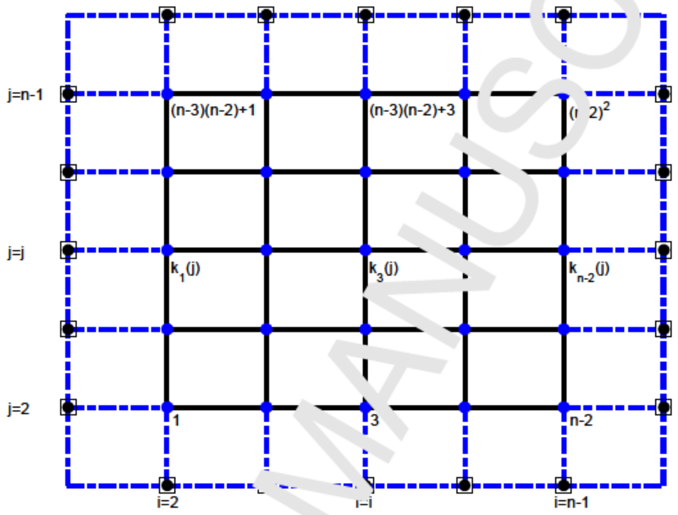


Figure 1: Illustration of how the domain Ω is discretised by $n \times n$ grid points. The dashed blue line represents the boundary $\partial\Omega$ of the discrete domain, with the boxed points representing the used boundary points, and the black lines show the $(n-2) \times (n-2)$ grid corresponding to the blue interior points. The indexing on the interior points show how the global index k is ordered lexicographically.

However, such an iterative method is not effective as a standalone solver since solving the discrete system of PDEs (2.10) pixel-wise can lead to a very high computational cost, especially for big images. This fact is well-known for simpler PDEs such as the Poisson equation (corresponding to $F_m = 0$ and $h \to 0$). One natural way of reducing the cost of calculating the displacement field is a NMG method in which this (slow) iterative method is used as a smoother.

There has already been a lot of work regarding the implementation of NMG methods [21,25,27,28,32] for related models, each having its own unigrid iterative solver, however most of these works do not address the non-linearity in the similarity measure directly, instead linear diagonal terms or augmented systems are used. Chumchob-Chen [16] proposed a robust solver which does directly deal with this non-linearity arising from the SSD term, however an inaccurate analysis of the NMG method lead to a less than optimal convergence factor for the NMG method which we will demonstrate in the next section.

2.3 The NMG method

There are two theoretical principles driving multigrid methods for linear PDEs. The first is that, although standard iterative methods such as the Jacobi and GS methods have poor convergence rates when used independently, they are effective at smoothing out any high frequency error components within a small number of iterations. This property leads to the second key principle of multigrid methods, namely low frequency error components can be well approximated on a coarser grid. Naturally an approximate and accurate solution on a coarser grid can then be interpolated back to the fine grid to approximate the original problem; this two-grid approach is significantly cheaper than working solely on the fine grid.

In fact this strategy allows us to obtain a more accurate approximation efficiently as we can perform a larger number of iterations on the coarser grid in less time when compared with iterating the fine grid alone. This fine-coarse-fine strategy, known as the two-grid V-cycle (see [5] for details), can however be repeated on the coarse grid to interact with even coarser grids until some coarsest grid with few points.

While multigrid frameworks are known, and indeed very easy to implement for linear cases, problems like (2.5) which are highly non-linear prove significantly more difficult to develop a converging NMG method. Now we present the FAS-NMG algorithm of [16] for (2.10) before we highlight the omissions in the analysis which resulted in an overestimated smoothing rate (thus leading to a less optimal NMG method with slower convergence rate), and include our more accurate analysis to overcome this problem. Here FAS stands for "full approximation scheme" by A. Brandt for solving a non-linear operator equation. First consider a two grid setting where Ω^h denotes a fine grid and Ω^H a coarse grid with $h = \frac{1}{n-1}$, $H = 2h$. Also denote the system (2.10) by the operator notation on Ω^h

$$\mathcal{N}^h(\boldsymbol{u}^h) = \mathcal{G}^h \tag{2.15}$$

with

$$\mathcal{N}^h = \begin{pmatrix} (\mathcal{N}_1^h)_k \\ (\mathcal{N}_2^h)_k \end{pmatrix}, \boldsymbol{u}^h = \begin{pmatrix} (u_1^h)_k \\ (u_2^h)_k \end{pmatrix}, \mathcal{G}^h = \begin{pmatrix} (g_1^h)_k \\ (g_2^h)_k \end{pmatrix}. \tag{2.16}$$

and where $(\mathcal{N}_1^h)_k = \left(F_1(\boldsymbol{u}^h)\right)_k - \alpha\left(\Delta^h u_1^h\right)_k$, $(\mathcal{N}_2^h)_k = \left(F_2(\boldsymbol{u}^h)\right)_k - \alpha\left(\Delta^h u_2^h\right)_k$, $(g_1^h)_k = (g_2^h)_k = 0$, $k = 1, 2, \dots, (n-2)^2$. The main steps of the FAS-NMG are as follows.

Smoothing step. Apply the iterative method (2.14) on grid Ω^h starting from some initial guess. This is the pre-smoothing step required to obtain a smooth approximation $\bar{\boldsymbol{u}}^h = (\bar{u}_1^h, \bar{u}_2^h)^T$ which has residual $\boldsymbol{r}^h = \mathcal{G}^h - \mathcal{N}^h(\bar{\boldsymbol{u}}^h)$.

To improve this smooth approximation, it remains to compute the algebraic error (or the residual correction) $\boldsymbol{e}^h = (e_1^h, e_2^h)^T = \boldsymbol{u}^h - \bar{\boldsymbol{u}}^h$ which cannot be computed directly on Ω^h.

Restriction. Since only smooth errors can be well approximated on a coarser grid, we first solve the FAS coarse grid residual equation

$$\mathcal{N}^H(\boldsymbol{u}^H) \equiv \mathcal{N}^H(\bar{\boldsymbol{u}}^H + \boldsymbol{e}^H) = \boldsymbol{r}^H + \mathcal{N}^H(\bar{\boldsymbol{u}}^H) \equiv \mathcal{G}^H \tag{2.17}$$

where $\bar{\boldsymbol{u}}^H = \mathcal{R}_h^H \bar{\boldsymbol{u}}^h$, $\boldsymbol{e}^H = \mathcal{R}_h^H \boldsymbol{e}^h$, $\boldsymbol{r}^H = \mathcal{R}_h^H \boldsymbol{r}^h$ and $\mathcal{R}_h^H$ is the restriction operator, which we take to be the full-weighted restriction operator defined by the following stencil

$$\mathcal{R}_h^H = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h^H \tag{2.18}$$

Coarse grid solution. For a two-grid method (or in a multigrid setting where Ω^H is the coarsest level and computations are inexpensive), the above coarse grid equation must be solved accurately to obtain solutions $\boldsymbol{u}^H$. Based on this $\boldsymbol{u}^H$ and its initial guess $\bar{\boldsymbol{u}}^H$, we obtain the residual correction

$$\boldsymbol{e}^H = \boldsymbol{u}^H - \bar{\boldsymbol{u}}^H. \tag{2.19}$$

Interpolation. Now we wish to use (2.19) to correct the approximations on the finer grid Ω^h; we do this by interpolating the corrections using bilinear interpolation. That is we compute

$$\boldsymbol{e}^h = \mathcal{I}_H^h \boldsymbol{e}^H, \quad \mathcal{I}_H^h = \frac{1}{4}\left]\begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}\right[_H^h. \tag{2.20}$$

Once the corrections have been interpolated to the next fine grid level, we use them to update the current grid level approximations via $\boldsymbol{u}^h = \bar{\boldsymbol{u}}^h + \boldsymbol{e}^h$. After the approximations have been corrected, we use a post-smoothing step to remove any interpolation errors. This process of interpolation, correction and smoothing is repeated until the approximations on the original grid level have been corrected and smoothed, thus resulting in our final solutions $\boldsymbol{u}^h$.

Remark 2.1. *According to the work done in [26], there are three conditions which need to be satisfied regarding the orders of the restriction and interpolation methods for a convergent NMG. For an order M PDE, we require*

$$(i)\ m_R + m_I \geq M; \quad (ii)\ m_I \geq M \text{ and } m_R \geq 0; \quad (iii)\ m_R \geq M \text{ and } m_I \geq 0$$

where m_R, m_I denote the high frequency orders of the restriction and interpolation schemes respectively. In our case we have $m_R = 2$, $m_I = 2$, for the full-weighted restriction and bilinear interpolation operators respectively, and so all three conditions are satisfied.

Below the FAS-NMG algorithm has been summarised

Algorithm 1 $u_h^{(k+1)} \quad FASNMG(R^h, T^h, n, h, level, u_h^{(k)}, \mathcal{G}^h, \alpha, \nu_1, \nu_2)$

```
1: Pre-smoothing step by performing $\nu_1$ steps (relaxation sweeps)   $\bar{u}_h^{(k)} \quad Smooth(R^h, T^h, u_h^{(k)}, \mathcal{G}^h, \alpha, \nu_1)$
2: Coarse-grid correction
       Compute the residual $r_h^{(k)} = \mathcal{G}^h - \mathcal{N}^h(u_h^{(k)})$
       Restrict residual and smooth approximations $r_H^{(k)} = \mathcal{R}_h^H r_h^{(k)}$, $\bar{u}_H^{(k)} = \mathcal{R}_h^H \bar{u}_h^{(k)}$
       Set $level \to level - 1$, $H = 2h$, $nc = \frac{n}{2}$
       Form RHS of coarse grid PDEs $\mathcal{G}^H = r^H + \mathcal{N}^H(\bar{u}_H^{(k)})$
       Solve residual equation on coarse grid to obtain approximation $\tilde{u}_H^{(k)}$
3: if $level = 1$ then
       Solve to obtain solutions $u_H^{(k)}$ to high accuracy using a coarsest grid solver.
4: else $level > 1$   Repeat the FAS-NMG procedure recursively to the next level i.e.
       $\bar{u}_H^{(k)} \quad FASNMG(R^H, T^H, nc, H, level - 1, \tilde{u}_H^{(k)}, \mathcal{G}^H, \alpha, \nu_1, \nu_2)$
5: end if
       Compute the correction $e_H^{(k)} = u_H^{(k)} - \bar{u}_H^{(k)}$
       Interpolate the correction to next fine grid level $e_h^{(k)} = \mathcal{I}_H^h e_H^{(k)}$
       Update current grid level approximations using correction $\hat{u}_h^{(k)} = \bar{u}_h^{(k)} + e_h^{(k)}$
6: Post-smoothing step by performing $\nu_2$ steps (relaxation sweeps)   $u_h^{(k+1)} \quad Smooth(R^h, T^h, \hat{u}_h^{(k)}, \mathcal{G}^h, \alpha, \nu_2)$
       Computes $u_h^{(k+1)}$ by performing $\nu_2$ relaxation sweeps of a smoother.
```

In [16], the coarsest solver that was adopted was an additive operator splitting (AOS) method. For the diffusion model, it takes the following form $u_m^{(k+1)} = \frac{1}{2}\sum_{s=1}^{2} \left[I - 2\tau\alpha L_{x_s}\right]^{-1}\left(u_m^{(k+1)} + \tau g_m - \tau F_m(\boldsymbol{u})\right)$ where I denotes the identity operator, $\tau > 0$ the time-step, g_m the RHS coming from the NMG framework, $F_m(\boldsymbol{u})$ the force terms given in (2.[illegible]) for $m = 1, 2$ and $L_{x_s} = \partial_{x_s x_s}$ denote the parts of the discrete Laplace operator in the x_s directions for $s = 1, 2$ respectively. The above equations are updated along the x_1, x_2 directions separately, thus leading to the system

$$\begin{cases} \left[I - 2\tau\alpha L_{x_1}\right] u_{m,p_1}^{(k+\frac{1}{2})} = u_m^{(k)} + \tau g_m - \tau F_m(\boldsymbol{u}), \\ \left[I - 2\tau\alpha L_{x_2}\right] u_{m,p_2}^{(k+\frac{1}{2})} = u_m^{(k)} + \tau g_m - \tau F_m(\boldsymbol{u}) \end{cases} \tag{2.21}$$

with the updates $u_m^{(k+1)} = \frac{1}{2}\left(u_{m,p_1}^{(k+\frac{1}{2})} + u_{m,p_2}^{(k+\frac{1}{2})}\right)$ for $m = 1, 2$.

Remark 2.2. *In [16], the h-ellipticity for the proposed smoother was computed in order to check whether the smoother was suitable for use in the NMG method. From the resulting calculation, the h-ellipticity was found to have a value of $\frac{1}{16}$, and it was concluded that the smoother was suitable for use in the NMG method. By performing the same calculation for our proposed smoother in §2.2, which is similar to the one used in [16], we also obtained a value of $\frac{1}{16}$ and thus reached the same conclusion.*

3 An improved analysis of the NMG algorithm of [16]

As mentioned, the above Algorithm 1 as implemented by Chumchob-Chen [16] could still be slow to converge to a solution from new experiments. We found that a major part of this convergence problem was a result of an inaccurate analysis of the smoothing rate, which lead to an overestimation of the rate. By re-evaluating the analysis of the NMG method, as well as building in some new components, lead to our NMG algorithm with a vastly improved convergence rate.

In this section we will outline our more detailed and accurate analysis of the NMG framework. We do this by analysing two key components of the NMG algorithm (namely the smoothing rate of the smoother and the coarsest grid solver), which leads to an optimal NMG method.

3.1 Smoother analysis using Local Fourier Analysis (LFA)

We begin our analysis of the NMG method by showing an improved, and more accurate, LFA of the smoother scheme that was described in [16]. A discrete error (e.g. residual) function on a grid can be written as a sum of two terms:

- high frequency error components (are not visible if the problem is restricted to a coarser grid);
- low frequency error components (that can be accurately represented on a coarser grid).

The sole purpose of the smoother, within a MG framework, is to remove any high frequency error components. Local Fourier Analysis (LFA) is used to measure how effective a given smoother scheme is.

Although LFA was originally designed to analyse discrete linear operator equations, it was extended by A. Brandt (see [38]) to study non-linear operators via a 'freezing' of localised coefficients. To start we first assume that we are working on an infinite grid, this then allows us to remove any influence from the boundary conditions. Next we assume that the discrete form of a non-linear operator, with variable coefficients, can be replaced locally by an operator with constant coefficients and extended to the infinite grid. We need to ensure all high frequency error components are removed prior to restriction to a coarse grid. As a result it is imperative that we know how effective our relaxation scheme is at smoothing out the errors so we can adjust the number of sweeps required for the pre- and post-smoothing steps. Using LFA we obtain a value μ which is defined to be the smoothing factor for a given relaxation scheme.

LFA for pointwise smoother from [16]. While the smoother we described in §2.2 is similar to the one used in [16], we found that the smoother analysis in [16] contained an omission which lead to a very over-optimistic smoothing rate (practically to a slow convergence if using it as a guide). In [16], the discrete system (2.10) was written in the following way

$$\mathcal{N}_+^h \boldsymbol{u}_{new}^h + \mathcal{N}_0^h \boldsymbol{u}_{new}^h + \mathcal{N}_-^h \boldsymbol{u}_{old}^h = \mathcal{G}^h \tag{3.1}$$

where $\boldsymbol{u}_{new}^h$, $\boldsymbol{u}_{old}^h$ denote the current and previous approximations of $\boldsymbol{u}^h$ respectively, and

$$\mathcal{N}_+^h = \begin{pmatrix} -\alpha\mathscr{L}_+^h & 0 \\ 0 & -\alpha\mathscr{L}_+^h \end{pmatrix}, \mathcal{N}_0^h = \begin{pmatrix} -\alpha\mathscr{L}_0^h + \sigma_{11}^h & \sigma_{12}^h \\ \sigma_{12}^h & -\alpha\mathscr{L}_0^h + \sigma_{22}^h \end{pmatrix}$$

$$\mathcal{N}_-^h = \begin{pmatrix} -\alpha\mathscr{L}_-^h & 0 \\ 0 & -\alpha\mathscr{L}_-^h \end{pmatrix}, \mathcal{G}^h = \begin{pmatrix} g_1^h - F_1^h \\ g_2^h - F_2^h \end{pmatrix} \tag{3.2}$$

with $\sigma_{pq}^h = \partial_{u_p} T_{\boldsymbol{u}}^h \partial_{u_q} T_{\boldsymbol{u}}^h$, g_m^h denote the RHS coming from the NMG scheme, F_m^h are the discrete force terms as given in (2.9) and where $\mathscr{L}_+^h$, $\mathscr{L}_0^h$, $\mathscr{L}_-^h$ define the following stencils

$$\mathscr{L}_+^h = \frac{1}{h^2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \mathscr{L}_0^h = \frac{1}{h^2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \mathscr{L}_-^h = \frac{1}{h^2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}. \tag{3.3}$$

for p, q, $m = 1, 2$. The smoothing rate in [16] was then calculated on a 32×32 grid after a total of 5 outer and 5 inner iteration loops had been performed, thus resulting in an average smoothing rate of $\mu_{avg} \approx 0.5$ when taking $\alpha = \frac{1}{10}$. However, in the analysis of [16] we notice that the $(u_m)_k^{(l)}$ terms, which result from the linearisation of the SSD term, where not included in the smoothing rate calculation. This omission means that the obtained rate of 0.5 was a vast overestimation of the actual smoothing rate, and as a result this lead to an underestimation of the number of pre-smoothing steps required before restriction. This means that when we restrict the problem to a coarser grid, there are still high frequency error components on the fine grid which have not been removed, and so the coarse grid correction that we obtain is much less accurate thus leading to more NMG cycles being required to reach an accurate solution. This omission, as we will now show, has a noticeable effect on the smoothing rate.

Revised LFA for pointwise smoother from §2.2. Here we will repeat the analysis of the smoothing rate, with the $(u_m)_k^{(l)}$ terms included, in order to illustrate the impact the addition of these terms have on the smoothing rate. We begin by writing the discrete equations (2.10) in the following form

$$\mathcal{N}^h \boldsymbol{u}^h + \mathcal{M}^h \boldsymbol{u}^h = \mathcal{G}^h \tag{3.4}$$

where $\mathcal{G}^h$ is as in (3.2), and

$$\mathcal{N}^h = \begin{pmatrix} -\alpha\Delta^h + \sigma_{11}^h & 0 \\ 0 & -\alpha\Delta^h + \sigma_{22}^h \end{pmatrix}, \mathcal{M}^h = \begin{pmatrix} -\sigma_{11}^h & 0 \\ 0 & -\sigma_{22}^h \end{pmatrix} \tag{3.5}$$

using the following representation of the discrete Laplace operator $\Delta^h \equiv \mathscr{L}_+^h + \mathscr{L}_0^h + \mathscr{L}_-^h$, with $\mathscr{L}_+^h$, $\mathscr{L}_0^h$, $\mathscr{L}_-^h$ as defined in (3.3), then we can express (3.4) in the following way

$$\mathcal{N}_+^h \boldsymbol{u}_{new}^h + \mathcal{N}_0^h \boldsymbol{u}_{new}^h + \mathcal{N}_-^h \boldsymbol{u}_{old}^h + \mathcal{M}^h \boldsymbol{u}_{ol}^h = \mathcal{G}^h \tag{3.6}$$

and subtracting (3.6) from (3.4) yields the local error equation given by

$$[\mathcal{N}_+^h + \mathcal{N}_0^h] \boldsymbol{e}_{new}^h = -[\mathcal{N}_-^h + \mathcal{M}^h] \boldsymbol{e}_{old}^h \tag{3.7}$$

where $\mathcal{N}_+^h$, $\mathcal{N}_0^h$, $\mathcal{N}_-^h$ are as defined in (3.2) and

$$\boldsymbol{e}_{new}^h = \left(e_{1\,new}^h, e_{2\,new}^h\right)^T, \boldsymbol{e}_{old}^h = \left(e_{1\,old}^h, e_{2\,old}^h\right)^T. \tag{3.8}$$

Using Fourier components, we can rewrite (3.7) in the following way

$$[\hat{\mathcal{N}}_+^h(\boldsymbol{\theta}) + \hat{\mathcal{N}}_0^h(\boldsymbol{\theta})] \psi_{\boldsymbol{\theta}}^{new} \exp\left(\frac{2i\theta_1 i\pi}{n} + \frac{2i\theta_2 j\pi}{n}\right) = -[\hat{\mathcal{N}}_-^h(\boldsymbol{\theta}) + \hat{\mathcal{M}}^h(\boldsymbol{\theta})] \psi_{\boldsymbol{\theta}}^{old} \exp\left(\frac{2i\theta_1 i\pi}{n} + \frac{2i\theta_2 j\pi}{n}\right) \tag{3.9}$$

where $\boldsymbol{i} = \sqrt{-1}$, $\boldsymbol{\theta} \in \boldsymbol{\Theta} = [-\pi, \pi)^2$ and $\psi_{\boldsymbol{\theta}}^*$ are Fourier coefficients. From here we determine the local smoothing rate μ_{loc} using the following

$$\mu_{\max} = \max_{\text{loc}} \mu_{loc}, \qquad \mu_{loc} \equiv \mu_{loc}(\boldsymbol{\theta}) = \sup\left\{\rho(\hat{\boldsymbol{S}}^h(\boldsymbol{\theta})) | \boldsymbol{\theta} \in \boldsymbol{\Theta}_{high}\right\} \tag{3.10}$$

where $\boldsymbol{\Theta}_{high} = \Theta \setminus \left[-\frac{\pi}{2}, \frac{\pi}{2}\right)^2$, $\rho(\cdot)$ denotes the spectral radius, and the amplification matrix $\hat{\boldsymbol{S}}^h(\boldsymbol{\theta})$ is given by

$$\hat{\boldsymbol{S}}^h(\boldsymbol{\theta}) = -[\hat{\mathcal{N}}_+^h(\boldsymbol{\theta}) + \hat{\mathcal{N}}_0^h(\boldsymbol{\theta})]^{-1} [\hat{\mathcal{N}}_-^h(\boldsymbol{\theta}) + \hat{\mathcal{M}}^h(\boldsymbol{\theta})] \tag{3.11}$$

with

$$\hat{\mathcal{N}}_+^h(\boldsymbol{\theta}) = \begin{pmatrix} -\frac{\alpha}{h^2}\left(e^{-\boldsymbol{i}\omega_1} + e^{-\boldsymbol{i}\omega_2}\right) & 0 \\ 0 & -\frac{\alpha}{h^2}\left(e^{-\boldsymbol{i}\omega_1} + e^{-\boldsymbol{i}\omega_2}\right) \end{pmatrix}, \hat{\mathcal{N}}_0^h(\boldsymbol{\theta}) = \begin{pmatrix} \frac{4\alpha}{h^2} + \sigma_{11}^h & 0 \\ 0 & \frac{4\alpha}{h^2} + \sigma_{22}^h \end{pmatrix},$$

$$\hat{\mathcal{N}}_-^h(\boldsymbol{\theta}) = \begin{pmatrix} -\frac{\alpha}{h^2}\left(e^{\boldsymbol{i}\omega_1} + e^{\boldsymbol{i}\omega_2}\right) & 0 \\ 0 & -\frac{\alpha}{h^2}\left(e^{\boldsymbol{i}\omega_1} + e^{\boldsymbol{i}\omega_2}\right) \end{pmatrix}, \hat{\mathcal{M}}^h(\boldsymbol{\theta}) = \begin{pmatrix} -\sigma_{11}^h & 0 \\ 0 & -\sigma_{22}^h \end{pmatrix} \tag{3.12}$$

where $\omega_m = \frac{2\theta_m\pi}{n}$ for $m = 1, 2$. Implementing the revised local smoothing rate formulae, under the same conditions that were used in [16], we obtained an average and maximum smoothing rate of $\mu_{avg} \approx 0.69854$ and $\mu_{max} \approx 0.74762$ respectively. By the smoothing rate of 0.5 in [16] within each outer iteration, 5 inner iterations would result in reduction of the error by 0.0313 which appeared satisfactory. However 5 inner iterations would reduce only by 0.17 and 0.23 respectively using our new smoothing rates μ_{avg} and μ_{max}. In order to reduce to the level of error claimed in [16], we estimate that we would require up to 12 inner iterations. So we see that the original analysis in [16] resulted in the estimated number of pre-smoothing steps being roughly half of the number of steps that would actually be required to reduce the error to quoted level.

3.2 Convergence analysis of two coarsest grid solvers by LFA

Next we give a simple solution to the challenging problem of getting the convergence rate of a non-linear iterative method. Here we remark that this analysis was not performed in [16]. Consequently, we can compare methods and guide the number of iterations to be prescribed on the coarsest grid. Recall that the AOS solver (2.21) was used by Chumchob-Chen [16]. Here we shall propose to use a fixed point type solver on the coarsest grid instead.

Our coarsest grid solver. From §2.2 we have the following lexicographically ordered discrete system of linear equations

$$\begin{aligned} &-\frac{\alpha}{H^2}\left((u_m)_{k-n}^{(l+1)}+(u_m)_{k-1}^{(l+1)}\right)+\left(\left((\partial_{u_m}T_{\boldsymbol{u}})^2\right)_k^{(l)}+\frac{4\alpha}{H^2}\right)(u_m)_k^{(l+1)} \\ &-\frac{\alpha}{H^2}\left((u_m)_{k+1}^{(l+1)}+(u_m)_{k+n}^{(l+1)}\right)=\left((\partial_{u_m}T_{\boldsymbol{u}})^2\right)_k^{(l)}(u_m)_k^{(l)}-(\partial_{u_m}T_{\boldsymbol{u}})_k^{(l)}\left((T_{\boldsymbol{u}})_k^{(l)}-(R)_k\right) \end{aligned} \tag{3.13}$$

for $m=1,2$. In matrix notation, we can express these equations as matrix equations $\boldsymbol{A}_m\boldsymbol{u}_m=\boldsymbol{f}_m$, where $\boldsymbol{u}_m$, $\boldsymbol{f}_m\in\mathbb{R}^{(n-2)^2\times 1}$ are column vectors and $\boldsymbol{A}_m\in\mathbb{R}^{(n-2)^2\times(n-2)^2}$ are the block tridiagonal system matrices with the following structure

$$\boldsymbol{A}_m=\begin{pmatrix} A_{m_2} & I_1 & & \\ I_1 & \ddots & \ddots & \\ & \ddots & \ddots & I_1 \\ & & I_1 & A_{m_{n-1}} \end{pmatrix},\ \boldsymbol{u}_m=\begin{pmatrix} (u_m)_{k_2(2)} \\ \vdots \\ (u_m)_{k_i(j)} \\ \vdots \\ (u_m)_{k_{n-2}(n-2)} \end{pmatrix},\ \boldsymbol{f}_m=\begin{pmatrix} (f_m)_{k_2(2)} \\ \vdots \\ (f_m)_{k_i(j)} \\ \vdots \\ (f_m)_{k_{n-2}(n-2)} \end{pmatrix} \tag{3.14}$$

where A_{m_j}, $I_1\in\mathbb{R}^{(n-2)\times(n-2)}$ are matrices with structure

$$A_{m_j}=\begin{pmatrix} (a_m)_{k_2(j)} & -\frac{\alpha}{H^2} & & \\ -\frac{\alpha}{H^2} & \ddots & \ddots & \\ & \ddots & \ddots & -\frac{\alpha}{H^2} \\ & & -\frac{\alpha}{H^2} & (a_m)_{k_{n-1}(j)} \end{pmatrix},\ I_1=-\frac{\alpha}{H^2}\begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{pmatrix} \tag{3.15}$$

with $(a_m)_{k_i(j)}=\left((\partial_{u_m}T_{\boldsymbol{u}})^2\right)_{k_i(j)}+\frac{4\alpha}{H^2}$ and where $k_i(j)=(j-2)(n-1)+(i-1)$ denotes a general lexicographically ordered discrete point (i,j), as shown in Figure 1. Also

$$(f_m)_{k_i(j)}=\left((\partial_{u_m}T_{\boldsymbol{u}})^2\right)_{k_i(j)}(u_m)_{k_i(j)}-(\partial_{u_m}T_{\boldsymbol{u}})_{k_i(j)}\left((T_{\boldsymbol{u}})_{k_i(j)}-(R)_{k_i(j)}\right) \tag{3.16}$$

for $m=1,2$ and $i,\ j=2,\ldots,n-1$. Then our proposed algorithm is as shown in Algorithm 2

In order to demonstrate the improvement in convergence rate of our proposed coarsest grid solver over the AOS scheme used in [16] we first need a way to measure the convergence rate. To do this we shall employ LFA to estimate the convergence rates of both of our proposed solver and the AOS solver. The purpose is to discriminate these two estimations. Unfortunately due to the non-linearity of the problem we are unable to obtain a sharp measure of the convergence rate, and so using LFA to obtain an approximation is the best option. It should be remarked that LFA used for this convergence analysis is only viable on a coarse grid (e.g. 8×8 mesh) as the rate is not sharp especially on a fine grid (e.g. 128×128 mesh).

Analysis of the proposed coarsest grid solver. To estimate the convergence rate $\mathcal{P}$ of a given solver, we follow a similar method to that in the smoother analysis shown in §3.1. That is we must evaluate the amplification matrix $\hat{\boldsymbol{S}}^H(\boldsymbol{\theta})$ at every discrete interior point (i,j) for $i,\ j=2,\ldots,n-1$ and where n denotes the size of the image dimensions. However, where we restricted $\boldsymbol{\theta}$ to only consider the high frequency range $\boldsymbol{\Theta}_{high}$ in the smoother analysis, now we consider $\boldsymbol{\theta}$ over the entire Fourier domain $\boldsymbol{\Theta}$. Since our proposed direct solver is based upon the pointwise smoother shown in §2.2, the derivation of the amplification matrix $\hat{\boldsymbol{S}}^H(\boldsymbol{\theta})$ is very similar to that shown in §3.1. Then, the convergence rate for our proposed direct solver can be estimated locally by the following

$$\mathcal{P}_{D\,\max}=\max_{\text{loc}}\mathcal{P}_{D\,loc},\qquad \mathcal{P}_{D\,loc}\equiv\mathcal{P}_{D\,loc}(\boldsymbol{\theta})=\sup\left\{\rho(\hat{\boldsymbol{S}}^H(\boldsymbol{\theta}))\middle|\boldsymbol{\theta}\in\boldsymbol{\Theta}\right\} \tag{3.17}$$

where $\boldsymbol{\Theta} \in [-\pi, \pi)^2$, $\rho(\cdot)$ denotes the spectral radius and $\hat{\boldsymbol{S}}^H(\boldsymbol{\theta})$ is the amplification matrix as given by

$$\hat{\boldsymbol{S}}^H(\boldsymbol{\theta}) = -\left[\hat{\mathcal{N}}_+^H(\boldsymbol{\theta}) + \hat{\mathcal{N}}_0^H(\boldsymbol{\theta}) + \hat{\mathcal{N}}_-^H(\boldsymbol{\theta})\right]^{-1} \hat{\mathcal{M}}^H(\boldsymbol{\theta})$$

with $\hat{\mathcal{N}}_+^H(\boldsymbol{\theta})$, $\hat{\mathcal{N}}_0^H(\boldsymbol{\theta})$, $\hat{\mathcal{N}}_-^H(\boldsymbol{\theta})$, $\hat{\mathcal{M}}^H(\boldsymbol{\theta})$ as in (3.12) and $H = 2h$.

Algorithm 2 $\boldsymbol{u}_H^{(l+1)}$ $\quad$ *DirectSolve*$(R^H, T^H, \boldsymbol{u}_H^{(k)}, \mathcal{G}^H, \alpha, IMAX, Tol)$

```
1: Initialise $\boldsymbol{u}_H^{(l)} = \boldsymbol{u}_H^{(k)}$
       Construct discrete Laplacian parts of sparse matrices $\boldsymbol{A_m}$
2: for $l = 1, \cdots, IMAX$ do
       Deform template image using $\boldsymbol{u}_H^{(l)} \to T_{\boldsymbol{u}}^H$
       Compute FD approximations for derivatives of $T_{\boldsymbol{u}}^H \to \partial_{u_1} T_{\boldsymbol{u}}^H$, $\partial_{u_2} T_{\boldsymbol{u}}^H$
       Compute RHS $f_m$ (matrices) and then convert to column vectors $\boldsymbol{f_m}$
       Add remaining diagonal parts to $\boldsymbol{A_m}$
       Compute $\boldsymbol{u}_{m\,H}^{(l+1)} \to \boldsymbol{u}_{m\,H}^{(l+1)} = \boldsymbol{A}_m^{-1} \boldsymbol{f_m}$
       Reshape $\boldsymbol{u}_{m\,H}^{(l+1)}$ to matrices $u_{m\,H}^{(l+1)}$
3:     if $\|u_{1\,H}^{(l+1)} - u_{1\,H}^{(l)}\|_2^2 < Tol$ and $\|u_{2\,H}^{(l+1)} - u_{2\,H}^{(l)}\|_2^2 < Tol$ then
           Exit for loop
4:     end if
5: end for
```

Analysis of the block formulation of our proposed coarsest grid solver. Previously in order to estimate the convergence rate for the pointwise case we would have a single equation of the form shown in (3.9) for each discrete interior point from which we would determine the amplification matrix, now however we construct the amplification matrix from a single system of equations with the following structure

$$\boldsymbol{B}\boldsymbol{\Psi}_{\boldsymbol{\theta}}^{new} = \boldsymbol{C}\boldsymbol{\Psi}_{\boldsymbol{\theta}}^{old} \tag{3.18}$$

where $\boldsymbol{B}, \boldsymbol{C} \in \mathbb{R}^{2(n-2)^2 \times 2(n-2)^2}$ and $\boldsymbol{\Psi}_{\boldsymbol{\theta}}^* \in \mathbb{R}^{2(n-2)^2 \times 1}$ are block matrices and block column vectors respectively with structure

$$\boldsymbol{B} = \begin{pmatrix} \boldsymbol{B}_1 & \boldsymbol{D} \\ \boldsymbol{D} & \boldsymbol{B}_2 \end{pmatrix}, \boldsymbol{C} = \begin{pmatrix} \boldsymbol{C}_1 & \boldsymbol{D} \\ \boldsymbol{D} & \boldsymbol{C}_2 \end{pmatrix}, \boldsymbol{\Psi}_{\boldsymbol{\theta}}^* = \begin{pmatrix} \psi_{\boldsymbol{\theta}}^* \\ \psi_{\boldsymbol{\theta}}^* \end{pmatrix} \tag{3.19}$$

with $\boldsymbol{B}_m$, $\boldsymbol{C}_m$, $\boldsymbol{D} \in \mathbb{R}^{(n-2)^2 \times (n-2)^2}$ and $\psi_{\boldsymbol{\theta}}^* \in \mathbb{R}^{(n-2)^2 \times 1}$ given by

$$\boldsymbol{B}_m = \begin{pmatrix} B_{m_2} & J_1 & & \\ J_2 & \ddots & \ddots & \\ & \ddots & \ddots & J_1 \\ & & J_2 & B_{m_{n-1}} \end{pmatrix}, \boldsymbol{C} = \begin{pmatrix} C_{m_2} & & \\ & \ddots & \\ & & C_{m_{n-1}} \end{pmatrix}, \boldsymbol{D} = \begin{pmatrix} D_2 & & \\ & \ddots & \\ & & D_{n-1} \end{pmatrix}, \psi_{\boldsymbol{\theta}}^* = \begin{pmatrix} (\psi_{\boldsymbol{\theta}}^*)_1 \\ \vdots \\ (\psi_{\boldsymbol{\theta}}^*)_k \\ \vdots \\ (\psi_{\boldsymbol{\theta}}^*)_{(n-2)^2} \end{pmatrix} \tag{3.20}$$

and where B_{m_j}, C_{m_j}, D_j, $J_m \in \mathbb{R}^{(n-2)\times(n-2)}$ are given by

$$B_{m_j} = \begin{pmatrix} (b_m)_{k_2(j)} & -\frac{\alpha}{H^2}e^{\boldsymbol{i}\omega_1} & & \\ -\frac{\alpha}{H^2}e^{-\boldsymbol{i}\omega_1} & & \ddots & \\ & \ddots & \ddots & -\frac{\alpha}{H^2}e^{\boldsymbol{i}\omega_1} \\ & & -\frac{\alpha}{H^2}e^{-\boldsymbol{i}\omega_1} & (b_m)_{k_{n-1}(j)} \end{pmatrix}, C_{m_j} = \begin{pmatrix} (c_m)_{k_2(j)} & & \\ & \ddots & \\ & & (c_m)_{k_{n-1}(j)} \end{pmatrix},$$

$$D = \begin{pmatrix} (d)_{k_2(j)} & & \\ & \ddots & \\ & & (d)_{k_{n-1}(j)} \end{pmatrix}, J_1 = \begin{pmatrix} -\frac{\alpha}{H^2}e^{\boldsymbol{i}\omega_2} & & \\ & \ddots & \\ & & -\frac{\alpha}{H^2}e^{\boldsymbol{i}\omega_2} \end{pmatrix}, J_2 = \begin{pmatrix} -\frac{\alpha}{H^2}e^{-\boldsymbol{i}\omega_2} & & \\ & \ddots & \\ & & -\frac{\alpha}{H^2}e^{-\boldsymbol{i}\omega_2} \end{pmatrix} \tag{3.21}$$

with $(b_m)_{k_i(j)} = \left((\partial_{u_m} T_{\boldsymbol{u}})^2\right)_{k_i(j)} + \frac{4\alpha}{H^2}$, $(c_m)_{k_i(j)} = \left((\partial_{u_m} T_{\boldsymbol{u}})^2\right)_{k_i(j)}$, $(d)_{k_i(j)} = (\partial_{u_1} T_{\boldsymbol{u}})_{k_i(j)} (\partial_{u_2} T_{\boldsymbol{u}})_{k_i(j)}$, $\omega_m = \frac{2\theta_m \pi}{n}$ and $k_i(j) = (j-2)(n-1) + (i-1)$ for $m = 1, 2$ and i, $j = 2, \ldots, n-1$. Then the convergence rate $\mathcal{P}_B$ for the block formulation of our direct solver is estimated from the following

$$\mathcal{P}_B \equiv \mathcal{P}_B(\boldsymbol{\theta}) = \sup\left\{\rho(\hat{\boldsymbol{S}}^H(\boldsymbol{\theta})) \middle| \boldsymbol{\theta} \in \boldsymbol{\Theta}\right\} \tag{3.22}$$

with amplification matrix $\hat{\boldsymbol{S}}^H(\boldsymbol{\theta}) = \boldsymbol{B}^{-1}\boldsymbol{C}$. On this coarsest grid, n is small so estimating $\mathcal{P}_B$ is feasible.

Convergence analysis for AOS solver. We again remark that an analysis to estimate the convergence of the coarsest solver in [16] was not performed. From [16], the AOS scheme for the diffusion model is shown in (2.21) for $m = 1, 2$. We use a similar method to the one shown in §3.1 to derive the amplification matrix for the AOS method. However, since the AOS scheme solves along the x_1 and x_2 directions separately, we will obtain two convergence rates $\mathcal{P}_{A_1}$, $\mathcal{P}_{A_2}$ for these directions respectively. We start by expressing the discrete versions of (2.21) by the following system

$$\mathcal{N}_m^H \boldsymbol{u}_{p_m}^H + \mathcal{M}_m^H \boldsymbol{u}_{p_m}^H = \mathcal{G}_m^H \tag{3.23}$$

with

$$\mathcal{N}_m^H = \begin{pmatrix} 1 - 2\tau\alpha\partial_{x_m x_m}^H & 0 \\ 0 & 1 - 2\tau\alpha\partial_{x_m x_m}^H \end{pmatrix}, \; \mathcal{M}_m^H = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \; \mathcal{G}_m^H = \begin{pmatrix} \tau g_1^H - \tau F_1^H(\boldsymbol{u}) \\ \tau g_2^H - \tau F_2^H(\boldsymbol{u}) \end{pmatrix} \tag{3.24}$$

where g_m^H are the discrete RHS coming from the NMG method and $F^H(\boldsymbol{u})$ are the discrete force terms given in (2.9). The x_1, x_2 directions of the discrete Laplace operator can be represented by $\partial_{x_m x_m}^H = \mathscr{L}_{m+}^H + \mathscr{L}_{m0}^H + \mathscr{L}_{m-}^H$, where $\mathscr{L}_{m+}^H$, $\mathscr{L}_{m0}^H$, $\mathscr{L}_{m-}^H$ define the following stencils

$$\mathscr{L}_{1+}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; \mathscr{L}_{10}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; \mathscr{L}_{1-}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\mathscr{L}_{2+}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \; \mathscr{L}_{20}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \; \mathscr{L}_{2-}^H = \frac{1}{H^2}\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{3.25}$$

then we can write (3.23) in the following way

$$\mathcal{N}_{m+}^H \boldsymbol{u}_{p_m\,new}^H + \mathcal{N}_{m0}^H \boldsymbol{u}_{p_m\,new}^H + \mathcal{N}_{m-}^H \boldsymbol{u}_{p_m\,old}^H + \mathcal{M}_m^H \boldsymbol{u}_{p_m\,old}^H = \mathcal{G}_m^H \tag{3.26}$$

where $\boldsymbol{u}_{p_m\,new}^H$, $\boldsymbol{u}_{p_m\,old}^H$ denote the current and previous approximations of $\boldsymbol{u}_{p_m}^H$ in the x_m directions respectively, and

$$\mathcal{N}_{m+}^H = \begin{pmatrix} -2\tau\alpha\mathscr{L}_{m+}^H & 0 \\ 0 & -2\tau\alpha\mathscr{L}_{m+}^H \end{pmatrix}, \; \mathcal{N}_{m0}^H = \begin{pmatrix} 1 - 2\tau\alpha\mathscr{L}_{m0}^H & 0 \\ 0 & 1 - 2\tau\alpha\mathscr{L}_{m0}^H \end{pmatrix}$$

$$\mathcal{N}_{m-}^H = \begin{pmatrix} -2\tau\alpha\mathscr{L}_{m-}^H & 0 \\ 0 & -2\tau\alpha\mathscr{L}_{m-}^H \end{pmatrix}, \; \mathcal{M}_m^H = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.27}$$

for $m = 1, 2$. Using a similar process to that shown in §3.1, for computing the smoothing rate, we estimate the convergence rate from the following

$$\mathcal{P}_{\max} = \max_{\text{loc}} \mathcal{P}_{A\,loc}, \qquad \mathcal{P}_{A\,loc} = \frac{1}{2}\left(\mathcal{P}_{A_1\,loc} + \mathcal{P}_{A_2\,loc}\right),$$
$$\mathcal{P}_{A_m\,loc} \equiv \mathcal{P}_{A_m\,loc}(\boldsymbol{\theta}) = \sup\left\{\rho\big(\hat{\boldsymbol{S}}_m^H(\boldsymbol{\theta})\big) \middle| \boldsymbol{\theta} \in \boldsymbol{\Theta}\right\} \tag{3.28}$$

where $\rho(\cdot)$ again denotes the spectral radius, and $\hat{\boldsymbol{S}}_m^h(\boldsymbol{\theta})$ denote the amplification matrices given by

$$\hat{\boldsymbol{S}}_m^H(\boldsymbol{\theta}) = -\left[\hat{\mathcal{N}}_{m+}^H(\boldsymbol{\theta}) + \hat{\mathcal{N}}_{m0}^H(\boldsymbol{\theta})\right]^{-1}\left[\hat{\mathcal{N}}_{m-}^H(\boldsymbol{\theta}) + \hat{\mathcal{M}}_m^H(\boldsymbol{\theta})\right] \tag{3.29}$$

and where

$$\hat{\mathcal{N}}_{m+}^H(\boldsymbol{\theta}) = \begin{pmatrix} -\frac{2\tau\alpha}{H^2}e^{-i\omega_m} & 0 \\ 0 & -\frac{2\tau\alpha}{H^2}e^{-i\omega_m} \end{pmatrix}, \; \hat{\mathcal{N}}_{m0}^H(\boldsymbol{\theta}) = \begin{pmatrix} 1 + \frac{4\tau\alpha}{H^2} & 0 \\ 0 & 1 + \frac{4\tau\alpha}{H^2} \end{pmatrix}$$

$$\hat{\mathcal{N}}_{m-}^H(\boldsymbol{\theta}) = \begin{pmatrix} -\frac{2\tau\alpha}{H^2}e^{i\omega_m} & 0 \\ 0 & -\frac{2\tau\alpha}{H^2}e^{i\omega_m} \end{pmatrix}, \; \hat{\mathcal{M}}_m^H(\boldsymbol{\theta}) = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.30}$$

Comparison of convergence rates for two coarsest grid solvers. Once we have an estimate of the convergence rate $\mathcal{P}$, we can compute the number of iterations l required to reach a desired tolerance 10^{-k} using the following

$$l = -\frac{k \ln(10)}{\ln(\mathcal{P})} \tag{3.31}$$

Grid Size	α	AOS Solver $\mathcal{P}_A$	AOS Solver Tol $10^{-1}/10^{-2}/10^{-3}$	Direct Solver (Pointwise) $\mathcal{P}_D$	Direct Solver (Pointwise) Tol $10^{-1}/10^{-2}/10^{-3}$	Direct Solver (Block) $\mathcal{P}_B$	Direct Solver (Block) Tol $10^{-1}/10^{-2}/10^{-3}$
4×4	$\frac{1}{10}$	0.99915	2709/5417/8124	0.40511	3/6/8	0.14573	2/3/4
	$\frac{1}{20}$	0.99957	5355/10708/16062	0.51635	4/7/11	0.2[illegible]	2/4/6
	$\frac{1}{30}$	0.99971	7940/15879/23817	0.61297	5/10/15	0.[illegible]5084	3/5/7
8×8	$\frac{1}{10}$	0.99937	3655/7309/10962	0.82924	13/25/37	0.[illegible]41[illegible]	3/6/8
	$\frac{1}{20}$	0.99968	7195/14390/21584	0.90661	24/47/71	0.630[illegible]	5/10/15
	$\frac{1}{30}$	0.99979	10965/21928/32892	0.93578	35/70/105	0.76812	9/18/27
16×16	$\frac{1}{10}$	0.99947	4344/8688/13031	0.97391	88/175/262	0.9963[illegible]	632/1262/1894
	$\frac{1}{20}$	0.99973	8528/17055/25582	0.98679	174/647/520	1.0000[illegible]	-
	$\frac{1}{30}$	0.99982	12792/25583/38374	0.99116	260/519/77[illegible]	[illegible]00	-

Table 1: Comparison 2 of convergence rates (averaged over 5 FAS-NMG cycles) for the Chumchob-Chen AOS solver and our direct solver. For each solver the convergence rates and number of iterations required to reach tolerances of 10^{-1}, 10^{-2}, 10^{-3} are shown for multiple α values on various coarsest grid sizes for the lung CT example (Example 2 in Figure 3).

Grid Size	α	AOS Solver $\mathcal{P}_A$	AOS Solver Tol $10^{-1}/10^{-2}/10^{-3}$	Direct Solver (Pointwise) $\mathcal{P}_D$	Direct Solver (Pointwise) Tol $10^{-1}/10^{-2}/10^{-3}$	Direct Solver (Block) $\mathcal{P}_B$	Direct Solver (Block) Tol $10^{-1}/10^{-2}/10^{-3}$
4×4	$\frac{1}{10}$	0.99915	2708/5416/8123	0.6547[illegible]	6/11/17	0.32791	3/5/7
	$\frac{1}{20}$	0.99957	5355/10708/16061	0.79307	10/20/30	0.51094	4/7/11
	$\frac{1}{30}$	0.99971	7940/15879/23817	0.85177	15/29/44	0.62553	5/10/15
8×8	$\frac{1}{10}$	0.99937	3655/7309/10962	0.9[illegible]	39/77/115	0.70146	7/13/20
	$\frac{1}{20}$	0.99968	7195/14390/21584	0.969[illegible]	74/148/222	0.88868	20/40/59
	$\frac{1}{30}$	0.99979	10965/21928/32892	0.97894	109/217/325	0.97361	87/173/259
16×16	$\frac{1}{10}$	0.99947	4344/8688/13031	0.98925	214/427/640	1.00000	-
	$\frac{1}{20}$	0.99973	8528/17055/25582	0.994[illegible]	428/856/1283	1.00000	-
	$\frac{1}{30}$	0.99982	12792/25583/38374	0.99643	644/1288/1932	1.00000	-

Table 2: Comparison 1 of convergence rates (averaged over 5 FAS-NMG cycles) for the Chumchob-Chen AOS solver and our direct solver. For each solver the convergence rates and number of iterations required to reach tolerances of 10^{-1}, 10^{-2}, 10^{-3} are shown for multiple α values on various coarsest grid sizes for the hand example (Example 3 in Figure 3)

From Tables 1 and 2 we see that our direct solver converges much faster than the Chumchob-Chen AOS solver on several different coarsest grid sizes for both Hand and Lung CT examples (Examples 1 and 2 in Figure 3) respectively, especially on the 4×4 and 8×8 grids; this improvement has a significant impact on the number of iterations required to reach a desired tolerance, which in turn will have a noticeable effect on the number of FAS NMG cycles needed to obtain a good registration result as well as the time taken. As is also clear from both tables, the rates are too high and both solvers are not effective on the less coarse 16×16 grid, possibly due to limitation of the analysis; we would conclude that the coarsest grid is kept as 8×8.

Hence the improved NMG method, to be denoted by **unconstrained INMG**, is taken as Algorithm 1 equipped with the coarsest grid solver by Algorithm 2 and the predicted number of smoothing steps of $\nu_1, \nu_2 \geq 8$ since $\mu^8_{\text{max}} = 0.74762^8 < 0.1$ is believed to be small enough.

4 Non-folding constraint model

We now present another model to deliver diffeomorphic transforms. Folding in the transformation is a problem which can occur in image registration, unless it is specifically controlled. In real applications the presence of folding would suggest an inaccurate registration result as such transformations are non-physical. In this section we will first introduce our proposed improved diffusion model, which removes any folding that may occur in the transformation φ, as well as including a NMG scheme (Algorithm 1). Then we will extend this model to increase robust with respect to the weighting parameter α.

4.1 Improved diffusion model formulation and optimise-discretise approach

In the work by Burger et al. [11], it was explained that the sign of the determinant $\det\nabla\varphi$ can indicate the presence of any folding in the transformation $\varphi = x + u$, or more specifically the sign of

$$\det\nabla\varphi = \left(1 + u_{1_{x_1}}\right)\left(1 + u_{2_{x_2}}\right) - u_{1_{x_2}} u_{2_{x_1}}. \tag{4.1}$$

If $\det\nabla\varphi \leq 0$ then this indicates that folding in the transformation is present, while if $\det\nabla\varphi > 0$ then no folding occurs in the transformation. In [11] this information was used to add an additional term into the diffusion energy functional (2.2) which penalises this determinant in order to produce diffeomorphic image registrations, thus resulting in the following 2D hyper-elastic energy functional

$$E^{\text{Hyper}}(u) = \frac{1}{2}\int_\Omega (T_u - R)^2 + \alpha\sum_{s=1}^{2}|\nabla u_s|^2 + \beta\left(\frac{(\det\nabla\varphi - 1)^2}{\det\nabla\varphi}\right)^2 d\Omega, \tag{4.2}$$

where $\alpha \in \mathbb{R}^+$, $0 \leq \beta \in \mathbb{R}$ are weighting parameters. Although it may be possible to develop an effective smoother for solving (4.2), which has a strong non-linearity, in this paper however we instead propose an extension to the diffusion model (2.2) as a simplification of the hyper-elastic model (4.2) to control any folding. We propose to introduce a constraint into the diffusion model which ensures a positive value of the determinant (4.1). In other words, we aim to solve the following minimisation problem

$$\min_u E^{\text{Diff}}(u), \quad \text{s.t.} \quad \det\nabla\varphi > 0 \tag{4.3}$$

or equivalently, using an optimise-discretise approach, we look to solve the following EL equations

$$-\alpha\Delta u_m + F_m(u) = 0 \quad \text{s.t.} \quad \det\nabla\varphi > 0 \tag{4.4}$$

with Neumann boundary conditions $\nabla u_m \cdot n = 0$ and where $F_m(u)$ are as in (2.6) for $m = 1, 2$.

4.2 Estimating the determinant using finite elements

In order for us to be able to impose the constraint in (4.4), we must first obtain an approximation of the determinant at every discrete interior point of Ω^h, that is we need to compute

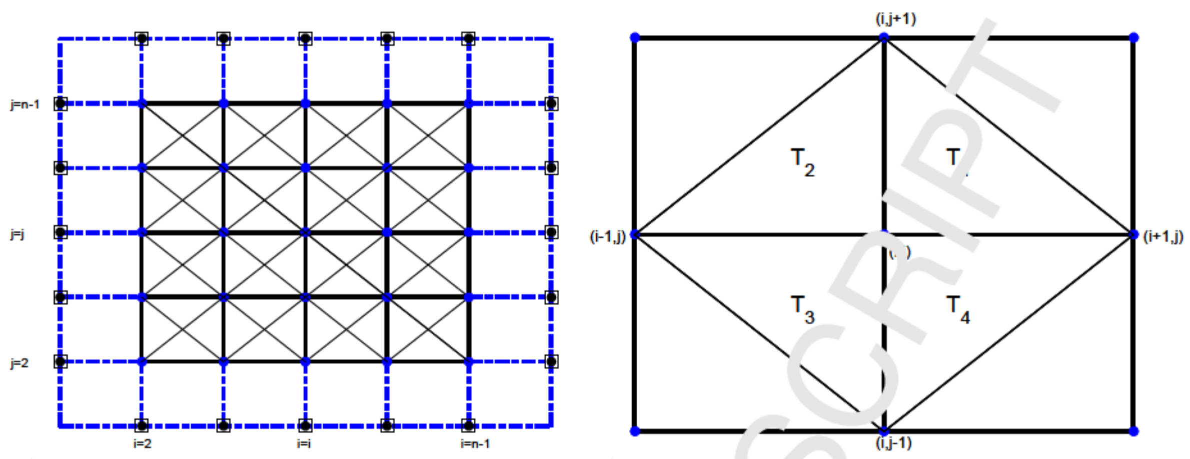
$$Q \equiv (Q_{ij}) = (\det\nabla\varphi)_{i,j} = \left(1 + (u_{1_{x_1}})_{i,j}\right)\left(1 + (u_{2_{x_2}})_{i,j}\right) - (u_{1_{x_2}})_{i,j}(u_{2_{x_1}})_{i,j} \tag{4.5}$$

where $Q \in \mathbb{R}^{(n-2)\times(n-2)}$ is the matrix consisting of determinant values at the discrete interior points (i,j) for $i, j = 2, \dots, n-1$. To compute the entry (Q_{ij}), we need to determine the discrete partial derivatives $(u_{m_{x_1}})_{i,j}$, $(u_{m_{x_2}})_{i,j}$ for $m = 1, 2$. We do this by splitting our discrete domain Ω^h into a mesh of finite elements consisting of piecewise linear triangular basis functions as shown in Figure 2(a). In fact for each interior point (i,j), we need to compute the determinant in each of the four triangles $T_1, \dots, T_4$ as shown in Figure 2(b). Doing this gives us a clearer picture of the local geometry surrounding the (i,j) point, thus allowing us to better detect any mesh folding of the transformation. Once we have determinant values for each of the triangles, we assign the smallest value to be our (Q_{ij}) entry, this in essence considers the worst possible case for each (i,j) allowing us to better detect and correct all potential folding in the transformation. Now for linear triangular basis functions, we can approximate $u_m(x)$ by the following linear functions

$$L_m(x) = a_{u_m} + b_{u_m}x_1 + c_{u_m}x_2 \tag{4.6}$$

where a_{u_m}, b_{u_m}, $c_{u_m} \in \mathbb{R}$ are coefficients to be determined for $m = 1, 2$. From (4.6) we see that the partial derivatives $u_{m_{x_1}}$, $u_{m_{x_2}}$ are given by the coefficients b_{u_m}, c_{u_m} respectively. Then looking at the first triangle T_1, at a general discrete interior point (i,j), we have the following system

$$\underline{\textbf{Triangle } T_1}: \begin{pmatrix} 1 & x_i & y_j \\ 1 & x_{i+1} & y_j \\ 1 & x_i & y_{j+1} \end{pmatrix}\begin{pmatrix} a_{1\,u_1} \\ b_{1\,u_1} \\ c_{1\,u_1} \end{pmatrix} = \begin{pmatrix} (u_1)_{i,j} \\ (u_1)_{i+1,j} \\ (u_1)_{i,j+1} \end{pmatrix}, \begin{pmatrix} 1 & x_i & y_j \\ 1 & x_{i+1} & y_j \\ 1 & x_i & y_{j+1} \end{pmatrix}\begin{pmatrix} a_{1\,u_2} \\ b_{1\,u_2} \\ c_{1\,u_2} \end{pmatrix} = \begin{pmatrix} (u_2)_{i,j} \\ (u_2)_{i+1,j} \\ (u_2)_{i,j+1} \end{pmatrix};$$



(a) Global mesh of finite elements of full discrete domain (b) Local mesh of finite elements at a general discrete point (i,j)

Figure 2: Finite element splitting of the discrete domain Ω^h using linear triangle basis functions

we obtain similar systems for each of the remaining triangles T_2, T_3 and T_4. Then, to compute the coefficients $a_{l\,u_m}$, $b_{l\,u_m}$, $c_{l\,u_m}$, we solve

$$s_l = A_l^{-1} v_{1\,l}, \quad t_l = A_l^{-1} v_{2\,l} \tag{4.7}$$

where $s_l = (a_{l\,u_1}, b_{l\,u_1}, c_{l\,u_1})^T$, $t_l = (a_{l\,u_2}, b_{l\,u_2}, c_{l\,u_2})^T$ are the column vectors of coefficients for $(u_1)_{i,j}$, $(u_2)_{i,j}$ respectively, A_l^{-1} are the inverses of the matrices corresponding to the edges of the triangles T_l and $v_{m\,l} = (u_{m\,1}, u_{m\,2}, u_{m\,3})^T$ are the values of u_m at each vertex of the triangles T_l for $l = 1, \ldots, 4$, $m = 1, 2$. Then, once all elements of Q have been computed, we take the minimum value of the matrix Q to be used to see if the constraint has been satisfied. This method can be summarised by Algorithm 3. Once we have a value for Q_{min}, we use Algorithm 4 to impose the constraint and determine whether we accept the updated transformation or not.

In practice, Algorithm 3 can be computationally expensive on larger grid sizes owing to the fact that we must solve eight inverse problems at every discrete interior point in the discrete domain Ω^h, consequently this has a severe impact on the CPU time of the NMG scheme for our constrained model. In Appendix A we demonstrate how Algorithm 3 can be optimised to significantly decrease CPU cost for each iteration of the determinant computation. The method outlined in Algorithm 8 is how we actually compute the determinant in practice, and the results shown in §5.2 are also obtained using this algorithm.

4.3 Numerical solution and NMG algorithm for a constrained diffusion model

Based on our NMG framework **unconstrained INMG**, we will solve our constrained diffusion model by NMG. Adding a constraint, the same pointwise smoother as the one shown in §2.2 and the same coarsest grid solver as the one described in §3.2 are used. Then our proposed NMG algorithm is shown in Algorithm 6, which we denote **constrained INMG**.

4.4 An adaptive α constrained diffusion model

While our **constrained INMG** does ensure that the deformations obtained are non-folding, in cases where folding is severe the deformation field $\boldsymbol{u}$ can be penalised so heavily that the deformed template image $T_{\boldsymbol{u}}$ may have moved very little when compared with the original template image T. To overcome this problem we propose an extension to our **constrained INMG** model, whereby we re-initialise the NMG method using a larger value of α if the constraint has not been satisfied within a small number of iterations. To construct this adaptive α scheme, we modify the determinant check shown in Algorithm

4 as seen in Algorithm 5. From Algorithm 5 we see that if we reach the iteration limit $LMAX$, we exit out of the FAS-NMG algorithm and this is when we re-initialise the NMG with a larger weighting parameter α. This process can be summarised by Algorithm 7, and where the algorithm AdaptFASNMG is the same as Algorithm 6 except now Algorithm 5 is used to check the constraint instead of Algorithm 4. Another advantage of the adaptive α scheme shown in Algorithm 7 is its robustness to the choice of parameter α. Even if the initial α is set too small such that severe folding would normally occur, because we keep re-initialising the problem with new values of α, we automatically find a pseudo-optimal α value where folding is avoided. This will be shown in the next section. Using the pointwise smoother from §2.2, and the coarsest grid solver from §3.2 along with Algorithm 7, then we denote our adaptive α model by **adaptive INMG**.

Algorithm 3 Q_{min} $ComputeQ(\boldsymbol{u}^h, n, h)$

```
1: for $i = 2, \ldots, n-1$ do
2:     for $j = 2, \ldots, n-1$ do
3:         for $l = 1, \ldots, 4$ do
               Compute the vectors $\boldsymbol{s}_l$, $\boldsymbol{t}_l$ using (4.7)
               Compute determinant for triangle $T_l \to \tilde{Q}_l = (1+b_{l\,u_1})(1+a_{l\,u_2}) - a_{l\,u_1}b_{l\,u_2}$
4:         end for
           Assign minimum $\tilde{Q}$ to be entry $(Q_{ij}) \to (Q_{ij}) = \min\{\tilde{Q}_1, \ldots, \tilde{Q}_4\}$
5:     end for
6: end for
       Take minimum entry in $\boldsymbol{Q}$ to be minimum determinant value $\to Q_{min} = \min\{\boldsymbol{Q}\}$
```

Algorithm 4 $\boldsymbol{u}_h^{(k+1)}$ $ConstrainU(\boldsymbol{u}_h^{(k)}, h, \lambda, LMAX)$

```
1: for $l = 1, \cdots, LMAX$ do
2:     Compute minimum value of determinant $Q_{min}$ using Algorithm 3
3:     if $Q_{min} > 0$ and $l \le LMAX$ then
           Accept update $\boldsymbol{u}_h^{(k+1)} = \boldsymbol{u}_h^{(k)}$
4:     else if $Q_{min} \le 0$ and $l < LMAX$ then
           Reject update and set $\boldsymbol{u}_h^{(k)} = \lambda \boldsymbol{u}_h^{(k)}$, $\lambda \in (0,1)$
5:     else if $Q_{min} \le 0$ and $l = LMAX$ then
           Error $\to$ Constraint failed
6:     end if
7: end for
```

Algorithm 5 $\left[\boldsymbol{u}_h^{(k+1)}, c, done\_alpha\right]$ $AdaptiveU(\boldsymbol{u}_h^{(k)}, h, \lambda, LMAX)$

```
1: Save current 'good' approximation $\to \hat{\boldsymbol{u}}_h^{(k)} = \boldsymbol{u}_h^{(k)}$, $c = 0$
2: for $l = 1, \cdots, LMAX$ do
3:     Compute minimum value of determinant $Q_{min}$ using Algorithm 3
4:     if $Q_{min} > 0$ and $l \le LMAX$ then
           Accept update $\boldsymbol{u}_h^{(k+1)} = \boldsymbol{u}_h^{(k)}$, $\hat{\boldsymbol{u}}_h^{(k)} = \boldsymbol{u}_h^{(k)}$, $c = c+1$, $done\_alpha = 1$, $break$
5:     else if $Q_{min} \le 0$ and $l < LMAX$ then
           Reject update and set $\boldsymbol{u}_h^{(k)} = \lambda \boldsymbol{u}_h^{(k)}$, $\lambda \in (0,1)$, $c = c+1$
6:     else if $Q_{min} \le 0$ and $l = LMAX$ then
           Reset to 'good' approximation $\to c = LMAX$, $\boldsymbol{u}_h^{(k+1)} = \hat{\boldsymbol{u}}_h^{(k)}$, $done\_alpha = 0$
7:     end if
8: end for
```

Algorithm 6 $u_h^{(k+1)}$ $\quad ConstFASNMG(R^h, T^h, n, h, level, u_h^{(k)}, \mathcal{G}^h, \alpha, \nu_1, \nu_2)$

```
1: Pre-smoothing step by performing $\nu_1$ steps (relaxation sweeps)    $\bar{u}_h^{(k)}$    $Smooth(R^h, T^h, u_h^{(k)}, \mathcal{G}^h, \alpha, \nu_1)$
2: Coarse-grid correction
       Compute the residual $r_h^{(k)} = \mathcal{G}^h - \mathcal{N}^h(u_h^{(k)})$
       Restrict residual and smooth approximations $r_H^{(k)} = \mathcal{R}_h^H r_h^{(k)}$, $\bar{u}_H^{(k)} = \mathcal{R}_h^H \bar{u}_h^{(k)}$
       Set $level \to level - 1$, $H = 2h$, $nc = \frac{n}{2}$
       Form RHS of coarse grid PDEs $\mathcal{G}^H = r^H + \mathcal{N}^H(\bar{u}_H^{(k)})$
       Solve residual equation on coarse grid to obtain approximations $\tilde{u}_H^{(k)}$
3: if $level = 1$ then
       Solve to obtain high accuracy solutions $u_H^{(k)}$ using a coarsest grid solver.
4:     Use Algorithm 4 to determine whether update is accepted
5: else $level > 1$   Repeat the FAS-NMG-CONST procedure recursively to the next level i.e.
       $\bar{u}_H^{(k)}$    $ConstFASNMG(R^H, T^H, nc, H, level - 1, \tilde{u}_H^{(k)}, \mathcal{G}^H, \alpha, \nu_1, \nu_2)$
6: end if
       Compute the correction $e_H^{(k)} = u_H^{(k)} - \bar{u}_H^{(k)}$
       Interpolate the correction to next fine grid level $e_h^{(k)} = \mathcal{I}_H^h e_H^{(k)}$
       Update current grid level approximations using correction $\hat{u}_h^{(k)} = \bar{u}_h^{(k)} + e_h^{(k)}$
7: Post-smoothing step by performing $\nu_2$ steps (relaxation sweeps)    $u_h^{(k+1)}$    $Smooth(R^h, T^h, \hat{u}_h^{(k)}, \mathcal{G}^h, \alpha, \nu_2)$
       Computes $u_h^{(k+1)}$ by performing $\nu_2$ relaxation sweeps of smoother
8: Use Algorithm 4 to determine whether update is accepted if on finest grid level $\Omega^h$
```

Algorithm 7 $u_h^{(k+1)}$ $\quad Adaptive\alpha\left(R^h, T^h, n, h, u_h^{(k)}, i^{\alpha}_{max}\right)$

```
1: Set $done\_NMG = 0$, $done\_alpha = 0$
2: while $done\_NMG \neq 1$ do
3:     if $i^{\alpha} = i^{\overline{\alpha}}_{max}$ then
        $LMAX = 100$
4:     end if
5:     while $done\_NMG \neq 1$ do
6:         Set previous ‘good’ approximation $\to u_h^{(k)} = \hat{u}_h^{(k)}$
7:         Perform FAS-NMG $\to \left[u_h^{(k+1)}, c\right]$    AdaptFASNMG $\left(R^h, T^h, n, j, level, \hat{u}_h^{(k)}, \mathcal{G}^h, \alpha, \nu_1, \nu_2\right)$
8:         if $c \leq LMAX$ and $done\_alpha \neq 1$ then
        break
9:         end if
10:        if NMG convergence criteria satisfied then
        $done\_NMG = 1$
11:        end if
12:    end while
13:    if $c \leq LMAX$ and $done\_alpha \neq 1$ then
14:        Set $\alpha = 2\alpha$, $i^{\alpha} = i^{\alpha} + 1$, $u_h^{(k)} = \hat{u}_h^{(k)}$
15:    end if
16: end while
```

5 Experimental results

Here we will present and compare the results of four models

- **M1** — the NMG method **CCNMG** from [16] i.e. Algorithm 1;
- **M2** — the improved NMG method **unconstrained INMG** of §3.2;
- **M3** — the NMG method **constrained INMG** of §4.3 i.e. Algorithm 6;
- **M4** — the NMG method **adaptive INMG** of §4.4 i.e. Algorithm 7.

Firstly we will demonstrate how our more accurate analysis of the smoothing rate, along with our new coarsest grid solver, impact the number of NMG cycles required for the method to converge when compared with **M1**. In addition we will also show how this improved convergence of our NMG method **M2** results in a significant decrease in CPU time, as well as an improvement in the accuracy of the registration, when compared with **M1**.

Secondly, we will show how our method **M3** overcomes the issue of transformation folding while still maintaining good accuracy and CPU times compared with our unconstrained model **M2** and the Chumchob-Chen model **M1**.

Thirdly we will show how our method **M4** not only overcomes the problem of mesh folding while keeping a good level of accuracy and CPU times, but also how it can maintain these good transforms while being robust to parameter choice when compared with the other models.

To gain a quantitative measure of the accuracy of the NMG methods, we use Structural Similarity (SSIM) [39] as well as the relative error given by Err $= \frac{\|T_{\boldsymbol{u}}-R\|_2^2}{\|R\|_2^2}$. Moreover, in order to highlight the convergence problem of the **M1**, and for fairness, we will consider a method to have converged only if any of the following stopping criteria has been satisfied:

- The average relative residual of the EL equations reaches a tolerance of $\varepsilon_1 = 10^{-2}$
- The maximum relative residual of the EL equations reaches a tolerance of $\varepsilon_2 = 10^{-2}$
- The number of NMG cycles reaches the maximum number of $\varepsilon_3 = 25$

We shall take 3 pairs of test images (shown in Fig.3) to experiment and compare registrations:
Example 1 — a pair of CT images from Fig.3(a, d),
Example 2 — a pair of CT images from Fig.3(b, e),
Example 3 — a second pair of Hand images from Fig.3(c, f).

Moreover, in Tables 5-6 we indicate whether a test has been 'successful' (results highlighted in green) or whether it has 'failed' (results highlighted in red). We say that a test has 'failed' if the maximum number of NMG cycles ε_3 has been reached, or if there is folding in the result (i.e. $Q_{min} < 0$). Additionally bold values indicate the results which give the best SSIM and relative error values for each test.

5.1 Comparative results of models M1 and M2

Here we will demonstrate the improvement of the new **M2** over **M1**. As mentioned in §3, our improvement is to overcome the convergence problem that was present in the former method.

Test on Example 1. From Figures 8 and 9, we see that our **M2** produces visually similar deformed template images $T_{\boldsymbol{u}}$ and final error images $|T_{\boldsymbol{u}} - R|$ when compared with those obtained from **M1**. The first two columns of Table 5 show several test results of varying resolutions and parameters α. There, abbreviations 'SSIM', 'Err', 'NMG' 'CPU' represent the final structural similarity, final relative error, number of multigrid cycles performed and CPU time respectively. When we look at the table we see that our **M2** requires consistently fewer NMG cycles to produce these accurate results. In fact, the **M1** method almost always fails to converge within the allowed number ε_3 of NMG cycles to the required tolerances. This confirms our statements earlier on the convergence problem of **M1**. Moreover, this also leads to a drastic improvement in CPU time, especially in the 512^2 and 1024^2 cases where the **M1** model requires a much larger number of NMG cycles.

Test on Example 2. Although visual differences between the models are small in Figures 6 and 7, in Table 4, we see that **M2** is better than **M1** (in all indicators: SSIM/Err/NMG cycles/CPU) for the first α value, but for the other two cases of α both models failed to give diffeomorphic maps due to $\det \nabla \varphi < 0$.

Test on Example 3. For the second lung CT example visual differences between the models are small in Figures 4 and 5. We can see that, from Table 3, **M2** is successful for all cases of α but **M1** failed in several cases. On convergence alone, **M1** is not as fast as **M2** because it takes many NMG cycles.

We remark that, in the **M1** method tested above, we have used the original CCNMG AOS solver on the coarsest grid but the (new) updated smoothing rates to predict the number of smoothing steps required on fine grids; that is to say, the NMG cycles displayed are better than the original work. To illustrate the importance of our re-analysis in LFA, we will give a brief comparison using the old and new smoothing rates for a specific test. Considering **Example 1** from Figure 8 of size 128^2 with $\alpha = \frac{1}{10}$, we obtained $SSIM/Err(\%)/NMG/CPU(s)$ values of 0.774/1.48/21/1.169 using the **M1** method with smoother steps based upon the rate $\mu = 0.5$. However if we perform the same test with smoother steps based upon our re-calculated rate $\mu = 0.74762$, we obtain values of 0.775/1.46/10/0.959. Clearly there is a vast improvement (reduction) in the number of NMG cycles required with small improvements in the other three values and the overall improvement of **M2** over **M1** is also due to the new coarsest grid solver.

5.2 Comparative results of models M2 and M3

In §4 we introduced our constrained version **M3** in order to prevent any folding from occurring in the transformation. This was achieved by ensuring $\det \nabla\varphi > 0$ for every discrete interior point in Ω^h. Here we will present results comparing **M2** and **M3** to show how this constraint does indeed prevent folding while still maintaining good accuracy and CPU time using the same three examples from §5.1. The abbreviation Q_{min} represents the minimum determinant value $\det \nabla\varphi$. Here small 'Err' means a small fitting error while $Q_{min} > 0$ implies a correct registration transformation.

Test on Example 1. From columns 2 and 3 of Table 5 we see that our **M2** always produces positive Q_{min} values; as a result we obtain the exact same results with our **M3** method with very small increases in CPU times owing to the constraint checking. This also translates to Figures 8 and 9 where we see that all images look very similar visually.

Test on Example 2. From Table 4 we see that **M3** has overcome the mesh folding problems of **M2** by positive Q_{min} values in all cases. In achieving this convergent non-folding result, the number of NMG cycles taken by **M3** is more than **M2**. Although the CPU times in these cases also increase noticeably, we do however still see a reduction and consistency in the number of NMG cycles when compared with the **M1** method. The CPU time increase could be reduced by a more computationally efficient implementation of our smoother code to penalise the transformation only in regions where folding is present.

Test on Example 3. Here we see the exact same pattern as in Example 1 since our **M3** produces positive determinant values in all cases and identical results to **M2** with small increases in CPU times as shown in Table 3, with improvements in all categories over the **M1** method especially in convergence and CPU times.

5.3 Comparative results of models M3 and M4

Additionally in §4 we introduce and extension to our **M3** model to be robust to parameter choice while maintaining a non-folding transformation. Here we will consider a case where severe folding would occur and our **M3** model, while producing a non-folding deformation, performs poorly in terms of registration accuracy whereas our **M4** model also avoids folding while producing good registration accuracy.

From Table 6 we see that although we obtain very good accuracy from our **M2** model, we also have severe folding in the transformations in all tests as indicated by the negative Q_{min} values. Looking at the results for our **M3** model we see that the folding problem has been overcome and all Q_{min} values are now positive, however we also see that we have lost the accuracy of the result with regard to error when compared with the **M2** results, especially on the 127^2 and 256^2 images. Our **M4** model on the other hand no only produces non-folding results like with our **M3** model, but also maintains a similar level of accuracy when compared with the results from our **M2** model. In addition we also see that our **M4** model achieves this with only a slight increase in CPU time when compared with those from the **M2** model, and is over twice as fast as our **M3** model. From Figures 10 and 11 we see that visually there is a noticeable difference between the deformed template from our **M3** model compared with those from our **M2** and **M4** models, especially in the error images.

5.4 Test of NMG efficiency and parameter robustness

NMG efficiency. In this work, we are concerned with transforms' quality and fast solution by a NMG. For the latter, we expect the optimal efficiency of $O(N \log N)$ complexity in achieving a fixed accuracy (with $N = n^2$ for $n \times n$ images). Let t_n denote the CPU times required by registering two $n \times n$ images. Then for an optimal NMG, we expect the CPU increase to be of ratio $t_n/t_{n/2} = Cn^2 \log n^2/(C(n/2)^2 \log(n/2)^2) = 4 + 4 \log 4/\log(n/2)^2 \approx 4.5$. In Table 7, we show test results of all four NMG methods for varying resolutions, where in **M1** we use the original analysis of [16] to set the number of smoothing steps. Clearly **M2**, **M3** and **M4** exhibit nearly optimal complexity but **M1** shows irregular patterns, which justify our re-analysis for Algorithm 1.

Finally to give an indication of the convergence history of **M1** and **M2**, we plot in Fig.12 the residuals for more NMG cycles. Evidently **M2** has faster convergence plot than **M1**.

Parameter robustness. In the diffusion model, the weighting parameter α indicates how strongly we wish to enforce smoothness on the deformation from the regularisation term. Specifically, a larger value of α will impose a strong penalisation on non-smooth deformations leading to no folding, however this also leads to a less accurate registration with regards to error. On the other hand a smaller value of α will lead to a more accurate registration in terms of error, but will also increase the likelihood of folding occurring. Moreover, selecting a 'good' value for α can be very time consuming as in general a pre-multigrid routine is usually required to find this 'best' α (for example the cooling process in [16]), which can noticeably increase the computational work and CPU time. For this reason, having a model which is robust to the choice of weighting parameter is very useful as the need for finding the 'best' value for α is less important. Here we will compare how the value of α impacts the relative error (denoted 'Err') and minimum determinant value (denoted 'Q_{min}') for models M2 and **M4**. From Figure 13(a) we see that as α gets smaller the error also decreases, however looking at Figure 13(b) we see that the value of Q_{min} is also decreasing to a point where it is always negative as highlighted by the dotted line. This suggests that our model **M2** has a limit where it maintains physically accurate non-folding deformations, and once past this point folding always occurs. Looking at Figure 14(a) we see that our **M4** model follows a similar pattern with regard to a decreasing error as α decreases like with our **M2** model, however from Figure 14(b) we see that our **M4** model always maintains the physical integrity of the deformation with $Q_{min} > 0$ for all tested values of α. From this we can conclude that our adaptive α model **M4** is very robust to the initial value of α, even for small values, while maintaining a consistently good registration accuracy in terms of error.

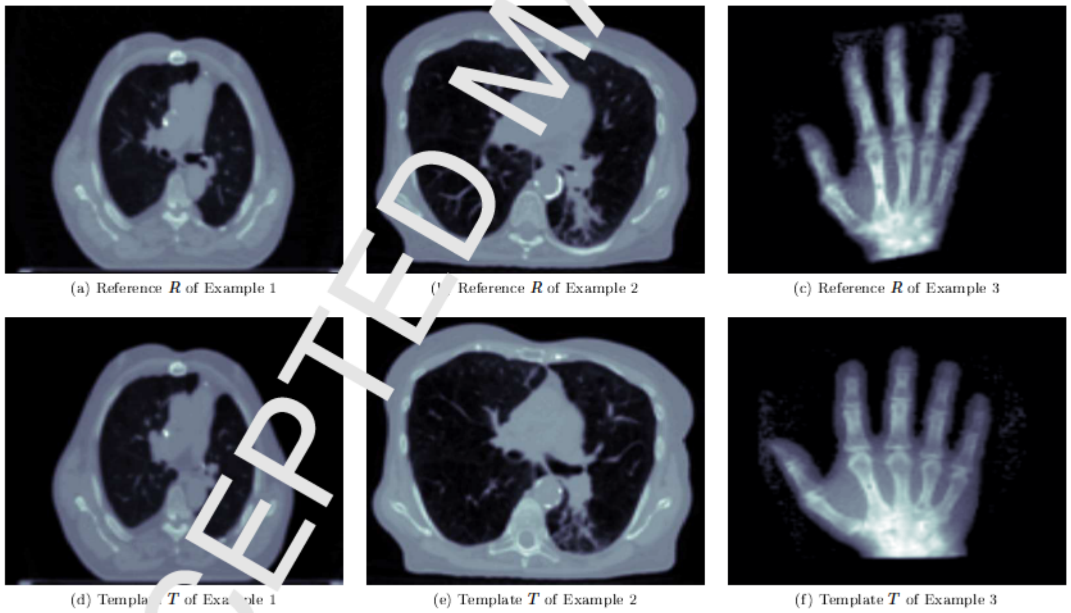

(a) Reference R of Example 1
(b) Reference R of Example 2
(c) Reference R of Example 3
(d) Template T of Example 1
(e) Template T of Example 2
(f) Template T of Example 3

Figure 3: Three Pairs of Test Images.

6 Conclusions

In this paper we have first presented an improved NMG method, with regard to convergence and accuracy, over that proposed by Chumchob-Chen through a more detailed and accurate analysis of the multigrid method, as well as a different coarsest grid solver. Secondly we proposed an extension to our NMG method with the aim of producing non-folding transformations, which was achieved by imposing an additional constraint into our improved NMG method. Next we extended our **constrained INMG** to be more robust to parameter choice while keeping non-folding deformations and good registration accuracy. We then used three examples to demonstrate the improvement in accuracy and NMG cycles required for convergence over the Chumchob-Chen NMG, as well as how our **constrained INMG** and **adaptive INMG** overcame folding by ensuring $\det \nabla \varphi > 0$.

Image Size n^2	α	**M1**	**M2**	**M3**
		SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}
128^2	$\frac{1}{5}$	0.930/0.54/2/0.391/0.797	**0.943/0.41**/1/0.333/0.819	**0.943/0.41**/1/0.[illegible]39/0.819
256^2		0.943/0.45/5/1.512/0.715	**0.951/0.42**/2/1.927/0.803	**0.[illegible]/[illegible]42**/2/2.051/0.803
512^2		0.959/0.44/13/22.387/0.854	**0.964/0.43**/2/9.426/0.801	**[illegible]964/[illegible]43**/2/9.721/0.801
1024^2		0.972/0.44/25/196.585/0.872	**0.975/0.43**/3/66.178/0.822	**[illegible]75/[illegible].43**/3/69.500/0.822
128^2	$\frac{1}{10}$	0.931/0.52/1/0.316/0.612	**0.945/0.39**/1/0.425/0.694	**0.94[illegible]/[illegible]39**/1/0.437/0.694
256^2		0.945/0.43/25/6.887/0.464	**0.953/0.40**/1/1.090/0.660	**[illegible]53/0.4[illegible]**/1/1.164/0.660
512^2		0.961/0.43/10/17.204/0.734	**0.965/0.41**/1/5.057/0.668	**0.96[illegible]/[illegible]41**/1/5.250/0.668
1024^2		0.974/0.43/23/180.785/0.745	**0.976/0.42**/1/22.972/0.685	**[illegible]76/0.42**/1/24.182/0.685
128^2	$\frac{1}{15}$	0.937/0.45/25/1.919/0.619	**0.947/0.38**/3/0.976/0.559	**0.94[illegible]/0.38**/3/1.010/0.559
256^2		0.948/0.40/25/6.820/0.230	**0.954/0.39**/1/1.080/0.511	**0.954/0.39**/1/1.146/0.511
512^2		0.962/0.41/12/20.657/0.631	**0.966/0.40**/1/4.886/0.526	**[illegible]966/0.40**/1/5.150/0.526
1024^2		0.975/0.41/18/141.395/0.644	**0.977/0.40**/1/24.642/0.55[illegible]	**0.977/0.40**/1/25.546/0.554

Table 3: Example 2 – Registration comparison of 3 methods on multiple image sizes for different α values, with an initial relative error of 0.60% and initial SSIM values of 0.933, 0.942, 0.957, 0.972 for the 128^2, 256^2, 512^2, 1024^2 images respectively.

Image Size n^2	α	**M1**	**M2**	**M3**
		SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}
128^2	$\frac{1}{5}$	0.750/1.28/4/0.530/0.642	**0.764/1.17**/2/0.586/0.6[illegible]	**0.764/1.17**/2/0.603/0.664
256^2		0.752/1.35/11/3.102/0.640	**0.786/1.14**/3/[illegible]926/0.6[illegible]	**0.786/1.14**/3/3.015/0.645
512^2		0.806/1.32/25/42.794/0.618	**0.832/1.1[illegible]**/[illegible]/18.[illegible]/[illegible]683	**0.832/1.18**/4/19.188/0.683
1024^2		0.860/1.34/25/199.920/0.640	**0.883/1.20**/4/[illegible]853/0.701	**0.883/1.20**/4/94.397/0.701
128^2	$\frac{1}{10}$	0.766/1.04/3/0.456/0.406	**0.783/[illegible]**/[illegible]/0.070	**0.783/0.95**/2/0.715/0.070
256^2		0.768/1.11/7/2.038/0.344	**0.803/[illegible]**/3/2.879/ − 0.028	0.800/0.95/6/6.251/0.027
512^2		0.819/1.07/20/34.047/0.280	**0.847/0.95**/[illegible]/14.244/0.091	**0.847/0.95**/3/14.784/0.091
1024^2		0.873/1.06/25/195.431/0.271	**0.[illegible]/[illegible].96**/4/6[illegible].196/0.145	**0.893/0.96**/4/71.186/0.145
128^2	$\frac{1}{15}$	0.774/0.89/3/0.488/0.080	**0.79[illegible]/0.8[illegible]**/3/[illegible].920/ − 0.687	0.757/1.18/8/3.424/0.015
256^2		0.802/0.77/6/1.786/ − 0.165	**0.811/[illegible]6**/2/1.952/ − 0.862	0.772/1.23/5/8.047/0.024
512^2		0.826/0.91/15/25.598/ − 0.122	**[illegible]54/0.[illegible]**/3/13.750/ − 0.680	0.827/1.18/6/40.789/0.012
1024^2		0.880/0.89/25/195.370/ − 0.156	**0.[illegible]/[illegible]80**/3/69.076/ − 0.584	0.881/1.16/6/182.460/0.011

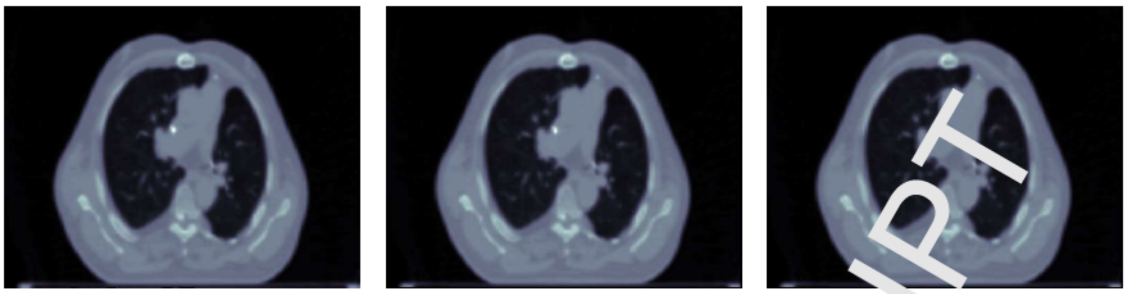
Table 4: Example 2 – Registration comparison of 3 methods on multiple image sizes for different α values, with an initial relative error of 1.99% and initial SSIM values of 0.667, 0.704, 0.769, 0.838 for the 128^2, 256^2, 512^2, 1024^2 images respectively. Clearly although **M2** does converge quickly, the **M3** offers both speed and correct transforms.

Image Size n^2	α	**M1**	**M2**	**M3**
		SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}
128^2	$\frac{1}{5}$	**0.742/2.42**/16/1.4[illegible]/0.66[illegible]	0.717/3.30/2/0.633/0.554	0.717/3.30/2/0.644/0.554
256^2		**0.743/2.61**/25/7.[illegible]7/0.[illegible]1	0.725/3.24/2/1.959/0.517	0.725/3.24/2/2.093/0.517
512^2		0.748/3.68/25/4[illegible]542/[illegible]17	**0.750/3.24**/2/9.397/0.498	**0.750/3.24**/2/9.691/0.498
1024^2		0.747/6.85/25/1[illegible]5.731/0.6[illegible]	**0.784/3.24**/2/45.445/0.486	**0.784/3.24**/2/47.728/0.486
128^2	$\frac{1}{10}$	**0.775/1.46**/[illegible]/[illegible]959/0.600	0.758/1.89/3/0.868/0.420	0.758/1.89/3/0.892/0.420
256^2		**0.776/1.46**/25/6.78[illegible]/[illegible]639	0.760/1.87/2/1.984/0.376	0.760/1.87/2/2.118/0.376
512^2		**0.778**/2.02/[illegible]/42.149/0.[illegible]02	**0.778/1.86**/2/9.350/0.348	**0.778/1.86**/2/9.706/0.348
1024^2		0.780/3.6[illegible]/25/[illegible]5.403/0.532	**0.807/1.87**/2/45.620/0.332	**0.807/1.87**/2/48.026/0.332
128^2	$\frac{1}{15}$	**0.790/[illegible].13**/[illegible]/0.814/0.563	0.783/1.33/3/0.891/0.324	0.783/1.33/3/0.922/0.324
256^2		**0.791/1.[illegible]**/[illegible]2/5.992/0.561	0.781/1.31/3/2.907/0.266	0.781/1.31/3/3.086/0.266
512^2		0.78[illegible]/1.40/25/[illegible]225/0.539	**0.794/1.31**/3/13.786/0.246	**0.794/1.31**/3/14.526/0.246
1024^2		0.7[illegible]9/2.3[illegible]/25/194.026/0.390	**0.819/1.31**/3/66.949/0.235	**0.819/1.31**/3/69.405/0.235

Table 5: Example 3 – Registration comparison of 3 methods on multiple image sizes for different α values, with an initial relative error of 13.25% and initial SSIM values of 0.551, 0.587, 0.639, 0.693 for the 128^2, 256^2, 512^2, 1024^2 images respectively.

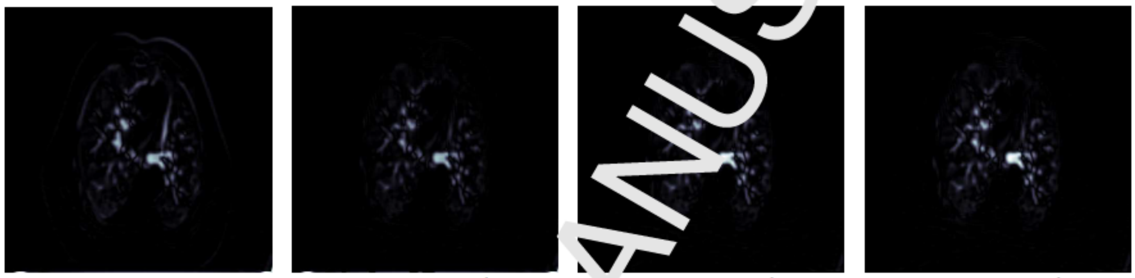
Image Size n^2		**M2**	**M3**	**M4**
		SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}	SSIM/Err (%)/NMG/CPU (s)/Q_{min}
128^2	$\frac{1}{40}$	**0.[illegible]12/0.95**/2/0.686/ − 3.078	0.630/7.56/6/2.676/0.032	0.758/1.91/3/0.711/0.554
256^2		**0[illegible]16/0.74**/2/2.458/ − 0.463	0.630/9.59/3/5.076/0.060	0.815/0.82/2/2.178/0.168
512^2		**[illegible].824**/0.82/2/9.729/ − 0.132	0.805/1.10/4/27.558/0.025	**0.824/0.74**/2/10.318/0.351
1024^2		0.832/0.78/2/45.762/ − 0.163	0.812/1.64/4/121.546/0.086	**0.842/0.73**/2/58.604/0.358

Table 6: Example 3 - Registration comparison of 3 methods on multiple image sizes for a 'bad' choice of α, with an initial relative error of 13.25% and initial SSIM values of 0.551, 0.587, 0.639, 0.693 for the 128^2, 256^2, 512^2, 1024^2 images respectively.



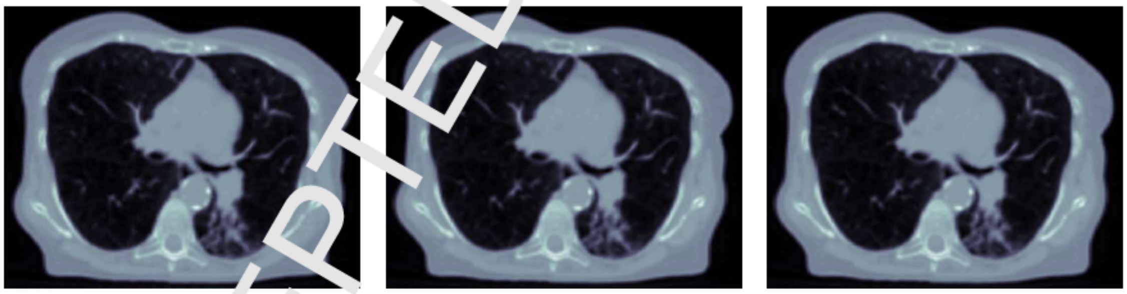
(a) **M1** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 0.43\%$ (b) **M2** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 0.41\%$ (c) **M3** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 0.41\%$

Figure 4: Example 1 – Registration of 3(a) R and 3(d) T of size 512×512 by 3 methods. Image (a) shows the deformed template image $T_{\boldsymbol{u}}$ obtained using the **M1**, while image (b) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M2** and image (c) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M3** for the parameter value $\alpha = \frac{1}{10}$.



(a) Original $|T - R|$ (b) **M1** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (c) **M2** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (d) **M3** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$

Figure 5: Example 1 – Difference images corresponding to registrations of Fig.4. Image (a) shows the initial error between T and R, while images (b), (c), (d) show the final errors between $T_{\boldsymbol{u}}$ and R for **M1**, our **M2** and our **M3** respectively.



(a) **M1** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 1.0\%$ (b) **M2** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 0.95\%$ (c) **M3** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 0.95\%$

Figure 6: Example 2 – Registration of 3(b) R and 3(e) T of size 512×512 by 3 methods. Image (a) shows the deformed template image $T_{\boldsymbol{u}}$ obtained using the **M1**, while image (b) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M2** and image (c) shows the deformed template image $T_{\boldsymbol{u}}$ for our constrained NMG for the parameter value $\alpha = \frac{1}{10}$.

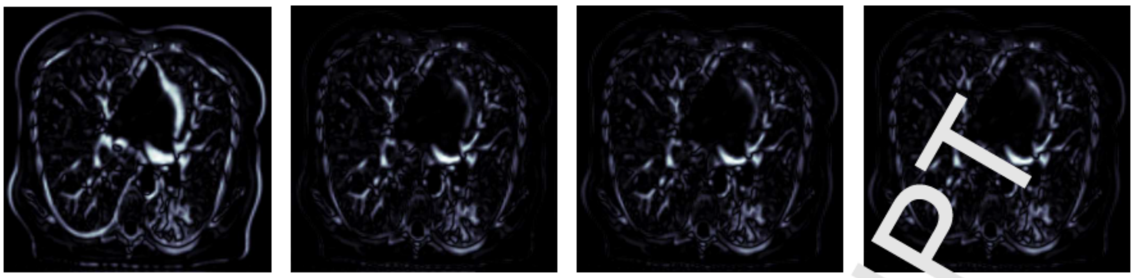

(a) Original $|T - R|$ (b) **M1** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (c) **M2** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (d) **M3** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$

Figure 7: Example 2 – Difference images corresponding to registrations of Fig.6. Image (a) shows the initial error between T and R, while images (b), (c), (d) show the final errors between $T_{\boldsymbol{u}}$ and R for the **M1**, our **M2** and our **M3** respectively.

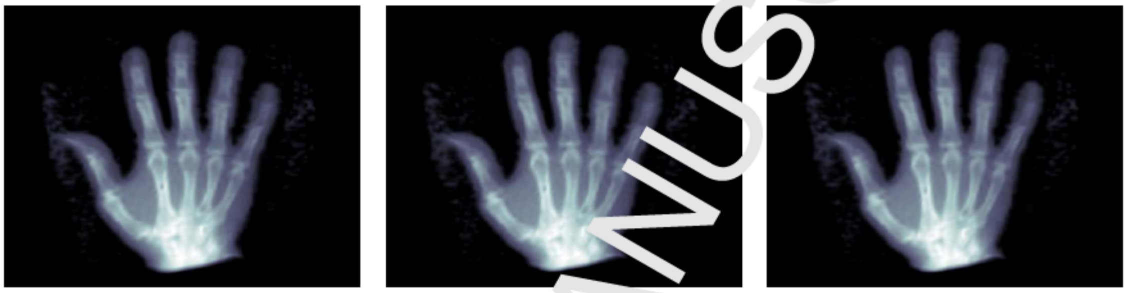

(a) **M1** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 2.02\%$ (b) **M2** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err =$ [illegible]% (c) **M3** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{10}$ $Err = 1.86\%$

Figure 8: Example 3 – Registration of $3(c)$ R and $3(f)$ T of size 512×512 by 3 methods. Image (a) shows the deformed template image $T_{\boldsymbol{u}}$ obtained using the **M1**, while image (b) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M2** and image (c) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M3** for the parameter value $\alpha = \frac{1}{10}$.

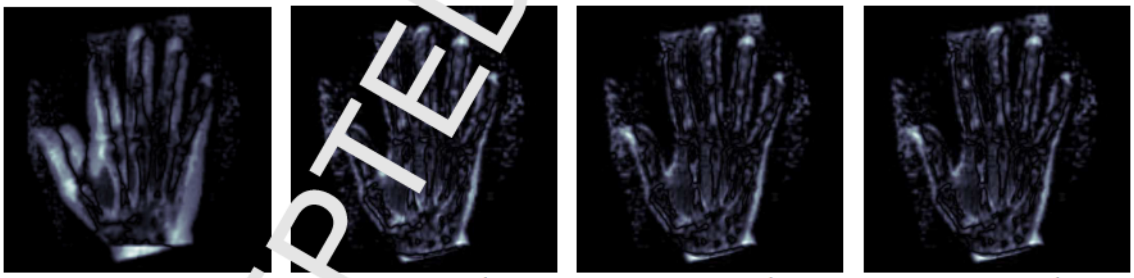
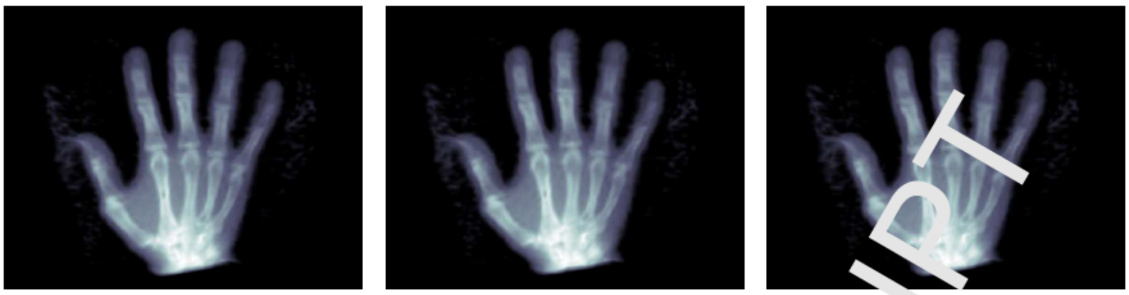
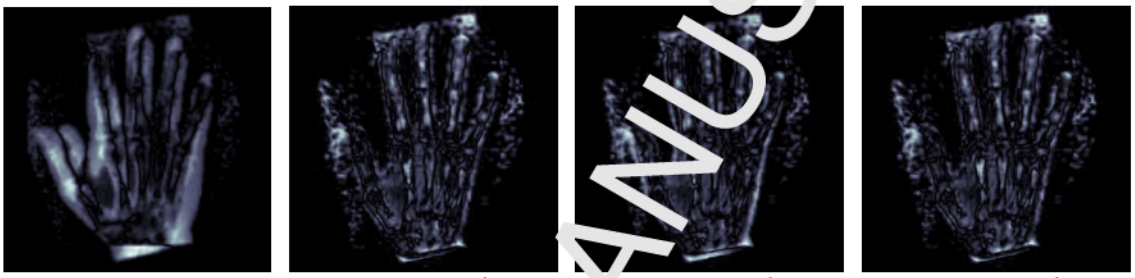

(a) $|T - R|$ (b) **M1** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (c) **M2** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$ (d) **M3** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{10}$

Figure 9: Example 3 – Difference images corresponding to registrations of Fig.8. Image (a) shows the initial error between T and R, while images (b), (c), (d) show the final errors between $T_{\boldsymbol{u}}$ and R for the **M1**, our **M2** and our **M3** respectively.



(a) **M2** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{40}$ $Err = 0.82\%$ (b) **M3** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{40}$ $Err = 1.10\%$ (c) **M4** $T_{\boldsymbol{u}}$ $\alpha = \frac{1}{40}$ $Err = 0.74\%$

Figure 10: Example **3** – Registration of 3(c) R and 3(f) T of size 512×512 by 3 methods. Image (a) shows the deformed template image $T_{\boldsymbol{u}}$ obtained using the **M2**, while image (b) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M3** and image (c) shows the deformed template image $T_{\boldsymbol{u}}$ for our **M4** for the 'bad' parameter value $\alpha = \frac{1}{40}$.



(a) Original $|T - R|$ (b) **M2** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{40}$ (c) **M3** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{40}$ (d) **M4** $|T_{\boldsymbol{u}} - R|$ $\alpha = \frac{1}{40}$

Figure 11: Example **3** – Difference images corresponding to registrations of Fig.10. Image (a) shows the initial error between T and R, while images (b), (c), (d) show the final errors between $T_{\boldsymbol{u}}$ and R for our **M2**, **M3** and **M4** respectively.

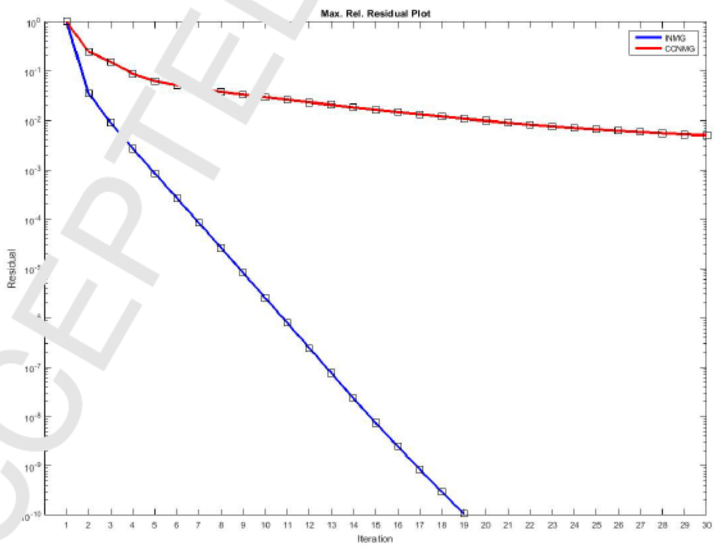
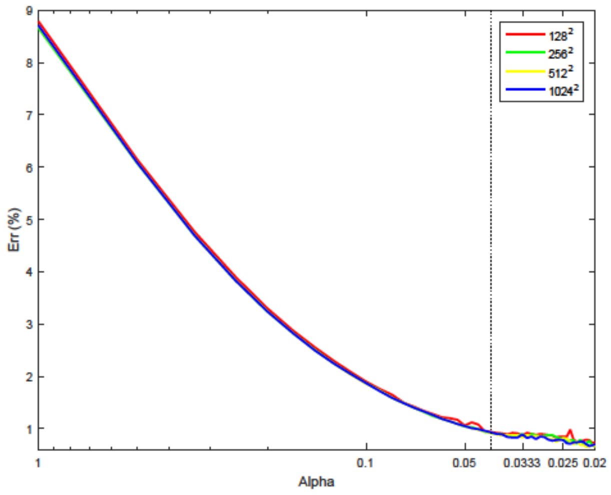
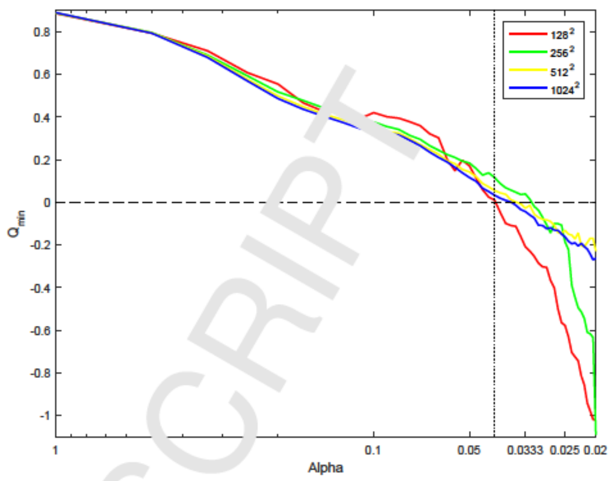


Figure 12: Comparison of the number of NMG cycles required for the maximum relative residual to reach a tolerance of 10^{-10} between our **M2** method and the **M1** method

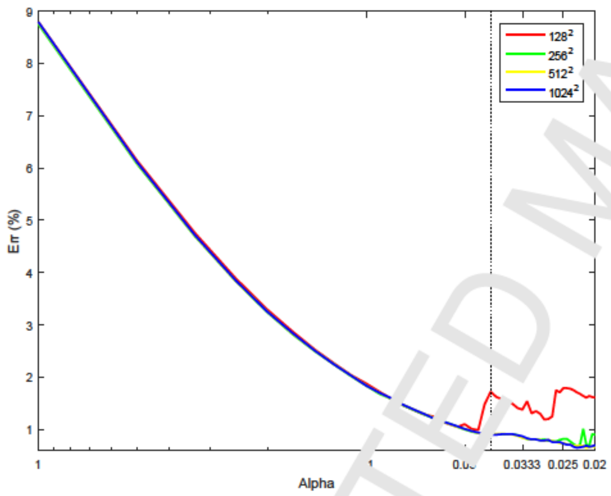


(a) Plot of relative error vs parameter α of model **M2** for Example 1

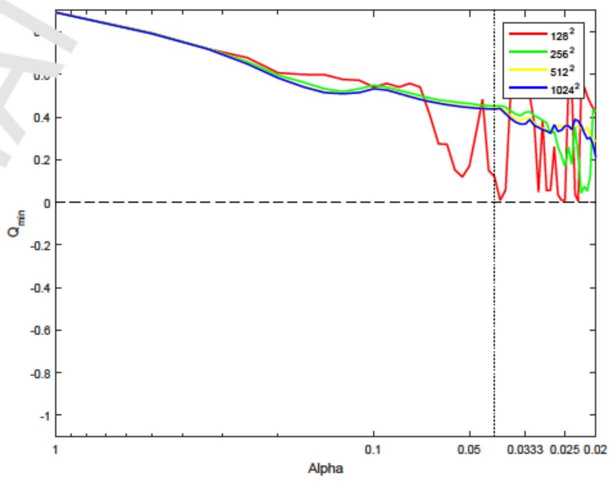


(b) Plot of minimum determinant value vs parameter α of model **M2** for Example 1

Figure 13: Test of robustness of model **M2** to the choice of parameter α (50 values).



(a) Plot of relative error vs parameter α of model **M4** for Example 1



(b) Plot of minimum determinant value vs parameter α of model **M4** for Example 1

Figure 14: Test of robustness of model **M4** to the choice of parameter α (50 values).

Image Size n^2	Image Example	α	**M1**		**M2**		**M3**		**M4**	
			CPU (s)	Ratio	CPU (s)	Ratio	CPU (s)	Ratio	CPU (s)	Ratio
128^2	Example 1 (CT)	$\frac{1}{10}$	0.316	–	0.425	–	0.437	–	0.452	–
256^2			6.887	21.794	1.090	2.565	1.164	2.666	1.304	2.885
512^2			17.204	2.498	5.057	4.639	5.250	4.510	6.202	4.756
1024^2			180.785	10.508	22.972	4.543	24.182	4.606	29.072	4.688
128^2	Example 2 (CT)	$\frac{1}{10}$	0.456	–	0.636	–	0.715	–	0.831	–
256^2			2.038	4.469	2.879	4.527	6.251	8.743	3.874	4.662
512^2			34.047	16.706	14.244	4.948	14.784	2.365	18.768	4.845
1024^2			195.431	5.740	68.196	4.788	71.186	4.815	87.203	4.646
128^2	Example 3 (Hand)	$\frac{1}{10}$	0.959	–	0.868	–	0.892	–	0.845	–
256^2			6.787	7.077	1.984	2.286	2.118	2.374	2.582	3.059
512^2			42.149	6.210	9.350	4.713	9.706	4.089	12.340	4.779
1024^2			195.403	4.636	45.620	4.879	48.026	4.948	58.466	4.738

Table 7: Test on optimal complexity in CPU time ratio for 4 NMG methods. The optimal ratio is 4 for an $O(N)$ method (with $N = n^2$). Clearly the newer NMGs are better.

References

[1] R. Bajscy and S. Kovačič. Multiresolution elastic matching. *Comp. Vision Graph.*, 46(1):1–21, 1989.

[2] S.M. Bartlett. An inverse matrix adjustment arising in discriminant analysis. *Ann. Math. Statist.*, 22(1):107–111, 1951.

[3] M. Bazargani, A. Anjos, F. G. Lobo, A. Mollahosseini, and H. R. Shahbazkia. Affine image registration transformation estimation using a real coded genetic algorithm with SBX. *CoRR*, abs/1204.2139, 2012.

[4] V. Boldea, G.C. Sharp, S.B. Jiang, and D. Sarrut. 4d-ct lung motion estimation with deformable registration: Quantification of motion nonlinearity and hysteresis. *Medical Physics*, 35(3):1008–1018, 2008.

[5] W.L Briggs, V.E. Henson, and S.F. McCormick. *A Multigrid Tutorial: Second Edition*. SIAM publications, 2000.

[6] C. Broit. *Optimal registration of deformed images*. PhD thesis, University of Pennsylvania, 1981.

[7] T. Brox, C. Bregler, and J. Malik. Large displacement optical flow. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2009.

[8] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr. Real-time optic flow computation with variational methods. *Computer Analysis of Images and Patterns*, 2756:222–229, 2003.

[9] A. Bruhn, J. Weickert, C. Feddern, T. Kohlberger, and C. Schnörr. Variational optic flow computation in real-time. *IEEE Transactions on Image Processing*, 14:608–615, 2006.

[10] A. Bruhn, J. Weickert, and C. Schnörr. Lucas/kanade meets horn/schunck: combining local and global optic flow methods. *International Journal of Computer Vision*, 61(3):211–231, 2005.

[11] M. Burger, J. Modersitzki, and L. Ruthotto. A hyperelastic regularization energy for image registration. *SIAM Journal on Scientific Computing*, 35(1):B132–B148, 2013.

[12] K. Cao, G.E. Christensen, K. Ding, K. Du, M.L. Raghavan, R.E. Amelon, K.M. Baker, E.A. Hoffman, and J.M. Reinhardt. Tracking regional tissue volume and function change in lung using image registration. *Journal of Biomedical Imaging*, 2012.

[13] E. Castillo, R. Castillo, Y. Zhang, and T. Guerro. Compressible image registration for thoraci computed tomography images. *Journal of Med. Biol. Eng.*, 29(5):222–233, 2009.

[14] G.E. Christensen, J.H. Song, W. Lu, I. El Naqa, and D.A. Low. Tracking lung tissue motion and expansion/compression with inverse consistent image registration and spirometry. *Medical Physics*, 34(6):2155–2163, 2007.

[15] N. Chumchob and K. Chen. A robust affine image registration method. *International Journal of Numerical Analysis and Modelling*, 6(2):311–334, 2009.

[16] N. Chumchob and K. Chen. A robust multigrid approach for variational image registration models. *Journal of Computational and Applied Mathematics*, 236(5):653–674, 2011.

[17] N. Chumchob and K. Chen. An improved variational image registration model and a fast algorithm for its numerical approximation. *Numerical Methods for Partial Differential Equations*, 28(6):1866–1995, 2012.

[18] N. Chumchob, K. Chen, and C. Brito-Loeza. A fourth order variational image registration model and its fast multigrid algorithm. *Multiscale Moddeling and Simulation*, 9(1):89–128, 2010.

[19] S. Dawn, V. Saxena, and B. Sharma. Remote sensing image registration techniques: a survey. In *Proceedings of the 4th International Conference on Image and Signal Processing*, pages 103–112. Springer-Verlag, 2010.

[20] C. Frohn-Schauf, S. Henn, L.Hömke, and K. Witsch. Total variation based image registration. In *International Conference on PDE-Based Image Processing and Related Inverse Problems Series: Mathematics and Visualization*, pages 305–323. Springer Verlag, 2006.

[21] C. Frohn-Schauf, S. Henn, and K. Witsch. Multigrid based total variation image registration. *Computing and Visualization in Science*, 11(2):101–113, 2008.

[22] J.C. Gee and K. Bajcsy. Elastic matching: continuum mechanical and probabilistic analysis. In *Brain Warping*, pages 18–3. Academic Press, 2000.

[23] V. Gorbunova, P. Lo, H. Ashraf, A. Dirksen, M. Nielsen, and M. de Bruijne. *Weight preserving image registration for monitoring disease progression in lung CT*, pages 863–870. Springer Berlin Heidelberg, 2008.

[24] V. Gorbunova, J. Sporring, P. Lo, M. Loeve, H.A. Tiddens, M. Nielsen, A. Dirksen, and M. de Bruijne. Mass preserving image registration for lung ct. *Medical Image Analysis*, 16(4):786–795, 2012.

[25] E. Haber and J. Modersitzki. A multilevel method for image registration. *SIAM J. Sci. Comput.*, 27(5):1594–1607, 2006.

[26] P.W. Hemker. On the order of prolongations and restrictions in multigrid procedures. *JCAM*, 32(3):423–429, 1990.

[27] S. Henn. A multigrid method for a fourth-order diffusion equation with application to image processing. *SIAM Journal on Scientific Computing*, 27(3):831–849, 2005.

[28] S. Henn and K. Witsch. Iterative multigrid regularization techniques for image matching. *SIAM Journal on Scientific Computing*, 23(4):1077–1093, 2001.

[29] L. Hömke. A multigrid method for anisotropic pdes in elastic image registration. *Numerical Linear Algebra with Applications*, 13:215–229, 2006.

[30] B.K.P. Horn and B.G.Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[31] H. J. Johnson and G. E. Christensen. Consistent landmark and intensity-based image registration. *IEEE Transactions on Medical Imaging*, 21(5):450–461, 2002.

[32] H. Köstler, K. Ruhanu, and R. Wienands. Multigrid solution of the optical flow system using a combined diffusion- and curvature-based regularizer. *Numer. Linear Algebra Appl.*, 15:201–218, 2008.

[33] T. Lin, C. Le Guyader, I.D. Dinov, P.M. Thompson, A.W. Toga, and L.A. Vese. Gene expression data to mouse atlas registration using a nonlinear elasticity smoother and landmark points constraints. *J. Sci. Comput.*, 50:586–609, 2012.

[34] J. Modersitzki. Numerical methods for image registration. *Oxford University Press*, 2004.

[35] T. Pock, M. Urschler, C. Zach, R. Beichel, and H. Bischof. A duality based algorithm for tv-l^1-optical-flow image registration. *LNCS*, 4792:511–518, 2007.

[36] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1):259–268, 1992.

[37] M. Stürmer and H. Köstler, U. Rüde. A fast multigrid solver for applications in image processing. *Numer. Linear Algebra with Appl.*, 15:187–200, 2008.

[38] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, 2001.

[39] W. Zhou, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image qualifty assessment:from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13:600–612, 2004.

[40] D. Zikic, W. Wein, A. Khamene, D.A. Clevert, and N. Navab. *Fast deformable registration of 3D-ultrasound data using a variational approach*, pages 915–923. Springer Berlin Heidelberg, 2006.

A Optimised version of Algorithm 3

In our constrained NMG, we check to see whether the constraint in (4.4) has been satisfied after the final post-smoothing step and solver step. While checking the constraint after the coarsest solver step is inexpensive computationally owing to the very small grid size, this is not the case when checking after the post-smoothing step. For each interior point Algorithm 3 needs to solve eight inverse problems which, even though we are only using 3×3 matrices, become very expensive on larger grids thus leading to a significant increase in CPU time. We will now look to exploit the structure and commonality between different interior points, of the matrices A_l, to create an optimised version of Algorithm 3. First we will look at the relation of the matrices A_l at the first interior point $(2,2)$ and a general interior point (i,j). Looking at the matrix A_1, we see that

$$\underline{\text{At } (2,2)}:\ A_1 = \begin{pmatrix} 1 & h & h \\ 1 & 2h & h \\ 1 & h & 2h \end{pmatrix},\quad \underline{\text{At } (i,j)}:\ \tilde{A}_1 = \begin{pmatrix} 1 & (i-1)h & (j-1)h \\ 1 & ih & (j-1)h \\ 1 & (i-1)h & jh \end{pmatrix}$$

since $\big((x_1)_2,(x_2)_2\big) = (h,h)$ and $\big((x_1)_i,(x_2)_j\big) = \big((i-1)h,(j-1)h\big)$, then $\tilde{A}_1$ can be written in the following way

$$\tilde{A}_1 = \begin{pmatrix} 1 & (x_1)_2 + (i-1)h & (x_2)_2 + (j-1)h \\ 1 & (x_1)_3 + (i-1)h & (x_2)_2 + (j-1)h \\ 1 & (x_1)_2 + (i-1)h & (x_2)_3 + (j-1)h \end{pmatrix} = A_1 + \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \big(0,(i-1)h,(j-1)h\big)$$

$$= A_1 + \boldsymbol{p}\boldsymbol{q}^T \tag{A.1}$$

with $\boldsymbol{p} = (1,1,1)^T$, $\boldsymbol{q} = \big(0,(i-1)h,(j-1)h\big)^T$. The matrices $\tilde{A}_l$ for the remaining triangles can be written in similar ways to (A.1), then we have

$$\tilde{A}_l = A_l + \boldsymbol{p}\boldsymbol{q}^T \tag{A.2}$$

with $\boldsymbol{p}$, $\boldsymbol{q}$ as before, and so the inverse $\tilde{A}_l^{-1} = (A_l + \boldsymbol{p}\boldsymbol{q}^T)^{-1}$, at a general discrete interior point, can be computed using the Sherman-Morrison formula [2] given by the following theorem

Theorem A.1. *(Sherman-Morrison) Suppose $A \in \mathbb{R}^{n\times n}$ is an invertible matrix, and $\boldsymbol{p}$, $\boldsymbol{q} \in \mathbb{R}^{n\times 1}$ are column vectors. Then $\big(A + \boldsymbol{p}\boldsymbol{q}^T\big)$ is invertible $\iff$ $1 + \boldsymbol{q}^T A^{-1}\boldsymbol{p} \neq 0$. If $\big(A + \boldsymbol{p}\boldsymbol{q}^T\big)$ is invertible, then its inverse is given by*

$$\big(A + \boldsymbol{p}\boldsymbol{q}^T\big)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{p}\boldsymbol{q}^T A^{-1}}{1 + \boldsymbol{q}^T A^{-1}\boldsymbol{p}} \tag{A.3}$$

where $\boldsymbol{p}\boldsymbol{q}^T$ denotes the outer product of the vectors $\boldsymbol{p}$, $\boldsymbol{q}$.

It can be shown that the multiplication $\boldsymbol{q}^T A_l^{-1}\boldsymbol{p} = 0\ \forall\ l = 1,\ldots,4$, therefore the invertibility condition from Theorem A.1 holds for every interior (i,j) for $i,\ j = 2,\ldots,n-1$ and thus the matrices $(A_l + \boldsymbol{p}\boldsymbol{q}^T)^{-1}$ are invertible for each $l = 1,\ldots,4$. Then we can use Theorem A.1 to rewrite the inverses $(A_l + \boldsymbol{p}\boldsymbol{q}^T)^{-1}$ as

$$\big(A_l + \boldsymbol{p}\boldsymbol{q}^T\big)^{-1} = A_l^{-1} - \frac{A_l^{-1}\boldsymbol{p}\boldsymbol{q}^T A_l^{-1}}{1 + \boldsymbol{q}^T A_l^{-1}\boldsymbol{p}}\,. \tag{A.4}$$

Next we use the fact that we need only determine the $b_{l\,u_m}$, $c_{l\,u_m}$ coefficients where $m = 1,2$, and so our original inverse problem (4.7) reduces to the following scalar equations

$$\begin{aligned} b_{l\,u_1} &= \omega_{u_1\,l}(2) - \mu_l\omega_{u_1\,l}(2), & c_{l\,u_1} &= \omega_{u_1\,l}(3) - \mu_l\omega_{u_1\,l}(3), \\ b_{l\,u_2} &= \omega_{u_2\,l}(2) - \mu_l\omega_{u_2\,l}(2), & c_{l\,u_2} &= \omega_{u_2\,l}(3) - \mu_l\omega_{u_2\,l}(3), \end{aligned} \tag{A.5}$$

where $\mu_l = \frac{(\omega_{p\,l}(2)q_2 + \omega_{p\,l}(3)q_3)}{1+(\omega_{p\,l}(2)q_2 + \omega_{p\,l}(3)q_3)}$ and $\omega_{p\,l}(2)$, $\omega_{p\,l}(3)$, q_2, q_3, $\omega_{u_m\,l}(2)$, $\omega_{u_m\,l}(3)$ denote the second and third components of $\omega_{p\,l} = A_l^{-1}\boldsymbol{p}$, $\boldsymbol{q}^T$ and $\omega_{u_m\,l} = A_l^{-1}v_{m\,l}$ respectively.

Therefore the key message is that per checking step across the entire grid only simple matrix-vector products are needed, if we invert matrices A_l^{-1} at the first pixel and then re-use them. Hence our optimised version of Algorithm 3 can be expressed by the following

Algorithm 8 $Q_{min} = FEMOpt(\boldsymbol{u}^h, n, h)$

```
1: for $l = 1, \ldots, 4$ do
       Compute matrices $A_l$ corresponding to first interior point $(2, 2)$
       Compute inverse matrices $A_l^{-1}$
       Compute second and third components of $A_l^{-1}\boldsymbol{p} \rightarrow \omega_{p\,l}(2)$, $\omega_{p\,l}(3)$
2: end for
3: for $i = 2, \ldots, n-1$ do
4:     for $j = 2, \ldots, n-1$ do
       Compute second and third components of $\boldsymbol{q}^T \rightarrow q_2 = (i-1)h$, $q_3 = (j-1)h$
5:         for $l = 1, \ldots, 4$ do
       Compute $\mu_l$
       Compute second and third components of $\omega_{u_1\,l}$, $\omega_{u_2\,l} \rightarrow \omega_{u_1\,l}(2)$, $\omega_{u_1\,l}(3)$, $\omega_{u_2\,l}(2)$, $\omega_{u_2\,l}(3)$
       Determine coefficients $b_{l\,u_1}$, $c_{l\,u_1}$, $b_{l\,u_2}$, $c_{l\,u_2}$ using (A.5)
       Compute determinant for triangle $T_l \rightarrow \tilde{Q}_l = (1 + b_{l\,u_1})(1 + c_{l\,u_2}) - c_{l\,u_1} b_{l\,u_2}$
6:         end for
       Assign minimum $\tilde{Q}$ to be entry $(Q_{ij}) \rightarrow (Q_{ij}) = \min\{\tilde{Q}_1, \ldots, \tilde{Q}_4\}$
7:     end for
8: end for
       Take minimum entry in $\boldsymbol{Q}$ to be minimum determinant value $\rightarrow Q_{min} = \min\{\boldsymbol{Q}\}$
```

Finally we show in Table 8 how much speed up can be achieved for a simple example. Clearly Algorithm 8 uses up to 30 times less CPU than Algorithm 3.

Image Size n	Unoptimised Time (s)	Optimised Time (s)
256^2	4.46	0.17
512^2	17.87	0.61
1024^2	71.53	2.40
2048^2	306.23	9.90

Table 8: Table showing the comparison of CPU times per iteration between old unoptimised FEM code and new optimised FEM code.