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

\[
T \circ \varphi(x) = T(\varphi(x)) \approx R(x) \text{ for } x = (x_1, \ldots, x_d)^T \in \Omega \subset \mathbb{R}^d.
\]

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

\[
\varphi \equiv \varphi(x) = x + u(x)
\]

where \( u \equiv u(x) = (u_1(x), \ldots, u_d(x))^T \) denotes the displacement field. Then the problem of determining \( \varphi \) is the same as finding \( u \). 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 \( \Omega \) given by the unit square, that is \( \Omega = [0, 1]^2 \subset \mathbb{R}^2 \). In order to determine \( u \), the variational minimisation problem will take the following form

\[
\min_u E(u) = D(R, T, u) + \alpha \mathcal{R}(u)
\]

where in the energy functional \( D(R, T, u) \) is a distance measure, \( \mathcal{R}(u) \) 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

\[
D(R, T, u) = \frac{1}{2} \int_\Omega \left( T(x + u) - R(x) \right)^2 d\Omega.
\]

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 shall 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)** \[20, 21, 35, 36\]:

\[
\mathcal{R}^{TV}(u) = \sum_{s=1}^2 \int_\Omega \left| \nabla u_s \right| d\Omega \quad \text{where } | \cdot | \text{ denotes the Euclidean}
\]
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

$$E^{\text{Diff}}(u) = \frac{1}{2} \int_{\Omega} \sum_{s=1}^{2} |\nabla u_s|^2 d\Omega.$$  

(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

$$\min_u E^{\text{Diff}}(u) = D(R, T, u) + \alpha E^{\text{Diff}}(u) = \frac{1}{2} \int_{\Omega} (T_u - R)^2 + \alpha \sum_{s=1}^{2} |\nabla u_s|^2 d\Omega$$  

(2.2)

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

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

(2.3)

which eventually result in the following integrals

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

(2.4)

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

$$-\alpha \Delta u_m + F_m(u) = 0$$  

(2.5)

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

$$F_m(u) = \partial_{u_m} T_u (T_u - R)$$  

(2.6)

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

norm;

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

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

\[
(\partial_{u_1} T_u)_{i,j} \approx \frac{1}{2h} \left( (T_u)_{i+1,j} - (T_u)_{i-1,j} \right),
\]

\[
(\partial_{u_2} T_u)_{i,j} \approx \frac{1}{2h} \left( (T_u)_{i,j+1} - (T_u)_{i,j-1} \right)
\]

\[
(\Delta u_m)_{i,j} \approx \frac{1}{h^2} \left( (u_m)_{i,j-1} + (u_m)_{i-1,j} - 4(u_m)_{i,j} + (u_m)_{i+1,j} + (u_m)_{i,j+1} \right)
\]

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

\[
-\alpha (\Delta u_m)_{i,j} + (F_m(u))_{i,j} = 0
\]

with

\[
(F_m(u))_{i,j} = (\partial_{u_m} T_u)_{i,j} \left( (T_u)_{i,j} - (R)_{i,j} \right)
\]

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 lexicographical 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

\[
-\alpha (\Delta u_m)_k + (F_m(u))_k = 0
\]

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

\[
-\alpha (\Delta u_m)_{k}^{(l+1)} + (F_m(u))_{k}^{(l+1)} = 0
\]

where

\[
(F_1(u))_{k}^{(l+1)} = \left( \partial_{u_1} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k \left( (T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)}))_k - (R(x_1, x_2))_k \right)
\]

\[
(F_2(u))_{k}^{(l+1)} = \left( \partial_{u_2} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k \left( (T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)}))_k - (R(x_1, x_2))_k \right).
\]

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

\[
(T(x_1 + u_1^{(l+1)}, x_2 + u_2^{(l)}))_k \approx \left( T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k + \left( (u_1)_k^{(l+1)} - (u_1)_k^{(l)} \right) \left( \partial_{u_1} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k
\]

\[
(T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}))_k \approx \left( T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k + \left( (u_2)_k^{(l+1)} - (u_2)_k^{(l)} \right) \left( \partial_{u_2} T(x_1 + u_1^{(l)}, x_2 + u_2^{(l)}) \right)_k
\]

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

\[
-\alpha (\Delta u_m)_{k}^{(l+1)} + (\partial_{u_m} T_u)_{k}^{(l)} \left( (T_u)_{k}^{(l)} + \left( (u_m)_k^{(l+1)} - (u_m)_k^{(l)} \right) \left( \partial_{u_m} T_u \right)_k - (R)_k \right) = 0
\]
with \((T_u)_k^{(l)} \equiv \left(T(x + u^{(l)})_k \right)_k\) etc. for \(m = 1, 2\). Using the FD approximations (2.7), we can write (2.13) in the following way

\[
-\frac{\alpha}{h^2} \left( (u_m)_{k-n}^{(l+1)} + (u_m)_{k-1}^{(l+1)} \right) + \left( \left( \partial_{u_m} T_u \right)_k^{(l)} + \frac{4\alpha}{h^2} \right) (u_m)_k^{(l+1)} \\
-\frac{\alpha}{h^2} \left( (u_m)_{k+1}^{(l+1)} + (u_m)_{k+n}^{(l+1)} \right) = \left( \partial_{u_m} T_u \right)_k^{(l)} (u_m)_k^{(l)} - \left( \partial_{u_m} T_u \right)_k^{(l)} \left( (T_u)_k^{(l)} - (R)_k \right)
\]

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 \(\Omega\) is discretized 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. Chunichob-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 rate 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 finer 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 $\Omega^h$ denotes a fine grid and $\Omega^H$ a coarse grid with $h = \frac{1}{n-1}$. Let $H = 2h$. Also denote the system (2.10) by the operator notation on $\Omega^h$

\[ N^h(u^h) = G^h \]  

(2.15) with

\[ N^h = \begin{pmatrix} (N^h_1)_{ik} \\ (N^h_2)_{ik} \end{pmatrix}, u^h = \begin{pmatrix} (u^h_1)_{ik} \\ (u^h_2)_{ik} \end{pmatrix}, G^h = \begin{pmatrix} (g^h_1)_{ik} \\ (g^h_2)_{ik} \end{pmatrix}, \]  

(2.16)

and where $(N^h_1)_{ik} = (F_1(u^h))_{ik} - \alpha (\Delta^h u^h_1)_{ik}$, $(N^h_2)_{ik} = (F_2(u^h))_{ik} - \alpha (\Delta^h u^h_2)_{ik}$, $(g^h_1)_{ik} = (g^h_2)_{ik} = 0$, $k = 1, 2, \ldots, (n-2)^2$. The main steps of the FAS-NMG are as follows.

**Smoothing step.** Apply the iterative method (2.14) on grid $\Omega^h$ starting from some initial guess. This is the pre-smoothing step required to obtain a smooth approximation $u^h = (u^h_1, u^h_2)^T$ which has residual $r^h = G^h - N^h(u^h)$.

To improve this smooth approximation, it remains to compute the algebraic error (or the residual correction) $e^h = (e^h_1, e^h_2)^T = u^h - \bar{u}^h$ which cannot be computed directly on $\Omega^h$.

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

\[ N^H(\bar{u}^H) = N^H(\bar{u}^H + e^H) = r^H + N^H(\bar{u}^H) = G^H \]  

(2.17)

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

\[ R^H_h = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}^H_k \]  

(2.18)

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

\[ e^H = u^H - \bar{u}^H. \]  

(2.19)

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

\[ e^h = I^h_h e^H, \quad I^h_h = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}^h \]  

(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 $u^h = u^h + e^h$. 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 smooth, thus resulting in our final solutions $u^h$. 


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

1. $m_R + m_I \geq M$;  
2. $m_I \geq M$ and $m_R \geq 0$;  
3. $m_R \geq M$ 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 $u_h^{(k+1)} \leftarrow \text{FAS-NMG}(R_h^k, T_h, u_h^k, \alpha, \nu_1, \nu_2)$

1. Pre-smoothing step by performing $\nu_1$ steps (relaxation sweeps) $\bar{u}_h^{(k)} \leftarrow \text{Smooth}(R_h^k, T_h, u_h^{(k)}, G_h^k, \alpha, \nu_1)$

2. Coarse-grid correction
   
   Compute the residual $r_h^{(k)} = \nabla h - N_h(u_h^{(k)})$
   
   Restrict residual and smooth approximations $r_h^{(k)} = R_h^k r_h^{(k)}$, $u_h^{(k)} = R_h^k u_h^{(k)}$
   
   Set level $\rightarrow$ level $- 1$, $H = 2h$, $nc = \frac{H}{2}$
   
   Form RHS of coarse grid PDEs $G^H = R^H(u_h^{(k)})$
   
   Solve residual equation on coarse grid to obtain approximations $u_h^{(k)}$

3. if $\text{level} = 1$ then
   
   Solve to obtain solutions $u_h^{(k)}$ to high accuracy using a coarse grid solver.

4. else $\text{level} > 1$ Repeat the FAS-NMG procedure recursively to the next level i.e.
   
   $\bar{u}_h^{(k)} \leftarrow \text{FAS-NMG}(R_h^k, T_h^k, \bar{u}_h^{(k)}, G_h^k, \alpha, \nu_1, \nu_2)$

5. end if

6. Post-smoothing step by performing $\nu_2$ steps (relaxation sweeps) $u_h^{(k+1)} \leftarrow \text{Smooth}(R_h^k, T_h^k, \bar{u}_h^{(k)}, G_h^k, \alpha, \nu_2)$

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 $u_m^{(k+1)} = \frac{1}{2} \sum_{s=1}^{2} \left[ I - 2\tau \alpha L_{x_s} \right]^{-1} \left( u_m^{(k+1)} + \tau g_m - \tau F_m(u) \right)$ where $I$ denotes the identity operator, $\tau > 0$ the time-step, $g_m$ the RHS coming from the NMG framework, $F_m(u)$ the force terms given in (2.6) for $m = 1, 2$ and $L_{x_s} = \partial_{x_s}$ 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

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

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 $\frac{1}{25}$, 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 $\frac{1}{10}$ 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 Chunchob-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 \( \mu \) 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

\[
N^h u^h_{\text{new}} + N^h u^h_{\text{old}} = G^h
\]  

(3.1)

where \( u^h_{\text{new}}, u^h_{\text{old}} \) denote the current and previous approximations of \( u^h \) respectively, and

\[
N^h_{+} = \begin{pmatrix}
-\alpha \mathcal{L}^h_{+} & 0 \\
0 & -\alpha \mathcal{L}^h_{+}
\end{pmatrix}, \quad N^h_{0} = \begin{pmatrix}
-\alpha \mathcal{L}^h_{0} + \sigma_{11}^h & \sigma_{12}^h \\
\sigma_{12}^h & -\alpha \mathcal{L}^h_{0} + \sigma_{22}^h
\end{pmatrix},
\]

\[
N^h_{-} = \begin{pmatrix}
-\alpha \mathcal{L}^h_{-} & 0 \\
0 & -\alpha \mathcal{L}^h_{-}
\end{pmatrix}, \quad G^h = \begin{pmatrix}
g_{1}^h - F_{1}^h \\
g_{2}^h - F_{2}^h
\end{pmatrix}
\]  

(3.2)

with \( \sigma_{pq} = \partial_{u_p} T^h_{u} \partial_{u_q} T^h_{u} \), \( g_{mh}^{\text{h}} \) denote the RHS coming from the NMG scheme, \( F_{mh}^{\text{h}} \) are the discrete force terms as given in (2.9) and where \( \mathcal{L}^h_{+}, \mathcal{L}^h_{0}, \mathcal{L}^h_{-} \) define the following stencils

\[
\mathcal{L}^h_{+} = \frac{1}{h^2} \begin{pmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0
\end{pmatrix}, \quad \mathcal{L}^h_{0} = \frac{1}{h^2} \begin{pmatrix}
0 & 0 & 0 \\
0 & -4 & 0 \\
0 & 0 & 0
\end{pmatrix}, \quad \mathcal{L}^h_{-} = \frac{1}{h^2} \begin{pmatrix}
0 & 1 & 0 \\
0 & 0 & 1 \\
0 & 0 & 0
\end{pmatrix}.
\]  

(3.3)

for \( p, q, m = 1, 2 \). The smoothing rate in [16] was then calculated on a 32 \( \times \) 32 grid after a total of 5 outer and 5 inner iteration loops had been performed, thus resulting in an average smoothing rate of \( \mu_{\text{avg}} \approx 0.5 \) when taking \( \alpha = \frac{1}{16} \). However, in the analysis of [16] we notice that the \((u_m)_{k}^{\text{h}}\) terms, which result from the linearisation of the SSD term, where not included in the smoothing rate calculation. This omission meant 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 \((u_m)_{k}^{\text{h}}\) 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

\[
N^h u^h + M^h u^h = G^h
\]  

(3.4)
where $G^h$ is as in (3.2), and

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

(3.5)

using the following representation of the discrete Laplace operator $\Delta^h \equiv L^h_+ + L^h_0 + L^h_-$, with $L^h_+, L^h_0, L^h_-$ as defined in (3.3), then we can express (3.4) in the following way

$$
\mathcal{N}^h u_{new}^h + \mathcal{N}^h u_{new}^h + \mathcal{M}^h u_{old}^h = G^h
$$

(3.6)

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

$$
[\mathcal{N}^h_+ + \mathcal{N}^h_0] e_{new}^h = -[\mathcal{N}^h_- + \mathcal{M}^h] e_{old}^h
$$

(3.7)

where $\mathcal{N}^h_+, \mathcal{N}^h_0, \mathcal{N}^h_-$ are as defined in (3.2) and

$$
e_{new}^h = (e_{1 new}^h, e_{2 new}^h)^T, \quad e_{old}^h = (e_{1 old}^h, e_{2 old}^h)^T.
$$

(3.8)

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

$$
[\hat{\mathcal{N}}^h_+ + \hat{\mathcal{N}}^h_0] \psi_{new} \exp \left( \frac{2i\theta_1 \pi}{n} + \frac{2i\theta_2 \pi}{n} \right) = -[\hat{\mathcal{N}}^h_- + \hat{\mathcal{M}}^h] \psi_{old} \exp \left( \frac{2i\theta_1 \pi}{n} + \frac{2i\theta_2 \pi}{n} \right)
$$

(3.9)

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

$$
\mu_{\text{max}} = \max_{\text{loc}} \mu_{\text{loc}}, \quad \mu_{\text{loc}} \equiv \mu_{\text{loc}}(\theta) = \sup \left\{ \rho(\hat{\mathcal{S}}^h(\theta)) \left| \theta \in \Theta_{\text{high}} \right. \right\}
$$

(3.10)

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

$$
\hat{\mathcal{S}}^h(\theta) = -[\hat{\mathcal{N}}^h_+ + \hat{\mathcal{N}}^h_0]^{-1} [\hat{\mathcal{N}}^h_- + \hat{\mathcal{M}}^h(\theta)]
$$

(3.11)

with

$$
\hat{\mathcal{N}}^h_+ = \begin{pmatrix} -\frac{\alpha}{\pi} (e^{-i\omega_1} + e^{-i\omega_2}) & 0 \\ 0 & -\frac{\alpha}{\pi} (e^{-i\omega_1} + e^{-i\omega_2}) \end{pmatrix}, \quad \hat{\mathcal{N}}^h_0 = \begin{pmatrix} \frac{\omega_0}{\pi} + \sigma_{11}^h & 0 \\ 0 & \frac{\omega_0}{\pi} + \sigma_{22}^h \end{pmatrix}, \quad \hat{\mathcal{N}}^h_- = \begin{pmatrix} -\frac{\alpha}{\pi} (e^{i\omega_1} + e^{i\omega_2}) & 0 \\ 0 & -\frac{\alpha}{\pi} (e^{i\omega_1} + e^{i\omega_2}) \end{pmatrix}, \quad \hat{\mathcal{M}}^h = \begin{pmatrix} -\sigma_{11}^h & 0 \\ 0 & -\sigma_{22}^h \end{pmatrix}
$$

(3.12)

where $\omega_m = \frac{\omega_0}{\pi}$ 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 $\mu_{\text{avg}} \approx 0.69854$ and $\mu_{\text{max}} \approx 0.74762$ 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 $\mu_{\text{avg}}$ and $\mu_{\text{max}}$. 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 Chunshob-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

\[
-\frac{\alpha}{H^2} \left( (u_m)_{k-n}^{l+1} + (u_m)_{k-1}^{l+1} \right) + \left( \left( \partial_{u_m} T u \right)^2 \right)_k^{l+1} = \frac{4\alpha}{H^2} (u_m)_k^{l+1}
\]

\[
-\frac{\alpha}{H^2} \left( (u_m)_{k+1}^{l+1} + (u_m)_{k+n}^{l+1} \right) = \left( \partial_{u_m} T u \right)_{k}^{l} (u_m)_{k}^{l} - \left( \partial_{u_m} T u \right)_{k}^{l} \left( T u_{k}^{l} - (R)_{k} \right)
\]

(3.13)

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

\[
A_m = \begin{pmatrix}
A_{m_1} & I_1 & & & \\
I_1 & A_{m_2} & I_1 & & \\
& I_1 & A_{m_2} & I_1 & \\
& & & \ddots & \\
& & & & I_1
\end{pmatrix},
\]

\[
u_m = \begin{pmatrix}
(u_m)_{k_2(2)} \\
(u_m)_{k_2(j)} \\
\vdots \\
(u_m)_{k_2(n-2)}
\end{pmatrix},
\]

\[
f_m = \begin{pmatrix}
(f_m)_{k_2(2)} \\
(f_m)_{k_2(j)} \\
\vdots \\
(f_m)_{k_2(n-2)}
\end{pmatrix}
\]

(3.14)

where \( A_{m_1}, I_1 \in \mathbb{R}^{(n-2) \times (n-2)} \) are matrices with structure

\[
A_{m_1} = \begin{pmatrix}
(a_m)_{k_2(j)} & -\frac{\alpha}{H^2} & & & \\
-\frac{\alpha}{H^2} & & & & \\
& & \ddots & & \\
& & & & -\frac{\alpha}{H^2}
\end{pmatrix},
\]

\[
I_1 = -\frac{\alpha}{H^2} \begin{pmatrix}
1 & & & & \\
\vdots & & & & \\
& & \ddots & & \\
& & & & 1
\end{pmatrix}
\]

(3.15)

with \((a_m)_{k_2(j)} = \left( \partial_{u_m} T u \right)^2_{k_2(j)} + \frac{\alpha}{H^2}\) and where \( k_2(j) = (j-2)(n-1) + (i-1) \) denotes a general lexicographically ordered discrete point \((i, j)\), as shown in Figure 1. Also

\[
(f_m)_{k_2(j)} = \left( \partial_{u_m} T u \right)^2_{k_2(j)} (u_m)_{k_2(j)} - \left( \partial_{u_m} T u \right)_{k_2(j)} \left( T u_{k_2(j)} - (R)_{k_2(j)} \right)
\]

(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 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. \( 8 \times 8 \) mesh) as the rate is not sharp especially on a fine grid (e.g. \( 128 \times 128 \) mesh).

Analysis of the proposed coarsest grid solver. To estimate the convergence rate \( P \) 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 \( \hat{S}^H(\theta) \) 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 \( \Theta \) to only consider the high frequency range \( \Theta_{\text{high}} \) in the smoother analysis, now we consider \( \Theta \) over the entire Fourier domain \( \Theta \). Since our proposed direct solver is based upon the pointwise smoother shown in §2.2, the derivation of the amplification matrix \( \hat{S}^H(\theta) \) 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

\[
\mathcal{P}_{D_{\text{max}}} = \max_{\Theta_{\text{loc}}} \mathcal{P}_{D_{\text{loc}}}, \quad \mathcal{P}_{D_{\text{loc}}} = \mathcal{P}_{D_{\text{loc}}}(\theta) = \sup \left\{ \rho(\hat{S}^H(\theta)) : \theta \in \Theta \right\}
\]

(3.17)

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

\[
\hat{S}^H(\theta) = -\left[ \hat{N}^H_+ (\theta) + \hat{N}^H_0 (\theta) + \hat{N}^H_- (\theta) \right]^{-1} \hat{M}^H(\theta)
\]

with \( \hat{N}^H_+ (\theta), \hat{N}^H_0 (\theta), \hat{N}^H_- (\theta), \hat{M}^H(\theta) \) as in (3.12) and \( H = 2h \).
Algorithm 2 $u^{(l+1)}_H \leftarrow \text{DirectSolve}(R^H, T^H, u^{(k)}_H, G^H, \alpha, \text{IMAX}, Tol)$

1: Initialise $u^{(0)}_H = u^{(k)}_H$
   Construct discrete Laplacian parts of sparse matrices $A_m$
2: for $l = 1, \ldots, \text{IMAX}$ do
   Deform template image using $u^{(l)}_H \rightarrow T^H$
   Compute FD approximations for derivatives of $T^H \rightarrow \partial_{u^i} T^H$,
   Compute RHS $f_m$ (matrices) and then convert to column vectors $f_m$
   Add remaining diagonal parts to $A_m$
   Compute $u^{(l+1)}_H = A_m^{-1} f_m$
   Reshape $u^{(l+1)}_H$ to matrices $u^{(l+1)}_{m H}$
3: if $\|u^{(l+1)}_{1 H} - u^{(l)}_{1 H}\|_2 < Tol$ and $\|u^{(l+1)}_{2 H} - u^{(l)}_{2 H}\|_2 < Tol$ 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

$$B \Psi_\theta^{new} = C \Psi_\theta^{old}$$ (3.18)

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

$$B = \begin{pmatrix} B_1 & D \\ D & B_2 \end{pmatrix}, \quad C = \begin{pmatrix} C_1 & D \\ D & C_2 \end{pmatrix}, \quad \Psi_\theta = \begin{pmatrix} \Psi_\theta^1 \\ \Psi_\theta^2 \end{pmatrix}$$ (3.19)

with $B_m, C_m, D \in \mathbb{R}^{(n-2)^2 \times (n-2)^2}$ and $\Psi_\theta^* \in \mathbb{R}^{(n-2)^2 \times 1}$ given by

$$B_m = \begin{pmatrix} B_{m_1} & J_1 \\ J_2 & B_{m,_{-1}} \end{pmatrix}, \quad C = \begin{pmatrix} C_{m_2} & \cdots & C_{m,-1} \\ \cdots & \cdots & \cdots \\ C_{m,-2} & \cdots & C_{m,_{-1}} \end{pmatrix}, \quad D = \begin{pmatrix} D_2 & \cdots & D_{n-1} \end{pmatrix}, \quad \psi_\theta = \begin{pmatrix} (\psi_\theta)_1 \\ \vdots \\ (\psi_\theta)_n \end{pmatrix}$$ (3.20)

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

$$B_{m_1} = \begin{pmatrix} \frac{(b_m)_{k_1}(j)}{m_1} & -\frac{\omega m}{m_1}, e^{-\omega m} \\ -\frac{\omega m}{m_1}, e^{-\omega m} & \frac{(b_m)_{k_{n-1}}(j)}{m_1} \end{pmatrix}, \quad C_{m_2} = \begin{pmatrix} (c_m)_{k_2(j)} \\ \vdots \\ (c_m)_{k_{n-1}(j)} \end{pmatrix}, \quad D = \begin{pmatrix} (d)_{k_2(j)} \\ \vdots \\ (d)_{k_{n-1}(j)} \end{pmatrix}, \quad J_1 = \begin{pmatrix} \frac{\omega m}{m_1}, e^{-\omega m} \\ \vdots \\ \frac{\omega m}{m_1}, e^{-\omega m} \end{pmatrix}, \quad J_2 = \begin{pmatrix} \frac{\omega m}{m_1}, e^{-\omega m} \\ \vdots \\ \frac{\omega m}{m_1}, e^{-\omega m} \end{pmatrix}$$ (3.21)

with $(b_m)_{k_1(j)} = \left((\partial_{u^i} T^u)^2\right)_{k_1(j)} + \frac{2\theta}{\pi}, (c_m)_{k_1(j)} = \left((\partial_{u^i} T^u)^2\right)_{k_1(j)}$,

$$D = \begin{pmatrix} (d)_{k_2(j)} \\ \vdots \\ (d)_{k_{n-1}(j)} \end{pmatrix}, \quad J_1 = \begin{pmatrix} \frac{\omega m}{m_1}, e^{-\omega m} \\ \vdots \\ \frac{\omega m}{m_1}, e^{-\omega m} \end{pmatrix}, \quad J_2 = \begin{pmatrix} \frac{\omega m}{m_1}, e^{-\omega m} \\ \vdots \\ \frac{\omega m}{m_1}, e^{-\omega m} \end{pmatrix}$$ (3.21)

with $(b_m)_{k_1(j)} = \left((\partial_{u^i} T^u)^2\right)_{k_1(j)} + \frac{2\theta}{\pi}, (c_m)_{k_1(j)} = \left((\partial_{u^i} T^u)^2\right)_{k_1(j)}$,

$$\omega m = \frac{2\theta}{\pi} \text{ and } k_1(j) = (j-2)(n-1) + (i-1) \text{ for } m = 1, 2 \text{ and } i, j = 2, \ldots, n-1. \text{ Then the convergence rate } \mathcal{P}_B \text{ for the block formulation of our direct solver is estimated from the following}

$$\mathcal{P}_B = \sup \left\{ \rho(\hat{S}^H(\theta)) \mid \theta \in \Theta \right\}$$ (3.22)

with amplification matrix $\hat{S}^H(\theta) = B^{-1} C$. 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

$$\mathcal{N}_m^H u_{p_m}^H + \mathcal{M}_m^H u_{p_m}^H = \mathcal{G}_m^H$$

(3.23)

with

$$\mathcal{N}_m^H = \begin{pmatrix} 1 - 2\tau_\alpha \hat{\rho}_m x_m & 0 \\ 0 & 1 - 2\tau_\alpha \hat{\rho}_m x_m \end{pmatrix}, \mathcal{M}_m^H = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}, \mathcal{G}_m^H = \begin{pmatrix} \tau g_m^H - \tau F_1^H(u) \\ \tau g_2^H - \tau F_2^H(u) \end{pmatrix}$$

(3.24)

where $g_m^H$ are the discrete RHS coming from the NMG method and $F_m^H(u)$ are the discrete force terms given in (2.9). The $x_1, x_2$ directions of the discrete Laplace operator can be represented by $\hat{\rho}_m x_m = \mathcal{L}_m^H + \mathcal{L}_m^H + \mathcal{L}_m^H$, where $\mathcal{L}_m^H$, $\mathcal{L}_m^H$, $\mathcal{L}_m^H$ define the following stencils

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

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

(3.25)

then we can write (3.23) in the following way

$$\mathcal{N}_m^H u_{p_m}^H_{\text{new}} + \mathcal{M}_m^H u_{p_m}^H_{\text{old}} + \mathcal{N}_m^H u_{p_m}^H_{\text{new}} + \mathcal{M}_m^H u_{p_m}^H_{\text{old}} = \mathcal{G}_m^H$$

(3.26)

where $u_{p_m}^H_{\text{new}}, u_{p_m}^H_{\text{old}}$ denote the current and previous approximations of $u_{p_m}^H$ in the $x_m$ directions respectively, and

$$\mathcal{N}_{m+}^H = \begin{pmatrix} -2\tau_\alpha \mathcal{L}_{m+}^H & 0 \\ 0 & -2\tau_\alpha \mathcal{L}_{m+}^H \end{pmatrix}, \mathcal{N}_{m-}^H = \begin{pmatrix} -2\tau_\alpha \mathcal{L}_{m-}^H & 0 \\ 0 & -2\tau_\alpha \mathcal{L}_{m-}^H \end{pmatrix}$$

$$\mathcal{M}_{m0}^H = \begin{pmatrix} 1 - 2\tau_\alpha \mathcal{L}_{m0}^H & 0 \\ 0 & 1 - 2\tau_\alpha \mathcal{L}_{m0}^H \end{pmatrix}$$

(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

$$\mathcal{P}_{A_{\text{max}}} = \max_{\text{loc}} \mathcal{P}_{A_{\text{loc}}}, \quad \mathcal{P}_{A_{\text{loc}}} = \frac{1}{2} (\mathcal{P}_{A_{1\text{loc}}} + \mathcal{P}_{A_{2\text{loc}}})$$

$$\mathcal{P}_{A_{m\text{loc}}} = \mathcal{P}_{A_{m\text{loc}}} (\theta) = \sup \{ \rho(S_m^H(\theta)) | \theta \in \Theta \}$$

(3.28)

where $\rho(\cdot)$ again denotes the spectral radius, and $S_m^H(\theta)$ denote the amplification matrices given by

$$S_m^H(\theta) = -[\mathcal{N}_{m+}^H(\theta) + \mathcal{N}_{m0}^H(\theta)]^{-1} [\mathcal{N}_{m-}^H(\theta) + \mathcal{M}_{m0}^H(\theta)]$$

(3.29)

and where

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

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

(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

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

(3.31)
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

$$\det \nabla \varphi = (1 + u_{1x}) (1 + u_{2x}) - u_{1x} u_{2x}.$$  \hspace{1cm} (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

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

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 $\Omega^h$, that is we need to compute

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

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 $i, j = 2, \ldots, n-1$. 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 $\Omega^h$ 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 $T_1, \ldots, T_4$ 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

\[
\text{Triangle } T_1 : \begin{pmatrix} 1 & x_i & y_j \\ 1 & x_{i+1} & y_j \\ 1 & x_i & y_{j+1} \end{pmatrix} = \begin{pmatrix} a_{1_{u_1}} & b_{1_{u_1}} & c_{1_{u_1}} \\ a_{1_{u_2}} & b_{1_{u_2}} & c_{1_{u_2}} \\ a_{1_{u_3}} & b_{1_{u_3}} & c_{1_{u_3}} \end{pmatrix} \cdot \begin{pmatrix} 1 & x_i & y_j \\ 1 & x_{i+1} & y_j \\ 1 & x_i & y_{j+1} \end{pmatrix} + \begin{pmatrix} (u_{1_{x_1}})_{i,j} \\ (u_{1_{x_2}})_{i,j} \\ (u_{1_{x_3}})_{i,j} \end{pmatrix} = \begin{pmatrix} (u_{2_{x_1}})_{i,j} \\ (u_{2_{x_2}})_{i,j} \\ (u_{2_{x_3}})_{i,j} \end{pmatrix};
\]
we obtain similar systems for each of the remaining triangles $T_2, T_3$ and $T_4$. Then, to compute the coefficients $a_{l,m}, b_{l,m}, c_{l,m}$, we solve

$$s_l = A^{-1}_l v_{1,l}, \quad t_l = A^{-1}_l v_{2,l}$$

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, $A^{-1}_l$ are the inverses of the matrices corresponding to the edges of the triangles $T_l$ and $v_{ml} = (u_{m1}, u_{m2}, u_{m3})^T$ 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 $\Omega^h$, 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 $\alpha$ 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 $u$ can be penalised so heavily that the deformed template image $T_u$ 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 $\alpha$ if the constraint has not been satisfied within a small number of iterations. To construct this adaptive $\alpha$ 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 $\alpha$. This process can be summarised by Algorithm 7, and where the algorithm AdaptFASINMG is the same as Algorithm 6 except now Algorithm 5 is used to check the constraint instead of Algorithm 4.
4. Another advantage of the adaptive $\alpha$ scheme shown in Algorithm 7 is its robustness to the choice of parameter $\alpha$. Even if the initial $\alpha$ is set too small such that severe folding would normally occur, because we keep re-initialising the problem with new values of $\alpha$, we automatically find a pseudo-optimal $\alpha$ 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 $\alpha$ model by adaptive INMG.

Algorithm 3 $Q_{\text{min}} \leftarrow \text{Compute} Q(u^k_h, n, h)$

1: for $i = 2, \ldots, n - 1$ do
2:     for $j = 2, \ldots, n - 1$ do
3:         for $l = 1, \ldots, 4$ do
4:             Compute the vectors $s_l, t_l$ using (4.7)
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
7:         Assign minimum $\tilde{Q}$ to be entry $(Q_{ij}) \rightarrow (Q_{ij}) = \min \{\tilde{Q}_1, \ldots, \tilde{Q}_4\}$
8:     end for
9: Take minimum entry in $Q$ to be minimum determinant value $\rightarrow Q_{\text{min}} = \min \{Q\}$

Algorithm 4 $u_h^{(k+1)} \leftarrow \text{Constrain} U(u_h^{(k)}), h, \lambda, \text{LMAX}$

1: for $l = 1, \ldots, \text{LMAX}$ do
2:     Compute minimum value of determinant $Q_{\text{min}}$ using Algorithm 3
3:     if $Q_{\text{min}} > 0$ and $l \leq \text{LMAX}$ then
4:         Accept update $u_h^{(k)} = u_h^{(k)}$
5:     else if $Q_{\text{min}} \leq 0$ and $l < \text{LMAX}$ then
6:         Reject update and set $u_h^{(k)} = \lambda u_h^{(k)}$, $\lambda \in (0, 1)$
7:     else if $Q_{\text{min}} \leq 0$ and $l = \text{LMAX}$ then
8:         Error $\rightarrow$ Constraint failed
9: end if
10: end for

Algorithm 5 $[u_h^{(k+1)}, c, \text{done}_\alpha] \leftarrow \text{Adaptive} U(u_h^{(k)}), h, \lambda, \text{LMAX}$

1: Save current ‘good’ approximation $\rightarrow u_h^{(k)} = u_h^{(k)}$, $c = 0$
2: for $l = 1, \ldots, \text{LMAX}$ do
3:     Compute minimum value of determinant $Q_{\text{min}}$ using Algorithm 3
4:     if $Q_{\text{min}} > 0$ and $l \leq \text{LMAX}$ then
5:         Accept update $u_h^{(k+1)} = u_h^{(k)}$, $u_h^{(k)} = u_h^{(k)}$, $c = c + 1$, $\text{done}_\alpha = 1$, break
6:     else if $Q_{\text{min}} \leq 0$ and $l < \text{LMAX}$ then
7:         Reject update and set $u_h^{(k)} = \lambda u_h^{(k)}$, $\lambda \in (0, 1)$, $c = c + 1$
8:     else if $Q_{\text{min}} \leq 0$ and $l = \text{LMAX}$ then
9:         Reset to ‘good’ approximation $\rightarrow c = \text{LMAX}$, $u_h^{(k+1)} = u_h^{(k)}$, $\text{done}_\alpha = 0$
10: end if
11: end for
Algorithm 6 $u_h^{(k+1)} \leftarrow \text{ConstFASNMG}(R^h, T^h, n, h, level, u_h^{(k)}, G^h, \alpha, \nu_1, \nu_2)$

1: Pre-smoothing step by performing $\nu_1$ steps (relaxation sweeps) $\tilde{u}_h^k \leftarrow \text{Smooth}(R^h, T^h, u_h^{(k)}, G^h, \alpha, \nu_1)$
2: Coarse-grid correction
   Compute the residual $r_h^{(k)} = G^h - \nu^h(\tilde{u}_h^k)$
   Restrict residual and smooth approximations $r_H^{(k)} = R_H^h r_h^{(k)}$, $\tilde{u}_H^k = R_H^h \tilde{u}_h^k$
   Set level $\rightarrow$ level $- 1$, $H = 2h$, $nc = \frac{7}{2}$
   Form RHS of coarse grid PDEs $G^H = r^H + N^H(\tilde{u}_H^k)$
   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.
   $\tilde{u}_H^k \leftarrow \text{ConstFASNMG}(R^H, T^H, nc, H, level - 1, u_H^{(k)}, G^H, \alpha, \nu_1, \nu_2)$
6: end if
   Compute the correction $e_H^k = u_H^k - \tilde{u}_H^k$
   Interpolate the correction to next fine grid level $e_h^k = I_H e_H^k$
   Update current grid level approximations using correction $u_h^{(k)} = \tilde{u}_h^k + e_h^k$
7: Post-smoothing step by performing $\nu_2$ steps (relaxation sweeps) $u_h^{(k+1)} \leftarrow \text{Smooth}(R^h, T^h, u_h^{(k)}, G^h, \alpha, \nu_2)$
   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 $u_h^{(k+1)} \leftarrow \text{Adaptive}_\alpha \left( R^h, T^h, n, h, u_h^{(k)}, \alpha, i_{max}^\alpha \right)$

1: Set done_NMG $= 0$, done_alpha $= 0$
2: while done_NMG $\neq 1$ do
3: if $i^\alpha = i_{max}^\alpha$ then
   LMAX = 100
4: end if
5: while done_NMG $\neq 1$ do
6: Set previous ‘good’ approximation $\rightarrow u_h^{(k)} = u_h^k$
7: Perform FAS-NMG $\rightarrow \left[ u_h^{(k+1)}, c \right] \leftarrow \text{AdaptFASNMG} \left( R^h, T^h, n, j, level, u_h^{(k)}, G^h, \alpha, \nu_1, \nu_2 \right)$
8: if $c \leq \text{LMAX and done}_\alpha \neq 1$ then
   break
9: end if
10: if NMG convergence criteria satisfied then
   done_NMG = 1
11: end if
12: end while
13: if $c \leq \text{LMAX and done}_\alpha \neq 1$ then
14: Set $\alpha = 2\alpha$, $i^\alpha = i^\alpha + 1$, $u_h^{(k)} = \tilde{u}_h^k$
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 Chunchob-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 \( \text{Err} = \frac{\|T_u - R\|^2}{\|T_u\|^2} \). 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 \( \varepsilon_3 \) 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 \( T_u \) and final error images \( |T_u - R| \) when compared with those obtained from M1. The first two columns of Table 5 show several test results of varying resolutions and parameters \( \alpha \). Here, 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 \( \varepsilon_3 \) 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 \( \alpha \) value, but for the other two cases of \( \alpha \) 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 \( \alpha \) 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 \( \alpha = \frac{1}{16} \), we obtained \( \text{SSIM}/\text{Err} \) 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 \( \det V \varphi > 0 \) for every discrete interior point in \( \Omega \). 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 \( \det V \varphi \). 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 on 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.

19
Finally to give an indication of the convergence history of $\textbf{M}_1$ and $\textbf{M}_2$, we plot in Fig.12 the residuals for more NMG cycles. Evidently $\textbf{M}_2$ has faster convergence plot than $\textbf{M}_1$.

**Parameter robustness.** In the diffusion model, the weighting parameter $\alpha$ indicates how strongly we wish to enforce smoothness on the deformation from the regularisation term. Specifically, a larger value of $\alpha$ 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 $\alpha$ 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 $\alpha$ can be very time consuming as in general a pre-multigrid routine is usually required to find this ‘best’ $\alpha$ (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 $\alpha$ is less important. Here we will compare how the value of $\alpha$ impacts the relative error (denoted ‘$\text{Err}$’) and minimum determinant value (denoted $Q_{\text{min}}$) for models $\textbf{M}_2$ and $\textbf{M}_4$. From Figure 13(a) we see that as $\alpha$ gets smaller the error also decreases, however looking at Figure 13(b) we see that the value of $Q_{\text{min}}$ is also decreasing to a point where it is always negative as highlighted by the dotted line. This suggests that our model $\textbf{M}_2$ 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 $\textbf{M}_4$ model follows a similar pattern with regard to a decreasing error as $\alpha$ decreases like with our $\textbf{M}_2$ model, however from Figure 14(b) we see that our $\textbf{M}_4$ model always maintains the physical integrity of the deformation with $Q_{\text{min}} > 0$ for all tested values of $\alpha$. From this we can conclude that our adaptive $\alpha$ model $\textbf{M}_4$ is very robust to the initial value of $\alpha$, even for small values, while maintaining a consistently good registration accuracy in terms of error.

![Figure 3: Three Pairs of Test Images.](image)

### 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 $\det \nabla \varphi > 0$. 


Table 3: Example 2 – Regression 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 128², 256², 512², 1024² images respectively.

<table>
<thead>
<tr>
<th>Image Size n²</th>
<th>α</th>
<th>M1</th>
<th></th>
<th>M2</th>
<th></th>
<th>M3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128²</td>
<td>±</td>
<td>0.930/0.945/0.912/0.917</td>
<td></td>
<td>0.943/0.951/0.927/0.930</td>
<td></td>
<td>0.943/0.951/0.927/0.930</td>
<td></td>
</tr>
<tr>
<td>256²</td>
<td>±</td>
<td>0.959±0.04/22.387/0.854</td>
<td></td>
<td>0.954±0.43/29.426/0.801</td>
<td></td>
<td>0.954±0.43/29.426/0.801</td>
<td></td>
</tr>
<tr>
<td>512²</td>
<td>±</td>
<td>0.972±0.44/25.196/0.872</td>
<td></td>
<td>0.975±0.45/66.178/0.822</td>
<td></td>
<td>0.975±0.45/66.178/0.822</td>
<td></td>
</tr>
<tr>
<td>1024²</td>
<td>±</td>
<td>0.987±0.52/30.316/0.612</td>
<td></td>
<td>0.984±0.39/70.452/0.604</td>
<td></td>
<td>0.984±0.39/70.452/0.604</td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Example 2 – Regression comparison of 3 methods on multiple image sizes for different α values, with an initial relative error of 1.90% and initial SSIM values of 0.667, 0.704, 0.769, 0.838 for the 128², 256², 512², 1024² images respectively. Clearly although M2 does converge quickly, the M3 offers both speed and correct transforms.

<table>
<thead>
<tr>
<th>Image Size n²</th>
<th>α</th>
<th>M1</th>
<th></th>
<th>M2</th>
<th></th>
<th>M3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128²</td>
<td>±</td>
<td>0.751/0.803/0.612/0.640</td>
<td></td>
<td>0.786/1.14/3.292/0.645</td>
<td></td>
<td>0.786/1.14/3.292/0.645</td>
<td></td>
</tr>
<tr>
<td>256²</td>
<td>±</td>
<td>0.806/1.32/25.142/0.618</td>
<td></td>
<td>0.832/1.18/14.189/0.663</td>
<td></td>
<td>0.832/1.18/14.189/0.663</td>
<td></td>
</tr>
<tr>
<td>512²</td>
<td>±</td>
<td>0.896/1.34/25.169/0.640</td>
<td></td>
<td>0.906/1.20/46.103/0.701</td>
<td></td>
<td>0.906/1.20/46.103/0.701</td>
<td></td>
</tr>
<tr>
<td>1024²</td>
<td>±</td>
<td>0.867/1.11/7.203/0.344</td>
<td></td>
<td>0.803/0.91/3.287/0.032</td>
<td></td>
<td>0.803/0.91/3.287/0.032</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Example 3 – Regression 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 128², 256², 512², 1024² images respectively.

<table>
<thead>
<tr>
<th>Image Size n²</th>
<th>α</th>
<th>M1</th>
<th></th>
<th>M2</th>
<th></th>
<th>M3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128²</td>
<td>±</td>
<td>0.724/0.84/1.614/0.664</td>
<td></td>
<td>0.717/1.30/2.636/0.654</td>
<td></td>
<td>0.717/1.30/2.636/0.654</td>
<td></td>
</tr>
<tr>
<td>256²</td>
<td>±</td>
<td>0.734/0.61/7.017/0.701</td>
<td></td>
<td>0.725/3.24/2.195/0.517</td>
<td></td>
<td>0.725/3.24/2.195/0.517</td>
<td></td>
</tr>
<tr>
<td>512²</td>
<td>±</td>
<td>0.768/3.66/25.452/0.717</td>
<td></td>
<td>0.750/3.24/3.997/0.498</td>
<td></td>
<td>0.750/3.24/3.997/0.498</td>
<td></td>
</tr>
<tr>
<td>1024²</td>
<td>±</td>
<td>0.736/0.65/25.196/0.648</td>
<td></td>
<td>0.784/2.24/4.454/0.486</td>
<td></td>
<td>0.784/2.24/4.454/0.486</td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Example 3 - Regression 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 128², 256², 512², 1024² images respectively.

<table>
<thead>
<tr>
<th>Image Size n²</th>
<th>α</th>
<th>M1</th>
<th></th>
<th>M2</th>
<th></th>
<th>M3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128²</td>
<td>±</td>
<td>0.818/0.74/2.4/0.45/ -0.463</td>
<td></td>
<td>0.830/0.93/3.507/0.000</td>
<td></td>
<td>0.815/0.82/2.178/0.169</td>
<td></td>
</tr>
<tr>
<td>256²</td>
<td>±</td>
<td>0.824/0.82/4.929/ -0.132</td>
<td></td>
<td>0.805/1.10/4.275/0.025</td>
<td></td>
<td>0.824/0.74/2.103/0.351</td>
<td></td>
</tr>
<tr>
<td>512²</td>
<td>±</td>
<td>0.832/0.78/4.576/0.161</td>
<td></td>
<td>0.812/1.64/11.546/0.086</td>
<td></td>
<td>0.842/0.78/2.586/0.358</td>
<td></td>
</tr>
<tr>
<td>1024²</td>
<td>±</td>
<td>0.774/0.66/15.139/0.644</td>
<td></td>
<td>0.797/0.40/21.642/0.554</td>
<td></td>
<td>0.774/0.40/21.642/0.554</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4: Example 1 - Registration of 3(a) $R$ and 3(d) $T$ of size $512 \times 512$ by 3 methods. Image (a) shows the deformed template image $T_u$ obtained using the M1, while image (b) shows the deformed template image $T_u$ for our M2 and image (c) shows the deformed template image $T_u$ for our M3 for the parameter value $\alpha = \frac{1}{10}$.

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 $T_u$ and $R$ for M1, our M2 and our M3 respectively.

Figure 6: Example 2 - Registration of 3(b) $R$ and 3(e) $T$ of size $512 \times 512$ by 3 methods. Image (a) shows the deformed template image $T_u$ obtained using the M1, while image (b) shows the deformed template image $T_u$ for our M2 and image (c) shows the deformed template image $T_u$ for our constrained NMG for the parameter value $\alpha = \frac{1}{10}$.
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 $T_u$ and $R$ for the M1, our M2 and our M3 respectively.

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

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 $T_u$ and $R$ for the M1, our M2 and our M3 respectively.
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 \( T_u \) obtained using the \( M2 \), while image (b) shows the deformed template image \( T_u \) for our \( M3 \) and image (c) shows the deformed template image \( T_u \) for our \( M4 \) for the ‘bad’ parameter value \( \alpha = 140 \).

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 \( T_u \) 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.
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


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 $3 \times 3$ 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

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

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

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

\[
= A_1 + pq^T
\]

(A.1)

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

\[
\tilde{A}_l = A_1 + pq^T
\]

(A.2)

with $p$, $q$ as before, and so the inverse $\tilde{A}_l^{-1} = (A_1 + pq^T)^{-1}$, 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 $p, q \in \mathbb{R}^{n \times 1}$ are column vectors. Then $(A + pq^T)$ is invertible $\iff 1 + q^T A^{-1} p \neq 0$. If $(A + pq^T)$ is invertible, then its inverse is given by

\[
(A + pq^T)^{-1} = A^{-1} - A^{-1} pq^T A^{-1} \frac{1}{1 + q^T A^{-1} p}
\]

(A.3)

where $pq^T$ denotes the outer product of the vectors $p, q$.

It can be shown that the multiplication $q_l^T A_l^{-1} p = 0 \forall l = 1, \ldots, 4$, 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 $(A_l + pq^T)^{-1}$ are invertible for each $l = 1, \ldots, 4$. Then we can use Theorem A.1 to rewrite the inverses $(A_l + pq^T)^{-1}$ as

\[
(A_l + pq^T)^{-1} = A_l^{-1} - A_l^{-1} pq^T A_l^{-1} \frac{1}{1 + q^T A_l^{-1} p}.
\]

(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

\[
\begin{align*}
b_{l u_1} &= \omega_{u_1} l(2) - \mu l \omega_{u_1} l(2), & c_{l u_2} &= \omega_{u_2} l(1) - \mu l \omega_{u_2} l(1), \\
b_{l u_2} &= \omega_{u_2} l(2) - \mu l \omega_{u_2} l(2), & c_{l u_2} &= \omega_{u_2} l(3) - \mu l \omega_{u_2} l(3),
\end{align*}
\]

(A.5)

where $\mu_l = \frac{\omega_{u_1} l(2) + \omega_{u_2} l(1) + \omega_{u_3} l(2)}{1 + \omega_{u_1} l(2) + \omega_{u_2} l(1) + \omega_{u_3} l(2)}$, and $\omega_{u_1} l(2), \omega_{u_2} l(3), q_2, q_3, \omega_{u_m} l(2), \omega_{u_m} l(3)$ denote the second and third components of $\omega_{u_1} l = A_{l}^{-1} p$, $q^T$ and $\omega_{u_m} l = A_{l}^{-1} q$ 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 \( Q_{\text{min}} = FEMOpt(u^h, n, h) \)

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 \( A_l^{-1} p \rightarrow \omega_{pl}(2), \omega_{pl}(3) \)
2: end for
3: for \( i = 2, \ldots, n - 1 \) do
4:     for \( j = 2, \ldots, n - 1 \) do
5:         Compute second and third components of \( q_T \rightarrow q_2 = (i - 1)h, q_3 = (j - 1)h \)
       for \( l = 1, \ldots, 4 \) do
       Compute second and third components of \( \omega_{u_1l}, \omega_{u_2l} \rightarrow \omega_{u_1l}(2), \omega_{u_1l}(3), \omega_{u_2l}(2), \omega_{u_2l}(3) \)
       Determine coefficients \( b_{l_{u_1l}}, c_{l_{u_1l}}, b_{l_{u_2l}}, c_{l_{u_2l}} \) using \((A.5)\)
       Compute determinant for triangle \( T_l \rightarrow \hat{Q}_l = (1 + b_{l_{u_1l}})(1 + c_{l_{u_2l}}) - c_{l_{u_1l}}b_{l_{u_2l}} \)
     end for
    Assign minimum \( \hat{Q} \) to be entry \((Q_{ij}) \rightarrow (Q_{ij}) = \min\{\hat{Q}_1, \ldots, \hat{Q}_4\} \)
7: end for
8: end for
Take minimum entry in \( Q \) to be minimum determinant value \( \rightarrow Q_{\text{min}} = \min\{Q\} \)

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.

<table>
<thead>
<tr>
<th>Image Size ( n \times n )</th>
<th>Unoptimised Time (s)</th>
<th>Optimised Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>256(^2)</td>
<td>4.46</td>
<td>0.17</td>
</tr>
<tr>
<td>512(^2)</td>
<td>17.87</td>
<td>0.61</td>
</tr>
<tr>
<td>1024(^2)</td>
<td>71.53</td>
<td>2.40</td>
</tr>
<tr>
<td>2048(^2)</td>
<td>306.23</td>
<td>9.90</td>
</tr>
</tbody>
</table>

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