Technical Report
On the Implementation of Non-Rigid Registration using Fluid Dynamics

G. Wollny

March 17, 2003

1 Preface

Fluid dynamics based registration was first introduced by Christensen [5]. Later Bro-Nielsen et al. [4] suggested speedups of the computational costly method by using digital filters. I implemented the registration approach using different solvers for the solution of the core problem of the fluid-dynamics based registration – the solution of the Navier-Stokes-Equation. The implementation is available [9] under the terms of the GNU General Public License [1]

This documentation comes in the hope that it is helpful, but I do not promise, that it is error-free nor that it is complete. Please address comments to <wollny@cns.mpg.de>.

2 A short Outline of the Method

In the following, an image is given as a mapping $I : \Omega \rightarrow V$ from its coordinate domain $\Omega \in \mathbb{R}^3$ to its intensity range $V \in \mathbb{R}$. Given a coordinate $\vec{x} \in \Omega$, and the intensity of the image I at this coordinate $I(\vec{x})$, the ordered pair $(\vec{x}, I(\vec{x}))$ is referred to as a voxel (volume element). Using a transformation $T : \Omega \rightarrow \Omega$, an image can be changed according to $I_T := I(T(\vec{x}))$. The set of all these transformations is called the *transformation space* Γ.

In this paper, the transformations correspond to spatial displacements of voxels and are described in the so-called *Eulerian reference frame*. Here the voxels are tracked by their position: A voxel originates at time $t_0 = 0$ at coordinate $\vec{x} \in \Omega$. As it moves through Ω, the displacement of a voxel $(\vec{x}, I(\vec{x}))$ at time *t* is given as a vector $\mathbf{u}(\vec{x}, t)$. The set of the displacements of all voxels of an image is called a displacement field over domain Ω, and its value at time t is denoted as $\mathbf{u}(t)$. The corresponding transformation *T* can be given coordinate-wise:

$$T_t(\vec{x}) := \vec{x} - \mathbf{u}(\vec{x}, t) \; \forall \; \vec{x} \in \Omega. \tag{1}$$

The concatenation of transformations is then given as

$$T_1 \circ T_2 := \vec{x} - \mathbf{u}_1(\vec{x} - \mathbf{u}_2(\vec{x})) - \mathbf{u}_2(\vec{x}), \tag{2}$$

The focus of the registration of one (study) image $S : \Omega \rightarrow V$ to another (reference) image $R : \Omega \rightarrow V$ is to find a transformation $T_{min} \in \Gamma$ that minimizes a given cost function $F(R, S_T)$ describing the similarity between transformed study image S and reference image R in conjunction with an energy normalization (smoothness) term $E(T)$ that enforces topology preservation:

$$T_{\min} := \arg\min_{T \in \Gamma} \left(F(R, S_T) + \kappa E(T) \right). \tag{3}$$

κ is a Lagrangian multiplier to balance between registration accuracy and transformation smoothness. Minimizing (3) can be done in terms of its first order derivative:

$$\kappa \frac{\partial}{\partial T} E(T) = -\frac{\partial}{\partial T} F(T, S, R). \tag{4}$$

In the non-rigid registration software I use the sum of squared differences as a cost function:

$$F_c(T, S, R) := \frac{1}{2} \int_\Omega \left[R(\vec{x}) - S(T(\vec{x})) \right]^2 d\vec{x}, \tag{5}$$

and fluid dynamics as smoothness measure.

Then the first order derivative of the cost function (5) can be used to estimate a deforming force:

$$\mathbf{f}(\vec{x},t) := -[S(T(\vec{x},t)) - R(\vec{x})] \; \nabla S|_{T(\vec{x},t)}\,, \tag{6}$$

and with $\kappa = 1.0$, this force (6), and fluid dynamics energy regularisation, (4) can be written as

$$\left(\mu\nabla^2 + (\mu+\lambda)\nabla(\nabla\cdot)\right)\mathbf{v}(\vec{x},t) = -\mathbf{f}(\vec{x},\mathbf{u}(\vec{x},t)). \tag{7}$$

In order to solve the registration problem, (7) is solved for constant time, and the deformation field $\mathbf{u}(t)$ is updated from the estimated velocity field using a time integration step with step-width Δt:

$$\mathbf{u}(\vec{x},t+\Delta t) := \mathbf{u}(\vec{x},t) + \Delta t\left[\mathbf{v}(\vec{x},t) - \nabla\mathbf{u}(\vec{x},t)\mathbf{v}(\vec{x},t)\right]. \tag{8}$$

The solution of the registration problem is summarized in algorithm 1

3 Solving the PDE

Solving PDE (7) is done on a discretization $\widehat{\Omega}$ of the continuous domain Ω.

Christensen's original approach uses *successive over-relaxation* (SOR) [8, pp.866-869] [7, 2, 6] (Algorithm 2).

As an improvement, an adaptive update scheme (SORA) is used in my implementation. In each SOR iteration $\vec{v}_{i,j,k}$ depends on the 19 values with indices

$$\kappa \in \Im := \left\{ \begin{pmatrix} i \\ j \\ k \end{pmatrix}, \begin{pmatrix} i\pm1 \\ j \\ k \end{pmatrix}, \begin{pmatrix} i \\ j\pm1 \\ k \end{pmatrix}, \begin{pmatrix} i \\ j \\ k\pm1 \end{pmatrix}, \begin{pmatrix} i\pm1 \\ j\pm1 \\ k \end{pmatrix}, \begin{pmatrix} i \\ j\pm1 \\ k\pm1 \end{pmatrix}, \begin{pmatrix} i\pm1 \\ j \\ k\pm1 \end{pmatrix} \right\}, \tag{9}$$

only. An adaptive update is now introduced, using an threshold

$$\hat{r} := \begin{cases} 0 & m = 1 \\ \overline{r}^{(m)} \cdot \frac{\overline{r}^{(m)}}{\overline{r}^{(m-1)}} \cdot \frac{1}{m^2} & otherwise \end{cases}, \tag{10}$$

with

$$\overline{r} := \frac{1}{X\cdot Y\cdot Z}\sqrt{\sum \|\vec{r}_{i,j,k}\|^2}, \tag{11}$$

to decide, which elements to update during the iterative solution of (7) (Algorithm 3).

Another approach to solve (7) is the *minimal residuum algorithm* (MINRES) [2], a variant of *conjugated gradients* also suitable for indefinite matrices as they arise when discretizing (7) (Algorithm 4).

Finally Bro-Nielsen approach is based on folding the input force $\mathbf{f}$ (6) with the impulse response of the Navier-Stokes-operator (Section 4.3).

4 Mathematical Derivations

4.1 Discretizing the Navier-Stokes-Equation

$$\mu\nabla^2\mathbf{v} + (\mu+\lambda)\nabla(\nabla\cdot\mathbf{v}) = -\mathbf{f} \tag{12}$$

$$\mu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)\mathbf{v} + (\mu+\lambda)\begin{pmatrix} \frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial x\partial y} & \frac{\partial^2}{\partial x\partial z} \\ \frac{\partial^2}{\partial x\partial y} & \frac{\partial^2}{\partial y^2} & \frac{\partial^2}{\partial y\partial z} \\ \frac{\partial^2}{\partial x\partial z} & \frac{\partial^2}{\partial y\partial z} & \frac{\partial^2}{\partial z^2} \end{pmatrix}\mathbf{v} = -\mathbf{f}, \tag{13}$$

For the *x*-component of (13) we may write:

$$\mu\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right)v^{(x)} + (\mu+\lambda)\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2 v^{(y)}}{\partial x\partial y} + \frac{\partial^2 v^{(z)}}{\partial x\partial z}\right) = -f^{(x)}, \tag{14}$$

$$(2\mu+\lambda)\frac{\partial^2 v^{(x)}}{\partial x^2} + \mu\left(\frac{\partial^2 v^{(x)}}{\partial y^2} + \frac{\partial^2 v^{(x)}}{\partial z^2}\right) + (\mu+\lambda)\left(\frac{\partial^2 v^{(y)}}{\partial x\partial y} + \frac{\partial^2 v^{(z)}}{\partial x\partial z}\right) = -f^{(x)}. \tag{15}$$

Discretizing this using numerical derivatives based on finite differences [8, pp. 186-189] yields

$$\begin{array}{lr} \frac{(2\mu+\lambda)}{h^2}\left(v^{(x)}_{i+1,j,k}+v^{(x)}_{i-1,j,k}-2v^{(x)}_{i,j,k}\right) & + \\ \frac{\mu}{h^2}\left(v^{(x)}_{i,j+1,k}+v^{(x)}_{i,j-1,k}-2v^{(x)}_{i,j,k}+v^{(x)}_{i,j,k+1}+v^{(x)}_{i-1,j,k-1}-2v^{(x)}_{i,j,k}\right) & + \\ \frac{(\mu+\lambda)}{4h^2}\left(v^{(y)}_{i+1,j+1,k}-v^{(y)}_{i-1,j+1,k}+v^{(y)}_{i-1,j-1,k}-v^{(y)}_{i+1,j-1,k}\right) & + \\ \frac{(\mu+\lambda)}{4h^2}\left(v^{(z)}_{i+1,j,k+1}-v^{(z)}_{i-1,j,k-1}+v^{(z)}_{i-1,j,k-1}-v^{(z)}_{i+1,j,k-1}\right) & =-f^{(x)} \end{array}. \tag{16}$$

With shortcuts $a=\frac{\mu}{h^2}$, $b=\frac{\mu+\lambda}{h^2}$ follows,

$$\begin{array}{lr} v^{(x)}_{i,j,k}+\frac{(a+b)}{(6a+2b)}\left(v^{(x)}_{i+1,j,k}+v^{(x)}_{i-1,j,k}\right) & + \\ \frac{b}{(6a+2b)}\left(v^{(x)}_{i,j+1,k}+v^{(x)}_{i,j-1,k}+v^{(x)}_{i,j,k+1}+v^{(x)}_{i-1,j,k-1}\right) & + \\ \frac{b}{4(6a+2b)}\left(v^{(y)}_{i+1,j+1,k}-v^{(y)}_{i+1,j-1,k}+v^{(y)}_{i-1,j-1,k}-v^{(y)}_{i+1,j-1,k}\right) & + \\ \frac{b}{4(6a+2b)}\left(v^{(z)}_{i+1,j,k+1}-v^{(z)}_{i-1,j,k-1}+v^{(z)}_{i-1,j,k-1}-v^{(z)}_{i+1,j,k-1}\right) & =\frac{1}{(6a+2b)}f^{(x)} \end{array}. \tag{17}$$

y- and z- components can be obtained in a similar manner.

With $\hat{\mathbf{f}}:=\frac{1}{(6a+2b)}\mathbf{f}$, writing (12) in its discretized representation yields a linear system

$$\mathbf{A}\mathbf{v}=\hat{\mathbf{f}}. \tag{18}$$

4.2 SOR update

Substituting $c=\frac{a+b}{6a+2b}$, $d=\frac{a}{6a+2b}$, $e=\frac{b}{4(6a+2b)}$ we obtain:

$$\begin{array}{rl} \mathbf{p}= & \hat{\mathbf{f}}_{i,j,k}+c\left(\mathbf{v}^{(m+1)}_{i-1,j,k}+\mathbf{v}^{(m)}_{i+1,j,k}\right)+ \\ & d\left(\mathbf{v}^{(m+1)}_{i,j-1,k}+\mathbf{v}^{(m)}_{i,j+1,k}+\mathbf{v}^{(m+1)}_{i,j,k-1}+\mathbf{v}^{(m)}_{i,j,k+1}\right) \end{array} \tag{19}$$

Setting $\mathbf{v}:=(rst)^T$ we may write

$$\begin{array}{rl} q_x=e & \left(s^{(m+1)}_{i-1,j-1,k}+s^{(m)}_{i+1,j+1,k}-s^{(m)}_{i-1,j+1,k}-s^{(m+1)}_{i+1,j-1,k}+\right. \\ & \left.t^{m+1}_{i-1,j,k-1}+t^{m}_{i+1,j,k+1}-t^{m}_{i-1,j,k+1}-t^{m+1}_{i+1,j,k-1}\right), \\ q_y=e & \left(r^{(m+1)}_{i-1,j-1,k}+r^{(m)}_{i+1,j+1,k}-r^{(m)}_{i-1,j+1,k}-r^{(m+1)}_{i+1,j-1,k}+\right. \\ & \left.t^{(m+1)}_{i,j-1,k-1}+t^{(m)}_{i,j+1,k+1}-t^{(m)}_{i,j-1,k+1}-t^{(m+1)}_{i,j+1,k-1}\right), \\ q_z=e & \left(r^{(m+1)}_{i-1,j,k-1}+r^{(m)}_{i+1,j,k+1}-r^{(m)}_{i-1,j,k+1}-r^{(m+1)}_{i+1,j,k-1}+\right. \\ & \left.s^{(m+1)}_{i,j-1,k-1}+s^{(m)}_{i,j+1,k+1}-s^{(m)}_{i,j-1,k+1}-s^{(m+1)}_{i,j+1,k-1}\right), \end{array} \tag{20}$$

hence for the residual vector

$$\mathbf{r}_{i,j,k}=\omega\left(\mathbf{p}+\mathbf{q}-\mathbf{v}^{m}_{i,j,k}\right), \tag{21}$$

and the SOR update of $\mathbf{v}_{i,j,k}$ is given by

$$\mathbf{v}^{(m+1)}_{i,j,k}=\mathbf{v}^{(m)}_{i,j,k}+\mathbf{r}_{i,j,k}. \tag{22}$$

4.3 Convolution filter

The linear operator of PDE (7) Λ is defined as:

$$\Lambda:=\mu\nabla^2+(\mu+\lambda)\nabla(\nabla\cdot) \tag{23}$$

and its eigenvalues are [5]:

$$\begin{array}{l} \kappa_{1,i,j,k}=-\pi^2(2\mu+\lambda)(i^2+j^2+k^2), \\ \kappa_{2,i,j,k}=\kappa_{3,i,j,k}=-\pi^2\mu(i^2+j^2+k^2), \end{array} \tag{24}$$

with associated eigenvectors:

$$\phi_{1,i,j,k}(\vec{x}) = \sqrt{\tfrac{8}{i^2+j^2+k^2}} \begin{pmatrix} i\ scc_{i,j,k}(\vec{x}) \\ j\ csc_{i,j,k}(\vec{x}) \\ k\ ccs_{i,j,k}(\vec{x}) \end{pmatrix},$$

$$\phi_{2,i,j,k}(\vec{x}) = \sqrt{\tfrac{8}{i^2+j^2}} \begin{pmatrix} -j\ scc_{i,j,k}(\vec{x}) \\ i\ csc_{i,j,k}(\vec{x}) \\ 0 \end{pmatrix}, \tag{25}$$

$$\phi_{3,i,j,k}(\vec{x}) = \sqrt{\tfrac{8}{(i^2+j^2)(i^2+j^2+k^2)}} \begin{pmatrix} ik\ scc_{i,j,k}(\vec{x}) \\ jk\ csc_{i,j,k}(\vec{x}) \\ -(i^2+j^2)\ ccs_{i,j,k}(\vec{x}) \end{pmatrix},$$

where $\vec{x} \in \Omega$,

$$\begin{aligned} scc_{i,j,k}(\vec{x}) &= \sin(i\pi x)\cos(j\pi y)\cos(k\pi z), \\ csc_{i,j,k}(\vec{x}) &= \cos(i\pi x)\sin(j\pi y)\cos(k\pi z), \\ ccs_{i,j,k}(\vec{x}) &= \cos(i\pi x)\cos(j\pi y)\sin(k\pi z), \end{aligned} \tag{26}$$

and

$$\Gamma_{i,j,k} = 2^{\text{sign}(i)+\text{sign}(j)+\text{sign}(k)}. \tag{27}$$

By introducing a filter width parameter $w > 0$, $w \in \mathbf{N}$, which spawns a filter of size $2w+1$, and with the shortcut:

$$\alpha_{i,j,k} = \frac{8}{\pi^2\mu(2\mu+\lambda)(i^2+j^2+k^2)^2\Gamma_{i,j,k}} \tag{28}$$

the components of the impulse response $\Theta \in \mathbb{R}^{3\times3}$ of the linear operator Λ can be written as [3]:

$$\Theta^x(\mathbf{y}) = \textstyle\sum_{i,j,k=0}^{2w} \alpha_{i,j,k} scc_{i,j,k}(\mathbf{y}_c) \begin{pmatrix} (\mu i^2+(2\mu+\lambda)(j^2+k^2))scc_{i,j,k}(\mathbf{y}+\mathbf{y_c}) \\ -(\mu+\lambda)ij\ csc_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \\ -(\mu+\lambda)ik\ ccs_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \end{pmatrix},$$

$$\Theta^y(\mathbf{y}) = \textstyle\sum_{i,j,k=0}^{2w} \alpha_{i,j,k} csc_{i,j,k}(\mathbf{y}_c) \begin{pmatrix} -(\mu+\lambda)ij\ scc_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \\ (\mu j^2+(2\mu+\lambda)(i^2+k^2))csc_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \\ -(\mu+\lambda)jk\ ccs_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \end{pmatrix}, \tag{29}$$

$$\Theta^z(\mathbf{y}) = \textstyle\sum_{i,j,k=0}^{2w} \alpha_{i,j,k} scc_{i,j,k}(\mathbf{y}_c) \begin{pmatrix} -(\mu+\lambda)ik\ scc_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \\ -(\mu+\lambda)jk\ csc_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \\ (\mu k^2+(2\mu+\lambda)(i^2+j^2))ccs_{i,j,k}(\mathbf{y}+\mathbf{y}_c) \end{pmatrix},$$

with $\mathbf{y}_c = (0.5, 0.5, 0.5)^T$ and $\mathbf{y} \in \{y_{r,s,t} = (\frac{r}{d}, \frac{s}{d}, \frac{t}{d})^T \,|\, r,s,t \in [-d,d] \cap \mathbf{Z}\}$.

5 Algorithms

5.1 Main Registration Algorithm

This algorithm is implemented in the files `vfluid/vfluid.(cc|hh)`.

Using the time step parameter $d \in [d_{\min}, d_{\max}]$ an adaptive control of the integration time step is achieved. $[d_{\min}, d_{\max}]$ should be chosen to permit a smooth but steady deformation, and $\Delta d \ll d_{\max} - d_{\min}$ is used to re-adjust d during the registration.

Algorithm 1 non rigid registration based on fluid dynamics

$d := d_{\max}$, i:=0, **u**(0):=**0,** $T := T_0$, $\hat{S} := S(T)$
calculate mismatch m_i by . (5)
repeat
  $i := i + 1$
  calculate $\mathbf{f}(t_i)$ (6)
  solve the linear PDE . (7) for velocity $\mathbf{v}(t_i)$ and force $\mathbf{f}(t_i)$
  label:
  choose $\Delta t = \frac{d}{|\vec{x} - \nabla \vec{u}(\vec{x})\mathbf{v}(\vec{x})|}$
  if $\min_{\vec{x}}$ det($\mathbf{I} - \nabla(u(\vec{x}) - \Delta t * (v(\vec{x}) - \nabla u(\vec{x})v(\vec{x})))) < 0.5$ **then**
    $T_{\vec{u}} := \vec{x} - \vec{u}(\vec{x})$
    $T := T \circ T_{\vec{u}}$, $\hat{S} := S(T)$, **u**:=0
  end if
  $\vec{u}(\vec{u}) \leftarrow \vec{u}(\vec{x}) + \Delta t * (v(\vec{x}) - \nabla u(\vec{x})v(\vec{x}))$
  calculate mismatch m_i using (5)
  if $m_i > m_{i-1}$and $d > d_{\min}$ **then**
    $d := \max(d - \Delta d, d_{\min})$
    goto label
  end if
  $d := \min(d + \Delta d, d_{max})$
until $m_i > m_{i-1}$
$T := T \circ \vec{u}(t_{i-1})$
T is the transformation minimizing the cost function (5)

5.2 Successive Over-Relaxation

This algorithm is implemented in `vfluid/sor_solver.(cc|hh)`.

Algorithm 2 SOR

$\hat{\mathbf{f}} = \frac{\mathbf{f}}{6a+2b}$, select values for $maxsteps$ and ε, set initial **v**
repeat
  for $k := 1$ to Z step 1 **do**
    for $j := 1$ to Y step 1 **do**
      for $i := 1$ to X step 1 **do**
        calculate $\mathbf{p}_{i,j,k}$ as given in (19) { 24 FLOPs }
        calculate $\mathbf{q}_{i,j,k}$ as given in (20) { 24 FLOPs }
        calculate $\mathbf{r}_{i,j,k}$ as given in (21) { 9 FLOPs }
        update $\mathbf{v}_{i,j,k}$ as given in (22) { 3 FLOPs }
        $r := r + \|\mathbf{r}_{i,j,k}\|^2$ 6 FLOPs
      end for
    end for
  end for
  { one iteration needs *O*(66*n*) FLOPs }
  if step=1 **then**
    $r_{init} := r$
  end if
until $steps \geq maxsteps$ or $r < \varepsilon * r_{init}$

5.3 Adaptive Update

This algorithm is implemented in `vfluid/sor_solver.(cc|hh)`.

Algorithm 3 SORA

1. $\hat{r} := 0$, $m := 1$
2. calculate the first iteration over the full domain as given in Algorithm 2, and the residue $r_{i,j,k} = \|\mathbf{r}_{i,j,k}\|$
3. if $r_{i,j,k} \geq \hat{r}$ mark $v^* \in \Im$ as to be updated in the next iteration
4. $r_{old} := r$, $r := \sum_{i,j,k} r_{i,j,k}$
5. set threshold $\hat{r}$ as given in (10)
6. in sub-sequential iterations *m* of Algorithm 2 update $v_{i,j,k}$ and $r_{i,j,k}$ only at marked positions, update the marks as given in step 3, and threshold $\hat{r}$ as given in step 5.

5.4 The Minimum Residual Algorithm

This algorithm is implemented in `vfluid/cg_solver.(cc|hh)` and `mia/cg.hh`.

Algorithm 4 MINRES

select values for $maxsteps$ and ε
set initial $\mathbf{v}_0$
$\mathbf{r}_0 = \widetilde{\mathbf{f}} - \mathbf{A}\mathbf{v}_0$
$\overline{\mathbf{r}}_0 = \mathbf{A}\mathbf{r}_0$
$\mathbf{p}_0 = \mathbf{r}_0$, $\overline{\mathbf{p}}_0 = \overline{\mathbf{r}}_0$, $\gamma_0 = \overline{\mathbf{r}}_0 * \mathbf{r}_0$
repeat
  $\mathbf{h_k} = \mathbf{A}\mathbf{p_k}$ { *O*(51*n*) FLOPs }
  $\alpha_k = \frac{\gamma_k}{\overline{\mathbf{p}}_k * \mathbf{h_k}}$ { *O*(*2n*) FLOPs }
  $\mathbf{v_{k+1}} = \mathbf{v_k} + \alpha_k \mathbf{p_k}$ { *O*(*2n*) FLOPs }
  $\mathbf{r_{k+1}} = \mathbf{r_k} - \alpha_k \mathbf{h_k}$ { *O*(*2n*) FLOPs }
  $\overline{\mathbf{r}}_{k+1} = \overline{\mathbf{r}}_k - \alpha_k * (\mathbf{A} * \overline{\mathbf{p}}_k)$ {*O*(*53n*) FLOPs }
  $\gamma_{k+1} = \overline{\mathbf{r}}_k * \mathbf{r}_k$ { *O*(*2n*) FLOPs }
  $\beta_k = \frac{\gamma_{k+1}}{\gamma_k}$
  $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k * p_{\mathbf{k}}$ { *O*(*2n*) FLOPs }
  $\overline{\mathbf{p}}_{k+1} = \overline{\mathbf{r}}_{k+1} + \beta_k * \overline{p}_{\mathbf{k}}$ { *O*(*2n*) FLOPs }
until $steps \geq maxsteps$ or $|\mathbf{r}_{k+1}| > \varepsilon\,|\mathbf{r}_0|$
{ one iteration needs *O*(*117n*) FLOPs }

References

[1] Gnu general public license. http://www.gnu.org/licenses/gpl.html.

[2] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. M. Donato, J. Dongarra, V. Eijkhout, R. Pozo, Ch. Romine, and H. Van der Vorst. Templates for the solution of linear systems: Building blocks from iterative methods. SIAM, http://www.netlib.org/templates/templates.ps, 1993.

[3] M. Bro-Nielsen. *Medical Image Registration And Surgery Simulation*. PhD thesis, Technical University of Denmark, 1996.

[4] M. Bro-Nielsen and C. Gramkov. Fast fluid registration of medical images. In *Visualisation in biomedical computing (VBC96)*, volume 1131 of *Lect. Notes Comp. Sci*, pages 267–276, Hamburg, Sep. 1996. Springer-Verlag.

[5] G. E. Christensen. *Deformable shape models for neuroanatomy*. DSc.-thesis, Server Institue of Technology, Washington University, Saint Louis, 1994.

[6] G. Engeln-Müllges and F. Reutter. *Formelsammlung zur Numerischen Mathematik mit Turbo Pascal Programmen*. Bibliographisches Institut & F. A. Brockhaus AG, Mannheim, 3. aufl. edition, 1991.

[7] C. C. Paige and M. A. Saunders. Solution of sparse indefinit systems of linear equations. *SIAM Journal of Numerical Analysis*, 12(4):866–869, 1975.

[8] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, New York, secon edition edition, 1992.

[9] G. Wollny. Mia - a toolchain for medical image analysis. http://mia.sourceforge.net, 2002.