CS556 - ITERATIVE AND MULTIGRID METHODS: FALL 2018

ALGEBRAIC MULTIGRID METHODS FOR NONLINEAR HYPERELASTICITY PROBLEMS

BHAVESH SHRIMALI TEJASWIN PARTHASARATHY *

Abstract. We survey the performance of algebraic multigrid (AMG) methods to solve nonlinear elliptic partial differential equations (PDEs) in hyperelasticity. We discretize the underlying PDE in $N = 2$ space dimensions using conforming quadrilateral (Q–k) finite elements and solve the resulting algebraic equations for the nodal displacements using two different AMG algorithms (a): Newton–AMG and (b) Full Approximation Scheme (FAS). We explore the effectiveness of AMG across a breadth of physical (e.g. material compressibility) and algorithmic (e.g. choice of near null-space candidate vectors) parameters. We highlight the potential benefits of employing AMG, both as a solver and preconditioner, for hyperelastic problems.

1. Problem Statement. The effective finite element solution of elliptic PDEs arising in solid mechanics, e.g. hyperelasticity (10), homogenization (8) and electro-magneto-elasticity (9) requires highly efficient and parallelizable iterative solvers. In this paper we seek to develop numerical solutions to a sample 2D BVP in hyperelasticity with the following choice of free energy function $W(\mathbf{F})$

$$W(\mathbf{F}) := \frac{\mu}{2}(I_1 - 2) - \mu \ln J + \frac{\lambda}{2}(J-1)^2, \quad \text{where} \quad I_1 = \mathbf{F}\cdot\mathbf{F},\ J = \det\mathbf{F}. \tag{1.1}$$

The corresponding constitutive relation and the strong form of the governing equation can then be compactly written as

$$\mathbf{S}(\mathbf{F};\mathbf{X}) = \frac{\partial W}{\partial \mathbf{F}}(\mathbf{F};\mathbf{X}) = \mu\mathbf{F} - \mu\mathbf{F}^{-T} + \lambda J(J-1)\mathbf{F}^{-T} \tag{1.2}$$

$$\nabla\mathbf{S} + \mathbf{b}_0 = \mathbf{0} \quad \text{in } \Omega \quad \text{and} \quad \begin{cases} \chi(\mathbf{X}) = \overline{\mathbf{F}}\mathbf{X}, & \text{for } \mathbf{X} \in \partial\Omega_x \\ \mathbf{S}\mathbf{n}_0 = \mathbf{s}_0 & \text{for } \mathbf{X} \in \partial\Omega_t \end{cases}, \tag{1.3}$$

where $\mathbf{X}$ represents the Lagrangian coordinate, $\mathbf{F} = \nabla\chi(\mathbf{X})$ is called the deformation gradient, $\mathbf{S}$ the first Piola-Kirchoff stress tensor, μ the initial shear modulus, and λ the lamé constant. The body force $\mathbf{b}_0$ in (1.3) is an important parameter to develop and test benchmark solutions. Alternatively the problem can also be expressed in the form of a variational principle

$$\min_{u\in\mathcal{U}} \Pi(u) := \int_\Omega W(\mathbf{F})\, d\mathbf{X} - \int_\Omega \mathbf{b}_0\cdot\mathbf{u}\, d\mathbf{X} - \int_{\partial\Omega_t} \mathbf{s}_0\cdot\mathbf{u}\, d\mathbf{X}. \tag{1.4}$$

where

$$\mathcal{U} = \mathbf{v} \ni \mathbf{v} \in W^{(1,k)}(\Omega), \mathbf{v} = \overline{\mathbf{F}}\mathbf{X} \quad \forall\, \mathbf{X} \in \partial\Omega_x \tag{1.5}$$

is a set of candidate solutions satisfying kinematic boundary conditions. The Euler-Lagrange equations associated with the above variational principle are given by

$$\mathcal{G}(\mathbf{u},\mathbf{v}) = \int_\Omega \frac{\partial W}{\partial \mathbf{F}}(\nabla\mathbf{u};\mathbf{X})\cdot\delta\nabla\mathbf{v}\, d\mathbf{X} - \int_\Omega \mathbf{b}_0\cdot\delta\mathbf{v}\, d\mathbf{X} - \int_{\partial\Omega_t}\mathbf{s}_0\cdot\delta\mathbf{v}\, d\mathbf{X} = 0 \quad \forall\, \mathbf{v}\in\mathcal{U}_0 \tag{1.6}$$

*bshrima2@illinois.edu (Bhavesh Shrimali) tp5@illinois.edu (Tejaswin Parthasarathy)

where

$$\mathcal{U}_0 = \mathbf{v} \ni \mathbf{v} \in W^{(1,k)}(\Omega), \mathbf{v} = \mathbf{0} \quad \forall\, \mathbf{X} \in \partial\Omega_x \tag{1.7}$$

Choosing a suitable partition of the domain Ω into $\mathcal{N}$ non-overlapping elements and a polynomial basis $\varphi(\mathbf{X})$ with compact support over each element to represent the primal field $\mathbf{u} = \varphi(\mathbf{X})\,\mathbf{d}$ and the test function $\mathbf{v} = \varphi(\mathbf{X})\,\mathbf{d}_0$ respectively, the discrete version of (1.6) and its linearized form (using Newton's method) with a slight abuse of notation, can be written as

$$\mathcal{G}(\mathbf{d}, \mathbf{d}_0) = 0 \quad \forall\, \mathbf{d}_0 \tag{1.8}$$

$$\mathcal{G}\left(\mathbf{d}^k, \mathbf{d}_0\right) + D\mathcal{G}\left(\mathbf{d}^k, \mathbf{d}_0\right) \delta\mathbf{d}^k = 0 \quad \forall\, \mathbf{d}_0 \tag{1.9}$$

2. Approach. We solve the resulting algebraic system (1.8) using non-linear AMG methods. Our motivation to use these techniques stems from their efficiency (3), high parallelizability (1; 2) and a spate of successful applications in solving such non-linear problems (4; 6), with good convergence properties and small time-to-solution (TTS). Moreover the discretized systems considered in this work are expected to be SPD, based on the energy minimization property (1.4), for which AMG methods have a history of success (3; 5). We focus only on two such methods—(a) Newton–AMG and (b) Full Approximation Scheme (FAS). Newton–AMG involves approximating the solution to (1.8), by equivalently solving a series of locally-valid linear systems (1.9) using AMG. FAS however, solves (1.8) by solving a series of smaller non-linear problems, using concepts of AMG.

For both the methods listed above, we utilize the method of manufactured solution to test the validity of our solver(s). We proceed to study the effect of physical (for e.g. λ, the lamé constant representing material compressibility) and algorithmic parameters (for e.g. the choice of near-null space candidate vectors) on the performance of AMG as both a solver and a pre-conditioner, which include the convergence factors and TTS. We then attempt to explain our observations based on physical interpretations.

We will perform the computation in `Python`, using vectorized code from `numpy` for dense- and `scipy.sparse` for sparse-matrix/vector operations, `sympy` for symbolic algebraic manipulations and `matplotlib`, `VTK` for visualization. We also utilize the AMG routines and canned solvers in `PyAMG` (7).

3. Implementation and Numerical Results. In this section we briefly describe the physical and numerical (conforming FE) framework and present representative simulations. We take a prototypical 2D domain in the Euclidean space $\Omega \subset \mathbb{R}^2 = [0, 10]^2$ for solving (1.3) and subject it to various prescribed boundary conditions of the form $(1.3)_2$ by changing $\overline{\mathbf{F}}$. The resulting geometric nonlinearities are successfully captured by our solver (see Figure 3.1) in the sense that it generates non-singular algebraic systems. Additionally we construct the algorithm such that it can be used for different variational problems (1.4) by directly changing the free- or stored-energy (1.1) in near-mathematical notation, without tweaking other parts of the algorithmic pipeline.

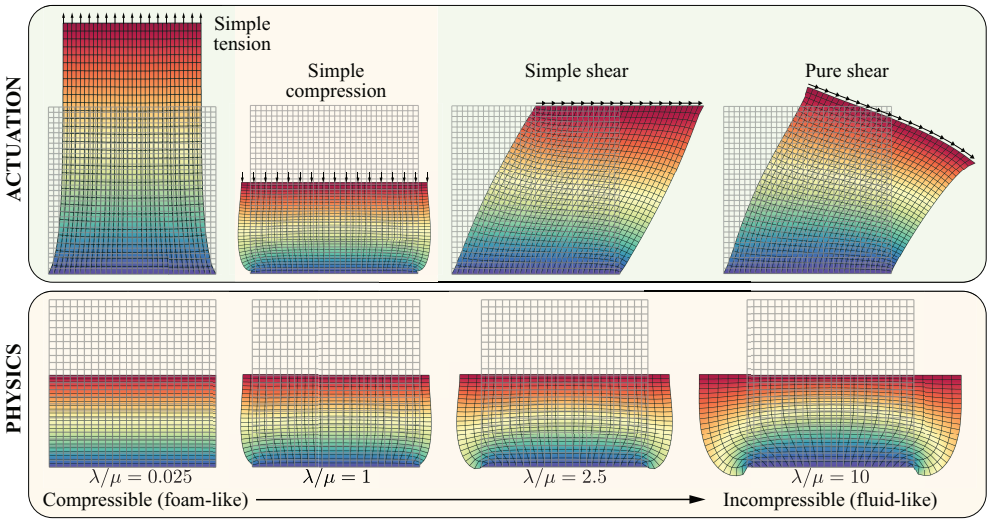

Fig. 3.1: Solver capability: Our FE strategy works across a range of actuation (prescribed displacements with stretches as high as 150%) and physical (prescribed as changes in the constitutive model) parameters. The colors represent the material deformation in the direction of these prescribed displacements (indicated by arrows on the corresponding boundary), with blue and red representing low and high deformations respectively.

We then demonstrate the capabilities of our FE solver to reproduce manufactured solutions of (1.3). The physical setup that reproduces one such solution, with $\overline{\mathbf{F}} = \text{diag}\left[\lambda_1, \overline{\lambda} = 1.5\right]$ is shown in Figure 3.2(a) and the corresponding non-linear deformation, captured using Q–1 elements, is shown in Figure 3.1(b). With this setting we observe a first-order spatial convergence, as seen in Figure 3.1(c). We add that the deformation is prescribed quasi-statically in steps (see inset, Figure 3.1(d)), with the nonlinear system of equations (1.8) at each step solved using Newton's method, which exhibits quadratic convergence (Figure 3.1(d)).

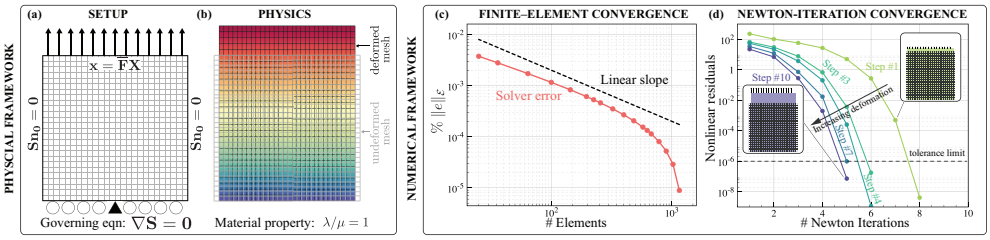

Fig. 3.2: Solver validity: We validate our solution for the simple tension test case shown in (a) against its manufactured solution, wherein we solve $\nabla\mathbf{S} = \mathbf{0}$ with prescribed displacement at the top (indicated by arrows). This leads to (b) high deformations, with red (blue) indicating high (low) values in the imposed direction. (c) By tracking the relative error in the energy norm $\|\cdot\|_{\mathcal{E}}$ (see **??** for details) with increasing spatial resolution, we observe that FEM converges linearly. To ensure global convergence, we prescribe deformations distributed among many steps (10 in this case), leading to (d, inset) quasi-static material deformations. We use Newton's method to perform the non-linear solve at each such step, and observe the (d) quadratic convergence at *every* step as expected.

3.1. Newton–AMG. We solve the linearized system (1.9) using AMG. We choose a model pure-shear problem (see Figure 3.1) in view of its (a) simplicity and (b) ability to incorporate compression, extension and shearing of elements from

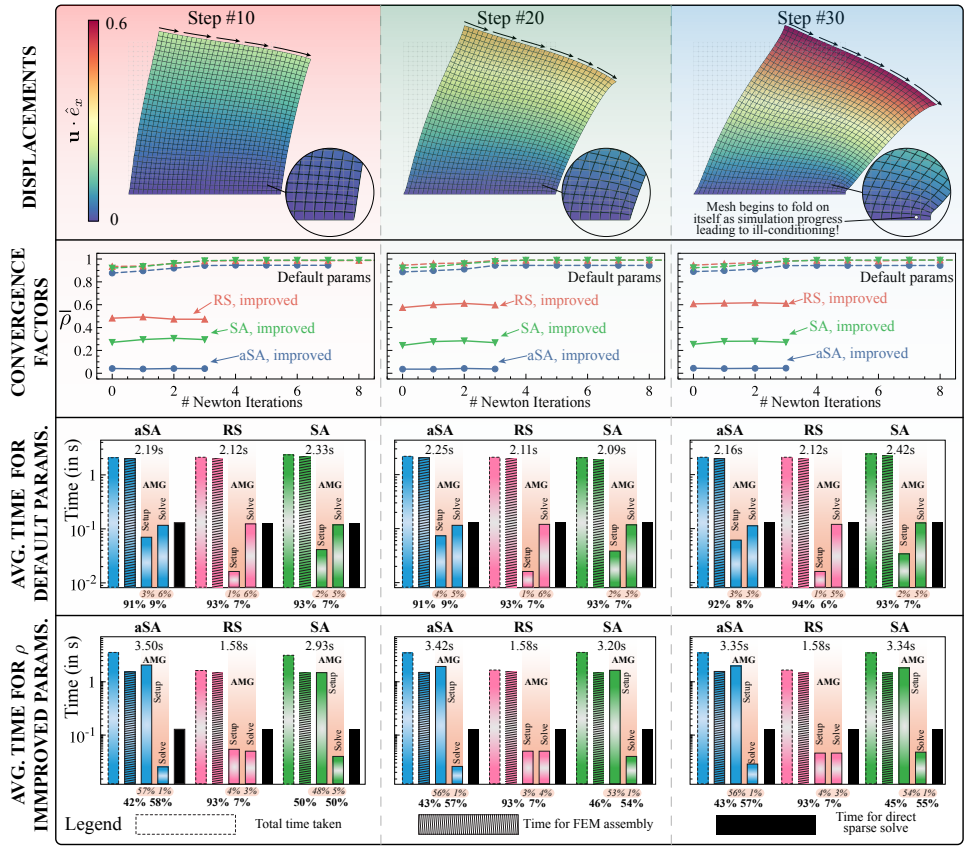

Fig. 3.3: AMG: To compare the performance of AMG as a preconditioner to CG in Newton's method, we simulate the pure-shear case shown in (*row 1*) above. Here we go from a well-conditioned system (in step #10, left) to an ill-conditioned system (in step #30, right), due to mesh folding. The colors indicate the x-displacement of the material, from blue (no displacement) to red (max displacement of 0.6). Using AMG, we see (*row 2*) bad performance in the convergence factors, using the default parameters of `PyAMG`, indicated by the dashed lines. We iterate on the parameter set to arrive at an improved set, with better performance (solid lines) for Ruge–Stüben (red lines), Smoothed Aggregation (SA, green lines) and adaptive Smoothed Aggregation (aSA, blue lines) preconditioners. We instrument the code and obtain average timing data for both default (*row 3*) and improved (*row 4*) parameters, represented as a bar plot in the semilog scale. Here dashed borders for the bar indicate total time, a hatched bar indicates assembly time and black bars represent the time taken to solve by the direct sparse solver routine of `Scipy.sparse`. The AMG times are marked with an orange halo, with both setup and solve times listed. The percentage split up between different components (and within each component, whenever possible is also shown).

characteristic physics, thus testing all 'modes', while simultaneously (c) enabling the comparison of AMG performance across a range of linear-system conditioning, from well-conditioned (step #10, #20 in the first row of Figure 3.3) to ill-conditioned (step #30 in the first row of Figure 3.3) systems due to eventual folding of the mesh with increasing deformation.

We obtain the solutions of the linear system using preconditioned conjugate-gradient with three different AMG preconditioning algorithms—(a) adaptive Smoothed Aggregation (aSA), (b) Ruge–Stüben with classical C/F splitting and (c) Smoothed Aggregation (SA)—and compare their performance metrics at the three aforementioned load-steps. The first metric relates to the averaged convergence factors $\overline{\rho}$ (de-

fined in the Algorithm A.1) for every Newton iteration and is shown in the second row of Figure 3.3. With the default parameter set from `PyAMG`, the $\overline{\rho}$ for all three variants $\in [0.9, 0.95]$, indicating an ineffective solver. We note that in some cases an increase in Newton Iterations is quintessential in reducing the residuals (till convergence) of the non-linear equations (1.8), as the maximum number of Krylov iterations within one Newton iteration was restricted.

We then seek to improve $\overline{\rho}$ using different set of parameters for each of the above algorithms. We report them in the same graph, with the corresponding improved parameter set reported in Table A.1. Not suprisingly, we observe improvement in performance, with aSA leading in the $\overline{\rho}$ metric, followed by SA and RS. We further observe remarkable consistency in convergence across conditioning (from step#10 to #30), with only a mild penalty for the ill-conditioned cases. Moreover, aSA has textbook performance ($\overline{\rho} = 0.04 - 0.06$), presumably because it finds the best null-space candidates, corresponding to the rigid translation and rotation modes of the system. SA, with manually estimated translation–rotation null-space candidates, suffers a penalty of $5\times$ in $\overline{\rho}$.

Investigating the time taken for obtaining the solutions for *one load step*, we obtain the third row of Figure 3.3, with the total, assembly, setup and solve times (and their percentage in the composition) reported for different AMG solvers for the default parameters. Across the different load steps and solvers, we notice that the total time taken is similar, with the bottleneck of our algorithmic implementation lying in the matrix assembly—indeed, we loop over all elements in our `Python` implementation, leading to characteristic computational inefficiencies. The RS preconditioner takes the least amount of time to setup, followed by SA (due to aggregation) and finally aSA (to adapt, aka form the candidate vectors). The solve times, however, are similar. A comparison with the direct sparse solver from `Scipy.sparse` indicates that the AMG solve is slightly more efficient, but comes at the expense of the required setup time.

The parameter set for the cases with improved convergence changes the time split-up. We now see (fourth row of Figure 3.3) that aSA takes the most amount of time, followed by SA and then RS, in contrast with similar total times observed earlier in row three. We directly correlate this to the setup times—the need for more adaptation (more candidate vectors), smoothing the interpolation operators etc. increase heavily the setup cost of aSA and SA. RS also sees and increased setup time, but not to the same extent as its counterparts. The cost incurred in the setup pays off during the solve, with exactly the reverse order—RS is more expensive, followed by SA and then aSA. We add that all AMG routines perform far better than a direct solver, once setup. An interesting observation pertains to the TTS—while aSA performs better in the $\overline{\rho}$ metric, RS performs the best in the metric of total time taken (which is of the most interest), taking similar time (in both solve and setup) to a direct sparse solve, for th system size investigated.

4. Conclusions. In this paper we investigated the performance of AMG preconditioners for solving system of nonlinear algebraic equations encountered in Hyperelasticity. Newton–AMG was chosen as the method to solve the nonlinear equations in view of (a) robustness of Newton's method to deliver quadratic convergence for variational problems, and (b) ability of AMG to provide accurate preconditioners to iterative Krylov-subspace solvers (`cg`,`gmres`, etc.). We recorded the performance of adaptive Smoothed Aggregation, Classical Ruge-Stüben and Smoothed Aggregation methods in solving a representative BVP. Our $\overline{\rho}$ experiments indicated that aSA can have remarkable performance——this feeds into the idea of having *AMG-memory*,

wherein we take the adaptation information from a prior iteration and feed it into consequent iteration(s), to potentially reduce setup times. From our timing experiments, we infer that balancing setup and solve times of AMG might be key to obtain improved performance—indeed this was the case with our improved RS parameter set, which had better TTS while still converging within four Newton iterations. While these methods perform reasonably well when compared to the available direct solvers, their full potential can be achieved for larger problem sizes and with a more robust framework.

References.

[1] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, *Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom*, in Proceedings of the 2004 ACM/IEEE conference on Supercomputing, IEEE Computer Society, 2004, p. 34.

[2] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, *Scaling hypre's multigrid solvers to 100,000 cores*, in High-Performance Scientific Computing, Springer, 2012, pp. 261–279.

[3] W. L. Briggs, S. F. McCormick, et al., *A multigrid tutorial*, vol. 72, Siam, 2000.

[4] M. W. Gee and R. S. Tuminaro, *Nonlinear algebraic multigrid for constrained solid mechanics problems using trilinos.*, tech. report, Sandia National Laboratories, 2006.

[5] V. E. Henson, *Multigrid methods for nonlinear problems: an overview*, tech. report, Lawrence Livermore National Lab., CA (US), 2002.

[6] A. Klawonn, M. Lanser, and O. Rheinbach, *Toward extremely scalable nonlinear domain decomposition methods for elliptic partial differential equations*, SIAM Journal on Scientific Computing, 37 (2015), pp. C667–C696.

[7] L. N. Olson and J. B. Schroder, *PyAMG: Algebraic multigrid solvers in Python v4.0*, 2018, https://github.com/pyamg/pyamg. Release 4.0.

[8] B. Shrimali, V. Lefèvre, and O. Lopez-Pamies, *A simple explicit homogenization solution for the macroscopic elastic response of isotropic porous elastomers*, Journal of the Mechanics and Physics of Solids, 122 (2019), pp. 364–380.

[9] L. Tian, L. Tevet-Deree, K. Bhattacharya, et al., *Dielectric elastomer composites*, Journal of the Mechanics and Physics of Solids, 60 (2012), pp. 181–198.

[10] J. A. Weiss, B. N. Maker, and S. Govindjee, *Finite element implementation of incompressible, transversely isotropic hyperelasticity*, Computer methods in applied mechanics and engineering, 135 (1996), pp. 107–128.

Appendix A. FE Solver. The FE solver used to generate the numerical results presented in section 3 uses Algorithm A.1 and the corresponding optimized paramters used in the calculations are summarized in Table A.1.

Algorithm A.1 FE Solver: Newton Multigrid

Define $W := W(I_1, I_2, I_3)$ and calculate its derivatives wrt invariants $\partial W/\partial I_i$
Define Element type , Quadrature rule, import geometry, mesh, connectivity (N_{el}) and assign boundary conditions
Create arrays for $\mathbf{S}$, $\mathbf{F}$, `dofstore`
Initialize dofs (`disp`) and deformation gradient $\mathbf{F}$, Jacobian $\partial^2 W/\partial \mathbf{F}^2$, and residuals $\partial W/\partial \mathbf{F}$.
Define load steps `nSteps` and initialize step counter i to 0, $i = \{1, 2, \ldots, \texttt{nSteps}\}$.

while $i <=$ `nSteps` **do**
  Initialize the Newton iteration counter `iterN` to 0, and apply `disp`= $\mathbf{d}$
  Assemble the global Jacobian ($\mathbf{K}$) and residual ($\mathbf{r}$) as explained below
  while $\|\mathbf{r}\{\mathbf{d}\}\|_2 >$ `tolNR` and `iterNR` <= `maxiters` **do**
    Initialize element counter k_{el} to 0
    while $k_{el} <= N_{el}$ **do**
      At each gauss point inside the element k_{el} calculate $\mathbf{F}$ using current value of $\mathbf{d}$
      Compute $\mathbf{S}$ from (1.2) at each quadrature point and append to the element tensor
      Integrate the Jacobian $\partial^2 W/\partial \mathbf{F}^2$ and the residual $\partial W/\partial \mathbf{F}$ over element k_{el} and local quadrature points $\{\xi, \eta\} \in (-1, 1)$ and calculate the element level tangent (for Newton)
      Assemble the element level energy $W(\mathbf{F})$ at quadrature points and store $\mathbf{S}$ and $\mathbf{F}$ for visualization
    end while
    Append the time taken for the assembly (`timeAssemb`) to global list
    Solve the linearized system $\mathbf{K}\,\Delta\mathbf{d} = \mathbf{r}$ (1.9) using MG (`aSA`, `RS`, `SA`)
    Store the MG-residuals `res` and determine the average convergence factor ρ
    Append the solve time `tN` to the global list `timeStep`
    $\mathbf{d} \leftarrow \mathbf{d} + \Delta\mathbf{d}$
    Compute the residual $\mathbf{r}(\mathbf{d})$
    `iterNR` $\leftarrow$ `iterNR`+1
  end while
  $i \leftarrow i + 1$
end while

Throughout the calculations we assume that the FE solution $\widetilde{\mathbf{u}}(\mathbf{X})$ is in the energy space $\mathcal{E}(\Omega)$ where

$$\mathcal{E}(\Omega) := \mathbf{u} \ni \|\mathbf{F}(\mathbf{u})\| < +\infty \quad \text{and} \quad \int_\Omega W(\mathbf{F})\, d\mathbf{X} < +\infty \tag{A.1}$$

This allows to determine the convergence of the FE solution in the energy norm. Since the problem (1.3) in general does not admit exact solution, we approximate the exact solution by the FE computed on the finest mesh ($\widetilde{\Omega}$). The relative error in the energy norm of the FE solution is given by

$$e_{\mathcal{E}(\Omega)} = \sqrt{\frac{\mathcal{U} - \mathcal{U}_h}{\mathcal{U}}} \quad \text{where} \quad \mathcal{U} = \int_{\widetilde{\Omega}} W(\mathbf{F})\, d\mathbf{X} \tag{A.2}$$

Solver	Optimized Parameters
aSA	initial_candidates = Bvec, num_candidates = 5, improvement_iters = 5, aggregate = 'naive'
GS	strength = 'evolution', CF = 'RS',max_levels = 50, pre/post-smoother = ('GS', 'symmetric'), max_coarse = 50
SA	smooth = ('energy'), strength = 'evolution', max_coarse = 50, max_levels= 15

Table A.1: Optimized Parameters for Multigrid Solvers used in Algorithm A.1