IMA Journal of Applied Mathematics (2025) **00**, 1–13
https://doi.org/10.1093/imamat/hxaf004

Data approximation by neural nets for the MRE inverse problem in the frequency and time domains

Penny J. Davies*
Department of Mathematics and Statistics, University of Strathclyde, 26 Richmond Street, Glasgow, G1 1XH, UK
*Corresponding author: penny.davies@strath.ac.uk

[Received on 18 April 2024; revised on 13 December 2024; accepted on 11 February 2025]

'Stacked' matrix approximation methods for computing the shear modulus in the magnetic resonance elastography (MRE) inverse problem have been shown to work well in both the frequency and time domain formulations, and are robust to moderate levels of noise. However, when finite differences are used to approximate derivatives of the measured displacement, the algorithms can break down at high noise levels. Here we show that if instead a neural network is used to approximate the derivatives of the noisy displacement data, then the overall MRE algorithms become much more robust to noise pollution. Extensive tests indicate that the new methods perform extremely well even in the presence of very high levels of noise, and no additional processing or smoothing of the data is required.

Keywords: magnetic resonance elastography (MRE); elasticity; biomechanics; inverse problem; neural networks.

1. Introduction

New methods for magnetic resonance elastography (MRE) are presented in Davies *et al.* (2019) (for the frequency domain problem) and Davies & Sack (2020) (for the time domain problem), and shown to work well when there are low to moderate levels of noise in the experimental measurements/simulations. However, because these methods use finite differences to approximate derivatives, they do not work well at high noise levels (see (Davies *et al.*, 2019; Davies & Sack, 2020) for more details and (Hanke & Scherzer, 2001) for a related discussion of numerical stability and sensitivity). The aim of this short update is to show how these algorithms can be made robust (even to very high levels of noise) by exploiting the excellent function approximation properties of neural networks (NNs) (see e.g. (Lu *et al.*, 2021, Section 2.4) and the references therein). The key step is to use automatic differentiation of an NN representation of the measured displacement data instead of a finite difference approximation. We show that the resulting method works remarkably well, with no additional data processing or smoothing required. Note that this is intended as a 'proof of concept' rather than a detailed investigation of the best way to approximate the displacement derivative in MRE algorithms. Other function and derivative approximations may also work well, but focusing on the NN approach here is a deliberate choice because there are many easily accessible NN codes available (via open source or in common programming environments such as Matlab) that applied mathematicians can use for function or derivative approximation. The appendix contains a short sample code based on Matlab's Deep Learning Toolbox.

MRE involves the noninvasive evaluation of a tissue's elastic shear modulus, with applications in disease diagnosis (as described e.g. by (McLaughlin *et al.*, 2010; Doyley, 2012; Dittmann *et al.*, 2016; Barnhill *et al.*, 2018)). A typical experimental scenario is to induce elastic waves in tissue by external

mechanical excitation (e.g. the patient lies on on a vibrating table) and measure the resulting displacement at fixed sites within the tissue using phase-contrast MRI. This provides a far richer data source than methods which measure displacement or strain at the tissue surface, but it is still an extremely challenging inverse problem, with many different methods proposed (see e.g. (Manduca *et al.*, 2001; Park & Maniatty, 2006; McLaughlin *et al.*, 2010; Sánchez *et al.*, 2010; Doyley, 2012; Barnhill *et al.*, 2018)).

The MRE approximation methods of Davies *et al.* (2019) and Davies & Sack (2020) involve combining measurements taken at different underlying mechanical oscillation frequencies into an over-determined least-squares system for $\boldsymbol{\mu}$, the reconstructed values of the shear modulus at equispaced points within the tissue. Extensive tests on 'synthetic' data in 1D and 2D space and experimental measurements in 3D space (in full 3D and also evaluated as 2D spatial slices) show very similar performance for both the frequency domain and time domain, with results tending to be better in higher space dimensions. In all cases, the methods work well when there are low to moderate levels of noise in the measured/simulated displacements, but high levels of noise results in the computed value of $\boldsymbol{\mu}$ being far too low—this is a well-known phenomenon in MRE, as noted in e.g. Arunachalam *et al.* (2017). The reason for this is explained in Davies & Sack (2020): the vector $\boldsymbol{\mu}$ (of shear modulus values at points in space) solves a least-squares system of the form

$$\min_{\boldsymbol{\mu}\in\mathbb{R}^n} \left\{ \|A_S\,\boldsymbol{\mu} - \boldsymbol{b}_S\| \right\}, \tag{1.1}$$

where the over-determined matrix A_S is sparse, and all its nonzero entries involve derivatives of the measured displacement. For example, in the frequency domain problem in 1D space considered in (Davies *et al.*, 2019, Section 4), the underlying equation is

$$\big(v'(x)\,\mu(x)\big)' = -\omega^2\, v(x)\,, \quad x \in (0,\,1)\,, \tag{1.2}$$

where $v(x)$ is the (measured/simulated) displacement, $\mu(x)$ the shear modulus (which is to be found), ω a nondimensionalized frequency proportional to the frequency of the underlying mechanical oscillations, and where a prime denotes a derivative with respect to x. The measured displacement is the average over an interval of length $\Delta x = 1/J$, and A_S in (1.1) is formed of a small number (typically around 5) of matrices $A^{(k)} \in \mathbb{R}^{(J-2)\times(J-1)}$ stacked vertically. Each matrix $A^{(k)}$ is bidiagonal with the nonzero entries in row j being a_j and $-a_{j+1}$ with $a_j = \Big(v(x_{j+1/2}) - v(x_{j-1/2})\Big)\,/\Delta x \approx v'(x_j)$. If the error in the measured displacement values is typically of size ε, then the error in the nonzero entries of A_S in (1.1) will be around $\varepsilon/\Delta x$, and so the method's error increases as either ε grows or the mesh Δx is refined, and in either case the components of the calculated solution $\boldsymbol{\mu}$ are reduced. An additional but less serious drawback of the 'stacked frequency' approach of Davies *et al.* (2019) and Davies & Sack (2020) is that it assumes that the underlying model is purely elastic—if it is not, then the measured shear modulus can depend on the oscillation frequency, and blending different frequencies into a single calculation obscures this.

The new MRE approximation proposed here is a stacked matrix method that

(i) uses the same underlying oscillation frequency for each matrix $A^{(k)}$, and

(ii) takes advantage of the excellent function approximation properties of NNs with a single hidden layer (see e.g. (Lu *et al.*, 2021, Section 2.4) and the references therein) to approximate the noisy measured/simulated data whose first derivatives can then be evaluated by automatic differentiation.

Point (ii) is the most significant new development and results in a system (1.1) for which the error in the nonzero components of each $A^{(k)}$ is no longer proportional to $1/\Delta x$. As well as being a lot more accurate for noisy data, it also means that the method's accuracy improves as the mesh is refined (methods which use finite differences for the terms a_j become less accurate as the mesh is refined unless the noise level is very low). This is important because it means that future advances in MRI technology that provide higher resolution measurements will directly translate into more accurate results for medical practitioners. Point (i) means that the shear modulus can be calculated from (separate) measurements at the same mechanical oscillation frequency, which will enable thorough testing at different frequencies to determine whether or not a tissue's shear modulus is frequency-dependent. If so, then appropriate viscous terms can be added in to the underlying mathematical model to provide a more accurate representation.

Because the aim here is to show the potential of the new method, we will restrict attention to model problems in 1D space as a prototype (noting that the methods of Davies *et al.* (2019) and Davies & Sack (2020) tend to work better as the number of space dimensions increases). We begin the next section with a brief summary of the stacked matrix approach for the new frequency and time domain versions of the 1D problem together with extensive numerical tests which show that they perform extremely well even at very high noise levels, and neither version requires pre-smoothing. The NN approach used to approximate the data and derivatives is described in Section 3. Its implementation using standard NN packages is straightforward, and a short code using Matlab's Deep Learning Toolbox is appended.

2. Underlying model and approximation methods

The time-dependent 1D model problem considered in Davies & Sack (2020) is to find the shear modulus $\mu(x)$ given measured values of the 2π time-periodic function $u(x,t)$ such that

$$\left(u_x\,\mu\right)_x = \omega^2\,\ddot{u}\,, \quad x \in (0,\,1)\,, \quad t \in (0,\,2\pi)\,, \tag{2.1}$$

where an overdot denotes a derivative with respect to time t and a subscript $_x$ denotes a space derivative. The problem has been nondimensionalized and ω is proportional to the underlying mechanical oscillation frequency (which is of the order of 30–100 Hz).

2.1 *Stacked matrix approximation for the frequency domain problem*

We first outline the solution algorithm for the frequency domain version (1.2) of this problem by setting $u(x,\,t) = v(x)\,\cos t$ (see (Davies *et al.*, 2019) for full details). The underlying approximation of (1.2) uses finite volumes based on a staggered grid, with v recorded at interval midpoints $x_{j-1/2}$ for $j = 1:J$, and its derivative $a(x) = v'(x)$ and $\mu(x)$ evaluated at the interior nodes x_j for $j = 1:J-1$ (where $x_r = r\,\Delta x$). This gives the underdetermined linear system

$$A\,\boldsymbol{\mu} = \boldsymbol{b} \tag{2.2}$$

for $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_{J-1})^T$, where $b_j = \omega^2\,\Delta x\,v(x_{j+1/2})$ for $j = 1:J-2$ and $A \in \mathbb{R}^{(J-2)\times(J-1)}$ is the bidiagonal matrix with entries

$$A = \begin{pmatrix} a_1 & -a_2 & 0 & 0 & \ldots & 0 & 0 \\ 0 & a_2 & -a_3 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & a_{J-2} & -a_{J-1} \end{pmatrix}$$

with $a_j \approx v'(x_j)$. The least-squares system (1.1) (with $n = J - 1$) is obtained by stacking up several such A and $\boldsymbol{b}$, each calculated from different simulated measurements of $v(x)$. As noted above the method in Davies *et al.* (2019) uses a different frequency ω for each simulation and the a_j are obtained from central finite difference approximations; here we use the **same** value of ω for each simulation (with randomly varying boundary values) and use the derivative of an NN to evaluate the a_j. The algorithm to compute A_S and $\boldsymbol{b}_S$ in the simulation is as follows:

- Choose ω, J, the exact shear modulus $\mu(x)$, the size ε of noise and the number q of matrices/vectors to stack.
- For each individual simulation choose random boundary values for $v(0)$ and $v(1)$ in the interval $[-1, 1]$ and calculate a very accurate forward solution $v(x)$ of (1.2) at the interval midpoints $x_{j-1/2}$ (either numerically, or it can be done exactly if $\mu(x)$ is piecewise constant). Then add noise to each component, replacing $v_{j-1/2}$ by $v_{j-1/2} + \varepsilon_j$ for pseudo-random $\varepsilon_j \in [-\varepsilon, \varepsilon]$.
- Calculate $a_j \approx v'(x_j)$: this is done as a first central difference approximation in Davies *et al.* (2019), but here we use an NN approximation (described in Section 3). Then construct A and $\boldsymbol{b}$ for this simulation and add them into the stacked versions A_S and $\boldsymbol{b}_S$.
- After q runs the system matrix A_S and vector $\boldsymbol{b}_S$ are complete, then solve the least-squares system $A_S \boldsymbol{\mu} = \boldsymbol{b}_S$ for the calculated shear modulus (evaluated at the interior nodes) and compare it with the exact value $\mu(x)$ at the nodepoints $x_j, j = 1 : J - 1$.

2.1.1 *Numerical results (frequency domain).* For easy comparison with Davies *et al.* (2019), we show results obtained from a stacked system of $q = 5$ submatrices with nondimensionalized frequency $\omega = 40$ (this is typically the lowest value of ω used in Davies *et al.* (2019)). The red solid line in Fig. 1 is the calculated value of $\boldsymbol{\mu}$ with $J = 64$ (top) and $J = 256$ (bottom) with a small amount of added noise ($\varepsilon = 0.05$) using the NN approach. The exact shear modulus $\mu(x)$ is the piecewise constant shape shown by the black dotted line, and the blue dashed line shows the shear modulus calculated using the method of Davies *et al.* (2019) but with all the submatrices found at this single frequency (there appears to be no difference in results between the multi-frequency approach of Davies *et al.* (2019) and using a single frequency). As described above, the method (Davies *et al.*, 2019) is about as effective as our new approach when the mesh is coarse (with this level of added noise), but refining the mesh at a fixed noise level destroys its performance, as is clearly shown in the figure.

Increasing the size of added noise destroys the performance of the finite-difference approach of Davies *et al.* (2019) but not the new NN method, as illustrated in Fig. 2. Here the mesh is fixed at $J = 128$, and both methods perform similarly with no added noise (top plot), but adding even a small amount of noise ($\varepsilon = 0.1$) stops the method of Davies *et al.* (2019) working. There is no degradation in the performance of the new NN method, and it continues to be remarkably robust for extremely high values of added noise. For example, the top plot of Fig. 3 compares the new NN method with no added noise (red solid) to that with $\varepsilon = 3$ (red dashed). The bottom plot shows the exact (black dotted) and noisy (red dashed) values of the displacement $v(x)$ used for one of the five stacked matrix simulations—the amplitude of the exact $v(x)$ used for each simulation varies (it depends on the random boundary values), but it is an $\mathcal{O}(1)$ quantity, and typically in the range 3–8.

All these numerical tests use Matlab's default NN training configuration of one hidden layer with 10 parameters (see Section 3 for more details) and this works well with low or moderate forcing frequencies ω, but a more highly oscillatory displacement function $v(x)$ will require more than this for a good

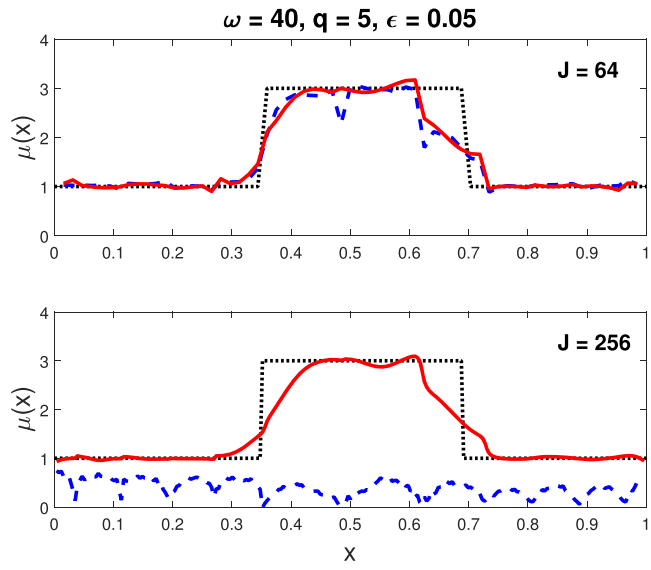

FIG. 1. Plot of calculated $\boldsymbol{\mu}$ (red solid line) obtained using the frequency domain NN method with a stacked system of $q = 5$ submatrices with $\omega = 40$ when $J = 64$ (top), $J = 256$ (bottom) and added noise-level $\varepsilon = 0.05$. The exact value $\mu(x)$ is shown as a black dotted line and the blue dashed line shows $\boldsymbol{\mu}$ calculated using the method of Davies *et al.* (2019).

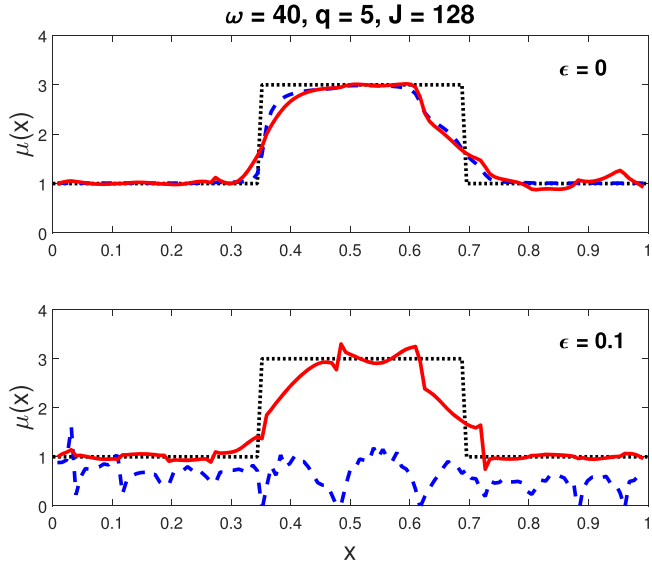

FIG. 2. Plot of calculated $\boldsymbol{\mu}$ (red solid line) obtained using the frequency domain NN method with a stacked system of $q = 5$ submatrices with $\omega = 40$ when $J = 128$ with no added noise (top) and added noise-level $\varepsilon = 0.1$ (bottom). The exact value $\mu(x)$ is shown as a black dotted line and the blue dashed line shows $\boldsymbol{\mu}$ calculated using the method of Davies *et al.* (2019).

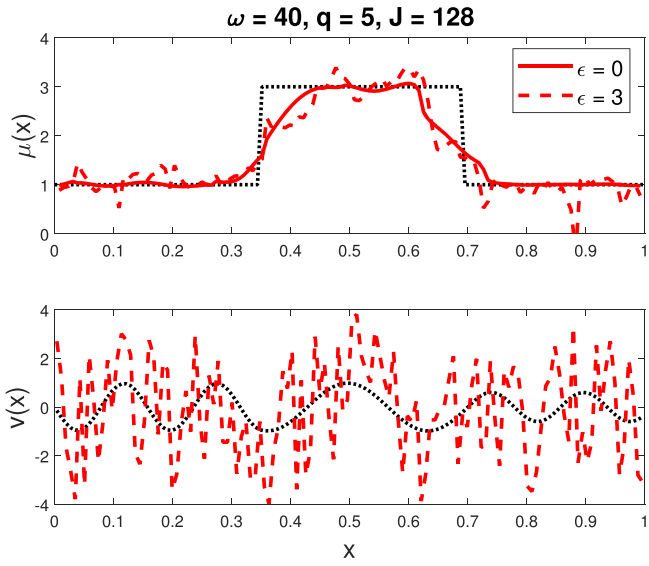

FIG. 3. Top: calculated $\boldsymbol{\mu}$ obtained using the frequency domain NN method with a stacked system of $q = 5$ submatrices with $\omega = 40$ when $J = 128$ with no added noise (red solid line) and added noise-level $\varepsilon = 3$ (red dashed line). Bottom: exact (black dotted) and noisy (red dashed) values of the displacement $v(x)$ used for one of the five stacked-matrix simulations.

approximation. In practice, the mechanical oscillation frequency cannot be too high, for patient comfort, and it is more important to develop methods which work well at lower frequencies.

2.2 *Stacked matrix approximation for the time domain problem*

The exact solution of (2.1) with appropriate initial and boundary conditions is $u(x, t) = v(x) \cos t$, where $v(x)$ can be calculated very accurately and efficiently for a given 'exact' shear modulus $\mu(x)$ as described in Davies & Sack (2020). The noisy simulated solution on a space-time observation mesh with size $\Delta x = 1/J$ and $h = 2\pi/M$ (where typically $M = 8$) is then

$$u_j^m = v(x_{j-1/2}) \cos(t_m) + \varepsilon_j^m \,, \quad j = 1 : J \,, \; m = 0 : M - 1 \,,$$

where again $x_r = r\,\Delta x$, $t_m = mh$ and each $\varepsilon_j^m \in [-\varepsilon, \varepsilon]$ is a pseudo-random error term. The usual solution method is to take the discrete Fourier transform (DFT) in time of the measured displacement u in (2.1) and throw away all but one of the two dominant DFT components, but as discussed in Davies & Sack (2020), both dominant components should be kept (they are complex conjugates). Discretising in space then leads to a system like (2.2) in which the matrix $A \in \mathbb{C}^{2\,(J-2)\times(J-1)}$ is block diagonal and the entries a_j are now column vectors of length 2 (the two components are complex conjugates). Similarly the right-hand side vector $\boldsymbol{b}$ is now complex, and of length $2\,(J-2)$, formed of consecutive entries that are complex conjugates. This matrix-vector system is again stacked (as described for the frequency domain problem above), giving the least-squares formulation

$$\min_{\boldsymbol{\mu} \in \mathbb{C}^n} \left\{ \| A_S\, \boldsymbol{\mu} - \boldsymbol{b}_S \| \right\}, \tag{2.3}$$

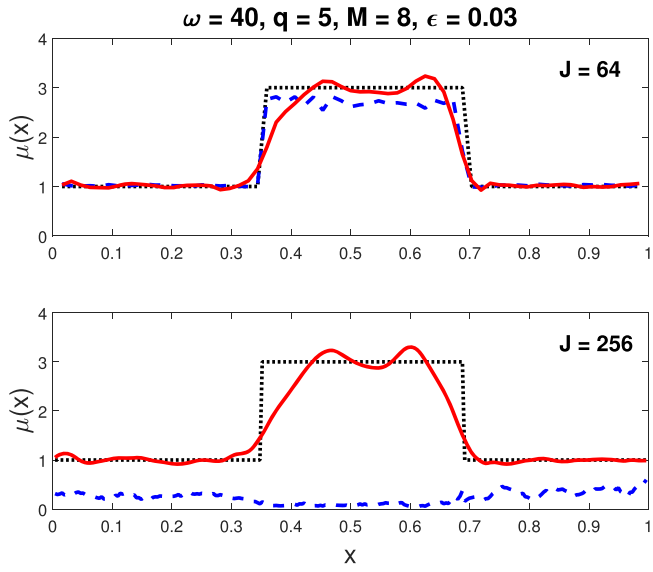

FIG. 4. Plot of calculated $\boldsymbol{\mu}$ (red solid line) obtained for the time-domain problem (2.1) using the new NN method with $M = 8$ time samples per period, a stacked system of $q = 5$ submatrices with $\omega = 40$ when $J = 64$ (top) $J = 256$ (bottom) and added noise-level $\varepsilon = 0.03$. The exact value $\mu(x)$ is shown as a black dotted line and the blue dashed line shows $\boldsymbol{\mu}$ calculated using the method of Davies & Sack (2020).

in terms of the complex 2-norm. It is shown in Davies & Sack (2020) that the structure of this problem results in the computed shear modulus $\boldsymbol{\mu}$ being real up to rounding error—it satisfies the normal equation

$$A_S^H A_S \boldsymbol{\mu} = A_S^H \boldsymbol{b}_S .$$

As for the frequency domain case above we use an NN approximation instead of computing the entries of the matrix A by finite differences. The details are given in Section 3, but the key point is to separately calculate an NN approximation in space of the real and imaginary parts (and their derivatives) of the dominant DFT component of u, and form the pairs of rows of A and $\boldsymbol{b}$ from this and its complex conjugate.

2.2.1 *Numerical results (time domain).* We show results obtained from a stacked system of $q = 5$ submatrices with $M = 8$ time samples a period, and using the single nondimensionalized frequency $\omega = 40$ (there appears to be no difference in results between Davies & Sack (2020) and using a single frequency for the stacked matrices). The behaviour of the new NN time-domain approximation is very similar to that of the frequency domain problem described above. Figure 4 shows that its performance does not degrade as the space mesh is refined, unlike the method of Davies & Sack (2020), whose solution is close to zero when $J = 256$ even with a very small amount of added noise. Its robustness with a small amount of added noise (when $J = 128$) is shown in Fig. 5 –although note that the solution calculated using the method of Davies & Sack (2020) is more accurate in the top plot with no added noise.

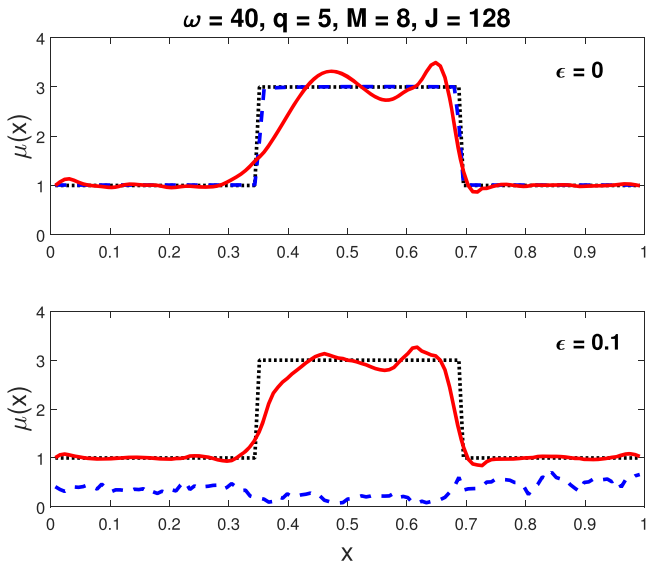

FIG. 5. Plot of calculated $\boldsymbol{\mu}$ (red solid line) obtained using the time domain NN method for a stacked system of $q = 5$ submatrices with $\omega = 40$ and $M = 8$ when $J = 128$ with no added noise (top) and added noise-level $\varepsilon = 0.1$ (bottom). The exact value $\mu(x)$ is shown as a black dotted line and the blue dashed line shows $\boldsymbol{\mu}$ calculated using the method of Davies & Sack (2020).

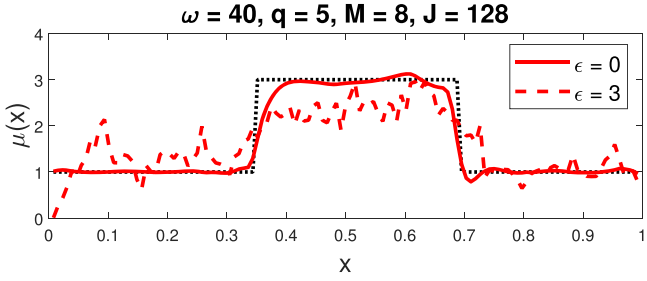

FIG. 6. Plot of calculated $\boldsymbol{\mu}$ obtained using the time domain NN method for a stacked system of $q = 5$ submatrices with $\omega = 40$ and $M = 8$ when $J = 128$ with no added noise (red solid line) and added noise-level $\varepsilon = 3$ (red dashed line).

The time domain version of the new method is also robust to extremely high noise levels, as shown in Fig. 6 in which the added noise has $\varepsilon = 3$.

3. Neural net approximation of the measured data and its derivatives

This section is aimed at applied mathematicians who are interested in using off-the-shelf software to easily obtain good approximations from noisy data, rather than at experts in NNs or deep learning. The appendix contains a short code based on Matlab's Deep Learning Toolbox, but lots of other platforms are available, e.g. the Python library DeepXDE is described in the excellent review (Lu *et al.*, 2021). The

article (Higham & Higham, 2019) is also an excellent introduction to the subject, and clearly explains how an NN is 'trained'.

In the frequency domain problem of Section 2.1, the aim is to find a function $y(x)$ to approximate the noisy data $\left\{(x_{j-1/2},\, v_{j-1/2})\right\}_{j=1}^{J}$ and then use automatic differentiation to evaluate $a_j = y'(x_j)$. This is straightforward using an NN with one hidden layer and a single output. In this case the function fit is obtained using $3N + 1$ parameters, comprised of two weight vectors (the columns $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ of the matrix $\boldsymbol{W} \in \mathbb{R}^{N\times 2}$), one bias (column) vector $\boldsymbol{B} \in \mathbb{R}^N$ and the real scalar bias B_L (all calculations reported here use the default of $N = 10$, so there are 31 parameters). It is common to first affinely map the two data intervals $[x_{1/2},\, x_{J-1/2}]$ and $[v^L,\, v^R]$ (where $v^L = \min\left\{v_{j-1/2}, j = 1 : J\right\}$ and $v^R = \max\left\{v_{j-1/2}, j = 1 : J\right\}$) onto $[-1,\, 1]$. These maps are respectively $S_x(t) = \alpha_x\, t + \beta_x$ and $S_y(t) = \alpha_y\, t + \beta_y$ for

$$\alpha_x = \frac{2}{x_{J-1}}\,, \quad \beta_x = -\frac{x_J}{x_{J-1}}\,, \quad \alpha_y = \frac{2}{v_R - v_L}\,, \quad \beta_y = -\frac{(v_R + v_L)}{v_R - v_L}\,.$$

The data-fitted approximation at $x \in [x_{1/2},\, x_{J-1/2}]$ is $y(x)$ obtained in the following steps:

$$\begin{aligned}
\widehat{x} &= S_x(x) \in [-1,\, 1] \quad \text{(scale } x - \text{values)} \\
\boldsymbol{Z} &= \boldsymbol{W}_1\, \widehat{x} + \boldsymbol{B} \in \mathbb{R}^N \quad \text{(hidden layer calculation on scaled values)} \\
\widehat{y} &= \boldsymbol{W}_2^T\, \sigma(\boldsymbol{Z}) + B_L \in \mathbb{R} \quad \text{(scaled output)} \\
y &= S_y^{-1}(\widehat{y}) \in \mathbb{R} \quad \text{(reverse the scaling of the fitted output)}\,,
\end{aligned}$$

where the 'cut-off' function σ is applied componentwise and regarded as a column vector (here σ is the hyperbolic tangent, tanh, although other choices are available as described in Higham & Higham (2019)). In component form, the middle two terms give

$$\widehat{y} = B_L + \sum_{k=1}^{N} W_{k2}\, \sigma\left(W_{k1}\, \widehat{x} + B_k\right).$$

Training the network involves optimising the distance $\left|v_{j-1/2} - y(x_{j-1/2})\right|$, $j = 1 : J$ over the parameters. There are various possibilities for this, and the examples shown in the previous sections all use the Levenberg–Marquardt backpropagation option in the toolbox. Once the parameters have been obtained, it is straightforward to calculate the derivative $y'(x)$ using the chain rule on the above steps (this is termed 'automatic differentiation'; see (Lu *et al.*, 2021) for more details). That is

$$\frac{\mathrm{d}y}{\mathrm{d}x} = \frac{\mathrm{d}y}{\mathrm{d}\widehat{y}}\,\frac{\mathrm{d}\widehat{y}}{\mathrm{d}\widehat{x}}\,\frac{\mathrm{d}\widehat{x}}{\mathrm{d}x} = \frac{\alpha_x}{\alpha_y}\,\frac{\mathrm{d}\widehat{y}}{\mathrm{d}\widehat{x}}$$

and if σ is the hyperbolic tangent function, then

$$\frac{\mathrm{d}\widehat{y}}{\mathrm{d}\widehat{x}} = \sum_{k=1}^{N} W_{k1}\, W_{k2}\, \mathrm{sech}^2\left(W_{k1}\,\widehat{x} + B_k\right).$$

The derivatives are evaluated at the interior nodes x_j and the entries $a_j = y'(x_j)$ are used in the matrix A of Section 2.1. Data fitting for the time domain problem of Section 2.2 is very similar, although now the derivative is taken of components of the DFT of the noisy data, which are complex. This is done by finding separate NN approximations for the real and imaginary parts (y_R and y_I, respectively) of one of the dominant DFT components and then the two-component column vector $\boldsymbol{a}_j$ used for the matrix A of Section 2.2 is

$$\boldsymbol{a}_j = \begin{pmatrix} y'_R(x_j) + i\, y'_I(x_j) \\ y'_R(x_j) - i\, y'_I(x_j) \end{pmatrix} \quad \text{for } j = 1 : J.$$

A short Matlab code that uses built-in routines to calculate the fitted function and its derivative (at a vector of points) to a noisy version of $y = x^2$ is given in the Appendix.

4. Discussion and conclusions

The numerical results for the 1D model problems in both the frequency (1.2) and time (2.1) domain are very encouraging. In particular, they show that off-the-shelf packages for NNs with one hidden layer can provide a good enough approximation of the derivative of very noisy displacement data to give good results when combined with the (robust) stacked MRE algorithms of Davies *et al.* (2019) and Davies & Sack (2020). The new methods need no other data processing or smoothing, and the results improve as the spatial mesh is refined. The results presented here are intended as a 'proof of concept' rather than a detailed investigation of the best way to approximate the displacement derivative, and have focused on the NN approach because there are many easily accessible NN codes available. Also, because the individual matrices stacked into the over-determined systems (1.1) and (2.3) can be obtained using the same underlying oscillation frequency, the new approach could be used to investigate frequency dependence.

The much improved performance of the new NN-based approach for these 1D prototype problems indicates that it is well worth investigating a similar approach for 2D and 3D problems. The 2D and 3D versions of the stacked (finite difference) MRE algorithms are described and tested in detail in Davies *et al.* (2019) and Davies & Sack (2020), and developing algorithms that instead use use an NN approximation to calculate the space derivatives of 2D or 3D displacement data is the focus of ongoing research.

References

Arunachalam, S. P., Rossman, P. J., Arani, A., Lake, D. S., Glaser, K. J., Trzasko, J. D., Manduca, A., McGee, K. P., Ehman, R. L. & Araoz, P. A. (2017) Quantitative 3D magnetic resonance elastography: comparison with dynamic mechanical analysis. *Magn. Reson. Med.*, **77**, 1184–1192.

Barnhill, E., Davies, P. J., Ariyurek, C., Fehlner, A., Braun, J. & Sack, I. (2018) Heterogeneous multifrequency direct inversion (HMDI) for magnetic resonance elastography with application to a clinical brain exam. *Med. Image Anal.*, **46**, 180–188.

Davies, P. J. & Sack, I. (2020) A stacked frequency approach for inhomogeneous time-dependent MRE: an inverse problem for the elastic shear modulus. *IMA J. Appl. Math.*, **86**, 121–145.

Davies, P. J., Barnhill, E. & Sack, I. (2019) The MRE inverse problem for the elastic shear modulus. *SIAM J. Appl. Math.*, **79**, 1367–1388.

Dittmann, F., Hirsch, S., Tzschätzsch, H., Guo, J., Braun, J. & Sack, I. (2016) In vivo wideband multifrequency MR elastography of the human brain and liver. *Magn. Reson. Med.*, **76**, 1116–1126.

Doyley, M. M. (2012) Model-based elastography: a survey of approaches to the inverse elasticity problem. *Phys. Med. Biol*, **57**, R35–R73.

Hanke, M. & Scherzer, O. (2001) Inverse problems light: numerical differentiation. *American Mathematical Monthly*, **108**, 512–521.

Higham, C. F. & Higham, D. J. (2019) Deep learning: an introduction for applied mathematicians. *SIAM Rev.*, **61**, 860–891.

Lu, L., Meng, X., Mao, Z. & Karniadakis, G. E. (2021) DeepXDE: a deep learning library for solving differential equations. *SIAM Rev.*, **63**, 208–228.

Manduca, A., Oliphant, T. E., Dresner, M. A., Mahowald, J. L., Kruse, S. A., Amromin, E., Felmlee, J. P., Greenleaf, J. F. & Ehman, R. L. (2001) Magnetic resonance elastography: non-invasive mapping of tissue elasticity. *Med. Image Anal.*, **5**, 237–254.

McLaughlin, J. R., Zhang, N. & Manduca, A. (2010) Calculating tissue shear modulus and pressure by 2D log-elastographic methods. *Inverse Probl.*, **26**, 085007.

Park, E. & Maniatty, A. M. (2006) Shear modulus reconstruction in dynamic elastography: time harmonic case. *Phys. Med. Biol.*, **51**, 3697–3721.

Sánchez, C., Drapaca, C., Sivaloganathan, S. & Vrscay, E. (2010) Elastography of biological tissue: direct inversion methods that allow for local shear modulus variations. *Image Anal. Recognit.*, **Pt II**, 195–206.

A. Appendix: calculating the derivative of fitted data

The following short code uses Matlab's Deep Learning Toolbox to calculate the derivative of the NN fit to a noisy approximation of $y = x^2$. The input parameters are the vector $\boldsymbol{x}$ of length J of interval midpoint values in (0, 1) and the noisy data is $\boldsymbol{y}^{\rm dat}_{j-1/2} = \boldsymbol{x}^2_{j-1/2} + \boldsymbol{\varepsilon}_j$ where each error component $\varepsilon_j \in [-\varepsilon, \varepsilon]$. The code calculates and plots the NN approximation $\boldsymbol{y}^{\rm fit}$ (top plot) to the noisy data and its derivative (lower plot) at the vector of interior nodes x_j for $j = 1 : J - 1$. The blue dashed 'F diff' line in the lower plot is the finite difference approximation of the derivative of the fitted data $\boldsymbol{y}^{\rm fit}$—it confirms that the derivative is correctly calculated (for a sufficiently fine mesh size J; in this case, its plot will be indistinguishable from that of the derivative).

```
% FNNapprox - Matlab code to approximate a noisy version of y = x^2 using a
% model with one hidden layer, and calculate its derivative
%

clear,  close all, format short e
```

```
J = 256;  % # intervals - will use midpoint values for the function fit
eps = 0.1; % error for random data

xmid = [1:2:2*J-1]/(2*J);  % ROW vector of midpoint values
yex = xmid.^2;  % exact values of y = x^2 (ROW)
ydat = yex + (2*eps)*(rand(1,J)-0.5);  % noisy values of y = x^2 (ROW)

%% Fit neural network using packages from the Deep Learning Toolbox
%  (the app is nftool)

% Choice of training function
% For a list of possible training functions type: help nntrain
trainFcn = 'trainbr';  % this is Levenberg-Marquardt backpropagation

% Choose network size & create it
hiddenLayerSize = 10;   % parameter dimension N
net = fitnet(hiddenLayerSize,trainFcn);

% Select division of data (training, validation, testing)
net.divideParam.trainRatio = 70/100;    % default values
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Train the network
[net,tr] = train(net,xmid,ydat);

%% Use net parameters to find function fit to (noisy) data
%  the shorthand command is yfit = net(xmid);

W1 = net.IW{1}; % COL N-vector (1st col of array W in Sec 3)
B = net.b{1,1}; % COL N-vector (as in Sec 3)

W2 = net.LW{2,1}; % ROW N-vector (transpose of 2nd col of array W in Sec 3)
BL = net.b{2,1};  % SCALAR (as in Sec 3)

xgain = net.inputs{1}.processSettings{1}.gain;  % alpha_x from Sec 3
ygain = net.outputs{2}.processSettings{1}.gain; % alpha_y from Sec 3

% map xmid and ydat onto [-1,1]
[xhat,xps] = mapminmax(xmid,-1,1);  % need xhat & use xps for deriv calc
[yhat,yps] = mapminmax(ydat,-1,1);  % yhat is dummy, need yps

Z = W1*xhat + B;  % MATRIX: N x J
yhat = W2*tansig(Z) + BL;   % ROW vector: length N
yfit = mapminmax('reverse',yhat,yps);  % reverse scaling to get fit to data
```

```
%% Calculate d (yfit)/dx at (different) row vector of x values

xd = [1:J-1]/J;  % interior nodes (ROW of length J-1)
xhat = mapminmax('apply',xd,xps);  % scale vector xd
Z = W1*xhat + B*ones(1,J-1);  % MATRIX: N x (J-1)
dydx = W2*(W1.*sech(Z).^2)*(xgain/ygain);  % derivative of yfit at xd (ROW)

fd = (yfit(2:J) - yfit(1:J-1))*J;
% fd is finite difference approx of derivative of fitted data

%% output & plot results

subplot(2,1,1), plot(xmid,yex,'k:', xmid,ydat,'b--', xmid,yfit,'r-')
ylabel('y', 'fontsize',14)
tstr1 = ['fit to y = x^2: J = ', int2str(J)];
tstr2 = [', \epsilon = ',num2str(eps,'%0.2g')];
title([tstr1,tstr2], 'fontsize',14)

legend('y-exact', 'y-data', 'y-fit', 'Location','NorthWest')

%  top plot shows function values at interval midpoints
%    exact: black dotted
%    noisy data: blue dashed
%    fit to noisy data: red solid

subplot(2,1,2), plot(xd,2*xd,'k:', xd,fd,'b--', xd,dydx,'r-')
xlabel('x', 'fontsize',14)
ylabel('dy/dx', 'fontsize',14)
legend('exact', 'F diff', 'y-fit', 'Location','SouthEast')

%  bottom plot shows derivative values at interior nodes
%    exact: black dotted
%    fitted data: blue dashed (finite difference approx)
%    fit to noisy data: red solid

dyerr = max(abs(fd-dydx));
disp(['L_\infty fit error: ', num2str(max(abs(dyerr)))])
```