A Plugin Framework for Extending the Simulation Capabilities of FEBio

Steve A. Maas,[1] Steven A. LaBelle,[1] Gerard A. Ateshian,[2] and Jeffrey A. Weiss[1,*]
[1]Department of Biomedical Engineering, and Scientific Computing and Imaging Institute, University of Utah, Salt Lake City, Utah and [2]Department of Mechanical Engineering, Columbia University, New York, New York

ABSTRACT The FEBio software suite is a set of software tools for nonlinear finite element analysis in biomechanics and biophysics. FEBio employs mixture theory to account for the multiconstituent nature of biological materials, integrating the field equations for irreversible thermodynamics, solid mechanics, fluid mechanics, mass transport with reactive species, and electrokinetics. This communication describes the development and application of a new "plugin" framework for FEBio. Plugins are dynamically linked libraries that allow users to add new features and to couple FEBio with other domain-specific software applications without modifying the source code directly. The governing equations and simulation capabilities of FEBio are reviewed. The implementation, structure, use, and application of the plugin framework are detailed. Several example plugins are described in detail to illustrate how plugins enrich, extend, and leverage existing capabilities in FEBio, including applications to deformable image registration, constitutive modeling of biological tissues, coupling to an external software package that simulates angiogenesis using a discrete computational model, and a nonlinear reaction-diffusion solver. The plugin feature facilitates dissemination of new simulation methods, reproduction of published results, and coupling of FEBio with other domain-specific simulation approaches such as compartmental modeling, agent-based modeling, and rigid-body dynamics. We anticipate that the new plugin framework will greatly expand the range of applications for the FEBio software suite and thus its impact.

INTRODUCTION

The FEBio project (1) was launched in 2007 to overcome the lack of a general purpose finite element (FE) software for solving problems in computational biomechanics. This deficiency hampered research progress, dissemination, and sharing of models and results. Commercial FE codes such as Ansys/Fluent (ANSYS, Canonsburg, PA), ABAQUS (Dassault Systèmes, Vélizy-Villacoublay, France), and Comsol Multiphysics (COMSOL, Burlington, MA) that were used historically for research in computational biomechanics are not specifically designed to serve the field, lack important simulation capabilities, are costly, and are closed source, which makes them difficult to verify and extend. These limitations led many researchers to develop custom FE codes for simulating domain-specific problems (2). Most of these custom codes were never distributed to the community.

The authors and many others in the research community felt there was a genuine need for a new FE simulation tool that would be designed by and for the biomechanics community. FEBio was designed around three pillars. First, it specifically targets the biomechanics community by focusing on simulation capabilities that are relevant in the field. This includes, for example, accurate constitutive relations for mechanical, transport, and electrokinetic properties of tissues and cells; the ability to easily model anisotropy and inhomogeneity; and the ability to prescribe boundary conditions and loading scenarios to model the complex interactions between biological structures. Second, both installation packages and the source code are freely available and designed such that it is easy for researchers to implement new algorithms (e.g., new constitutive models). This greatly reduced the need for new custom-written codes and overcame the problems with verification and repeatability of results. Third, we place substantial emphasis on thorough documentation, support, and outreach to the community, which has made it much easier than before for researchers to develop new ideas and share them with others.

Recently, we developed a plugin framework to allow users to extend the standard feature set of FEBio with less effort and to couple FEBio with other simulation codes. Although users can download and modify the FEBio source

Submitted May 2, 2018, and accepted for publication September 12, 2018.
*Correspondence: jeff.weiss@utah.edu
Editor: Vivek Shenoy.
https://doi.org/10.1016/j.bpj.2018.09.016

code directly, building the entire source code can be challenging, especially because of its dependency on third-party libraries. Further, disseminating custom modifications to the source code may be challenging because changes made by users may become incompatible with future releases of FEBio. The plugin framework overcomes these problems because plugins can be developed independently from the FEBio source code. In addition, the plugin framework facilitates dissemination and sharing because plugins can be distributed separately from the FEBio source code and installation packages (https://febio.org/plugins).

The objective of this communication is to introduce FEBio and the capabilities of the new plugin framework to the biophysics community. The exposition covers the modular framework of FEBio and its kernel mechanism, which were central to the development of the plugin framework. Several plugin examples illustrate the generality and versatility of the framework, including a material plugin based on a recent proposal for a novel constitutive model for describing a fibrous material with deformation induced anisotropy (3), an implementation of the hyperelastic warping algorithm for deformable image registration (4), a plugin that couples a discrete model of angiogenesis with FEBio to simulate the interactions between angiogenic microvessels and the ECM (5–8), and a plugin for solving reaction-diffusion-convection problems.

MATERIALS AND METHODS

Overview of FEBio

FEBio uses the FE method to discretize the equations for conservation of mass, linear momentum, and charge. The resulting equations allow fully coupled simulation of solid mechanics, solid-fluid mixtures (9), fluid mechanics (10), fluid-solid interactions, transport (11,12), reaction and diffusion of neutral and charged species (12,13), contact (11,14–17), prestrain (18), and growth and remodeling (13). The governing equations are formulated based on mixture theory. A mixture consists of any number of constituents α that may occupy the same region. The apparent mass density of constituent α is denoted by ρ^α. The complete set of governing equations can be summarized as follows:

$$\frac{\partial \rho^\alpha}{\partial t} + \mathrm{div}(\rho^\alpha \boldsymbol{v}^\alpha) = \widehat{\rho}^\alpha, \tag{1}$$

$$\rho^\alpha \boldsymbol{a}^\alpha = \mathrm{div}\boldsymbol{T}^\alpha + \rho^\alpha \boldsymbol{b}^\alpha + \widehat{\boldsymbol{p}}_d^\alpha, \tag{2}$$

$$\sum_\alpha - \boldsymbol{T}_d^\alpha : \boldsymbol{L}^\alpha + \widehat{\boldsymbol{p}}_d^\alpha \cdot \boldsymbol{u}^\alpha + \widehat{\rho}^\alpha \left(\mu^\alpha + \frac{1}{2} \boldsymbol{u}^\alpha \cdot \boldsymbol{u}^\alpha \right) \leq 0, \tag{3}$$

$$\boldsymbol{b}^\alpha = -(z^\alpha F_c / M^\alpha) \mathrm{grad}\psi \,, \tag{4}$$

and

$$\sum_\alpha z^\alpha \rho^\alpha \Big/ M^\alpha = 0. \tag{5}$$

Equation 1 represents the mass balance equation for constituent α with velocity $\boldsymbol{v}^\alpha$. Equation 2 is the balance of linear momentum for each constituent, where $\boldsymbol{a}^\alpha$ is the acceleration, $\boldsymbol{T}^\alpha$ is the Cauchy stress, $\boldsymbol{b}^\alpha$ is the body force, and $\widehat{\boldsymbol{p}}_d^\alpha$ is the internal momentum supply to constituent α due to interactions with all other mixture constituents. Equation 3 is the entropy inequality, which places constraints on the constitutive models. Here, $\boldsymbol{L}^\alpha$ is the rate of deformation tensor, $\boldsymbol{u}^\alpha = \boldsymbol{v}^\alpha - \boldsymbol{v}$, and $\boldsymbol{v}$ is the mixture velocity. The last term in this equation represents the rate form of the familiar constraint in chemical kinetics that reactions may proceed spontaneously only if they produce a net decrease in the mixture free energy. Electrokinetics is introduced by using Coulomb's law as a body force acting on electrically charged constituents, Eq. 4, where z^α is the charge number and M^α is the molar mass of constituent α, F_c is Faraday's constant, and ψ is the electric potential in the mixture. To solve for ψ, we employ the electroneutrality condition, Eq. 5. FE discretization of Eqs. 1, 2, 3, 4, and 5 above results in a nonlinear system of equations that is solved using Newton-based methods, including quasi-Newton methods such as Broyden–Fletcher–Goldfarb–Shanno (BFGS) and Broyden's method (19).

A more detailed overview of FEBio's features can be found in the FEBio User Manual and the FEBio Theory Manual (https://febio.org/febio-help/). An overview of FEBio development and its features with historical context can be found in our recent publication (2).

Overview of the plugin mechanism

A plugin is essentially a "dynamically" linked library (also known as a shared object on some operating systems) that is linked to the executable at runtime. They differ from "static" libraries, which are linked to the executable at build time. The main advantage of dynamic libraries is that they are not required when building the executable and thus can be developed and maintained independently of the executable.

In FEBio, plugins can be used to add new features, such as new constitutive models, boundary loads, body loads, tasks, plot data, and log data. They can also be used to implement an entirely new physics module, as illustrated by the FEBioChem plugin described below. The main advantage of a plugin versus extending the source code directly is that plugins are loaded at runtime and can be developed and maintained outside of the FEBio source code. At runtime, FEBio parses a configuration file. This file contains a list of all plugins that the user wants to use, and FEBio loads each plugin file in this list. During this process, the plugin is given a chance to initialize itself. During this initialization step, it registers the new features with FEBio. For each feature, metadata are passed to FEBio, such as a textual name and a list of parameters that FEBio can use to allocate and modify the data of the plugin features. After the registration process, FEBio can use the new features in the plugin. For instance, after loading a new material plugin, the input module will be able to process the definition of the new material in the input file.

A more detailed explanation of the plugin framework can be found in the Supporting Materials and Methods. Detailed instructions on how to create plugins can be found in the FEBio Developer's Manual (http://febio.org/support/). The specific plugins described below are available for download at the FEBio website (http://febio.org/plugins/).

RESULTS

FEBio supports a number of different types of plugins, including material, plot data, callback, task, and solver plugins. These are described in more detail in the Supporting Materials and Methods. The following sections provide examples of material, nonlinear constraint, task, and solver plugins. Each type of plugin is demonstrated by a specific example application.

Material plugins

FEBio offers a rich library of constitutive models for representing the solid component of mixtures, including isotropic and anisotropic hyperelasticity, viscohyperelasticity (20), specialized materials for modeling fibrous tissues, damage mechanics, and growth and remodeling. Nevertheless, the development of novel constitutive models for describing the material response of biological tissues remains an active area of research as our understanding of how the intricate interactions between different physical scales affect the macroscopic material response continues to grow. As a result, the ability to add novel constitutive models to FEBio has been a high priority. Making this task easier for our users was the main initial impetus for the development of the plugin framework.

A new constitutive model can be implemented via a material plugin. Constitutive models are used to evaluate solution-dependent quantities, such as stress, permeability, diffusivity, etc. The quantities that need to be defined by the plugin depend on the particular physics module that the plugin extends. For instance, when implementing a new constitutive model for structural mechanics, the plugin must implement the evaluation of the second-order Cauchy stress tensor and the fourth-order spatial elasticity tensor. Implementation of the often complicated tensorial expressions for these quantities is greatly simplified by the extensive tensor classes in FEBio (see Supporting Materials and Methods).

As an illustration, we describe a material plugin that implements a constitutive model for representing fibrous ECMs as proposed by Wang et al. (3). The constitutive model is designed to reproduce experimental observations of long-range force transmission by cells in collagen gels due to realignment of collagen fibrils during loading. This is achieved by representing two fiber families. The first family aligns with the principal axes of the deformation, producing an increasingly anisotropic response as the deformation increases in magnitude, whereas the second family provides an isotropic background stress that opposes alignment.

The strain-energy density function of this constitutive model is given by:

$$W = W_b + W_f. \tag{6}$$

Here, W_b captures the isotropic response:

$$W_b = \frac{\mu}{2}\left(\tilde{I}_1 - 3\right) + \frac{\kappa}{2}(J-1)^2, \tag{7}$$

where μ is the shear modulus, κ is the bulk modulus, $\tilde{I}_1$ is the first invariant of the deviatoric right deformation tensor $\tilde{\mathbf{C}} = J^{-2/3}\mathbf{C} = J^{-2/3}\mathbf{F}^T\mathbf{F}$, $\mathbf{F}$ is the deformation gradient, and $J = \det(\mathbf{F})$ is the volume ratio. W_f is the contribution from the aligned fibers:

$$W_f = \sum_{a=1}^{3} f(\lambda_a), \tag{8}$$

where the λ_a are the principal stretch ratios. The particular form of f can be found in the Supporting Materials and Methods.

To illustrate the behavior of the material, an isotropically contracting sphere representing a contracting cell of radius R was positioned at the center of a cube of size $L = 10\,R$. Cellular contraction was simulated by displacing the surface of the sphere toward its center to achieve a radial strain $u_0/R = 0.3$. The resulting normalized displacement profiles as a function of normalized radial distance from the surface of the sphere demonstrate that the Wang et al. constitutive model produces greater matrix displacements and results in a longer range of influence of the simulated cellular traction forces than a standard neo-Hookean material (Fig. 1 *A*). This is visualized in space by examining the fringe plots of total matrix displacement (Fig. 1, *B* and *C*). These results are similar to those obtained by the authors for spherical cells in a cylindrical matrix (Fig. 2 in (3)). Please see the Supporting Materials and Methods for further details of the constitutive model and the material coefficients used in the simulations.

Nonlinear constraint plugin

A nonlinear constraint enforces a nonlinear condition on field variables such as nodal displacements or concentrations. In FEBio, nonlinear constraints are enforced via an augmented Lagrangian method (21), which solves for the Lagrange multipliers using an iterative loop outside of the Newton nonlinear iterative loop. Contact between deformable bodies and incompressibility are typical examples of

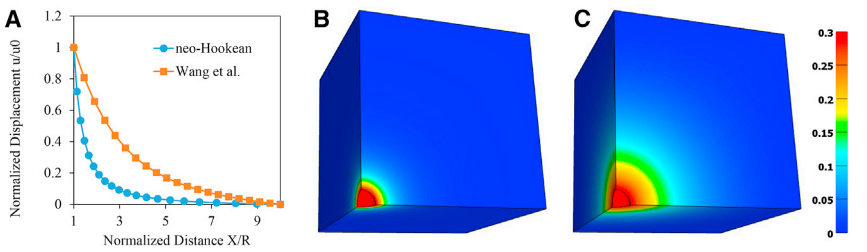


FIGURE 1 Simulation of force transmission to the ECM due to cellular contraction using a neo-Hookean constitutive model and the Wang et al. constitutive model (3). A 1/8 symmetry model was used for the calculations, simulating a spherical, isotropically contracting cell of radius R centered in a cube of size $L = 10R$. (*A*) Normalized displacement u/u_0 as a function of normalized distance X/R for a radial contractile strain of $u_0/R = 0.3$. (*B*) A contour plot of displacement magnitude for the neo-Hookean matrix is sho febio.org/plugins wn. (*C*) A contour plot of the displacement magnitude for the fibrous matrix. The results illustrate that the fibrous constitutive model proposed by Wang et al. exhibits long-range force transmission. To see this figure in color, go online.

nonlinear constraints that are encountered in biomechanics simulation. Nonlinear constraint plugins allow users to easily implement additional nonlinear constraints.

The FEWarp plugin below is an example of a nonlinear constraint plugin. Hyperelastic Warping is a deformable image registration method that is used to find the deformation map that aligns a template image data set with a target image data set (4,22–27). This technique not only enables markerless strain measurement from sequences of images but in many cases can be used to predict the stress field as well and compensate for missing or incomplete boundary conditions (22). Two or more image data sets are used. The data set in the reference configuration is referred to as the template, and one or more data sets in different deformed configurations serve as the target. An FE discretization and interpolation of the template image data is deformed into alignment with the target. An energy functional is defined, consisting of an image-based term that measures alignment and a hyperelastic strain energy term that serves to regularize the problem. Minimization of the energy functional produces alignment of the deformed template image with the target image. If accurate stress calculations are desired, realistic constitutive models and material properties can be used. If the focus is on strain measurements from image data only, this is not necessary, in which case the image data are treated as a hard constraint, enforced using an augmented Lagrangian method (28). We use successive Gaussian blurring to evolve the solution from the level of coarse features to fine details in the image data. One of the strengths of this approach is that the deformation can be obtained without the specific knowledge of the applied loads and boundary conditions to the model. Because the method is based on continuum mechanics, the resulting deformation is guaranteed to be diffeomorphic. The method has been applied successfully to characterize the mechanics of tissues such as ligaments (22) and the left ventricle of the heart (29).

In the example illustrated in Fig. 2, the FEWarp plugin was used to track the deformation of the left ventricle during simulation of diastole. A three-dimensional (3D) MRI image was taken at the start of diastole. A forward FE analysis resulted in a deformed model that simulates the condition at the end of diastole. From this deformed configuration, a target image was generated. Then, the warping algorithm was applied to the template image and attempted to recover the deformed target image to validate the warping approach.

Task plugin

At the highest level, after parsing the input file, FEBio executes a single task. The default task solves the model defined by the FEBio input file, but other tasks can be executed as well. For instance, the optimization module in FEBio is implemented as a task that calls the default solver task repeatedly while minimizing an objective function by modifying model parameters. In essence, a task defines the outer loop that controls the work that FEBio performs. A task plugin allows the user to dictate how FEBio is used for a particular task at a high level.

The AngioFE plugin (7) is an example of a task plugin that illustrates how FEBio can be linked to other libraries

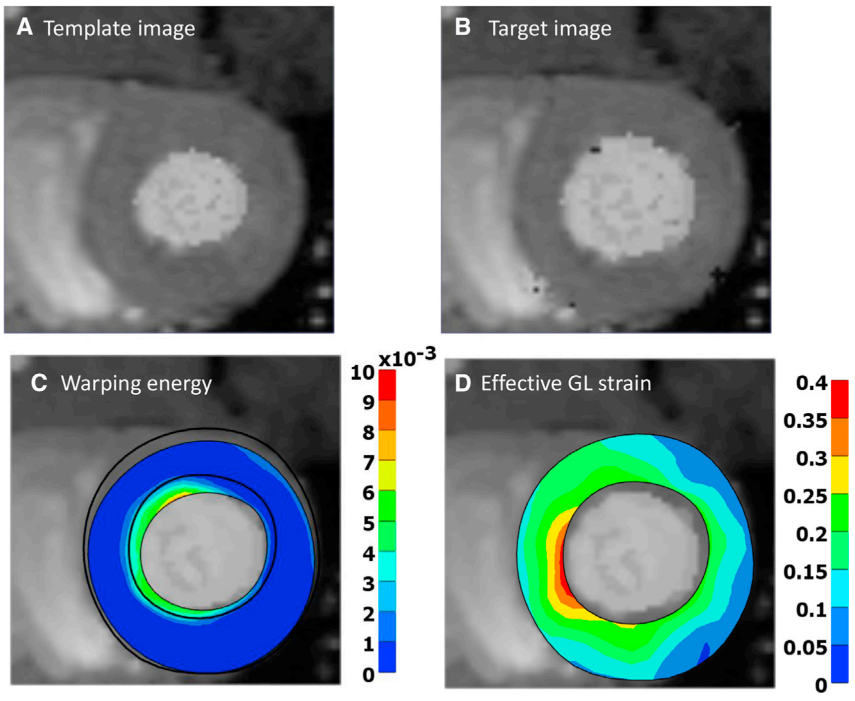


FIGURE 2 Two-dimensional slice from a 3D warping analysis that tracks the deformation of the left ventricle during diastole. (*A*) The template image slice at start of diastole, (*B*) the target image slice at end of diastole, (*C*) and warping energy, measuring the mismatch between the template and the target image. The black contour shows the outline of the target geometry. (*D*) Effective Green Lagrange strain in the final, deformed state. To see this figure in color, go online.

via the plugin framework. It uses the "Angio" library, which simulates the growth, branching, and anastomosis of angiogenic microvessels embedded in a 3D collagen matrix (8,30). During angiogenesis, growing neovessels deform the collagen matrix. In turn, deformation of the matrix directly affects the growth of the vessels through changes in matrix density and collagen fibril alignment. To couple the Angio growth library with FEBio, a task plugin was developed to allow the two codes to interact. The task plugin first seeds the matrix with parent microvessels using the Angio library and executes an initial growth step. The parent vessels are allowed to grow, governed by heuristics that control the growth rate, branching probability, and anastomosis. Then, the discrete model of the microvascular network is passed to FEBio, which solves for the matrix deformation. The mechanical interaction between the vessels and matrix is modeled as a spatially varying contractile stress that is centered at the tip of each microvessel sprout. The new, deformed configuration is then passed back to the Angio library, which calculates the next growth step in the deformed model (7).

In the example application (Fig. 3), an in vitro model of neovessel growth is simulated using the AngioFE plugin. The simulation represents 6 days of growth in vitro. A cylindrical core of 3 mg/mL collagen is seeded with parent microvessels, and this core is surrounded by a field of avascular 3 mg/mL collagen (Fig. 3 *A*). A high-density interface forms between the core and field at the time of polymerization, and this density gradient is represented in the underlying model. The simulation geometry and parameters mimic an experimental model that is used to assess neovessel invasion across tissue interfaces. The high-density gradient that forms between the core and field is sufficient to prevent the angiogenic microvessels from crossing the interface into the field.

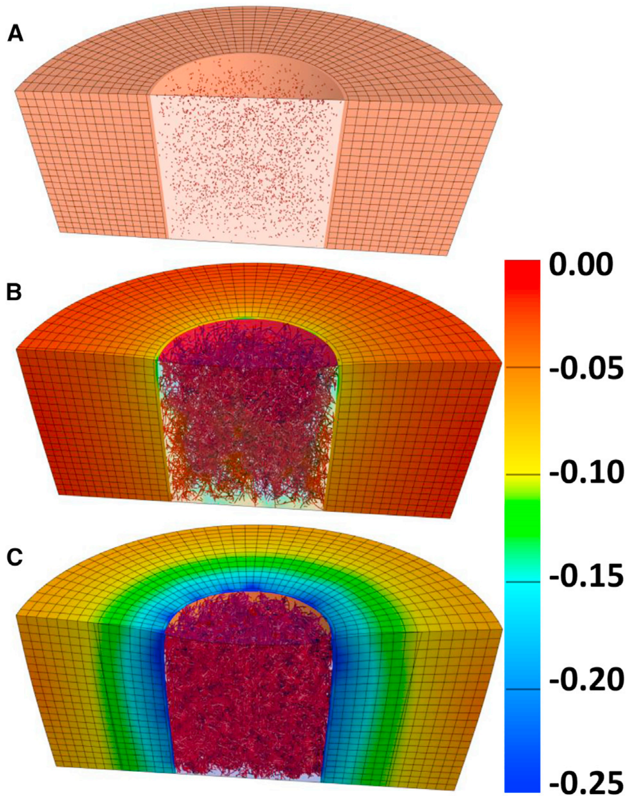


FIGURE 3 Simulation of microvessel growth in an in vitro model of microvessel invasion across a tissue interface. (*A*) The initial geometry, illustrating the cores of collagen seeded with parent microvessels and the surrounding field of avascular collagen. The interface is too thin to be seen in this image. (*B* and *C*) At times X and Y during the simulation, microvessels grow into a dense vascular network, contracting the surrounding matrix as they grow. Vessels are unable to cross the interface because of the large density gradient. Fringe plots show Green-Lagrange deviatoric strain. To see this figure in color, go online.

Solver plugins

FEBio is a multiphysics solver that couples fluid and solid mechanics with growth, reaction-diffusion, and electrokinetic effects. This is embodied in the multiphasic mixture module, which solves the balance of linear momentum, mass balance, and diffusion equations simultaneously using a monolithic approach. At times, users may wish to solve governing equations with the FE method that are not supported within FEBio, or they may wish to focus on a subset of the physics supported by FEBio. For such cases, "solver plugins" can be developed.

As an illustration of this type of plugin, the FEBioChem plugin implements a nonlinear solver for the reaction-diffusion-convection equation. Using this plugin, users can solve for the concentrations of chemical species that diffuse, convect, and undergo chemical reactions in space and time. Although the multiphasic mixture module can address such problems, this plugin offers a simplified context that does not model solid matrix deformation or osmotic effects in the fluid solution, providing a simplified approach to modeling systems of chemical reactions. These simplifications result in a significant speedup compared to the analogous multiphasic model, as the additional displacement and pressure nodal degrees of freedom associated with the mixture framework are eliminated. In the future, we plan to add support for surface diffusion, biomolecular reactions between surface-bound molecules, and modeling reversible reactions with a single equation. The convection feature makes use of a user-defined velocity field.

To illustrate the function and utility of the FEBioChem plugin, we reproduced the results of a computational study by Vempati et al. (31) that examined the proteolysis of vascular endothelial growth factor (VEGF) in the context of matrix metalloprotease secretion, VEGF-extracellular matrix (ECM) binding, VEGF proteolysis from VEGF165 to VEGF114, and VEGF receptor-mediated recapture (Fig. 4). The computational model simulates the stalk cells and tip cells of a capillary sprout, surrounded by a basement

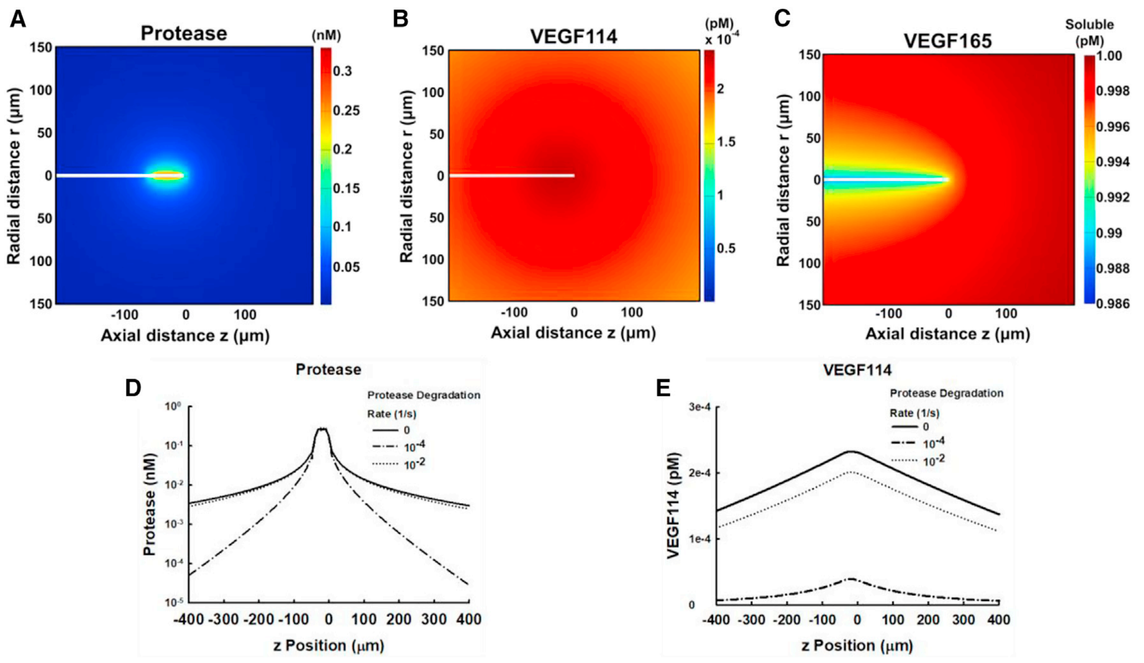


FIGURE 4 Reproduction of results in Vempati et al. (31) using FEBioChem. The panels in this figure correspond to Fig. 2, *A–E* in the publication. (*A*) A line of cells (*white*) extends from the left edge of the computational domain, surrounded by a thin basement membrane and ECM. The fringe plot shows predicted protease distribution produced by the tip (*rightmost*) cell, in equilibrium with flux and diffusion. (*B*) VEGF114 distribution near the tip partially governed by $VEGF_{165} + Protease \rightarrow VEGF_{114} + Protease$. (*C*) The initial 1 pM concentration of VEGF 165 is slightly altered by protease activity. (*D*) Axial protease distributions. Protease degradation reduces the concentration of protease that is available to modify VEGF_{165}. (*E*) Axial VEGF_{114} distribution. Protease degradation reduces conversion of VEGF_{165} to VEGF_{114}. To see this figure in color, go online.

membrane within an ECM volume. Predicted concentrations of protease and VEGF were compared to those in the publications and most species agreed to within 1% (all species agreed within ~5%; see Supporting Materials and Methods for further details). Furthermore, the spatial distributions showed excellent correspondence with the figures in the corresponding publication. FEBioChem's simplified context can represent complex reaction networks, diffusion, and species production and degradation in porous media and materials.

DISCUSSION

The plugin framework is a powerful new capability that allows users to extend and customize FEBio to their specific needs. A large variety of plugin types are supported, including material, task, solver, body loads, boundary loads, initial conditions, nonlinear constraints, plot and log data, callback, and loadcurve plugins. Some of these plugins were illustrated in Results above, including a plugin to create a new constitutive model, a plugin that implements an image-based constraint, a plugin to link FEBio to an external library that simulates angiogenesis, and a plugin that adds reaction-diffusion modeling capabilities to FEBio. A description of the other available types of plugins can be found in the Supporting Materials and Methods. New types of plugins can be added easily as the need arises. The plugin approach in FEBio has been applauded by many of our users as flexible and easy to use. For instance, a recent publication implemented a new constitutive model in both FEBio and the commercial software ABAQUS (now called Simulia; Dassault Systèmes), and concluded that the plugin framework in FEBio made it far easier to implement the new constitutive model than the approach required for ABAQUS (32).

It is worth noting that the additional specialized xml-formatted input files required by some plugins need to be created in a text editor. PreView, our FE preprocessor that is used for setting up FEBio models, does not support the creation of these additional input files, although it does support the addition of user materials and the FEBioChem plugin capabilities (these plugins add features directly through FEBio's ".feb" input file). This arrangement is intentional, as new plugins will have specific requirements for input that we cannot anticipate. For all the plugins that are developed by our lab and those that we distribute on behalf of outside developers, we provide detailed documentation of any additional input file formats that are needed by the plugin, and this information will always be up to date at https://www.febio.org.

To the best of our knowledge, there is no other freely available FE package that is designed specifically for simulation in biomechanics while offering such a wide variety of simulation capabilities. Nevertheless, it is informative to place FEBio in the context of other open-source software packages that serve the biomechanics community. OpenSim (http://opensim.stanford.edu/) is a modeling framework for simulation of human movement using forward and inverse rigid body dynamics. Muscles and tendons are modeled as discrete elements connected to rigid bodies. It is highly specialized for these applications and does not include any FE-based algorithms. FEBio has some rigid-body modeling capabilities and allows rigid-body dynamics to be coupled with FE analysis. Like FEBio, Artisynth (https://www.artisynth.org/Main/HomePage) supports the combined simulation of multibody and FE analysis. In this regard, it offers modeling capabilities that are similar to FEBio, but it focuses only on solid mechanics. SimVascular is focused on patient-specific blood flow analysis and includes capabilities for fluid mechanics and fluid-structure interaction. These capabilities overlap with FEBio, but SimVascular is limited to these types of physical simulations, caters to a much more specific group of users, and includes features that are specific to constructing models of blood flow in arteries that are not part of FEBio's capabilities. Similarly, Continuity (https://continuity.ucsd.edu/) is an FE software that supports simulation of cardiac biomechanics, transport, and electrophysiology. It is tightly tailored to this application domain, and although some of the capabilities of Continuity exist in FEBio, it is once again focused on a highly specific group of users. In all of these cases, FEBio offers some functionality that overlaps with other software packages. This is not surprising, given that FEBio was developed to be a general-purpose FE software that targets a range of different types of simulations in biomechanics using mixture theory. That said, it is not our goal to replace the impressive and feature-rich software packages mentioned above. In fact, much of our current focus with application of the plugin framework is the integration of FEBio with other software packages so that users can take advantage of each software's strengths. Through this approach, we hope to enable simulations that couple different modeling approaches and/or span different physical scales.

The FEBio software suite and the plugins discussed in this manuscript are available for download at https://www.febio.org.

SUPPORTING MATERIAL

Supporting Materials and Methods, five figures, three tables, and one data file are available at http://www.biophysj.org/biophysj/supplemental/S0006-3495(18)31069-5.

AUTHOR CONTRIBUTIONS

S.A.M. and J.A.W. conceived the project. S.A.M. wrote the plugin interface software. All authors participated in data analysis and interpretation. S.A.M., S.A.L., and J.A.W. drafted the manuscript. All authors edited the manuscript and gave final approval for publication.

ACKNOWLEDGMENTS

Financial support from National Institutes of Health grant #R01EB015133 (S.A.M., G.A.A., and J.A.W.), National Institutes of Health grant #R01HL131856 (S.A.M. and J.A.W.), National Institutes of Health grant #R01AR069297 (S.A.M. and J.A.W.), and National Institutes of Health grant #R01GM104139 is gratefully acknowledged.

REFERENCES

1. Maas, S. A., B. J. Ellis, ..., J. A. Weiss. 2012. FEBio: finite elements for biomechanics. *J. Biomech. Eng.* 134:011005.
2. Maas, S. A., G. A. Ateshian, and J. A. Weiss. 2017. FEBio: history and advances. *Annu. Rev. Biomed. Eng.* 19:279–299.
3. Wang, H., A. S. Abhilash, ..., V. B. Shenoy. 2014. Long-range force transmission in fibrous matrices enabled by tension-driven alignment of fibers. *Biophys. J.* 107:2592–2603.
4. Rabbitt, R. D., J. A. Weiss, ..., M. I. Miller. 1995. Mapping of hyperelastic deformable templates. *Proc. SPIE Int. Soc. Opt. Eng.* 2552:252–264.
5. Edgar, L. T., J. B. Hoying, ..., J. A. Weiss. 2014. Mechanical interaction of angiogenic microvessels with the extracellular matrix. *J. Biomech. Eng.* 136:021001.
6. Edgar, L. T., J. B. Hoying, and J. A. Weiss. 2015. In silico investigation of angiogenesis with growth and stress generation coupled to local extracellular matrix density. *Ann. Biomed. Eng.* 43:1531–1542.
7. Edgar, L. T., S. A. Maas, ..., J. A. Weiss. 2015. A coupled model of neovessel growth and matrix mechanics describes and predicts angiogenesis in vitro. *Biomech. Model. Mechanobiol.* 14:767–782.
8. Edgar, L. T., C. J. Underwood, ..., J. A. Weiss. 2014. Extracellular matrix density regulates the rate of neovessel growth and branching in sprouting angiogenesis. *PLoS One.* 9:e85178.
9. Todd, J. N., T. G. Maak, ..., J. A. Weiss. 2018. Hip chondrolabral mechanics during activities of daily living: role of the labrum and interstitial fluid pressurization. *J. Biomech.* 69:113–120.
10. Ateshian, G. A., J. J. Shim, ..., J. A. Weiss. 2018. Finite element framework for computational fluid dynamics in FEBio. *J. Biomech. Eng.* 140:021001–021001-17.
11. Ateshian, G. A., S. Maas, and J. A. Weiss. 2012. Solute transport across a contact interface in deformable porous media. *J. Biomech.* 45:1023–1027.
12. Ateshian, G. A., S. Maas, and J. A. Weiss. 2013. Multiphasic finite element framework for modeling hydrated mixtures with multiple neutral and charged solutes. *J. Biomech. Eng.* 135:111001.
13. Ateshian, G. A., R. J. Nims, ..., J. A. Weiss. 2014. Computational modeling of chemical reactions and interstitial growth and remodeling involving charged solutes and solid-bound molecules. *Biomech. Model. Mechanobiol.* 13:1105–1120.
14. Ateshian, G. A., S. Maas, and J. A. Weiss. 2010. Finite element algorithm for frictionless contact of porous permeable media under finite deformation and sliding. *J. Biomech. Eng.* 132:061006.
15. Ateshian, G. A., C. R. Henak, and J. A. Weiss. 2015. Toward patient-specific articular contact mechanics. *J. Biomech.* 48:779–786.
16. Henak, C. R., A. E. Anderson, and J. A. Weiss. 2013. Subject-specific analysis of joint contact mechanics: application to the study of osteoarthritis and surgical planning. *J. Biomech. Eng.* 135:021003.
17. Maas, S. A., B. J. Ellis, ..., J. A. Weiss. 2016. Finite element simulation of articular contact mechanics with quadratic tetrahedral elements. *J. Biomech.* 49:659–667.
18. Maas, S. A., A. Erdemir, ..., J. A. Weiss. 2016. A general framework for application of prestrain to computational models of biological materials. *J. Mech. Behav. Biomed. Mater.* 61:499–510.
19. Matthies, H., and G. Strang. 1979. The solution of nonlinear finite element equations. *Int. J. Numer. Methods Eng.* 14:1613–1626.

20. Ateshian, G. A. 2015. Viscoelasticity using reactive constrained solid mixtures. *J. Biomech.* 48:941–947.

21. Laursen, T. A., and B. N. Maker. 1995. Augmented Lagrangian quasi-Newton solver for constrained nonlinear finite element applications. *Int. J. Numer. Methods Eng.* 38:3571–3590.

22. Phatak, N. S., S. A. Maas, …, J. A. Weiss. 2009. Strain measurement in the left ventricle during systole with deformable image registration. *Med. Image Anal.* 13:354–361.

23. Phatak, N. S., Q. Sun, …, J. A. Weiss. 2007. Noninvasive determination of ligament strain with deformable image registration. *Ann. Biomed. Eng.* 35:1175–1187.

24. Veress, A. I., G. T. Gullberg, and J. A. Weiss. 2005. Measurement of strain in the left ventricle during diastole with cine-MRI and deformable image registration. *J. Biomech. Eng.* 127:1195–1207.

25. Veress, A. I., G. Klein, and G. T. Gullberg. 2013. A comparison of hyperelastic warping of PET images with tagged MRI for the analysis of cardiac deformation. *Int. J. Biomed. Imaging.* 2013:728624.

26. Veress, A. I., J. A. Weiss, …, R. D. Rabbitt. 2002. Strain measurement in coronary arteries using intravascular ultrasound and deformable images. *J. Biomech. Eng.* 124:734–741.

27. Weiss, J. A., A. I. Veress, …, R. D. Rabbitt. 2006. Strain measurement using deformable image registration. *In* Mechanics of Biological Tissue. G. A. Holzapfel and R. W. Ogden, eds. Springer, pp. 489–501.

28. Veress, A. I., N. Phatak, and J. A. Weiss. 2005. Deformable image registration with hyperelastic warping. *In* Handbook of Biomedical Image Analysis: Vol. 3, Registration Models (Part A). J. S. Suri, D. L. Wilson, and S. Laxminarayanan, eds. Kluwer Academic/Plenum Publishers, pp. 487–534.

29. Veress, A. I., J. A. Weiss, …, G. T. Gullberg. 2008. Measuring regional changes in the diastolic deformation of the left ventricle of SHR rats using microPET technology and hyperelastic warping. *Ann. Biomed. Eng.* 36:1104–1117.

30. Edgar, L. T., S. C. Sibole, …, J. A. Weiss. 2013. A computational model of in vitro angiogenesis based on extracellular matrix fibre orientation. *Comput. Methods Biomech. Biomed. Engin.* 16:790–801.

31. Vempati, P., F. Mac Gabhann, and A. S. Popel. 2010. Quantifying the proteolytic release of extracellular matrix-sequestered VEGF with a computational model. *PLoS One.* 5:e11860.

32. Pierrat, B., J. G. Murphy, …, M. D. Gilchrist. 2016. Finite element implementation of a new model of slight compressibility for transversely isotropic materials. *Comput. Methods Biomech. Biomed. Engin.* 19:745–758.

Biophysical Journal, Volume *115*

Supplemental Information

A Plugin Framework for Extending the Simulation Capabilities of FEBio

Steve A. Maas, Steven A. LaBelle, Gerard A. Ateshian, and Jeffrey A. Weiss

SUPPORTING MATERIAL

Details of the Plugin Framework in FEBio.

FEBio is structured as a hierarchy of modules or libraries, each module collecting algorithms and data structures for addressing a specific aspect of a FE analysis (Figure S1). Some modules address different types of physics, such as structural mechanics (FEBioMech), mechanics of mixtures (FEBioMix), and fluid mechanics (FEBioFluid). Other modules deal with file input (FEBioXML) and output (FEBioPlot). A separate library deals with solving the linear system equations (NumCore). The FEBioLib library is the portal that users can use to interact with all the FEBio features. The FEBio executable is a command-line front-end to the FEBioLib library.

For the most part, each module works independently of others, but all modules interact with a special module, termed the FEBio kernel (FECore). One of the most important

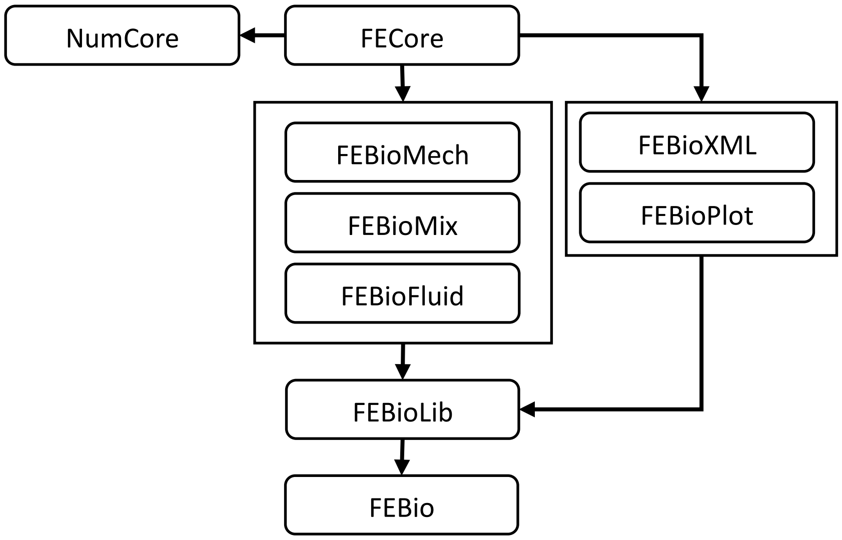


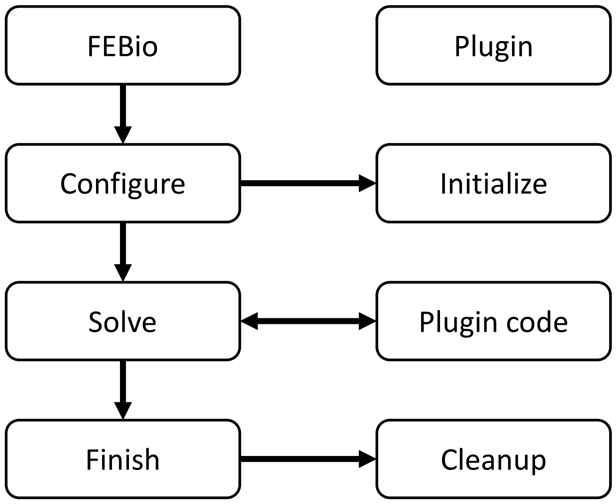
Figure S1. Overview of the modular structure of FEBio. The FECore library is the kernel that centralizes all the features and that all other modules depend on.

responsibilities of this kernel module is to keep track of all the features that are implemented by the separate modules. Each physics module informs the kernel of the materials, boundary conditions, solvers, etc., that it implements. This information can then be used, for instance, by the input module for parsing the FEBio input file. This approach makes it easy to add new features, as all new features remain centralized within their respective module. For instance, addition of constitutive models to the physics modules do not require any changes to the input or output modules.

The plugin framework is part of the FECore kernel library[*], which contains most of the base classes from which new features can be derived. It also contains the essential algorithms and data structures for defining and solving FE problems, and thus all the relevant code for creating FEBio plugins. Most of the other modules in FEBio use the FECore library to implement new physics-based solvers. For instance, the FEBioMech library implements solutions algorithms for solving quasi-static or dynamic structural mechanics problems. Similarly, a plugin will use the FECore library to implement the new functionality. It may also require some of the other modules if the plugin extends functionality of a particular module. For instance, a plugin that implements a new elastic constitutive model will also depend on the FEBioMech library.

Plugins interact with FEBio as follows (Figure S2). The path to a plugin file is specified in the FEBio configuration file. When FEBio starts, it parses this file and attempts to load each plugin listed in it. When a plugin is loaded, it registers its new functionality with FEBio. This registration process is important as it allows the new features to be recognized automatically in the FEBio input file or output file. The plugin allocates any resources it may need during this phase. During the solution phase, FEBio will call the plugin whenever it needs data. The timing

[*] Documentation on the plugin framework can be found at http://febiodoc.sci.utah.edu/doxygen/.



<u>Figure S2</u>. Schematic of how FEBio interacts with a plugin. During initialization, plugins are loaded. In the solution phase the plugin code whenever FEBio needs data from it. Finally, plugins are given a chance to cleanup any allocated resources before FEBio ends.

of calls to the plugin from FEBio greatly depends on the nature of the plugin. For instance, if the plugin implements an elastic constitutive model, FEBio calls the plugin when it needs to calculate the stress or the elasticity tensor. Finally, before FEBio terminates, the plugin is given an opportunity to cleanup and deallocate its resources.

FEBio has an expansive collection of tensor classes that greatly facilitate the implementation of complex tensor expressions. It offers various classes for representing first (i.e. vectors) second, third, fourth, fifth, and sixth order tensors and common operations that can be made with tensors (e.g. addition, multiplication, contraction, etc.). To maximize efficiency, different tensor symmetries are implemented in different classes. For example, the *mat3ds* class implements a second-order symmetric tensor, *mat3da* implements a skew-symmetric tensor, and *mat3dd* implements a diagonal second-order tensor. These tensor classes make the implementation of new material plugins much easier for the user (Figure S3).

```
mat3ds FENeoHookean::Stress(FEMaterialPoint& mp)
{
   FEElasticMaterialPoint& pt =
                    *mp.ExtractData<FEElasticMaterialPoint>();
   double detF = pt.m_J;
   double detFi = 1.0/detF;
   double lndetF = log(detF);
   // calculate left Cauchy-Green tensor
   mat3ds b = pt.LeftCauchyGreen();
   // lame parameters
   double lam = m_v*m_E/((1+m_v)*(1-2*m_v));
   double mu  = 0.5*m_E/(1+m_v);
   // Identity
   mat3dd I(1);
   // calculate stress
   mat3ds s = (b - I)*(mu*detFi) + I*(lam*lndetF*detFi);
   return s;
}
```

Figure S3. Stress evaluation for the neo-Hookean material. This figure illustrates the use of classes from FEBio's tensor class library (mat3ds for symmetric second-order tensors, mat3dd for diagonal second order tensors) and defines tensor operations. This library greatly simplifies the implementation of complicated tensor expressions.

Material Plugin for Wang et al. Reproduction

Strain Energy Function. The strain energy density function for the constitutive model in (2) is given by

$$W = W_b + W_f \,. \tag{S1}$$

Here, W_b captures the isotropic response:

$$W_b = \frac{\mu}{2}\left(\tilde{I}_1 - 3\right) + \frac{\kappa}{2}\left(J-1\right)^2 , \tag{S2}$$

where μ is the shear modulus, κ is the bulk modulus, $\tilde{I}_1$ is the first invariant of the deviatoric right deformation tensor $\tilde{\mathbf{C}} = J^{-2/3}\mathbf{C} = J^{-2/3}\mathbf{F}^T\mathbf{F}$, $\mathbf{F}$ is the deformation gradient, and J = det($\mathbf{F}$) is the volume ratio. W_f is the contribution from the aligned fibers:

$$W_f = \sum_{a=1}^{3} f\left(\lambda_a\right) \ , \tag{S3}$$

Where the λ_a are the principal stretch ratios.

Stress Tensor. The 2nd PK stress is given by

$$\mathbf{S} = 2\frac{\partial W}{\partial \mathbf{C}} = \mathbf{S}_b + \mathbf{S}_f \, , \tag{S4}$$

$$\mathbf{S}_b = \mu J^{-2/3}\left(\mathbf{I} - I_1 \frac{1}{3}\mathbf{C}^{-1}\right) + \kappa\left(J-1\right) J\mathbf{C}^{-1}, \tag{S5}$$

where μ is the shear modulus and κ is the bulk modulus. The contribution from the fibers is given by:

$$\mathbf{S}_f = 2\sum_{a=1}^{3} \frac{d^2 f}{d\lambda_a^2} \mathbf{N}_a \otimes \mathbf{N}_a \ , \tag{S6}$$

where $\mathbf{N}_a$ is a unit vector defining the direction associated with the a_{th} principal stretch in the reference configuration. The Cauchy stress follows from the push-forward of $\mathbf{S}$:

$$\boldsymbol{\sigma} = \frac{1}{J}\mathbf{F}\mathbf{S}\mathbf{F}^T = \frac{1}{J}\mathbf{F}\left(\mathbf{S}_b + \mathbf{S}_f\right)\mathbf{F}^T = \boldsymbol{\sigma}_b + \boldsymbol{\sigma}_f \ . \tag{S7}$$

$$\boldsymbol{\sigma}_b = \frac{\mu}{J}\operatorname{dev}\left(\tilde{\mathbf{b}}\right) + \kappa\left(J-1\right)\mathbf{1} \ , \tag{S8}$$

where the "dev" operator extracts the deviatoric part of a 2nd order tensor in the current configuration. The fiber contribution to the Cauchy stress is given by

$$\boldsymbol{\sigma}_f = \sum_{a=1}^{3} \frac{\lambda_a}{J}\frac{df}{d\lambda_a} \mathbf{n}_a \otimes \mathbf{n}_a = \sum_{a=1}^{3} \sigma_a \mathbf{n}_a \otimes \mathbf{n}_a \ , \tag{S9}$$

where $\mathbf{n}_a$ is a unit vector defining the direction associated with the a_{th} principal stretch in the current configuration.

The authors proposed the following model to represent the strain-dependent anisotropic contribution from the fibers:

$$\frac{df}{d\lambda_a} = \begin{cases} 0 & \lambda_a < \lambda_1 \\ \dfrac{E_f \left(\dfrac{\lambda_a - \lambda_1}{\lambda_2 - \lambda_1} \right)^n (\lambda_a - \lambda_1)}{n+1}, & \lambda_1 \le \lambda_a \le \lambda_2 \\ E_f \left[\dfrac{\lambda_2 - \lambda_1}{n+1} + \dfrac{(1 + \lambda_a - \lambda_2)^{m+1} - 1}{m+1} \right], & \lambda_a \ge \lambda_2 \end{cases} \tag{S10}$$

where $\lambda_1 = \lambda_c - \lambda_t / 2$ and $\lambda_2 = \lambda_c + \lambda_t / 2$ define the applicable ranges of the piecewise function. These parameters are chosen so that the principal stress contributions from each principal stretch λ_a vanish below a critical (tensile) principal stretch λ_c, and show a stiffening response characterized by a fiber modulus E_f and a hardening exponent m. There is a transition region when the principal stretch is between λ_1 and λ_2, and the authors in the original publication chose the transition exponent as n = 5. The transition width λ_t is a user defined parameter, chosen by the authors in the original publication to be some small fraction of λ_c.

Elasticity Tensor. The material version of the 4th order elasticity tensor is given by:

$$C_{IJKL} = 2\frac{dS_{IJ}}{dC_{KL}} = C^b_{IJKL} + C^f_{IJKL} \ , \tag{S11}$$

The isotropic term is:

$$C^b_{IJKL} = \frac{2\mu}{3} J^{-2/3} \left[-C^{-1}_{KL} \left(\delta_{IJ} - \frac{1}{3} I_1 C^{-1}_{IJ} \right) - \left(\delta_{KL} C^{-1}_{IJ} - I_1 I^C_{IJKL} \right) \right]. \\ + \kappa \left[\left(2J-1\right) J C^{-1}_{KL} C^{-1}_{IJ} - 2\left(J-1\right) J I^C_{IJKL} \right] \tag{S12}$$

The spatial version of the isotropic part of the elasticity tensor then follows:

$$c^b_{ijkl} = \frac{\mu}{J} \left[\frac{2}{3} \tilde{I}_1 I_{ijkl} - \frac{2}{3} \left(\delta_{kl} \tilde{b}_{ij} + \tilde{b}_{kl} \delta_{ij} \right) + \frac{2}{9} \tilde{I}_1 \delta_{ij} \delta_{kl} \right]. \\ + \kappa \left[\left(2J-1\right) \delta_{ij} \delta_{kl} - 2\left(J-1\right) I_{ijkl} \right] \tag{S13}$$

The fiber contribution is easier to deduce in direct notation in the spatial configuration. The fiber contribution to the spatial elasticity tensor is:

$$\boldsymbol{c}^f = \sum_a^3 \frac{1}{J} \left[\frac{\partial}{\partial \lambda_a} \left(\frac{\partial f}{\partial \lambda_a} \frac{1}{\lambda_a} \right) \right] \lambda_a^3 \mathbf{n}_a \otimes \mathbf{n}_a \otimes \mathbf{n}_a \otimes \mathbf{n}_a \\ + \sum_{\substack{a,b=1 \\ a \neq b}}^3 \frac{\sigma_a \lambda_b^2 - \sigma_b \lambda_a^2}{\lambda_a^2 - \lambda_b^2} \left(\mathbf{n}_a \otimes \mathbf{n}_b \otimes \mathbf{n}_a \otimes \mathbf{n}_b + \mathbf{n}_a \otimes \mathbf{n}_b \otimes \mathbf{n}_b \otimes \mathbf{n}_a \right) . \tag{S14}$$

The factor in square brackets can be expanded to

$$\frac{\partial}{\partial \lambda_a} \left(\frac{\partial f}{\partial \lambda_a} \frac{1}{\lambda_a} \right) \lambda_a^3 = \lambda_a \left(\lambda_a \frac{\partial^2 f}{\partial \lambda_a^2} - \frac{\partial f}{\partial \lambda_a} \right). \tag{S15}$$

When $\lambda_b \to \lambda_a$,

$$\lim_{\lambda_b \to \lambda_a} \frac{\sigma_a \lambda_b^2 - \sigma_b \lambda_a^2}{\lambda_a^2 - \lambda_b^2} = \frac{\lambda_a^2}{2J} \left(\frac{d^2 f}{d \lambda_a^2} \right) - \sigma_a . \tag{S16}$$

Finally,

$$\frac{\partial^2 f}{\partial \lambda_a^2} = \begin{cases} 0 & \lambda_a < \lambda_1 \\ E_f \left(\dfrac{\lambda_a - \lambda_1}{\lambda_2 - \lambda_1} \right)^n, & \lambda_1 \le \lambda_a \le \lambda_2 \\ E_f \left(1 + \lambda_a - \lambda_2\right)^m, & \lambda_a \ge \lambda_2 \end{cases} \tag{S17}$$

The example problem analyzed in Figure 1 differs slightly from the example in the original publication of Wang et al (2). We used a cubic geometry for the extracellular matrix while the publication used a cylindrical geometry. As expected, this does not affect the conclusions regarding comparisons of the model to the neo-Hookean model, as our results are very similar to those obtained for the cylindrical geometry in the original publication (Figure 2 in (2)). The material coefficients used in the neo-Hookean constitutive model to generate the results in Figures 1A and 1B were E = 2.0 KPa, and ν = 0.3, while the material coefficients used in the fiber-stiffening model for results in Figures 1A and 1C were μ = 0.7692 KPa, κ = 1.667 KPa, E_f = 134.6 KPa, λ_c = 1.02, λ_t = 0.255, n = 5, m = 30.

To illustrate that the plugin accurately reproduces the stress-strain behavior of the constitutive model, we simulated uniaxial extension and compared the stress-strain curve to the published result (Figure S3). This graph exactly reproduces the result in Figure 1c of the original publication. The material coefficients used to generate the results in Figure S3 were μ = 0.7692 KPa, κ = 1.667 KPa, E_f = 22.88 KPa, λ_c = 1.1, λ_t = 0.0255, n = 5, m = 10, as specified in the original publication.

Vempati et al. Reproduction

We reproduced the results of the finite volume model reported in Vempati et al. (1) to study autocrine endothelial signaling in angiogenesis using the FEBioChem plugin. The central objective of the study was to determine quantitatively if a single protease secreting cell at the front of an angiogenic microvessel could modify local concentrations of the cytokine vascular endothelial growth factor (VEGF).

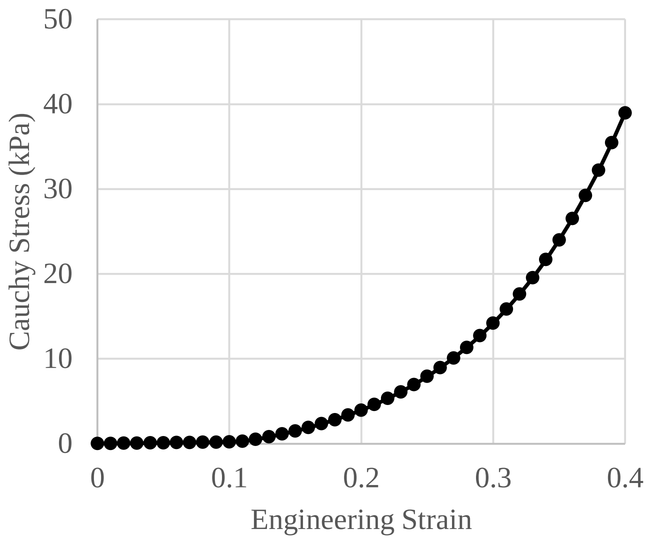


Figure S4. The stress-strain curve of a uniaxial tensile test for the constitutive model proposed by Wang et al (1), obtained from a material plugin developed in FEBio. This graph recovers the results in Figure 1c from the corresponding publication.

Geometry. The idealized geometry of the model comprised a cylindrical line of cells surrounded by an inner tube to represent the basement membrane and an outer tube to represent the extracellular matrix. The model in the original manuscript was solved using the finite volume method with axisymmetric geometry. Since FEBio is inherently a 3D analysis code, axisymmetry was emulated via a judicious choice of geometry and boundary conditions. This was accomplished by representing the cylindrical domain with a 9 degree wedge. All geometric measures were obtained directly from the text of the original manuscript. Meters were used as the length unit in our reproduction rather than micrometers, so input parameters were adjusted accordingly from the values provided in the publication, and output concentrations should be interpreted as kM.

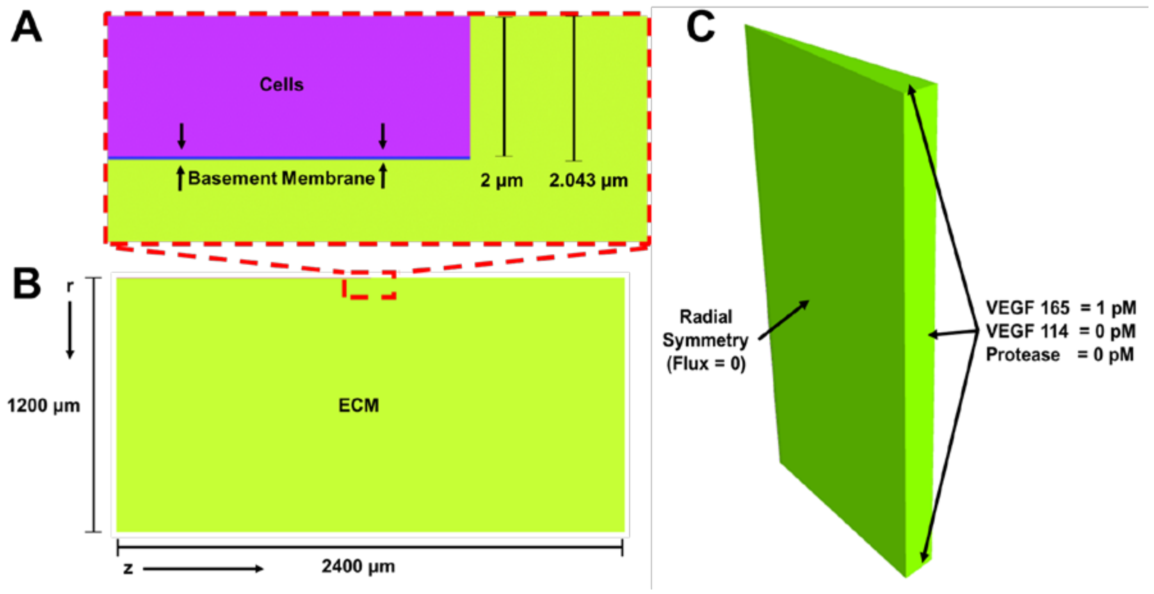


Figure S5. Geometry for Vempati et al. reproduction. **A)** A line of cells (purple, 2 μm radius) extends 1200 μm from the left face along the central *z*-axis. The cells are covered by a 0.043 μm thick basement membrane (blue). The right-most 40 μm of the basement membrane constitutes the tip cell, where flux of protease is specified. **B)** A cross-section through the *r*-*z* plane of the simulation domain. Most of the domain consists of extracellular matrix (green). **C)** Rotated geometry highlighting initial and boundary conditions applied to the wedge. Radial symmetry was maintained by prescribing solute flux as zero on faces normal to the radial direction. Far-field effects are controlled by prescribing concentrations on faces where r = 1200 μm or z = 0 μm or z = 2400 μm.

Finite Element Mesh. The cell volume and extracellular matrix elements above the cell volume were represented as a column of 300 8 μm tall (z-direction) 6-node linear pentahedral wedge elements (FEBio penta6). The basement membrane was constructed from a column of 300 8 μm tall 8-node trilinear hexahedral elements (FEBio hex8). The remainder of the radial volume was spanned by 598 single element wide (θ-direction) columns of ~2 μm deep (r-direction) hex8 elements yielding a total of 600 elements total in the radial direction. (matching the finite element discretization used in the original publication). The depth of the cell elements was 2 μm, the basement membrane was 0.0143 μm thick, and the span of all extracellular membrane elements was 2 μm. In comparison, Vempati et al. constrained their mesh side lengths between 4 and 8 μm through their domain. Further, while we represented the basement membrane using 3 thin elements in the radial direction, Vempati et al. used a single line of nodes to represent the basement membrane.

Materials. Reactions within FEBioChem are specified at the material level, so separate materials were created for the cells, basement membrane, and extracellular matrix. FEBioChem allows the user to model the reactions by specifying the stoichiometric coefficients of the reactants and products for each solute or solid-bound molecule participating in a given reaction. The solid volume fraction of each material and diffusivity for each solute were defined at the material level using parameters from Vempati et al. (note that the solid volume fraction was calculated from the available volume fraction as $\varphi_r^\alpha = 1 - K_{av}$ where φ_r^α is the volume fraction of a species α in the reference configuration and K_{av} is the average volume fraction available for reaction and diffusion).

Initial Conditions. Vempati et al. assumed a uniform concentration of soluble VEGF_{165} throughout the simulation volume, excluding the cell volume which does not participate in reactions. Solid-bound concentrations were converted to initial apparent densities using the equation $\rho_r^\alpha = c_r^\alpha M^\alpha(1-\varphi_r^\alpha)$, and these concentrations were then prescribed at the material level (ρ_r^α and c_r^α are the density and concentration of a species α in the reference configuration and M^α is the molar mass for the chosen species).

Boundary Conditions. The steady state concentration of membrane bound VEGF_{165} ($V_{165}H$) was determined from the dissociation constant-defined equation $[V_{165}H] = \frac{[H_{Total}][V_{165}]}{K_d+V_{165}}$ and used to prescribe elemental concentrations for the basement membrane and extracellular matrix to improve computation time. Dirichlet boundary conditions were used such that $[V_{165}] = 1\text{ pM}, [V_{114}] = 0, [P] = 0\ \ \forall\, r = 1200\ \mu\text{m} \cup z = \pm 1200\ \mu\text{m}$. VEGF_{165} flux across the faces defined by $r = 1200\ \mu\text{m}$ and $z = \pm 1200\ \mu\text{m}$ was fixed. Symmetry across the cut faces was maintained by fixing the flux of each solute on the wedge faces. Protease secretion from the tip cell surface was prescribed at a constant value of $2.7\cdot10^{-12}$ mol/(m^2·s). In the original publication, VEGF Receptor 2 (R2) was specified as an expression rate in units of concentration per second, w we represented this with the reaction $[\ ] \rightarrow R_2$ within the tip and stalk membrane materials rather than a solute flux applied to a surface. Similarly, degradation and cellular internalization for a given species C was specified by the reaction $C \rightarrow [\ ]$. To speed up computation, we initially superimposed 1 pM of VEGF_{165} to each node in the extracellular matrix since the far-field VEGF_{165} distribution is mostly unaffected by protease. Thus, we performed our simulation in a single step whereas the original publication used one step to determine the VEGF_{165} distribution and another step to introduce protease.

Boundary Flux. Vempati et al enforced flux continuity at the boundary between the ECM and BM to account for diffusive hindrance using a lumped boundary condition. The finite element method is readily able to handle transport between thin membranes and large element volumes. Further, we modeled the basement membrane as a 3-element deep column rather than a single node, removing the need for this consideration from our model.

Molecular Species. The molecular species that were included in the analysis are specified in the Table S1. A density of 1,400 kg/m^3 was used for all proteins (3). The density and individual molar masses were used to specify concentrations of solid bound species. Diffusion terms for soluble species were consistent with Vempati et al. for each material (ECM, BM).

Species	**Code Name**	**Code ID**	**Molar mass (kDa)** (4)
VEGF 165	V165	Sol 1	38.2 (5)
Protease	P	Sol 2	83 (1)
VEGF 114	V114	Sol 3	28.4 (6)
Heparan Sulfate Proteoglycan (HSPG)	H	Sbs 1	110 (1)
Matrix bound VEGF 165	V165H	Sbs 2	148.2
VEGF Receptor 2	R2	Sbs 3	151.5 (7)
Receptor bound VEGF 165	V165R2	Sbs 4	189.7
Receptor bound VEGF 114	V114R2	Sbs 5	179.9

Table S1. Solutes (Sol) and solid-bound species (Sbs) used in the reproduction.

Reactions. Reactions are specified at the material level in FEBioChem. The full list of reactions used in the simulations is detailed in Table S2.

Reaction Name	**Code Name**	**Chemical Reaction**	**Reaction Rate**
VEGF 165 Membrane binding	V165 MB	V165 + H →V165H	420 kM^{-1}·s^{-1}
VEGF 165 Membrane release	V165 MR	V165H → V165 + H	0.01 s^{-1}
VEGF 165 Proteolysis	V165 P	V165 + P → V114 + P	0.631 kM^{-1}·s^{-1}
Membrane bound VEGF 165 proteolysis	V165H P	V165H + P → V114 + H + P	0.631 kM^{-1}·s^{-1}
VEGF Receptor 2 expression	R2 exp	[ ] → R2	1.04792·10^{-6} kM·s^{-1}
VEGF Receptor 2 internalization	R2 int	R2 → [ ]	2.8·10^{-4} s^{-1}

VEGF 165 Receptor binding	V165 RB	V165 + R2 → V165R2	$1 \cdot 10^{4}$ $kM^{-1} \cdot s^{-1}$
VEGF 165 Receptor release	V165 RR	V165R2 → V165 + R2	0.01 s^{-1}
VEGF 165 Internalization	V165R2 int	V165R2 → [ ]	$2.8 \cdot 10^{-4}$ s^{-1}
VEGF 114 Receptor binding	V114 RB	V114 + R2 → V114R2	$1 \cdot 10^{4}$ $kM^{-1} \cdot s^{-1}$
VEGF 114 Receptor release	V114 RR	V114R2 → V114 + R2	0.01 s^{-1}
VEGF 114 Internalization	V114R2 int	V114R2 → [ ]	$2.8 \cdot 10^{-4}$ s^{-1}

Table S2. Reaction descriptions and relevant parameters.

Simulation Conditions

FEBioChem uses a trapezoidal rule by default for time integration, whereas Vempati et al. used a fully-implicit scheme for 1st order derivatives and a central difference approximation with successive over-relaxation (SOR) update for spatial derivatives. FEBio's automatic timestepper was used with a maximum timestep of 10 seconds. The simulation was run until steady state was achieved at ~7 hours simulation time (Vempati et al. achieved steady state in ~10 simulation hours). The FEBioChem convergence norm remained below $1x10^{-13}$ for soluble species and below $1x10^{-17}$ for all solid bound species (for comparison, Vempati et al enforced convergence by allowing $1x10^{-7}$ fractional change in species concentrations for any time-step).

Reproduction assessment

Concentrations of protease and VEGF were measured in various locations and compared to the corresponding publication. Most species were within 1% agreement. Further, the general protease and VEGF 114 distributions in the axial direction (Figure 4 D-E) align with the distributions in the corresponding publication (Figures 2 D-E).

Metric	**Location**	**FEBioChem Calculation**	**Vempati Calculation**	**Difference FEBioChem (%)**
[P]	Inner surface of tip cell membrane	$3.02 \cdot 10^{-1}$ nM	$3.20 \cdot 10^{-1}$ nM	-5.60
$[V_{114}]$	Inner surface of tip cell membrane	$2.33 \cdot 10^{-4}$ pM	$2.40 \cdot 10^{-4}$ pM	-3.08
$[V_{165}]_{min}$	Basement membrane	$9.88 \cdot 10^{-1}$ nM	$9.86 \cdot 10^{-1}$ nM	0.24
$[V_{165}H]_{max}$	Basement membrane	$5.43 \cdot 10^{2}$ pM	$5.46 \cdot 10^{2}$ pM	-0.63
$[V_{165}H]_{min}$	Basement membrane	$5.40 \cdot 10^{2}$ nM	$5.38 \cdot 10^{2}$ nM	0.45

$[V_{165}H]_{max}$	ECM	$3.15 \cdot 10^1$ pM	$3.15 \cdot 10^1$ pM	0.00
$[V_{165}H]_{min}$	ECM	$3.11 \cdot 10^1$ pM	$3.10 \cdot 10^1$ pM	0.32

Table S3: Comparison of species concentrations at steady state.

FEBio plugin types

As of the latest FEBio release (FEBio 2.7), the following plugins are supported.

- **Materials**: A material plugin implements a new constitutive model that is used to evaluate field variables that are needed to advance the solution (e.g. stress and elasticity tangent for a structural mechanics material).
- **Body load**: a body load adds a volumetric "source" term to the weak formulation of a particular finite element formulation. In the context of mechanics, body loads can be used to model the effects of gravity, centrifugal acceleration, etc.
- **Surface load**: A surface load adds a term to the weak formulation that can represent an external load on the system.
- **Nonlinear constraint**: A nonlinear constraint plugin can enforce a nonlinear relationship between the relevant field variables. This plugin type essentially taps into FEBio's augmented Lagrangian framework.
- **Task**: A task is the highest execution level of FEBio and dictates what FEBio does. Tasks can be useful for implementing algorithms that require repeated calls to FEBio's solvers, for instance, optimization algorithms, or interactions with external libraries.
- **Plot variable**: FEBio's plot file format is self-describing and extendible, which makes it possible to customize its content. The content can be customized via plot variable plugins, which define additional data that can be stored to the plot file.

- **Log file data**: Similarly to FEBio’s plot file, the log file can also be customized. A log file data plugin allows users to create new data variables that will be written to the log file.
- **Callback**: A callback plugin allows users to interact more directly with its solution pipeline. This can be used as an alternative for interacting with external libraries when tasks are not possible. For instance, when time stepping needs to be controlled from the external library.
- **Solver**: Solver plugins implement new finite element formulations that are not directly supported in FEBio. Usually, a solver plugin will also need to implement new materials, surface loads, body loads, and other model components that the solver needs to support.
- **Loadcurve**: A loadcurve controls the time dependency of a model parameter. The default load curve that FEBio supports requires the definition of a set of time-value pairs and parameters that the define how FEBio is to interpolate (and extrapolate) the data. Users can create custom load curves that generate the time-dependency via a procedural method.

REFERENCES

1. Vempati, P., F. Mac Gabhann, and A. S. Popel. 2010. Quantifying the proteolytic release of extracellular matrix-sequestered VEGF with a computational model. PLoS One 5:e11860.
2. Wang, H., A. S. Abhilash, C. S. Chen, R. G. Wells, and V. B. Shenoy. 2014. Long-Range Force Transmission in Fibrous Matrices Enabled by Tension-Driven Alignment of Fibers. Biophysical Journal 107:2592-2603.
3. Quillin, M. L., and B. W. Matthews. 2000. Accurate calculation of the density of proteins. Acta Crystallogr D Biol Crystallogr 56:791-794.
4. Fischer, H., I. Polikarpov, and A. F. Craievich. 2004. Average protein density is a molecular-weight-dependent function. Protein Sci 13:2825-2828.
5. Inc., S. B. 2018. Product Data Sheet - Recombinant Human VEGF-165 (Vascular Endothelial Growth Factor-165). Warwick, PA.
6. Inc, S. B. 2018. Product Data Sheet - Recombinant Human VEGF-121 (Vascular Endothelial Growth Factor-121). Warwick, PA.
7. PhosphoSitePlus. 2018. VEGFR2 - Human. Cell Signaling Technology.