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

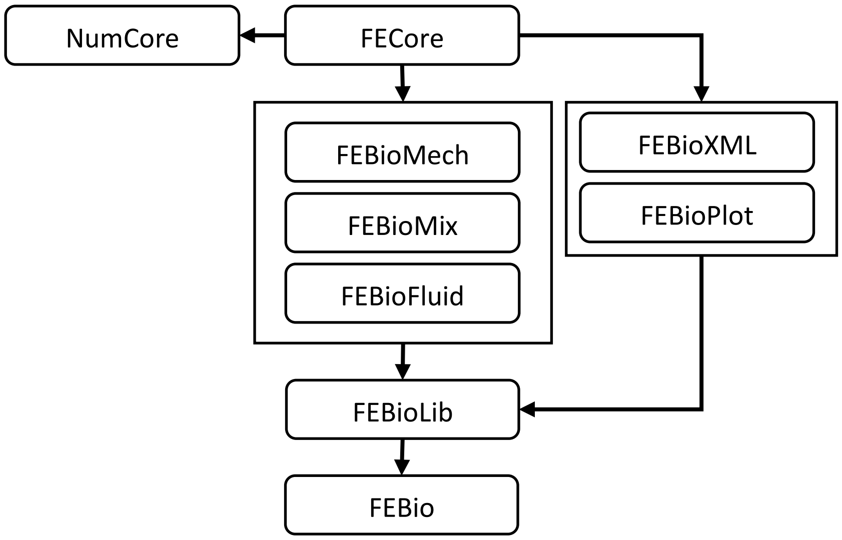


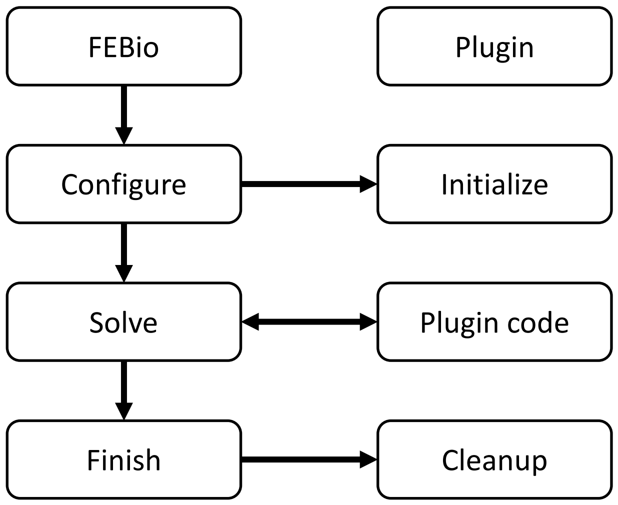
Figure S1. Overview of the modular structure of FEBio. The FECore library is the kernel that centralizes all the features and that all other modules depend on.

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:

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

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:

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

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 $\lambda_b \rightarrow \lambda_a$,

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

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).

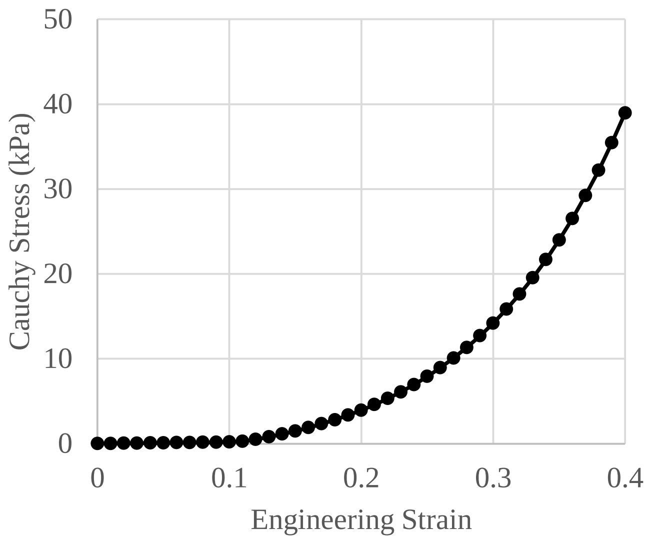


Figure S4. The stress-strain curve of a uniaxial tensile test for the constitutive model proposed by Wang et al (1), obtained from a material plugin developed in FEBio. This graph recovers the results in Figure 1c from the corresponding publication.

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.

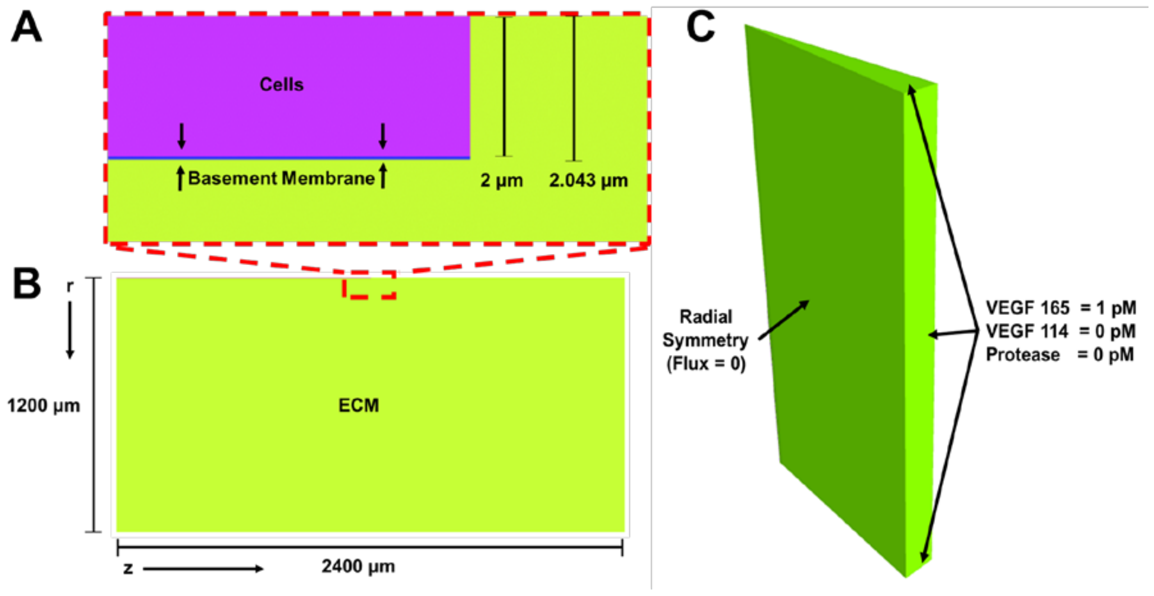


Figure S5. Geometry for Vempati et al. reproduction. **A)** A line of cells (purple, 2 µm radius) extends 1200 µm from the left face along the central *z*-axis. The cells are covered by a 0.043 µm thick basement membrane (blue). The right-most 40 µm of the basement membrane constitutes the tip cell, 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 $2.7\cdot 10^{-12}$ mol/(m$^2\cdot$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.