

# Feedforward equilibrium trajectory optimization with GSPulse

J.T. Wai<sup>\*,a</sup>, M.D. Boyer<sup>\*\*,a</sup>, D.J. Battaglia<sup>a</sup>, A. Merle<sup>b</sup>, F. Carpanese<sup>b</sup>, F. Felici<sup>c</sup>, M. Kochan<sup>d</sup>, E. Kolemen<sup>e,f</sup>

<sup>a</sup>*Commonwealth Fusion Systems, Devens, MA, USA*

<sup>b</sup>*École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*

<sup>c</sup>*Google Deepmind, London, UK*

<sup>d</sup>*Culham Centre for Fusion Energy, Abingdon, United Kingdom*

<sup>e</sup>*Princeton University, Princeton, New Jersey, USA*

<sup>f</sup>*Princeton Plasma Physics Laboratory, Princeton, New Jersey, USA*

---

## Abstract

One of the common tasks required for designing new plasma scenarios or evaluating capabilities of a tokamak is to design the desired equilibria using a Grad-Shafranov (GS) equilibrium solver. However, most standard equilibrium solvers are time-independent and do not include dynamic effects such as plasma current flux consumption, induced vessel currents, or voltage constraints. Another class of tools, plasma equilibrium evolution simulators, do include time-dependent effects. These are generally structured to solve the forward problem of evolving the plasma equilibrium given feedback-controlled voltages. In this work, we introduce GSPulse, a novel algorithm for equilibrium trajectory optimization, that is more akin to a pulse planner than a pulse simulator. GSPulse includes time-dependent effects and solves the inverse problem: given a user-specified set of target equilibrium shapes, as well as limits on the coil currents and voltages, the optimizer returns trajectories of the voltages, currents, and achievable equilibria. This task is useful for scoping performance of a tokamak and exploring the space of achievable pulses. The computed equilibria satisfy both Grad-Shafranov force balance and axisymmetric circuit dynamics. The optimization is performed by restructuring the free-boundary equilibrium evolution (FBEE) equations into a form where it is computationally efficient to optimize the entire dynamic sequence. GSPulse can solve for hundreds of equilibria simultaneously within a few minutes. GSPulse has been validated against NSTX-U and MAST-U experiments and against SPARC feedback control simulations, and is being used to perform scenario design for SPARC. The computed trajectories can be used as feedforward inputs that are connected to the feedback controller to inform and improve feedback performance. The code for GSPulse is available open-source at [github.com/jwai-cfs/GSPulse\\_public](https://github.com/jwai-cfs/GSPulse_public).

---

Email: jwai@cfs.energy

\* Portions of this work were completed while affiliated with Princeton University

\*\* Portions of this work were completed while affiliated with Princeton Plasma Physics Laboratory

## 1. Introduction

### 1.1. Motivation

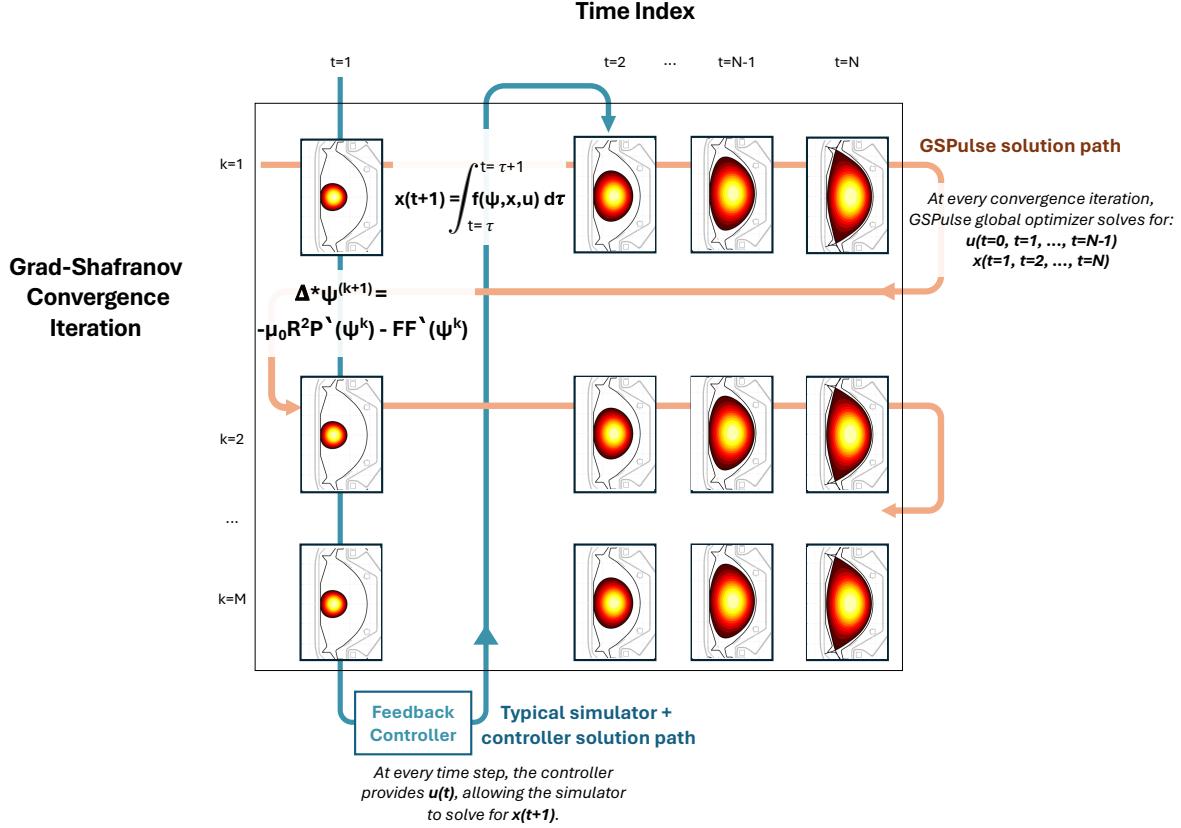
During a tokamak pulse, time-varying voltages and currents are applied to coils within the tokamak, which drive plasma current and create magnetic fields that confine the plasma. The shape and distribution of these magnetic field surfaces is referred to as the plasma equilibrium, and one of the common tasks required for designing new plasma pulses or scenarios is to compute these equilibria using a Grad-Shafranov equilibrium solver. When evaluating a particular scenario it can be important to consider characteristics of the equilibrium such as: the overall shape that can be achieved, the location of the strikepoint legs which influences power exhaust, what the required coil currents are, and what range of plasma current or other core plasma properties are feasible given the device hardware constraints. This type of equilibrium analysis is critical for defining boundaries of the machine's operational envelope.

With feedforward trajectory design, we aim to solve for time-dependent sequences of plasma equilibria since there are a variety of time-dependent effects that are not captured by static equilibrium analysis and can only be captured with dynamic models. One example is the flux transformer effect, where the plasma current must be sustained by continuously evolving the other coils to create an electric field. As the coil currents are ramped throughout the pulse to provide this electric field they will eventually reach their hardware limits. This influences the range of equilibria that are achievable at different times during the pulse. Another dynamic effect is the evolution of currents that are induced in the tokamak's vacuum vessel conducting structures. The magnitude and distribution of these vessel currents are both caused by the equilibrium evolution and play a role in modifying the evolution. Lastly, the impact of power supply voltage limits on the equilibrium is a dynamic effect. The voltages constrain the speed at which the currents and equilibrium can evolve throughout the pulse.

The GSPulse code implements a novel algorithm for designing feedforward equilibrium trajectories which is useful for a variety of operational and design tasks. For example:

- Evaluating the operational envelope of a machine, such as what pulses are achievable given constraints on the coil voltages, currents, and ohmic flux supply.
- Designing new scenarios and reference targets for a pulse. Since not every requested plasma shape or scenario is achievable, it is important to design reasonable reference targets so that the feedback controller is not requested to drive the system along unsuitable trajectories.
- On many tokamaks, portions of the pulse routinely use only feedforward actuation when feedback control is difficult or unavailable. For example, the sequence for plasma initiation usually includes driving the coils along preprogrammed trajectories to apply large voltages for creating plasma breakdown conditions and sustaining plasma burnthrough. This is often done in feedforward-only before transitioning to feedback control. Some reasons for this are that during the highly dynamic early portion of the pulse the phase delay associated with feedback control might be unacceptably high, and that, at low plasma current the signal-to-noise ratio is lower and some real-time reconstruction algorithms may not converge reliably [1] (although simpler non-Grad-Shafranov algorithms could suffice).
- Improving the feedback control performance. There are several ways that feedback control can be improved with complementary application of feedforward. For example, feedback phase delays can be reduced by applying voltage feedforward, improving robustness or other control properties. Another example is that the feedback controller may have extra degrees of freedom, since often multiple different coil current combinations can lead to approximately the same shape. We can use current feedforward to bias the feedback control along advantageous trajectories, for example, trajectories that: are smooth in time, require less actuator effort, avoid imbalanced currents or

coils “fighting” each other, or have more margin to reaching hardware constraints (e.g. force limits).



**Figure 1:** Schematic of the GSPulse algorithm. A sequence of equilibria satisfy the free-boundary equilibrium evolution (FBEE) model if each equilibrium is a fully converged Grad-Shafranov solution and if equilibria are linked in time by the appropriate dynamics. These two conditions are represented by the two axes dimensions, time and Grad-Shafranov convergence. A typical approach for pulse simulating is to converge an equilibrium, determine feedback voltage commands, and evolve the system one step forward in time and solve for the next equilibrium. GSPulse restructures the solution order. Beginning with a rough, unconverged estimate of an equilibrium sequence, GSPulse solves a global optimization problem for the full time evolution of the conductor currents  $x(t)$  and voltages  $u(t)$ . Then, the algorithm takes a step along the convergence dimension, performing a Picard iteration of the equilibrium flux for all time slices. By rearranging the problem structure in this form, the GSPulse optimizer can perform global path-planning for time-dependent tasks such as avoiding hardware constraints or identifying smooth trajectories.

### 1.2. Pulse simulating vs pulse planning

One important point to note is the distinction between pulse simulating and pulse planning (see table 1). While both classes of algorithms follow the same governing equations for free-boundary equilibrium evolution (FBEE), they do so with different methodologies and application purposes. A pulse simulator, sometimes called the forward evolutive problem, evolves the equilibrium step-by-step given a set of voltages received from a feedback controller in the loop. By contrast, the GSPulse pulse planner solves the inverse evolutive problem of computing trajectories of voltages, currents, and equilibria given a set of target equilibrium features. It finds an equilibrium evolution that best matches a set of user-specified targets. This is fundamentally a global problem in that we need to optimize the trajectory considering the entirety of the evolution, which modifies the Grad-Shafranov convergence strategy (fig. 1) discussed in the next section.

	<b>Static</b>	<b>Evolutive</b>
<b>Reconstruction</b>	<b>Equilibrium reconstruction:</b> Given diagnostic measurements, compute the best-fit plasma equilibrium.	<b>Dynamic equilibrium reconstruction:</b> Given a time-series of diagnostic measurements and taking into account the model dynamics (e.g. via Kalman filters), compute the best-fit evolution of plasma equilibria.
<b>Forward</b>	<b>Forward equilibrium solver:</b> Given conductor currents and plasma constraints, compute the plasma equilibrium.	<b>Pulse simulator:</b> Provided coil voltages at each time step, compute the evolution of conductor currents and plasma equilibria.
<b>Inverse</b>	<b>Inverse equilibrium solver:</b> Given a desired equilibrium shape and other targets, compute the conductor currents and plasma equilibrium.	<b>Pulse planner:</b> Given a desired evolution of equilibrium shapes and other targets, compute the evolution of conductor voltages, currents, and plasma equilibria.

**Table 1:** A classification scheme for various types of equilibrium algorithms, although it is noted that not all equilibrium codes necessarily fit cleanly into this schema and might combine features of multiple categories. GSPulse is an “evolutive inverse” equilibrium solver for use in pulse planning, and when solving for a single equilibrium, collapses into a static inverse solver.

An additional difference is that, due to the convergence strategy and inherent vertical stabilization (discussed further in section 2.1), the GSPulse pulse planner can take larger time steps than a pulse simulator while remaining vertically stable. This allows for a reduced computational cost of the solution. Some notable examples of forward evolutive pulse simulators are DINA [2], FGE [3], FreeGSNKE [4], and GS evolve [5].

### 1.3. Comparison against previous approaches

The core idea of the GSPulse algorithm is that it rearranges the structure of the FBEE model to enable performing a time-dependent optimization over the entire sequence of equilibria. This improves on some other methods for designing feedforward trajectories. While there is no standard method for defining these trajectories, one of the typical approaches would be to first design a series of static equilibria and then go back and modify these equilibria with an attempt at compensating for the dynamic effects such as induced vessel currents. One of the downsides to this approach is that it can be computationally expensive, since each of the equilibria must fully converge more than once. Additionally, while the equilibria may have the appropriate dynamic linking, it is still difficult to tune the trajectory into an optimal trajectory, since all of the optimization comes from user-effort in defining the sequence of equilibria, and not from an explicit trajectory optimization.

Several earlier solvers have also treated the inverse-evolutive optimization problem, including CEDRES++ [6] and an algorithm by Blum [7], which have now been incorporated into the NICE code [8]. These solvers use Sequential Quadratic Programming (SQP) methods to optimize for sequences of equilibria performing a similar task as GSPulse. The main differences of these solvers versus GSPulse are: the exact formulation of the cost functional where GSPulse includes additional terms on the derivatives and smoothness of trajectories, and that the SQP-based solvers iterate on the full state vector solution directly, whereas GSPulse splits the iteration into optimizing the vacuum conductor evolution (via Model Predictive Control formulation) and Picard iterations of the plasma state. This framework allows GSPulse to re-use sensitivity matrices used in the optimization, and forgo additional gradient or adjoint state calculations. The implicit assumption is that it is sufficient to simply update the effect of the plasma contribution to the cost function at each iteration, and not necessary to use the gradient of the cost w.r.t plasma contribution within the optimization step. This structure is very beneficial for computational efficiency although perhaps reduces the optimality of the final solution.

Some other codes within this design space are Tokamaker [9], which also includes some time-dependent effects and can solve for sequences of equilibria, but without performing global trajectory optimization. Additionally, recent work on the FBT [10] equilibrium code has extended it to the dynamic case to create the FBT-Evolutive pulse planner, by incorporating the GSPulse equations presented in the next sections (i.e., MPC-like global vacuum optimization coupled with Grad-Shafranov Picard iterations).

## 2. Algorithm principles

**Feedforward trajectory problem**

**Given:**

- $R, M$  : axisymmetric conductor resistances and mutual inductances
- $R_p(t), W^{\text{th}}(t), I_p^{\text{targ}}(t)$  : plasma resistance, thermal energy, and target plasma current
- basis functions for  $FF'$ ,  $P'$ : Grad-Shafranov source profile basis functions
- $r_{\text{bbbs}}(t), z_{\text{bbbs}}(t), r_x(t), z_x(t)$  : target plasma boundary and x-points

**Minimize a weighted cost function of shaping errors and actuator effort,**

$$J = \int_{\Omega_t} \|u\|_{W_u}^2 + \|\dot{u}\|_{W_{\dot{u}}}^2 + \|\ddot{u}\|_{W_{\ddot{u}}}^2 + \|e\|_{W_e}^2 + \|\dot{e}\|_{W_{\dot{e}}}^2 + \|\ddot{e}\|_{W_{\ddot{e}}}^2 dt \quad (\text{eq. 1})$$

**Subject to free-boundary equilibrium evolution (FBEE), and constraints:**

$\Delta^* \psi = -\mu_0 r^2 P' - FF' - \mu_0 r J_\phi^{\text{ext}}$ <i>(FBEE)</i>	$I_{\min} < I < I_{\max}$ <i>(typical actuator constraints)</i>
$v_i = R_i I_i + M_{ij} \dot{I}_j + \dot{\Phi}_i^{\text{pla}}$ <i>(eq. 3, circuit dynamics)</i>	$V_{\min} < V < V_{\max}$
$V_p = -\dot{\psi}_{bry} = R_p I_p + \frac{1}{I_p} \frac{d}{dt} \left( \frac{1}{2} L_I I_p^2 \right)$ <i>(eq. 4, Ejima)</i>	$\dot{V}_{\min} < \dot{V} < \dot{V}_{\max}$

**Figure 2:** Summary of the feedforward trajectory optimization problem. GSPulse minimizes a quadratic cost function of the isoflux shaping errors and actuator effort, subject to the free-boundary equilibrium evolution (FBEE) governing equations, and also subject to any power supply constraints.

This section describes the working principles for the GSPulse algorithm. The full derivation is provided in [Appendix A](#). The premise is to minimize a cost function associated with the shaping evolution, while simultaneously satisfying the FBEE governing equations and actuator constraints. The cost function used is:

$$J = \int_{\Omega_t} \|u\|_{W_u}^2 + \|\dot{u}\|_{W_{\dot{u}}}^2 + \|\ddot{u}\|_{W_{\ddot{u}}}^2 + \|e\|_{W_e}^2 + \|\dot{e}\|_{W_{\dot{e}}}^2 + \|\ddot{e}\|_{W_{\ddot{e}}}^2 dt \quad (1)$$

In this equation,  $J$  is the cost,  $\Omega_t$  is the time period of interest,  $u$  is the shaping coil power supply voltages, and  $e$  is the vector of isoflux shaping errors which can include signals like coil currents, shape control point flux errors, field at the x-points, and the boundary flux or loop voltage.  $W$  are weighting matrices for each cost term. In the implementation, the cost function is discretized to discrete time points and equilibria. Note that there are quadratic penalties on the signal values, as well as on the signal first and second derivatives (with respect to time). First-derivative weights are used to

penalize trajectories that ramp too aggressively, and second-derivate weights are used to penalize non-smooth trajectories. Using principles derived from Model Predictive Control (MPC), the cost function subject to the FBEE constraints can be reformulated into the form of a standard constrained quadratic program (QP). The standard form QP can be solved by standard software packages such as MATLAB's **quadprog**.

The current implementation of GSPulse is able to solve the cost function for parameters that are linear with respect to voltage or current (electric fields, magnetic fields, shape flux errors, voltages, and currents) and future work will aim to extend this to additionally optimize for parameters that are nonlinear with respect to currents, such as structural forces.

The FBEE governing equations are (using a COCOS-17 coordinate system [11]):

$$\Delta^* \psi = -\mu_0 r^2 P' - FF' - \mu_0 r J_\phi^{ext} \quad (2)$$

$$v_i = R_i I_i + M_{ij} \dot{I}_j + \dot{\Phi}_i^{pla} \quad (3)$$

$$V_p = -\dot{\psi}_{bry} = R_p I_p + \frac{1}{I_p} \frac{d}{dt} \left( \frac{1}{2} L_I I_p^2 \right) \quad (4)$$

See [Appendix A.1](#) for the definitions of each variable. The first equation is the well-known Grad-Shafranov equation [12, 13] for axisymmetric equilibrium force balance, the second describes axisymmetric circuit dynamics within a tokamak, and the last describes the plasma current evolution. The plasma current governing equation is sometimes presented in different forms. The form here used by GSPulse is the scalar Ejima equation [14] which describes how the plasma surface voltage must compensate for plasma resistance and changes to the plasma current and inductance. Note that Ejima equation (and GSPulse) do not consider non-inductive current, but could be extended by replacing the resistive voltage term above with  $R_p(I_p - I_{non-inductive})$  [15]. GSPulse can also be configured to accept the loop voltage or boundary flux directly, instead of computing it from the Ejima model.

The employed solution strategy is to use a Picard iteration scheme for the Grad-Shafranov plasma flux distribution and couple this with a dynamic optimization. This is analogous to the Picard iteration scheme employed in many static equilibrium solvers, that alternates between updating the conductor currents and the Grad-Shafranov equilibrium. The major difference is that in the GSPulse case we include the dynamics as a constraint in the optimization and also solve for all of the voltages and currents over the time window of interest. Figure 1 gives a cartoon of the convergence strategy used in GSPulse, where we alternate between the global MPC-like optimization and plasma Picard updates of the flux distribution.

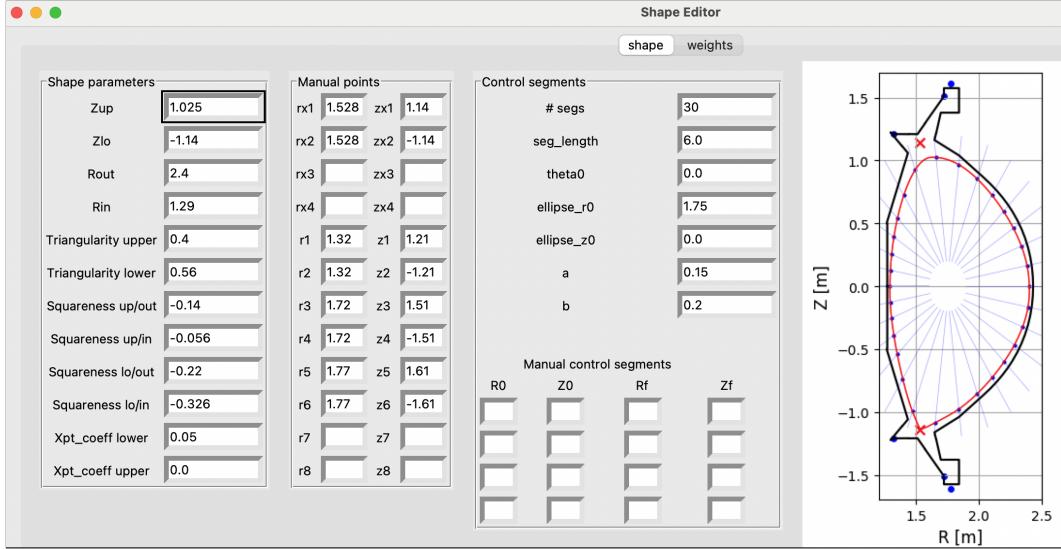
A rough outline of the algorithm is provided below. For a detailed mathematical treatment, see [Appendix A](#).

- 1. Configuration:** In this step the user defines a standard set of inputs which includes the tokamak geometry, mutual inductance matrices, coil and vessel resistances, core plasma properties needed to solve the Grad-Shafranov equation such as the  $P'$  and  $FF'$  profile shapes, constraints on the plasma properties such as the plasma current, thermal energy, or internal inductance, and the target shapes and weights. Note that GSPulse itself does not derive mutual inductance and Green's function tables from the tokamak geometry, but instead has been configured to accept geometry and inductance tables computed from both the TokSys [16] and MEQ [17] suites of equilibrium codes.

For the inductive plasma current drive model, the user can either specify a waveform for plasma

resistance, which is used in the Ejima equation, or directly specify the target boundary flux evolution. Shape targets can be created with the shape editor GUI (fig. 3). Weights are specified for various shaping parameters and coil current and coil voltage characteristics. These are used to tune the solution to give a desired trajectory – for example they can be used to tune how important achieving a particular strike point is, or to penalize non-smooth trajectories for the currents.

As part of the configuration, the user can also specify the initial conditions for the solver such as the initial coil and vessel currents and initial voltages. If unspecified, the initial conditions are treated as free parameters for the optimization.



**Figure 3:** GSPulse shape editor GUI using an outline of the SPARC limiter. The user specifies target shapes in terms of a smaller number of parameters such as elongation and triangularity, and the corresponding timebase for the shapes.

2. **Initialization:** In this step, based on the target shape and target  $I_p$  the algorithm obtains a rough estimate of the plasma current distribution at each step.
3. **Compute target boundary flux:** In this step, GSPulse integrates the scalar Ejima equation (eq. (A.5)) to compute the boundary flux evolution corresponding to the target  $I_p$  evolution. The boundary flux becomes a target for the conductor optimization.
4. **Optimize conductor evolution:** Re-formulate the circuit dynamics and shaping cost function in terms of a quadratic program, and solve this QP to obtain the conductor evolution that minimizes shaping errors.
5. **Grad-Shafranov Picard iteration:** Given the conductor currents and profiles from the  $k^{th}$  iteration, perform a Picard update of the Grad-Shafranov equation to find the flux distribution at the  $(k+1)^{th}$  iteration. GSPulse has the option of performing this step natively, or via calling subroutines of the MEQ suite [18] of equilibrium codes.
6. **Repeat from Step 3 until convergence.**

#### 2.1. Vertical stabilization during convergence iterations

One consideration is that equilibrium solvers that use Picard iterations can be vertically unstable. It is well-known that, by itself, the Grad-Shafranov Picard iteration

$$\Delta^* \psi^{k+1} = -\mu_0 R P'(\psi^k) - F F'(\psi^k), \quad (5)$$

is unstable when applied repeatedly without feedback [19]. However, this tends to be more of a problem for reconstruction solvers and forward solvers. For example, RT-EFIT [20] and the original LIUQE [18] explicitly add a vertical shift free parameter that could be updated across iterations, and forward simulators like FreeGSNKE [4], FGE [3], and NICE forward mode [8] avoid Picard iterations in favor of Newton or Newton-like methods.

However, as an inverse solver, GSPulse tends to be robust against this vertical instability in practice. This is linked to the shape target objective within the optimization, which applies corrective action between iterations to keep the plasma shape from drifting. The stability is also related to the time resolution of the GSPulse configuration. When very small timesteps are used (roughly on the order of the vertical instability timescale), the enforced dynamic constraint prevents GSPulse from taking sufficiently large stabilizing action and the Picard vertical instability can reappear. With larger timesteps, GSPulse is generally stable given appropriate weighting of the cost function objectives. The ability to take large time steps is beneficial for computational time and makes GSPulse an efficient tool for exploring design space.

### 3. Results

#### 3.1. Validation against NSTX-U 204660

GSPulse has been validated against experimental results and simulations for several tokamaks including NSTX-U [21] and MAST-U [22]. Figure 4 illustrates a validation study against NSTX-U shot 204660. The objective of this GSPulse run is to re-create the exact progression of equilibria and coil currents that was achieved during the actual feedback-controlled pulse [23], as defined by the NSTX-U EFIT01 reconstruction [24]. Since GSPulse internally enforces the FBEE model, we expect that any modeling discrepancies or implementation errors would prevent the algorithm from being able to exactly match both the target shapes and coil currents, up to the limit of accuracy for EFIT01.

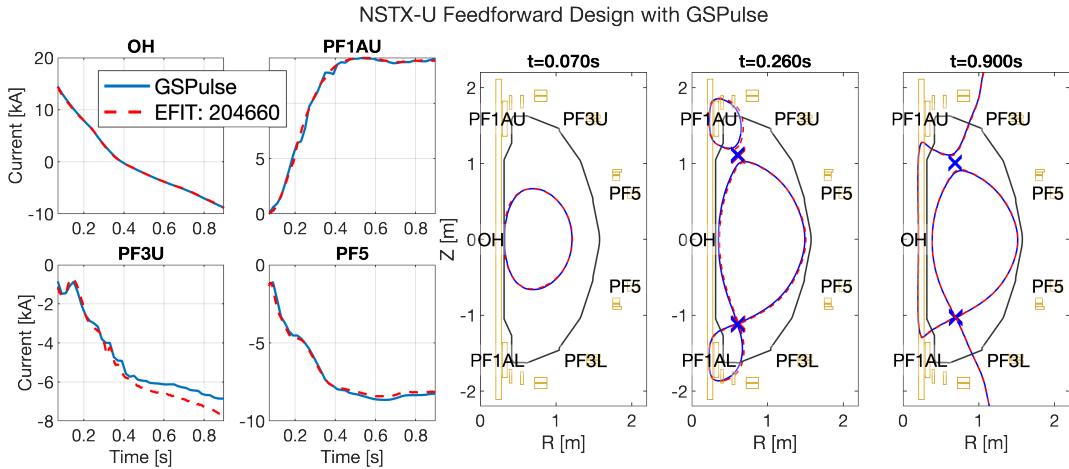
For this validation case, GSPulse is configured as follows:

- The initial condition (equilibrium, coil currents, and vessel currents) are provided from an EFIT [25] reconstruction at  $t = 50\text{ms}$ . The evolution of the boundary flux is specified as a direct target based on the EFIT01 (magnetics-only) loop voltage. Core plasma properties (stored thermal energy,  $I_p$ , and  $P'$  and  $FF'$  profile basis function shapes) are also obtained from the EFIT01 values.
- The shape targets are derived from the actual shape progression of the pulse. This is a non-optimal shape progression since it contains disturbances and feedback oscillations, but is used for the validation case. For shape targets, we specify only the x-point locations and the core plasma boundary. The strike points are not included as targets in this case.
- The coil current trajectories from shot 204660 are included as targets for GSPulse and weighted lightly. Of course the advantage of GSPulse is that it can identify new, optimal coil trajectories and in general it is not necessary to a-priori specify coil currents. However, they are included for the validation run in order to guide the solution to the best experimental fit, since some coils can have similar impacts on the equilibrium.
- We solve for an equilibrium at every 20ms interval between  $t=50\text{ms}$  and  $t=910\text{ms}$  (44 equilibria in total). This calculation takes 1-2 minutes to run on a laptop.

The results of the validation run are shown in fig. 4. In general we see robust agreement for both the achieved equilibria and current trajectories. The current in the PF3U coil begins to deviate from

the experimental values in the second half of the pulse, reaching a variation of -7.2kA versus -8kA by end-of-pulse. This hints at a small amount of model discrepancy; for example, inaccurate estimates of the distribution of resistances and inductances within the vessel wall could have this type of effect. It is also possible that this discrepancy could be resolved with more refined estimates of the internal profile evolution. Since EFIT01 uses simple low-order polynomials for the internal profile shapes, and the GSPulse validation cases enforces the same profile evolution, inaccurate EFIT01 profiles could lead GSPulse to inaccurate estimates of the induced vessel currents.

One of the emergent features from this run, observable in the equilibria of fig. 4, is that the outer strikepoint moves radially outwards throughout the pulse, even though the inner strike point position, x-point position, and core boundary shape remain relatively fixed. This NSTX-U pulse used a subset of the divertor shaping coils and consequently could not control both the outer strike point and x-point positions. The motion of the strike point is driven by a change in the time-varying current of the OH coil which does not supply a uniform field, a known feature of the NSTX-U design [26]. Since GSPulse is adequately capable of modeling this type of effect, it could be a useful operator tool for evaluating the tradeoffs in shape evolution for different choices of coils and coil limits under time-varying conditions.



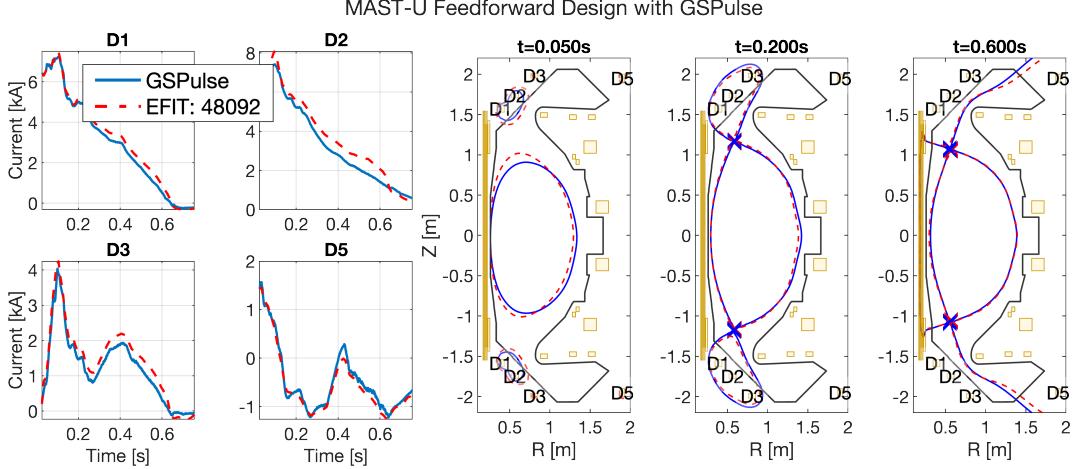
**Figure 4:** Validation results of GSPulse against NSTX-U shot 204660. The coil currents are obtained from EFIT01 (magnetics-only). Flattop for this pulse is at 0.38 seconds but the non-uniform field produced by the OH coil requires the PF3 current to continue drifting during flattop in order to maintain core plasma shape. The result is that the outer strike point is not held constant during the pulse if the core shape is also fixed.

### 3.2. Validation against MAST-U 48092

Figure 5 illustrates a similar validation calculation performed for MAST-U shot 48092. The configuration setup is nearly identical to the NSTX-U validation case above. In particular, the initial condition and plasma properties are taken from the EFIT reconstructions. Large weights are applied for matching the target shape evolution, and small weights are applied for matching the coil currents. Again, the goal is to find a GSPulse solution that best-matches the experiment, and verify that the internal FBEE model is consistent with reality. The run is set up to solve for equilibria every 10ms between t=25ms and t=755ms.

The results for this run are shown in fig. 5 again showing strong agreement in the coil currents and equilibrium progression. There are some slight variations in the currents, and the early plasma boundary is slightly less elongated (order 5-10cm). This is reasonable, as during the earliest portions of the pulse, the combination of low plasma current and large vessel currents make the equilibrium more sensitive to the wall model inductances and resistances. During the pulse, the strike leg is moved

radially outwards into the super-X divertor chamber. There is a 5cm discrepancy in the strike leg position which also hints at some model mismatch in this area of the tokamak.



**Figure 5:** Validating GSPulse against MAST-U shot 48092 shows consistency between predictions and experiment.

For these validation cases, the configuration has been setup to exactly match the actual pulse. However, after model verification, it is straightforward to set up GSPulse to identify desirable trajectories that are not a-priori specified. Future work will also aim to improve the trajectories beyond the existing scenarios, for example, finding smoother and less aggressive coil waveforms, reducing the amount of time that the plasma is limited, and bringing the strikepoint or super x-points into position faster.

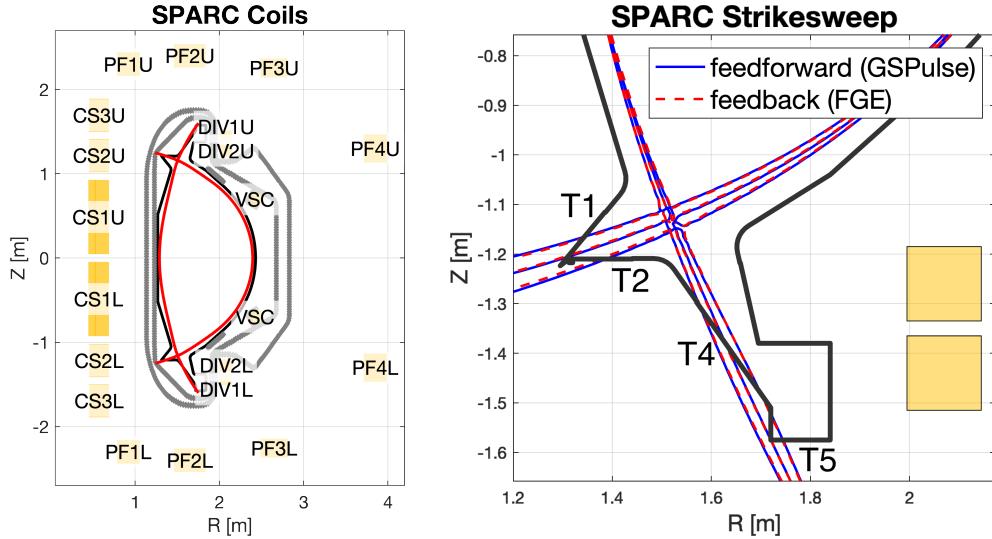
### 3.3. Design of SPARC strikesweep scenario and validation against FGE

We also validate GSPulse against the FGE [3] code during a strikepoint sweeping simulation for the SPARC tokamak [27]. FGE has been used and tested extensively on a number of tokamaks. It was also the simulation environment for a reinforcement learning magnetic controller [28] that successfully controlled the TCV tokamak.

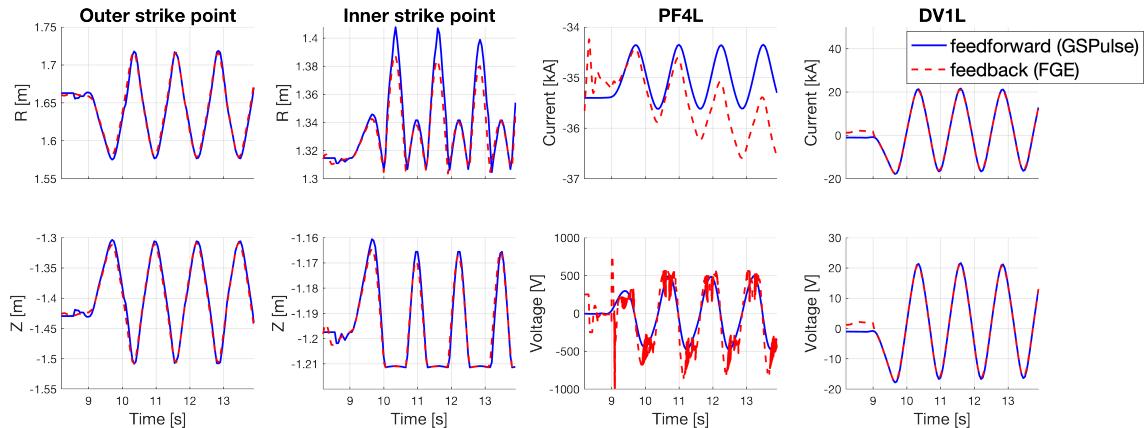
This validation case is configured as follows:

- The initial condition is an 8.7MA, double-null, equilibrium. GSPulse is used to design the coil current and voltage trajectories for sweeping the strikepoints of this equilibrium across the divertor tile surfaces, as shown in fig. 6b, at a frequency of 0.8Hz.
- The FGE feedback simulation is configured to use the same initial condition as GSPulse. Since FGE is a pulse simulator, we need to specify feedback control within the loop. A vertical controller is designed that uses the VSC coils to apply feedback on the plasma vertical position. Additionally, a PID radial controller is designed that uses the PF4U/L coils to apply feedback on the plasma radial outer gap. Additionally, the voltages computed by GSPulse are used as feedforward inputs to the simulation. In summary, the voltage commands applied during the simulation are a combination of vertical feedback, radial feedback, and GSPulse-computed voltages.
- Both FGE and GSPulse use the same geometry definitions, inductance matrices, and resistances. GSPulse is configured to solve for the voltages and equilibria every 50ms, and FGE is configured for a timestep of 1ms.

The pulse-planning (GSPulse) and pulse-simulating (FGE) results are shown in figs. 6 and 7. There is strong agreement between these models, again illustrating that GSPulse is reliably solving the FBEE problem and can be used predictively. As seen in fig. 7, there is a little bit of drift in the PF4 current which also has a small impact on the inner strike point position. This drift is a result of using the PF4 coils for outer gap feedback control in addition to feedforward voltage actuation, and could be compensated by also adding a command to track the feedforward currents. Smaller differences could also be achieved by designing the feedback simulation to more closely match the feedforward-only case by removing derivative control (which adds some noise to the voltage) and using the GSPulse-achieved shapes as shape targets, however, the simulation as depicted is more representative of how feedforward targets would be added to the actual SPARC feedback controller.



**Figure 6:** a) Geometry layout of the SPARC shaping coils. b) The equilibrium last-closed flux surface at different times within the strikepoint sweeping scenario. The feedforward results are obtained with GSPulse and show good agreement with the full closed-loop feedback control simulation computed with FGE. Note that this example is an early exploratory strikesweep scenario, and the SPARC design has evolved to disallow sweeping the inner strikepoint across the Tile 1/Tile 2 boundary.



**Figure 7:** Representative time traces for the strikepoint sweeping scenario, comparing the feedforward predictions from GSPulse against the full closed-loop feedback control simulation using FGE. There is strong agreement in the predicted strikepoint locations, although a slight drift in the PF4 current due to outer gap control introduces some effect on the inner strike point.

### 3.4. SPARC scenario design: plasma breakdown and startup

Feedforward is most commonly employed during the early phases of a pulse, when the sequencing for plasma breakdown and startup is critical, and at low  $I_p$  before reliable equilibrium reconstructions are available. GSPulse is a natural tool for designing the electromagnetic trajectories required for plasma breakdown since this is a trajectory optimization of the vacuum fields [29]. Figures 8 and 9 illustrate an example designing plasma breakdown conditions for SPARC. The electromagnetic conditions required for reliable plasma startup can be summarized as [30, 31]:

$$\begin{aligned} E_\phi &> 0.3 \text{ V/m, no ECH pre-heating} \\ E_\phi B_\phi / B_\theta &> 1000 \text{ V/m} \end{aligned} \quad (6)$$

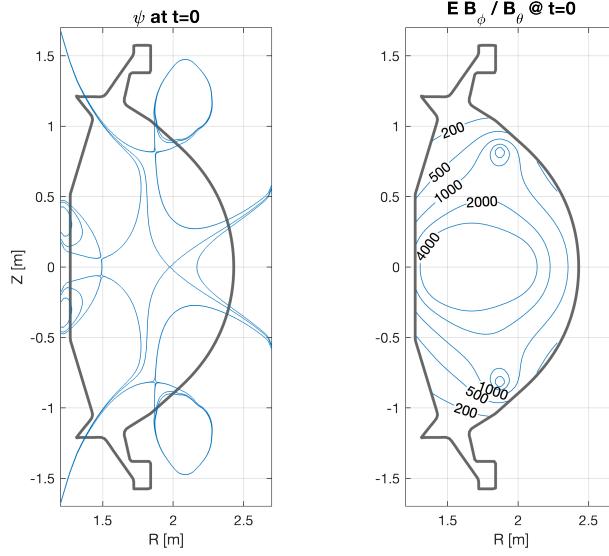
These are “instantaneous” conditions that must be satisfied near the plasma formation point (typically near the center of the machine or on the inboard limiter) in order to achieve main ion burn-through and plasma breakdown. However, in practice a number of additional time-dependent characteristics are necessary or desirable. These include: avoiding accidentally creating breakdown conditions (reverse breakdown) before the desired start time, sustaining high electric field for a sufficiently long time to burn through the main plasma ion impurities, achieving a vertical field evolution that provides radial force balance as the plasma current channel forms and grows, achieving a radial field evolution that transitions from field null conditions to providing passive vertical stability on a reasonable timescale such that the growing plasma does not go unstable, and pre-charging and balancing the coil currents to maximize the flux that is available for inductive current drive during the pulse. For additional detail on specifying the breakdown trajectory targets, we refer readers to Leuer’s work for KSTAR and EAST [31]. However, assuming that the target electric and magnetic field evolution have been identified with the appropriate physics modeling, finding the feedforward currents and voltages is easily transformed to a target-tracking optimization problem for GSPulse.

From the GSPulse perspective, the objective can be summarized as:

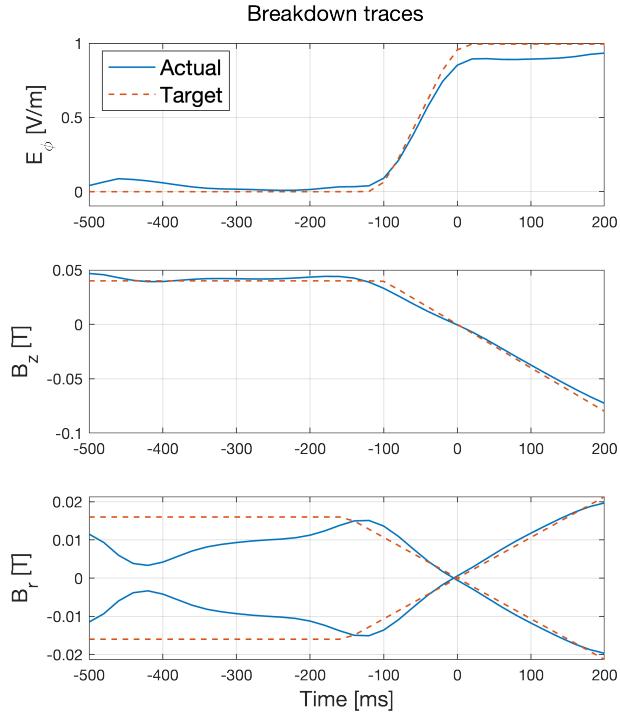
- Track a trajectory for  $E_\phi(t) = -\frac{d\psi}{dt}$ .
- Track trajectories for  $B_r(t)$ ,  $B_z(t)$ .
- Maximize  $\psi$  at  $t = 0$ .
- Satisfy coil current and voltage constraints.

The location for evaluating  $E$ ,  $B$  and  $\psi$  could be either area-averaged quantities usually near the center of the tokamak or at specified locations and is generally an operator preference or tuning parameter. The target trajectories for SPARC at specified locations are shown in fig. 9 and indicate that the breakdown conditions can be reliably met on SPARC. These results are generally consistent with previous breakdown simulations where coil current and voltage trajectories were tuned by hand [32, 33]. The peak electric field is 0.9V/m and limited by the power supply voltage capabilities of the CS1 coil. This method is able to find a larger area-averaged  $E_\phi B_\phi / B_\theta$  than was previously found by hand. However, the main advantage is that the operator will be able to tune the breakdown scenario by specifying the desired signal waveforms directly, instead of having to simultaneously adjust and balance multiple coils.

Currently within GSPulse, the plasma startup optimization (vacuum-only) and standard equilibrium evolution (vacuum optimization plus plasma Picard iterations) are treated as separate problems, due to the numerical difficulty of converging very low  $I_p$  equilibria that are present shortly after  $t = 0$ . However, future work will extend GSPulse to solve across this transition period and perform breakdown optimization simultaneously with the equilibrium evolution in the first few seconds of the pulse.



**Figure 8:** Left: magnetic flux surfaces at  $t = 0$  with multiple x-points implying a large area with low  $B_p$ . Right: Contours of  $E_\phi B_\phi / B_\theta$  indicating that there is a large area satisfying the  $> 1000 \text{V/m}$  requirement from eq. (6).



**Figure 9:** Target and achieved breakdown traces for electric field (evaluated at  $R=1.6\text{m}$ ,  $Z=0\text{m}$ ), vertical field (evaluated at  $R=1.6\text{m}$ ,  $Z=0\text{m}$ ) and radial field (evaluated at  $R=1.6\text{m}$ ,  $Z=\pm 0.6\text{m}$ ). The target trajectories were designed following principles from [31], and the achieved trajectories were computed with the GSPulse optimization.

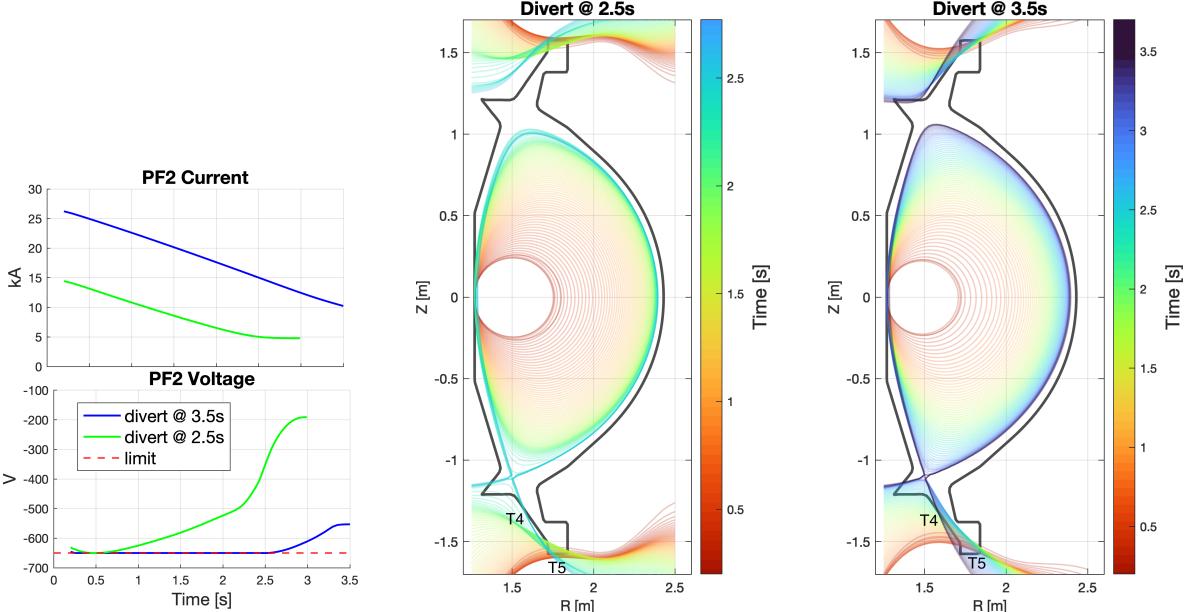
### 3.5. SPARC scenario design: diverting as fast as possible

Figure 10 gives an example of using GSPulse to improve the scenario design for the objective of diverting the plasma as fast as possible. Given an initial  $I_p = 200\text{kA}$  equilibrium shortly after plasma

breakdown, the goal is to ramp the plasma current at 1MA/s and simultaneously divert the plasma as fast as possible. Diverting is beneficial because it reduces the influx of impurities to the plasma and allows for earlier ICRF heating. Restrictions are imposed that the strikepoints must be placed only on the divertor tiles, and that the outer strikepoint avoid contact with tile 5 which is not intended for high heat flux.

Within GSPulse, this objective is set up as a point A to point B optimization. In other words, the initial condition is fully specified, and the final equilibrium shape is also specified, but there are no shape targets or weights associated with the intermediate times. There are only penalties on the smoothness of the trajectory. We solve for an equilibrium every 30ms or roughly 80-120 equilibria in these scenarios, scanning the endpoint time until the trajectory results are satisfied.

The results are summarized in fig. 10. With the initial condition specified by the nominal breakdown scenario, the time to divert is 3.5 seconds. The challenge in this case is ramping the PF2 coil current fast enough to reduce the curvature on the outer strike point so that it always stays on tile 4 and avoids touching tile 5 (labeled T4/T5 in fig. 10). Note that the isoflux line is allowed to touch tile 5 when the plasma is still limited, but not when diverted. This constraint is verified manually and is not part of the GSPulse optimization constraints, except that the target strikepoint is specified. If we loosen the restriction on the initial plasma current equilibrium, and allow the PF2 current to float as a free parameter, then this reduces the time to divert to 2.5 seconds. This again points at the value of simultaneously optimizing the plasma breakdown scenario with the first few seconds of plasma equilibria, since the details of plasma breakdown do affect features like time-to-divert.



**Figure 10:** Optimizing a SPARC scenario in order to minimize the time required to divert the plasma. With the nominal plasma breakdown scenario, it takes 3.5 seconds to divert because of the requirement to keep the strikepoint from hitting tile 5 which is limited by the PF2 current evolution. If PF2 current at  $t = 0$  is allowed to be a free parameter of the optimization then GSPulse identifies a lower starting current that keeps the outer strike point leg straighter and allows for diverting by 2.5 seconds.

#### 4. Conclusion and outlook

In conclusion, the GSPulse algorithm is a novel method of solving free-boundary equilibrium evolution (FBEE) that reformulates the problem in order to efficiently perform trajectory optimization.

This has a wide variety of applications for scenario development and scoping, as well as utility in providing feedforward trajectories to improve feedback control performance. GSPulse has been validated against experiments and the pulse simulator FGE. GSPulse continues to undergo active development and feature additions. Additional features that are under development include: adding the capability to optimize across the plasma breakdown transition (before plasma, and during very low- $I_p$  plasmas), adding a Python API to the code base so that GSPulse can interface with Python codes, and extending the optimization to be able to optimize for more complex error types, such as the structural forces exerted on the coils which have a nonlinear relationship with respect to coil currents. The code is available open-source at [https://github.com/jwai-cfs/GSPulse\\_public](https://github.com/jwai-cfs/GSPulse_public).

## Appendix A. GSPulse Algorithm

### Appendix A.1. Model equations:

This appendix describes the GSPulse algorithm in detail. To recall, the objective of GSPulse is to minimize a quadratic cost function on the actuator effort and shaping errors:

$$J = \int_{\Omega_t} \|u\|_{W_u}^2 + \|\dot{u}\|_{W_{\dot{u}}}^2 + \|\ddot{u}\|_{W_{\ddot{u}}}^2 + \|e\|_{W_e}^2 + \|\dot{e}\|_{W_{\dot{e}}}^2 + \|\ddot{e}\|_{W_{\ddot{e}}}^2 dt \quad (\text{A.1})$$

This cost function is discretized such that the specific implementation that is optimized for is:

$$J := \sum_{k=k_0}^N \|u_{k-1}\|_{W_u}^2 + \|\Delta u_{k-1}\|_{W_{\Delta u}}^2 + \|\Delta^2 u_{k-1}\|_{W_{\Delta^2 u}}^2 + \|e_k\|_{W_e}^2 + \|\Delta e_k\|_{W_{\Delta e}}^2 + \|\Delta^2 e_k\|_{W_{\Delta^2 e}}^2, \quad (\text{A.2})$$

$k_0 := 1$  for absolute terms, 2 for first-order difference ( $\Delta$ ) terms,

3 for second-order difference ( $\Delta^2$ ) terms

The cost should be minimized while subject to the Grad-Shafranov equilibrium force balance condition, axisymmetric conductor circuit dynamics, and  $I_p$  dynamics via the Ejima equation:

$$\begin{aligned} \Delta^* \psi &= -\mu_0 R J_\phi \\ J_\phi &:= J_\phi^{pla} + J_\phi^{vac} \\ J_\phi^{pla} &= RP'(\psi) + \frac{FF'(\psi)}{\mu_0 R} \end{aligned} \quad (\text{A.3})$$

$$u_i = R_i I_i + M_{ij} \dot{I}_j + \dot{\Phi}_i^{pla} \quad (\text{A.4})$$

$$V_p = -\dot{\psi}_{bry} = R_p I_p + \frac{1}{I_p} \frac{d}{dt} \left( \frac{1}{2} L_I I_p^2 \right) \quad (\text{A.5})$$

where,

- $u_k$  = actuator input (voltage) at time step k  
 $\Delta u_k := u_{k+1} - u_k$  = actuator input first-order-differences  
 $\Delta^2 u_k := u_{k+2} - 2u_{k+1} + u_k$  = actuator input second-order-differences  
 $W_{(\cdot)}$  = cost function weighting matrix  
 $e_k$  = output errors such as isoflux shaping and field errors at time step k  
 $\Delta e_k := e_{k+1} - e_k$  = output error first-order differences  
 $\Delta^2 e_k := e_{k+2} - 2e_{k+1} + e_k$  = output error second-order differences  
 $\Delta^*(\cdot) = r \frac{\partial}{\partial r} \left( \frac{1}{r} \frac{\partial(\cdot)}{\partial r} \right) + \frac{\partial^2(\cdot)}{\partial z^2}$  = Grad-Shafranov operator  
 $\mu_0$  = vacuum permeability constant  
 $r$  = radial coordinate  
 $J_\phi$  = toroidal current density distribution  
 $P$  = pressure  
 $\psi$  = poloidal flux per unit radian  
 $F = rB_t$  = radius times toroidal magnetic field.  
 $i, j$  = indices for conducting elements  
 $R_i$  = resistance in conductor i  
 $I_i$  = current in conductor i  
 $M_{ij}$  = mutual inductance between conductors i and j  
 $\dot{\Phi}_i^{pla}$  = plasma coupling term, discussed in text  
 $V_p$  = plasma loop voltage  
 $\psi_{bry}$  = flux at plasma boundary  
 $R_p$  = total plasma resistance  
 $I_p$  = total plasma current  
 $t$  = time  
 $L_I = \int_{\Omega_p} \frac{B_p^2}{\mu_0 I_p^2} dV$  plasma internal inductance (unnormalized)

### Appendix A.2. Algorithm Steps

The core idea of the GSPulse algorithm is to use a Picard iteration scheme for the Grad-Shafranov plasma flux distribution coupled with a time-dependent, global, dynamic optimization of the vacuum conductors. A diagram of the convergence strategy was given in fig. 1, illustrating the iterations between global optimization and plasma Picard updates of the flux distribution. After parsing the configuration inputs, the GSPulse algorithm performs the following actions:

1. **Initialization:** Estimate the initial plasma current distribution at each time step.
2. **Compute target boundary flux:** Integrate the Ejima equation to find the target boundary flux.
3. **Optimize conductor evolution:** Set up and solve a QP for the conductor evolution.
4. **Grad-Shafranov Picard iteration:** Update the plasma flux distribution
5. **Convergence:** Repeat from Step 2 until convergence metrics are satisfied.

These steps are described in detail below:

### Step 1: Initialization

The purpose of this step is to obtain a rough estimate of the plasma current distribution. We use the target shape boundary and find the geometric centroid. Then each point on the grid is written in terms of a scaled distance  $x$  with  $x=0$  at the centroid and  $x=1$  at the boundary. The current is estimated with parabolic distribution:

$$J_\phi = \hat{J}(1 - x^a)^b \quad (\text{A.7})$$

where  $a$  and  $b$  are constants and  $\hat{J}$  is a constant scaled to match the target  $I_p$ .

### Step 2: Calculate target boundary flux

The plasma internal inductance  $L_I$  can be measured from the current distribution. With initial condition  $\psi_{bry}(t = 0)$  obtained from the starting equilibrium, the Ejima eq. (A.5) can be integrated to find the  $\psi_{bry}(t)$  that provides the surface voltage to drive the target  $I_p$ .

### Step 3: Optimize conductor evolution

This step updates the coil currents and applied flux, and is done by casting the update into the form of a quadratic optimization problem similar to model predictive control.

The first step is casting the circuit dynamics equation into state space form. The circuit equation is:

$$v_i = R_i I_i + M_{ij} \dot{I}_j + \dot{\Phi}_i^{pla} \quad (\text{A.8})$$

We also have that the plasma-induced flux at any specific conductor is related through the grid mutuals and the plasma current distribution.

$$\dot{\Phi}_i^{pla} = M_{ig} \dot{I}_g \quad (\text{A.9})$$

where  $M_{ig}$  is the mutual inductance between conductor  $i$  and each grid location  $g$ , and  $I_g$  is the plasma current in grid cell  $g$ . (Note the connection to the Grad-Shafranov equation:  $I_g$  is the plasma current distribution corresponding to the current density  $J_\phi$ ).

The conductor evolution is written for both coil and vessel elements. Combining eqs. (A.8) and (A.9) and being explicit about coils or vessel elements, we arrive at:

$$\begin{bmatrix} V_c \\ 0 \end{bmatrix} = R_{cv} \begin{bmatrix} I_c \\ I_v \end{bmatrix} + M_{cv, cv} \begin{bmatrix} \dot{I}_c \\ \dot{I}_v \end{bmatrix} + M_{cv, g} \dot{I}_g \quad (\text{A.10})$$

Re-arranging this matrix equation gives us:

$$\dot{x} = Ax + Bu + w \quad (\text{A.11})$$

where we have used the following substitutions

$$\begin{aligned} x &= [I_c^T \ I_v^T]^T \\ u &= V_c \\ A &= -M_{cv, cv}^{-1} R_{cv} \\ B &= M_{cv, cv}^{-1} \begin{bmatrix} I \\ 0 \end{bmatrix} \\ w &= -M_{cv, cv}^{-1} M_{cv, g} \dot{I}_g \end{aligned} \quad (\text{A.12})$$

Note that  $w$  represents the influence of the plasma current on the conductors and can be calculated directly given the plasma current distribution at each time. Except for  $w$  which depends on the Grad-Shafranov solutions and is updated each iteration, the remaining dynamic terms are constant for a given machine (i.e.  $A$  and  $B$  do not change across GS iterations). This dynamics model is converted to discrete-time using the zero order hold method. For conciseness, we abuse the notation by using the same labels, making note that  $A$ ,  $B$ , and  $w$  from hereon refer to their discrete time versions.

$$x_{k+1} = Ax_k + Bu_k + w_k \quad (\text{A.13})$$

The shaping targets include parameters such as the flux at each of the control points, flux at target boundary location, and field at the target x-points. For any output  $y$  that is a linear function  $f$  of the grid flux distribution  $\psi_g$  (which is true for flux and field measurements) then the output can be represented:

$$\begin{aligned} y &= f(\psi_g) \\ &= f(\psi_g^{app}) + f(\psi_g^{pla}) \\ &= f(M_{g, cv} I_{cv}) + f(\psi_g^{pla}) \\ &:= f(M_{g, cv} I_{cv}) + y^{pla} \end{aligned} \quad (\text{A.14})$$

In other words, the output is separated into a part that depends on the coil and vessel currents and part that depends on the plasma flux distribution. Moreover, for flux and field measurements the function  $f(\cdot)$  is a linear mapping (e.g.  $f(x) = Cx$  where  $C$  is the greens functions for flux or field) so that we can write

$$y = CI_{cv} + y^{pla} \quad (\text{A.15})$$

or using the state-space notation

$$y_k = Cx_k + y_k^{pla} \quad (\text{A.16})$$

The dynamics eq. (A.13) and output eq. (A.16) form the basis of the prediction model for the optimization. In both equations we have intentionally separated the linear and nonlinear portions. The nonlinear terms  $w_k$  and  $y_k^{pla}$  are measured from the plasma current distribution and updated each iteration, while other terms can be computed once and re-used across iterations. Our cost function for the optimizer was eq. (A.2), repeated here:

$$J := \sum_{k=k_0}^N \|u_{k-1}\|_{W_u}^2 + \|\Delta u_{k-1}\|_{W_{\Delta}u}^2 + \|\Delta^2 u_{k-1}\|_{W_{\Delta}^2 u}^2 + \|e_k\|_{W_e}^2 + \|\Delta e_k\|_{W_{\Delta}e}^2 + \|\Delta^2 e_k\|_{W_{\Delta}^2 e}^2 \quad (\text{A.17})$$

Where  $u_k$  is the power supply voltage and  $e_k = r_k - y_k$  is the shape error between the reference target and actual, and the  $W_{(x)}$  are user-defined weights on the magnitude, derivative, and second derivative of the voltages and errors.

The next few steps map the cost function into a standard quadratic program which can be solved by many available software packages such as MatLab's **quadprog**. The next steps aim to transform the above cost function into the standard quadratic program form,

$$J = \hat{p}^T H \hat{p} + 2f^T \hat{p}, \quad (\text{A.18})$$

where  $\hat{p}$  is the set of primary optimization variables to be solved for, and for the GSPulse problem we use:

$$\hat{p} := \begin{bmatrix} x_1 \\ \hat{u} \end{bmatrix} \quad (\text{A.19})$$

To begin, we make the following definitions:

$$\begin{aligned}
\hat{u} &:= [u_0^T \ u_1^T \ \dots \ u_{N-1}^T]^T \\
\widehat{\Delta u} &:= [(u_1 - u_0)^T \ (u_2 - u_1)^T \ \dots \ (u_{N-1} - u_{N-2})^T]^T \\
\widehat{\Delta^2 u} &:= [(u_2 - 2u_1 + u_0)^T \ (u_3 - 2u_2 + u_1)^T \ \dots \ (u_{N-1} - 2u_{N-2} + u_{N-3})^T]^T \\
\hat{e} &:= [e_1^T \ e_2^T \ \dots \ e_N^T]^T \\
\widehat{\Delta e} &:= [(e_2 - e_1)^T \ (e_3 - e_2)^T \ \dots \ (e_N - e_{N-1})^T]^T \\
\widehat{\Delta^2 e} &:= [(e_3 - 2e_2 + e_1)^T \ (e_4 - 2e_3 + e_2)^T \ \dots \ (e_N - 2e_{N-1} + e_{N-2})^T]^T \\
\hat{w} &:= [w_0^T \ w_1^T \ \dots \ w_{N-1}^T]^T \\
\hat{x} &:= [x_1^T \ x_2^T \ \dots \ x_N^T]^T \\
\hat{y} &:= [y_1^T \ y_2^T \ \dots \ y_N^T]^T \\
\hat{r} &:= [r_1^T \ r_2^T \ \dots \ r_N^T]^T \\
\hat{C} &:= \text{blkdiag}(\underbrace{C, C, \dots, C}_{\times N})
\end{aligned} \tag{A.20}$$

$$\hat{W}_{(\alpha)} := \text{blkdiag}(\underbrace{W_{(\alpha)}, W_{(\alpha)}, \dots, W_{(\alpha)}}_{\times (N-k_0+1)}), \quad \alpha \in u, \Delta u, \Delta^2 u, e, \Delta e, \Delta^2 e$$

$k_0$  := summation starting index = 1 for absolute terms, 2 for first-order difference

( $\Delta$ ) terms, 3 for second-order difference ( $\Delta^2$ ) terms

Note that we can write matrix mappings between the  $\hat{u}$  and  $\Delta \hat{u}$ , since these are just linear combinations of each other.

$$\widehat{\Delta u} = \underbrace{\begin{bmatrix} -I & I \\ & \ddots \\ & & -I & I \end{bmatrix}}_{:= S_{\Delta u}} \hat{u} \tag{A.21}$$

Similarly for the 2nd derivatives, we have that:

$$\widehat{\Delta^2 u} = \underbrace{\begin{bmatrix} I & -2I & I \\ & \ddots & \\ & I & -2I & I \end{bmatrix}}_{:= S_{\Delta^2 u}} \hat{u} \tag{A.22}$$

And for equivalently-defined transition matrices for the errors, we have that:

$$\begin{aligned}
\widehat{\Delta e} &= S_{\Delta e} \hat{e} \\
\widehat{\Delta^2 e} &= S_{\Delta^2 e} \hat{e}
\end{aligned} \tag{A.23}$$

Thus the first 3 terms and last 3 terms in the cost function eq. (A.2) can be combined to rewrite the cost function as a matrix equation:

$$\begin{aligned}
J &= \hat{u}^T H_u \hat{u} + \hat{e}^T H_e \hat{e} \\
H_u &:= \hat{W}_u + S_{\Delta u}^T \hat{W}_{\Delta u} S_{\Delta u} + S_{\Delta^2 u}^T \hat{W}_{\Delta^2 u} S_{\Delta^2 u} \\
H_e &:= \hat{W}_e + S_{\Delta e}^T \hat{W}_{\Delta e} S_{\Delta e} + S_{\Delta^2 e}^T \hat{W}_{\Delta^2 e} S_{\Delta^2 e}
\end{aligned} \tag{A.24}$$

Now, lets turn our attention to the dynamics. Using a technique from Model Predictive Control, we can form a prediction model of the dynamic evolution. From the state-space dynamics equation we have that:

$$\begin{aligned}
x_1 &= x_1 \\
x_2 &= Ax_1 + Bu_1 + w_1 \\
x_3 &= Ax_2 + Bu_2 + w_2 \\
&= A(x_1 + Bu_1 + w_1) + Bu_2 + w_2 \\
&= A^2 x_1 + [AB \quad B] \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} + [A \quad I] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
x_4 &= \dots
\end{aligned} \tag{A.25}$$

Extending this to all times gives:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \underbrace{\begin{bmatrix} I \\ A \\ \vdots \\ A^N \end{bmatrix}}_E x_1 + \underbrace{\begin{bmatrix} 0 & & & & \\ 0 & B & & & \\ 0 & AB & B & & \\ \vdots & \vdots & \ddots & & \\ 0 & A^{N-1}B & A^{N-2}B & \dots & B \end{bmatrix}}_F \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{u_{N-1}} + \underbrace{\begin{bmatrix} 0 & & & & \\ 0 & I & & & \\ 0 & A & I & & \\ \vdots & \vdots & \ddots & & \\ 0 & A^{N-1} & A^{N-2} & \dots & I \end{bmatrix}}_{F_w} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{N-1} \end{bmatrix}}_{w_{N-1}}$$

Or equivalently,

$$\hat{x} = Ex_1 + F\hat{u} + F_w\hat{w} \tag{A.27}$$

Then using eq. (A.16) the predicted errors are:

$$\begin{aligned}
\hat{e} &= \hat{r} - \hat{y} \\
&= \hat{r} - \hat{y}^{pla} - \hat{C}\hat{x} \\
&= \hat{r} - \hat{y}^{pla} - \hat{C}(Ex_1 + F\hat{u} + F_w\hat{w}) \\
&:= M\hat{p} + d
\end{aligned} \tag{A.28}$$

where to obtain the last line we have defined new variables,

$$\begin{aligned}
M &:= -\hat{C} [E \quad F] \\
\hat{p} &:= \begin{bmatrix} x_1 \\ \hat{u} \end{bmatrix} \\
d &:= \hat{r} - \hat{y}^{pla} - \hat{C} F_w \hat{w}
\end{aligned} \tag{A.29}$$

Note that  $\hat{p}$  consists of the initial starting currents  $x_1$  and the voltages at all times, and is the set primary optimization variables to be solved for by the quadratic program eq. (A.18). That is, the output of the QP solve is  $\hat{p}$ . The elements of  $x_1$  can be specified either as constrained parameters, for the use case where an initial condition is specified, or they can be specified as free parameters for the optimizer to solve for.

The relation between the voltages and the primary optimization variables  $\hat{p}$  is:

$$\hat{u} = \underbrace{\begin{bmatrix} 0 \\ I \end{bmatrix}}_{:= T_{up}} \hat{p} \tag{A.30}$$

Then, plugging in eqs. (A.28) and (A.30) into eq. (A.24), we arrive finally at the standard form of the quadratic program:

$$\begin{aligned}
J &= \hat{u}^T H_u \hat{u} + \hat{e}^T H_e \hat{e} \\
&= \hat{p}^T T_{up}^T H_u T_{up} \hat{p} + (M \hat{p} + d)^T H_e (M \hat{p} + d) \\
&= \hat{p}^T H \hat{p} + 2f^T \hat{p} \\
H &:= T_{up}^T H_u T_{up} + M^T H_e M \\
f &:= M^T H_e^T d
\end{aligned} \tag{A.31}$$

Solving the quadratic program eq. (A.31) can be done with standard QP solver packages such as **quadprog** in MATLAB. The solution to the QP is the initial starting currents as well as the trajectory of voltages. The voltage trajectory can be mapped back to the trajectory of currents and errors, via eq. (A.27) and eq. (A.28). Constraints on the voltages, currents, and outputs can be added to the QP problem via the mappings presented in this derivation. In the standard QP form, constraints enter in the form of  $A\hat{p} < b$ , which is a linear mapping of the primary optimization variables. The following equations describe how to put various types of constraints in this form.

For constraints on the voltage ( $\hat{u}$ ), which would include restrictions such as power supply voltage limits or slewrate limits, we use eq. (A.30) to write:

$$\begin{aligned}
A_u \hat{u} &< b_u \\
\implies A_u T_{up} \hat{p} &< b_u
\end{aligned} \tag{A.32}$$

Similarly for constraints on the coil currents ( $\hat{x}$ ) we use eq. (A.27) to write:

$$\begin{aligned} A_x \hat{x} &< b_x \\ \implies A_x(Ex_1 + F\hat{u} + F_w\hat{w}) &< b_x \\ \implies A_x([E\ F]\hat{p} + F_w\hat{w}) &< b_x \\ \implies A_x[E\ F]\hat{p} &< b_x - A_xF_w\hat{w} \end{aligned} \tag{A.33}$$

And for constraints on the output errors ( $\hat{e}$ ) we use eq. (A.28) to write:

$$\begin{aligned} A_e \hat{e} &< b_e \\ \implies A_e(M\hat{p} + d) &< b_e \\ \implies A_eM\hat{p} &< b_e - A_e d \end{aligned} \tag{A.34}$$

Lastly, we conclude with a few observations about this QP.

- Note that, if only 1 time step is being solved for ( $N = 1$ ), then the dynamical prediction model shrinks to identity and the QP only solves for the  $x_1$  set of currents. In other words, it behaves exactly as a static inverse Picard-based equilibrium solver updating a single set of currents at each iteration. This means that, in addition to being a dynamic equilibrium trajectory solver, GSPulse can also function as a standard inverse equilibrium solver, since the static problem is a sub-problem of the dynamic one.
- Assuming that the state-space output matrix  $\hat{C}$  is linear, then  $H$  stays constant across iterations and only  $d$  (eq. (A.29)) needs to be updated. This means that many of the matrix objects can be pre-computed only one time and re-used, saving on computational time since a large fraction of the compute is absorbed by forming and multiplying the large matrices. Therefore it is very advantageous to ensure that the output matrix  $\hat{C}$  is linear and does not need to be updated, which is an advantage for specifying outputs in terms of flux units instead of spatial unit. For outputs that have units of B-field, flux, or current, then  $\hat{C}$  is linear.

#### Step 4: Grad-Shafranov Picard iteration

In this step, the goal is to perform a Picard update the total flux distribution according to the Grad-Shafranov equation in order to converge the solution numerically. The full Picard iteration process is given as follows.

In the QP optimizer step, we computed the trajectory currents in the external conductors. (Actually, the QP solver obtained the initial currents and trajectory of voltages, but these are mapped to the conductor currents via the prediction model eq. (A.27).) This allows us to update the vacuum flux distribution via:

$$\psi_{vac}^{k+1} = M_{g,cv}I_{cv}^{k+1}, \tag{A.35}$$

where  $k$  is the iteration index. Next, we do a partial update of the total flux distribution, taking the partially-updated flux as a sum of the previous plasma flux contribution and the updated vacuum flux.

$$\psi^{k+1/2} := \psi_{pla}^k + \psi_{vac}^{k+1} \tag{A.36}$$

The remaining steps are to update the  $P'$  and  $FF'$  profiles and total flux of the Grad-Shafranov equation:

$$\begin{aligned} J_{\phi,pla}^{k+1} &= RP'(\psi^{k+1/2}) + \frac{FF'(\psi^{k+1/2})}{\mu_0 R} \\ \Delta^* \psi^{k+1} &= -\mu_0 R \left( J_{\phi,pla}^{k+1} + J_{\phi,vac}^{k+1} \right) \end{aligned} \quad (\text{A.37})$$

The present implementation of GSPulse allows for two different methods of constraining the  $P'$  and  $FF'$  profiles used in the Picard update. The first method is that GSPulse has been integrated with the FBT code developed by EPFL that is part of the open-source MEQ toolbox. FBT contains a number of flexible and fast methods for constraining the  $P'$  and  $FF'$  profiles, including various basis function types for the profile shapes, and multiple options to specify combinations of core plasma properties such as the total plasma current  $I_p$ , total stored thermal energy  $W_{th}$ , plasma beta  $\beta_p$ , or q-profile value on-axis  $q_A$ .

The second method is a built-in GSPulse option that does not require installation of the FBT/MEQ codes, but has slightly less flexibility in specification of the core plasma properties. With this method, the user must specify input basis function shapes for the  $P'$  and  $FF'$  profiles, as well as input waveforms for the plasma current  $I_p$  and thermal energy  $W_{th}$ . At each iteration, GSPulse scales the coefficients multiplying the basis function profiles in order to match the target  $I_p$  and  $W_{th}$ .

To describe the built-in method with more detail:

After using the optimizer to obtain the updated conductor currents, we compute  $\psi_{k+1/2}$  from eq. (A.36) which is the partially-updated flux on the grid. Then we use touch-point finder, x-point finder, and o-point finder algorithms on this flux distribution. Out of the identified potential touch-points, x-points, and o-points, we select the magnetic axis and boundary-defining point of the flux distribution (where the boundary-defining point is either a touch-point or an x-point). The algorithm to identify which points are the magnetic axis and boundary follows the logic presented in [18] section 2.3, which is an efficient method for identifying the boundary. The output of all these steps is to identify the flux at the magnetic axis  $\psi_{mag}$  and the flux at the boundary  $\psi_{bry}$  as well as to identify the physical boundary of the plasma.

Since the user has provided input profile shapes on a normalized  $\psi$  basis, our job is to appropriately scale the coefficients in order to match the target  $I_p$  and  $W_{th}$ . Let the input basis shapes be  $P'_b(\psi_N)$  and  $FF'_b(\psi_N)$  where  $\psi_N$  is the normalized flux:

$$\psi_N := \frac{\psi - \psi_{bry}}{\psi_{mag} - \psi_{bry}} \quad (\text{A.38})$$

The total profiles are then:

$$\begin{aligned} P'(\psi) &= c_p P'_b(\psi_N) / (\psi_{mag} - \psi_{bry}) \\ FF'(\psi) &= c_f FF'_b(\psi_N) / (\psi_{mag} - \psi_{bry}) \end{aligned} \quad (\text{A.39})$$

The scaling coefficients  $c_p$  and  $c_f$  are terms to be solved for, and the last term arises from de-normalizing the  $\psi$  basis, since for any parameter  $x$ :

$$\begin{aligned}\frac{dx}{d\psi} &= \frac{dx}{d\psi_N} \frac{d\psi}{d\psi_N} \\ &= \frac{dx}{d\psi_N} \frac{1}{(\psi_{mag} - \psi_{bry})}\end{aligned}\tag{A.40}$$

The first constraint is on the total plasma current,

$$I_p = \int_{\Omega_p} J_{\phi,pla} dA\tag{A.41}$$

Substituting in the Grad-Shafranov eq. (A.3), we obtain that

$$I_p = \frac{1}{\psi_{mag} - \psi_{bry}} \left[ c_p \int_{\Omega_p} RP'_b dA + c_f \int_{\Omega_p} \frac{FF'_b}{\mu_0 R} dA \right]\tag{A.42}$$

This equation depends linearly on the profile coefficients, while the integrals are computed numerically at each iteration. The second constraint is on the stored thermal energy  $W_{th}$ , where we have that:

$$\begin{aligned}W_{th} &= \int_{\Omega_p} \frac{3}{2} P(\psi_N) dV \\ &= \int_{\Omega_p} 3\pi RP(\psi_N) dA,\end{aligned}\tag{A.43}$$

where the pressure is found from integrating the  $P'$  basis function from the edge of the plasma inwards (assuming that the pressure is zero at the edge).

$$P(\psi_N) = c_p \int_{\hat{\psi}_N=1}^{\hat{\psi}_N=\psi_N} P'_b d\hat{\psi}_N\tag{A.44}$$

The linear system of 2 equations (eqs. (A.42) and (A.44)) and 2 unknowns can be solved to obtain the profile coefficients. This gives us the necessary total  $P'$  and  $FF'$  profiles, which are used to obtain the total flux distribution according to eq. (A.37).

At this stage we have now updated the plasma current and total flux distributions, and the solver can proceed to the next iteration.

### Appendix A.3. Spline basis compression:

The computational complexity of obtaining an optimal solution to the QP (eq. (A.31)) can vary dramatically depending on the number of equilibria that are solved for and the complexity of the evolution. Solving this type of optimization problem can have poor numerical scaling, since in general the number of optimization variables that is solved for is  $n = (N_{equilibria}+2)*N_{coils}$ , and the theoretical time complexity for solving linear-constrained quadratic programs is  $O(n^3)$ . In practice the run-time performance also depends on the complexity of the equilibrium evolution. Observationally, highly dynamic events such as strike point sweeping take more time to solve. Therefore it is advantageous to employ methods that reduce the computational complexity of the QP perhaps at the expense of sacrificing some of the optimality of the solution.

One technique for data compression is to use a spline basis for the primary optimization variables. If we have a spline basis of the form,

$$\hat{p} = S_m \hat{c} \quad (\text{A.45})$$

where  $\hat{p}$  is the original primary variables,  $S_m$  is a splining matrix, and  $\hat{c}$  are the spline basis coefficients. Then, instead of solving for the original variables we optimize for the spline coefficients. The quadratic program is then transformed as:

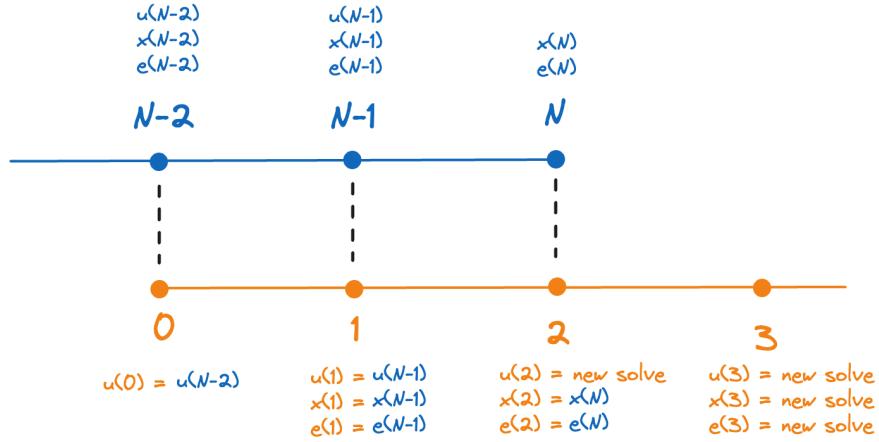
$$\begin{aligned} \min J &= \hat{p}^T H \hat{c} + 2f^T \hat{p} \\ &= \hat{c}^T S_m^T H S_m \hat{c} + 2f^T S_m \hat{c} \\ \text{subj to: } & \\ A\hat{p} &\leq b \\ \implies AS_m\hat{c} &\leq b \end{aligned} \quad (\text{A.46})$$

After solving for the spline coefficients, the original variables are constructed from the spline basis above eq. (A.45).

#### *Appendix A.4. Optimization with multiple time intervals:*

Another important technique for reducing the time complexity of the QP-solve is to solve for the evolution in multiple stages. For example, instead of solving for 200 equilibria at a time, we may solve for 50 equilibria at a time across 4 overlapping stages reducing the complexity from  $O(200^3)$  to  $4 \times O(50^3)$ . This reduces the optimality of the solve since the first stage is not looking far into the future. However, in many cases this optimality reduction is not significant, i.e. details of the rampdown trajectory do not significantly affect the details of the rampup trajectory.

GSPulse enables this multiple-interval approach by overlapping the equilibria from one stage to the next. The 1<sup>st</sup> and 2<sup>nd</sup> equilibria are automatically enforced to be the  $(N - 1)^{\text{th}}$  and  $N^{\text{th}}$  equilibria from the previous stage. These equality constraints enforce continuity of the trajectory, and also give smooth trajectories provided that the user has weighting penalties on the second derivative (non-smooth trajectories).



**Figure A.11:** Overlapping strategy when using GSPulse to optimize across multiple intervals. The previous stage is shown in blue and subsequent stage in orange. The stages are overlapped by 2 equilibria (2 time points each for the voltages  $u$ , currents  $x$ , and shaping errors  $e$ ).

The multiple-interval strategy is shown in fig. A.11. As shown in the figure, the subsequent stage needs to enforce equality constraints on  $u_0, u_1, x_1, e_1, x_2$  and  $e_2$ .

- The terms  $x_1, u_0$  and  $u_1$  make up the first three terms of the optimization variables  $\hat{p}$  (see eq. (A.29)). Therefore, these terms can be enforced by equality constraint on the QP solver.
- The equality constraint on  $x_2$  is enforced automatically and implicitly through the other terms, since  $x_2 = Ax_1 + Bu_1 + w_1$  and all terms on the RHS are specified.
- The remaining terms are  $e_1$  and  $e_2$ . Our prediction model for the errors was eq. (A.28) and repeated here:

$$\hat{e} := \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_N \end{bmatrix} = M\hat{p} + d \quad (\text{A.47})$$

We achieve equality constraint by replacing the rows of  $M$  and elements of  $d$  corresponding to  $e_1$  and  $e_2$ :

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_N \end{bmatrix} = \underbrace{\begin{bmatrix} & 0 \\ & 0 \\ \hline M_{((2 \times n_e + 1):N,:)} \end{bmatrix}}_M \hat{p} + \underbrace{\begin{bmatrix} e_1 \\ e_2 \\ \hline d_{(2 \times n_e + 1):N} \end{bmatrix}}_{\bar{d}} \quad (\text{A.48})$$

Then, eq. (A.48) is used in place of eq. (A.47) when constructing the QP, for any stage after the first.

## Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Fusion Energy Sciences, under Awards DE-AC02-09CH11466, DE-SC0015480, DE-SC0021275, and DE-SC0015878.

This work has been part-funded by the EPSRC Energy Programme [grant number EP/W006839/1]. To obtain further information on the data and models underlying this paper (in Section 3.2) please contact PublicationsManager@ukaea.uk.

This work was supported by Commonwealth Fusion Systems.

## Disclaimer

This report is prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

## References

- [1] J. T. Wai, M. D. Boyer, E. Kolemen, Neural net modeling of equilibria in NSTX-U, Nuclear Fusion 62 (2022) 086042. URL: <https://doi.org/10.1088/1741-4326/ac77e6>. doi:[10.1088/1741-4326/ac77e6](https://doi.org/10.1088/1741-4326/ac77e6).
- [2] R. Khayrutdinov, V. Lukash, Studies of plasma equilibrium and transport in a tokamak fusion device with the inverse-variable technique, Journal of Computational Physics 109 (1993) 193–201. URL: <http://dx.doi.org/10.1006/jcph.1993.1211>. doi:[10.1006/jcph.1993.1211](http://dx.doi.org/10.1006/jcph.1993.1211).
- [3] F. Carpanese, Development of free-boundary equilibrium and transport solvers for simulation and real-time interpretation of tokamak experiments, Ph.D. thesis, 2021. URL: <https://infoscience.epfl.ch/handle/20.500.14299/175817>. doi:[10.5075/EPFL-THESIS-7914](https://infoscience.epfl.ch/handle/20.500.14299/175817).
- [4] N. C. Amorisco, et al., Freegsnke: A python-based dynamic free-boundary toroidal plasma equilibrium solver, Physics of Plasmas 31 (2024). URL: <http://dx.doi.org/10.1063/5.0188467>. doi:[10.1063/5.0188467](http://dx.doi.org/10.1063/5.0188467).
- [5] A. Welander, et al., Closed-loop simulation with grad-shafranov equilibrium evolution for plasma control system development, Fusion Engineering and Design 146 (2019) 2361–2365. doi:[10.1016/j.fusengdes.2019.03.191](https://doi.org/10.1016/j.fusengdes.2019.03.191).
- [6] H. Heumann, et al., Quasi-static free-boundary equilibrium of toroidal plasma with cedres++: Computational methods and applications, Journal of Plasma Physics 81 (2015). URL: <http://dx.doi.org/10.1017/S0022377814001251>. doi:[10.1017/S0022377814001251](http://dx.doi.org/10.1017/S0022377814001251).
- [7] J. Blum, et al., Automating the design of tokamak experiment scenarios, Journal of Computational Physics 394 (2019) 594–614. URL: <http://dx.doi.org/10.1016/j.jcp.2019.05.046>. doi:[10.1016/j.jcp.2019.05.046](http://dx.doi.org/10.1016/j.jcp.2019.05.046).

- [8] B. Faugeras, An overview of the numerical methods for tokamak plasma equilibrium computation implemented in the nice code, *Fusion Engineering and Design* 160 (2020) 112020. URL: <http://dx.doi.org/10.1016/j.fusengdes.2020.112020>. doi:[10.1016/j.fusengdes.2020.112020](https://doi.org/10.1016/j.fusengdes.2020.112020).
- [9] C. Hansen, et al., Tokamaker: An open-source time-dependent grad-shafranov tool for the design and modeling of axisymmetric fusion devices, *Computer Physics Communications* 298 (2024) 109111. URL: <http://dx.doi.org/10.1016/j.cpc.2024.109111>. doi:[10.1016/j.cpc.2024.109111](https://doi.org/10.1016/j.cpc.2024.109111).
- [10] F. Hofmann, FBT - a free-boundary tokamak equilibrium code for highly elongated and shaped plasmas, *Computer Physics Communications* 48 (1988) 207–221. URL: [http://dx.doi.org/10.1016/0010-4655\(88\)90041-0](http://dx.doi.org/10.1016/0010-4655(88)90041-0). doi:[10.1016/0010-4655\(88\)90041-0](https://doi.org/10.1016/0010-4655(88)90041-0).
- [11] O. Sauter, S. Medvedev, Tokamak coordinate conventions:, *Computer Physics Communications* 184 (2013) 293–302. URL: <http://dx.doi.org/10.1016/j.cpc.2012.09.010>. doi:[10.1016/j.cpc.2012.09.010](https://doi.org/10.1016/j.cpc.2012.09.010).
- [12] H. Grad, H. Rubin, Hydromagnetic equilibria and force-free fields, *Proceedings of the 2nd UN Conf. on the Peaceful Uses of Atomic Energy* 31 (1958) 190.
- [13] V. Shafranov, Plasma equilibrium in a magnetic field, *Reviews of Plasma Physics* (1966) 103.
- [14] S. Ejima, et al., Volt-second analysis and consumption in doublet III plasmas, *Nuclear Fusion* 22 (1982) 1313–1319. URL: <https://doi.org/10.1088/0029-5515/22/10/006>. doi:[10.1088/0029-5515/22/10/006](https://doi.org/10.1088/0029-5515/22/10/006).
- [15] J. Romero, Plasma internal inductance dynamics in a tokamak, *Nuclear Fusion* 50 (2010) 115002. URL: <https://doi.org/10.1088/0029-5515/50/11/115002>. doi:[10.1088/0029-5515/50/11/115002](https://doi.org/10.1088/0029-5515/50/11/115002).
- [16] D. Humphreys, et al., Development of iter-relevant plasma control solutions at DIII-D, *Nuclear Fusion* 47 (2007) 943–951. URL: <http://dx.doi.org/10.1088/0029-5515/47/8/028>. doi:[10.1088/0029-5515/47/8/028](https://doi.org/10.1088/0029-5515/47/8/028).
- [17] MEQ Suite, <https://www.epfl.ch/research/domains/swiss-plasma-center/meq/>, 2025. [Accessed 22-05-2025].
- [18] J.-M. Moret, et al., Tokamak equilibrium reconstruction code liuqe and its real time implementation, *Fusion Engineering and Design* 91 (2015) 1–15. URL: <http://dx.doi.org/10.1016/j.fusengdes.2014.09.019>. doi:[10.1016/j.fusengdes.2014.09.019](https://doi.org/10.1016/j.fusengdes.2014.09.019).
- [19] J. Johnson, et al., Numerical determination of axisymmetric toroidal magnetohydrodynamic equilibria, *Journal of Computational Physics* 32 (1979) 212–234. URL: [http://dx.doi.org/10.1016/0021-9991\(79\)90129-3](http://dx.doi.org/10.1016/0021-9991(79)90129-3). doi:[10.1016/0021-9991\(79\)90129-3](https://doi.org/10.1016/0021-9991(79)90129-3).
- [20] J. Ferron, et al., Real time equilibrium reconstruction for tokamak discharge control, *Nuclear Fusion* 38 (1998) 1055–1066. URL: <http://dx.doi.org/10.1088/0029-5515/38/7/308>. doi:[10.1088/0029-5515/38/7/308](https://doi.org/10.1088/0029-5515/38/7/308).
- [21] J. Berkery, et al., NSTX-U research advancing the physics of spherical tokamaks, *Nuclear Fusion* 64 (2024) 112004. URL: <http://dx.doi.org/10.1088/1741-4326/ad3092>. doi:[10.1088/1741-4326/ad3092](https://doi.org/10.1088/1741-4326/ad3092).
- [22] J. Harrison, et al., Overview of new MAST physics in anticipation of first results from MAST Upgrade, *Nuclear Fusion* 59 (2019) 112011.
- [23] M. Boyer, et al., Plasma boundary shape control and real-time equilibrium reconstruction on NSTX-U, *Nuclear Fusion* 58 (2018) 036016. doi:[10.1088/1741-4326/aaa4d0](https://doi.org/10.1088/1741-4326/aaa4d0).

- [24] S. Sabbagh, et al., Equilibrium properties of spherical torus plasmas in NSTX, Nuclear Fusion 41 (2001) 1601–1611. doi:[10.1088/0029-5515/41/11/309](https://doi.org/10.1088/0029-5515/41/11/309).
- [25] L. Lao, et al., Reconstruction of current profile parameters and plasma shapes in tokamaks, Nuclear Fusion 25 (1985) 1611–1622. doi:[10.1088/0029-5515/25/11/007](https://doi.org/10.1088/0029-5515/25/11/007).
- [26] J. Menard, et al., Overview of the physics and engineering design of NSTX upgrade, Nuclear Fusion 52 (2012) 083015. URL: <http://dx.doi.org/10.1088/0029-5515/52/8/083015>. doi:[10.1088/0029-5515/52/8/083015](https://doi.org/10.1088/0029-5515/52/8/083015).
- [27] A. J. Creely, et al., Overview of the sparc tokamak, Journal of Plasma Physics 86 (2020). URL: <http://dx.doi.org/10.1017/S0022377820001257>. doi:[10.1017/s0022377820001257](https://doi.org/10.1017/s0022377820001257).
- [28] J. Degrave, et al., Magnetic control of tokamak plasmas through deep reinforcement learning, Nature 602 (2022) 414–419. URL: <http://dx.doi.org/10.1038/s41586-021-04301-9>. doi:[10.1038/s41586-021-04301-9](https://doi.org/10.1038/s41586-021-04301-9).
- [29] L. E. di Grazia, et al., Automated shot-to-shot optimization of the plasma start-up scenario in the tcv tokamak, Nuclear Fusion 64 (2024) 096032. URL: <http://dx.doi.org/10.1088/1741-4326/ad67ee>. doi:[10.1088/1741-4326/ad67ee](https://doi.org/10.1088/1741-4326/ad67ee).
- [30] B. Lloyd, et al., Low voltage ohmic and electron cyclotron heating assisted startup in DIII-D, Nuclear Fusion 31 (1991) 2031–2053. URL: <http://dx.doi.org/10.1088/0029-5515/31/11/001>. doi:[10.1088/0029-5515/31/11/001](https://doi.org/10.1088/0029-5515/31/11/001).
- [31] J. Leuer, et al., Plasma startup design of fully superconducting tokamaks EAST and KSTAR with implications for ITER, IEEE Transactions on Plasma Science 38 (2010) 333–340. URL: <http://dx.doi.org/10.1109/TPS.2009.2037890>. doi:[10.1109/tps.2009.2037890](https://doi.org/10.1109/tps.2009.2037890).
- [32] J. T. Wai, Shape and divertor control in tokamaks, Ph.D. thesis, Princeton University, 2023. URL: <http://arks.princeton.edu/ark:/88435/dsp01hq37vr85z>.
- [33] J. T. Wai, et al., Startup scenario development for SPARC, APS-DPP (2022). URL: <https://meetings.aps.org/Meeting/DPP22/Session/UP11.20>.