NATIONAL INSTITUTE OF AEROSPACE

Multi-Physics Analysis and External Code Compatibility

Dr. Heather Kline

National Institute of Aerospace

Agenda

- Previous work with SU2: Adjoints & Multi-Objective Optimization
- Multi-Physics Analysis Example: Boundary Layer Stability Analysis
  - Overview of boundary layer stability methods and contrast to empirical transition prediction.
  - Coupling e^N and CFD for Design.
- Coupling with external tools
  - Motivations for linking to external (to SU2) tools and potential challenges.
  - Dynamic linked libraries with run-time binding as a convenient solution.

Agenda

- **Previous work with SU2: Adjoints & Multi-Objective Optimization**
- Multi-Physics Analysis Example: Boundary Layer Stability Analysis
  - Overview of boundary layer stability methods and contrast to empirical transition prediction.
  - Coupling e^N and CFD for Design.
- Coupling with external tools
  - Motivations for linking to external (to SU2) tools and potential challenges.
  - Dynamic linked libraries with run-time binding as a convenient solution.

Previous work with SU2: Adjoints & Multi-Objective Optimization

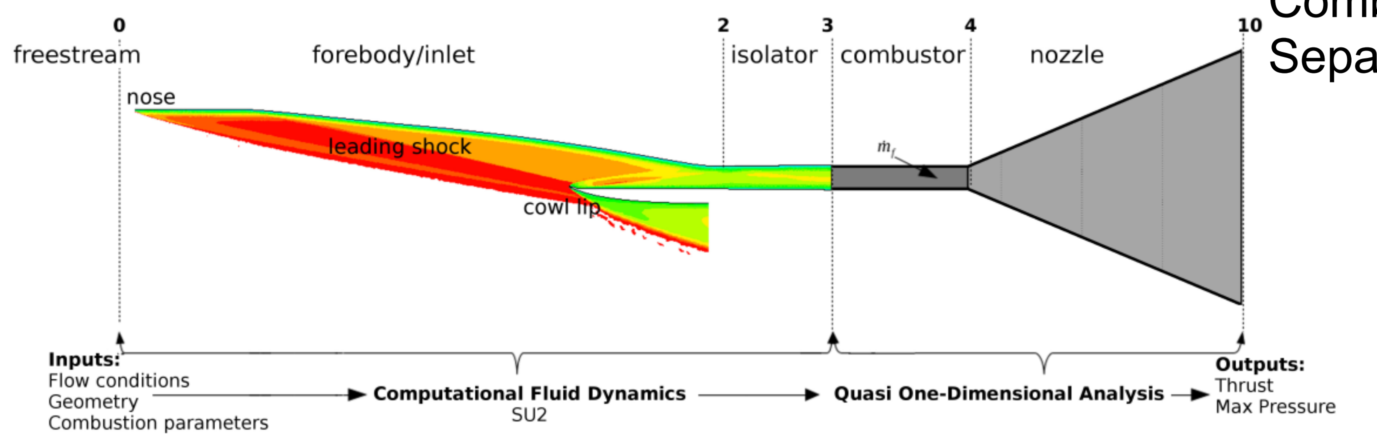
NATIONAL INSTITUTE OF AEROSPACE

- Multi-fidelity model with gradient information passed across isolator outlet boundary in order to compute continuous adjoint for functions dependent on the entire flowpath.
- Fluid-Structure Interaction
- Multi-objective continuous and discrete adjoints, reducing gradient cost by combining multiple functionals into a single adjoint evaluation.
  - See multi-objective shape design tutorial

$$\delta\left(J_1 + f(J_2)\right) = \delta J_1 + \frac{\partial f}{\partial J_2}\delta J_2.$$

Var.	$\frac{\partial\left(C_D \times 10^5 + \bar{P}_t \times 10^{-5}\right)}{\partial x_i}$ (simultaneous)	$\frac{\partial C_D}{\partial x_i} \times 10^5 + \frac{\bar{P}_t}{\partial x_i} \times 10^{-5}$ (separate)
0	-1.50232998E+2	-1.50232998E+2
1	-9.40390906E+1	-9.40390906E+1
2	-4.40948556E+1	-4.40948556E+1
3	-1.58777572E+1	-1.58777572E+1
4	-4.30593276E+0	-4.30593276E+0
5	-7.47768208E-1	-7.47768208E-1
6	8.61790879E-2	8.61790874E-2

Time to complete 2 optimizer iterations:
Combined (1 adjoint/step): 3m 10s
Separate (2 adjoint/step): 5m 35s

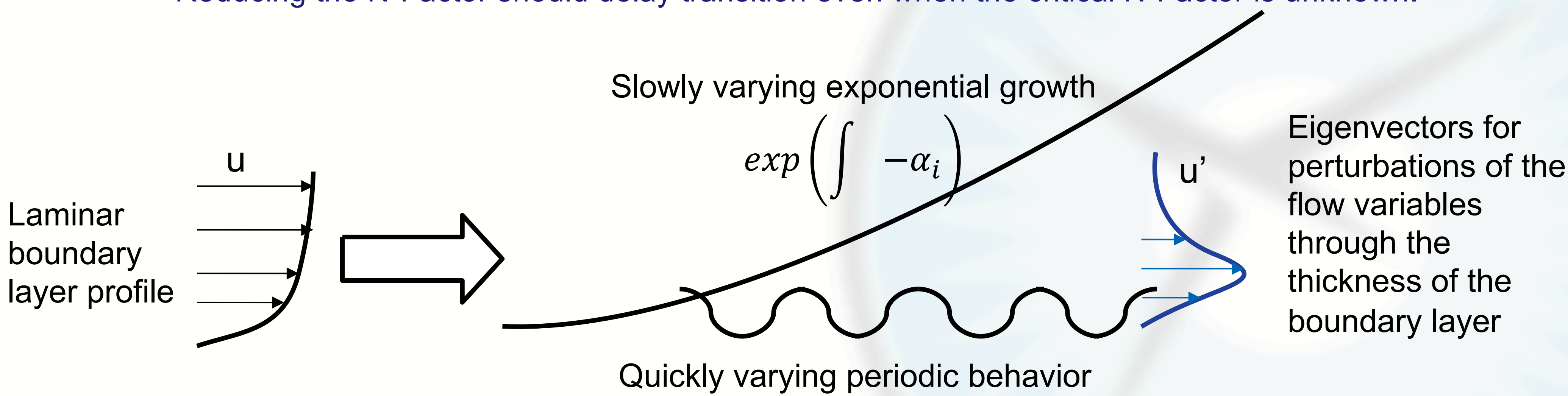


Agenda

- Previous work with SU2: Adjoints & Multi-Objective Optimization
- **Multi-Physics Analysis Example: Boundary Layer Stability Analysis**
  - Overview of boundary layer stability methods and contrast to empirical transition prediction.
  - Coupling e^N and CFD for Design.
- Coupling with external tools
  - Motivations for linking to external (to SU2) tools and potential challenges.
  - Dynamic linked libraries with run-time binding as a convenient solution.

Overview of boundary layer stability methods and contrast to empirical transition prediction.

- e^N methods predict perturbations of flow quantities resulting from a disturbance of a known frequency using linearizations of the Navier-Stokes equations and an assumed Fourrier series solution.
  - LST: Linear/Local Stability Theory – assumes parallel flow and neglects nonlinear effects.
  - PSE: Parabolized Stability Equations – includes nonparallel effects, neglects nonlinear effects.
  - NPSE: Nonlinear Parabolized Stability Equations – requires amplitude of initial disturbance.
- 'Semi-empirical' due to need for a critical amplification/N-Factor.
  - Critical N-Factor 4-6 common for wind tunnels, 8-15 for flight experiments.
  - Reducing the N-Factor should delay transition even when the critical N-Factor is unknown.



Empirical Transition vs. Boundary Layer Stability

NATIONAL INSTITUTE OF AEROSPACE

- **Empirical Methods: Langtry-Menter, critical Reynolds numbers, Reynolds/Mach ratio**
- Low computational cost.
- Straightforward implementation
- Valid only within the experimental data used to produce the empirical fits.
- No information about transition mechanisms.

- **Boundary Layer Stability Analysis: e^N methods: LST, PSE, NPSE**
- Physics-based, applicable to a wider range of conditions and provides information about transition mechanism.
- Higher computational cost; both in the analysis itself and requirements of the mean flow solutions.
- More information required in coupling – and more available in results.
- Some methods require, and all benefit from, knowledge of the disturbance environment.

Coupling e^N and CFD for Design.

NATIONAL INSTITUTE OF AEROSPACE

- **Proposed project** (led by Pedro Paredes at NIA): implement LST/PSE within SU2 for design optimization, using algorithmic differentiation to produce the discrete adjoint for use with natural laminar flow design.
  - Linear stability based methods are routinely employed for transition analysis, but has not yet been included in an adjoint-based optimization framework.
- **Potential Challenges**
  - Entire boundary layer profiles needed, which must be along normals to the surface.
  - Laminar flow solutions required for input to the LST.
  - Interpolation likely to be required for transferring information to boundary layers.
  - SU2 already has non-matching mesh tools that may be exploited for this purpose.
  - Extrapolation of transition location on the surface to the volume, including intermittency, may be needed.

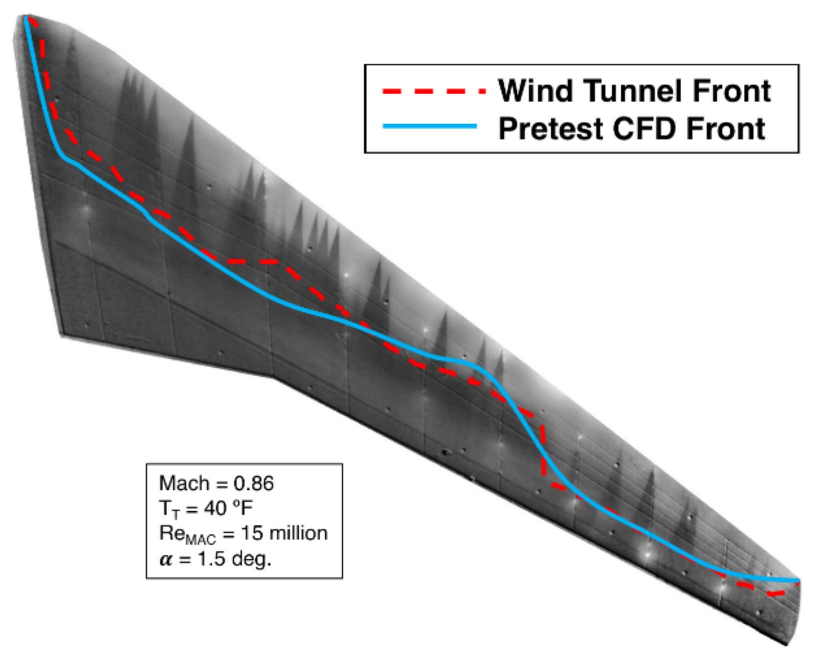


Figure 32. TSP image with the approximated transition front shown (red dashed line) from test condition M = 0.86, T_T = 40 °F, α = 1.5 deg., and Re_{MAC} = 15.0 million. The pretest CFD transition predictions at corresponding CFD conditions for a critical N-factor of 6 is overlaid on the image.

Lynde, Michelle N., et al. "Preliminary Results from an Experimental Assessment of a Natural Laminar Flow Design Method." AIAA Scitech 2019 Forum. 2019.

Agenda

NATIONAL INSTITUTE OF AEROSPACE

- Previous work with SU2: Adjoints & Multi-Objective Optimization
- Multi-Physics Analysis Example: Boundary Layer Stability Analysis
  - Overview of boundary layer stability methods and contrast to empirical transition prediction.
  - Coupling e^N and CFD for Design.
- **Coupling with external tools**
  - Motivations for linking to external (to SU2) tools and potential challenges.
  - Dynamic linked/shared libraries with run-time binding as a convenient solution.

Motivations for linking to external (to SU2) tools and potential challenges.

- Motivations:
  - Take advantage of fully-developed features in external codes
  - Avoid licensing issues
  - Flexibility to use multiple options
  - Reduced maintenance requirements
- Challenges:
  - Designing an efficient, flexible, and easy-to-use Application Programming Interface (API)
  - No control over the content of the external code
- Possible Solutions:
  - File I/O
  - Python wrapper
  - Compile-time linking to shared libraries
  - Run-time linking to shared libraries

Dynamic linked/shared libraries with run-time binding as a convenient solution.

- No recompilation required:
  - User specifies the path to their compiled library following API defined within SU2 – through an configuration file parameter or through an environment variable.
  - SU2 tests whether the library exists, then binds a function call to that library.
  - SU2 routines call the function as though it was internal.
  - Output errors if the path is incorrect or if the API does not match.
  - Can exist alongside with built-in functions.
- Well-thought out API necessary
- Computationally efficient – more expensive than built-in functions, less expensive than python wrapping
- Facilitates code-to-code comparison, collaborative/concurrent development
- Regression tests for the API only – minimal additional SU2 maintenance cost
- No SU2 changes required when external code is updated: the same executable can be used with multiple libraries
- Convenient for development of new features

Dynamic Linked/Shared Libraries

- Depends on: dlfnc.h
- User compiles their function (or a wrapper to their function) as a dynamic linked/shared library (.so / .dll) with -shared and -fpic
- Path to the object can be specified in the configuration file
- C-based: no overloading, treats references as pointers

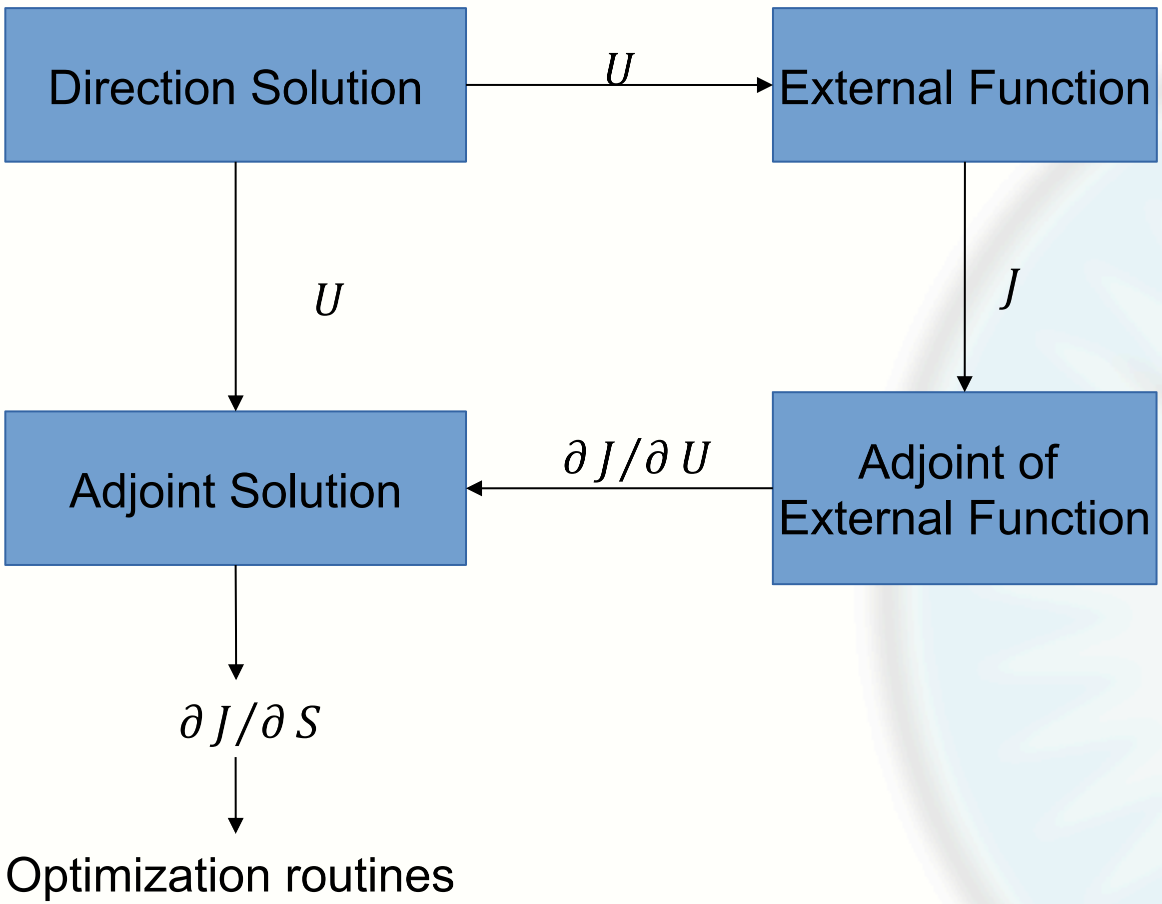
In SU2 (or other driver program)

```
#include dlfnc.h
…
void *handle = dlopen("path_to_compiled_object", RTLD_LAZY);
int (*foo)(double*, int*)
foo = (int (*)(double*, int*))dlsym(handle,"foo");
…
int c = (*foo)(&a, &b);
```

In external program/wrapper

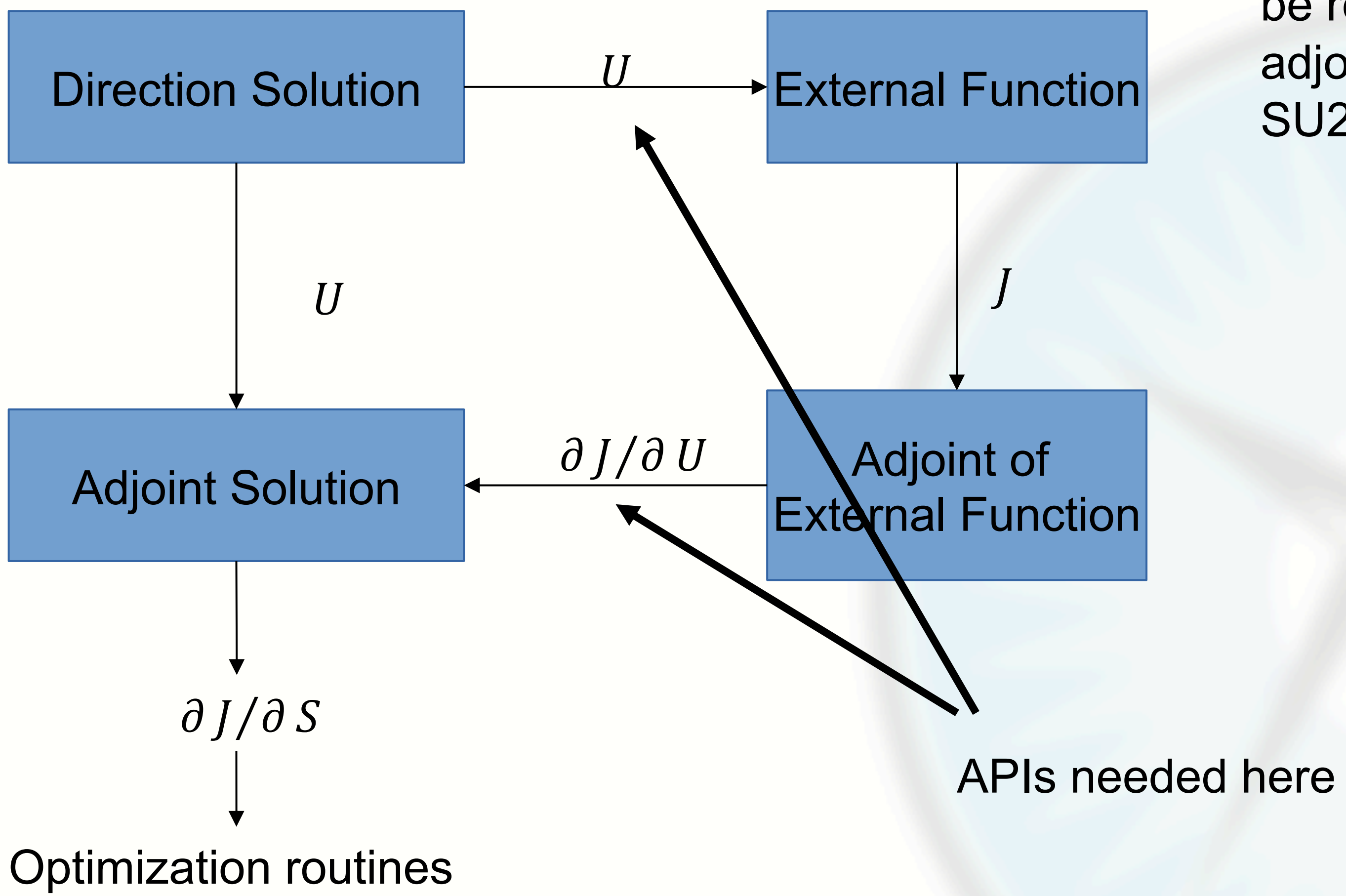
```
extern "C" {
  int foo(const double*, const int*);
}
…
int foo (const double* a, const int* b){
  … code code code
  return c;
}
```

Adjoints and External Tools



The Jacobian of the outputs with respect to the function inputs will be required by the adjoint solution within SU2.

Adjoints and External Tools



The Jacobian of the outputs with respect to the function inputs will be required by the adjoint solution within SU2.

Thank you for your attention